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. 9383 _____ Нистратов Д.Г.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

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.

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.

Вариант 2

В A* эвристическая функция для каждой вершины задаётся неотрицательным числом во входных данных.

Постановка задачи.

На вход подается два значения, указывающие начальную и конечную точку в поиске пути. Далее до прерывания пользователем идет считывание ребер графа с указанием дополнительных значений. В Жадном алгоритме – длинна ребер, а в A* - длинна и эвристика для каждой вершины. Результатом программы должна являться строка с перечислением вершин, необходимых для кротчайшего пути из начальной в конечную точку графа.

Выполнение работы.

1) Жадный алгоритм

Для хранения введенной информации был описан класс Graph, содержащий имя узлов, а также вес между ними.

Для реализации алгоритма была описана функции Greed, осуществляющая обход по графу и производящая поиск наименьшего соседа, с помощью дополнительной функции findMinPath, выполняющая обход всех соседей данного узла и сверяющая их стоимость.

2) A* алгоритм

При чтении информации с потока информация заносится в создание классы: *Neigbour, Node, Graph.* Класс Node описывает информацию узла, а именно название узла, эвристику, список доступных для перемещения узлов и информацию о прошлом узле. Класс Neigbour хранит название узла и стоимость пути к данному узлу. Класс Graph описывает главные функции программы:

void inputNode(char first, char second, float weight); - осуществляет ввод узла, а также его соседей по заданному весу.

Node getMinNode(); - осуществляет поиск наименьшей стоимости узла в списке доступных для перемещения узлов.*

*void AStar(); - алгоритм A**

Реализация алгоритма A*:

1) Устанавливается начальное значение в старте.
2) Цикл пока список узлов не пуст.
3) Устанавливается текущий узел равный наименьшей стоимости в списке.
4) Осуществляется проверка на совпадение с конечным узлом
5) Высчитывается новая стоимость
6) Сравнивается новая стоимость с предыдущей
7) Если новая стоимость оказалась меньше, то изменяется значение стоимости и начальным узлом устанавливается текущий узел.

Сложность алгоритма.

Жадный алгоритм: Сложность O(N^2)

A*: Сложность O(N logN)

Тестирование.

Тестирование программы произведено с помощью сторонней библиотеки catch2. Тесты описаны в файле testA.cpp и testG.cpp.

Результаты тестирования приведены в Таблица 1 для Жадного алгоритма и в *Таблица 2* для A*.

Таблица 1

Входные данные	*Ответ*	*Описание теста*
a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	*abcde*	*Первый наименьший путь*
a e a b 3.0 b c 1.0 c d 1.0 a d 2.0 d e 1.0	*ade*	*Первый наименьший путь через ade*
a e a b 1.0 b c 1.0 c d 1.0 a d 1.0 d e 1.0	*abcde*	*Одинаковая стоимость*

c e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0	*cde*	*Начало не с 1 элемента*
a e a b 1.0 b c 1.0 c d 1.0 a d 1.0 d e 1.0	*ab*	*Путь задан сразу*

Таблица 2

Входные данные	*Ответ*	*Описание теста*
a e a b 3.0 4.0 3.0 b c 1.0 3.0 2.0 c d 1.0 2.0 1.0 a d 5.0 4.0 1.0 d e 1.0 1.0 0.0	*ade*	*Одинаковые стоимости*
a e a b 3.0 4.0 3.0 b c 1.0 3.0 2.0 c d 1.0 2.0 1.0 a d 6.0 4.0 1.0 d e 1.0 1.0 0.0	*abcde*	*Наименьший путь*

a e a b 3.0 0.0 0.0 b c 1.0 0.0 0.0 c d 1.0 0.0 0.0 a d 5.0 0.0 0.0 d e 1.0 0.0 0.0	*ade*	*Пустые значения эвристики*
c e a b 3.0 4.0 3.0 b c 1.0 3.0 2.0 c d 1.0 2.0 1.0 a d 5.0 4.0 1.0 d e 1.0 1.0 0.0	*cde*	*Начало не с 1 точки*
a b a b 3.0 4.0 3.0	*ab*	*Путь задан сразу*
a e a b 3.0 -4.0 -3.0 b c 1.0 -3.0 -2.0 c d 1.0 2.0 1.0 a d 5.0 4.0 -1.0 d e 1.0 -1.0 0.0	*ade*	*Отрицательные значения эвристики*

Вывод.

