

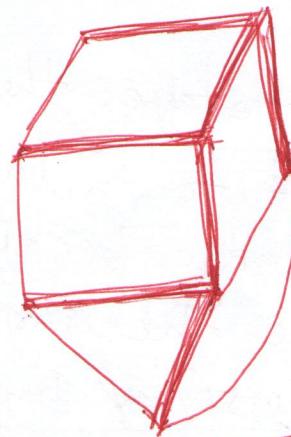
Minimum Spanning Trees

Let $G = (V, E)$ is undirected graph

Spanning Tree in G

↳ Connected subgraph without cycle.

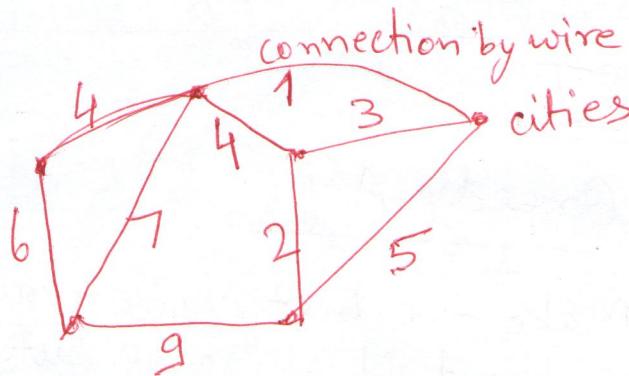
↳ It should include all vertices.



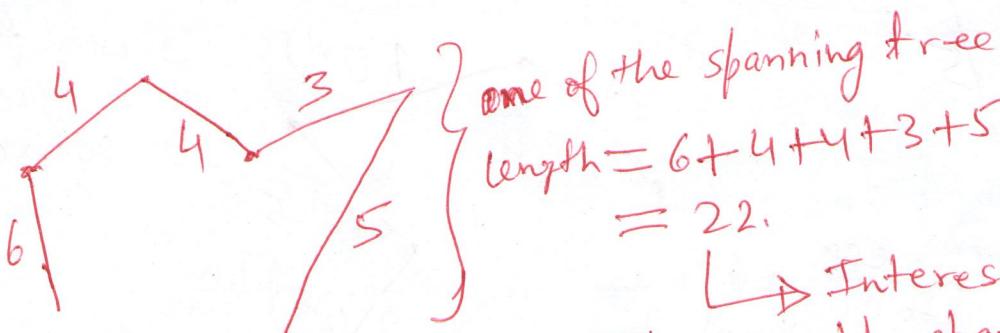
Spanning tree has $|V| - 1$ ~~vertices~~
edges.

Minimum Spanning tree:

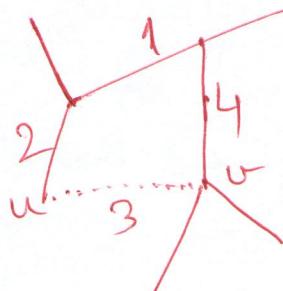
Spanning tree of minimum weight/length.
 $G = (V, E)$ $\text{length}(l) : E \rightarrow \mathbb{R}^+$
 or weight(w) : $E \rightarrow \mathbb{R}^+$



Length of the spanning tree
 = sum of the lengths of
 the edges in the tree.

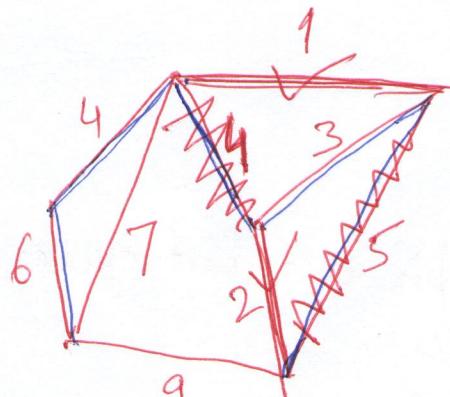


↳ Interested in finding
 the spanning tree having
 the lowest length.



⇒ In this case edge with length 4 can
 be dropped and edge with length 3
 can be included.

