МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра

обчислювальної техніки

**ЛАБОРАТОРНА РОБОТА 4**

з дисципліни

“Операційні Системи”

на тему

"**Дослідження принципів проектування та роботи динамічного та статичного планування**"

Студента 3 курсу

групи ІП-53

Сулими О.С.

Перевірив: Сімоненко А.В.

Київ 2017

**Завдання:**

12. Скласти програму визначення (локалізації) зони пошуку рішення для графа з пересилками (число вершин не менше 30).

**Лістинг:**

#include <map>

#include <vector>

#include <conio.h>

#include <iostream>

#include <algorithm>

using namespace std;

struct Top

{

int number;

int weight;

vector<Top\*> neightbours;

vector<int> weightEdges;

bool free = false;

bool parent = false;

};

void critical(vector<int>\* path, int\* length, Top top);

void copyVector(vector<int>\* dest, vector<int> source);

bool condition(vector<Top\*> tops);

void main()

{

int nTops, nEdges;

cin >> nTops >> nEdges;

vector<int> weightTops;

int\*\* adjacencyMatrix = new int\*[nTops];

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

{

adjacencyMatrix[i] = new int[nTops];

for (int j = 0; j < nTops; j++)

{

adjacencyMatrix[i][j] = -1;

}

}

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

{

int num, weightEdge, neightbour;

cin >> num >> weightEdge >> neightbour;

adjacencyMatrix[num][neightbour] = weightEdge;

}

/\*for (int i = 0; i < nTops; i++)

{

for (int j = 0; j < nTops; j++)

{

cout << adjacencyMatrix[i][j] << " ";

}

cout << endl;

}\*/

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

{

int weight;

cin >> weight;

weightTops.push\_back(weight);

}

vector<Top\*> tops(nTops);

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

{

for (int j = 0; j < nTops; j++)

{

if (tops[i] == nullptr)

tops[i] = new Top();

tops[i]->number = i;

tops[i]->weight = weightTops[i];

if (adjacencyMatrix[i][j] > -1)

{

if (tops[j] == nullptr)

tops[j] = new Top();

tops[i]->neightbours.push\_back(tops[j]);

tops[i]->weightEdges.push\_back(adjacencyMatrix[i][j]);

}

}

}

Top\* top = new Top();

top->number = 1;

top->weight = 2;

Top\* top1 = new Top();

top1->number = 2;

top1->weight = 2;

Top\* top2 = new Top();

top2->number = 3;

top2->weight = 2;

Top\* top3 = new Top();

top3->number = 4;

top3->weight = 1;

top->neightbours.push\_back(top1);

top->neightbours.push\_back(top2);

top->weightEdges.push\_back(2);

top->weightEdges.push\_back(4);

top2->neightbours.push\_back(top3);

top2->weightEdges.push\_back(5);

while (condition(tops))

{

vector<int> criticalPath;

int maxLength = 0;

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

{

if (tops[i]->free)

continue;

vector<int> path;

int length = 0;

critical(&path, &length, \*tops[i]);

if (length > maxLength)

{

maxLength = length;

copyVector(&criticalPath, path);

}

}

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

{

tops[criticalPath[i]]->free = true;

}

for (int i = 1; i < criticalPath.size(); i++)

{

int numEdge;

for (int j = 0; j < tops[criticalPath[i - 1]]->neightbours.size(); j++)

{

if (tops[criticalPath[i - 1]]->neightbours[j]->number == criticalPath[i])

{

numEdge = j;

}

}

tops[criticalPath[i - 1]]->weightEdges[numEdge] = 0;

}

}

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

{

tops[i]->free = false;

}

vector<int> criticalPath;

int maxLength = 0;

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

{

if (tops[i]->free)

continue;

vector<int> path;

int length = 0;

critical(&path, &length, \*tops[i]);

if (length > maxLength)

{

maxLength = length;

copyVector(&criticalPath, path);

}

}

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

{

cout << criticalPath[i] << " ";

}

cout << endl;

\_getch();

}

void critical(vector<int>\* path, int\* length, Top top)

{

path->push\_back(top.number);

\*length += top.weight;

if (top.neightbours.size() == 0)

return;

if (top.neightbours.size() > 1)

{

int myMax = 0;

int maxI;

vector<vector<int>> myPathes;

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

{

if (top.neightbours[i]->free)

continue;

vector<int> myPath;

copyVector(&myPath, \*path);

int myLength = top.weightEdges[i];

critical(&myPath, &myLength, \*top.neightbours[i]);

myPathes.push\_back(myPath);

if (myLength >= myMax)

{

myMax = myLength;

maxI = myPathes.size() - 1;

}

}

\*length += myMax;

copyVector(path, myPathes[maxI]);

return;

}

if (top.neightbours[0]->free)

return;

critical(path, length, \*top.neightbours[0]);

\*length += top.weightEdges[0];

}

void copyVector(vector<int>\* dest, vector<int> source)

{

dest->clear();

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

{

dest->push\_back(source[i]);

}

}

bool condition(vector<Top\*> tops)

{

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

{

if (!tops[i]->free)

return true;

}

return false;

}