BFS (Cont.) and Bipartite Graphs

Subhabrata Samajder

IIIT, Delhi

Winter Semester,
17th May, 2023

BFS (Cont.)

Recap

- It is one of the simplest algorithms for searching a graph.
- Many important graph algorithms use similar ideas. Like,
  - Prim's minimum-spanning-tree algorithm.
  - Dijkstra's single-source shortest-paths algorithm.

Recap

- It is one of the simplest algorithms for searching a graph.
- Many important graph algorithms use similar ideas. Like,
  - Prim's minimum-spanning-tree algorithm.
  - Dijkstra's single-source shortest-paths algorithm.
- For a source vertex s, the algorithm systematically explores the edges of G to "discover" every vertex that is reachable from s.

Recap

- It is one of the simplest algorithms for searching a graph.
- Many important graph algorithms use similar ideas. Like,
  - Prim's minimum-spanning-tree algorithm.
  - Dijkstra's single-source shortest-paths algorithm.
- For a source vertex s, the algorithm systematically explores the edges of G to "discover" every vertex that is reachable from s.
- Computes the shortest distance from s to each reachable vertex.

Recap

- It is one of the simplest algorithms for searching a graph.
- Many important graph algorithms use similar ideas. Like,
  - Prim's minimum-spanning-tree algorithm.
  - Dijkstra's single-source shortest-paths algorithm.
- For a source vertex s, the algorithm systematically explores the edges of G to "discover" every vertex that is reachable from s.
- Computes the shortest distance from s to each reachable vertex.
- Produces a "breadth-first tree" with root s and containing all reachable vertices.

Recap

- It is one of the simplest algorithms for searching a graph.
- Many important graph algorithms use similar ideas. Like,
  - Prim's minimum-spanning-tree algorithm.
  - Dijkstra's single-source shortest-paths algorithm.
- For a source vertex s, the algorithm systematically explores the edges of G to "discover" every vertex that is reachable from s.
- Computes the shortest distance from s to each reachable vertex.
- Produces a "breadth-first tree" with root s and containing all reachable vertices.
- Works on both directed and undirected graphs.

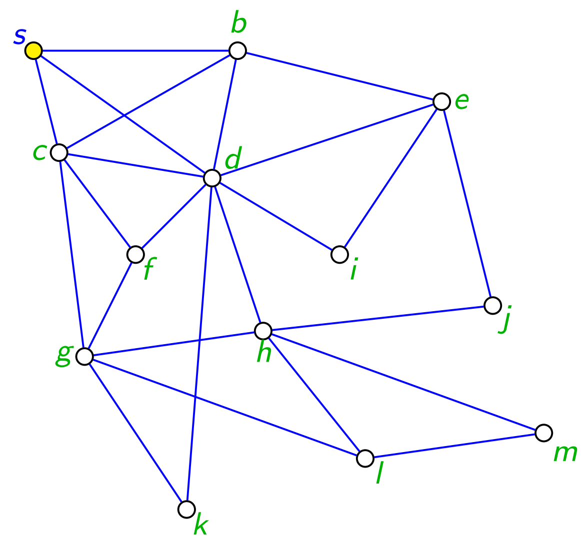
Recap

- It is so named because it expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier.
- That is, the algorithm discovers all vertices at distance k from a source s before discovering any vertices at distance $k+1$.

Recap

- It is so named because it expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier.
- That is, the algorithm discovers all vertices at distance k from a source s before discovering any vertices at distance $k+1$.
- Keep track of progress by coloring each vertex white, gray, or black.
- All vertices start out white.
- May later become gray and then black.
- A vertex is *discovered*, the first time it is encountered during the search, at which time it becomes non-white.
- Distinguishes between gray and black vertices to ensure that the search proceeds in a breadth-first manner.

$\mathrm{BFS}(G, s)$: Recap

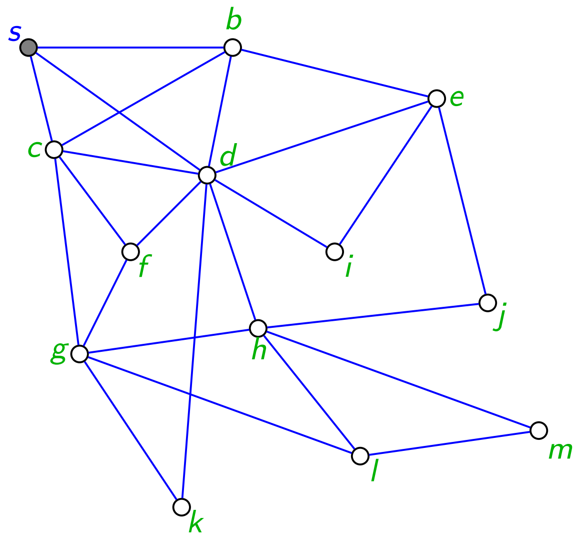


I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow \mathrm{WHITE};$
    $d[u] \leftarrow \infty;$
    $\pi[u] \leftarrow \mathrm{NIL};$ }

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

BFS(G, s): Recap

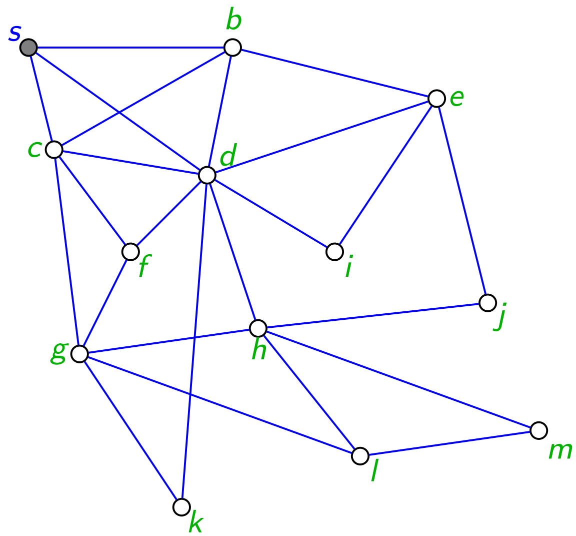


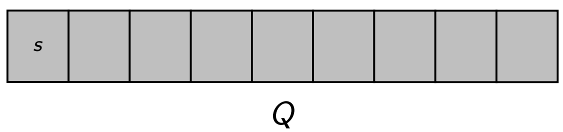
V_0: $\{\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow \text{WHITE};$
    $d[u] \leftarrow \infty;$
    $\pi[u] \leftarrow \text{NIL};$ }
  $color[s] \leftarrow \text{GRAY};$

BFS(G, s): Recap





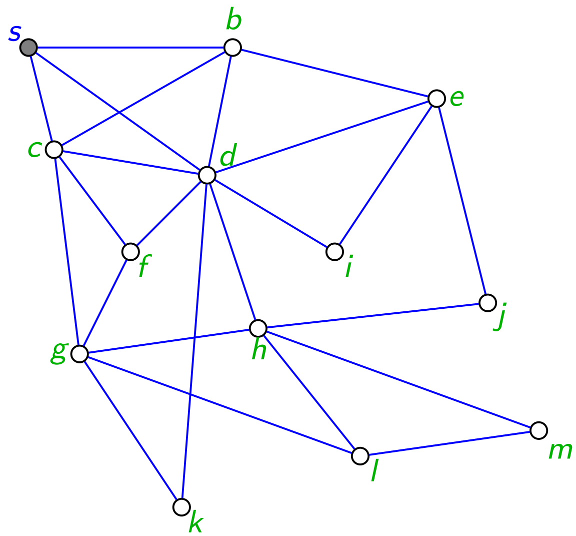
V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow \text{WHITE}$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \text{NIL}$; }

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

BFS(G, s): Recap



Q

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

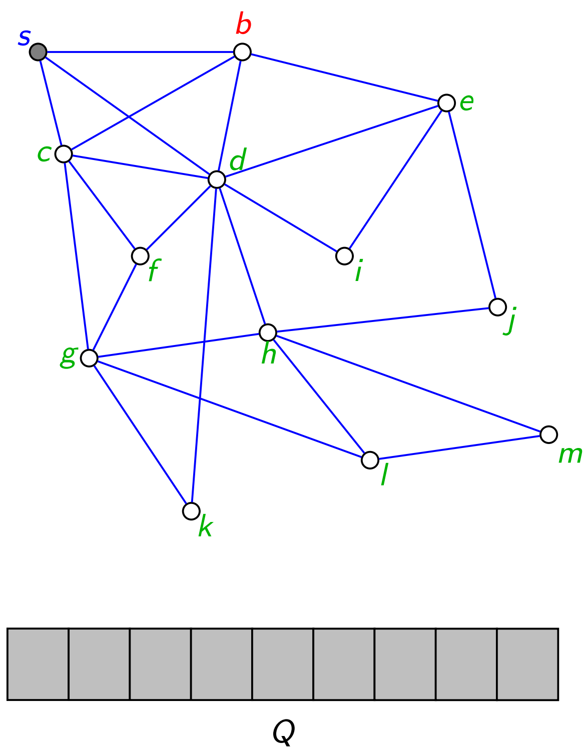
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

  $color[s] \leftarrow$ 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): Recap

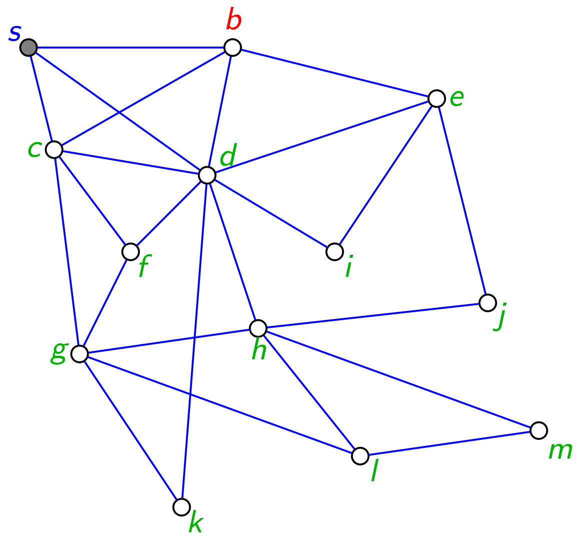


V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

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;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ 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): Recap



Q

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

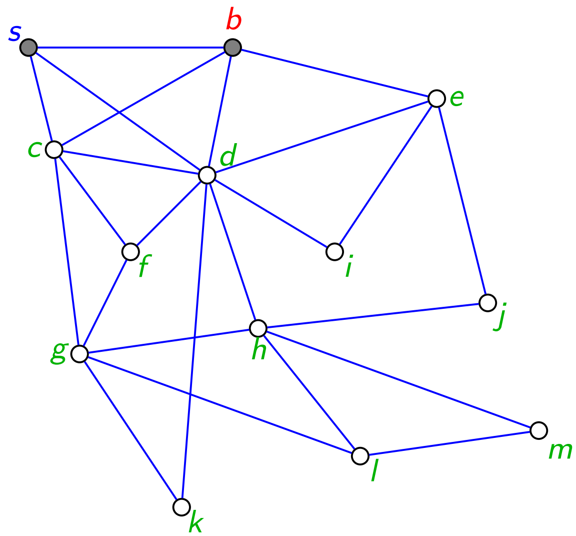
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

  $color[s] \leftarrow$ 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): Recap



Q

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

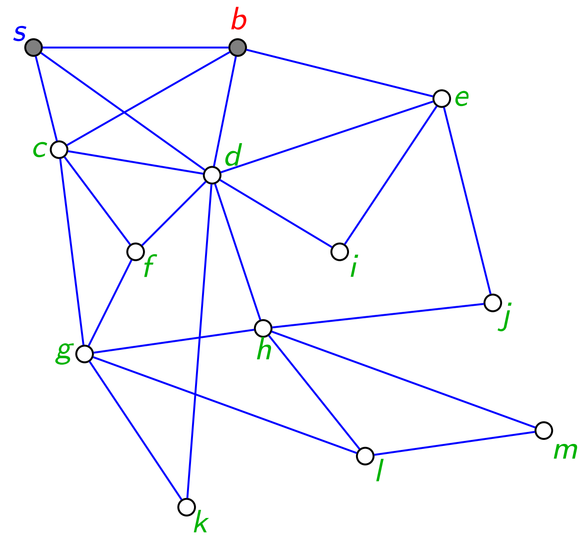
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

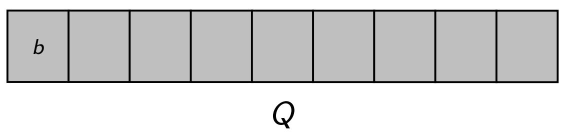
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

  $color[s] \leftarrow$ 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;

BFS(G, s): Recap





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

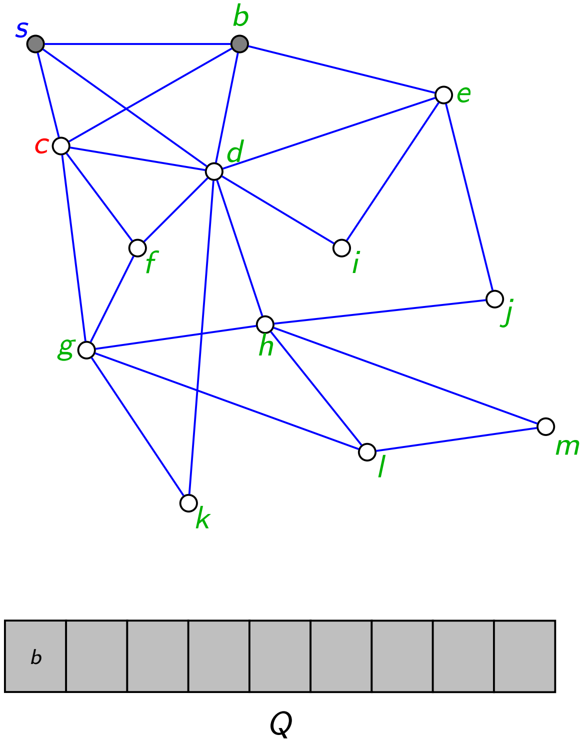
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow \text{WHITE}$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \text{NIL}$; }

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

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

BFS(G, s): Recap



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

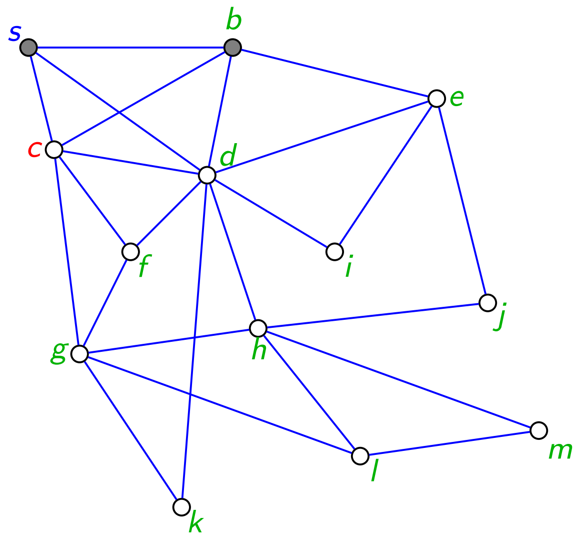
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

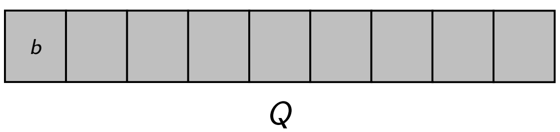
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

  $color[s] \leftarrow$ 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): Recap





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

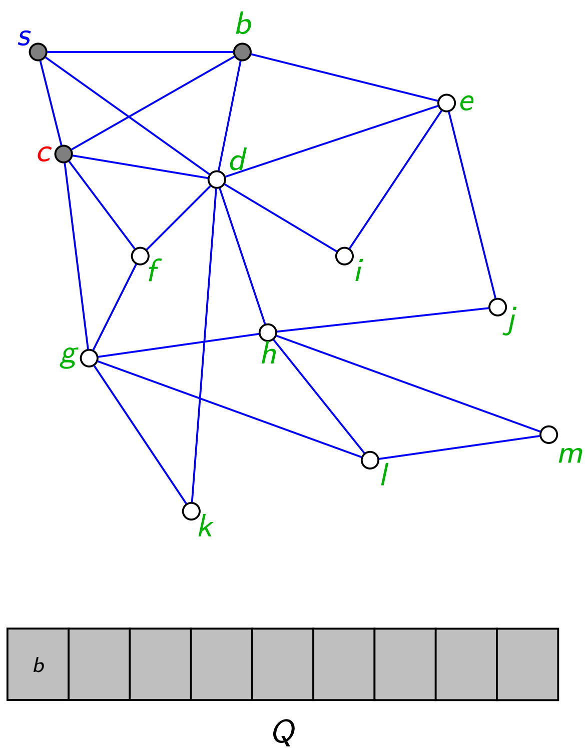
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow \mathrm{WHITE}$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }

  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$; // Q denotes a queue
  $\textsc{Enqueue}(Q, s)$;

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

BFS(G, s): Recap

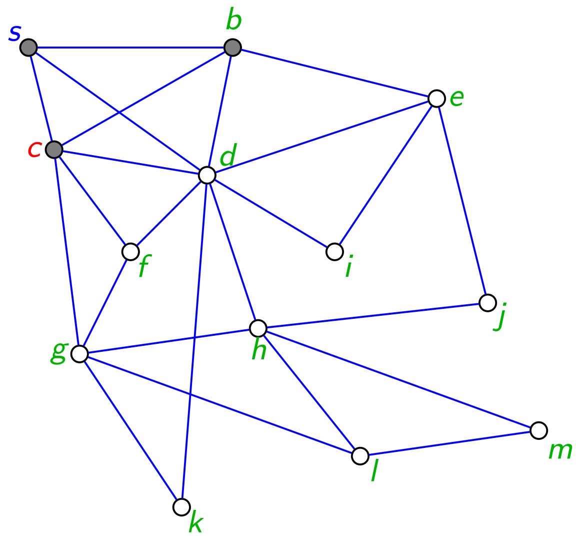


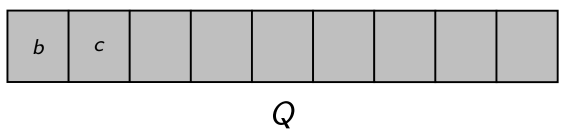
V_0: $\{s\}$
V_1: $\{b\}$
V_2: $\{\}$
V_3: $\{\}$

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;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ 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): Recap





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

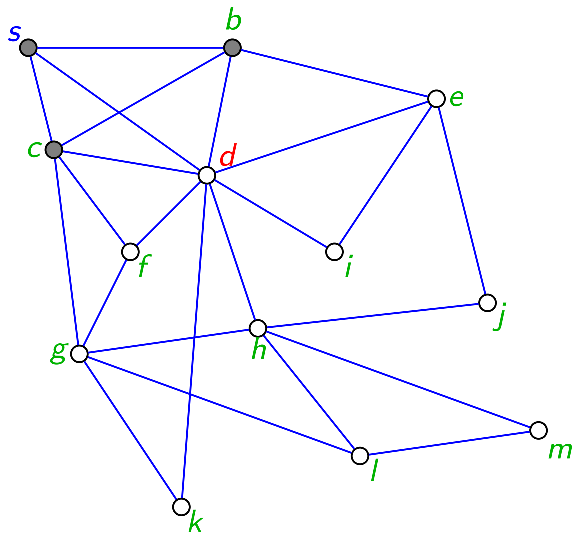
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

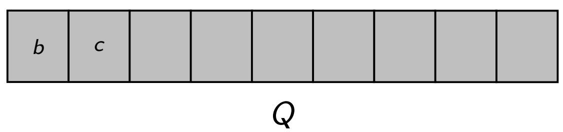
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

  $color[s] \leftarrow$ 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): Recap





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

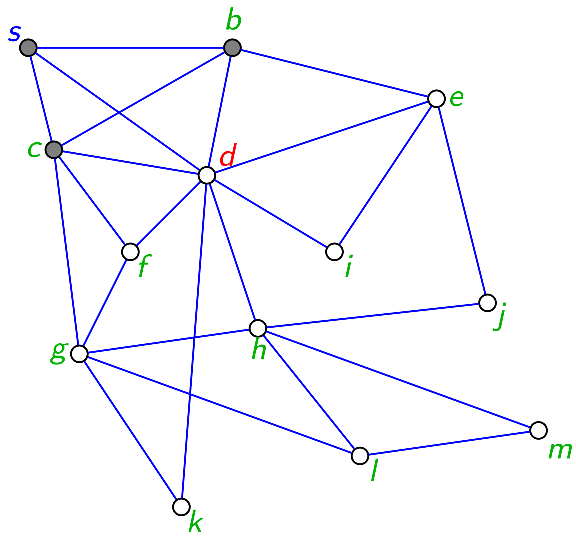
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

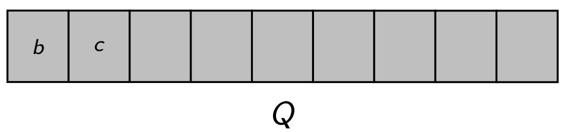
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

  $color[s] \leftarrow$ 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): Recap





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

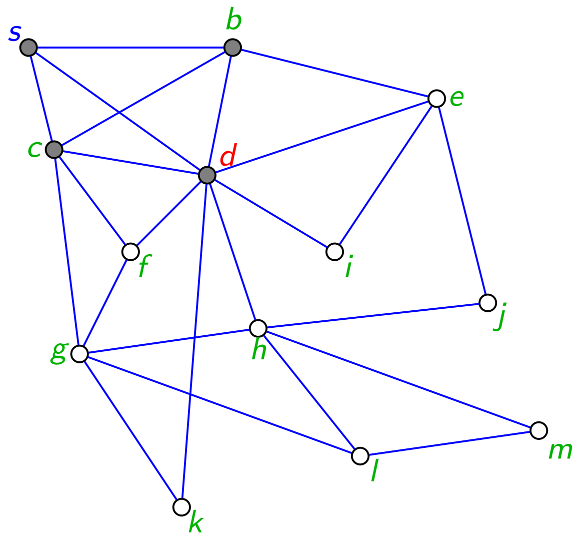
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow \mathrm{WHITE}$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }

  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  $\textsc{Enqueue}(Q, s)$;

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

BFS(G, s): Recap



Q

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

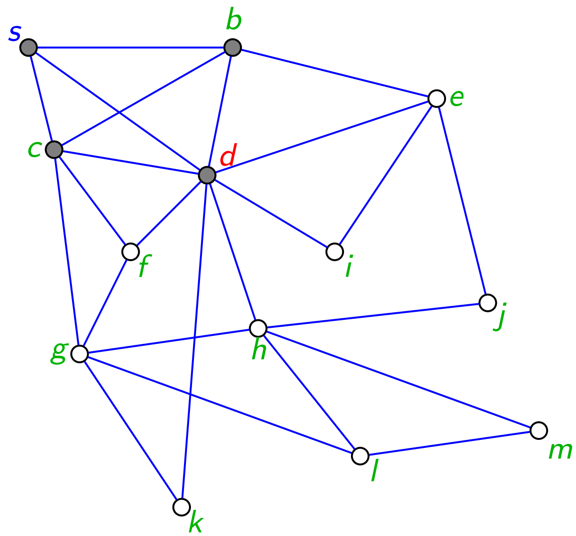
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

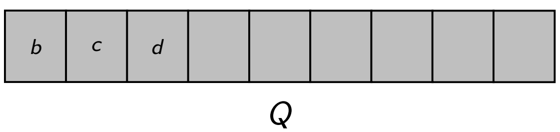
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

  $color[s] \leftarrow$ 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): Recap





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

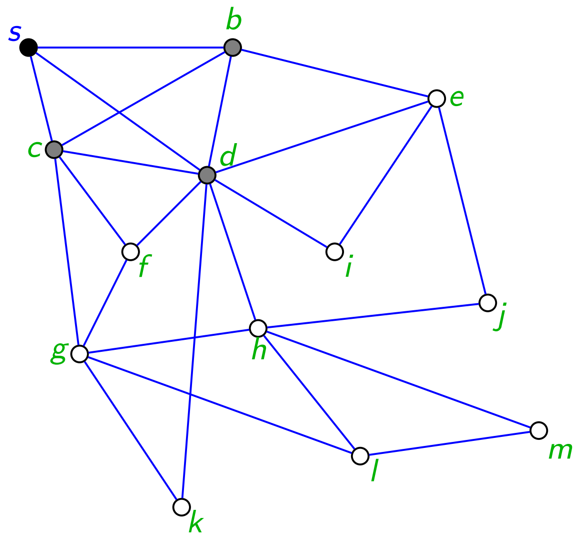
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

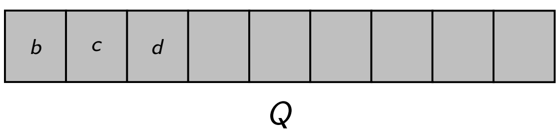
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

  $color[s] \leftarrow$ 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): Recap





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.

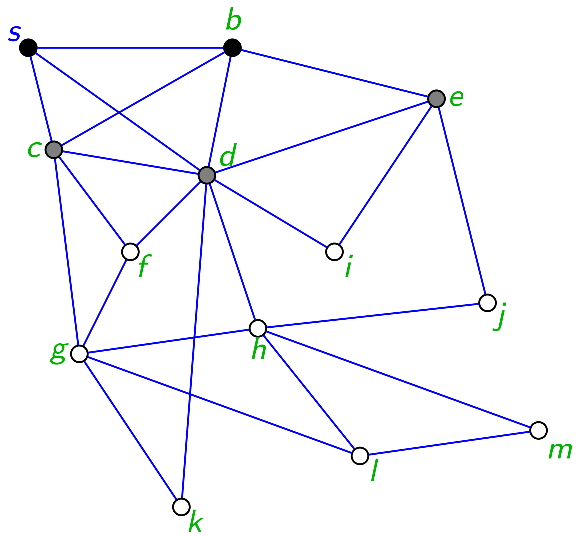
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

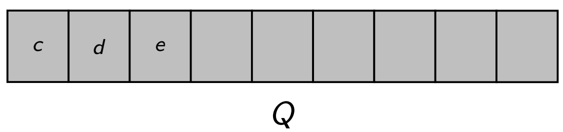
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  ENQUEUE(Q, s);

  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): Recap



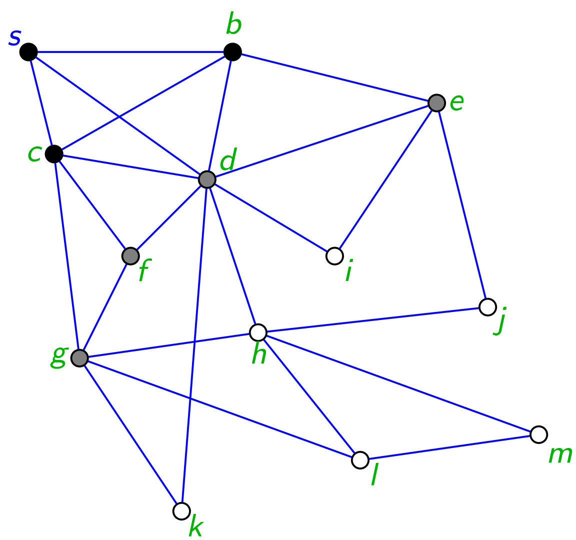


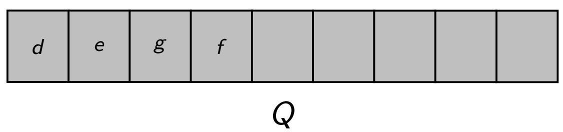
V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e\}$
V_3: $\{\}$

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;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ 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): Recap



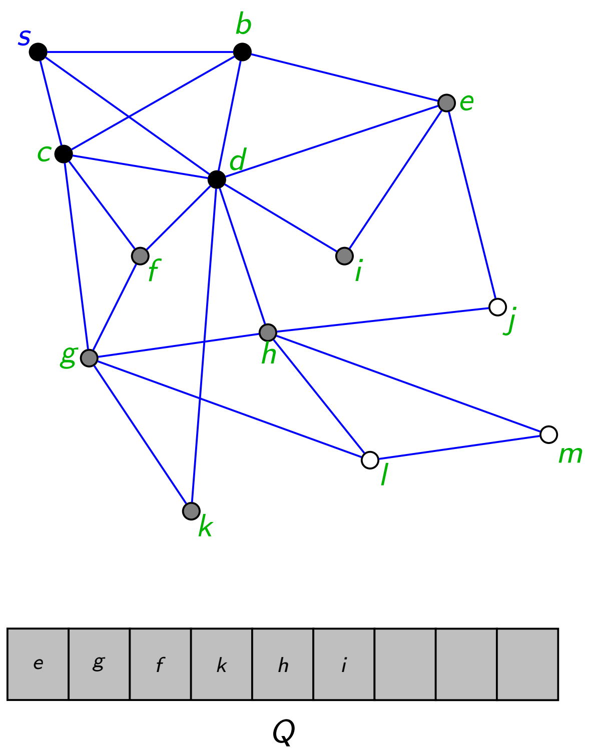


V_0: $\{s\}$
V_1: $\{b,\ c,\ d\}$
V_2: $\{e,\ g,\ f\}$
V_3: $\{\}$

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;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ 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): Recap



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.

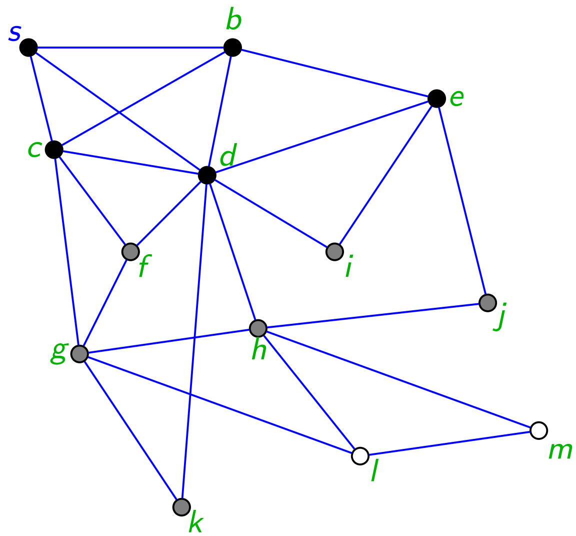
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

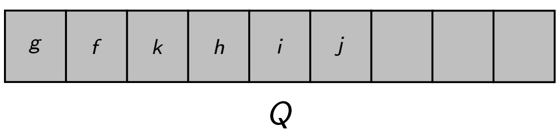
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  ENQUEUE(Q, s);

  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): Recap





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.

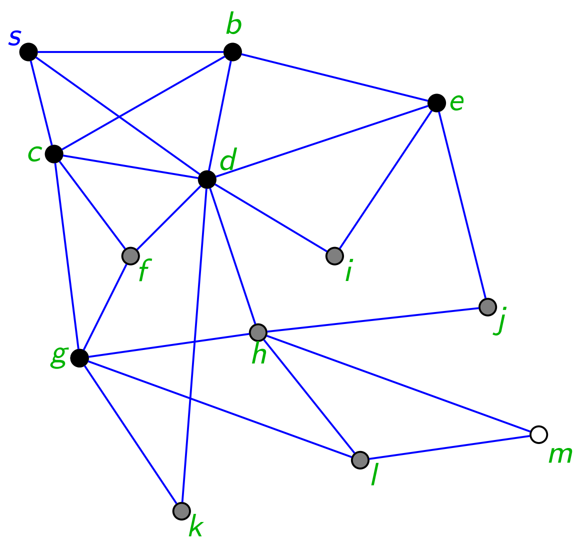
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow \mathrm{WHITE}$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }

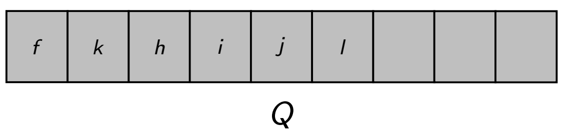
  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  $\textsc{Enqueue}(Q, s)$;

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

End

BFS(G, s): Recap



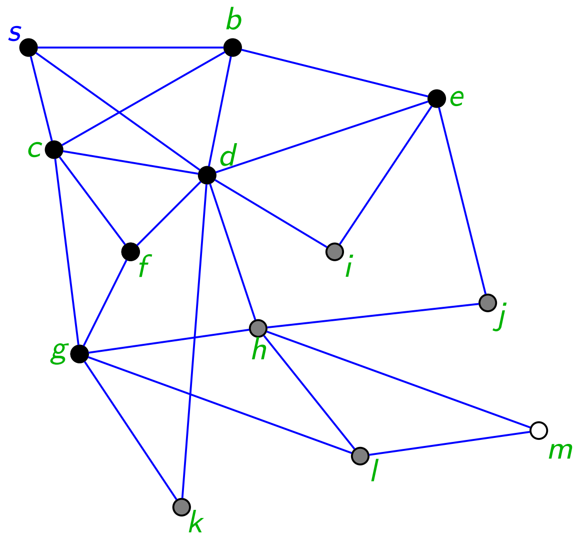


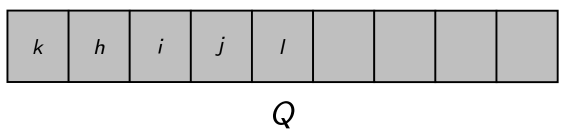
V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l\}$

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;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ 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): Recap





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.

