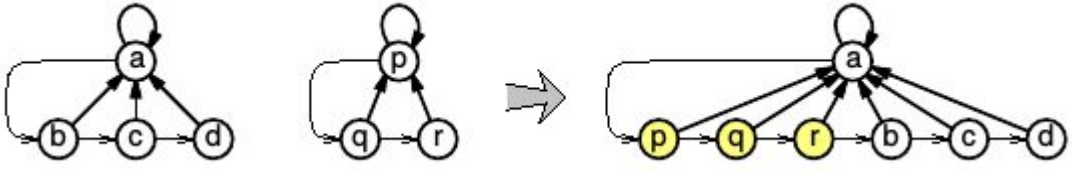
CSE525 Lec16

Disjoint-set

Debajyoti Bera (M21)

Shallow Reversed Trees + Threading




```
MakeSet(x):
    leader(x) ← x
    next(x) ← x
```

```
Find(x):
    return leader(x)
```

```
Union(x, y):
    x̄ ← Find(x)
    ȳ ← Find(y)

    y ← ȳ
    leader(y) ← x̄
    while (next(y) ≠ Null)
        y ← next(y)
        leader(y) ← x̄

    next(y) ← next(x̄)
    next(x̄) ← ȳ
```

Complexity of ...

- Makeset
- Find
- Union

Give a technique to **generate** a sequence of ops. on **n** elements so that Union(x) takes Θ(n) time on average.

Shallow Reversed Trees + Threading + Wtd Union

```
def NewUnion(x,y):
    x = leader(x)
    y = leader(y)
    if size(x) < size(y)
        Union(y,x)
        size(y) += size(x)
    else
        Union(x,y)
        size(x) += size(y)
```

During union, join the smaller tree to the root of the larger tree (not the other way).

Q: How to know "smaller" and "larger" ?

Re-implement and derive complexity of MakeSet, Find, Union.

O(1) O(1) O(log s)

If at any time a ∈ set with s elements, a's leader must have changed at most log(s) times

Lemma: The leader/root-pointer of any element changes at most log(n) times (during Union()). In fact, if all sets have at most s elements, leaders have changed at most log(s) times.

Consider a union that changed leader(a). a must belong to the smaller set. ∴ size of new set containing a ≥ 2 * size of old set containing a

Union(1st, 2nd) ... Union(1st, 2nd)

Lemma: t MakeSet & s WtdUnion takes O(t + s log s) time. (*Assume t > s for simpli.*)

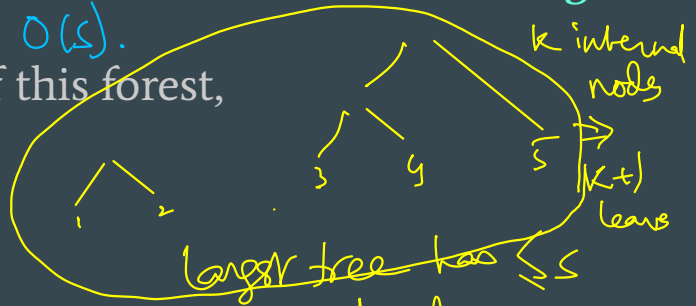
1st union 1, 2
2nd " (1,2)(3) 3rd +s

How to prove that the total time for s wtd. unions is O(s log s). Prove each union is O(log s).

Prove that largest set anytime during these calls is O(s).

Claim: Represent the unions as a binary forest. In any tree of this forest,

#internal nodes = ? union # leaves = ? elements



Claim: Internal nodes => Union calls, leaf nodes => sets.

Claim: Largest size of a set is ≤ s+1 = O(s)

Largest tree has ≤ s internal nodes ⇒ s+1 leaves.

(simpler analysis) Claim: s unions involve O(s) element. Thus, largest size after any union is O(s).

Total O(s) elements in non-singleton sets/involved in union. The leader pointer of any element in any of these sets changed at most O(log s) times. ∴ Total cost of all unions O(s log s).

