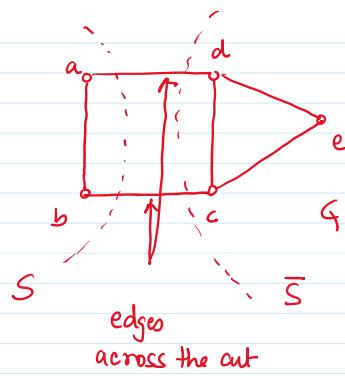


Cut in a graph A cut in  $G$  is a partition of vertex set into two parts

$$S = \{a, b\}$$

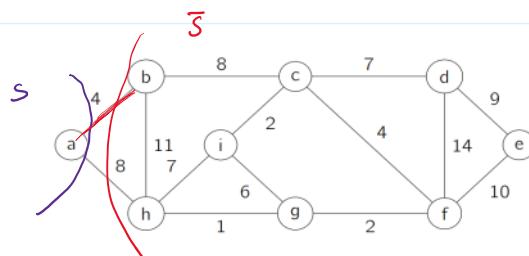
$$\bar{S} = \{b, c, e\}$$



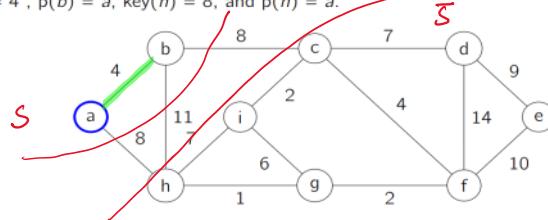
$$\# \text{ of possible cut } 2^n - 1 / 2^{n-2}$$

$$S = \{a\}$$

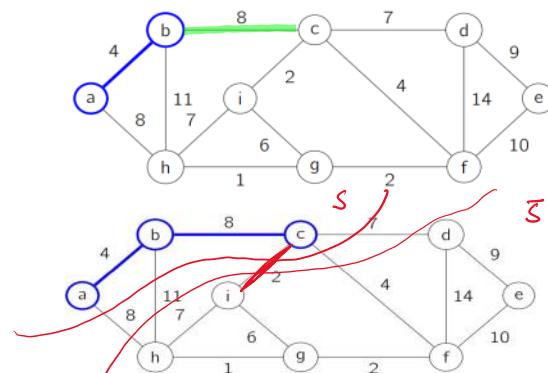
$$\bar{S} = V/S$$



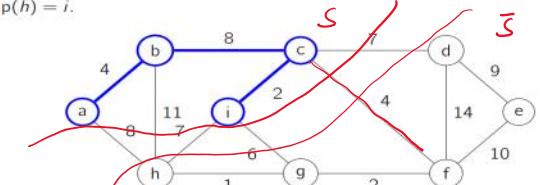
Suppose we select node  $a$  to be the source node,  $r$ . We then extract node  $a$  from  $Q$  and set  $\text{key}(b) = 4$ ,  $p(b) = a$ ,  $\text{key}(h) = 8$ , and  $p(h) = a$ .



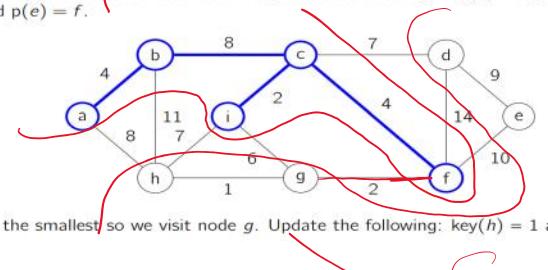
Since  $\text{key}(b)$  is now the smallest value in the priority queue, we visit node  $b$ . Because  $p(b) = a$  we add edge  $(a, b)$  to the set  $A$ . We then update the keys and parent fields of nodes that have edges connecting to  $b$ . Thus we set  $\text{key}(c) = 8$  and  $p(c) = b$ .



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

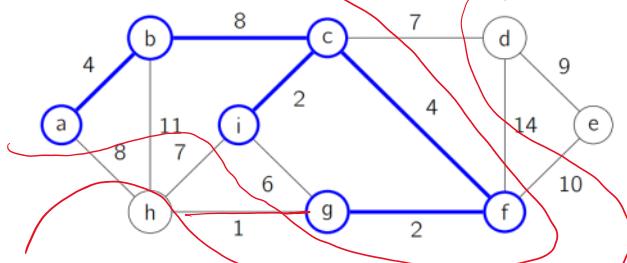


$\text{key}(f)$  is the smallest so we visit node  $f$ . Update the following:  $\text{key}(g) = 2$ ,  $p(g) = f$ ,  $\text{key}(e) = 10$ , and  $p(e) = f$ .

