2024/10/29 - Data Structures

* Graph terminology
* BFS , DFS

→ Read the obsidian notes ☺

<u>Problem 22.1.3</u> : Compute $G^T = (V, E^T)$ from G, $E^T = \{(u, v) \in V \times V \mid (v, u) \in E\}$

let $G^T.adj$ [|V|] be the adjacency list of G^T.

for each vertex $v \in G.V$

    for each vertex $u \in G.adj[v]$ // $(v, u) \in E$

        $G^T.adj[u].push\_back(v)$ // $(u, v) \in E^T$

let $G^T.A$ represent the adjacency matrix of G^T

for $i = 1$ to $|V|$

  for $j = 1$ to $|V|$

    $G^T.A[j][i] = G.A[i][j]$

22.1.4: Multigraph to equivalent undirected graph

Given: $G = (V, E)$ is a multigraph

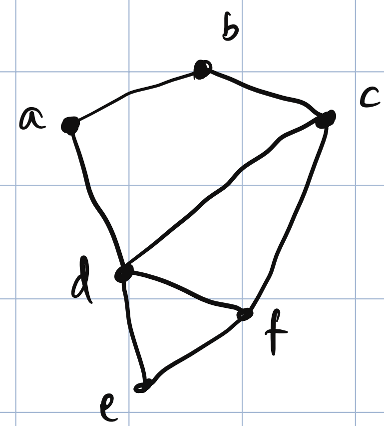
Let $G.adj[|V|]$ be the adjacency list of G.

Let $G' = (V, E')$ be the equivalent undirected graph with adjacency list $G'.adj[|V|]$.

Equivalent - Undirected - Graph

```
int π[|V|] = {0}          // array of |V| elements, initialized to 0.
for each vertex u ∈ G.V
    for each v ∈ G.adj[u]
        if (v ≠ u && π[v] ≠ u)
            π[v] = u ,   G'.adj[u].push_back(v)
```

22.1.5 : Square of a directed graph



Let $G^2.\text{adj}[|V|]$ represent the adjacency list of $G^2 = (V, E^2)$ where $E^2 = \{ (u, v) \in V \times V \mid G.\text{adj}[u] = v \vee G.\text{adj}[\text{adj}[u]] = v \}$

Square - graph

```
for each u in G.E
    for each v in G.adj[u]
        G^2.adj[u].push_back(v)
        for each x in G.adj[v], G^2.adj[u].push_back(x)
```

$$\sum_{u} \left(\deg(u) \left(1 + \sum_{1}^{\deg(u)} \deg(v)_{{}^{2}E} \right) \right)$$

$$O(V(1+E))$$

Adjacency matrix $\longrightarrow$ compute A^2

for $i = 1$ to $|V|$

  for $j=1$ to $|V|$

$$G^2.A[i][j] = \sum_{k=1}^{|V|} \{G.A[i][k] \times G.A[k][j]\}$$

$O(V^3)$

22.1.6 Universal sink

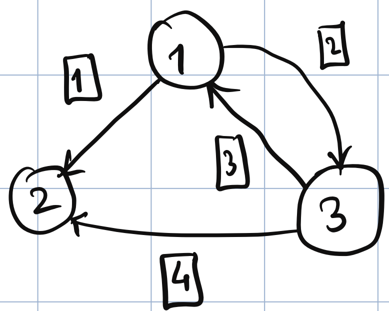
```
sink = true
for i = 1 to |V|
    for j = 1 to |V|
        if (j ≠ i && (A[i][j] == 1
                || A[j][i] == 0))
            sink = false

    if (sink == true)
        return true

return false
```

$$\begin{bmatrix} 0 & 0 & \cdots \\ 1 & & \\ 1 & & \\ \vdots & & \end{bmatrix}$$

```
univ-sink (G)
  i = j = 1
    while i < |V| and j < |V|
        if G.A[i][j] == 1
            i++;
        else j++;
  is_sink(G, i)
```

22.1.7 Incidence matrix

$$b_{ij} = \begin{cases} -1 & i \xrightarrow{j} x \quad x \in V \\ 1 & x \xleftarrow{j} i \quad x \in V \\ 0 & \text{o/w} \end{cases}$$



$$\begin{array}{c c} & \begin{array}{cccc} \boxed{1} & \boxed{2} & \boxed{3} & \boxed{4} \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 1 & 1 & -1 & 0 \\ -1 & 0 & 0 & -1 \\ 0 & -1 & 1 & 1 \end{bmatrix} \end{array} = B$$

$$B\,B^T = \begin{bmatrix} 1 & 1 & -1 & 0 \\ -1 & 0 & 0 & -1 \\ 0 & -1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & -1 & -2 \\ -1 & 2 & -1 \\ -2 & -1 & 3 \end{bmatrix}$$

Let $\quad B = (b_{ie})_{n \times m}$

$$B^T = (b^T_{ej})_{m \times n} = (b_{je})_{m \times n}$$

and $\quad C = (BB^T)_{n \times n}$

then $\quad c_{ij} = \sum_{e \in E} b_{ie} \cdot b_{je}$

If $\quad i = j$

$$c_{ii} = \sum_{e \in E} (b_{ie})^2$$

$b_{ie}^2 = 1$ if e has i as one of its ending or starting point

$= 0$ o/w

$\therefore \quad C_{ii} = \deg(i) \longrightarrow ①$

If $i \neq j$,

$$C_{ij} = \sum_{e \in E} b_{ie}\, b_{je}$$

$b_{ie}\, b_{je} = 1 \longrightarrow$ not possible, since edges are directed

$b_{ie}\, b_{je} = 0 \longrightarrow$ if $e \neq (i,j)$ and $e \neq (j,i)$

$b_{ie}\, b_{je} = -1 \longrightarrow$ if $e = \underset{1 \times (-1)}{(i,j)}$ or $e = \underset{-1 \times 1}{(j,i)}$

$\therefore \quad \underset{i \neq j}{C_{ij}} = -$(# of edges that connect i and j) $\longrightarrow ②$

22.1.8 Hash table instead of linked lists

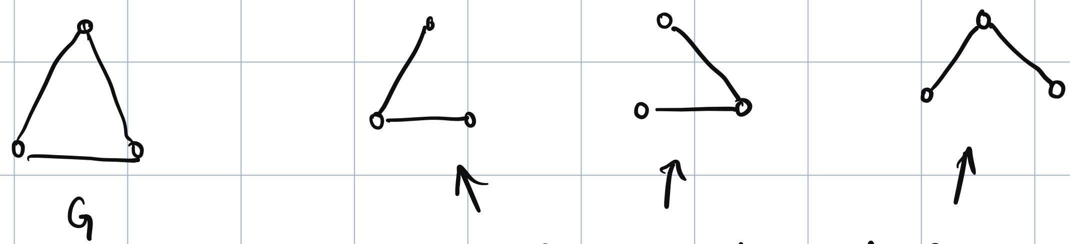
2024/10/30

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

Spanning tree:

(i) should include all the vertices in G

(ii) connected subgraph without cycle



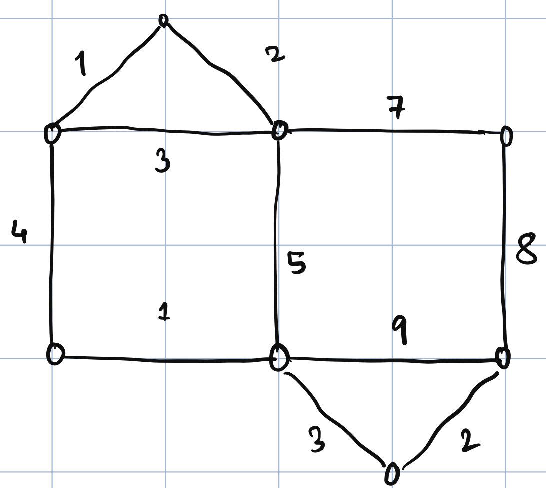
Spanning trees of G

} weight of each edge is same.

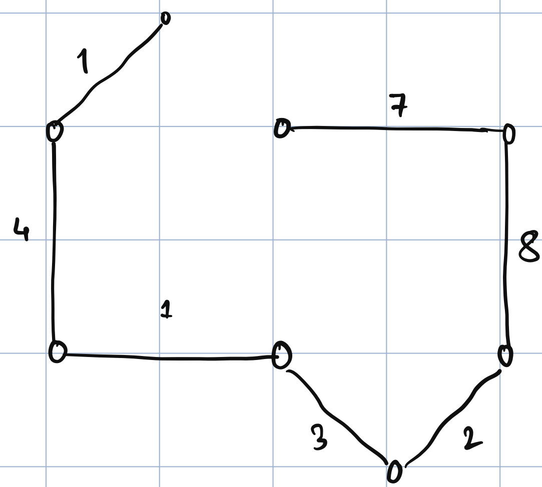
Minimum Spanning Tree :

$$G = (V, E)$$

$$W : E \to \mathbb{R}^+$$



G



$\underbrace{\sum w = 26}$

weight associated with a spanning tree

Weight of spanning tree = sum of weight of all edges in T

Minimum Spanning Tree := spanning tree of min weight.

How to find Minimum Spanning Tree?

Kruskal's Algorithm

Input $G = (V, E)$, $|V| = n$, $|E| = m$, $W: E \to \mathbb{R}^+$

① Sort edges based on the ascending order of weight:

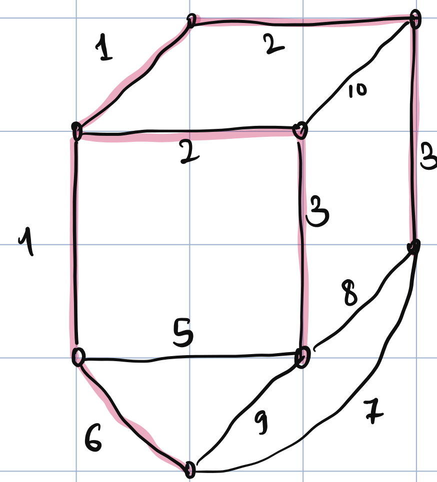
$(e_1, \dots, e_m) \longrightarrow W(e_i) \leq W(e_{i+1})$

② $T \leftarrow \phi$ /* T stores edges of MST */

③ For $i = 1$ to m and while (# edges(T) $\leq n-1$) do

if $e_i \cup T$ is a tree then

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



$g_i :=$ wt of i^{th} edge in Kruskal

$f_i :=$ wt of i^{th} edge in Optimum

Proof of correctness: Kruskal's algorithm outputs MST

Assumption: All edges weight are distinct

KRUSKAL: $g_1 < g_2 < g_3 < \overset{g_i}{\cdots} < g_{n-1}$

$\quad\;\, \parallel \quad\; \parallel \quad\; \parallel \quad \nparallel$

Optimum: $f_1 < f_2 < f_3 < \overset{f_i}{\cdots} < f_{n-1}$

Suppose the sets differ at first position at i^{th} position

Case 1: $g_i < f_i$

We take $T_{opt} \setminus \{f_i\} \cup \{g_i\} := T_{opt}'$

Case 1.1: T_{opt}' does not contain cycle.

$\Rightarrow$ T_{opt}' is spanning tree of weight lesser than T_{opt}'

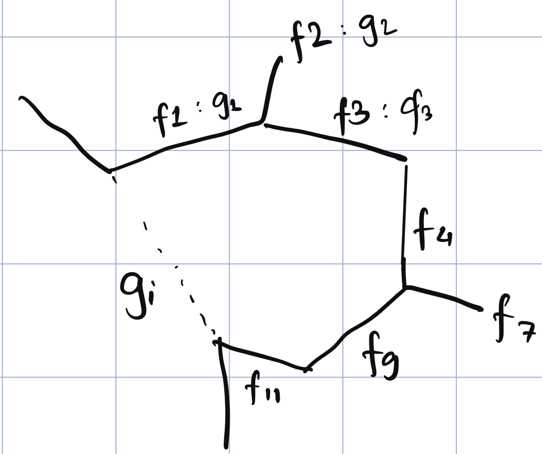
$\underset{T_{opt}}{\Rightarrow} \underset{MST}{\Leftarrow}$

Case 1.2: T'_{opt} contains a cycle

Why OPT prefer f_i over g_i — the only possible reason is adding g_i will introduce a cycle

Let $\underset{\overset{\|}{g_1}}{f_1}$, $\underset{\overset{\|}{g_3}}{f_3}$, $\underset{\overset{\|}{g_4}}{f_4}$, $\underset{\overset{\|}{g_9}}{f_9}$, $\underset{\overset{\|}{g_{11}}}{f_{11}}$

make cycle with g_i



then

$g_1, g_3, g_4, g_9, g_{11}, g_i$

make a cycle

Case 2: $g_i > f_i$

Why did KRUSKAL not pick f_i?

The only reason could be $\{g_1, \ldots, g_{i-1}\} \cup f_i$ forms a cycle

But $\{g_1, \ldots, g_{i-1}\} \cup f_i$

Proof of correctness – M2:

Lemma: This algorithm creates a spanning tree.

