Trees!

Written by Rachel J. Morris, last updated Mar 29, 2018

About

Trees are a handy structure in Data Structures, and are also a part of Graph Theory.

Topics

1. Intro to Trees
2. Propositions & Theorems
3. Spanning Tree Algorithms

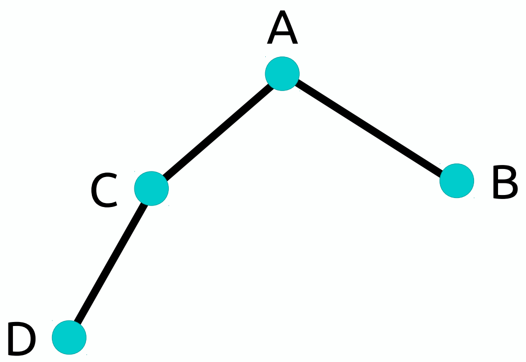
Intro to Trees

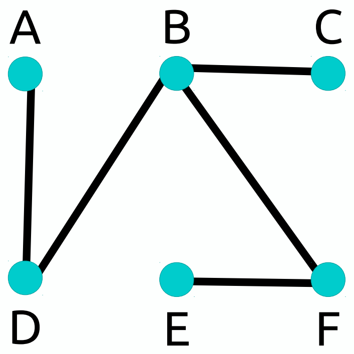
1. Intro to Trees

Connected Graph: A graph where there exists a walk between any two arbitrarily chosen nodes.

Simple Graph: A graph with no loops and no parallel edges.

Tree: A connected, simple graph with no cycles.





Notes

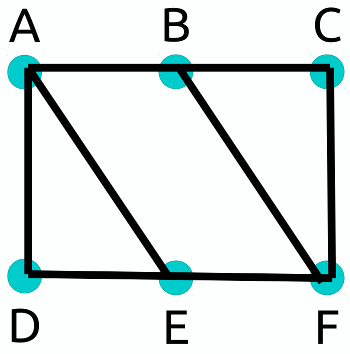
Tree: A connected, simple graph with no cycles.

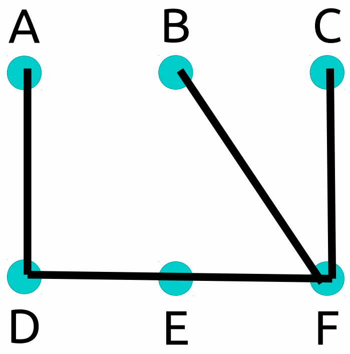
1. Intro to Trees

Connected Graph: A graph where there exists a walk between any two arbitrarily chosen nodes.

Simple Graph: A graph with no loops and no parallel edges.

Spanning Tree: Given some simple, connected graph G, a subgraph T is a spanning tree of G if T is a tree and every node in G is a node in T.





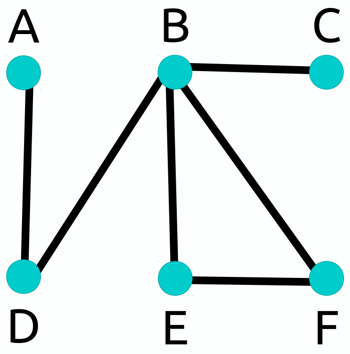
Notes

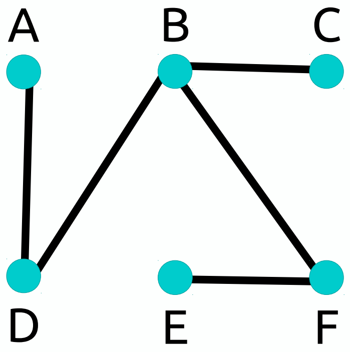
Tree: A connected, simple graph with no cycles.

Propositions & Theorems

2. Propositions & Theorems

Proposition 3: If you have a connected graph with a cycle in it, and you remove an edge from the cycle, then the graph is still connected.



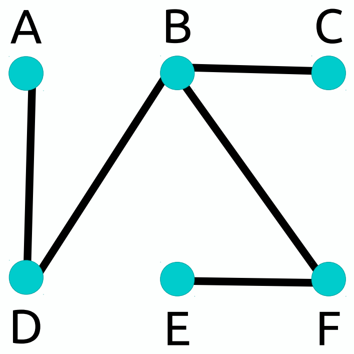


Notes

Tree: A connected, simple graph with no cycles.

2. Propositions & Theorems

Proposition 4: For a connected graph with at least one edge, if there are no cycles in the graph, then the graph has at least one vertex with degree 1.

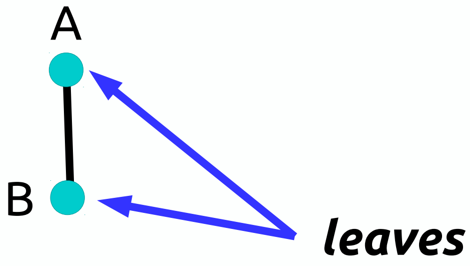


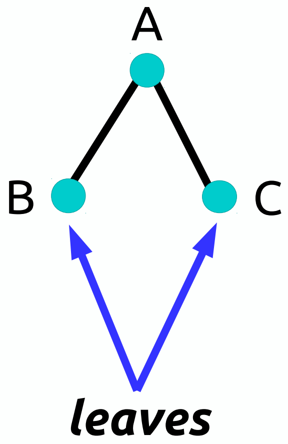
Notes

Tree: A connected, simple graph with no cycles.

2. Propositions & Theorems

Proposition 5: For any tree with at least one edge, the tree has at least two leaves.



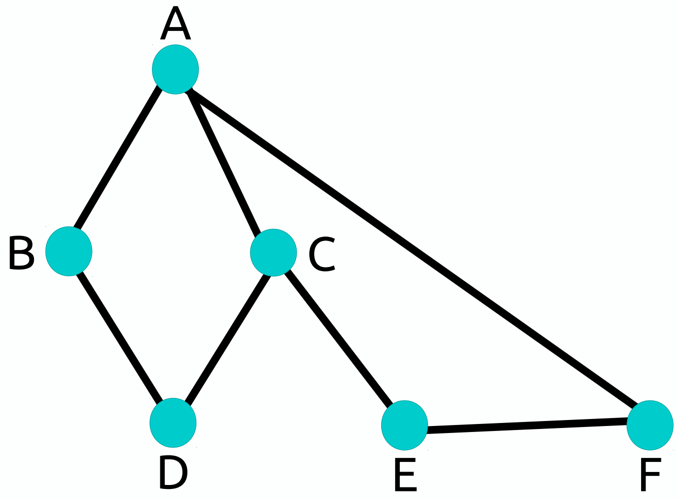


Notes

Tree: A connected, simple graph with no cycles.

2. Propositions & Theorems

Proposition 6: If a simple graph has a walk from vertex a to vertex b, then there is also a path from vertex a to vertex b.



Notes

Simple Graph: A graph with no loops and no parallel edges.

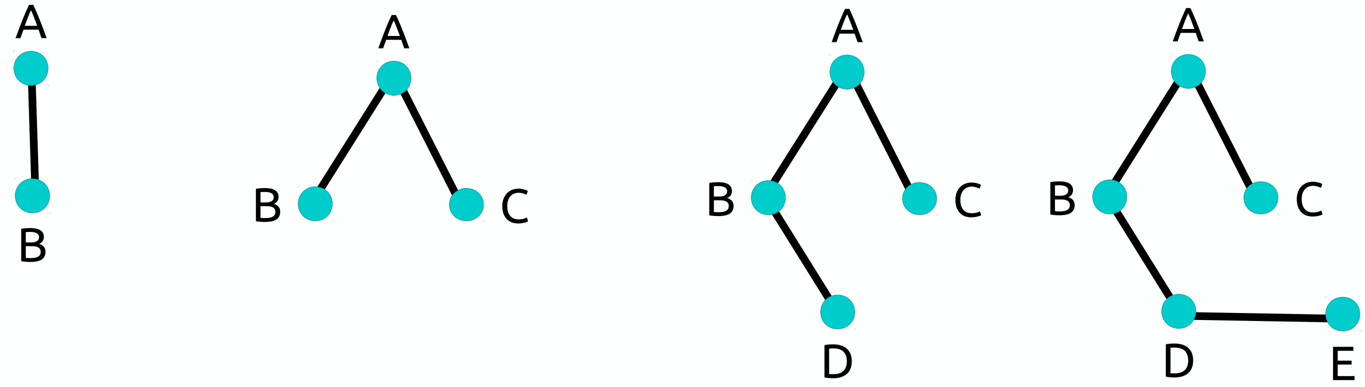
Walk: A series of alternating vertices/edges.

Trail: A walk with no repeated edges.

Path: A walk with no repeated vertices.

2. Propositions & Theorems

Theorem 7:
A tree with n edges has $n + 1$ vertices.



Notes

Simple Graph: A graph with no loops and no parallel edges.

Walk: A series of alternating vertices/edges.

Trail: A walk with no repeated edges.

Path: A walk with no repeated vertices.

2. Propositions & Theorems

Theorem 2:
A connected graph is Eulerian if (and only if) every node has an even degree.

Notes

Simple Graph: A graph with no loops and no parallel edges.

Walk: A series of alternating vertices/edges.

Trail: A walk with no repeated edges.

Path: A walk with no repeated vertices.

Spanning Tree Algorithms

3. Spanning Tree Algorithms

(Basic) Spanning Tree Algorithm:

Input: A simple, connected graph G_0.

Algorithm:

For each i >= 1, as long as there is a cycle in G_{i-1}…

Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.

Output: The final result G_k will be a spanning tree of G_0.

From Discrete Mathematics, Ensley & Crawley, page 526

Notes

Simple Graph: A graph with no loops and no parallel edges.

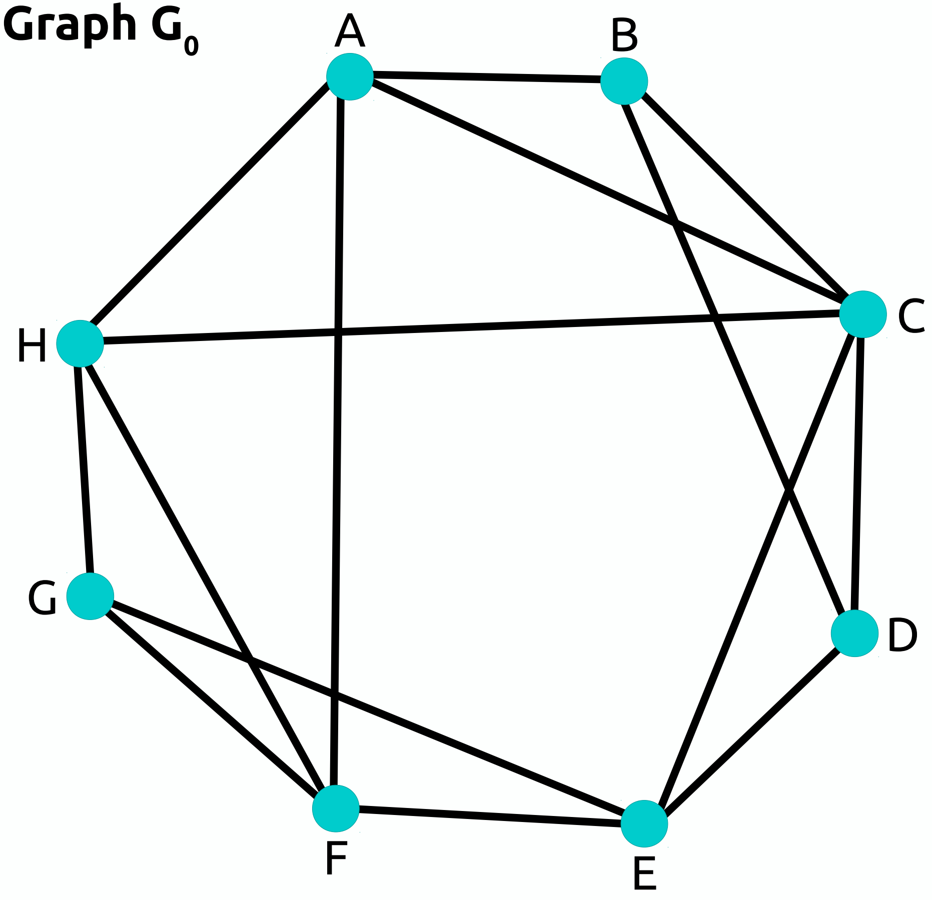
Walk: A series of alternating vertices/edges.

Trail: A walk with no repeated edges.

Path: A walk with no repeated vertices.

3. Spanning Tree Algorithms

Graph G_0



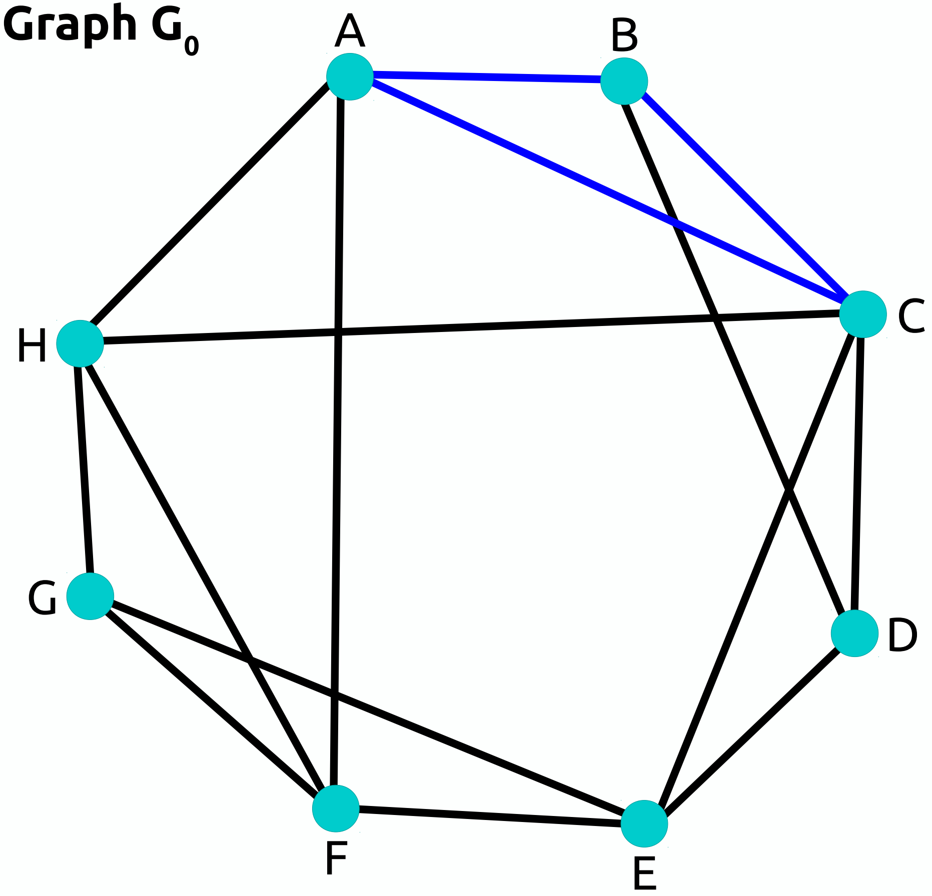
First, we begin with some simple, connected graph.

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a cycle in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_0



First, we begin with some simple, connected graph.

Cycles, exist, so we will start the for loop.

