Algorithms and Data Structures

Module 2

Lecture 6
Minimum spanning trees

Adigeev Mikhail Georgievich

mgadigeev@sfedu.ru

Practical case

Let us consider several sites and a set of roads that connect the sites.

The task is to repair some roads to satisfy the following conditions:

1) There is a way to go from any site to any other site by repaired roads.
2) The cost of repair is minimal. The estimated cost of repair is known for each road.

This is a practical situation. Let us build a mathematical model for it.

Mathematical model

Sites and roads can be represented as vertices and edges of graph $G(V, E)$.

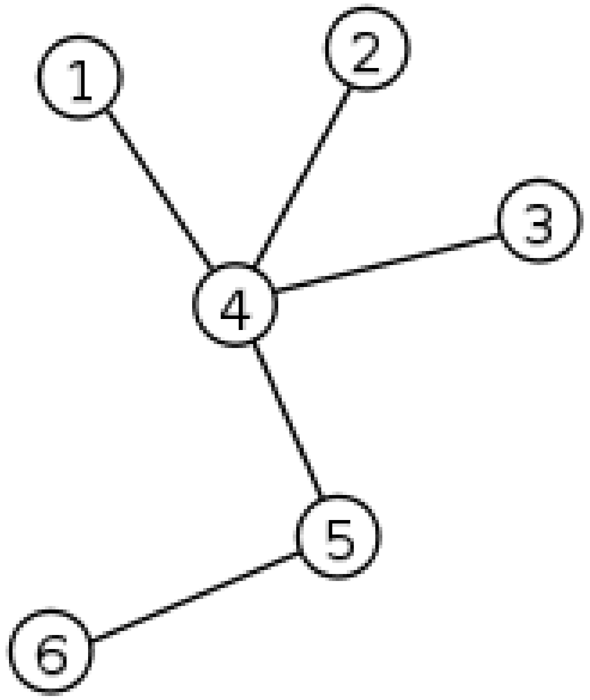
The solutions can be represented as edge subset E'. Consider the partial graph $G'(V, E')$. What are the conditions that G' must satisfy?

1) There is a way to go from any site to any other site by repaired roads. => the graph $G'(V, E')$ must be connected.
2) The cost of repair is minimal. => we need to build the minimal cost connected graph.

Trees

Tree is a connected acyclic graph, i.e. a graph without (undirected) cycles.

An acyclic (not necessarily connected) graph is called a ***forest***.

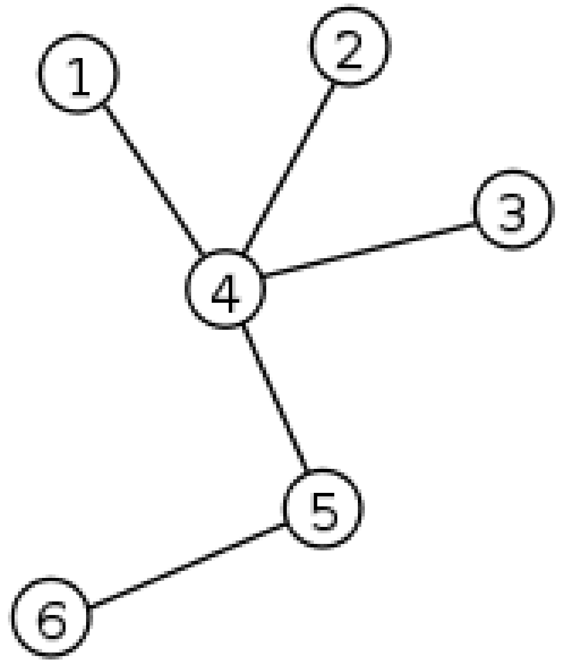


Trees

Theorem (properties of trees).

A graph $G(V, E)$ is a tree iff any of the following equivalent conditions hold:

1) *G* is connected and acyclic (contains no cycles).
2) *G* is acyclic, and a simple cycle is formed if any edge is added to *G*.
3) *G* is connected, but would become disconnected if any single edge is removed from *G*.
4) Any two vertices in *G* can be connected by a unique simple path.
5) *G* is connected and has *n* − 1 edges ($n = |V|$).
6) *G* has no simple cycles and has *n* − 1 edges.



