Satyam Rawat
2017006
G1
16 (Class Roll No)

DAA
Tutorial – 5

Q1> What is difference between DFS & BFS. Please write their applications.

Soln.

DFS	BFS
1.) Uses stack data structure	Uses queue data structure
2.) It is used to find a path (if exists) from a source node to destination	It is used find the path from source node to destination with minimum number of nodes,
3.) Stands for Depth First Search	Stands for breadth first search
4.) It visits the children before the siblings of a node	It visits siblings before the children
5.) Applications – • ~~GPS Navigation System~~ • Detecting cycle in a graph • Topological Sorting	Applications – • GPS Navigation System • Social Networking Websites

Q2> Which Data Structures are used to implement BFS & DFS and Why?

Ans In BFS we use queue data structure as queue is used to remember, to get the next vertex to start the search when an end occurs in any iteration.

Queue is used when things don't have to be processed immediately, but in FIFO order.

DFS uses stack data structure since it traverses a graph in depthward motion. It uses a 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 & dense graphs? Which representation of graph is better for sparse & dense graphs?

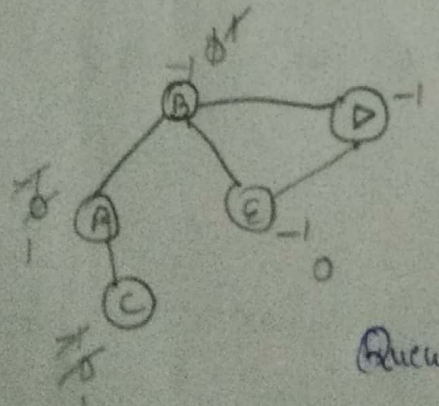
Ans Dense graph is a graph in which the no of edge is close to the maximal no of edges.

Sparse graph is a graph in which the number of edges is close to the minimal no of edges. It can be disconnected graph.

Adjacency List are preferred for sparse graph & adjacency matrix for dense graph.

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

Ans Cycle detection in undirected graph (BFS)



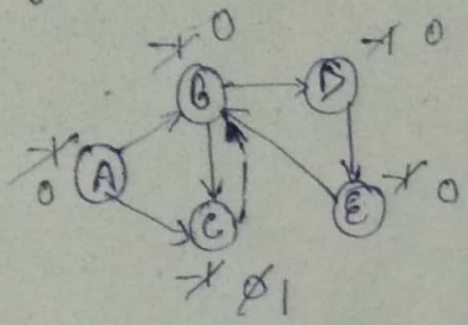
-1 = Unvisited
0 = Inserted into queue
1 = Explored

Queue : | A | B | C | D | E |

Visited : | A | B | C | D |

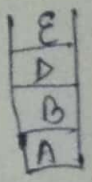
When D checks its adjacent vertices, it finds E with 0.
If any vertex finds the adjacent vertex with flag 0, it contains cycle.

Cycle detection in Directed Graph (DFS)



-1 – Unvisited
0 – visited & in stack
1 – visited & popped out from stack

Stack:



Visited Set
A B C D

Parent Map:

Vertex	Parent
A	—
B	A
C	B
D	B
E	D

B → D → E → B

Here from E we reach B again with 0,
⇒ It contains cycle

Ans 5:

Disjoint set data structure is also known as union find data structure & merge find set. It is a data structure that contains a collection of disjoint or non overlapping sets.
The disjoin set means that when the set is partitioned into the disjoint subsets, various operations can be performed on it.
In this case, we can add new sets, we can merge the sets, & we can also find the representative no of a set.
It allows us to find out whether two elements are in

the same set or not efficiently.

Operations on disjoint set:

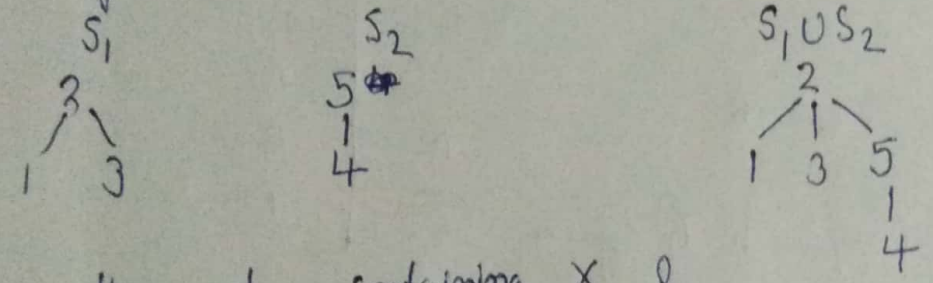
1) Union →

(a) If S1 & S2 are 2 disjoint sets, their union $S_1 \cup S_2$ is a set of all elements x such that x is in either S_1 or S_2.

(b) As the sets should be disjoint, $S_1 \cup S_2$ replaces S_1 & S_2 which no longer exists.

