Lecture 8

Instructor: Karteek Sreenivasaiah

11th September 2018

Graph
(directed)

A (directed) graph G is a two tuple (V, E) where:

- V is a set of elements called “vertices”.
- $E \subseteq V \times V$ is a binary relation. Elements in E are called “edges”.

Note: There are several definitions and variants of graphs.
Graphs are a way to study the relationships among a set of elements.

Example - directed graph

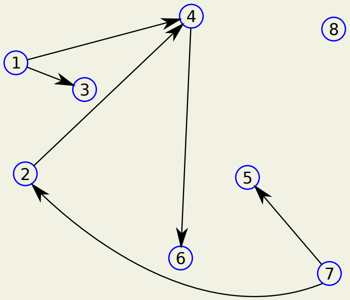
Consider:

$$
\begin{aligned}
V =& \{1, 2, 3, 4, 5, 6, 7, 8\} \\
E =& \{\ (1, 3), (1, 4), \\
& (2, 4), (4, 6), \\
& (7, 2), (7, 5)\ \}
\end{aligned}
$$

Example - directed graph

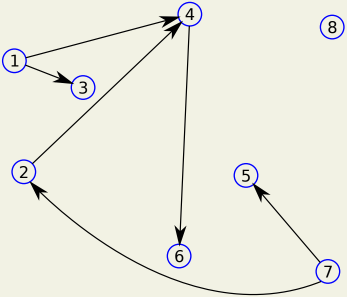
$$V = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$E = \{\ (1,3), (1,4),$$
$$(2,4), (4,6),$$
$$(7,2), (7,5)\ \}$$



Example - directed graph

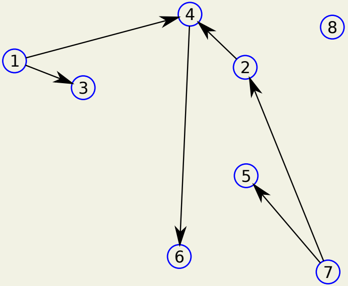
$$
\begin{aligned}
V =& \{1, 2, 3, 4, 5, 6, 7, 8\} \\
E =& \{\ (1, 3), (1, 4), \\
& (2, 4), (4, 6), \\
& (7, 2), (7, 5)\ \}
\end{aligned}
$$



The vertices can be drawn anywhere! The edges are what matter.

Example - directed graph

$$
\begin{aligned}
V =& \{1, 2, 3, 4, 5, 6, 7, 8\} \\
E =& \{\ (1, 3), (1, 4), \\
& \quad (2, 4), (4, 6), \\
& \quad (7, 2), (7, 5)\ \}
\end{aligned}
$$

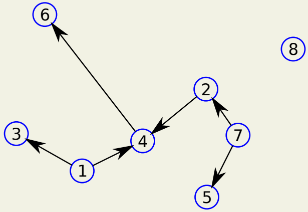


The vertices can be drawn anywhere! The edges are what matter.

Example - directed graph

$$V = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$E = \{\ (1,3), (1,4),$$
$$(2,4), (4,6),$$
$$(7,2), (7,5)\ \ \}$$

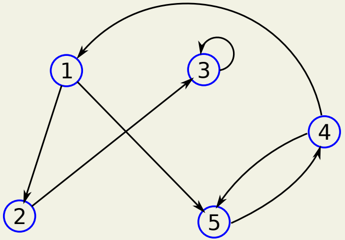


The vertices can be drawn anywhere! The edges are what matter.

