

Теория графов и её приложения

Отчёт по проектному заданию

Козырев С. А., Куклин Д. В.

Факультет прикладной математики — процессов управления
Санкт-Петербургский государственный университет

6 июня 2020 г.



Постановка задачи

Требовалось:



Постановка задачи

Требовалось:

- 1 построить граф дорог Российского города,



Постановка задачи

Требовалось:

- ① построить граф дорог Российского города,
- ② оценить удобство размещения зданий,



Постановка задачи

Требовалось:

- ① построить граф дорог Российского города,
- ② оценить удобство размещения зданий,
- ③ спланировать размещение зданий.



Что мы выбрали

① C++



Что мы выбрали

- ① C++ , так как
 - большинство проектов написаны либо на C, либо на C++,



Что мы выбрали

- ① C++ , так как
 - большинство проектов написаны либо на C, либо на C++,
 - среди остальных C++ является наиболее быстрым,



Что мы выбрали

① C++ , так как

- большинство проектов написаны либо на C, либо на C++,
- среди остальных C++ является наиболее быстрым,
- Python простой,



Что мы выбрали

- ① C++ , так как
 - большинство проектов написаны либо на C, либо на C++,
 - среди остальных C++ является наиболее быстрым,
 - Python простой,
- ② *libosmium*,



Что мы выбрали

- ① C++ , так как
 - большинство проектов написаны либо на C, либо на C++,
 - среди остальных C++ является наиболее быстрым,
 - Python простой,
- ② *libosmium*,
- ③ Нижний Новгород.



Спецификация *OSM* определяет следующие структуры данных:

- *Node* (узел),
- *Way* (путь),
- *Relation* (отношение).



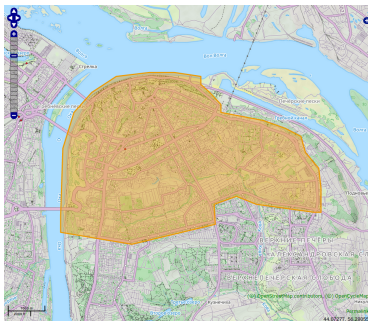


Рис.: Карта центра Нижнего Новгорода



Добавление путей

```
unordered_map<uint64_t, bool> marked {};  
  
void way(Way& way) noexcept {  
    if (!way.has_key("highway")) { return; }  
    for (auto& node: way.nodes()) {  
        if (marked.contains(node)) {  
            marked[node] |= true;  
        } else {  
            marked.insert({ node, false });  
        }  
    }  
}
```



Добавление путей

```
for (auto& curr: way.nodes()) {  
    if (curr != first && curr != last) {  
        if (marked.at(curr)) {  
            auto d = ... // One-way or two-way.  
            auto w = haversine(pred, curr);  
            routes.add_edge({ pred, curr }, w, d);  
            pred = &curr;  
        }  
    }  
}  
  
auto w = haversine(pred, last), d = ...;  
routes.add_edge({ pred, last }, w, d);
```



Расстояние между узлами

Расстояние между двумя узлами можно вычислить, используя формулу гаверсинуса

$$\eta(\Theta) = \eta(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \eta(\lambda_2 - \lambda_1),$$
$$d = 2r \arcsin(\sqrt{\eta(\Theta)}),$$

где Θ — центральный угол, $\varphi_{1,2}$ — широты в радианах, $\lambda_{1,2}$ — высоты в радианах и $\eta(\theta) = \sin^2\left(\frac{\theta}{2}\right)$ для произвольного угла θ .



Структура Node

```
struct Node {  
private:  
    using Angle = long double;  
    uint64_t m_id = 0;  
    Angle m_latitude = 0;  
    Angle m_longitude = 0;  
};
```



Структура Graph

```
struct Graph {  
private:  
    using Edge = pair<Node, Node>;  
    using OutEdges = unordered_map<Node, Distance>;  
    using AdjList = unordered_map<Node, OutEdges>;  
  
    AdjList m_data {};  
};
```



Для каждого здания на карте

- ① вычисляем барицентр здания,
- ② находим ближайший к зданию узел дороги,
- ③ в структуре *Building* связываем со зданием найденный узел.