В ходе лабораторный работы был исследован жадный алгоритм поиска пути в графе и алгоритм A*. Были написаны программы реализующие нахождение заданного пути в графе, а также были рассмотрены различия данных алгоритмов и проведены расчеты сложности алгоритмов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: graph.cpp

```
#include "graph.h"


void Graph::inputNode(char first, char second, float weight){
    Node* firstNode = nullptr;
    Node* secondNode = nullptr;


    firstNode = getNodeByName(first);
    if (!firstNode) {
        firstNode = new Node(first);
        nodes.push_back(firstNode);
    }


    secondNode = getNodeByName(second);
    if (!secondNode) {
        secondNode = new Node(second);
        nodes.push_back(secondNode);
    }


    firstNode->neighbours.push_back(new Neighbour(secondNode, weight));
}


std::vector<Node *> Graph::getNodes(){
    return nodes;
}
```

```
/* WORKS ONLY ON WINDOWS!!!!! std::cin.clear() ????
void Graph::setHeuristic(){
    std::cout << "Set Heuristics: " << std::endl;
    for (auto& node : nodes){
        //node->heuristic = abs((int)end - (int)node->name);

        std::cout << node->name << ": ";
        std::cin >> node->heuristic;
        std::cout << std::endl;
        if (node->heuristic < 0){
            node->heuristic = abs(node->heuristic);
        }
    }
}*/

Node* Graph::getNodeByName(char name){
    for (auto& elem: nodes){
        if (elem->name == name){
            return elem;
        }
    }
    return nullptr;
}

int Graph::getIndexByNode(Node* node){
    for (int i = 0; i < nodes.size(); i++){
```

```
        if (node->name == nodes[i]->name){
            return i;
        }
    }
    return -1;
}

Node* Graph::getMinNode(){
    Node* temp = nullptr;
    for (auto& node : nodes){
        if (!temp){
            temp = node;
            continue;
        }
        if (node->cost == temp->cost && node->name > temp->name){
            temp = node;
        }
        else if (node->cost < temp->cost){
            temp = node;
        }
    }
    return temp;
}

void Graph::printGraph(){
    for (auto& elem : nodes){
```

```
            std::cout << "Node : " << elem->name << " Heuristic: " << elem-
>heuristic << std::endl;
            for (auto& neighbour_in : elem->neighbours){
               std::cout << "Neigbour : " << neighbour_in->nodeName-
>name << " Weight: " << neighbour_in->weight << \
               " Cost: " <<  neighbour_in->weight + neighbour_in->nodeName-
>heuristic << std::endl;
            }
        }
    }
    void Graph::printNodes(){
        for (auto& a : nodes){
            std::cout << a->name << std::endl;
        }
    }

    void Graph::printAnswer(){
        std::string answer;
        while (endNode->cameFrom){
            answer += endNode->name;
            endNode = endNode->cameFrom;
        }
        answer += endNode->name;
        std::reverse(answer.begin(), answer.end());
        std::cout << answer << std::endl;
    }

    std::string Graph::getAnswer(){
```

```
    std::string answer;
    while (endNode->cameFrom){
        answer += endNode->name;
        endNode = endNode->cameFrom;
    }
    answer += endNode->name;
    std::reverse(answer.begin(), answer.end());
    return answer;
}

void Graph::AStar(){
    startNode = getNodeByName(start);
    startNode->cost = startNode->heuristic;
    Node* currNode;

    while (!nodes.empty()){
        currNode = getMinNode();
        nodes.erase(nodes.begin() + getIndexByNode(currNode));

        if (currNode->name == end){
            endNode = currNode;
            break;
        }

        for (auto& node : currNode->neighbours){

            float newCost = currNode->cost - currNode->heuristic + node->weight;
```

```
            if (node->nodeName->cost > newCost){
                node->nodeName->cost = newCost + node->nodeName->heuristic;
                node->nodeName->cameFrom = currNode;
            }
        }
    }
}
```