Example - directed graph

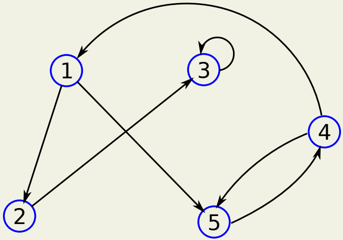
$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{\ (1, 2), (1, 5), (2, 3), (3, 3), (4, 1), (4, 5), (5, 4)\ \}$$



Example - directed graph

$$
\begin{aligned}
V =& \{1, 2, 3, 4, 5\} \\
E =& \{ (1, 2), (1, 5), \\
& (2, 3), (3, 3), \\
& (4, 1), (4, 5), \\
& (5, 4) \}
\end{aligned}
$$



Terminology:

- A vertex v is a *neighbour* or *adjacent* to u if $(u, v) \in E$.
- The neighbourhood $\mathcal{N}(u)$ of a vertex u is the set of all neighbours of u.

Graph

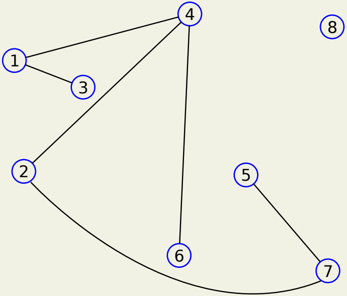
(undirected)

A (undirected) graph G is a two tuple (V, E) where:

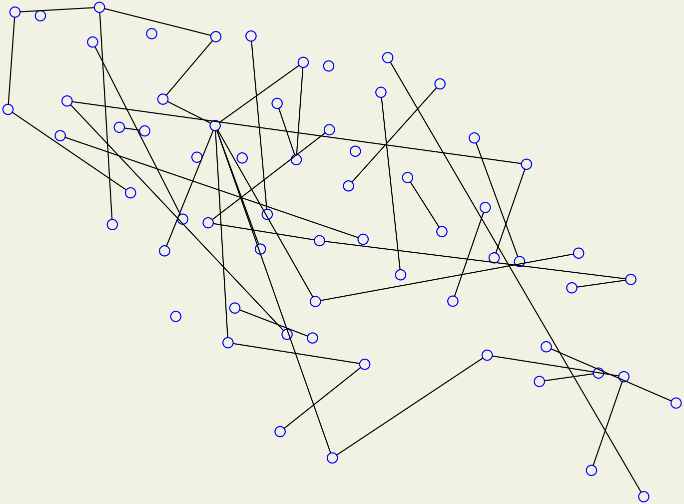
- V is a set of elements called "vertices".
- E is a set of (unordered) pairs of vertices from V.

Example - undirected graph

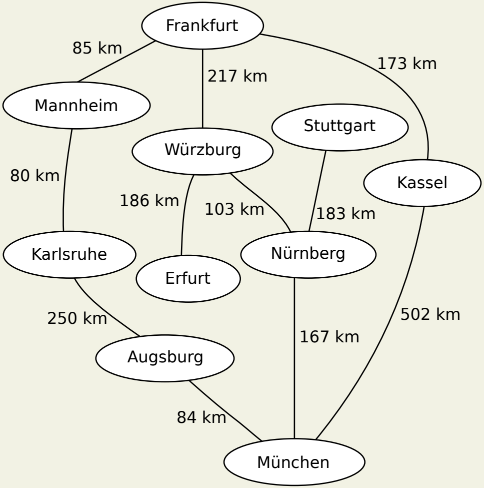
$$
\begin{aligned}
V =& \{1, 2, 3, 4, 5, 6, 7, 8\} \\
E =& \{\ \{1, 3\}, \{1, 4\}, \\
& \{2, 4\}, \{4, 6\}, \\
& \{7, 2\}, \{7, 5\}\ \}
\end{aligned}
$$



Example - undirected graph



Example - undirected graph



(source: wikipedia.org)

Weighted graphs have a *weight* assigned to each edges using a weight function.

