

COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

LECTURE 17

GREEDY ALGORITHMS III: MINIMUM SPANNING TREES

SEPT 02, 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 $\{c_e\}_{e \in E}$

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.

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.

Spanning tree T of graph G :

"tree": connected and acyclic

"spanning": the subgraph (V, T) is connected

THE MINIMUM SPANNING TREE PROBLEM

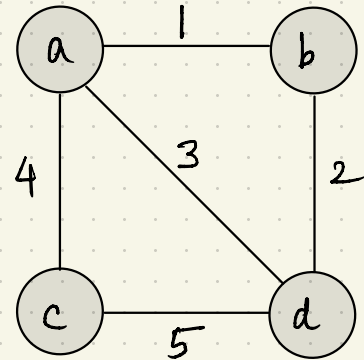
input: an undirected graph $G = (V, E)$ with real-valued edge costs $\{c_e\}_{e \in E}$

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

Spanning tree T of graph G :

"tree": connected and acyclic

"spanning": the subgraph (V, T) is connected



THE MINIMUM SPANNING TREE PROBLEM

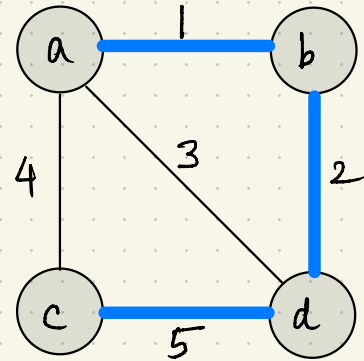
input: an undirected graph $G = (V, E)$ with real-valued edge costs $\{c_e\}_{e \in E}$

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

Spanning tree T of graph G :

"tree": connected and acyclic

"spanning": the subgraph (V, T) is connected



spanning tree
but not MST

THE MINIMUM SPANNING TREE PROBLEM

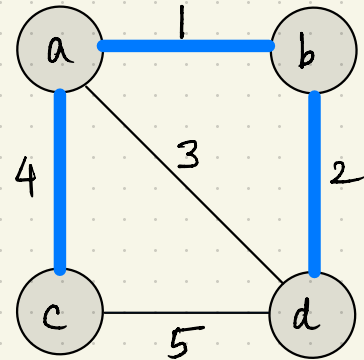
input: an undirected graph $G = (V, E)$ with real-valued edge costs $\{c_e\}_{e \in E}$

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

Spanning tree T of graph G :

