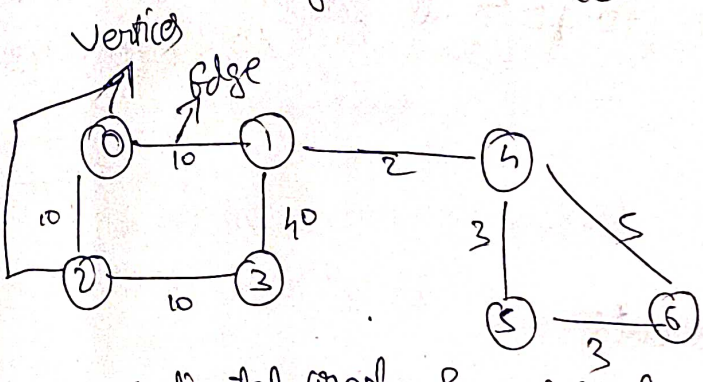
Apr-24

Graphs

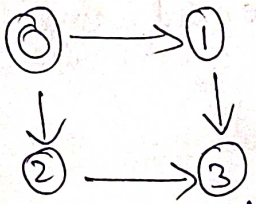
All trees are directed and acyclic, graphs and connected

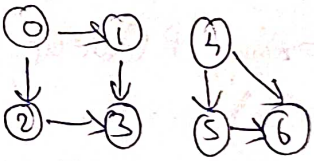
Edges — Vertices



Undirected graph & weighted

Types in graph
* Directed Graph
* Undirected Graph
* Weighted graph
* Unweighted graph





Disconnected graph

Directed graph & Unweighted

Variation in questions
* path
* group
* cyclic
* MST < Prim's Algo, Kruskal's Algo
* matrix (Top, bottom, left, right)

10^3 vertices $\Rightarrow 10^6$ space $\times$ (Note: Max. space you can use is 10^5)

Space is wasted

Construction in graphs
* Adjacency Matrix
* Adjacency List
* Hashmap

Construction using Adjacency list

V_1 (src) — V_2 (nbr) @wt

V_2 (src) — V_1 (nbr) @wt

Has Path

* DFS Approach, visited
* No need to change visited of src to false, (no need to backtrack) because from src → DFS will allow you to travel completely. (dest. connected or not connected) so no need to make false and try another path to check. Check video for clarity.

V → Vertices
E → Edges

TC $O(VE)$

[$\because E \to V^2$]

In terms of V

TC $O(V^3)$

Printpath

* Make a DFS travel
* Have to backtrack (to go to all paths).

Multisolver

* Travel DFS
* ~~Find~~ Find requirements
* For Kth largest, used floor logic 'k' times to find kth
  ↳ (After learning P. Queue, to implement using P.Q for Kth)

Get Connected components

* Have a visited arr.
* Make a DFS at every vertex from visited arr.
  ↳ ~~Nav~~ Travel and collect components only if vertex is unvisited.
  ↳ Else ignore and move forward.

Is Graph Connected

App1: Get all connected components. If size of ~~called~~ all collected connected comps == 1, then connected, else not connected

App2: Make a DFS from a single vertex.
Maintain a visited array (Don't backtrack here)
Loop on visited, if any vertex is not visited, then not connected, else connected.

(You can get ~~the~~ number of connected components by BFS as well, just use a counter.)

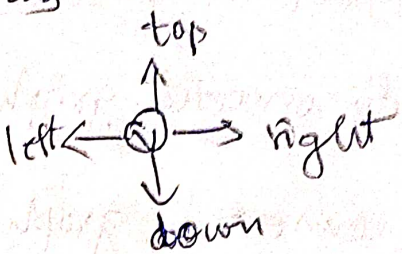
Number of Islands

* Make dfs from every point moving to its neighbours (top, left, down, right).

Techniques: Application of connected components

* Maintain visited as well while traveling.

In matrix, every elements represents a matrix having neighbours at top, left, down, right.



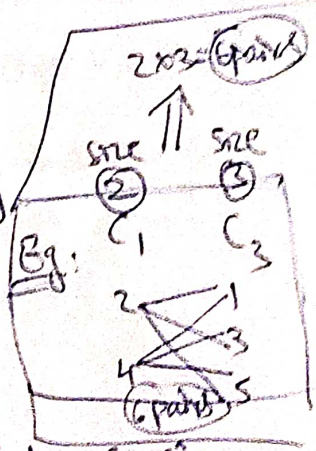
> In interviews, sometimes even diagonal can be said as neighbours. Just be prepared to tackle it.

