МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

“Системы искусственного интеллекта”

Изучение алгоритмов поиска

Вариант №4 (Рига - Уфа)

Студент:

Воробьев Кирилл Олегович

Группа P33302

Преподаватель:

Королёва Ю.А.

Санкт-Петербург, 2022

Цель работы:

Исследование алгоритмов решения задач методом поиска

Задание:

Имеется транспортная сеть, связывающая города СНГ. Сеть представлена в виде таблицы связей между городами. Связи являются двусторонними, т.е. допускают движение в обоих направлениях. Необходимо проложить маршрут из одной заданной точки в другую.

Этап 1. Неинформированный поиск. На этом этапе известна только топология связей между городами. Выполнить:

1) поиск в ширину;

2) поиск глубину;

3) поиск с ограничением глубины;

4) поиск с итеративным углублением;

5) двунаправленный поиск.

Отобразить движение по дереву на его графе с указанием сложности каждого вида поиска. Сделать выводы.

Этап 2. Информированный поиск. Воспользовавшись информацией о протяженности связей от текущего узла, выполнить:

1) жадный поиск по первому наилучшему соответствию;

2) затем, используя информацию о расстоянии до цели по прямой от каждого узла, выполнить поиск методом минимизации суммарной оценки A*.

Отобразить на графе выбранный маршрут и сравнить его сложность с неинформированным поиском. Сделать выводы.

Выполнение работы:

Неинформативный поиск

```
map<string, vector<pair<string, int>>>
        cities = {
        {"Brest",             {
                                    make_pair("Vilnius", 531),
                                    make_pair("Vitebsk", 638),
                                    make_pair("Kaliningrad", 699)
                              }},

        {"Vilnius",           {
                                    make_pair("Brest", 531),
                                    make_pair("Vitebsk", 360),
                                    make_pair("Daugavpils", 211),
                                    make_pair("Daugavpils", 211),
```

```
                                make_pair("Kaliningrad", 333),
                                make_pair("Kaunas", 102),
                                make_pair("Kiev", 734)
                        }},

        {"Vitebsk",             {
                                make_pair("Brest", 638),
                                make_pair("Vilnius", 360),
                                make_pair("Voronezh", 869),
                                make_pair("Volgograd", 1455),
                                make_pair("Nizhniy Novgorod", 911),
                                make_pair("Saint-Petersburg", 602),
                                make_pair("Orel", 522)
                        }},

        {"Voronezh",            {
                                make_pair("Vitebsk", 869),
                                make_pair("Volgograd", 581),
                                make_pair("Yaroslavl", 739)
                        }},

        {"Volgograd",           {
                                make_pair("Vitebsk", 1455),
                                make_pair("Voronezh", 581),
                                make_pair("Zhytomyr", 1493)
                        }},

        {"Nizhniy Novgorod",    {
                                make_pair("Vitebsk", 911),
                                make_pair("Moscow", 411)
                        }},

        {"Daugavpils",          {
                                make_pair("Vilnius", 211)
                        }},

        {"Kaliningrad",         {
                                make_pair("Brest", 699),
                                make_pair("Vilnius", 333),
                                make_pair("Saint-Petersburg", 739)
                        }},

        {"Kaunas",              {
                                make_pair("Riga", 267),
                                make_pair("Vilnius", 102)
                        }},

        {"Kiev",                {
                                make_pair("Zhytomyr", 131),
                                make_pair("Vilnius", 734),
                                make_pair("Chisinau", 467),
                                make_pair("Odessa", 487),
                                make_pair("Kharkov", 471)
                        }},

        {"Zhytomyr",            {
                                make_pair("Kiev", 131),
                                make_pair("Donetsk", 863),
                                make_pair("Volgograd", 1493)
                        }},

        {"Donetsk",             {
                                make_pair("Zhytomyr", 863),
                                make_pair("Moscow", 1084),
```

