

Tutorial 5

Q.1) what's difference between DFS & BFS. write applications of both algorithm

A.1)

DFS

BFS

- Uses stack to find shortest path
- Stands for Depth first search
- Suitable for searching webtree closer to given source
- Recursive algorithm-like backtracking
- Requires less memory
- ^{not} considers all neighbours first i.e. and then explores all path through this decision eg:- acyclic graph, topological order
- Use Queue to find shortest path.
- stands for Breadth first search
- suitable where solutions are closer to source
- NO concept of backtracking
- Requires more memory.
- considers all neighbours first i.e. not suitable for decision making.
eg:- Bipartite graph, shortest path etc.

Q.2) which Data structure is used to implement DFS & BFS & why?

A.2) DFS Algorithm traverses a graph in depthwise motion & uses stack to remember to get next vertex to search, when a dead end occurs in any iteration. So basically in order to reach to the deepest node first stack - used.

- BFS uses queue to avoid recursion. Queue keeps track of locations remaining to be searched as opposed to needing a potentially large amount of processor stack space for recursive solution.

Q.3) what do you mean by sparse & dense graphs? which representation of graph is better for both.

A.3) Dense Graph

Sparse Graph

- A graph in which number of edges is close to maximal no. of edges
- Graphs are densely connected.
- $G_1 = (V_1, E)$ in which $|E| = O(|V|^2)$
- If a graph is dense we should store it as adjacency matrix.
- A graph in which no. of edges is minimal to no. of edges.
- Graphs are sparsely connected (e.g. Trees).
- $G_1 = (V_1, E)$ in which $|E| = O(|V|)$.
- If graph - sparse - store it in a list of edges.

Q.4) How to detect cyclic graph using BFS & DFS?

A.4) using BFS:-

- i) compute indegree (no. of incoming edges) for each vertex present in graph & initialize count of visited nodes as 0.
- ii) Pick all vertices with in-degree as 0's, add to queue (enqueue operation).
- iii) Remove vertex from queue.
 - Increment count of visited nodes by 1

- Decrease in-degree by 1 for all neighbouring node.
- If in-degree of a neighbouring node is reduced to zero, add to queue.

4.) Repeat step 3 until queue is empty.

Q.4)

Using DFS:-

- 1.) Create graph using given number of edges & vertices.
- 2.) Create recursive f() that have current index or vertices, visited array & parent node.
- 3.) Mark visited node as visited.
- 4.) find unvisited vertices that are adjacent to current node.
- 5.) If adjacent node not parent - already visited - return true.

(Q.5) What do you mean by disjoint-set data structure?

Explain 3 operations along with examples, which can be performed on disjoint sets.

A.5.) Disjoint-set data structure:- also called union find data structure or merge-find set is data structure that stores a collection of disjoint (non-overlapping) sets. Three operations performed by Disjoint-set data structure:-

- 1.) Making new set containing a new element.
- 2.) Finding representative of set containing given element.
- 3.) Merging two sets.

1.) Making New Set:- Make set operation adds a new element into a new set containing only the new element, &

new set is added to data structure.

(ii) finding Set Representatives

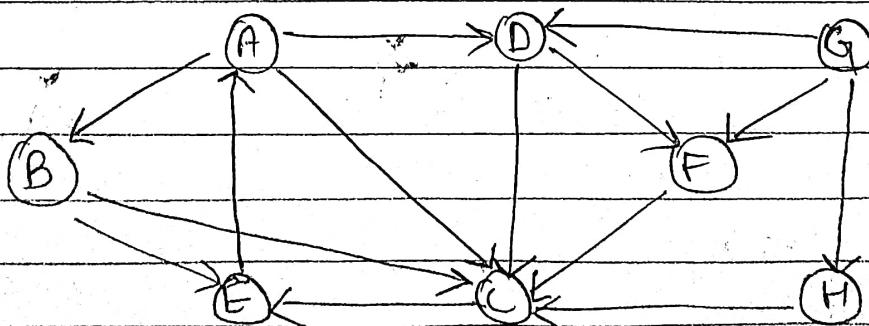
find operation follows chain of parent pointers from a specified query node x until it reaches a root element. Root element represents set to which x belongs and may be x itself.

(iii) Merging Two Sets:-

Operation union(x, y) replaces set containing x & set containing y with their union.

(Q.6) RUN DFS & BFS on graph shown on R.H.S.

A.6.)



B.6.)