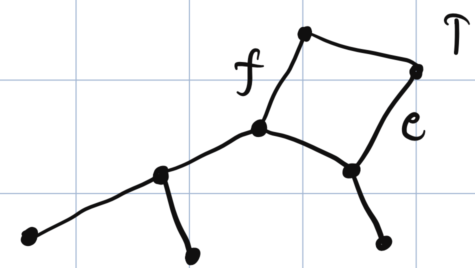
Lemma: This algorithm yields a spanning tree of minimum cost

Proof via induction:

Inductive hypothesis: Suppose that the first k edges are part of a MST T.

Base step: empty set of edges (part of a minimum spanning tree).

Inductive step: Show that adding $(k+1)^{th}$ edge e is either a part of T or some other spanning tree of equal cost.

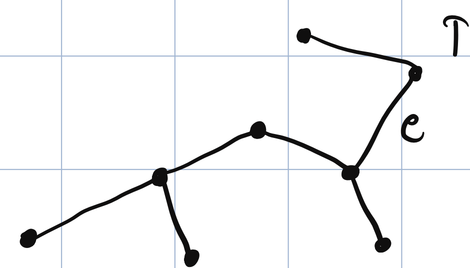


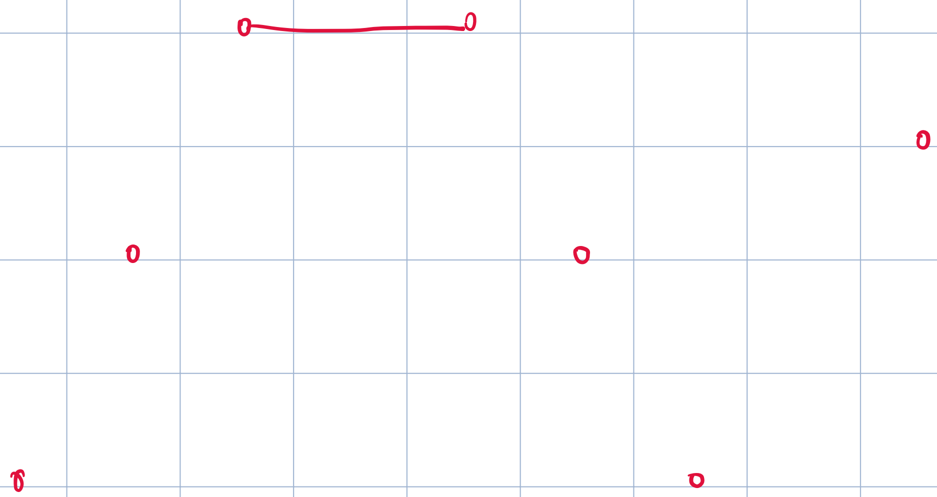
1. This edge e is actually part of T.
2. This edge e is not part of T.
   ↳ $\{e\} \cup T$ forms a cycle
   → some edges in this cycle are not a part of first k edges

→ Guarantee $w(f) \geq w(e)$ ($\because$ we are adding edges in increasing order)

→ Remove f

$$w(T \setminus \{f\} \cup \{e\}) \leq w(T)$$

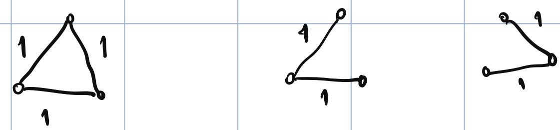




Is MST unique?

Yes if all edges are distinct

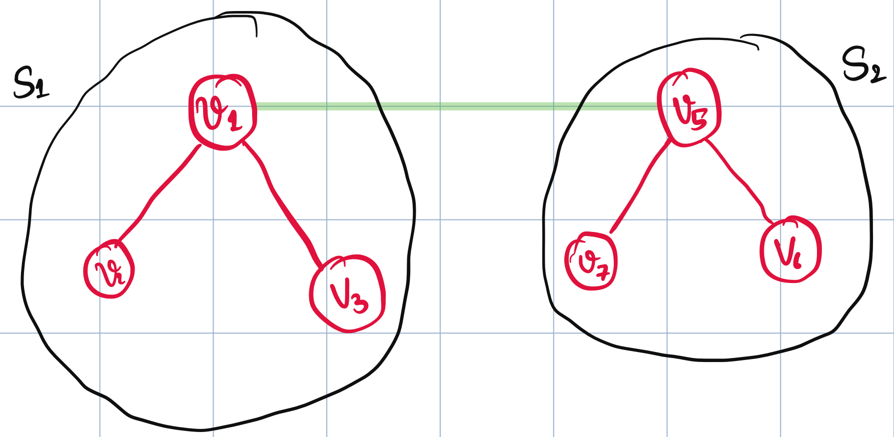
No o/w.



Initially, we start with n connected components

In each iteration we add one edge that reduces # of connected component by 1.

Finally we are left with 1 connected component ~ MST



Set union

$$S_1 = \{v_1, v_2, v_3\}$$

$$S_2 = \{v_5, v_6, v_7\}$$

after adding edge

$$S_1 \cup S_2 = \{v_1, v_2, v_3, v_5, v_6, v_7\}$$

Find :

Suppose edge (v_2, v_3) is added $\rightarrow$ cycle (when both vertices forming the edge belong to the same set)

Return true if v_2, v_3 are in same set

Union Find data structure

Kruskal's algorithm

1. $T \leftarrow \phi$
2. $S = \{v_1\} \dots \{v_n\}$
3. Sort the edges in increasing order of weights

$e_1 < \dots < e_m$ $\Big\}$ $O(m \log m)$

④ For $i = 1$ to m and while $(\#\text{ edges}(T) < n-1)$

do {

let $e_i = (u, v)$

if $(\text{Find}(u) \neq \text{Find}(v))$

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

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

}

\# of times union runs $\longrightarrow$ $n-1$ $\quad O((n-1) \cdot U)$

\# of times find runs $\longrightarrow$ $4m$ $\quad O((m) \cdot F)$

Total running time : $O(m \log m + (n-1)U + m \cdot F)$

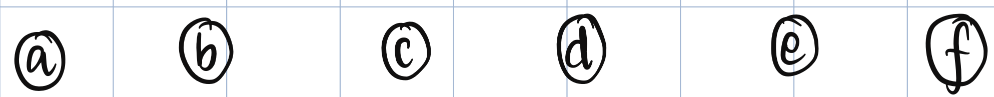
$O(m \log m + (n-1) + m \log n) = O(m \log n)$

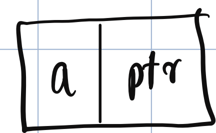
Naive solution: Use linked list

Union $O(1)$ time

Find $O(m)$ time ← not acceptable

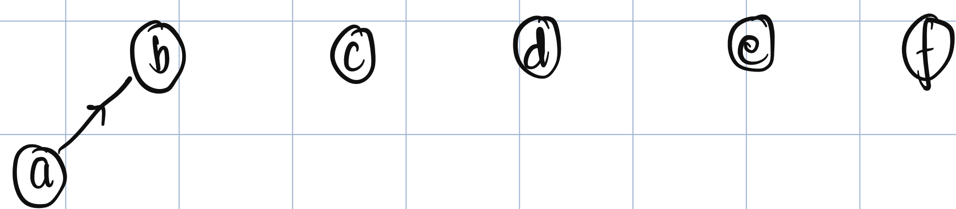
$\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f\}$





