**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по курсовой работе**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Потоки в сетях**

**Вариант 3**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8301 |  | Онвука П.А. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020

**Цель работы**

Реализовать алгоритм проталкивания предпотока в С++.

**Постановка задачи**

Реализовать алгоритм проталкивания предпотока для нахождения максимально потока в транспортной сети.

**Описание алгоритма решения и** **Обоснование выбора структур данных**

Алгоритм проталкивания предпотока решает задачу нахождения максимального потока в транспортной сети. Пусть дан граф G, в котором выделены две вершины: исток S и сток T, а у каждого ребра определена пропускная способность C(u,v). Поток F можно представить как поток вещества, которое могло бы пройти по сети от истока к стоку, если рассматривать граф как сеть труб с некоторыми пропускными способностями. То есть поток — функция F(u,v), определённая на множестве рёбер графа. Задача заключается в нахождении максимального потока.

Для реализации алгоритма использовал класс stream, включающий класс string( который состоит из двух char - названий вершин, и целого числа – веса ребра между вершинами).

List представляет собой текст из файла и для хранения списка вершин.

матрица смежности используется для представления графа.

**Оценка временной сложности методов**

|  |  |
| --- | --- |
| метод | Временная сложность |
| input() | O(n) |
| SetVertexCount() | O(n2) |
| CreateAdjacencyMatrix() | O(n3) |
| printMatrix() | O(n2) |
| getMaxFlow() | O(n4) |

