Irregular Data Structures

Linked lists, trees, and graphs

Motivation

- Irregular data structures needed to overcome disadvantages of arrays
  - Easy expansion and contraction to keep up with dynamic data size
  - Modeling of irregular data, with complex "neighboring" relationship

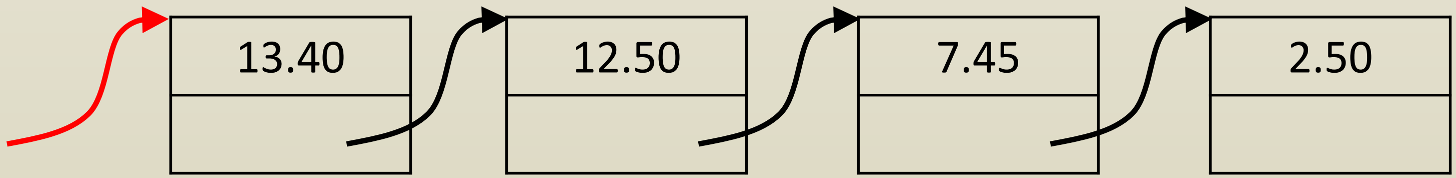
Cost

- Irregular data structures
  - Increased complexity
  - Decreased efficiency
    - Structure stored explicitly, not all storage used to store data
  - No direct access to all data

Linked list

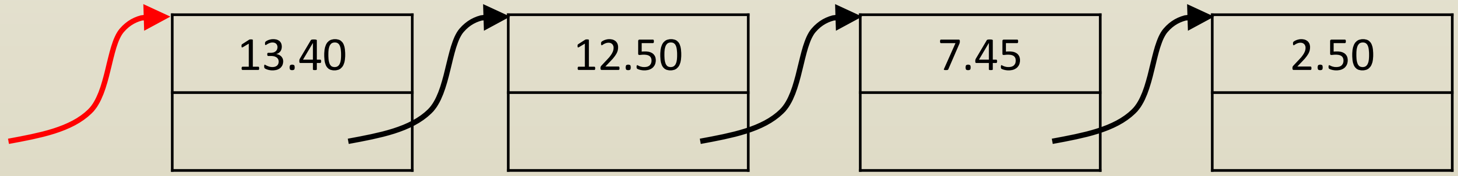
- A 1-D sequence data structure
- Not an array
- Each data element is linked to the next
- Link: memory address pointing to a data element
- Link list node: data element + link

- Example
  - credit card transaction amounts in dollars, sorted
  - Links stored explicitly
    - E.g. 32 bit / link
  - Actual address irrelevant here
    - Link shown with arrow
  - Link to first element has to be known (shown in red)
  - Link of last element is null



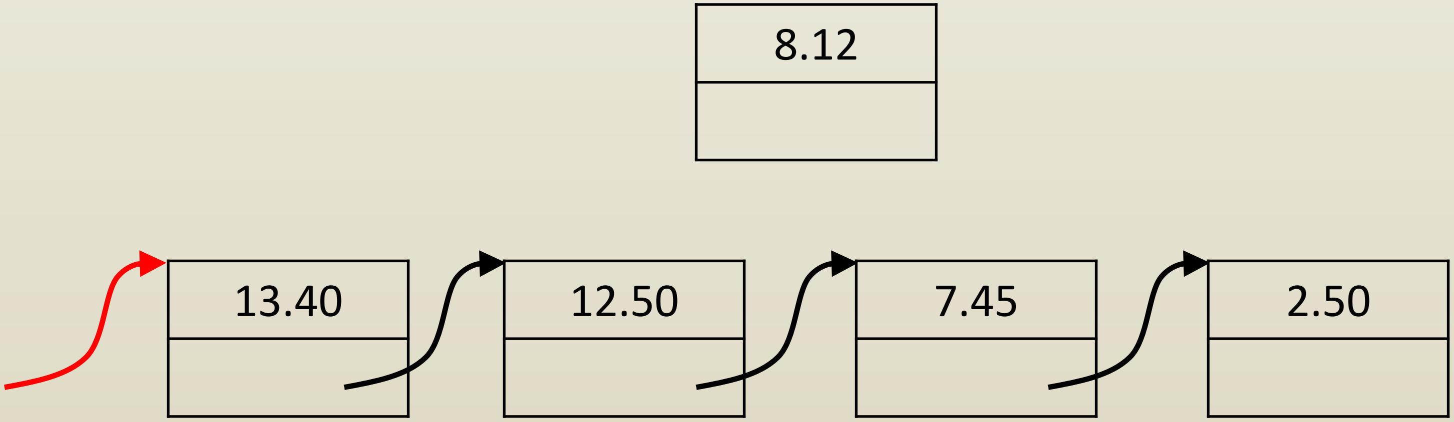
Linked list

- Add a new transaction in the amount of $8.12
  - Start at first node (using known red arrow link)
  - Is amount (13.40) smaller than $8.12?
  - No, use node link to go to next node
  - Is amount (12.50) smaller than $8.12?
  - No, use node link to go to next node
  - Is amount (7.45) smaller than $8.12?



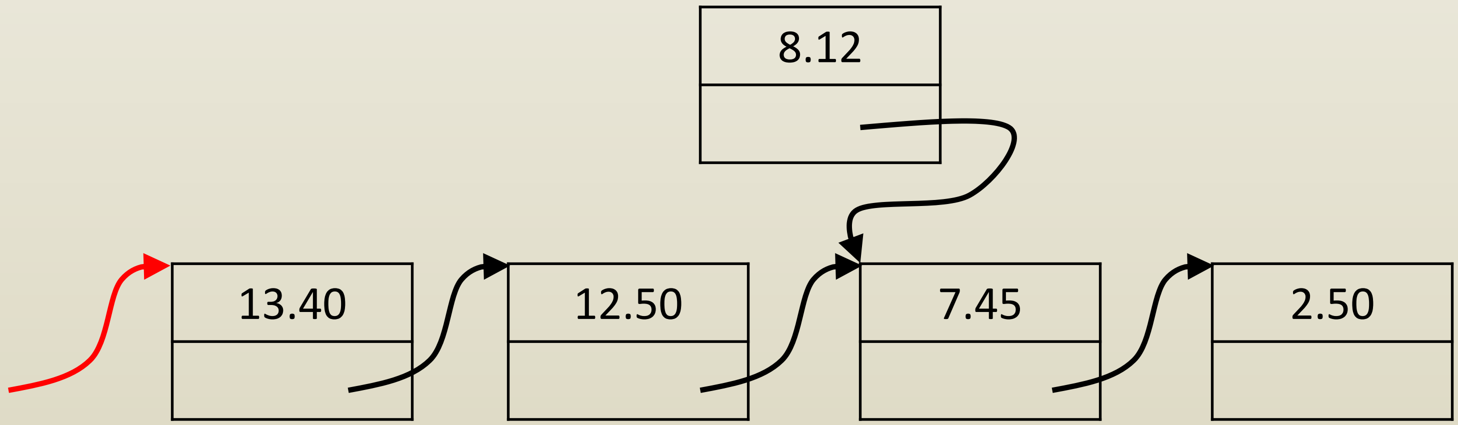
Linked list

- Add a new transaction in the amount of $8.12
  - Yes, insert new node
    - Make new node
    - Set new node amount to $8.12



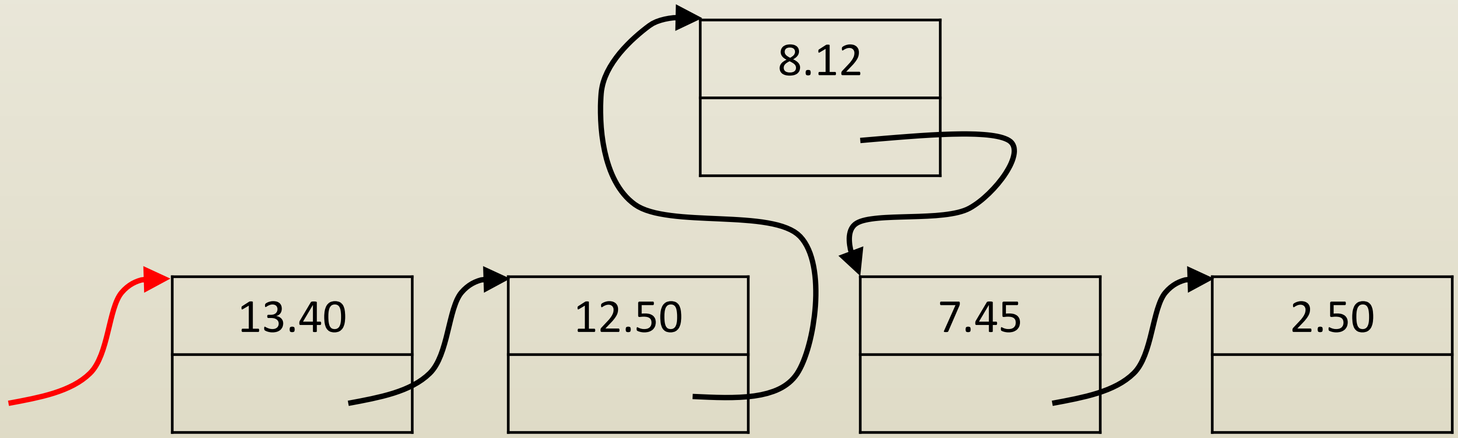
Linked list

- Add a new transaction in the amount of $8.12
  - Yes, insert new node
    - Make new node
    - Set new node amount to $8.12
    - Set new node link to next node



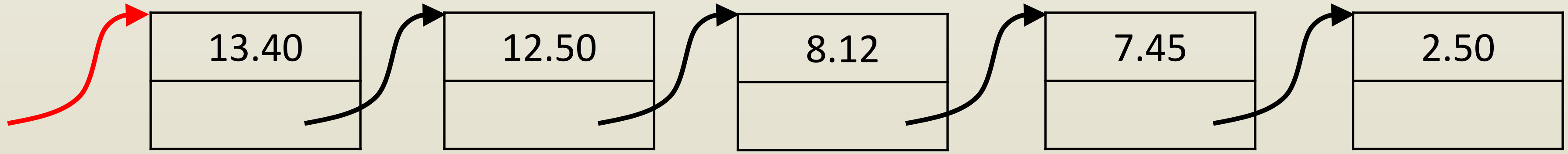
Linked list

- Add a new transaction in the amount of $8.12
  - Yes, insert new node
    - Make new node
    - Set new node amount to $8.12
    - Set new node link to next node
    - Set previous node link to new node



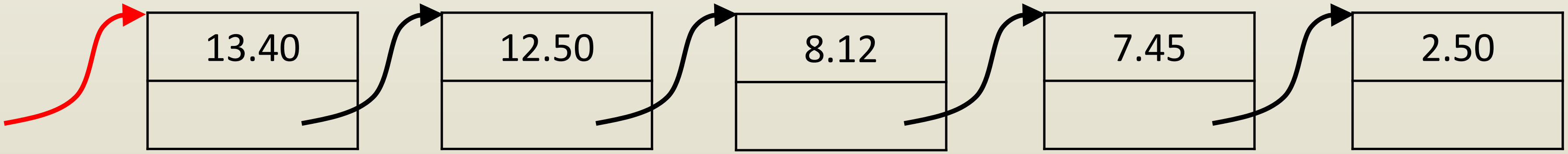
Linked list

- Add a new transaction in the amount of $8.12



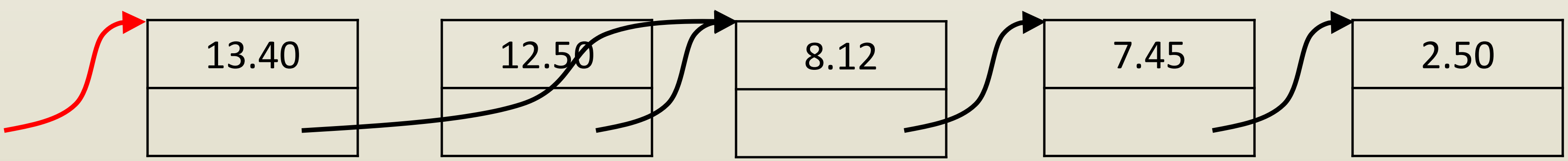
Linked list

- Delete transaction $12.50
  - Move to node storing $12.50 transaction



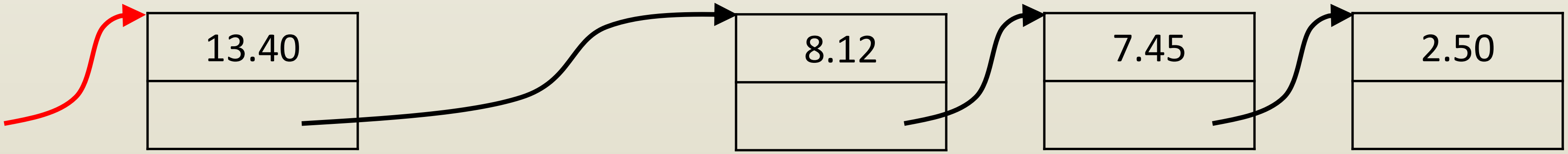
Linked list

- Delete transaction $12.50
  - Move to node storing $12.50 transaction
  - Set link of previous node to next node



