CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Recap → Graph Traversal: BFS Time Complexity analysis

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = Ø
5  s. color = GRAY
6  s. d = 0
7  s. π = Ø
8  Q = NULL
9  ENQUEUE(Q,s)
10 while (Q != NULL)
11    u = DEQUEUE (Q)
12    for each v in G. Adj [u]
13       if v. color == WHITE
14          v. color = GRAY
15          v. d = u.d + 1
16          v. π = u
17          ENQUEUE(Q, v)
18    u.color = BLACK
```

Steps 1-4 are executed "n" times → O (n) → n = |V|= # of vertices

Steps 5-9 are executed once → O (1)

Steps 10-18 :

(i) # of time the while loop executes for DEQUEUE and ENQUEUE → O (n)
(ii) In the for loop, obtaining the adjacency list: # of elements in adjacency list is equal to # of edges → m = |E|
(iii) Steps 13 - 17 are executed in constant time → O (1)
(iv) Steps 18 is executed in constant time → O (1)

Total time complexity $= O(n) + O(n) + O(m)$
$= O(n+m)$
$= O(V+E)$

Graph Traversal: BFS Space Complexity analysis

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = Ø
5  s. color = GRAY
6  s. d = 0
7  s. π = Ø
8  Q = NULL
9  ENQUEUE(Q,s)
10 while (Q != NULL)
11    u = DEQUEUE (Q)
12    for each v in G. Adj [u]
13       if v. color == WHITE
14          v. color = GRAY
15          v. d = u.d + 1
16          v. π = u
17          ENQUEUE(Q, v)
18    u.color = BLACK
```

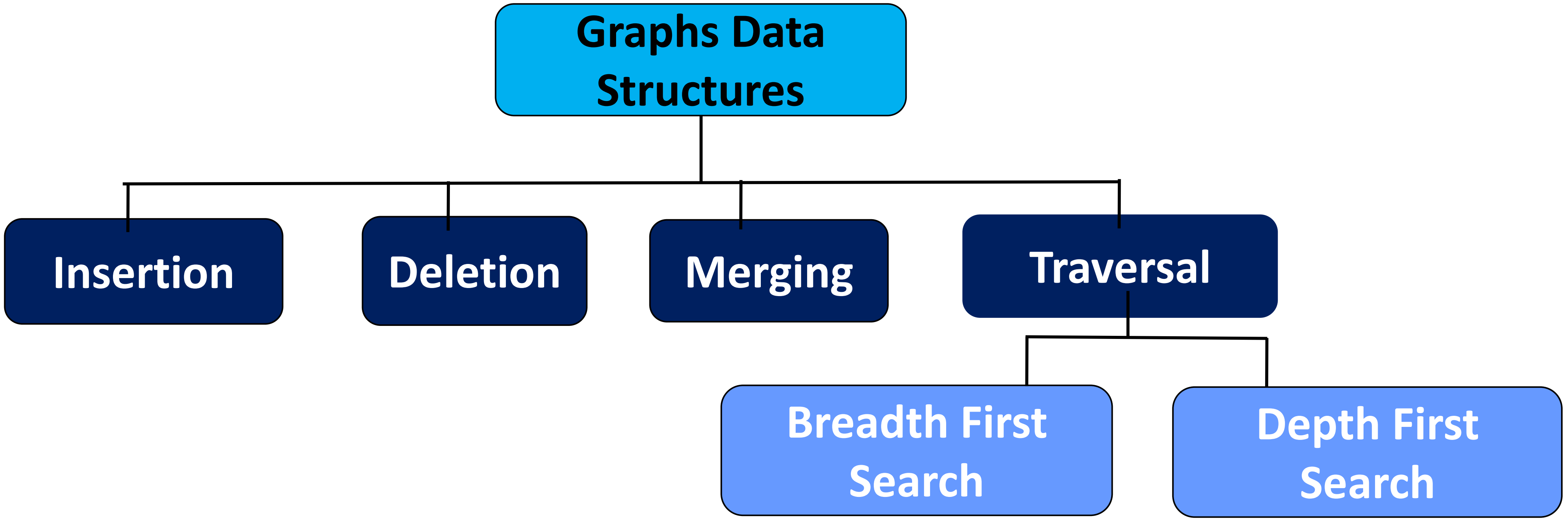
Steps 1-4:
Space required to store the color coding information, predecessor information, and for the Queue → O (n) + O (n) + O (n) → O (n) → n → |V|= # of vertices

Steps 5-9: Constant

Steps 10-18 :

(i) Space required for storing the adjacency list: # of elements in adjacency list is equal to # of edges → m = |E| → O (m)
(ii) If we ignore the space required for adjacency list → O (n)
(iii) If we considered space required for adjacency list → O(n+m)

Graphs: Operations

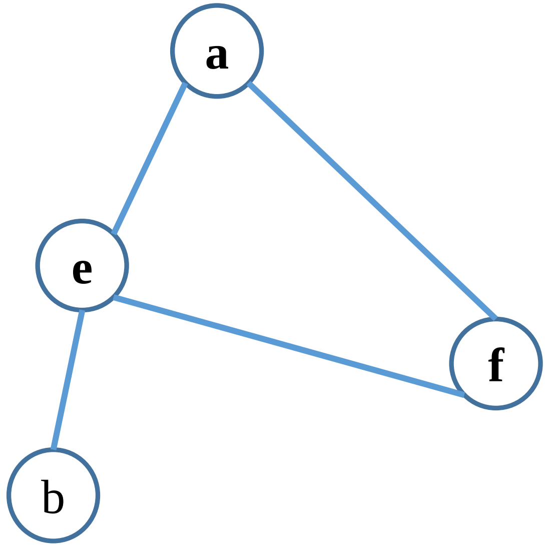


Graph Traversal: DFS

- ❖ DFS traversal → Stack data structures
- ❖ DFS traversal → similar to the preorder traversal of a tree

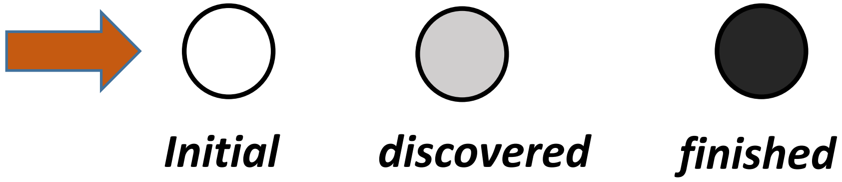
❖ *Procedure*:

- ✓ Initially, DFS starts at a given vertex, visit the adjacent unvisited vertex. Mark it as visited and PUSH into the Stack
- ✓ If no adjacent vertex is found, POP a vertex from the Stack.
- ✓ Repeat this process until the stack is empty



Depth-First Search → DFS

- ❖ DFS is also useful for finding shortest path distance in the graph.
- ❖ DFS forms a *depth-first forest comprising several* depth-first trees
- ❖ The implementation of DFS → Stack data structure.
- ❖ DFS colors vertices during the search to indicate their state



- ❖ DFS follows a ***<u>timestamping</u>*** technique
    - ✓ *v. d timestamp* → vertex search <u>discovered</u> time
    - ✓ *v. f timestamp* → vertex search <u>finished</u> time

Graph Traversal: DFS (1)

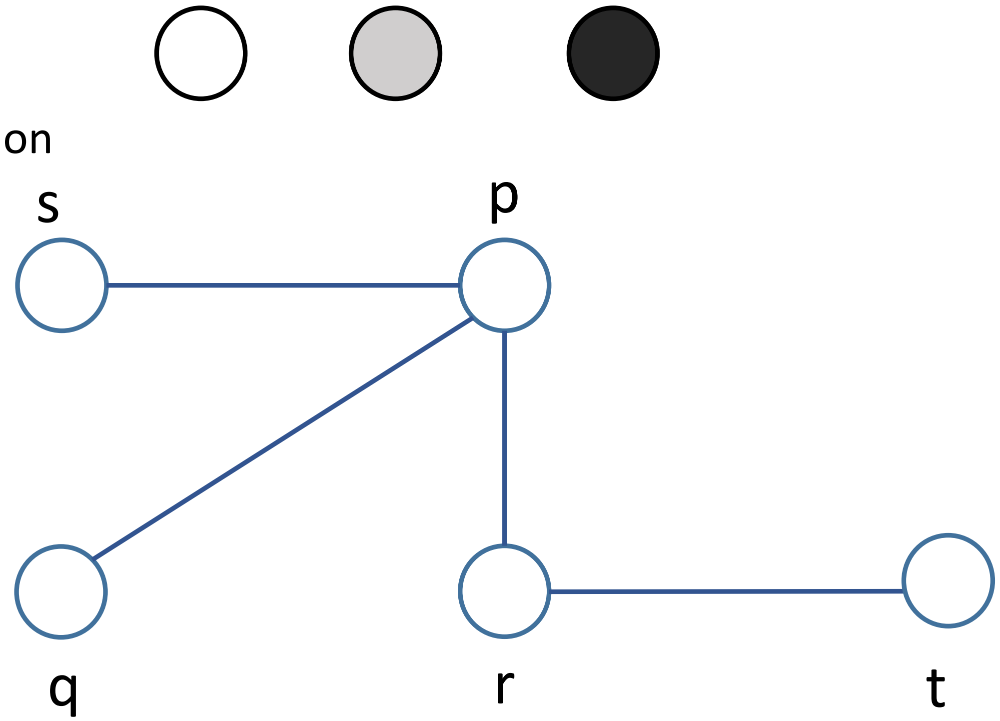
```
DFS(G,s)
1  for each vertex u in G. V
2      u. color = WHITE
3      u. π = ∅
4  time = 0
5  for each vertex u in G. V
        if  u. color == WHITE
            DFS_VISIT (G, u)
DFS_VISIT (G, u)
1  time = time + 1
2  u. d = time
3  u. color = GRAY
4  for each v ∈ G. Adj [u]
5     if  v. color == WHITE
6          v. π = u
7          DFS_VISIT (G, v)
8  u. color = BLACK
9  time = time + 1
10 u. f = time
```