2

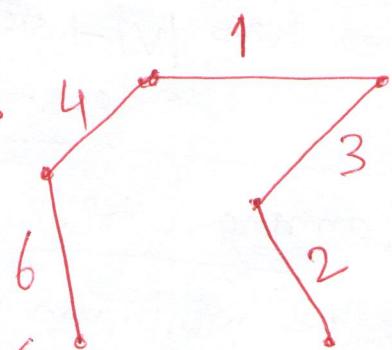


$$6+4+4+3+5=22$$

reduction after dropping the edge with more length and including the edge with lower length.

$$22 - 3 - 3 = \textcircled{16}$$

Resultant tree after reduction



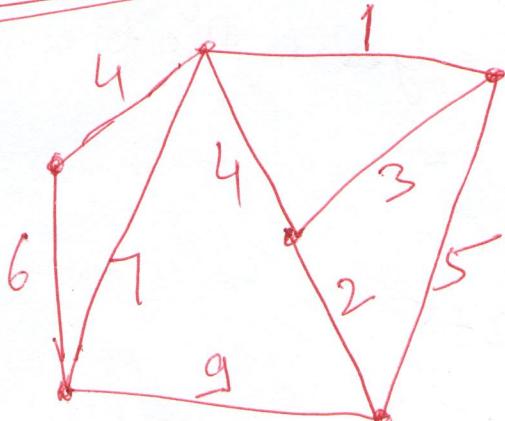
Now at this stage, there is no benefit to include any other edge because they will increase the length.

This is minimum spanning tree (MST) of length 16.
This algorithm will be costlier algorithm.

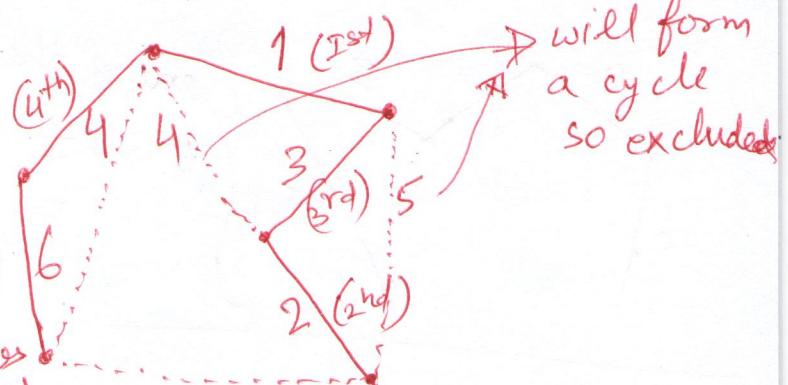
Kruskal's Algorithm for MST.

Greedy Algo.

Make the best choice available at each step without thinking of future.



We can stop at this point as all vertices are included.



Kruskal's algorithm for mst

(3)

1. Sort edges in increasing order of length
 $(e_1, e_2, e_3, \dots, e_m)$

$$T \leftarrow \emptyset$$

$$\text{s.t. } l(e_i) \leq l(e_{i+1})$$

~~2.~~ for $i \leftarrow 1$ to m do

if $T \cup \{e_i\}$ is a tree Then

$$T \leftarrow T \cup \{e_i\}$$

3. return T

Proof of correctness of Kruskal's algorithm

Assumption: all edge lengths are distinct \Rightarrow Same proof can be extended to non-distinct edge lengths also.

Kruskal's $\Rightarrow g_1 < g_2 < g_3 < \dots < g_i < \dots < g_{n-1}$

MST $\Rightarrow f_1 < f_2 < f_3 < \dots < f_i < \dots < f_{n-1}$

~~Optimum Tree considering best approach~~ Proof by contradiction

Suppose these sets differ & the first place they differ is i .

\hookrightarrow means $g_k = f_k$ for $k \in [1, i-1]$

& $g_i \neq f_i$

Case 1: $g_i < f_i \Rightarrow g_i \neq f_k$ for $k \in [1, i-1]$

because $g_k = f_k$ for $k \in [1, i-1]$

& $g_i \neq f_k$ for $k \in (i+1, n-1)$
 because $g_i < f_i$

$\Rightarrow g_i$ is not present in optimum solution

So Add g_i to optimum solution (opt tree) \Rightarrow

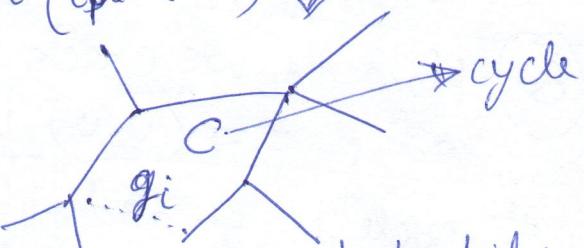
\hookrightarrow This will form a cycle (C).