**Пример работы программы**

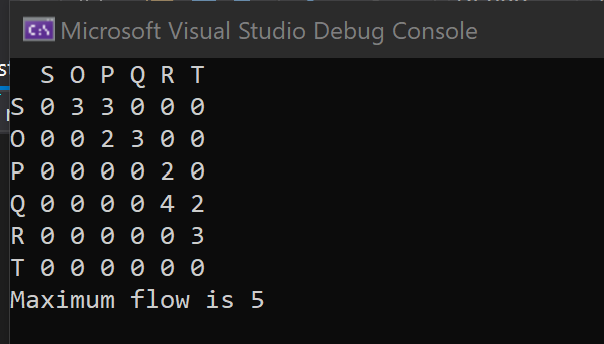
****

Рисунок 1

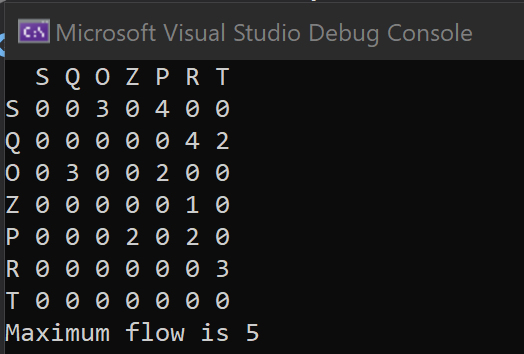


Рисунок 2

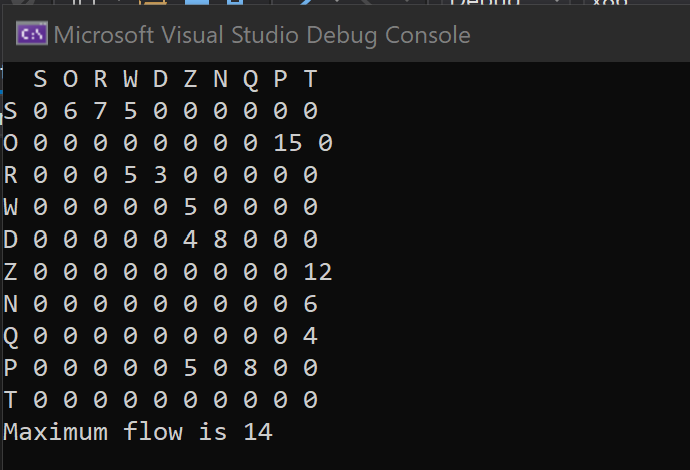


Рисунок 3

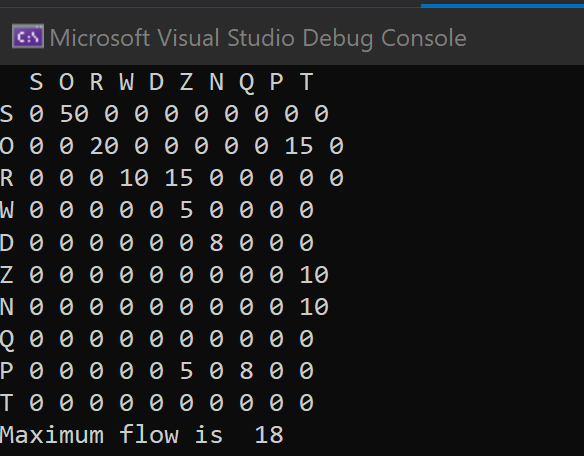


Рисунок 4

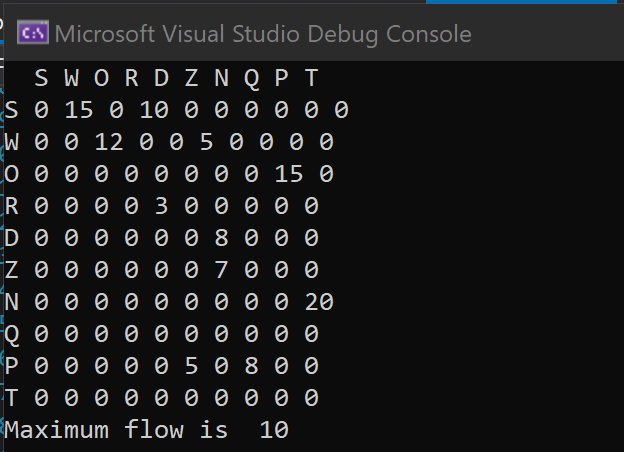


Рисунок 5

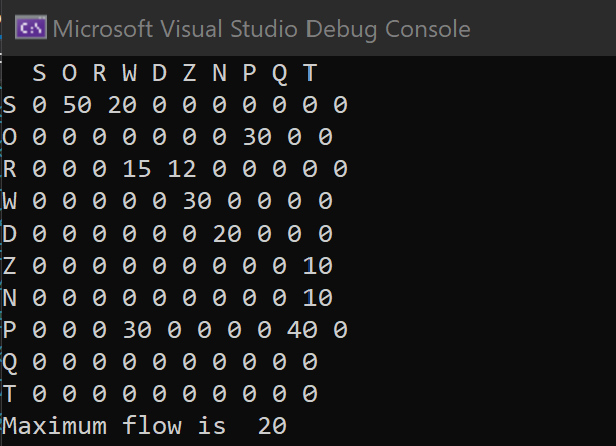


Рисунок 6

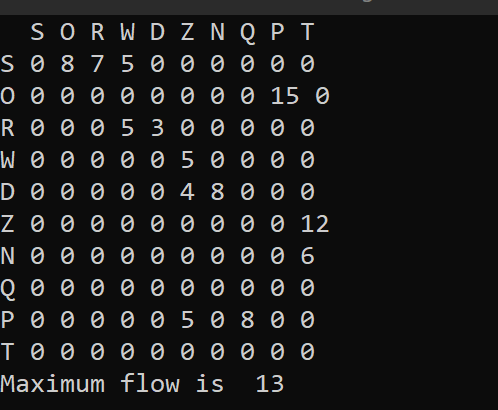


Рисунок 7

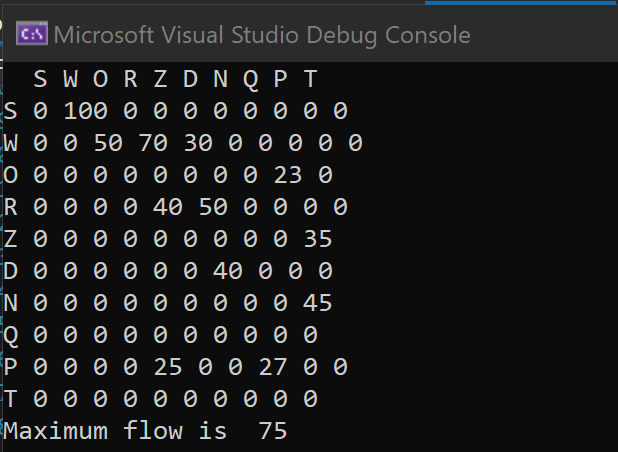


Рисунок 8

**Листинг**

#include "Flow.h"

using namespace std;

void stream::input(ifstream& file) //input text from text file

{

if (file.eof() != true)

{

file.unsetf(ios::skipws);

List<char>\* temp\_number;

string\* str;

while (!file.eof())

{

int step = 1;

temp\_number = new List<char>;

str = new string;

char symbol = 0;

while (symbol != '\n')

{

file >> symbol;

if (file.eof())

return;

if (symbol == ' ' || symbol == '\n')

{

if (step == 2)

{

temp\_number = new List<char>;

}

if (step == 3)

{

if (temp\_number->get\_size() == 0)

throw out\_of\_range("wrong number");

else

{

int temp = 0;

for (int i = 0; i < temp\_number->get\_size(); i++)

{

if (temp\_number->at(i) >= 48 && temp\_number->at(i) <= 57)

{

temp = temp \* 10 + (temp\_number->at(i) - 48);

}

else

throw out\_of\_range("wrong number");

}

str->weight = temp;

}

temp\_number = new List<char>;

}

step++;

}

else {

switch (step)

{

case 1:

str->vertex\_1 = symbol;

break;

case 2:

str->vertex\_2 = symbol;

break;

case 3:

temp\_number->push\_back(symbol);

break;

default:

break;

}

}

}

if (step == 1)

throw out\_of\_range("wrong string");

this->File.push\_back(str);