Steps 1-3

Steps 4

Global time counter

Initialization



Graph Traversal: DFS (2)

```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = ∅
time = 0
for each vertex u in G. V
    if  u. color == WHITE
        DFS_VISIT (G, u)
```

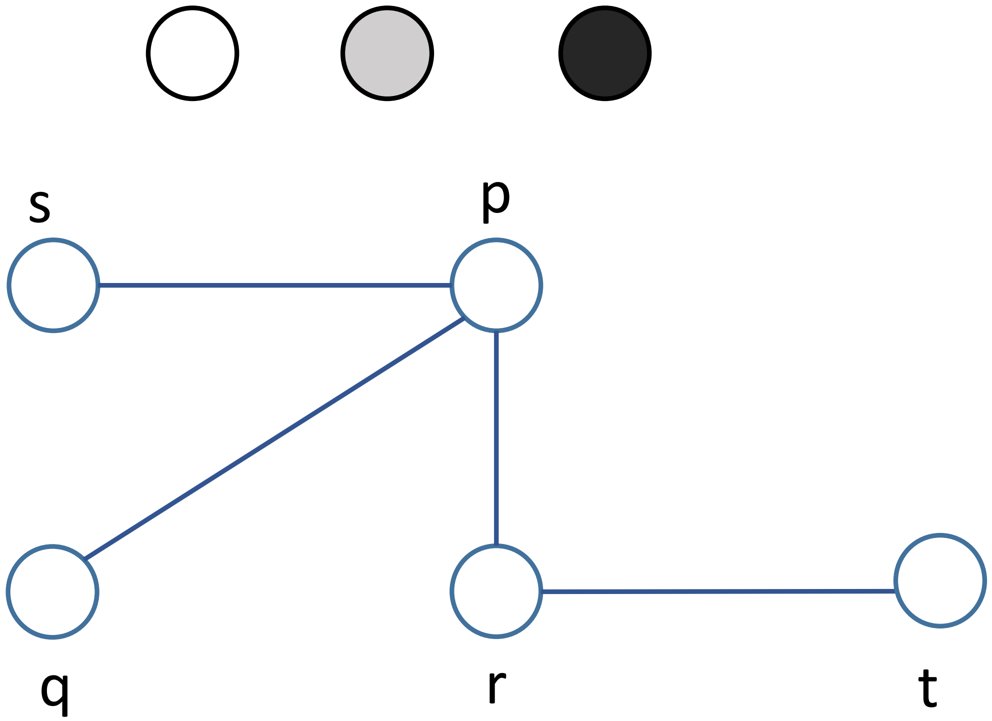
u = s

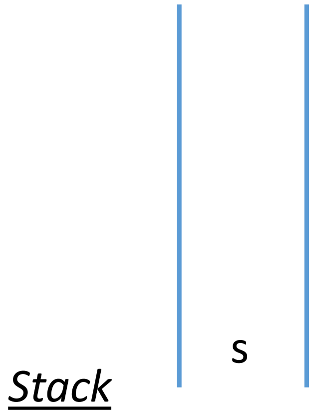
```
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
    if  v. color == WHITE
        v. π = u
        DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

time = 0 + 1 = 1
s. d = 1
s. color = GRAY

v = G. Adj [s]
v = p
p. predecessor = s

DFS_VISIT (G, *p*)





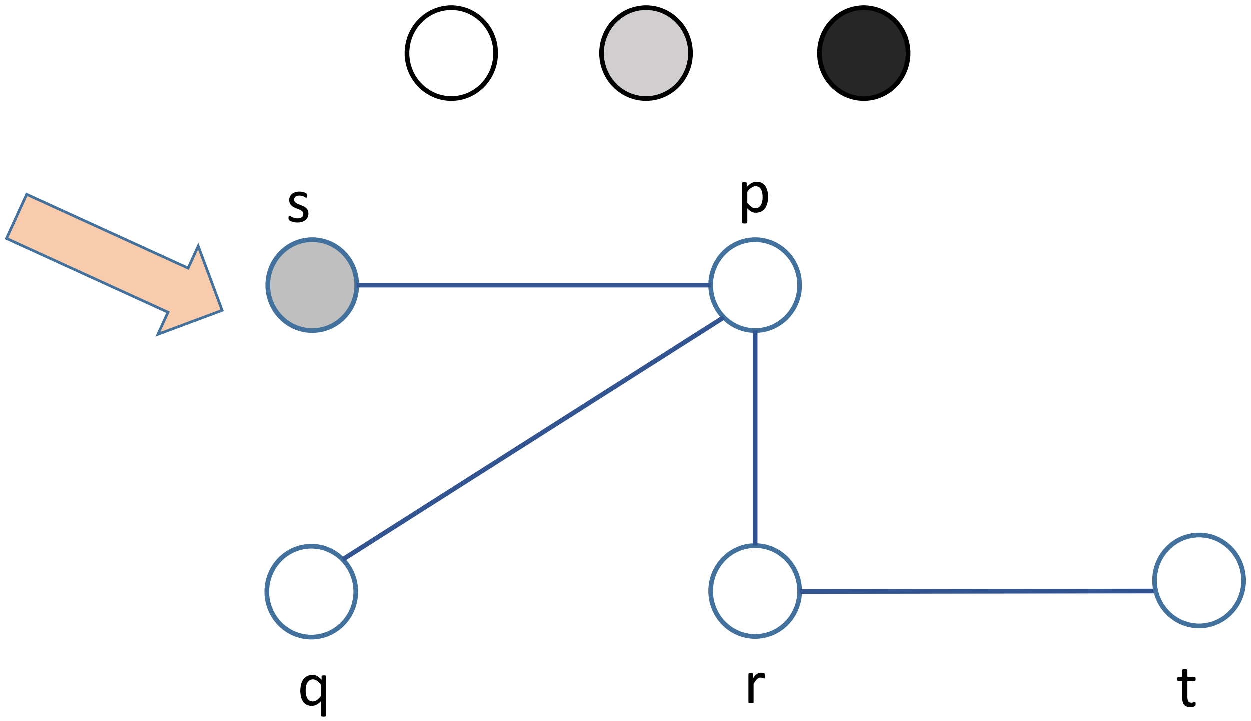
Graph Traversal: DFS (3)

```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = ∅
time = 0
for each vertex u in G. V
    if  u. color == WHITE
        DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
   if  v. color == WHITE
       v. π = u
       DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

time = 1 + 1 = 2
p. d = 2
p. color = GRAY

v = G. Adj [p]
v = q, r
q. predecessor = p

DFS_VISIT (G, q)