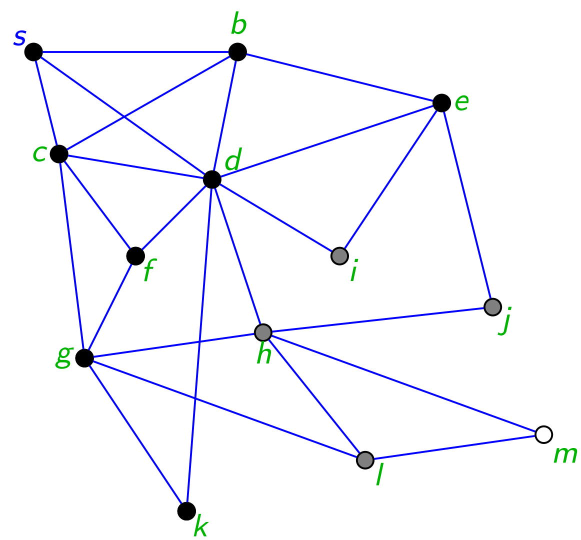
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

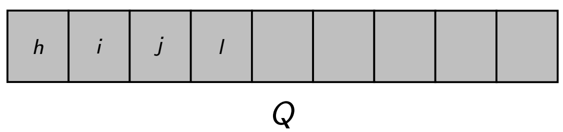
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  ENQUEUE(Q, s);

  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): Recap



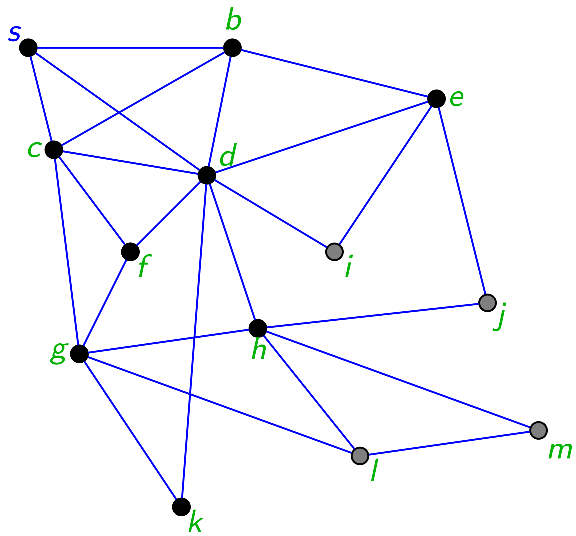


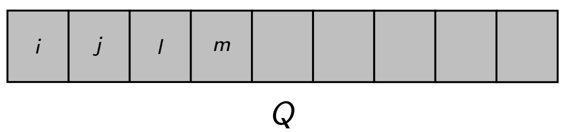
V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l\}$

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;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ 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): Recap





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.

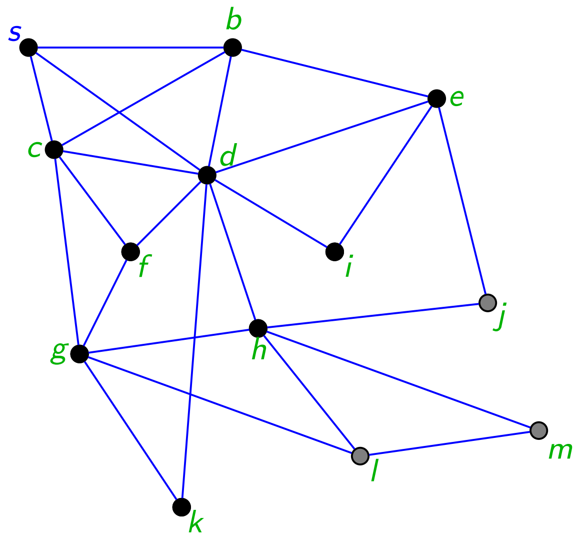
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

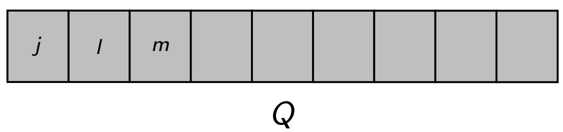
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  ENQUEUE(Q, s);

  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): Recap





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.

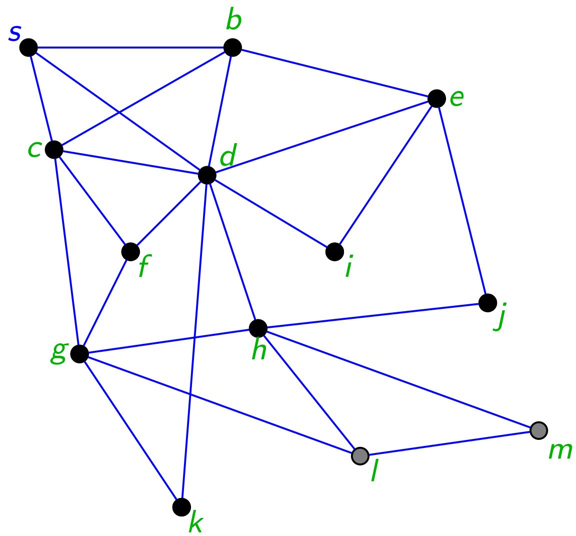
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

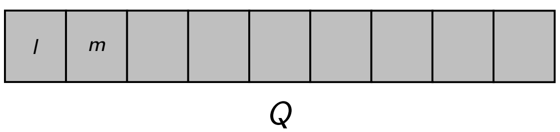
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  ENQUEUE(Q, s);

  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

$\text{BFS}(G, s)$: Recap





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.

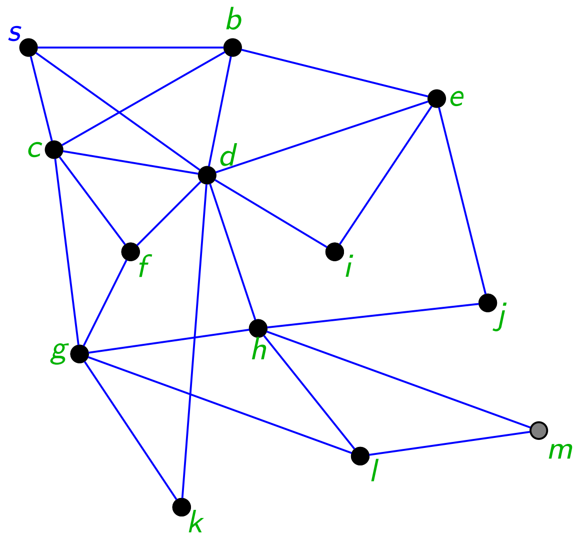
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow \text{WHITE}$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \text{NIL}$; }

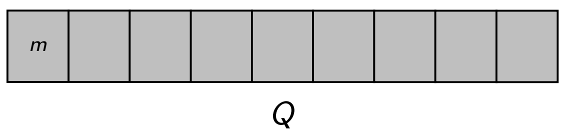
  $color[s] \leftarrow \text{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \text{NIL}$;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  $\text{ENQUEUE}(Q, s)$;

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

End

BFS(G, s): Recap





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.

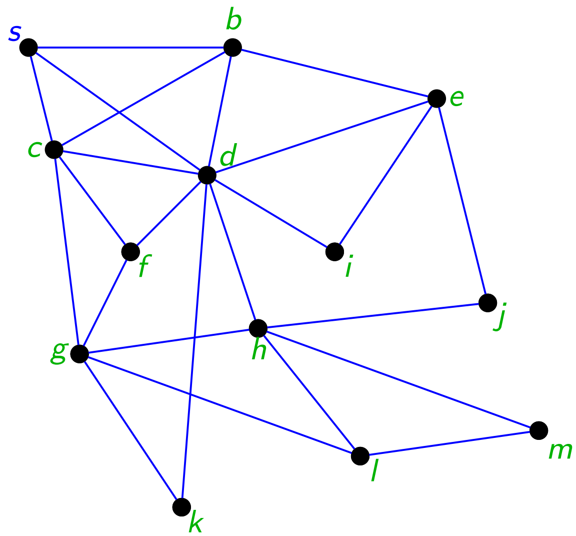
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow$ WHITE;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

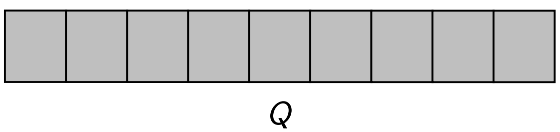
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;   // Q denotes a queue
  ENQUEUE(Q, s);

  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): Recap





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;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ 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

A Loop Invariant

"At the beginning of the **while** loop, the queue Q consists of the set of gray vertices of G."

A Loop Invariant

"At the beginning of the **while** loop, the queue Q consists of the set of gray vertices of G."

- Prior to the first iteration, the only gray vertex, and the only vertex in Q, is the source vertex s.

A Loop Invariant

"At the beginning of the **while** loop, the queue Q consists of the set of gray vertices of G."

- Prior to the first iteration, the only gray vertex, and the only vertex in Q, is the source vertex s.
- For each vertex $v \in Adj[u]$, if $color[v]$ is white, then it has not yet been discovered,
  - It is first grayed, and
  - its distance $d[v]$ is set to $d[u] + 1$.
  - Then, u is recorded as its parent.
  - Finally, it is placed at the tail of the queue Q.
- When all the vertices on u's adjacency list have been examined, u is blackened.

A Loop Invariant

"At the beginning of the **while** loop, the queue Q consists of the set of gray vertices of G."

- Prior to the first iteration, the only gray vertex, and the only vertex in Q, is the source vertex s.
- For each vertex $v \in Adj[u]$, if $color[v]$ is white, then it has not yet been discovered,
  - It is first grayed, and
  - its distance $d[v]$ is set to $d[u] + 1$.
  - Then, u is recorded as its parent.
  - Finally, it is placed at the tail of the queue Q.
- When all the vertices on u's adjacency list have been examined, u is blackened.
- Thus whenever a vertex is painted gray it is also ENQUEUED, and whenever a vertex is DEQUEUED it is also painted black..

Observations About $\mathrm{BFS}(G, s)$

- The result depends upon the order in which the neighbors of a given vertex are visited.
- But, the distances d is not affected by it.
- Any vertex v enters the queue at most once.
- Before entering the queue, distance $d[v]$ is updated.
- When a vertex v is DEQUEUED, v processes all its unvisited neighbors by
  - computing their distances and
  - then enqueuing them.
- A vertex v in the queue is surely removed from the queue during the algorithm.

Cost Analysis

- **Initialization phase:** $\mathcal{O}(|V|)$
- **Note:** After initialization, no vertex is ever whitened.
- Each vertex is ENQUEUED only if it is white $\Rightarrow$ each vertex is ENQUEUED at most once and hence DEQUEUED at most once.
- ENQUEUING and DEQUEUING takes $\mathcal{O}(1)$ time.
- So the total time devoted to queue operations is $\mathcal{O}(|V|)$.
- Adjacency list of each vertex is scanned only when the vertex is DEQUEUED $\Rightarrow$ each adjacency list is scanned at most once.
- Recall the sum of the lengths of all the adjacency lists is $\Theta(|E|)$.
- $\therefore$ total time spent in scanning adjacency lists is $\mathcal{O}(|E|)$.
- **Complexity:** $\mathcal{O}(|V| + |E|)$.
- $\therefore$ BFS runs in time linear in the size of the adjacency-list representation of G.

Correctness of BFS

Question: What do we mean by correctness of $\mathrm{BFS}(G, s)$?

Answer:

- All vertices reachable from s get visited.
- Vertices are visited in non-decreasing order of distance from s.
- At the end of the algorithm, $d[v] = \delta(s, v)$ for all $v \in V \setminus \{s\}$.

Lemma 1

Lemma

Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$,

$$\delta(s, v) \leq \delta(s, u) + 1.$$

Lemma 2

- We want to show that BFS properly computes $d[v] = \delta(s, v)$ for each vertex $v \in V$.
- But first we show that $\delta(s, v) \leq d[v]$.

Lemma 2

- We want to show that BFS properly computes $d[v] = \delta(s, v)$ for each vertex $v \in V$.
- But first we show that $\delta(s, v) \leq d[v]$.

Lemma

Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $d[v]$ computed by BFS satisfies $d[v] \geq \delta(s, v)$.

Lemma 3

- To prove that $d[v] = \delta(s, v)$, we must first show more precisely how the queue Q operates during the course of BFS.
- The next lemma shows that at all times, there are at most two distinct d values in the queue.

Lemma 3

- To prove that $d[v] = \delta(s, v)$, we must first show more precisely how the queue Q operates during the course of BFS.
- The next lemma shows that at all times, there are at most two distinct d values in the queue.

Lemma

Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $\langle v_1, v_2, \ldots, v_r \rangle$, where v_1 is the head of Q and v_r is the tail. Then, $d[v_r] \leq d[v_1] + 1$ and $d[v_i] \leq d[v_{i+1}]$ for $i = 1, 2, \ldots, r-1$.

Corollary 1

The following corollary shows that the d values at the time that vertices are ENQUEUED are monotonically increasing over time.

Corollary

Suppose that vertices v_i and v_j are ENQUEUED *during the execution of BFS, and that v_i is enqueued before v_j. Then $d[v_i] \leq d[v_j]$ at the time that v_j is* ENQUEUED.

Correctness of BFS

Theorem (Correctness of breadth-first search)

Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then, during its execution,

- *BFS discovers every vertex $v \in V$ that is reachable from the source s, and*
- *upon termination, $d[v] = \delta(s, v)$ for all $v \in V$.*
- *Moreover, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $\pi[v]$ followed by the edge $(\pi[v], v)$.*

BFS Tree

Breadth-first Search Tree

- BFS constructs a breadth-first tree.
- Initially containing only its root, which is the source vertex s.
- Recall that whenever a white vertex v is discovered in the course of scanning the adjacency list of an already discovered vertex u, the vertex v and the edge (u, v) are added to the tree.
- We say that u is the predecessor or parent of v in the breadth-first tree.
- Since a vertex is discovered at most once, it has at most one parent.
- Ancestor and descendant relationships in the breadth-first tree are defined relative to the root s as usual:
    - if u is on a path in the tree from the root s to vertex v, then u is an ancestor of v and v is a descendant of u.

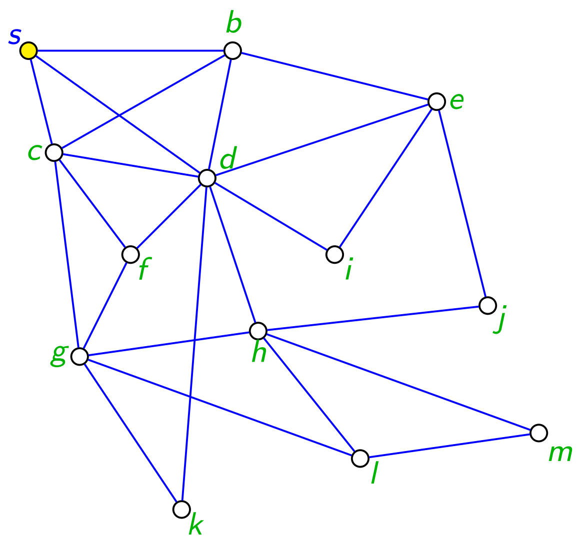
Breadth-first Search Tree (Cont.)

- The tree is represented by the π field in each vertex.
- **Formally:** For a graph $G = (V, E)$ with source s, define the predecessor subgraph of G as $G_\pi = (V_\pi, E_\pi)$, where

$$\begin{aligned} V_\pi &= \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\} \text{ and} \\ E_\pi &= \{(\pi[v], v) : v \in V_\pi \setminus \{s\}\}. \end{aligned}$$

- The predecessor subgraph G_π is a breadth-first tree.
- V_π consists of the vertices reachable from s.
- For all $v \in V_\pi$, there is a unique simple path from s to v in G_π that is also a shortest path from s to v in G.
- It is in fact a tree, since it is connected and $|E_\pi| = |V_\pi| - 1$. The edges in E_π are called tree edges.

