Union-Find

Giovanni Righini

UNIVERSITÀ DEGLI STUDI DI MILANO

Union-Find data-structure

Assume we have a set N of n items, partitioned into subsets.

We need to execute two operations on N:

- testing whether two items are in the same subset or not (*Find*);
- merging two subsets into one (*Union*).

Possible implementations:

- arrays;
- linked lists;
- trees.

Each subset is identified by a representative item: its head.

Union-Find data-structure: arrays

For each $i \in N$:

- $v[i]$ is the item (ID, value, pointer...);
- $h[i]$ is the head of its subset (index).

i	1	2	3	4	5	6	7	8	9	10
ID	A	B	C	D	E	F	G	H	I	L
h	1	1	1	6	9	6	6	6	9	9

correponds the partition $\{A, B, C\}, \{D, F, G, H\}, \{E, I, L\}$.

Complexity:

- *Find*: $O(1)$ with the test $h[i] = h[j]$.
- *Union*: $O(n)$ because all values of h in a merged subset must be updated.

Union-Find data-structure: arrays

i	1	2	3	4	5	6	7	8	9	10
ID	A	B	C	D	E	F	G	H	I	L
h	1	1	1	6	9	6	6	6	9	9
card	3	?	?	?	?	4	?	?	3	?

Union$(2, 4)$:

i	1	2	3	4	5	6	7	8	9	10
ID	A	B	C	D	E	F	G	H	I	L
h	6	6	6	6	9	6	6	6	9	9
card	?	?	?	?	?	7	?	?	3	?

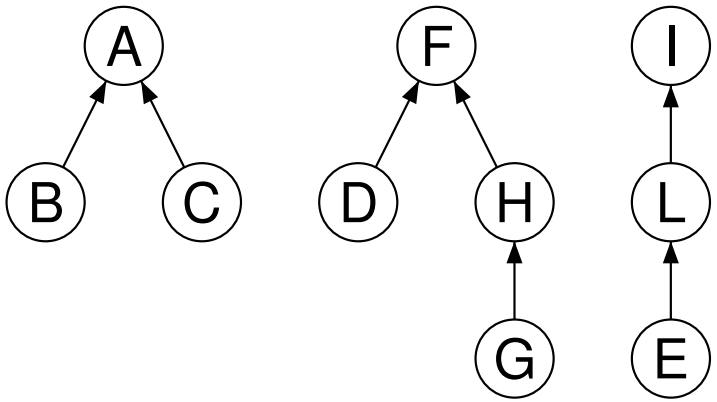
Modifying the smallest subset yields total complexity $O(n \log n)$ for all *Union* operations, i.e. amortized complexity $O(\log n)$ for each *Union* operation.

The same results hold for the implementation with linked lists.

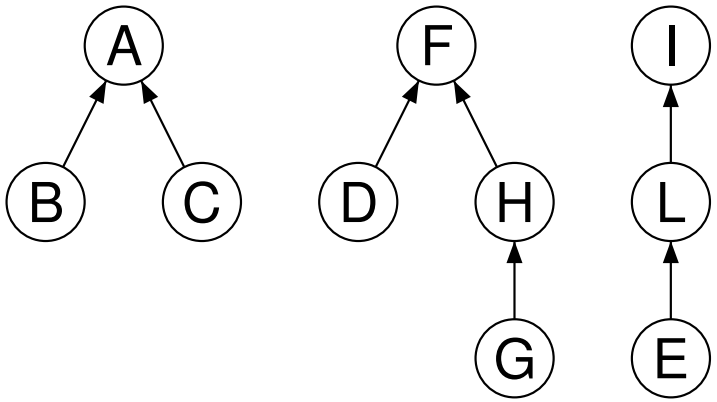
Union-Find data-structure: trees

Each subset is a tree; its head is the root element.