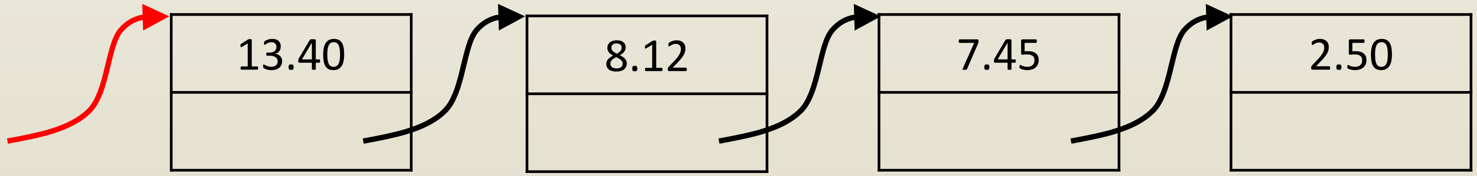
Linked list

- Delete transaction $12.50
  - Move to node storing $12.50 transaction
  - Set link of previous node to next node
  - Delete current node



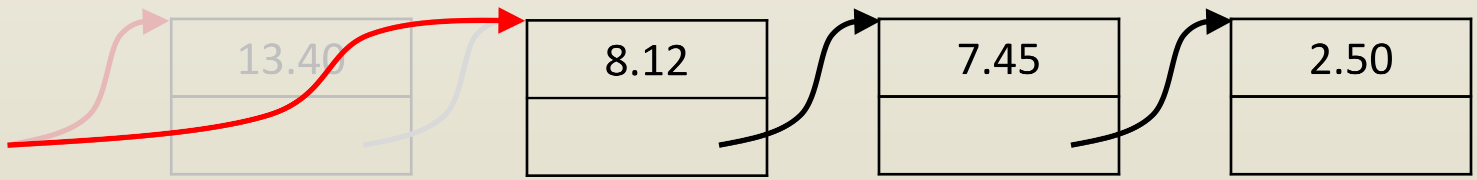
Linked list

- Delete transaction $12.50



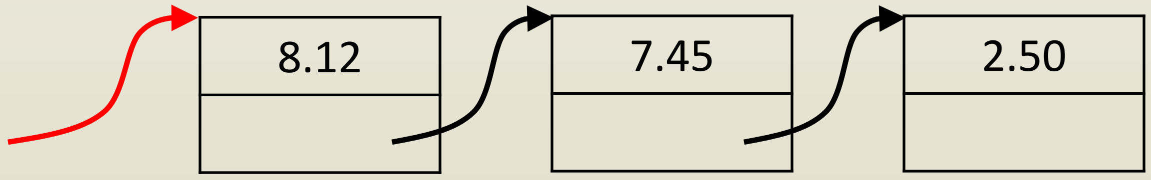
Linked list

- Delete transaction $13.40
  - Special case
  - Set red link equal to link of first node
  - Delete first node



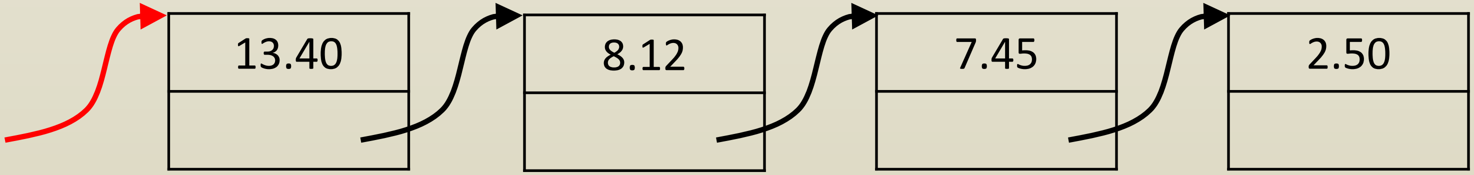
Linked list

- Delete transaction $13.40
  - Special case
  - Set red link equal to link of first node
  - Delete first node



Linked list

- Advantages
  – List grows and shrinks as needed, w/o having to modify entire list
  – Insertion & deletion imply local changes
- Disadvantages
  – You cannot find third transaction directly
    - Have to traverse list
  – Storing link implies overhead



iClicker question

When inserting a transaction with value 1.00 in the linked list below, which of the following statements is true:

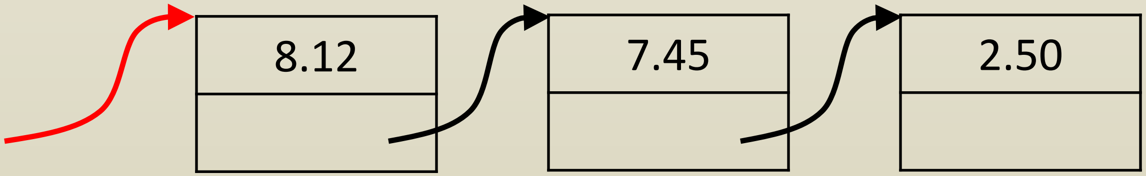
A. The transaction cannot be inserted since there is no transaction of smaller value.

B. The next node link of the new node will be NULL.

C. The insertion point is found when the next node link of the current node is found to be NULL.

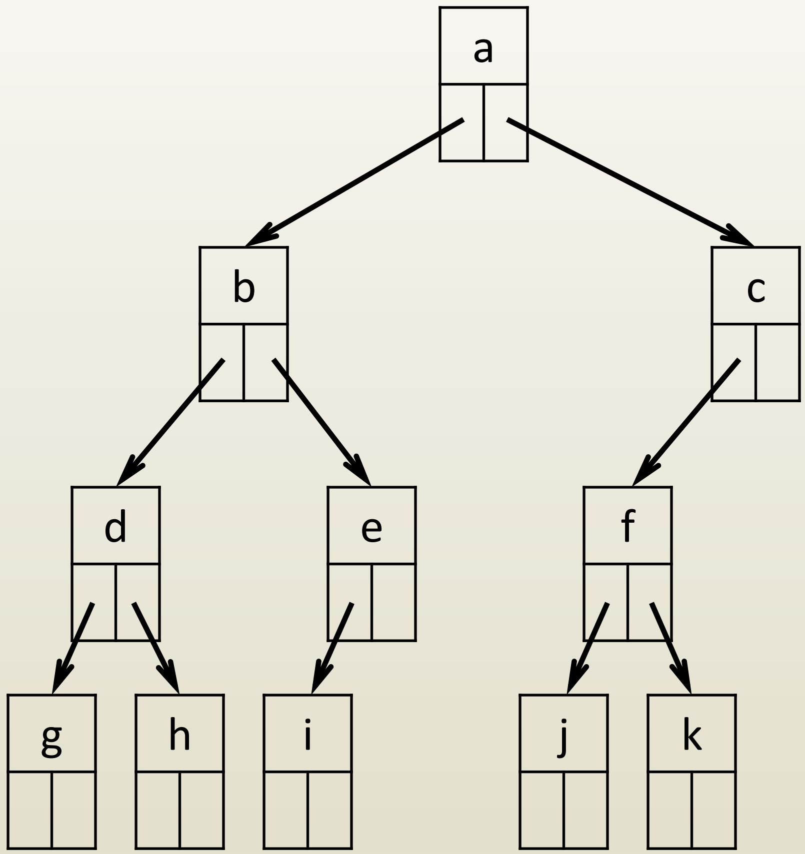
D. A, B, and C are true.

E. B and C are true.



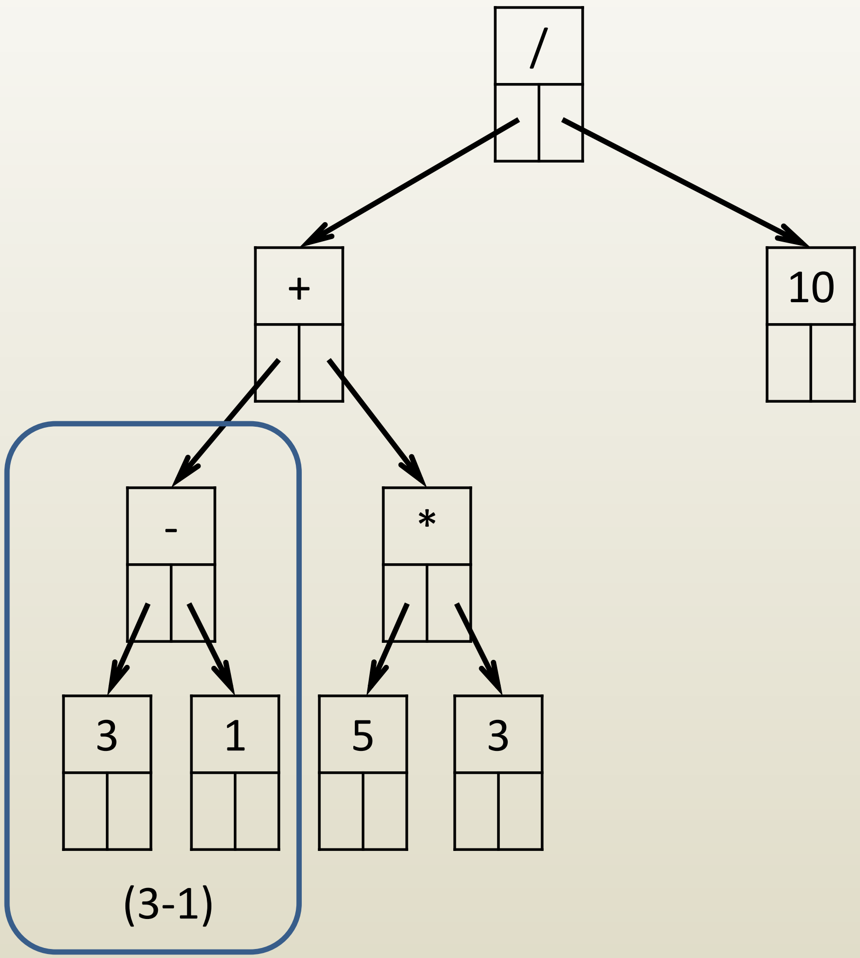
Binary tree

- Definition
  - A hierarchical data structure
  - A (parent) node links to 0, 1, or 2 (children) nodes
  - The starting node is called root; the root is not the child of any node
  - Nodes with 0 children are called leafs
  - Non-leaf nodes are called internal



