Connectedness in Graphs

- $G = (V, E)$ is a graph.
- *Connectivity* is one of the basic concepts of graph theory.
- A graph is connected when there is a path between every pair of vertices.
- In an undirected graph $G$, two vertices $u$ and $v$ are called *connected* if $G$ contains a path from $u$ to $v$.
- A directed graph is called *weakly connected* if replacing all of its directed edges with undirected edges produces a connected graph.
**Cut**: A partition of the vertices of a graph into two disjoint subsets. Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition. These edges are said to cross the cut.

In a flow network, an **s-t cut** requires source and sink to be in different subsets, and its cut-set only consists of edges going from source’s side to the sink’s side. The capacity of an s-t cut is $= \text{sum of capacity of each edge in the cut-set.}$
A cut $C = (S, T)$ is a partition of $V$ into two subsets $S$ and $T$. $C = (S, T) = \{(u, v) \in E \mid u \in S, v \in T \}$. (min-cut vs max-cut)

- $S = \{a, b, c\}$ and $T = \{d, e\}$, and in other $S = \{1, 3, 4\}$ and $T = \{2, 5\}$
It is strongly connected, simply strong.

A cut, vertex cut, or separating set of a connected graph $G$ is a set of vertices whose removal renders $G$ disconnected.

A complete graph with $n$ vertices, denoted $K_n$, has no vertex cuts at all, but connectivity $k(K_n) = n - 1$. 
Any graph $G$ (complete or not) is said to be $k$-connected if it contains at least $k+1$ vertices, but does not contain a set of $k-1$ vertices whose removal disconnects the graph; and $k(G)$ is defined as the largest $k$ such that $G$ is $k$-connected.

Thus, a connected graph is 1-connected and a biconnected graph is 2-connected.

Analogous concepts can be defined for edges. In the simple case in which cutting a single, specific edge would disconnect the graph, that edge is called a bridge.
Computational aspects:
The problem of determining whether two vertices in a graph are connected can be solved efficiently using a search algorithm,

1. Begin at any arbitrary node of the graph, \( G \)
2. Proceed from that node using either depth-first or breadth-first search, counting all nodes reached.
3. Once the graph has been entirely traversed, if the number of nodes counted is equal to the number of nodes of \( G \), the graph is connected; otherwise it is disconnected.
One of the most important facts about connectivity in graphs is Menger’s theorem, which characterizes the connectivity and edge-connectivity of a graph in terms of the number of independent paths between vertices.

If $u$ and $v$ are vertices of a graph $G$, then a collection of paths between $u$ and $v$ is called independent if no two of them share a vertex.

The vertex-connectivity statement of Menger’s theorem:
Let $G$ be an undirected graph, $x$ and $y$ two nonadjacent vertices. Then size of the minimum vertex cut for $x$ and $y$ (the minimum number of vertices whose removal disconnects $x$ and $y$) is equal to the maximum number of pairwise vertex-independent paths from $x$ to $y$. 
The edge-connectivity version of Menger’s theorem is as follows:

Let $G$ be a finite undirected graph and $x$ and $y$ two distinct vertices. Then size of the minimum edge cut for $x$ and $y$ (the minimum number of edges whose removal disconnects $x$ and $y$) is equal to the maximum number of pairwise edge-independent paths from $x$ to $y$.

The number of mutually independent paths between $u$ and $v$ is $k(u, v)$, and the number of mutually edge-independent paths between $u$ and $v$ is $\lambda(u, v)$.

By Menger’s theorem, for any two vertices $u$ and $v$ in a connected graph $G$, the numbers $k(u, v)$ and $\lambda(u, v)$ can be determined efficiently using the max-flow min-cut algorithm.
The number of distinct connected labeled graphs with $n$ nodes is:

<table>
<thead>
<tr>
<th>$n$</th>
<th>graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
</tr>
<tr>
<td>5</td>
<td>728</td>
</tr>
<tr>
<td>6</td>
<td>26704</td>
</tr>
<tr>
<td>7</td>
<td>1866256</td>
</tr>
<tr>
<td>8</td>
<td>251548592</td>
</tr>
</tbody>
</table>
A Network is a directed graph (digraph) $D = (V, A)$ with a capacity function $C : A \rightarrow \mathbb{R}$ assigning arcs to non-negative real values. $V$ can be partitioned into three sets: the sources $X$, sinks $Y$, and intermediate $I$. $X$, $Y$ must be nonempty.

To a network we may associate a flow $f : V \rightarrow \mathbb{R}$ assigning arcs to non-negative real values such that $0 \leq f(a) \leq c(a)$ for $a \in A$ and $f_{in} = f_{out}$ for all $v \in I$, where,

$$f_{in}(v) = \sum_{uv \in A} f(uv), \quad (1)$$

and

$$f_{out}(v) = \sum_{vu \in A} f(uv), \quad (2)$$
Min-Cut Max-flow Theorem....

