Graphs and Networks

IE 509 Computer Programming

Jayendran V
IEOR @ IIT Bombay

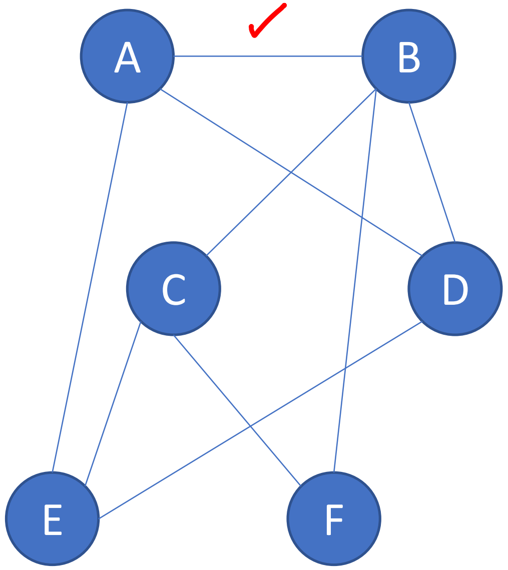
Introduction

nodes arcs

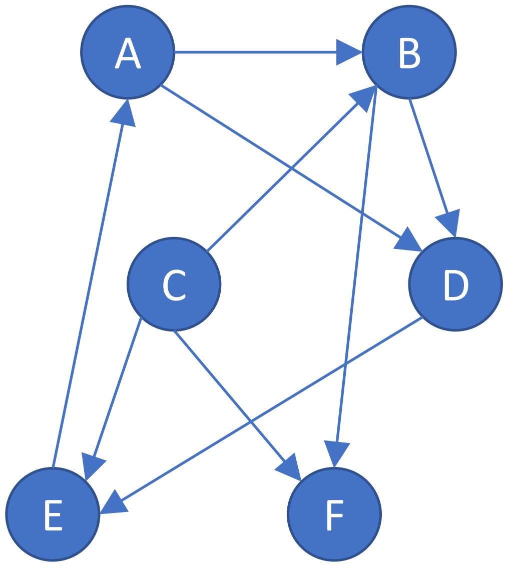
- Graph *G* consists of a set of vertices *V* connected by a set of edges *E*

Unweighted

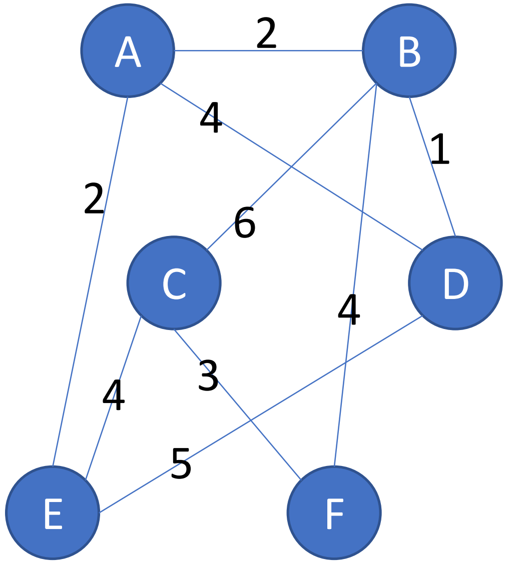
Undirected



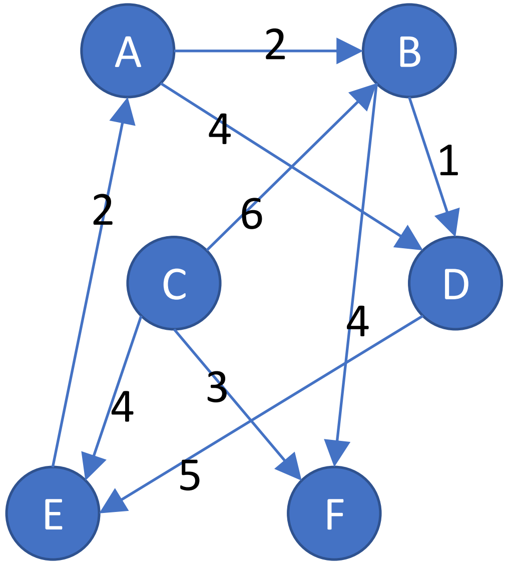
Directed



Undirected, Weighted



Directed, Weighted



- Others:
  - Infinite graphs, Complete graphs,

Representing Graphs in Python.

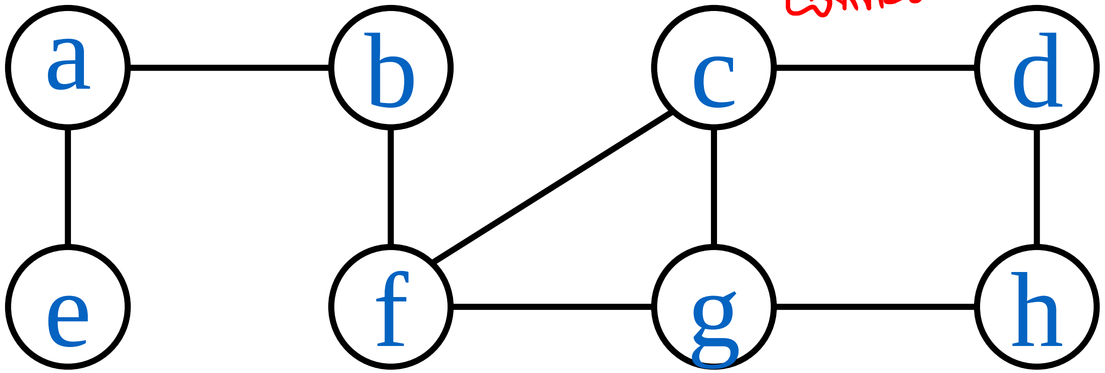
- The vertices and edges can be represented as a dictionary
  - The vertices are the ‘key’ and the ‘value’ is a list of connected edges
  - This is known as Adjacency List representation
- Example

list of all adjacent(ly) connected vertices

KEY

VALUES AS LIST

```
G = { "a" : ['b','e'],
      'b' : ['a', 'f'],
      'c' : ['f', 'g', 'd'],
      'd' : ['c', 'h'],
      'e' : ['a'],
      'f' : ['b', 'c', 'g'],
      'g' : ['c', 'f', 'h'],
      'h' : ['g', 'd'] }
```



- Disadvantage
  - No quick way to find if edge(u, v) is present in graph.
- Alternate: Adjacency Matrix representation

Basic Graph functions

- Encode a graphs as a dictionary
- Display list of vertices
- Display list of edges
- Find a given edge
- Add new vertex and edges
- …

We have a graph G

$V(G)$: represents a list of vertices of graph G

Graph Searching

- Given: undirected or directed Graph G = (V, E) unweighted
- Need to explore G to understand its structure, by systematically following the edges of the graph

- Breadth First Search (BFS)
- Depth First Search (DFS)

- Graph searching is *fundamental* to many graph algorithms

Breadth First Search (BFS)

- Given a graph G = (V, E) and a source vertex *s*, BFS systematically explores edges of G to discover every vertex reachable from *s*
  - BFS computes the distance (smallest number of edges) from *s* to every reachable vertex. → shortest path..
  - It also produces a "breadth-first" tree with root *s*

- BFS discovers all vertices a distance *k* from *s* before discovering any vertices at distance *k* + 1.
  - Hence the name!

Let’s understand the BFS algorithm

- In BFS, we will pick a vertex (root), then discover its ‘children’, then their children and so on.
- For use in algo, let’s *color the vertices* Color [v]
  - White vertices: Vertex not yet discovered. Initially all vertices are white
  - Grey vertices: Vertex discovered but not fully explored
  - Black vertices: Vertex discovered and fully explored
- Algorithm should also output the
  - distance of all reachable vertex from the root.
  - Path to reach a vertex from the root (Breadth first tree)
    - We can simply record the Immediate predecessor of a vertex

BFS algorithm

BFS takes as input Graph G, and starting vertex s

vertex Edges

~~Pseudo~~ Pseudo code

set of vertices of G

BFSAlgo(G, s)

1. For each $v \in V[G]$
2. color[v] ← 'white'
3. distance[v] ← ∞
4. pre[v] ← NULL
5. Next
6. Queue: Q
7. distance[s] ← 0
8. Enqueue(Q, s)

Initialisation

FIFO logic

Q: s

join at end of queue

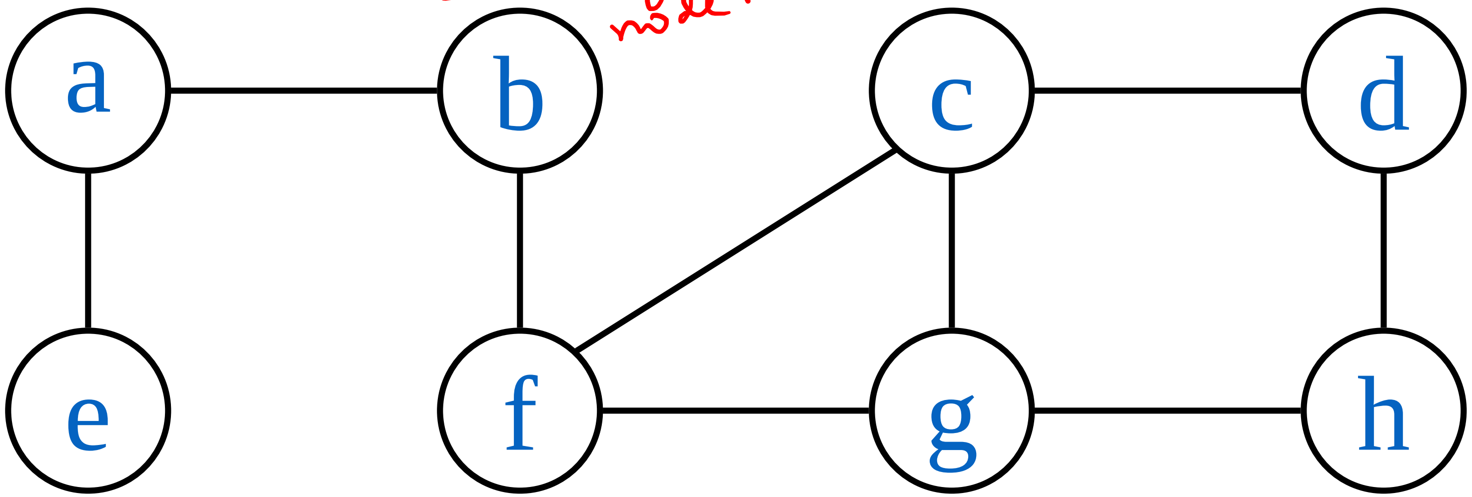
9. while (Q not empty)
10. u ← Dequeue(Q)
11. For each $v \in Adj[u]$
12. If (color[v] ==‘white’)
13. color[v] ← 'grey'
14. distance[v] ← distance[u] + 1
15. pre[v] ← u
16. Enqueue(Q, v)
17. endif
18. Next
19. color[u] ← 'black'
20. End while

remove from front of queue

Breadth-First Search: Example

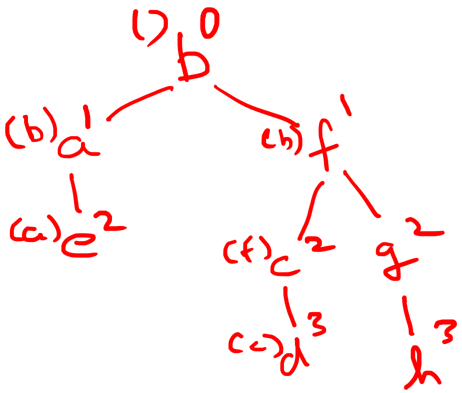
- Consider Graph, G, given

Starting node.



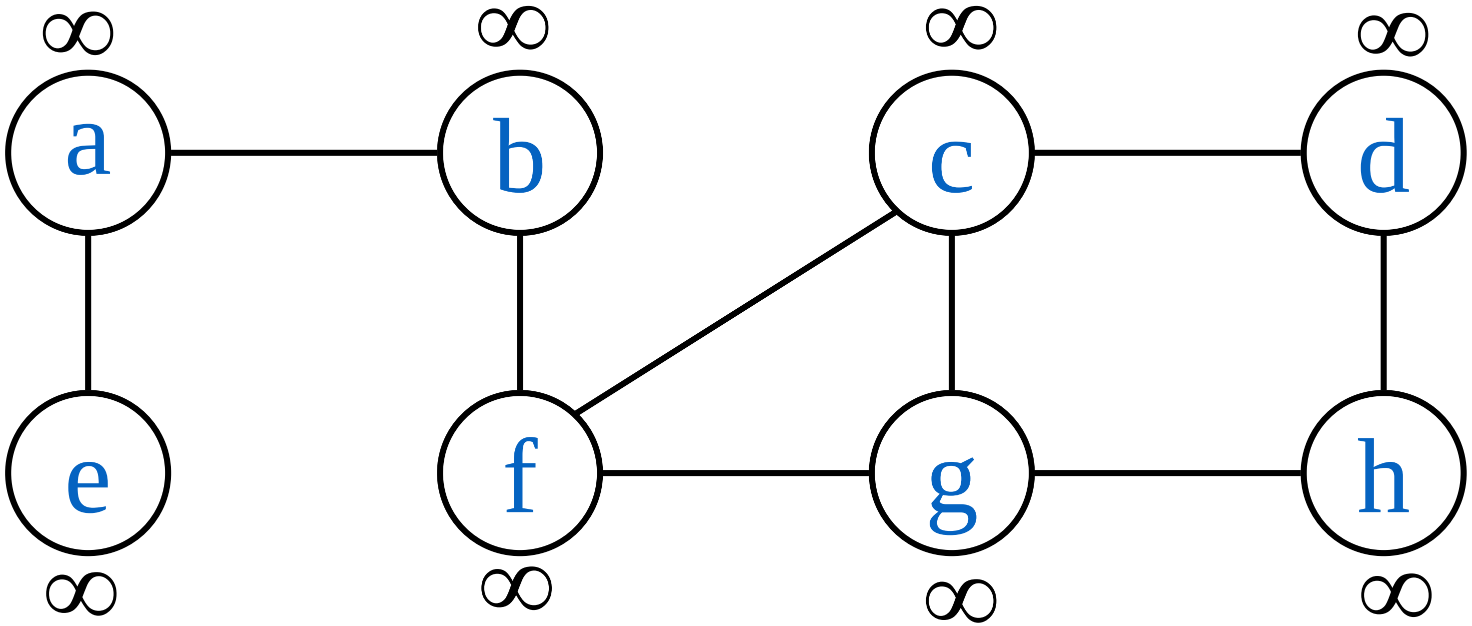
distance
(pre)

- ~~How~~ is the graphs stored?
- Suppose we call BFS(G, b)



