Minimum Spanning Trees

Algorithms and Applications

Varun Ganesan

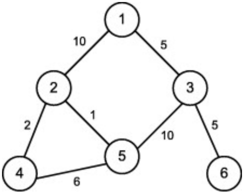
18.304 Presentation

Outline

Outline

Graphs

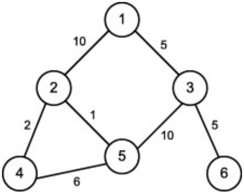
- In graph theory, a **graph** is an ordered pair $G = (V, E)$ comprising a set of **vertices** or nodes together with a set of **edges**.



- **Edges** are 2-element subsets of V which represent a connection between two vertices.
  - Edges can either be directed or undirected.
  - Edges can also have a **weight** attribute.

Graphs

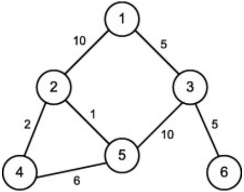
- In graph theory, a **graph** is an ordered pair $G = (V, E)$ comprising a set of **vertices** or nodes together with a set of **edges**.



- **Edges** are 2-element subsets of V which represent a connection between two vertices.
    - Edges can either be directed or undirected.
    - Edges can also have a **weight** attribute.

Graphs

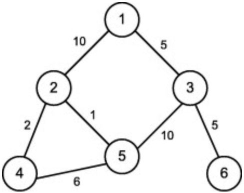
- In graph theory, a **graph** is an ordered pair $G = (V, E)$ comprising a set of **vertices** or nodes together with a set of **edges**.



- **Edges** are 2-element subsets of V which represent a connection between two vertices.
  - Edges can either be directed or undirected.
  - Edges can also have a **weight** attribute.

Graphs

- In graph theory, a **graph** is an ordered pair $G = (V, E)$ comprising a set of **vertices** or nodes together with a set of **edges**.



- **Edges** are 2-element subsets of V which represent a connection between two vertices.
  - Edges can either be directed or undirected.
  - Edges can also have a **weight** attribute.

Connectivity

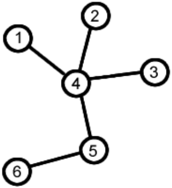
- A graph is **connected** when there is a path between every pair of vertices.
- A **cycle** is a path that starts and ends with the same vertex.
- A **tree** is a connected, acyclic graph.

Connectivity

- A graph is **connected** when there is a path between every pair of vertices.
- A **cycle** is a path that starts and ends with the same vertex.
- A **tree** is a connected, acyclic graph.

Connectivity

- A graph is **connected** when there is a path between every pair of vertices.
- A **cycle** is a path that starts and ends with the same vertex.
- A **tree** is a connected, acyclic graph.



Outline

1. Definitions
   - Graph Terminology
   - Minimum Spanning Trees
2. Common Algorithms
   - Kruskal's Algorithm
   - Prim's Algorithm
3. Applications

Spanning Trees

- Formally, for a graph $G = (V, E)$, the spanning tree is $E' \subseteq E$ such that:
  - $\exists u \in V : (u, v) \in E' \vee (v, u) \in E' \; \forall v \in V$
  - In other words: the subset of edges spans all vertices.
  - $|E'| = |V| - 1$
  - In other words: the number of edges is one less than the number of vertices, so that there are no cycles.

Spanning Trees

- Formally, for a graph $G = (V, E)$, the spanning tree is $E' \subseteq E$ such that:
  - $\exists u \in V : (u, v) \in E' \vee (v, u) \in E' \; \forall v \in V$
  - In other words: the subset of edges spans all vertices.
  - $|E'| = |V| - 1$
  - In other words: the number of edges is one less than the number of vertices, so that there are no cycles.

Spanning Trees

- Formally, for a graph $G = (V, E)$, the spanning tree is $E' \subseteq E$ such that:
  - $\exists u \in V : (u, v) \in E' \vee (v, u) \in E' \; \forall v \in V$
  - In other words: the subset of edges spans all vertices.

Linear Graph

- $|E'| = |V| - 1$
- In other words: the number of edges is one less than the number of vertices, so that there are no cycles.

What Makes A Spanning Tree The Minimum?

MST Criterion: When the *sum* of the edge weights in a spanning tree is the minimum over all spanning trees of a graph

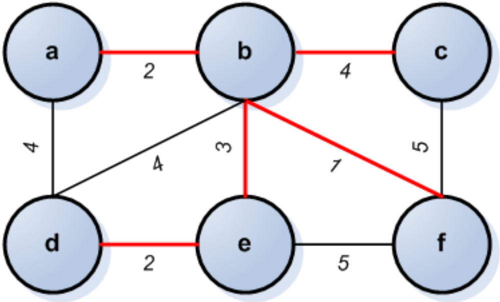


Figure: Suppose (b, f) is removed and (c, f) is added...

