

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.7304

Сергеев И.Д.

Преподаватель

Филатов Ар.Ю.

Санкт-Петербург

2019

1. Постановка задачи

1.1. Цель работы

Исследование жадного алгоритма и A^* и их реализация на языке c++.

1.2. Формулировка задачи

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.

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.

1.3. Основные теоретические положения

Жадный алгоритм – алгоритм, заключающийся в принятии локально-оптимальных решений на каждом этапе, допуская, что каждое решение является оптимальным. Известно, что если структура задачи задается матроидом, тогда применение жадного алгоритма выдаст глобальный оптимум.

A^* - модификация алгоритма Дейкстры, оптимизированная для единственной конечной точки. Алгоритм Дейкстры может находить пути ко всем точкам, а A^* находит к одной. Он отдает приоритеты путям, которые ведут ближе к цели.

2. Ход работы

2.1. Жадный алгоритм выполнен на векторе. В функцию подается начальная вершина, далее вызывается функция, находящая минимальный путь среди соседей и снова рекурсивно вызывается первая функция для следующей вершины. Далее отдельная функция восстанавливает проделанный путь и выводит его на экран.

```

#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <cmath>

using namespace std;

class Peak {
public:
    char name;
    Peak *cameFrom = nullptr;
    bool visited;
    vector<Peak*> neighbors;
    Peak(char name) {
        this->name = name;
        this->visited = false;
    }
    bool operator ==(const Peak& p) {
        return this->name == p.name;
    }
};

class Edge {
public:
    Edge(Peak *from, Peak *to, double cost) {
        this->start = from;
        this->end = to;
        this->cost = cost;
    }
    Peak *start;
    Peak *end;
    double cost;
};

string reconstruct(Peak *end) {
    Peak *current = end;
    string path = "";
    path += current->name;
    while (current->cameFrom != nullptr) {
        current = current->cameFrom;
        path = current->name + path;
    }
    return path;
}

double distBetween(Peak *current, Peak *neighbor, std::vector<Edge*> &edges) {
    for (Edge *e : edges) {
        if (e->start == current && e->end == neighbor) {
            return e->cost;
        }
    }
    return 10000.0;
}

void greed(vector<Edge*> &edges, Peak *start, Peak *goal) {
    start->visited = true;
    if (start->name == goal->name) return;
    for (int i = 0; i < start->neighbors.size(); i++) {

```

```

double min_w = 10000.0;
Peak *next = nullptr;
for (Peak *p : start->neighbors) {
    if (distBetween(start, p, edges) < min_w && p->visited == false) {
        min_w = distBetween(start, p, edges);
        next = p;
    }
}
if (next != nullptr) {
    next->cameFrom = start;
    greed(edges, next, goal);
}
else return;
}
return;
}

```

```

int main()
{
    vector<Edge*> edges;
    vector<Peak*> peaks;
    Peak *startPeak = nullptr;
    Peak *goalPeak = nullptr;
    char start, end;
    cin >> start >> end;
    string line;
    while (getline(cin, line)) {
        char from;
        char to;
        double cost;
        stringstream ss(line);
        ss >> from >> to >> cost;
        Peak *p_from = nullptr;
        Peak *p_to = nullptr;
        for (Peak *p : peaks) {
            if (p->name == from) p_from = p;
            if (p->name == to) p_to = p;
        }
        if (p_from == nullptr) {
            p_from = new Peak(from);
            peaks.push_back(p_from);
        }
        if (p_to == nullptr) {
            p_to = new Peak(to);
            peaks.push_back(p_to);
        }
        if (p_from->name == start) startPeak = p_from;
        if (p_to->name == end) goalPeak = p_to;
        (p_from->neighbors).push_back(p_to);
        edges.push_back(new Edge(p_from, p_to, cost));
    }
    greed(edges, startPeak, goalPeak);
    cout << reconstruct(goalPeak) << endl;
    return 0;
}

```

2.2. В данном алгоритме немного меняется стратегия выбора следующего узла. В качестве дополнительной оценки мы используем эвристическую функцию, которая считает расстояние до конечной вершины. A* Реализован на основе вектора.

```
#include <iostream>
```

```

#include <string>
#include <sstream>
#include <algorithm>
#include <vector>
#include <limits>
#include <cmath>

using namespace std;

class Peak {
public:
    char name;
    double gScore;
    double fScore;
    Peak *cameFrom = nullptr;
    std::vector<Peak*> neighbors;

    Peak(char name) {
        this->name = name;
        this->gScore = 10000.0;
        this->fScore = 10000.0;
    }
    bool operator ==(const Peak& p) {
        return this->name == p.name;
    }
};

class Edge {
public:
    Edge(Peak *from, Peak *to, double cost) {
        this->start = from;
        this->end = to;
        this->cost = cost;
    }
    Peak *start;
    Peak *end;
    double cost;
};

double calHeuristic(Peak *a, Peak *b) {
    return abs(a->name - b->name);
}

string reconstruct(Peak *end) {
    Peak *current = end;
    string path = "";
    path += current->name;
    while (current->cameFrom != nullptr) {
        current = current->cameFrom;
        path = current->name + path;
    }
    return path;
}

double distBetween(Peak *current, Peak *neighbor, vector<Edge*> &edges) {
    for (Edge *e : edges) {
        if (e->start == current && e->end == neighbor) {
            return e->cost;
        }
    }
    return 10000.0;
}

string A_Star(vector<Edge*> &edges, Peak *start, Peak *goal) {