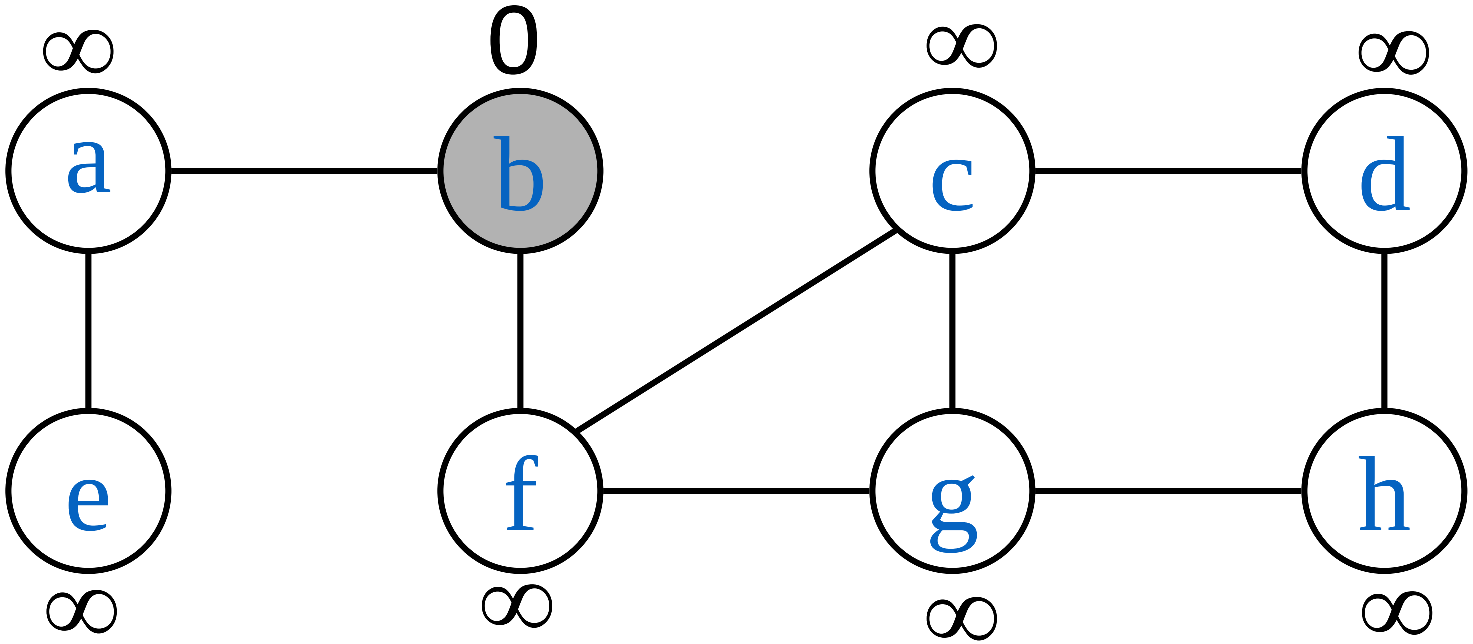
Breadth-First Search: Example

Initialization



Breadth-First Search: Example

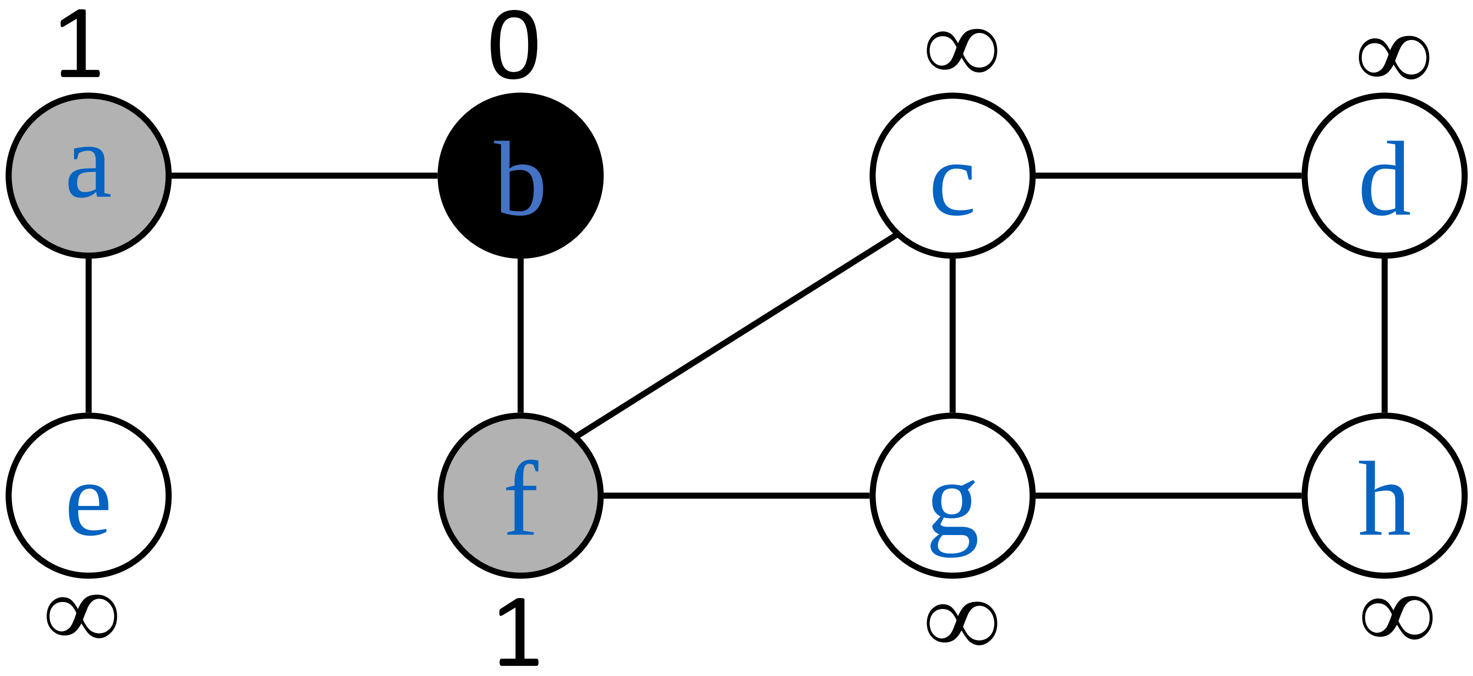
Iteration (i)



Q: b

Breadth-First Search: Example

Iteration (ii)

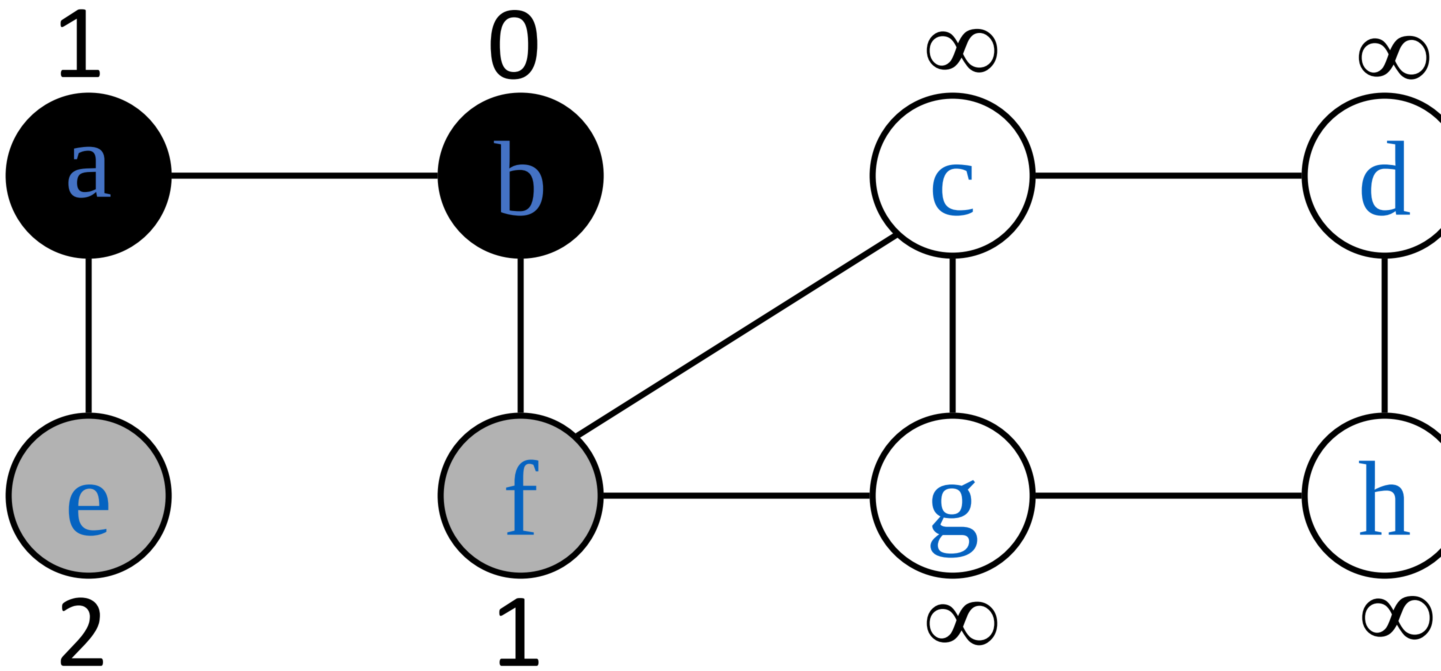


Q:

a	f

Breadth-First Search: Example

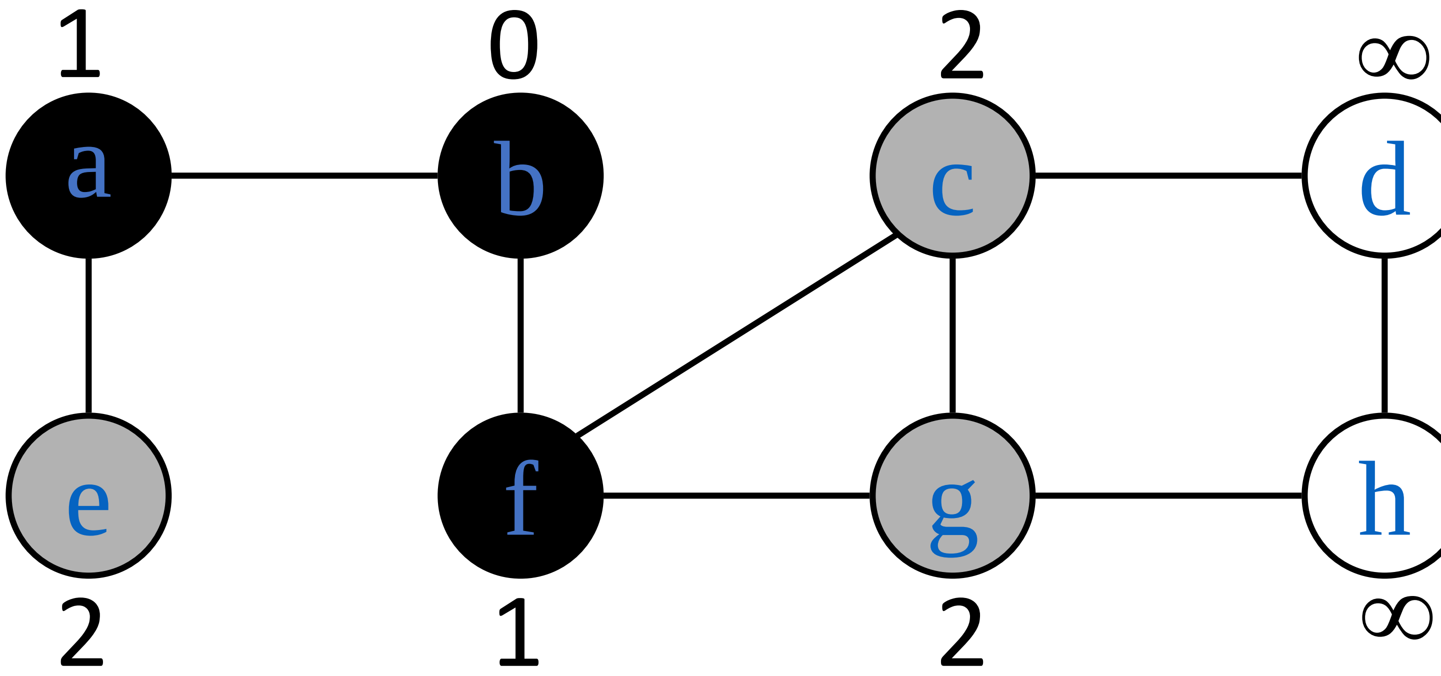
Iteration (iii)



Q: | f | e |

Breadth-First Search: Example

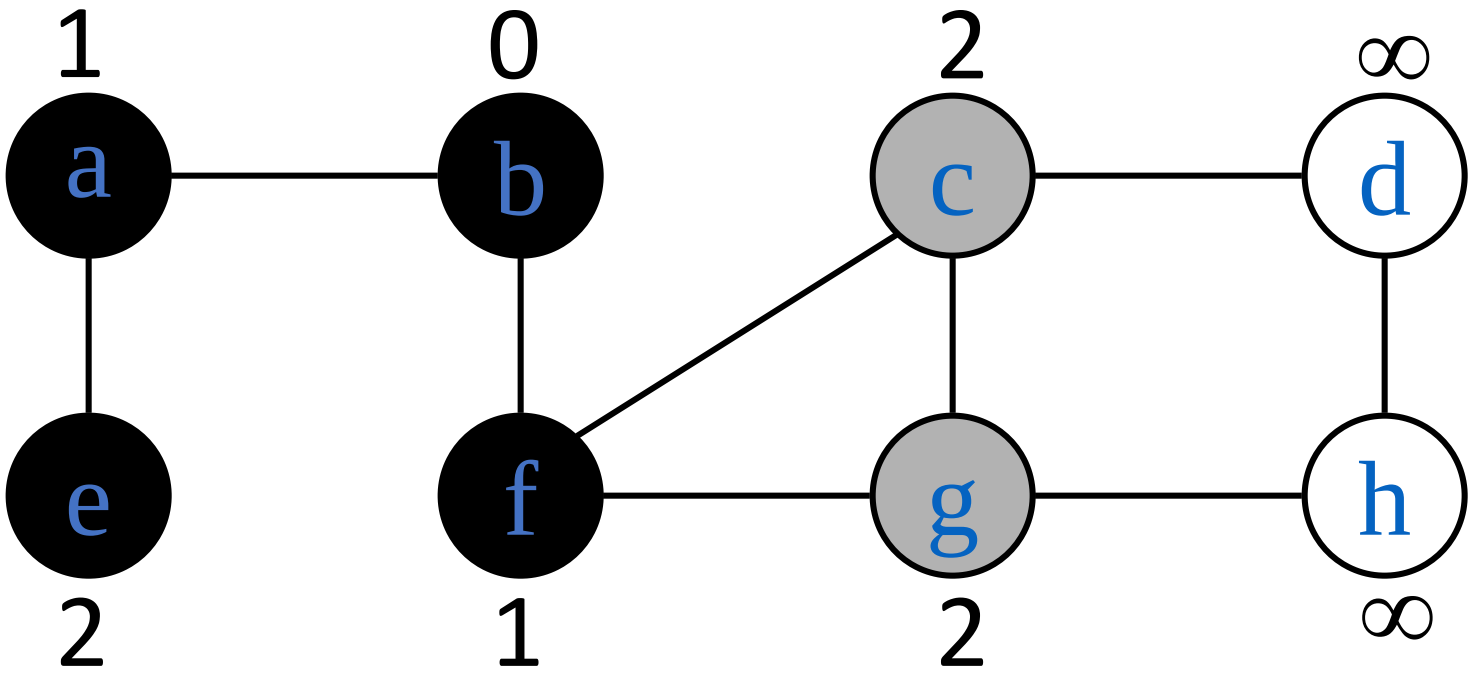
Iteration (iv)



Q:	e	c	g

Breadth-First Search: Example

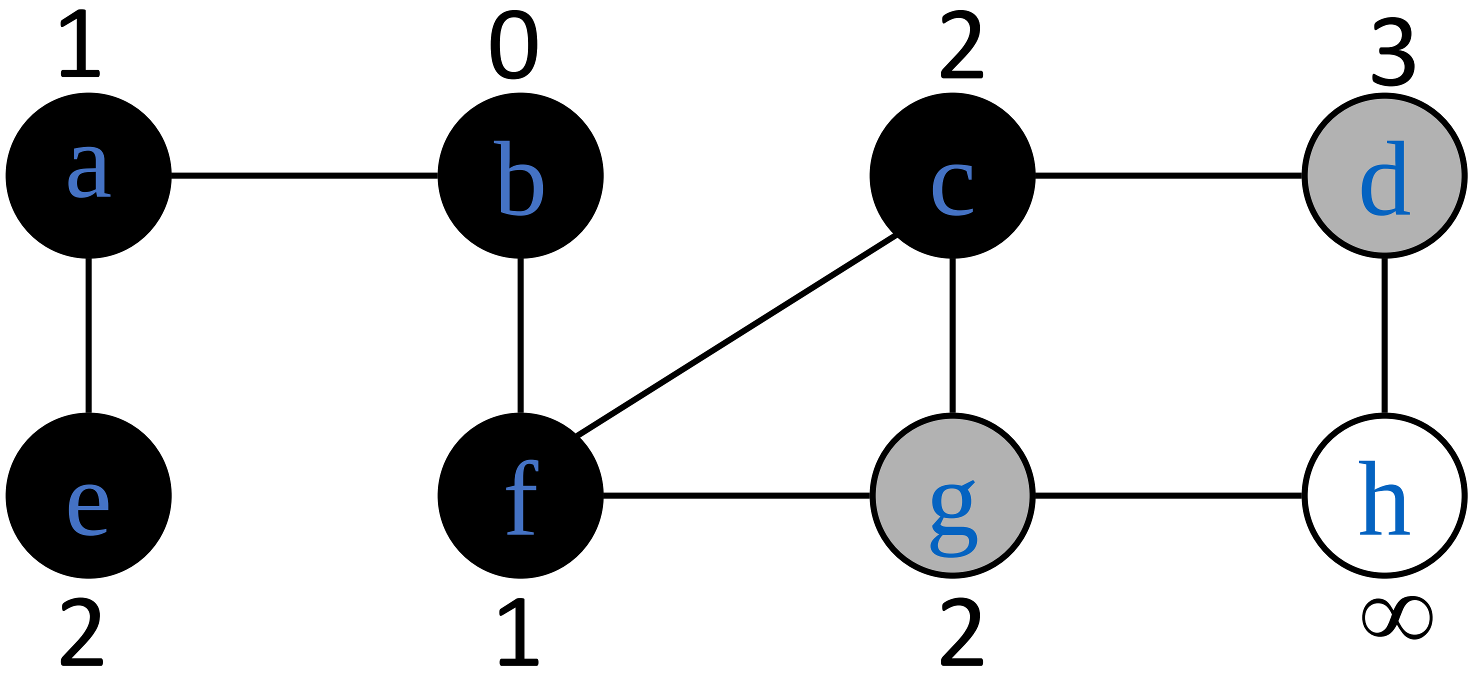
Iteration (v)