Структура Building

```
struct Building {  
private:  
    enum class Type {  
        House,  
        Facility  
    };  
  
    uint64_t m_id = 0;  
    Type m_type;  
    Angle m_latitude = 0;  
    Angle m_longitude = 0;  
    Node m_closest_node; // Closest road node.  
};
```



Структура Map

```
struct Map {  
    struct Path {  
        private:  
            Building m_from, m_to;  
            Distance m_distance;  
    };  
  
    struct TracedPath: public Path {  
        private:  
            vector<Node> m_trace;  
    };  
  
private:  
    Buildings m_buildings;  
    Graph m_graph;  
};
```



Проблема производительности

Очевидно, карта составляется долго.

Libosmium каждый раз производит чтение карты и заполнение структур.

Вопрос итоговой сложности является открытым, но любой желающий может получить на него ответ, изучив исходный код библиотеки.



Мы обошли проблему, применив кэширование к структуре Map.

```
template<typename T>
bool serialize(fs::path& filename, T&& data) {
    ofstream binary { filename };
    boost::binary_oarchive archive { binary };
    archive << data;
    binary.close();
    return true;
}
```



```
template<typename T>
bool deserialize(fs::path& filename, T&& data) {
    if (!fs::exists(filename)) { return false; }
    ifstream binary { filename };
    boost::binary_iarchive archive { binary };
    archive >> data;
    binary.close();
    return true;
}
```



Задача

Требовалось выбрать M объектов инфраструктуры и N жилых зданий и оценить удобство их размещения, используя расстояние как метрику.

```
const vector<string> houses =  
    { "apartments", "bungalow", "cabin", "detached",  
      "dormitory", "farm", "ger", "hotel", "house",  
      "houseboat", "residential", "terrace" };  
const vector<string> facilities =  
    { "fire_station", "hospital", "retail",  
      "kiosk", "supermarket" };
```



Выбор ближайшего узла

Для каждого здания запоминаем ближайший узел.

```
void way(Way& way) noexcept {  
    const auto location = barycenter(way);  
    auto closest = min_element(routes.nodes(),  
        [&](const auto& lhs, const auto& rhs) {  
        return  
            haversine(lhs, location) <  
            haversine(rhs, location);  
        });  
    auto b = Building { way, location, closest };  
    buildings.push(b);  
}
```



Алгоритм Дейкстры

Нами был реализован алгоритм Дейкстры с временной сложностью $O(n \log n)$.

Используя алгоритм, несложно как и определить ближайшие объекты, так и построить дерево кратчайших путей.



```
auto minmax(const Map& map, const Buildings& from,
            const Buildings& to) {
    unordered_map<Building, Distance> furthest {};
    for (auto& f: from) {
        auto paths = map.shortest_paths(f, to);
        furthest[f] = max_element(
            paths.cbegin(), paths.cend(),
            [](const auto& lhs, const auto& rhs) {
                return
                    lhs.distance() > rhs.distance() &&
                    lhs.distance() < INF;
            }).distance();
    }
}
```



```
auto result = min_element(  
    from.cbegin(), from.cend(),  
    [&](const auto& lhs, const auto& rhs) {  
        return furthest[lhs] < furthest[rhs];  
    });  
return result;  
}
```



Иерархическая кластеризация

- вычисляем матрицу расстояний
- из каждого дома делаем кластер
- дальше добавляем новые кластеры путём объединения двух ближайших, пока не останется только один класс без предка



- вычисляем матрицу расстояний



Иерархическая кластеризация

- вычисляем матрицу расстояний
- из каждого дома делаем кластер



Иерархическая кластеризация

- вычисляем матрицу расстояний
- из каждого дома делаем кластер
- дальше добавляем новые кластеры путём объединения двух ближайших, пока не останется только один класс без предка



