25.3 Kruskal's Algorithm

Finding MST.

Another algorithm that finds the MST of a graph is Kruskal's Algorithm. Instead of constructing the MST by traversing through the **nodes** like Prim's Algorithm, Kruskal's Algorithm finds the MST by traversing through the **edges**.

Conceptual Visualization and Implementation

The algorithm is as follows:

1. Sort all the edges from lightest to heaviest.
2. Taking one edge at a time (in sorted order), add it to the MST under construction if doing so **does not introduce a cycle.**
3. Repeat until there are V - 1 edges.

Why does it work?

Kruskal's algorithm works because of the following reasons:

- Recall the two "sets" introduced in last section. Any edge we add to the MST will be connecting one node from "set one" (nodes that are in the MST under construction), and another node from "set two" (all other nodes).
- The added edge is not part of cycle because we are only adding an edge if it does not introduce a cycle.
- By looking at edge candidates in order from lightest to heaviest, the added edge must be the lightest edge across this cut. (if there was a lighter edge that would be across this cut, it would have been added before this, and adding this one would cause a cycle to appear).

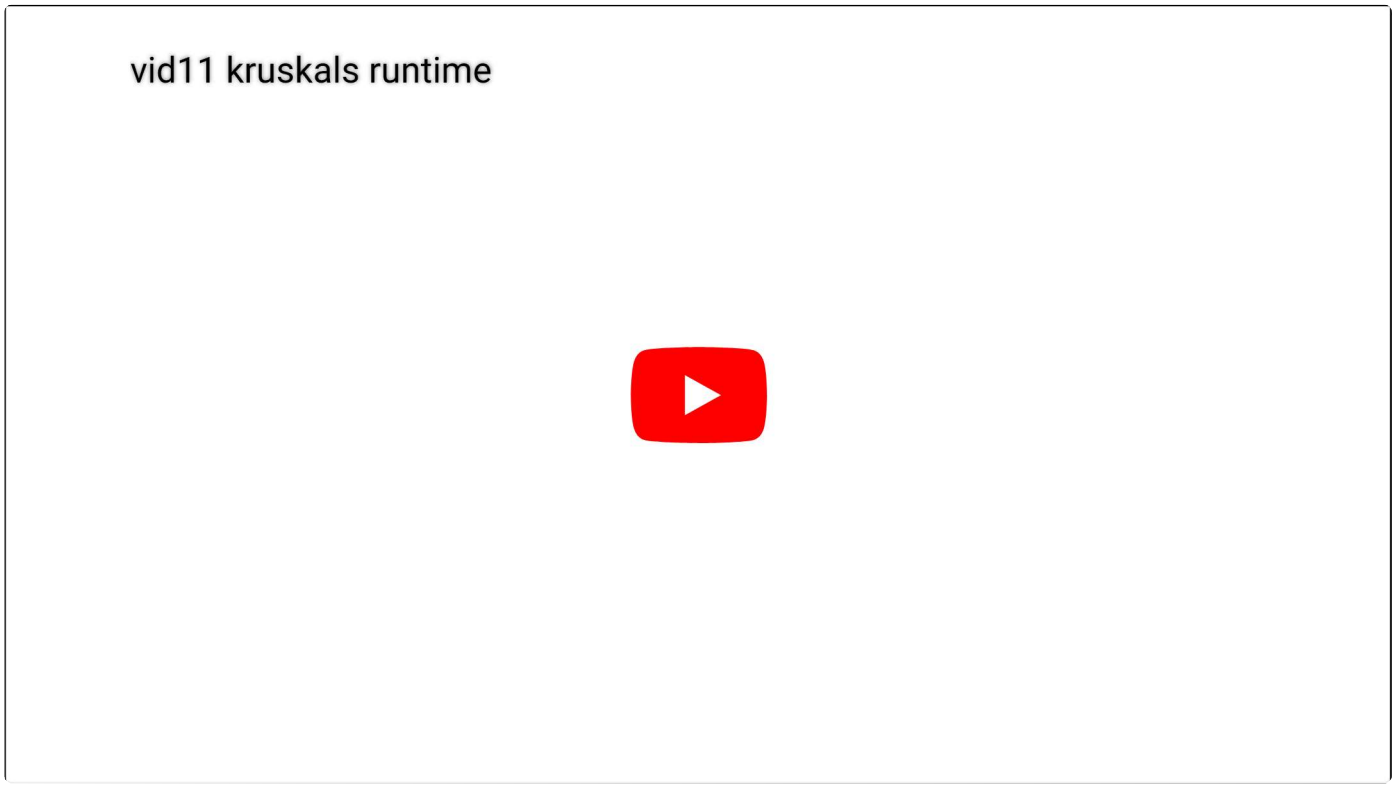
Hence, this algorithm works also by **the Cut Property.**

Kruskal's vs Prim's

It's important to note that the MST returned by Kruskal's might not be the same one returned by Prim's, but both algorithms will always return a MST.

Since both are minimal (optimal), they will both give valid optimal answers (they are tied as equally minimal / same total weight, and this is as low as it can be).

Runtime Analysis



Runtime with Unsorted Edges

Since the underlying data structures of implementing Kruskal's Algorithm are a Priority Queue (to sort the edges), and a Disjoined Set (to connect the edges), the runtime of Kruskal's Algorithm is in tandem with the runtime analysis of Heaps and WQU.

Specifically, we are using Weighted Quick Union with Path Compression to check whether or not an added edge will introduce a cycle.

The operations that this algorithm utilizes and their corresponding runtime are:

- Insert: $O(|E|log|E|)$

- Delete minimum: $O(|E|log|E|)$
- Union: $O(|V|log*|V|)$
- isConnected: $O(|E|log*|V|)$

The bottleneck of the algorithm is sorting all of the edges to start (insert and delete minimum. Hence, the runtime of Kruskal's Algorithm is:

$$O(|E|log|E|)$$

Runtime with Sorted Edges

Since there are no need to sort the edges, the "Insert" and "Delete minimum" operations of a heap are not called, where only WQU operations are called.

Therefore, if we are given pre-sorted edges and don't have to pay for that, then the runtime is:

$$O(|E|log*|V|)$$

Where $log*$is the Ackermann function.

Previous
25.2 Prim's Algorithm

Next
25.4 Chapter Summary

Last updated 1 year ago