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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Построение и анализ алгоритмов»

ТЕМА: Жадный алгоритм и A*

Студент гр. 9382 _____ Рыжих Р.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Разработать жадный алгоритм и алгоритм A* для поиска пути в графе.

Задание.

Жадный алгоритм.

Разработайте программу, которая решает задачу построения пути в ориентированном графе при помощи жадного алгоритма. Жадность в данном случае понимается следующим образом: на каждом шаге выбирается последняя посещённая вершина. Переместиться необходимо в ту вершину, путь до которой является самым дешёвым из последней посещённой вершины. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес.

Пример входных данных

a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
d e 1.0

В первой строке через пробел указываются начальная и конечная вершины

Далее в каждой строке указываются ребра графа и их вес

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной. Для приведённых в примере входных данных ответом будет

abcde

Алгоритм A*.

Разработайте программу, которая решает задачу построения кратчайшего пути в ориентированном графе методом A*. Каждая вершина в графе имеет буквенное обозначение ("a", "b", "c"...), каждое ребро имеет неотрицательный вес. В качестве эвристической функции следует взять близость символов, обозначающих вершины графа, в таблице ASCII.

Пример входных данных

a e

a b 3.0

b c 1.0

c d 1.0

a d 5.0

d e 1.0

В первой строке через пробел указываются начальная и конечная вершины

Далее в каждой строке указываются ребра графа и их вес

В качестве выходных данных необходимо представить строку, в которой перечислены вершины, по которым необходимо пройти от начальной вершины до конечной. Для приведённых в примере входных данных ответом будет

ade

Вариант 8

Перед выполнением A* выполнять предобработку графа: для каждой вершины отсортировать список смежных вершин по приоритету.

Описание алгоритма.

Жадный алгоритм:

1. Начиная со стартовой вершины просматриваются смежные вершины от текущей. Среди этих смежных вершин выбирается та, у которой

вес ребра наименьший. Данная новая вершина прибавляется к текущему пути, а просматриваемая вершина считается пройденной.

2. Далее происходит то же самое для вершины, которая была выбрана на предыдущем шаге.
3. Если все смежные вершины от текущей пройдены, то нужно вернуться в пути на одну вершину назад.
4. Алгоритм считается завершенным, как только будет рассматриваться конечная вершина.

Алгоритм A*:

1. На каждом шаге выбирается вершина с наименьшим приоритетом. Приоритет определяется с помощью функции для оценки приоритета, которая состоит из эвристической функции и текущего расстояния от начальной вершины.
2. Далее для данной вершины рассматриваются смежные ей вершины.
3. Для каждой смежной вершины проверяется ее кратчайший путь до начальной вершины.
4. Если текущий путь короче, чем кратчайший путь, то текущий путь становится кратчайшим.
5. Далее данная смежная вершина помещается в очередь с приоритетом, где значение приоритета определяется при помощи функции оценки приоритета.
6. Алгоритм считается завершенным, как только будет рассматриваться конечная вершина.

Сложность.

Жадный алгоритм:

Сложность по времени работы O(n * E), где n – количество вершин, E — количество ребер, так как на каждом шаге алгоритма рассматриваются смежные ребра.

Для хранения графа используется список смежности, поэтому в этом случае сложность O(E), где E — количество ребер в графе. При этом используется стек с вершинами, следовательно сложность будет O(n + E), где n – количество вершин в графе.

Алгоритм A*:

Лучший случай, когда имеется более подходящая эвристическая функция (в данном случае, у нас стандартная эвристика), которая позволяет делать каждый шаг в нужном направлении. Сложность по времени будет O(n + E), где n – количество вершин, E – количество ребер графа.

Худший случай, когда определение нужного направления происходит достаточно долго, тогда нужно проходить всевозможные пути. Следовательно, время работы будет расти экспоненциально по сравнению с длиной оптимального пути.

В лучшем случае для каждой вершины будет храниться путь от начала до самой вершины. Оценка сложности по памяти будет O(n * (n + E)), где V — количество вершин, E — количество ребер графа.

В худшем случае все пути будут храниться в очереди, и сложность по памяти будет экспоненциальной.

Описание функций и структур данных.

Структуры данных:

class FindingPath – класс для поиска кратчайшего пути.

map<char, vector<pair<char, double>>> graph — структура данных для хранения графа.

map<char, bool> visited — структура данных для отслеживания посещенных вершин.

int Heuristic(char a, char b) —эвристическая функция (алгоритм A*).

map<char, pair<vector<char>, double>> ShortPathes — структура данных, отвечающая за текущие кратчайшие пути от начальной вершины (алгоритм A*).

priority_queue<pair<char, double>, vector<pair<char, double>>, Sorting> PriorityQueue — очередь в алгоритме A*. Состоит из названия вершины и оценочной функции(кратчайшее расстояние до вершины + эвристическая функция). Для очереди есть специальный компаратор *Sorting*, который определяет приоритет.

