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. 9383 _____ Ноздрин В.Я.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Реализовать жадный алгоритм и альгоритм A*. Оценить сложность алгоритмов.

Задание 1. Вариант – 1 (В A* вершины именуются целыми числами).

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

Задание 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

Описание жадного алгоритма.

Для решения задачи был реализован жадный алгоритм поиска пути в графе. Построение пути начинается с выбора стартовой вершины. Дальше на каждом шаге делается следующиее:

1. Выбирается ребро с наименьшим весом, выполняется переход по этому ребру к новой вершине, которая становится текущей и помечается посещенной.
2. Если из текущей вершины нет исходящих ребер в непосещенные вершины, текущей вершиной становится предыдущая посещенная вершина.
3. Если текущая вершина совпадает с конечной или рассмотрены все возможные вершины, алгоритм завершен.

Оценка сложности жадного алгоритма.

Так как алгоритм переберает все ребра и проходит по всем вершинам (в худшем случае), сложность данного алгоритма будет **O(|V+|E|).**

Описание алгоритма A*.

Для решения задачи был реализован алгоритм A* поиска пути в графе. Вершины графа помечаются в «открытые» и «закрытые». И применяется следующий алгоритм:

1. Текущая вершина — открытая вершина, выбранная с наименьшей полной стоимостью f(V)=g(V)+h(V)
2. Если текущая вершина — конечная, алгоритм завершен.
3. Текущая вершина помечается закрытой.
4. Для каждого потомка текущей вершины:
    1. Рассчитывается функция полного пути f(V)=g(V)+h(V).
    2. Если вершина не открыта, устанавливается ей значение функции и запоминается текущая верщина как предок.

3. Если вершина открыта, сравнивается ее текущее значение функции полного пути с новым значением. Если новое значение меньше, у потомка обновляется значение функции и поле предка.

Если путь существует, алгортим будет завершен в один момент на пункте 2 алгоритма. Если этого не происходит, заканчиваются открытые вершины и алгоритм завершается с сообщением, что пути не существует.

Оценка сложности алгоритма A*.

Функция полного пути f(V)=g(V)+h(V) вычисляется как минимальный уже построенный путь g(V), и эвристическая функция h(V). От выбора этой самой эвристической функции зависит сложность алгоритма A*.

В худшем случае количество обрабатываемых вершин растет экспоненциально в зависимости от длины оптимального пути. Но если выполнено следующее неравенство, сложность становится полиномиальной:

$|h(V)-h^{эс}(V)|\leq O(logh^{эс}(V))$, где h*(V) – точная оценка длины пути из текущей вершины V в конечную.

Описание функций и структур данных.

class Vertex – класс, хранящий информацию о вершинах. Вспомогательный инструмент в реализации алгоритма.

bool belongs – метод, проверяющий принадлежность числа вектору чисел.

void addToQueue() – метод, добавляющий вершину в очередь с приоритетом, хранящую открытые вершины. Также обрабатывает вершины и устанавливает предков.

std::string makePath() – метод, возвращающий решение для данного набора данных в виде строки.

int heuristicFunction(const int&, const int&) – эвристическая функция.

Выводы.

Применен на практике алгоритм поиска минимального пути в графе A*. Исследованы сложности жадного алгоритма и алгоритма A*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <iostream>
#include <vector>
#include <tuple>

#include "A.cpp"
//#include "greedy.cpp"

#define MODE 1

#if MODE == 1
int main() {
/**1 5
1 2 3.0
2 3 1.0
3 4 1.0
1 4 5.0
4 5 1.0*/
  int start, end;
  std::vector<std::tuple<int, int, float>> edges;
  AStar::readData(start, end, edges);
  std::cout << AStar::makePath(start, end, edges);
  return 0;
}
#elif MODE == 2
int main() {
  char  start, end;
  std::vector<std::tuple<char, char, float>> edges;
  Greedy::readData(start, end, edges);
  std::cout << Greedy::makePath(start, end, edges);
  return 0;
}
#else
int main() {
  return 0;
}
#endif // MODE
```

Файл Greedy.cpp:

```
#include <iostream>
#include <vector>
#include <tuple>

namespace Greedy {
  std::tuple<char, char, float> readEdge(std::string);

  void readData(char &, char &, std::vector<std::tuple<char, char,
float>> &);

  __attribute__((unused)) void printData(char &, char &,
std::vector<std::tuple<char, char, float>> &);
```

```
  std::tuple<char, char, float> *getVerticeGreedy(char, const
std::vector<std::tuple<char, char, float>> &);

  std::string makePath(char, char, std::vector<std::tuple<char, char,
float>>);
}

std::tuple<char,char,float> Greedy::readEdge(std::string edge) {
  char  v1  = edge[0],
       v2  = edge[2];
  float len = stof(edge.substr(4));
  return std::make_tuple(v1, v2, len);
}

void Greedy::readData(char& start, char& end,
std::vector<std::tuple<char, char, float>>& edges) {
  std::string edge;
  std::getline(std::cin, edge);
  start = edge[0];
  end   = edge[2];
  do {
    std::getline(std::cin, edge);
    if (!std::cin)
      break;
    if (edge.length() >= 5)
      edges.emplace_back(readEdge(edge));
    else
      break;
  } while (std::cin);
}

