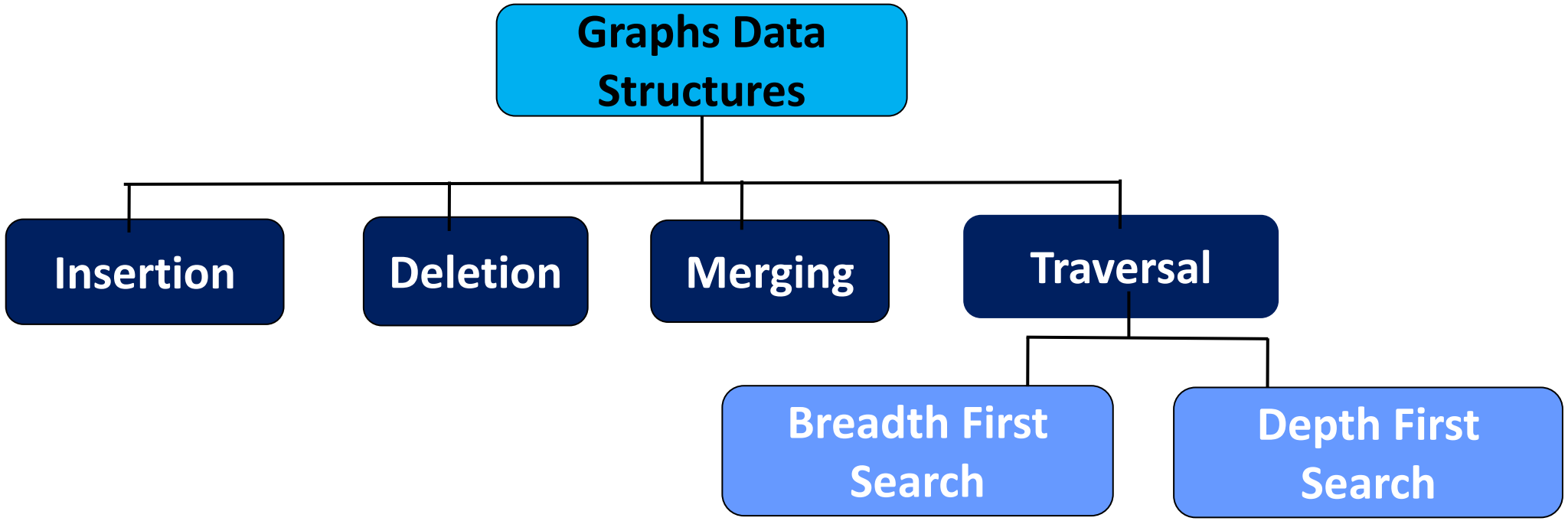
CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

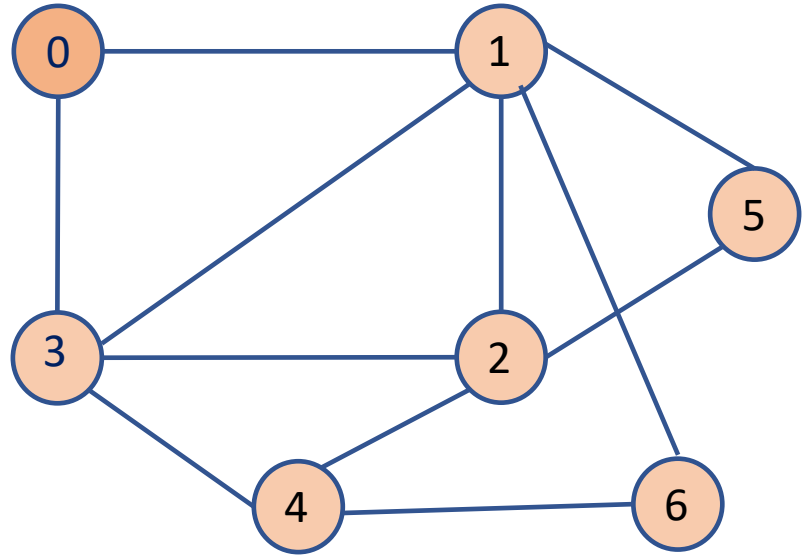
k.kondepu@iitdh.ac.in

Graphs: Operations



Recap: BFS

- BFS is useful for finding shortest path distance in the graph.
- In BFS nodes can be visited level-by-level, so it is called level-order traversal
- The implementation of BFS → Queue data structure.
- *Procedure*:
  - ✓ Initially, BFS starts at a given vertex, which is at level#0. In the first stage it visits all vertices at level#1 (i.e., adjacent vertices of the starting vertex of the graph).
  - ✓ After that, it visits all vertices at the level#2. These new vertices are the ones which are adjacent to level#1 vertices.
  - ✓ Repeat this process until all the levels of the graph is completed.