}

}

}

void stream::SetVertexCount()

{

for (int i = 0; i < this->File.get\_size(); i++)

{

int step = 1;

while (step != 3)

{

if (this->All\_vertex.get\_size() == 0)

{

switch (step)

{

case 1:

this->All\_vertex.push\_back(this->File.at(i)->vertex\_1);

break;

case 2:

this->All\_vertex.push\_back(this->File.at(i)->vertex\_2);

break;

}

}

else

{

int match = 0;

switch (step)

{

case 1:

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

{

if (this->File.at(i)->vertex\_1 == this->All\_vertex.at(j))

{

match = 1;

break;

}

}

break;

case 2:

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

{

if (this->File.at(i)->vertex\_2 == this->All\_vertex.at(j))

{

match = 1;

break;

}

}

break;

}

if (match == 0)

{

switch (step)

{

case 1:

this->All\_vertex.push\_back(this->File.at(i)->vertex\_1);

break;

case 2:

this->All\_vertex.push\_back(this->File.at(i)->vertex\_2);

break;

}

}

}

step++;

}

}

}

void stream::CreateAdjacencyMatrix()

{

int source = -1;

int sink = -1;

for (int i = 0; i < this->All\_vertex.get\_size(); i++) {

if (this->All\_vertex.at(i) == 'S')

source = i;

if (this->All\_vertex.at(i) == 'T')

sink = i;

}

if (sink == -1 || source== -1)

throw out\_of\_range("invalid graph");

this->All\_vertex.swap(0, source);

this->All\_vertex.swap(this->All\_vertex.get\_size() - 1, sink);

this->matrix = new int\* [this->All\_vertex.get\_size()];

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

this->matrix[i] = new int[this->All\_vertex.get\_size()];

}

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

{

this->matrix[i][j] = 0;

