

COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

LECTURE 18

GREEDY ALGORITHMS IV: MINIMUM SPANNING TREES (CONTD.)

SEPT 03, 2026

|

ROHIT VAISH

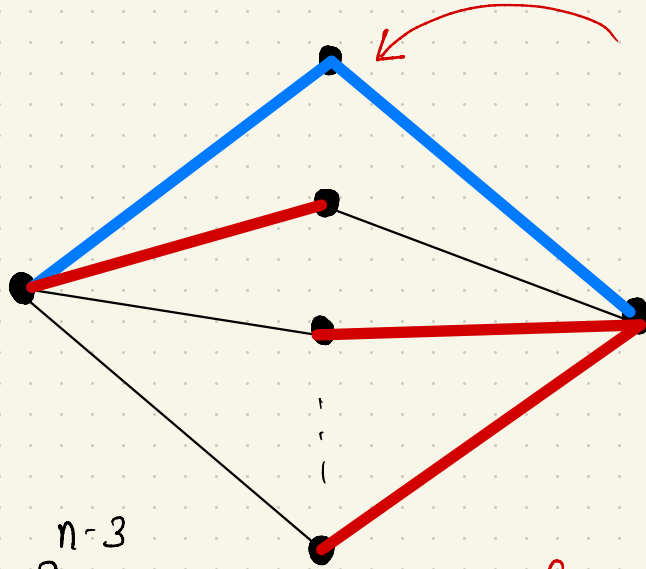
THE MINIMUM SPANNING TREE PROBLEM

THE MINIMUM SPANNING TREE PROBLEM

input: an undirected graph $G = (V, E)$ with real-valued edge costs $\{c_e\}_{e \in E}$

output: a spanning tree $T \subseteq E$ of G with minimum possible sum $\sum_{e \in T} c_e$ of edge costs.

EXPONENTIALLY MANY SPANNING TREES



Some middle node
must have both edges
in spanning tree

spanning trees $\geq 2^{n-3}$

Cannot hope to find an MST
via brute force

for every other node,
pick either left or right edge

ASSUMPTIONS

1. Input graph G is **connected**.

- else no spanning tree
- easy to check connectedness (e.g., via BFS or DFS)

2. Edge costs are **distinct**.

- cleaner proofs of correctness
- algorithms (Prim's, Kruskal's) correct without this assumption

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

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

(II) T^* is an MST

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

Proof: (I) Prim's algorithm computes a **spanning tree** T^*
- use basic properties of graphs and spanning trees

(II) T^* is an **MST**

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

- use basic properties of graphs and spanning trees

(II) T^* is an **MST**

- use **cut property**

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

- use basic properties of graphs and spanning trees

(II) T^* is an MST

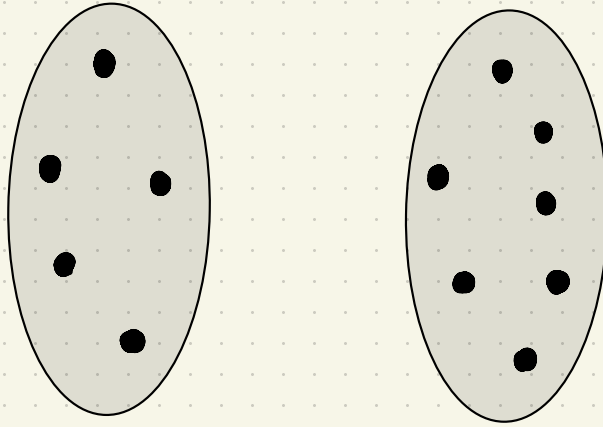
- use cut property

CUTS

A **cut** of a graph $G=(V,E)$ is a partition of the vertex set V into two non empty sets.

