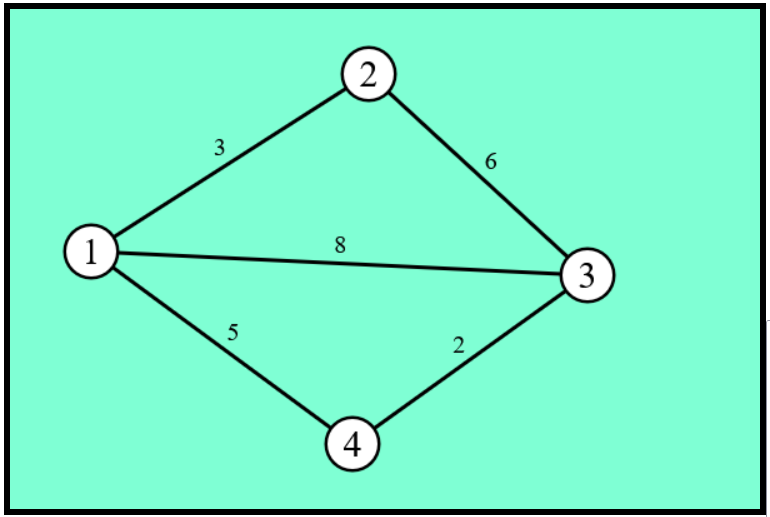
**Алгоритм Дейкстры**

**Рассмотрим следующую задачу**:

Дан взвешенный граф (т.е. у каждого ребра есть вес). Длина пути – это сумма весов рёбер на пути. Нужно найти кратчайший путь между заданными вершинами. **Веса рёбер неотрицательные.**

Например, на рисунке ниже кратчайший путь между вершинами 1 и 3 имеет вид 1-4-3 (длина пути 7).



**Почему не работает обход в ширину?**

Идея обхода в ширину была в том, чтобы перебирать вершины в порядке возрастания кратчайшего пути до них от стартовой вершины. Сначала в таком порядке перебиралась стартовая вершина, потом все вершины на расстоянии 1, потом все вершины на расстоянии 2 и т.д.

Этот порядок перебора достигался с помощью очереди, в которой соблюдались следующие условия:

1. Вершины в очереди всегда расположены по неубыванию кратчайшего расстояния до них

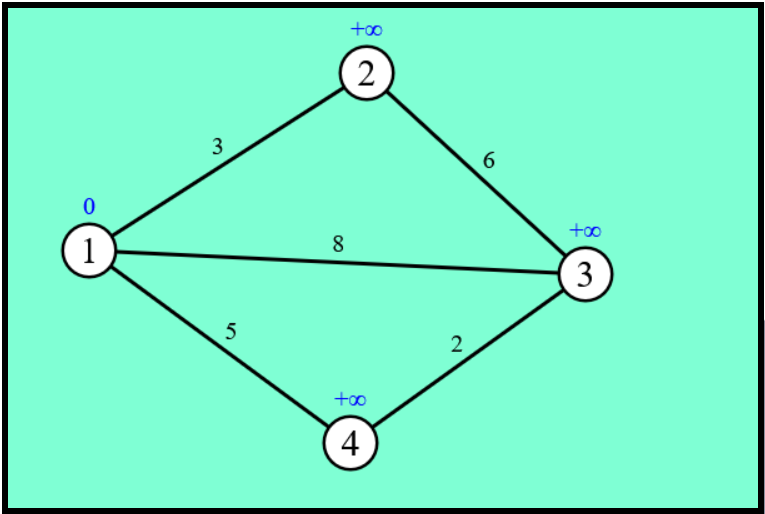
2. Расстояния до всех вершин в очереди отличаются не более чем на 1.

Алгоритм брал первую вершину в очереди (назовём её v) и перебирал все рёбра из неё в ещё не посещённые вершины. Расстояние до каждой из них на 1 больше, чем до v, так что их можно добавить в конец очереди с соблюдением этих 2 условий.

Но если запустить обход на взвешенном графе, то порядок перебора вершин будет другим. На примере выше это, например, порядок 1 2 3 4, что не соответствует идее перебирать вершины по возрастанию расстояния до них от стартовой. Правильный порядок должен быть 1 2 4 3.

