Disjoint Set Data Structure

Union-find data structure

Kruskal's algo

ADT (Abstract Data Type)

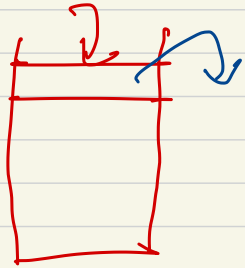
Implementation view

For designing your algo

→ You need certain operations

Operational view

Stacks / Queues



push

pop

top

isEmpty

→ arrays

→ Linked List

→ algo efficiently

As a user, you want to focus more on operational view

ADT : We agree on a set of operations

Algo using ADT
(w/o worrying about implementation view)

Disjoint Set/ Union Find as ADT

Setting These are n elements $x_1, \ldots x_n$
divided into disjoint sets

Initially each element is a group in itself.

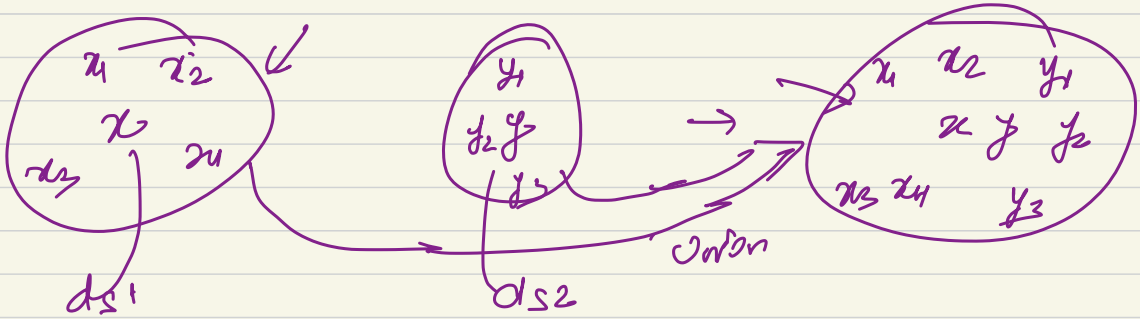
Operations ⟶

① MakeSet(x) :- Make a singleton set with x — i'd "x"

Disjoint Set ADT

② FindSet(x) :- Given x, find id of the set it belongs to
Find

③ Union(x,y) :- Create a single set out of two sets containing x & y
Merge
New set can have id of any of these sets

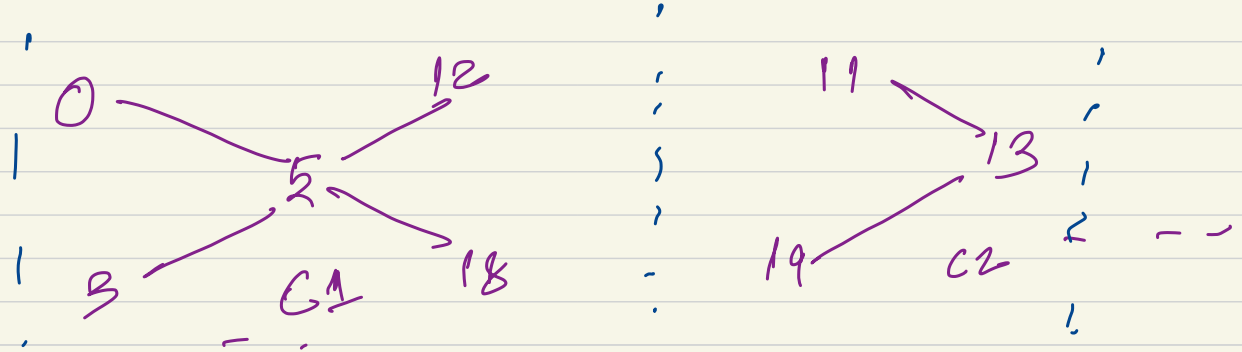


Ex:- Think of a social n/w

0, 1, _ _ n-1 V

Set of people

and a set of edges that define E



You're given → # of people 100

People 0, — , 99

5, 18

12, 5

0, 5

Task: Find # of connected components

For each person, print

Algo that'll use Disjoint Set ADT

- for each $x \in V$, do
  makeset(x)
- for each $(x,y) \in E$ do
  if find(x) $\neq$ find(y)
  Union(x,y)
- For each $x \in V$
  output - Person 'x' belongs to component 'find(x)'