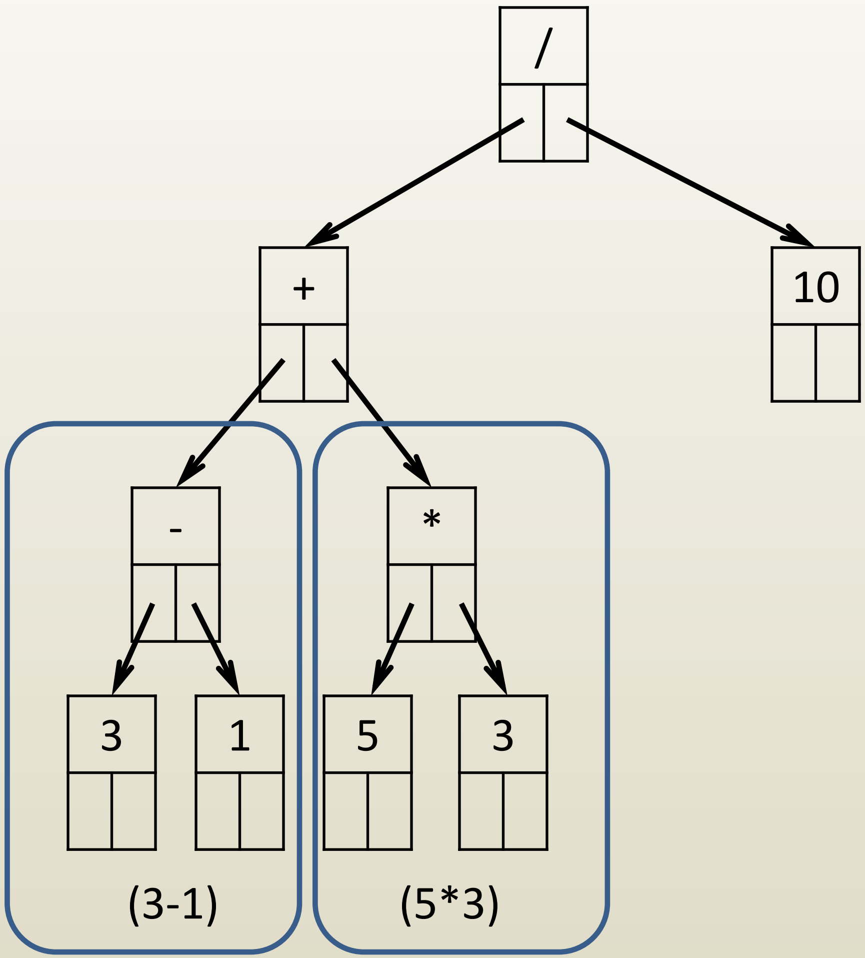
Arithmetic expression binary tree

- Operators at internal nodes
- Operands at leafs



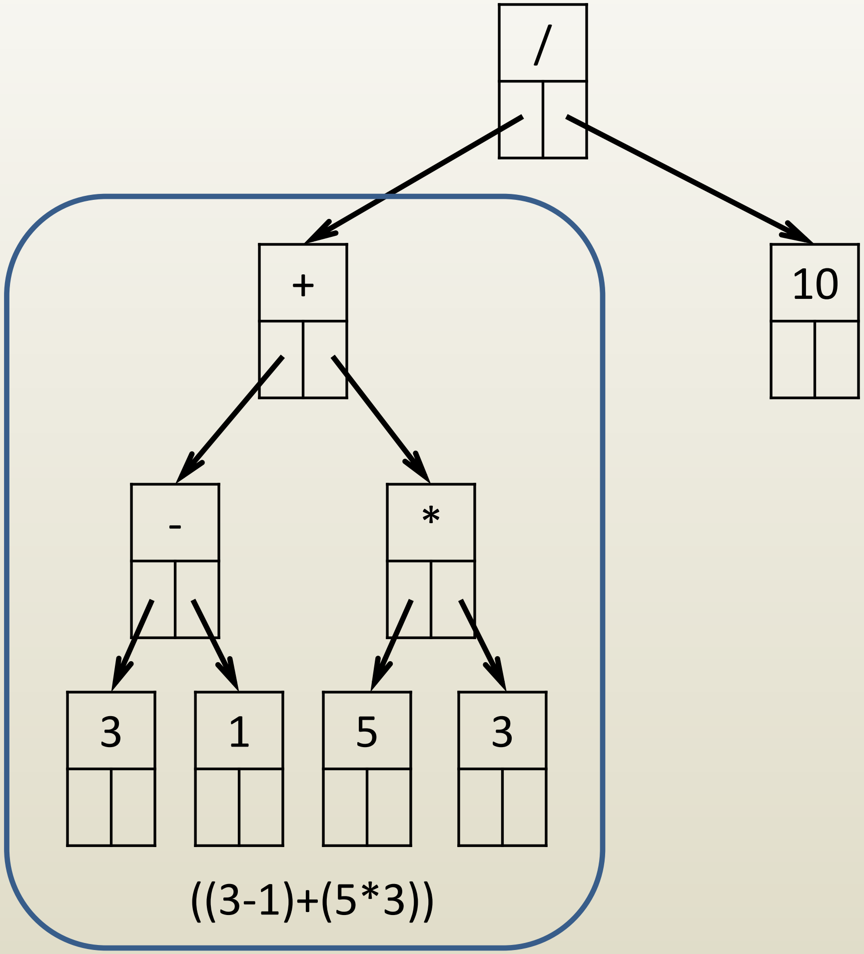
Arithmetic expression binary tree

- Operators at internal nodes
- Operands at leafs



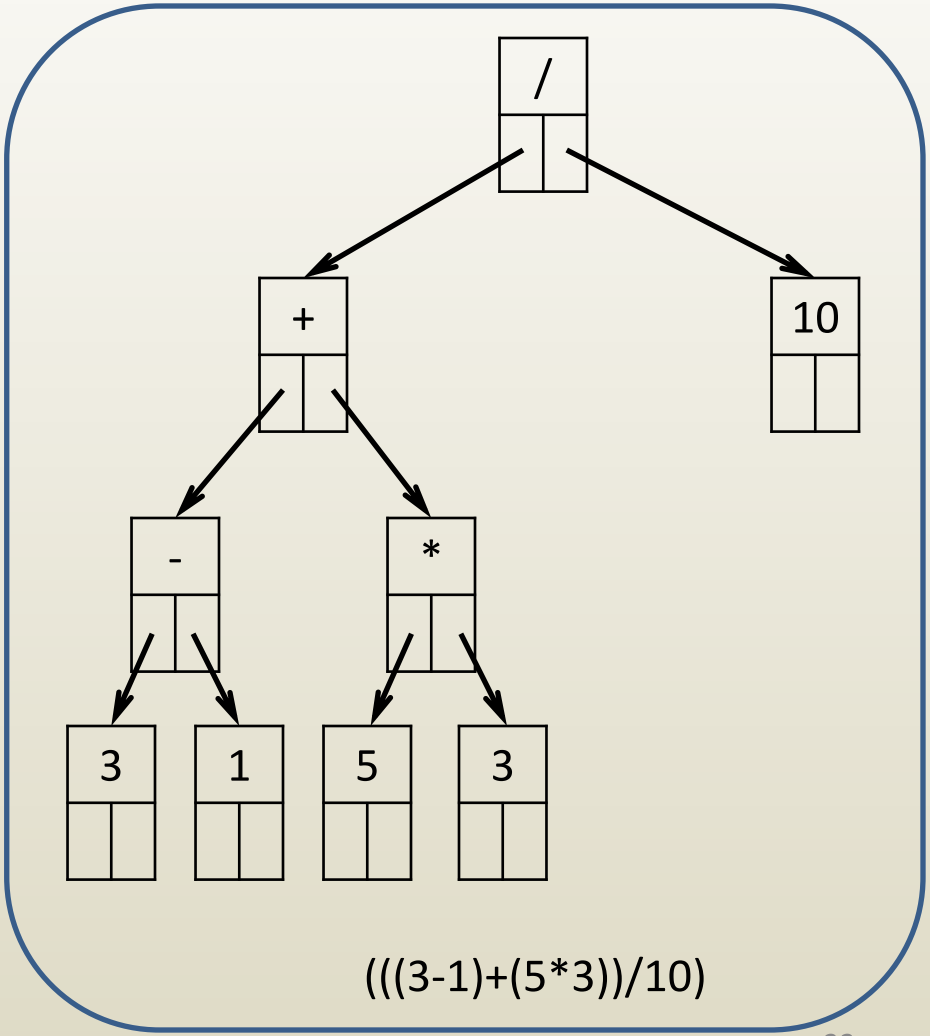
Arithmetic expression binary tree

- Operators at internal nodes
- Operands at leafs



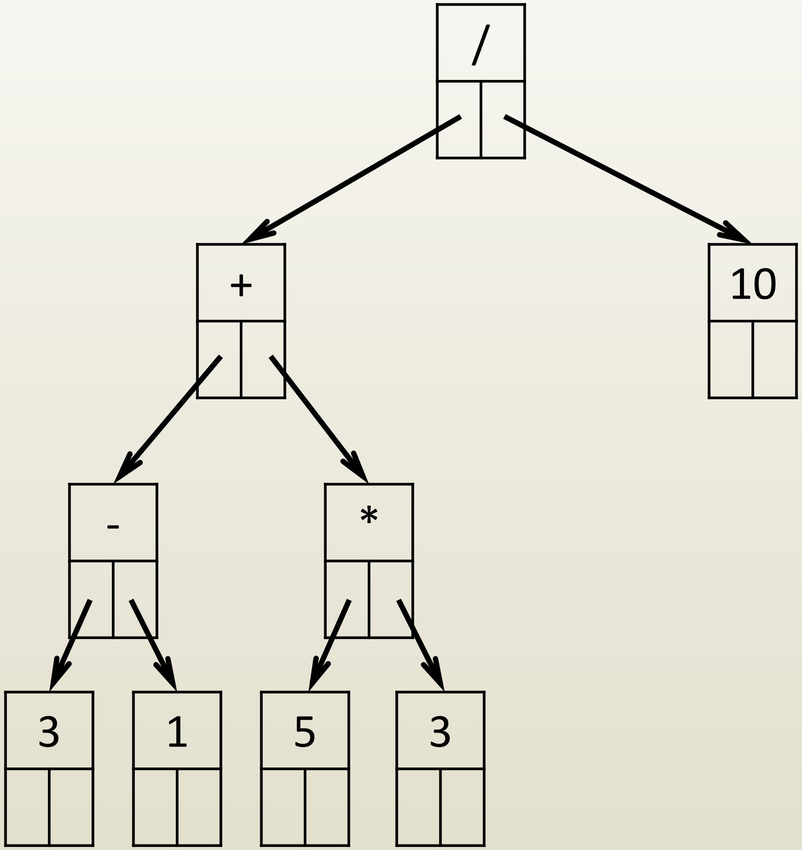
Arithmetic expression binary tree

- Operators at internal nodes
- Operands at leafs



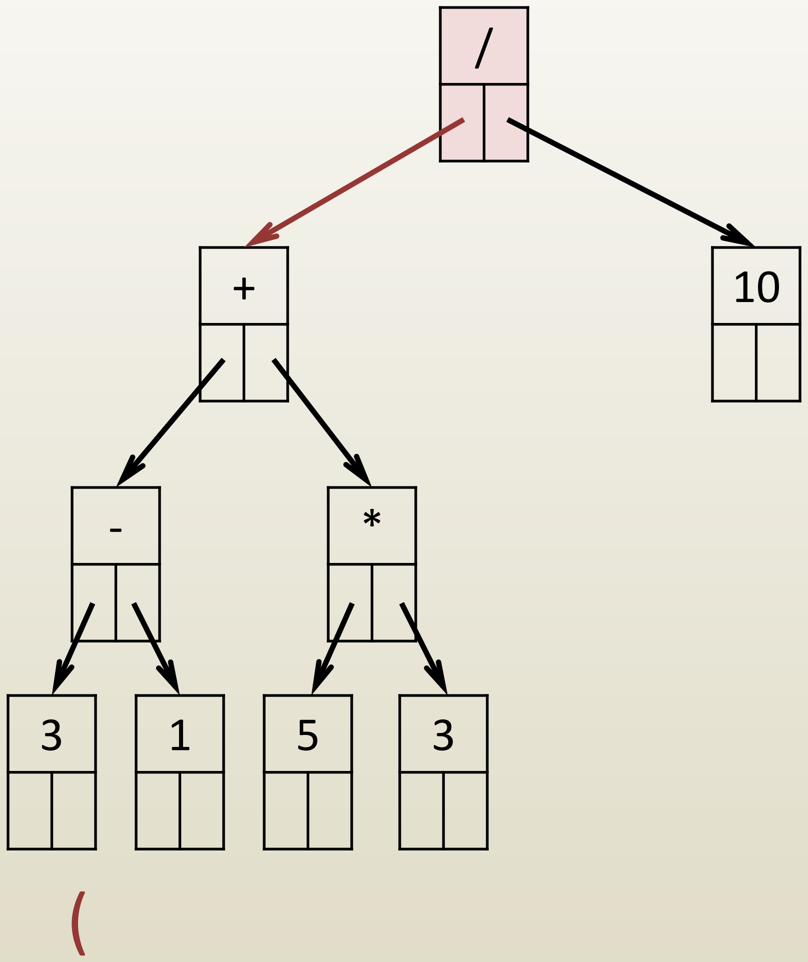
Arithmetic expression binary tree

- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



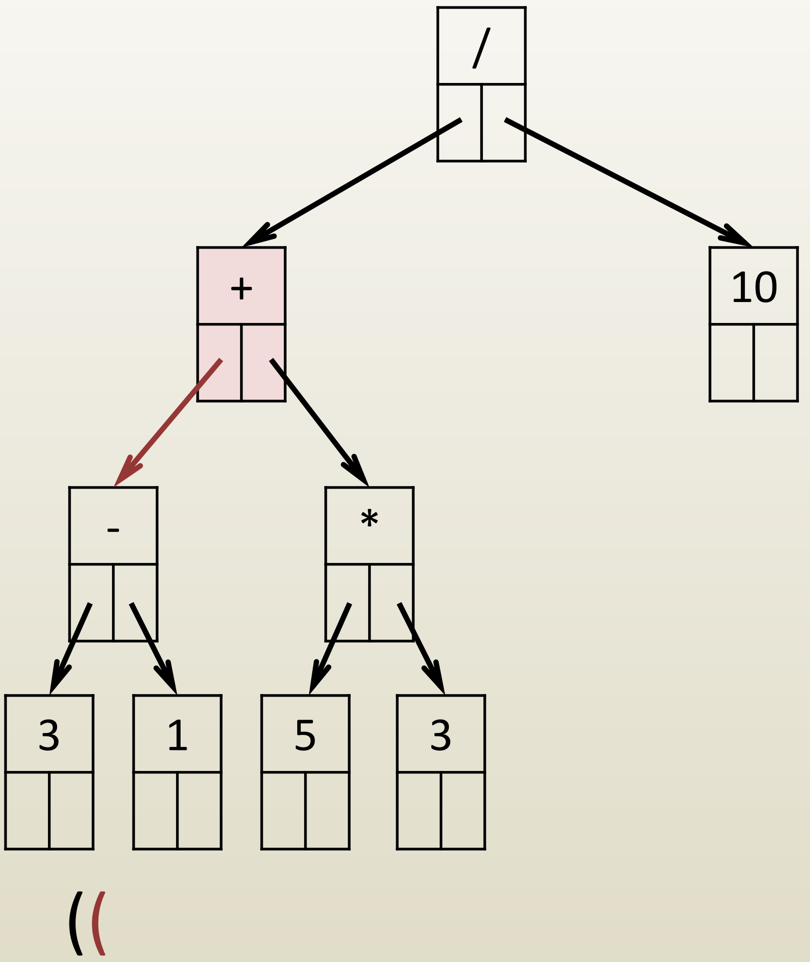
Arithmetic expression binary tree

- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



