Lakshay Sharma
AI and DS
10

1) Breadth first Search (BFS)

(i) BFS uses queue data structure for finding the shortest path.

(ii) BFS is better when target is closer to source

(iii) As BFS consider all neighbour so it is not suitable for decision tree used in puzzle games.

(iv) BFS is slower than DFS

(v) Time complexity → O(V+E)

Depth first search (DFS)

(i) DFS use stack data structure

(ii) DFS is better when target is far from source.

(iii) DFS is more suitable for decision tree, As with one decision, we need to traverse further to augment the decision.

(iv) DFS is faster than BFS

(v) Time complexity → O(V+E)

Application of DFS:-

(i) Used to create minimum spanning tree for all pair shortest path tree.

(ii) We can detect cycles in a graph.

(iii) Used to find between two given vertices u and v.

(iv) ~~Used~~ Topological sorting can be done using DFS.

(v) Used to find strongly connected components of graph.

Application of BFS:-

(i) In ~~file~~ - to - peer network like bit - torrent, BFS is used to find all neighbour nodes.

(ii) Using GPS navigation system BFS is used to find neighbouring places.

(iii) In network when we want to broadcast some packets, we use BFS.

(iv) BFS is used in Ford - Fulkerson Algo. to find max flow in a network.

Q2> In BFS, we use queue data structure because we dont know the size of the frontier in advance, queue is more memory efficient. Also queue data structures are considered inhonestly "fair". The FIFO concept that underlines a queue will ensure that those things that were disovered first will be explored first.

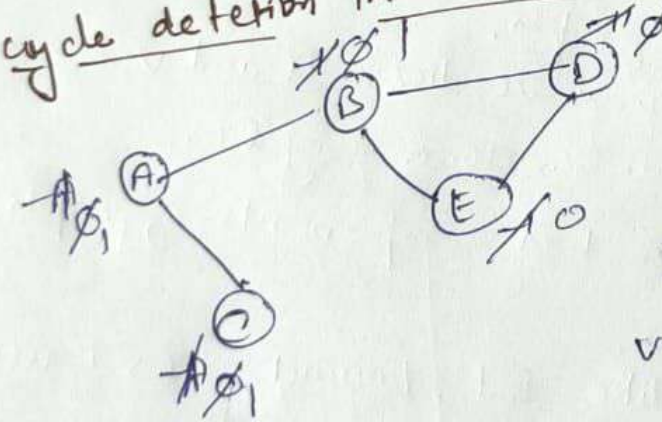
In DFS we uses stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

Q3> Sparse graphs :- sparse is a graph in which the no. of edges is close to minimum no. of edges. It can be disconnected graph.

Dense graph :- Dense graph is a graph in which the no. of edges is close to the maximal no. of edges.

- Adjacency lists are preferred for sparse graphs.
- Adjancy matrix for dense graph.

Q4> cycle detetion in Undirected Graph (BFS) :-



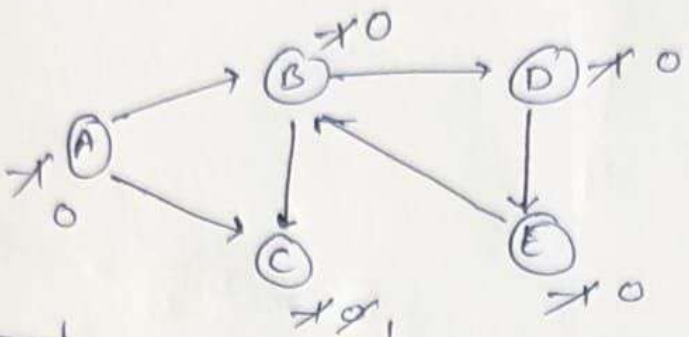
→ 1 = unvisited
0 = into the queue
1 = traversed.

queue = | A | B | C | D | E |

visited set = | A | B | C | D |

- when D checks its adjacement vertices it find E with 0.
- If any vertex finds the adjacemant vertex with flag 0 then it contains cycle.

Cycle detection in directed graph (DFS) ③



-1 = unvisited
0 = visited & in stack
1 = visited & popped out from stack.

Visited set :- A B C D E

⇒ B → D → E → B

parent map

vertex	Parent
A	—
B	A
C	B
D	B
E	D

- there E finds B (adjacent vertex of E) with 0 ⇒ it contains a cycle.

5) Disjointed data structure :-

- The disjoint set data structure is also known as union-find data structure & merge-find set. It is a data structure that contains a collections of disjoint or non-overlapping sets.
- The disjoint set means that when the set is partitioned the disjoint subsets, various operations can be performed on it.
- In the case, we can add new sets we can merge the sets, we can also find the representative numbers of a set. It also allows to find out whether the two elements are in same set or not efficiently.

• Operations on disjoint set :-

① Union

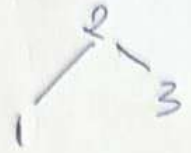
(a) If S1 and S2 are two disjoint sets, their union S1 U S2 is a set of all elements x, such that x is in either S1 or S2

(b) As the set should be disjointed S1 U S2 replaces S1 and S2 which no longer exists.

(c) Union is acheived by simply making one of the trees as a subtree of other i.e. to set parent field of one of the roots of the trees of other root.

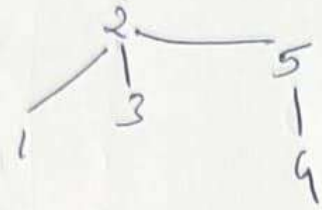
Ext

S1



S2

S1 U S2



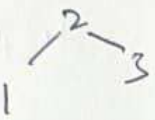
merge the set containy x and containy y into one.

