VE281

Data Structures and Algorithms

Binary Tree Traversal

Midterm

- 10:00 am -- 11:40 am, Nov. 8th, 2012.
- Closed book and closed notes.
- Written exam.
- No electronic devices are allowed.
  - These include laptops and cell phones.
  - We will show a clock on the screen.
- Abide by the Honor Code!

Review

- Hash Table Size and Rehashing
  - Amortized analysis of rehashing
- Trees
  - Root, leaf, subtree, parent, child, sibling, path
  - Depth, height, level, degree of a node/tree
- Binary Trees
  - The relation between the number of nodes and the height.
  - Proper, complete, and perfect binary trees.
  - Numbering nodes in a perfect binary tree.
  - Representing a binary tree.

Outline

- Binary Tree Traversal

Binary Tree Traversal

- Many binary tree operations are done by performing a traversal of the binary tree.
- In a traversal, each node of the binary tree is visited exactly once.
- During the visit of a node, all actions (making a clone, displaying, evaluating the operator, etc.) with respect to this node are taken.

Binary Tree Traversal Methods

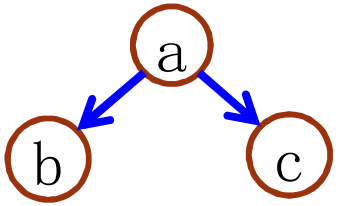
- Depth-first traversal
  - Pre-order
  - Post-order
  - In-order

- Level order traversal

Pre-Order Depth-First Traversal

Procedure

- Visit the node
- Visit its left subtree
- Visit its right subtree

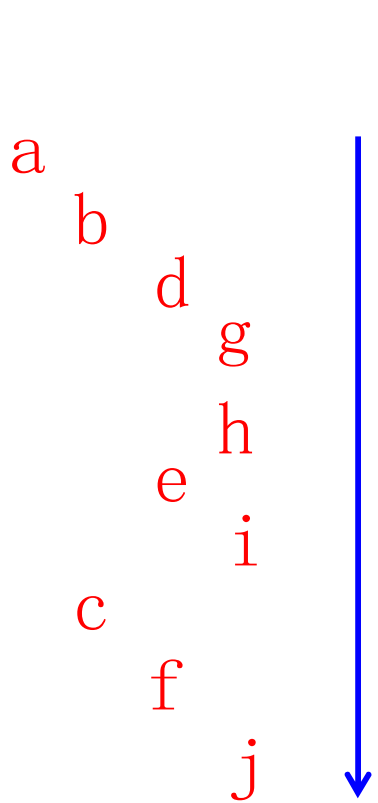


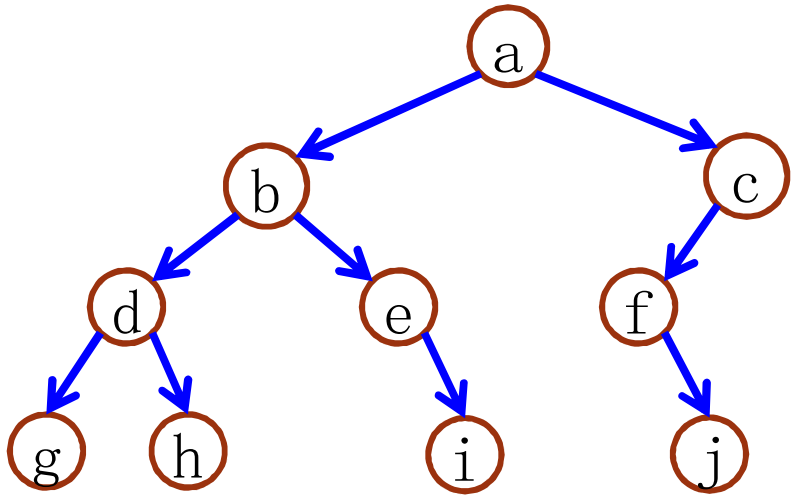
a→b→c

```
void preOrder(node *n) {
  if(!n) return;
  visit(n);
  preOrder(n->left);
  preOrder(n->right);
}
```

