Depth First Search

Ms. Sasmita Kumari Nayak
Computer Science & Engineering

Depth First Search (DFS)

- It is a graph traversal algorithm.
- Visits all the vertices and edges of G
- Stack data structure is used in the implementation of depth first search.
- The order of the search is down paths and from left to right. The root is examined first; then the left child of the root; then the left child of this node, etc. until a leaf is found. At a leaf, backtrack to the lowest right child and repeat.

Cont…

- Each node have 3 categories
  - unvisited,
  - discovered and
  - complete.
- Vertices go through white, gray and black stages of color.
  - White : initially
  - Gray: when discovered first
  - Black: when finished i.e. the adjacency list of the vertex is completely examined.
- Also records timestamps for each vertex
  - d[v]: when the vertex is first discovered
  - f[v]: when the vertex is finished

Objective and Working Procedure

Objective

- Given a graph, to print all possible paths between source and destination, i.e. from source vertex to destination vertex.

Working procedure

- Initially, all nodes are unvisited. After visiting a node for the first time, it becomes discovered.
- A node is complete if all of its adjacent nodes have been visited.
- Thus, all the adjacent nodes of a complete node are either discovered or complete.

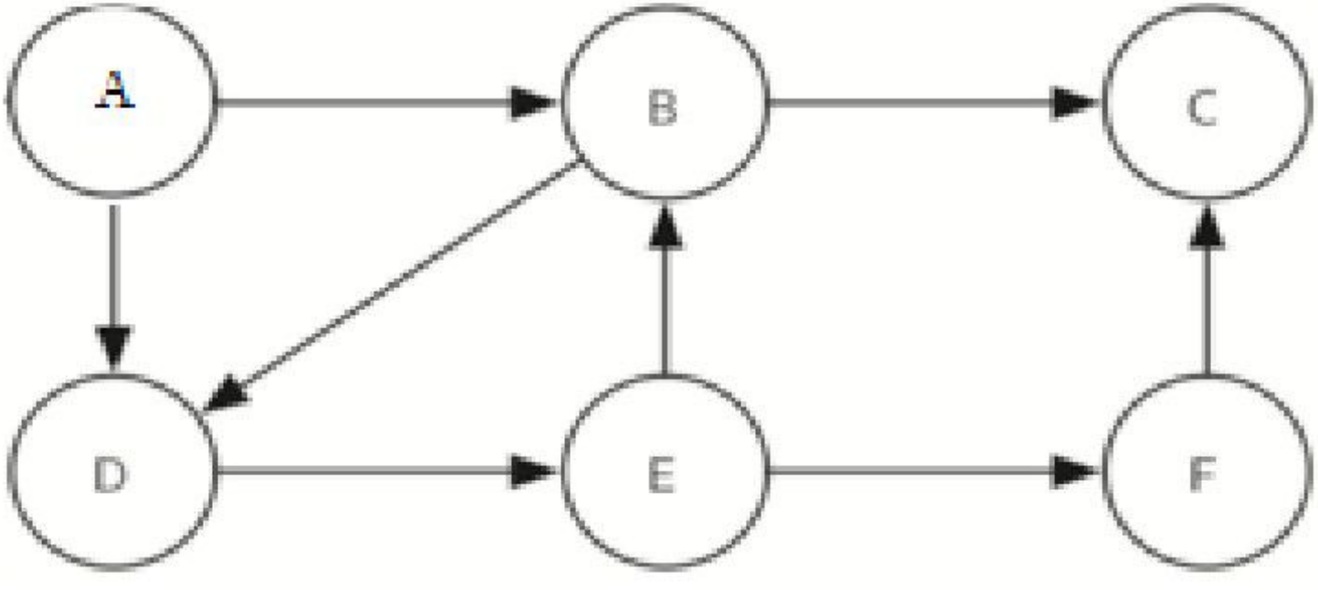
Algorithm

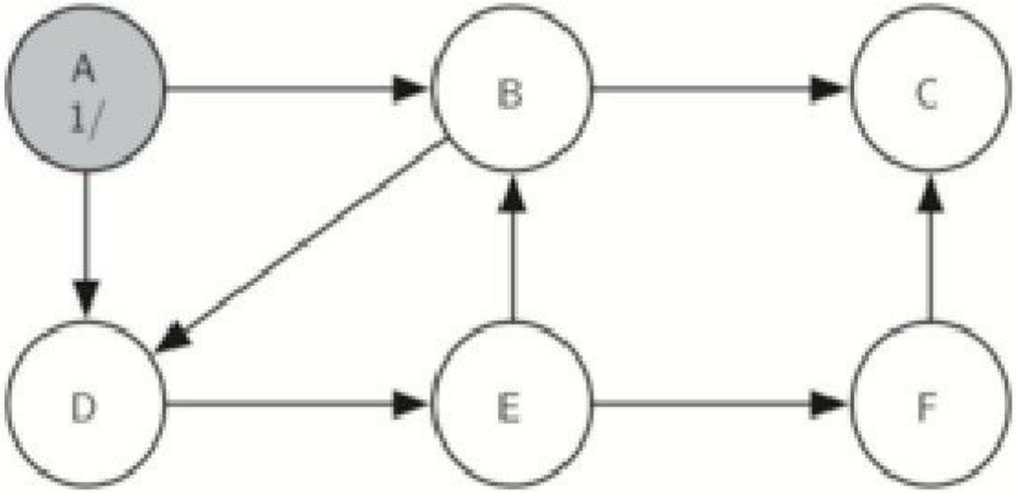
```
DFS(G)
1  for each vertex u ∈ V [G]
2       do color[u] ← WHITE                  // color all vertices white, set their parents NIL
3          π[u] ← NIL
4  time ← 0                                  // zero out time
5  for each vertex u ∈ V [G]                 // call only for unexplored vertices
6       do if color[u] = WHITE               // this may result in multiple sources
7             then DFS-VISIT(u)

DFS-VISIT(u)
1    color[u] ← GRAY                        ▹White vertex u has just been discovered.
2    time ← time +1
3    d[u] ← time                            // record the discovery time
4    for each v ∈ Adj[u]                        ▹Explore edge(u, v).
5       do if color[v] = WHITE
6             then π[v] ← u                 // set the parent value
7                  DFS-VISIT(v)                 // recursive call
8    color[u] ← BLACK                           ▹ Blacken u; it is finished.
9    time ← time +1
10   f [u] ← time
```

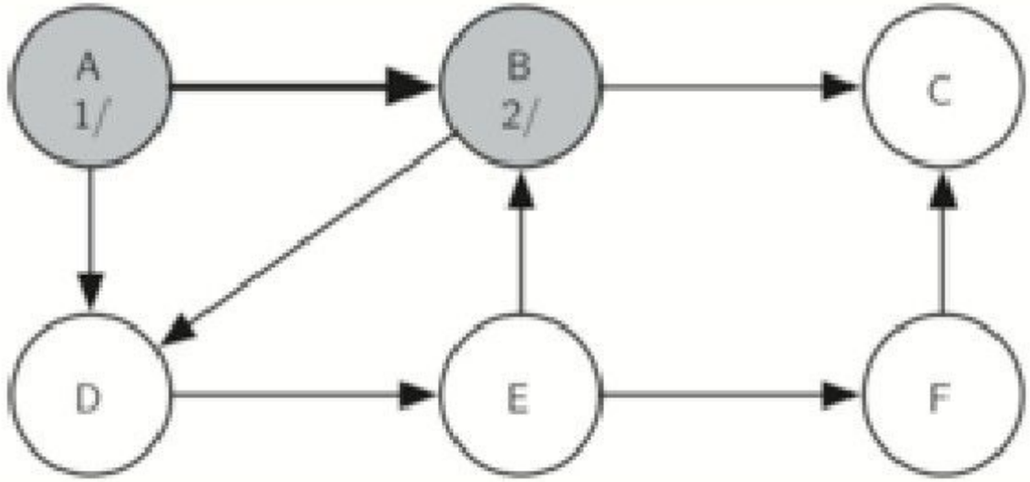
Example

- Use the DFS algorithm to traverse the given graph.