Complexity of my algo:

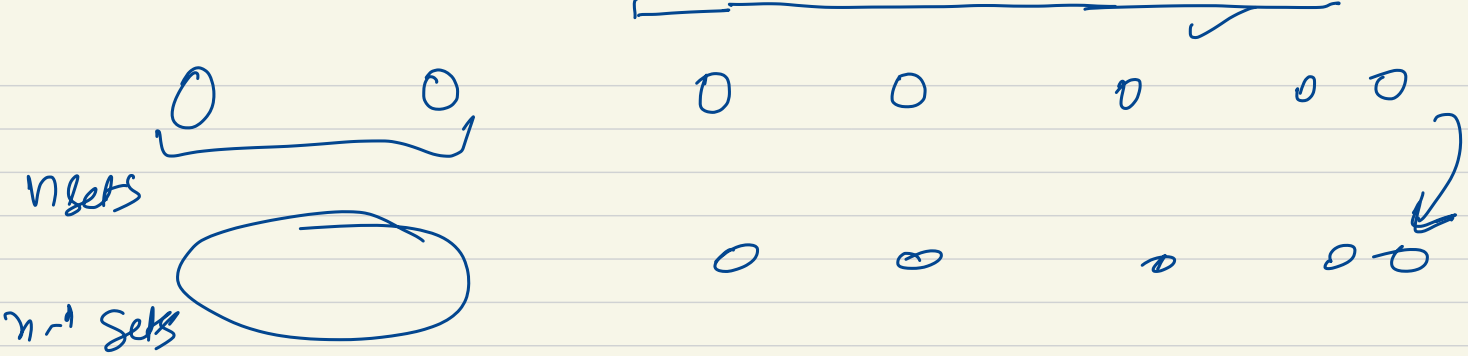
$O(n)$ makeset + $O(m)$ finds + $O(n)$ Unions $\times$ $O(n)$ finds

1, 3
:

the component it belongs to

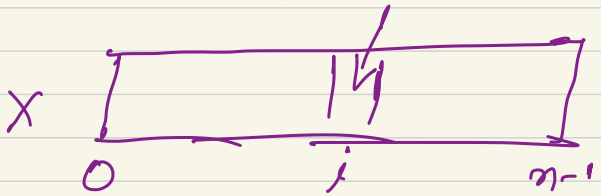
Additional Constraint :-

1. Edges can be given in any order
2. You can't store the edges, you can only scan these once



Implementation of Disjoint sets

→ Arrays (single array)

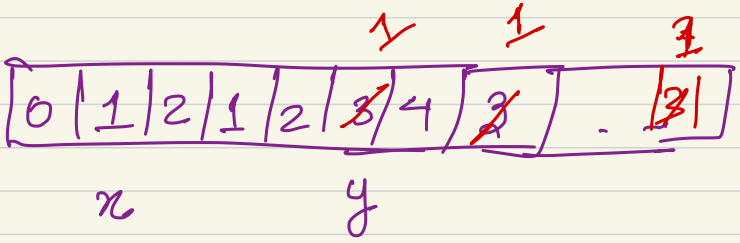


X[i]: id of the set that ith element belongs to

$O(n)$ makeset : $O(n)$

Find Set (x) : $O(1)$

Union (x, y)



A = Find Set (x) (1)

B = Find Set (y) (3)

