Subject : DAA
Problem Statement : Write a program to solve the travelling salesman problem and to print the path and the cost using LC Branch and Bound.

```
#include<bits/stdc++.h>
using namespace std;

// defining varibales
#define N 4
#define INF INT_MAX

// structure for Node
struct Node
{

    vector<pair<int, int> > path;
    int matrix_reduced[N][N];
    int cost;
        int vertex;
    int level;
};

// fucntion for creating a node
Node* newNode(int matrix_parent[N][N], vector<pair<int, int> > const &path, int level, int i, int j)
{
    Node* node = new Node;
    node->path = path;

    if (level != 0)
    {

        node->path.push_back(make_pair(i, j));
    }

    memcpy(node->matrix_reduced, matrix_parent,
        sizeof node->matrix_reduced);

    for (int k = 0; level != 0 && k < N; k++)
    {
        node->matrix_reduced[i][k] = INF;
        node->matrix_reduced[k][j] = INF;
    }

    node->matrix_reduced[j][0] = INF;
    node->level = level;
    node->vertex = j;

    return node;
}

// function to reduce the matrix row wise
```

```
void rowReduction(int matrix_reduced[N][N], int row[N])
{

    fill_n(row, N, INF);

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (matrix_reduced[i][j] < row[i]) {
                row[i] = matrix_reduced[i][j];
            }
        }
    }

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (matrix_reduced[i][j] != INF && row[i] != INF) {
                matrix_reduced[i][j] -= row[i];
            }
        }
    }

}

// function to reduce the matrix column wise
void columnReduction(int matrix_reduced[N][N], int col[N])
{
    fill_n(col, N, INF);

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (matrix_reduced[i][j] < col[j]) {
                col[j] = matrix_reduced[i][j];
            }
        }
    }

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (matrix_reduced[i][j] != INF && col[j] != INF) {
                matrix_reduced[i][j] -= col[j];
            }
        }
    }
}
```

```
// function to print the matrix
void printMatrix(int matrix_reduced[N][N]){
    for(int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            if (matrix_reduced[i][j] == INF){
                cout<<"INF"<<" ";
            }else{
                cout<<matrix_reduced[i][j]<<" ";
            }
        }
        cout<<"\n";
    }

}

// function to calculate cost of matrix after row and column reduction
int costCal(int matrix_reduced[N][N])
{
    int cost = 0;

    int row[N];
    rowReduction(matrix_reduced, row);

    int col[N];
    columnReduction(matrix_reduced, col);

    for (int i = 0; i < N; i++)
    {
        cost += (row[i] != INT_MAX) ? row[i] : 0,
            cost += (col[i] != INT_MAX) ? col[i] : 0;
    }

    cout<<"\nReduced Matrix: \n\n";
    printMatrix(matrix_reduced);

    return cost;
}

// function to print the final path taken
void pathTaken(vector<pair<int, int> > const &list)
{
    cout << "\n\nPath taken: \n\n";
    for (int i = 0; (unsigned) i < list.size(); i++) {
        cout << list[i].first + 1 <<"->"<< list[i].second + 1 << endl;
    }
}

// structure
struct comp
{
    bool operator()(const Node* lhs, const Node* rhs) const {
```

```
        return lhs->cost > rhs->cost;
    }
};

// function to implement traveling salesman problem
int TSP(int costMatrix[N][N])
{
    priority_queue<Node*, vector<Node*>, comp> pq;

    vector<pair<int, int> > v;

    Node* root = newNode(costMatrix, v, 0, -1, 0);

    root->cost = costCal(root->matrix_reduced);

    pq.push(root);

    while (!pq.empty())
    {
        Node* min = pq.top();
        pq.pop();

        int i = min->vertex;

        if (min->level == N - 1)
        {
            min->path.push_back(make_pair(i, 0));
            pathTaken(min->path);
            return min->cost;
        }

        for (int j = 0; j < N; j++)
        {
            if (min->matrix_reduced[i][j] != INF)
            {
                Node* child = newNode(min->matrix_reduced, min->path, min->level + 1, i, j);
                child->cost = min->cost + min->matrix_reduced[i][j] + costCal(child->matrix_reduced);
                pq.push(child);
            }
        }
        delete min;
    }
    return 0;
}

// main funciton
int main()
{
    int costMatrix[N][N], result;
    // taking user input for cost matrix
    cout <<"Enter the cost matrix  :: \n";
    for (int i = 0; i < N; i++){
```

```
      for (int j = 0; j < N; j++){
        if ( i == j){
          costMatrix[i][j] = INF;
        }
        else{
          cout << "Enter the cost of edge "<<i+1<<" -> "<<j+1<<" : ";
          cin>>costMatrix[i][j];
        }
      }
    }
    result = TSP(costMatrix);
    cout << "\n\nTotal Cost :: "<< result <<"\n\n";

    return 0;
}

/*
OUTPUT

Enter the cost matrix  ::
Enter the cost of edge 1 -> 2 : 10
Enter the cost of edge 1 -> 3 : 15
Enter the cost of edge 1 -> 4 : 20
Enter the cost of edge 2 -> 1 : 10
Enter the cost of edge 2 -> 3 : 35
Enter the cost of edge 2 -> 4 : 25
Enter the cost of edge 3 -> 1 : 15
Enter the cost of edge 3 -> 2 : 35
Enter the cost of edge 3 -> 4 : 30
Enter the cost of edge 4 -> 1 : 20
Enter the cost of edge 4 -> 2 : 25
Enter the cost of edge 4 -> 3 : 30

Reduced Matrix:

INF 0 0 0
0 INF 20 5
0 20 INF 5
0 5 5 INF

Reduced Matrix:

INF INF INF INF
INF INF 10 0
0 INF INF 5
0 INF 0 INF

Reduced Matrix:

INF INF INF INF
0 INF INF 5
INF 10 INF 0
```

0 0 INF INF

Reduced Matrix:

INF INF INF INF
0 INF 20 INF
0 20 INF INF
INF 0 0 INF

Reduced Matrix:

INF INF INF INF
INF INF 0 INF
0 INF INF INF
INF INF INF INF

Reduced Matrix:

INF INF INF INF
0 INF INF INF
INF 0 INF INF
INF INF INF INF

Reduced Matrix:

INF INF INF INF
INF INF INF 0
INF INF INF INF
0 INF INF INF

Reduced Matrix:

INF INF INF INF
0 INF INF INF
INF INF INF INF
INF 0 INF INF

Reduced Matrix:

INF INF INF INF
INF INF INF INF
INF INF INF 0
0 INF INF INF

Reduced Matrix:

INF INF INF INF
INF INF INF INF
0 INF INF INF
INF INF 0 INF

Reduced Matrix:

INF INF INF INF
INF INF INF INF
INF INF INF INF
INF INF INF INF

Reduced Matrix:

INF INF INF INF
INF INF INF INF
INF INF INF INF
INF INF INF INF

Path taken:

1->3
3->4
4->2
2->1

Total Cost :: 80
*/