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*.

Студентка гр. 8382 _____ Наконечная А. Ю.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с алгоритмом A* и научиться применять его на практике. Написать программу реализовывающую поиск пути в графе.

Постановка задачи.

1) 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

2) 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

Индивидуальное задание.

Вариант 1.

В A* вершины именуются целыми числами (в т. ч. отрицательными).

Описание алгоритма A* (индивидуальное задание).

В первой строке на вход программе подаются начальная и конечная вершины. Далее в каждой строчке указываются рёбра графа и их вес.

Связи графа хранятся в контейнере STL map, в цикле while происходит заполнение info. Цикл прекратит работу в тот момент, когда очередной cin

вернёт false. Далее будут выведены значения графа, который ввёл пользователь.

В начале функции aStar находятся следующие данные. Для хранения пройденных и рассматриваемых вершин используются два последовательных контейнера list. Для хранения вершин, по которым в дальнейшем будет восстанавливаться путь, используется контейнер map. Также для хранения значений эвристической функции (h(x)), соседних вершин, стоимостей путей от начальной вершины (g(x)) и оценок f(x) для каждой вершины используется данный контейнер.

Сперва функция помещает начальную вершину в список open, подсчитывает значение эвристической функции для каждой вершины и находит g(x) + h(x) для начальной вершины.

Далее следует цикл while, который выполняет свою работу, пока список open не опустеет. Если такое произойдёт, то после выполнения цикла будет возвращено значение false, и программа выдаст сообщение о том, что путь не был найден.

В самом цикле while сначала выбирается вершина из списка open с минимальным значением функции f(x). Если вершина является конечной, то происходит вывод полученного пути и программа завершает работу. Иначе эта вершина добавляется в список рассмотренных, и далее происходит поиск соседних вершин. Каждая соседняя вершина проходит несколько проверок. Во-первых, если вершина уже была рассмотрена, то цикл переходит к следующей, иначе для данной вершины находится значение tmpG (g(x) для обрабатываемого соседа). Во-вторых, если вершина не находится в списке open, то её необходимо туда добавить и обновить значения g(x), f(x). Иначе, если tmpG меньше уже вычисленного ранее значения g(x) для рассматриваемого соседа, то необходимо обновить свойства вершины.

Сложность алгоритма A*.

Временная сложность. Сложность алгоритма зависит от эвристики. В худшем случае, число вершин, исследуемых алгоритмом, растёт экспоненциально, но сложность становится полиномиальной, когда эвристика удовлетворяет условию:

$| h(n) - h^*(n) | < O(\log h^*(n))$

h(n) — расчётное расстояние;

h*(n) — точное расстояние.

В данном случае, эвристика не является наилучшей, т. к. эвристическая функция не моделирует перемещение по поверхности (евклидово расстояние), что позволило бы найти точное расстояние между двумя вершинами.

Память. При более точной эвристической функции сложность по памяти может составлять O(|E|+|V|), т. к. есть возможность сразу построить правильный путь без возвращения к предыдущим вершинам.

В ином случае, каждый шаг будет неверным и придётся просматривать каждое ребро графа. Тогда в худшем случае сложность может оказаться экспоненциальной.

Описание структур данных.

std::map<int, std::map<int, float>> info — контейнер STL map, хранит информацию о вершинах и их рёбрах.

std::list <int> close — контейнер STL list, хранит вершины, которые были рассмотрены в процессе выполнения алгоритма.

std::list <int> open — контейнер STL list, хранит вершины, которые должны быть рассмотрены.

std::map <int, int> from — контейнер STL map, необходим для сохранения пути от конца до начала и последующего его восстановления.

std::vector <int> N — контейнер STL vector, хранит вершины соседние с рассматриваемой в данный момент.

std::map <int, float> G — контейнер STL map, необходим для хранения оценки расстояний от начальной вершины.

std::map <int, float> H - контейнер STL map, хранит значения эвристической функции.

std::map <int, float> F - контейнер STL map, хранит значения f(x).