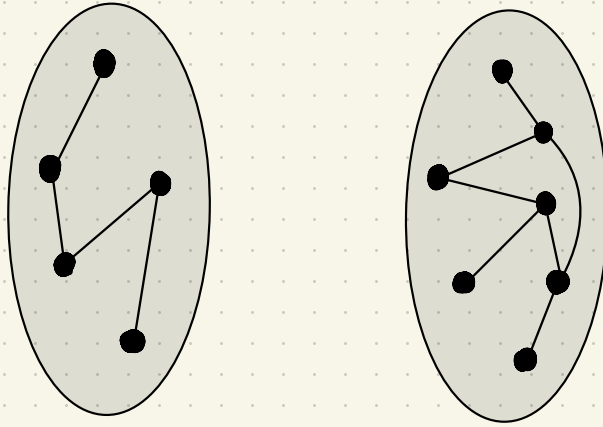
CUTS

A **cut** of a graph $G=(V,E)$ is a partition of the vertex set V into two non empty sets.



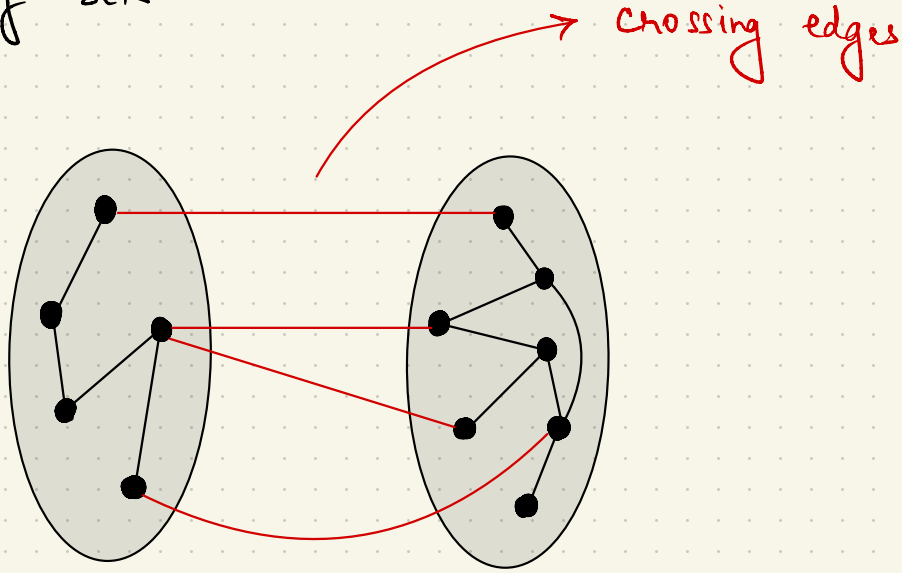
CUTS

A **cut** of a graph $G=(V,E)$ is a partition of the vertex set V into two non empty sets.



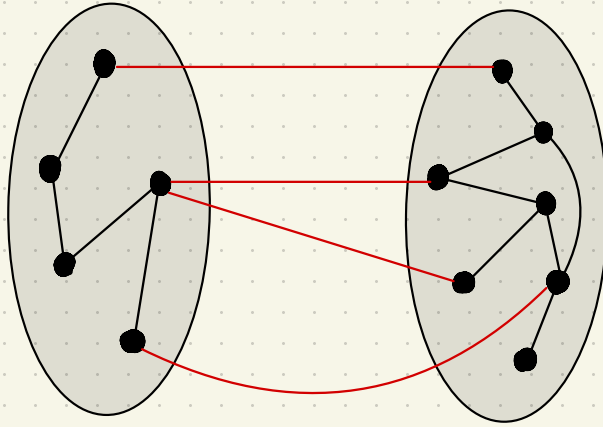
CUTS

A **cut** of a graph $G=(V,E)$ is a partition of the vertex set V into two non empty sets.



