CHAPTER 7

TREES

HOMEWORK

- Again, all homework is from the Exercises
  - No problems are from the Review Exercises

BINARY TREES

- Binary trees are quite useful
- Each node in a binary tree has 0, 1, or 2 children
- We can then refer to the left child and the right child

FULL BINARY TREES

- A full binary tree is a binary tree where each node has 0 or 2 children
- This is optimal for searching
- If there are i internal nodes, then there are
  - 2i + 1 nodes
    - Each node has two children, plus there's a root
  - i+1 terminal nodes
    - 2i+1 total vertices – i internal vertices = i+1 vertices

AN APPLICATION OF BINARY TREES-A SINGLE ELIMINATION TOURNAMENT

- This is an example of a binary tree
- How many matches are played?
  - This is the same as asking how many interior nodes there are
- If there are n contestants/teams, there are n terminal nodes
- There are n-1 internal nodes, and so, n-1 games

ANOTHER APPLICATION OF BINARY TREES-AN EXPRESSION TREE

- **We use the laws of algebra to build the tree**

BINARY SEARCH TREES

- A binary search tree is a binary tree where
  - Data in each left child is less than its parent’s
  - Data in each right child is greater than its parent’s
- How do we search for data in a binary search tree?
- What if the data is not there?

BALANCED TREES

- Binary search trees are useful for searching
- A balanced tree is one that does not have “uneven paths”
  - This means that no node has a child subtree that is more than one node longer than its other child subtree
- A balanced binary search tree is optimal for searching

AVL TREES

- **This is purely a programming topic**
- **When adding data to a binary search tree, we add a leaf**
- **Adding too many leaves can cause the tree to become unbalanced**
- **We then have to rebalance the tree by adjusting it**
- **An AVL tree is a binary search tree that is "self-balancing"**
  - **This means that every time a node is added or deleted, the code that manages the tree rebalances it**

TRAVERSING A BINARY TREE

- Often it's useful to "print" a tree
  - The tree will be printed across the page or in a series of rows
  - It is almost impossible to "draw" the tree
  - We have to print the nodes in some order
- Suppose you want to search to see if a node is in a tree
- Suppose each node holds a numeric value
  - The values could represent balances that you may want to adjust as a group
  - The may represent account numbers that you want to renumber
- In any case, we need to visit every node in the tree

TRAVERSALS

- We call this idea a traversal
  - This means a way of visiting every node in the tree
  - It's essentially a list of the nodes in some order
- These are the common orders we use
  - Inorder
  - Preorder
  - Postorder
  - Breadth first
  - Depth first
    - This is just another name for the Preorder traversal

INORDER, PREORDER, POSTORDER TRAVERSALS

- Inorder
  - Left child, node itself, right child
  - Of course, if a child is a subtree, visit it inorder recursively
- Preorder
  - Node itself, left child, right child (again recursively)
- Postorder
  - Left child, right child, node itself (again recursively)

DECISION TREES

- Decision trees are useful for making decisions
  - In a decision tree, the interior nodes are labeled with questions
  - The edges are labeled with answers
  - You follow the edges to arrive at a conclusion
- See the troubleshooting tree for an example
- A special case of a decision tree is a game tree
  - In a game tree, the interior nodes store the “state” of the game
  - The edges are labeled with moves that can be made in the game
    - The moves are the valid moves from the state stored in the node