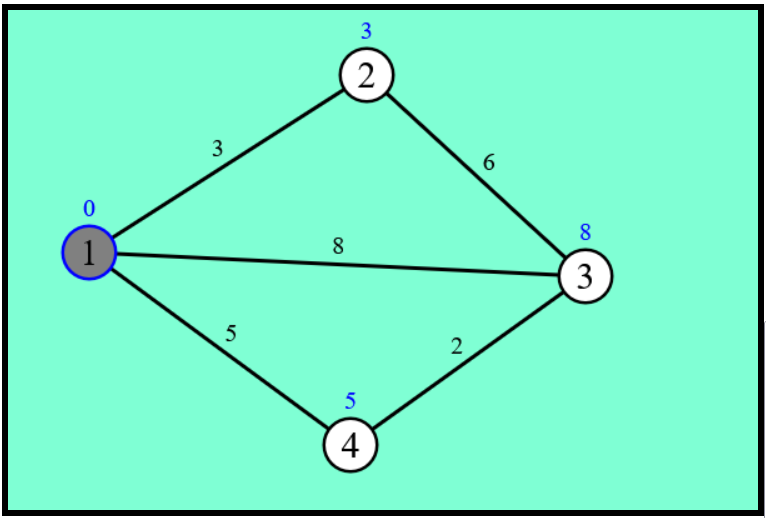
**Как изменить идею для взвешенных графов?**

Алгоритм Дейкстры использует ту же идею перебора вершин в порядке возрастания расстояния до них. Для каждой вершины будем хранить расстояние до неё от стартовой вершины (изначально расстояние до стартовой вершины равно 0, а до остальных вершин +∞). Пусть расстояние до всех вершин хранится в массиве d.

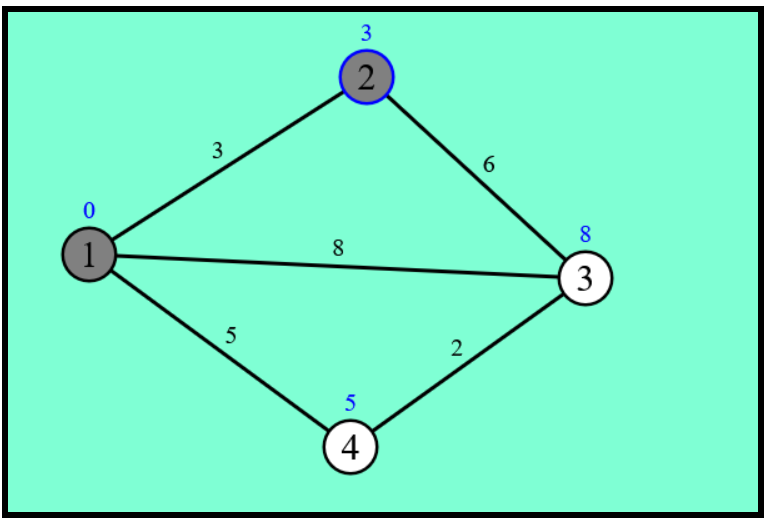


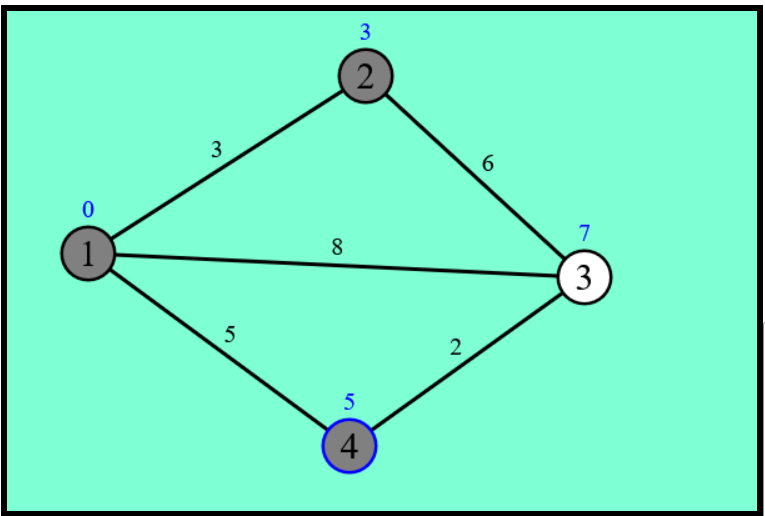
На каждом шаге алгоритма будем брать ещё не рассмотренную вершину с минимальным расстоянием (теперь решение с очередью не работает, но можно её найти перебором) и из неё обновлять расстояние до других вершин переходами по рёбрам.