Матрица расстояний

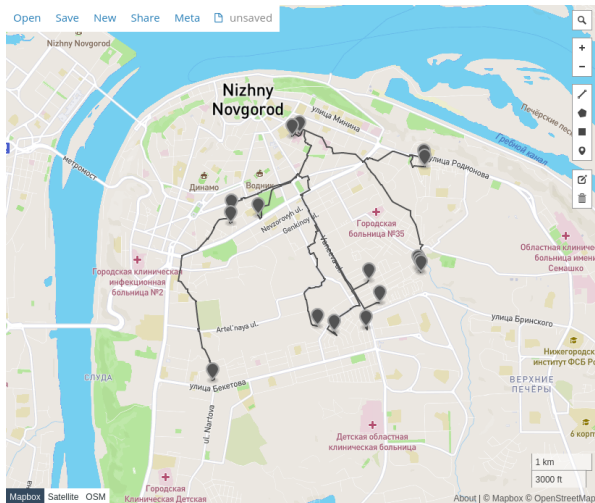
Матрицу расстояний вычисляем путём запуска алгоритма Дейкстры из каждой вершины.



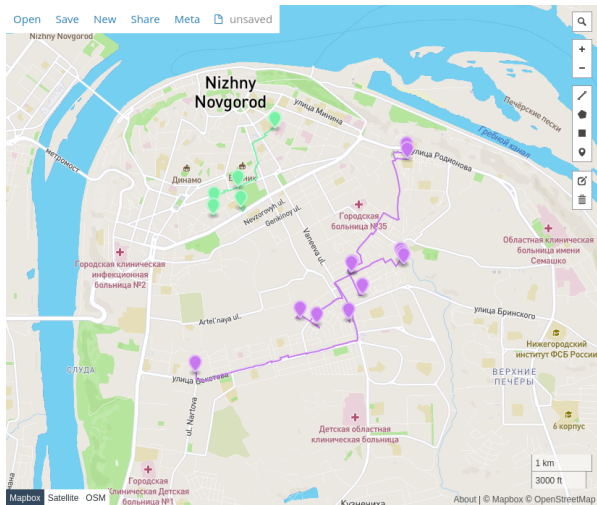
Дано 15 домов, будем разбивать их на кластеры, находить длину деревьев кратчайших путей, и сумму кратчайших путей.



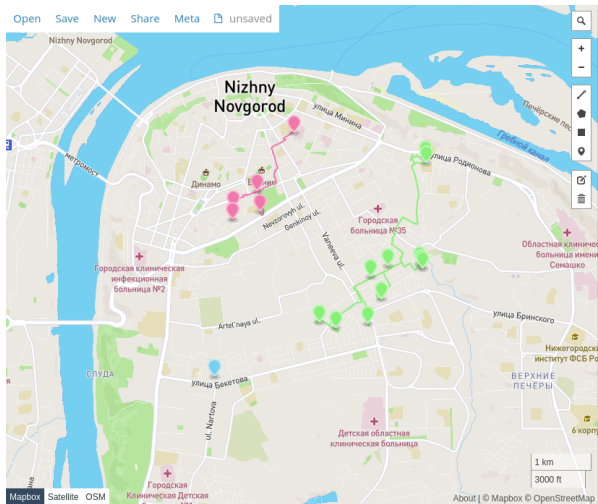
1 кластер; 32,3 км; 12,8 км



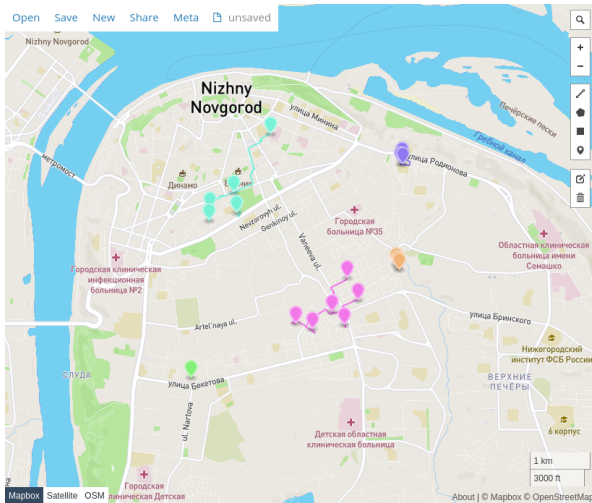
2 кластера; 13,2 км; 8,2 км



3 кластера; 11,3 км; 6,3 км



5 кластеров; 4,6 км; 3,9 км



Дендрограмма