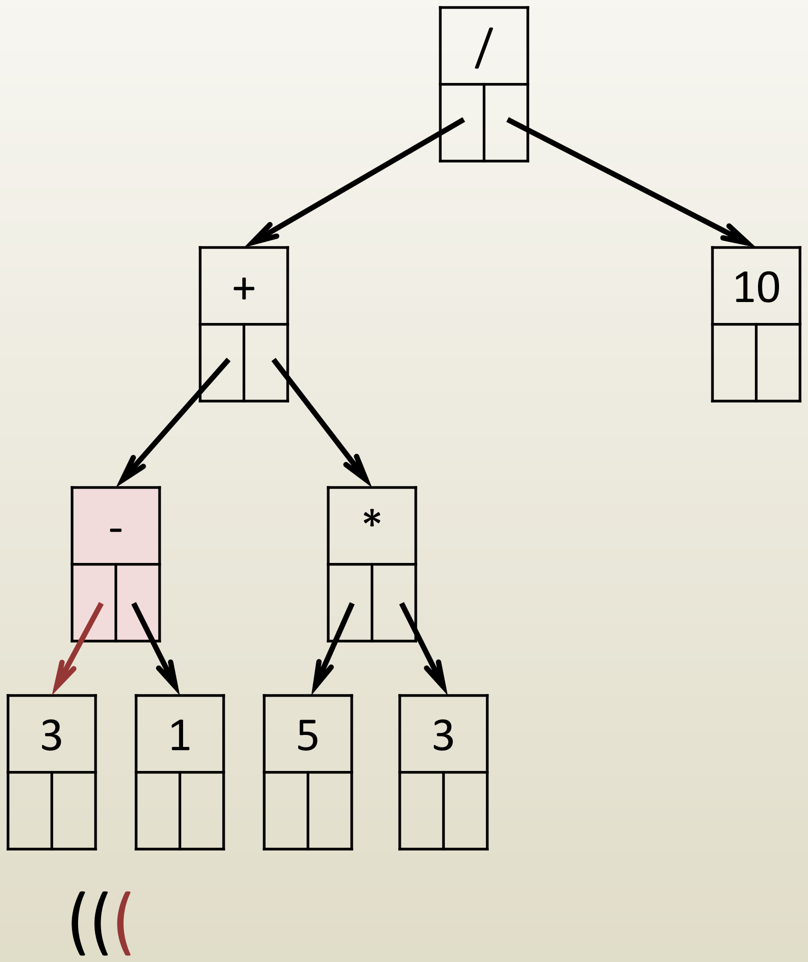
Arithmetic expression binary tree

- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



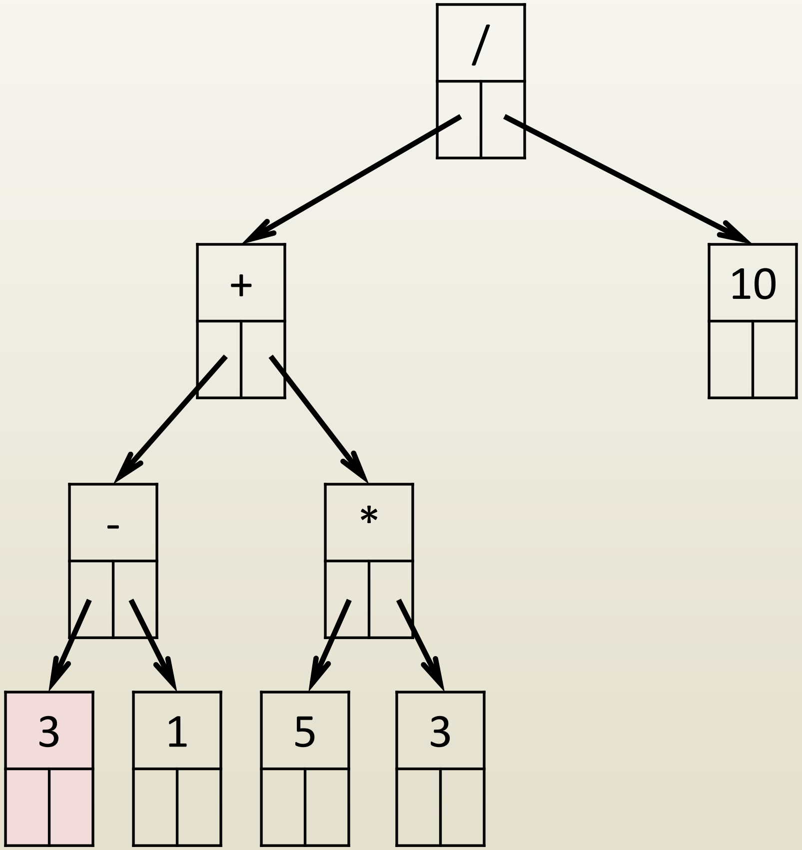
Arithmetic expression binary tree

- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right



Arithmetic expression binary tree

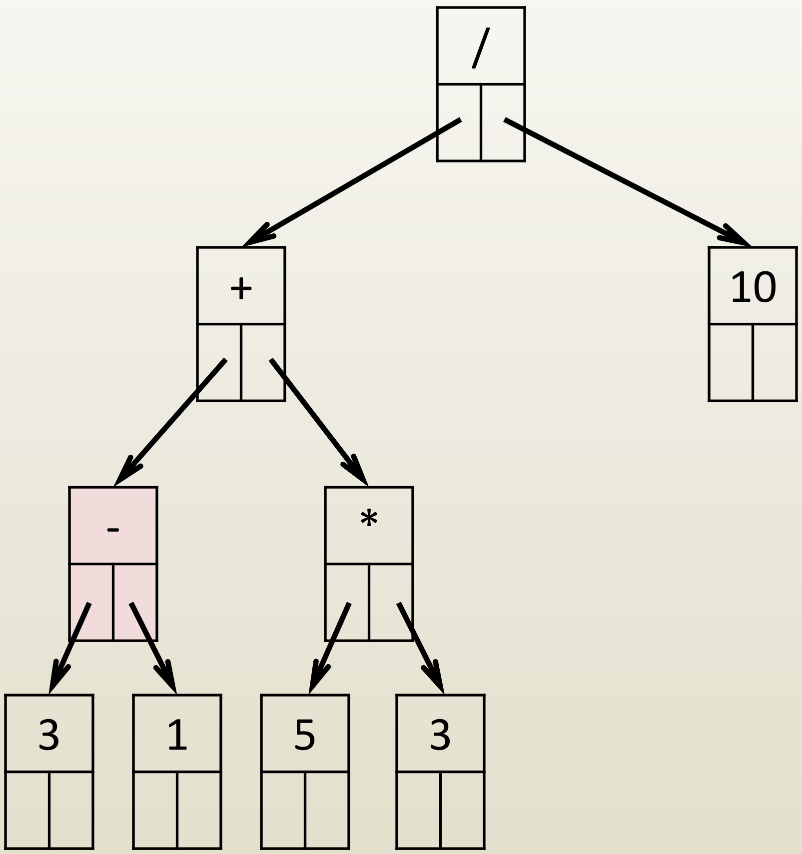
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3

Arithmetic expression binary tree

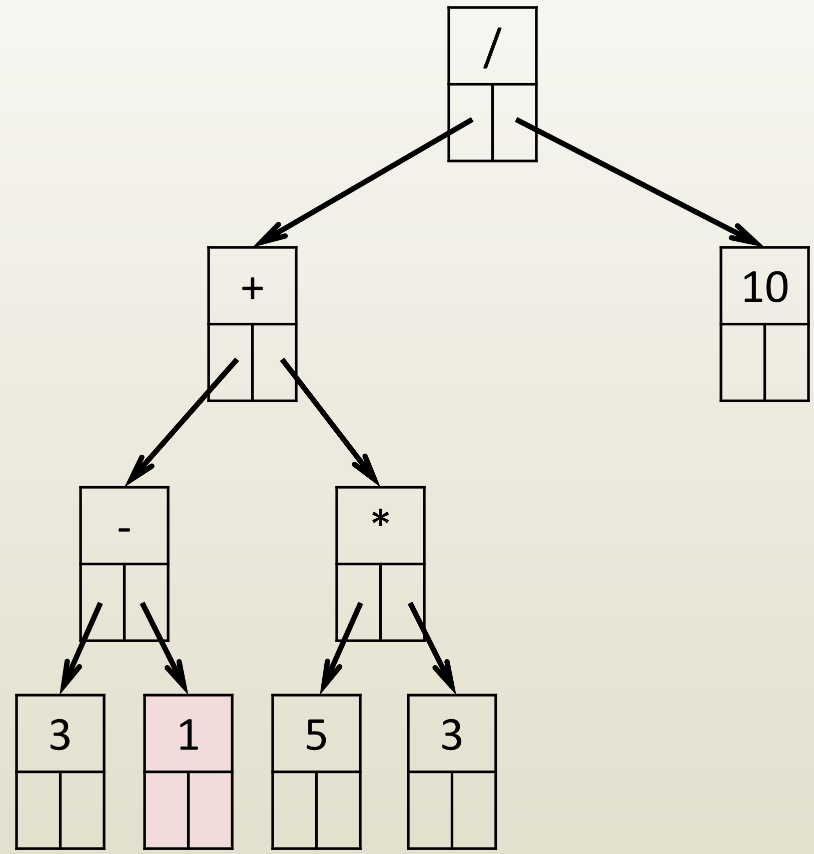
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-

Arithmetic expression binary tree

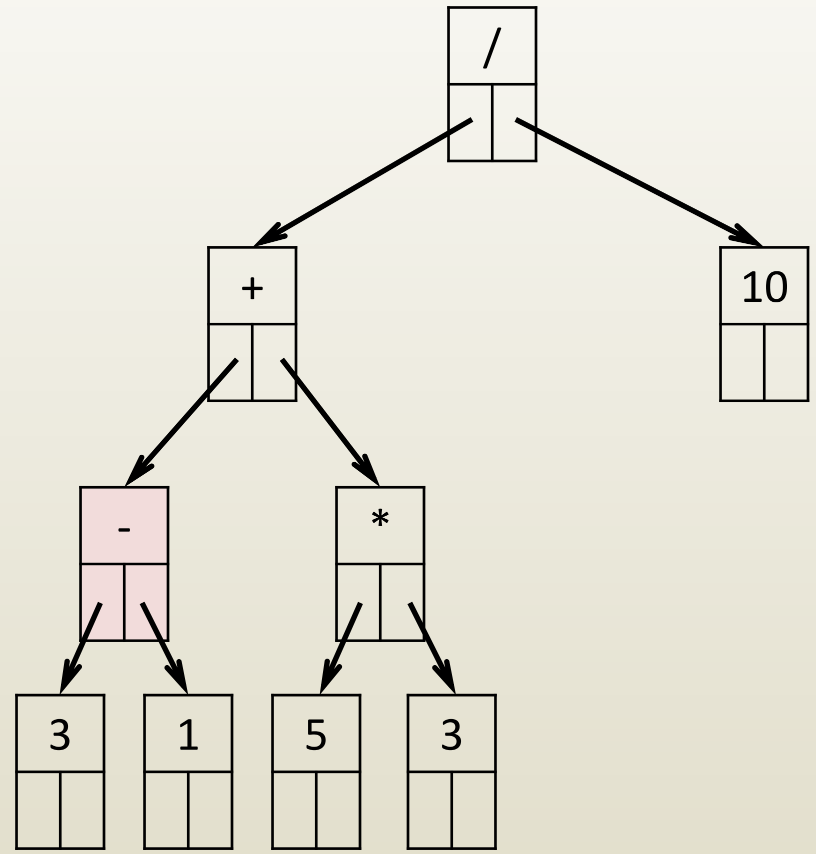
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right



(((3-1

Arithmetic expression binary tree

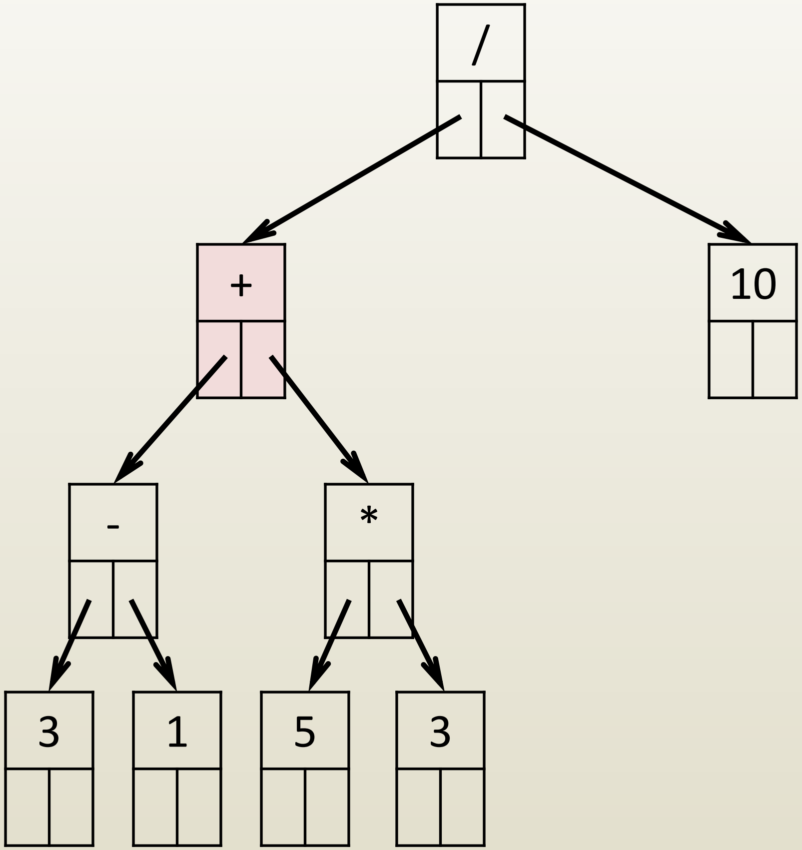
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-1)