__attribute__((unused)) void Greedy::printData(char& start, char& end,
std::vector<std::tuple<char, char, float>>& edges) {
  std::cout << start << ' ' << end << '\n';
  for (auto i : edges)
    std::cout << std::get<0>(i) << ' ' << std::get<1>(i) << ' ' <<
std::get<2>(i) << '\n';
}

std::tuple<char,char,float>* Greedy::getVerticeGreedy(char start, const
std::vector<std::tuple<char,char,float>>& edges) {
  std::tuple<char, char, float> *min = nullptr;
  for (auto i: edges) {
    if (std::get<0>(i) == start) {
      if (!min) {
        min = new std::tuple<char, char, float>();
        *min = i;
      }
      if (std::get<2>(*min) > std::get<2>(i))
        *min = i;
    }
  }
  return min;
}

std::string Greedy::makePath(char start, char end,
std::vector<std::tuple<char, char, float>> edges) {
```

```
  std::string result, errorMsg = "ERROR:\tNO PATH";
  while (true) {
    result += start;
    if (start == end)
      return result;
    if (edges.empty())
      return errorMsg;

    std::tuple<char,char,float> *min = getVerticeGreedy(start, edges);
    if (!min) {
      result.erase(result.size() - 1);
      start = result[result.size() - 1];
      result.erase(result.size() - 1);
      continue;
    }

    start = std::get<1>(*min);
    for (int i = 0; i < edges.size(); i++)
      if (std::get<1>(edges[i]) == start) {
        edges.erase(edges.begin() + i);
        i--;
      }
  }
}
```

Файл A.cpp:

```
#include <algorithm>
#include <iostream>
#include <queue>
#include <tuple>
#include <vector>

namespace AStar {
  /// Input/output functions
  std::vector<std::string> split(std::string&,
std::vector<std::string>&);
  std::tuple<int, int, float> readEdge(std::string);
  void readData(int&, int&, std::vector<std::tuple<int, int, float>>&);
  void printData(int&, int&, std::vector<std::tuple<int, int, float>>&);
  /// Realisation
  int heuristicFunction(const int&, const int&);
  class Vertex {
   public:
    int name, route_prev;
    float f, g, h;

    Vertex(int name, float g, int end) : name(name), g(g), route_prev(-1)
{
      h = (float) heuristicFunction(name, end);
      f = g + h;
    }

    friend bool operator<=(const Vertex &a, const Vertex &b) {
      return a.f <= b.f;
    }
```

```
    friend bool operator>=(const Vertex &a, const Vertex &b) {
      return a.f >= b.f;
    }

    friend bool operator==(const Vertex &a, const Vertex &b) {
      return a.name == b.name;
    }
  };
  bool belongs(int, const std::vector<int>&);
  void addToQueue(Vertex&, int&, const std::vector<std::tuple<int, int,
float>>&, std::vector<Vertex>&,
std::priority_queue<Vertex,std::vector<Vertex>,
                  std::greater_equal<>>&, std::vector<int>&);
  std::string makePath(int, int, const std::vector<std::tuple<int, int,
float>>&);
}

std::vector<std::string> AStar::split(std::string &str,
std::vector<std::string> &result) {
  for (int i = 0, j = 0; j < str.size(); j++) {
    if (str[j] == ' ' || str[j] == '\n') {
      result.emplace_back(str.substr(i, j - i));
      i = j + 1;
    } else if (j == str.size() - 1)
      result.emplace_back(str.substr(i));
  }
  return result;
}

std::tuple<int, int, float> AStar::readEdge(std::string edge) {
  std::vector<std::string> vec;
  split(edge, vec);
  int v1 = stoi(vec[0]);
  int v2 = stoi(vec[1]);
  float len = stof(vec[2]);
  return std::make_tuple(v1, v2, len);
}

void AStar::readData(int &start, int &end, std::vector<std::tuple<int,
int, float>> &edges) {
  std::string edge;

  std::getline(std::cin, edge);
  std::vector<std::string> vec;
  split(edge, vec);

  start = stoi(vec[0]);
  end = stoi(vec[1]);

  do {
    std::getline(std::cin, edge);
    if (!std::cin)
      break;
    int c = 0;
    for (auto x: edge)
      if (x == ' ')
        c++;
```

