Introduction: Multimedia Databases

Outline of Today's Lecture

- What is multimedia data ?
- What are multimedia data applications ?
- How should multimedia applications access different media types ?
- What kinds of operations do users wish to perform on multimedia data ?
- What infrastructure is needed to support these functionalities:
  - **Content:** Extracting media content. Indexing it. Querying it.
  - **Physical Storage:** Storing media data on primary, secondary, and tertiary storage devices. Building media servers.
  - **Creating Presentations:** How should a media presentation be created (perhaps in response to a query)? How should it be delivered to users at remote sites?
- Overview of the course

Today's Media Types

- Text/Document
- Image
- Video
- Audio
- "Classical" Data (e.g. relations, flat files, object bases, etc.).

Video and audio differ from the other media types listed above because of their temporal nature. In particular:

- Video/Audio retrievals must appear to be continuous, hiccup-free presentations.
- Video/Audio support operations like fast-forward, rewind, and pause, that were not supported by classical data types.

What is an MM_DBMS

A *multimedia database management system* (MM_DBMS) is a framework that manages *different types of data potentially represented in a wide diversity of formats on a wide array of media sources.* An MM_DBMS must:

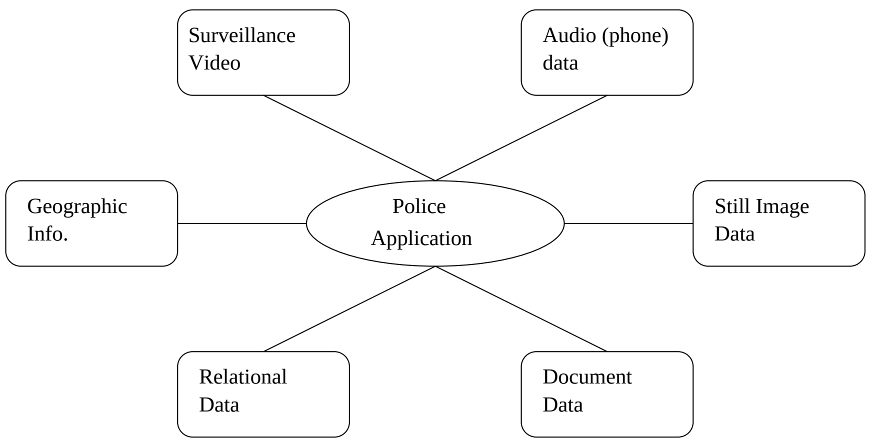
- Have the ability to uniformly **query** data (media data, textual data) represented in different formats.
- Have the ability to **simultaneously query** different media sources and conduct classical database operations across them.
- Have the ability to **retrieve media objects** from a local storage device in a smooth jitter free (i.e. continuous) manner.
- Have the ability to take the answer generated by a query (the notion of "answer to a query" may be mathematical structure of some sort) and develop a **presentation** of that answer in terms of audio-visual media.
- Have the ability to **deliver** this presentation in a way that satisfies various **Quality of Service** requirements.

Sample Multimedia Scenario

Consider a police investigation of a large-scale drug operation. This investigation may generate the following types of data:

- **Video** data captured by surveillance cameras that record the activities taking place at various locations.
- **Audio** data captured by legally authorized telephone wire-taps.
- **Image** data consisting of still photographs taken by investigators.
- **Document** data seized by the police when raiding one or more places.
- **Structured relational data** containing background information, bank records, etc., of the suspects involved.
- **Geographic information systems** data containing geographic data relevant to the drug investigation being conducted.

Data sources used in Sample Multimedia Scenario



Example Image Queries for Multimedia Scenario

Query 1:

- Police officer `John Macho-Dude` has a photograph in front of him.
- He wants to find the identity of the person in the picture.
- Query: "Retrieve all images from the image library in which the person appearing in the (currently displayed) photograph appears."

Query 2:

- Police officer `John Macho-Dude` wants to examine pictures of `Denis Dopeman`.
- Query: "Retrieve all images from the image library in which `Denis Dopeman` appears."

Issues Raised by Example Image Queries

- Two *basic* kinds of queries:
  - Image-based queries
  - Keyword-based queries
- In the first, police officer `John Macho-Dude`, gives a *image* as input (query image). He wants as output, a ranked list of images that are "similar" to the query image.
- To support this, we need to know what "similarity" means.
- We need to know what "ranking" means.
- We need to be able to efficiently support these operations.
- In the second, police officer `John Macho-Dude`, gives a *keyword* as input (name of suspect – `Denis Dopeman`). He wants as output, those photographs that are known to contain an image-object whose name attribute is `Denis Dopeman.`
- To support this, we need to know how to associate different attributes with images (or parts of images).
- We need to know how to effectively index and retrieve images based on such attributed.

Example Audio Query in Sample Multimedia Scenario

Query 1:

- Police officer `John Macho-Dude` is listening to an audio surveillance tape.
- Example: Tape contains a conversation between individual A (person under surveillance) and individual B (somebody meeting person A).
- Query: "Find the identity of individual B, given that individual A is `Denis Dopeman.`

Query 2:

- Police officer `John Macho-Dude` wants to review all audio-logs that `Denis Dopeman` participated in during some specified time period.
- Query: "Find all audio tapes in which `Denis Dopeman` was a participant."

Example Text/Video Query

Text Query:

- Police officer `John Macho-Dude` is browsing an archive of text documents – these include old newspaper archives, police department files on old, unsolved murder cases, witness statements, etc.
- Query: "Find all documents (from the corpora of text documents) that deal with the Cali drug cartel's financial transactions with ABC Corp."

Video Query:

- Police officer `John Macho-Dude` is examining a surveillance video of a particular person being fatally assaulted by an assailant. However, the assailant's face is occluded and image processing algorithms return very poor matches. `John Macho-Dude` thinks the assault was by someone known to the victim.
- Query: "Find all video segments in which the victim of the assault appears."
- By examining the answer to the above query, `John Macho-Dude` hopes to find other people who have previously interacted with the victim.

Simple Heterogeneous Query

- All queries discussed thus far involve one media type.
- For example, each query accesses only image or audio or video data, but does not access a mix of these media types.
- Complex queries will "mix and match" data from these different media sources.
- Such "mixing and matching" is difficult, even for purely textual data sources.
- TEXTUAL EXAMPLE: Find all individuals
  - who have been convicted of attempted murder in North America and
  - who have recently had electronic fund transfers made into their bank accounts from ABC Corp.
- Answering this query is problematic because:
  - Determining all people convicted of different crimes may require accessing a wide variety of databases belonging to different police jurisdictions and courts
  - ABC Corp. may have accounts in hundreds of banks worldwide each of which uses different formats and different database systems

Heterogeneous Multimedia Query

- Find all individuals who have been photographed with Jose Orojuelo and who have been convicted of attempted murder in North America and who have recently had electronic fund transfers made into their bank accounts from ABC Corp.
- This query requires that:
  - We find all people satisfying the conditions in the "Simple" Heterogeneous Query and
  - We access a mugshot-database, **mugshotdb**, containing the names and pictures of various individuals
  - We access a surveillance photograph database of still images
  - We access a **surveillance video** database to see if a meeting between the suspect and Jose Orojuelo was recorded on video
  - We access **image processing algorithms** to determine who occurs in which video/still photograph.

Multimedia Research Issues: Queries

Let us suppose (for now) that we already have a pre-existing body of multimedia data that we wish to access.

- Need a single language within which multimedia data of different types can be accessed.
- In classical database theory, there are binary operations that *combine* different relations in different ways (e.g. join, union, difference, intersection, Cartesian Product). In the same way, this language must be able to specify combination operations across different media types (rather than just across different relations).
- This language must be able to access:
  - "metadata" describing the content of different media sources and
  - "raw" data supported by the different media sources.
- This language must be able to merge, manipulate and "join" together, results from different media sources.
- Once such languages are devised, we need techniques to:
  - optimize a single query (through the notion of a *query plan*
  - develop servers that can optimize processing of a set of queries.

Multimedia Research Issues: Content

- What is content of a media source ? Under what conditions can content be described textually and under what conditions must it be described directly through the original media type?
- How should we extract the content of:
  - an image
  - a video-clip
  - an audio-clip
  - a free/structured text document.
- How should we index the results of this extracted content?
- What is retrieval by similarity?
- What algorithms can be used to efficiently retrieve media data on the basis of similarity ?
- If one has to design a multimedia database using legacy media sources as well as to-be-organized media data, what is the best way of creating such a database?
- How should queries be relaxed so that not only the originally stated query, but also "similar" queries get processed?
- What are efficient algorithms for processing such queries?

Multimedia Research Issues: Storage

- How do the following (standard) storage devices work?
  - disk systems
  - CD-ROM systems
  - tape systems and tape libraries
- How is data laid out on such devices?
- How do we design disk/CD-ROM/tape servers so as to optimally satisfy different clients concurrently, when these clients execute the following operations
  - playback
  - rewind
  - fast forward
  - pause

Multimedia Research Issues: Presentations and Delivery

- How do we specify the content of multimedia presentations?
- How do we specify the form (temporal/spatial layout) of this content?
- How do we create a presentation schedule that satisfies these temporal/spatial presentation requirements?
- How can we deliver a multimedia presentation to users when there is:
  - a need to interact with other remote servers to assemble the presentation (or parts of it)
  - a bound on the buffer, bandwidth, load, and other resources available on the system
  - a mismatch between the host server's capabilities and the customer's machine capabilities?
- How can such presentations optimize Quality of Service?

Overview of the Textbook

- **Part I: Preliminaries**
  - Introduction to Relational DBMSs
  - Introduction to Object Oriented Database Management Systems
- **Part II: Organizing Multimedia Content**
  - Specialized Indexing Structures
  - Image Databases
  - Text/Document Databases
  - Video Databases
  - Audio Databases
  - Multimedia Databases (consisting of a mix of all the above, as well as other unlisted types).
- **Part III: Physical Storage and Retrieval**
  - Retrieving Multimedia Data from Disks
  - Retrieving Multimedia Data from CD-ROMS
  - Retrieving Multimedia Data from Tapes
- **Part IV: Creating and Delivering Multimedia Presentations**
  - Creating Distributed Multimedia Presnetations
  - Distributed Media Servers
  - Future Directions

Plan for this Course (Undergraduate Level)

Week	Chapter/Sections	Remarks
1	Ch. 1-2	Half a lecture on Introduction, rest of Chapter 2.
2	Ch. 3	One lecture on Secs. 3.1-3.3, another on Sec. 3.4-3.5
3	Secs. 4.1-4.2	One lecture on k-d trees, one on point quadtrees.
4	Secs. 4.3-4.5	One lecture on MX-quadtrees, one on R-trees.
5	Secs. 5.1-5.6,5.8	Skip section 5.7
6	Ch. 6	Use lots of examples, drop mathematical details on Sec. 6.4
7	Ch. 7	one lecture on Secs. 7.1, 7.2.1, one lecture on Secs. 7.2.3, 7.3
8	Secs. 9.1-9.4	One lecture on Secs. 9.1-9.2, one on Secs. 9.3-9.4
9	Secs. 9.1-9.4	One lecture on Secs. 9.1-9.2, one on Secs. 9.3-9.4
10	Secs. 9.5-9.6	One lecture on Sec. 9.5, one on Sec. 9.6
11	Sec. 10.1-10.4	Explain Sec. 10.4 by example.
12	Secs. 11.1-11.3 Sec. 12.1-12.2	One lecture on CD-Roms, skim mathematics One lecture on tapes, skim mathematics.
13	Ch. 13	One lecture on Secs. 13.1-13.2, one on Secs. 13.3-13.5
14	Ch. 13	One lecture on Secs. 14.1-14.4, one on Sec. 14.5

Plan for this Course (Graduate Level)

Week	Chapter/Sections	Remarks
1	Ch. 1-3	Brief Introduction. What is a multimedia database ?
2	Sec. 4.1,4.2	1 lecture each on k-d trees and Point Quadtrees
		Explain why these are useful for images.
3	Sec 4.3,4.4	1 lecture each on MX-quadtrees and R-trees.
4	Ch. 5	one lecture on Secs. 5.1-5.4, rest in second lecture.
5	Ch. 6	one lecture of Secs. 6.1-6.5, one lecture on Secs. 6.6-6.7
6	Ch. 7	one lecture on Secs. 7.1, 7.2.1, one lecture on Secs. 7.2.3, 7.3
7	Ch. 8 + midterm exam	One lecture on Ch. 7.
8	Secs. 9.1-9.4	One lecture on Secs. 9.1-9.2, one on Secs. 9.3-9.4
9	Secs. 9.5-9.6, 10.1	One lecture on sec. 9.5-9.6, one on Secs. 10.1
10	Secs.10.2-10.4	One lecture on Sec. 10.2-10.3, one on sec. 10.4
11	Ch. 11	Use plenty of examples contained in Overheads.
12	Ch. 12	Use plenty of examples contained in Overheads.
13	Ch. 13	One lecture on Secs. 13.1-13.2, one on Secs. 13.3-13.5
14	Ch. 14	One lecture on Secs. 14.1-14.4, one on Sec. 14.5

Relational DBMSs

Relational DBMSs

- A relational DBMS manipulates *tables¿*
- A table consists of *rows* and *columns*.
- Each row is called a *tuple*.
- Each column represents an *attribute*.
- Every attribute has an associated set, called the *domain* of the attribute.
- The i'th entry in a given tuple is in the i'th column of the table.
- The i'th entry in a given tuple is drawn from the domain of the attribute associated with the i'th column of the table.

Example

- Consider a relation called `emp`.
- The schema of this relation may be

(`COMP`, `SSN`, `FNAME`, `LNAME`, `STREETNUM`, `STREETNAME`, `CITY`, `STATE`, `ZIP`)

where:

1. `COMP,FNAME,LNAME,STREETNAME,CITY` all denote the set of alphanumeric strings,
2. `STATE` denotes the set of all two-letter strings denoting US states (e.g. MD,VA, etc.) – this represents an enumerated type,
3. `SSN` denotes the set of nine-digit positive integers,
4. `ZIP` denotes the set of five-digit positive integers.
5. `STREETNUM` denotes the set of positive integers.

An instance of the `emp` relation may be:

COMP	SSN	FNAME	LNAME	STREET NUM	STREET NAME	CITY	STATE	ZIP
ABC Corp.	992786589	John	Smith	27	Canal St.	Fairfax	VA	22087.
ABC Corp.	287456725	Denis	Jones	786	Baker St.	Manassas	VA	22185.
ABC Corp.	548923764	Jane	Fox	1224	Cowper Dr.	Bethesda	MD	20984.
ABC Corp.	983744470	Lisa	Barnes	17	Edgar Ct.	Rockville	MD	20887.
ABC Corp.	189465394	Jill	Davis	26	Canal St.	Fairfax	VA	22087.
XYZ Corp.	198473891	Bill	Bosco	11	Lake Dr.	Richmond	VA	23876.
XYZ Corp.	837464632	Bill	Dashell	45	Forest St.	Baltimore	MD	24533.
XYZ Corp.	193746472	David	Johns	581	Lugar Dr.	Rockville	MD	20845.
XYZ Corp.	193284646	Jim	Hatch	2374	Whitman Dr.	Fairfax	VA	22087.
XYZ Corp.	193746466	Tina	Budge	198	Wallis St.	Bethesda	MD	20984.

Relational Algebra

The relational algebra consists of a set of operations that can be used to manipulate tables. Each operation in the algebra takes one or more tables as input, and produces a table as output. The standard operations in RA are listed below.

- Selection
- Projection
- Cartesian Product
- Union
- Difference
- Join
- Intersection

Selection

- Takes as input, a relation R and a condition C.
- $\sigma_C(R)$ says: "Select all tuples in relation R that satisfy condition C."
- The result of executing $\sigma_C(R)$ is a set of tuples drawn from R.
- EX: Suppose we wish to select all tuples in the `emp` relation that have `ABC Corp.` as its `COMP` field. This can be expressed as the selection:

$$\sigma_{\texttt{COMP=ABCCorp.}}(\texttt{emp}).$$

- Result:

COMP	SSN	FNAME	LNAME	STREET NUM	STREET NAME	CITY	STATE	ZIP
ABC Corp.	992786589	John	Smith	27	Canal St.	Fairfax	VA	22087.
ABC Corp.	287456725	Denis	Jones	786	Baker St.	Manassas	VA	22185.
ABC Corp.	548923764	Jane	Fox	1224	Cowper Dr.	Bethesda	MD	20984.
ABC Corp.	983744470	Lisa	Barnes	17	Edgar Ct.	Rockville	MD	20887.
ABC Corp.	189465394	Jill	Davis	26	Canal St.	Fairfax	VA	22087.

Projection

- We specify a relation and one or more columns from that relation that we are interested in.
- The result of a projection is a table that is identical to the original relation, except that all columns not explicitly named are eliminated.
- EX: Suppose we wish to project out the `COMP` and `CITY` fields of the `emp` relation.
- This is specified in the relational algebra as

$$\pi_{\texttt{COMP,CITY}}(\texttt{emp})$$

- Result:

COMP	CITY
ABC Corp.	Fairfax
ABC Corp.	Manassas
ABC Corp.	Bethesda
ABC Corp.	Rockville
XYZ Corp.	Richmond
XYZ Corp.	Baltimore
XYZ Corp.	Rockville
XYZ Corp.	Fairfax
XYZ Corp.	Bethesda

- Note: The tuple `(ABC Corp., Fairfax)` could be placed in the result of the projection in two ways – either via the record associated with John Smith, or via the record associated with Jill Davis. As both these (different) tuples in the original `emp` relation agree on the fields being projected, we place only one copy of the resulting projected tuple, `(ABC Corp., Fairfax)`, instead of both.

Cartesian Product

- Let relation R_1 have the schema $(A_1, \ldots, A_n)$.
- Let R_2 have the schema $(A'_1, \ldots, A'_m)$.
- The Cartesian Product, $R_1 \times R_2$, has the scheme $(A_1, \ldots, A_n, A'_1, \ldots, A'_m)$ and consists of all tuples $(t_1, \ldots, t_n, t'_1, \ldots, t'_m)$ such that:
  - $(t_1, \ldots, t_n)$ is a tuple in R_1 and $(t'_1, \ldots, t'_m)$ is a tuple in R_2.
- EX: Let $R_1 = \pi_{\texttt{COMP,CITY}}(\texttt{emp})$ and let R_2 be

COMPANY	EMPLOYEES
ABC Corp.	30000
XYZ Corp.	15000

- Then $R_1 \times R_2$ is:

COMP	CITY	COMPANY	EMPLOYEES
ABC Corp.	Fairfax	ABC Corp.	30000
ABC Corp.	Manassas	ABC Corp.	30000
ABC Corp.	Bethesda	ABC Corp.	30000
ABC Corp.	Rockville	ABC Corp.	30000
ABC Corp.	Fairfax	XYZ Corp.	15000
ABC Corp.	Manassas	XYZ Corp.	15000
ABC Corp.	Bethesda	XYZ Corp.	15000
ABC Corp.	Rockville	XYZ Corp.	15000
XYZ Corp.	Richmond	ABC Corp.	30000
XYZ Corp.	Baltimore	ABC Corp.	30000
XYZ Corp.	Rockville	ABC Corp.	30000
XYZ Corp.	Fairfax	ABC Corp.	30000
XYZ Corp.	Bethesda	ABC Corp.	30000
XYZ Corp.	Richmond	XYZ Corp.	15000
XYZ Corp.	Baltimore	XYZ Corp.	15000
XYZ Corp.	Rockville	XYZ Corp.	15000
XYZ Corp.	Fairfax	XYZ Corp.	15000
XYZ Corp.	Bethesda	XYZ Corp.	15000

Union

- Relations R_1 and R_2 are *union-compatible* if they have the same scheme.
- Union of relations R_1 and R_2 in this case, denoted $R_1 \cup R_2$, has the same scheme as R_1 (and R_2).
- $R_1 \cup R_2$ contains a tuple t iff t is either in relation R_1 or in relation R_2.
- Let R_1 be the relation $R_1 = \pi_{\texttt{FNAME,LNAME}}(\texttt{emp})$ and suppose R_2 is the relation given by:

FNAME	LNAME
Jack	Arnold
John	Smith
Ted	Garroway

- Then $R_1 \cup R_2$ is:

FNAME	LNAME
Jack	Arnold
John	Smith
Ted	Garroway
Denis	Jones
Jane	Fox
Lisa	Barnes
Jill	Davis
Bill	Bosco
Bill	Dashell
David	Johns
Jim	Hatch
Tina	Budge

Difference

- R_1 and R_2 are *difference-compatible* if they have the same scheme.
- Difference of R_1 and R_2 is denoted $R_1 - R_2$.
- $R_1 - R_2$ contains a tuple t iff t is in R_1, but is not in R_2.
- With the previous example, we see that $R_1 - R_2$ is

FNAME	LNAME
Denis	Jones
Jane	Fox
Lisa	Barnes
Jill	Davis
Bill	Bosco
Bill	Dashell
David	Johns
Jim	Hatch
Tina	Budge

Join

- The `join` operation takes as input, two tables and a boolean condition, C.
- Typically, the boolean condition C involves a condition linking a given attribute in the first table with an attribute in the second.
- EX: Suppose we have a relation called `crime` of the form shown below:

SSN	FIRST	LAST	CONVICTION	DAY	MONTH	YEAR
992786589	John	Smith	drug	17	may	1990
992786589	John	Smith	assault	5	aug	1986
983744470	Lisa	Barnes	tax fraud	14	june	1987
837464632	Bill	Dashell	drug	27	sep	1990
837464632	Bill	Dashell	theft	11	jul	1986
193284646	Jim	Hatch	mail fraud	21	aug	1984
193284646	Jim	Hatch	drug	14	feb	1989

  Suppose we wish to create a table that has the scheme `(LNAME,FNAME, CITY, CONVICTION)` – find all people (and the cities they live in) in the `emp` relation who have been convicted.

- This requires that we:

  1. match social security numbers from the two relations `emp` and `crime` to "match" a person in the `emp` relation with a person in the `crime` relation, and
  2. then he needs to project out the relevant fields (i.e. `LNAME,FNAME` and `CONVICTION`).

- First operation can be done by '

$$\texttt{emp} \bowtie_{\texttt{emp.SSN}=\texttt{crime.SSN}} \texttt{crime}.$$

- Both operations can be done by

$$\pi_{\texttt{FNAME,LNAME,CITY,CONVICTION}}(\texttt{emp} \bowtie_{\texttt{emp.SSN}=\texttt{crime.SSN}} \texttt{crime}).$$

- Result:

FNAME	LNAME	CITY	CONVICTION
John	Smith	Fairfax	drug
John	Smith	Fairfax	assault
Lisa	Barnes	Rockville	tax fraud
Bill	Dashell	Baltimore	drug
Bill	Dashell	Baltimore	theft
Jim	Hatch	Fairfax	mail fraud
Jim	Hatch	Fairfax	drug

Join (Contd.)

- Suppose R_1 and R_2 are two relations whose attribute names are distinct.
- Then $R_1 \bowtie_C R_2$ is given by:

$$R_1 \bowtie_C R_2 \stackrel{def}{=} (\sigma_C(R_1 \times R_2)).$$

Intersection

- Intersection of two relations R_1, R_2 is defined if R_1, R_2 are union compatible.
- Denoted $R_1 \cap R_2$.
- Definition:

$$R_1 \cap R_2 \stackrel{def}{=} R_1 - (R_1 - R_2).$$

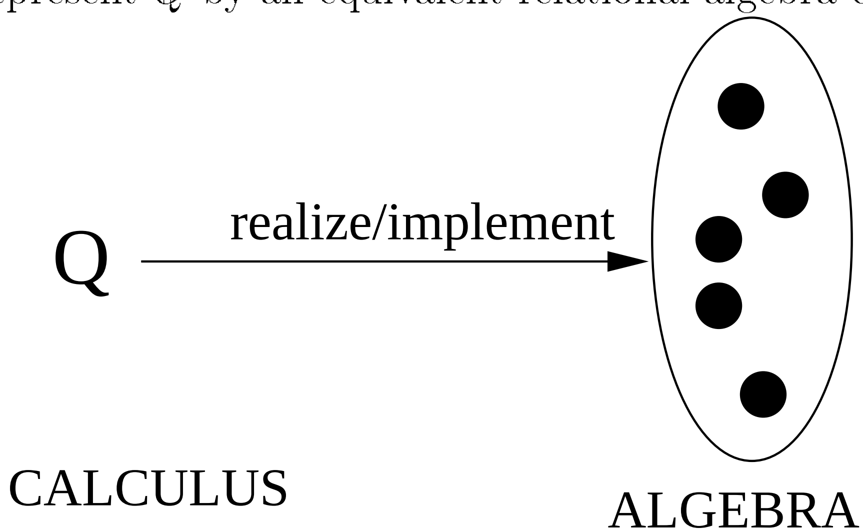
Relational Calculus

- When a user specifies a query in the relational algebra, he specifies a *sequence* of operations to be performed.
- Example: Consider the relational algebra query:

$$\pi_{\texttt{FNAME,LNAME,CITY,CONVICTION}}(\texttt{emp} \bowtie_{\texttt{emp.SSN=crime.SSN}} \texttt{crime}).$$

This query explicitly says: "First perform the specified join and then project out the specified columns."

- Relation Calculus is declarative.
- A Relational Calculus Query specifies *what* to compute, but not *how* to compute it.
- Given a relational calculus query Q, there may be many ways to represent Q by an equivalent relational algebra query.



Relational Calculus Syntax

- $R_1, \ldots, R_k$ – relations.
- Each relation R_i has a schema Sch_i.
- **Constant Symbols:** For each attribute symbol $A \in \cup_{i=1}^{k} Sch_i$, and each member $o \in dom(A)$, o is a constant symbol.
- **Variable Symbols:** There exists an infinite set of variable symbols $t_1, \ldots, t_r, \ldots$ ranging over tuples.
- **Predicate Symbols:** $\in, =, <, >, \leq, \geq, \neq$ are all predicate symbols.
- **Relational calculus atom**:
  - If t is a tuple variable, and R_i is a relation, then $(t \in R_i)$ is an atom. The (sole) occurrence of t in this atom is said to be *free*. t is said to range over R_i.
  - If t is a tuple variable ranging over relation R_i and A is an attribute in Sch_j, then $t.A$ is said to be a *tuple-term*. If $t.A$ is a tuple term, and τ is either a tuple term or a constant symbol, and Θ is any of the comparison operations $=, <, >, \leq, \geq, \neq$, then $(t.A\,\Theta\,\tau)$ and $(\tau\,\Theta\,t.A)$ are atoms. t is said to be free in this atom.
- Example: $(t \in \texttt{bank})\,\&\,(t.LNAME = Smith)\,\&\,(t.AMOUNT > 5,200)$.

- Example: $(t_1 \in \texttt{bank}) \,\&\, (t_2 \in \texttt{emp}) \,\&\, (t_1.FNAME = t_2.FNAME) \,\&\, (t_1.LNAME = t_2.LNAME) \,\&\, (t_1.TRANS = deposit)$.

Relational Calculus Queries

- **Relational Calculus Query:**

  $$\{t \mid F\}$$

  where F is a relational calculus formula in which t appears free.

- Says: Find all tuples t such that condition F is true.

- EX: Find all tuples in the `bank` relation where the amount field of the tuple exceeds \$ 5,200.
  $\{t \mid (t \in \texttt{bank}) \,\&\, (t.AMOUNT > 5,200)\}$

- Find all tuples t_1 such that there exists a tuple $t_2 \in \texttt{emp}$ such that these two tuples have the same FNAME and LNAME fields and such that tuple t_1 reflects a deposit of over \$ 5,200.
  $\{t_1 \mid (\exists t_2)((t_1 \in \texttt{bank}) \,\&\, (t_2 \in \texttt{emp}) \,\&\, (t_1.FNAME = t_2.FNAME) \,\&$
  $(t_1.LNAME = t_2.LNAME) \,\&\, (t_1.TRANS = \mathit{deposit})$
  $\&\, (t_2.AMOUNT > 5,200))\}$.

Linking the Relational Calculus and Relational Algebra

- Relational calculus expresses declarative queries;
- Relational algebra is used to implement these queries.
- Question: Can all relational calculus queries be implemented via equivalent relational algebra queries?
- Answer: No. Only queries that satisfy a property called *safety* can be implemented.

Safety

Suppose F is a relational calculus formula. The *space* of F is defined as follows.

- The space of a relation calculus atom $t \in R$ is the set of all tuples in R.
- The space of a relation calculus atom $(t.A_i \Theta \tau)$ is the set of all tuples over $dom(A_1) \times dom(A_n)$ that satisfy this condition. Here, $(A_1, \ldots, A_n)$ is the scheme of the relation R over which tuple t ranges.
- The space of the relational calculus formula $(F_1 \,\&\, F_2)$ is the intersection of the spaces of F_1 and F_2.
- The space of the relational calculus formula $(F_1 \lor F_2)$ is the union of the spaces of F_1 and F_2.
- The space of the relational calculus formula $\neg F$ involving free variables $t_1, \ldots, t_n$ over relations $R_1, \ldots, R_n$ is the complement of the space of F w.r.t. the appropriate attribute domains of $R_1, \ldots, R_n$.
- The space of the relational calculus formulas $(\forall t \in R)F$ and $(\exists t \in R)F$ coincide with the space of F.

A relational calculus query $\{t \mid F\}$ is said to be *safe* if the space of F is finite.

THEOREM: Consider the relational algebra and the relational calculus.

1. If $\mathcal{A}$ is an expression in the relational algebra, then there exists a (safe) relational calculus query Q such that the set of answers to A coincides with the set of answers to Q.
2. If Q is a query in the safe relational calculus, then there exists an algebraic expression A such that the set of answers to A coincides with the set of answers to Q.

SQL

The most basic SQL query is of the form:

SELECT $attr_1, attr_2, \ldots, attr_n$
FROM $R_1 < V_1 >,\ R_2 < V_2 >, \ldots,\ R_k < V_k >$
<WHERE $F >$.

Here,

- $R_1, \ldots, R_k$ are relation names,
- $V_1, \ldots, V_k$ are tuple variables that range over the relations $R_1, \ldots, R_k$, respectively.
- The V_i's are optional (denoted by their being enclosed within pointed brackets).
- The WHERE clause is optional.
- The above SQL query corresponds to the relational calculus query:
$$\{t \mid (V_1 \in R_1) \,\&\ldots\&\, (V_n \in R_n) \,\&\, F\}$$
where t has the scheme $(attr_1, \ldots, attr_n)$ where each $attr_i$ is associated with one of the relations $R_1, \ldots, R_k$.
- The algebraic version of this query may be expressed as:
$$\pi_{attr_1,\ldots,attr_n}(\sigma_G(R_1 \times \cdots \times R_n))$$

where the relations R_i are (for the sake of simplicity) assumed to have attributes with mutually distinct names and G is obtained from F by replacing all expressions of the form $V_i.attr$ in F are replaced by $attr$.

SQL Example

– Algebra: $\pi_{\texttt{COMP,CITY}}(\texttt{emp})$

– SQL:

```
SELECT      COMP,CITY
FROM        emp
```

– Algebra:

$$\pi_{\texttt{TRANS,DAY,MTH,YR}}(\sigma_{\texttt{FNAME=John}\ \&\ \texttt{LNAME=Smith}\ \&\ \texttt{AMOUNT>6000}}(\texttt{bank}))$$

– SQL:

```
SELECT      TRANS,DAY,MTH,YR
FROM        emp
WHERE       FNAME = John & LNAME = Smith &
            AMOUNT > 6000.
```

– Algebra:

$$\texttt{emp} \bowtie_{\texttt{emp.SSN=crime.SSN}} \texttt{crime}$$

– SQL:

```
SELECT      FNAME, LNAME
FROM        emp E, crime C
WHERE       E.SSN = C.SSN.
```

Object Oriented Databases

Object Oriented Databases

Relational model of data has lots of drawbacks. Some of these are:

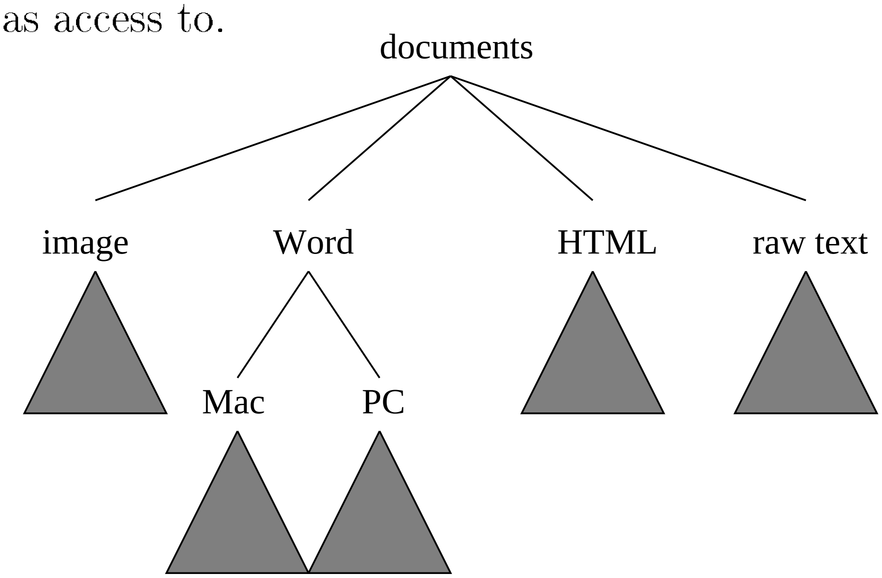
- Data is organized in the form of relatively "flat" tuples, and there is little scope for tuples with fields that reflect complex data structures.
- The schemes of relations are relatively static, and seamlessly extending them to handle temporary variations in data formats (e.g. through the addition of one or more columns, or the deletion of one or more columns) is not well supported.
- The fact that certain relationships might exist between the content of (part of) one table and (part of) another relational table, must be explicitly encoded through the use of constructs such as integrity constraints.

The basic idea behind the OO paradigms is that data-items being manipulated by an application are "organized" as follows:

- *objects* being manipulated by the application;
- *classes* that are collections of objects, either implicitly or explicitly specified
- a *hierarchy* that imposes an acyclic graph structure on the set of classes.

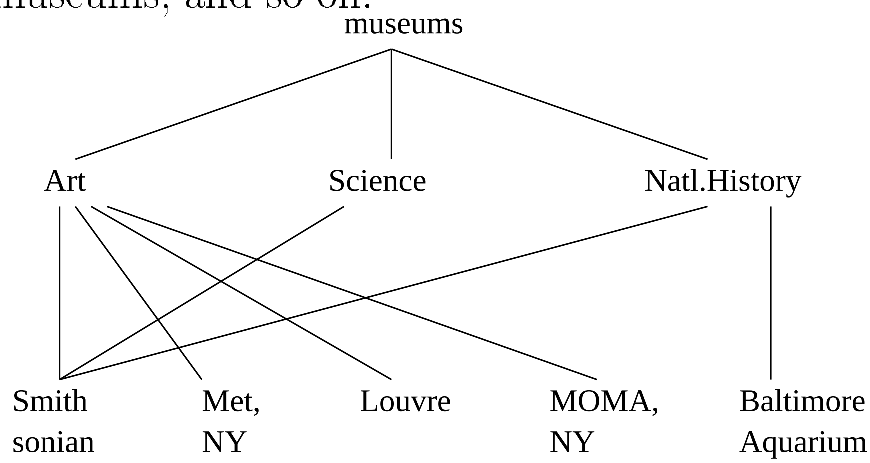
Example Hierarchy

Suppose $d_1, \ldots, d_k$ is the set of *all* documents that the police officer has access to.



Another Example Hierarchy

Suppose we consider an application involving museums. Each museum in this case, may be thought of as an object. Museums may then be grouped into art museums, science museums, natural history museums, and so on.



Object Alphabet

An *object alphabet* consists of:

- A set, called Oid_Set, whose elements are called object-ids;
- A set, called Cid_Set, whose elements are called class-ids;
- A set, Attr_Set, of elements called attributes. Associated with each attribute $a \in$ Attr_Set is a set, $dom(a)$, called the *domain* of the attribute.

Example:

- The set of oids may consist of all strings of the form $\{\flat i \mid i$ is a positive integer $\}$.
- The set of class-id's may consist of all strings of the form $\{\sharp i | i$ is an alphanumeric string $\}$.
- Attributes:
  - `author`, whose domain is all alphabetic strings;
  - `date-created`, `date-last-modified`: whose domain consists of all valid (day,mth,year) triples;
  - `admission-fee`: whose domain consists of all non-negative reals having at most 2 decimal places.

Values

- Suppose $\Sigma = (\mathsf{Oid\_Set}, \mathsf{Cid\_Set}, \mathsf{Attr\_Set})$ is an object alphabet
- Suppose $\mathsf{Attr\_Core} \subseteq \mathsf{Attr\_Set}$ is some designated set of "core" attributes.

The set of values generated by Σ and $\mathsf{Attr\_Core}$, denoted $\mathsf{Values}(\Sigma, \mathsf{Attr\_Core})$ is defined inductively as follows:

- Every member of $\mathsf{Oid\_Set} \cup \cup_{A \in \mathcal{A}} dom(A)$ is a value.
- nil is a special value.
- If $A_1, \ldots, A_n \in \mathsf{Attr\_Ncore}$, and $c_i \in dom(A_i)$ for all $1 \leq i \leq n$, then $[a_1 = c_1; \ldots; a_n = c_n]$ is a value.
- If $v_1, \ldots, v_m$ are values, then $< v_1, \ldots, v_m >$ is a value called a *tuple* value.
- If $v_1, \ldots, v_m$ are values, then $\{v_1, \ldots, v_m\}$ is a value called a *set* value.

Example (of Values)

Let us return to our Police example. Here, our object alphabet Σ may be given by:

- $\mathsf{Oid\_Set} = \{\flat 1, \flat 2, \flat 3, \flat 4, \flat 5, \flat 6\}$.
- $\mathsf{Cid\_Set} = \{\sharp html, \sharp image, \sharp rawtext, \sharp word\_mac, \sharp word\_pc\}$.
- $\mathsf{Attr\_Set}$ may be given by:
  - $\mathsf{Attr\_Core}$: $\{\mathbf{real}, \mathbf{bool}, \mathbf{int}, \mathbf{string}\}$. The domain of these attributes is given in the obvious way.
  - $\mathsf{Attr\_Ncore}$: $\{\mathbf{author}, \mathbf{date-created}, \mathbf{date-last-modified}$ $\mathbf{related-docs}\}$. The attribute, **related-docs**, may have as its domain, the set $2^{\mathsf{Oid\_Set}}$,i.e. the power set of the space of object-ids.

The space of values, $\mathsf{Values}(\Sigma)$ associated with this example, includes:

- $[\mathbf{author} = \text{John Smith}]$;
- $[\mathbf{author} = \text{John Smith}; \mathbf{date-created} = (15,\text{Jan}, 1996)]$;
- $\{[\mathbf{author} = \text{John Smith}], \mathbf{author} = \text{John Smith}; \mathbf{date-created} = (15,\text{Jan}, 1996)]\}$.
- $< [\mathbf{author} = \text{John Smith}], \{[\mathbf{author} = \text{Lisa Adams}], \mathbf{author} = \text{John Smith}; \mathbf{date-created} = (15,\text{Jan}, 1996)]\} >$.

- $[\textbf{author} = \text{Lisa Adams}, \textbf{related} - \textbf{doc} = \{\flat 2, \flat 3\}.]$

Object

Definition: An object o w.r.t. an object alphabet Σ and a core set Attr_Core of attributes, is a pair (id_o, val_o) where $id_o \in$ Oid_Set is called the *identity* of the object, and val_o is called the (property) value of o.

This definition is nice because:

- It says that an object has an associated identity (which tells us how to refer to the object) and
- An object has an associated set of properties (with corresponding values).

EXAMPLE:

- Take an whose id is ♭59878 (corresponding to a person like John Smith)
- Take another object whose id is ♭818932 corresponding to Jill Smith
- They may have different properties – e.g. Jill Smith may have a property called `gynecologist`, with value Donna Manson, but John Smith doesn't have such a property.

Declaring Objects

declare *oid*
values *value.*

EXAMPLE:

declare $\flat 2$
values [**author** = John Smith,
url = http://www.somewhere.com/index.html
date – created = (15,Jan, 1996)
date – last – modified = (19,Nov,1996)].

Says that a document, with object id $\flat 2$ exists, and that this document has four attributes, viz. **author**, **url**, **date-created**, and **date-last-modified**, with the values specified above.

EXAMPLE:

declare $\flat 3$
values [**author** = John Smith,
url = http://www.somewhereelse.com/index.html ,
{[**link** = $\flat^1$], [**link** = $\flat^2$]}].

This HTML document that the police investigator in our sample multimedia scenario might be investigating, is different from the preceding one in a few significant ways. First, it contains links to two objects, viz. $\flat 2$ and $\flat 1$, but it also contains a set of link properties. Furthermore, the attributes, **date-created** and **date-last-modified**, that were associated with the (seemingly similar) document $\flat 2$ are not associated with this document.

Types and Classes

- Each type Attr_Core is a type. (Assume Attr_Core contains the "standard" data types).
- Each member of Cid_Set is a type.
- **Record types** that have *fields*, each with an associated type. $[f_1 : \tau_1; \ldots; f_n : \tau_n]$ denotes a record type having fields $f_1, \ldots, f_n$ of type $\tau_1, \ldots, \tau_n$, respectively.
- **Set types** that are of the form $\{\tau\}$ which denotes a set of data items, each of type τ.
- **List types** that are of the form $< \tau >$ which denotes a list of data items, each of type τ.

EXAMPLE:

[**author**: string;
url: urltype;
date-created: datetype;
date-last-modified: datetype]

EXAMPLE:

[**address**: string;
director: string;
special-exhibits: { string };
affiliated-museums: { Oid_Set }]

Class Hierarchy

DEF: A class hierarchy is a triple, $(G, \leq, \textsf{types})$ where:

- G is a set of objects and classes;
- $\leq$ is a partial ordering on G;
- $\textsf{types}$ is a map that associates a type with each $g \in G$ such that the following condition holds:

$$(\forall g_1, g_2 \in G) g_1 \leq g_2 \rightarrow \textsf{types}(g_1)\ \underline{\textsf{subtype}}\ \textsf{types}(g_2).$$

DEF: The $\textsf{subtype}$ relation is defined as follows:

- $[f_1 : \tau_1, \ldots, f_{n+k} : \tau_{n+k}]$ is a subtype of $[f_1 : \tau_1, \ldots, f_n : \tau_n]$.
- If τ_1 is a subtype of τ_2, then $\{\tau_1\}$ is a subtype of $\{\tau_2\}$.
- If τ_1 is a subtype of τ_2, then $< \tau_1 >$ is a subtype of $< \tau_2 >$.

The above relation is closed under reflexivity and transitivity.

Example Types

Type of `museum` might be:

[**address**: string
director: string
departments: { string }
budget: real.]

Type of `art` (museum) might be:

[**address**: string
director: string
departments: { string }
budget: real
old-master-collection: { string };
modern-art-collection: { string };
lithograph-collection: { string };
cartographic-collection: { string }]

Note that the type of `art` is a subtype of the type associated with `museum`.

Methods

DEF: *Methods* are programs associated with any object/class $g \in G$ that manipulate the structures declared in g's type definition.

- Each class g has an associated set of programs, methods(g), that apply to that class.
- Provides *encapsulation.*
- Objects in a class can be manipulated only by methods applicable to that class, no others.
- Methods provide an elegant interface via which other classes/objects and/or other third party programs, may access an object or a class.

EXAMPLE:

- The class `documents` may have a method called $\texttt{FindDocs}_d$ which takes as input, any name (type `string`) and returns as output, a set of documents.
- The class `image` may have a method called `FindDocs` (referred to by us as $\texttt{FindDocs}_i$) of which takes as input, any name (type `string`) and returns as output, a set of images.

- The class `Word` may have a method called `FindDocs` (referred to by us as $\texttt{FindDocs}_w$) which takes as input, any name (type `string`) and returns as output, a set of Word documents.

Suppose the function `FindDocs` is invoked by the class `image`. Then the code associated with the function $\texttt{FindDocs}_i$ is executed, as there is a version of `FindDocs` that is *directly associated* with this class.

On the other hand, consider invoking the method `FindDocs` in the class `PC_Word`. This class has no method called `FindDocs` *explicitly* associated with it. Thus, we must examine if one of its ancestors has `FindDocs` associated with it. The answer is yes, both the class `Word`, and the class `Documents`, have `FindDocs` associated with it. However, the class `Word` is *more specific* and applies "better" (informally speaking) to the case of PC-Word documents, so we apply the function $\texttt{FindDocs}_w$ to objects in the class `PC_Word`.

Object Definition Language (ODL)

- Provide a simple language within which both objects and object interfaces can be defined.
- External programs wishing to access/manipulate the object must do so using the methods provided by the object.
- To do so, however, they must have access to the signatures (I/O types) of those objects.
- ODL provides a formal syntax for this purpose

ODL Example

interface html:documents (1)
(**extent** html_documents (2)
keys url:**persistent** (3)
{<properties>
<operations>}
);

- Line (1) above states that the type, **`html`** is a subtype of the bigger type, **`documents`**.
- Line (2) above states that the type **`html`** applies to all HTML documents, i.e. it specifies that this type definition applies to all HTML documents in the object oriented database.
- Line (3) specifies that the type HTML has a key attribute, called **`url`**, which is persistent. This means that memory allocated to such persistent objects does not cause the object to "disappear" when the process that creates it terminates.
- In addition, the ODL definition of the object **`html`** contains two additional features, viz. a list of *properties* and a list of *relationships*.
- These may be "filled in" as shown on the next slide.