Название файла: graph.h

```
#pragma once
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <sstream>

class Node;

class Neighbour{
public:
    Neighbour(Node* nodeName, float weight) : nodeName(nodeName), weight(weight) {};
    Node* nodeName;
    float weight;
};

class Node{
```

```
public:
    Node(char name) : name(name) {};
    char name;
    float heuristic = __FLT_MAX__, cost = __FLT_MAX__;
    std::vector<Neighbour*> neighbours;
    Node* cameFrom = nullptr;
};

class Graph{
public:
    Graph(char start, char end) : start(start), end(end) {};

    void inputNode(char first, char second, float weight);
    //void setHeuristic();

    Node* getNodeByName(char name);
    int getIndexByNode(Node* node);
    Node* getMinNode();
    std::vector<Node *> getNodes();

    void printGraph();
    void printNodes();
    void printAnswer();
    std::string getAnswer();

    void AStar();
```

```
private:
    char start, end;
    std::vector<Node *> nodes;

    Node *startNode, *endNode;
};
```

Название файла: greedy.cpp

```
#include "greedy.h"

int findMinPath(std::vector<Graph>& graph, char start, int min_weight, std::vector<char
> corr_path) {
    Graph min;
    for (int i = 0; i < graph.size(); i++) {
        if ((graph[i].start == start) && (graph[i].correct)){
            min = graph[i];
            min_weight = i;
            break;
        }
    }

    if (min.weight == -1){
        graph[min_weight].correct = false;
        return -1;
    }

    for (int i = 0; i < graph.size(); i++) {
```

```
            if ((graph[i].weight < min.weight) && (graph[i].start == start) && (graph[i].correct)
) {
                min = graph[i];
                min_weight = i;
            }
        }
        if (std::find(corr_path.begin(), corr_path.end(), graph[min_weight].end) != corr_path.en
d()) {
            graph[min_weight].correct = false;
            return -1;
        }
        return min_weight;
    }

    void Greedy(std::vector<Graph>& graph, char start, char end, std::vector<char>& corr_p
ath)
    {
        char current = start;
        int min_weight = 0;
        while (1) {
            corr_path.push_back(current);
            min_weight = findMinPath(graph, current, min_weight, corr_path);
            if (min_weight == -1) {
                min_weight = 0;
                current = start;
                corr_path.clear();
            }
            else {
```

```
            current = graph[min_weight].end;
        }

        if (graph[min_weight].end == end) break;
    }
    corr_path.push_back(end);
}

std::string getAnswer(std::vector<char> path){
    std::string answer;
    for (auto& a : path){
        answer += a;
    }
    return answer;
}
```

Название файла: greedy.h

```
#pragma once
#include <iostream>
#include <vector>
#include <algorithm>

class Graph {
public:
    char start, end;
    float weight;
    bool correct;
```

```
    Graph() : start('\0'), end('\0'), weight(-1), correct(true) {};
    Graph(char from, char to, double weight) : start(from), end(to), weight(weight), correct(tr
ue) {}
};

int findMinPath(std::vector<Graph>& graph, char start, int min_weight, std::vector<char
> corr_path);
void Greedy(std::vector<Graph>& graph, char start, char end, std::vector<char>& corr_p
ath);
std::string getAnswer(std::vector<char> path);
```

Название файла: mainA.cpp

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

int main()
{
    char start, end;
    std::cin >> start >> end;
    Graph *graph = new Graph(start, end);
    char v1, v2;
    float h1, h2;
    float weight;
    std::string answer;
    int prevsize = 0;
    float heuristic;
```

```
    std::cout << "Input graph in this format: Node1 Node2 Weight Heuristic1 Heuristic2" <
< std::endl ;
    while (std::cin >> v1 >> v2 >> weight >> h1 >> h2)
    {
        graph->inputNode(v1, v2, abs(weight));
        graph->getNodeByName(v1)->heuristic = h1;
        graph->getNodeByName(v2)->heuristic = h2;
    }
    graph->AStar();
    graph->printAnswer();
    return 0;
}
```

Название файла: mainG.cpp

```
#include "greedy.h"

int main() {
    char start, end;
    std::cin >> start >> end;

    std::vector<Graph> graph;
    std::vector<char> corr_path;
    Graph cur_path;
    char first_point, second_point;
    float weight;

    while (std::cin >> first_point) {
```

```
        std::cin >> second_point >> weight;
        Graph cur_path(first_point, second_point, weight);
        graph.push_back(cur_path);
    }