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

~~Not present in optimum solution~~ \hookrightarrow Not possible, that's why Kruskal's algo picked it.

\Rightarrow It means, it is longest edge. However,
 \Rightarrow Thus it is contradiction.

If g_i is longest, all other edges must be from g_1 to g_{i-1} in cycle $= f_1 \to f_{i-1}$



Following concerns:
 1) Why it is correct? Is it possible to have some other MSTs?

2) How to check this statement? Is it hiding many things?

④

Case 2: $g_i > f_i$

$$\begin{array}{ll} \text{Kruskals} & g_1 < g_2 < \dots < g_{i-1} < \boxed{g_i} < \dots < g_{n-1} \\ \text{optimum} & f_1 < f_2 < \dots < f_{i-1} < \boxed{f_i} < \dots < f_{n-1} \end{array}$$

Case 2: $f_i < g_i \Rightarrow f_i \neq g_k$ for $k \in [1, i]$

because $f_k = g_k$ for $k \in [1, i]$

why did kruskal not pick f_i ?

& $f_i \neq g_k$ for $k \in [i+1, n-1]$

because $f_i < g_i$

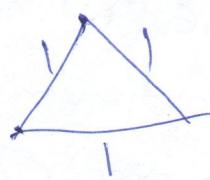
Because it formed a cycle with $f_i \cup \{g_1, \dots, g_{i-1}\}$.
 $\equiv f_i \cup \{f_1, \dots, f_{i-1}\}$.

\Rightarrow It means there is cycle in optimum
 also which is not possible so contradiction.

\Rightarrow It means there is no place where these two sets
 differ \Rightarrow So both sets are same.

\Rightarrow Kruskal's algo finds the MST.

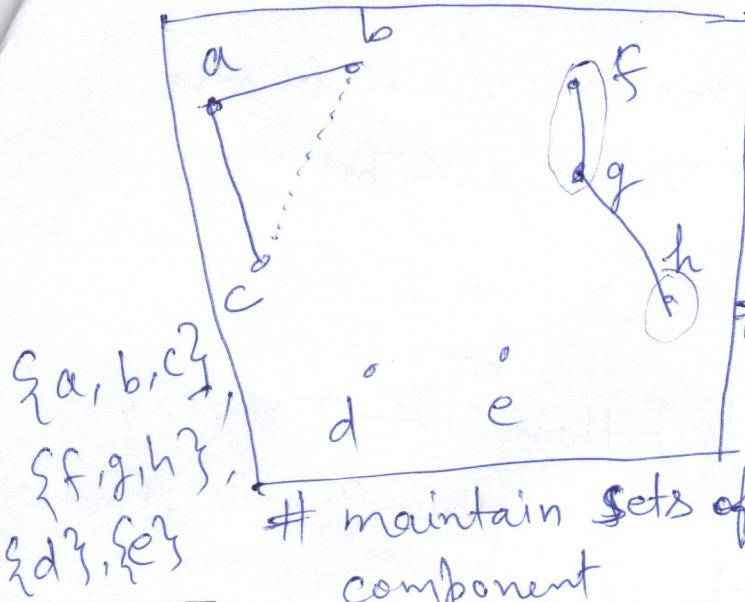
Is the MST unique? Yes, if edge lengths are
 No, otherwise.



Three MST.

② 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
 \hookrightarrow means u and v are in the same connected components.



whenever an edge is included it combines two connected components into one connected component.

Number of connected component is decreases by one every time a new edge is added.

maintain sets of vertices in each connected component

whenever we try to include an edge and both end points of that edge are in the same set, then a cycle will be formed.

This data structure needs to be maintained.

The Union Data Structure.