```
    if (c == 2)
      edges.emplace_back(readEdge(edge));
    else
      break;
  } while (std::cin);
}

void AStar::printData(int &start, int &end, std::vector<std::tuple<int,
int, float>> &edges) {
  std::cout << start << ' ' << end << '\n';
  for (auto i : edges)
    std::cout << std::get<0>(i) << ' ' << std::get<1>(i) << ' ' <<
std::get<2>(i) << '\n';
}

int AStar::heuristicFunction(const int &v, const int &u) {
  return abs(v - u);
}

bool AStar::belongs(int name, const std::vector<int> &close) {
  return std::any_of(close.cbegin(), close.cend(), [name](auto x)
{ return x == name; });
}

void AStar::addToQueue(Vertex &cur, int &end, const
std::vector<std::tuple<int, int, float>> &edges,
                       std::vector<Vertex>&
vertices,

std::priority_queue<Vertex,std::vector<Vertex>,std::greater_equal<>>&
open,
                       std::vector<int>&
close) {
  for (const auto &edge : edges) {
    if (cur.name == std::get<0>(edge)) {
      if (belongs(std::get<1>(edge), close))
        continue;
      Vertex x(std::get<1>(edge), std::get<2>(edge) + cur.g, end);
      x.route_prev = (int) vertices.size() - 1;
      open.push(x);
    }
  }
}

std::string AStar::makePath(int start, int end, const
std::vector<std::tuple<int, int, float>>& edges) {
  std::string result, errorMsg = "ERROR:\tNO PATH";

  std::vector<Vertex> vertices;
  std::priority_queue<Vertex, std::vector<Vertex>, std::greater_equal<>>
open;
  std::vector<int> close;

  Vertex cur(start, 0, 0);
  open.push(cur);

  while (cur.name != end && !open.empty()) {
    cur = open.top();
```

```
    open.pop();
    if (belongs(cur.name, close))
      continue;
    vertices.push_back(cur);
    addToQueue(cur, end, edges, vertices, open, close);
  }

  if (open.empty())
    return errorMsg;

  // restore the path
  // cur == end
  std::vector<int> res;
  while (true) {
    if (cur.route_prev == -1) {
      res.push_back(cur.name);
      break;
    }
    res.push_back(cur.name);
    cur = vertices[cur.route_prev];
  }

  for (int i = (int)(res.size()) - 1; i >= 0; i--)
    result += std::to_string(res[i]) + ' ';

  return result;
}
```

Файл tests.cpp:

```
#define CATCH_CONFIG_MAIN
#include <catch2/catch.hpp>
#include "../src/Squaring.h"

TEST_CASE( "Base case is correct", "[base case]" ) {
  Squaring squaring(10);
  squaring.baseCase1();

  std::vector<std::vector<int>> testSquare = {
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  2},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  3},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  4},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  5},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  6},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  7},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  8},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  9},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  10},
      {11, 12, 13, 14, 15, 16, 17, 18, 19, 20},
  };
  REQUIRE(squaring.getCurrentSquaring() == testSquare);
}

TEST_CASE( "Backtrack is correct", "[backtrack]" ) {
  Squaring squaring(10);
  squaring.baseCase1();
```

```
  squaring.backtrack();
  std::vector<std::vector<int>> testSquare = {
      {1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  };
  REQUIRE(squaring.getCurrentSquaring() == testSquare);

  squaring.backtrack();
  testSquare = {
      {1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  };
  REQUIRE(squaring.getCurrentSquaring() == testSquare);

  squaring.backtrack();
  squaring.backtrack();
  testSquare = {
      {1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
      {1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
      {1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
      {1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
      {1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};
  REQUIRE(squaring.getCurrentSquaring() == testSquare);
}

TEST_CASE( "Pop is correct", "[pop]" ) {
  Squaring squaring(10);
  squaring.baseCase1();
  squaring.pop();

  std::vector<std::vector<int>> testSquare = {
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  2},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  3},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  4},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  5},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  6},
      {1,  1,  1,  1,  1,  1,  1,  1,  1,  7},
```

```
        {1,  1,  1,  1,  1,  1,  1,  1,  1,  8},
        {1,  1,  1,  1,  1,  1,  1,  1,  1,  9},
        {1,  1,  1,  1,  1,  1,  1,  1,  1,  10},
        {11, 12, 13, 14, 15, 16, 17, 18, 19, 0},
    };
    REQUIRE(squaring.getCurrentSquaring() == testSquare);

    squaring.pop();
    testSquare = {
        {1,  1,  1,  1,  1,  1,  1,  1,  1, 2},
        {1,  1,  1,  1,  1,  1,  1,  1,  1, 3},
        {1,  1,  1,  1,  1,  1,  1,  1,  1, 4},
        {1,  1,  1,  1,  1,  1,  1,  1,  1, 5},
        {1,  1,  1,  1,  1,  1,  1,  1,  1, 6},
        {1,  1,  1,  1,  1,  1,  1,  1,  1, 7},
        {1,  1,  1,  1,  1,  1,  1,  1,  1, 8},
        {1,  1,  1,  1,  1,  1,  1,  1,  1, 9},
        {1,  1,  1,  1,  1,  1,  1,  1,  1, 10},
        {11, 12, 13, 14, 15, 16, 17, 18, 0, 0},
    };
    REQUIRE(squaring.getCurrentSquaring() == testSquare);
}
```