

# COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

## LECTURE 19

### GREEDY ALGORITHMS V: KRUSKAL'S ALGORITHM

SEPT 08, 2026

|

ROHIT VAISH

# PRIM'S ALGORITHM

**input:** connected undirected graph  $G = (V, E)$ , edge costs  $\{c_e\}_{e \in E}$

**output:** edges of minimum spanning tree of  $G$

// initialization

$X := \{s\}$  //  $s$  is an arbitrarily chosen vertex

$T := \emptyset$  // invariant: edges in  $T$  span  $X$

// main loop

while  $X \neq V$

let  $e = (u, v)$  be a minimum cost edge with  $u \in X$ ,  $v \notin X$

add  $e$  to  $T$

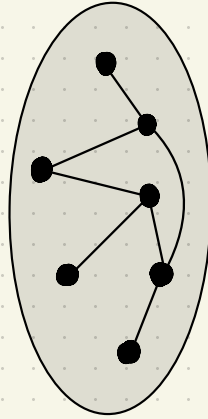
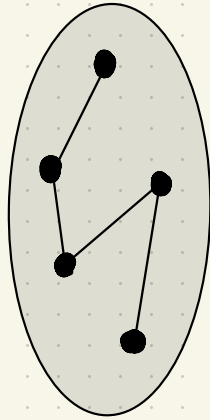
add  $v$  to  $X$

return  $T$



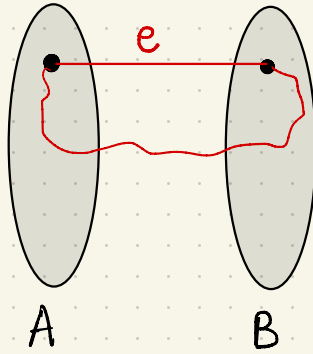
# EMPTY CUT LEMMA

**Lemma:** A graph is **not** connected if and only if there exists a cut with no crossing edges.



# DOUBLE CROSSING LEMMA

**Lemma:** Suppose the cycle  $C \subseteq E$  has an edge crossing the cut  $(A, B)$ .  
Then, there exists another edge of  $C$  that crosses  $(A, B)$ .



## DOUBLE CROSSING LEMMA

**Lemma:** Suppose the cycle  $C \subseteq E$  has an edge crossing the cut  $(A, B)$ .  
Then, there exists another edge of  $C$  that crosses  $(A, B)$ .

**Lonely cut corollary:** If  $e$  is the only edge crossing a cut  $(A, B)$ ,  
then  $e$  is not contained in any cycle.

# THE CUT PROPERTY

**Claim:** If an edge  $e$  is the cheapest edge crossing a cut  $(A, B)$ , then  $e$  belongs to **every** MST of graph  $G$ .

# CORRECTNESS OF PRIM'S ALGORITHM

**Theorem:** Prim's algorithm always computes an MST.

**Corollary:** Distinct edge costs  $\Rightarrow$  MST is unique.

# RUNNING TIME OF PRIM'S ALGORITHM

# RUNNING TIME OF PRIM'S ALGORITHM

// initialization

$X := \{s\}$       //  $s$  is an arbitrarily chosen vertex

$T := \emptyset$       // invariant : edges in  $T$  span  $X$

// main loop

while  $X \neq V$

    let  $e = (u, v)$  be a minimum cost edge with  $u \in X$ ,  $v \notin X$

    add  $e$  to  $T$

    add  $v$  to  $X$

return  $T$

# RUNNING TIME OF PRIM'S ALGORITHM

// initialization

$X := \{s\}$

//  $s$  is an arbitrarily chosen vertex

$T := \emptyset$

// invariant: edges in  $T$  span  $X$

// main loop

$O(n)$  iterations

while  $X \neq V$

let  $e = (u, v)$  be a minimum cost edge with  $u \in X$ ,  $v \notin X$

add  $e$  to  $T$

add  $v$  to  $X$

return  $T$



# RUNNING TIME OF PRIM'S ALGORITHM

// initialization

$X := \{s\}$  //  $s$  is an arbitrarily chosen vertex

$T := \emptyset$  // invariant: edges in  $T$  span  $X$

// main loop

while  $X \neq V$

$O(n)$  iterations

$O(m)$  time

let  $e = (u, v)$  be a minimum cost edge with  $u \in X, v \notin X$

add  $e$  to  $T$

add  $v$  to  $X$

return  $T$

# RUNNING TIME OF PRIM'S ALGORITHM

// initialization

$X := \{s\}$  //  $s$  is an arbitrarily chosen vertex

$T := \emptyset$  // invariant: edges in  $T$  span  $X$

// main loop

while  $X \neq V$

$O(n)$  iterations

$O(m)$  time

let  $e = (u, v)$  be a minimum cost edge with  $u \in X$ ,  $v \notin X$

add  $e$  to  $T$

add  $v$  to  $X$

return  $T$

$O(mn)$  time overall

# RUNNING TIME OF PRIM'S ALGORITHM

// initialization

$X := \{s\}$

//  $s$  is an arbitrarily chosen vertex

$T := \emptyset$

// invariant: edges in  $T$  span  $X$

// main loop

$O(n)$  iterations

~~$O(m)$~~  time use heap!