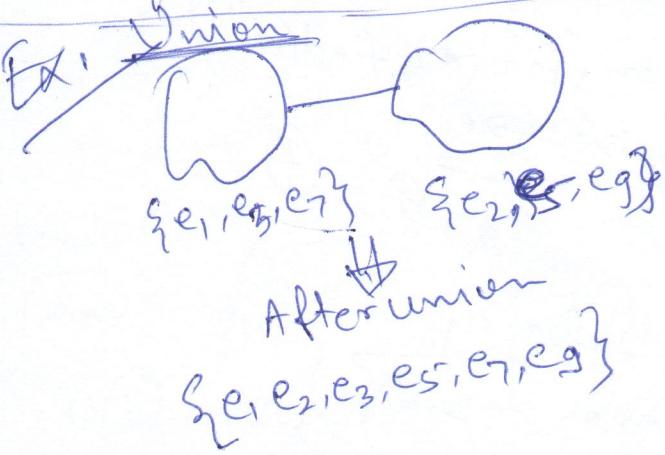
Initially n connected components (i.e., all vertices without edges)

Eventually we have 1 connected component

$$V = \{e_1, e_2, \dots, e_n\} \Rightarrow \text{Collection of disjoint sets}$$

where, $\{e_1\}, \{e_2\}, \dots, \{e_n\}$ are sets

maintain a collection of disjoint sets under the operations of union.



Find Operation

find(x, y) returns true if x, y are in the same set.

(or) returns two sets if x, y are not in the same set corresponding to $x \neq y$.

Basically
→ find(x, y) should return the sets corresponding to x and y . If both are different, call union.

⑥

Kruskal's Algo.

- sort the edges in increasing order of lengths
 $e_1 < e_2 < \dots < e_m$

- for $i = 1$ to m do

let $e_i = (u, v)$

if $\text{find}(u) \neq \text{find}(v)$ then

$T \leftarrow T \cup \{e_i\}$

$\text{Union}(\text{Find}(u), \text{Find}(v))$

Unions number of times: $n-1$

finds number of times: $\frac{m}{2}$

If Union & Find take $O(U)$ and $O(F)$ time

then [running time = $O(m \log m + U.n + F.m)$]

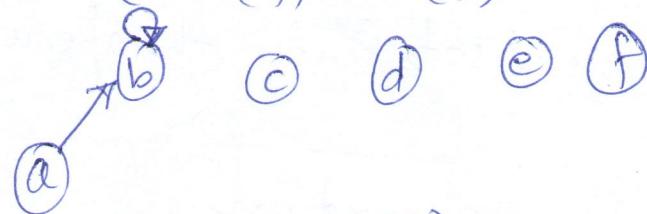
→ Needs to find a good data structure to have smaller $T, U \& F.$ times.

Union using Tree

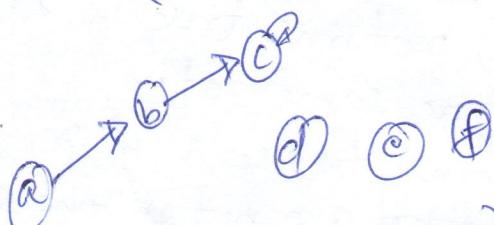
$$U = \{a, b, c, d, e, f\}$$



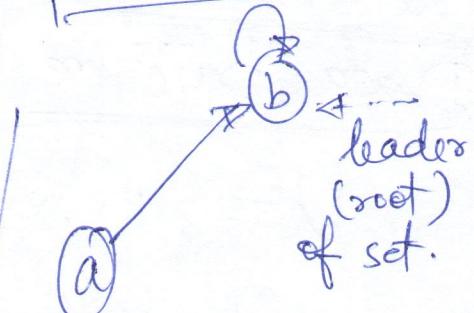
① $\text{union}(\text{find}(a), \text{find}(b))$



② $\text{union}(\text{find}(a), \text{find}(c))$

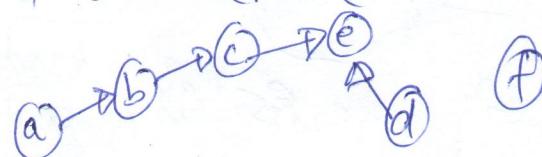


③ $\text{union}(\text{find}(d), \text{find}(e))$



find will return the reference of root node.