Claim: Time for each union is O(log (s)).

Avg.

Reverse-trees + Union-by-depth :

Makeset Find Union

Shallow-trees + Union-by-Weight :

Makeset Find Union

Total cost of S unions

$O(1)$ pointer changes for each union $\rightarrow O(S)$

\+ cost of leader changes $= O(S) * O(\lg S)$ (see below)

↳ # leader changes of x_1 x_2 $x_3 \ldots$ of y_1 y_2 $y_3 \ldots$ of $z_1 \ldots$

- Size of the set of each of these elements at any point of time $= O(S)$

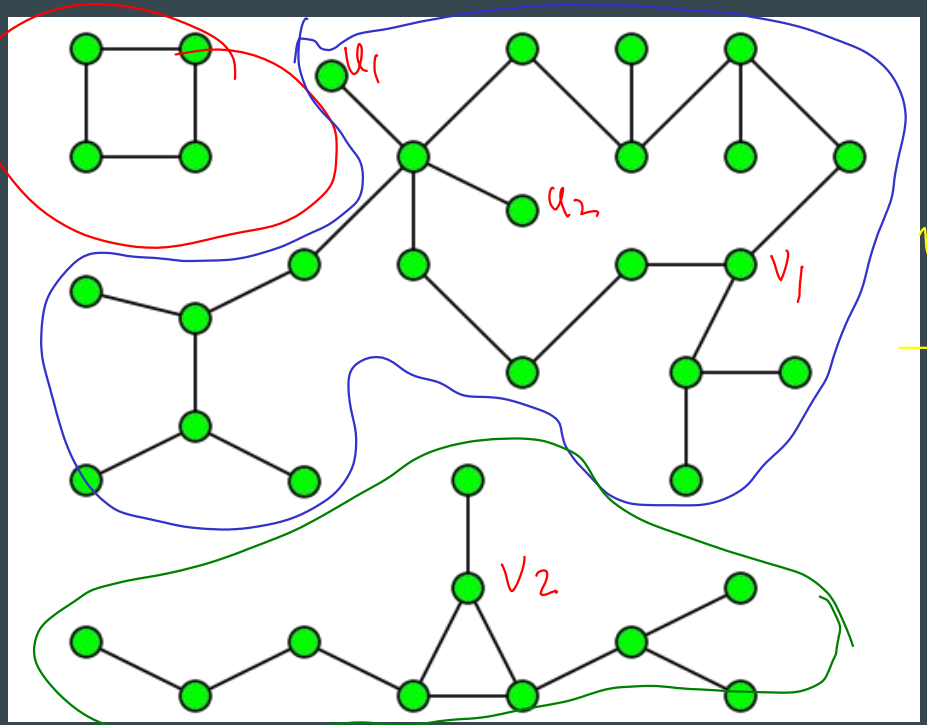
  $\Rightarrow$ # leader changes of each of these $= O(\lg S)$ (by earlier Lemma)

- Total # of elements involved in union-ed sets $O(S)$

Complexity using shallow tree + Threaded arr/dictionary
$O(V) + O(S) + O(E \lg E)$

Connected Component Labelling

Naive implementation: $\forall (u_i, v_i) \in S$
check if v_i is reachable from u_i
$q(V+E)$



Input : Undirected graph G = (V,E),
$S = \{(u_1,v_1), (u_2,v_2), \ldots (u_q,v_q)\}$

Output : For each (u_i,v_i), whether u_i, v_i are in same connected component?

Use $O(1)$ $O(1)$ $O(1)$ implementation of DS

def CCBatch(G,S):

$O(V)$ Makeset (v_i) $\forall v_i \in V$

$O(E)$ for every edge $(u,v) \in E$:
union(u,v)

$O(S)$ for every $(u_i, v_i) \in S$,
output find(u_i) = find(v_i)

Total :- $O(V+E+S)$ Complexity: # MakeSet, Find, Merge