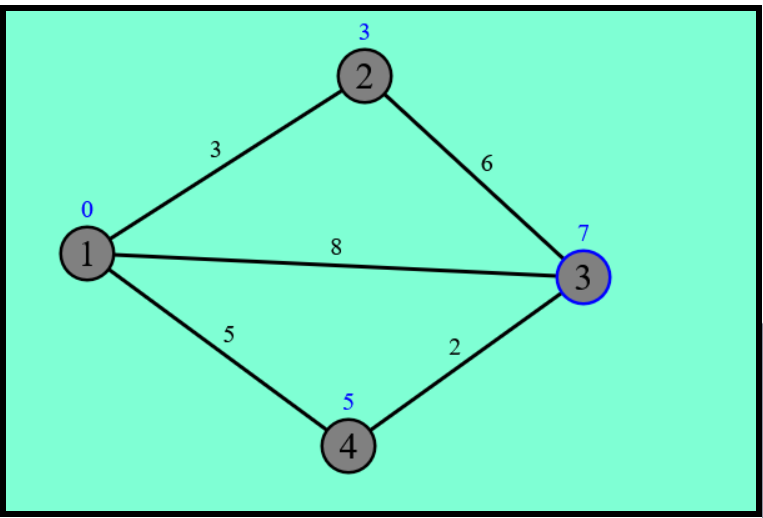
Пусть сейчас ещё не рассмотренная вершина с минимальным расстоянием – это вершина v. Отмечаем её как рассмотренную и перебираем из неё все рёбра. Если есть какое-то ребро из v в u веса w, то обновляем расстояние до вершины u по формуле d[u] = min(d[u], d[v] + w).



Дальше продолжаем искать ещё не рассмотренную вершину с минимальным расстоянием.







Теперь можно ещё раз сформулировать весь алгоритм:

1. Заведём массив d, в котором будут храниться кратчайшие расстояния от стартовой вершины до всех вершин. Изначально расстояние до стартовой вершины равно 0, до всех остальных +∞.

2. Пока есть ещё не рассмотренные вершины, выбираем вершину v с минимальным значением d[v] (среди ещё не рассмотренных). Отмечаем её как рассмотренную. Перебираем все рёбра из вершины v. Если ребро ведёт в вершину, текущее расстояние до которой больше, чем расстояние до v плюс вес ребра, то обновляем расстояние до неё.

Пусть n – число вершин, m – число рёбер. Тогда алгоритм будет работать за время **O(n2 + m),** потому что n раз запускается поиск минимальной вершины, а также нужно пройти по всем m рёбрам.

**Как ускорить алгоритм**

Вместо того чтобы n раз искать минимум перебором, можно поддерживать какую-нибудь структуру данных из STL, например, set.

Будем хранить все ещё не рассмотренные вершины в виде пар (d[v], v) в структуре set, которая поддерживает операции добавления элемента, удаления и поиска минимального элемента.

Так как set делает все операции за **O(log n),** время работы алгоритма будет **O(m log n),** что работает быстрее **для разреженных графов** (где m << n2).

**Почему алгоритм действительно перебирает вершины в порядке возрастания расстояний**

Пусть сейчас мы рассматриваем вершину v с расстоянием d[v] от стартовой. Так как мы всегда рассматриваем вершину с минимальным значением d из ещё не рассмотренных, то все остальные значения d у нерассмотренных вершин не меньше d[v]. И они никогда не станут меньше d[v], потому что **в графе нет отрицательных рёбер.**

Все вершины с расстоянием меньше d[v] уже были рассмотрены и больше из них ничего обновляться не будет. А из вершин с расстоянием не меньше d[v] нельзя сделать операцию обновления (при проходе по неотрицательному ребру) и получить расстояние меньше d[v].

Таким образом, в любой момент времени, когда алгоритм выбирает какую-то вершину v для рассмотрения, **настоящее расстояние** до неё от стартовой вершины (а не только текущее значение в массиве d) является минимальным среди нерассмотренных вершин, если в графе нет отрицательных рёбер.