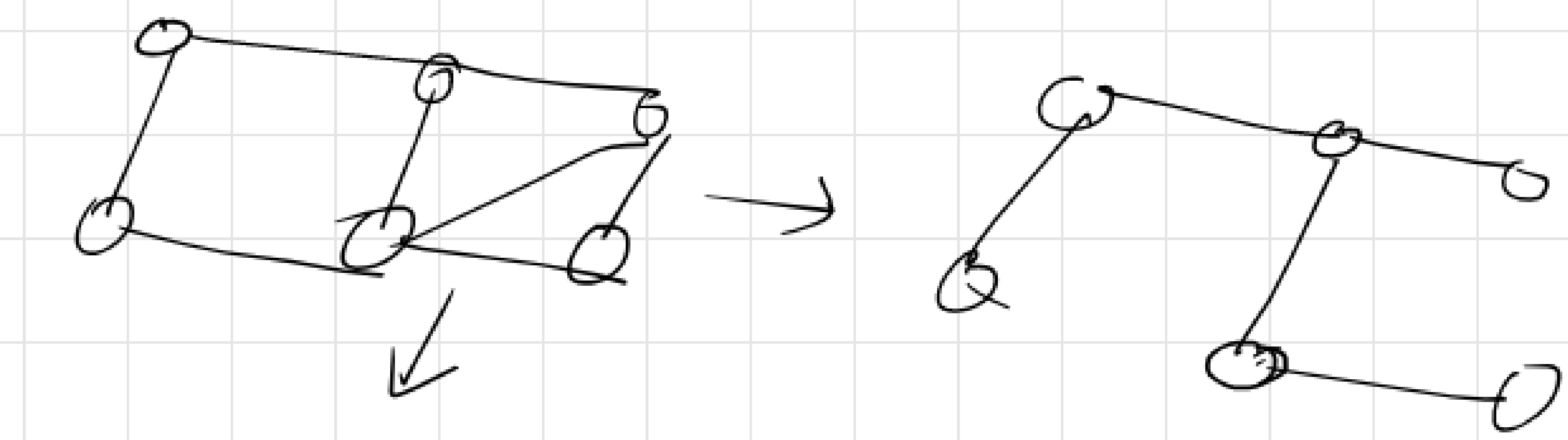
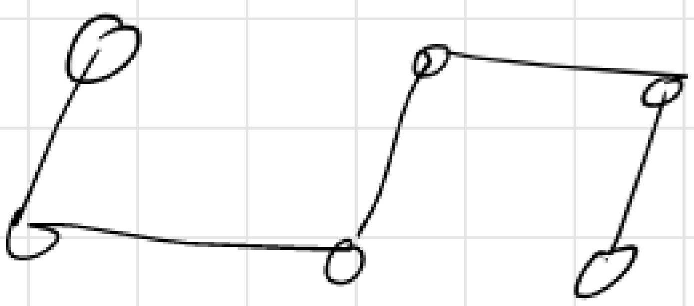
Minimum Spanning Tree

Undirected

Spanning Tree

(Tree + all vertices)





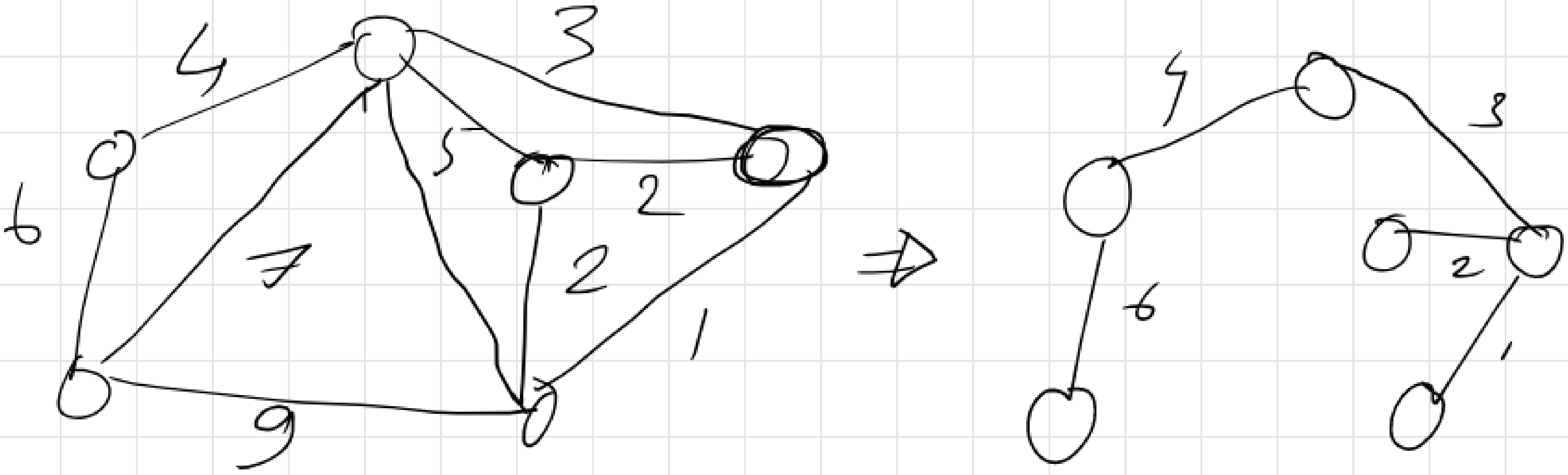
(there can be multiple spanning tree)