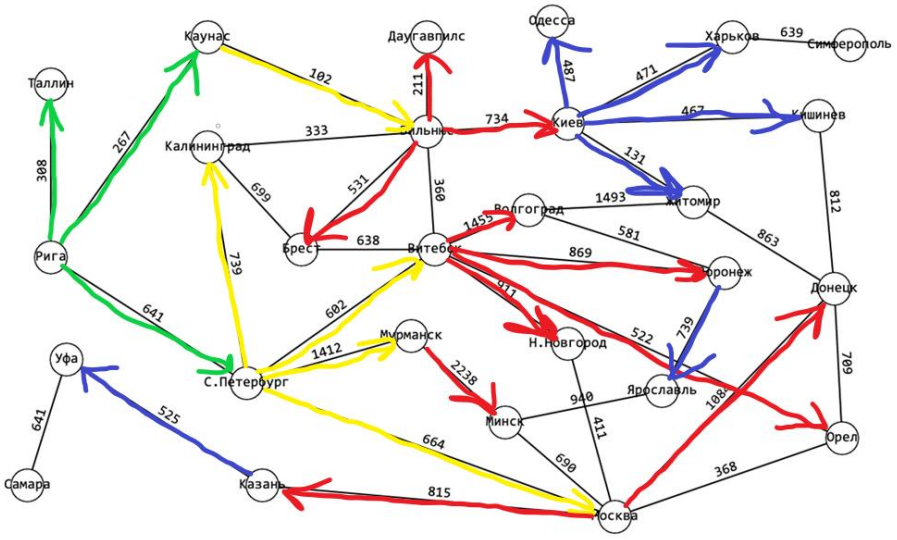
```
                                make_pair("Chisinau", 812),
                                make_pair("Orel", 709)
                        }},

        {"Chisinau",          {
                                make_pair("Kiev", 467),
                                make_pair("Donetsk", 812)
                        }},

        {"Saint-Petersburg", {
                                make_pair("Vitebsk", 602),
                                make_pair("Kaliningrad", 739),
                                make_pair("Riga", 641),
                                make_pair("Moscow", 664),
                                make_pair("Murmansk", 1412)
                        }},

        {"Riga",              {
                                make_pair("Saint-Petersburg", 641),
                                make_pair("Kaunas", 267),
                                make_pair("Tallin", 308)
                        }},

        {"Moscow",            {
                                make_pair("Saint-Petersburg", 664),
                                make_pair("Kazan", 815),
                                make_pair("Nizhniy Novgorod", 411),
                                make_pair("Minsk", 690),
                                make_pair("Donetsk", 1084),
                                make_pair("Orel", 368)
                        }},

        {"Kazan",             {
                                make_pair("Moscow", 815),
                                make_pair("Ufa", 525)
                        }},

        {"Minsk",             {
                                make_pair("Moscow", 690),
                                make_pair("Yaroslavl", 940),
                                make_pair("Murmansk", 2238)
                        }},

        {"Murmansk",          {
                                make_pair("Saint-Petersburg", 1412),
                                make_pair("Minsk", 2238)
                        }},

        {"Orel",              {
                                make_pair("Moscow", 368),
                                make_pair("Vitebsk", 522),
                                make_pair("Donetsk", 709)
                        }},

        {"Odessa",            {
                                make_pair("Kiev", 487)
                        }},

        {"Tallin",            {
                                make_pair("Riga", 308)
                        }},

        {"Kharkov",           {
                                make_pair("Kiev", 471),
```

```
                                        make_pair("Simferopol", 639)
                                }},

        {"Simferopol",          {
                                        make_pair("Kharkov", 639)
                                }},

        {"Yaroslavl",           {
                                        make_pair("Voronezh", 739),
                                        make_pair("Minsk", 940)
                                }},

        {"Ufa",                 {
                                        make_pair("Kazan", 525),
                                        make_pair("Samara", 461)
                                }},

        {"Samara",              {
                                        make_pair("Ufa", 461)
                                }}
};

map<string, bool> visited;
map<string, string> parent;

string bfs(const string &from, const string &to) {
    queue<string> q;
    q.push(from);
    visited[from] = true;
    while (!q.empty()) {
        string current = q.front();
        q.pop();
        for (auto &city: cities[current]) {
            string second_city = city.first;
            if (!visited[second_city]) {
                visited[second_city] = true;
                q.push(second_city);
                parent[second_city] = current;
            }
        }
    }
    string city = to;
    string way = to;
    while (!parent[city].empty() && parent[city] != "null") {
        way = parent[city] + " -> " + way;
        city = parent[city];
    }
    return way;
}

string dfs(const string &from, const string &to) {
    visited[from] = true;
    for (auto it = cities[from].begin(); it != cities[from].end(); it++) {
        string current = it->first;
        if (!visited[current]) {
            parent[current] = from;
            dfs(current, to);
        }
    }
    string city = to;
    string way = to;
    while (!parent[city].empty() && parent[city] != "null") {
        way = parent[city] + " -> " + way;
        city = parent[city];
```

