Disjoint set

07 April 2025 16:23

↳ grouping n distinct elements.

{ - - - } { - - - - } → finding the set → with particular representative
→ uniting 2 sets

every set by one unique element from set. eg:→ min of set.

→ Operations on disjoint set
→ linked list representation
→ Rooted trees

disjoint set data structure → $S = \{S_1, S_2, S_3, \ldots S_k\}$ with representative

disjoint and dynamic sets
↳ changes

Representing each element of a set by an OBJECT, say X.

{a} {b} {c}
are objects.

operations → i) MakeSet(X) → initializes the set with X only
ii) UNION(x,y) → unites 2 sets S_x and S_y
iii) FIND-SET(X) → returns a pointer to the representative of unique set having X.

Time analysis on 2 parameters → n → no. of MAKE_SET operations
↳ m → no of MAKE-SET, UNION and FIND-SET operations total.

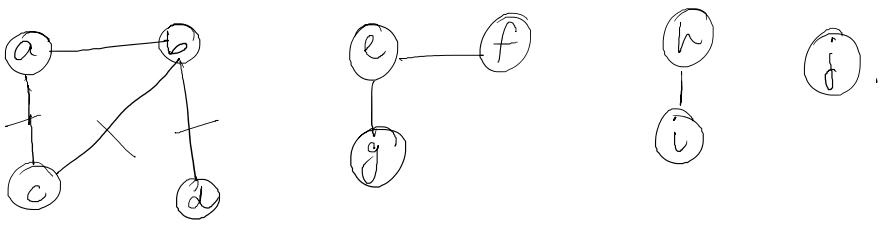
each UNION operation reduces set by 1; ⇒ (n-1) UNION → only one set remains for n diff sets.

(n-1) UNION → 1 set

∴ UNION operations → atmost (n-1)

$m \geq n$ { n → no. of make-set, assumed to be performed first}

CONNECTED COMPONENTS :→



Edge processed	collection of disjoint sets.									
initial	a	b	c	d	e	f	g	h	i	j
(b,d)	a	(b,d)	c		e	f	g	h	i	j
(a,c)	(a,c)	(b,d)			e	f	g	h	i	j
(c,b)	(a,c,b,d)				e	f	g	h	i	j
(a,b)	(a,c,b,d)				e	f	g	h	i	j
(e,g)	(a c b d)				e g	f		h	i	j
(e,f)	(a c b d)				e g f			h	i	j
(h,i)	(a c b d)				e f g			h i		j

4 connected components are

$\{a, b, c, d\}, \{e, f, g\}, \{i, h\}, \{j\}$

PSEUDO CODE

V → G.V

E → G.E

```
CONNECTED - COMPONENTS (G)
1.) for each vertex v ∈ G.V
2.)     MAKE_SET(v);
3.) for each edge e(u,v) ∈ G.E
4.)     if (FIND_SET(u) ≠ FIND_SET(v))
5.)         UNION (u,v)
```

SAME_COMPONENT (u,v) → check if same set or not.

```
1.) IF (FIND_SET (u) = FIND - SET(v))
```

2) return 1;

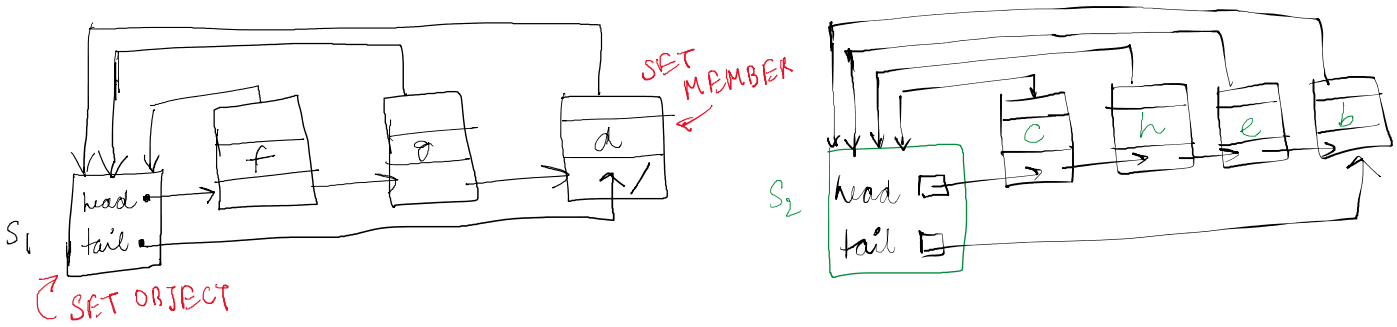
3.) return 0;

LINKED LIST representation.

