Breadth First Search

Subhabrata Samajder

IIIT, Delhi
Winter Semester,
12th May, 2023

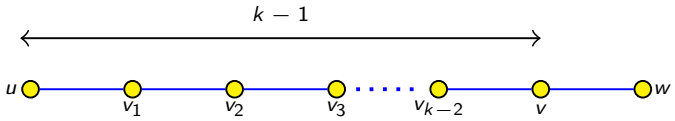
We shall introduce this traversal technique through an interesting problem: Computing distances from a vertex.

Notations and Observations

Length of a path: Is the number of edges on the path.

Example: A path of length 6 between u and v.

Question: If $uv_1 \cdots v_{k-2}v$ is a path of length $k-1$ from u to v, then what is the length of the path $uv_1 \cdots v_{k-2}vw$?

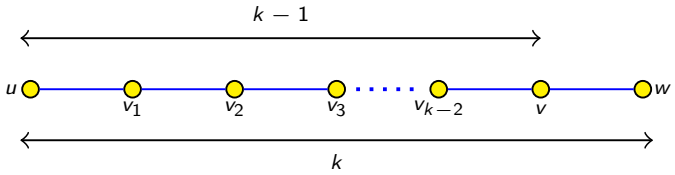


Notations and Observations

Length of a path: Is the number of edges on the path.

Example: A path of length 6 between u and v.

Question: If $uv_1 \cdots v_{k-2}v$ is a path of length $k-1$ from u to v, then what is the length of the path $uv_1 \cdots v_{k-2}vw$?



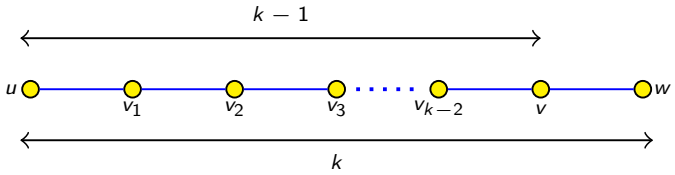
Answer: k.

Notations and Observations

Length of a path: Is the number of edges on the path.

Example: A path of length 6 between u and v.

Question: If $uv_1 \cdots v_{k-2}v$ is a path of length $k-1$ from u to v, then what is the length of the path $uv_1 \cdots v_{k-2}vw$?



Answer: k.

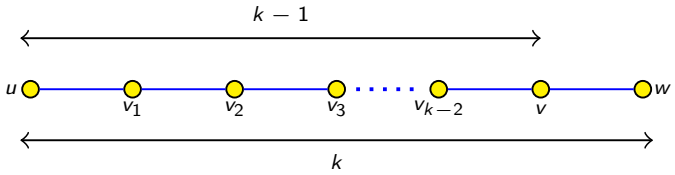
Question: What can be the maximum possible length of any path in a graph?

Notations and Observations

Length of a path: Is the number of edges on the path.

Example: A path of length 6 between u and v.

Question: If $uv_1 \cdots v_{k-2}v$ is a path of length $k-1$ from u to v, then what is the length of the path $uv_1 \cdots v_{k-2}vw$?



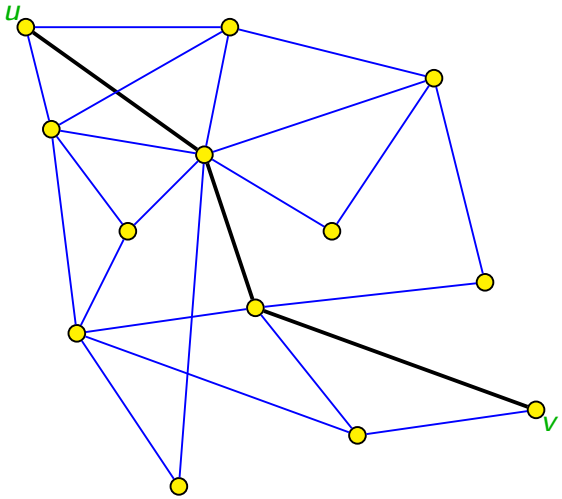
Answer: k.

Question: What can be the maximum possible length of any path in a graph?

Answer: $|V| - 1$.

Notations and Observations (Cont.)

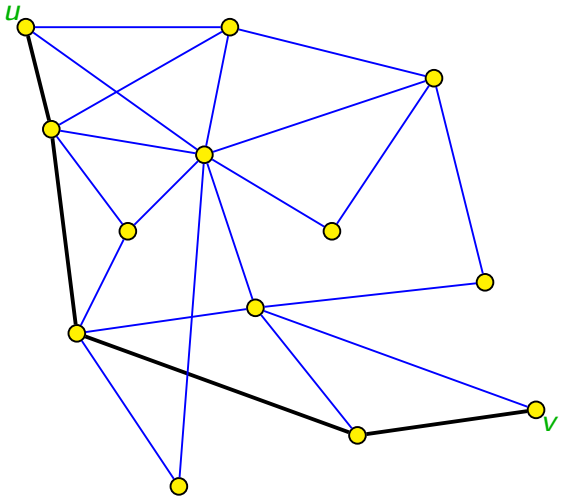
Shortest path from u to v: A path from u to v of least length.



Paths from u to v

Notations and Observations (Cont.)

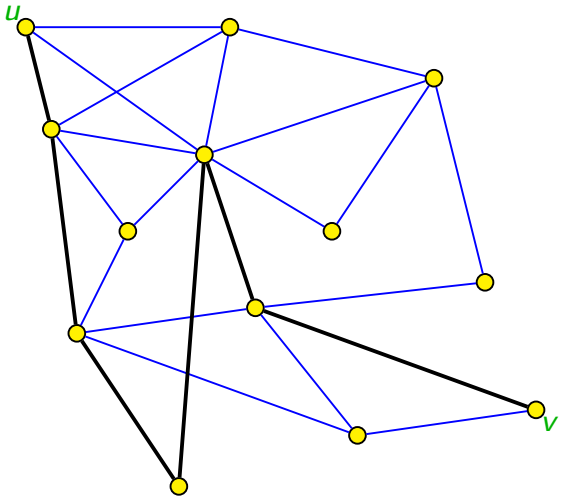
Shortest path from u to v: A path from u to v of least length.



Paths from u to v

Notations and Observations (Cont.)

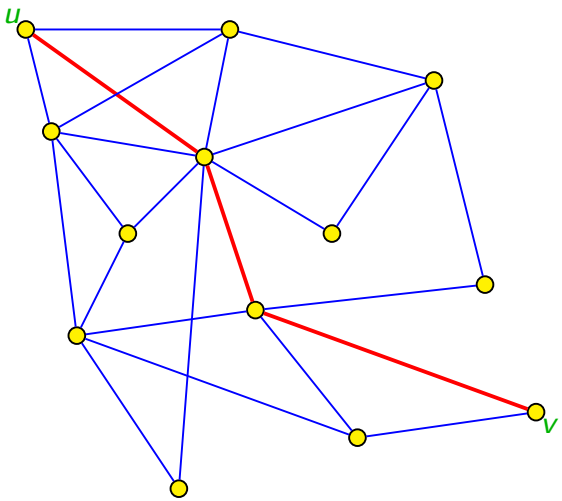
Shortest path from u to v: A path from u to v of least length.



Paths from u to v

Notations and Observations (Cont.)

Shortest path from u to v: A path from u to v of least length.

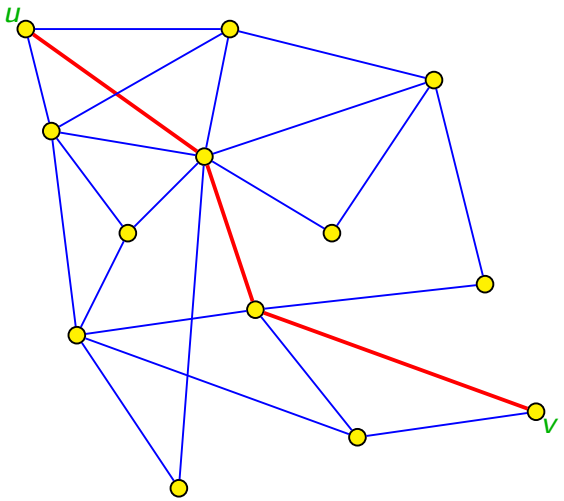


Shortest Path from u to v

Note: In general, may not be unique!

Notations and Observations (Cont.)

Shortest path from u to v: A path from u to v of least length.