    Greedy(graph, start, end, corr_path);

    std::cout << getAnswer(corr_path) << std::endl;
    return 0;
}
```

Название файла: testA.cpp

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

#include "../source/graph.h"
#include <sstream>

TEST_CASE("A* test for different values" ) {
    std::stringstream ss;
    char v1, v2;
    float weight, h1, h2;
    SECTION("Значение стоимости пути одинаковые") {
        Graph *graph = new Graph('a', 'e');
        ss << "a b 3.0 4.0 3.0 \n \
          b c 1.0 3.0 2.0 \n \
          c d 1.0 2.0 1.0 \n \
```

```
    a d 5.0 4.0 1.0 \n \
    d e 1.0 1.0 0.0 \n";
  while (ss >> v1 >> v2 >> weight >> h1 >> h2)
  {
    graph->inputNode(v1, v2, abs(weight));
    graph->getNodeByName(v1)->heuristic = h1;
    graph->getNodeByName(v2)->heuristic = h2;
  }
  graph->AStar();
  REQUIRE(graph->getAnswer() == "ade");
}
SECTION("Путь abcde меньше ade") {
  Graph *graph = new Graph('a', 'e');
  ss << "a b 3.0 4.0 3.0 \n \
    b c 1.0 3.0 2.0 \n \
    c d 1.0 2.0 1.0 \n \
    a d 6.0 4.0 1.0 \n \
    d e 1.0 1.0 0.0 \n";
  while (ss >> v1 >> v2 >> weight >> h1 >> h2)
  {
    graph->inputNode(v1, v2, abs(weight));
    graph->getNodeByName(v1)->heuristic = h1;
    graph->getNodeByName(v2)->heuristic = h2;
  }
  graph->AStar();
  REQUIRE(graph->getAnswer() == "abcde");
}
```

```
SECTION("Значения эвристики пустые") {
    Graph *graph = new Graph('a', 'e');
    ss << "a b 3.0 0.0 0.0 \n \
      b c 1.0 0.0 0.0 \n \
      c d 1.0 0.0 0.0 \n \
      a d 5.0 0.0 0.0 \n \
      d e 1.0 0.0 0.0 \n";
    while (ss >> v1 >> v2 >> weight >> h1 >> h2)
    {
        graph->inputNode(v1, v2, abs(weight));
        graph->getNodeByName(v1)->heuristic = h1;
        graph->getNodeByName(v2)->heuristic = h2;
    }
    graph->AStar();
    REQUIRE(graph->getAnswer() == "ade");
}
SECTION("Начало графа не с 1 точки") {
Graph *graph = new Graph('c', 'e');
ss << "a b 3.0 4.0 3.0 \n \
    b c 1.0 3.0 2.0 \n \
    c d 1.0 2.0 1.0 \n \
    a d 5.0 4.0 1.0 \n \
    d e 1.0 1.0 0.0 \n";
while (ss >> v1 >> v2 >> weight >> h1 >> h2)
{
    graph->inputNode(v1, v2, abs(weight));
    graph->getNodeByName(v1)->heuristic = h1;
```

```
    graph->getNodeByName(v2)->heuristic = h2;
}
graph->AStar();
REQUIRE(graph->getAnswer() == "cde");
}
SECTION("Отрицательные значения веса") {
    Graph *graph = new Graph('a', 'e');
    ss << "a b -3.0 4.0 3.0 \n \
       b c -1.0 3.0 2.0 \n \
       c d -1.0 2.0 1.0 \n \
       a d -5.0 4.0 1.0 \n \
       d e -1.0 1.0 0.0 \n";
    while (ss >> v1 >> v2 >> weight >> h1 >> h2)
    {
        graph->inputNode(v1, v2, abs(weight));
        graph->getNodeByName(v1)->heuristic = h1;
        graph->getNodeByName(v2)->heuristic = h2;
    }
    graph->AStar();
    REQUIRE(graph->getAnswer() == "ade");
}
SECTION("Отрицательные значения эвристики") {
    Graph *graph = new Graph('a', 'e');
    ss << "a b 3.0 -4.0 -3.0 \n \
       b c 1.0 -3.0 -2.0 \n \
       c d 1.0 2.0 1.0 \n \
       a d 5.0 4.0 -1.0 \n \
```

```
        d e 1.0 -1.0 0.0 \n";
    while (ss >> v1 >> v2 >> weight >> h1 >> h2)
    {
        graph->inputNode(v1, v2, abs(weight));
        graph->getNodeByName(v1)->heuristic = h1;
        graph->getNodeByName(v2)->heuristic = h2;
    }
    graph->AStar();
    REQUIRE(graph->getAnswer() == "ade");
  }
  SECTION("Пусть задан сразу") {
    Graph *graph = new Graph('a', 'b');
    ss << "a b 3.0 4.0 3.0 \n";
    while (ss >> v1 >> v2 >> weight >> h1 >> h2)
    {
        graph->inputNode(v1, v2, abs(weight));
        graph->getNodeByName(v1)->heuristic = h1;
        graph->getNodeByName(v2)->heuristic = h2;
    }
    graph->AStar();
    REQUIRE(graph->getAnswer() == "ab");
  }

}
```