Level#0 (breadth) = 0

Level#1 = 1 3

Level#2 = 2 5 6 4

Assignment#6: BFS (1)

- Objective: Implement of "*Breadth – First – Search*", use it for a single source shortest path

- Inputs: Command-line argument: Single command line input is required
    - *The **input** file*

Example: ./bfs input.graph

(i) the first line of input.graph file contain: # of vertices |V| # of edges |E|

For example: |V| = 12 |E| = 7

(ii) every other line represents and edge between vertex x and vertex y

For example: 0 4 → an edge between 0 to 4

0 5 → an edge between 0 to 5

// ***Content of input.graph***

12 7 → first line → |V| |E|
0 4
0 5
0 10
5 6
5 7
6 8
7 9

Assignment#6: BFS (2)

// ***Content of input.graph***

12 7 → first line → |V||E|
0 4
0 5
0 10
5 6
5 7
6 8
7 9

- Output: A file → ***sd.txt***
  - What the output file should contain?
    - The output file should contain single source shortest path problem on undirected graphs

(i) the first line contain distance from source vertex 0 to 0

(ii) If there is no path exists from the source vertex 0 to any other given vertex (means if no shortest path exists from the source vertex), then the output file should contain "-1" at that vertex

(iii) If there is no vertex is presented in the given input file, in that case also the output file should contain value "-1" at that vertex

(iv) If there is a edge connecting from source to particular vertex, then update the total distance.

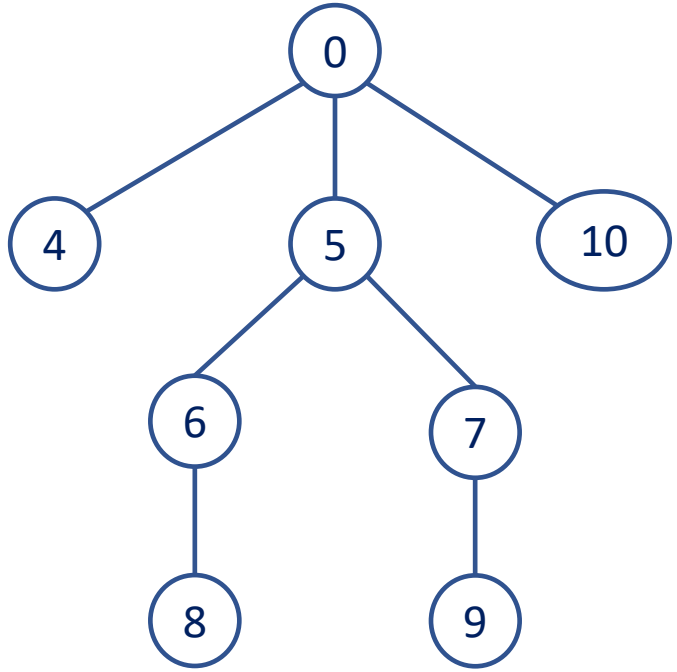
Note: Each vertex will be at unit distance. It will be as good as total edge counts

Assignment#6: BFS (3)

// ***Content of input.graph***

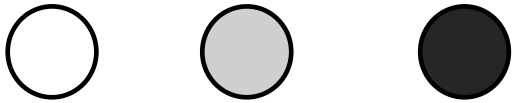
12 7 → first line → |V| |E|
0 4
0 5
0 10
5 6
5 7
6 8
7 9

	// ***Content of sd.txt***
0	0
1	-1
2	-1
3	-1
4	1
5	1
6	2
7	2
8	3
9	3
10	1
11	-1



1) the first line contains distance from source vertex 0 to 0
2) If there is no path exists from the source vertex 0 to any other given vertex (means if no shortest path exists from the source vertex), then the output file should contain “-1” at that vertex
3) If there is no vertex is presented in the given input file, in that case also the output file should contain value “-1” at that vertex
4) If there is an edge connecting from source to a particular vertex, then update the total distance.
5) Note: Each vertex will be at a unit distance. It will be as good as total edge counts

