Shreya Thigale
TE-IX
33162.

PAGE No.
DATE / /

ASSIGNMENT-13

TITLE: Travelling Salesman Problem (BB)

PROBLEM STATEMENT:
Write a program to solve the travelling salesman problem and to print the path and cost using Branch and Bound.

OBJECTIVE:
To understand and implement Least Cost Branch and Bound algorithm for solving Travelling Salesman problem and study Branch and Bound strategy.

THEORY:

Branch and Bound Strategy

Branch and bound (BB or B&B) is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as general real valued problems. A branch-and bound algorithm consists of systemic enumeration of candidate solutions by mean of state space search: the set of candidate solutions is thought of as forming a rooted tree with full set at root.

PAGE No.
DATE / /

Least Cost Branch and Bound

In ordu to use LC branch bound to search the travelling salesperson state space tree, we need to define a cost function $c()$ and two other functions $\hat{c}()$ and $u()$ such that $\hat{c}(R) \leq c(R) \leq u(R)$ for all nodes R.

Example:

∞	20	30	10	11
15	∞	16	4	2
3	5	∞	2	4
19	6	18	∞	3
16	4	7	16	∞

Cost matrix

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	0	12	∞

Reduced cost matrix $L=25$

∞	∞	∞	∞	∞
∞	∞	11	2	0
0	∞	∞	0	2
15	∞	12	∞	0
11	∞	0	12	∞

a) path 1,2; node 2

∞	∞	∞	∞	∞
1	∞	∞	2	0
∞	3	∞	0	2
4	3	∞	∞	0
0	0	∞	12	∞

b) path 1,3; node 3

∞	∞	∞	∞	∞
12	∞	11	∞	0
0	3	∞	∞	2
∞	3	12	∞	0
11	0	0	∞	∞

c) path 1,4; node 4

∞	∞	∞	∞	∞
10	∞	9	0	∞
0	3	∞	0	∞
12	0	9	∞	∞
∞	0	0	12	∞

d) path 1,5; node 5

∞	∞	∞	∞	∞
∞	∞	11	∞	0
0	∞	∞	∞	2
∞	∞	∞	∞	∞
11	∞	0	∞	∞

e) path 1,4,2; node 6

∞	∞	∞	∞	∞
1	∞	∞	∞	0
∞	1	∞	∞	0
∞	∞	∞	∞	∞
0	0	∞	∞	∞

f) path 1,4,3; node 7

∞	∞	∞	∞	∞
1	∞	0	∞	∞
0	3	∞	∞	∞
∞	∞	∞	∞	∞
∞	0	0	∞	∞

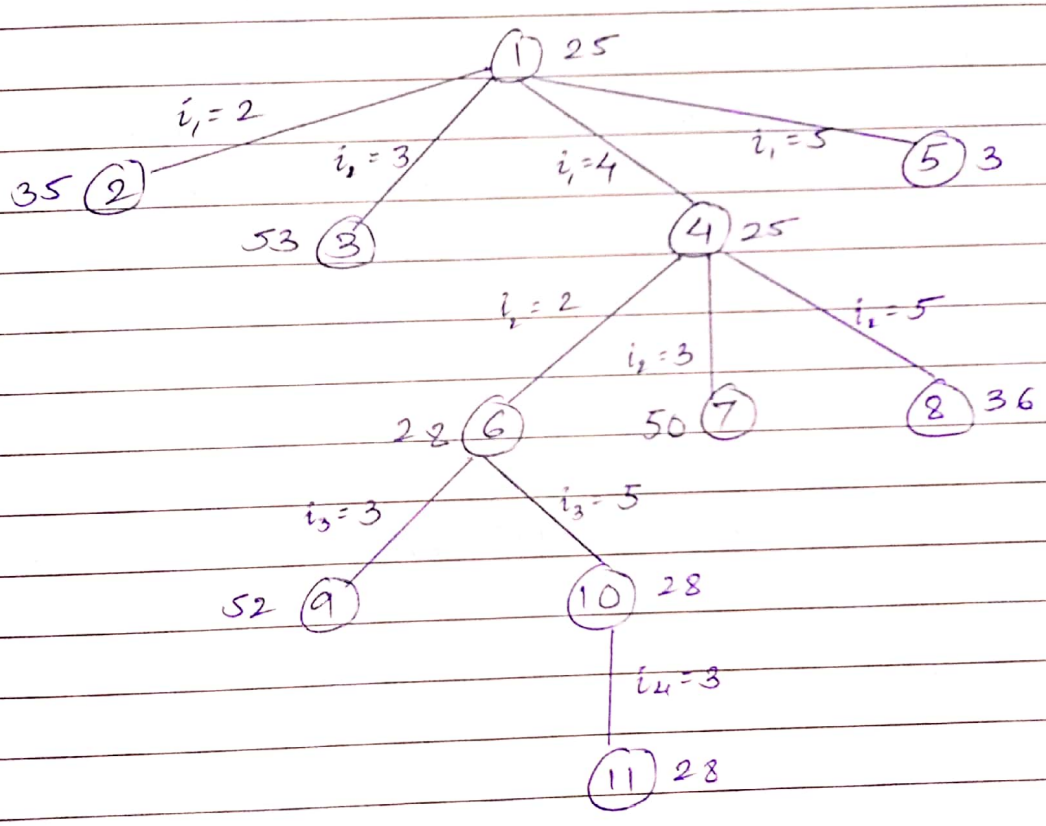
g) path 1,4,5; node 8

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	0
∞	∞	∞	∞	∞
0	∞	∞	∞	∞

h) path 1,4,2,3; node 9

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
0	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	0	∞	∞

i) path 1,4,2,5; node 10