if $A \neq B$

Union (A, B) $\Rightarrow$ change id of all elements belonging to set B to 1

$O(m)$

$O(n + m + n^2)$ = $O(m+n^2)$ = $O(n^2)$

Implementation using linked list

Array $O(m+n^2)$

Disjoint Set ADT

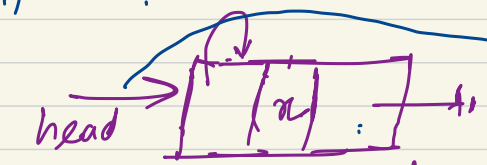
$O(m)$ makeset

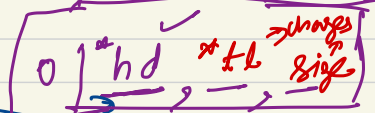
$O(m)$ find

$O(n)$ union

Each set will have a linked list

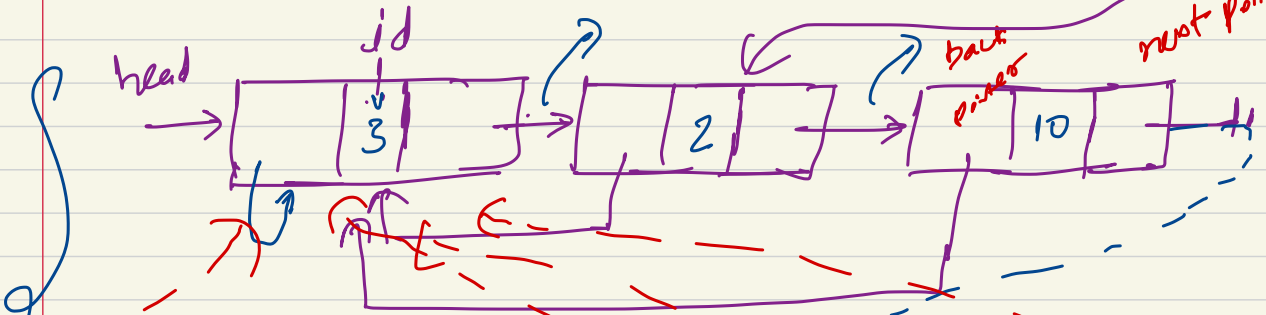
MakeSet $O(1)$



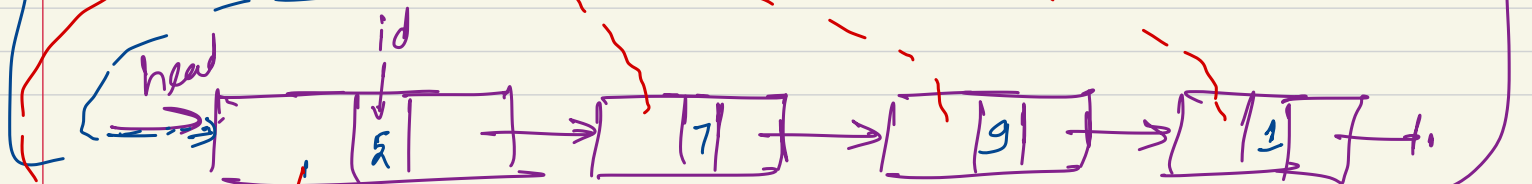


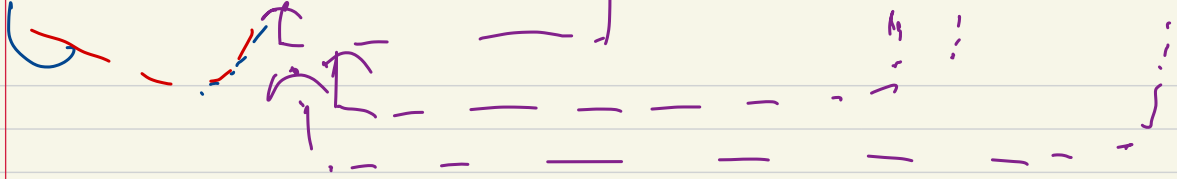
$O(n)$ linked lists

2 ll n elements



FindSet $O(1)$





Find Set(x) Union(x,y) O(n)

Put one linked list at the tail of another

A B

⇒ change back pointers for all the nodes in A

Complexity $O(m+n^2)$ No gain

1. Append smaller list at the tail of longer

Five to update back pointers for the smaller list.

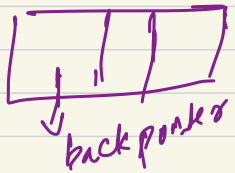
Does that help?

In the worst case, smaller list can be n/2 elements

O(n) in the worst case

Deeper look

Take a particular node



Amortized Analysis

First time you update back pointer → count the size of the resultant linked list ≥ 2

2nd _____ _____ ≥ 4

Avg. over all operations

→ $O(\log n)$ amortized complexity for back-pointer update

3rd

k^{th}

2^3

2^k

⇒ for each node, it can only update back pointers $O(\log n)$ time

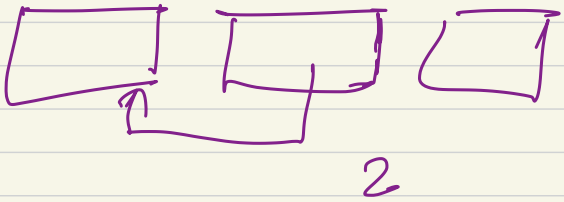
⇒ Total back pointer updates $O(n \log n)$