Outline

1. Definitions
   - Graph Terminology
   - Minimum Spanning Trees
2. Common Algorithms
   - Kruskal's Algorithm
   - Prim's Algorithm
3. Applications

Main Idea

- Start with $|V|$ disjoint components.
- Consider lesser weight edges first to incrementally connect components.
- Make certain to avoid cycles.
- Continue until spanning tree is created.

Main Idea

- Start with $|V|$ disjoint components.
- Consider lesser weight edges first to incrementally connect components.
- Make certain to avoid cycles.
- Continue until spanning tree is created.

Main Idea

- Start with $|V|$ disjoint components.
- Consider lesser weight edges first to incrementally connect components.
- Make certain to avoid cycles.
- Continue until spanning tree is created.

Main Idea

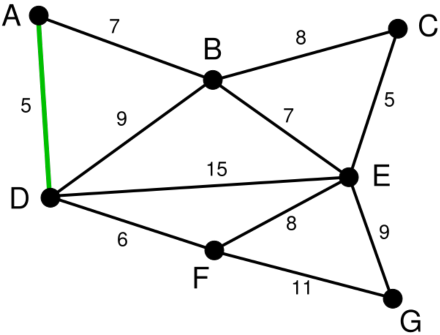
- Start with $|V|$ disjoint components.
- Consider lesser weight edges first to incrementally connect components.
- Make certain to avoid cycles.
- Continue until spanning tree is created.

Pseudocode

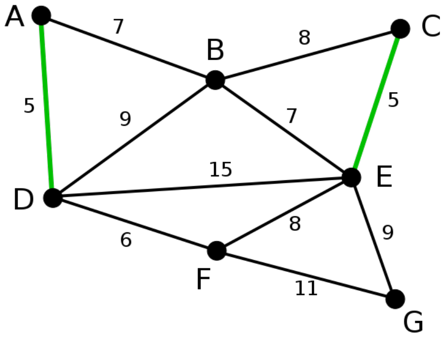
Kruskal's Algorithm

$A = 0$
foreach $v \in G.V$:
MAKE-SET(v)
foreach (u, v) ordered by $weight(u, v)$, increasing:
if $FIND\text{-}SET(u) \neq FIND\text{-}SET(v)$:
$A = A \cup (u, v)$
$UNION(u, v)$
return A

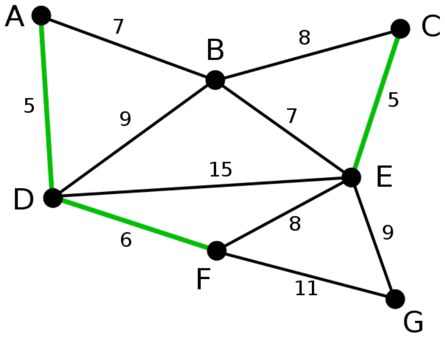
Example



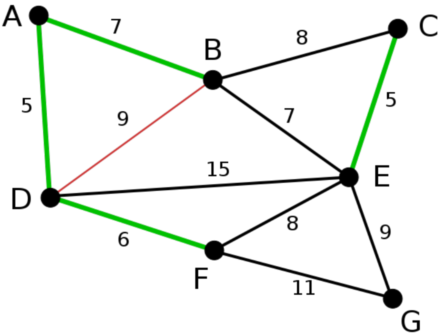
Example



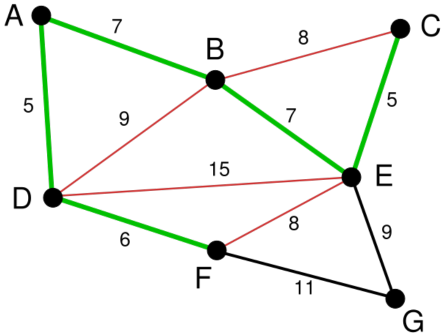
Example



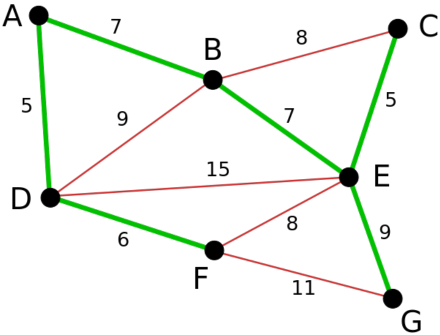
Example



Example



Example



Proof Of Correctness

- Spanning Tree Validity
  - By avoiding connecting two already connected vertices, output has no cycles.
  - If G is connected, output must be connected.
- Minimality
  - Consider a lesser total weight spanning tree with at least one different edge $e = (u, v)$.
  - If e leads to less weight, then e would have been considered before some edge that connects u and v in our output.

