Connectivity and Fault-Tolerance of Networks

Textbook Reading:

Algorithms: Special Topics

Chapter 3, Sections 3.3 and 3.4, pp. 86-92

Chapter 4, Subsection 4.2, pp. 124-125

Routing Tables

- A fundamental network problem is to route data from a source vertex *u* in the network to a destination vertex *v.*
- The condition of the network being connected ensures that a packet of data can be routed from *u* to *v* along a path from *u* to *v*.
- There are many possible paths joining *u* and *v*, so that we need a routing scheme in order to route the packet along a specific path.
- One such scheme utilizes a routing table associated with a spanning tree *T* rooted at the destination node *v*. For each node *u* different from *v*, the routing table stores the parent node of *u* in *T*.
- A packet at any node *u* can then be rooted to *v* by successively moving to parent nodes until *v* is reached. In order to optimize the length of the path that is generated, often *T* is chosen to be a shortest path tree.
- By associating a routing table (spanning tree rooted at *v*) with each node *v* of the network, we can route packets from any source node *u* to any destination node *v.*

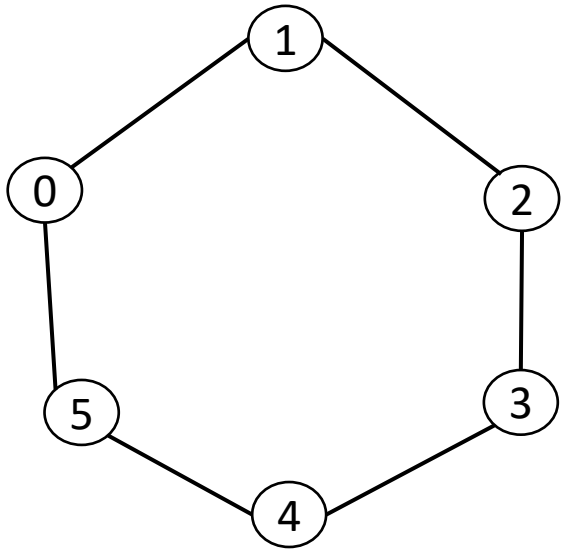
Independent Spanning Trees and Fault-Tolerant Routing Tables

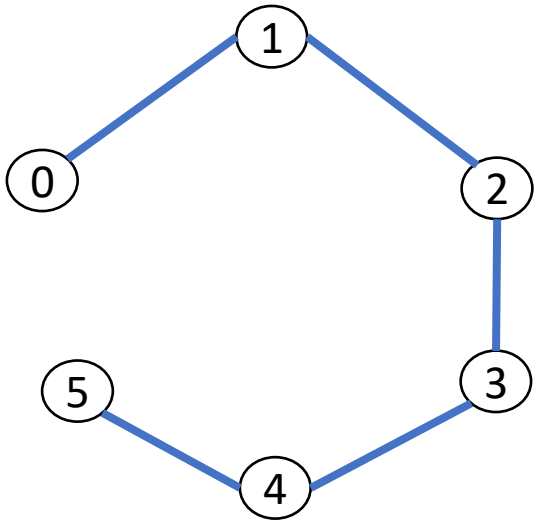
Two spanning trees T_1 and T_2 rooted at a vertex v are **independent** if, for any vertex u, the path in T_1 from u to v and the path in T_2 from u to v have no interior vertices in common.

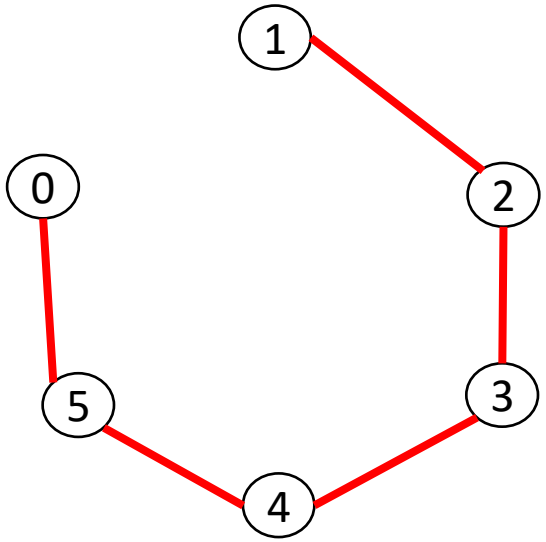
A set of k spanning trees are independent if they are pairwise independent.