Название файла: testG.cpp

```
#define CATCH_CONFIG_MAIN
```

```
#include "catch.hpp"

#include "../source/greedy.h"

TEST_CASE("A* test for different values" ) {
    std::stringstream ss;
    std::vector<Graph> graph;
    std::vector<char> corr_path;
    char first_point, second_point;
    float weight;
    SECTION("Первый наименьший") {
        ss << "a b  3.0\n \
           b c 1.0  \n \
           c d 1.0  \n \
           a d 5.0  \n \
           d e 1.0  \n";
        while (ss >> first_point) {
            ss >> second_point >> weight;
            Graph cur_path(first_point, second_point, weight);
            graph.push_back(cur_path);
        }

        Greedy(graph, 'a', 'e', corr_path);
        REQUIRE(getAnswer(corr_path) == "abcde");
    }
    SECTION("Первый наименьший №2") {
        ss << "a b  3.0\n \
```

```
    b c 1.0  \n \
    c d 1.0  \n \
    a d 2.0  \n \
    d e 1.0  \n";
  while (ss >> first_point) {
     ss >> second_point >> weight;
     Graph cur_path(first_point, second_point, weight);
     graph.push_back(cur_path);
  }

  Greedy(graph, 'a', 'e', corr_path);
  REQUIRE(getAnswer(corr_path) == "ade");
}
SECTION("Одинаковые стоимости") {
  ss << "a b  1.0\n \
    b c 1.0  \n \
    c d 1.0  \n \
    a d 1.0  \n \
    d e 1.0  \n";
  while (ss >> first_point) {
     ss >> second_point >> weight;
     Graph cur_path(first_point, second_point, weight);
     graph.push_back(cur_path);
  }

  Greedy(graph, 'a', 'e', corr_path);
  REQUIRE(getAnswer(corr_path) == "abcde");
```

```
}
SECTION("Начало не с первого элемента") {
    ss << "a b  3.0\n \
       b c 1.0  \n \
       c d 1.0  \n \
       a d 2.0  \n \
       d e 1.0  \n";
    while (ss >> first_point) {
        ss >> second_point >> weight;
        Graph cur_path(first_point, second_point, weight);
        graph.push_back(cur_path);
    }

    Greedy(graph, 'c', 'e', corr_path);
    REQUIRE(getAnswer(corr_path) == "cde");
}
SECTION("Путь задан сразу") {
    ss << "a b  3.0\n ";
    while (ss >> first_point) {
        ss >> second_point >> weight;
        Graph cur_path(first_point, second_point, weight);
        graph.push_back(cur_path);
    }

    Greedy(graph, 'a', 'b', corr_path);
    REQUIRE(getAnswer(corr_path) == "ab");
}
```

```
}
```

Название файла: Makefile

```
BUILD = build

SOURCE = source

TEST = tests


all: releaseA releaseG testsA testsG


releaseA: mainA.o graph.o

    g++ mainA.o graph.o -o labA


releaseG: mainG.o greedy.o

    g++ mainG.o greedy.o -o labG


testsA: testA.o graph.o

    g++ testA.o graph.o -o testA


testsG: testG.o greedy.o

    g++ testG.o greedy.o -o testG


mainA.o: $(SOURCE)/mainA.cpp

    g++ -c -O2 $(SOURCE)/mainA.cpp -o mainA.o


mainG.o: $(SOURCE)/mainG.cpp

    g++ -c -O2 $(SOURCE)/mainG.cpp -o mainG.o
```

```
graph.o: $(SOURCE)/graph.cpp
    g++ -c -O2 $(SOURCE)/graph.cpp -o graph.o

greedy.o: $(SOURCE)/greedy.cpp
    g++ -c -O2 $(SOURCE)/greedy.cpp -o greedy.o

testA.o: $(TEST)/testA.cpp
    g++ -c $(TEST)/testA.cpp -o testA.o

testG.o: $(TEST)/testG.cpp
    g++ -c $(TEST)/testG.cpp -o testG.o

clean:
    rm -rf *.o
```