

Graphs and BFS

Instructors Subrahmanyam Kalyanasundaram
Karteek Sreenivasaiah

Week 7

Graphs

Abstract Data Type

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:

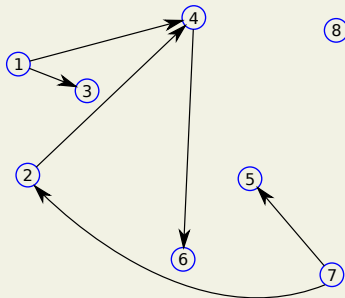
$$V = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

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

Example - directed graph

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

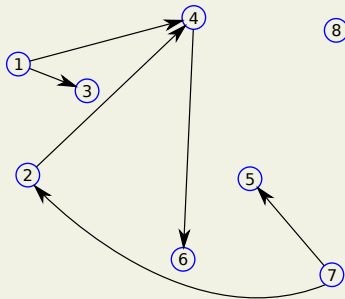
$$E = \{ (1, 3), (1, 4), \\ (2, 4), (4, 6), \\ (7, 2), (7, 5) \}$$



Example - directed graph

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

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

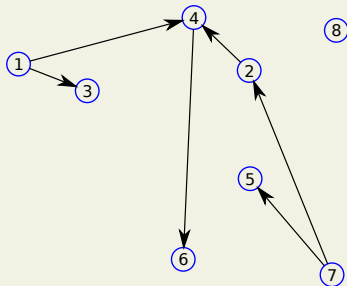


The vertices can be drawn anywhere! The edges are what matter.

Example - directed graph

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

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

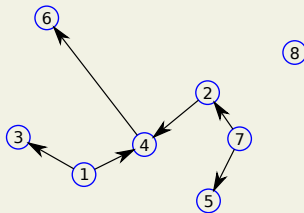


The vertices can be drawn anywhere! The edges are what matter.

Example - directed graph

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

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

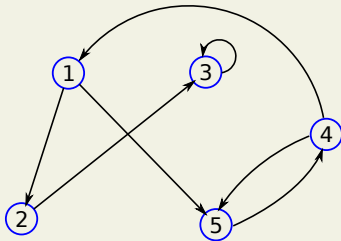


The vertices can be drawn anywhere! The edges are what matter.

Example - directed graph

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

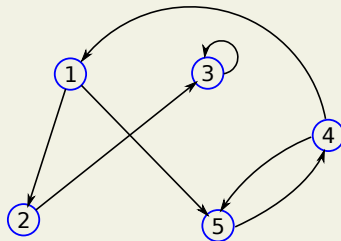
$$E = \{ (1, 2), (1, 5), \\ (2, 3), (3, 3), \\ (4, 1), (4, 5), \\ (5, 4) \}$$



Example - directed graph

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

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



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 .

Graphs

Graph (undirected)

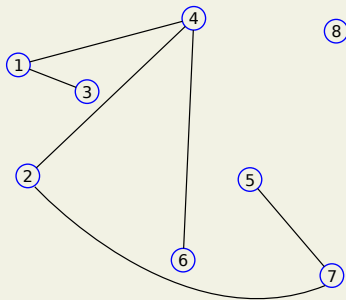
A (undirected) graph G is a two tuple (V, E) where:

