# [MIT's Introduction to Algorithms, Lectures 17, 18 and 19: Shortest Path Algorithms](http://www.catonmat.net/blog/mit-introduction-to-algorithms-part-twelve/" \o "MIT's Introduction to Algorithms, Lectures 17, 18 and 19: Shortest Path Algorithms)

Top of Form

This article is part of the article series "[MIT Introduction to Algorithms](http://www.catonmat.net/series/mit-introduction-to-algorithms)."

[<- previous article](http://www.catonmat.net/blog/mit-introduction-to-algorithms-part-eleven/) [next article ->](http://www.catonmat.net/blog/mit-introduction-to-algorithms-part-thirteen/)

<http://www.catonmat.net/blog/mit-introduction-to-algorithms-part-twelve/>

Bottom of Form

This is the twelfth post in an article series about MIT's lecture course "**Introduction to Algorithms**." In this post I will review a trilogy of lectures on graph and shortest path algorithms. They are lectures seventeen, eighteen and nineteen. They'll cover **Dijkstra's Algorithm**, **Breadth-First Search Algorithm** and **Bellman-Ford Algorithm** for finding single-source shortest paths as well as **Floyd-Warshall Algorithm** and **Johnson's Algorithm** for finding all-pairs shortest paths.

These algorithms require a thorough understanding of graphs. See the [previous lecture](http://www.catonmat.net/blog/mit-introduction-to-algorithms-part-eleven/) for a good review of graphs.

Lecture seventeen focuses on the single-source shortest-paths problem: Given a graph G = (V, E), we want to find a shortest path from a given source vertex s ∈ V to each vertex v ∈ V. In this lecture the weights of edges are restricted to be positive which leads it to Dijkstra's algorithm and in a special case when all edges have unit weight to Breadth-first search algorithm.

Lecture eighteen also focuses on the same single-source shortest-paths problem, but allows edges to be negative. In this case a negative-weight cycles may exist and Dijkstra's algorithm would no longer work and would produce incorrect results. Bellman-Ford algorithm therefore is introduced that runs slower than Dijkstra's but detects negative cycles. As a corollary it is shown that Bellman-Ford solves Linear Programming problems with constraints in form xj - xi <= wij.

Lecture nineteen focuses on the all-pairs shortest-paths problem: Find a shortest path from u to v for every pair of vertices u and v. Although this problem can be solved by running a single-source algorithm once from each vertex, it can be solved faster with Floyd-Warshall algorithm or Johnson's algorithm.

## Lecture 17: Shortest Paths I: Single-Source Shortest Paths and Dijkstra's Algorithm

Lecture seventeen starts with a small review of paths and shortest paths.

It reminds that given a graph G = (V, E, w), where V is a set of vertices, E is a set of edges and w is weight function that maps edges to real-valued weights, a **path** p from a vertex u to a vertex v in this graph is a sequence of vertices (v0, v1, ..., vk) such that u = v0, v = vk and (vi-1, vi) ∈ E. The weight w(p) of this path is a sum of weights over all edges = w(v0, v1) + w(v1, v2) + ... + w(vk-1, vk). It also reminds that a **shortest path** from u to v is the path with minimum weight of all paths from u to v, and that a shortest path in a graph might not exist if it contains a negative weight cycle.

The lecture then notes that shortest paths exhibit the optimal substructure property - a subpath of a shortest path is also a shortest path. The proof of this property is given by cut and paste argument. If you remember from previous two lectures on [dynamic programming](http://www.catonmat.net/blog/mit-introduction-to-algorithms-part-ten/) and [greedy algorithms](http://www.catonmat.net/blog/mit-introduction-to-algorithms-part-eleven/), an optimal substructure property suggests that these two techniques could be applied to solve the problem efficiently. Indeed, applying the greedy idea, Dijkstra's algorithm emerges.

Here is a somewhat precise definition of single-source shortest paths problem with non-negative edge weights: Given a graph G = (V, E), and a starting vertex s ∈ V, find shortest-path weights for all vertices v ∈ V.

Here is the greedy idea of Dijkstra's algorithm:

* 1. Maintain a set S of vertices whose shortest-path from s are known (s ∈ S initially).
* 2. At each step add vertex v from the set V-S to the set S. Choose v that has minimal distance from s (be greedy).
* 3. Update the distance estimates of vertices adjacent to v.

I have also posted a video interview with Edsger Dijkstra - [Edsger Dijkstra: Discipline in Thought](http://www.catonmat.net/blog/edsger-dijkstra-discipline-in-thought/), please take a look if you want to see how Dijkstra looked like. :)

The lecture continues with an example of running Dijkstra's algorithm on a non-trivial graph. It also introduces to a concept of a shortest path tree - a tree that is formed by edges that were last relaxed in each iteration (hard to explain in English, see lecture at 43:40).

The other half of lecture is devoted to three correctness arguments of Dijkstra's algorithm. The first one proves that relaxation never makes a mistake. The second proves that relaxation always makes the right greedy choice. And the third proves that when algorithm terminates the results are correct.

At the final minutes of lecture, running time of Dijkstra's algorithm is analyzed. Turns out that the running time depends on what data structure is used for maintaining the priority queue of the set V-S (step 2). If we use an array, the running time is O(V2), if we use binary heap, it's O(E·lg(V)) and if we use Fibonacci heap, it's O(E + V·lg(V)).

Finally a special case of weighted graphs is considered when all weights are unit weights. In this case a single-source shortest-paths problem can be solved by a the Breadth-first search (BFS) algorithm that is actually a simpler version of Dijkstra's algorithm with priority queue replaced by a FIFO! The running time of BFS is O(V+E).