**ASSIGNMENT 4**

**Title:** Write a program to solve the travelling salesman problem and to print the path and the cost using LC Branch and Bound.

**Software Requirement:** Ubuntu, C++ Compiler

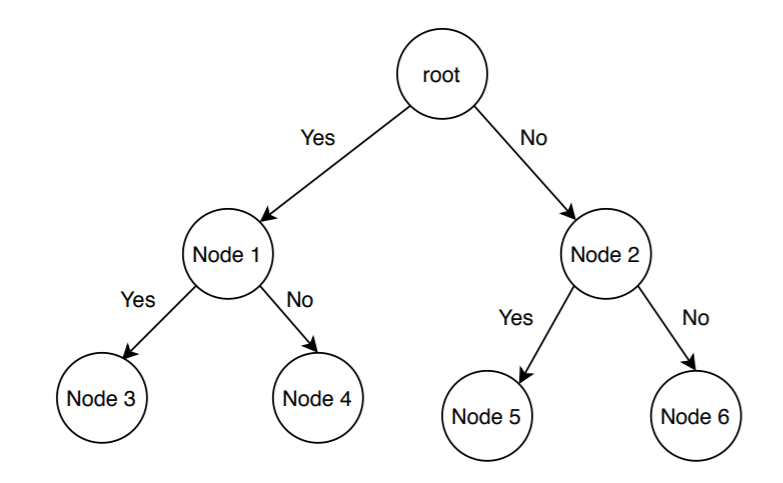
**Theory:**

Like backtracking Branch & Bound is used to solve constraint satisfaction problem and optimization problem. Both uses state space tree algorithms.

**1.Branch and Bound**

Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. The Branch and Bound Algorithm technique solves these problems relatively quickly.

The Backtracking Solution can be optimized if we know a bound on best possible solution subtree rooted with every node. If the best in subtree is worse than current best, we can simply ignore this node and its subtrees. So we compute bound (best solution) for every node and compare the bound with current best solution before exploring the node.



**2. General Procedure for B&B**

* Construct a state space tree such that E node remains an E-node until it becomes dead node.
* Maintain the list of live nodes using suitable data structure according to the search technique incorporated in the B&B algorithm.
  + For LC Search: a Min-Heap data structure
  + For FIFO search: a queue data structure
  + For LIFO search: a stack data structure
* Select the next E-Node from the list of live nodes by following on of the searching techniques
  + LC Search
  + FIFO Search
  + LIFO Search
* At each node, compute the values of lower and upper bounds on the possible solutions from that node.
* Apply the bounding functions to check whether a node produces a better solution than the best solution obtained thus far. Otherwise that node is killed without its further exploration.
* Enumerate another branch to get abetter solution.
* Repeat step 1 to 6 until an optimal solution to the problem is obtained.

**3.Least Cost Search**

* Least Cost(LC) search uses a heuristic cost function to assign the ranks to the live nodes.
* It estimates the extra computational work (cost) to reach an answer state from the current live node and accordingly assigns the rank to that live node.
* The cost at any node k is described by:
  + The minimum number of nodes required to be generated in a subtree of k to reach an answer node.
  + The number of levels the nearest answer state is distant from k.

**4.Travelling Salesman Problem**

The TSP ask to minimize the cost of a tour that visits all cities only once and ends at the starting city. Since it is minimization problem, it is directly solved v=by the B&B algorithm without changing the sign of an objective function.

**Problem Statement:** 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.

TSP has many practical applications. It is used in network design, and transportation route design. The objective is to minimize the distance. We can start tour from any random city and visit other cities in any order. With n cities, n! different permutations are possible. Exploring all paths using brute force attacks may not be useful in real life applications.

**5. LCBB using Static State Space Tree for Travelling Salesman Problem**

Branch and bound is an effective way to find better, if not best, solution in quick time by pruning some of the unnecessary branches of search tree. It works as follow: Consider directed weighted graph G = (V, E, W), where node represents cities and weighted directed edges represents direction and distance between two cities.

1. Initially, graph is represented by cost matrix C, where

Cij = cost of edge, if there is a direct path from city i to city j

Cij = ∞, if there is no direct path from city i to city j.

2. Convert cost matrix to reduced matrix by subtracting minimum values from appropriate rows and columns, such that each row and column contains at least one zero entry.

3. Find cost of reduced matrix. Cost is given by summation of subtracted amount from the cost matrix to convert it in to reduce matrix.

4. Prepare state space tree for the reduce matrix

5. Find least cost valued node A (i.e. E-node), by computing reduced cost node matrix with every remaining node.

6. If <i, j> edge is to be included, then do following :

(a) Set all values in row i and all values in column j of A to ∞

(b) Set A[j, 1] = ∞

(c) Reduce A again, except rows and columns having all ∞ entries.

7. Compute the cost of newly created reduced matrix as,

Cost = L + Cost(i, j) + r

Where, L is cost of original reduced cost matrix and r is A[i, j].