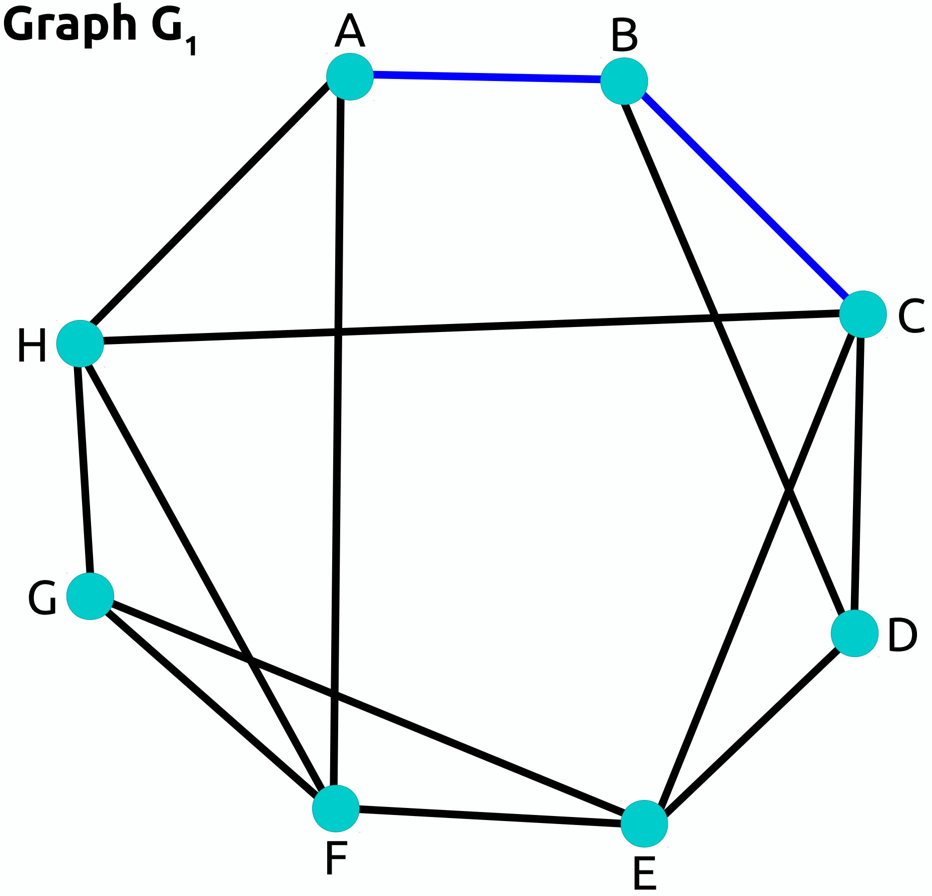
Choose any arbitrary cycle in the graph.

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a cycle in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_1



First, we begin with some simple, connected graph.

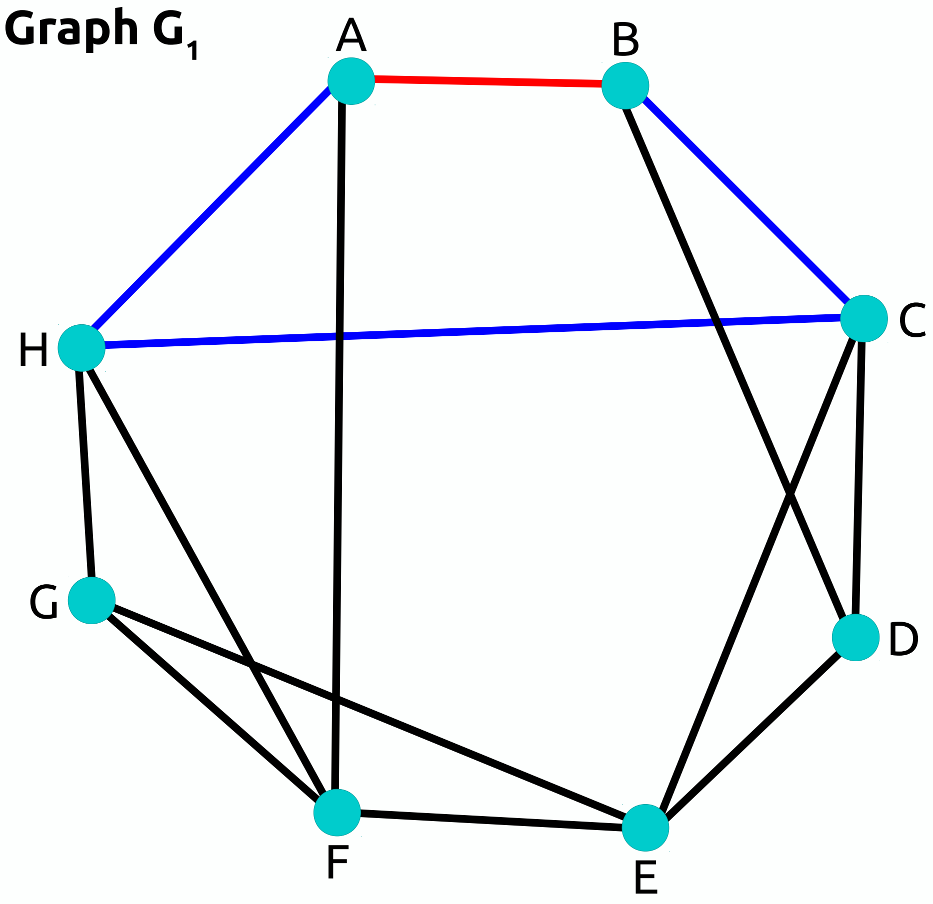
Choose any edge from this cycle and remove it.

This takes us from the initial state, G_{0}, to the next state, G_1.

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a <u>cycle</u> in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms



We still have cycles, so we continue.

Identify a cycle…

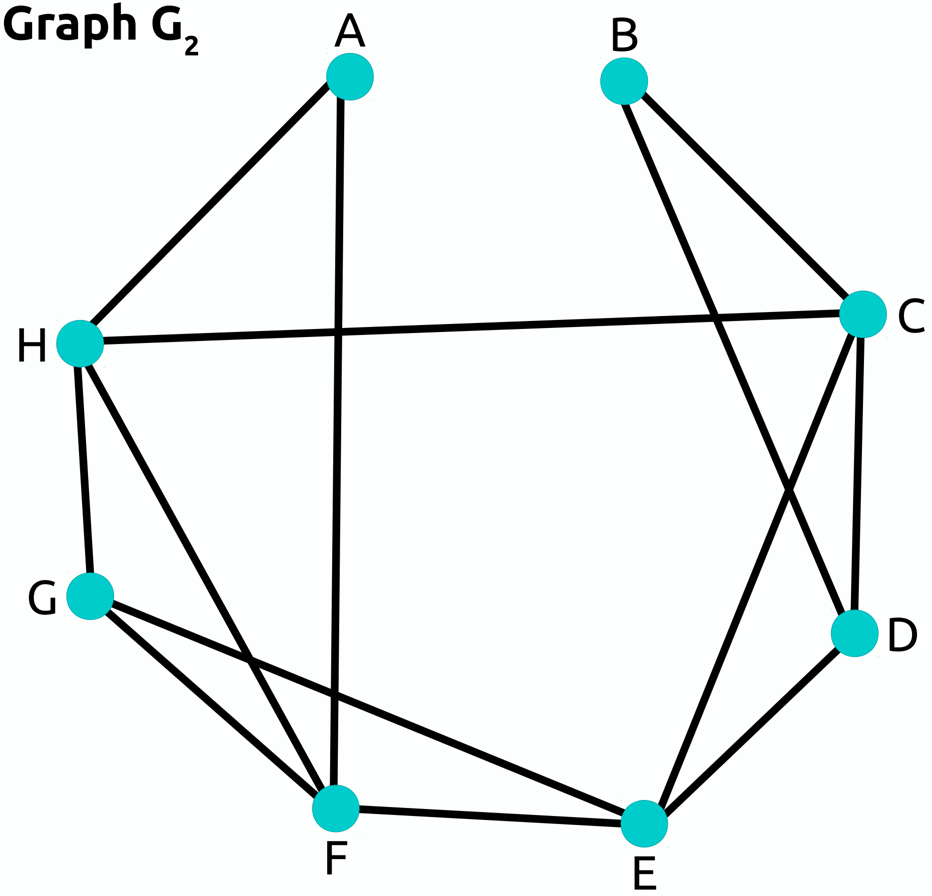
Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a <u>cycle</u> in G_{i-1}…
  - Choose an edge e in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting e from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_2



We still have cycles, so we continue.

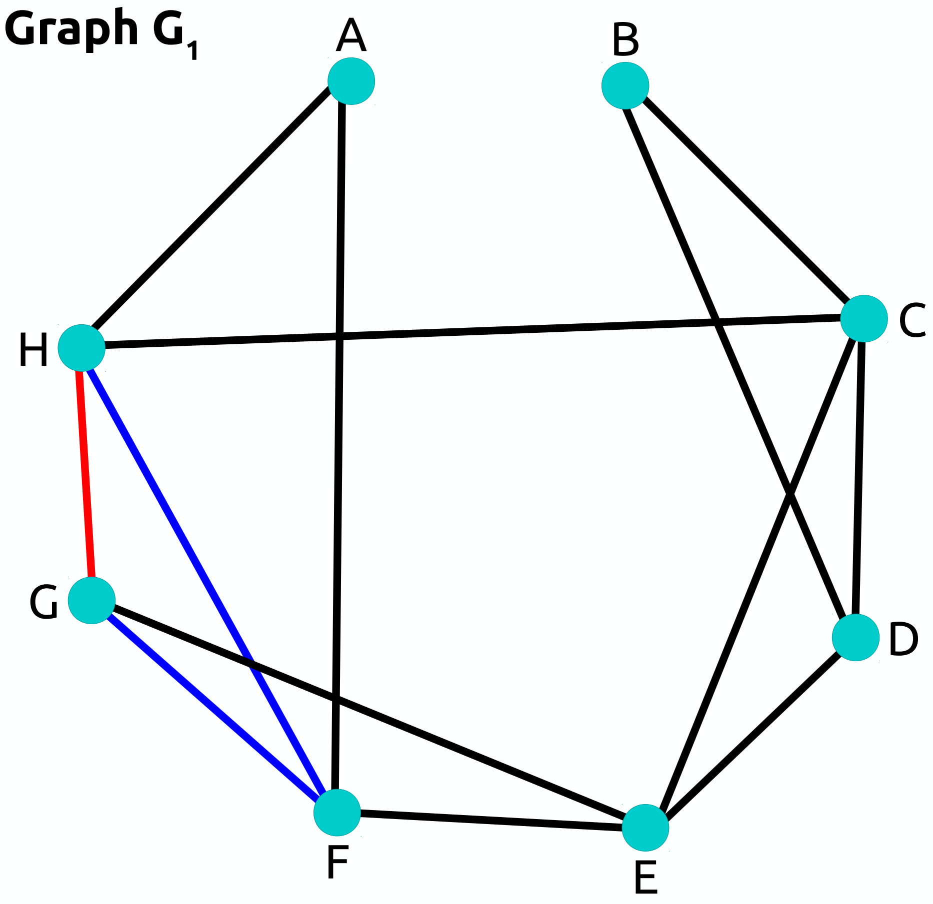
Identify a cycle…

Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a cycle in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms



We still have cycles, so we continue.

Identify a cycle…

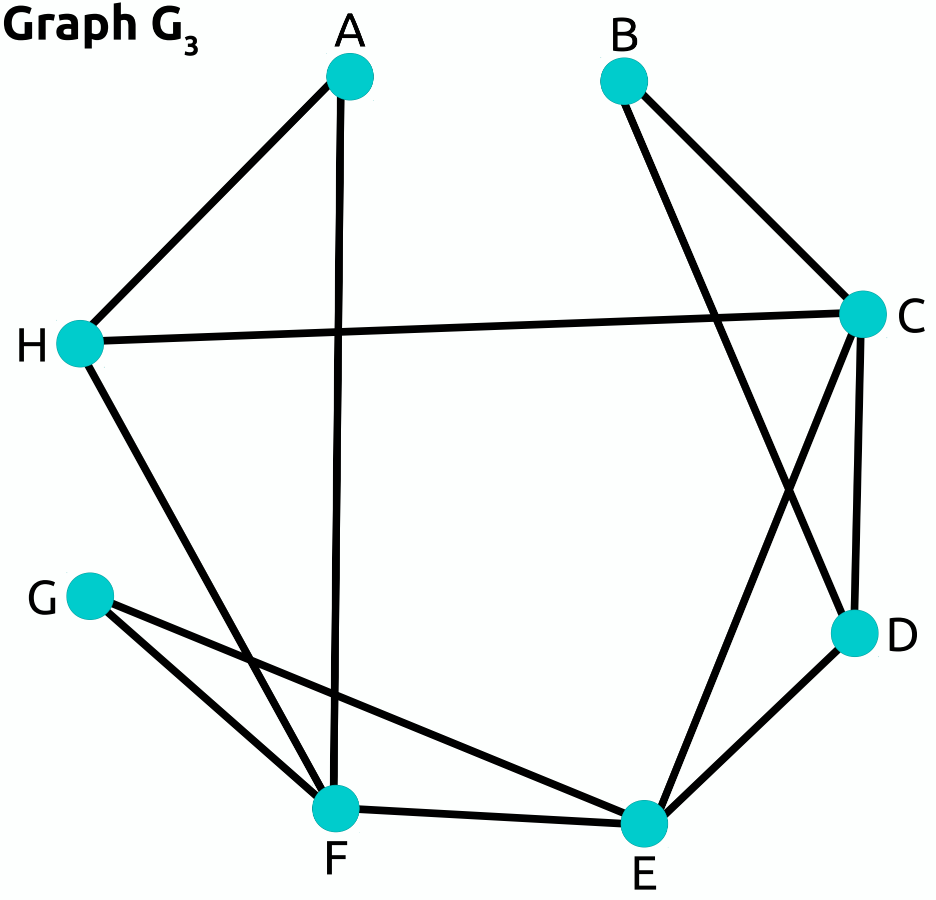
Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a cycle in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_3



We still have cycles, so we continue.

Identify a cycle…

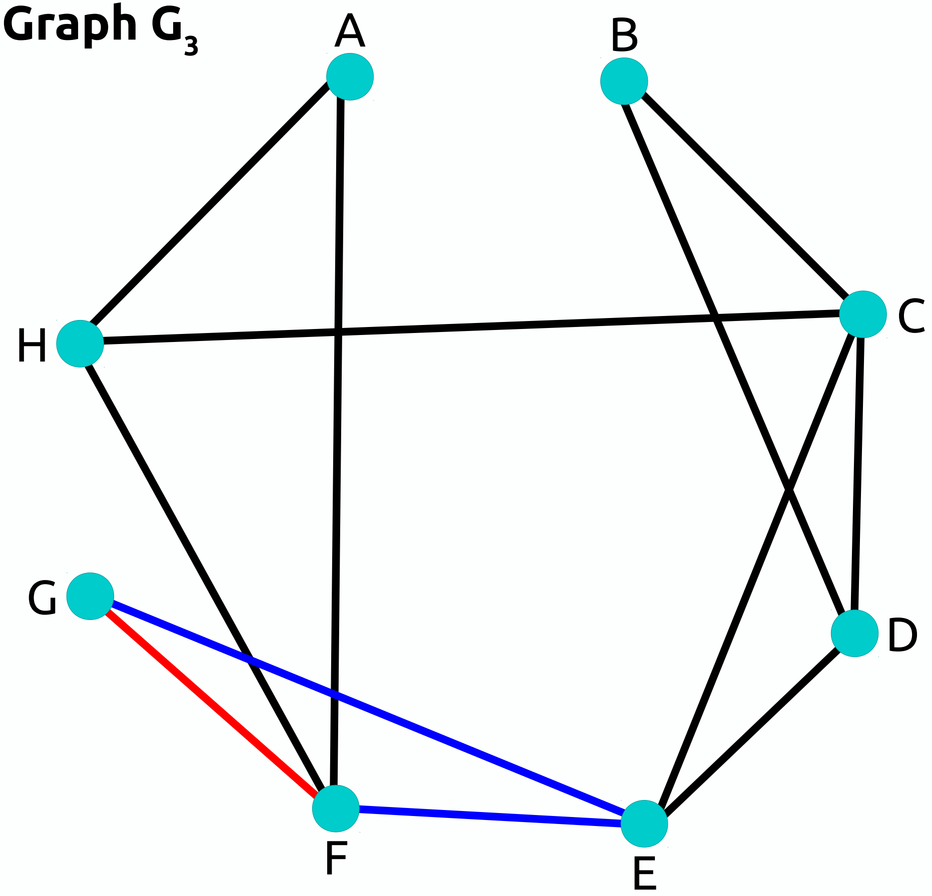
Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a <u>cycle</u> in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_3



We still have cycles, so we continue.