\# edges in spanning tree $= n-1$

Minimum Spanning Tree

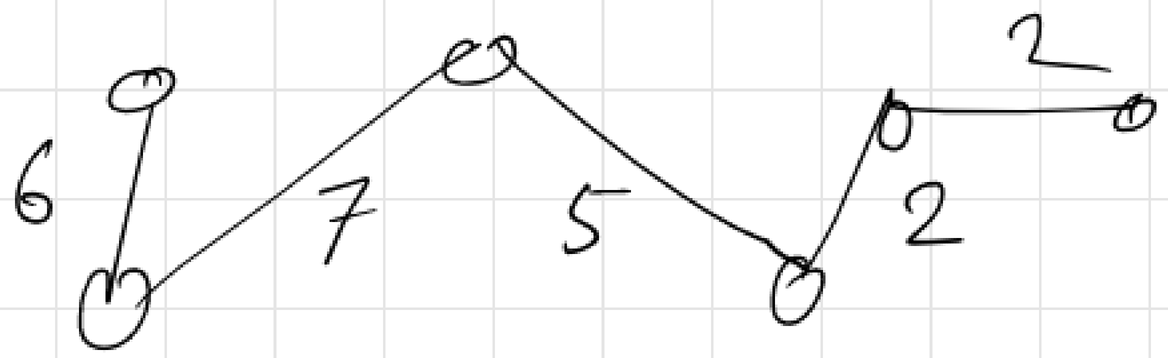
A spanning tree of minimum weight (length)

$G = (V, E) \qquad W: E \longrightarrow \mathbb{R}^+$ (non-neg)



$= 16$

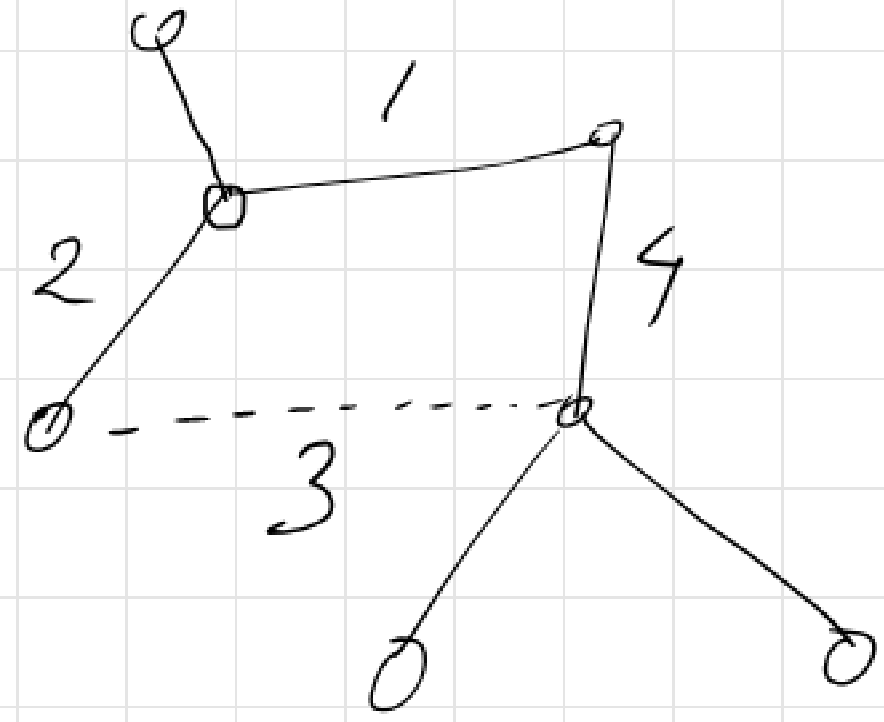
(weight) Length of spanning tree = Sum of the lengths of the edges in the tree



$= 22$

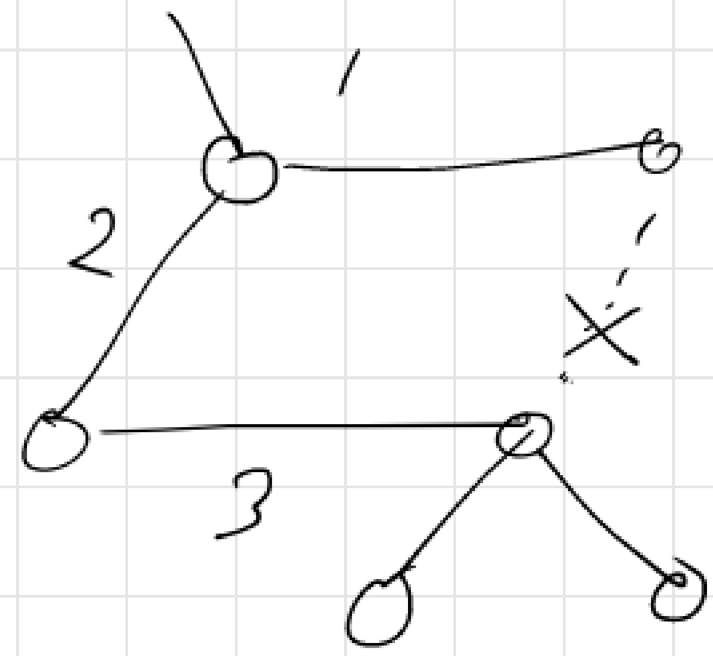
Obs 1: Adding an edge to a spanning tree will form a cycle. So, it will no more be a tree anymore

Obs 2



(Can I add edge with weight 3 and remove edge with weight 4 from the cycle)

⇓



[Still be tree?? why??]

[Construct a spanning tree (not necessarily optimal), add small edge weight to form cycle, delete edge with max weight of a cycle; and repeat]

[Will this give MST??]

Yes,

but with high time complexity

Properties of MST

Property 1

If an edge (i, j) is part of MST T of G, its two end vertices are part of an IJ-cut and (i, j) is the minimum weight edge in the IJ-cut set.

