02 - Data Structures Part II

9/20/2017

These slides will be put on:
github.com/AuburnACM/Competitive-Programming

Agenda

- Announcements
- Prefix Tree
- Union Find
- More Data Structures
- Competition Review
- Your Turn!

Prefix Trees

- Commonly used on / with strings
- Can be used to represent a set of strings to see if a given string is contained in the set or get all words in a set with a given prefix.
- Also called a “Trie”

Prefix Trees

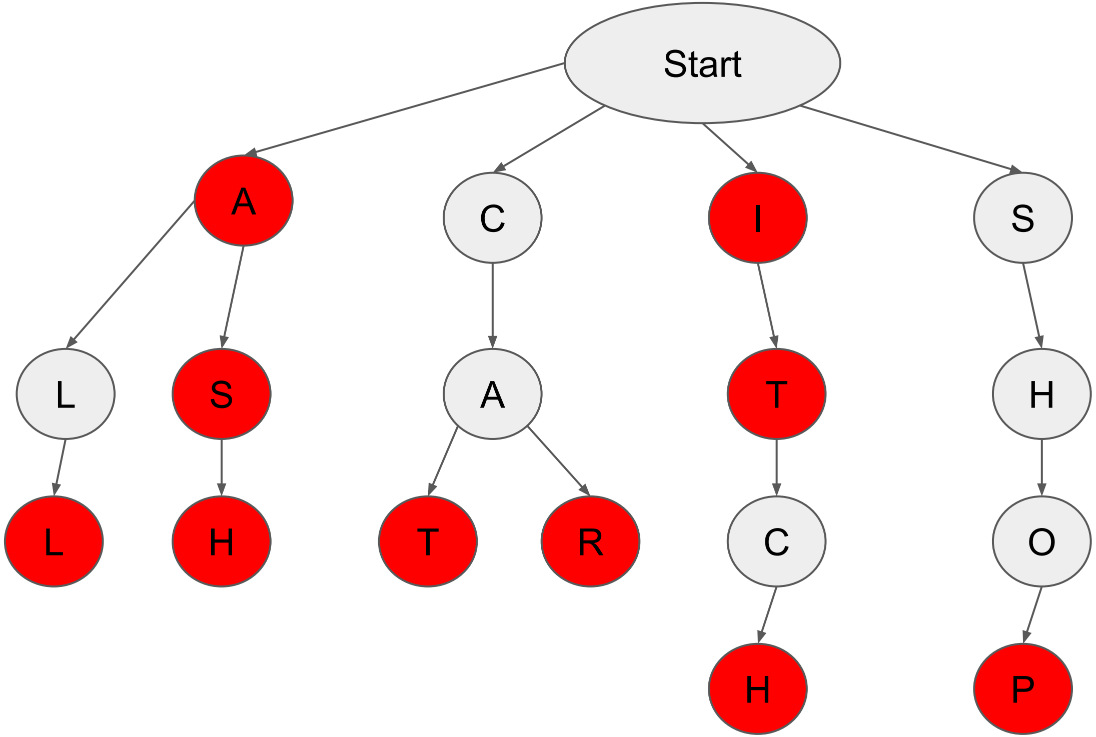
Root - Beginning of word.

Each node represents a letter, and each root to node path represents a string.

Each node’s subtrees will hold all valid words starting with the given root-node string as a prefix.

Each node also contains a flag indicating whether or not this string is a valid word.

Prefix Trees



Prefix Trees - Implementation

```
class PrefixTreeNode {
    boolean endOfWord;
    Map<Character, PrefixTreeNode> children;

    PrefixTreeNode() {
        children = new HashMap<Character, PrefixTreeNode>();
    }
}

class PrefixTree {
    PrefixTreeNode head;

    PrefixTree() {
        head = new PrefixTreeNode();
    }
}
```

Prefix Trees - Implementation

```
class PrefixTree {
     ...
     void add(String s) {
          PrefixTreeNode n = head;
          for(char c : s.toCharArray()) {
               if (!n.children.containsKey(c)) {
                    n.children.put(c, new PrefixTreeNode());
               }
               n = n.children.get(c);
          }
          n.endOfWord = true;
     }
}
```