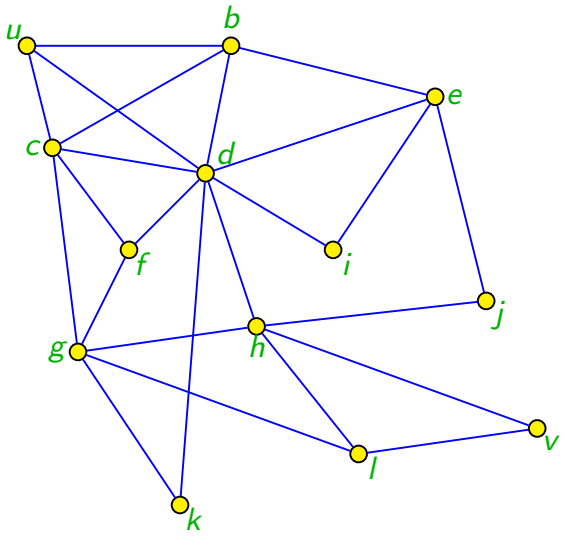
Shortest Path from u to v

Note: In general, may not be unique!

$\delta(u, v)$: Length of the Shortest path distance from u to v.

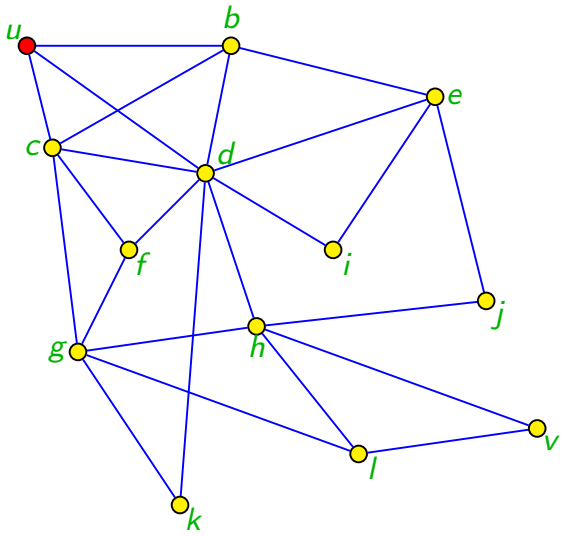
Convention: $\delta(u, v) = \infty$, if there is no path from u to v.

Shortest Path in Undirected Graphs



Problem: How to compute distance to all vertices reachable from u in a given a undirected graph?

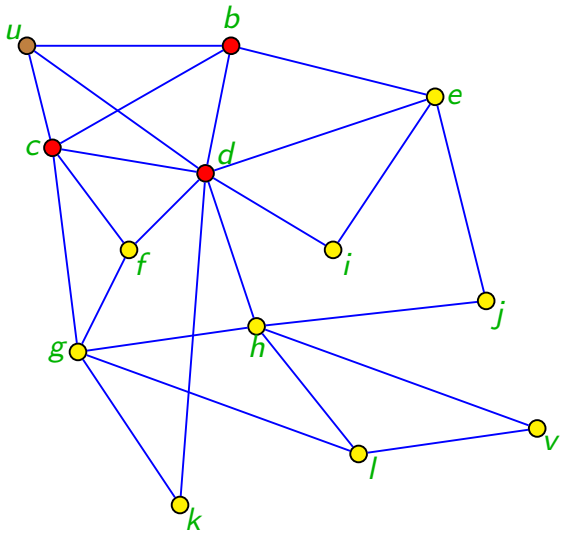
Shortest Path in Undirected Graphs



Problem: How to compute distance to all vertices reachable from u in a given a undirected graph?

Vertices at distance 0 from u:

Shortest Path in Undirected Graphs

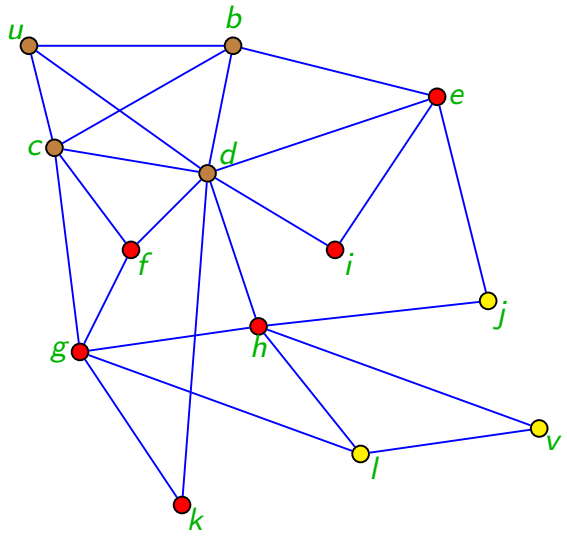


Problem: How to compute distance to all vertices reachable from u in a given a undirected graph?

Vertices at distance 0 from u: $\{u\}$

Vertices at distance 1 from u:

Shortest Path in Undirected Graphs



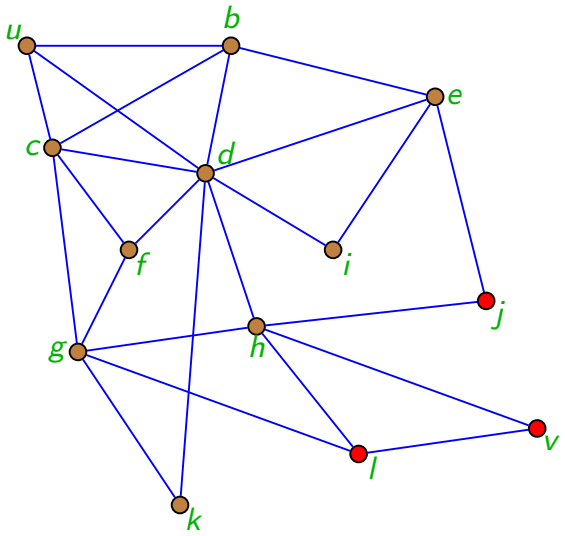
Problem: How to compute distance to all vertices reachable from u in a given a undirected graph?

Vertices at distance 0 from u: $\{u\}$

Vertices at distance 1 from u: $\{b, c, d\}$

Vertices at distance 2 from u:

Shortest Path in Undirected Graphs



Problem: How to compute distance to all vertices reachable from u in a given a undirected graph?

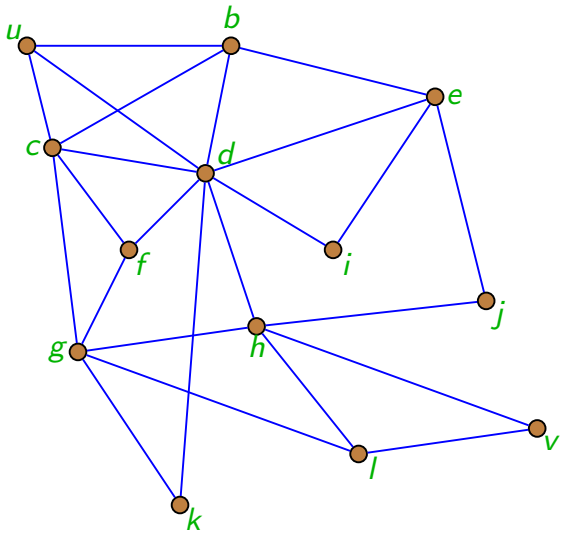
Vertices at distance 0 from u: $\{u\}$

Vertices at distance 1 from u: $\{b, c, d\}$

Vertices at distance 2 from u: $\{e, f, g, h, i, k\}$

Vertices at distance 3 from u:

Shortest Path in Undirected Graphs



Problem: How to compute distance to all vertices reachable from u in a given a undirected graph?

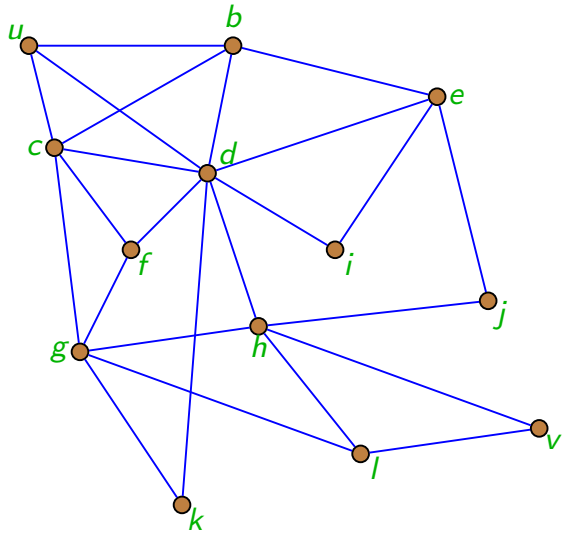
Vertices at distance 0 from u: $\{u\}$

Vertices at distance 1 from u: $\{b, c, d\}$

Vertices at distance 2 from u: $\{e, f, g, h, i, k\}$

Vertices at distance 3 from u: $\{j, l, v\}$

Shortest Path in Undirected Graphs



Problem: How to compute distance to all vertices reachable from u in a given a undirected graph?

