ASSIGNMENT - 13

TITLE: TSP USING BRANCH AND BOUND

PROBLEM STATEMENT: To understand & implement least cost branch and bound algorithm for solving TSP & study BB strategy.

THEORY:

- TRAVELLING SALESMAN PROBLEM:
Let G(V,E) be a directed graph defining an instance of TSP. Let c_{ij} be the edge $<i,j>$, $|V| = n$. We may assume tour starts & ends at 1. So solution space S = {1, M, 1}. This permutation {(2, 3, ... n)} S maybe organised into state space tree.

- LEAST COST BRANCH AND BOUND:
In order to use LCBB to search TSP tree, we need to define a cost function C( ) & 2 other functions ε( ) & U( ) such that $\varepsilon(R) < C(R) < U(R)$ is such that solution node is with least C(·) corresponds to G. With every node in TSP, state space tree, we may associate a reduced cost matrix. A be reduced matrix for R node. If S is not a leaf, then reduced cost matrix for S maybe obtained as follows:

i. Change all entries in row i & column j of A to ∞. This prevents use of any more edges leaving vertex i or entering vertex j.
ii. Set A (j, 1) to ∞. This prevents the use of edge <j, i>
iii. Reduce all rows & columns in the resulting matrix except for rows & columns containing only ∞.

23 · 2

- ALGORITHM:

1. Read no. of cities n & read tsp_cost matrix
2. Initialise red_matrix to tsp_cost matrix
3. cost = reduce_matrix (tsp_cost_matrix)
   // perform row & col. reduction & find cost matrix
4. node. cost [] cost obtained in (3)
5. node. path [0] = 1 // no. of cities traversed = 1
   node. path [1] = 1 // start from city 1
   node. matrix = reduced matrix obtained in (3)
5. node = expand (node) // perform expansion of
   // live node & get first solution.
6. if node. cost < list [1] . cost , go to (5)
7. else
8. while (1) do
9.   if size of heap is 0 , break
10.   node = delete () // delete node from heap
11.   node = expand (node) // again start expansion
12.   if node. cost < list [1]. cost , go to (4)
13. end do
14. print path using node. path
15. print cost at that node.
16. You can verify cost using original cost matrix as well
17. Stop

function Expansion (node)
// Input: root's info
// Output: last node with path storing all cities
// cost
1. while (1)
2. do

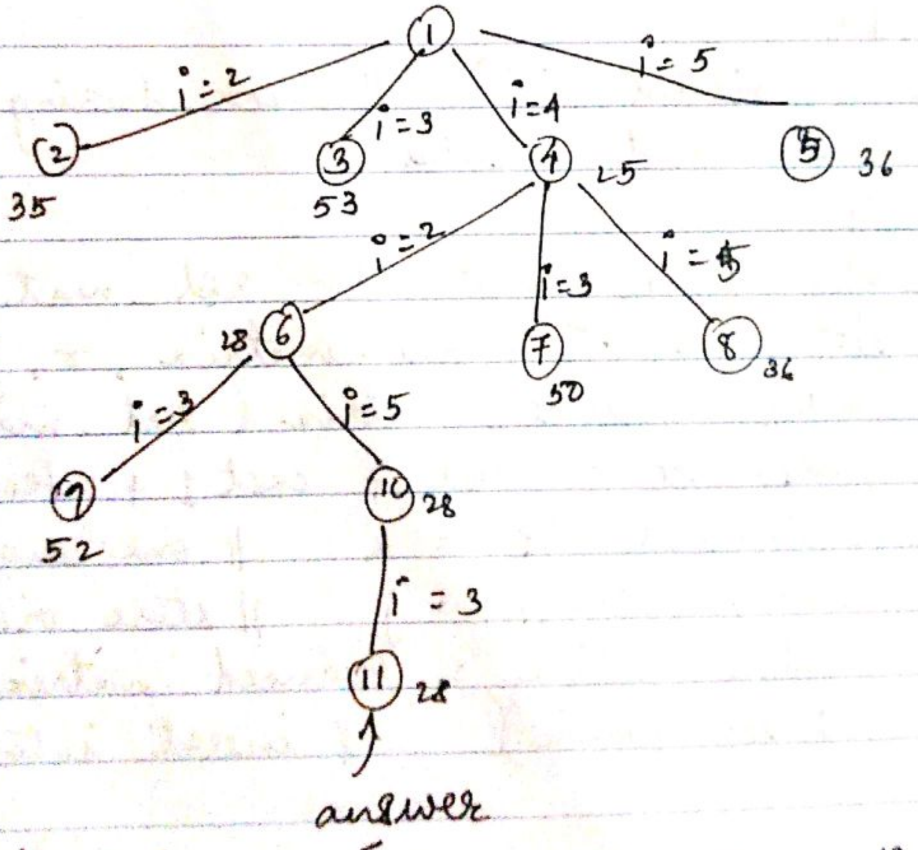
3. count = node. path [0] // count for no. of cities
// traversed
4. K = count + 1 // index to store next city to be
// traversed.
5. cost = node. cost
6. store node matrix to some temporary matrix,
say temp_mat to be used for expansion.
7. r = node. path [count] // last city in path
8. for i = 1 to n, set visited [i] = 0
9. for j = 1 to count, set visited [path [i]] = 1
10. for j = 0 to n
11. begin for
12. if (! visited [j]) // check during expansion
// live node not visited
13. begin if
14. copy temp_mat to red_mat
15. set_infinity (red-matrix, r, j)
16. cost 1 = reduce_matrix (red_mat) // for red^n
17. node. cost = cost + cost 1 + temp_mat [r] [j]
18. node. path [0] = k // one more city visited
19. node. path [k] = j // store visited city
20. node. matrix = reduced matrix obtained in (
21. insert (node) // insert into heap
22. end if
23. end for
24. if K = n, then break // all visited, first feasible
// solution found.
25. node = delete () // delete from heap, this node
// has min. cost so it becomes current node for expansion
26. end while
27. return node

Example :

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \rightarrow \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

cost matrix — reduced cost matrix

State Space Tree generated by LCBB :



for node 2 :
path (1, 2)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

for node 3 :
path (1,3)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

for node 4 :
path (1, 4)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

for node 5 :
path (1,5)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

for node 6 :
path (1,6)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

for node 7 :
path(1,4,3)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

for node 8 :
path (1,4,5)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \end{bmatrix}$$

for node 9 :
path (1,4,2,3)

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

for node 10 :
path

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

Input : cost matrix TSP graph
Output : reduced matrix obtained by LCBB

- CONCLUSION:
Thus we have studied & implemented LCBB for T