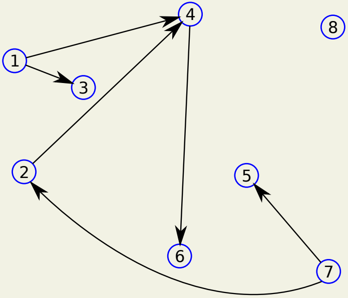
Data structure

Two standard data structures to represent graphs:

- Adjacency matrix
- Adjacency list

Adjacency Matrix

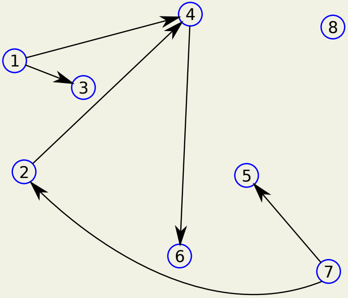
$$\left[\begin{array}{c|cccccccc} A & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$



$$A[u, v] = 1 \iff (u, v) \in E$$

Adjacency Matrix

$$\begin{array}{c|c} A & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \hline \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} & \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

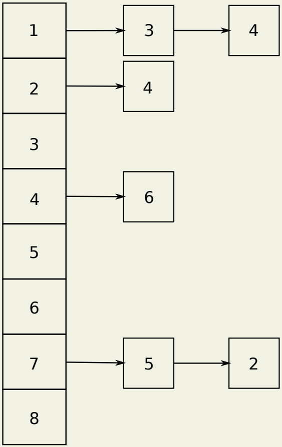


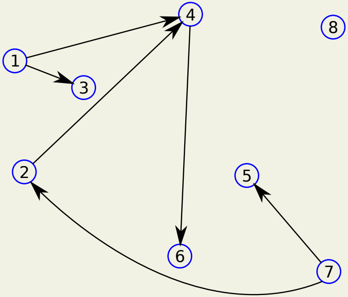
$$A[u, v] = 1 \iff (u, v) \in E$$

For an undirected graph:

- $u, v \in E \iff A[u, v] = A[v, u] = 1$
- The adjacency matrix for an undirected graph is a symmetric matrix

Adjacency Lists





Graph algorithms

Some natural question to ask about an input graph:

- Starting from a vertex s, what vertices are reachable?
- What is the shortest path from a vertex s to a vertex v?

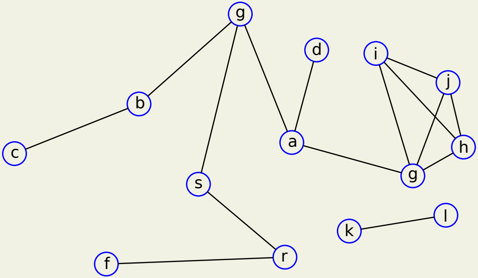
Algorithms that work on an input graph are called graph algorithms. One of the fundamental graph algorithms is the Breadth-first Search.

Breadth-first Search

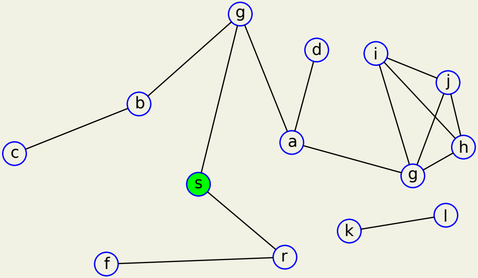
The idea is to explore the graph "radially outward" from the source.

In each step, we expand our exploration by visiting the neighborhood of all explored vertices.

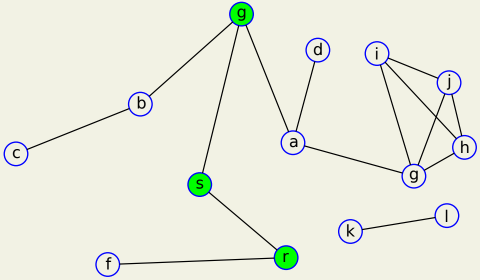
Breadth-first Search (idea)



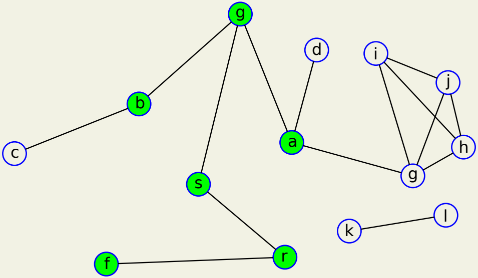
Breadth-first Search (idea)