p
s

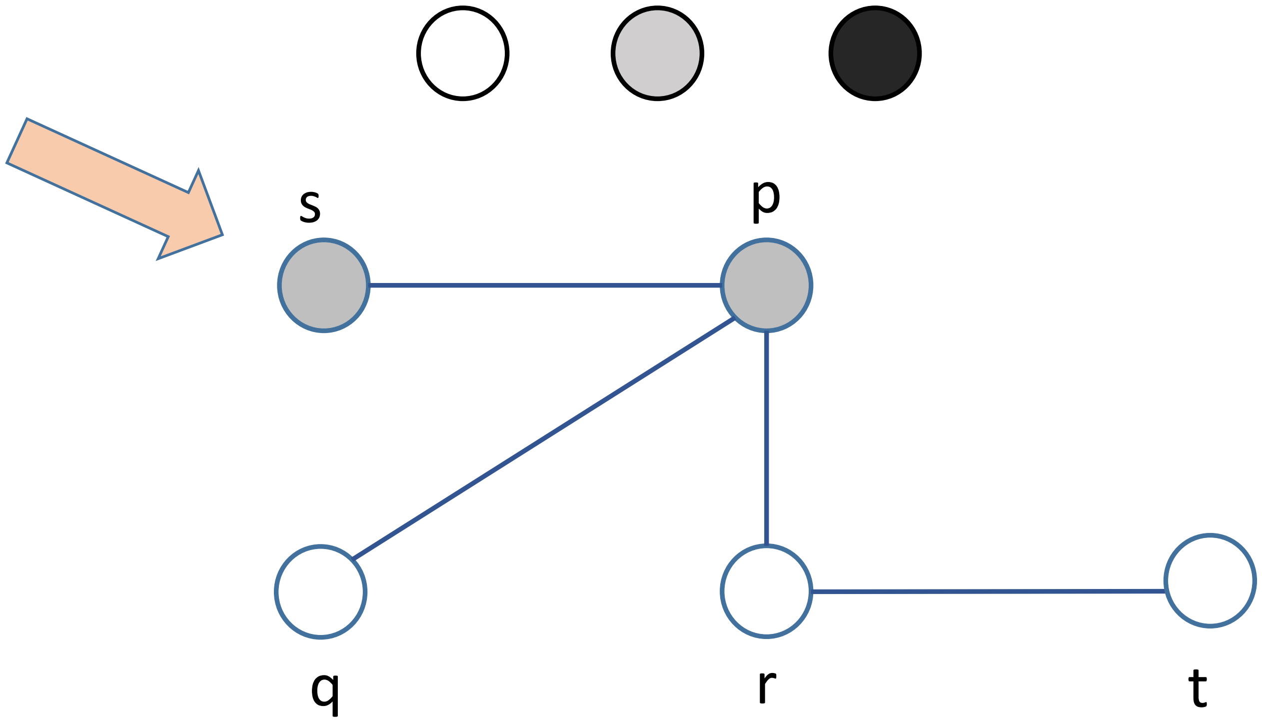
Stack

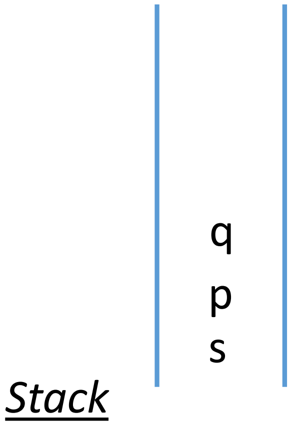
Graph Traversal: DFS (4)

```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = ∅
time = 0
for each vertex u in G. V
      if  u. color == WHITE
           DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
   if  v. color == WHITE
         v. π = u
         DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

time = 2 + 1 = 3
q. d = 3
q. color = GRAY

v = G. Adj [q]
v =





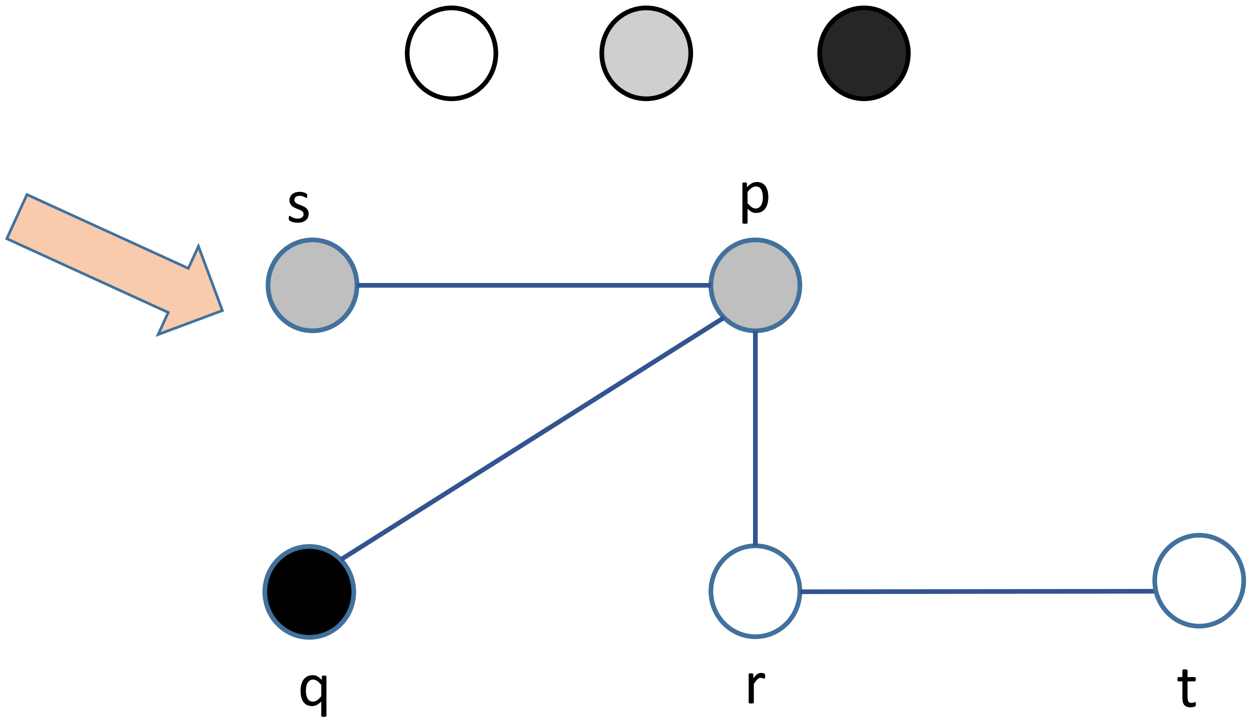
Graph Traversal: DFS (4)

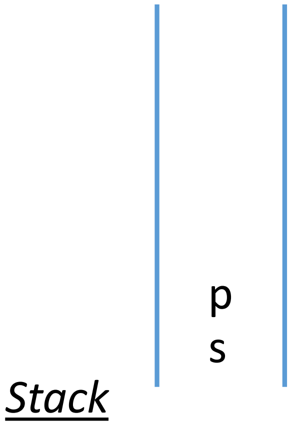
```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = ∅
time = 0
for each vertex u in G. V
     if  u. color == WHITE
          DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
   if  v. color == WHITE
         v. π = u
         DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

time = 2 + 1 = 3
q. d = 3
q. color = GRAY

v = G. Adj [q]
v =

q. color = BLACK
q. time = 3 + 1 = 4
q. f = 4





Graph Traversal: DFS (5)

```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = Ø
time = 0
for each vertex u in G. V
    if  u. color == WHITE
        DFS_VISIT (G, u)
```

