Tutorial - 5

Name :- Dalesh Goel

Section :- F

Roll no. :- 51

Univ. Roll no. :- 2016713

Q.1. what is difference b/w DFS and BFS. Write applications of both the algorithms.

BFS	DFS
• It stands for Breadth First search.	• It stands for Depth First Search.
• It uses Queue data structure	• It uses stack data structure
• It is more suitable for searching vertices which are closer to given source.	• It is more suitable when there are solutions away from source.
• BFS considers all neighbours first of therefore not suitable for decision making trees used in games & puzzles.	• DFS is more suitable for games & puzzles. We make a decision, then explore all paths through this decision. And if decision leads to win situation we stop.
• There is no concept of backtracking.	• It is recursive algo. that uses backtracking.
• It requires more memory	• It requires less memory.

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.

Q.2. Which data structures are used to implement BFS and DFS and why?

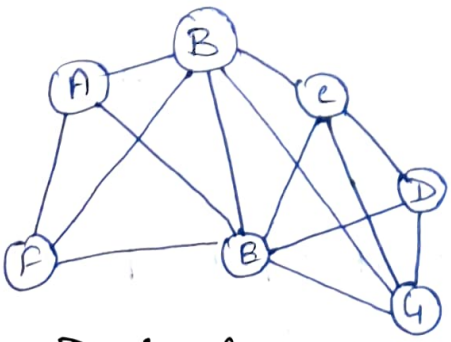
→ For implementing BFS we need a queue data structure for finding shortest path b/w any node. We use queue because they don't have to be processed immediately, but have to be processed in FIFO order like BFS. BFS searches for nodes level wise, i.e. it searches nodes wrt their distance from root (source). 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 ~~wa~~ ward motion and uses stack to remember to get next vertex to start a search, when a dead end occurs in any iteration

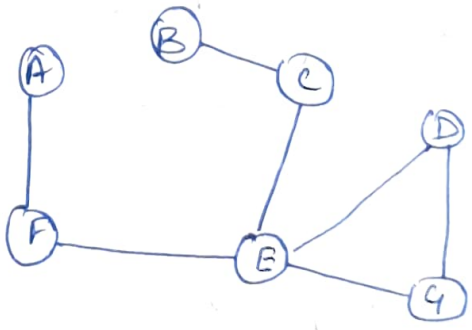
Q.3. What do you mean by sparse and dense graph which representation of graph is better for sparse and dense graph?

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

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



Dense Graph



Sparse Graph

- For sparse graph it is preferred to use Adjancy List.
- For dense graph it is preferred to use Adjacency Matrix.

Q.4. How can you detect a cycle in graph using BFS and DFS?

→ For detecting cycle in graph using BFS we need to use Kahn's algorithm for Topological sorting

Steps:-

1. Compute in-degree (no. of incoming edges) for each of vertex present in graph & initialize count of visited node as 0.
2. Pick all vertices with in-degree as 0 and add them in queue.
3. Remove vertex from queue and then
   - Increament count of visited nodes by 1
   - Decrease in-degree by 1 for all its neighbouring nodes.
   - If in-degree of neighbouring nodes is not equal to no. of nodes in graph has cycle, otherwise not.
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.

Q.5. What do you mean by disjoint set data structure? Explain 3 operations along with examples which can be performed on disjoint sets?

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

3 operations:-

Find:- can be implemented by recursively traversing 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]);
    }
}
```

Union!- It takes 2 elements as input. And find representation of their sets using the find operation and finally puts either one of the trees under root node of other tree, effectively merging trees and sets.

eg:-

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

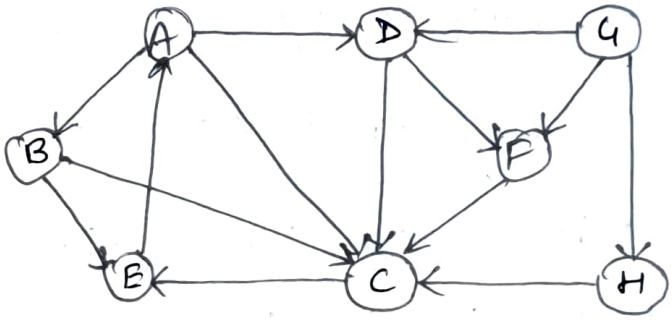
Union by Rank!- we need a new array rank[] size of array same as parent array. If i is representative of set, rank[i] is height of tree. We need to minimize height of tree. If we are uniting 2 trees, we call them left and right then it all depends on rank of left and right
- If rank of left is less than right then it's best to move left under right & vice versa.
- 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 (irank < jrank)
        this.parent[irep] = jrep;
      else if (jrank < irank)
        this.parent[jrep] = irep;
      else {
        this.parent[irep] = jrep;
        Rank[jrep]++;
      }
    }
```

