|  |
| --- |
|  |
| **CSE 2208 Algorithms Lab**  **Assignment No**:04    **Assignment Topic**:  1. Travelling salesman problem (Using Branch and bound) |

|  |  |
| --- | --- |
| **Date of Performance**: 04-03-2020  **Date of Submission**: 11-03-2020 | **Name**: Mubina Ashrafi  **Student ID**: 18-01-04-030  **Lab Group**: A2  **Department of CSE, AUST.** |

1. Travelling salesman problem (Using Branch and bound)

#include <iostream>

#include <vector>

#include <queue>

#include <utility>

#include <cstring>

#include <climits>

using namespace std;

#define N 5

#define INF INT\_MAX

struct Node

{

vector<pair<int, int>> path;

int reducedMatrix[N][N];

int cost;

int vertex;

int level;

};

Node\* newNode(int parentMatrix[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->reducedMatrix, parentMatrix,

sizeof node->reducedMatrix);

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(int reducedMatrix[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 (reducedMatrix[i][j] < row[i])

row[i] = reducedMatrix[i][j];

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];

}

int columnReduction(int reducedMatrix[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 (reducedMatrix[i][j] < col[j])

col[j] = reducedMatrix[i][j];

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];

}

int calculateCost(int reducedMatrix[N][N])

{

int cost = 0;

int row[N];

rowReduction(reducedMatrix, row);

int col[N];

columnReduction(reducedMatrix, col);

for (int i = 0; i < N; i++)

cost += (row[i] != INT\_MAX) ? row[i] : 0,

cost += (col[i] != INT\_MAX) ? col[i] : 0;

return cost;

}

void printPath(vector<pair<int, int>> const &list)

{

for (int i = 0; i < list.size(); i++)

cout << list[i].first + 1 << " -> "

<< list[i].second + 1 << endl;

}

struct comp {

bool operator()(const Node\* lhs, const Node\* rhs) const

{

return lhs->cost > rhs->cost;

}

};

int solve(int costMatrix[N][N])

{

priority\_queue<Node\*, std::vector<Node\*>, comp> pq;

vector<pair<int, int>> v;

Node\* root = newNode(costMatrix, 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(make\_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);

}

}

delete min;

}

}

int main()

{

// cost 34

int costMatrix[N][N] =

{

{ 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 << "\n\nTotal Cost is " << solve(costMatrix);

return 0;

}