МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Потоки в сети

Студентка гр. 7383 _____ Рудоман В.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы

Исследовать и реализовывать задачу нахождения максимального потока в сети, применяя алгоритм Форда – Фалкерсона.

Формулировка задачи: найти максимальный поток в сети, а также фактическую величину потока, протекающего через каждое ребро, используя алгоритм Форда – Фалкерсона. Сеть (ориентированный взвешенный граф) представляется в виде триплета из имён вершин и целого неотрицательного числа – пропускная способность (веса).

Вариант 1c: граф представлен в виде списка смежности, поиск пути задаётся через поиск в ширину.

Входные данные: в первой строке указывается количество ориентированных рёбер графа, затем идут значения начальной и конечной вершин. Далее вводят данные о рёбрах графа и их весе, пропускной способности.

Выходные данные: максимальный поток в сети, а также фактическая величина потока, протекающего через каждую дугу, все рёбра отсортированы в лексикографическом порядке.

Реализация задачи

Struct cell – структура массива вершин.

Struct edge – структура ребра.

Struct str_ing – структура строки матрицы.

Class Graf – класс граф.

int add_vertex – метод добавления вершины.

void add_edge – добавление ребра.

int place – поиск индекса строки графа по имени вершины.

int algo – Алгоритм Форда-Фалкерсона.

int how_vertex – возвращает размер матрицы.

Graf& operator-= – отнимает один граф от другого.

void print_graf – вывод матрицы на экран.

int ret – возвращает значение ребра из from в to.

int compare – компаратор для qsort, сортирует лексиграфически по 1-ой, потом по 2-ой строке.

Исследование

Поскольку на каждой итерации поток как минимум увеличивается на один, а поиск пути в графе происходит за O(|E|) операций, то сложность алгоритма составляет O(|F|E|), где F – максимальный поток в сети. Данная оценка требует знать величину максимального потока, но так как он не может превышать сумму пропускных способностей истока и сумму пропускных способностей стока, то можно заменить F на максимальную из этих двух сумм. Тогда O(|M|E|).

Выводы

В ходе лабораторной работы был изучен алгоритм поиска максимального потока в сети, используя алгоритм Форда – Фалкерсона. Был написан код на языке программирования C++, который применял этот метод для поставленной задачи. Сложность реализованного алгоритма составляет O(|M|E|).

ПРИЛОЖЕНИЕ А

Тестовые случаи

Ввод	Вывод
7af ab 7 a c6 bd6 c f9 de3 df4 e c2	12 a b6 a c6 bd6 c f8 de2 df4 e c2
13 a h ab 6 a c6 bd4 be2 cb 2 c e9 df4 dg2 ed 8 eg 7 fh 7 g f 11 gh4	11 a b6 a c5 bd4 be2 c b0 c e5 df4 dg0 e d0 e g7 f h7 gf3 gh4

