

Московский авиационный институт  
(национальный исследовательский университет)

Факультет информационных технологий и прикладной  
математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу «Дискретный анализ»

Студент: И. П. Попов  
Преподаватель: С. А. Сорокин  
Группа: М8О-306Б-20  
Дата:  
Оценка:  
Подпись:

Москва, 2023

# Курсовой проект

## Задача:

Реализуйте систему для поиска пути в графе дорог с использованием эвристических алгоритмов.

```
./prog preprocess -nodes <nodes file> -edges <edges file> -output <preprocessed graph>
```

| Ключ    | Значение                     |
|---------|------------------------------|
| -nodes  | входной файл с перекрёстками |
| -edges  | входной файл с дорогами      |
| -output | выходной файл с графом       |

```
./prog search -graph <preprocessed graph> -input <input file> -output <output file> [-full-output]
```

| Ключ         | Значение  |
|--------------|---|
| -graph       | входной файл с графом                             |
| -input       | входной файл с запросами                          |
| -output      | выходной файл с ответами на запросы               |
| -full-output | переключение формата выходного файла на подробный |

**Файл узлов:** <id> <lat> <lon>

**Файл рёбер:** <длина дороги в вершинах [n]> <id 1> <id 2> ... <id n>

## Выходной файл:

Если опция -full-output не указана: на каждый запрос в отдельной строке выводится длина кратчайшего пути между заданными вершинами с относительной погрешностью не более 1e-6.

Если опция -full-output указана: на каждый запрос выводится отдельная строка, с длиной кратчайшего пути между заданными вершинами с относительной погрешностью не более 1e-6, а затем сам путь в формате как в файле рёбер.

Расстояние между точками следует вычислять как расстояние между точками на сфере с радиусом 6371км, если пути между точками нет, вывести -1 и длину пути в вершинах 0.

# 1 Описание

**Поиск  $A^*$**  (произносится «А звезда» или «А стар», от англ. A star) — в информатике и математике, алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).

Порядок обхода вершин определяется эвристической функцией «расстояние + стоимость» (обычно обозначаемой как  $f(x)$ ). Эта функция — сумма двух других: функции стоимости достижения рассматриваемой вершины ( $x$ ) из начальной (обычно обозначается как  $g(x)$  и может быть как эвристической, так и нет), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначается как  $h(x)$ ).

Функция  $h(x)$  должна быть допустимой эвристической оценкой, то есть не должна переоценивать расстояния к целевой вершине. Например, для задачи маршрутизации  $h(x)$  может представлять собой расстояние до цели по прямой линии, так как это физически наименьшее возможное расстояние между двумя точками.

Этот алгоритм был впервые описан в 1968 году Питером Хартом, Нильсом Нильсоном и Бертрамом Рафаэлем. Это по сути было расширение алгоритма Дейкстры, созданного в 1959 году. Новый алгоритм достигал более высокой производительности (по времени) с помощью эвристики. В их работе он упоминается как «алгоритм А». Но так как он вычисляет лучший маршрут для заданной эвристики, он был назван  $A^*$ .

## 2 Исходный код

Процесс написания программы состоит из следующих этапов:

1. Реализация вспомогательных типов 2. Реализация предобработки файла 3. Реализация алгоритма  $A^*$  4. Реализация восстановления пути

Начнём с реализации некоторых вспомогательных типов - для хранения вершины в массиве вершин и для вершины при её обработке алгоритмом  $A^*$ , а также структура для хранения ребра.

```
1 struct Node{
2     uint32_t id;
3     double lat;
4     double lon;
5
6     friend istream& operator>>(istream& is, Node& node){
7         is >> node.id >> node.lat >> node.lon;
8         return is;
9     }
10
11     friend bool operator<(Node& lhs, Node& rhs){
12         return lhs.id < rhs.id;
13     }
14 };
15
16 const double ANGLE = 180.0;
17 const double SPHERE_RADIUS = 6371 * 1e3;
18
19 struct Path{
20     uint32_t to;
21     double cost;
22
23     friend bool operator<(const Path& a, const Path& b){
24         if(a.cost != b.cost){
25             return a.cost > b.cost;
26         }
27         return a.to > b.to;
28     }
29 };
30
31 struct Edge{
32     uint32_t from;
33     uint32_t to;
34
35     friend bool operator<(const Edge& a, const Edge& b){
36         if(a.from != b.from){
37             return a.from < b.from;
38         }
```

```

39         return a.to < b.to;
40     }
41 };