Arithmetic expression binary tree

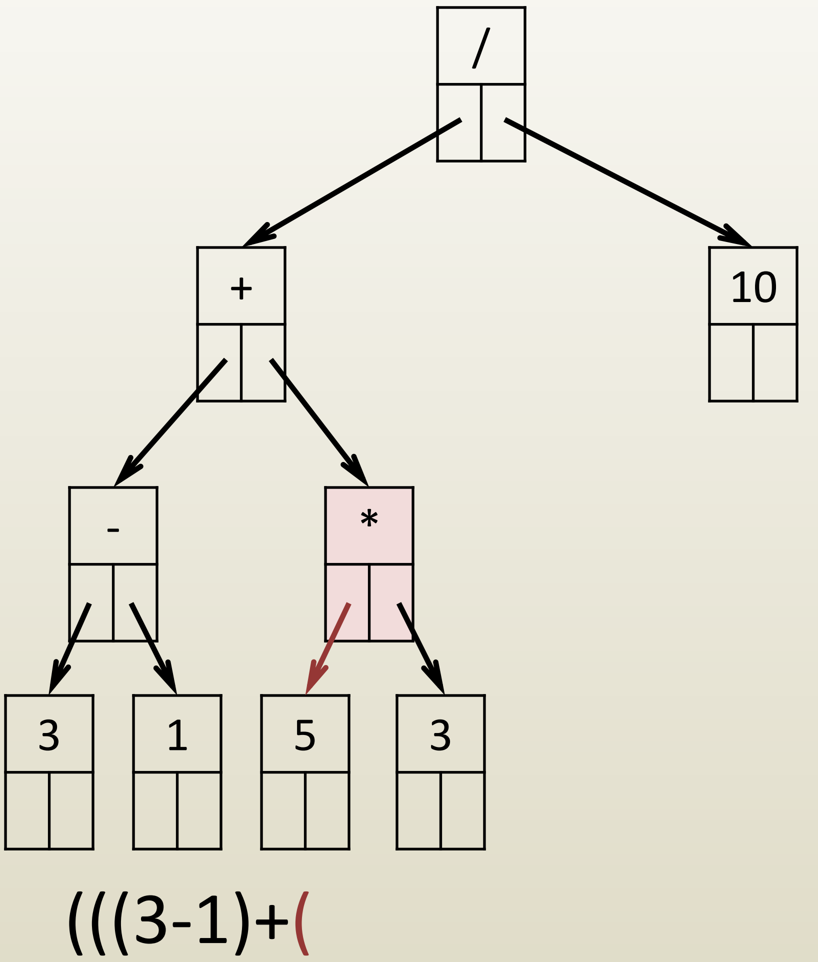
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-1)+

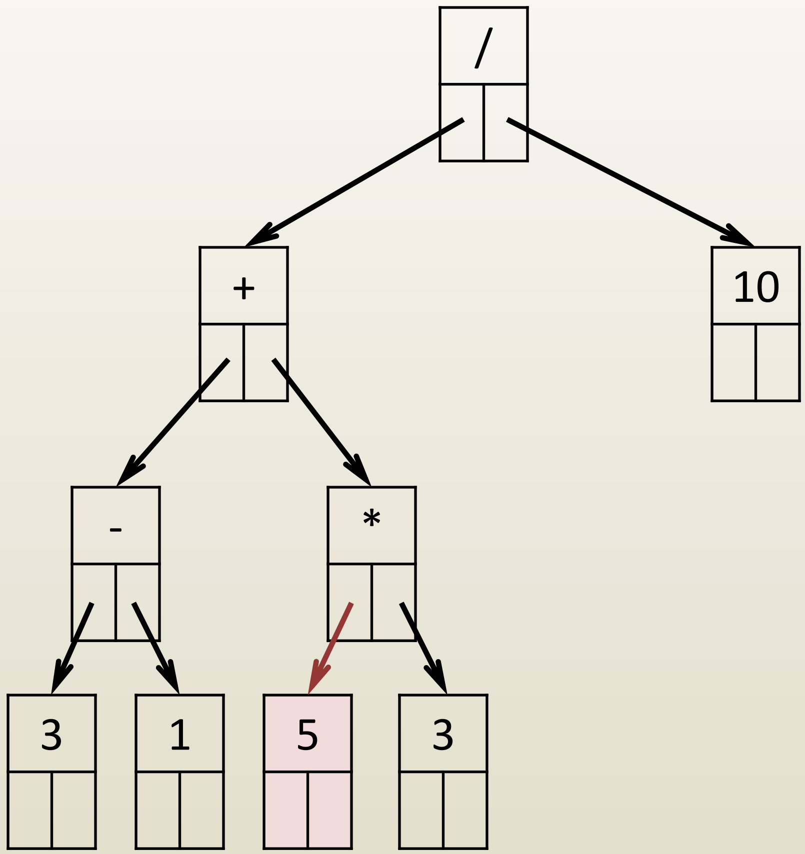
Arithmetic expression binary tree

- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



Arithmetic expression binary tree

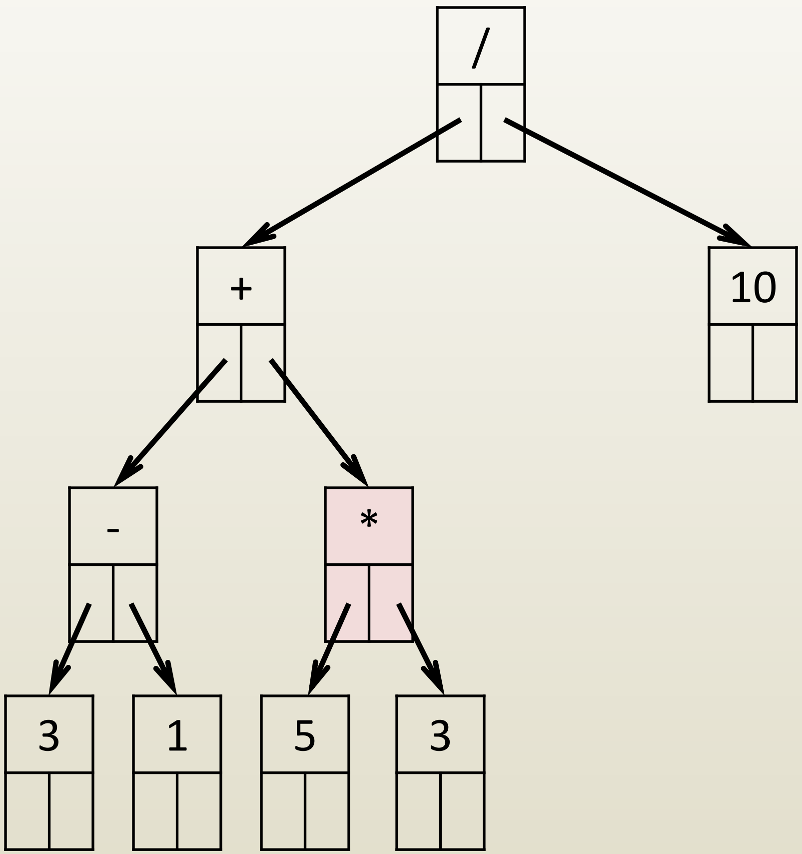
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-1)+(5

Arithmetic expression binary tree

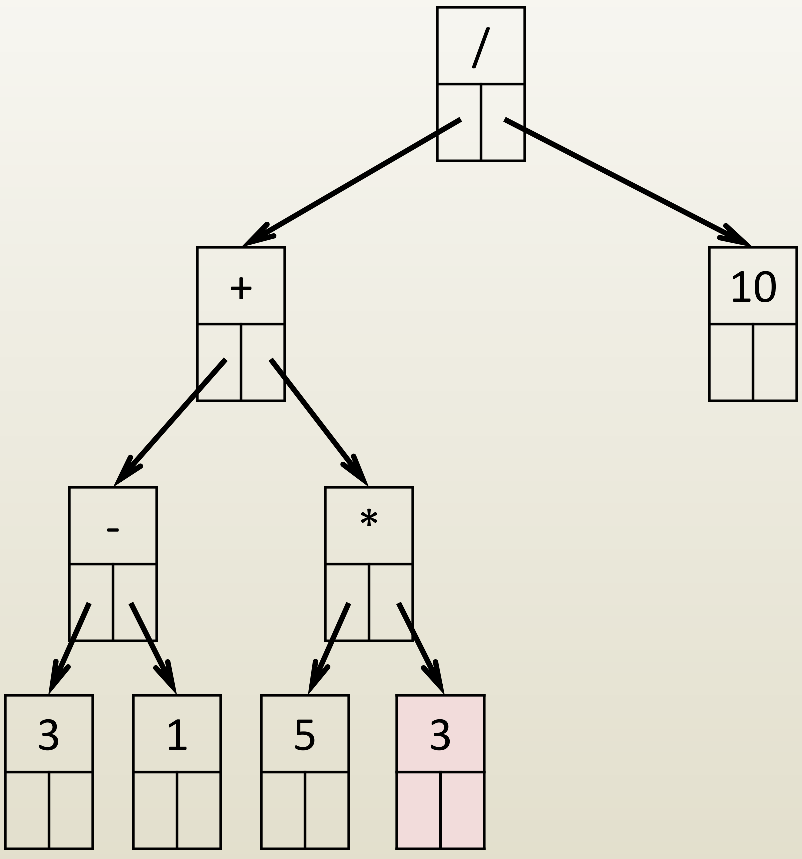
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-1)+(5*

Arithmetic expression binary tree

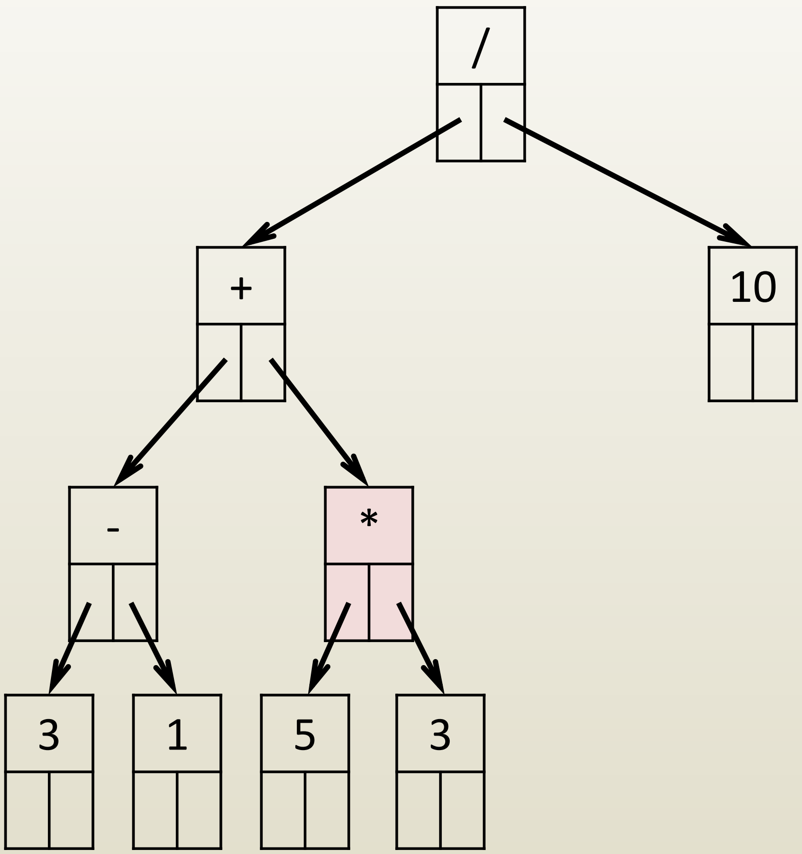
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-1)+(5*3

Arithmetic expression binary tree

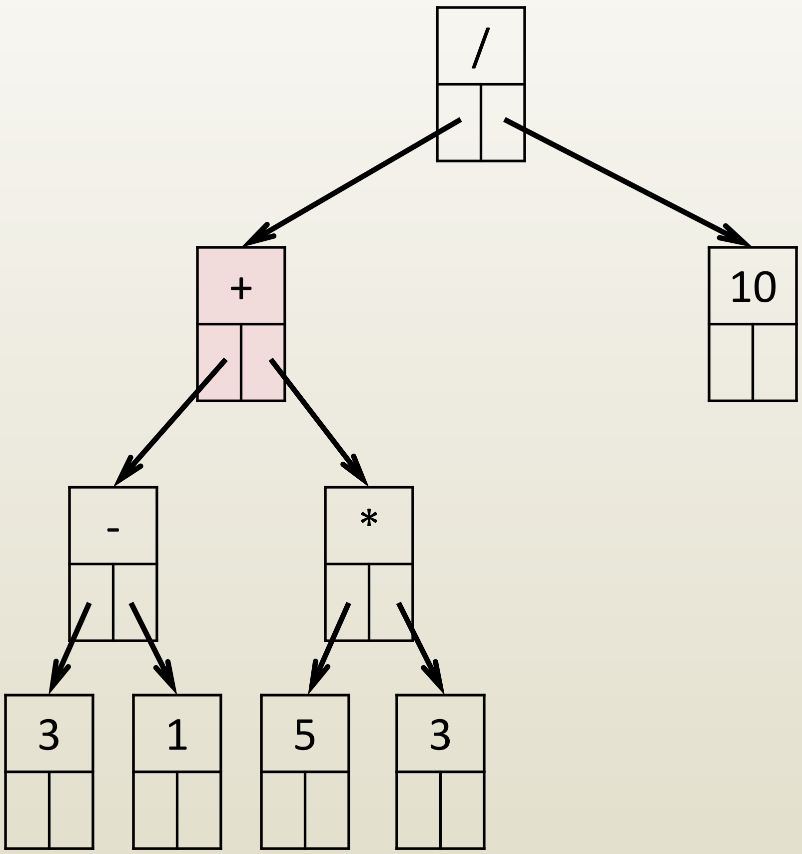
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-1)+(5*3)

