Лекция 6
Алгоритмы на графах. Алгоритмы поиска пути на взвешенных графах

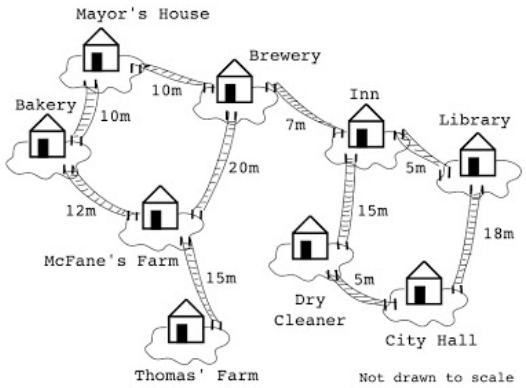
Анализ и разработка алгоритмов

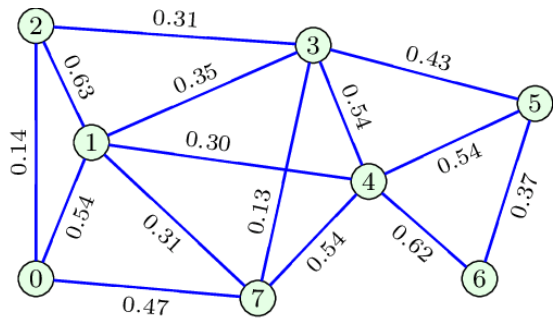
УНИВЕРСИТЕТ ИТМО

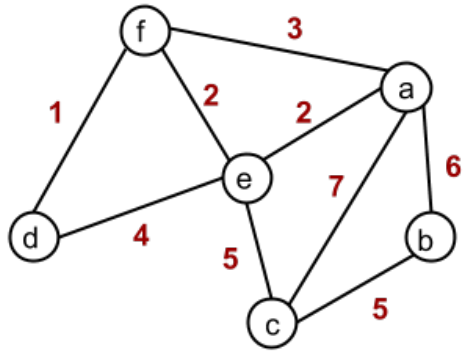
Содержание

Взвешенные графы

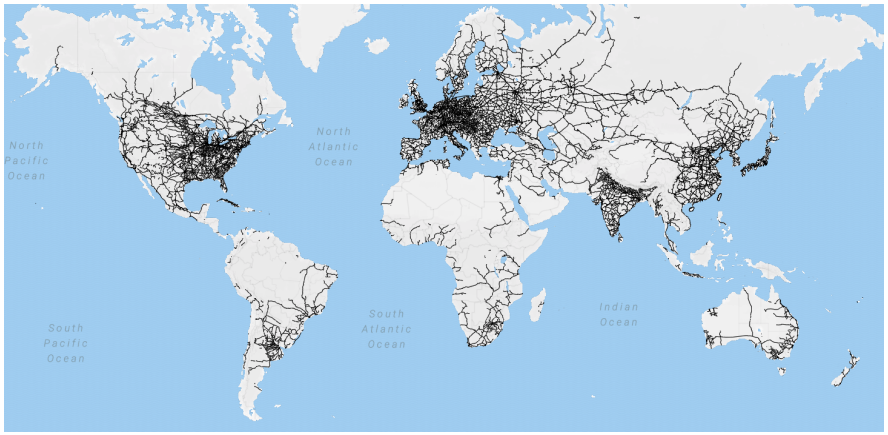
Взвешенный граф — это граф, в котором каждому ребру приписан *вес* (некоторое число).



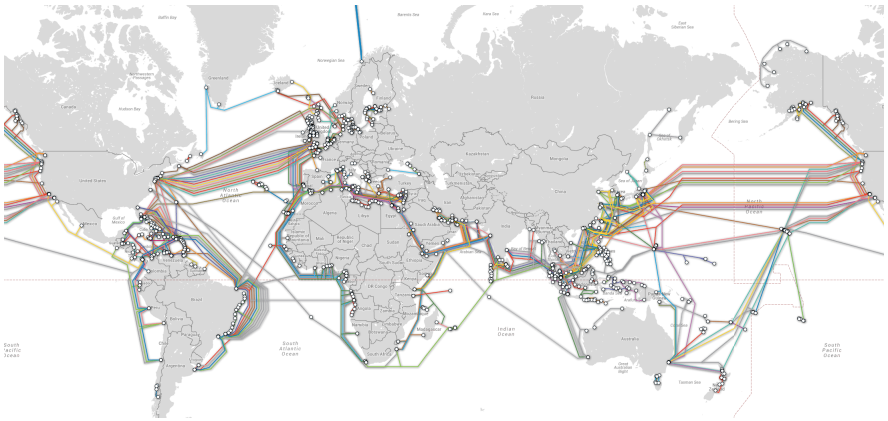




Сеть железных дорог



Подводная кабельная сеть



Алгоритм Дейкстры (DA)

