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: Алгоритмы на графах

Студентка гр. 7383 _____ Ханова Ю.А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2019

Цель работы

Исследовать и реализовать задачу построения кратчайшего пути в ориентированном графе помощью метода A*.

Формулировка задачи: необходимо разработать программу, которая решает задачу построения кратчайшего пути в ориентированном графе методом A* до заданной вершины. 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.

Входные данные: в первой строчке через пробел указываются начальная и две конечные вершины. Далее в каждой строке указываются ребра графа и их вес. Вариант 1м: Матрица смежности. В A* вершины именуются целыми числами (в т. ч. отрицательными).

Реализация задачи

В данной работе для решения поставленной цели был написан класс `Graph` и несколько методов, содержащихся в данном классе. А также написана структура для очереди с приоритетом.

Параметры, хранящиеся в структуре данных `struct Vertex`

- `prior` – приоритет нахождения в очереди;
- `double evr` – разница между значениями вершин;
- `vector <int> res` – путь в индексном виде.

Конструктор класса создает двумерный массив целых чисел, заполняет его нулями, добавляет в вектор `way` начало пути.

Ниже представлены поля класса:

`double **matrix` — матрица смежности графа.

`vector<int> way` — вектор, хранящий кратчайший путь.

`size_t size` — размер матрицы смежности.

Далее представлены методы класса:

`void A_star(vector<int>&vertex,int start, int finish, priority_queue<Vertex>&p_queue, double dist);` - основной метод, выполняющий алгоритм поиска пути, в нем в очередь

с приоритетом добавляются пути и вершины и сравниваются для поиска оптимального решения.

`void Print();` - печатает итоговый найденный путь

`double Evristic(vector<int> vert,int i, int j);` - ищет эвристику (сравнивает соответствующие элементы вектора вершин.

`void New_matrix(int vert, vector<int>&vertex, vector<int>&v, vector<int>&u, vector<double>&w);` - сопоставляет входные данные с матрицей смежности, заполняя ее существующими путями.

`int Find_i(vector<int>&vert, int f);` - ищет элемент в данном векторе и возвращает его индекс.

`bool operator < (const Vertex &v1, const Vertex &v2);` - компаратор для очереди с приоритетом.

В главной функции `main()` создается класс для графа и считывается начальная и конечная вершины. Долее в цикле считываются данные из какой вершины в какую вершину есть путь определенной длины, данные записываются в вектора и вызывается метод, заполняющий матрицу. Вызывается метод поиска кратчайшего пути алгоритмом A*.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Так же было проведено исследование алгоритма. Функция `A_star` проходит по смежным ребрам вершины с наименьшей эвристической функцией. В худшем случае могут быть просмотрены все пути данного графа. Тогда сложность зависит от количества ребер и количества вершин графа. В таком случае временную сложность алгоритма можно свести к показательной.

Аналогичной будет сложность по памяти, т.к. в худшем случае придется хранить в очереди приоритетов всевозможные пути.

Выводы

В ходе выполнения лабораторной работы была решена задача нахождения кратчайшего пути в графе методом A* на языке C++, и исследован алгоритм A*. Полученный алгоритм имеет сложность показательную как по времени, так и по памяти. Так же была изучена структура данных очередь с приоритетом.

Была написана программа, строящая граф в виде матрицы смежности, очередь с приоритетом, и вычисляющая кратчайший путь от заданной вершины до конечной, если такой существует.

