1. Difference b/w BFS and DFS.

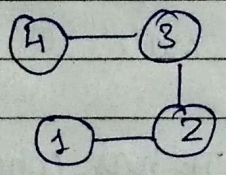
BFS	DFS
◦ It stands for breadth first search	◦ It stands for depth first search.
◦ It uses queue data structure for finding path.	◦ It uses stack data structure.
◦ BFS gives shortest path from source to destination.	◦ DFS gives one of possible paths from source to destination.
◦ BFS considers all the neighbours of source node and then their neighbours.	◦ DFS considers a source node, then considers its source node till the end.
◦ Time complexity List: $O(V+E)$ Matrix: $O(V^2)$	◦ Time Complexity List: $O(V+E)$ Matrix: $O(V^2)$
◦ No concept of back tracking as it do not uses recursion	◦ It has concept of backtracking ∵ it uses recursion.
Application: 1. Shortest path. 2. MST for unweighted graph 3. Peer to Peer networks. 4. Social Networking sites. 5. GPS navigation. 6. Cycle detection indication	Application: 1. Topological sorting 2. Finding strongly connected components of graph. 3. Solving puzzels with only one solution. 4. Path finding.

2. Data structures used in case of BFS and DFS are queue and stack respectively.

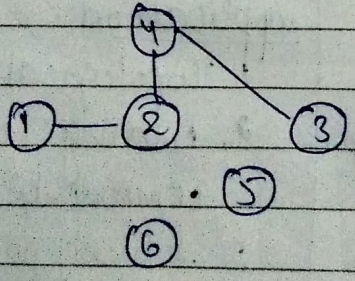
→ BFS requires queue data structure because it traverses a graph in breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

→ DFS requires stack data structure because it traverses a graph in depthward motion and uses a stack to remember to get to the next vertex to start a search, when a dead end occurs in iteration.

3. Dense graph is a graph in which number of edges is close to the maximal number of edges.



Sparse graph is a graph in which number of edges is close to minimal number of edges.



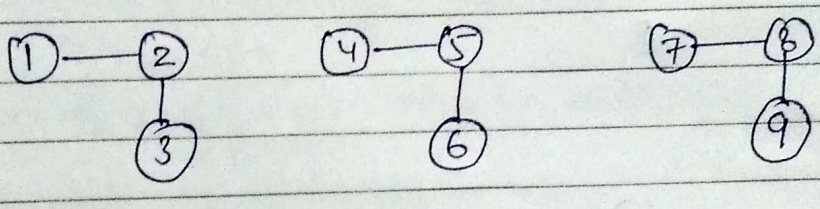
→ It is ideal to use sparse graph by adjacency list and dense graph by adjacency matrix.

4. Disjoint Set Data Structure:

A disjoint set data structure also called as union-find data structure or merge. Find data structure is a data structure that stores a partition of set into disjoint subsets.

It provides operations for adding sets, merging sets and finding a representative member of a set.

e.g. $S1 = \{1, 2, 3\}$
$S2 = \{4, 5, 6\}$
$S3 = \{7, 8, 9\}$



operations performed:

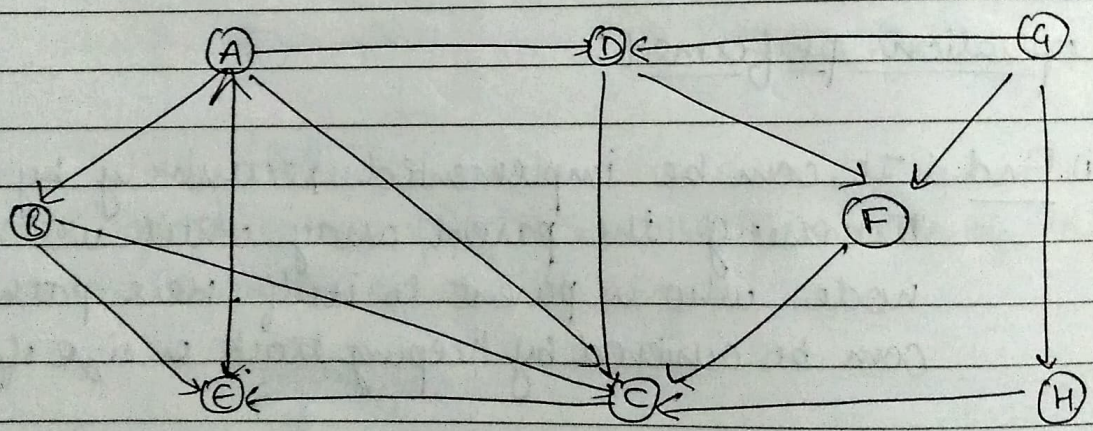
(i) Find: It can be implemented recursively by traversing the parent array until we hit a node who is parent to itself. Here, path comparison can be achieved by keeping track of size of the node.

