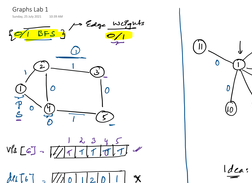
Graphs Lab 1

Sunday, 26 July 2020 10:58 AM

{ BFS }

→ Edge weights 0/1



vis[5] = [T T T T T]

dist[5] = [0 1 2 0 1]

Note: BFS work for unweighted

we consider weight of every edge 1

{ → BFS on a graph of all edge weights are same }

dist[12] = [0 0 0 0 2 1 1 1 1 0 0 0]

vis[12] = [T T T T T T T T T T T T]

Idea: After popping a node from front say u

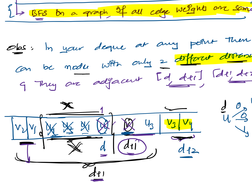
1) All unvisited nodes, adjacent to u with edge weight = 0 we push them node at front d[v] = d[u]

2) All unvisited node, adjacent to u with edge weight = 1) we push them node at back d[v] = d[u]+1

While Deque

Repeat this process until is empty

obs: In your deque at any point there can be nodes with only 2 different distances & they are adjacent [d1, d+1] [d+1, d+2] . . .

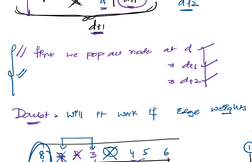


// first we pop all node at d

→ d+1

→ d+2

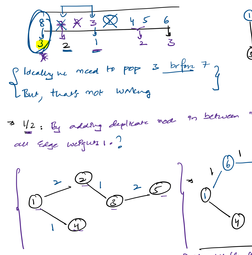
Doubt: will it work if edge weights are {1 & 2}



{ Actually we need to pop 3 before 7 }

{ But, this not working }

→ 1/2: By adding dummy node in between make all edge weights 1.?



1) Now we can apply BFS:

Given N & M Edges

At max how many dummy node we will add:

At most M-E Edges: weight 2: {M-E dummy nodes}

Total Node: N+M-E

Total Edges: M+M-E

{ Simply those of we apply BFS }

→ 1/100

Insert 99 dummy nodes

→ Challenge: Not really practicable, because, of extra space

B2. Reversing Edges

Given a directed graph A nodes & M edges. Find min no. of edges needs to be reversed to reach Node 1 from Node 1

Ex: A = 5

B = { [1, 2] [4, 2] [3, 2] [1, 3] [4, 5] }

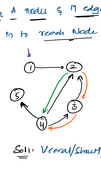
→ Reversing 2 Edges: 2→3, 4→1

or

→ Reverse 1 Edge: 2→4

Sol: Floyd/Dijkstra/Topsort/Approx/Angels/Kadane/Extra.

BFS: Consider all edge undirected & apply BFS.



ans: 2

1 — 2 — 4 — 5 : 2

1 → 2 → 3 → 6 → 4 → 5 : 1

Ex: A = 7

B = { [1, 2] [3, 2] [3, 4] [3, 4] [6, 2] [5, 4] [3, 5] }

Idea: Since we don't know what edge to reverse, take all rev edges as well

Steps:

1) Add new edges with 1

2) Add link

3) Apply 0/1 BFS

TC: O(N+M) SC: O(N+M)

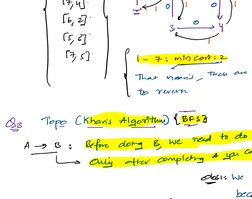
{ I → 7 : min cost = 2 }

{ That means, there are min edge I need to reverse }

Obs: Topo (Khan's Algorithm) { BFS }

A → B : Before doing B we need to do A

→ Only after completing A we can do B.



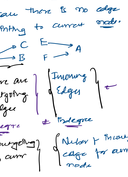
If u → v: In[v]++,

In[] = [X X X X X X X X X X X]

0 0 0 0 0 0 0 0 0 0 0

Q: [2 7 10 8 1 3 4 5 7 6 9]

obs: We can directly execute 2, 7, 10 because there is no edge pointing to current node.



→ { There are outgoing edges } { Incoming edges }

analogy:

{ Number of outgoing edge from curr node } { Number of incoming edge for current node }

obs: We can execute the nodes with 0 indegree

With Edge Informat

1) Construct Adj List

2) Construct Indeg[]

3) All Nodes having 0 indegree push them queue

Int C = 0

While (q.size() > 0) {

int u = q.front();

q.pop();

ans[C] = u; C++;

// pop all outgoing edge of u & reduce their node indegree. { Adj list of u }

for (int v : g[u]) {

Indeg[v]--;

if (Indeg[v] == 0) { // That means all dependency of v are already cleared, we can execute v

q.push(v)

}

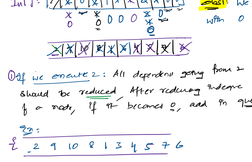
}

}

if (C < N) { // we have cycle }

TC: O(N+M) SC: O(N)

Q2: In[] = [X 0 X 1 X 2]



obs: If no of nodes pointed to 1 or more, that means, there is a cycle.

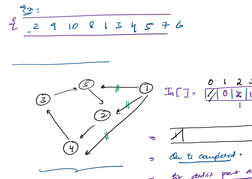
→ Q2 is completed.

→ We didn't push any nodes

{ That means there is cycle? }

obs1: If there is no node with 0 indegree that means {all nodes} belong to a cycle?

[Move to field]



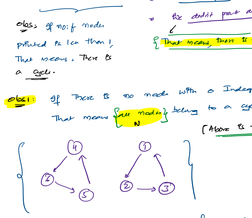
obs2: If all N node together form a cycle, then indeg should Connected Components = 1

Interrupted topological Sort

1) Out of all nodes with indegree 0, we will choose the smaller node first

In: [X X 1 X X 0 X 1]

0 0 0 0 0



Queue

{ 8, 4, 1 }

{ Note: When every the we pick node dequeue }

{ Indegree of a node becomes 0 when its parent is kept }

DFS on undirected graph, want have cycle Edge

→ { Forward & Cross Edge } { Not on graph & check it out back }

→ Backward Edge

→ Tree Edge