while  $X \neq V$

let  $e = (u, v)$  be a minimum cost edge with  $u \in X$ ,  $v \notin X$

add  $e$  to  $T$

add  $v$  to  $X$

return  $T$

$O(mn)$  time overall

# RUNNING TIME OF PRIM'S ALGORITHM

// initialization

$X := \{s\}$  //  $s$  is an arbitrarily chosen vertex

$T := \emptyset$  // invariant: edges in  $T$  span  $X$

// main loop

while  $X \neq V$

$O(n)$  iterations

~~$O(m)$~~  use heap!  
time

let  $e = (u, v)$  be a minimum cost edge with  $u \in X, v \notin X$

add  $e$  to  $T$

add  $v$  to  $X$

return  $T$

$O(m \log n)$

(see "Additional Reading")

~~$O(mn)$~~  time overall

# KRUSKAL'S ALGORITHM

# KRUSKAL'S ALGORITHM

Jarník



1930

Prim



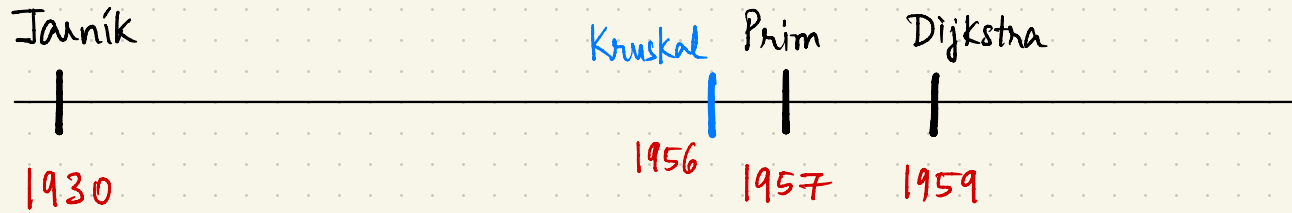
1957

Dijkstra

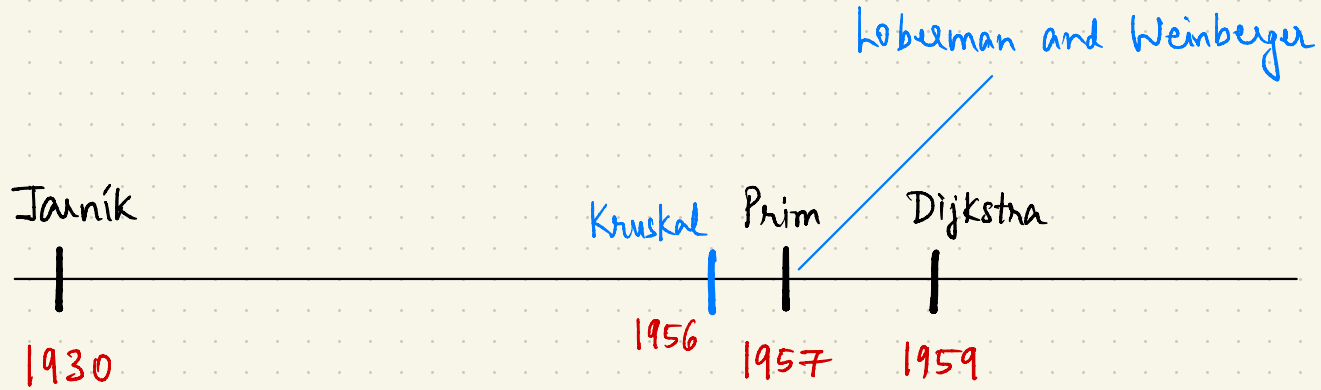


1959

# KRUSKAL'S ALGORITHM

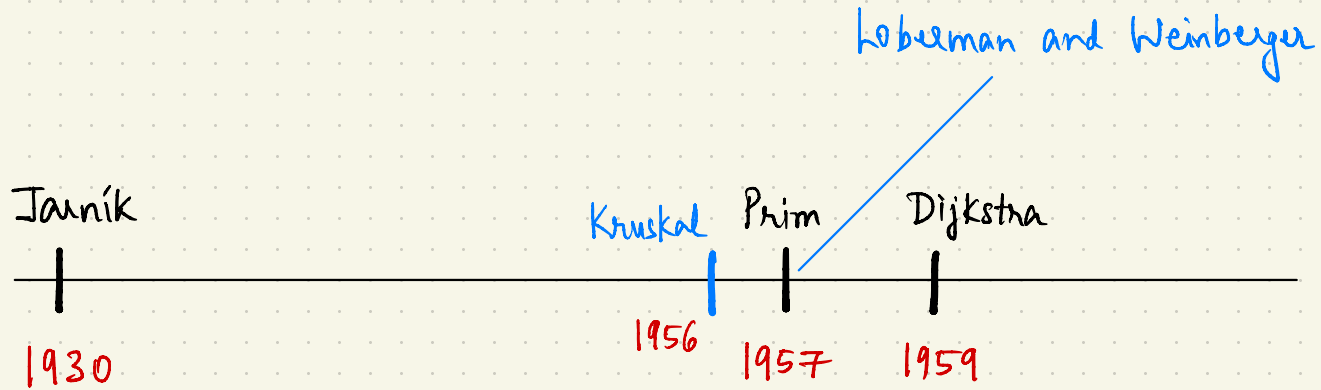


# KRUSKAL'S ALGORITHM



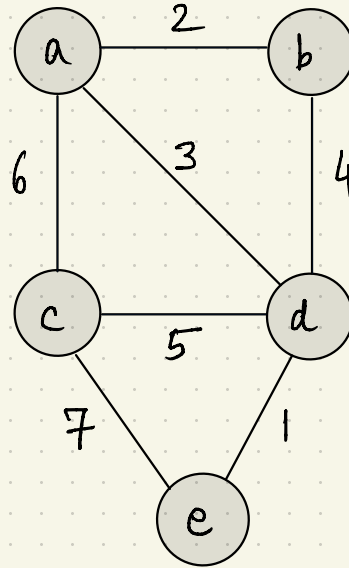


# KRUSKAL'S ALGORITHM

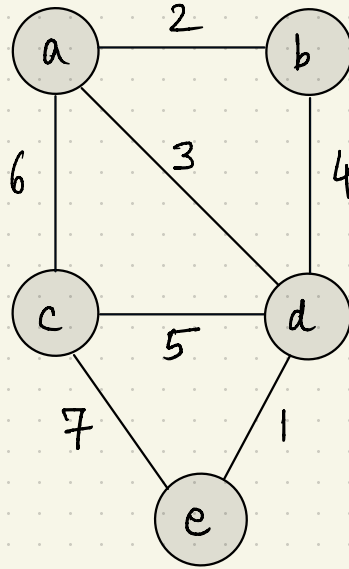


Application : Single-linkage clustering  
(Additional reading)

# KRUSKAL'S ALGORITHM

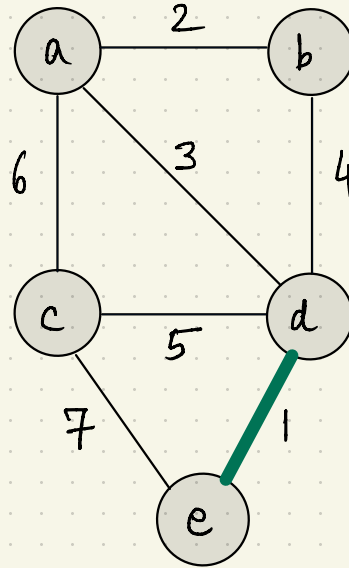


# KRUSKAL'S ALGORITHM

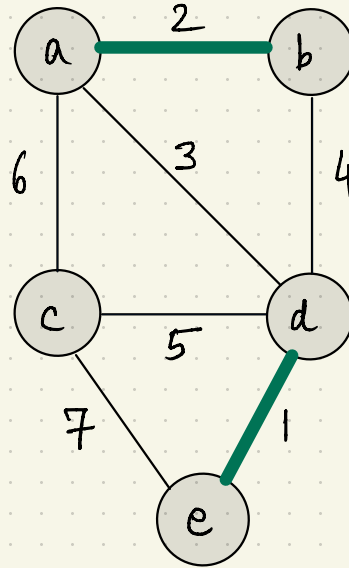


**Idea :** Pick the cheapest available edge without creating a cycle.

# KRUSKAL'S ALGORITHM

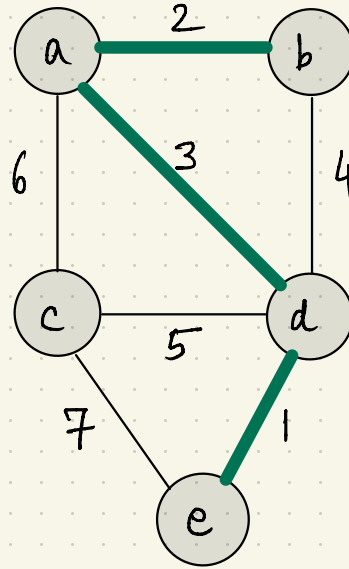


# KRUSKAL'S ALGORITHM

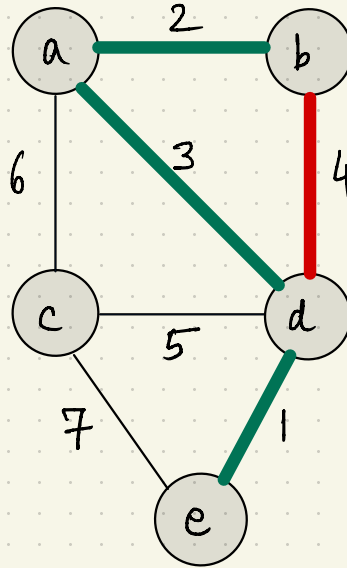


forest!

