Lecture 20: The Dynamic Equivalence Problem (a.k.a. Disjoint Sets, Union/Find etc.)

- The Plot:
  - A new problem: Dynamic Equivalence
  - The setting:
    - Motivation and Definitions
  - The players:
    - Union and Find, two ADT operations
    - Up-tree data structure
  - Suspense-filled cliffhanger (to be continued…next time)
- Covered in Chapter 8 of the textbook

Motivation

- Consider the relation "=" between integers
  1. For any integer A, A = A
  2. For integers A and B, A = B means that B = A
  3. For integers A, B, and C, A = B and B = C means that A = C
- Consider cities connected by two-way roads
  1. A is trivially connected to itself
  2. A is connected to B means B is connected to A
  3. If A is connected to B and B is connected to C, then A is connected to C
- Consider electrical connections between components on a computer chip
  - 1, 2, and 3 are again satisfied

Equivalence Relations

- An equivalence relation R obeys three properties:
  1. reflexive: for any *x*, *x*R*x* is true
  2. symmetric: for any *x* and *y*, *x*R*y* implies *y*R*x*
  3. transitive: for any *x*, *y*, and *z*, *x*R*y* and *y*R*z* implies *x*R*z*
- Preceding relations are all examples of *equivalence relations*
- What are not equivalence relations?

Equivalence Relations

- An equivalence relation R obeys three properties:
  1. reflexive: for any *x*, *x*R*x* is true
  2. symmetric: for any *x* and *y*, *x*R*y* implies *y*R*x*
  3. transitive: for any *x*, *y*, and *z*, *x*R*y* and *y*R*z* implies *x*R*z*
- Preceding relations are all examples of *equivalence relations*
- What are not equivalence relations?
  - What about "<" on integers? (1 and 2 are violated)
  - What about "≤" on integers? (2 is violated)
  - What about "is having an affair with" in a soap opera?
    - Victor i.h.a.a.w. Ashley i.h.a.a.w. Brad does not imply Victor i.h.a.a.w. Brad

Equivalence Classes and Disjoint Sets

- The operator R divides all the elements into <u>disjoint sets</u> of “equivalent” items
- Let ~ be an equivalence relation. Then, if A~B, then A and B are in the same <u>equivalence class</u>.
- Examples:
  - On a computer chip, if ~ denotes “electrically connected,” then sets of connected components form equivalence classes
  - On a map, cites that have two-way roads between them form equivalence classes
  - The relation “Modulo N” divides all integers in N equivalence classes
    - E.g. Under Mod 5, <u>0</u> ~ 5 ~ 10 ~ 15 …, <u>1</u> ~ 6 ~ 11 ~ 16 …, <u>2</u> ~ 7 ~ 12 ~ …, <u>3</u> ~ 8 ~ 13 ~ …, and <u>4</u> ~ 9 ~ 14 ~ …
    - 5 equivalence classes (remainders 0 through 4 when divided by 5)

Problem Definition

- Given a set of elements and some equivalence relation ~ between them, we want to figure out the equivalence classes
- Given an element, we want to <u>find</u> the equivalence class it belongs to
  - E.g. Under mod 5, 13 belongs to the equivalence class of 3
  - E.g. For the map example, want to find the equivalence class of Redmond (all the cities it is connected to)
- Given a new element, want to add it to an equivalence class (<u>union</u>)
  - E.g. Under mod 5, since 18 ~ 13, perform a union of 18 with equivalence class of 13
  - E.g. For the map example, Woodinville is connected to Redmond, so add Woodinville to equivalence class of Redmond

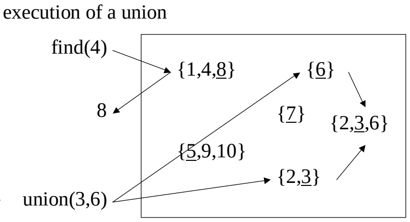
Disjoint Set ADT

- Stores N unique elements
- Two operations:
  - <u>Find</u>: Given an element, return the name of its equivalence class
  - <u>Union</u>: Given the names of two equivalence classes, merge them into one class (which may have a new name or one of the two old names)
- ADT divides elements into E equivalence classes, $1 \leq E \leq N$
  - Names of classes are arbitrary e.g. 1 through N, so long as Find returns the same name for 2 elements in the same equivalence class