```

Реализация предобработки графа для более удобного его представления. Обработка вершин:

```

1  vector<Node> nodes;
2  Node cur;
3  while(fscanf(nodesFile, "%u%lf%lf", &cur.id, &cur.lat, &cur.lon) > 0){
4      cur.id -= 1;
5      nodes.push_back(cur);
6  }
7
8  sort(nodes.begin(), nodes.end());
9
10 ids.resize(nodes.size());
11 for(size_t i = 0; i < nodes.size(); ++i){
12     ids[i] = nodes[i].id;
13 }
14
15 size_t size = nodes.size();
16 fwrite(&size, sizeof(size), 1, outputFile);
17 fwrite(&ids[0], sizeof(ids[0]), ids.size(), outputFile);
18 for(size_t i = 0; i < size; ++i){
19     fwrite(&nodes[i].lat, sizeof(nodes[0].lat), 1, outputFile);
20     fwrite(&nodes[i].lon, sizeof(nodes[0].lon), 1, outputFile);
21 }

```

Обработка рёбер с учётом ранее обработанных номеров вершин - для каждой вычисляется количество смежных к ней вершин:

```

1  vector<Edge> edges;
2  uint32_t n, curId, prevId;
3  while(fscanf(edgesFile, "%u%u", &n, &curId) == 2){
4      for(size_t i = 1; i < n; ++i){
5          prevId = curId;
6          fscanf(edgesFile, "%u", &curId);
7          edges.push_back({prevId - 1, curId - 1});
8          edges.push_back({curId - 1, prevId - 1});
9      }
10 }
11
12 sort(edges.begin(), edges.end());
13
14 uint32_t i = 0;
15 for(size_t k = 0; k < ids.size(); ++k){
16     uint32_t curEdgeFrom = ids[k];
17     while(i < edges.size() && edges[i].from == curEdgeFrom){
18         ++i;

```

```

19     }
20     fwrite(&i, sizeof(i), 1, outputFile);
21 }
22
23 for(Edge& edge : edges){
24     fwrite(&edge.to, sizeof(edge.to), 1, outputFile);
25 }

```

Реализация самого алгоритма A\*. При обработке вершины из файла загружаются смежные к ней:

```

1 double AStar(uint32_t start, uint32_t goal, vector<uint32_t>& ids,
2             vector<uint32_t>& offsets, FILE* inputFile, uint32_t infoStart, uint32_t
3             adjStart,
4             vector<double>& g, vector<double>& f, vector<uint32_t>& parent){
5
6     g.assign(g.size(), -1.0);
7     f.assign(f.size(), -1.0);
8     priority_queue<Path> q;
9
10    start = binSearch(start, ids);
11    goal = binSearch(goal, ids);
12
13    Node goalNode = getNode(goal, inputFile, infoStart);
14
15    g[start] = 0;
16    f[start] = g[start] + calcDistance(getNode(start, inputFile, infoStart), goalNode);
17    q.push({start, f[start]});
18
19    parent[start] = start;
20
21    while(!q.empty()){
22        uint32_t cur = q.top().to;
23        double curCost = q.top().cost;
24        q.pop();
25
26        if(cur == goal){
27            break;
28        }
29
30        if(curCost > f[cur]){
31            continue;
32        }
33
34        size_t startLoad = 0, toLoad = 0;
35        if(cur == 0){
36            toLoad = offsets[0];
37        } else{
38            startLoad = offsets[cur - 1]; // previous end offset
39            toLoad = offsets[cur] - offsets[cur - 1];

```

```

39     }
40
41     vector<uint32_t> curAdj(toLoad);
42     fseek(inputFile, adjStart + startLoad * sizeof(uint32_t), SEEK_SET);
43     for(size_t i = 0; i < toLoad; ++i){
44         fread(&curAdj[i], sizeof(uint32_t), 1, inputFile);
45     }
46
47     Node curNode = getNode(cur, inputFile, infoStart);
48
49     for(uint32_t next : curAdj){
50         next = binSearch(next, ids);
51
52         Node nextNode = getNode(next, inputFile, infoStart);
53
54         double tentativeScore = g[cur] + calcDistance(nextNode, curNode);
55
56         if((g[next] < 0) || (1e-6 < (g[next] - tentativeScore))){
57             g[next] = tentativeScore;
58             f[next] = g[next] + calcDistance(nextNode, goalNode);
59
60             parent[next] = cur;
61             q.push({next, f[next]});
62         }
63     }
64 }
65
66 if(g[goal] == -1.0){
67     return __DBL_MAX__;
68 }
69
70 return g[goal];
71 }