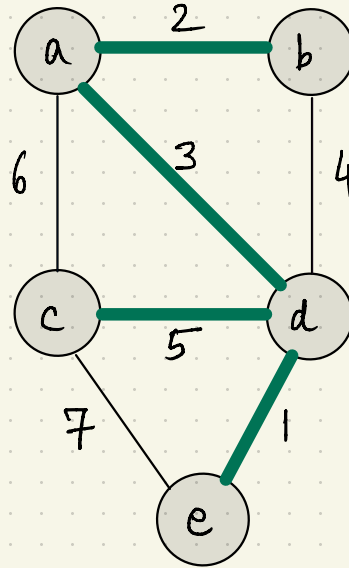
# KRUSKAL'S ALGORITHM



# KRUSKAL'S ALGORITHM

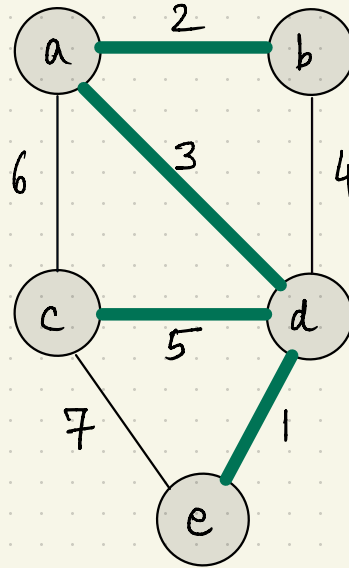


# KRUSKAL'S ALGORITHM





# KRUSKAL'S ALGORITHM



done!

# KRUSKAL'S ALGORITHM

input: connected undirected graph  $G = (V, E)$ , edge costs  $\{c_e\}_{e \in E}$

output: edges of minimum spanning tree of  $G$

# KRUSKAL'S ALGORITHM

input: connected undirected graph  $G = (V, E)$ , edge costs  $\{c_e\}_{e \in E}$

output: edges of minimum spanning tree of  $G$

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

# KRUSKAL'S ALGORITHM

input: connected undirected graph  $G = (V, E)$ , edge costs  $\{c_e\}_{e \in E}$

output: edges of minimum spanning tree of  $G$

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

// main loop

for each edge  $e \in E$  in increasing order of cost

# KRUSKAL'S ALGORITHM

**input:** connected undirected graph  $G = (V, E)$ , edge costs  $\{c_e\}_{e \in E}$

**output:** edges of minimum spanning tree of  $G$

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

// main loop

for each edge  $e \in E$  in increasing order of cost

    | if  $T \cup \{e\}$  is acyclic

    |     | add  $e$  to  $T$

# KRUSKAL'S ALGORITHM

**input:** connected undirected graph  $G = (V, E)$ , edge costs  $\{c_e\}_{e \in E}$

**output:** edges of minimum spanning tree of  $G$

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

// main loop

for each edge  $e \in E$  in increasing order of cost

    if  $T \cup \{e\}$  is acyclic

        add  $e$  to  $T$

return  $T$

# CORRECTNESS OF KRUSKAL'S ALGORITHM

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:**



# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** (I) Kruskal's algorithm computes a spanning tree  $T^*$

(II)  $T^*$  is an MST

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** (I) Kruskal's algorithm computes a spanning tree  $T^*$

- acyclic:
- spanning:
- connected:

(II)  $T^*$  is an MST

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** (I) Kruskal's algorithm computes a spanning tree  $T^*$

- acyclic : by design!
- spanning:
- connected:

(II)  $T^*$  is an MST

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** (I) Kruskal's algorithm computes a spanning tree  $T^*$

- acyclic : by design!
  - spanning!
  - connected!
- } coming up!

(II)  $T^*$  is an MST

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

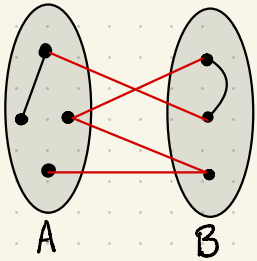
②  $T^*$  is **connected** :

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ .



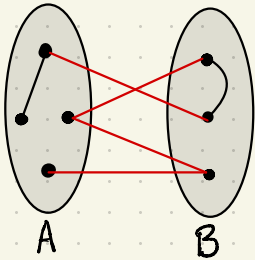


# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$

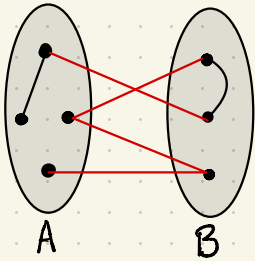


# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)



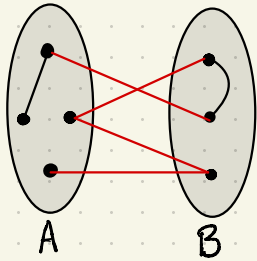
# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)

Kruskal's algorithm considers **every** edge of  $G$ .



