MATH 322 – Graph Theory
Fall Term 2021

Notes for Lecture 12

Thursday, October 14

More criteria/methods for determining definitively the parameters $\lambda(G)$ and $\kappa(G)$

(and whether (some of) the inequalities in $\kappa(G) \leqslant \lambda(G) \leqslant \delta(G)$ are strict)

Local Vertex Connectivity

Definition

Let G be a connected graph of order n (other than K_n), and let u, v be two **non-adjacent** vertices of G.

A *vertex cut for u and v* is a subset V' of $V(G) \setminus \{u, v\}$ with the property that

$$\text{there is no } u - v \text{ path in } G - V'.$$

The *local vertex connectivity $\kappa(u, v)$* is the **minimum cardinality of a vertex cut for u and v**.

Note. It's not hard to convince ourselves that $\kappa(u, v) = \kappa(v, u)$.

Local Vertex Connectivity

Definition

Let G be a connected graph of order n (other than K_n), and let u, v be two **non-adjacent** vertices of G.

A *vertex cut for u and v* is a subset V' of $V(G) \setminus \{u, v\}$ with the property that

$$\text{there is no } u - v \text{ path in } G - V'.$$

The *local vertex connectivity $\kappa(u, v)$* is the **minimum cardinality of a vertex cut for u and v**.

Note. It's not hard to convince ourselves that $\kappa(u, v) = \kappa(v, u)$.

Important Observation

We have that $\kappa(G)$ equals the minimum of the quantities $\kappa(u, v)$ that we obtain if we consider all pairs (u, v) of **non-adjacent** vertices of G:

$$\kappa(G) = \min\big\{\kappa(u, v) : u, v \in V(G),\ u \neq v,\ uv \notin E(G)\big\}.$$

$$\kappa(G) = \min\{\kappa(u, v) : u, v \in V(G),\ u \neq v,\ uv \notin E(G)\}.$$

Local Edge Connectivity

Definition

Let G be a connected graph of order n, and let w, z be two vertices of G.

An *edge cut for w and z* is a subset E' of $E(G)$ with the property that

$$\text{there is no } w - z \text{ path in } G - E'.$$

The *local edge connectivity $\lambda(w, z)$* is the **minimum cardinality of an edge cut for w and z**.

Note. It's not hard to convince ourselves that $\lambda(w, z) = \lambda(z, w)$.

Local Edge Connectivity

Definition

Let G be a connected graph of order n, and let w, z be two vertices of G.

An *edge cut for w and z* is a subset E' of $E(G)$ with the property that

$$\text{there is no } w - z \text{ path in } G - E'.$$

The *local edge connectivity $\lambda(w, z)$* is the **minimum cardinality of an edge cut for w and z**.

Note. It's not hard to convince ourselves that $\lambda(w, z) = \lambda(z, w)$.

Important Observation

We have that $\lambda(G)$ equals the minimum of the quantities $\lambda(w, z)$ that we obtain if we consider all pairs (w, z) of different vertices of G:

$$\lambda(G) = \min\{\lambda(w, z) : w, z \in V(G),\ w \neq z\}.$$

$$\lambda(G) = \min\{\lambda(w, z) : w, z \in V(G),\ w \neq z\}.$$

But is there an efficient way to compute
the local vertex connectivities $\kappa(u, v)$
and the local edge connectivities $\lambda(w, z)$
for a given graph G?

But is there an efficient way to compute
the local vertex connectivities $\kappa(u, v)$
and the local edge connectivities $\lambda(w, z)$
for a given graph G?

Idea discussed during the last lecture:

Looking at paths connecting different vertices in a given connected graph G, and 'counting' how many such paths we can find for the various pairs of vertices we can consider,

and also 'measuring' 'how different' these paths are,

can give us a good idea about the parameters $\kappa(G)$ and $\lambda(G)$ (and also about the local vertex and edge connectivities of G).

Disjoint Paths

Definition 1

Let G be a graph, and let u, v be two vertices of G. Suppose that $P_1, P_2, \ldots, P_l$ are l different $u-v$ paths in G.

The collection $\{P_1, P_2, \ldots, P_l\}$ is called *internally disjoint* (or alternatively *vertex-disjoint*) if, for any two different paths in this collection, **their only common vertices** are the vertices u and v (in other words, if none of the internal vertices in any one of these paths appears in another path too).

Disjoint Paths

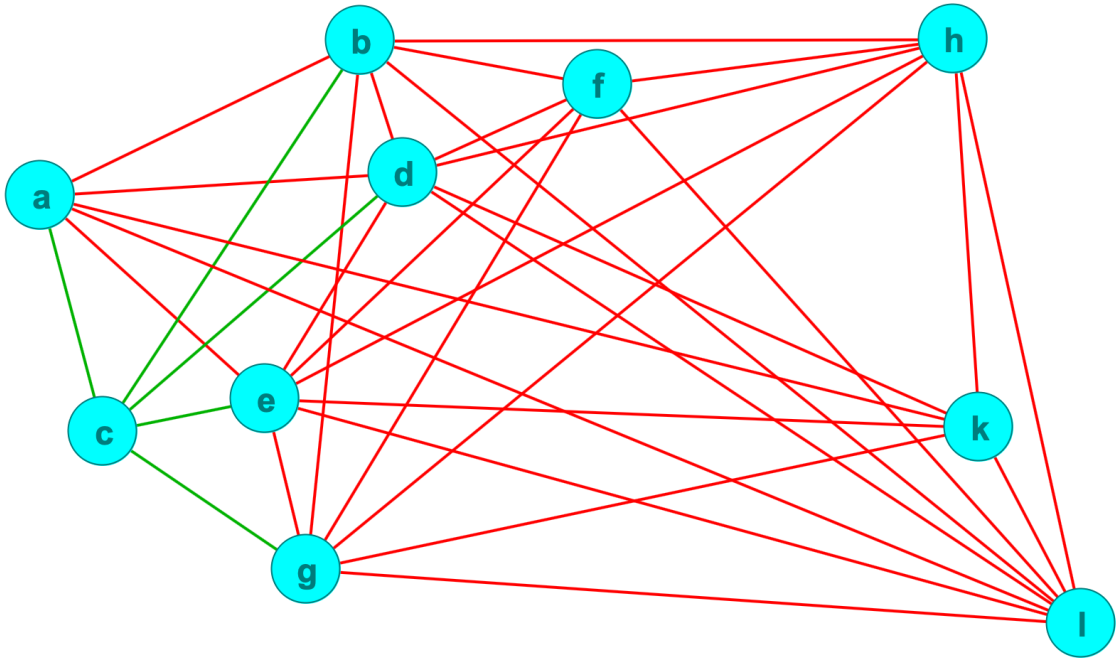
Definition 1

Let G be a graph, and let u, v be two vertices of G. Suppose that $P_1, P_2, \ldots, P_l$ are l different $u-v$ paths in G.

