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 leaf 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 a decision tree for a knapsack problem

The Knapsack Problem

- Suppose we are given a set of objects, each with a value and a 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

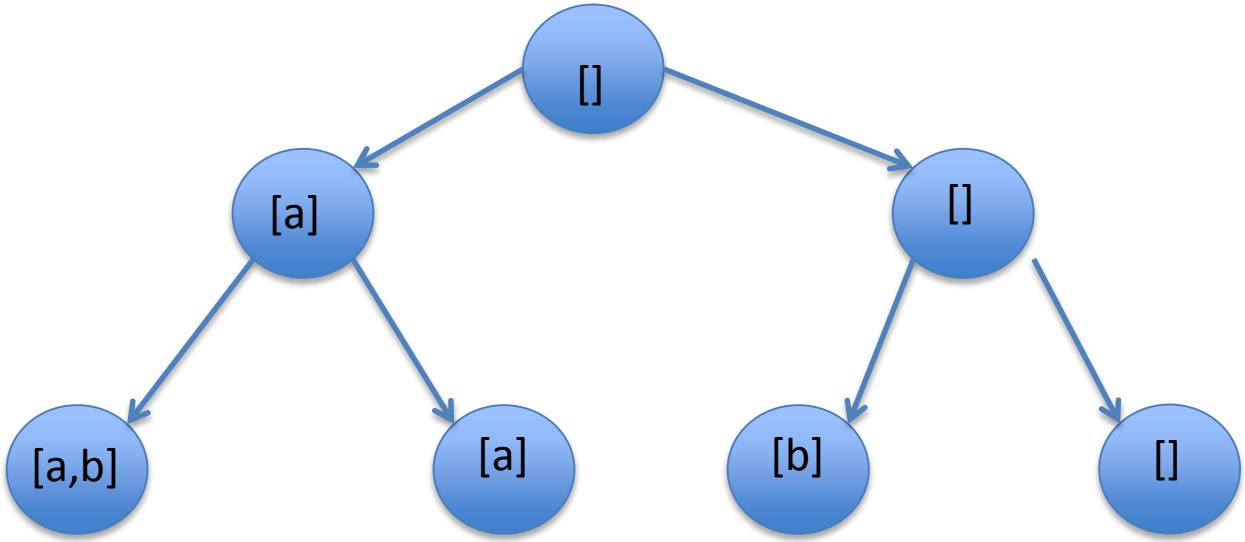
Building a decision tree

- For 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

Building a decision tree

```
def buildDTree(sofar, todo):
    if len(todo) == 0:
        return binaryTree(sofar)
    else:
        withelt = buildDTree(sofar + [todo[0]], todo[1:])
        withoutelt = buildDTree(sofar, todo[1:])
        here = binaryTree(sofar)
        here.setLeftBranch(withelt)
        here.setRightBranch(withoutelt)
        return here
```

Example decision tree



Given elements a and b, decide whether to include them

Searching a decision tree

```
def DFSDTree(root, valueFcn, constraintFcn):
    stack= [root]
    best = None
    visited = 0
    while len(queue) > 0:
        visited += 1
        if constraintFcn(stack[0].getValue()):
            if best == None:
                best = stack[0]
            elif valueFcn(stack[0].getValue()) > valueFcn(best.getValue()):
                best = stack[0]
            temp = stack.pop(0)
            if temp.getRightBranch():
                queue.insert(0, temp.getRightBranch())
            if temp.getLeftBranch():
                queue.insert(0, temp.getLeftBranch())
        else:
            stack.pop(0)
    print 'visited', visited
    return best
```

Searching a decision tree

```
def BFSDTree(root, valueFcn, constraintFcn):
    queue = [root]
    best = None
    visited = 0
    while len(queue) > 0:
        visited += 1
        if constraintFcn(queue[0].getValue()):
            if best == None:
                best = queue[0]
            elif valueFcn(queue[0].getValue()) > valueFcn(best.getValue()):
                best = queue[0]
            temp = queue.pop(0)
            if temp.getLeftBranch():
                queue.append(temp.getLeftBranch())
            if temp.getRightBranch():
                queue.append(temp.getRightBranch())
        else:
            queue.pop(0)
    print 'visited', visited
    return best
```

An example

```
a = [6,3]
b = [7,2]
c = [8,4]
d = [9,5]

treeTest = buildDTree([], [a,b,c,d])

def sumValues(lst):
    vals = [e[0] for e in lst]
    return sum(vals)

def WeightsBelow10(lst):
    wts = [e[1] for e in lst]
    return sum(wts) <= 10

DFSDTree(treeTest, sumValues, WeightsBelow10)
BFSDTree(treeTest, sumValues, WeightsBelow10)
```