Minimum spanning tree

So, the solutions must be a *minimum spanning tree (MST)*.

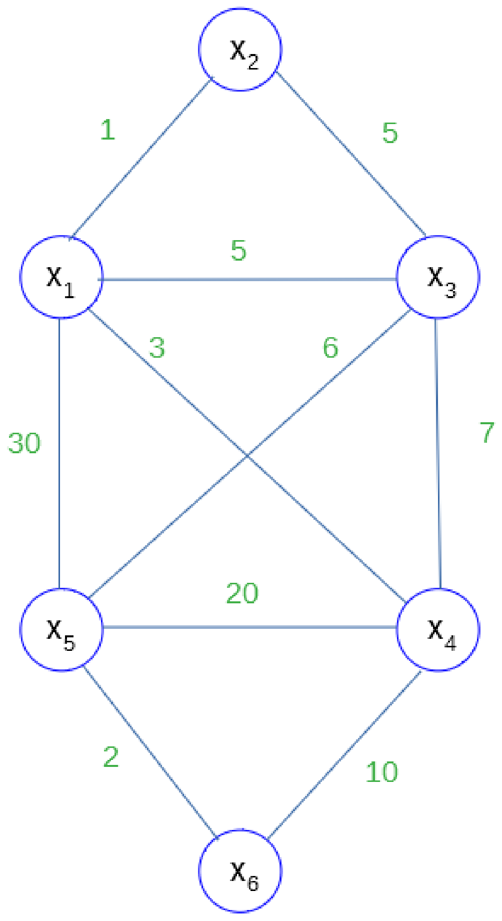
✓ A *tree*, i.e. a connected acyclic subgraph.

✓ Spanning tree. *Spanning* subgraph is a connected subgraph that contains all vertices of the graph.

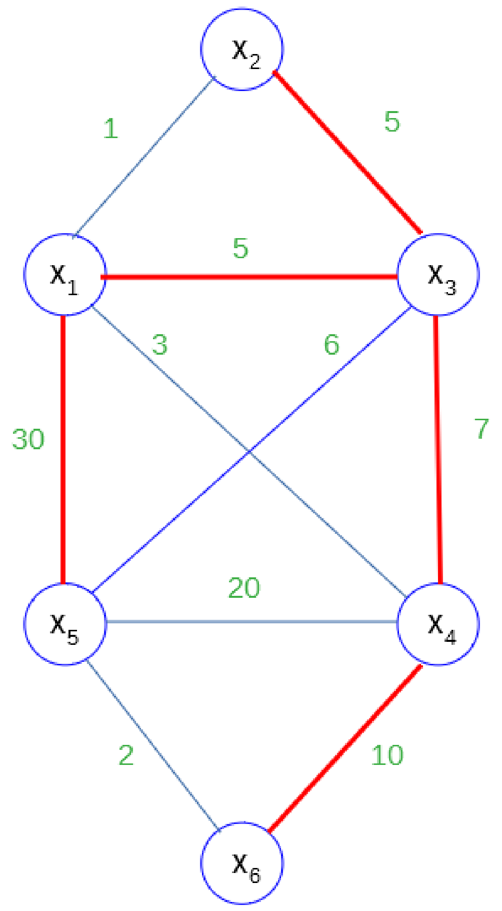
✓ *Minimum* spanning tree is a spanning tree of minimum weight.

