15-122: Principles of Imperative Computation

Recitation 13: Union-finding your Roots — Rob Simmons

Kruskal's algorithm

Kruskal's algorithm is an algorithm to find a minimum weight spanning tree (often called a minimum spanning tree) on a graph. Visualization: `http://www.cs.usfca.edu/~galles/visualization/Kruskal.html`.

A spanning tree of a graph is a subgraph that is a tree and that connects all vertices of the graph. (Remember that a tree is a connected graph with no cycles.)

A minimum spanning tree (MST) is simply a spanning tree whose edges have the minimum total weight of any spanning tree on the graph. For any given graph, there may be multiple different MSTs.

Kruskal's algorithm is an algorithm that finds the minimum spanning tree for a graph. The algorithm works as follows:

1. Sort all edges by weight, from smallest weight to largest weight.
2. Go through the edges in order. If adding an edge would not create a cycle in the graph, add it. When we have a minimum spanning tree (when the number of edges we've added is one less than the number of vertices), we're done.

Union find

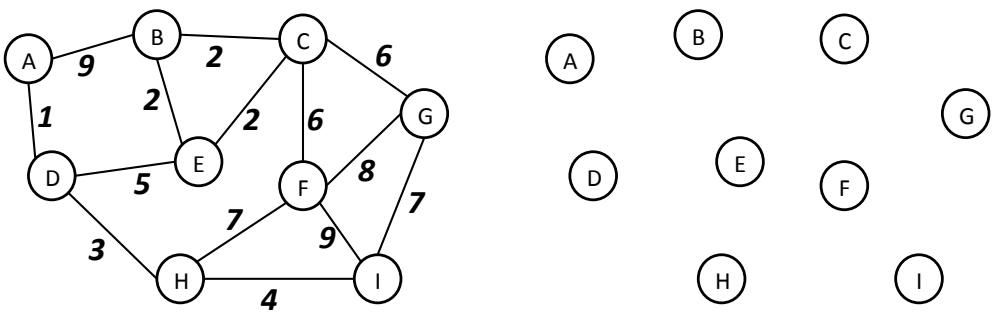
We can answer the question of whether an edge creates a cycle by tracking equivalence classes of vertices. Two vertices are equivalent if there's a path from one to the other using the spanning tree edges we've already added during Kruskal's algorithm.

We use the union find data structure to efficiently maintain equivalence classes. We think of union find as maintaining a directed graph called the equivalence graph, where each vertex has at most one outgoing edge, and where following a series of directed edges eventually reaches the canonical representative of the equivalence class, which has no outgoing edge.

Two elements are equivalent if and only if they have the same canonical representative. We can make two elements equivalent (when we add an edge to the MST) by finding their canonical representatives and connecting them with an edge.

Checkpoint 0

Run Kruskal's algorithm on the weighted graph to the left to form a spanning tree. Track the structure of the equivalence graph to the right. There's more than one answer.



Which edges that aren't even part of the original weighted graph end up as directed edges in the equivalence graph?

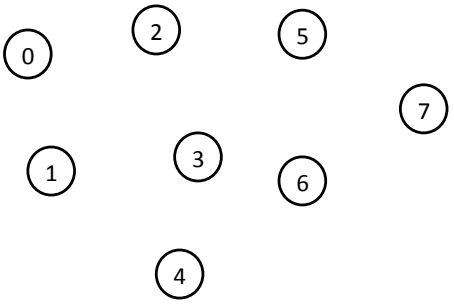
Union-find implementation

Union find's equivalence graphs can be represented with arrays: the indices corresponding to non-canonical representatives store the element that their directed edge in the equivalence graph points to. Canonical representatives can store their own index, but in order to efficiently implement union-find, it's useful for canonical representatives to store a negative number, the absolute value of which is the maximum height of any path to that canonical representative.

Checkpoint 1

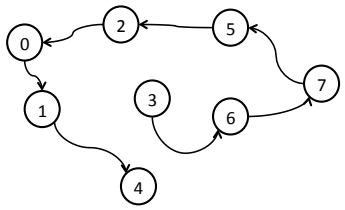
For the following series of union operations, make the worst possible decision at every step when you connect canonical representatives to try to make the paths as long as possible. Draw the state of the array after every step. How long is the longest path?

	0	1	2	3	4	5	6	7
union 1 & 2								
union 5 & 7								
union 6 & 7								
union 1 & 5								
union 3 & 6								
union 4 & 7								



Checkpoint 2

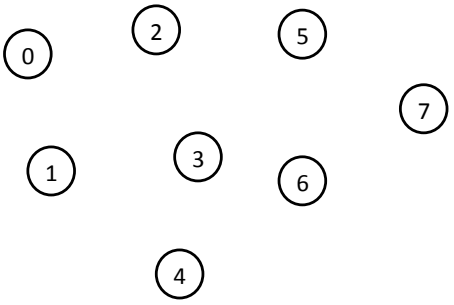
Describe a series of unions that could lead to this equivalence graph:



Checkpoint 3

If we use the tree-height storing version of union find on the unions from Checkpoint 2, what happens?

	-1	-1	-1	-1	-1	-1	-1	-1
union _ & _								
union _ & _								
union _ & _								
union _ & _								
union _ & _								
union _ & _								
union _ & _								



Checkpoint 4

If we use the tree-height storing version of union-find, what's a worst-case equivalence graph? What's a series of union operations that would create it?