Identify a cycle…

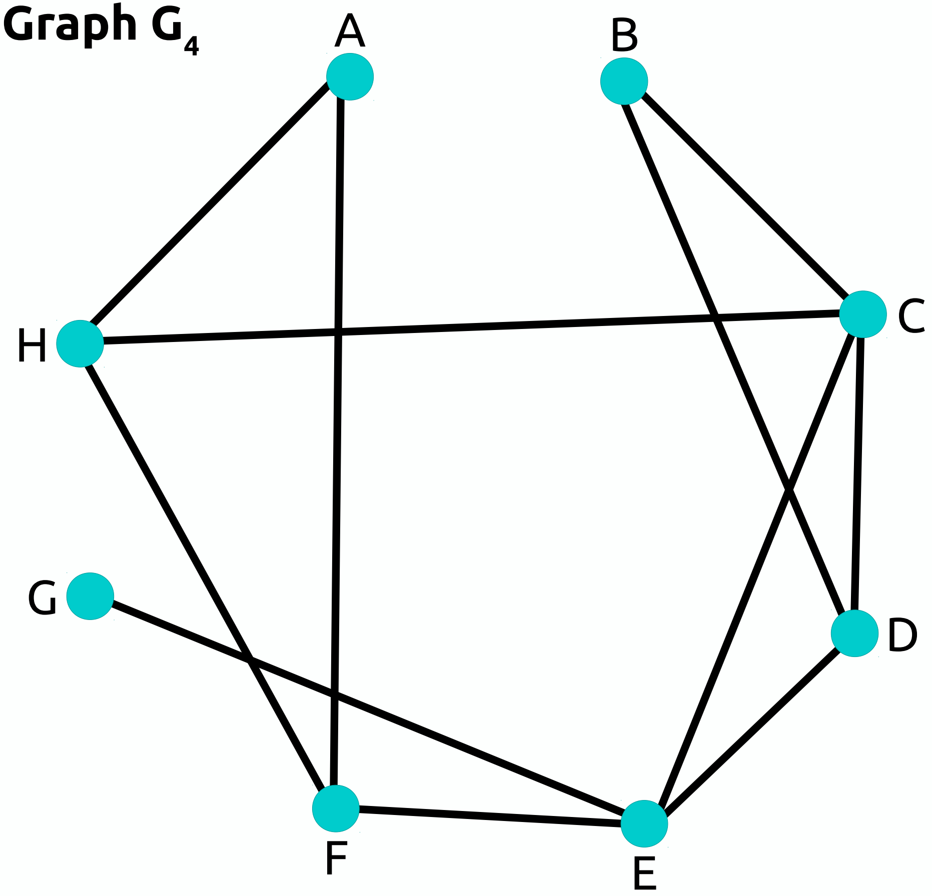
Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a <u>cycle</u> in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_4



We still have cycles, so we continue.

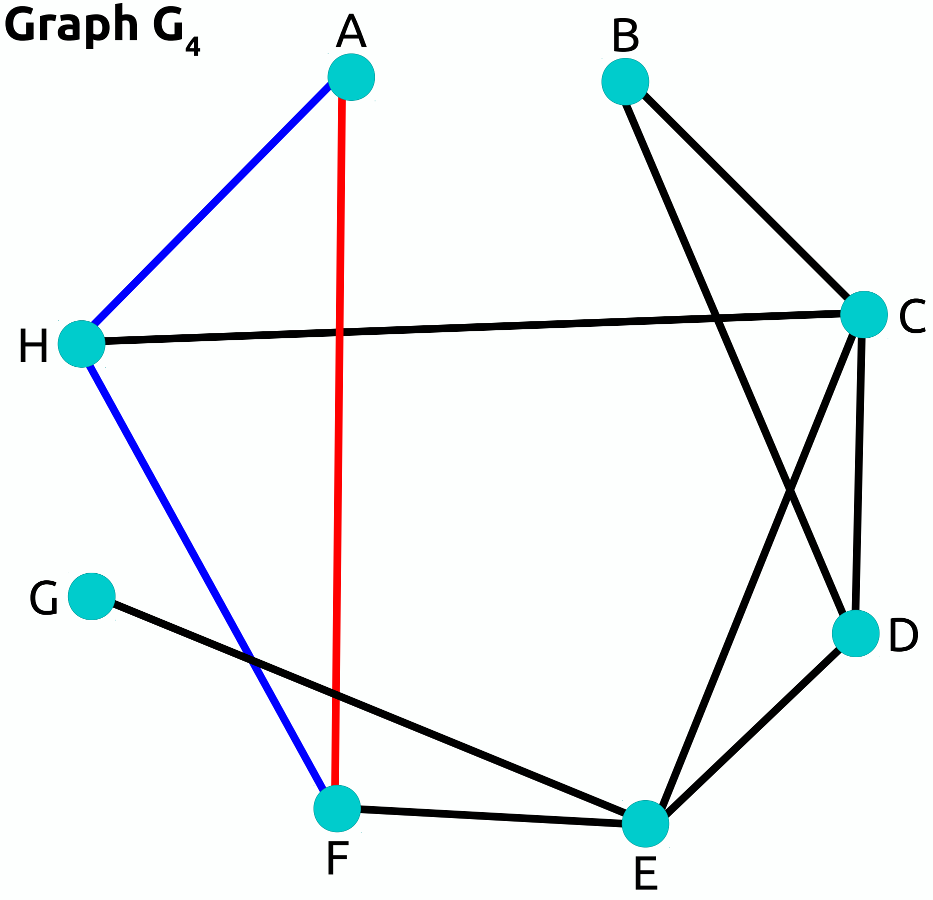
Identify a cycle…

Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a <u>cycle</u> in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms



We still have cycles, so we continue.

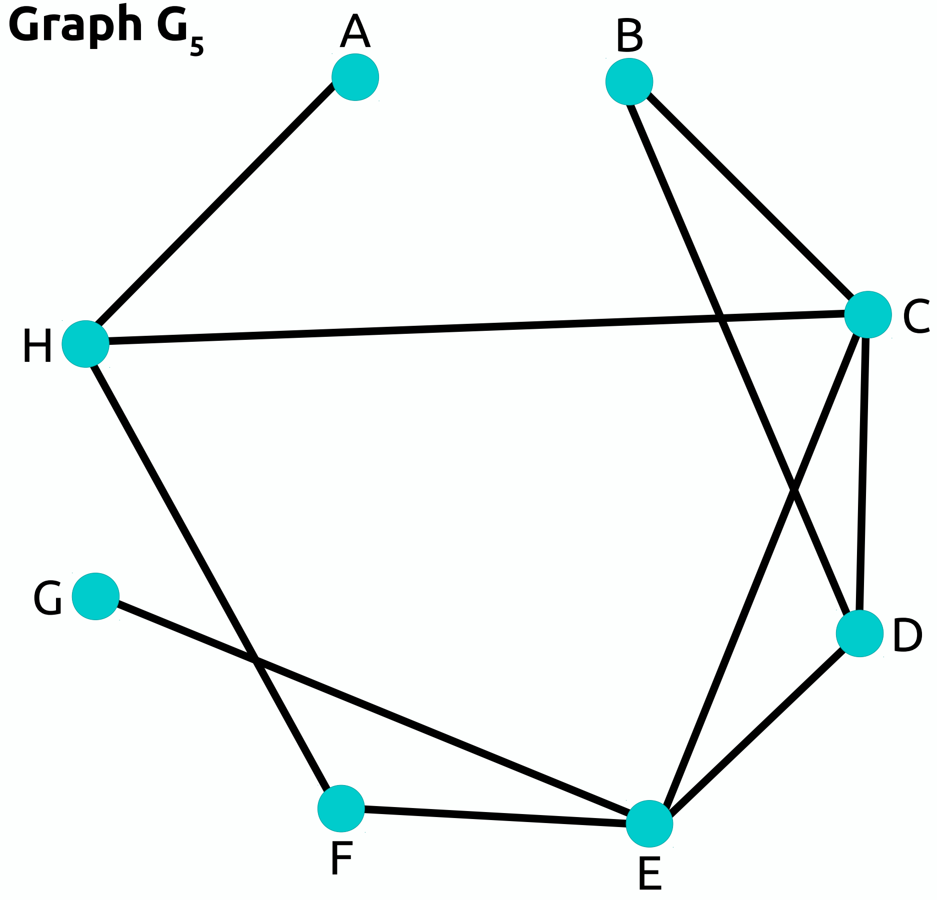
Identify a cycle…

Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a cycle in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms



We still have cycles, so we continue.

Identify a cycle…

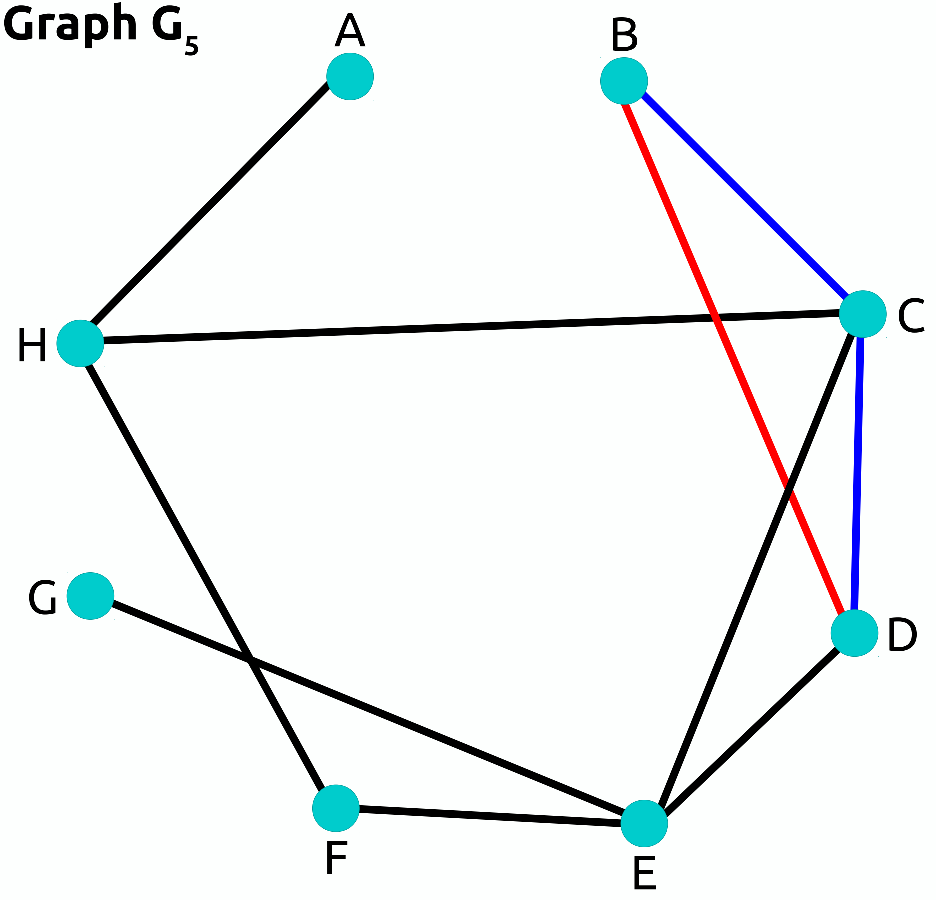
Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a cycle in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_5



We still have cycles, so we continue.

Identify a cycle…

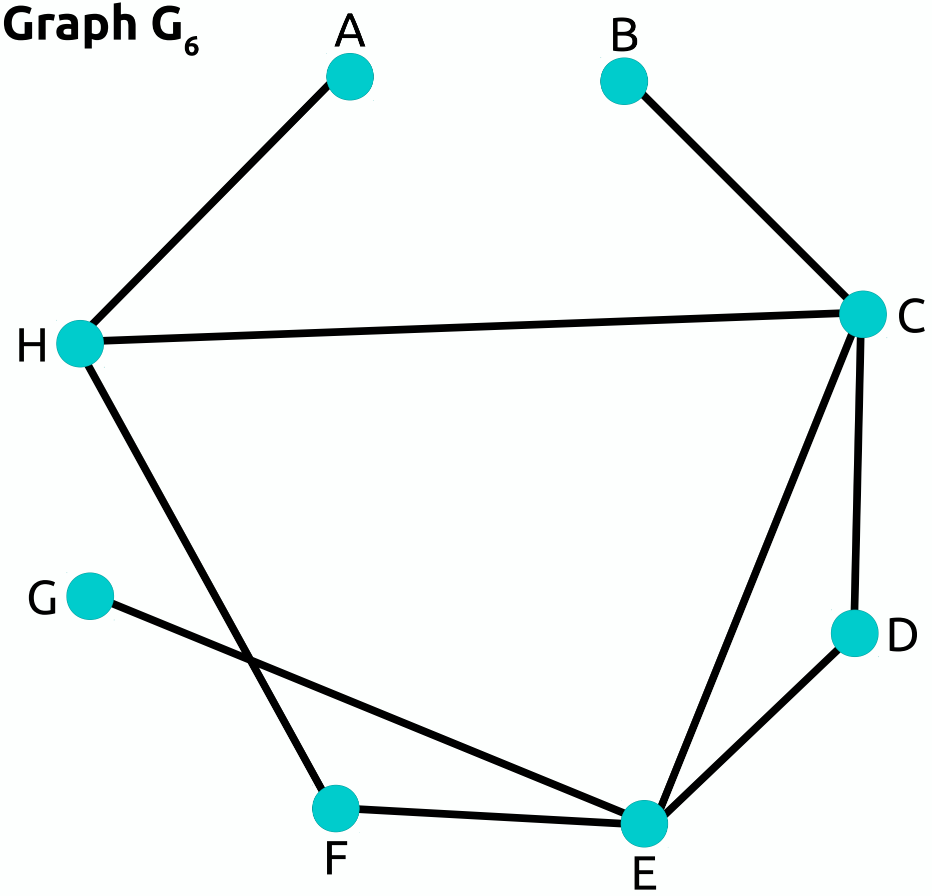
Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a <u>cycle</u> in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_6



We still have cycles, so we continue.

Identify a cycle…

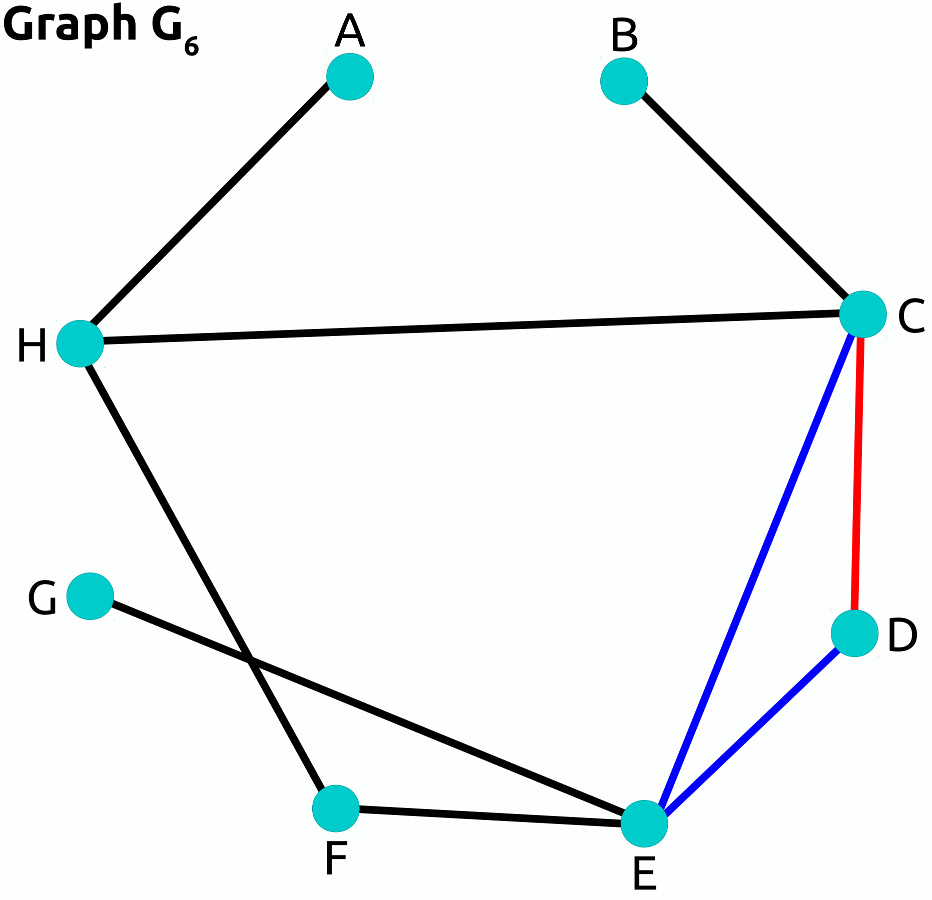
Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a <u>cycle</u> in G_{i-1}…
  - Choose an edge e in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting e from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_6



We still have cycles, so we continue.

