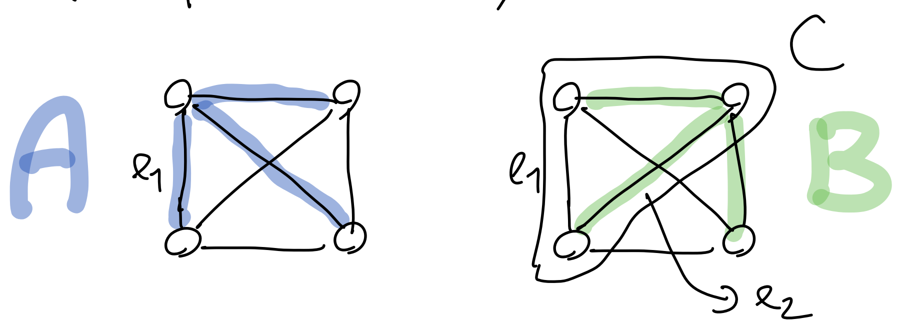
Exercise (uniqueness of MST):



$A \neq B \Rightarrow \exists$ an edge in one but not in the other; since weights are distinct $\exists!$ with min. weight $\rightarrow$ call it e_1; wlog $e_1 \in A$

the argument is an exchange argument:

- add e_1 to $B \Rightarrow$ this creates a cycle C; A is (M)ST $\Rightarrow$
  $\Rightarrow$ no cycles $\Rightarrow$ C has an edge $e_2 \notin A \Rightarrow \underline{w(e_1) < w(e_2)}$
- remove e_2 from B
  $\Rightarrow$ get a new spanning tree with weight $< w(B)$:
  contradiction, because B is MST!

---

frequent operation: cycle check

(new) data structure:

<u>Union-Find</u> (aka disjoint-set) 1964

is a data structure to manage <u>disjoint sets</u> of objects operations : init : given an array X of objects, create a union-find data structure with each object $x \in X$ in its own set

find : given an object x, return the name of the set that contains x

union : given two objects x, y merge the sets that contain x and y into a single set
(if x, y are already in the same set then do nothing)

Can be implemented with these complexities:
- init $O(n)$
- find $O(\log n)$
- union $O(\log n)$

$n =$ ~~#~~ of objects in the data structure

Fast Kruskal's implementation with Union-Find

Idea: Union-Find keeps track of the connected components of the current solution; $A \cup \{(v, w)\}$ creates a cycles $\iff$ v, w are already in the same connected component

Kruskal (G)

$A = \emptyset$

$U = \text{init}(V)$ // Union-Find data structure

sort edges of E by weight

for each edge $e = (v, w)$ in nondecreasing order of weight do

if $\text{Find}(v) \neq \text{Find}(w)$ then

// no v-w path in A, so OK to add e

$A = A \cup \{(v, w)\}$

// update due to component union

Union (v, w)

return A

Complexity:

init	$O(n)$
sorting	$O(m \log n)$
$2m$ Find	$O(m \log n)$
$n-1$ Union	$O(n \log n)$
A update	$O(n)$
	$O(m \log n)$

Implementation of Union-Find

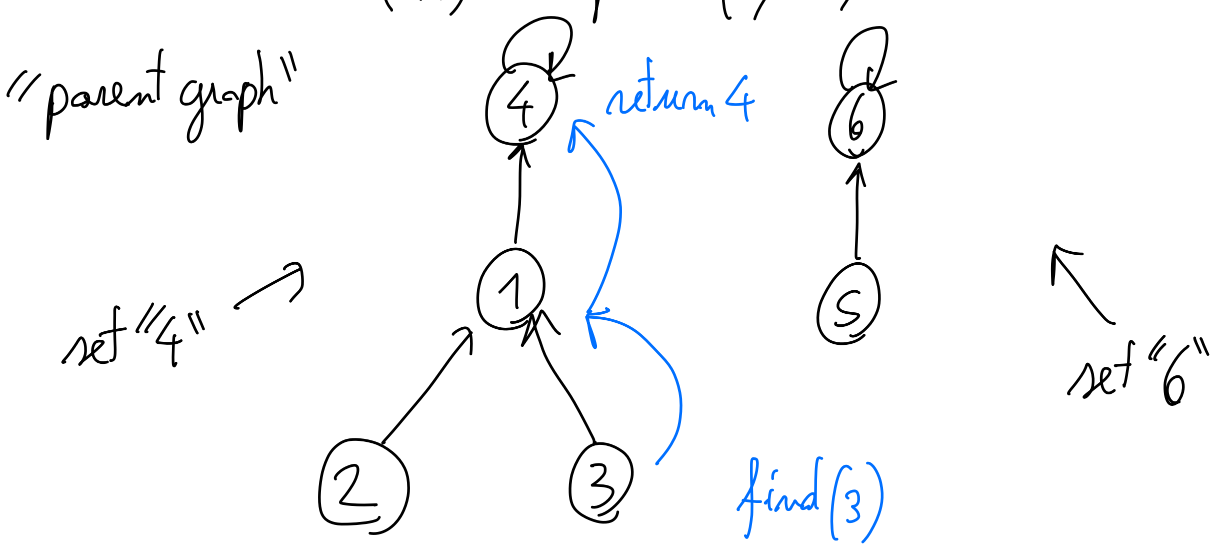
We'll use an <u>array</u>, which can be visualized as a set of <u>directed trees</u>. Each element of the array has a field parent(x) that contains the index of the array of some object y.

