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»

Вариант 3

Студент гр. 8301 _____ Готовский К.В.

Преподаватель _____ Тутева А.В.

Санкт-Петербург

2020

Цель работы

Реализовать программу принимающую список ребер из файла, представляющий собой граф. Далее следует рассчитать максимальный поток в заданном графе методом Проталкивания предпотока.

Описание реализуемых и вспомогательных классов

Класс Flow, содержит поля: int* excess flow (массив избытков вершин), int** capacity_edge (остаточная сеть), int* h (функция высоты), int vertexCount (количество вершин), int sourceVertex (исток), int destinationVertex (сток).

Класс содержит следующие методы:

- Констуктор – получает на вход файл, в котором список строк, обрабатывает их и в результате выдает список смежности, где над диагональю пропускные способности прямых рёбер, а под диагональю пропускные способности «обратных рёбер».
- Деструктор – вызывает метод clear (на основе обычного удаления двоичного дерева).
- `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

- `void push(int edge, int vertex)` – O(1)
- `void lift(int edge)` – O(|V|)
- `void discharge(int edge)` – O(|V| |E|)
- `int max_flow()` – O($|V|^2$ |E|).

Описание реализованных unit-тестов

Реализованные мною тесты проверяют ситуации с 6 вершинами и с 20. Они проверяют ситуацию с одним ребром из стока в сток, а также когда есть не только ребро из истока в сток, но и другие рёбра. Так же тесты проверяют корректность обработки исключительных ситуаций, например, когда пользователь не ввел одну из позиций, либо ввёл её некорректно, а также забыл ввести исток или сток.

Обоснование выбора используемых структур данных

Я использую MAP для того чтобы индивидуализировать вершины индексами. Данную структуру я использую по причине того, что она позволяет не сохранять повторяющиеся данные и быстрый доступ к ним. List я использую для перебора вершин сети в функции max_flow. В структуре List есть удобный функционал в отличии от обычного массива, нам не нужно хранить размер массива, также мы можем быстро добавлять и удалять элементы, без траты времени на их перезапись в новый массив (в нашем случае push_front добавление в начало работает за O(1)).

Пример работы программы

Пример

```
S A 3
S C 2
S B 2
S D 1
A B 7
B F 5
C B 1
C E 6
D C 2
D E 2
E F 4
E I 1
F I 6
F G 3
G A 4
G H 7
H T 9
I T 7
I H 2
```

```
Max flow for the input graph is: 8
```

```
S A 20
S C 47
S B 13
S D 57
A B 12
B F 23
C B 15
C E 6
D C 13
D E 15
E F 36
E I 56
F I 100
F G 21
G A 42
G H 24
H T 56
I T 23
I H 15
```

```
Max flow for the input graph is: 44
```

```
S A 20
S C 47
A B 13
S D 57
A B 12
C F 23
C D 15
C E 6
D C 13
D E 15
E F 36
E I 56
F I 100
A G 21
G A 42
G H 24
I K 56
I T 23
A H 15
```

```
Max flow for the input graph is: 23
```

```
S A 90
S C 35
A B 73
S D 60
A B 89
C F 96
C D 86
C E 16
D C 23
D E 65
E F 36
E I 44
F I 60
A G 17
G A 84
G H 38
I K 52
I T 39
A H 94
```

```
Max flow for the input graph is: 39
```