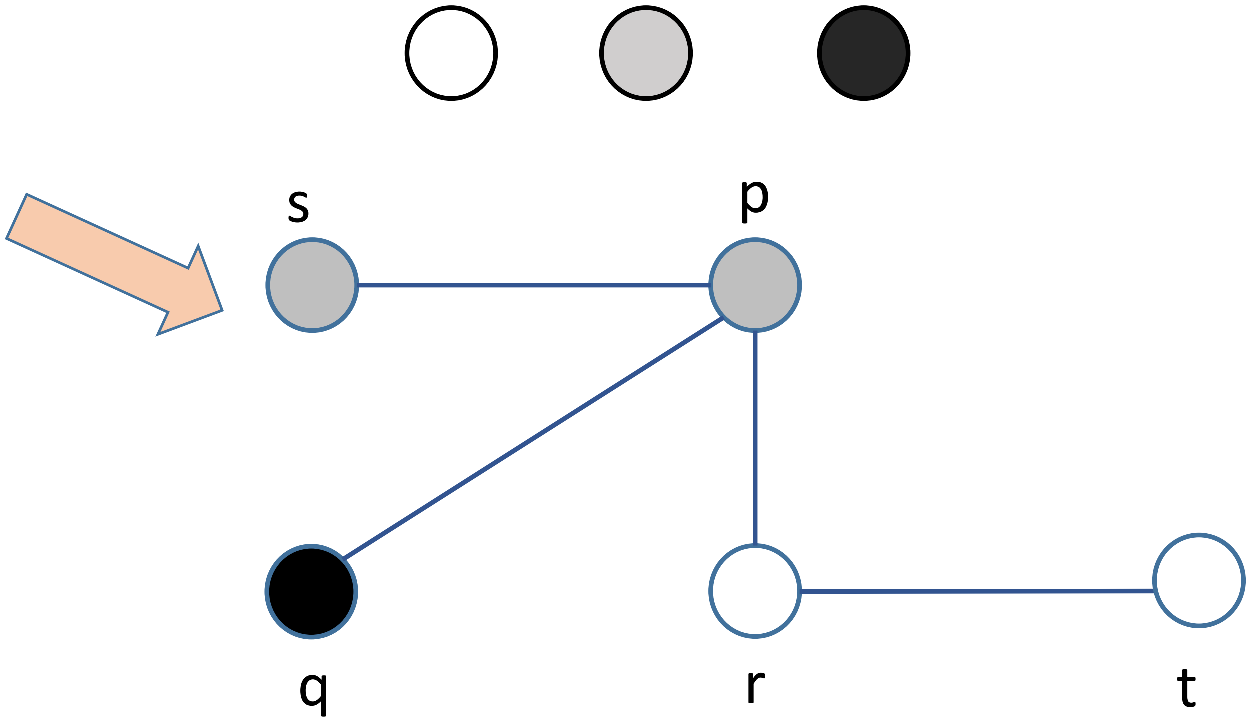
```
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
   if  v. color == WHITE
        v. π = u
        DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

v = G. Adj [p]

v = r

r. predecessor = p

DFS_VISIT (G, r)



r

p

s

Stack

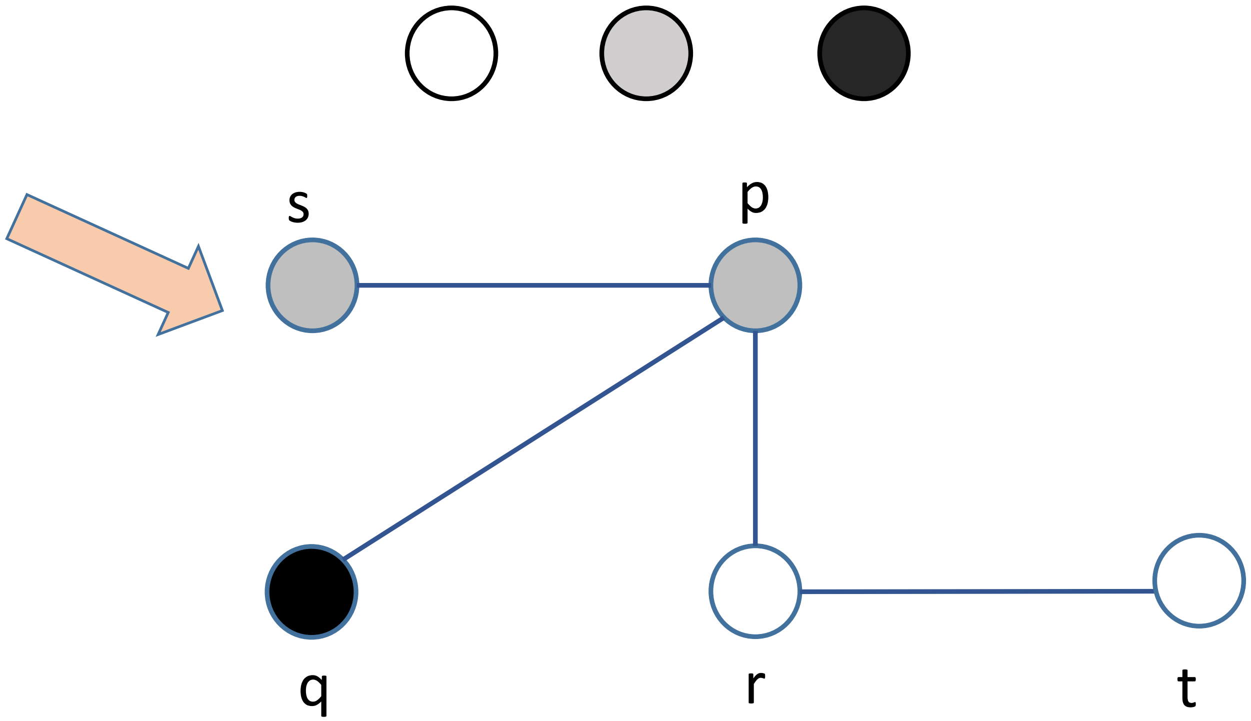
Graph Traversal: DFS (6)

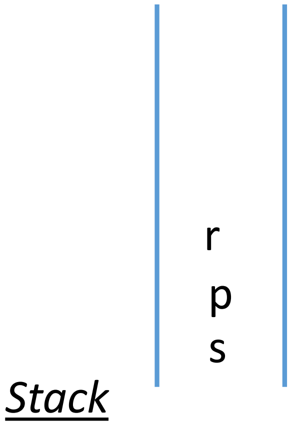
```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = Ø
time = 0
for each vertex u in G. V
    if u. color == WHITE
        DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
    if v. color == WHITE
        v. π = u
        DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

time = 4 + 1 = 5
r. d = 5
r. color = GRAY

v = G. Adj [r]
v = t
t. predecessor = r

DFS_VISIT (G, t)



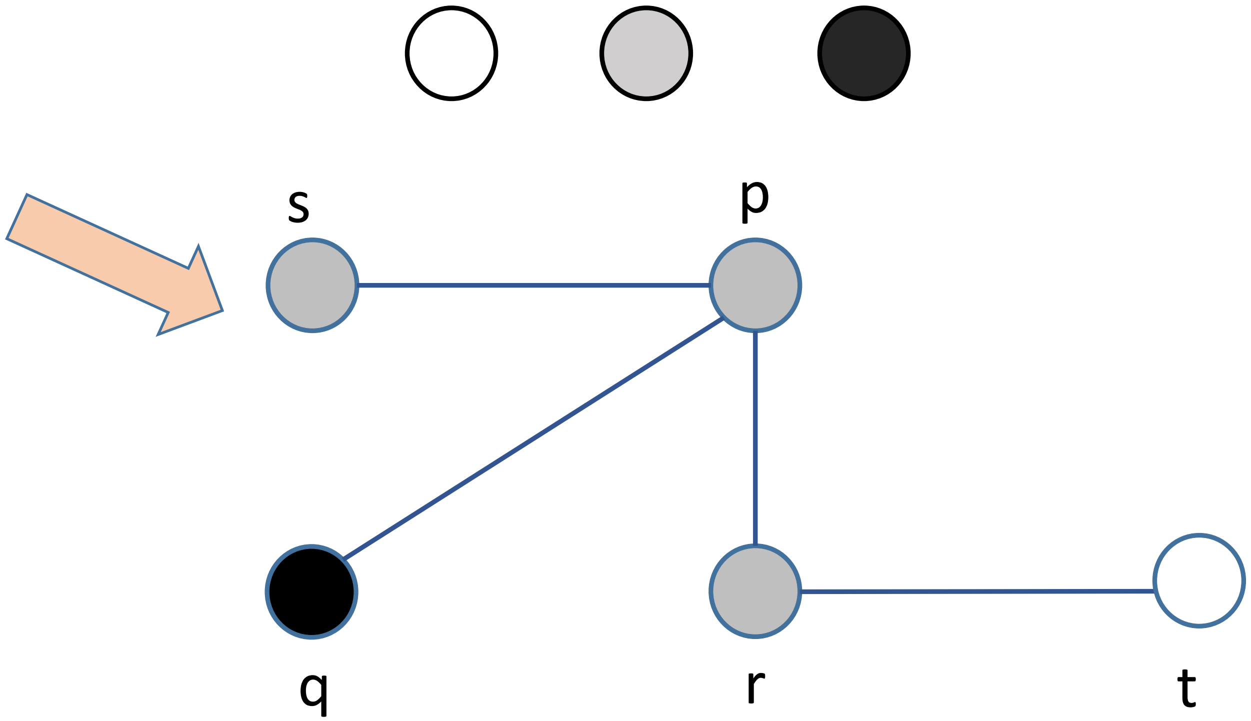


Graph Traversal: DFS (7)

```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = Ø
time = 0
for each vertex u in G. V
    if  u. color == WHITE
        DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
   if  v. color == WHITE
        v. π = u
        DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

time = 5 + 1 = 6
t. d = 6
t. color = GRAY

v = G. Adj [t]
v =



t
r
p
s

Stack

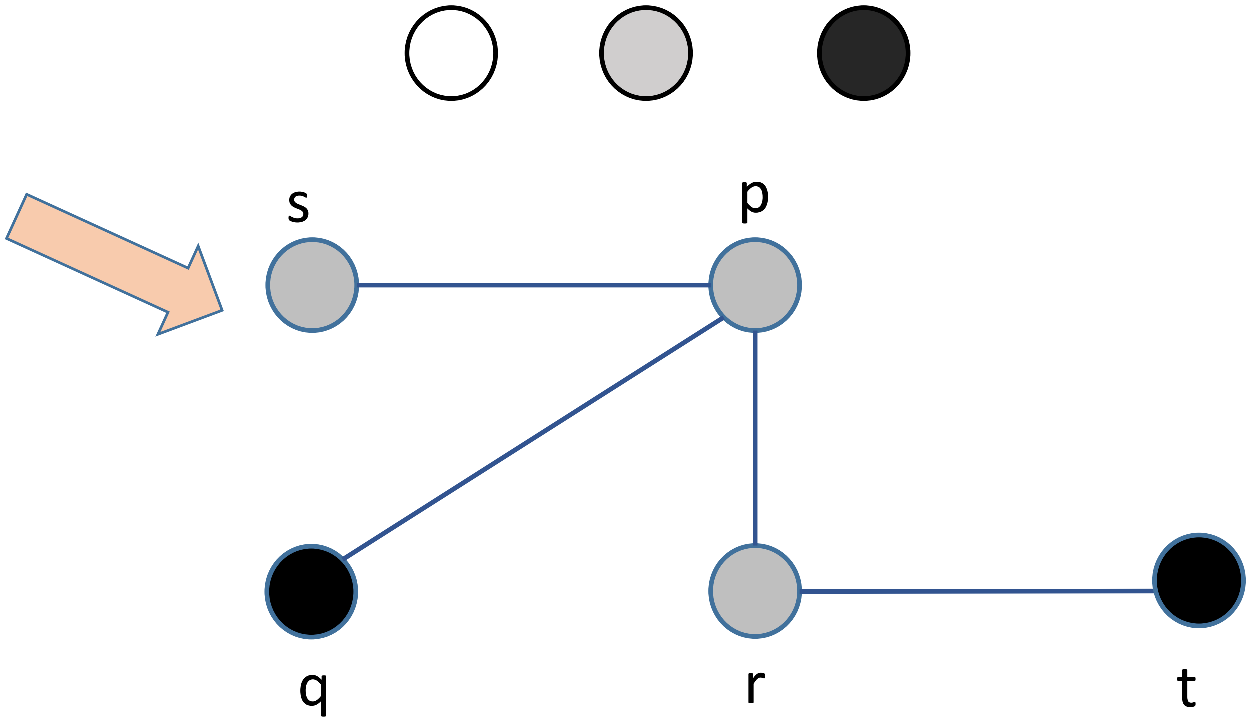
Graph Traversal: DFS (8)

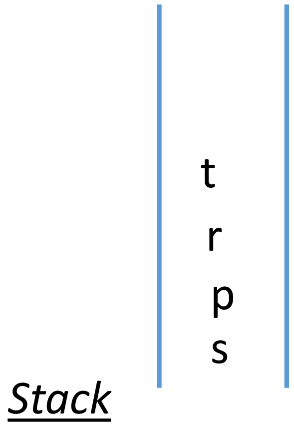
```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = ∅
time = 0
for each vertex u in G. V
    if u. color == WHITE
        DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
  if v. color == WHITE
       v. π = u
       DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

