1 Final Chapter 1 Stuff

There are a few things related to material in Chapter 1 that weren't covered in the book. We're going to quickly go over this material.

Directed graphs can also have bipartiteness. A directed graph is bipartite if and only if there are no odd directed cycles. We can use similar reasoning to prove this as we did with undirected graphs.

For an undirected bipartite graph, you can order the vertices such that the adjacency matrix has the following form:

$$\begin{bmatrix} 0_{r,r} & B \\ B^T & 0_{s,s} \end{bmatrix}$$

where B is an $r \times s$ matrix. For a directed bipartite graph, we would have a similar format of:

$$\begin{bmatrix} 0_{r,r} & B_1 \\ B_2 & 0_{s,s} \end{bmatrix}$$

where there is no transpose relation between B_1 and B_2.

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 the adjacency matrix 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**. We can use depth-first search to easily create such an ordering. Note the similarity of the algorithm to our algorithms for connectivity. Also note: Tarjan originally published this algorithm.

---

procedure DFSTopoSort(Graph G)
  for all $v \in V(G)$ **do**
    mark(v)$\leftarrow -1$ ▷ All vertices unmarked initially
  $L \leftarrow \emptyset$ ▷ Sorted list, initially empty
  for all $v \in V(G)$ **do**
    if mark(v)$= -1$ **then**
      visit(v)
  return L

Similarly, we can consider the SCCs of a graph each as single vertices, and create a DAG based on the directed relations between each SCC. We can then order vertices such that all vertices in an SCC are listed together and the SCCs are ordered such that each SCC

```
procedure VISIT(v)
    if mark(v)= 0 then          ▷ Temp mark, already visited on this current traversal
        return ERROR            ▷ Not a DAG, can't topo sort
    if mark(v)= −1 then         ▷ Unmarked
        mark(v)← 0              ▷ Give temp mark
        for all u ∈ N⁺(v) do
            visit(u)
        mark(v)← 1              ▷ Final mark
        L ← v + L               ▷ Add v to head of L
    return
```

only points to SCCs with a later ordering. The structure of the adjacency matrix that results from such an ordering is called **block triangular**.

Aside: we didn't yet explicitly cover the difference between **induced** and **noninduced** subgraphs. For both, we consider a subgraph S of G that contains a set of vertices $V(S) \in V(G)$. We would say that the subgraph is induced if $\forall e(u,v) \in E(G)$ s.t. $u, v \in V(S) \implies e(u,v) \in E(S)$. The subgraph is noninduced if it only contains a subset of the edges $e(u,v) \in E(G)$ s.t. $u, v \in V(S)$.

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.

Properties of a tree T:

1. T is connected
2. T is 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
6. Adding any edge to a tree creates a cycle

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. 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.

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, 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.

Show 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

---

procedure RECREATETREE(Prüfer code a created with vertex set S)
- $V(T) \leftarrow S$ ▷ Tree has vertex set S
- $E(T) \leftarrow \emptyset$ ▷ Initialize tree edges as empty
- initialize all vertices in S as unmarked
- **for** $i = 1 \ldots (n - 2)$ **do**
  - $x \leftarrow$ least unmarked vertex in S not in $a_{i \ldots (n-2)}$
  - mark x in S
  - $E(T) \leftarrow (x, a_i)$
- $x, y \leftarrow$ remaining unmarked vertices in S
- $E(T) \leftarrow (x, y)$
- **return** T