Описание функций.

std::map <int, float> heuristic(int minPeak, int maxPeak, int end) — функция, оценивающая расстояния до цели. int minPeak — минимальная вершина в графе; int maxPeak — максимальная вершина в графе; int end — конечная вершина. Функция возвращает контейнер, заполненный оценками от соответствующей вершины до цели.

int minF(std::list <int> open, std::map <int, float> F) — функция находит минимальное значение f(x) среди вершин, которые доступны для рассмотрения. std::list <int> open — список вершин, которые можно рассматривать; std::map <int, float> F — контейнер, хранящий значения f(x). Функция возвращает вершину с минимальным значением f(x).

std::vector <int> neighbours(std::map <int, std::map <int, float>> info, int curr, int minPeak, int maxPeak) — функция находит всех соседей для рассматриваемой вершины. std::map <int, std::map <int, float>> info — контейнер, хранящий вершины и рёбра графа; int curr — рассматриваемая вершина; int minPeak — минимальная вершина; int maxPeak — максимальная вершина. Функция возвращает контейнер, заполненный соседними вершинами.

bool inList(std::list <int> l, int currN) — функция проверяет, находится ли вершина в списке. std::list <int> l — рассматриваемый список; int currN —

проверяемая вершина. Функция возвращает true, если вершина в списке, в другом случае будет возвращено false.

void reconstruction(std::map <int, int> from, int begin, int end) — функция реконструирует путь и выводит его на экран. std::map <int, int> from — контейнер, который хранит в себе путь; int begin — начальная вершина; int end - конечная вершина.

void printList(std::list <int> l) — выводит содержимое списка на экран. std::list <int> l — список, чьё содержимое будет выведено.

bool aStar(std::map <int, std::map <int, float>> info, int begin, int end, int minPeak, int maxPeak) — функция, которая осуществляет сам алгоритм A*. std::map <int, std::map <int, float>> info — контейнер, хранящий вершины и рёбра графа; int begin — начальная вершина; int end — конечная вершина; int minPeak — минимальная вершина; int maxPeak — максимальная вершина. Функция возвращает true, если путь был найден. В другом случае будет возвращено false.

Тестирование.

№ теста	Тест	Результат
1	1 4 1 2 1 2 3 2 3 4 3	Из вершины: 1 в вершину: 2 ребро: 1 Из вершины: 2 в вершину: 3 ребро: 2 Из вершины: 3 в вершину: 4 ребро: 3 Значения эвристической функции: 3 2 1 0 Текущая вершина: 1 Рассматриваемые вершины: 1 Пройденные вершины:

		Стоимости путей от начальной вершины: Вершина: 1 g(x): 0 Вершина: 1 f(x): 3 _____ Вершины соседние с 1 2 Текущая вершина: 2 Рассматриваемые вершины: 2 Пройденные вершины: 1 Стоимости путей от начальной вершины: Вершина: 1 g(x): 0 Вершина: 2 g(x): 1 Вершина: 1 f(x): 3 Вершина: 2 f(x): 3 _____ Вершины соседние с 2 3 Текущая вершина: 3 Рассматриваемые вершины: 3 Пройденные вершины: 2 1 Стоимости путей от начальной вершины: Вершина: 1 g(x): 0 Вершина: 2 g(x): 1

		Вершина: 3 g(x): 3 Вершина: 1 f(x): 3 Вершина: 2 f(x): 3 Вершина: 3 f(x): 4 _____ Вершины соседние с 3 4 Текущая вершина: 4 Рассматриваемые вершины: 4 Пройденные вершины: 3 2 1 Стоимости путей от начальной вершины: Вершина: 1 g(x): 0 Вершина: 2 g(x): 1 Вершина: 3 g(x): 3 Вершина: 4 g(x): 6 Вершина: 1 f(x): 3 Вершина: 2 f(x): 3 Вершина: 3 f(x): 4 Вершина: 4 f(x): 6 _____ Результат: 1234
2	-1 -5 -1 -2 18 -1 -3 4	Из вершины: -4 в вершину: -5 ребро: 1 Из вершины: -2 в вершину: -4 ребро: 3 Из вершины: -1 в вершину: -4 ребро: 2