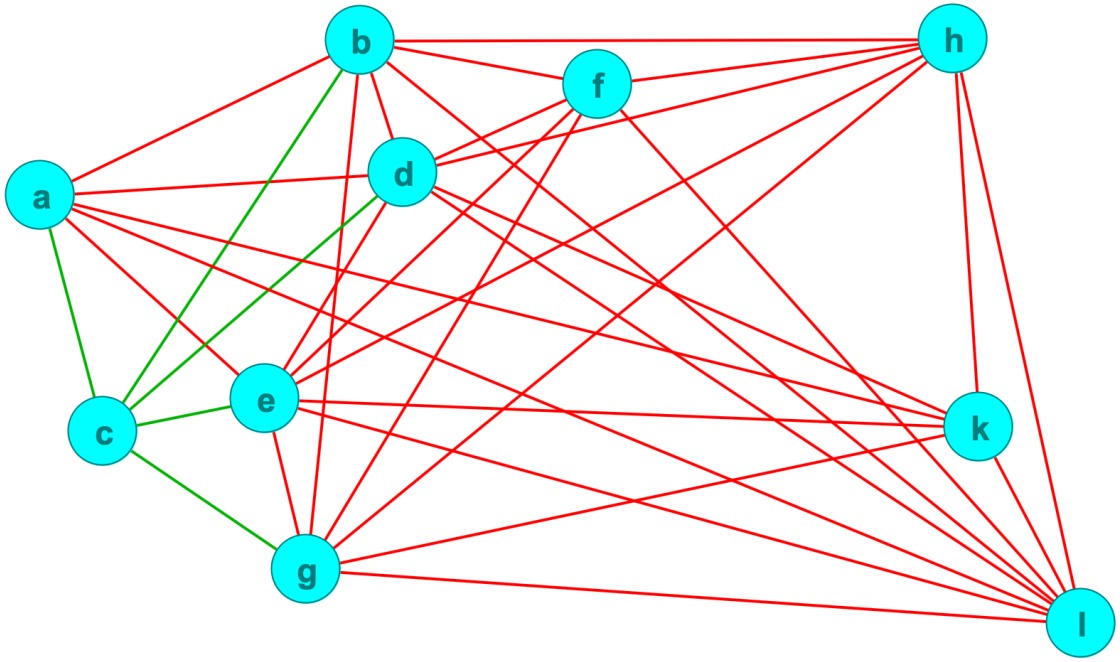
The collection $\{P_1, P_2, \ldots, P_l\}$ is called *internally disjoint* (or alternatively *vertex-disjoint*) if, for any two different paths in this collection, **their only common vertices** are the vertices u and v (in other words, if none of the internal vertices in any one of these paths appears in another path too).

We write $\kappa'(u, v)$ for the maximum possible cardinality that an internally disjoint collection of $u-v$ paths in G can have.

Back to examples

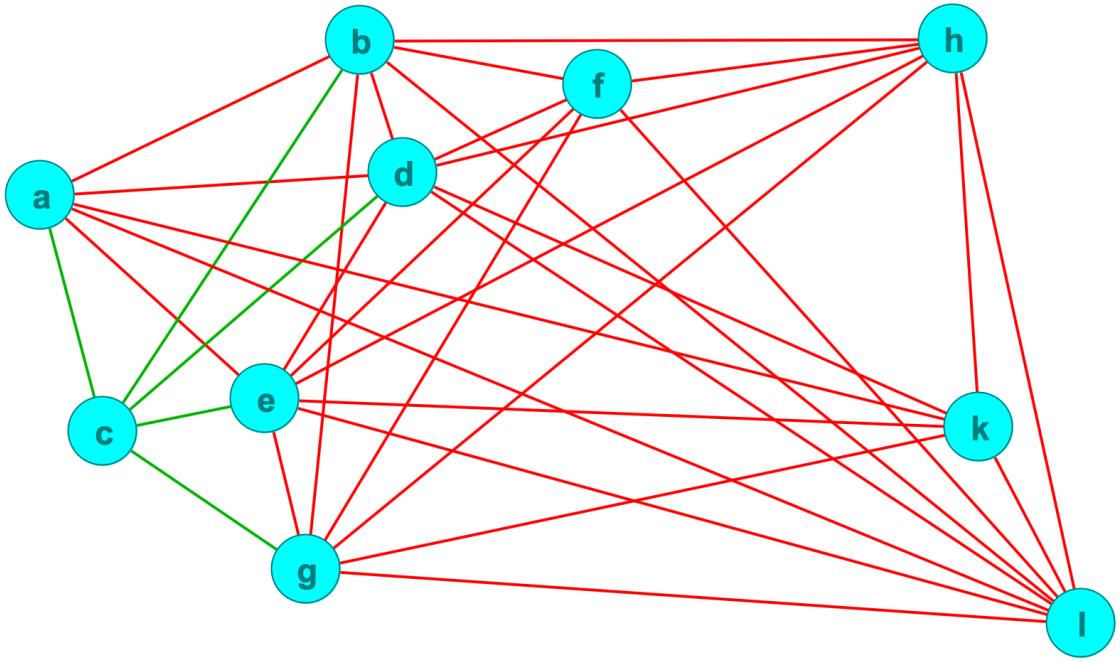


Back to examples



Question 1. Do we have $\kappa'(a, c) \geqslant 5$?

Back to examples



Question 1. Do we have $\kappa'(a, c) \geqslant 5$?

Question 2. Can you find 5 pairwise internally disjoint $c-h$ paths?

Disjoint Paths (cont.)

Definition 2

Let G be a graph, and let w, z be two vertices of G. Suppose that $Q_1, Q_2, \ldots, Q_s$ are s different $w-z$ paths in G.

The collection $\{Q_1, Q_2, \ldots, Q_s\}$ is called *edge-disjoint* if, for any two different paths Q_i, Q_j in this collection, Q_i and Q_j contain **no** common edges.

Disjoint Paths (cont.)

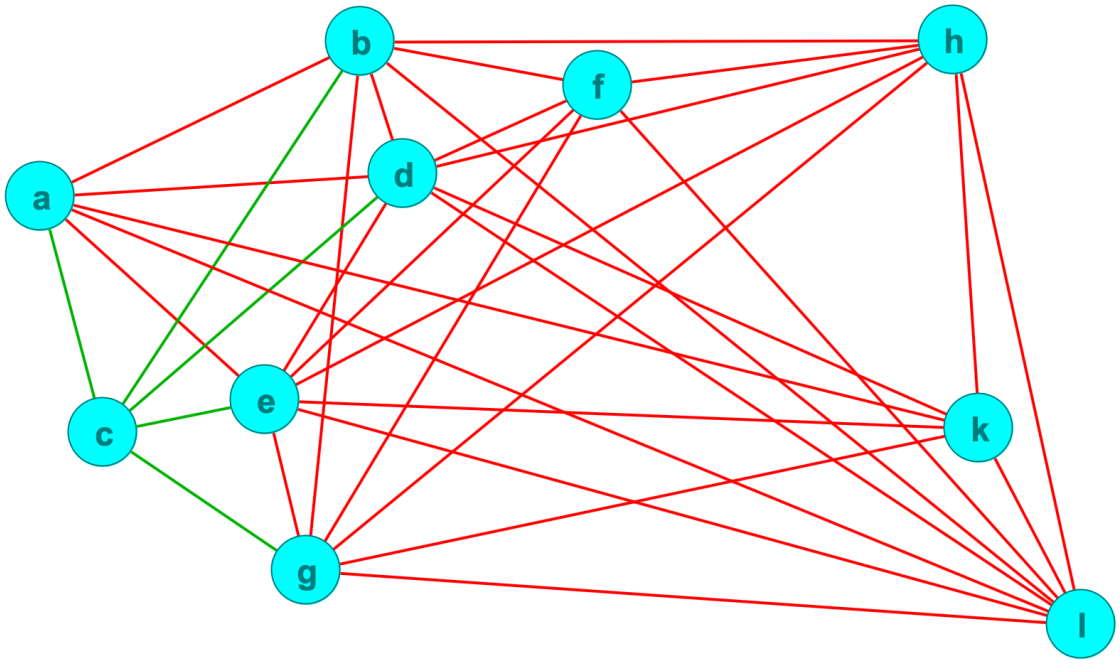
Definition 2

Let G be a graph, and let w, z be two vertices of G. Suppose that $Q_1, Q_2, \ldots, Q_s$ are s different $w-z$ paths in G.