CUTS

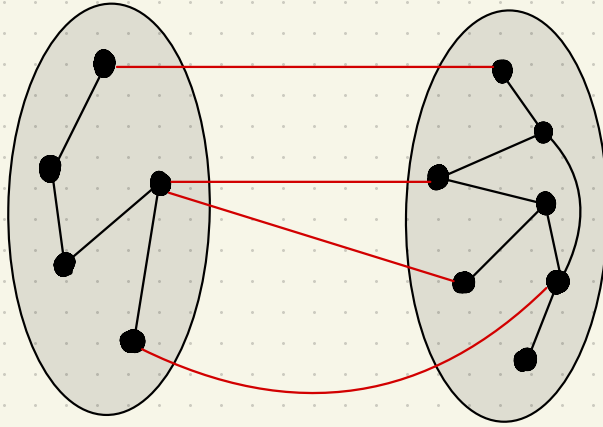
A **cut** of a graph $G=(V,E)$ is a partition of the vertex set V into two non empty sets.



How many cuts does a graph with n vertices have?

CUTS

A **cut** of a graph $G=(V,E)$ is a partition of the vertex set V into two non empty sets.



How many cuts does a graph with n vertices have? $2^{n-1} - 1$

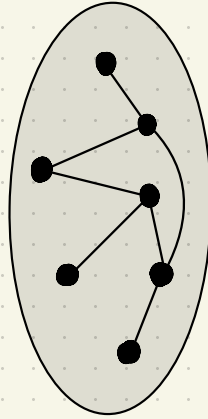
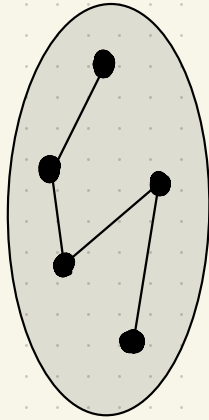
3 easy facts about cuts

EMPTY CUT LEMMA

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

EMPTY CUT LEMMA

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



EMPTY CUT LEMMA

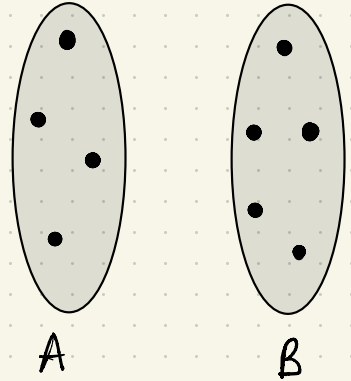
Lemma: A graph is **not** connected if and only if there exists a cut with no crossing edges.

Proof: ($\Leftarrow$)

EMPTY CUT LEMMA

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

Proof: ($\Leftarrow$) Say cut (A, B) has no crossing edges.

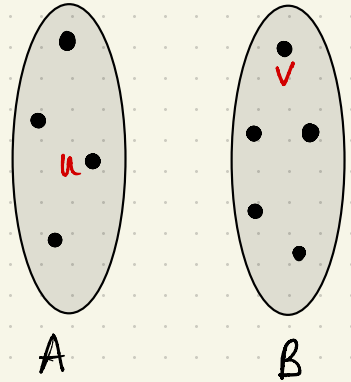


EMPTY CUT LEMMA

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

Proof: ($\Leftarrow$) Say cut (A, B) has no crossing edges.

Pick any $u \in A$ and $v \in B$.

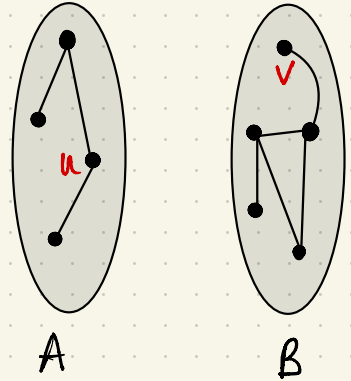


EMPTY CUT LEMMA

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

Proof: ($\Leftarrow$) Say cut (A, B) has no crossing edges.

Pick any $u \in A$ and $v \in B$.



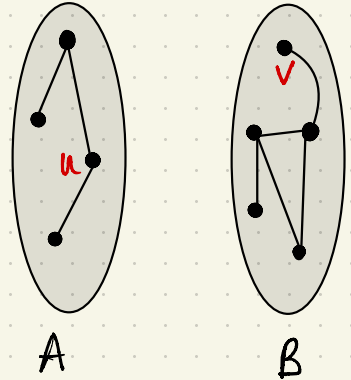
EMPTY CUT LEMMA

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

Proof: ($\Leftarrow$) Say cut (A, B) has no crossing edges.

