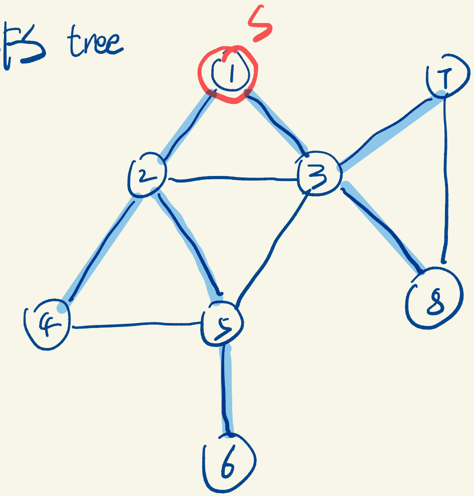
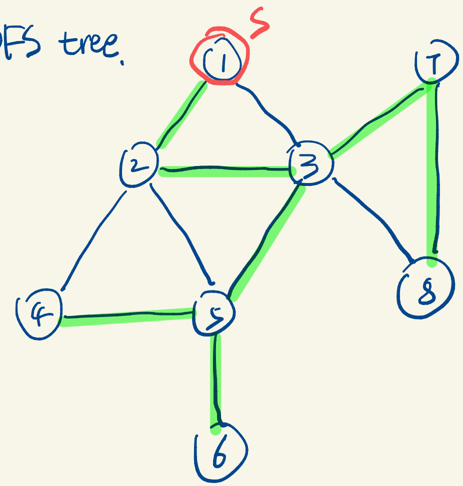
(III) Properties of BFS & DFS, with applications.

▷ BFS & DFS naturally induce a tree.

BFS tree



DFS tree.



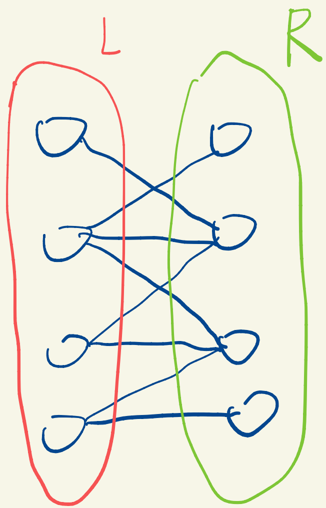
Fact: ① If G is a tree $\Rightarrow$ BFS Tree = DFS Tree

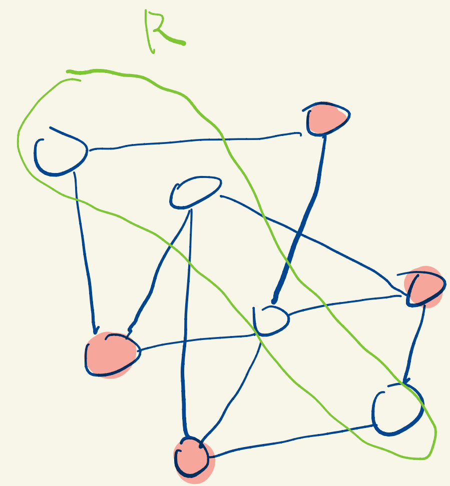
② BFS Tree = DFS Tree $\Rightarrow$ G is a tree.

(I) Exmp 1: Testing Bipartiteness

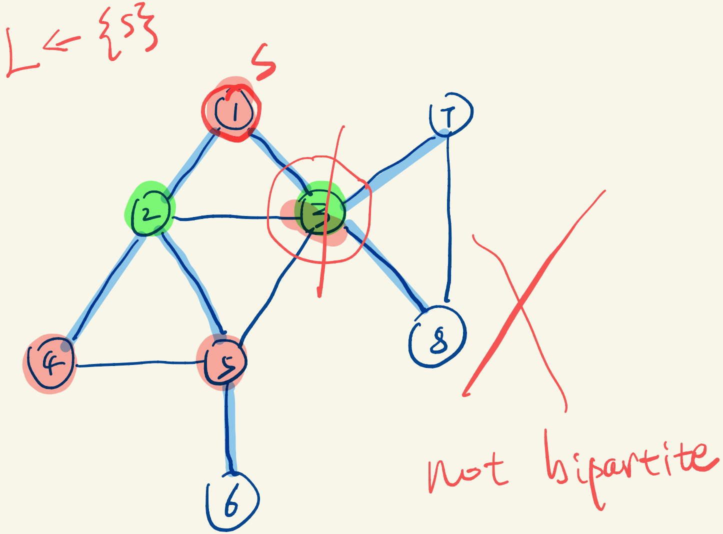
▷ **Def**: $G = (V, E)$ is bipartite if we can partition V into set L and R, s.t. $L \cup R = V$, $L \cap R = \phi$, and $\forall (u, v) \in E$, either $u \in L$ & $v \in R$, or $u \in R$ & $v \in L$.

▷ Exmp:





▷ Alg for testing bipartiteness: BFS, DFS




```
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(v)
        else if color[v] = color[u]:
            return False
        else: continue
```

Ex: solve it using DFS

(II) Exmp 2: Word Ladder

Def: word: a string formed by letters, e.g. "acdfejk"

adjacent words: word A and B are adjacent if they differ in exactly one letter, e.g. "acdgf" and "addgf".

