NSUPS Bootcamp Week 7

Graph Theories

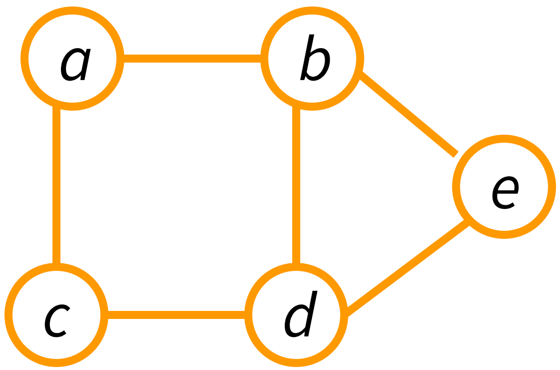
http://bit.ly/bootcamp07

Graphs

A graph G = (V, E)

- V = set of vertices, E = set of edges
- *Dense* graph: $|E| \approx |V|^2$; *Sparse* graph: $|E| \approx |V|$
- *Undirected graph:*
  - edge (u,v) = edge (v,u)
  - No self-loops
- *Directed* graph:
  - Edge (u,v) goes from vertex u to vertex v, notated $u \rightarrow v$
- A *weighted graph* associates weights with either the edges or the vertices

Adjacency Matrix Representation



	a	b	c	d	e
a	0	1	1	0	0
b	1	0	0	1	1
c	1	0	0	1	0
d	0	1	1	0	1
e	0	1	0	1	0

Check for an edge in constant time

Adjacency Matrix Representation

Memory required

$O(V+V^2)=O(V^2)$

Preferred when

The graph is **dense:** $E = O(V^2)$

Advantage

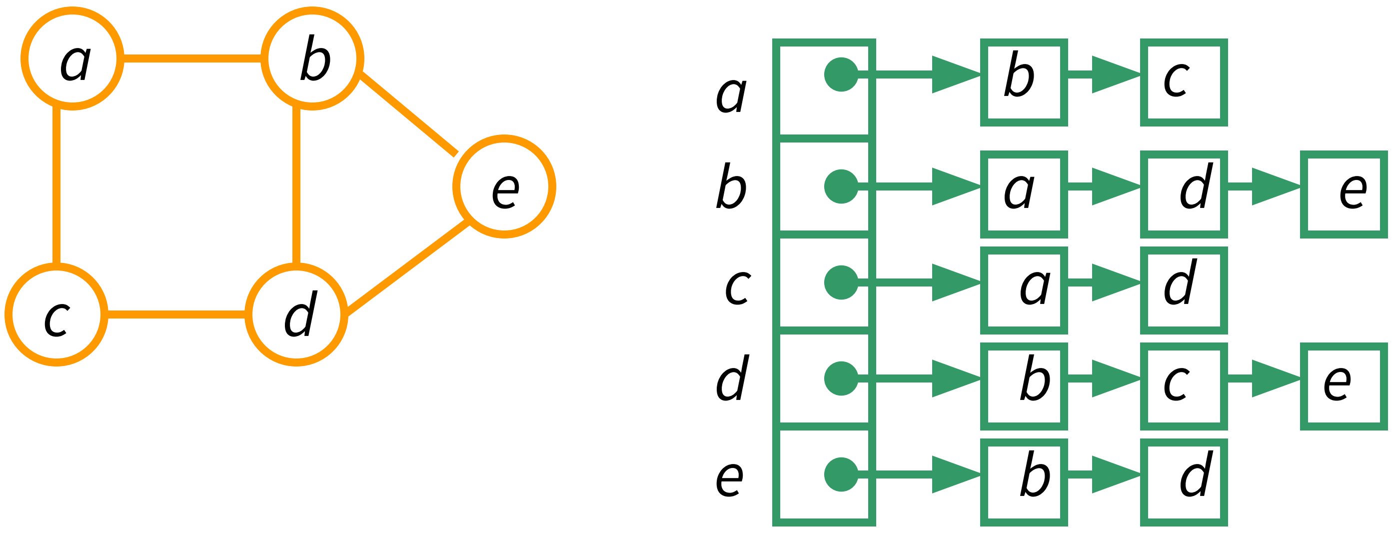
Can quickly determine if there is an edge between two vertices

Disadvantage

No quick way to determine the vertices adjacent **from** another vertex.

Listing all the neighbors of ANY vertex takes $O(V)$ time.

Adjacency List Representation



Space-efficient for sparse graphs

Adjacency List Representation

Memory required

O(V + E)

O(V) for sparse graphs since E=O(V)

$O(V^2)$ for dense graphs since $E=O(V^2)$

Preferred when

for **sparse** graphs: E = O(V)

Disadvantage

No quick way to determine whether there is an edge between vertices u and v

Advantage

Can quickly determine all the vertices adjacent **from** a given vertex, *v:* takes O(N(v)) time, where N(v) is the number of neighbors of v.

Graph Searching

Given: a graph G = (V, E), directed or undirected

Goal: methodically explore every vertex and every edge

Ultimately: build a tree on the graph

- Pick a vertex as the root
- Choose certain edges to produce a tree
- Note: might also build a *forest* if graph is not connected

Graph Traversals

Ways to traverse/search a graph

- Visit every vertex exactly once

Breadth-First Search

Depth-First Search

Breadth-First Search

"Explore" a graph, turning it into a tree

- One vertex at a time
- Expand frontier of explored vertices across the *breadth* of the frontier

Builds a tree over the graph

- Pick a *source vertex* to be the root
- Find ("discover") all of its children, then their children, etc.

Breadth-First Search

- Associate vertex “colors” to guide the algorithm
  - White vertices have not been discovered
    - All vertices start out white
  - Grey vertices are discovered but not fully explored
    - They may be adjacent to white vertices
  - Black vertices are discovered and fully explored
    - They are adjacent only to black and gray vertices
- Explore vertices by scanning adjacency list of grey vertices

BFS Trees

BFS tree is not necessarily unique for a given graph

Depends on the order in which neighboring vertices are processed

During the breadth-first search, assign an integer to each vertex

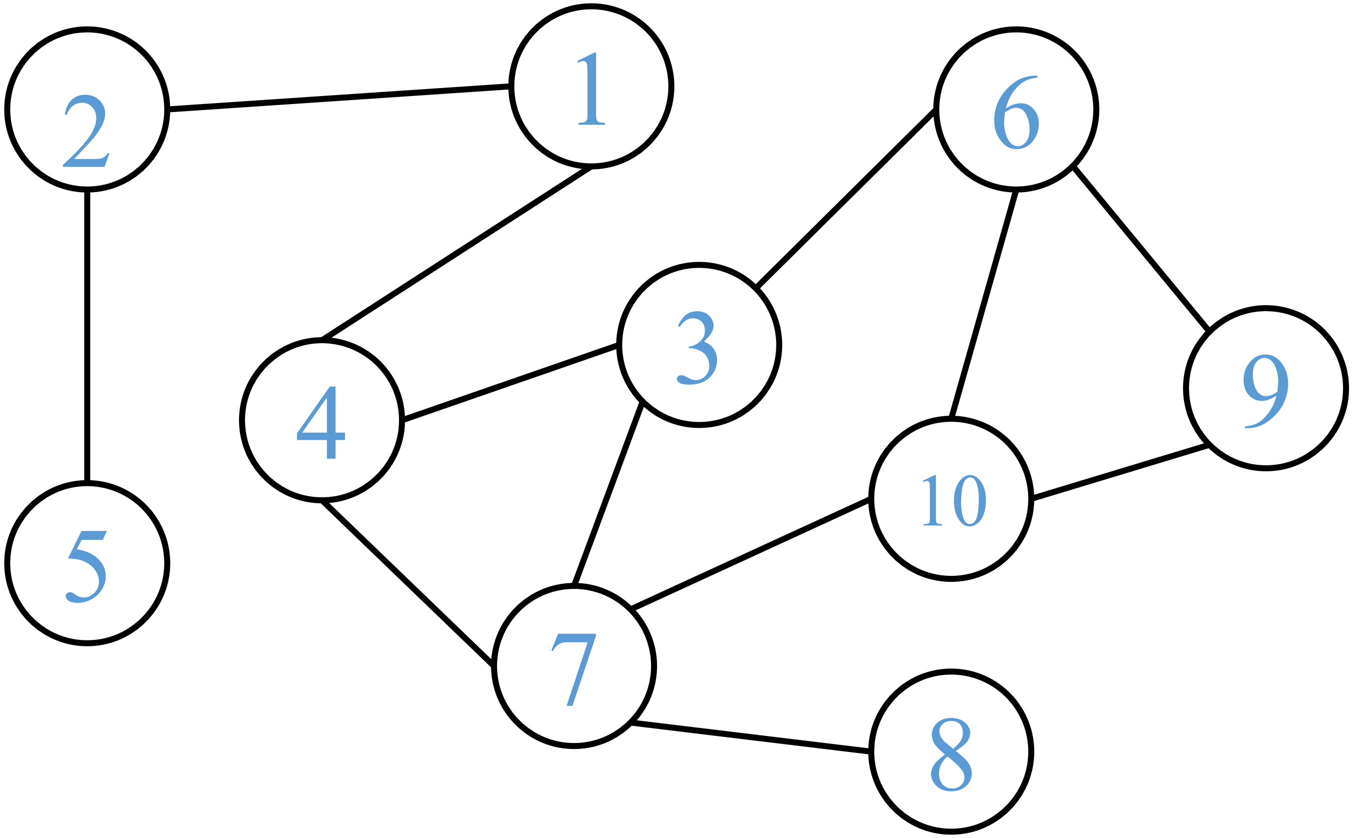
Indicate the distance of each vertex from the source s

