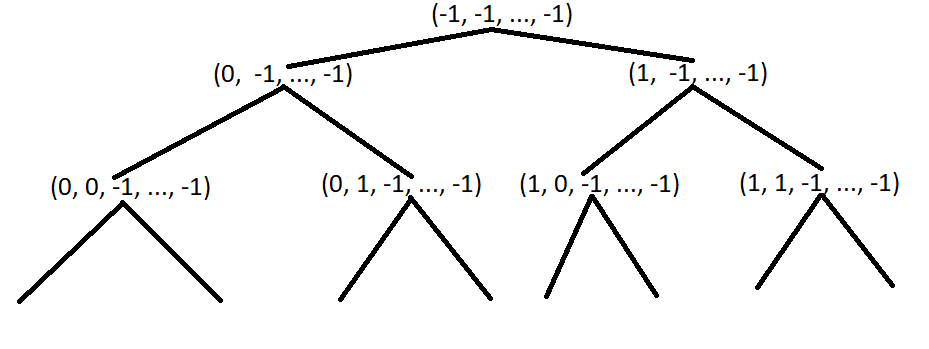
**Lab 5 Instructions**

In today’s lab you will use the A\* algorithm for solving the 0/1 knapsack problem.

Caution: Please do not use dynamic programming. That is not the purpose of this lab and will not be considered.

In this problem you are given N items. The profits and weights associated are p(1), p(2) … p(N) and w(1), w(2) … w(N). Also, the capacity of the knapsack is W. the problem is to pack the knapsack with the items such that the knapsack does not overflow. The objective is to maximize the total profit.

The problem space is defined by an array of length N where N is the number of items. The elements of the array is either 0 or 1. If the ith element is 0 then it implies that the item has not been included in the knapsack. If the ith element is 0 then it implies that the item has been included in the knapsack. Thus, the size of the problem space is 2N. in actual implementation you will use a third value for the matrix elements. This value can be -1. If the ith element is -1 then it implies that no decision has been made about that item. The search tree has a root node (-1, -1, …, -1). At the ith layer it assigns a value to the ith element. The first few layers of the tree is given in the figure below:



Note that the possible solutions are available only at the leaf nodes of the tree since we would have taken decisions on all items only at the leaf nodes. Also note that only some leaf nodes are feasible solutions since the others may violate the basic constraint that the total weight of included items can not exceed the capacity of the knapsack.

Recall that in A\* we use a heuristic that is a sum of the actual profit and an estimate of the maximum profit that can be obtained from the unassigned portion. Thus, if we want to calculate the heuristic value of a node say (1, 0, 1, -1, -1, …, -1) then it means that the actual profit accrued till now is p(1) + p(3) and the total capacity used is w(1) + w(3). To get an estimate of the maximum profit that we can get from the remaining capacity and the remaining items we will run the fractional knapsack problem for items (4 – N) and for a knapsack of capacity (W – w(1) – w(3)).

Define a solution vector (representing the problem space) as the array S, of length N. Initialize the elements as -1. Define current index, d, as 0. Define current\_weight as

Similarly, define current profit as

As a first step, learn to build the tree recursively. For each node calculate the current\_weight and current\_profit. In case the current\_weight is greater than the knapsack capacity, W, then the subtree below that node will not be explored further. Also, maintain a variable called current\_best\_profit and current\_best\_solution. The latter is an array of length N. Initialize current\_best\_profit to some large negative number. When ever your code reaches a leaf node, if the current\_profit is greater than the current\_best\_profit then update the value of the current\_best\_profit and also copy the elements of the solution vector, S, into the array current\_best\_solution. The procedure explained above essentially amounts to an un-informed exhaustive DFS search, with no heuristic.

As a second step, introduce the heuristic to prune the tree generated in the first step. The basic idea is that when ever we visit a node then we calculate maximum estimated profit using the fractional knapsack algorithm. This is calculated as the profit returned by fractional knapsack given the remaining capacity (i.e. W – current\_weight) and the items from (d+1) to N where d is the current depth. We add the current\_profit to the profit returned by the fractional knapsack to get the maximum estimated profit. Now, if the maximum estimated profit is less than the current\_best\_profit then we know that the node need not be expanded further. Hence, we do not descend that subtree any further (i.e. we do not make the recursive call from that state).

**Please show your work, even if it is partial, since every lab is being graded and grades will be awarded based on what you show during the lab.**