Листинг

Flow.h

```
1.  #pragma once
2.  #include <fstream>
3.  #include "List.h"
4.  #include<string>
5.  #include"Map.h"
6.  #include "Algorithm.h"
7.  using namespace std;
8.  class Flow {
```

```
public:
    ~Flow() {
        delete[] excess_flow;
        delete[] height;
        for(int i=0;i<vertexCount;++i)
        delete[] capacity_edge[i];
    }
    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) {//greater than or equal to 5, because this is the minimum possible input(
                if (!((s1[0] >= 'A' && s1[0] <= 'Z') && (s1[1] == ' '))) {
                    throw std::exception(string(("Error entering the first character in the string or miss
                }
                if (!((s1[2] >= 'A' && s1[2] <= 'Z') && (s1[3] == ' '))) {
                    throw std::exception(string(("Error entering the second character in the string or mis
                }
                string cur;
                for (int i = 4; i < s1.size(); ++i) {
                    if (s1[i] >= '0' && s1[i] <= '9')
                        cur += s1[i];
                    else {
                        throw std::exception(string(("Error entering the third character (bandwidth) in th
))).c_str());
                    }
                }
                if (!Map_from_char_to_number->find_is(s1[0])) {//checking the presence of a symbol in the
                    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 std::exception(string(("A data-entry error. Check the correctness of the input in th
            }
            ++str_num;
        }
        if (Map_from_char_to_number->find_is('S'))
            sourceVertex = Map_from_char_to_number->find('S');
        else {
            throw std::exception("Source is missing");
        }

        if (Map_from_char_to_number->find_is('T'))
            destinationVertex = Map_from_char_to_number->find('T');
        else {
            throw std::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 vert1, vert2, cap;
            getline(file, s1);
            vert1 = Map_from_char_to_number->find(s1[0]);
            vert2 = Map_from_char_to_number->find(s1[2]);
            if(vert1==vert2)
                throw std::exception(string("The path from the vertex to itself is impossible in the strin
            capacity_edge[vert1][vert2] = stoi(s1.substr(4));
            ++str_num;
        }
    }
////////////////////////////////////////////////////////////////////////////////////////////////////////////
    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> l;
                int cur;
                int cur_index = 0;
                int old_height;
                for (int i = 0; i < vertexCount; i++)
                    if (i != sourceVertex && i != destinationVertex)
                        l.push_front(i);
                cur = l.at(0);
                while (cur_index < l.get_size())
                {
                    old_height = height[cur];
                    discharge(cur);
                    if (height[cur] != old_height)
                    {
                        l.push_front(cur);
                        l.remove(++cur_index);
                        cur = l.at(0);
                        cur_index = 0;
                    }
                    ++cur_index;
                    if (cur_index < l.get_size())
                        cur = l.at(cur_index);

                }
                return excess_flow[destinationVertex];
            }
            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;
        }

```

```
142.            void lift(int edge)
143.            {
144.                int min = 2 * vertexCount + 1;
145.
146.                for (int i = 0; i < vertexCount; i++)
147.                    if (capacity_edge[edge][i] && (height[i] < min))
148.                        min = height[i];
149.                height[edge] = min + 1;
150.            }
151.
152.            void discharge(int edge)
153.            {
154.                int vertex = 0;
155.                while (excess_flow[edge] > 0)
156.                {
157.                    if (capacity_edge[edge][vertex] && height[edge] == height[vertex] + 1)
158.                    {
159.                        push(edge, vertex);
160.                        vertex = 0;
161.                        continue;
162.                    }
163.                    ++vertex;
164.                    if (vertex == vertexCount)
165.                    {
166.                        lift(edge);
167.                        vertex = 0;
168.                    }
169.                }
170.            }
171.        private:
172.            int* excess_flow;
173.            int** capacity_edge;
174.            int* height;
175.            int vertexCount, sourceVertex, destinationVertex;
176.        };
```