Example of two independent spanning trees rooted at 0

$G =$





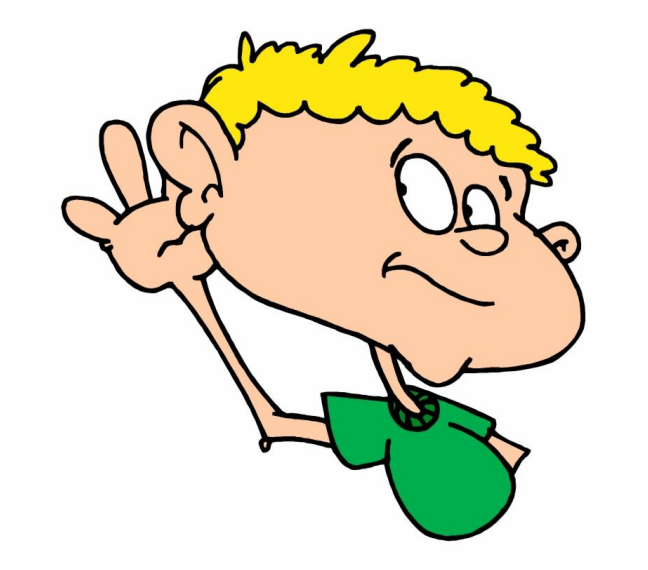


Theorem 3.3.3. A graph *G* is biconnected iff it contains two independent trees rooted at any vertex *v*.

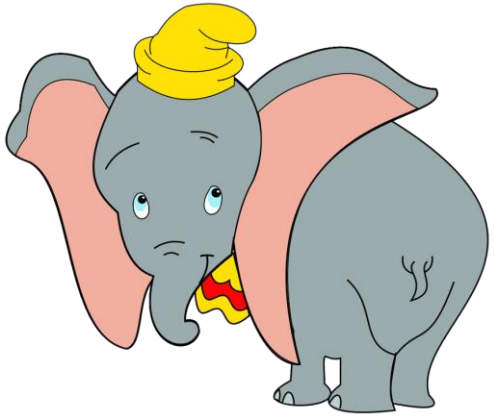
PSN. Prove one direction, i.e., if *G* contains two independent trees rooted at any vertex *v* then the graph *G* is biconnected.

Ear Decomposition

To prove the converse we will need the concept of an open ear decomposition.



Ear Decomposition

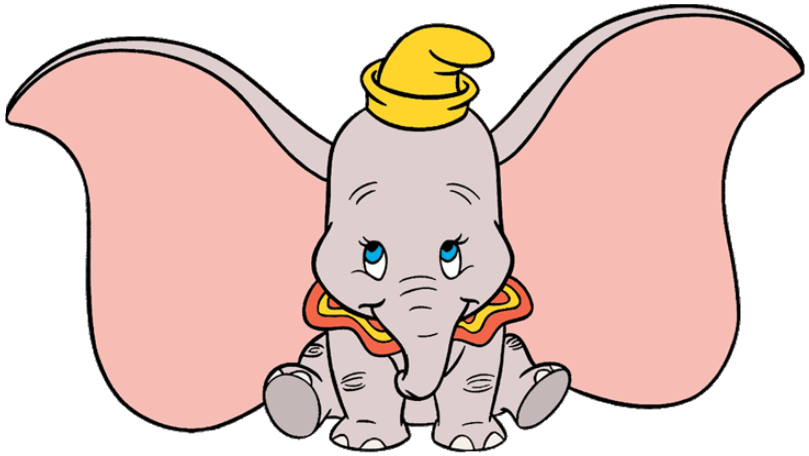


Given any two adjacent vertices s and t, an *ear decomposition* of G is a sequence of paths $P_0, P_1, \ldots, P_p$, with P_0 consisting of the single edge st, and where P_i has only its end vertices in common with the subgraph G_i whose vertex set and edge set are the union of the vertex sets and edges sets, respectively, of the simple paths $P_0, P_1, \ldots, P_{i-1}$, $i = 1, \ldots, p$.

The set of paths partition the edges of G, so that $G = G_p$. The term "ear decomposition" derives from thinking about P_i as an "ear" that is added onto the graph G.

