CS112: Data Structures

Lecture 9

Uses of trees

Exam 1 results

- **Problem 2 was a large part of the problem**
- **Adding 10 points,**
- **Also making 20 extra credit**
  - **percentage is out of 130**
- **Sakai will stay the raw score**

Confusion

- **“Each element of A has a 1/4 probability of being found in B.”**

  Is NOT the same as

- **“1/4 of the elements in A will be in B.”**

Example: toss two coins

- Toss two fair coins - a “double flip”
- Do it 4 times - a “trial”
- Probability of double flip heads-heads = 1/4
- Does not mean that exactly one of the 4 double flips in a trial will always be heads-heads
- Does mean that the long run average number of heads-heads/trial approaches 1

Worst case: no probability

- **Probability only matters for average cost**
  - **"Probability of case j is 1/n"**
- **Worst case cost: find the one case with highest cost**
  - **No matter how likely or unlikely**

Example

- Unordered arrays A & B, may have repeats, both n elements
- For each element in A, check if also in B
- One worst case, n = 4:
  - A: 10, 10, 10, 10
  - B: 20, 23, 19, 10

Review
Built-in Hashing in Java

- The class java.util.HashMap<K, V>
  - Mapping from (unique) key to a value
  - Note: generic with two class parameters:
    - K: class of keys
    - V: class of values
  - E.g. Driver's license ID (String)
    => Driver object (name, address, etc.):
    java.util.HashMap<String, Driver>
  - See Driver.java and UseDriverMap.java

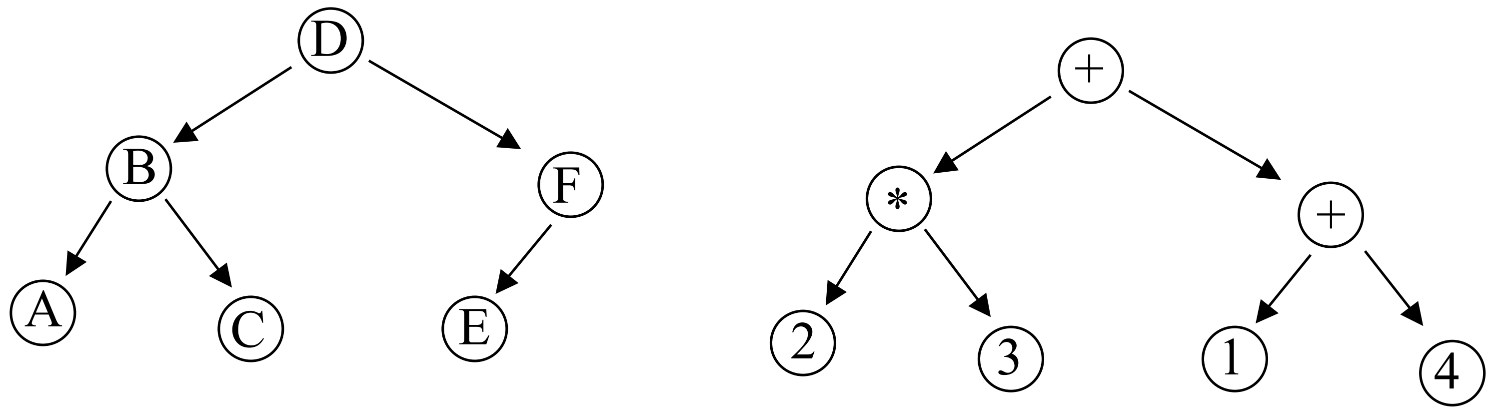
Traversals

```
preOrderPrint(tree):
   if (tree.root == null)
      return
   print(tree.node)
   preOrderPrint(tree.lst)
   preOrderPrint(tree.rst)
```

```
postOrderPrint(tree):
   if (tree.root == null)
      return
   postOrderPrint(tree.lst)
   postOrderPrint(tree.rst)
   print(tree.node)
```

```
inOrderPrint(tree):
   if (tree.root == null)
      return
   inOrderPrint(tree.lst)
   print(tree.node)
   inOrderPrint(tree.rst)
```