Q.6. Run BFS and DFS on graph shown below

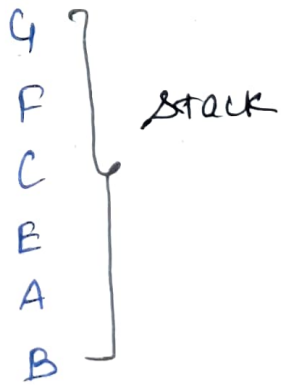


BFS :-

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

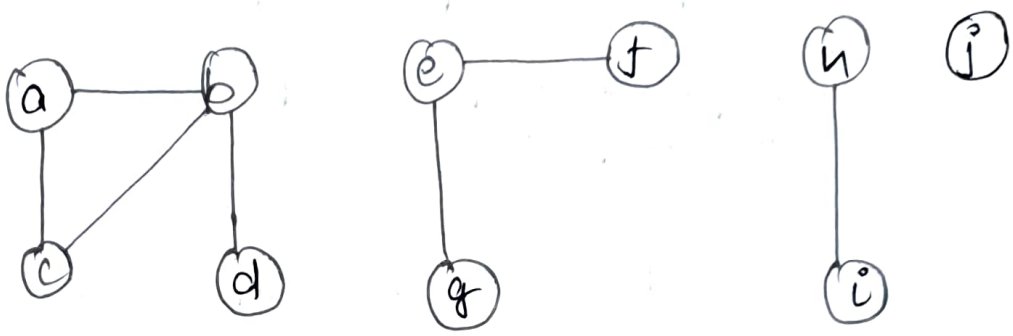
Path ⟹ G → H → C → B → A → B

DFS :-



Path → G → F → C → E → A → B

Q.7. Find out no. of connected components and nodes in each component using disjoint set data structure.



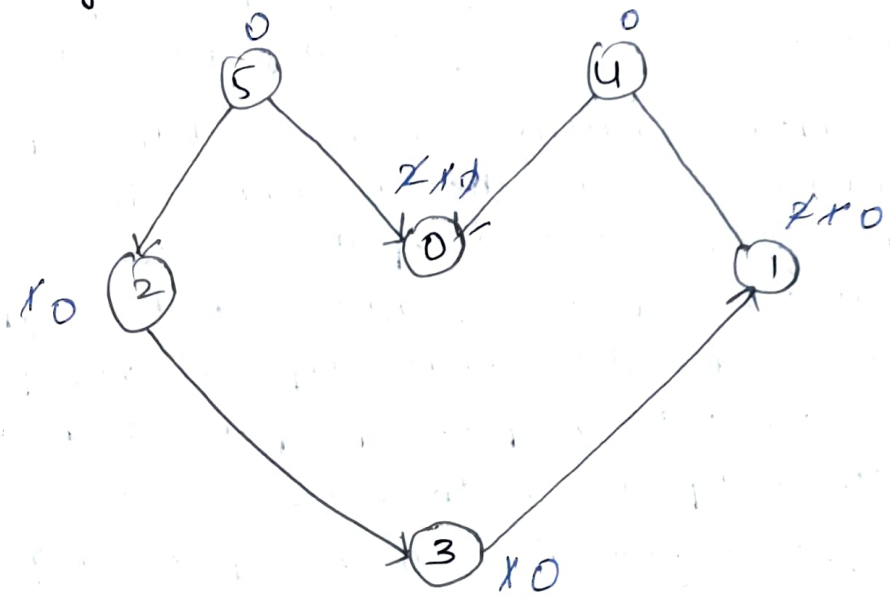
→ V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

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

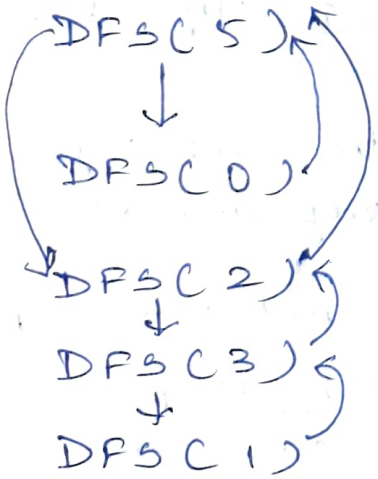
No. of connected components = 3

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



→ We have source node as 5

Applying topological sort



DFS(4)
↓
NOT possible

q: 5/4; pop 5 & decrement indegree of it by 1

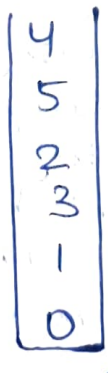
q: 4/2; pop 4 & decrement indegree & push 0

q: 2/0; pop 2 & decrement indegree & push 3

q: 0/3; pop 0, pop 3 & push 1

q: 1; pop 1

DFS



4 → 5 → 2 → 3 → 1 → 0

Q.9. Heap data structure can be to implement priority queue. Name two graph algorithm where you need to use priority queue and why?

→ 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 & Linked List. The graphs like Dijkstra's shortest path algorithm, Prism'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 algorithm.
- Prism's Algorithm :- It is used to store keys of node and extract minimum key node at every step.

Q.10. Differentiate b/w Min-Heap and Max-Heap.

Min-Heap	Max-Heap
• The min-key element is present at root.	• The max. key element is present at root.
• It uses ascending priority.	• It uses descending priority.
• The smallest element has priority while construction of Min-heap.	• The largest element has priority while construction of Max-heap.
• The smallest element is first to be popped from the heap.	• The largest element is first to be popped from the heap.