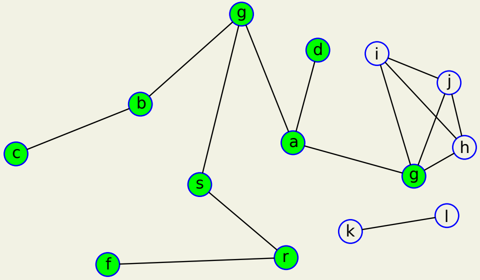
Breadth-first Search (idea)



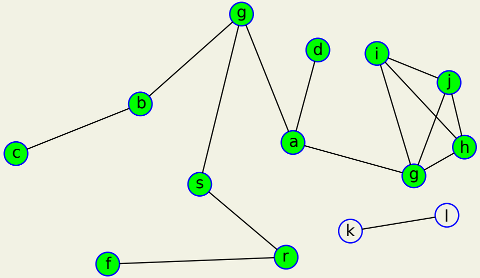
Breadth-first Search (idea)



Breadth-first Search (idea)



Breadth-first Search (idea)



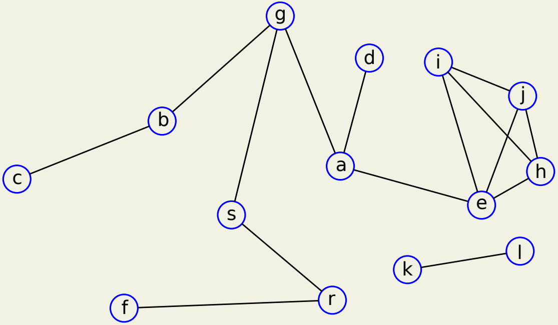
Breadth-first Search (idea)

Important:

- Do not visit an already explored vertex.
- Keep track of distance from source.
- Terminate algorithm when no new vertices can be explored.