Property 2 If a graph doesn't have unique edge weights, there could be more than one MST for the graph

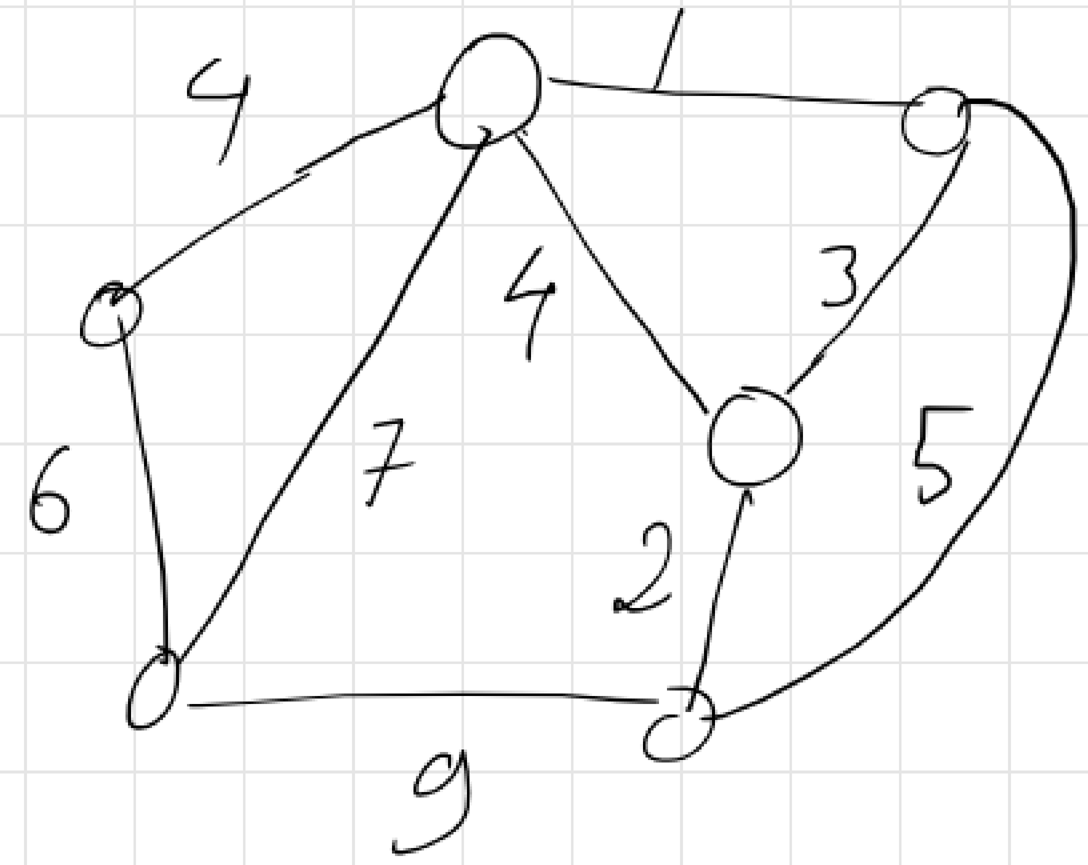
Property 3 If all edges in a graph, have unique weights, then there can be only one MST of G.

Property 4

In a cycle, heavy weight edge

won't be there

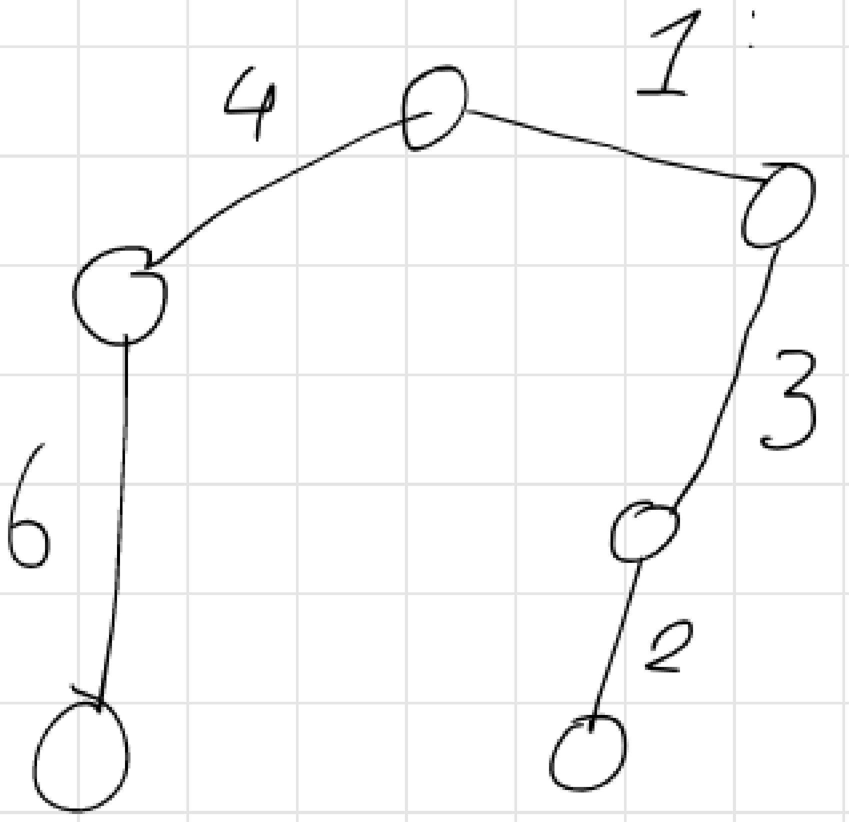
"Kruskal's Algorithm"



greedy Algo

→ make the best choice available

→ pick edge with smallest length



$= 16$ ["Is this the best possible?"]