Example:

index of x	parent(x)
1	4
2	1
3	1
4	4
5	6
6	6

vertices: (indexes of) objects

arc $(x,y) \iff \text{parent}(x) = y$



set of objects $\longleftrightarrow$ set of directed trees in a parent graph

name of the set = root

Init: 

Find: parent $\to$ parent $\to$ ... $\to$ root

find(x): 1) starting from x's position in the array, traverse parent edges until reaching a position j s.t. parent$(j) = j$

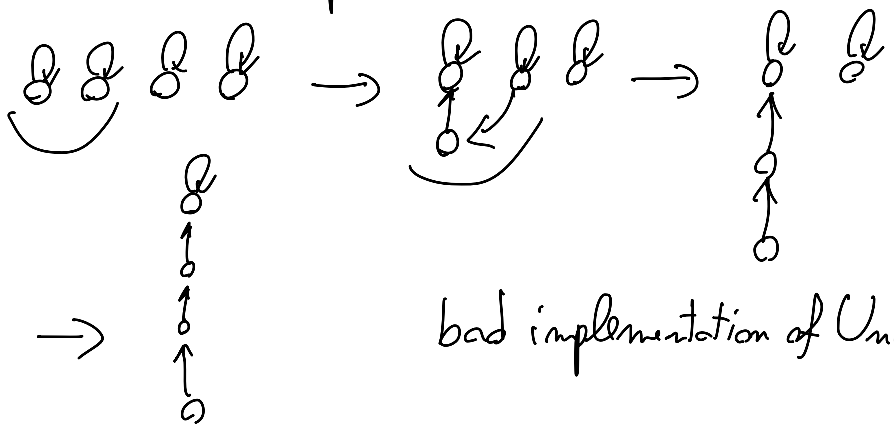
2) return j

Def.: the depth of an object x is the number of edges traversed by Find(x)

example: depth$(4) = 0$
depth$(1) = 1$
depth$(2) = 2$

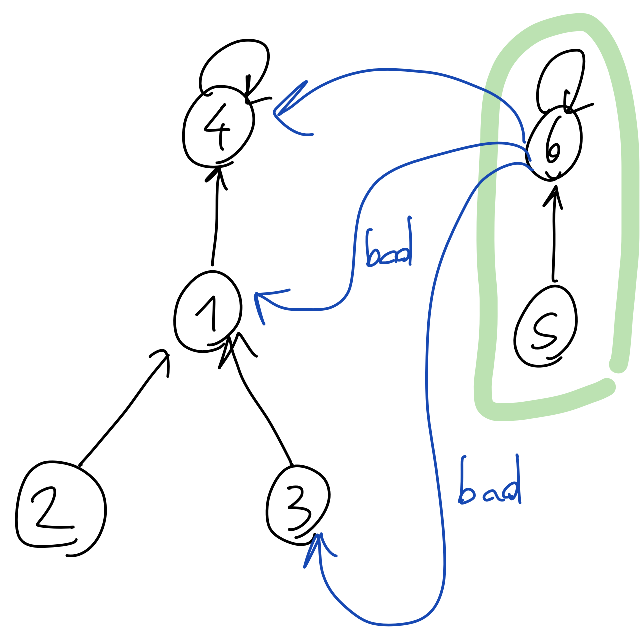
Complexity of Find(x): largest depth $O(n)$

it depends on Union's implementation:

example: 

bad implementation of Union

Union : Union(x, y) $\longrightarrow$ the 2 trees of the parent graph containing x and y must be merged in a single tree. The simplest way is to point one of the 2 root to another node of the other tree



We need to decide:

1) Which of the 2 roots remains a root
2) To which node should a root point

2) a root must point to the other root
(in order to have the minimum increase in depth)

1) idea: minimize the # of objects whose depth increases; "union-by-size"
(alternative idea: the root of the less tall tree points to the tallest tree: "union-by-rank")

Union(x, y): 1) invoke Find(x) and Find(y) to obtain the names i and j of the sets that contain x and y
if $i = j$ return

2) if size$(i) \geq$ size(j) then
parent$(j) = i$
size(i) = size(i) + size(j)
else
parent$(i) = j$
size(j) = size(i) + size(j)

$\Rightarrow$ Complexity di Find(x) (and of Union(x,y)) is $O(\log n)$

Why? Initially, depth$(x) = 0 \quad \forall x$

Depth(x) can only increase because of a Union in which the root of the tree of x points to another root. This happens only when the tree of x gets merged to a tree of size not smaller

$\Rightarrow$ when the depth of x increases, the size of the tree of x at least doubles.

How many times can this happen? $\leq \log_2 n$ times, therefore the depth of x cannot increase more than $\log_2 n$ times.

We have 2 $\neq$ alg. with complexity $O(m \log n)$

$O(m)$? open problem!