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. 7304 Кошманов Н.А.

Преподаватель Филатов А.Ю.

Санкт-Петербург

2019

Цель работы:

Ознакомиться с жадным алгоритмом и методом A*, получить навыки их программирования и применения на языке программирования C++.

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.
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.

Основные теоретические положения:

Жадный алгоритм — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным. Известно, что если структура задачи задается матроидом, тогда применение жадного алгоритма выдаст глобальный оптимум.

A* — это модификация алгоритма Дейкстры, оптимизированная для единственной конечной точки. Алгоритм Дейкстры может находить пути ко всем точкам, A* находит путь к одной точке. Он отдаёт приоритет путям, которые ведут ближе к цели.

Ход работы:

1. **Жадный алгоритм.**
   Выполнен жадный алгоритм на основе стека: пути, по которым не пошли на каждом шаге записываются в стек в порядке приоритета. Таким образом, если из текущей вершины не будет выхода, алгоритм вернется на шаг назад и возьмет последний путь из стека.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

struct WAY{
   char from;
   char to;
   double cost;
};

bool cmp (WAY* a, WAY* b) {
   return a->cost < b->cost;
}

inline void search(vector<WAY*> ways, vector<WAY*> stack, WAY* task, char current, vector<char>*
res) {
   if (task->to == current) {
      return;
   }

   vector<WAY*> buf;
   for (int i = 0; i < ways.size(); i++) {
      if (ways[i]->from == current) {
          buf.push_back(ways[i]);
      }
   }

   if (buf.size() == 0) {
      char next = stack[stack.size() - 1]->to;
      stack.pop_back();
      res->pop_back();
      res->push_back(next);
      search(ways, stack, task, next, res);
      return;
   }

   std::sort(buf.begin(), buf.end(), cmp);
   char next = buf[0]->to;

   for (unsigned long i = (buf.size() - 1); i > 0; i--) {
      stack.push_back(buf[i]);
   }
   res->push_back(next);
   search(ways, stack, task, next, res);
}

int main(int argc, const char * argv[]) {
   vector<WAY*> ways;
   vector<WAY*> history;
   WAY* task = new WAY;
   vector<char> res;

   scanf("%c %c\n", &(task->from), &(task->to));
   res.push_back(task->from);
```

```
    while (true) {
        WAY* ptr = new WAY;
        if ((cin >> ptr->from) && (cin >> ptr->to) && (cin >> ptr->cost))
            ways.push_back(ptr);
        else break;
    }

    search(ways, history, task, task->from, &res);

    for (int i = 0; i < res.size(); i++)
        cout << res[i];
    return 0;
}
```

2. Алгоритм A*.

В данном алгоритме необходимо пройти по всем возможным путям из из А в Б. На каждом шаге берутся вершины в порядке приоритетов (Разница символов в таблице ASCII, стоимость пути). Если дошли до искомой вершины, то текущее решение сравнивается с лучшим.

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

using namespace std;

struct WAY{
    char from;
    char to;
    double cost;
};

bool cmp (WAY* a, WAY* b) {
    int v = (int)(a->to) - (int)(b->to);
    if (v == 0) return a->cost > b->cost;
    else return v < 0;
}

bool isNewWayBest(vector<WAY*> bestWay, vector<WAY*> newWay) {
    if (!bestWay.size()) return true;

    double bestCost = 0.0;
    double newCost = 0.0;

    for (int i = 0; i < bestWay.size(); i++)
        bestCost += bestWay[i]->cost;

    for (int i = 0; i < newWay.size(); i++)
        newCost += newWay[i]->cost;

    if (newCost != bestCost)
        return newCost < bestCost;

    for (int i = 0; i < newWay.size(), i < bestWay.size(); i++) {
        int a1 = (int)(newWay[i]->to) - (int)(newWay[i]->from);
```

```
        int a2 = (int)(bestWay[i]->to) - (int)(bestWay[i]->from);
        if (a1 > a2) return true;
    }

    return newWay.size() < bestWay.size();
}

inline void findWay(vector<WAY*> ways, vector<WAY*> report, WAY* task, vector<WAY*>*
bestWay) {
    std::sort(ways.begin(), ways.end(), cmp);
    if (!report.size()) {
        for (int i = 0; i < ways.size(); i++) {
            if (ways[i]->from == task->from) {
                vector<WAY*> rep;
                rep.push_back(ways[i]);
                findWay(ways, rep, task, bestWay);
            }
        }
        return;
    }

    if (report[report.size() - 1]->to == task->to) {
        if (isNewWayBest(*bestWay, report))
            *bestWay = report;
        return;
    }

    for (int i = 0; i < ways.size(); i++)
        if (report[report.size() - 1]->to == ways[i]->from) {
            vector<WAY*> ptr = report;
            ptr.push_back(ways[i]);
            findWay(ways, ptr, task, bestWay);
        }
}

int main(int argc, const char * argv[]) {
    vector<WAY*> ways;
    WAY* task = new WAY;
    vector<WAY*> report;
    vector<WAY*> best;

    scanf("%c %c\n", &(task->from), &(task->to));

    while (true) {
        WAY* ptr = new WAY;
        if ((cin >> ptr->from) && (cin >> ptr->to) && (cin >> ptr->cost))
            ways.push_back(ptr);
        else break;
    }

    findWay(ways, report, task, &best);
    cout << best[0]->from;
    for (int i = 0; i < best.size(); i++) {
        cout << best[i]->to;
    }
    return 0;
}
```

Результат:

Из рисунков 1 и 2 видно, что разработанные программы выполняют поставленные задачи, а именно: Программа 1 находи самый дешевый на каждом шаге путь из вершины А в вершину Б. Программа 2 находит самый дешевый путь с учетом разниц значений символов в таблице ANSII на каждом шаге.

```
a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
d e 1.0abcdeProgram ended with exit code: 0
```

Рисунок 1.

```
a e
a b 3.0
b c 1.0
c d 1.0
a d 5.0
d e 1.0adeProgram ended with exit code: 0
```

Рисунок 2.

Вывод:

Таким образов, в ходе данной лабораторной работы было подробно изучено написание жадного алгоритма и алгоритма А*. Полученный результат удовлетворяет задание лабораторной работы.