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 крат- чайшего пути между заданными вершинами с относительной погрешностью не более 1e-6.

Если опция –full-output указана: на каждый запрос выводится отдельная строка, с длиной кратчай- шего 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 A». 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         }
```

```
        return a.to < b.to;
    }
};
```

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

```
vector<Node> nodes;
Node cur;
while(fscanf(nodesFile, "%u%lf%lf", &cur.id, &cur.lat, &cur.lon) > 0){
    cur.id -= 1;
    nodes.push_back(cur);
}

sort(nodes.begin(), nodes.end());

ids.resize(nodes.size());
for(size_t i = 0; i < nodes.size(); ++i){
    ids[i] = nodes[i].id;
}

size_t size = nodes.size();
fwrite(&size, sizeof(size), 1, outputFile);
fwrite(&ids[0], sizeof(ids[0]), ids.size(), outputFile);
for(size_t i = 0; i < size; ++i){
    fwrite(&nodes[i].lat, sizeof(nodes[0].lat), 1, outputFile);
    fwrite(&nodes[i].lon, sizeof(nodes[0].lon), 1, outputFile);
}
```

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

```
vector<Edge> edges;
uint32_t n, curId, prevId;
while(fscanf(edgesFile, "%u%u", &n, &curId) == 2){
    for(size_t i = 1; i < n; ++i){
        prevId = curId;
        fscanf(edgesFile, "%u", &curId);
        edges.push_back({prevId - 1, curId - 1});
        edges.push_back({curId - 1, prevId - 1});
    }
}

sort(edges.begin(), edges.end());

uint32_t i = 0;
for(size_t k = 0; k < ids.size(); ++k){
    uint32_t curEdgeFrom = ids[k];
    while(i < edges.size() && edges[i].from == curEdgeFrom){
        ++i;
```

```
    }
    fwrite(&i, sizeof(i), 1, outputFile);
}

for(Edge& edge : edges){
    fwrite(&edge.to, sizeof(edge.to), 1, outputFile);
}
```

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

```
double AStar(uint32_t start, uint32_t goal, vector<uint32_t>& ids,
             vector<uint32_t>& offsets, FILE* inputFile, uint32_t infoStart, uint32_t
                 adjStart,
             vector<double>& g, vector<double>& f, vector<uint32_t>& parent){

    g.assign(g.size(), -1.0);
    f.assign(f.size(), -1.0);
    priority_queue<Path> q;

    start = binSearch(start, ids);
    goal = binSearch(goal, ids);

    Node goalNode = getNode(goal, inputFile, infoStart);

    g[start] = 0;
    f[start] = g[start] + calcDistance(getNode(start, inputFile, infoStart), goalNode);
    q.push({start, f[start]});

    parent[start] = start;

    while(!q.empty()){
        uint32_t cur = q.top().to;
        double curCost = q.top().cost;
        q.pop();

        if(cur == goal){
            break;
        }

        if(curCost > f[cur]){
            continue;
        }

        size_t startLoad = 0, toLoad = 0;
        if(cur == 0){
            toLoad = offsets[0];
        } else{
            startLoad = offsets[cur - 1]; // previous end offset
            toLoad = offsets[cur] - offsets[cur - 1];
```

```
        }

        vector<uint32_t> curAdj(toLoad);
        fseek(inputFile, adjStart + startLoad * sizeof(uint32_t), SEEK_SET);
        for(size_t i = 0; i < toLoad; ++i){
            fread(&curAdj[i], sizeof(uint32_t), 1, inputFile);
        }

        Node curNode = getNode(cur, inputFile, infoStart);

        for(uint32_t next : curAdj){
            next = binSearch(next, ids);

            Node nextNode = getNode(next, inputFile, infoStart);

            double tentativeScore = g[cur] + calcDistance(nextNode, curNode);

            if((g[next] < 0) || (1e-6 < (g[next] - tentativeScore))){
                g[next] = tentativeScore;
                f[next] = g[next] + calcDistance(nextNode, goalNode);

                parent[next] = cur;
                q.push({next, f[next]});
            }
        }
    }