time = 5 + 1 = 6
t. d = 6
t. color = GRAY

v = G. Adj [t]
v =

t. color = BLACK
q. time = 6 + 1 = 7
q. f = 7

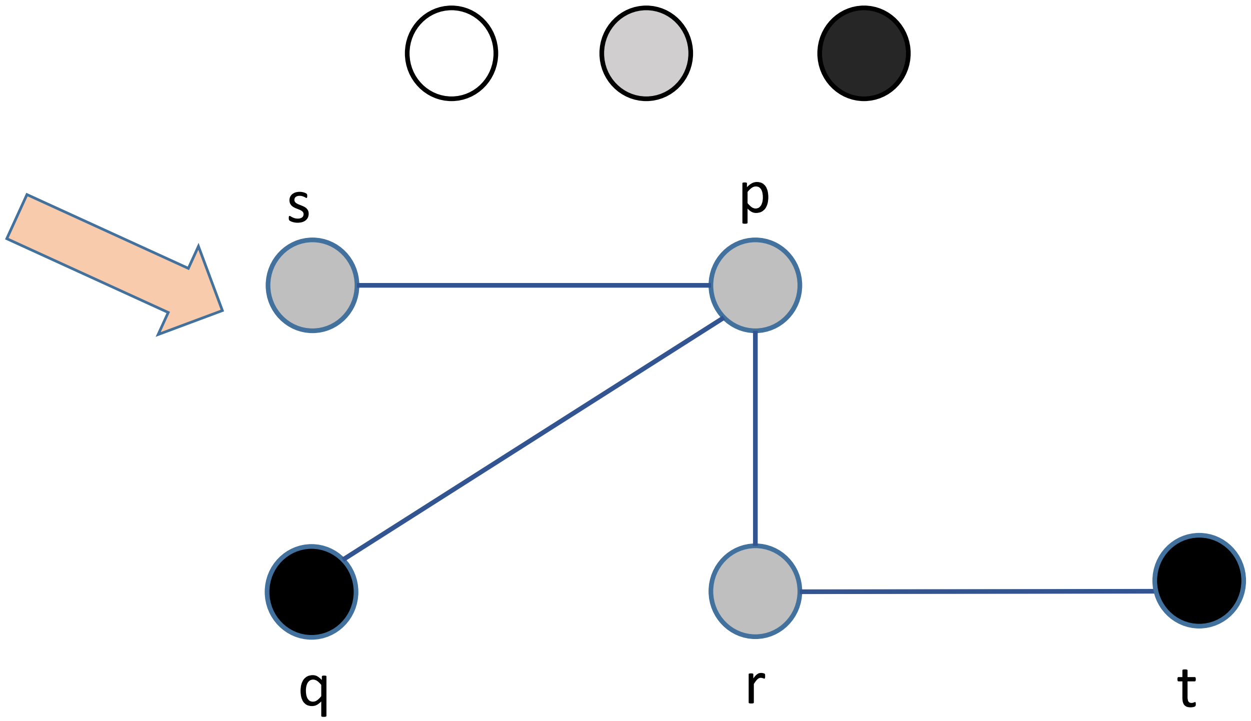


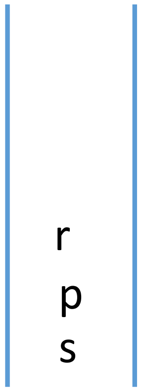


Graph Traversal: DFS (8)

```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = ∅
time = 0
for each vertex u in G. V
     if  u. color == WHITE
          DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
   if  v. color == WHITE
          v. π = u
          DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

r. color = BLACK
r. time = 7 + 1 = 8
r. f = 8



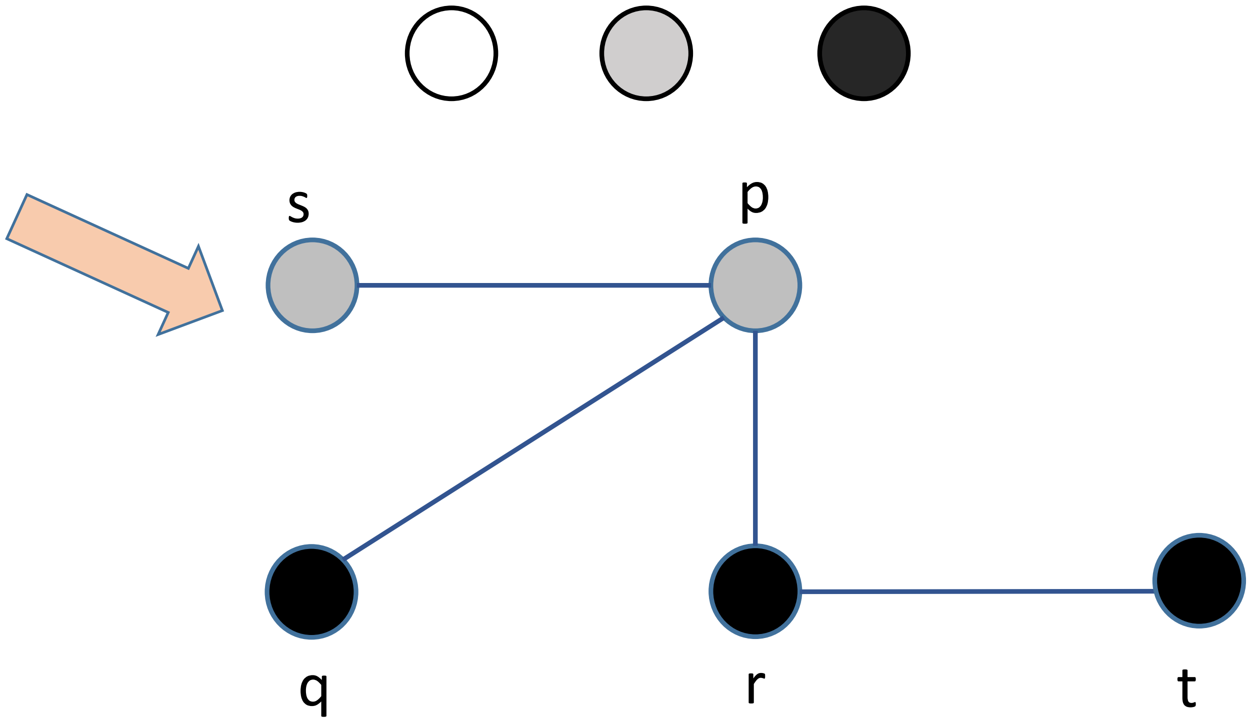


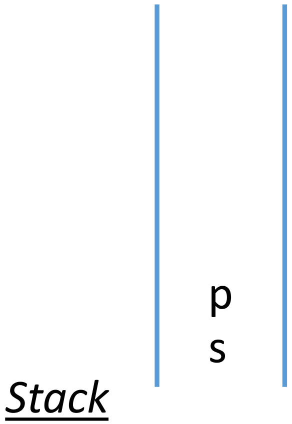
Stack

Graph Traversal: DFS (9)