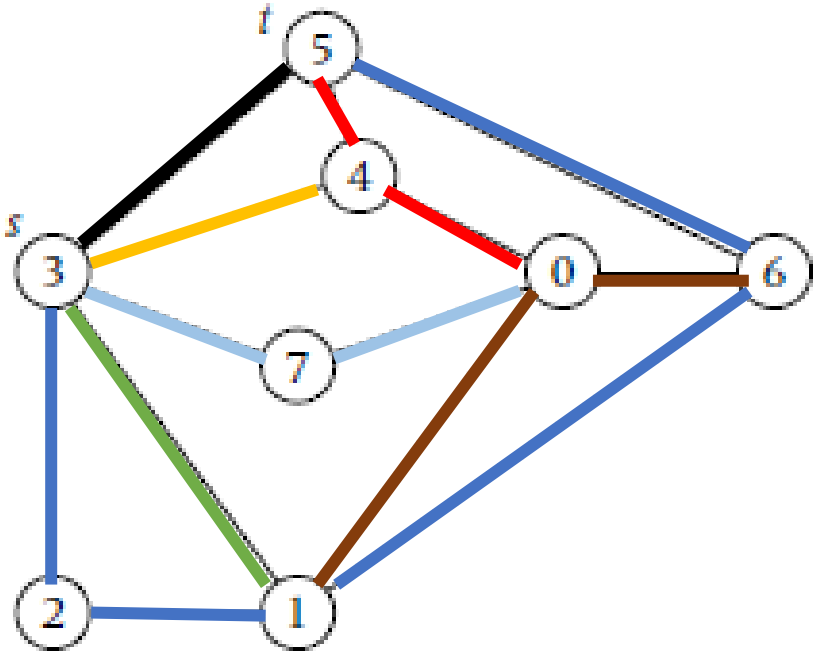
Open Ear Decomposition

If no ear is a closed path, i.e., if the two end vertices of each path (ear) P_i are distinct then we say that the ear decomposition is **open**.



Open Ear Decomposition for an Example Graph

$G =$



P_0: 3,5 P_1: 3,2,1,6,5 P_2: 3,1 P_3: 1,0,6 P_4: 0,4,5 P_5: 3,4 P_6: 3,7,0

Open Ear Decomposition Algorithm

We now discuss an algorithm *OpenEarDecomposition* for computing an open ear decomposition of a biconnected graph *G*, starting with the edge *st*. We will assume that the graph *G* is implemented using adjacency lists.

Algorithm *OpenEarDecomposition*

Stage 1. Perform a depth-first search starting at vertex *s* (and first traversing edge *st*). Similar to the algorithm *ArticulationPoints*, computing the *DFS* numbers *DFSNum*[0:*n* – 1] and the lowest numbers *Lowest*[0:*n* – 1].

Stage 2. Rearrange adjacency lists so that, for each vertex *u*, the first edge *uv* is either an edge of the *DFS* tree such that *Lowest*[*v*] = *Lowest*[*u*], or a non-tree edge such that *DFSNum*[*v*] = *Lowest*[*u*].

Stage 3. Perform a second depth-first search of *G* starting with edge *st* using the rearranged adjacency list resulting from the second stage. The sequence of edges generated by this second depth-first search can be partitioned into simple paths, where the first path is the edge *st* and each subsequent path begins immediately after a non-tree edge has been traversed and ends with a non-tree edge. The set of paths generated in this way determines an open ear decomposition.

Theorem 3.3.1. A graph G is biconnected if, and only if, for any edge st of G there exists an open ear decomposition beginning with ear st.

