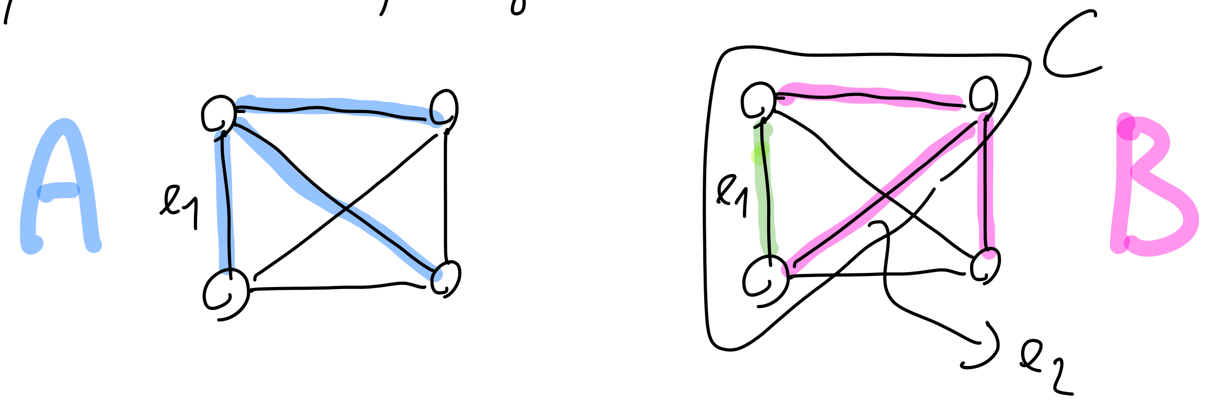
Exercise (Uniqueness of MST):

By contradiction, imagine we have two $\neq$ MSTs:



$A \neq B \implies \exists \geq 1$ edges in one but not in the other; since weights are all distinct $\exists!$ edge, among those, with min. weight. Call it e_1; wlog assume $e_1 \in A$

Now, a cut & paste argument:

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

Exercises:

1) Is the converse true?
2) Show that the <u>second-best MST</u>, that is, the spanning tree of second-smallest total weight, is not necessarily unique when weights are all distinct

---

frequent operation in Kruskal's alg: cycle check (equiv., path check)

What's a data structure to do that quickly?

<u>Union-Find</u> (a.k.a. disjoint-set) 1964

is a data structure to manage <u>disjoint sets</u> of objects

Operations supported: <u>Init</u>: given an array X of objects, create a union-find data str. with each object $x \in X$ in its own set

<u>Find</u>: given an object x, return the name of the set that contains x

<u>Union</u>: 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 = n° of objects in the data structure

Fast Kruskal's implementation with Union-Find

Idea: U-F keeps track of the connected components of the nt solution; $A \cup \{(v, w)\}$ creates a cycle $\iff$ v, w are already in the same connected component

Kruskal(G)

$A = \emptyset$

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

sort edges of E by weight

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

if Find(v) $\neq$ 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 updates: $O(n)$

Total: $O(m \log n)$

Implementation of Union-Find

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

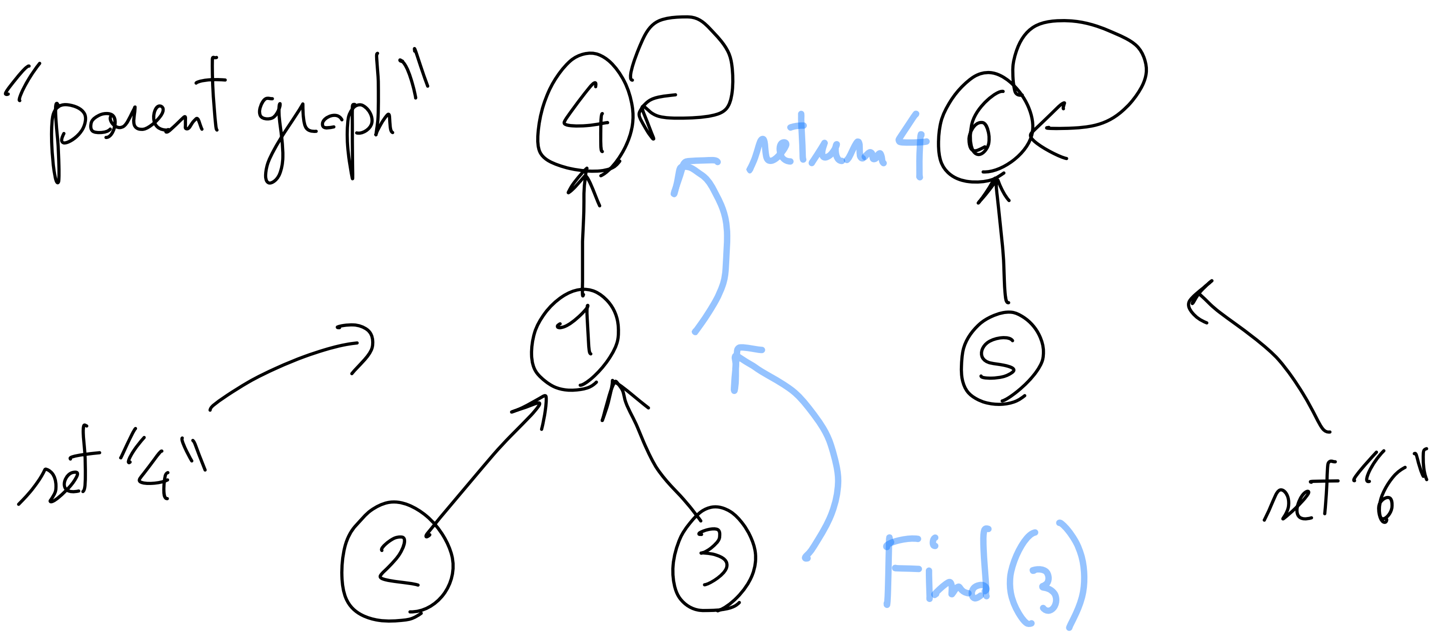
Example :