State space tree generated by procedure LCBB.

ALGORITHM FOR TSP using LCBB method:

Data structures:

```
struct Item {
int cost;
int path[10];
```

```
    int *matrix[10][10];
};
typedef struct item hterm;

hterm node, list[25];
int visited[10];
int red_mat[10][10], temp[10][10]
```

1. Read the no. of cities n and read the tsp-cost matrix
2. Initialize red-matrix to tsp-cost matrix
3. cost = reduce-matrix (tsp-cost-matrix)
4. node.cost ← cost
   node.path[0] = 1;
   node.path[1] = 1;
   node.matrix = reduced matrix
5. node = expand (node);
6. if node.cost < list[1].cost
   goto step 14
7. else
8. while (1) do
9. if size of heap is 0 break;
10. node = delete();
11. node = expand (node);
12. if (node.cost < list[1].cost)
13. end do
14. print path using node.path
15. print cost at node
16. you can verify that cost using original cost matrix as well
17. stop.

PAGE No.
DATE / /

Function expansion (node)

1. while (1) do
2. do
3. count = node.path[0];
4. k = count + 1;
5. cost = node.cost;
6. Store node.matrix to some temporary matrix say temp-matrix
7. r = node.path[count]
8. for i=1 to n set visited(i) = 0;
9. for i=1 to count set visited[path[i]] = 1;
10. for j=2 to n
11. Begin for
12. if (!visited[j])
13. Begin if
14. copy the temp-matrix to red matrix
15. set infinity (red-matrix, r, j)
16. cost1 = reduce-matrix(red-matrix);
17. node.cost = cost + cost1 + temp-matrix[r][j];
18. node.path[0] = k;
19. node.path[k] = j;
20. node.matrix = reduced.matrix optained in Step 16
21. Insert(node)
22. end if
23. end for
24. if (k != n) break;
25. node = Bracket()
26. End while
27. Return node.

DATE / /

Analysis:

While worst case TSP complexity will not be any better than $O(n^2 2^n)$, the use of good bounding functions will enable these branch and bound algorithms to solve some problem instances in much less time than required by dynamic programming algorithm.

CONCLUSION:

Thus we have studied Least Cost Branch and Bound strategy for TSP and also implemented it successfully.