```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = ∅
time = 0
for each vertex u in G. V
       if  u. color == WHITE
           DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
   if  v. color == WHITE
         v. π = u
         DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

p. color = BLACK
p. time = 8 + 1 = 9
p. f = 9





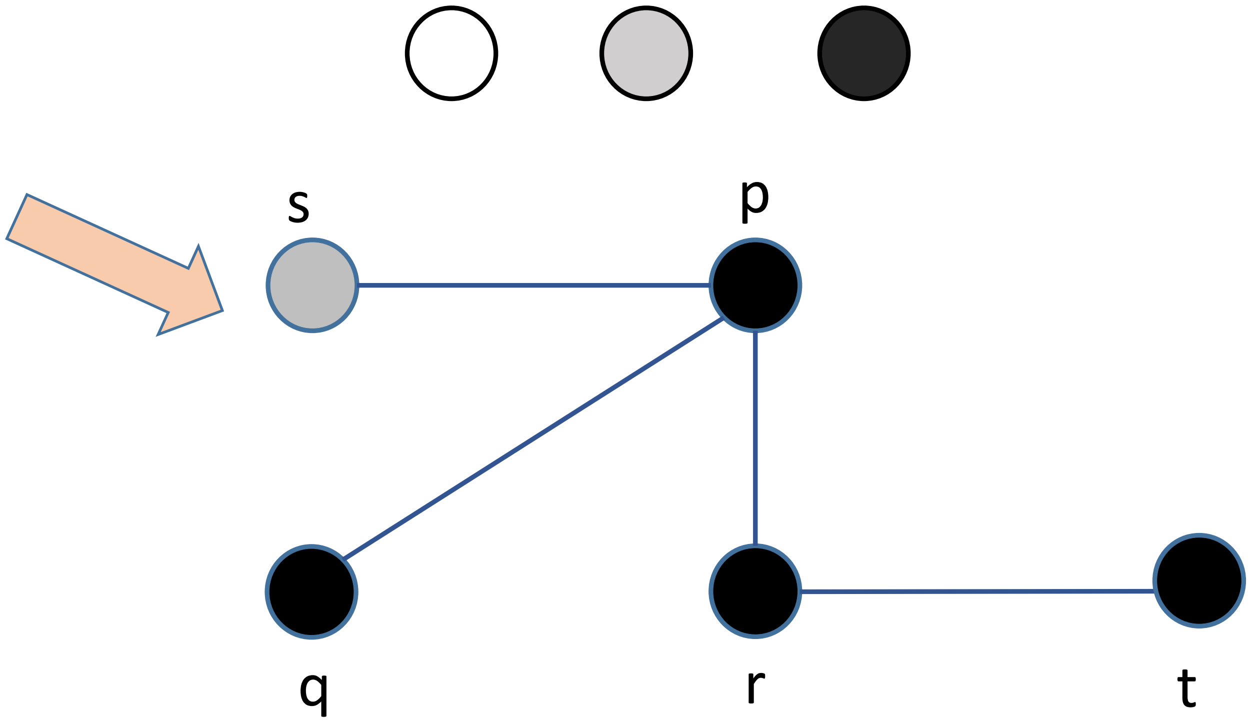
Graph Traversal: DFS (10)

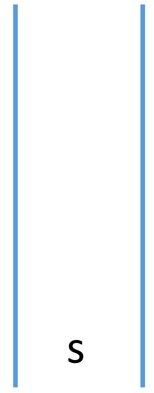
```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = ∅
time = 0
for each vertex u in G. V
    if  u. color == WHITE
        DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
   if  v. color == WHITE
         v. π = u
         DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

s. color = BLACK

s. time = 9 + 1 = 10

s. f = 10





Stack

Graph Traversal: DFS (11)

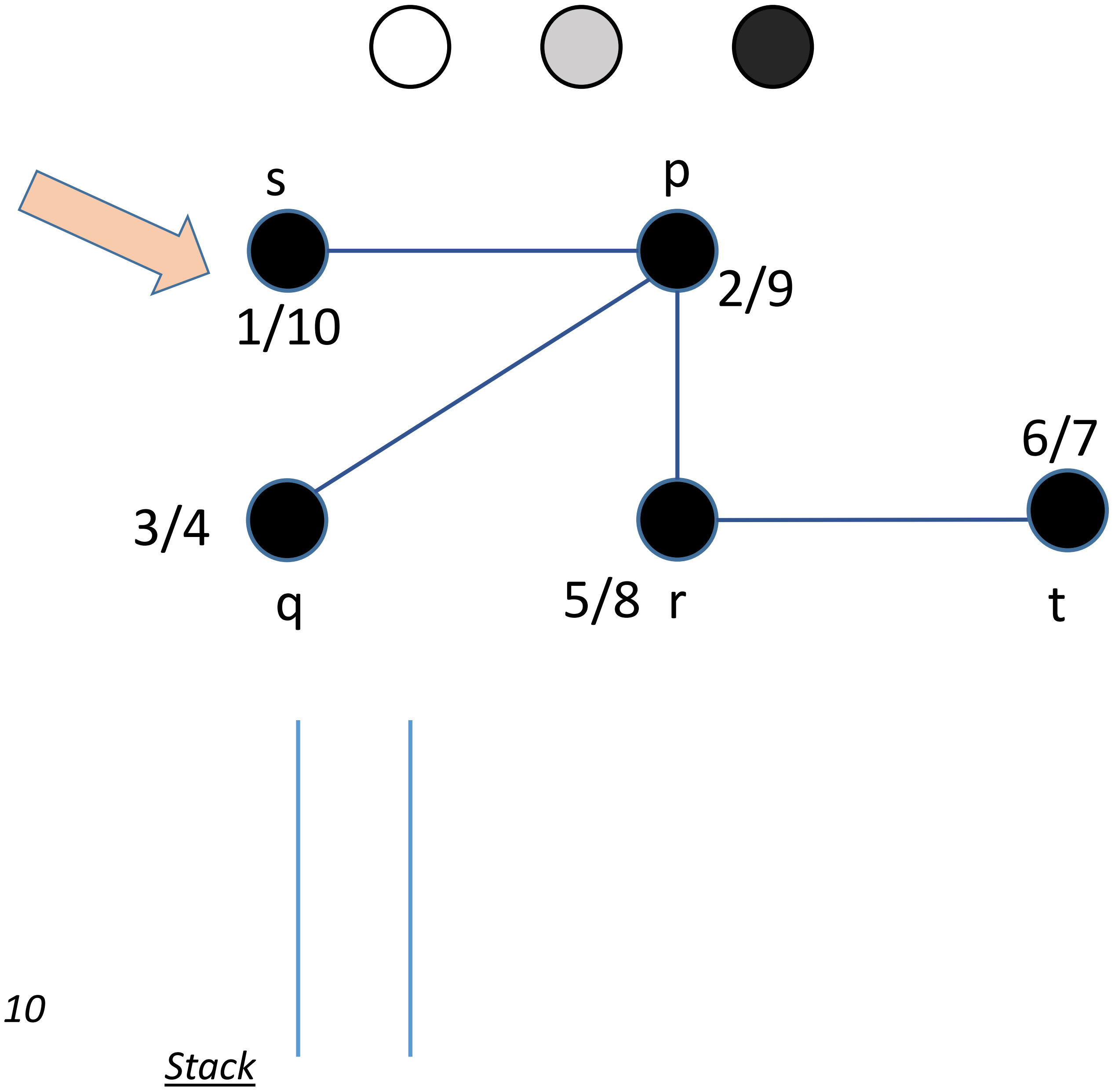
```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = Ø
time = 0
for each vertex u in G. V
    if  u. color == WHITE
        DFS_VISIT (G, u)
```

```
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
    if  v. color == WHITE
        v. π = u
        DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