Breadth-First Search

BFS(*G*, *s*)

```
for each vertex u ϵ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = Ø
ENQUEUE(Q, s)
while Q ≠ Ø
    u = DEQUEUE(Q)
    for each v ϵ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

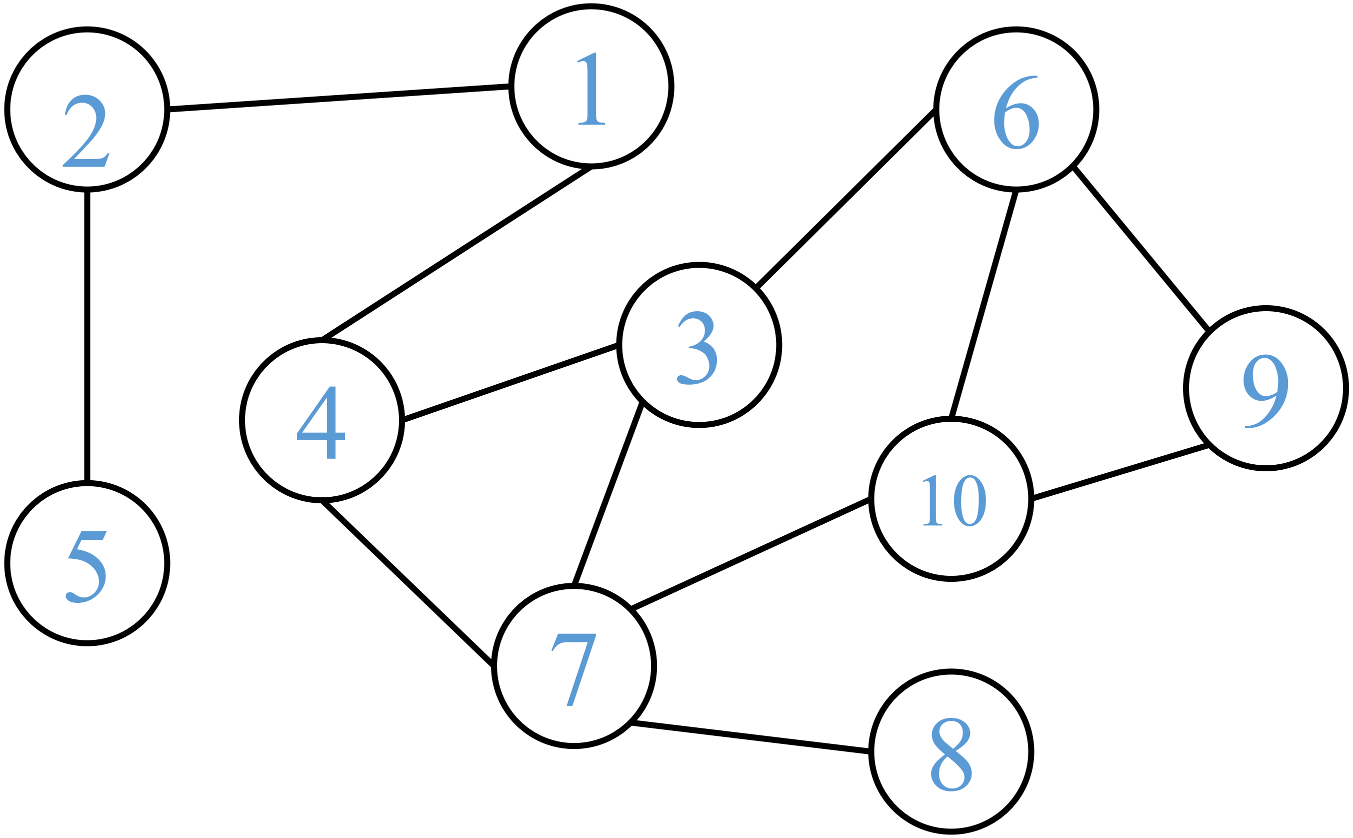
Breadth-First Search: Example



visited[]
[1] = 0
[2] = 0
[3] = 0
[4] = 0
[5] = 0
[6] = 0
[7] = 0
[8] = 0
[9] = 0
[10] = 0

Q:

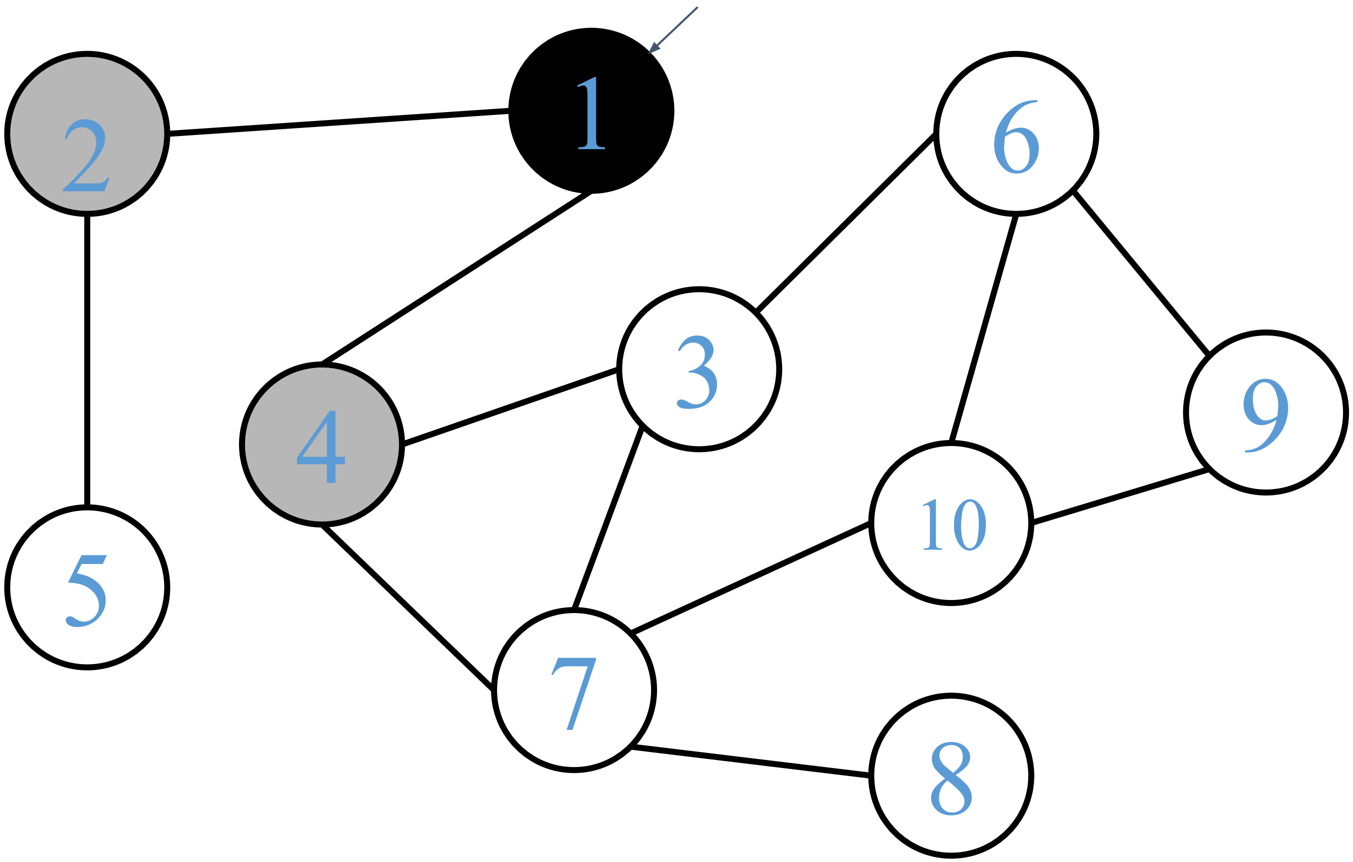
Breadth-First Search: Example



visited[]
[1] = 1
[2] = 0
[3] = 0
[4] = 0
[5] = 0
[6] = 0
[7] = 0
[8] = 0
[9] = 0
[10] = 0

Q: $\boxed{1}$

Breadth-First Search: Example

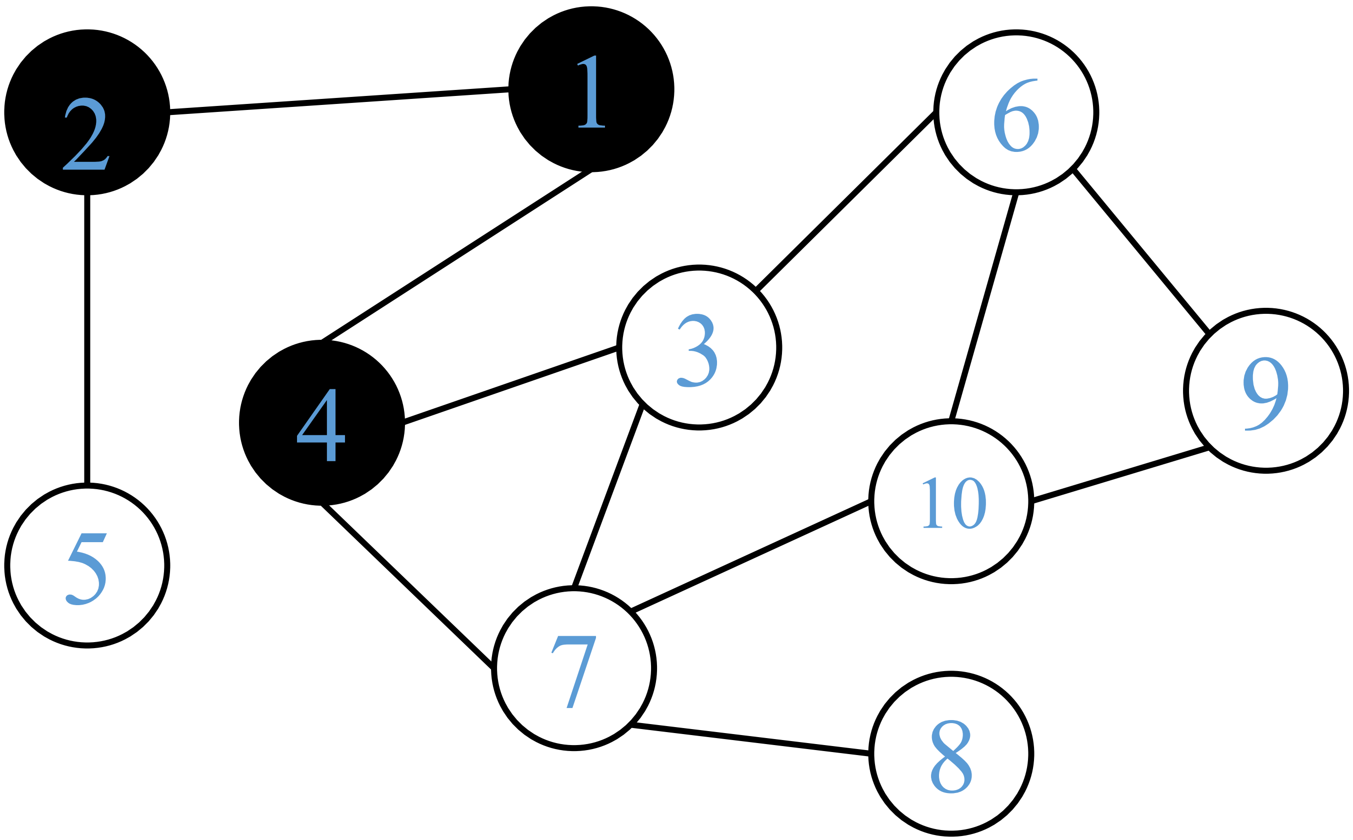


visited[]
[1] = 1
[2] = 0
[3] = 0
[4] = 0
[5] = 0
[6] = 0
[7] = 0
[8] = 0
[9] = 0
[10] = 0

Q: $\boxed{1}$

Top of *Q*: 1

Breadth-First Search: Example

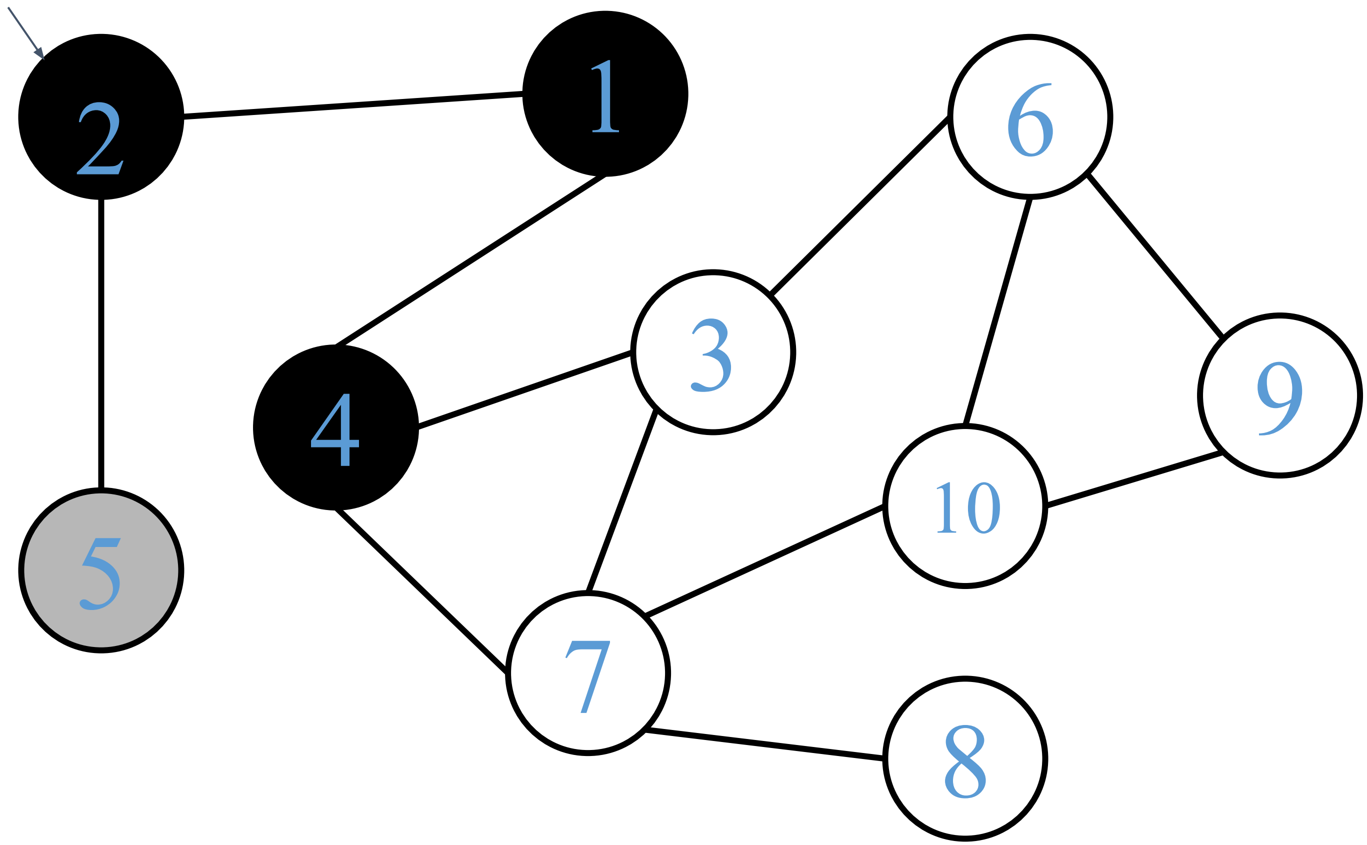


visited[]

[1] = 1
[2] = 1
[3] = 0
[4] = 1
[5] = 0
[6] = 0
[7] = 0
[8] = 0
[9] = 0
[10] = 0

Top of Q: 2 4

Breadth-First Search: Example

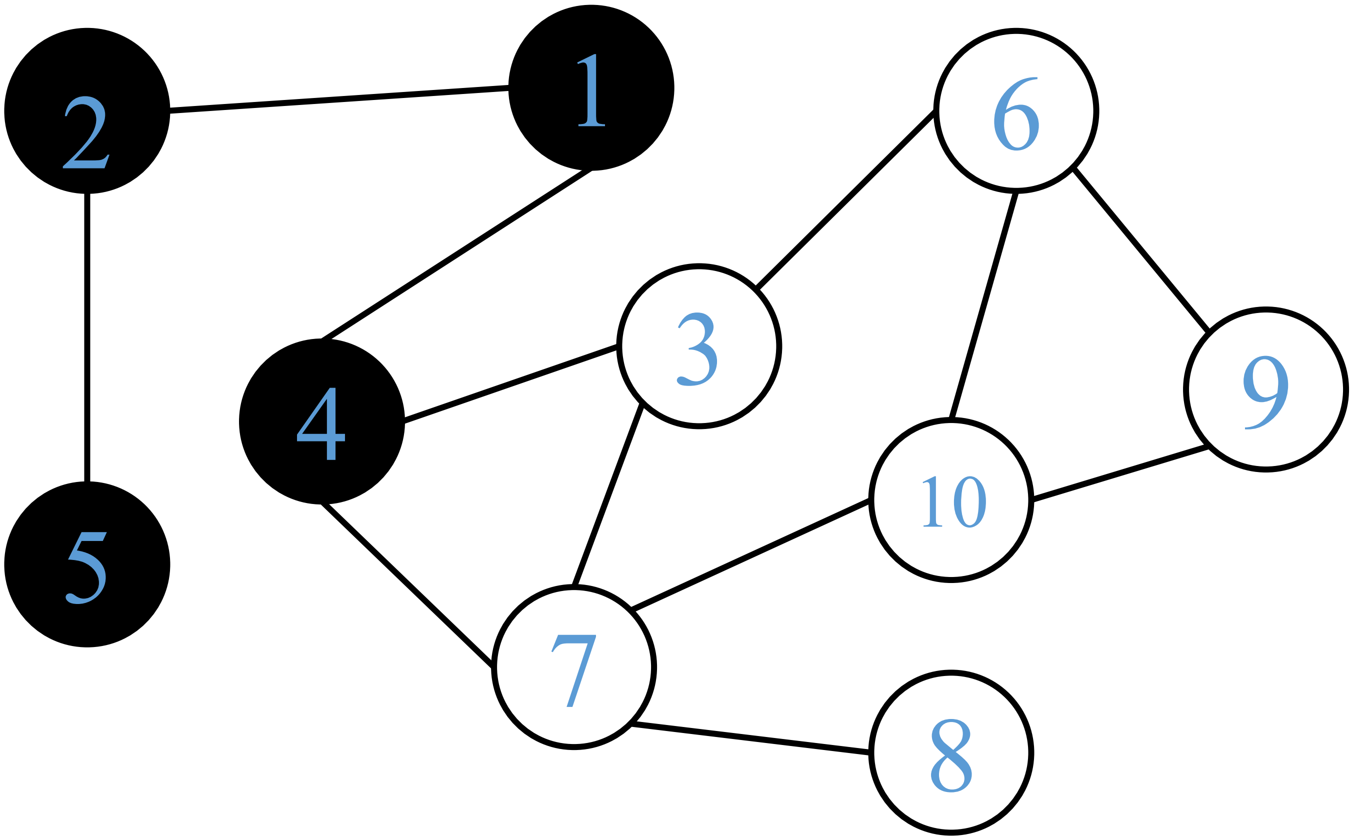


visited[]
[1] = 1
[2] = 1
[3] = 0
[4] = 1
[5] = 0
[6] = 0
[7] = 0
[8] = 0
[9] = 0
[10] = 0

Top of Q: 2

Q: | *2* | *4* |

Breadth-First Search: Example

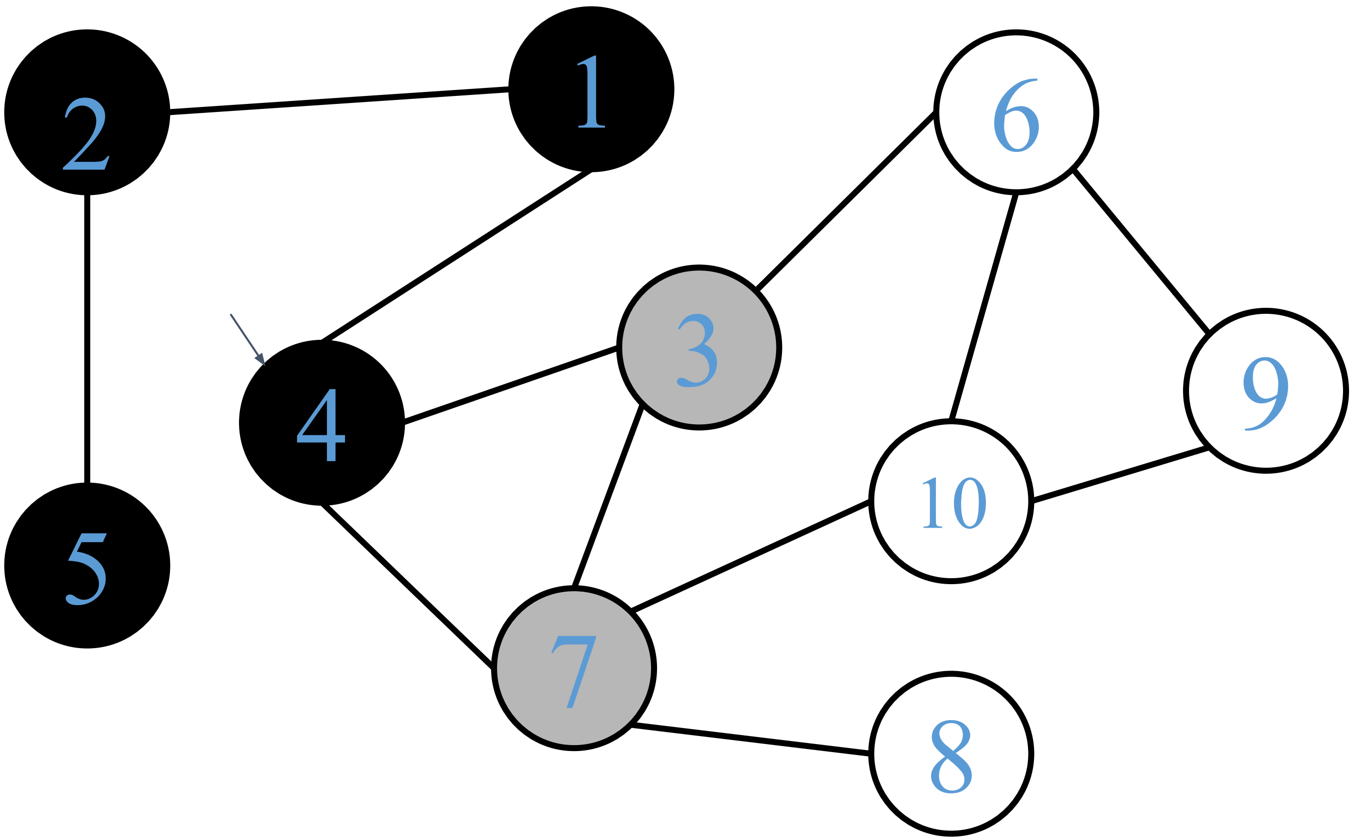


visited[]
[1] = 1
[2] = 1
[3] = 0
[4] = 1
[5] = 1
[6] = 0
[7] = 0
[8] = 0
[9] = 0
[10] = 0

Q: | *4* | *5* |

Top of Q | 4 |

Breadth-First Search: Example

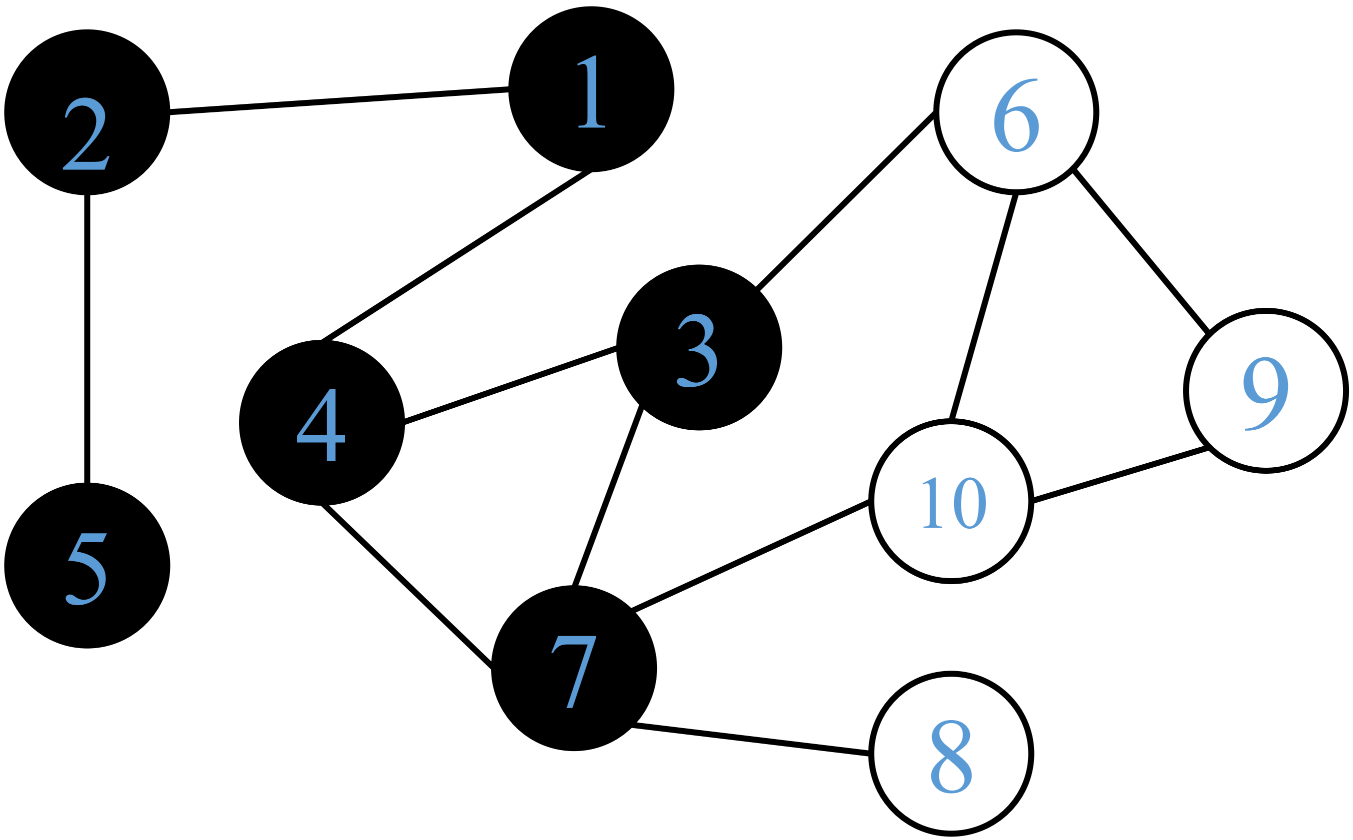


visited[]
[1] = 1
[2] = 1
[3] = 0
[4] = 1
[5] = 1
[6] = 0
[7] = 0
[8] = 0
[9] = 0
[10] = 0

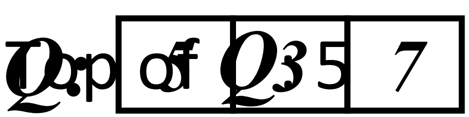
Q: | 4 | 5 |

Top of Q 4

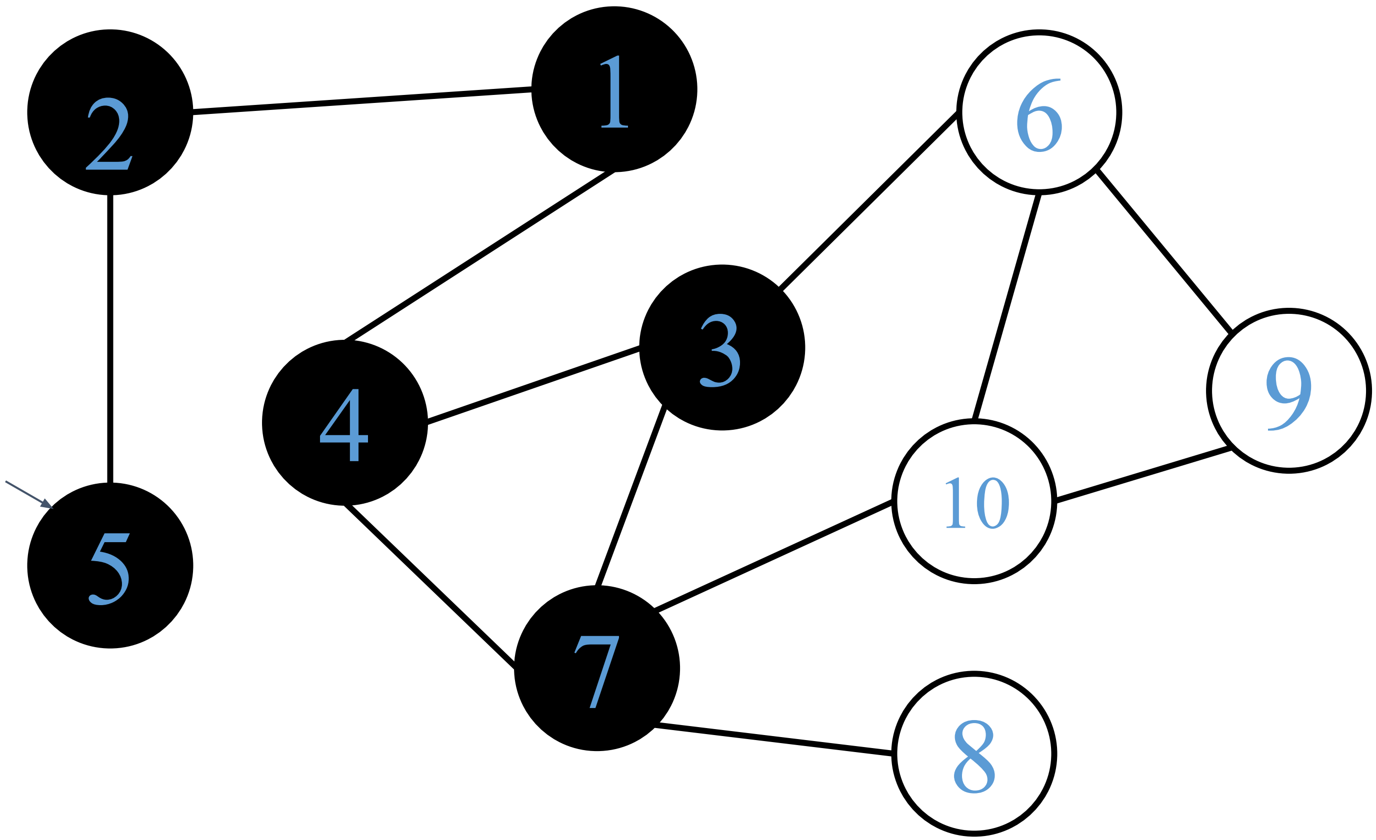
Breadth-First Search: Example



visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 0
[7] = 1
[8] = 0
[9] = 0
[10] = 0



Breadth-First Search: Example

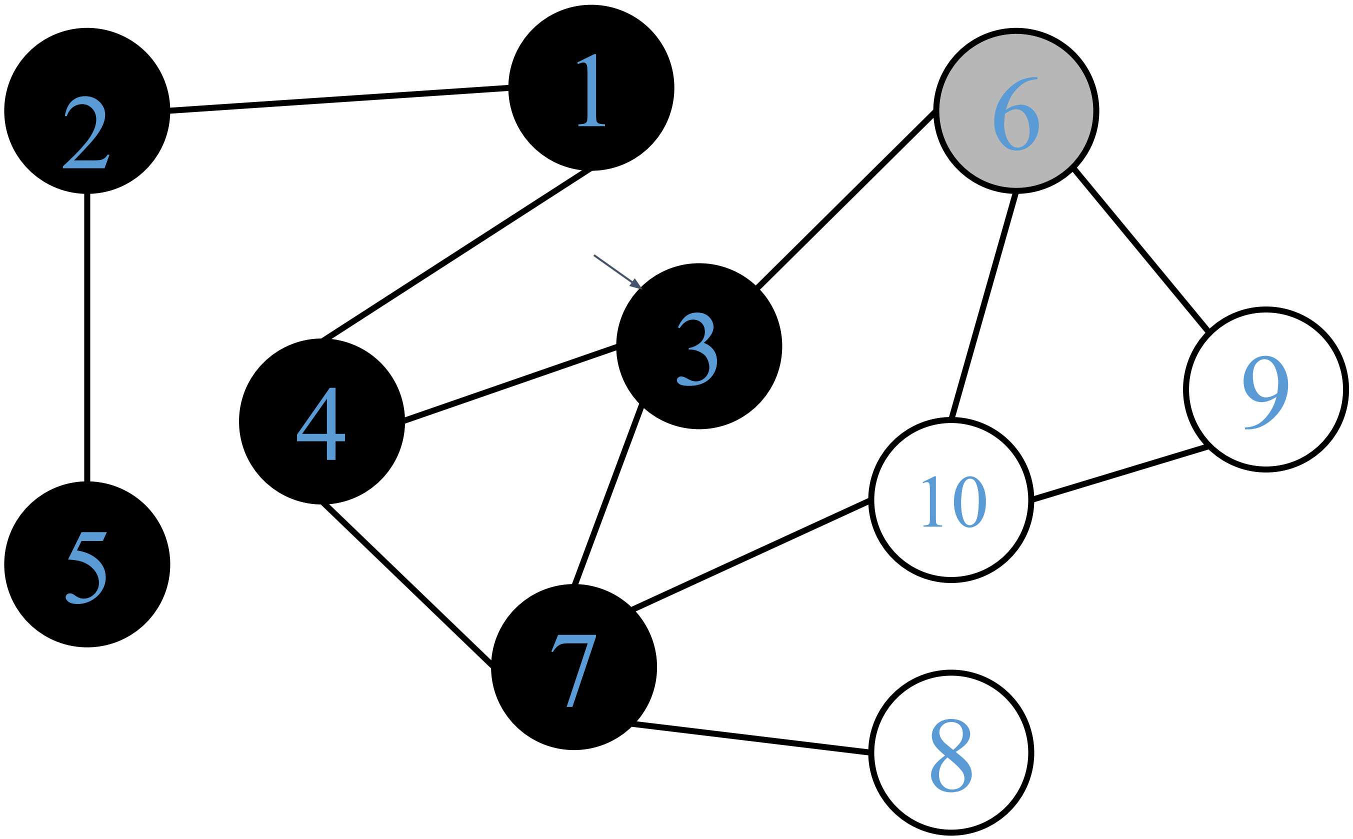


visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 0
[7] = 1
[8] = 0
[9] = 0
[10] = 0

Top of Q 5

Q: | 5 | 3 | 7 |

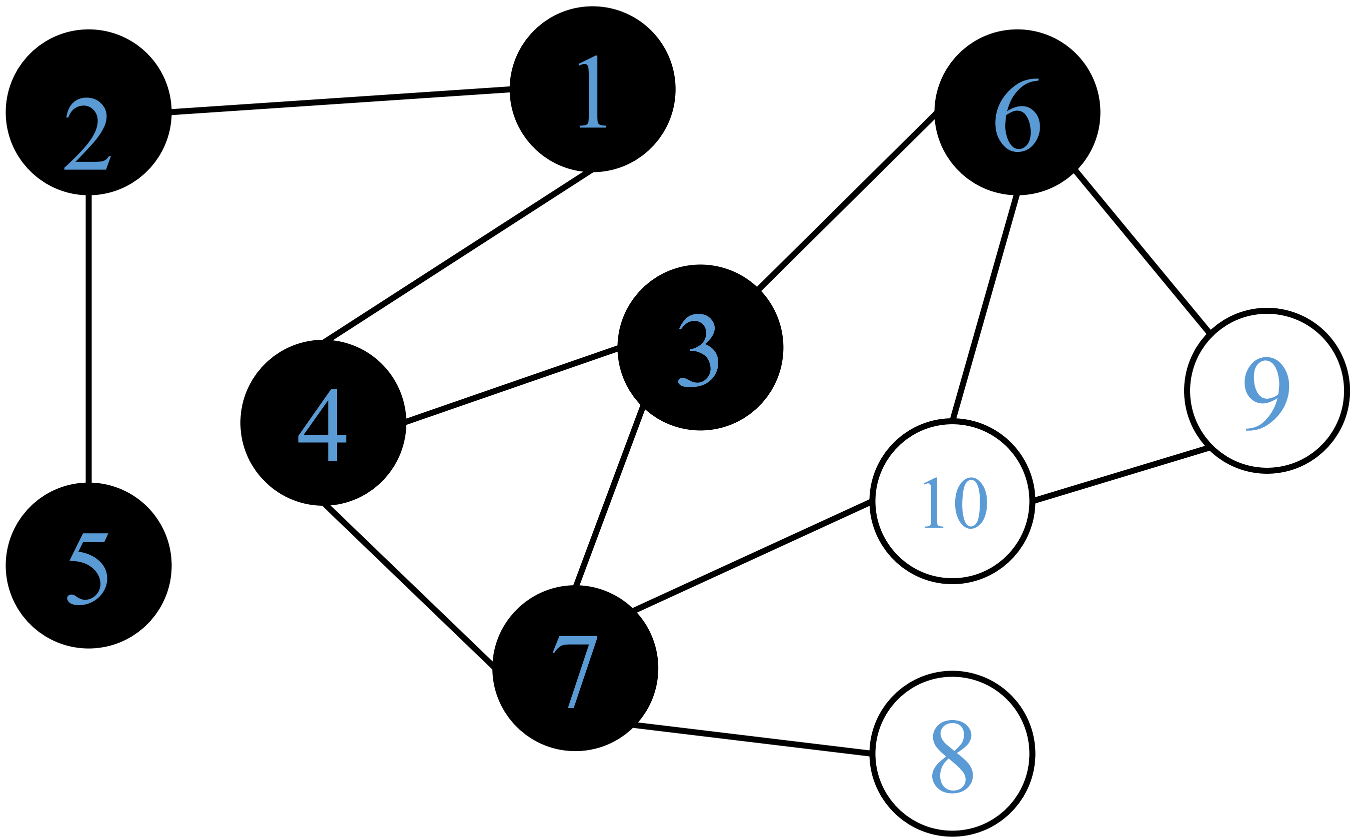
Breadth-First Search: Example



visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 0
[7] = 1
[8] = 0
[9] = 0
[10] = 0

Top of Q: 3 7

Breadth-First Search: Example

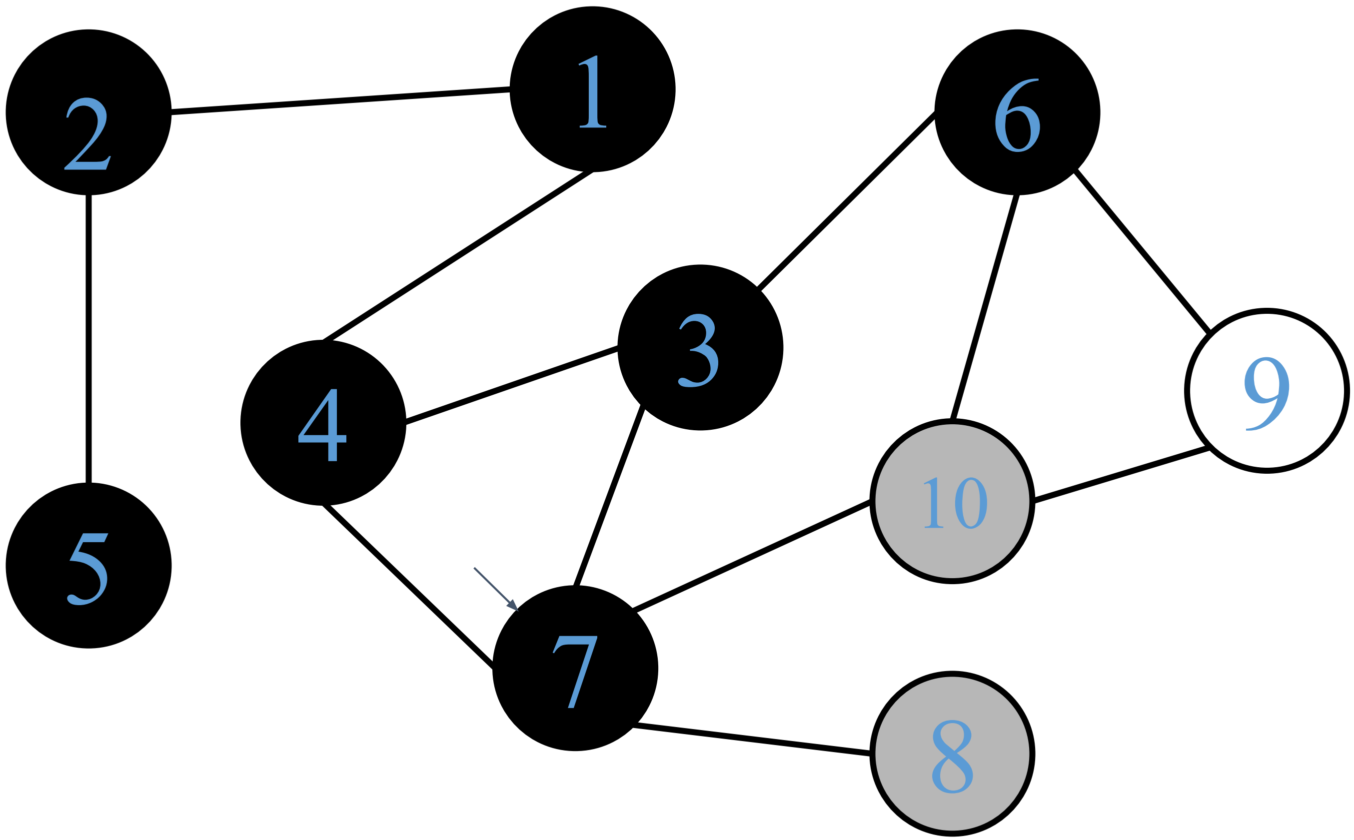


visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 0
[7] = 1
[8] = 0
[9] = 0
[10] = 0

Q: 7 6

Top of Q 7

Breadth-First Search: Example

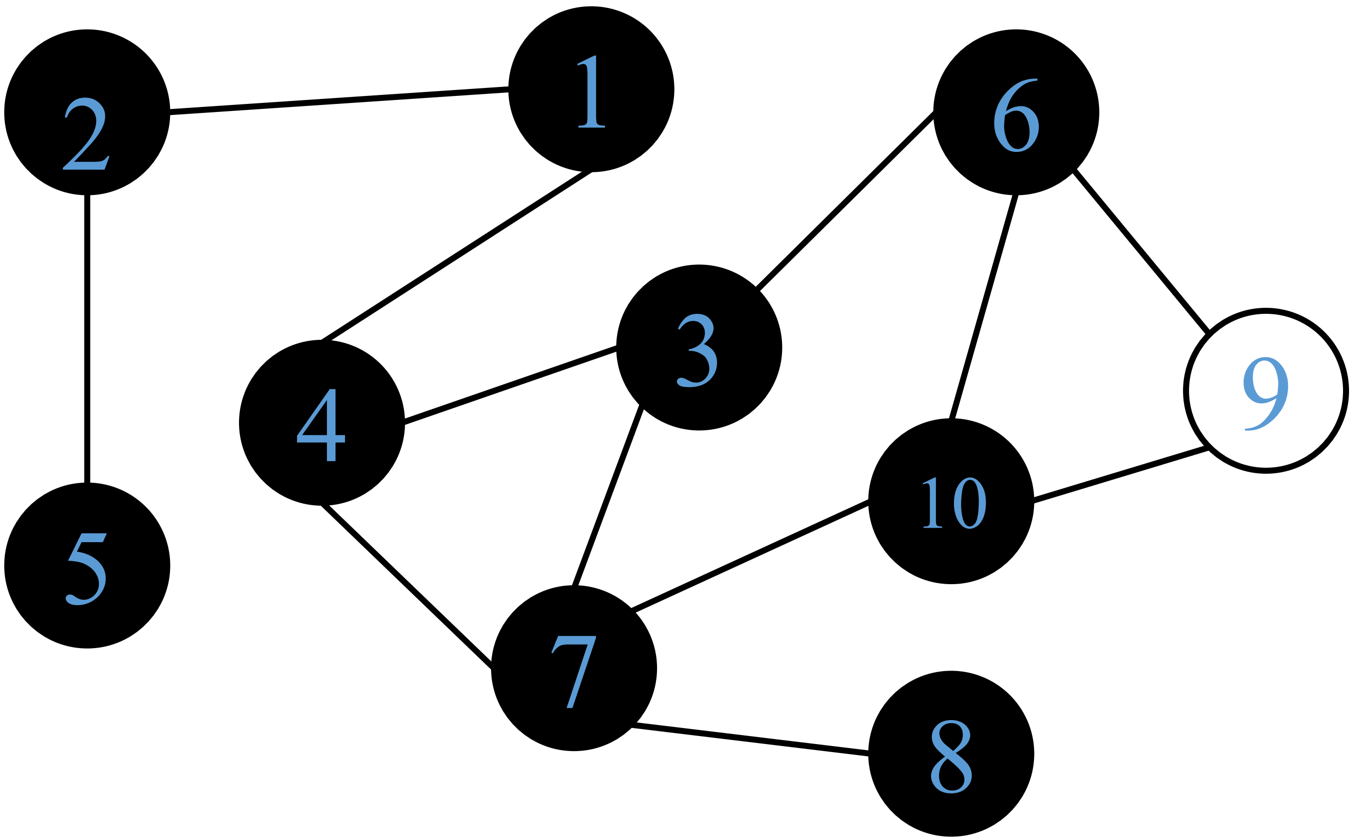


visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 1
[7] = 1
[8] = 0
[9] = 0
[10] = 0

Top of Q

Q: 7 | 6 7

Breadth-First Search: Example



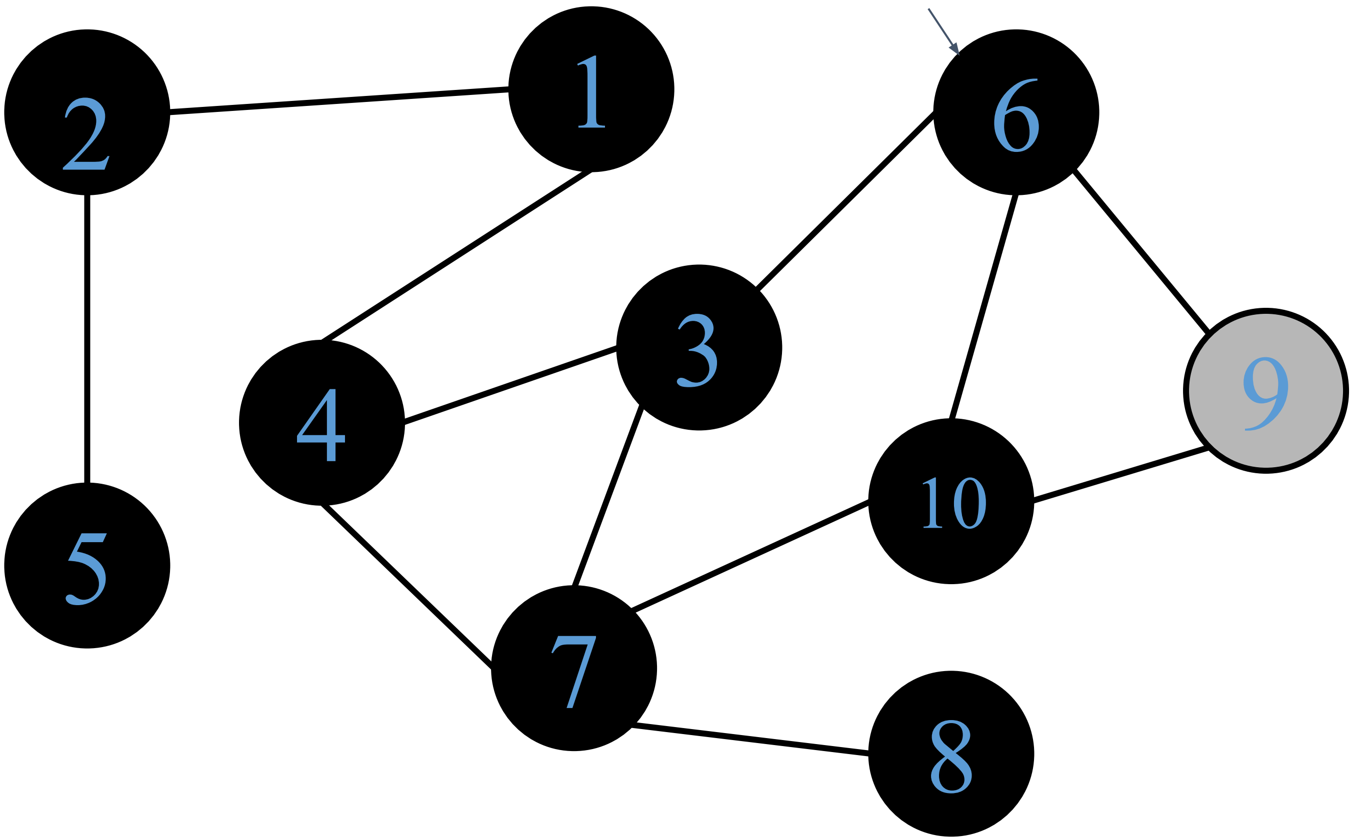
visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 1
[7] = 1
[8] = 1
[9] = 0
[10] = 1

Top of Q: 6

Q:

6	10	8

Breadth-First Search: Example



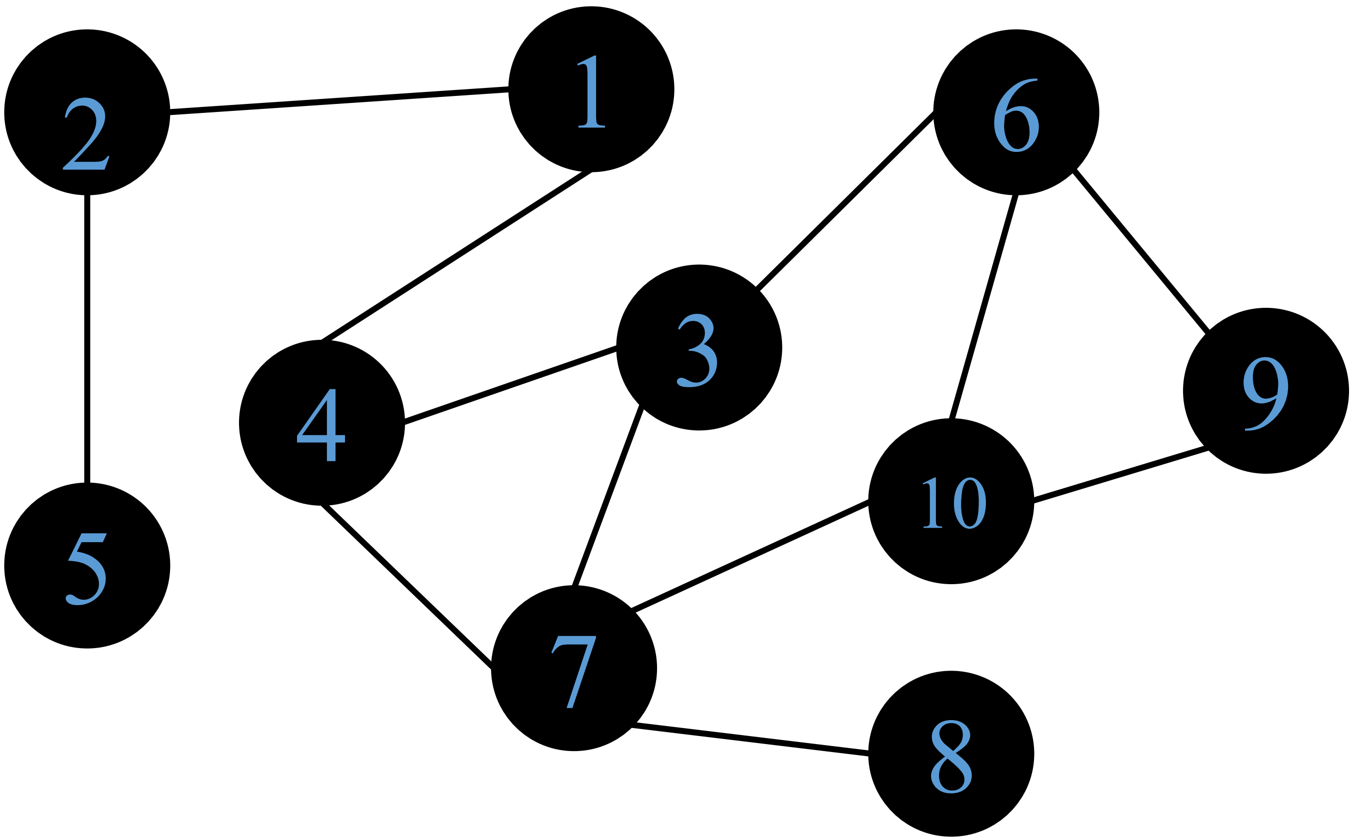
visited[]

[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 1
[7] = 1
[8] = 1
[9] = 0
[10] = 1

Top of Q: 6 | 10 | 8

Q: 6 | 10 | 8

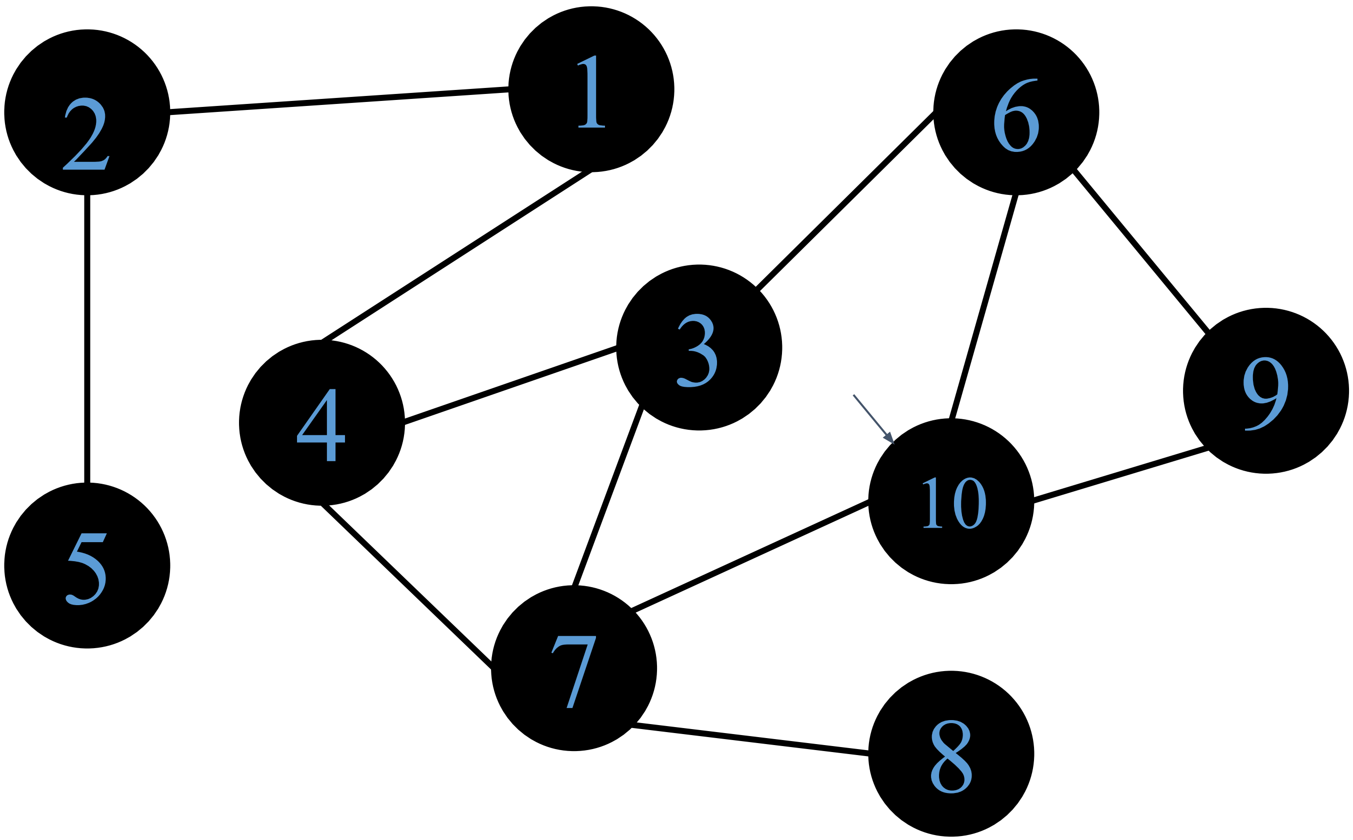
Breadth-First Search: Example



visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 1
[7] = 1
[8] = 1
[9] = 1
[10] = 1

Q: Top | 10 | 8 | 10 9

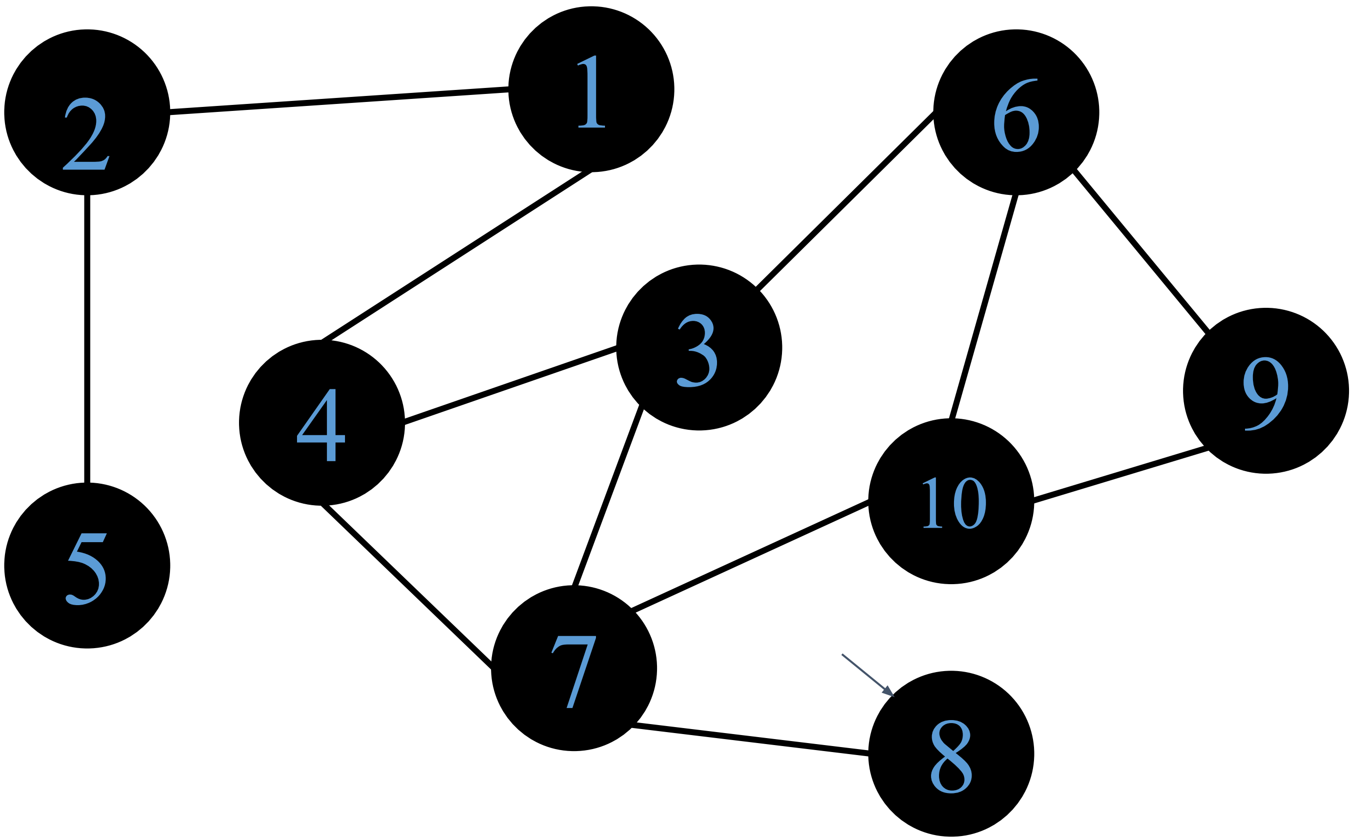
Breadth-First Search: Example



visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 1
[7] = 1
[8] = 1
[9] = 1
[10] = 1

Q: 10 8 9

Breadth-First Search: Example

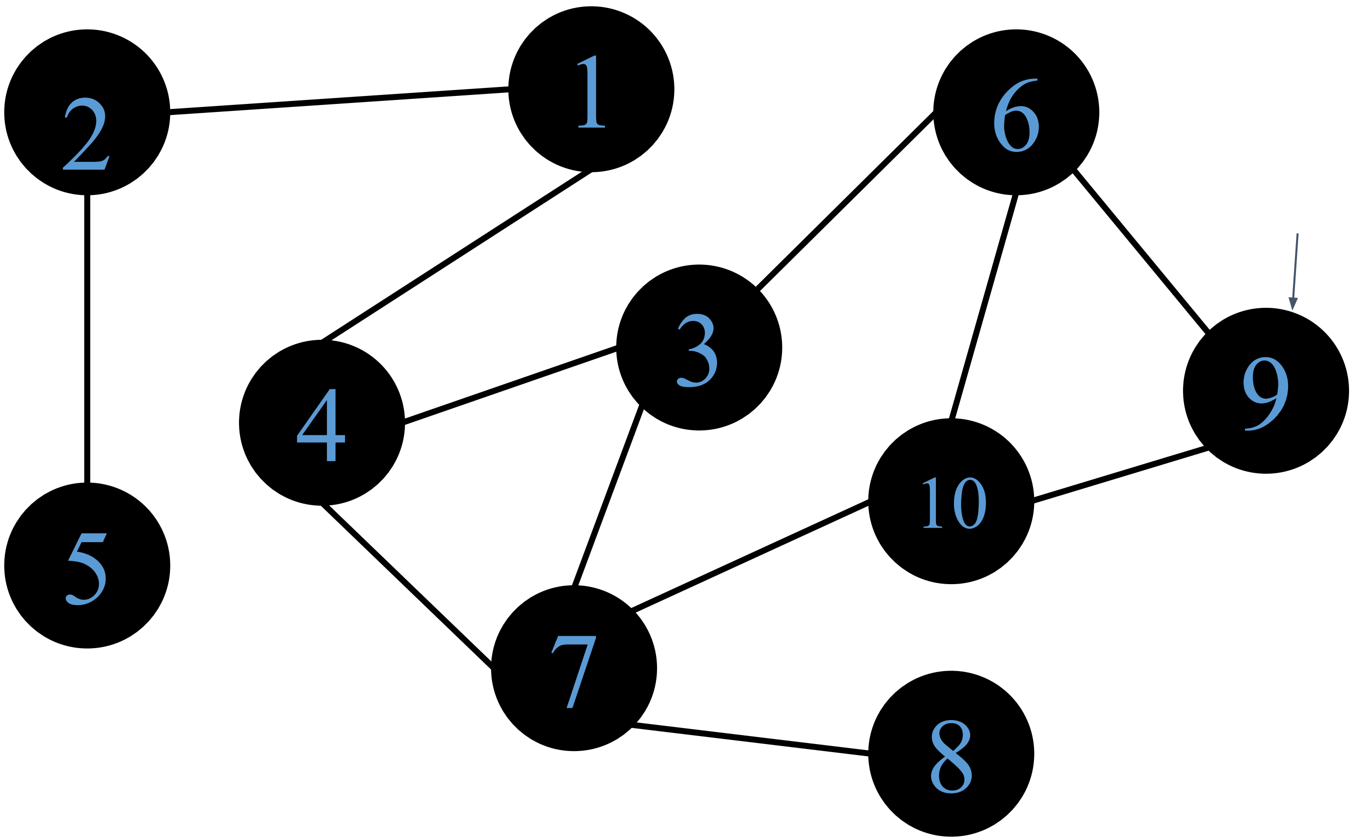


visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 1
[7] = 1
[8] = 1
[9] = 1
[10] = 1

Top of Q: 8

Q: 9 8

Breadth-First Search: Example

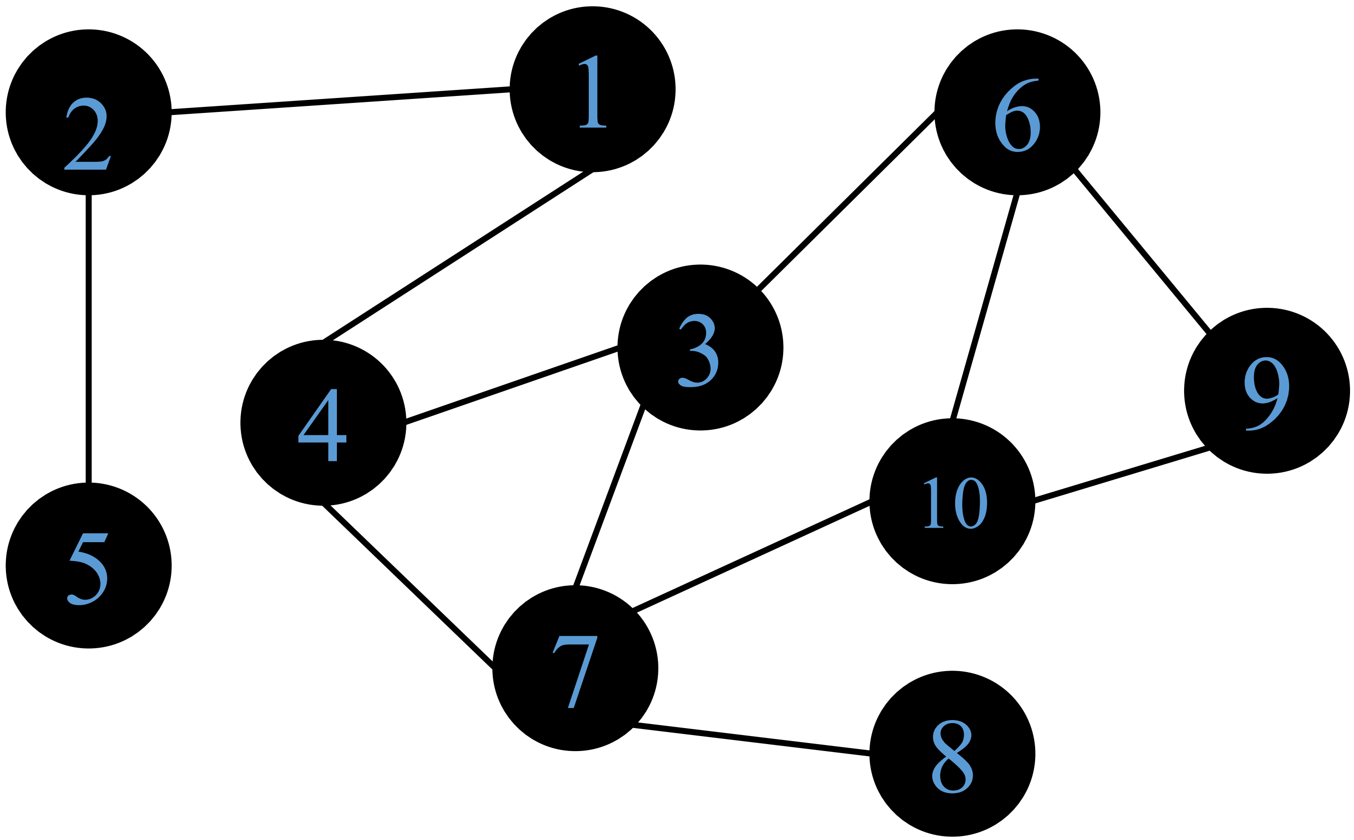


visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 1
[7] = 1
[8] = 1
[9] = 1
[10] = 1

Q:

Top of Q: 9

Breadth-First Search: Example



visited[]
[1] = 1
[2] = 1
[3] = 1
[4] = 1
[5] = 1
[6] = 1
[7] = 1
[8] = 1
[9] = 1
[10] = 1

Q:

BFS Running Time

Initialization of each vertex takes $\boldsymbol{O(V)}$ time

Every vertex is enqueued once and dequeued once, taking $\boldsymbol{O(V)}$ time

When a vertex is dequeued, all its neighbors are checked to see if they are unvisited, taking time proportional to number of neighbors of the vertex, and summing to $\boldsymbol{O(E)}$ over all iterations

Total time is $\boldsymbol{O(V+E)}$

Breadth-First Search: Properties

BFS calculates the *shortest-path distance* to the source node

- Shortest-path distance $\delta(s,v)$ = minimum number of edges from s to v, or ∞ if v not reachable from s

BFS builds *breadth-first tree*, in which paths to root represent shortest paths in G

- Thus we can use BFS to calculate shortest path from one vertex to another in $O(V+E)$ time

Depth-First Search

Depth-first search is another strategy for exploring a graph

- Explore "deeper" in the 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

Depth-First Search

Vertices initially colored white

Then colored gray when discovered

Then black when finished

DFS Tree

Actually might be a DFS forest (collection of trees)

Keep track of parents

Depth-First Search

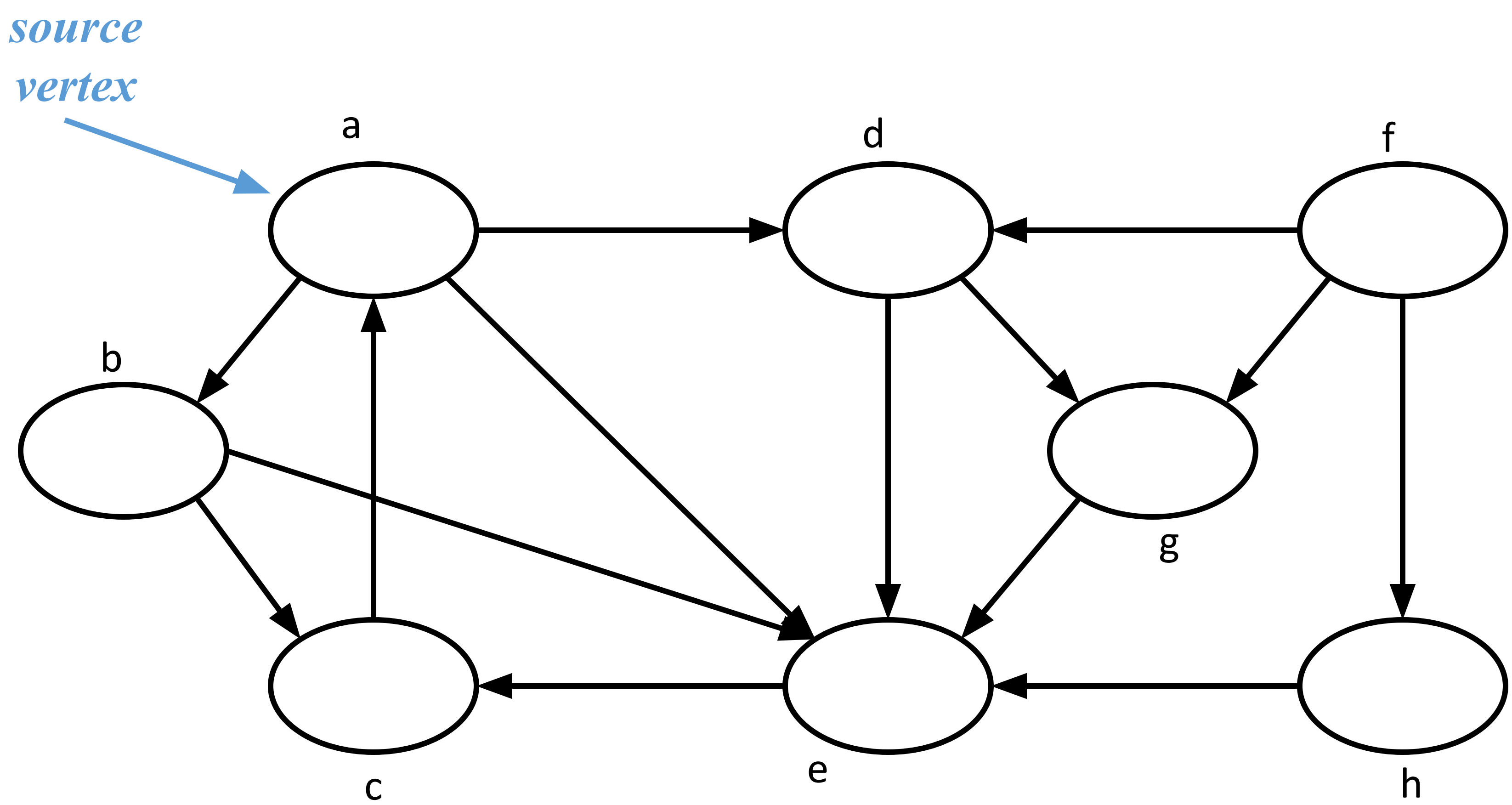
DFS (*G*)