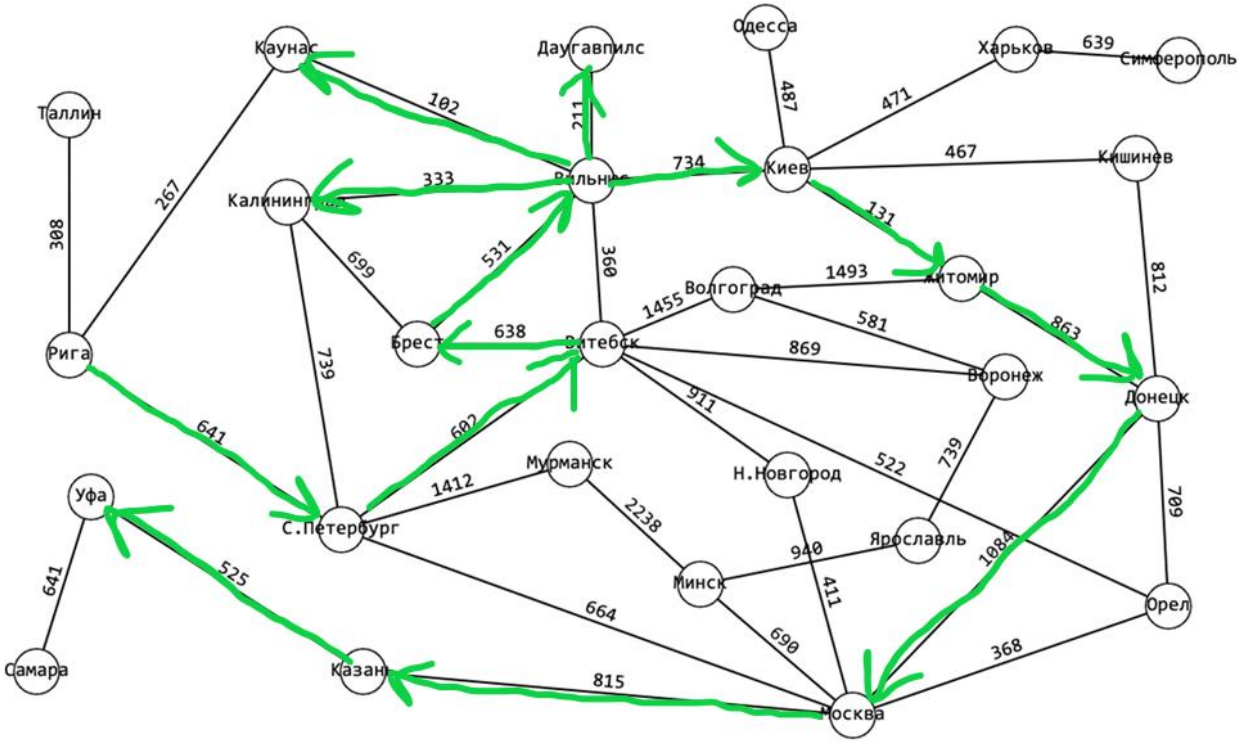
```
    }
    return way;
}

string dls(const string &from, const string &to, int limit) {
    visited[from] = true;
    if (limit == 0)
        return "Way not found";
    if (from == to)
        return "true";
    for (auto it = cities[from].begin(); it != cities[from].end(); it++) {
        string current = it->first;
        if (!visited[current]) {
            parent[current] = from;
            if (dls(current, to, limit - 1) == "true")
                return "true";
        }
    }
    if (parent[to].empty() || parent[to] == "null") return "Way not found";
    string city = to;
    string way = to;
    while (!parent[city].empty() && parent[city] != "null") {
        way = parent[city] + " -> " + way;
        city = parent[city];
    }
    return way;
}

string iddfs(const string &from, const string &to) {
    for (int i = 1;; i++) {
        clear_visited();
        if (dls(from, to, i) != "Way not found")
            break;
    }
    if (parent[to].empty() || parent[to] == "null") return "Way not found";
    string city = to;
    string way = to;
    while (!parent[city].empty() && parent[city] != "null") {
        way = parent[city] + " -> " + way;
        city = parent[city];
    }
    return way;
}

void bfs_modified(queue<string> *q, set<string> *visited_map, map<string,
string> *parent_map) {
    string current = q->front();
    q->pop();
    for (auto it = cities[current].begin(); it != cities[current].end();
it++) {
        string to = it->first;
        if (visited_map->count(to) == 0) {
            visited_map->insert(to);
            q->push(to);
            (*parent_map)[to] = current;
        }
    }
}

string is_intersecting(const set<string> &s_visited, const set<string>
&t_visited) {
    vector<string> common_data;
    set_intersection(s_visited.begin(), s_visited.end(), t_visited.begin(),
t_visited.end(),
```

```
                    back_inserter(common_data));
    if (!common_data.empty())
        return common_data.back();
    else
        return "null";
}

string bidirectional_search(const string &from, const string &to) {
    map<string, string> s_parent, t_parent;
    queue<string> s_queue, t_queue;
    set<string> s_visited, t_visited;
    string result_way;
    s_queue.push(from);
    s_visited.insert(from);
    s_parent[from] = "null";
    t_queue.push(to);
    t_visited.insert(to);
    t_parent[to] = "null";
    string result;
    while (!s_queue.empty() && !t_queue.empty()) {
        bfs_modified(&s_queue, &s_visited, &s_parent);
        bfs_modified(&t_queue, &t_visited, &t_parent);
        string intersection = is_intersecting(s_visited, t_visited);
        if (intersection != "null") {
            result = intersection;
        }
    }
    if (result == "null") {
        return "Way not found";
    } else {
        if (!(s_parent[to].empty() || s_parent[to] == "null")) {
            string city_s = to;
            string way_s = to;
            while (!s_parent[city_s].empty() && s_parent[city_s] != "null") {
                way_s = s_parent[city_s] + " -> " + way_s;
                city_s = s_parent[city_s];
            }
            result_way += way_s;

            if (!(t_parent[to].empty() || t_parent[to] == "null")) {
                string city_t = to;
                string way_t = to;
                while (!t_parent[city_t].empty() && t_parent[city_t] !=
"null") {
                    way_t = t_parent[city_t] + " -> " + way_t;
                    city_t = t_parent[city_t];
                }
                result_way += way_t;
            }
        }
        return result_way;
    }
}
```