Q: | c | g |

Breadth-First Search: Example

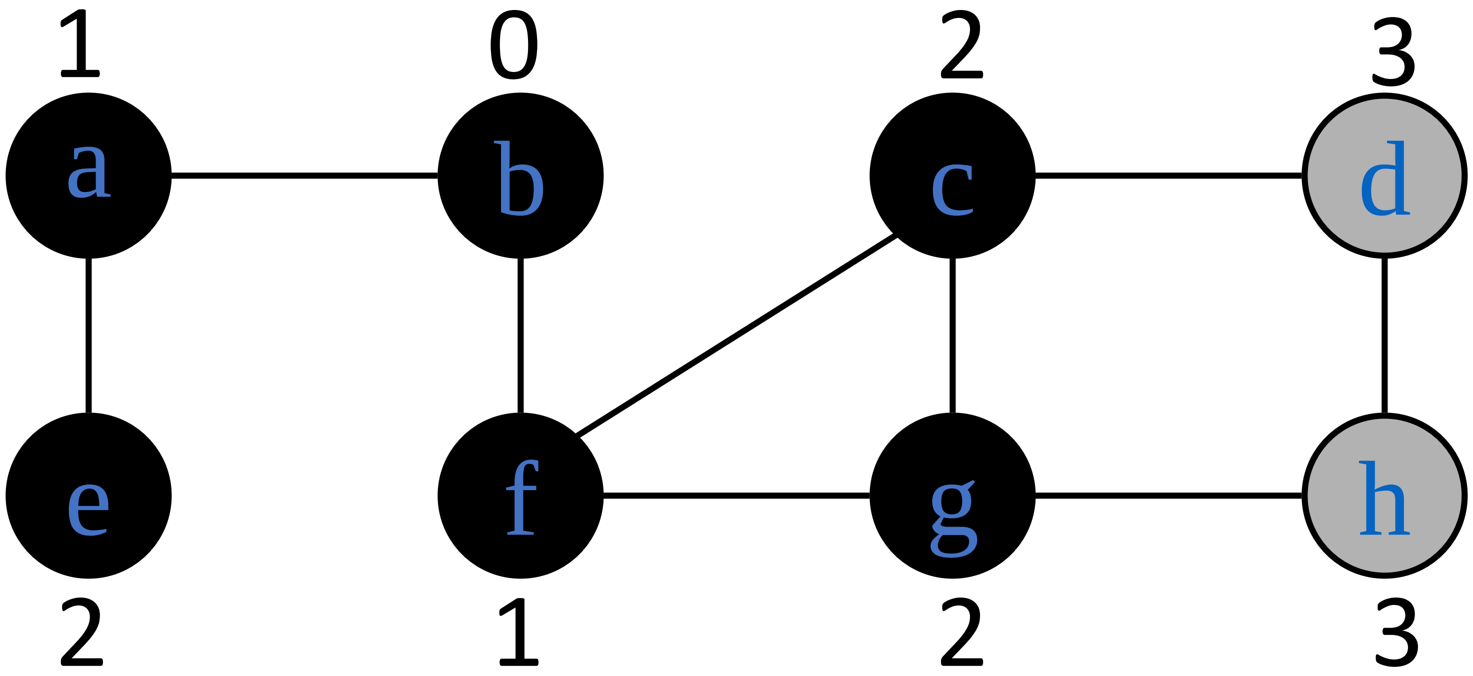
Iteration (vi)



Q: | g | d |

Breadth-First Search: Example

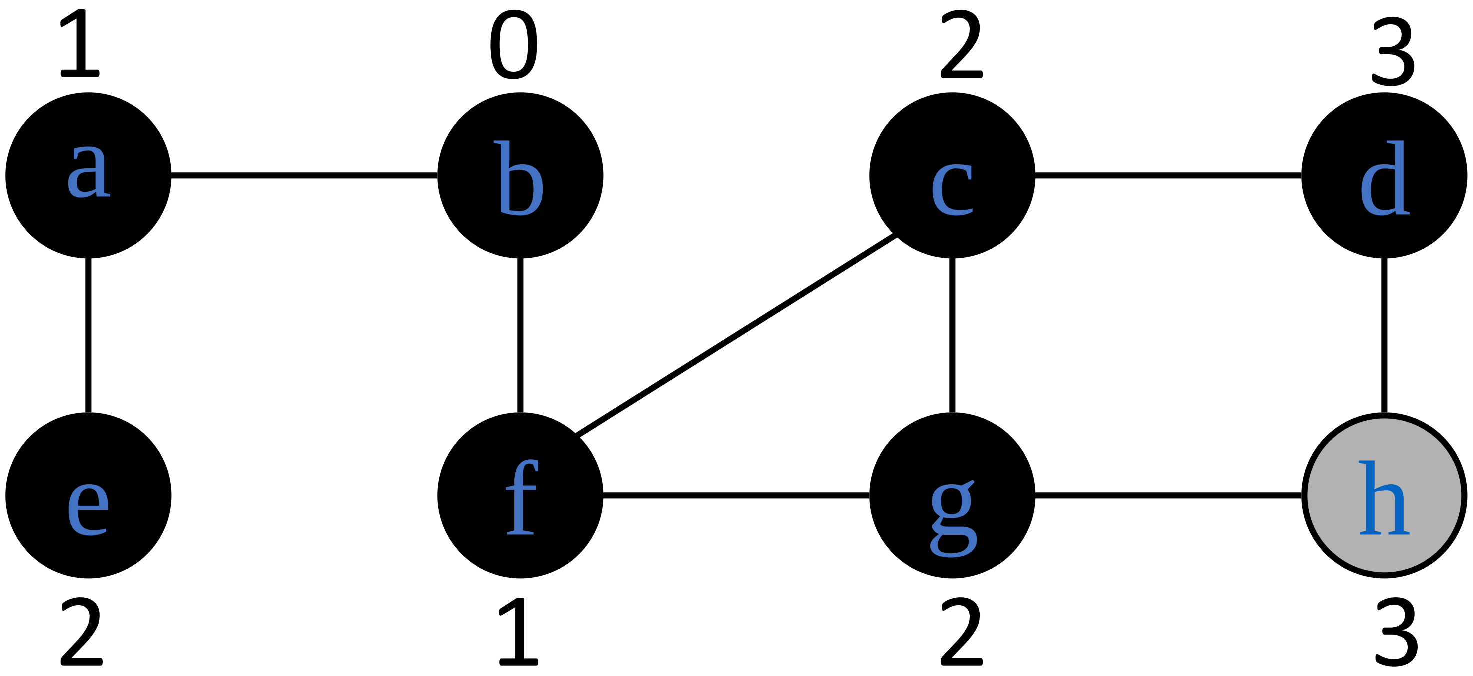
Iteration (vii)



Q: | d | h |

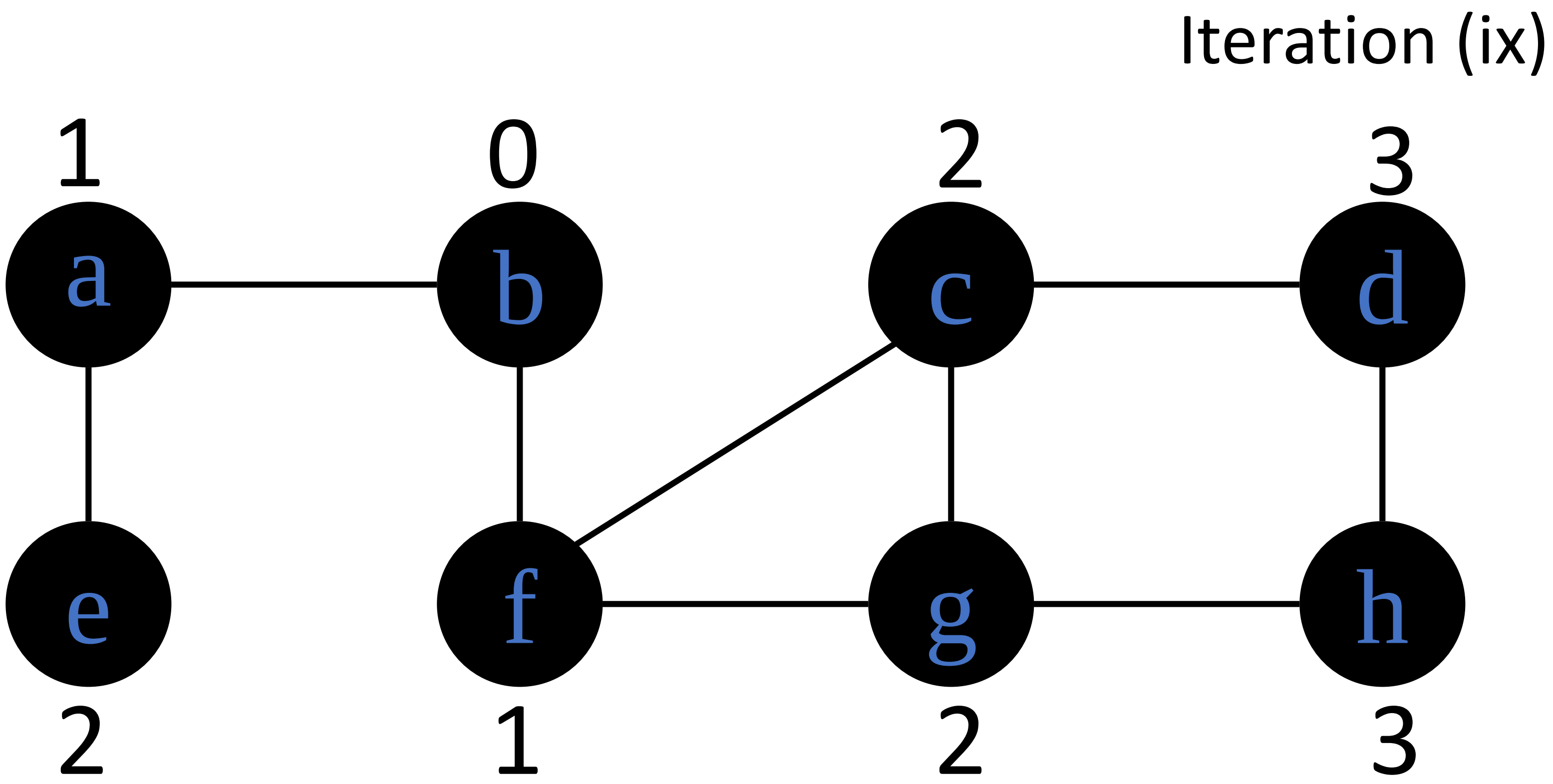
Breadth-First Search: Example

Iteration (viii)



Q: $\boxed{h}$

Breadth-First Search: Example



$Q: \quad \emptyset$

BFS - Properties

- Calculates the shortest path distance to source node
  - Shortest path is the minimum number of edges from *s* to reachable node *v* and ∞ if *v* is unreachable
  - Breadth-first tree represents the shortest path

- Note: Slightly different variants of BFS are possible. For e.g., instead of assigning colors, we can simply maintain a list of explored vertices.

Depth First Search (DFS)

- Depth First Search (DFS) is another strategy for exploring a graph
  - Explore "deeper" in graph whenever possible
  - Edges are explored out of the most recently discovered vertex *v* that still has unexplored edges
  - When all of *v*'s edges have been explored, backtrack to the vertex from which *v* was discovered
  - Continue process until all reachable vertices from current source vertex are discovered
  - Repeat the search procedure for other undiscovered vertices, if any.

DFS – Setup

- Vertices are assigned colors: **`color[v]`**
  - White → indicates undiscovered vertex. Initially all vertices are white
  - Gray → indicates that vertex has been discovered, but not fully explored
  - Black → indicates a fully explored vertex

- Predecessor of vertex is stored: **`pre[v]`**
  - Predecessor sub-graph of DFS forms a forest $\equiv$ multiple, ≥ 1, tree
  - (recall: predecessor sub-graph of BFS forms tree)

DFS – Setup (2)

iteration number

- Timestamps also maintained for each vertex
  - Each vertex *v* has two timestamps
  - `d[v]`: records timestamp when *v* is discovered, i.e. Vertex *v* made gray
  - `f[v]`: records timestamp when search has finished examining *v*’s adjacency list , i.e. Vertex *v* made black

DFS algorithm

DFS takes as input Graph G

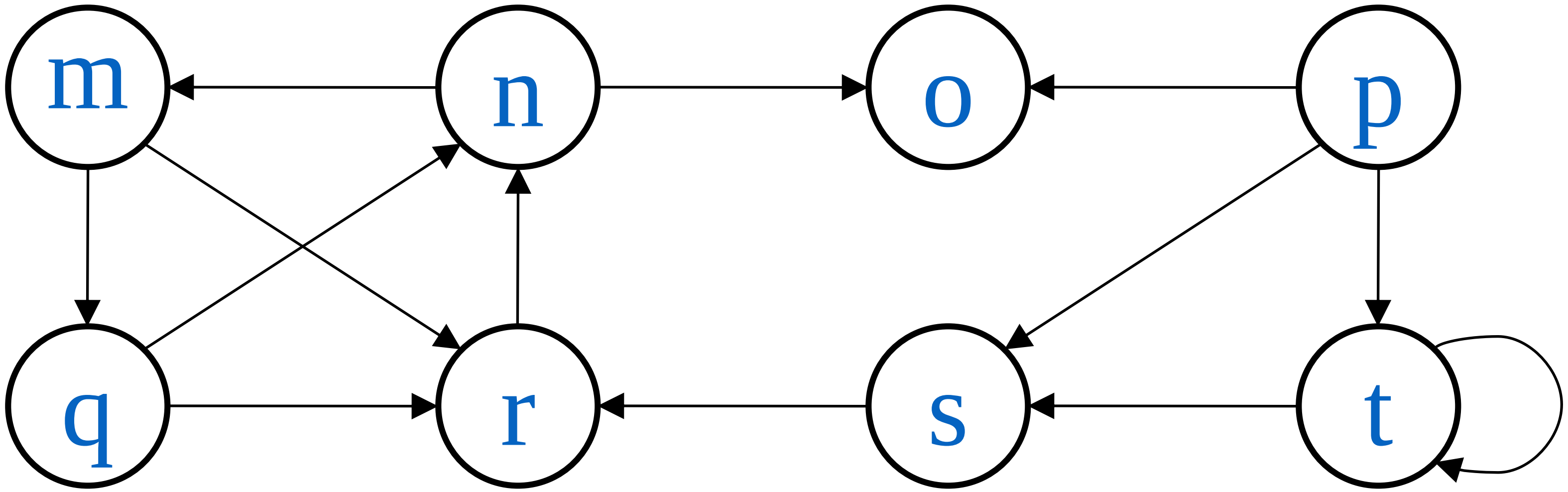
```
DFS(G)
1.  For each v ∈ V[G]
2.      color[v] ← 'white'
3.      pre[v] ← NULL          initialisation
4.  Next
5.  time ← 1
6.  For each u ∈ V[G]
7.      if color[u] =='white'
8.          DFSVisit(G, u)
9.      endif
10. Next
```

```
DFSVisit(G, u):
1. color[u] ← 'grey'
2. time ← time + 1
3. d[u] ← time
4. For each v ∈ Adj[u]
5.      If (color[v] =='white')
6.          pre[v] ← u
7.          DFSVisit(G, v)
8.      endif
9. Next
10. color[v] ← 'black'
11. time ← time + 1
12. f[u] ← time
```

Depth-First Search: Example

- Consider Graph, G, given

G = { m : [q, r],
n : [m, o],
o : [ ],
p : [o, s, t],
q : [n, r],
r : [n],
s : [r],
t : [s, t] }

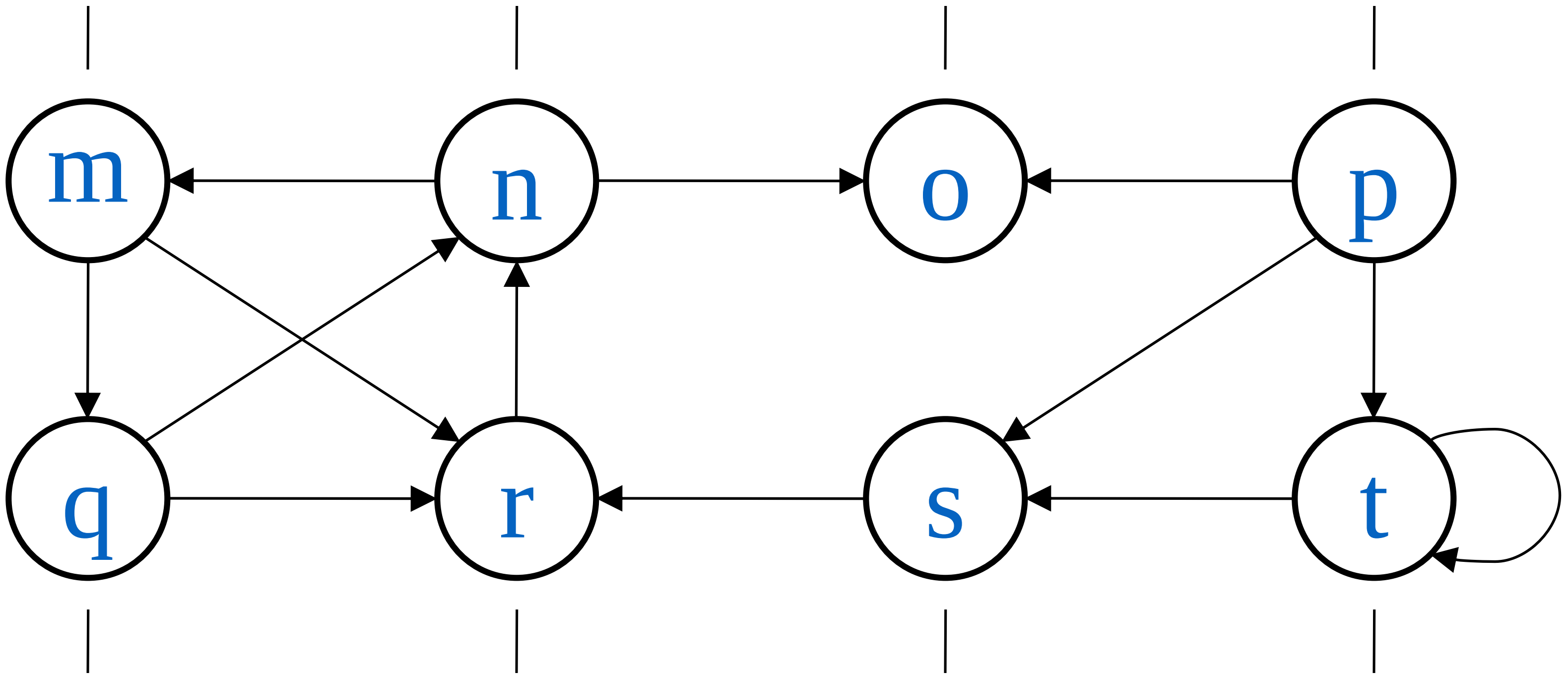


- How is the graphs stored?