④ $\text{union}(\text{find}(a), \text{find}(f))$



Time complexity

Union → $O(1)$

Find → $O(n)$

Still not very good complexity

Union by Rank

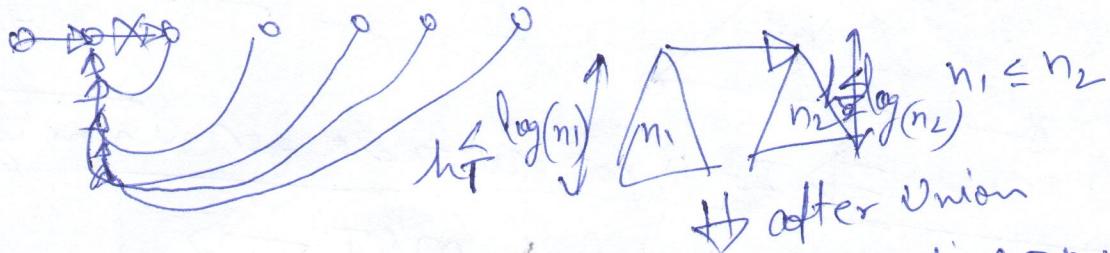


no. of nodes
 $n_1 < n_2$

claim

A tree with n_1 node has height $\leq \log(n_1)$

Using Induction



If $n_1 \leq n_2$

height of resulting tree $\leq \max(h_2, h_1 + 1)$

After Union

no. of vertices $= n_1 + n_2$
need to show height
 $\leq \log(n_1 + n_2)$

Union by height

It also works.

$$\begin{aligned} &= (\text{1st case}) \quad h_2 \leq \log_2(n_2) \leq \log_2(n_1 + n_2) \\ &= (\text{2nd case}) \quad h_1 + 1 \leq (\log_2 n_1 + 1) = \log_2(2n_1) \\ &\leq \log_2(n_1 + n_2) \end{aligned}$$



$h_1 \leq h_2$
claim

A tree of height h has at least 2^h nodes.

$$h = \max(h_2, h_1 + 1)$$

$$\text{no. of nodes} = n_1 + n_2 \geq 2^{h_1} + 2^{h_2} \geq 2^{h_2}$$

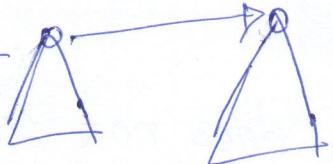
$$2^{h_1} + 2^{h_2} \geq 2^{h_1} + 2^{h_1} = 2^{h_1+1}$$

$$\Rightarrow \text{no. of nodes} \geq \max(2^{h_2}, 2^{h_1+1}) = 2^{\max(h_2, h_1 + 1)}$$

$$\Rightarrow \boxed{\text{no. of nodes}(n) \geq 2^h} = 2^h$$

Union:
(Time)

$O(1)$



only reference updating

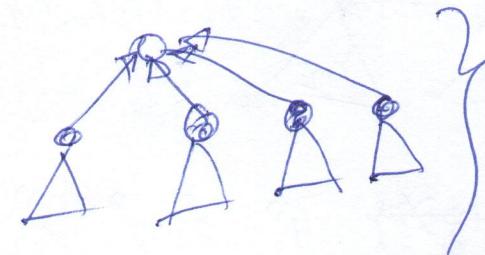
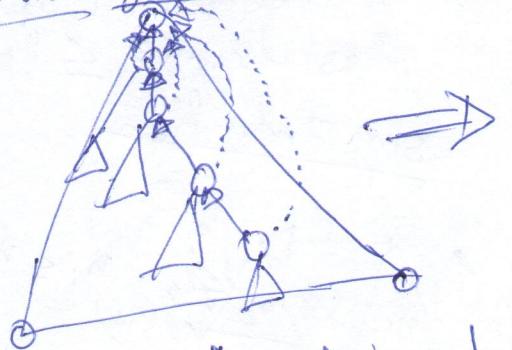
In a connected tree
 $\log n \leq \log m \leq 2 \log n$

find: time $O(\log_2 n)$.

height of the tree

Kruskal's algo time is
 $O(m \log m + n + m \log_2 n)$
 $= O(m \log_2 n)$

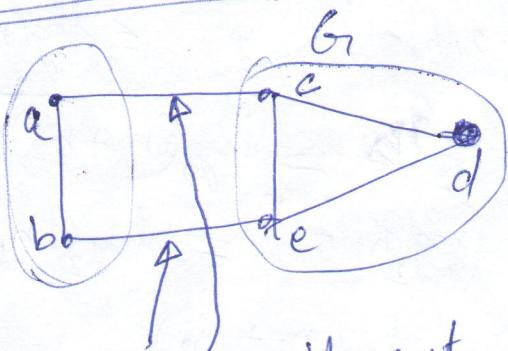
⑥ Path Compression for Union by Rank \rightarrow Improving find time further.