ODL Example

interface html:documents (1)
(**extent** html_documents (2)
keys url:**persistent** (3)
{ **attribute string** author; (4)
attribute date date_created; (5)
attribute date date_last_modified; (6)
relationship Set<Persons> author **inverse** Persons:written_work; (7)
relationship Set<Persons> reader **inverse** Persons:works_read; (8)
}

Object Query Language (OQL)

- Introduced by the ODMG group.
- If Q is a query expressed in SQL, then Q is also a valid OQL query.
- SQL queries can only access "flat" relational tables.
- In contrast, objects may have a nested structure, as well as include fields that contain the so-called *collection types* – sets, lists, and bags. OQL provides facilities to access such data types as well.

EXAMPLE

```
SELECT   struct(field1:x.url, field2:x.link)
FROM     Word x
WHERE    x.author="John Smith".
```

- This query first finds all objects x that are `Word` documents, and then identifies those `Word` documents that satisfy the conditions x.author="John Smith".
- For each such x, it returns a *structure* containing two fields: the `url` field of x, and the `link` field of x. Note that the link field of x may be a linked list. Thus, an answer to this query may look like:

www.somewhere.com/index.html	www.somewhere.com/file1.html www.somewhere.com/file2.html www.abc.com/file1.html.
www.cs.umd.edu/users/john/index.html	www.somewhere.com/index.html www.somewhere.com/file1.html
...	...

More OQL Examples

```
SELECT    y.author
FROM
          (SELECT  x.link
           FROM    Word x
           WHERE   x.author="John Smith") y.
```

- Query is complicated for two reasons: first, it contains a nested query, and second, it contains this new variable y.
- This query says: *first find all Word documents authored by John Smith and let y refer to any such object* – this is accomplished by the inner SELECT statement.
- Then return the $y.author$ field for all such documents.

Object-Relational Systems

- Relational databases have proved very useful in querying "flat" data.
- Is it possible to extend the relational model of data to handle complex data, rather than flat data?
- The answer is yes, and the resulting paradigm is typically called an object-relational database.

Example Object-Relational Systems

- Consider a bank relation with schema

  (FNAME, LNAME, ACCTYPE, TRANS, AMOUNT, DAY, MTH, YR)

  and a crime relation with schema

  (SSN, FIRST, LAST, CONVICTION, DAY, MTH, YR).

- Suppose we wished to extend both these schemes to also include an image.
- Adequate to merely extend the schemes of the two relations involved to include a PIC field as follows.

  (FNAME, LNAME, ACCTYPE, TRANS, AMOUNT, DAY, MTH, YR, PIC).

  (SSN, FIRST, LAST, CONVICTION, DAY, MTH, YR, PIC).

- Suppose that we have a tuple in the bank relation of the form:

FNAME	LNAME	ACCTYPE	TRANS	AMOUNT	DAY	MTH	YR	PIC
Jill	Davis	savings	withdrawal	1400	1	jan	1993	image1

- Suppose furthermore, that Jill Davis goes to the bank and reports that her bankcard was stolen from her, and that she did not make this withdrawal.
- In this case, an obvious check to run is to examine the surveillance image, stored in the file image1, to see if it is in fact Jill Davis or not. Suppose Ms. Davis' assertion is correct and it is apparent that the person depicted in image1 is not her.

- next logical step for the police to perform is to attempt to match the contents of `image1` against the images contents of the `crime` database.

- This requires execution of a query that says: *Select all tuples in the* `crime` *relation that "match" (using face recognition techniques) the image depicted in* `image1`*.*

- Relational database query languages such as SQL cannot support this query directly because the selection condition requires a comparison operator (`match`) that is not typically supported by a relational database.

- An object database query language cannot be used either because the data is not stored in an object oriented system.

Object-Relational Schemes

- Suppose we have a set $\mathcal{O}$ of objects, each with associated properties and methods.
- An *object relational scheme* is of the form

$$(\mathtt{A_1} : \mathtt{T_1}, \ldots, \mathtt{A_n} : \mathtt{T_n})$$

where the A_i's are attribute names, and the T_i's are objects.

- The scheme of the extended `crime` relation can be written as:

$$(\mathtt{FNAME} : \mathtt{str}, \mathtt{LNAME} : \mathtt{str}, \mathtt{ACCTYPE} : \mathtt{int}, \mathtt{TRANS} : \mathtt{tt},$$

$$\mathtt{AMOUNT} : \mathtt{real}, \mathtt{DAY} : \mathtt{dom}, \mathtt{MTH} : \mathtt{mths}, \mathtt{YR} : \mathtt{int}, \mathtt{PIC} : \mathtt{image}).$$

$$(\mathtt{SSN} : \mathtt{ssntyp}, \mathtt{FIRST} : \mathtt{str}, \mathtt{LAST} : \mathtt{str}, \mathtt{CONVICTION} : \mathtt{str},$$

$$\mathtt{DAY} : \mathtt{dom}, \mathtt{MTH} : \mathtt{mths}, \mathtt{YR} : \mathtt{int}, \mathtt{PIC} : \mathtt{image}).$$

- Examples of object conditions:
  - $match(image1, image2) > 0.7$;
  - $match(image1, image2) > match(image3, image2)$;
  - $match(image1, image2) < t.SAL$. (Though this condition may not make much sense from an intuitive point of view, it is a syntactically valid condition if $t.SAL$ is a real valued field).
- We may now extend SQL to access the functions associated with objects and express our desired query as follows:

SELECT FNAME, LNAME
FROM crime C, bank B
WHERE match(image1,B.PIC) ¿ 0.9.

- This query says "Select the FNAME and LNAME fields of all tuples t in the `crime` relation such that the match function defined in the object `image` returns a value of over 90% when it matches image1 and the `PIC` field of t."
- In general, if $m(arg1, \ldots, argn)$ is a valid method invocation, and x is either another method invocation or a value of the same type as the output type of method m, then $m(arg1, \ldots, argn) = x$ is an *object* condition. If x is a numeric value and the above condition holds, then $m(arg1, \ldots, argn) > x$, $m(arg1, \ldots, argn) \geq x$, $m(arg1, \ldots, argn) \leq x$, are also *object conditions.*

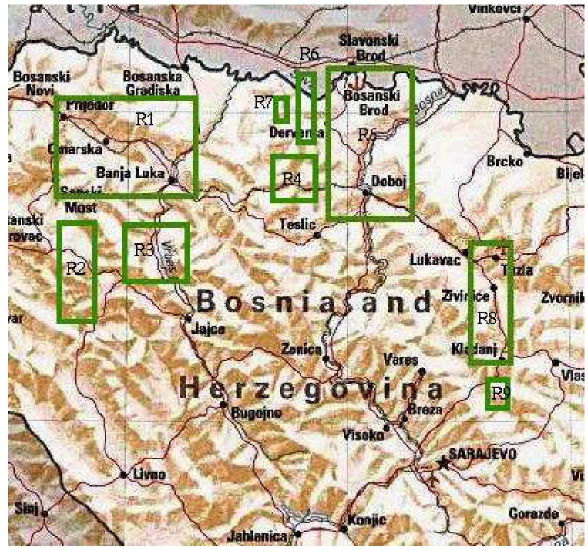
Multidimensional Data Structures

Multidimensional Data Structures

- An important source of media data is geographic data.
- A geographic information system (GIS) stores information about some physical region of the world.
- A map is just viewed as a 2-dimensional image, and certain "points" on the map are considered to be of interest.
- These points are then stored in one of many specialized data structures.
  - k-d Trees
  - Point Quadtrees
  - MX-Quadtrees
- Alternatively, we may wish to store certain rectangular regions of the map.
- We will study one data structure – the R-tree – that is used to store such rectangular data.

Example Maps

(a) Map with Marked Points



(b) Map with Marked Regions

k-D Trees

- Used to store k dimensional point data.
- It is *not* used to store *region* data.
- A 2-d tree (i.e. for $k = 2$) stores 2-dimensional point data while a 3-d tree stores 3-dimensional point data, and so on.

Node Structure

nodetype = **record**
  INFO: **infotype**;
  XVAL: **real**;
  YVAL: **real**;
  LLINK: ↑**nodetype**
  RLINK: ↑**nodetype**
end

<table>
<tr><td>INFO</td><td>XVAL</td><td>YVAL</td></tr>
<tr><td>LLINK</td><td colspan="2">RLINK</td></tr>
</table>

- INFO field is any user-defined type whatsoever.
- XVAL and YVAL denote the coordinates of a point associated with the node.
- LLINK and RLINK fields point to two children.

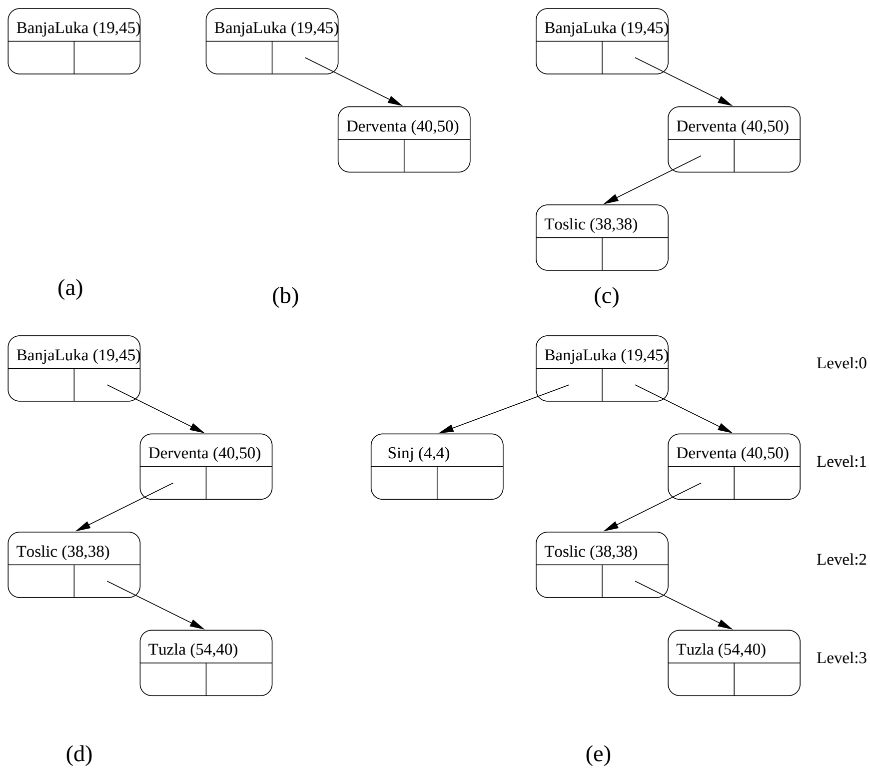
2-d trees, formally

Level of nodes is defined in the usual way (with root at level 0).

Def: A 2-d tree is any binary tree satisfying the following condition:

1. If N is a node in the tree such that $level(N)$ is even, then every node M in the subtree rooted at $N.LLINK$ has the property that $M.XVAL < N.XVAL$ *and* every node P in the subtree rooted at $N.RLINK$ has the property that $P.XVAL \geq N.XVAL$.
2. If N is a node in the tree such that $level(N)$ is odd, then every node M in the subtree rooted at $N.LLINK$ has the property that $M.YVAL < N.YVAL$ *and* every node P in the subtree rooted at $N.RLINK$ has the property that $P.YVAL \geq N.YVAL$.

Example 2-d Trees

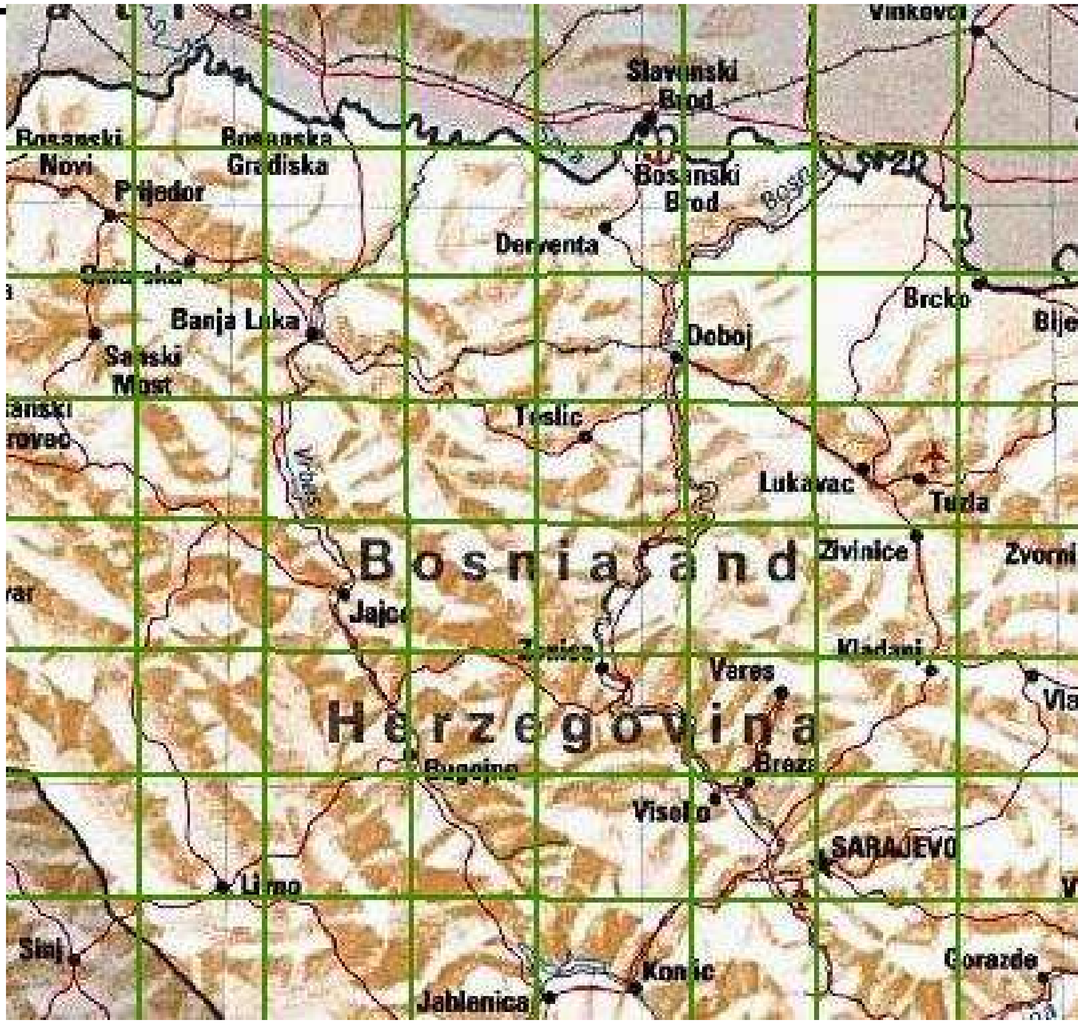


Insertion/Search in 2-d Trees

To insert a node N into the tree pointed to by T, do as follows:

- Check to see if N and T agree on their XVAL and YVAL fields.
- If so, just overwrite node T and we are done.
- Else, branch left if $N.XVAL < T.XVAL$ and branch right otherwise.
- Suppose P denotes the child we are examining. If N and P agree on their XVAL and YVAL fields. just overwrite node P and we are done, else branch left if $N.YVAL < P.YVAL$ and branch right otherwise.
- Repeat this procedure, branching on XVAL's when we are at even levels in the tree, and on YVALs when we are at odd levels in the tree.

Example of Insertion



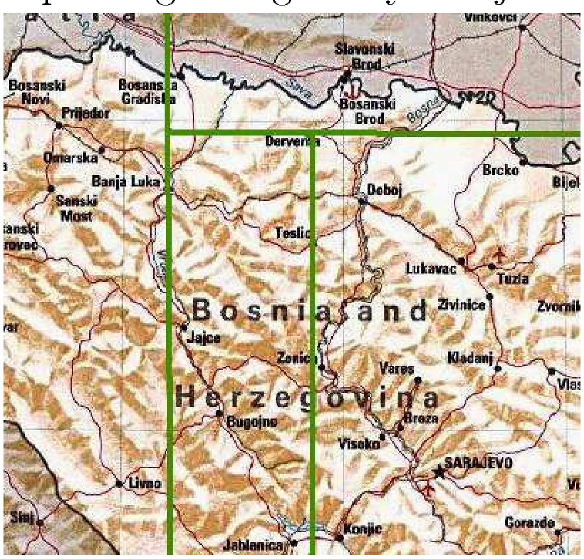
Suppose we wish to insert the following points.

City	(XVAL,YVAL)
Banja Luka	(19,45)
Derventa	(40,50)
Toslic	(38,38)
Tuzla	(54,35)
Sinj	(4,4)

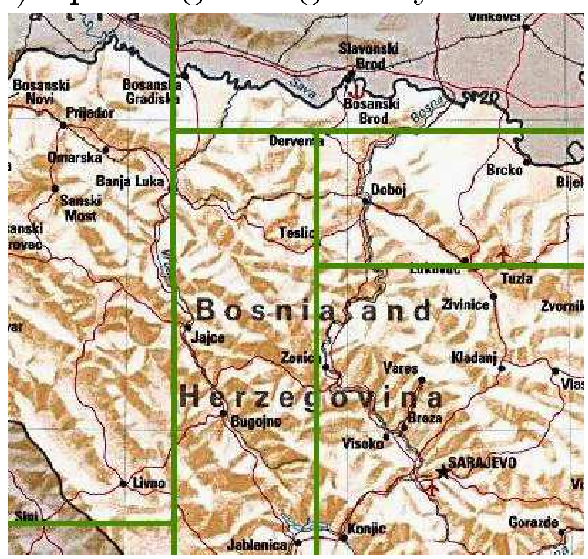
Example of Insertion

(a) Splitting of region by Banja Luka

(b) Splitting of region by Derventa



(c) Splitting of region by Toslic



(d) Splitting of region by Sinj

Deletion in 2-d Trees

Suppose T is a 2-d tree, and (x, y) refers to a point that we wish to delete from the tree.

- Search for the node N in T that has $N.XVAL = x$ and $N.YVAL = y$.
- If N is a leaf node, then set the appropriate field (LLINK or RLINK) of N's parent to NIL and return N to available storage.
- Otherwise, either the subtree rooted at $N.LLINK$ (which we will denote by T_ℓ) or the subtree rooted at $N.RLINK$ (which we will denote by T_r) is non-empty.

  (Step 1) Find a "candidate replacement" node R that occurs either in T_i for $i \in \{\ell, r\}$.

  (Step 2) Replace all of N's non-link fields by those of R.

  (Step 3) Recursively delete R from T_i.
- The above recursion is guaranteed to terminate as T_i for $i \in \{\ell, r\}$ has strictly smaller height than the original tree T.

Finding Candidate Replacement Nodes for Deletion

- The desired replacement node R must bear the same spatial relation to all nodes P in both T_ℓ and T_r that N bore to P
- I.e. if P is to the southwest of N, then P must be to the southwest of R, if P is to the northwest of N, then P must be to the northwest of R, and so on.

This means that the desired replacement node R must satisfy the property that:

1. Every node M in T_ℓ is such that: $M.XVAL < R.XVAL$ if $level(N)$ is even and $M.YVAL < R.YVAL$ if $level(N)$ is odd.
2. Every node M in T_r is such that: $M.XVAL \geq R.XVAL$ if $level(N)$ is even and $M.YVAL \geq R.YVAL$ if $level(N)$ is odd.

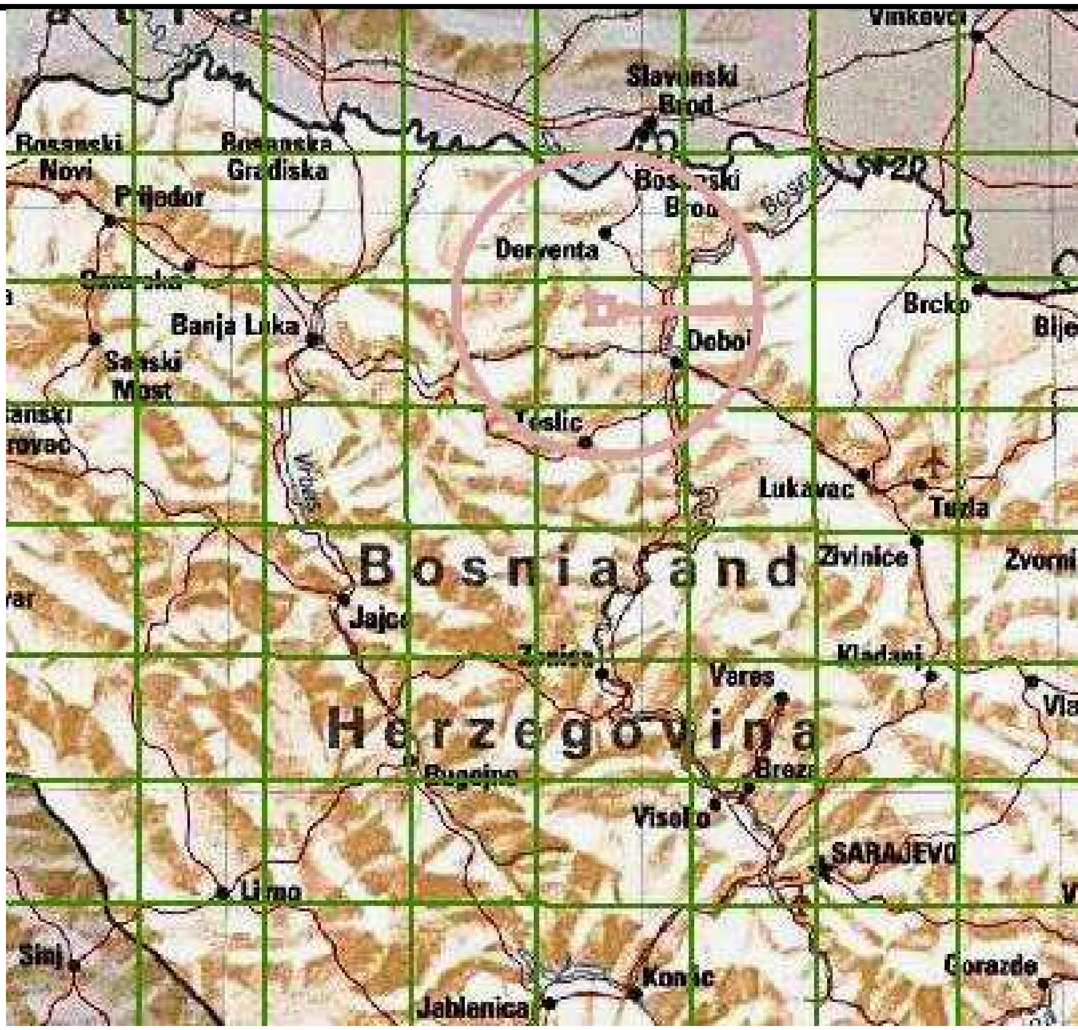
- If T_r is not empty, and $level(N)$ is even, then any node in T_r that has the *smallest* possible $XVAL$ field in T_r is a candidate replacement node.
- But if T_r is empty, then we might not be able to find a candidate replacement node from T_ℓ (why?).

- In this case, find the node R' in T_ℓ with the smallest possible XVAL field. Replace N with this.
- Set $N.RLINK = N.LLINK$ and set $N.LLINK = NIL$.
- Recursively delete R'.

Range Queries in 2-d Trees

- A *range query* with respect to a 2-d tree T is a query that specifies a point (x_c, y_c), and a distance r.
- The *answer* to such a query is the set of all points (x, y) in the tree T such that (x, y) lies within distance d of (x_c, y_c).
- I.e. A range query defines a circle of radius r centered at location (x_c, y_c), and expects to find al points in the 2-d tree that lie within the circle.
- Recall that each node N in a 2-d tree implicitly represents a region R_N.
- If the circle specified in a query has no intersection with R_N, then there is no point searching the subtree rooted at node N.

Example Range Query

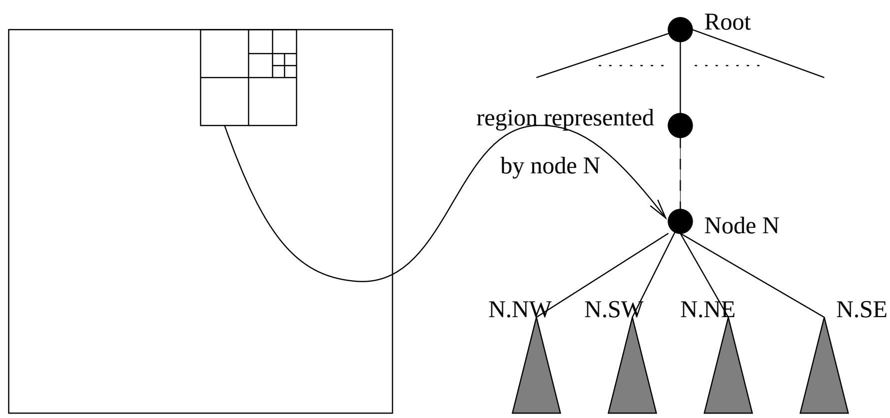


Point Quadtrees

- Point quadtrees always split regions into *four* parts.
- In a 2-d tree, node N splits a region into two by drawing *one* line through the point $(N.XVAL, N.YVAL)$.
- In a point quadtree, node N splits the region it represents by drawing *both* and horizontal *and* a vertical line through the point $(N.XVAL, N.YVAL)$.
- These four parts are called the NW (northwest), SW (southwest), NE (northeast) and SE (southest) quadrants determined by node N.
- Each of these quadrants corresponds to a child of node N. Thus, quadtree nodes may have upto 4 children each.
- Node structure in a point quadtree:

```
qtnodetype = record
    INFO: infotype;
    XVAL: real;
    YVAL: real;
    NW,SW,NE,SE: ↑qtnodetype
end
```

Nodes in Point Quadtrees Implicitly Represent Regions

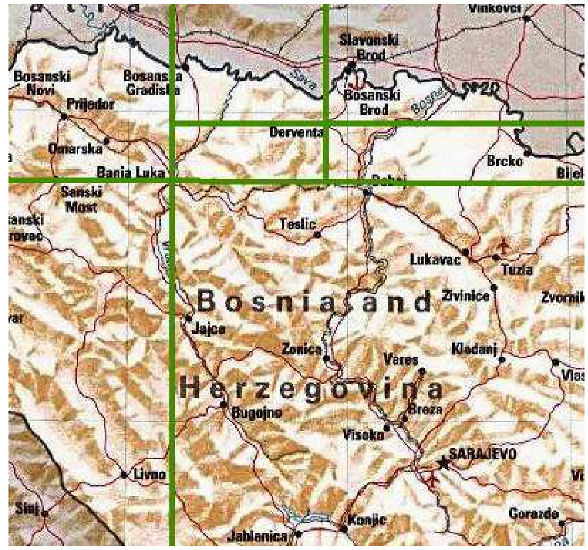


Insertion into Point Quadtrees

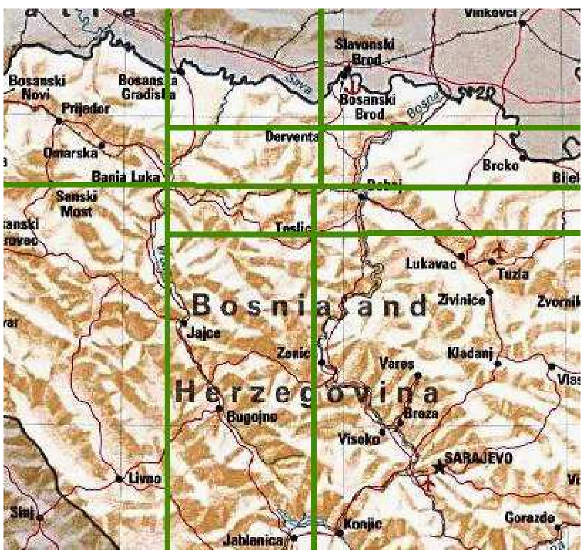
City	(XVAL,YVAL)
Banja Luka	(19,45)
Derventa	(40,50)
Toslic	(38,38)
Tuzla	(54,35)
Sinj	(4,4)

Insertion into Point Quadtrees

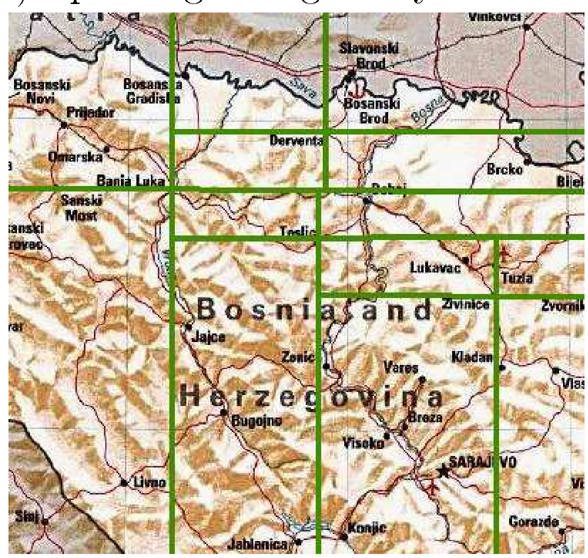
(a) Splitting of region by Banja Luka



(b) Splitting of region by Derventa



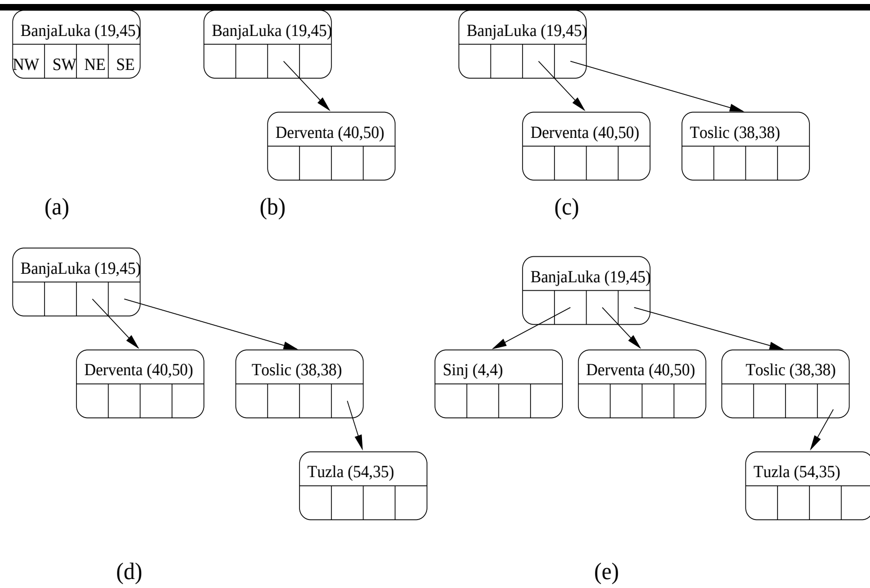
(c) Splitting of region by Toslic



(d) Splitting of region by Tuzla

(e) Splitting of region by Sinj

Insertion into Point Quadtrees



Deletion in Point Quadtrees

- If the node being deleted is a leaf node,'deletion is completely trivial: we just set the appropriate link field of node N's parent to NIL and return the node to available storage.
- As in the case of deletion in 2-d trees, we need to find an appropriate replacement node for non-leaf nodes being deleted.
- Is this easy?
- No. Why? Return to Previous slide.

Expanded Node Type

- Expand the node structure `qtnodetype` to a new node structure `newqtnodetype`

- **qtnodetype = record**
  INFO: **infotype**;
  XVAL,YVAL: **real**;
  XLB,YLB,XUB,YUB: **real** $\cup \{-\infty, +\infty\}$
  NW,SW,NE,SE: $\uparrow$**qtnodetype**
  end

- When inserting a node N into the tree T, we need to ensure that:
  - If N is the root of tree T, then $N.XLB = -\infty$, $N.YLB = -\infty$, $N.XUB = +\infty$, $N.YUB = +\infty$.
  - If P is the parent of N then the following table describes what N's XLB, YLB, XUB, YUB fields should be, depending upon whether N is the NW, SW, NE, SE child of P. We use the notation $w = (P.XUB - P.XLB)$ and $h = (P.YUB - Y.YLB)$.

Case	N.XLB	N.XUB	N.YLB	N.YUB
N=P.NW	P.XLB	P.XLB $+w \times 0.5$	P.YLB $+h \times 0.5$	P.YUB
N=P.SW	P.XLB	P.XLB $+w \times 0.5$	P.YLB	P.YLB $+h \times 0.5$
N=P.NE	P.XLB $+w \times 0.5$	P.XUB	P.YLB $+h \times 0.5$	P.YUB
N=P.SE	P.XLB $+w \times 0.5$	P.XUB	P.YLB	P.YLB $+h \times 0.5$

Deletion in Point Quadtrees, Continued

– When deleting an interior node N, we must find a replacement node R in one of the subtrees of N (i.e. in one of N.NW,N.SW,N.NE,N.SE) such that:

  * every other node R_1 in N.NW is to the north west of R,
  * every other node R_2 in N.SW is to the south west of R,
  * every other node R_3 in N.NE is to the north east of R and
  * every other node R_4 in N.SE is to the south east of R.

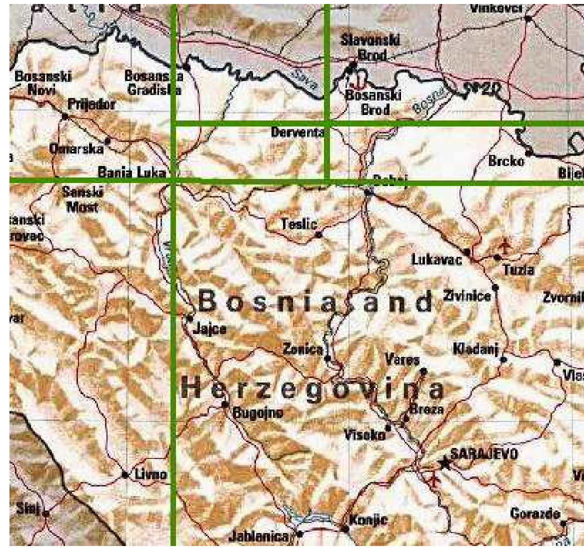
– Consider the figure on the next page.

– Suppose we wish to delete Banja Luka from this quadtree. In this case, one such replacement node can in fact be found, viz. Toslic.

– However, in general, it may not always be possible to find such a replacement node. See the figure in the page after next.

Deletion of Banja Luka

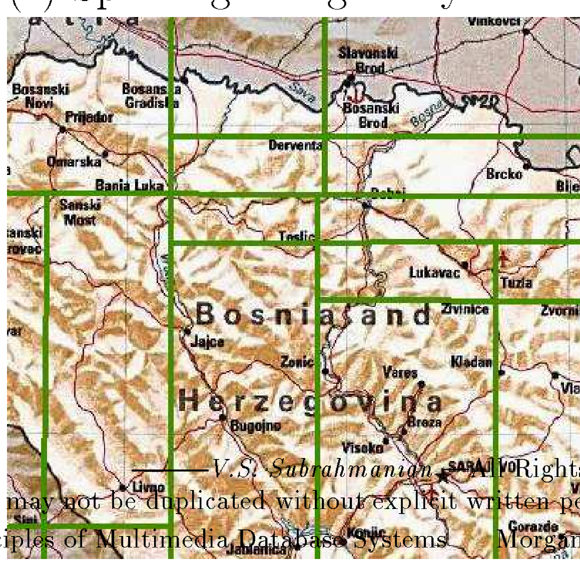
(a) Splitting of region by Banja Luka



(b) Splitting of region by Derven

(c) Splitting of region by Toslic

(d) Splitting of region by Tuzla



Impossibility of finding Replacement Candidates

Thus, in general, deletion of an interior node N may require reinsertion of all nodes in the subtrees pointed to by $N.NE$, $N.SE$, $N.NW$ and $N.SW$. In the worst case, this may require almost all nodes to be reinserted.

Range Searches in Point Quadtrees

– Each node in a point quadtree represents a region.

– Do not search regions that do not intersect the circle defined by the query.

proc RangeQueryPointQuadtree(T:newqtnodetype, C:circle);

1. If $region(T) \cap C = \emptyset$ then **Halt**
2. else
   (a) If $(T.XVAL, T.YVAL) \in C$ then **print** $(T.XVAL, T.YVAL)$;
   (b) **RangeQueryPointQuadtree**(T.NW,C);
   (c) **RangeQueryPointQuadtree**(T.SW,C);
   (d) **RangeQueryPointQuadtree**(T.NE,C);
   (e) **RangeQueryPointQuadtree**(T.SE,C);

end proc

The MX-Quadtree

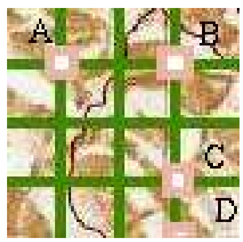
- For both 2-d trees as well as point quadtrees, the "shape" of the tree depends upon the order in which objects are inserted into the tree.
- In addition, both 2-d trees and point quadtrees split regions into 2 (for 2-trees) or 4 (for point quadtrees) subregions – however, the split may be uneven depending upon exactly where the point $(N.XVAL, N.YVAL)$ is located inside the region represented by node N.
- MX-quadtrees attempt to: ensure that the shape (and height) of the tree are *independent* of the number of nodes present in the tree, as well as the order of insertion of these nodes.
- MX-quadtrees also attempt to provide efficient deletion and search algorithms.

The MX-Quadtree

– Assume that the map being represented is "split up" into a grid of size $(2^k \times 2^k)$ for some k.

– The application developer is free to choose k as s/he likes to reflect the desired granularity, but once s/he chooses k, s/he is required to keep it fixed.

Example:



The MX-Quadtree

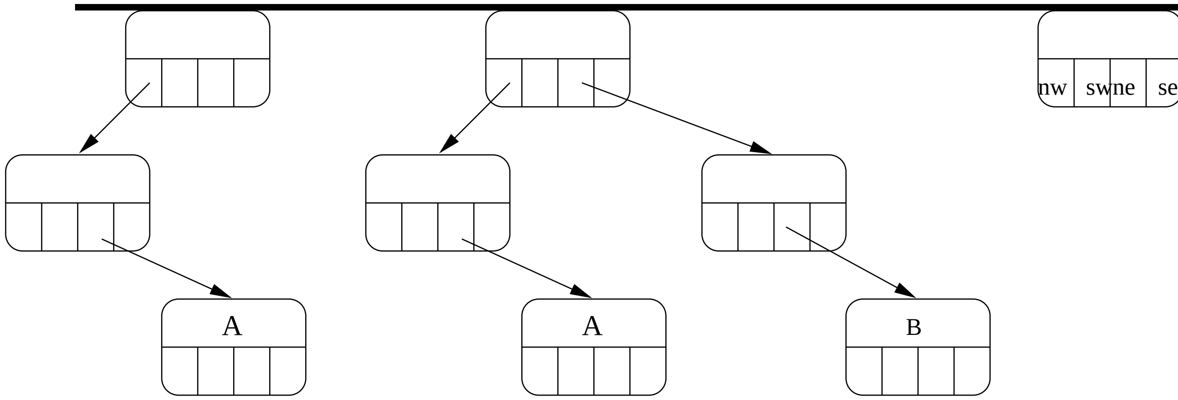
– **Node Structure**: Exactly the same as for point quadtrees, execpt that the root of an MX-quadtree represents the region specified by XLB= 0, XUB= 2^k, YLB= 0, YUB= 2^k.

– When a region gets "split", it gets split down the middle. Thus, if N is a node, then the regions represented by the four children of N are described by the following table.

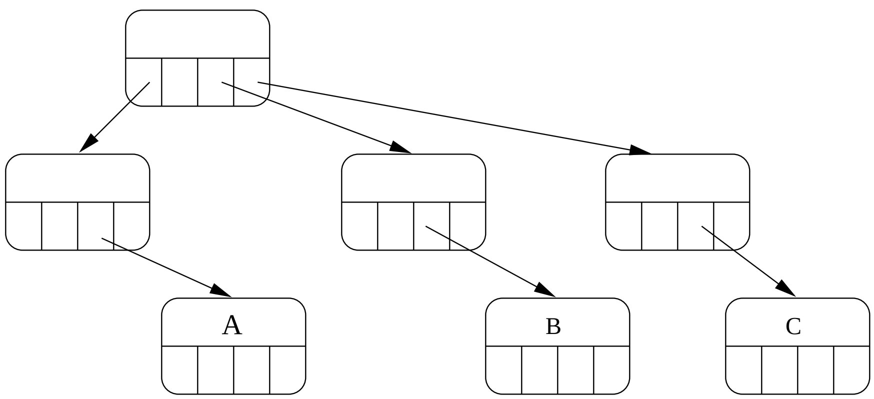
Child	XLB	XUB	YLB	YUB
NW	N.XLB	N.XLB+$\frac{w}{2}$	N.YLB+$\frac{w}{2}$	N.YLB+w
SW	N.XLB	N.XLB+$\frac{w}{2}$	N.YLB	N.YLB+$\frac{w}{2}$
NE	N.XLB+$\frac{w}{2}$	N.XLB+w	N.YLB+$\frac{w}{2}$	N.YLB+w
SE	N.XLB+$\frac{w}{2}$	N.XLB+w	N.YLB	N.YLB+$\frac{w}{2}$

Here, w denotes the width of the region represented by N.

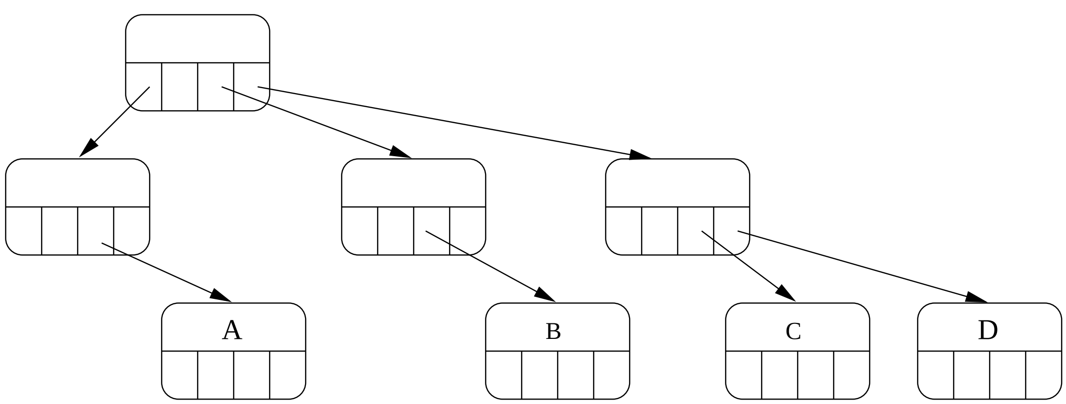
Insertion in MX-Quadtrees



(a) (b)

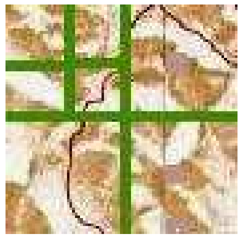


(c)

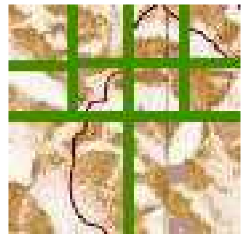


(d)

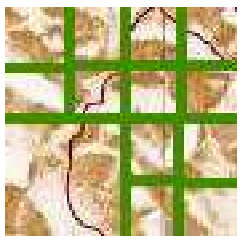
Insertion in MX-Quadtrees



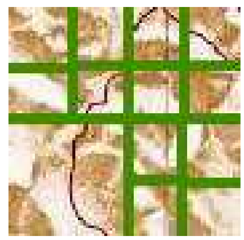
After insertion of A



After Insertion of B



After insertion of C



After Insertion of D

Deletion in MX-Quadtrees

- Deletion in an MX-quadtree is a fairly simple operation, because all points are represented at the leaf level.
- If N is an interior (i.e. non-leaf) node in an MX-quadtree whose root is pointed to by T, then the region implicitly represented by node N contains at least one point that is explicitly contained in the tree.
- If we wish to delete a point (x, y) from tree T, we try to preserve this property.
- This can be done as follows.
  * First, we set the appropriate link of N's parent to NIL.
  * We then check if all the four link fields of M are NIL.
  * . If so, we examine M's parent (let us call it P for now). As M is P's child, we find a link field `dir1` such that P.`dir1` $= M$. We then set P.`dir1` = `NIL` and then (as before) check to see if P's four link fields are all NIL.
  * f so, we continue this process.
- Total time required for deletion is O(k).

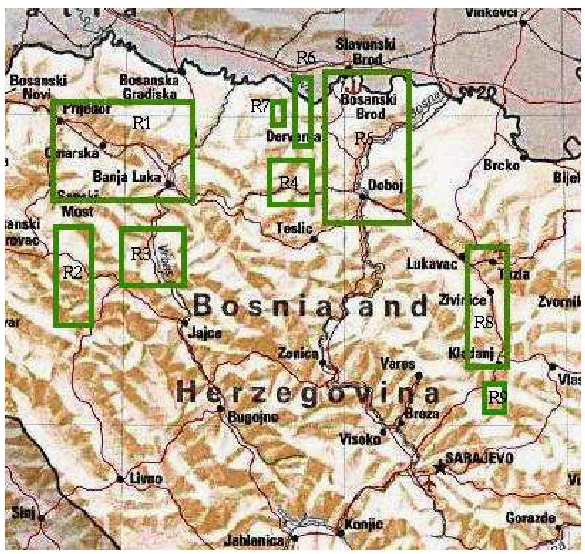
Range Queries in MX-Quadtrees

Handled in exactly the same way as for point quadtrees. But there are two differences:

- The *content* of the XLB,XUB,YLB,YUB fields is different from that in the case of point quadtrees.
- As points are stored at the leaf level, checking to see if a point is in the circle defined by the range query needs to be performed only at the leaf level.

R-Trees

- Used to store *rectangular regions* of an image or a map such as those shown below.
- R-trees are particularly useful in storing very large amounts of data on disk.
- They provide a convenient way of minimizing the number of disk accesses.