Arithmetic expression binary tree

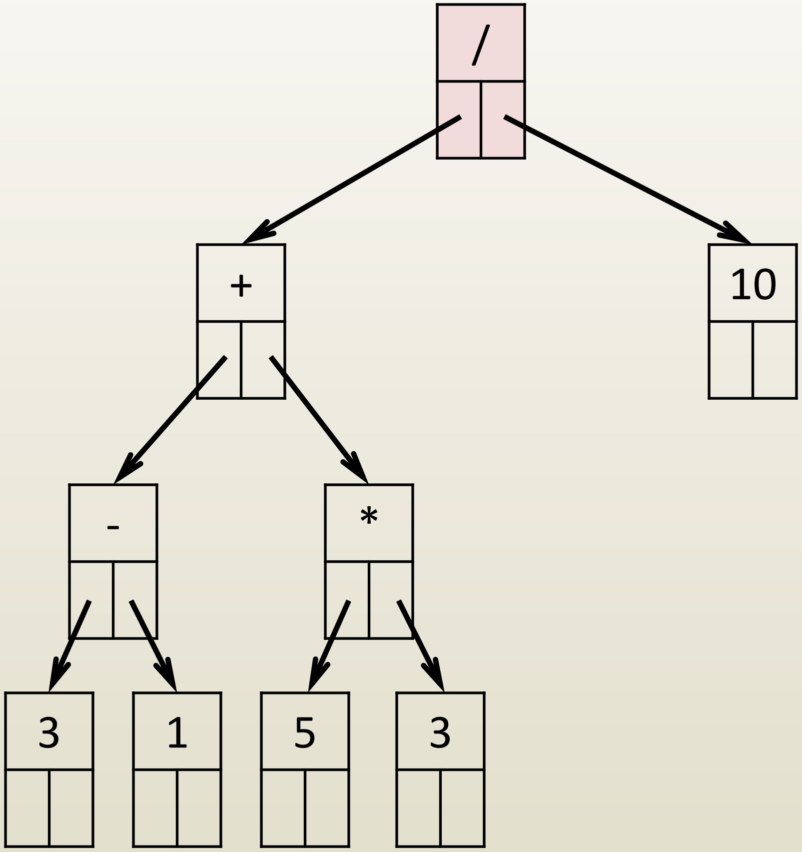
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write “(“ before going left
  - Write current node symbol before going right
  - Write “)” after having gone right



(((3-1)+(5*3))

Arithmetic expression binary tree

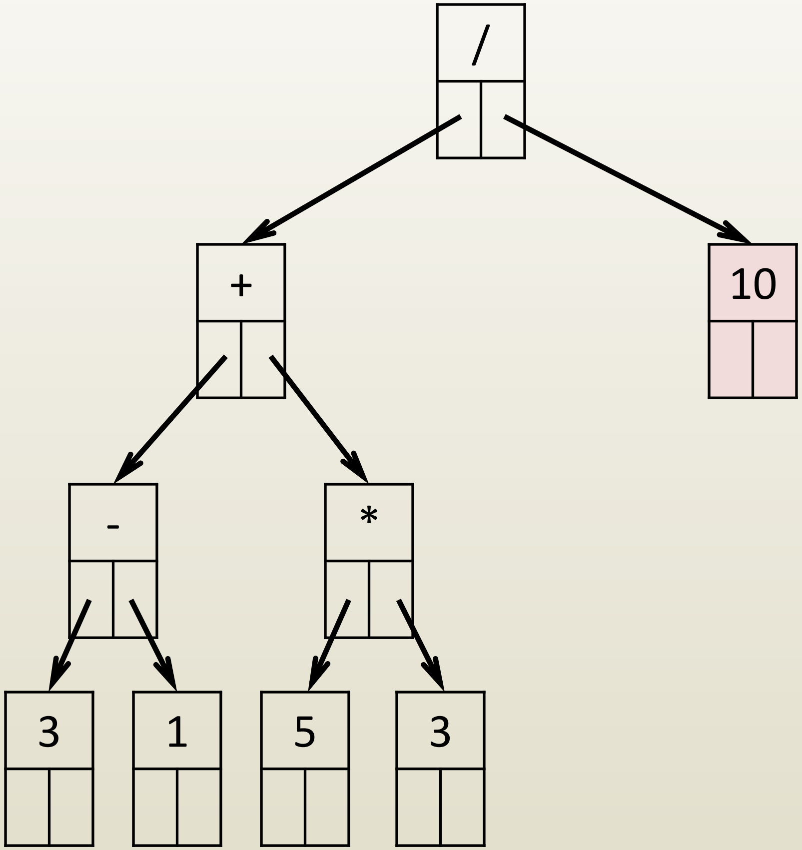
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-1)+(5*3))/

Arithmetic expression binary tree

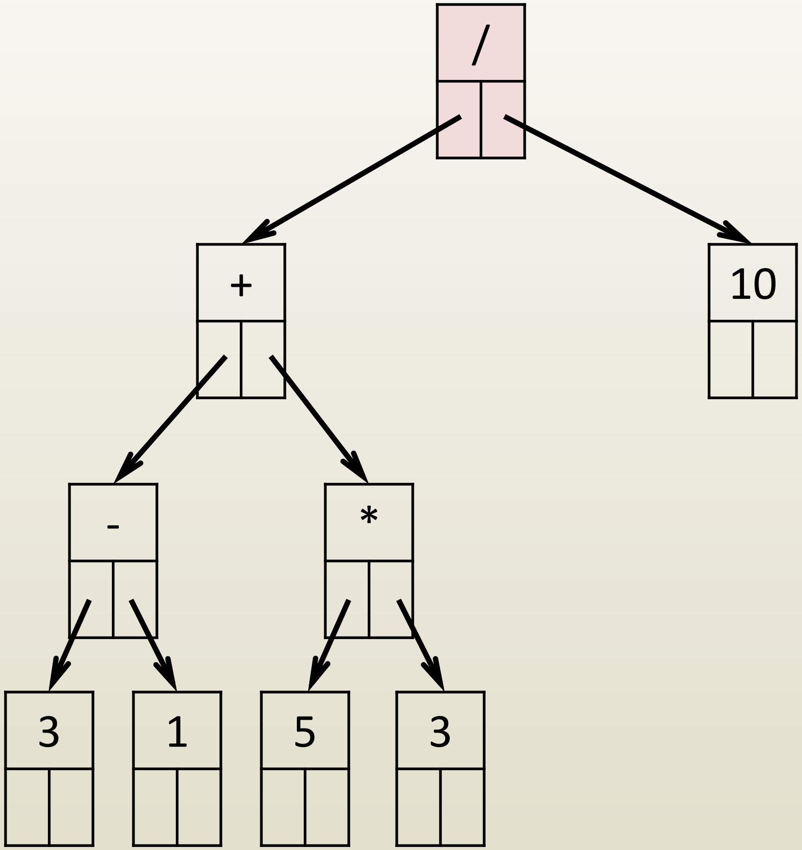
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-1)+(5*3))/10

Arithmetic expression binary tree

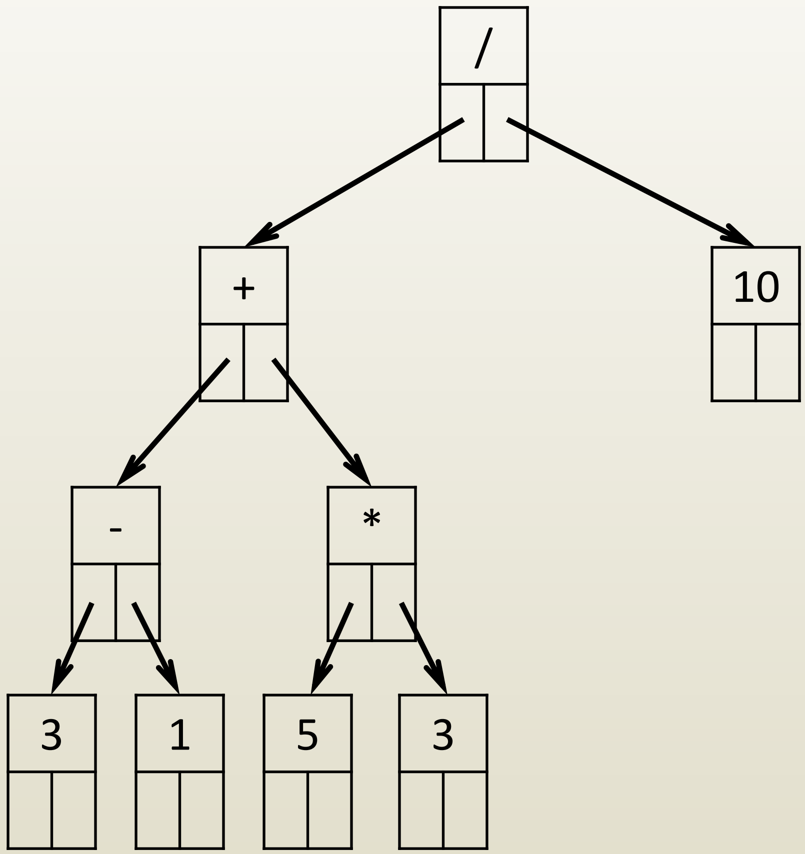
- Arithmetic expression can be recovered by traversing tree
- Traversal: visiting all nodes
- Traversal rules
  - Start at root
  - For every node
    - go left until dead end
    - then go right until dead end
    - then go back up
- Printout rules
  - Write "(" before going left
  - Write current node symbol before going right
  - Write ")" after having gone right



(((3-1)+(5*3))/10)

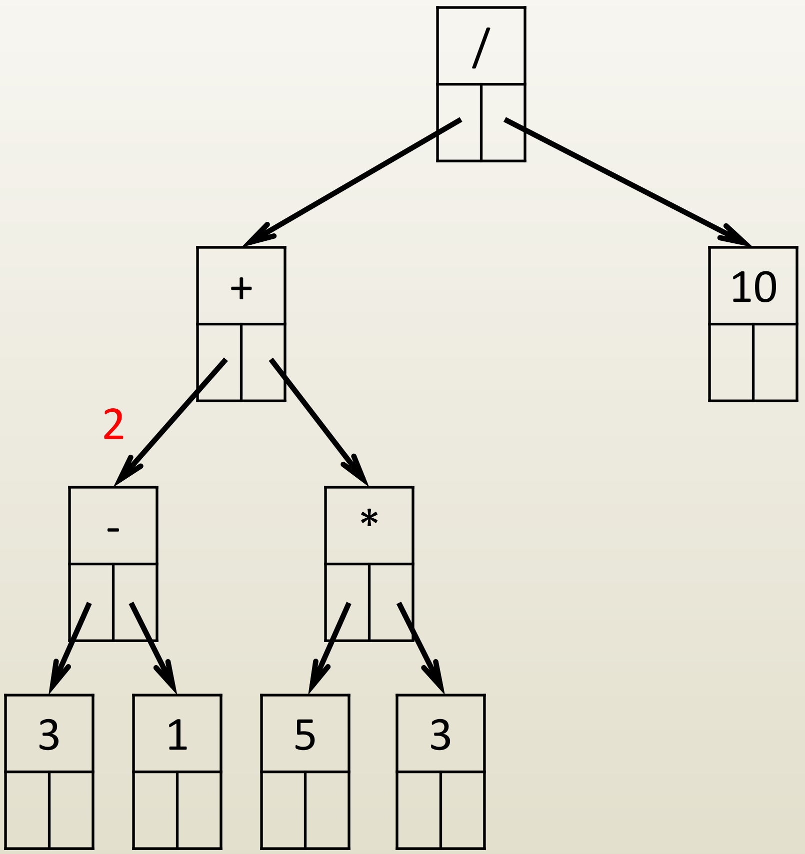
Arithmetic expression binary tree

- Arithmetic expression can be evaluated by traversing tree
- Evaluation rules
  - if leaf, return operand
  - valLeft = Evaluate left
  - valRight = Evaluate right
  - return valLeft operator valRight