Pre-Order Depth-First Traversal

Example



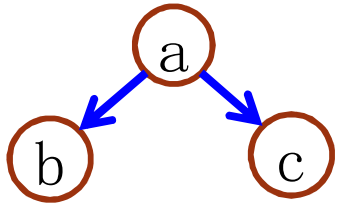


a→b→d→g→h→e→i→c→f→j

Post-Order Depth-First Traversal

Procedure

- Visit the left subtree
- Visit the right subtree
- Visit the node

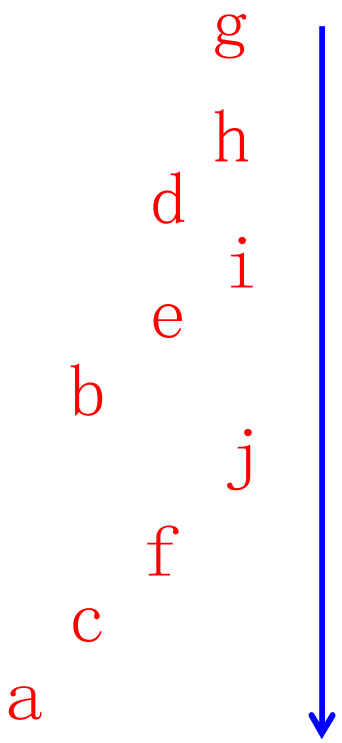


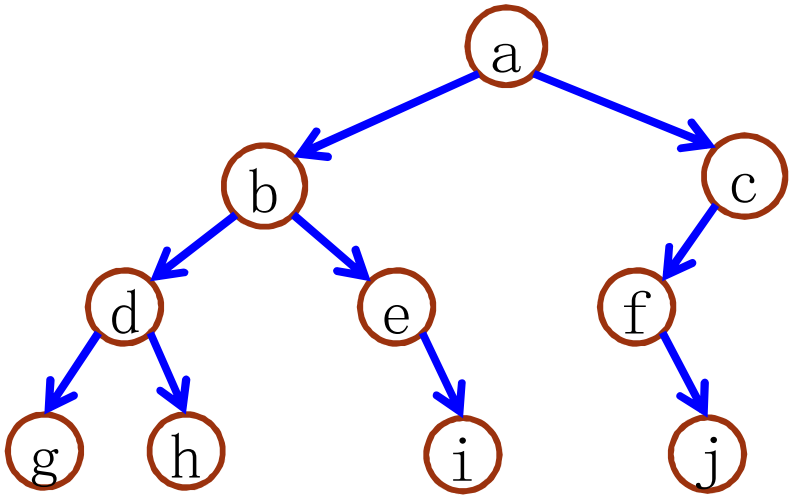
b→c→a

```
void postOrder(node *n) {
  if(!n) return;
  postOrder(n->left);
  postOrder(n->right);
  visit(n);
}
```

Post-Order Depth-First Traversal

Example



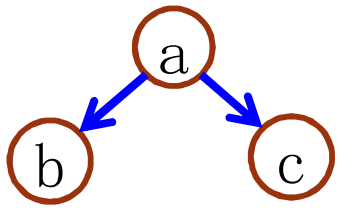


g→h→d→i→e→b→j→f→c→a

In-Order Depth-First Traversal

Procedure

- Visit the left subtree
- Visit the node
- Visit the right subtree

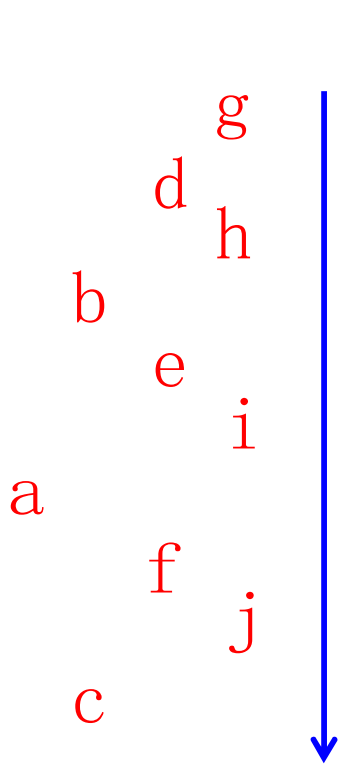


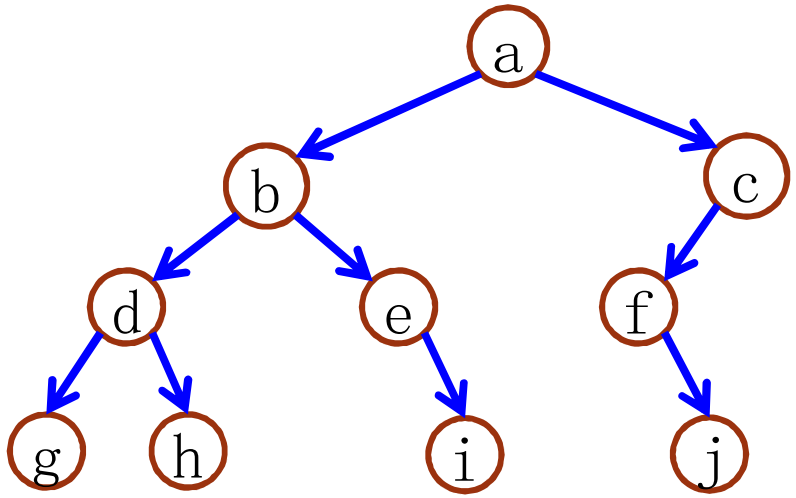
b→a→c

```
void inOrder(node *n) {
  if(!n) return;
  inOrder(n->left);
  visit(n);
  inOrder(n->right);
}
```

In-Order Depth-First Traversal

Example





g→d→h→b→e→i→a→f→j→c

Pre-Order Depth-First Traversal

Alternative Implementation

- Can we implement pre-order depth-first traversal without using recursion?
- Answer: use a stack.