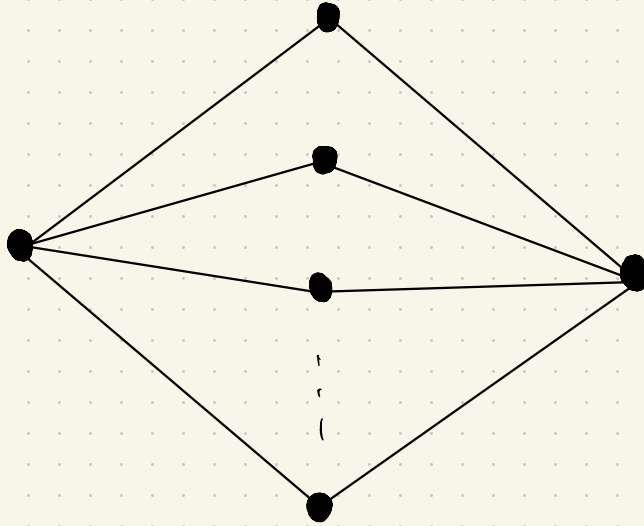
"tree": connected and acyclic

"spanning": the subgraph (V, T) is connected

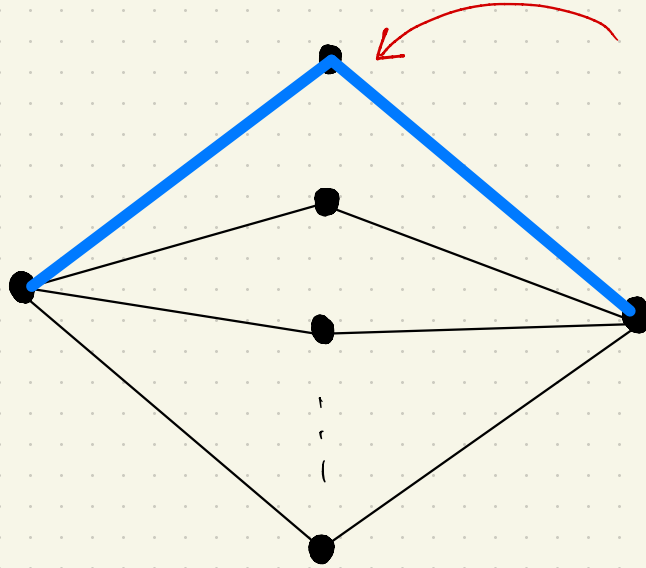


MST!

