Поиск кратчайших путей на графах

В данной небольшой статье я хочу затронуть тему поиска кратчайших путей на графах. Опять же, здесь не будет ничего нового, просто небольшой ликбез по тому, что люди уже придумали и реализовали.

1 Формулировка задачи

Предположим, у нас есть неориентированный граф с вершинами и рёбрами с весами на нём. Нам бы хотелось найти минимальное расстояние между двумя вершинами s и t. Это можно сделать несколькими способами. Самым известным является алгоритм Дейкстры.

2 Алгоритм Дейкстры

Давайте вспомним, как он работает. Алгоритм Дейкстры начинается с установки начальной вершины и работы от этой точки. Он работает по принципу «жадного» алгоритма, что означает, что на каждом шаге он стремится минимизировать текущую общую стоимость пути.

Сначала инициализируются два множества:

- множество, содержащее уже обработанные вершины (изначально пустое);
- множество, содержащее все остальные вершины графа (изначально содержит все вершины графа).

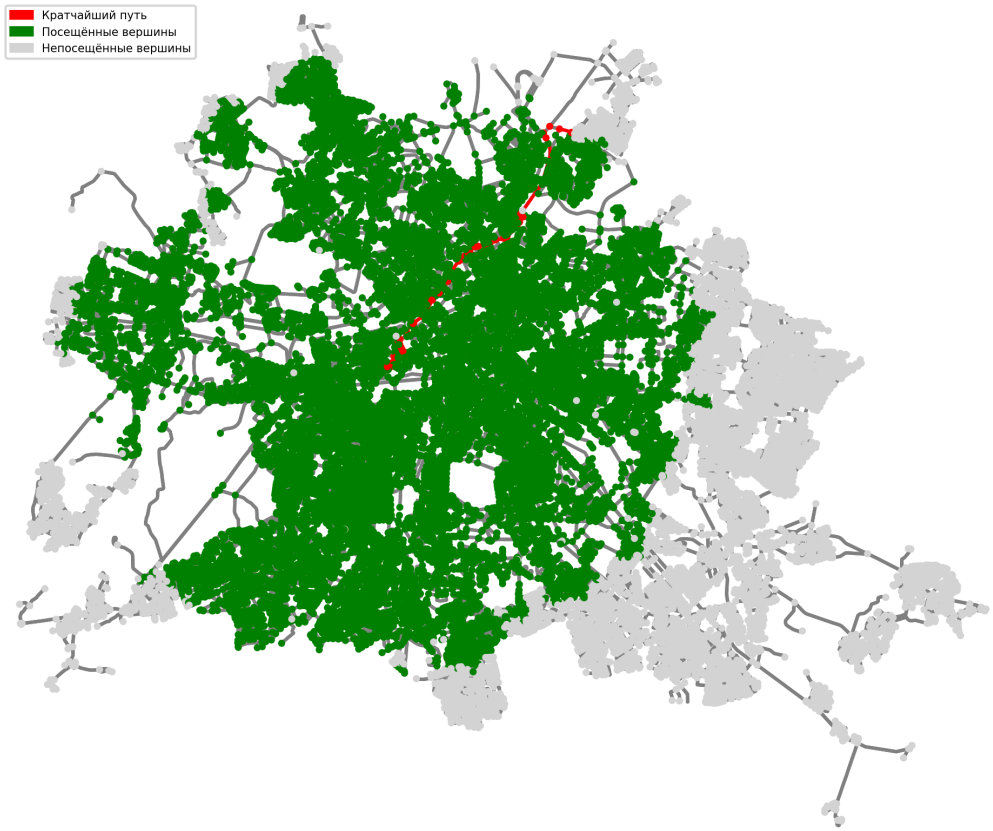
Также каждой вершине графа присваивается вес, который представляет минимальную известную стоимость пути от начальной вершины до данной. Для начальной вершины этот вес равен 0, для всех остальных вершин — ∞.

На каждом шаге алгоритм выбирает вершину из непосещённого множества с наименьшим весом, перемещает эту вершину в множество посещённых вершин и обновляет веса всех соседей выбранной вершины. Вес

соседа обновляется, если через выбранную вершину можно добраться до этого соседа с меньшей стоимостью.

Процесс продолжается, пока не будут посещены все вершины или пока не будет найден путь до конечной вершины t.

Давайте проиллюстрируем его работу на примере дорожного графа Берлина:



3 Есть ли что-то ещё?

Да, можно, например, запустить Дейкстру из обеих вершин. Этот алгоритм называется **Bidirectional Dijkstra**. Его реализация полностью аналогична Дейкстре, но теперь шаг за шагом мы поочерёдно вызываем Дейкстру для двух вершин s и t.

