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: Потоки в сети

Студент гр. 8304 Алтухов А.Д.

Преподаватель Размочаева Н. В.

Санкт-Петербург

2020

Цель работы.

Построить алгоритм Форда-Фалкерсона для нахождения максимального потока в сети, определить его сложность.

Вариант 6.

Поиск по правилу: каждый раз выполняется переход по дуге, соединяющей вершины, имена которых в алфавите ближе всего друг к другу. Если таких дуг несколько, то выбрать ту, имя конца которой в алфавите ближайшее к началу алфавита.

Основные теоретические положения.

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 w_{ij} - ребро графа

v_i v_j w_{ij} - ребро графа

…

Выходные данные:

P_{max} - величина максимального потока

v_i v_j w_{ij} - ребро графа с фактической величиной протекающего потока

v_i v_j w_{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).

Описание алгоритма.

Изначальный поток равен нулю. На каждой итерации алгоритма ищется путь из истока в сток по ребрам с немаксимальным текущим потоком. Поиск происходит соответственно варианту. Далее находится ребро с минимальной оставшейся пропускной способностью, к потокам задействованных в пути ребер прибавляется эта минимальная величина потока. Алгоритм заканчивает свою работу, если больше нет возможных путей.

Величина максимального потока равна сумме потоков ребер, инцидентных истоку.

На каждом шаге алгоритм добавляет поток увеличивающего пути к уже имеющемуся потоку. Следовательно, на каждом шаге алгоритм увеличивает поток по крайней мере на единицу, следовательно, он сойдётся не более чем за $O(f)$ шагов, где f — максимальный поток в графе. Можно выполнить каждый шаг за время $O(E)$, где E — число рёбер в графе, тогда общее время работы алгоритма ограничено $O(Ef)$.

Требуемая память: $O(V^2)$, так как для хранения связей используется матрица смежности.

Описание основных структур данных и функций.

class Graph — класс, представляющий собой граф и методы работы с ним.

int search(int vertex, int min, int depth) — поиск пути, соответствующий варианту.

int maxFlow() — поиск максимального потока.

Тестирование.

Таблица 1 – Результаты тестирования.

Ввод	**Вывод**
7 a f a b 7 a c 6 b d 6 c f 9 d e 3 d f 4 e c 2	12 a b 6 a c 6 b d 6 c f 8 d e 2 d f 4 e c 2
16 a e a b 20 b a 20 a d 10 d a 10 a c 30 c a 30 b c 40 c b 40 c d 10 d c 10 c e 20 e c 20	60 a b 20 a c 30 a d 10 b a 0 b c 0 b e 30 c a 0 c b 10 c d 0 c e 20 d a 0 d c 0 d e 10 e b 0

b e 30 e b 30 d e 10 e d 10	e c 0 e d 0

Вывод.

