CS541 Spring 2012

Assignment 3 Sample Solution

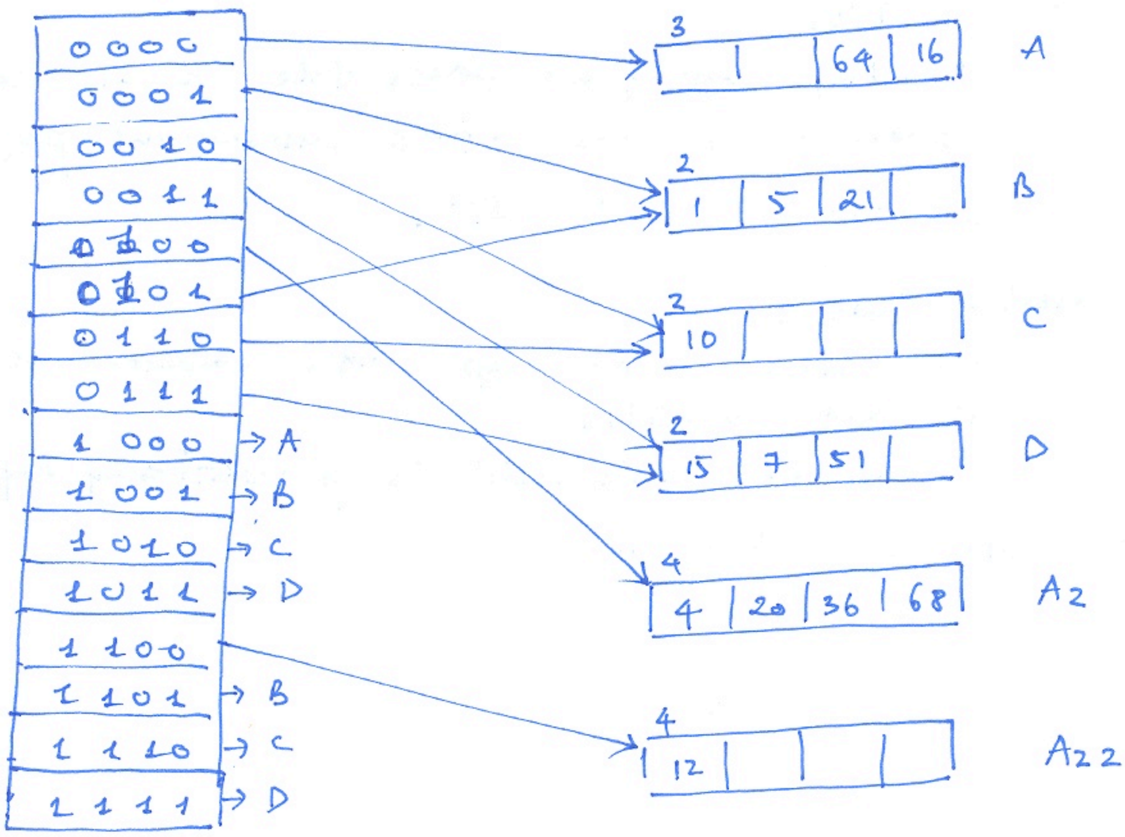
Duong Nguyen

Textbook Exercises

1. (Ex 11.1)

1, The last entry that was inserted into the extendible hash index can be any of the entries. For any entry, we can find a sequence of inserts with that entry at last that produce the configuration.
The last entry inserted doesn't cause a split because:
if it belongs to buckets B, C, D, those buckets still have enough free slots. On the other hand, those buckets don't have split image.
if it belongs to bucket A or A_2, then in order to be the result of a split, the total entries in A and A_2 must be 5, not 6.

2, Insert 68 causes a split



2. (Ex 12.2) : Just consider I/O cost.

1, $\sigma_{a<50.000}(R)$

Sorted File:

Locate the (first) page containing the first record satisfying the condition takes a constant time of computation.

Load that page takes 1 I/O

Then we load subsequent pages take 1 I/O for each page.

we load ~~into~~ in total 5000 pages → 5000 I/Os

Clustered B+ tree:

Locate the first data page takes $\log_F 500.000$ I/O

where F is fan-out of B+ tree

500.000 is the total number of pages

when F = 100 ⇒ locate first page take 3 I/O.

Loading subsequent pages take 1 I/O each page ⇒ we load 4999 subsequent pages.