# CORRECTNESS OF KRUSKAL'S ALGORITHM

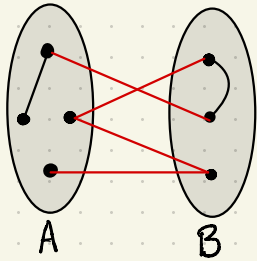
**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)

Kruskal's algorithm considers **every** edge of  $G$ .

Consider the **first** time Kruskal sees an edge  $e$  that crosses  $(A, B)$ .



# CORRECTNESS OF KRUSKAL'S ALGORITHM

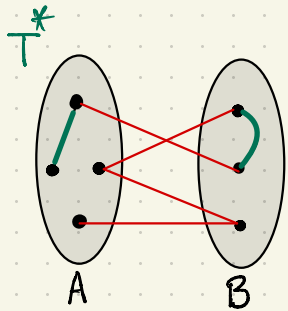
**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)

Kruskal's algorithm considers **every** edge of  $G$ .

Consider the **first** time Kruskal sees an edge  $e$  that crosses  $(A, B)$ .



# CORRECTNESS OF KRUSKAL'S ALGORITHM

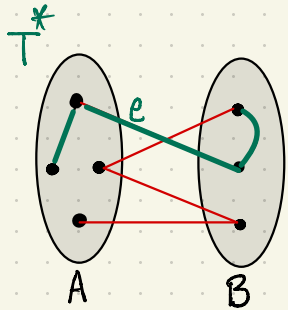
**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)

Kruskal's algorithm considers **every** edge of  $G$ .

Consider the **first** time Kruskal sees an edge  $e$  that crosses  $(A, B)$ .



# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

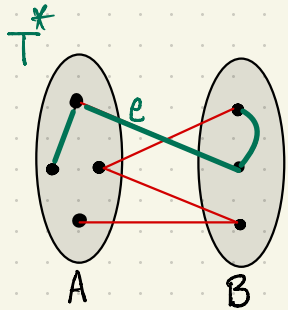
**Proof** : ①  $T^*$  is **cyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)

Kruskal's algorithm considers **every** edge of  $G$ .

Consider the **first** time Kruskal sees an edge  $e$  that crosses  $(A, B)$ .

Then,  $e$  is a "lonely" crossing edge wrt  $T^*$ .



# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

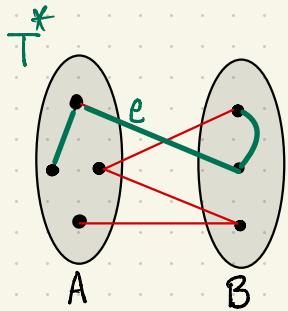
②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)

Kruskal's algorithm considers **every** edge of  $G$ .

Consider the **first** time Kruskal sees an edge  $e$  that crosses  $(A, B)$ .

Then,  $e$  is a "lonely" crossing edge wrt  $T^*$ .

$\Rightarrow e$  does not create a cycle in  $T^*$  and so Kruskal includes it.



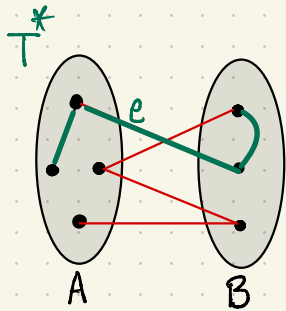


# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)



Kruskal's algorithm considers **every** edge of  $G$ .

Consider the **first** time Kruskal sees an edge  $e$  that crosses  $(A, B)$ .

Then,  $e$  is a "lonely" crossing edge wrt  $T^*$ .

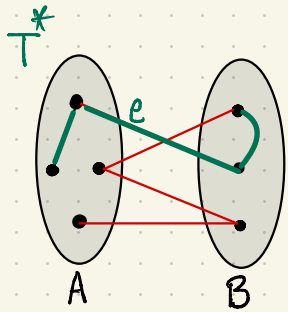
$\Rightarrow e$  does not create a cycle in  $T^*$  and so Kruskal includes it.  
Since cut  $(A, B)$  was chosen arbitrarily  $\Rightarrow$  no empty cuts  $\Rightarrow$  connected.

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **cyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)



Kruskal's algorithm considers **every** edge of  $G$ .

Consider the **first** time Kruskal sees an edge  $e$  that crosses  $(A, B)$ .

Then,  $e$  is a "lonely" crossing edge wrt  $T^*$ .

$\Rightarrow e$  does not create a cycle in  $T^*$  and so Kruskal includes it.

Since cut  $(A, B)$  was chosen arbitrarily  $\Rightarrow$  no empty cuts  $\Rightarrow$  connected.