BFS(G, s)

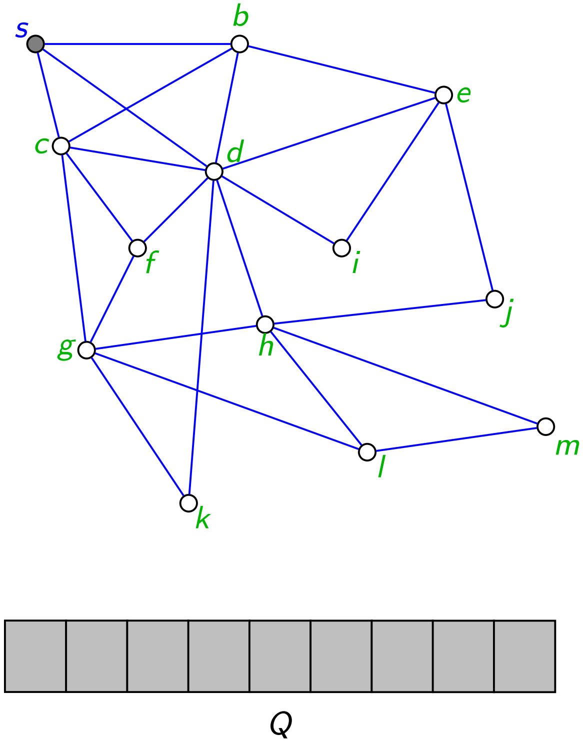


I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }

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

BFS(G, s)



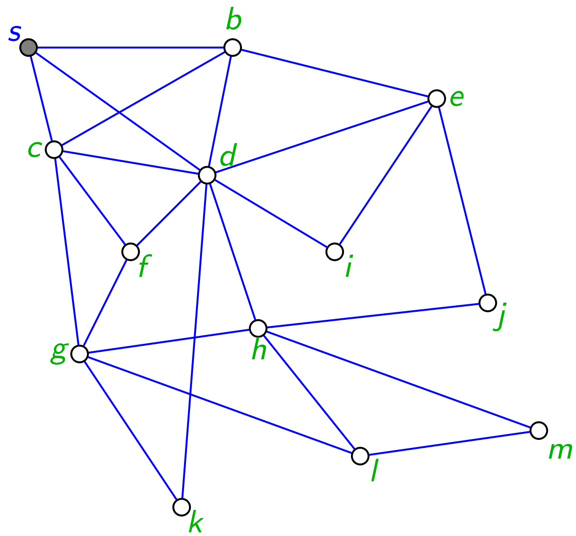
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

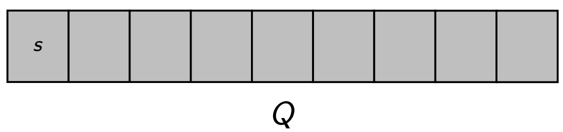
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }

  $color[s] \leftarrow \mathrm{GRAY}$;

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

BFS(G, s)



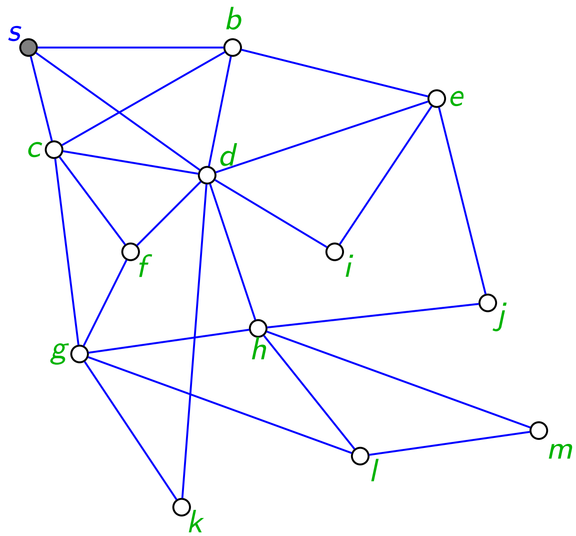


I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
for each vertex $u \in V \setminus \{s\}$ {
$color[u] \leftarrow white$;
$d[u] \leftarrow \infty$;
$\pi[u] \leftarrow$ NIL; }
$color[s] \leftarrow$ GRAY;
$d[s] \leftarrow 0$;
$\pi[s] \leftarrow$ NIL;
$Q \leftarrow \emptyset$;
ENQUEUE(Q, s);

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

BFS(G, s)



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

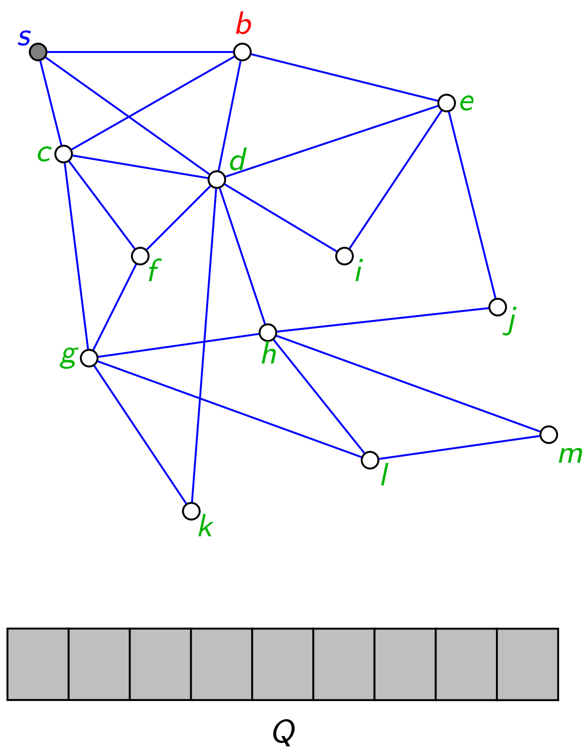
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }

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

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

BFS(G, s)

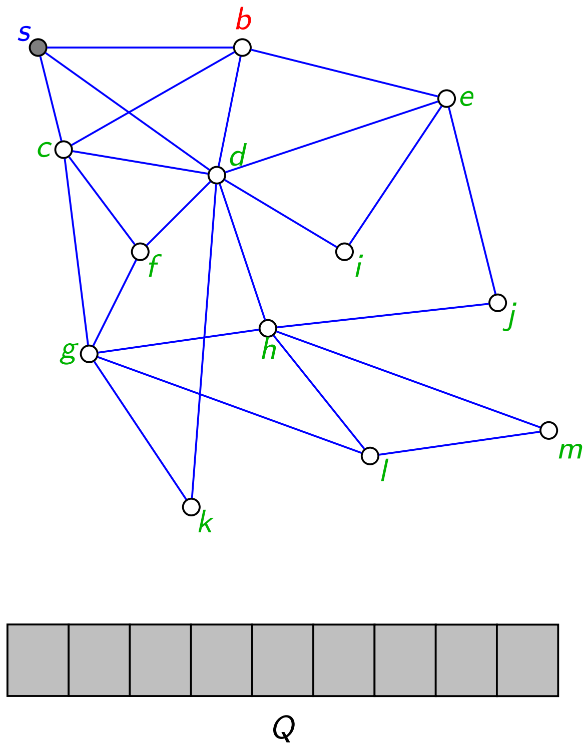


V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \text{NIL}$; }
  $color[s] \leftarrow \text{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \text{NIL}$;
  $Q \leftarrow \emptyset$;
  $\text{Enqueue}(Q, s)$;
  while $(Q \neq \emptyset)$ {
    $u \leftarrow \text{Dequeue}(Q)$;
    for each $v \in Adj[u]$ {

BFS(G, s)

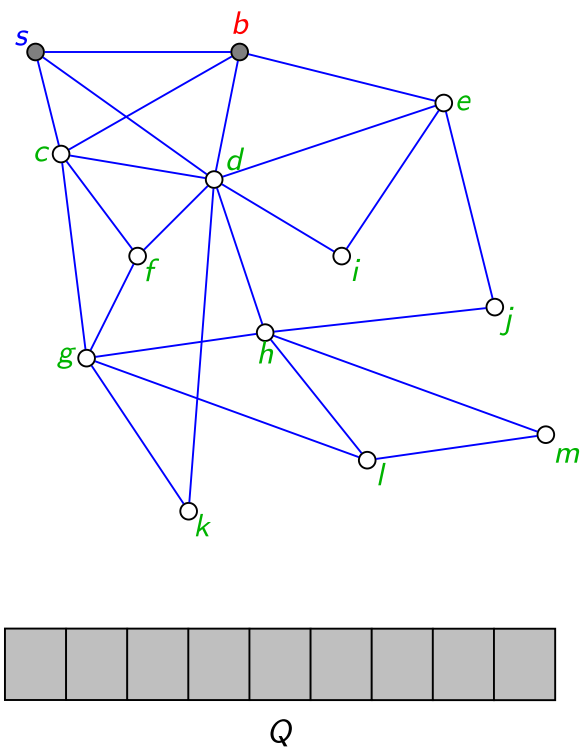


V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \text{NIL}$; }
  $color[s] \leftarrow \text{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \text{NIL}$;
  $Q \leftarrow \emptyset$;
  $\text{ENQUEUE}(Q, s)$;
  while $(Q \neq \emptyset)$ {
    $u \leftarrow \text{DEQUEUE}(Q)$;
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {

BFS(G, s)

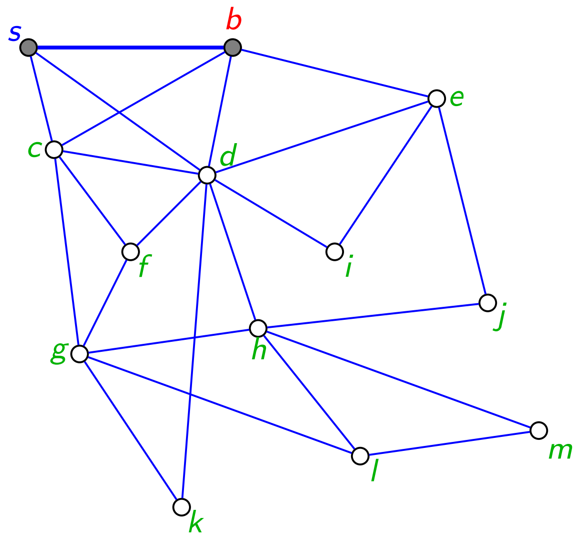


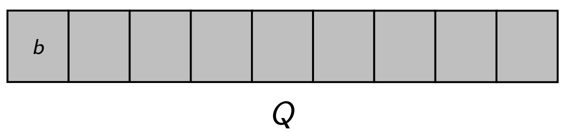
V_0: $\{s\}$
V_1: $\{\}$
V_2: $\{\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }
  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$;
  $\textsc{Enqueue}(Q, s)$;
  while $(Q \neq \emptyset)$ {
    $u \leftarrow \textsc{Dequeue}(Q)$;
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow \mathrm{GRAY}$;

$\mathrm{BFS}(G, s)$



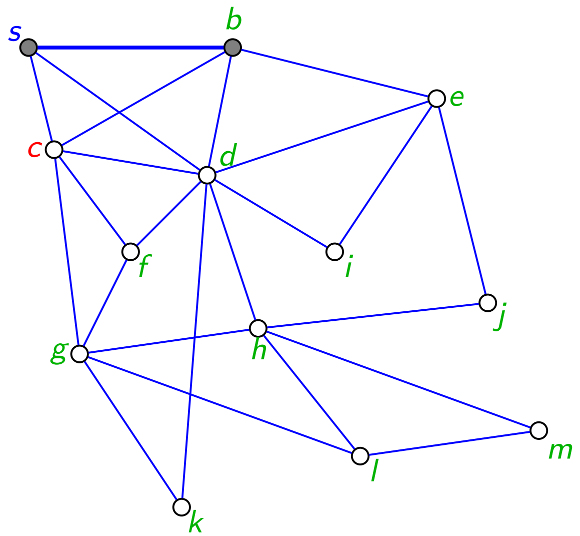


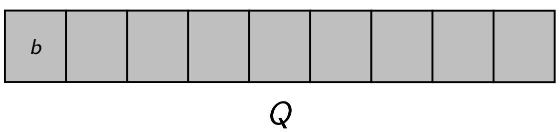
V_0: $\{s\}$
V_1: $\{b\}$
V_2: $\{\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }
  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$;
  $\textsc{Enqueue}(Q, s)$;
  while $(Q \neq \emptyset)$ {
    $u \leftarrow \textsc{Dequeue}(Q)$;
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow \mathrm{GRAY}$;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        $\textsc{Enqueue}(Q, v)$; } }

BFS(G, s)





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

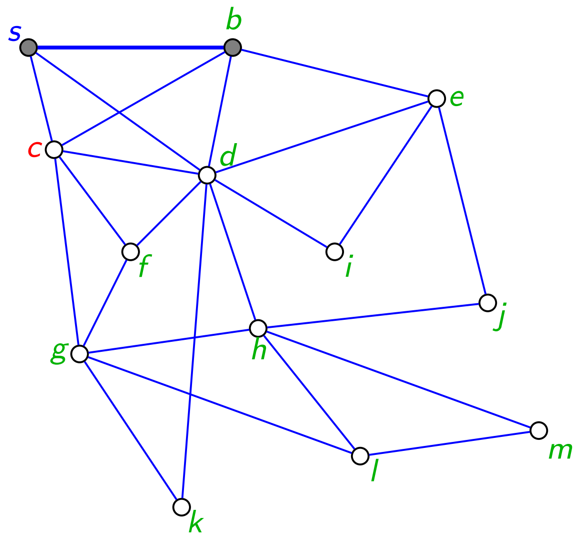
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

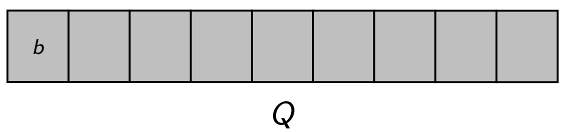
Begin
 for each vertex $u \in V \setminus \{s\}$ {
  $color[u] \leftarrow white$;
  $d[u] \leftarrow \infty$;
  $\pi[u] \leftarrow \mathrm{NIL}$; }

 $color[s] \leftarrow \mathrm{GRAY}$;
 $d[s] \leftarrow 0$;
 $\pi[s] \leftarrow \mathrm{NIL}$;
 $Q \leftarrow \emptyset$;
 ENQUEUE(Q, s);

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

BFS(G, s)





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

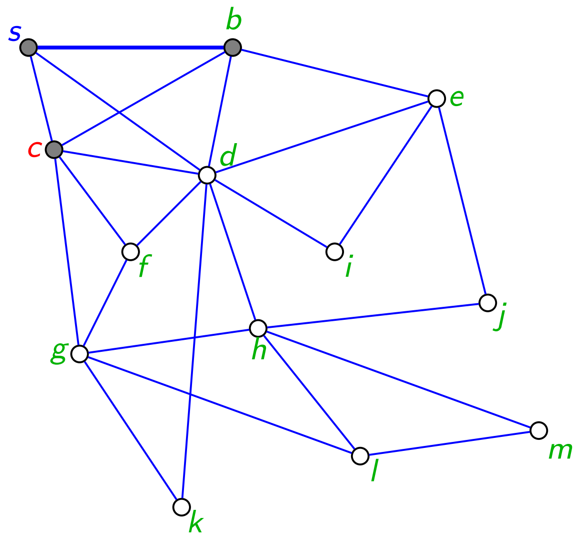
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

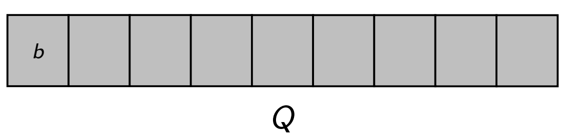
Begin
for each vertex $u \in V \setminus \{s\}$ {
$color[u] \leftarrow white$;
$d[u] \leftarrow \infty$;
$\pi[u] \leftarrow \mathrm{NIL}$; }

$color[s] \leftarrow \mathrm{GRAY}$;
$d[s] \leftarrow 0$;
$\pi[s] \leftarrow \mathrm{NIL}$;
$Q \leftarrow \emptyset$;
$\textsc{Enqueue}(Q, s)$;

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

BFS(G, s)





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

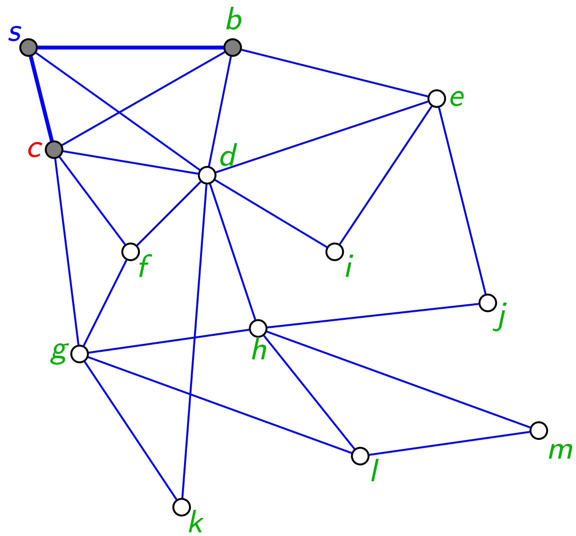
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