Kruskal's algorithm

1. Sort edges in increasing order of length
   $(e_1, e_2, e_3 \ldots e_m)$ $[l(e_i) \le l(e_j)]$
   ↳ $O(m \log m)$
2. $T \leftarrow \emptyset$
3. for $i \leftarrow 1$ to m → [How to check]
4. if $\{e_i\} \cup T$ is a tree
5. $T \leftarrow T \cup \{e_i\}$
6. return (T)

Correctness Assume all edge lengths are distinct

let edges picked by Kruskal Algo

$$g_1 < g_2 < g_3 \quad [g_i] \cdots < g_{n-1}$$

Some OPT MST: $f_1 < f_2 < f_3 \quad \boxed{f_i} \cdots < f_{n-1}$

Suppose these sets differ and at the first place they differ is i

$$g_1 = f_1, \; g_2 = f_2, \; \cdots \; g_{i-1} = f_{i-1}$$

$$[g_i \neq f_i]$$

Case 1 $l(g_i) < l(f_i)$

g_i cannot be any one of $f_{i+1} \cdots f_{n-1}$

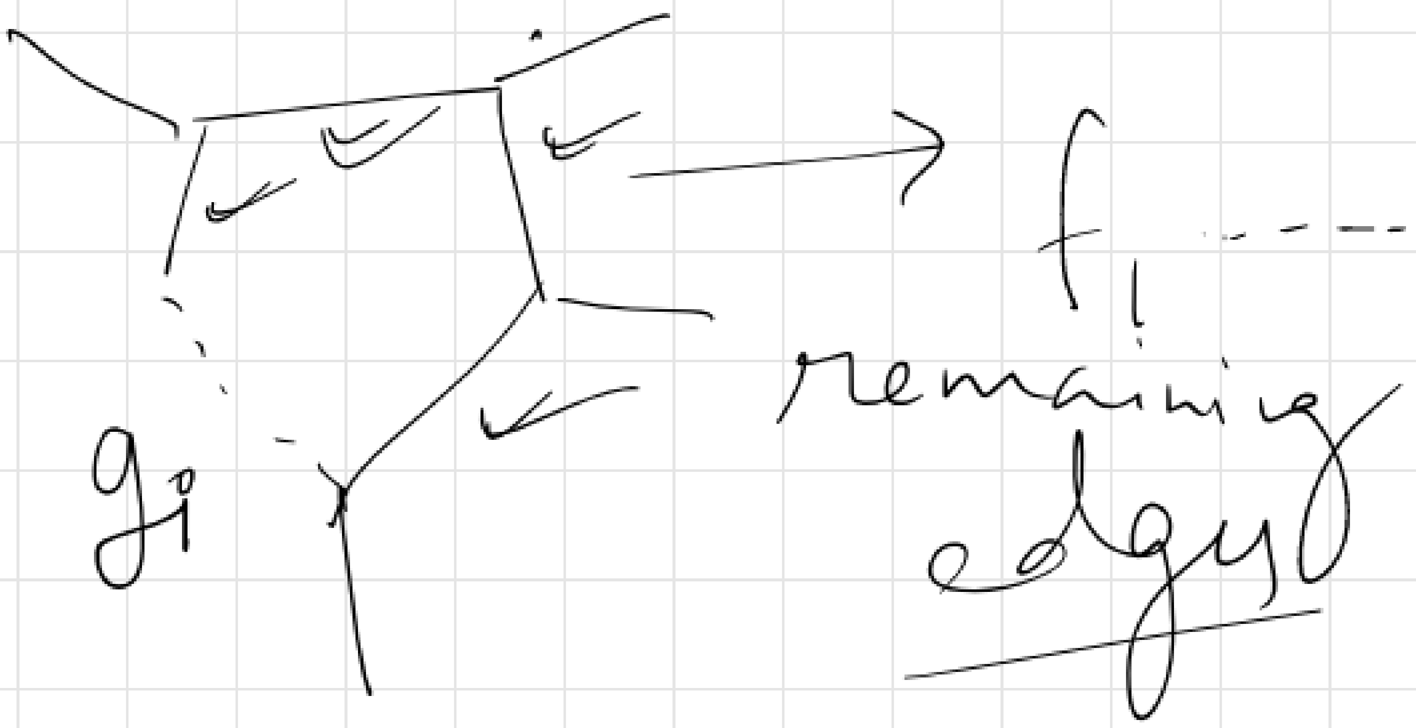
g_i cannot be $f_1, f_2 \cdots f_{i-1}$

g_i is not there in OPTimal MST

Add g_i to the OPT tree,

$\Rightarrow$ form a cycle in a tree

Can g_i be the longest edge on the cycle C??



[g_i will not be the longest edge on the cycle]

[so remove the longest edge from the cycle C, still will be a tree] or [We get new tree with less weight] $\Rightarrow \Leftarrow$

Case 2 $\quad l(f_i) < l(g_i)$

$f_1 = g_1 \quad f_2 = g_2 \qquad f_{i-1} = g_{i-1}$

f_i is distinct from $g_1 \cdots g_{i-1}$

Why did Kruskal not pick f_i ?

It might form a cycle

$f_i \cup \{g_1 \cdots g_{i-1}\}$ contains a cycle

$= f_i \cup \{f_1 \cdots f_{i-1}\}$ which mean in optimal there is a cycle. (which is not possible)

Both cases are not possible

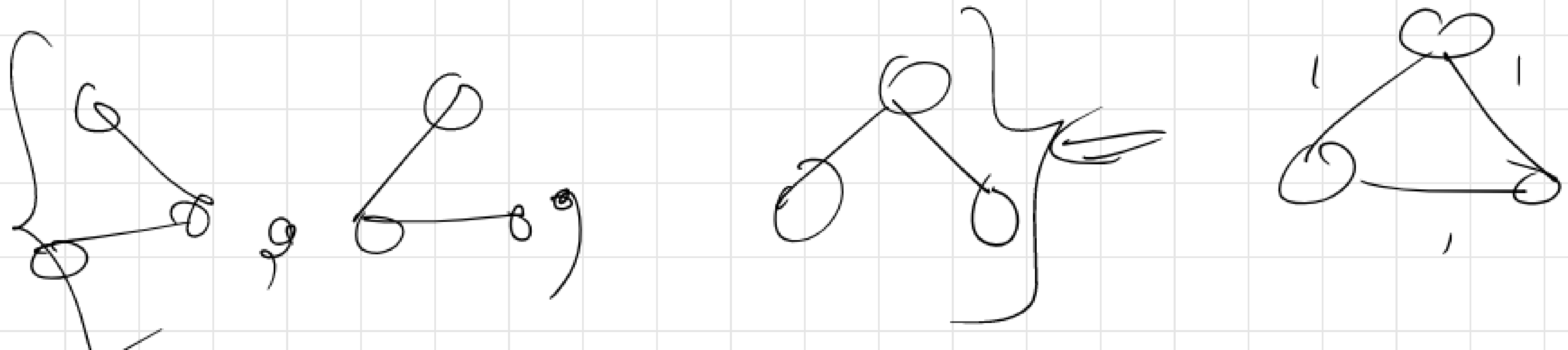
Thus $g_i = f_i$ $\forall i = 1$ to m

(Kruskal Algo give one of the optimal Solutions)

Is the MST unique??