Traversals

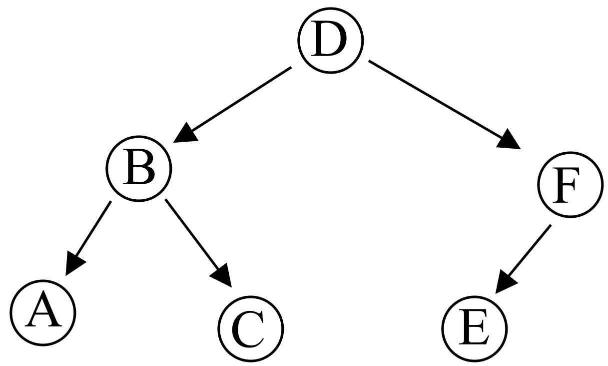


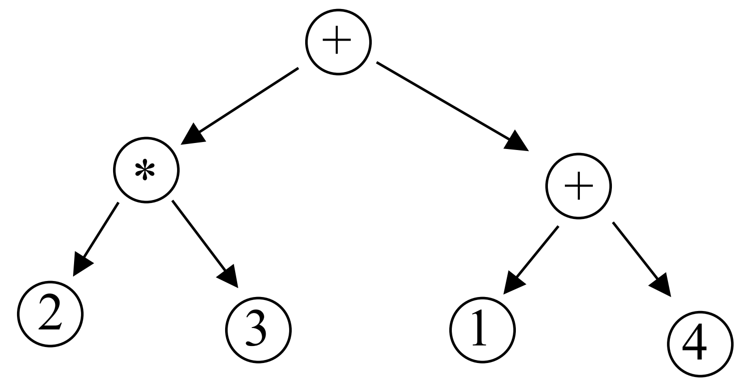
PreOrder ______________________________

InOrder ______________________________

PostOrder ______________________________

Traversals





PreOrder __D B A C F E + * 2 3 + 1 4

InOrder __A B C D E F 2 * 3 + 1 + 4

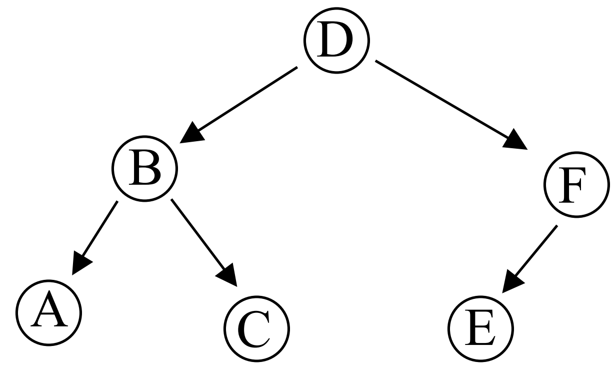
PostOrder A C B E F D 2 3 * 1 4 + +

Non-recursive Traversals

- **Problem: If a node has more than one child**
  - **can't work on all children, grandchildren, … at once**
  - **have to store children that have been found but not processed**
- **Solution: store in stack or queue**

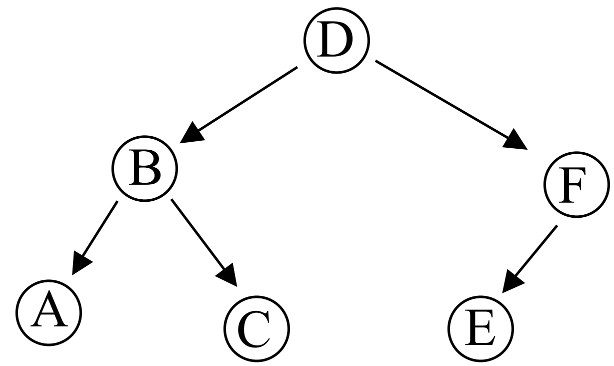
Stack-based Traversal

```
push(root);
while(! isEmpty( )){
  next = pop( );
  if (next != null){
    print next.data;
    push next.rightSubTree;
    push next.leftSubTree;
}}
```



Queue-based Traversal

```
enqueue root;
while(! isEmpty( )){
  next = dequeue( );
  if (next != null){
    print next.data;
    enqueue next.leftSubTree;
    enqueue next.rightSubTree
}}
```



Breadth vs Depth first

- Stack: depth first
  - do all children before anything else
- Queue: breadth first
  - do all at same level before anything else

Size of Stack / Queue

- **Stack: path from root to leaf: O(depth)**
- **Queue: entire level: $O(2^{depth})$**
  - **That's a lot!**
  - **Solution: Iterative Deepening**

Iterative Deepening

- **print all nodes at depth d:**

```
idfs(tree, d)
    if d == 0
        print tree.data
    else idfs(tree.lst, limit-1)
         idfs(tree.rst, limit-1)
```

- **Try all depths**

```
for(j=0; j<maxDepth; j++)
    idfs(tree, j)
```

Iterative Deepening

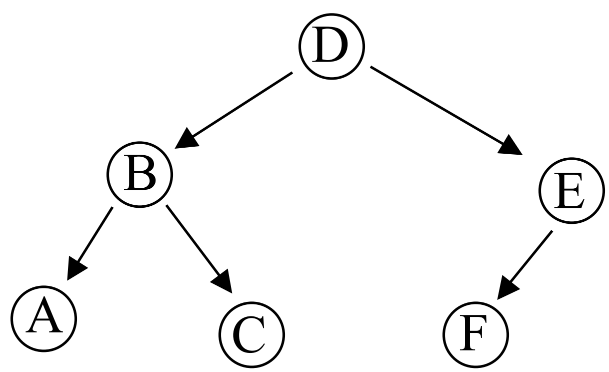
- **How much extra work?**
  - **How many leaves in complete binary tree of depth d? 2^d**
  - **How many non-leaves: 2^d-1**
- **Time overhead: roughly a factor of 2**