```
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

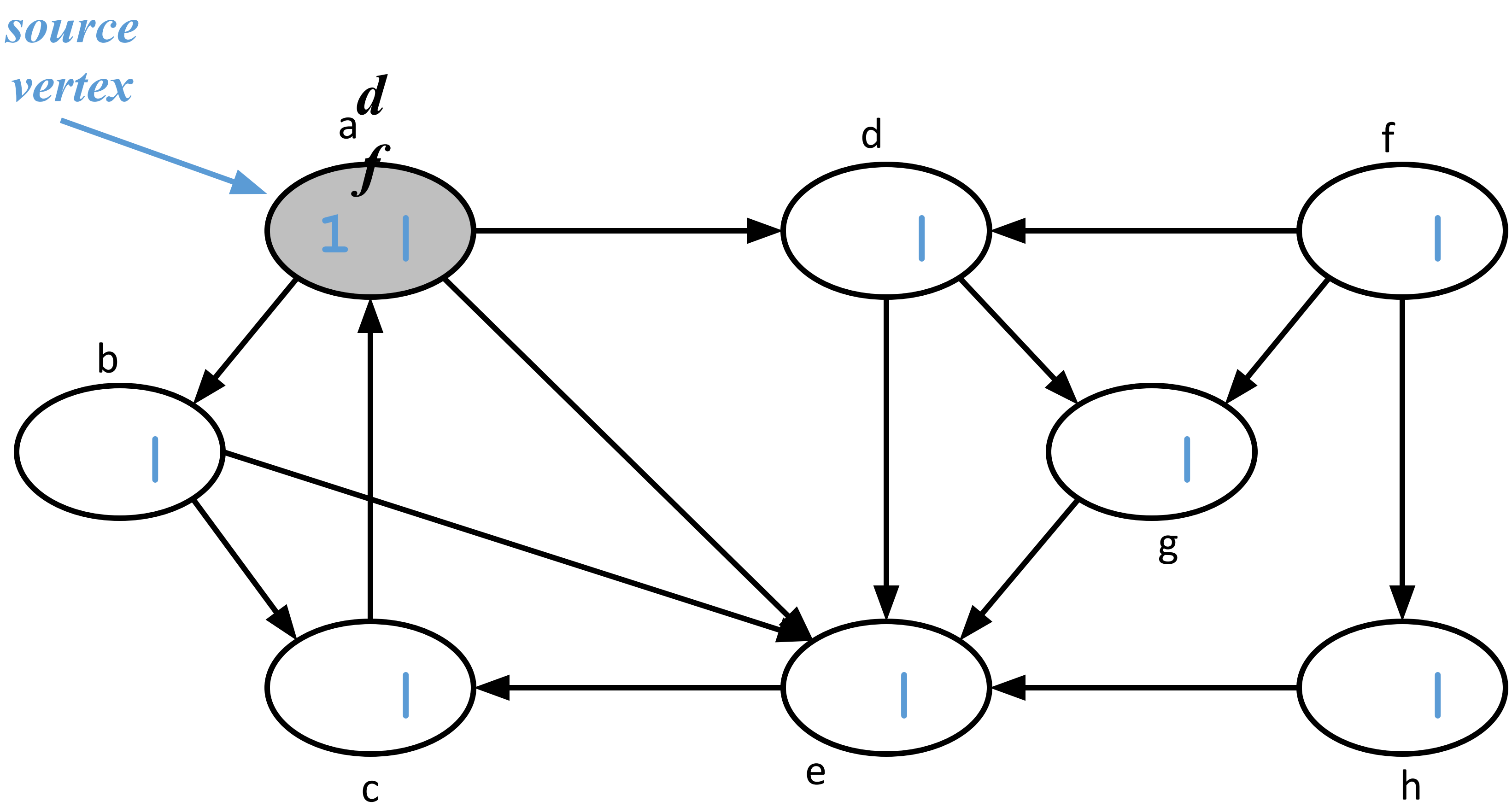
DFS-VISIT(*G*, *u*)

```
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
```