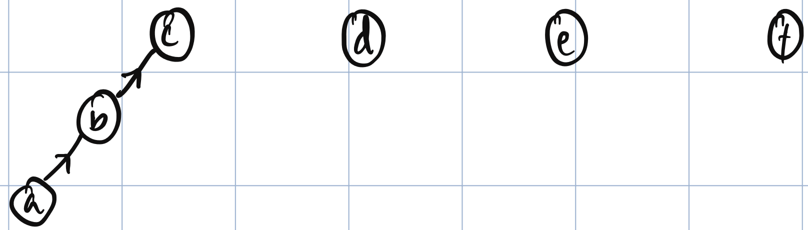
Union (find (a), find (b))

$\Downarrow$



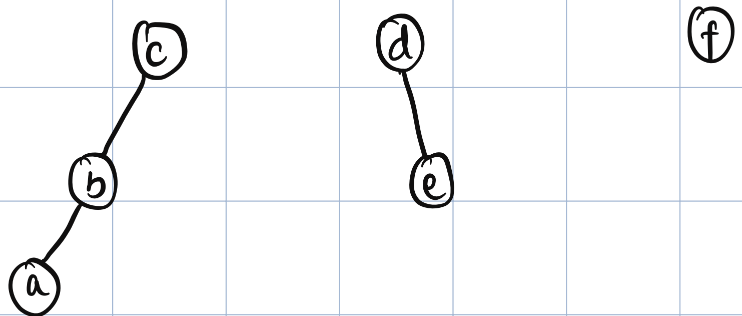
$\{a, b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f\}$

Union(find(a) , find(c))



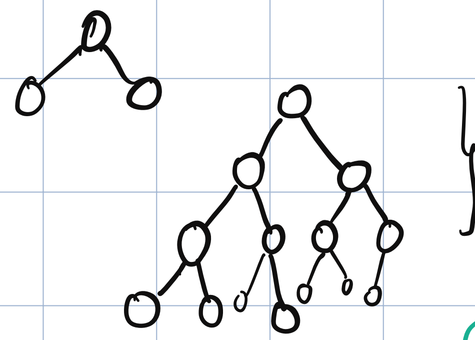
{a, b, c} {d} {e} {f}

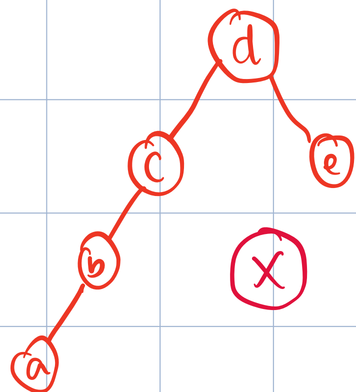
Union (find(d), find(e))



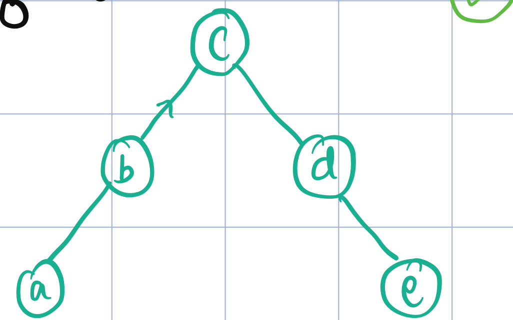
{a, b, c} {d, e} {f}

Union (find (a) , find (d))

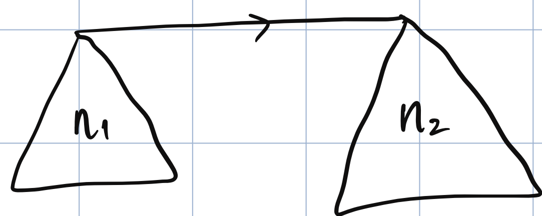




or



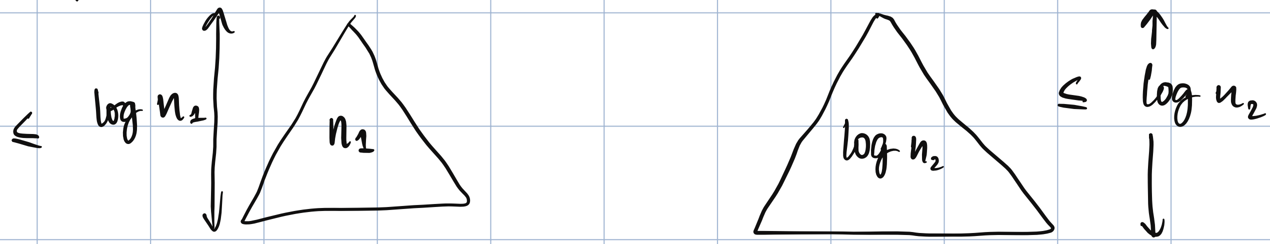
Union by rank



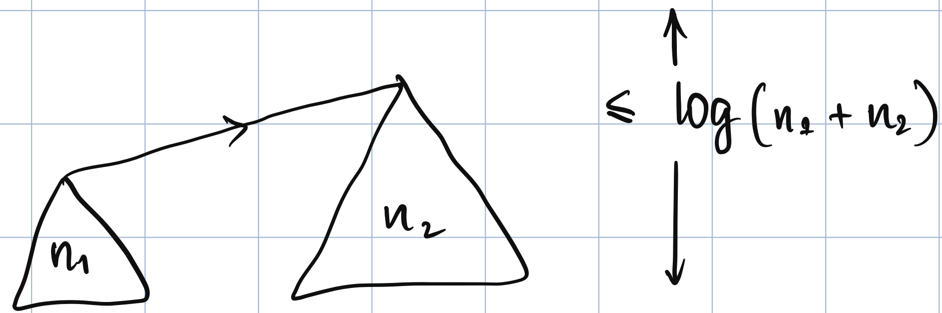
$n_1 < n_2$

Theorem: A tree with n nodes has height $\leq \log n$ (if we follow union by rank procedure)

Proof: Proof by induction



$\Longrightarrow$



Suppose $n_1 < n_2$, then height of resultant tree

$$= \max\{\log n_2 ,\ \log n_1 + 1\}$$??