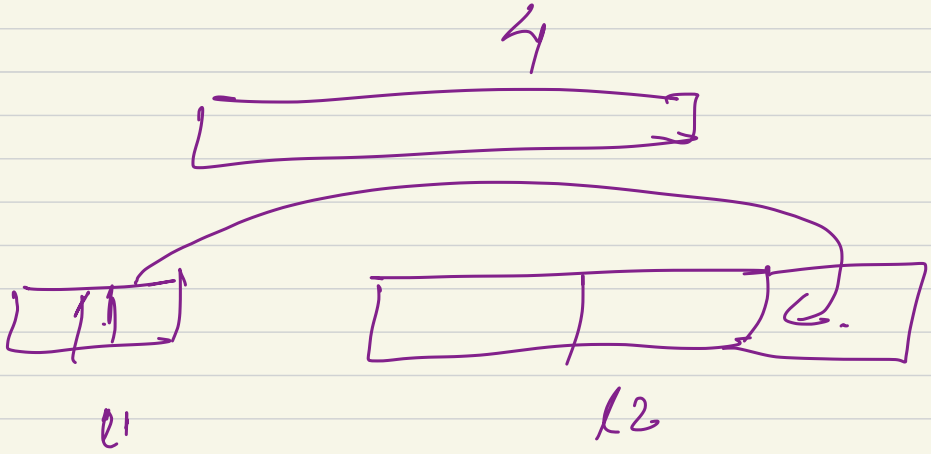
$O(n)$ unions $O(n \log n)$ time

Complexity with Linked List

$O(m + n \log n)$ $+ m(n)$

we're gaining over array


2


4
l1
l2

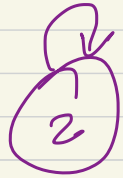
$O(m+n^2)$ $O(m+n\log n)$

Rooted Trees

- Not a binary tree
- Inverted tree

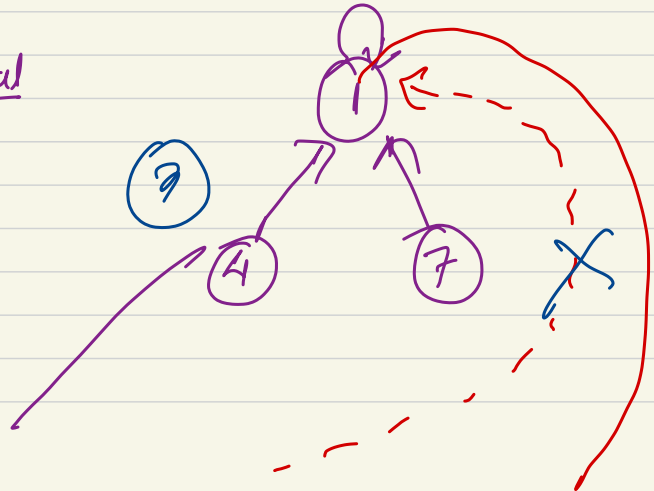
MakeSet

$O(1)$



id (value)
parent pointer

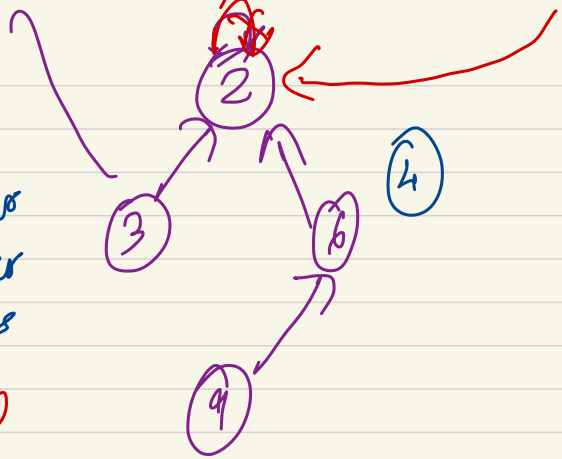
General



Id of set = Id of root

Find (4)

You follow parent pointers until you reach the root node



[parent points to itself]

Connect smaller trees to larger trees always

$h = O(\log n)$

Union $O(1)$

Parent points of the root of set A is set of the root of set B.