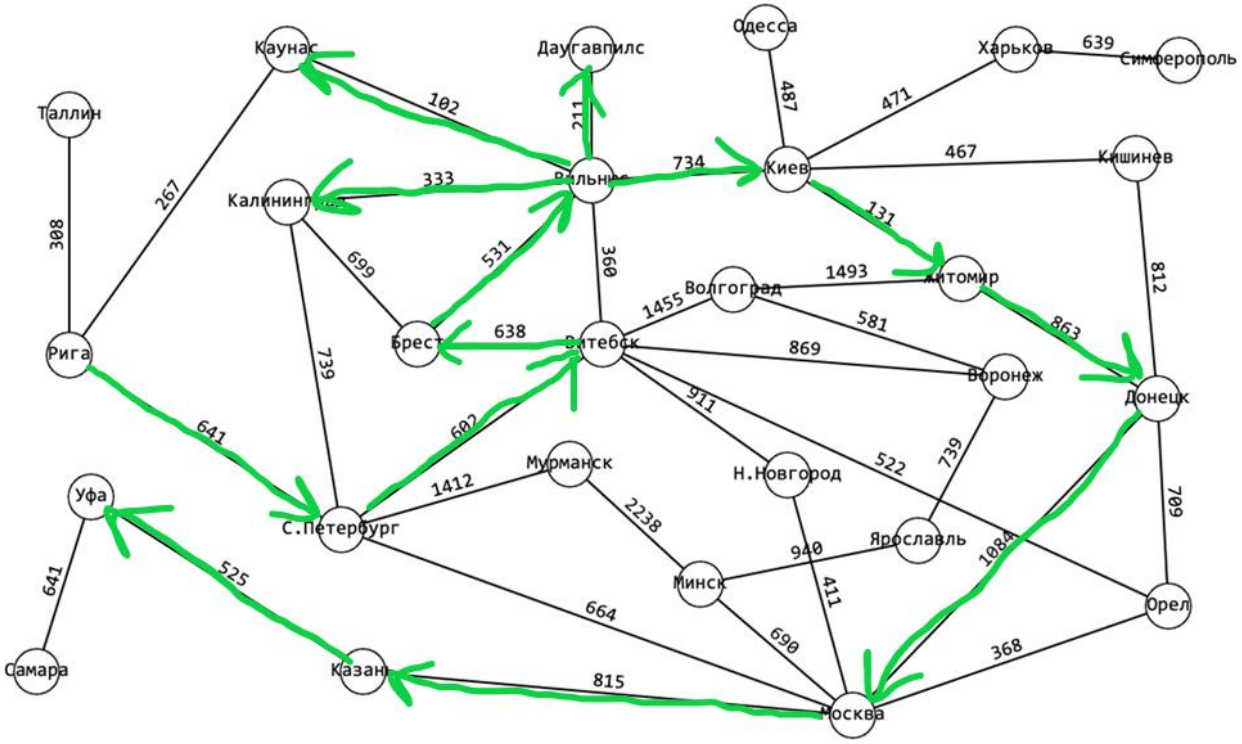
Поиск в ширину



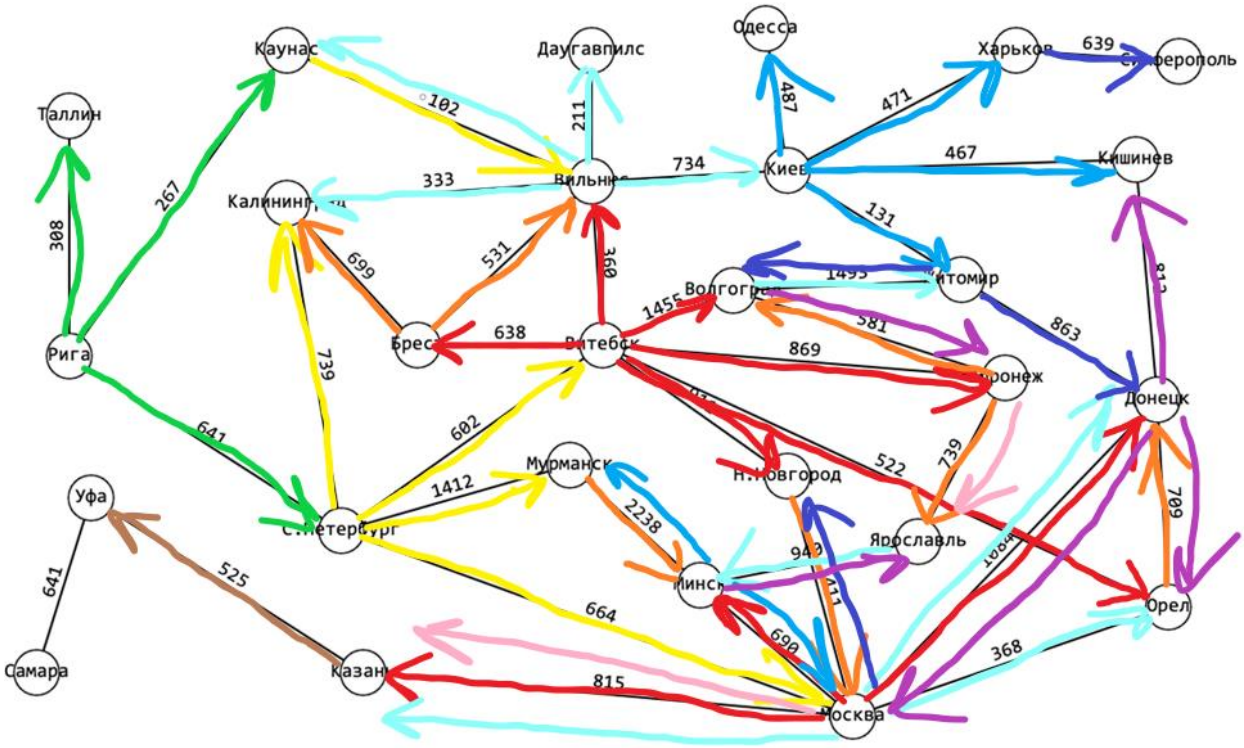
Поиск в глубину



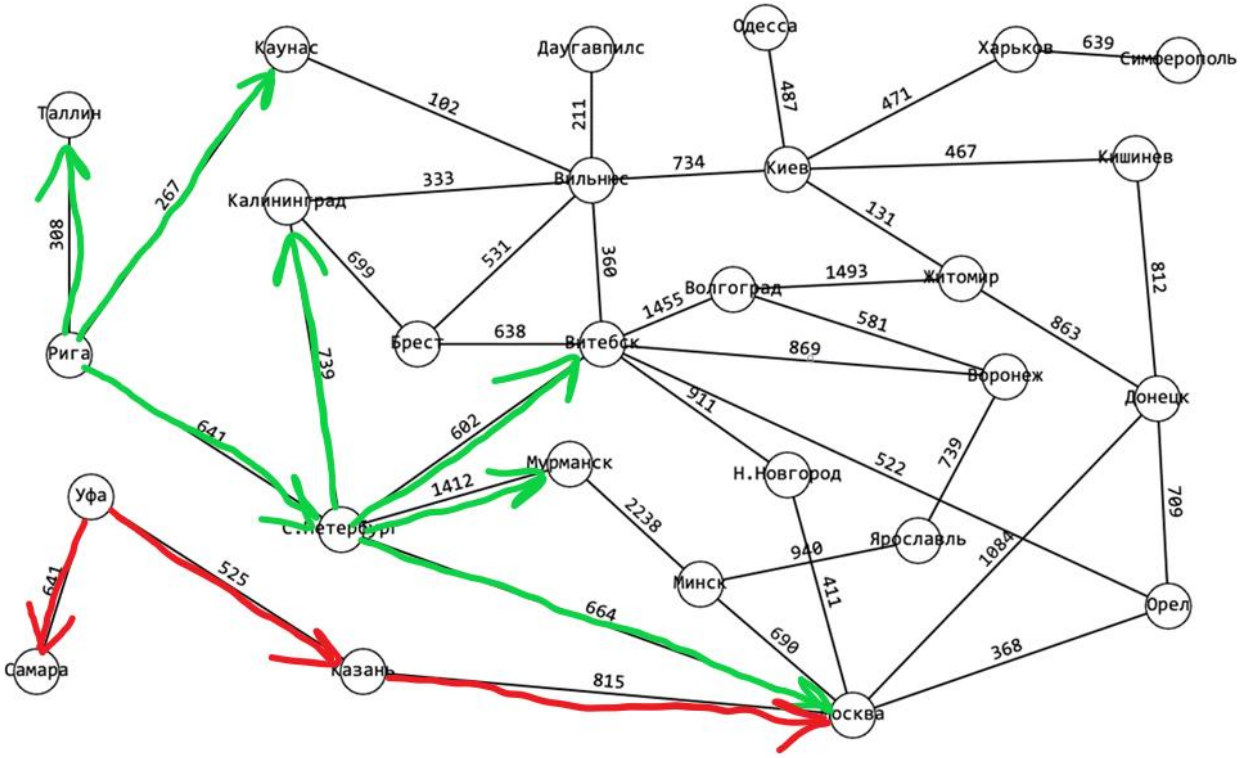
Поиск с ограничением глубины



Поиск с итеративным углублением



Двунаправленный поиск



Алгоритм	Поиск в ширину	Поиск в глубину	Поиск с ограничением глубины	Поиск с итеративным углублением	Двунаправленный поиск
Сложность	O(N + E)	O(N + E)	O(N + E)	O(N + E)	O(N + E)

Выводы:

В данной ситуации все алгоритмы поиска имеют одинаковую асимптотическую сложность, но в силу входных данных и особенностей каждого алгоритма – поиск с итеративным углублением будет занимать больше времени, чем остальные алгоритмы.

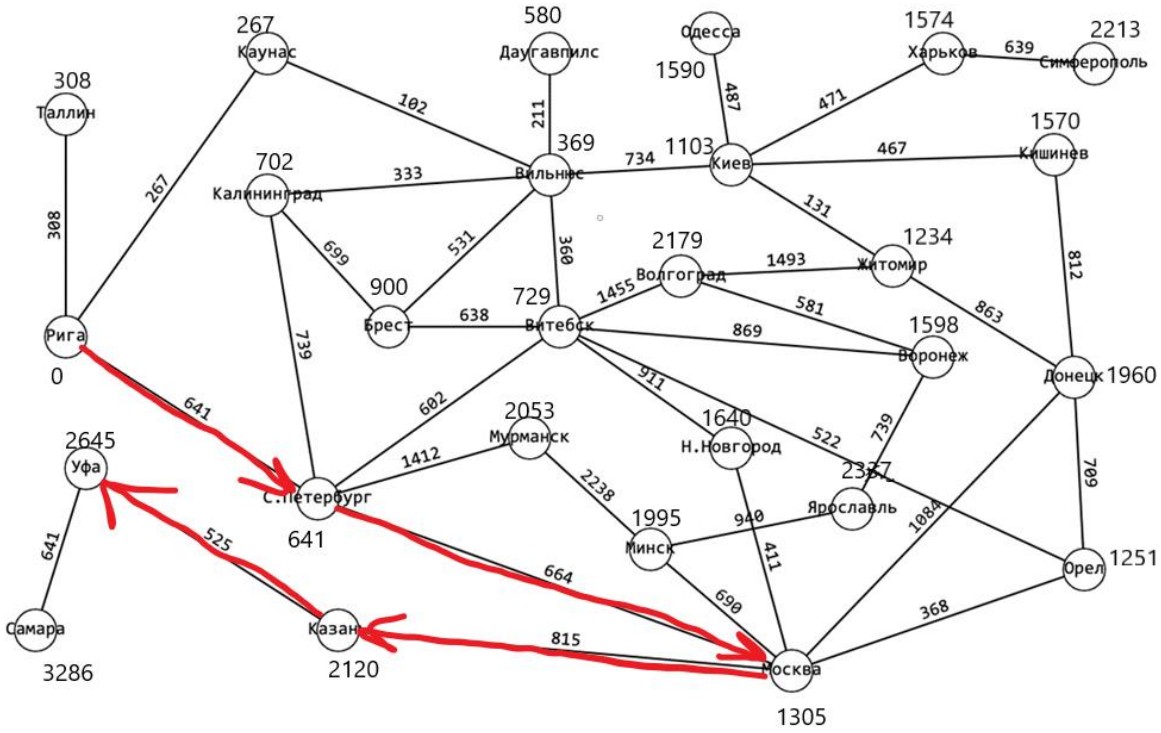
Информативный поиск

```
map<string, map<string, int>> straight_ways = {
        {"Ufa", {
                make_pair("Brest", 2147),
                make_pair("Vilnius", 1962),
                make_pair("Vitebsk", 1641),
                make_pair("Voronezh", 1168),
                make_pair("Volgograd", 1035),
                make_pair("Nizhniy Novgorod", 774),
                make_pair("Daugavpils", 1860),
                make_pair("Kaliningrad", 2260),
                make_pair("Kaunas", 2044),
                make_pair("Kiev", 1777),
                make_pair("Zhytomyr", 1907),
