02-613 Week 2

Algorithms and Advanced Data Structures

Aidan Jan

September 3, 2025

Union-Find Abstract Data Type

In Kruskal's algorithm, we need a data structure to quickly find cycles. We need a way to:

- quickly determine which subgraph a given node is in
- quickly merge two subgraphs
- a constructor for the data structure

One data structure we can use is the Union-Find data structure. What we basically do is store a collection of trees.

- Create a struct for a node, containing the connections for the graph in an array, but also, each has an additional variable, a parent pointer.
- To determine which group the node is in, follow the parent pointers up until you get to a node with no parent pointer. (This gives every subgraph a unique identifier.)
- To merge two groups, simply take the root of the smaller tree, and add the root of the bigger tree as a parent to the smaller tree.
  - To do this quickly, we need a size array. Without size array would be $\mathrm{O}(n)$. With size array would be $\mathrm{O}(1)$.

We can actually optimize this further if we convert the structs to arrays.

- This increases the speed of the find operation from $\mathrm{O}(\log n)$ to $\mathrm{O}(1)$.
- However, this decreases the speed of the union operation from $\mathrm{O}(1)$ to $\mathrm{O}(k \log k)$, where k is the number of unions that has occurred before.
- This is an upgrade since we will likely use the find operation much more than the union operation.

Aside: theorem

After k unions on n items, $\mathrm{O}(k \log k)$ total time.

Proof. k unions touches at most $2k$ items, and any item v is relabeled $\log_2(2k)$ times. This is because we start with each vertex in its own group, and every union at maximum doubles its size. As a result, each union has $2k \log_2(2k)$ of work, which is $\mathrm{O}(k \log k)$. In the worst case, this converges to $\mathrm{O}(\log n)$. □

Kruskal's Runtime

With this data structure, we can now calculate the running time of Kruskal's algorithm:

- Sort the edges: $\mathrm{O}(m \log m)$
- Test every edge: $\mathrm{O}(m)$ edges $\times$ $(2 \cdot \texttt{find()}) + \texttt{union()}$
- Therefore, the total time is $\mathrm{O}(m \log m + 2m + m \log n)$. Since there are more edges than vertices, the $\mathrm{O}(m \log m)$ dominates. Sorting the edges takes the longest amount of time.