In total: 5002 I/Os

Linear ~~hashed~~ hashed index:

Hash index is no help with condition that is not equality.

Scan the hash index is prohibitively expensive

⇒ Sorted File is cheapest.

2, $\sigma_a = 50{,}000$ (R)

Sorted File:

If we use binary search we can compute the page containing the record (just 1 since R.a is a candidate key) in $\log_2 500.000 = 19$ I/Os.

(Assume pages have continuous page IDs).

However, if we take into account the fact that each page has 10 records ~~them~~ and values of R.a ranges from 0 to 4.999.999 then we can locate the page to be page 5000 th. And the task is done in just 1 I/Os.

B + tree:

Takes $\log_F 500.000 = 3$ I/Os (typical case)

Hashed index:

It takes about 1.2 I/O to retrieve the (linear) hash bucket

It takes 1 more I/O to retrieve the data page

=> in total 2.2 I/Os.

=> The cheapest plan is sorted file or linear hash index (depending on the search algorithm used in sorted file).

3, $\sigma_{a > 50,000 \wedge a < 50,010}(R)$

Sorted File:

Locate the page that contains the first record with R.a > 50.000 takes

$\log_2 500.000 = 19$ I/Os if we use binary search

or 1 I/Os if we compute directly, i.e. it must be the page ~~5~~ 5000th.

All records that satisfy $a > 50000 \wedge a < 50,010$ are in the same page, thus no additional I/O is required. (Note the last record of that page has a = 50.010 and it confirms that no more record can be found).

B+ tree:

Locate the first page takes $\log_F 500.000 = 3$ I/Os.

No additional page is needed.

Hashed index:

Hash index doesn't help with this query.

Scan hash index is prohibitedly expensive.

⇒ cheapest plan is Sorted File or B+ tree, depending on the search algorithm used in sorted file.

4, $\sigma_{a \neq 50.000}(R)$

Sorted File:

Scan the whole file : takes 500.000 I/Os.

B+ tree:

Get to the left most data pages takes

$\log_K 500.000 = 19$ I/Os

Scan the whole file via the linked list takes 500.000 I/Os

=> in total 500019 I/Os.

Hashed index:

Scan the index is prohibitedly expensive. The total number of I/Os depends on the hash function and other factors. In worst case, each ~~data entr~~ index data entry can cause an I/O. Caching pages can reduce the number of I/Os but it is still very large.

=> cheapest plan: Sorted File.

3. (ex 15.6.2)

$\sigma_c(\pi_\ell(R \times S))$

Clearly, attributes involved in c must be a subset of attributes involved in ℓ.

Assume R has 3 attributes r_1, r_2, r_3

S has 3 attributes s_1, s_2, s_3.

a. $\sigma_c(\pi_l(R \times S)) = \pi_{l_1}(\sigma_{c_1}(R) \times S)$

Then:

$c \equiv c_1$

$l \equiv l_1$.

c_1 involves only attributes of R

(*): otherwise we can push projection of R on those attributes.

l_1 has all attributes of R.(*) ~~that appear in c_1~~ ~~(and may be some other attribute of R)~~

has all attributes of S or has no attribute of S, otherwise, we can push projection of S on those attributes.

Example:

c_1 involves r_1

l_1 involves $r_1\ r_2\ r_3$

$$\sigma_{r_1=1}(\pi_{r_1 r_2 r_3}(R \times S)) \equiv \pi_{r_1 r_2 r_3}(\sigma_{r_1=1}(R) \times S)$$

b. $\sigma_c(\pi_l(R \times S)) = \pi_{l_1}(\sigma_{c_1}(R) \times \sigma_{c_2}(S))$

Then:

$c = c_1 \wedge c_2$

c_1 involves attributes of R only

c_2 " S "

$l_1 = l$.

If the attributes of R appear in l_1 is just a subset of all R's attributes, then we can push the projection ahead, i.e. perform projection before the cartesian product. Thus l_1 must have all attributes of R. (l_1 can't have no attribute of R since $c_1 \subseteq l_1$).

The same for S, i.e. l_1 has all attributes of S.