(c) Union is achieved by simply making one of the trees as a subtree of other ie to set parent yield of one of the roots of the trees to other root.

eg

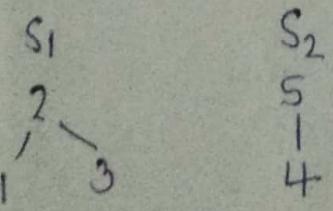


Merge the sets containing X & containing Y into one

2) Find →

Given an element X, to find the set containing it.

ex

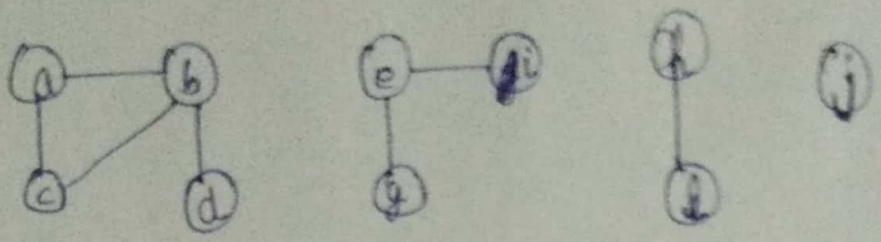


find (3) ⇒ S_1 return in which set X belongs

find (5) ⇒ S_2

3) Make Set (X) → Create a set containing X

Q7>



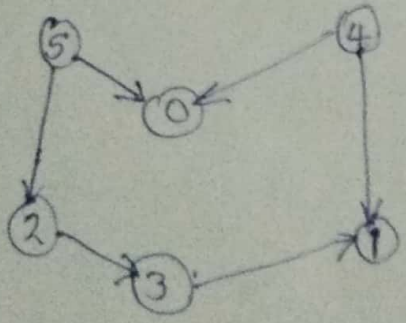
$V = \{a, b, c, d, e, g, h, i, j, k\}$

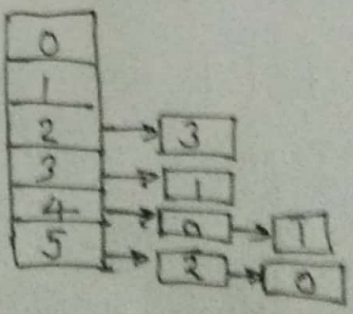
$E = \{(a,b), (a,c), (b,c), (b,d), (e,i), (e,g), (h,k), (j)\}$

	{a}	{b}	{c}	{d}	{e}	{g}	{h}	{i}	{j}	{k}
(a,b)	{a,b}		{c}	{d}	{e}	{g}	{h}	{i}	{j}	{k}
(a,c)	{a,b,c}		{d}	{e}	{g}	{h}	{i}	{j}		{k}
(b,d)	{a,b,c,d}		{e}	{g}	{h}	{i}	{j}			{k}
(e,i)	{a,b,c,d}		{e,i}	{g}	{h}		{j}			{k}
(e,g)	{a,b,c,d}		{e,i,g}		{h}		{k}			{j}
(h,k)	{a,b,c,d}		{e,i,g}		{h,k}					{j}
(j)	{a,b,c,d}		{e,i,g}		{h,k}					{j}

We have {a,b,c,d}, {e,i,g}, {h,k} & {j}

Q8:>





Algo:

1> Go to node 0, it has no outgoing edges so push node 0 into the stack & make it visited.

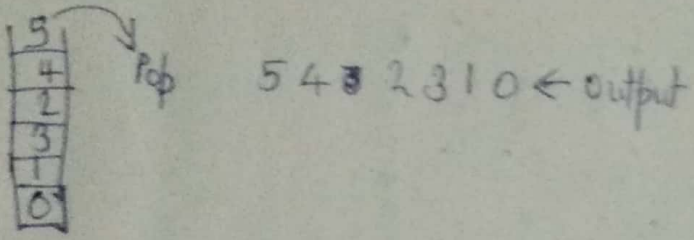
2.) Go to node 1 again it has no outgoing edges, so push node 1 onto stack & mark it visited

3.) Go to node 2, process all the adjacent nodes & mark node 2 visited

4.) Node 3 is already visited so continue with next node.

5.) Go to node 4, all its adjacent nodes are already visited so push node 4 into the stack & mark it visited.

6.) Go to node 5, all its adjacent nodes are already visited so push node 5 onto the stack & mark it visited.



Ans 9.)

Heap is generally preferred for priority queue implementation because heap provide better performance compared to arrays or linked list.

Algorithms where priority queue is used :

1 Dijkstra's Shortest Path Algorithm →

When the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijkstra's algorithm.

2 Prim's Algorithm →

To store keys of nodes & extract minimum key node at every step

Min Heap	Max Heap
1.) For every pair of the parent and descendant child node, the parent node always has lower value than child node	1.) For every pair of parent & child node, the parent node has greater value than child node.
2.) The value of nodes increases as we traverse from root to leaf node.	2.) The value of node decreases as we traverse from root to leaf node.
3.) Root node has lowest value	3.) Root node has greatest value