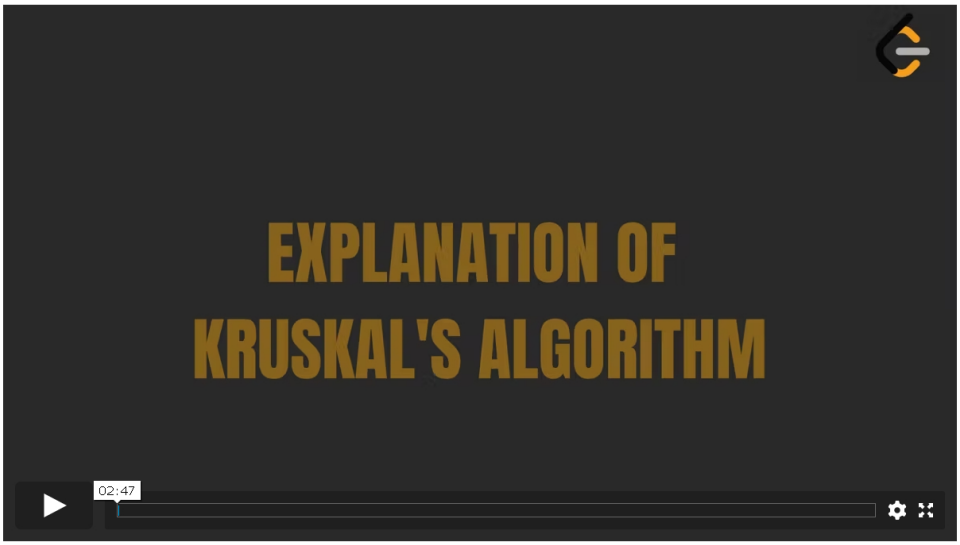
Kruskal's Algorithm

Report Issue

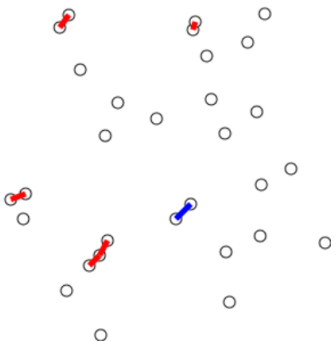
"Kruskal's algorithm" is an algorithm to construct a "minimum spanning tree" of a "weighted undirected graph".

Video Explanation

The following video will introduce Kruskal's Algorithm and show how it can be applied to construct a "minimum spanning tree" of a "weighted undirected graph".



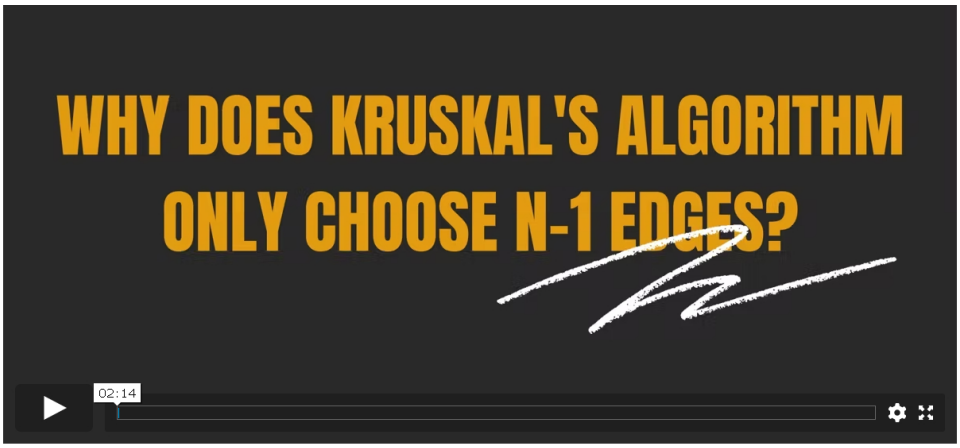
Visual Example



The above animation shows how Kruskal's algorithm ***grows*** the minimum spanning tree by adding edges. In this example, the distance between two vertices is the edge weight. We try adding each edge, one at a time, from the lowest weight edge up to the highest weight edge. If either of the edges' vertices is not already part of the MST, then the edge is added to the MST.

Why does Kruskal's Algorithm only choose N-1 edges?

In the following video, we'll prove that we need to choose exactly $N-1$ edges of the graph with N edges in total to construct a "minimum spanning tree" of that graph.



Please note that in the video above, all the graphs are linked lists. However, it's not necessary for a "minimum spanning tree" to be a linked list only. In other words, we can form an MST without forming a linked list. In any case, only `N-1` edges are needed.

Why can we apply the "greedy strategy"?

In the subsequent video, you'll see why the greedy approach does work in accomplishing our task.

Complexity Analysis

- Time Complexity: $O(E \cdot \log E)$. Here, E represents the number of edges.
    - At first, we need to sort all the edges of the graph in ascending order. Sorting will take $O(E \log E)$ time.
    - Next, we can start building our minimum spanning tree by selecting which edges should be included. For each edge, we will look at whether both of the vertices of the edge belong to the same connected component; which is an $O(\alpha(V))$ operation, where α refers to the Inverse Ackermann function. In the worst case, the tree will not be complete until we reach the very last edge (the edge with the largest weight), so this process will take $O(E\alpha(V))$ time.
    - Therefore, in total, the time complexity is $O(E \log E + E\alpha(V)) = O(E \log E)$.
- Space Complexity: $O(V)$. V represents the number of vertices. Keeping track of the root of every vertex in the union-find data structure requires $O(V)$ space. However, depending on the sorting algorithm used, different amounts of auxiliary space will be required to sort the list of edges in place. For instance, Timsort (used by default in python) requires $O(E)$ space in the worst-case scenario, while Java uses a variant of quicksort whose space complexity is $O(\log E)$.