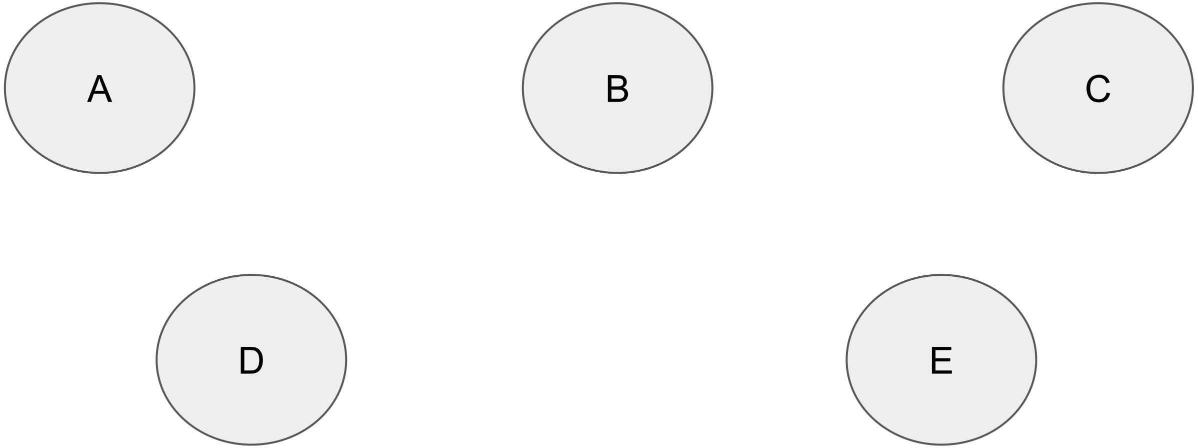
Prefix Trees - Implementation

```
class PrefixTree {
    ...
    boolean contains(String s) {
        PrefixTreeNode n = head;
        for(char c : s.toCharArray()) {
            if (!n.children.containsKey(c)) {
                return false;
            }
            n = n.children.get(c);
        }
        return n.endOfWord;
    }
}
```

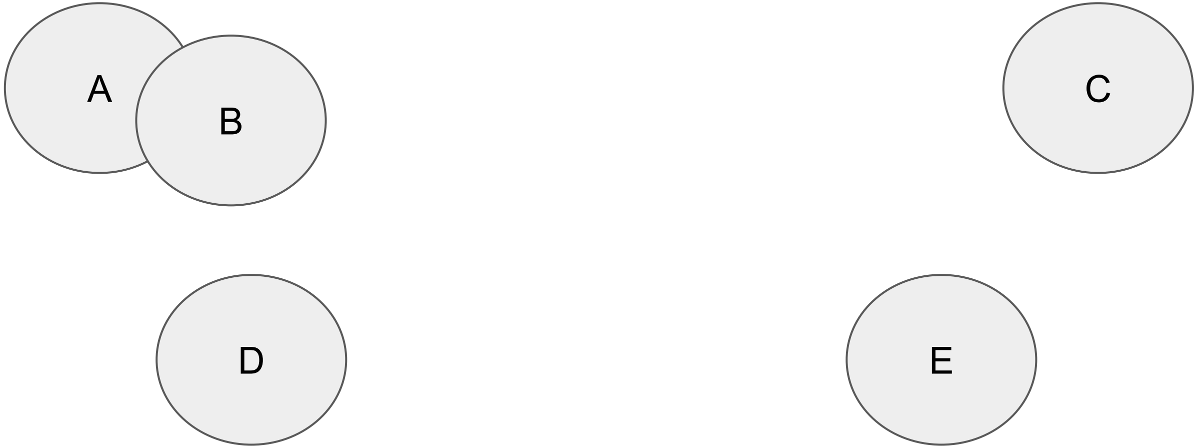
Union Finds (Disjoint Data Sets)

- A union find represents a number of sets of data that initially start off disjoint, but can eventually be combined into larger sets.
- We can then determine if two elements are in the same set.
- This is a transitive operation.