- In other words, the flow over any arc is no more than its capacity.
- The value of flow $f$, denoted by $val f$ is defined as,

$$\text{val } f = \sum_{x \in X} f_{\text{out}}(x) - f_{\text{in}}(x) \quad (3)$$

or, by the above notation, $\text{val } f = f_{\text{out}}(X) - f_{\text{in}}(X)$. Given a network, the natural optimization problem is: what is the maximum value attained by any flow?

- A **cut** $(S, \bar{S})$ in a network is the set of arcs

$\{ss \in A \mid s \in S, \bar{s} \in \bar{S}\}$

where $X \subseteq S \subseteq V - Y$ and $\bar{S} = V - S$. The capacity of a cut $K$, denoted as $\text{cap } K$, is defined as,

$$\text{cap } K = \sum_{a \in K} c(a) \quad (4)$$
Min-Cut Max-flow Theorem

- Finally, for each cut $K = (S, \bar{S})$ we can define an anticut $\bar{K} = (\bar{S}, S) = \{\bar{s}s \in A \mid s \in S, \bar{s} \in \bar{S}\}$.
- **Max-flow min-cut Theorem.** Maximum of all flow values (i.e., the value of the maximum flow), is equal to the minimum of all cut capacities (i.e., capacity of the minimum cut).

**Lemma**

*Given a network, for any flow $f$ and cut $K$ on the network, $\text{val } f \leq \text{cap } K$.***

**Proof.** Let $K = (S, \bar{S})$. As $S$ is comprised of sources and intermediates, clearly,

$$\text{val } f = f_{\text{out}}(X) - f_{\text{in}}(X) = f_{\text{out}}(S) - f_{\text{in}}(S)$$

since the intermediates contribute nothing to flow value.
Consider an arc with both end points in $S$: its flow is counted in both $f_{\text{out}}(S)$ and $f_{\text{in}}(S)$, and thus makes no net impact on the flow value. Therefore, the only arcs flows which positively impact $\text{val} f$ are those originating in $S$ and terminating in $\overline{S}$, which are precisely flows over cut $K$. Thus,

$$\text{val} \ f \leq \sum_{a \in K} f(a) \leq \sum_{a \in K} c(a) = \text{cap} \ K.$$ 

Some applications:

- Given any digraph with at least two vertices, designate some vertex $x$ the source and vertex $y$ the sink, and let all arcs have unit capacity.
Some applications

Then a flow on this network counts (via its value) a number of arc-disjoint directed $x, y$-paths, and a cut counts (via its capacity) a number of arcs whose deletion destroyes all $x, y$-paths.

Menger’s Theorem: Let $x, y$ be distinct vertices of a digraph $D$. The maximum number of arc-disjoint directed $x, y$-paths in $D$ equals the minimum number of arcs whose deletion destroyes all directed $x, y$-paths in $D$.

Similarly, let $x, y$ be distinct vertices of a graph $G$. The maximum number of edge-disjoint $x, y$-paths in $G$ equals the minimum number of edges whose deletion destroyes all $x, y$-paths in $G$. 
A minimum spanning tree of an undirected graph can be easily obtained using classical algorithms by Kruskal or Prim. A number of algorithms have been proposed to enumerate all spanning trees of an undirected graph.

Let in an undirected graph $G = (V, E)$, $E = \{(u, v) \mid u, v \in V\}$. In weighted graph, $w : E \to \mathbb{R}$, which assigned weight each edge (called cost).

Spanning tree is graph consisting all the vertices, and all are connected by minimum number of edges.
Finding minimum spanning trees

- **Kruskal’s algorithm.** Repeat until entire new graph $M$ has $n-1$ edges, and initially $M$ was empty. Add to $M$ the shortest edge, which does not make it a circle.

- **Prim’s Algorithm.** Repeat following until $M$ has $n-1$ edges, with $M$ initially empty. Add the shortest edge with $v_i \in M$ and $v_j \notin M$. 
Finding all the spanning trees

- Cayley’s formula counts the number of spanning trees on a complete graph. Cayley’s formula is a result in graph theory named after Arthur Cayley. It states that for every \( n \), the number of trees on \( n \) labeled vertices is \( n_{n-2} \). There are \( 2^{2-2} = 1 \) trees in \( K_2 \), \( 3^{3-2} = 3 \) trees in \( K_3 \), and \( 4^{4-2} = 16 \) trees in \( K_4 \).

- Suppose we have \( V \) nodes and \( E \) edges.
  1. Get all edges of the graph
  2. Get all possible combinations of \( V-1 \) out of \( E \) edges.
  3. Filter out non-spanning-tree out of the combinations (for a spanning tree, all nodes inside one set of \( V-1 \) edges should appear exactly once)

(Alternatively, make all edges of equal weight, and then find all minimum spanning trees).