Graph Traversal: BFS → Color coding Intuition: CLRS Book

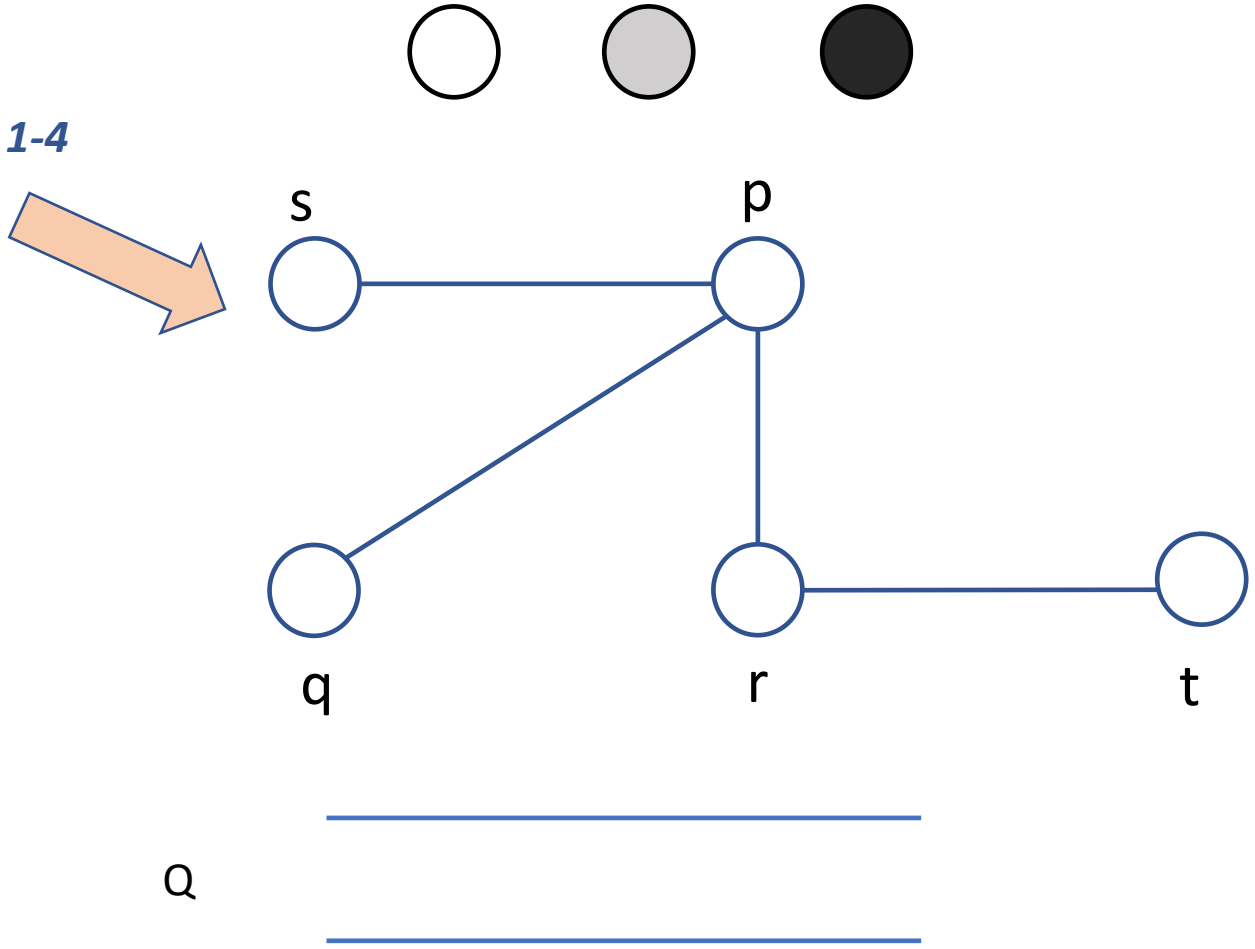


- Any vertex during the BFS is either WHITE, GRAY and BLACK
- A *WHITE* vertex → this particular vertex is not yet visited
- A *GRAY* vertex → this particular vertex is presented in the QUEUE. This vertex has been visited, but all its adjacent vertices are not yet visited
- A BLACK vertex → this particular vertex is visited and also all its adjacent vertices are also visited. It is also Dequeued from the QUEUE
- Note that the notation only applicable as per CLRS text book

Graph Traversal: BFS Code Snippet (1)

