

For Wednesday

- Read Weiss, chapter 7, sections 1-3
- Homework
 - Weiss, chapter 4, exercise 8. Make sure you include parentheses where needed.

Programming Assignment 1

- Any questions?

Homework

Special Cases

- Full binary tree
 - A binary tree of height h that contains exactly $2^h - 1$ elements
 - In other words, if one element is added to the tree, the height must increase
- Complete binary tree

Linked Binary Trees

- A node is represented as a struct or a class
- Each node has two pointers to other nodes
- One pointer is to the Left child; the other is to the Right child
- An empty subtree is represented by a null pointer
- Sometimes it is convenient to include a pointer to the node's parent

Representation of Binary Trees

- We can represent binary trees in arrays
- We don't use index 0
- The root is at index 1
- Node i 's children are at indexes $2i$ and $2i+1$
- Advantages?
- Disadvantages?

Binary Tree Traversal

- In a binary tree **traversal**, we **visit** each node in the tree exactly once
- There are four different orders in which we often choose to visit the nodes
 - Preorder
 - Inorder
 - Postorder
 - Level order

Preorder Traversal

- ```
void PreOrder(BinTreeNode* tree)
{ // PreOrder traversal of tree
 if (tree) {
 Visit(tree); // visit the root
 PreOrder(tree->left) // do left
subtree
 PreOrder(tree->right) // do right
subtree
 }
```

# Inorder Traversal

- ```
void InOrder(BinTreeNode* tree)
{ // InOrder traversal of tree
  if (tree) {
    InOrder(tree->left) // do left
subtree
    Visit(tree); // visit the root
    InOrder(tree->right) // do right
subtree
  }
```

Postorder Traversal

- ```
void PostOrder(BinTreeNode* tree)
{ // PostOrder traversal of tree
 if (tree) {
 PostOrder(tree->left) // do left subtree
 PostOrder(tree->right) // do right subtree
 Visit(tree); // visit the root
 }
}
```

# Level Order Traversal

```
void LevelOrder(BinTreeNode* tree)
{ // PreOrder traversal of tree
 Queue q;
 while(tree) {
 Visit(tree); // visit tree
 // put children on the queue
 if (tree->left) q.Add(tree->left);
 if (tree->right) q.Add(tree->right);
 // get next node to visit
 if (q.IsEmpty())
 tree = NULL;
 else
 tree = q.Delete();
 }
}
```

# Priority Queues

- Same basic operations as a standard queue:
  - insert an item
  - delete an item
  - look at first item
  - check for empty queue
- But, order of item removal is not based on the order of item insertion (as in stacks and queues)
- Instead, each item has a **priority** associated with it

# Priority Queue ADT

- **AbstractDataType** *MaxPriorityQueue* {  
    **instances:**  
        finite collection of elements; each  
        with a priority  
    **operations:**  
        *Create()*  
        *Size()*  
        *Max()*  
        *Insert(element)*  
        *DeleteMax()*  
}

# Uses of a Priority Queue

- Operating systems
- Best first search
- Simulations
- Others?

# Implementation

- Unordered linear list
  - Insert time
  - Delete time
- Ordered linear list
  - Insert time
  - Delete time

# Min Tree

- A tree (binary or not)
- Each child has a value bigger than its parent
- Or each parent has a value smaller than any of its children (if any)
- So the smallest value in the tree is ?
- Maximum trees are simply reversed

# Heaps

- A minimum binary heap is a min tree that is also a complete binary tree
- Usually represented in an array
- Height of a complete binary tree in terms of  $N$ ?

# Heap Operations

- Insert
- DeleteMin
- DecreaseKey
- IncreaseKey
- Remove
- BuildHeap