Branch and Bound

→ This technique is mostly used to solve optimization problems.

→ In branch and bound, a state space tree is constructed in such a way that all children of an E-node are generated before any other live node becomes an E-node.

→ Generated nodes that cannot possibly lead to a feasible solution are discarded. The remaining nodes are added to the list of live nodes, and then one node from this list is selected to become the next E-node. This expansion process continues until either the answer node is found or the list of live nodes becomes empty.

→ The next E-node can be selected in one of three ways:

1. FIFO (or) Breadth First Search: This scheme extracts nodes from the list of live nodes in the same order as they are placed in it. The live nodes list behaves as a queue.
2. LIFO (or) D Search: The live nodes list behaves as a stack.
3. LC Search (Least Cost Search) (or) Best First Search: The nodes are assigned ranks based on certain criteria and they are extracted in the order of Best-Rank-First.

→ Brach and Bound involves two iterative steps:

1. Branching:
   Splitting the problem into a number of subproblems.
   (or)
   Generating all the children of an E-node in the state space tree.

2. Bounding:
   Finding an ***optimistic estimate*** of the best solution to the subproblem.

Optimistic estimate: Upper bound for maximization problems.
Lower bound for minimization problems.

→ In case of LC Search Brach and Bound (LCBB) method, after generating the children of an E-node, the node with the best bound value (i.e., smallest lower bound in case of minimization problems or largest upper bound in case of maximization problems) is chosen from the list of all live nodes and is made the next E-node.

→ We terminate the search process at the current node of an LCBB algorithm because of any one of the following reasons:

1. The node represents an infeasible solution because constraints are not satisfied.

2. The bound value of the node is not better than the value of the best solution seen so far.

0/1 Knapsack problem:

Given *n* items with profits *(p1, p2, ..., pn)* and weights *(w1, w2, ..., wn)* and knapsack capacity *M*.

maximize $\sum_{i=1}^{n} p_i x_i$
Subject to the constraints
$\sum_{i=1}^{n} w_i x_i \leq M$
and
$x_i \in \{0,1\}$, $1 \leq i \leq n$.

Solution to the 0/1 Knapsack problem using LCBB:

→ 0/1 knapsack problem is maximization problem.

How to find the optimistic estimate (i.e., upper bound)?

We relax the integral constraint, i.e., $x_i \in \{0,1\}$, $1 \leq i \leq n$. That means, fractions of the items are allowed.

→ Arrange the items in the decreasing order of profit densities (*pi/wi* values).
→ In the state space tree, at every node we record three values, viz.,
W: Sum of the weights of the objects considered till now
P: Sum of the profits of the objects considered till now
UB: Upper bound on the optimal profit

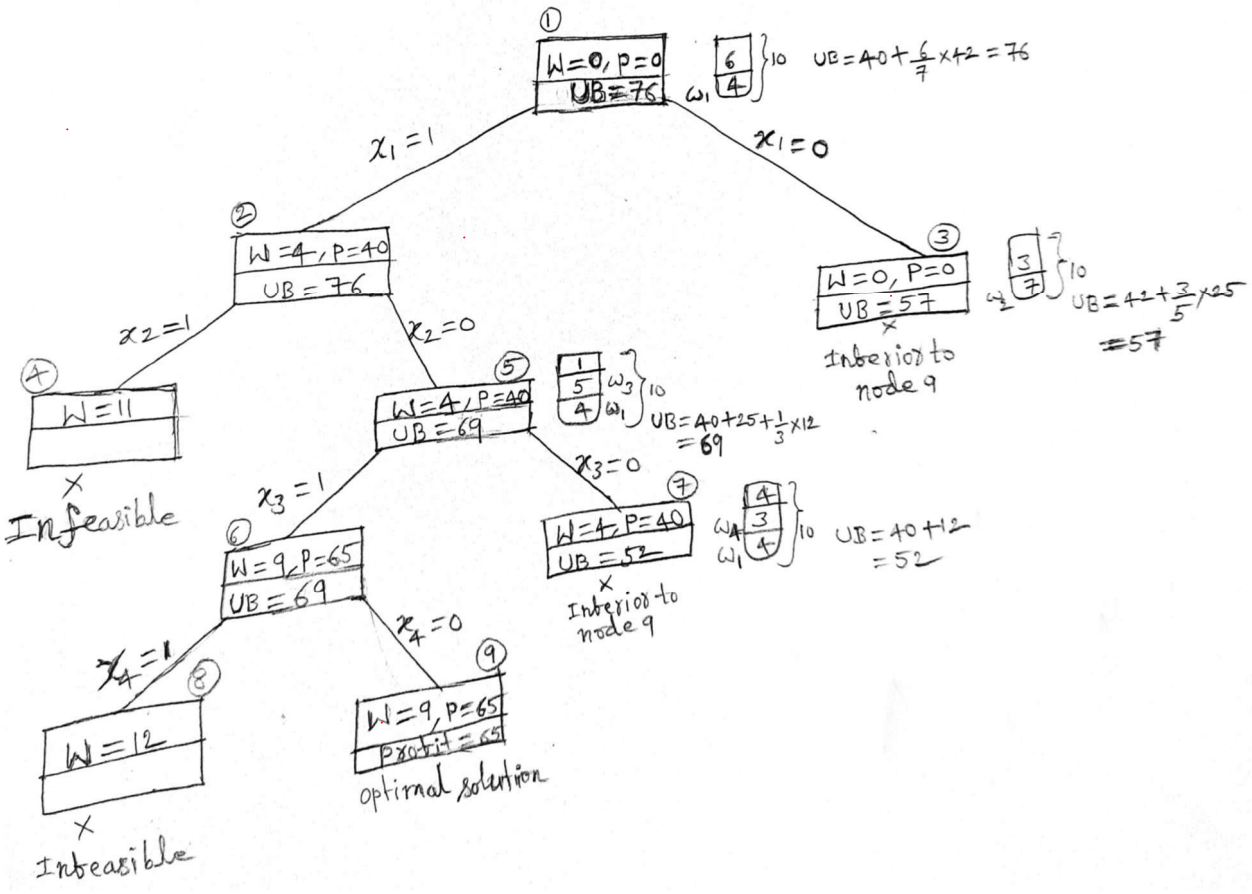
→ The upper bound is computed as follows:
Upper Bound = Sum of the profits of the items provided the total weight is less than or equal to knapsack capacity considering the fractions of items.

Example:

n=4; (w1, w2, w3, w4) = (4,7,5,3); (p1, p2, p3, p4) = (40,42,25,12) and *M=10.*

Solution:

i	*1*	*2*	*3*	*4*
Pi	*40*	*42*	*25*	*12*
wi	*4*	*7*	*5*	*3*
pi/wi	*10*	*6*	*5*	*4*



The optimal solution is: *(x1, x2, x3, x4) = (1,0,1,0)*

Exercise:

n=3 ; (w1, w2, w3) = (2,1,3) ; (p1, p2, p3,p4) = (10,4,6) and *M=5*.