Tutorial-5

Ques1

BFS

* uses queue data structure
* Stands for Breadth first search
* Can be used to find single source shortest path in a unweighted graph if we reach a vertex with min no. of edges from a source vertex.
* Siblings are visited before the children

Applications:

* Shortest path & minimum spanning tree for unweighted graph
* peer to peer networks
* Social network websites
* GPS navigation system

DFS

* uses stack data structure stands for depth first search.
* we might traverse through more edges to reach a destination vertex from a source.
* children are visited before the siblings.

Applications:

* detecting cycle in a graph.
* path finding
* Topological sorting
* Solving puzzles with only one soln.

Ques 2

In BFS we use queue data structure as queue is used when things don't have to be processed immediately, but have to be processed in FIFO order like BFS.

In DFS stack is used as DFS uses backtracking. for DFS, we retrieve it from root to the farthest node as much as possible, this is the same idea as LIFO [used by stack]

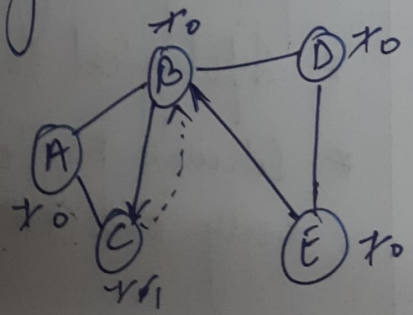
Ques 3

Dense graph is a graph in which the no. of edges is close to the maximal no. of edges.
Sparse graph is a graph in which the no. of edges ~~it can be disconnected or~~ is close to the minimal no. of edges. It can be disconnected graph.

* adjancy list are prefered for sparse graph & Adjancy matrix for dence graph

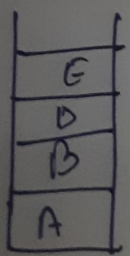
Ques 4

Cycle detection in ~~undirected~~ graph (BFS)



-1 = unvisited
0 = visited & in stack
1 = visited & popped out from stack

Stack



Visited set:

ABCDE

B → D → E → B

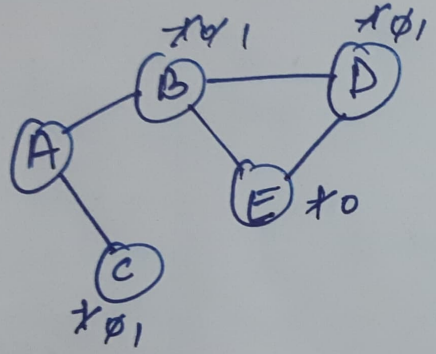
Parent map

Vertex	Parent
A	–
B	A
C	B
D	B
E	D

thus E finds B (adjacency vertex of E) with DO

⇒ it contains a cycle

Cycle detection in undirected graph



1 = unvisited
0 = put into the queue (nodes)
1 = traversed.

queue: | A | B | C | D | E |

Visited Set: | A | B | C | D |

when D checks it adjacent vertex with flag 0, the it contain cycle.

Ques 5

The disjoint set data structure is also known as union find data structure & merge-find set. It is a data structure that contains a coll^{n} of disjoint ~~set means~~ that or non-overlapping sets. The disjoint set means that when the set is partitioned into the disjoint subsets, various op^{n} can be performed on it.

In this case, we can add new sets, we can merge the sets & we can also find the representation member of set. It also allows to find out whether the two elements are in the same set or not efficiently.

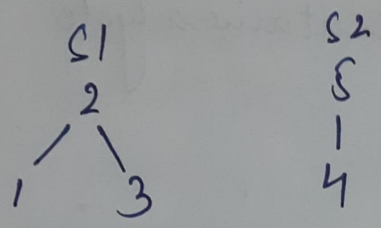
operations on disjoints set are :-

1. Union

* If S1 & S2 are two disjoints sets, their union S1 ∪ S2 is a set of all elements x such that x is in either S1 or S2.
* As the set should be disjoint S1 ∪ S2 replaces

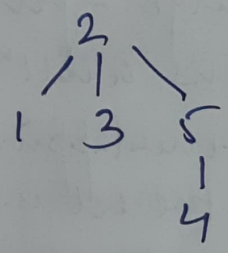
S1 & S2 which no longer exists.