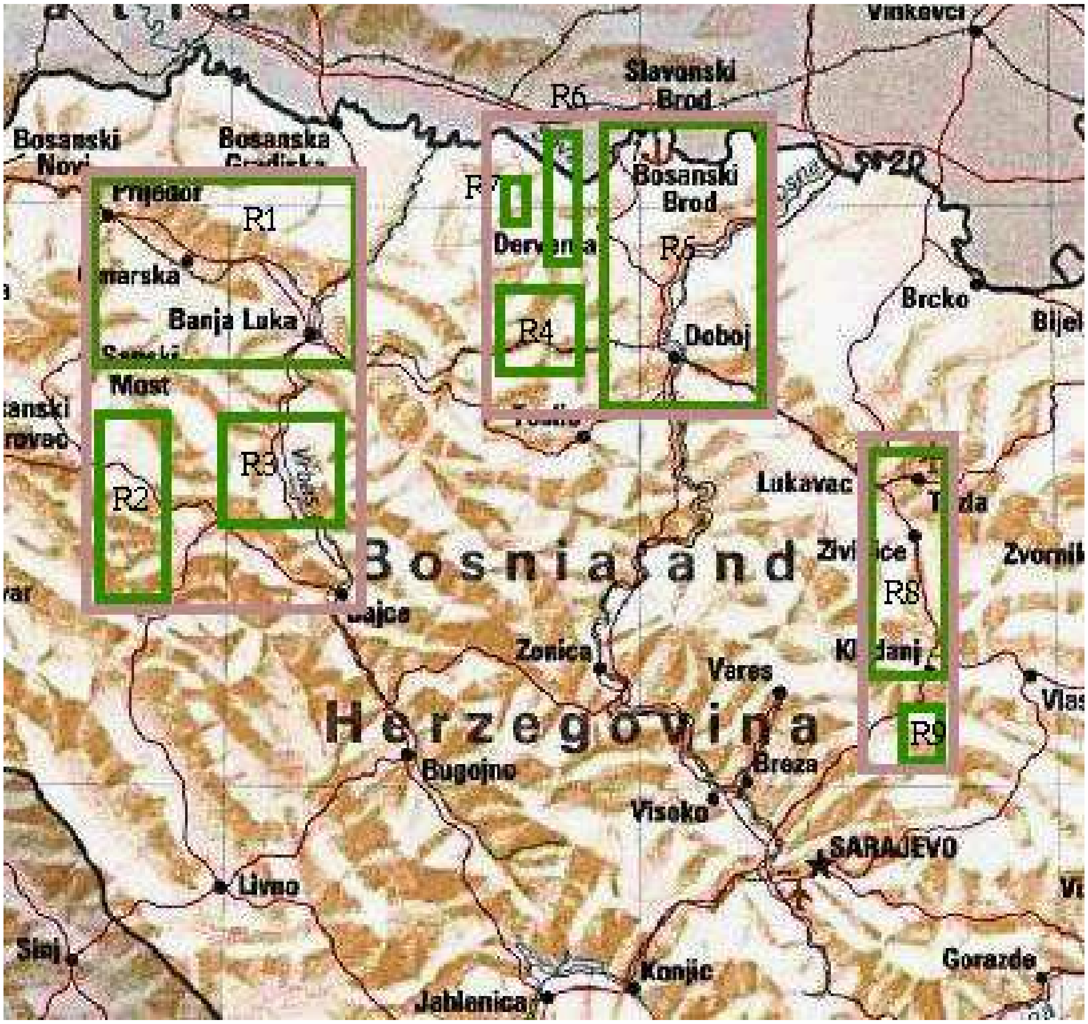


R-Trees

- Each R-tree has an associated *order*, which is an integer K.
- Each nonleaf R-tree node contains a set of at most K rectangles and at least $\lceil K/2 \rceil$ rectangles (with the possible exception of the root).
- Intuitively, this says that each nonleaf node in the R-tree, with the exception of the root, must be at least "half" full.
- This feature makes R-trees appropriate for disk based retrieval because each disk access brings back a page containing several (i.e. at least $\frac{K}{2}$ rectangles).

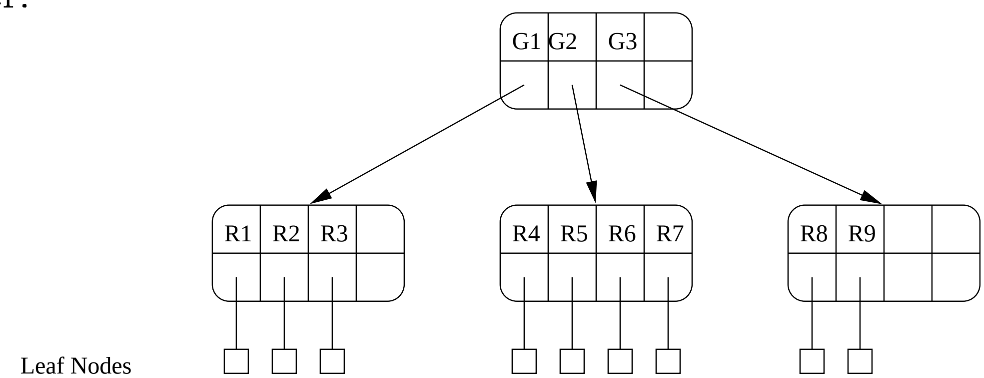
R-trees manipulate two kinds of rectangles:

- "Real" rectangles (such as those shown in the map on the previous slide) or
- "Group" rectangles uch as those shown below.


R1
R2
R3
R4
R5
R6
R7
R8
R9
Bosanski Novi
Bosanska Gradiska
Prijedor
Banja Luka
Slavonski Brod
Bosanski Brod
Doboj
Brcko
Lukavac
Zvornik
Bosnia and
Herzegovina
Zenica
Vares
Bugojno
Breza
Visoko
SARAJEVO
Livno
Sinj
Konjic
Jablenica
Gorazde
Vinkovci
Klkovac
Mostar
Tuzla
Jajce

Example R-Tree

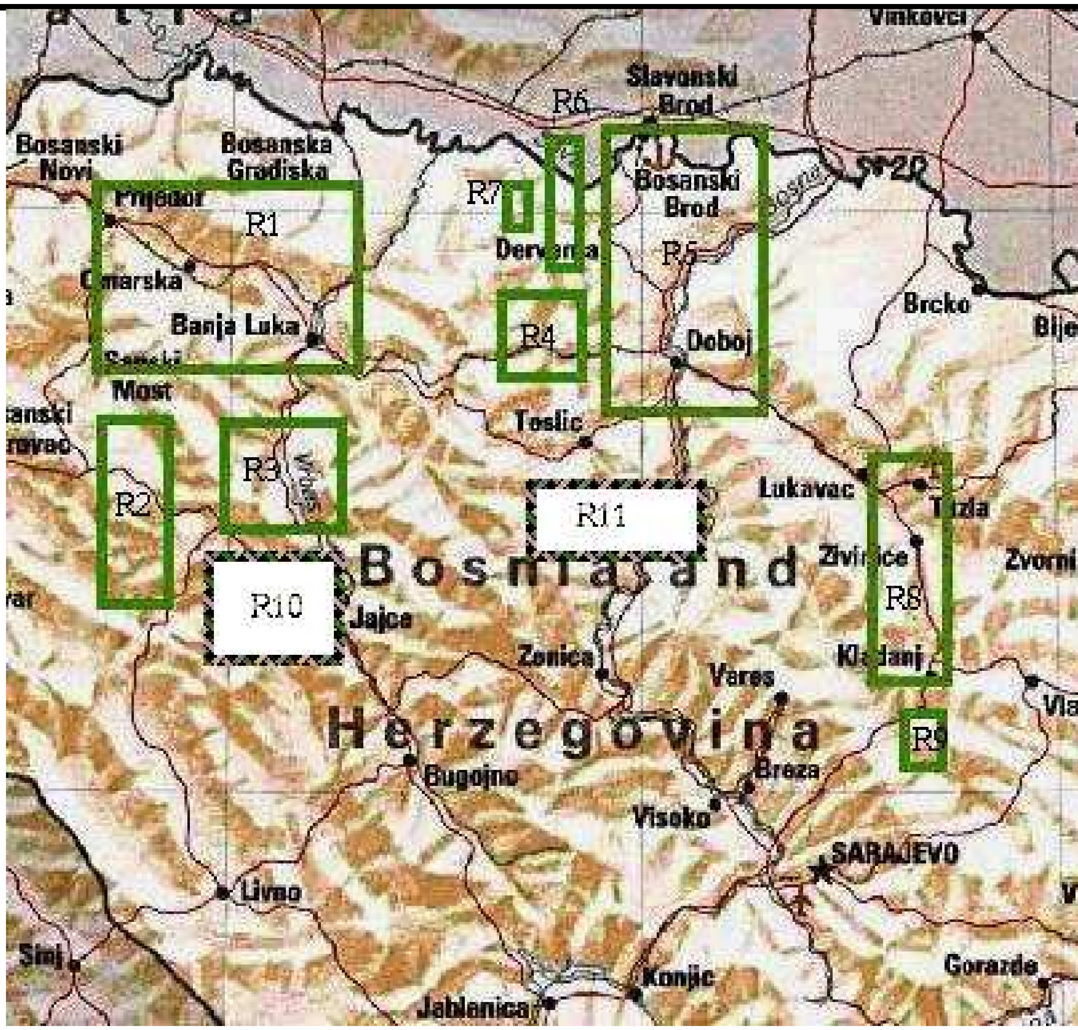
This is an R-tree of order 4, associated with the rectangles shown earlier.



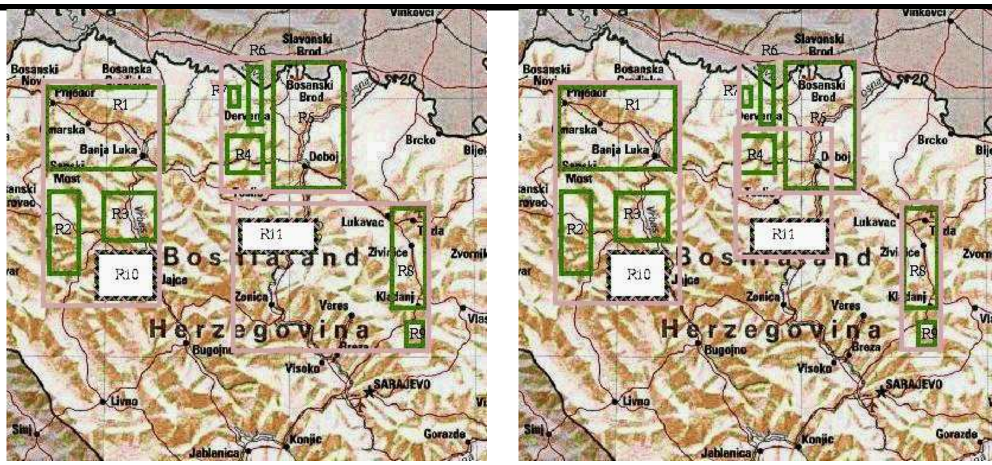
R-tree nodes have the following structure:

rtnodetype = record
  $Rec_1, \ldots, Rec_K$: **rectangle**;
  $P_1, \ldots, P_K$: $\uparrow$**rtnodetype**
end

Insertion into an R-Tree

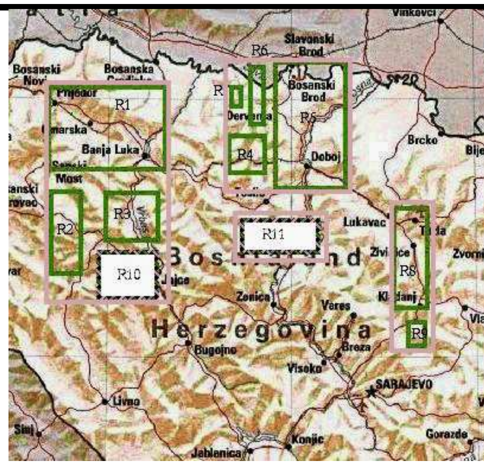


Insertion into an R-Tree



(a) (b)

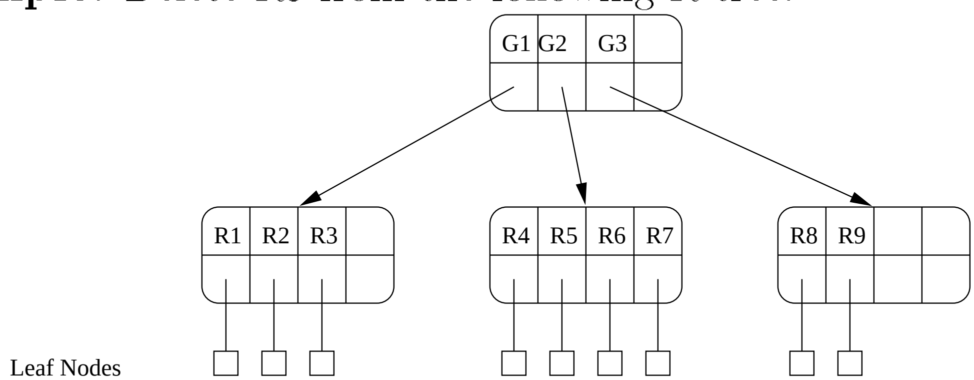
An Incorrect Insertion into an R-Tree



Deletion in R-Trees

- Deletion of objects from R-trees may cause a node in the R-tree to "underflow" because an R-tree of order K must contain at least $\lceil K/2 \rceil$ rectangles (real or group) in it.
- When we delete a rectangle from an R-tree, we must ensure that that node is not "underfull."

Example: Delete R9 from the following R-tree.



Deletion in R-Trees

- If we delete R9, then the node containing rectangle R9 would have only one node in it.
- In this case, we must create a new logical grouping.
- One possibility is to reallocate the groups as follows:

Group	Rectangles
G1	R1,R2,R3
G2	R4,R6,R7
G3	R5,R8

- The new new R-tree is:

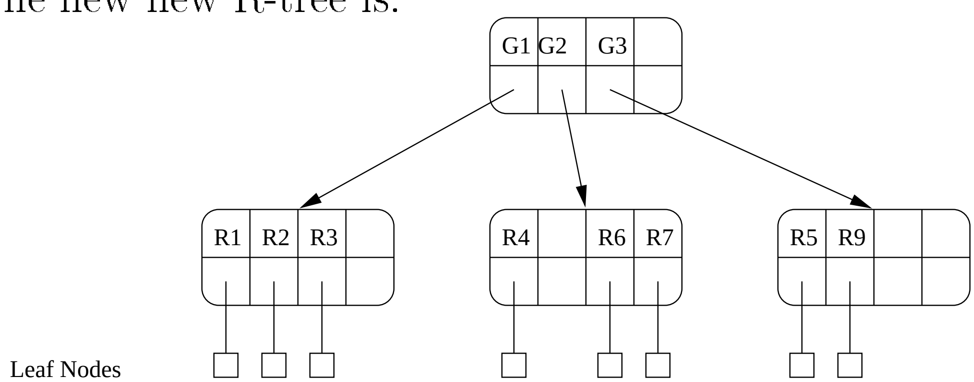


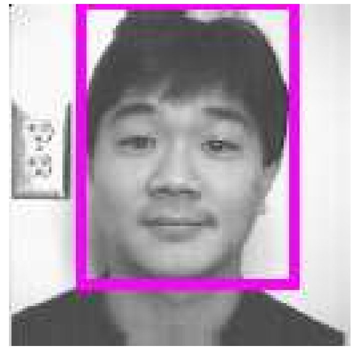
Image Databases

Image Databases

- In standard relational databases, the user types in a query, and obtains an answer in response.
- In image databases, things are different: for example, a police investigator may have in front of him/her, a surveillance photograph of someone, whose identity s/he may not know, but wishes to determine. Thus, s/he may wish to ask a query of the form: *Here's a picture of a person. Can you retrieve all pictures from the image database that are "similar" to this person and tell me the identities of the people in the pictures you return to me ?*
- This query is fundamentally different from ordinary queries for two reasons:
  1. First, the query includes a picture as part of the query.
  2. Second, the query asks about "similar" pictures and hence, uses a notion of "imprecise match" whose definition needs to be precisely articulated (it is possible to reason precisely about imprecise data !)

Example Face Database

pic1.gif



pic2.gif

pic3.gif

pic4.gif

pic5.gif

pic6.gif

pic7.gif

Raw Images

The content of an image consists of all "interesting" objects in that image. Each object is characterized by:

- a *shape descriptor* that describes the shape/location of the region within which the object is located inside a given image.
- a *property descriptor* that describes the properties of the individual pixels (or groups of pixels) in the given image. Examples of such properties include: red-green-blue (RGB) values of the pixel (or aggregated over a group of pixels), grayscale levels in the case of black and white images, etc. In general, it will be infeasible to associate properties with individual pixels, and hence, cells (rectangular "groups" of pixels) will be used most of the time.
- We assume the existence of a set **Prop** of properties. A property consists of two components –
  1. A *property name* – e.g. "Red", "Green", "Blue" and
  2. a *a property domain* which specifies the range of values that the property can assume – e.g $\{0, \ldots, 8\}$.

Example

Example: Consider the image file `pic1.gif` on the preceding slide. This image has two objects of interest - let call these two objects o_1 and o_2.

- The shapes of these objects are captured by rectangles shown. Formally, object o_1's shape may be specified by:

  $rectangle : XLB = 10; XUB = 60, YLB = 5; YUB = 50.$

- The *property descriptor* associated with an individual cell (group of pixels) may look like this:

  1. Red = 5;
  2. Green = 1;
  3. Blue = 3.

  Other properties like texture may also be included.

Definitions

- Every image I has an associated pair of positive integers (m, n), called the *grid-resolution* of the image. This divides the image into $(m \times n)$ *cells* of equal size, called the *image grid*.
- Each cell in a given gridded $(m \times n)$ image I consists of a collection of pixels.
- A *cell property* is a triple $(\mathsf{Name}, \mathsf{Values}, \mathsf{Method})$ where Name is a string denoting the property's name, Values is a set of values that that property may assume, and Method is an algorithm that tells us how to compute the property involved.

Example: Consider black and white images.

$$(\mathsf{bwcolor}, \{\mathsf{b}, \mathsf{w}\}, \mathsf{bwalgo})$$

could be a cell property with property name $\mathsf{bwcolor}$ and the possible values are b (black) and w (white), respectively. bwalgo then is an algorithm which may take a cell as input, and return as output, either black or white, by somehow combining the black/white levels of the pixels in the cell.

Example: Consider gray scale images (with 0 =white and 1 =black), we may have the cell property

$$(\mathsf{graylevel}, [0, 1], \mathsf{grayalgo})$$

where our property name is named graylevel, and its possible values are real numbers in the $[0, 1]$ interval, and the associated method grayalgo takes as input, a cell, and computes its gray level.

Image Definitions

- An *object shape* is any set P of points such that if $p, q \in P$, then there exists a sequence of points $p_1, \ldots, p_n$ all in P such that:

  1. $p = p_1$ and $q = p_n$ and
  2. for all $1 \leq i < n$, p_{i+1} is a neighbor of p_i, i.e. if $p_i = (x_i, y_i)$ and $p_{i+1} = (x_{i+1}, y_{i+1})$, then (x_{i+1}, y_{i+1}) satisfies one of the following conditions:

$$
\begin{array}{cc}
(x_{i+1}, y_{i+1}) = (x_i + 1, y_i) & (x_{i+1}, y_{i+1}) = (x_i - 1, y_i) \\
(x_{i+1}, y_{i+1}) = (x_i, y_i + 1) & (x_{i+1}, y_{i+1}) = (x_i, y_i - 1) \\
(x_{i+1}, y_{i+1}) = (x_i + 1, y_i + 1) & (x_{i+1}, y_{i+1}) = (x_i + 1, y_i - 1) \\
(x_{i+1}, y_{i+1}) = (x_i - 1, y_i + 1) & (x_{i+1}, y_{i+1}) = (x_i - 1, y_i - 1)
\end{array}
$$

- A rectangle is an object shape, P, such that there exist integers XLB, XUB, YLB, YUB such that

$$P = \{(x, y) \mid XLB \leq x < XUB \,\&\, YLB \leq y < YUB\}.$$

Image Database

Def: An *image database*, IDB, consists of a triple (GI, Prop, Rec) where:

1. GI is a set of gridded images of the form $(Image, m, n)$ and
2. Prop is a set of cell properties, and
3. Rec is a mapping that associates with each image, a set of rectangles denoting objects.

Issues in Image Databases

- First and foremost, images are often very large objects consisting of a $(p_1 \times p_2)$ pixel array. Explicitly storing properties on a pixel by pixel basis is usually infeasible. This has led to a family of *image compression* algorithms that attempt to compress the image into one containing fewer pixels.
- Given an image I (compressed or raw), there is a critical need to determine what "features" appear in the image. This is typically done by breaking up the image into a set of homogeneous (w.r.t. some property) rectangular regions, each of which is called a *segment*. The process of finding these segments is called *segmentation*.
- Once image data has been segmented, we need to support "match" operations that map either a whole image or a segmented portion of an image against another whole/segmented image.

Compressed Image Representations

- Consider a 2-dimensional image I consisting of $(p_1 \times p_2)$ pixels.
- Let $I(x, y)$ be a number denoting one or more attributes of the pixel.
- he creation of the compressed representation, $\mathsf{cr}(I)$, of image I consists of two parts:
  1. **Size Selection:** The larger the size, the greater is the fidelity of the representation. However, as the size increases, so does the complexity of creating an index for manipulating such representations, and searching this index. Let $\mathsf{cr}(I)$ be of size $h_1 \times h_2$ where $h_i \leq p_i$.
  2. **Transform Selection:** The user must select a transformation, which, given the image I, and any pairs of number $1 \leq i \leq h_1$, and $1 \leq j \leq h_2$ will determine what the value of $\mathsf{cr}(i, j)$ is.
  3. There are many such transforms.

The Discrete Fourier Transform (DFT)

$$\underline{\mathsf{DFT}}(i,j) = \frac{1}{\sqrt{p_1}} \times \frac{1}{\sqrt{p_2}} \times \sum_{a=0}^{p_1} \sum_{b=0}^{p_2} \left(I(a,b) \times \underline{\exp}\left(\frac{-2\pi \underline{j} a \times i}{p_1}\right) \times \frac{-2\pi \underline{j} b \times i}{p_2}\right).$$

where: j is the well known complex number, $\sqrt{-1}$.

DFT has many nice properties.

- **Invertibility:** It is possible to "get back" the original image I from its DFT representation. Useful for decompression.
- *Note that practical realizations of DFT ave often sacrificed this property by applying the DFT together with certain other non-invertible operations.*
- **Distance Preservation:** DFT preserves Euclidean distance. This is important in image matching applications where we may wish to use distance measures to represent similarity levels.

The Discrete Cosine Transform

$$\underline{\mathsf{DCT}}(i,j) = \frac{2}{\sqrt{p_1 \times p_2}} \alpha(i) \times \alpha(j) \sum_{r=0}^{p_1-1} \sum_{s=0}^{p_2-1} \left(\cos\left(\frac{(2r+1) \times \pi \dot{i}}{2r} \right) \times \cos\left(\frac{(2s+1) \times \pi \dot{j}}{2s} \right) \right)$$

where:

$$\alpha(i), \alpha(j) = \begin{cases} \frac{1}{\sqrt{2}} & \text{when } u, v = 0 \\ 1 & \text{otherwise.} \end{cases}$$

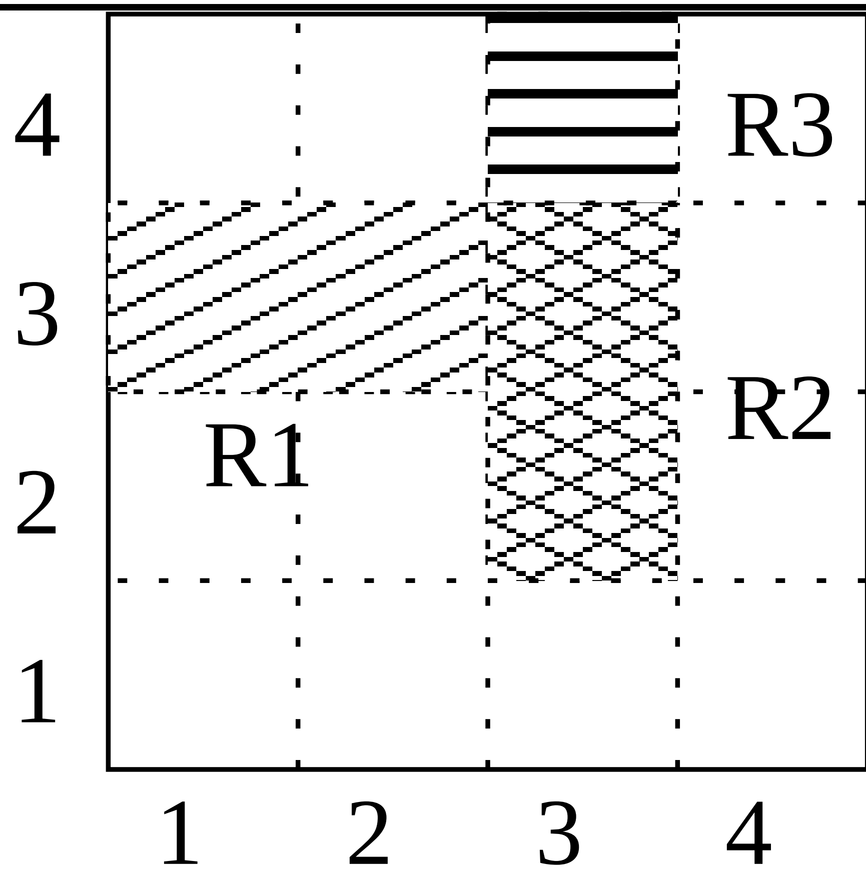
- DCT also is easily invertible
- DCT can be computed quite fast.

Other compression techniques include Wavelets.

Image Processing: Segmentation

- This is the process of taking as input, an image, and producing as output, a way of "cutting up" the image into disjoint regions such that each region is "homogeneous".
- Suppose I is an image containing $(m \times n)$ cells.
- A *connected region*, $\Re$, in image I, is a set of cells such that if cells $(x_1, y_1), (x_2, y_2) \in \Re$, there there exists a sequence of cells $C_1, \ldots, C_n$ in $\Re$ such that:
  1. $C_1 = (x_1, y_1)$ and
  2. $C_n = (x_2, y_2)$ and
  3. The Euclidean distance between cells C_i and C_{i+1} for all $i < n$ is 1.

Example: Connected Regions



- Each of $R1, R2, R3$ is a connected region.
- $(R_1 \cup R_2)$ is a connected region;
- $(R_2 \cup R_3)$ is a connected region;
- $(R_1 \cup R_2 \cup R_3)$ is a connected region;
- But $(R_1 \cup R_3)$ is *not* a connected region. The reason for this is that the Euclidean distance between the cell $(2, 3)$ which represents the rightmost of the two cells of $R1$ and the cell $(3, 4)$ which represents the only cell of R_3 is $\sqrt{2} > 1$.

Homogeneity Predicates

- A *homogeneity predicate* associated with an image I is a function H that takes as input, any connected region$\Re$ in image I, and returns either "true" or "false."
- **Example:** Suppose δ is some real number between 0 and 1, inclusive, and we are considering black and white images. We may define a simple homogeneity predicate, H_{δ}^{bw} as follows: $H_{\delta}^{bw}(R)$ returns "true" if over $(100 * \delta)\%$ of the cells in region R have the same color.
  - Consider three regions now, as described in the following table:

Region	Num. of Black Pixels	Num. of White Pixels
R_1	800	200
R_2	900	100
R_3	100	900

  - Suppose we consider some different predicates, $H_{0.8}^{bw}$, $H_{0.89}^{bw}$ and $H_{0.92}^{bw}$.
  - The following table shows us the results returned by these three homogeneity predicates on the above table R.

Region	$H_{0.8}^{bw}$	$H_{0.89}^{bw}$	$H_{0.92}^{bw}$
R_1	true	false	false
R_2	true	true	false
R_3	true	true	false

Another Homogeneity Predicate Example

- Suppose each pixel has a real value between 0 and 1, inclusive.
- This value is called the bw-level.
- 0 denotes "white", 1 denotes "black", and everything in between denotes a shade somewhere between black and white.
- Suppose f assigns numbers between 0 and 1 (inclusive) to each cell. In addition, you have a "noise factor" $0 \leq \eta \leq 1$, and a threshold δ as in the preceding case.
- $H^{f,\eta,\delta}(R)$ is now "true" iff

$$\frac{\{(x,y) \mid \ |\mathsf{bwlevel}(x,y) - f(x,y)| < \eta\}}{(m \times n)} > \delta.$$

- What this homogeneity predicate does is to use a "baseline" function f, and a maximal permissible noise level η. It considers the bw-level of cell (x,y) to be sufficiently similar to that predicted by f if

$$|\mathsf{bwlevel}(x,y) - f(x,y)| < \eta,$$

  i.e. if the two differ by no more than η.
- It then checks to see if sufficiently many cells (which is determined by the factor δ) in the region "match" the predictions made by f. If so, it considers the region R to be homogeneous, and returns "true." Otherwise, it returns "false."

Segmentation

Def: Given an image I represented as a set of $(m \times n)$ pixels, we define a *segmentation* of image I w.r.t. a homogeneity predicate P to be a set $R_1, \ldots, R_k$ of regions such that:

1. $R_i \cap R_j = \emptyset$ for all $1 \leq i \neq j \leq k$ and
2. $I = R_1 \cup \cdots \cup R_k$;
3. $H(R_i) =$"true" for all $1 \leq i \leq k$;
4. For all distinct i, j, $1 \leq i, j \leq n$ such that $R_i \cup R_j$ is a connected region,it is the case that $H(R_i \cup R_j) =$"false."

Example: For example, consider a simple (4×4) region containing the bw-levels shown in the table below.

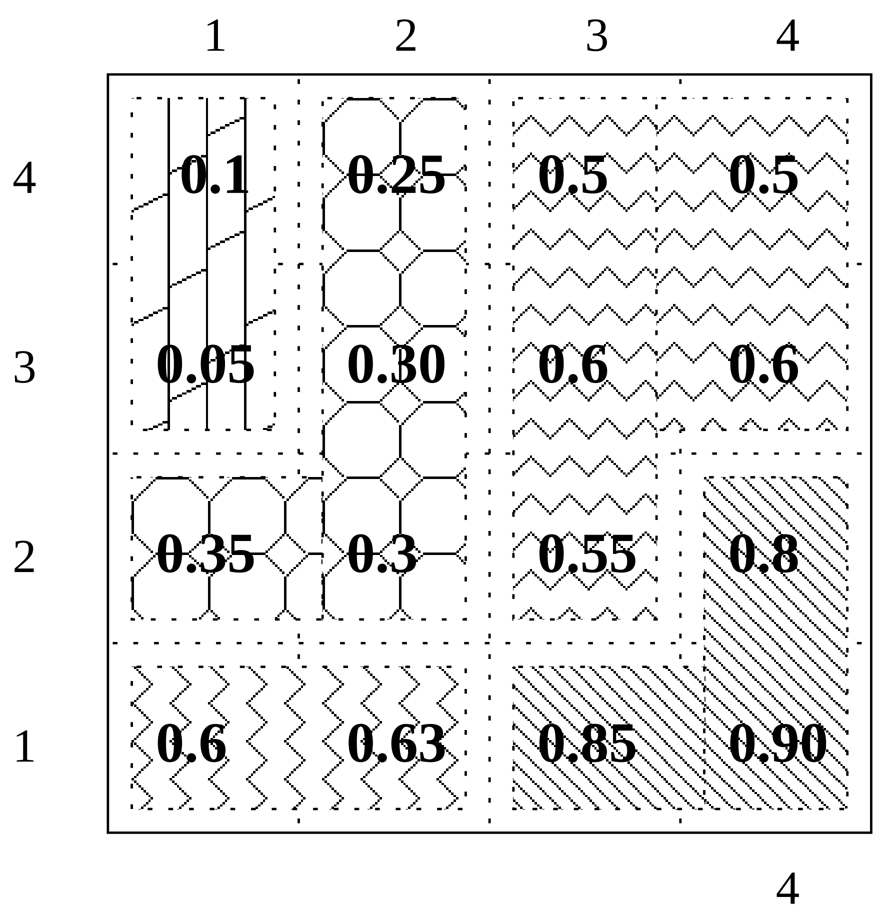
Row/Col	1	2	3	4
1	0.1	0.25	0.5	0.5
2	0.05	0.30	0.6	0.6
3	0.35	0.30	0.55	0.8
4	0.6	0.63	0.85	0.90

Consider now, the homogeneity predicate $H_1^{dyn,0.03}$. This homogeneity predicate says that a region R is to be considered homogeneous iff there exists an r such that each and every cell in the region has a bw-level v such that

$$|v - r| \leq 0.03.$$

According to this classification, it is easy to see that the following five regions constitute a valid segmentation of the above image w.r.t. $H_1^{dyn,0.03}$.

$$\begin{aligned}
R_1 &= \{(1,1),(1,2)\}. \\
R_2 &= \{(1,3),(2,1),(2,2),(2,3)\}. \\
R_3 &= \{(3,1),(3,2),(3,3),(4,1),(4,2)\}. \\
R_4 &= \{(3,4),(4,3),(4,4)\}. \\
R_5 &= \{(1,4),(2,4)\}.
\end{aligned}$$



Segmentation Algorithm Sketch

- **Split:** In this method, we start with the whole image. If it is homogeneous, then we are done, and the image is a valid segmentation of itself. Otherwise, we split the image into two parts, and recursively repeat this process, till we find a set $R_1, \ldots, R_n$ of regions that are homogeneous, and satisfy all conditions, except the fourth condition in the definition of "homogeneity" predicates.
- **Merge:** We now check which of the R_i's can be merged together. At the end of this step, we will obtain a valid segmentation $R'_1, \ldots, R'_k$ of the image, where $k \leq n$ and where each R'_i is the union of some of the R_j's.

Segmentation Algorithm

```
function segment(I:image);
   SOL = ∅;
   check_split(I);
   merge(SOL);
end function

function check_split(R);
   if H(R) =``true'' then addsol(R)
   else
     { X = split(R);
       check_split(X.part1);
       check_split(X.part2);
     }
end function

procedure addsol(R);
   SOL = SOL ∪ {R}
end procedure
```

Segmentation Algorithm (Contd.)

function `merge(S);`
  while $S \neq \emptyset$ **do** {
    Pick some $Cand$ in S;
    merged = false;
    S = S $-\{Cand\}$;
    Enumerate S as $C_1, \ldots, C_k$;
    while $i \leq k$ **do**
      { **if** adjacent(Cand,C_i) **then**
        { Cand = Cand $\cup C_i$;
          S= S $-\{C_i\}$;
          merged= true;
        }
      else $\{i = i + 1;$
          if merged **then** S=S $\cup \{Cand\}$;
          merged=false
          }
      }};

end function

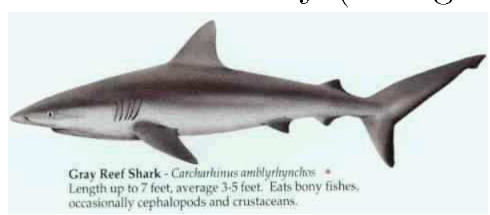
Similarity Based Retrieval

Which of these images is similar to the other?

(a) One Monkey (chimp)

(b) Another Monkey (orangutan)

(a) One Shark (tiger)



(b) Another Shark (grayreef)

Similarity Based Retrieval

Two approaches:

- **(The Metric Approach)** Assume there is a distance metric d that can compare any two image objects. The closer two objects are in distance, the more similar they are considered to be. *Given an input image i, find the "nearest" neighbor of i in the image archive.* This is the most widely followed approach in the database world.
- **(The Transformation Approach)** The metric approach assumes that the notion of similarity is "fixed", i.e. in any given application only one notion of similarity is used to index the data (though different applications may use different notions of similarity). Computes the "cost" of transforming one image into another based on user-specified cost functions that may vary from one query to another.

Metric Appraoch

- Suppose we consider a set Obj of objects, having pixel properties $p_1, \ldots, p_n$, as described earlier in this chapter. Thus, each object o may be viewed as a set $S(o)$
- A function d from some set X to the unit interval $[0,1]$ is said to be a distance function if it satisfies the following axioms for all $x, y, z \in X$:

$$\begin{aligned} d(x,y) &= d(y,x). \\ d(x,z) &\leq d(x,z) + d(z,y). \\ d(x,x) &= 0. \end{aligned}$$

- Let d_{Obj} be a distance function on the space of all objects in our domain, i.e. d_{Obj} is a distance function on a $k = (n+2)$ dimensional space.
- **Example:** Obj to consist of (256×256) images having three attributes (red, green, blue) each of which assumes a value from the set $\{0, \ldots, 7\}$. Could have:

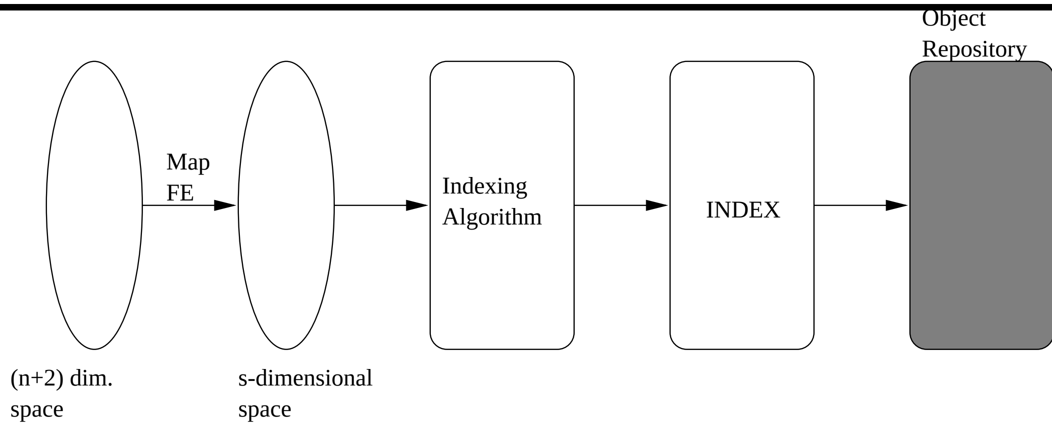
$$\begin{aligned} d_i(o_1, o_2) &= \sqrt{\sum_{i=1}^{256} \sum_{j=1}^{256} (diff_r[i,j] + diff_g[i,j] + diff_b[i,j])} \\ diff_r[i,j] &= (o_1[i,j].red - o_2[i,j].red)^2 \\ diff_g[i,j] &= (o_1[i,j].green - o_1[i,j].green)^2 \\ diff_b[i,j] &= (o_1[i,j].blue - o_1[i,j].blue)^2 \end{aligned}$$

- Such computations can be cumbersome because the double summation leads to 65536 expressions being computed inside the sum.

Metric Approach

- How can this massive similarity computation be avoided?
- Suppose we have a "good" feature extraction function <u>fe</u>.
- Use <u>fe</u> to map objects into single points in a s-dimensional space, where s would typically be pretty small compared to $(n+2)$.
- This leads to two reductions:
  1. First, recall that an object o is a *set of points* in an $(n+2)$ dimensional space. In contrast, <u>fe</u>(o) is a *single point*.
  2. Second, <u>fe</u>(o) is a point in an s-dimensional space where $s \ll (n+2)$.
- Mapping must preserve distance, i.e. if o_1, o_2, o_3 are objects such that the distance $d(o_1, o_2) \leq d(o_1, o_3)$, then $d'(\underline{\text{fe}}(o_1), \underline{\text{fe}}(o_2)) \leq d'(\underline{\text{fe}}(o_1), \underline{\text{fe}}(o_3))$ where d is a metric on the original $(n+2)$ dimensional space, and d' is a metric on the new, s-dimensional space. In other words, the feature extraction map should preserve the distance relationships in the original space.

Reducing Dimensionality of Feature Space



Index Creation Algorithm

Input: Obj, a set of objects.

1. $T = NIL$. (∗ T is an empty quadtree, or R-tree for s-dimensional data ∗)
2. **if** $Obj = \emptyset$ **then return** T and **halt**.
3. **else**
   (a) Compute <u>fe</u>(o),
   (b) Insert <u>fe</u>(o) into T.
   (c) $Obj = Obj - \{o\}$.
   (d) Goto 2

Finding the Best Matches

FindMostSimilarObject Algorithm

Input: a tree T of the above type. An object o.

1. bestnode = NIL;
2. **if** $T = NIL$ **then return** bestnode. **Halt**
3. **else**

   - find the nearest neighbors of $\underline{\mathsf{fe}}(o)$ in T using a nearest neighbor search technique. If multiple such neighbors exist, return them all.

Finding "Sufficiently" Similar Objects

FindSimilarObjects Algorithm

Input: a tree T of the above type. An object o. A tolerance $0 < \epsilon \leq 1$.

1. Execute a range query on tree T with center $\underline{\mathsf{fe}}(o)$ and radius ϵ.
2. Let $o_1, \ldots, p_r$ be all the points returned.
3. **for** $i = 1$ **to** r **do**
   (a) if $d(o, \underline{\mathsf{fe}}^{-1}(o_i)) \leq \epsilon$ **then print** $\underline{\mathsf{fe}}^{-1}(o_i)$.

The above algorithm works *only if* the distance metric in the space of small dimensionality (i.e. dimension s) *consistently overestimates* the distance metric d.

The Transformation Approach

- Based on the principle that given two objects o_1, o_2, the level of *dis-similarity* between o_1, o_2 is proportional to the (minimum) cost of transforming object o_1 into object o_2, or vice-versa.
- We start with a set of *transformation operators*, $to_1, \ldots, to_r$, e.g.
  - translation
  - rotation
  - scaling – uniform and nonuniform
  - excision (that culls out a part of an image)
- The transformation of object o into object o' is a *sequence* of transformation operations $to_1, \ldots, to_r$ and a sequence of objects $o_1, \ldots, o_r$ such that:
  1. $to_1(o) = o_1$ and
  2. $to_i(o_{i-1}) = o_i$ and
  3. $to(o_r) = o'$.

  The *cost* of the above transformation sequence, TS is given by:

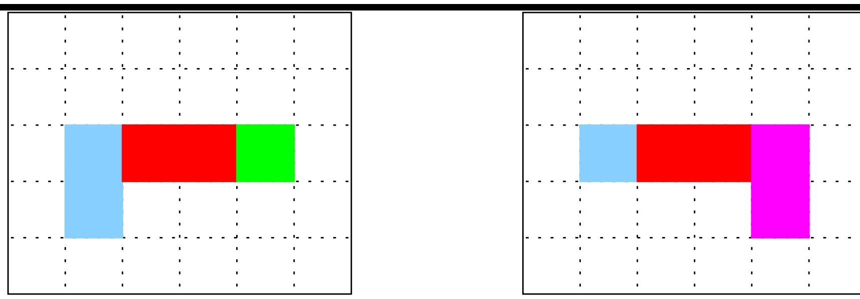
$$\mathsf{cost}(TS) \;=\; \sum_{i=1}^{r} \mathsf{cost}(to_i).$$

The Transformation Approach

- Suppose $\mathsf{TSeq}(o, o')$ is the set of all transformation sequences that convert o into o'.
- *The dissimilarity* between o and o', denoted $\mathsf{dis}(o, o')$ w.r.t. a set TR of transformation operators, and a set CF of cost functions is given by:

$$\mathsf{dis}(o, o') \;=\; \min\{\mathsf{cost}(TS) \mid TS \in \mathsf{TSeq}(o, o') \cup \mathsf{TSeq}(o', o)\}.$$

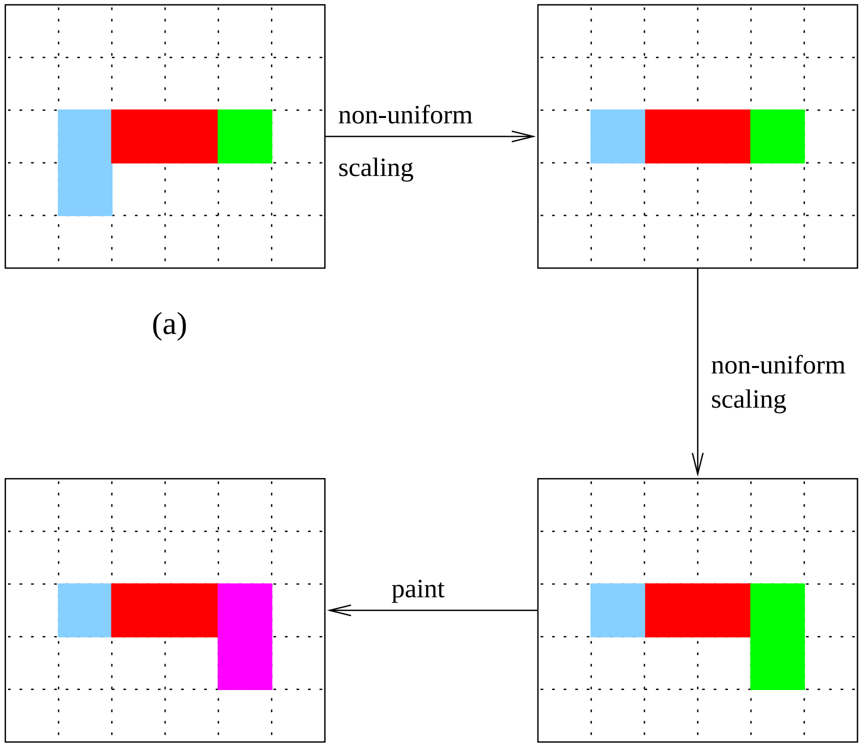
Example



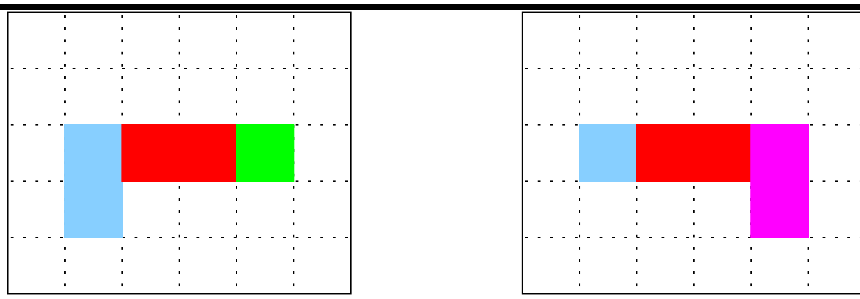
(a) (b)

TS_1: This transformation sequence consists of a

- non-uniform scaling operation (scale the blue part of o_1 50% in the vertical upward direction, leaving the horizontal unchanged),
- a non-uniform scaling operation (scale the green part of object o_1 by a 100% increase in the vertical, downward direction, with no change in the horizontal).
- The third operation applies the `paint` operation, painting the two pixels colored magenta to green.
- The Figure below depicts the intermediate steps.


non-uniform
scaling
(a)
non-uniform
scaling
paint

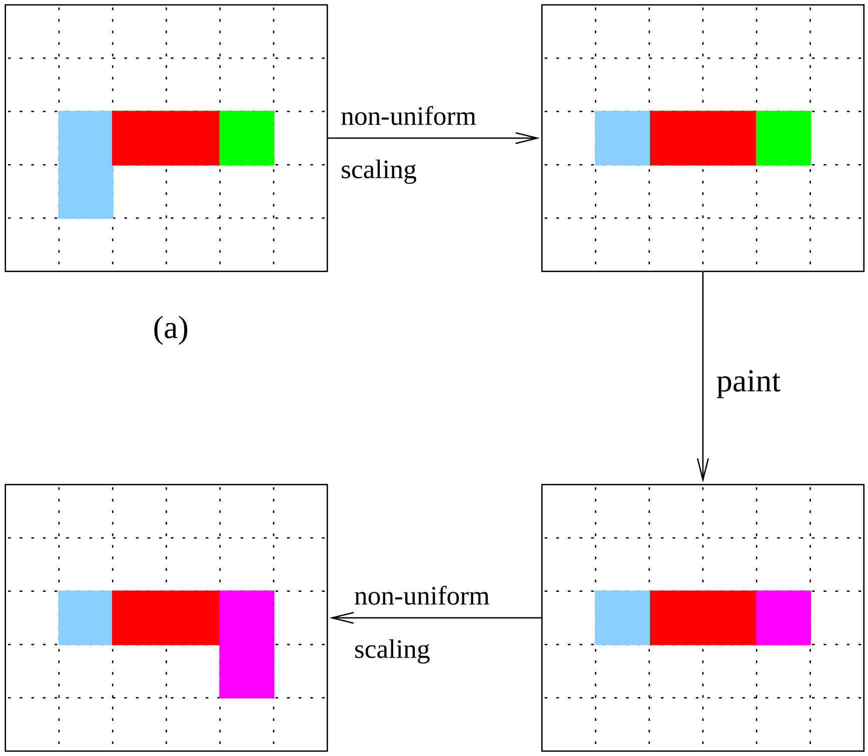
Example



(a) (b)

TS_2: This transformation sequence consists of a

- non-uniform scaling operation (scale the blue part of o_1 50% in the vertical upward direction, leaving the horizontal unchanged).
- apply the `paint` operation, painting the green object magenta.
- apply the non-uniform scaling operation (scale the magenta part of object o_1 by a 100% increase in the vertical, downward direction, with no change in the horizontal).
- The following figure depicts the intermediate steps.