В ходе работы был построен алгоритм Форда-Фалкерсона с поиском пути, соответствующим варианту.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

```
#include <iostream>
#include <map>
#include <vector>
#include <algorithm>
#include <stack>
#include <utility>
#include <queue>
#include <set>
#include <string>
#include <locale>
#include <fstream>

std::ostream* out;
std::istream* in;

std::string operator*(std::string s, size_t count){
	std::string ret;
	for (size_t i = 0; i < count; ++i)
	{
		ret = ret + s;
	}
	return ret;
}

bool comp(std::pair<int, int> v1, std::pair<int, int> v2) {

	if ((abs(v1.first - v1.second)) == (abs(v2.first - v2.second))) {
		return v1.first < v2.first;
	}
	return abs(v1.first - v1.second) < abs(v2.first - v2.second);
}

class Graph {

	int source;
	int drain;

	std::vector<std::vector<std::pair<int, int>>> matrix; //first - capacity, second - current flow

	std::map<int, std::vector<std::pair<int, int>>> children; // в паре first - вершина, second - родитель
	std::vector<bool> visited;


public:
```

```
	Graph(int source, int drain) :source(source - 97), drain(drain - 97) {

		for (int i = 0; i < std::max(source - 97 + 1, drain - 97 + 1);
i++) {

			visited.push_back(false);
			matrix.push_back(std::vector<std::pair<int, int>>());
			//lastDirection.push_back(std::vector <int>());
			for (int j = 0; j < std::max(source - 97 + 1, drain - 97 +
1); j++) {
				matrix[i].push_back({-1,0});
				//lastDirection[i].push_back(0);
			}
		}

	}

	void expandMatrix(int maxSize) {

		for (int i = 0; i < matrix.size(); i++) {
			for (int j = matrix.size(); j < maxSize; j++) {
				matrix[i].push_back({ -1, 0 });
				//lastDirection[i].push_back(0);
			}
		}

		for (int i = matrix.size(); i < maxSize; i++) {

			visited.push_back(false);
			matrix.push_back(std::vector<std::pair<int, int>>());
			for (int j = 0; j < maxSize; j++) {
				matrix[i].push_back({ -1, 0 });
			}
		}
	}

	void setNode(int from, int whereto, int weight) {
		if (std::max(from - 97, whereto - 97) >= matrix.size()) {
			expandMatrix(std::max(from - 97 + 1, whereto - 97 + 1));
		}
		matrix[from - 97][whereto - 97] = { weight,0 };// a = 97
		children[from - 97].push_back({ whereto - 97, from-97 });
	}

	void printMatrix() {

		for (int i = 0; i < matrix.size(); i++) {
			for (int j = 0; j < matrix[i].size(); j++) {
				std::cout << (char)(i + 97) << " " << (char)(j + 97)
<< " " << matrix[i][j].first << "\n"; // a = 97
			}
```

```
            }
      }
      void printRealFlows() {
            for (int i = 0; i < matrix.size(); i++) {
                  for (int j = 0; j < matrix[i].size(); j++) {
                        if (matrix[i][j].first > -1) {
                              if (matrix[j][i].first > -1 &&
matrix[i][j].second > 0 && matrix[j][i].second > 0) {
                                    int min = std::min(matrix[i][j].second,
matrix[j][i].second);
                                    matrix[i][j].second -= min;
                                    matrix[j][i].second -= min;
                              }
                              if (matrix[i][j].second < 0)
                                    matrix[i][j].second -= matrix[i][j].second;
                              if (matrix[j][i].second < 0)
                                    matrix[j][i].second -= matrix[j][i].second;

                              *out << (char)(i + 97) << " " << (char)(j + 97)
<< " " << matrix[i][j].second << "\n"; // a = 97
                        }

                  }
            }
      }



      int search(int vertex, int min, int depth) {

            std::string space = " ";
            space = space * depth;
            *out << space << "Текущая вершина: " << (char)(vertex+97) <<
"\n";
            if (vertex == drain)
                  return min;

            std::sort(children[vertex].begin(), children[vertex].end(),
comp);
            visited[vertex] = true;

            for (auto& next : children[vertex]) {

                  if ((!visited[next.first]) &&
(matrix[vertex][next.first].second < matrix[vertex][next.first].first))
{//если не посещена и осталась вместимость

                        int delta = search(next.first, std::min(min,
matrix[vertex][next.first].first - matrix[vertex][next.first].second),
depth + 1);
                        if (delta > 0) {
```

```
                              *out << space << "Найденный путь: "<< delta <<
"\n";
                              matrix[vertex][next.first].second += delta;
                              if (matrix[next.first][vertex].first > -1 ) {
                                    matrix[next.first][vertex].second -= delta;
                              }
                              return delta;

                        }
                  }

            }

            return 0;
      }



      int maxFlow() {
            int flow = 0;
            int curFlow = 0;

            //for (int i = 0; i < matrix.size(); i++) {
            //    std::sort(children[i].begin(), children[i].end(), [&,
this](std::pair<int, int> v1, std::pair<int, int> v2) {return
matrix[v1.second][v1.first] < matrix[v2.second][v2.first]; });
            //}
            while ((curFlow = search(source, 999999, 0)) != 0) {
                  flow += curFlow;
                  curFlow = 0;
                  for (int i = 0; i < matrix.size(); i++) {
                        visited[i] = false;
                  }
            }

            return flow;
      }
};

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


      char start = '\0';
      char end = '\0';

      int edges = 0;

      int inputMode, outputMode;
```

```
	std::cout << "Ввод из... (0 - из консоли, 1 - из файла): ";
	std::cin >> inputMode;
	std::cout << "Вывод из... (0 - из консоли, 1 - из файла): ";
	std::cin >> outputMode;


	std::ifstream inFile("input.txt");
	std::ofstream outFile("output.txt");
	in = inputMode == 0 ? &std::cin : &inFile;
	out = outputMode == 0 ? &std::cout : &outFile;

	*in >> edges;
	*in >> start >> end;

	char from = '\0';
	char whereto = '\0';
	int weight = 0;

	Graph graph(start, end);
	while (edges) {
		*in >> from >> whereto >> weight;
		graph.setNode(from, whereto, weight);
		edges--;
	}
	//graph.printMatrix();
	int flow = graph.maxFlow();
	*out << flow << "\n";
	graph.printRealFlows();

	inFile.close();
	outFile.close();

	return 0;
}
```