Также, в классе FindingPath присутствуют следующие поля:

char start – начальная вершина.

char end – конечная вершина.

int number – количество вершин

Функции:

FindingPath::Read() — функция для считывания данных. Также для алгоритма A* считывается эвристические функции (по заданию).

vector<char> FindingPath:: AStar() — функция, которая реализует алгоритм A*. Функция возвращает вектор, состоящий из вершин, которые входят в кратчайший путь.

vector<char> FindingPath::GreedyAlgorithm() — функция, которая реализует жадный алгоритм. Функция возвращает вектор, состоящий из вершин, которые входят в кратчайший путь.

Демонстрация работы.
Жадный алгоритм

Ввод	Вывод
a g a b 3.0 a c 1.0 b d 2.0	Для вершины a есть следующие смежные вершины: b(3) c(1) g(8)

b e 3.0 d e 4.0 e a 1.0 e f 2.0 a g 8.0 f g 1.0 0	Отсортированные вершины: c(1) b(3) g(8) Для вершины b есть следующие смежные вершины: d(2) e(3) Отсортированные вершины: d(2) e(3) Для вершины d есть следующие смежные вершины: e(4) Отсортированные вершины: e(4) Для вершины e есть следующие смежные вершины: a(1) f(2) Отсортированные вершины: a(1) f(2) Для вершины f есть следующие смежные вершины: g(1) Отсортированные вершины: g(1) Жадный алгоритм: Текущая вершина - a Рассматривается смежная вершина - c Идём в вершину c, т.к. наименьший вес (1) и/или идёт первой Рассматривается смежная вершина - b

	Рассматривается смежная вершина - g Текущая вершина - c Текущая вершина - a Рассматривается смежная вершина - c Рассматривается смежная вершина - b Идём в вершину b, т.к. наименьший вес (3) и/или идёт первой Рассматривается смежная вершина - g Текущая вершина - b Рассматривается смежная вершина - d Идём в вершину d, т.к. наименьший вес (2) и/или идёт первой Рассматривается смежная вершина - e Текущая вершина - d Рассматривается смежная вершина - e Идём в вершину e, т.к. наименьший вес (4) и/или идёт первой Текущая вершина - e Рассматривается смежная вершина - a Рассматривается смежная вершина - f Идём в вершину f, т.к. наименьший вес (2) и/или идёт первой

	Текущая вершина - f Рассматривается смежная вершина - g Идём в вершину g, т.к. наименьший вес (1) и/или идёт первой Конец алгоритма! Ответ:abdefg

Алгоритм A*

Ввод	Вывод
a e a b 8.0 a c 1.0 c d 1.0 d e 1.0 b e 1.0 0	Для вершины a есть следующие смежные вершины: b(8) c(1) Отсортированные вершины: c(1) b(8) Для вершины b есть следующие смежные вершины: e(1) Отсортированные вершины: e(1) Для вершины c есть следующие смежные вершины: d(1) Отсортированные вершины: d(1) Для вершины d есть следующие смежные вершины: e(1) Отсортированные вершины: e(1) Алгоритм A*: Текущая вершина - a Рассматривается смежная для a вершина c В путь родительской вершины добавляется текущая вершина c(1)

	Эвристика для вершин a и c = 2 Рассматривается смежная для a вершина b В путь родительской вершины добавляется текущая вершина b(8) Эвристика для вершин a и b = 1 Текущая вершина - c Рассматривается смежная для c вершина d В путь родительской вершины добавляется текущая вершина d(2) Эвристика для вершин c и d = 1 Текущая вершина - d Рассматривается смежная для d вершина e В путь родительской вершины добавляется текущая вершина e(3) Эвристика для вершин d и e = 1 Найдена конечная вершина! Ответ:acde

Тестирование.