                make_pair("Donetsk", 1462),
                make_pair("Chisinau", 2076),
                make_pair("Saint-Petersburg", 1637),
                make_pair("Riga", 1992),
                make_pair("Moscow", 1169),
                make_pair("Kazan", 451),
                make_pair("Minsk", 1838),
                make_pair("Murmansk", 1967),
                make_pair("Orel", 1319),
                make_pair("Odessa", 1993),
                make_pair("Tallin", 1943),
                make_pair("Kharkov", 1438),
                make_pair("Simferopol", 1900),
                make_pair("Yaroslavl", 1045),
                make_pair("Samara", 421),
        }}
};

map<string, bool> visited;

pair<string, int> greedy_best_first_search_algorithm(const string &from,
const string &to) {
    string current = from;
    visited[from] = true;
    string way = " -> ";
    int way_length = 0;
    while (current != to && !current.empty()) {
        if (current != from) way += current + " -> ";
        int min_len = INT_MAX;
        string city;
        int plus_way;
        for (auto &second_city: cities[current]) {
            if (straight_ways[to][second_city.first] < min_len &&
!visited[second_city.first]) {
                plus_way = second_city.second;
                city = second_city.first;
                min_len = straight_ways[to][second_city.first];
            }
        }
        way_length += plus_way;
        current = city;
        visited[current] = true;
    }
    if (current != to) return make_pair("Way is not found", 0);
    else return make_pair(from + way + to, way_length);
}

int a_star_search_algorithm(const string &from, const string &to) {
```

```
    map<int, string> mapa;
    int function = 0;
    mapa[function] = from;
    map<string, int> minimum;
    while (!mapa.empty()) {
        auto it = mapa.begin();
        int current_function = it->first;
        if (it->first != 0) {
            current_function -= straight_ways[to][it->second];
        }
        string current_city = it->second;
        mapa.erase(it);
        visited[current_city] = true;
        if (current_city == to) {
            return current_function;
        }
        for (const auto &iter: cities[current_city]) {
            string city = iter.first;
            function = current_function + straight_ways[to][city] +
iter.second;
            if (!visited[city] && (minimum.count(city) == 0 || minimum[city]
> function)) {
                minimum[city] = function;
                parent[city] = current_city;
                mapa[function] = city;
            }
        }
    }
    return INT_MAX;
}
```