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞        //Distance from the source node
4     u. π = Ø        //predecessor
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
```

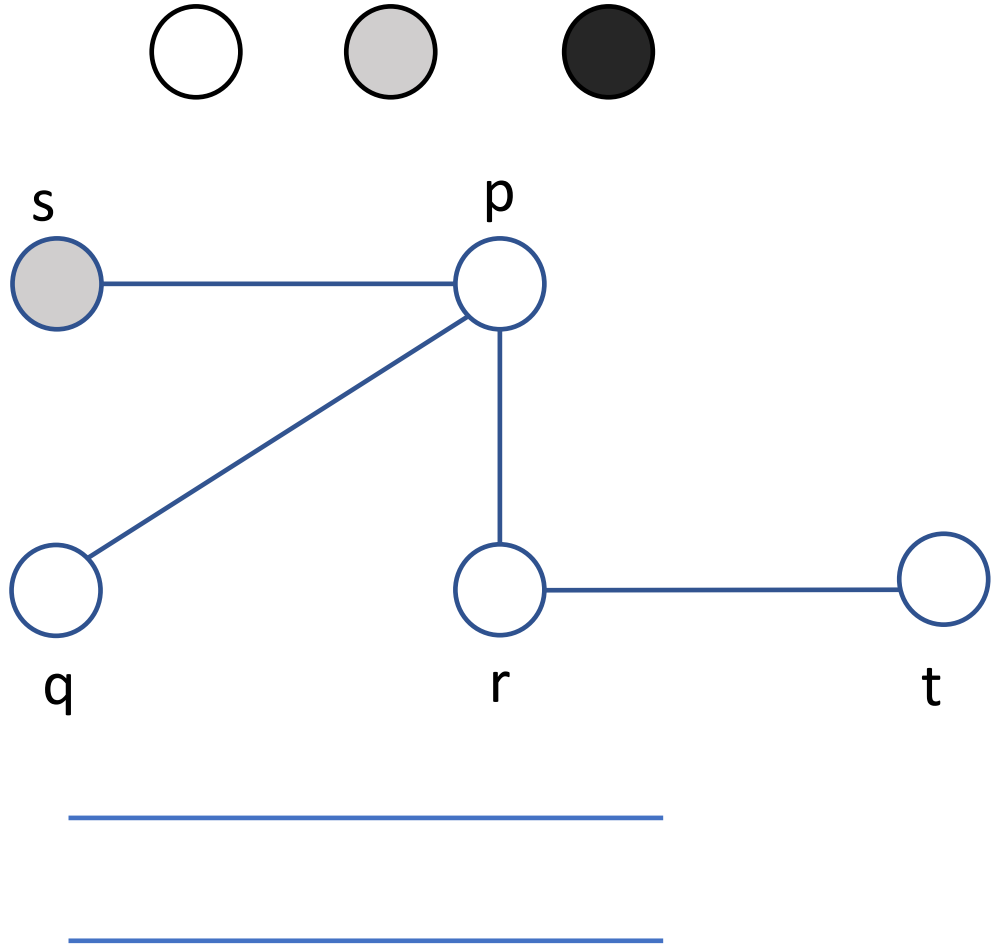
Steps 1-4



Graph Traversal: BFS Code Snippet (2)

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = ∅
5  s. color = GRAY
6  s. d = 0
7  s. π = ∅
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 5-9



Graph Traversal: BFS Code Snippet (3)

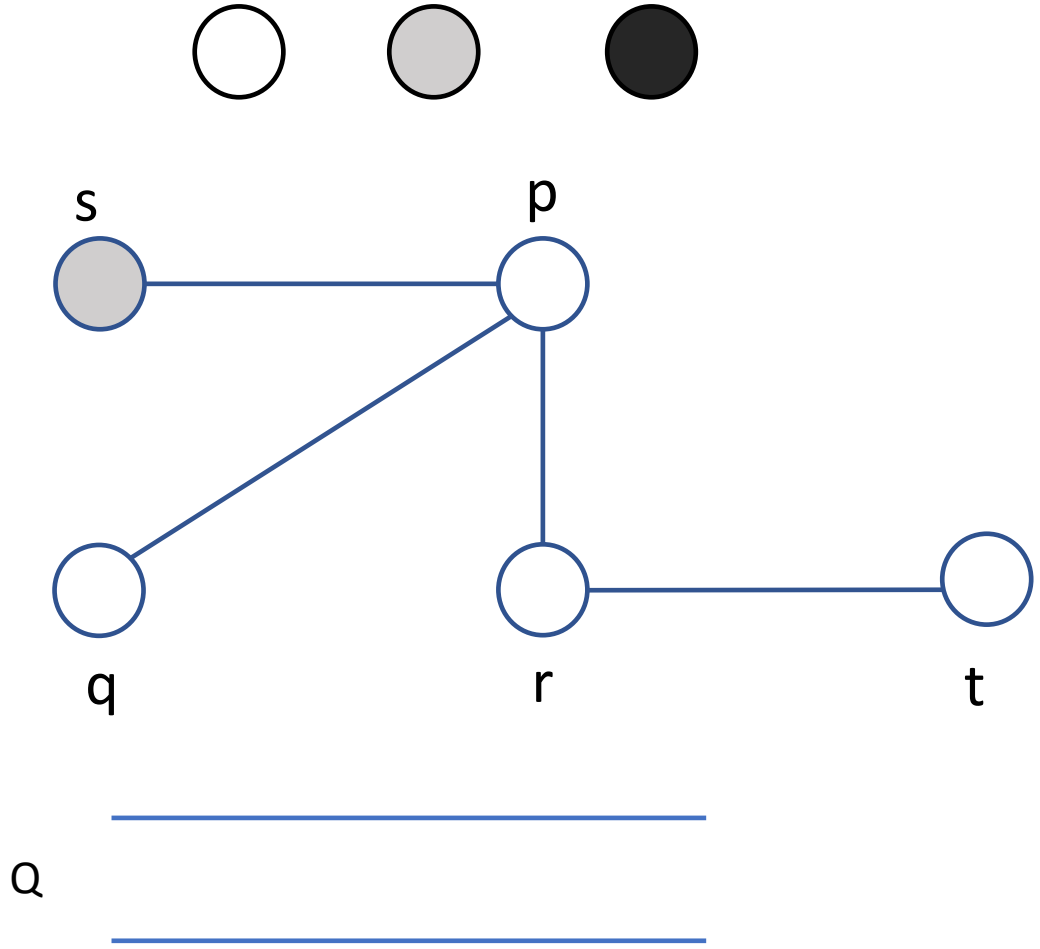
```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = ∅
5  s. color = GRAY
6  s. d = 0
7  s. π = ∅
8  Q = NULL
9  ENQUEUE(Q,s)
10 while (Q != NULL)
11    u = DEQUEUE (Q)
12    for each v in G. Adj [u]
13       if v. color == WHITE
14          v. color = GRAY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

Steps 10-17

u = s

v = G. Adj [s]

