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 транспортной сети

Студентка гр. 7383 _____ Иолшина В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Содержание

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 5м: граф представлен в виде матрицы смежности, поиск пути по вершинам более близким к началу алфавита. 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

В данной работе используются главная функция main() и дополнительные функции.

Была реализована структура, описывающая ребро графа:

```
struct way
{
  int max_b = 0;
  int result_b = 0;
  bool flag = false;
};
```

И реализован следующий класс, описывающий граф:

```
class Graph
{
 char start;
 char finish;
 int max_flow;
 vector<char> find_path()
 vector<char> vertex;
 way** edges;
 int get_index(char name)
 int find_flow()
 void print_answer()
}
```

- max_b хранит значение максимальной пропускной способности
- result_b хранит значение фактической пропускной способности
- start название вершины, являющийся истоком
- finish название вершины, являющийся стоком
- max_flow максимальный поток в ребре
- vector<char> find_path() находит возможный путь, переходя каждый раз в вершину, которая находится ближе к началу алфавита
- vector<char> vertex хранит вершины графа
- way** edges матрица смежности, хранящая ребра графа
- int get_index(char name) находит индекс вершины по её названию, а также записывает вершины в массив vertex
- int find_flow() используя алгоритм Форда-Фалкерсона, находит максимальный поток в сети и фактическую пропускную

способность

- void print_answer() выводит на экран вершины отсортированные в лексикографическом порядке

Тестирование

Программа собрана в операционной системе Ubuntu 16.04.2 LTS", с использованием компилятора g++ (Ubuntu 5.4.0-6ubuntu1~16.04.5). В других ОС и компиляторах тестирование не проводилось.

Тестовые случаи представлены в Приложении А.

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 написания лабораторной работы был изучен алгоритм поиска максимального потока в сети - алгоритм Форда – Фалкерсона. Сложность реализованного алгоритма составляет O(|M||E|).

ПРИЛОЖЕНИЕ А

ТЕСТОВЫЕ СЛУЧАИ

Ввод	Вывод	Верно?
4 a d a b 1 c b 1 a c 2 c d 3	2 a b 0 a c 2 c b 0 c d 2	Да
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	Да

ПРИЛОЖЕНИЕ Б

Исходный код программы

```
#include <vector>
#include <stack>
#include <iostream>
#include <algorithm>
#define N 20

using namespace std;

struct way
{
  int max_b = 0;
  int result_b = 0;
  bool flag = false;
};

class Graph
{
  char start;
  char finish;
  int max_flow;

  vector<char> find_path()
  {
    vector<char> tmp_path;
    vector<char> vis = vertex; //
    stack<int> stack;
    stack.push(0);

    do
    {
      vector<char> arr;  //массив вершин, в которые можно пойти из данной
      int size = arr.size();
```

```
    int cur = stack.top();
    tmp_path.push_back(vertex[cur]);
    if(cur == 1) return tmp_path;
    for(int i = 1; i < vertex.size(); i++)
    {
      if(vis[i]!=0 && (edges[cur][i].max_b))
      {
         arr.push_back(vertex[i]);
      }
    }
    if(arr.size() == size)
    {
      int i = stack.top();
      stack.pop();
      tmp_path.pop_back();
      while(!stack.empty() && !tmp_path.empty() && (edges[stack.top()][i].max_b))
      {
        if(stack.top() == 1)
        {
          if(tmp_path.back() != vertex[1])
            tmp_path.push_back(vertex[1]);
          return tmp_path;
        }
        tmp_path.pop_back();
        i = stack.top();
        stack.pop();
      }
    }
    else
    {
      sort(arr.begin(), arr.end());
      for(int i = arr.size() - 1; i >= 0; i--)
         stack.push(find(vertex.begin(), vertex.end(), arr[i]) - vertex.begin());
    }
  }  while (!stack.empty());
  return vector<char>();
}
```

```
public:
  vector<char> vertex;
  way** edges;

  Graph(char start, char finish)
  {
    max_flow = 0;
    edges = new way*[N];
    for (int i = 0; i < N; i++)
      edges[i] = new way[N];
  }

  ~Graph()
  {
  for (int i = 0; i < N; i++)
    delete[] edges[i];
  delete[] edges;
  }

  int get_index(char name)
  {
    for (int i = 0; i<vertex.size(); i++)
    {
      if (vertex[i] == name)
        return i;
    }
    vertex.push_back(name);
  return vertex.size()-1;
    }

  int find_flow()
  {
    vector<char> path = find_path();
    int cur_b = 0;
    while(path.size())
```

```
  {
    int i = get_index(path[0]);
    int j = get_index(path[1]);
    int tmp = edges[i][j].max_b;
    for (int k = 2; k < path.size(); k++)
    {
      i=j;
      j = get_index(path[k]);
      if (edges[i][j].max_b < tmp)
      tmp = edges[i][j].max_b;
    }

    max_flow += tmp;

    i = get_index(path[0]);
    j = get_index(path[1]);
    edges[i][j].max_b -= tmp;
    edges[i][j].result_b += tmp;
    for (int k = 2; k < path.size(); k++)
    {
      i = j;
      j = get_index(path[k]);
      edges[i][j].max_b -= tmp;
      edges[i][j].result_b += tmp;
    }

    path = find_path();
  }
  return  max_flow;
}

void print_answer()
{
  int i, j;
  vector<char> order = vertex;
  sort(order.begin(), order.end());
  cout << max_flow << endl;
```

```
    for (int k = 0; k < vertex.size(); k++)
    {
      i = get_index(order[k]);
      for (int l = 0; l < vertex.size(); l++)
      {
        j = get_index(order[l]);
        if (edges[i][j].flag)
          cout << vertex[i] << ' ' << vertex[j] << ' ' << edges[i][j].result_b << endl;
      }
    }
  }
};


int main()
{
    int n;
    cin >> n;
    char start, finish;
    cin >> start >> finish;
    Graph graph(start, finish);
    graph.vertex.push_back(start);
    graph.vertex.push_back(finish);
    for (int i = 0; i < n; i++)
    {
      int l;
      cin >> start >> finish >> l;
      int index_f = graph.get_index(start);
      int index_d = graph.get_index(finish);
      graph.edges[index_f][index_d].max_b = l;
      graph.edges[index_f][index_d].flag = true;
    }
    graph.find_flow();
    graph.print_answer();

    return 0;
}
```