Reduces the height of the tree.

\Rightarrow It is not suitable for union by height.

Prim's Algorithm for MST \Rightarrow



Edges in the cut

A cut in graph G is a partition of the vertex set into parts.

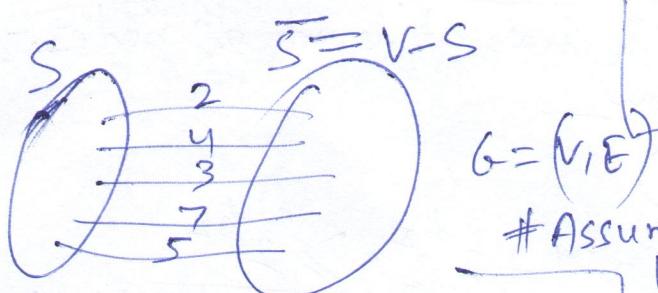
$\{a, b\}$ $\{c, d, e\}$

many cut possible

$\{a, c, d\}$ $\{b, e\}$

$\{c\}$ $\{a, b, d, e\}$

Total no. of cuts = $\frac{2^n - 1}{2}$



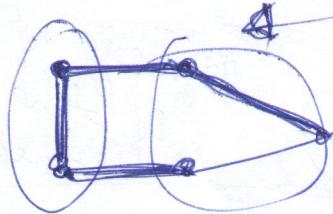
Assume edge lengths are distinct

↑ for simplification, However same argument would be valid for repetition also.

claim: for any cut (S, \bar{S}) ,

the minimum edge in the cut belongs to the MST.

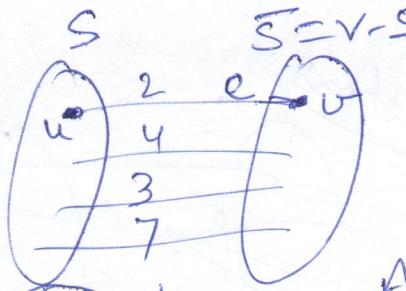
\Rightarrow However, more than one edge of cut can be also part of MST.



If the minimum edge of cut is not in MST then that MST is not correct.

Proof by Contradiction:

Let T be a MST which does not contain edge e .



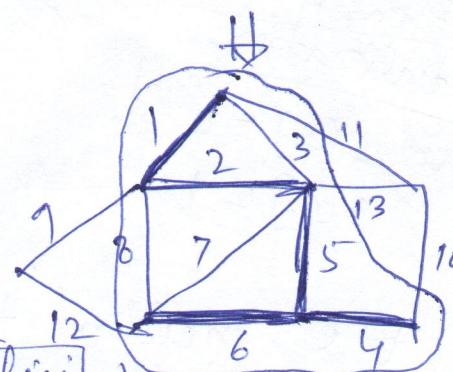
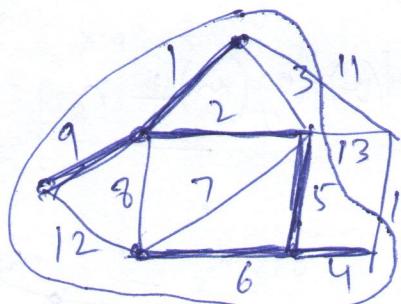
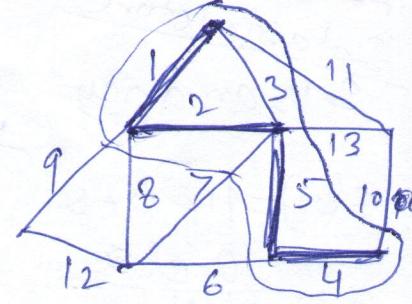
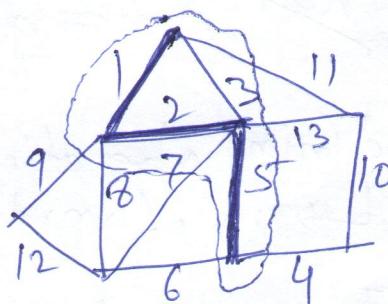
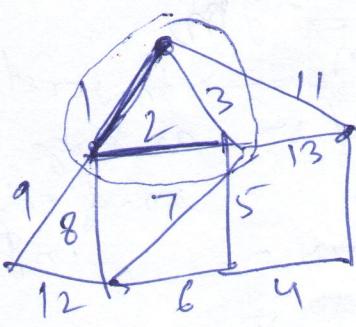
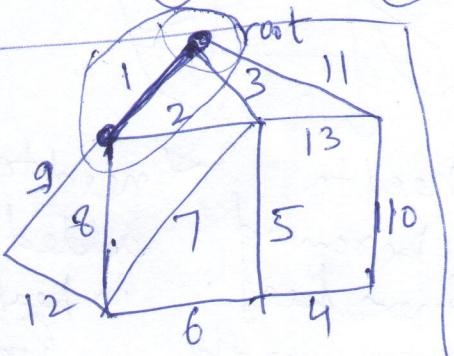
Proof: By Contradiction