Задача: для заданных графа (с *положительными* весами) и вершины-источника s, найти кратчайшие пути от s до остальных вершин.
Основная идея: алгоритм генерирует дерево кратчайших путей (SPT) с корнем s путем обработки двух множеств: одно множество содержит вершины, включенные в SPT, другое — вершины, еще не включенные в SPT. На каждом шаге алгоритм находит вершину, не включенную в SPT и имеющую минимальное расстояние от источника.

Алгоритм (сложность от $O(|V| \log |V|)$ до $O(|V|^2)$)

- Создайте множество SPT *sptSet*, содержащее вершины из SPT (для которых минимальное расстояние от s подсчитано). Изначально $sptSet = \emptyset$.
- Задайте значение расстояния от s: 0 — для s, ∞ — для остальных.
- Пока *sptSet* не содержит все вершины:
  - Выберите $u \notin sptSet$, которая имеет минимальное значение расстояния.
  - Включите u в *sptSet*.
  - Обновите значения расстояний для смежных вершин u: для каждой смежной вершины v, если сумма значения расстояния до u и веса ребра (u, v) меньше значения расстояния до v, то обновите значение для v.

Эдсгер Дейкстра в интервью с Филипом Л. Франой, Communications of ACM, 2001

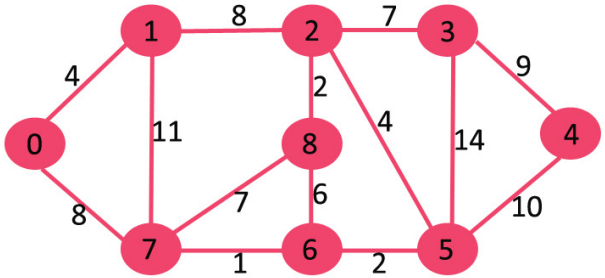
Какой самый короткий путь из Роттердама в Гронинген, в общем — из данного города в данный? Это определяет алгоритм для кратчайшего пути, который я разработал примерно за двадцать минут.

Однажды утром я ходил по магазинам в Амстердаме со своей молодой невестой и, устав, мы присели на террасе кафе, чтобы выпить чашку кофе. Там я придумал алгоритм для кратчайшего пути... Как я уже сказал, это было двадцатиминутное изобретение.

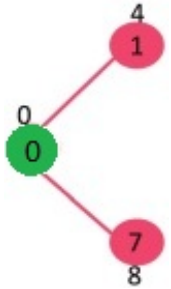
Фактически он был опубликован в 1959 году, три года спустя. Публикация все еще читаема, это, на самом деле, довольно приятно. Одна из причин, по которой это так приятно, было то, что я создал ее без карандаша и бумаги. Позже я узнал, что одним из преимуществ проектирования без карандаша и бумаги является то, что вы почти вынуждены избегать всех сложностей, которых можно избежать.

В конце концов, этот алгоритм стал, к моему великому изумлению, одним из краеугольных камней моей славы.

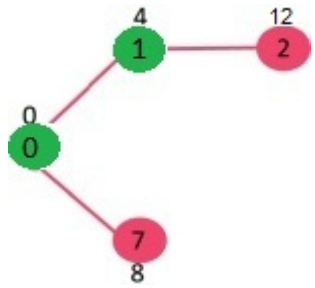
Пример



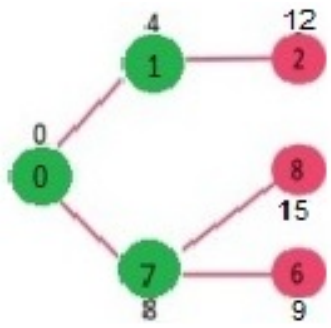
В начале $sptSet = \emptyset$, значение расстояния для источника 0 равно 0, значение расстояния для остальных вершин равно ∞.
Ниже подграф показывает вершины и значения расстояний от источника (только если они конечны). Вершины, включенные в *sptSet*, показаны зеленым цветом.