Vertices at distance 0 from u: $\{u\}$

Vertices at distance 1 from u: $\{b, c, d\}$

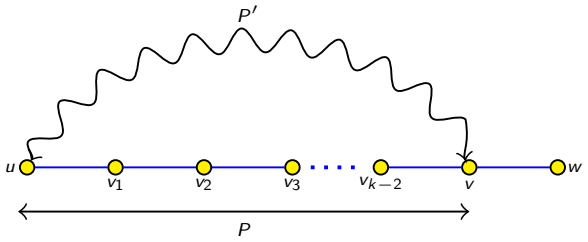
Vertices at distance 2 from u: $\{e, f, g, h, i, k\}$

Vertices at distance 3 from u: $\{j, l, v\}$

Note: While reporting, you have (sub)consciously used an important property of shortest paths. Can you state this property?

An Important Property of Shortest Paths

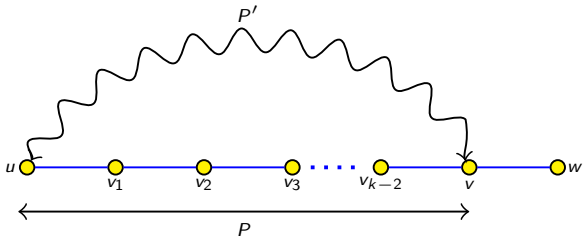
Let $P = uv_1 \cdots v_{k-2}v$.



Question: If $uv_1 \cdots v_{k-2}vw$ is a shortest path from u to w, then what can you say about P?

An Important Property of Shortest Paths

Let $P = uv_1 \cdots v_{k-2}v$.

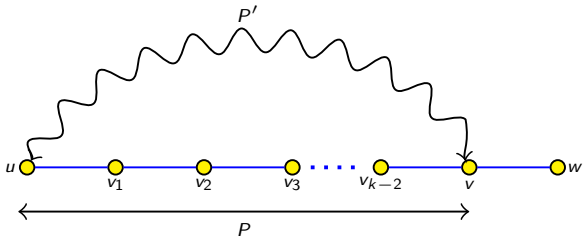


Question: If $uv_1 \cdots v_{k-2}vw$ is a shortest path from u to w, then what can you say about P?

Observation: If $uv_1 \cdots v_{k-2}vw$ is a shortest path from u to w, then P is also a shortest path from u to v.

An Important Property of Shortest Paths

Let $P = uv_1 \cdots v_{k-2}v$.



Question: If $uv_1 \cdots v_{k-2}vw$ is a shortest path from u to w, then what can you say about P?

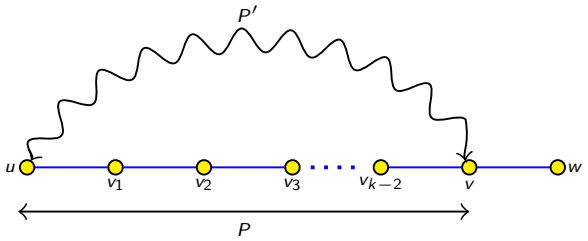
Observation: If $uv_1 \cdots v_{k-2}vw$ is a shortest path from u to w, then P is also a shortest path from u to v.

Proof: The proof is via contradiction. Suppose, if possible, P is not a shortest path between u and v. Let P' be a shortest path between u and v. Then,

$$Length(P') < Length(P).$$

An Important Property of Shortest Paths

Let $P = uv_1 \cdots v_{k-2}v$.



Question: If $uv_1 \cdots v_{k-2}vw$ is a shortest path from u to w, then what can you say about P?

Observation: If $uv_1 \cdots v_{k-2}vw$ is a shortest path from u to w, then P is also a shortest path from u to v.

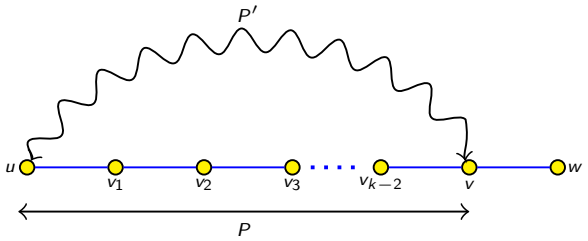
Proof: The proof is via contradiction. Suppose, if possible, P is not a shortest path between u and v. Let P' be a shortest path between u and v. Then,

$$Length(P') < Length(P).$$

Question: What happens if we concatenate P' with edge (v, w)?

An Important Property of Shortest Paths

Let $P = uv_1 \cdots v_{k-2}v$.



Question: If $uv_1 \cdots v_{k-2}vw$ is a shortest path from u to w, then what can you say about P?

Observation: If $uv_1 \cdots v_{k-2}vw$ is a shortest path from u to w, then P is also a shortest path from u to v.

Proof: The proof is via contradiction. Suppose, if possible, P is not a shortest path between u and v. Let P' be a shortest path between u and v. Then,

$$Length(P') < Length(P).$$

Question: What happens if we concatenate P' with edge (v, w)?
Ans: $Length(P') + 1 < Length(P) + 1 = Length(uv_1 \cdots v_{k-2}vw)$.
$\Rightarrow\Leftarrow$

Relationship Among Vertices at Different Distances from u

V_0: Vertices at distance 0 from $u = \{u\}$.

V_1: Vertices at distance 1 from u = Neighbours of V_0.

V_2: Vertices at distance 2 from u = Neighbours of V_1 that do not belong to $V_0 \cup V_1$.

$\vdots$

V_{i+1}: Vertices at distance $i+1$ from u = Neighbours of V_i that do not belong to $V_{i-1} \cup V_i$.

Relationship Among Vertices at Different Distances from u

V_0: Vertices at distance 0 from $u = \{u\}$.

V_1: Vertices at distance 1 from u = Neighbours of V_0.

V_2: Vertices at distance 2 from u = Neighbours of V_1 that do not belong to $V_0 \cup V_1$.

$\vdots$

V_{i+1}: Vertices at distance $i+1$ from u = Neighbours of V_i that do not belong to $V_{i-1} \cup V_i$.

Question: How to distinguish the neighbors of V_i which belong to V_{i+1} from those which belong to V_j, $j \leq i$?

How Can We Compute V_{i+1}?

Key idea: Compute V_i's in increasing order of i.

Initialize:

- $Distance[v] \leftarrow \infty$ for each vertex $v \in V \setminus \{u\}$.
- $Distance[u] \leftarrow 0$.

Computing V_i's:

- First compute V_0
- Then compute V_1
- ...
- Once we have computed V_i, then for every neighbor v of a vertex in V_i, we have
  - If $v \in V_j$ for some $j \in \{0, 1, 2, \ldots, i-1\}$, then $Distance[v] < i$.
  - If $v \in V_{i+1}$, then $Distance[v] = \infty$.

How Can We Compute V_{i+1}?

Key idea: Compute V_i's in increasing order of i.

Initialize:

- $Distance[v] \leftarrow \infty$ for each vertex $v \in V \setminus \{u\}$.
- $Distance[u] \leftarrow 0$.

Computing V_i's:

- First compute V_0
- Then compute V_1
- ...
- Once we have computed V_i, then for every neighbor v of a vertex in V_i, we have
  - If $v \in V_j$ for some $j \in \{0, 1, 2, \ldots, i-1\}$, then $Distance[v] < i$.
  - If $v \in V_{i+1}$, then $Distance[v] = \infty$.

We can thus distinguish the neighbors of V_i which belong to V_{i+1} from those which belong to V_j.

A Neat Algorithm for Computing Distances from u

Conclusion: We therefore need an algorithm which traverses/visits the vertices in non-decreasing order of distances from u.

Inherent Ordering: Thus, all vertices at distance k needs to be processed before vertices at distance $k+1$ (FIFO).

A Neat Algorithm for Computing Distances from u

Conclusion: We therefore need an algorithm which traverses/visits the vertices in non-decreasing order of distances from u.

Inherent Ordering: Thus, all vertices at distance k needs to be processed before vertices at distance $k+1$ (FIFO). This is achieved by using a Queue.

A Neat Algorithm for Computing Distances from u

Conclusion: We therefore need an algorithm which traverses/visits the vertices in non-decreasing order of distances from u.

Inherent Ordering: Thus, all vertices at distance k needs to be processed before vertices at distance $k+1$ (FIFO). This is achieved by using a Queue.

This traversal algorithm is called breadth first search (BFS) traversal.

Breadth First Search (BFS)

Gray vs. Black Vertices

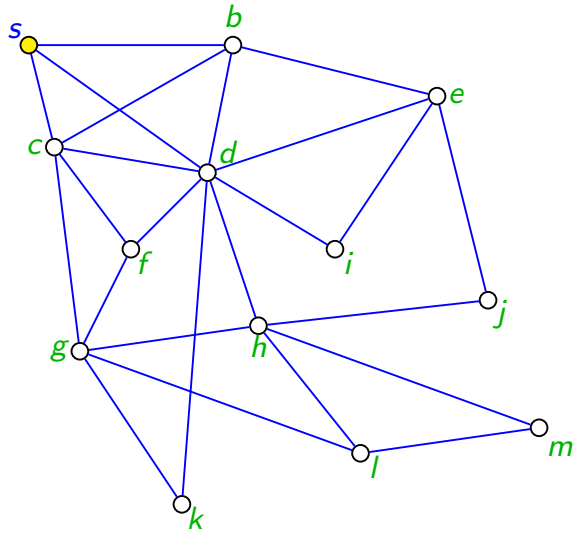
Black vertices: If $(u, v) \in E$ and vertex u is black, then

- vertex v is either gray or black.
- That is, *all vertices adjacent to black vertices have been* **discovered**.

