1 Disjoint-set data structure.

1.1 Operations

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.

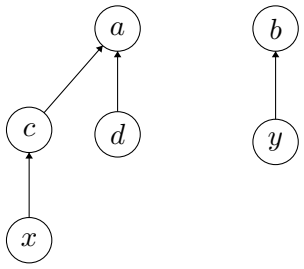
1.2 Optimization Heuristics

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

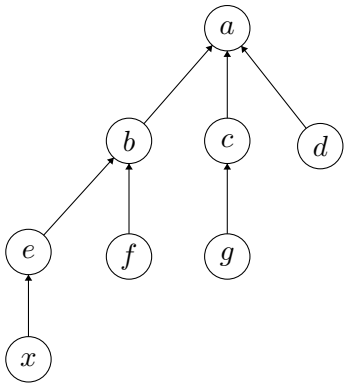
- **Union by rank.** When performing a UNION operation, we prefer to merge the shallower tree into the deeper tree.
- **Path compression.** After performing a FIND operation, we can simply attach all the nodes touched directly onto the root of the tree.

Exercise 1. 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.



Exercise 2. When using the union by rank optimization *only*, what is the asymptotic runtime of the operation FIND(x)?

2 Greedy Algorithms

A **greedy algorithm** is an algorithm which attempts to find the globally optimal solution to a problem by making *locally optimal* decisions.
Examples from Lecture:

- **Kruskal's:** We greedily choose the lightest edge in the graph that doesn't form a cycle.
- **Prim's:** We greedily choose the lightest edge adjacent to the spanning tree we've formed so far.
- **Horn Formula:** We start by assigning all variables to FALSE and only set variables to true when an implication forces you to.
- **Huffman Encoding:** We greedily construct the encoding by taking the two least used characters and merging them into one new character.
- **Set Cover** ($O(k \log n)$ approximation): We greedily choose the set that covers the most number of the remaining uncovered elements at the given iteration.

In order to prove that a greedy algorithm is correct, we need to show that indeed choosing the locally optimal solution keeps us *on track* to finding the globally optimal solution. The cut property with MST's guarantees that the greedy MST algorithms are correct.

Problems with correct greedy algorithms are rare. We will soon be introduced to **Dynamic Programming**, a technique that lets you solve a much wider range of optimization problems. It is arguably the most useful technique you will learn in CS 124.

Exercise 3. Suppose that we have a set $S = \{a_1, \ldots, a_n\}$ of proposed activities. Each activity a_i has a start time s_i and a finish time f_i. We can only run one activity at a time. Your job is to find a maximal set of compatible activities. Which of the following greedy algorithms is correct?

(a) Sort all the activities by their duration and greedily picking the shortest activity that does not conflict with any of the already chosen activities.

(b) Pick the activity that conflicts with the fewer number of remaining activities. Remove the activities that the chosen activity conflicts with. Break ties arbitrarily.

(c) Sort all the activities by their end time and greedily pick the activity with the earliest end time that does not conflict with any of the already chosen activities.

Exercise 4. Let's go back to *greedy.c* from the first CS 50 problem set. The question was to determine the fewest number of US coins necessary to make change for a given amount of money.

(a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

(b) Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

(c) Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.

3 Divide and Conquer

Divide and Conquer algorithms work by recursively breaking the problem into smaller pieces and solving the subproblems. You need to think about how to recursively break down the problem as well as how to combine the various pieces together.

Examples from Lecture:

- **Mergesort and StoogeSort:** We recursively sorted a fraction of the list and then did some work to combine those fractions of the list together.
- **Integer Multiplication:** We split two integers in half each, forming 4 pieces and then performed only 3 multiplications on half-sized numbers.
- **Strassen's Algorithm:** We perform matrix multiplication on two $n \times n$ matrices by performing 7 multiplications on $n/2 \times n/2$ matrices.

Exercise 5. Given an array of n elements, you want to determine whether there exists a majority element (that is an element which occurs at least $\lceil \frac{n+1}{2} \rceil$ times) and if so, output this element. Show how to do this in time $O(n \log n)$ using Divide and Conquer.

Exercise 6. (2015 Problem Set 4) You are given an n-digit positive integer x written in base-2. Give an efficient algorithm to return its representation in base-10 and analyze its running time. Assume you have black-box access to an integer multiplication algorithm which can multiply two n-digit numbers in time $M(n)$ for some $M(n)$ which is $\Omega(n)$ and $O(n^2)$.