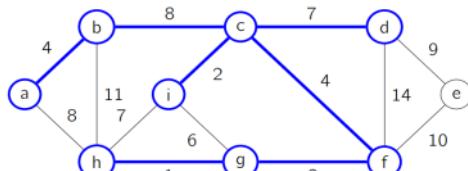


$\text{key}(g)$  is the smallest so we visit node  $g$ . Update the following:  $\text{key}(h) = 1$  and  $p(h) = g$ .

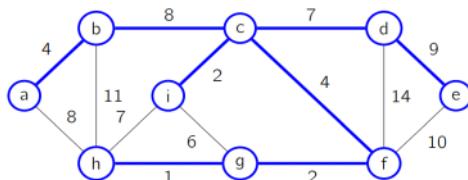
  
key( $g$ ) is the smallest so we visit node  $g$ . Update the following: key( $h$ ) = 1 and  $p(h) = g$ .



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

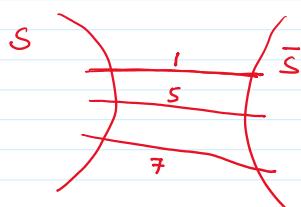


Finally, key( $e$ ) is the smallest so we visit node  $e$ . There are no updates at this step and the algorithm will detect that  $Q$  is empty at the next iteration and return.



claim For any cut  $(S, \bar{S})$ , the min weight edge in cut should belong to MST.  
 $\bar{S} := V/S$

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



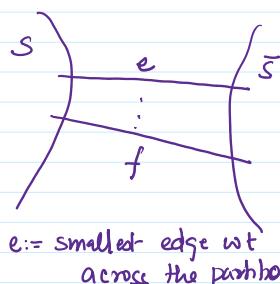
Proof: By contradiction. Consider a cut  $(S, \bar{S})$

$T :=$  MST that doesn't contain edge  $e$ .

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

Addition of  $e$  in  $T$  creates a cycle. (say  $C$ )

$C$  has to contain at least one more edge (say  $f$ ) across the partition.

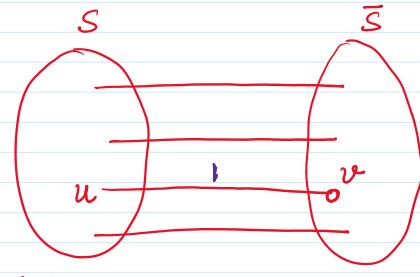


$e :=$  smaller edge wt across the partition

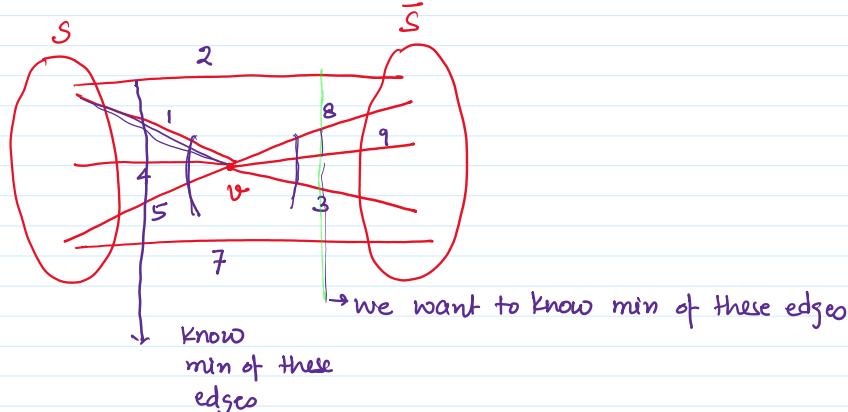
$\Rightarrow C$  contains an edge of weight more than weight of  $e$  between  $S$  &  $\bar{S}$ .

remove edge  $f$

$\Rightarrow$  Spanning tree of weight smaller than that of  $T$ . Contradiction!!