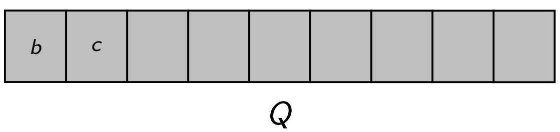
Begin
for each vertex $u \in V \setminus \{s\}$ {
$color[u] \leftarrow white$;
$d[u] \leftarrow \infty$;
$\pi[u] \leftarrow \mathrm{NIL}$; }

$color[s] \leftarrow \mathrm{GRAY}$;
$d[s] \leftarrow 0$;
$\pi[s] \leftarrow \mathrm{NIL}$;
$Q \leftarrow \emptyset$;
$\textsc{Enqueue}(Q, s)$;

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

BFS(G, s)





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

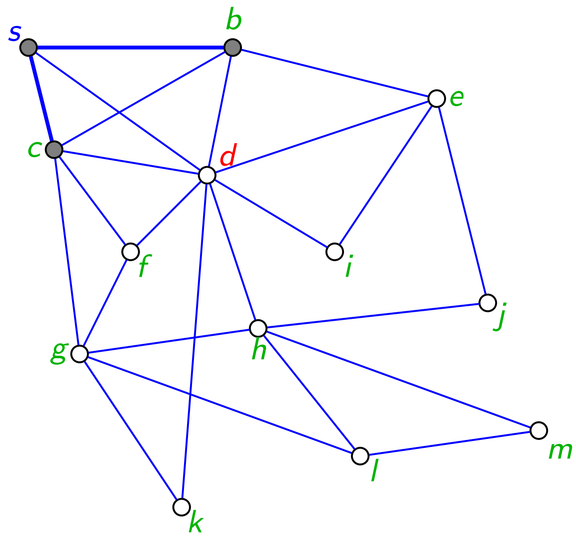
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

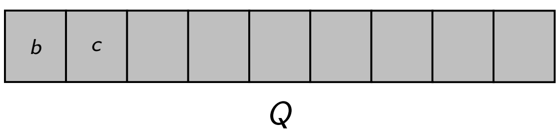
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \text{NIL}$; }

  $color[s] \leftarrow \text{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \text{NIL}$;
  $Q \leftarrow \emptyset$;
  ENQUEUE(Q, s);

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

BFS(G, s)





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

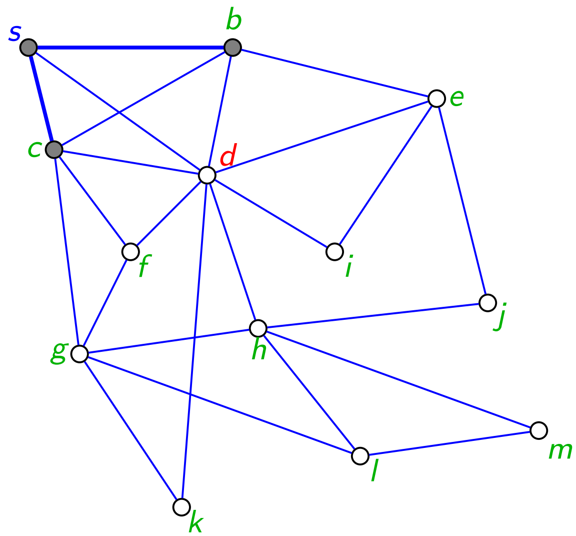
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

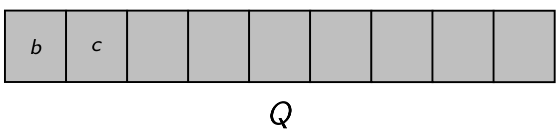
Begin
for each vertex $u \in V \setminus \{s\}$ {
$color[u] \leftarrow white$;
$d[u] \leftarrow \infty$;
$\pi[u] \leftarrow \mathrm{NIL}$; }

$color[s] \leftarrow \mathrm{GRAY}$;
$d[s] \leftarrow 0$;
$\pi[s] \leftarrow \mathrm{NIL}$;
$Q \leftarrow \emptyset$;
$\textsc{Enqueue}(Q, s)$;

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

BFS(G, s)



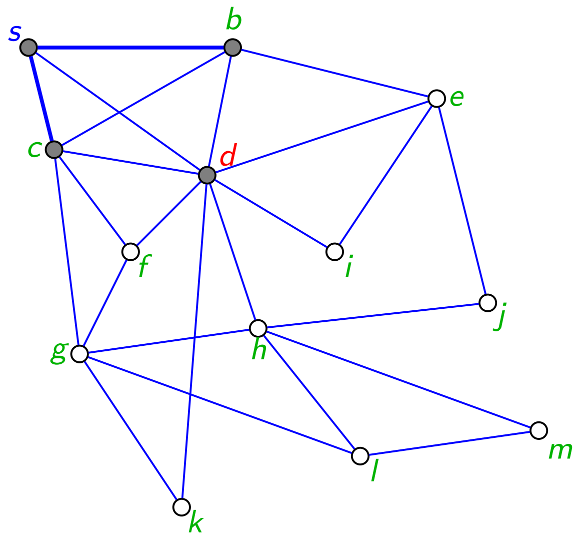


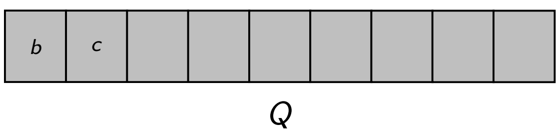
V_0: $\{s\}$
V_1: $\{b, c\}$
V_2: $\{\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;
  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: $\{\}$

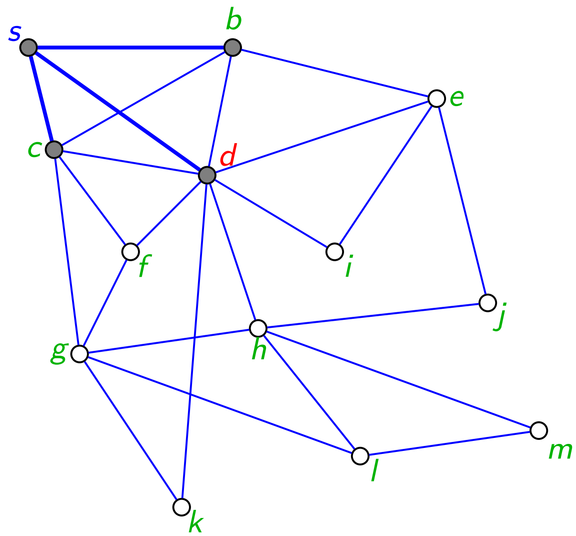
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

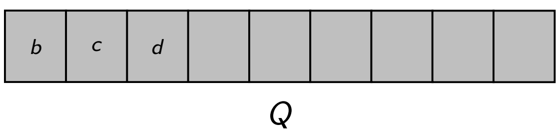
Begin
for each vertex $u \in V \setminus \{s\}$ {
$color[u] \leftarrow white$;
$d[u] \leftarrow \infty$;
$\pi[u] \leftarrow \mathrm{NIL}$; }

$color[s] \leftarrow \mathrm{GRAY}$;
$d[s] \leftarrow 0$;
$\pi[s] \leftarrow \mathrm{NIL}$;
$Q \leftarrow \emptyset$;
$\textsc{Enqueue}(Q, s)$;

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

BFS(G, s)





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

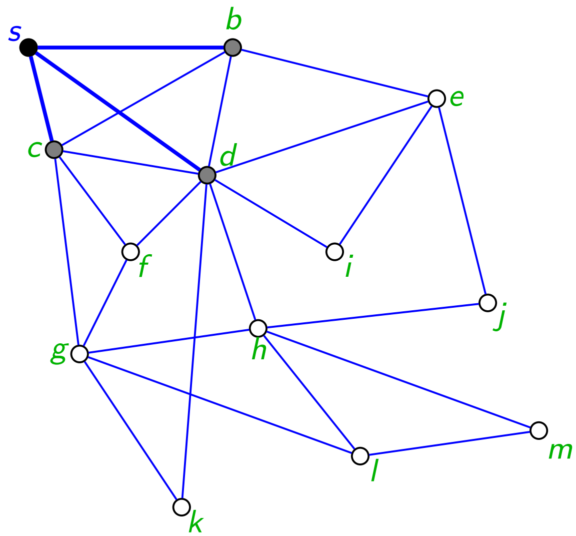
I/P: A graph $G = (V, E)$ is represented using adjacency lists and a source s.

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \text{NIL}$; }

  $color[s] \leftarrow \text{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \text{NIL}$;
  $Q \leftarrow \emptyset$;
  $\text{Enqueue}(Q, s)$;

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

BFS(G, s)



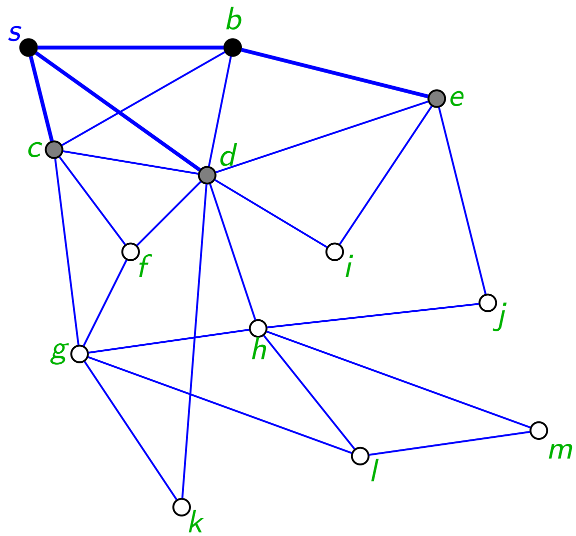
Q

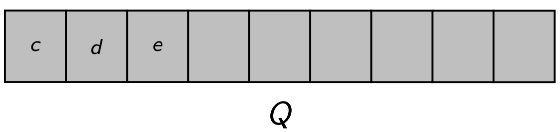
V_0: $\{s\}$
V_1: $\{b,\ c,\ d\}$
V_2: $\{\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;
  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)



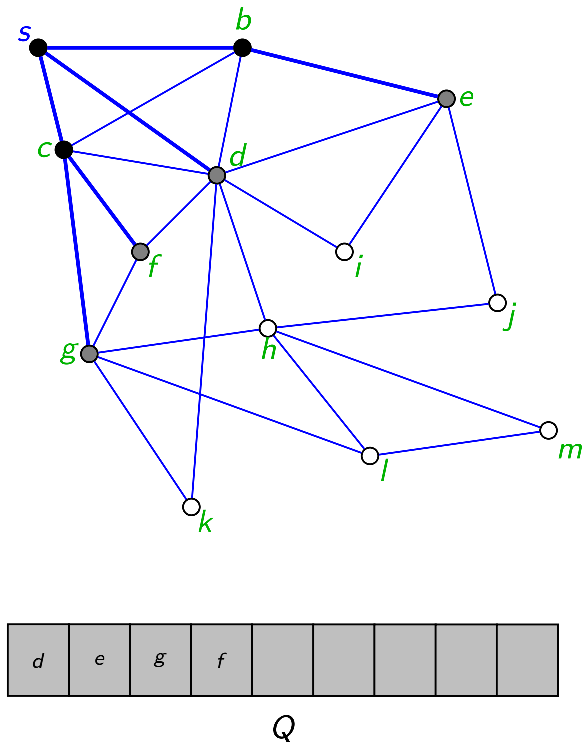


V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }
  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$;
  $\textsc{Enqueue}(Q, s)$;
  while $(Q \neq \emptyset)$ {
    $u \leftarrow \textsc{Dequeue}(Q)$;
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow \mathrm{GRAY}$;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        $\textsc{Enqueue}(Q, v)$; } }
    $color[u] \leftarrow \mathrm{BLACK}$; }
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.

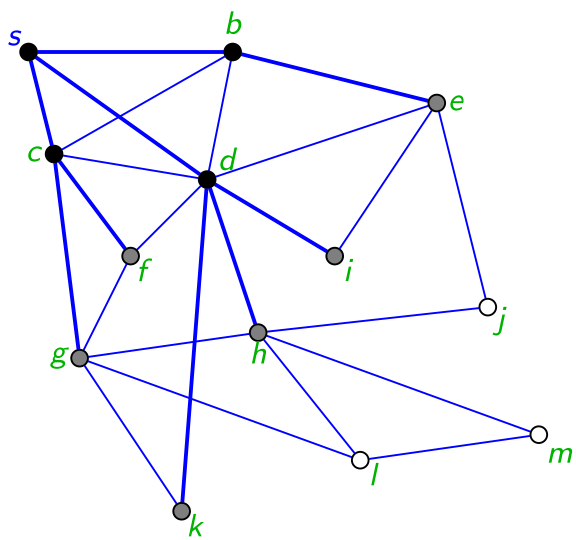
Begin
for each vertex $u \in V \setminus \{s\}$ {
$color[u] \leftarrow white$;
$d[u] \leftarrow \infty$;
$\pi[u] \leftarrow$ NIL; }

$color[s] \leftarrow$ GRAY;
$d[s] \leftarrow 0$;
$\pi[s] \leftarrow$ NIL;
$Q \leftarrow \emptyset$;
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)



e	g	f	k	h	i			

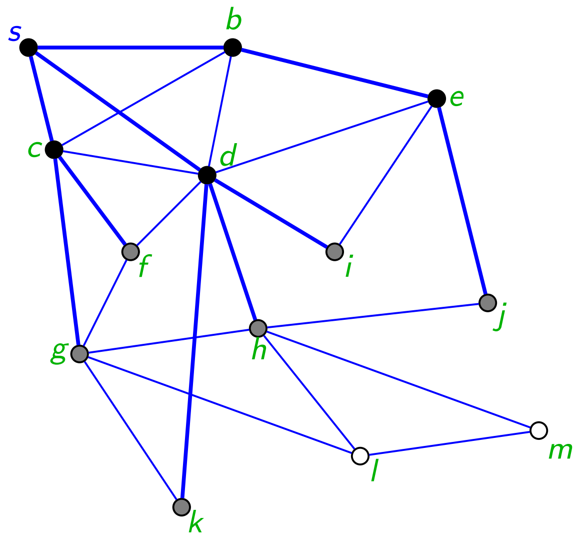
Q

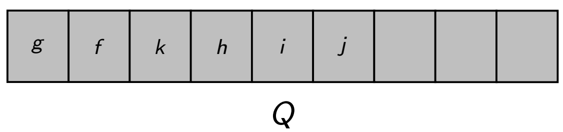
V_0: $\{s\}$
V_1: $\{b,\ c,\ d\}$
V_2: $\{e,\ g,\ f,\ k,\ h,\ i\}$
V_3: $\{\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;
  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)



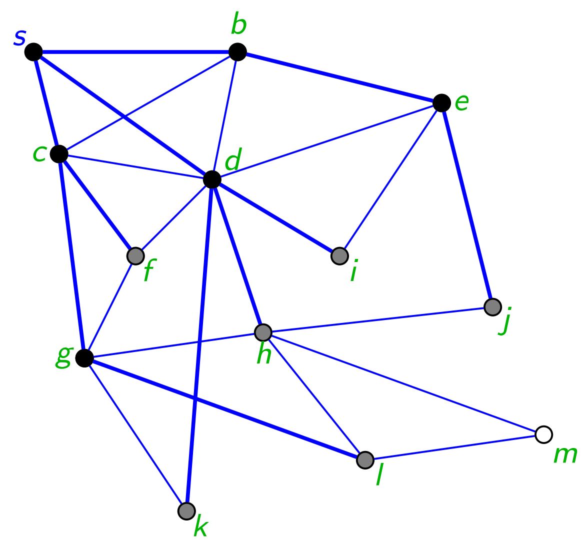


V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;
  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)



f	k	h	i	j	l			

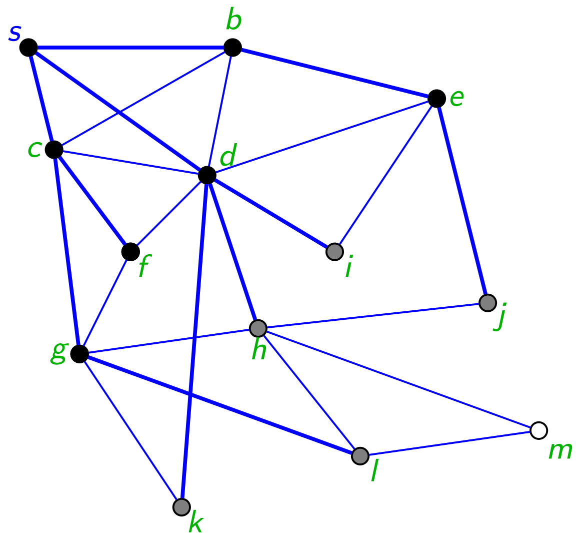
Q

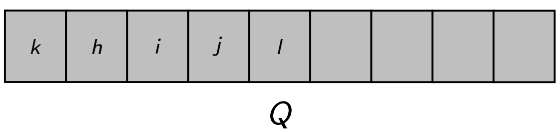
V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;
  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)



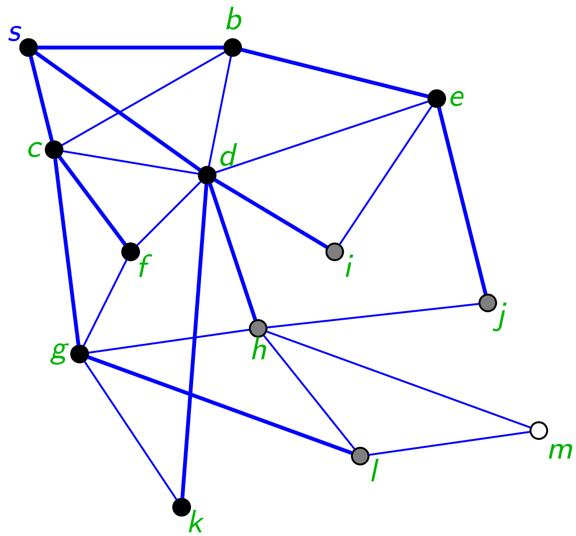


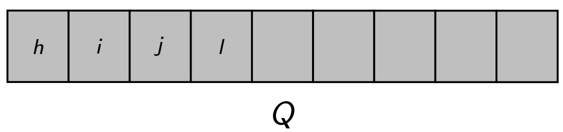
V_0: $\{s\}$
V_1: $\{b,\ c,\ d\}$
V_2: $\{e,\ g,\ f,\ k,\ h,\ i\}$
V_3: $\{j,\ l\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }
  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$;
  $\textsc{Enqueue}(Q, s)$;
  while $(Q \neq \emptyset)$ {
    $u \leftarrow \textsc{Dequeue}(Q)$;
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow \mathrm{GRAY}$;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        $\textsc{Enqueue}(Q, v)$; } }
    $color[u] \leftarrow \mathrm{BLACK}$; }
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.

