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: Поиск кратчайшего пути на графе жадным алгоритмом и A*

Студентка гр. 1383 _____ Федорова О.В.

Преподаватель _____ Токарев А.П.

Санкт-Петербург

2023

Цель работы.

Познакомиться и реализовать алгоритмы поиска кратчайшего пути между двумя вершинами на графе. Для поиска использовать жадный алгоритм и A*.

Задание.

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

Пример выходных данных:

abcde

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

Пример выходных данных:

ade

Вариант 9

Добавить графическое представление графа

Выполнение работы.

Решение задачи жадным алгоритмом тривиально - обход графа, начиная со стартовой вершины, выбор самой выгодной (на данный момент) и переход в неё.

Решение задачи алгоритмом A*. Для ее решения были реализованы следующие функции и структуры:

Структура `Node` представляет вершину графа и содержит следующие поля:

- `key`: символ, соответствующий вершине.
- `g`: стоимость пути от начальной вершины до текущей вершины.
- `f`: оценочное общее расстояние от начальной вершины до целевой вершины, с учетом уже пройденного пути и эвристической функции.
- `tentative_is_better`: флаг, указывающий, является ли предлагаемый путь лучшим, чем текущий путь к вершине.
- `next`: отображение, хранящее связи с соседними вершинами и их стоимости.

Функции и процедуры:

- `h_x` - эвристическая функция, оценивающая расстояние от текущей вершины до целевой вершины.
- `get_table` - заполняет таблицу вершин графа на основе ввода пользователем.
- `min_f` - находит вершину с минимальным значением `f` из множества вершин `map`.
- `print_map` - выводит содержимое таблицы вершин графа.
- `print_path` - выводит кратчайший путь от начальной вершины до целевой вершины, используя информацию о предшествующих вершинах `from_set`.

- `a_star` - основная функция, реализующая алгоритм A*. Она выполняет поиск кратчайшего пути от начальной вершины `start` до целевой вершины `end`.

в функции `main` считываются начальная и конечная вершины, вызывается `get_table` для заполнения таблицы вершин графа, а затем вызывается `a_star` для поиска кратчайшего пути.

Суть работы алгоритма в том, что в функции a_star создаются три контенера для открытых вершин, обработанных вершин и вершин, которые занесены в путь. В начале инициализируются поля f и g стартовой вершины, далее начинается основной цикл работы в цикле while пока существуют открытые вершины. Создается указатель curr по которому можно будет двигаться по спискам вершин, на первом шаге цикла в списке открытых вершин находится одна - стартовая, но далее в open_set будут добавляться все ее соседние вершины, последовательно обрабатываться, удаляться после обработки и добавляться соседние с ними вершины.

В начале цикла происходит проверка на конец, если достигнута вершина end, то программа печатает путь до нее и выходит. В ином случае вершина считается обработанной и удаляется из списка open_set, переходя в список closed_set. Далее в цикле for происходит обработка всех соседних вершин, если вершина в списке обработанных вершин, но она является выигрышной для прохождения, то она становится потенциально следующей. Если вершины нет ни в каком списке, то ее надо добавить в from_set, если вершина уже есть в списке открытых и ее g меньше, чем потенциальная стоимость пути - эта вершина помечается лучшей.

Если текущая вершина помечена лучшей, то она добавляется в список from_set и ее g и f обновляются, после чего вершина тоже будет считаться обработанной.

Для выполнения задания по варианту использовалась библиотека SFML. Граф рисуется по заданным путям в файле в формате, совпадающим с вводом данных для задания(сначала 2 вершины - начальная и конечная, затем строчки : вершина - откуда вершина - куда число - стоимость). После считывания, программа считает количество вершин - каждая вершина будет являться вершиной правильного n-угольника, где n - количество вершин. В таком случае можно будет гарантировать, что весь граф будет виден и его можно будет нарисовать для любого количества вершин, так как вычисление координат вершин правильного многоугольника сводится к простым формулам линейной алгебры.

Вывод:

В ходе выполнения работы были изучены алгоритмы поиска кратчайших путей на графах, и были реализованы и протестированы жадный алгоритм и A*.