* union is achieved by simply making one of the tree as a subtree of other i.e. to set parent field of one of the roots of the tree to other root.



merge the sets containing x & containing y into one

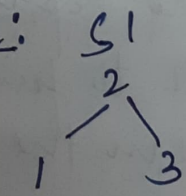
S1 ∪ S2

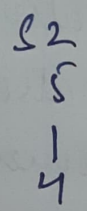


2. find

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

Example:





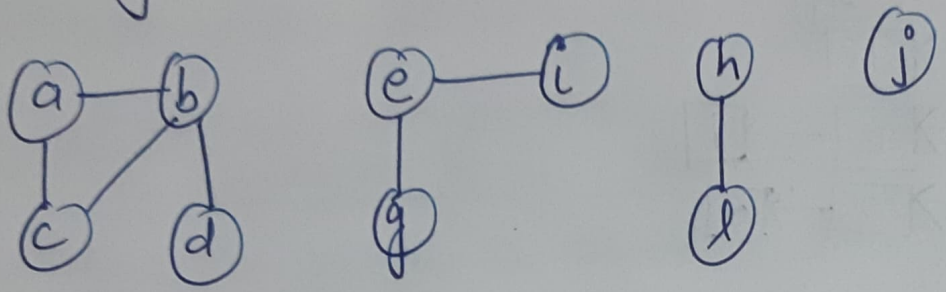
find (3) => S1

find (5) => S2

return in which set x belongs

3. make-set (x): creat a set containing x

Ques 7



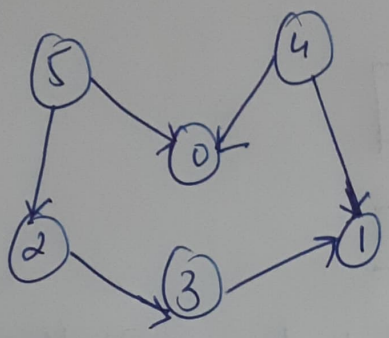
$V = \{a, b, c, d, e, g, h, i, j, l\}$

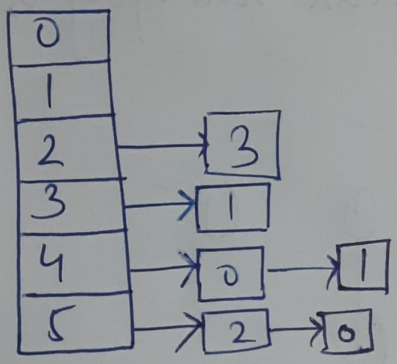
$E = \{(a,b), (a,c), (b,c), (b,d), (e,i), (e,g), (h,l), (j)\}$

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

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

Ques 8

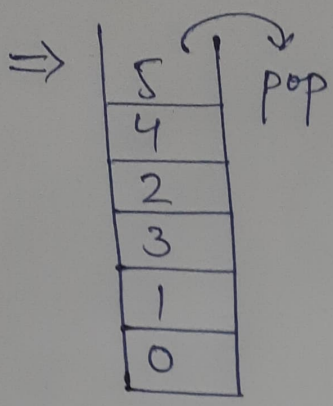




Algo:

* Go to node 0, it has no outgoing edges so push node 0 into the stack & mark it visited
* Go to node 1, again it has no outgoing edges, so push node 1 into the stack & mark it visited
* Go to node 2, process all the adjacent nodes & mark node 2 visited
* node 3 is already visited so continue with next time node
* Go to node 4, all its adjacent nodes are already visited so push node 4 into the stack & mark it visited
* Go to node 5, all ~~the~~ its adjacent nodes are already

Visited so push node 5 into the stack & mark it visited.



5 4 2 3 1 0 output

Ans9

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

Algorithms where priority queue is used:

1. Dijkstra's ~~where priority queue is used:~~ Shortest path Algorithm:
   where 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 key of nodes & extract minimum key node at every step.

Min heap	Max heap
* for every pair of the parent & descendant child node, the parent node always has lower value than descended child node. * The value of nodes inc. as we traverse from root to leaf node. * root node has lowest value.	* for every pair of parent & descendant child node, the parent node has greater value than descended child node. * The value of nodes dec. as we traverse from root to leaf node. * root node has greatest value.