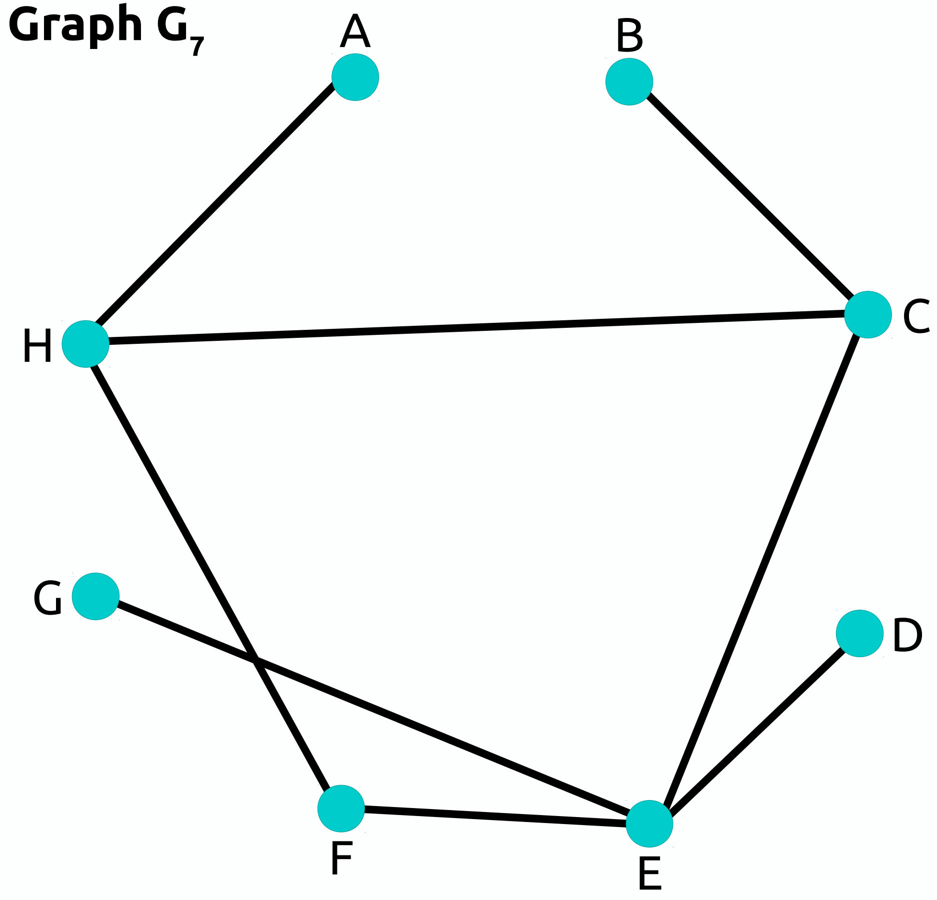
Identify a cycle…

Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each $i >= 1$, as long as there is a <u>cycle</u> in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms



We still have cycles, so we continue.

Identify a cycle…

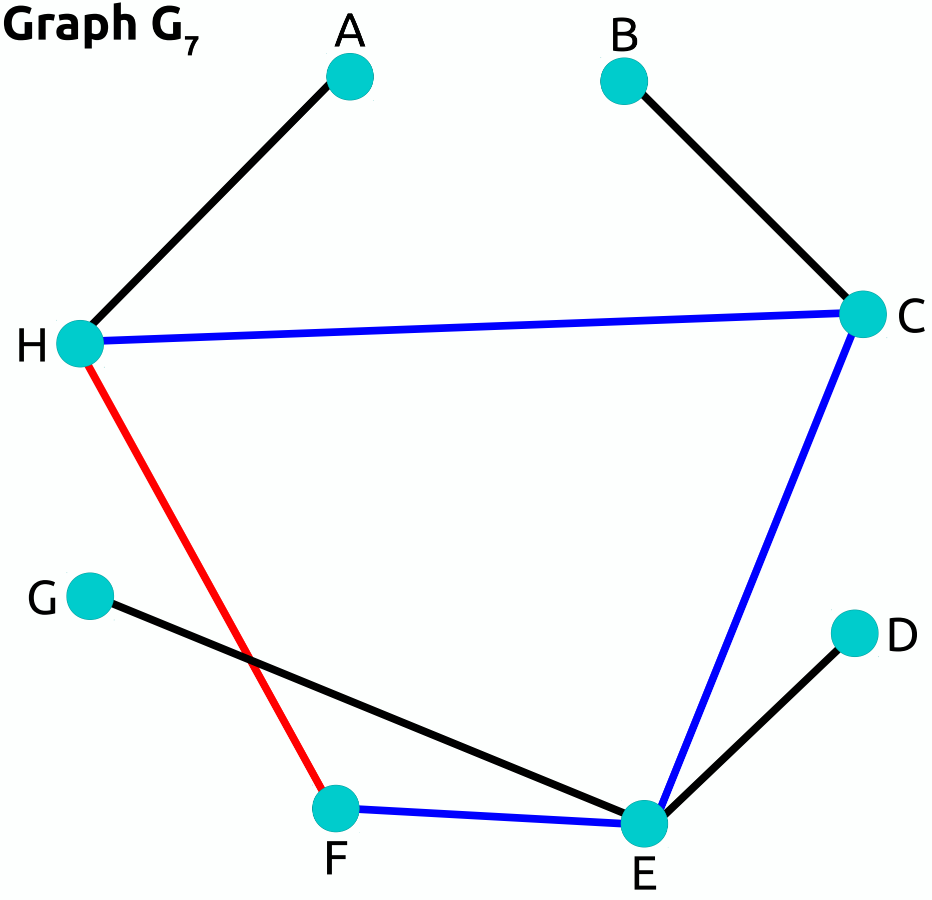
Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a cycle in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_7



We still have cycles, so we continue.

Identify a cycle…

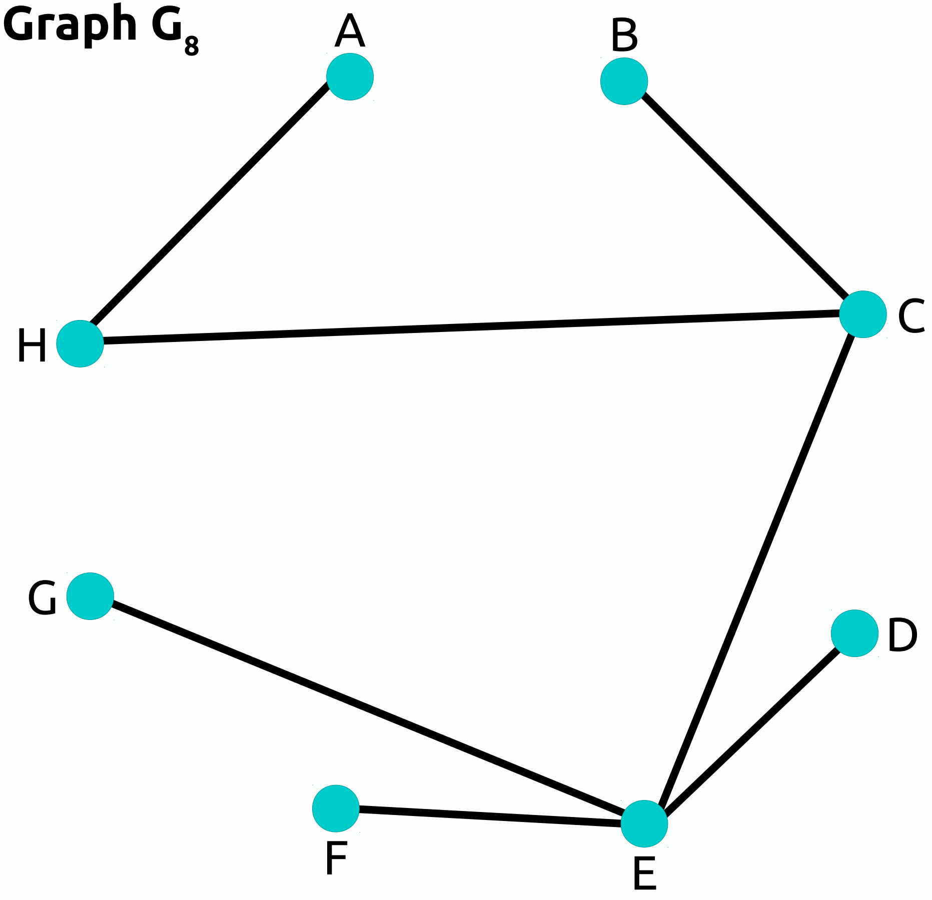
Remove an edge…

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a cycle in G_{i-1}…
  - Choose an edge *e* in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting *e* from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Graph G_8



Now we have no more cycles, and we are done.

This is one of many possible spanning trees of the original graph, G_0.

(Basic) Spanning Tree Algorithm:

- Input: A simple, connected graph G_0.
- For each i >= 1, as long as there is a cycle in G_{i-1}…
  - Choose an edge e in any cycle of G_{i-1}, and form the subgraph G_i of G_{i-1} by deleting e from G_{i-1}.
- Output: The final result G_k will be a spanning tree of G_0.

3. Spanning Tree Algorithms

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

For each k from {1, 2, …, n}:

- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

From Discrete Mathematics, Ensley & Crawley, page 527

Notes

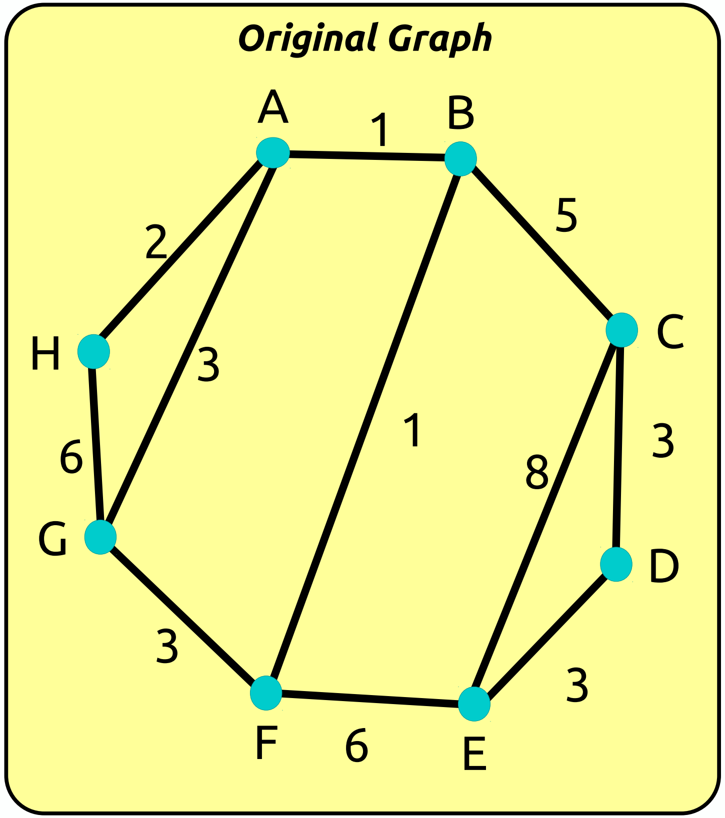
Simple Graph: A graph with <u>no loops</u> and <u>no parallel edges</u>.

Walk: A series of alternating vertices/edges.

Trail: A walk with no repeated edges.

Path: A walk with no repeated vertices.

3. Spanning Tree Algorithms



First, we begin with some simple, connected weighted graph.

We are going to build out a tree using the algorithm, and taking one edge at a time.

Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

For each k from {1, 2, …, n}:

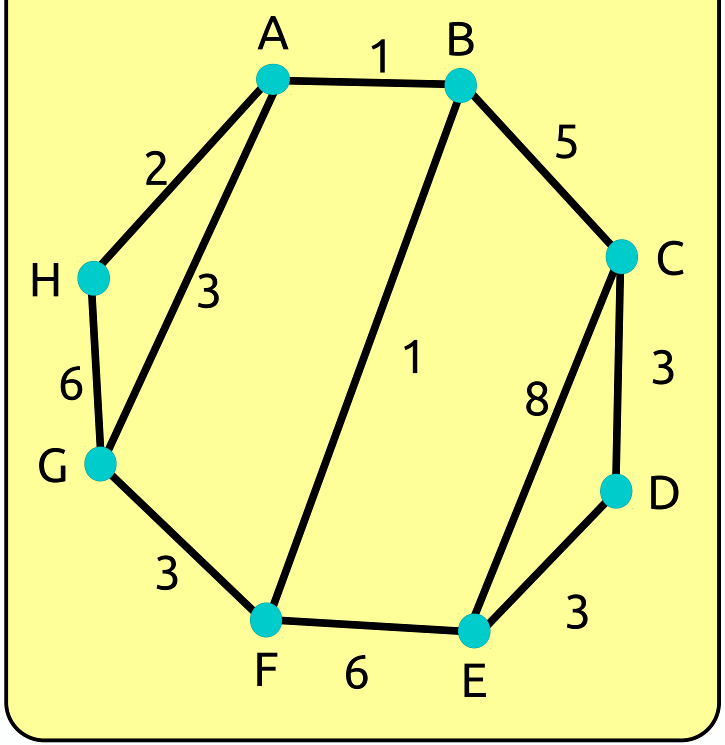
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

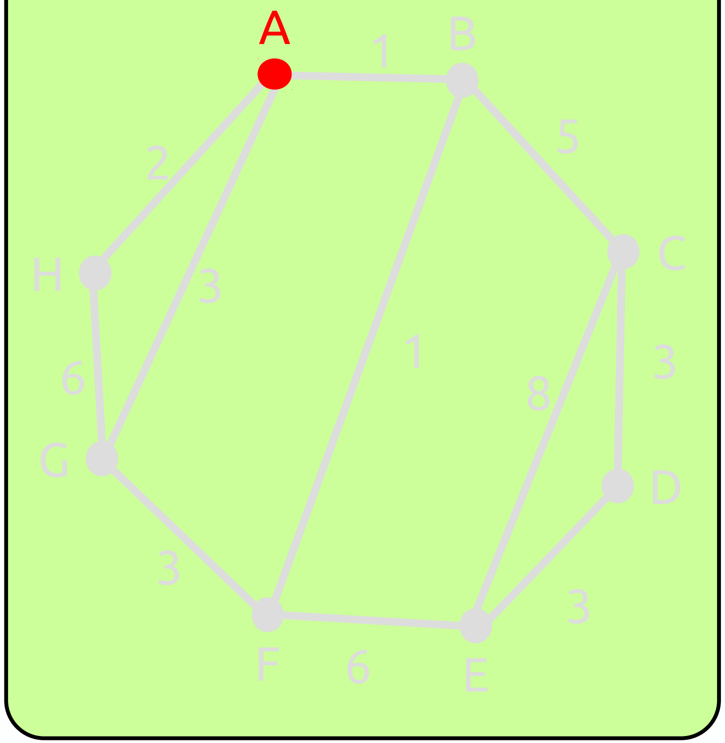
3. Spanning Tree Algorithms

We can start with any node, so I will start with node A. So $T_0 = \{v_A\}$ to start with.

