EXPERIMENT 13

AIM:

Implement 0/1 knapsack by branch and bound.

BRANCH AND BOUND

1.Difference between backtracking and branch & bound

2.0/1 knapsack problem

3.Travelling salesman problem

4.15 puzzle

Branch and Bound

Here we have 2 terms

Branching: covering the feasible region by several smaller feasible sub-regions. (Splitting into sub-regions)

Bounding: is a fast way of finding upper and lower bounds

What are the differences between 'Backtracking' and 'Branch & Bound' Algorithm techniques?

Backtracking	Branch-and-Bound
[1] It is used to find all possible solutions available to the problem.	[1] It is used to solve optimization problem.
[2] It traverse tree by DFS (Depth First Search).	[2] It may traverse the tree in any manner, DFS or BFS.
[3] It realizes that it has made a bad choice & undoes the last choice by backing up.	[3] It realizes that it already has a better optimal solution that the pre-solution leads to so it abandons that pre-solution.
[4] It search the state space tree until it found a solution.	[4] It completely searches the state space tree to get optimal solution.
[5] It involves feasibility function	[5] It involves bounding function.

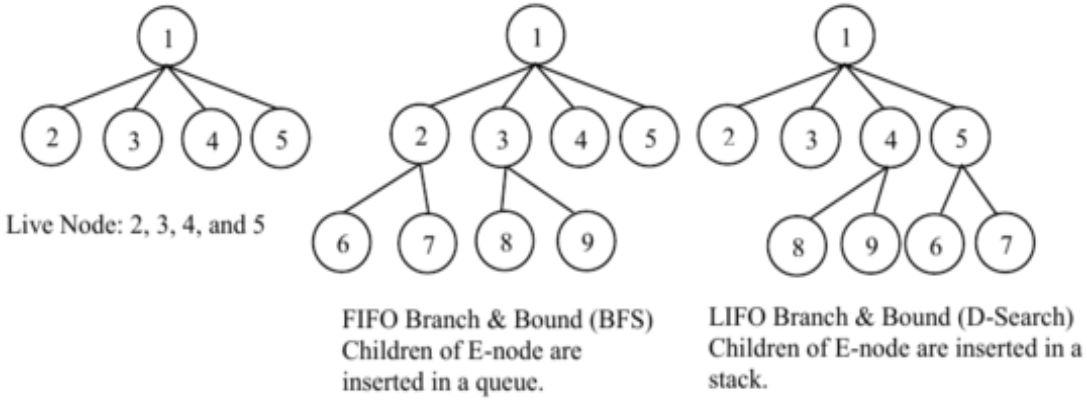
General method:

Branch and Bound is another method to systematically search a solution space. Just like backtracking, we will use bounding functions to avoid generating subtrees that do not contain an answer node. However branch and Bound differs from backtracking in two important manners:

1. It has a branching function, which can be a depth first search, breadth first search or based on bounding function.

2. It has a bounding function, which goes far beyond the feasibility test as a mean to prune efficiently the search tree.

- There are basically three types of nodes involved in Branch and Bound
- **1. Live node** is a node that has been generated but whose children have not yet been generated.
- **2. E-node** is a live node whose children are currently being explored. In other words, an E-node is a node currently being expanded.
- **3. Dead node** is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded.



Branch and Bound refers to all state space search methods in which all children of the E-node are generated before any other live node becomes the E-node

Branch and Bound is the generalisation of both graph search strategies, BFS and D-search.

- A BFS like state space search is called as FIFO (First in first out) search as the list of live nodes in a first in first out list (or queue).
- A D search like state space search is called as LIFO (Last in first out) search as the list of live nodes in a last in first out (or stack).

Least Cost (LC) search:

In both LIFO and FIFO Branch and Bound the selection rule for the next E-node in rigid and blind. The selection rule for the next E-node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly.