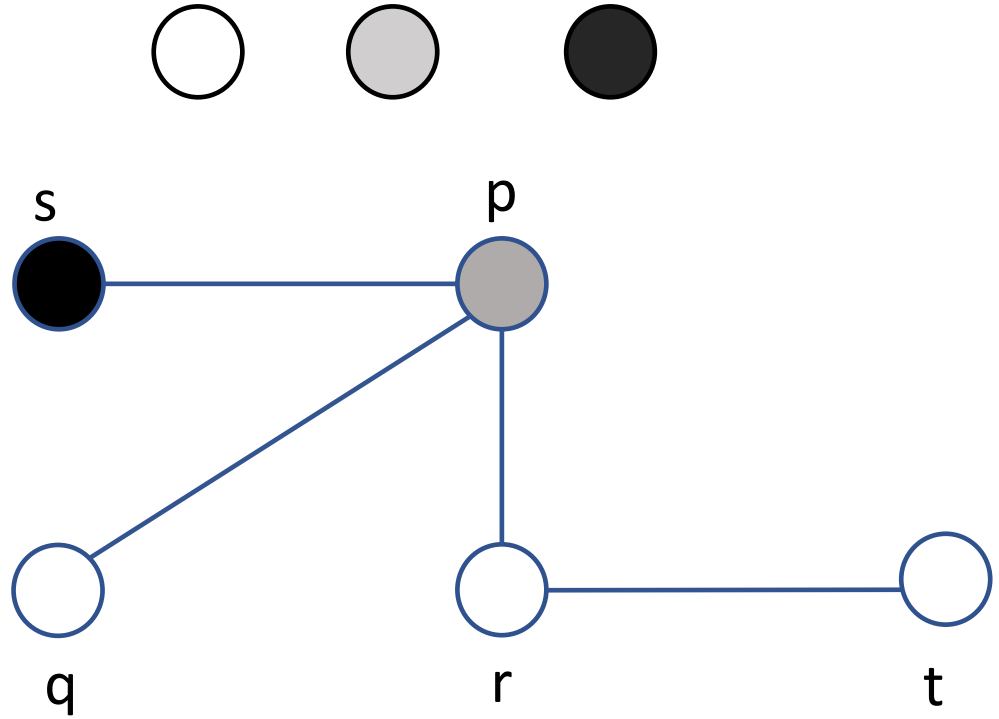
= p



Graph Traversal: BFS Code Snippet (4)

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = ∅
5  s. color = GRAY
6  s. d = 0
7  s. π = ∅
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 18



u = s

v = G. Adj [s]

= p

Q

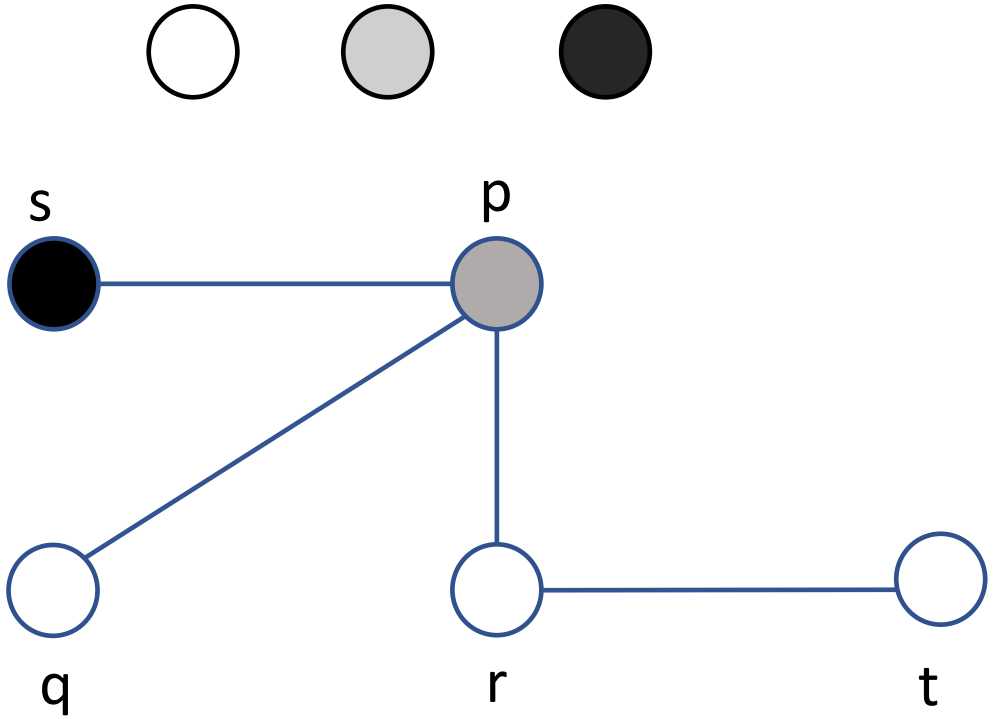
Graph Traversal: BFS Code Snippet (5)

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = ∅
5  s. color = GRAY
6  s. d = 0
7  s. π = ∅
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 10-17

Steps 18

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



Q p

Graph Traversal: BFS Code Snippet (6)