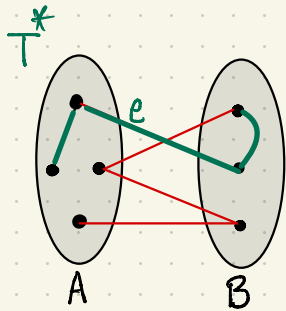
③  $T^*$  is **spanning** :

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Claim** : Kruskal's algorithm always computes a spanning tree.

**Proof** : ①  $T^*$  is **acyclic** : By design!

②  $T^*$  is **connected** : Consider any cut  $(A, B)$ . Empty cut lemma  $\Rightarrow$  at least one edge of  $G$  crosses this cut (since  $G$  is connected)



Kruskal's algorithm considers **every** edge of  $G$ .

Consider the **first** time Kruskal sees an edge  $e$  that crosses  $(A, B)$ .

Then,  $e$  is a "lonely" crossing edge wrt  $T^*$ .

$\Rightarrow e$  does not create a cycle in  $T^*$  and so Kruskal includes it. Since cut  $(A, B)$  was chosen arbitrarily  $\Rightarrow$  no empty cuts  $\Rightarrow$  connected.

③  $T^*$  is **spanning** : If not, consider the cut  $\left( \begin{array}{l} \text{included} \\ \text{in } T^* \end{array}, \begin{array}{l} \text{excluded} \\ \text{from } T^* \end{array} \right)$ . □

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** (I) Kruskal's algorithm computes a spanning tree  $T^*$

- acyclic
- spanning
- connected

(II)  $T^*$  is an MST

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** (I) Kruskal's algorithm computes a spanning tree  $T^*$

- acyclic
- spanning
- connected

(II)  $T^*$  is an MST

Using cut property.

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .

# CORRECTNESS OF KRUSKAL'S ALGORITHM

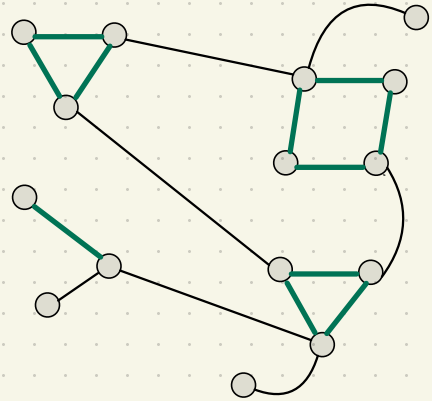
**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .  
Consider the iteration where edge  $(u, v)$  is added to current set  $T$ .

# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .  
Consider the iteration where edge  $(u,v)$  is added to current set  $T$ .

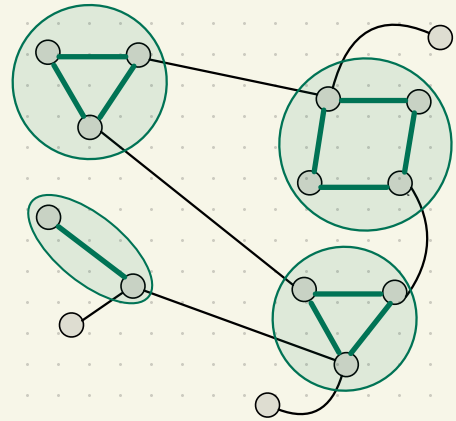




# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .  
Consider the iteration where edge  $(u,v)$  is added to current set  $T$ .



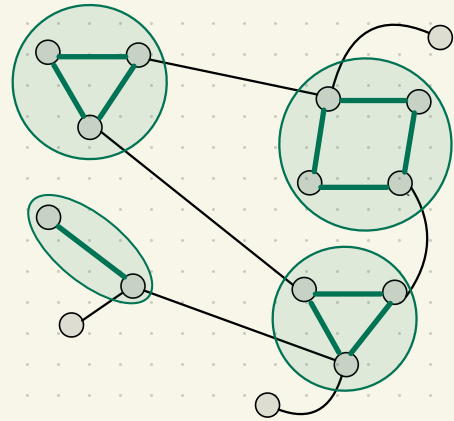
# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .

Consider the iteration where edge  $(u,v)$  is added to current set  $T$ .

Since  $T \cup \{(u,v)\}$  has no cycle,  $T$  has no  $(u,v)$  path.



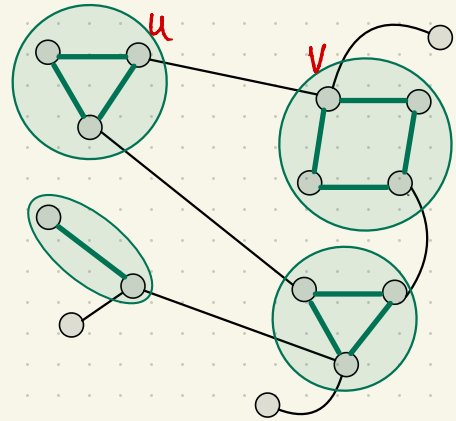
# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .

Consider the iteration where edge  $(u,v)$  is added to current set  $T$ .

