Aditi Banerji
38-F

Tutorial - 5

Q1. What is difference between DFS and BFS. Write applications both the algorithms.

Sol^n

BFS

1) It stands for Breadth First Search.
2) It uses queue data structure.
3) It is more suitable for searching vertices which are closer to given source.
4) BFS considers all neighbours first and therefore not suitable for decision making trees used in games & puzzles.
5) There siblings are visited before children.
6) There is a Concept of Backtracking.
7) It Requires more Memory.
8) Bipartite graph and shortest path, peer to per Networking, Crawlers in Search Engine and GPS Navigation System.

DFS

1) It stands for Depth first Search.
2) It uses stack data structure.
3) It is more suitable when there are solutions away from Source.
4) DFS is more suitable for game or puzzle problems. We make a decision then explore all paths through this decision. And if decision leads to win situation, we stop.
5) There children are visited before siblings.
6) It is a Reccursive Algorithm that uses backtracking.
7) It requires less Memory.
8) Acyclic graph, topological order, Scheduling problems, Sudoku puzzle.

Q2. Which data structure are need to implement BFS and DFS and why?

↳ For implementing BFS, we need a queue data structure for finding shortest path between any node. We use queue because ~~things~~ because things don't have to be processed immediately, but have to be processed in FIFO order like BFS.

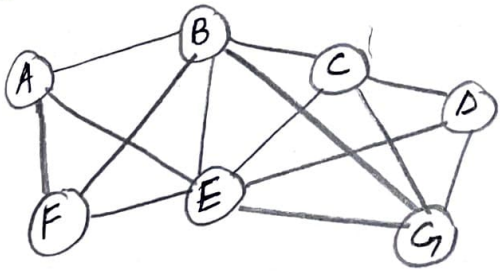
BFS searches for nodes level wise, i.e it searches nodes with respect to their distance from source. for this queue is better to use in BFS.

For Implementing DFS we need a stack data structure as it transverse 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

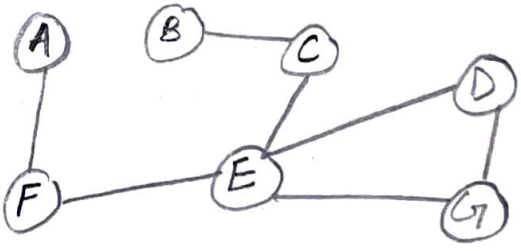
Q3. what do you mean by sparse and dense graph? which Representation of graph is better of space and dense graph?

→ Dense graph is a graph in which number of edge is close to maximal number of edges.

Sparse graph is a graph in which number of edges is very less.



Dense graph
(many edges b/w nodes)



Sparse graph
(few cycles b/w nodes)

→ for sparse graph it is prefered to use Adjancy list.
→ for Dense graph it is prefered to use Adjancy Matrix.

Q4. How can you detect a cycle in a graph using BFS and DFS? (3)

→ For Detecting a cycle in a graph using BFS we need to use Kahn's Algorithm for Topological Sorting.
The steps involved are:

1) Compute in-degree (number of incoming edges) for each of the vertex present in graph and initialize count of visited nodes as 0.
2) Pick all vertices with 0-in degree and then add them in queue.
3) Remove a vertex from queue and then
   - Increment count of visited node by 1
   - Decrease in-degree by 1 for all its neighbouring nodes.
   - If in-degree of neighbouring node is reduced to 0 then add to queue.
4) Repeat (3) until queue is empty.
5) If count of visited node is not equal to number of nodes in graph, graph has cycle otherwise not.

→ For Detecting cycle in DFS we need to do following:
DFS for a connected graph produces a tree. There is a cycle in a 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 as output. To detect 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 that is a cycle.

c) **Union By Rank :**

We need a new array rank[]. Size of Array same as parent array. If i 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.

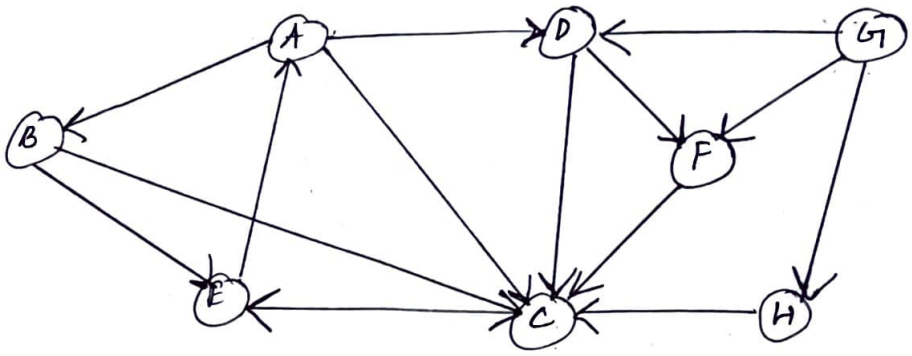
→ If Rank of Left is less than right then it's best to move left under right and vice-versa.