Complexity Analysis

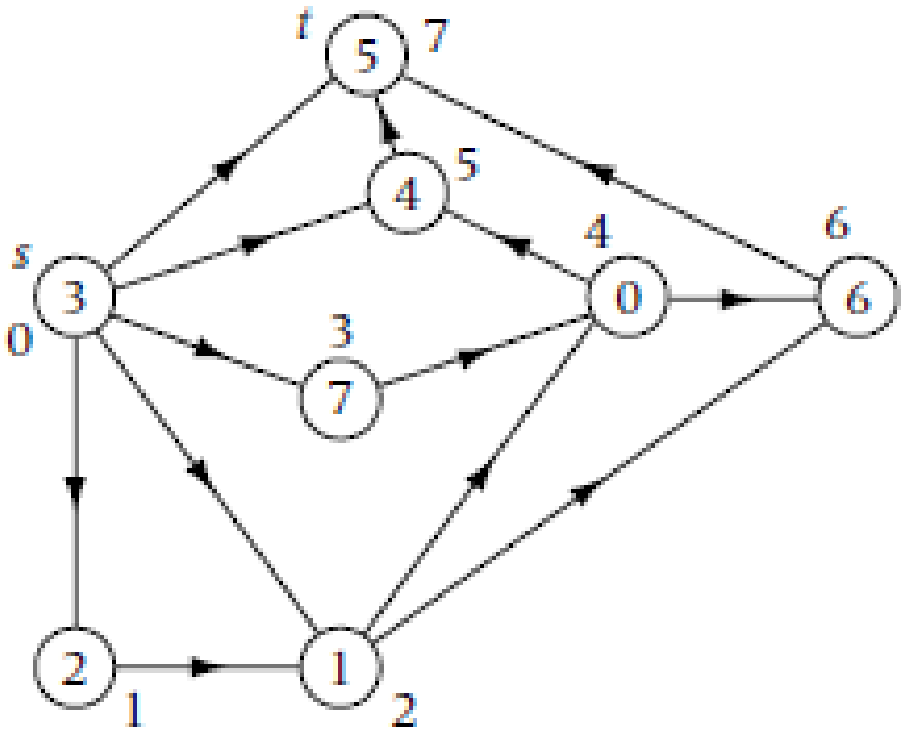
Since depth-first search has $\Theta(n + m)$ worst-case complexity it follows that *OpenEarDecomposition* has worst-case complexity

$$W(n) \in \Theta(n + m)$$

s-t orderings, *s-t* numberings and *s-t* orientations

- An ***s-t* ordering** of a graph *G* is an ordering of the vertices such that *s* is first in the order, *t* is last and every other vertex *v* has at least one vertex *u* in its neighborhood having a smaller *s-t* order and one having a larger *s-t* order.
- A labeling of the vertices with numbers that satisfies an *s-t* ordering, is called an *s-t* numbering.
- An *s-t* orientation can be obtained from an *s-t* ordering by directing each edge *e* from the end vertex of *e* having lower *s-t* order to the end having higher order.
- Conversely, given an *s-t* orientation, we obtain an *s-t* ordering by performing a topological sort of associated acyclic digraph.

An Example *s-t* Numbering and *s-t* Orientation



Computing an *s*-*t* Orientation

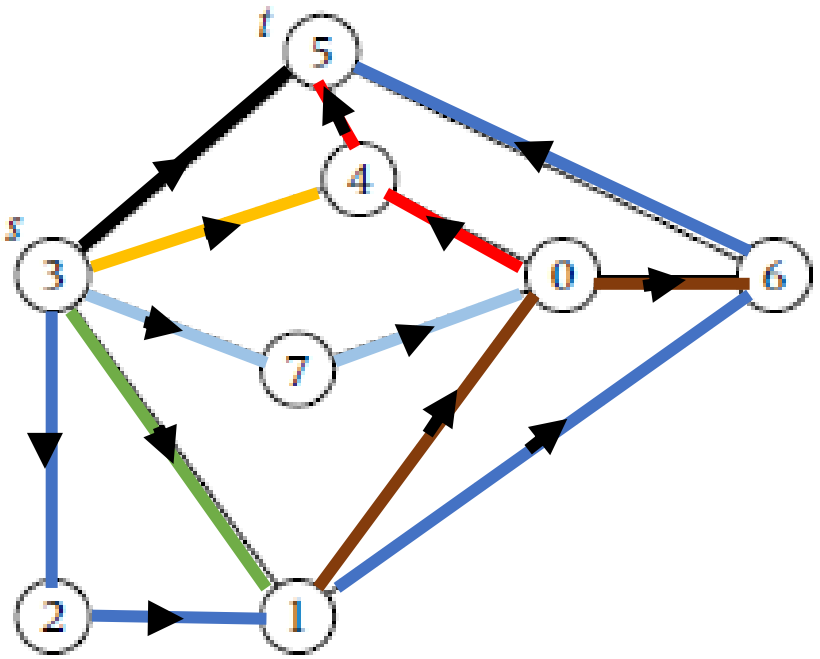
Given an open ear decomposition of a biconnected graph we can compute an *s*-*t* orientation, and thus an *s*-*t* numbering as follows.

Given an ear $P_i = u_1, u_2, \ldots, u_p$, we will orient the edges of P_i such that it forms a directed path from u_1 to u_p, in which case we will say that it is directed out of u_1, or a directed path from u_p to u_1, in which case we will say that it is directed into u_1.