Жадный алгоритм:

Входные данные	Выходные данные
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0 0	abcde

a g a b 3.0 a c 1.0 b d 2.0 b e 3.0 d e 4.0 e a 3.0 e f 2.0 a g 8.0 f g 1.0 0	abdefg
a g a b 3.0 a c 1.0 b d 2.0 b e 3.0 d e 4.0 e a 3.0 e f 2.0 a g 8.0 f g 1.0 c m 1.0 m n 1.0 0	abdefg
a d a b 1.0 b c 1.0 c a 1.0 a d 8.0	abcad

0	

Алгоритм A*:

Входные данные	Выходные данные
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0 0	ade
a d a b 1.0 b c 9.0 c d 3.0 a d 9.0 a e 1.0 e d 3.0 0	aed
a f a c 1.0 a b 1.0 c d 2.0 b e 2.0 d f 3.0 e f 3.0 0	acdf
a d	abd

a b 3.0 b c 2.0 b d 2.0 c d 4.0 a c 5.0 0	

Выводы.

В ходе выполнения лабораторной работы был разработан эадный алгоритм, а также разработан алгоритм A*

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Алгоритм A*

```
#include <iostream>
#include <vector>
#include <map>
#include <queue>
#include <algorithm>
using namespace std;



class FindingPath {
public:
    FindingPath() = default;
    vector<char> GreedyAlgorithm();
    vector<char> AStar();
    void Sort();
    void SortAStar();
    void Read();
    int Heuristic(char a, char b);

private:
    map<char, vector<pair<char, double>>> graph;
    map<char, bool> visited;
    char start;
    char end;
    int number;
};

struct Sorting { //функция сортировки для приоритетной очереди
    bool operator() (pair<char, double> a, pair<char, double> b) {
        //если стоимость двух вершин равна, то возвращается меньшая из них в алфавитном
порядке, если стоимость разная, то большая из них
        if (a.second == b.second)
            return (a.first < b.first);
        else
            return (a.second > b.second);
    }
};

int FindingPath::Heuristic(char a, char b) {
    return abs(a - b);
}

vector<char> FindingPath::AStar() {  //A*
    cout << "\nАлгоритм A*:\n";
    map<char, pair<vector<char>, double>> ShortPathes;  //текущие кратчайшие пути
    vector<char> vertex;
    priority_queue < pair<char, double>, vector<pair<char, double>>, Sorting> Priori-
tyQueue; //очередь в алгоритме

    PriorityQueue.push({ start, 0 });
    vertex.push_back(start);
    ShortPathes[start].first = vertex;

    while (!PriorityQueue.empty()) {  //пока очередь не пуста
        if (PriorityQueue.top().first == end) {  //если найдена конечная вершина
            cout << "Найдена конечная вершина!\n";
            return ShortPathes[end].first;  //то заканчивается поиск
        }
```

```
        auto TmpVertex = PriorityQueue.top();  //достается приоритетная вершина из оче-
реди
        cout << "Текущая вершина - " << TmpVertex.first << endl;
        PriorityQueue.pop();

        for (auto& i : graph[TmpVertex.first]) {  //рассматриваются все вершины, которые
соединены с текущей вершиной
            cout << "   Рассматривается смежная для " << TmpVertex.first << " вершина "<<
i.first << endl;
            double CurLength = ShortPathes[TmpVertex.first].second + i.second;
            if (ShortPathes[i.first].second == 0 || ShortPathes[i.first].second > Cur-
Length) { //если пути нет или найденный путь короче
                cout << "       В путь родительской вершины добавляется текущая вершина "
<< i.first <<"(" << CurLength << ")"<< endl;
                vector<char> path = ShortPathes[TmpVertex.first].first;  //добавляется в
путь родительской вершины текущая вершина с кратчайшим путем
                path.push_back(i.first);
                ShortPathes[i.first] = { path, CurLength };  //обновление пути и расстоя-
ния
                int heur = Heuristic(TmpVertex.first, i.first);
                cout << "   Эвристика для вершин " << TmpVertex.first << " и " << i.first
<< " = " << heur << endl << endl;
