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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Построение и анализ алгоритмов»
Тема: Максимальный поток

Студент гр. 9382 _____ Сорокумов С.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2021

Цель работы.

Изучить алгоритмы поиска максимального потока в сети. Изучить алгоритмы нахождения пути в графе. Написать программу для нахождения потока в сети используя указанный алгоритм. Показать асимптотику алгоритма по времени и по памяти. Провести тестирование, предоставить работающую программу и отчёт.

Задание.

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 - пропускной способности (веса).

Входные данные:

N - количество ориентированных рёбер графа

v_0 - исток

v_n - сток

$v_i\ v_j\ \omega_{ij}$ - ребро графа

$v_i\ v_j\ \omega_{ij}$ - ребро графа

...

Выходные данные:

P_{max} - величина максимального потока

$v_i\ v_j\ \omega_{ij}$ - ребро графа с фактической величиной протекающего потока

$v_i\ v_j\ \omega_{ij}$ - ребро графа с фактической величиной протекающего потока

...

В ответе выходные рёбра отсортируйте в лексикографическом порядке по первой вершине, потом по второй (в ответе должны присутствовать все

указанные входные рёбра, даже если поток в них равен 0).

Sample Input:

7

a

f

a b 7

a c 6

b d 6

c f 9

d e 3

d f 4

e c 2

Sample Output:

12

a b 6

a c 6

b d 6

c f 8

d e 2

d f 4

e c 2

Вариант 3.

Поиск в глубину. Рекурсивная реализация.

Описание алгоритма.

Подающиеся на вход программе ребра программа обрабатывает так, что помимо реального ребра, она создает еще и мнимые(обратные).

Затем она сразу приступает к поиску потока. Функция поиска

int find_flow(size_t start, size_t finish, int flow, int count_rec)

принимает на вход следующие элементы: *size_t start* – индекс стартовой вершины, *size_t finish* – индекс вершины финиша, *int flow* – поток, *int count_rec* – счетчик рекурсии (нужен для корректного вывода действий функции).

Действия функции:

1. Проверка на достижение вершины финиша. Если он достигнут, то возвращается поток *flow*.
2. Текущая вершина помечается посещенной.
3. Далее происходит перебор всех вершин инцидентных с текущей.
4. И если инцидентная вершина не была посещена. И по ней можно пустить ненулевой поток, то: 5-7
5. В *min_result* записываем значение функции *find_flow*, но уже со следующими параметрами: в качестве стартовой вершины передается инцидентная, в качестве финишной – финишная, а в качестве потока минимум из потока переданного текущей функции и потока, который можно пусть по выбранному ребру, и счетчик рекурсии + 1.
6. Если *min_result*> 0 (можно пустить ненулевой поток по выбранному ребру), то вычитаем этот поток(*min_result*) из остаточной пропускной способности выбранного ребра. А к обратному ребру прибавляем *mit_result* к остаточной пропускной способности.
7. Возвращаем *min_result*.
8. В общем цикле поиска всех потоков происходит суммирование найденных потоков и снятие меток посещенности для всех вершин пока находится ненулевые потоки.

Код программы предоставлен в приложении А.

Асимптотика алгоритма.

Добавляя поток увеличивающего пути к уже имеющемуся потоку, максимальный поток будет получен, когда нельзя будет найти

увеличивающий путь. Тем не менее, если величина пропускной способности – иррациональное число, то алгоритм может работать бесконечно. В целых числах таких проблем не возникает и время работы ограничено O(|*E*|*f*), где E – число рёбер в графе, *f* – максимальный поток в графе, так как каждый увеличивающий путь может быть найден за O(E) и увеличивает поток как минимум на 1.

Сложность по памяти составляет O(|V|+|E|).

Описание структур данных.

