DIVYANSH SHARMA
CST CORE, Rollno 46
2017644.

DESIGN & ANALYSIS OF ALGORITHMS
TUTORIAL 3

Q1).

BFS	DFS
*) stands for "Breadth first search"	stands for "depth first search"
*) Uses queue data structure	Uses stack data structure
*) suitable for searching vertices which are closer to given source	suitable for searching vertices away from the given source
*) we reach a vertex with min no of edges from a source vertex	we might traverse more edges to reach a destination vertex from source.

APPLICATIONS:

BFS: shortest path & MST for unweighted graph
GPS navigation systems.
Cycle detection in undirected graph.

DFS: Topological sort
Cycle detection in graphs
Solving puzzles with only one solution.

Q2). The BFS algorithm traverses a graph in breadth ward motion. Thus it uses a queue (fifo) to remember to get the next vertex to start a search iteration if and when ~~as~~ a dead end occurs in any situation.

The DFS algorithm traverse in depth ward motion. Thus it visits nodes of an entire branch of tree before traversing to adjacent nodes. So to store a keep track of current node the ~~data~~ Lifo data structure is implemented. After it reaches the depth of node then all nodes are popped out of the stack. Only now it begins searching ~~for~~ in the adjacent nodes that havent been visited yet.

Q3) Sparse Graph: when a graph contains less edges than it possibly can it is termed as a sparse graph

Dense Graph: A graph in which the no of edges is closer/almost equal to the possible number of edges is termed as a dense graph

for sparse graphs, Adjacenty lists ~~a matrix~~ representation is preferred
for dense graphs Adjacency matrix representation is preferred.

Q4) Disjoint set: Also known as union-find or merge-find set, this data structure contains a collection of disjoint or non overlapping sets. The disjoint set means that when partitioned, various other operations can be performed on those subsets.

The following are among many operations that can be performed on disjoint sets.

→ Making new sets:

```
function Makeset (x) is
    if x is not already in the forest then
    x.parent = x
    x.size = 1
    x.rank = 0
  end if
  end function
```

⇒ finding set representatives.

```
function Find (x) is
if   x.parent ≠ x then
     x.parent = Find (x.parent)
     return x.parent
else
      return x
     end if
end function
```

→ merge 2 sets:

```
function union (x,y) is
  x = Find (x)
  y = Find (y)
if  x = y  then
   return
end if
if x.size < y.size then
(x,y) = (y,x)
end if
y.parent = x
x.size = x.size + y.size
end function
```

2

Q5). Detection using BFS in an undireected graph.

→ for every visited vertex 'v', if theres an adjacent 'u' such that u is already visited and u is not a parent of v, then there is a cycle in the graph.

→ If we dont find such an adjacent for any vertex, we can say that there is no cycle.

→ We use a parent array to keep track of the parent vertex for a vertex so that we dont consider the visited parent as part of cycle.

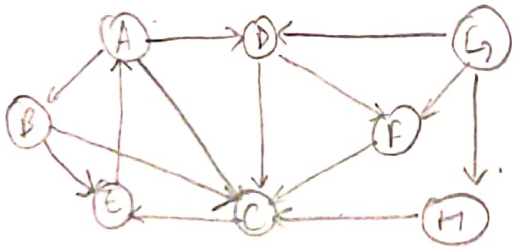
Using DFS in undireeted graphs-

→ Run DFS traversal from every unvisited node. DFS for a connected graph produced a tree. There is a cycle in graph only if theres a back edge present in graph.

→ A back edge is an edge joining a node to itself (self loop) or one of its ancestors in tree produced from DFS

→ To find the back edge to any of its ancestors keep a visited array & if theres a back edge to any visited node then there is a loop & return true.

Q6)



BFS :

Node :	B	E	C	A	D	F
Parent :	–	B	B	E	A	D

unvisited : G & H.

∴ Path = B → E → A → D → F

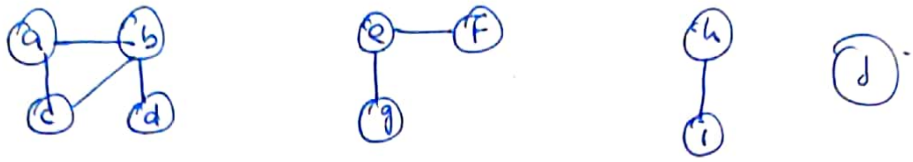
DFS :

node :	B	B	C	E	A	D	F
stack :	B	CE	EE	AE	DE	FE	E

∴ Path : B → C → E → A → D → F

3

Q7)



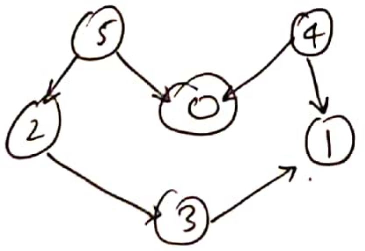
V = {a} {b}, {c} {d} {e} {f} {g} {h} {i} {j}

E = {a,b} {a,c} {c,b} {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} {hi} {j}.

$\boxed{\text{no of connected components} = 3}$

Q8).



Adjacent list

0 →

1 →

2 → 3

3 → 1

4 → 0,1

5 → 2,0.

visited :-

f	f	f	f	f	f.
0	1	2	3	4	5

stack (empty).

step 1: Topological sort (0) visited [0] = true

list is empty ∴ no more recursion call stack |0|

step 2: T.S (1) visited [1] = true

list is empty ∴ no more recursion call stack |0|1|

Step 3: T.S (2), visited [2] = true

↓

T.S (3), visited [3] = true

'1' is already visited, ∴ no more recursion call stack.

0	1	3	2

Step 4: T.S (4) visited [4] = true

'0', '1' are already visited no more recursion call stack

0	1	3	2	4

Step 5: T.S (5), visited [5] = true

↓

'2', '0' are already visited. no more recursion call stack.

0	1	3	2	4	5

Step 6: Print all elements of stack from top to bottom

5, 4, 2, 3, 1, 0

Q9). We can use heaps to implement priority queue. It takes O(log N) time to insert & delete each element in priority queues. Heaps are ideal for implementing a priority queue due to the largest & smallest element at tree root for a max heap & min heap respectively.

Few graph algorithms -

⇒ Dijkstra's: when stored in form of adjacency list, priority queue is implemented on the graph to extract minimum.

⇒ Prim's: To store of keys of nodes & extract minimum key node at every step of the algorithm

⇒ Heap sort: typically implemented using Heap which is an implementation of Priority queue.

5

10).

MAX HEAP	MIN HEAP.
*) Uses descending priority	Uses Ascending Priority
*) Thus, the largest element has priority during construction	Thus, the smallest element has priority during construction.
*). The key present at root node must be greater than /equal to keys present at its children	key present at root node must be less than / equal to among the keys at all of its children.
*). root contains the maximum key element	root contains the minimum key element.

x —— x —— x.

6