Adesh Sengar
I, 14

Ques1 Tutorial 5

| BFS | DFS |
|---|---|
| • Breadth first search | • Depth first search |
| • BFS uses queue to find the shortest path | • It uses stack to find the shortest path. |
| • BFS is better when target is closer to source | • DFS is better when target is far from source |
| • As BFS considers all neighbours so it is not suitable for decision | • DFS is more suitable for decision tree. As with one decision we need to traverse further to argument the decision. If we search the conclusion. |
| • BFS is slower than DFS | |

Application of DFS

- Using DFS we can find path b/w two vertices.
- we can perform topological sorting which is used to scheduling ~~jobs~~.
- We can use dfs to ~~detch~~ detect cycles.
- Using DFS, we can find strongly connected components of a graph.

Application of BFS:-

- BFS may also used to detect cycles.
- finding shortest path and minimal spanning tree in unweighted graphs.
- ~~Are~~ networking finding a route for packet transmission.

- Finding a route through GPS navigation system.

Ques2. BFS uses Queue data structure. In BFS you make any node in the graph as source node and start traversing from it. BFS traverses all the nodes in the graph and keeps dropping them as completed. BFS visited an adjacent unvisited node, marks it as done & insert in into Queue.

DFS uses stack data structure because DFS traverse a graph in a depthword motion and uses a stack to remember to get the next vertext to start a search, when a dead end occurs in any iteration.

Ques3 Sparse Graph:- A graph in which the number of edge is much less than the possible no. of edges.

Dense Graph:- A dense graph is a graph in which the no. of edges is close to the maximal no. of edges of edges.

→ If the graph is sparse, we should store it as list of edges.

Alternatively, if a graph is dense, we should store it as a adjacency matrix.

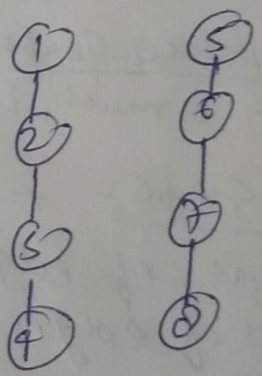
Q.4 DFS can be used to ~~dete~~ detect cycle in a graph. DFS for a connected graph, produces a tree. There is a cycle in a graph only if there is a back edge present in the graph. A back edge is an edge that is ~~form~~ from a node to itself or one of its ancestor in the tree produced by DFS.

BFS can also be used to detect cycles. Just perform BFS while keeping a list of previous nodes at each node visited or else constructing a tree from the starting node.
If I visit a node that is already marked by BFS, I found a cycle.

Ques Disjoint set data structures:-

- It allows to find out whether the two elements are in the same set or not efficiently.
- A disjoint set can be defined as the subsets when there is no common element b/w two sets.

e.g $S_1 = \{1, 2, 3, 4\}$
$S_2 = \{5, 6, 7, 8\}$




Operations performed

(i) find:-

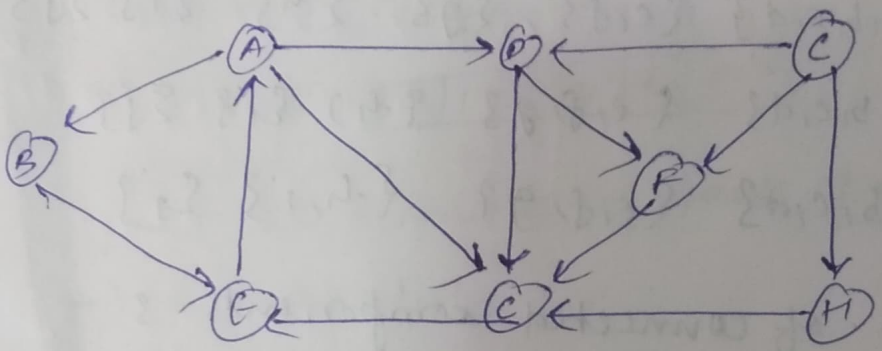
```
int find (int v)
  { if (v == parent[v])
      return v;
    return parent[v] = find (parent[v]);
  }
```

Union:-

```
void union (int a, int b)
{   a = find(a)
    b = find(b)
```

```
if (a != b)
 { if (size[a] < size[b])
     { swap(a, b)
     }
   parent[b] = a;
   size[a] += size[b];
 }
}
```

Que6




BFS :- Node :- B E C A D F

Parent :- – B B E A D

path :- B → E → A → D → F

DFS :

| Node processed | B | B | C | E | A | D | F |
|---|---|---|---|---|---|---|---|
| Stack | B | CE | GE | ME | DE | FE | E |

path :- B → C → E → A → D → F

Que7

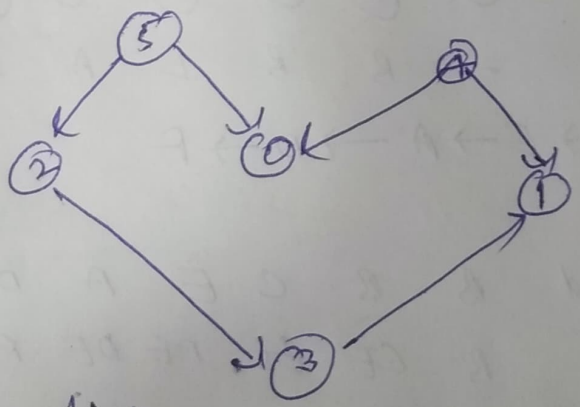
V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

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

Que 8




Adjacency list

0 →

1 →

2 → 3

3 → 1

4 → 0, 1

5 → 2 0

visited

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| false | false | false | false | false | false |

Stack (empty)

Step1:- Topological sort(0), visited[0] = true

Stack | 0 |

Step2:- Topological sort(1), visited[1] = true.

Stack | 0 | 1 |

Step3:- topological sort(2), visited[2] = true

Topological sort(3), visited[3] = true

Stack | 0 | 1 | 3 | 2 |

Step4:- Stack | 0 | 1 | 3 | 2 | 4 |

Step5:- Stack | 0 | 1 | 3 | 2 | 4 | 5 |

Step6:- Print all elements of stack from top to bottom

→ 5, 4, 2, 3, 1, 0

Que 9 Algorithms that uses Priority Queue

(i) Dijkstra's shortest path algorithm using priority queue

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

(ii) Prim's Algorithm:- It is used to implement prims algorithm to store key of nodes to extract minimum key node at every step.

(iii) Data Compression:- It is used in Huffman's code which is used to compress data.

Que 10

| min heap | max heap |
| --- | --- |
| • In min heap the key present at root node must be less than or equal to among the keys present at all its children | • In max heap the key present at root node must be greater or equal to the key present at all its children. |
| • uses the ascending priority | • uses descending priority |
| • The minimum key present at the root node | • The maximum key present at the root node |