Disjoint Sets

CSE 373 - Data Structures
May 17, 2002
Readings and References

• Reading
  › Chapter 8, *Data Structures and Algorithm Analysis in C*, Weiss

• Other References
Relations on a set

• Consider the relation “=” between integers
  › For any integer a, a = a
  › For integers a and b, a = b means that b = a
  › For integers a, b, and c, a = b and b = c means that a = c
Relations on a set

• Consider cities connected by two-way roads
  › Seattle is connected to itself
  › Seattle is connected to Everett means Everett is connected to Seattle
  › If Seattle is connected to Everett and Everett is connected to Bellingham, then Seattle is connected to Bellingham

• Consider electrical connections between components on a computer chip
Equivalence Relations

• An equivalence relation $R$ obeys three properties:
  › **reflexive:** for any $x$, $xRx$ is true
  › **symmetric:** for any $x$ and $y$, $xRy$ implies $yRx$
  › **transitive:** for any $x$, $y$, and $z$, $xRy$ and $yRz$
    implies $xRz$

• Preceding relations are all examples of
  *equivalence relations*
Equivalence Relations

• What are some relations that are not equivalence relations?
  › What about “<” on integers?
    • not reflexive, not symmetric
  › What about “≤” on integers?
    • not symmetric
  › What about “is having an affair with” in a soap opera?
    • Victor IHAAW Ashley IHAAW Brad does not imply
      Victor IHAAW Brad ↴ not transitive
    • probably not reflexive, although in the soaps, who
      knows ...
Equivalence Classes & Disjoint Sets

- A specific equivalence relation operator $R$ divides all the elements into disjoint sets of related items.
- Let “~” be an equivalence relation.
- If $a \sim b$, then $a$ and $b$ are in the same equivalence class.
Equivalence Class Examples

• If ~ denotes “electrically connected,” then sets of connected components on a computer chip form equivalence classes

• On a map, cites that have two-way roads between them form equivalence classes
  › as long as you say that reflexive means that just sitting in town satisfies Seattle ~ Seattle
    • path length = 0
  › We don’t have loop roads that go out and come back
    • path length = 1
Modulo example

• The relation “Modulo N” divides all integers in N equivalence classes.
  › For example, “a mod 5” on the integers produces 5 equivalence classes (remainders 0 through 4 when the integers are divided by 5)
    • \(0\) ~ 5 ~ 10 ~ …
    • \(1\) ~ 6 ~ 11 ~ …
    • \(2\) ~ 7 ~ 12 ~ …
    • \(3\) ~ 8 ~ 13 ~ …
    • \(4\) ~ 9 ~ 14 ~ …
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 find 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 Everett (all the cities it is connected to)
Problem Definition

- Given a new element, want to add it to an equivalence class \textbf{(union)}
  - Add 18 to the “a mod 5” relation already containing the numbers shown
    - Since $18 \sim 3 \sim 8 \sim 13$, perform a union of 18 with equivalence class of 3, 8, and 13
  - Add Monroe to the city connection relation
    - Everett is connected to Monroe, so add Monroe to the same equivalence class as Everett, Seattle, and Bellingham
Disjoint Set ADT

- **Find**: Given an element, return the “name” of its equivalence class
  - note that we are finding the equivalence class, not the element
- **Union**: Given the “names” of two equivalence classes, merge them into one class
  - may have a new name or one of the two old names
Disjoint Set ADT

- The disjoint set ADT divides elements into equivalence classes and manages the combination of classes depending on the relation of interest
  - 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 belongs to exactly one set (its equivalence class)

- **Dynamic equivalence property**
  - the name of the equivalence class that an element is in may change after a union
  - however, all elements in the class will always have the same equivalence class name
More Formal Definition

- Given a set $U = \{a_1, a_2, \ldots, a_n\}$
- Maintain a partition of $U$, a set of subsets (or equivalence classes) of $U$ denoted by $\{S_1, S_2, \ldots, S_k\}$ such that:
  - each pair of subsets $S_i$ and $S_j$ are disjoint: $S_i \cap S_j = \emptyset$
  - 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
Simple array implementation?

- How about an array implementation?
  - Array A → A[i] holds the class name for element i
  - Running time for Find(i)?
    - just return A[i] : O(1)
  - Running time for Union(i,j)?
    - 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 : O(N)
Linked List Implementation?

• How about linked lists?
  › One linked list for each equivalence class
  › Running time for Find(i)?
    • must scan all lists in worst case: \(O(N)\)
  › Running time for Union(i,j)?
    • just append one list to the other: \(O(1)\)

• Tradeoff between Union-Find – cannot do both in \(O(1)\) time
  › M Finds and N-1 Unions (the max)
    • array \(O(M + N^2)\) or lists \(O(MN+N)\)
Let’s use 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.
  - input string → integer

Up-trees are usually not binary!
Example of Find

Find: Just traverse from the node to the root.

\[
\begin{align*}
\text{find}(f) &= c \\
\text{find}(e) &= a
\end{align*}
\]

Runtime = ?
Example of Union

Union: Just hang one root from the other.

union(c,a)

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

Runtime = ?