EXPONENTIALLY MANY SPANNING TREES

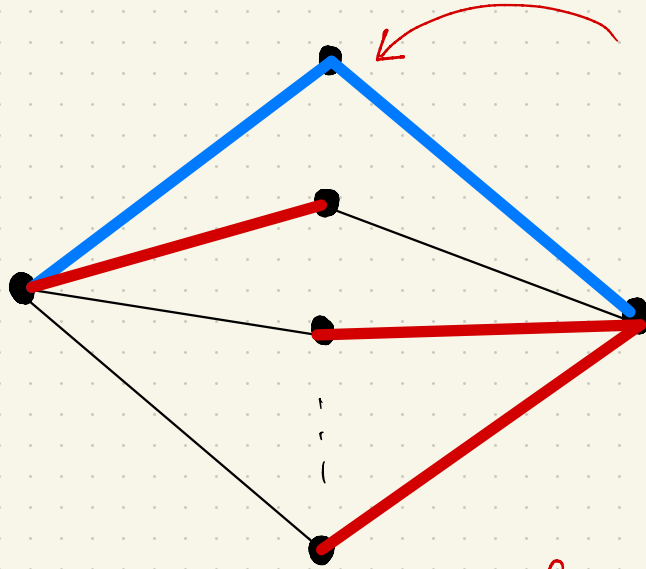


EXPONENTIALLY MANY SPANNING TREES



Some middle node
must have both edges
in spanning tree

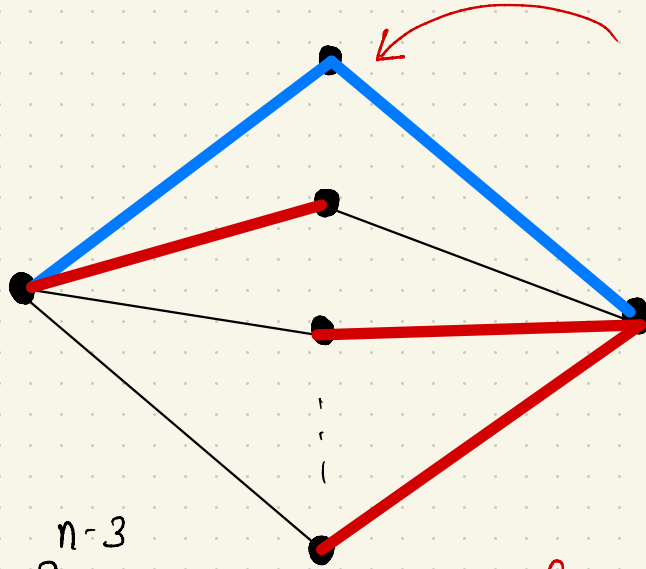
EXPONENTIALLY MANY SPANNING TREES



Some middle node
must have both edges
in spanning tree

for every other node,
pick either left or right edge

EXPONENTIALLY MANY SPANNING TREES

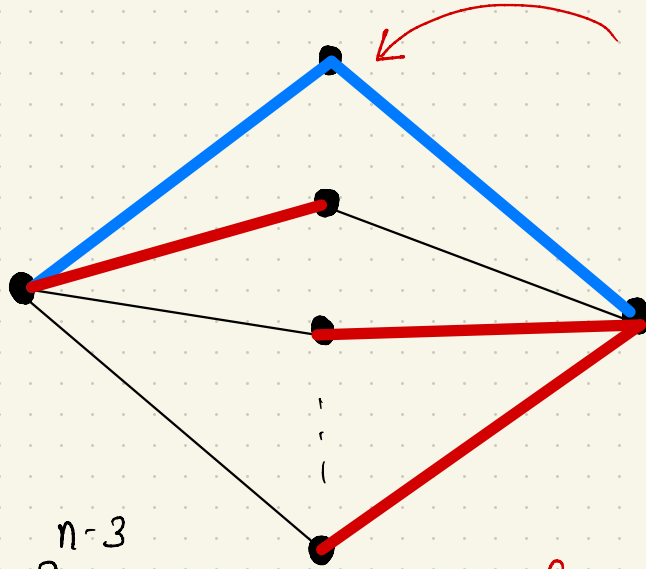


Some middle node
must have both edges
in spanning tree

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

for every other node,
pick either left or right edge

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

ASSUMPTIONS

1. Input graph G is connected.

ASSUMPTIONS

1. Input graph G is connected.

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

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**.

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

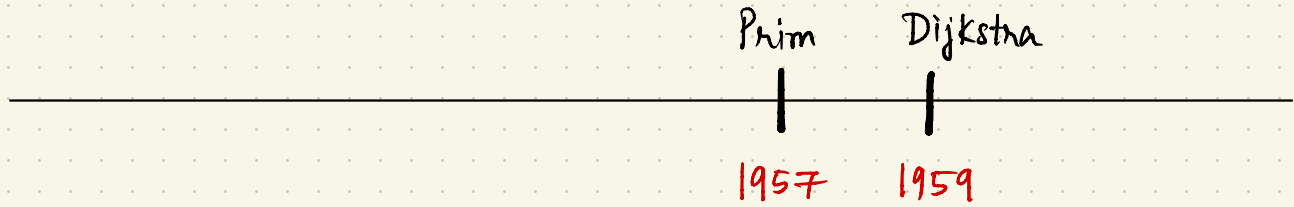
PRIM'S ALGORITHM

Prim

|

1957

PRIM'S ALGORITHM



PRIM'S ALGORITHM

Jarník



1930

Prim



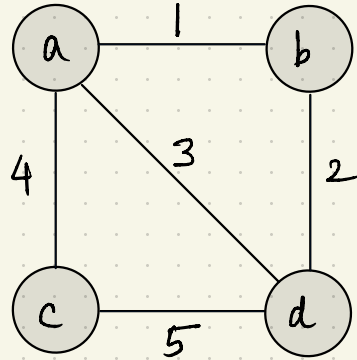
1957

Dijkstra

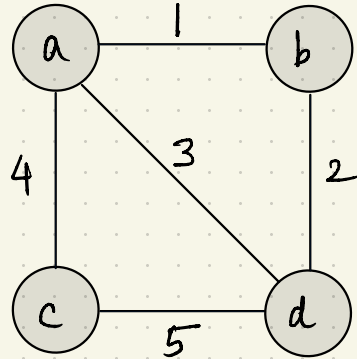


1959

PRIM'S ALGORITHM

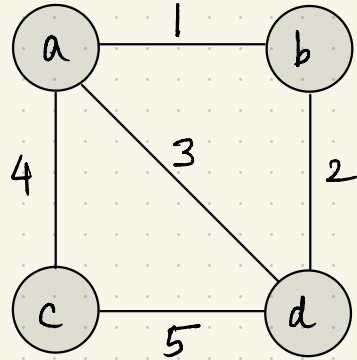


PRIM'S ALGORITHM



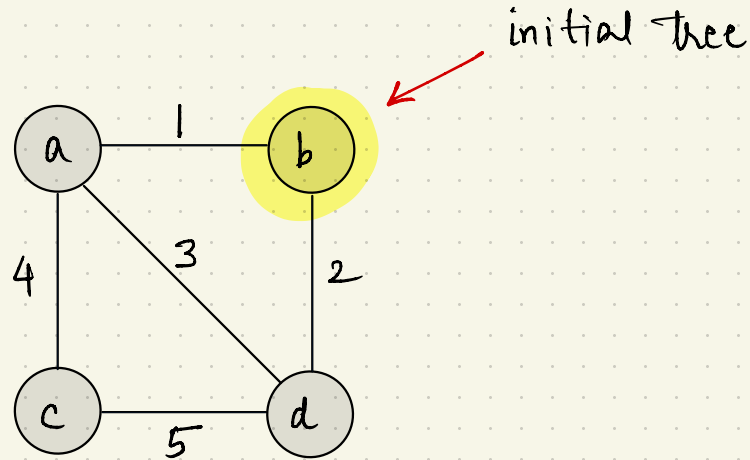
Compare with Dijkstra's shortest-path algorithm

