**TASK 13**

**Branch & Bound - Travelling Salesman problem**

### In a bustling city, a courier named Maya takes on a challenging assignment to deliver packages to n different locations efficiently. The goal is to find the shortest route that allows her to visit each delivery point exactly once and return to the starting point.

### Input

First line contains an integer N = number of cities

N lines follow

Each line contains label x y

where:

label = label of the city (String)

x = x co-ordinate of the city (Real Valued)

y = y co-ordiate of the city (Real Valued)

### Output

Output an optimal path in space separated manner.

### Sample Input#1

4

0 0 0

1 0 1

2 1 0

3 1 1

### Sample Output#1

0 1 2 3

**Test Case 1:** Derive the solution of TSP using B&B and prove that it is optimal when compared with Brute Force technique.

**Test Case 2:** Analyze that the Travelling Salesman Problem (TSP) is an NP hard problem.

**Aim:**

Create to C program to implement Traveling Salesman Problem using Branch and Bound algorithm.

**Algorithm**:

**Step1.** Input the first node as root. Identify the root, if there is still a node to expand then do the

branching if there is no more node (empty) then the search is done.

**Step2.** From the branch, choose the node (*i*) with the minimum cost. If exist several nodes with the same cost, then choose arbitrarily.

**Step3.** Identify the node (*i*),

**Step 3.1** If the (*i*) is the solution –no more queue node– then stop the search,

**Step 3.2** If the (*i*) is not the solution, then do the branching and choose the node (*i*) with the minimum cost. If exist several nodes with the same cost, then choose arbitrarily.

**Step 4.** Back to step 3.

**Program:**

#include <stdio.h>

int a[10][10], visited[10], n, cost = 0;

void get();

void mincost(int city);

int least(int c); // Function prototype added

void put();

void get() {

int i, j;

printf("Enter No. of Cities: ");

scanf("%d", &n);

printf("\nEnter Cost Matrix: \n");

for (i = 0; i < n; i++) {

printf("\n Enter Elements of Row# : %d\n", i + 1);

for (j = 0; j < n; j++)

scanf("%d", &a[i][j]);

visited[i] = 0;

}

printf("\n\nThe cost list is:\n\n");

for (i = 0; i < n; i++) {

printf("\n\n");

for (j = 0; j < n; j++)

printf("\t %d", a[i][j]);

}

}

void mincost(int city) {

int i, ncity;

visited[city] = 1;

printf("%d -> ", city + 1);

ncity = least(city);

if (ncity == 999) {

ncity = 0;

printf("%d", ncity + 1);

cost += a[city][ncity];

return;

}

mincost(ncity);

}

int least(int c) {

int i, nc = 999;

int min = 999, kmin;

for (i = 0; i < n; i++) {

if ((a[c][i] != 0) && (visited[i] == 0)) {

if (a[c][i] < min) {

min = a[i][0] + a[c][i];

kmin = a[c][i];

nc = i;

}

}

}

if (min != 999)

cost += kmin;

return nc;

}

void put() {

printf("\n\nMinimum cost: %d\n", cost);

}

void main() {

get();

printf("\n\nThe Path is:\n\n");

mincost(0);

put();

}

**Output:**

Enter No. of Cities: 6

Enter Cost Matrix:

Enter Elements of Row# : 1

99 10 15 20 99 8

Enter Elements of Row# : 2

5 99 9 10 8 99

Enter Elements of Row# : 3

6 13 99 12 99 5

Enter Elements of Row# : 4

8 8 9 99 6 99

Enter Elements of Row# : 5

99 10 99 6 99 99

Enter Elements of Row# : 6

10 99 5 99 99 99

The Path is:

1 –>6 –>3 –>4 –>5 –>2 –>1

Minimum cost: 46

**Test Case 1:** Derive the solution of TSP using B&B and prove that it is optimal when compared with Brute Force technique.

**Branch and Bound (B&B):**

B&B is a technique used to solve combinatorial optimization problems like the Traveling Salesman Problem (TSP).

It involves exploring the search space using a systematic approach and pruning branches that are not promising, hence reducing the search space.

**Brute Force Technique:**

Brute force involves trying all possible permutations of cities and calculating the total distance for each permutation.

It guarantees finding the optimal solution but becomes impractical for large problem instances due to its exponential time complexity.

**Proving Optimality with B&B:**

B&B also guarantees finding the optimal solution but is more efficient than brute force due to pruning.

Conducting experiments to measure the execution time of both algorithms for different problem sizes, demonstrating that B&B is significantly faster.

**Test Case 2: Analyzing TSP as an NP-Hard Problem:**

**Definition of NP-Hard:**

A problem is NP-hard if it is at least as hard as the hardest problems in NP (nondeterministic polynomial time) under polynomial-time reductions.

NP-hard problems do not have known efficient solutions, but if a polynomial-time solution exists for any NP-hard problem, then all problems in NP can be solved in polynomial time.

**Characteristics of TSP:**

TSP is a classic example of an NP-hard problem.

It involves finding the shortest tour that visits each city exactly once and returns to the starting city.

TSP's decision version (where the goal is to decide whether there exists a tour shorter than a given length) is NP-complete, meaning that TSP is at least as hard as any problem in NP.

**Proving NP-Hardness:**

TSP's NP-hardness can be demonstrated by reducing a known NP-complete problem to TSP.

For example, the Hamiltonian cycle problem, which seeks a cycle that visits every vertex exactly once, is NP-complete.

By reducing Hamiltonian cycle to TSP, we can show that solving TSP efficiently implies solving Hamiltonian cycle efficiently, thereby proving TSP's NP-hardness.

By analyzing the problem characteristics and demonstrating reductions from known NP-complete problems, we can establish TSP as an NP-hard problem.

**Result:**

Thus the Traveling Salesman Problem using Branch and Bound algorithm was executed successfully.