}

}

for (int i = 0; i < this->File.get\_size(); i++)

{

int step = 1;

int index\_1 = -1;

int index\_2 = -1;

while (step != 3)

{

int match = 0;

switch (step)

{

case 1:

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

{

if (this->File.at(i)->vertex\_1 == this->All\_vertex.at(j))

{

match = 1;

index\_1 = j;

break;

}

else

{

match = 0;

index\_1 = -1;

}

}

break;

case 2:

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

{

if (this->File.at(i)->vertex\_2 == this->All\_vertex.at(j))

{

match = 1;

index\_2 = j;

break;

}

else

{

match = 0;

index\_2 = -1;

}

}

break;

}

step++;

}

this->matrix[index\_1][index\_2] = this->File.at(i)->weight;

}

}

void stream::printMatrix() //print Adjacency Matrix

{

cout << " ";

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

cout << " " << this->All\_vertex.at(i);

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

cout << endl;

cout << this->All\_vertex.at(i) << " ";

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

{

cout << this->matrix[i][j] << " ";

}

}

cout << endl;

}

bool stream::IsTransport\_network()

{

int\*\* temp\_matrix = new int\* [this->All\_vertex.get\_size()];

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

temp\_matrix[i] = new int[this->All\_vertex.get\_size()];

}

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

{

if (this->matrix[i][j] != 0)

temp\_matrix[i][j] = 1; // 1 it is white color, 2 it is gray, 3 it is black

else

temp\_matrix[i][j] = 0;

}

}

bool this\_matrix = true;

int double\_match = -1;

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

if (temp\_matrix[j][0] != 0)

temp\_matrix[j][0] = 2; //gray color mean that you in this vertex

//black mean that you left from this vertex

Find(0, this->All\_vertex.get\_size() - 1, temp\_matrix, this\_matrix);

return this\_matrix;

}

void stream::Find(int top, int end, int\*\* color\_matrix, bool& temp)

{

int match = 0;

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

if (this->matrix[top][i] != 0)

{

switch (color\_matrix[top][i])

{

case 1:

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

if (color\_matrix[j][i] != 0)

color\_matrix[j][i] = 2;

break;

case 2:

temp = false;

return;

break;

default:

break;

}

match++;

Find(i, end, color\_matrix, temp);

}

}

if (match == 0 && top == end) {

temp \* true;

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

if (color\_matrix[j][top] != 0)

color\_matrix[j][top] = 3;

return;

}

else if (match == 0 && top != end) {

temp = false;

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

if (color\_matrix[j][top] != 0)

color\_matrix[j][top] = 3;

return;

}

else {

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

if (color\_matrix[j][top] != 0)

color\_matrix[j][top] = 3;

return;

}

}

void stream::push(int previous\_vertex, int next\_vertex, int\*\* flow, int\* excess\_flow) //push stream to next vertex

{

int possible\_flow;

if (excess\_flow[previous\_vertex] < this->matrix[previous\_vertex][next\_vertex] - flow[previous\_vertex][next\_vertex])

possible\_flow = excess\_flow[previous\_vertex];

else

possible\_flow = this->matrix[previous\_vertex][next\_vertex] - flow[previous\_vertex][next\_vertex];

flow[previous\_vertex][next\_vertex] = flow[previous\_vertex][next\_vertex] + possible\_flow;

flow[next\_vertex][previous\_vertex] = -flow[previous\_vertex][next\_vertex];

excess\_flow[previous\_vertex] = excess\_flow[previous\_vertex] - possible\_flow;

excess\_flow[next\_vertex] = excess\_flow[next\_vertex] + possible\_flow;

}

bool stream::lift(int vertex, int\* height\_of\_vertexes, int\*\* flow, int max) //lift vertex

{

int min\_height = max;

for (int i = 0; i < this->All\_vertex.get\_size(); i++) {

if (this->matrix[vertex][i] - flow[vertex][i] > 0) {

if (min\_height > height\_of\_vertexes[i])

min\_height = height\_of\_vertexes[i];