Since  $T \cup \{(u,v)\}$  has no cycle,  $T$  has no  $(u,v)$  path.



# CORRECTNESS OF KRUSKAL'S ALGORITHM

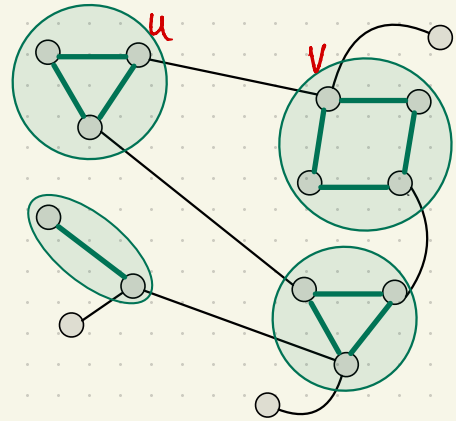
**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .

Consider the iteration where edge  $(u,v)$  is added to current set  $T$ .

Since  $T \cup \{(u,v)\}$  has no cycle,  $T$  has no  $(u,v)$  path.

$\Rightarrow$  before adding  $(u,v)$ ,  $\exists$  empty cut  $(A,B)$  separating  $u$  and  $v$   
s.t.  $u \in A, v \in B$  (by empty cut lemma)



# CORRECTNESS OF KRUSKAL'S ALGORITHM

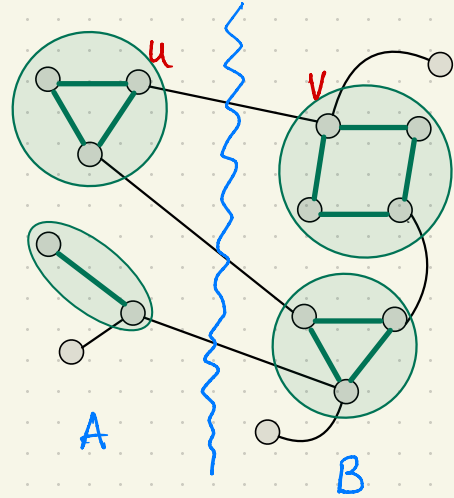
**Theorem:** Kruskal's algorithm always computes an MST.

**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .

Consider the iteration where edge  $(u,v)$  is added to current set  $T$ .

Since  $T \cup \{(u,v)\}$  has no cycle,  $T$  has no  $(u,v)$  path.

$\Rightarrow$  before adding  $(u,v)$ ,  $\exists$  empty cut  $(A,B)$  separating  $u$  and  $v$   
s.t.  $u \in A, v \in B$  (by empty cut lemma)



# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

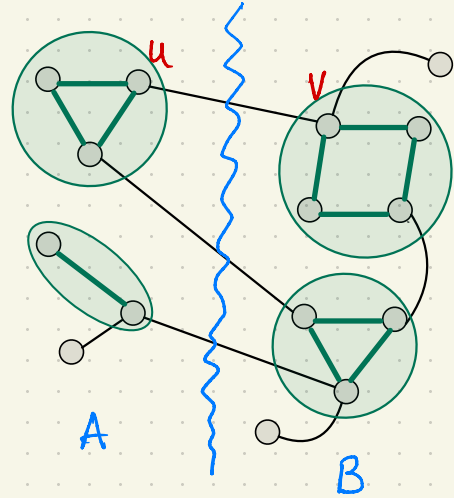
**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .

Consider the iteration where edge  $(u,v)$  is added to current set  $T$ .

Since  $T \cup \{(u,v)\}$  has no cycle,  $T$  has no  $(u,v)$  path.

$\Rightarrow$  before adding  $(u,v)$ ,  $\exists$  empty cut  $(A,B)$  separating  $u$  and  $v$  s.t.  $u \in A, v \in B$  (by empty cut lemma)

💡  $(u,v)$  is the **first**, and therefore **cheapest**, edge crossing the cut  $(A,B)$ .



# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

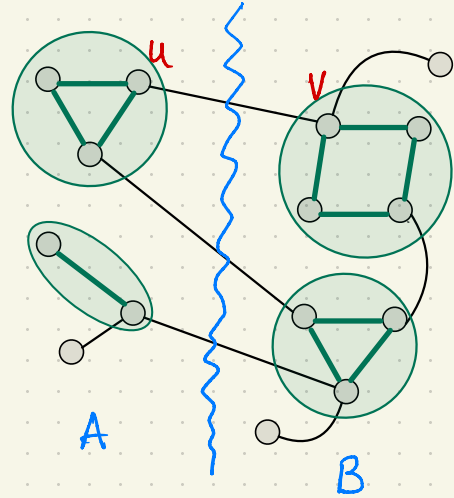
**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .

Consider the iteration where edge  $(u,v)$  is added to current set  $T$ .

Since  $T \cup \{(u,v)\}$  has no cycle,  $T$  has no  $(u,v)$  path.