	-1 -4 2 -2 -4 3 -4 -5 1	Из вершины: -1 в вершину: -3 ребро: 4 Из вершины: -1 в вершину: -2 ребро: 18 Значения эвристической функции: 0 1 2 3 4 Текущая вершина: -1 Рассматриваемые вершины: -1 Пройденные вершины: Стоимости путей от начальной вершины: Вершина: -1 g(x): 0 Вершина: -1 f(x): 4 _____ Вершины соседние с -1 -4 -3 -2 Текущая вершина: -4 Рассматриваемые вершины: -2 -3 -4 Пройденные вершины: -1 Стоимости путей от начальной вершины: Вершина: -4 g(x): 2 Вершина: -3 g(x): 4 Вершина: -2 g(x): 18 Вершина: -1 g(x): 0 Вершина: -4 f(x): 3 Вершина: -3 f(x): 6

		Вершина: -2 f(x): 21 Вершина: -1 f(x): 4 _____ Вершины соседние с -4 -5 Текущая вершина: -5 Рассматриваемые вершины: -5 -2 -3 Пройденные вершины: -4 -1 Стоимости путей от начальной вершины: Вершина: -5 g(x): 3 Вершина: -4 g(x): 2 Вершина: -3 g(x): 4 Вершина: -2 g(x): 18 Вершина: -1 g(x): 0 Вершина: -5 f(x): 3 Вершина: -4 f(x): 3 Вершина: -3 f(x): 6 Вершина: -2 f(x): 21 Вершина: -1 f(x): 4 _____ Результат: -1-4-5
3	-1 -5 -1 -2 7 -1 -3 3	Из вершины: -3 в вершину: -4 ребро: 8 Из вершины: -2 в вершину: -5 ребро: 4 Из вершины: -2 в вершину: -3 ребро: 1

	-2 -3 1 -3 -4 8 -2 -5 4	Из вершины: -1 в вершину: -3 ребро: 3 Из вершины: -1 в вершину: -2 ребро: 7 Значения эвристической функции: 0 1 2 3 4 Текущая вершина: -1 Рассматриваемые вершины: -1 Пройденные вершины: Стоимости путей от начальной вершины: Вершина: -1 g(x): 0 Вершина: -1 f(x): 4 _____ Вершины соседние с -1 -3 -2 Текущая вершина: -3 Рассматриваемые вершины: -2 -3 Пройденные вершины: -1 Стоимости путей от начальной вершины: Вершина: -3 g(x): 3 Вершина: -2 g(x): 7 Вершина: -1 g(x): 0 Вершина: -3 f(x): 5 Вершина: -2 f(x): 10 Вершина: -1 f(x): 4

		_____ Вершины соседние с -3 -4 Текущая вершина: -2 Рассматриваемые вершины: -4 -2 Пройденные вершины: -3 -1 Стоимости путей от начальной вершины: Вершина: -4 g(x): 11 Вершина: -3 g(x): 3 Вершина: -2 g(x): 7 Вершина: -1 g(x): 0 Вершина: -4 f(x): 12 Вершина: -3 f(x): 5 Вершина: -2 f(x): 10 Вершина: -1 f(x): 4 _____ Вершины соседние с -2 -5 -3 Текущая вершина: -5 Рассматриваемые вершины: -5 -4 Пройденные вершины: -2 -3 -1 Стоимости путей от начальной вершины: Вершина: -5 g(x): 11

		Вершина: -4 g(x): 11 Вершина: -3 g(x): 3 Вершина: -2 g(x): 7 Вершина: -1 g(x): 0 Вершина: -5 f(x): 11 Вершина: -4 f(x): 12 Вершина: -3 f(x): 5 Вершина: -2 f(x): 10 Вершина: -1 f(x): 4 _____ Результат: -1-2-5

Описание жадного алгоритма.

Считывание необходимой информации происходит также, как и в алгоритме A* (см. Описание алгоритма A*).

В начале функции greedy находятся следующие данные. Для хранения пройденных и рассматриваемых вершин используются два последовательных контейнера list. Для хранения вершин, по которым в дальнейшем будет восстанавливаться путь, используется контейнер map. Также в начале присутствует контейнер vector, который в дальнейшем будет хранить соседние вершины.

В цикле while происходит следующее. Пока список рассматриваемых вершин не пуст, находится вершина, в которую идёт минимальное ребро из последней посещённой вершины. Если такая вершина является конечной, то происходит вывод пути. В другом случае, вершина добавляется в список рассмотренных и находятся все её соседи. Если вершина не была рассмотрена, то она добавляется в список open. В следующем условии

проверяется случай, когда у вершины нет соседей. Если это не так, то переменная previous сохраняет текущую вершину и переходит к следующей. В другом случае, все рёбра с такой вершиной будут обнулены, а значение previous возвращается к предыдущей вершине. Иначе говоря, поиск продолжится, как только получится выйти из тупика.