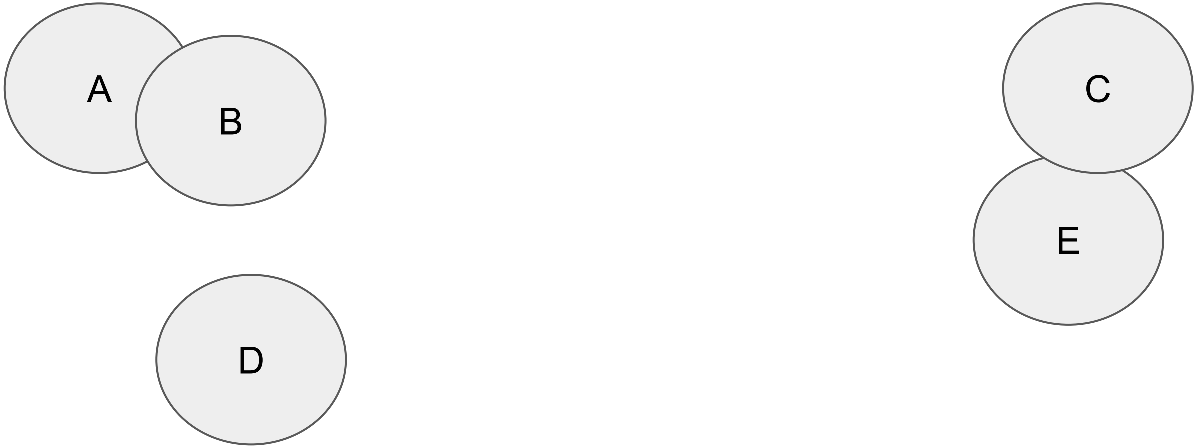
Disjoint Data Set Example



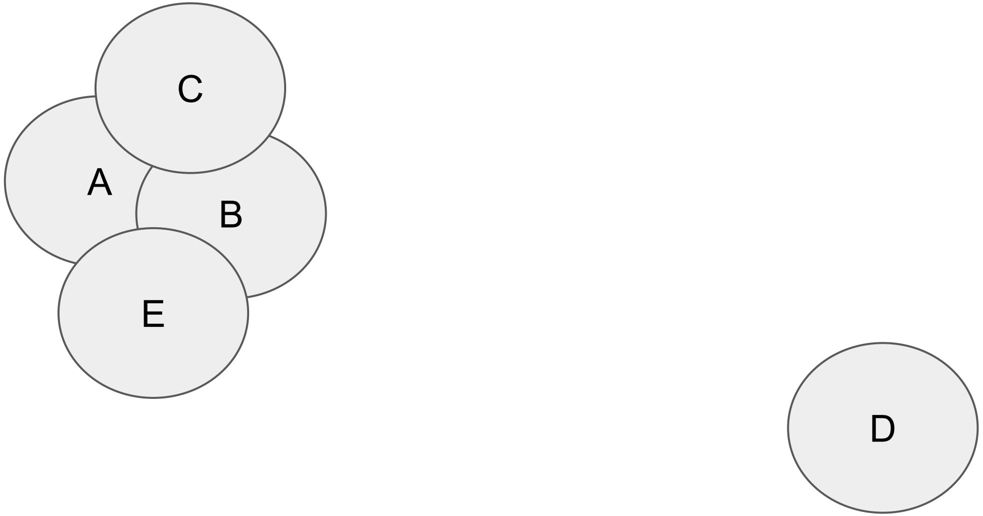
Disjoint Data Set Example



Disjoint Data Set Example



Disjoint Data Set Example



How does a Union-Find address this?

It supports two operations:

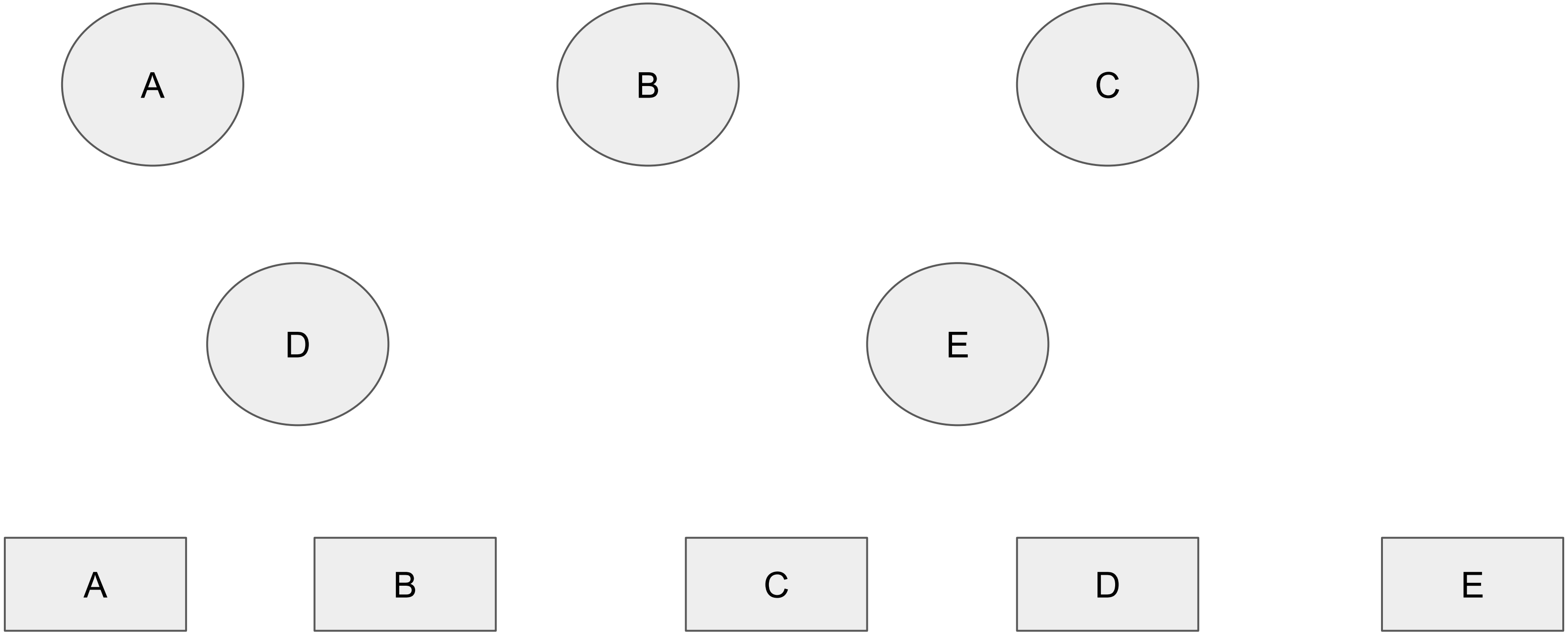
- Union - Merge 2 groups together.

