**Implementation of Branch and Bound technique**

**to solve Travelling salesman problem**

**Aim:**

To implement the Traveling Salesman Problem (TSP) using branch and bound.

**Problem Description:**

The Traveling Salesman Problem involves finding the minimum cost incurred while visiting all cities exactly once, starting and ending at the same city.

**Algorithm:**

* Start with an initial city as the current city and an empty route.Generate all possible extensions of the current route by adding one city at a time to it. Each extension represents a potential partial tour.
* Assign a lower bound to each partial tour. This lower bound is obtained by calculating the length of the partial tour so far and estimating the length of the remaining tour using a heuristic or lower bound technique (e.g., the length of a minimum spanning tree).
* Prune branches that have a higher cost than the best solution found so far. If the lower bound of a partial tour exceeds the cost of the best solution, discard it.
* Explore the remaining branches recursively by selecting the one with the lowest lower bound and repeating steps 2 to 4.
* When all branches have been explored, the algorithm terminates, and the best solution found during the process is the optimal solution to the TSP.

**Code:**

import math

maxsize = float('inf')

def copyToFinal(curr\_path):

    final\_path[:N + 1] = curr\_path[:]

    final\_path[N] = curr\_path[0]

def firstMin(adj, i):

    min = maxsize

    for k in range(N):

        if adj[i][k] < min and i != k:

            min = adj[i][k]

    return min

def secondMin(adj, i):

    first, second = maxsize, maxsize

    for j in range(N):

        if i == j:

            continue

        if adj[i][j] <= first:

            second = first

            first = adj[i][j]

        elif(adj[i][j] <= second and

            adj[i][j] != first):

            second = adj[i][j]

    return second

def TSPRec(adj, curr\_bound, curr\_weight,

            level, curr\_path, visited):

    global final\_res

    if level == N:

        if adj[curr\_path[level - 1]][curr\_path[0]] != 0:

            curr\_res = curr\_weight + adj[curr\_path[level - 1]]\

                                        [curr\_path[0]]

            if curr\_res < final\_res:

                copyToFinal(curr\_path)

                final\_res = curr\_res

        return

    for i in range(N):

        if (adj[curr\_path[level-1]][i] != 0 and

                            visited[i] == False):

            temp = curr\_bound

            curr\_weight += adj[curr\_path[level - 1]][i]

            if level == 1:

                curr\_bound -= ((firstMin(adj, curr\_path[level - 1]) +

                                firstMin(adj, i)) / 2)

            else:

                curr\_bound -= ((secondMin(adj, curr\_path[level - 1]) +

                                firstMin(adj, i)) / 2)

            if curr\_bound + curr\_weight < final\_res:

                curr\_path[level] = i

                visited[i] = True

                TSPRec(adj, curr\_bound, curr\_weight,

                    level + 1, curr\_path, visited)

            curr\_weight -= adj[curr\_path[level - 1]][i]

            curr\_bound = temp

            visited = [False] \* len(visited)

            for j in range(level):

                if curr\_path[j] != -1:

                    visited[curr\_path[j]] = True

def TSP(adj):

    curr\_bound = 0

    curr\_path = [-1] \* (N + 1)

    visited = [False] \* N

    for i in range(N):

        curr\_bound += (firstMin(adj, i) +

                    secondMin(adj, i))

    curr\_bound = math.ceil(curr\_bound / 2)

    visited[0] = True

    curr\_path[0] = 0

    TSPRec(adj, curr\_bound, 0, 1, curr\_path, visited)

N = 5

adj =[[0,3,1,5,8],

      [3,0,6,7,9],

      [1,6,0,4,2],

      [5,7,4,0,3],

      [8,9,2,3,0]]

# final\_path[] stores the final solution

# i.e. the // path of the salesman.

final\_path = [None] \* (N + 1)

# visited[] keeps track of the already

# visited nodes in a particular path

visited = [False] \* N

# Stores the final minimum weight

# of shortest tour.

final\_res = maxsize

TSP(adj)

print("Minimum cost :", final\_res)

print("Path Taken : ", end = ' ')

for i in range(N + 1):

    print(final\_path[i], end = ' ')

**Output:**

****

**Time Complexity:**

* Factorial time complexity, typically represented as O(n!), where n is the number of cities. The algorithm explores all possible permutations of cities to find the optimal tour.

**Algorithm Analysis:**

* The Branch and Bound technique systematically explores the solution space by branching on unvisited cities and using a lower bound function to estimate minimum tour costs. It efficiently prunes unpromising subproblems, making the search more focused and efficient.

**Result:**

Thus travelling salesman problem has been solved using branch and bound successfully.