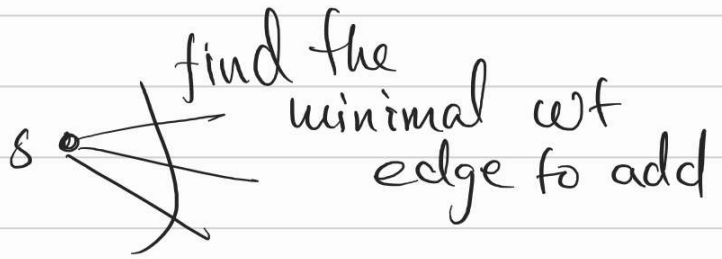
in the cut.

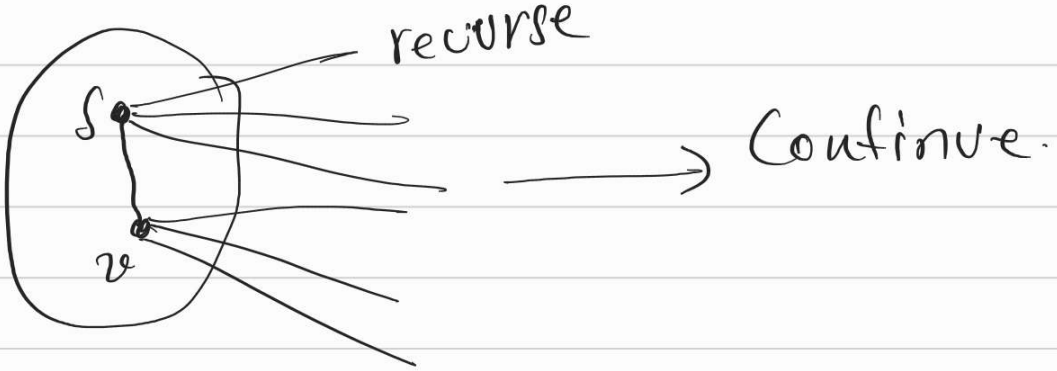
09/03/2022

MST (contd.)

Lemma :- Prim's algorithm produces a minimal spanning tree.

Proof :-

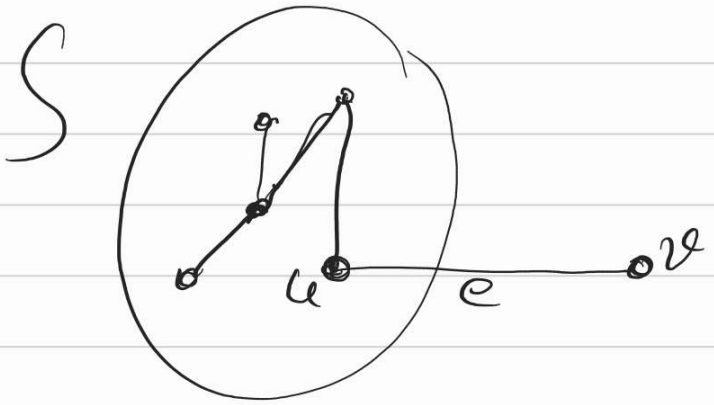




It produces a spanning tree by the behaviour of the algorithm.

why minimal ?

Consider an edge $e=(u,v)$ added by the algorithm.



S and $V \setminus S$.

By definition e is the minimal wt. edge crossing the cut $(S, V \setminus S)$.

Hence by cut property it belongs to every minimal spanning tree

∎

Reverse-Delete Algo: (1) Sort edges in decreasing order

(2) delete edges if deleting it doesn't disconnect the graph.

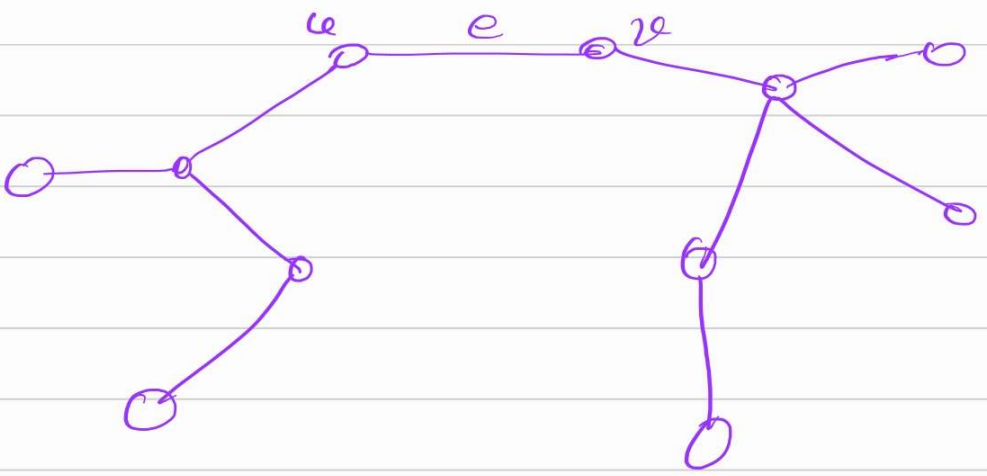
(Cycle Property)