ПРИЛОЖЕНИЕ Б

Код программы

```
#include <queue>
#include <stdio.h>
#include <iostream>
#include <cmath>
#include <vector>
#include <cstdlib>

using namespace std;


struct cell                //Структура массива вершин
{
char name;
int value;
};

struct edge                //Структура ребра
{
      char from;
      char to;
      int value;
};

struct str_ing             //Структура строки матрицы
{
      char name;       //Имя вершины
      char from;       //Имя предыдущей вершины(нужно в методе algo)
      int how;         //Вес пути до вершины(нужно в методе algo)
      vector <int> vect;    //Вектор весов ребер
};

class Graf                           //Класс графа
{
private:
```

```
	vector<struct str_ing> matrix;			//Вектор строк
public:
	Graf()					//Дэфолтный конструктор
	{
		matrix.resize(0);
	}

	int add_vertex(char name)			//Метод добавления вершины
	{
		if(place(name)!=-1)			//Если нужная вершина есть
			return 1;			//Выходим
		int max=matrix.size();
		int count=0;
		while(count<max)			//Расширяем каждую строку
матрицы
		{
			matrix[count].vect.resize(max+1);
			matrix[count].vect[max]=0;
			count++;
		}
		matrix.resize(max+1);			//Добовляем новую
строку
		matrix[max].name=name;				//И
инициализируем ее
		matrix[max].from='-';
		matrix[max].how=-1;
		count=0;
		matrix[max].vect.resize(max+1);
		max=matrix.size();
		while(count<max)			//Обнуление новой строки
		{
			matrix[max-1].vect[count]=0;
			count++;
		}
		cout << "Добавлена вершина '" << name << "'." << endl;
		return 0;
	}
```

```
	void add_edge(char from, char to, int value)
	//Добавление ребра
	{
		add_vertex(from);
		add_vertex(to);
		matrix[place(from)].vect[place(to)]=value;
		cout << "Добавлено ребро из '" << from << "' в '" << to <<
"' с весом '" << value << "'." << endl;
	}


	int place(char name)			//Поиск индекса строки
графа по имени вершины
	{
		if(matrix.size()==0)		//Если граф пуст
			return -1;
		int count=0;
		while(matrix[count].name!=name && count<matrix.size())
			count++;
		if(count==matrix.size())		//Если нужной вершины
нет в графе
			return -1;
		return count;
	}


	int algo(char from,char to)			//Алгоритм Форда-
Фалкерсона
	{
		char point;
		char str[matrix.size()+1];
		struct cell mass[matrix.size()];
		int count;
		int count_from;
		int count_cell=0;
		int key=1;
		char max;
		int min;
		int colcul=0;
```

```
          while(key!=6)                                    //Пока не
найдем поток
          {
               switch(key)                                 //Выбор
действия
               {
                    case 1:
     //Устанавливаем текущую вершину в исток
                    {
                         cout << endl << "Действие первое." << endl;
                         matrix[place(from)].from='!';
                         point=from;
                         cout << "Текущая вершина:" << point << endl;
                         key=2;                              //Переходим
во 2-е действие
                         break;
                    }
                    case 2:                                  //Создаем
массив путей из текущей вершины
                    {
                         cout << endl << "Действие второе." << endl;
                         count_cell=0;
                         count=0;
                         while(count<matrix.size())      //Обнуляем
старый массив
                         {
                              mass[count].name='-';
                              mass[count].value=0;
                              count++;
                         }
                         count_from=place(point);
                         count=0;
                         while(count<matrix.size())      //Заполняем
                         {
                              if(matrix[count_from].vect[count]>0 &&
matrix[count].from=='-')
                              {

     mass[count_cell].name=matrix[count].name;
```

```
     mass[count_cell].value=matrix[count_from].vect[count];
                                        count_cell++;
                                  }
                                  count++;
                            }
                            count=0;
                            while(count<count_cell)
                            {
                                  str[count]=mass[count].name;
                                  count++;
                            }
                            str[count]='\0';
                            cout << "Из вершины '" << point << "' мы
можем перейти в вершины <" << str << ">." << endl;
                            if(count_cell!=0)               //Если
массив не пуст переходим в 3-е действие
                                  key=3;
                            else                        //Если пуст то в
4-е
                                  key=4;
                            break;
                      }
                      case 3:                                 //Переходим
по ребру в меньшую по названию вершину
                      {
                            cout << endl << "Действие третье." << endl;
                            count=0;
                            max=str[count];
                            while(count<count_cell)
                            {
                                  if(max>str[count])
                                        max=str[count];
                                  count++;
                            }
                            cout << "Среди вершин <" << str << ">
вершина с меньшим именем: '" << max << "'." << endl;
                            count_cell=place(max);
```

```
                        count=0;
                        while(str[count]!=max)
                              count++;
                        cout << "Ставим метку на новой вершине." <<
endl;
                        matrix[count_cell].from=point;
      //Метим вершину
                        matrix[count_cell].how=mass[count].value;
                        if(matrix[count_cell].name==to)      //Если
переходим в конечную вершину, переходим в 5-е действие
                              key=5;
                        else                        //Если нет, то
меняем текущую вершину и переходим в 2-е действие
                        {
                              point=matrix[count_cell].name;
                              cout << "Меняем текущую вершину на '"
<< point << "'." << endl;
                              key=2;
                        }
                        break;
                  }
                  case 4:
      //Откатываемся на предыдущую вершину
                  {
                        cout << endl << "Действие четвертое." <<
endl;
                        if(point==from)                //Если
сейчас мы в истоке, переходим в 6-е действие
                        {
                              cout << "Завершаем работу." << endl;
                              key=6;
                        }
                        else                        //Если нет, то
переходим в прошлую вершинуи заново ищем новые пути
                        {
                              point=matrix[place(point)].from;
                              cout << "Откатываемся на вершину '" <<
point << "'." << endl;
                              key=2;
```

```
                        }
                        break;
                  }
                  case 5:                              //Считаем
минимальный поток по найденому пути и изменяем матрицу
                  {
                        cout << endl << "Действие пятое." << endl;
                        point=to;
                        count=place(to);
                        min=matrix[count].how;
                        point=matrix[count].from;
                        cout << "Подсчитаем минимальный поток в
пути:" << to;
                        while(point!=from)             //Проходим
по пути ища мин. поток
                        {
                              count=place(point);
                              cout << point;
                              if(min>matrix[count].how)
                                    min=matrix[count].how;
                              point=matrix[count].from;
                        }



                        cout << point << "(Читать с конца к
началу)." << endl;
                        cout << "Минимальный поток: " << min << "."
<< endl;
                        point=to;
                        cout << "Заменим значения матрицы с:" <<
endl;
                        print_graf();
                        while(point!=from)             //Проходим
по пути меняя значения ребер
                        {
                              count=place(point);
                              count_from=place(matrix[count].from);
                              matrix[count].vect[count_from]+=min;
```

```
                                    matrix[count_from].vect[count]-=min;
                                    point=matrix[count].from;
                              }
                              colcul+=min;
      //Увеличиваем знасение потока в графе
                              count=0;
                              while(count<matrix.size())      //Обнуляем
метки
                              {
                                    matrix[count].from='-';
                                    matrix[count].how=-1;
                                    count++;
                              }
                              cout  << endl << "На: " << endl;
                              print_graf();
                              cout << endl;
                              key=1;
                              break;
                        }
                  }
            }
            return colcul;
      }

      int how_vertex()                //Возвращает размер матрицы
      {
            return matrix.size();
      }

      Graf& operator-=(Graf& other)             //Отнимает один граф
от другого
      {
            int x=0,y=0;
            if(matrix.size()==other.how_vertex()) //Проходи по всем
полям матрицы и производим разность
            {
                  while(x<matrix.size())
```

```
                {
                        while(y<matrix.size())
                        {
                                matrix[x].vect[y]-=other.matrix[x].vect[y];
                                y++;
                        }
                        x++;
                        y=0;
                }
        }
    }

    void print_graf()                                   //Вывод матрицы
на экран
    {
        int x=0,y=0;
        cout <<"%" << " ";
        while(x<matrix.size())                          //Выводим имена
вершин
        {
                cout  << matrix[x].name << " ";
                x++;
        }
        cout << endl;
        x=0;
        while(x<matrix.size())                          //Выводим
значения
        {
                cout << matrix[x].name << " ";
                while(y<matrix.size())
                {
                        cout << matrix[x].vect[y] << " ";
                        y++;
                }
                cout << " from:" << matrix[x].from << " how:" <<
matrix[x].how << endl;
                y=0;
```

```
                x++;
            }
        }

        int ret(char from,char to)                  //Возвращает значение
ребра из from в to
        {
            return matrix[place(from)].vect[place(to)];
        }
};

int compare(const void* val1,const void* val2)
        //Компаратор для qsort, сортирует лексиграфически по 1-ой , потом
по 2-ой строке
{
        if((*(struct edge*)val1).from==(*(struct edge*)val2).from)
            return (*(struct edge*)val1).to-(*(struct edge*)val2).to;
        return (*(struct edge*)val1).from-(*(struct edge*)val2).from;
}

int main()
{
        Graf graf,graf_2;
        int k,value,count=0;
        char from,to;
        cin >> k >> from >> to;
        struct edge edges[k];
        char a,b;
        while(count<k)                          //Считываем данные
        {
        cin >> edges[count].from >> edges[count].to >>
edges[count].value;
        graf.add_edge(edges[count].from,edges[count].to,edges[count].valu
e);
        graf_2.add_edge(edges[count].from,edges[count].to,edges[count].va
lue);
        count++;
        }
```

```
	cout << endl << "Первоночальный граф:" << endl;
	graf.print_graf();
	cout << endl;
	cout << "Начало работы алгоритма." << endl;
	int max=graf_2.algo(from,to);		//Запуск алгоритма
	cout << "Максимальный поток = " << max << endl;
	graf-=graf_2;
	count=0;
	while(count<k)				//Изменение значения ребер
	{

	edges[count].value=graf.ret(edges[count].from,edges[count].to);
		count++;
	}
	qsort(edges,k,sizeof(struct edge),compare);	//Сортировка
ребер
	cout  << "Максимальный поток:" << max << endl;
	//Вывод потока
	count=0;
	cout << "Значения в ребрах:" << endl;
	cout << max << endl;
	while(count < k)			//Вывод ребер
	{
		if(edges[count].value<0)
			edges[count].value=0;
		cout << edges[count].from << " " << edges[count].to << " "
<< edges[count].value << endl;
		count++;
	}

}
```