}

}

if (min\_height == max)

return false;

height\_of\_vertexes[vertex] = min\_height + 1;

return true;

}

int stream::getMaxFlow() //pushing the pre-stream algorithm

{

if (this->IsTransport\_network() != true)

throw std::out\_of\_range("wrong graph");

int max = this->All\_vertex.get\_size() + 1;// maximum weight

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

if (this->matrix[i][j] > max)

max = this->matrix[i][j];

max++;

int\*\* flow = new int\* [this->All\_vertex.get\_size()];

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

flow[i] = new int[this->All\_vertex.get\_size()];

}

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

for (int j = 0; j < this->All\_vertex.get\_size(); j++)

flow[i][j] = 0;

}

for (int i = 1; i < this->All\_vertex.get\_size(); i++)

{

flow[0][i] = this->matrix[0][i];

flow[i][0] = -this->matrix[0][i];

}

int\* height\_of\_vertexes = new int[this->All\_vertex.get\_size()];

height\_of\_vertexes[0] = this->All\_vertex.get\_size();

for (int i = 1; i < this->All\_vertex.get\_size(); i++)

height\_of\_vertexes[i] = 0;

int\* excess\_flow = new int[this->All\_vertex.get\_size()];

for (int i = 1; i < this->All\_vertex.get\_size(); i++)

excess\_flow[i] = flow[0][i];

for (; ;)

{

int i;

for (i = 1; i < this->All\_vertex.get\_size() - 1; i++)

if (excess\_flow[i] > 0)

break;

if (i == this->All\_vertex.get\_size() - 1)

break;

int j;

for (j = 0; j < this->All\_vertex.get\_size(); j++)

if (this->matrix[i][j] - flow[i][j] > 0 && height\_of\_vertexes[i] == height\_of\_vertexes[j] + 1)

break;

if (j < this->All\_vertex.get\_size())

push(i, j, flow, excess\_flow);

else

if (lift(i, height\_of\_vertexes, flow, max) != true)

break;

}

int max\_flow = 0;

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

if (flow[i][this->All\_vertex.get\_size() - 1])

max\_flow += flow[i][this->All\_vertex.get\_size() - 1];

cout << "Maximum flow is " << max\_flow << endl;

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

delete[] flow[i];

}

return max\_flow;

}

#include "List.cpp"

#include <fstream>

#include <iostream>

class stream

{

private:

class string

{

public:

char vertex\_1;

char vertex\_2;

int weight;

string() :vertex\_1(0), vertex\_2(0), weight(0) {};

~string() {}

};

List<string\*> File;;

int\*\* matrix;

List<char> All\_vertex;

public:

stream() : matrix(NULL) {};

~stream() {

File.~List();

All\_vertex.~List();

for (int i = 0; i < this->All\_vertex.get\_size(); i++)

{

delete[] matrix[i];

}

}

void input(ifstream& file); //input text from text file

void SetVertexCount(); //find all Vertex

void CreateAdjacencyMatrix(); //create Adjacency Matrix

void printMatrix(); //print Adjacency Matrix

bool IsTransport\_network();//checks if a graph is a transport network

void Find(int top, int end, int\*\* color\_matrix, bool& temp);//dfs algoritm with color

void push(int u, int v, int\*\* f, int\* e);//push stream to next vertex

bool lift(int u, int\* h, int\*\* f, int max);//lift vertex

int getMaxFlow();//pushing the pre-stream algorithm

int get\_file\_size() //count of strings

{

return this->File.get\_size();

}

int get\_number\_of\_vertex()//count of veretxs

{

return this->All\_vertex.get\_size();

}

};

#include "Flow.h"

int main()

{

stream graph;

ifstream file;

file.open("Ten\_vertices\_7.txt");

graph.input(file);

graph.SetVertexCount();

graph.CreateAdjacencyMatrix();

graph.printMatrix();

graph.IsTransport\_network();

graph.getMaxFlow();

file.close();

