Tutorial - 5

NAME → DIVYANSHU KAIRA
Section - F
Roll No - 56
U. Roll No - 2016740

Q1) Write difference between DFS and BFS. Write application of both the algorithms

Ans

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 closure to given source	• It is more suitable when there are solution away from source.
• BFS considers all neighbour first and therefore not suitable for decision making trees used in games and puzzle	• DFS is more suitable for game or puzzle problem. We make a decesion then explore all paths through this decesion, Leads to win situation we stop
) Here siblings are visited before children	• Here children are visited before sibling.
) There is no concept of backtracking	) It is a recursive algorithm that uses backstracking
) It requires more memory	) It requires less memory.

Application

) 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 puzzle.

Q4) which data structure are used to implement BFS and DFS and why?

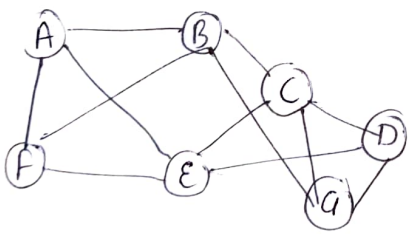
↳ For implementing BFS we need a queue data structure for finding shortest path between any node we use queue because things don't have to be processed immediately, but have to be processed in FIFO order like BFS. BFS searches for nodes level wise, it searches nodes w.r.t these distance from root (source). 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 mard motion and uses stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

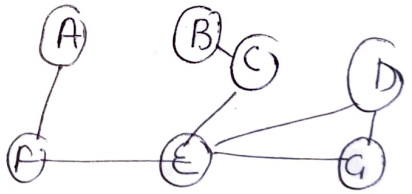
Q5) What do you mean by sparse and dense graph? Which representation of graph is better for aparse and dense graph?

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

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



Dense Graph
(many edge b/w nodes)



Space Graph
(few edges b/w nodes)

:) For space graph it is prepared to use Adjacency list
:) For dense graph it is preferred to use Adjacency matrix.

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

Ans For detecting cycle in a graph using BFS we need to use Kohn's algorithm for topological sorting.
The steps involved are:

1) Compute in degree (no. of incoming edges) for each of vertex present in graph & intialize 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
   a) increment count of visited nodes by 1
   b) Decrease in-degree by 1 for all its neighbouring nodes
   c) If in degree of neighbouring nodes is reduced to zero then add to queue.

4) Repeat

5) Until queue is empty

6) 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 a cycle in graph if there is a back edge present A black 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 found as output. To ~~output~~ detect cycle, check for a cycle in individual tree by checking back edge. To detect a back stage, keep track of vertices currently for DFS traversal. If a vertex is searched that is already in recursion stack, then there is a cycle.

5) What do you mean by disjoint set data structure? Explain operation along with example which can be performed on disjoint sets?

Ans A disjoint set is a data structure that keeps track of set of elements partioned into disjoined subset. In order words a disjoint set is a group of sets where no item can be in more than one set.

3 operations:

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

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

b) Union. It takes 2 element as input And find representatives of this sets using find operation and finally puts either one of the tree under root of other tree, effectively merging the trees and sets

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

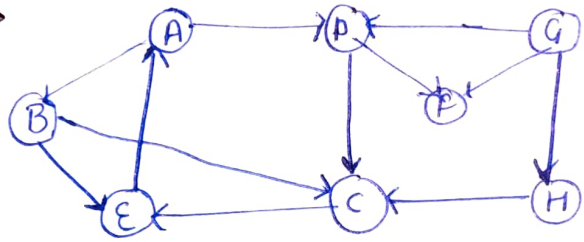
c) Union by Rank: We need a new array rank(), same of array same as parent array. If i is representative of set rank[i] is height of tree we need teminimic height of tree. If we are uniting 2 trees, we call them left is less than right than its best to move left under right and vice versa