- Suppose we call DFS(G)

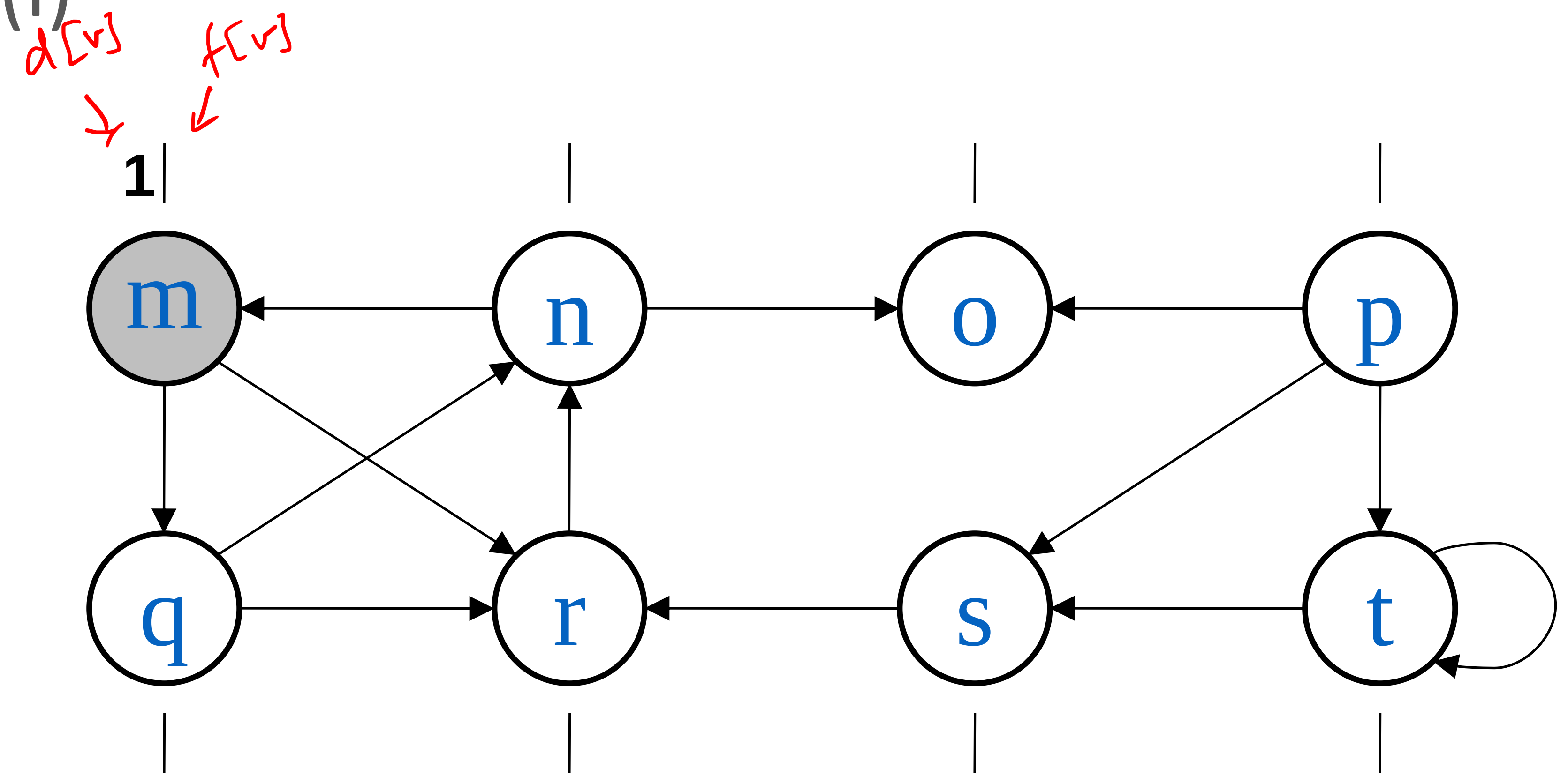
Depth-First Search: Example

- Initialization



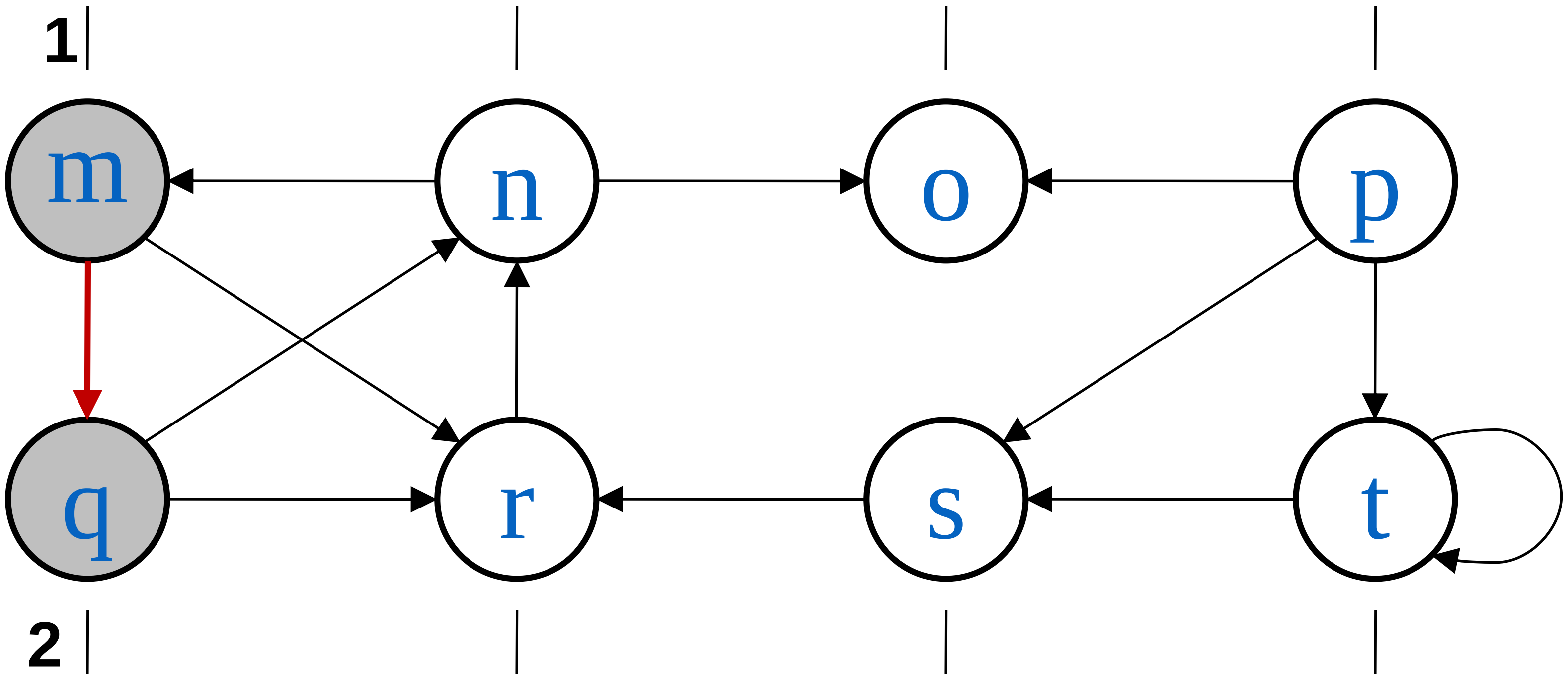
Depth-First Search: Example

- (i)



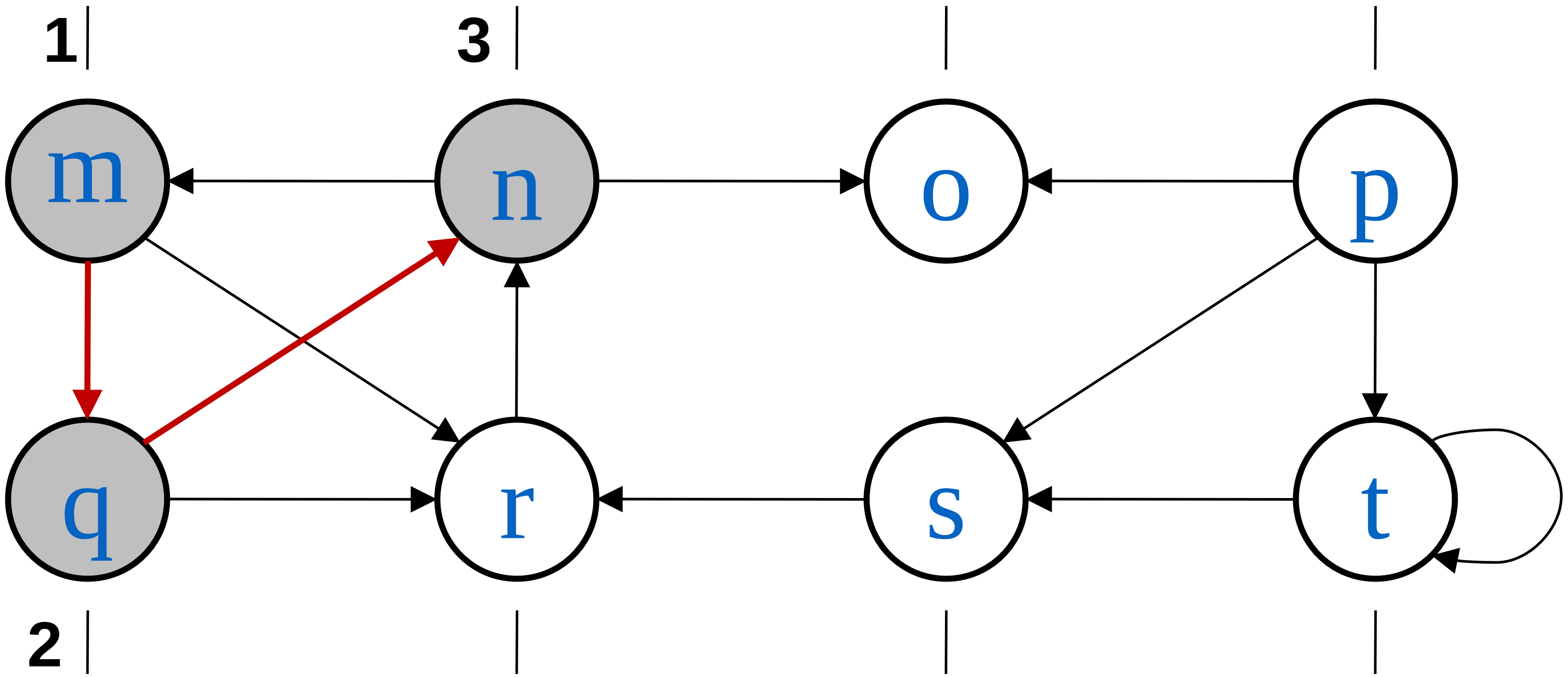
Depth-First Search: Example

- (ii)



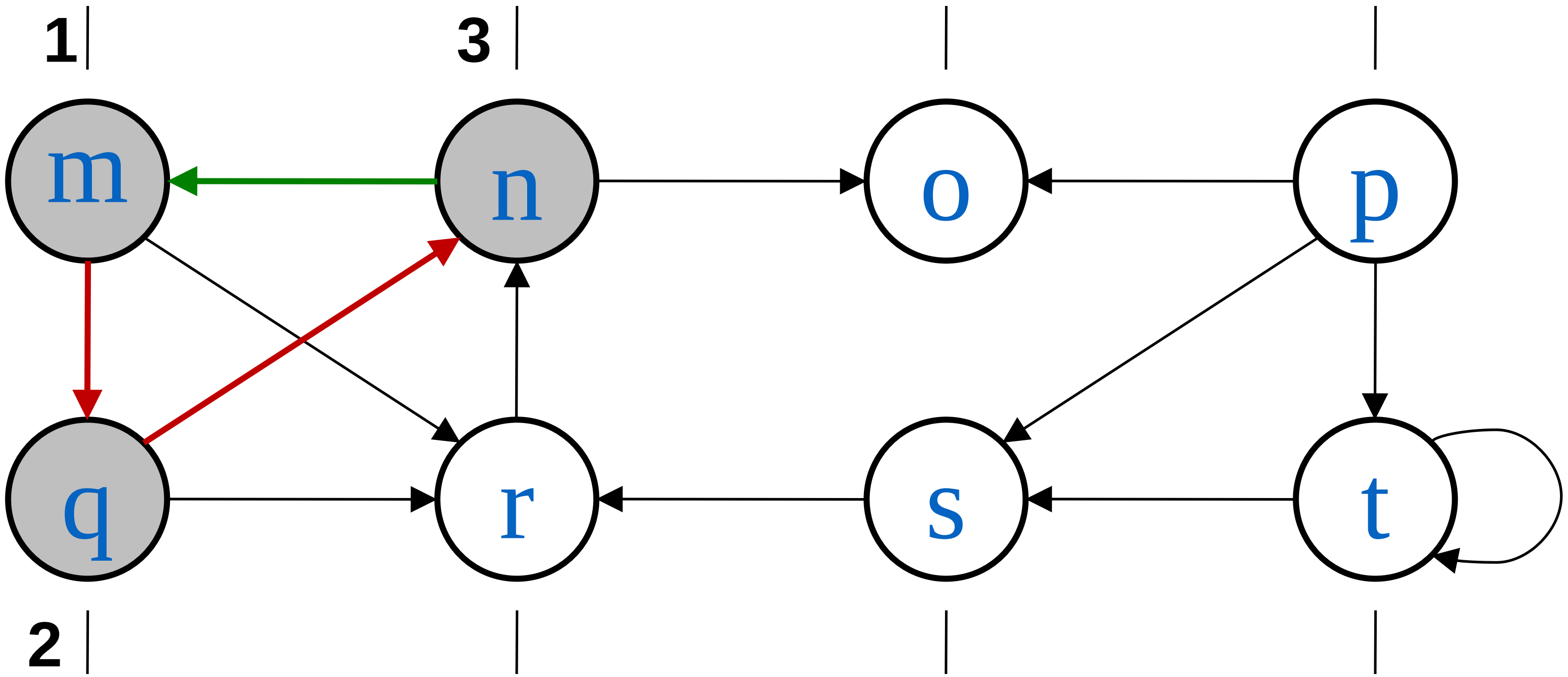
Depth-First Search: Example

- (iii)



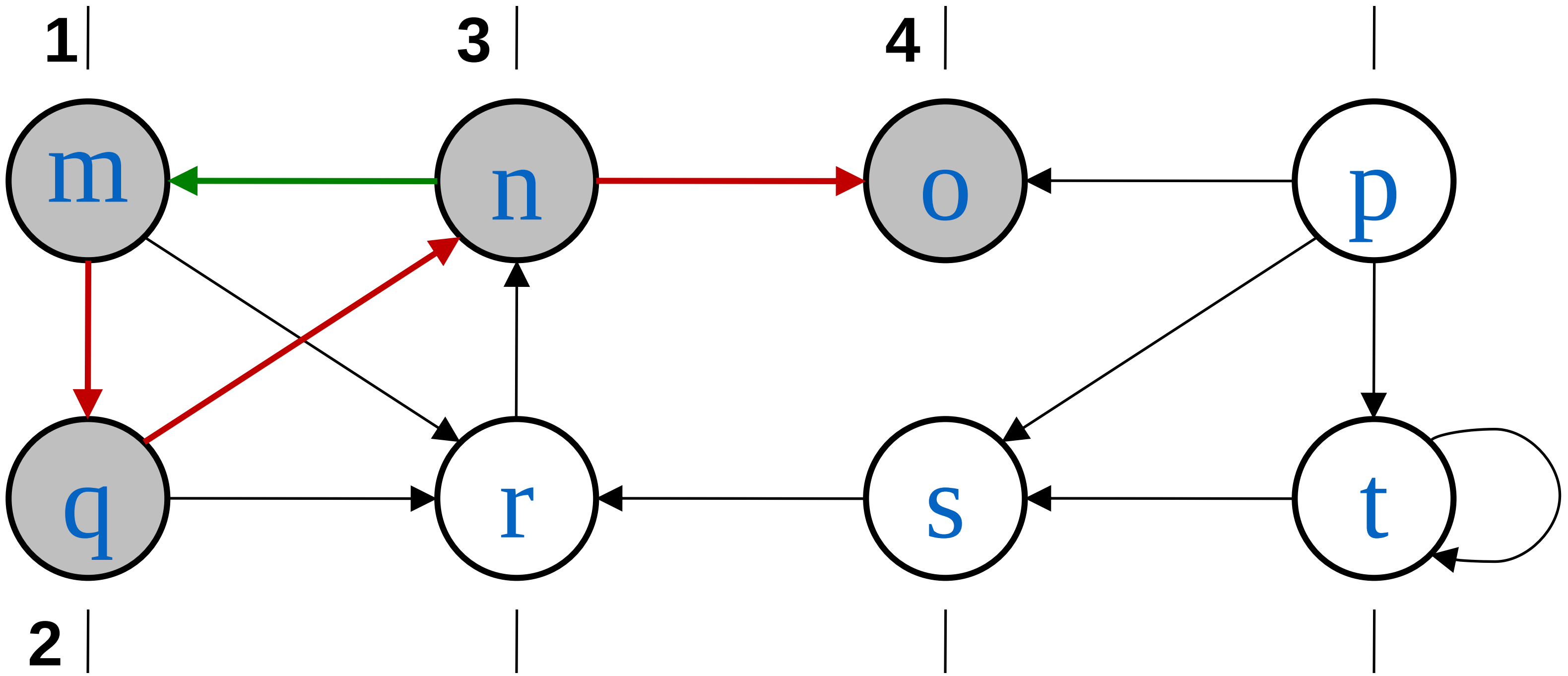
Depth-First Search: Example

- (iv)