Сложность жадного алгоритма.

Временная сложность. На каждом шаге цикла просматриваются все рёбра, выходящие из вершины. Таким образом, в худшем случае будет рассмотрено O(|V|*|E|). То есть, придётся пройти по всем вершинам и просмотреть все рёбра.

Память. Сложность по памяти O(|E|+|V|), так как в контейнере хранятся все рёбра и вершины.

Описание структур данных.

std::map<int, std::map<int, float>> info — контейнер STL map, хранит информацию о вершинах и их рёбрах.

std::list <int> close — контейнер STL list, хранит вершины, которые были рассмотрены в процессе выполнения алгоритма.

std::list <int> open — контейнер STL list, хранит вершины, которые должны быть рассмотрены.

std::map <int, int> from — контейнер STL map, необходим для сохранения пути от конца до начала и последующего его восстановления.

std::vector <int> N — контейнер STL vector, хранит вершины соседние с рассматриваемой в данный момент.

Описание функций.

char min(std::map <char, std::map <char, float>> info, std::list <char> open, char previous) — функция находит вершину, в которую идёт минимальное ребро. std::map <char, std::map <char, float>> info, std::list <char> open (см. Описание структур данных.), char previous — вершина, из которой выходит минимальное ребро. Возвращается вершина, в которую идёт минимальное ребро.

std::vector <char> neighbours(std::map <char, std::map <char, float>> info, char curr, char minPeak, char maxPeak) — функция находит всех соседей для рассматриваемой вершины. std::map <char, std::map <char, float>> info — контейнер, хранящий вершины и рёбра графа; char curr — рассматриваемая вершина; char minPeak — минимальная вершина; char maxPeak — максимальная вершина. Функция возвращает контейнер, заполненный соседними вершинами.

bool inList(std::list <char> l, char currN) — функция проверяет, находится ли вершина в списке. std::list <char> l — рассматриваемый список; char currN — проверяемая вершина. Функция возвращает true, если вершина в списке, в другом случае будет возвращено false.

void reconstruction(std::map <char, char> from, char begin, char end) — функция реконструирует путь и выводит его на экран. std::map <char, char> from — контейнер, который хранит в себе путь; char begin — начальная вершина; char end - конечная вершина.

bool greedy(std::map <char, std::map <char, float>> info, char begin, char end, char minPeak, char maxPeak) — функция, которая осуществляет сам жадный алгоритм. std::map <char, std::map <char, float>> info — контейнер, хранящий вершины и рёбра графа; char begin — начальная вершина; char end — конечная вершина; char minPeak — минимальная вершина; char maxPeak — максимальная вершина. Функция возвращает true, если путь был найден. В другом случае будет возвращено false.

Тестирование.

№ теста	Тест	Результат
1	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	abdefg
2	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	abcd
3	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

Выводы.

В ходе выполнения лабораторной работы были изучены и применены на практике жадный алгоритм и алгоритм A*.

