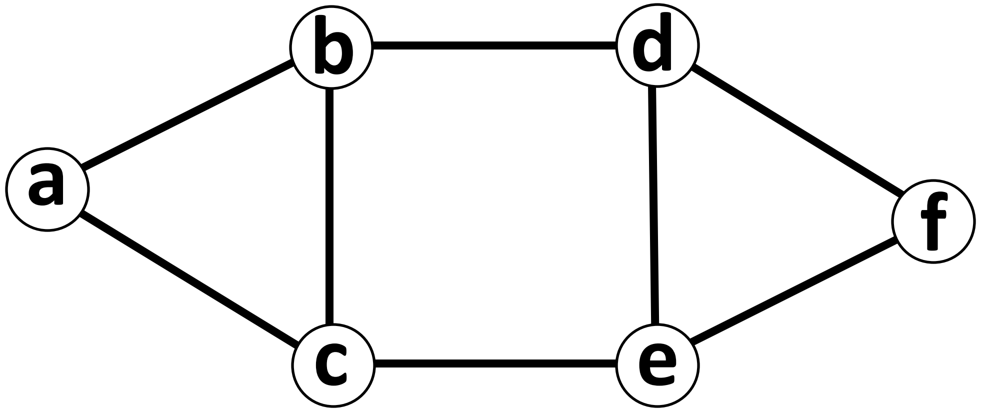
Minimum Spanning Trees

CSCI 532

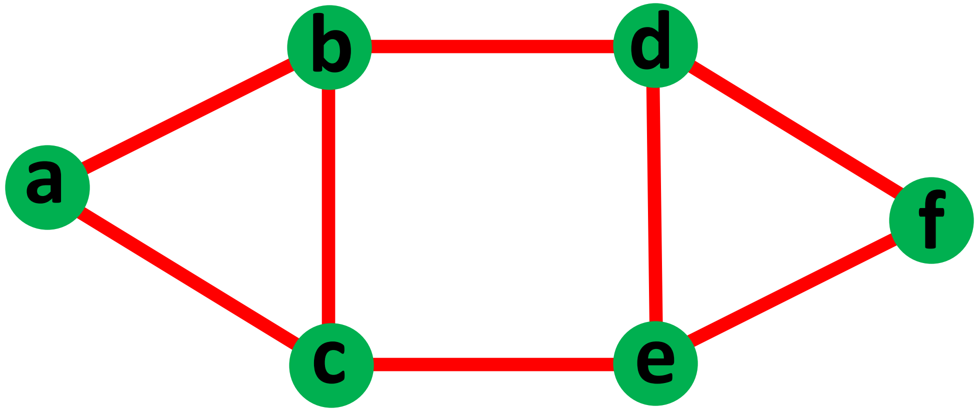
Graphs



Entity	Neighbors
a	b,c
b	a,c,d
c	a,b,e
d	b,e,f
e	c,d,f
f	d,e

Graphs are mathematical objects that represent connectivity relationships between entities.

Graphs



Edges

$$G = (V, E)$$

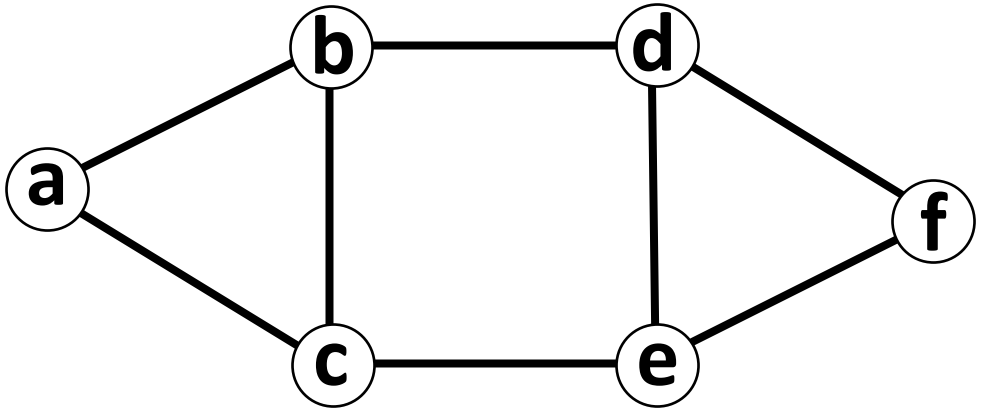
Vertices

(or Nodes)

Vertex	Neighbors
a	b,c
b	a,c,d
c	a,b,e
d	b,e,f
e	c,d,f
f	d,e

Graphs are mathematical objects that represent connectivity relationships between entities.

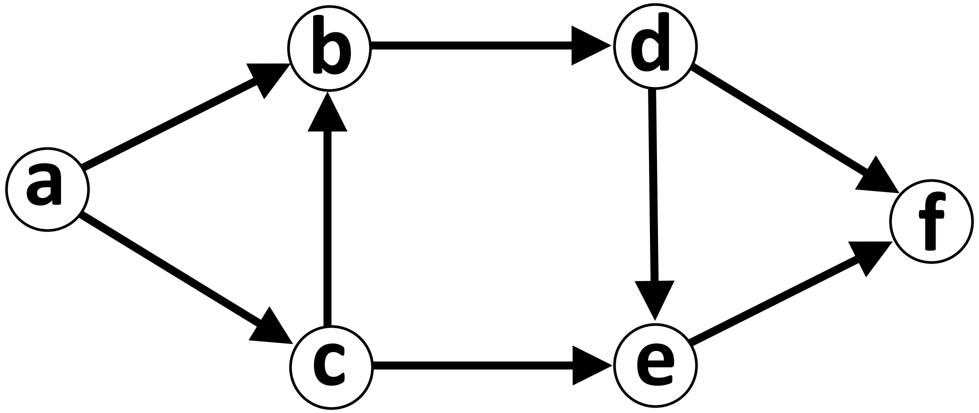
Graphs



$$\boldsymbol{G} = (\underset{\textbf{(or Nodes)}}{\underset{\textbf{Vertices}}{\underset{\uparrow}{\boldsymbol{V}}}}, \overset{\textbf{Edges}}{\overset{\downarrow}{\boldsymbol{E}}})$$

- Edges can be undirected…

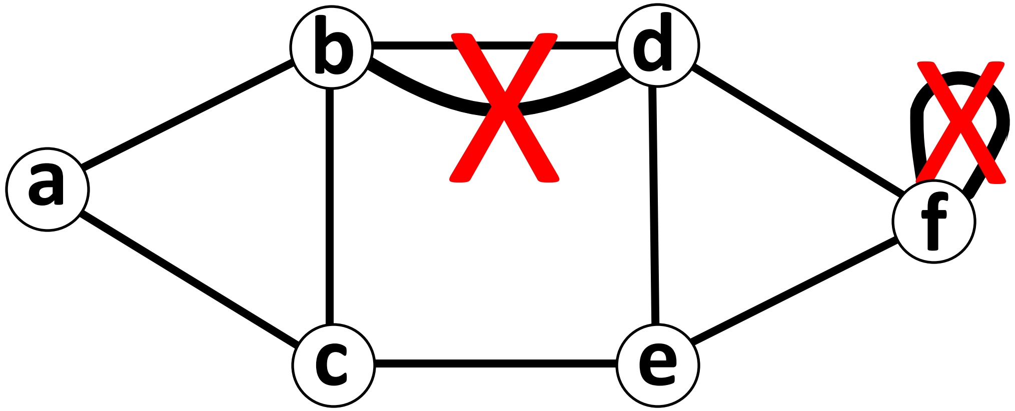
Graphs



$$G = (\underset{\substack{\uparrow \\ \textbf{Vertices} \\ \textbf{(or Nodes)}}}{V}, \overset{\substack{\textbf{Edges} \\ \downarrow}}{E})$$

- Edges can be undirected or directed.

Graphs



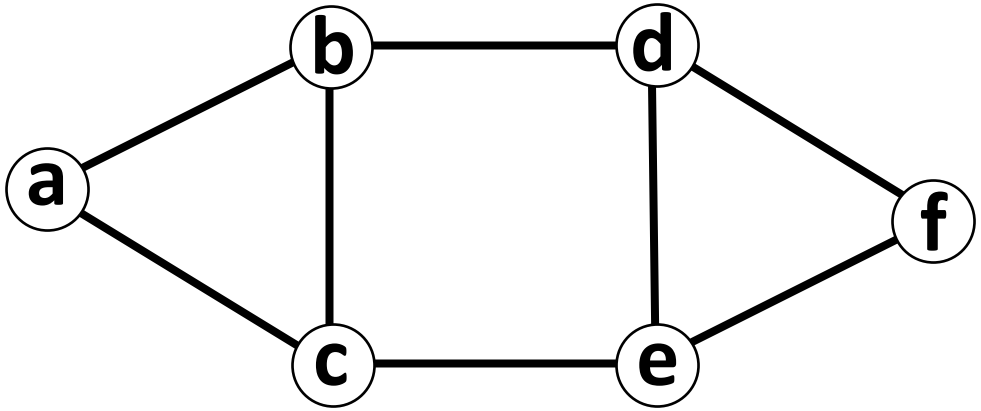
Edges

$$G = (V, E)$$

Vertices
(or Nodes)

- Edges can be directed or undirected.
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.

Graphs



Edges

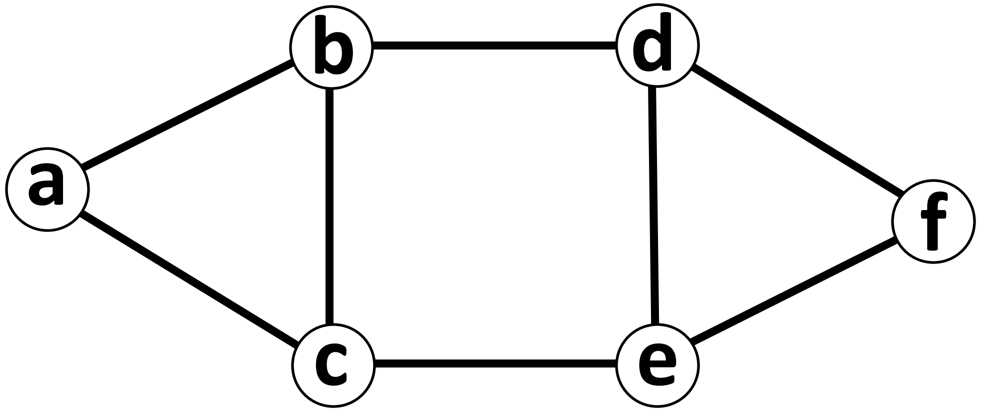
$$G = (V, E)$$

Vertices
(or Nodes)

- Edges can be directed or undirected.
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.

a,c,e,f ✓	**a,c,d,f** ✗
b,d ✓	**c,e,d,f,e** ✗

Graphs



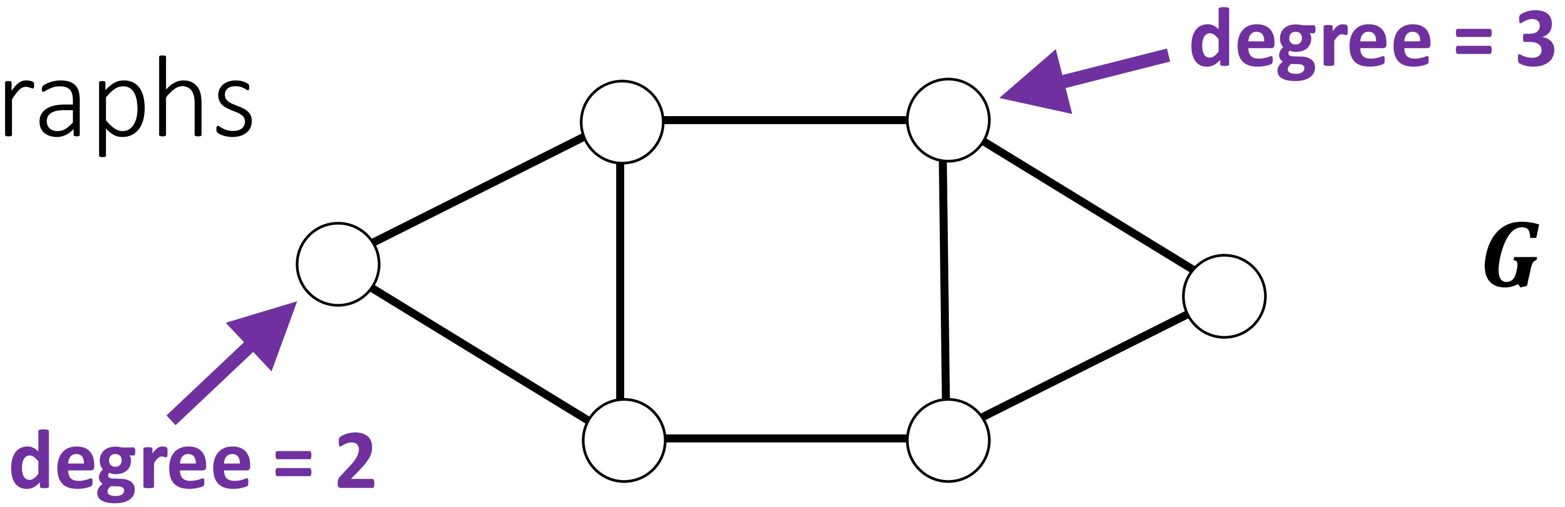
$$G = (\underset{\substack{\uparrow \\ \textbf{Vertices} \\ \textbf{(or Nodes)}}}{V}, \overset{\substack{\textbf{Edges} \\ \downarrow}}{E})$$

- Edges can be directed or undirected.
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices that start and end at same vertex.