The search for an answer node can be speeded by using an "intelligent" ranking function $\hat{c}(\cdot)$ for live nodes. The next E-node is selected on the basis of this ranking function. The node x is assigned a rank using:

$$\hat{c}(x) = f(h(x)) + \hat{g}(x)$$

where, $\hat{c}(x)$ is the cost of x.

h(x) is the cost of reaching x from the root and f(.) is any non-decreasing function.

$\hat{g}(x)$ is an estimate of the additional effort needed to reach an answer node from x.

A search strategy that uses a cost function $\hat{c}(x) = f(h(x) + g(x)$ to select the next E-node would always choose for its next E-node a live node with least $\hat{c}(.)$ is called a LC–search (Least Cost search)

Control Abstraction for Branch and Bound(LC Method)

```
Algorithm LCSearch(t)
{       //Search t for an answer node
        if *t is an answer node then output *t and return;
        E := t;         //E-node.
        initialize the list of live nodes to be empty;
        repeat
        {
                for each child x of E do
                {
                        if x is an answer node then output the path from x to t and return;
                        Add (x);                        //x is a new live node.
                        (x → parent) := E;              // pointer for path to root
                }
                if there are no more live nodes then
                {
                        write ("No answer node");
                        return;
                }
                E := Least();
        } until (false);
}
```

LC Method Control Abstarction Explanation

- The search for an answer node can often be speeded by using an "intelligent" ranking function, c(.), for live nodes.
- The next £-node is selected on the basis of this ranking function.
- Let T be a state space tree and c( ) a cost function for the nodes in T. If X is a node in T then c(X) is the minimum cost of any answer node in the subtree with root X. Thus, c(T) is the cost of a minimum cost answer node

LC Method Control Abstarction Explanation

- The algorithm uses two subalgorithms LEAST(X) and ADD(X) to respectively delete and add a live node from or to the list of live nodes.
- LEAST{X) finds a live node with least c( ). This node is deleted from the list of live nodes and returned in variable X.
- ADD(X) adds the new live node X to the list of live nodes.
- Procedure LC outputs the path from the answer node it finds to the root node T.

0/1 knapsack problem using Branch and Bound

The 0/1 knapsack problem

- Positive integer $P_1, P_2, \ldots, P_n$ (profit)
  $W_1, W_2, \ldots, W_n$ (weight)
  M (capacity)

$$\text{maximize} \quad \sum_{i=1}^{n} P_i X_i$$

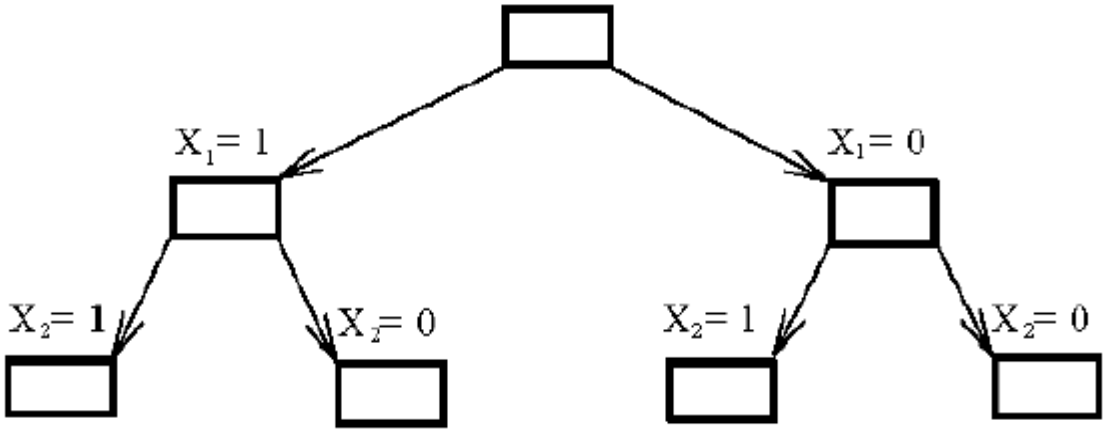
$$\text{subject to} \quad \sum_{i=1}^{n} W_i X_i \le M \quad X_i = 0 \text{ or } 1,\ i = 1, \ldots, n.$$

The problem is modified:

$$\text{minimize} \quad -\sum_{i=1}^{n} P_i X_i$$

Here why the problem is modified to minimum is that branch and bound can be applied only for minimization problem but not maximization problem. Since here 0/1 is a maximization problem, to convert that into minimization we do it by taking negative value of cost and bound.

