Disjoint-Set Operations

Supplemental reading in CLRS: Chapter 21 (§21.4 is optional)

When implementing Kruskal's algorithm in Lecture 4, we built up a minimum spanning tree T by adding in one edge at a time. Along the way, we needed to keep track of the connected components of T; this was achieved using a disjoint-set data structure. In this lecture we explore disjoint-set data structures in more detail.

Recall from Lecture 4 that a **disjoint-set data structure** is a data structure representing a dynamic collection of sets $\mathbf{S}=\{S_1,\ldots,S_n\}$. Given an element u, we denote by S_u the set containing u. We will equip each set S_i with a representative element $\text{rep}[S_i]$.[1] This way, checking whether two elements u and v are in the same set amounts to checking whether $\text{rep}[S_u] = \text{rep}[S_v]$. The disjoint-set data structure supports the following operations:

- MAKE-SET(u): Creates a new set containing the single element u.
  - u must not belong to any already existing set – of course, u will be the representative element initially
- FIND-SET(u): Returns the representative $\text{rep}[S_u]$.
- UNION(u,v): Replaces S_u and S_v with $S_u \cup S_v$ in $\mathbf{S}$. Updates representative elements as appropriate.

[1] This is not the only way to do things, but it's just as good as any other way. In any case, we'll need to have some symbol representing each set S_i (to serve as the return value of FIND-SET); the choice of what kind of symbol to use is essentially a choice of notation. We have chosen to use a "representative element" of each set rather than create a new symbol.