Приложение А

Исходный код программы

```
	#include <iostream>
#include <vector>
#include <map>
#include "math.h"
struct Node{
	char key = '0';
	double g = -1;
	double f = -1;
	bool tentative_is_better = false;
	std::map < Node *, double> next;
};

double h_x(Node * curr, char end) {
```

```
        return fabs((double)((int)(curr->key) - (int)end));
}
void get_table(std::map <char, Node * > & list){
        std::string asstr, aestr;
        std::string sym;
        for(int i = 0; i < 26; i++) {
        Node * new_node = new Node;
        new_node->key = (char )((int)('a') + i);
        list.insert({new_node->key, new_node});
        }
        double cst;
        while(std::cin) {
        std::cin>>asstr>>aestr>>cst;
        (list[asstr[0]]->next).insert({list[aestr[0]], cst});
        }
}

Node * min_f(std::map<char, Node *> map, char end) {
        double min = -1;
        Node * new_key;
        for(std::pair <char, Node *> i : map) {
        if(i.second->g != -1 && (min == -1 || min > i.second->f ||
        min == i.second->f && h_x(i.second, end) <= h_x(new_key, end))){
        min= i.second->f;
        new_key = i.second;
        }
        }
        return new_key;
}

void print_map(std::map <char, Node *> table) {
        for(auto i : table) {
        std::cout<<i.first<<" : " << i.second->key<<" cost " <<
i.second->g<<std::endl;
        }
}
void print_path(std::map <char, Node *> from_set, char start, char end) {
        std::string ans;
        ans += end;
        char curr = end;
        while(curr != start) {
        curr = from_set.find(curr)->second->key;
        ans += curr;
        }
        for(int i = ans.size() - 1; i >= 0; i--) {
```

```
		std::cout<<ans[i];
		}
}
void a_star(char start, char end, std::map <char, Node *> & table) {
		std::map <char, Node *> closed_set;
		std::map <char, Node *> open_set;
		std::map <char, Node *> from_set;
		open_set.insert({start, table.find(start)->second});
		table.find(start)->second->g = 0;
		table.find(start)->second->f = 0 + h_x(table.find(start)->second, end);
		while (!open_set.empty()) {
		Node * curr;
		curr = min_f(open_set, end);
		if(curr->key == end) {
		print_path(from_set, start, end);
		return;
		}
		open_set.erase(curr->key);
		closed_set.insert({curr->key, curr});
		for(auto neighbour  : curr->next) {
		double tentative_to_score = curr->g + curr->next[neighbour.first];
		if(closed_set.find(neighbour.first->key) != closed_set.end()) {//отличие от
псевдокода
		if(tentative_to_score <= neighbour.first->g || neighbour.first->g == -1)
			neighbour.first->tentative_is_better = true;
		else
			neighbour.first->tentative_is_better = false;
		}
		if(open_set.find(neighbour.first->key) == open_set.end() &&
		closed_set.find(neighbour.first->key) == closed_set.end()) {
		open_set.insert({neighbour.first->key ,neighbour.first});//отличие от
псевдокода, на сайте они проверяют лишь на наличие в open_set а надо еще и
в close_set
		neighbour.first->tentative_is_better = true;
		} else {
		if (tentative_to_score < neighbour.first->g || neighbour.first->g == -1) {
			neighbour.first->tentative_is_better = true;
		} else {
			neighbour.first->tentative_is_better = false;
		}
		}
		if(neighbour.first->tentative_is_better) {
		from_set[neighbour.first->key] = curr;
		neighbour.first->g = tentative_to_score;
		neighbour.first->f = tentative_to_score + h_x(neighbour.first, end);
```

```
            neighbour.first->tentative_is_better = false;
            }
            }
            }
            print_path(from_set, start, end);
}


int main() {
            std::string s_str, e_str;
            std::cin>>s_str>>e_str;
            char s = s_str[0], e = e_str[0];
            std::map <char, Node *> vec;
            get_table(vec);
            a_star(s, e, vec);
            return 0;
}
```