Disjoint Set ADT Properties

- Disjoint set equivalence property: every element of a DS ADT belongs to exactly one set (its equivalence class)
- *Dynamic* equivalence property: the set of an element can change after execution of a union

Example:
Initial Classes =
{1,4,8}, {2,3},
{6}, {7},
{5,9,10}
Name of equiv.
class underlined



Formal Definition (for Math lovers' eyes only)

- Given a set $U = \{a_1, a_2, \dots, a_n\}$
- Maintain a *partition* of U, a set of subsets (or equivalence classes) of U denoted by $\{S_1, S_2, \dots, S_k\}$ such that:
  - each pair of subsets S_i and S_j are disjoint: $S_i \cap S_j = \varnothing$
  - together, the subsets cover U: $U = \bigcup_{i=1}^{k} S_i$
  - each subset has a unique name
- Union(a, b) creates a new subset which is the union of a's subset and b's subset
- Find(a) returns a unique name for a's subset

Implementation Ideas and Tradeoffs

- How about an array implementation?
  - N element array A → A[i] holds the class name for element i
  - E.g. if 18 ~ 3, pick 3 as class name and set A[18] = A[3] = 3
  - Running time for Find(i) = ? (i = some element)
  - Running time for Union(i,j) = ? (i and j are class names)

Implementation Ideas and Tradeoffs

- How about an array implementation?
  - N element array A → A[i] holds the class name for element i
  - E.g. if 18 ~ 3, pick 3 as class name and set A[18] = A[3] = 3
  - Running time for Find(i) = O(1) (just return A[i])
  - Running time for Union(i,j) = O(N)
    - If first N/2 elements have class name 1 and next N/2 have class name 2, Union(1,2) will need to change class names of N/2 items
- How about linked lists?
  - One linked list for each class
  - Running time for Union(i,j) and Find(i) = ?

Implementation Ideas and Tradeoffs

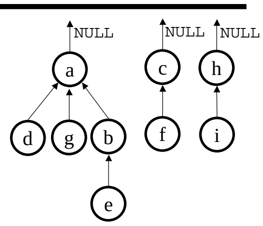
- How about an array implementation?
  - N element array A → A[i] holds the class name for element i
  - E.g. if 18 ~ 3, pick 3 as class name and set A[18] = A[3] = 3
  - Running time for Find(i) = O(1) (just return A[i])
  - Running time for Union(i,j) = O(N)
- How about linked lists?
  - One linked list for each class
  - Running time for Union(i,j) = O(1) (just append one list to the other)
  - Running time for Find(i) = O(N) (must scan all lists in worst case)
- Tradeoff between Union-Find – cannot do both in O(1) time
  - N-1 Unions (the max) and M Finds → $O(M + N^2)$ or $O(N + MN)$
  - Can we do this in O(M + N) time? We will answer this question in this class and next…but first…

Let's find a new Data Structure

- Intuition: Finding the representative member (= class name) of a set is like the *opposite* of finding a key in a given set
- So, instead of trees with pointers from each node to its children, let's use trees with a pointer from each node to its parent
- Such trees are known as Up-Trees

Up-Tree Data Structure

- Each equivalence class (or discrete set) is an up-tree with its root as its representative member
- All members of a given set are nodes in that set's up-tree
- Hash table maps input data to the node associated with that data e.g. input string → integer

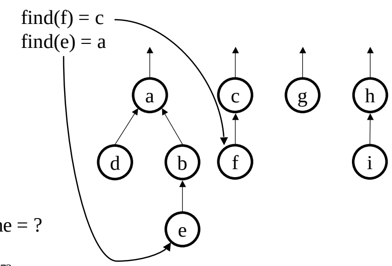


{a,d,g,b,e} {c,f} {h,i}

Up-trees are usually **not** binary!

Example of Find

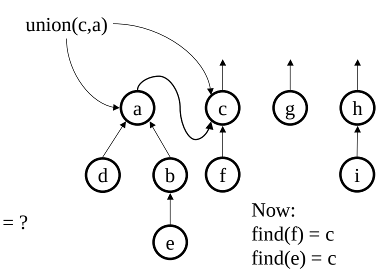
Find: Just traverse to the root!



Runtime = ?

Example of Union

Union: Just hang one root from the other!



Runtime = ?

Now:
find(f) = c
find(e) = c

To be continued next class…

(same place, same time)

Meanwhile…

Finish reading chapter 8