

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. 8304

Порывай П.А

Преподаватель

Размочаева Н.В

Санкт-Петербург

2020

Цель работы

Разработать 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 6

Реализация очереди с приоритетами, используемой в A^* , через двоичную кучу.

Описание алгоритма

В процессе работы алгоритма для вершин рассчитывается функция $f(v) = g(v) + h(v)$, где $g(v)$ - наименьшая стоимость пути в v из стартовой вершины, $h(v)$ - эвристическое приближение стоимости пути от v до конечной цели. Фактически, функция $f(v)$ - длина пути до цели, которая складывается из пройденного расстояния $g(v)$ и оставшегося расстояния $h(v)$. Исходя из этого, чем меньше значение $f(v)$, тем раньше мы откроем вершину v , так как через неё мы предположительно достигнем расстояние до цели быстрее всего. Открытые алгоритмом вершины хранятся в очереди с приоритетом по значению $f(v)$. Сложность алгоритма: $O(|v|*|v| + |k|)$, где v - множество вершин, k - множество ребер.

Описание основных структур данных и функций

`struct node_in_queue`

Узел в очереди с приоритетом на основе "Heap". Каждый новый узел помещается в "Heap",

С уменьшением приоритета

`struct node`

Узел для массива вершин, которые помещаем в очередь

`Class Heap`

Реализация "кучи", используется при выводе ответа

```
double m[100][100]
```

Матрица для хранения длин ребер

```
vector<node*> closedset
```

Вектор для хранения “открытых вершин”

```
vector<node*> openset
```

Вектор для “закрытых вершин”

```
void A(node* start, node* end, double m[100][100])
```

Основной алгоритм, записанный выше, функция здесь принимает матрицу с длинами ребер. Добавлен вывод промежуточных результатов

```
void printWay(Heap from, char start, char end)
```

Вывод ответа, использующий очередь с приоритетом, в Heap узлы просматриваются с конца, ключ curr меняется местом со значением h[i].neighbour , пока curr != start

Тестирование

В Таблице 1 не указаны промежуточные результаты, которые вывожу в программе.

Таблица 1 в формате(ввод/вывод)

| | |
|--|-----|
| a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0 | ade |
| a g a b 3.0 a c 1.0 b d 2.0 b e 3.0 d e 4.0 | ag |

| | |
|---|------|
| e a 1.0 e f 2.0 a g 8.0 f g 1.0 | |
| 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 | abef |
| a c a c 1.0 a b 2.0 b c 3.0 | ac |
| a b a b 1.0 a c 2.0 | ab |
| a c a b 2.0 b c 3.0 a m 1.0 m c 3.0 | amc |

Выводы

В ходе выполнения работы были получены навыки использования “ассоциативного массива”, “кучи”, написания алгоритма A* по псевдокоду.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
#include<iostream>
#include <string>
#include<vector>
#include<cstdlib>
#include<math.h>
#include<map>
#include<stack>
#include<cstdlib>
using namespace std;

struct node_in_queue {

    char curr;
    char neighbour;
    int prior;
};
struct node {

    char name;
    double G;
    double F;

};

class Heap;

void A(node start, node end, double m[100][100]);
void printWay(Heap from, char start, char end);

class Heap {
    static const int SIZE = 100; // максимальный размер кучи

    int HeapSize; // размер кучи
public:
    int get_HeapSize() {

        return HeapSize;
    }
    node_in_queue* h; // указатель на массив кучи
    Heap(); // конструктор кучи
    void addelem(node_in_queue); // добавление элемента кучи
    node_in_queue* print_heap_priority() {

        return h;
    }
};