Signature of a Binary Tree

- Signature of a data structure
  - Store off line
  - Use later to reconstruct the data structure
- For binary tree: can we use traversals?
  - No traversal by itself is enough to reconstruct a tree
  - But combination of preorder and inorder will do the job

Traversals -> Tree

- **Preorder: D B A C E F**
- **Inorder: A B C D F E**



Traversals -> Tree

- **Preorder: D B A C E F**
- **Inorder: A B C D F E**

- **First node in preorder is root of the tree**

Traversals -> Tree

- **Preorder: D B A C E F**
- **Inorder: A B C D F E**
- **First node in preorder is root of the tree**
- **Everything in inorder to left of root is left subtree, so recur**
- **Everything in inorder to right of root is right subtree, so recur**

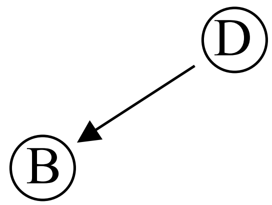
You draw the tree

Pre D B A C E F G H I

In A B C D F E H G I

Another Signature

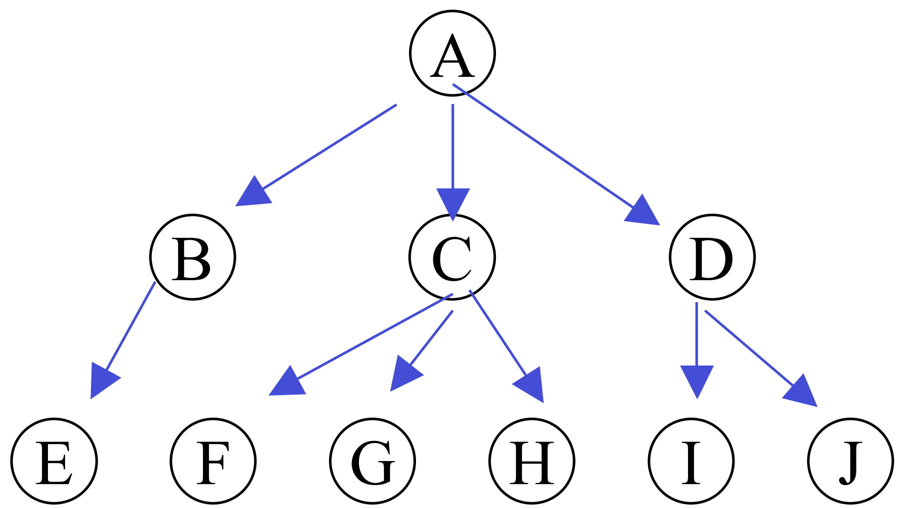
- **Imagine a function**
  newNode(data, leftSTree, rightSTree)



newNode(D, newNode(B, null, null), null)