                PriorityQueue.push({ i.first, heur + ShortPathes[i.first].second });
//записывается в очередь текущая вершина
            }

        }

    }
    return ShortPathes[end].first;
}


void FindingPath::SortAStar() {
    for (auto it = graph.begin(); it != graph.end(); ++it) {
        cout << "Для вершины " << it->first << " есть следующие смежные вершины:\n";
        for (int i = 0; i < it->second.size(); i++) {
            cout << it->second[i].first << '(' << it->second[i].second << ')' << ' ';
        }
        cout << endl;
        std::sort(it->second.begin(), it->second.end(), [](pair<char, double>& a,
pair<char, double>& b) -> bool
            {
                if (a.second == b.second)
                    return (a.first < b.first);
                else
                    return (a.second < b.second);
            });
        cout << "Отсортированные вершины:\n";
        for (int j = 0; j < it->second.size(); j++) {
            cout << it->second[j].first << '(' << it->second[j].second << ')' << ' ';
        }
        cout << endl;
    }
}


void FindingPath::Sort() {
    for (auto it = graph.begin(); it != graph.end(); ++it) {
        cout << "Для вершины " << it->first << " есть следующие смежные вершины:\n";
        for (int i = 0; i < it->second.size(); i++) {
            cout << it->second[i].first << '(' << it->second[i].second << ')' << ' ';
        }
        cout << endl;
```

```
        std::sort(it->second.begin(), it->second.end(), [](pair<char, double>& a,
pair<char, double>& b) -> bool {return a.second < b.second; });

        cout << "Отсортированные вершины:\n";
        for (int j = 0; j < it->second.size(); j++) {
            cout << it->second[j].first << '(' << it->second[j].second << ')' << ' ';
        }
        cout << endl;
    }
}


void FindingPath::Read() {
    char start, end;
    cin >> start >> end;
    this->start = start;
    this->end = end;
    int count = 0;

    while (cin >> start) {
        if (start == '0')  //символ остановки ввода данных
            break;
        double weight;
        cin >> end >> weight;
        graph[start].push_back({ end,weight });
        visited[start] = false;
        visited[end] = false;
        count++;
    }
    this->number = count;
}



vector<char> FindingPath::GreedyAlgorithm() {
    cout << "\nЖадный алгоритм:\n";
    double min;
    vector<char> result;
    result.reserve(this->number);
    result.push_back(this->start);

    char CurVertex = this->start;

    while (CurVertex != this->end) {
        cout << "Текущая вершина - " << CurVertex << endl;
        char NextVertex;
        min = 100;
        bool found = false;

        for (auto& i : this->graph[CurVertex]) {
            cout << "Рассматривается смежная вершина - " << i.first << endl;
            if (!visited[i.first] && i.second < min) {
                cout << "Идём в вершину " << i.first << ", т.к. наименьший вес (" <<
i.second << ") и/или идёт первой\n";
                min = i.second;
                NextVertex = i.first;
                found = true;

            }
        }
        cout << endl;
        visited[CurVertex] = true;

        if (!found) {
            if (!result.empty()) {
```

```
                result.pop_back();
                CurVertex = result.back();
            }
            continue;
        }
        CurVertex = NextVertex;
        result.push_back(CurVertex);
    }
    cout << "Конец алгоритма!\n";
    return result;
}


int main() {
    setlocale(LC_ALL, "Russian");
    FindingPath answer;
    answer.Read();
    answer.SortAStar();
    vector<char> out = answer.AStar();
    cout << "Ответ:";
    for (auto& i : out) {
        cout << i;
    }
    return 0;
}
```