UnitTestFlow_Push_Relabel.cpp

```
1.  #include "pch.h"
2.  #include "CppUnitTest.h"
3.  #include"../Flow.h"
4.  #include <fstream>
5.  using namespace Microsoft::VisualStudio::CppUnitTestFramework;
6.
7.  namespace UnitTestFlowPushRelabel
8.  {
9.      TEST_CLASS(UnitTestFlowPushRelabel)
10.     {
11.     public:
12.
13.         TEST_METHOD(TestMethod_Correct_output_for_6_vertexes)
14.         {
15.             ifstream input("C:\\Users\\ROG\\source\\repos\\Flow_Push_Relabel\\UnitTestFlow_Push_Relabel\\I
16.             Flow flow(input);
17.             Assert::AreEqual(flow.max_flow(), 5);
18.         }
19.         TEST_METHOD(TestMethod_Exception_entering_the_first_character) {
20.             try {
21.                 ifstream input("C:\\Users\\ROG\\source\\repos\\Flow_Push_Relabel\\UnitTestFlow_Push_Relabe
22.                 Flow flow(input);
23.             }
24.             catch (exception & ex) {
25.                 Assert::AreEqual(ex.what(),"Error entering the first character in the string or missing a
26.             }
27.         }
28.         TEST_METHOD(TestMethod_Exception_entering_the_second_character) {
```

```
29.             try {
30.                 ifstream input("C:\\Users\\ROG\\source\\repos\\Flow_Push_Relabel\\UnitTestFlow_Push_Relabe
31.                 Flow flow(input);
32.             }
33.             catch (exception & ex) {
34.                 Assert::AreEqual(ex.what(), "Error entering the second character in the string or missing 
35.             }
36.         }
37.         TEST_METHOD(TestMethod_Exception_entering_the_third_number_flow) {
38.             try {
39.                 ifstream input("C:\\Users\\ROG\\source\\repos\\Flow_Push_Relabel\\UnitTestFlow_Push_Relabe
40.                 Flow flow(input);
41.             }
42.             catch (exception & ex) {
43.                 Assert::AreEqual(ex.what(), "Error entering the third character (bandwidth) in the string 
44.             }
45.         }
46.         TEST_METHOD(TestMethod_Exception_empty_string) {
47.             try {
48.                 ifstream input("C:\\Users\\ROG\\source\\repos\\Flow_Push_Relabel\\UnitTestFlow_Push_Relabe
49.                 Flow flow(input);
50.             }
51.             catch (exception & ex) {
52.                 Assert::AreEqual(ex.what(), "A data-entry error. Check the correctness of the input in the
53.             }
54.         }
55.         TEST_METHOD(TestMethod_Correct_output_for_6_vertexes_and_edge_from_source_to_sink)
56.         {
57.             ifstream input("C:\\Users\\ROG\\source\\repos\\Flow_Push_Relabel\\UnitTestFlow_Push_Relabel\\I
58.             Flow flow(input);
59.             Assert::AreEqual(flow.max_flow(), 25);
60.         }
61.         TEST_METHOD(TestMethod_Correct_output_for_2_vertexes_edges_from_source_to_sink)
62.         {
63.             ifstream input("C:\\Users\\ROG\\source\\repos\\Flow_Push_Relabel\\UnitTestFlow_Push_Relabel\\I
64.             Flow flow(input);
65.             Assert::AreEqual(flow.max_flow(), 20);
66.         }
67.         TEST_METHOD(TestMethod_Exception_there_is_a_path_from_the_vertex_to_itself) {
68.             try {
69.                 ifstream input("C:\\Users\\ROG\\source\\repos\\Flow_Push_Relabel\\UnitTestFlow_Push_Relabe
70.                 Flow flow(input);
71.             }
72.             catch (exception & ex) {
73.                 Assert::AreEqual(ex.what(), "The path from the vertex to itself is impossible in the strin
74.             }
75.         }
76.         TEST_METHOD(TestMethod_Correct_output_for_20_vertexes)
77.         {
78.             ifstream input("C:\\Users\\ROG\\source\\repos\\Flow_Push_Relabel\\UnitTestFlow_Push_Relabel\\I
79.             Flow flow(input);
80.             Assert::AreEqual(flow.max_flow(), 19);
81.         }
82. 
83.     };
84. }
```

Вывод

В данной лабораторной работе я ознакомился с алгоритмом Проталкиванияпред потока и смог применить его в нахождении максимального потока в транспортной сети, а также закрепил свои навыки в объектно-ориентированном программировании.