s. color = BLACK

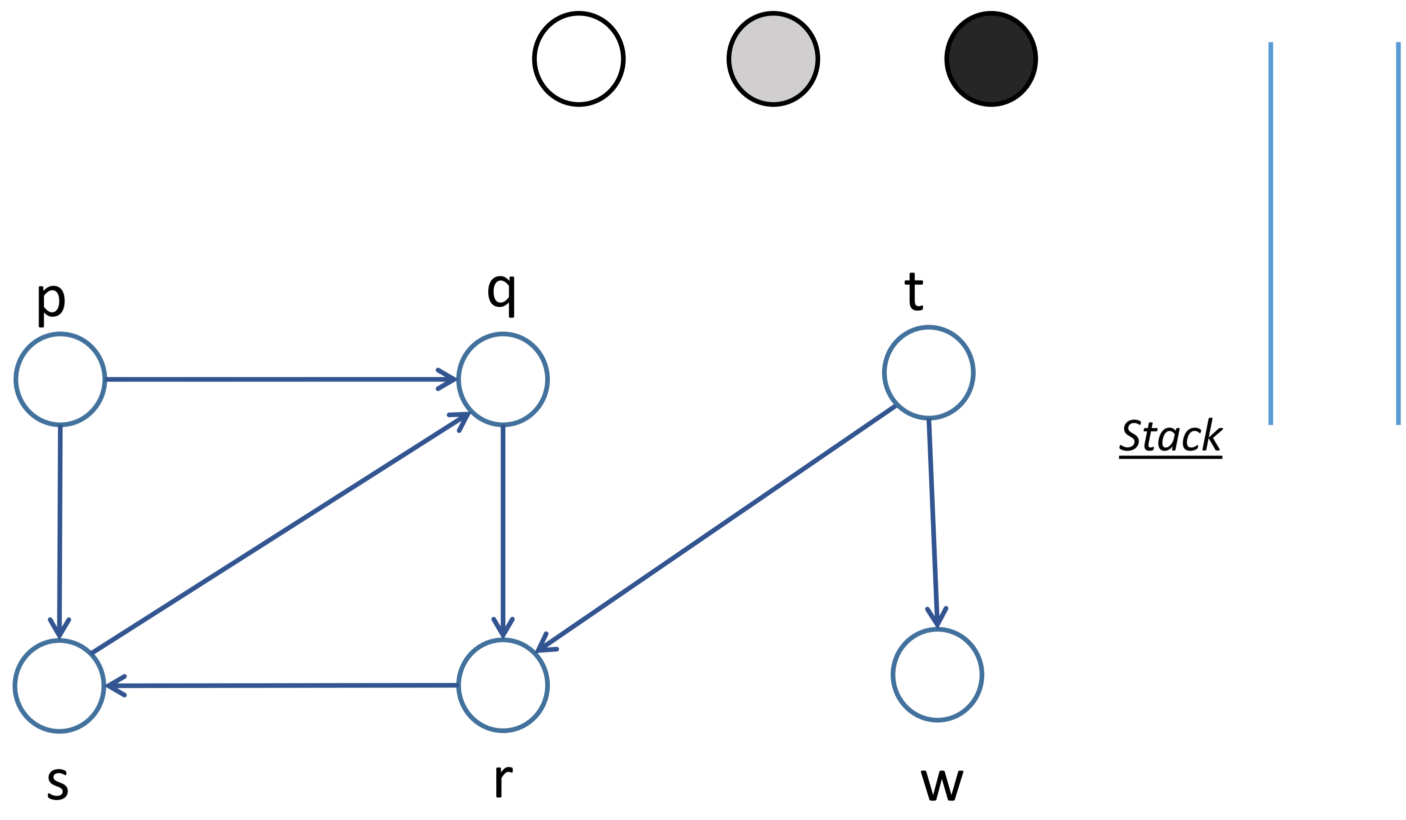
s. time = 9 + 1 = 10

s. f = 10

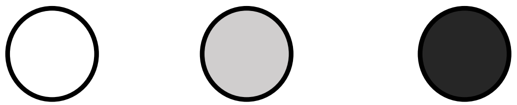


Graph Traversal: DFS (12)

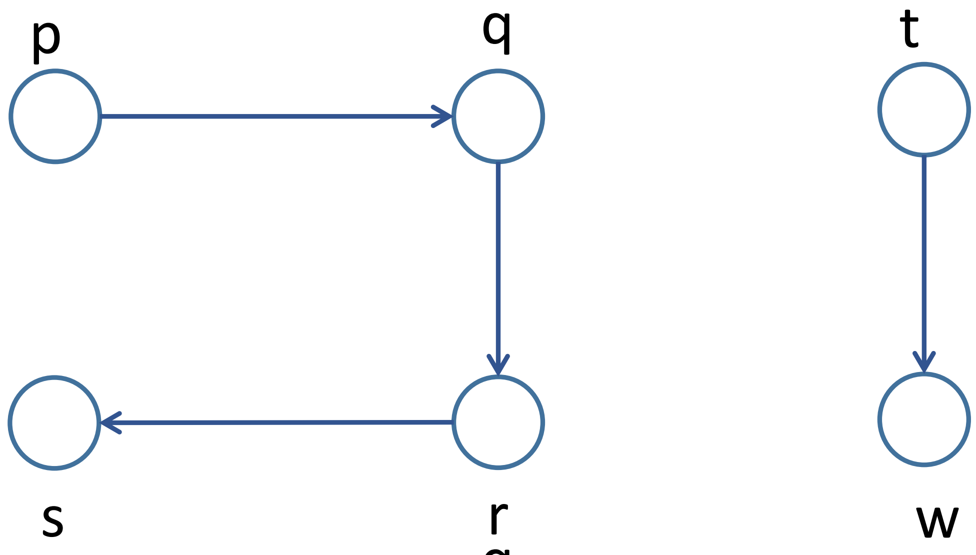
```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = Ø
time = 0
for each vertex u in G. V
    if  u. color == WHITE
        DFS_VISIT (G, u)

DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
    if  v. color == WHITE
        v. π = u
        DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```

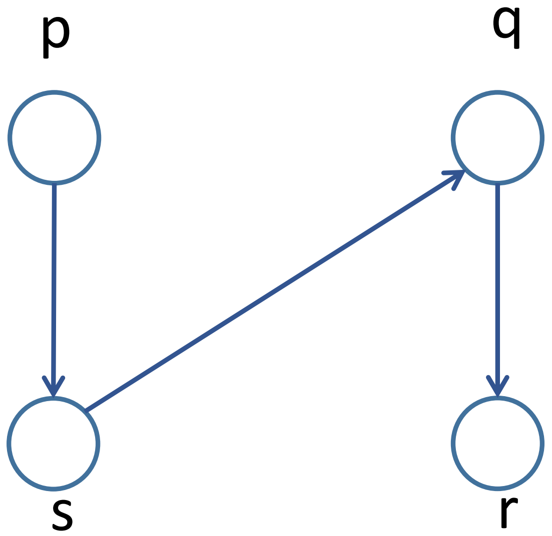


Graph Traversal: DFS (13)



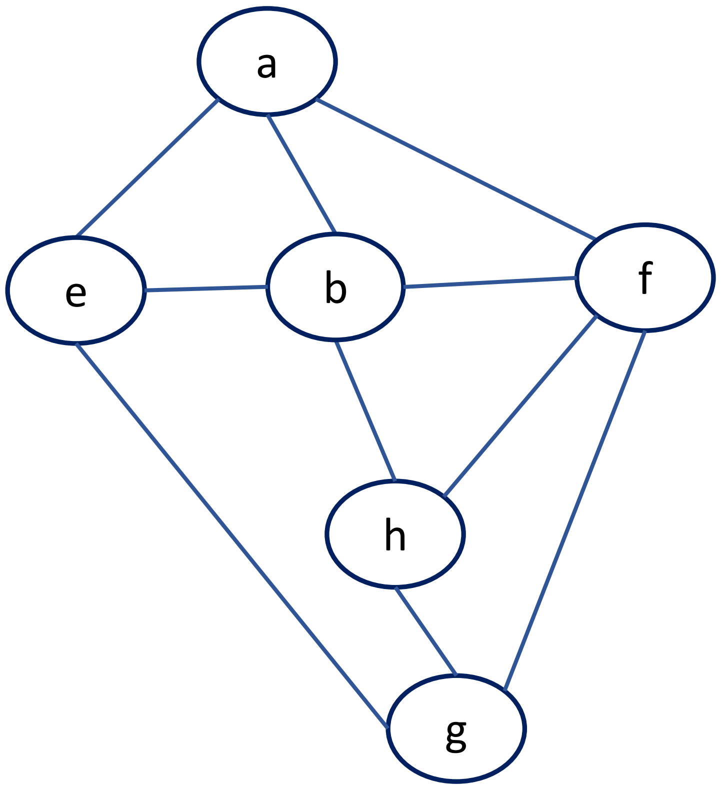
```
DFS(G,s)
for each vertex u in G. V
    u. color = WHITE
    u. π = Ø
time = 0
for each vertex u in G. V
      if  u. color == WHITE
          DFS_VISIT (G, u)
DFS_VISIT (G, u)
time = time + 1
u. d = time
u. color = GRAY
for each v ∈ G. Adj [u]
   if  v. color == WHITE
         v. π = u
         DFS_VISIT (G, v)
u. color = BLACK
time = time + 1
u. f = time
```





Exercise: DFS (1)

Consider the following graph: Which of the following orderings are possible using DFS



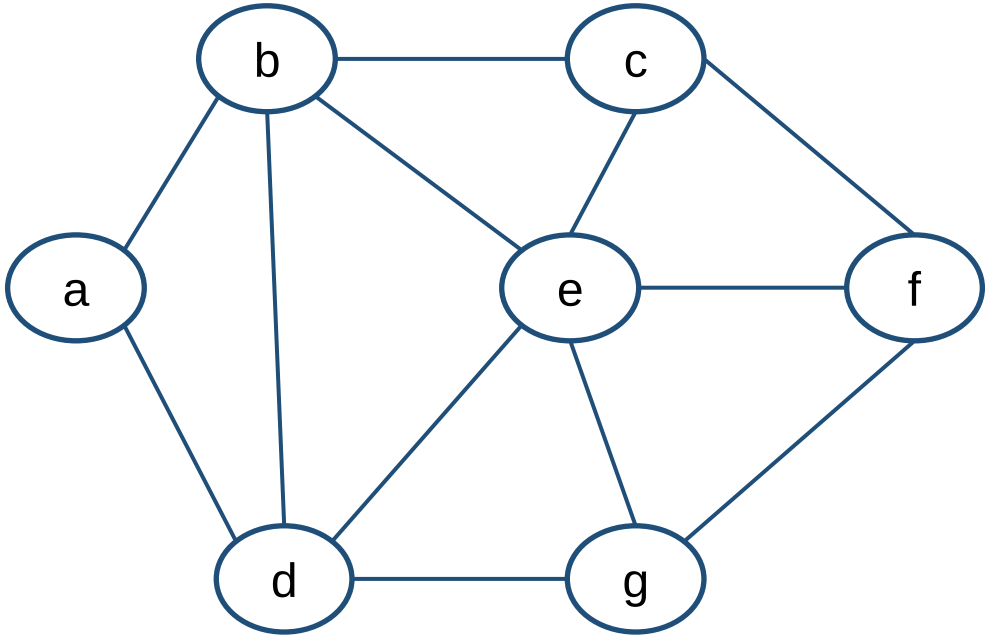
I. abeghf ; II. abfehg ; III. abfhge ; IV. afghbe

A. I, II and IV only
B. I and IV only
C. II, III and IV only
D. I, III and IV only

Exercise: DFS (1)

Consider the following graph: Which of the following orderings are possible using DFS

1. *a b e f d g c*
2. *a b e f c g d*
3. *a d g e b c f*
4. *a d b c g e f*



A. 1 and 3 only
B. 2 and 3 only
C. 2, 3 and 4 only
D. 1, 2 and 3 only

Graph Traversal: DFS Time Complexity analysis