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = ∅
5  s. color = GRAY
6  s. d = 0
7  s. π = ∅
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
```

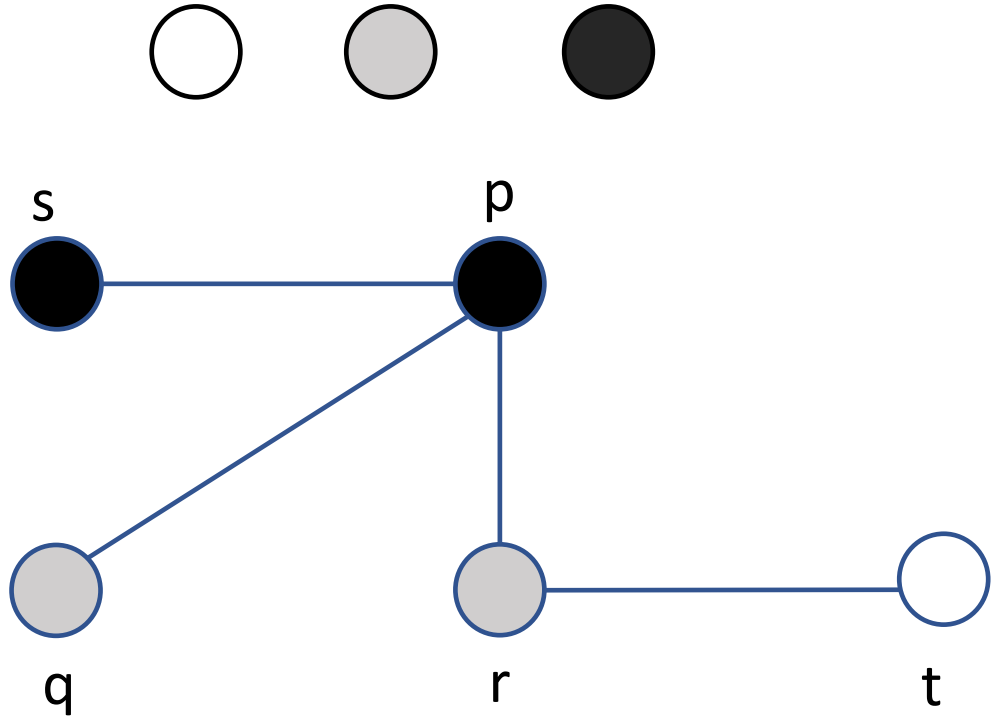
Steps 10-17

Steps 18

u = q

v = G. Adj [p]

=



Q q r

Graph Traversal: BFS Code Snippet (7)

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = ∅
5  s. color = GRAY
6  s. d = 0
7  s. π = ∅
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
```

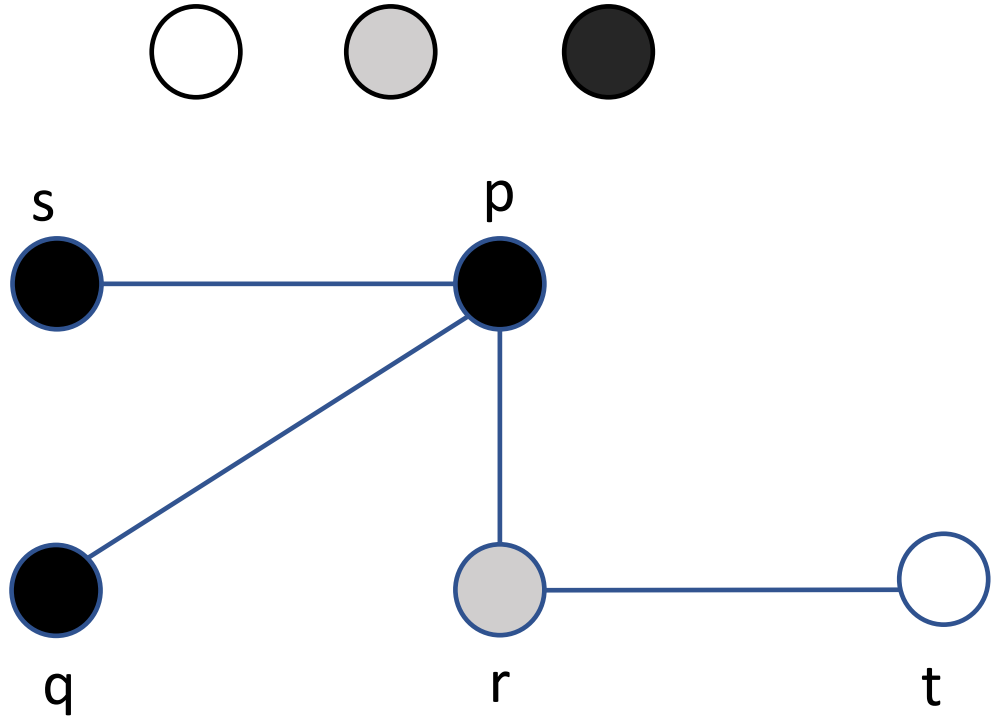
Steps 10-17

Steps 18

u = r

v = G. Adj [p]

= t



Q r

Graph Traversal: BFS Code Snippet (8)