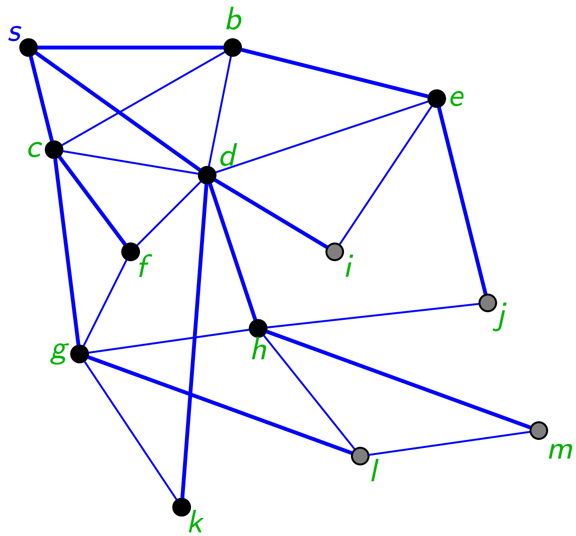
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }

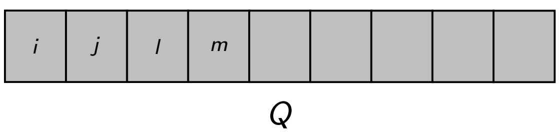
  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$;
  $\textsc{Enqueue}(Q, s)$;

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

End

$\mathrm{BFS}(G, s)$





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.

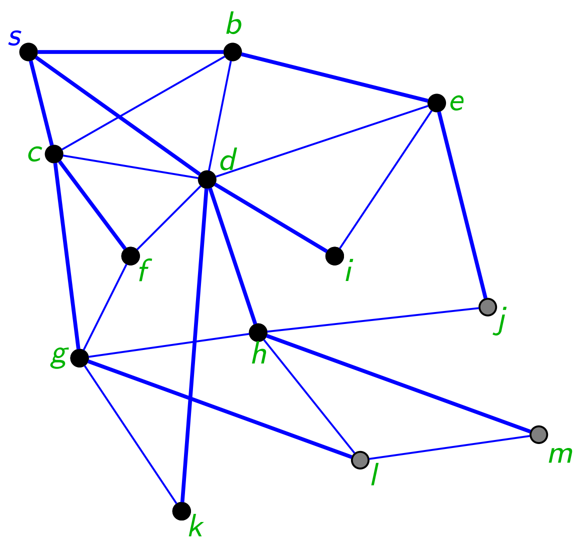
Begin
for each vertex $u \in V \setminus \{s\}$ {
$color[u] \leftarrow white$;
$d[u] \leftarrow \infty$;
$\pi[u] \leftarrow \mathrm{NIL}$; }

$color[s] \leftarrow \mathrm{GRAY}$;
$d[s] \leftarrow 0$;
$\pi[s] \leftarrow \mathrm{NIL}$;
$Q \leftarrow \emptyset$;
$\textsc{Enqueue}(Q, s)$;

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

End

BFS(G, s)



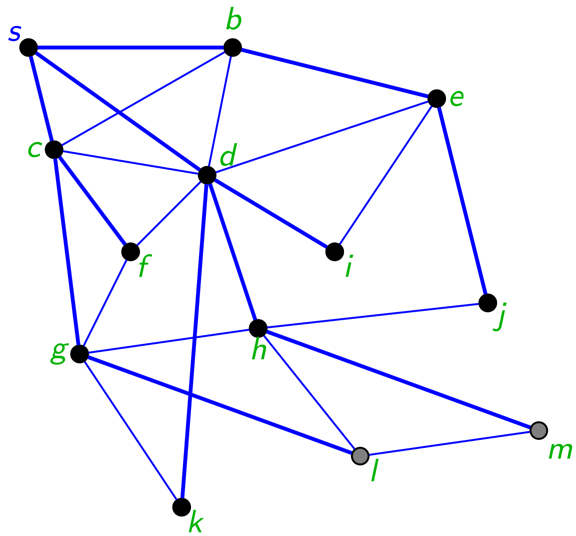
Q

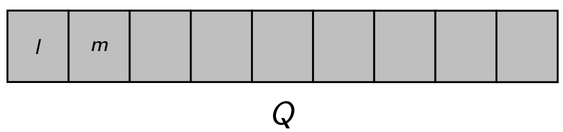
V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l, m\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;
  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.

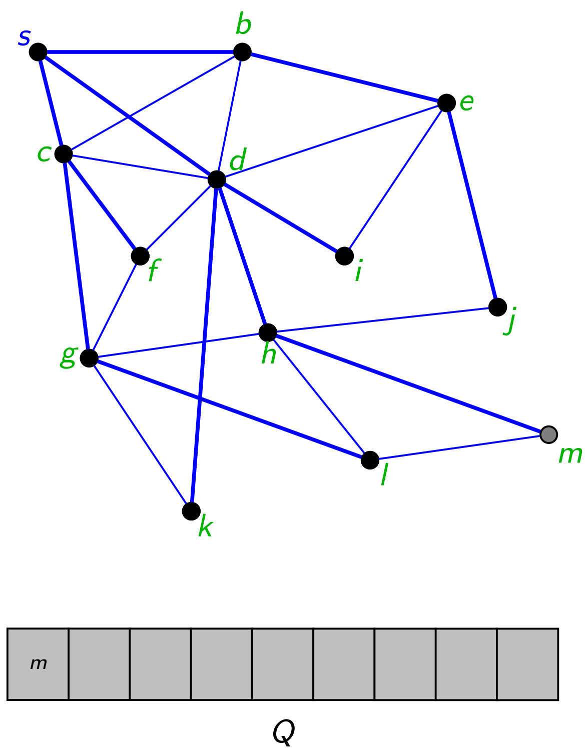
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \text{NIL}$; }

  $color[s] \leftarrow \text{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \text{NIL}$;
  $Q \leftarrow \emptyset$;
  $\text{ENQUEUE}(Q, s)$;

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

End

BFS(G, s)

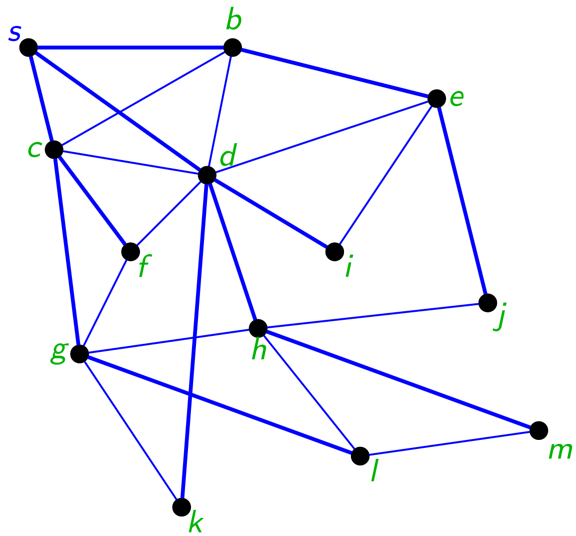


V_0: $\{s\}$
V_1: $\{b, c, d\}$
V_2: $\{e, g, f, k, h, i\}$
V_3: $\{j, l, m\}$

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow$ NIL; }
  $color[s] \leftarrow$ GRAY;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow$ NIL;
  $Q \leftarrow \emptyset$;
  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)



Q

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.

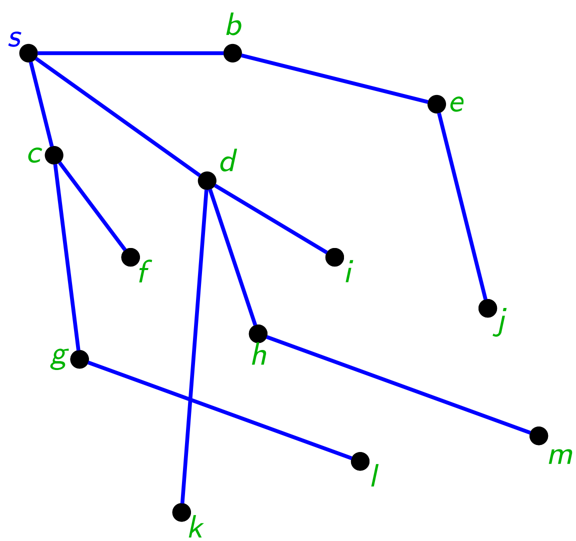
Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }

  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$;
  $\textsc{Enqueue}(Q, s)$;

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

End

BFS(G, s)

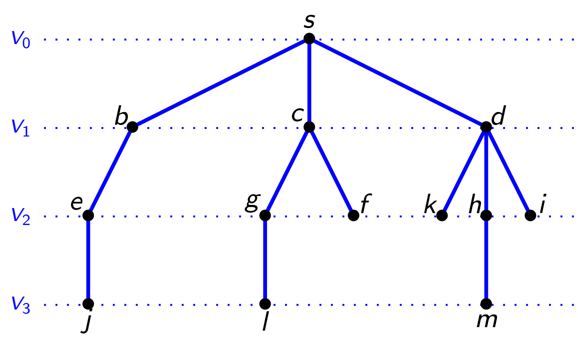


BFS Tree

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

Begin
  for each vertex $u \in V \setminus \{s\}$ {
    $color[u] \leftarrow white$;
    $d[u] \leftarrow \infty$;
    $\pi[u] \leftarrow \mathrm{NIL}$; }
  $color[s] \leftarrow \mathrm{GRAY}$;
  $d[s] \leftarrow 0$;
  $\pi[s] \leftarrow \mathrm{NIL}$;
  $Q \leftarrow \emptyset$;
  $\textsc{Enqueue}(Q, s)$;
  while $(Q \neq \emptyset)$ {
    $u \leftarrow \textsc{Dequeue}(Q)$;
    for each $v \in Adj[u]$ {
      if $(color[v] = white)$ {
        $color[v] \leftarrow \mathrm{GRAY}$;
        $d[v] \leftarrow d[u] + 1$;
        $\pi[v] \leftarrow u$;
        $\textsc{Enqueue}(Q, v)$; } }
    $color[u] \leftarrow \mathrm{BLACK}$; }
End

BFS Tree



Note:

- BFS partitions the vertices according to their distance from s.
- A vertex at level i can not have any neighbor from level $i - 2$ or higher.

Correctness of BFS Tree

After BFS has been run from a source s on a graph G, the following lemma shows that the predecessor subgraph is a breadth-first tree.

Lemma

When applied to a directed or undirected graph $G = (V, E)$, procedure BFS constructs π so that the predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ is a breadth-first tree.

Determine if a Graph is *Bipartite*

Complete Graphs and Cliques

Definition (Complete Graph)

A graph is said to be **complete** if every two vertices of G are connected by an edge.

Note:

- Undirected graph: $|E| = \binom{|V|}{2}$.
- Directed graph: $|E| = |V|(|V| - 1)$.

Complete Graphs and Cliques

Definition (Complete Graph)

A graph is said to be **complete** if every two vertices of G are connected by an edge.

Note:

- Undirected graph: $|E| = \binom{|V|}{2}$.
- Directed graph: $|E| = |V|(|V| - 1)$.

Definition (Clique)

A complete subgraph H of a graph G is called a **clique** of G.

Definition (Independent Set)

An **independent** set S of a graph $G = (V, E)$ is a subset of V, such that no two vertices of S are connected by an edge.

Definition (Independent Set)

An **independent** set S of a graph $G = (V, E)$ is a subset of V, such that no two vertices of S are connected by an edge.

Definition (Vertex Cover)

A **vertex cover** is a subset $T \subseteq V$, s.t., for any edge $e \in E$, e is incident on some vertex in T.

Complement of a Graph

Definition (Complement of a Graph)

Complement of a graph $G = (V, E)$ is the graph $\bar{G} = (V, \bar{E})$, where

$$\bar{E} = \{\{u, v\} : u \neq v \text{ and } \{u, v\} \notin E\}.$$

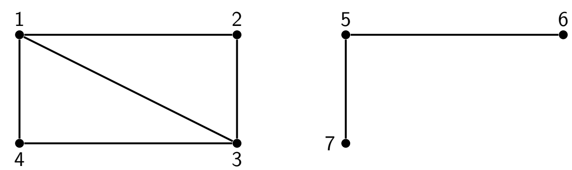
Complement of a Graph

Definition (Complement of a Graph)

Complement of a graph $G = (V, E)$ is the graph $\bar{G} = (V, \bar{E})$, where

$$\bar{E} = \{\{u, v\} : u \neq v \text{ and } \{u, v\} \notin E\}.$$

Example:



$$
\begin{aligned}
G &= (V, E), \text{ where} \\
V &= \{1, 2, 3, 4, 5, 6, 7\} \\
E &= \{(1, 2), (1, 3), (1, 4), (2, 3), (3, 4), (5, 6), (5, 7)\}.
\end{aligned}
$$

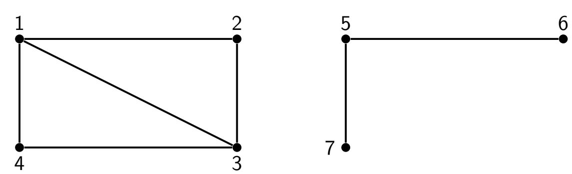
Complement of a Graph

Definition (Complement of a Graph)

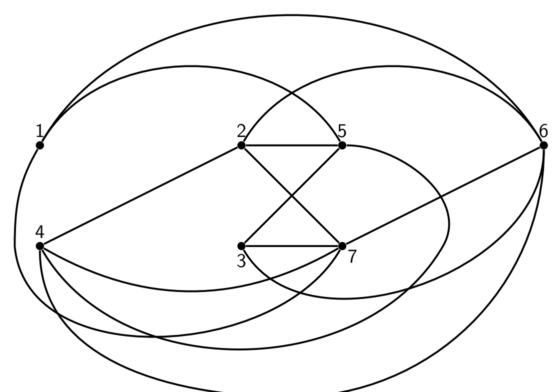
Complement of a graph $G = (V, E)$ is the graph $\bar{G} = (V, \bar{E})$, where

$$\bar{E} = \{\{u, v\} : u \neq v \text{ and } \{u, v\} \notin E\}.$$

Example:

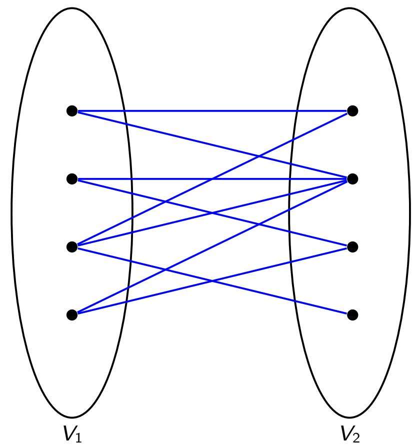


$$\begin{aligned} G &= (V, E), \text{ where} \\ V &= \{1, 2, 3, 4, 5, 6, 7\} \\ E &= \{(1, 2), (1, 3), (1, 4), (2, 3), (3, 4), (5, 6), (5, 7)\}. \end{aligned}$$



$$\begin{aligned} \bar{G} &= (V, E), \text{ where} \\ V &= \{1, 2, 3, 4, 5, 6, 7\} \\ \bar{E} &= \{(1, 5), (1, 6), (1, 7), (2, 4), (2, 5), (2, 6), (2, 7), (3, 5), (3, 6), (3, 7), \\ &\quad (4, 5), (4, 6), (4, 7), (6, 7)\}. \end{aligned}$$

Bipartite Graph

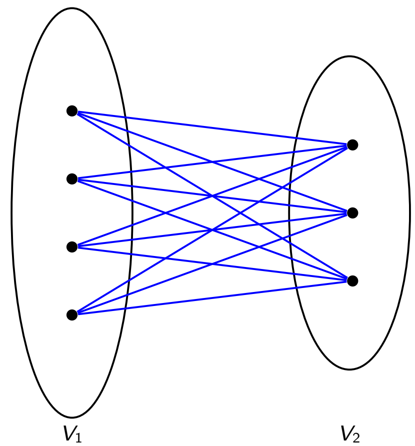


Definition (Bipartite Graph)

Let $G = (V, E)$ be a graph. It is said to be **bipartite or bigraph** if $V = V_1 \cup V_2$, where $V_1, V_2 \neq \emptyset$ and $V_1 \cap V_2 = \emptyset$, s.t., V_1 and V_2 are independent sets of G and any $e \in E$ is incident on a vertex in V_1 and a vertex in V_2.

