EdX 6.00x Notes

Lecture 13:

- Tree definition
  - A tree consists of one or more nodes
    - A node typically has a value associated with it
  - Nodes are connected by branches
  - A tree starts with a root node
  - Except for leaves, each node has one or more children
    - We refer to a node which has a child as the parent node
  - In simple trees, no child has more than one parent, but the generalization (often called a graph) is also very useful

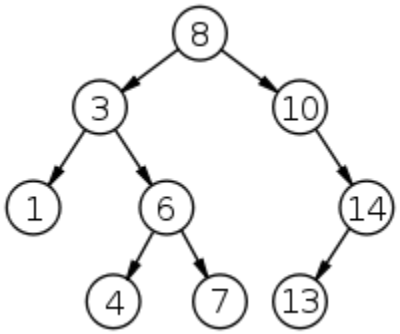
- Binary Trees
  - A binary tree is a special version of a tree, where each node has at most two children
  - Binary trees are very useful when storing and searching ordered data, or when determining the best decision to make in solving many classes of problems
    - Such trees are often called decision trees

- Searching a Tree
  - Depth First Search (DFS)
    - Start with the root
    - At any node, if we haven't reached our objective, take the elft branch first
    - When get to a leaf, backtrack to the first decision point and take the right branch
  - Breadth First Search (BFS)
    - Start with the root
    - Then proceed to each child at the next level, in order
    - Continue until reach objective

- Depth First Search for Containment
  - Idea is to keep a data structure (called a stack) that holds nodes still to be explored
  - Use an evaluation function to determine when reach objective (i.e. containment, whether value of node is equal to desired value)
  - Start with the root node
  - Then add children, if any, to front of data structure, with left branch first
  - Continue in this manner

- Stack vs. Queue
    - Stack: Last In, First Out (LIFO) data structure
        - Used in Depth First Search (DFS)
    - Queue: First In, First Out (FIFO) data structure
    - Used in Breath First Search (BFS)

- Ordered Search a.k.a. Binary Search Tree
    - Suppose we know that the tree is ordered, meaning that for any node, all the nodes to the "left" are less than that node's value, and all the nodes to the "right" are greater than node's value



- Decision Trees
    - A decision tree is a special type of binary tree (though could be more general tree with multiple children)
    - At each node, a decision is made, with a positive decision taking the left branch, and a negative decision taking the right branch
    - When we reach a node that satisfies some goal, the path back to the root node defines the solution to the problem captured by the tree
- Building a Decision Tree
    - One way to approach decision trees is to construct an actual tree, then search it
    - An alternative is to implicitly build the tree as needed
    - As an example, we will build that decision tree for a knapsack problem

- The Knapsack Problem
    - Suppose we are given a set of objects, each with a value an weight
    - We have a finite sized knapsack, into which we want to store some of the items
    - We want to store the items that have the most value, subject to the constraint that there is a limit to the cumulative size that will fit

- Building a Decision Tree (For Knapsack Problem)
    - For the knapsack problem, we can build a decision tree as follows:
        - At the root level, we decide whether to include the first element (left branch) or not (right branch)
        - At the nth level, we make the same decision for the nth element
        - By keeping track of what we have included so far, and what we have left to consider, we can generate a binary tree of decisions
- Power Set
    - A set of all the subsets of an item

- Decision Trees
    - Depth first and breath first still search the same number of nodes, the order is simply different
    - If we are willing to settle for "good enough", then there is a difference in work done by the two search methods
- Searching an Implicit Tree
    - Our approach is inefficient, as it constructs the entire decision tree, and then searches it
    - An alternative is only generate the nodes of the tree as needed
    - Here is an example for the case of a knapsack problem, the same idea could be captured in other search problems
- Notes:
    - We can have a decision tree that has more than two decisions per node.
    - Explicit search of a decision tree means the entire tree has to be built before beginning to search for an item.
    - Implicit search of a decision tree means that we build the entire tree and then remove nodes that we know will not be part of the path to the item.
- What if our trees are overgrown
    - We have been explicitly assuming that there are no "loops" in our trees, i.e. that a child has one parent, and that no node is the parent of a child closer to the root
    - What if we relax this constraint?
    - Generalization is called a graph
        - Lots of great graph search problems
        - For now, we can think about ways to support search for binary trees that might have loops

- Searching these “trees” (graphs)
    - What happens if we run depth first search on this?
        - An infinite loop in many cases when item present, and always if item not present.
    - What happens if we run breadth first search on this?
        - Inefficient as repeats nodes, but still works if item present, infinite loop if not present.

- Searching Graphs
    - With an item in the graph:
        - Depth-first search **must** keep a list of the nodes visited to avoid the possibility of infinite loops when searching for the item.
        - Breadth-first search **does not** need to keep a list of the nodes visited to avoid possible infinite loops when searching for the item.
    - With an item not in the graph:
        - Depth-first search **will not** run into an infinite loop if it uses a list of the nodes visited when searching for the item.
        - Breadth-first search will avoid running in an infinite loop if it makes use of a list of the nodes visited when searching for the item.