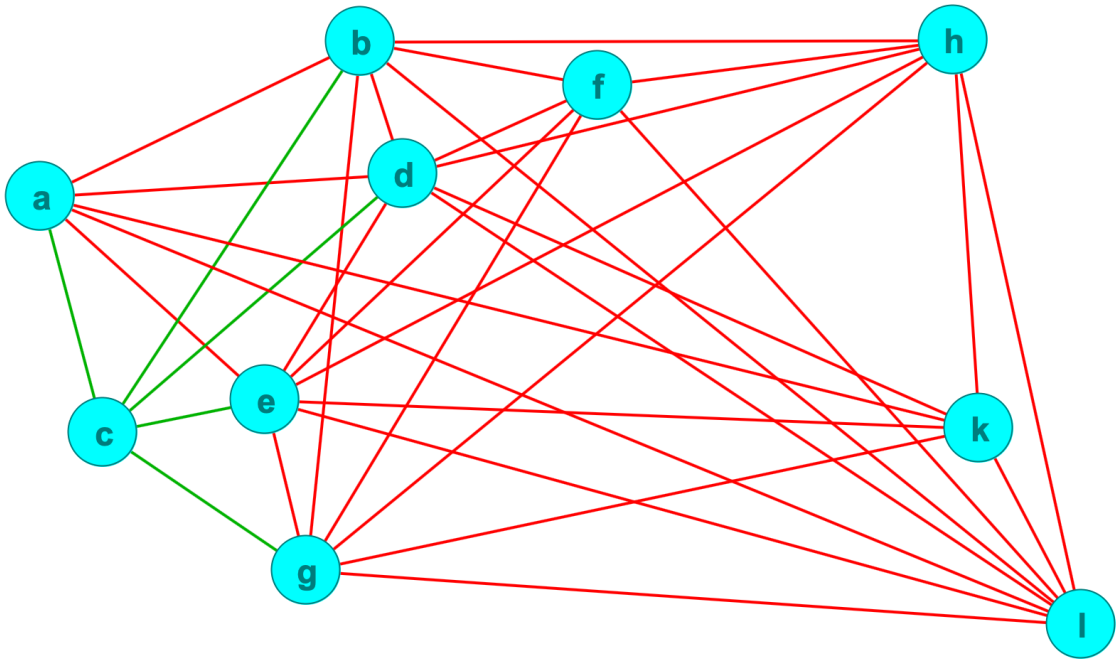
The collection $\{Q_1, Q_2, \ldots, Q_s\}$ is called *edge-disjoint* if, for any two different paths Q_i, Q_j in this collection, Q_i and Q_j contain **no** common edges.

We write $\lambda'(w, z)$ for the maximum possible cardinality that an edge-disjoint collection of $w-z$ paths in G can have.

Back to examples



Back to examples



Question 3. Focusing e.g. on the vertices g and h, can you find edge-disjoint paths that connect them which are not vertex-disjoint (equivalently, internally disjoint)?

A very useful theorem

Menger's theorem *(vertex form)*

Let G be a connected graph of order n (other than K_n), and let u, v be two **non-adjacent** vertices of G.

Then the **minimum** cardinality of a vertex cut for u and v equals the **maximum** cardinality of an internally disjoint collection of $u-v$ paths in G. In other words,

$$\kappa(u, v) = \kappa'(u, v).$$

A very useful theorem

Menger's theorem *(vertex form)*

Let G be a connected graph of order n (other than K_n), and let u, v be two **non-adjacent** vertices of G.

Then the **minimum** cardinality of a vertex cut for u and v equals the **maximum** cardinality of an internally disjoint collection of $u-v$ paths in G. In other words,

$$\kappa(u, v) = \kappa'(u, v).$$

Menger's theorem *(edge form)*

Let G be a connected graph of order n, and let w, z be two vertices of G. Then the **minimum** cardinality of an edge cut for w and z equals the **maximum** cardinality of an edge-disjoint collection of $w-z$ paths in G.

In other words,

$$\lambda(w, z) = \lambda'(w, z).$$

Why the theorem is so useful to us

Recall first:

$$\kappa(G) = \min\{\kappa(u,v) : u,v \in V(G),\ u \neq v,\ uv \notin E(G)\},$$
$$\text{while} \quad \lambda(G) = \min\{\lambda(w,z) : w,z \in V(G),\ w \neq z\}.$$

Important Corollary of Menger's Theorem

- Let G be a connected graph of order n (other than K_n). Then $\kappa(G) \geqslant t$ if and only if, for any two **non-adjacent** vertices u, v of G, we can find **at least** t pairwise internally disjoint paths in G that connect u and v.

Why the theorem is so useful to us

Recall first:

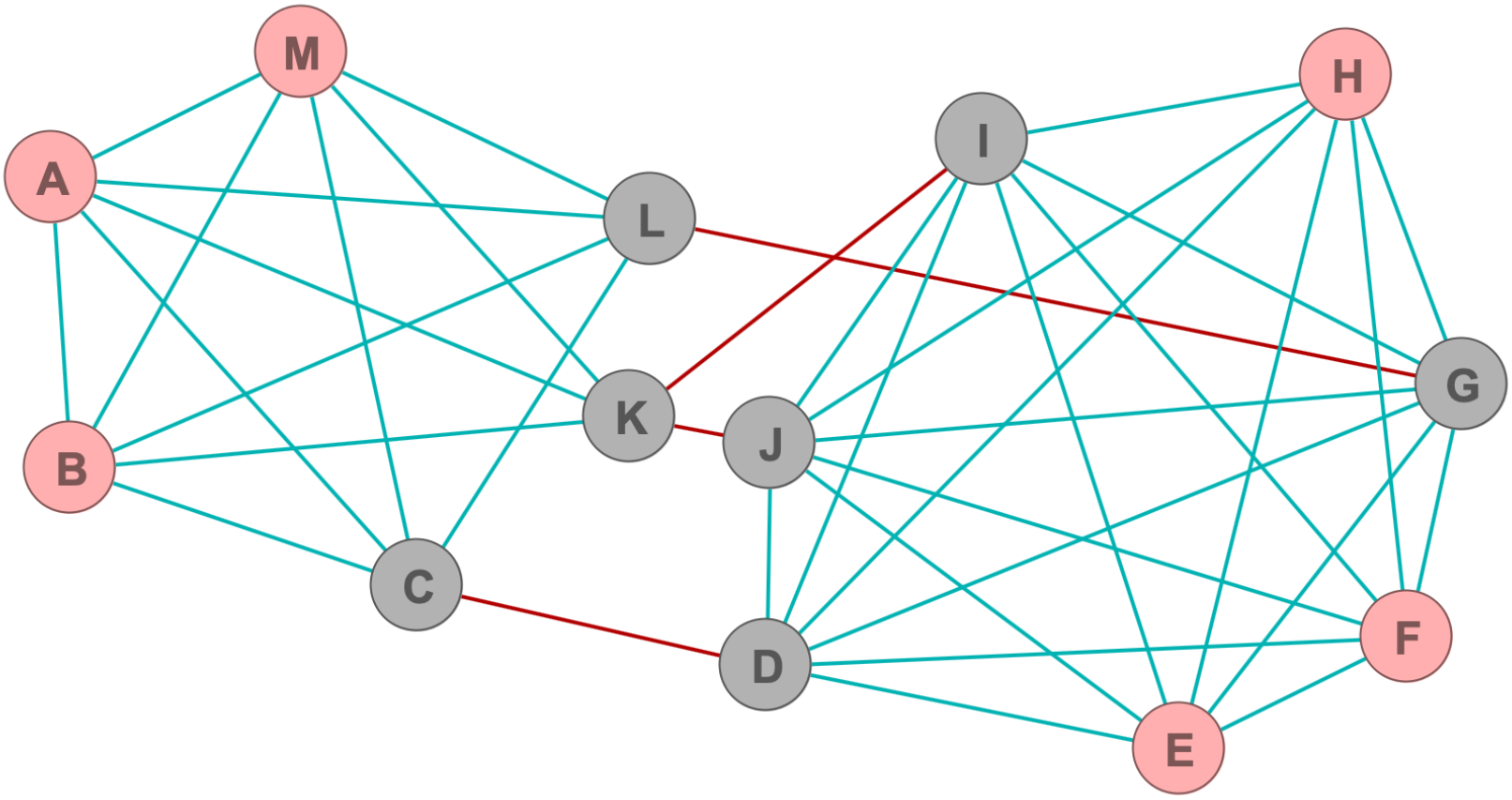
$$\kappa(G) = \min\{\kappa(u,v) : u,v \in V(G),\ u \neq v,\ uv \notin E(G)\},$$
$$\text{while} \quad \lambda(G) = \min\{\lambda(w,z) : w,z \in V(G),\ w \neq z\}.$$

Important Corollary of Menger's Theorem

- Let G be a connected graph of order n (other than K_n). Then $\kappa(G) \geqslant t$ if and only if, for any two **non-adjacent** vertices u, v of G, we can find **at least** t pairwise internally disjoint paths in G that connect u and v.

