Assignment No : 4

Travelling Salesman Problem

Code

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;
const int N = 5;
const int INF = INT_MAX;
struct Pair {
 int first, second;
 Pair(int first, int second) : first(first),
second(second) {}
};
struct Node {
 vector<Pair> path;
 vector<vector<int>> reducedMatrix;
 int cost;
 int vertex;
 int level;
 Node(int n) {
 reducedMatrix.resize(n,
vector<int>(n, 0));
 }
};
struct Comp {
 bool operator()(const Node& lhs, const
```

```
Node& rhs) {
 return lhs.cost > rhs.cost;
 }
};
Node newNode(vector<vector<int>>&
parentMatrix, vector<Pair>& path, int
level, int i, int j) {
 Node node(N);
 node.path = path;
 if (level != 0) {
 node.path.push_back(Pair(i, j));
 }
 for (int x = 0; x < N; x++) {
 copy(parentMatrix[x].begin(),
parentMatrix[x].end(),
back_inserter(node.reducedMatrix[x]));
 }
 for (int k = 0; level != 0 && k < N; k++) {
 node.reducedMatrix[i][k] = INF;
 node.reducedMatrix[k][j] = INF;
 }
 node.reducedMatrix[j][0] = INF;
 node.level = level;
 node.vertex = j;
 return node;
}
int rowReduction(vector<vector<int>>&
reducedMatrix, vector<int>& row) {
 for (int i = 0; i < N; i++) {
 row[i] = INF;
 for (int j = 0; j < N; j++) {
```

```
 if (reducedMatrix[i][j] < row[i]) {
 row[i] = reducedMatrix[i][j];
 }
 }
 }
 int cost = 0;
 for (int i = 0; i < N; i++) {
 for (int j = 0; j < N; j++) {
 if (reducedMatrix[i][j] != INF &&
row[i] != INF) {
 reducedMatrix[i][j] -= row[i];
 }
 if (reducedMatrix[i][j] < INF) {
 cost += reducedMatrix[i][j];
 }
 }
 }
 return cost;
}
int
columnReduction(vector<vector<int>>&
reducedMatrix, vector<int>& col) {
 for (int i = 0; i < N; i++) {
 col[i] = INF;
 for (int j = 0; j < N; j++) {
 if (reducedMatrix[i][j] < col[i]) {
 col[i] = reducedMatrix[i][j];
 }
 }
 }
 int cost = 0;
```

```
 for (int i = 0; i < N; i++) {
 for (int j = 0; j < N; j++) {
 if (reducedMatrix[i][j] != INF &&
col[j] != INF) {
 reducedMatrix[i][j] -= col[j];
 }
 if (reducedMatrix[i][j] < INF) {
 cost += reducedMatrix[i][j];
 }
 }
 }
 return cost;
}
int calculateCost(vector<vector<int>>&
reducedMatrix) {
 vector<int> row(N);
 vector<int> col(N);
 int cost = 0;
 cost += rowReduction(reducedMatrix,
row);
 cost +=
columnReduction(reducedMatrix, col);
 return cost;
}
void printPath(const vector<Pair>& list) {
 for (const Pair& pair : list) {
 cout << (pair.first + 1) << " -> " <<
(pair.second + 1) << endl;
 }
}
int solve(vector<vector<int>>&
```

```
costMatrix) {
 priority_queue<Node, vector<Node>,
Comp> pq;
 vector<Pair> v;
 vector<vector<int>> parentMatrix(N,
vector<int>(N, 0));
 for (int x = 0; x < N; x++) {
 copy(costMatrix[x].begin(),
costMatrix[x].end(),
back_inserter(parentMatrix[x]));
 }
 Node root = newNode(parentMatrix, v,
0, -1, 0);
 root.cost =
calculateCost(root.reducedMatrix);
 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(Pair(i, 0));
 printPath(min.path);
 return min.cost;
 }
 for (int j = 0; j < N; j++) {
 if (min.reducedMatrix[i][j] != INF) {
 Node child =
newNode(min.reducedMatrix, min.path,
min.level + 1, i, j);
 child.cost = min.cost +
```

```
min.reducedMatrix[i][j] +
calculateCost(child.reducedMatrix);
 pq.push(child);
 }
 }
 }
return -1; // In case no solution is found
}
int main() {
 vector<vector<int>> costMatrix = {
 {INF, 10, 8, 9, 7},
 {10, INF, 10, 5, 6},
 {8, 10, INF, 8, 9},
 {9, 5, 8, INF, 6},
 {7, 6, 9, 6, INF}
 };
 cout << "Total cost is " <<
solve(costMatrix) << endl;
 return 0;
}
```

OUTPUT

Track

1 -> 2 -> 5 -> 4 -> 5 -> 1

0