ПРИЛОЖЕНИЕ А

Исходный код программы

a_star_int.cpp

```
#include <iostream>
#include <map>
#include <list>
#include <vector>

std::map <int, float> heuristic(int minPeak, int maxPeak, int end){
//подсчёт значений эвристической функции
    std::map <int, float> H;
    std::cout << "Значения эвристической функции: " << std::endl;
    for (int i = minPeak; i <= maxPeak; i++){
        if (i > end){ //подсчитывается близость вершин к конечной вершине
            H[i] = i - end;
        }
        else{
            H[i] = end - i;
        }
        std::cout << H[i] << " ";
    }
    std::cout << std::endl;
    return H;
}

int minF(std::list <int> open, std::map <int, float> F){//поиск минимального
значения f(x)
    int res = open.back();
    float min = F[res];
    open.pop_back();

    while (!open.empty()){
        if (F[open.back()] <= min){
            res = open.back();
            min = F[open.back()];
        }
        open.pop_back();
    }
    return res;
}

std::vector <int> neighbours(std::map <int, std::map <int, float>> info, int
curr, int minPeak, int maxPeak){ //поиск соседей
    std::vector <int> N;
    std::cout << "Вершины соседние с " << curr << std::endl;
    for (int i = minPeak; i <= maxPeak; i++){
        if (info[curr][i]){ //если существует связь между вершинами
```

```
            N.push_back(i);
            std::cout << i << " ";
        }
    }
    std::cout << std::endl;
    return N;
}

bool inList(std::list <int> l, int currN){ //проверка на присутствие вершины
в списке
    while(!l.empty()){
        if (currN == l.back()){
            return true;
        }
        l.pop_back();
    }
    return false;
}

void reconstruction(std::map <int, int> from, int begin, int end){
//реконструкция пути
    int curr = end;
    std::list <int> path;
    path.push_front(curr);
    while(curr != begin){
        curr = from[curr];
        path.push_front(curr);
    }
    std::cout << "Результат: " << std::endl;
    for (auto it : path){
        std::cout << it;
    }
    std::cout << std::endl;
}

void printList(std::list <int> l){
    while(!l.empty()){
        std::cout << l.back() << " ";
        l.pop_back();
    }
    std::cout << std::endl;
}

bool aStar(std::map <int, std::map <int, float>> info, int begin, int end,
int minPeak, int maxPeak){
    std::list <int> close; //список пройденных вершин
    std::list <int> open = {begin}; //список рассматриваемых вершин
    std::map <int, int> from; //используется для восстановления пути
    std::vector <int> N; //контейнер с соседями
```

```
    std::map <int, float> G; //хранит стоимости путей от начальной вершины
    G[begin] = 0;
     std::map <int, float> H = heuristic(minPeak, maxPeak, end); //контейнер
с значениями эвристической функции
    std::map <int, float> F; //оценки f(x) для каждой вершины
    F[begin] = G[begin] + H[begin];

    while(!open.empty()) {
        int curr = minF(open, F);

        std::cout << "Текущая вершина: " << curr << std::endl;
        std::cout << "Рассматриваемые вершины: " << std::endl;
        printList(open);
        std::cout << "Пройденные вершины: " << std::endl;
        printList(close);
        std::cout << "Стоимости путей от начальной вершины: " << std::endl;
        for (auto it : G){
             std::cout << "Вершина: " << it.first << " g(x): " << it.second
<< std::endl;
        }
        for (auto it : F){
             std::cout << "Вершина: " << it.first << " f(x): " << it.second
<< std::endl;
        }
        std::cout << "_____________________________" << std::endl;
        std::cout << std::endl;

        if (curr == end){ //если дошли до конца
            reconstruction(from, begin, end);
            return true; //путь найден
        }

        open.remove(curr);
        close.push_back(curr);

        N = neighbours(info, curr, minPeak, maxPeak);
        for (auto it : N){
            if (inList(close, it)){ //если вершина была пройдена
                continue;
            }
            bool tmpFlag; //обновление свойств текущего соседа
               float tmpG = G[curr] + info[curr][it]; //вычисление g(x) для
обрабатываемого соседа
                  if (!inList(open, it)){ //если сосед текущей вершины не
находится в списке рассматриваемых вершин
                open.push_back(it);
                tmpFlag = true;
            }
            else{
```

```
                if (tmpG < G[it]){ //если вычисленная стоимость меньше
                    tmpFlag = true;
                }
                else{
                    tmpFlag = false;
                }
            }
            if (tmpFlag){ //обновление свойств соседа
                from[it] = curr;
                G[it] = tmpG;
                F[it] = G[it] + H[it];
            }
        }
    }
    return false; //путь не найден
}

int main(){
    int out = 0, in = 0, begin = 0, end = 0, maxPeak = 0, minPeak = 0;
    float weight = 0;

    std::cin >> begin;
    std::cin >> end;

    std::map<int, std::map<int, float>> info; //контейнер хранит связи графа

    int k = 0;
    while (std::cin) { //Для прекращения работы цикла следует нажать ctrl +
D

        std::cin >> out;
        std::cin >> in;
        std::cin >> weight;

        if (weight < 0){
            std::cout << "Ошибка: вес не может быть отрицательным";
            return 0;
        }

        if (minPeak == 0) { //инициализируем минимальную вершину
            minPeak = begin;
        }
        if (maxPeak == 0) { //инициализируем максимальную вершину
            maxPeak = end;
        }
             if (in > maxPeak) { //если вершина in или out больше чем
максимальная, то она становится максимальной
            maxPeak = in;
        }
```

```
        if (out > maxPeak) {
            maxPeak = out;
        }
             if (in < minPeak) { //если вершина in или out меньше чем
минимальная, то она становится минимальной
            minPeak = in;
        }
        if (out < minPeak) {
            minPeak = out;
        }

        info[out][in] = weight;
        k++;
    }

    std::cout << std::endl;
    for(int i = minPeak; i <= maxPeak; i++){
        for(int j = minPeak; j <= maxPeak; j++) {
            if (info[i][j])
                std::cout << "Из вершины: " << i << " в вершину: " << j << "
ребро: " << info[i][j] << std::endl;
        }
    }
    std::cout << std::endl;

    bool res = aStar(info, begin, end, minPeak, maxPeak);

    if (!res){
        std::cout << "Ошибка: путь не был найден\n";
    }

    return 0;
}
```