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

vertices : (indexes of) objects

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

"parent graph"



sets of objects $\leftrightarrow$ set of directed trees in a parent graph

name of the set = root

<u>Init</u> :

<u>Find</u> : parent $\to$ parent $\to$ .... $\to$ root

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

2) return j

Definition : the <u>depth</u> of an object x is the number of arcs traversed by Find(x)

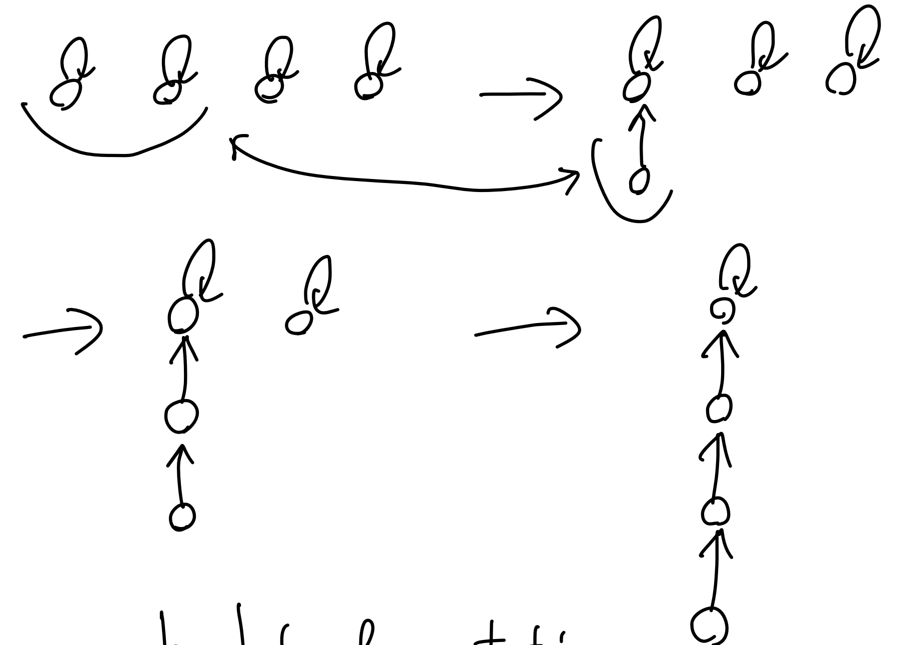
depth (4) = 0

depth (1) = 1

depth (2) = 2

Complexity of Find(x): $O(n)$ for sure, but it depends on Union's implementation:

example:

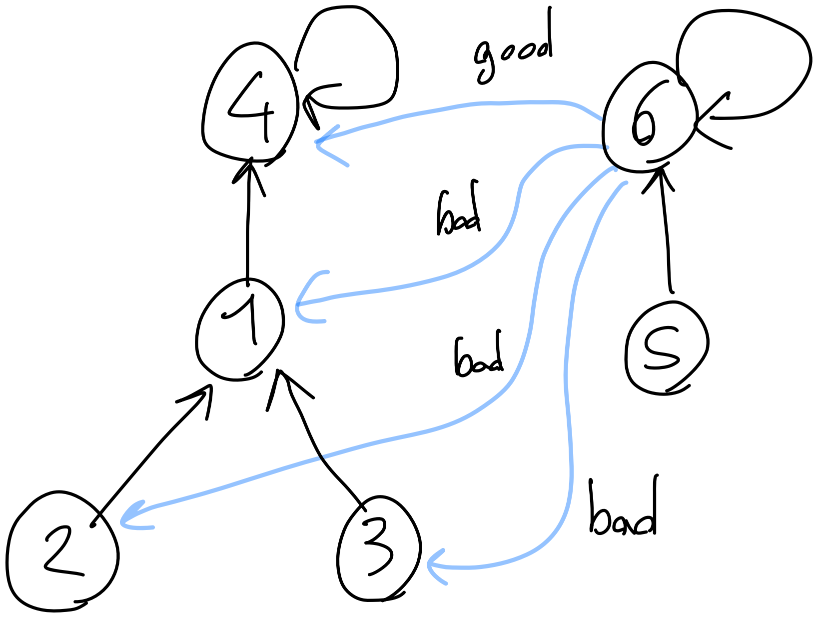


a bad implementation

Union: Union(x, y) $\rightarrow$ 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 roots 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 (to have the minimum increase in depth)

1) idea: minimize the n° 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

new field to be added

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

Exercise: show it