(III) Properties of BFS & DFS, with applications.

- BFS & DFS naturally induce a tree.

BFS tree

DFS tree

Fact: 1. If \( G \) is a tree \( \Rightarrow \) BFS Tree = DFS Tree

2. BFS Tree = DFS Tree \( \Rightarrow \) \( G \) is a tree.
(1) Def: $G = (V, E)$ is bipartite if we can partition $V$ into set $L$ and $R$, s.t. $L \cup R = V$, $L \cap R = \emptyset$, and $\forall (u, v) \in E$, either $u \in L \& v \in R$, or $u \in R \& v \in L$.

Exmp: 

\[
\begin{array}{c|c|c}
L & & R \\
\hline
\text{Diagram 1} & & \text{Diagram 2}
\end{array}
\]
Algorithm for testing bipartiteness: BFS, DFS

Ex: solve it using DFS

```
L <= \{1, 2, 3, 4, 5\}

1 \rightarrow 2 \rightarrow 3 \rightarrow 4

\text{not bipartite}
```

```
color <- array of size n
color[v] <- null for all v
color[s] <- R
Q <- queue := [s]

While Q is not empty:
    u <- Q.pop()
    for each nbr v of u:
        if color[v] = null:
            color[v] <- reverse of color[u]
            Q.push(u)
        else if color[v] = color[u]:
            return False
        else:
            continue
```
Def: word: a string formed by letters, e.g. "acolfjek"
adjacent words: word A and B are adjacent if they differ in exactly
one letter, e.g. "acdgf" and "addgf".

Prob def

Input: Two words S and T
A list of words $A = \{W_1, W_2, \ldots, W_k\}$

Output: "Yes" if we can change S to T by moving between
adjacent words in $A \cup \{S, T\}$
"No" if otherwise.
Alg for word ladder: DFS

- each vertex correspond to a word
- two vertices are adjacent if the corresponding words are adjacent.

key operation
Given a
check all its nbr.

Question:
How do we check nbrs efficiently?
**Exmp (III) Topological Sort.**

**Prob:** Input: a **directed acyclic graph** $G = (V, E)$

Output: ordering $\pi : V \rightarrow \{1, 2, 3, \ldots, n\}$

such that $\pi(u) < \pi(v)$ if $\exists (u, v)$-path

**Def (DAG) A directed graph without cycles:**

![Diagrams](image.png)
Alg:

\[i = 1\]

While \( V \) is not empty:

1. take \( v \in V \) s.t. \( v \) has no incoming edges
2. remove \( v \) and all its outgoing edges.
3. \( \tau(v) \leftarrow i \) 

\[i \leftarrow i + 1\]

Ex: why step 1 can always be executed when \( V \) is not empty, given DAG \( G \)?
Implementing topological sort.

\[ d[U] \leftarrow \text{\# of incoming edges of } U \text{, for each } U \in V \]

\[ Q \leftarrow \{ v : d[U] = 0 \} \]

\[ i \leftarrow 1 \]

\textbf{while } Q \neq \emptyset:

\[ v \leftarrow Q.pop() \]

\[ \tau(v) \leftarrow i, \ i \leftarrow i + 1 \]

\textbf{for each nbr } U \text{ of } v:\n
\[ d[U] \leftarrow d[U] - 1 \]

\textbf{if } d[U] = 0:

\[ Q.push(U) \]

\textbullet can use the alg to check if the input graph is DAG.