Tutorial-5
DAA

Name - Ayushman Bhatt
Section - AI & DS
Class Roll No - 21
University Roll No - 2017640

Answer 1

BFS	DFS
(i) It stands for Breadth First Search.	It stands for Depth First Search.
(ii) It uses Queue data structure.	It uses Stack data structure.
(iii) It is more suitable for searching vertices which are closer to given source.	It is more suitable when there are solutions away from source.
(iv) BFS considers all neighbors first & therefore not suitable for decision making trees used in games & puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if the decision leads to win situation, we stop.
(v) Here siblings are visited before children.	Here siblings are visited ~~after~~ after the children, that means the children are visited before the siblings.
(vi) There is no concept of back-tracking.	It is a recursive algorithm that uses backtracking.
(vii) It requires more memory.	It requires less memory

Applications :-

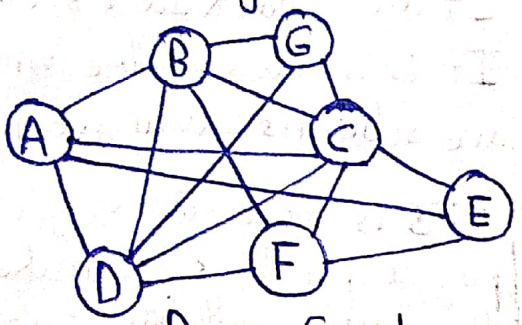
- BFS → Bipartite graph and shortest path, peer to peer networking, crawlers in search engine and GPS navigation system.
- DFS → Acyclic graph, topological order, scheduling problems, sudoku puzzles.

Answer 2 For implementing BFS we need a queue data structure for finding shortest path between any node. We use queue because things don't have to be processed immediately, but have to be processed in FIFO order like BFS. BFS searches for nodes level-wise, i.e., it searches nodes with respect to their distance from root (source). For this queue is better to use in BFS (Breadth First Search).

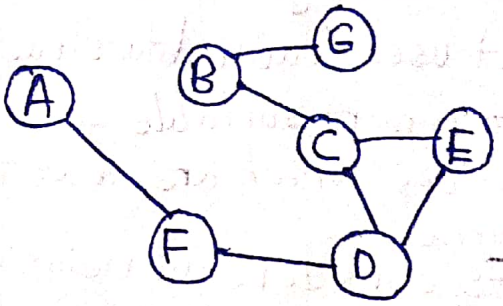
For implementing DFS we need a stack data structure as it traverses a graph in depthward motion and uses stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration. (2)

Answer3 » Dense graph is a graph in which no. of edges is close to maximal no. of edges.

» Sparse graph is a type of graph in which no. of edges are minimal or very less.



Dense Graph
(many edges in between nodes)



Sparse Graph
(Few edges in between nodes)

⊙ For sparse graph it is preffered to use Adjacency List.

⊙ For dense graph it is preferred to use Adjacency Matrix.

Answer 4 For detecting cycle in using BFS we need to use Kahn's algorithm for Topological sorting.

The steps involved are:-

1) Compute in-degree (no. of incoming edges) for each of vertex present in graph & initialize count of visited nodes as 0.
2) Pick all vertices with in-degree as 0 and add them in queue.
3) Remove a vertex from queue and then →
   - Increment count of visited nodes by 1.
   - Decrease in-degree by 1 for all its neighbouring nodes.
   - If in-degree of neighbouring nodes is reduced to zero then add to queue.
4) Repeat step 3) until queue is empty.
5) If count of visited nodes is not equal to no. of nodes in graph has cycle, otherwise not.

For detecting cycle in graph using DFS we need to do following:- DFS for a connected graph produces a tree. There is cycle in graph if there is a back edge present in the graph. A back edge is an edge that is from 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 detect a cycle, check for a cycle in individual trees by checking back edges. To detect a back edge, keep track of vertices currently in recursion track for DFS traversal. If a vertex is reached that is already in recursion stack, then there is a cycle.

Answer 5 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 our one set.

3 operations →

⇒ Find → can be implemented by recursively traversing the parent array until we hit a node which is a parent to itself.

Example:-

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

⇒ Union → takes 2 elements as input and finds representatives of their sets using the find operation and finally puts either one of the trees under root node of other tree, effectively merging the trees and sets.

Example:-

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

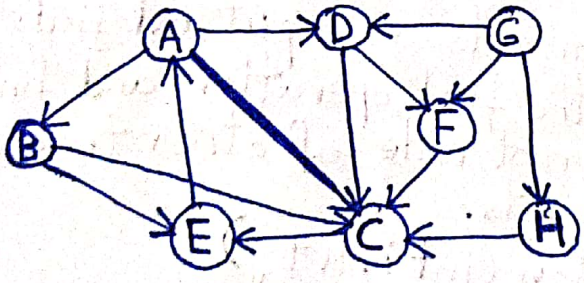
-> Union by Rank → We need array rank[]. Size of array is same as parent array. If i is representative of set, rank[i] is height of tree. We need to minimize height of tree. If we are uniting 2 trees, we call them left and right, then it all depends on rank of left and right.