The 0/1 knapsack problem



The numbering of the node is given using BFS.

The Branching Mechanism in the Branch-and-Bound Strategy

LC Branch and Bound procedure:

1. Draw a state space tree and set upper = ∞
2. Compute C^(x), u(x) for each node.
3. U(x)=-∑Pi

   C^(x)=u(x)-[m - current total weight] *[actual profit of remaining object]
   [actual weight of remaining object]
4. If u (x) is minimum than upper them upper will set to u(x).
5. If C^(x) > upper, kill node x.
6. Otherwise the min cost C^(x) becomes E-node and Generate children for E-node.
7. Repeat steps 2 to 6 until all the nodes get covered.
8. The minimum cost C^(x) becomes the answer node. Trace the path in backward direction from x to root for solution subset.

Algorithm for computing U(x)

```
algorithm ubound ( cp, cw, k, m )
{
// Input:  cp: Current profit total
// Input:  cw: Current weight total
// Input:  k:  Index of last removed item
// Input:  m:  Knapsack capacity

    profit = cp;
    weight = cw;
    for ( i = k + 1; i <= n; i++ )
        if ( weight + w[i]  <= m )   // w[i] is the weight of ith item
        {
            weight += w[i];
            profit -= p[i];          // p[i] is the profit due to ith item
        }

    return profit;
}
```

Algorithm for computing c^(x)

