Tutorial - 5

Name: HARSH RASTOGI　　Class Roll No - 06

Section - F　　University Roll No - 2016761

1. What is the difference between DFS and BFS? Write applications of both the algorithms.

BFS	DFS
1.) It stands for Breadth First Search.	1.) It stands for Depth First search.
2.) It uses queue data structure.	2.) It uses stack data structure.
3.) It is more suitable for searching.	3.) It is more suitable when there are solutions away from source.
4.) BFS considers all neighbours first & therefore, not suitable for decision making trees used in games & puzzles.	4.) DFS is more suitable when games & puzzles problems, We make a decision then explore all paths through the decision.
5.) Here siblings are visited before children	5.) Here children are visited before siblings.
6.) There is no concept of ~~ba~~ backtracking.	6.) It is a recursive algorithm that uses backtracking.
7.) It requires more memory.	7.) It requires less memory.

Applications:-

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

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

2: Which data structure are used to implement BFS & DFS and why?

Ans:-

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 immediately in FIFO order like BFS. BFS searches for nodes level wise; it searches nodes w.r.t. their distance from root (source). For this queue is better to use in BFS.

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

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

Ans:-

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

Sparse graph is graph in which no. of edges is very less.

For sparse graph, it is preferred to use Adjacency list.

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

4. How to detect a cycle in a graph using BFS & DFS?

Ans:- For detecting cycle in a graph, using BFS, we need to use Kahn's algorithm, for Topological sorting.

The steps involved are:-

1.) computer in-degree (no. of incoming edges) for each of vertex present in graph and 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 indegree 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 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 back edge, keep track of vertices currently in recurrence track.

5. What do you mean by disjoint set data structure? Explain 3 ~~co~~operations

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

3 operations:-

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

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

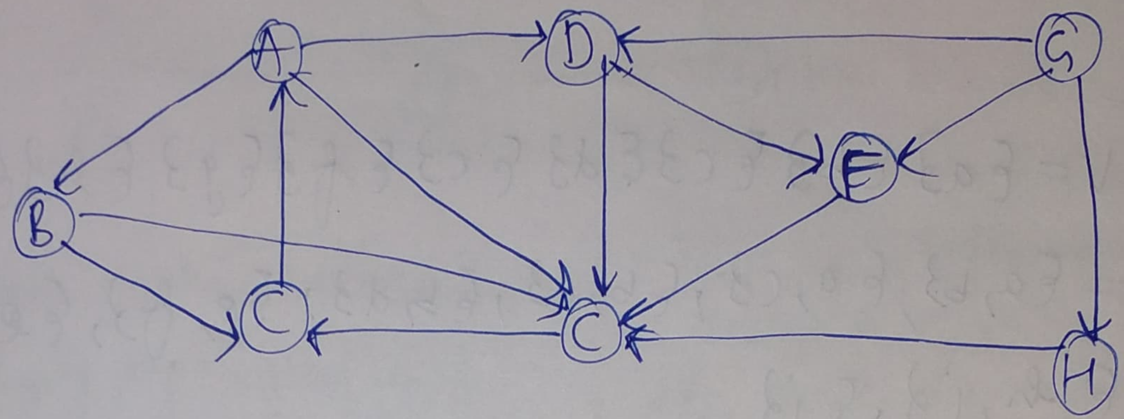
2.) Union → It takes 2 elements as input. And find repeatitiveness of these sets using the find operation. and finally puts either one of the tree under restnode of other 2, effectively merging the 2 and sets.

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

3) 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 minimise height of tree. If we are uniting 2 trees, we call them left & right, then it is all depends on rank of left and right.
If rank of left is less than right then it all depends on rank of left and right.
If ranks are equal, rank of result will always be one greater than rank of trees.

Eg→
```
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(jrank < irank)
      {  this.parent[irep] = jrep;
      }
      else if(jrank > irank)
      {  this.parent[jrep] = irep;
      }
else {
  this.parent[irep]=jrep;
   Rank [jrep]++;
   }
}
```