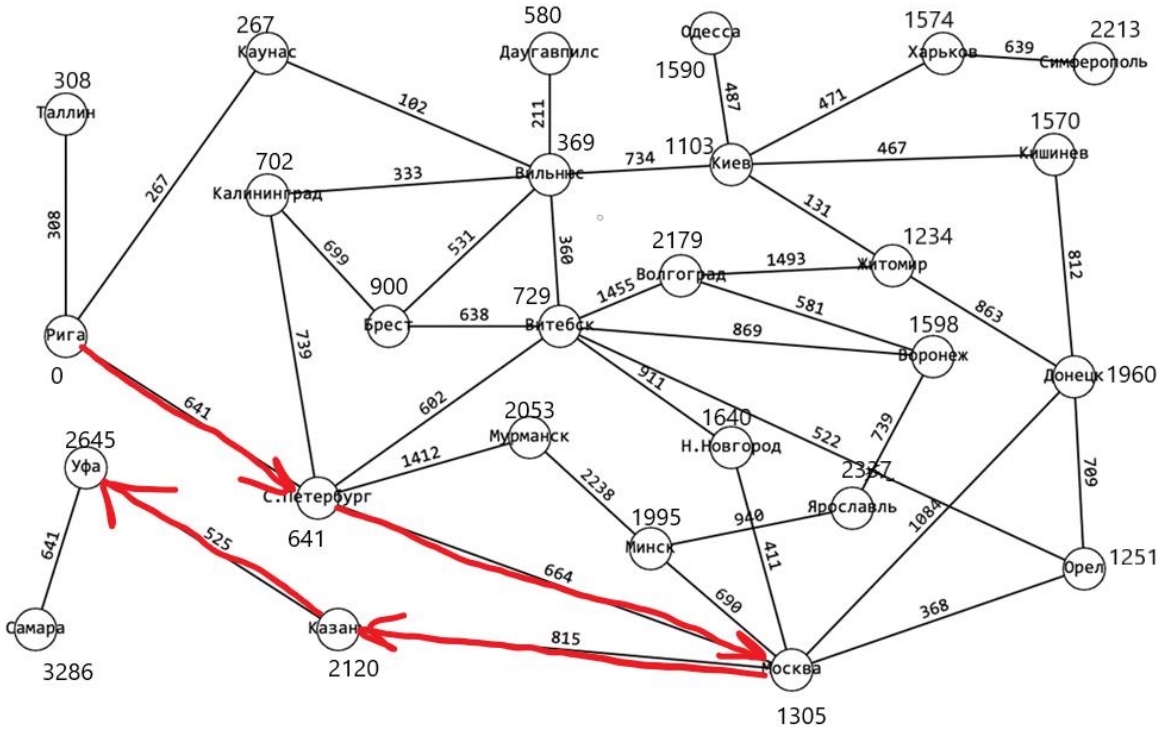
Изначально самый короткий путь:

Рига -> Санкт-Петербург -> Москва -> Казань -> Уфа 2645



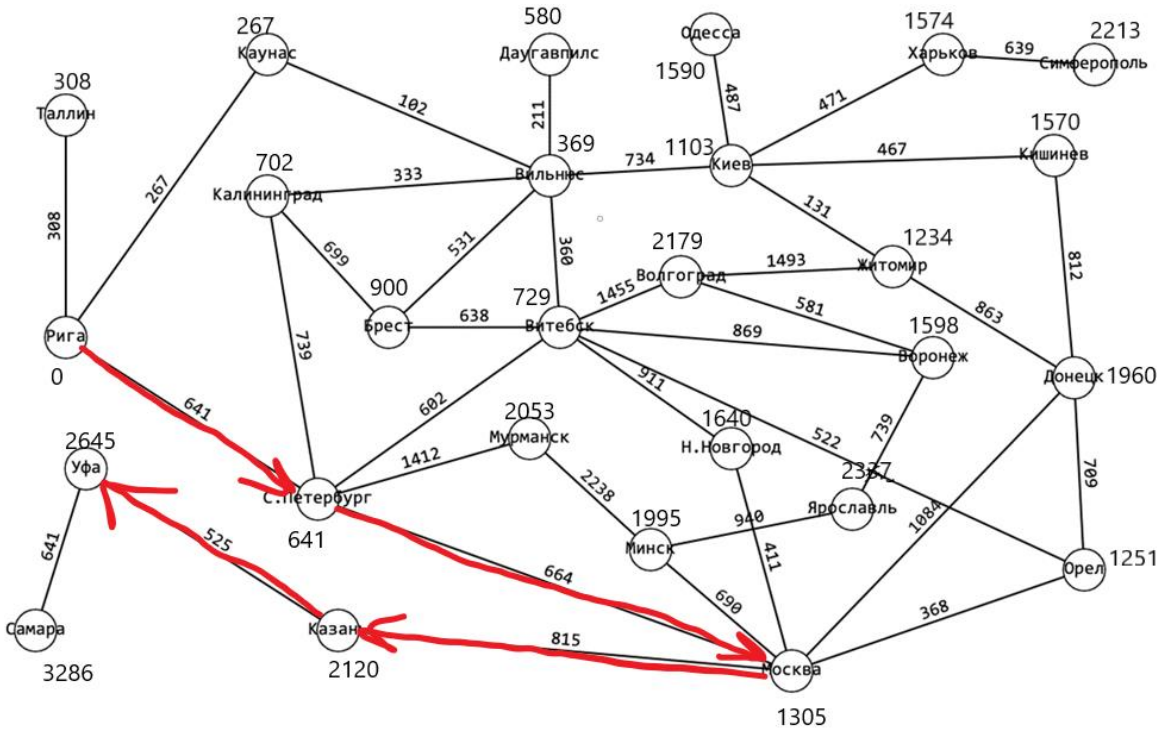
Жадный поиск

Рига -> Санкт-Петербург -> Москва -> Казань -> Уфа 2645



A*

Рига -> Санкт-Петербург -> Москва -> Казань -> Уфа 2645



Выводы:

Алгоритм	Жадный поиск	A*
Сложность	O(N)	O(N^M) N - глубина решения M – фактор ветвления

Жадный алгоритм поиска работает быстрее, чем A*. В случае моих входных данных оба алгоритма оказались точными (выбрали верный путь) и одинаково быстрыми, но в случае же других данных существует вероятность, что жадный поиск выберет не максимально точный (не наикратчайший по длине пути), а A* будет работать дольше в силу увеличения ветвления (в сравнении с жадным). A* выполняет условия полноты и оптимальности, а это значит, что найденное решение обязательно будет оптимальным из всех.