(a)

- If we assume that the cost functions associated with non-uniform scaling are independent of color, and the `paint` operation merely counts the number of pixels being painted, then it is easy to see that transformation TS_2 accomplishes the desired transformation at a cheaper cost (as it paints one less pixel than transformation TS_1.

Transformation Model vs. the Metric Model

Advantages of the Transformation Model over the Metric Model

- First and foremost, the user can "set up" his own notion of similarity by specifying that certain transformation operators may/may not be used.
- Second, the user may associate, with each transformation operator, a *cost function* that assesses a cost to each application of the operation, depending upon the arguments to the transformation operator. This allows the user to personalize the notion of similarity for his/her needs.

Advantages of the Metric Model over the Transformation Model

By forcing the user to use one and only one (dis)similarity metric, the system can facilitate the indexing of data so as to optimize the one operation of finding the "nearest" neighbor (i.e. least dis-similar) object w.r.t. the query object specified by the user.

Alternative Image DB Paradigms

- **Representing `IDBs` as Relations**
- **Representing `IDBs` with Spatial Data Structures**
- **Representing `IDBs` using Image Transformations**

Two different photographs of the same person may vary, depending upon a variety of factors such as:

1. the time of the day at which the two photographs were taken;
2. the lighting conditions under which the photographs were taken;
3. the camera used;
4. the exact position of the subject's head and his/her facial expression.
5. etc.

Representing Image DBs with Relations

Suppose $\texttt{IDB} = (\mathsf{GI}, \mathsf{Prop}, \mathsf{Rec})$.

1. Create a relation called `images` having the scheme:

   $$(\texttt{Image}, \texttt{ObjID}, \texttt{XLB}, \texttt{XUB}, \texttt{YLB}, \texttt{YUB})$$

   where `Image` is the name of an image file, `ObjID` is a dummy name created for an object contained in the image, and `XLB, XUB,YLB,YUB` describe the rectangle in question. If R is a rectangle specified by `XLB, XUB,YLB,YUB` and R is in $\mathsf{Rec}(I)$, then there exists a tuple

   $$(\texttt{I}, \texttt{newid}, \texttt{XLB}, \texttt{XUB}, \texttt{YLB}, \texttt{YUB})$$

   in the relation `images`.

2. For each property $p \in \mathsf{Prop}$, create a relation R_p having the scheme:

   $$(\texttt{Image}, \texttt{XLB}, \texttt{XUB}, \texttt{YLB}, \texttt{YUB}, \texttt{Value}).$$

   Here, `Image` is the name of an image file. Unlike the preceding case though, `XLB,XUB,YLB,YUB` denote a rectangular *cell* in the image, and `Value` specifies the value of the property p.

Example:

Image	ObjId	XLB	XUB	YLB	YUB
pic1.gif	o_1	10	60	5	50
pic1.gif	o_2	80	120	20	55
pic2.gif	o_3	20	65	20	75
pic3.gif	o_4	25	75	10	60
pic4.gif	o_5	20	60	30	80
pic5.gif	o_6	0	40	15	50
pic6.gif	o_7	20	75	15	80
pic6.gif	o_8	20	70	130	185
pic7.gif	o_9	15	70	15	75

Image Properties

- Pixel Level Properties: e.g. RGB values
- Object/Region Level Properties: e.g. NAME, AGE
- Image Level Properties: e.g. when image was captured, where, and by whom

Querying (Relational Representations of) Image DBs

- Eliciting the contents of an image is done using image processing algorithms.
- Image processing algorithms are usually only partially accurate.
- This implies that tuples placed in a relation by an image processing program has certain associated probabilistic attributes.

- Each has object has a value associated with each property $p \in \mathsf{Prop}$.
- Each relation R_p associated with object/region level properties as well as image level properties has just two attributes: an object (or region) id, and a value for the property.
- Example:

Relation `name`

ObjId	Name
o_1	Jim Hatch
o_2	John Lee
o_3	John Lee
o_4	Jim Hatch
o_5	Bill Bosco
o_6	Dave Dashell
o_7	Ken Yip
o_8	Bill Bosco
o_7	Ken Yip

- This description must be enhanced to include probabilistic object identification.

Probabilistic Version of the name relation

Probabilistic Version of the `name` relation

ObjId	Name	Prob
o_1	Jim Hatch	0.8
o_1	Dave Fox	0.2
o_2	John Lee	0.75
o_2	Ken Yip	0.15
o_3	John Lee	1
o_4	Jim Hatch	1
o_5	Bill Bosco	1
o_6	Dave Dashell	1
o_7	Ken Yip	0.7
o_7	John Lee	0.3
o_8	Bill Bosco	0.6
o_8	Dave Dashell	0.2
o_8	Jim Hatch	0.10
o_9	Ken Yip	1

Reading:

1. The probability that "John Lee" is the name attribute of o_2 is 0.75.
2. The probability that "Ken Yip" is the name attribute of o_2 is 0.15.
3. There is, in this case, a 10% missing probability.

Complex Queries

- Suppose we ask the query *What is the probability that* `pic1.gif` *contains both Jim Hatch and Ken Yip?* Is the answer the product of the two probabilities, i.e. is it $(0.8 \times 0.15) = 0.12$?
- *What is the probability that* `pic6.gif` *contains both Jim Hatch and Ken Yip?* Is the answer $(0.7 \times 0.1) = 0.07$?
- In general, the answer is NO.

Example: Consider a hypothetical image `pic8.gif` with two objects o_{10}, o_{11} in it, and suppose our table above is expanded by the insertion of the following new tuples identified by the image processing algorithm.

ObjId	Name	Prob
o_{10}	Ken Yip	0.5
o_{10}	Jim Hatch	0.4.
o_{11}	Jim Hatch	0.8
o_{11}	John Lee	0.1

If we are ignorant about the dependencies between different events (as we are in the above case, then we are forced to confront *four possibilities*:

Possibility 1 o_{10} is Ken Yip and o_{11} is John Hatch.

Possibility 2 o_{10} is Ken Yip and o_{11} is *not* John Hatch.

Possibility 3 o_{10} is *not* Ken Yip but o_{11} is John Hatch.

Possibility 4 o_{10} is *not* Ken Yip and o_{11} is *not* John Hatch.

Complex Queris

- Suppose p_i denotes the probability of Possibility i, $1 \leq i \leq 4$. Then, we can say that:

$$\begin{aligned} p_1 + p_2 &= 0.5. \\ p_3 + p_4 &= 0.5. \\ p_1 + p_3 &= 0.8. \\ p_2 + p_4 &= 0.2 \\ p_1 + p_2 + p_3 + p_4 &= 1. \end{aligned}$$

- The first equation follows from the fact that o_{10} is Ken Yip according to possibilities 1 and 2, and we know from the table, that the probability of o_{10} being Ken Yip is 0.5.
- The second equation follows from the fact that o_{10} is someone other than Ken Yip according to possibilities 3 and 4, and we know from the table, that the probability of o_{10} not being Ken Yip is 0.5.
- The third equation follows from the fact that o_{11} is Jim Hatch according to possibilities 1 and 3, and we know from the table, that the probability of o_{11} being Jim Hatch is 0.8.
- Finally, the last equation follows from the fact that o_{11} is someone other than Jim Hatch according to possibilities 2 and 4, and we know from the table, that the probability of o_{11} not being John Hatch is 0.2.

- *In order to determine the probability that* `pic8.gif` *contains both Ken Yip and John Hatch, we must attempt to solve the above system of linear equations for* p_1*, keeping in mind the fact that all scenarios possible are covered by our four possibilities.* The result we obtain, using a linear programming engine, is that p_1's probability is *not uniquely determinable. It could be as low as* 0.3 *or as high as* 0.5*, or anywhere in between.* In particular, note that merely multiplying the probability of 0.5 associated with Ken Yip being object o_{10} and the probability value 0.8 of m Hatch being object o_{11} leads to a probability of 0.4 which is certainly inside this interval, but does not accurately capture the four possibilities listed above.

- Requires the use of interval probabilities.

Interval Probability Model

Interval Probabilistic Version of the **name** relation with **pic8.gif** included

ObjId	Name	Prob (Lower)	Prob (Upper)
o_1	Jim Hatch	0.77	0.83
o_1	Dave Fox	0.17	0.23
o_2	John Lee	0.72	0.78
o_2	Ken Yip	0.12	0.18
o_3	John Lee	0.97	1.00
o_4	Jim Hatch	0.97	1.00
o_5	Bill Bosco	0.97	1.00
o_6	Dave Dashell	0.97	1.00
o_7	Ken Yip	0.67	0.73
o_7	John Lee	0.27	0.33
o_8	Bill Bosco	0.57	0.63
o_8	Dave Dashell	0.17	0.23
o_8	Jim Hatch	0.07	0.13
o_9	Ken Yip	0.97	1.00
o_{10}	Ken Yip	0.47	0.53
o_{10}	Jim Hatch	0.37	0.43
o_{11}	Jim Hatch	0.77	0.83
o_{10}	Hohn Lee	0.07	0.13

- *"Find an image that contains both Ken Yip and Jim Hatch."*

- Let us re-examine the image `pic8.gif` and see what the probability of this image containing both Ken Yip and Jim Hatch is.
- Constraints generated:

$$0.47 \leq p_1 + p_2 \leq 0.53.$$

$$0.47 \leq p_3 + p_4 \leq 0.53.$$

$$0.77 \leq p_1 + p_3 \leq 0.83.$$

$$0.17 \leq p_2 + p_4 \leq 0.23.$$

$$p_1 + p_2 + p_3 + p_4 = 1.$$

- Solving the above linear program for minimal and maximal values of the variable p_1, we obtain 0.24 and 0.53, respectively.
- Beauty is that when implementing this, we can avoid solving the linear program altogether.

A General Approach

- A *probabilistic relation* over a scheme $(A_1, \ldots, A_n)$ is an ordinary relation over the scheme $(A_1, \ldots, A_n, LB, UB)$ where the domain of the LB and UB attributes is the unit interval $[0, 1]$ of real numbers.
- In particular, the relation `name` is a probabilistic relation that has three attributes:

  $$(\texttt{ImageId}, \texttt{ObjectId}, \texttt{Name})$$

  of the sort we have already seen thus far.
- The `name` relation satisfies some integrity constraints:

  $$(\forall \texttt{t}_1, \texttt{t}_2)\texttt{t}_1.\texttt{ObjId} = \texttt{t}_2.\texttt{ObjId} \rightarrow \texttt{t}_1.\texttt{ImageId} = \texttt{t}_2.\texttt{ImageId}.$$

  This constraint states that an ObjectId can be associated with only one image, i.e. distinct images have distinct ObjectIds. The following constraint says that the LB field of any tuple is always smaller than the UB field.

  $$(\forall \texttt{t})\texttt{t.LB} \leq \texttt{t.UB}.$$
- An *image database* consists of a probabilistic relation called `name` of the above form, together with a set of *ordinary* (i.e. non-probabilistic) relations $R_1, \ldots, R_k$ corresponding to image properties.

Membership Queries

- A *membership query* in an image database is a query of the form: *Find all images in the image database that contain objects named $s_1, \ldots, s_n$.*

- SELECT ImageId
  FROM `name` $\mathtt{T_1, \ldots, T_n}$
  WHERE $\mathtt{T_1.Name = s_1}$ AND$\cdots$ AND $\mathtt{T_n.Name = s_n}$ AND
  $\mathtt{T_1.ImageId = T_2.ImageId}$ AND$\cdots$ AND
  $\mathtt{T_1.ImageId = T_n.ImageId}$.

  The *result of this membership query* is a table containing *three* fields – the `ImageId` fields that is explicitly listed in the query, a `LB` field, and a `UB` field. (im, ℓ, u) is in the result iff for each $1 \leq j \leq n$, there exists a tuple $t_j \in$ `name` such that:

  1. $\mathtt{t.ImageId} = im$ and
  2. $\mathtt{t.LB} = \ell_i$ and $\mathtt{t.UB} = u_i$ and
  3. $[\ell, u] = [\ell_1, \mathtt{u_1}] \otimes [\ell_2, \mathtt{u_2}] \otimes \cdots \otimes [\ell_\mathtt{n}, \mathtt{u_n}]$

  where

$$[x, y] \otimes [x', y'] \ = \ [\max(0, x + x' - 1), \min(y, y')].$$

Other Queries

- 'Find all people who have had deposits of over 9000 dollars, and who have been photographed with Denis Jones.
- SELECT I.ImageId
  FROM name I, bank B
  WHERE I CONTAINS *B.name*, Denis Jones AND
  B.trans=deposit AND B.amount> 9000 AND
  B.name = I.name.

Representing Image DBs with R-Trees

1. Create a relation called `occursin` with two attributes

   $$(\texttt{ImageId}, \texttt{ObjId})$$

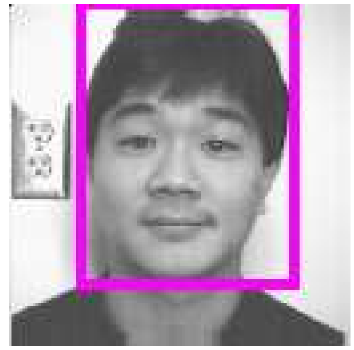
   specifying which objects appear in which images.

2. Create *one* R-tree that stores all the rectangles. If the same rectangle (say with $XLB = 5, XUB = 15, YLB = 20, YUB = 30$) appears in two images, then we have an overflow list associated with that node in the R-tree.

3. Each rectangle has an associated set of fields that specifies the object/region level properties of that rectangle.

Example

pic1.gif



pic2.gif

pic3.gif

pic4.gif

pic5.gif

pic6.gif

pic7.gif

occursin Relation

pic1.gif	o_1
pic1.gif	o_2
pic2.gif	o_3
pic3.gif	o_4
pic4.gif	o_5
pic5.gif	o_6
pic6.gif	o_7
pic6.gif	o_8
pic7.gif	o_9

R-tree representation

facenode = **record**
Rec_1, Rec_2, Rec_3: **rectangle**;
P_1, P_2, P_3: $\uparrow$**rtnodetype**
end

rectangle = **record**
XLB,XUB,YLB,YUB: **integer**;
objlist: $\uparrow$objnode;
day,mth, yr: **integer**;
camera_type: **string**;
place: **string**
end

objnode = **record**
objid: **string**;
imageid: **string**;
info: **infotype**
end

infotype = **record**
objname: **string**;
Lp, Up: **real**: (∗ lower and upper probability bounds ∗)
Next: $\uparrow$**objinfo**
end

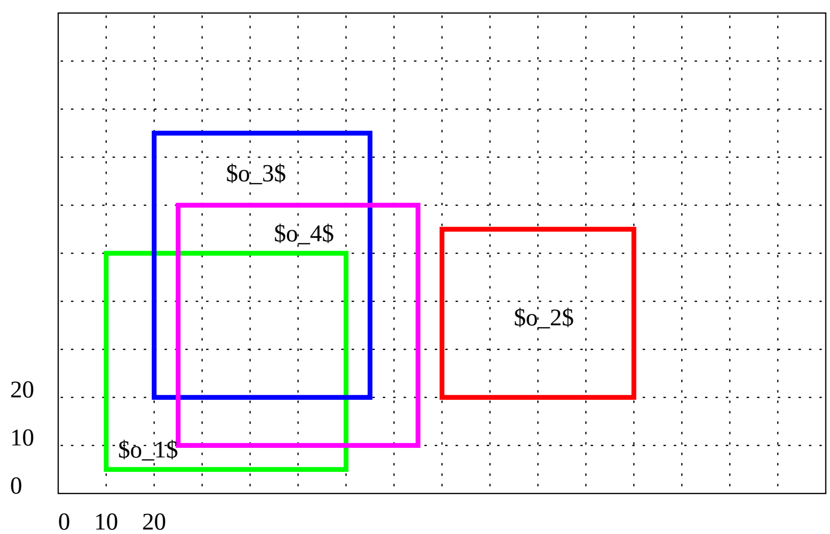
R-Tree Construction

- Get Rectangles
- Create R-tree
- Flesh out Objects

Get Rectangles

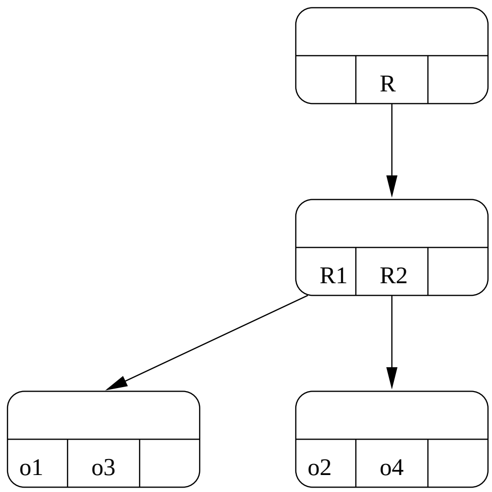
First and foremost, we may construct a small table describing all the rectangles that occur, and the images they occur in.

ObjId	ImageId	XLB	XUB	YLB	YUB
o_1	pic1.gif	10	60	5	50
o_2	pic1.gif	80	120	20	55
o_3	pic2.gif	20	65	20	75
o_4	pic3.gif	25	75	10	60



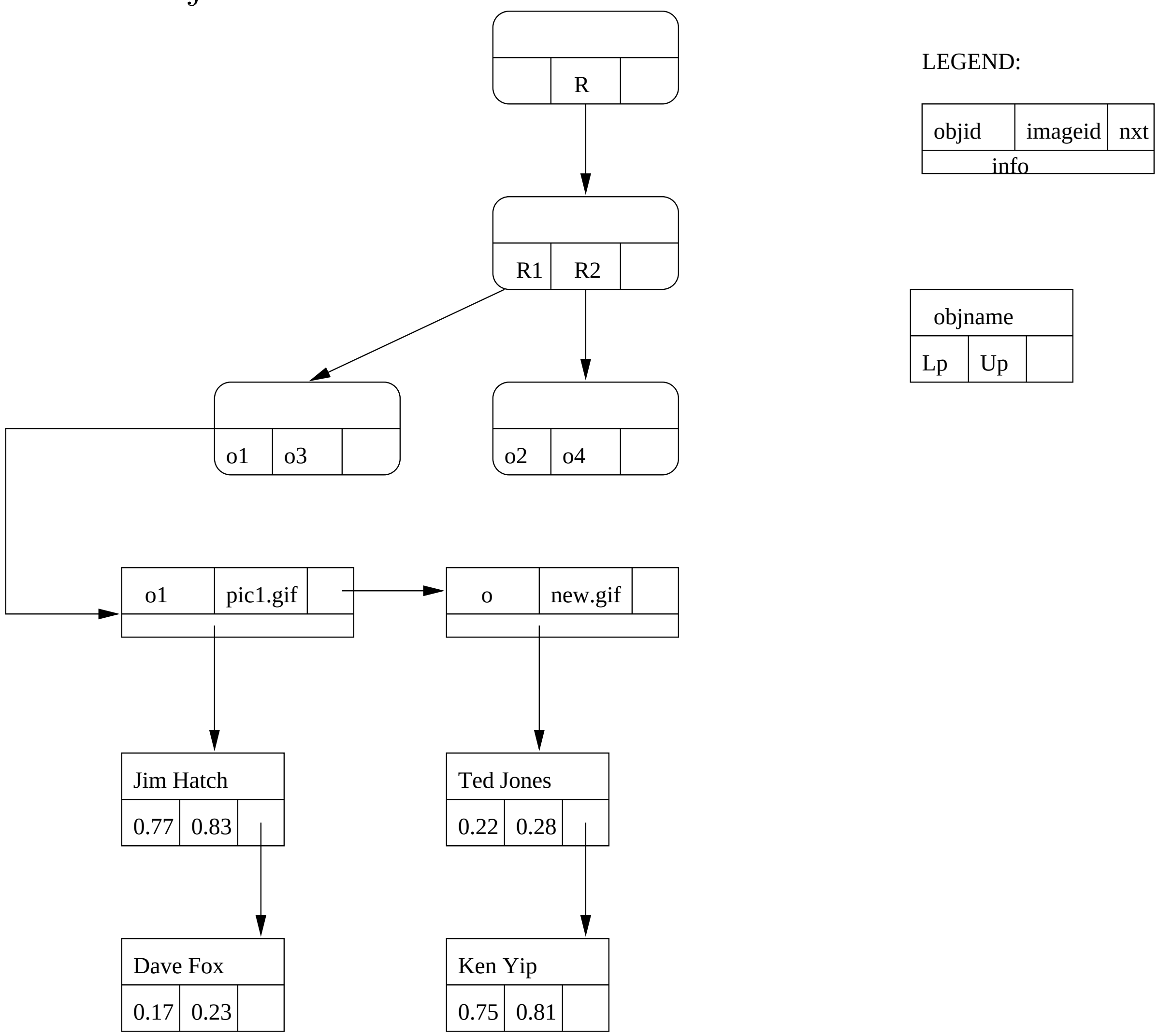
Create R-Tree

We then create an R-tree representing the above rectangles. At this stage the object nodes in the R-tree may still not be "filled in" completely.



Flesh Out Objects

We then "flesh out" and appropriately fill in the fields of the various objects stored in the R-tree.



Generalized R-trees

- Previous representation does not provide an efficient way to perform nearest neighbor searches.
- Why? Only 2-attributes (bounding rectangles) are stored.
- In general, an object o has an associated $(n+2)$-dimensional vector.
- If each point in the 2-d rectangle has n attributes, then we need to use a generalized rectangle.
- A *generalized rectangle* for a space of dimensionality g (specifically consider $g = n + 2$) may be defined by a set of constraints of the form:

$$\ell_1 \leq x_1 \leq u_1$$

$$\ell_2 \leq x_2 \leq u_2$$

$$\dots$$

$$\ell_g \leq x_g \leq u_g$$

- When $g = 2$, we have $n = 0$, and in this case, an ordinary 2-dimensional rectangle is a special case of this definition.
- Sets of generalized rectangles are represented by generalized R-trees, or gR-trees.

gR-Trees

A generalized R-tree (gR-tree) of order K is exactly like an R-tree except for the following factors:

- First, each node N represents a generalized bounding rectangle $GBR(N)$ of dimensionality $(n+2)$, which is represented by $2 \times (n+2)$ real number fields, one for the lower bound and upper bound, respectively, of each dimension.
- When a node N is split, the union of the generalized bounding rectangles associated with its children equals the generalized bounding rectangle associated with N.
- Each node (other than the root and the leaves) contains at most K generalized bounding rectangles and at least $\lceil K/2 \rceil$ generalized rectangles.
- As usual, all $(n+2)$-dimensional "data" rectangles are stored in leaves.

Nearest Rectangular Neighbors

- Suppose R_Q is a query rectangle (which may represent an image object).
- Find all rectangles in a gR-tree T that are as close to R_Q as possible (where closeness is defined by a metric d on points).
- We extend the metric d to apply to rectangles as follows:

$$d(R, R') \;=\; \min\{d(p, p') \mid p \in R, p' \in R'\}.$$

Algorithm

Algorithm 1 *NN_Search_GR(T,R_Q)*

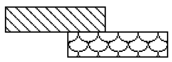
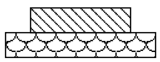
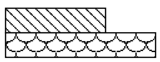
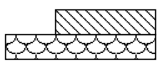
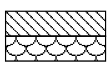
```
SOL = NIL; (* no solution so far *);
Todo = List containing T only;
Bestdist = ∞; (* distance of best solution from R_Q *);
while Todo ≠ NIL do
{
   F = first element of Todo;
   Todo = delete F from Todo;
   if d(GBR(F), R_Q)) < Bestdist then
   {
      Compute children N_1, ..., N_r of F;
      if N_i's are leaves of T then
      {
         N_min = any N_i at minimal distance from R_Q;
         Ndist = d(GBR(N_i), R_Q));
         if Ndist < Bestdist then
         {
            Bestdist = Ndist; SOL = N_min;
         }
      }
      else Todo = insert all N_i's into Todo in order of distance from R_Q;
   }
}
Return SOL;
end
```


Retrieving Images by Spatial Layout

Given an image I, and two objects (represented by rectangles) $o1$, $o2$ in I, a user may wish to ask queries of the form:

1. Is $o1$ to the South of $o2$?
2. Is $o1$ to the South-East of $o2$? item Is $o1$ to the left of $o2$?
3. Are $o1$ and $o2$ overlapping ?

One-dimensional Precedence Relations

o1 o2	Description	Notation
	o1 before o2	B(o1,o2)
	o1 meets o2	M(o1,o2)
	o1 overlaps o2	OV(o1,o2)
	o1 during o2	D(o1,o2)
	o1 starts o2	S(o1,o2)
	o1 finishes o2	F(o1,o2)
	o1 equal o2	EQ(O1,o2)

2-Dimensional Precedence Relations

It is easy to extend the precedence relation along one dimension, to the case of two dimensions, is straightforward. If we use the notation $o[x]$ and $o[y]$ to denote the projection of object o on the x and y axes, respectively, then it is easy to capture our spatial relationships as follows:

1. We say $o1$ is South of $o2$ iff $B(o1[y], o2[y])$ and either (i) $D(o1[x], o2[x])$ or (ii) $D(o2[x], o1[x])$ or (iii) $S(o1[x], o2[x])$ or (iv) $S(o2[x], o1[x])$ holds or (v) $F(o1[x], o2[x])$ or (vi) $F(o2[x], o1[x])$ holds or (vii) $E(o1[x], o2[x])$ holds.
2. Likewise, we say that $o1$ is to the Left of $o2$ iff either (i) $B(o1[x], o2[x])$ holds or (ii) $M(o1[x], o2[x])$ holds.

Text/Document Databases

What is a text database?

- User wants to find documents related related to a topic T.
- The search program tries to find the documents in the "document database" that contain the string T.
- This has two problems:
  1. **Synonymy:** Given a word T (i.e. specifying a topic), the word T does not occur anywhere in a document D, even though the document D is in fact closely related to the topic T in question.
  2. **Polysemy:** The same word may mean many different things in different contexts.
- Consider a text database that only indexes the following titles.

DocumentID	String
d_1	Jose Orojuelo's Operations in Bosnia.
d_2	The Medelin Cartel's Financial Organization.
d_3	The Cali Cartel's Distribution Network.
d_4	Banking Operations and Money Laundering.
d_5	Profile of Hector Gomez.
d_6	Connections between Terrorism and Asian Dope Operations.
d_7	Hector Gomez's: How he Gave Agents the Slip in Cali.
d_8	Sex, Drugs, and Videotape.
d_9	The Iranian Connection.
d_{10}	Boating and Drugs: Slips owned by the Cali Cartel.

Organization of this Topic

- Measures of performance of a text retrieval system.
- Latent Semantic Indexing
- Telescopic-Vector Trees for Document Retrieval.

Precision and Recall

- Suppose D is a finite set of documents.
- Suppose $\mathcal{A}$ is any algorithm that takes as input, a topic string T, returns as output, a set $(\mathcal{T})$ of documents.
- **Precision** of algorithm $\mathcal{A}$ w.r.t. the predicate *relevant* and test set D_{test} is P_t% for topic $t \in T_{test}$ iff:

$$P_t = 100 \times \frac{1 + card(\{d \in D_{test} \mid d \in \mathcal{A}(t) \wedge relevant(t,d) \text{ is true}\})}{1 + card(\{d \in D_{test} \mid d \in \mathcal{A}(t)\})}.$$

(To avoid division by zero, we add one to both the numerator and denominator above). We say that the precision of algorithm $\mathcal{A}$ w.r.t. the predicate *relevant*, the document test set D_{test} and the topic test set T_{test} is P% iff:

$$P = \frac{\Sigma_{t \in T_{test}} P_t}{card(T_{test})}.$$

- Precision basically says: How many of the answers returned are in fact correct.
- **Recall** of an algorithm $\mathcal{A}$ is a measure of "how many" of the right documents are in fact retrieved by the query.
- Rrecall R_t associated with a topic t is given by the formula:

$$R_t = 100 \times \frac{1 + card(\{d \in D_{test} \mid d \in \mathcal{A}(t) \wedge relevant(t,d) \text{ is true}\})}{1 + card(\{d \in D_{test} \mid relevant(t,d) \text{ is true}\})}.$$

The overall recall rate R associated with test sets D_{test} of documents, and T_{test} of topics, is given by:

$$R = \frac{\sum_{t \in T_{test}} R_t}{card(T_{test})}.$$

Stop Lists, Word Stems, and Frequency Tables

- **Stop List:** This is a set of words that donot "discriminate" between the documents in a given archive.
- E.g. Cornell SMART system has about 440 words on its stop list.
- **Word Stems:** Many words are small syntactic variants of each other. For example, the words **`drug`**, **`drugged`**,**`drugs`**, are all similar in the sense that they share a common "stem", viz. the word **`drug`**.
- Most document retrieval systems first eliminate words on stop lists and they also reduce words to their stems, before creating a frequency table.
- **Frequency Tables:** Suppose
  - D is a set of N documents, and
  - T is a set of M words/terms occurring in the documents of D.
  - Assume that no words on the stop list for D occur in T,and that all words in T have been stemmed.

  The *frequency table*, FreqT, associated with D and T is an $(M \times N)$ matrix such that $\mathsf{FreqT}(i, j)$ equals the number of occurrences of the word t_i in document d_j.

Example

DocumentID	String
d_1	Jose Orojuelo's Operations in Bosnia.
d_2	The Medelin Cartel's Financial Organization.
d_3	The Cali Cartel's Distribution Network.
d_4	Banking Operations and Money Laundering.
d_5	Profile of Hector Gomez.
d_6	Connections between Terrorism and Asian Dope Operations.
d_7	Hector Gomez's: How he Gave Agents the Slip in Cali.
d_8	Sex, Drugs, and Videotape.
d_9	The Iranian Connection.
d_{10}	Boating and Drugs: Slips owned by the Cali Cartel.

The associated frequency table might be given by:

Term/Doc	d_8	d_9	d_{10}	d_{11}
sex	1	0	0	0
drug	1	0	1	3
videotape	1	0	0	0
iran	0	1	0	0
connection	0	1	0	0
boat	0	0	1	0
slip	0	0	1	0
own	0	0	1	0
cali	0	0	1	0
cartel	0	0	1	0

Another example

Consider the frequency table shown below.

Term/Doc	d_1	d_2	d_3	d_4	d_5	d_6
t_1	615	390	10	10	18	65
t_2	15	4	76	217	91	816
t_3	2	8	815	142	765	1
t_4	312	511	677	11	711	2
t_5	45	33	516	64	491	59

- d_1, d_2 are similar because the distribution of the words in d_1 "mirrors" the distribution of the words for d_2.
- Both contain lots of occurrences of t_1, t_4 and relatively few occurrences of t_2, t_3, and moderately many occurrences of t_5.
- d_3 and d_5 are aalso similar.
- But d_4, d_6 stand out as sharply different.
- But is this enough?
- Merely counting words does not indicate the importance of the words, in the document. What about document lengths?
- Usually, a frequency table represents not just the number of occurrences of a (stemmed) word in a document,but also its important.

Queries

- User wants to execute the query: *Find the 25 documents that are maximally relevant w.r.t. banking operations and drugs?*.
- Query Q is trying to retrieve documents relevant to two keywords, which after "stemming" are:

  drug, **bank**.

- Think of the query Q as a document. Thus, Q is a column vector.
- We want to find the columns in FreqT that are "as close" as possible to the vector associated with Q.
- Example closeness metrics include:
  1. **Term Distance:** Suppose $vec_Q(i)$ denotes the number of occurrences of term t_i in Q. Then the *term distance* between Q and document d_r is given by:

     $$\sqrt{\sum_{j=1}^{M} (vec_Q(j) - \mathsf{FreqT}(j,r))^2}.$$

     Of course, this is a rather arbitrary metric.
  2. **Cosine Distance:** This metric is used extensively in the document database world, and it may be described as follows:

     $$\frac{\Sigma_{j=1}^{M}(vec_Q(j) \times \mathsf{FreqT}(j,r))}{\sqrt{\Sigma_{j=1}^{M} vec_Q(j)^2} \times \sqrt{\Sigma_{j=1}^{M} \mathsf{FreqT}(j,r)^2}}.$$

- Complexity of retrievals may be O($N \times M$) which could be staggering in size.

Latent Semantic Indexing: The Basic Idea

- The number of documents M and the number of terms N is very large. For example, N could be over 10,000,000, as English words as well as proper nouns can be indexed.
- What LSI tries to do is to find a relatively small subset of K words which discriminate between the M documents in the archive.
- LSI is claimed to work effectively for around $K = 200$.
- Advantage: Each document is now a column vector of length 200, instead of length N.
- This is obviously a big plus.
- Bottom line: *How do we find a relatively small subset of K words which discriminate between the M documents in the archive.*
- Use a technique called singular valued decomposition.
- 4-step approach used by LSI:

  1. **(Table Creation)** Create frequency matrix FreqT.
  2. **(SVD Construction)** Compute the singular valued decompositions, (A, S, B) of FreqT.
  3. **(Vector Identification)** For each document d, let $vec(d)$ be the set of all terms in FreqT whose corresponding rows have not been eliminated in the singular matrix S.

4. **(Index Creation)** Store the set of all $vec(d)$'s, indexed by any one of a number of techniques (later we will discuss one such technique called a TV-tree).

Singular Valued Decompositions

- A matrix $\mathcal{M}$ is said to be of order $(m \times n)$ if it has m rows and n columns.
- If $\mathcal{M}_1, \mathcal{M}_2$ are matrices of order $(m_1 \times n_1)$ and $(m_2 \times n_2)$ respectively, then we say that the product, $(\mathcal{M}_1 \times \mathcal{M}_2)$, is *well defined* iff $n_1 = m_2$.
- If:

$$\mathcal{M}_1 = \begin{pmatrix} a_1^1 & a_2^1 & \cdots & a_{m_1}^1 \\ a_1^2 & a_2^2 & \cdots & a_{m_1}^2 \\ \cdots & \cdots & \cdots & \cdots \\ a_1^{n_1} & a_2^{n_1} & \cdots & a_{m_1}^{n_1} \end{pmatrix}; \mathcal{M}_2 = \begin{pmatrix} b_1^1 & b_2^1 & \cdots & b_{m_2}^1 \\ b_1^2 & b_2^2 & \cdots & b_{m_2}^2 \\ \cdots & \cdots & \cdots & \cdots \\ b_1^{n_2} & b_2^{n_2} & \cdots & b_{m_2}^{n_2} \end{pmatrix}$$

then the product, $(\mathcal{M}_1 \times \mathcal{M}_2)$ is the matrix

$$(\mathcal{M}_1 \times \mathcal{M}_2) = \begin{pmatrix} c_1^1 & c_2^1 & \cdots & c_{m_1}^1 \\ c_1^2 & c_2^2 & \cdots & c_{m_1}^2 \\ \cdots & \cdots & \cdots & \cdots \\ c_1^{n_2} & c_2^{n_2} & \cdots & c_{m_1}^{n_2} \end{pmatrix}$$

where:

$$c_j^i = \sum_{r=1}^{n_1} \left(a_r^i \times b_j^r \right).$$

- EX:

$$\begin{pmatrix} 3 & 2 \\ 4 & 8 \end{pmatrix} \times \begin{pmatrix} 1 & 4 & 3 \\ 2 & 4 & 6 \end{pmatrix} = \begin{pmatrix} 7 & 20 & 21 \\ 20 & 48 & 60 \end{pmatrix}.$$

SVD Continued

- Given a matrix $\mathcal{M}$ of order $(m \times n)$, the *transpose* of $\mathcal{M}$, denoted $\mathcal{M}^T$, is obtained by converting each row of $\mathcal{M}$, into a column of $\mathcal{M}^T$.

- EX:

$$\begin{pmatrix} 7 & 20 & 21 \\ 20 & 48 & 60 \end{pmatrix}^T = \begin{pmatrix} 7 & 20 \\ 20 & 48 \\ 21 & 60 \end{pmatrix}.$$

- Vector = matrix of order $(1 \times m)$.

- Two vectors x, y of the same order are said to be *orthogonal* iff $x^T y = 0$.

- EX:

$$\begin{aligned} x &= (10, 5, 20). \\ y &= (1, 2, -1). \end{aligned}$$

These two vectors are orthogonal because

$$x^T y = \begin{pmatrix} 10 \\ 5 \\ 20 \end{pmatrix} \times (1 \quad 2 \quad -1) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

SVD Continued

- Matrix $\mathcal{M}$ is said to be **orthogonal** iff $(\mathcal{M}^T \times \mathcal{M})$ is the identity matrix (i.e. the matrix, all of whose entries are 1). For example, consider the matrix:

$$\mathcal{M} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$

  This matrix is othogonal.

- Matrix $\mathcal{M}$ is said to be a *diagonal matrix* iff the order of $\mathcal{M}$ is $(m \times m)$ and for all $1 \leq i, j \leq m$, it is the case that:

$$i \neq j \rightarrow \mathcal{M}(i,j) = 0.$$

- EX: A and B below are diagonal matrices, but C is not:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{pmatrix}; B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}; C = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}.$$

- Diagonal matrix $\mathcal{M}$ of order $(m \times m)$ is said to be *non-decreasing* iff for all $1 \leq i, j \leq m$,

$$i \leq j \rightarrow \mathcal{M}(i,i) \leq \mathcal{M}(j,j).$$

- Above, A is a non-decreasing diagonal matrix, but B is not.

SVD Continued

- A *Singular Value Decomposition* of FreqT is a triple (A, S, B) where:
  1. $\mathsf{FreqT} = (A \times S \times B^T)$ and
  2. A is an $(M \times M)$ orthogonal matrix such that $A^T A = I$ and
  3. B is an $(N \times N)$ orthogonal matrix such that $B^T B = I$ and
  4. S is a diagonal matrix called a *singular matrix*.
- Theorem: Given any matrix $\mathcal{M}$ of order $(m \times n)$, it is possible to find a singular value decomposition, (A, S, B) of $\mathcal{M}$ such that S is a *non-decreasing* diagonal matrix.
- EX: The SVD of the matrix

$$\begin{pmatrix} 1.44 & 0.52 \\ 0.92 & 1.44 \end{pmatrix}$$

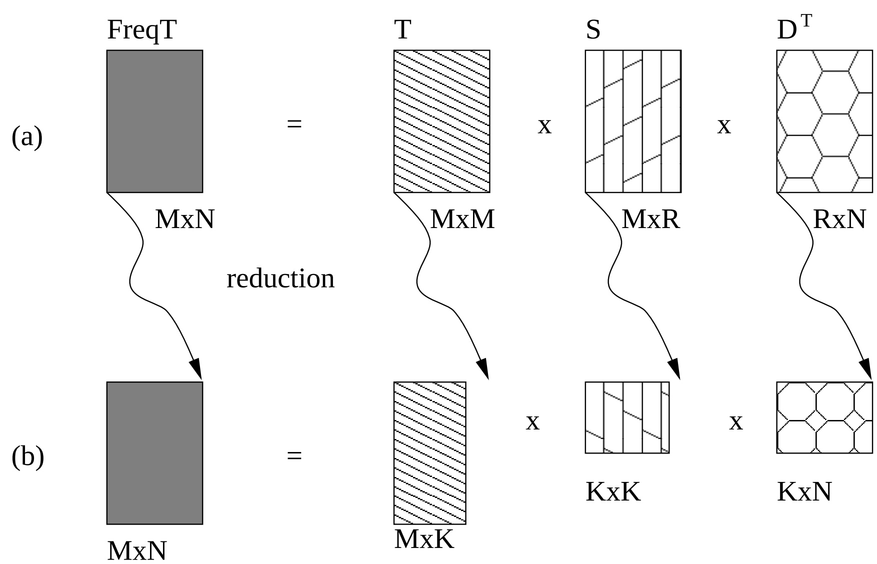
is given by:

$$\begin{pmatrix} 0.6 & -0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 0.8 & 0.6 \\ 0.6 & -0.8 \end{pmatrix}.$$

Here, the singular values are 5 and 2, and it is easy to see that the singular matrix is non-decreasing.

Returning to LSI

- Given a frequency matrix FreqT, we can decompose it into an SVD $\mathcal{TSD}^T$ where $\mathcal{S}$ is non-decreasing.
- If FreqT is of size $(M \times N)$, then $\mathcal{T}$ is of size $(M \times M)$ and $\mathcal{S}$ is of order $(M \times R)$ where R is the rank of FreqT, and $\mathcal{D}^T$ is of order $(R \times N)$.
- We can now "shrink" the problem substantially by eliminating the least significant singular values from the singular matrix $\mathcal{S}$.



LSI Continued

Shrinking the matrices is done as follows.

- Choose an integer k that is substantially smaller than R.
- Replace $\mathcal{S}$ by $\mathcal{S}^\star$, which is a $(k \times k)$ matrix, such that $\mathcal{S}^\star(i,j) = \mathcal{S}(i,j)$ for $1 \leq i,j \leq k$.
- Replace the $(R \times N)$ matrix $\mathcal{D}^T$ by the $(k \times N)$ matrix $\mathcal{D}^{\star T}$ where: $\mathcal{D}^{\star T}(i,j) = \mathcal{D}^T(i,j)$ if $1 \leq i \leq k$ and $1 \leq j \leq N$.

Bottom line:

- Throw away the least significant values, and retain the rest of the matrix involved.
- Key claim in LSI Is that if k is chosen judiciously, then the k rows appearing in the singular matrix $\mathcal{S}^\star$ represent the k "most important" (from the point of view of retrieval) terms occurring in the entire document collection.