```



### 3 Консоль

```
lunidep@DESKTOP-I095C6F:~/da-cp make preprocess_europe
./prog preprocess -nodes europe.nodes -edges europe.edges -output europe.graph
lunidep@DESKTOP-I095C6F:~/da-cp make search_europe
g++ main.cpp preprocess.cpp search.cpp -std=c++20 -pedantic -Wall -Werror -o
prog -g
./prog search -graph europe.graph -input input.txt -output output.txt
lunidep@DESKTOP-I095C6F:~/da-cp cat output.txt
1406425.391301
```

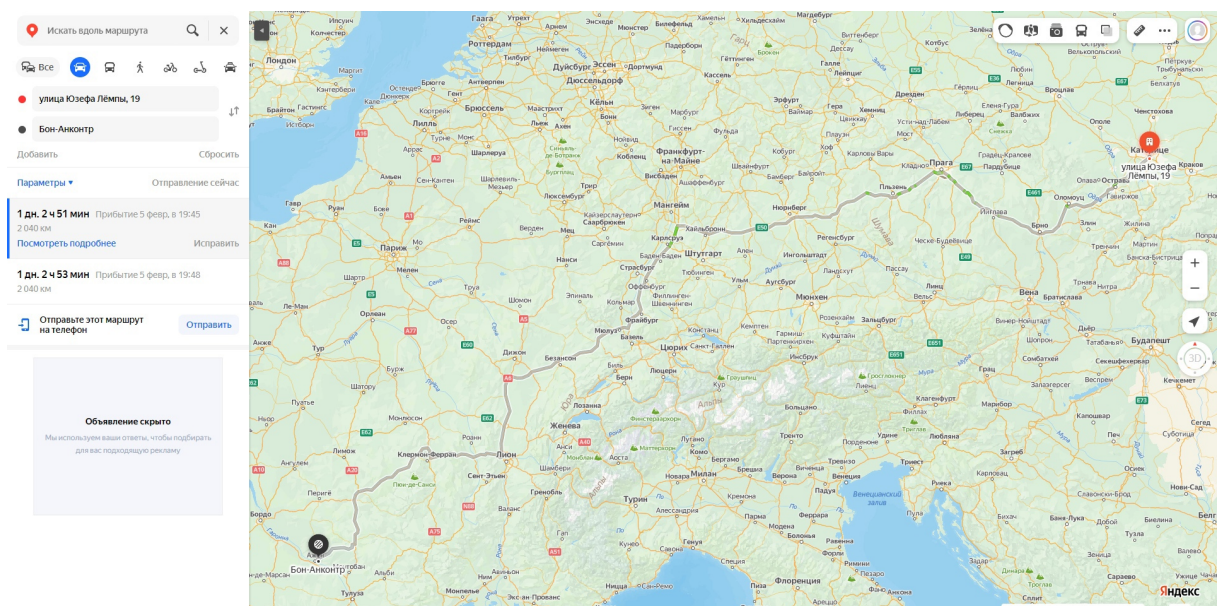
## 4 Тест производительности

Алгоритм запускается для двух случайных вершин и результат сравнивается с результатом, полученным из Яндекс карт.

Тест №1:

Вершины: 1378263516(50.0716306 23.9665022) и 945677474(44.2126213 0.6913325).