$$\overset{?}{\leq} \log(n_1 + n_2)$$

$$\log n_2 \leqslant \log (n_1 + n_2),$$

$$\frac{n_1 + n_2}{n_1} \geqslant 2$$

$$(\log n_2) + 1 \leqslant \log(n_1) + \log\left(\frac{n_1 + n_2}{n_1}\right)$$

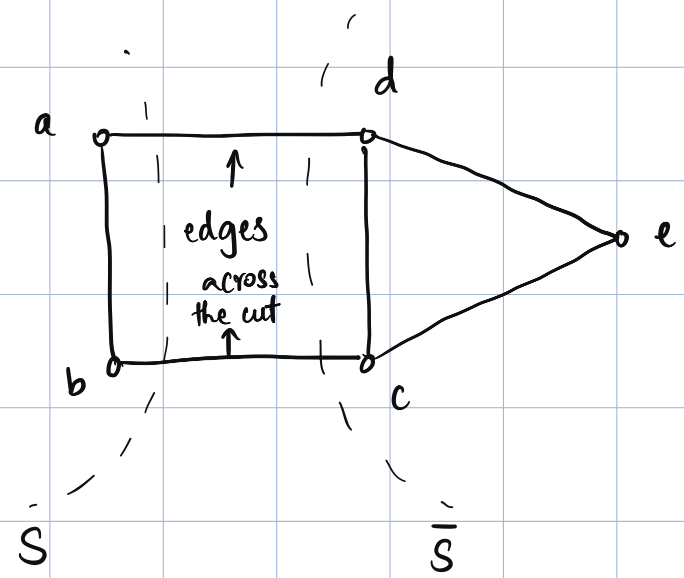
$$\log\left(\frac{n_1 + n_2}{n_1}\right) \geqslant 1$$