- Find - Find which group an element is done in. All elements in the same disjoint set will return the same value (often called the representative element) for the find operation.

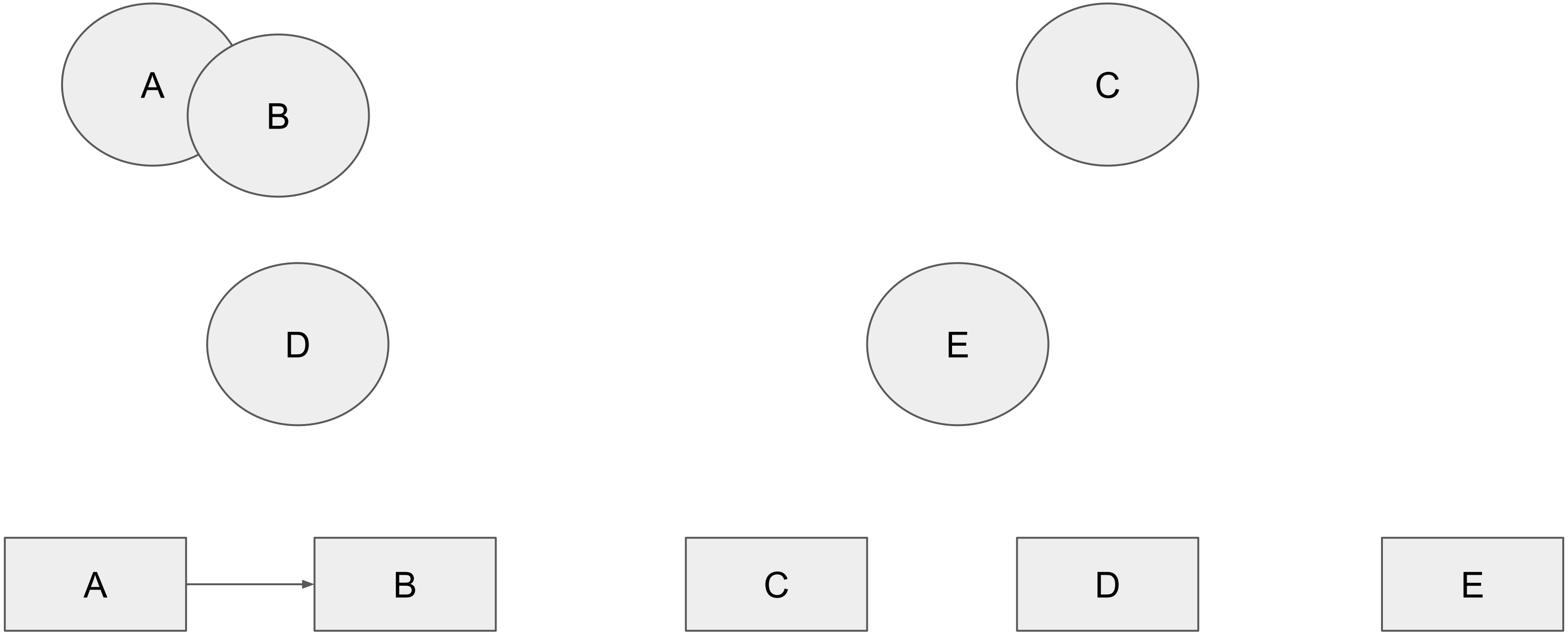
How can we represent this?

- We can use a Directed Acyclic Graph (DAG) To represent these operations.
- Each vertex will point to another element in the set, which creates a path we can follow until we eventually reach one element with no “parents”.
- This can be the representative element for this set.
- To union, make one element’s set a sub-tree of the other.