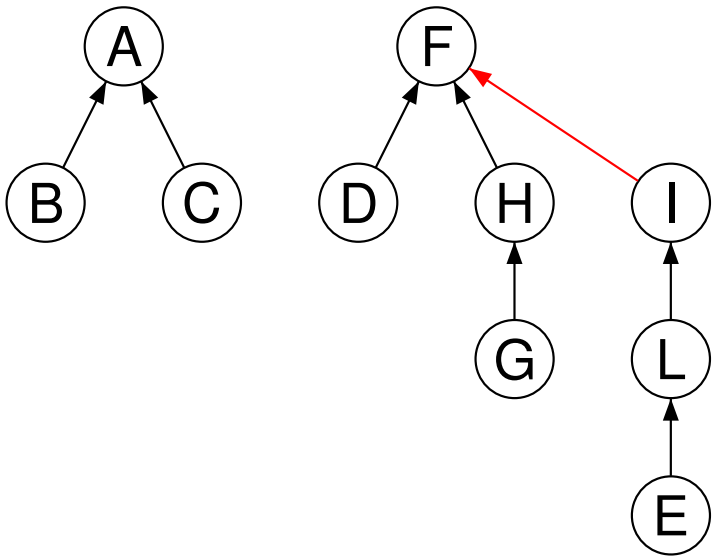
ID	A	B	C	D	E	F	G	H	I	L
pred	-	A	A	F	L	-	H	F	-	I



Union-Find data-structure: trees



$Union(E, G)$: $head(E) = I$, $head(G) = F$.
One of the roots is appended to the other.



Union-Find data-structure: trees

Complexity:

- *Find*: $O(n)$ (when a tree is a path)
- *Union*: $O(1)$

Objective: improve the complexity of *Find*: balance the trees.

Ideas for *Union*:

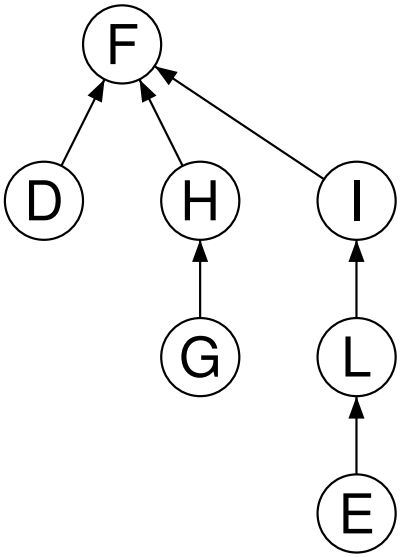
- Union-by-size: append the root of the tree with fewer nodes to the root of the tree with more nodes;
- Union-by-rank: append the root of the tree with smaller depth to the root of the tree with larger depth.

Complexity of *Find*: $O(\log n)$.

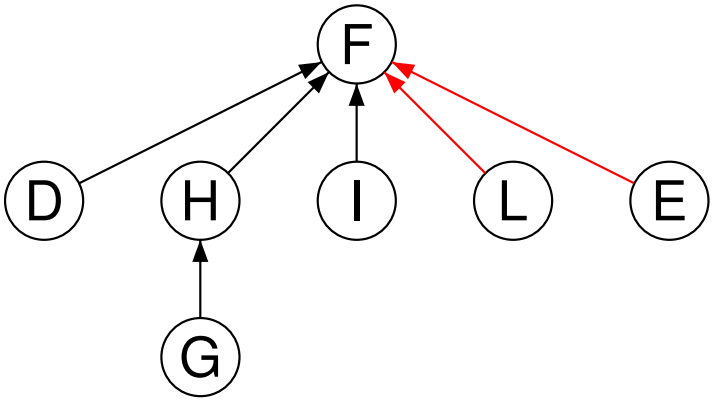
Union-Find data-structure: trees

Idea for *Find*: path compression: when the root is searched from node i by $Find(i)$, all nodes between i and the root are made successors of the root.

Before $Find(E)$



After $Find(E)$



Total complexity of m executions of *Find* with $m > n$ is $O(m \log^* n)$.

The log^* function

$$\log^* 2 = 1$$

$$\log^* 2^2 = \log^* 4 = 2$$

$$\log^* 2^{(2^2)} = \log^* 16 = 3$$

$$\log^* 2^{(2^{(2^2)})} \log^* 65536 = 4$$

$$\log^* 2^{(2^{(2^{(2^2)})})} = 5$$

For all ”reasonable” values of n, $\log^* n \leq 5$.

Even better bounds were proven by Tarjan.