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

2024/11/01 – Prim's Algorithm

Cut in a graph

A cut in G is a partition of vertex set into two parts



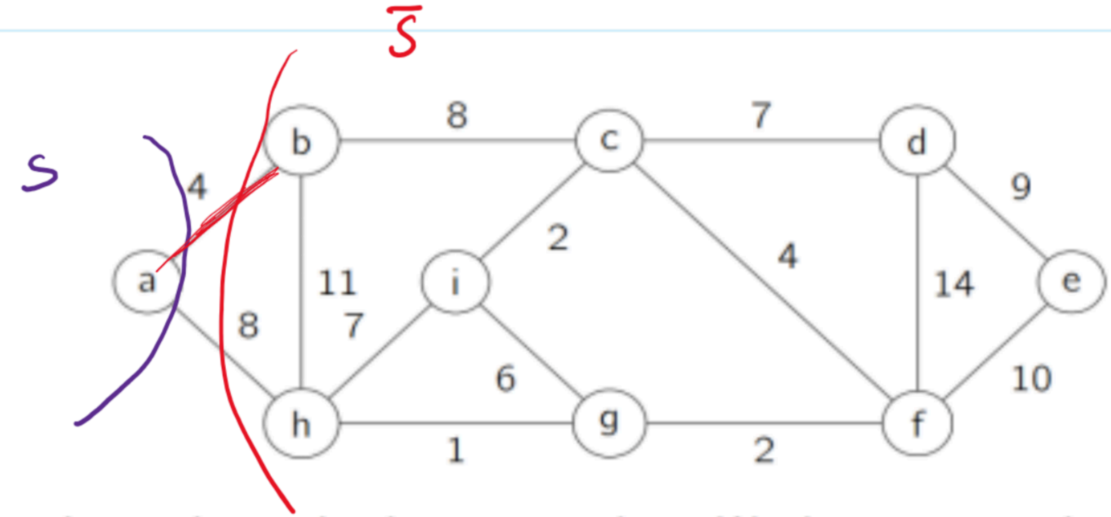
$S = \{a, b\}$

$\bar{S} = \{b, c, e\}$

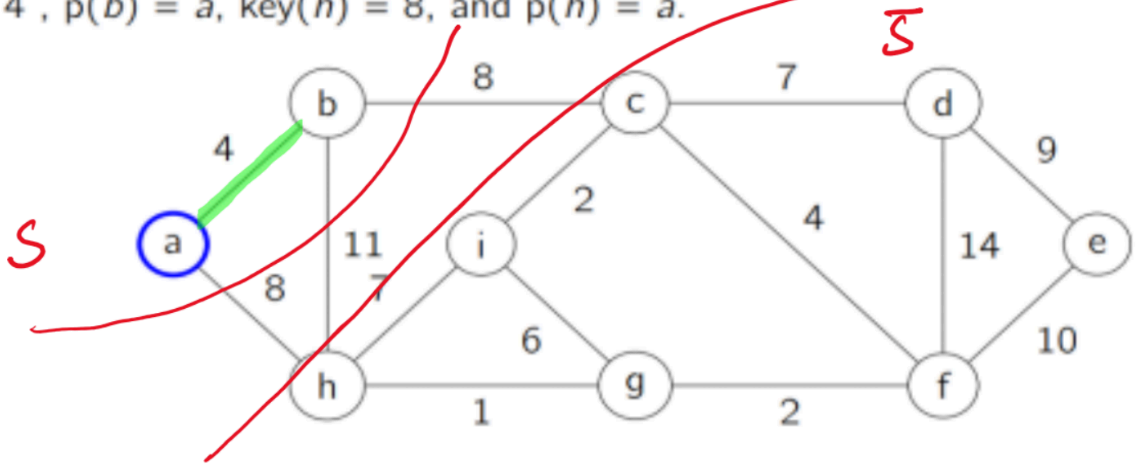
$\#$ of possible cuts $= 2^n - 2$

$S = \phi$

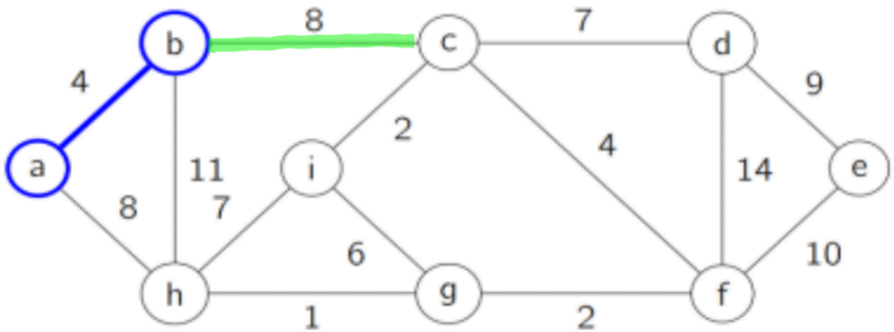
$\bar{S} = \phi$

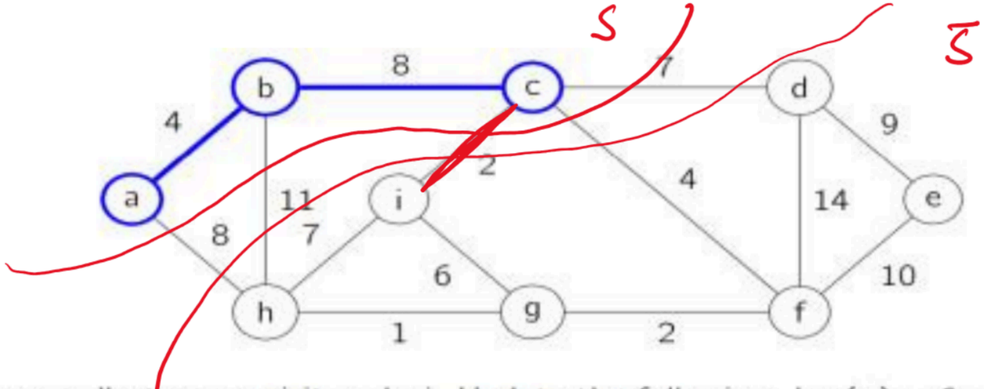


Suppose we select node a to be the source node, r. We then extract node a from Q and set key(b) = 4 , p(b) = a, key(h) = 8, and p(h) = a.

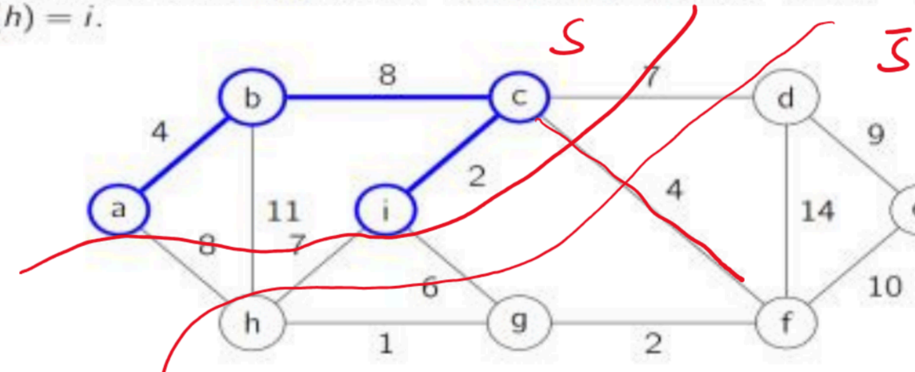


Since key(b) is now the smallest value in the priority queue, we visit node b. Because p(b) = a we add edge (a, b) to the set A. We then update the keys and parent fields of nodes that have edges connecting to b. Thus we set key(c) = 8 and p(c) = b.

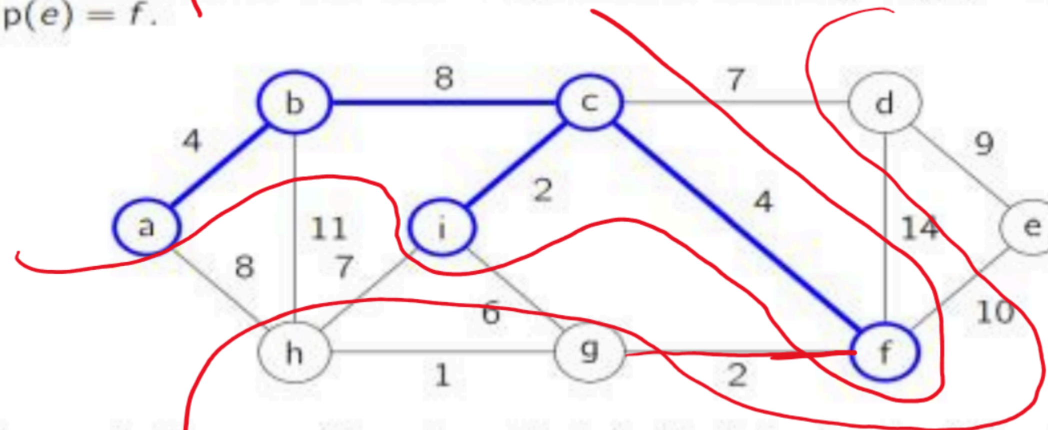




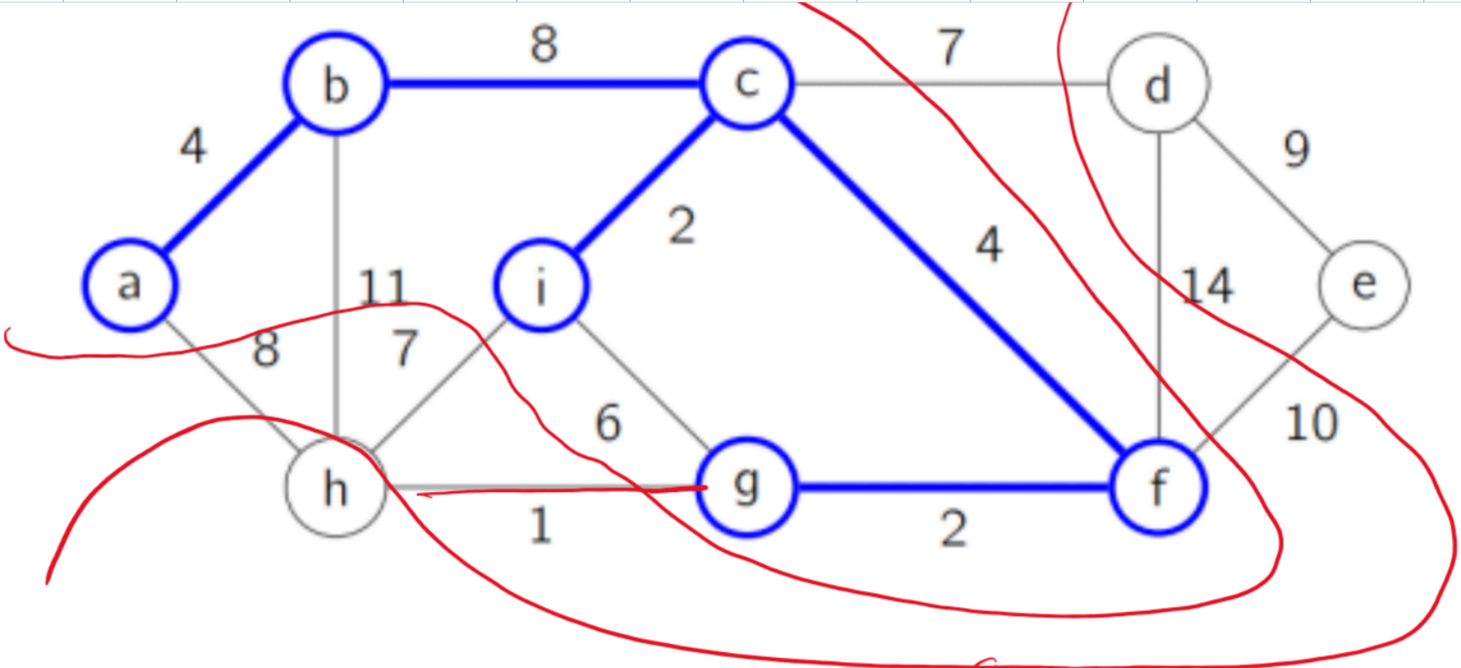
key(i) is the smallest so we visit node i. Update the following: key(g) = 6, p(g) = i, key(h) = 7, and p(h) = i.