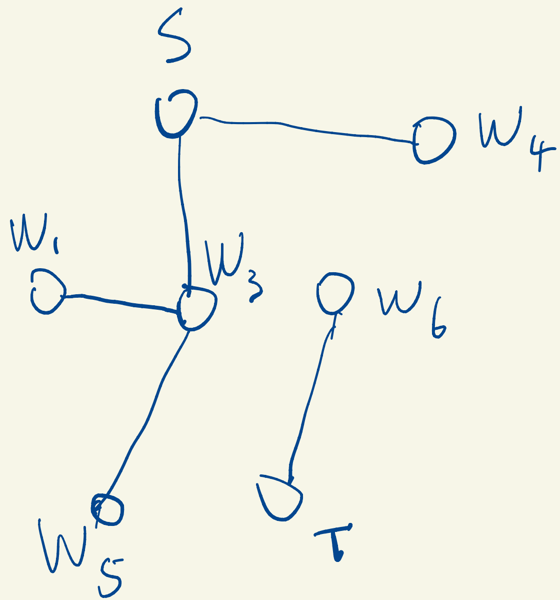
Prob def

Input: Two words S and T

A list of words $\mathcal{A} = [W_1, W_2, \cdots, W_k]$

Output:
- "Yes" if we can change S to T by moving between adjacent words in $\mathcal{A} \cup \{S, T\}$
- "No" if otherwise.

▷ Alg for word ladder: DFS



Key operation

Given u

check all its nbr.

Question:

How do we check nbrs efficiently?

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

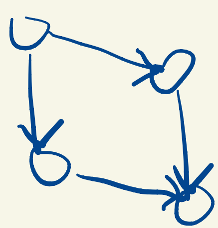
Exmp (III) Topological sort.

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

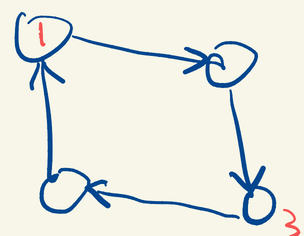
Output: ordering $\pi : V \longmapsto \{1,2,3,\cdots,n\}$

s.t. $\pi(u) < \pi(v)$ if $\exists\,(u,v)$-path

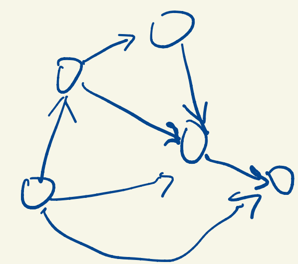
Def (DAG) A directed graph without cycles:



✓



✗



✓

Alg:

$i = 1$

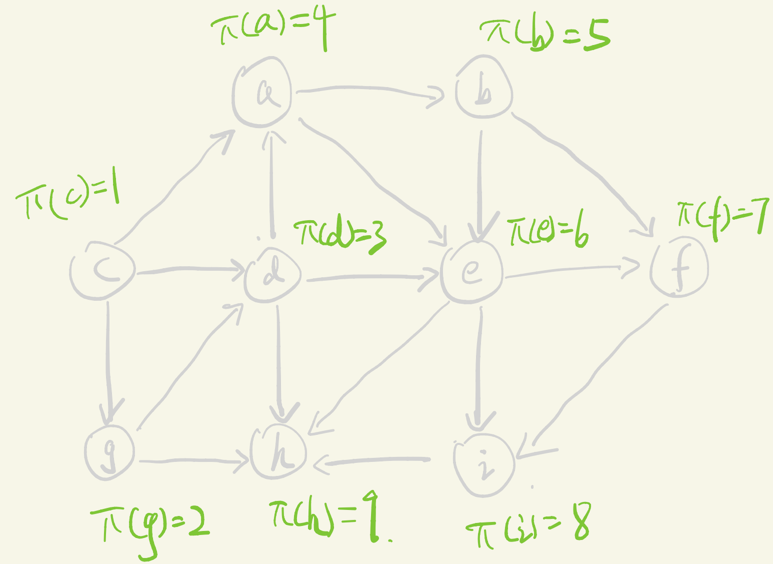
while V is not empty:

① take $v \in V$ s.t. v has no incoming edges

② remove v and all its outgoing edges.

③ $\pi(v) \longleftarrow i$.

$i \longleftarrow i + 1$



Ex: why step ① can always be executed when V is not empty, given DAG G?

▷ Implementing topological sort.

$d[v] \leftarrow$ # of incoming edges of v, for each $v \in V$

$Q \leftarrow [v : d[v] = 0]$

$i \leftarrow 1$.

while $Q \neq \phi$:

  $v \leftarrow Q.\text{pop}()$

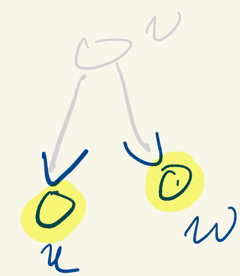
  $\pi(v) \leftarrow i$, $i \leftarrow i+1$

  for each nbr u of v:

    $d[u] \leftarrow d[u] - 1$

    if $d[u] = 0$

      $Q.\text{push}(u)$



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