**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

# **по лабораторной работе №2**

**по дисциплине «Постороение и анализ алгоритмов»**

**Тема: «Жадный алгоритм и А\*»**

**Вариант 3c**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7382 |  | Гиззатов А.С. |
| Преподаватель |  | Фирсов М.А. |

**Санкт-Петербург**

**2019**

**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

abcde

Разработайте программу, которая решает задачу построения кратчайшего пути в ориентированном графе методом А\*. 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

Вар. 3c. Списки смежности. Написать функцию, проверяющую эвристику на допустимость и монотонность.

**Пояснение задания.**

На вход программе подается граф. Программа находит наименьший путь при помощи жадного алгоритма и алгоритма A\*.

**Описание алгоритмов.**

Жадный алгоритм:

Сначала записывается текущее значение cur как начальное, потом в текущий путь записываем начальное значение cur. После циклически проверяется на тупик и если он есть, то вершина записывается в dead\_locks , а за текущее значение принимается предыдущее концу. Если же тупика нет то проверяем есть ли вершина в списках dead\_locks и points\_hist. Если ее нет в обоих списках то она становится минимальной

**Описание структур**

Класс Graph.

**Описание функций**

В программе использовалось 1 класс и 5 методов:

* **void add\_way(char from, char to, double length)**

Добавление путя в список.

* **int heuristic(char a,char b)**

Эвристическая функция.

* **double find\_way\_cost(char from, char to)**

Нахождение веса ребра между двумя вершинами

* **bool has\_any\_way(std::list<char> &points\_hist, std::list<char> &deadlocks, char point)**

Проверка существует ли путь.

* **void find\_min\_way(char from, char to)**

Метод нахождения минимального пути в графе, и выписывания его в терминал.

* **int main()**

Главная функция программы. Считывает входные данные, запускает алгоритм поиска минимального пути.

В алгоритме А\* были использованы такие же методы.

**Тестирование.**

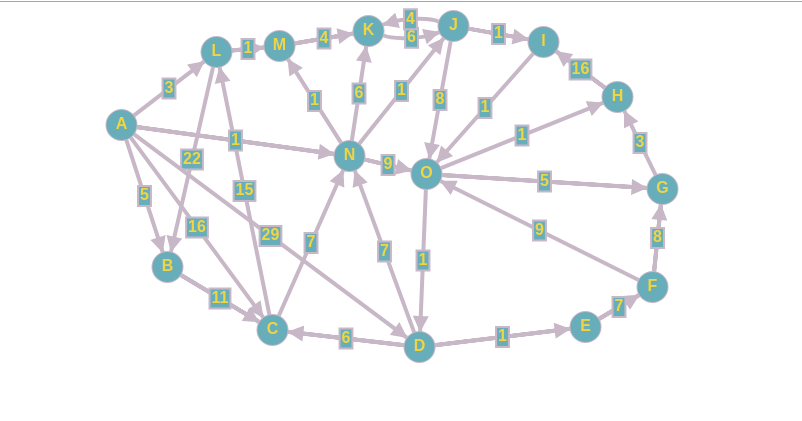


Рис.1 - Тестовый граф.

|  |  |
| --- | --- |
| Вводимые данные | Результат: |
| a e a b 3.0 b c 1.0 c d 1.0 a d 5.0 d e 1.0 | ade |
| a m  a l 3  a b 5  l m 1  l b 22  c l 15  b c 11  m k 4  k j 6  j o 8  n o 9  n m 1  c n 7  d c 6  d n 7  o d 1  j o 8  d e 1  e f 7  f g 8  g h 3  h i 16  o h 1  o g 5  f o 9  j i 1  i o 1  a n 1  a d 29  n k 6  j k 4  n j 1 | anjiodefghiodclm |
| a k  a l 3  a b 5  l m 1  l b 22  c l 15  b c 11  m k 4  k j 6  j o 8  n o 9  n m 1  c n 7  d c 6  d n 7  o d 1  j o 8  d e 1  e f 7  f g 8  g h 3  h i 16  o h 1  o g 5  f o 9  j i 1  i o 1  a n 1  a d 29  n k 6  j k 4  n j 1 | anjiodefghiodclmk |
| a o  a l 3  a b 5  l m 1  l b 22  c l 15  b c 11  m k 4  k j 6  j o 8  n o 9  n m 1  c n 7  d c 6  d n 7  o d 1  j o 8  d e 1  e f 7  f g 8  g h 3  h i 16  o h 1  o g 5  f o 9  j i 1  i o 1  a n 1  a d 29  n k 6  j k 4  n j 1 | anjio |

**Вывод.**

В процессе выполнения лабораторной работы были изучены алгоритм А\* и жадный алгоритм. Также был освоена работа с эвристической функцией.