1. Push the root node into an empty stack.
2. While the stack is not empty, pop a node from the stack.
   1. Visit the node.
   2. Push its **right** child (if exists) into the stack.
   3. Push its **left** child (if exists) into the stack.

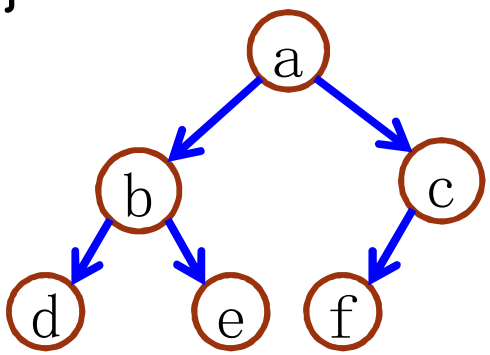
Loop

Note the orde

Pre-Order Depth-First Traversal using Stack

Code and Example

```
void preOrder(node *root) {
  stack s; // Empty stack
  s.push(root);
  while(!s.isEmpty()) {
    node *n = s.pop();
    visit(n);
    if(n->right) s.push(n->right);
    if(n->left) s.push(n->left);
  }
}
```



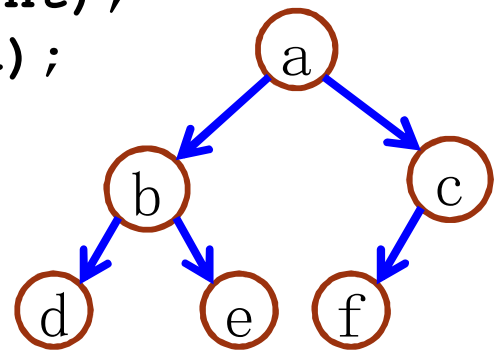
Stack: a c b e d f

Output: a b d e c f

Post-Order Depth-First Traversal using Stack

- How can you implement post-order depth-first traversal using stack?
  - Will the following code work?

```
void postOrder(node *root) {
  stack s; // Empty stack
  s.push(root);
  while(!s.isEmpty()) {
    node *n = s.pop();
    if(n->right) s.push(n->right);
    if(n->left) s.push(n->left);
    visit(n);
  }
}
```



Post-Order Depth-First Traversal using Stack

- We add an entry **visited** to the node struct:

```
struct node {
   Item item;
   bool visited;
   node *left;
   node *right;
};
```

- Initial value of **visited** is false.
- **visited** becomes true only when all the nodes in a tree rooted at this node have been visited.

Pre-Order Depth-First Traversal using Stack

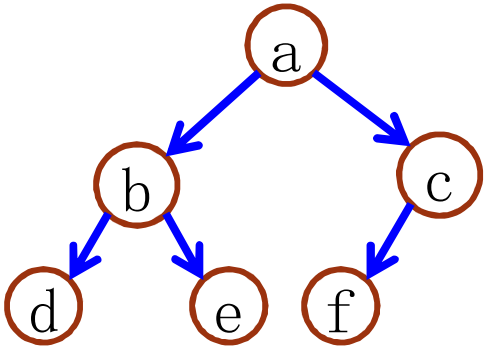
Procedure

1. Push the root node into an empty stack.
2. While the stack is not empty, we **examine (not POP)** the top node of the stack:
   1. If its left child exists and the **visited** entry of its left child is false, push the left child into the stack.
   2. **Otherwise**, if its right child exists and the **visited** entry of its right child is false, push the right child into the stack.
   3. **Otherwise**, print the top node, set its **visited** entry to true, and pop it from the stack.

Post-Order Depth-First Traversal using Stack

Code and Example

```
void postOrder(node *root) {
  stack s; // Empty stack
  s.push(root);
  while(!s.isEmpty()) {
    node *n = s.top();
    if(n->left && !n->left->visited)
      s.push(n->left);
    else if(n->right && !n->right->visited)
      s.push(n->right);
    else {
      print(n);
      n->visited = true;
      s.pop();
    }
  }
}
```