- Let H be a connected graph of order n. Then $\lambda(H) \geqslant s$ if and only if, for any two different vertices w, z of H, we can find **at least** s pairwise edge-disjoint paths in H that connect w and z.

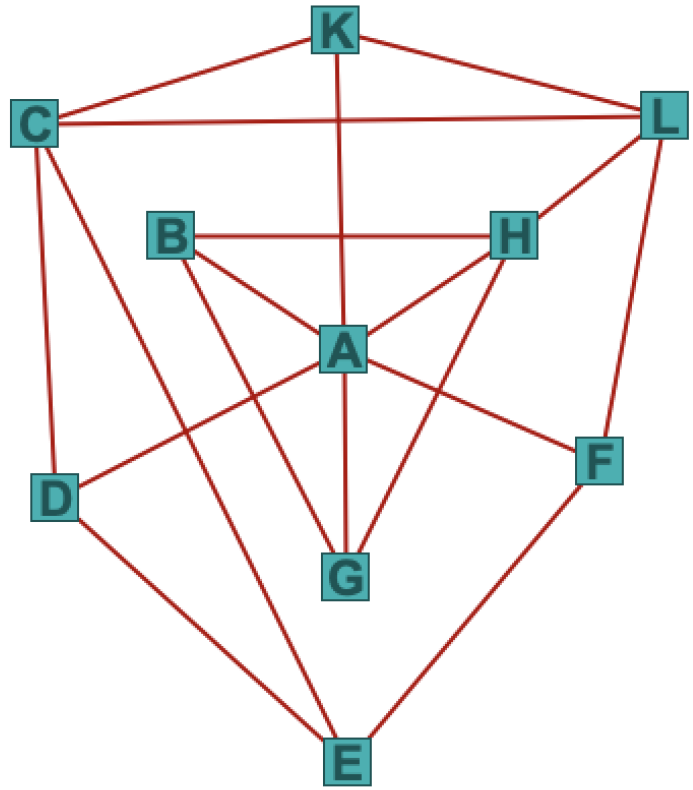
Relying on Menger's theorem to handle previous examples



Determine precisely $\lambda(G)$ and $\kappa(G)$.

Relying on Menger's theorem for to handle previous examples (cont.)

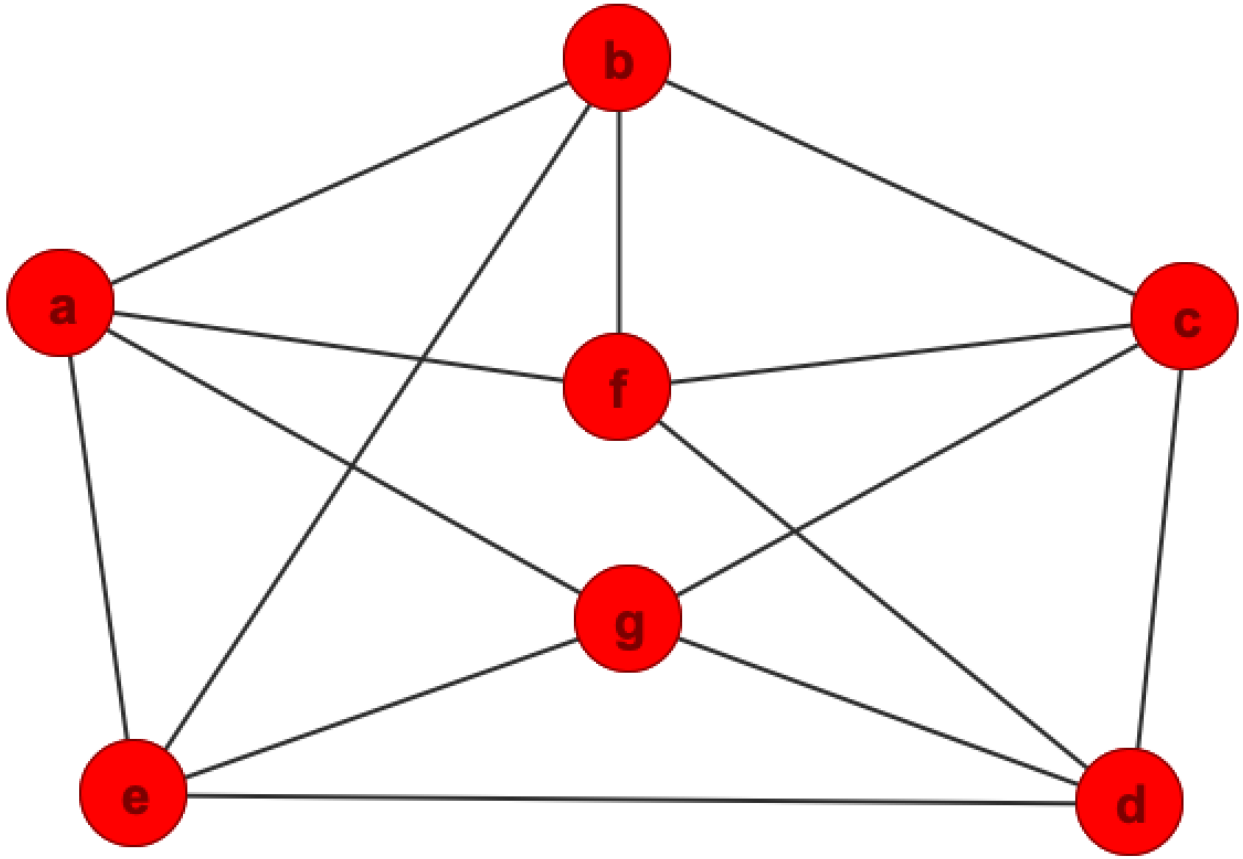
Past Exam Problem.



(a) Show that $\kappa(G_0) = 2$. Give a full justification.

(b) What is $\lambda(G_0)$? Determine it precisely, and justify your answer fully.

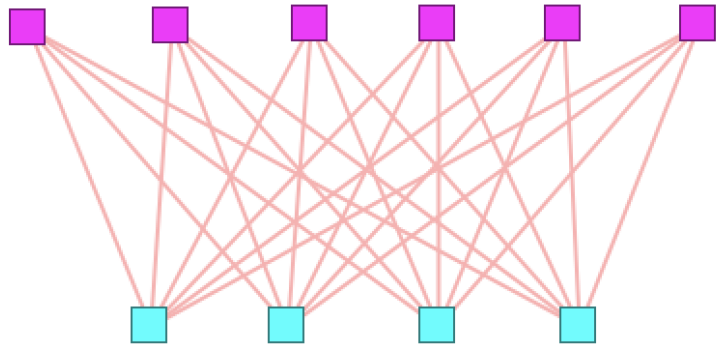
Relying on Menger's theorem for to handle previous examples (cont.)

