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: Максимальный поток

Студент гр. 9383 _____ Ноздрин В.Я.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Реализовать жадный алгоритм и альгоритм A*. Оценить сложность алгоритмов.

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 w_ij — ребра

Описание алгоритма.

Для решения задачи был реализован алгоритм Форда-Фалкерсона поиска максимального потока в графе. Идея алгоритма заключается в следующем. Изначально величине потока присваивается значение 0:
f(u,v)=0 для всех u,v из V. Затем величина потока итеративно увеличивается посредством поиска увеличивающего пути (путь от источника s к стоку t, вдоль которого можно послать больший поток). Процесс повторяется, пока можно найти увеличивающий путь.

Оценка сложности жадного алгоритма.

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).

Описание функций и структур данных.

class Solution – класс, хранящий информацию о вершинах. Вспомогательный инструмент в реализации алгоритма.

double mainAlgorithm – метод, выполняющий построение потока.

void printCapacity() – метод, выводящий на экран пропускные способности ребер.

Выводы.

Применен на практике алгоритм поиска максимального потока в графе. Исследован алгоритм Форда-Фалкерсона на предмет сложности.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <iostream>
#include "Solution.h"


int main() {
  Solution f;
  double res = f.mainAlgorithm();
  std::cout << res << '\n';
  f.printCapacity();
  return 0;
}
```

Файл Solution.cpp:

```
#include "Solution.h"

Solution::Solution() {
  int n;
  std::cin >> n >> start >> end;
  char first, second;
  double weight;
  for (int i = 0; i < n; i++) {
    std::cin >> first >> second >> weight;
    edges[first][second] = weight;
  }
}

void Solution::printEdges(const std::map<char, double>& edges, const
char& vertex) {
  for (auto& edge : edges)
    std::cout << vertex << " -> " << edge.first << " (" << edge.second <<
")\n";
}

void Solution::printFrontier(const std::queue<char>& f) {
  std::queue<char> tmp = f;
  std::vector<char> arr;
  while (!tmp.empty()) {
    arr.push_back(tmp.front());
    tmp.pop();
  }
  for (auto& el : arr)
    std::cout << el;
  std::cout << '\n';
}

void Solution::printCapacity(){
  for (auto& a : capacity)
    for (auto& b : a.second)
      std::cout << a.first << ' ' << b.first << ' ' << b.second << '\n';
}
```

```
bool Solution::isNewFrontier(const std::queue<char>& f, char v) {
  auto fCopy = f;
  while (!fCopy.empty()) {
    if (fCopy.front() == v)
      return true;
    fCopy.pop();
  }
  return false;
}

void Solution::findPath() { // width-first search for a path
  char current = start;
  bool isFound = false;
  std::queue<char> frontier, newFrontier;
  std::map<char, char> from;
  path.clear();
  visited.clear();
  visited.emplace(start);
#if COMMENTS
  std::cout << "\nSearching path...\n";
#endif
  while (!isFound) {
    std::vector<std::pair<char, double>> curPaths =
std::vector<std::pair<char, double>>();
#if COMMENTS
    std::cout << "Current vertex: " << current << '\n';
#endif
    if (edges.find(current) != edges.end()) {
      std::map<char, double> foundEdges = edges.find(current)->second;
      for (auto& edge : foundEdges)
        if (edge.second > 0)
          curPaths.emplace_back(edge.first, edge.second);
#if COMMENTS
      printEdges(foundEdges, current);
#endif
    }
#if COMMENTS
    else
      std::cout << "No edges found\n";
#endif
    for (auto& vertex : curPaths) {
#if COMMENTS
      std::cout << "Step \"" << current << "->" << vertex.first << "(" <<
vertex.second << ")\"...\n";
#endif
      if ((visited.find(vertex.first) == visited.end()) // vertex is not
visited
      && (!isNewFrontier(newFrontier, vertex.first) || vertex.second >
edges[from[vertex.first]][vertex.first])) {
        if (!isNewFrontier(newFrontier, vertex.first))
          newFrontier.push(vertex.first);
        from[vertex.first] = current;
#if COMMENTS
        std::cout << "added to new frontier.\n";
#endif
      }
#if COMMENTS
      else
```

```
        std::cout << "Was visited -> slip.\n";
#endif
    }

    if (frontier.empty() && !(newFrontier.empty())) {
      auto tmp = newFrontier;
      while (!tmp.empty()) {
        if (tmp.front() == end) {
#if COMMENTS
          std::cout << "End vertex reached. Path is found.\n";
#endif
          isFound = true;
          break;
        }
        visited.emplace(tmp.front());
        tmp.pop();
      }
      frontier = newFrontier;
      while (!newFrontier.empty())
        newFrontier.pop();
    } else if (frontier.empty() && newFrontier.empty()) {
#if COMMENTS
      std::cout << "No more paths found\n";
#endif
      break;
    }
#if COMMENTS
    std::cout << "Frontier:\n";
    printFrontier(frontier);
#endif
    current = frontier.front();
    frontier.pop();
  }

  if (isFound) {
    char cur = end;
    while (cur != start) {
      path.push_back(cur);
      cur = from[cur];
    }
    path.push_back(start);
    std::reverse(path.begin(), path.end());
#if COMMENTS
    std::cout << "Path: ";
    for (auto& v : path)
      std::cout << v;
    std::cout << '\n';
#endif
  }
}

double Solution::mainAlgorithm() {
  double curMaxFlow = 0;
  for (auto& v1 : edges)
    for (auto& v2 : v1.second) {
      capacity[v1.first][v2.first] = 0;
    }
```

```
  while (findPath(), !path.empty()) {
    char first = start;
    char second;
    double MinCapacity = 1000;
#if COMMENTS
    std::cout << "Path\n";
#endif
    for (int i = 1; i < path.size(); i++) {
      second = path[i];
      double CurCapacity = edges[first][second];
      if (CurCapacity < MinCapacity)
        MinCapacity = CurCapacity;
#if COMMENTS
      std::cout << "\t" << first << "->" << second << "(" << CurCapacity
<< ")\n";
#endif
      first = second;
    }
#if COMMENTS
    std::cout << "Path capacity = " << MinCapacity << '\n';
#endif
    first = start;
    for (int i = 1; i < path.size(); i++) {
      second = path[i];
      edges[first][second] -= MinCapacity;
      edges[second][first] += MinCapacity;
      first = second;
    }
    curMaxFlow += MinCapacity;
    first = start;
    for (int i = 1; i < path.size(); i++) {
      second = path[i];
      if (capacity.find(first) != capacity.end() &&
capacity[first].find(second) != capacity[first].end())
        capacity[first][second] += MinCapacity;
      else
        capacity[second][first] -= MinCapacity;
      first = second;
    }
  }

  return curMaxFlow;
}
```

Файл Solution.h:

```
#ifndef LAB3_SOLUTION_H
#define LAB3_SOLUTION_H

#define COMMENTS 0

#include <iostream>
#include <map>
#include <string>
#include <set>
#include <vector>
#include <queue>
```

```
#include <algorithm>

class Solution {
 public:
  Solution();
  static void printEdges(const std::map<char, double>& edges, const char&
vertex);
  static void printFrontier(const std::queue<char>& f);
  void printCapacity();
  static bool isNewFrontier(const std::queue<char>& f, char v);
  void findPath();
  double mainAlgorithm();
 private:
  char start{}, end{};
  std::map<char, std::map<char, double>> edges, capacity;
  std::set<char> visited;
  std::vector<char> path;
};

#endif //LAB3_SOLUTION_H
```