Это может быть полезно, если мы хотим найти минимальное расстояние между вершинами в «дорожном» графе.

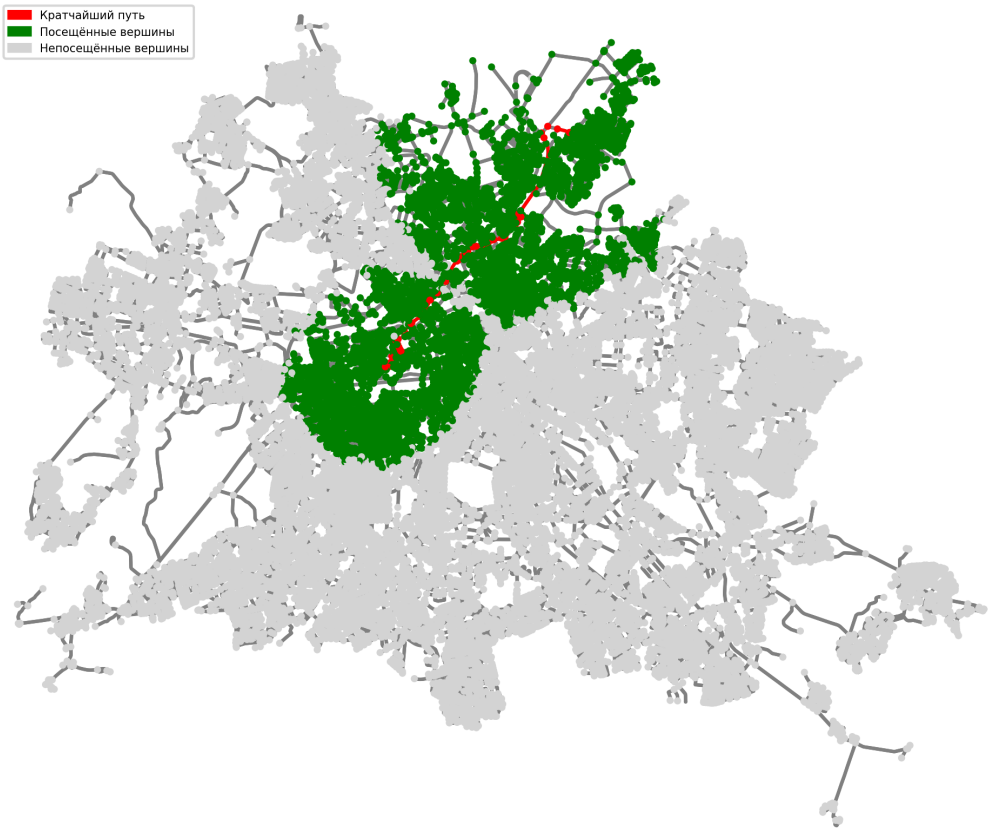
На картинке проиллюстрирован принцип работы, а также интуитивное понимание, почему этот алгоритм может оказаться лучше Дейкстры

в некоторых ситуациях.

Если оценить количество итераций, исходя из площади круга радиуса r, то можно приблизительно оценить, насколько алгоритм Bidirectional Dijkstra лучше:

$$S_1 = \pi(2r)^2,$$
$$S_2 = 2\pi r^2.$$

На дорожных картах мы уверенно получаем двухкратный выигрыш по скорости, используя для поиска пути Bidirectional Dijkstra.



4 Но не во всех ситуациях Bidirectional Dijkstra оптимальный

Приведём пример.

Пусть у нас есть n вершин. Можно себе представить, что наши вершины расположены по кругу и соединены с соседями ребром веса 1. При этом вершины s и t соседние, и между ними вес $\frac{n}{2}$.

Если мы запустим Дейкстру из вершины s, то мы за $\frac{n}{2}$ шагов справимся найти минимальный путь до вершины t. Но при реализации Bidirectional Dijkstra нам необходимо будет суммарно произвести n итераций.

На картинке выше мы можем заметить, что не всё так гладко.

Странно учитывать на дорожных картах при построении маршрута с запада на восток города, которые расположены западнее s. Вряд ли кратчайший путь будет проходить через них.

Поэтому плавно перейдём к следующему алгоритму и рассмотрим его ключевую идею — это **A***.

5 Алгоритм A*

Алгоритм A* работает на основе оценки стоимости пути до цели. Эта стоимость вычисляется как сумма двух компонент:

- $g(x)$ — уже известная стоимость пути от начальной вершины до x;
- $h(x)$ — эвристическая оценка стоимости пути от x до t. Главное, чтобы $h(x)$ было строго меньше, чем минимальное расстояние от x до t!