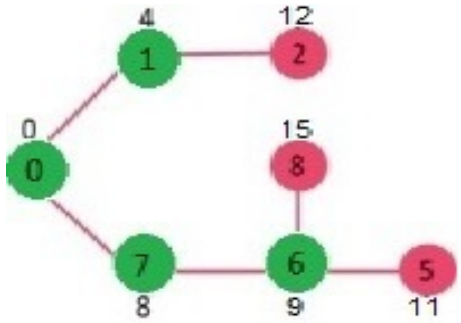
Выберите вершину $\notin$ *sptSet* с минимальным значением расстояния. Вершина 0 выбрана и включена в *sptSet*, поэтому $sptSet = \{0\}$. Обновите значения расстояний для смежных вершин вершины 0. Смежные вершины 0 — 1 и 7. Значения расстояния 1 и 7 становятся 4 и 8 соответственно.



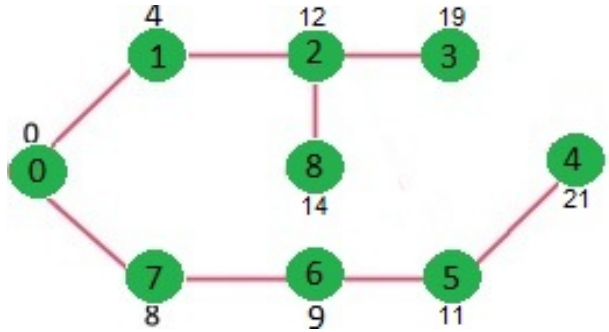
Выберите вершину $\notin$ *sptSet* с минимальным значением расстояния. Вершина 1 выбирается и включается в *sptSet*, поэтому $sptSet = \{0, 1\}$. Обновите значения расстояний смежных вершин вершины 1. Значение расстояния вершины 2 становится 12.



Выберите вершину $\notin$ *sptSet* с минимальным значением расстояния. Вершина 7 выбирается и включается в *sptSet*, поэтому $sptSet = \{0, 1, 7\}$. Обновите значения расстояний смежных вершин вершины 7. Значения расстояний вершин 6 и 8 становятся 15 и 9 соответственно.



Выберите вершину $\notin$ *sptSet* с минимальным значением расстояния. Вершина 6 выбирается и включается в *sptSet*, так что $sptSet = \{0, 1, 7, 6\}$. Обновите значения расстояний смежных вершин вершины 6. Значения расстояний вершин 5 и 8 становятся 11 и 12 соответственно.



Повторяйте вышеупомянутые шаги, пока *sptSet* не будет включать все вершины. В итоге получим SPT.

Демонстрация: пример, большой граф

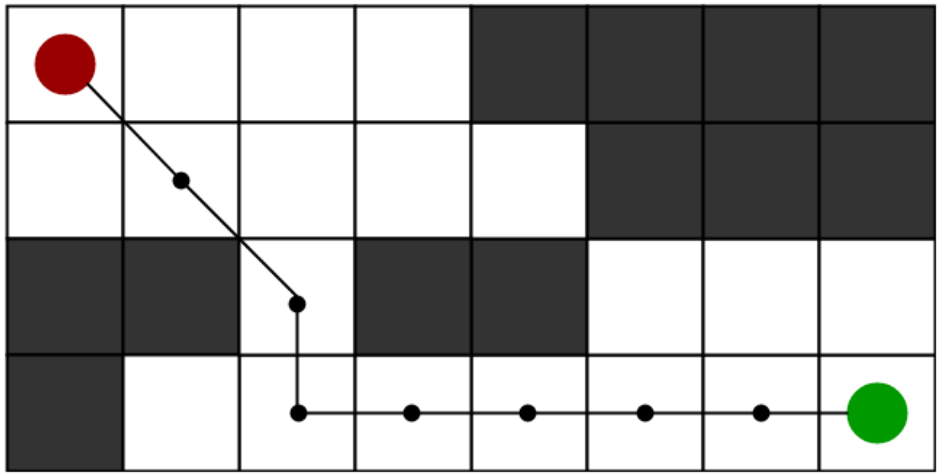
Алгоритм A*

Задача: для данных графа (с *положительными* весами), источника s и цели t найти кратчайший путь от s до t.

Основная идея: на каждой итерации A* определяет, как продлить путь, исходя из стоимости текущего пути от s до точки продления и **оценки** стоимости пути от точки продления до t. Временная сложность $O(|E|)$.

Получается DA, если вышеупомянутая оценка опущена.

Дана сетка с препятствиями, представимая в виде взвешенного графа (вершины = ячейки, взвешенные ребра = кратчайшие пути и их длины).



Вообще, перемещение в разные ячейки может иметь разную стоимость.