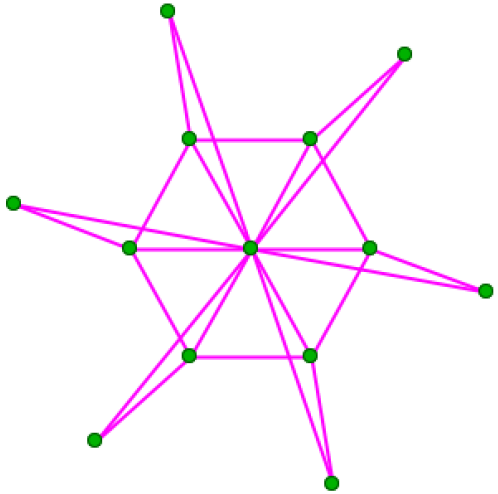


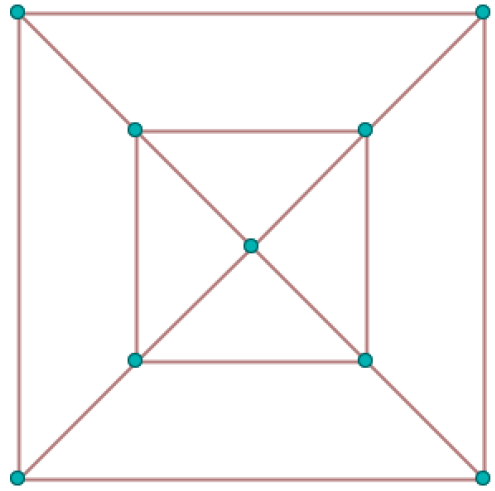
Question. What is $\kappa(G)$ and $\lambda(G)$ here?

More examples

Now we can determine the vertex connectivity and the edge connectivity of all different graphs that we have seen (either by working with the definitions directly (that is, by looking for (all 'smallest possible') vertex and edge cuts), or by using Menger's theorem too and thus counting different internally disjoint or edge-disjoint paths).







Next Main Topic:

What (other) ways/notions can we come up with to capture/describe how 'efficiently' connected a given connected graph is?

Spanning tree of a graph

Definition

Let G be a connected graph, and let T be a subgraph of G.
T is called a *spanning tree* of G if

- T is a tree

Spanning tree of a graph

Definition

Let G be a connected graph, and let T be a subgraph of G.
T is called a *spanning tree* of G if

- T is a tree
- and T contains all vertices of G.

Spanning tree of a graph

Definition

Let G be a connected graph, and let T be a subgraph of G.
T is called a *spanning tree* of G if

- T is a tree
- and T contains all vertices of G.

Observation 1

A spanning tree of G is a *minimal* connected subgraph of G that contains all the vertices of G ("minimal" here means that, if we remove any more edges from T, then we will instead end up with a disconnected subgraph of G).

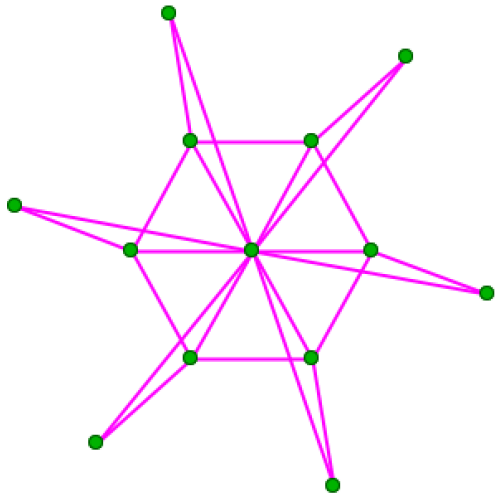
We could think of a spanning tree of a connected graph G as a very simple 'skeletal frame' for the graph.

Observation 2

In general, a connected graph G may have several spanning trees *(in fact, you could check that this is true for all connected graphs which are NOT trees themselves; left as practice; see also next theorem and the way to justify it)*.

Observation 2

In general, a connected graph G may have several spanning trees *(in fact, you could check that this is true for all connected graphs which are NOT trees themselves; left as practice; see also next theorem and the way to justify it)*.

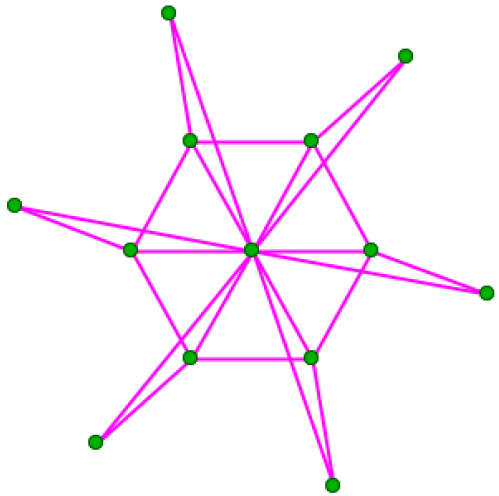


A 'flower' graph

and three spanning trees for it:

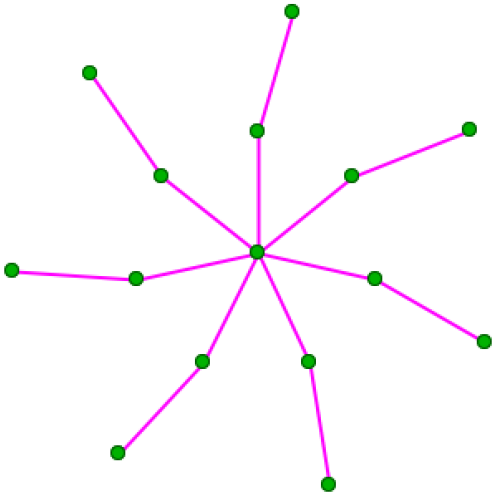
Observation 2

In general, a connected graph G may have several spanning trees *(in fact, you could check that this is true for all connected graphs which are NOT trees themselves; left as practice; see also next theorem and the way to justify it)*.



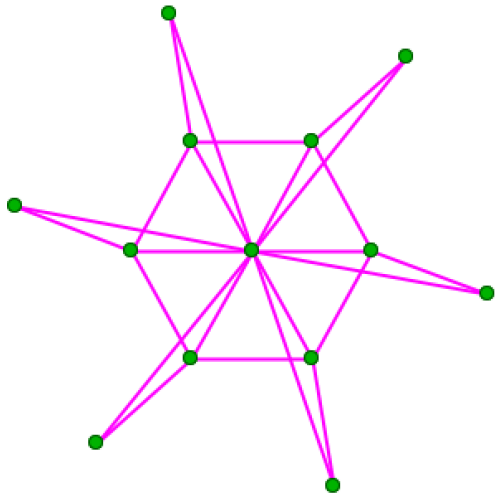
A 'flower' graph

and three spanning trees for it:



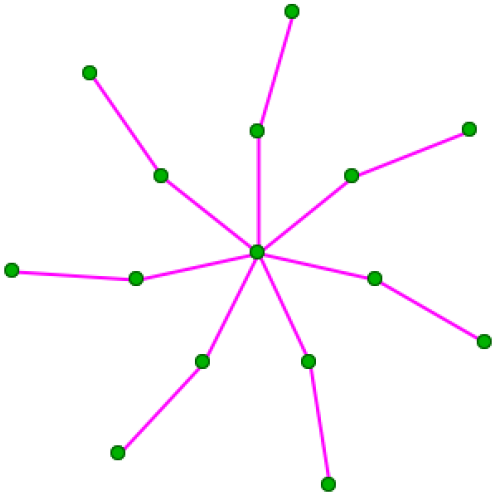
Observation 2

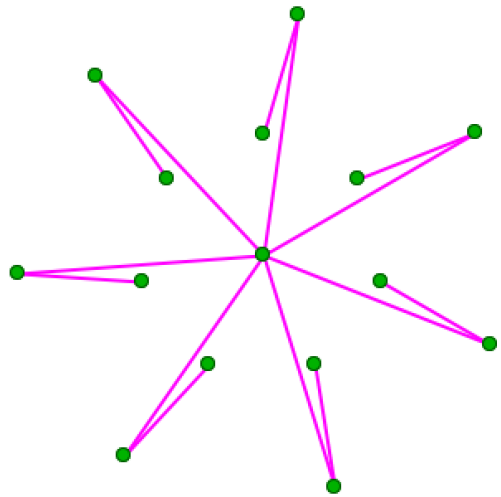
In general, a connected graph G may have several spanning trees *(in fact, you could check that this is true for all connected graphs which are NOT trees themselves; left as practice; see also next theorem and the way to justify it)*.