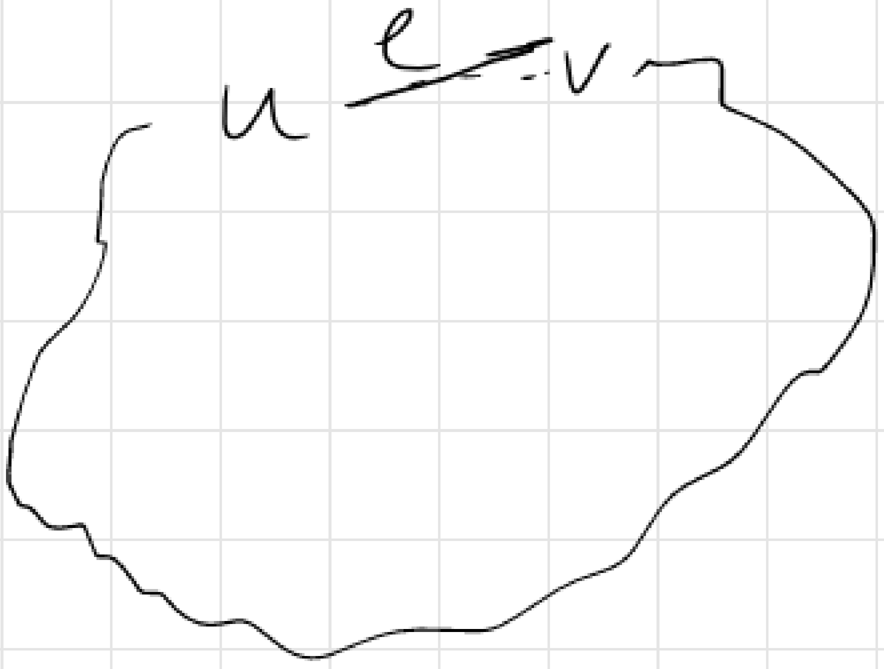
→ Yes, if edge lengths are unique

→ no, otherwise



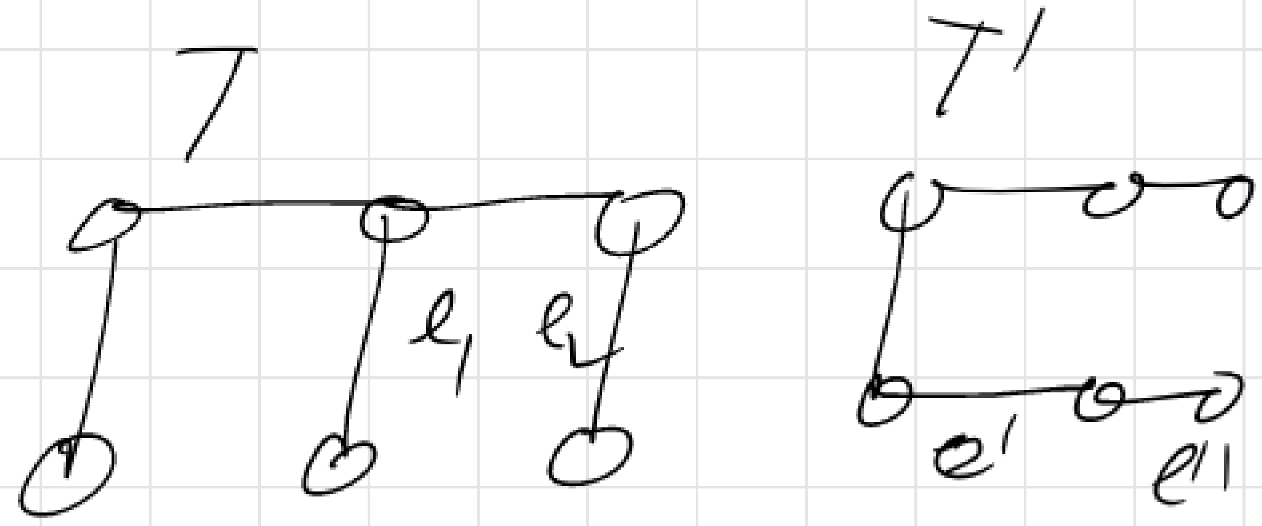
How do we check if a cycle is formed when an edge $e=(u,v)$ is included?

→ Cycle is formed iff u & v are already connected.



⇒ Maintain all connected components.

Correctness



$T \rightarrow$ by Kruskal

$T' \rightarrow$ OPT

There should be at least one edge difference between T and T'

W.L.O.G assume e_1 is in T, but not in T'.

add e_1 to T' $\Rightarrow$ $T' \cup \{e_1\}$

create a cycle C.

There must be an edge e' in C which is not in T.

