**Алгоритм Дейкстра**

Поиск кратчайшего пути по схеме, где множество точек соединены между собой отрезками в виде такой схемы можно представить многие объекты реального мира, поэтому практических примеров использования алгоритма много:

* автоматическое построение маршрута на онлайн-карте;
* поиск системой бронирования наиболее быстрых или дешевых билетов, в том числе с возможными пересадками;
* моделирование движения робота, который перемещается по местности;
* разработка поведения неигровых персонажей, создание игрового ИИ в геймдеве;
* автоматическая обработка транспортных потоков;
* маршрутизация движения данных в компьютерной сети;
* расчет движения тока по электрическим цепям.

**Описание**

Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

**Инициализация.**

1. Устанавливаем город 1 в качестве начального узла
2. Устанавливаем расстояние между городом 1 и другими городами бесконечность, за исключением расстояния между городом 1 и самим собой которое принимает значение равное 0.

Метка самой вершины, a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как не посещённые.

**Шаг алгоритма.**

1. Выбираем узел в с наименьшим значением в качестве текущего узла и посещаем всех при
2. Как только мы посетим всех соседей текущего узла и обновим их текущее расстояние
3. Вернемся к первому шагу и повторим до тех пор пока не посетим все узлы графа

Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовём соседями этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом.

Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма.

**Дерево** - Это связный граф без петель и цифр

**Корневой узел** - это самый верхний узел дерева

**Лист** - это узел не имеющий дочерних элементов

**Внутренний узел** - любой узел дерева имеющий потомков и таким образом не являющийся листом

**Основные операции:**

* вставка нового элемента в определённую позицию;
* вставка поддерева; υ добавление ветви дерева (называется прививкой);
* нахождение корневого элемента для любого узла;
* нахождение наименьшего общего предка двух вершин;
* перебор всех элементов дерева;
* перебор элементов ветви дерева;
* поиск изоморфного поддерева;
* поиск элемента;
* удаление ветви дерева (называется обрезкой);
* удаление поддерева;
* удаление элемента.

**Область применения**

**Виды деревьев:**

**Двоичное дерево** - это иерархическая структура данных каждый узел имеет не более 2 потомков, как правило данный узел называется родительским узлом, а дети называются левыми и правыми наследник. Обычно используют дерево поиска и

**Сбалансированное дерево** - это дерево, в котором заполнены все уровни возможно кроме последнего чем ближе дерево поиска ближе к сбалансированному тем эффективнее операции с ним. Добавление или удаление эл дерева поиска легко нарушает сбалансированность. Поддержание сбалансированности слишком дорого, на практике чаще используется почти сбалансированные деревья

**Методы обхода:**

1. Поиск в глубину
   1. (L) Рекурсивно обходим левое поддерево. Этот шаг завершается при попадании опять в узел N.
   2. (R) Рекурсивно обходим правое поддерево. Этот шаг завершается при попадании опять в узел N.
   3. (N) Обрабатываем сам узел N.
2. Поиск в ширину

**Двоичная куча (В презентации)**

Это такое двоичное дерево для которого выполнены условия:

1. Значение в любой вершине не больше чем в начальной
2. На случайном слое вершин 2^i кроме последнего, слои номеруются с нуля
3. Последний слой заполнен слева направо

Высота кучи это: О(log n), где n кол-во узлов дерева

Двоичные кучи используют для того чтобы извлекать минимум из набора чисел за О(log n).

**Операции над двоичной кучей (В презентации):**

* **Добавление элемента**
* **Упорядочение двоичной кучи**
* **Построение двоичной кучи -** наиболее очевидный способ посторить кучу из неупорядоченного массива, это по очереди добавить всего его элементы. Однако можно построить кучу быстрее за О(n). Для этого сначалла строим дерево из всех элементов массива не заботясь о соблюдении основного сво-ва кучи а потом вызывать метод упорядочивания для всех вершин у которых есть хотябы один потомков. В упорядоченной двоичной кучей максимальный элемент всегда хранится в корне
* **Извлечение (удаление) максимального элемента**