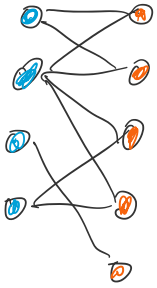
Lec06b-applications-graph-traversal

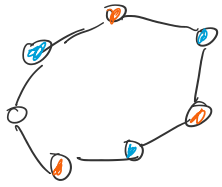
KT 3.4/3.6

Sunday, September 10, 2023 10:10 PM

A graph is **bipartite** if ∃ a two-coloring.
i.e. if you can divide the nodes into two colors such that all edges connect nodes of diff. colors



?



Bipartite graphs can't contain odd cycles

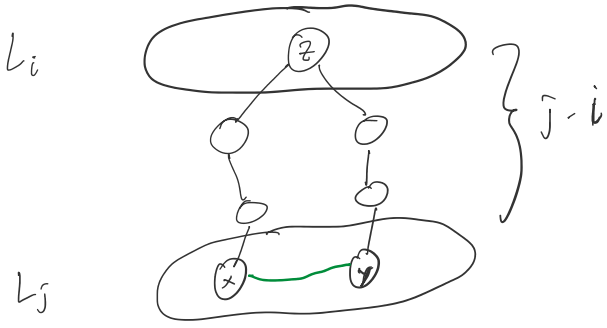
How to test bipartiteness?

Do a BFS from any node. (swapping color each layer)
Check if any edge is monolayer. (Recall edges are always either between adj layers or the same layer)

proof of correctness:

1. If there are no monolevel edges, then every edge is between adj levels, which we can color oppositely.
2. If ∃ edge $(x,y) \in G$ on the same layer L_j, let $z \in L_i$, $i < j$ be the least common ancestor of x, y in the BFS tree.

⇒ $z - x - y - z$ is a cycle of length $2(j-i)+1$, which is odd.
⇒ G is not bipartite.



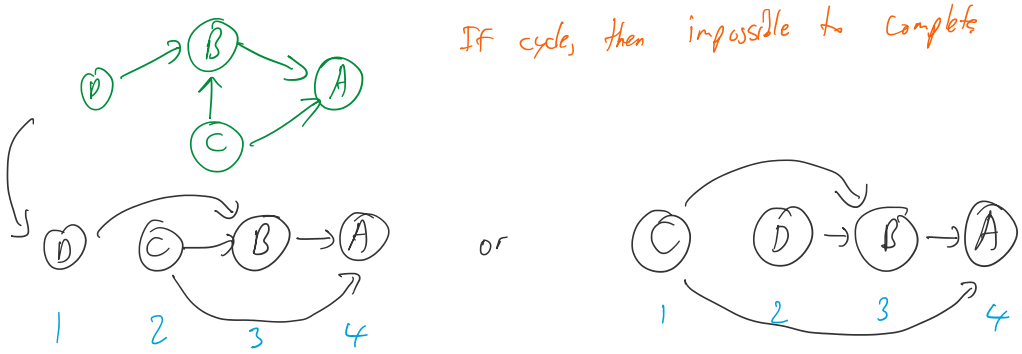
A DAG (directed acyclic graph) is a directed graph that contains no (directed) cycles.
(After leaving a node, you can never return)

Ex. Project dependencies. How do you order jobs so that when a job is started, all its dependencies are done?

Ex. A requires both B and C, but B requires C and D

started, all its dependencies

Task A requires both B and C, but B requires C and D



If cycle, then impossible to complete

Topological sort: Given a DAG $D = (V, E)$ find a bijective mapping f from V to $\{1, \ldots, |V|\}$ s.t. $\forall (u,v) \in E$, $f(u) < f(v)$.

Thm Every DAG has a vertex with no incoming edges (i.e. possible starting pts)

proof. Suppose not
Then keep following edges backward, and in $\leq n+1$ steps will have repeated a node $\Rightarrow$ cycle $\Rightarrow$ contradiction.

Topo sort alg 1:

Let $i = 1$
While $i \leq |V|$
  Find node u w/ no incoming edges.
  Set $f(u) = i$
  Delete u from graph
  i++

Works because after removing u, still a DAG since we didn't add cycles, so we can't get stuck.

Implementation:

Array Incone[w] = # incoming edges for w
List S of nodes w/ no incoming edges

{ decrement Incone[w] for all neighbors
  if Incone[w] = 0, add w to S

Running time:

Initialize Incone[v] by counting edges via DFS. $O(|V| + |E|)$.

{ Loop happens $|V|$ times.
  Each neighbor decrement takes $O(1)$ time, and total times we encounter a neighbor is $|E|$
  $O(|V| + |E|)$ time

Topo sort alg 2: (via DFS finishing times)

Given a DFS, can associate 2 numbers w/ every node:

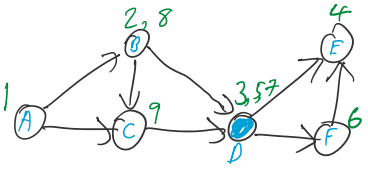
Discovery time: $d[u]$ = time at which u is first visited

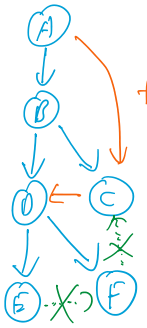
Given a DFS, can associate 2 numbers w/ every node:

discovery time: $d[u]$ = time at which u is first visited

finishing time: $f[u]$ = time at which u and all neighbors are fully explored (last visit)

Clearly $d[u] \leq f[u]$





forward jump

right-left edge

left-right edge can't exist, because would be part of down subtree

backward jump can't exist, because would create cycle.

Algorithm:

Every edge (u,v) in a DAG has $f[v] < f[u]$ (deeper parts of the tree finish first)

Reverse ordering by finishing times gives a topo sort.