Pick any $u \in A$ and $v \in B$.

Any $u \rightsquigarrow v$ path must have some edge crossing the cut (A, B) .



EMPTY CUT LEMMA

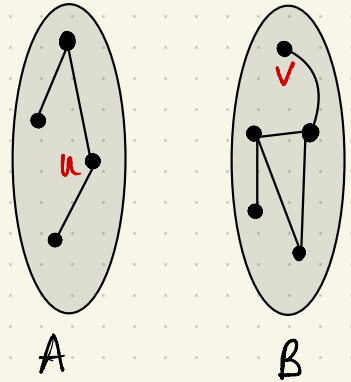
Lemma: A graph is **not** connected if and only if there exists a cut with no crossing edges.

Proof: ($\Leftarrow$) Say cut (A, B) has no crossing edges.

Pick any $u \in A$ and $v \in B$.

Any $u \rightsquigarrow v$ path must have some edge crossing the cut (A, B) .

Since (A, B) has no crossing edges, there cannot be a $u \rightsquigarrow v$ path $\Rightarrow$ graph is not connected.



EMPTY CUT LEMMA

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

Proof: ($\Rightarrow$)

EMPTY CUT LEMMA

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

Proof: ($\Rightarrow$) Graph not connected $\Rightarrow$ no path between some pair of vertices u and v .

EMPTY CUT LEMMA

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

Proof: $(\Rightarrow)$ Graph not connected $\Rightarrow$ no path between some pair of vertices u and v .

Define $A := \{\text{vertices reachable from } u \text{ in the graph}\}$.

$B := \{\text{all other vertices}\}$

EMPTY CUT LEMMA

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

Proof: $(\Rightarrow)$ Graph not connected $\Rightarrow$ no path between some pair of vertices u and v .

Define $A := \{\text{vertices reachable from } u \text{ in the graph}\}$.

$\underbrace{\hspace{10em}}$ u 's connected component

$B := \{\text{all other vertices}\}$

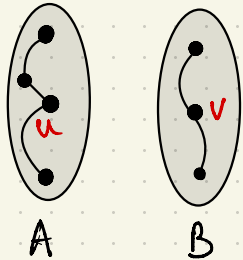
$\underbrace{\hspace{10em}}$ other connected components

EMPTY CUT LEMMA

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

Proof: $(\Rightarrow)$ Graph not connected $\Rightarrow$ no path between some pair of vertices u and v .

Define $A := \{\text{vertices reachable from } u \text{ in the graph}\}$.



$B := \{\text{all other vertices}\}$

u 's connected component

other connected components

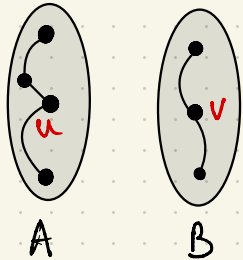
Note: $u \in A$ and $v \in B \Rightarrow (A, B)$ is a cut.

EMPTY CUT LEMMA

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

Proof: $(\Rightarrow)$ Graph not connected $\Rightarrow$ no path between some pair of vertices u and v .

Define $A := \{\text{vertices reachable from } u \text{ in the graph}\}$.



$B := \{\text{all other vertices}\}$

u 's connected component

other connected components

Note: $u \in A$ and $v \in B \Rightarrow (A, B)$ is a cut.

Observe: No edges crossing (A, B) otherwise A would be bigger.



EMPTY CUT LEMMA

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

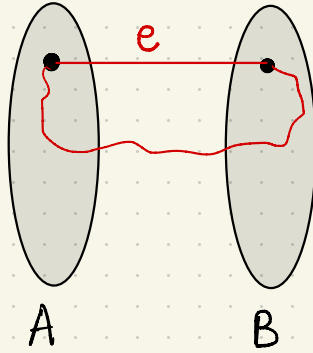
So,

a graph is connected $\iff$ no empty cuts.