Сумма

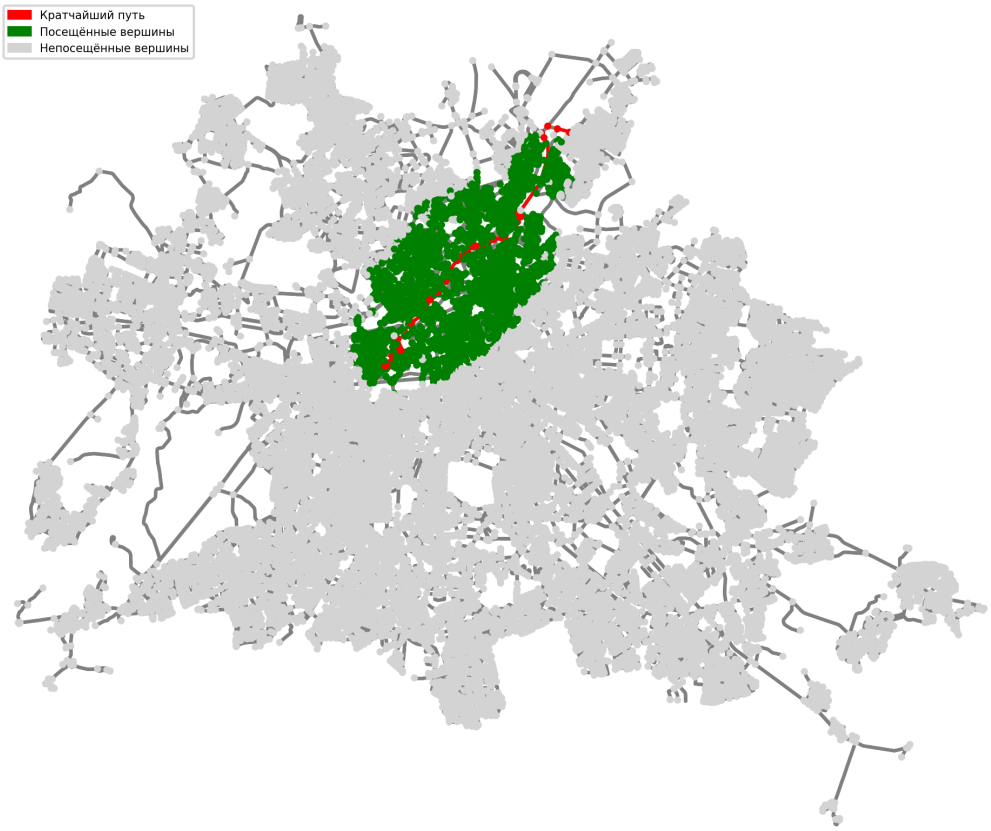
$$f(x) = g(x) + h(x)$$

даёт оценку общей стоимости пути через данную вершину. Эвристика $h(x)$ играет ключевую роль в алгоритме A*. Она должна быть адекватной, чтобы алгоритм был эффективным.

На каждом шаге алгоритм выбирает вершину с наименьшей оценкой $f(x)$ из списка открытых вершин (вершин, которые уже были обнаружены, но ещё не обработаны), затем смотрит на соседей этой вершины и обновляет их стоимости $g(y)$, если через текущую вершину можно добраться до них быстрее. Если вершина ещё не была открыта, она добавляется в список открытых вершин.

Алгоритм работает абсолютно так же, как Дейкстра, но с небольшим отличием — в виде выбора функции $f(x)$, вместо обычного расстояния $g(x)$.

Процесс продолжается, пока вершина t не станет обработанной, то есть в тот момент, когда $f(t)$ минимально среди всех открытых вершин.



В качестве h здесь используется эвклидово расстояние между городами.

Математически несложно показать, что данный алгоритм строго лучше Дейкстры, ведь не существует такой вершины u, которую мы обработали в алгоритме A*, но при этом упустили в Дейкстре.

Это следует из выбора функции $h(x)$ — она всегда меньше реального расстояния от x до t.

6 Landmarks и улучшение эвристики

А что если мы не знаем структуру нашего графа и понятия не имеем, как оценить снизу расстояние между городами? Тут на помощь нам придут **landmarks**!

