Union/Find
Aka: Disjoint-set forest

Alon Efrat

Problem definition

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

Problem: Suggest a data structure that efficiently supports
two operations
• \( \text{Find}(i, j) \) – reports if the atom \( i \) and atom \( j \) belong to the
same set.
• \( \text{Union}(i, j) \) – unify (merged) all elements of the set
containing atom \( i \) with the set containing \( j \).

• Example – on the board.

Naïve attempts

Idea: Each element “knows” to which set it belongs
(recall – each element 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 – lots of work if the set is large.
A Promising attempts

**Idea:** Store each set as a tree. 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 of \{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

Example – Union(5,11)

It this efficient?

We want short trees.
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 \( r1 = \text{Find}_\text{root}(i) \); Let \( r2 = \text{Find}_\text{root}(j) \);

// 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 obtained a tree of height exactly \( h \).

\( T_1 \) has height exactly \( h-1 \), so it has \( \geq 2^{h-1} \) nodes.

\( T_2 \) must have more nodes (why?) so it also has \( \geq 2^{h-1} \) nodes.

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.

**Find_root\( (j) \)**:

If \( p[j] \neq j \) then \( p[j]=\text{Find}_\text{root}(p[j]) \);
return \( p[j] \)
**Make sense – but how fast is it?**

**Thm:** Any sequence of \( m \) U/F operations takes \( O(m \alpha(n)) \) on a set of \( n \) atoms.

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

However, it does it very slowly.

\[ \alpha(n) < 4 \text{ when } n < 10^{16} \]

**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 \( S=\text{EmptySet}. \)
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 \( O(|E| \alpha(|E|)) \).