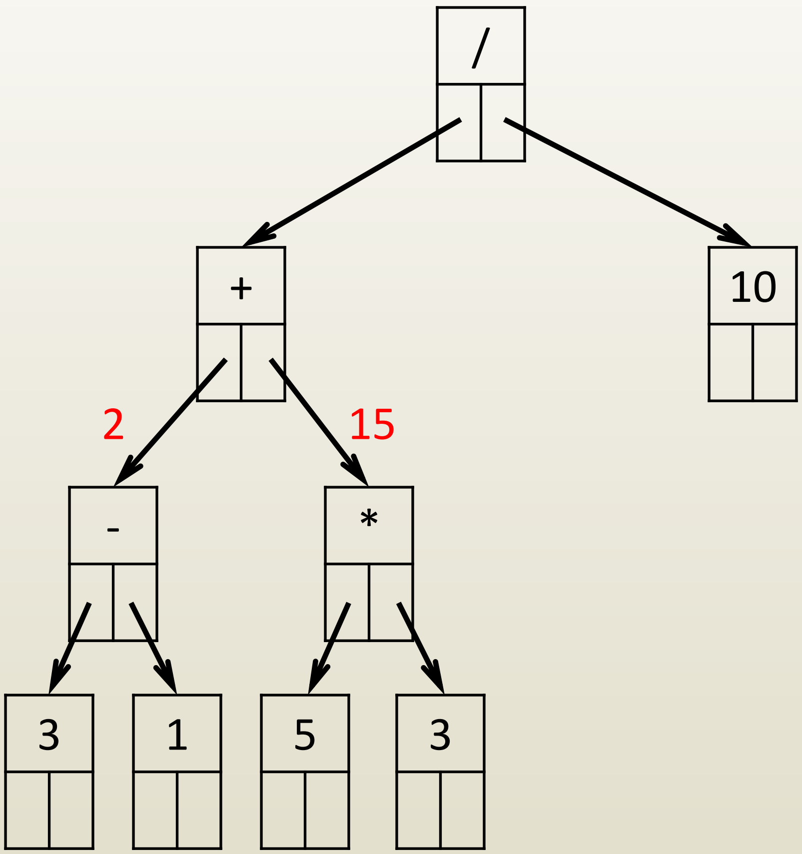
Arithmetic expression binary tree

- Arithmetic expression can be evaluated by traversing tree
- Evaluation rules
  - if leaf, return operand
  - valLeft = Evaluate left
  - valRight = Evaluate right
  - return valLeft operator valRight



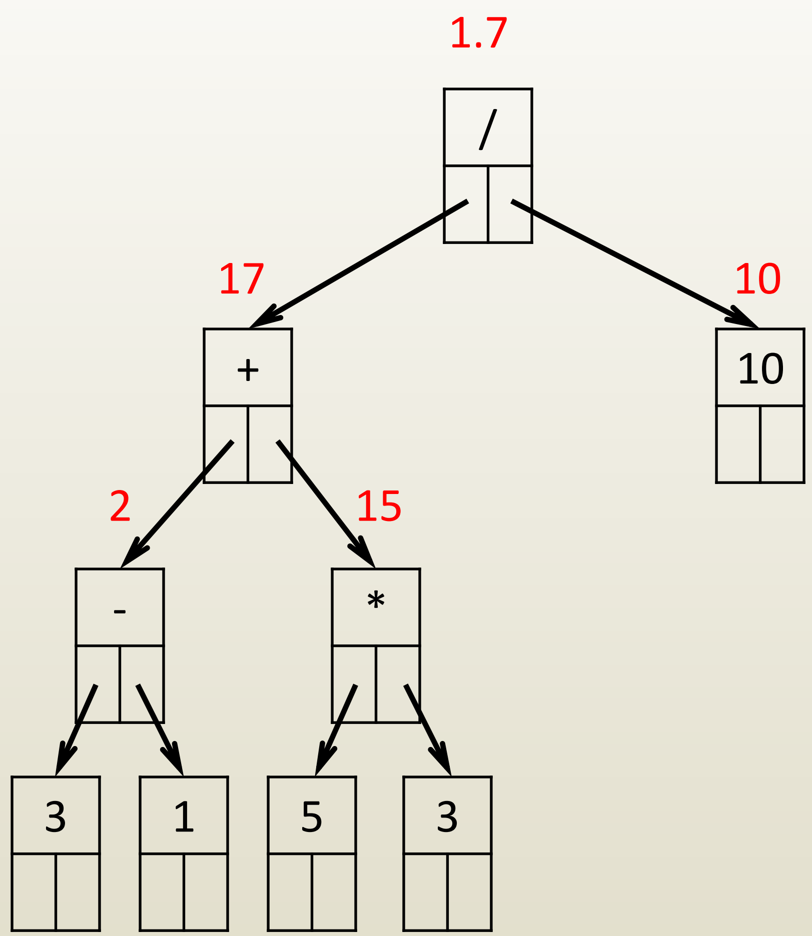
Arithmetic expression binary tree

- Arithmetic expression can be evaluated by traversing tree
- Evaluation rules
  - if leaf, return operand
  - valLeft = Evaluate left
  - valRight = Evaluate right
  - return valLeft operator valRight



Arithmetic expression binary tree

- Arithmetic expression can be evaluated by traversing tree
- Evaluation rules
  - if leaf, return operand
  - valLeft = Evaluate left
  - valRight = Evaluate right
  - return valLeft operator valRight



iClicker question

- Which traversal called COUNT counts the number of leafs in a binary tree.

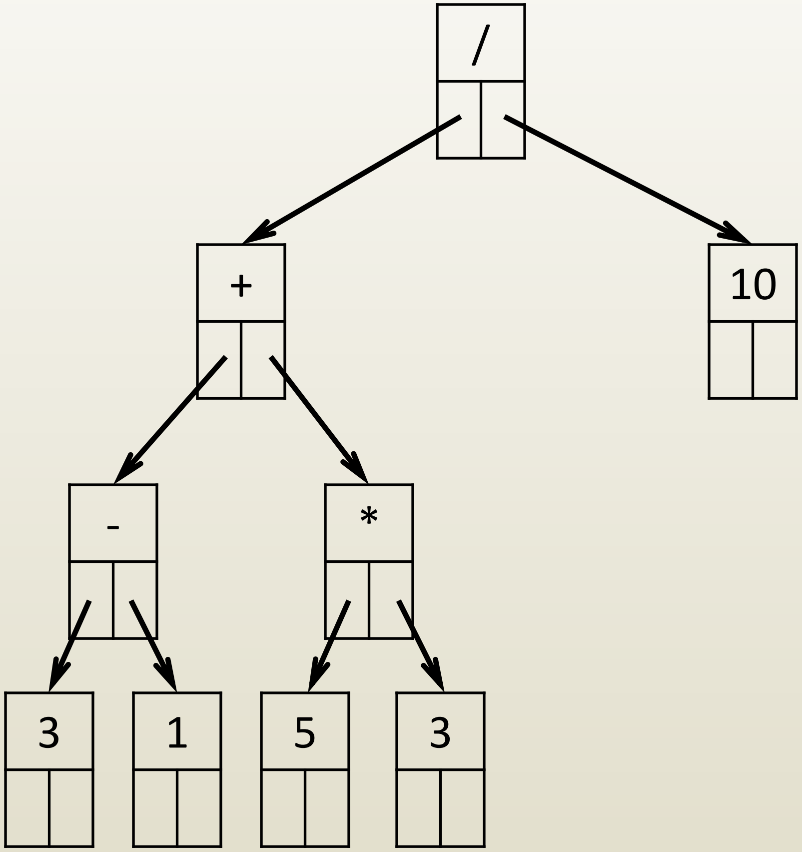
A. If leaf, return 1. If not leaf, return COUNT(left child) + COUNT(right child)

B. If leaf, return 1. If not leaf return 0. COUNT(left child). COUNT(right child).

C. If leaf, return operand. If not leaf, return COUNT(left child) + COUNT(right child)

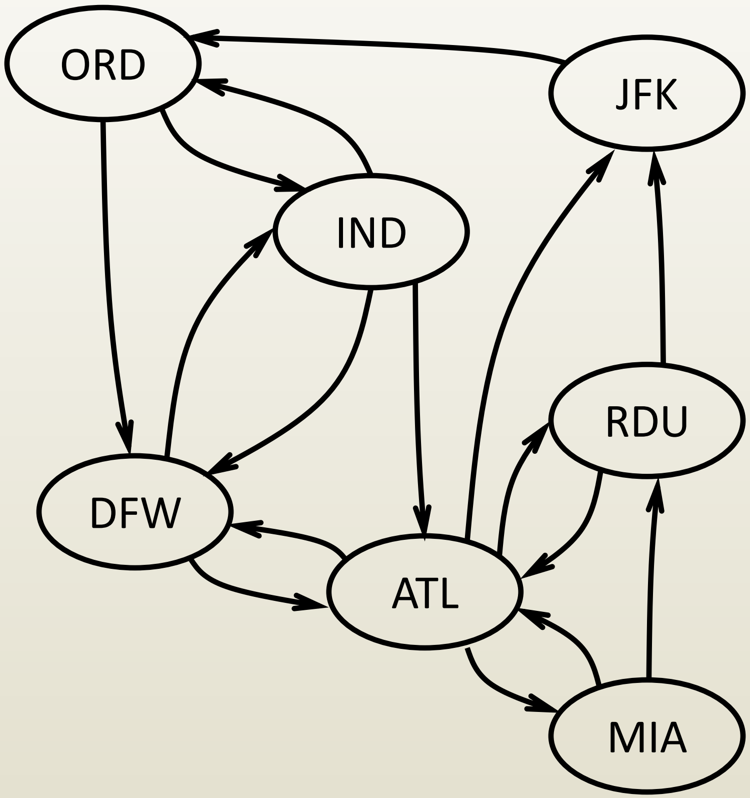
D. None of the above.

E. All of the above.



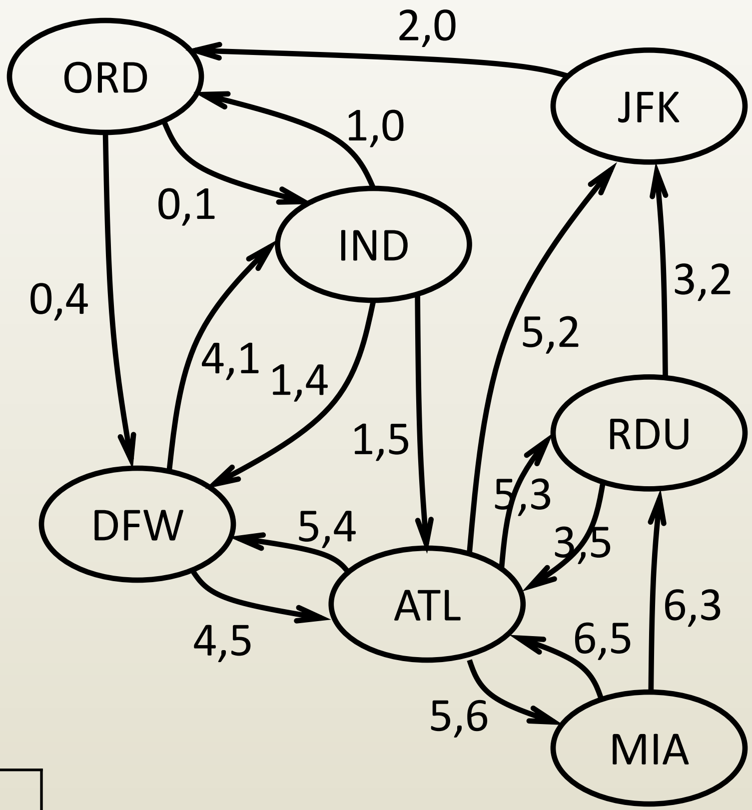
Graphs

- Graphs
  - Nodes (also called vertices) connected by links, called edges
  - Nodes have a variable number of incident edges
  - Great flexibility
- Example: airline routes



Graph data structure encodings

- "List of edges"
  - Array of nodes & array of edges
  - Edges pair of node indices
  - Origin node first
  - Destination node second

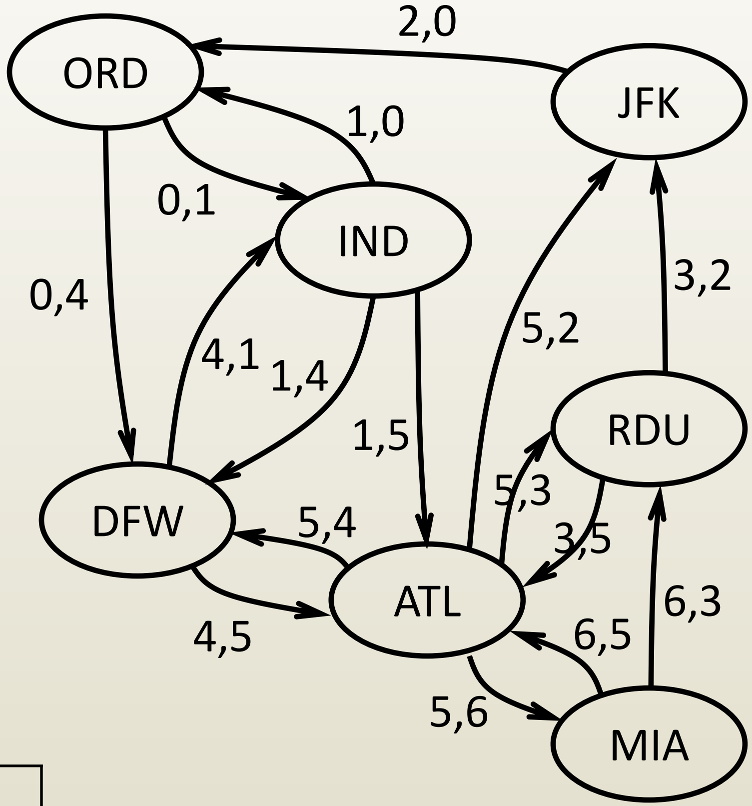


0	1	2	3	4	5	6
ORD	IND	JFK	RDU	DFW	ATL	MIA

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0,4	2,0	3,2	6,3	5,6	4,5	6,5	5,4	4,1	1,4	1,5	5,3	3,5	5,2	1,0	0,1