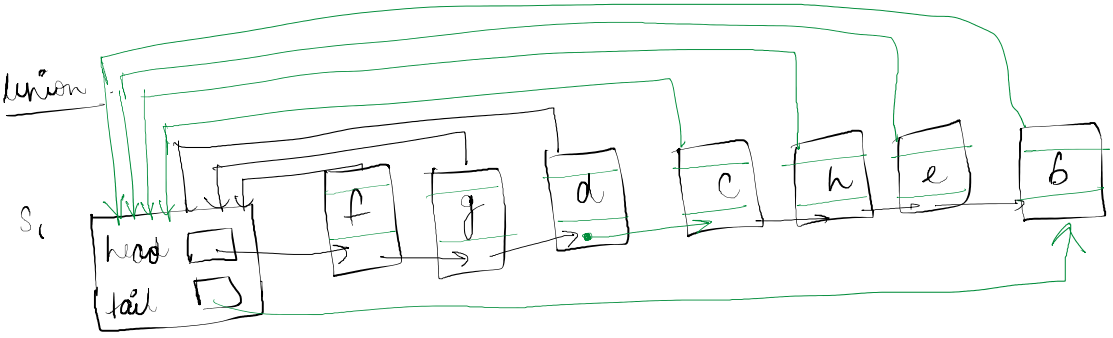
i) object for each set has head and tail pointers

Head → points to first object } set member
tail → points to last object.

a.)



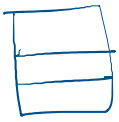
b.) Union



Changes in green.

MAKE_SET(X) } O(1) → make set just creates

FIND_SET(X) → find_set follows pointer to the objects and gives the object → head back



UNION(x, y) :> linear time → cardinality of y.

Suppose we make n objects then apply UNION.

operation	No. of objects updated
MAKE-SET(x_1)	1
MAKE-SET(x_1)	1
⋮	⋮
MAKE-SET(x_1)	1
UNION(x_2, x_1)	2
UNION(x_3, x_2)	3
UNION(x_4, x_3)	⋮
⋮	
UNION(x_n, x_{n-1})	$(n-1)$

n

$1+2+3+\ldots+(n-1) = \frac{n(n-1)}{2} = O(n^2)$

$\therefore$ For union operation : $\rightarrow \sum_{i=1}^{n-1} i = O(n^2)$

WEIGHTED-UNION HEURISTIC :- always append shorter to longer

May be appending longer to shorter.

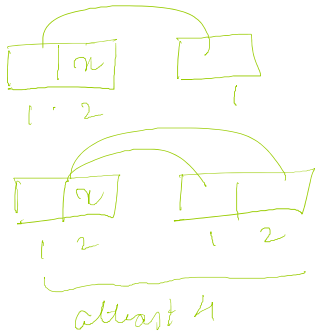
$m \rightarrow$ MAKESET, UNION, FIND_SET

and $n \rightarrow$ MAKESET.

$T.C = O(m + n \log n)$

* Each union 2 sets → almost $(n-1)$
* for each object → upper bound on T.C.

Say for object x. → at start when pointer update → atleast 2 elements
→ 2nd pointer update → atleast 4 elements.



$\therefore$

$1 \rightarrow 2$

$2 \rightarrow 4$

$3 \rightarrow 8$

$\vdots$

$k \rightarrow 2^k = n$

$\Rightarrow k = \lceil \lg n \rceil$

upper bound.
$\therefore$ ceiling value.

For $x \rightarrow \lceil \lg x \rceil$ pointer updates

$\therefore$ for n elements $n\lceil \lg n \rceil$

$\therefore$ UNION $\rightarrow O(n \log n)$ — (i)

updating tail pointer $= O(1)$ for $k = \log n$ times $\rightarrow$ each object.

each MAKE_SET and FIND_SET $\rightarrow O(1) \rightarrow$ each object.