Let T be a MST which does not contain edge e .

Add e to T

So cycle will be formed, say C .
 C contains at least one edge of the cut.
 $\Rightarrow C$ contains an edge of length more than the length of e .

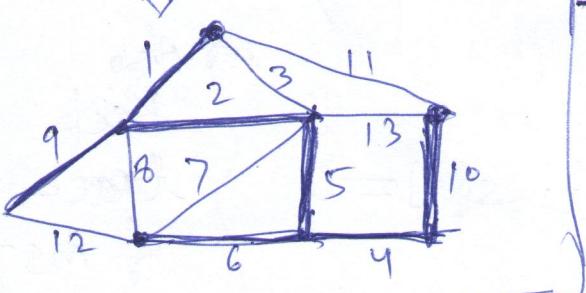
By removing this edge from $T \cup \{e\}$ we get a smaller tree. (contradiction).



How to implement efficiently

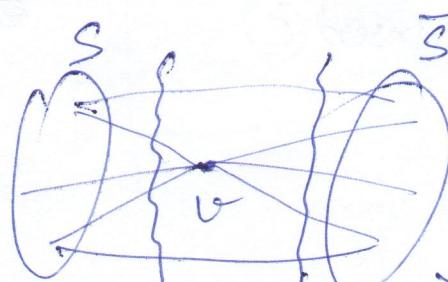
S

S-bar



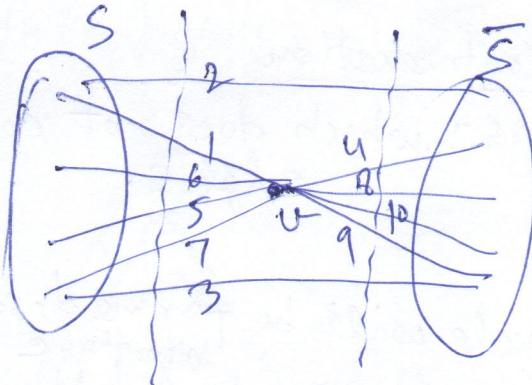
One-bit information is needed to classify any vertex either in set S or $S-bar$.

let The vertex going to move from $S-bar$ to S .

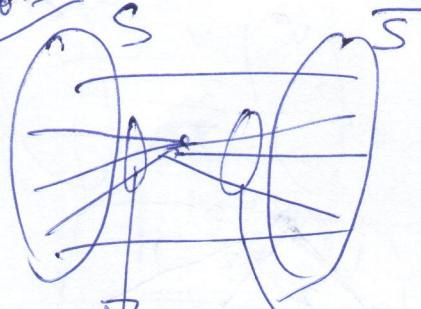


after v moved
we have to find min of
these edges.

⑩



Heap operations



use heap \Rightarrow will always contain the cut edges

remove edges from v to s
& add edges from v to \bar{s}

Data structure

s[i] = 1 if v \in S

= 0 else

& Heap H.