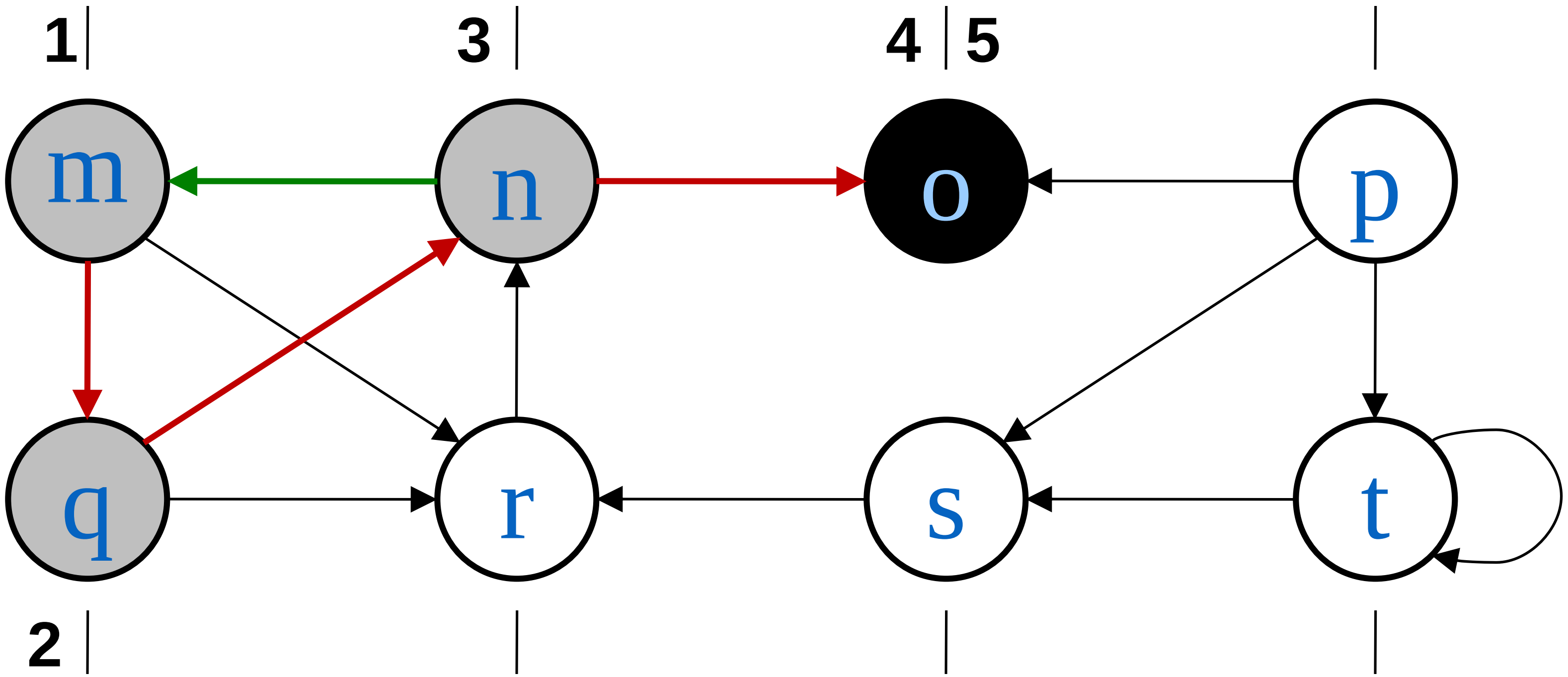
Depth-First Search: Example

- (v)



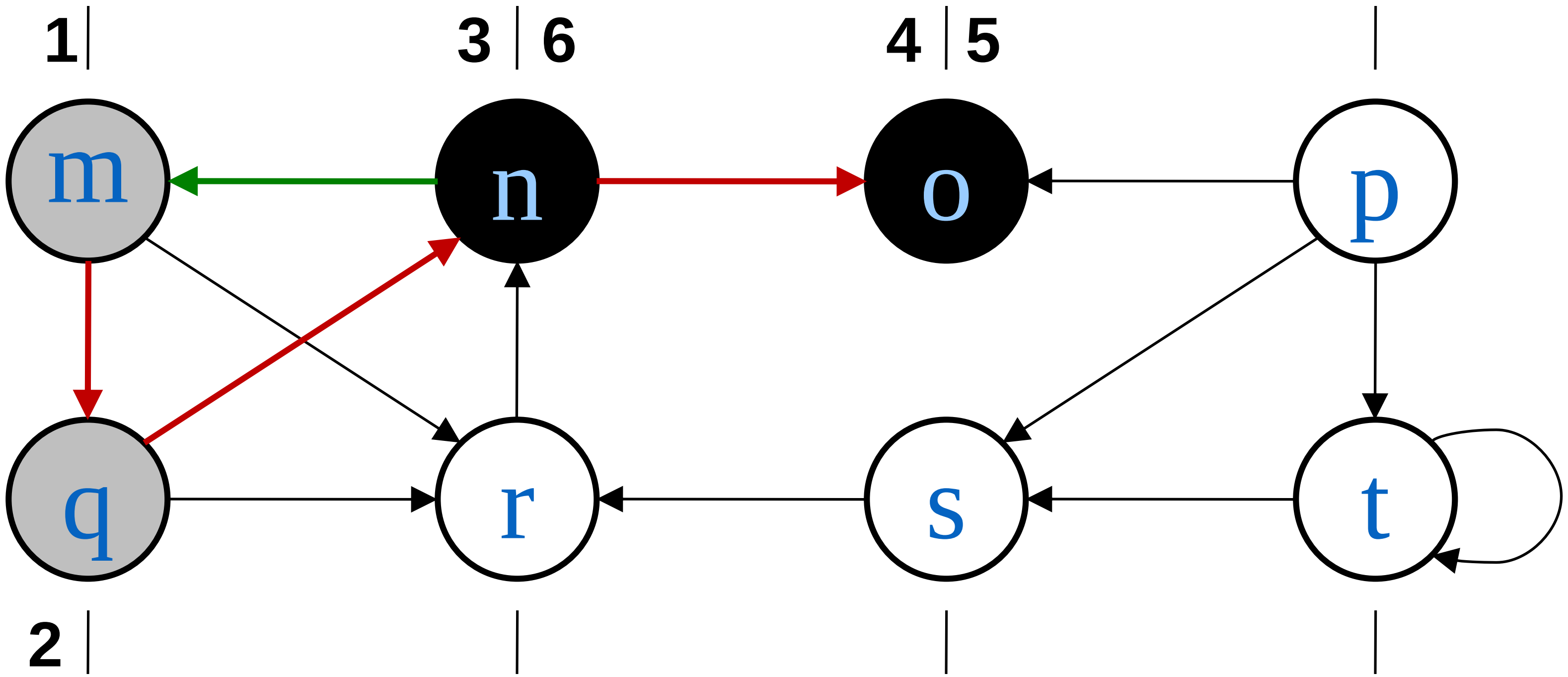
Depth-First Search: Example

- (vi)



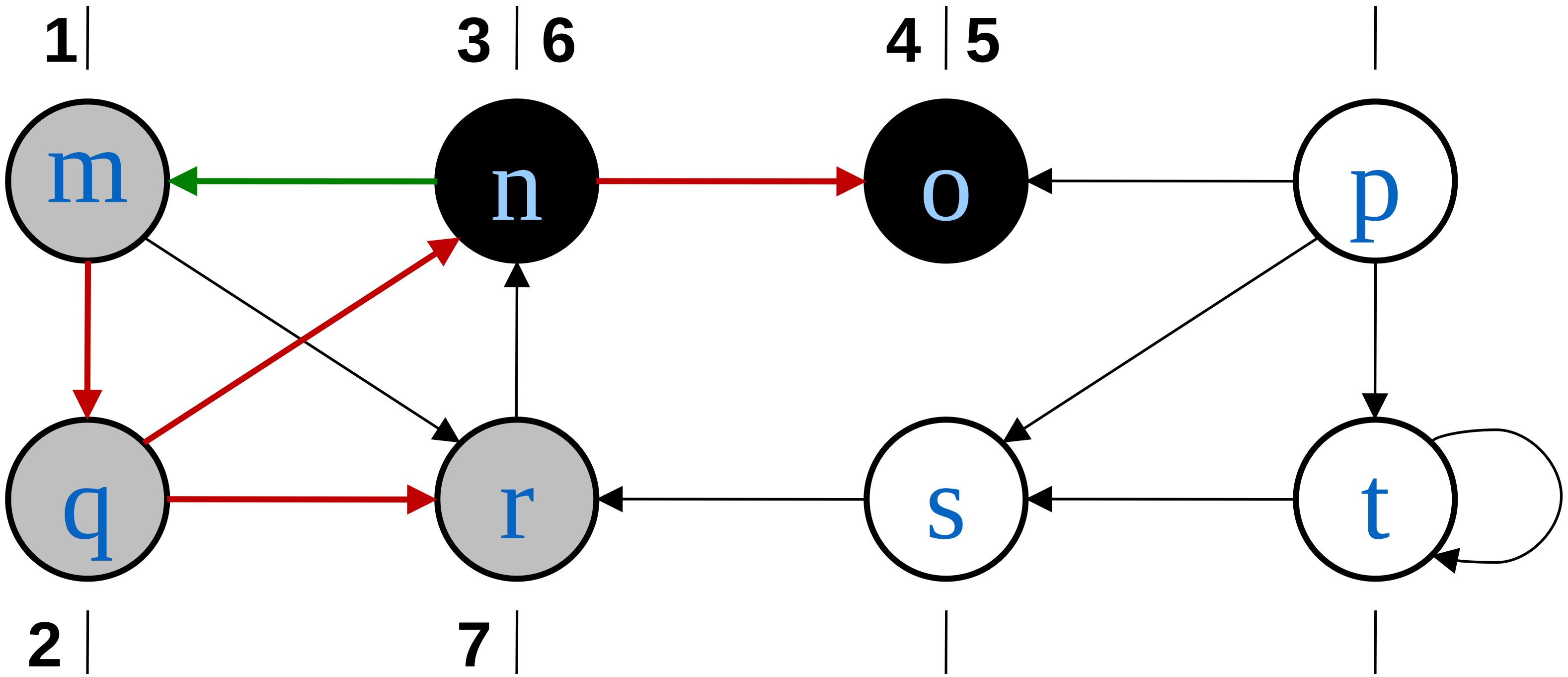
Depth-First Search: Example

- (vii)



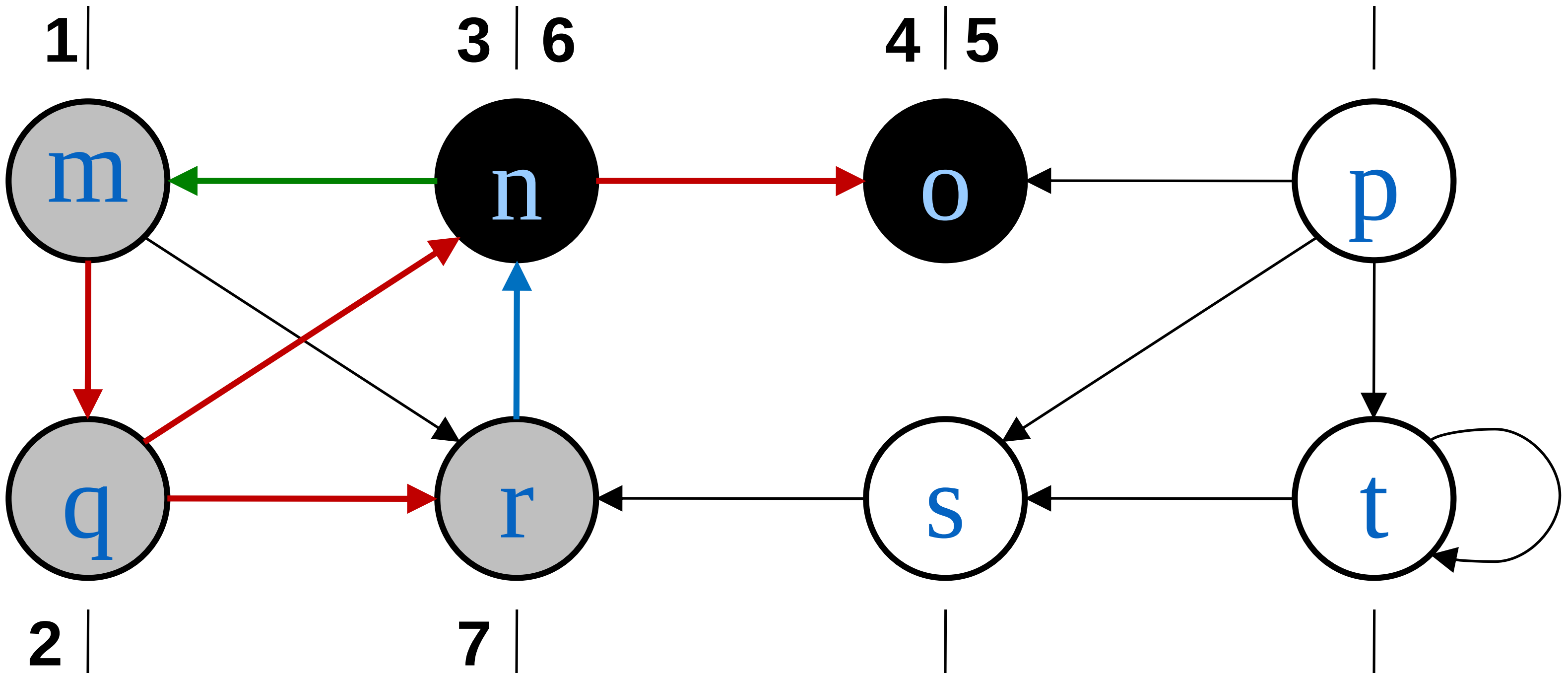
Depth-First Search: Example

- (viii)



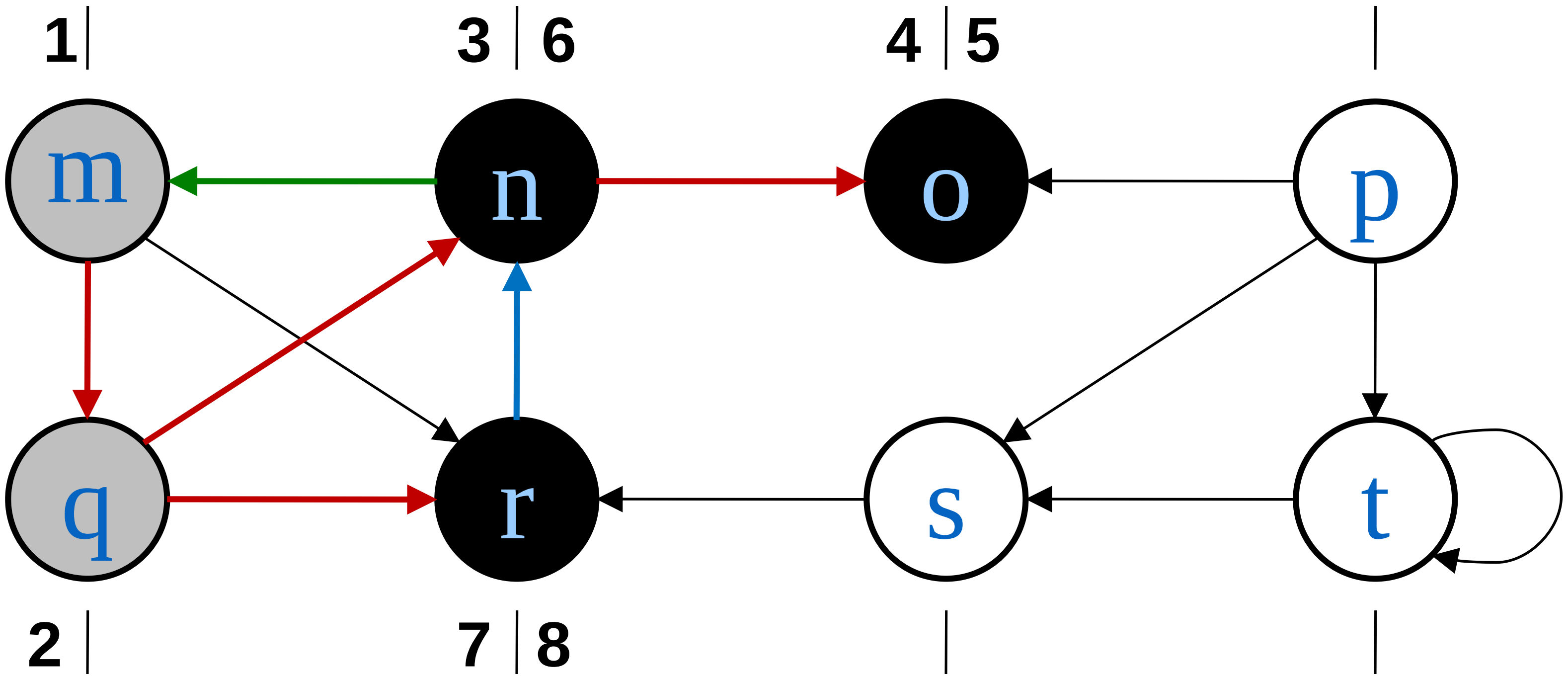
Depth-First Search: Example

- (ix)