Original Graph



Tree T_0



Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

For each k from {1, 2, …, n}:

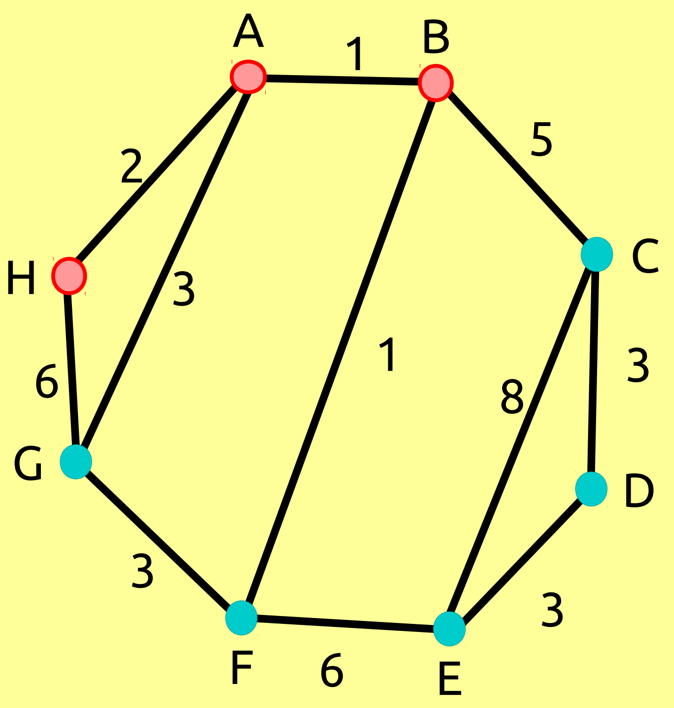
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

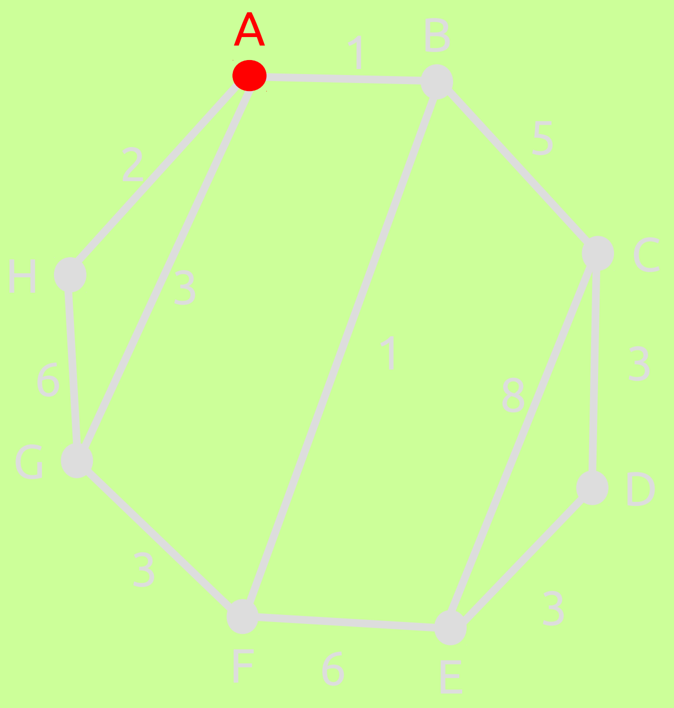
3. Spanning Tree Algorithms

Next, we build E_1, the list of edges in the graph that are connected to any nodes we currently have in the tree. For now, this means just the neighbors of A.

Original Graph



Tree T_0



Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

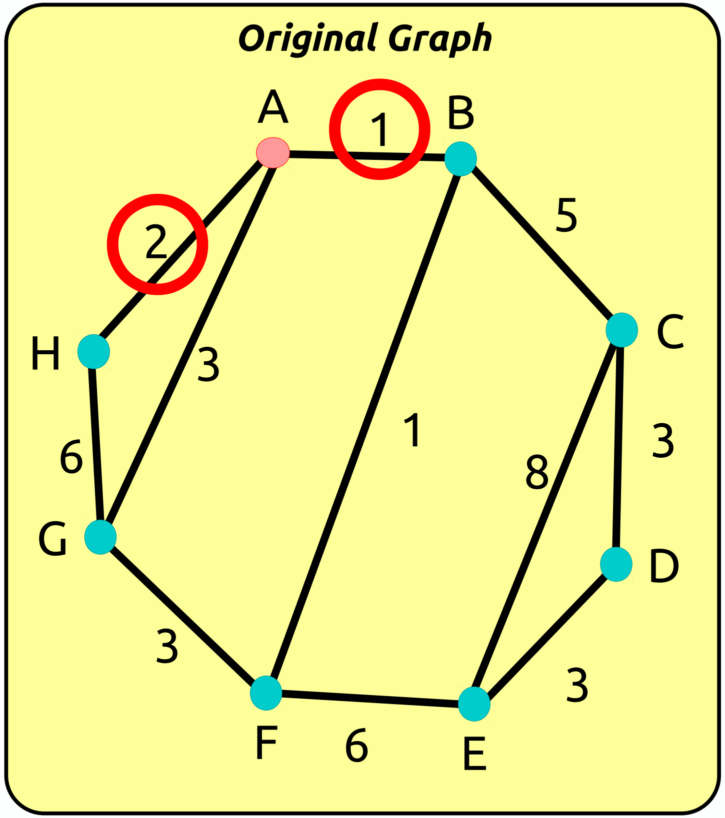
For each k from {1, 2, …, n}:

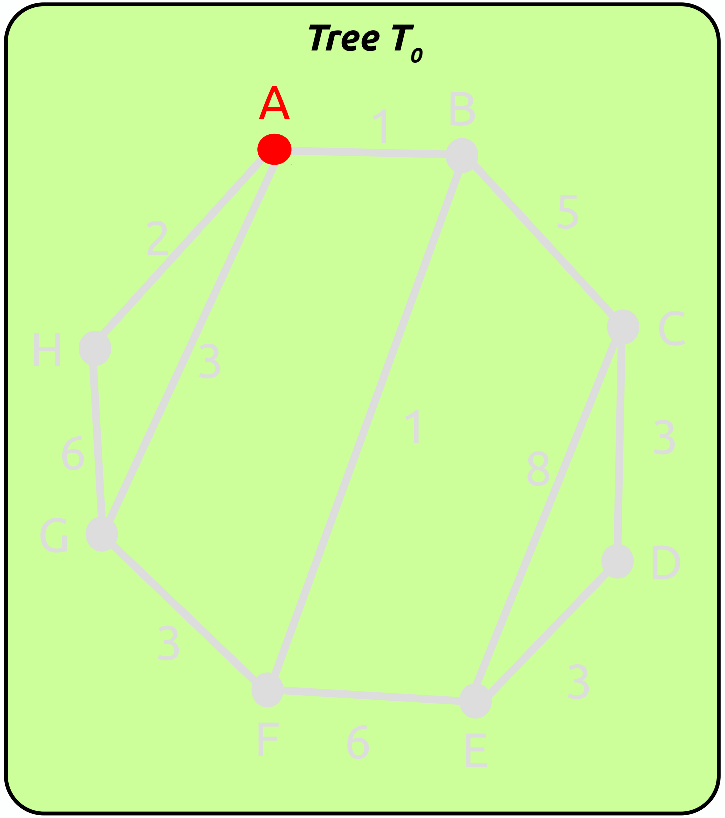
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

The connected edges have weights 2 and 1, so we choose the smallest one and put that edge – and its endpoint B – into our tree.





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

For each k from {1, 2, …, n}:

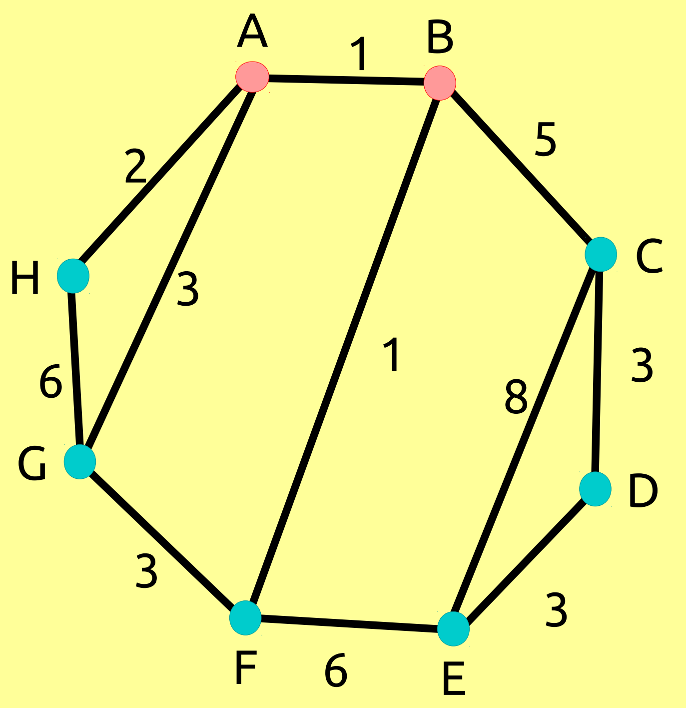
- Let $E_k = \{e$ an edge in $G : e$ has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

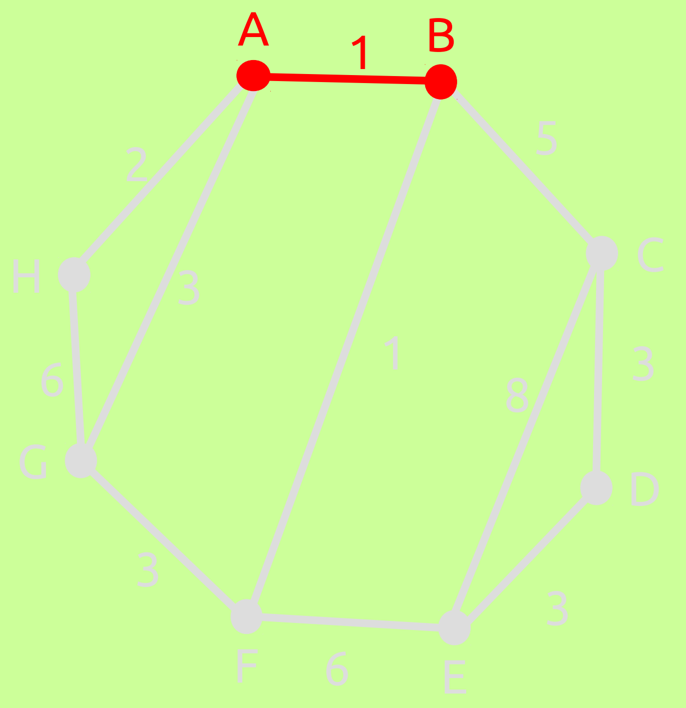
3. Spanning Tree Algorithms

The connected edges have weights 2 and 1, so we choose the smallest one and put that edge – and its endpoint B – into our tree.

Original Graph



Tree T_1



Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

For each k from {1, 2, …, n}:

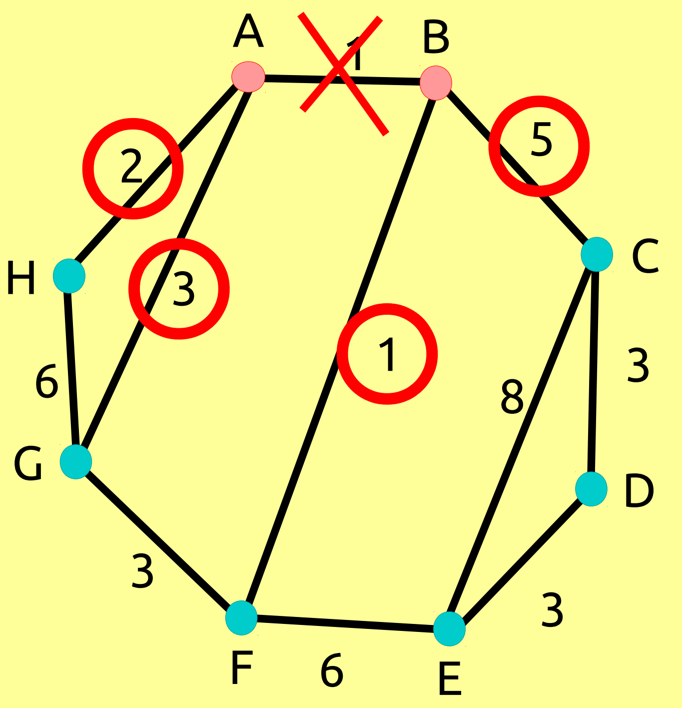
- Let E_k = {e an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in T_{k-1}}.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

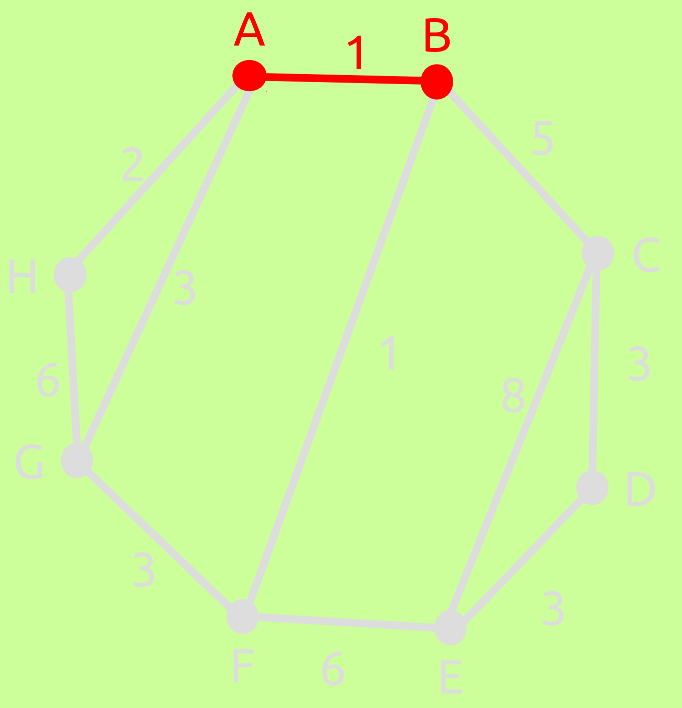
3. Spanning Tree Algorithms

Next, we look at all edges connected to A and B in the original graph, and identify which has the smallest weight.