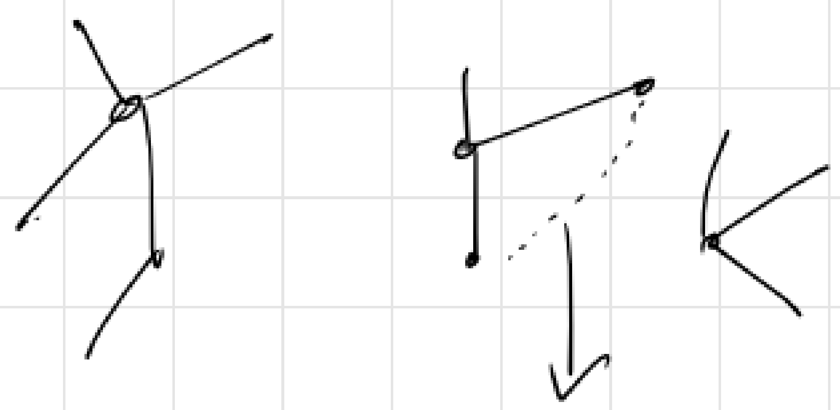
$$w(e') \geq w(e_1)$$

In T': remove e', add e_1.

$$w(\overset{new}{T'}) = w(\overset{old}{T'})$$

do it for all edges not in T' but in T. #

"Union - find data Structure"



forms Cycle

maintain a collection of components

Initially n connected components,
In the end there is only one component.

$V = \{v_1, v_2 \cdots v_n\}$

Initially n-sets $\{v_1\}$ $\{v_2\}$ $\cdots$ $\{v_n\}$
{disjoint n sets}

Operation on sets

Union & Find

Find: to know in which set an element belongs to

Find (x) → returns the set in which x lies

Union(Find(x), Find(y))

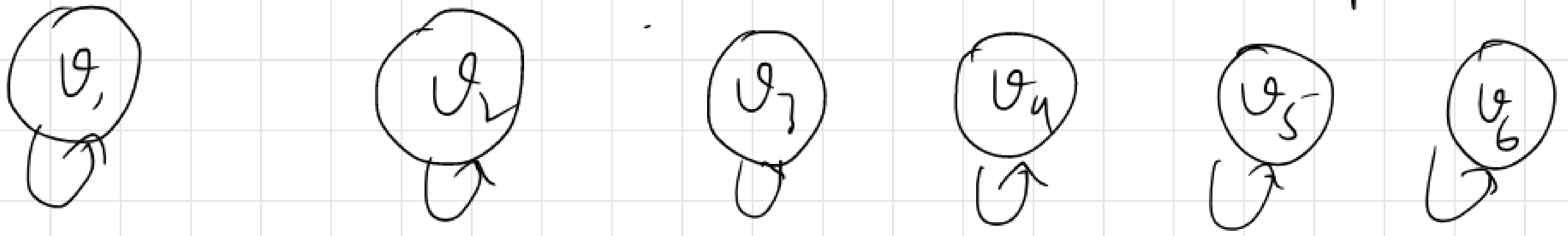
1. Sort the edges increasing order of weight
2. $T \leftarrow \emptyset$; Initialize singleton sets
3. For $i = 1$ to m
   let $e_i = (u, v)$
   if find $(u) \neq$ find (v)
   $T \leftarrow T \cup \{e_i\}$ Union(Find(u), Find(v))

$$\left.\begin{array}{l}\#\text{unions} = n-1 \\ \#\text{find} \le 2m\end{array}\right\} \quad \begin{array}{l} X \\ Y \end{array}$$

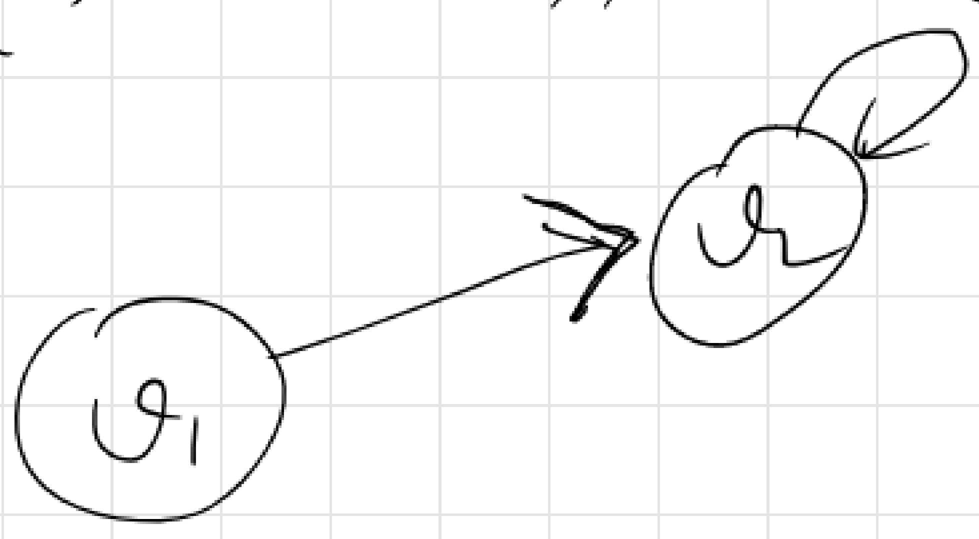
$$\boxed{\begin{array}{l}\underline{\text{Running time}} \\ m\log m + nX + mY\end{array}}$$

$U = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

Each set has an one pointer



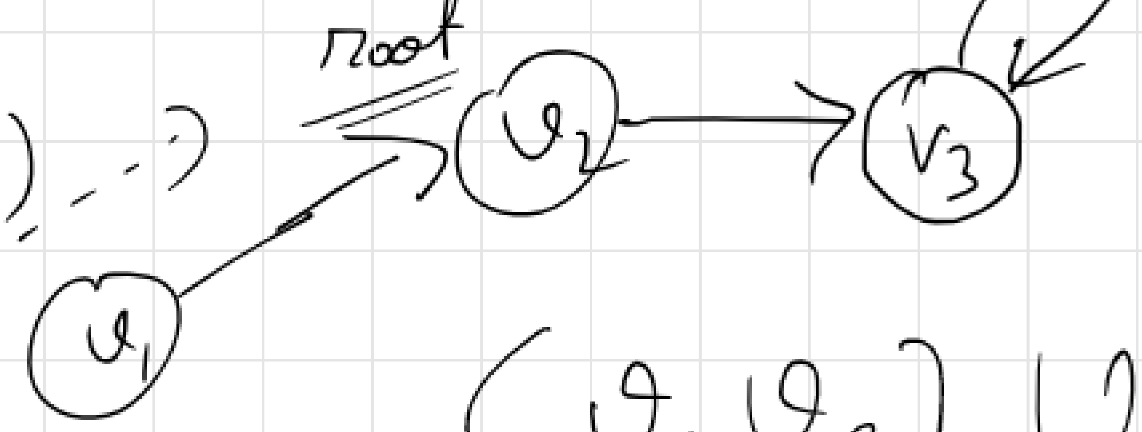
Union(find(v_1), find(v_2))