ПРИЛОЖЕНИЕ А.
КОД ПРОГРАММЫ

lr2.cpp

```
#include <iostream>
#include <vector>
#include <cmath>
#include <string.h>
#include "queue"

using namespace std;

struct Vertex{
 vector <int> res;
 double prior;
 double evr;
 Vertex():prior(0),evr(0){}
 Vertex(double _p, double _e):prior(_p),evr(_e){}
};

class Graph{
private:
 vector<int> way;
 double ** matrix;
 size_t size;
public:
   Graph(size_t _N, int start): size(_N){
     way.push_back(start);
     matrix = new double *[size];
     for(int i = 0; i < size; i++)
       matrix[i] = new double[size];
     for(int i = 0; i < size; i++){
       for(int j = 0; j < size; j++){
         matrix[i][j] = 0;
       }
     }
   }
   ~Graph(){
     way.clear();
     for(int i = 0; i < size; i++)
       delete [] matrix[i];
   }
   void A_star(vector<int>&vertex,int start, int finish,
priority_queue<Vertex>&p_queue, double dist);
   void Print();
   double Evristic(vector<int> vert,int i, int j);
   void New_matrix(int vert,vector<int> &vertex,vector<int> &v,vector
<int> &u,vector<double> &w);
   int Find_i(vector<int>&vert, int f);

};
```

```
bool operator < (const Vertex &v1, const Vertex &v2){
   return v1.prior > v2.prior;
}

void Graph::Print(){
   for(int i = 0; i < way.size(); i++)
       cout << way[i] << ' ';
}

void Graph::Matrix(int from, int to, double way){
   matrix[from][to] = way;
}

void Graph::New_matrix(int vert,vector<int> &vertex,vector<int>
&v,vector <int> &u,vector<double> &w){
   for(int i = 0; i < size; i++){
       if(v[i] == vert) Matrix(Find_i(vertex, v[i]), Find_i(vertex,
u[i]), w[i]);
   }
}
int Graph::Find_i(vector<int>&vert, int f){
   for(int i = 0; i<vert.size(); i++){
       if (vert[i]==f) return i;
   }
}

double Graph::Evristic(vector<int> vert,int i, int j){
   return abs(vert[i] - vert[j]);
}

void Graph::A_star(vector<int>&vertex,int start, int finish,
priority_queue<Vertex>&p_queue, double dist){
 vector<int> str;
 while(1){
   for(int i = 0; i < size; i++){
     if(matrix[Find_i(vertex,start)][Find_i(vertex,i)] != 0){
       Vertex tmp;
       tmp.evr = Evristic(vertex, finish, i);
       tmp.prior = matrix[Find_i(vertex,start)][Find_i(vertex,i)]  +
dist + tmp.evr;
       for(auto j: str)
           tmp.res.push_back(Find_i(vertex,j));
       tmp.res.push_back(vertex[i]);
       p_queue.push(tmp);
     }
   }
   if(p_queue.empty()) break;
   else {
       Vertex last;
       last = p_queue.top();
       p_queue.pop();
       start = last.res[last.res.size()-1];
```

```
        str = last.res;
        dist = last.prior - last.evr;
    }
    if(str[str.size()-1] == finish){
        for(auto i: str)
            way.push_back(i);
      break;
    }
  }
}

void Graph::Print_m(){
 for(int i = 0; i<size; i++){
   for(int j = 0; j<size; j++){
     cout << matrix[i][j] << ' ';
   }
   cout << endl;
 }
}

int main(){
     int start, finish, from, to;
     double way;
     cin >> start >> finish;
     priority_queue <Vertex> p_queue;
     vector <int> vertex;
     vector<int> v_from;
     vector <int> v_to;
     vector<double> v_way;
     bool ch1,ch2;
     while(cin >> from >> to >> way){
         ch1 = false;
         ch2 = false;
         v_from.push_back(from);
         v_to.push_back(to);
         v_way.push_back(way);
         for(int i = 0; i < vertex.size(); i++){
           if(v_from.back() == vertex[i]) ch1 = true;
           if(v_to.back() == vertex[i]) ch2 = true;
         }
         if(ch1 == false) vertex.push_back(v_from.back());
         if(ch2 == false) vertex.push_back(v_to.back());
     }
     Graph new_graph(vertex.size(), start);
     for(int i = 0; i < vertex.size(); i++){
       new_graph.New_matrix(vertex[i],vertex, v_from,v_to,v_way);
     }
     new_graph.A_star(vertex,start, finish, p_queue, 0);
     new_graph.Print();
     return 0;
}
```

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Результаты тестов представлены в табл. 1.

Входные данные	Выходные данные
-3 3 -3 2 1 -2 1 3 -2 0 5 1 2 5 2 0 6 2 3 3 3 4 2	-3 -2 1 2 3
3 0 3 5 1 3 7 1 3 0 2 7 0 3	3 0
-2 6 -2 0 5 -2 1 1 -2 5 2 0 5 1 5 6 1	-2 5 6