Gray vertices:

- *Gray vertices may have some adjacent white vertices*
- They represent the frontier between discovered and undiscovered vertices.

BFS(G, s)

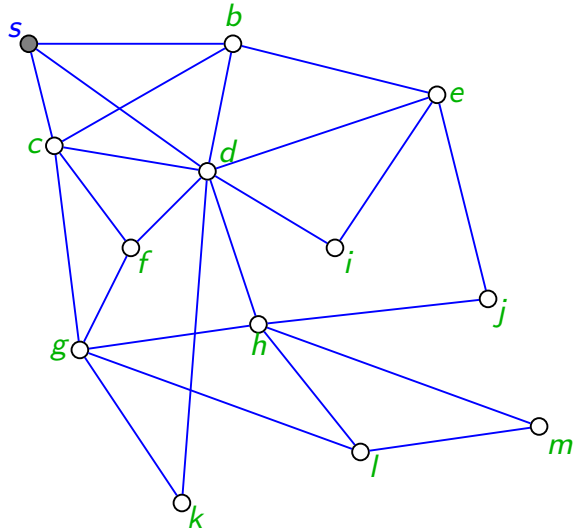


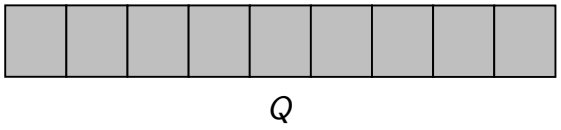
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

V_0: {}
V_1: {}
V_2: {}
V_3: {}

BFS(G, s)



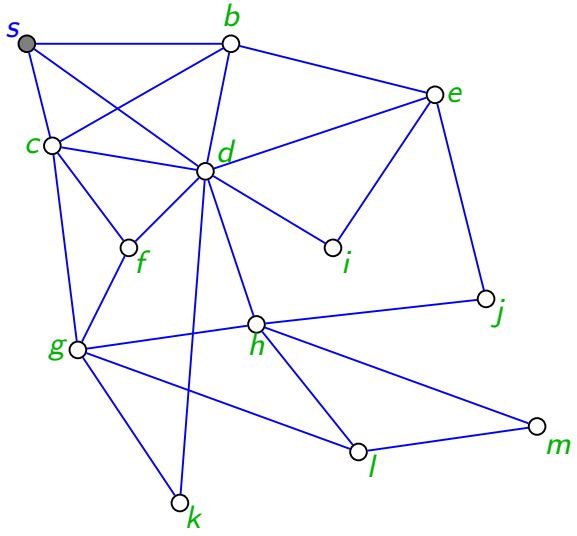


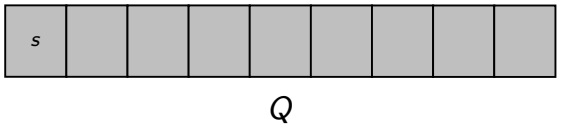
V_0: {}
V_1: {}
V_2: {}
V_3: {}

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u
  $color[s] \leftarrow$ GRAY;

BFS(G, s)



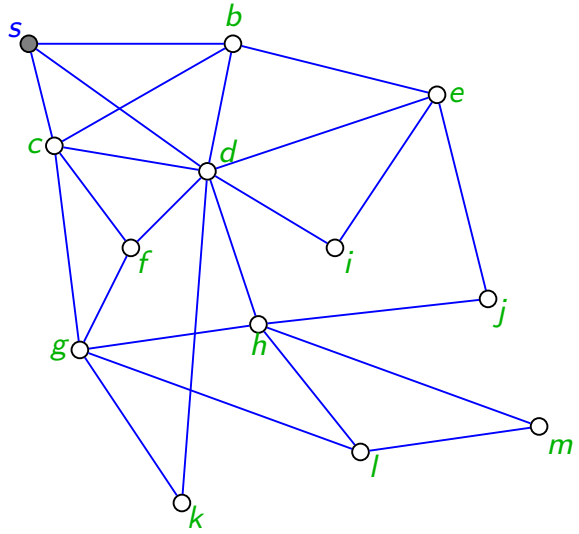


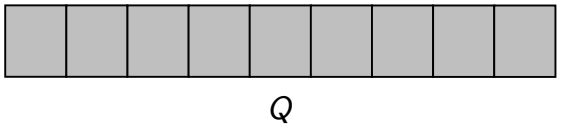
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

BFS(G, s)





V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

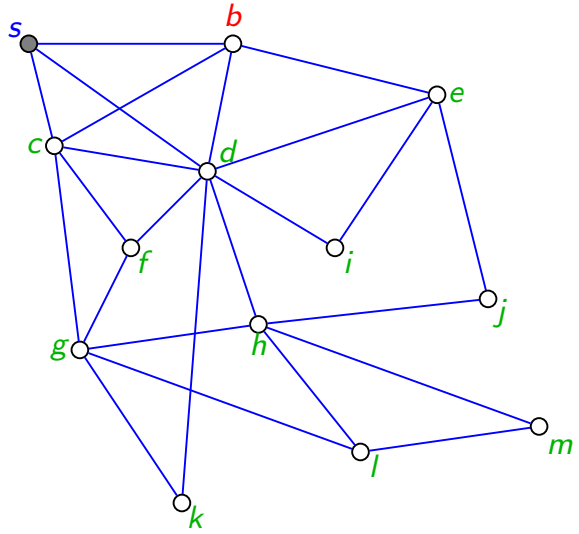
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

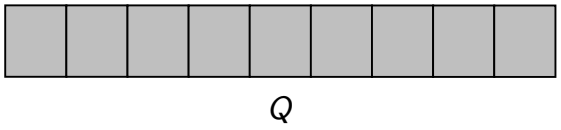
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);

BFS(G, s)





V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

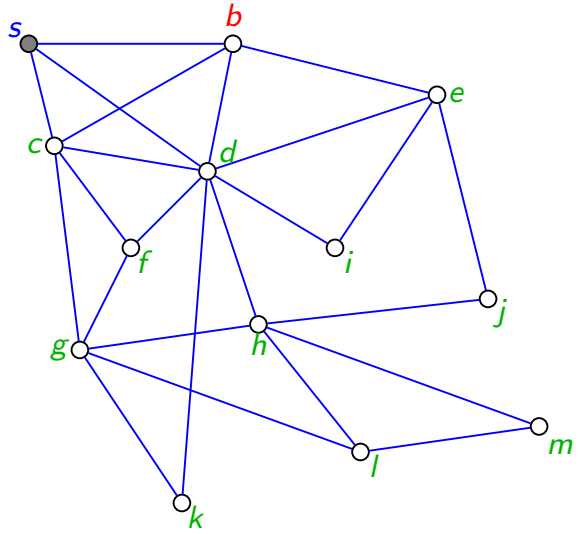
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

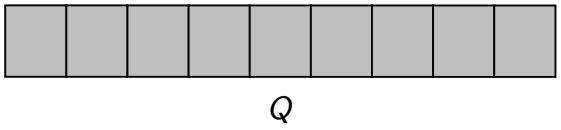
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {

BFS(G, s)





V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

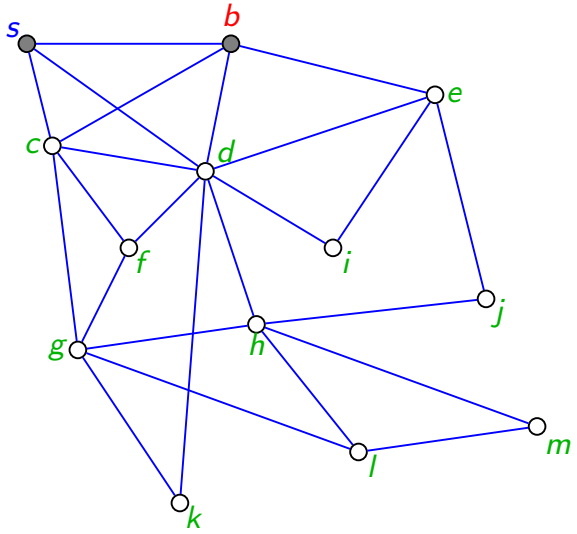
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

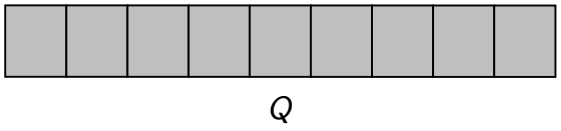
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {

BFS(G, s)