A 'flower' graph

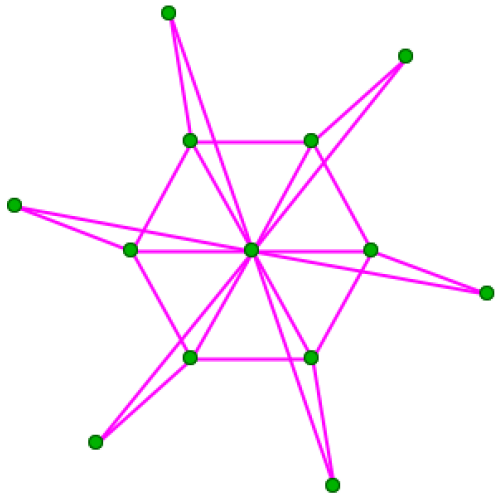
and three spanning trees for it:





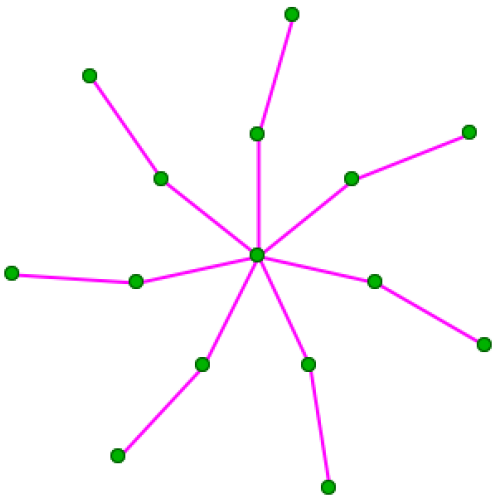
Observation 2

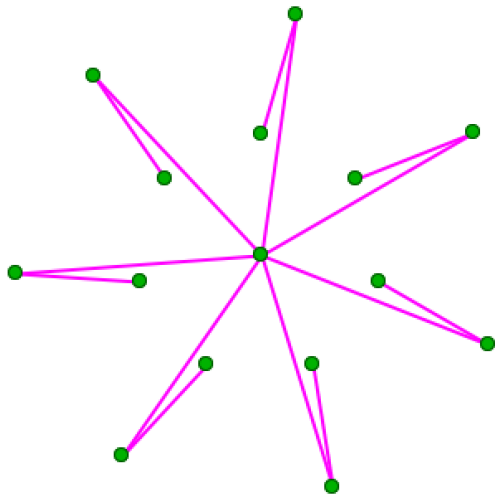
In general, a connected graph G may have several spanning trees *(in fact, you could check that this is true for all connected graphs which are NOT trees themselves; left as practice; see also next theorem and the way to justify it)*.

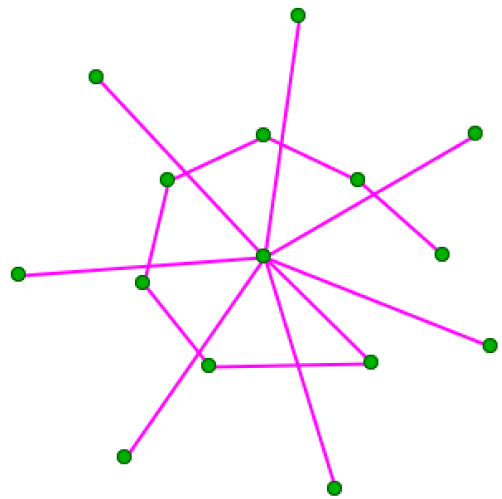


A 'flower' graph

and three spanning trees for it:







Important Observation

Theorem

Every connected graph G has at least one spanning tree.

Important Observation

Theorem

Every connected graph G has at least one spanning tree.

Proof of the theorem. Assume that G has order n with $n \geqslant 3$ (since otherwise G is already a tree and the conclusion is obvious).

Important Observation

Theorem

Every connected graph G has at least one spanning tree.

Proof of the theorem. Assume that G has order n with $n \geqslant 3$ (since otherwise G is already a tree and the conclusion is obvious).

We can prove the theorem using induction in the size of G.

Important Observation

Theorem

Every connected graph G has at least one spanning tree.

Proof of the theorem. Assume that G has order n with $n \geqslant 3$ (since otherwise G is already a tree and the conclusion is obvious).

We can prove the theorem using induction in the size of G.

- **Base Case:** *What is the minimum size to consider here?*

Important Observation

Theorem

Every connected graph G has at least one spanning tree.

Proof of the theorem. Assume that G has order n with $n \geqslant 3$ (since otherwise G is already a tree and the conclusion is obvious).

We can prove the theorem using induction in the size of G.

• **Base Case:** *What is the minimum size to consider here?* Recall that the minimum size of a connected graph on n vertices is $n - 1$. But then, as we saw in Theorem 1a of Lecture 7, if we consider a connected graph G of order n with precisely $n - 1$ edges, this graph will be a tree, and hence the entire graph G will be a spanning tree of itself.

Important Observation

Theorem

Every connected graph G has at least one spanning tree.

Proof of the theorem. Assume that G has order n with $n \geqslant 3$ (since otherwise G is already a tree and the conclusion is obvious).

We can prove the theorem using induction in the size of G.

• **Base Case:** *What is the minimum size to consider here?* Recall that the minimum size of a connected graph on n vertices is $n-1$. But then, as we saw in Theorem 1a of Lecture 7, if we consider a connected graph G of order n with precisely $n-1$ edges, this graph will be a tree, and hence the entire graph G will be a spanning tree of itself.

• **Induction Step:** Assume now that, for some s with $n-1 \leqslant s < \binom{n}{2}$, we have already shown that

every connected graph H on n vertices
which has size s has at least one spanning tree.

Important Observation

Theorem

Every connected graph G has at least one spanning tree.

Proof of the theorem. Assume that G has order n with $n \geqslant 3$ (since otherwise G is already a tree and the conclusion is obvious).

We can prove the theorem using induction in the size of G.

• **Base Case:** *What is the minimum size to consider here?* Recall that the minimum size of a connected graph on n vertices is $n-1$. But then, as we saw in Theorem 1a of Lecture 7, if we consider a connected graph G of order n with precisely $n-1$ edges, this graph will be a tree, and hence the entire graph G will be a spanning tree of itself.

• **Induction Step:** Assume now that, for some s with $n-1 \leqslant s < \binom{n}{2}$, we have already shown that

every connected graph H on n vertices
which has size s has at least one spanning tree.

Consider now a connected graph G on the same set of vertices which has size $s+1$. Then $s+1 > n-1$, and hence G cannot be a tree (recall again Thm 1a of Lec 7).

But this implies that there exists at least one edge of G (say edge e_0) which is NOT a bridge (recall also Thm 2a of Lec 10).

Important Observation

Theorem

Every connected graph G has at least one spanning tree.

Proof of the theorem. Assume that G has order n with $n \geqslant 3$ (since otherwise G is already a tree and the conclusion is obvious).

We can prove the theorem using induction in the size of G.

• **Base Case:** *What is the minimum size to consider here?* Recall that the minimum size of a connected graph on n vertices is $n-1$. But then, as we saw in Theorem 1a of Lecture 7, if we consider a connected graph G of order n with precisely $n-1$ edges, this graph will be a tree, and hence the entire graph G will be a spanning tree of itself.

• **Induction Step:** Assume now that, for some s with $n-1 \leqslant s < \binom{n}{2}$, we have already shown that

every connected graph H on n vertices
which has size s has at least one spanning tree.

Consider now a connected graph G on the same set of vertices which has size $s+1$. Then $s+1 > n-1$, and hence G cannot be a tree (recall again Thm 1a of Lec 7).

But this implies that there exists at least one edge of G (say edge e_0) which is NOT a bridge (recall also Thm 2a of Lec 10). Then the subgraph $G - e_0$ will be connected, and will have size equal to $e(G) - 1 = (s+1) - 1 = s$. Thus by the Inductive Hypothesis, $G - e_0$ will have at least one spanning tree T_1.

Important Observation

Theorem

Every connected graph G has at least one spanning tree.

Proof of the theorem. Assume that G has order n with $n \geqslant 3$ (since otherwise G is already a tree and the conclusion is obvious).