Название структуры	**Модификатор доступа**	**Объект**	**Параметры**	**Описание**	**Возвращаемое значение**
struct Connect	public	type value;	-	Именование вершины	-
	public	double weight;	-	Вес пути до этой вершины	-
	public	bool real;	-	Реальное ребро при real = 1, при real = 0 – ребро мнимое.	-
struct Edge	public	char from;	-	Вершина начала ребра.	-
	public	char to;	-	Вершина конца ребра.	-
	public	int weight;	-	Вес ребра.	-
class Vertex	public	char key;	-	Наименование вершины.	-
	public	bool visited = false;	-	Метка посещенности, при false – не посещена.	-
	public	Connect *connects;	-	Массив связей с вершиной key.	-
	public	size_t size_v;	-	Хранит размер массива connects.	-
	public	void resize_con()	-	Увеличивает размер connects по надобности.	-

	public	size_t find_con(const char litter)	const char litter – символ для поиска.	Производит поиск среди элементов массива connects	Возвращает индекс элемента в connects.
class Directed_Graph	private	Vertex *list	-	Хранит список вершин.	-
	private	size_t size_l = 0	-	Хранит размер list	-
	private	void print()	-	Распечатывает граф	-
	private	void push(const type key, const type value, const float weight)	const type key – начало ребра; const type value – конец ребра; const float weight – вес ребра;	Добавляет ребро из key в value весом weight.	-
	private	size_t find_or_add(const type litter)	const char litter – символ для поиска.	Находит элемент в противном случае добавляет его	Возвращает индекс элемента в list.
	private	void resize_list()	-	Изменяет размер list	-
	private	void unvisit()	-	Снимает метки посещенности вершин.	-
	private	void print_flow()	-	Распечатывает поток со всеми ребрами.	-
	private	int find_flow(size_t start, size_t finish, int flow, count_rec)	size_t start – индекс начальной вершины; size_t finish – индекс	Находит поток в графе.	Возвращает полученный поток.

			конечной вершины; int flow – текущий поток; int count_rec – счетчик рекурсии;		

Тестирование.

Номер теста	Тест:	Результат:
1	19 a i a b 2 a c 3 a d 5 b c 4 b d 1 c j 5 c e 4 c i 2 c f 3 d e 12 d g 9 d f 4 e i 34 f h 6 f i 5	10 a b 2 a c 3 a d 5 b c 2 b d 0 c e 0 c f 0 c i 0 c j 5 d e 4 d f 0 d g 1 e i 9 f h 1 f i 0 g f 1 g h 0

	g f 12 g h 8 h i 7 j e 6	h i 1 j e 5
2	11 a g a b 3 a d 3 b c 4 c a 3 c d 1 c e 2 d e 2 d f 6 e b 1 e g 1 f g 9	5 a b 2 a d 3 b c 2 c a 0 c d 1 c e 1 d e 0 d f 4 e b 0 e g 1 f g 4
	30 a n a b 6 a c 6 a d 8 a e 9 b c 3 b f 4 c d 4 c g 4	26 a b 6 a c 6 a d 8 a e 6 b c 2 b f 4 c d 4 c g 4 d e 0 d h 5

	d e 3 d h 5 d i 10 e i 6 f c 9 f j 5 g d 10 g k 5 h g 8 h l 5 h m 12 i h 7 i m 7 j g 10 j n 6 k h 12 k j 8 k n 9 l k 8 l n 7 m l 7 m n 6	d i 7 e i 6 f c 0 f j 4 g d 0 g k 5 h g 0 h l 5 h m 7 i h 7 i m 6 j g 1 j n 6 k h 0 k j 3 k n 9 l k 7 l n 5 m l 7 m n 6
4	7 a f a b 7 a c 6 b d 6 c f 9	12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4

	d e 3 d f 4 e c 2	e c 2

Выводы.