V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

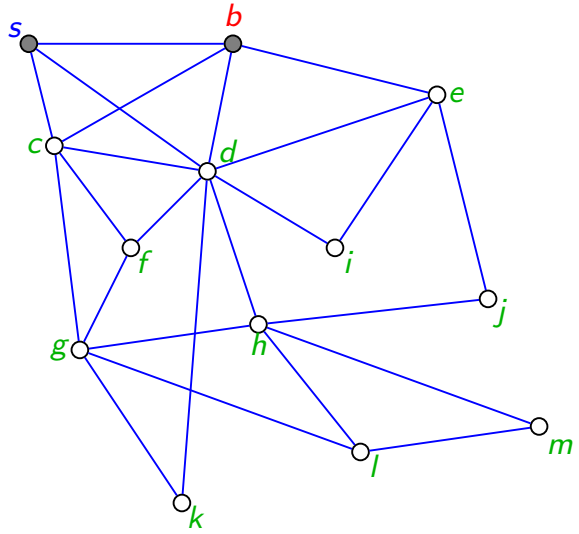
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

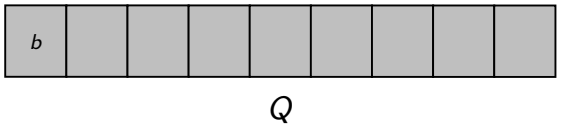
```
Begin
  for each vertex u ∈ V \ {s} {
    color[u] ← WHITE;
    d[u] ← ∞;     // d[u] = δ(s, u)
    π[u] ← NIL; }     // π[u] = predecessor of u

  color[s] ← GRAY;
  d[s] ← 0;
  π[s] ← NIL;
  Q ← ∅;     // Q denotes a queue
  ENQUEUE(Q, s);

  while (Q ≠ ∅) {
    u ← DEQUEUE(Q);
    for each v ∈ Adj[u] {
      if (color[v] = white) {
        color[v] ← GRAY;
```

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b\}$
V_2: $\{\}$
V_3: $\{\}$

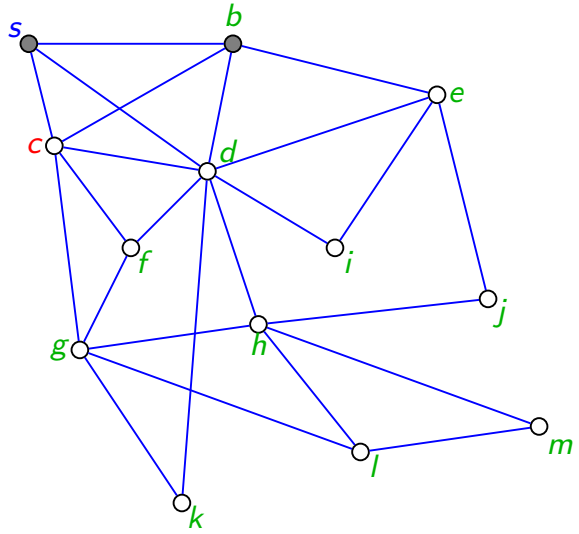
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

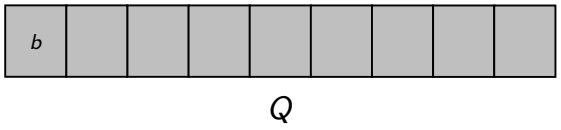
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b\}$
V_2: $\{\}$
V_3: $\{\}$

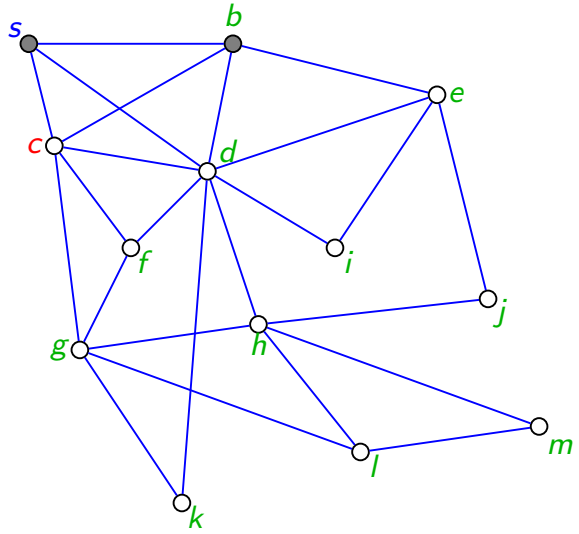
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

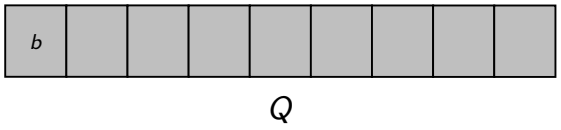
```
Begin
  for each vertex u ∈ V \ {s} {
    color[u] ← WHITE;
    d[u] ← ∞;     // d[u] = δ(s, u)
    π[u] ← NIL; }     // π[u] = predecessor of u

  color[s] ← GRAY;
  d[s] ← 0;
  π[s] ← NIL;
  Q ← ∅;     // Q denotes a queue
  ENQUEUE(Q, s);

  while (Q ≠ ∅) {
    u ← DEQUEUE(Q);
    for each v ∈ Adj[u] {
      if (color[v] = white) {
        color[v] ← GRAY;
        d[v] ← d[u] + 1;
        π[v] ← u;
        ENQUEUE(Q, v); } }
```

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b\}$
V_2: $\{\}$
V_3: $\{\}$

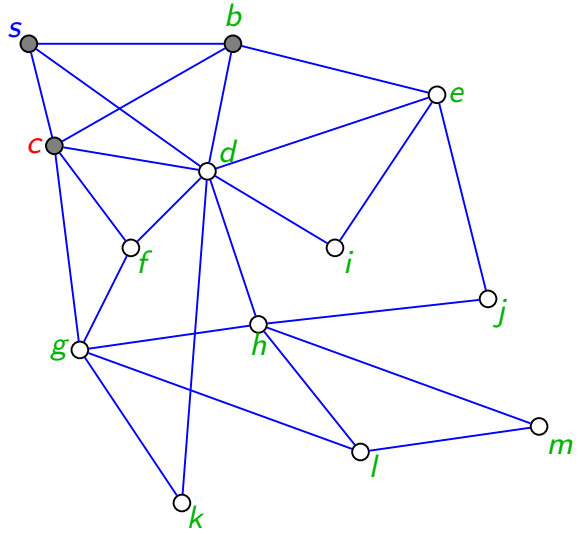
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

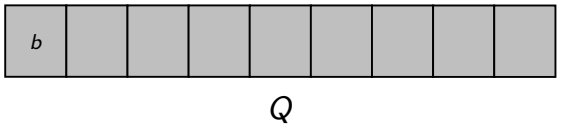
```
Begin
  for each vertex u ∈ V \ {s} {
    color[u] ← WHITE;
    d[u] ← ∞;     // d[u] = δ(s, u)
    π[u] ← NIL; }     // π[u] = predecessor of u

  color[s] ← GRAY;
  d[s] ← 0;
  π[s] ← NIL;
  Q ← ∅;     // Q denotes a queue
  ENQUEUE(Q, s);

  while (Q ≠ ∅) {
    u ← DEQUEUE(Q);
    for each v ∈ Adj[u] {
      if (color[v] = white) {
        color[v] ← GRAY;
        d[v] ← d[u] + 1;
        π[v] ← u;
        ENQUEUE(Q, v); } }
```

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b\}$
V_2: $\{\}$
V_3: $\{\}$

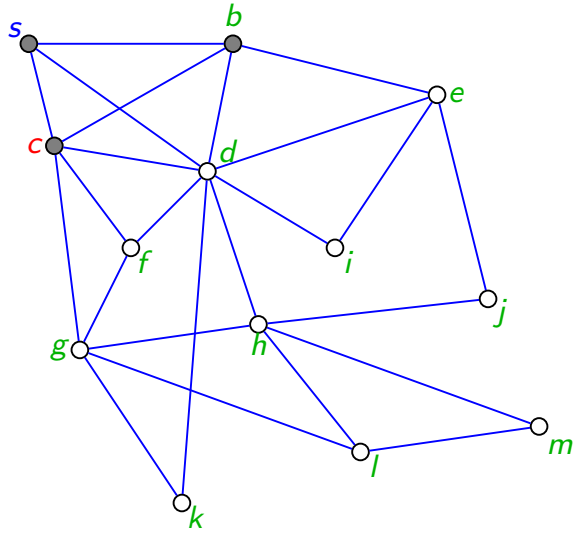
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

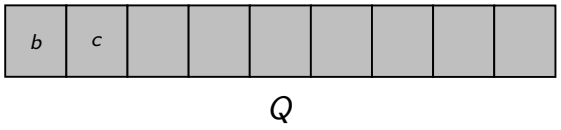
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c\}$
V_2: $\{\}$
V_3: $\{\}$