* $c(x) < \text{bound}(-q, \sum_{q \le i < j} w_i x_i, j-1)$

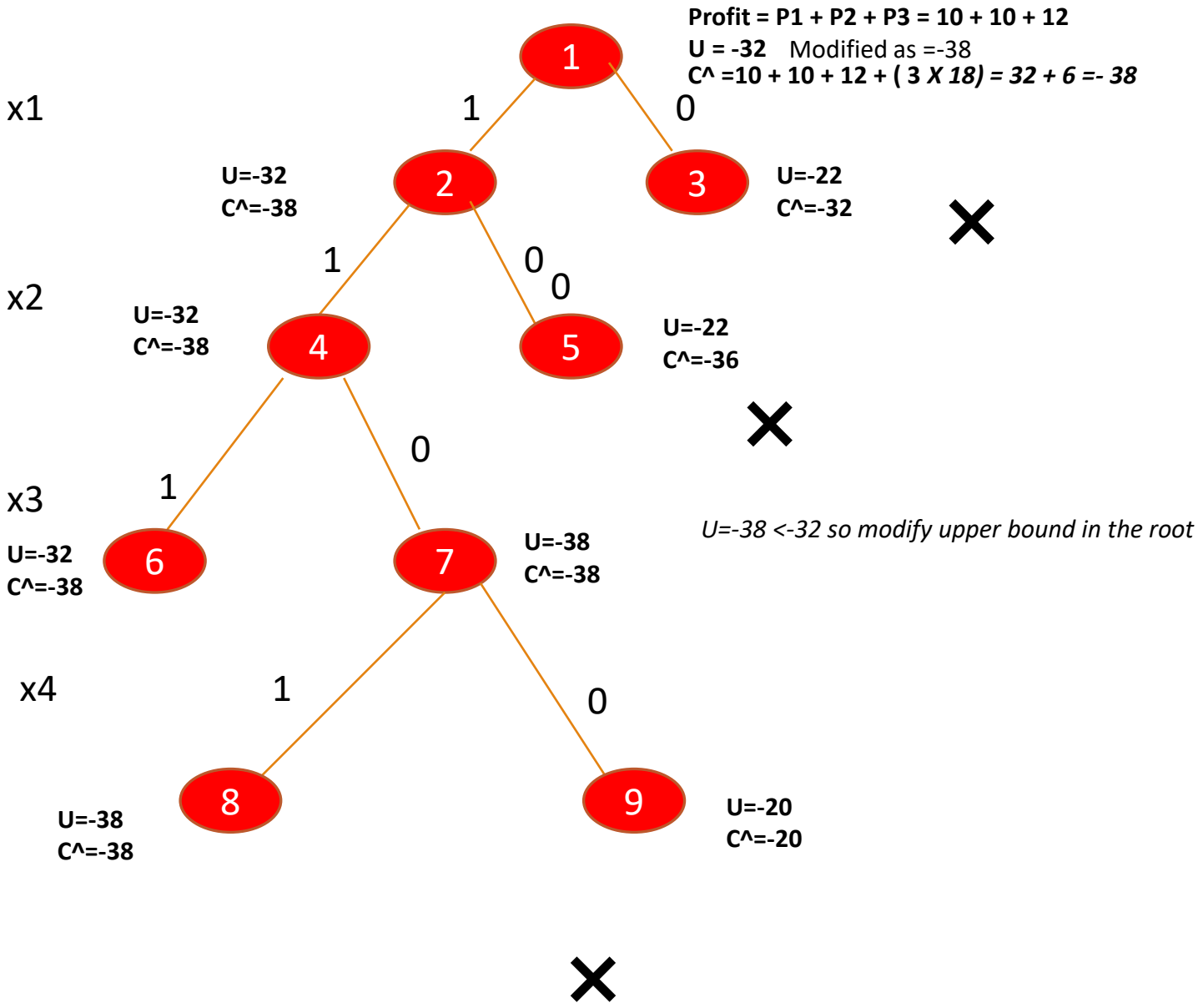
```
algorithm bound ( cp, cw, k )
{
// Input:  cp: Current profit total
// Input:  cw: Current weight total
// Input:  k:  Index of last removed item
// Input:  m:  Knapsack capacity

    profit = cp;
    weight = cw;
    for ( i = k + 1; i <= n; i++ )
    {
        weight += w[i];         // w[i] is the weight of ith item
        if ( weight < m )
            profit += p[i];     // p[i] is the profit due to ith item
        else
            return ( profit + ( 1 - ( weight - m ) / w[i] ) * p[i] );
    }

    return profit;
}
```

0/1 Knapsack Example using LCBB (Least Cost)

- Example (LCBB)
- Consider the knapsack instance:
- n = 4;
- (pi, p2,
- p3, p4) = (10, 10, 12, 18);
- (wi. w2, w3, w4) = (2, 4, 6, 9) and
- M = 15.


Profit = P1 + P2 + P3 = 10 + 10 + 12
U = -32 Modified as =-38
C^ =10 + 10 + 12 + (3 X 18) = 32 + 6 =- 38
x1
x2
x3
x4
1
2
3
4
5
6
7
8
9
U=-32
C^=-38
U=-22
C^=-32
U=-32
C^=-38
U=-22
C^=-36
U=-32
C^=-38
U=-38
C^=-38
U=-38
C^=-38
U=-20
C^=-20
U=-38 <-32 so modify upper bound in the root

Viva Questions

1. Draw the potion of state space tree generated by LC-knapsack for the following instances

M=12,

Profit	10	12	20	25
Weight	2	3	4	5

2. Distinguish LIFO & FIFO Branch and Bound
3. Define the term branch and bound technique with an example
4. Apply B&B technique on 15-puzzle game. Describe the solution
5. Distinguish Backtracking and Branch and Bound

Video Link References

1. https://www.youtube.com/watch?v=QwS20ytvThE