1 Disjoint-set data structure

1.1 Operations

The disjoint-set data structure enable us to efficiently perform operations such as placing elements into sets, querying whether two elements are in the same set, and merging two sets together. (Thus they’re very useful for simulating graph connectivity.) Must implement the following operations:

- **MAKESET**(x): create a new set containing the single element x.
- **UNION**(x, y): replace sets containing x and y by their union.
- **FIND**(x): return name of set containing x.

We add for convenience the function **LINK**(x, y) where x, y are roots: LINK changes the parent pointer of one of the roots to be the other root. In particular, **UNION**(x, y) = **LINK**(**FIND**(x), **FIND**(y)), so the main problem is to make the **FIND** operations efficient.

1.2 Optimization Heuristics

We have two main methods of optimization for disjoint-set data structures:

- **Union by rank.** When performing a **UNION** operation, we prefer to merge the shallower tree into the deeper tree.
- **Path compression.** After performing a **FIND** operation, we can simply attach all the nodes touched directly onto the root of the tree.

**Exercise 1.** When using neither union by rank nor path compression, what is the asymptotic runtime of **FIND**(x)? **UNION**(x, y)?

**Exercise 2.** Draw how the disjoint set data structure changes after each of the following operations using a particular heuristic:

(a) **UNION**(x, y) with union by rank.
Exercise 3. When using the union by rank optimization only, what is the asymptotic runtime of the operation FIND(x)?

Exercise 4. Suppose that you are given an undirected friendship map for everyone at Harvard. (Vertices are students, and there is an edge between two students if they are friends.) Define a clique as a strongly connected component of this graph (note for those familiar with graph theory, this is not the typical definition of a clique on a graph).

(a) How could you (not using the union-find data structure) determine the number of cliques?

(b) How could you solve this same problem using the union-find data structure?
(c) How do the runtimes between the two approaches differ?

(d) What if now you are at Visitas and people are constantly making friends. You would like to know how many cliques there are at any given point in time with the knowledge that more edges are constantly being drawn on the graph. How do both approaches generalize?

(e) What about if people are also allowed to have falling-outs and unfriend each-other? Is there any way to modify union-find to work in this situation as well?