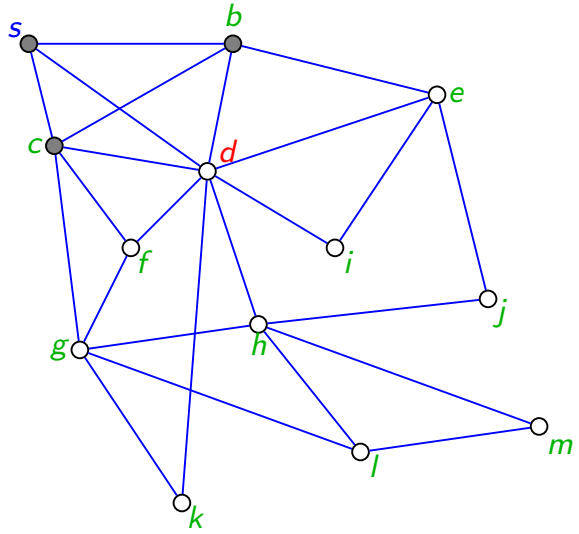
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

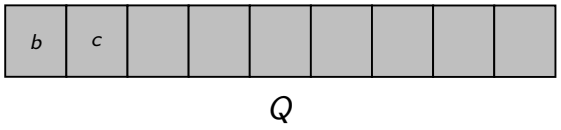
```
Begin
  for each vertex u ∈ V \ {s} {
    color[u] ← WHITE;
    d[u] ← ∞;     // d[u] = δ(s, u)
    π[u] ← NIL; }     // π[u] = predecessor of u

  color[s] ← GRAY;
  d[s] ← 0;
  π[s] ← NIL;
  Q ← ∅;     // Q denotes a queue
  ENQUEUE(Q, s);

  while (Q ≠ ∅) {
    u ← DEQUEUE(Q);
    for each v ∈ Adj[u] {
      if (color[v] = white) {
        color[v] ← GRAY;
        d[v] ← d[u] + 1;
        π[v] ← u;
        ENQUEUE(Q, v); } }
```

BFS(G, s)





Q

V_0: $\{s\}$
V_1: $\{b, c\}$
V_2: $\{\}$
V_3: $\{\}$

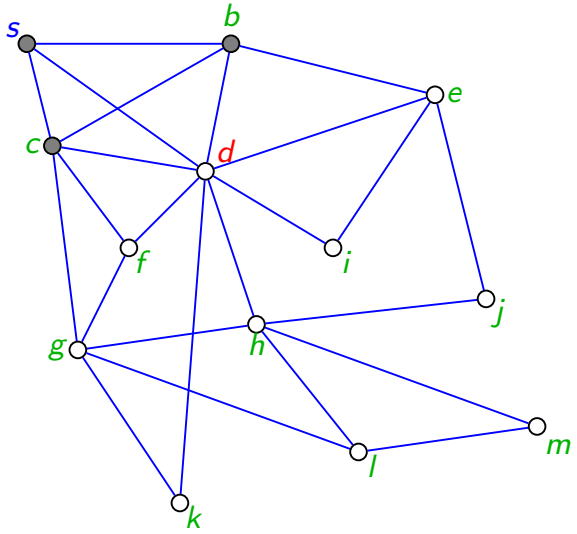
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

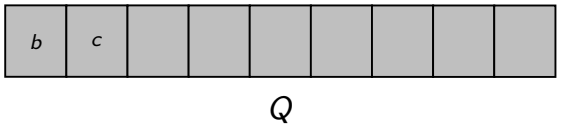
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c\}$
V_2: $\{\}$
V_3: $\{\}$

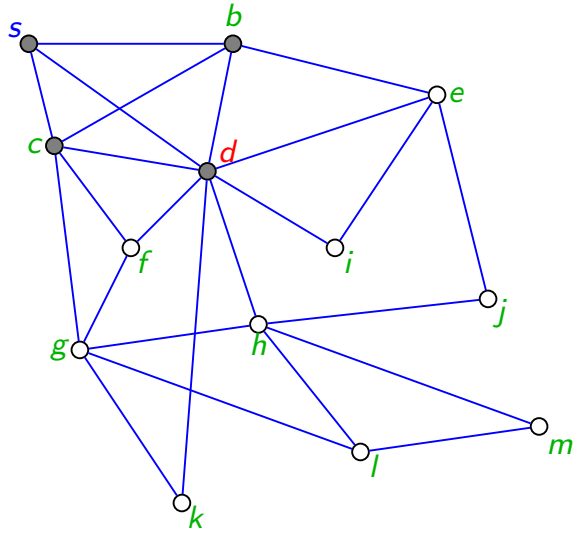
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

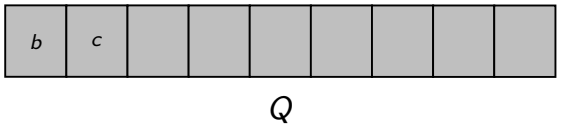
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while ($Q \neq \emptyset$) {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if ($color[v] = white$) {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c\}$
V_2: $\{\}$
V_3: $\{\}$

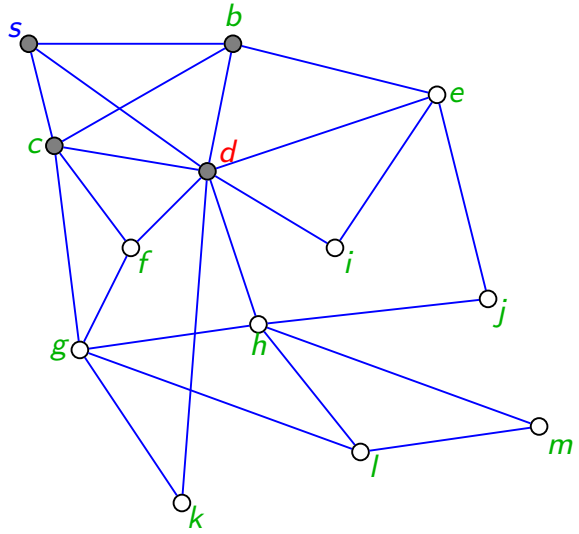
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

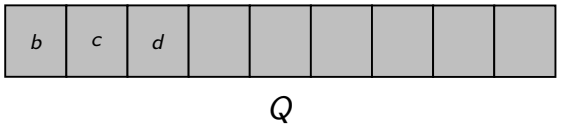
```
Begin
  for each vertex u ∈ V \ {s} {
    color[u] ← WHITE;
    d[u] ← ∞;     // d[u] = δ(s, u)
    π[u] ← NIL; }     // π[u] = predecessor of u

  color[s] ← GRAY;
  d[s] ← 0;
  π[s] ← NIL;
  Q ← ∅;     // Q denotes a queue
  ENQUEUE(Q, s);

  while (Q ≠ ∅) {
    u ← DEQUEUE(Q);
    for each v ∈ Adj[u] {
      if (color[v] = white) {
        color[v] ← GRAY;
        d[v] ← d[u] + 1;
        π[v] ← u;
        ENQUEUE(Q, v); } }
```

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{\}$
V_3: $\{\}$

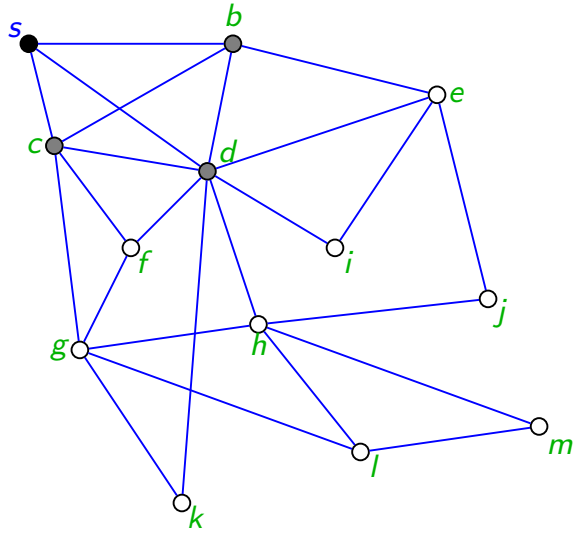
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

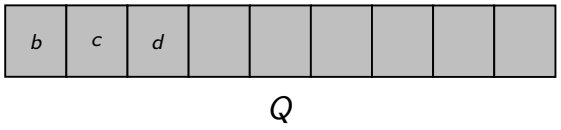
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{\}$
V_3: $\{\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

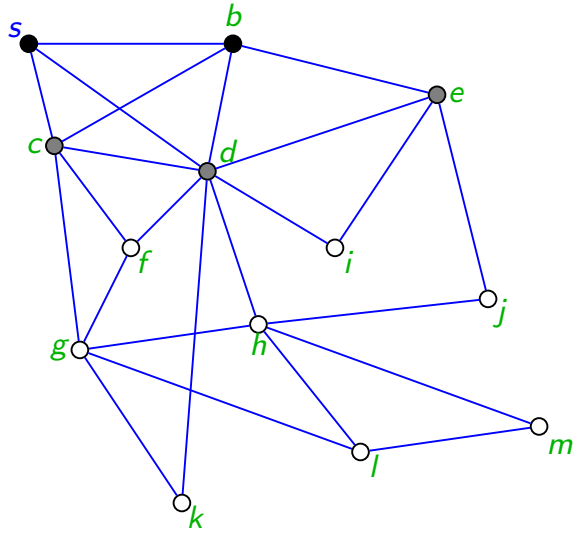
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

