Lecture Notes on
Data Structures

M1522.000900

Seoul National University

Fall 2022

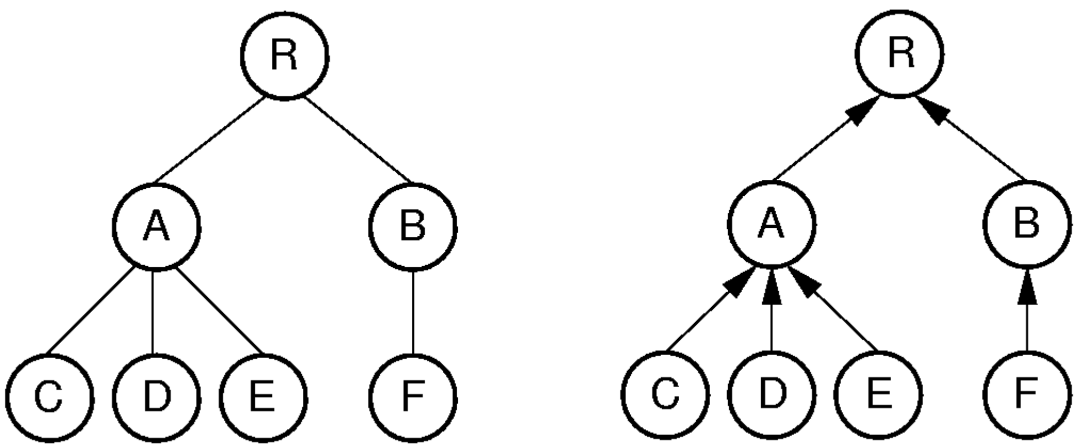
Part VII

Union / Find

Problem 1

Consider a collection (or forest) of (not necessarily binary) trees. For a given pair of nodes x and y (randomly selected from the forest), determine whether x and y belong to the same tree or not.

- Kruskal's MST algorithm will reject $\overline{xy}$ if x and y belong to the same tree.
- We can determine that x and y belong to the same tree iff $\text{root}(x) = \text{root}(y)$.
- Can you write an efficient algorithm for this task using a traditional pointer-based data structure?

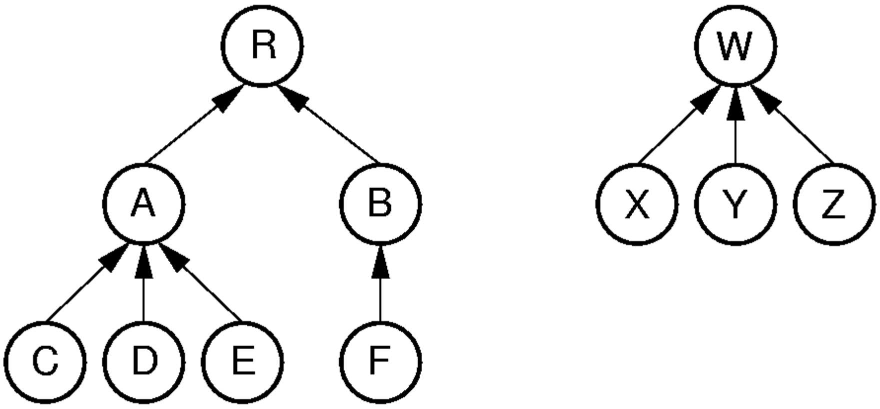


- Node structure:

Key	Parent

- Used for general trees. (There are a few alternatives too.)
- Limited applicability but useful for Union/Find.

Parent Pointers for a Forest



	0	0	1	1	1	2		7	7	7
R	A	B	C	D	E	F	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10

- Commonly stored in an array. The order of elements does not matter.
- The number of nulls = the number of trees.

