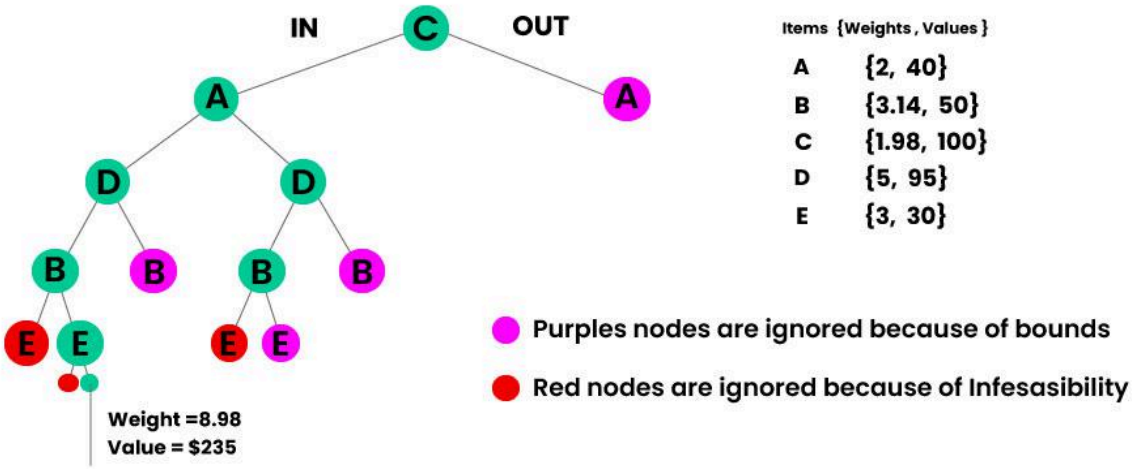
UNIT V – BRANCH AND BOUND AND BACKTRACKING

Backtracking: N-Queens problem - Hamiltonian cycles – Graph coloring – Sum of subset. Branch and bound: The method – FIFO branch and bound- LC branch and bound – 0/1 Knapsack problem - Traveling salesman problem.

0/1 KNAPSACK PROBLEM

Given two arrays v[] and w[] that represent values and weights associated with n items respectively. Find out the maximum value subset(Maximum Profit) of v[] such that the sum of the weights of this subset is smaller than or equal to Knapsack capacity W.

The backtracking based solution works better than brute force by ignoring infeasible solutions. We can do better (than backtracking) if we know a bound on the best possible solution subtree rooted with every node. If the best in subtree is worse than the current best, we can simply ignore this node and its subtrees. So we compute the bound (best solution) for every node and compare the bound with the current best solution before exploring the node.



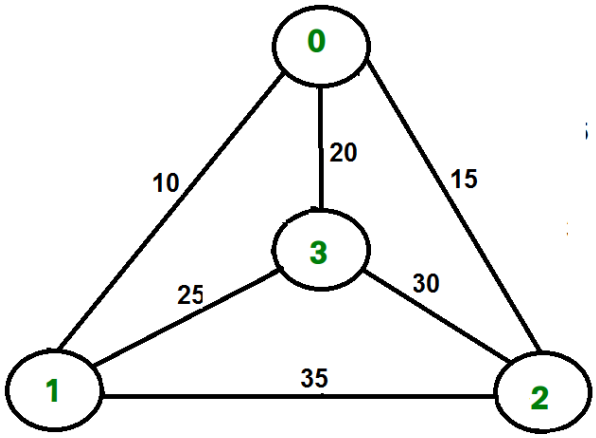
Follow the steps to implement the above idea:

- Sort all items in decreasing order of ratio of value per unit weight so that an upper bound can be computed using Greedy Approach.
- Initialize maximum profit, **maxProfit = 0**, create an empty queue, **Q**, and create a dummy node of decision tree and enqueue it to **Q**. Profit and weight of dummy node are **0**.
- Do the following while **Q** is not empty.
  - Extract an item from **Q**. Let the extracted item be **u**.
  - Compute profit of next level node. If the profit is more than **maxProfit**, then update **maxProfit**.

- Compute bound of next level node. If bound is more than **maxProfit**, then add the next level node to **Q**.
- Consider the case when the next level node is not considered as part of the solution and add a node to queue with level as next, but weight and profit without considering next level nodes.

TRAVELING SALESMAN PROBLEM

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.



Branch and Bound Solution

As seen in the previous articles, in the Branch and Bound method, for the current node in the tree, we compute a bound on the best possible solution that we can get if we down this node. If the bound on the best possible solution itself is worse than the current best (best computed so far), then we ignore the subtree rooted with the node.

Note that the cost through a node includes two costs.

1. Cost of reaching the node from the root (When we reach a node, we have this cost computed)
2. Cost of reaching an answer from current node to a leaf (We compute a bound on this cost to decide whether to ignore subtree with this node or not).

In cases of a maximization problem, an upper bound tells us the maximum possible solution if we follow the given node. For example in 0/1 knapsack we used the Greedy approach to find an upper bound.

In cases of a minimization problem, a lower bound tells us the minimum possible solution if we follow the given node. For example, in the Job Assignment Problem, we get a lower bound by assigning the least cost job to a worker.

In branch and bound, the challenging part is figuring out a way to compute a bound on the best possible solution. Below is an idea used to compute bounds for Travelling salesman problems. The cost of any tour can be written as below.

Cost of a tour T = (1/2) * ? (Sum of cost of two edges adjacent to u and in the tour T)
where u ? V

For every vertex u, if we consider two edges through it in T, and sum their costs. The overall sum for all vertices would be twice of cost of tour T (We have considered every edge twice.)

(Sum of two tour edges adjacent to u) >= (sum of minimum weight two edges adjacent to u)

Cost of any tour >= 1/2) * ? (Sum of cost of two minimum weight edges adjacent to u)
where u ? V

For example, consider the above shown graph. Below are minimum cost two edges adjacent to every node.

Node	Least cost edges	Total cost
0	(0, 1), (0, 2)	25
1	(0, 1), (1, 3)	35
2	(0, 2), (2, 3)	45
3	(0, 3), (1, 3)	45

Thus a lower bound on the cost of any tour =
1/2(25 + 35 + 45 + 45)
= 75