Example

Suppose FreqT has the SVD

$$\begin{pmatrix} a_1^1 & a_2^1 & a_3^1 & a_4^1 & a_5^1 \\ a_1^2 & a_2^2 & a_3^2 & a_4^2 & a_5^2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_1^M & a_2^M & a_3^M & a_4^M & a_5^M \end{pmatrix} \begin{pmatrix} 20 & 0 & 0 & 0 & 0 \\ 0 & 16 & 0 & 0 & 0 \\ 0 & 0 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0.08 & 0 \\ 0 & 0 & 0 & 0 & 0.004 \end{pmatrix} \begin{pmatrix} b_1^1 & b_2^1 & b_3^1 & \cdots & b_N^1 \\ b_1^2 & b_2^2 & b_3^2 & \cdots & b_N^2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ b_1^5 & b_2^5 & b_3^5 & \cdots & b_N^5 \end{pmatrix}$$

- If we set 3 as the threshold, then we obtain the following result (after eliminating the 4'th and 5'th singular values.

$$\begin{pmatrix} a_1^1 & a_2^1 & a_3^1 \\ a_1^2 & a_2^2 & a_3^2 \\ \cdots & \cdots & \cdots \\ a_1^M & a_2^M & a_3^M \end{pmatrix} \begin{pmatrix} 20 & 0 & 0 \\ 0 & 16 & 0 \\ 0 & 0 & 12 \end{pmatrix} \begin{pmatrix} b_1^1 & b_2^1 & b_3^1 & \cdots & b_N^1 \\ b_1^2 & b_2^2 & b_3^2 & \cdots & b_N^2 \\ b_1^3 & b_2^3 & b_3^3 & \cdots & b_N^3 \end{pmatrix}$$

Analysis

- Usually, R is taken to be 200.
- The size of the original frequency table is $(M \times N)$ where M is the number of terms, and N is the number of documents. We may easily have $M = 1,000,000$ and $N = 10,000$, even for just a small document database such as that consisting of the University of Maryland's Computer Science technical reports.
- The size of the three matrices, *after* we have reduced the size of the singular matrix to, say 200, is:
  - The first matrix's size is $M \times R$. With the above numbers, this is $1000000 \times 200 = 200,000,000$.
  - The singular matrix's size is $200 \times 200 = 400,000$. (In fact, of these 400,000 entries, only 200 at most need to be stored, as all other entries are zero).
  - The last matrix's size is $R \times N$. With the above numbers, this is 200×10000.

  Adding up the above, we get a total of 202,000,200 entries in the tables after SVD's are applied. This is approximately 200 million.
- In contrast, $(M \times N)$ is close to 10,000-million: in other words, the SVD trick reduced the space utilized to about $\frac{1}{50}$'th of that required by the original frequency table.

LSI: Document Retrieval using SVDs

Two questions:

1. Given two documents d_1, d_2 in the archive, how "similar" are they?
2. Given a query string/document Q, what are the n documents in the archive that are "most relevant" for the query ?

- Suppose $\mathbf{x} = (x_1, \ldots, x_w)$ and $\mathbf{y} = (y_1, \ldots, y_w)$.
- *Dot product* of $\mathbf{x}$ and $\mathbf{y}$, denoted $\mathbf{x} \odot \mathbf{y}$ is given by

$$\mathbf{x} \odot \mathbf{y} = \sum_{i=1}^{w} x_i \times y_i.$$

- The similarity of these two documents w.r.t. the SVD representation, $\mathcal{TS}^\star \times \mathcal{D}^{\star T}$, of a frequency table is given by computing the dot product of the two *columns* in the matrix $\mathcal{D}^{\star T}$ associated with these two documents.

$$\sum_{z=1}^{R} \mathcal{D}^{\star T}[i, z] \times \mathcal{D}^{\star T}[j, z].$$

- When we are asked to find the top p matches for Q, we are trying to find p documents $d_{\alpha(1)}, \ldots, d_{\alpha(p)}$ such that:
    1. for all $1 \leq i \leq j \leq p$, the similarity between vec_Q and $d_{\alpha(i)}$ is greater than or equal to the similarity between vec_Q and $d_{\alpha(j)}$.

2. There is no other document d_z such that the similarity between d_z and vec_Q exceeds that of $d_{\alpha(p)}$.

- This can be done by using any generalized, high dimensional data structure that supports nearest neighbor searches.

Telescopic Vector (TV) Trees

- Access to point data in very large dimensional spaces should be highly efficient.
- A document d may be viewed as a vector $\vec{d}$ of length k, where the singular valued matrix, after decomposition, is of size $(k \times k)$.
- Thus, each document may be thought of as a point in a k-dimensional space.
- A *document database* may be thought of as a collection of such points, indexed appropriately.
- When a user u presents a query Q, s/he is in effect specifying, a vector $vec(Q)$ of length k. We must find the p documents in the database that are maximally relevant to Q.
- This boils down to attempting to find the k-nearest neighbors present in the document database, of the query Q.
- The TV-tree is a data structure that borrows from R-trees in this effort.
- The TV-tree attempts to *dynamically and flexibly* decide how to branch, based on the data that is being examined. If lots of vectors all agree on certain attributes (e.g. if lots of documents all have many common terms), then we must organize our index by branching on those terms (i.e. fields of the vectors) that *distinguish* between these vectors/documents.

Organization of a TV-Tree

- NumChild: This is the maximal number of children that any node in the TV-tree is allowed to have.
- α: α is a number, greater than 0 and less than k, called the *number of active dimensions*.
- $\mathsf{TV}(k, \mathsf{NumChild}, \alpha)$ denotes TV-tree used to store k-dimensional data with NumChild as the maximal number of children, and α as the number of active dimensions.
- Each node in a TV-tree represents a region. For this purpose, each node N in a TV-tree contains three fields:
  1. $N.Center$: This represents a point in k-dimensional space.
  2. $N.Radius$: This is a real number greater than 0.
  3. $N.ActiveDims$: This is a list of at most α dimensions. Each of these dimensions is a number between 1 and k. Thus, $N.ActiveDims$ is a subset of $\{1, \ldots, k\}$ of cardinality α or less,

Region associated with a node N

- Suppose $\mathbf{x}$ and $\mathbf{y}$ are points in k-dimensional space, and $ActiveDims$ is some set of active dimensions. The *active distance* between $\mathbf{x}$ and $\mathbf{y}$, denoted $\mathsf{act\_dist}(\mathbf{x}, \mathbf{y})$ is given by:

$$\mathsf{act\_dist}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i \in ActiveDims} \mathbf{x}_i^2 - \mathbf{y}_i^2}.$$

  Here, $\mathbf{x}_i$ and $\mathbf{y}_i$ denote the value of the i'th dimension of $\mathbf{x}$ and $\mathbf{y}$, respectively.

- EX: $k = 200$ and $\alpha = 5$ and the set $ActiveDims = \{1, 2, 3, 4, 5\}$. Suppose:

$$\begin{aligned} \mathbf{x} &= (10, 5, 11, 13, 7, x_6, x_7, \ldots, x_{200}). \\ \mathbf{y} &= (2, 4, 14, 8, 6, y_6, y_7, \ldots, y_{200}). \end{aligned}$$

  Then the active distance between $\mathbf{x}$ and $\mathbf{y}$ is given by:

$$\begin{aligned} \mathsf{act\_dist}(\mathbf{x}, \mathbf{y}) &= \sqrt{(10-2)^2 + (5-4)^2 + (11-14)^2 + (13-8)^2 + (7- \\ &= \sqrt{100} \\ &= 10. \end{aligned}$$

- Node N represents the region containing all points $\mathbf{x}$ such that the active distance (w.r.t. the active dimensions in $N.ActiveDims$) between $\mathbf{x}$ and $N.Center$ is less than or equal to $N.Radius$.

- EX: If we had a node N with its center at

$$N.Center \ = \ (10, 5, 11, 13, 7, 0, 0, 0, 0, \ldots, 0)$$

  and $N.ActiveDims = \{1, 2, 3, 4, 5\}$, then this node represents the region consisting of all points $\mathbf{x}$ such that:

$$\sqrt{(\mathbf{x}_1 - 10)^2 + (\mathbf{x}_2 - 5)^2 + (\mathbf{x} - 11)^2 + (\mathbf{x}_4 - 13)^2 + (\mathbf{x}_5 - 7)^2} \ \leq \ N.Ra$$

  We use the notation $Region(N)$ to denote the region represented by a node N in a TV-tree.

- N also contains an array, Child of NumChild pointers to other nodes of the same type.

Proprties of TV-Trees

- All data is stored at the leaf nodes;
- Each node in a TV-tree (except for the root and the leaves) must be at least half full, i.e. at least half the Child pointers must be non-NIL.
- If N is a node, and $N_1, \ldots, N_r$ are all its children, then

$$Region(N) = \bigcup_{i=1}^{r} Region(N_i).$$

Insertion into TV-trees

There are thre key steps.

1. **Branch Selection:** The first operation is called branch selection. When we insert a new vector into the TV-tree, and we are at node N (with children N_i, for $1 \leq i \leq \mathsf{NumChild}$), we need to determine which of these children to insert the key into.

2. **Splitting:** The second approach is what to do, when we are at a leaf node that is full and cannot accommodate the vector $\mathbf{v}$ we are inserting. This causes a split in that node.

3. **Telescoping:** Suppose a node N is split into subnodes N_1, N_2. In this case, it may well turn out that the vectors in $Region(N_1)$ all agree on not just the active dimensions of the parent N, but a few more as well. The addition of these extra dimensions is called *telescoping*. Telescoping may also involve the *removal* of some active dimensions, as we shall see later.

Example

- 5-dimensional space.
- TV$(5, 3, 2)$.
- Space is a sphere centered at $(0, 0, 0, 0, 0)$ with radius 50.
- Initially, tree is empty.
- Insert $(5, 3, 20, 1, 5)$. This is handled straightforwardly by the creation of a root node with
  1. $Root.Center = (0, 0, 0, 0, 0)$;
  2. $Root.Radius = 50$.
  3. In this case, the root is also a leaf, with a pointer to the information relevant to the point $\mathbf{v}_1 = (5, 3, 20, 1, 5)$.
  4. Suppose $Root.ActiveDims = \{2, 3\}$.

  See (a).
- Insert $\mathbf{v}_2 = (0, 0, 18, 42, 4)$.
- Insert $\mathbf{v}_3 = (0, 0, 19, 39, 6)$. At this stage, the root is "full".
- Insert $\mathbf{v}_4 = (9, 10, 2, 0, 16)$.
  1. We must *split* the root.
  2. Take the four vectors involved and "group" them together into two groups, say. v_1, v_4 and v_2, v_3. See figure (d).
  3. Insert $\mathbf{v}_5 = (18, 5, 27, 9, 9)$. Branch selection needed to determine how to branch. See Figure 2(a).

4. Insert $\mathbf{v}_6 = (0, 0, 29, 0, 3)$. Again, we must perform branch selection, and this time, we may choose to branch right, as shown in Figure 2(b).

Figure 1

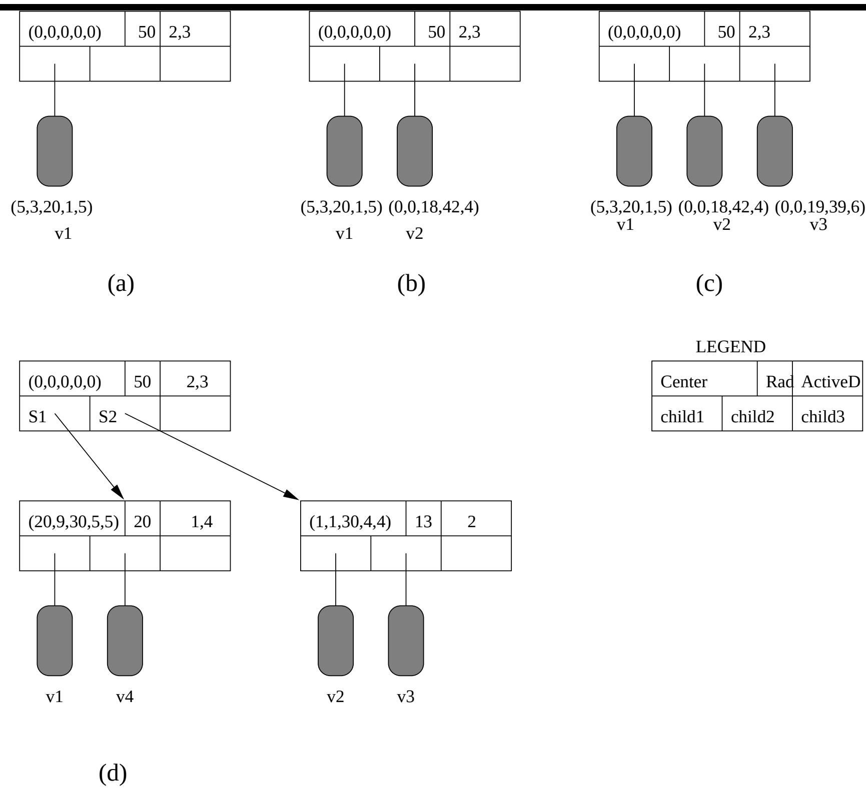
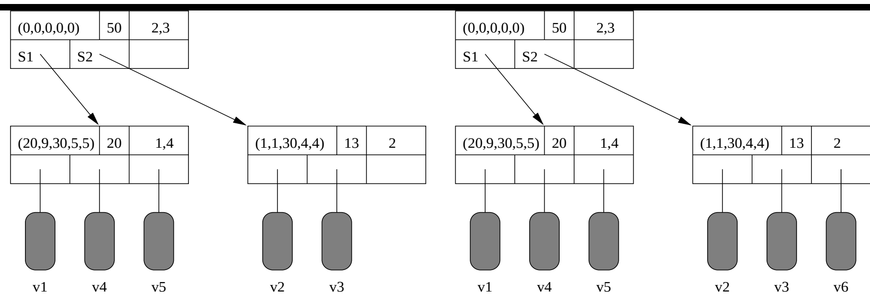

(0,0,0,0,0)
50
2,3
(5,3,20,1,5)
v1
(a)
(0,0,18,42,4)
v2
(b)
(0,0,19,39,6)
v3
(c)
S1
S2
(20,9,30,5,5)
20
1,4
(1,1,30,4,4)
13
2
v4
(d)
LEGEND
Center
Rad
ActiveD
child1
child2
child3

Figure 2


(0,0,0,0,0)
50
2,3
S1
S2
(20,9,30,5,5)
20
1,4
(1,1,30,4,4)
13
2
v1
v4
v5
v2
v3
(0,0,0,0,0)
50
2,3
S1
S2
(20,9,30,5,5)
20
1,4
(1,1,30,4,4)
13
2
v1
v4
v5
v2
v3
v6

Branch Selection

- Consider node N with $1 \leq j \leq \mathsf{NumChild}$ children, denoted $N_1, \ldots, N_{\mathsf{NumChild}}$.
- $exp_j(\mathbf{v})$ denotes the amount we must expand $N_j.Radius$ so that $\mathbf{v}$'s active distance from $N_j.Center$ falls within this radius.
- First, select all j's such that $exp_j(\mathbf{v})$ is minimized.
  EX: if we have nodes $N_1, \ldots, N_5$ with exp values $10, 40, 19, 10, 32$ respectively, the two candidates selected for possible insertion are N_1 and N_4. If a tie occurs, as in the above case, pick the node such that the distance from the center of that node to $\mathbf{v}$ is minimized.

Splitting

- When we attempt to insert a vector **v** into a leaf node N that is already full, then we need to split the node.
- Wemust create subnodes N_1, N_2, and each vector in node N must fall into one of the regions represented by these two subnodes.
- Split vectors in leaf N into two groups (G_1, H_1).
- We may be able to enclose all vectors in G_1 within a region with center c_1 and radius r_1 and all vectors in H_1 within a region with center c_2 and radius r_2.
- Many such splits are possible in general.
- Split (G_1, H_1) is *finer than* split (G_2, H_2) iff the sums of the radii, $(r_1 + c_1')$ is smaller than the sum of the radii $(r_2 + r_2')$.
- Still not enough to uniquely identify a "finest" split.
- If (G_1, H_1) and (G_2, H_2) are splits such that neither is finer than the other and no other split is finer than each of them, then we say that (G_1, H_1) is *more conservative* than (G_2, H_2) iff

$$r_1 + r_1' - \mathsf{act\_dist}(c_1, c_1') \ \leq \ r_2 + r_2' - \mathsf{act\_dist}(c_2, c_2').$$

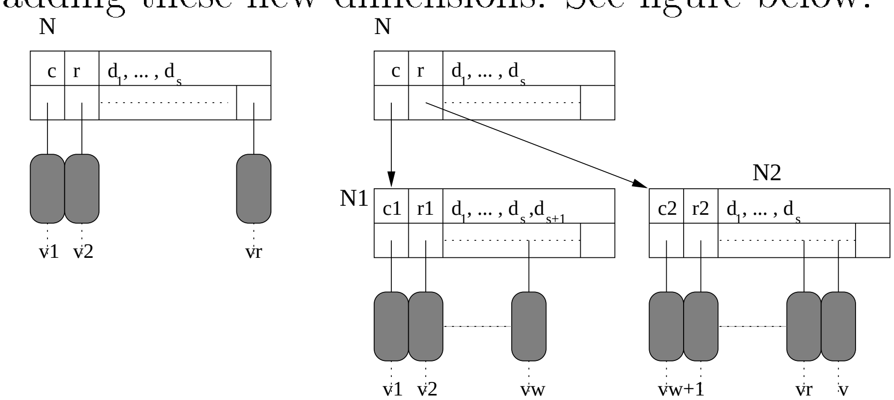
- Split (G, H) is the selected split iff:

1. there is no split (G', H') that is finer than (G, H) and

2. there is no split (G', H') that satisfies condition (1) (i.e. there exists no split (G^*, H^*) finer than (G', H')) and that is more conservative than (G, H).

Telescoping

- Suppose N is the node into which we are to insert a vector $\mathbf{v}$.
- Insertion of $\mathbf{v}$ may cause two types of changes to N: it may cause N to be *split* into two subnodes N_1, N_2, or it may "modify" the set of active dimensions of node N (e.g. if vector $\mathbf{v}$ does not agree, on the active dimensions, with other vectors stored at node N).
- When node N gets split into two sub-nodes N_1, N_2, the set of vectors at either node N_1 or node N_2 (but not both) must be a subset of the set of vectors at node N before the insertion.
- Suppose N_1 has this property.
- The vectors in N_1 may agree not only on the active dimensions of N, but on some other dimensions as well. In this case, we can *expand* the set of active dimensions of node N, by adding these new dimensions. See figure below.



- The other case when telescoping occurs is when a vector is added to a node, N, but no split occurs. If N originally contained the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_r$ and $\mathbf{v}$ is the vector being added, even though vectors $\mathbf{v}_1, \ldots, \mathbf{v}_r$ originally agreed on the active dimensions $d_1, \ldots, d_s$ of node N, they only agree now on a subset (for example, $d_2, \ldots, d_s$) and hence, the set of active dimensions of node N must be *contracted* to reflect this fact.

Searching in TV-Trees

Algorithm 2 *Search(T,***v***)*

if $Leaf(T)$ **then Return** $(T.Center = \mathbf{v})$; **Halt** }
else
{ **if** $\mathbf{v} \in Region(T)$ **then**
Return $\bigvee_{i=1}^{\mathsf{NumChild}} Search(T.\mathsf{Child}[i], \mathbf{v})$
}
end

Nearest Neighbor Retrievals in TV-Trees

$$min(N, \mathbf{v}) = \begin{cases} 0 & \text{if } \mathbf{v} \in Region(N) \\ \textsf{act\_dist}(\mathbf{v}, N.Center) - N.Radius & \text{otherwise} \end{cases}$$
$$max(N, \mathbf{v}) = \textsf{act\_dist}(\mathbf{v}, N.Center) + N.Radius.$$

- Maintain an array SOL of length p,i.e. with indices running 1 through p.
- The algorithm $NNSearch$ uses a routine called $Insert$ that takes as input, a vector vec, and an array SOL maintained in *non-descending order* of active distance from vec.
- $Insert$ returns as output, the array SOL with vec inserted in it, at the right place, and with the p'th element of SOL eliminated.

Nearest Neighbor Retrievals in TV-Trees (Contd.)

Algorithm 3 *NNSearch(T,v,p)*

for $i = 1$ **to** p **do** $SOL[i] = \infty$;
NNSearch1(T,v,p);

end (⋆ *end of program NNSearch* ⋆)

procedure *NNSearch1(T,v,p);*

if $Leaf(T)$ & act_dist$(T.val, \mathbf{v}) < SOL[p]$ **then**
  Insert T.val into SOL;
else
  {
   if $Leaf(T)$ **then** $r = 0$;
   else { *Let* $N_1, \ldots, N_r$ *be the children of* T;
   Order the N_i*'s in ascending order w.r.t.* $min(N_i, \mathbf{v})$;
   Let $N_{\eta(1)}, \ldots, N_{\eta(r)}$ *be the resulting order;*
   };
   $done = false$; $i = 1$;
   while $((i \leq r) \wedge \neg done)$ **do**
    {
     $NNSearch(N_{\eta(i)}, \mathbf{v}, p)$;
     if $SOL[p] < min(N_{\eta(i+1)}, \mathbf{v})$ **then**
      $done = true$;
     $i = i + 1$;
    }; (⋆ *end of* **while** ⋆)
  } (⋆ *end of* **else** ⋆)
Return *SOL;*
end proc (⋆ *end of subroutine NNSearch1* ⋆)

Other Retrieval Techniques: Inverted Indices

- A document record contains two fields – a `doc_id` and a `postings_list`.
- Postings list is a list of terms (or pointers to terms) that occur in the document. Sorted using a suitable relevance measure.
- A `term record` consists of two similar fields: a `term` field (string), and a `postings_list`.
- Two hash tables are maintained: a DocTable and a TermTable. The DocTable is constructed by hashing on the `doc_id` key, while the TermTable is obtained by hashing on the `term` key.
- To find all documents associated with a term, we merely return the postings list.
- Used in many commercial systems such as MEDLARS and DIALOG.

Other Retrieval Techniques: Signature Files

- Aassociate a *signature* s_d, with each document d.
- A signature is a representation of an ordered list of terms that describe the document.
- The list of terms from which s_d is derived is obtained after performing a frequency analysis, stemming, and usage of stop lists.
- If signature list consists of the ordered list of words $w_1, w_2, \ldots, w_r$.
- This means that word w_1 is most important when describing the document, word w_2 is second most important,and so forth.
- Signature of the document d is a *bit-representation* of this list, usually obtained by encoding the list after using hashing, and then superimposing a coding scheme.

Video Databases

Introduction

Retrieval requests may take one of two forms.

- **Retrieving a Specified Video:** User specifies the video he wants to see, e.g. "Show me *The Sound of Music*".
- **Identifying and Retrieving Video Segments:** User might express a query such as *Find all videos in which John Wayne appears with a gun.* This query requires that we:
  - identify the movies in which John Wayne appears with a gun and
  - identify the segments within those movies in which John Wayne appears with a gun.
- Once we can organize the content of a single video, we can organize the content of a set of videos.

Organizing Content of a Single Video

We must ask ourselves the following questions:

1. *Which* aspects of the video are likely to be of interest to the users who access the video archive ?
2. *How* can these aspects of the video be stored efficiently, so as to minimize the time needed to answer user queries ?
3. What should *Query Languages* for video data look like and how should the relational model of data be extended to handle video information ?
4. Can the *Content Extraction* process be automated, and if so, how can the reliability of such content extraction techniques be taken into account when processing queries?

Video Content: Which Aspects of a Video To Store?

Example: An 8-hour, one day lecture of a short course given by a professor on the topic *Multimedia Databases*. In this case, the video contains a set of "items of interest." These items of interest could include:

1. *People* such as the professor, any guest lecturer (or lecturers) who speak at selected times in the course, and any students who might ask questions and/or distinguish themselves in other ways; For instance, `Prof. Felix` might be one such person, while `Erica` might be a student.

2. *Activities* that occur in the class such as *lecturing* (on a particular topic, by a particular individual), or *questioning* (by a particular student), or *answering* a question posed by a particular student. Other activities could involve general group discussions, and/or coffee breaks.

   In addition, actvities have attributes, e.g. *lecturing(quadtrees, Prof. Felix)* indicates an activity involving Prof. Felix lecturing on quadtrees, and *questioning(Erica, Prof. Felix)* indicating that Prof. Felix was questioned by Erica.

Movie Example

- Consider the movie, *Sound of Music*.
- Items of interest include:
  1. *People* such as Maria, Count Von Trapp, and others;
  2. *Inanimate objects* such as the piano in Count Von Trapp's house;
  3. *Animate objects* such as the ducks and birds in the pond;
  4. *Activities* such as singing and dancing, with their associated list of attributes. For example, the activity *singing* may have two attributes:
     (a) `Singer` specifying which person is singing and
     (b) `Song` specifying the name of the song and
- Certain common characteristics occur. Given any frame f in the video, the frame f has a set of associated objects and associated activities.
- Objects/activities have certain properties, and these properties may vary from one frame to another.
- **Creating a video database means we should be able to index all these associations.**

Properties

- **Property:** Consists of a pair ($\mathsf{pname}, \mathsf{Values}$) where:
  - pname is the *Name* of the property,
  - Values is a set.
- **Property Instance:** An expression of the form $\mathsf{pname} = v$ where $v \in \mathsf{Values}$.
- Example properties:
  1. $(height, \mathbf{R}^+)$ consists of the "height" property with real-values;
  2. $(primarycolors, \{red, green, blue\})$ consists of a property called *primarycolors* with values red,green, blue.

Object Scheme

- **Object Scheme:** A pair (fd, fi) where:

  1. fd is a set of *frame-dependent* properties;
  2. fi is a set of *frame-independent* properties.
  3. fi and fd are disjoint sets.

  fd and fi are disjoint.

- If (pname, Values) is a property in fd, then this means that the property named pname may assume different instances, depending upon the video frame being considered. E.g. the property *shirtcolor* varies from frame to frame.

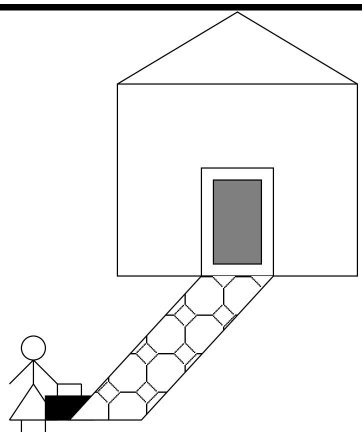
- **Object Instance:** An *Object Instance* is a triple (oid, os, ip) where:

  1. oid is a string called the object-id and
  2. os = (fd, fi) is an object structure and
  3. ip is a set of statements such that:
     (a) for each property (pname, Values) in fi, ip contains *at most* one property instance of (pname, Values) and;
     (b) for each property (pname, Values) in fd, and each frame f of the video, ip contains *at most* one property instance of (pname, Values) – this property instance is denoted by the expression pname $= v$ IN f.

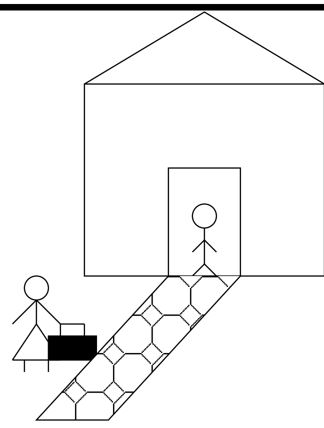
Example

- Surveillance video of 5 frames.
- Show surveillance video of the house of Denis Dopeman.
- Frame 1: We see Jane Shady at the path leading to Mr. Dopeman's door. She is carrying a briefcase.
- Frame 2: She is halfway on the path to the door. Door opens. Mr. Dopeman appears at the door.
- Frame 3: Jane Shady and Denis Dopeman are ext to each other at the door; Jane Shady is still carrying the briefcase.
- Frame 4: Jane Shady is walking back, and Denis Dopeman has the brief case.
- Frame 5: Jane Shady is at the beginning of the path to Denis Dopeman's door. Door is shut.

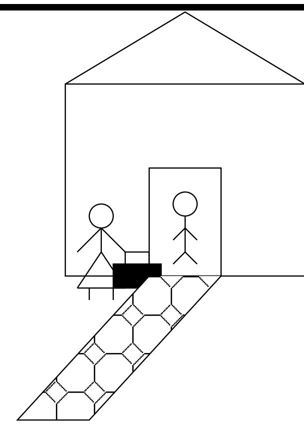
Example, contd.



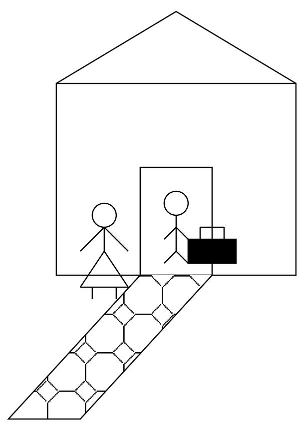
Frame 1



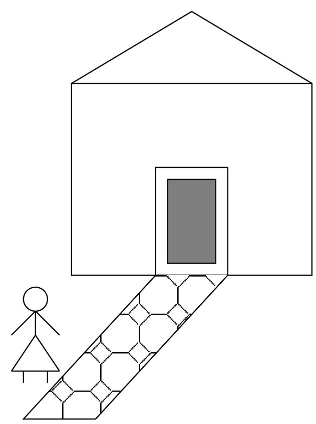
Frame 2



Frame 3



Frame 4



Frame 5

Frame-dependent properties

Frame	Objects	Frame-dependent properties
1	Jane Shady	`has(briefcase), at(path_front)`
	dopeman_house	`door(closed)`
	Briefcase	
2	Jane Shady	`has(briefcase), at(path_middle)`
	Denis Dopeman	`at(door)`
	dopeman_house	`door(open)`
	Briefcase	
3	Jane Shady	`has(briefcase), at(door)`
	Denis Dopeman	`at(door)`
	dopeman_house	`door(open)`
	Briefcase	
4	Jane Shady	`at(door)`
	Denis Dopeman	`has(briefcase),at(door)`
	dopeman_house	`door(open)`
	Briefcase	
5	Jane Shady	`at(path_middle)`
	dopeman_house	`door(closed)`
	Briefcase	

Frame independent properties

Object	Frame-independent property	Value
Jane Shady	age	35
	height	170 (cms)
dopeman_house	address	6717 Pimmit Drive Falls Church, VA 22047.
	type	brick
	color	brown
Denis Dopeman	age	56
	height	186
briefcase	color	black
	length	40 (cms)
	width	31 (cms)

Activity Schema

- An *Activity Scheme*, ACT_SCH, is a finite set of properties such that if $(\mathsf{pname}, \mathsf{Values}_1)$ and $(\mathsf{pname}, \mathsf{Values}_2)$ are both in ACT_SCH, then $\mathsf{Values}_1 = \mathsf{Values}_2$.
- **Example:** Consider the activity `ExchangeObject` such as the exchange of objects between Jane Shady and Denis Dopeman. his activity has the three-pair scheme consisting of the pairs:

  1. `(Giver,Person)`: This pair specifies that the activity `ExchangeObject` has a property called `Giver` specifying who is transferring the object in question. This says that the property `Giver` is of type `Person`. `Person` is the set of all persons.
  2. `(Receiver, Person)` This pair specifies that the activity `ExchangeObject` has a property called `Receiver` specifying who is receiving the object in question.
  3. `(Item, Thing)`: This pair specifies the item being exchanged. `Thing` is the set of all "exchange-able" items.

  Thus, the exchange of the briefcase that occurred between Jane Shady and Denis Dopeman can be captured as an activity scheme with `Giver` = Jane Shady, `Receiver` = Denis Dopeman, and `Item` = briefcase.

Activity/Event

- An *Activity* is a pair:
  1. AcID: the "name" of the activity of scheme ACT_SCH and
  2. for each pair (pname, Values) $\in$ ACT_SCH, an equation of the form pname $= v$ where $v \in$ Values.
- Any activity has an associated activity scheme, and each property of the activity has an associated value from its set of possible values.
- **Example:**
  1. The activity `Lecturing` may have the scheme

     $$\{(\texttt{Lecturer}, \texttt{Person}), (\texttt{Topic}, \texttt{String})\}$$

     and may contain the equations:

     $$\begin{aligned}\texttt{Lecturer} &= Prof.\ Felix.\\ \texttt{Topic} &= Video\ Databases.\end{aligned}$$

  2. Likewise, the activity `Questioning` may have the scheme

     $$\{(\texttt{Questioner}, \texttt{Person}), (\texttt{Questionee}, \texttt{Person}),$$
     $$(\texttt{Question}, \texttt{String}), (\texttt{Answer}, \texttt{String})\}$$

     and may contain the equations:

     $$\begin{aligned}\texttt{Questioner} &= Erica.\\ \texttt{Questionee} &= Prof.\ Felix.\\ \texttt{Question} &= \text{How many children does a quadtree node have?}\\ \texttt{Answer} &= \text{At most 4.}\end{aligned}$$

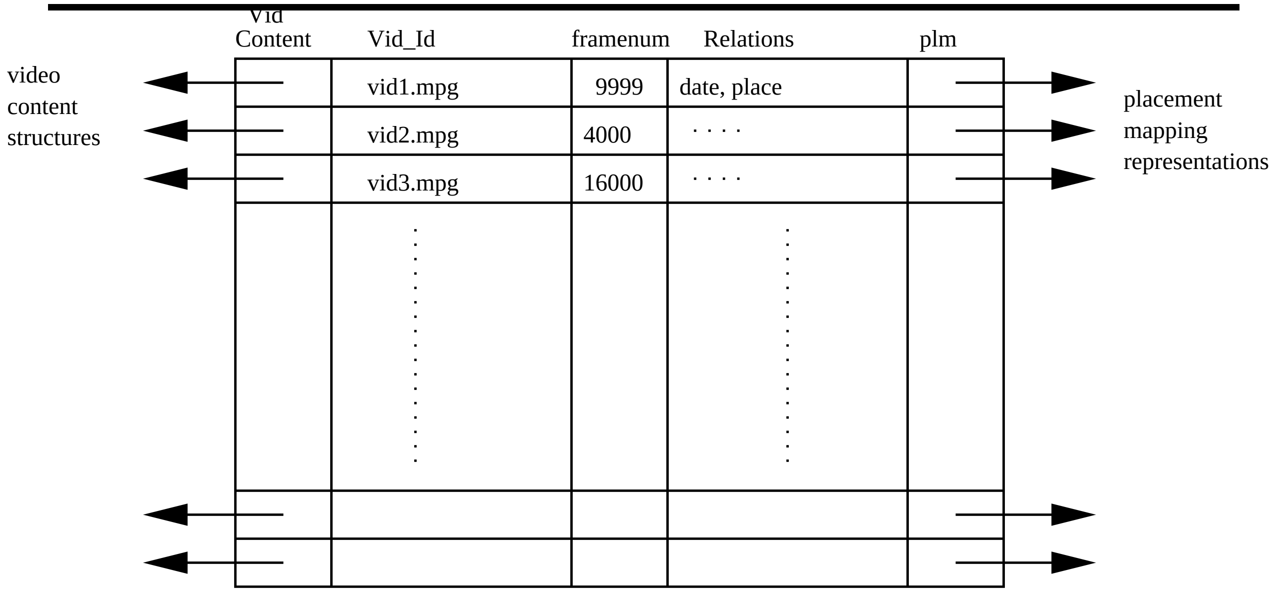
Video Content

- Suppose v is a video.
- Let $\mathsf{framenum}(v)$ specify the total number of frames of video v.
- The *content of* v consists of a triple $(\mathsf{OBJ}, \mathsf{AC}, \lambda)$ where:
  1. $\mathsf{OBJ} = \{\mathsf{oid}_1, \ldots, \mathsf{oid}_n\}$ is a finite set of object instances;
  2. $\mathsf{AC} = \{\mathsf{AcID}_1, \ldots, \mathsf{AcID}_k\}$ is a finite set of activities/events and
  3. λ is a map from $\{1, \ldots, \mathsf{framenum}(v)\}$ to $2^{\mathsf{OBJ} \cup \mathsf{AC}}$.
- Intuitively,
  1. OBJ represents the set of objects of interest in the video and
  2. AC represents the set of activities of interest in the video and
  3. λ tells us which objects and which activities are associated with any given frame f of the video.
- Though this definition assumes that λ will be specified on a frame by frame basis, this is not required, as we will see later.

Video Library

- A *video library*, VidLib, consists of a finite set of 5-tuples (Vid_Id, VidContent , framenum, plm, $\Re$) where:
  1. Vid_Id is the *Name* of the video and
  2. VidContent is the *Content* of the video and
  3. framenum is the number of frames in the video and
  4. plm is a *placement mapping* that specifies the address of different parts of the video.
  5. $\Re$ is a set of relations about videos "as a whole".

Organization of a simple video library



Query Languages for Video Data

Querying video involves the following types of queries.

- **Segment Retrievals:** Find all segments, from one or more videos in the library, that satisfy a given condition.
- **Object Retrievals:** Given a video v and a segment $[s, e]$ (start frame through end frame) of the video, find all objects that occurred in:
  - all frames between s and e (inclusive),
  - some frame between s and e (inclusive).
- **Activity Retrievals:** Given a video v and a segment $[s, e]$ (start frame through end frame) of the video, find all activities occurred in:
  - all frames between s and e (inclusive),
  - some frame between s and e (inclusive).
- **Property-based Retrievals:** Find all videos, and video segments in which objects/activities with certain properties occur.

Video Functions

- `FindVideoWithObject(o)`: Given the name of a data object o, this function returns as output, a set of triples of the form:

$$(\texttt{VideoId}, \texttt{Startframe}, \texttt{EndFrame})$$

  such that if (v, s, e) is a triple returned in the output, then video v's segment starting at frame s and ending at frame e has the object o in all frames between and including s, e.

- `FindVideoWithActivity(a)`: This does exactly the same as above, except that it returns all triples (v, s, e) such that video v's segment starting at frame s and ending at frame e has the activity a in it. For each property p, the notation $a.p$ specifies the value of that property.

- `FindVideoWithActivityandProp(a,p,z)`: This does exactly the same as above, except that it returns all triples (v, s, e) such that video v's segment starting at frame s and ending at frame e has the activity a in it with z as the value of property p.

- `FindVideoWithObjectandProp(o,p,z)`: This does exactly the same as above, except that it returns all triples (v, s, e) such that video v's segment starting at frame s and ending at frame e has the object o in it with z as the value of property p.

- `FindObjectsInVideo(v,s,e)`: Given the name of a video, and a start and end frame, this returns all objects that appear in all segments of the video between s and e (inclusive).
- `FindActivitiesInVideo(v,s,e)`: Identical to the above, except it applies to activities, not objects.
- `FindActivitiesAndPropsinVideo(v,s,e)`: Given the name of a video, a start and end frame, this returns a set of records of the form

$$\texttt{activityname} : \texttt{prop1} = \texttt{entity1}; \texttt{prop2} = \texttt{entity2}; \cdots$$

$$\texttt{propk} = \texttt{entityk}$$

  comprising all activities, and their associated roles, that occur in all times between s and e of video v.
- `FindObjectsAndPropsinVideo(v,s,e)`: Identical to the above, except that it applies to objects, not to activities.

Video Query Languages

- Standard SQL query has the form:

SELECT	`field1,...,fieldn`
FROM	`relation1` ⟨R1⟩,
	`relation2` ⟨R2⟩, . . .,
	`relationk` ⟨Rk⟩
WHERE	`Condition`.

- Expand this so that:

1. The SELECT statement may contain entries of the form

   $$\mathsf{Vid\_Id} : [\mathsf{s}, \mathsf{e}]$$

   denoting the selection of a video with id, Vid_Id, and with the relevant segment comprised of frames between s and e inclusive.

2. The FROM statement may contain entries of the form:

   $$\texttt{video}\langle source \rangle \langle V \rangle$$

   which says that V is a variable ranging over videos from the source named.

3. The WHERE condition allows statements of the form

   $$term\ \underline{\mathsf{IN}}\ func\_call$$

   where:

(a) *term* is either a variable or an object or an activity, or a property value and

(b) *func_call* is any of the eight video functions listed above.

Examples

- *"Find all videos and their relevant segments from video library* VidLib$_1$ *that contain Denis Dopeman."*

```
SELECT    vid:[s,e]
FROM      video:VidLib1
WHERE     (vid, s, e) IN
          FindVideoWithObject(Denis Dopeman).
```

- *"Find all videos and their relevant segments from video library* VidLib$_1$ *that contain Denis Dopeman and Jane Shady."*

```
SELECT    vid:[s,e]
FROM      video:VidLib1
WHERE     (vid, s, e) IN
          FindVideoWithObject(Denis Dopeman) AND
          (vid, s, e) IN
          FindVideoWithObject(Jane Shady).
```

Examples, Continued

- *"Find all videos and their relevant segments from video library* VidLib_1 *that contain Denis Dopeman and Jane Shady exchanging a briefcase."*

```
SELECT    vid:[s,e]
FROM      video:VidLib1
WHERE     (vid, s, e) IN
          FindVideoWithObject(Denis Dopeman)
          AND
          (vid, s, e) IN
          FindVideoWithObject(Jane Shady)
          AND
          (vid, s, e) IN
          FindVideoWithActivityandProp
                  (ExchangeObject,Item,Briefcase)
          AND
          (vid, s, e) IN
          FindVideoWithActivityandProp
                  (ExchangeObject,Giver,Jane Shady)
          AND
          (vid, s, e) IN
          FindVideoWithActivityandProp
                  (ExchangeObject, Receiver, Denis Do
```

Indexing Video Content

- Now that we have defined content, we need to index it.
- We have 8 types of video retrieval functions. Indexing must support efficient execution of these 8 function types.
- It is impossible to store video content on a frame by frame basis due to the fact that a single 90 minute video contains close to ten million frames.
- We need Compact Representations to store video content.
- Two such data structures:
  - Frame Segment Tree
  - R-Segment Tree

Frame Segment Trees

- **Frame-sequence** is a pair $[i, j)$ where $1 \leq i, \leq j \leq n$. $[i, j)$ represents the set of all frames between i (inclusive) and j (non-inclusive), i.e.

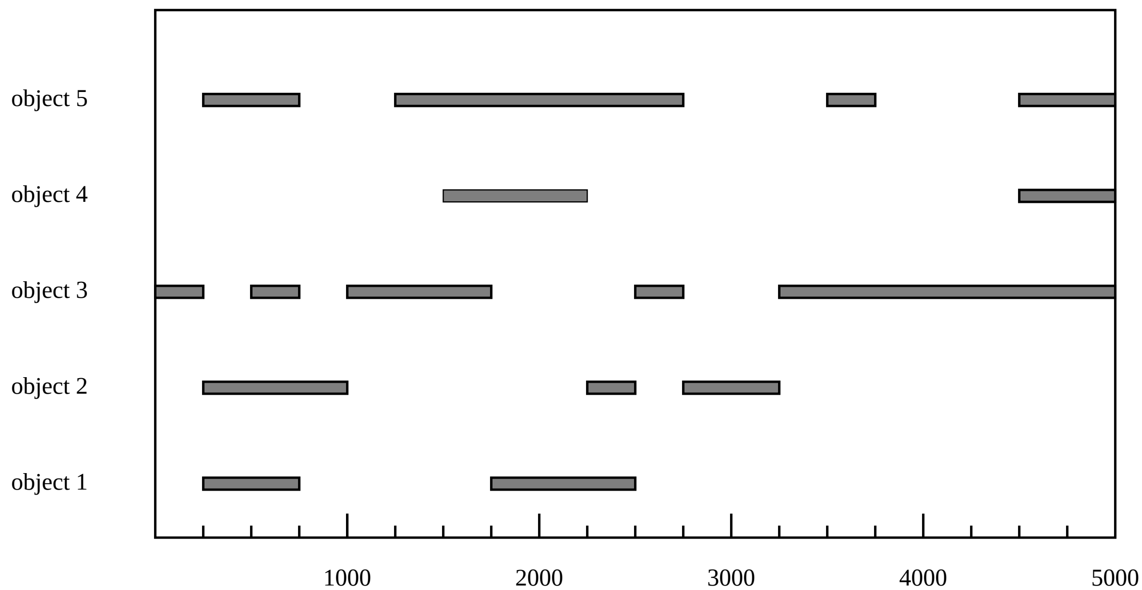
$$[i, j) = \{k \mid i \leq k < j\}.$$

- EX: $[6, 12)$ denotes the set of frames $\{6, 7, 8, 9, 10, 11\}$.
- **Frame-sequence Ordering:** $[i_1, j_1) \sqsubseteq [i_2, j_2)$ iff $i_1 < j_1 \leq i_2 < j_2$.
- $[i_1, j_1) \sqsubseteq [i_2, j_2)$ means that the sequence of frames denoted by $[i_1, j_1)$ precedes the sequence of frames denoted by $[i_2, j_2)$.
- EX: Cconsider frame-sequences $fs_1 = [10, 15)$, $fs_2 = [8, 10)$ and $fs_3 = [11, 13)$.
  - $fs_2 \sqsubseteq fs_1$
  - $fs_2 \sqsubseteq fs_3$
  - $fs_1 \not\sqsubseteq fs_3$.
- **Well-Ordered Set of Frame-sequences:** A set, X, of frame-sequences is said to be *well-ordered* iff:
  1. X is finite, i.e. $X = \{[i_1, j_1), \ldots, [i_r, j_r)\}$ for some integer r, and
  2. $[i_1, j_1) \sqsubseteq [i_2, j_2) \sqsubseteq \ldots \sqsubseteq [i_r, j_r)$.
- EX: $X = \{[1, 4), [9, 13), [33, 90)\}$ is a well-ordered set of frame-sequences because $[1, 4) \sqsubseteq [9, 13) \sqsubseteq [33, 90)$.

Frame Segment Trees, Continued

- **Solid Set of Frame-sequences:** A set, X, of frame-sequences is said to be *solid* iff
  1. X is well-ordered, and
  2. there is no pair of frame-sequences in X of the form $[i_1, i_2)$ and $[i_2, i_3)$.
- Take $X = \{[1,5), [5,7), [9,11)\}$.
- X is not solid. Why?
- Take $Y = \{[1,7), [9,11)\}$. This is solid.
- **Segment Association Map:** Suppose $(\mathsf{OBJ}, \mathsf{AC}, \lambda)$ represents the content of a video v. A *Segment Association Map* σ_v associated with video v is the map defined as follows:
  1. σ_v's domain is $\mathsf{OBJ} \cup \mathsf{AC}$ and
  2. σ_v returns, for each $x \in \mathsf{OBJ} \cup \mathsf{AC}$, a *solid* set of frame-sequence, denoted $\sigma_v(x)$ such that:
     (a) if $[s, e) \in \sigma_v(x)$, then for all $s \leq f < e$, it is the case that $x \in \lambda(f)$ and
     (b) for all frames f and all $x \in \mathsf{OBJ} \cup \mathsf{AC}$, if $x \in \lambda(f)$, then there exists a frame-sequence $[s, e) \in \sigma_v(x)$ such that $f \in [s, e)$.