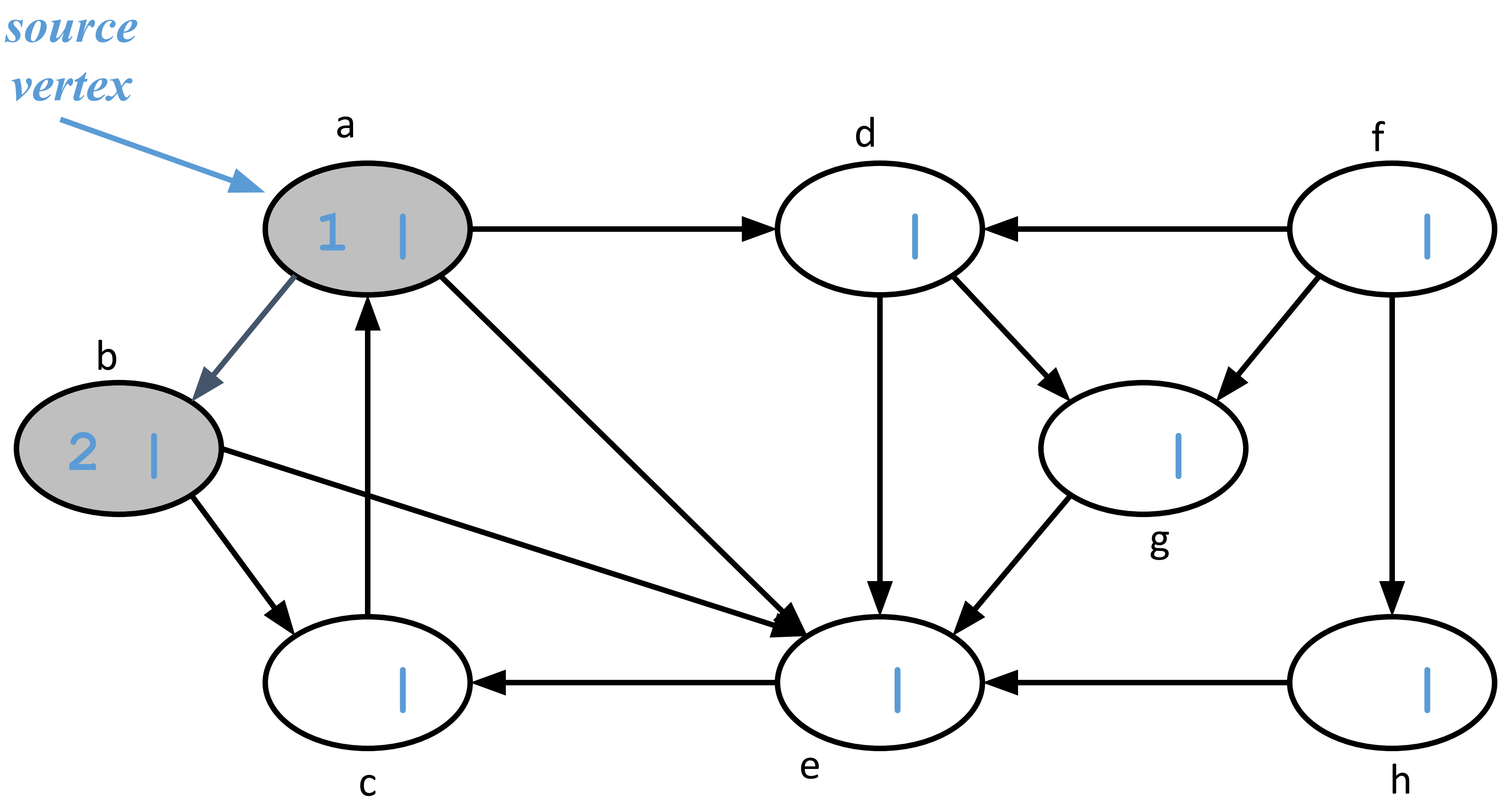
DFS Example



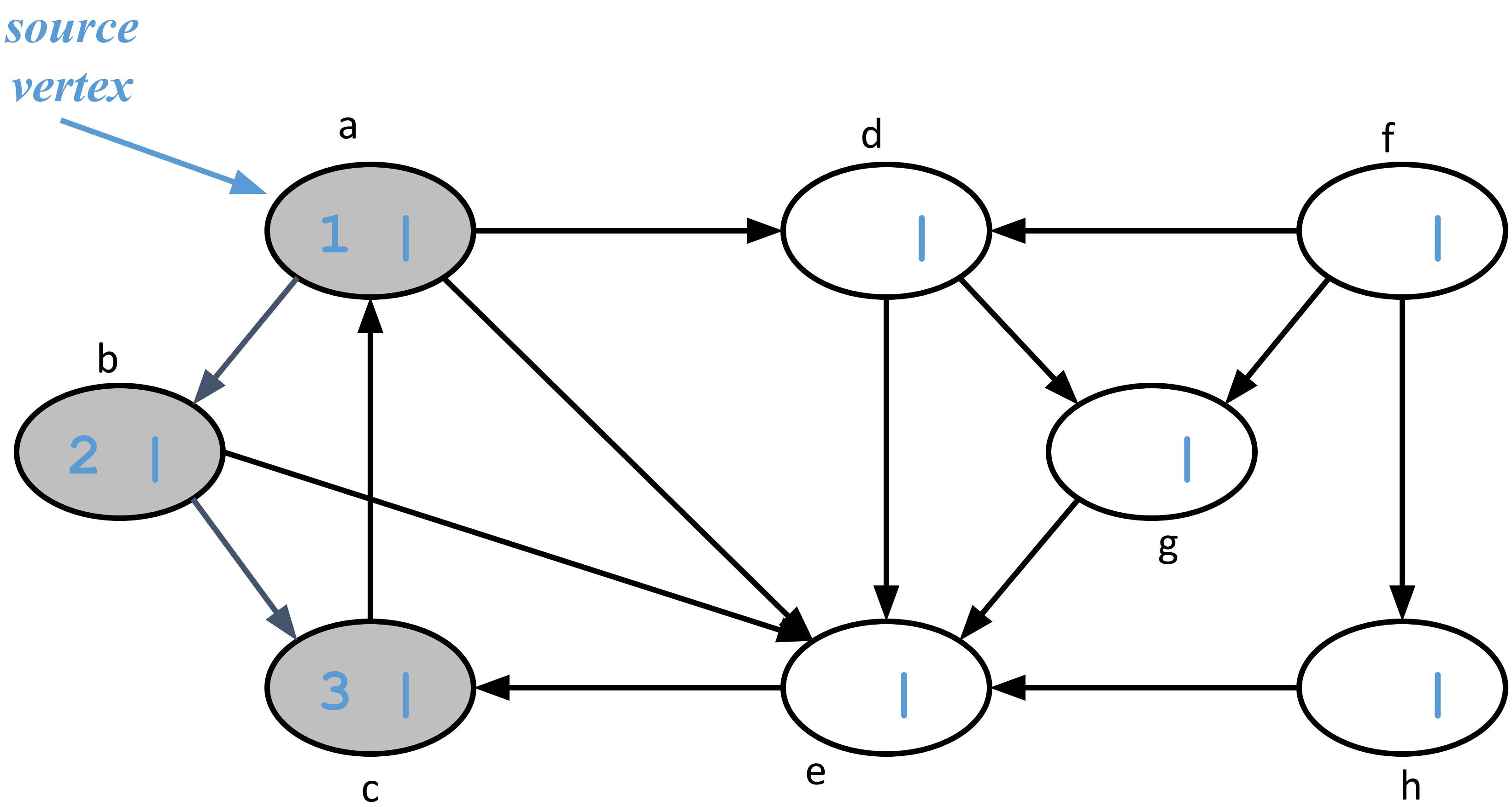
DFS Example



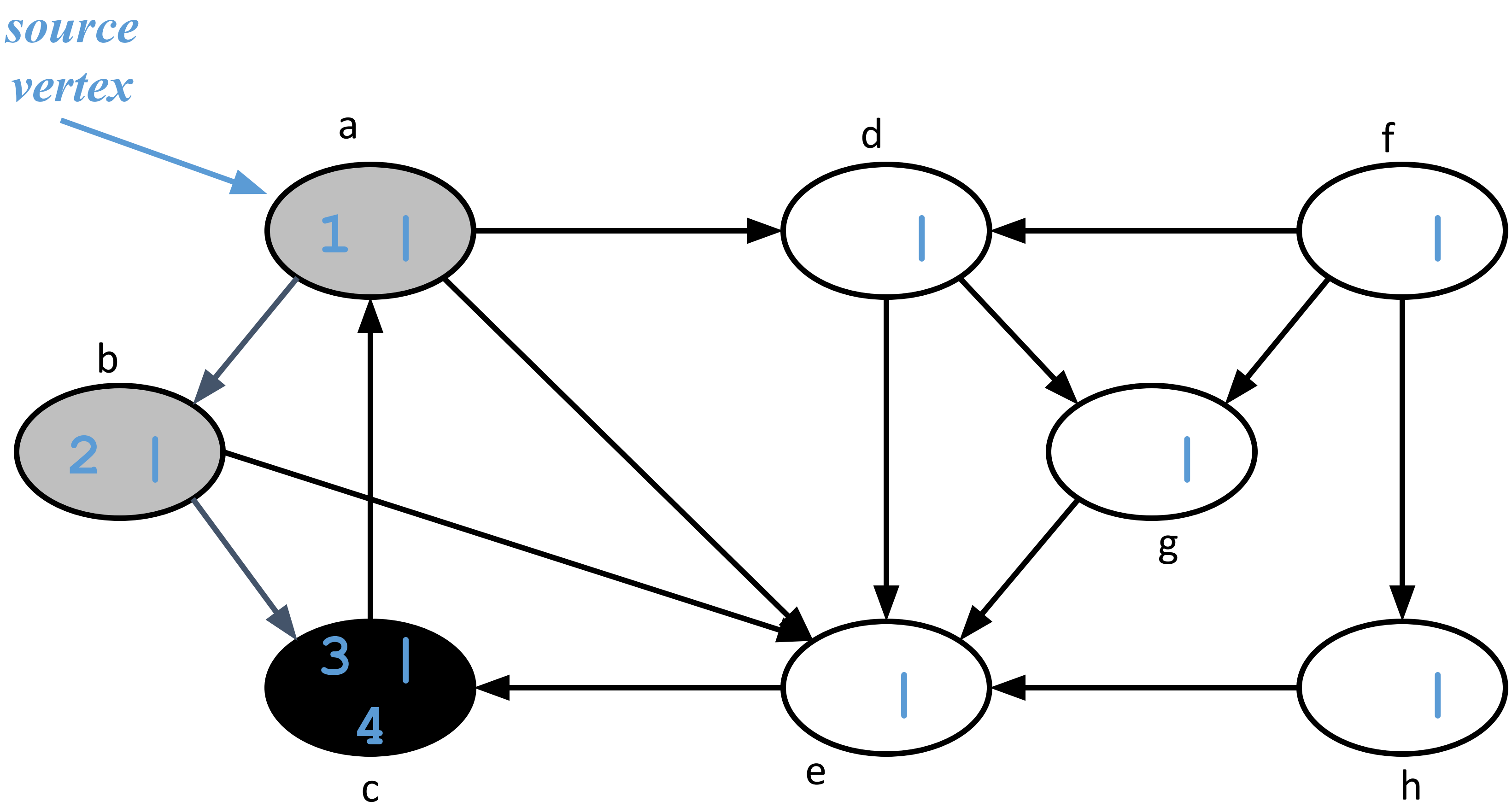
DFS Example



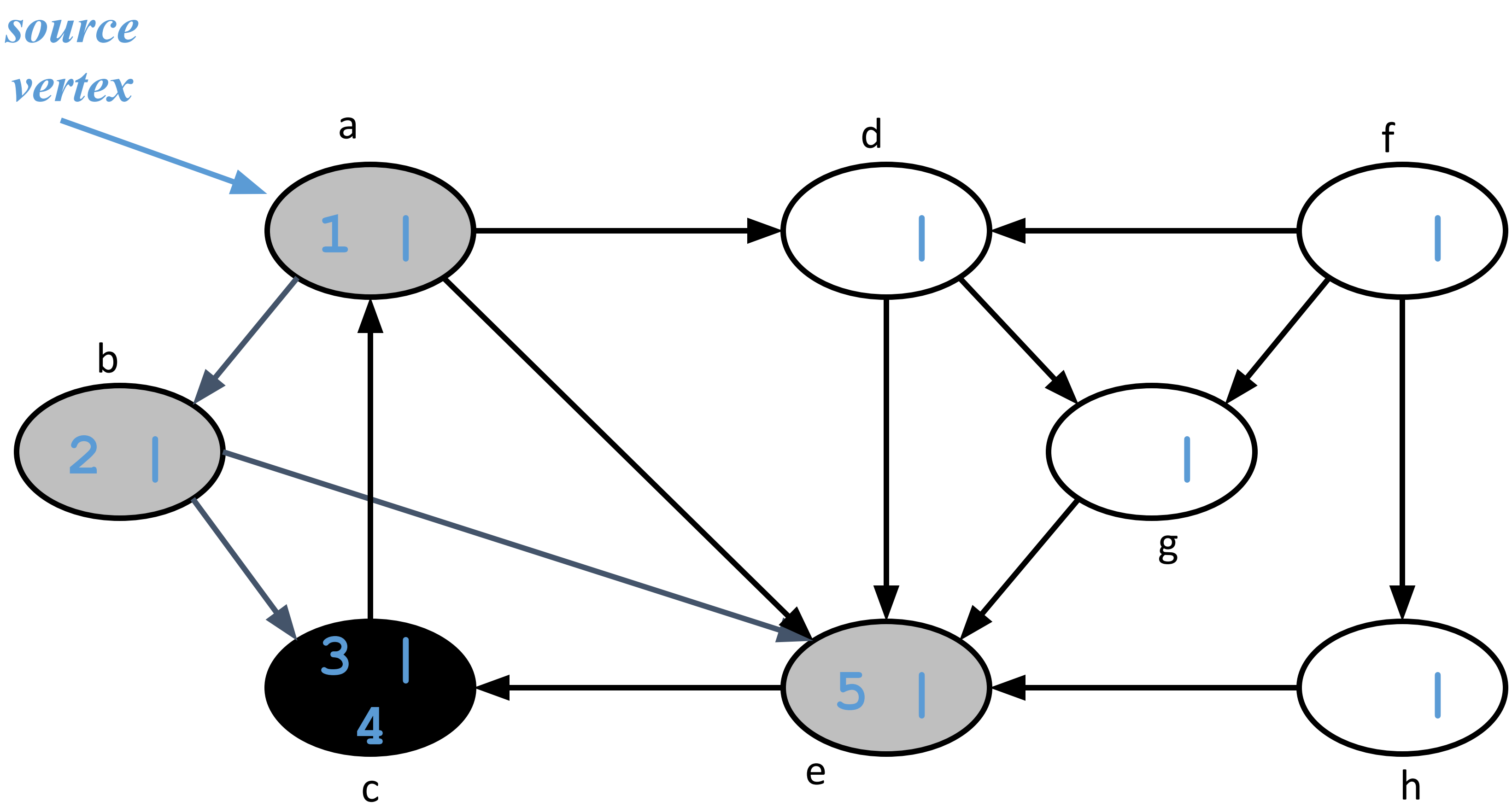
DFS Example



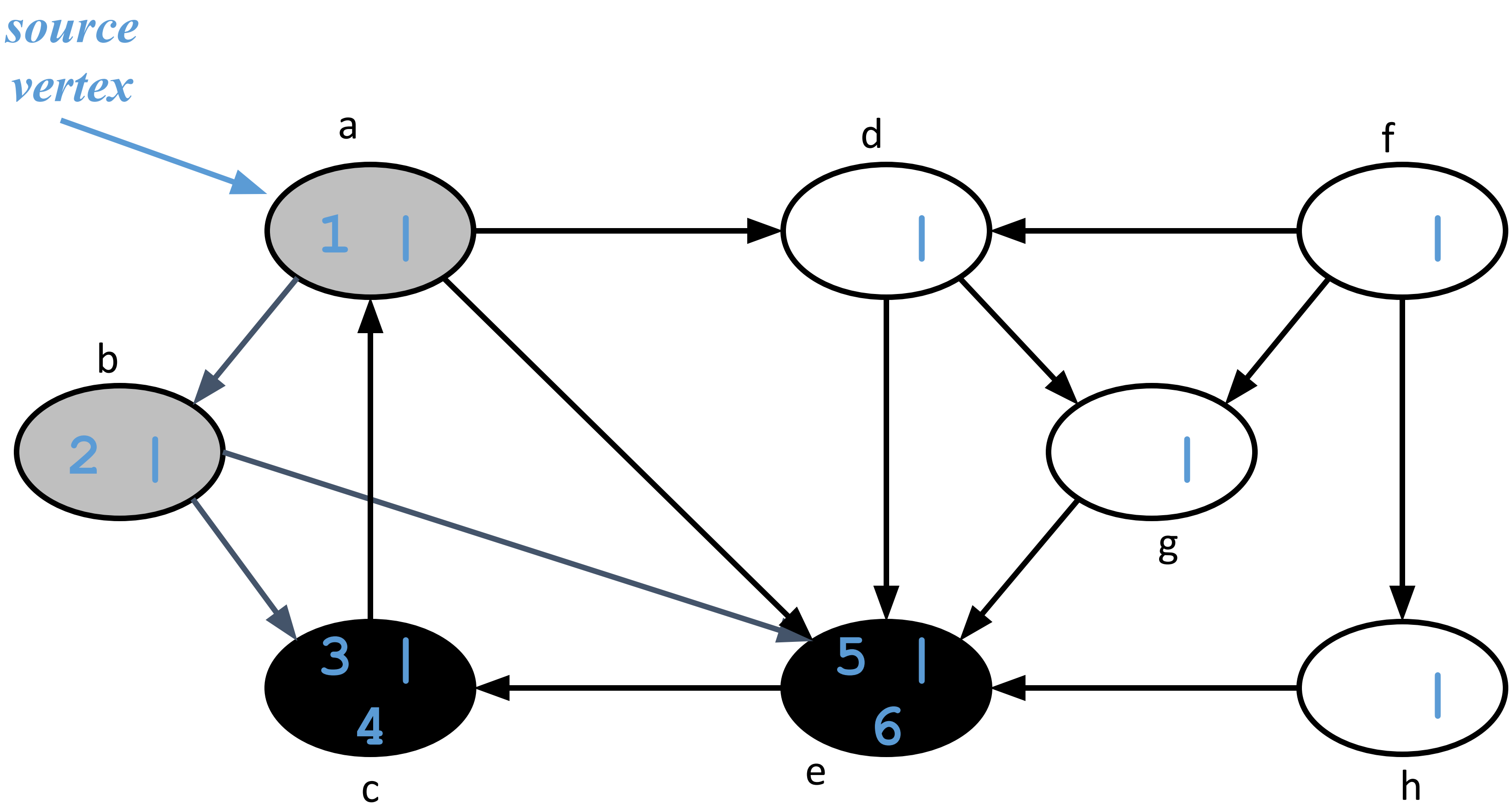
DFS Example



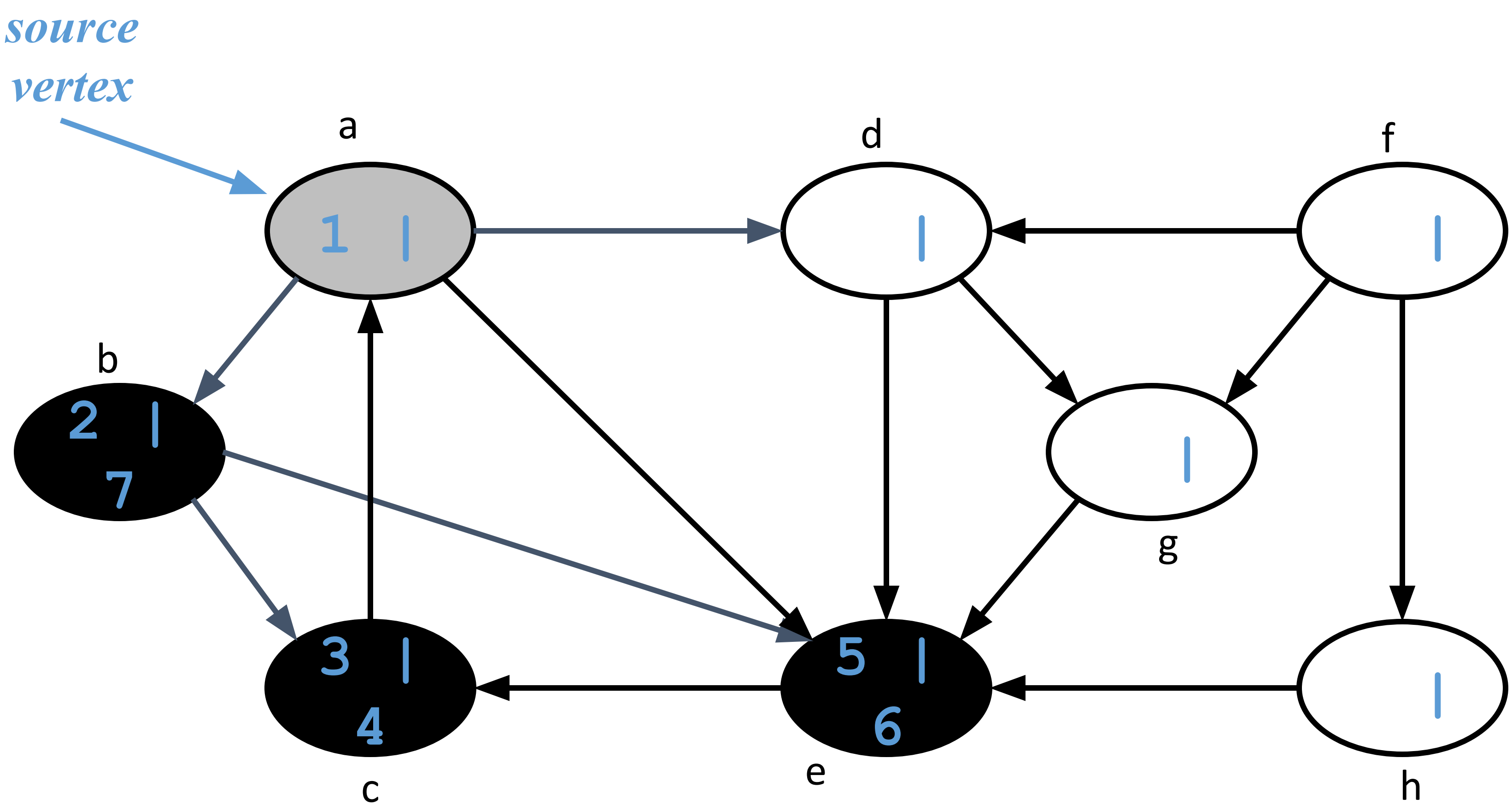
DFS Example



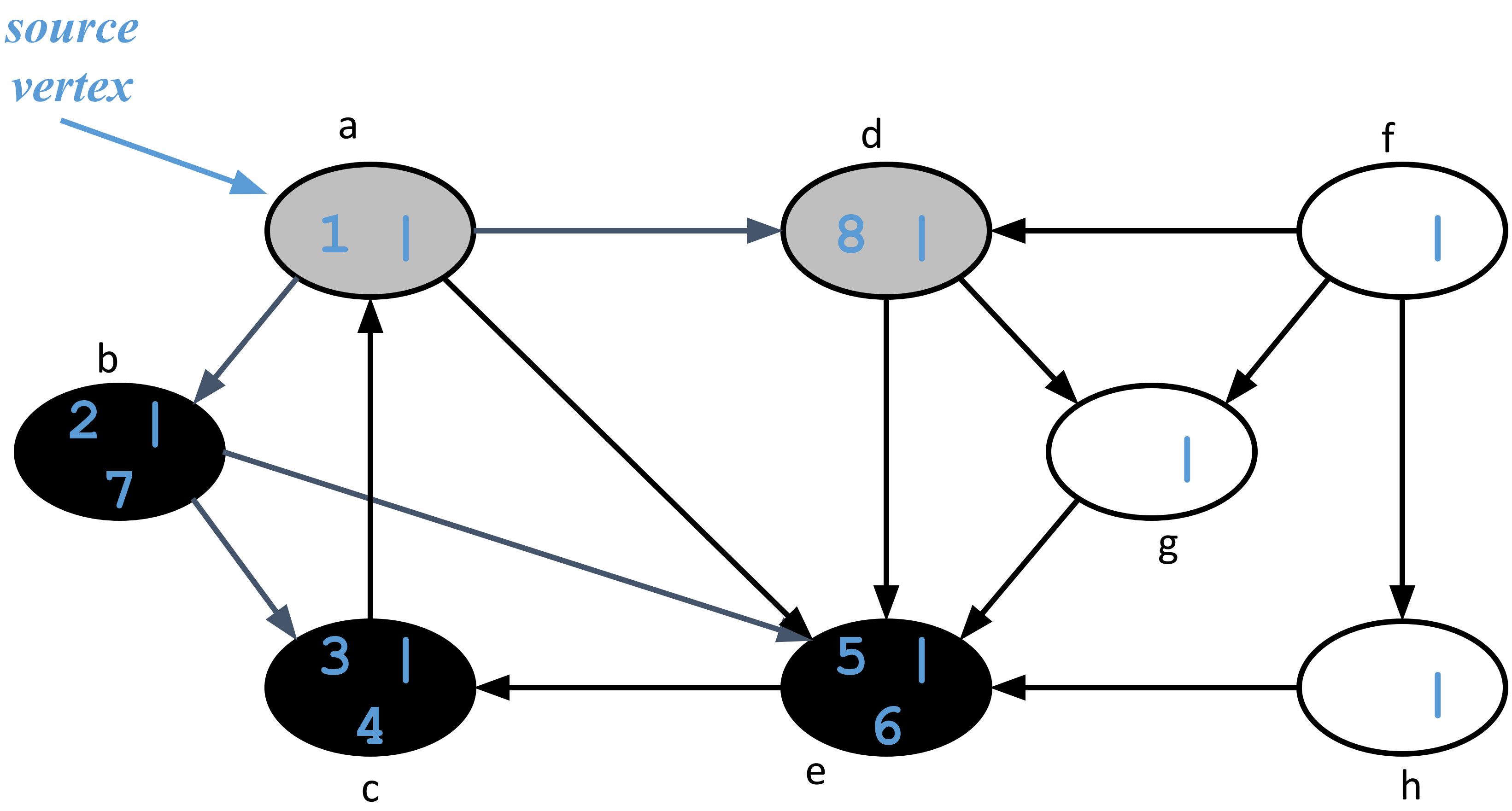
DFS Example



