Disjoint Sets Data Structure: Implementation and Applications

Anjeneya Swami Kare

Assistant Professor
School of CIS
University of Hyderabad

Table of contents

Outline

Disjoint Sets: definition

- Need to maintain a collection of sets.
- Sets are dynamic and disjoint.
- Each set has a representative (a member of the set).

Disjoint Sets: Operations

MAKE-SET(x)

- Create a new set S_x whose only member is x.
- x will be the representative of the set.

UNION(x, y)

- Unite the sets that contain x and y, (say S_x and S_y) into a new set $S_x \cup S_y$.
- S_x and S_y will be destroyed.
- Representative of S_x (or S_y) will be the new representative of $S_x \cup S_y$.

FIND(x)

- Return the representative of the set containing x (S_x).

Disjoint Sets: Example

Disjoint Sets: Time Complexity

- n: The number of MAKE-SET operations.
- m: The total number of MAKE-SET, UNION and FIND operations.
- The number of UNION operations $< n$.
- The total number of operations $m \geq n$.
- We analyze time complexity with respect to n and m for a particular application.

Disjoint Sets Data Structure also known as UNION-FIND Data Structure.

Outline

Disjoint Sets: Implementation

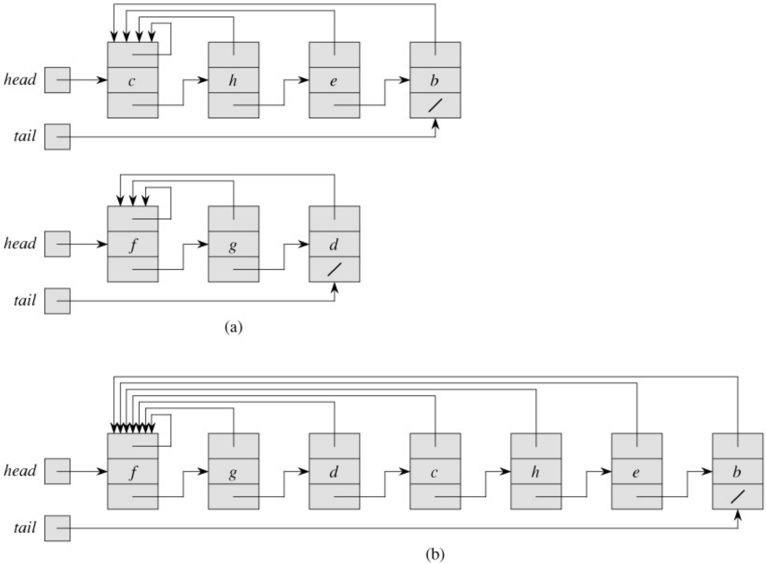
How do you Implement?

Disjoint Sets: Implementation

How do you Implement?

- Using Linked Lists.
- Using Rooted Trees (Forests).
- Heuristics to improve the running time.

Implementation: Using Linked Lists



[1]

[1] source: Introduction to algorithms, CLRS

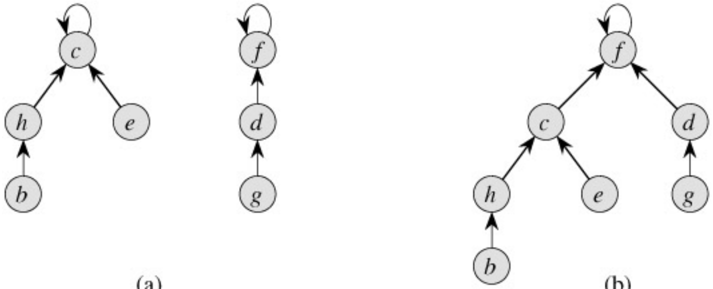
Implementation: Using Linked Lists

Using Linked lists worst case time complexity is $\Theta(m + n^2)$.

Weighted Union Heuristic

- Append the smaller set to the larger set.
- Time complexity improves to $\Theta(m + n \log n)$.