DOUBLE CROSSING LEMMA

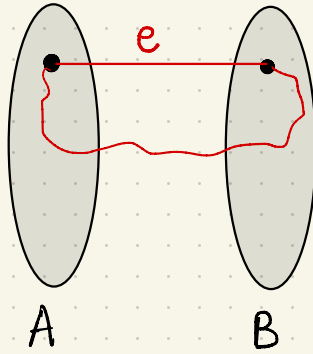
DOUBLE CROSSING LEMMA

Lemma: Suppose the cycle $C \subseteq E$ has an edge crossing the cut (A, B) .



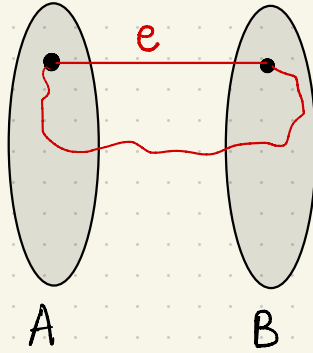
DOUBLE CROSSING LEMMA

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) .



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) .



Proof: Exercise

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.

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

(II) T^* is an MST

CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

Proof : ① Algorithm maintains the invariant that T spans X .

// 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

CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

Proof : ① Algorithm maintains the invariant that T spans X . (check!)

CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

Proof : ① Algorithm maintains the invariant that T spans X . (check!)

② ["spanning"] Algorithm can't get stuck with $X \neq V$

CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

Proof : ① Algorithm maintains the invariant that T spans X . (check!)

② ["spanning"] Algorithm can't get stuck with $X \neq V$
otherwise the cut $(X, V \setminus X)$ must be empty
Empty cut lemma $\Rightarrow$ graph is disconnected.

CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

Proof : ① Algorithm maintains the invariant that T spans X . (check!)

② ["spanning"] Algorithm can't get stuck with $X \neq V$
otherwise the cut $(X, V \setminus X)$ must be empty
Empty cut lemma $\Rightarrow$ graph is disconnected.

③ ["acyclic"] No cycle ever gets created in T .

CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

Proof : ① Algorithm maintains the invariant that T spans X . (check!)

② ["spanning"] Algorithm can't get stuck with $X \neq V$
otherwise the cut $(X, V \setminus X)$ must be empty

Empty cut lemma $\Rightarrow$ graph is disconnected.

③ ["acyclic"] No cycle ever gets created in T .

Consider any iteration. Say an edge e gets added to T .

CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

Proof : ① Algorithm maintains the invariant that T spans X . (check!)

② ["spanning"] Algorithm can't get stuck with $X \neq V$
otherwise the cut $(X, V \setminus X)$ must be empty

Empty cut lemma $\Rightarrow$ graph is disconnected.

③ ["acyclic"] No cycle ever gets created in T .

Consider any iteration. Say an edge e gets added to T .

Then, e is the **first** edge crossing the cut $(X, V \setminus X)$ that gets added.

CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

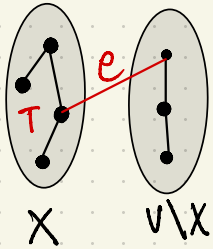
Proof : ① Algorithm maintains the invariant that T spans X . (check!)

② ["spanning"] Algorithm can't get stuck with $X \neq V$
otherwise the cut $(X, V \setminus X)$ must be empty
Empty cut lemma $\Rightarrow$ graph is disconnected.

③ ["acyclic"] No cycle ever gets created in T .

Consider any iteration. Say an edge e gets added to T .

Then, e is the **first** edge crossing the cut $(X, V \setminus X)$ that gets added.



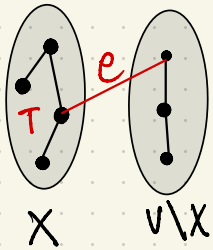
CORRECTNESS OF PRIM'S ALGORITHM

Claim : Prim's algorithm always computes a spanning tree.

Proof : ① Algorithm maintains the invariant that T spans X . (check!)