Perfect friends

* Get all connected components first

connComps = [C_1, C_2, C_3, C_4]

* Then find pairs from it by pairing (calculate it with each).



eg:

* No of pair two connComps can make,

pairs = C_1.size() * C_2.size

* Like this map every components with each other.

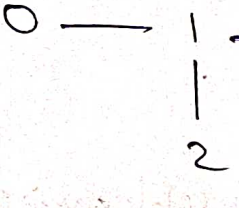
Hamiltonian Path and Cycle : (Use path so far logic)

Hamiltonian path : when you can visit all vertex of graph only once.

Hamiltonian cycle : when first vertex and the last vertex has a direct edge.

Hamiltonian path & cycle can be applied only on connected graphs. There is no point of finding HPC on disconnected graphs.

Not all connected graph have hamiltonian path. eg: ⇒



⇒ No hamiltonian path.

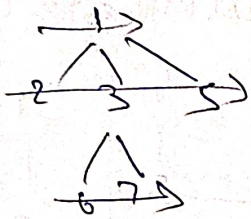
Generic tree is a special type of graph.

* Connected
* Directed (parent to child)
* No cycle

Trees

* Pre, post, In (recursion)

* Level order



a) normal level order:

1 2 3 5 6 7

b) level order linewise

1

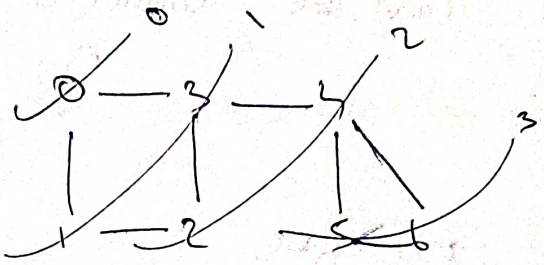
2 3 5

6 7

Graphs

* Depth first traversal (DFS) (recursion)

* Breadth first traversal (BFS)



a) normal level order: 0 1 3 2 4 5 6

b) level order linewise:

0 (0 edges far) from src

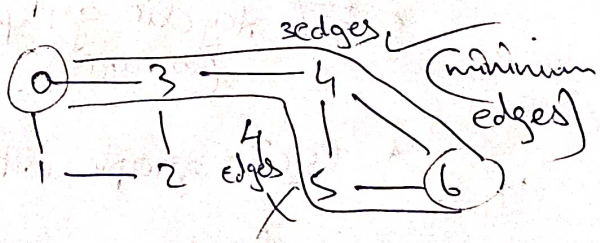
1 3 (1 edges far) "

2 4 (2 edges far) "

5 6 (3 edges far) "

(∵ This BFS takes vertex of minimum edges first)

eg:



BFS of graph: Use queue

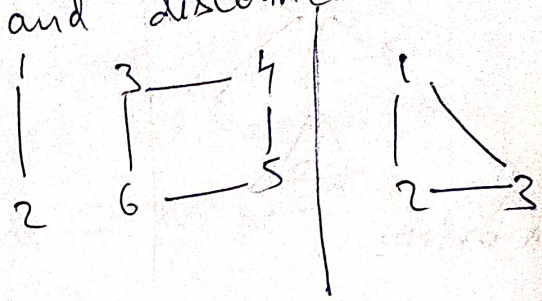
* Remove
* Mark
* ~~Pr~~ work
* add nbr (unvisited)

If visiting already visited node after removal, then just ignore.

Is Graph cyclic

If there is any cycle (even one), then it is cyclic. If any component of graph has cycle, then it is cyclic. Here graph can be connected and disconnected components

eg:



* Use BFS

* Using DFS ⇒ for every visited vertex 'v' (Refer GFG) if there is an adjacent 'u' such that u is already visited and u is not parent of v, then there is a cycle in graph