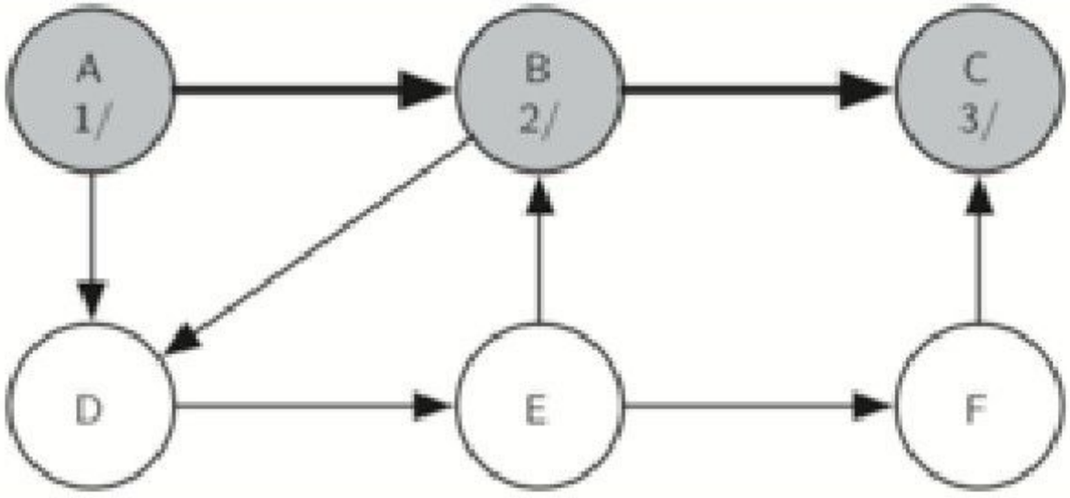




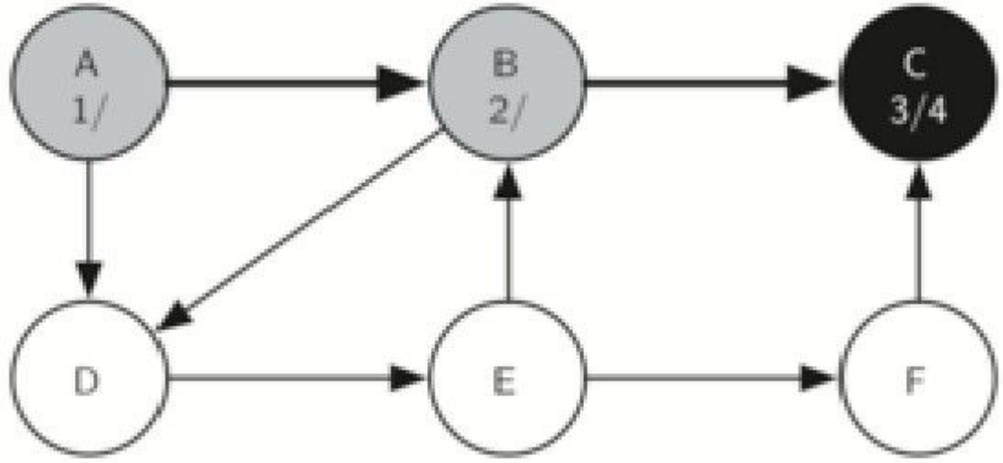
(a)



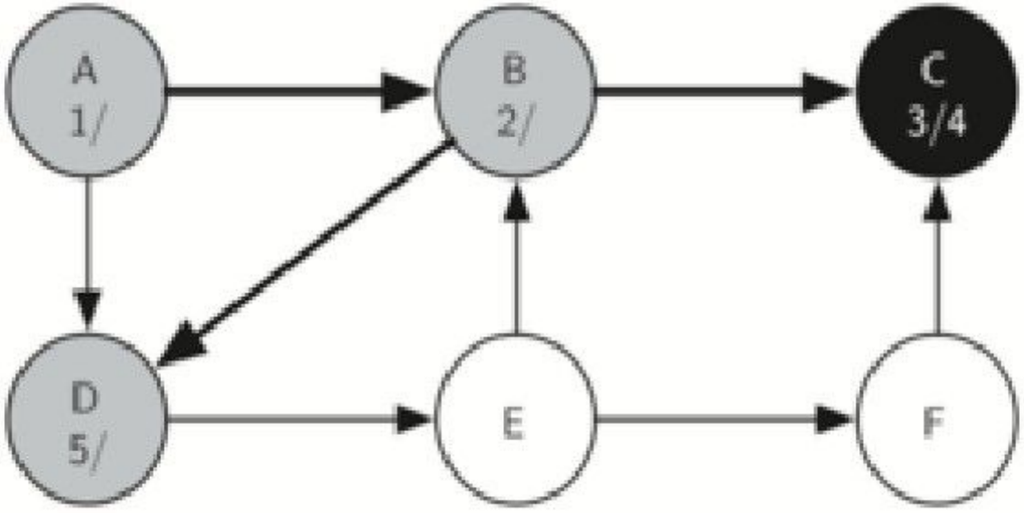
(b)