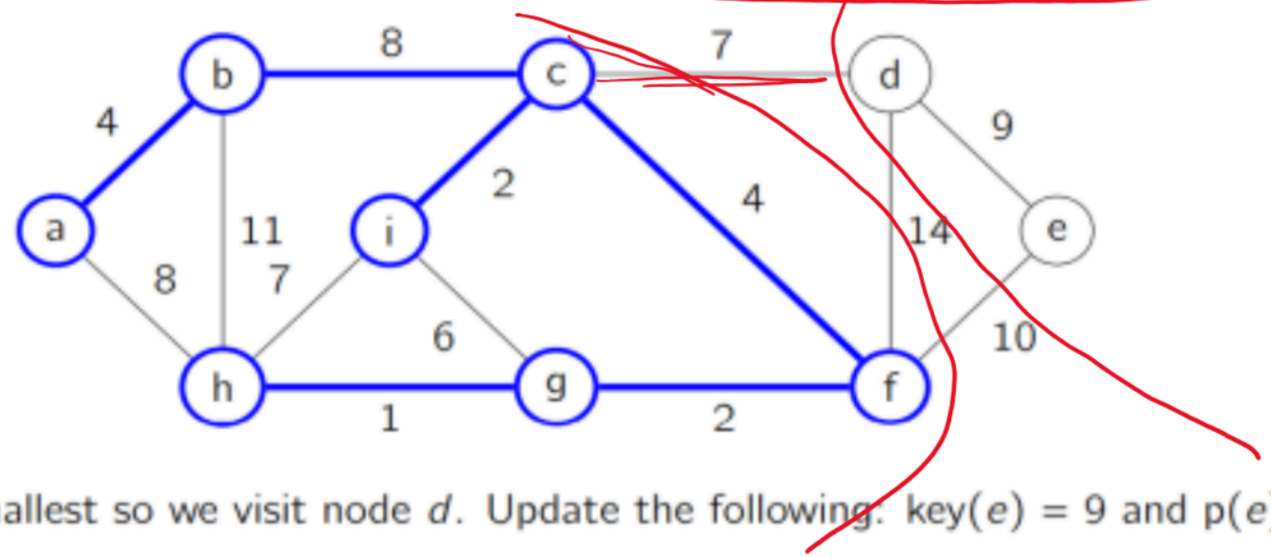


key(f) is the smallest so we visit node f. Update the following: key(g) = 2, p(g) = f, key(e) = 10, and p(e) = f.

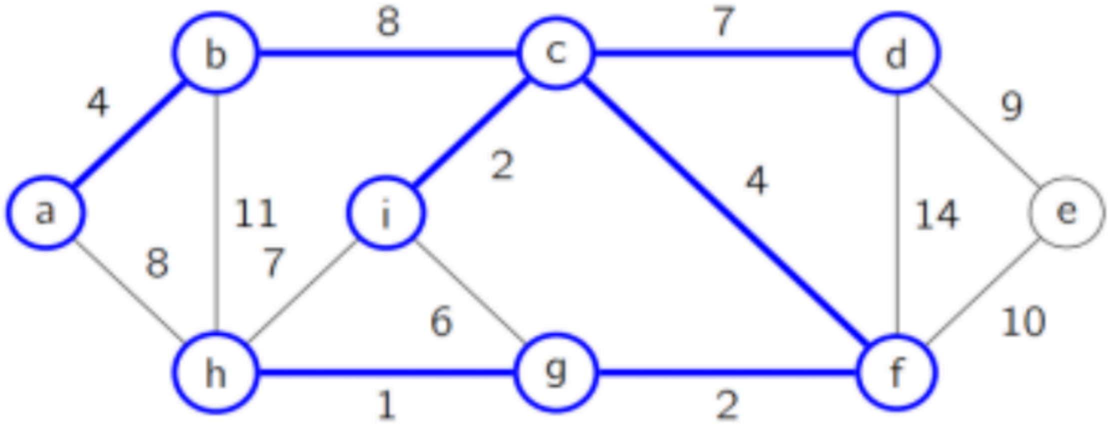


key(g) is the smallest so we visit node g. Update the following: key(h) = 1 and p(h) = g.

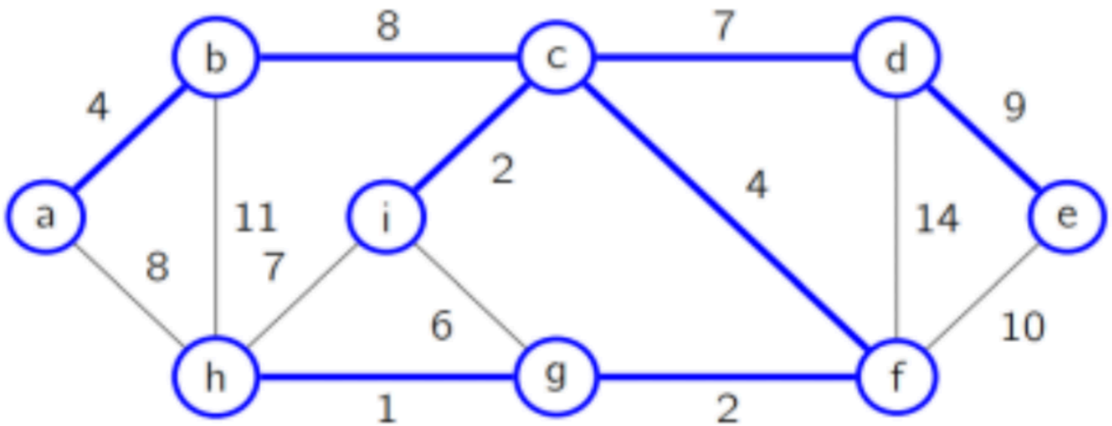




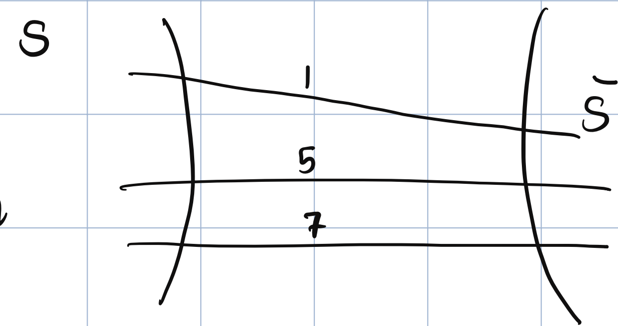
key(d) is the smallest so we visit node d. Update the following: key(e) = 9 and p(e) = d.