Minimum spanning tree

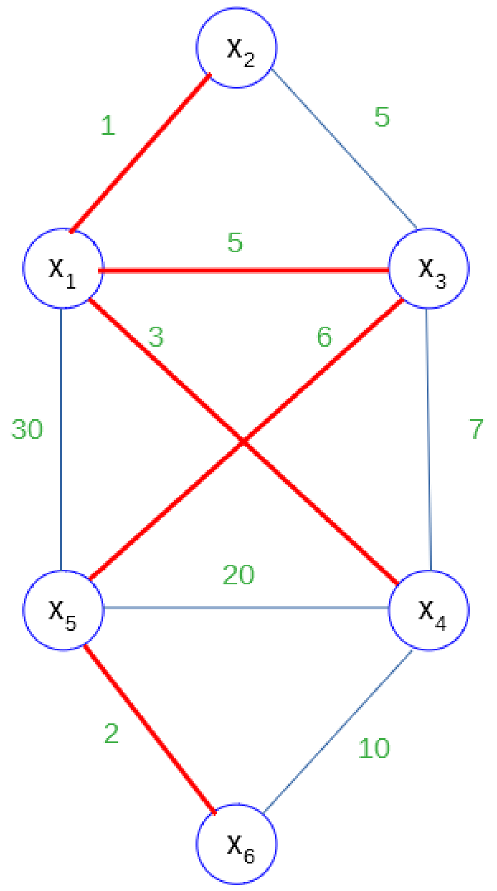
Weighted graph



Spanning tree, weight = 57



Minimum spanning tree, weight = 17



MST: algorithms

How can we build a MST for the given graph?

- ✓ A brute force search through all spanning trees of the given graph.

  Unfortunately, the Cayley's theorem states that in the worst case (for a complete graph) the quantity of spanning trees for an *n*-vertex is n^{n-2}.

- ✓ Develop and apply more efficient algorithm.

For the problem of building a minimum spanning tree, we can apply a ***greedy strategy***.

Greedy algorithms

Key characteristics of a greedy algorithm:

1. Can solve an optimization problem.
2. Builds solution iteratively, adding one element after another.
3. At each step, adds the element which is the best at the current situation.
4. Does not revise the decisions (one-pass algorithm).

Greedy algorithms

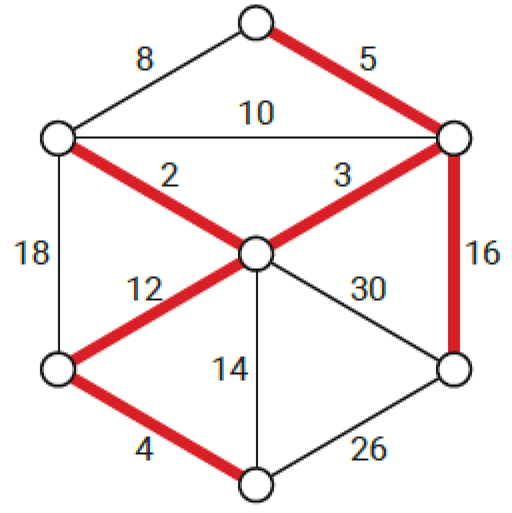
- One can construct many different greedy algorithms for a problem.
- Greedy solution may be bad (not optimal).
- Greedy algorithms are usually efficient.

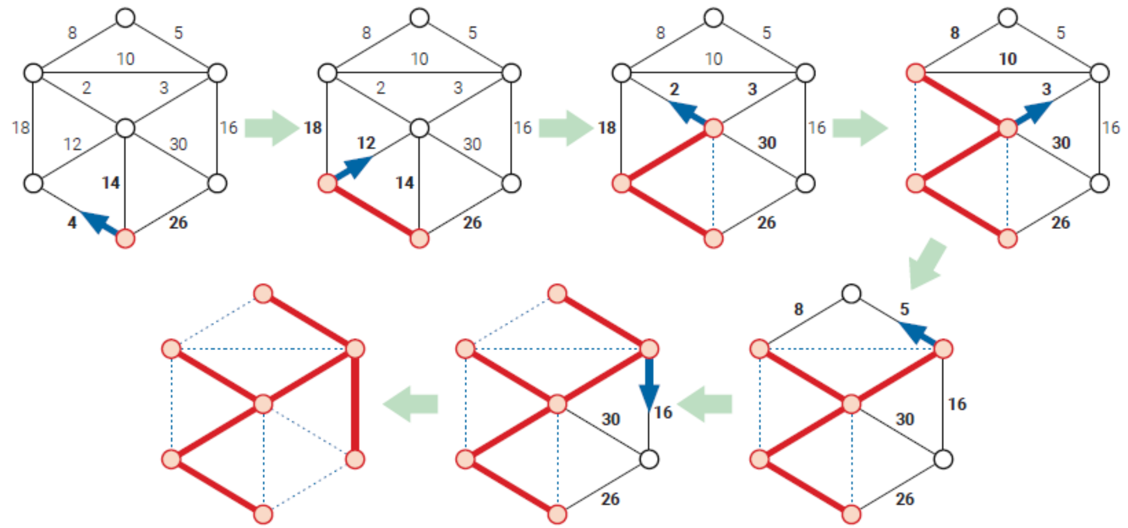
MST: algorithms

A greedy strategy: start with an empty subgraph; add the *lightest* edge such that it does not create a cycle on the subgraph (the lightest *safe* edge).