Жадный алгоритм

```
#include <iostream>
#include <vector>
#include <map>
#include <queue>
#include <algorithm>
using namespace std;



class FindingPath {
public:
    FindingPath() = default;
    vector<char> GreedyAlgorithm();
    vector<char> AStar();
    void Sort();
    void SortAStar();
    void Read();
    int Heuristic(char a, char b);

private:
    map<char, vector<pair<char, double>>> graph;
    map<char, bool> visited;
    char start;
    char end;
    int number;
};

struct Sorting { //функция сортировки для приоритетной очереди
    bool operator() (pair<char, double> a, pair<char, double> b) {
        //если стоимость двух вершин равна, то возвращается меньшая из них в алфавитном
порядке, если стоимость разная, то большая из них
        if (a.second == b.second)
            return (a.first < b.first);
        else
            return (a.second > b.second);
    }
};
```

```
int FindingPath::Heuristic(char a, char b) {
    return abs(a - b);
}

vector<char> FindingPath::AStar() {  //A*
    cout << "\nАлгоритм A*:\n";
    map<char, pair<vector<char>, double>> ShortPathes;  //текущие кратчайшие пути
    vector<char> vertex;
    priority_queue < pair<char, double>, vector<pair<char, double>>, Sorting> Priori-
tyQueue; //очередь в алгоритме

    PriorityQueue.push({ start, 0 });
    vertex.push_back(start);
    ShortPathes[start].first = vertex;

    while (!PriorityQueue.empty()) {  //пока очередь не пуста
        if (PriorityQueue.top().first == end) {  //если найдена конечная вершина
            cout << "Найдена конечная вершина!\n";
            return ShortPathes[end].first;  //то заканчивается поиск
        }

        auto TmpVertex = PriorityQueue.top();  //достается приоритетная вершина из оче-
реди
        cout << "Текущая вершина - " << TmpVertex.first << endl;
        PriorityQueue.pop();

        for (auto& i : graph[TmpVertex.first]) {  //рассматриваются все вершины, которые
соединены с текущей вершиной
            cout << "   Рассматривается смежная для " << TmpVertex.first << " вершина "<<
i.first << endl;
            double CurLength = ShortPathes[TmpVertex.first].second + i.second;
            if (ShortPathes[i.first].second == 0 || ShortPathes[i.first].second > Cur-
Length) { //если пути нет или найденный путь короче
                cout << "       В путь родительской вершины добавляется текущая вершина "
<< i.first <<"(" << CurLength << ")"<< endl;
                vector<char> path = ShortPathes[TmpVertex.first].first;  //добавляется в
путь родительской вершины текущая вершина с кратчайшим путем
                path.push_back(i.first);
                ShortPathes[i.first] = { path, CurLength };  //обновление пути и расстоя-
ния
                int heur = Heuristic(TmpVertex.first, i.first);
                cout << "   Эвристика для вершин " << TmpVertex.first << " и " << i.first
<< " = " << heur << endl << endl;
                PriorityQueue.push({ i.first, heur + ShortPathes[i.first].second });
//записывается в очередь текущая вершина
            }

        }

    }
    return ShortPathes[end].first;
}


void FindingPath::SortAStar() {
    for (auto it = graph.begin(); it != graph.end(); ++it) {
        cout << "Для вершины " << it->first << " есть следующие смежные вершины:\n";
        for (int i = 0; i < it->second.size(); i++) {
            cout << it->second[i].first << '(' << it->second[i].second << ')' << ' ';
        }
        cout << endl;
        std::sort(it->second.begin(), it->second.end(), [](pair<char, double>& a,
pair<char, double>& b) -> bool
            {
```

```
                if (a.second == b.second)
                    return (a.first < b.first);
                else
                    return (a.second < b.second);
            });
        cout << "Отсортированные вершины:\n";
        for (int j = 0; j < it->second.size(); j++) {
            cout << it->second[j].first << '(' << it->second[j].second << ')' << ' ';
        }
        cout << endl;
    }
}


void FindingPath::Sort() {
    for (auto it = graph.begin(); it != graph.end(); ++it) {
        cout << "Для вершины " << it->first << " есть следующие смежные вершины:\n";
        for (int i = 0; i < it->second.size(); i++) {
            cout << it->second[i].first << '(' << it->second[i].second << ')' << ' ';
        }
        cout << endl;
        std::sort(it->second.begin(), it->second.end(), [](pair<char, double>& a,
pair<char, double>& b) -> bool {return a.second < b.second; });

        cout << "Отсортированные вершины:\n";
        for (int j = 0; j < it->second.size(); j++) {
            cout << it->second[j].first << '(' << it->second[j].second << ')' << ' ';
        }
        cout << endl;
    }
}


void FindingPath::Read() {
    char start, end;
    cin >> start >> end;
    this->start = start;
    this->end = end;
    int count = 0;

    while (cin >> start) {
        if (start == '0')  //символ остановки ввода данных
            break;
        double weight;
        cin >> end >> weight;
        graph[start].push_back({ end,weight });
        visited[start] = false;
        visited[end] = false;
        count++;
    }
    this->number = count;
}



vector<char> FindingPath::GreedyAlgorithm() {
    cout << "\nЖадный алгоритм:\n";
    double min;
    vector<char> result;
    result.reserve(this->number);
    result.push_back(this->start);

    char CurVertex = this->start;

    while (CurVertex != this->end) {
```

```
        cout << "Текущая вершина - " << CurVertex << endl;
        char NextVertex;
        min = 100;
        bool found = false;

        for (auto& i : this->graph[CurVertex]) {
            cout << "Рассматривается смежная вершина - " << i.first << endl;
            if (!visited[i.first] && i.second < min) {
                cout << "Идём в вершину " << i.first << ", т.к. наименьший вес (" <<
i.second << ") и/или идёт первой\n";
                min = i.second;
                NextVertex = i.first;
                found = true;

            }
        }
        cout << endl;
        visited[CurVertex] = true;

        if (!found) {
            if (!result.empty()) {
                result.pop_back();
                CurVertex = result.back();
            }
            continue;
        }
        CurVertex = NextVertex;
        result.push_back(CurVertex);
    }
    cout << "Конец алгоритма!\n";
    return result;
}


int main() {
    setlocale(LC_ALL, "Russian");
    FindingPath answer;
    answer.Read();
    answer.Sort();
    vector<char> out = answer.GreedyAlgorithm();
    cout << "Ответ:";
    for (auto& i : out) {
        cout << i;
    }
    return 0;
}
```