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

(ii) Union: It takes as input two elements and finds the parent of the inputs using find operation and performs merging of the one child to the parent node.

```
void union (int a, int b)
  {
    a = find(a);
    b = find(b);
    if (a==b)
    {
     parent[a] = b;
    }
  }
```

6.



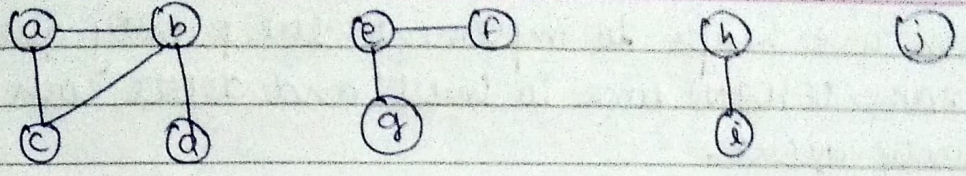
BFS.

Node	B	E	C	A	D	F
Parent	-	B	B	E	A	D

DFS.

B	E	A	D	F	C

7.



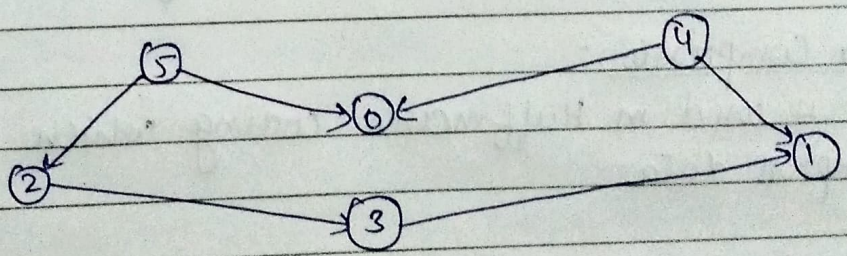
$U = \{a, b, c, d, e, f, g, h, i, j\}$

Here, make, find, and union operation will be performed.

→ First each vertice will be parent of itself.
→ If an edge exists between vertices, than we perform find operation and check if their parent is same and perform union operation on them.

$\therefore S_1 = \{a, b, c, d\}$
$S_2 = \{e, f, g\}$
$S_3 = \{h, i\}$
$S_4 = \{j\}$

8.



toposort(0) : push(0).
toposort(1) : push 1.
toposort(2) → toposort(3)
first push 3 - then push 2.
toposort(4) : push 4.
toposort(5) : push 5.

5
4
2
3
1
0

$\therefore$ | 5 | 4 | 2 | 3 | 1 | 0 |

9. We can use heaps to implement the priority queue. It will take O(log N) time to insert and delete each element in priority queue.

→ Based on heap structure, priority queue has also two types – max priority and min priority.

Some algorithms where we need to use priority queue:

(i) Dijkstra's:
- used for calculating shortest path in algorithm.
- when the graph is stored in form of adjacency list or matrix, priority queue can be used to extract efficiently.

(ii) Prim's:
- used to implement prim's algo to store keys of nodes and extract minimum key node at every step.

(iii) Data Compression:
- It is used in Huffman's coding which is used to compress data.

10.	Min heap	Max heap
	• In a min heap, the key present at root must be less than or equal to the children	• In max heap, key at root must be greater than the key at the children.
	• Min. key element is at root of heap.	• Max. key element is at root of heap.
	• uses ascending priority.	• uses descending priority.
	• Smallest element has a priority.	• Largest element has a priority.
	• Smallest element is to be popped from heap.	• Largest element is to be popped from heap.