**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 реализуемых и вспомогательных классов

Класс Flow, содержит поля: int\* excess flow (массив избытков вершин), int\*\* capacity\_edge (остаточная сеть), int vertexCount (количество верщин), int sourceVertex (исток), int destinationVertex (сток).

Класс содержит следующие методы:

*void push(int edge, int vertex)* – функция, проталкивающая поток из u в v, равный min{e[edge],cf(edge, vertex)}, и пересчитывающая остаточную сеть и избытки.

*void lift(int edge)* – функция, поднимающая вершину на минимальную высоту, достаточную для возможности проталкивания потока

*void discharge(int edge)* – функция, выполняющая лифтинг и проталкивание, пока это возможно

*int max\_flow()* – функция, вычисляющая максимальный поток в сети

Оценка временной сложности алгоритмов

|  |  |
| --- | --- |
| Функции | Оценка временной сложности |
| push(int edge, int vertex) | O(1) |
| int max\_flow() | O(|V|2 |E|). |
| void discharge(int edge) | O(|V| |E|) |
| void lift(int edge) | O(|V|) |

Описание реализованных unit-тестов

Внедренный тест проверяет правильность ввода файла. Он также проверяет правильность обработки исключений, например, при вводе файла без источника или с отрицательным целым числом. Тест использует граф с 6 и 20 вершинами для проверки правильности реализации алгоритма.

**Обоснование выбора используемых структур данных**

Map использовался для индивидуализации вершин с индексами, поскольку он позволяет не сохранять дублирующиеся данные, а также для быстрого доступа к ним. Список использовался для перебора сетевых узлов в

функции max\_flow. Список использовался потому, что он более удобен в отличие от массива, также мы можем быстро добавлять и удалять элементы из списка, не переписывая их как массив.

**Примеры работы программы**

S O 20

S R 10

R W 5

R D 3

W Z 5

D N 8

Z T 10

N T 10

O P 15

P Q 8

P Z 5

Q T 10



S O 50

S R 20

R W 15

R D 12

W Z 30

D N 20

P W 30

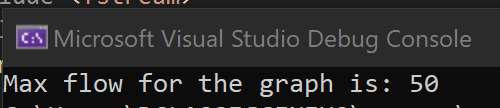
Z T 10

N T 10

O P 30

P Q 40

Q T 50



S O 20

S R 20

O W 5

R W 5

R D 3

W Z 12

D N 8

Z Q 10

Z T 10

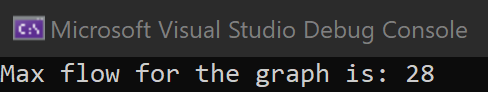
N T 10

O P 15

P Q 15

P Z 5

Q T 20



S O 50

O R 20

R W 10

R D 15

W Z 5

D N 8

Z T 10

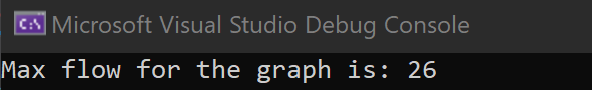
N T 10

O P 15

P Q 8

P Z 5

Q T 10



S W 100

W O 50

W R 70

R Z 40

R D 50

W Z 30

D N 40

Z T 35

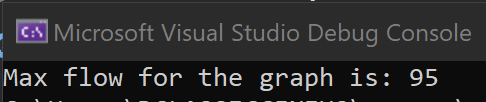
N T 45

O P 23

P Q 27

P Z 25

Q T 20



S O 3

S P 3

O Q 3

O P 2

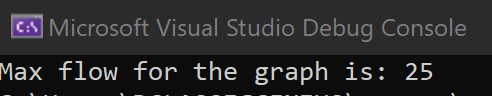
P R 2

Q R 4

Q T 2

R T 3

S T 20



S E 10

S A 14

A F 7

A B 10

B G 1

B I 5

B C 2

C I 2

C D 1

D K 4

E L 8

E F 9

F M 6

F G 8

G N 4

G H 3

H I 2

I J 7

I O 10

I R 3

I D 2

K R 3

L O 5

L M 7

M P 3

M N 4

M G 5

N Q 13

N H 1

J N 6

J Q 1

J U 5

J R 4

O P 4

O M 2

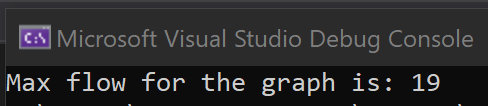
P N 2

Q T 7

Q U 3

R T 11

U T 5



S W 15

W O 12

S R 10

R D 3

W Z 5

D N 8

Z N 7

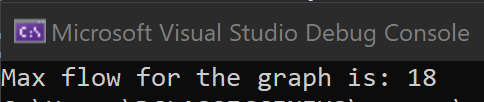
N T 20

O P 15

P Q 8

P Z 5

Q T 10



Листинг

Flow.h

#pragma once

#include <fstream>

#include "List.h"