    if(g[goal] == -1.0){
        return __DBL_MAX__;
    }

    return g[goal];
}
```

3 Консоль

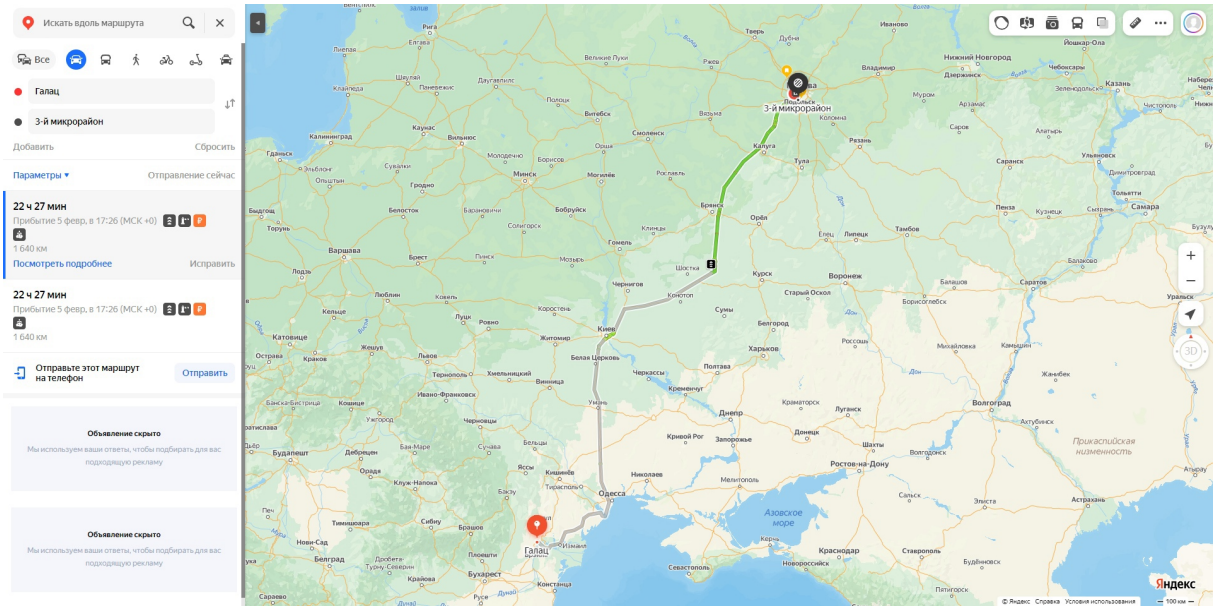
```
lunidep@DESKTOP-IO95C6F:~/da-cp make preprocesseurope
./prog preprocess -nodes europe.nodes -edges europe.edges -output europe.graph
lunidep@DESKTOP-IO95C6F:~/da-cp make search_europe
g++ main.cpp preprocess.cpp search.cpp -std=c++20 -pedantic -Wall -Werror -o
prog -g
./prog search -graph europe.graph -input input.txt -output output.txt
lunidep@DESKTOP-IO95C6F:~/da-cp cat output.txt
1406425.391301
```

4 Тест производительности

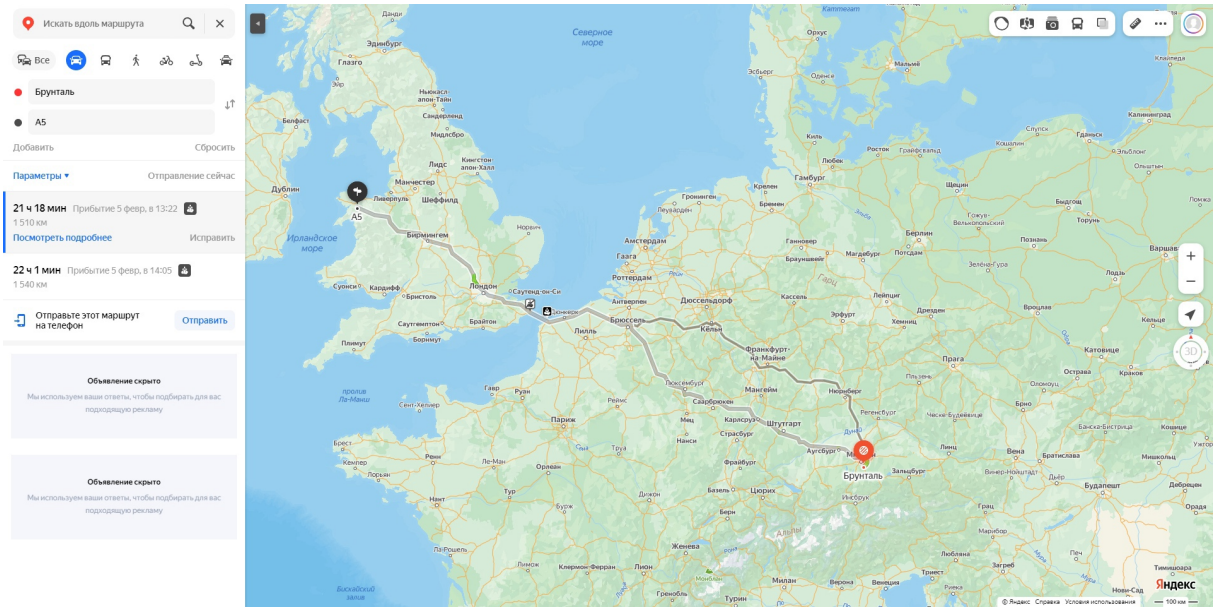
Алгоритм запускается для двух случайных вершин и результат сравнивается с результатом, полученным из Яндекс карт.

Тест №1:
Вершины: 1378263516(50.0716306 23.9665022) и 945677474(44.2126213 0.6913325).
Расстояние: 2177076,893823
Время вычисления: 3m22.033s

Тест №2:
Вершины: 99999431 (45.4500687 28.0243567) и 99999832(55.6211301 37.504896).
Расстояние: 1500199,914925
Время вычисления: 27.249s



Тест №3:
Вершины: 14658632(48.040794 11.65938) и 14671427(53.1228894 -4.0159544).
Расстояние: 1406425.391301
Время вычисления: 3m3.101s



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.))