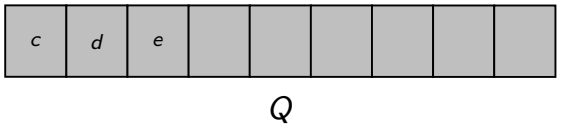
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e\}$
V_3: $\{\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

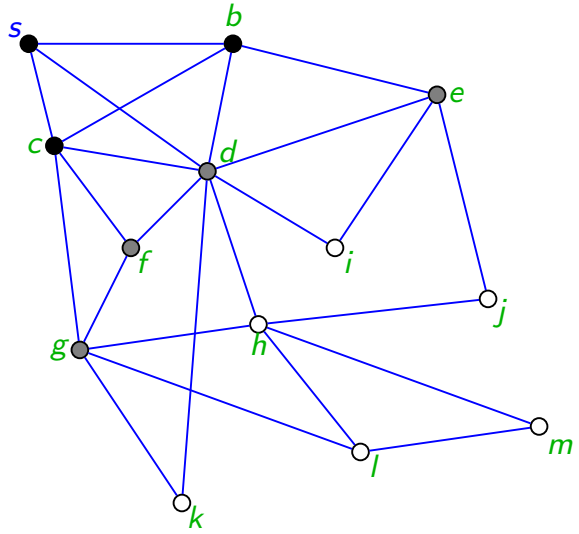
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

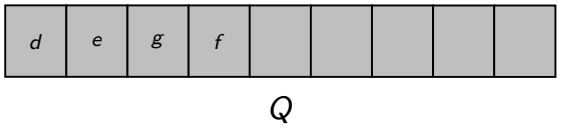
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f\}$
V_3: $\{\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

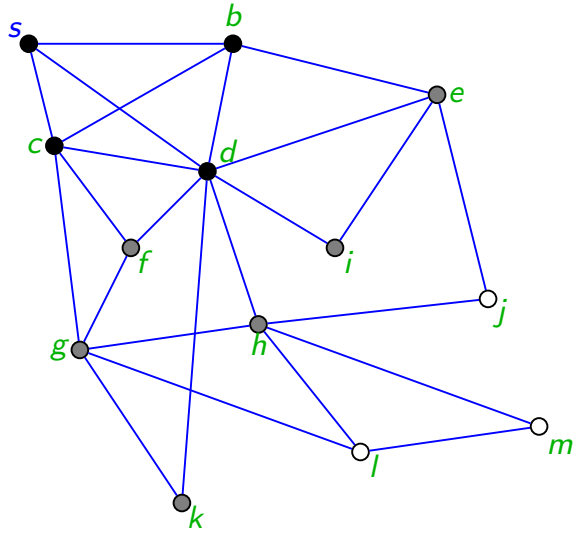
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

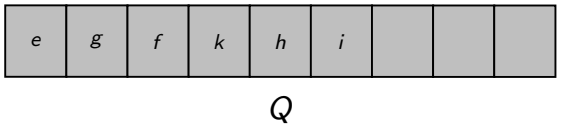
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{\}$

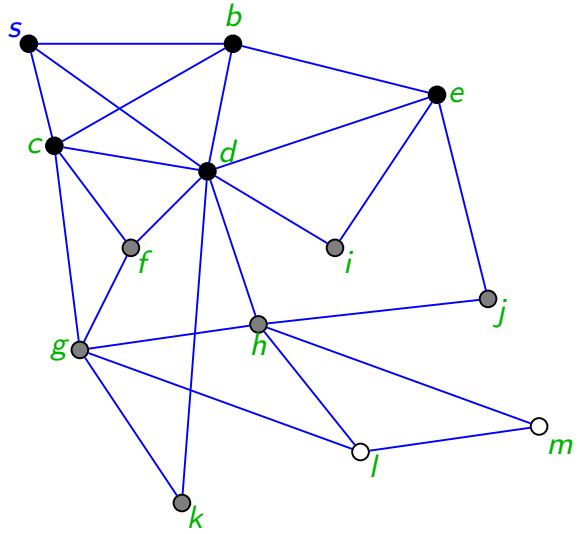
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

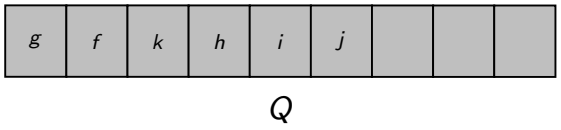
```
Begin
  for each vertex u ∈ V \ {s} {
    color[u] ← WHITE;
    d[u] ← ∞;     // d[u] = δ(s, u)
    π[u] ← NIL; }     // π[u] = predecessor of u

  color[s] ← GRAY;
  d[s] ← 0;
  π[s] ← NIL;
  Q ← ∅;     // Q denotes a queue
  ENQUEUE(Q, s);

  while (Q ≠ ∅) {
    u ← DEQUEUE(Q);
    for each v ∈ Adj[u] {
      if (color[v] = white) {
        color[v] ← GRAY;
        d[v] ← d[u] + 1;
        π[v] ← u;
        ENQUEUE(Q, v); } }
    color[u] ← BLACK; }
End
```

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j\}$

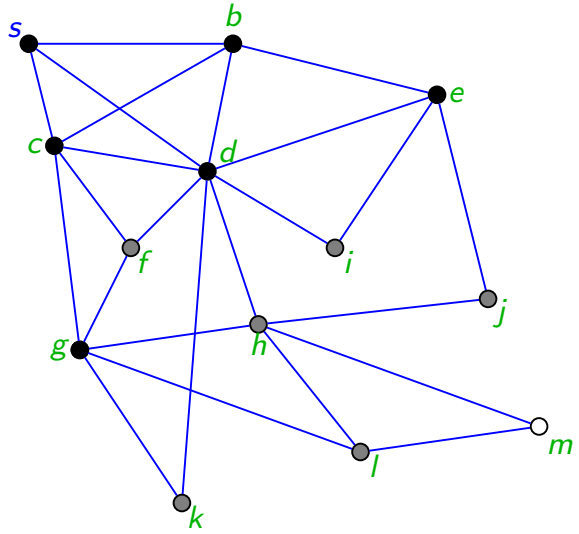
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

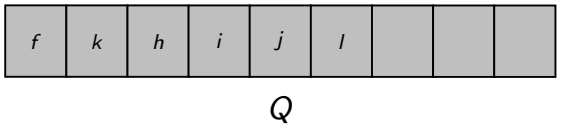
```
Begin
  for each vertex u ∈ V \ {s} {
    color[u] ← WHITE;
    d[u] ← ∞;     // d[u] = δ(s, u)
    π[u] ← NIL; }     // π[u] = predecessor of u

  color[s] ← GRAY;
  d[s] ← 0;
  π[s] ← NIL;
  Q ← ∅;     // Q denotes a queue
  ENQUEUE(Q, s);

  while (Q ≠ ∅) {
    u ← DEQUEUE(Q);
    for each v ∈ Adj[u] {
      if (color[v] = white) {
        color[v] ← GRAY;
        d[v] ← d[u] + 1;
        π[v] ← u;
        ENQUEUE(Q, v); } }
    color[u] ← BLACK; }
End
```

BFS(G, s)





Q

V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l\}$

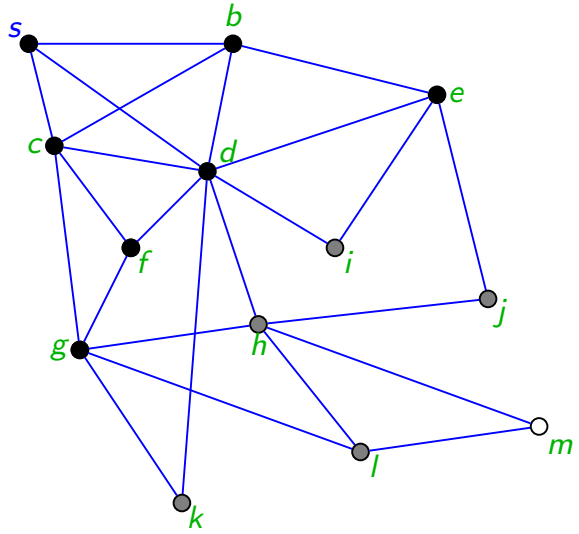
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