#include<string>

#include"Map.h"

using namespace std;

class Flow {

public:

Flow(ifstream& file)

{

Map<char, int>\* Map\_from\_char\_to\_number = new Map<char, int>();

vertexCount = 0;

int str\_num = 1;

while (!file.eof()) {

string s1;

getline(file, s1);

if (s1.size() >= 5) {// minimum possible input(two letters, two spaces,one digit)

if (!((s1[0] >= 'A' && s1[0] <= 'Z') && (s1[1] == ' '))) {

throw exception("Error entering the first character in the string or missing a space after it. ");

}

if (!((s1[2] >= 'A' && s1[2] <= 'Z') && (s1[3] == ' '))) {

throw exception("Error entering the second character in the string or missing a space after it. ");

}

string cur;

for (int i = 4; i < s1.size(); i++) {

if (s1[i] >= '0' && s1[i] <= '9')

cur += s1[i];

else {

throw exception("Error entering the third character in the string, the presence of a space after it or a negative number. ");

}

}

if (!Map\_from\_char\_to\_number->find\_is(s1[0])) {

Map\_from\_char\_to\_number->insert(s1[0], vertexCount);

++vertexCount;

}

if (!Map\_from\_char\_to\_number->find\_is(s1[2])) {

Map\_from\_char\_to\_number->insert(s1[2], vertexCount);

++vertexCount;

}

}

else

{

throw exception("A data-entry error. Check the correctness of the input in the file. ");

}

++str\_num;

}

if (Map\_from\_char\_to\_number->find\_is('S'))

sourceVertex = Map\_from\_char\_to\_number->find('S');

else {

throw exception("Source is missing");

}

if (Map\_from\_char\_to\_number->find\_is('T'))

destinationVertex = Map\_from\_char\_to\_number->find('T');

else {

throw exception("Sink is missing");

}

file.clear();

file.seekg(ios::beg);

excess\_flow = new int[vertexCount];

height = new int[vertexCount];

capacity\_edge = new int\* [vertexCount];

for (int i = 0; i < vertexCount; i++) {

excess\_flow[i] = 0;

height[i] = 0;

}

for (int i = 0; i < vertexCount; i++) {

capacity\_edge[i] = new int[vertexCount];

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

capacity\_edge[i][j] = 0;

}

str\_num = 1;

while (!file.eof()) {

string s1;

int vertex1, vertex2, capacity;

getline(file, s1);

vertex1 = Map\_from\_char\_to\_number->find(s1[0]);

vertex2 = Map\_from\_char\_to\_number->find(s1[2]);

if (vertex1 == vertex2)

throw exception("There is a path from the vertex to itself. ");

capacity\_edge[vertex1][vertex2] = stoi(s1.substr(4));

++str\_num;

}

}

template<typename T>

T min(T a, T b) {

return a > b ? b : a;

}

int max\_flow() {

if (vertexCount > 2) {

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

{

if (i == sourceVertex)

continue;

excess\_flow[i] = capacity\_edge[sourceVertex][i];

capacity\_edge[i][sourceVertex] += capacity\_edge[sourceVertex][i];

}

height[sourceVertex] = vertexCount;

List<int> list;

int current;

int current\_index = 0;

int old\_height;

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

if (i != sourceVertex && i != destinationVertex)

list.push\_front(i);

current = list.at(0);

while (current\_index < list.get\_size())

{

old\_height = height[current];

discharge(current);

if (height[current] != old\_height)

{

list.push\_front(current);

list.remove(++current\_index);

current = list.at(0);

current\_index = 0;

}

++current\_index;

if (current\_index < list.get\_size())

current = list.at(current\_index);

}

if (excess\_flow[destinationVertex] > 0)

return excess\_flow[destinationVertex];

else

throw exception("There is no path from the source to the sink");

}

else

return capacity\_edge[0][1];

}

void push(int edge, int vertex)

{

int f = min(excess\_flow[edge], capacity\_edge[edge][vertex]);

excess\_flow[edge] -= f;

excess\_flow[vertex] += f;

capacity\_edge[edge][vertex] -= f;

capacity\_edge[vertex][edge] += f;

}

void lift(int edge)

{

int min = 2 \* vertexCount + 1;