Lemma: Assume all edge weights are distinct. Let C be a cycle in the graph G and let $e = (u, v)$ be the edge with maximal wt. on this cycle C. Then e doesn't belong to any minimal spanning tree.

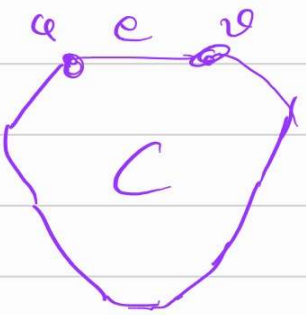
Proof: Let T be a spanning tree that contains the edge $e = (u, v)$.

It suffices to show that T doesn't have minimal wt.

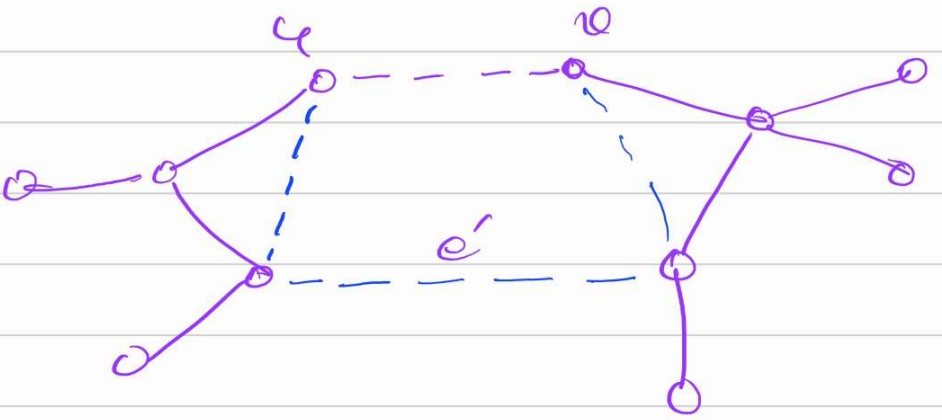
In other words, we will construct another spanning tree T' s.t. weight of T' is smaller than weight of T.



T



if we delete e from T we obtain.

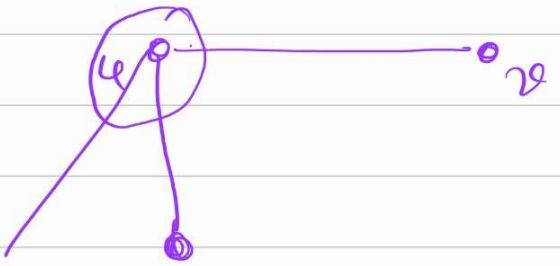


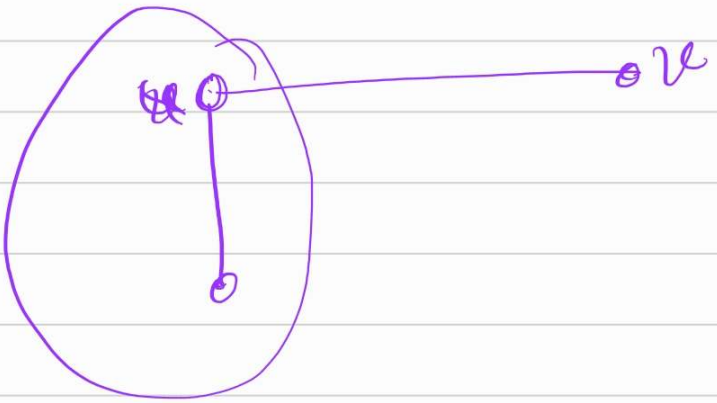
Consider $T' = T \setminus \{e\} \cup \{e'\}$.

Easy to see T' is a spanning tree.

Since e' and e are part of the cycle C, $wt(e') < wt(e)$.

$\therefore$ T' has smaller wt. than T.