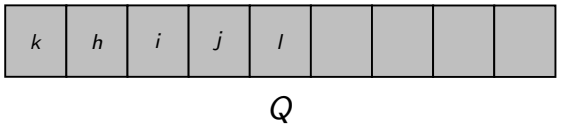
```
Begin
  for each vertex u ∈ V \ {s} {
    color[u] ← WHITE;
    d[u] ← ∞;     // d[u] = δ(s, u)
    π[u] ← NIL; }     // π[u] = predecessor of u

  color[s] ← GRAY;
  d[s] ← 0;
  π[s] ← NIL;
  Q ← ∅;     // Q denotes a queue
  ENQUEUE(Q, s);

  while (Q ≠ ∅) {
    u ← DEQUEUE(Q);
    for each v ∈ Adj[u] {
      if (color[v] = white) {
        color[v] ← GRAY;
        d[v] ← d[u] + 1;
        π[v] ← u;
        ENQUEUE(Q, v); } }
    color[u] ← BLACK; }
End
```

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

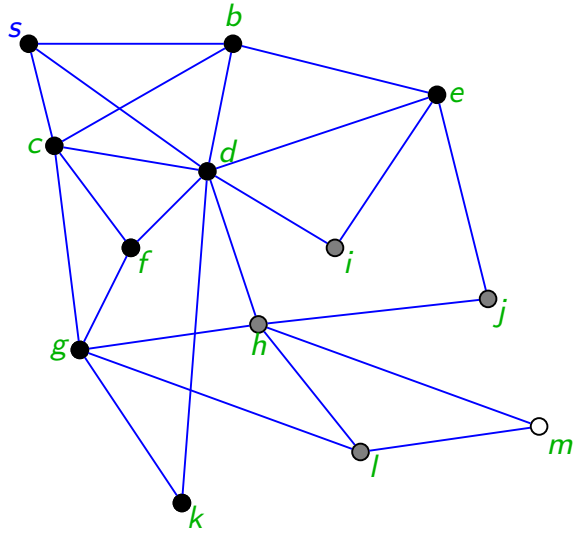
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

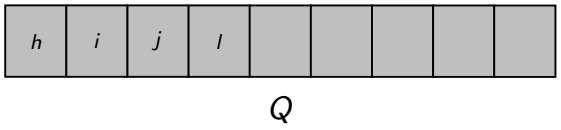
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

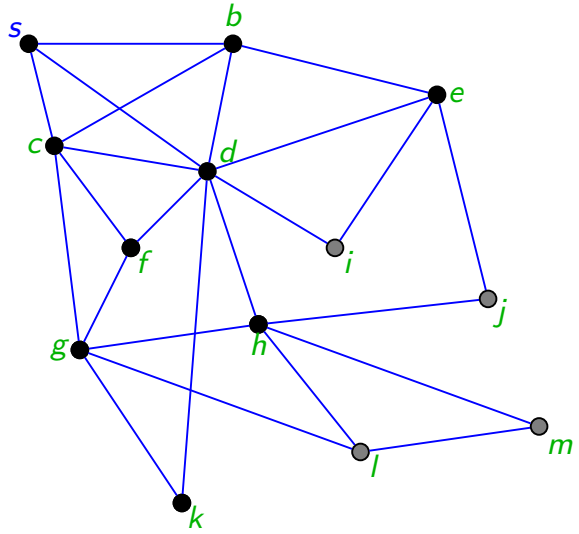
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

BFS(G, s)



V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l, m\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

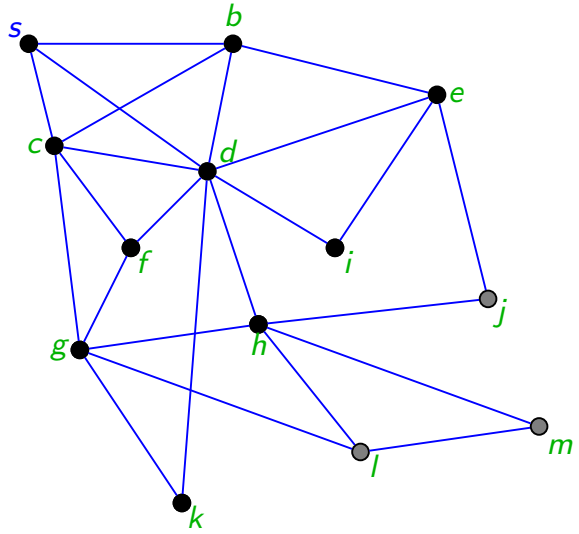
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

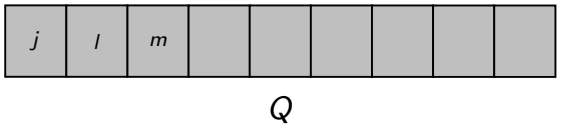
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l, m\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

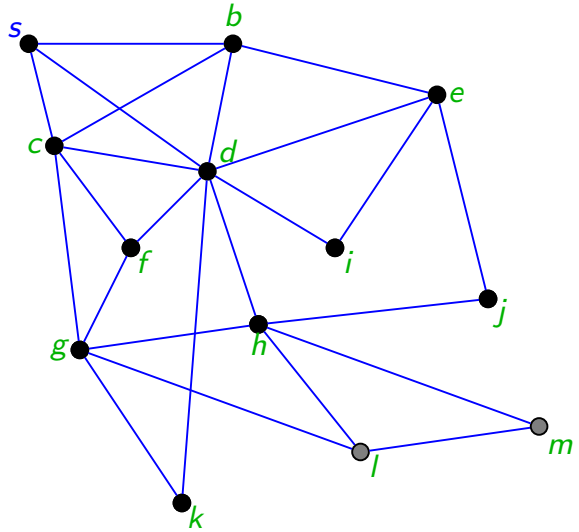
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

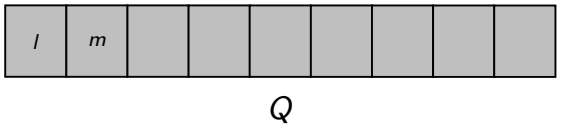
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l, m\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

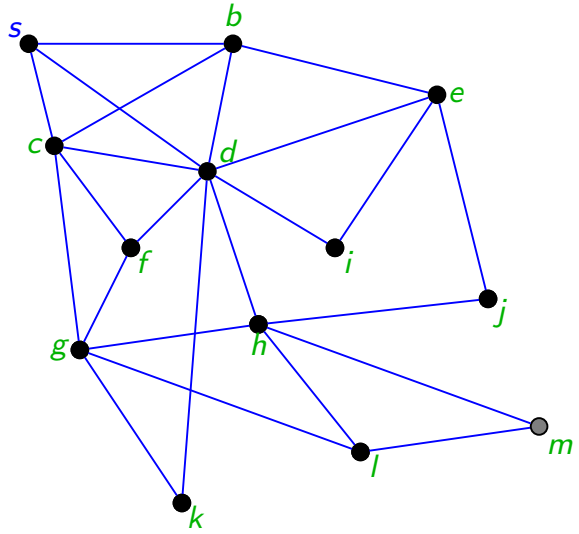
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

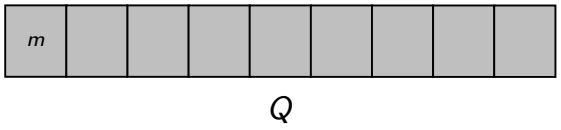
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l, m\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

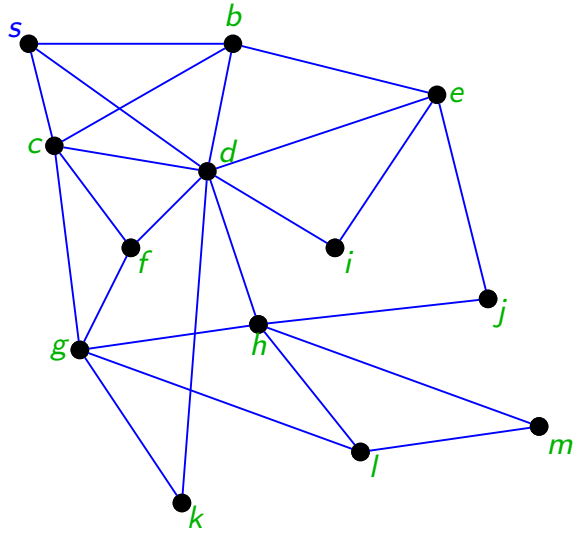
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

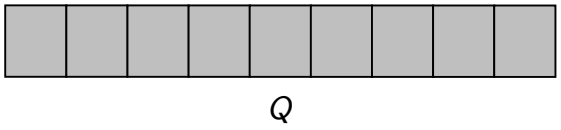
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

BFS(G, s)





V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l, m\}$

I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$; // $d[u] = \delta(s, u)$
    $\pi[u] \leftarrow$ NIL; } // $\pi[u]$ = predecessor of u

  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  ENQUEUE(Q, s);

  while $(Q \neq \emptyset)$ {
    $u \leftarrow$ DEQUEUE(Q);
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow$ GRAY;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        ENQUEUE(Q, v); } }
    $color[u] \leftarrow$ BLACK; }

End

Books and Other Materials Consulted

1. *Introduction to Algorithms* by Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein.

2. Taken from Prof. Surendar Baswana (CSE, IIT Kanpur) lecture slides.

Thank You for your kind attention!

Questions!!