## 6.1 Directed Acyclic Graphs

**Directed acyclic graphs**, or **DAGs** are acyclic directed graphs where vertices can be ordered in such at way that no vertex has an edge that points to a vertex earlier in the order. This also implies that we can permute the adjacency matrix in a way that it has only zeros on and below the diagonal. This is a *strictly* **upper triangular** matrix. Arranging vertices in such a way is referred to in computer science as **topological sorting**. Likewise, consider if we performed a strongly connected components (SCC) decomposition, contracted each SCC into a single vertex, and then topologically sorted the graph. If we order vertices within each SCC based on that SCC's order in the contracted graph, the resulting adjacency matrix will have the SCCs in "blocks" along the diagonal. This could be considered a **block triangular** matrix.

## 6.2 Trees

A graph with no cycle is **acyclic**. A **forest** is an acyclic graph. A **tree** is a connected undirected acyclic graph. If the underlying graph of a DAG is a tree, then the graph is a **polytree**. A **leaf** is a vertex of degree one. Every tree with at least two vertices has at least two leaves. A **spanning subgraph** of $G$ is a subgraph that contains all vertices. A **spanning tree** is a spanning subgraph that is a tree. All graphs contain at least one spanning tree.

Necessary properties of a tree $T$:

1. $T$ is (minimally) connected
2. $T$ is (maximally) acyclic
3. $T$ has $n - 1$ edges
4. $T$ has a single $u, v$-path $\forall u, v \in V(T)$
5. Any edge in $T$ is a cut edge; any non-leaf vertex is a cut vertex
6. Adding any edge to a tree (without adding vertices) creates a cycle

Which of these properties are also sufficient?

Generally, because any possible tree can be constructed by iteratively adding vertices and edges to the simplest possible tree (single vertex/single edge), we can often use weak/ordinary (instead of strong) induction in proofs on trees.

Prove using weak induction that every tree is bipartite.

**Note:** In prior classes (especially the CS students), you might have used *structural induction* in the context of proofs on trees. The difference with structural induction is that there does not need to be an explicit "countable" variable on which induction is performed, though the recursive idea behind the proof technique is similar. We won't be covering structural induction in this course.

## 6.3 Distances

The **distance** between a $u$ and $v$ in $G$ written as $d(u, v)$ is the length of the shortest $u, v$-path. Remember that the **diameter** of a graph is $max_{u,v \in V(G)} d(u, v)$, or the maximum $d(u, v)$ among all possible $u, v$ pairs. The **eccentricity** of a vertex $\epsilon(u)$ is $max_{v \in V(G)} d(u, v)$, or maximum $u, v$ path from that $u$. The **radius** of a graph is the minimum eccentricity of any vertex, or $min_{v \in V(G)} d(u, v)$. The **center** of a graph is the subgraph induced by the vertices of minimum eccentricity.

## 6.4 Enumeration

**Cayley's Formula** states that with a vertex set of size $n$ there are $n^{n-2}$ possible trees. What this means is that there is $n^{n-2}$ ways to form a list of length $n - 2$ with entries created from a given vertex set. A list for a specific tree is its **Prüfer code**. For a given tree $T$ with some logical ordering of vertex identifiers, we can create its **Prüfer code** by first deleting the lowest value leaf and outputting that leaf's neighbor as a value in our code. We can also use a given vertex set $S$ and a **Prüfer code** to recreate $T$. See the proof in the book that for a set $S \in \mathbb{N}$ of size $n$, there are $n^{n-2}$ trees with vertex set $S$.

Below are the algorithms for creating a **Prüfer code** from $T$ and recreating $T$ from a **Prüfer code**.

---

**procedure** CREATEPRUFER(Tree $T$ with vertex set $S$)
  $a \leftarrow \emptyset$ ▷ Initialize Prüfer code to null
  **for** $i = 1 \ldots (n - 2)$ **do**
    $l \leftarrow$ least remaining leaf in $T$
    $T \leftarrow (T - l)$
    $a_i \leftarrow$ remaining neighbor of $l$ in $T$
  **return** $a$

---

How many different ways can we create a graph given a vertex set of size $n$? **Cayley's Formula** states that with a vertex set of $n$ there are $n^{n-2}$ possible trees. Another way to think about it: the number of possible trees is the number of spanning trees of complete graph. So how do we compute the number of spanning trees of a general graph?

```
procedure RECREATETREE(Prüfer code a created with vertex set S)
    V(T) ← S                                          ▷ Tree has vertex set S
    E(T) ← ∅                                          ▷ Initialize tree edges as empty
    initialize all vertices in S as unmarked
    for i = 1 ... (n − 2) do
        x ← least unmarked vertex in S not in a_{i...(n−2)}
        mark x in S
        E(T) ← (x, a_i)
    x, y ← remaining unmarked vertices in S
    E(T) ← (x, y)
    return T
```

We can use a simple recurrence relation to do so. The number of possible spanning trees in a graph $\tau(G)$ is equal to the sum of the number of spanning trees of the graph with an edge removed $\tau(G - e)$ plus the the number of spanning trees of the graph with an edge contracted $\tau(G \cdot e)$. An **edge contraction** involves combining the endpoints $u, v$ of a given edge $e$ into a single vertex, such that the new vertex has incident edges of all original edges of both $u$ and $v$ except for $e$.

## 6.5 Graceful Labeling

A **graceful labeling** of a graph is a labeling of all $n$ vertices of a graph with unique labels from $0 \dots m$, such that each of the $m$ edges has a unique value computed as the difference between the labels of its endpoints. A graph is **graceful** if it has a graceful labeling.

**Ringel-Kotzig Conjecture**: all trees are graceful. This is unproven, however, certain subsets of of trees have been proven to be. These include paths and **caterpillar graphs**. Caterpillar graphs are trees in which a single path is incident to or contains every edge in the graph. Hypercubes have also been proven to be graceful.