Union/Find Algorithms

Algorithm 1 (Union/Find)

```
Union(i,j)          // Merge two trees nodes i & j belong to
   root1 = Find(i);
   root2 = Find(j);
   if (root1 != root2) Parent[root2] = root1;

Find(i)     // Return the root of the tree node i belongs to
   if (Parent[i] == -1) return i;
   else return Find(Parent[i]);
```

For a given equivalence relation R on a set S, find all equivalence classes of S with respect to R.

- Classify the elements of S into a partition (*i.e.*, disjoint subsets) of S based on the relation R (*i.e.*, related pairs in R).
- For example, consider an equivalence relation $R = \{(a_1, a_2), (a_1, a_5), (a_3, a_4)\}$ on a set $S = \{a_1, a_2, a_3, a_4, a_5\}$. Then, the partition of S with respect to R is $\{\{a_1, a_2, a_5\}, \{a_3, a_4\}\}$.

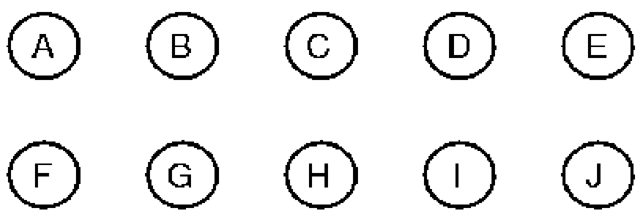
Algorithm 2

```
// Initially, each element is in its own subset (or tree).
for(i=0; i < N ;i++) Parent[i] = -1;
for each pair (x, y) do
   Union(x,y);     // Merge two subsets x and y belong to.
```


Example 1 (Finding Equivalence Classes)

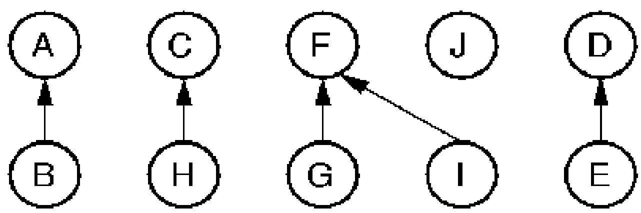
For a given set of related pairs (A, B), (C, H), (F, G), (D, E), (F, I), (A, H), (G, E), (E, H), classify the ten elements A through J into disjoint subsets.

A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



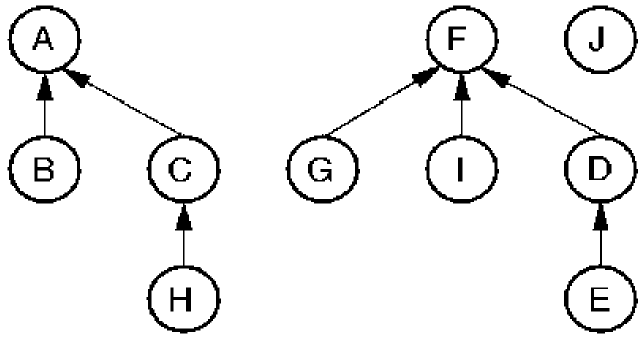
After processing (A, B), (C, H), (F, G), (D, E), (F, I):

	0			3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



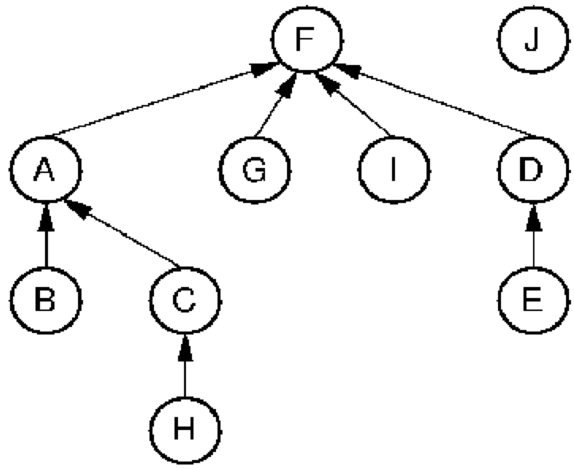
After processing (A, H), (G, E):

	0	0	5	3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



After processing (E, H):

5	0	0	5	3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



Cost of Finding Equivalence Classes

For a set of n elements with m related pairs, the cost is $\mathcal{O}(m \times n)$ because

- Union calls Find twice for each related pair, and takes $\mathcal{O}(1)$ time to merge two trees.
- Find takes $\mathcal{O}(\textit{height})$ times, and $\textit{height} \in \mathcal{O}(n)$.