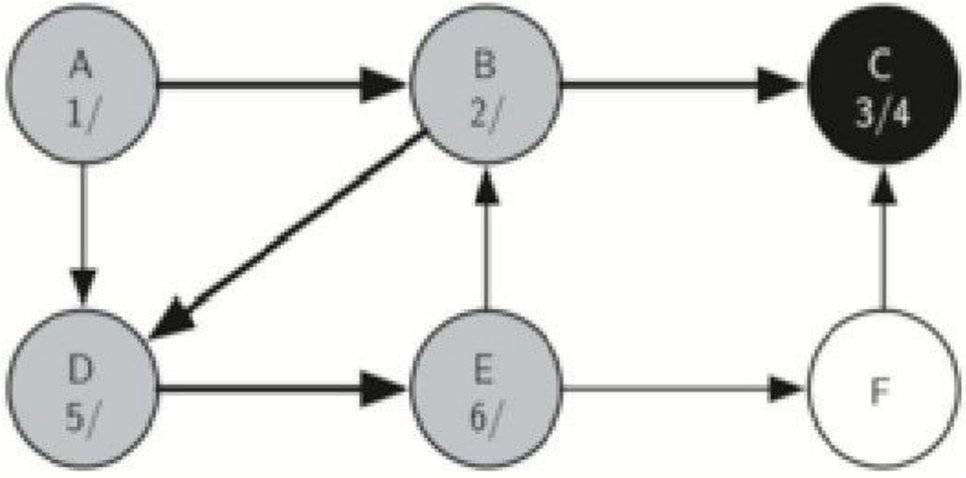
(c)



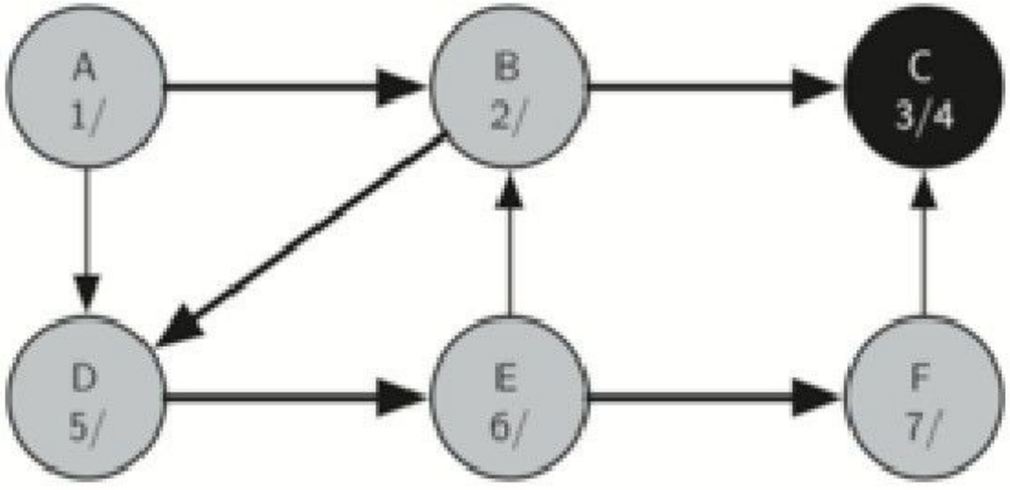
(d)


