Q 1) Difference btw DFS & BFS. Write applications of both algorithm

	BFS	DFS
12)	Breadth First Search	Depth First Search
	Uses queue	Uses stack
	Suitable when destination is close to start node.	Suitable when destination is far from source.
	Not suitable for decision making trees used in games & puzzles	DFS is more suitable for games or puzzle problems.
13)	Siblings visited before children	Siblings visited after children
	Requires more memory	Less memory.
	No concept of backtracking.	Recursive algo that uses backtracking

Applications :-

BFS → Bipartite graph and shortest path, peer to peer networking, crawlers in search engine & GPS navigation system.

DFS acyclic graph, topological order, scheduling problems, sudoku puzzle

15	16	17	2018	2019
↓	↓	↓	↓	↓
6	7	10	11	12

Q. Which data structure are used to implement BFS & DFS and How

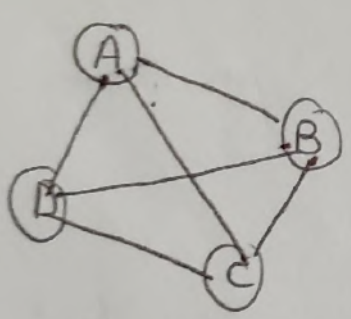
For BFS we use queue

For DFS we use a stack

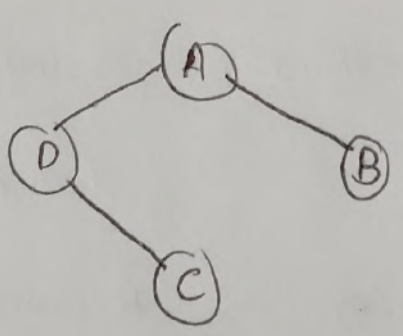
Ques What do you mean by sparse and dense graphs. Which representation of graph is better for spare & dense graph?

Dense graph is where no. of edge is close to maximal no. of edge.

Sparse graph has less no of edge.



Dense



Sparse

For sparse graph we use adjacency list.

For dense graph we use adjacency matrix

How do we detect a cycle in a graph using BFS & DFS

For detecting a cycle in graph using BFS, we use Kahn's algorithm for Topological sorting

Steps involved are

1) Compute in-degree (no of incoming edges) for each of vertex present in graph & initialise no of visited nodes as 0.

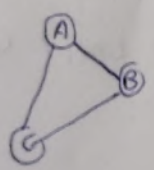
2) Pick all vertices with in-degree as 0 & add them in queue.

3) Remove a vertex from queue and then.

   1) Increment visited nodes
   2) Decrease in degree by 1 for all its neighbouring node
   3) If in degree of neighbouring node is reduced to zero then add to queue.

4) Repeat 3) until queue is empty.

5) If count of visited nodes is not equal to no of nodes in graph has cycle, otherwise not.



A → 2

B → 2

C → 2

To use DFS for the same,
DFS for a connected graph produces a tree. There is a cycle in a graph if there is a back edge present in graph. A back edge is an edge that is from a node to itself (self-loop) or one of its ancestors in the tree produced by DFS. For a disconnected graph, get DFS forest as output. To do cycle, check for a cycle in individual trees by checking back edges. To detect a back edge, keep track of vertices currently in recursion stack for DFS traversal. If a vertex is reached that is already in recursion stack, then there is a cycle.

6 What do you mean by disjoint set data structure? Explain 3 operations along with examples which can be performed on disjoin sets?

A disjoint set is a data structure that keeps track of set of elements partitioned into several disjoint subsets. In other words, a disjoint set is a group of sets where no item can be in more than one set.

3 operations

Find Implemented recursively traversing parent array until we hit a node who is parent to itself.

```
int find (int i)
{   if (parent[i]==i)
      { return i;
    }
    else { return find(parent[i]);
    }
}
```

ion → Takes 2 elements as input and find representative of their set, using the find operation & finally puts either one of the trees under root of other tree, effectively merging the trees & sets.

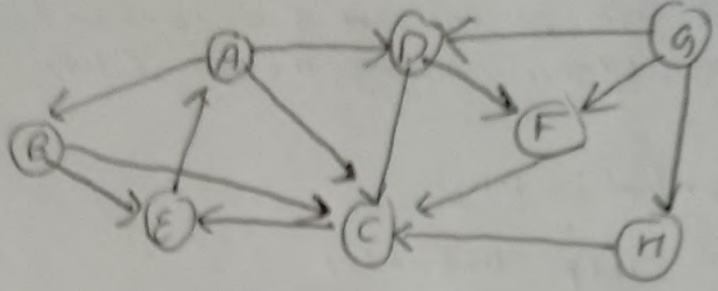
```
void union (int i, int j)
{
    int irep = this.find(i)
    int & jrep = this.find(j)
    this.parent[irep] = jrep;
}
```

union by Rank → we need a new array rank[]. Size of array same as parent array. If i is representative of set, rank[i] is height of tree. We need to minimize height of tree. If we are uniting 2 trees, we call them left & right. Then it all depends on the rank of left & right. If rank of left is less than right then it is best to move left under right & vice versa.