ПРИЛОЖЕНИЕ А

Исходный код программы

greedy.cpp

```
#include <iostream>
#include <map>
#include <list>
#include <vector>

char min(std::map <char, std::map <char, float>> info, std::list <char>
open, char previous){
    char res = previous;
    float min = 1000;

    while (!open.empty()){
            if (info[previous][open.front()] < min && info[previous]
[open.front()]){
            res = open.front();
            min = info[previous][open.front()];
        }
        open.erase(open.begin());
    }
    return res;
}

std::vector <char> neighbours(std::map <char, std::map <char, float>>
info, char curr, char minPeak, char maxPeak){
    std::vector <char> N;
    for (char i = minPeak; i <= maxPeak; i++){
        if (info[curr][i]){
            N.push_back(i);
        }
    }
    return N;
}

bool inList(std::list <char> l, char currN){
    while(!l.empty()){
        if (currN == l.back()){
            return true;
        }
        l.pop_back();
    }
    return false;
}

void reconstruction(std::map <char, char> from, char begin, char end){
    char curr = end;
```

```
    std::list <char> path;
    path.push_front(curr);
    while(curr != begin){
        curr = from[curr];
        path.push_front(curr);
    }
    for (auto it : path){
        std::cout << it;
    }
}

bool greedy(std::map <char, std::map <char, float>> info, char begin,
char end, char minPeak, char maxPeak){
    std::list <char> close;
    std::list <char> open = {begin};
    std::map <char, char> from;
    std::vector <char> N;
    char previous = begin;

    while(!open.empty()) {
        char curr = min(info, open, previous);
        if (curr == end){
            reconstruction(from, begin, end);
            return true;
        }
        open.remove(curr);
        close.push_back(curr);
        N = neighbours(info, curr, minPeak, maxPeak);
        for (auto it : N){
            if (inList(close, it)){
                continue;
            }
            if (!inList(open, it)){
                open.push_back(it);
            }
            from[it] = curr;
        }
        if (!N.empty()) {
            previous = curr;
        }
        else{
            for (char i = minPeak; i <= maxPeak; i++){
                if (info[i][curr]){
                    previous = i;
                }
                info[i][curr] = 0;
            }
        }
    }
```

```
    return false;
}

int main(){
    char out = 0, in = 0, begin = 0, end = 0, maxPeak = 0, minPeak = 0;
    float weight = 0;

    std::cin >> begin;
    std::cin >> end;

    std::map<char, std::map<char, float>> info;

    int k = 0;
    while (std::cin) {
        std::cin >> out;
        std::cin >> in;
        std::cin >> weight;

        if (minPeak == 0) {
            minPeak = begin;
        }
        if (maxPeak == 0) {
            maxPeak = end;
        }
        if (in > maxPeak) {
            maxPeak = in;
        }
        if (out > maxPeak) {
            maxPeak = out;
        }
        if (in < minPeak) {
            minPeak = in;
        }
        if (out < minPeak) {
            minPeak = out;
        }

        info[out][in] = weight;
        k++;
    }

    bool res = greedy(info, begin, end, minPeak, maxPeak);

    return 0;
}
```