An example of a video's content



Example, continued

- 5000 frames in example.
- The table below shows how many frames each object appears in:

Object	Number of frames
object1	1250
object2	1500
object3	3250
object4	1000
object5	2750

- To explicitly represent the mapping λ associated with the content of this video, we would need to have a total of 9750 tuples.
- Instead, represent information with 16 tuples as shown below.

Segment Table:

Object	Segment
object1	250–750
object1	1750–2500
object2	250–1000
object2	2250–2500
object2	2750–3250
object3	0–250
object3	500–750
object3	1000–1750
object3	2500–2750
object3	3250–5000
object4	1500–2250
object4	4500–5000
object5	250–750
object5	1250–2750
object5	3500–3750
object5	4500–5000

Frame-segment tree structure

- Suppose there are n objects $o_1, \ldots, o_n$ in our video v and m activities $a_1, \ldots, a_m$.
- Then we have a total of:

$$\sum_{i=1}^{n} (card(\sigma_v(o_i))) + \sum_{j=1}^{m} (card(\sigma_v(a_j)))$$

  entries in the table *just for one single video.*

- FS-trees use the following components:
  - `OBJECTARRAY:` specifies, for each object, an *ordered linked list* of pointers to nodes in the frame segment tree specifying which segments the object appears in.
  - `ACTIVITYARRAY:` specifies, for each activity, an *ordered linked list* of pointers to nodes in the frame segment tree specifying which segments the activity occurs in.
  - The FS-tree is now constructed from the segment table.

Frame-segment trees, continued

- **Step 1:** Let $[s_1, e_1), \ldots, [s_w, e_w)$ be all the intervals in the "Segment" column of the segment table. Let

  $$q_1, \ldots, q_z$$

  be an enumeration, in ascending order, of all members of $\{s_i, e_i \mid 1 \leq i \leq w\}$ with duplicates eliminated.

  If z is not an exponent of 2, then do as follows: let r be the smallest integer such that $z < 2^r$ and $2^r > \mathsf{framenum}(v)$. Add new elements $q_{z+1}, \ldots, q_{2^r}$ such that $q_{2^r} = \mathsf{framenum}(v) + 1$ and $q_{z+j} = q_z + j$ (for $z + j < 2^r$).

  By virtue of the above argument, we may proceed under the assumption that z is an exponent of 2, i.e. $z = 2^r$ for some r.

- **Step 2:** The frame-segment tree is a binary tree constructed as follows.

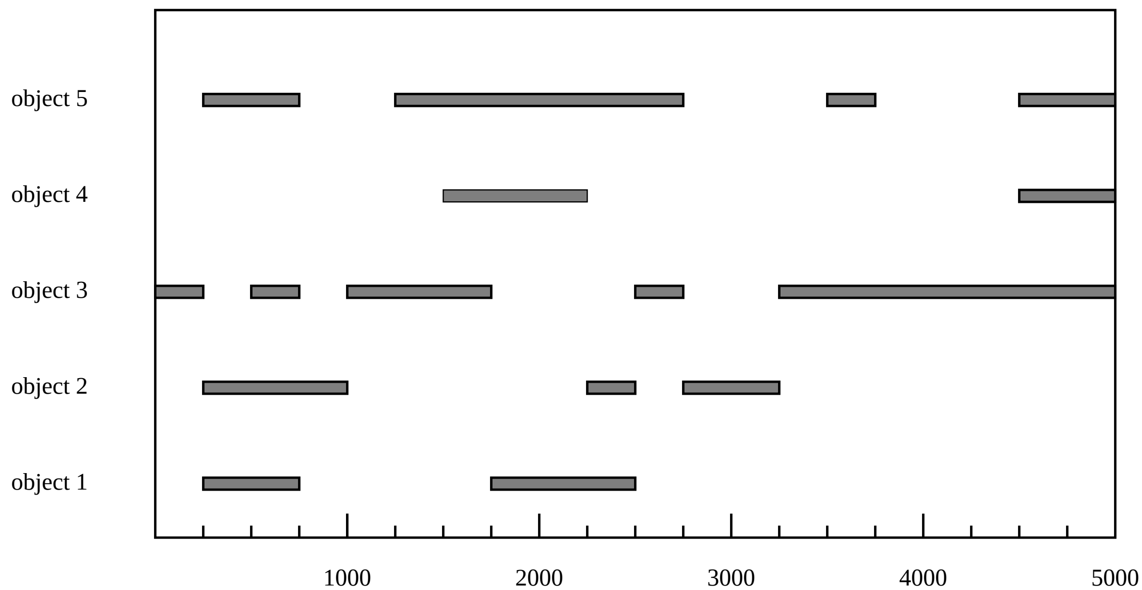
  1. Each node in the frame segment tree represents a *frame-sequence* $[x, y)$ starting at frame x and including all frames up to, but not including, frame y.
  2. Every leaf is at level r. The leftmost leaf denotes the interval $[z_1, z_2)$, the second from left-most represents the interval $[z_2, z_3)$, the third from left-most represents the interval $[z_3, z_4)$ and so on. If N is a node with two children

representing the intervals $[p_1, p_2), [p_2, p_3))$, then N represents the interval $[p_1, p_3)$. Thus, the root of the segment tree represents the interval $[q_1, q_z)$ if q_z is an exponent of 2; otherwise it represents the interval $[q_1, \infty)$.

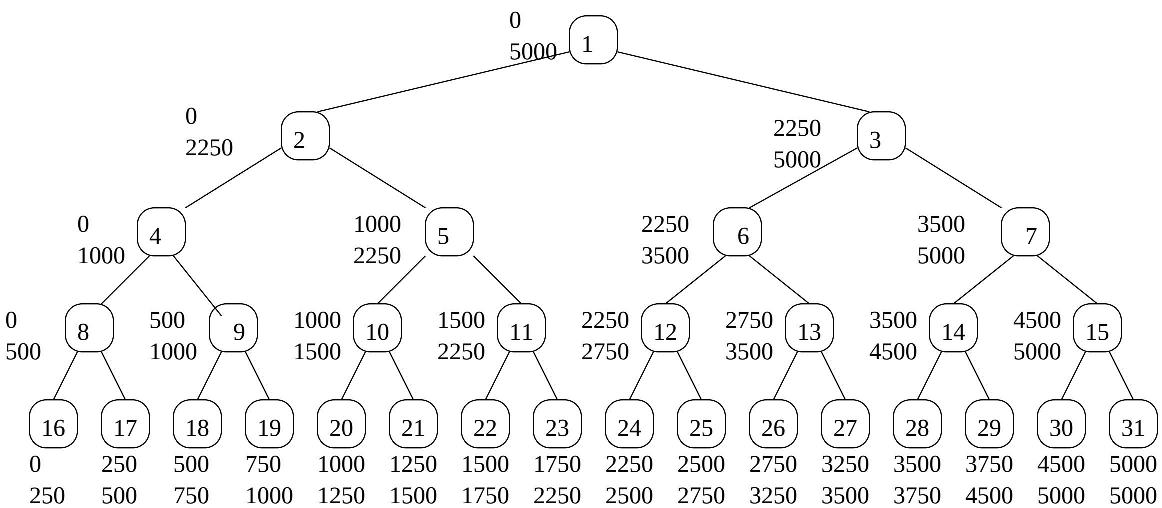
3. The number inside each node may be viewed as the address pf that node.

4. The set of number placed next to a node denotes the id-numbers of video objects and activities that appear in the entire frame-sequence associated with that node. Thus, for example, if a node N represents the frame sequence $[i, j)$ and object o occurs in all frames in $[i, j)$, then object o labels node N (unless object o labels an ancestor of node N in the tree).

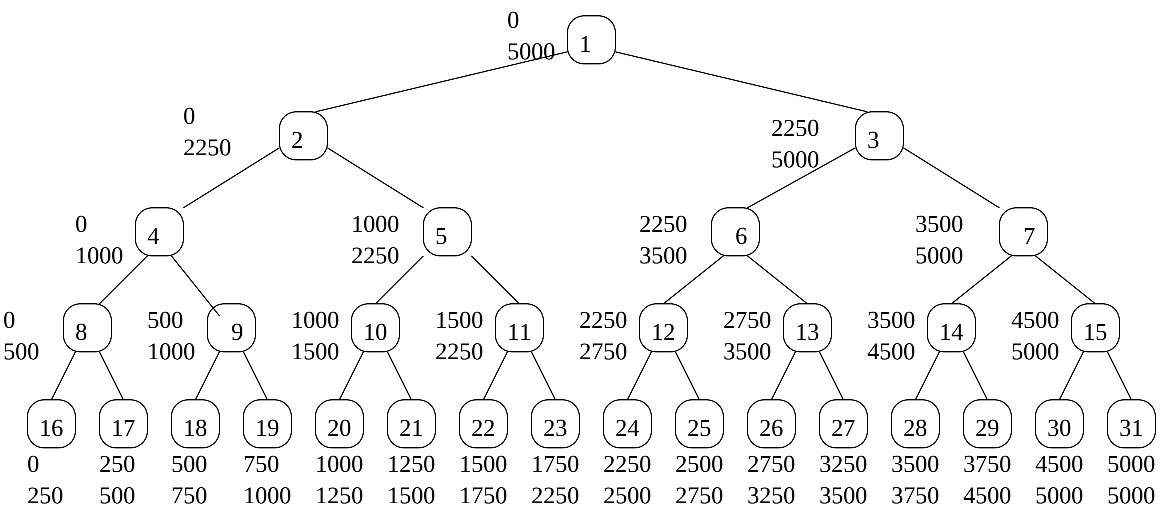
Example


object 5
object 4
object 3
object 2
object 1
1000
2000
3000
4000
5000

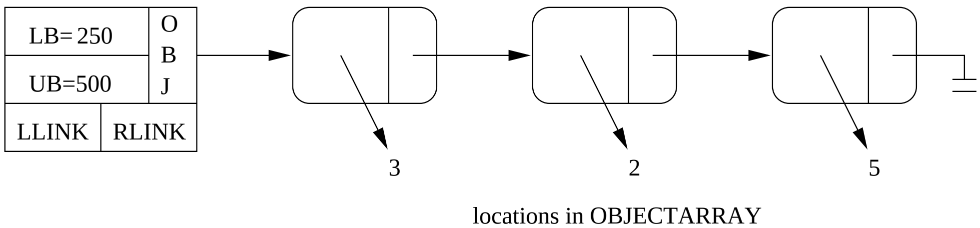
Example, continued



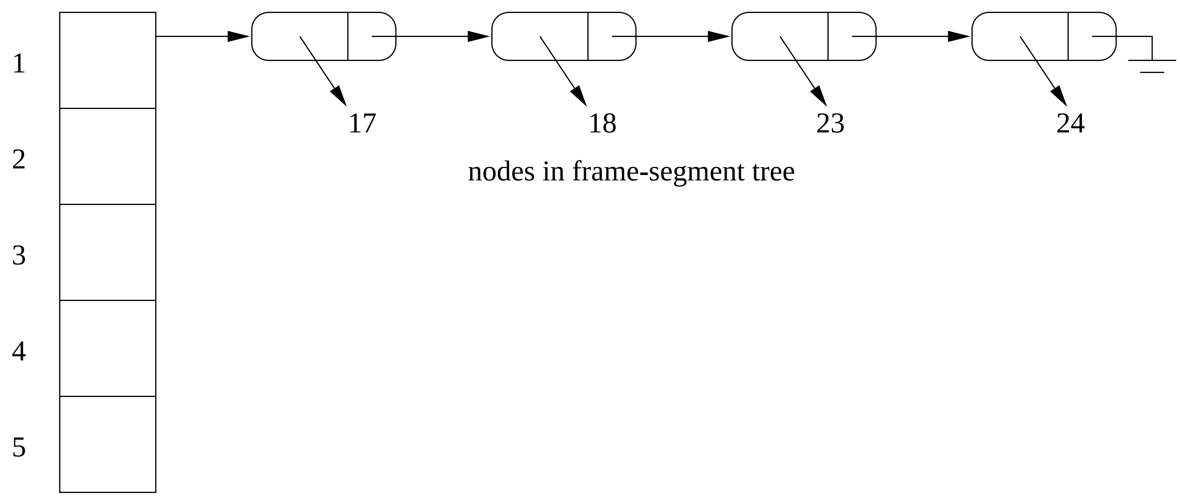
Example, continued



Example, continued



Example, continued



Video Library

- Suppose our video library, VidLib, contains videos $v_1, \ldots, v_n$.
- Create a table called `INTOBJECTARRAY` having the scheme `(VID_ID,OBJ,PTR)`.
- Tuple (v, o, ptr) is in `INTOBJECTARRAY` iff the pair (o, ptr) is in the `OBJECTARRAY` associated with video v.
- Create a table called `INTACTIVITYARRAY` having the scheme `(VID_ID,ACT,PTR)`.
- Tuple (v, a, ptr) is in `INTOBJECTARRAY` iff the pair (a, ptr) is in the `ACTIVITYARRAY` associated with video v.
- For each v_i, a frame segment tree, fst(v_i) is associated with video v_i.
- Only difference from before is that pointers from the frame segment tree point to locations in `INTOBJECTARRAY` and `INTACTIVITYARRAY` rather than to `OBJECTARRAY` and `ACTIVITYARRAY` as described earlier.

Implementing Video Operations

- `FindVideoWithObject(o)`:

  SELECT VIDEO_ID
  FROM `INTOBJECTARRAY`
  WHERE `OBJ` = o.

- `FindVideoWithActivityandProp(a,p,z)`:

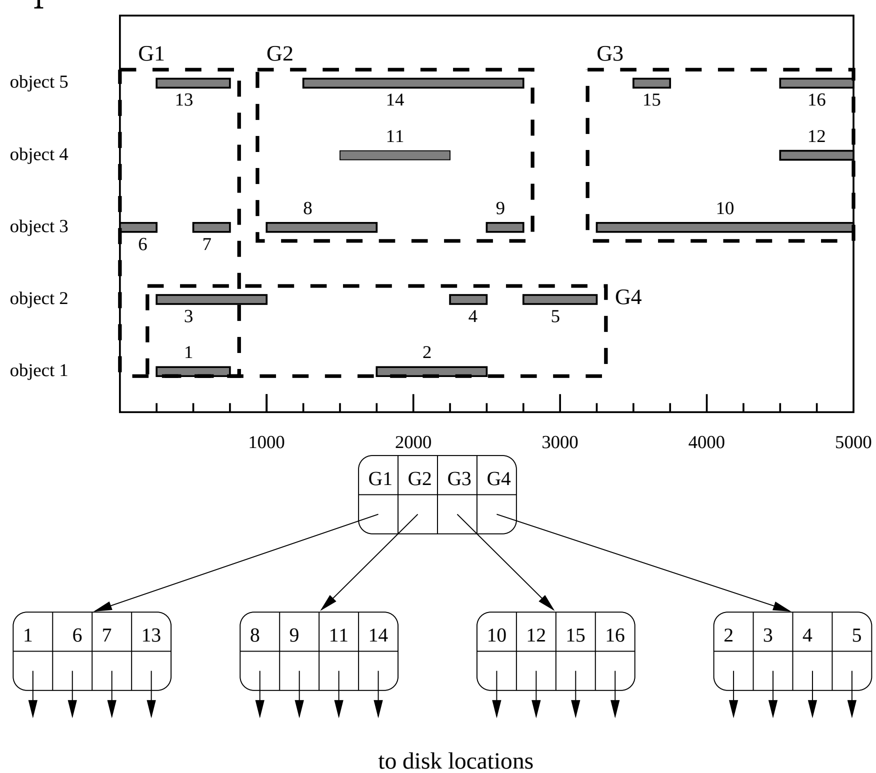
  SELECT VIDEO_ID
  FROM `INTOBJECTARRAY` t
  WHERE `OBJ` = o AND t.p = z.

- `FindObjectsInVideo(v,s,e)`:

```
Algorithm 4 FindObjectsInVideo(R,s,e)
  S = NIL; (* no objects found so far *)
  if R =NIL then { Return S; Halt }
  else
    { if [R.LB, R.UB) ⊆ [s, e) then S = append(S,preorder(R))
      else
      { if [R.LB, R.UB) ∩ [s, e) ≠ ∅ then
        { S=append(S,R.obj);
          S=append(S,FindObjectsInVideo(R.LLINK,s,e));
          S=append(S,FindObjectsInVideo(R.RLINK, s,e));
        }
      }
    }
  return(S);   end
```

RS-Trees

- Very similar to the frame segment tree.
- The concepts of `OBJECTARRAY` and `ACTIVITYARRAY` remain the same as before.
- Instead of using a segment tree to represent the frame sequences (such as those shown in Figure ??), we take advantage of the fact that a sequence $[s, e)$ is a *rectangle* of length $(e - s)$ and of width 0.
- We already know how to represent a set of rectangles using an R-tree.
- Example:



Video Segmentation

- We have assumed a *logical delineation* of video data – video is brokenup into homogeneous segements.
- Usually, a video is created by taking a set of *shots*.
- These shots are then composed together using specified *composition operators*.
- Shots are usually taken with a fixed set of cameras each of which has a constant relative velocity.
- A *shot composition* operator, often referred to as an *edit effect* is an operation that takes as input, two shots, S_1, S_2, and a duration, t, and *merges* the two shots into a composite shot in within time t.
- Thus, for example, suppose we wish to compose together, two shots S_1, S_2 and suppose these two shots have durations t_1, t_2 respectively. If f is a shot composition operator, then

  $$f(S_1, S_2, t)$$

  creates a segment of video of length $(t_1 + t_2 + t)$. S_1 is first shown and then undergoes a continuous transformation over a time interval t leading to the presentation of S_2 next.
- $f(S_1, S_2, t)$ then is a continuous sequence of video.
- In general, a video as a whole may be represented as:

  $$(f_n(\ldots f_2(f_1(S_1, S_2, t_1), S_3, t_2) \ldots, S_n, t_n).$$

Shot Composition Operators

- **Shot Concatenation:** Concatenates the two shots (even if the transition is not smooth). If shotcat is a shot concatenation operator, then t must be zero, i.e. whenever we invoke shotcat(S_1, S_2, t), the third argument t must be set to 0.
- **Spatial Composition:** Examples include: a *translate* operation which causes two successive shots to be overlayed one on top of the other. For instance, suppose we want to show shot S_1 first, followed by shot S_2. This is done by first overlaying shots S_1 *on top* of shot S_2 and then moving (i.e. translating) shot S_1 away, thus exposing shot S_2.
- **Chromatic Composition:** *fades* and *dissolves*. Both these operations are chromatic scaling oprations that try to continuously transform each pixel (x, y) in the first shot into the corresponding pixel in the second shot.

Video Segmentation Problem

- Given a video V, express the video V in the form:

$$V = f_n(\ldots f_2(f_1(S_1, S_2, t_1), S_3, t_2) \ldots, S_n, t_n).$$

- That is, given video V, find n and shots $S_1, \ldots, S_n$, times $t_1, \ldots, t_n$, and composition operations $f_1, \ldots, f_n$ such that the above equation holds.

Video Standards

- All video compression standards attempt to compress videos by performing an *intra-frame* analysis.
- Each frame is divided up into blocks. Compare different frames to see which data is "redundant" in the two frames.
- Drop redundant data to compress.
- Compression quality is measured by:
  - the fidelity of the color map – how many colors of the original video occur when the compressed video is decompressed ?
  - the pixel resolution per frame – how many pixels per frame of the video have been dropped ?
  - the number of frames per second – how many frames have been dropped?
- Compression Standards: MPEG-1,2,3, Cinepak, JPEG video etc.

MPEG-1

- Stores videos as a sequence of I, P and B frames.
- I frames are independent images called "intra frames". Basically a still image.
- P-frame is computed from the closest I-frame *preceding* it by interpolation (using DCT).
- B-frames are computed by interpolating from the two closest P or I frames.

MPEG-1, Continued

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
I	B	B	P	B	B	I	B	B	P	B	B	I	B	B	P	B	B	I	B	·····

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
I	B	B	P	B	B	I	B	B	P	B	B	I	B	B	P	B	B	I	B	·····

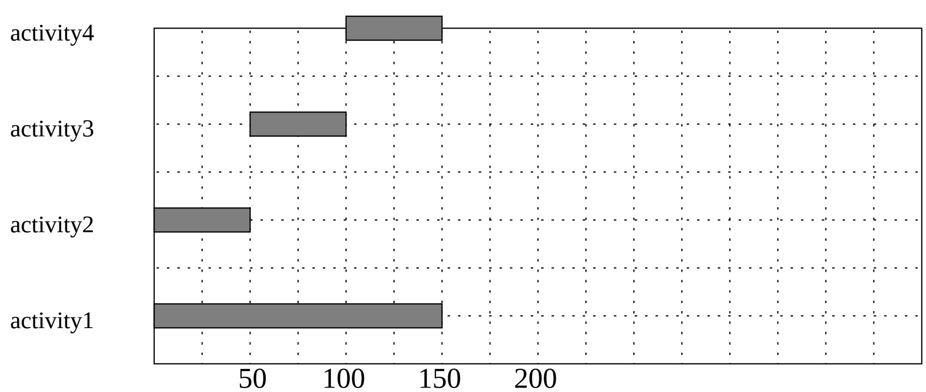
Other Compression Standards

- MPEG-2 uses higher pixel resolution and a higher data rate, thus making it superior to MPEG-1 in terms of the quality of the video as seen by the user. However, it requires higher bandwidth thus making it feasible for some, but not all applications.
- MPEG-3 s even higher sampling rates and frames per second that MPEG-2.

Audio Databases

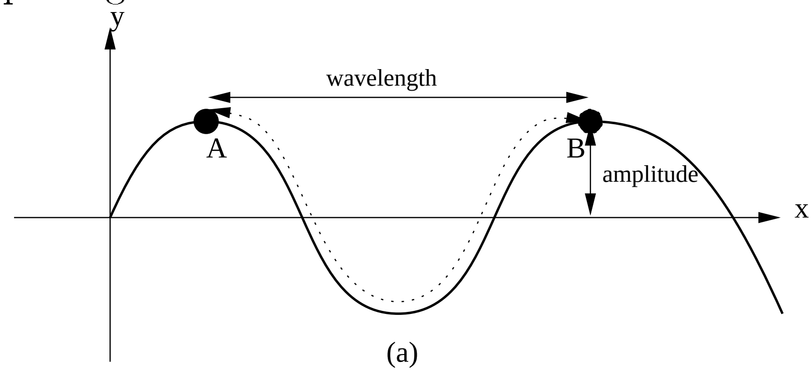
Audio Databases

- Using metadata to represent audio content is done in a very similar way as we did for video.
- For example, consider an audio-recording of an opera.
- Given any interval of time in the opera, there are some activities that occur in that time interval, and some objects that oocur in it.
- For example, we may have the following objects:
  1. `Singers`: a set valued field containing records having a `Role`, `SingerType` and `SingerName` field.
  2. `Score:` this may be a field of type `music_doc` which points to a relevant part of the music score associated with the time segment $[5, 9)$.
  3. `Transcript:` This may be a field of type `document` that points to the relevant part of the libretto (words being sung) during the time segment $[5, 9)$.

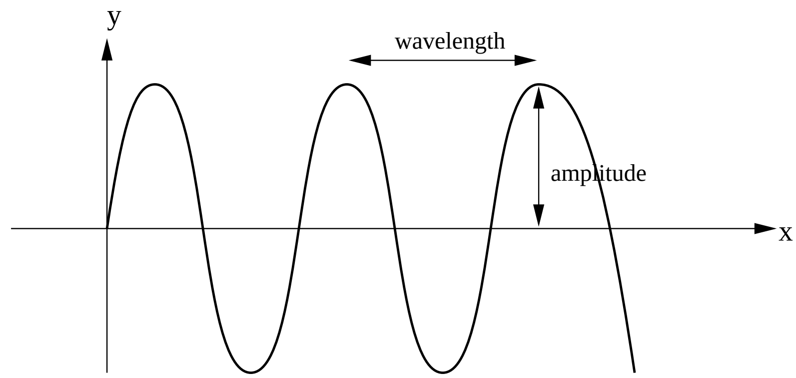


Signal-based Audio Content

- Audio data is considered as a signal, $\phi(x)$, over time x.
- Different *features* of the signal ϕ are extracted, indexed and stored for efficient retrieval.
- Example Signals:



(a)



(b)

- *Period of vibration* T = time taken for a "particle" in the wave to return to its starting position.
- *Frequency of vibration* f = number of vibrations per second. $f = \frac{1}{T}$.

- of the wave is the speed at which the crests and troughs move to the right. If if w denotes the wavelength of the wave, then

$$\begin{aligned} v &= \frac{w}{T}. \\ &= w \times f. \end{aligned}$$

- *Amplitude* a of a wave is the maximum intensity of the signal associated with the wave.

Segmentation

- Split up the audio signal into relatively homogeneous "windows." This may be done in one of two ways.
  - Application developer can specify, a priori, a window size w (in seconds or milliseconds), and assume that the wave's properties within that window are obtained by averaging.
  - Use a homogeneity predicate as in the case of images.

Feature Extraction

- After segmentation, the audio signal may be viewed as a sequence of n windows, $w_1, \ldots, w_n$.
- For each window, we extract some features associated with the audio signal.
- If k features are extracted, then an audio signal may be considered to be a sequence of n points in a k-dimensional space.
- Example features:
  - **Intensity:**

    $$I = 2 \times \pi^2 \times f^2 \times \mu \times a^2 \times v$$

    where f is the frequency of the wave (in Hz), μ is the density of the material through which the sound is being propagated (in kg/m^3), a is the amplitude of the wave in meters, and v is the velocity of the wave in meters per second. The intensity is in Watts per square meters.
  - **Loudness:** Suppose L_0 denotes the loudness associated with the lowest frequency (about 15 Hz) that a human ear can detect, and suppose we are looking at a wave of intensity I. Then the loudness of I, in decibels, is given by:

    $$L = 10 \times \log\left(\frac{I}{L_0}\right).$$

    Notice that when $I = L_0$, then $L = 10 \times \log(1) = 0$.

– **Pitch:** The pitch, $p(f, a)$ of an audio signal is computed as a derived quantity from the frequency f and amplitude a of the signal. Typically, given any window of the sort shown in Figure **??**, the pitch is computed using a gcd algorithm in terms of the frequency and amplitude.

– **Brightness:** The brightness of a signal ϕ in a window w is a measure of how "clear" the sound is. For instance, a muffled sound is less bright than the sound resulting from breaking glass.

Capturing Audio Content through Discrete Transformations

- Even a relatively short audio recording (say of about 10 minutes duration when played back), may have as many as 100,000 windows, if one assumes that each window represents a fairly "smooth" signal.
- Reduce the number of windows of homogeneous segments through the use of the Discrete Fourier Transform (DFT), and the Discrete Cosine Transform (DCT).
- Suppose we have a total of n windows after segmenting the audio signal, but we wish to store this in an array, A, of size N, where N is considerably smaller than n.
- For each field f, we may compute a value for $A[i].f$, $1 \leq i \leq n$, as follows, using the DFT:

$$A[r].f \;=\; \sum_{j=0}^{N-1} \phi(j).f\; exp(\frac{-2\pi \times \mathsf{i} \times r \times j}{N})$$

- Here, $\phi(j).f$ refers to the value of property f at time j of the signal ϕ. As usual, the symbol i denotes the complex number $\sqrt{-1}$.

Indexing Audio Data

Algorithm 5 *CreateAudioIndex(K,N)*

Index = *NIL*; ($\star$ *index is initially empty* $\star$)
for $i = 1$ **to** K **do**
{
for $j = 0$ **to** $(N-1)$ **do** $A^i[j] = DFT(\phi_i)$;
$A^i[N] = i$;
($\star$ *insert vector* $A^i[j]$ *into TV-tree* $\star$)
Index = *insert*($A^i[j]$,*Index*)
}
end

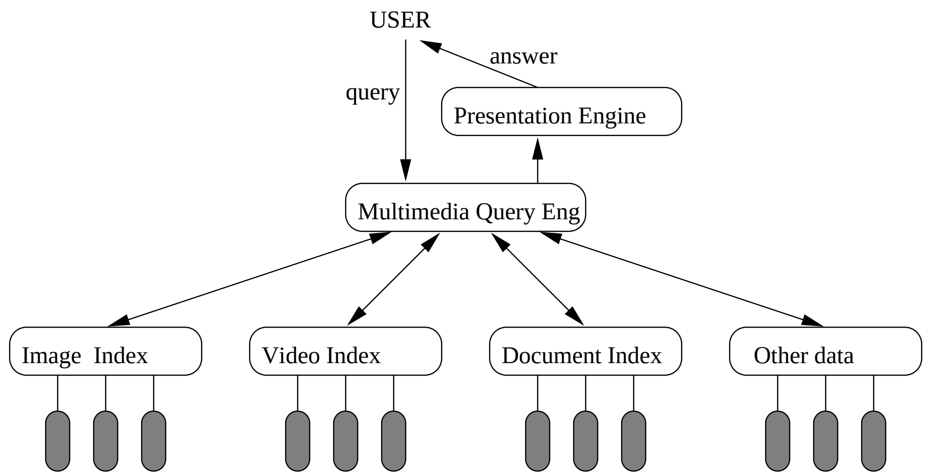
Multimedia Databases

- Thus, far we have shown how to represent and organize the content of different types of *individual* media data.
- What about databases containing a mix of media types?
- Such databases can arise in one of three ways:
  - All the media data is "legacy" data.
  - Some of the media data is "legacy" data, others are being specially crafted.
  - None of the media data is "legacy" data, all of it is being specially crafted.

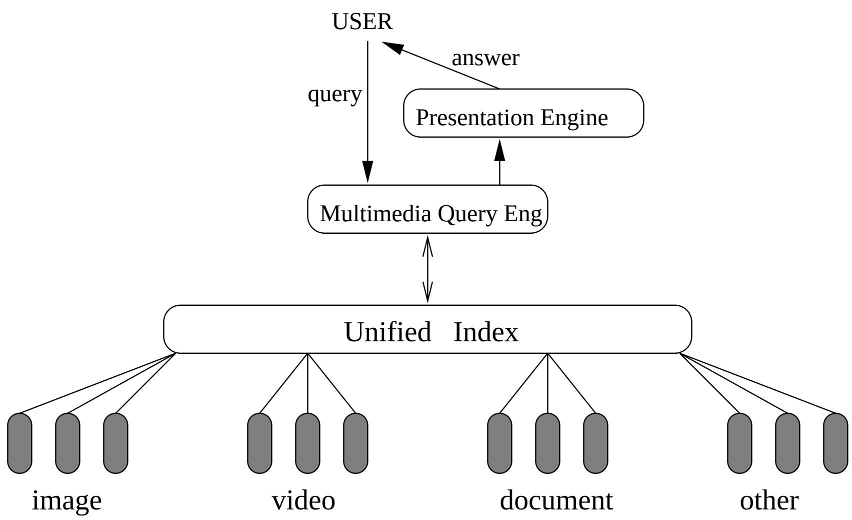
Multimedia Database Architectures

- **(The Principle of Autonomy)** Should we choose to group together all images, all videos, all documents, and index each of them in a way that is maximally efficient for the expected types of accesses we plan to make on those objects ?
- **(The Principle of Uniformity)** Should we try to find a single abstract structure $\mathcal{A}$ that can be used to index *all* the above media types, and that can thus be used to create a "unified index" that can then be used to access the different media objects ?
- **(The Principle of Hybrid Organization)** A third possibility is to use a hybrid of the previous two principles. In effect, according to this principle, certain media types use their own indexes, while others use the "unified" index.

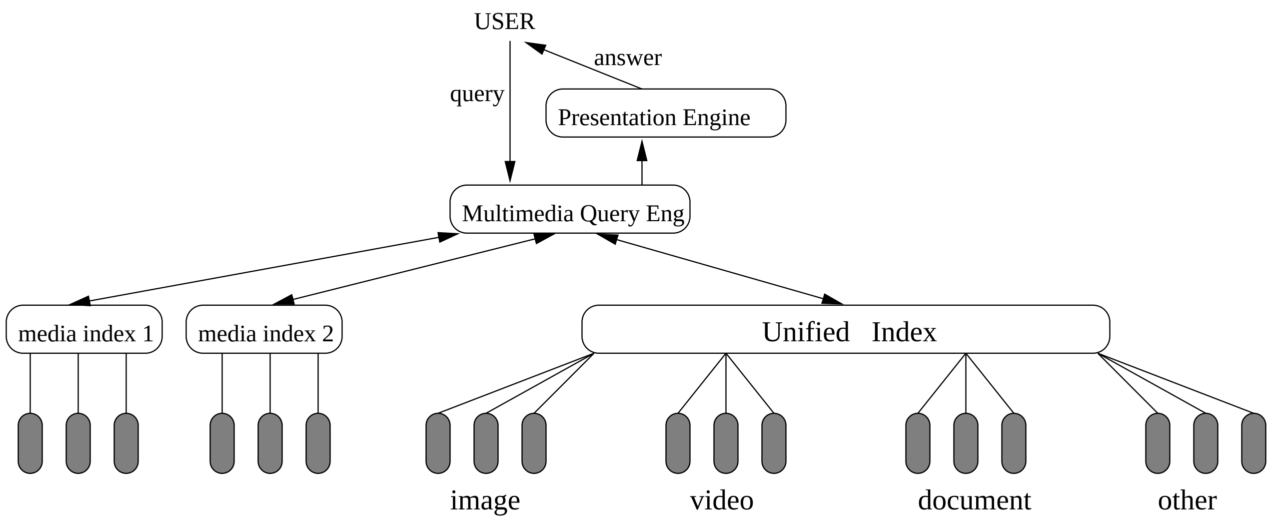
The Principle of Autonomy



The Principle of Uniformity



The Principle of Hybrid Organization



Organizing Multimedia Data based on the Principle of Uniformity

Suppose we consider the following statements about media data. These statements may be made by a human or may be produced as the output of an image/video/text content retrieval engine.

- The image `photo1.gif` shows Jane Shady, Denis Dopeman and an unidentified third person, in Medellin, Colombia. The picture was taken on January 5, 1997.
- The video-clip `video1.mpg` shows Jane Shady giving Daniel Dopeman a briefcase (in frames 50-100). The video was obtained from surveillance set up at Denis Dopeman's house in Rockville, Maryland, in October, 1996.
- The document `dopeman.txt` contains background information on Denis Dopeman, obtained from Mr. Dopeman's FBI file.

All the above statements are *Metadata* statements.

- Associate, with each media object o_i, some meta-data, $\mathsf{md}(o_i)$.
- $\mathsf{md}(o_i)$ may be produced by a human or a program.
- If our archive contains objectx $o_1, \ldots, o_n$, then index the metadata $\mathsf{md}(o_1), \ldots, \mathsf{md}(o_n)$ in a way that provides efficient ways of implementing the expected accesses that users will make.

Media-Abstractions

- If we consider the content of media data of different types, what is it that is common to all these media types, and what is it that is different ?
- We would like a single data structure that can represent different instances of this common "core".
- Media abstractions are mathematical structures representing such media content.
- Media abstractions may be implemented through a single data structure.

Media-Abstractions continued

- A *media-abstraction* is an 8-tuple

$$(\mathcal{S}, \underline{\mathsf{fe}}, \mathsf{ATTR}, \lambda, \Re, \mathcal{F}, \mathsf{Var}_1, \mathsf{Var}_2)$$

  where:

  - $\mathcal{S}$ is a set of objects called *states*, and
  - $\underline{\mathsf{fe}}$ is a set of objects called *features*, and
  - ATTR is a set of objects called *attribute values*, and
  - $\lambda : \mathcal{S} \rightarrow 2^{\underline{\mathsf{fe}}}$ is a map from states to sets of features, and
  - $\Re$ is a set of relations on $\underline{\mathsf{fe}}^i \times \mathsf{ATTR}^j \times \mathcal{S}$ for $i, j \geq 0$, and
  - $\mathcal{F}$ is a set of relations of $\mathcal{S}$ and
  - Var_1 is a set of objects, called *variables*, ranging over $\mathcal{S}$, and
  - Var_2 is a set of variables ranging over $\underline{\mathsf{fe}}$.

Media-Abstractions: Intuition

- A "state" is the smallest "chunk" of media data that we wish to reason about.
- A "feature" is any object in a state that is deemed to be of "interest." Features can in principle include both objects and activities.
- A feature may have one or more attributes associated with it.
- The feature extraction map specifies what features occur in which states. In some cases, feature extraction maps are implemented as content extraction programs, while in other cases, they may involve humans manually specifying content.
- $\Re$ is a set of state-dependent and state-independent relations, e.g. `left-of` is state dependent, while `age` is not.
- $\mathcal{F}$ contains inter-state relations, e.g `before`(s1,s2) may say that state s1 occurred before state s2.

Image Data Viewed as a Media-Abstraction

- The set of states consists of {pic1.gif, . . . , pic7.gif}.
- **Features:** The set of features consists of the names of the people shown in the photographs. Let us call this list: Bob, Jim, Bill, Charlie, and Ed.
- **Extraction Map λ:** This map tells us, for each state, which features occur in that state. The table below contains this description:

State	Feature
pic1.gif	bob,jim
pic2.gif	jim
pic3.gif	bob
pic4.gif	bill
pic5.gif	charlie
pic6.gif	ed, bill
pic7.gif	ed

- The set of relations may contain just two relations: a state-dependent relation called `left_of` and a state-independent relation called `father`, with the obvious meanings.
- The set of inter-state relations may be empty.

Video Data Viewed as a Media-Abstraction

- **States:** The set of states consists just of frames 1 through 5.
- **Features:** The set of features consists of: Jane Shady, Denis Dopeman, Dopeman_house and briefcase,
- **Extraction Map λ:** The extraction map, λ, is described by the following simple table:

State	Feature
frame1	Dopeman_house, briefcase, Jane Shady
frame2	Dopeman_house, briefcase, Jane Shady, Denis Dopeman
frame3	Dopeman_house, briefcase, Jane Shady, Denis Dopeman
frame4	Dopeman_house, briefcase, Jane Shady, Denis Dopeman
frame5	Dopeman_house, Jane Shady

- **Relations:**
  1. `have` is a state-dependent relation specifying who has an object in a given state. This may be given by the following simple table:

Person	Object	State
Jane Shady	briefcase	1
Jane Shady	briefcase	2
Jane Shady	briefcase	3
Denis Dopeman	briefcase	4

2. `spouse` is a state-independent relation specifying the name of the spouse of an individual. This may be given by the simple table:

Person	Spouse
Jane Shady	Peter Shady
Jane Shady	Peter Shady
Denis Dopeman	Debra Dopewoman

3. **Inter-State Relations:** This may consist of just one relation called `before(s1,s2)` which holds iff state `s1` occurs before state `s2`.

Simple Multimedia Database

- A *simple multimedia database* is a finite set, $\mathcal{M}$, of media-abstractions.
- Simple multimedia databases are naive.
- A a media abstraction may list "Church" as a feature; however, when searching for "Cathedrals" or "Monuments", we may not find the church, because the system does not know that cathedrals and churches are (more or less) synonymous, and that all churches are monuments (but not vice-versa).
- Also, users often search for media objects containing one or more features, and then "refine" the search later when they find that the media-objects returned by their query, though correct, do not correspond precisely to what they wanted.

Structured Multimedia Database

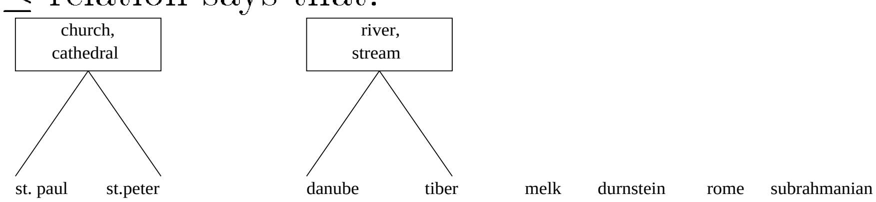
A *structured multimedia database system,* SMDS, is a 5-tuple $(\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \equiv, \leq, \mathsf{inh}, \mathsf{subst})$ where:

- $\mathcal{M}_i = (\mathcal{S}^i, \underline{\mathsf{fe}}^i, \mathsf{ATTR}^i, \lambda^i, \Re^i, \mathcal{F}^i, \mathsf{Var}_1^i, \mathsf{Var}_2^i)$ is a media-abstraction, and
- $\equiv$ is an equivalence relation on $\mathcal{F} = \cup_{i=1}^{n} \underline{\mathsf{fe}}^i$;
- $\leq$ is a partial ordering on the set $\mathcal{F}/\equiv$ of equivalence classes on $\mathcal{F}$, and
- $\mathsf{inh} : \mathcal{F}/\equiv \rightarrow 2^{\mathcal{F}/\equiv}$ such that $[f_1] \in \mathsf{inh}([f_2])$ implies that $[f_1] \leq [f_2]$. Thus, inh is a map that associates with each feature f, a set of features that are "below" f according to the $\leq$-ordering on features.
- subst is a map from $\cup_{i=1}^{n} \mathsf{ATTR}^i$ to $2^{\cup_{i=1}^{n} \mathsf{ATTR}^i}$.

Example SMDS

Media	Object	Part/frame	Feature(s)
image	photo1.gif	-	church, durnstein, danube, subrahmanian
image	photo2.gif	-	cathedral, melk, subrahmanian
image	photo3.gif	-	church, st. paul, rome
video	video1.mpg	1-5	church, durnstein, stream
video	video1.mpg	6-10	stream
audio	audio1.wav	1-20	st. peters, tiber, rome

- three media-abstractions, one each associated with image, video, and audio data.
- the set of features, $\mathcal{F}$ contains: **church, durnstein, danube, subrahmanian, cathedral, melk, st. paul, rome, stream, restaurant, st. peters, tiber.**
- $\equiv$ says that:
  - **church** $\equiv$ **cathedral**.
  - **river** $\equiv$ **stream**.
- the $\leq$ relation says that:



Querying SMDSs (Uniform Representation)

Basic SMDS functions are:

- `FindType(Obj)`: This function takes as input, a media object `Obj` as input, and returns the output type of the object. For example,

$$\texttt{FindType(im1.gif)} = gif.$$
$$\texttt{FindType(movie1.mpg)} = mpg.$$

- `FindObjWithFeature(f)`: This function takes as input, a feature `f` and returns as output, the set of all media-objects that contain that feature. For example,

$$\texttt{FindObjWithFeature(john)} = \{\text{im1.gif,im2.gif,im3.gif, video1.mpg:[1,5]}.$$
$$\texttt{FindObjWithFeature(mary)} = \{\text{video1.mpg:[1,5], video1.mpg:[15,50]}.$$

- `FindObjWithFeatureandAttr(f,a,v)`: This function takes as input, a feature `f`, an attribute name `a` associated with that feature, and a value `v`. It returns as output, all objects o that contain the feature and such the value of the attribute `a` in object o is `v`. For example,

  1. `FindObjWithFeatureandAttr(Jane Shady,suit,blue)`: This query asks to find all media objects in which Jane Shady appears in a blue suit.

2. `FindObjWithFeatureandAttr(Elephant,bow,red)`: This query asks to find all media objects in which an elephant wearing a red bow appears.

- `FindFeaturesinObj(Obj)`: This query asks to find all features that occur within a given media object. It returns as output, the set of all such features. For example,

  1. `FindFeaturesinObj(im1.gif)`: This asks for all features within the image file `im1.gif`. It may return as output, the objects John, and Lisa.
  2. `FindFeaturesinObj(video1.mpg:[1,5])`: This asks for all features within the first 5 frames of the video file `video1.mpg`. The answer may include objects such as Mary and John.

- `FindFeaturesandAttrinObj(Obj)`: This query is exactly like the previous query except that it returns as output, a relation having the scheme:

  $$(\texttt{Feature}, \texttt{Attribute}, \texttt{Value})$$

  where the triple (f, a, v) occurs in the output relation iff feature f occurs in the query `FindFeaturesinObj(Obj)` and feature f's attribute a is defined and has value v. For example,

  `FindFeaturesandAttrinObj(im1.gif` may return as answer, the table:

Feature	Attribute	Value
John	age	32
John	address	32 Pico Lane, Mclean, VA 22050.
Mary	age	46
Mary	address	16 Shaw Road, Dumfries, VA 22908.
Mary	employer	XYZ Corp.
Mary	boss	David

SMDS-SQL

- All ordinary SQL statements are SMDS-SQL statements. In addition:
- The SELECT statement may contain *media-entities*. A *media entity* is defined as follows:
  1. If m is a continuous media object, and i, j are integers, then $m : [i, j]$ is a media-entity denoting the set of all frames of media object m, that lie between (and inclusive of) segments i, j.
  2. If m is not a continuous media-object, then m is a media entity.
  3. If m is a media-entity, and a is an attribute of m, then $m.a$ is a media-entity.
- The FROM statement may contain entries of the form

  $$\langle media \rangle \ \langle source \rangle \langle M \rangle$$

  which says that only all media-objects associate with the named media type and named data source are to be considered when processing the query, and that M is a variable ranging over such media objects.
- The WHERE statement allows (in addition to standard SQL constructs), expressions of the form:

  $$term \ \mathsf{IN} \ func\_call$$

  where:

1. *term* is either a variable (in which case it ranges over the output type of *func_call*) or an object having the same output type as *func_call* and
2. *func_call* is any of the five function calls listed above.