Finally, key(e) is the smallest so we visit node e. There are no updates at this step and the algorithm will detect that Q is empty at the next iteration and return.



Claim: For any cut $(S, \bar{S})$, the minimum weight edge should belong to MST



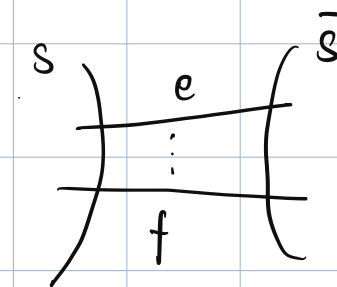
Remark: More than one edge of cut can also be part of MST. But minimum weight edge must be part of MST

Proof: By contradiction, consider a cut $(S, \bar{S})$

$T :=$ MST that does not contain edge e.

$T \leftarrow T \cup \{e\}$

Addition of e creates a cycle (say C)



$e :=$ smallest edge weight across the partition

C has to contain atleast one more edge (say f) across the partition.

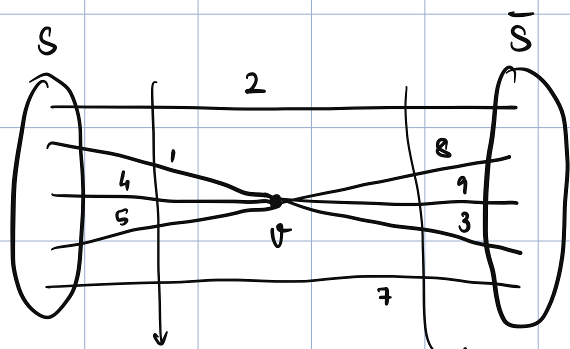
$\Rightarrow$ C contains an edge of weight more than weight of e. (b/w S and $\bar{S}$)

Remove edge f

$\Rightarrow$ spanning tree of weight smaller than that of T

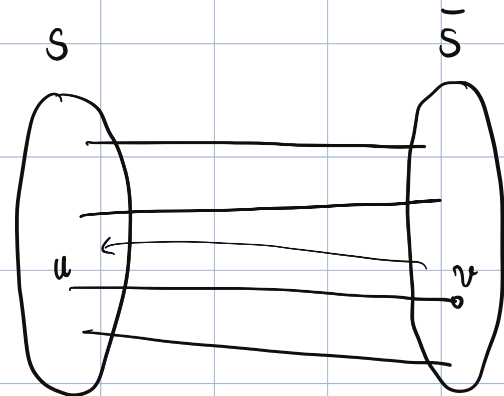
$\Rightarrow \Leftarrow$
T MST

Prim's Algorithm $\longrightarrow$ see slides



know min of these edges

We want to know minimum of these edges



Let (u, v) be the minimum weight edge

edges of weight 4, 5 need to be removed from heap.

edges of weight 8, 9, 3 need to be added in heap.

At any point, heap maintains the edges across the partition.

For every vertex v that we process, we remove some edges and add some edges in heap

- Revise insertion and deletion in heap

Overall time complexity $= \left(\sum_{v} \deg(v)\right) \log m + n \cdot O(1)$

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

Algorithm

S is an array.

$S[v] = 1 \implies v \in S$

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

$S[r] = 1$ /* root vertex */

$\forall e$ incident to r

H.insert (e) // insert e in heap

while heap is not empty do

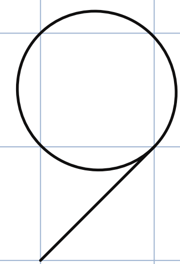
f = H.findmin( ) /* f is the edge with min wt across the cut

let v be the endpt. of f s.t. $S[v] = 0$

for all edges $e = (v,w)$ adjacent to v do

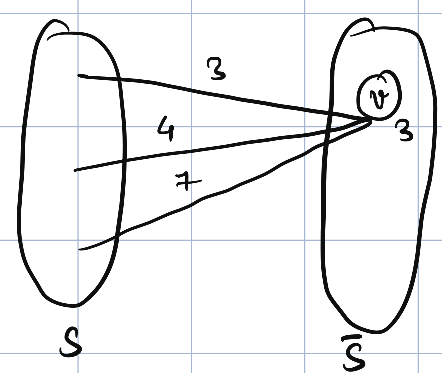
$\deg(v) \left\{ \begin{array}{ll} \text{if } S[w] = 0 : & H.\text{insert}(e) \\ \text{else} & H.\text{delete}(e) \end{array} \right.$

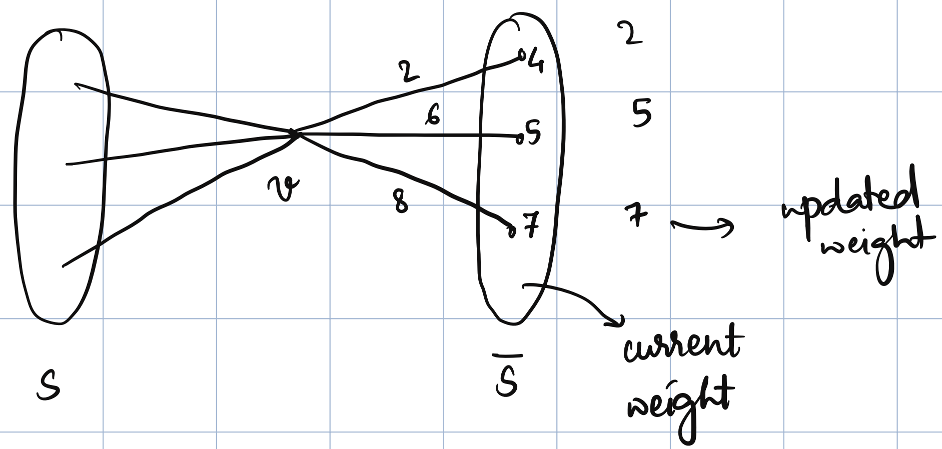
$S[v] = 1$



Modification

For each vertex in $\bar{S}$, store a label which is weight of the minimum edge across the cut, we create a heap of these numbers.





2
6
8
v
4
5
7
2
5
7
updated weight
current weight
S
$\bar{S}$