Let  $uv$  be the min wt edge



edges of weight 4,5  
need to remove from  
Heap

edges of wt 8,9,3 need to added in Heap

- At any point Heap maintains edges across the partition.
- For every vertex that we process, we remove some edges and add some edges in  
 $\uparrow$   
Heap.
- Overall time complexity =  $(\sum_v \deg(v)) \log m + n \cdot O(1) = O(n+m \log m)$

### Algorithm

$S$  is an array

$S[v] = 1$  if  $v \in S$   $\uparrow$  to check if  $v \in S$  or not  
&  $0 > 0.00$

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

$S[r] = 1$   $\uparrow$  root vertex

$T \leftarrow \emptyset$

$\forall e$  incident to  $r$  do

$H.\text{insert}(e)$

insert() inserting  $e$  into Heap  $H$

while Heap is not empty do

$f = H.\text{findmin}()$   $\uparrow f$  is the edge with min wt across cut  
 $T \leftarrow T \cup \{f\}$   
 Let  $v$  be the endpt of  $f$  &  $S[v] = 0$   
 for all edges  $e$  adjacent to  $v$  do  
 $= (v, w)$   
 $\uparrow$  if ( $S[w] == 0$ )

```

 $\deg(v) \rightarrow$  if ( $s[w] == 0$ )
    H.insert(e)
else
    H.delete(e)

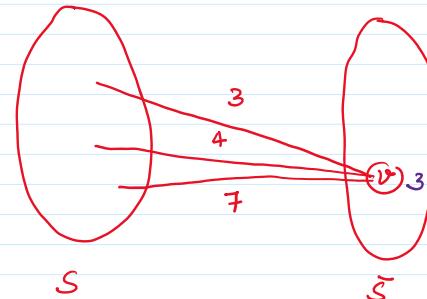
 $s[v] = 1$ 

```

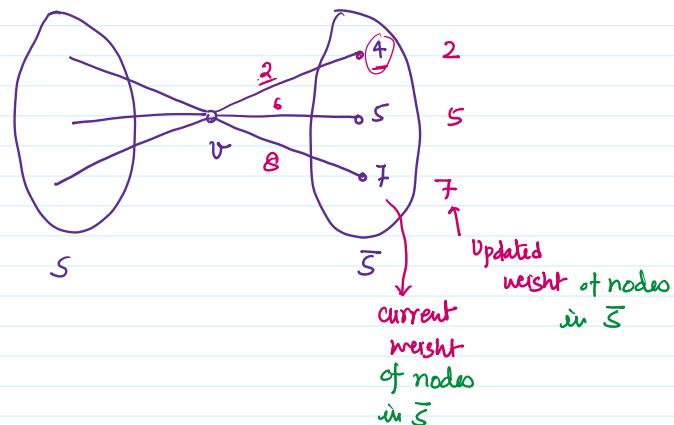
Modification For each vertex in  $\bar{S}$  store a label which is wt of min edge across the cut. We create a heap of these numbers

When we move a vertex  $v$  from  $\bar{S}$  to  $S$   
we update the weight of vertices in  $\bar{S}$

Heap is going to contain vertices of  $\bar{S}$ .



$+v, s[v]=0, H.insert(v, \infty)$   
 $s[r]=1, H.decreasePriority(r, 0)$



while Heap is not empty do

$v = H.findMin()$



for all  $w$  adjacent to  $v$  do

```

 $\deg(v) \rightarrow$  if ( $s[w] == 0$ ) then
    if  $label[w] > weight(v, w)$ 
        then  $label[w] = weight(v, w)$ 
        H.decreasePriority(w, label(w))

```

$\left. \begin{array}{l} \\ \\ \end{array} \right\} \quad \left. \begin{array}{l} L \quad H \cdot \text{decrease priority } (w, \text{label}(w)) \\ \text{use} \\ H \cdot \text{delete } (w, \text{label}(w)) \\ S[v] \leftarrow 1 \end{array} \right\}$

$\left. \begin{array}{l} * \quad H \cdot \text{insert } (v, t) := \text{insert node } v \text{ in heap with priority } w \\ H \cdot \text{delete } (w, \text{label}(w)) := \text{delete node } w \text{ from heap} \\ H \cdot \text{decrease priority } (w, \text{label}(w)) := \text{decrease priority of node } w \text{ to} \\ \qquad \qquad \qquad \text{label}(w) \end{array} \right\}$

In modified algo heap is created over node in  $S$ . Time complexity of insert  
delete from heap is  $O(\log n)$ .  $n$  # nodes in graph

whereas in previous algo it is created over edges in  
partition ( $\Rightarrow S$ ). Time complexity of insert/delete  $O(\log m) = O(\log n)$   $m < n^2$   
 $m :=$  # of edges in graph

Asymptotic time complexity of both algorithms are same.

Empirical time complexity of second is better than first one.