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = ∅
5  s. color = GRAY
6  s. d = 0
7  s. π = ∅
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
```

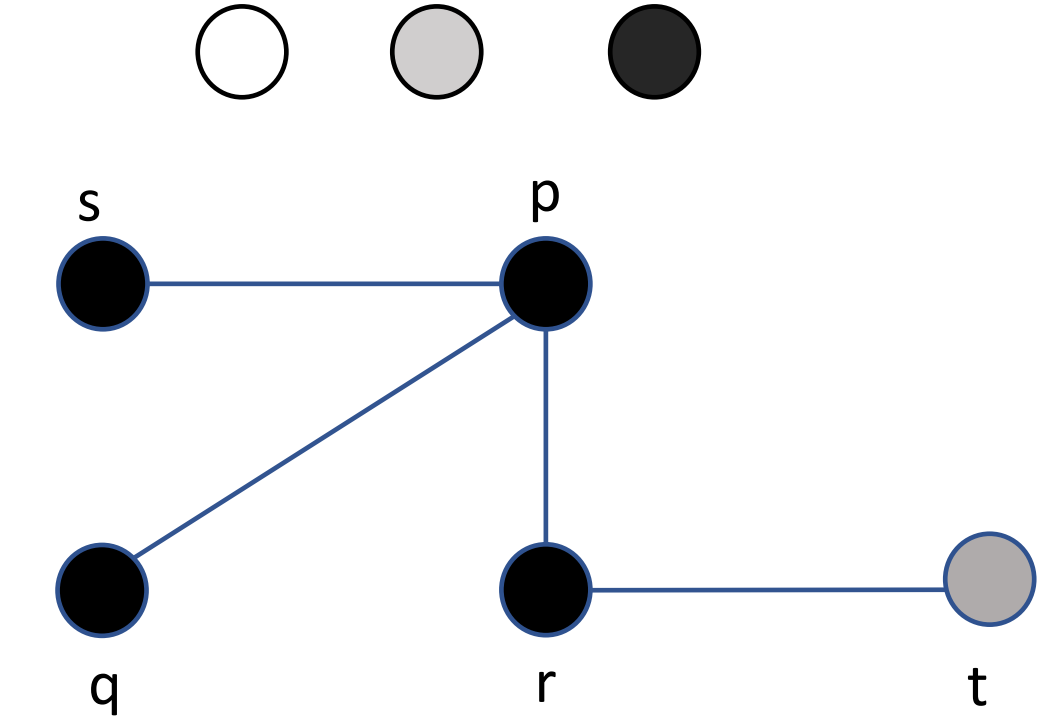
Steps 10-17

Steps 18

u = t

v = G. Adj [p]

=

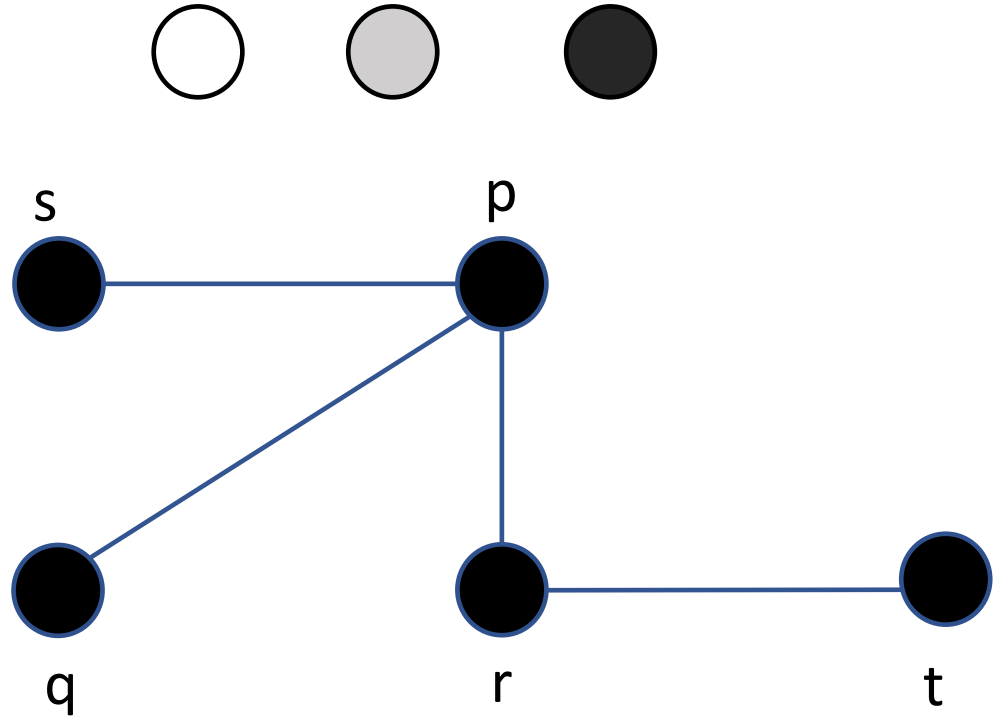


Q t

Graph Traversal: BFS Code Snippet (9)

```
BFS(G,s)
1  for each vertex u in G. V - {s}
2     u. color = WHITE
3     u. d = ∞
4     u. π = ∅
5  s. color = GRAY
6  s. d = 0
7  s. π = ∅
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 18



u = t

v = G. Adj [p]

=

Q

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 = ∅
5  s. color = GRAY
6  s. d = 0
7  s. π = ∅
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)

