Tutorial 5

Q1

BFS	DFS
i) Uses ~~Stack~~ queue Data structure	ii) Uses ~~stack~~ queue data structure.
ii) Here we reach a vertex with min. no. of edges from a source vertex.	ii) We might traverse through more edges to reach a destination vertex from a source.
iii) It consider all neighbour first and therefore not suitable for decision making trees used in games or puzzles.	iii) Here we make a decision, then explore all ~~paths~~ through this decisions. And if this decision leads to win situations, we stop.
iv) Time complexity of BFS is $O(V+E)$ in case of adjancency list and $O(V^2)$ in case of adjancency matrix.	iv) Time complexity of Dfs is $O(V+E)$ in case of Adjacency list and $O(V^2)$ in case of adjacency matrix
v) Here, Siblings are visited before the children.	v) Children are visited before the siblings.

Applications

i) Peer to Peer Networks:

ii) Social Networking Websites: In social networks, we can find people within a given distance 'k' from a person using Breadth first search till 'k' levels.

iii) GPS Navigation System: Breadth first search is used to find all neighboring locations.

i) Detecting cycle in a graph.

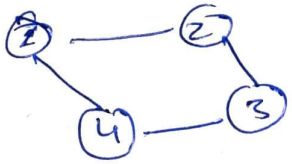
ii) Topological Sorting.

iii) Solving puzzles with only one soln.

Q2 BFS uses queue. The algorithm makes sure that every node is visited not more than once. That's why we use queue because it maintains a queue of visited nodes.

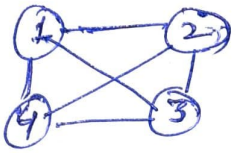
DFS uses stack because stack is used in the standard implementation of depth first search. It is used to store the elements which are to be explored.

Q3 Sparse graph is a graph in which the no. of edges is close to the minimal no. of edges.



sparse graph

dense graph is a graph in which no. of edges is close to maximal no. of edges.



dense graph

for sparse graph we uses adjacency list.
for dense graph we uses adjacency matrix

Q4 Detecting cycle in BFS

Step1:- Compute in-degree (no. of incoming edges) for each of the vertex present in the graph and initialize the count of visited nodes as 0

Step2:- Pick all vertices with in-degree as 0 and add them into a queue.

Step3:- Remove a vertex from the queue.
i) Increment count of visited nodes by 1.
ii) Decrease in-degree by 1 for all its neighbouring nodes
iii) If in-degree of a neighbouring nodes is reduced to 0, add it to the queue

Step4:- Repeat steps until the queue is empty.

Step5:- If count of visited nodes is not equal to no. of nodes in graph has cycle, otherwise not

Detect cycle in DFS

Step1:- Create graph using no. of edges & vertices.

Step2:- Create a recursive func. that initialises the current index for vertex, visited and recursion stack.

Step3:- Mark the current node as visited and also mark the index in recursion stack.

Step4:- find all the vertices which are not visited and are adjacent to the current node. Recursively call the func. for those vertices, if the recursive func. returns true, return true.

Step5:- If the adjacent vertices are already marked in the recursion stack then return true.

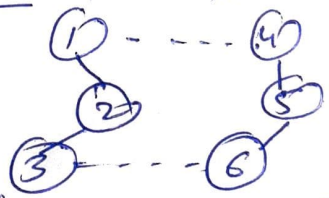
Step6:- Create a wrapper class, that calls the recursive func. for all the vertices and if any func. returns true retur true. Else if for all vertices the func. returns false return false.

Q5

Disjoint set is a data structure that stores a collection of disjoint (non-overlapping) sets.

Operations

find:- Tells the set to which an element belongs



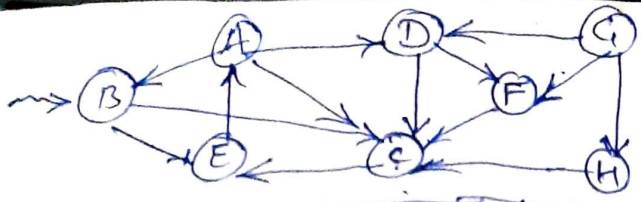
find(1) = S_1, Here $S_1 = \{1,2,3\}$

$S_2 = \{4,5,6\}$

Union:- Merge two sets when an edge is added.

$S_1 \cup S_2 = \{1,2,3,4,5,6\}$

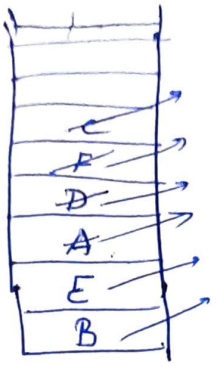
(6)



Queue | B | E | C | A | D | F | G | H |

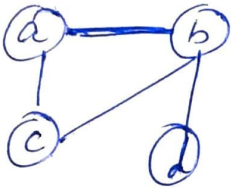
B/S: B, E, C, A, D, F, ~~G, H~~

~~B, E~~
stack



D/S: B, E, A, D, F, C

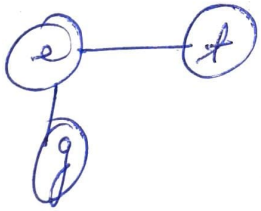
(7)



$V = \{a, b, c, d\}$

$= \{\{a,b\}, \{a,c\}, \{b,c\}, \{b,d\}$

	~~{a}{b}, {a,c}, {b,c}, {b,d}~~
~~{a,b}~~	{a}, {b}, {c}, {d}
{a,b}	{a,b} {c} {d}
{a,c}	{a, b, c} {d}
{b,c}	{a, b, c} {d}
{b,d}	{a, b, c, d} ← 1 connected graph

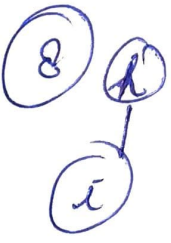


$V = \{e, t, g\}$

$E = \{\{e,t\}, \{e,g\}\}$

	{e} {t} {g}
{e,t}	{e,t} {g}
{e,g}	{e,t,g} ← 1 connected graph

(8)

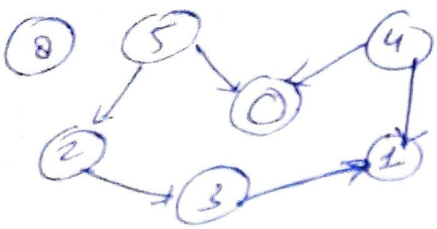


(9)

$V = \{h, i, j\}$

$E = \{\{h,i\}, \{\}\}$

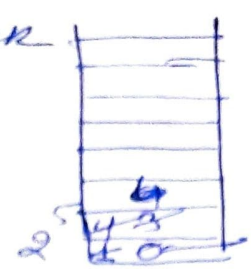
	{h} {i} {j}
{h,i}	{h,i} {j} ← 2 connected graph
{j}	{h,i} {j}



Topological sort

Stack | 0 | 1 | 3 | 2 | 4 | 5 |

DFS stack



0, 1, 2, 3, 4, 5

(9) Priority queue is used in many algorithms :-

- Dijkstra's algorithm :- finding a shortest path in a graph
- Prim's algorithm :- Constructing a min. spanning tree of a graph.
- Huffman's algorithm :- Constructing an optimum prefix-free encoding of a string.

(10)

Max heap	Min heap
i) key present at the root node must be greater than or equal to among the keys present at all of its children.	i) key present at root node must be less than or equal to among the keys present at all of its children
ii) Max. key present at the root.	ii) Min. key present at root.
iii) Uses descending priority	iii) Uses ascending priority.