- If rank of left is less than right then it's best to make left under right and vice-versa.
- If ranks are equal, rank of result will always be one greater than the rank of trees.

Example:-

```
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]++;
  }
}
```

Answer 6



BFS

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

Path → G → H → C → E → A → B

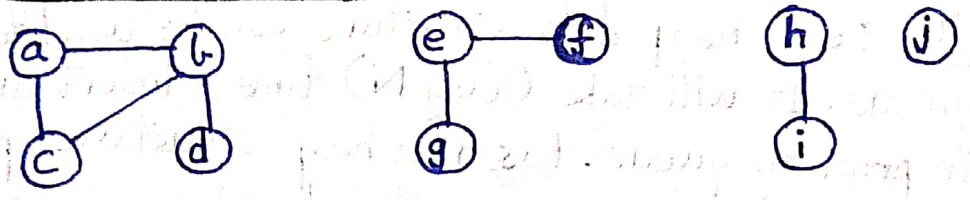
DFS

~~G~~ H D ~~F~~ ~~C~~ ~~E~~ ~~A~~ ~~B~~ } NODES VISITED

G F C E A B } STACK

Path → G → F → C → E → A → B

Answer 7



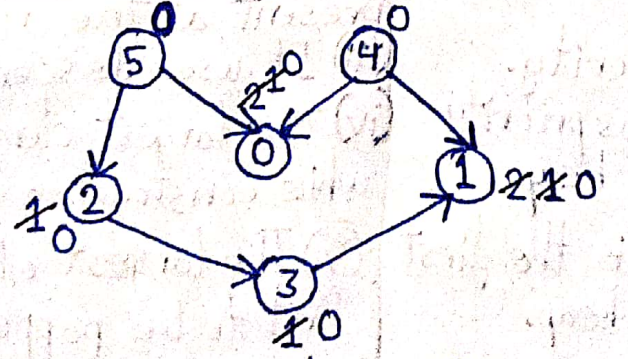
$V = \{a\}\{b\}\{c\}\{d\}\{e\}\{f\}\{g\}\{h\}\{i\}\{j\}$

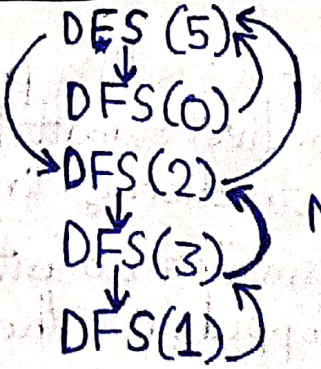
$E = \{a,b\},\{a,c\},\{b,c\},\{b,d\},\{e,f\},\{e,g\},\{h,i\},\{j\}$

(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}

No. of connected components = 3 Ans

Ans 8





DFS(4) → Not possible

We take source node as 5.

Applying Topological sort ⟶ q: 5/4; Pop 5 & decrement indegree of it by 1. ⑥

DFS

4
5
2
3
1
0

Stack

4 → 5 → 2 → 3 → 1 → 0

q: 4/2; Pop 4 & decrement indegree & push 0.
q: 2/0; Pop 2 & decrement indegree & push 3.
q: 0/3; Pop 0, Pop 3, Push 1.
q: 1; Pop 1.
Answer: 5 4 2 0 3 1

Topological Sort

Answer 9: Yes, heap data structure can be used to implement 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 two types max-priority queue based on max heap and min priority queue based on min-heap. Heaps provide better performance comparison to array and Linked list.
The graphs like Dijkstra's Shortest Path algorithm, Prim's Minimum Spanning Tree use Priority Queue.
➢ Dijkstra's Algorithm → When graph is stored in form of adjacency list or matrix, priority queue is used to extract minimum efficiently when implementing the algorithm.
➢ Prim's Algorithm → It is used to store keys of nodes and extract minimum key node at every step.

Answer 10

Min-Heap	Max-Heap
(i) In min-heap, key present at root node must be less than or equal to keys present among all of its children.	(i) In max-heap, the key present at root node must be greater than or equal to keys present among all of its children.
(ii) The minimum key element is present at the root.	(ii) The maximum key element is present at the root.
(iii) It uses ascending priority.	(iii) It uses descending priority.
(iv) The smallest element has priority while construction of Min-heap.	(iv) The largest element has priority while construction of Max-heap.
(v) The smallest element is the first to be popped from the heap.	(v) The largest element is the first to be popped from the heap.