Example:

c_1 involves r_1

c_2 involves s_1

l_1 involves all attributes of R and S

$$\sigma_{r_1=1 \wedge s_1=2}(\pi_{r_1 r_2 r_3 s_1 s_2 s_3}(R \times S))$$

$$\equiv \pi_{r_1 r_2 r_3 s_1 s_2 s_3}(\sigma_{r_1=1}(R) \times \sigma_{s_1=2}(S))$$

c. $\sigma_c(\pi_l(R \times S)) \equiv \sigma_c(\pi_{l_1}(\pi_{l_2}(R) \times S))$

Then:

$l_1 = l$

l_2 involves only R

c : all conjuncts in c involve both R and S; otherwise we can push the conjunct ahead.

Since we don't push projection of S ahead, l_1 either:

+ contains all attributes of S
+ contains no attribute of S. This doesn't make sense because if that is the case, l_1 contains only attributes of R and l_1 must be subset of l_2, and thus we just need l_1, no need for l_2, i.e. the equivalent expression is $\sigma_c(\pi_{l_1}(R) \times S)$.

For those attributes of R appear in l_1, they must also appear in l_2, i.e. $(l_1 \cap R) \subseteq l_2$. Also $l_1 \cap R = l_2$, otherwise we should $\pi_{l_1 \cap R}(R)$ instead of $\pi_{l_2}(R)$.

example:

l_2 involves r_1

l_1 involves r_1, s_1, s_2, s_3

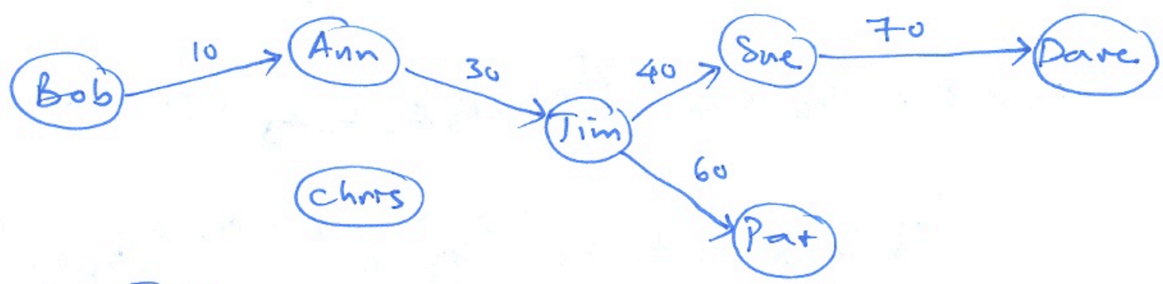
c involves r_1, s_1

$$\sigma_{r_1=s_1}(\pi_{r_1 s_1 s_2 s_3}(R \times S)) \equiv \sigma_{r_1=s_1}(\pi_{r_1 s_1 s_2 s_3}(\pi_{r_1}(R) \times S)).$$

Security / Access Control

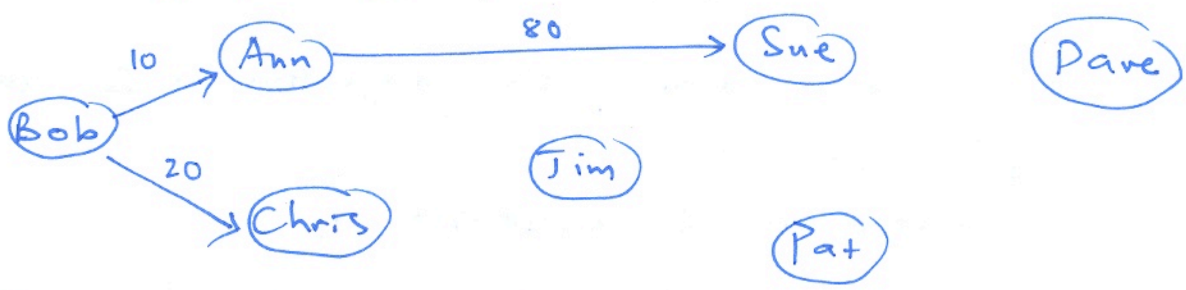
1. At time 100, Bob ~~into~~ execute
   REVOKE SELECT ON T FROM CHRIS

   Then the graph will be



   1. True
   2. True
   3. Sue, Jim, Ann, Bob.

2. At time 100 Ann: REVOKE SELECT ON T FROM JIM
   At time 110 Chris: REVOKE SELECT ON T FROM JIM
   The graph after that will be:



   1. True
   2. False
   3. Sue, Jim, Ann, Bob. (in old graph)
      in new graph, Dave doesn't have select privilege anymore.