$\Rightarrow$  before adding  $(u,v)$ ,  $\exists$  empty cut  $(A,B)$  separating  $u$  and  $v$   
s.t.  $u \in A, v \in B$  (by empty cut lemma)

💡  $(u,v)$  is the **first**, and therefore **cheapest**, edge crossing the cut  $(A,B)$ .  
[first: lonely edge corollary]  
[cheapest: greedy criterion]



# CORRECTNESS OF KRUSKAL'S ALGORITHM

**Theorem:** Kruskal's algorithm always computes an MST.

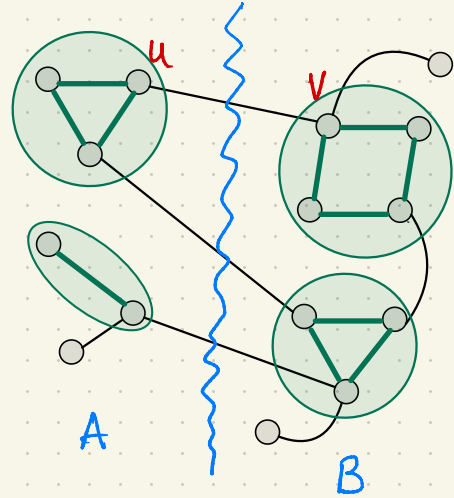
**Proof:** We know Kruskal's algorithm computes a spanning tree, say  $T^*$ .

Consider the iteration where edge  $(u,v)$  is added to current set  $T$ .

Since  $T \cup \{(u,v)\}$  has no cycle,  $T$  has no  $(u,v)$  path.

$\Rightarrow$  before adding  $(u,v)$ ,  $\exists$  empty cut  $(A,B)$  separating  $u$  and  $v$  s.t.  $u \in A, v \in B$  (by empty cut lemma)

💡  $(u,v)$  is the **first**, and therefore **cheapest**, edge crossing the cut  $(A,B)$ . [first: lonely edge corollary]  
[cheapest: greedy criterion]



$\Rightarrow (u,v)$  is justified by cut property.





# RUNNING TIME OF KRUSKAL'S ALGORITHM

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

// main loop

for each edge  $e \in E$  in increasing order of cost

    | if  $T \cup \{e\}$  is acyclic

        | add  $e$  to  $T$

return  $T$

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

$O(m \log m)$

// main loop

for each edge  $e \in E$  in increasing order of cost

    if  $T \cup \{e\}$  is acyclic  
        add  $e$  to  $T$

return  $T$

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

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

// main loop

for each edge  $e \in E$  in increasing order of cost

    if  $T \cup \{e\}$  is acyclic  
        add  $e$  to  $T$

return  $T$

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

$O(m \log n)$

// main loop

for each edge  $e \in E$  in increasing order of cost

| if  $T \cup \{e\}$  is acyclic

| | add  $e$  to  $T$

return  $T$

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

$O(m \log n)$

// main loop

for each edge  $e \in E$  in increasing order of cost

$O(m)$

    if  $T \cup \{e\}$  is acyclic

        add  $e$  to  $T$

return  $T$

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

$O(m \log n)$

// main loop

for each edge  $e \in E$  in increasing order of cost

$O(m)$

| if  $T \cup \{e\}$  is acyclic ————— ?  
| | add  $e$  to  $T$

return  $T$

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

$O(m \log n)$

// main loop

for each edge  $e \in E$  in increasing order of cost

$O(m)$

| if  $T \cup \{e\}$  is acyclic

$O(n)$

| | add  $e$  to  $T$

BFS for edges in  $T$

return  $T$



# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

$O(m \log n)$

// main loop

for each edge  $e \in E$  in increasing order of cost

$O(m)$

| if  $T \cup \{e\}$  is acyclic —————  $O(n)$

| | add  $e$  to  $T$

return  $T$

$O(mn)$  overall

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

$O(m \log n)$

// main loop

for each edge  $e \in E$  in increasing order of cost

$O(m)$

    if  $T \cup \{e\}$  is acyclic

~~$O(n)$~~

$O(1)$  via  
union-find

        add  $e$  to  $T$

return  $T$

$O(mn)$  overall

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

$O(m \log n)$

// main loop

for each edge  $e \in E$  in increasing order of cost

$O(m)$

| if  $T \cup \{e\}$  is acyclic

~~$O(n)$~~

$O(1)$  via  
union-find

| | add  $e$  to  $T$

return  $T$

$O(m \log n)$

~~$O(mn)$~~  overall

# RUNNING TIME OF KRUSKAL'S ALGORITHM

// initialization

$T := \emptyset$

Sort edges of  $E$  by cost

$O(m \log n)$

// main loop

for each edge  $e \in E$  in increasing order of cost

$O(m)$

| if  $T \cup \{e\}$  is acyclic

~~$O(n)$~~

$O(1)$  via  
union-find

| | add  $e$  to  $T$

return  $T$

$O(m \log n)$

~~$O(mn)$~~  overall

Additional reading