Complexity of find(x) → $O(h)$

$= O(n)$

a = find(x)
b = find(y)
if $a \neq b$

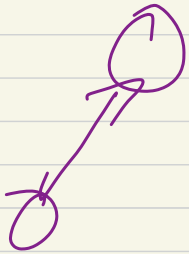
Union(a, b) → $O(1)$

$O(mn + n)$

If you follow height balanced, a tree of height h must have atleast 2^h nodes.

Prove by $n d^n$

Height 1



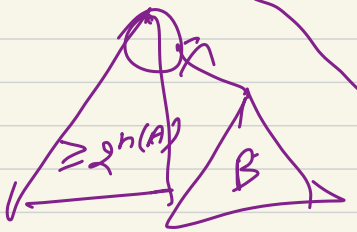
2 nodes

$h(A)$ $\geq 2^{h(A)}$

$h(B)$ $\geq 2^{h(B)}$

A is larger than or equal to B $\Rightarrow$ B becomes a subtree of A

$h = \max\{h(A), h(B)+1\}$



$n \geq 2^h$

$h \geq h(A)$ Trivial!

$2^{h(A)} + 2^{h(B)}$

$\geq 2^{h(B)} + 2^{h(B)}$

$= 2^{h(B)+1}$

$n = n(A) + n(B)$

$\geq 2n(B)$

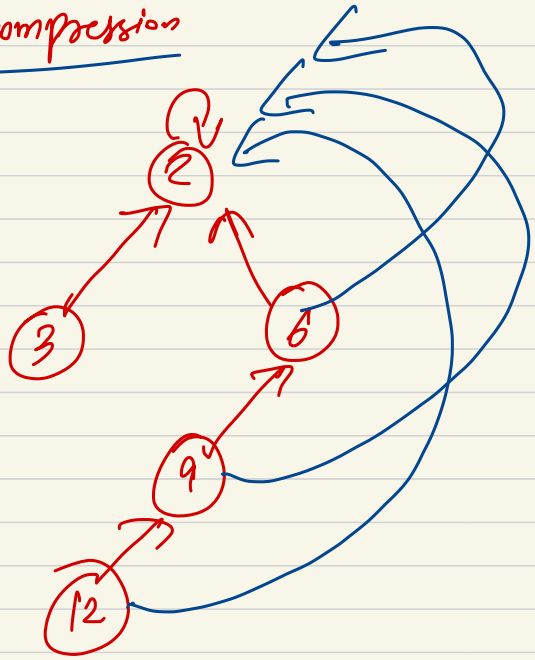
$\geq 2^{h(B)+1}$

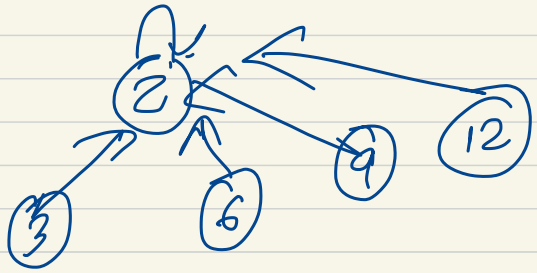
$\Rightarrow$ Find complexity $(\log n)$

Tree — $O(m \log n + n)$

Linked list — $O(m + n \log n)$

2. Path compression





Find(12)

Find root (2)

$O(m \log^* n)$ complexity for $O(m)$ finds

Base $\log^* 1 = \log^* 2 = 1$

$n \geq 2$ $\log^* n = 1 + \log^* (\lceil \log_2 n \rceil)$

$\log^* 4 = 1 + \log^* 2 = 2$

$\log^* 14 = 1 + \log^* 4 = 3$

$\vdots$

$\log^* 60000 = 4$

$\vdots$

$n \leq 2^{65536}$

$\log^* n \leq 5$

Constant

→ Complexity $\simeq O(n+m)$

$O(n)$ makeset

$+ O(m)$ find $+ O(n)$ Union

Rooted Trees are default ds for Disjoint Sets

$\log^* n$: number of times you have to take log of n before it reaches 1.

Intuitively

$14 \rightarrow 4 \rightarrow 2 \rightarrow 1$

$\log^* 14 = 3$