② ["spanning"] Algorithm can't get stuck with $X \neq V$
otherwise the cut $(X, V \setminus X)$ must be empty
Empty cut lemma $\Rightarrow$ graph is disconnected.

③ ["acyclic"] No cycle ever gets created in T .



Consider any iteration. Say an edge e gets added to T .

Then, e is the **first** edge crossing the cut $(X, V \setminus X)$ that gets added.
By lonely cut corollary, e cannot participate in a cycle.

CORRECTNESS OF PRIM'S ALGORITHM

Claim: Prim's algorithm always computes a spanning tree.

Proof: ① Algorithm maintains the invariant that T spans X . (check!)

② ["spanning"] Algorithm can't get stuck with $X \neq V$
otherwise the cut $(X, V \setminus X)$ must be empty
Empty cut lemma $\Rightarrow$ graph is disconnected.

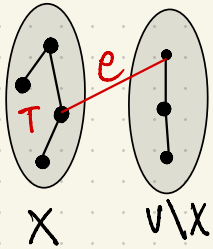
③ ["acyclic"] No cycle ever gets created in T .

Consider any iteration. Say an edge e gets added to T .

Then, e is the **first** edge crossing the cut $(X, V \setminus X)$ that gets added.

By lonely cut corollary, e cannot participate in a cycle.

④ ["connected"] T is connected (check!)



CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

(II) T^* is an MST

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

(II) T^* is an **MST**



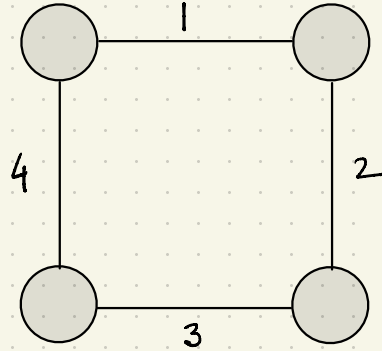
When is it **safe** to add an edge? **Cut property!**

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 .