Proof Of Correctness

- Spanning Tree Validity
  - By avoiding connecting two already connected vertices, output has no cycles.
  - If G is connected, output must be connected.
- Minimality
  - Consider a lesser total weight spanning tree with at least one different edge $e = (u, v)$.
  - If e leads to less weight, then e would have been considered before some edge that connects u and v in our output.

Proof Of Correctness

- Spanning Tree Validity
  - By avoiding connecting two already connected vertices, output has no cycles.
  - If G is connected, output must be connected.
- Minimality
  - Consider a lesser total weight spanning tree with at least one different edge $e = (u, v)$.
  - If e leads to less weight, then e would have been considered before some edge that connects u and v in our output.

Proof Of Correctness

- Spanning Tree Validity
  - By avoiding connecting two already connected vertices, output has no cycles.
  - If G is connected, output must be connected.
- Minimality
  - Consider a lesser total weight spanning tree with at least one different edge $e = (u, v)$.
  - If e leads to less weight, then e would have been considered before some edge that connects u and v in our output.

Proof Of Correctness

- Spanning Tree Validity
  - By avoiding connecting two already connected vertices, output has no cycles.
  - If G is connected, output must be connected.
- Minimality
  - Consider a lesser total weight spanning tree with at least one different edge $e = (u, v)$.
  - If e leads to less weight, then e would have been considered before some edge that connects u and v in our output.

Proof Of Correctness

- Spanning Tree Validity
  - By avoiding connecting two already connected vertices, output has no cycles.
  - If G is connected, output must be connected.
- Minimality
  - Consider a lesser total weight spanning tree with at least one different edge $e = (u, v)$.
  - If e leads to less weight, then e would have been considered before some edge that connects u and v in our output.

Outline

1. Definitions
   - Graph Terminology
   - Minimum Spanning Trees
2. Common Algorithms
   - Kruskal's Algorithm
   - Prim's Algorithm
3. Applications

Main Idea

- Start with *one* (any) vertex.
- Branch outwards to grow your connected component.
- Consider only edges that leave the connected component.
- Add smallest considered edge to your connected component.
- Continue until a spanning tree is created.

Main Idea

- Start with *one* (any) vertex.
- Branch outwards to grow your connected component.
- Consider only edges that leave the connected component.
- Add smallest considered edge to your connected component.
- Continue until a spanning tree is created.

Main Idea

- Start with *one* (any) vertex.
- Branch outwards to grow your connected component.
- Consider only edges that leave the connected component.
- Add smallest considered edge to your connected component.
- Continue until a spanning tree is created.

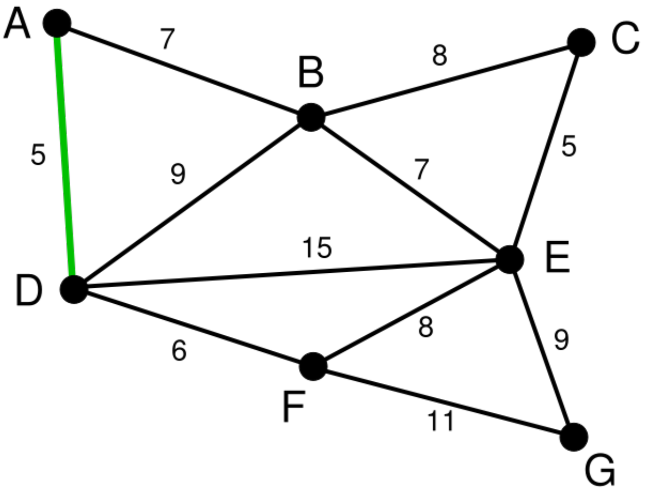
Main Idea

- Start with *one* (any) vertex.
- Branch outwards to grow your connected component.
- Consider only edges that leave the connected component.
- Add smallest considered edge to your connected component.
- Continue until a spanning tree is created.

Main Idea

- Start with *one* (any) vertex.
- Branch outwards to grow your connected component.
- Consider only edges that leave the connected component.
- Add smallest considered edge to your connected component.
- Continue until a spanning tree is created.

Example



Proof of Correctness

Both the spanning tree and minimality argument are nearly identical for Prim's as they are for Kruskal's.

Some Applications

- ***Taxonomy***
- ***Clustering Analysis***
- ***Traveling Salesman Problem Approximation***