We can prove the theorem using induction in the size of G.

• **Base Case:** *What is the minimum size to consider here?* Recall that the minimum size of a connected graph on n vertices is $n-1$. But then, as we saw in Theorem 1a of Lecture 7, if we consider a connected graph G of order n with precisely $n-1$ edges, this graph will be a tree, and hence the entire graph G will be a spanning tree of itself.

• **Induction Step:** Assume now that, for some s with $n-1 \leqslant s < \binom{n}{2}$, we have already shown that

every connected graph H on n vertices
which has size s has at least one spanning tree.

Consider now a connected graph G on the same set of vertices which has size $s+1$. Then $s+1 > n-1$, and hence G cannot be a tree (recall again Thm 1a of Lec 7).

But this implies that there exists at least one edge of G (say edge e_0) which is NOT a bridge (recall also Thm 2a of Lec 10). Then the subgraph $G - e_0$ will be connected, and will have size equal to $e(G) - 1 = (s+1) - 1 = s$. Thus by the Inductive Hypothesis, $G - e_0$ will have at least one spanning tree T_1.

Remark: T_1 is also a spanning tree of G (since $T_1 \subseteq G - e_0 \subseteq G$ and T_1 contains all the vertices in $V(G - e_0) = V(G)$).

'Algorithmic method' suggested by the proof for finding a spanning tree

Let G be a connected graph on n vertices.

- If G is not already a tree, then G contains cycles, and hence it also contains edges which are not bridges.

'Algorithmic method' suggested by the proof for finding a spanning tree

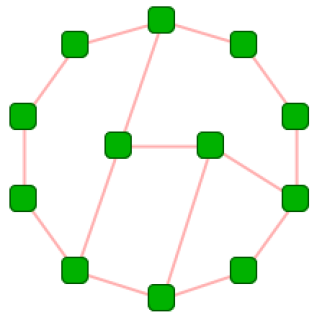
Let G be a connected graph on n vertices.

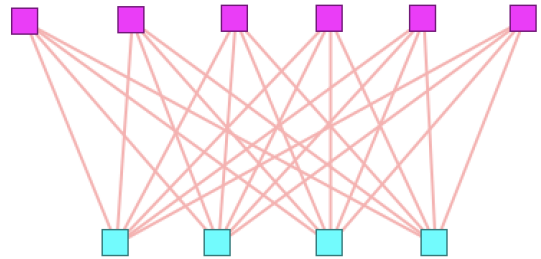
- If G is not already a tree, then G contains cycles, and hence it also contains edges which are not bridges.
- Pick an edge which is not a bridge, and remove it. Check whether the resulting subgraph G_1 is a spanning tree of G. If not, then repeat this step for G_1.

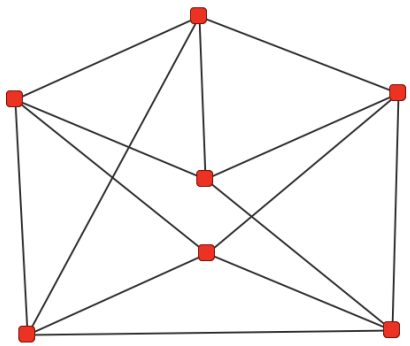
'Algorithmic method' suggested by the proof for finding a spanning tree

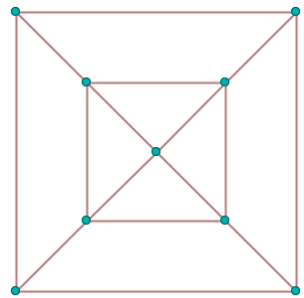
Let G be a connected graph on n vertices.

- If G is not already a tree, then G contains cycles, and hence it also contains edges which are not bridges.
- Pick an edge which is not a bridge, and remove it. Check whether the resulting subgraph G_1 is a spanning tree of G. If not, then repeat this step for G_1.









Practice Question. Find spanning trees for the above graphs.

A variation coming up in applications

Definition

Let $K = (V(K), E(K))$ be a graph. A *weight function* for K is a function

$$w : E(K) \to [0, +\infty)$$

mapping each edge of K to a non-negative real number.

A variation coming up in applications

Definition

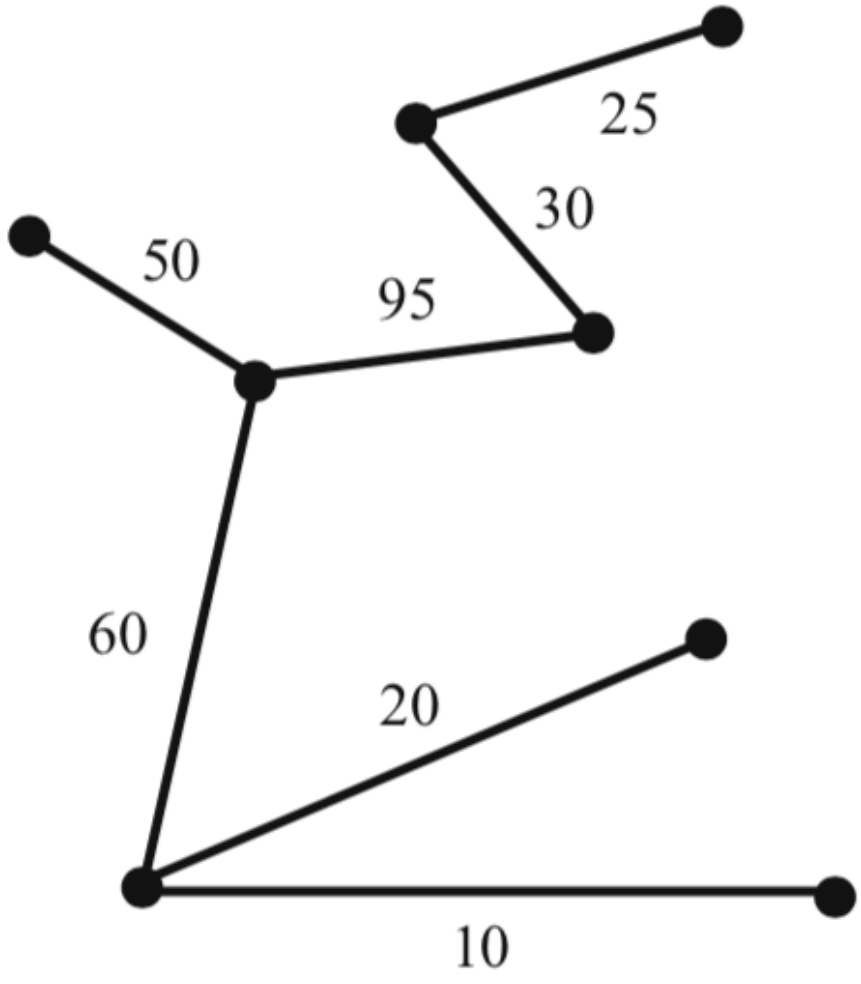
Let $K = (V(K), E(K))$ be a graph. A *weight function* for K is a function

$$w : E(K) \to [0, +\infty)$$

mapping each edge of K to a non-negative real number.

The graph K together with a weight function for it is called a *weighted graph*.

