Ayush Agarwal
Sec-I
2011030

Tutorial-5

1) Difference btw DFS & BFS. Please write the application of both algorithm.

BFS	DFS
(a) Stands for Breadth First Search	(a) Stands for Depth for search
(b) BFS uses queue to find shortest path.	(b) It uses stack to find shortest path
(c) BFS is better when target is close to source	(c) DFS is better when target is far from source
(d) As BFS considers all neighbour so it is not suitable for decision tree used in puzzle games	(d) DFS is more suitable for Decision tree. As with one decision we need to traverse further to argument the decision. If we search the solution
(e) BFS is slower than DFS	

Application of DFS →

(a) Using DFS we can find path btw two vertices.
(b) We can perform topological sorting which is used to schedule jobs
(c) We can use DFS to detect cycles.
(d) Using DFS we can find strongly connected components of graph

Application of BFS.

(a) BFS may also used to detect cycles.
(b) finding shortest path & minimal spanning tree.
(c) In networking finding a route for packet transmission
(d) finding a route through GPS navigation system

2) Which Data structures are used to implement BFS & DFS & why?

a) BFS uses Queue data structure. In BFS you mark any node in the graph as source node & start traversing from it. BFS traversal all nodes in the graph & keeps dropping them as completed. BFS visited an adjacent unvisited node, mark it as done and insert it into queue.

b) DFS uses stack data structure because DFS traverse a graph in depthward motion & uses a stack to remember to get next vertex & start a search, when a dead end occurs in any iteration.

3) What do you mean by Sparse and dense graph?

Sparse Graph: A graph in which the number of edges 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

b) 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.

4) How to Detect a cycle in graph using BFS & DFS.

a) DFS can be used to detect a cycle in 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 graph. A back edge is an edge that is from a node to itself or one of its ancestors in a tree produced by DFS.

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

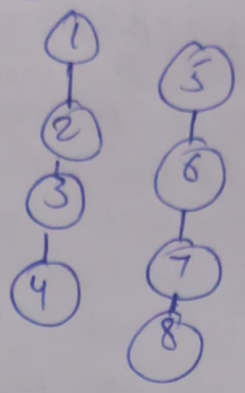
(5) What is Disjoint set data structure.

(a) 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 where there is no common element btw two sets.

Eg $S_1 = \{1, 2, 3, 4\}$

$S_2 = \{5, 6, 7, 8\}$



operation performed

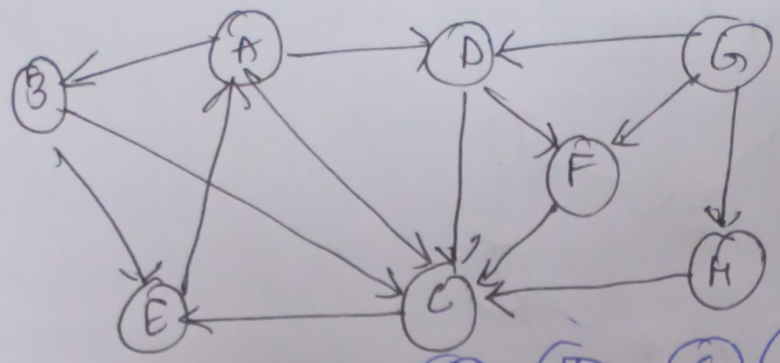
(i) find() -

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

(ii) 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];
  }
}
```

Ques 6 Run BFS & DFS on graph shown on the figure:



BFS ; Node :- B E C A D F

parent :- B B E A D

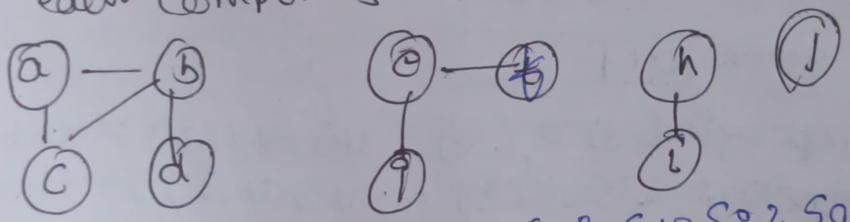
path :- B → E → A → D → F

DFS :- Node :- B B L E A D F

stack /3 CE EE AE DE FE E

path :- B → C → E → A → D → F

(7) Find out number of connected components & vertices in each components using disjoint set data structure



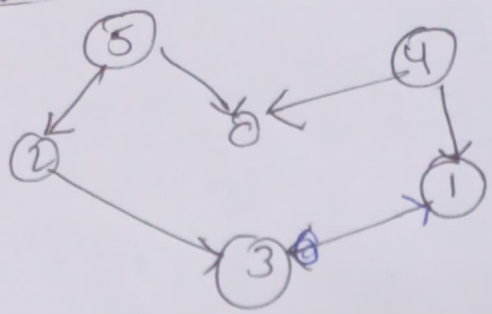
V → {a} {b} {c} {d} {e} {f} {i} {q} {h} {j}

E → {ab} {ac} {bc} {bd} {eg} {ef} {hi} {j}

{ab}	{ab} {c} {d} {e} {f} {q} {h} {i} {j}
{ac}	{abc} {d} {e} {f} {q} {h} {i} {j}
{bc}	{abcd} {e} {f} {q} {h} {i} {j}
{bd}	{abcdq} {ef} {h} {i} {j}
{eg}	{abcdq} {efq} {h} {i} {j}
{ef}	{abcdq} {efq} {h} {i} {j}
{hi}	

No of connected comp = 3

(8) Topological sort



Adjacency list

0 →
1 →
2 → 3
3 → 1
4 → (0, 1)
5 → (2, 0)

Visited

0	1	2	3	4	5
F	F	F	F	F	F

Stack (empty)

Step 1) 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 |

step 4
stak | 0 | 1 | 3 | 2 | 4 |

step5 stack | 0 | 1 | 3 | 2 | 4 | 5 |

step6 print all element of stack from top to bottom

5, 4, 2, 3, 1, 0

(9) Heap data structure can be used to implement priority queue. Name few graphs algo where you can use priority queue.

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

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

(ii) Prim's Algo

(iii) Data compression using Huffman code.

(10) Min Heap	Max Heap
In Min Heap the key present at root node must be less than or equal to among the key present at all (•) Use ascending priority (•) The minimum key present at the root node.	⇒ In Max heap the key present at root node must be greater or equal to key present at all children (•) Use descending priority (•) Max key present at root node.