Name: AKSHAT SINGHAL
SEC : CSE-SP1 1
Roll no : 04

Page No.
Date : / /20

Tutorial 5

1.

BFS	DFS
① Queue data structure is used	① Stack data structure is used.
② Useful when searching for vertices closer to	② useful when destination node is farther from source node.
③ Not suitable for decision making trees used in games or puzzles.	③ More suitable for game or puzzle problems.
④ Siblings are visited before the children.	④ Children are visited before the siblings.
⑤ Requires more memory	⑤ Requires lesser memory.
APPLICATION: In peer-to-peer networks, used to find all neighbors.	APPLICATION: We can detect cycles in a graph.

2. BFS is implemented using queue data structure because BFS explores the closest vertices first and then moves away from the source. So, we need a data structure that gives us the oldest element based on the order they were inserted, that is why we use queue for BFS.

DFS is used to detect cycles, so we have to backtrack, backtracking is possible by using stack, here we use stack for DFS.

Page No.
Date : / /20

3. In sparse graph, the no. of edges is closer to the minimal no. of edges.

In dense graph, the no. of edges is closer to the maximal no. of edges.

A sparse graph should be stored as a list of edges and a dense graph should be stored as an adjacency matrix.

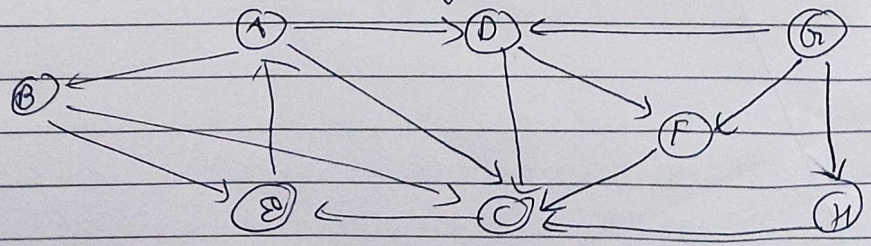
4. In both BFS & DFS, we maintain a visited list and while traversing a level, if we encounter a node that was already in the list, we come across a cycle. And we say that cycle exist in graph.

5. A disjoint set data structure is also known as union - find data structure and merge - find set. It contains collection of disjoint or non - overlapping sets.

Operation that can be performed on this data structure are:

1. Intersection
2. union
3. delete an element.

6. Run BFS & DFS on graph



Page No.
Date : / /20.

BFS:

Start node: G , Dest: B

queue	Actions	visited
G	Insert D, F, H Remove G	G
D | F | H	Insert C Remove D	G, D, F, H
F | H | C	Remove F	G, D, F, H C
H | C	Remove H	G, D, F, H, C
C	Insert E Remove C	G, D, F, H, C
E	Remove E Insert A	G, D, F, H, C, E
A	Remove A Insert B	G, D, F, H, C, E, A

Path is G D F H C E A B

G → D → F → H → C → E → A → B

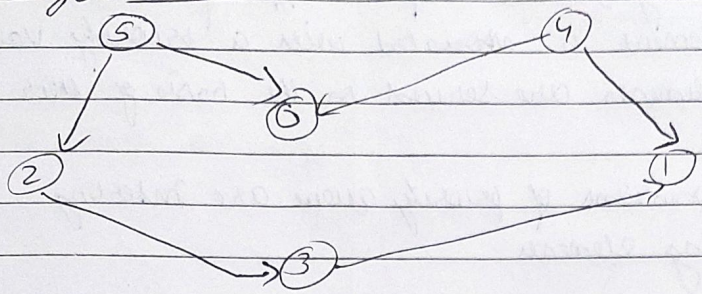
DFS:

Stack	Node	visited
G	Insert D	G
D, G	Insert C	G, D
C, D, G	Insert E	G, D, C
E, C, D, G	Insert A	G, D, C, E
A, E, C, D, G	Insert B	G, D, C, E, A

Path is: G → D → C → E → A → B

Page No.
Date : / /20

3. Topological sort.



5
4
2
3
1
0

T.S (0)
no neibhour

T.S (1)
no neighbour

T.S (2)
↓
T.S(3)
↓
no neighbour

T.S (4)

T.S (5)

5, 4, 2, 3, 1, 0 Ans

Q. A priority queue is a special type of queue in which each element is associated with a priority value. And elements are served on the basis of their priority.

Basic operations of priority queue are Inserting, removing & peeking elements.

using heap data structure:

Insert: Insert new element at end of heap, then heapify the tree

deleting: Select element, swap with last element, Remove last element

Heapify the tree

Peeking: Return root node

Applications:

- Dijsktra's Algo
- for implementing stack
- data compression in Huffman code

10 Max & Min Heap diff:

Min	Max
• In Min-heap the key present at the root must be less than or equal among keys present at all of its children.	• In Max-heap, the key present at the root node must be greater or equal among the keys present at all of its children.
• All parent nod are smaller than child node	• All parent node are larger than child node
• Min-heap uses ascending ~~property~~ priority.	• Max-heap uses descending priority.
• In Min-heap, highest priority is of smallest element	• In Max-heap highest priority is of largest ele