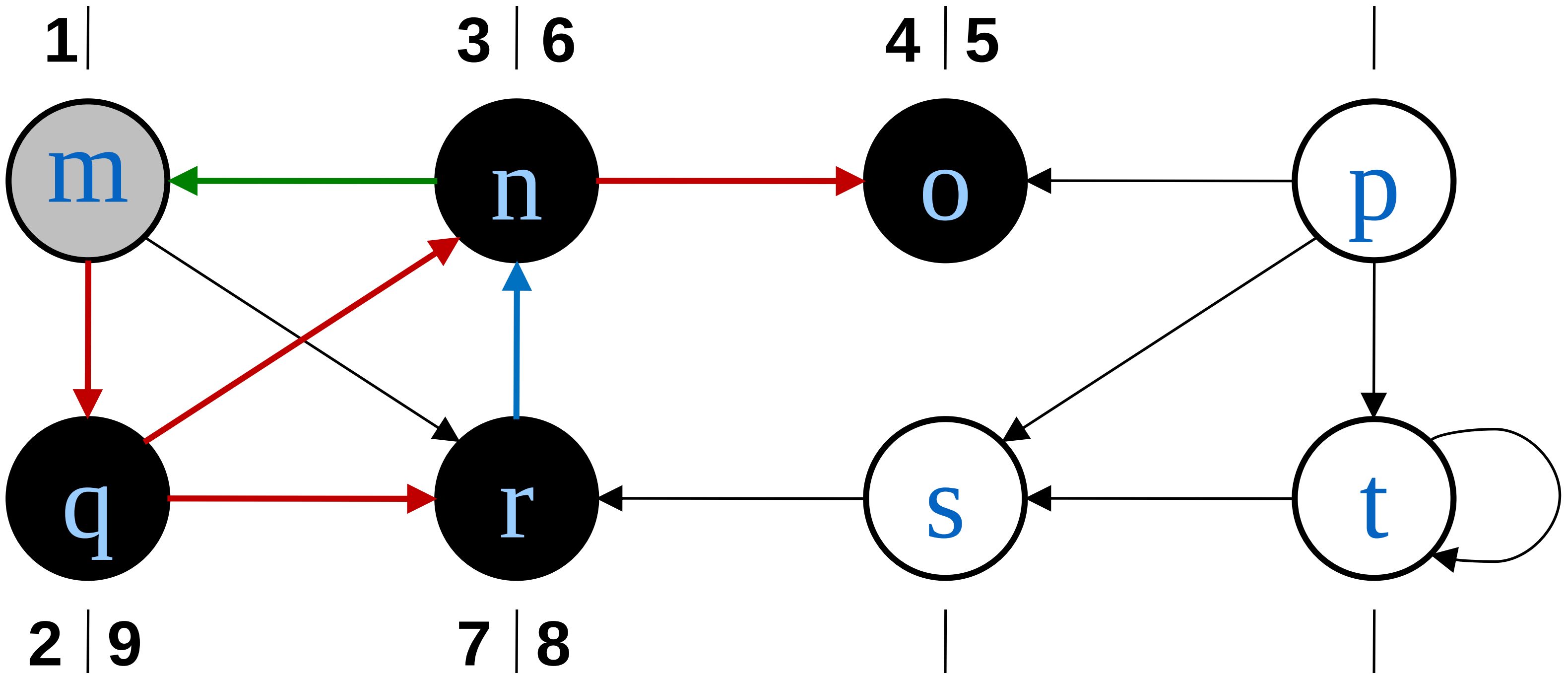
Depth-First Search: Example

- (x)



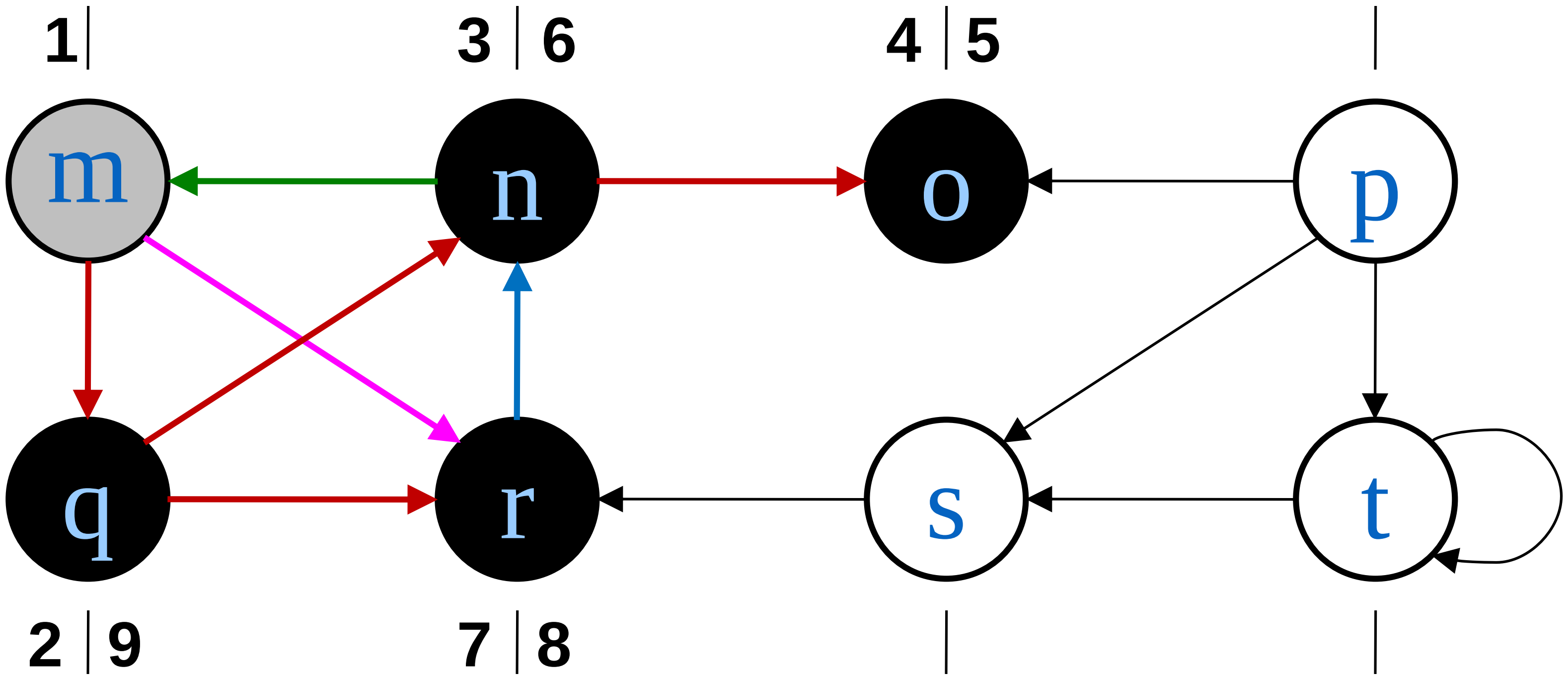
Depth-First Search: Example

- (xi)



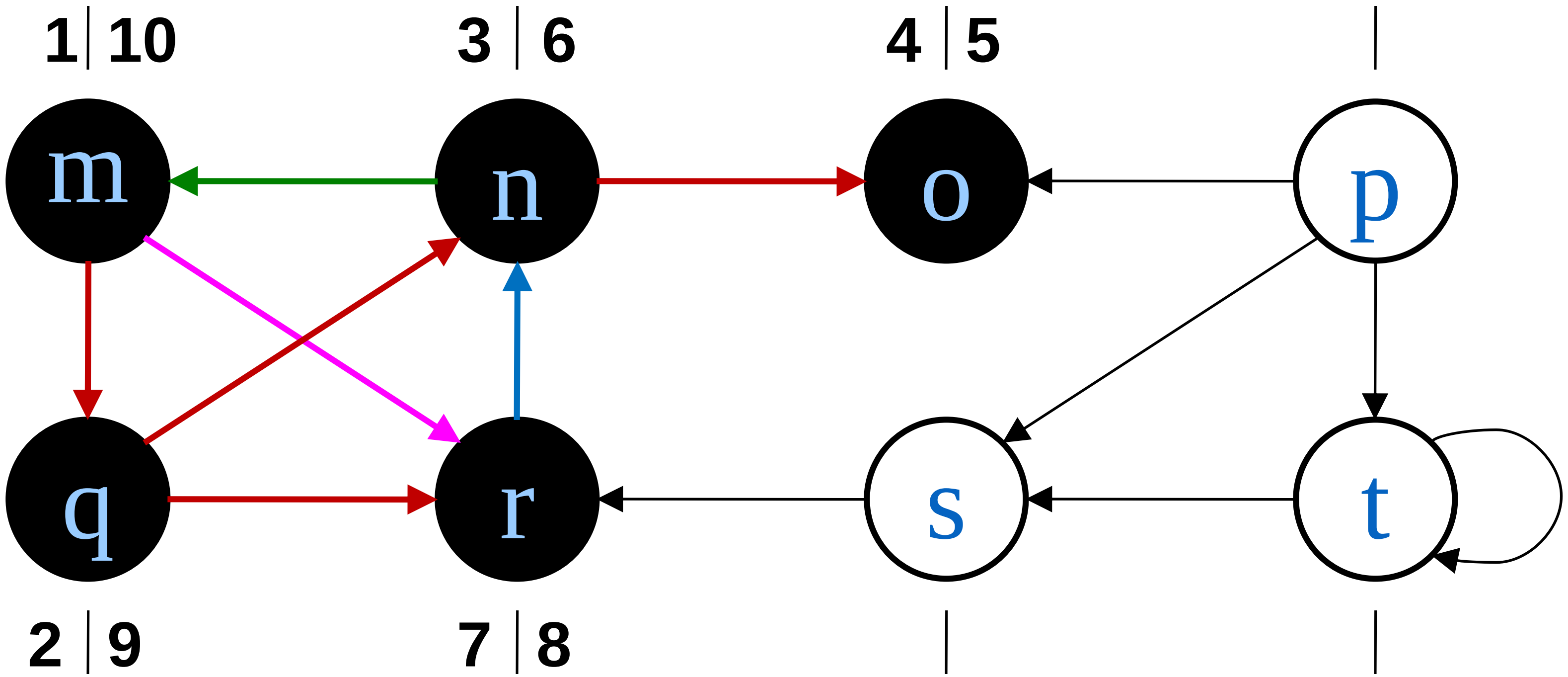
Depth-First Search: Example

- (xii)



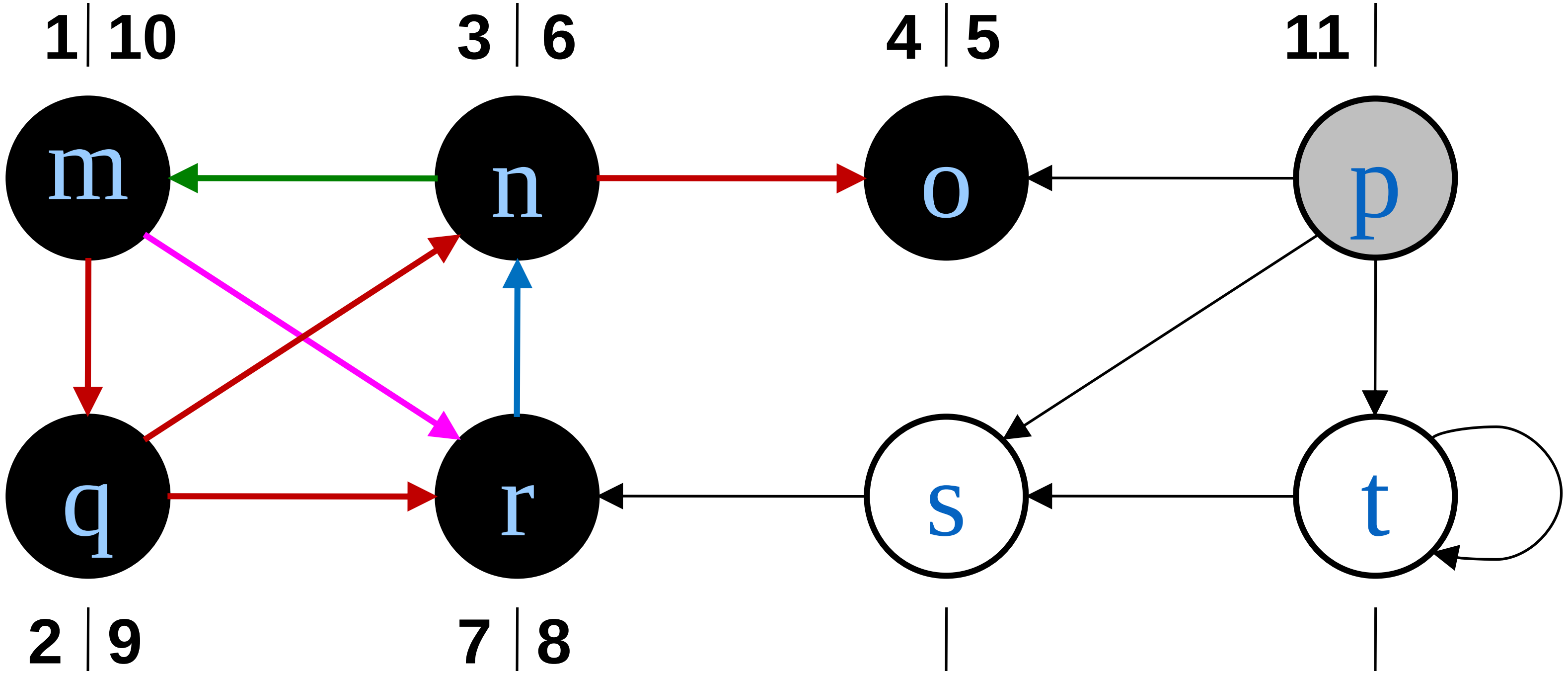
Depth-First Search: Example

- (xiii)



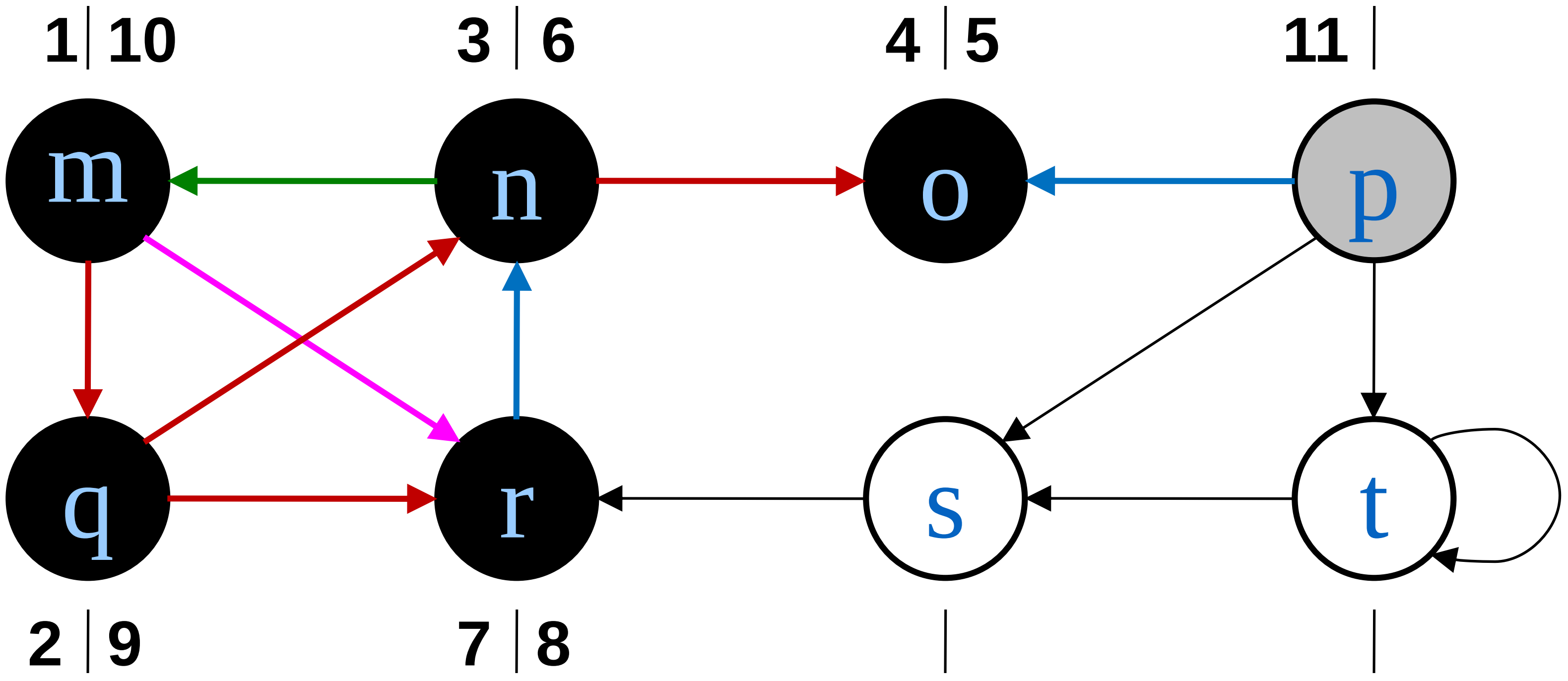
Depth-First Search: Example

- (xiv)