A bigraph graph is usually written as $G = (V_1, V_2, E)$.

Complete Bipartite Graph



Complete Bipartite Graph $K_{4,3}$

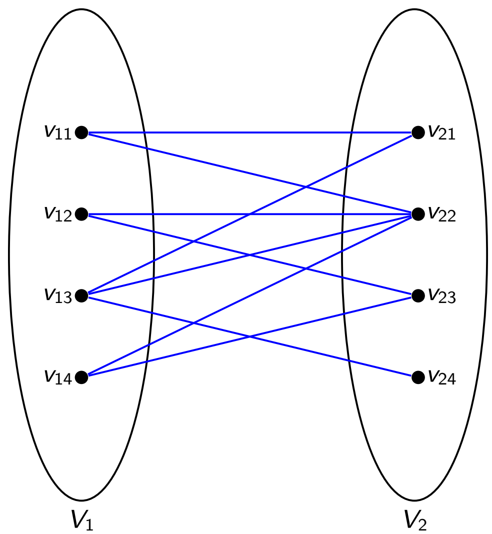
Definition (Complete Bipartite Graph)

Let $G = (V_1, V_2, E)$ be a bigraph. It is said to be a **complete bigraph** if for any $v_1 \in V_2$ and $v_2 \in V_2$, $\{v_1, v_2\} \in E$.

Let $|V_1| = m$ and $|V_2| = n$, then a complete bipartite graph is denoted as $K_{m,n}$.

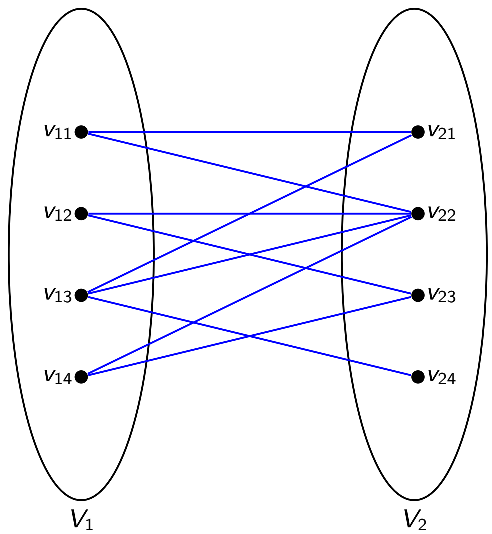
Nontriviality

Question: Is the following graph a Bipartite graph?



Nontriviality

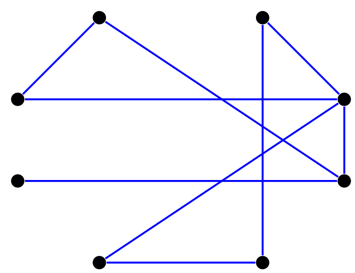
Question: Is the following graph a Bipartite graph?



Answer: Yes (Easy!).

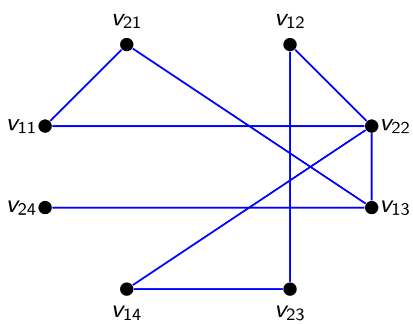
Nontriviality

Question: Is the following graph a Bipartite graph?

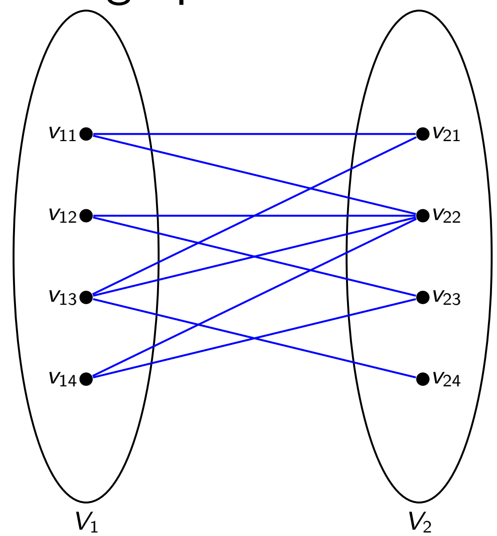


Nontriviality

Question: Is the following graph a Bipartite graph?



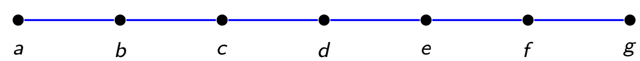
$=$



Answer: Yes (Not so easy!).

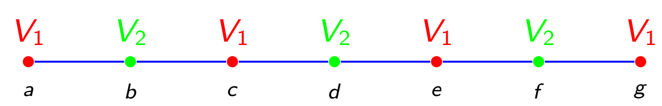
Line Graph?

Question: Is this a path Bipartite graph?



Line Graph?

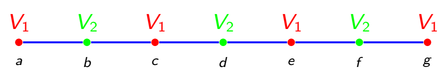
Question: Is this a path Bipartite graph?



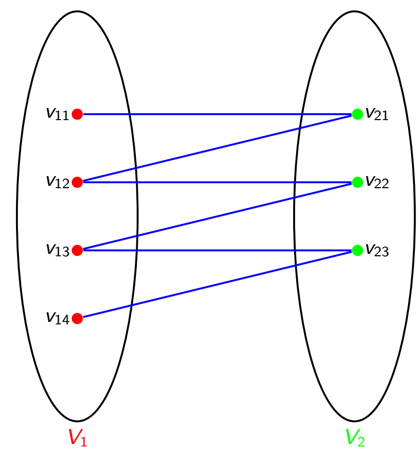
Answer: Yes.

Line Graph?

Question: Is this a path Bipartite graph?



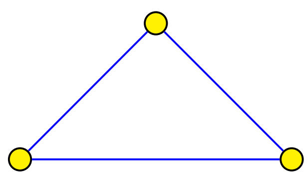
=

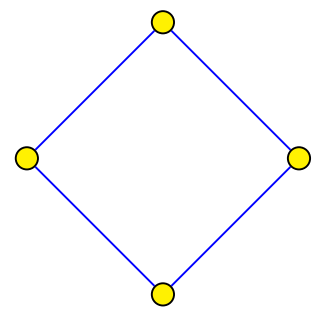


Answer: Yes.

Cycle?

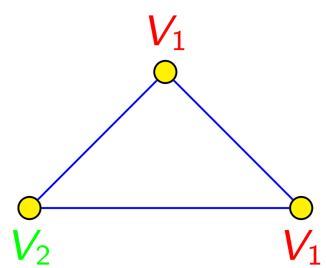
Question: Is the following graph a Bipartite graph?



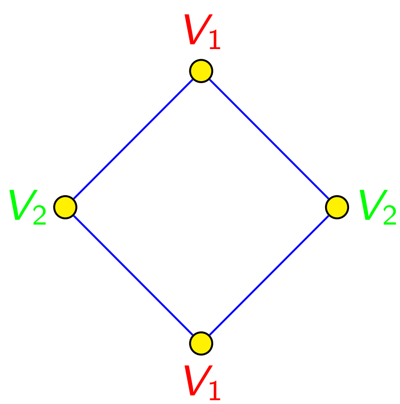


Cycle?

Question: Is the following graph a Bipartite graph?



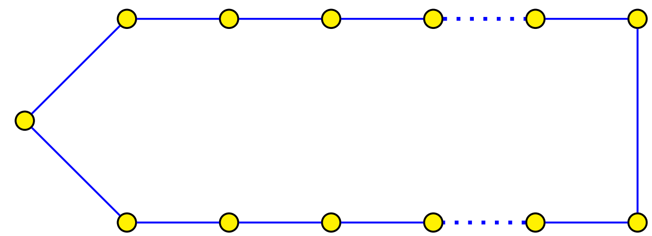
$\therefore$ **Non-bipartite**



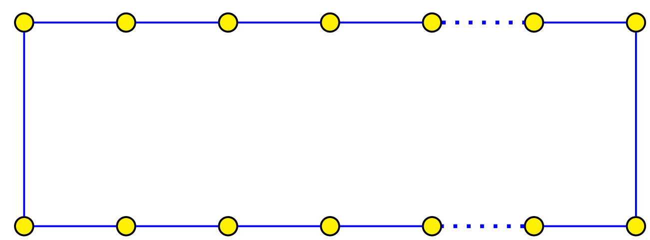
$\therefore$ **Bipartite**

In General:

Odd Cycle:



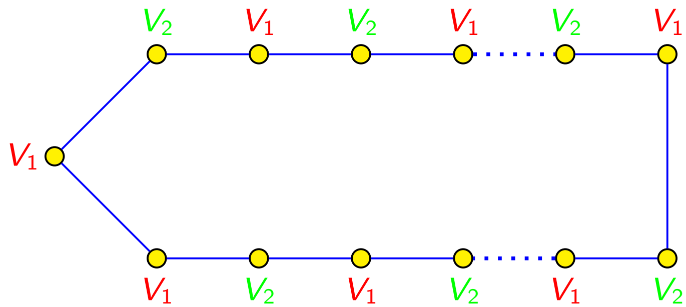
Even Cycle:



Cycle?

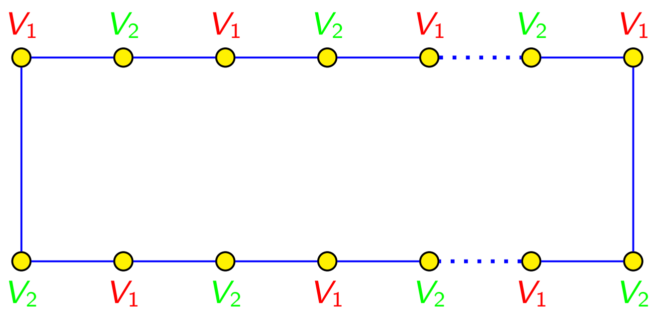
In General:

Odd Cycle:



$\therefore$ **Non-bipartite**

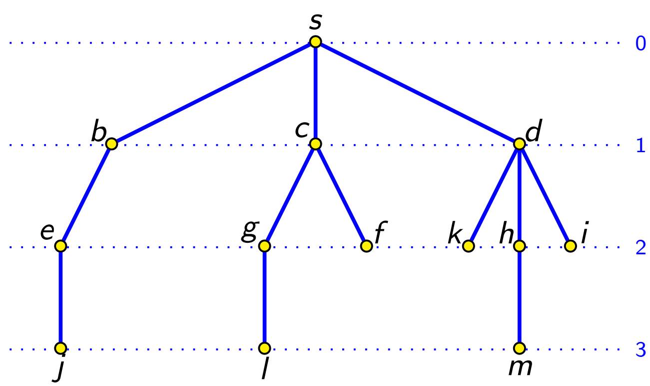
Even Cycle:



$\therefore$ **Bipartite**

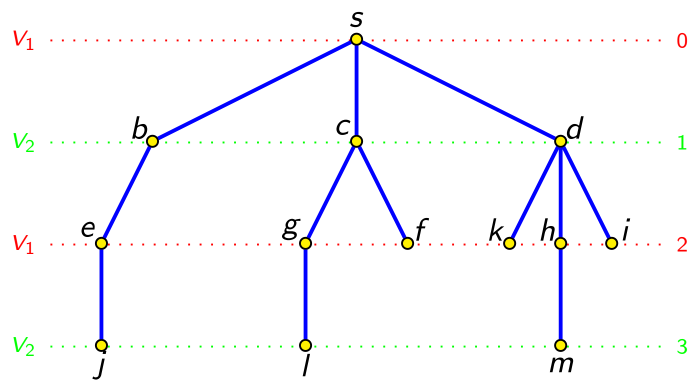
Tree?

Question: Is the following graph a Bipartite graph?



Tree?

Question: Is the following graph a Bipartite graph?



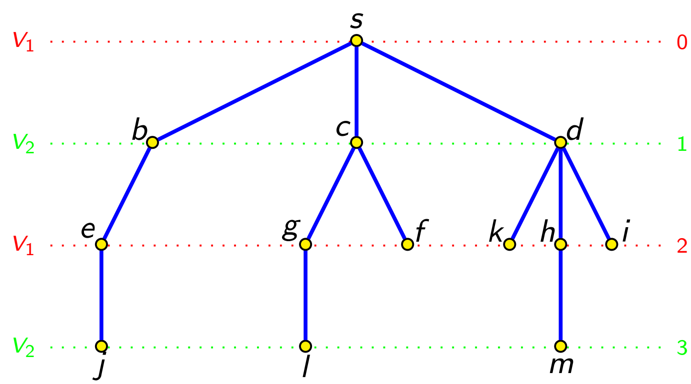
Answer: Yes.

- **Even level vertices:** V_1
- **Odd level vertices:** V_2

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

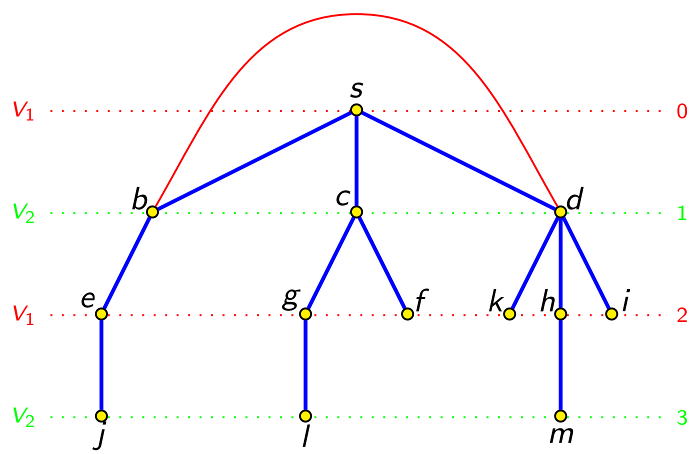
Step 1: Run BFS algorithm to generate the BFS tree.



An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



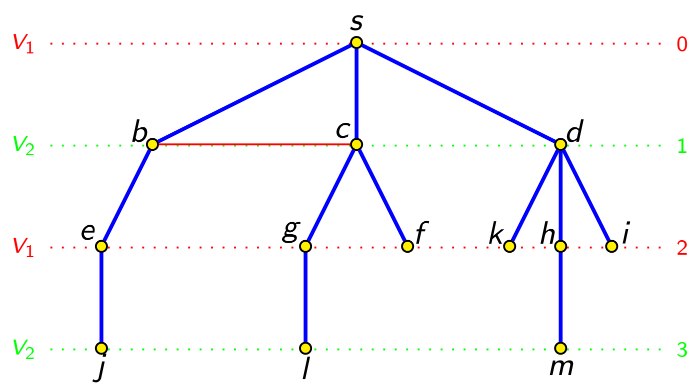
Step 2: Consider the non-tree edges one by one:

- $\{b, d\}$: Creates a triangle (odd cycle)!
  Conclusion: Non-bipartite.