A* algorithm

Дана сетка с препятствиями, исходная ячейка c_s и целевая ячейка c_t. Нужно как можно быстрее достичь c_t из c_s (если возможно).

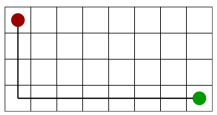
A* продлевает путь в соответствии с $f(c_k) = g(c_k) + h(c_k)$, где g считает стоимость перехода от c_s до c_k по сгенерированному пути, а h оценивает стоимость перехода от c_k до c_t.

Как рассчитать h?

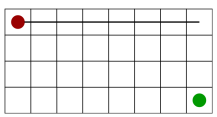
Точно (много времени): вычислите расстояние между каждой парой ячеек перед запуском A*.

Приблизительно: рассчитайте расстояния без учета препятствий, например, с помощью следующих метрик:

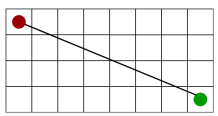
Manhattan

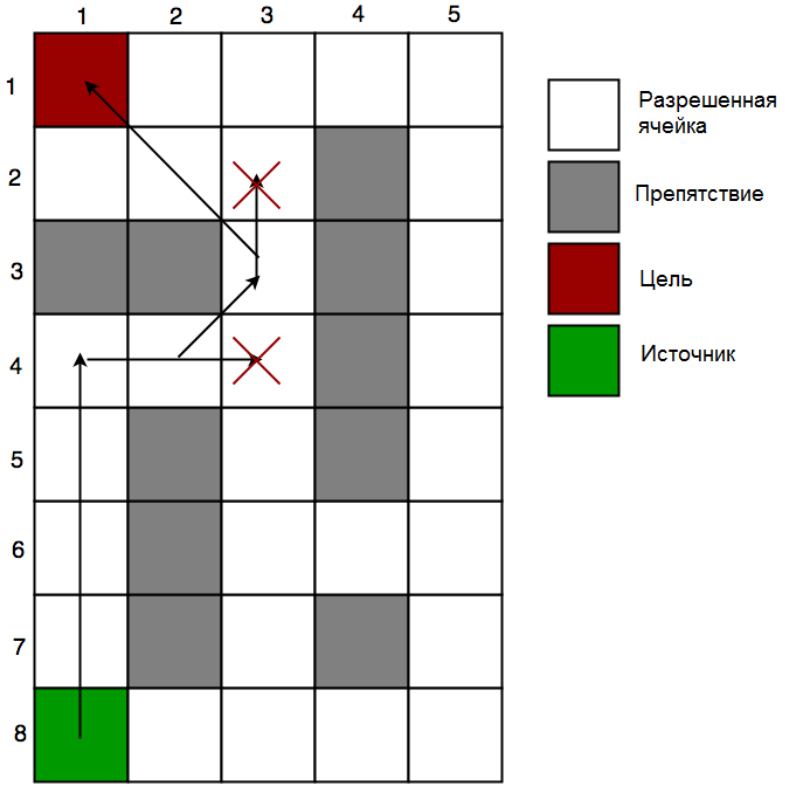


Диагональная



Евклидова





Демонстрация: A* и его приложения

Алгоритм Беллмана-Форда (BFA)

Задача: для данного взвешенного графа (с *отрицательными* весами) и источником s найти кратчайшие пути от s до всех остальных вершин.

Если граф содержит отрицательный цикл C_- (т.е. цикл, сумма ребер в котором отрицательна), достижимый из s, то кратчайшего пути не существует: *любой путь, имеющий вершину в C_-, может быть сделан короче при еще одном обходе C_-*. В таком случае BFA сообщает о C_-.

Примечание: DA не работает для отрицательных весов. BFA проще в реализации, чем DA, но сложность BFA $O(|V||E|)$, когда у DA она $O(|V|^2)$.

Идея: На i-ой итерации BFA вычисляет кратчайшие пути, которые имеют не более i ребер. Поскольку в любом простом пути не более $|V| - 1$ ребер, $i = 1, \ldots, |V| - 1$. Предполагая, что нет C_-, если мы вычислили кратчайшие пути из не более чем i ребер, то итерация по всем ребрам гарантирует получение кратчайших путей из не более чем $i + 1$ ребро. Чтобы проверить, есть ли C_-, сделайте $|V|$-ую итерацию. Если хотя бы один из кратчайших путей становится короче, то есть C_-.

Демонстрация: алгоритм пошагово, пример с C_-

Спасибо за внимание!