- If rank of left is ~~zero~~ less than right its best to move left under right and 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];
   if(irank < jrank)
     this.parent[jrep]=irep;
   else{
     this.parent[irep]=jrep;
      Rank[jrep]++;
   }
}
```

5) Run BFS and DFS on graph shown below.



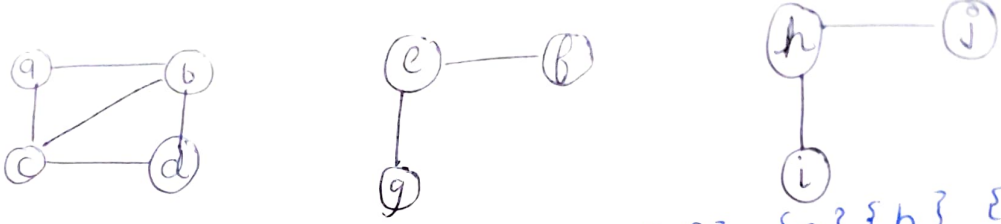
BFS child G H D E G C E A B
parent G G G H C E A

Path → G → H → C → E → A → B

Node visited	Stack
G	G
D	F
H	C
F	E
C	A
E	B
A	
B	

Path → G → F → C → E → A → B

7) Find out no. of connected components and vertices in each component using disjoint set 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) $\{a,b,c,d\}\{e\}\{f\}\{g\}\{h\}\{i\}\{j\}$

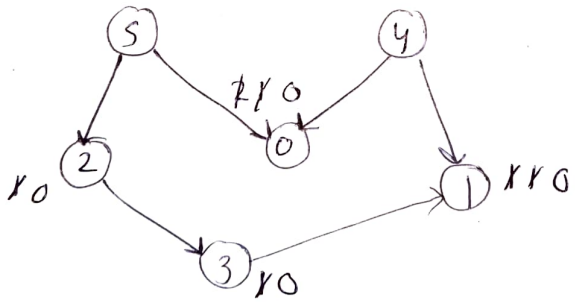
(e,f) $\{a,b,c,d\}\{e,f\}\{g\}\{h\}\{i\}\{j\}$

(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

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



Ans we take source node as 5

Applying Topological Sort → q: 5/4 ; Pop 5 and decrement indegree of it by 1

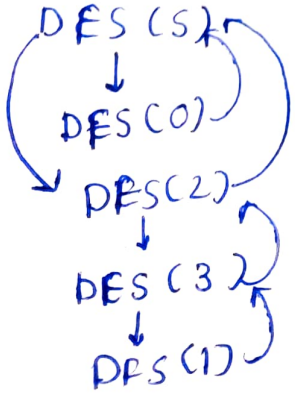
q: 4/2 ; Pop 4 and decrement indegree & push 0

q: 2/0 Pop 2 and decrement indegree and push 3

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

q: 1; Pop 1

Ans 5 4 2 0 3 1



DFS(4)
↓
not panible

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

Ans 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 has two types max-priority queue based on max-heap and min priority queue based on min-heap Heaps provide better performance comparison to arrays LOL.

The graphs like Dijkotra's shortest path algorithm, Prisms minimum Spanning Tree use Priority Queue.

- Dijkstra's Algorithm → when graph is stored in form of adjacent list or matrix, priority queue is used to extract minimum efficiently when implementing the algorithm
- Prisms Algorithm → It is used to store keys of nodes and extract minimum key node at every step.

Q10) Difference between min-heap and max-heap.

↳

Min-heap	max-heap
a) In min-heap, key present at root node must be less than or equal to among keys present at all of its children	a) In max-heap the key present at root node must be greater than or equal to among keys present at all of its children
b) The minimum key element is present at the heap root	b) The maximum key element is present at the root.
c) It uses ascending priority	c) It is uses descending priority
d) The smallest element has priority while construction of min-heap	d) The largest element has priority while construction of max heap.
e) The smallest element is the first to be popped from the heap	e) The largest element is the first element to be popped