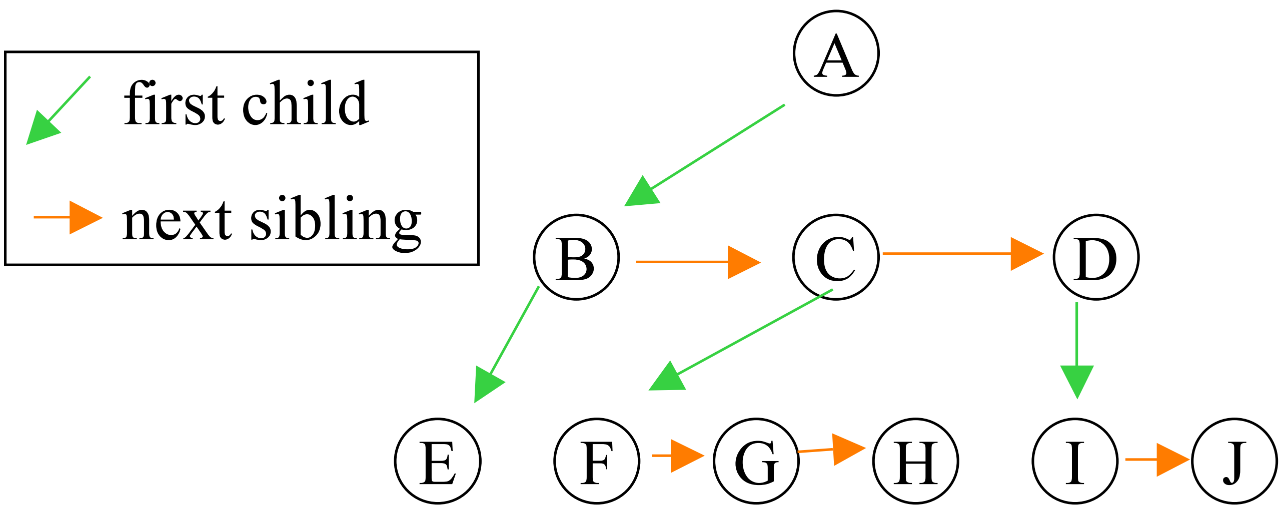
New: General Trees

- Each node has an arbitrary number of children
- Problem: representation of a node



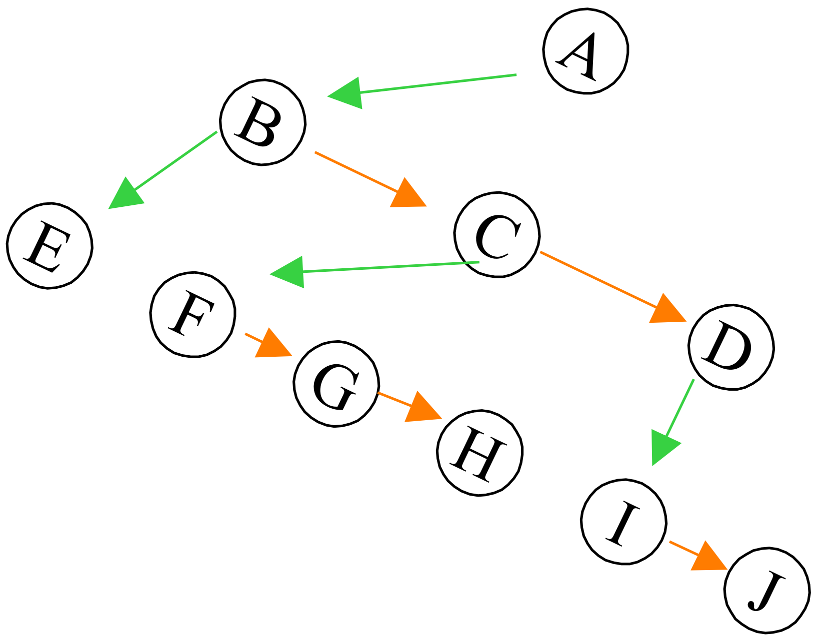
General Trees

- **Each node has an arbitrary number of children**
- **Problem: representation**
- **Solution: linked list of children**



General Tree as Binary

- First child <=> Left child
- Next sib <=> Right child



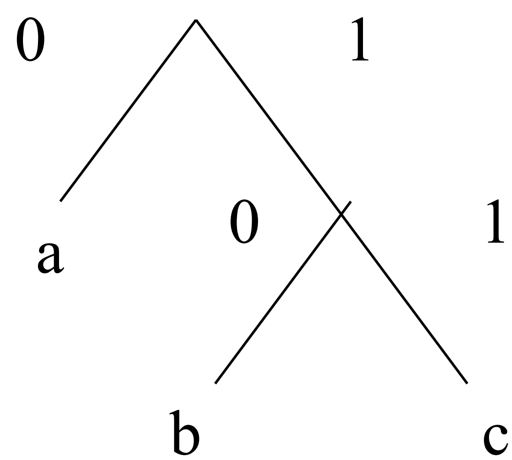
Data Compression

- **In most data some symbols appear more often than others**
  - **Eg English text ‘e’ appears more often than ‘q’**
- **In ascii code, each character is 8 bits.**
- **Suppose we had a code in which common symbols took fewer bits and uncommon symbols took more bits**

Huffman Code

- **EG 3 symbols: a, b, c, with a most frequent**
- **Code: 0 = a, 10 = b, 11 = c**
  - **abacac = 010011011**
  - **9 bits = 1.5 bits/character,**
  - **Versus 2 bits/character for fixed length code**
- **Decode: 11010 = cab**
- **Suppose code was 1 = a, 10 = b, 11 = c**
  - **Is 111 ca or ac or aaa?**
  - **No character's code can be prefix of another**

Huffman Code as a Tree



Symbols only at leaves

Algorithm

- **2 queues: S initially holds 1-node trees for all symbols, least weight = total probability first**
  T empty

while S not empty

find two least-weight trees in S, T and dequeue them

make a tree with these two as subtrees

enqueue on T

Example

A	.05
B	.1
C	.1
D	.2
E	.25
F	.3

Book code

- **See Huffman.java and HuffmanDriver.java in dsoi.progs.src.zip in apps/tree/**