Tutorial - 5

Name - Dhruv Goyal

Section - F

Roll No - 54

University Roll No - 2016729

Q1) What is difference b/w DFS & BFS. Write applications of both the algorithms

BFS	DFS
a) It stands for Breadth first search	It stand for depth first search
b) It uses queue	It uses stack
c) It is more suitable for searching	It is more suitable when there are solutions away from source.
d) BFS considers all neighbours first of therefore not suitable for decision making trees used in games of puzzle	DFS is more suitable for game or puzzle problem. We make a decision then explore all paths through this decision And if decision leads to wing situations we stop.
e) Here siblings are visited before children.	Here children are visited before siblings
f) There is no concept of back tracking	It is recursive algorithm that uses back tracking.
g) It requires more memory	It requires less memory.

#Applications -

a) BFS - Bipartite graph & shortest path, peer to peer networking, crawlers in search engine of GPS navigation system.

b) DFS - acyclic graph topological order, scheduling problems, sudoku puzzles

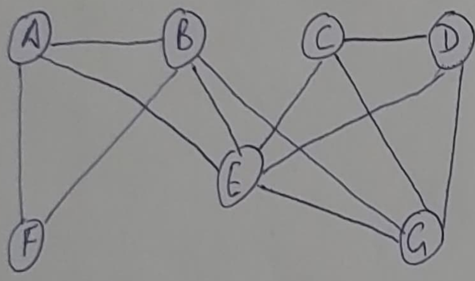
Q2) Which data structure are used to implement BFS & DFS & why?

For implementing BFS we need a queue a data structure for finding a shortest path b/w any node. We use queue because things don't have to be processed immedially, but have in FIFO order like BFS. BFS search for nodes levelwise ie it searches nodes w.r.t their distance from root (source) for this queue is better to use in BFS.

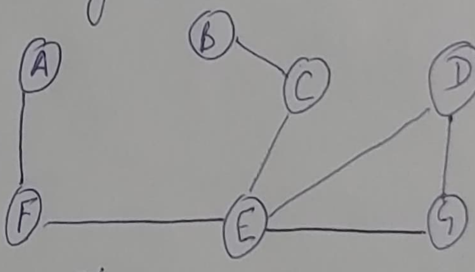
For implementing DFS we need a stack data structure as it traverses a graph in depth order motion & uses stack to remember to get next vertex to start a search. When a dead end occurs in any iteration.

Q3.) What do you mean by sparce graph and dense graph? Which represtation of graph is better for space & dense graph?

Dense graph is a graph in which no of edges is close to maximal no of edges
Sparse graph is graph is which no of edges is very less.



Dense graph



Sparse graph

For sparse graph it is perferered to use adjency list.
For dense graph it is preferred to use adjacency Matrix.

Q4.) How you can detect a cycle in graph BFS & DFS?

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

The steps involved are-

1.) Compute in-degree (no of incoming edge) for each vertex present in graph & initialize count of visited node as 0.

2.) Pick all vertices with in degree as 0 & add them in queue.

3.) Remove a vertex from queue & then

- increment count of visited node 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 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 connected graph produces a tree. There is cycle in graph if there is a back edge present in graph. A back edge is an edge that is in form a node to itself (self loop) or one of its ancestors in tree produced by DFS. For a disconnected graph get DFS forest as output. To detect cycle, check for cycle in individual trees by checking back edge. 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.

Q5.) What do you mean by disjoint set data structure? Explain 3 operations along with example which can be performed on disjoint set?

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

3 Operations -

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

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

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

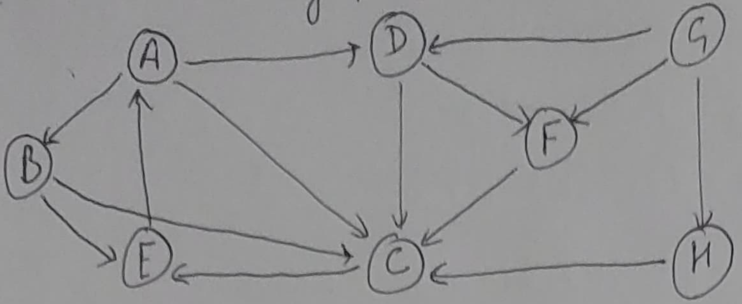
```
void union (int i, int j) {
    int irep = this.Find(i);
    int jrep = this.Find(j);
    this.parent[irep] = jrep;
    }
```

- 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 tominimice height of tree. If we are uniting 2-trees, we call them left & right, then it all depends on rank of left & right.
    - If rank of left is less than right then its 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]++;
    }
}
```

Q6.) Run BFS & 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~~

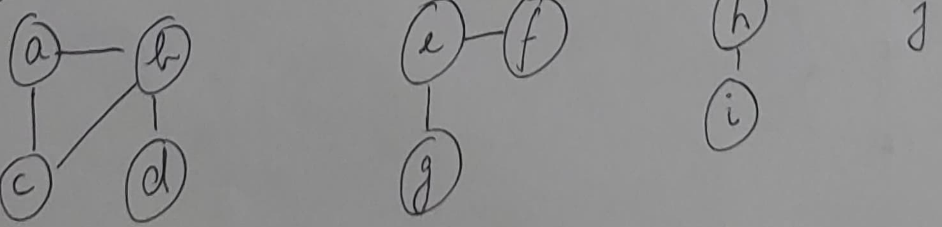
Nodes Visited

G
F
C
E
A
B

Stack

Path - G → F → C → E → A → B

Q7.) Find out no. of connected components and vertices in each component using disjoint set data structure.



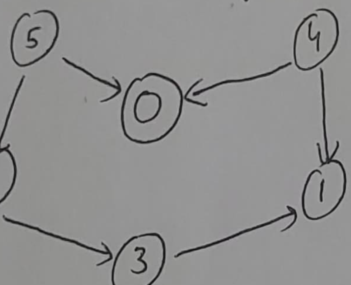
V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

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}

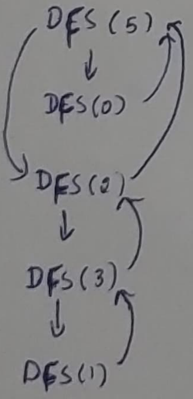
No of connected components = 3

Q8) Apply topological sort & DFS on graph having vertices 0 to 5



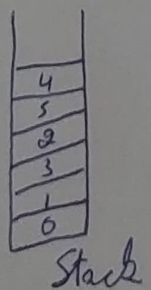
We have source node as 5.

Apply topological sort



DFS(4)
↓
Not possible.

DFS



Stack

4 → 5 → 2 → 3 → 1 → 0

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 by implement priority queue. It will take O(log N) time to insert & delete each element in priority queue. Based on heap structure priority queue has two types max priority queue based on max heap & min priority queue based on min-heap. Heap provide better performance comparision to array & linked list.

The graphs 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 algorithm.
- Prim's Algorithm - It is used to store keys of nodes & extract minimum key node at every ste

Q10.) Difference b/w Min-heap & Max-heap.

Min-heap	Max-heap
·) Min-heap, key is present at root node must be less then or equal to among keys present at all of its children.	·) In max-heap the key present at root node must be greater than or equal to among keys present at all its children
·) The min key element is present at root.	·) The max key element is present at root.
·) The 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 the first to popped from heap.	·) The largest element is first to popped from the Heap.