(and usually with no other repeated vertices.)

c,b,d,e,c ✓ **a,c,e,f** ✗

Graphs



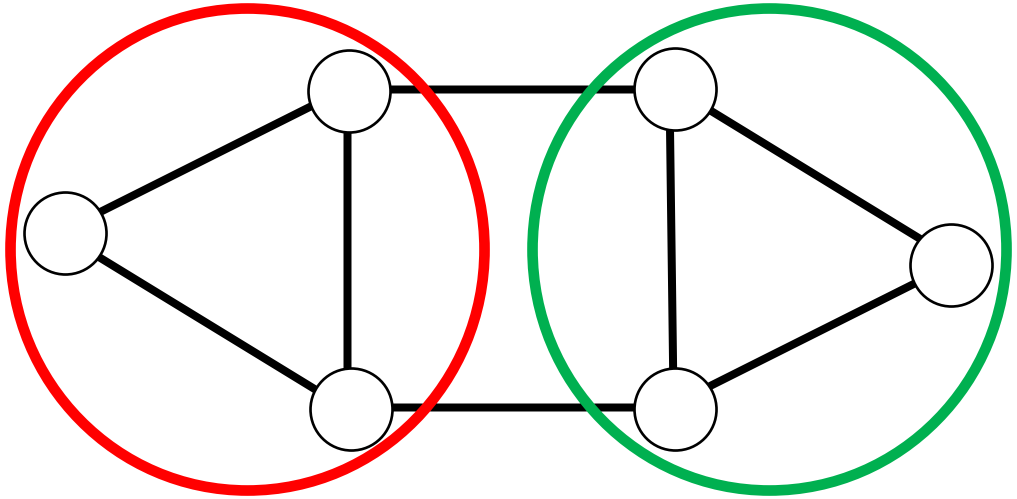
Edges

$$\boldsymbol{G = (V, E)}$$

Vertices
(or Nodes)

- Edges can be directed or undirected.
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices that start and end at same vertex.
- Degree of a vertex = $\deg(v)$ = # of edges touching it (undirected).

Graphs



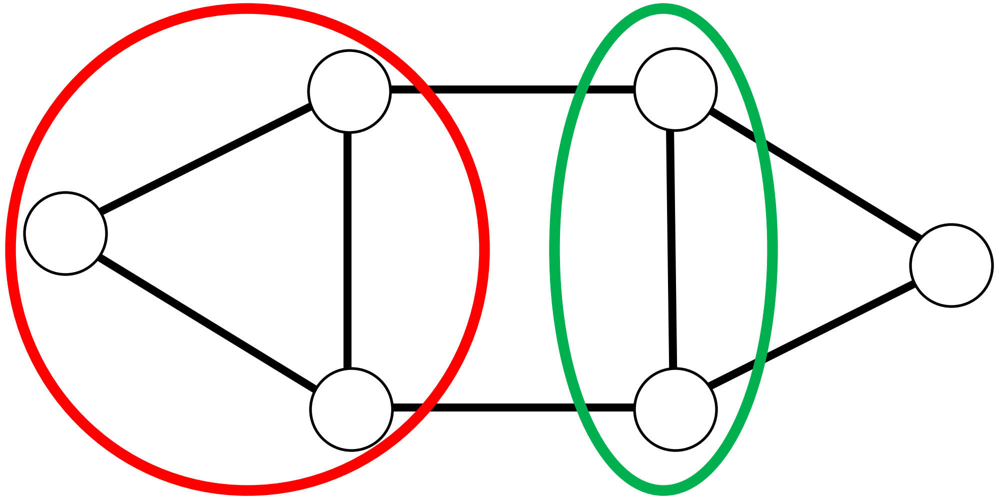
Edges

$$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$$

Vertices
(or Nodes)

- Edges can be directed or undirected.
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices that start and end at same vertex.
- Degree of a vertex = $\deg(v)$ = # of edges touching it (undirected).
- Cut = Partition of vertices into two disjoint subsets.

Graphs



Edges

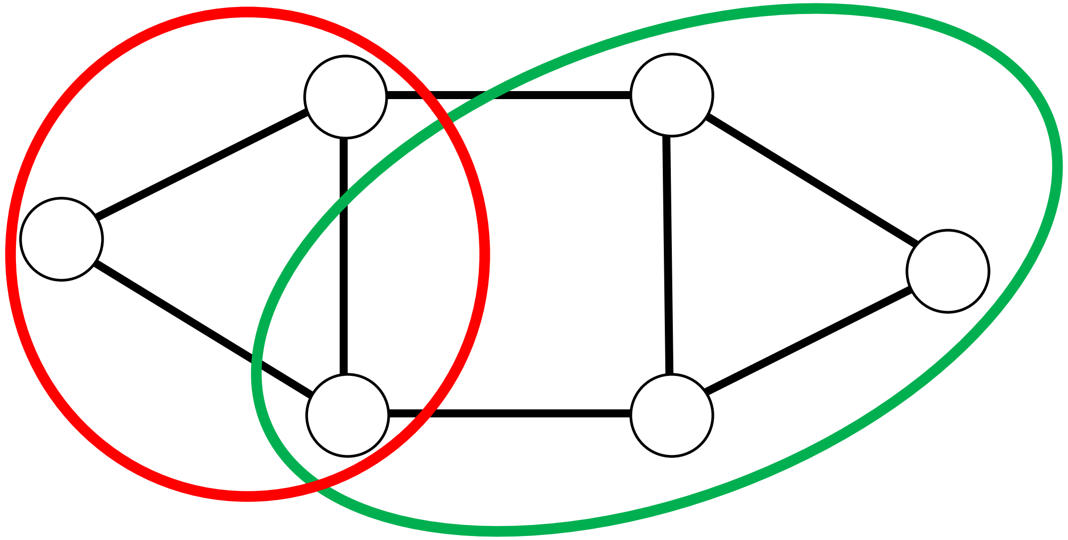
$$\boldsymbol{G = (V, E)}$$

Vertices

(or Nodes)

- Edges can be directed or undirected.
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices that start and end at same vertex.
- Degree of a vertex = $\deg(v)$ = # of edges touching it (undirected).
- Cut = Partition of vertices into two disjoint subsets.

Graphs



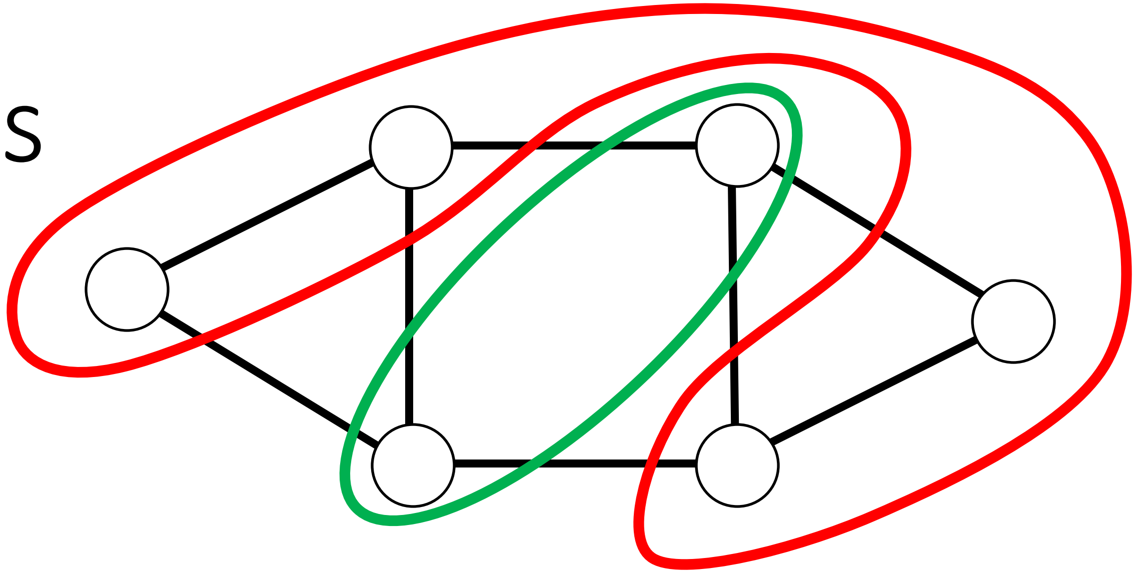
$$\underset{\substack{\uparrow \\ \textbf{Vertices} \\ \textbf{(or Nodes)}}}{}\boldsymbol{G = (V, E)}$$

Edges

Vertices (or Nodes)

- Edges can be directed or undirected.
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices that start and end at same vertex.
- Degree of a vertex = $\deg(v)$ = # of edges touching it (undirected).
- Cut = Partition of vertices into two disjoint subsets.

Graphs



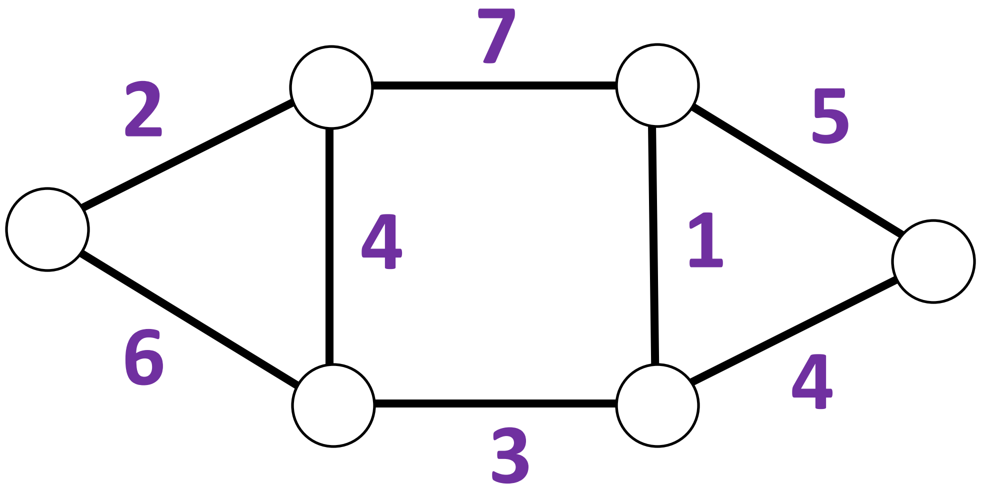
Edges

$$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$$

Vertices
(or Nodes)

- Edges can be directed or undirected.
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices that start and end at same vertex.
- Degree of a vertex = $\deg(v)$ = # of edges touching it (undirected).
- Cut = Partition of vertices into two disjoint subsets.

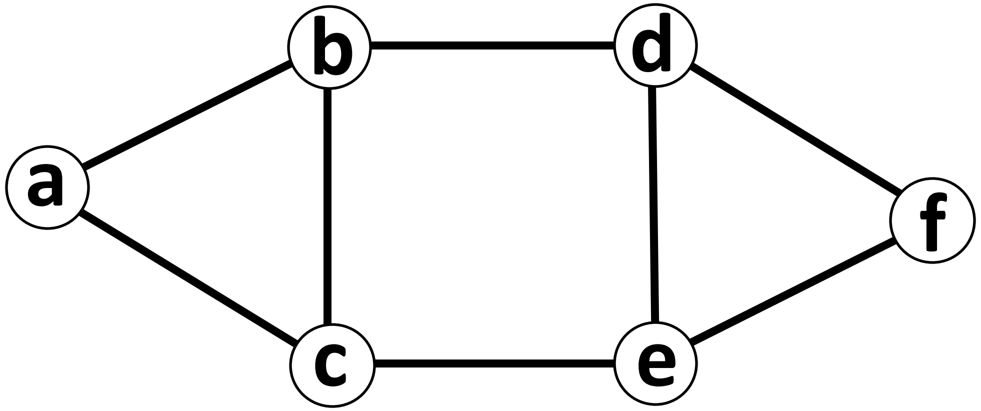
Graphs



$$\boldsymbol{G = (V, E)}$$

- Edges can be directed or undirected.
- Simple graph = At most one edge between pair of vertices and no edges that start and end at same vertex.
- Path = Sequence of vertices connected by edges without loops.
- Cycle = Sequence of vertices that start and end at same vertex.
- Degree of a vertex = $\deg(v)$ = # of edges touching it (undirected).
- Cut = Partition of vertices into two disjoint subsets.
- Edges (or vertices) can be weighted (cost associated with using it).