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

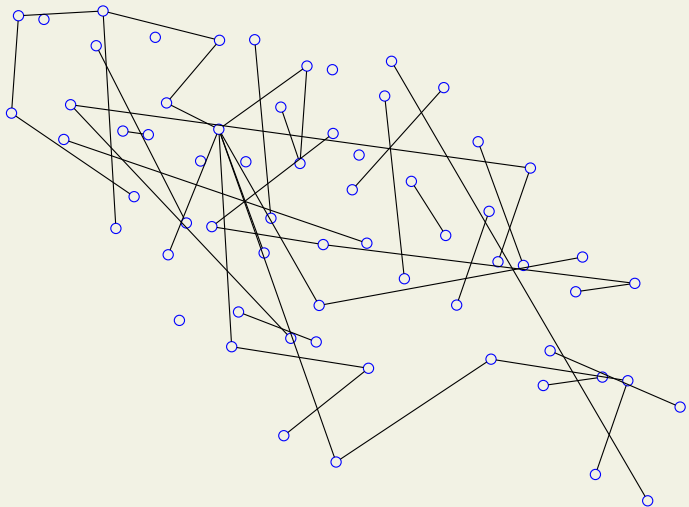
Example - undirected graph

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

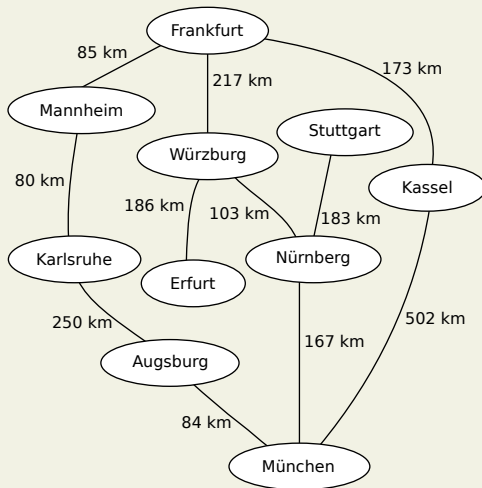
$$E = \{ \{1, 3\}, \{1, 4\}, \\ \{2, 4\}, \{4, 6\}, \\ \{7, 2\}, \{7, 5\} \}$$



Example - undirected graph



Example - undirected graph



(source: wikipedia.org)

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

Definitions

- ▶ **Neighbour of v** : A vertex u such that $(v, u) \in E$ or $\{v, u\} \in E$
- ▶ **Neighbourhood of v ($\mathcal{N}(v)$)**: The set of all neighbours of v
- ▶ **Degree of v** : Cardinality of $\mathcal{N}(v)$

In a directed graph, we can talk about in-neighbours, in-degree, out-neighbours, out-degree etc.

Data structure

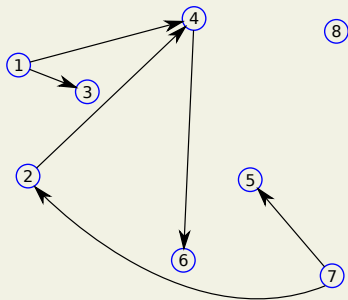
Two standard data structures to represent graphs:

- ▶ Adjacency matrix
- ▶ Adjacency list

Adjacency Matrix

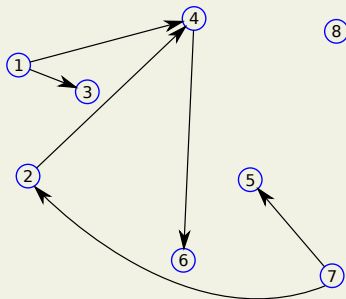
A	1	2	3	4	5	6	7	8
1	0	0	1	1	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	1	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0

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



Adjacency Matrix

A	1	2	3	4	5	6	7	8
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

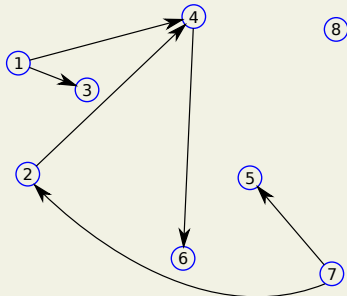
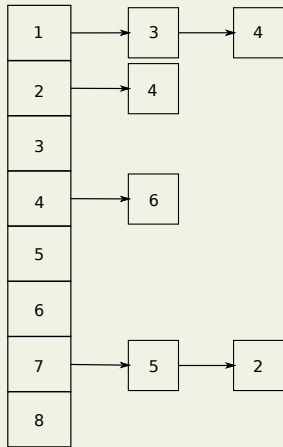


$$A[u, v] = 1 \iff (u, v) \in E$$

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.

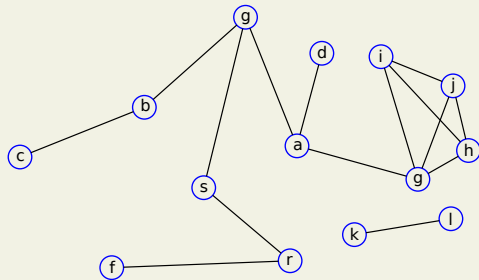
Breadth-first Search

Breadth-first Search

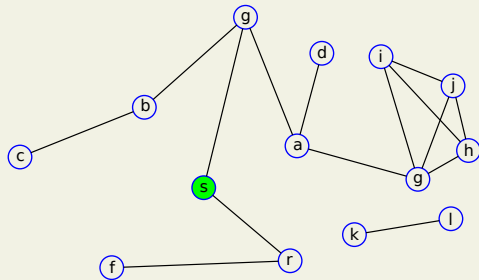
The idea is to explore the graph “radially outward” from the source.