PRIM'S ALGORITHM



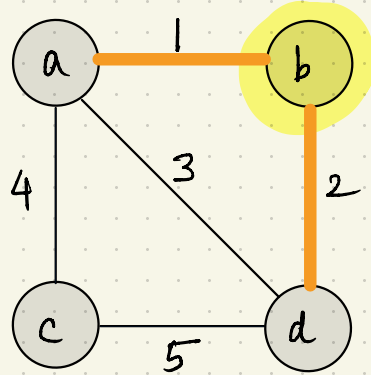
Idea: Grow a tree by adding the cheapest gateway edge at each stage.

PRIM'S ALGORITHM



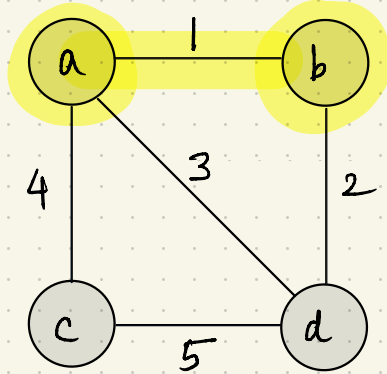
Idea: Grow a tree by adding the cheapest gateway edge at each stage.

PRIM'S ALGORITHM



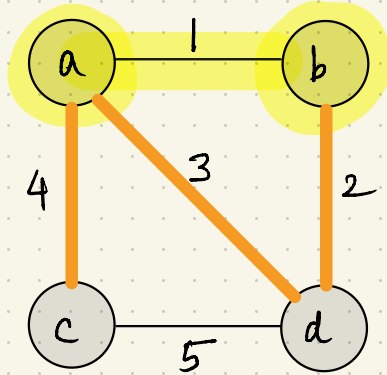
Idea: Grow a tree by adding the cheapest gateway edge at each stage.

PRIM'S ALGORITHM



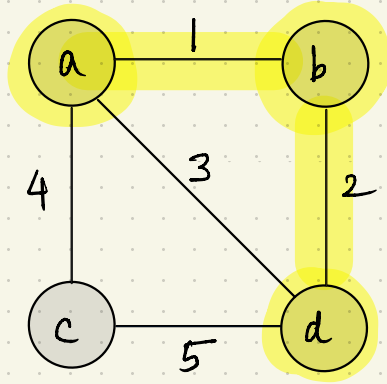
Idea: Grow a tree by adding the cheapest gateway edge at each stage.

PRIM'S ALGORITHM



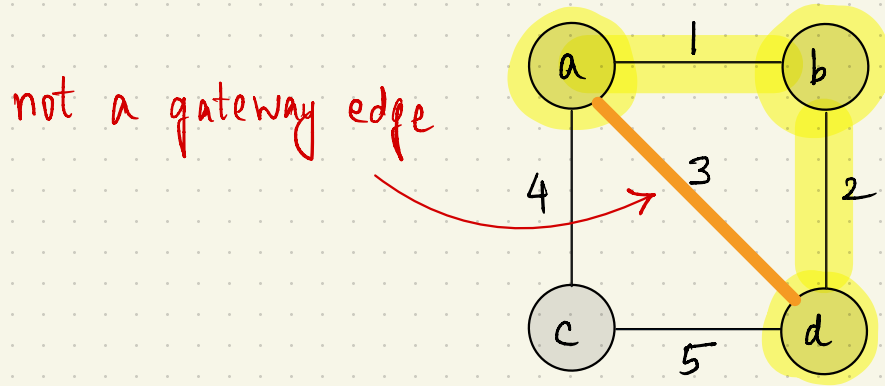
Idea: Grow a tree by adding the cheapest gateway edge at each stage.

