ASSIGNMENT 4

```
#include<bits/stdc++.h>
using namespace std;
#define N 4
#define INF INT_MAX

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

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;
}

void reduce_row(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];
            }
        }
    }
}

void reduce_column(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];
            }
        }
    }
}

void display_matrix(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<<"∞ ";
```

```
            }
            else
            {
                cout<<matrix_reduced[i][j]<<" ";
            }
        }
        cout<<"\n";
    }
}

int compute_cost(int matrix_reduced[N][N]){
    int cost = 0;
    int row[N];
    reduce_row(matrix_reduced, row);
    int col[N];
    reduce_column(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";
    display_matrix(matrix_reduced);

    return cost;
}


void display_path(vector<pair<int, int> > const &list)
{
    cout << "\n\nPath :: \n\n";
    for (int i = 0; (unsigned) i < list.size(); i++) {
        cout << list[i].first + 1 <<" -> "<< list[i].second + 1 << endl;
    }
}

struct comp
{
    // comparator structure for node cost (operator override)
    bool operator()(const Node* lhs, const Node* rhs) const
    {
        return lhs->cost > rhs->cost;
```

```
    }
};

int solveTSP(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 = compute_cost(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));
            display_path(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] +
compute_cost(child->matrix_reduced);
                pq.push(child);
            }
        }
        delete min;
    }
    return 0;
}

int main(){
    int costMatrix[N][N], result;
    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 = solveTSP(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 ::

∞ 0 0 0
0 ∞ 20 5
0 20 ∞ 5
0 5 5 ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
```

```
∞ ∞ 10 0
0 ∞ ∞ 5
0 ∞ 0 ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
0 ∞ ∞ 5
∞ 10 ∞ 0
0 0 ∞ ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
0 ∞ 20 ∞
0 20 ∞ ∞
∞ 0 0 ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
∞ ∞ 0 ∞
0 ∞ ∞ ∞
∞ ∞ ∞ ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
0 ∞ ∞ ∞
∞ 0 ∞ ∞
∞ ∞ ∞ ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
∞ ∞ ∞ 0
∞ ∞ ∞ ∞
0 ∞ ∞ ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
```

```
0 ∞ ∞ ∞
∞ ∞ ∞ ∞
∞ 0 ∞ ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
∞ ∞ ∞ ∞
∞ ∞ ∞ 0
0 ∞ ∞ ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
∞ ∞ ∞ ∞
0 ∞ ∞ ∞
∞ ∞ 0 ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
∞ ∞ ∞ ∞
∞ ∞ ∞ ∞
∞ ∞ ∞ ∞

Reduced Matrix ::

∞ ∞ ∞ ∞
∞ ∞ ∞ ∞
∞ ∞ ∞ ∞
∞ ∞ ∞ ∞


Path ::

1 — 3
3 — 4
4 — 2
2 — 1


Total Cost :: 80
```