A
1/
B
2/
C
3/4
D
5/
E
F

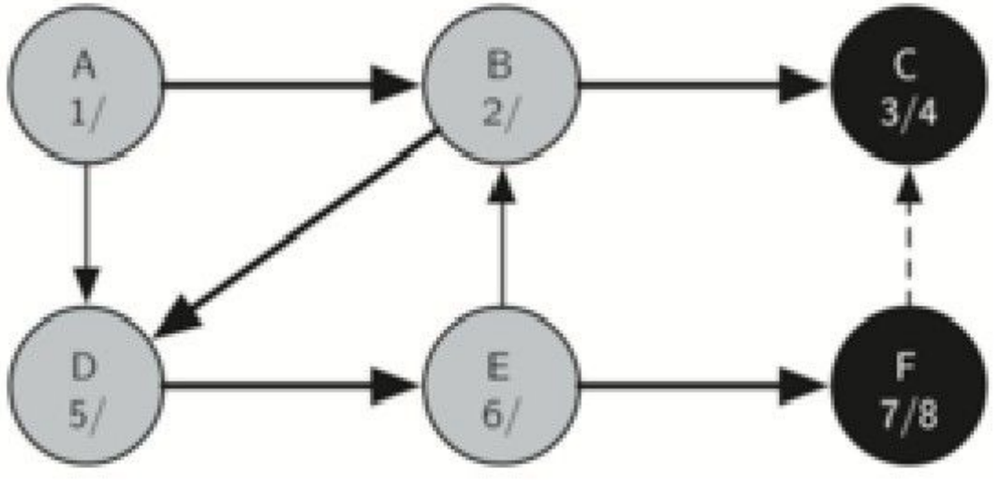
(e)


A
1/
B
2/
C
3/4
D
5/
E
6/
F

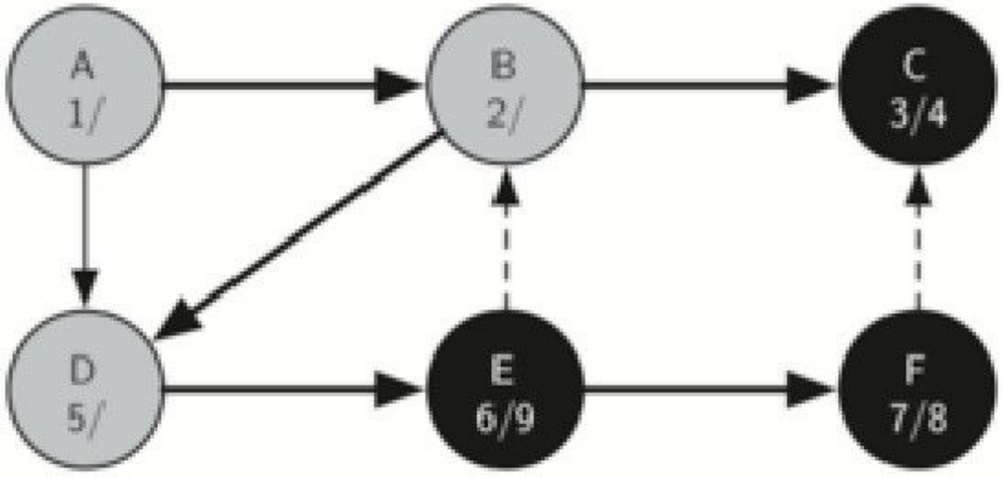
(f)