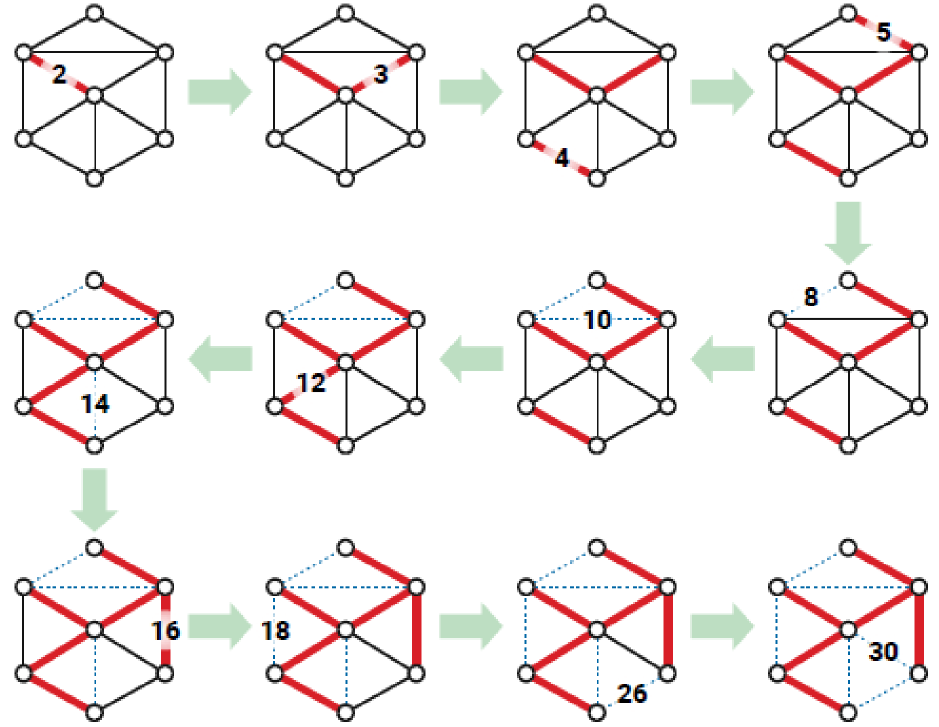
- Kruskal's algorithm: build a *spanning* forest, adding edges until there is one component (tree).
- Prim's algorithm: build the *tree*, adding edges until it spans the graph.

MST: algorithms





Prim's algorithm



Kruskal's algorithm

http://jeffe.cs.illinois.edu/teaching/algorithms/

Kruskal's algorithm

Given a connected graph $G(V, E), |V| = n, |E| = m$.

1. $T = \emptyset$
2. Sort the set of edges by increasing their weights.
3. Scan the sequence of edges. For each edge:
   - If the current edge is safe: add this edge to T.
   - Otherwise: just skip this edge (=do nothing with it).

Kruskal's algorithm

Given a connected graph $G(V, E), |V| = n, |E| = m$.