Examples of weighted graphs



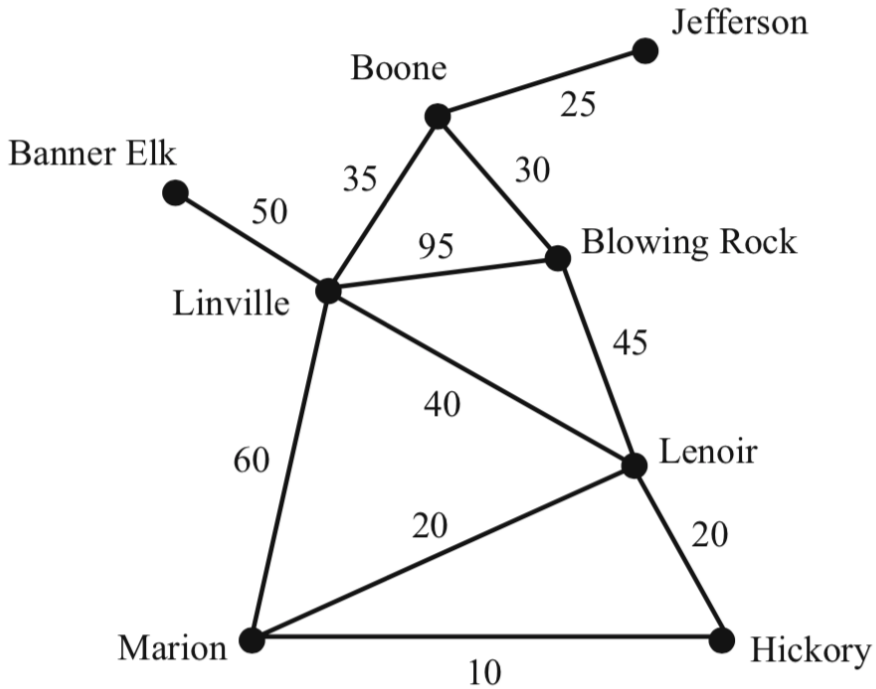
Examples of weighted graphs in applications

Problem that can lead to such an example *(from the Harris-Hirst-Mossinghoff book)*
The North Carolina Department of Transportation (NCDOT) has decided to implement a rapid rail system to serve eight cities in the western part of the state.
— Some of the cities are currently joined by roads or highways, and the state plans to lay the tracks right along (some of) these roads.
— Due to mountainous terrain, some of the roads are steep and curvy, and so laying track along these roads would be difficult and expensive.

Examples of weighted graphs in applications

Problem that can lead to such an example *(from the Harris-Hirst-Mossinghoff book)*
The North Carolina Department of Transportation (NCDOT) has decided to implement a rapid rail system to serve eight cities in the western part of the state.
— Some of the cities are currently joined by roads or highways, and the state plans to lay the tracks right along (some of) these roads.
— Due to mountainous terrain, some of the roads are steep and curvy, and so laying track along these roads would be difficult and expensive.

Due to the above, the NCDOT has hired a consultant to study the roads and to assign a difficulty rating to each one. Here is the graph the consultant has returned:



Examples of weighted graphs in applications

What would be an 'efficient' plan for the NCDOT to follow:
find a spanning tree of this graph.

Examples of weighted graphs in applications

What would be an 'efficient' plan for the NCDOT to follow:
find a spanning tree of this graph.

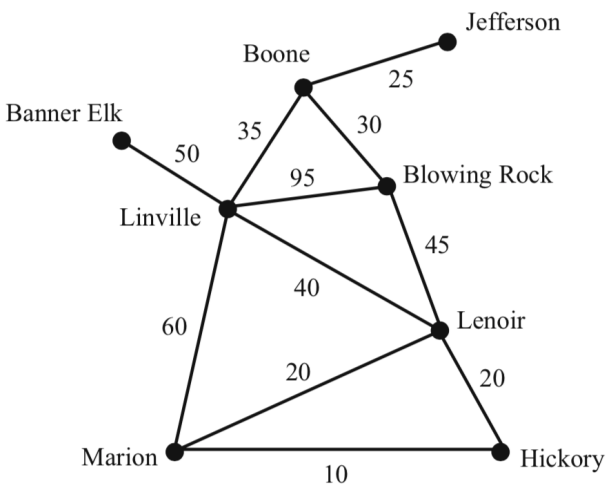
In fact, now it also seems to make sense to 'take even more factors into consideration', and look for a *minimum weight spanning tree*.

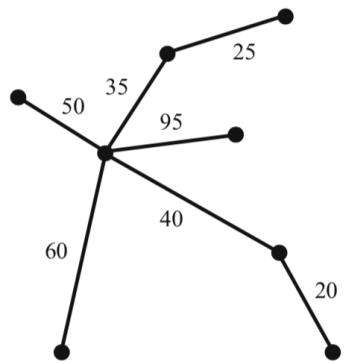
Examples of weighted graphs in applications

What would be an 'efficient' plan for the NCDOT to follow:
find a spanning tree of this graph.

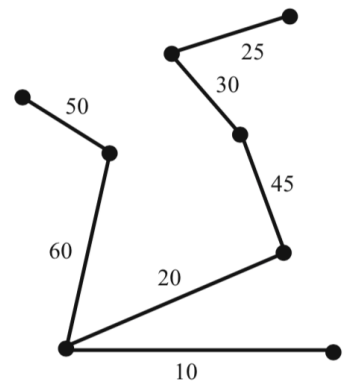
In fact, now it also seems to make sense to 'take even more factors into consideration', and look for a *minimum weight spanning tree*.

Some spanning trees of the above graph:

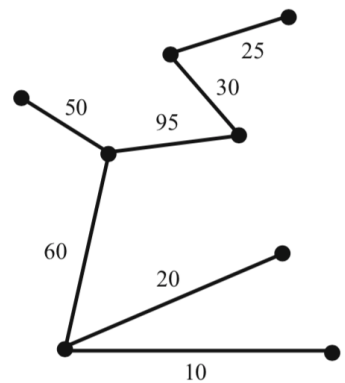




Total Weight = 325



Total Weight = 240



Total Weight = 290

Most ‘efficient’ solution

Most ‘efficient’ plan for the NCDOT to follow:

find a spanning tree of this graph

Most ‘efficient’ solution

Most ‘efficient’ plan for the NCDOT to follow:
find a **minimum weight** spanning tree of this graph

Most ‘efficient’ solution

Most ‘efficient’ plan for the NCDOT to follow:
find a **minimum weight** spanning tree of this graph

(in other words, try to minimise the total cost (*total weight*) of laying the necessary infrastructure).

Most 'efficient' solution

Most 'efficient' plan for the NCDOT to follow:
find a **minimum weight** spanning tree of this graph

(in other words, try to minimise the total cost (*total weight*) of laying the necessary infrastructure).

A problem of this type is usually referred to as

the connector problem.

Algorithms for finding minimum weight spanning trees?

One such algorithm is

Kruskal's algorithm

Suppose you are given a weighted connected graph G, and you are looking for a minimum weight spanning tree of it.

Algorithms for finding minimum weight spanning trees?

One such algorithm is

Kruskal's algorithm

Suppose you are given a weighted connected graph G, and you are looking for a minimum weight spanning tree of it. Then:

1. Begin by finding an edge of minimum weight, and mark it.

Algorithms for finding minimum weight spanning trees?

One such algorithm is

Kruskal's algorithm

Suppose you are given a weighted connected graph G, and you are looking for a minimum weight spanning tree of it. Then:

1. Begin by finding an edge of minimum weight, and mark it.
2. Out of all the edges that remain unmarked **and which do not form a cycle with any of the already marked edges**, pick an edge of minimum weight and mark it.

Algorithms for finding minimum weight spanning trees?

One such algorithm is

Kruskal's algorithm

Suppose you are given a weighted connected graph G, and you are looking for a minimum weight spanning tree of it. Then:

1. Begin by finding an edge of minimum weight, and mark it.
2. Out of all the edges that remain unmarked **and which do not form a cycle with any of the already marked edges**, pick an edge of minimum weight and mark it.
3. If the set of the already marked edges gives a spanning tree of G, then terminate the process. Otherwise, return to Step 2.

Algorithms for finding minimum weight spanning trees?

One such algorithm is

Kruskal's algorithm

Suppose you are given a weighted connected graph G, and you are looking for a minimum weight spanning tree of it. Then:

1. Begin by finding an edge of minimum weight, and mark it.
2. Out of all the edges that remain unmarked **and which do not form a cycle with any of the already marked edges**, pick an edge of minimum weight and mark it.
3. If the set of the already marked edges gives a spanning tree of G, then terminate the process. Otherwise, return to Step 2.

Theorem (analogous to above)

For every weighted connected graph G, Kruskal's algorithm gives a minimum weight spanning tree.

Applying Kruskal's algorithm to the above example