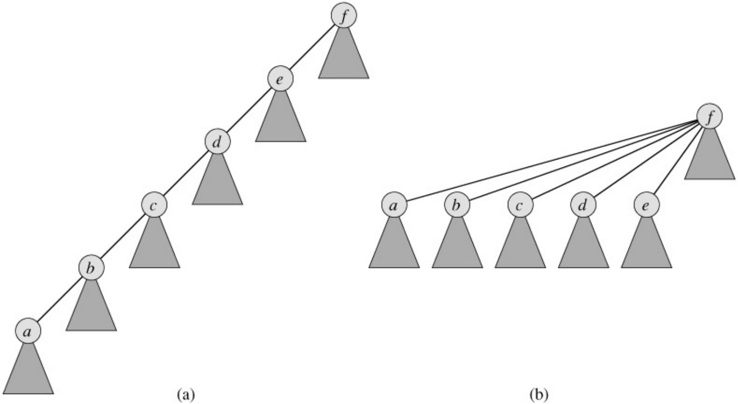
Implementation: Rooted Forests



(a) (b)

We can make tree with smaller height points to the tree with larger height (Union by Rank)

Implementation: Rooted Forests



Make all nodes in the FIND-PATH points to root (Path Compression)

Implementation: Rooted Forests

```
MAKE-SET(x)
p[x] ← x
rank[x] ← 0

UNION(x, y)
LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
if rank[x] > rank[y]
   then p[y] ← x
   else p[x] ← y
        if rank[x] = rank[y]
           then rank[y] ← rank[y] + 1
```

The FIND-SET procedure with path compression is quite simple.

```
FIND-SET(x)
if x ≠ p[x]
   then p[x] ← FIND-SET(p[x])
return p[x]
```

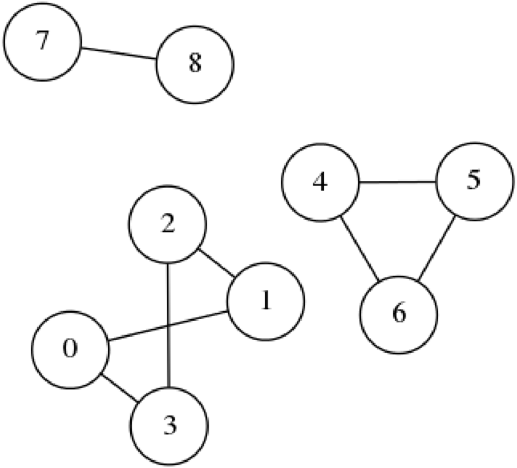
Implementation: Rooted Forests

Union by Rank and Path Compression

- Time complexity improves to $\Theta(m\alpha(n))$.
- $\alpha(n)$ is a very slow growing function.

Outline

Application: Connected Components



Application: Connected Components Algorithm

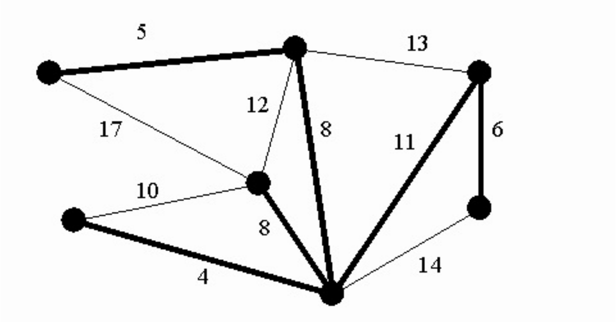
CONNECTED-COMPONENTS(G)

for each vertex $v \in V[G]$
    do MAKE-SET(v)
for each edge $(u, v) \in E[G]$
    do if FIND-SET(u) $\neq$ FIND-SET(v)
        then UNION(u, v)

SAME-COMPONENT(u, v)

if FIND-SET(u) = FIND-SET(v)
    then return TRUE
    else return FALSE

Application: Minimum Spanning Tree



Application: Minimum Spanning Tree Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Reference

Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein

Thank you

Questions?