SMDS-SQL Examples

- *Find all image/video objects containing both Jane Shady and Denis Dopeman.* This can be expressed as the SMDS-SQL query:

```
SELECT    M
FROM      smds source1 M
WHERE     (FindType(M)=Video OR FindType(M)=Image)
          AND
          M IN FindObjWithFeature(Denis Dopeman)
          AND
          M IN FindObjWithFeature(Jane Shady).
```

- *Find all image/video objects containing Jane Shady wearing a purple suit.* This can be expressed as the SMDS-SQL query:

```
SELECT    M
FROM      smds source1 M
WHERE     (FindType(M)=Video OR FindType(M)=Image)
          AND
          M IN FindObjWithFeatureandAttr(Jane Shady,s
```

- *Find all images containing Jane Shady and a person who appears in a video with Denis Dopeman.* Unlike the preceding queries, this query involves computing a "join" like operations across different data domains. In order to do this, we use existential variables such as the variable "Person" in the query below, which is used to refer to the existence of an "unknown" person whose identity is to be determined.

```
SELECT    M,Person
FROM      smds source1 M,M1
WHERE     FindType(M)=Image) AND
          FindType(M1)=Video) AND
          M IN FindObjWithFeature(Jane Shady) AND
          M1 IN FindObjWithFeature(Denis Dopeman)
          AND
          Person IN FindFeaturesinObj(M)AND
          Person IN FindFeaturesinObj(M1)AND
          Person≠Jane Shady AND Person≠Denis Dopeman.
```

Querying Hybrid Representations of Multimedia Data

- SMDS-SQL may be used to query multimedia objects which are stored in the uniform representation.
- "What is it about the hybrid representation that causes our query language to change?"
- In the uniform representation, all the data sources being queried are SMDSs, while in the hybrid representation, different (non-SMDS) representations may be used.
- A hybrid media representation basically consists of two parts – a set of media objects that use the uniform representation (which we have already treated in the preceding section), and a set of media-types that use their own specialized access structures and query language.
- To extend SMDS-SQL to Hybrid-Multimedia SQL (HM-SQL for short), we need to do two things:
  - First, HM-SQL must have the ability to express queries in each of the specialized languages used by these non-SMDS sources.
  - Second, HM-SQL must have the ability to express "joins" and other similar binary algebraic operations between SMDS-sources, and non-SMDS sources.

HM-SQL

HM-SQL is exactly like SQL except that the SELECT, FROM, WHERE clauses are extended as follows:

- the SELECT and FROM clauses are treated in exactly the same way as in SMDS-SQL.
- The WHERE statement allows (in addition to standard SQL constructs), expressions of the form:

  $$term \ \mathsf{IN\ MS} : func\_call$$

  where:

  1. *term* is either a variable (in which case it ranges over the output type of *func_call*) or an object having the same output type as *func_call* as defined in the media-source MS and
  2. either MS =SMDS and em func_call is one of the five SMDS functions described earlier, or
  3. MS is not an SMDS-media source, and *func_call* is a query in QL(MS).
- Thus, there are 2 differences between HM-SQL and SMDS-SQL:
  1. *func_call*s occurring in the WHERE clause must be explicitly annotated with the media-source involved, and

2. queries from the query languages of the individual (non-SMDS) media-source implementations may be embedded within an HM-SQL query. This latter feature makes HM-SQL very powerful indeed as it is, in principle, able to express queries in other, third-party, or legacy media implementations.

HM-SQL Examples

- *Find all video clips containing Denis Dopeman, from both the video sources, video1, and video2.*

```
SELECT    M
FROM      smds video1, videodb video2
WHERE     M IN smds:FindObjWithFeature(Denis Dopeman) OR
          M IN videodb:FindVideoWithObject(Denis Dopeman).
```

- *Find all people seen with Denis Dopeman in either video1, video2, or idb.*

```
(SELECT   P1
FROM      smds video1 V1
WHERE     V1 IN smds:FindObjWithFeature(Denis Dopeman) AND
          P1 IN smds:FindFeaturesinObj(V1) AND
          P1≠ Denis Dopeman)
UNION
(SELECT   P2
FROM      videodb video2 V2
WHERE     V2 IN smds:FindObjWithFeature(Denis Dopeman) AND
          P2 IN videodb:FindObjectsinVideo(V1) AND
          P2≠ Denis Dopeman)
UNION
(SELECT   *
FROM      imagedb idb I2
WHERE     V2 IN image:getpic(Denis Dopeman) AND
          P2 IN imagedb:getfeatures(V1) AND
          P2≠ Denis Dopeman).
```

Indexing SMDSs with Enhanced Innverted Indexes

Suppose $(\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}, \equiv, \leq, \mathsf{inh}, \mathsf{subst})$ where

$$\mathcal{M}_i = (\mathcal{S}^i, \underline{\mathsf{fe}}^i, \mathsf{ATTR}^i, \lambda^i, \Re^i, \mathcal{F}^i, \mathsf{Var}_1^i, \mathsf{Var}_2^i)$$

is an SMDS.

1. FEATURETABLE: This is a hash table whose entries are features in $\cup_{i=1}^n \underline{\mathsf{fe}}^i$. Each hash table location i contains a bucket of "featurenodes" that hash to location i.
2. STATETABLE: This is a hash table whose entries are states in $\cup_{i=1}^n \mathcal{S}^i$. Like the FEATURETABLE, the STATETABLE contains a bucket of "statenodes" that hash to the specified location.
3. FEATURENODEs: Each featurenode contains:
   - the name of the feature (e.g. "Denis Dopeman"),
   - a list of children nodes (if f_1, f_2 are features in $\cup_{i=1}^n \underline{\mathsf{fe}}^i$, we say that f_2 is a child of f_1 iff $f_2 \leq f_1$ and there is no other feature $f_3 \in \cup_{i=1}^n \underline{\mathsf{fe}}^i$ such that $f_2 < f_3 < f_1$)
   - a list of pointers to statenodes (see below) that contain that feature (e.g. in the case of an image database, this would be a pointer to the nodes associated with images that contain the feature in question, e.g. Denis Dopeman)

- a list of pointers to other featurenodes that are appropriate substitutes for the featurenode in question. Specifically, if we consider a feature node associated with feature f, then a pointer to feature g is in this list iff $g \mathsf{inh}(f)$.

4. STATENODEs: A statenode consists of just two components: a pointer to a file containing the media-object (image, video, audio, document, etc.) that the state in question refers to, and a linked list whose members point to featurenodes – intuitively, there is a pointer to a featurenode f iff the feature in question is in the state.

Data Structure

```
type featurenode = record of  /* nodes in feature graph */
     name : string;  /* nameof feature */
     children: ^node1; /* points to a list of pointers to the children */
     statelist: ^node2; /* points to a list of pointers to states that*/
                             /* contain this feature */
     replacelist: ^node3; /* points to a list of descendants whose */
                           /* associated states can be deemed to have the */
                           /* the feature associated with this node */
     end record;

type node1 record of
     element: ^featurenode; /* points to a child of a featurenode */
     next :   ^node1; /* points to next child */
     end record;

type node2 record of
     state: ^statenode; /* pointer to the list associated with a state */
     link:  ^node2;  /* next node */
     end record;

type node3 record of
     feat: ^featurenode; /* pointer to a node that can be deemed to have */
                          /* the feature associated with the current node */
     link1: ^node3;
     end record;

type statenode record of
     rep:  ^framerep;
     flist: ^node4;
     end record

type node4 record of
     f : ^featurenode;
     link2: ^node4;
     end record:
```

Example

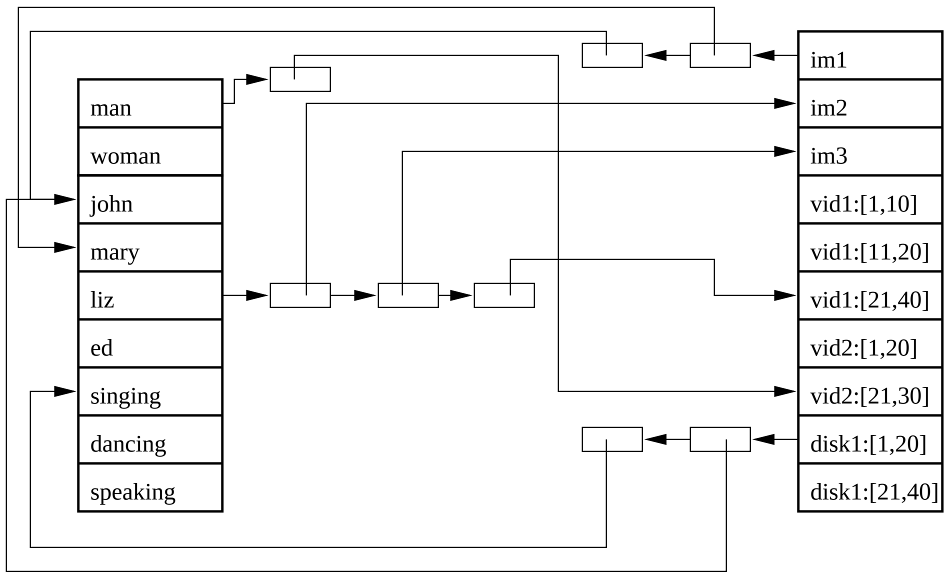
- Consider a very simple toy SMDS containing three media-abstractions: images, video, and audio, with following content:

Media-Abstraction	State	Features
image	im1	john, mary
	im2	john, mary, liz
	im3	liz, mary
video	vid1:[1,10]	john, singing
	vid1:[11,20]	john,mary, dancing
	vid1:[21,40]	john, mary, liz, singing, dancing
	vid2:[1,20]	ed, speaking
	vid2:[21,30]	man, speaking
audio	disk1:[1,20]	john, singing
	disk1:[21,40]	woman, speaking

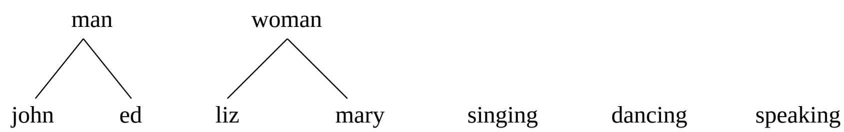
- Next slide shows one possible STATETABLE (without hashing) and a possible FEATURETABLE.
- Slide after that shows the $\leq$ ordering on features.
- We have shown the `featurelist` associated with the states `im1` and `disk1:[1,20]`. Other feature lists are not shown for the sake of simplicity.
- Likewise, we have shown the `statelists` associated with the features `man` and `liz`.

- The relations associated with these media-abstractions may be stored within standard relational databases.

Example of STATETABLE



Example of $\leq$ Ordering



Example Algorithm FindObjwithFeature(f)

Algorithm 6 *FindObjwithFeature(f);*

1. $SOL = \emptyset$;
2. *Hash feature f and resolve collisions (if any) till feature f is located (at location i say) in the FEATURETABLE;*
3. *SOL = append(SOL,Featuretable[i].Statelist);*
4. **Forall** *children f' of some member of $[f]$* **do** *SOL = append(SOL,FindObjwithFeature(f'))*
5. **Return** *SOL.*

This assumes that if f' is a descendant of f then f' occurs in every state in which f occurs. We may easily modify the above algorithm if this assumption is not valid.

Find features in state s

1. Hash state s and resolve collisions (if any) till state s is located (at location i say) in the STATETABLE;
2. **Return** STATETABLE[i].Featurelist.

Check if state s contains feature f

1. Hash state s and resolve collisions (if any) till state s is located (at location i say) in the STATETABLE;
2. **If** f is in STATETABLE[i].Featurelist, then **Return** true **else Return** false.

Query Relaxation/Expansion

- The inh and subst components of an SMDS are used to determine how queries must be relaxed.
- When a user poses a query Q, we *somehow modify* that query Q into a set $\{Q_1, \ldots, Q_k\}$ of queries.
- This set is partially ordered and contains the original query Q as the maximal element of the ordering.
- Intuitively, if Q' is a child of Q'' then this means that Q' is obtained from Q'' by making a modification or relaxation.
- Thus, query relaxation depends on two aspects:
  - what are the "modification" operations that are allowed to modify queries?
  - what is the ordering on the set of modified queries?

FindObjwithFeatureandInh(f,inh)

Algorithm 7 *FindObjwithFeatureandInh(f,*inh*);*

1. *SOL* $= \emptyset$;
2. *Hash feature f and resolve collisions (if any) till feature f is located (at location i say) in the FEATURETABLE;*
3. *SOL = append(SOL,Featuretable[i].Statelist);*
4. **Forall** *children* f' *of some member of* inh([f]) **do** *SOL = append(SOL,FindObjwithFeatureandInh(f', $\emptyset$))*
5. **Return** *SOL.*

Query Relaxation, Continued

– Q_1 is a *feature-relaxation* of Q_2, denoted $Q_1 \sqsubseteq Q_2$ iff there exist features $f_1, f_2 \in \cup_{i=1}^{n} \underline{\mathsf{fe}}^i$ such that

  * $Q_1 \preceq Q_2[f_1/f_2]$ (where the notation $Q_2[f_1/f_2]$ denotes the replacement of all occurrences of f_1 in Q_2 by f_2) and
  * $f_2 \in \mathsf{inh}([f_1])$.

– **EX:** Q_2 is:

```
SELECT   M
FROM     smds video1, videodb video2
WHERE    M IN smds:FindObjWithFeature(Denis Dopeman) OR
         M IN videodb:FindVideoWithObject(Denis Dopeman)
```

– Suppose David Johns in in inh(Denis Dopeman).

– Q_1 is:

```
SELECT   M
FROM     smds video1, videodb video2
WHERE    M IN smds:FindObjWithFeature(David Johns) OR
         M IN videodb:FindVideoWithObject(David Johns)
```

– Q_1 is a feature relaxation of Q_2.

Relaxation Examples

– Q_4:

```
SELECT   M
FROM     smds
WHERE    M IN smds:FindObjWithFeature(Denis Dopeman) AND
         M IN smds:FindObjWithFeatureandAttr(briefcase,color,black).
```

– Q_5:

```
SELECT   M
FROM     smds
WHERE    M IN smds:FindObjWithFeature(Denis Dopeman) AND
         M IN smds:FindObjWithFeatureandAttr(package,color,grey).
```

– Q_6:

```
SELECT   M
FROM     smds
WHERE    M IN smds:FindObjWithFeature(Denis Dopeman) AND
         M IN smds:FindObjWithFeatureandAttr(briefcase,color,grey).
```

– (Feature Replacement) obtained Q_4 by replacing occurrences of "package" in Q_3 with "briefcase"

– (Attribute Replacement) obtained Q_5 by replacing occurrences of "black" by "grey"

– (Feature-cum-Attribute Replacement) obtained Q_6 by replacing occurrences of "black" by "grey" and occurrences of "package" with "briefcase."

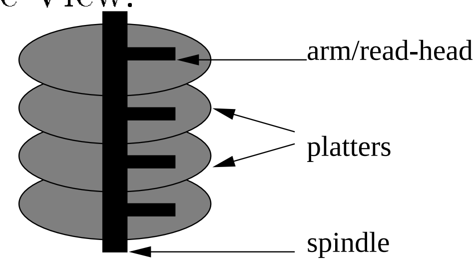
Relaxations, Continued

- Q_1 is said to be a *attribute-relaxation* of Q_2, denoted $Q_1 \sqsubseteq_a Q_2$ iff there exist attributes $a_1, a_2 \in \cup_{i=1}^{n} \mathsf{ATTR}^i$ such that $Q_1 = S_2[a_1/a_2]$ and $a_2 \in \mathsf{subst}(a_1)$.
- *Relaxation* of queries is defined inductively as follows:
  - If Q_1 is a feature-relaxation of Q_2, then Q_1 is a relaxation of Q_2;
  - If Q_1 is a attribute-relaxation of Q_2, then Q_1 is a relaxation of Q_2;
  - If Q_1 is a relaxation of Q_3 and Q_3 is a relaxation of Q_2, then Q_1 is a relaxation of Q_2.
- $\mathsf{Relax}(Q)$ denotes the set of all relaxations of query Q (w.r.t. some SMDS).

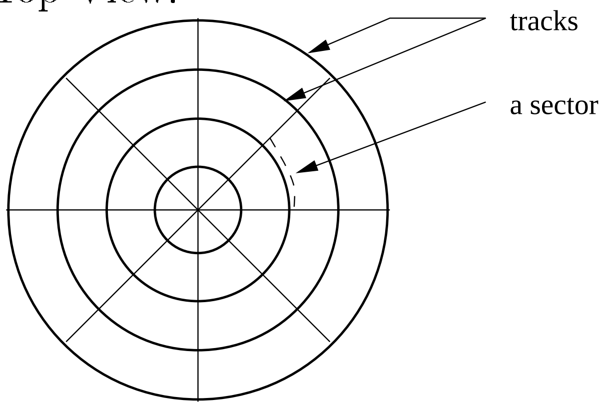
Retrieving Multimedia Data from Disks

Overview of Disks

- Disk drive Side View.



- Disk drive Top View.



- **Tracks:** Each disk platter consists of a number of concentric "tracks."
- **Cylinders:** Suppose we are intersected in track number i for some i. Each platter in our disk device contains such a track. The set of such tracks, one from each platter, is called a *cylinder*.
- **Regions:** Each disk platter is divided into k regions (for some fixed k. Each region represents a wedge of the platter with angle $\frac{360}{k}$.

- **Sectors:** That part of a track that intersects a wedge is called a sector. Clearly, if we have n tracks altogether, then we will have n sectors per wedge.

Disk Retrieval

- Associated with each disk platter is a disk arm that contains a read-write head to read/write from/to that platter.
- When a disk address is to be accessed, the disk controllers (the programs that control the position of the disk head relative to the disk) start by pursuing two steps:
  1. **Seek Operations:** First, find the track (and hence the cylinder) on which the address is located. Seek time has 4 phases:
     (a) acceleration phase
     (b) constant velocity/coast phase
     (c) deceleration phase
     (d) settle phase.
  2. **Rotational Operation:** Once the head is positioned over the right track, the disk spindle rotates so that the sector containing the desired physical address is located directly under the read/write head. The time taken for this is called *rotational latency.*
- **Read Head:** Associated with each disk arm is a read/write head that contains the necessary hardware to read data from a sector, or write data onto a sector.
- **Transfer Rate:** Rate at which data is read/written. Varies based on whether reading or writing.

Notation

Symbol	Meaning
$tnum$	total number of tracks
$rnum$	total number of regions
itd	distance between two tracks
ss	spin speed of disk in rotations per minute
rv	average radial velocity = average movement of disk head along arm
`dtr`	The transfer rate of the disk
rd	Recording density in Mbytes per sector

- Suppose we wish to read sector i (on track t_i) on a given platter, and the read head on that platter is currently over sector j in track t_j.
- $$Readtime(i, j) = \frac{rd}{\mathtt{dtr}} + spintime(i, j) + Sk(t_i, t_j)$$
  where $spintime(i, j)$ is the amount of time required to spin from sector i to sector j and $Sk(t_i, t_j)$ is the amount of time for the read head to move from track t_i to track t_j.
- The seek time required to find track i from track j (assuming the head is currently positioned at track j) is given (crudely) by:

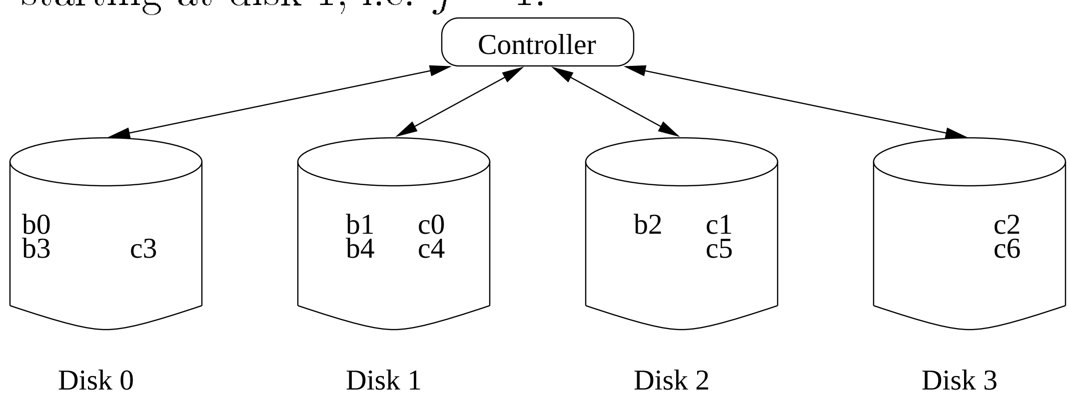
$$Sk(t_i, t_j) = \frac{\mathrm{abs}(t_i - t_j)}{rv}.$$

- Similarly,

$$spintime(i,j) \;=\; \mathrm{abs}((i-j) \bmod rnum) \times \frac{360}{rnum} \times \frac{1}{av}.$$

Raid-0 Architecture

- We have a set of n disks, labeled $0, 1, \ldots, (n-1)$.
- A k-*stripe* is a set of k drives for some integer $k < n$ which divides n.
- When storing a set $b_0, b_1, \ldots, b_{r-1}$ of contiguous blocks in terms of a k striped layout, what we do is the following. We store block b_0 on disk 0, block b_1 on disk 1, block b_2 on disk 2, and so on.
- Example striping: 2 movies. The blocks of the first movie are denoted by $b0, b1, b2, b3, b4$. These are striped with $k = 3$ starting at disk 0. Second movie has six blocks, denoted by $c0, \ldots, c5$, and these are being striped with $k = 4$ and starting at disk 1, i.e. $j = 1$.

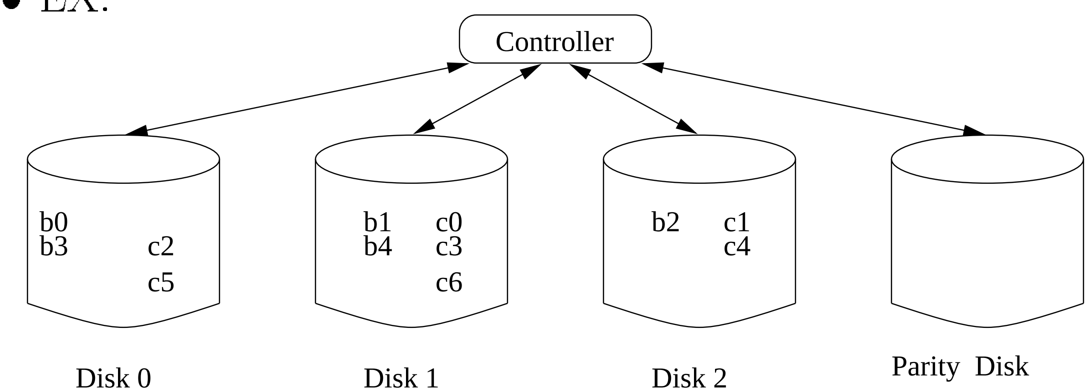


Raid-1 Architecture

- If there are N disks available altogether, then $n = \frac{N}{2}$ disks are utilized.
- For each disk, there is a "mirror" disk.
- Raid-1 is predicated on the assumption that there is a very low probability that a disk and its mirror will fail simultaneously.
- When we wish to read from a disk, we read from the disk (if it is active) or we read from its mirrored disk (if the disk has crashed). When writing to disk d, we must write on both the disk and its mirror.

Raid-5 Architecture

- Each cluster of k disks has one disk reserved as a "parity" disk.
- Suppose that $k = n$, i.e. we have only one cluster (RAID-5 also applies when $k \neq n$, this is just an assumption for illustrative purposes).
- Let us further assume that these disks are numbered $0, 1, \ldots, (n-1)$.
- EX:



- In this case, movie blocks are striped across $(n-1)$ of the n disks available, and disk number $(n-1)$ is reserved as a "parity" disk.
- Suppose we use $D_i.j$ to denote the value of the j'th bit of disk i and suppose disk n is the parity disk. If the symbol $\oplus$ denotes the exclusive-or operator, then

$$D_{n-1}.j \; = \; D_0.j \oplus \cdots \oplus D_{n-2}.j.$$

- I.e. the j'th bit of the parity disk is obtained by taking the exclusive-or of the j'th bits of all the other disks.
- This can be used to recover data if one disk crashes.

Example

- Consider a simple example, where $n = 3$.
- The truth table for exclusive or is:

D_1	D_2	D_3	(Parity disk) $D_p = D_1 \oplus D_2 \oplus D_3$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0

- Suppose disk D_2 crashes and we wish to find the value for a specific bit j.
- Read the value of bit j in disks D_1, D_3 and the parity disk D_p.
- For instance, hese values might be $0, 1, 1$ respectively.
- Then examine the above truth table to see which row has $D_1 = 0, D_3 = 1$ and $D_p = 1$.
- The second last row in the table satisfies this condition.
- From this, we can infer the value of this bit in disk D_2 to be 0.
- How can we do this in general?

Service Algorithms

- Given a set of clients each of whom wants to read some data from disk, how do we do schedule their reads?
- Several algorithms proposed for this task.
- Most such algorithms *must* execute very fast, i.e. it cannot take too long for the algorithm to determine order of reads.
- Some well known algorithms:
  - First Come First Serve (FCFS)
  - SCAN
  - SCAN Earliest Deadline First (SCAN-EDF)

First Come First Served

- Each client retrieval request has an associated time stamp.
- Clients are serviced in order of their associated time-stamp.
- EXAMPLE: Suppose the disk read head is currently over track i, sector j. The table below shows some numbers specifying client requests, when they were made, and how long it takes to reposition the disk head from its current location to the desired location.

RequestID	ReqTime	Est. Seek	Est. Rotational Delay
r1	10	24	3
r2	8	12	5
r3	14	30	6
r4	11	18	4

- In this case, FCFS will serve the requests in the order: r2,r1,r4,r3.
- The last 2 columns are completely ignored by FCFS.

SCAN Algorithm

- Suppose the disk read head is currently over track i, sector j.
- In this case, we order requests in the order of the number of tracks to be traversed from track i, moving either outwards first and then inwards, or vice versa, but not both. (Of course, the further away the tracks
- We then service the requests in the order prescribed.
- EXAMPLE: Suppose we return to the last example, and each track requires 3 units of time to be traversed (estimate). Then

RequestID	ReqTime	Est. Seek (Num of tracks)	Est. Rotational Delay
r1	10	24(8)	3
r2	8	12(4)	5
r3	14	30(10)	6
r4	11	18(6)	4

- If we assume that all of r1–r4 are in tracks beyond track i (i.e. between track i and the outer rim of the disk) then it is easy to see that we will provide service in the order r2,r4,r1,r3.

SCAN-EDF Algorithm

- EDF stands for "Earliest Deadline First".
- Here, we first group all requests in ascending order of their deadline.
- Suppose the resulting groups are G1,G2,...Gn.
- Each group is serviced using SCAN.
- EXAMPLE:

RequestID	ReqTime	Est. Seek	Est. Rotational Delay	Deadline
		(Num of tracks)		
r1	10	24(8)	3	100
r2	8	12(4)	5	120
r3	14	30(10)	6	120
r4	11	18(6)	4	100

- Here we have two groups, G1 which contains r1,r4, and G2 which contains r2,r3.
- G1 is serviced first using SCAN. By the same assumptions as on the previous slide, we first service r4, then r1.
- G2 is then serviced using SCAN. By the same assumptions as on the previous slide, we first service r2, then r3.
- Thus, the overall order of service is r4,r1,r2,r3.

Building Disk-based Media Servers

- Must service multiple clients simultaneously.
- Clients do not want just playback functionality, they also want to perform interactive operations like rewind, fast forward, pause, etc.
- For each client, the server must:
  - provide continuous playback
  - this requires filling his buffer at just the "right" rate.
  - Too fast → buffer might get overwritten.
  - Too slow → client might experience service interruption.

Symbol	Meaning
$\texttt{bnum}(\mathcal{M}_i)$	Number of blocks in movie $\mathcal{M}_i$
$\texttt{buf}(i)$	The total buffer space associated with disk server i
$\texttt{cyctime}(i,t)$	the total cycle time for server i at time t
$\texttt{dtr}(i)$	The total disk bandwidth associated with disk server i
$\texttt{switchtime}(i,t)$	the time required for disk server i to switch from one client's job to another client's job at time t
$\texttt{cons}(i,t)$	The consumption rate of client C_i at time t
$\texttt{data}(i,t)$	The event specification for the client C_i at time t
$\texttt{timealloc}(i,j,t)$	The time-slice allocated to client j at time t
$\texttt{active}(t)$	The set of all clients that are active at time t
$\texttt{d\_active}(i,t)$	The set of all clients that have been assigned a non-zero time-allocation by disk server i
$\wp(\mathcal{M}_i, b)$	The set of servers that contain block b of movie $\mathcal{M}_i$ according to placement mapping $\wp$
$\mu_t(i)$	The set of servers handling requests by client C_i at time t
$\texttt{bufreq}(j,i,t)$	The buffer space needed at server i to match the consumption rate of client j
$\mathcal{S}(t)$	The state of a movie-on-demand system

Figure 1: Notation and terminology

Parameters

- We assume that we have n disk servers, $d_1, \ldots, d_n$ and m movies $m_1, \ldots, m_k$.
- Each movie is a contiguous sequence of blocks. Block size can be fixed in any way.

Client Request

- $\texttt{data}(i,t)$ is a set of pairs of the form (m,b) where m is a movie, and b is a block in that movie.
- Playback:

$$\texttt{data}(i,t) \;=\; \{(\texttt{m},\texttt{b}),(\texttt{m},\texttt{b}+\texttt{1}),\ldots,\texttt{m}(\texttt{b}+\texttt{r}-\texttt{1})\}.$$

  Here r is the number of blocks the client watches per time unit.

- **`Fast forward:`** Suppose ffs is a positive integer called the "fast forward step."

$$\texttt{data}(i,t) \;=\; \{(\texttt{m},\texttt{b}+\texttt{i}\times\mathsf{ffs})\,|\,\texttt{i}<\texttt{r}\,\&\,(\texttt{b}+\texttt{i}\times\mathsf{ffs})<\mathsf{bnum}(\texttt{m})\}.$$

- **`Rewind:`** Suppose rws is the rewind step.

$$\texttt{data}(i,t) \;=\; \{(\texttt{m},\texttt{b}-\texttt{i}\times\mathsf{rws})\,|\,\texttt{i}<\texttt{r}\,\&\,(\texttt{b}-\texttt{i}\times\mathsf{ffs})>\texttt{1}\}.$$

- **`Pause:`** If when the user pauses, he was watching block b of movie m, then:

$$\texttt{data}(i,t) \;=\; \{(m,b)\}.$$

Client Request

- Reformulate `data`(i,t) as a single quadruple

$$\texttt{data}(i,t) \;=\; (m, b, len, step)$$

- This means that client C_i wishes to view the following blocks of movie m:

$$b, (b + step), (b + 2 \times step), \ldots, (b + (len - 1) \times step).$$

- `Play - normal viewing:` In this case, $step = 1$.
- `Pause:` In this case, $step = 0$.
- `Fast Forward with speed ffs:` In this case, $step = \mathsf{ffs}$.
- `Rewind at speed rws:` In this case, $step = -\mathsf{rws}$.

Algorithm to Support VCR-Functionality (Sketch)

- When a set of events (transactions) occur, each of these transactions must have an associated priority.
- Initial priority assignments.

Transaction	Priority
Exiting client	5
Continuing Client - normal viewing	4
Continuing Client - fast forwarding	4
Continuing Client - rewind	4
Continuing Client - pause	4
New (entering) client	2

- **Splitting:** This causes a user's transaction to be split into two or more pieces (called **twins**). Transaction is satisfied iff both twins can be satisfied.
- **Switching:** Causes a user's transaction (or its descendent subtransactions) to be switched from the server that was originally handling the request, to another server.

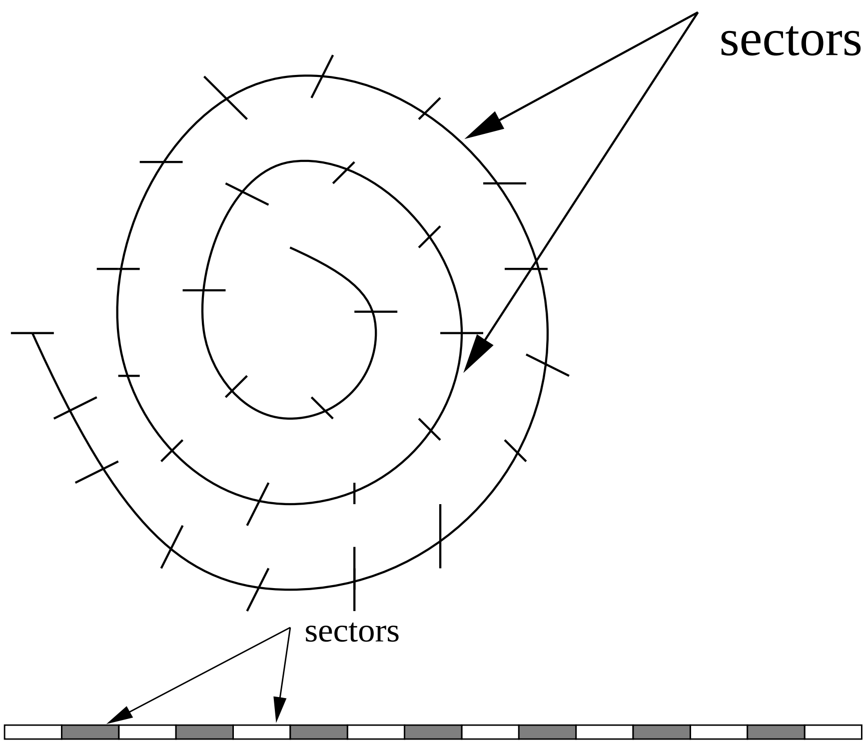
Algorithm to Support VCR-Functionality (Sketch)

- Algorithm considers clients in decreasing order of priority.
- Note that exiting clients have top priority (as they free up resources) and continuing clients have high prioriity too.
- Satisfy clients in decreasing order of priority.
- If a client request cannot be satisfied, and it cannot be switched to another server, then split it, reinsert into client list with slightly lower priority (3 for split contuining clients).
- New client's priority increased if not served in the loop, but can never reach 3.

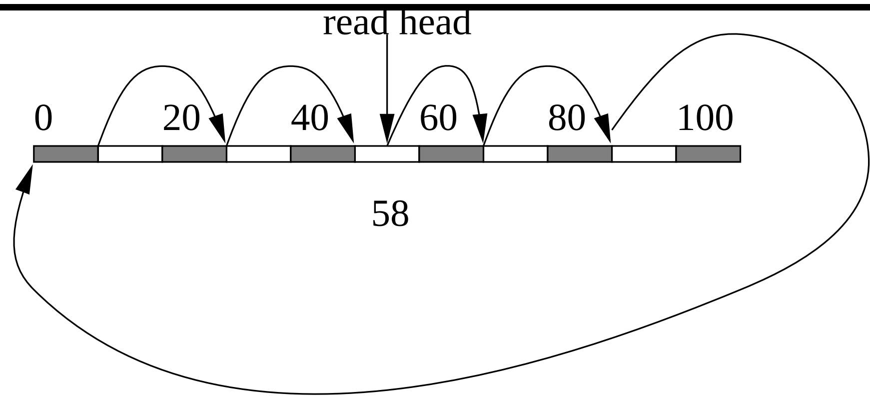
Retrieving Multimedia Data from CD-ROMs

Retrieving Multimedia Data from CD-ROMs

- CD-ROM driver typically contains one platter.
- The CD-ROM contains a single spiral track, that is traversed by the read head.
- Spiral track is divided up into equal sized sectors.
- hus, unlike a disk drive system where the disk head moves at a constant angular velocity, in the case of a CD-ROM based system, the disk head moves at a constant linear velocity across these tracks.



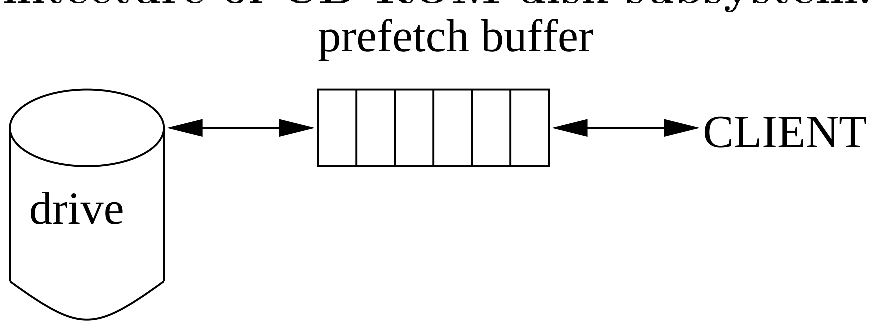
Example



- A particular CD-ROM contains 100 sectors.
- The read head is currently at location 58.
- A client wishes to read the sectors 10,30,50,70,90.
- This client has enough buffer to accommodate 3 sectors.
- **Possibility 1:** CD-ROM reads sector 70 first, then sector 90, then sector 10, then sector 30, and then sector 50.
- Most CD-ROM drivers do not allow this possibility.
- **Possibility 2:** Reset the disk head to point to zero, and then move the head so that sectors 10,30,50 get buffered. We then consume 10 and buffer 70. Next we consume sector 30 and buffer 90. Next, we consume sector 50,70 and 90.

Reading from a CD-ROM

- Reading is done in *rounds*.
- Each round starts with the read head at location 1.
- In any given round, we attempt to read a sorted (in ascending order of sector number) set of sectors.
- **Architecture of CD-ROM disk subsystem:**



Buffer Requirements

Symbol	Description	Units
ss	sector size	bytes
bw_d	bandwidth of disk to prefetch buffer	bytes/sec
dcr	decompression rate	bytes/sec
cr	compression ratio	integer
$cons$	consumption rate of client	bytes/sec
sk	average seek time	sec
t_{fill}	buffer filling time	sec

- Need to ennsure two properties:
  - **Continuity of playback:** The client should be able to read data from the buffer without any interruption.
  - **Buffer utilization:** At no time should the buffer get over-written.
- Suppose we know all the quantities in the above table, and want to determine buffer size.

Buffer Requirements

- If cr is greater than the rate at which the buffer is filled, then it is possible that at some time, the buffer is empty, but the client is trying to read something.
- If cr is less than the rate at which the buffer is filled, then it is possible that data is being written into the buffer too fast, leading to the possibility that the buffer is over-written.
- In t_{fill} seconds, the server can read

  $$(t_{fill} \times bw_d) \text{ bytes}$$

  of compressed data into the buffer.
- In one second, the client can consume

  $$cons \text{ bytes}$$

  of *uncompressed* data.
- One byte of compressed data is equivalent to cr bytes of uncompressed data.
- The maximal amount of uncompressed data consumable by the client in t_{fill} seconds is given by the equation:

  $$t_{fill} = (\delta \times t_{fill}) + \frac{\delta \times t_{fill} \times dcr \times cr}{cons}$$

  where δ is a real number between 0 and 1, inclusive, denoting the fraction of time within the cycle of t_{fill} seconds, in which the client is decompressing compressed data.

- Solving the above equation to eliminate the variable δ, we get:

$$\begin{aligned} t_{fill} &= (\delta \times t_{fill}) + \frac{\delta \times t_{fill} \times dcr \times cr}{cons}, \text{ i.e.} \\ 1 &= \delta \times (1 + \frac{dcr \times cr}{cons})\text{i.e.} \\ \delta &= \frac{cons}{cons + dcr \times cr}. \end{aligned}$$

- Therefore, in $t_{fill} + sk$ seconds, the server can write $(t_{fill} - sk) \times bw_d$ bytes (of uncompressed data) into the buffer, and the client can consume $\frac{\delta \times t_{fill} \times dcr \times cr}{bw_c}$ bytes (of uncompressed data).

- We want

$$\begin{aligned} (t_{fill} - sk) \times bw_d &= \frac{\delta \times t_{fill} \times dcr \times cr}{bw_c}. \\ &= \frac{cons}{cons + dcr \times cr} \times \frac{t_{fill} \times dcr \times cr}{cons}. \\ &= \frac{t_{fill} \times dcr \times cr}{cons + dcr \times cr} \end{aligned}$$

- This yields that the minimal buffer size needed is $(bw_d \times t_{fill})$. Hence, the minimal amount of buffer needed is given by the expression:

$$bw_d \times \left(\frac{(sk - bw_d) \times (cons + dcr \times cr)}{(bw_d \times cons) + (dcr \times cr) \times (bw_d - 1)} \right).$$

Scheduling Retrieval of Multiple Sectors from CD-ROMs

- Consider a CD-ROM server that receives requests for a set $\{s_1, \ldots, s_k\}$ of sectors.
- Many different algorithms will be studied:
  - First Come First Serve (FCFS)
  - SCAN Algorithm
  - Scan EDF Algorithm

First Come First Serve (FCFS)

- Processes requests according to their arrival time.
- The total seek time taken to serve the entire set of k requests is given by:
$$\frac{\Sigma_{i=1}^{k} abs(s_i - s_{i-1})}{lv}$$
where s_0 is defined to be sector 1, and lv is the linear velocity of the read head, expressed in sectors per second.
- EX: If we consider an FCFS approach to serving requests for sectors $25, 5, 35, 15, 5, 10$ in a case where the angular velocity is 2 sectors per millisecond, then the total time taken is:

$$\begin{aligned} seek &= \frac{abs(25-1) + abs(5-25) + abs(35-5) + abs(15-35) + abs(5-15) + abs(10-5)}{2} \\ &= \frac{24+20+30+20+10+5}{2} \\ &= 54.5 \text{ milliseconds.} \end{aligned}$$

SCAN Algorithm

- We first ollect a set of requests, and then sort the sectors in increasing order of seek distance.
- If the read-head is initially not at the start location, then this might lead to a bidirectional sweep.
- If we reconsider the request for the sectors $25, 5, 35, 15, 5, 10$ when the read head is positioned at sector 1, the SCAN algorithm would first sort these into the order: $5, 10, 15, 25, 35$.
- 5 is read only once (as opposed to FCFS)
- Seek time:

$$
\begin{aligned}
seek &= \frac{(5-1)+(10-5)+(15-10)+(25-15)+(35-25)}{2} \\
&= 17 \quad \text{milliseconds.}
\end{aligned}
$$

Scan EDF Algorithm

- The SCAN-EDF algorithm sorts incoming service requests on two keys: first it sorts them in ascending order of deadline; then in terms of sector number,
- If f a set of service requests have the same deadline, it then processes those service requests using the SCAN algorithm.
- EX: Suppose the table below shows a set of sectors being requested, as well as the deadline by which those sectors need to be read.

Job_Id	Sector	Deadline
1	15	10
2	20	5
3	10	10
4	35	10
5	50	5

- Two groups of requests: G_1 consists of jobs 2 and 5 (both of which have the same deadline, viz. 5) and G_2 contains jobs 1,3,4 (all of which have the same deadline, viz. 10).
- G_1 is serviced first, as it has the earlier deadline, and G_2 is serviced later.
- The servicing of G_1 causes sector 20 to be read first and sector 50 next (as the SCAN algorithm is applied within a group).

The servicing of G_2 causes sectors 10,15,35 to be read in that order. Thus, the order in which the read-head reads data is

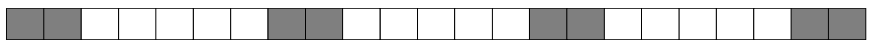
$$20, 50, 10, 15, 35.$$

Placement of Files on a CD-ROM

- A real time file f is a triple (ℓ_f, b_f, p_f) where:
  - $\ell_f \geq 0$ is an integer called the *length* of the file. Intuitively, file f is broken down into ℓ_f "blocks." Different blocks in an RTF may be stored at dispersed locations on the CD-ROM.
  - $b_f \geq 1$ is an integer called the *block size* of the file. Intuitively, b_f is the number of sectors contained within a block. All these sectors are assumed to be contiguous, and in fact, all sectors of a given block are stored in contiguous sectors of the CD-ROM.
  - $p_f \geq 0$ is an integer called the *period* of file f. Intuitively, p_f specifies the distance (in sectors) between the first sector of two consecutive blocks of a real time file.

Real-time file layout

- EX: Consider a real time file, f_1 described by the triple $(4, 2, 7)$. This means that file f_1 has 4 blocks in it, and that each block contains 2 sectors of data.



- Given a real time file f, we define the set of sectors occupied by the i'th block of file f as:

$$occ_i(f) = \{j \mid st(f) + (i-1) \times p_f \leq j \leq st(f) + (i-1) \times p_f + b_f - 1\}.$$

  As there are ℓ_f blocks in file f, it follows that the sectors occupied by file f as a whole is given by the expression:

$$\begin{aligned} occ(f) &= \bigcup_{i=1}^{\ell_f} occ_i(f) \\ &= \bigcup_{i=1}^{\ell_f} \{j \mid st(f) + (i-1) \times p_f \leq j \leq st(f) + (i-1) \times p_f + b_f - 1\}. \end{aligned}$$

- If $st(f) = 3$ in our example, then

$$\begin{aligned} occ_1(f_1) &= \{j \mid 3 \leq 3 + 4 - 1\} \\ &= \{3, 4, 5, 6\}. \\ occ_2(f_1) &= \{j \mid 3 + 7 \leq j \leq 3 + 7 + 4 - 1\} \\ &= \{10, 11, 12, 13\}. \\ occ_3 &= \{j \mid 3 + 2 \times 7 \leq j \leq 3 + 3 \times 7 + 4 - 1\} \\ &= \{17, 18, 19, 20\}. \end{aligned}$$

Real-time file layout

- Suppose we have a CD-ROM containing N sectors, numbered 1 through N.
- Suppose we have a set $\mathcal{F}$ of real time files.
- The start assignment problem is the problem of finding a function $st : \mathcal{F} \rightarrow \{1, \ldots, N\}$ such that:

  Non-Collision Axiom: For all $f_i, f_j \in \mathcal{F}$, $f_i \neq f_j \Rightarrow occ(f_i) \cap occ(f_j) = \emptyset$. If such a function st exists, then it is called a *placement function.*

- SAP tries to assign a start location to each file in such a way that no sector on the CD-ROM contains data belonging to two or more files.
- EX: Suppose we consider two simple real time files, f_1, f_2 characterized by the triples $(2, 2, 5)$ and $(3, 1, 4)$, respectively, and our CD-ROM has 10 sectors labeled 1 through 10. Then the following is a valid start assignment :

$$\begin{aligned} st(f_1) &= 2. \\ st(f_2) &= 1. \end{aligned}$$

  Note that the function st satisfies the non-collision axiom because

$$occ(f_1) = \{2, 3, 7, 8\}.$$

$$occ(f_2) = \{1, 5, 9\}$$

- Korst and Pronk prove that SAP is NP-complete.