Original Graph



Tree T_1



Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

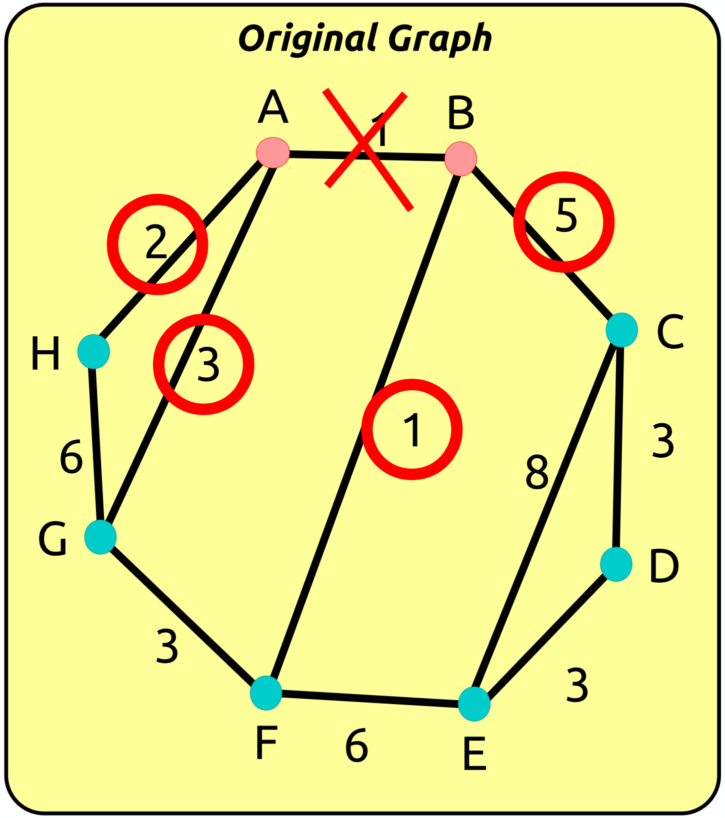
For each k from {1, 2, …, n}:

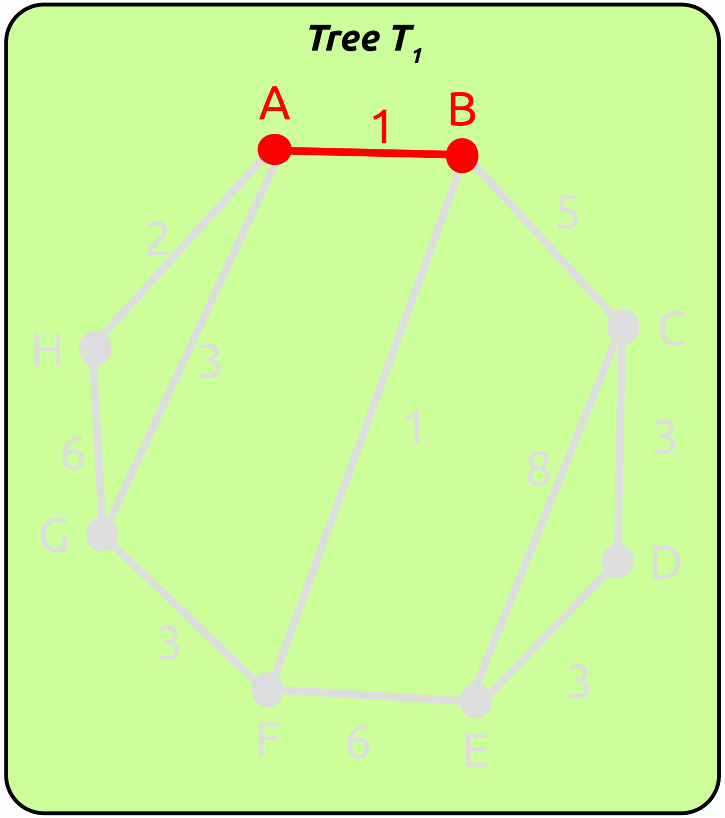
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

The next smallest edge is the connection from B to F, so we add that edge and F to the tree.





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

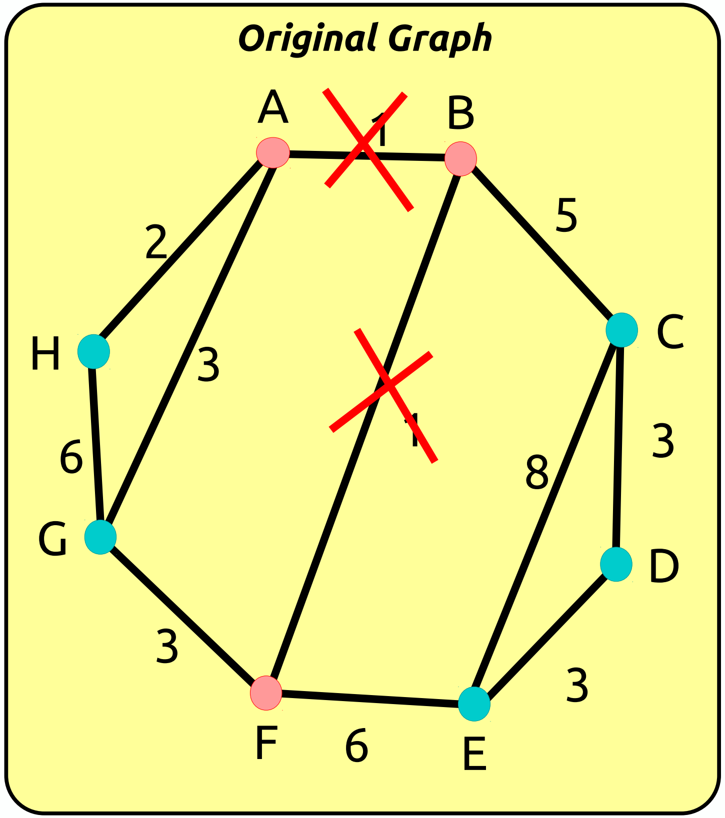
For each k from {1, 2, …, n}:

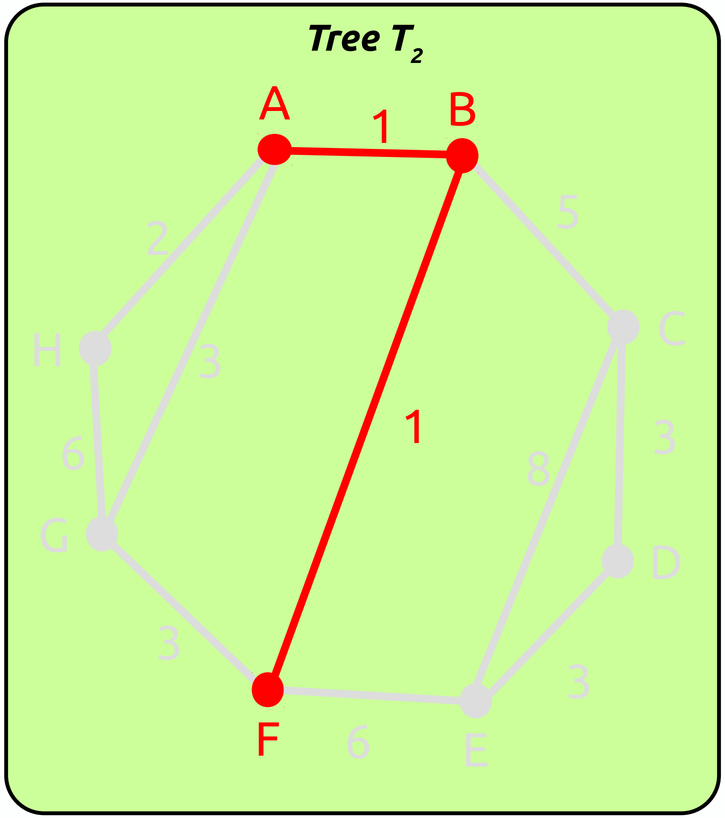
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

The next smallest edge is the connection from B to F, so we add that edge and F to the tree.





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

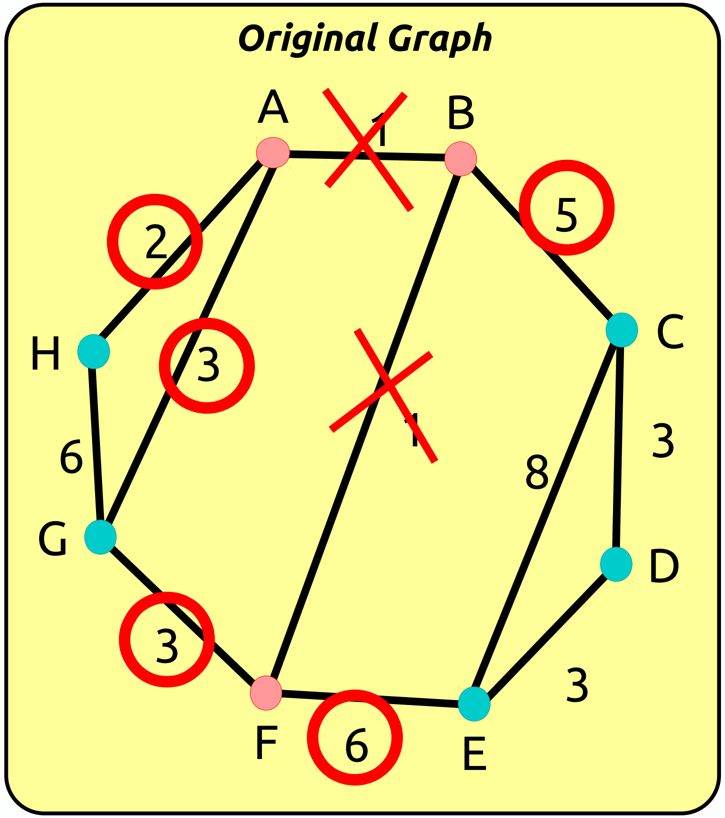
For each k from {1, 2, …, n}:

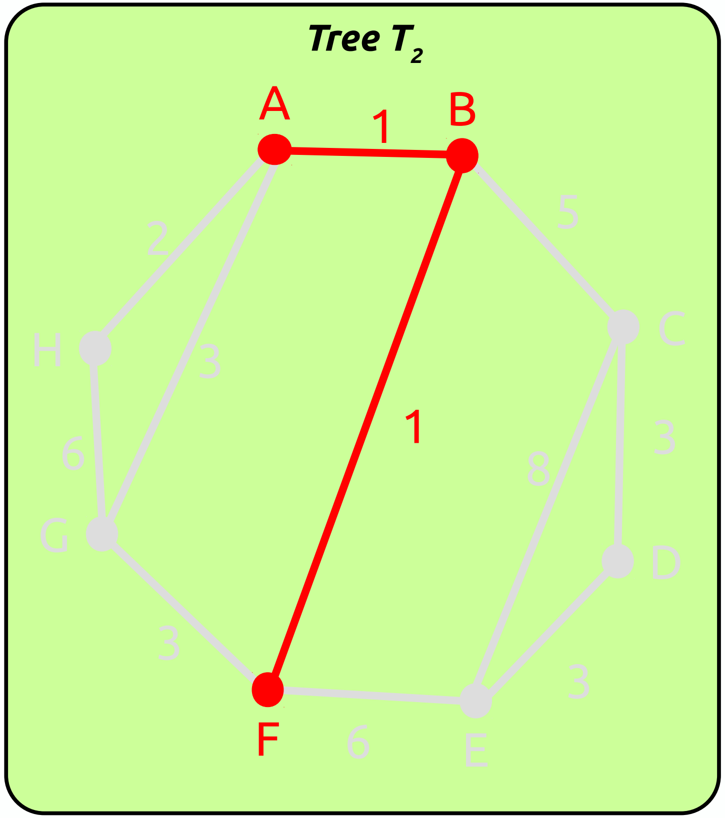
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

And we continue…





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

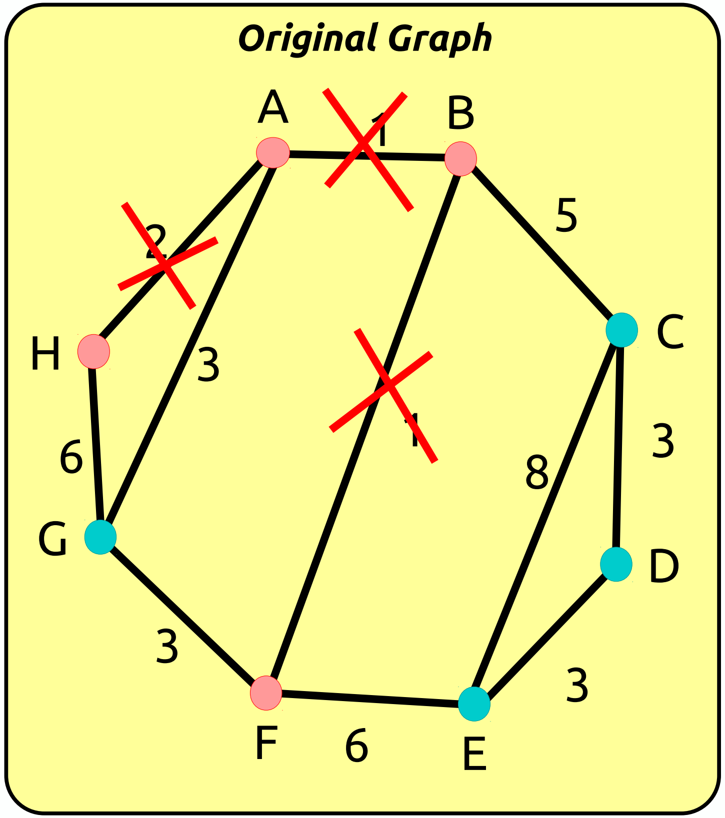
For each k from {1, 2, …, n}:

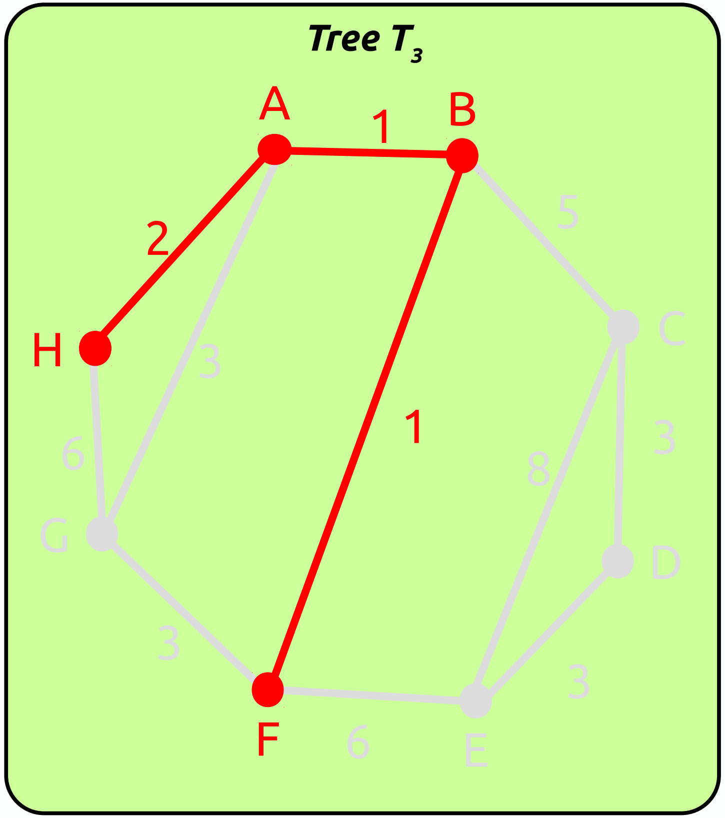
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

And we continue…





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

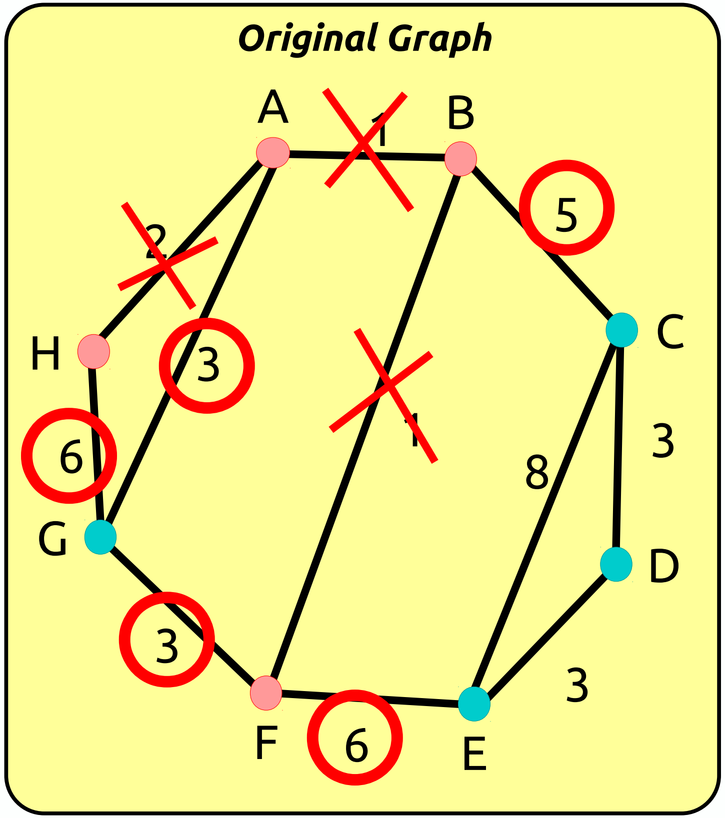
For each k from {1, 2, …, n}:

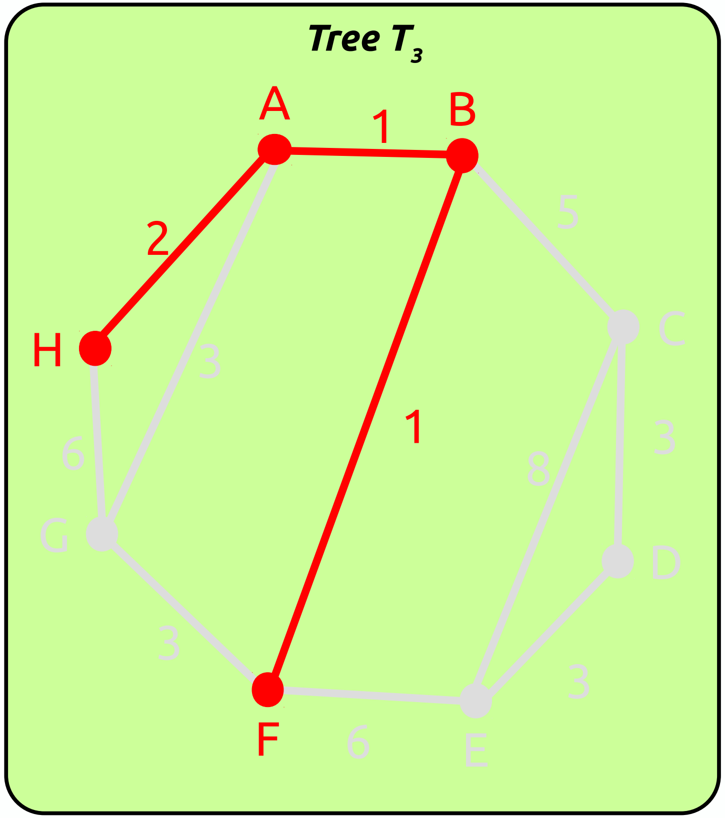
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

We can choose either of the 3 edges, since they have the same weight.