```

```

vector<Peak*> closedSet;
vector<Peak*> openSet;
openSet.push_back(start);
start->gScore = 0;
start->fScore = calHeuristic(start, goal);
while (openSet.size() != 0) {
    Peak *current = openSet.at(0);
    int current_index = 0;
    for (int i = 0; i < openSet.size(); i++) {
        if (openSet[i]->fScore < current->fScore) {
            current = openSet[i];
            current_index = i;
        }
    }
    if (current == goal) {
        return reconstruct(current);
    }
    openSet.erase(openSet.begin() + current_index);
    closedSet.push_back(current);
    for (Peak *neighbor : current->neighbors) {
        if (find(closedSet.begin(), closedSet.end(), neighbor) != closedSet.end()) {
            continue;
        }
        double tentative_gScore = current->gScore + distBetween(current, neighbor,
edges);
        if (std::find(openSet.begin(), openSet.end(), neighbor) != openSet.end()) {
            if (tentative_gScore >= neighbor->gScore)
                continue;
        }
        else {
            openSet.push_back(neighbor);
        }
        neighbor->cameFrom = current;
        neighbor->gScore = tentative_gScore;
        neighbor->fScore = neighbor->gScore + calHeuristic(neighbor, goal);
    }
}
return "";
}

```

```

int main()
{
    vector<Edge*> edges;
    vector<Peak*> peaks;
    Peak *startPeak = nullptr;
    Peak *goalPeak = nullptr;
    char start, end;
    cin >> start >> end;
    string line;
    while (getline(cin, line)) {
        char from;
        char to;
        double cost;
        stringstream ss(line);
        ss >> from >> to >> cost;
        Peak *p_from = nullptr;
        Peak *p_to = nullptr;
        for (Peak *p : peaks) {
            if (p->name == from) p_from = p;
            if (p->name == to) p_to = p;
        }
    }
}

```

```

    if (p_from == nullptr) {
        p_from = new Peak(from);
        peaks.push_back(p_from);
    }
    if (p_to == nullptr) {
        p_to = new Peak(to);
        peaks.push_back(p_to);
    }
    if (p_from->name == start) startPeak = p_from;
    if (p_to->name == end) goalPeak = p_to;
    (p_from->neighbors).push_back(p_to);
    edges.push_back(new Edge(p_from, p_to, cost));
}

cout << A_Star(edges, startPeak, goalPeak) << endl;
for (Edge *e : edges) {
    delete e;
}
for (Peak *p : peaks) {
    delete p;
}
return 0;
}

```

2.3. Результат работы программы

The screenshot shows a web browser window with the URL `https://ideone.com/3JpRVJ`. The code editor contains the following C++ code:

```

108.     p_to = new Peak(to);
109.     peaks.push_back(p_to);
110. }
111. if (p_from->name == start) startPeak = p_from;
112. if (p_to->name == end) goalPeak = p_to;
113. (p_from->neighbors).push_back(p_to);
114. edges.push_back(new Edge(p_from, p_to, cost));
115. }
116. greed(edges, startPeak, goalPeak);
117. cout << reconstruct(goalPeak) << endl;
118. return 0;
119. }

```

The output of the program is shown in the 'stdout' section:

```

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

```

The status bar at the bottom indicates 'Успешно #stdin #stdout 0s 15240KB'.

The screenshot displays the IDeone online C++ compiler interface. The browser tabs at the top include "Жадный алгоритм и A* - Ц...", "Диалоги", "PIAA_2019/Lab2Report.pdf", and several IDeone.com tabs. The address bar shows "https://ideone.com/msXgXQ". The code editor contains a C++ program for the A* algorithm, with line numbers 137 through 148 visible. The code includes a loop for processing edges and a final return statement. Below the code, the status bar indicates "Успешно #stdin #stdout 0s 15240KB" and "comments (0)". The input section, labeled "stdin", shows the following data: "a e", "a b 3.0", "b c 1.0", "c d 1.0", "a d 5.0", and "d e 1.0". The output section, labeled "stdout", shows the result: "ade". At the bottom of the interface, there is a footer for "Sphere Research Labs" and a Windows taskbar at the very bottom with the date "27.02.2019" and time "21:04".

3. Вывод

В результате работы программы был исследованы жадный алгоритм и A* и их отличия. Полученный результат удовлетворяет результатам данной лабораторной работы.