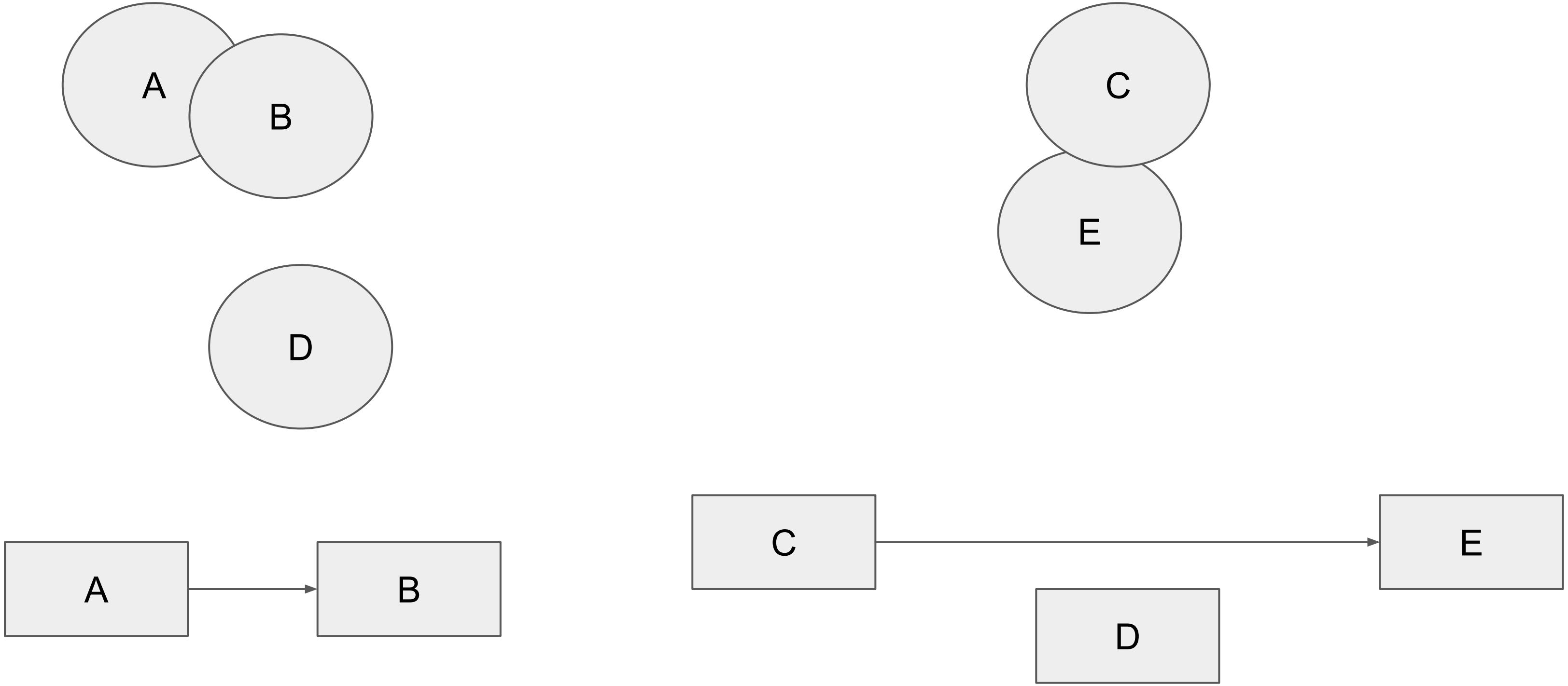
Union - Find Example



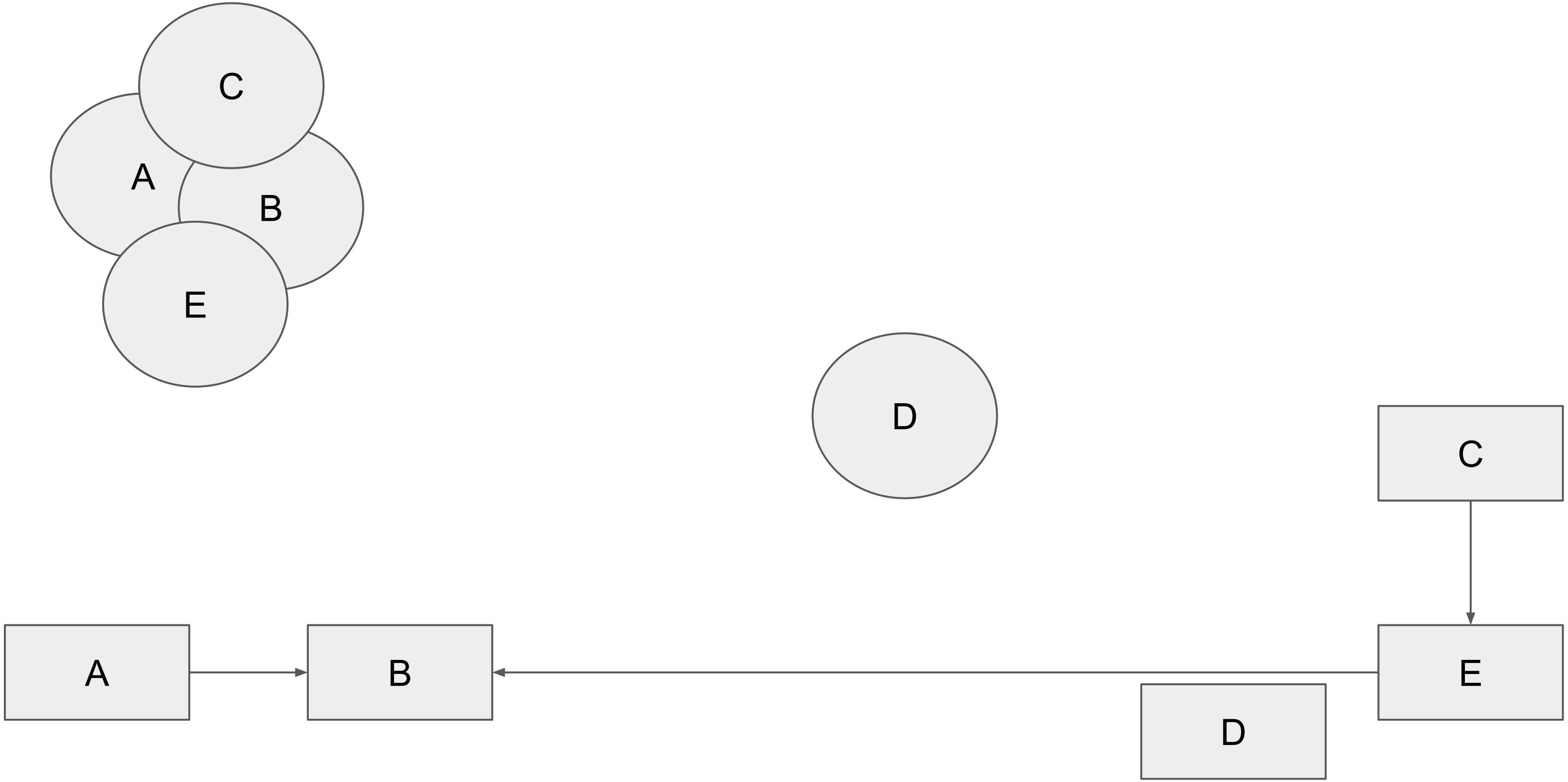
Union - Find Example



Union - Find Example



Union - Find Example



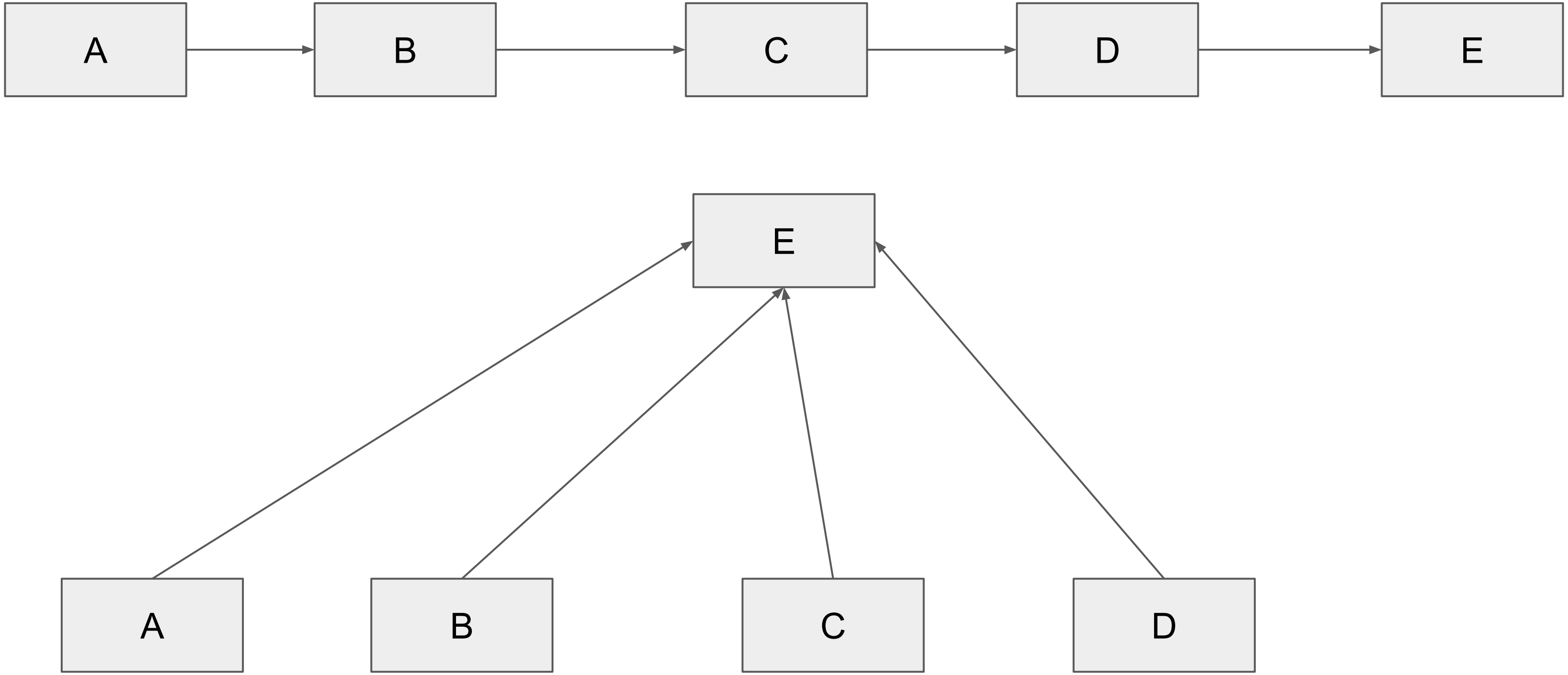
Union - Find Implementation

- Instead of using Nodes, just use an array!
- Each element will point to the index of it’s “parent”.
- If an element points to itself, it has no parents, and is the representative element of each set.
- To **find** the representative element of another element, repeatedly get the parent element, until you get to an element that is it’s own parent.