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

For each k from {1, 2, …, n}:

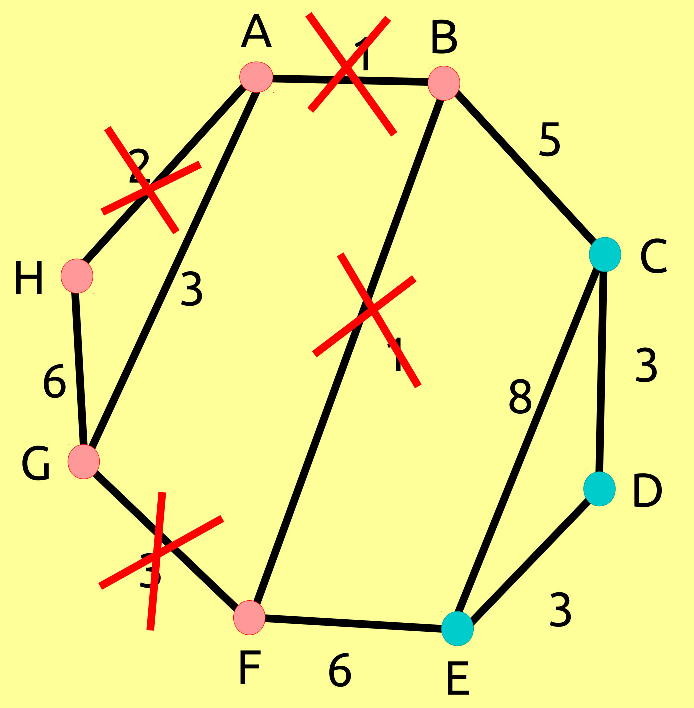
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

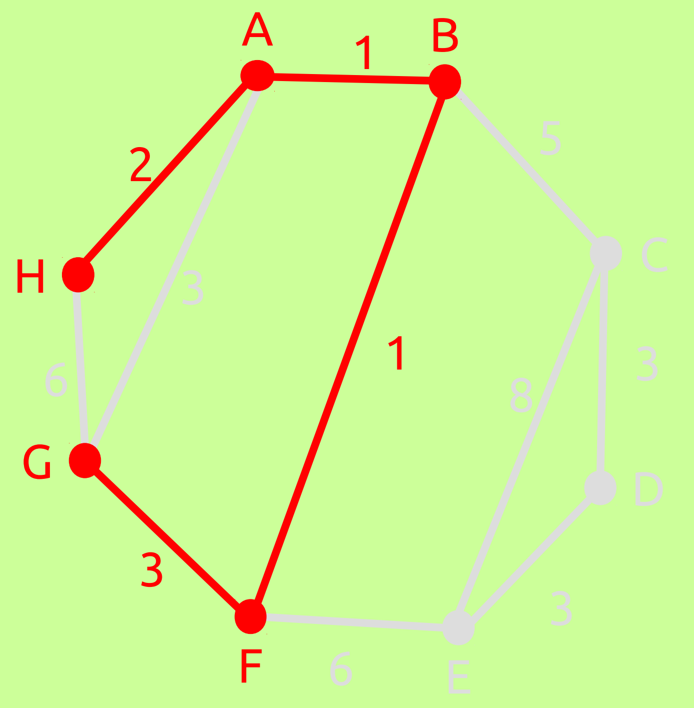
3. Spanning Tree Algorithms

And we continue…

Original Graph



Tree T_4



Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

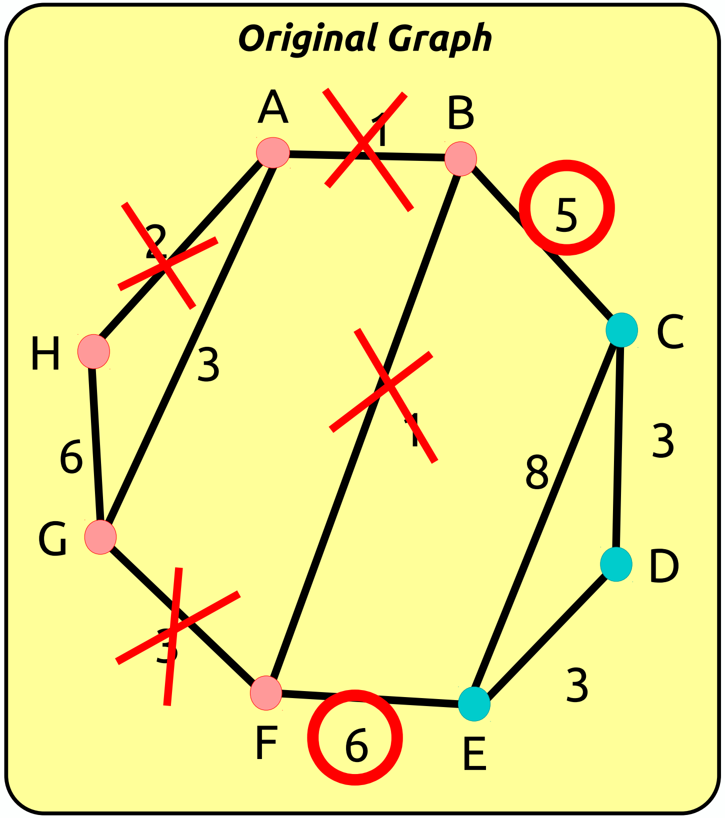
For each k from {1, 2, …, n}:

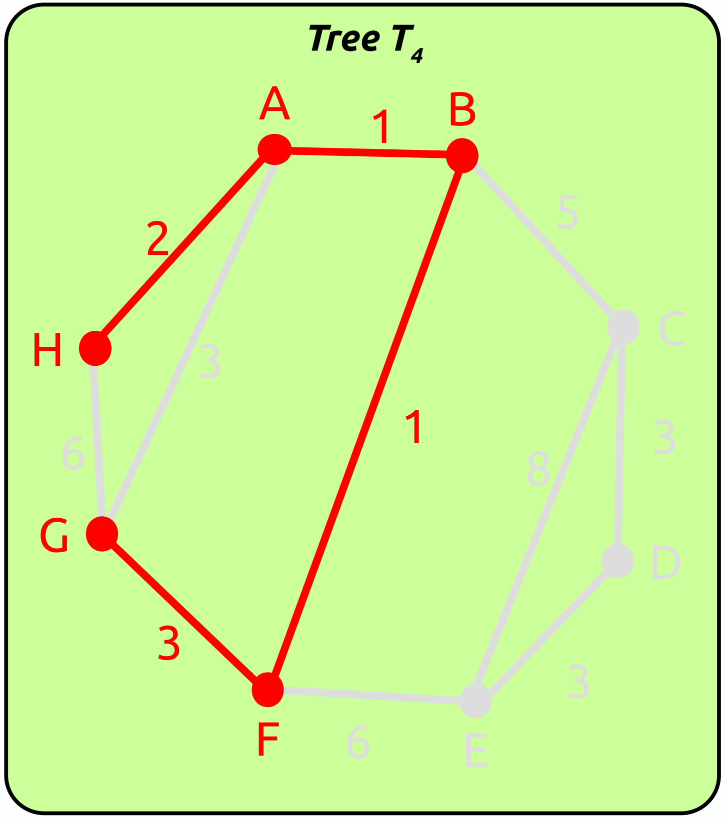
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

We don't check any edges that are connecting nodes already in the tree.





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

For each k from {1, 2, …, n}:

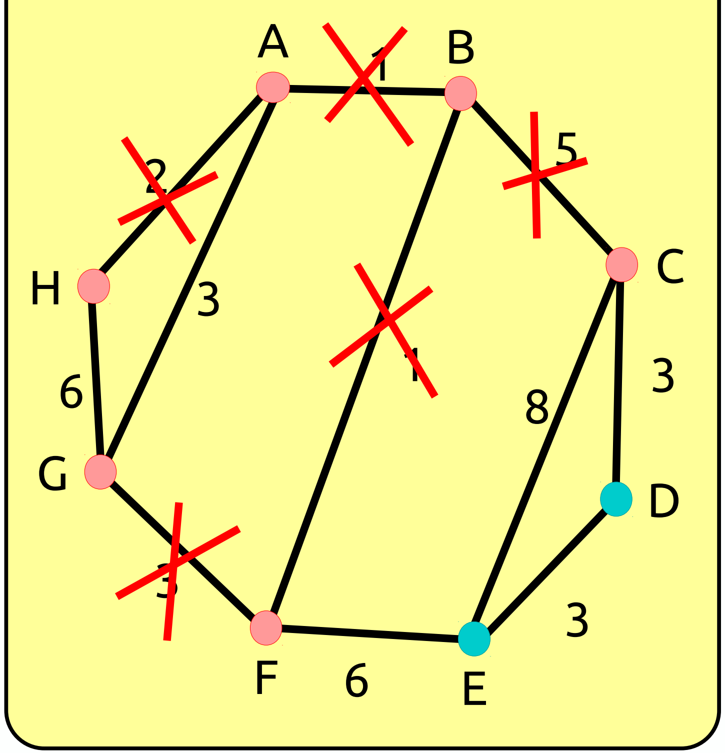
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

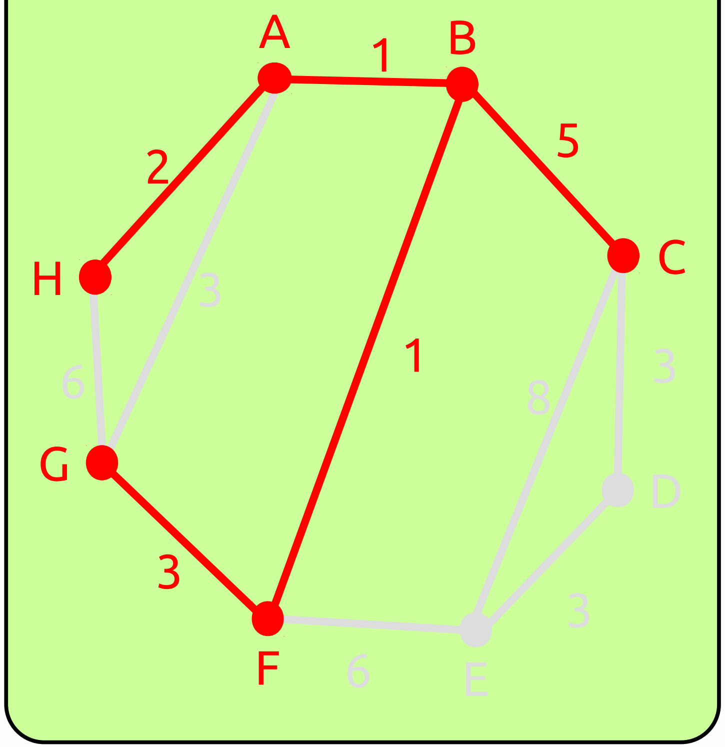
3. Spanning Tree Algorithms

And we continue…

Original Graph



Tree T_5



Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

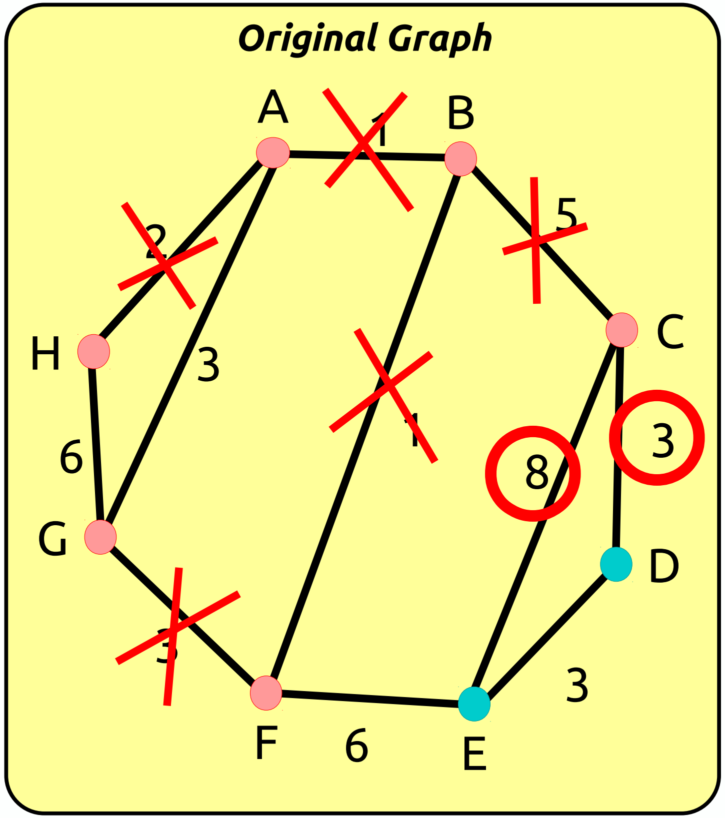
For each k from {1, 2, …, n}:

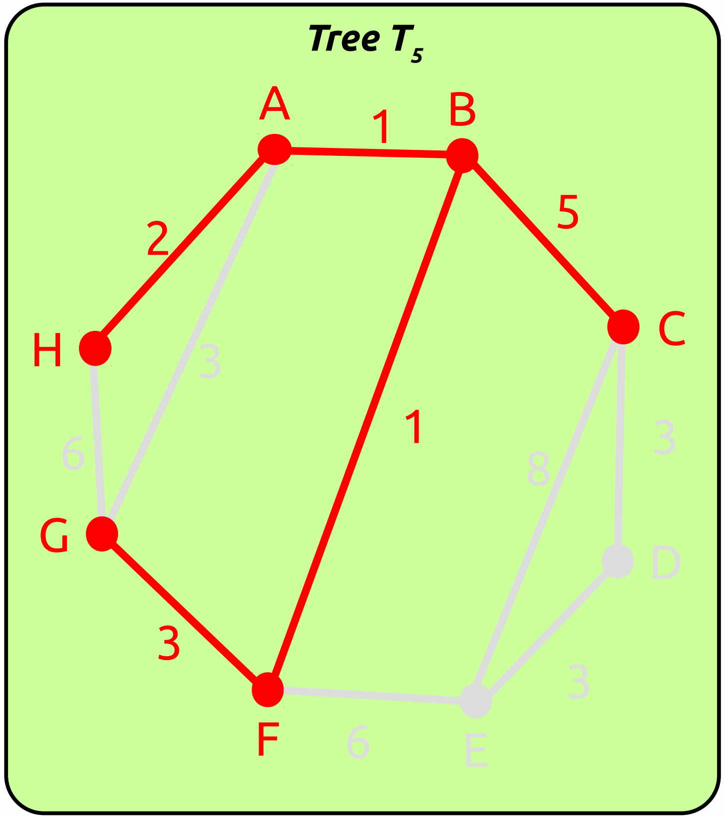
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