If ranks are equal, rank of result will always be one greater than rank of tree.

```
void union (int i, int j)
{ int irep = this.find(i);
  int jrep = this.find(j);
  if (irep == jrep) return;
    irank = Rank[irep];
    jrank = Rank[jrep];
    if (irank < jrank)
    this.parent[irep] = jrep;
    else if (jrank < irank)
     this.parent[jrep] = irep;
     else { this.parent[irep] = jrep;
            Rank[jrep]++;
     } }
```

Q) Run BFS & DFS on below graph



BFS

child	G	H	D	F	C	E	A	B
Parent		G	G	G	H	C	E	A

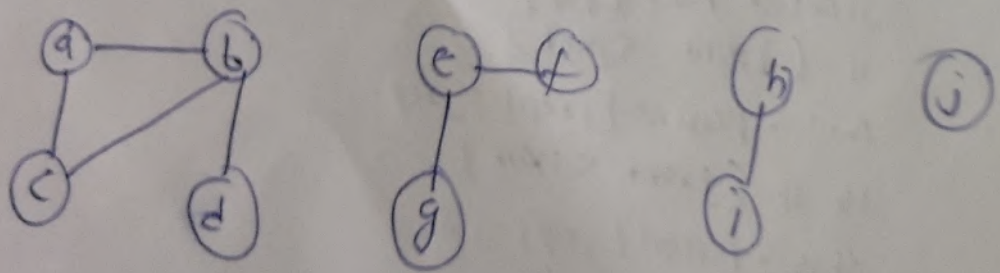
Path $G \rightarrow H \rightarrow C \rightarrow E \rightarrow A \rightarrow B$

DFS

Nodes visited	STACK
G	G
D	F
H	C
F	E
C	A
E	B
A	
B	

Path → $G \rightarrow F \rightarrow C \rightarrow E \rightarrow A \rightarrow D$

Find no-of connected components and vertices in each component using disjoint set data structure



V = {a} {b} {c} {d} {e} (f) (g) (h) (i) (j)

E = {a,b}, {a,c}, {b,c}, {b,d}, {e,f}, {e,g}, {h,i}, {i,j}

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

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

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

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

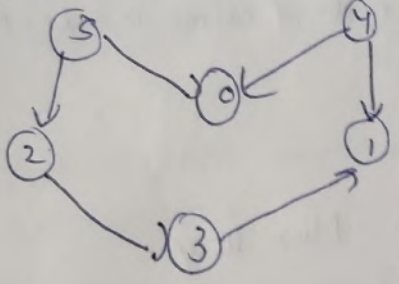
(e,f) {a,b,c,d} {e,f} {g} {h} {i} {j}

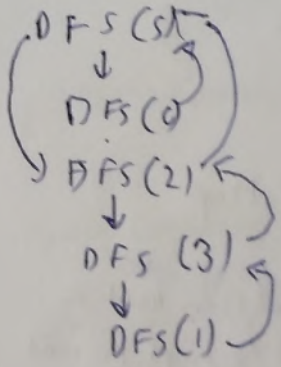
(e,g) {a,b,c,d} {e,f,g} {h} {i} {j}

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

(3)

Apply topological sort & DFS on graph having vertices from 0 - 5





DFS(4)
↓
Not Possible

4
5
2
3
1
0

4 → 5 → 2 → 3 → 1 → 0

Heap can be used to implement priority queue.
N can few graph algorithm where you

Yes heaps are used to implement priority queue.
It will take O(logN) time to insert & delete each element in priority queue.

Based on heap structure there are
1) Max Heap
2) Min Heap

Graph algorithms like Djikstra's, Prim's used priority queue.

1) Djikstra → Priority queue is used to extract minimum
2) Prim's used to store keys of nodes & extract min key node at every step

Min Heap

Key present at root node must be less than or equal to keys present at all of it's children

Min key element is present at root.

Uses ascending priority

Smallest element is popped first

Max Heap

Key present at root node must be greater than or equal to keys present at all of it's children

Max key element is present at root.

Uses descending priority

Largest element is popped first