- To **union** two elements’ sets together, get their representative elements and make one the parent of the other.

Union - Find Implementation

```
class UnionFind {
      int[] parent;

      UnionFind(int N) {
           parent = new int[N];
           for (int i = 0; i < N; i++) parent[i] = i;
      }

      int find(int a) {
           if (parent[a] == a) return a;
           return find(parent[a])
      }

      void union(int a, int b) {
           parent[find(a)] = find(b);
      }
}
```

Union - Find Path Compression



Union - Find Path Compression

- Every time we do a **find** operation, once we find our representative element, but before we return it as our result, set our parent element to be the representative element.

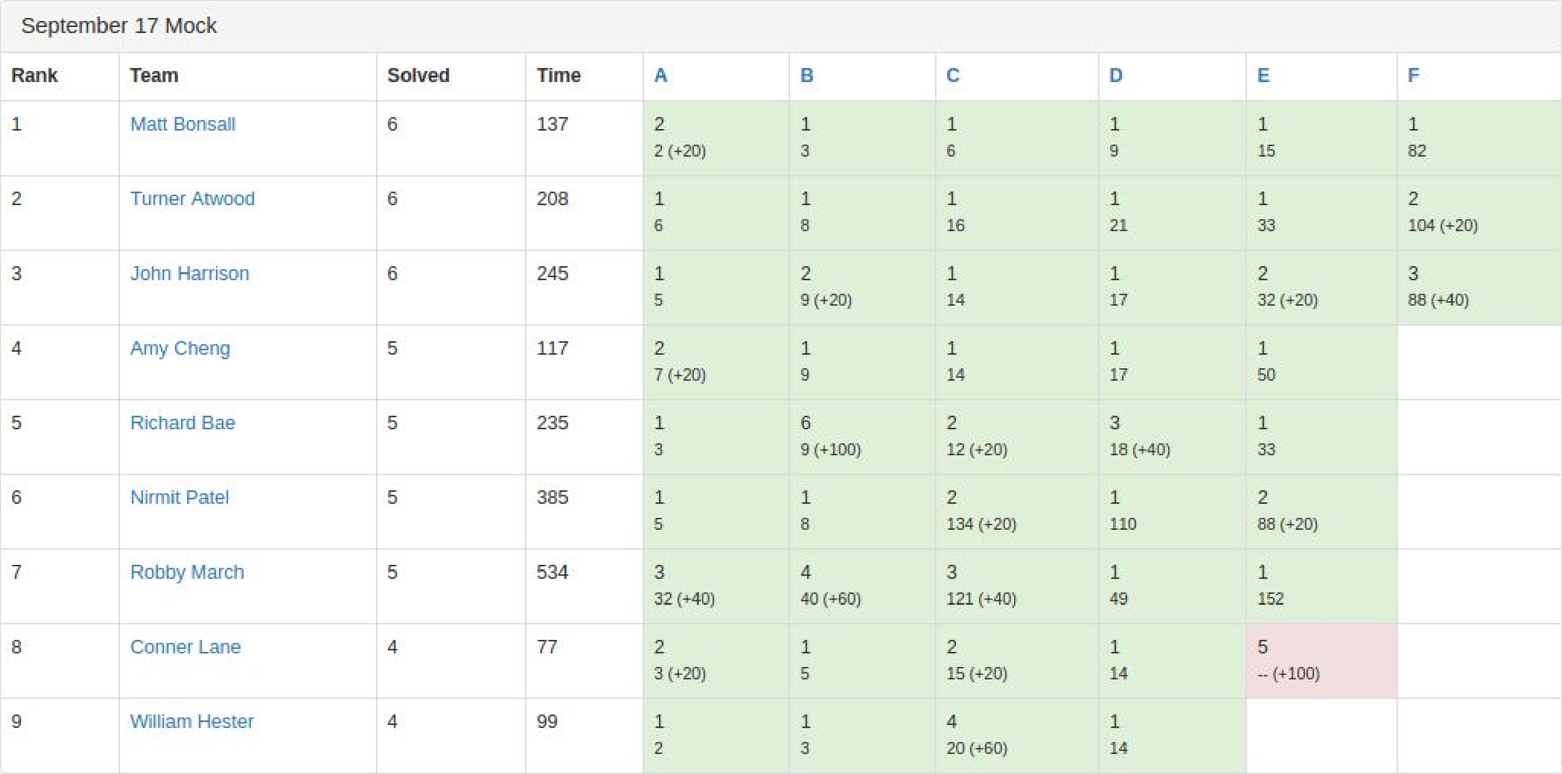
Union - Find v2 Implementation

```
class UnionFind {
      int[] parent;

      UnionFind(int N) {
            parent = new int[N];
            for (int i = 0; i < N; i++) parent[i] = i;
      }

      int find(int a) {
            if (parent[a] == a) return a;
            parent[a] = find(parent[a])
            return parent[a];
      }

      void union(int a, int b) {
            parent[find(a)] = find(b);
      }
}
```