- We begin by orienting the ears that have one end at *s* so that they are all directed out of *s* and push each of them on a stack.
- While the stack is not empty we pop an ear *P*, and scan *P* in the direction it is oriented performing the following operations. For each vertex *u* that is encountered we orient the ears not already oriented that have one end at *u*, so that they are all directed out of *u* and push each of these ears on the stack.

Open Ear Decomposition for an Example Graph

$G =$



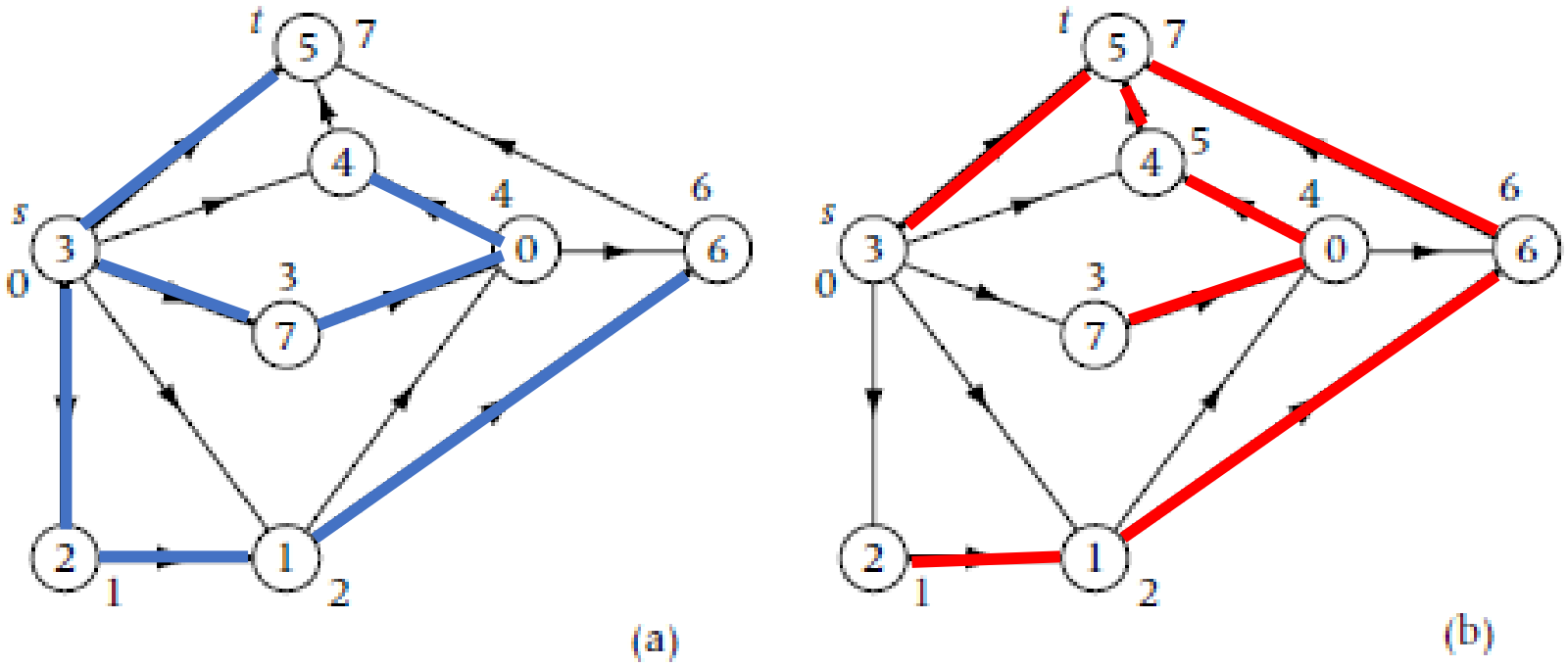
P_0: 3,5 P_1: 3,2,1,6,5 P_2: 3,1 P_3: 1,0,6 P_4: 0,4,5 P_5: 3,4 P_6: 3,7,0

Theorem 3.3.2. A graph *G* is biconnected if and only if for any edge *st* there exists an *s*-*t* orientation (*s*-*t* ordering).

