Lecture 21: Union-Find

(CLRS 21.1-21.3)

June 19th, 2002

1 Union-Find

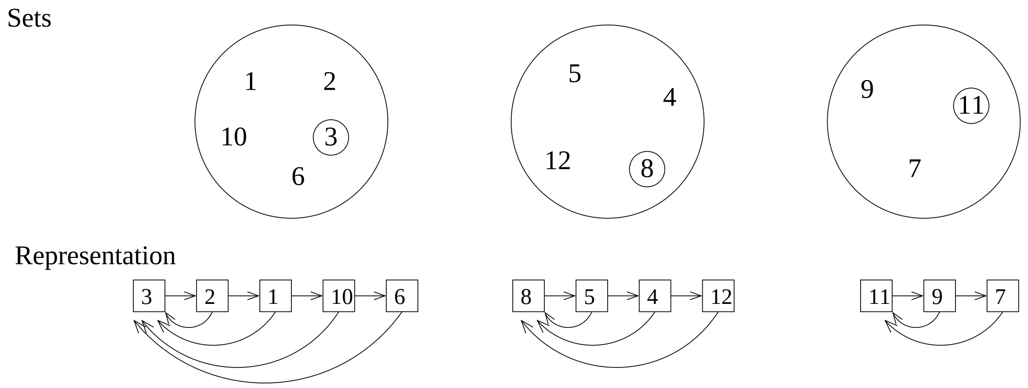
- We discussed Kruskal's minimum spanning tree algorithm

```
KRUSKAL

T = ∅
FOR each vertex v ∈ V DO
    MAKE-SET(v)
OD
Sort edges of E in increasing order by weight
FOR each edge e = (u, v) ∈ E in order DO
    IF FIND-SET(u) ≠ FIND-SET(v) THEN
        T = T ∪ {e}
        UNION-SET(u, v)
    FI
OD
```

- Kruskal's algorithm uses a *Union-Find* data structure supporting:
  - MAKE-SET(v): Create set consisting of v
  - UNION-SET(u, v): Unite set containing u and set containing v
  - FIND-SET(u): Return unique representative for set containing u
- In the algorithm we performed $|V|$ MAKE-SET, $|V| - 1$ UNION-SET, and $2|E|$ FIND-SET operations.
- Simple solution to Union-Find problem (maintain set system under FIND-SET and UNION-SET)
  - Maintain elements in same set as a linked list with each element having a pointer to the first element in the list (unique representative)

Example:



- Make-Set(v): Make a list with one element $\Rightarrow O(1)$ time
- Find-Set(u): Follow pointer and return unique representative $\Rightarrow O(1)$ time
- Union-Set(u, v): Link first element in list with unique representative Find-Set(u) after last element in list with unique representative Find-set(v) $\Rightarrow O(|V|)$ time (as we have to update all unique representative pointers in list containing u)

- With this simple solution the $|V|-1$ Union-Set operations in Kruskal's algorithm may take $O(|V|^2)$ time.

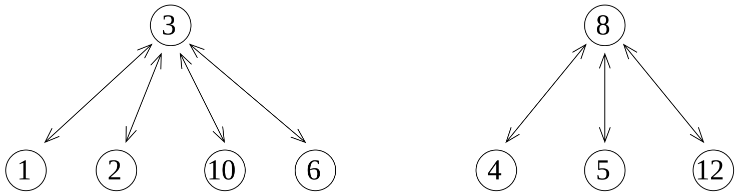
- We can improve the performance of Union-Set with a very simple modification: Always link the smaller list after the longer list ($\Rightarrow$ update the pointers of the smaller list)
  - One Union-Set operation can still take $O(|V|)$ time, but the $|V|-1$ Union-Set operations takes $O(|V|\log|V|)$ time altogether (one Union-Set takes $O(\log|V|)$ time *amortized*):
    - Total time is proportional to number of unique representative pointer changes
    - Consider element u:
      After pointer for u is updated, u belongs to a list of size at least double the size of the list it was in before
      $\Downarrow$
      After k pointer changes, u is in list of size at least 2^k
      $\Downarrow$
      Pointer can be changed at most $\log|V|$ times.