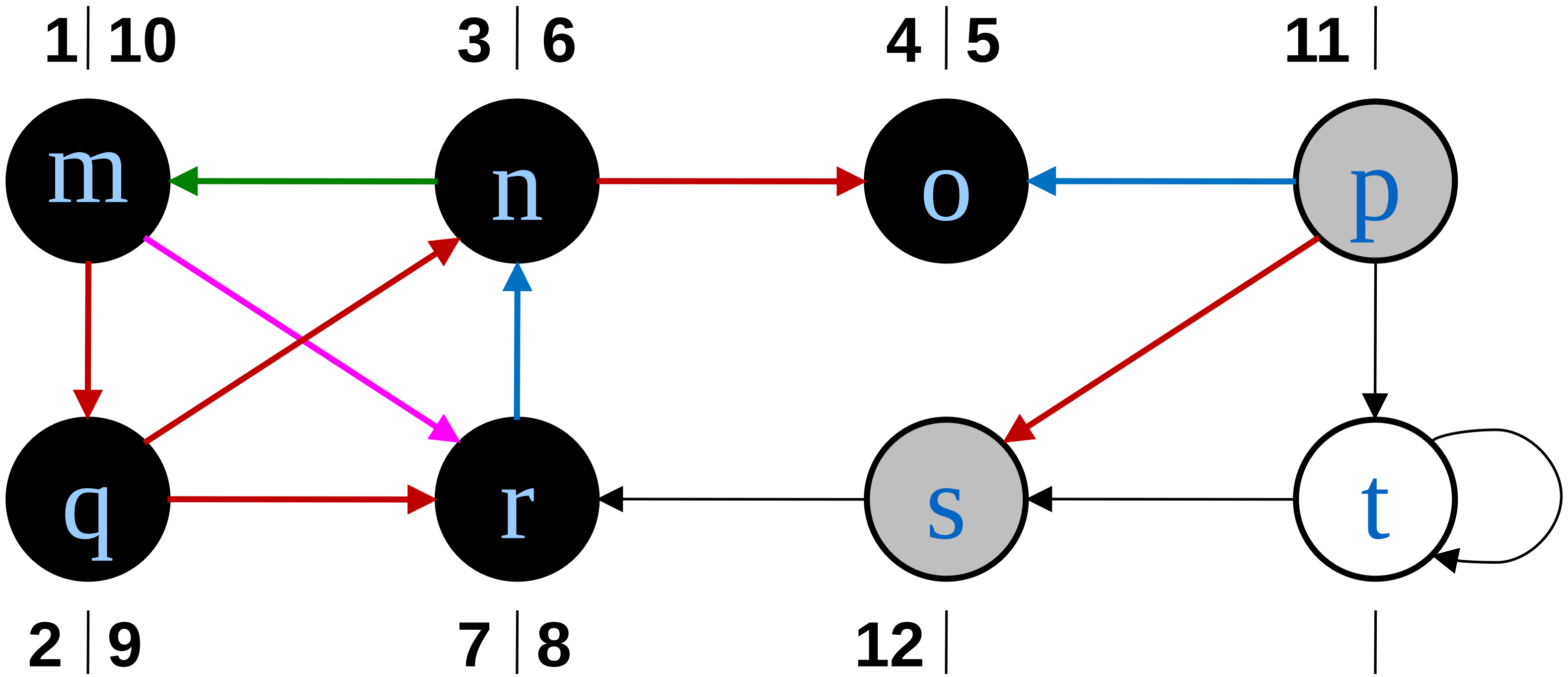
Depth-First Search: Example

- (xv)



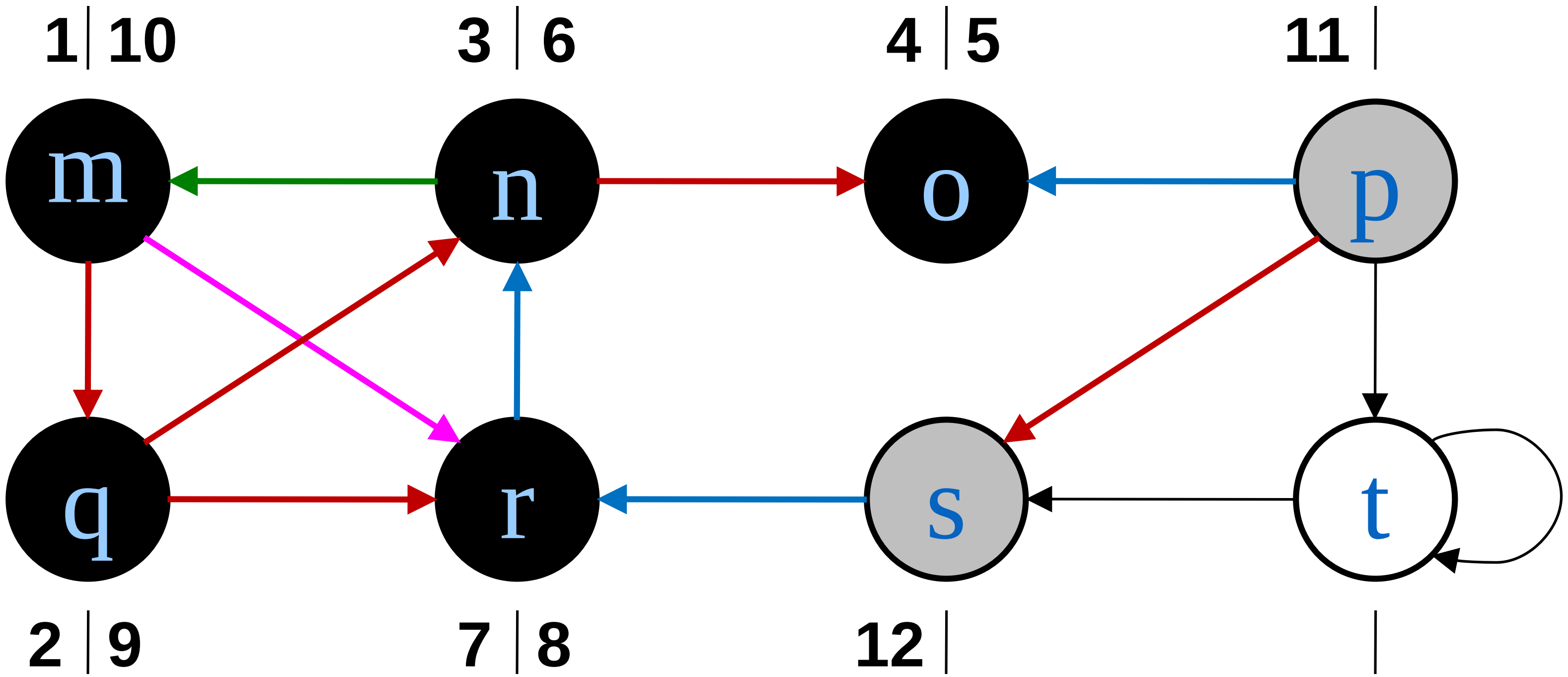
Depth-First Search: Example

- (xvi)



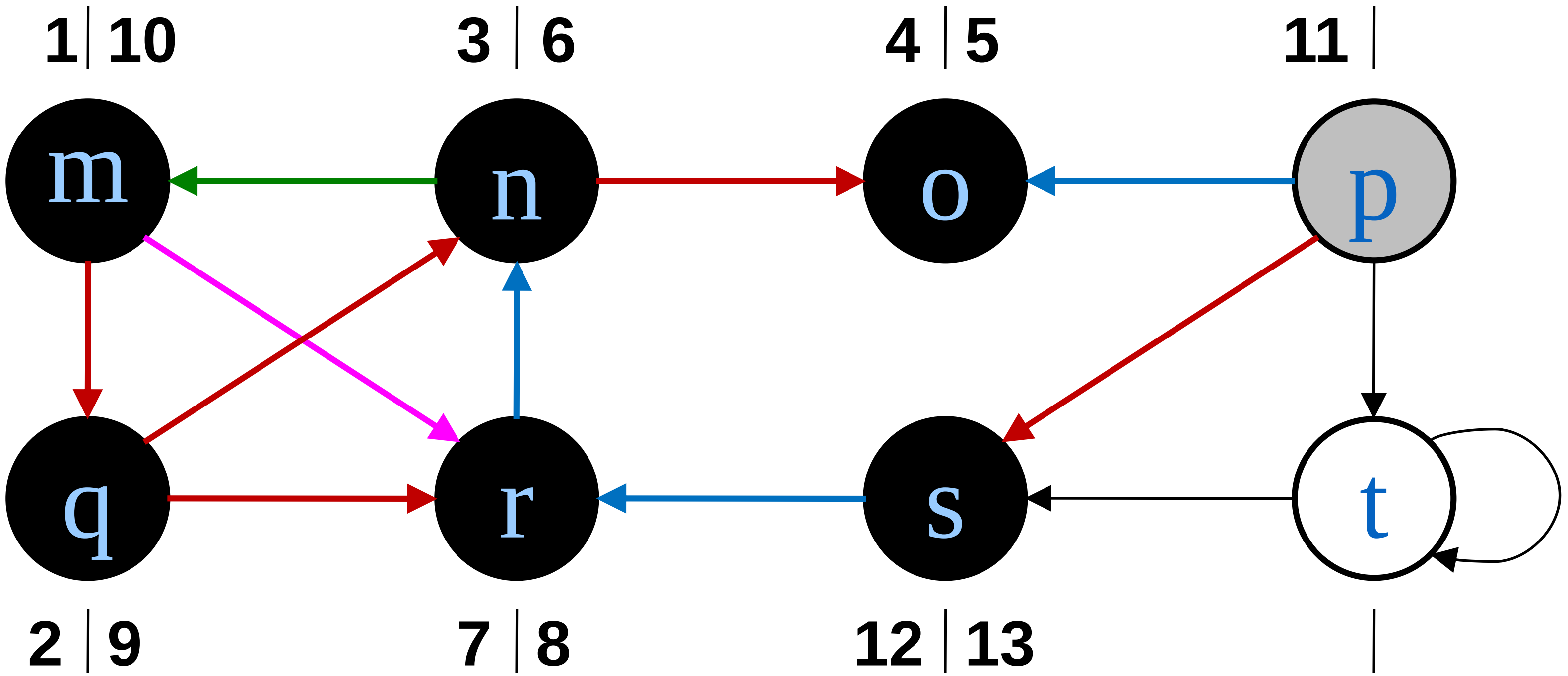
Depth-First Search: Example

- (xvii)



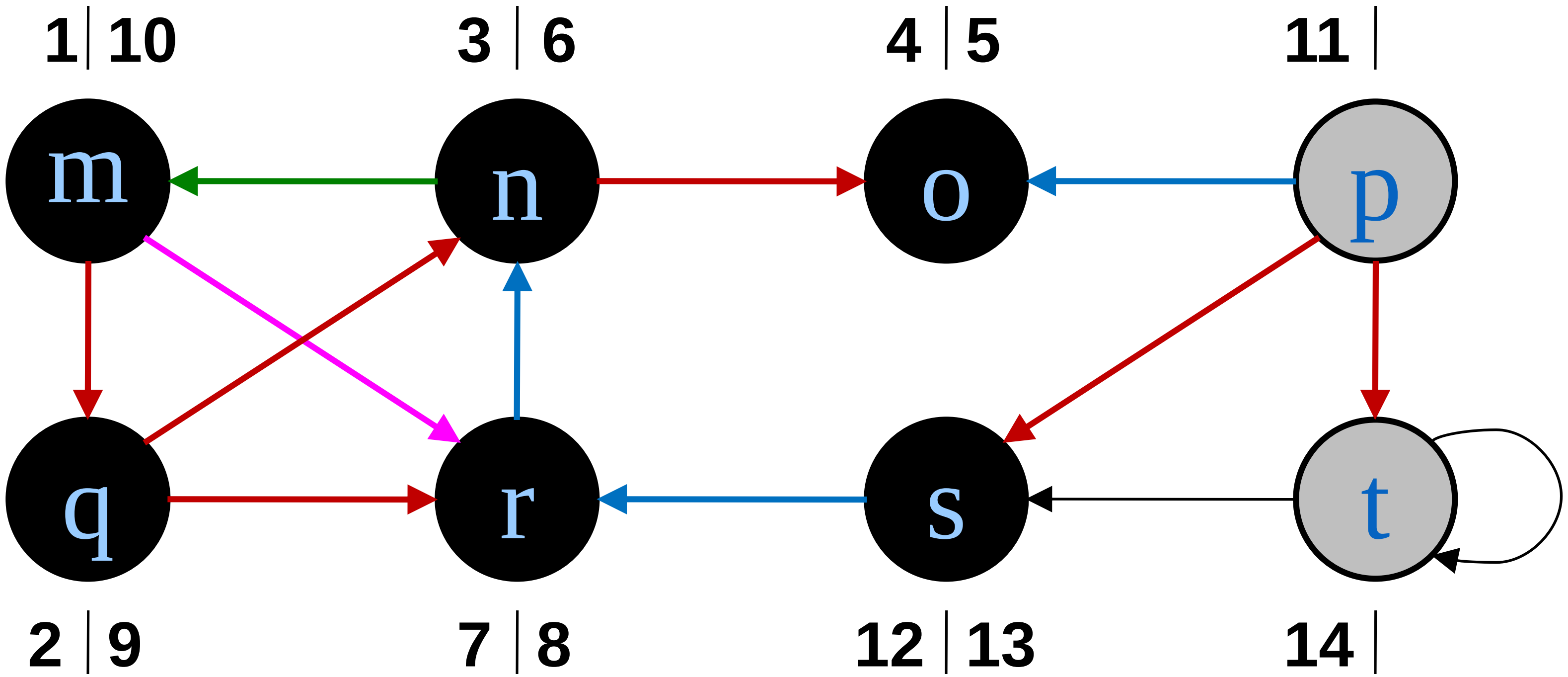
Depth-First Search: Example

- (xviii)



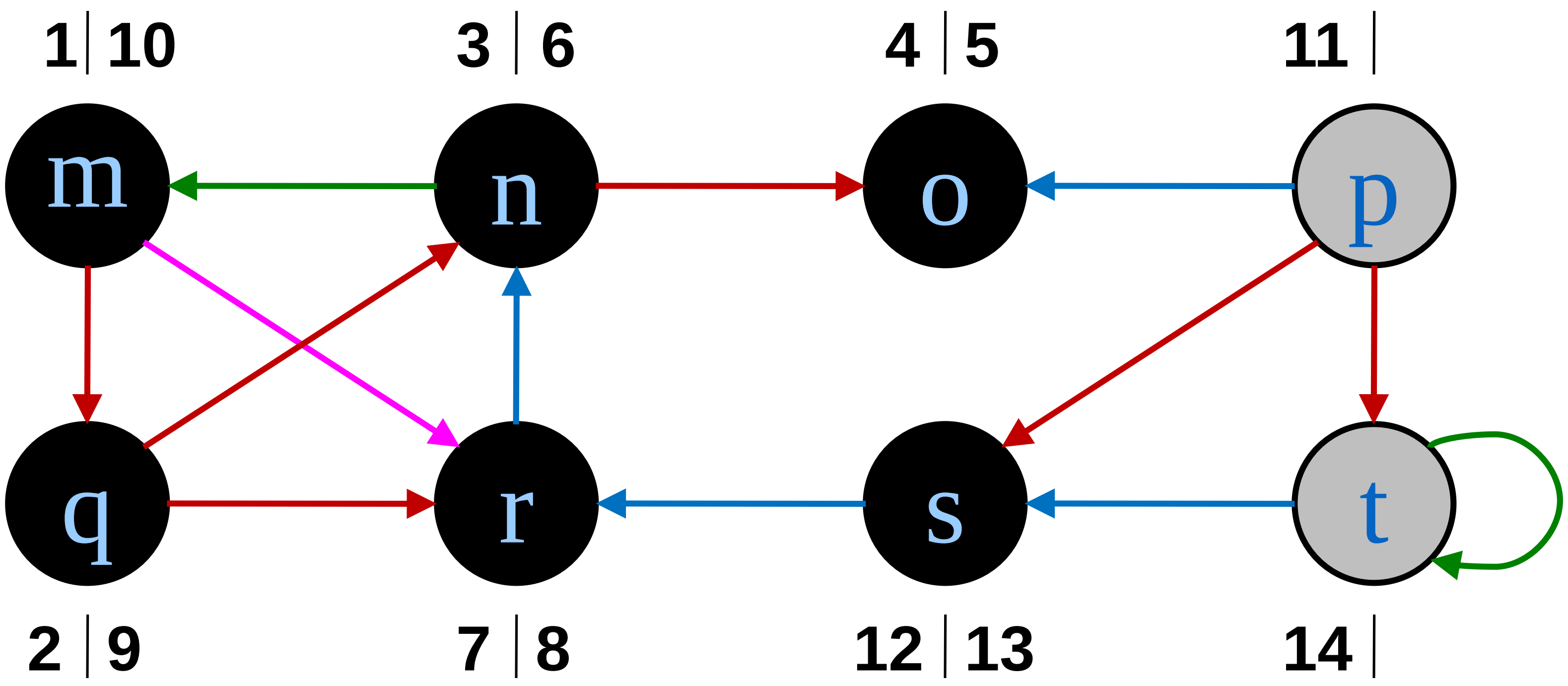
Depth-First Search: Example

- (xix)