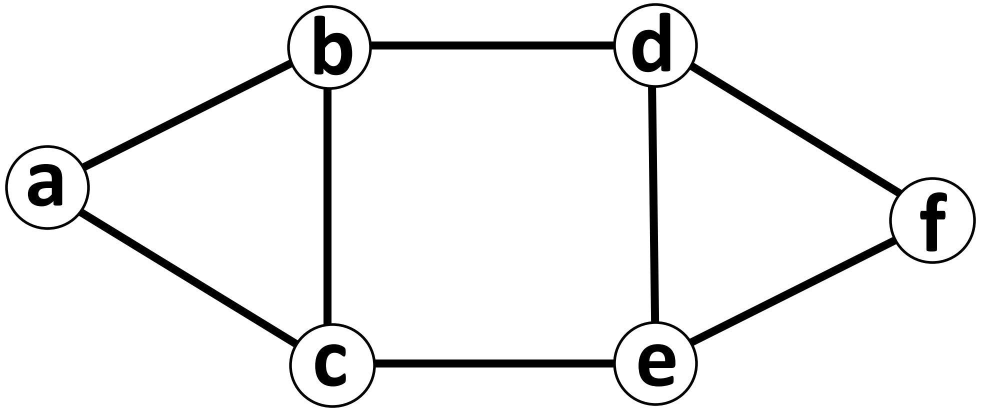
Graphs



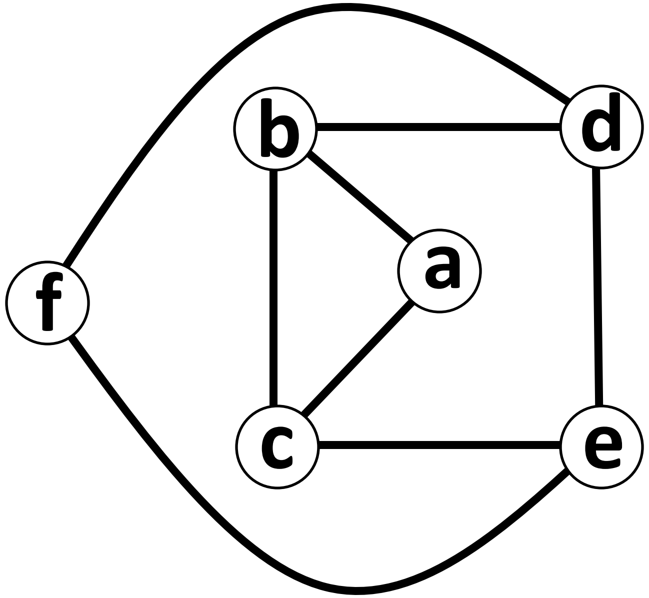
Vertex	Neighbors
a	b,c
b	a,c,d
c	a,b,e
d	b,e,f
e	c,d,f
f	d,e

Graphs are mathematical objects that represent connectivity relationships between entities.

Graphs



=



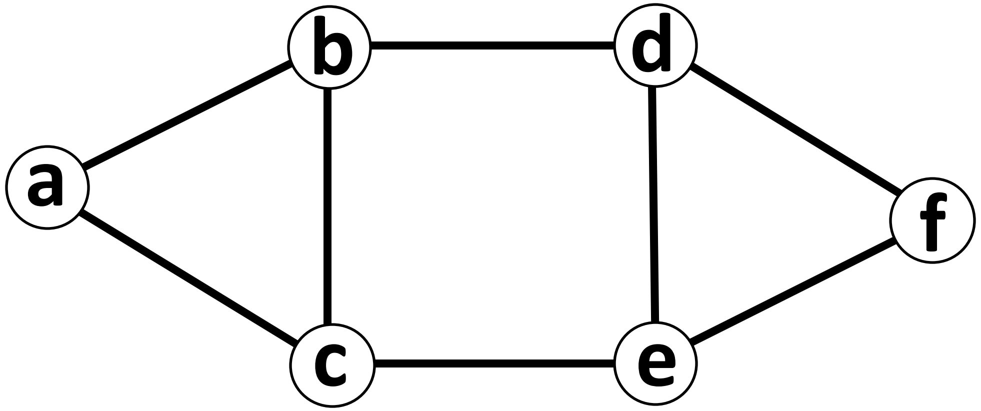
Vertex	Neighbors
a	b,c
b	a,c,d
c	a,b,e
d	b,e,f
e	c,d,f
f	d,e

=

Vertex	Neighbors
a	b,c
b	a,c,d
c	a,b,e
d	b,e,f
e	c,d,f
f	d,e

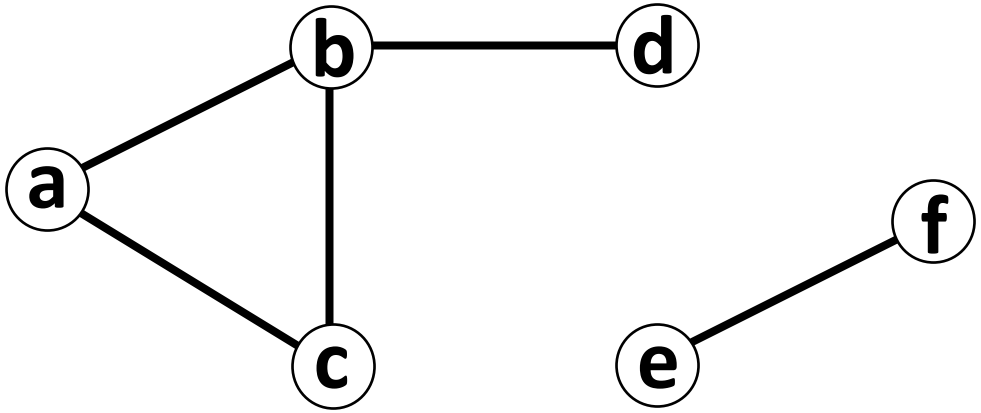
Topologically equivalent
(i.e., same connectivity)

Special Graphs

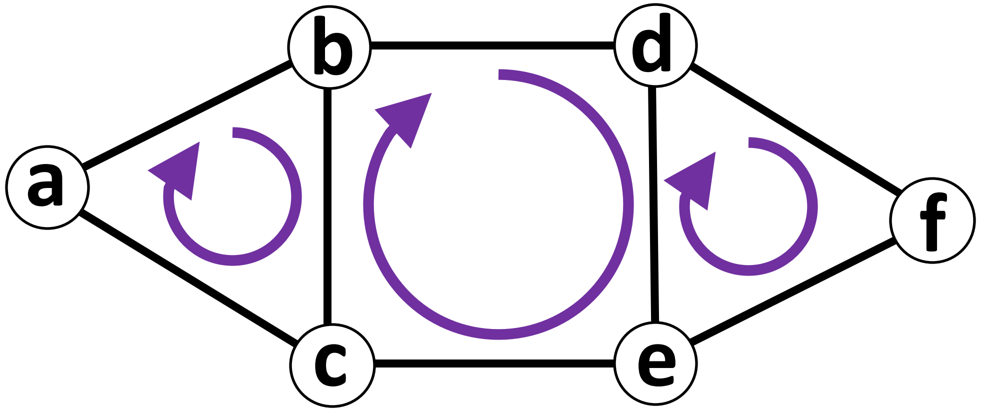


$$\left.\begin{array}{l}\textbf{Vertices (or Nodes)}\\ \textbf{Edges}\end{array}\right\} \boldsymbol{G = (V, E)}$$

- Connected Graph = Graph that has a path between every vertex pair.

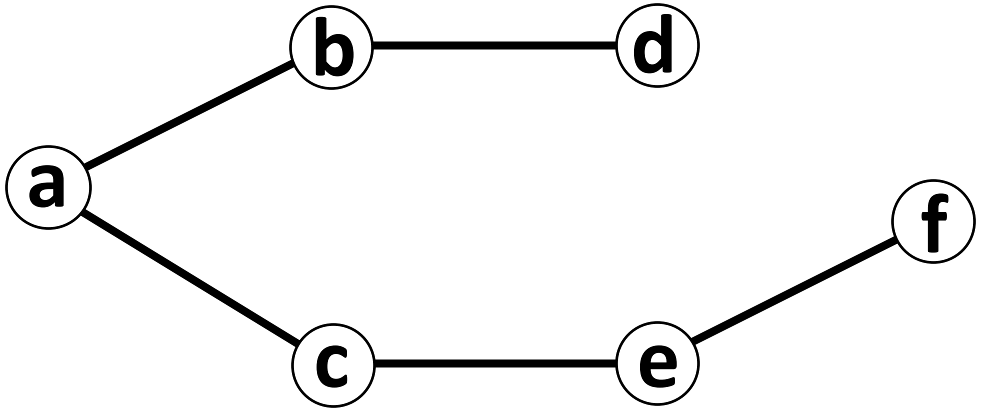


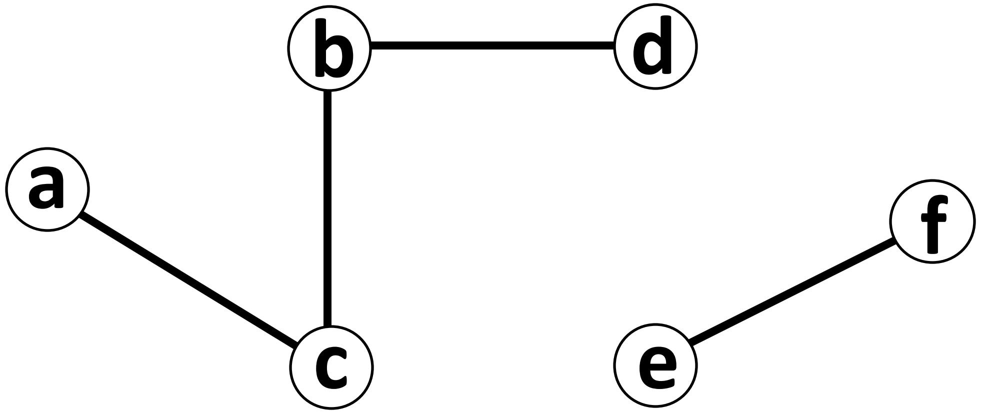
Special Graphs



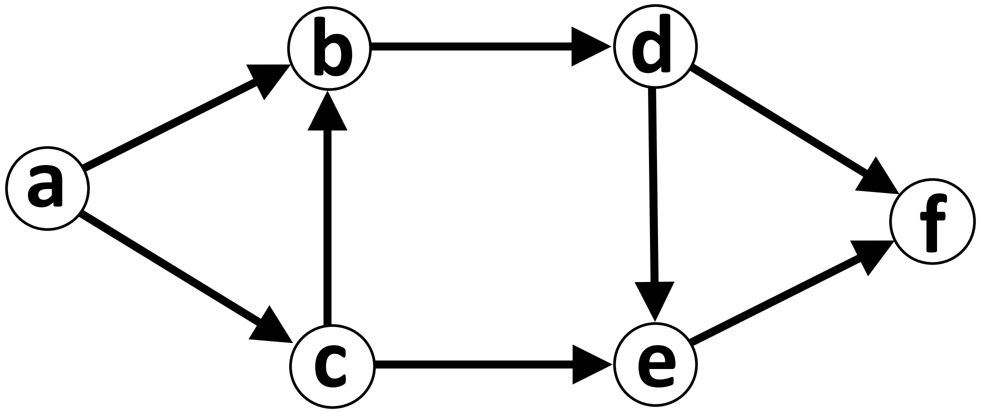
$$\left.\begin{array}{l}\textbf{Vertices (or Nodes)}\\ \textbf{Edges}\end{array}\right\} \boldsymbol{G = (V, E)}$$

- Connected Graph = Graph that has a path between every vertex pair.
- Acyclic Graph = Graph with no cycles.



