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)?

**Solution**

The worst possible case is when the elements get added into what resembles a linked list, for \(O(n)\) cost of both finding and merging.

**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.
(b) FIND($x$) with path compression.

![Diagram](attachment:image.png)

Solution

Exercise 3. When using the union by rank optimization only, what is the asymptotic runtime of the operation FIND($x$)?

Solution

When we use the union by rank heuristic, we have the property that the rank of a tree only increases when we union together two trees of the same rank. (Recall that rank = height). If you union a tree with rank 3 with one of the rank 4, you get a tree of rank 4. Only when you union two trees of rank 4 can you get a tree of rank 5.

With this property, we can show that a tree of rank $h$ has at least $2^h$ nodes. This can be proved using induction.

Base case. When the rank of a tree is 0, there is always 1 node, which is at least $2^0$.

Inductive step. Suppose it is true that every tree of rank $h$ has at least $2^h$ nodes. If we had a tree of rank $h + 1$, we know that it must have come about through the union of two trees of rank $h + 1$ at some point in the history of this tree. By the inductive hypothesis, those two trees of rank $h$ must have had $\geq 2^h$ nodes each, and thus the number of nodes in this tree of rank $h + 1$ must be at least $2^h + 2^h = 2^{h+1}$.
Therefore, we have that the run-time of \texttt{FIND}(x) when using only the union by rank heuristic is $O(\log n)$ because if there are $n$ nodes, then the maximum rank of any tree is $\log n$.

**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?

**Solution**

(a) We can use BFS to find the number of connected components in the spirit of pset 2 question 2.

(b) We can \texttt{makeset} for every student, and then take unions for every edge that we see.

(c) Let $n$ be the number of students, and $m$ the number of friendships. BFS would take time $O(n + m)$ and union-find would take time $O((n + m) \log^* n)$. 
(d) We would have to re-run BFS after every edge that we add, adding a factor of $m$ to our runtime analysis. On the other hand, if we use union-find, we can keep on doing exactly what we were doing before, and thus the runtime stays the same.

(e) Both BFS as well as using union-find would have to re-compute after every edge deletion. To the extent that edge additions are more common, we may still prefer union-find, but there is no straightforward manipulation to be able to use our previously computed results incrementally when edges can be deleted.