}

TEST

#include "pch.h"

#include "CppUnitTest.h"

#include "../maxFlow/Flow.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace maxFlowTest

{

TEST\_CLASS(maxFlowTest)

{

public:

Assert assert;

TEST\_METHOD(Input\_)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Text1.txt");

matrix.input(file);

assert.AreEqual(matrix.get\_file\_size(), 8);

file.close();

}

TEST\_METHOD(Input\_InvalidGraph)

{

stream\* matrix;

matrix = new stream();

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\invalid\_graph.txt");

matrix->input(file);

matrix->SetVertexCount();

auto funPtr = [matrix] {matrix->CreateAdjacencyMatrix(); };

assert.ExpectException<std::out\_of\_range>(funPtr);

file.close();

}

TEST\_METHOD(Test\_with\_negative\_input)

{

stream\* matrix;

matrix = new stream();

auto funPtr = [matrix] {

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\negative\_weight.txt");

matrix->input(file); };

assert.ExpectException<std::out\_of\_range>(funPtr);

}

TEST\_METHOD(\_Input\_no\_flow\_from\_S\_to\_T)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\no\_flow.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.IsTrue(matrix.IsTransport\_network());

file.close();

}

TEST\_METHOD(TestSetVertexCount\_1)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Text1.txt");

matrix.input(file);

matrix.SetVertexCount();

assert.AreEqual(matrix.get\_number\_of\_vertex(), 6);

}

TEST\_METHOD(TestSetVertexCount\_2)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Text2.txt");

matrix.input(file);

matrix.SetVertexCount();

assert.AreEqual(matrix.get\_number\_of\_vertex(), 7);

}

TEST\_METHOD(IsTransportNetwork\_1)

{

//normal transport graph

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Text2.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.IsTrue(matrix.IsTransport\_network());

}

TEST\_METHOD(IsTransportNetwork\_2)

{

//graph with 2 end vertexes

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Text3.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.IsFalse(matrix.IsTransport\_network());

}

TEST\_METHOD(IsTransportNetwork\_3)

{

//graph with a cycle

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\cycle\_graph.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.IsFalse(matrix.IsTransport\_network());

}

TEST\_METHOD(CreateAdjencyMatrix\_)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Text2.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

}

TEST\_METHOD(getMaxFlow\_1)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Text1.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 5);

}

TEST\_METHOD(getMaxFlow\_2)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Text2.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 7);

}

TEST\_METHOD(getMaxFlow\_3)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Text4.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 7);

}

TEST\_METHOD(getMaxFlow\_4)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\text5.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 10);

}

TEST\_METHOD(getMaxFlow\_Ten\_vertices\_1)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Ten\_vertices\_1.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 21);

}

TEST\_METHOD(getMaxFlow\_Ten\_vertices\_2)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Ten\_vertices\_2.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 26);

}

TEST\_METHOD(getMaxFlow\_Ten\_vertices\_3)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Ten\_vertices\_3.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 28);

}

TEST\_METHOD(getMaxFlow\_Ten\_vertices\_4)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Ten\_vertices\_4.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 14);

}

TEST\_METHOD(getMaxFlow\_Ten\_vertices\_5)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Ten\_vertices\_5.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 18);

}

TEST\_METHOD(getMaxFlow\_Ten\_vertices\_6)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Ten\_vertices\_6.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 50);

}

TEST\_METHOD(getMaxFlow\_Ten\_vertices\_7)

{

stream matrix;

ifstream file;

file.open("C:\\Users\\DCLASSICGENIUS\\source\\repos\\maxFlow\\maxFlowTest\\Ten\_vertices\_7.txt");

matrix.input(file);

matrix.SetVertexCount();

matrix.CreateAdjacencyMatrix();

assert.AreEqual(matrix.getMaxFlow(), 95);

}

};

}