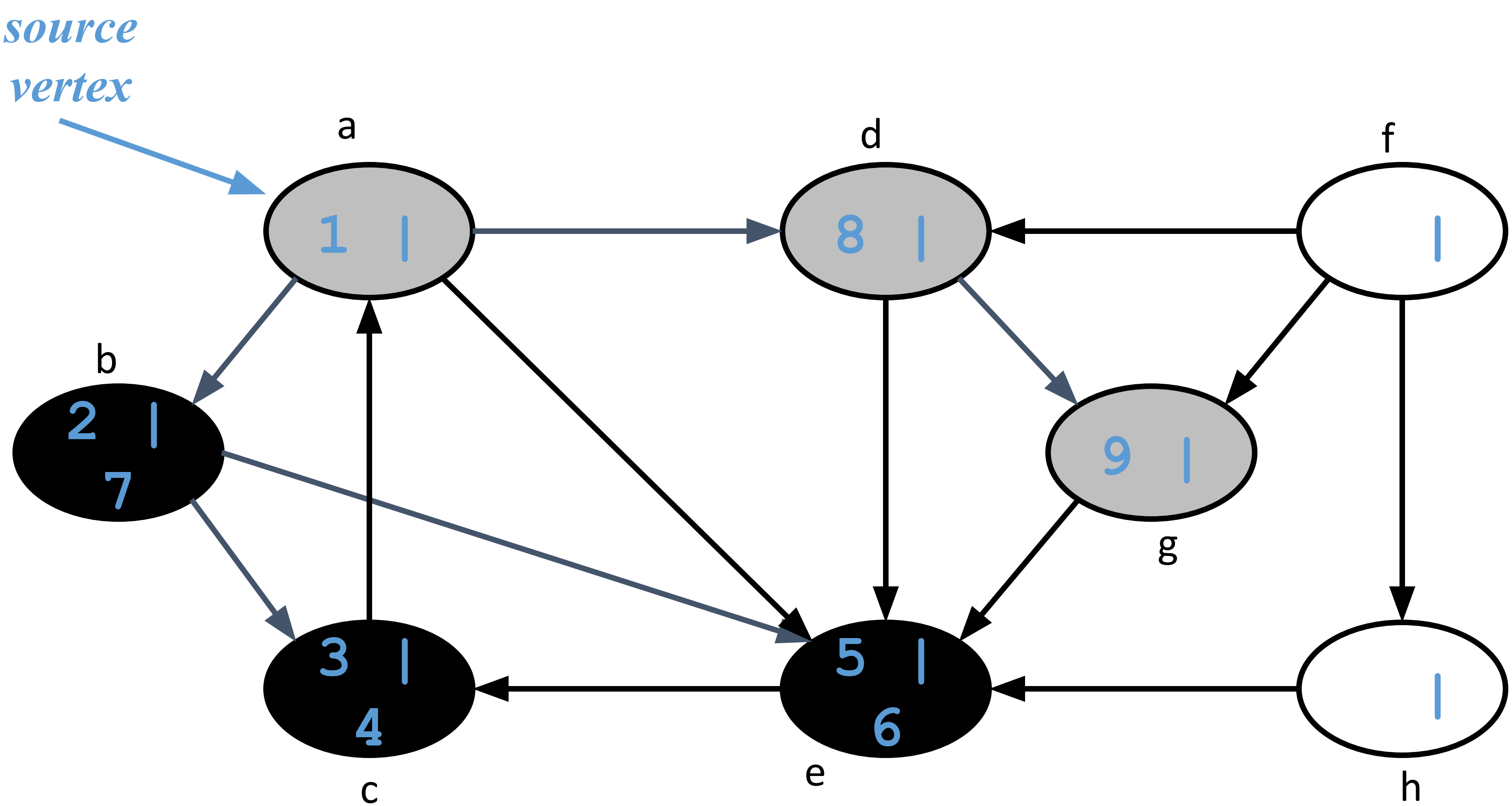
DFS Example



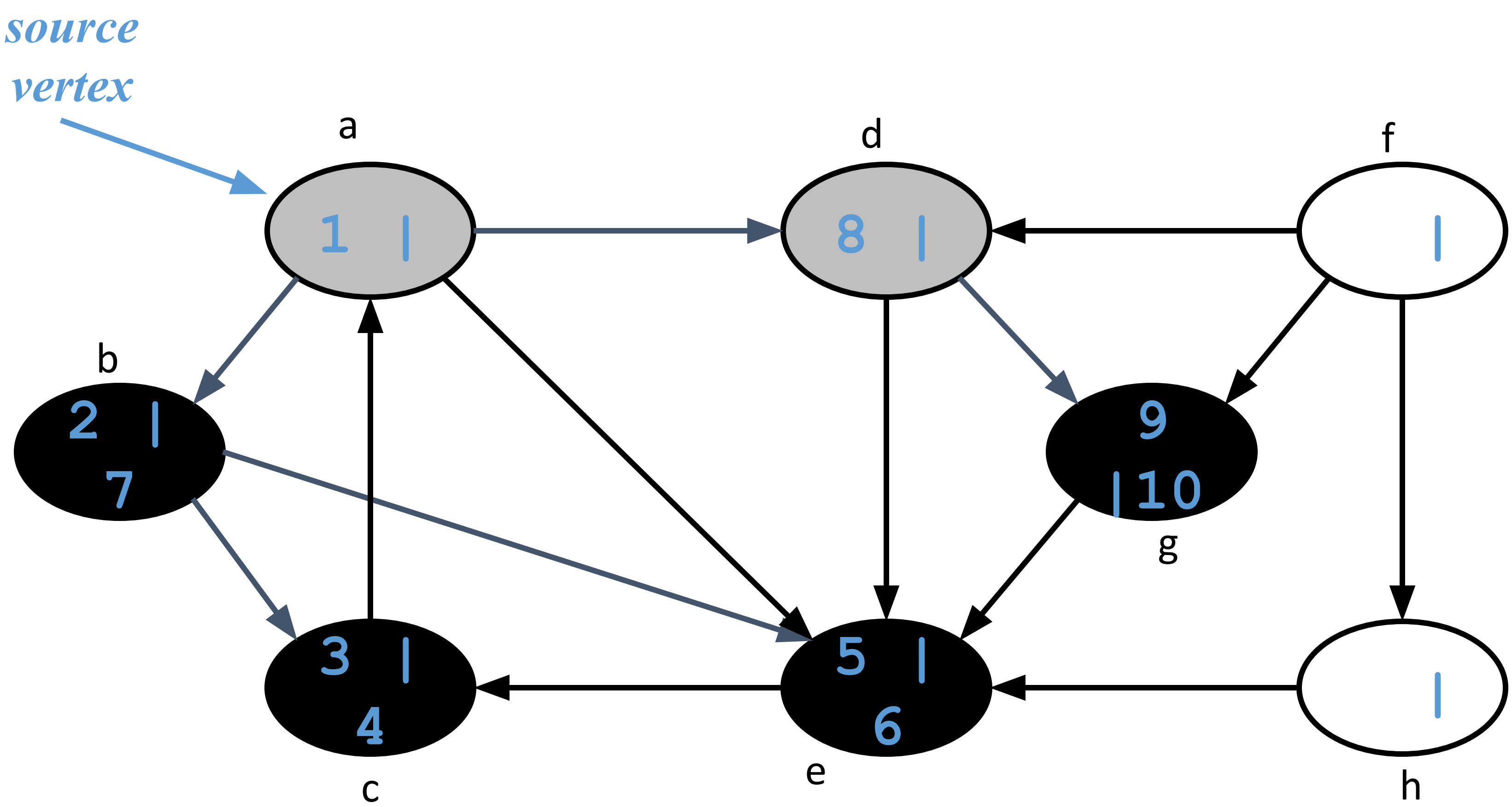
DFS Example



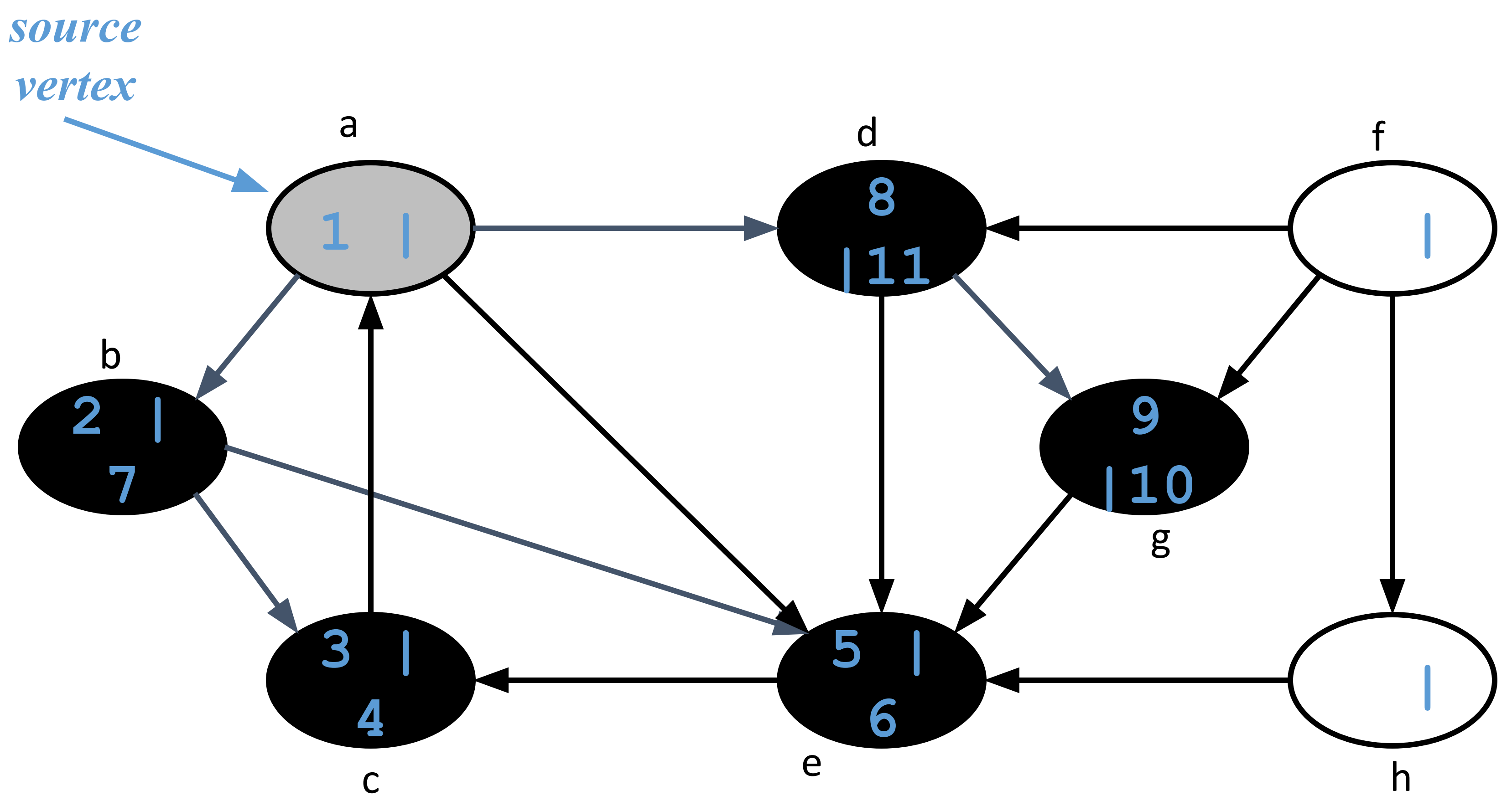
DFS Example



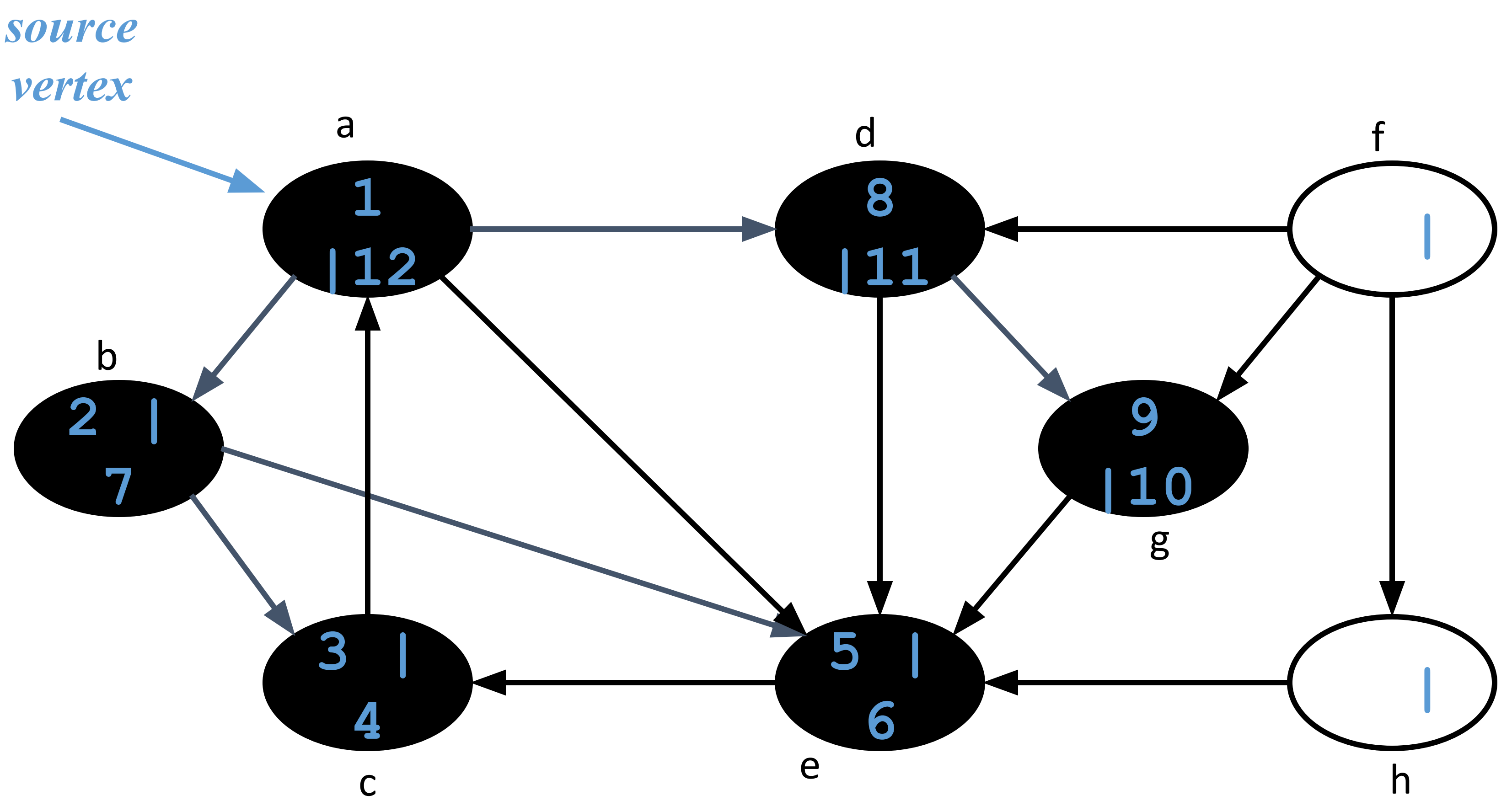
DFS Example



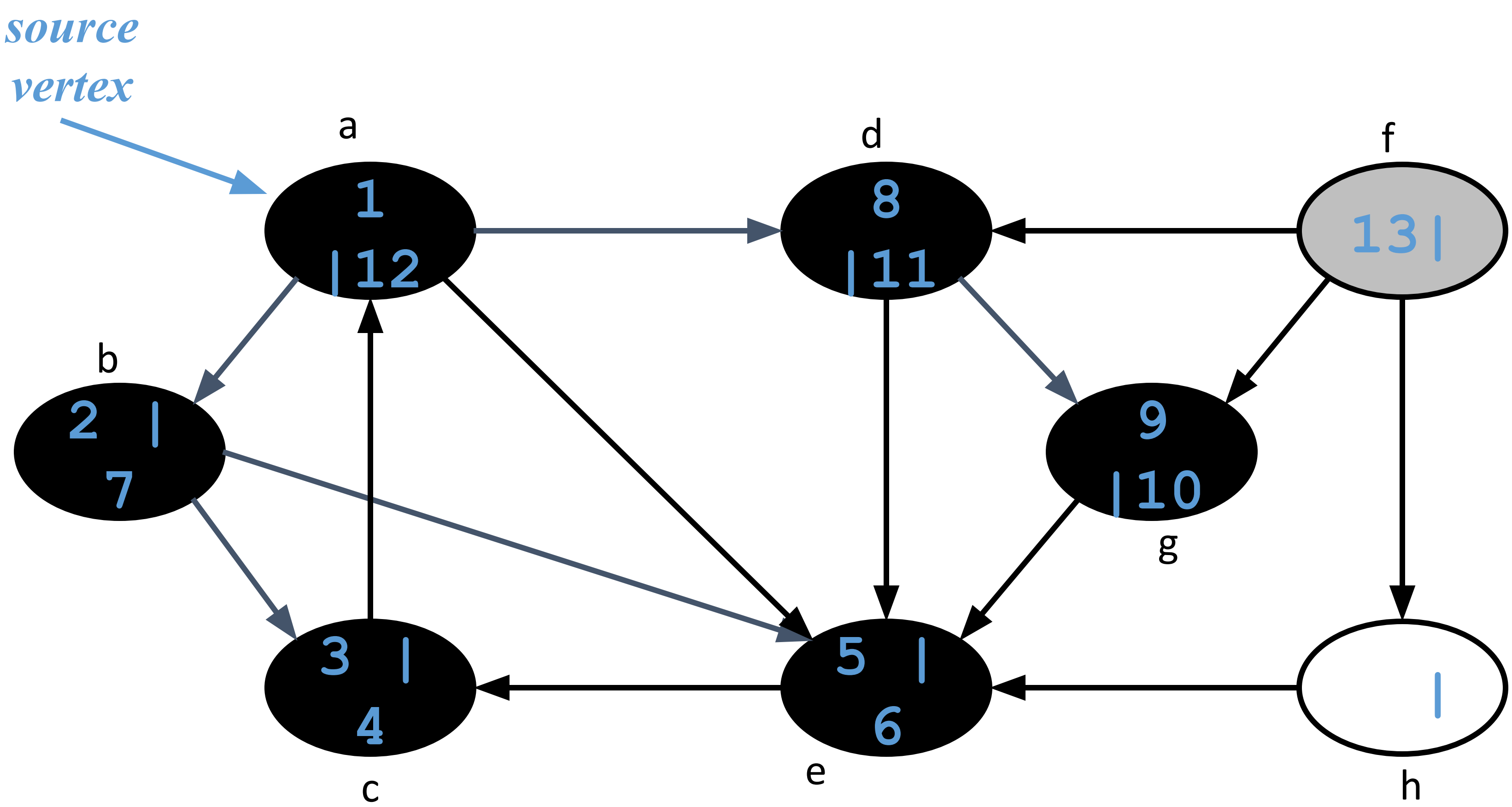
DFS Example



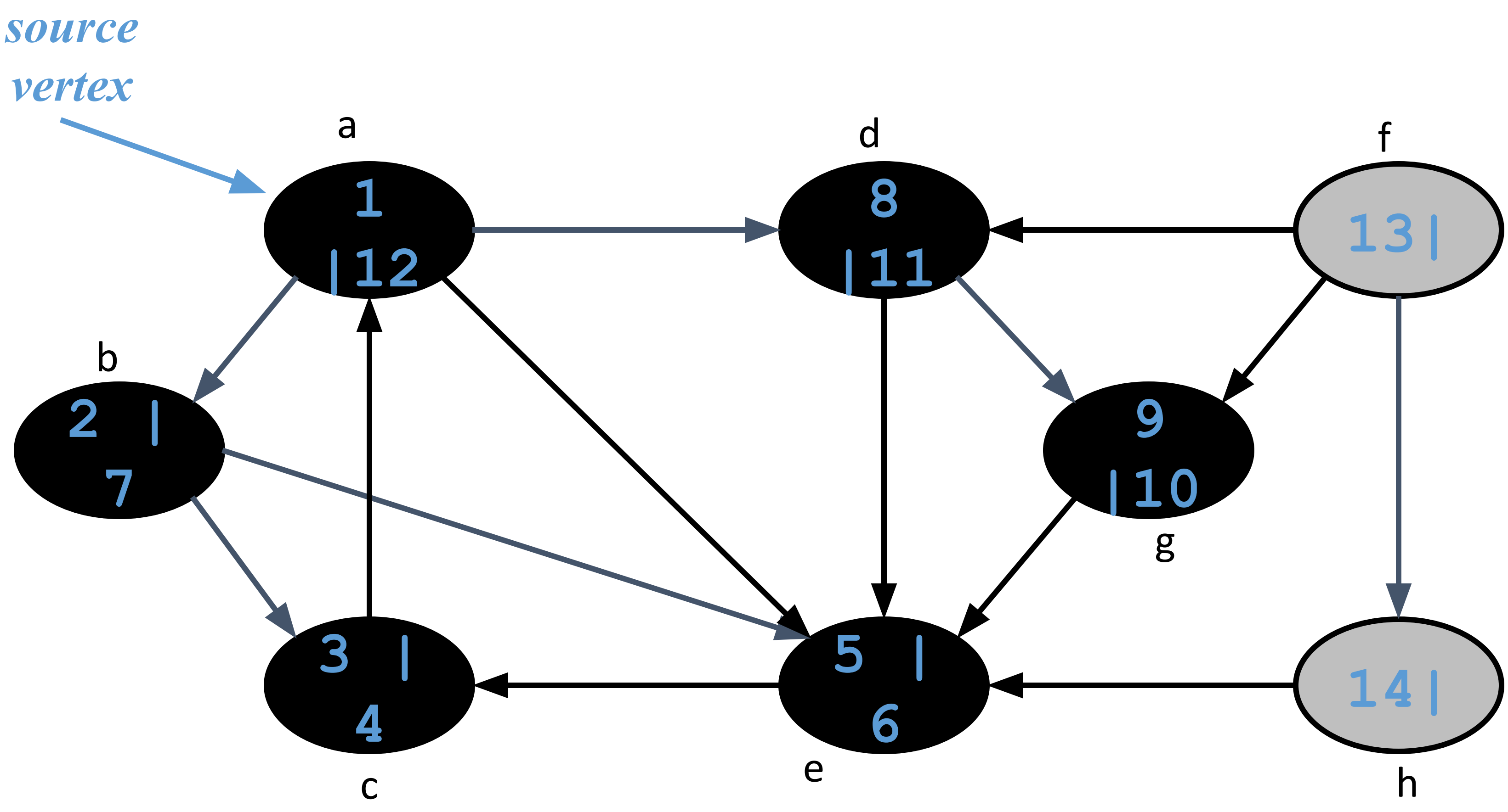
DFS Example



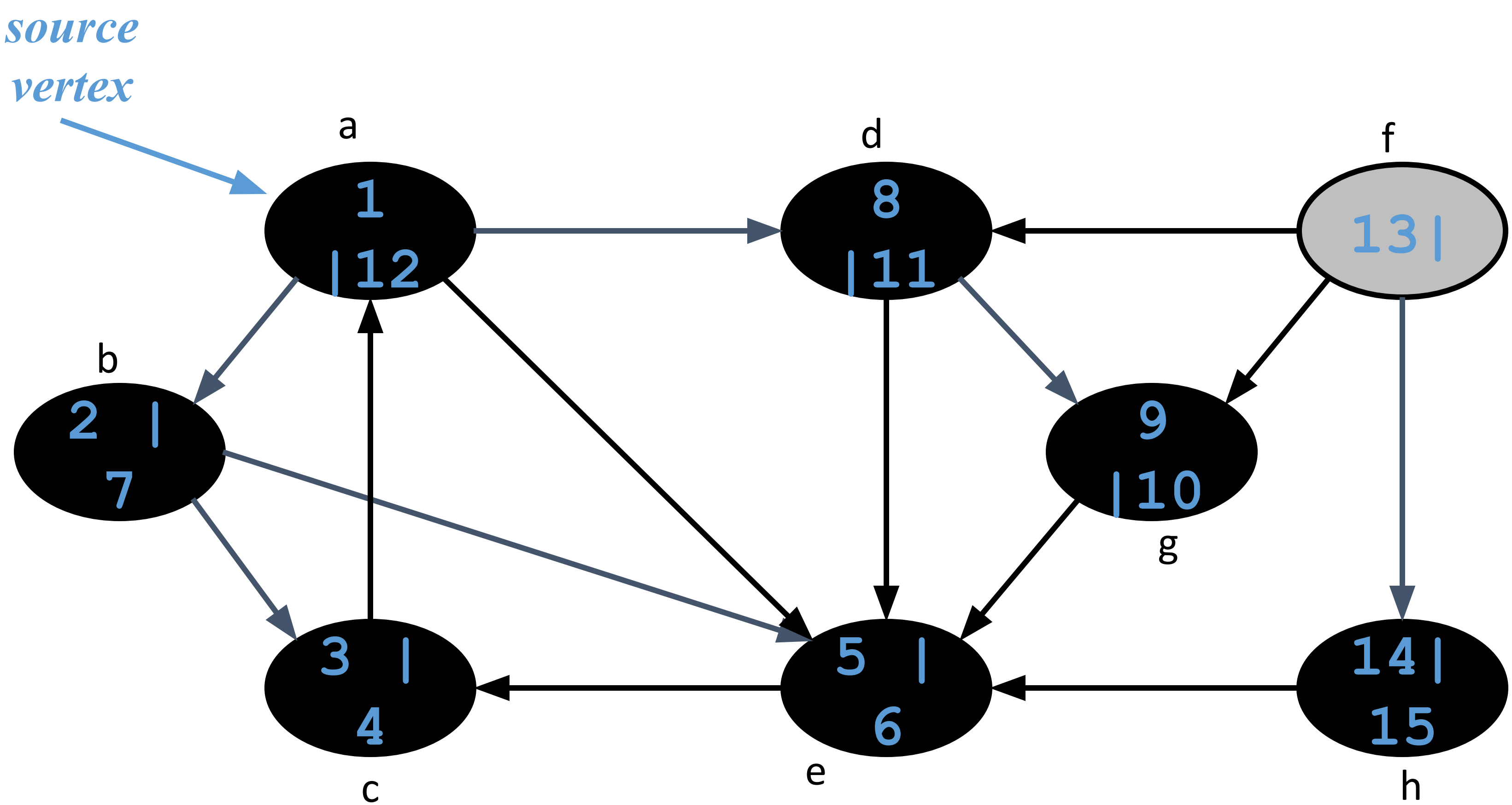
DFS Example