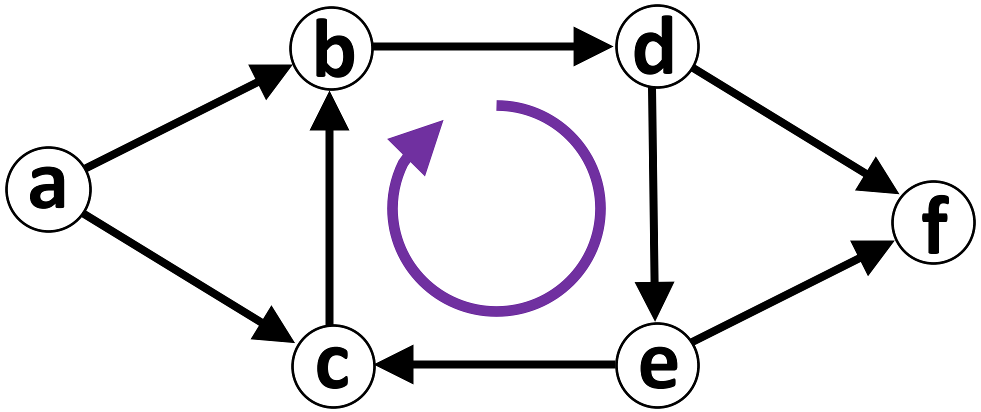


Special Graphs

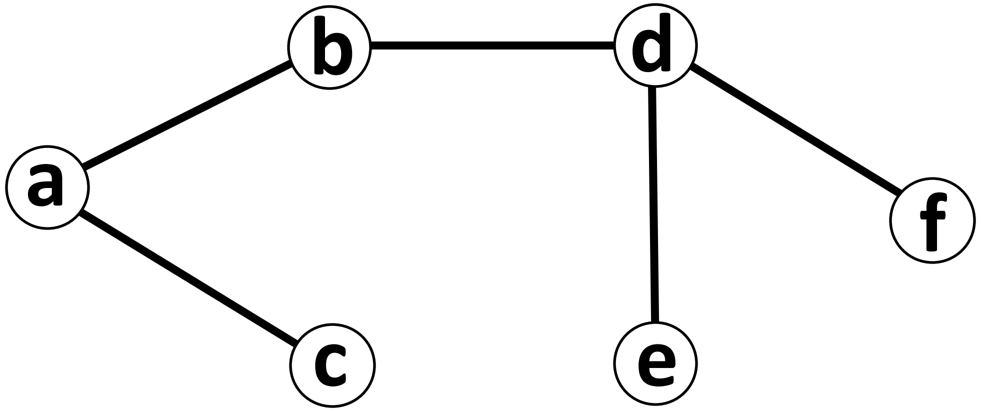


$$\left.\begin{array}{l}\textbf{Vertices (or Nodes)}\\ \textbf{Edges}\end{array}\right\} \boldsymbol{G = (V, E)}$$

- Connected Graph = Graph that has a path between every vertex pair.
- Acyclic Graph = Graph with no cycles.
- Directed Acyclic Graph (DAG) = Directed graph with no cycles.

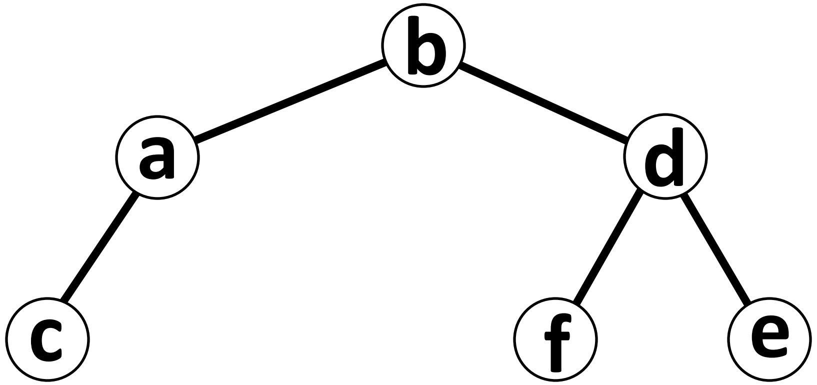


Special Graphs

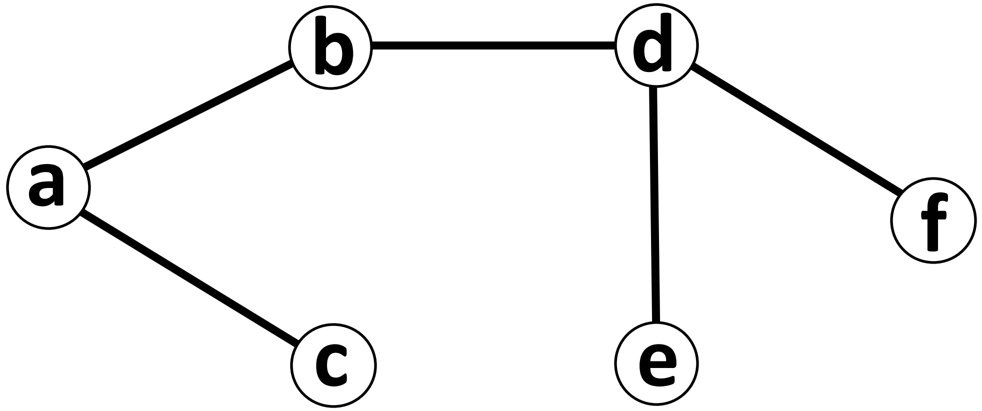


$$\left.\begin{array}{l}\textbf{Vertices (or Nodes)}\\ \textbf{Edges}\end{array}\right\} \boldsymbol{G = (V, E)}$$

- Connected Graph = Graph that has a path between every vertex pair.
- Acyclic Graph = Graph with no cycles.
- Directed Acyclic Graph (DAG) = Directed graph with no cycles.
- Tree = Connected acyclic graph.

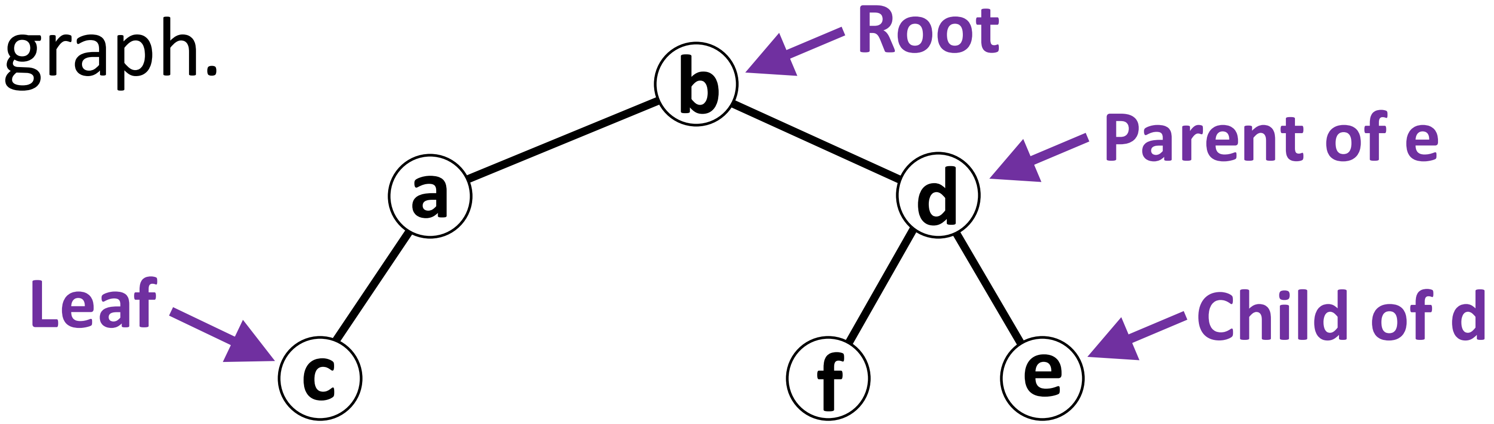


Special Graphs

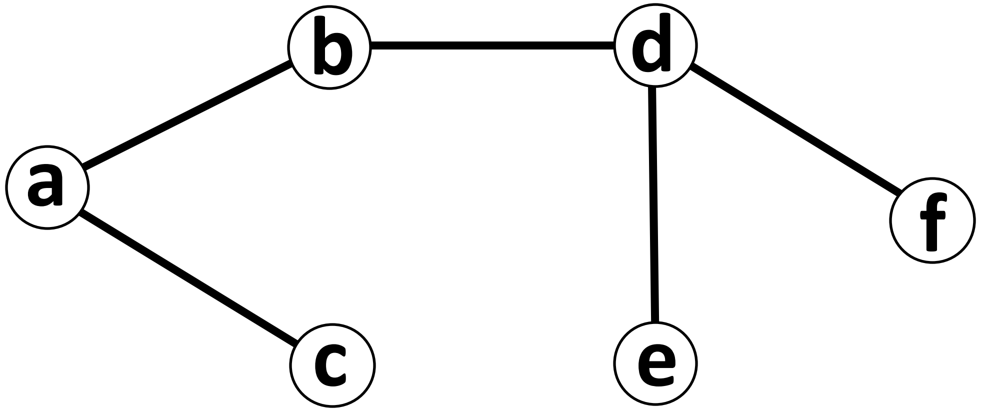


$$\left.\begin{array}{l}\textbf{Vertices (or Nodes)}\\ \textbf{Edges}\end{array}\right\}\ \boldsymbol{G = (V, E)}$$

- Connected Graph = Graph that has a path between every vertex pair.
- Acyclic Graph = Graph with no cycles.
- Directed Acyclic Graph (DAG) = Directed graph with no cycles.
- Tree = Connected acyclic graph.

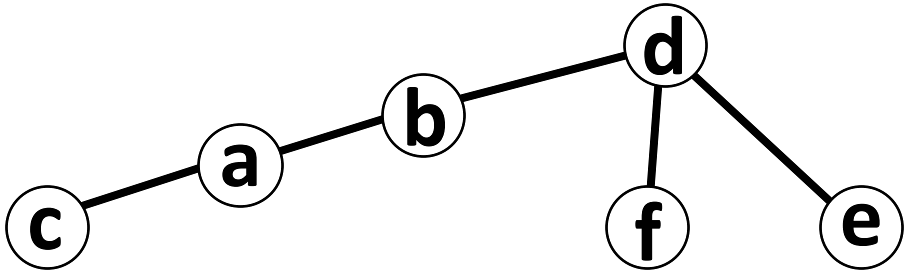


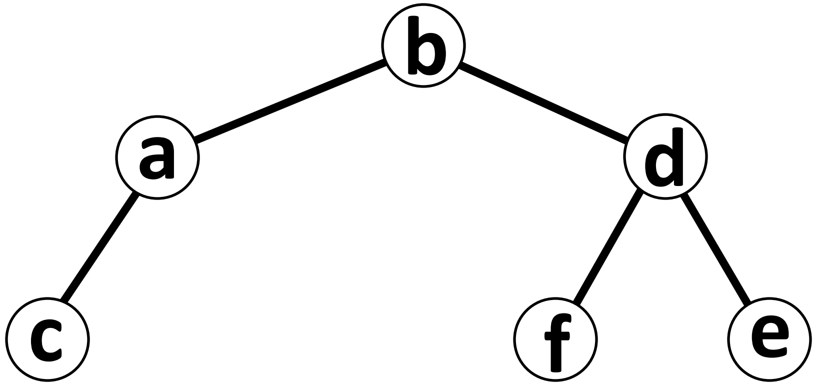
Special Graphs



$$\left.\begin{array}{l}\textbf{Vertices (or Nodes)}\\ \textbf{Edges}\end{array}\right\} \boldsymbol{G = (V, E)}$$

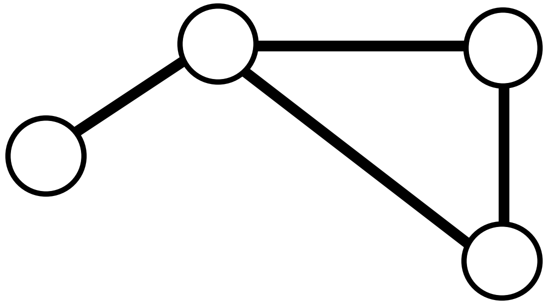
- Connected Graph = Graph that has a path between every vertex pair.
- Acyclic Graph = Graph with no cycles.
- Directed Acyclic Graph (DAG) = Directed graph with no cycles.
- Tree = Connected acyclic graph.