Heap::Heap() {
    h = new node_in_queue[SIZE];
    HeapSize = 0;
}
```

```

void Heap::addelem(node_in_queue n) {
    bool is_in = false;

    int i, parent;
    i = HeapSize;
    h[i] = n;
    parent = (i - 1) / 2;
    while (parent >= 0 && i > 0) {
        if (h[i].prior > h[parent].prior) {
            node_in_queue temp = h[i];
            h[i] = h[parent];
            h[parent] = temp;
        }
        i = parent;
        parent = (i - 1) / 2;
    }
    HeapSize++;
}

void A(node* start, node* end, double m[100][100]) {

    double min_f = 9999999;
    node* min_node = nullptr;
    int min_i;
    vector<node*> closedset;
    vector<node*> openset;

    int prior_heap = 100;
    Heap fromset;

    start->G = 0;
    start->F = start->G + (end->name - start->name);
    openset.push_back(start);
    bool stop = false;
    while (openset.size() != 0) {

        for (int i = 0; i < openset.size(); i++) {
            if (openset[i]->F < min_f) { // =

                min_f = openset[i]->F;
                min_node = openset[i];
                min_i = i;

            }
            else if (openset[i]->F == min_f) {

                if (openset[i]->name > min_node->name) {
                    min_f = openset[i]->F;
                    min_node = openset[i];
                    min_i = i;
                }
            }
        }
    }
}

```

```

min_f = 99999999;

node* curr = min_node;

if (curr->name == end->name) {
    printWay(fromset, start->name, end->name);
    char c, n;
    c = fromset.h[fromset.get_HeapSize() - 1].curr;
    n = fromset.h[fromset.get_HeapSize() - 1].neighbour;
    //std::cout << c << n;
    return;
}

openset.erase(openset.begin() + min_i);
closedset.push_back(curr);

for (int i = 0; i < 100; i++)
    if (m[curr->name - 97][i] != 0) {
        bool cont = false;

        for (int j = 0; j < closedset.size(); j++)
            if (i + 97 == closedset[j]->name) //ВОЗМОЖНО

                cont = true;

        if (cont == true)
            continue;

        bool tentative_is_better;
        node* neighbour = nullptr;
        double tentative_g_score = m[curr->name - 97][i];

        bool in_openset = false;

        for (int j = 0; j < openset.size(); j++)
            if (i + 97 == openset[j]->name)
                in_openset = true;

        if (in_openset == false) {
            neighbour = new node;
            neighbour->name = i + 97;
            openset.push_back(neighbour);

            tentative_is_better = true;
        }
        else {
            for (int j = 0; j < openset.size(); j++)
                if (openset[j]->name == i + 97) {
                    neighbour = openset[j];
                }
            if (tentative_g_score < neighbour->G)
                tentative_is_better = true;
            else

```

ошибка

```

        tentative_is_better = false;
    }

    if (tentative_is_better == true) {
        for (int j = 0; j < openset.size(); j++)
            if (openset[j]->name == i + 97) {
                neighbour = openset[j];
            }
        neighbour->G = tentative_g_score;
        neighbour->F = neighbour->G + (end->name -
neighbour->name);
        node_in_queue node = { curr->name, neighbour-
>name, prior_heap };
        fromset.addelem(node);
        std::cout << "\n Вершины, которые идут в очередь\n";
        std::cout << curr->name << neighbour->name <<
"\n";
        prior_heap--;
    }
}

}

return;

}

void printWay(Heap from, char start, char end) {

    std::stack<char> stack;

    node_in_queue* h = from.h;
    int i = from.get_HeapSize()-1 ;//
    stack.push(end);
    char current = h[i].curr ;

    while (current != start) {

        if (current == h[i].neighbour) {

            stack.push(current);

            current = h[i].curr;
        }
        i--;
    }

    stack.push(start);

    while (!stack.empty()) {
        std::cout << stack.top();
    }
}

```



```

        stack.pop();

    }

    std::cout << "\n";

}

int main() {

    double m[100][100];

    for (int i = 0; i < 100; i++)
        for (int j = 0; j < 100; j++)
            m[i][j] = 0;

    char a, b;
    double c;
    char start, end;

    std::cin >> start >> end;

    while (std::cin >> a >> b >> c)
        m[a - 97][b - 97] = c;

    node* start1 = new node;
    node* end1 = new node;
    start1->name = start;
    end1->name = end;
    A(start1, end1, m);

}

```