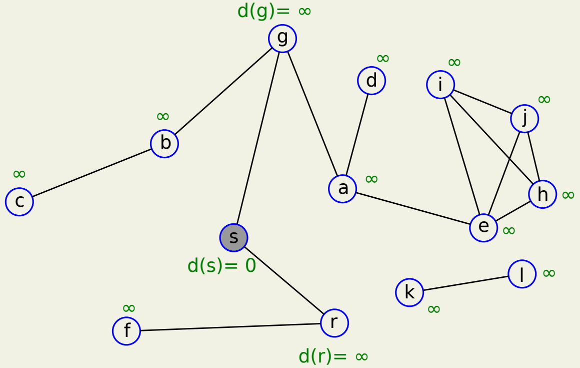
Breadth-first Search

Queue: $\emptyset$



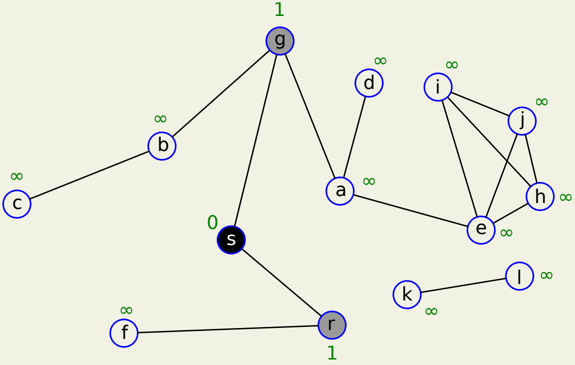
Breadth-first Search

Dequeued vertex: $\boxed{}$ Queue: $\boxed{s}$



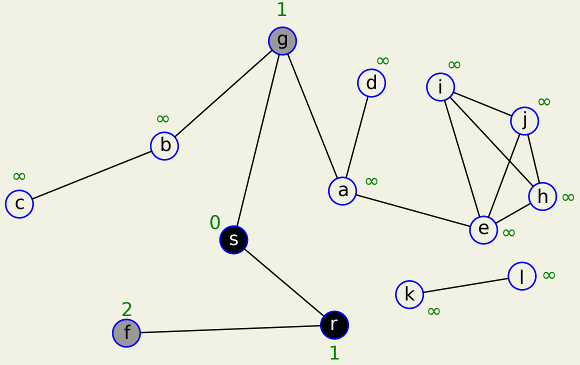
Breadth-first Search

Dequeued vertex: $\boxed{s}$ Queue: $\boxed{r \mid g}$



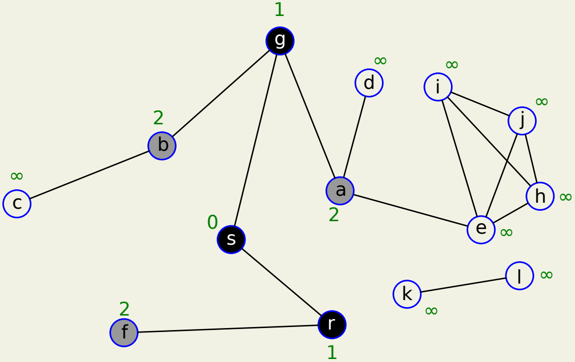
Breadth-first Search

Dequeued vertex: $\boxed{r}$ Queue: $\boxed{g \mid f}$



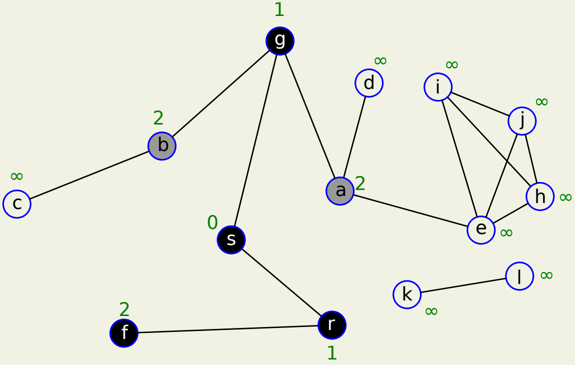
Breadth-first Search

Dequeued vertex: $\boxed{g}$ Queue: $\boxed{f}\,\boxed{a}\,\boxed{b}$



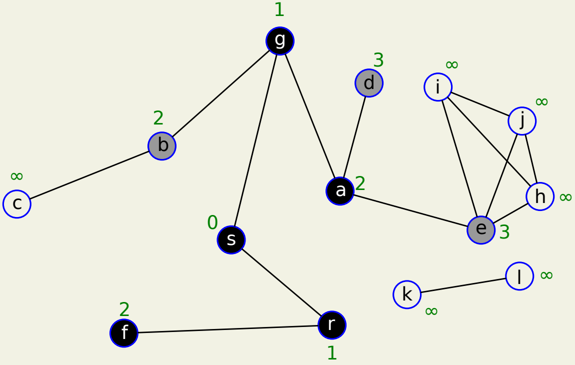
Breadth-first Search

Dequeued vertex: $\boxed{f}$ Queue: $\boxed{a \mid b}$



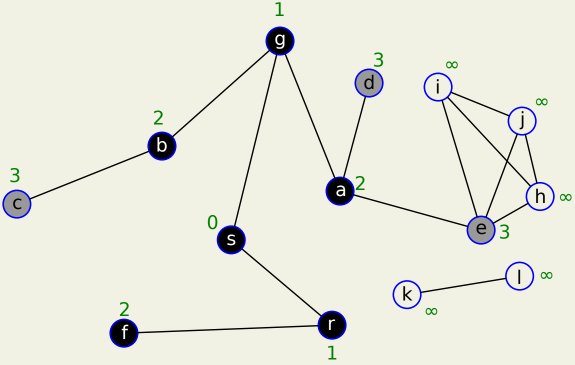
Breadth-first Search

Dequeued vertex: $\boxed{a}$ Queue: $\boxed{b}\boxed{e}\boxed{d}$



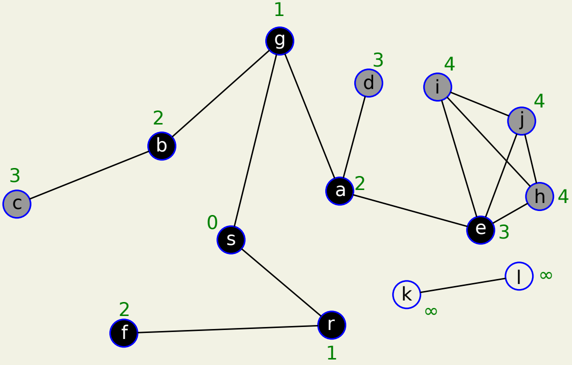
Breadth-first Search

Dequeued vertex: | b | Queue: | e | d | c |



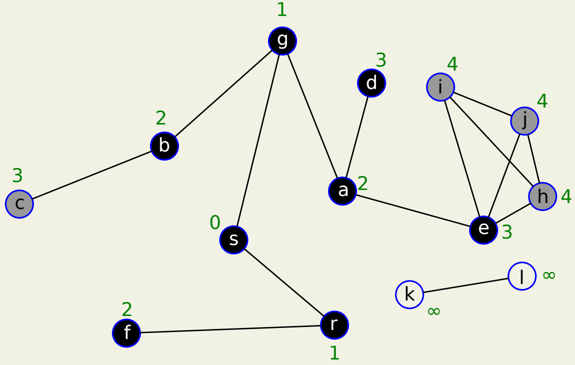
Breadth-first Search

Dequeued vertex: | *e* | Queue: | *d* | *c* | *j* | *h* | *i* |



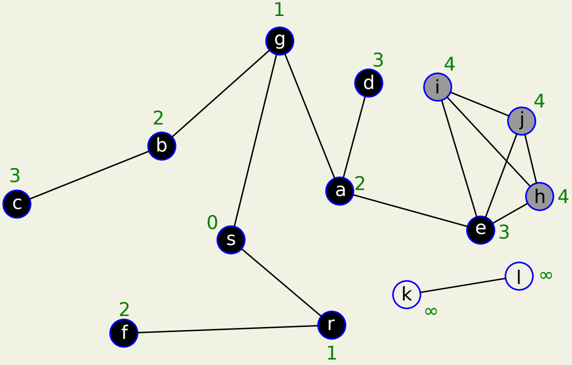
Breadth-first Search

Dequeued vertex: | *d* | Queue: | *c* | *j* | *h* | *i* |