A
1/
B
2/
C
3/4
D
5/
E
6/
F
7/

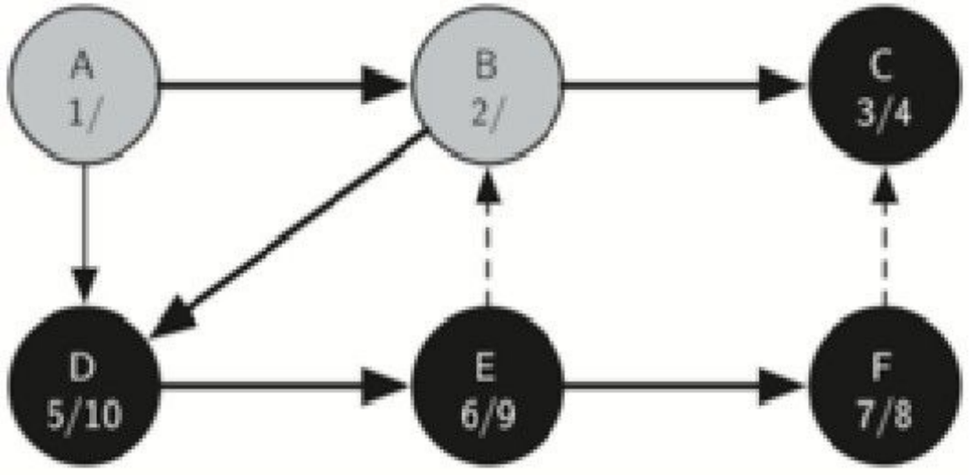
(g)


A
1/
B
2/
C
3/4
D
5/
E
6/
F
7/8

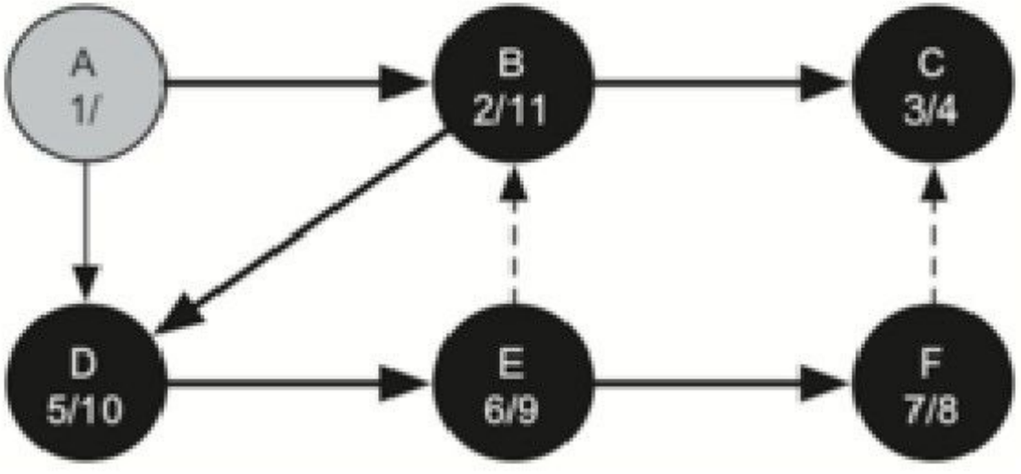
(h)


A
1/
B
2/
C
3/4
D
5/
E
6/9
F
7/8

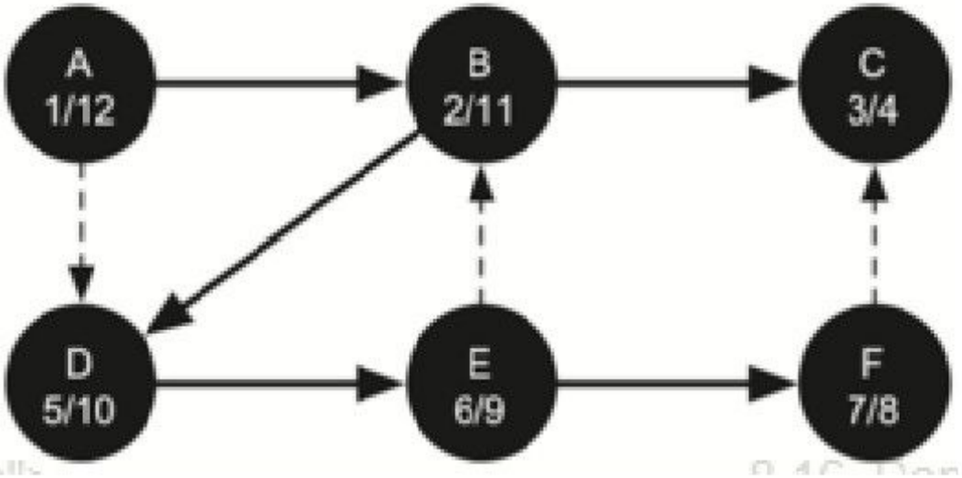
(i)