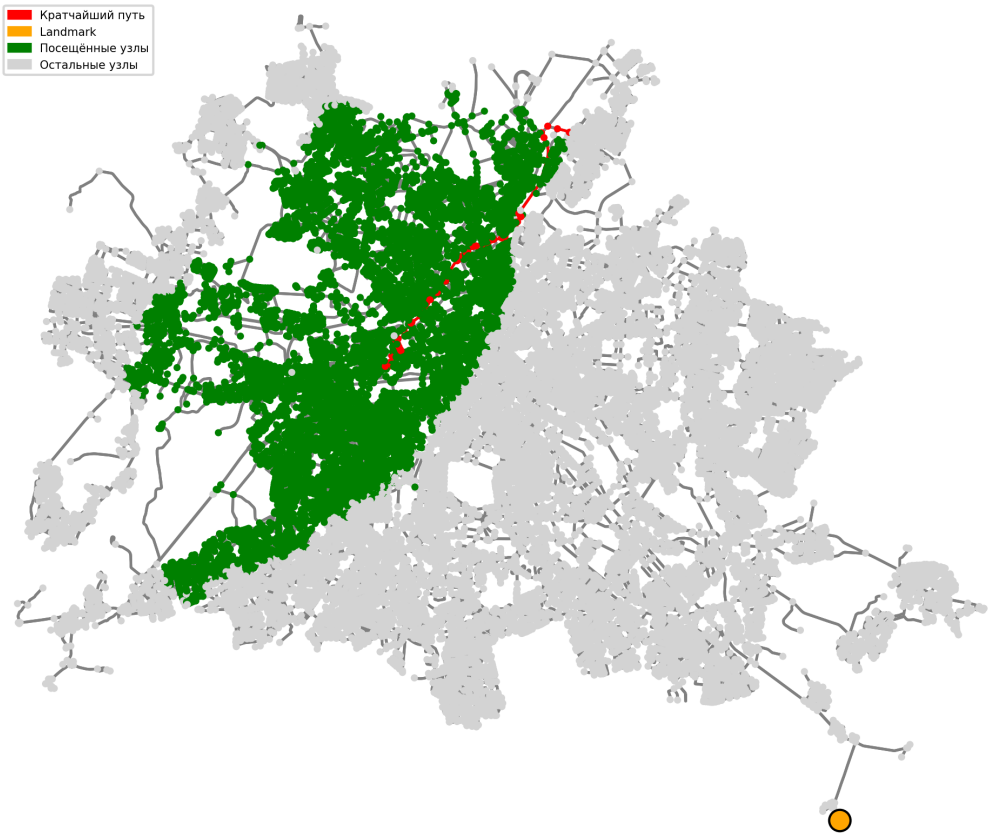
Давайте выберем любую вершину графа l, посчитаем расстояние от неё до всех остальных вершин. Теперь оценим:

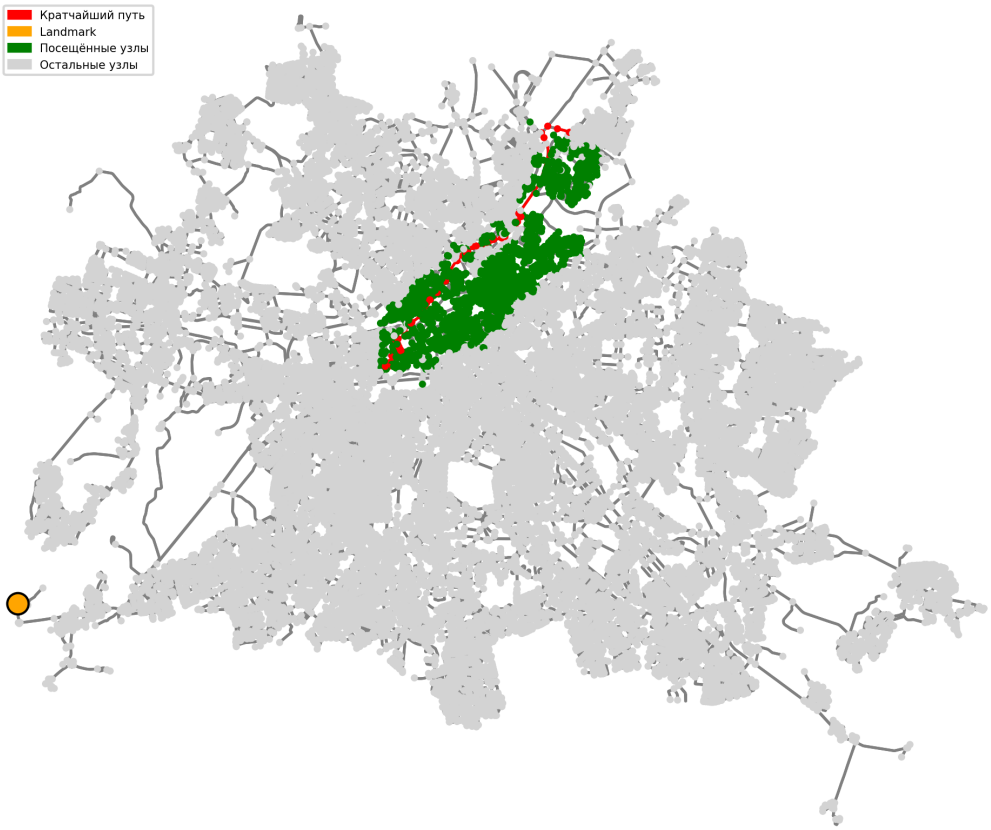
$$h(x) = |\text{dist}(x, l) - \text{dist}(t, l)|.$$