В ходе выполнения лабораторной работы был изучен алгоритм нахождения максимального потока в графе Форда-Фалкерсона. Для хранения графа использовались списки смежности.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ MAIN.CPP

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <math.h>

int INF = (int) 1e9;

using namespace std;

struct Connect
                        //Список связей
{
      char value;
      int weight;
      bool real;
};
struct Edge
                              //Ребро
{
      char from;
      char to;
      int weight;
};
class Vertex
                        //Вершина
{
public:
      char key;
      bool visited = false;
      Connect *connects;
      size_t size_v = 0;

      void resize_con()
                        //Функция изменения размера списка инцидентных
ребер
      {
            if(size_v%5 != 0 && size_v) return;
            Connect *tmp = new Connect[size_v + 5];
            for(int i = 0; i < size_v; i++)
                  tmp[i] = connects[i];
            if(size_v) delete [] connects;
            connects = tmp;
      };
      size_t find_con(const char litter)
                  //Функция нахождения вершины
      {
```

```
            for(int i = 0; i < size_v; i++)
                  if(connects[i].value == litter)
                        return i;
            return 0;
      };
};

int comp(const Edge &val1, const Edge &val2)
            //Функция сравнения ребер
{
      if(val1.from == val2.from)
            return val1.to < val2.to;
      return val1.from < val2.from;
};

class Directed_Graph{
private:
      Vertex *list;
      size_t size_l = 0;
public:
      ~Directed_Graph()
      {
            for(size_t i = 0; i < size_l; i++)
                  delete [] list[i].connects;
            delete [] list;
      }
      Directed_Graph()
      : list(new Vertex[3])
      {};

      void print()
                        //Функция вывода графа
      {
            for(int i = 0; i < size_l; i++)
            {
                  cout << list[i].key << " : ";
                  for(int j = 0; j < list[i].size_v; j++)
                        if(list[i].connects[j].real)
                              cout << list[i].connects[j].value << "(" <<
list[i].connects[j].weight << ") ";
                  cout << endl;
            }
      };

      void push(const char key, const char value, const int weight, bool
real)//Функция добавляет новое ребро
      {
            find_or_add(value);
            Vertex &v = list[find_or_add(key)];
            v.resize_con();
```

```
            v.connects[v.size_v].value = value;
            v.connects[v.size_v].weight = weight;
            v.connects[v.size_v].real = real;
            v.size_v++;
      };
      size_t find_or_add(const char litter)
                  //Функция либо находит, либо добавляет новую вершину
      {
            for(int i = 0; i < size_l; i++)
                  if(list[i].key == litter)
                        return i;
            resize_list();
            list[size_l].key = litter;
            return size_l++;
      };
      void resize_list()
                        //Функция изменения размера списка вершин с их
ребрами
      {
            if(size_l%3 != 0 || !size_l) return;

            Vertex *tmp = new Vertex[size_l + 3];

            for(int i = 0; i < size_l; i++)
            {
                  tmp[i] = list[i];
                  tmp[i].connects = new Connect[list[i].size_v];
                  tmp[i].connects = list[i].connects;
            }
            list = tmp;
            for(int i = 0; i < size_l; i++)
                  list[i].connects = tmp[i].connects;
      };

      void unvisit()
                        //Функция обращает все ребра в непройденные
      {
            for(int i = 0; i < size_l; i++)
                        list[i].visited = false;
      };

      int find_flow(size_t start, size_t finish, int flow, int count_rec)
                        //Функция нахождения потока
      {
            cout << "Проходим в вершину (" << list[start].key << ")." <<
endl;
            if(start == finish)
            {
                  for(int k = 0; k < count_rec; k++)
                        cout << "\t";
```