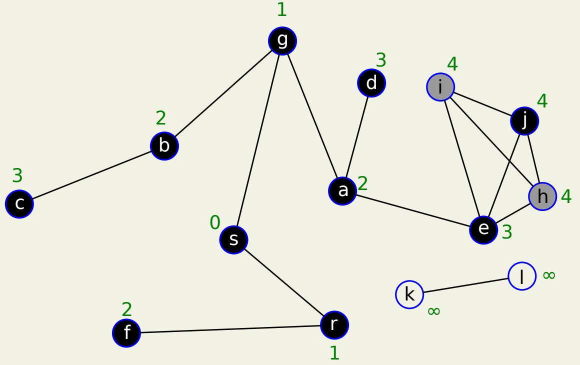
Breadth-first Search

Dequeued vertex: $\boxed{c}$ Queue: $\boxed{j \mid h \mid i}$



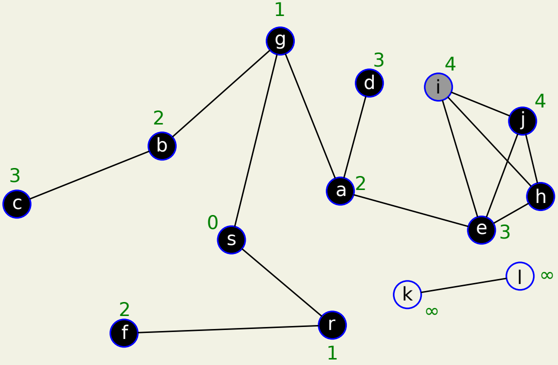
Breadth-first Search

Dequeued vertex: $\boxed{j}$ Queue: $\boxed{h \mid i}$



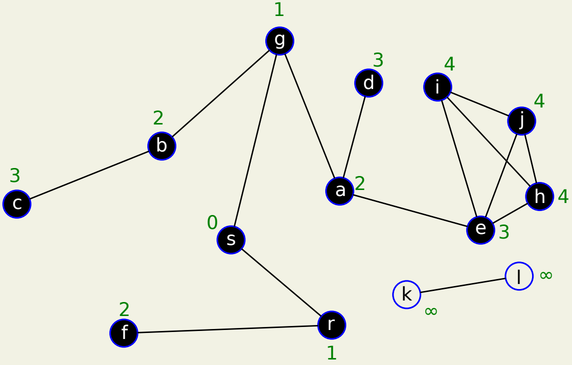
Breadth-first Search

Dequeued vertex: $\boxed{h}$ Queue: $\boxed{i}$



Breadth-first Search

Dequeued vertex: $\boxed{i}$ Queue: $\emptyset$



Algorithm 1 Breadth-first Search from vertex s

Color all vertices WHITE.
For all $u \in V$, $d[u] \leftarrow \infty$, $\pi[u] \leftarrow$ NIL.
$d[s] \leftarrow 0$.
Initialize queue $Q \leftarrow \emptyset$.
ENQUEUE(Q, s)
while $Q \neq \emptyset$ **do**
  $u \leftarrow$ DEQUEUE(Q)
  for each $v \in \mathcal{N}(u)$ **do**
    if color(v) $=$WHITE **then**
      color$[v] \leftarrow$ GRAY
      $d[v] \leftarrow d[u] + 1$
      $\pi[v] \leftarrow u$
      ENQUEUE(Q, v)
    end if
  end for
  color$[u] \leftarrow$ BLACK.
end while

Correctness of BFS

Notation: Let $\delta(s, v)$ denote the minimum number of edges on a path from s to v.

Theorem

Let $G = (V, E)$ be a graph. When BFS is run on G from vertex $s \in V$:

1. Every vertex that is reachable from s gets discovered.