```
DFS(G,s)
1  for each vertex u in G. V
2      u. color = WHITE
3      u. π = ∅
4  time = 0
5  for each vertex u in G. V
6          if u. color == WHITE
7              DFS_VISIT (G, u)
DFS_VISIT (G, u)
1  time = time + 1
2  u. d = time
3  u. color = GRAY
4  for each v ∈ G. Adj [u]
5     if v. color == WHITE
6          v. π = u
7          DFS_VISIT (G, v)
8  u. color = BLACK
9  time = time + 1
10 u. f = time
```

Steps 1-3 are executed "n" times → O (n) → n = |V|= # of vertices

Steps 5-7 are executed for every vertex → O (n)

Steps 1-10 → DFS_VISIT (G, *u*) ***:***

(i) In the for loop, obtaining the adjacency list: # of elements in adjacency list is equal to # of edges → m = |E| → O (m)

Total time complexity = O (n) + O (n) + O (m)
= O (n+m)
= O (V+E)

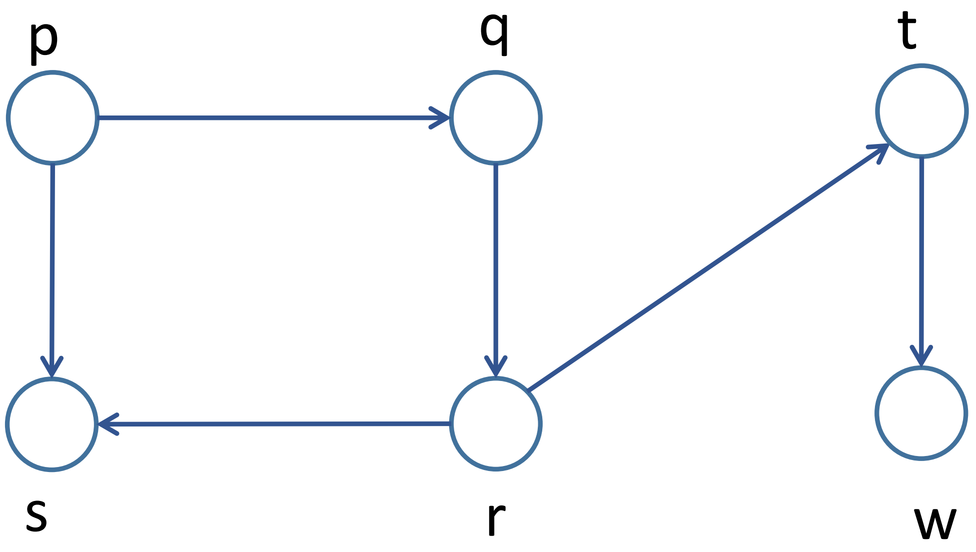
Graph Traversal Applications: DFS

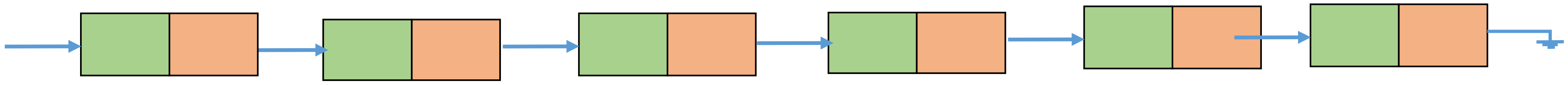
- ❖ DFS is also useful for finding shortest path distance in the graph.
- ❖ DFS forms a *depth-first forest comprising several* depth-first trees
- ❖ The implementation of DFS → Stack data structure.
- ❖ DFS colors vertices during the search to indicate their state
- ❖ DFS follows a ***<u>timestamping</u>*** technique
    - ✓ *v. d timestamp* → vertex search <u>discovered</u> time
    - ✓ *v. f timestamp* → vertex search <u>finished</u> time

Graph Traversal Applications: DFS

TOPOLOGICAL – SORT (G)

1 call DFS (G) to compute finished times *v. f* for each vertex *v*
2 as each vertex is finished, insert it onto the front of a linked list
3 return the linked list of vertices

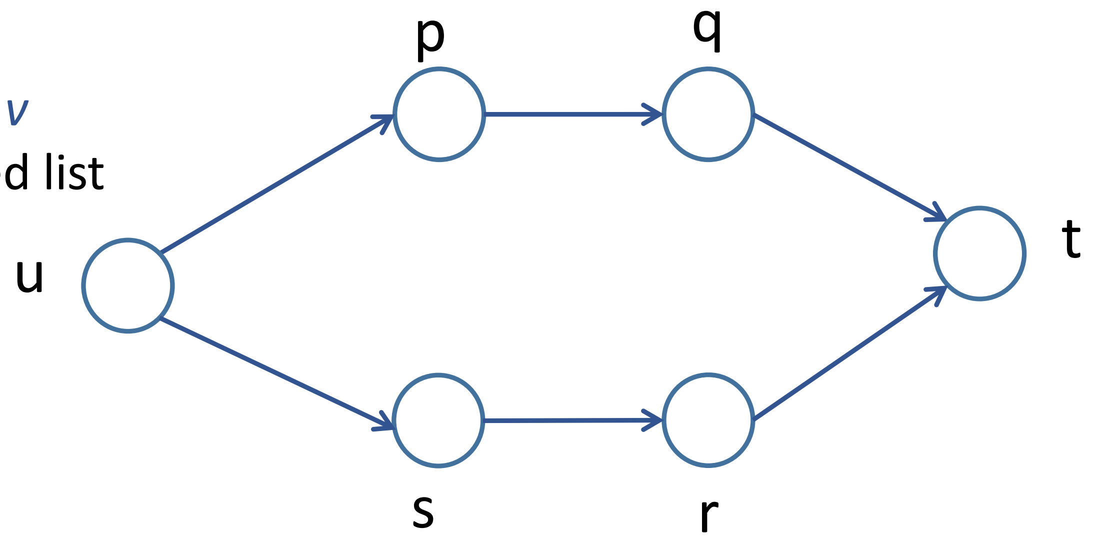


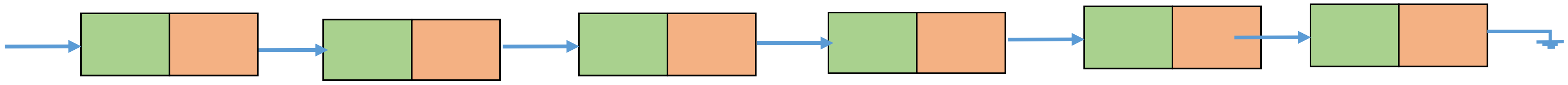


Graph Traversal Applications: DFS

TOPOLOGICAL – SORT (G)

1 call DFS (G) to compute finished times *v. f* for each vertex *v*
2 as each vertex is finished, insert it onto the front of a linked list
3 return the linked list of vertices





Topological – Sorting : Time Complexity analysis

TOPOLOGICAL – SORT (G)

1 call DFS (G) to compute finished times $v.f$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 return the linked list of vertices

→ *O (n+m)*
→ *Insertion into linked list O (1)*
→ *For n vertices* → *O (n)*
→ *To return* → *O (1)*

Total time complexity = *O (n+m) + O (n) + O (1)*
= *O (n+m)*
= *O (V+E)*

thank you!

email:
k.kondepu@iitdh.ac.in