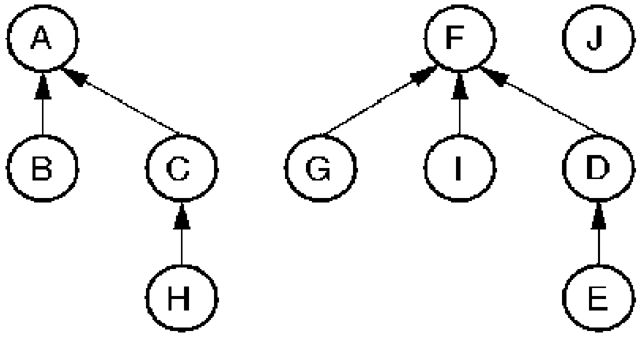
Optimizations for Union/Find:

- **Union-by-size** makes the smaller tree a subtree of the larger one.
  - Keep track of the size of a tree in the `Parent` field of the root.
  - If `Parent[i]` < 0, then $|$`Parent[i]`$|$ is the tree size.
- **Path compression**: Find(i) makes all nodes on the path $i \rightsquigarrow \textit{root}(i)$ children of the root. Use the following algorithm.

```
if (Parent[i] < 0) return i;
else return Parent[i] = Find(Parent[i]);
```

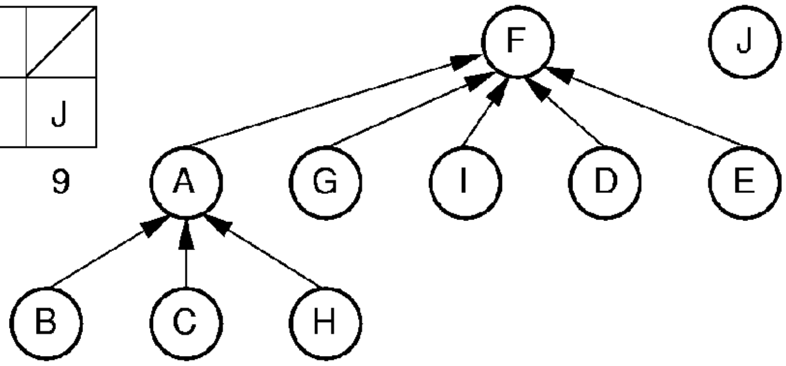
Before processing (H, E):

	0	0	5	3		5	2	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



After processing (H, E) with path compression:

5	0	0	5	5		5	0	5	
A	B	C	D	E	F	G	H	I	J
0	1	2	3	4	5	6	7	8	9



Cost of Improved Union/Find

Lemma 1

Union-by-size limits the depth of trees to $\mathcal{O}(\log n)$.

Proof. (Sketch) Each time two trees are Union'ed, no more than half of the nodes have their levels incremented by one. □

Theorem 2

With both Union-by-size and Path compression applied, the running time of m Unions and Finds is $\mathcal{O}(m \times \log^* n)$.

$\log^* n = k$ if $\underbrace{\log \log \ldots \log n}_{k \text{ times}} \leq 1$.

For example, $\log^* 65536 = 4$ and $\log^* 2^{65536} = 5$. Therefore, the running time is almost $\mathcal{O}(m)$, because $\log^* n$ is very close to a constant.

Remark 1

Union-by-height that makes the shallower tree a subtree of the other also limits the depth of trees to $\mathcal{O}(\log n)$. However, it is less obvious how Union-by-height can be combined with path compression, because path compression changes the height of a tree.