Очевидно, что $h(x)$ в данном случае действительно будет меньше $\text{dist}(x, t)$. Хочется, чтобы $h(x)$ было в действительности почти равно $\text{dist}(x, t)$. Если $h(x) = \text{dist}(x, t)$, то это нам будет гарантировать, что на каждой итерации алгоритма A* мы будем выбирать вершину, которая лежит на минимальном пути между s и t.

Можно справедливо заметить, что если landmark взять на границе графа, то это будет оптимально. Скорее всего, при такой конфигурации $h(x)$ будет аппроксимировать $\text{dist}(x, t)$.

Сравним две картинки:

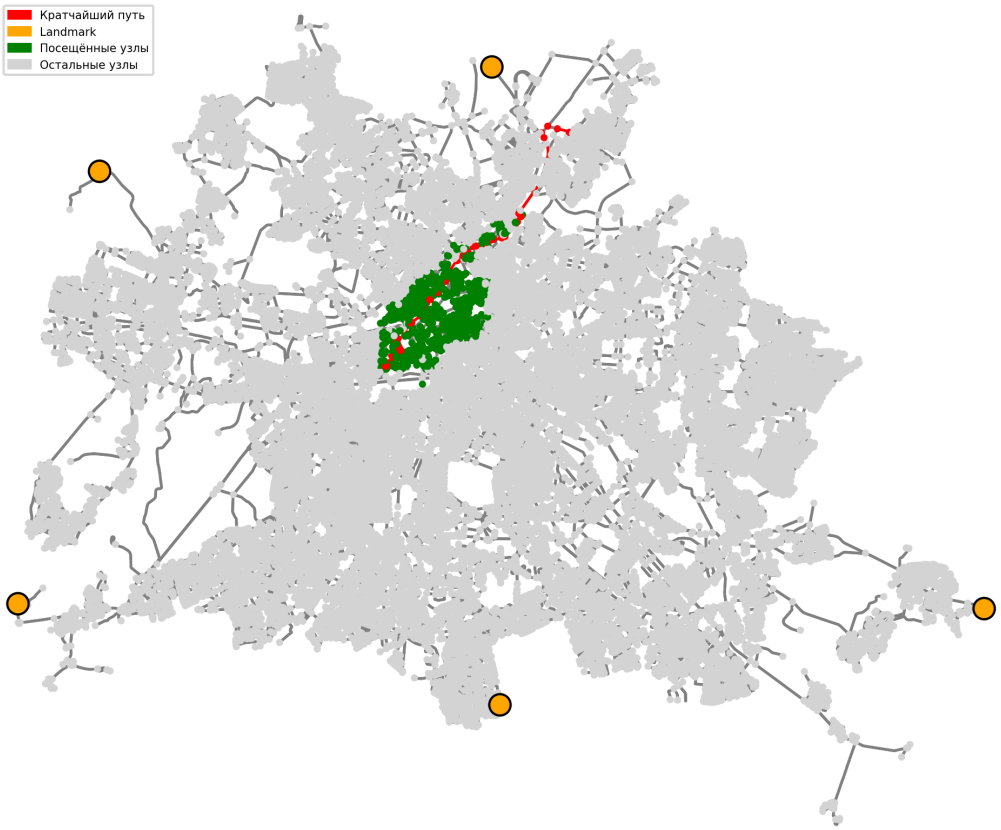




Видно, что от удачного выбора landmark многое зависит. Хотя внимательный читатель может поспорить с тем, что во второй картинке тоже есть изъян: вдруг нам необходимо посчитать расстояние между другими двумя вершинами? В таком случае landmark на первой иллюстрации может оказаться лучше.

Чтобы этого избежать, давайте добавим сразу несколько l_i и оценим:

$$h(x) = \max_i \big|\mathrm{dist}(x, l_i) - \mathrm{dist}(t, l_i)\big|.$$



Очевидно, что ситуация стала намного лучше! На этом пока что все Спасибо за прочтение!