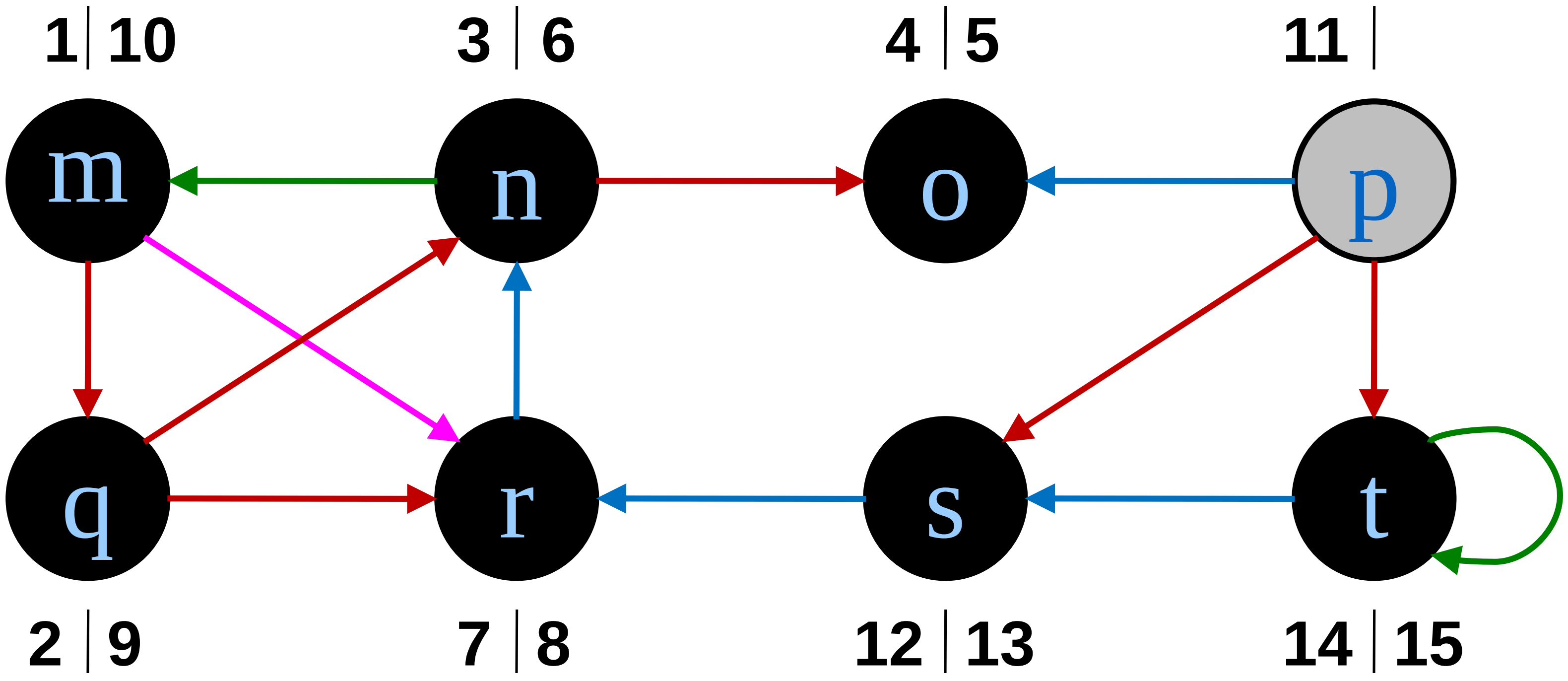
Depth-First Search: Example

- (xx, xxi)

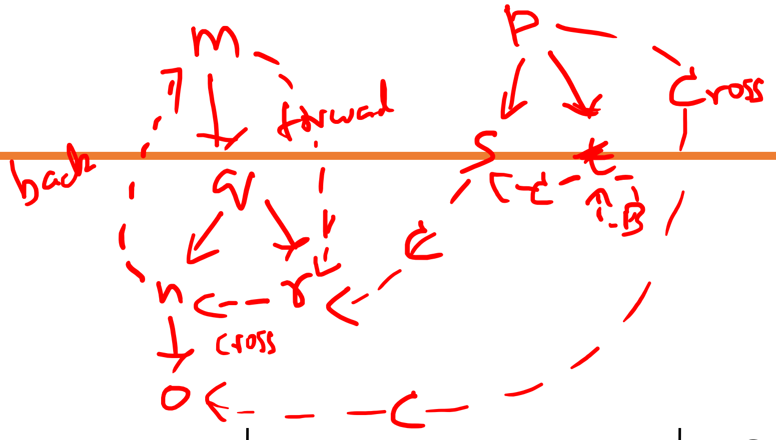


Depth-First Search: Example

- (xxii)

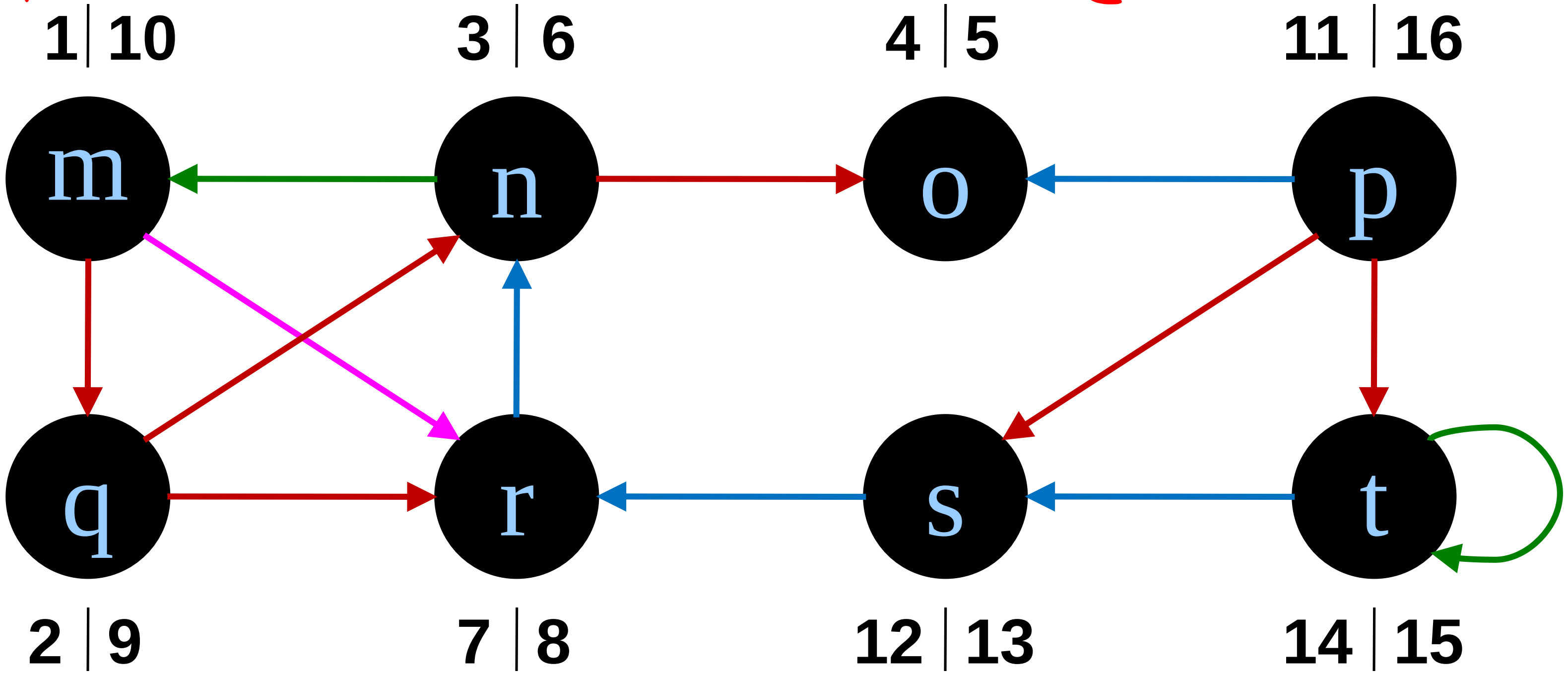


Depth-First Search: Example



- (xxiii)

d[ ] f[ ]

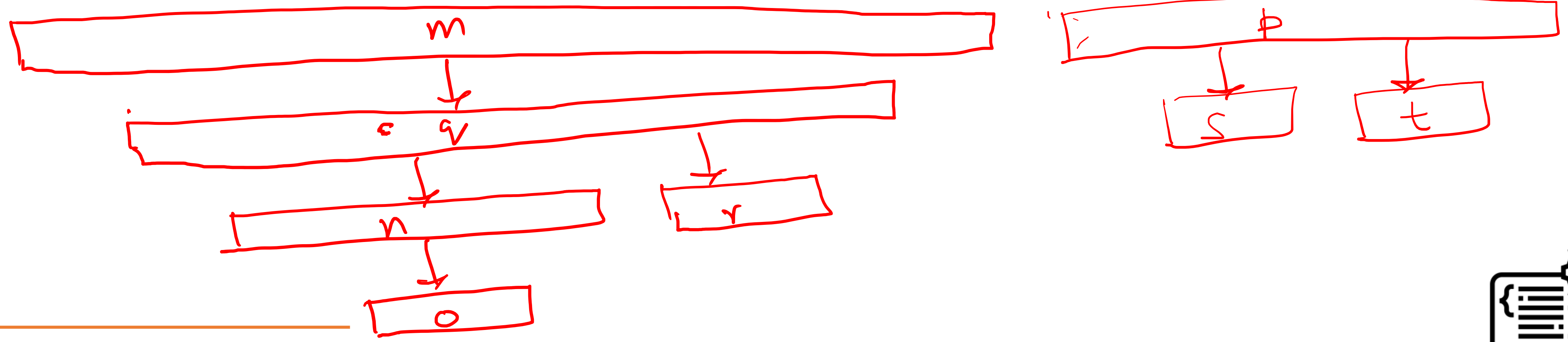


DFS Property: Parenthesis Structure

- Discovery `d[u]` and finishing times `f[u]` have *parenthesis* structure.
- Let's represent
  - discovery of vertex *u* with left parenthesis: `(u`
  - Finishing of vertex *u* with right parenthesis: `u)`
- Parenthesis structure of the example

TIME STAMP →

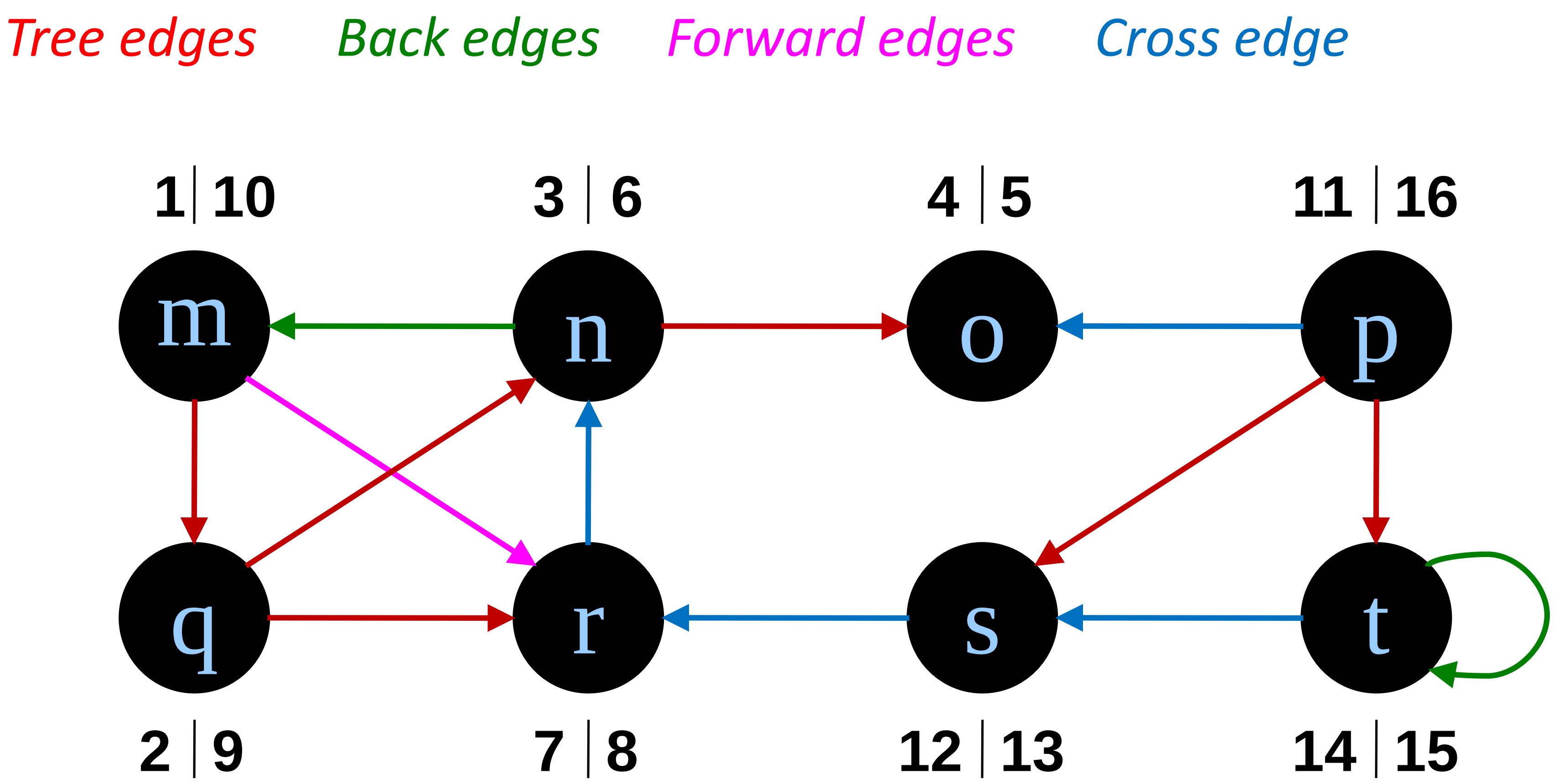
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(m	(q	(n	(o	o)	n)	(r	r)	q)	m)	(p	(s	s)	(t	t)	p)



DFS Property: Edges Classification

- Tree edges are edges encountered when vertex is first discovered (vertex is white).
- Back edges are edges connecting a vertex *u* to ancestor *v* in depth-first tree
  - encountering a gray vertex (from gray to gray)
- Forward edges are non-tree edges connecting a vertex *u* to descendent *v* in depth-first tree.
  - from gray to black
- Cross edge are other edges that go between
  - vertices of the same tree, as long as one vertex is not an ancestor of other, or
  - go between vertices of different trees.

Depth-First Search: Edges



- How do we modify algorithm to identify edges? → HW

Onwards to Python

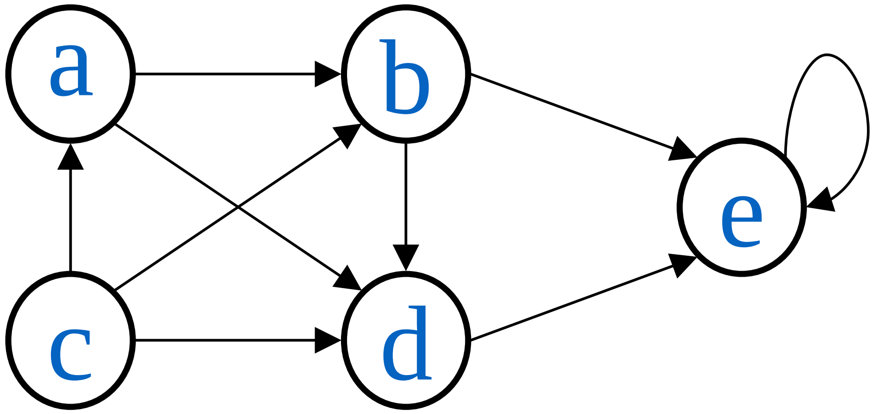
- Graph1.ipynb
  - Representing a graph using dictionary data structure
  - Display the vertices and edges
  - Add a new vertex, new edge

- **TO DO:**
  - Write a program to find an edge. That is, the program should take a graph and an edge as input, and check if it is present or not.

Onwards to Python.. BFS

Compare pseudocode with Python code

- GraphBFS.ipynb: Function to do BFS (as per algo discussed in slides)
- **TO DO:**
  1. Write a code-snippet to display the shortest *path* from source vertex to all the other vertices
  2. Enter the given directed graph, and call BFS with source node “a”. Compute and display the shortest path from “a” to all the other nodes



Onwards to Python.. DFS

- GraphDFS.ipynb: Function to do DFS (as per algo discussed in slides)
- **TO DO:**
  1. Write a code snippet to classify the edges (tree edge/ back edge/ forward edge/ cross edge). You can modify the DFS to do the above.
  2. Run DFS program for the undirected graph example used in the BFS.