And we continue…





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

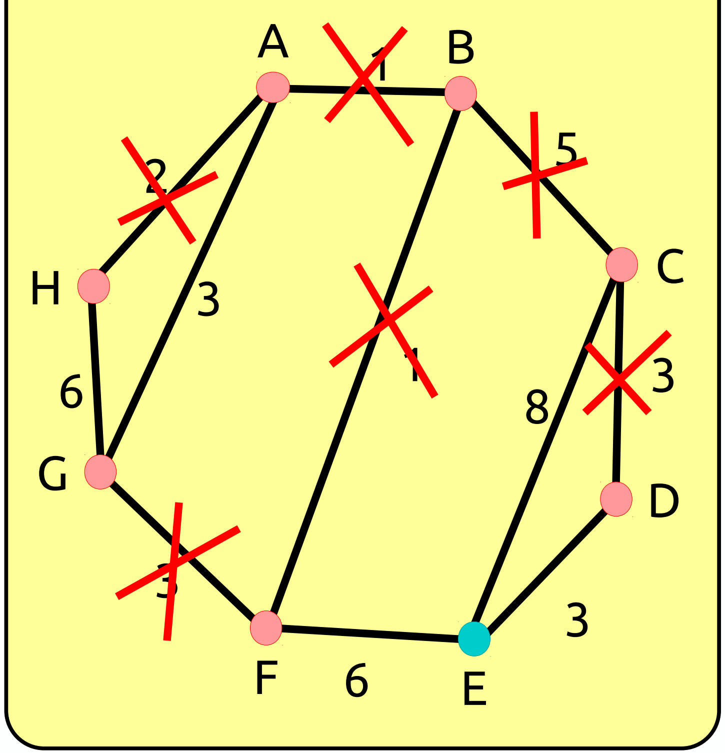
For each k from {1, 2, …, n}:

- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

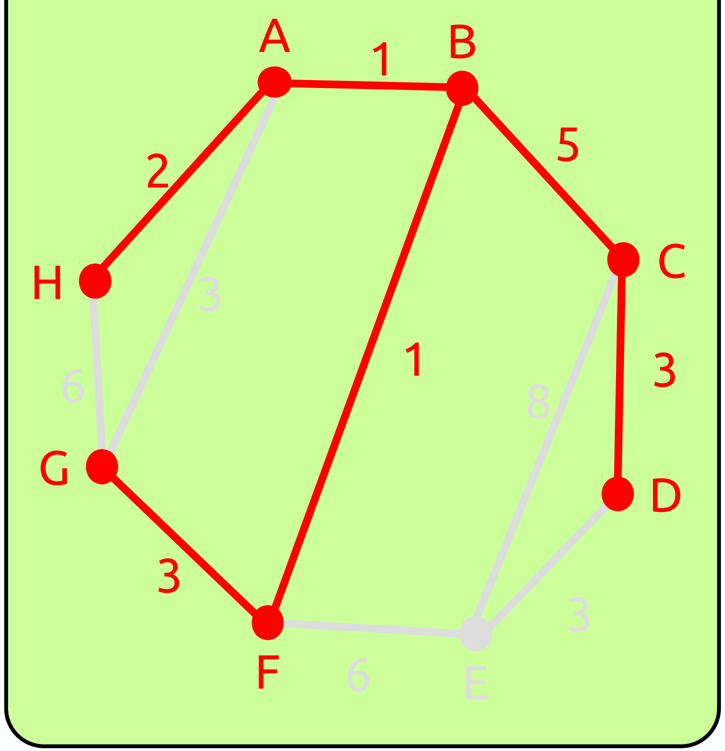
Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

And we continue…



Original Graph



Tree T_6

Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

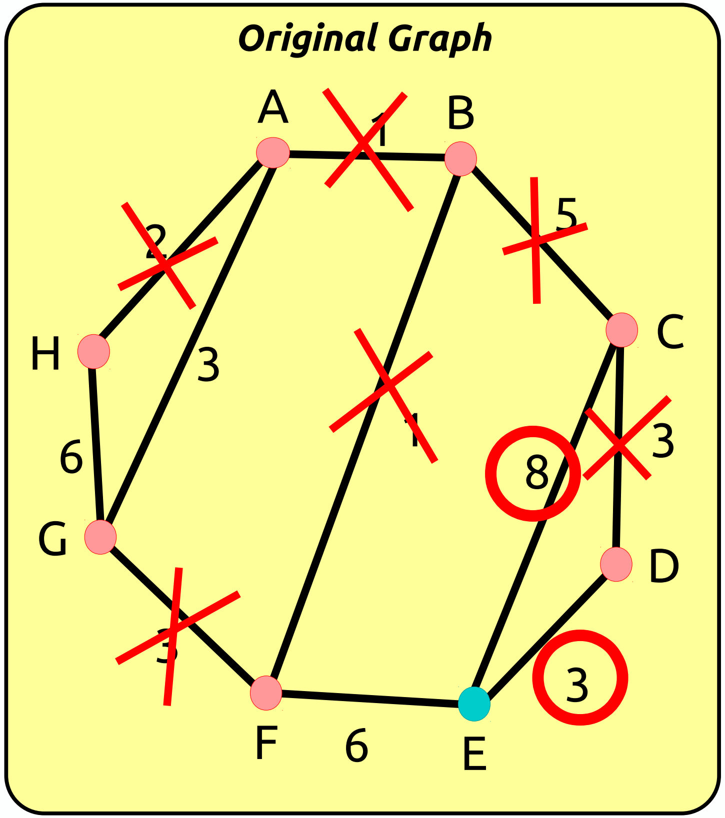
For each k from {1, 2, …, n}:

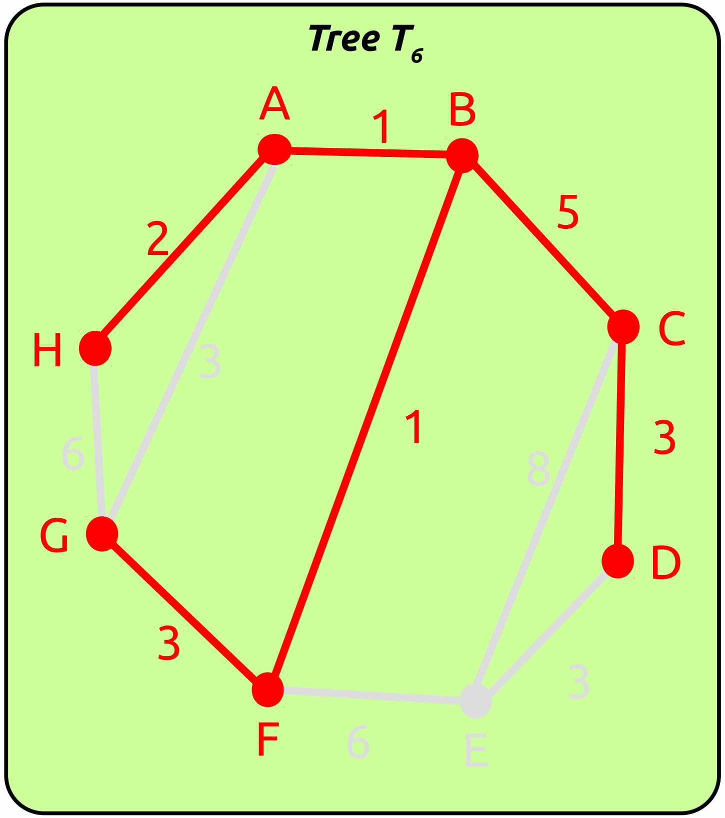
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

And we continue…





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

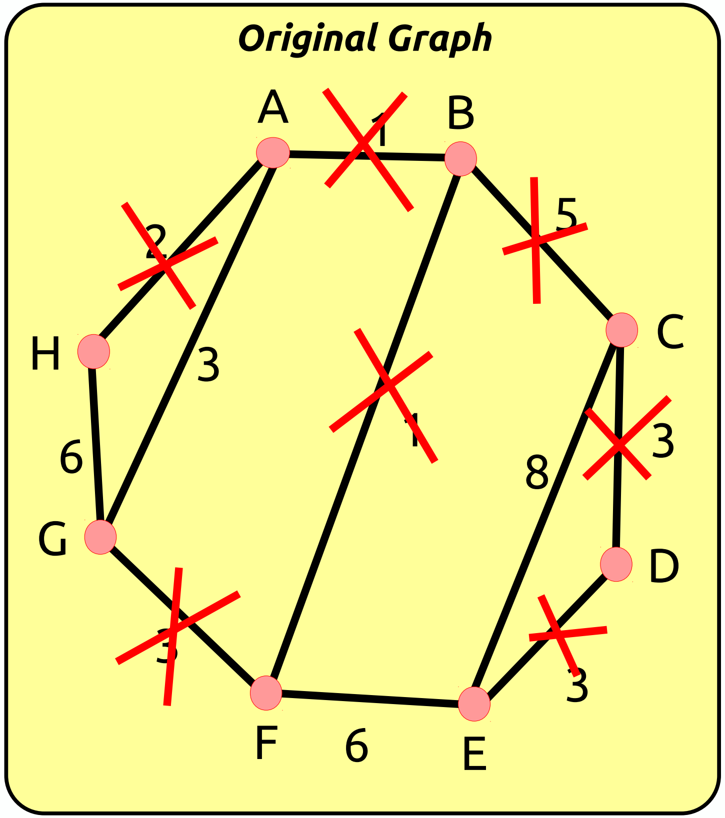
For each k from {1, 2, …, n}:

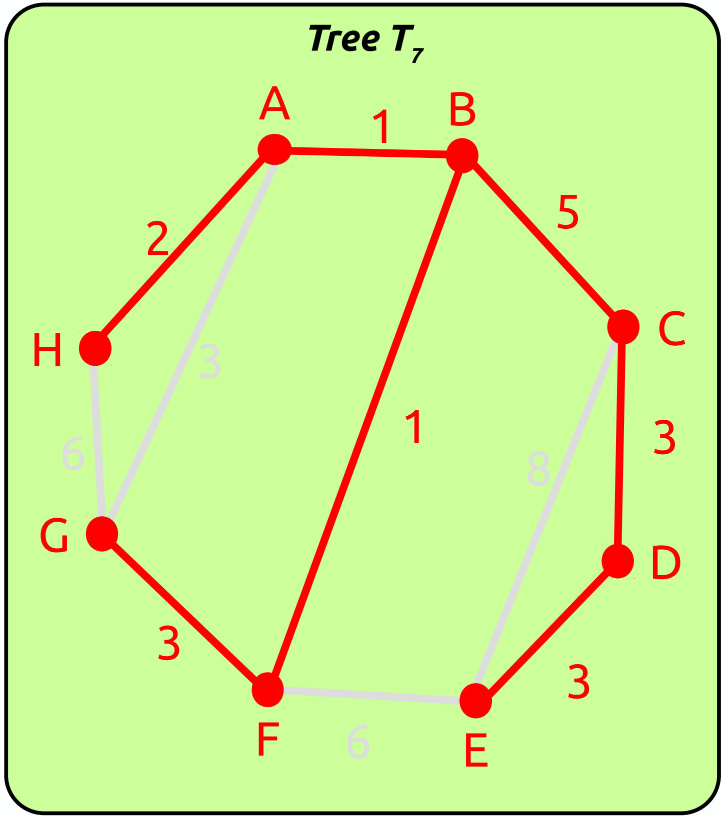
- Let $E_k = \{e$ an edge in $G : e$ has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

3. Spanning Tree Algorithms

And now we have covered all the nodes in the graph.





Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

For each k from {1, 2, …, n}:

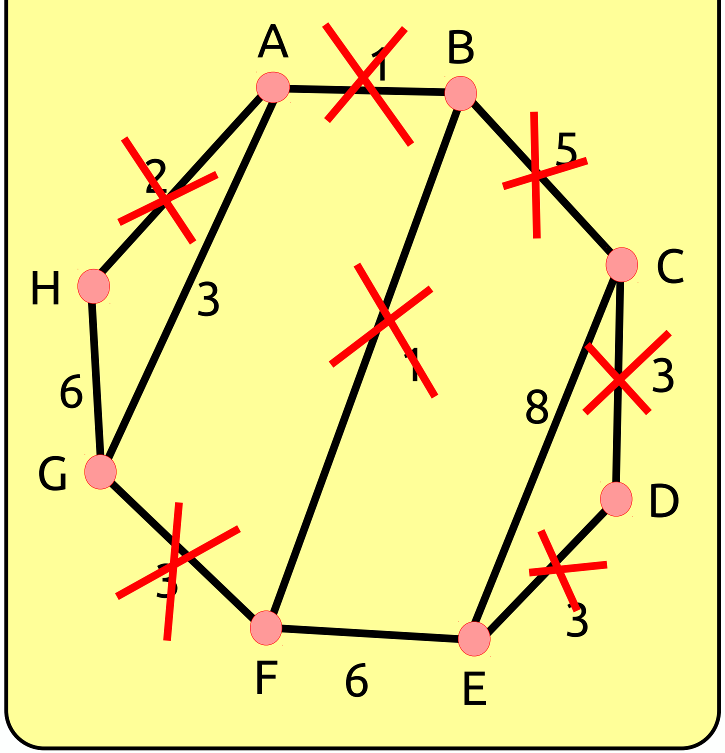
- Let $E_k = \{e$ an edge in G : e has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

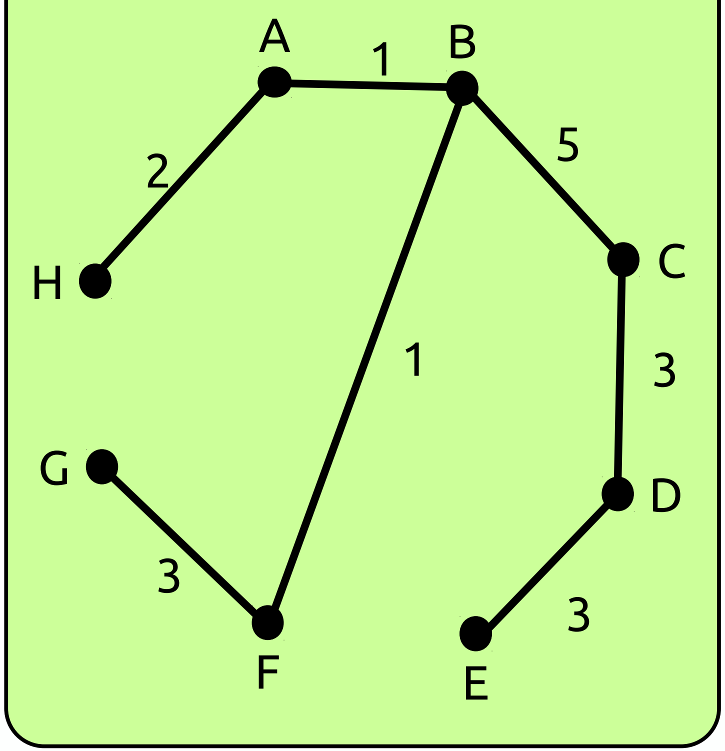
3. Spanning Tree Algorithms

The result is a minimal spanning tree.

Original Graph



Tree T_7



Notes

Prim's Spanning Tree Algorithm:

Input: A simple, connected graph G with $n + 1$ nodes.

Initialize: Let v_0 be any node in G, and let $T_0 = \{v_0\}$ be a tree with one node and no edges.

For each k from {1, 2, …, n}:

- Let $E_k = \{e$ an edge in $G : e$ has one endpoint in T_{k-1} and the other endpoint not in $T_{k-1}\}$.
- Let e_k be the edge in E_k with the smallest weight. (In case of a tie, choose any edge of the smallest weight.)
- Let T_k be the tree obtained by adding edge e_k (along with its node not already in T_{k-1}) to T_{k-1}.

Output: The final result T_n is the tree returned by the algorithm.

Conclusion

Trees!