$\therefore O(m)$ overall — (ii)

(i) and (ii)

$T.C = O(m + n \log n)$

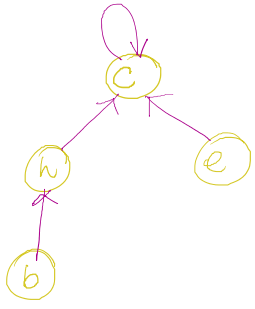
DISJOINT-SET FORESTS → asymptotically optimal → union by Rank
→ path compression.

each node → one member

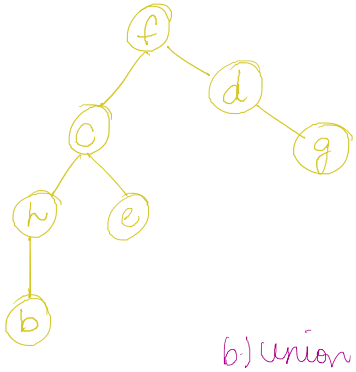
each tree → one set.

each Member → points to its own parent

Representative → Root.



a.) disjoint.



b.) Union

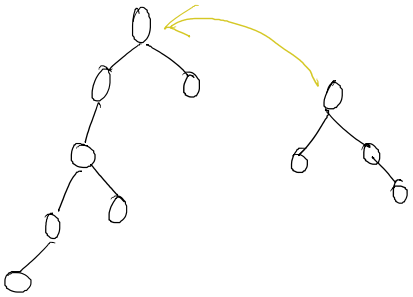
FIND_SET → follow up till the root.

Union by Rank

point of less with lower nodes point to root with more nodes

union by rank

Root of tree with fewer nodes point to root with more nodes

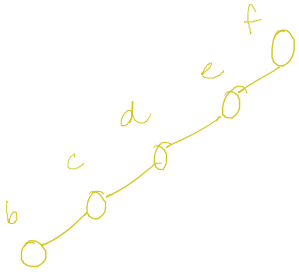


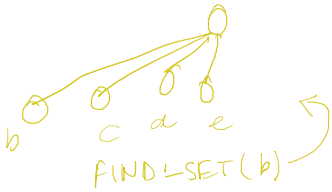
Maintain Rank along with Root.

PATH COMPRESSION

Try to point each node directly to the root

helps during FIND-SET





PSEUDOCODE :→ for union by Rank (on each node → x.rank (upper bound on height of x))

MAKE_SET → rank 0.

UNION → both equal heights (choose any and increment rank by 1).

→ one larger → no change (put small on big)

Parent → x.p.

MAKE_SET(x)

1.) $x.p = x$

2.) $x.rank = 0$.

UNION(x,y)

1.) LINK(FIND_SET(x), FIND-SET(y))

LINK(x,y)

1.) if $x.rank > y.rank$

2.) $y.p = x$

FIND_SET(x)

1.) if $(x \neq x.p)$

2.) $x.p$ = FIND-SET$(x.p)$;

2.) $y.p = x$

3.) else $x.p = y$

4.) if $(x.rank == y.rank)$

5.) $y.rank += 1;$

2.) $x.p = \ldots$

3.) return $x.p;$

Returns root, while fallback assigns all traversed node connection direct to Root.

Time complexity.

i.) union by Rank = $O(m \lg n)$

ii) By Path compression = $O(n + f \cdot (1 + \log_{2+f/n} n))$.

f → find set operations

n → make set operations

iii) Both → $O(m\,\alpha(n))$ where $\alpha(n)$ is a very slow growing function i.e Superlinear.

$\alpha(n)$ → Ackermann's function

for all practical purpose, $\alpha(n) \leq 4$

NAME OF OPERATION	WEIGHTED UNION	ROOTED TREE HEURISTICS (Union by rank)	ROOTED TREE HEURISTICS (Union by PATH COMPRESSION)
MAKE_SET (n_1=n)	O(1)	O(1)	O(1)
UNION (n_2=n-1)	O(lg(n)) (Amortized cost)	O(1)	O(1)
FIND_SET (n_3)	O(1)	O(lg(n))	$O(1 + \log_{(2+ n3/n)} n)$
(n_1+n_2+n_3)=m	O(m + n lg(n))	O(n + m lg(n)) if m > n, then O(m lg(n))	$O(n + n_3 (1+ \log_{(2+ n3/n)} n))$