PRIM'S ALGORITHM



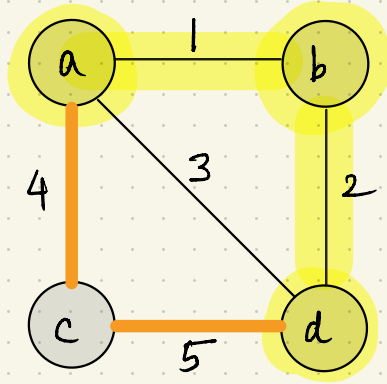
Idea: Grow a tree by adding the cheapest gateway edge at each stage.

PRIM'S ALGORITHM



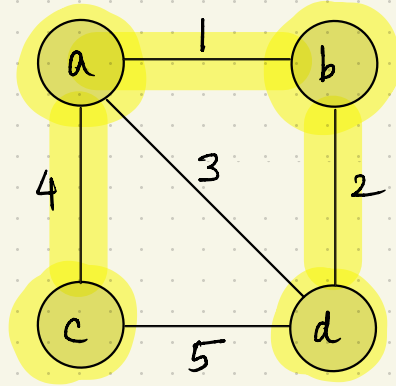
Idea: Grow a tree by adding the cheapest gateway edge at each stage.

PRIM'S ALGORITHM



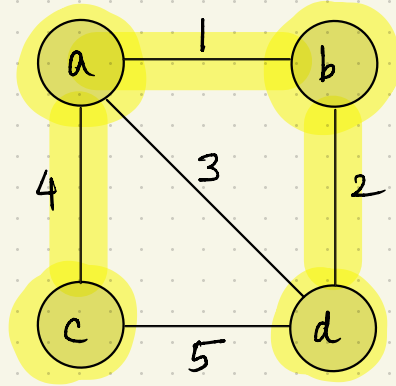
Idea: Grow a tree by adding the cheapest gateway edge at each stage.

PRIM'S ALGORITHM



Idea: Grow a tree by adding the cheapest gateway edge at each stage.

PRIM'S ALGORITHM



done!

Idea: Grow a tree by adding the cheapest gateway edge at each stage.

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

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

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

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 $X \neq V$

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 $u \in X$, $v \notin X$

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

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



How come Dijkstra's shortest-path algorithm requires non-negative edge lengths but Prim's MST algorithm doesn'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

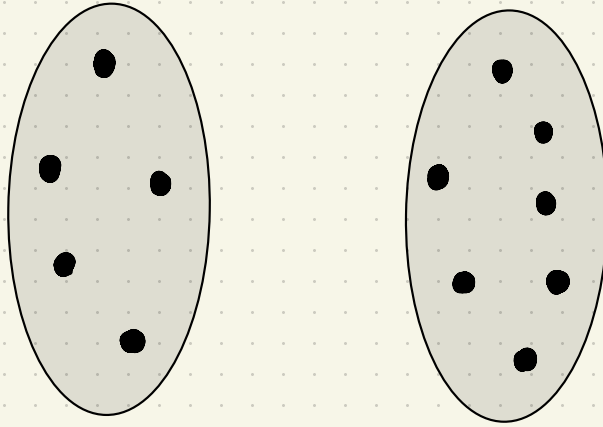
- use cut property

CUTS

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

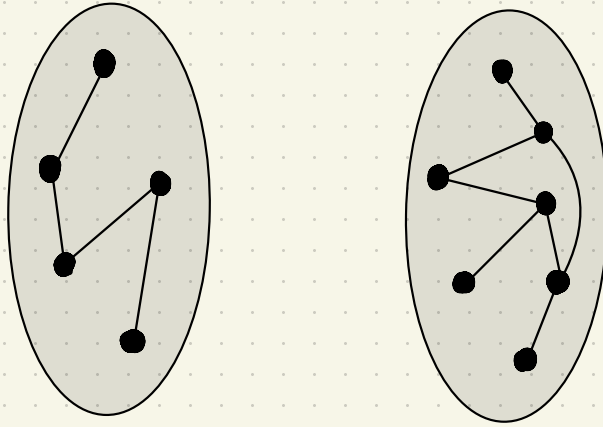
CUTS

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.