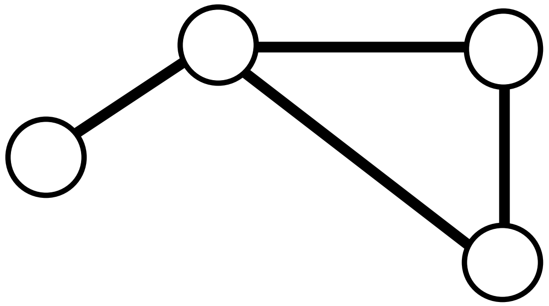
Topologically equivalent, but information may be lost…

Minimum Spanning Tree (MST)



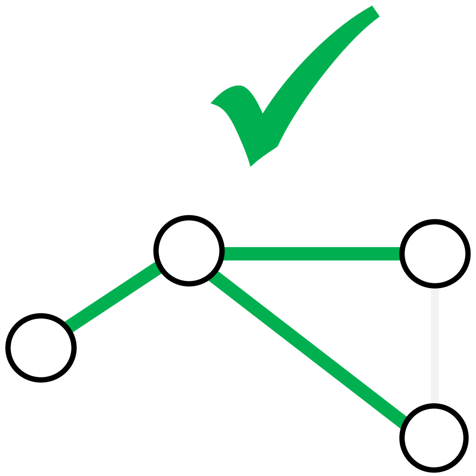
Given a connected graph, a subset of edges is a…

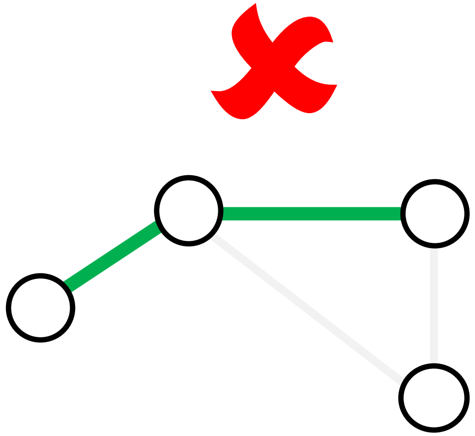
Minimum Spanning Tree (MST)



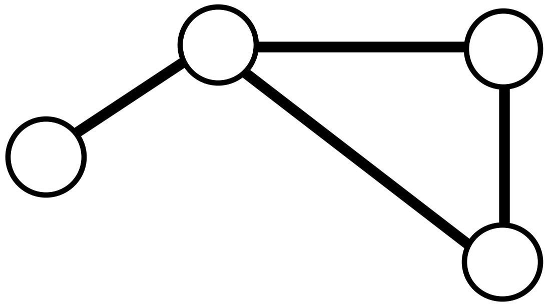
Given a connected graph, a subset of edges is a…

Spanning tree if it is a tree and includes all vertices in the graph.



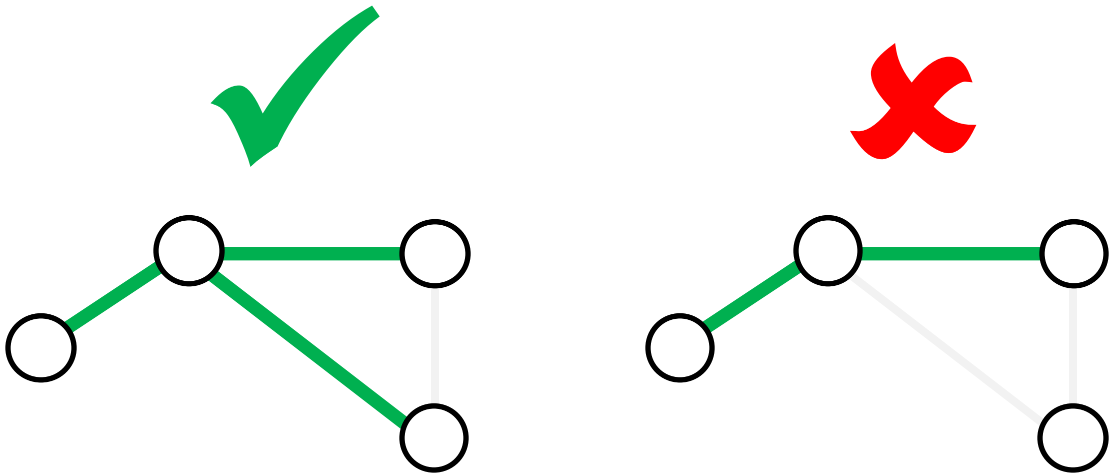


Minimum Spanning Tree (MST)

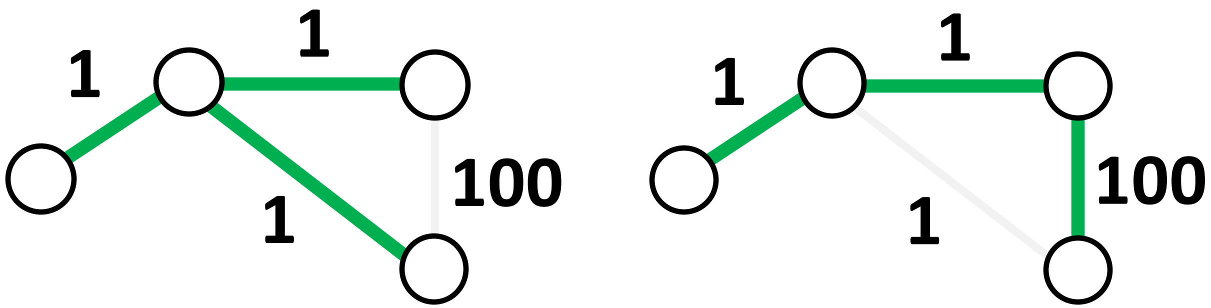


Given a connected graph, a subset of edges is a...

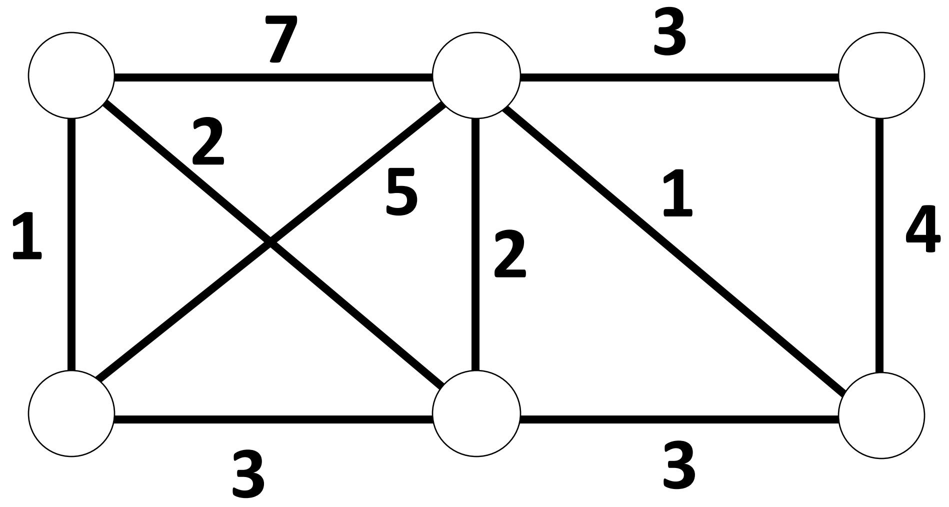
Spanning tree if it is a tree and includes all vertices in the graph.



Minimum spanning tree if it is a spanning tree whose sum of edge costs is the minimum possible value.

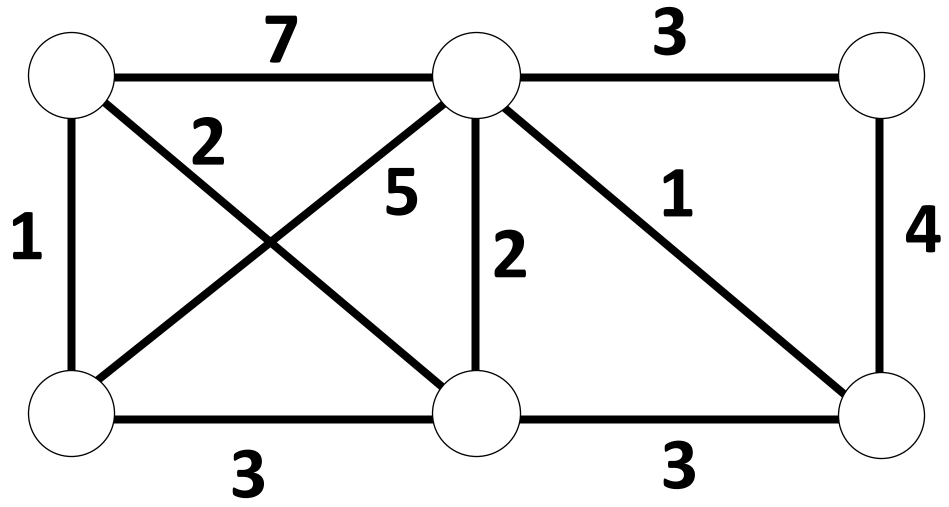


Kruskal’s MST Algorithm



<u>Goal:</u> Given a connected graph, find its Minimum Spanning Tree.

Kruskal’s MST Algorithm

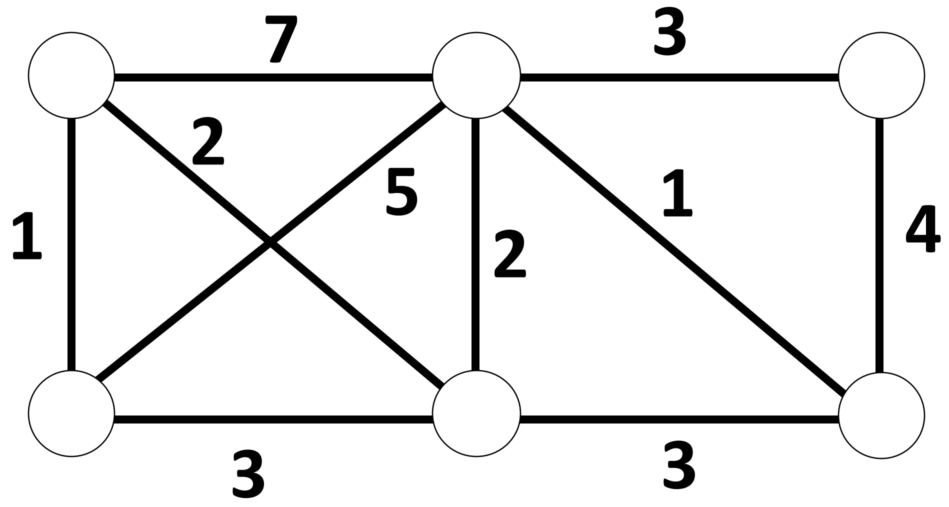


Greedy Algorithms:

- Make the choice that best helps some objective.
- Do not look ahead, plan, or revisit past decisions.
- Hope that optimal local choices lead to optimal global solutions.

Kruskal’s MST Algorithm

Algorithm: ??

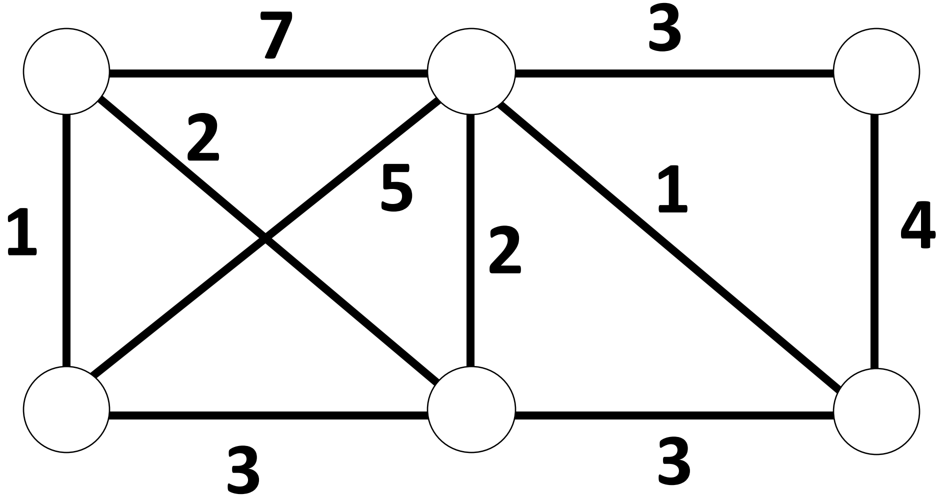


Greedy Algorithms:

- Make the choice that best helps some objective.
- Do not look ahead, plan, or revisit past decisions.
- Hope that optimal local choices lead to optimal global solutions.