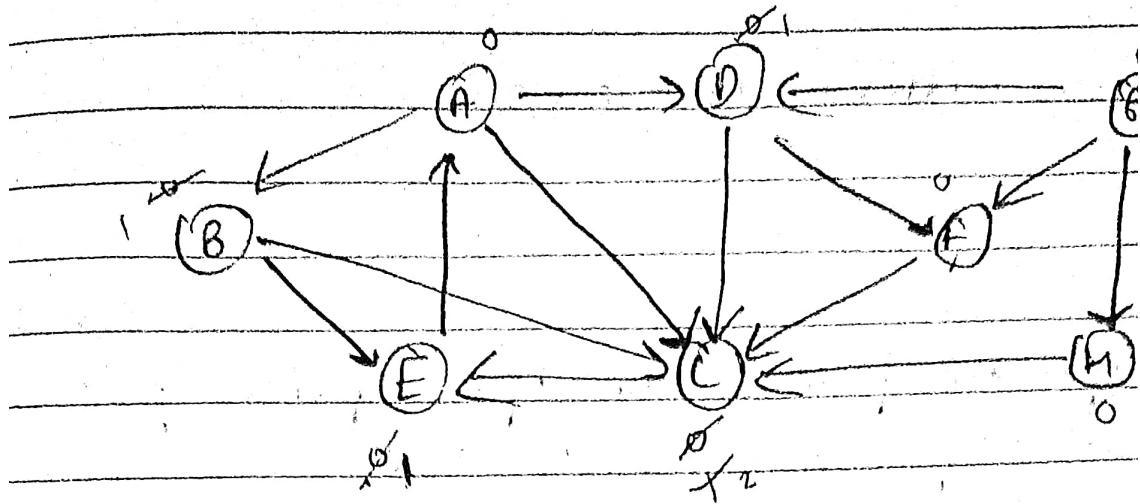
BFS :- $\text{A} \rightarrow \text{parent node}$

$\text{B} \leftarrow \text{child node}$

- taking any one child node, say 'B'

$\text{C} \leftarrow \text{child node}$ & furthermore ...

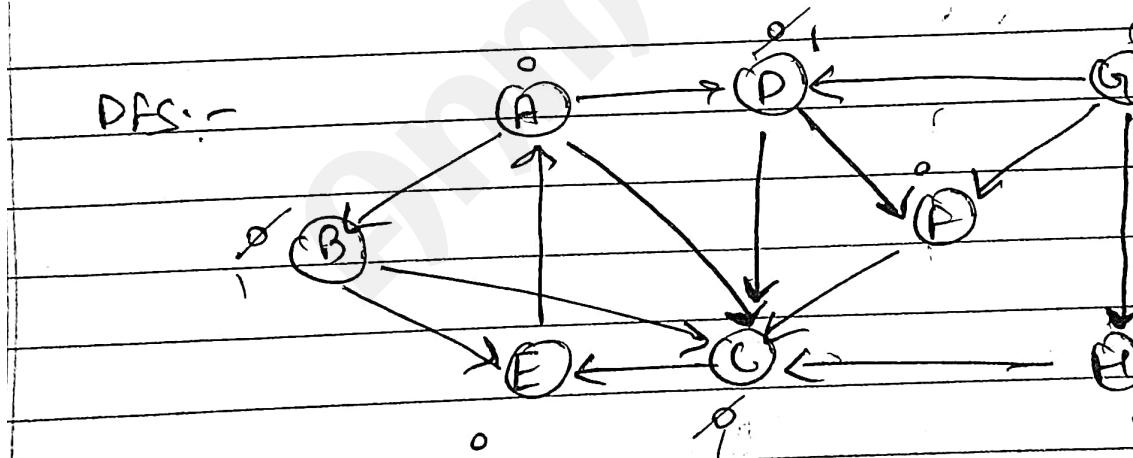
(A, B)



Patient	A	A	A	B	B	D	D
Node	B	C	D	E	F	C	F

path: - $A \rightarrow B \rightarrow C$.

DFS:-



node stack
processed ←

A A

BWR ABED

B BE

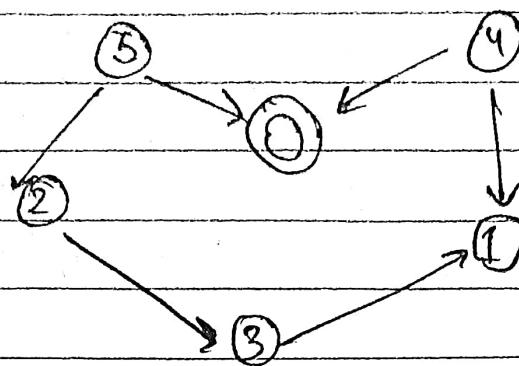
E BEC

C EC

path $A \rightarrow B \rightarrow C$

Q.7.) Apply topological sorting & DFS on graph having vertices from 0 to 5

(a.)



Adj cent list (G)

$0 \rightarrow 1$

$1 \rightarrow$

$2 \rightarrow 3$

$3 \rightarrow 1$

$4 \rightarrow 0, 1$

$5 \rightarrow 0, 2$

0 1 2 3 4 5

visited - - - - -

stack (empty)

step 1: Topological sort(0), visited[0] = true

list-empty (no recursion call).

stack [0]

Step 2: Topological sort[2], visited[2] = true
" " [3] = true

- already visited, no more execution

call

stack [0 | 1 | 3 | 2]

Step 5 - " [5] " [5] = true.

2, 0 already visited no recursion

call

stack [0 | 1 | 2 | 3 | 4 | 5]

Print all elements of stack from top → bottom.

Q.8) Can heap be used to implement priority queue?

Name few algorithm where you need to use priority queue?

A.8) Heaps can be used to implement priority queue. It takes $O(\log N)$ time to insert & delete each element in priority queue.

Algorithm where priority queue can be applied are Dijkstra, Prim's algorithm

Q.9) Difference between Max & Min heap?

It is used to
maximum element
in heap

It is used to access
minimum element in heap