```
                cout << "Дошли до конца." << endl;
                return flow;
                        //возвращяем полученный минимум на пути
            }
            list[start].visited = true;
                        //помечаем ребро посещенным
            for(int k = 0; k < count_rec; k++)
                cout << "\t";
            cout << "Перебираем все инцидентные вершины для (" <<
list[start].key <<"):" << endl;
            for(size_t edge = 0; edge < list[start].size_v; edge++)
            //Перебираем все инцидентные ребра вершине start
            {
                size_t                          to                          =
find_or_add(list[start].connects[edge].value);  //Находим       индекс
инцидентной вершины в списке
                for(int k = 0; k < count_rec; k++)
                    cout << "\t";
                cout << edge + 1 << ") вершина (" << list[to].key << ")
с     остаточной     пропускной     способностью     -     "     <<
list[start].connects[edge].weight << endl;
                if(!list[to].visited                                        &&
list[start].connects[edge].weight  >  0)//Если  вершина  не  посещена  и
остаточный вес ребра не нулевой, проходим
                {
                    int min_result = find_flow(to, finish, min(flow,
list[start].connects[edge].weight),   count_rec   +   1);//Полученный
максимальный поток через минимальное ребро
                    for(int k = 0; k < count_rec; k++)
                        cout << "\t";
                    cout << "Полученный поток - " << min_result <<
endl;
                    if(min_result > 0)
                    //И если поток не нулевой, проходим
                    {
                        for(int k = 0; k < count_rec; k++)
                            cout << "\t";
                        cout  <<  "Вычитаем  из  текущей  пропускной
способности       реального       ребра       поток:       "       <<
list[start].connects[edge].weight
                        <<  "  -  "  <<  min_result  <<  "  =  "  <<
list[start].connects[edge].weight - min_result << endl;
                        list[start].connects[edge].weight        -=
min_result;     //Вычитаем поток из пропускной способности для реального
ребра
                        //Обратное
                        size_t                  con                 =
list[to].find_con(list[start].key);  //Находим индекс обратного ребра
                        for(int k = 0; k < count_rec; k++)
                            cout << "\t";
```

```
                              cout << "Добавляем к мощности мнимого ребра
поток: " << list[to].connects[con].weight
                              << " + " << min_result << " = " <<
list[to].connects[con].weight + min_result << endl;
                              list[to].connects[con].weight += min_result;
          //И к обратному прибавляем поток
                              return min_result;
                        //Возвращаем результат
                        }
                  }
                  else
                  {
                        for(int k = 0; k < count_rec; k++)
                        cout << "\t";
                        if(list[to].visited)
                              cout << "Вершина (" << list[to].key <<") уже
посещена." << endl;
                        else
                              cout     <<     "Недостаточная     пропускная
способность." << endl;
                  }
            }
            return 0;
                        //если не нашли поток из этой вершины вернем 0
      }
      void print_flow()
                        //Функция печати ребер с их потоком
      {
            vector <Edge> vec;
                        //Контейнер для ребер с потоком
            for(int i = 0; i < size_l; i++)
                  for(int j = 0; j < list[i].size_v; j++)
                        if(!list[i].connects[j].real)
                  //Добавляем, если ребро мниное
                              vec.push_back({list[i].connects[j].value,
list[i].key,    abs(list[i].connects[j].weight    <    0    ?    0    :
list[i].connects[j].weight) });
            sort(vec.begin(), vec.end(), comp);
                  //Сортируем содержимое контейнера с помощью comp
            for_each(vec.begin(), vec.end(), [](const Edge& obj){cout <<
obj.from << " " << obj.to << " " << obj.weight << endl;});//Выводим
            if(!vec.empty()) vec.clear();
                                          //Очистка контейнера
      };
};

int main(){
    setlocale(LC_ALL, "rus");

     int N;
```

```
	char start, finish;
	cin >> N >> start >> finish;
		// считываем количество ребер, начальную и конечную
вершины

	Directed_Graph graf;
	char from, to;
	int weight;
	for(size_t i = 0; i < N; i++)
	{
		cin >> from >> to >> weight;
		graf.push(from, to, weight, true);
			//Добавляем реальное ребро
		graf.push(to, from, 0, false);
			//Добавляем мнимое ребро
	}
	cout << "Представление графа в виде списков:" << endl;
	graf.print();
				//Вывод графа
	cout << endl;

	Нахождение максимального потока
	cout << "Нахождение максимального потока." << endl;
	int max_flow = 0;
	int iter_res;
	int i = 1;
	cout << "____Круг #" << i++ << endl;
	while((iter_res	=	graf.find_flow(graf.find_or_add(start),
graf.find_or_add(finish), INF, 0)) > 0) //Пока есть путь в графе
	{
		cout << "____Круг #" << i++ << endl;
		graf.unvisit();
				//Пометим ребра непройденными
		max_flow += iter_res;
			//К итоговому потоку добавим найденный
	}
	cout << max_flow << endl;
			//Выводим максимальный поток

	graf.print_flow();
				//Выводим ребра с их потоками
	return 0;
}
```