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

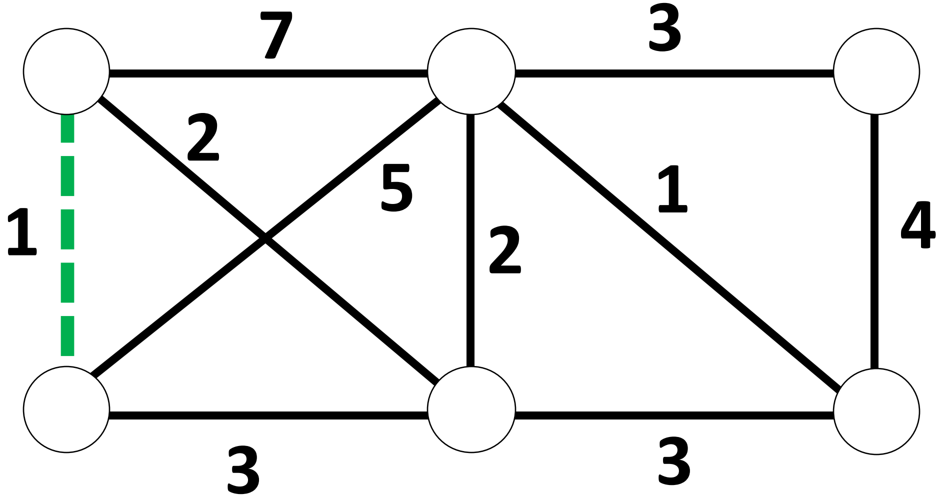


Greedy Algorithms:

- Make the choice that best helps some objective.
- Do not look ahead, plan, or revisit past decisions.
- Hope that optimal local choices lead to optimal global solutions.

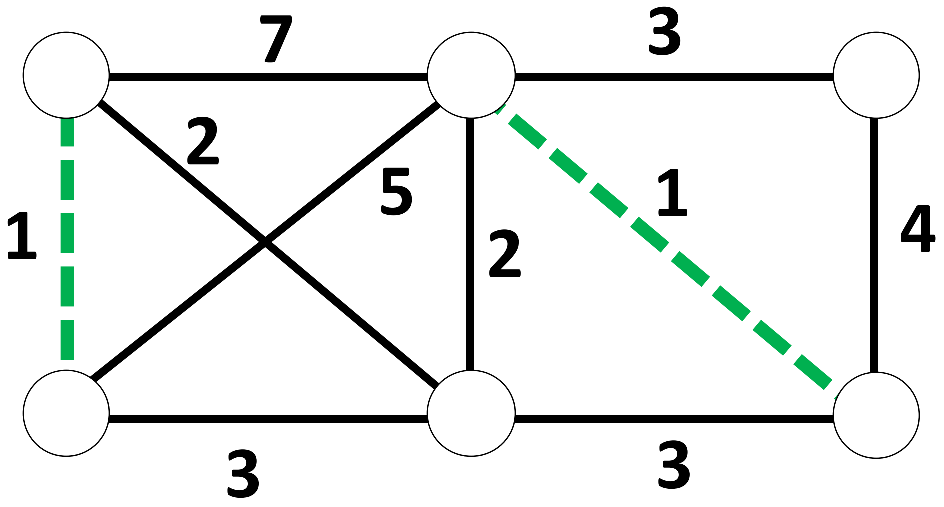
Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.



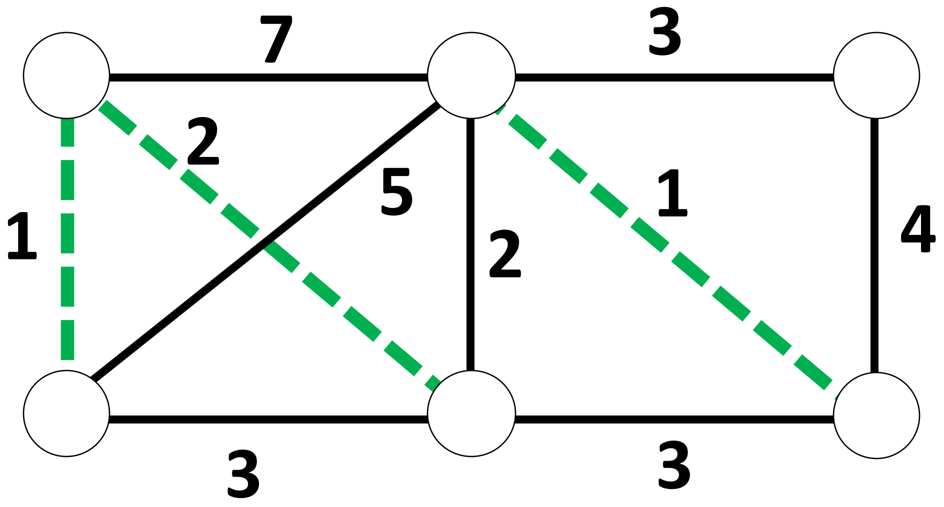
Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.



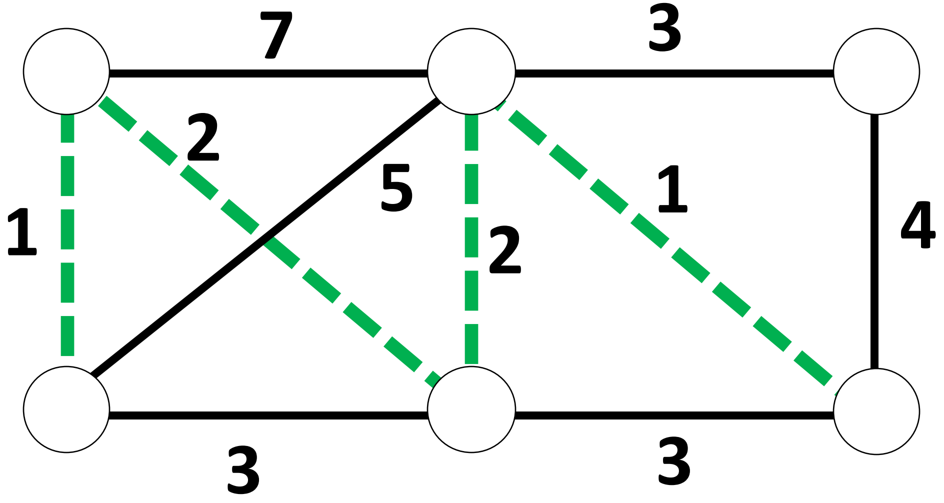
Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.



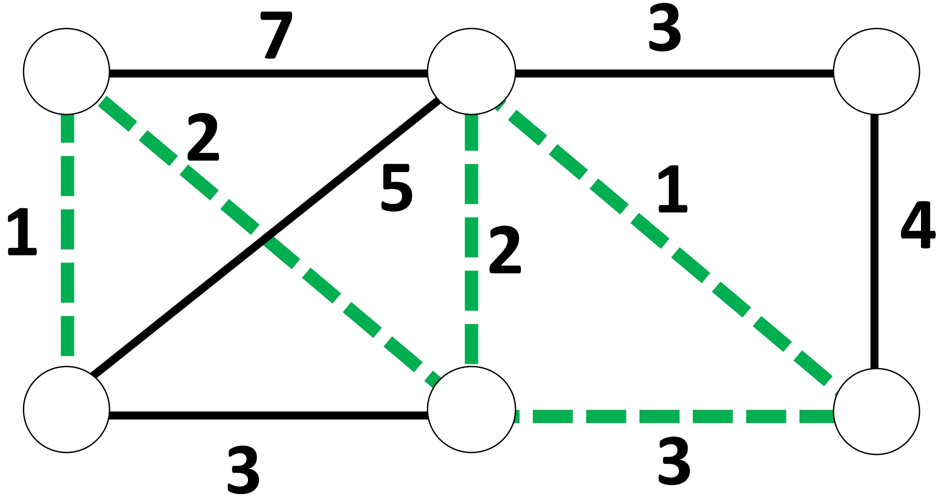
Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.



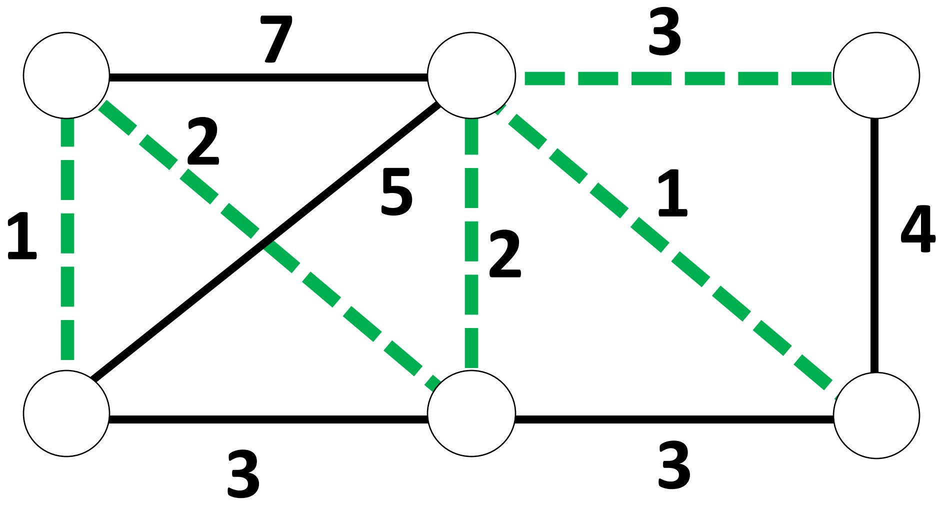
Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.



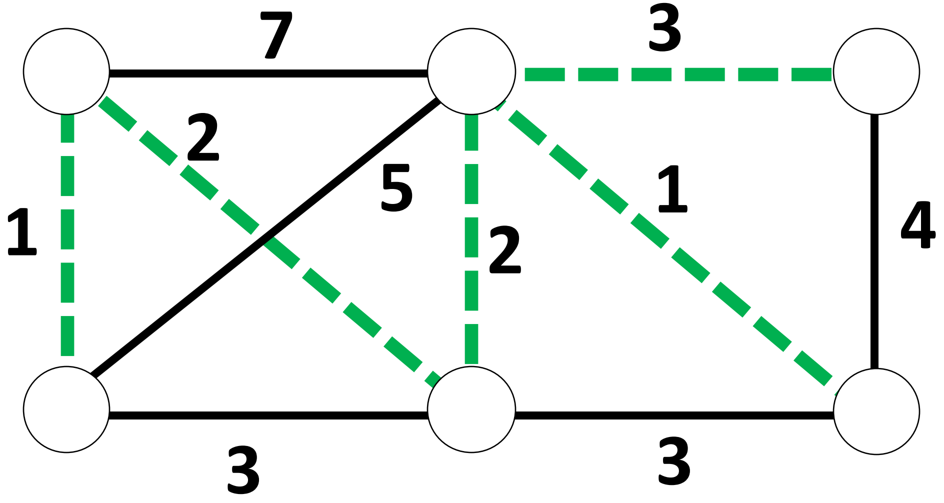
Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.



Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.



What are some questions we may have about the algorithm?

1. Is the solution valid? (Does it actually find a spanning tree?)
2. What is the running time?
3. Is the solution optimal?

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of validity: ?

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of validity: Let $G = (V, E)$ be the connected graph, and $T \subseteq E$ be the set of edges resulting from Kruskal’s algorithm.

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of validity: Let $G = (V, E)$ be the connected graph, and $T \subseteq E$ be the set of edges resulting from Kruskal’s algorithm.

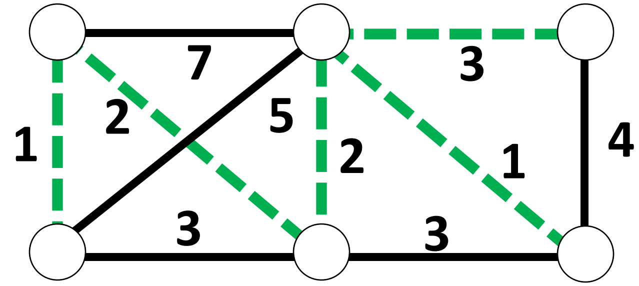
What do we need to show?

Kruskal's MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of validity: Let $G = (V, E)$ be the connected graph, and $T \subseteq E$ be the set of edges resulting from Kruskal's algorithm.

T is a tree because it is connected (otherwise we could have added more edges without creating cycles) and there are no cycles.



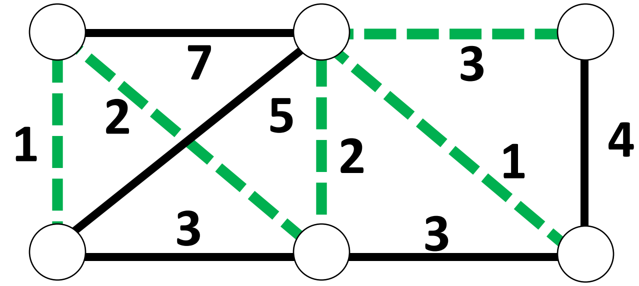
Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of validity: Let $G = (V, E)$ be the connected graph, and $T \subseteq E$ be the set of edges resulting from Kruskal’s algorithm.

T is a tree because it is connected (otherwise we could have added more edges without creating cycles) and there are no cycles.

T spans G because if it did not, we could have added more edges to connected unreached nodes without creating cycles.