---

Is Graph Bipartite

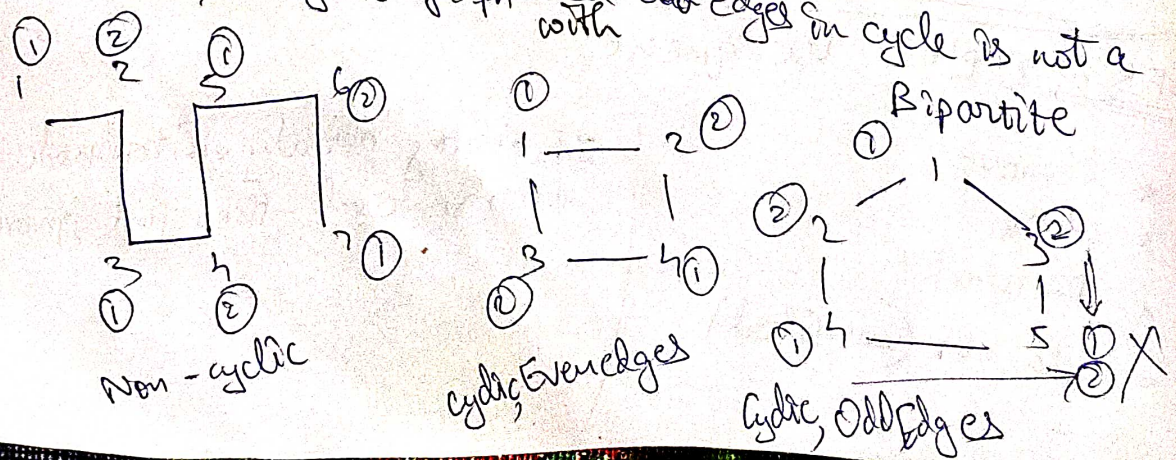
(Here graph can be connected or disconnected as well)

* All vertices edge must be across the sets.

* Acyclic graph is a Bipartite graph. (non-cyclic)

* Cyclic graph with even edges (in cycle) is a Bipartite

* Cyclic graph with odd edges in cycle is not a Bipartite



* All components has to be bipartite, for a graph to be bipartite

* Apply BFS

Spread of Infection

* Apply BFS
* Doesn't matter If we have disconnected components (because anyways we won't be reaching that vertex from our source)

compareTo( )

+ve → this > other
-ve → this < other
0 → this == other

(If +ve value, do some work)

~~Dijkstra's~~ Djikstra's Algo (Shortest Path in weights)

* Single source to all dest.
* Work only for positive weights
* Use Priority Queue (BFS)

Minimum Spanning Tree (Prim's Algo)

(~~Minimum wi~~ Minimum wire to connect all PCs)

i) Acyclic, connected graph is a tree
ii) to travel all the vertices (span)
iii) Use Priority Queue (BFS) (all vertices should be included).

Here we try to break cycle, & we break cycle by removing the heaviest edge out.

Djikstra's → Competition in path
Prim's (MST) → Competition in edges

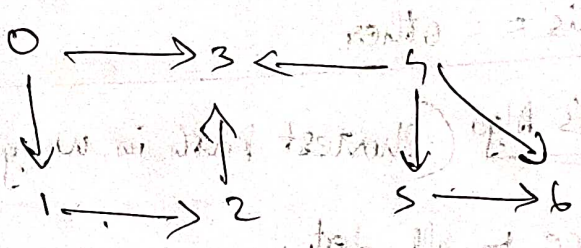
Prim's	Dijkstra's
* Src doesn't matter, you will have the same MST from any source	* Src matters to travel path
→ Edges ~~of~~	* Path ~~wsf~~
* wt	* wsf

Topological sort: DAG (Directed ~~Acyclic~~ graph) (Can be applied only on DAG)

Aeyclic

Valid T.S

4 5 6 0 1 2 3



Topological sort: A permutation of all vertices such that for ~~all~~ all edges of type u-v, u should always come before v.

order of work or compilation is opposite of topological sort i.e., T.S u→v (u is dependent on v)

In order of compilation, since u is dependent on v, so v must be complete ~~first~~ before u.

T.S	OC
4	3
5	2
6	1
0	0
1	6
2	5
3	4

* Use stack & DFS travel

(A graph can have more than one topological order. It's just that the T.S condition has to be satisfied)

* Topological sort → Push in stack (post order)
* Order of compilation → Print directly (post order)

Iterative Depth First Search:

Replace BFS Queue with Stack & follow BFS Algo pattern.

↳ As a result you will travel in DFS (Reverse Euler order)

DFS travel Order doesn't matter in graph. The point is you have to travel unlike trees

(*) This approach can be used in trees as well (output in reverse Euler order)