Placing 2 Real-Time Files

Algorithm 8 *Placement(f_1, f_2)*

(⋆ $f_1 : (\ell_1, b_1, p_1)$, $f_2 : (\ell_2, b_2, p_2)$ ⋆)
maxstart1 $= N - (b_1 - 1) \times p_1 - (b_1 - 1)$;
maxstart2 $= N - (b_2 - 1) \times p_2 - (b_2 - 1)$;
nogo = *maxstart1* $\leq 0 \vee$ *maxstart2* ≤ 0;
if *nogo* **then Return** *"No Solution"*. **Halt.**
for $i = 1$ **to** *maxstart1* **do**
{
  $st(f_1) = i$;
  for $j = 1$ **to** *maxstart2* **do**
  {
    $st(f_2) = j$;
    if $occ(f_1) \cap occ(f_2) = \emptyset$ **then**
    Return $st(f_1) = i, st(f_2) = j$. **Halt.**
  }
}
Return *"No Solution"*. **Halt.**
end

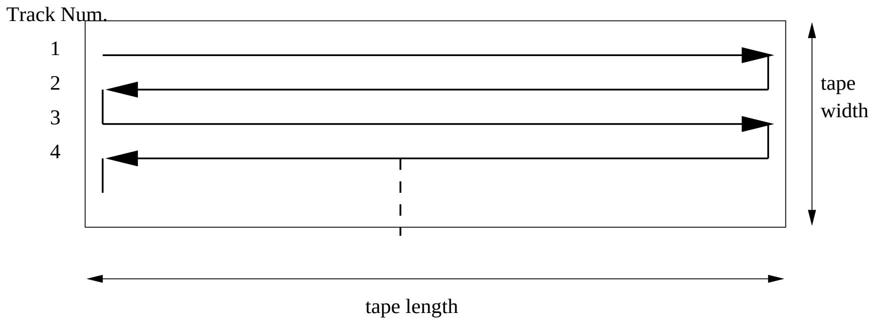
Tape Recording Mechanisms

- Three basic methods are used to store data onto tapes:
  - Serpentine recordings,
  - Helical recordings,
  - Transverse recordings.
- In multimedia applications, there is often a *Robotic Tape Library* and a fixed set of *Players*. The robotic arm reaches into the library, retrieves a requested tape, and inserts it into an available player for playback.
- We will study all the above aspects.

Serpentine Tapes

- Tape contains several *tracks* that are parallel to the length of the tape.
- Each track has a "track number" and a linear set of tape "blocks."
- When reading:
  - The tape is first "rolled" forward (i.e. in the left to right direction) and the read-head of the tape driver is positioned over track 1.
  - When we reach the end of track 1, the read head gets repositioned over track 2, and we read the contents of track 2 moving from right to left.
  - When we reach the end of track 2, the read head gets re-positioned over track 3, and we read from left to right.
  - The process continues till we reach the end of the the tape.

Serpentine Tapes



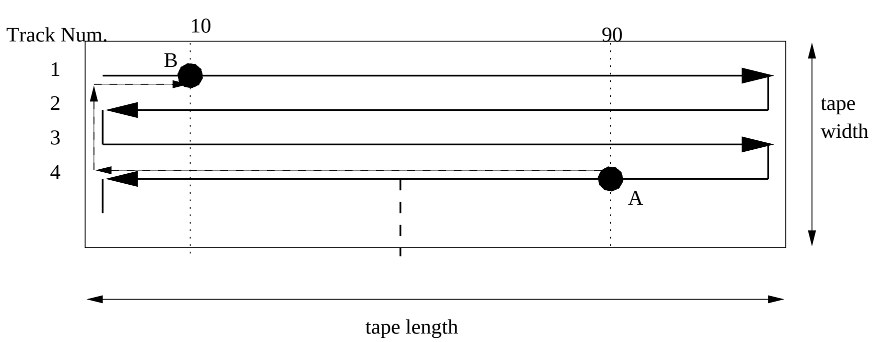
Serpentine Tape Traversal

Algorithm 9 *Tape_traverse(n)*

```
i = 1; block = 1;
while i ≤ n do
{
   end_of_track = false;
   while ¬end_of_track do
   {
      if i mod 2 = 1 then block = block + 1
      else block = block − 1;
      Read block;
   }
   i = i + 1: (⋆ shift tracks ⋆)
   Position read head over track i;
}
end
```

Example

- Suppose the read head is currently positioned over track 4, and we are reading block 90 (shown as A in the figure) on this track.
- Suppose we wish to read block B of track 1 now.
- Then the tape must be rewound (as shown by the dotted lines) to the beginning of track 4, then the read head must be switched to track 1 (jumping tracks 2 and 3) and finally, we move the tape ahead to block B. (Many systems do not support such jumps).



Alternative 1: Reading from a Serpentine Tape

1. Rewind tape to the left till the read head is positioned over block 1 of track t_1. This requires traversing $(b_1 - 1)$ blocks.
2. Then re-position the read head to track t_2, jumping $\text{abs}(t_1 - t_2)$ tracks. This requires a jump over $\text{abs}(t_1 - t_2)$ tracks.
3. Roll tape forward till the read head is positioned over block b_2. This requires traversing $(b_2 - 1)$ blocks.

- Let ff and rew denote the fast forward and rewind speeds (in blocks/sec)
- Let $trkspeed$ denotes the number of tracks that can be jumped per second.
- Then the time τ_1 taken for alternative 1 is given by:

$$\tau_1 = \frac{(b_1 - 1)}{rew} + \frac{\text{abs}(t_1 - t_2)}{trkspeed} + \frac{(b_2 - 1)}{ff}.$$

Alternative 2: Reading from a Serpentine Tape

1. Fast forward tape to the right till the read head is positioned over the last block (denoted $nblock$) of the track. This requires traversing $(nblock - b_1)$ blocks.
2. Then re-position the read head to track t_2, jumping abs$(t_1 - t_2)$ tracks. This requires a jump over abs$(t_1 - t_2)$ tracks.
3. Rewind the tape (moving left) till the read head is positioned over block b_2. This requires traversing $(nblock - b_2)$ blocks.

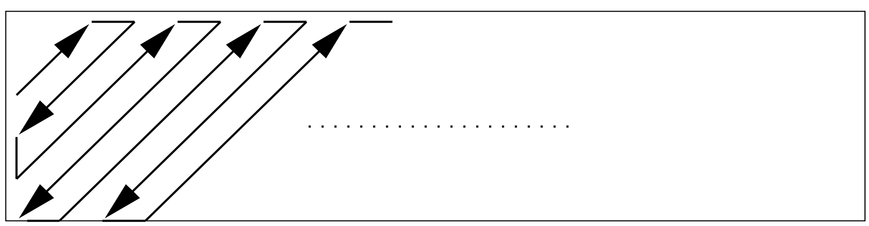
- The time τ_2 required for alternative 2 is given by:

$$\tau_2 = \frac{(nblock - b_1)}{rew} + \frac{\mathrm{abs}(t_1 - t_2)}{trkspeed} + \frac{(nblock - b_2)}{ff}.$$

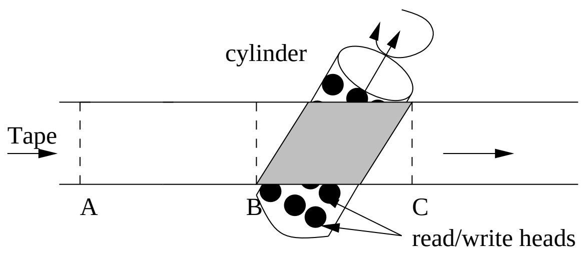
- It is important to note that the above calculations change when jumps over multiple tracks are not allowed.

Helical Tape Recording

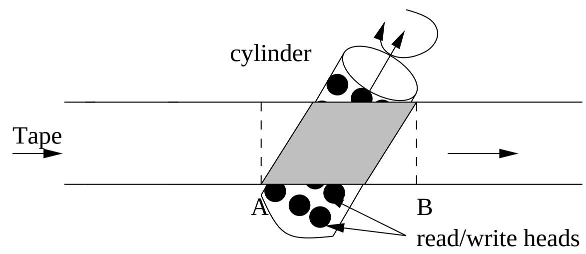
- Tracks are "diagonal" tracks.
- Tape winds around a cylinder in a spiral fashion.
- Read/write heads are embedded in the surface of the cylinder.
- The axis across which the cylinder rotates is somewhat tilted, relative to the tape itself.
- The heads "pass" the linear movement of the tape, different parts of the tape, corresponding to ngular, diagonal tracks.



Example



(a) Reading of tape by read-heads

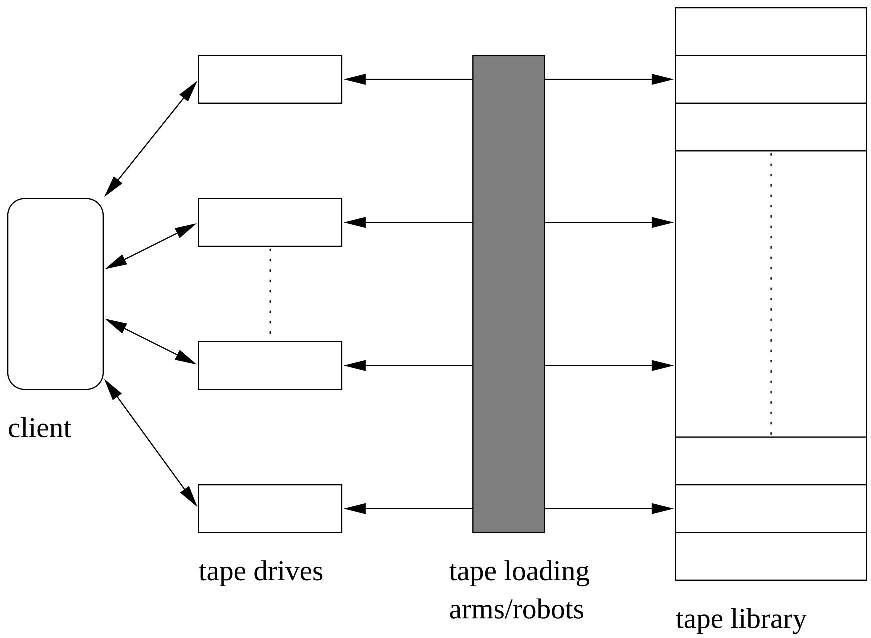


(b) Reading of tape by read-heads after tape movement/cylinder rotation

Handling "Bad" Sectors

- When recording a tape, a two step procedure is followed when we are writing block b:
  - first we write it, and
  - then we immediately read it back.
- If the result of reading back is the same as what was written (this test may be efficiently computed via a "checksum" operation) then it means that the block is okay, otherwise it means there is a problem.
- If there is a problem, then we add this block to a list of "bad blocks" and try to re-write the information onto the next block.
- When reading the tape, the list of "bad blocks" must be taken into account.

Robotic Tape Library Architecture



- Relative cost of obtaining the tape from the "shelf" and loading it into a tape drive is a very expensive, time consuming operation.
- Thus, minimizing such accesses is a key requirement of tape-based storage and retrieval algorithms.

Retrieval from Tape Libraries

- Suppose that we have a set $td_1, \ldots, td_r$ of tape drives, but only one robot arm.
- When client C requests a tape, the following steps are performed, once the robot arm is available to service the request:
  - check if there is a free drive into which the robot can insert the desired tape. If not we must wait.
  - Once such a drive is available, we
    - rewind the tape currently in the drive, then
    - eject it from the drive, then
    - return it to its correct location in the tape library,
    - pick up the requested tape,
    - insert it into the tape drive, and then
    - let the tape driver access the desired blocks for the client.

Algorithm ape_retrieve(tape_id,Set_of_drives)

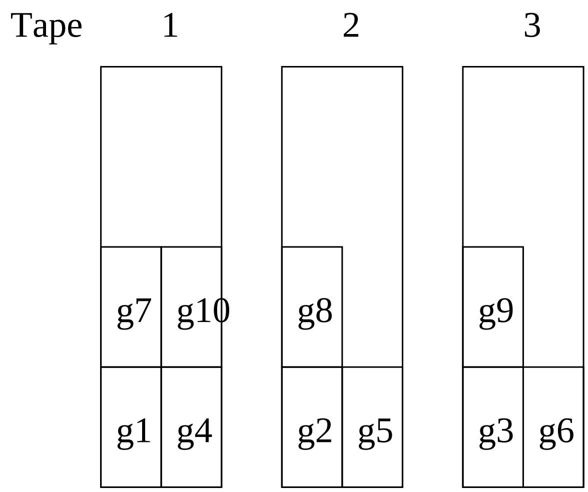
Algorithm 10 *Tape_retrieve(tape_id,Set_of_drives)*

```
indrive = false;
if all drives in Set_of_drives are busy then
   { wait for time δ;
   Call Tape_retrieve(tape_id,Set_of_drives) again
   };
else
   {
      { If there is a free tape drive with tape
         tape_id in it, then set TD to this t ape drive;
         and set indrive to true; else set TD to a ny free tape drive
      };
      if (¬ indrive) then
   {
      if there is a tape τ in TD then
      { rewind τ;
         eject τ;
         pick up τ with the robot arm;
         return τ to the tape library;
      };
      Pick up the requested tape, tape_id;
      Insert tape_id into TD;
   };
   };
Fast forward tape_id to the desired starting block in drive TD;
Playback requested blocks.
end
```

Striping

- *Granule Size:* First, we must divide the object o up into equal sized granules. *What is the impact of granule size on retrieval efficiency?*
- *Stripe Width:* Next, we must determine how many tapes the object o will be striped across. *What is the impact of stripe width on retrieval efficiency?*
- Example:
  - media object o of size 200 MB
  - granule size 20 MB
  - stripe width 3

Example



Stripe Table

- associates with each object o:
  - its size
  - its granule size
  - its stripe width
  - the set of tapes on which it (i.e. the object o) is stored.
- EXAMPLE:

Object	Size	Granule Size	Stripe Width	List of Tapes
o_1	200	40	3	t_1, t_2, t_3
o_1	200	40	3	t_4, t_5, t_6
o_2	100	25	3	t_1, t_2, t_3
o_2	100	20	3	t_4, t_5, t_6
...	...	...	...	...

Striped_tape_retrieve Algorithm

Algorithm 11 *Striped_tape_retrieve(obj)*

SW = S_Table.Stripe_width;
Tapes_avail = $\{$*tape* $|$ *t is not currently being used*$\}$*;*
Tape_sets_needed= $\{$*S_Table.List* $|$ *S_Table.Object = obj*$\}$*;*
while $(\forall \ell \in$ *Tape_sets_needed)* $\ell \not\subseteq$ *Tapes_avail* **do** *wait;*
Sel_list = ℓ *for some* $\ell \in$ *Tape_sets_needed such that*
    $\ell \subseteq$ *Tapes_avail;*
while *SW drives are not available* **do** *wait;*
Let FTD *= the set of all free tape drives;*
  while $FTD \neq \emptyset$ **do**
  {
    while $\ell \neq \emptyset$ **do**
      { *select* $tape_i \in \ell$; $\ell = \ell - \{tape_i\}$;
      if there is a tape drive in FTD *with tape*
      ℓ_i *in it, then set* td_i *to this tape drive,*
      set indrive to true, and set $FTD = FTD - \{td_i\}$;
      };
  if *(¬ indrive)* **then**
  {
    select any tape drive $td_i \in FTD$;
    if *there is a tape* τ *in* td_i **then**   { *rewind* τ;
      eject τ;
      pick up τ *with the robot arm;*
      return τ *to the tape library;*
    };
    Pick up the requested tape, t_i;
    Insert t_i *into* td_i;
    $FTD = FTD - \{td_i\}$;
};
Fast forward tape_id to the desired starting block;
Playback requested blocks.
  }
end

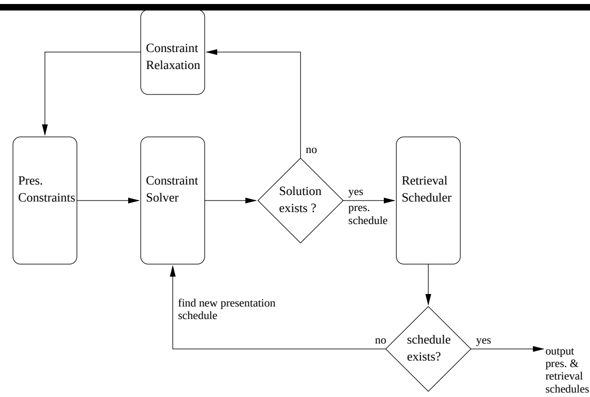
Creating and Delivering Networked Multimedia Presentations

- When creating a multimedia presentation, three basic questions must be answered:
  - *What* objects should be included in the presentation ?
  - *When* should these objects to be presented to the user ?
  - *Where* should the objects appear on the screen ?

  These questions must be answered by the presentation author who may not be a computer scientist.

- Once these have been specified, the Presentation Server must generate a *Retrieval Plan* that allows it to retrieve the objects required for the presentation keeping in mind:
  - when the objects need to be presented
  - bandwidth limitations on the network
  - resource (load, buffer) limitations on the server
  - resource (load, buffer) limitations on the client
  - mismatches between delivery rate and client consumption rate.

Architecture



Creating a Presentation

- Suppose objects $o_1, \ldots, o_n$ are to be presented.
- Temporal constraints specify how the objects should be laid out in time. Example:
  - The presentation of objects 1 and 2 must start at the same time.
  - The termination of objects 2 and 3 must occur at the same time.
  - Object 3 must start at the time object 1 ends.
- Spatial constraints specify how the objects should be laid out in space. Example:
  - Object 1 must be to the left of object 2.
  - Object 1 must be above object 3.

Constraint Language

- **Constants:** Every integer (positive and negative) is a constant.
- **Variables:** Associated with each o_i are two integer variables, s_i (denoting the "start" of o_i) and e_i (denoting the "end" of o_i).
- **Elementary Terms:** Elementary terms are defined inductively as follows:
  1. Every constant is an elementary term.
  2. Every variable is an elementary term.
- **Difference Constraint:** If t_1, t_2 are elementary terms, and c is a constant, then

$$t_1 - t_2 \leq c$$

is a difference constraint.

Example

- $e_1 - s_1 \leq 10$.
  This constraint says that object o_1 must end with 10 time units of the time its presentation starts.

- $s_2 - e_1 \leq 0; e_1 - s_2 \leq 0$.
  These two constraints jointly state that object o_2's presentation starts as soon as object o_1's presentation ends.

- $s_2 - e_1 \leq 3$.
  This constraint says that object o_2's presentation starts within 3 time units of the end of the presentation of object o_1.

Definitions

- A *temporal presentation* is a pair $TP = (O, \mathsf{DC})$ where O is a finite set of objects, and DC is a finite set of difference constraints in the constraint language generated by O.
- Suppose DC is a set of difference constraints over $O = \{o_1, \ldots, o_n\}$ of virtual objects. A *solution* of DC is an assignment of integers to each of the variables, $s_1, \ldots, s_n, e_1, \ldots, e_n$, which makes all the constraints true.
- EX:

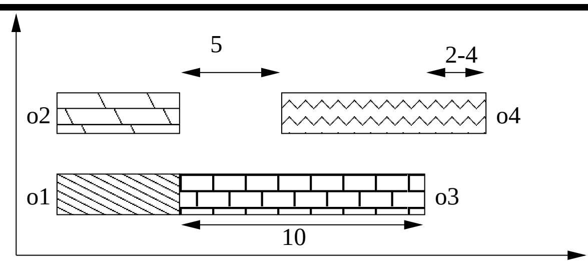
$$\begin{aligned} s_1 - s_2 &= 0. \\ e_1 - e_2 &= 0. \\ s_3 - e_1 &= 0. \\ s_3 - e_2 &= 0. \\ e_3 - s_3 &= 10. \\ s_4 - e_2 &= 5. \\ e_4 - e_3 &\leq 4. \\ e_3 - e_4 &\leq -2. \end{aligned}$$

  The figure on the next page shows these constraints.
- Sets of constraints can have 0, 1 or many solutions. EX:

Solution	s_1	e_1	s_2	e_2	s_3	e_3	s_4	e_4
σ_1	0	10	0	10	10	20	15	22.
σ_2	0	10	0	10	10	20	15	23.
σ_3	0	10	0	10	10	20	15	24.
σ_4	3	10	3	10	10	20	15	22.
σ_5	3	10	3	10	10	20	15	23.
σ_6	3	10	3	10	10	20	15	24.

- A *temporal presentation* $TP = (O, \mathsf{DC})$ is *feasible* iff the set, DC, of difference constraints, has a solution σ. In this case, σ is said to be a *schedule* for TP.

Constraints Viewed Graphically



Definition

- The *start* and *end* of σ, denoted $start(\sigma)$ and $end(\sigma)$, are defined to be:

$$
\begin{aligned}
start(\sigma) &= \min(\{\sigma(s_i)\,|1 \leq i \leq n\}. \\
end(\sigma) &= \max(\{\sigma(e_i)\,|1 \leq i \leq n\}.
\end{aligned}
$$

- We are assuming that the constraints $\{s_i - e_i \leq 0|1 \leq i \leq n\}$ are included in DC.

- Returning to our example:

Solution	Start	End
σ_1	0	22
σ_2	0	23
σ_3	0	24
σ_4	3	22
σ_5	3	23
σ_6	3	24

How should we create presemtation schedules?

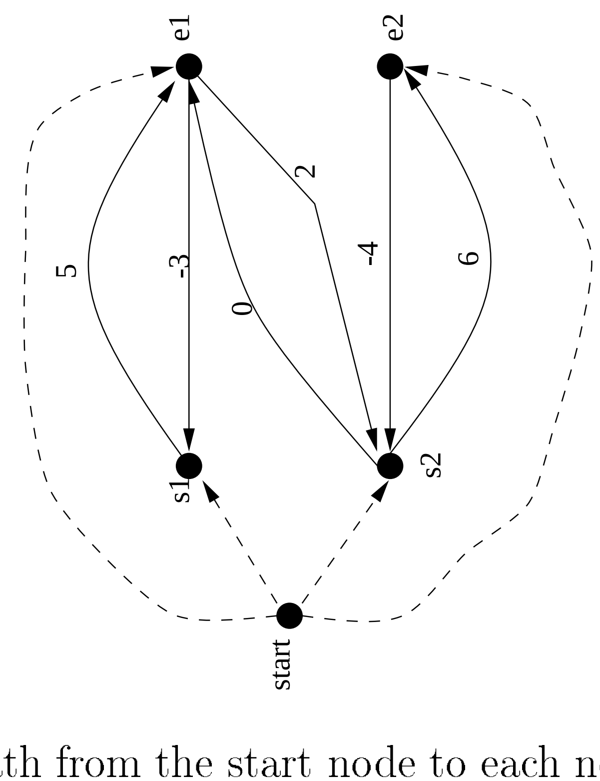
- There is a classic algorithm called the Bellman Ford Algorithm.
- Takes as input, a set S of difference constraints.
- Convert S to a graph.
- S has a solution iff the graph has no negative cycles.
- A negative cycle is a cycle whose edges sum up to a negative number.

Example

- Constraints:

$$
\begin{aligned}
s_1 - e_1 &\leq -3. \\
e_1 - s_1 &\leq 5. \\
s_2 - e_2 &\leq -4. \\
e_2 - s_2 &\leq 6. \\
e_1 - s_2 &\leq 0. \\
s_2 - e_1 &\leq 2.
\end{aligned}
$$

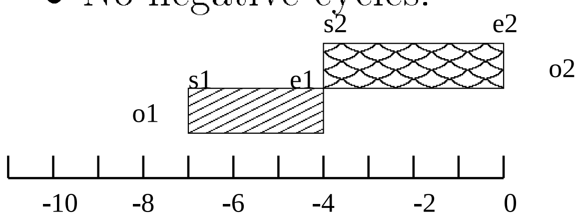
- Graph:

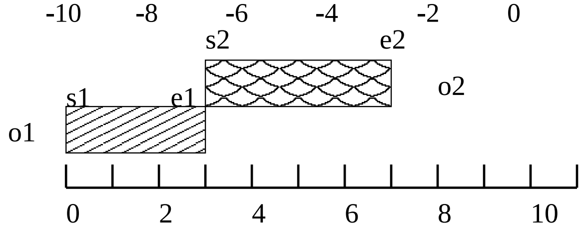


- Shortest path from the start node to each node:

Node N	Shortest Path (S.P.) from **start** to N	Cost of S.P.
s_1	$start\stackrel{0}{\rightarrow}e_2\stackrel{-4}{\rightarrow}s_2\stackrel{0}{\rightarrow}e_1\stackrel{-3}{\rightarrow}s_1$	-7.
e_1	$start\stackrel{0}{\rightarrow}e_2\stackrel{-4}{\rightarrow}s_2\stackrel{0}{\rightarrow}e_1$	-4
s_2	$start\stackrel{0}{\rightarrow}e_2\stackrel{-4}{\rightarrow}s_2$	-4
e_2	$start\stackrel{0}{\rightarrow}e_2$	0

- No negative cycles.





Theorems

- Let the cost of a path be the sum of the costs of edges along the path.
- A *cycle* in $\mathcal{G}$ is a sequence of nodes $v_1, \ldots, v_k$ such that $v_k = v_1$ and for all $1 \leq j < k$, (v_j, v_{j+1}) is an edge in the graph.
- $v_1, \ldots, v_k$ is said to be a *negative cycle* in graph $\mathcal{G}$ iff $\Sigma_{j=1}^{k-1} \wp(v_j, v_{j+1})$ is a negative number.
- A set DC of difference constraints has no solution iff $\mathcal{G}_{\mathsf{DC}}$ has a negative cycle.
- Suppose a graph has no negative cycles. Then the cost of the shortest path from the start node to that node (s_i or e_i) directly allows us to get a solution to the original set of difference constraints.

Approach

The Bellman Ford algorithm associates with each node N in $\mathcal{G}_{\mathsf{DC}}$, the following two fields:

- $Bestval$: This specifies the cheapest path from the start node to the node N, that has been discovered thus far.
- $Bestpar$: $Bestpar(N)$ specifies the immediate predecessor of node N along the best path from the start node to node N, that we have found thus far.

Initialization

Algorithm 12 *Initialize(V, E);*

(⋆ $n = card(V); m = card(E);$ ⋆)
for $i = 1$ **to** n **do**
{
  $Bestval(v_i) = \infty$;
  $Bestpar(v_i) =$ `NIL`
}
end

Refinement

Takes two nodes v_i, v_j as input, and determines if the shortest path from the start node to v_j is made cheaper by "going through" v_i.

Algorithm 13 *Refine*(v_i, v_j);

$new = Bestval(v_i) + \omega(v_i, v_j)$;
if $new < Bestval(v_j)$ **then**
{
  $Bestval(v_j) = new$;
  $Bestpar(v_j) = v_i$
}
end

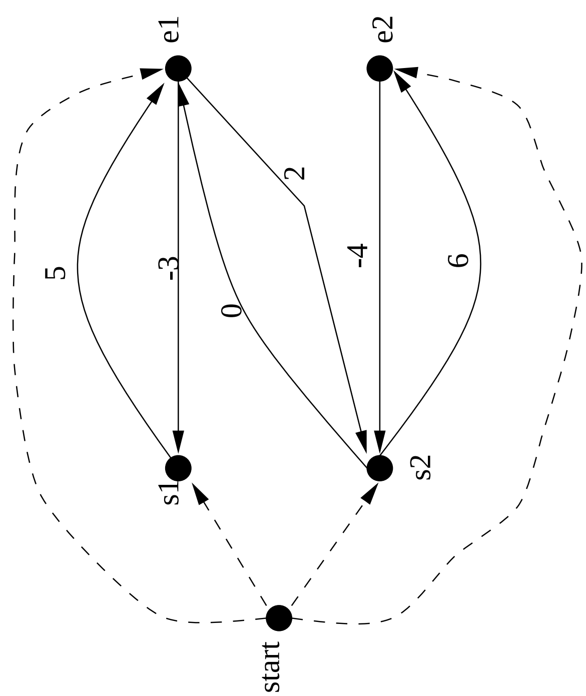
Main Algorithm

Algorithm 14 *Bellman_Ford*(V, E, ω);

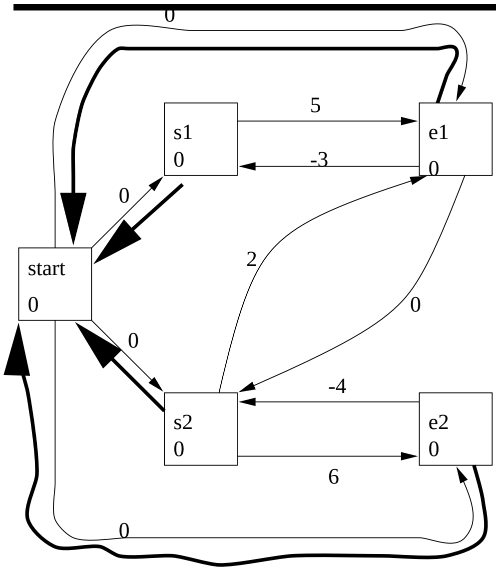
$n = card(V); m = card(E)$;
($\star$ *we assume* $V = \{v_1, \ldots, v_n\}$ $\star$);
($\star$ *we assume* $E = \{(v_1^1, v_1^2), \ldots, (v_m^1, v_m^2)\}$ $\star$);
Initialize(V, E);
for $i = 1$ **to** $(n-1)$ **do**
  for $j = 1$ **to** m **do**
    Refine(v_j^1, v_j^2);
  end ($\star$ *inner for*$\star$);
end ($\star$ *outer for*$\star$);
for $j = 1$ *to* m **do**
  if $Bestval(v_j^2) > Bestval(v_j^1) + \omega(v_j^1, v_j^2)$ **then**
    Return *error and* **Halt**;
end ($\star$ *for*$\star$);
for $i = 1$ **to** n **do**
  Return $v_i.Name = v_i.Bestval$.
end ($\star$ *for*$\star$);
end ($\star$ *algo* $\star$

Examples

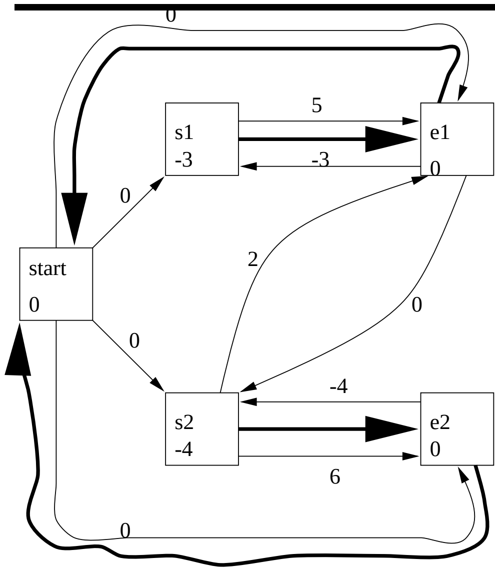
$$
\begin{aligned}
s_1 - e_1 &\leq -3. \\
e_1 - s_1 &\leq 5. \\
s_2 - e_2 &\leq -4. \\
e_2 - s_2 &\leq 6. \\
e_1 - s_2 &\leq 0. \\
s_2 - e_1 &\leq 2.
\end{aligned}
$$



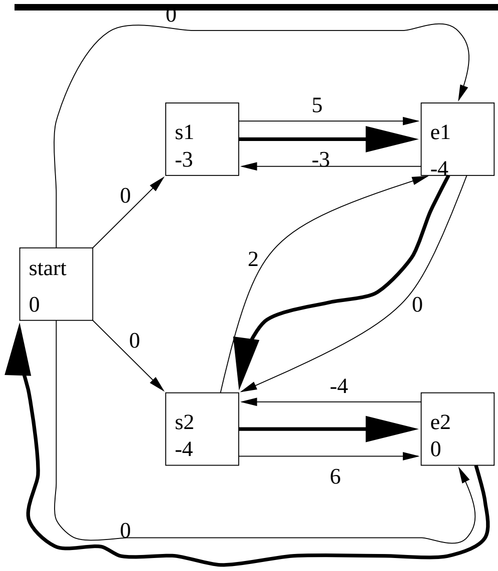
Example, Continued



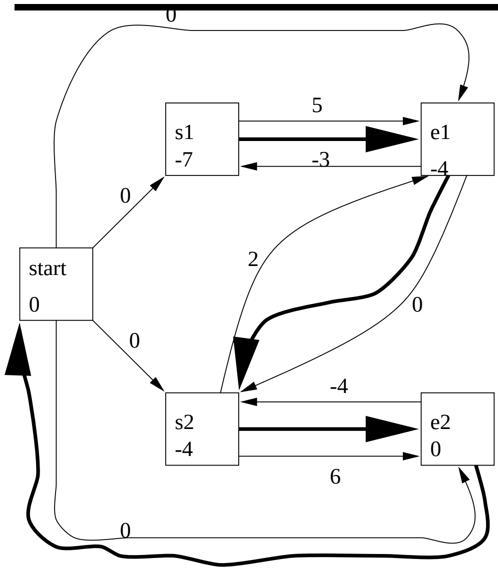
Example, Continued



Example, Continued



Example, Continued



Spatial Constraints

Given a set $\{\mathsf{vo}_1, \ldots, \mathsf{vo}_n\}$ of virtual objects, we associate, with each object vo_i, the following variables:

- W_i: width of the window in which object vo_i is shown;
- H_i: height of the window in which object vo_i is shown;
- X_i: x-coordinate of the lower left corner of the window in which object vo_i is shown;
- Y_i: y-coordinate of the lower left corner of the window in which object vo_i is shown.
- R_i: This variable is equal to $(W_i + X_i)$ and denotes the right vertical edge of the window containing vo_i.
- U_i: This variable is equal to $(H_i + Y_i)$ and denotes the right vertical edge of the window containing vo_i.

Difference constraints using these variables may be specified and solved in exactly that same way as for temporal constraints.

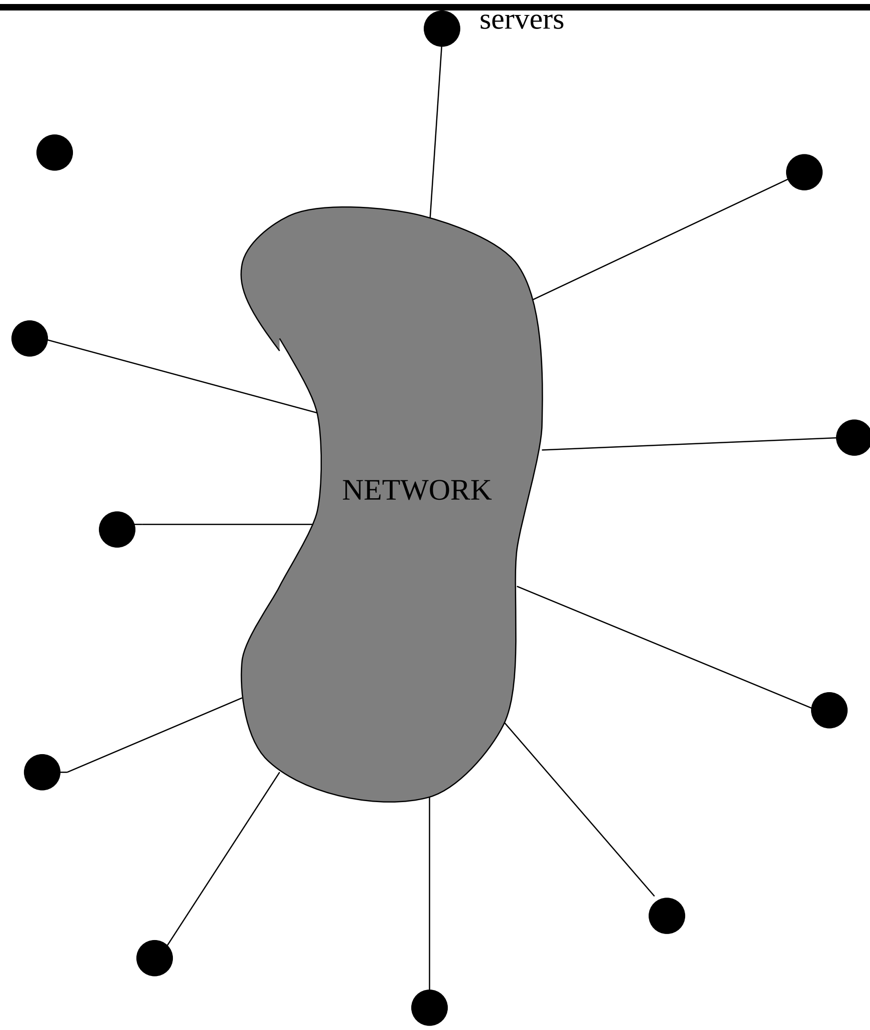
Example

Relationship	Constraint
vo_i is to the left of vo_j	$R_i - R_j \leq 0.$
vo_i is to the right of vo_j	$R_j - R_i \leq 0.$
vo_i is above vo_j	$U_j - Y_i \leq 0$
vo_i is below vo_j	$Y_i - U_j \leq 0.$

Distributed Media Servers

- We have seen how to create *Presentation Schedules.*
- A presentation schedule specifies *when* the display of media objects should start, and when they should end.
- A given set of presentation constraints may have zero, one or more presentation schedules that satisfy the presentation constraints.
- Nt all presentation schedules are *deliverable.*
- To how an object o at time t, the server retrieving o must obtain commitments of resources from several sources.
- These committments include buffer resources, bandwidth resources, and load guarantees.
- Retrieval scheduling takes a presentation schedule as input, and tries to create a retrieval schedule that ensures that objects are retrieved into the client's buffer before the client's associated presentation schedule requires the object to be displayed.

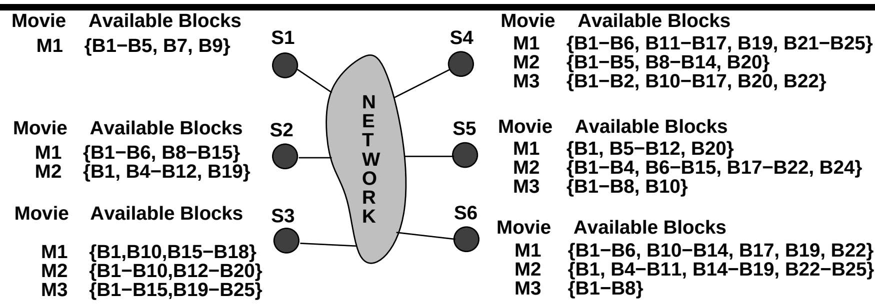
Retrieval Schedules: Architecture



Distributed Multimedia Server Systems

- A *customer* interested in retrieving data at or by a given time contacts his/her local server
- This local originating server is then charged with the responsibility of creating a Retrieval Schedule by interacting with other servers on the network.
- Each server on the network has an associated body of media objects stored on its local disk.
- A *placement mapping* is a mapping, $\wp$ that takes as input,
  1. a movie $m_i \in \mathcal{MOVIE}$ and
  2. a block number, $1 \leq b \leq \mathsf{bnum}(m_i)$, and returns as output, a subset of V. Here, V is the set of all servers available in the DMSS system.

Example Placement Mapping



Server/client Parameters

- Server Buffer Size
- Customer Consumption Rate
- Customer Buffer Size
- Server-server Bandwidth
- Customer-Server Bandwidth

Data structures: Commitment Record List

- Commitment Record List: Each server maintains a commitment record list, specifying what commitments it has made. When a request for services is received by the server, the commitment record list is consulted to determine if the service request can be satisfied. If so (i.e. when a new commitment is made), the commitment record list is updated to reflect the new commitment.

-

BegCom	This specifies the start time of a commitment.
FinCom	This specifies the finish time of a commitment.
Client	This could either be a customer, or another server to whom a commitment is being made.
Movie	This specifies what movie forms part of the commitment.
BlockSt	This specifies the starting block of the movie.
BlockEnd	This specifies the ending block of the movie associated with this commitment.
BWCom	This specifies the amount of bandwidth committed to this commitment.

Example Committtment Record List

- EX:

BegCom	FinCom	Client	Movie	BlockSt	BlockEnd	BWCom
5	15	John	The Rope	B5	B35	0.5 MB/sec
5	10	s_4	The Abyss	B25	B45	0.25 MB/sec
15	25	s_5	Dracula	B50	B70	0.5 MB/sec
...	...	...	...	...	...	...

Data structures: Retrieval Record

- Given any customer request r, the originating server for that customer creates a set of retrieval records for it. The retrieval records specify the retrieval plan put together by the originating server.
- Retrieval record format:

1	**Orig**	Specifies the server that originated the request.
2	**Target**	Specifies the server that will satisfy the request.
3	**Movie**	Specifies the movie-id associated with the request.
4	**Start**	Specifies the first movie block being requested.
5	**End**	Specifies the last block being requested.
6	**Reqtime**	This is the value at which block request is initiated.
7	**ConOK**	This is the time at which the connection is successfully made.
8	**BWAssign**	This is the bandwidth assigned to the request by the target server.
9	**DelivSt**	This is the time at which delivery starts.

- (Continued on next page).

Retrieval Record Format, Continued

10	**DelivEnd**	For each block b_w where $r.Start \leq w \leq r.End$, $r.DelivEnd[w] = r.DelivSt + \frac{(w - r.Start + 1) \times \texttt{bsize}}{r.BWAssign}$
11	**CustShipSt**	For each block b_w where $r.Start \leq w \leq r.End$, $r.CustShipSt[w] \geq r.DelivEnd[w]$
12	**CustShipEnd**	For each block b_w where $r.Start \leq w \leq r.End$, $r.CustShipEnd[w] = r.CustShipSt[w] + \frac{\texttt{bsize}}{\texttt{bw}(r.Orig, C)}$
13	**CustConsStart**	For each block b_w where $r.Start \leq w \leq r.End$, $r.CustConsStart[w] \geq r.CustShipEnd[w]$
14	**CustConsEnd**	For each block b_w where $r.Start \leq w \leq r.End$, $r.CustConsEnd[w] = r.CustConsStart[w] + \frac{\texttt{bsize}}{ccr(C)}$

Retrieval Plans

- A *retrieval plan* is a finite sequence $r_1, \ldots, r_n$ of retrieval records such that for all $1 \leq i < n$, r_i.**BlockEnd** $+ 1 =$ r_{i+1}.**BlockSt**.
- The retrieval plan must satisfy the following constraints:
  - **Continuity of Playback:** At all times t such that r_1.**CustConsStart** $\leq t \leq$ r_n.**CustConsEnd**, there must exist exactly one block b_i of the movie such that b_i is being viewed by the customer.
  - **Server-Server Bandwidth Constraint:** At all times t such that r_1.**DelivSt** $\leq t \leq$ r_1.**DelivEnd**, and for all network connections between 2 servers (s_1, s_2), the total assignment of bandwidth to jobs using the channel (s_1, s_2) should be less than or equal to the total physical bandwidth of the channel.
  - **Server-Customer Bandwidth Constraint:** At all times t such that r_1.**CustShipSt** $\leq t \leq$ r_n.**CustConsEnd**, the total assignment of bandwidth to jobs using the channel between the customer and the originating server should be less than or equal to the total physical bandwidth of the channel.
  - **Customer Buffer Constraint:** At all times t such that r_1.**CustShipSt** $\leq t \leq$ r_n.**CustConsEnd**, the total number of blocks that have been shipped by the originating server to the customer, but that have not yet

been consumed by the customer, should occupy space less than or equal to the total buffer space available at the customer's machine.

- **Originating Server Buffer Constraint:** At all times t such that $r_1.$**DelivEnd** $\leq t \leq r_n.$**CustShipEnd**, the total number of blocks that have been shipped by a remote server to the originating server, but that have not yet been shipped to the customer, should occupy space less than or equal to the total buffer space available at the customer's machine.

Optimal Retrieval Plans

Many possible objective functions, but we consider two.

- *Minimizing customer wait time* RP is said to be *wait-minimal* iff there is no other retrieval plan RP' that can satisfy the request such that RP' has a strictly smaller wait time than RP.
- *Minimizing the access bandwidth* RP is said to be *access-bandwidth* minimal iff the sum of the number of disk accesses and the number of network accesses is minimal, i.e. there is no other retrieval plan RP' such that the sum of the number of disk accesses and the number of network accesses made by RP' is strictly less than the sum of the number of disk accesses and the number of network accesses made by RP.
- Many other objective functions may also be used, and the algorithm we provide below takes an objective function as input, and always produces an optimal retrieval plan w.r.t. that objective function as output.

Basic idea for creating Retrieval Plans

- Initial client retrieval request is a triple, $Req(C) = (m, b_1, b_2)$. Intuitively, $Req(C) = (m, b_1, b_2)$ denotes the request, by user C, for all blocks b of movie m such that $b_1 \leq b \leq b_2$.
- The *Problem* of creating retrieval plans has an IMPLICIT RETRIEVAL TREE associated with it.
- Each node of the tree is labeled with a segment of blocks to be retrieved. Thus, the root of the tree is labeled with (b_1, b_2).
- If node N is attempting to get blocks (b'_1, b'_2) and there is a server that can deliver (b'_1, b'_3) for $b'_3 \leq b'_2)$, then there is an edge labeled with the server's name (plus some other parameters) and the child is labeled with (b'_3+1, b'_2) indicating that these are yet to be accomplished.
- The edge is also labeled with a set of time-bandwidth pairs – intuitively, if (t, bw) is in this set, it means that it is possible to deliver the set of blocks starting at time t and bandwidth bw. Bandwidth and buffer constraints taken into account here.
- Details of the algorithm are in the book, pages 384–385.