Kruskal’s MST Algorithm

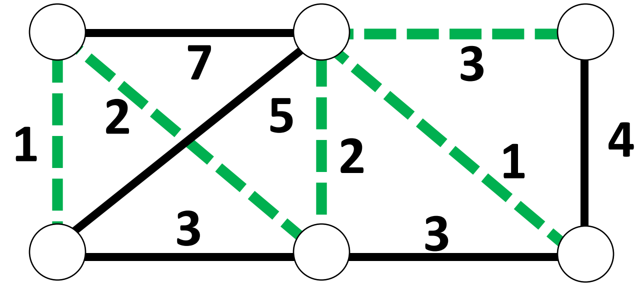
Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of validity: Let $G = (V, E)$ be the connected graph, and $T \subseteq E$ be the set of edges resulting from Kruskal’s algorithm.

T is a tree because it is connected (otherwise we could have added more edges without creating cycles) and there are no cycles.

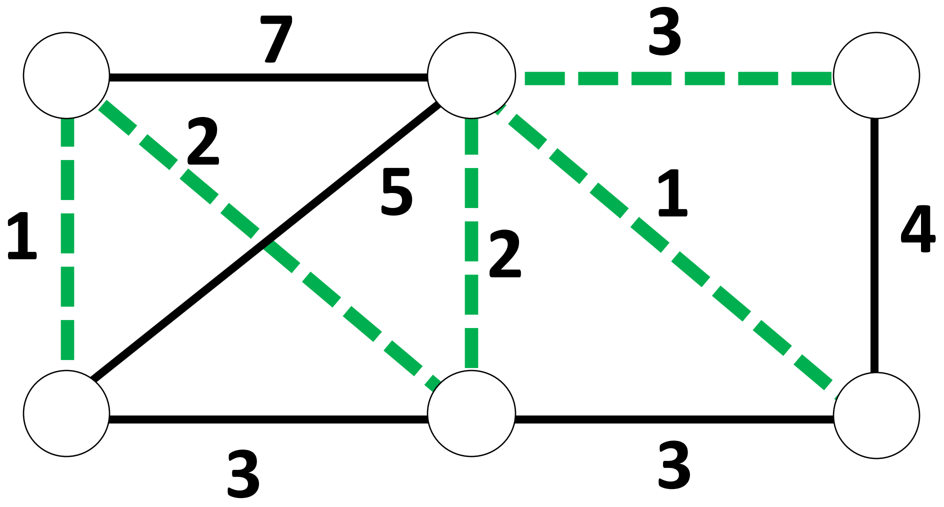
T spans G because if it did not, we could have added more edges to connected unreached nodes without creating cycles.

$\therefore$ T is a spanning tree of G



Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.



What are some questions we may have about the algorithm?

1. ~~Is the solution valid? (Does it actually find a spanning tree?)~~
2. What is the running time?
3. Is the solution optimal?

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Running Time:

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Running Time:

```
findMST(G=(V,E)) {
  T = Ø
  sort(E) //smallest to largest weight
  for (e in E) {
    if (T U {e} is acyclic) {
      T = T U {e}
    }
  }
  return T
}
```

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Running Time:

```
findMST(G=(V,E)) {
  T = Ø
  sort(E) //smallest to largest weight ← $O(|E|\log(|E|))$
  for (e in E) {
    if (T U {e} is acyclic) {
      T = T U {e}
    }
  }
  return T
}
```

Kruskal's MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Running Time:

```
findMST(G=(V,E)) {
   T = Ø
   sort(E) //smallest to largest weight ← $O(|E| \log(|E|))$
   for (e in E) { ← $O(|E|)$
      if (T ∪ {e} is acyclic) {
         T = T ∪ {e}
      }
   }
   return T
}
```

Kruskal’s MST Algorithm

<u>Algorithm:</u> Add the edge with smallest weight, that does not create a cycle.

<u>Running Time:</u>

```
findMST(G=(V,E)) {
  T = ∅
  sort(E) //smallest to largest weight ← $O(|E|\log(|E|))$
  for (e in E) { ← $O(|E|)$
    if (T U {e} is acyclic) { ← $O(|V|+|E|)$ using BFS
      T = T U {e}
    }
  }
  return T
}
```

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Running Time:

```
findMST(G=(V,E)) {
  T = ∅
  sort(E) //smallest to largest weight ← $\boldsymbol{O(|E|\log(|E|))}$
  for (e in E) { ← $\boldsymbol{O(|E|)}$
    if (T ∪ {e} is acyclic) { ← $\boldsymbol{O(|V|+|E|)}$ using BFS
      T = T ∪ {e}
    }
  }
  return T
}
```

Running time

$\in \boldsymbol{O\big(|E|\log(|E|) + |E|(|V|+|E|)\big)}$

$\in \boldsymbol{O\big(|E|^2 + |E||V|\big)}$

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Running Time:

```
findMST(G=(V,E)) {
   T = ∅
   sort(E) //smallest to largest weight
   for (e in E) {
      if (T ∪ {e} is acyclic) {
         T = T ∪ {e}
      }
   }
   return T
}
```

Can be improved to $O(1)$, thus $O(|E| \log(|E|))$ overall

$\leftarrow O(|E| \log(|E|))$

$\leftarrow O(|E|)$

$\leftarrow O(|V| + |E|)$ **using BFS**

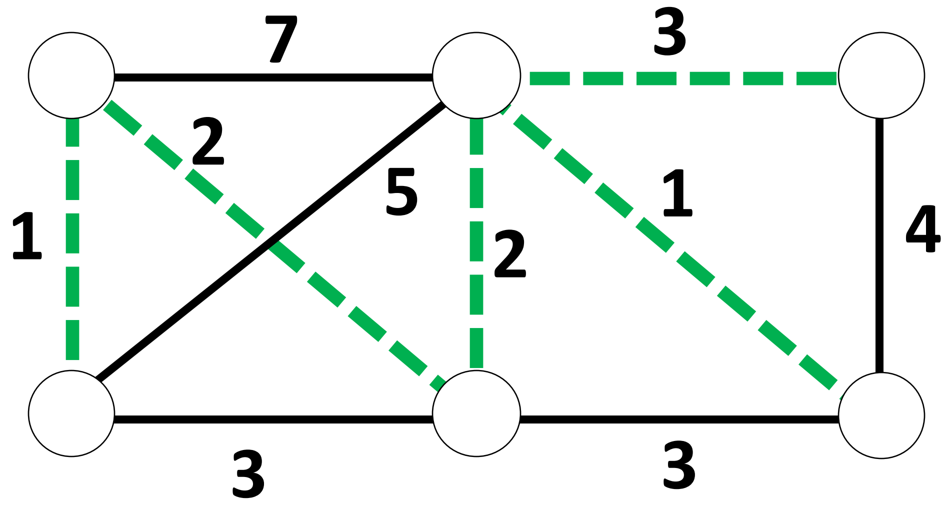
Running time

$$\in O\big(|E| \log(|E|) + |E|(|V| + |E|)\big)$$

$$\in O\big(|E|^2 + |E||V|\big)$$

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.



What are some questions we may have about the algorithm?

1. ~~Is the solution valid? (Does it actually find a spanning tree?)~~
2. ~~What is the running time?~~
3. Is the solution optimal?

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

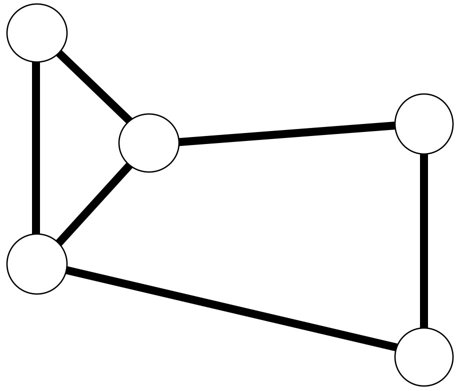
Proof of optimality: T is an MST, because???

MST Cut Property

Assume unique edge costs.

<u>Lemma:</u> Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

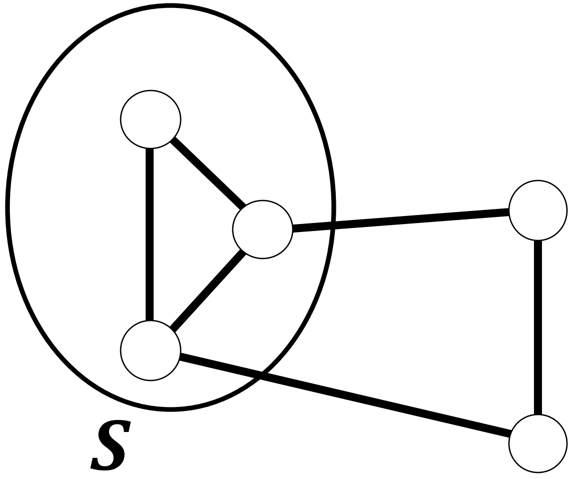
<u>Proof:</u>



MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

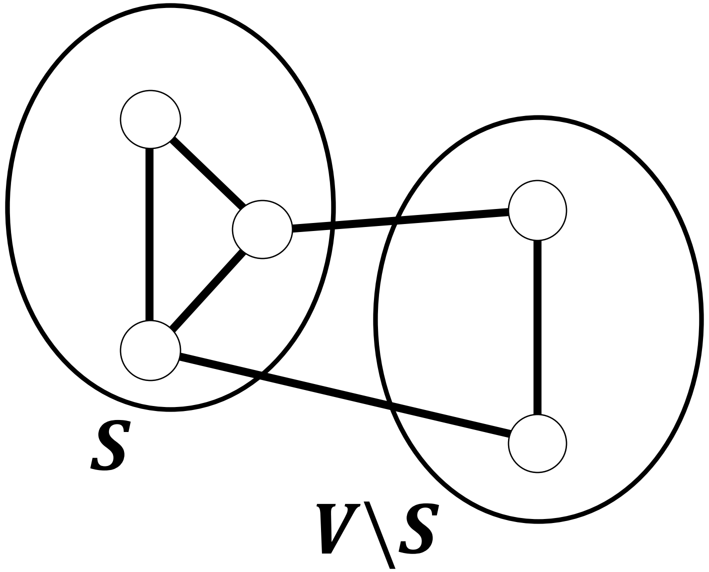
Proof:



MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

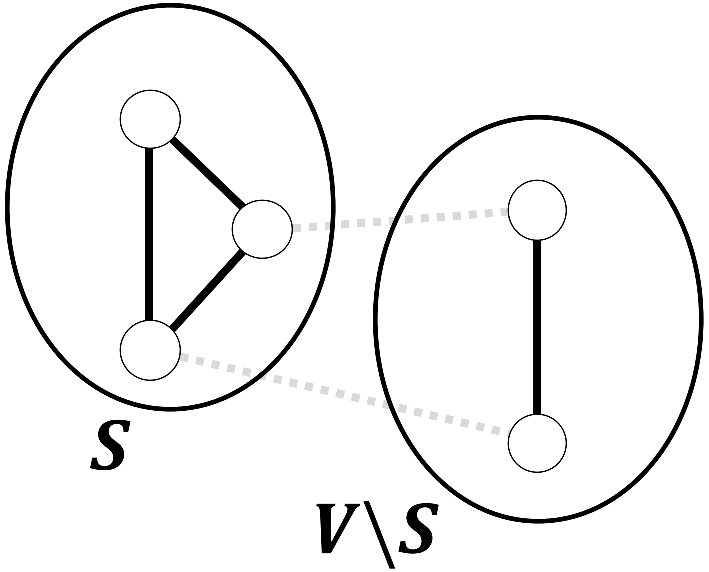
Proof:



MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V\backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V\backslash S$ (otherwise it would not be a tree).

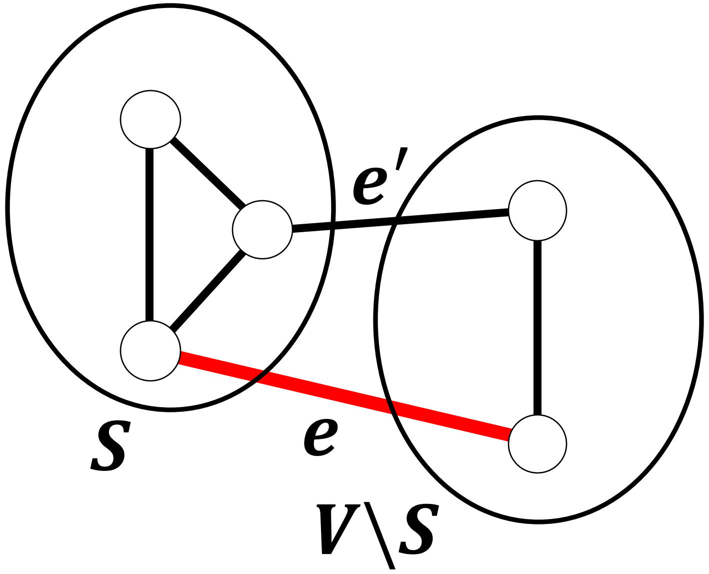


MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let $\boldsymbol{e}$ be the cheapest edge between S and $V \backslash S$.