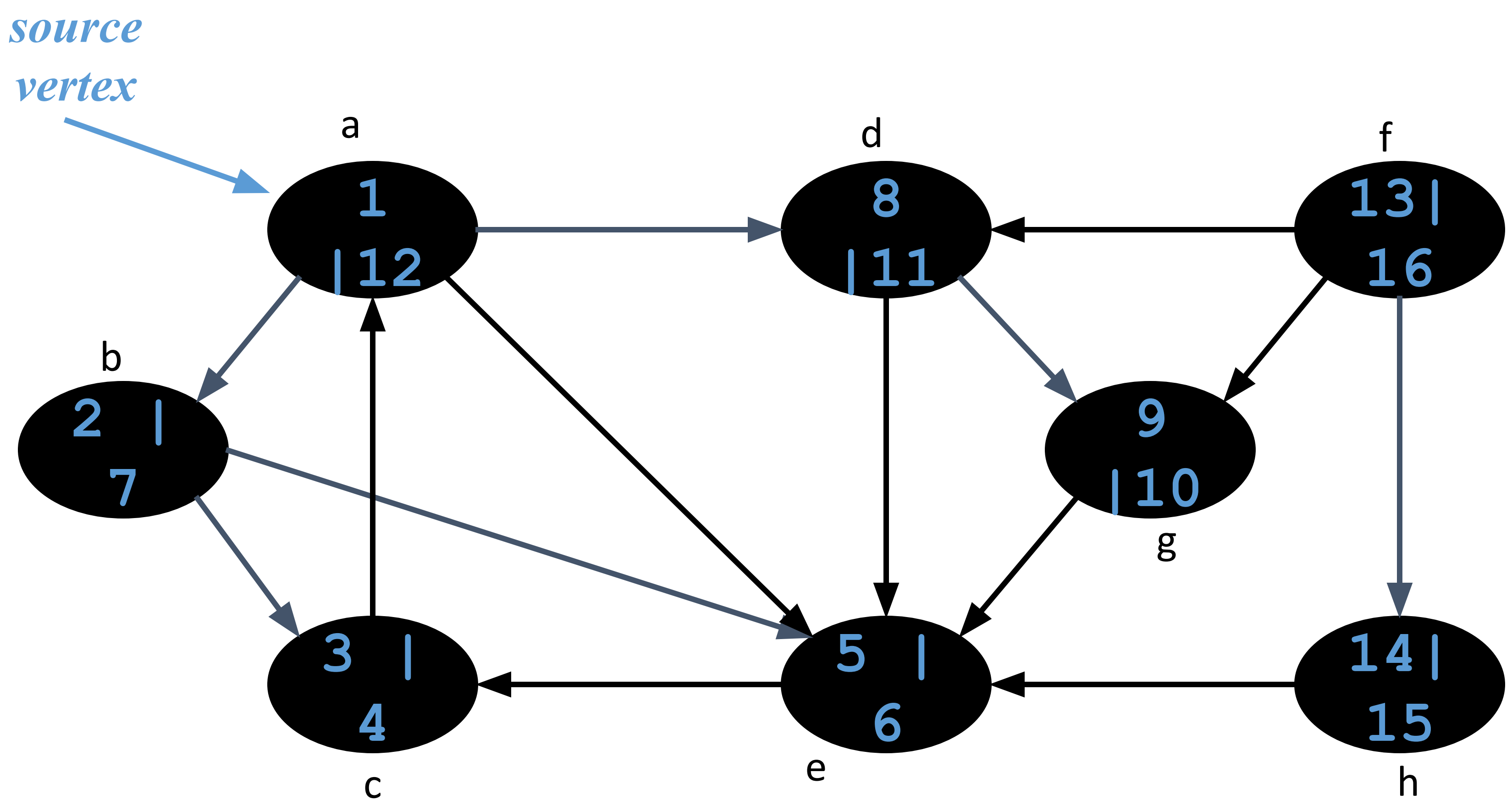
DFS Example



DFS Example



DFS Example



Running Time of DFS

initialization takes ***O(V)*** time

second for loop in non-recursive wrapper considers each vertex, so ***O(V)*** iterations

one recursive call is made for each vertex

in recursive call for vertex u, all its neighbors are checked; total time in all recursive calls is ***O(E)***

Total time is ***O(V+E)***