Each set represatd by the root

Union(find(v_1), find(v_3))

find(v_1) ...

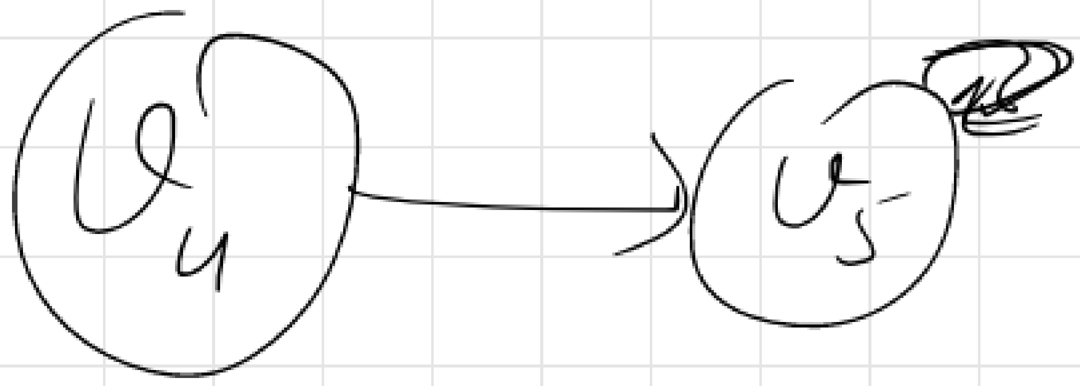


$\{v_1, v_2\} \cup \{v_3\} = \{v_1\ v_2\ v_3\}$

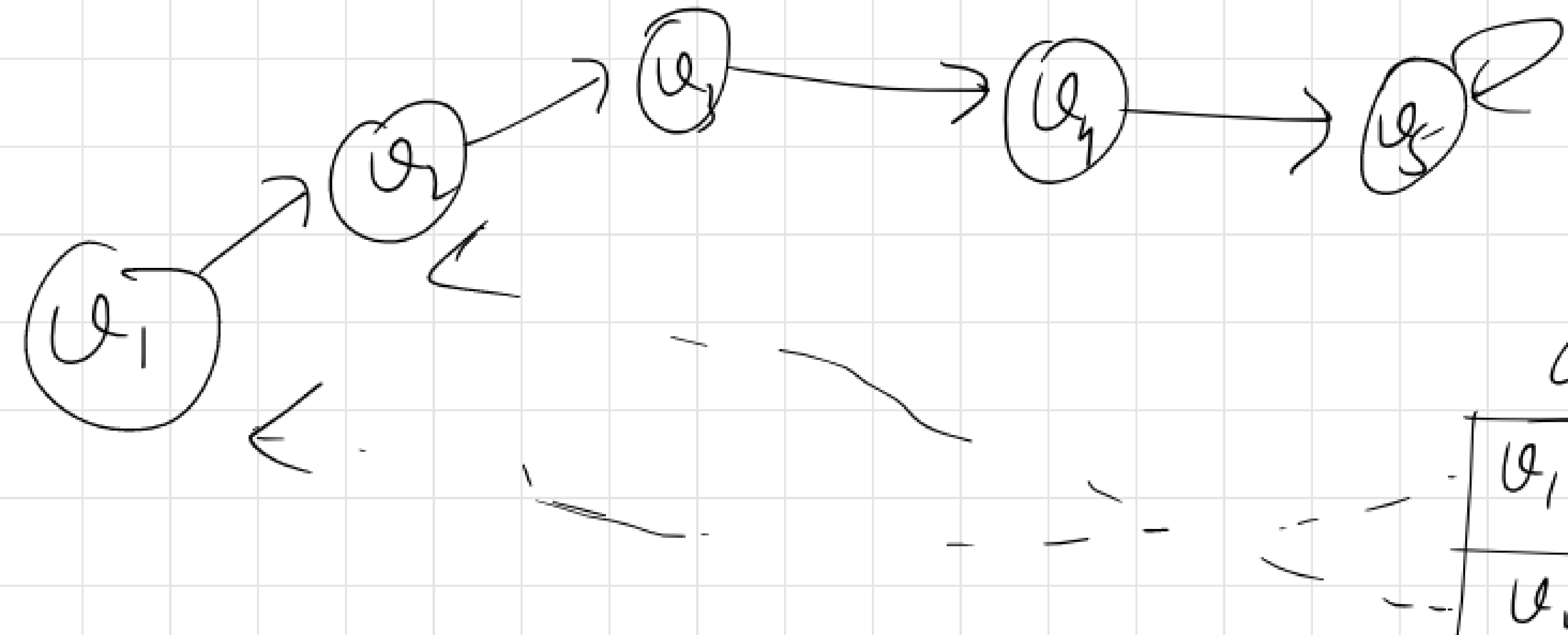
root

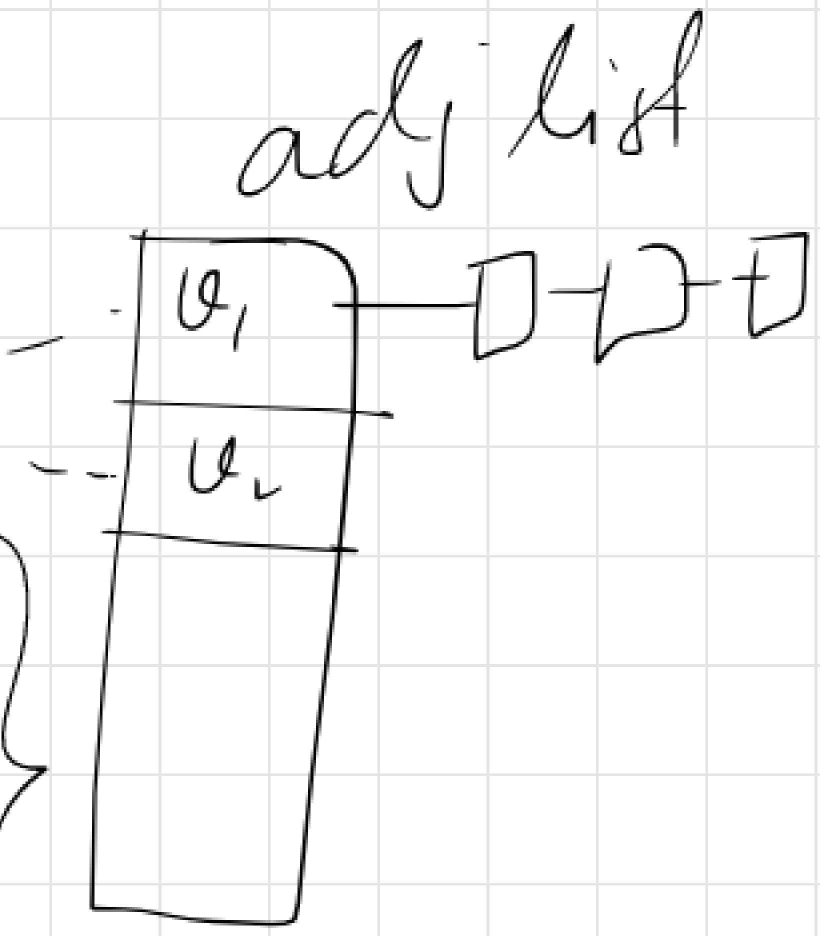
find(v_1) = v_3

Union(find(v_4), find(v_5))



Union(find(v_1), find(v_4))





Union (Reference of two nodes)

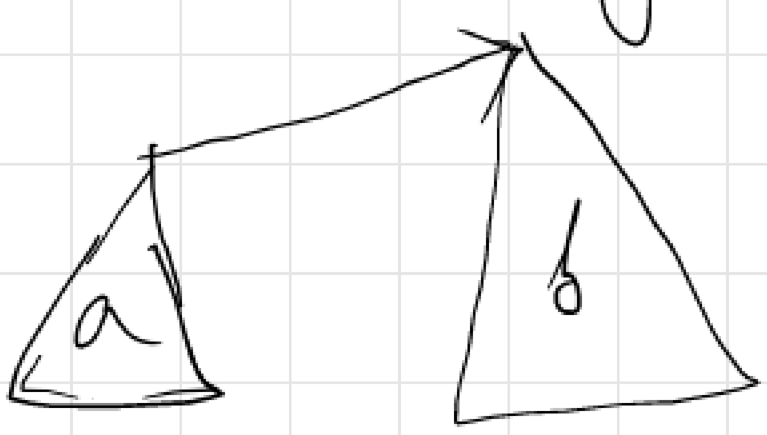
$\rightarrow O(1)$

<u>Find</u> $\rightarrow O(n)$

Union by rank

(New Rule)

nodes



$a < b$

How high tree can become if Union by rank rule applied

A tree with n nodes has height $\le \log n$

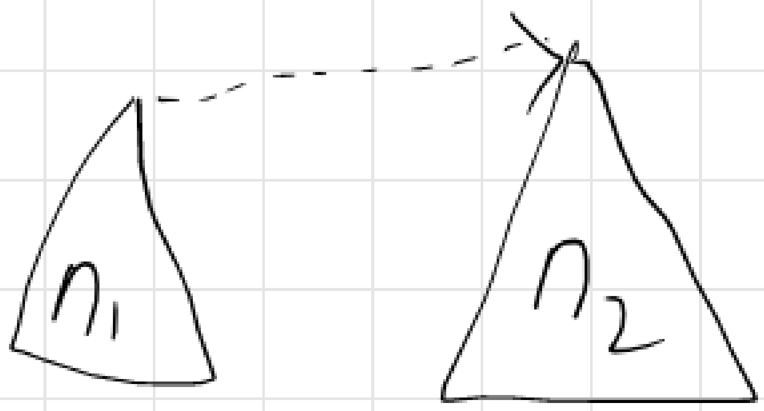
(by ind)

$n_1 \le n_2$

$\le \log n_1$ $\qquad$ $\le \log n_2$

(Goal $\le \log(n_1 + n_2)$ of new tree after union)

$n_1 < n_2$



new tree height $= \max\{\log(n_1) + 1, \log(n_2)\}$

$$\log(n_1)+1 \le \log 2n_1 \le \log(n_1+n_2) \quad (\because n_1 \le n_2)$$

$$\log(n_2) \le \log(n_1+n_2)$$

$n_1 = n_2$ Easy

Union by height will also work

$h_1 \le h_2$

A tree of height h atleast 2^n nodes

Proof : Homework.

Union by rank rule (Again..)

keep track # nodes in a tree in the root node

Union = $O(1)$

find = $O(\text{height}) = O(\log n)$

Time Complexity

$$m \log m + n + m \log n$$

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

$$\boxed{\log_2 n \le \log m \le 2 \log n \qquad \log m \in \Theta(\log n)}$$

$$\boxed{\cancel{m = O} \quad m = O(n^2) \quad \log m = \log n^2}$$