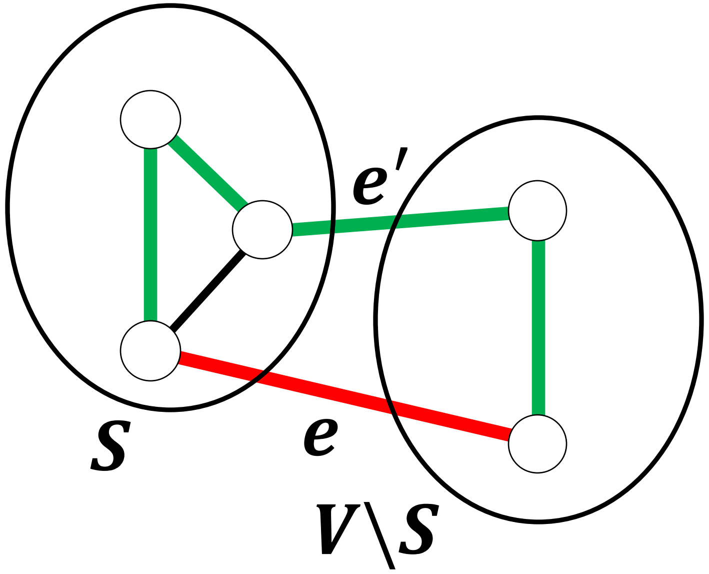
MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V\backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V\backslash S$ (otherwise it would not be a tree).

Let $\boldsymbol{e}$ be the cheapest edge between S and $V\backslash S$.

Suppose $\boldsymbol{T}$ is a spanning tree that does not include $\boldsymbol{e}$.



MST Cut Property

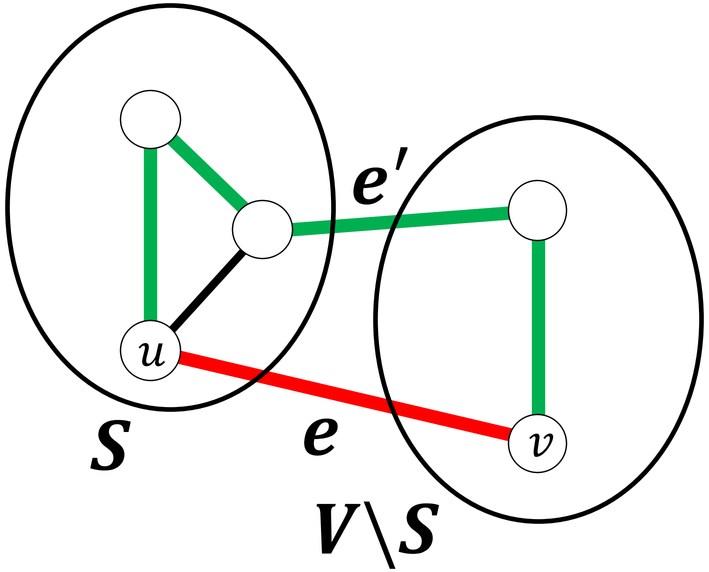
Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let $\boldsymbol{e}$ be the cheapest edge between S and $V \backslash S$.

Suppose $\boldsymbol{T}$ is a spanning tree that does not include $\boldsymbol{e}$. Then:

1. $\boldsymbol{T} \cup \{\boldsymbol{e}\}$ must have a cycle. Because?



MST Cut Property

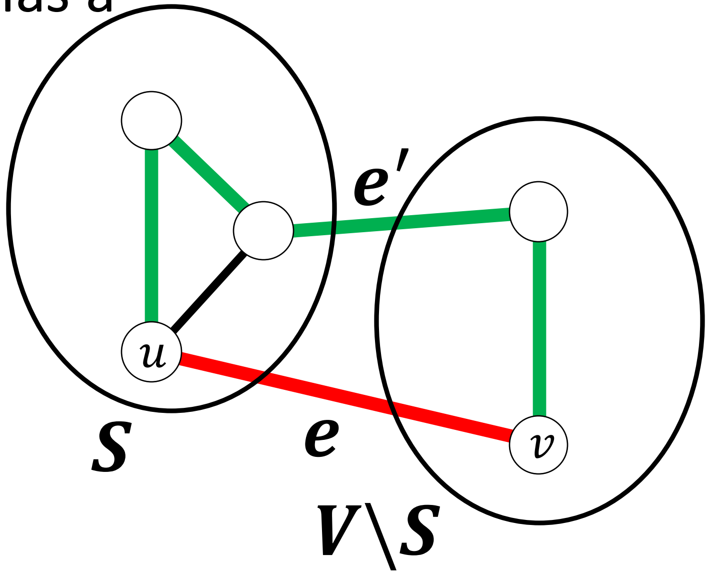
Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let $\boldsymbol{e}$ be the cheapest edge between S and $V \backslash S$.

Suppose $\boldsymbol{T}$ is a spanning tree that does not include $\boldsymbol{e}$. Then:

1. $\boldsymbol{T} \cup \{\boldsymbol{e}\}$ must have a cycle. (Since spanning tree $\boldsymbol{T}$ already has a path between u and v, adding $\boldsymbol{e}$ will create a cycle.)



MST Cut Property

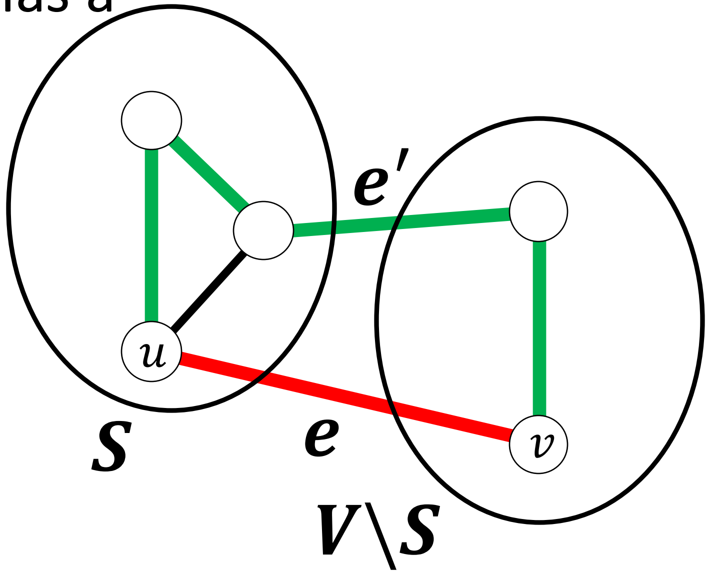
Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let $\boldsymbol{e}$ be the cheapest edge between S and $V \backslash S$.

Suppose $\boldsymbol{T}$ is a spanning tree that does not include $\boldsymbol{e}$. Then:

1. $\boldsymbol{T} \cup \{\boldsymbol{e}\}$ must have a cycle. (Since spanning tree $\boldsymbol{T}$ already has a path between u and v, adding $\boldsymbol{e}$ will create a cycle.)
2. That cycle must have another edge e' between S and $V \backslash S$. (Since there must be a path from $u \in S$ to $v \in V \backslash S$ in $\boldsymbol{T}$)



MST Cut Property

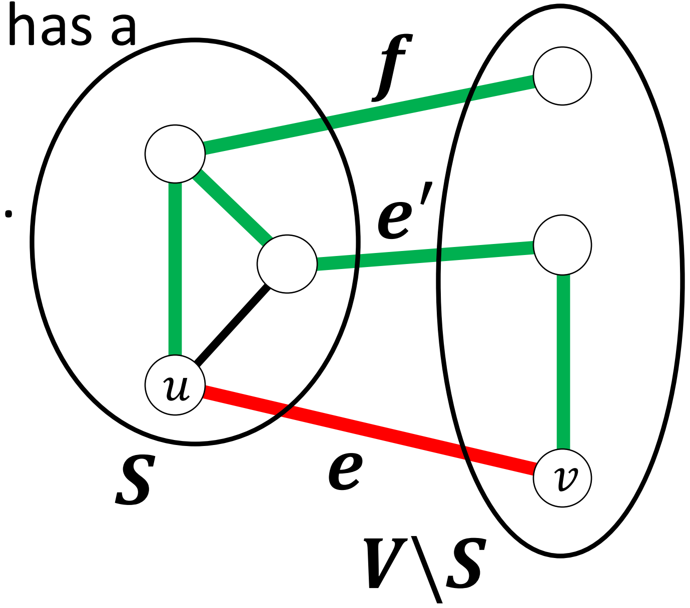
Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let $\boldsymbol{e}$ be the cheapest edge between S and $V \backslash S$.

Suppose $\boldsymbol{T}$ is a spanning tree that does not include $\boldsymbol{e}$. Then:

1. $\boldsymbol{T} \cup \{\boldsymbol{e}\}$ must have a cycle. (Since spanning tree $\boldsymbol{T}$ already has a path between u and v, adding $\boldsymbol{e}$ will create a cycle.)
2. That cycle must have another edge e' between S and $V \backslash S$. (Since there must be a path from $u \in S$ to $v \in V \backslash S$ in $\boldsymbol{T}$)



Need to make sure we pick an edge between S and $V \backslash S$ on the cycle!

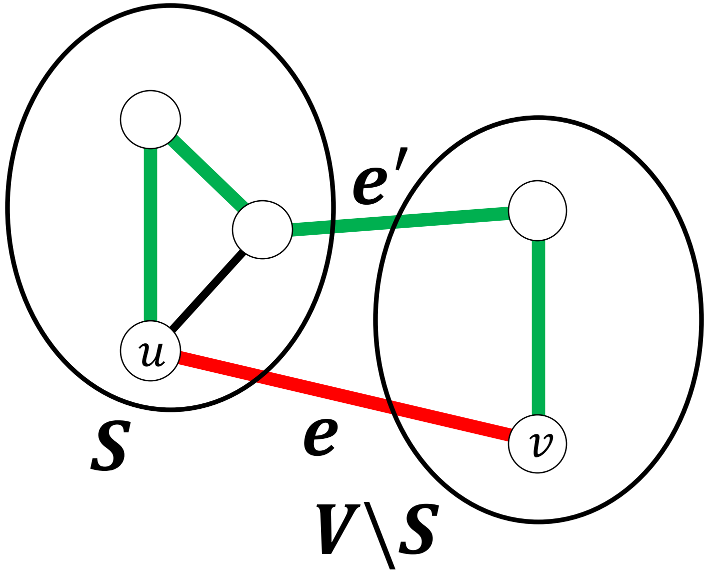
MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let $\boldsymbol{e}$ be the cheapest edge between S and $V \backslash S$.

Suppose $\boldsymbol{T}$ is a spanning tree that does not include $\boldsymbol{e}$. $\boldsymbol{T} \cup \{\boldsymbol{e}\}$ must have a cycle and that cycle must have another edge e' between S and $V \backslash S$.



MST Cut Property

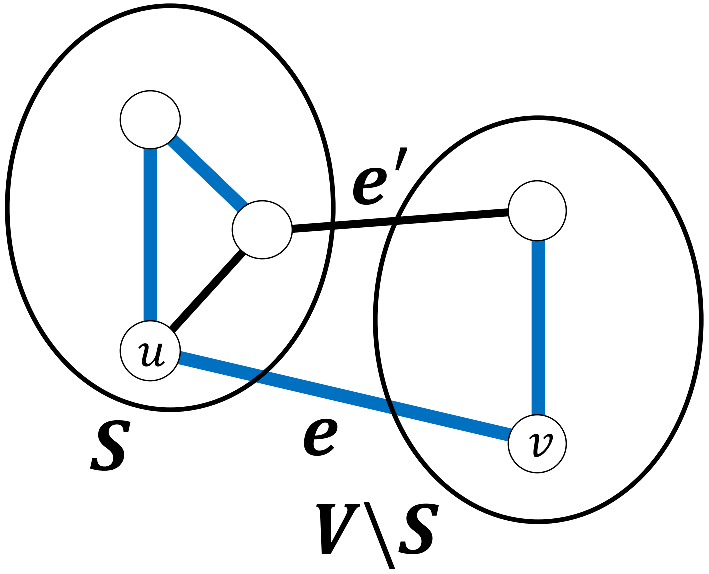
Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let e be the cheapest edge between S and $V \backslash S$.

Suppose T is a spanning tree that does not include e. $T \cup \{e\}$ must have a cycle and that cycle must have another edge e' between S and $V \backslash S$.

Remove e' to form $\boldsymbol{T'} = T \cup \{e\} \backslash \{e'\}$.



e'
u
v
S
e
$V \backslash S$