A
1/
B
2/
C
3/4
D
5/10
E
6/9
F
7/8

(j)


A
1/
B
2/11
C
3/4
D
5/10
E
6/9
F
7/8

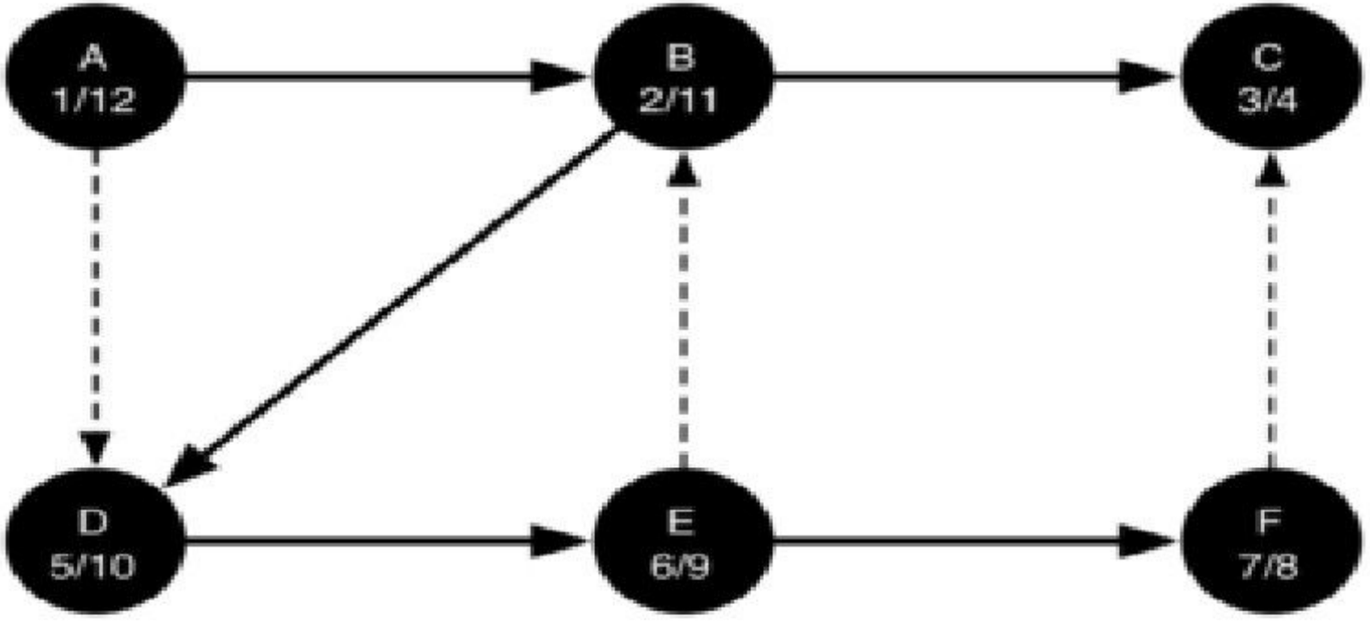
(k)


A
1/12
B
2/11
C
3/4
D
5/10
E
6/9
F
7/8

(l)