2. On termination, $d[v] = \delta(s, v)$.

Show (1) is an exercise.

Proof of correctness

Proof

Suppose, for the sake of contradiction, (2) does not hold.
Let v be the vertex with smallest $\delta(s, v)$ such that $d[v] \neq \delta(s, v)$.
Claim 1: $d[v] \geq \delta(s, v)$
Let u be the vertex just before v on any path from s to v.
Claim 2: $\delta(s, v) \leq \delta(s, u) + 1$.
Choose a *shortest* path from s to v.
Let the vertex u immmediately precede v
Then $\delta(s, v) = \delta(s, u) + 1 = d[u] + 1$.
So we have:

$$d[v] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1$$

Proof cont...

We have:

$$d[v] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1$$

Consider the time step when u is dequeued.

- Case 1: v was white.
  The algo sets $d[v] = d[u] + 1$.
  This contradicts the eq above.
- Case 2: v is black.
  Then, v was dequeued before u.
  Claim 3: If v was dequeued before u, then $d[v] \leq d[u]$.

Proof cont...

- Case 3: v was gray.
  Vertex v was colored gray after dequeuing some vertex w earlier.
  So $d[v] = d[w] + 1$.
  By Claim 3, $d[w] \leq d[u]$ since w was dequeued before u.
  This gives: $d[v] = d[w] + 1 \leq d[u] + 1$.