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

if (capacity\_edge[edge][i] && (height[i] < min))

min = height[i];

height[edge] = min + 1;

}

void discharge(int edge)

{

int vertex = 0;

while (excess\_flow[edge] > 0)

{

if (capacity\_edge[edge][vertex] && height[edge] == height[vertex] + 1)

{

push(edge, vertex);

vertex = 0;

continue;

}

vertex++;

if (vertex == vertexCount)

{

lift(edge);

vertex = 0;

}

}

}

~Flow() {

delete[] excess\_flow;

delete[] height;

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

delete[] capacity\_edge[i];

}

private:

int\* excess\_flow;

int\*\* capacity\_edge;

int\* height;

int vertexCount, sourceVertex, destinationVertex;

};

Flow.cpp

#include <iostream>

#include <fstream>

#include"Flow.h"

int main(int argc, char\* argv[])

{

try {

ifstream input("Ten\_vertices\_1.txt");

Flow flow(input);

int max\_flow = flow.max\_flow();

cout << "Max flow for the graph is: " << max\_flow;

}

catch (exception& error) {

cout << error.what();

}

return 0;

}

Unit test

#include "pch.h"

#include "CppUnitTest.h"

#include"../Flow/Flow.h"

#include <fstream>

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MaxFlowTests

{

TEST\_CLASS(MaxFlowTests)

{

public:

Assert assert;

TEST\_METHOD(Correct\_input\_for\_6\_vertexes)

{

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text1.txt");

Flow flow(textInput);

assert.AreEqual(flow.max\_flow(), 5);

}

TEST\_METHOD(TestMethod\_Exception\_entering\_the\_first\_character) {

try {

ifstream textIinput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text2.txt");

Flow flow(textIinput);

}

catch (exception& error) {

assert.AreEqual(error.what(), "Error entering the first character in the string or missing a space after it. ");

}

}

TEST\_METHOD(Exception\_entering\_the\_second\_character) {

try {

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text3.txt");

Flow flow(textInput);

}

catch (exception& error) {

assert.AreEqual(error.what(), "Error entering the second character in the string or missing a space after it. ");

}

}

TEST\_METHOD(Exception\_entering\_the\_third\_character) {

try {

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text4.txt");

Flow flow(textInput);

}

catch (exception& error) {

assert.AreEqual(error.what(), "Error entering the third character in the string, the presence of a space after it or a negative number. ");

}

}

TEST\_METHOD(Exception\_empty\_string) {

try {

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text5.txt");

Flow flow(textInput);

}

catch (exception& error) {

assert.AreEqual(error.what(), "A data-entry error. Check the correctness of the input in the file. ");

}

}

TEST\_METHOD(Correct\_output\_for\_6\_vertexes\_)

{

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text6.txt");

Flow flow(textInput);

assert.AreEqual(flow.max\_flow(), 25);

}

TEST\_METHOD(Correct\_output\_for\_2\_vertexes)

{

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text7.txt");

Flow flow(textInput);

assert.AreEqual(flow.max\_flow(), 20);

}

TEST\_METHOD(Exception\_path\_from\_vertex\_to\_itself) {

try {

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text8.txt");

Flow flow(textInput);

}

catch (exception& error) {

assert.AreEqual(error.what(), "There is a path from the vertex to itself. ");

}

}

TEST\_METHOD(Correct\_output\_for\_20\_vertexes)

{

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text9.txt");

Flow flow(textInput);

assert.AreEqual(flow.max\_flow(), 19);

}

TEST\_METHOD(Exception\_negative\_number\_in\_the\_input) {

try {

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text10.txt");

Flow flow(textInput);

}

catch (exception& error) {

assert.AreEqual(error.what(), "Error entering the third character in the string, the presence of a space after it or a negative number. ");

}

}

TEST\_METHOD(Exception\_missing\_sink) {

try {

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text11.txt");

Flow flow(textInput);

}

catch (exception& error) {

assert.AreEqual(error.what(), "Sink is missing");

}

}

TEST\_METHOD(Exception\_missing\_source) {

try {

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text12.txt");

Flow flow(textInput);

}

catch (exception& error) {

assert.AreEqual(error.what(), "Source is missing");

}

}

TEST\_METHOD(Exception\_two\_different\_connectivity\_components) {

try {

ifstream textInput("C:\\Users\\DCLASSICGENIUS\\source\\repos\\Flow\\maxFlowTest\\Text13.txt");

Flow flow(textInput);

}

catch (exception& error) {

assert.AreEqual(error.what(), "There is no way from the source to the sink");

}

}

};

}