Other Data Structures

- Link - Cut Tree
- Fenwick Tree / Segment Tree
- Suffix Tree / Suffix Array
- K-D Tree

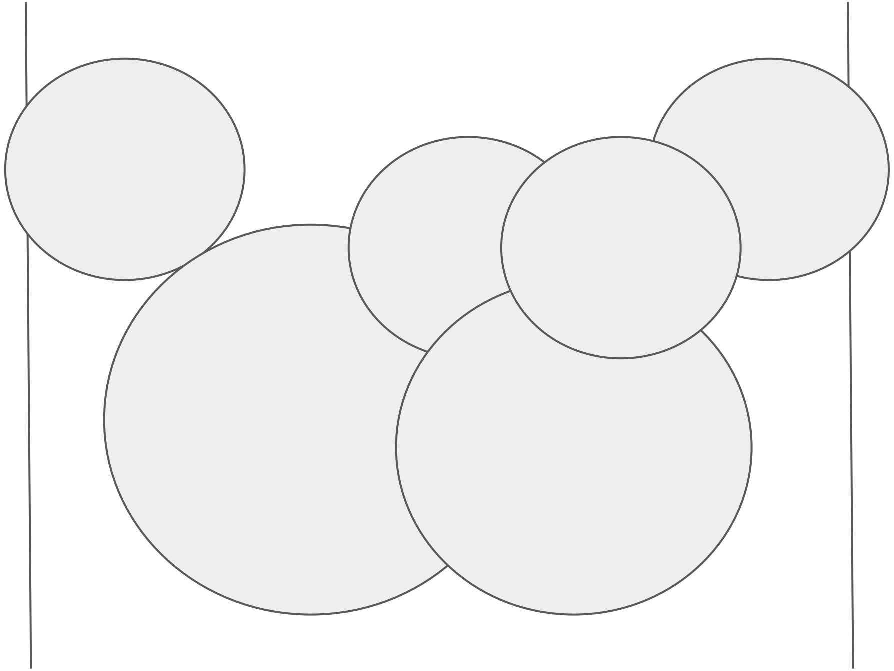
Competition Review



September 17 Mock

Rank	Team	Solved	Time	A	B	C	D	E	F
1	Matt Bonsall	6	137	2 2 (+20)	1 3	1 6	1 9	1 15	1 82
2	Turner Atwood	6	208	1 6	1 8	1 16	1 21	1 33	2 104 (+20)
3	John Harrison	6	245	1 5	2 9 (+20)	1 14	1 17	2 32 (+20)	3 88 (+40)
4	Amy Cheng	5	117	2 7 (+20)	1 9	1 14	1 17	1 50	
5	Richard Bae	5	235	1 3	6 9 (+100)	2 12 (+20)	3 18 (+40)	1 33	
6	Nirmit Patel	5	385	1 5	1 8	2 134 (+20)	1 110	2 88 (+20)	
7	Robby March	5	534	3 32 (+40)	4 40 (+60)	3 121 (+40)	1 49	1 152	
8	Conner Lane	4	77	2 3 (+20)	1 5	2 15 (+20)	1 14	5 -- (+100)	
9	William Hester	4	99	1 2	1 3	4 20 (+60)	1 14		

Problem F - Tower Defense



Your Turn!

Give these Kattis problems a try!

Prefix Trees:

- Bing it On! (https://open.kattis.com/problems/bing)

Union Find:

- Union-Find (https://open.kattis.com/problems/unionfind)
- Virtual Friends (https://open.kattis.com/problems/virtualfriends)