② Find:-

— given an element x, to find the set containy it

Ext

S1



S2

find (3) → S1

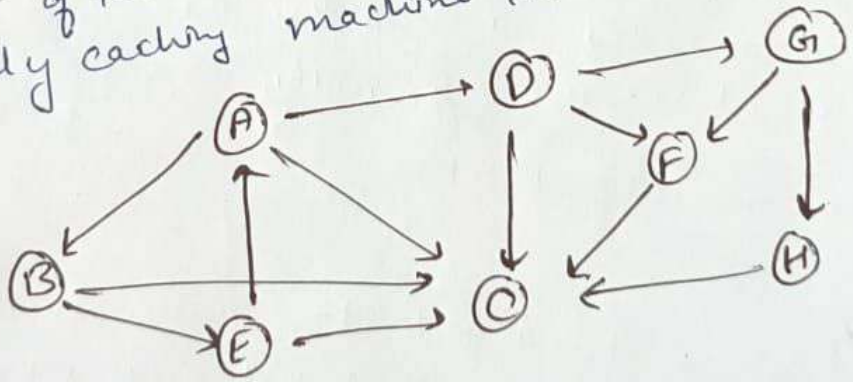
find (5) → S2

returns the which set x belongs.

③ Path comparison (modification to find())

— It speeds up the data structure by comparing the height of the trees. It can be achieved by inserting a small caching machine into a find operation.

6)



BFS :-

Queue	Action	Visited Node
G	Insert G	G
~~G~~ DFH	Remove G Insert D, FH	GDFH
~~D~~FHC	Remove D Insert C	GDFHC
~~F~~HC	Remove F	GDFHC
~~H~~C	Remove H	GDFHC
~~C~~E	Remove C Insert E	GDFHCE
~~E~~A	Remove E Insert A	GDFHCEA
~~A~~B	Remove A Insert B	GDFHCEAB
~~B~~	Remove B	GDFCEAB

Path :- G → D → F → C → E → A → B

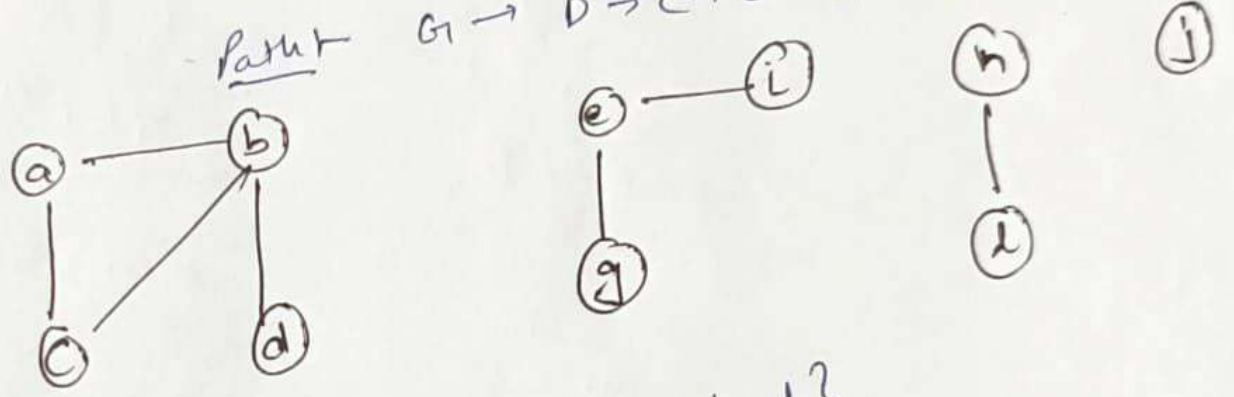
DFS :-

Stack	Action	Node visited
G	Push G	G
GD	Push D	GD
GDC	Push C	GDC
GDCE	Push E	GDCE
GDCEA	Push A	GDCEA
GDCEAB	Push B	GDCEAB
GDCEA~~B~~	PoP B	GDCAB
GDCE~~A~~	PoP A	

GDC~~H~~	Pop E	GDCEAB
GD~~C~~	POP C	GDCEAB
GDF	Push F	GDCEABF
GD~~F~~	POP F	GDCEABF
GD~~D~~	POP D	GDCEABF
G~~D~~	PUSH H	GDCEABFH
GH	Pop H	GDCEABFH
G~~H~~	Pop G	GDCEABFH
G		

Path: G → D → C → E → A → B → F → H.

7)



$V = \{a, b, c, d, e, g, h, i, j, l\}$

$E = \{(a,b), (a,c), (b,c), (b,d), (e,i), (e,g), (h,i), (j)\}$

	{a}	{b}	{c}	{d}	{e}	{g}	{h}	{i}	{j}	{l}
(a,b)	{a,b}	{c}	{d}	{e}	{g}	{h}	{l}	{j}	{e}	
(a,c)	{a,b,c}	{d}	{e}	{g}	{h}	{i}	{j}	{l}		
(b,c)	{a,b,c}	{d}	{e}	{g}	{h}	{i}	{j}	{l}		
(b,d)	{a,b,c}	{d}	{e}	{g}	{h}	{i}	{j}	{l}		
(e,i)	{a,b,h,e,d}	{e,i}	{g}	{h}	{j}	{l}				
(e,g)	{a,b,c,d}	{e,i,g}	{h,l}	{j}						
(h,i)	{a,b,c,d}	{e,i,g}	{h,l}	{j}						
(j)										

We have, {a,b,c,d} {e,i,g} {h,l} {j}