$O(n)$

1. $T = \emptyset$

$O(m \log m)$

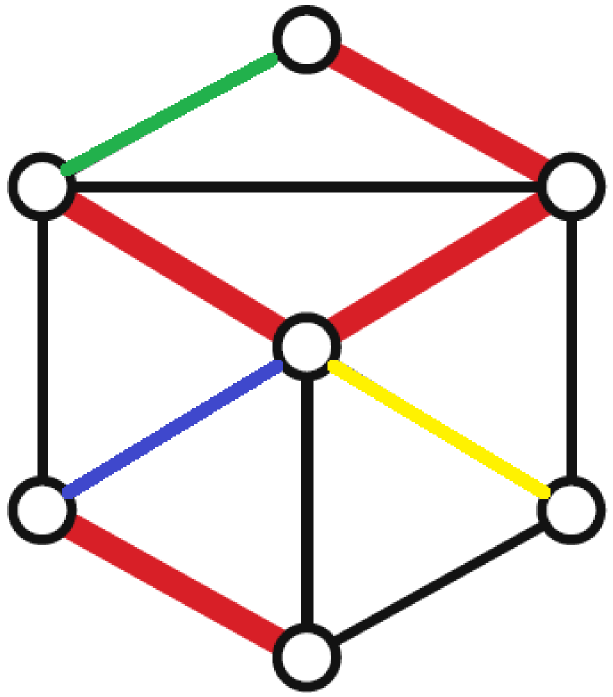
2. Sort the set of edges by increasing their weights.
3. Scan the sequence of edges. For each edge:
   - If the current edge is safe: add this edge to T.
   - Otherwise: just skip this edge (=do nothing with it).

m iteration

???

Kruskal's algorithm: safety check

Given the graph $G(V, E)$, spanning forest T and the current edge $e = (u, v) \in E$, how can we check whether e is safe (=adding e to T does not create a cycle)?



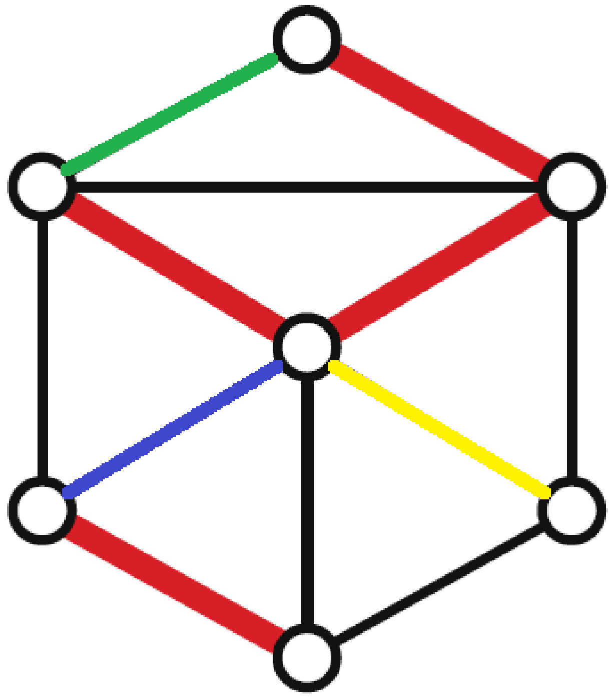
Red edges belong to *T*.

The blue and yellow edges are safe.

The green edge is unsafe.

Kruskal’s algorithm: safety check

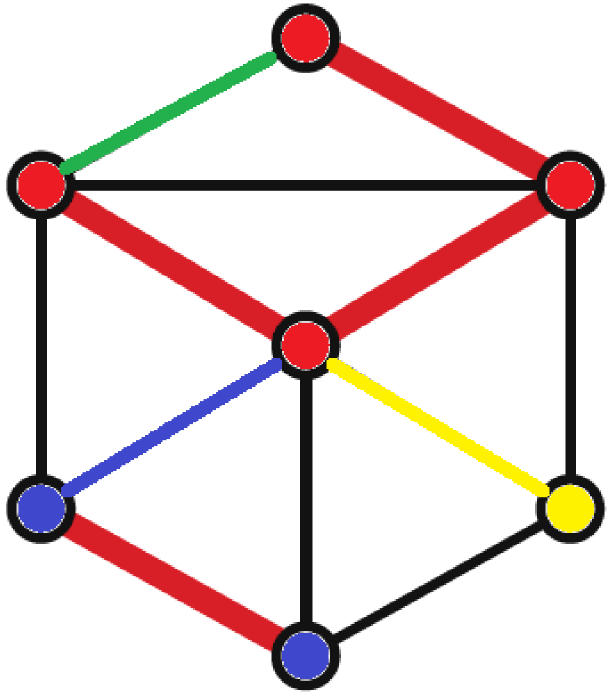
Given the graph $G(V, E)$, spanning forest T and the current edge $e = (u, v) \in E$, how can we check whether e is safe?