**Приложение 1. Код программы.**

**#include <iostream>**

**#include <vector>**

**struct min\_sq {**

**short x;**

**short y;**

**short size;**

**};**

**class Square {**

**short \*\*sq;**

**short size;**

**short min;**

**short x;**

**short y;**

**int min\_count;**

**std::vector<min\_sq> final\_sq;**

**public:**

**unsigned long long iter;**

**Square(short n) {**

**size=n;**

**sq= new short\*[n]; //memmory allocation**

**for(short i=0; i<n; i++) {**

**sq[i]=new short[n];**

**for(short j=0; j<n; j++) {**

**sq[i][j]=0;**

**}**

**}**

**final\_sq.resize(0);**

**x=0;**

**y=0;**

**min=20;**

**min\_count = 0;**

**iter = 0;**

**}**

**bool find\_zero(short fst\_size) { //finding zero element**

**for(short i=size-fst\_size; i<size; i++) {**

**for(short j=size-fst\_size; j<size; j++) {**

**if(sq[i][j]==0) {**

**y=i;**

**x=j;**

**return true;**

**}**

**}**

**}**

**x=-1;**

**y=-1;**

**return false;**

**}**

**void set\_sq(short x,short y, short size) { //filling the square with size**

**for(short i=y; i<y+size; i++) {**

**for(short j=x; j<x+size; j++) {**

**sq[i][j]=size;**

**iter++;**

**}**

**}**

**}**

**void to\_zero(short x, short y, short sz) { //filling square with 0**

**for(short i=y; i<y+sz && i<size; i++) {**

**for(short j=x; j<x+sz && j<size; j++)**

**sq[i][j]=0;**

**}**

**}**

**short max\_length(short x, short y) { //finding maximum length before the end of the square**

**short j=1;**

**for(; j+x<size && j+y<size; j++) {**

**for(short i=y; i-y<=j; i++) {**

**if(sq[i][x+j]!=0)**

**return j;**

**}**

**for(short i=x; i-x<=j; i++) {**

**if(sq[y+j][i]!=0)**

**return j;**

**}**

**}**

**if(j==size)**

**return j-1;**

**else**

**return j;**

**}**

**void print() {**

**for(short i=0; i<size; i++) {**

**for(short j=0; j<size; j++)**

**std::cout<<sq[i][j]<<' ';**

**std::cout<<std::endl;**

**}**

**std::cout << std:: endl;**

**}**

**short find\_min(short fst\_size,short i,short j,short k){ //part of the algorithm**

**int count;**

**for(short ii=size-1; ii>=size-fst\_size; ii--) {**

**for(short jj=size-1; jj>=size-fst\_size; jj--) {**

**// print();**

**for(short kk=max\_length(jj, ii); kk>0; kk--) {**

**set\_sq(0,0,fst\_size);**

**count=3;**

**if(sq[i][j]==0) {**

**count++;**

**set\_sq(j,i,k);**

**}**

**if(sq[ii][jj]==0 && max\_length(jj,ii)==kk) {**

**set\_sq(jj,ii,kk);**

**count++;**

**}**

**while(count < min && find\_zero(fst\_size)) {**

**set\_sq(x,y,max\_length(x,y));**

**count++;**

**}**

**if(count==min)**

**//print();**

**min\_count++;**

**if(count<min && x==-1 && y==-1) { //if we found square with**

**print();**

**min=count;**

**save();**

**min\_count=1;**

**set\_sq(fst\_size,0,max\_length(fst\_size,0));**

**set\_sq(0,fst\_size,max\_length(0,fst\_size));**

**continue;**

**}**

**if(size%2!=0)**

**to\_zero(size-fst\_size,size-fst\_size,fst\_size);**

**if(size%2==0)**

**to\_zero(size/2,size/2,size/2);**

**//iter++;**

**}**

**}**

**}**

**return fst\_size;**

**}**

**void to\_split() { //part of the algorithm for searching minimal squares**

**short count, fst\_size;**

**if(size % 2 ==0)**

**fst\_size=size/2;**

**else if(size % 3 == 0)**

**fst\_size=size\*2/3;**

**else if(size % 5 == 0)**

**fst\_size = size\*3/5;**

**else**

**fst\_size= size/2+1;**

**set\_sq(fst\_size,0,max\_length(fst\_size,0));**

**set\_sq(0,fst\_size,max\_length(0,fst\_size));**

**for(short i=size-fst\_size; i<size-(size-fst\_size)/2; i++) {**

**for(short j=size-fst\_size; j<size; j++) {**

**for(short k=max\_length(j, i); k>0; k--) {**

**//print();**

**fst\_size = find\_min(fst\_size,i,j,k);**

**}**

**}**

**}**

**std::cout<<min<<std::endl;**

**print\_res();**

**}**

**void save() { //saving the square with minimal squares**

**int count=0;**

**final\_sq.resize(min);**

**for(short i=0; i<size; i++) {**

**for(short j=0; j<size; j++) {**

**if(sq[i][j]!=0) {**

**final\_sq[count].x=j+1;**

**final\_sq[count].y=i+1;**

**final\_sq[count].size=sq[i][j];**

**to\_zero(j,i,sq[i][j]);**

**count++;**

**}**

**}**

**}**

**}**

**void print\_res() { //printing the resulting combinations of the squares**

**for(short i=0; i<final\_sq.size(); i++) {**

**std::cout<<final\_sq[i].y<<' '<<final\_sq[i].x<<''<<final\_sq[i].size<<std::endl;**

**}**

**}**

**~Square() {**

**for(int i=0; i<size; i++)**

**delete[] sq[i];**

**delete sq;**

**}**

**};**

**int main() {**

**int n;**

**std::cout << "Введите размер стороны столешницы" <<std::endl;**

**std::cin >> n;**

**if(n==1) {**

**std::cout<<1<<std::endl<<"1 1 1\n";**

**return 0;**

**}**

**Square sq(n);**

**sq.to\_split();**

**std::cout << sq.iter<< "-количество итераций" << std::endl;**

**return 0;**

**}**