Computing Independent Trees

- Given an *s*-*t* ordering we can immediately generated two independent spanning trees rooted at *s*.
- For each vertex *v* different from *s*, let *lower*(*v*) be any vertex in the neighborhood of *v* that has lower *s*-*t* order and for each vertex different from *t* let *higher*(*v*) be any vertex in the neighborhood of *v* having higher *s*-*t* order, respectively. Construct a tree T_1 such that the parent of *v* is *lower*(*v*) for each vertex different from *s* and *t*, and a tree T_2 such that the parent of *v* is *higher*(*v*) for each vertex different from *s* and *t*. In both trees make *s* the parent of *t*.
- Equivalently, given an *st*-orientation, we can generate two independent trees T_1 and T_2 rooted at *s* by choosing one vertex in the in-neighborhood and out neighborhood, respectively, of each vertex *v* different from *s* and *t* and making it the parent of *v*. Again, in both trees we make *s* the parent of *t*.

Obtaining Independent Trees from *s*-*t* Orientation



Blue tree out of *s* (vertex 0) is obtained by choosing one edge from the in-neighborhood of each vertex different from *s* and the red tree into *t* (vertex 5) is obtained by choosing one edge from the out-neighborhood of each vertex different from *t*.

Independent Trees

Theorem 3.3.3. A graph is biconnected iff it has two independent spanning trees rooted at any vertex *v*.

k-Connectivity of Digraphs

Two vertices *s* and *t* of a digraph are **edge *k*-connected** if there is a directed path from *s* to *t* after the deletion of any *k* – 1 edges.

Two vertices *s* and *t* of a digraph are **vertex *k*-connected** if there is a directed path from *s* to *t* after the deletion of any *k* – 1 vertices.

A digraph is edge *k*-connected iff every pair of vertices is *k*-connected. Similarly a digraph is vertex *k*-connected iff every pair of vertices is *k*-connected

PSN. Give a transformation, showing that an algorithm for determining whether *s* and *t* are edge *k*-connected yields and algorithm for determining whether they are vertex *k*-connected.

k-Connectivity of Graphs

- Edge and vertex *k*-connectivity for graphs are defined the same as for digraphs.
- Recall the that undirected graph is combinatorially equivalent to its associated symmetric digraph obtained by replacing each edge with two directed edges joining the same pair of vertices but in opposite directions.
- Thus, *k*-connectivity for digraphs includes *k*-connectivity for graphs as a special case.

Computing Connectivity using Flow Theory

The ***s*-*t* connectivity** is the maximum *k* such that *s* and *t* are *k*-connected.

We can use flow theory to solve the problem of computing the *s*-*t* connectivity.

Assign each edge a unit capacity. Then the *s*-*t* connectivity is the value of a maximum flow from *s* to *t*.

Menger's Theorem

As a special case of the Max-Flow Min-Cut Theorem we obtain Menger's Theorem by giving each edge a unit capacity.

Theorem 4.2.11 (Menger) Let $D = (V,E)$ be a digraph. Then, for $s, t \in V$, the maximum number of pairwise edge-disjoint paths from s to t equals the minimum size of a cut separating s and t.

The following corollary is the analog of Theorem 4.2.11 for graphs.

Corollary 4.2.12 (Menger) Let $G = (V,E)$ be an (undirected) graph. Then, for $s, t \in V$, the maximum number of pairwise edge-disjoint paths from s to t equals the minimum size of a cut separating s and t.

Menger's Theorem for Vertex Connectivity

Using the result from the PSN the edge version of Menger's Theorem yields the following vertex version. A **vertex cut** separating *s* an *t* is a set of vertices not containing *s* nor *t* whose deletion (together with all incident edges) disconnects *s* and *t*.

Theorem 4.2.11' (Menger) Let $D = (V,E)$ be a digraph. Then, for $s, t \in V$, the maximum number of pairwise **vertex**-disjoint paths from *s* to *t* equals the minimum size of a **vertex** cut separating *s* and *t*.

The following corollary is the analog of Theorem 4.2.11 for graphs.

Corollary 4.2.12' (Menger) Let $G = (V,E)$ be an (undirected) graph. Then, for $s, t \in V$, the maximum number of pairwise **vertex**-disjoint paths from *s* to *t* equals the minimum size of a **vertex** cut separating *s* and *t*.

Conjecture

- Annexstein and Berman generalized the concept of an *s*-*t* number to a *k*-directional embedding and showed that all 3-connected graphs have 3-directional embeddings in the plane.
- As a corollary it follows that a graph is 3-connected iff it contains 3 (pairwise) independent spanning trees.
- It is conjectured that a graph is k-connected iff there exist k independent spanning trees. This has been proven for $k \leq 4$.

EAR-itating Joke

Why shouldn’t you tell a secret to a corn field?

Because its full of ears.

Now that was a corny joke.

And yes, it was rather a-maize-ing!