- With improvement, Kruskal's algorithm runs in time $O(|E|\log|E| + |V|\log|V|) = O((|E| + |V|)\log|E|) = O(|E|\log|V|)$ like Prim's algorithm.

1.1 Improved Union-Find

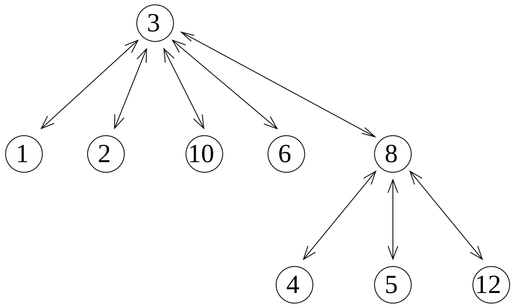
- It turns out that Union-Find can be improved (but without leading to an improvement of Kruskal's algorithm)
  - Linked list representation can also be viewed as trees of height 1

    Example :



  - Instead of updating root pointers when performing Union-set, we could just link one tree below the root of the other

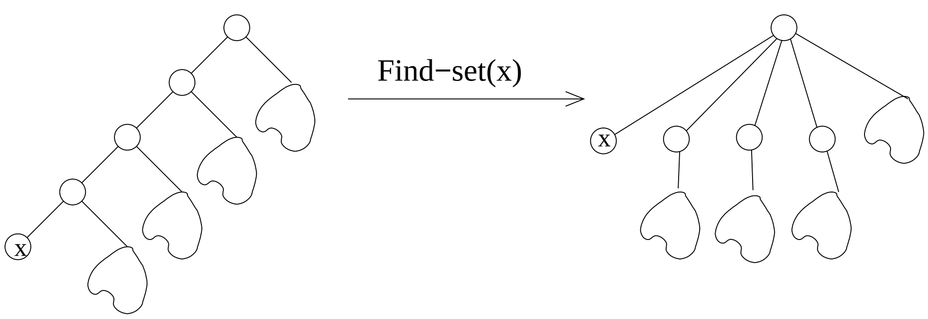
    Example: Union-set(2,6)



    Union-Set and Find-Set takes $O(\log |V|)$ time if we always insert small tree below larger tree (trees have height $O(\log |V|)$)
    $\Downarrow$
    $|E|$ Find-set operations takes $O(|E| \log |V|)$ time
  - If we furthermore perform *path-compression*, $|E|$ Find-set operations can be performed even faster

    Path-compression: When following path during Find-Set we link traversed nodes directly to the root:

    Example :



    Note that a lot of paths are shortened (decreasing time spent on future Find-set operations) without using extra time

It can be shown that $O(|E| \log^* |V|)$ is the total time used on the $O(|E|)$ FIND-SET and UNION-SET operations

- $\log^* n$ is an extremely slow growing function
  - Consider $g(n) = \begin{cases} 2^1 & \text{if } i = 0 \\ 2^2 & \text{if } i = 1 \\ 2^{g(n-1)} & \text{if } i \geq 2 \end{cases}$

    $\Downarrow$

    $g(0) = 2$

    $g(1) = 2^2 = 4$

    $g(2) = 2^{2^2} = 2^4 = 16$

    $g(3) = 2^{2^{2^2}} = 2^{16} = 65536$

    $\vdots$

    $g(i) = 2^{2^{\cdot^{\cdot^{2}}}}$ (2-stack of height i)

    $\Downarrow$

    $g(n)$ *extremely* fast growing function.
  - Define $\log^{(i)} n = \begin{cases} n & \text{if } i = 0 \\ \log \log^{(i-1)} n & \text{otherwise} \end{cases}$
  - $\log^* n = \min\{i \geq 0 : \log^{(i)} n \leq 1\}$

    $\Downarrow$

    $\log^* n$ is minimal number of times we need to take log to get below 1

    $\Downarrow$

    $\log^* n$ is inverse of $g(n)$

    $\Downarrow$

    $\log^* n$ *extremely* slow growing function
  - $\log^* n \leq 5$ for all practical values of n
  - One can even prove that with path-compression $O(|E| \cdot \alpha(|V|))$ is the total time spent on $|E|$ FIND-SET operations, where $\alpha(n)$ is a function growing even slower than $\log^* n$ (Inverse Ackerman function)
    * $\alpha(n) < 4$ for all practical values of n