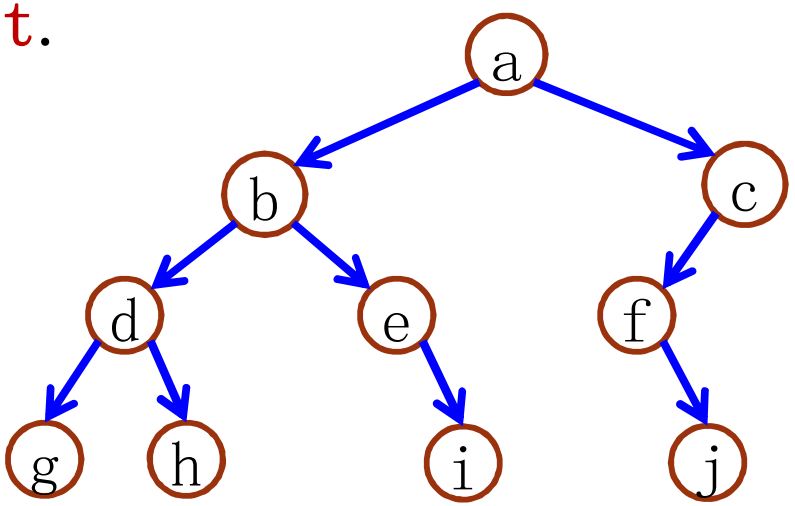
Stack: a b d e c f

Output: d e b f c a

node:	a	b	c	d
visited:	t	t	t	t	t

Level-Order Traversal

- We want to traverse the tree level by level **from top to bottom**.
- Within each level, traverse **from left to right**.



How can we implement this traversal?

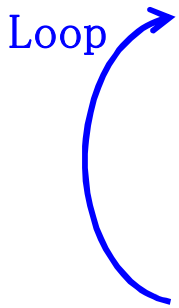
a→b→c→d→e→f→g→h→i→j

Level-Order Traversal

Procedure

- Use a queue!

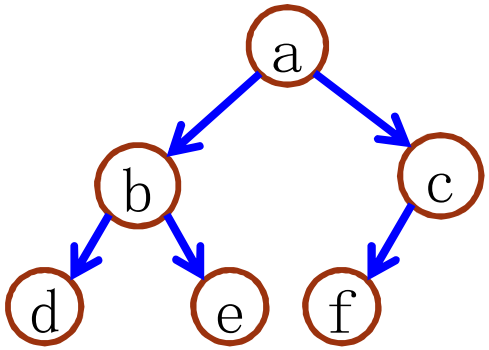
1. Enqueue the root node into an empty queue.
2. While the queue is not empty, dequeue a node from the front of the queue.
    1. Visit the node.
    2. Enqueue its left child (if exists) and right child (if exists) into the queue.



Level-Order Traversal

Code and Example

```
void levelOrder(node *root) {
  queue q; // Empty queue
  q.enqueue(root);
  while(!q.isEmpty()) {
    node *n = q.dequeue();
    visit(n);
    if(n->left) q.enqueue(n->left);
    if(n->right) q.enqueue(n->right);
  }
}
```

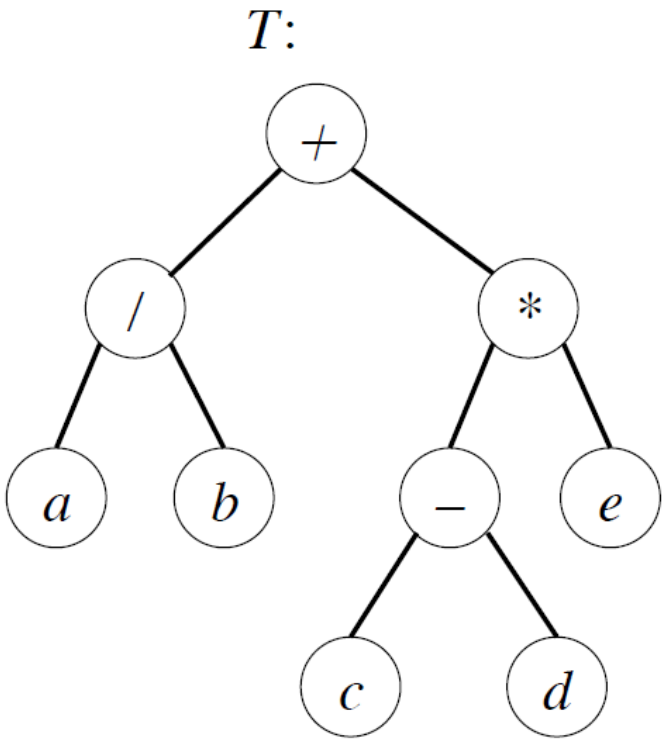


Queue: a b c d e f

Output: a b c d e f

Binary Tree Traversal

- The expression $a/b + (c - d)e$ has been encoded as a tree T.
  - The leaves are **operands**.
  - The internal nodes are **operators**.
- How would you traverse the tree T to print out the expression?
  - In-order depth-first traversal.



- What is the expression printed out by post-order depth-first traversal?
  - $ab/cd - e * +$
  - **Reverse Polish Notation**