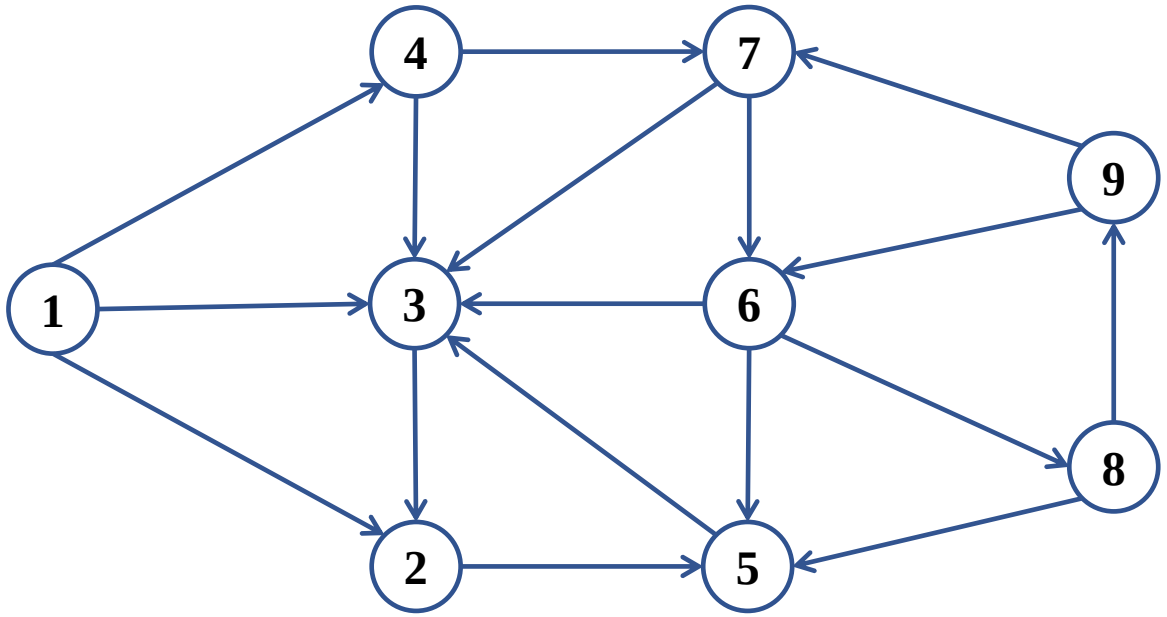
Steps 10-18 :

(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)

Exercise: BFS (1)

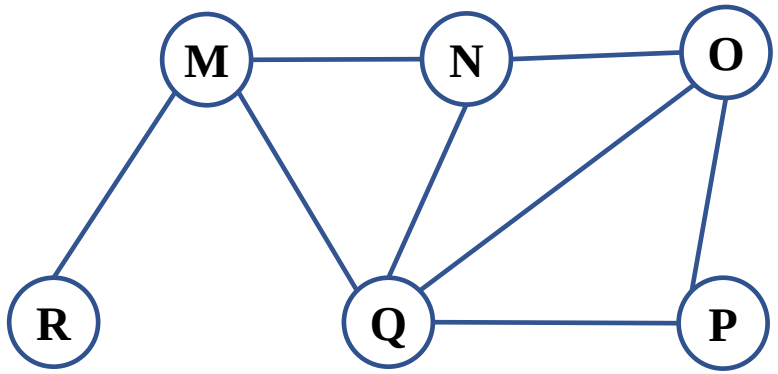
What is the shortest-path P from 1 to 8?



Exercise: BFS (2)

The BFS algorithm has been implemented using the queue data structure. Which one of the following is a possible order of visiting the nodes in the graph below?

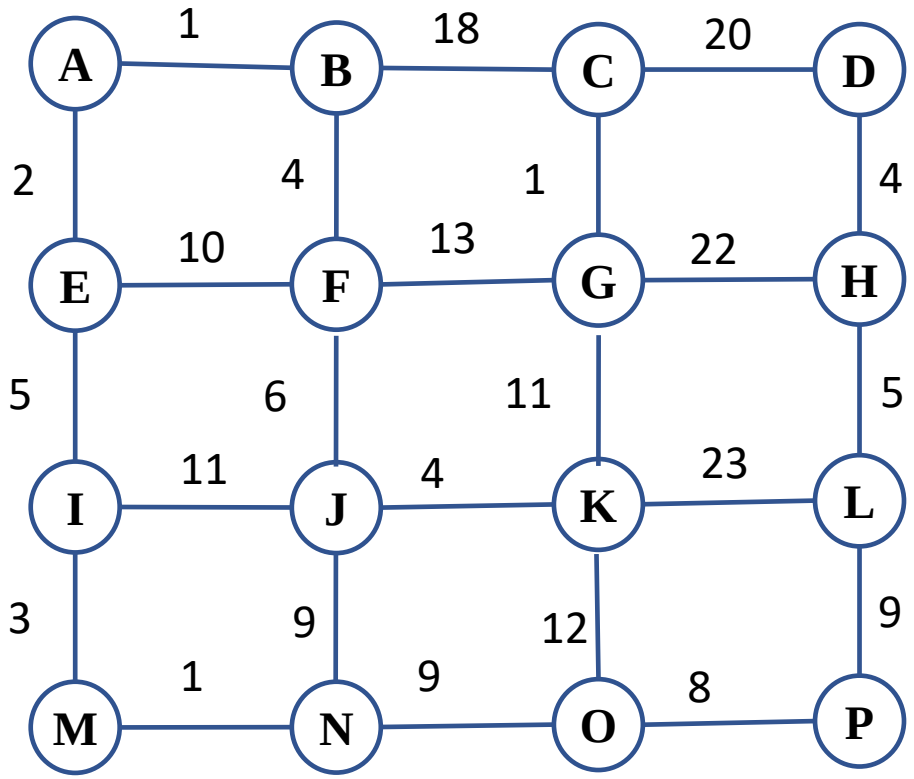
A. MNOPQR
B. NQMPOR
C. QMNROP
D. POQNMR



Exercise: BFS (3)

Consider the following graph: Which of the following orderings are possible using BFS with starting *vertex A*.

A. A, B, E, C, F, I, D, G, J, M, H, K, N, L, O, P
B. A, E, B, I, F, C, M, J, G, D, N, K, H, O, L, P
C. A, E, B, F, I, C, G, J, M, D, H, K, N, L, O, P
D. All the above



thank you!

email:
k.kondepu@iitdh.ac.in