6. Run BFS and DFS on graph shown below.



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

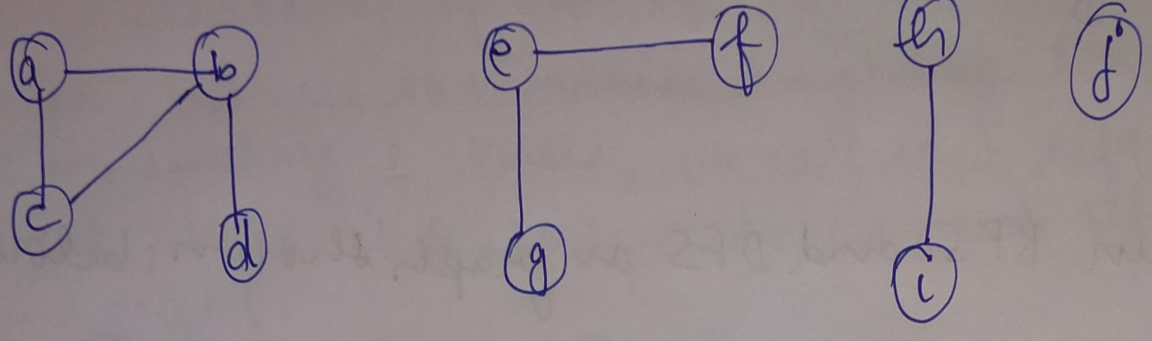
DFS

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

G
F
C
E
A
B
STACK

Path → G → F → C → E → A → B

7. Find out no. of connected components and vertices in each component using adjacent set of data structure.



Ans:

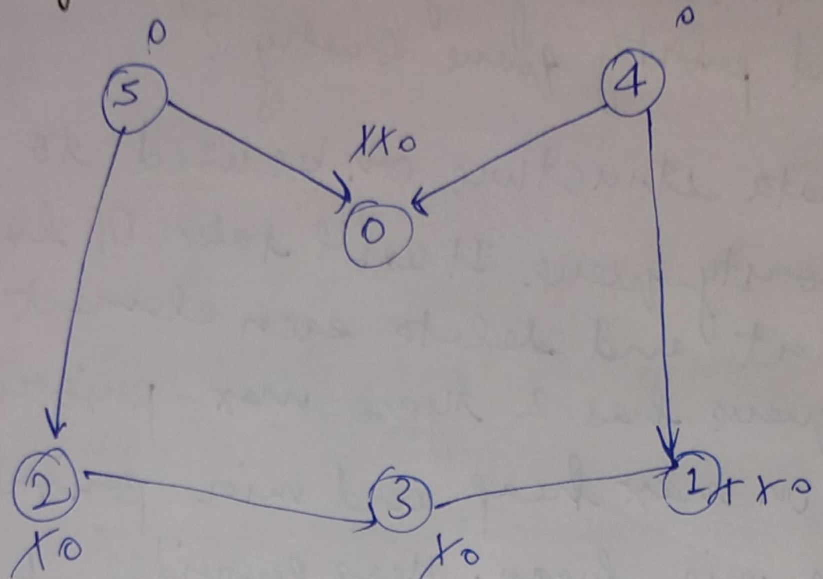
V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

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.

8. Apply topological sort & DFS on graph having vertices from 0 to 5.



We take source node as 5.

Applying topological sort

q : 5/4 ; Pop 5 & decrement in degree of it by 1.

q : 4/2 ; Pop 4 & decrement indegree & push 0

q : 2/0 , Pop 2 & decrement in degree & push 3.

q : 0/3 Pop 0, Pop 3 Push 1.

q : 1 ; Pop 1

Answer : 5 4 2 0 3 1

DFS(5)
↓
DFS(0)

DFS(2)
↓
DFS(3)
↓
DFS(1)

DFS(4)
↓
Not possible

DFS

4
5
2
3
1
0

stack

4 → 5 → 2 → 3 → 1 → 0 Ans

Ques 9.
Heap data structure can be used to implement priority queue. Name few graph algorithm where you find priority queue & why?

Ans:-
Yes, heap data structure can be used to implement priority queue. It will take O(logN) time to insert and delete each element in priority queue has 2 types max-priority queue based on max heap and min priority queue based on min-heap. Heap provides better performance comparision to array.
The graphs like Dijkstra's shortest path algorithm, Prim's algorithm use Priority queu

* <u>Dijkstra's Algorithm:-</u> when graph is stand in form of adjacency matrix or list, priority queu is used to extract minimum efficiency when implementing the algorithm.
* <u>Prim's Algorithm</u> :- It is used to store keys of nodes and extract minimum key node at every step.

Q10

Difference between min-heap and max-heap.

Ans:- <u>Min-heap</u>

In min-heap, key present at root node must be less than or equal to among keys present at all of its children.

The minimum key element is present at the root.

It uses ascending priority.

The smallest element has priority while construction of min-heap.

The smallest element is the first to be popped from the heap.

<u>Max heap</u>

In max-heap, the key present to be root node must be greater than or equal to among keys present

The maximum key element is present at the root.

It uses descending priority.

The largest element has priority while construction of Max-heap.