Lec07-graph-traversal-applications

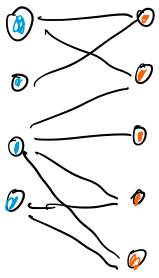
Sunday, September 8, 2024 10:23 AM

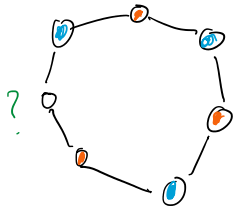
Applications

KT 3.4/3.6

A graph is bipartite if $\exists$ 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 colors at each layer)

Check if any edge is monolayer. (recall edges are always either between adj layers or the same layer)

proof of correctness

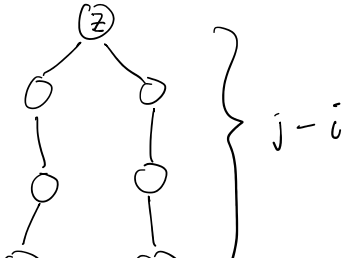
1. If there are no monolayer edges, then every edge is b/t adj layers, which we can color oppositely.

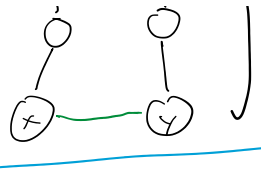
2. If $\exists$ edge $(x,y) \in G$ on the same layer L_j, let $z \in L_i$, $i<j$ be the least common ancestor of x, y in the BFS tree.

   $\Rightarrow$ $z - x - y - z$ is a cycle of length $2(j-1)+1$, odd

   $\Rightarrow$ G is not bipartite.

L_i



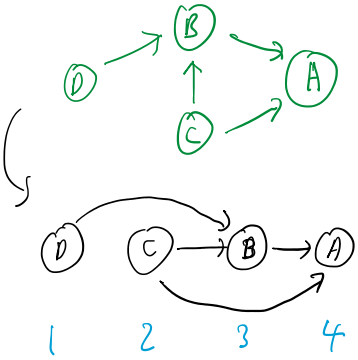


A DAG (directed acyclic graph) is a directed graph that contains no (directed) cycles.
(after leaving a node, you can never return)

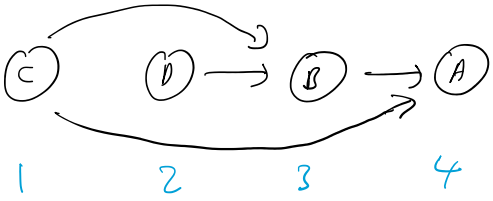
Ex. Project dependencies — How do you order jobs so that when a job is started, all its dependencies are done?

Task A needs B & C. B needs C & D.

If cycle, impossible to complete.



or



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 node with no incoming edges (i.e. possible start pt)

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 $f(u) = i$

Implementation

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

Set $f[u] = i$

Delete u from graph ——— { decrement Income[w] for all neighbors. if Income[w] = 0, add w to S

$i++$

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

Running time:

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

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

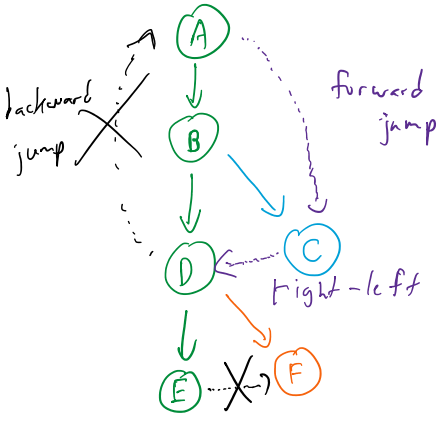
Topo sort alg 2: (via DFS finishing times)

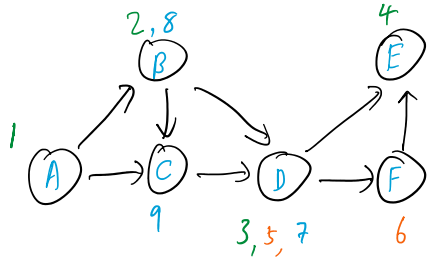
Given a DFS, can associate 2 numbers w/ every node.

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

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

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





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

backward jump can't exist because would

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 tree finish first)

Reverse ordering by finishing times gives a topo sort.