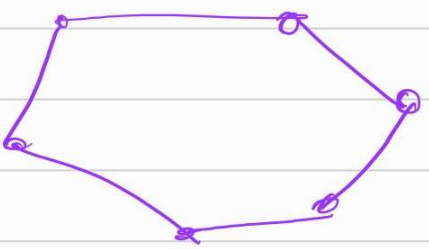




Lemma: Reverse-Delete Algorithm produces the minimal Spanning tree.

Proof:- If produces a connected graph.

Suppose there is a cycle. Consider the edge with max. weight on this cycle.



Why was this edge not deleted?

Why is it minimal?

Consider an edge e deleted by this algorithm.

When it is about to be deleted it must be part of a cycle.

On this cycle this is the first edge to be considered.

⇒ this is the maximal wt. edge on this cycle.

Now invoking cycle property you conclude that this edge doesn't belong to any minimal spanning tree.

⇒ edges at the end are part of some minimal spanning tree.

⇒ the output is, in fact, a minimal spanning tree.
(use the assumption edge weights are distinct).

→ Mixing cut property and cycle property in any way gives another algorithm for MST.

→ perturb edge wt. with small amount so that original ordering between edge wt. remain preserved.

$$e_1 < e_2 < e_3 = e_4 = e_5 < e_6 < e_7 = e_8 = e_9$$

(perturbation

$$e_1' < e_2' < e_3' < e_4' < e_5' < e_6' < e_7' < e_8' < e_9'$$

▷ break ties arbitrarily but in a consistent way. (put a total order on the edges).

$e_1 < e_2 < e_3 < e_4 < e_5 < e_6 < e_7 < e_8 < e_9$

$S = \{a, b, c\}$ $\xrightarrow{\text{a total order means}}$ $a < b < c$, $b < a < c$, $b < c < a$.

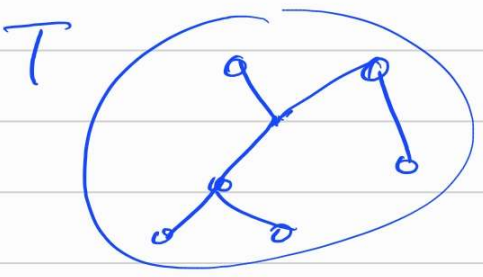
You can order the elements in a line from minimum to maximum.

Implementing Kruskal's and Prim's Algo.

$|E| = m$ and $|V| = n$

Goal to get a runtime : $O(m \log n)$.

Prim's Algorithm:



Using priority queue. — Extract Min, Insert.

maintain edges in a priority queue with keys as their weights.

Suppose Starting at s:

→ Add all the edges incident to s to the priority queue.

→ Extract-min. This gives an edge e.

→ Check if both the end-points of e belongs to T.
if yes then discard.
if not then add this to tree.

→ add edges incident to this edge to the priority queue.

Extract min will take $O(\log m)$

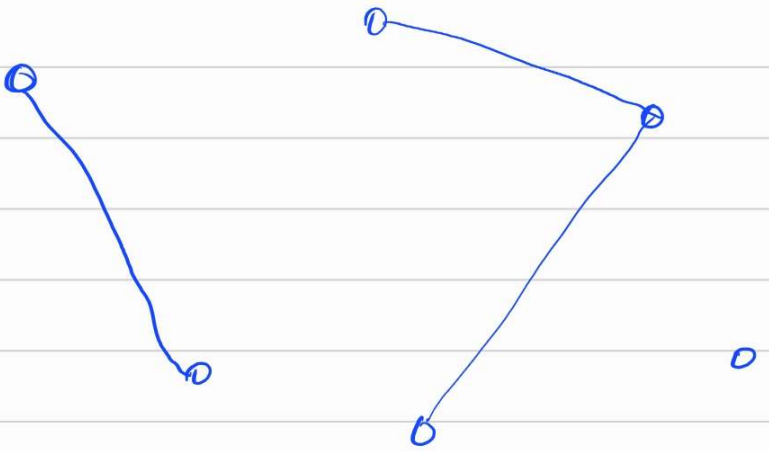
$\parallel$

$O(\log n)$

In the worst case,
you will call Extract-min for every edge once.

$\therefore$ total runtime $O(m \log n)$.

Implementing
Kruskal's Algo:



Union-Find data structure

→ MakeUnionFind(S)

outputs singleton sets containing each element of S.

if $|S| = n$, then it takes $O(n)$ times.

→ Find(u): Outputs the name of the set that contains u.
— $O(1)$ time.

→ Union(u, v): Merge the two sets containing u and v.
— $O(\log n)$ time

— an implementation s.t.
a sequence of k union operation take a total of $O(k \log k)$ time.