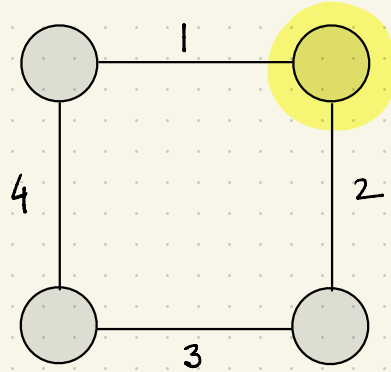
THE CUT PROPERTY

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



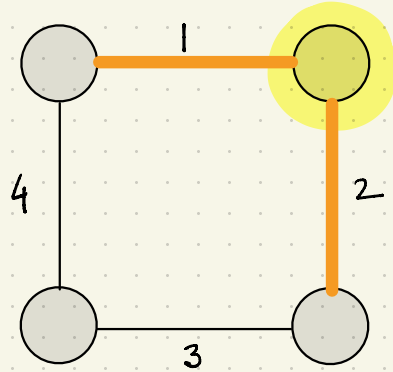
THE CUT PROPERTY

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



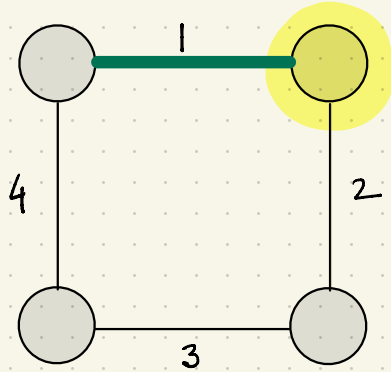
THE CUT PROPERTY

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



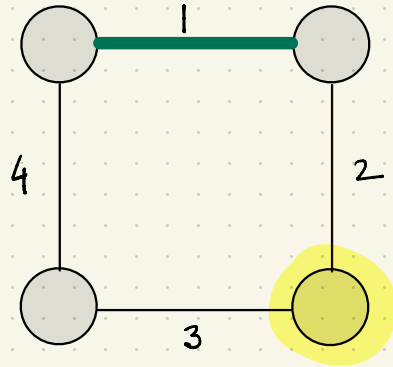
THE CUT PROPERTY

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



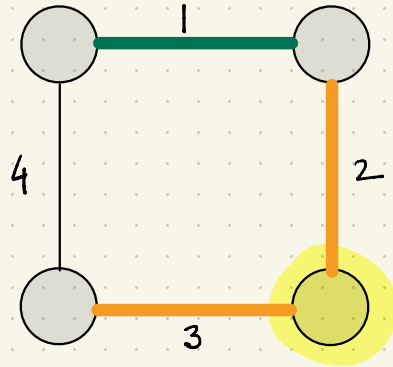
THE CUT PROPERTY

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



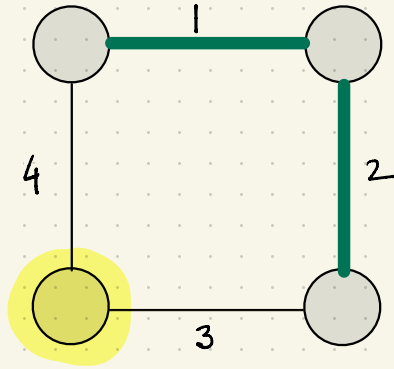
THE CUT PROPERTY

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



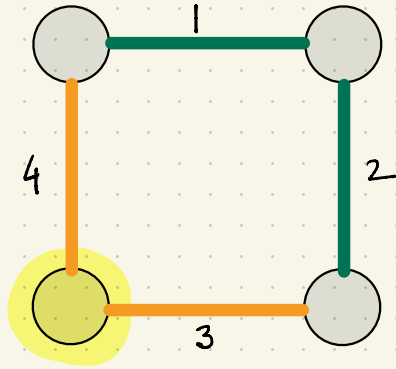
THE CUT PROPERTY

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



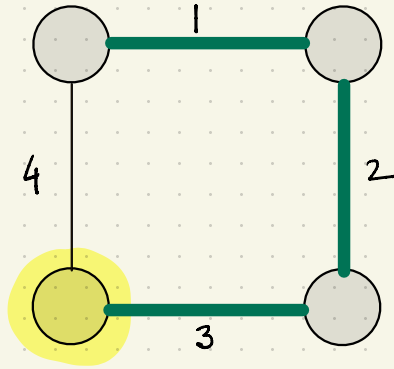
THE CUT PROPERTY

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



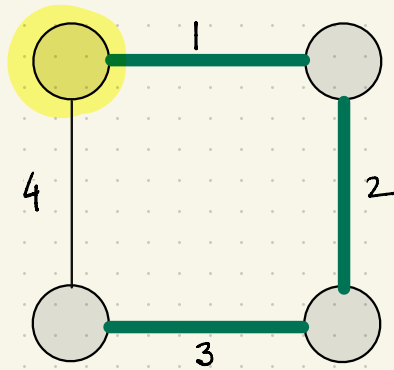
THE CUT PROPERTY

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



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 .



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 .

Proof: Tutorial sheet 7

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

 Every edge $e \in T^*$ selected by Prim's algorithm is explicitly justified by the cut property.

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

 Every edge $e \in T^*$ selected by Prim's algorithm is explicitly justified by the cut property.

$\Rightarrow$ every edge of T^* belongs to every MST

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

 Every edge $e \in T^*$ selected by Prim's algorithm is explicitly justified by the cut property.

$\Rightarrow$ every edge of T^* belongs to every MST

$\Rightarrow T^*$ is a subset of every MST.

CORRECTNESS OF PRIM'S ALGORITHM

Theorem: Prim's algorithm always computes an MST.

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

💡 Every edge $e \in T^*$ selected by Prim's algorithm is explicitly justified by the cut property.

$\Rightarrow$ every edge of T^* belongs to every MST

$\Rightarrow T^*$ is a subset of every MST.

$\Rightarrow T^*$ is the **unique** MST.