Naïve approach: add the new edge and check graph T ∪ $\{e\}$ for presence of cycles. Algorithm: a modification of DFS. Complexity: $O(m) = O(n^2)$ for each check and $O(m^2) = O(n^4)$ for the total time.

Kruskal's algorithm: safety check

Given the graph $G(V, E)$, spanning forest T and the current edge $e = (u, v) \in E$, how can we check whether e is safe (=adding e to T does not create a cycle)?



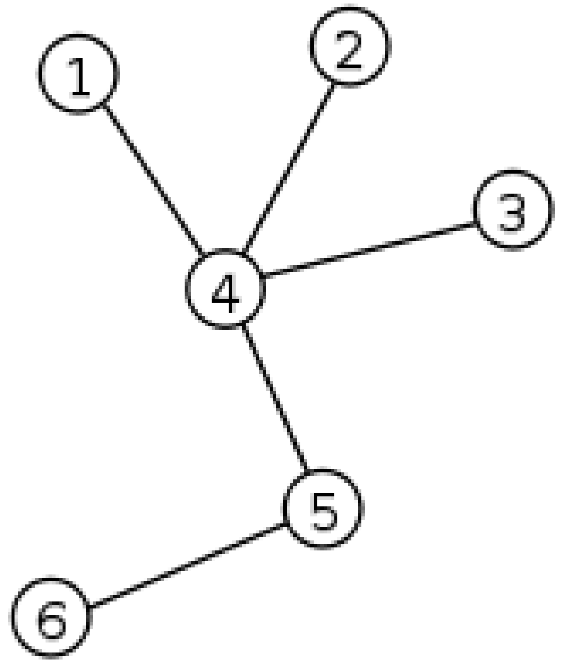
Rule: $e = (u, v)$ is safe iff its endpoints u and v belong to different components of T; otherwise e is unsafe.

Kruskal’s algorithm: safety check

Theorem (properties of trees).

A graph $G(V, E)$ is a tree iff any of the following equivalent conditions hold:

1) G is connected and acyclic (contains no cycles).
2) G is acyclic, and a simple cycle is formed if any edge is added to G.
3) G is connected, but would become disconnected if any single edge is removed from G.
4) Any two vertices in G can be connected by a unique simple path.
5) G is connected and has $n - 1$ edges ($n = |V|$).
6) G has no simple cycles and has $n - 1$ edges.



Kruskal's algorithm: safety check

Rule: $e = (u, v)$ is safe iff its endpoints u and v belong to different components of T; otherwise e is unsafe.

$\Rightarrow$ We need to keep a component ID for each vertex, and we also need to update this information after adding a new edge to the forest.

$\Rightarrow$ We need a Union-Find (Merge-Find) data structure that keeps a collection of disjoint subsets (components) of a set and implements operations:

- `MakeSet(v)`: creates a set $\{v\}$.
- `Find(v)`: returns the unique ID of the subset containing v.
- `Union(u,v)`: unions (merges) subsets containing u and v to a single subset.

Kruskal’s algorithm: safety check

Theorem. Union-Find can be implemented with the following time complexities:

1) `MakeSet` takes $O(1)$ time / operation; total time is $O(n)$.
2) `Find` takes $O(\log n)$ time / operation; total time is $O(m \log n)$.
3) `Union` takes $O(1)$ time / operation; total time is $O(n)$.

The total time complexity of Kruskal’s algorithm: $O(m \log m) = O(m \log n)$.

Kruskal’s algorithm

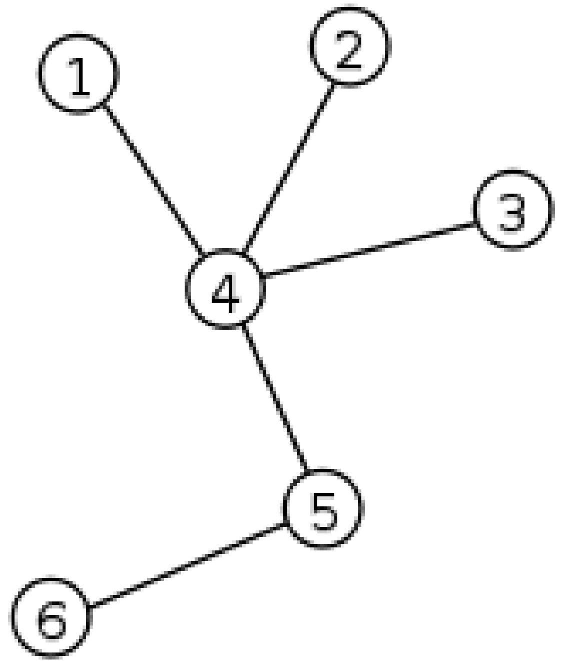
We need to prove that Kruskal’s algorithm is correct. For this purpose we need the tree theorem and one of the minimality criteria for a spanning tree.

