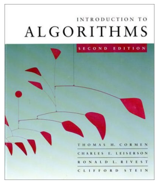
CS 445



Union/Find
Aka: Disjoint-set forest

Alon Efrat

Problem definition

Given: A set of **atoms** ***S={1,2…n}***
E.g. each represents a commercial name of a drugs.
This set consists of different disjoint subsets.

Problem: suggest a data structures that efficiently supports two operations

- **Find(*i,j*)** – reports if the atom *i* and atom ***j*** belong to the same set.
- **Union(*i,j*)** – unify (merged) all elements of the set containing ***i*** with the set containing ***j***.

•*Example – on the board.*

Naïve attempts

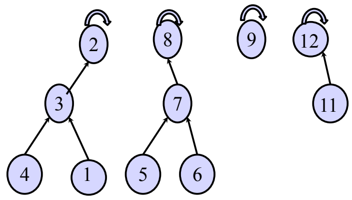
Idea: Each element "knows" to which set it belongs
(recall – each atom belongs to exactly one set)

Bad idea: once two sets are merged, we need to scan all elements of one set and "tell" them that they belong to a different set – requires lots of work if the set is large.

A Promising Attempt

- Store a forest of trees.
- Each set is stored as a tree (each node is an atom)
- Every node points to the parent
  (different than standard trees)

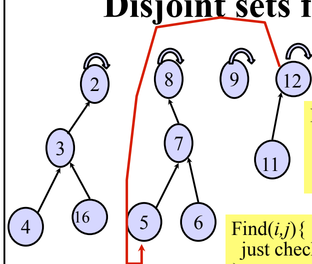
Only the root "knows" the name of the set.



So the `name' of the set *{2,3,4,1}* is **2.**
The name of the set of *{5,6,7,8}* is **8**.
The name of the set of *{9}* is **9**.
The name of the set of *{11,12}* is **12.**

To find if two atoms belong to the same set, just check if they belong to same tree: Follow the parent pointers from each of them up all the way to the root. Check if this is the same root.

Disjoint sets forests - cont



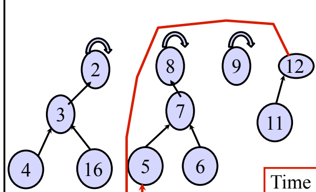

```
Find_root(j){
   If (p[j] ≠ j) return Find_root(p[j]);
   // p[j]  - points to the parent
   Else return j ; }
```

```
Find(i,j){
  just check if  Find_root(i) == Find_root(j)
}
```

```
Union(i,j){
        Let r = Find_root(j)
        p[r]=i
}
```

Example – Union(5,11)

It this efficient?

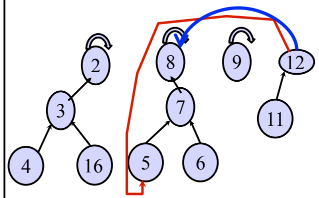


Time per operation depends on the height of the tree. Might be $\Theta(n)$ in the worst case.

(Prove)

So n operations takes $\Theta(n^2)$

Could we do better ?



Example – Union(5,11)

Improved union operation – version 1

First improvement

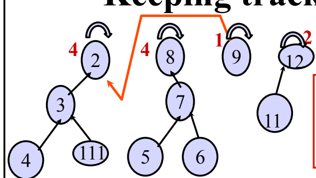
```
Union(i,j){
        Let r = Find_root(j)
        p[r]=Find_root(i)
  /* rather than p[r]= i ;  */
}
```

Note that we can also do

```
Union(i,j){
        Let r = Find_root(i)
        p[r]=Find_root(j)

}
```

Keeping tracks of # nodes



Every **root** (only roots) stores the number of nodes in its tree
Let ***r.n*** denote this field in the root *r*.

```
Union(i,j){
  Let r1 = Find_root(i);  Let r2 = Find_root(j);
  /* connect the root of the small tree as a child of the root of
        the larger tree */
  if (r1.n< r2.n )  {    p[r1]=r2 ;    r2.n += r1.n ; }
  else  {  p[r2]=r1 ;    r1.n += r2.n    }
}
```

Example: Union(9,3)

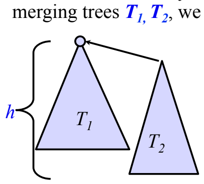
Proving bounds on the height

Assume we start from a forest where each node is a singleton
(a set of one element), and we perform a sequence of union operations.

Lemma: The height of every tree is $\leq \log_2 n$.   (*n* – number of atoms)

Proof: Show by induction that every tree of height ***h*** has $\geq 2^h$ nodes.

Assume true for every tree of height ***h'<h***, and assume that after merging trees T_1, T_2, we created a tree of height exactly ***h***.



T_2 has height is exactly $h-1$, so it has $\geq 2^{h-1}$ nodes.

T_1 also has 2^{h-1} nodes, (why ?)

Together they have $2^{h-1} + 2^{h-1} = 2^h$ nodes.

Further improvement: path compression

So far we know that every tree has height $O(\log n)$, so this bounds the time for each operation.

Path compression: during either union or find operation, we scan a sequence of nodes on our way from a node j to the root.

Idea: set the parent pointer of all these node to points to the root. (Slightly more work to perform it, but pays off in next operations)

```
Find_root( j ){
  If p[j] ≠ j  then p[j]=Find_root(p[j]);
  return p[j]
}
```

Make sense – but how fast is it ?

Thm: Consider a set of n atoms

Any sequence of m U/F operations takes $O(\, m\, \alpha(n)\,)$.

Here $\alpha(n)$ is the inverse function of Ackerman function, and is approaching infinity as n approaching infinity.

However, it does it very slowly.

$\alpha(n) < 4$ when $n < 10^{80}$.

Connected Components in Undirected graphs

Let $G(V,E)$ be a graph.

We say that a subset C of V is a connected component (CC) if

1. for every pair $u,v \in C$, there is a path connecting them, and all the vertices of this path belong to C. And in addition
2. For any vertex $u \in C$, and any vertex v that does not belong to C, there is no path in $G(V,E)$ connecting u to v.

Examples 1: If $G(V,E)$ is connected then V is a CC.

Example 2: If $G(V,E)$ contains no edges, then every node is CC, which contains only itself.

Example 3: If $G(V,E)$ is a tree, and we deleted an edge from E, then in the resulting graph there are 2 CCs.

Minimum Spanning Trees

G(V,E) with positive weights on its edges.

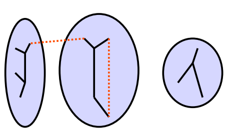
A Minimum spanning tree (MST) is any graph ***T*** such that

1. Every vertex of *V* appears in ***T***, and
2. ***T*** is connected (has a path between every two vertices)
3. ***T*** is a subset of ***E***
4. Sum of weights of its edges are as small as possible

Application: Kruskal algorithm

Kruskal algorithm for finding a MST.
Input: Graph G(V,E). Output: Minimal Spanning Tree for G.

1) Assume $E=\{e_1,..e_m\}$ is sorted from cheapest edge to most expensive edge.
2) Set *S=EmptySet.*
3) For *i=1..m*
4) If $e_i \cup S$ does not contain a cycle, add e_i to S
   / We use U/F structure to answer last test */*



If *E* is sorted, then the time is $O(|E|\ \alpha(|E|))$