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



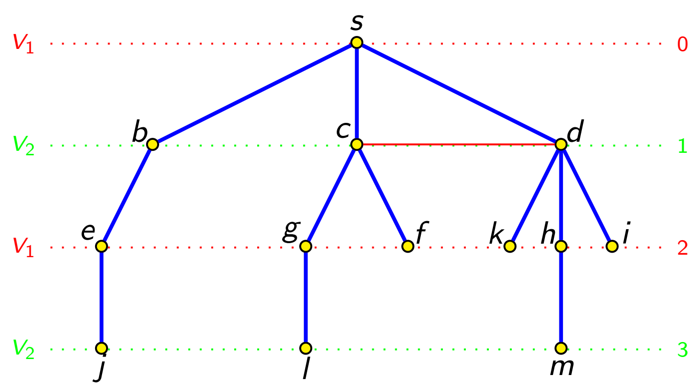
Step 2: Consider the non-tree edges one by one: Moving on . . .

- $\{b, c\}$: Creates a triangle (odd cycle)!
  Conclusion: Non-bipartite.

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



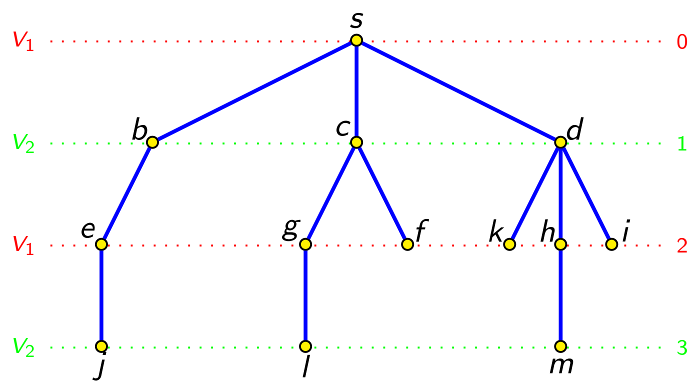
Step 2: Consider the non-tree edges one by one:

- $\{c, d\}$: Creates a triangle (odd cycle)!
  Conclusion: Non-bipartite.

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



Step 2: Consider the non-tree edges one by one:

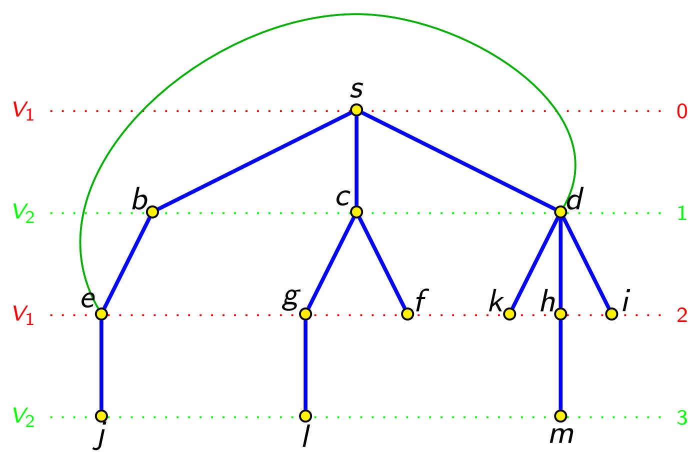
- **Observation:** If there is an edge with both end points at same level, then this edge creates a triangle (odd cycle). Therefore the graph is Non-bipartite.

$$E_{\text{Odd}} = \{\{b, d\}, \{b, c\}, \{c, d\}\}$$

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.

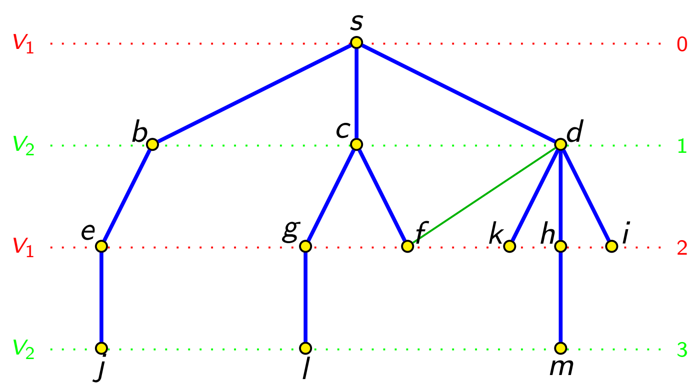


Step 2: Consider the non-tree edges one by one:

- $\{d, e\}$: Creates an even cycle!
  Conclusion: May be Bipartite!

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



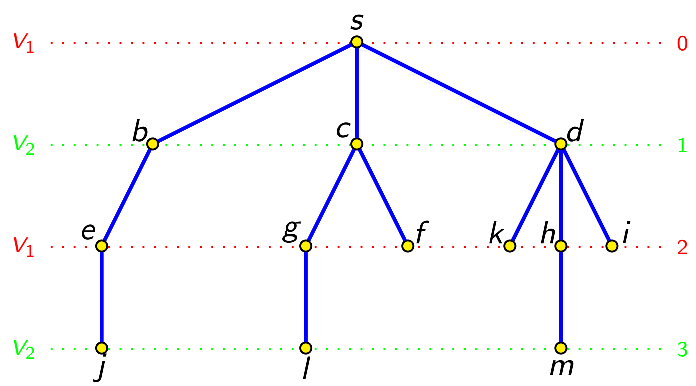
Step 2: Consider the non-tree edges one by one:

- $\{d, f\}$: Creates an even cycle!
  Conclusion: May be Bipartite!

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



Step 2: Consider the non-tree edges one by one:

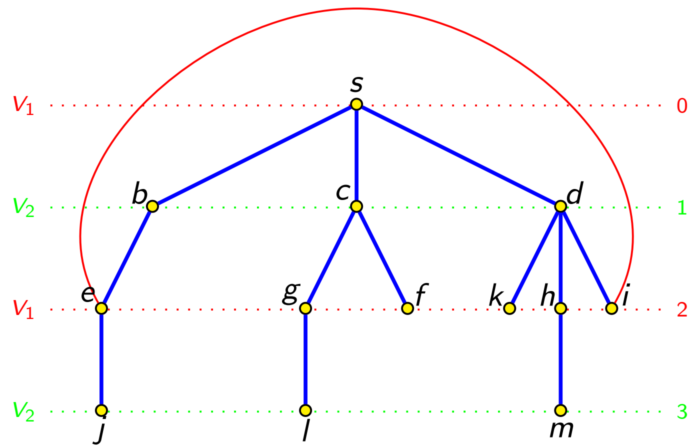
- **Observation:** If there is a non-tree edge that connects two consecutive levels of a BFS tree, then this edge creates an even cycle. In this case the graph may be Bipartite.

$$E_{\text{Even}} = \{\{f, c\}, \{c, s\}, \{s, d\}, \{d, f\}\}$$

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



Step 2: Consider the non-tree edges one by one:

- $\{e, i\}$: Creates a triangle (odd cycle)!
  Conclusion: Non-bipartite.

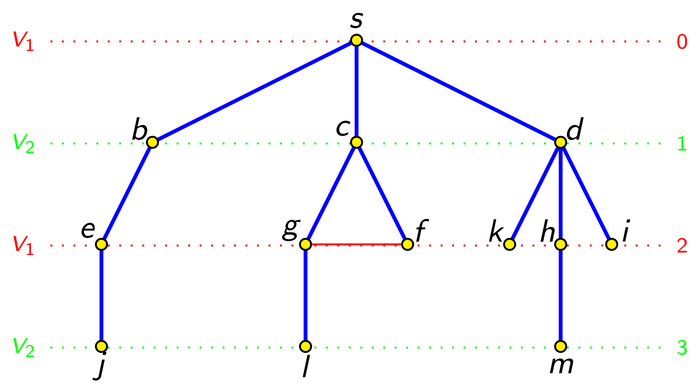
  Observation (Refined): If there is an edge with both end points at same level, then this edge creates an odd cycle. Therefore the graph is Non-bipartite.

$$E_{\text{Odd}} = \{\{e, b\}, \{b, s\}, \{s, d\}, \{d, i\}, \{e, i\}\}$$

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



Step 2: Consider the non-tree edges one by one:

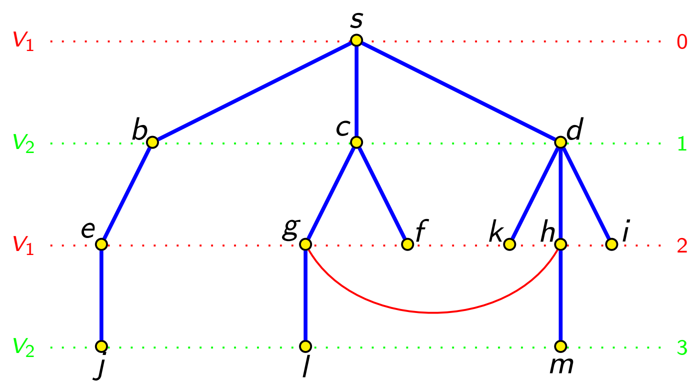
- $\{f, g\}$: Creates a triangle (odd cycle)!
  Conclusion: Non-bipartite.

$$E_{\text{Odd}} = \{\{g, c\}, \{c, f\}, \{f, g\}\}$$

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



Step 2: Consider the non-tree edges one by one:

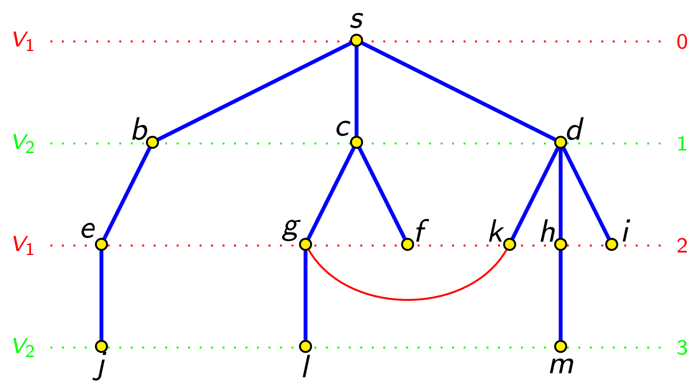
- $\{g, h\}$: Creates a triangle (odd cycle)!
  Conclusion: Non-bipartite.

$$E_{\text{Odd}} = \{\{g, c\}, \{c, s\}, \{s, d\}, \{d, h\}, \{g, h\}\}$$

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



Step 2: Consider the non-tree edges one by one:

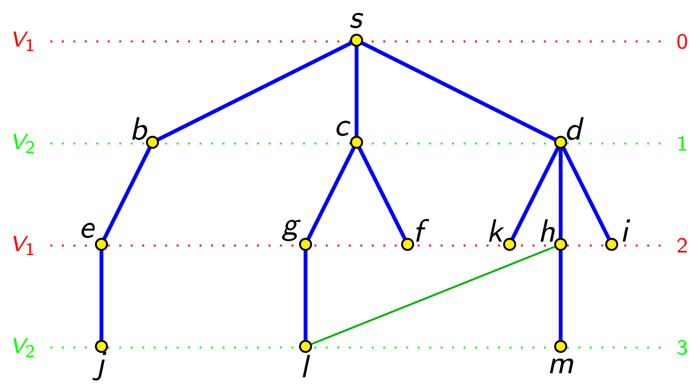
- $\{g, k\}$: Creates a triangle (odd cycle)!
  Conclusion: Non-bipartite.

$$E_{\text{Odd}} = \{\{g, c\}, \{c, s\}, \{s, d\}, \{d, k\}, \{g, k\}\}$$

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.

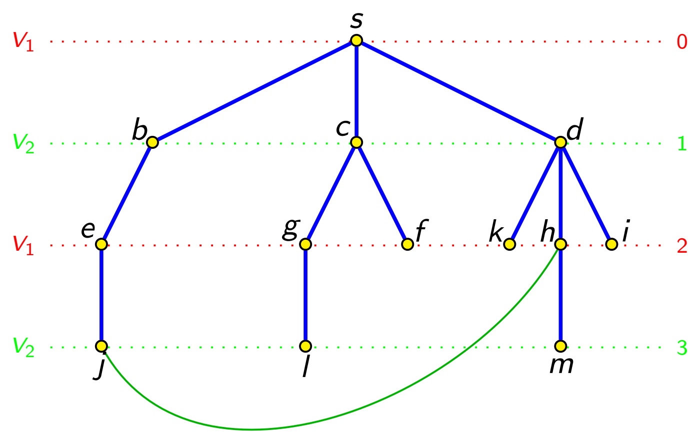


Step 2: Consider the non-tree edges one by one:

- $\{h, l\}$: Creates an even cycle!
  Conclusion: May be Bipartite!

$$E_{\text{Even}} = \{\{l, g\}, \{g, c\}, \{c, s\}, \{s, d\}, \{d, h\}, \{h, l\}\}$$

Assumption: The graph is a single connected component.
Step 1: Run BFS algorithm to generate the BFS tree.



Step 2: Consider the non-tree edges one by one:

- $\{h, j\}$: Creates an even cycle!
  Conclusion: May be Bipartite!

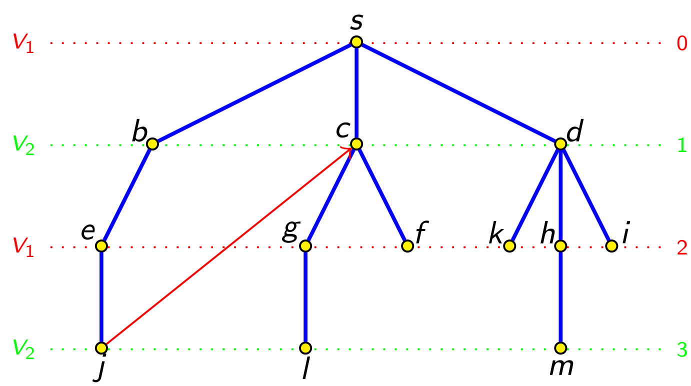
$$E_{\text{Even}} = \{\{j, e\}, \{e, b\}, \{b, s\}, \{s, d\}, \{d, h\}, (h, j)\}$$

Note: For undirected graphs the difference between levels can be at most 1.

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



Step 2: Consider the non-tree edges one by one:

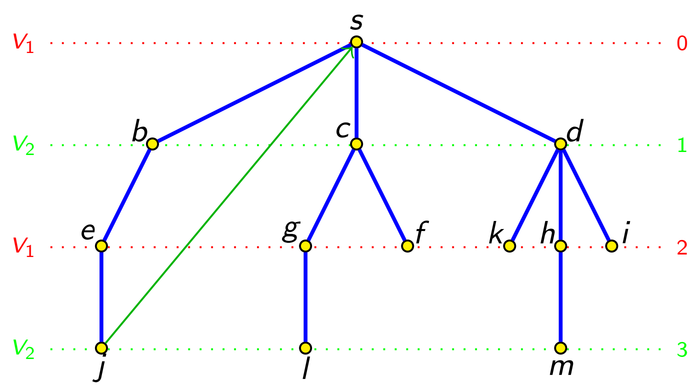
- Consider the directed edge (j, c): Creates an odd cycle!
  Conclusion: Non-bipartite.

  Observation (General): If there is an edge with both end points are at levels, s.t., the difference between their levels is an even number, then this edge creates an odd cycle. Therefore the graph is Non-bipartite.

An Algorithm for Determining if a Given Graph is Bipartite

Assumption: The graph is a single connected component.

Step 1: Run BFS algorithm to generate the BFS tree.



Step 2: Consider the non-tree edges one by one:

- Similarly, consider the directed edge (j, s): Creates an even cycle!
  Conclusion: May be Bipartite!

  Observation (General): If there is an edge with both end points are at levels, s.t., the difference between their levels is an odd number, then this edge creates an even cycle. Therefore the graph maybe Bipartite.

Cost Analysis

- BFS Algorithm: $\mathcal{O}(|V| + |E|)$.

- Scanning non-tree edges: $\mathcal{O}(|E| - (|V| - 1)) = \mathcal{O}(|E|)$.

  Note: Checking of non-tree edges can by done in a constant time by checking d-values of its corresponding vertices.

- **Total Complexity:** $\mathcal{O}(|V| + |E|) + \mathcal{O}(|E|) = \mathcal{O}(|V| + |E|)$.

Books and Other Materials Consulted

1. Definitions taken from Discrete Mathematics Lecture Notes (M. Tech (CS), Monsoon Semester, 2007) taught by Prof. Palash Sarkar (ASU, ISI Kolkata).

2. Bipartite checking algorithm from Prof. Surendar Baswana (CSE, IIT Kanpur) lecture slides.

Thank You for your kind attention!

Questions!!