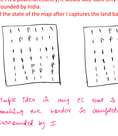
Graphs 3

Monday, August 29th, 2022 10:00 PM

Q. Given a State of a map after a war b/w I & P. Initially it was all I. P attacked and occupied some region.
Since I is a peaceful country, it would take back only that land which is completely surrounded by India.
Find the state of the map after I captures the land back.



⇒ Simple idea is any cc that is not touching are border is completely surrounded by I.

⇒ Iterate over first & last row & cols and any P that is on edge boundary do dfs on it & explore/mark it with W. Now in W will be connected & any other P is replaced by I.

$V = row$ ⇒ $TC = O(r*c)$
$E = 2*r$ $\quad SC = O(max)$

$SC = O(max)$ → for dfs call stack. (can be avoided as its not like necessary)

Q. Given N dependent jobs. Find out the order in which you can program these jobs.

⇒
P is dependent on B

First check which job is not dependent of any other job & start from it.

Only possible on a DAG

⊙ Now ans of the problem is the opposite of Topo sort of the graph.

TOPOLOGICAL SORT (DAG)

Linear ordering of nodes s.t if there is a path from node (2) to node (5) then (2) should come before (5) in the ordering

Topo Sort: 1 2 4 3 5 6
1 2 3 4 5 6

① First create and in-degree array to store no of incoming edges of every node

② Remove node with indegree 0, store it and reduce the indegree of its neighbours by 1.

③ Store all indegree 0 nodes in a container like queue/stack/array.

⇒ ① if $u \to v$ $\quad in[v] += 1$

② Push all nodes with indeg 0 in a Q.

③ While Q not empty
→ take element out : P
→ Put P in ans container
→ Iterate on adj[P] & decrement indegree of neighbours by 1
→ put that 0 neighbour in Q.

• What if there are cycles:
Eventually Q will become empty & size of ANS(.) != N

• Time Complexity
Edge list → E
indegree Array → V

in overall
$TC = O(V+E)$
$SC = O(V)$ → Q / indegree array

• Another Idea
While doing DFS from any random node, when returning from the function, store it in a container of vector!

for i in range (1, N+1)
if not vit[i]:
dfs(i)

dfs(u):
vit[u] = T
for v in adj[u]:
if not vit[v]:
dfs(v)
ans.append(u)

Topo Sort: ans[::-1]

TC: $O(E+V)$ → simple DFS
SC: $O(V)$ → vit, recursion

this is the reversed method for Topo Sort (multi 0 find nodes are required for same reason)

Bipartite Graph

A graph which can be broken down into two disjoint sets U & V s.t no two nodes in U or V have an edge b/w them.

yes

U V

no
⇒

⇒ Given a graph, check if it is bipartite

⇒ Let's consider this tree

yes this is Bipartite. In fact all Trees are bipartite

yes

NO ⇒

⊙ All odd length cycles are → not bipartite
even length cycles → are bipartite

Graph Coloring

⊙ Chromatic No → Min no of colors required to color the graph s.t no 2 adjacent nodes have the same color.

= 3

⊙ Fully connected graphs
→ Every node is connected to every node

Ch NO = V

⊙ For Even length cycle

Ch no. = 2

⊙ Finding Chromatic no
→ NP polynomial time Solution
→ NP Hard problem

⊙ For a graph
⇒ Ch no > 2 → add Edge → Non-Bipartite cycle

⇒ Ch no <= 2 → No add Edge → Bipartite cycles

no matter how many possible ways we try, the colors will always remain same hence non-Bipartite

vit = [F] x N
col = [0] x N
def (v, c):
col[v] = c
vit[v] = c
for u in adj[v]:
if not vit[u]:
dfs(u, 1-c) → 1-c / 1-c
elif col[u] == c:
is_Bipartite = False

TC: $O(V+E)$
SC: $O(V)$

⊙ Since Graph can be disconnected:
for i in range(1, N+1):
if not vit[i]:
dfs(i, 0)