Initialization

$\forall v \quad s[v] = 0$

$s[r] = 1$

heap

$$\text{Time} = d(v) \cdot \log(m)$$

\hookrightarrow degree of v

+ find(min) operation

\hookrightarrow no. of vertices times

(log m)

↓

in worst case
as large as
no. of edges
in graph.

$$\Rightarrow \text{Time} = d(v) \log(m) + n \cdot q(1)$$

$$\text{Time} = d(v) \log(m) + n$$

$\forall e$ incident to r do

H.insert(e)

while !H.empty() do {

f = H.findmin()

let v be the end point of f s.t. $s[v] = 0$

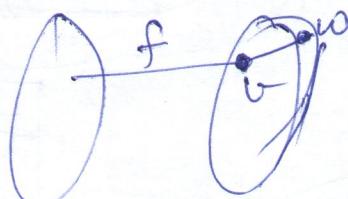
for all $e^{(v,w)}$ adjacent to v do

if $s[w] == 0$ then H.insert(e)

else H.delete(e)

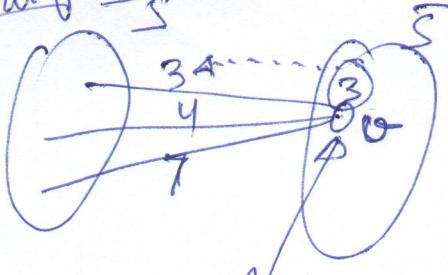
$s[v] = 1$

loop executed $n-1$ times.

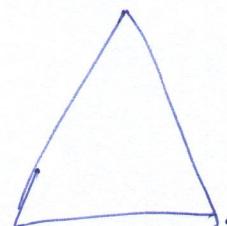


Also
keep
track
of MST
edges.

A modification



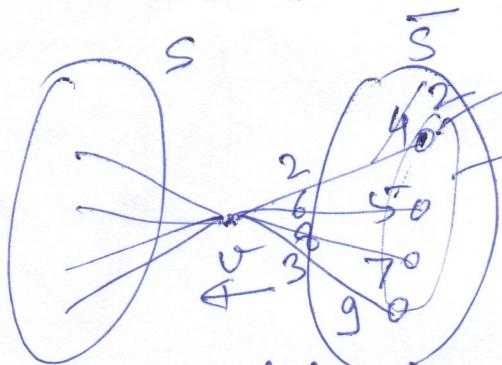
heap of vertices ⑪



If there is no edge from S to v then level is ∞ .

A number 3 is associated with v .
min edge length from S to v .

heap has one element for each vertex in S



level will be decreased to 2.
(updated).
the levels need to be updated if v moved from S to s

\Rightarrow find min and decrease-priority operations are needed for heap.

~~label~~ $s[i].label[i]$
 $\forall v \ s[v] = 0, H.insert(v, \infty)$
 $s[v] = 1, H.decreasePriority(v, 0)$

while $H.empty()$ do {

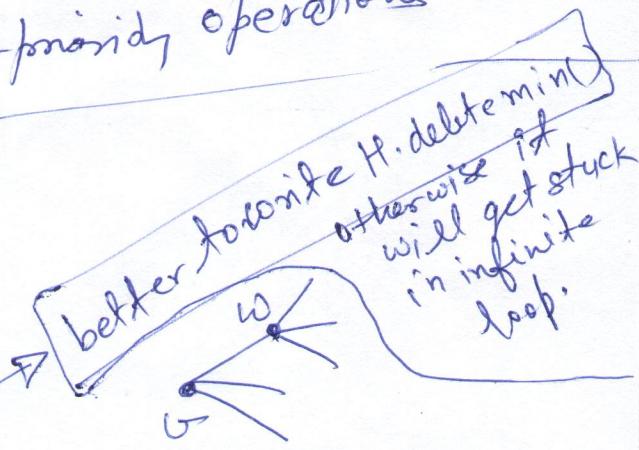
$v = H.findMin()$

for all w adjacent to v do

if $s[w] == 0$ then

| if $label[w] > \text{old length}(v, w)$ then
| $label[w] = \text{length}(v, w)$
| $H.decreasePriority(w, label[w])$

$s[v] = 1$



running same complexity as before