Algorithms Assignment 8

Clare Minnerath

Compare 0-1 Knapsack Problem:

The backtracking algorithm for the 0-1 knapsack problem goes thru the state space tree using a depth first search (preorder). The branch-and-bound approach gets rid of this requirement and allows for any traversal of the state space tree. The examples from the text are the breadth first search and the best-first-search which greedily chooses what promising node's children it will search next. Both algorithms determine a node is promising as long as its weight is less than the limit and its bound is greater than the current maximum profit. In larger instances, the best-first-search version of the branch-and-bound will likely search the least nodes in the state space tree. However, for certain instances other methods may prove superior.

33. Implementation: assign8.py

First Steps to algorithm:

1. Set maxprofit to 0
2. Visit node (0, 0) (the root).
(a) Compute its profit and weight: profit = 0, weight = 0.
(b) Compute its bound. Because 2 + 5 + 7 = 14, and 14 > 9, the value of W , the third item would bring the sum of the weights above W. Therefore, k = 2 (start @ index 0), and we have:
totalweight = 7, bound = 60
(c) Determine that the node is promising because its weight 0 is less than 16, the value of W , and its bound 60 is greater than 0, the value of maxprofit.

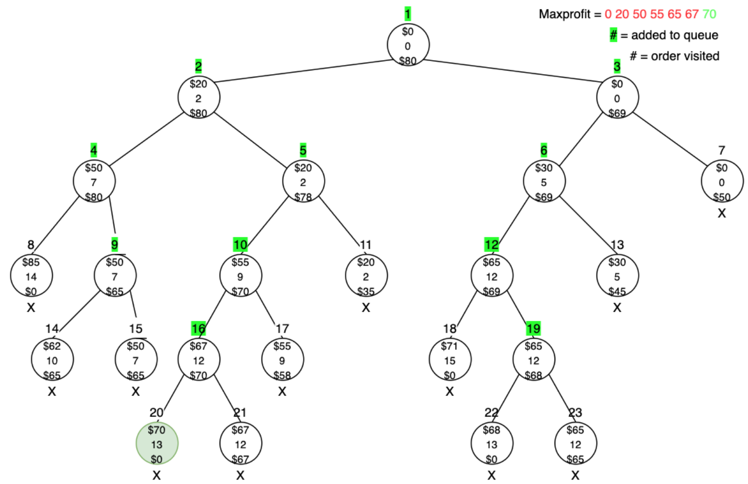
3. Visit node (1, 1)
(a) Compute its profit and weight: profit = 20, weight = 2
(b) Because its weight 2 is less than or equal to 9, the value of W, and its profit 20 is greater than 0, the value of maxprofit, set maxprofit to 20.
(c) We won't call promising/compute bound since we aren't at the $k - 1^{th}$ level and had moved to the left.

4. Visit node (2, 1)
(a) Compute its profit and weight: profit = 50, weight = 7
(b) Because its weight 7 is less than or equal to 9, the value of W, and its profit 50 is greater than 20, the value of maxprofit, set maxprofit to 50.
(c) We're at the $k - 1^{th}$ level so we won't go left. We'll only check tot the right since the k^{th} node is guaranteed to go over W.

5. Visit node (3, 1)
(a) Compute its profit and weight: profit = 50, weight = 7
(b) Because its weight 7 is less than or equal to 9, the value of W, and its profit 50 is equal to 50, the value of maxprofit stays the same.
(c) Compute its bound. Because 7 + 3 = 10, and 10 > 9, the value of W , the fourth item would bring the sum of the weights above W. Therefore, k = 3, and we have:
totalweight = 17, bound = 58

The pattern continues and the resulting solution is to take the 1st and 3rd weights for a profit of 55.

1. The algorithm is implemented as shown in the tree below. The final solution is weights: [2, 7, 3, 1] and profit = 70.



2. Implementation: assign8.py