8. If all nodes are not visited then go to step 4.

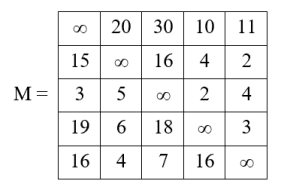
Reduction procedure is described below:

Row Reduction:

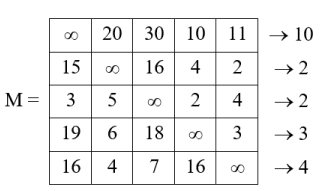
Matrix M is called reduced matrix if each of its row and column has at least one zero entry or entire row or entire column has ∞ value. Let M represents the distance matrix of 5 cities. M can be reduced as follow:

MRowRed  = {Mij – min {Mij | 1 ≤ j ≤ n, and Mij < ∞ }}

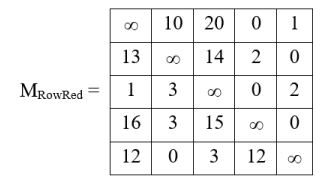
Consider the following distance matrix:



* Find the minimum element from each row and subtract it from each cell of matrix.



* Reduced matrix would be:



* Row reduction cost is the summation of all the values subtracted from each rows:

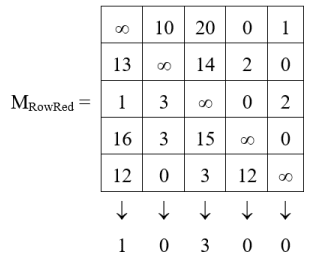
Row reduction cost (M) = 10 + 2 + 2 + 3 + 4 = 21

Column reduction:

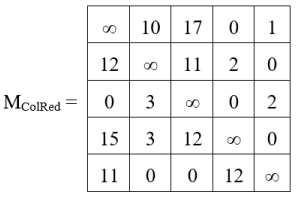
Matrix MRowRed is row reduced but not the column reduced. Matrix is called column reduced if each of its column has at least one zero entry or all ∞ entries.

MColRed = {Mji – min {Mji | 1 ≤ j ≤ n, and Mji < ∞ }}

* To reduced above matrix, we will find the minimum element from each column and subtract it from each cell of matrix.



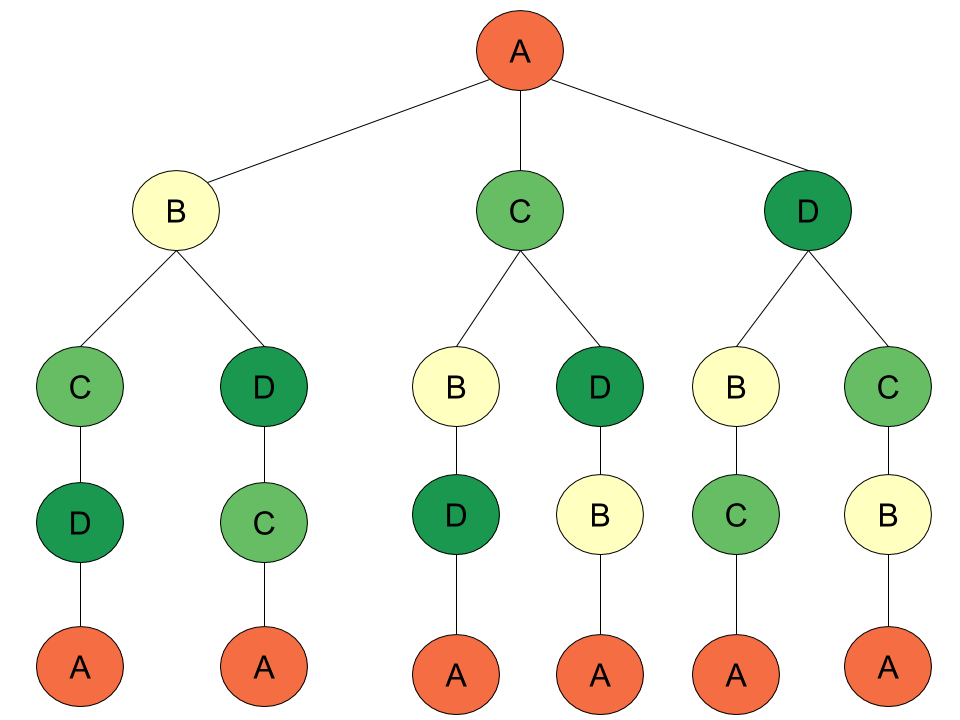
* Column reduced matrix MColRed would be:



* Each row and column of MColRed has at least one zero entry, so this matrix is reduced matrix.

Column reduction cost (M) = 1 + 0 + 3 + 0 + 0 = 4

State space tree for 4 city problem is depicted in following figure.



**9. Complexity Analysis**

* Creating all the possible extensions of E-nodes in terms of tree nodes. Which is nothing but a permutation. Suppose we have N cities, then we need to generate all the permutations of the (N-1) cities, excluding the root city.
* Hence the time complexity for generating the permutation is O((n-1)!), which is equal to O(2 (n-1)).
* Hence the final time complexity of the algorithm can be O(n2 \* 2n).

**Conclusion:**

Travelling salesman problem using LC Branch and Bound is implemented successfully.