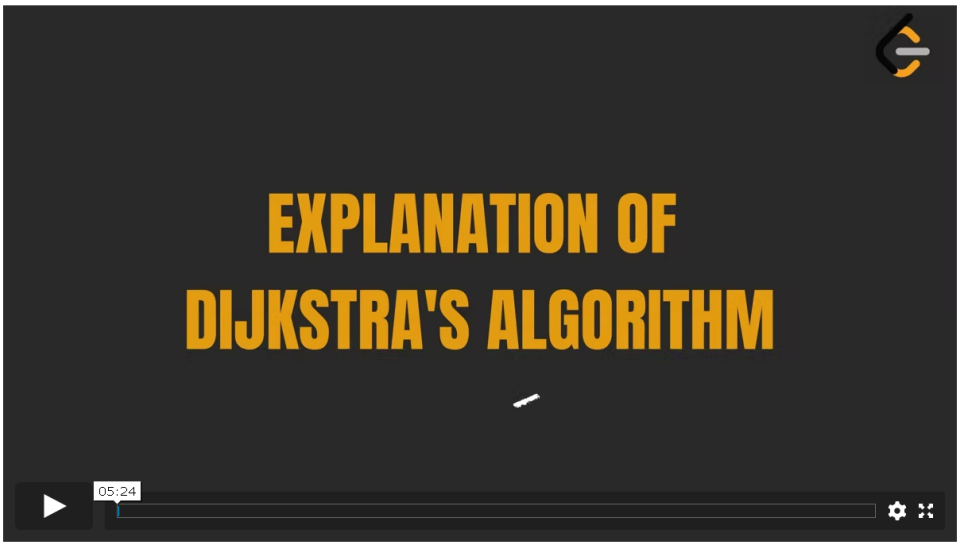
Dijkstra's Algorithm

Report Issue

"Dijkstra's algorithm" solves the "single-source shortest path" problem in a weighted directed graph with non-negative weights.

Video Explanation

In the next video, we'll explain how to solve the "single-source shortest path" problem using Dijkstra's Algorithm.



The Main Idea

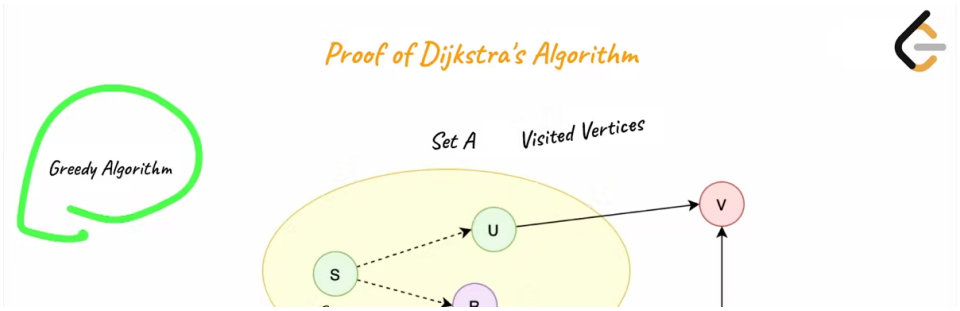
We take the starting point `u` as the center and gradually expand outward while updating the "shortest path" to reach other vertices.

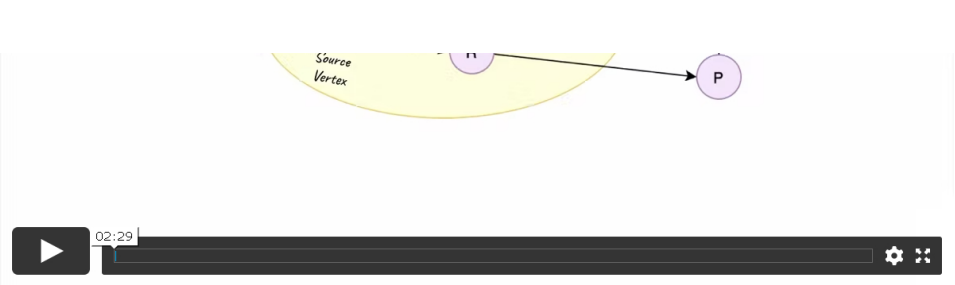
"Dijkstra's Algorithm" uses a "greedy approach". Each step selects the "minimum weight" from the currently reached vertices to find the "shortest path" to other vertices.

Proof of the Algorithm

Now let's prove that Dijkstra's Algorithm actually leads to the correct answer. We'll do that in the next video.

The "greedy approach" only guarantees that, at each step, it takes the optimal choice in the current state. It does not guarantee that the final result is optimal. So, how does "Dijkstra's Algorithm" ensure that its final result is optimal?





Limitation of the Algorithm

"Dijkstra's Algorithm" can only be used on graphs that satisfy the following condition:

- Weights of all edges are non-negative.

We'll discuss the aforementioned limitation of the Dijkstra's Algorithm in the following video.

Complexity Analysis

V represents the number of vertices, and E represents the number of edges.

- Time Complexity: $O(E + V \log V)$ when a Fibonacci heap is used, or $O(V + E \log V)$ for a Binary heap.
  - If you use a Fibonacci heap to implement the "min-heap", extracting minimum element will take $O(\log V)$ time while key decreasing operation will take amortized $O(1)$ time, therefore, the total time complexity would be $O(E + V \log V)$.
  - If you use a Binary heap, then the time complexity would be $O(V + E \log V)$.
- Space Complexity: $O(V)$. We need to store V vertices in our data structure.