→ If Rank 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]++;
  }
}
```

Q6. Run BFS and DFS on graph shown below: (6)



- **BFS traversal:**

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

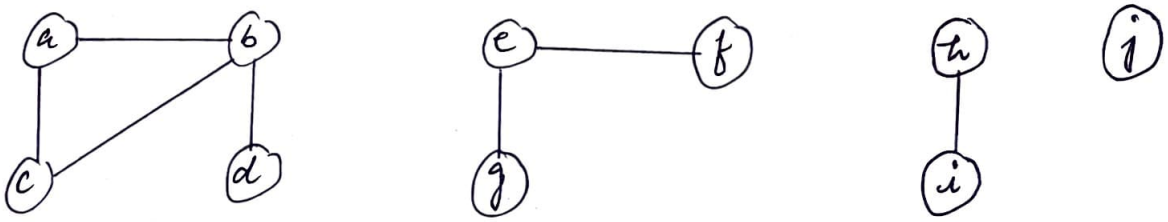
Path: G → H → C → E → A → B

- **DFS traversal:**

Nodes visited	Stack
~~G~~	G
D	F
H	C
~~F~~	E
~~C~~	A
~~E~~	B
~~A~~	
~~B~~	

Path: G → F → C → E → A → B

Q7. Find out Number of Connected Components and vertices in each component using Disjoint Set Data structure.



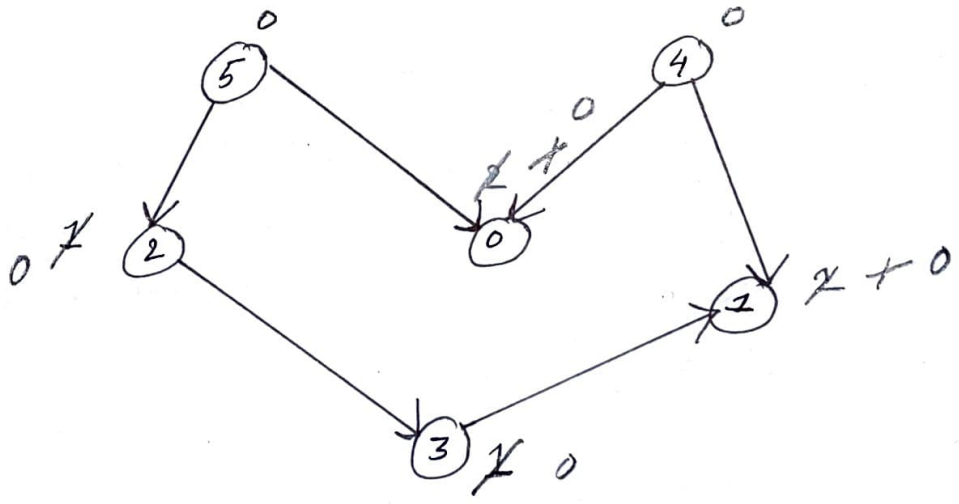
→ Vertices $V = \{a\}\{b\}\{c\}\{d\}\{e\}\{f\}\{g\}\{h\}\{i\}\{j\}$

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

(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}

Number of Connected Components = 3

Q8. Apply Topological Sort and DFS on graph having vertices from each 0 to 5.



↳ we take Source node as '5'

Applying Topological Sort :

1st we push in-degree of value '0' in queue ie 5 & 4

q : 5 | 4

Now pop '5' from 'q' and decrement in-degree of its adjacent by 1 ie 0 and 2

Since in-degree of (2) is now '0' we push it into queue.

ie q: 4 / 2

Now, pop 4 and decrement in-degree of its adjacent ie '0' and '1' and push '0' onto queue.

ie q: 2 / 0

Now pop 2, and decrement in-degree of 3, and push 3 into queue.

ie q: 0 / 3

Now pop 0 Since 0 doesnot have any adjacent

Now pop 3, and decrement in-degree of 1 and push 1

ie q: 1

Now pop 1. , End of Topological Sort.

Answer: 5 4 2 0 3 1

→ Applying DFS.:

DFS (5)
DFS (0)
DFS (2)
DFS (3)
DFS (1)

and By using a loop for disconnected graph we call DFS (4)

∴ DFS traversal: 5 → 0 → 2 → 3 → 1 → 4

Q9. Heap data structure can be used to implement priority queue. Name few 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 data structure priority queue has 2 types ie max-priority

queue based on max-heap and min-priority queue based on min-heap. heap provide better performance comparison to array and linked list.

The graph like Dijktra's shortest path Algorithm, Prim's Minimum Spanning Tree use Priority Queue.

- **Dijktra's Algorithm**: when graph is stored in form of Adjancey 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.

Q10. Differentiate between Min-heap and Max-heap.

Min-heap	Max-heap
1) In min-heap key present at root node must be less than or equal to among keys present at all of its children.	1). In max-heap the key present at root node must be greater than or equal to among keys present at all of it's children.
2) The minimum key element is present at the Root.	2) The maximum key element is present at Root.
3) The smallest element has priority while construction of Min-heap.	3) The largest element has priority while construction of max-heap.
4) The smallest element is the first to be popped from the heap.	4) The largest element is the first to be popped from the heap.