Trees

Theorem (properties of trees).

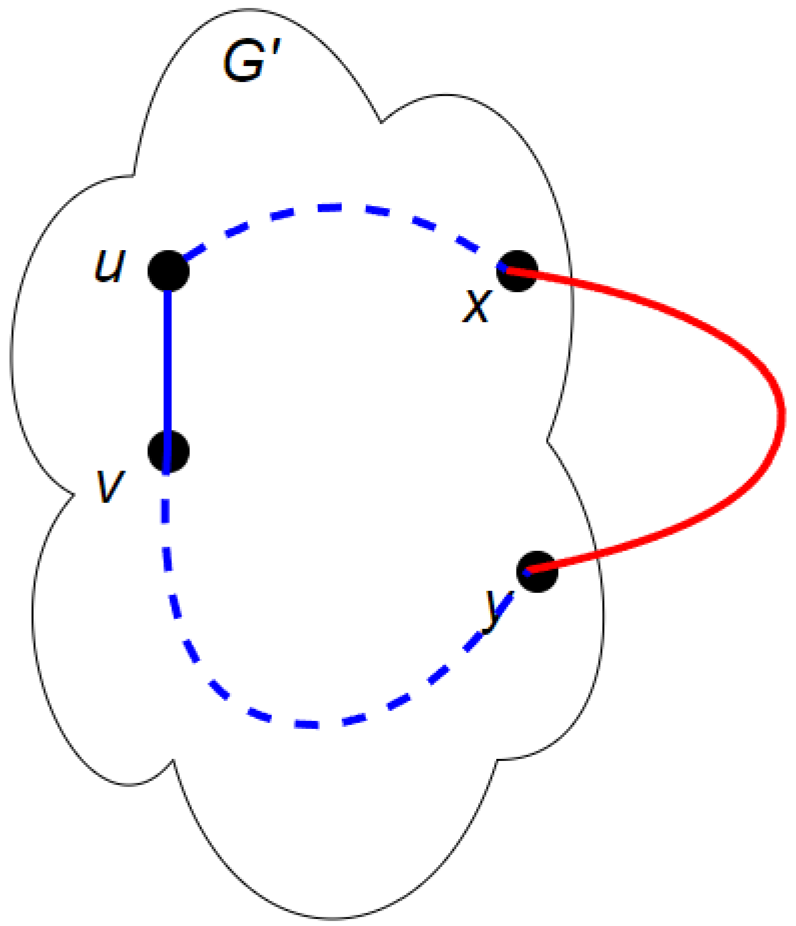
A graph $G(V, E)$ is a tree iff any of the following equivalent conditions hold:

1) G is connected and acyclic (contains no cycles).
2) G is acyclic, and a simple cycle is formed if any edge is added to G.
3) G is connected, but would become disconnected if any single edge is removed from G.
4) Any two vertices in G can be connected by a unique simple path.
5) G is connected and has $n - 1$ edges ($n = |V|$).
6) G has no simple cycles and has $n - 1$ edges.



Kruskal’s algorithm

Consider a spanning tree G' and an edge e not contained in G'. By theorem, the graph arising from G' by adding e contains a unique cycle. Let us denote this cycle by $C_{G'}(e)$.



Kruskal’s algorithm

Theorem (Cycle Criterion).

A spanning tree $G'(V',E')$ is minimal iff for each non-tree edge $(x,y) \in E \backslash E'$ and any edge $(u,v) \in C_{G'}(x,y)$, the following condition holds: $w(u,v) \leq w(x,y)$.