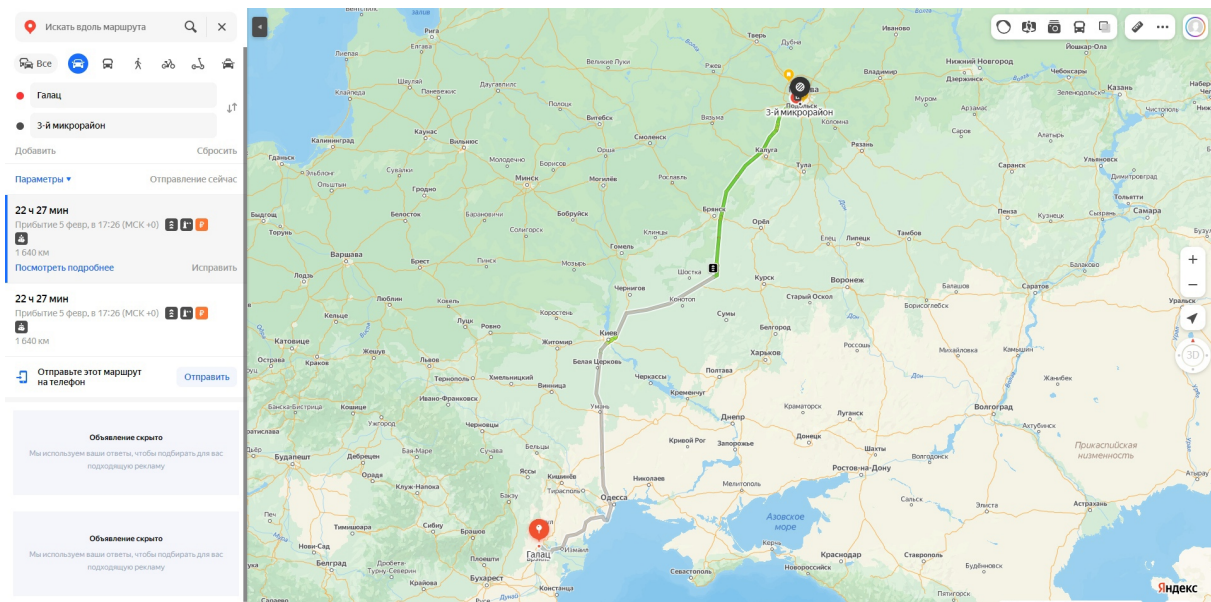
Расстояние: 2177076,893823



Тест №2:

Вершины: 99999431 (45.4500687 28.0243567) и 99999832(55.6211301 37.504896).

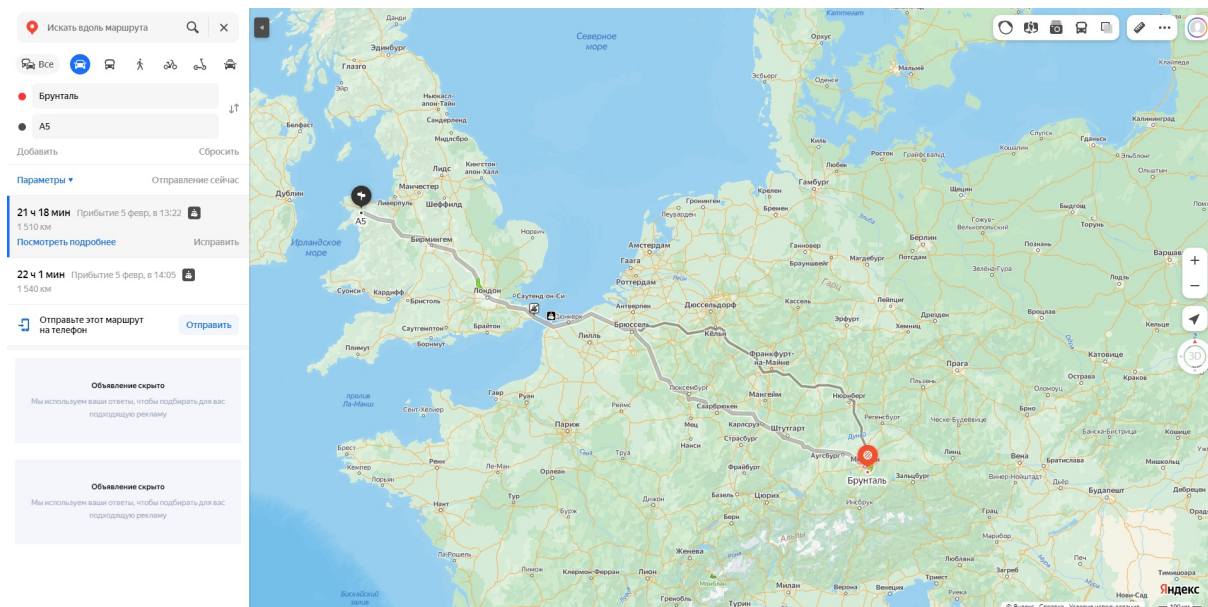
Расстояние: 1500199,914925



Тест №3:

Вершины: 14658632(48.040794 11.65938) и 14671427(53.1228894 -4.0159544).

Расстояние: 1406425.391301



## 5 Выводы

В ходе выполнения курсовой работы был изучен и реализован алгоритм  $A^*$ . Главной сложностью, помимо непосредственной реализации  $A^*$  стал грамотный парсинг графа в файл и обратно, чтобы при этом он занимал как можно меньше памяти. Мною был реализован вариант, который позволяет непосредственно в процессе работы  $A^*$  мы для текущей вершины загружаем смежные к ней из файла.

## Список литературы

- [1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))