Result

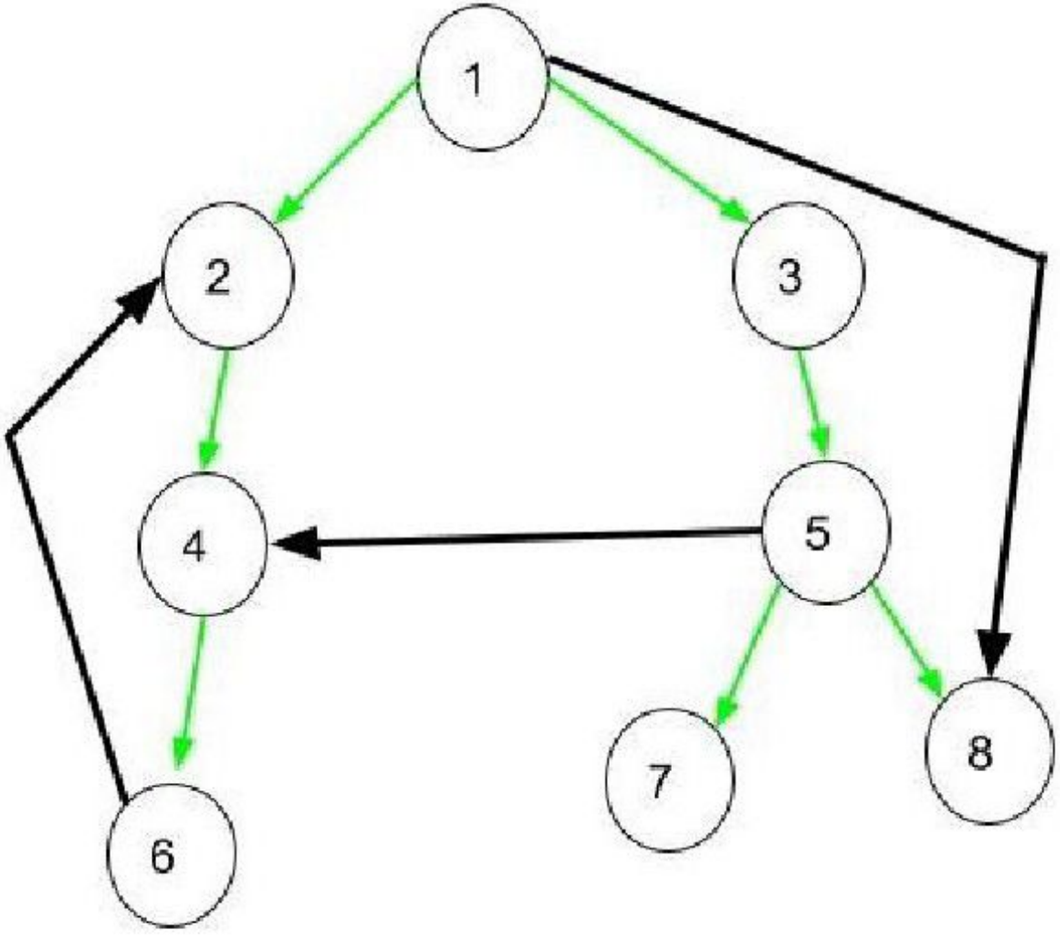
- In DFS, the result will be a tree.



Edges of DFS graph

- **Tree Edge**: It is an edge which is present in the tree obtained after applying DFS on the graph. All the Green edges are tree edges.
- **Forward Edge**: It is an edge (u, v) such that v is descendant but not part of the DFS tree. Edge from **1 to 8** is a forward edge.
- **Back edge**: It is an edge (u, v) such that v is ancestor of edge u but not part of DFS tree. Edge from **6 to 2** is a back edge. Presence of back edge indicates a cycle in directed graph.
- **Cross Edge**: It is a edge which connects two node such that they do not have any ancestor and a descendant relationship between them. Edge from node **5 to 4** is cross edge.

Example



- **Tree Edge**: All the Green edges are tree edges.
- **Forward Edge**: Edge from **1 to 8** is a forward edge.
- **Back edge**: Edge from **6 to 2** is a back edge.
- **Cross Edge**: Edge from node **5 to 4** is cross edge.

DFS Time Complexity

- The total running time for DFS is O (V+E).

Disadvantage

- DFS is not guaranteed to find the solution.
- There is no guarantee to find a minimal solution, if more than one solution exists.

BFS Vs. DFS

Sr. No.	Key	BFS	DFS
1	Definition	BFS, stands for Breadth First Search.	DFS, stands for Depth First Search.
2	Data structure	BFS uses Queue to find the shortest path.	DFS uses Stack to find the shortest path.
3	Source	BFS is better when target is closer to Source.	DFS is better when target is far from source.
4	Suitablity for decision tree	As BFS considers all neighbour so it is not suitable for decision tree used in puzzle games.	DFS is more suitable for decision tree. As with one decision, we need to traverse further to augment the decision. If we reach the conclusion, we won.
5	Speed	BFS is slower than DFS.	DFS is faster than BFS.
6	Time Complexity	Time Complexity of BFS = O(V+E) where V is vertices and E is edges.	Time Complexity of DFS is also O(V+E) where V is vertices and E is edges.