Graph data structure encodings

- "List of edges"
  - Used only for sparse graphs (i.e. a small number of edges)
  - Difficult to find whether there is an edge between two nodes (requires traversal of edge list)

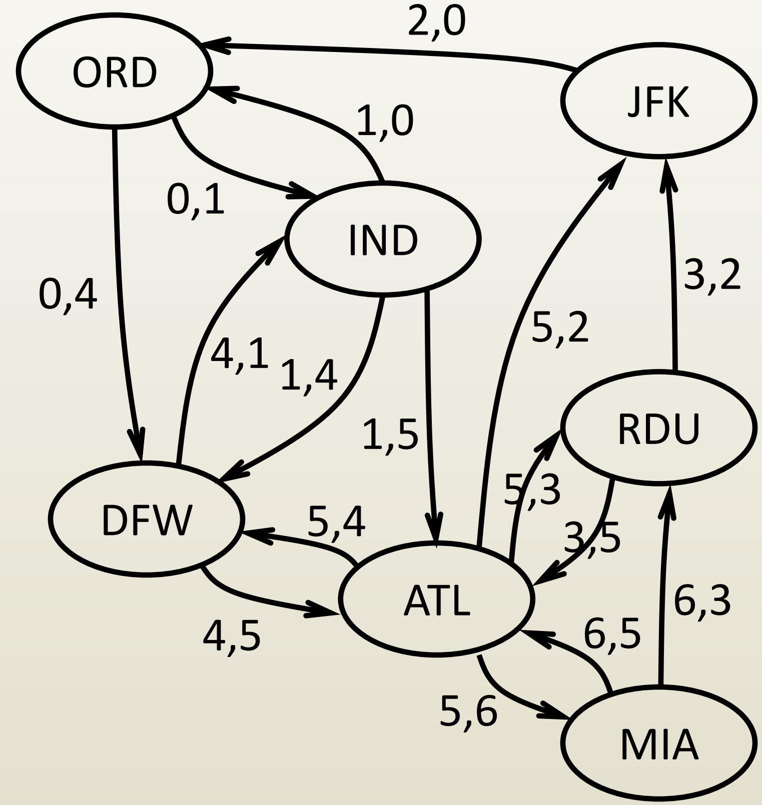


0	1	2	3	4	5	6
ORD	IND	JFK	RDU	DFW	ATL	MIA

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0,4	2,0	3,2	6,3	5,6	4,5	6,5	5,4	4,1	1,4	1,5	5,3	3,5	5,2	1,0	0,1

Graph data structure encodings

- "Adjacency lists"
  - One array for each node
  - Array stores adjacent nodes



0	1	2	3	4	5	6
ORD	IND	JFK	RDU	DFW	ATL	MIA

Adjacency list for node 0

0	1
1	4

Adjacency list for node 1

0	1	2
0	4	5

Adjacency list for node 2

0
0

Adjacency list for node 3

0	1
2	5

Adjacency list for node 4

0	1
1	5

Adjacency list for node 5

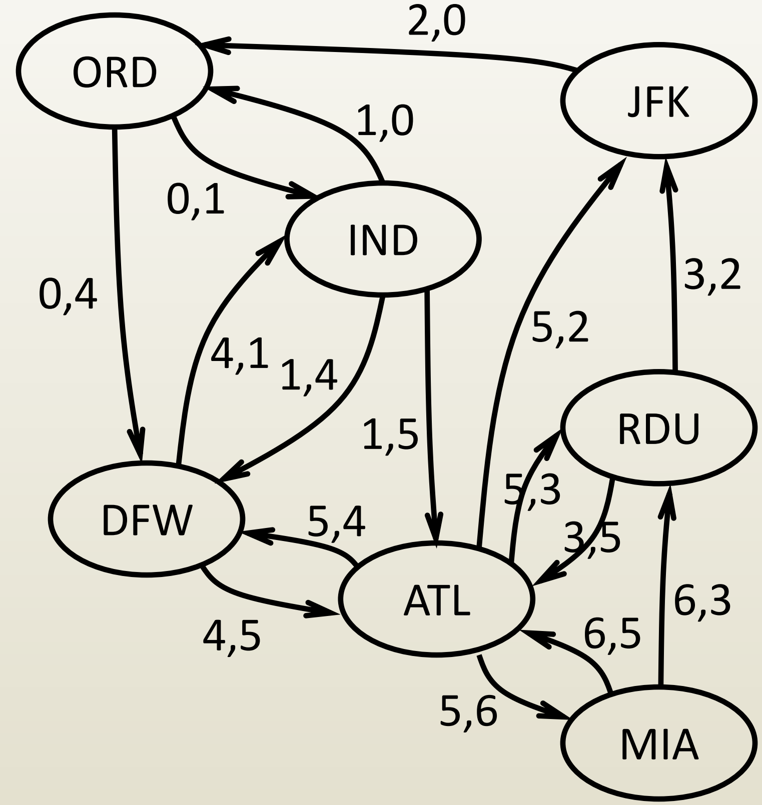
0	1	2	3
2	3	4	6

Adjacency list for node 6

0	1
3	5

Graph data structure encodings

- "Adjacency lists"
  - Finding an edge only requires traversing the starting node's adjacency list



0	1	2	3	4	5	6
ORD	IND	JFK	RDU	DFW	ATL	MIA

Adjacency list for node 0

0	1
1	4

Adjacency list for node 1

0	1	2
0	4	5

Adjacency list for node 2

0
0

Adjacency list for node 3

0	1
2	5

Adjacency list for node 4

0	1
1	5

Adjacency list for node 5

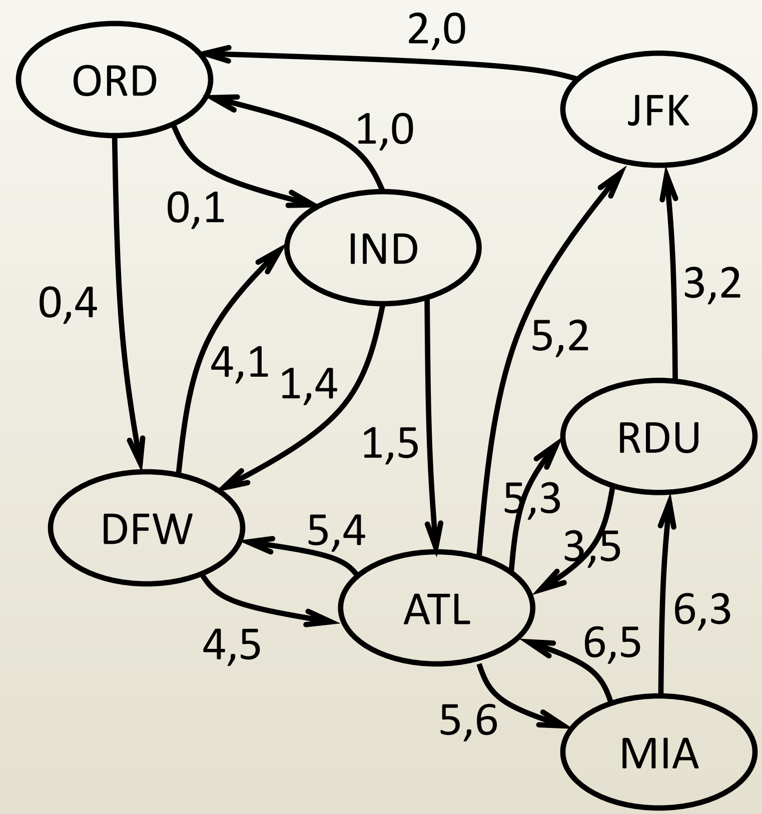
0	1	2	3
2	3	4	6

Adjacency list for node 6

0	1
3	5

Graph data structure encodings

- "Adjacency matrix"
  - A 2-D matrix
  - Row corresponds to start node
  - Column corresponds to end node
  - 0 if no edge, 1 if edge

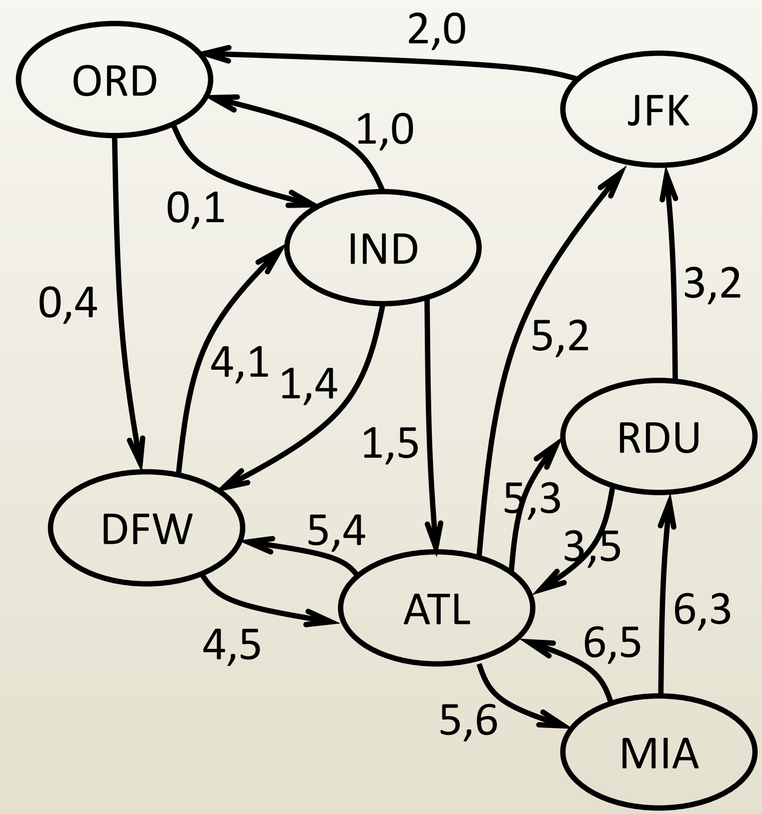


0	*1*	*2*	*3*	*4*	*5*	*6*
ORD	IND	JFK	RDU	DFW	ATL	MIA

	0	*1*	*2*	*3*	*4*	*5*	*6*
0	0	1	0	0	1	0	0
1	1	0	0	0	1	1	0
2	1	0	0	0	0	0	0
3	0	0	1	0	0	1	0
4	0	1	0	0	0	1	0
5	0	0	1	1	1	0	1
6	0	0	0	1	0	1	0

Graph data structure encodings

- "Adjacency matrix"
  - An edge is found in constant time
    - Is there an edge between ATL and ORD?
    - A[5][0] is 0 so the answer is no
  - Storage quadratic in number of nodes
    - Inefficient for sparse graphs

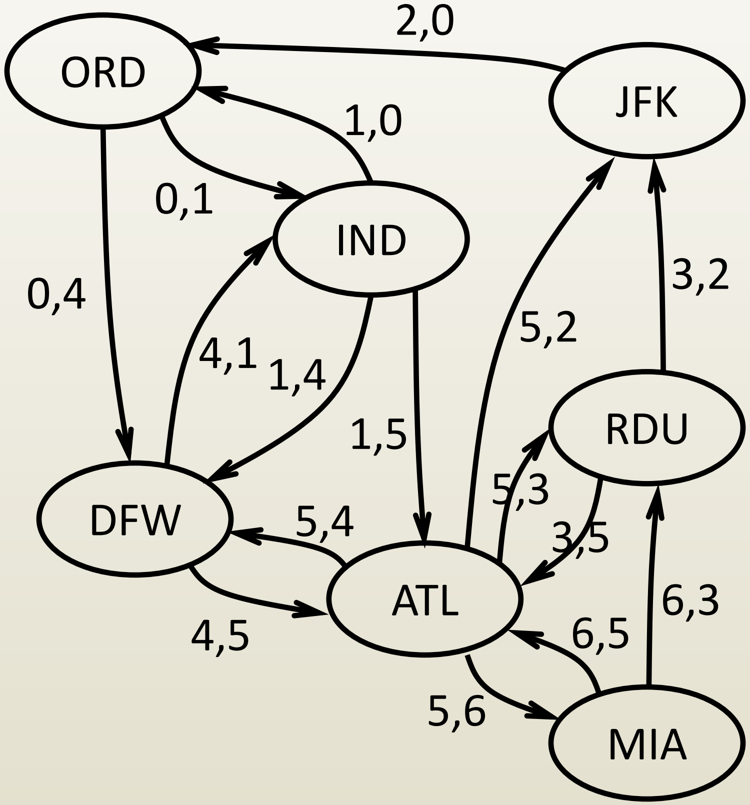


0	1	2	3	4	5	6
ORD	IND	JFK	RDU	DFW	ATL	MIA

	0	1	2	3	4	5	6
0	0	1	0	0	1	0	0
1	1	0	0	0	1	1	0
2	1	0	0	0	0	0	0
3	0	0	1	0	0	1	0
4	0	1	0	0	0	1	0
5	0	0	1	1	1	0	1
6	0	0	0	1	0	1	0

Directed graphs

- So far we talked about directed graphs
  - an edge started from one node and ended at another
  - the edge could only be traversed in one direction



Undirected graphs

- In an undirected graph edges can be traversed in either direction.