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

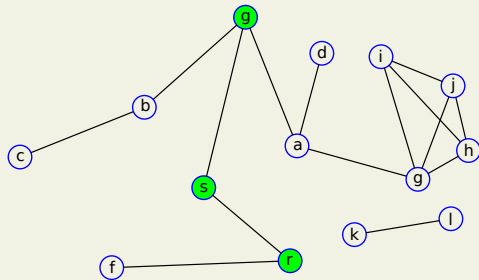
Breadth-first Search (idea)



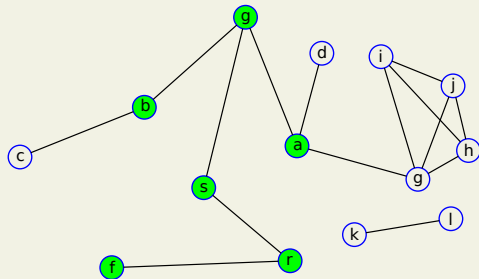
Breadth-first Search (idea)



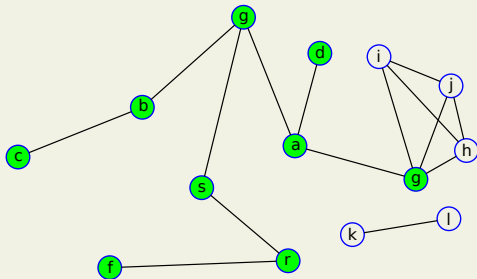
Breadth-first Search (idea)



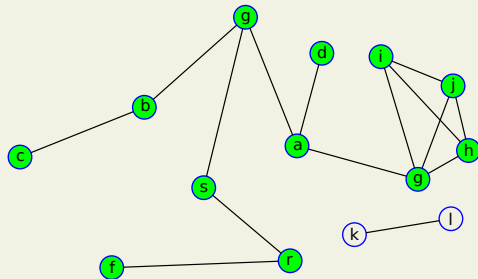
Breadth-first Search (idea)



Breadth-first Search (idea)



Breadth-first Search (idea)



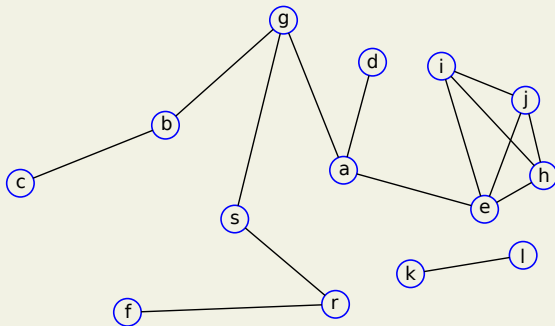
Breadth-first Search (idea)

Important:

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

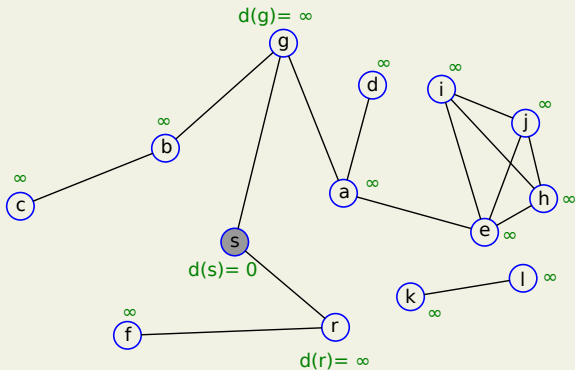
Breadth-first Search

Queue: \emptyset



Breadth-first Search

Dequeued vertex: Queue:



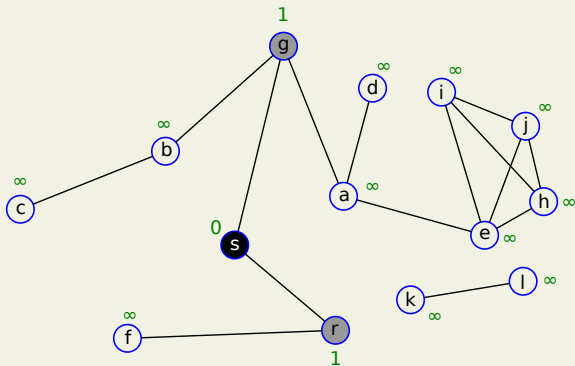
Breadth-first Search

Dequeued vertex:

s

 Queue:

r	g
---	---



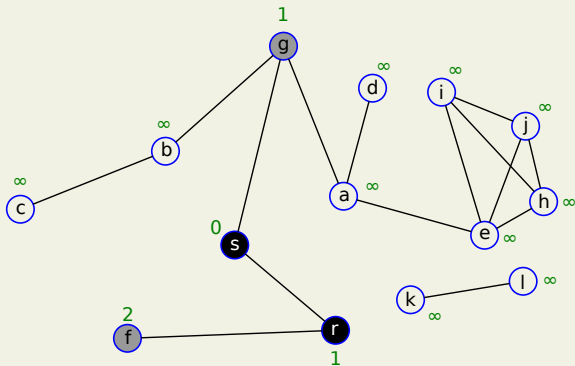
Breadth-first Search

Dequeued vertex:

<i>r</i>

 Queue:

<i>g</i>	<i>f</i>
----------	----------



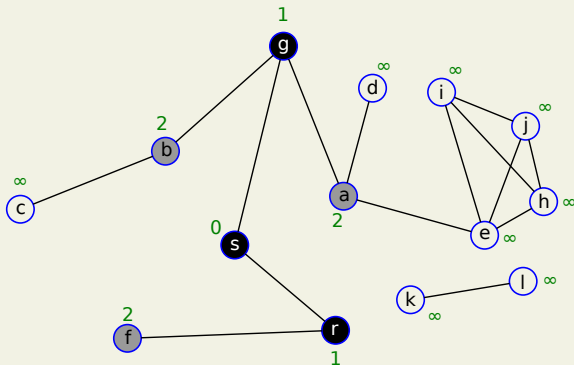
Breadth-first Search

Dequeued vertex:

<i>g</i>

 Queue:

<i>f</i>	<i>a</i>	<i>b</i>
----------	----------	----------



Breadth-first Search

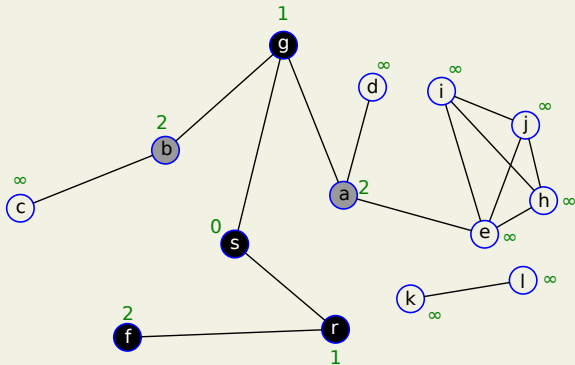
Dequeued vertex:

f

 Queue:

a

b



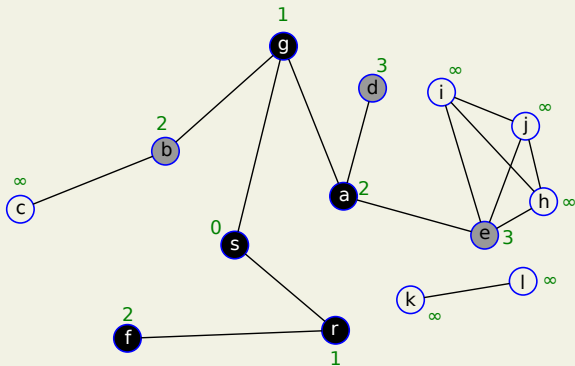
Breadth-first Search

Dequeued vertex:

<i>a</i>

 Queue:

<i>b</i>	<i>e</i>	<i>d</i>
----------	----------	----------



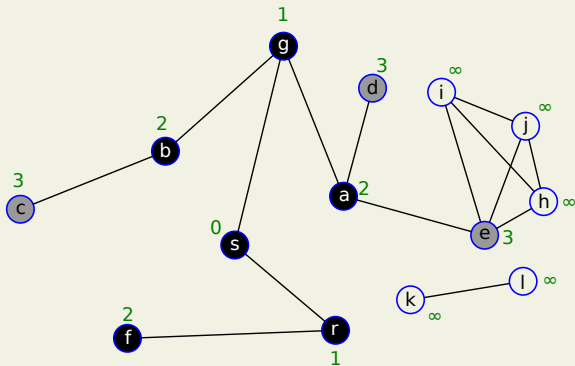
Breadth-first Search

Dequeued vertex:

<i>b</i>

 Queue:

<i>e</i>	<i>d</i>	<i>c</i>
----------	----------	----------



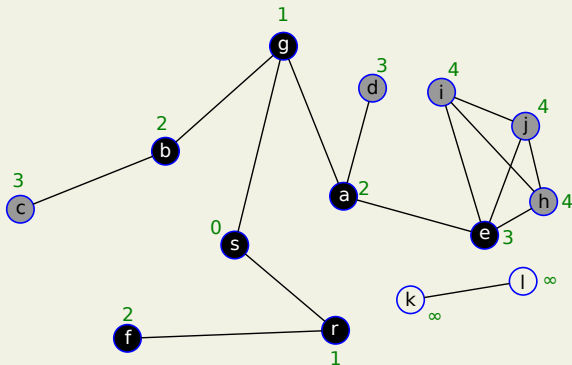
Breadth-first Search

Dequeued vertex:

<i>e</i>

 Queue:

<i>d</i>	<i>c</i>	<i>j</i>	<i>h</i>	<i>i</i>
----------	----------	----------	----------	----------



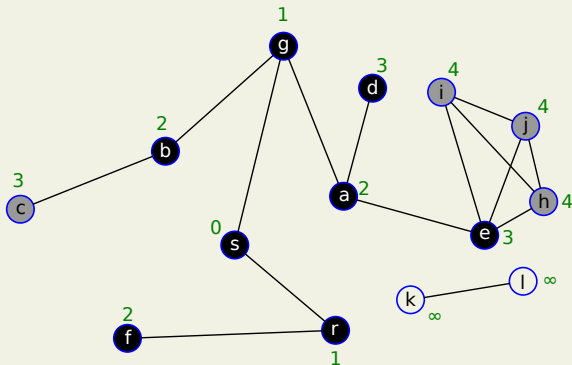
Breadth-first Search

Dequeued vertex:

<i>d</i>

 Queue:

<i>c</i>	<i>j</i>	<i>h</i>	<i>i</i>
----------	----------	----------	----------



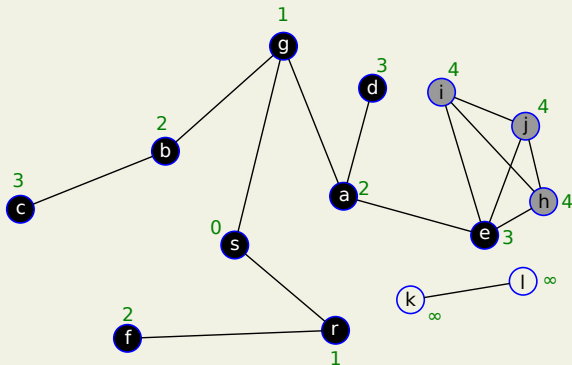
Breadth-first Search

Dequeued vertex:

<i>c</i>

 Queue:

<i>j</i>	<i>h</i>	<i>i</i>
----------	----------	----------



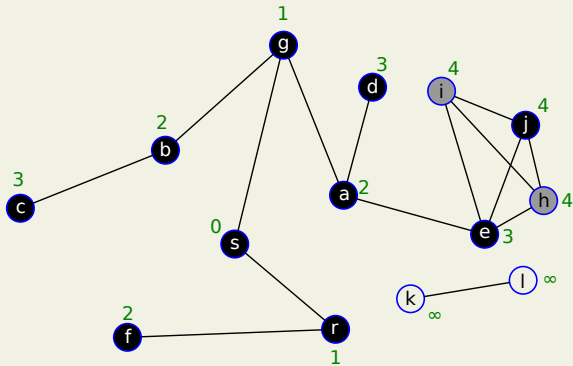
Breadth-first Search

Dequeued vertex:

<i>j</i>

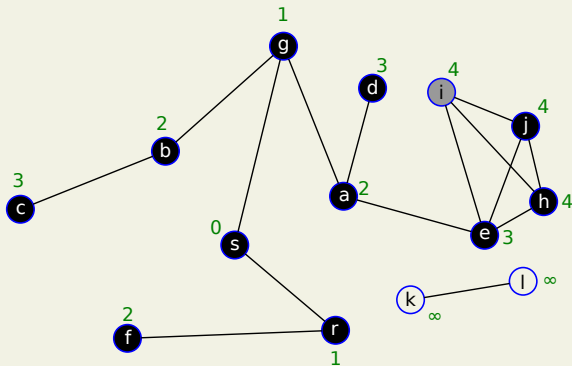
 Queue:

<i>h</i>	<i>i</i>
----------	----------



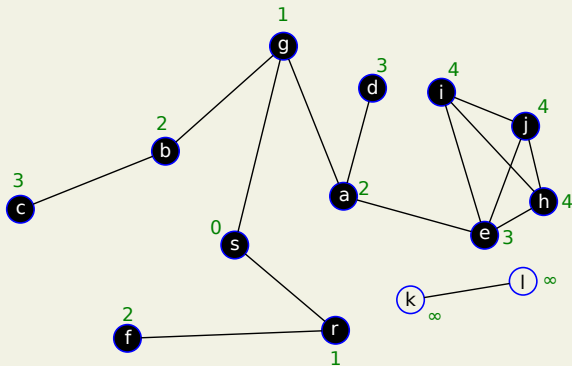
Breadth-first Search

Dequeued vertex: h Queue: i



Breadth-first Search

Dequeued vertex: *i* Queue: \emptyset

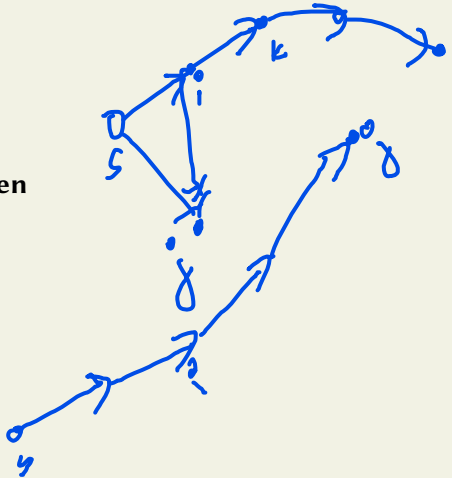


Algorithm 1 Breadth-first Search from vertex s

```

1: Color all vertices WHITE.
2: For all  $u \in V$ ,  $d[u] \leftarrow \infty$ ,  $\pi[u] \leftarrow \text{NIL}$ .
3:  $d[s] \leftarrow 0$ .
4: Initialize queue  $Q \leftarrow \emptyset$ .
5: ENQUEUE( $Q, s$ )
6: while  $Q \neq \emptyset$  do
7:    $u \leftarrow \text{DEQUEUE}(Q)$ 
8:   for each  $v \in \mathcal{N}(u)$  do
9:     if  $\text{color}(v) = \text{WHITE}$  then
10:        $\text{color}[v] \leftarrow \text{GRAY}$ 
11:        $d[v] \leftarrow d[u] + 1$ 
12:        $\pi[v] \leftarrow u$ 
13:       ENQUEUE( $Q, v$ )
14:     end if
15:   end for
16:    $\text{color}[u] \leftarrow \text{BLACK}$ .
17: end while

```



Time Complexity of BFS

- ▶ Each enqueue/dequeue takes $O(1)$ time.
- ▶ Total queue operations take $O(|V|)$ time.
- ▶ Each list in the adj. list is scanned once. This requires total $\Theta(|E|)$. This is assuming the graph is provided using adjacency list.
- ▶ Initialization required $\Theta(|V|)$.
- ▶ Total running time is $O(|V| + |E|)$.

Time Complexity of BFS

- ▶ Each enqueue/dequeue takes $O(1)$ time.
- ▶ Total queue operations take $O(|V|)$ time.
- ▶ Each list in the adj. list is scanned once. This requires total $\Theta(|E|)$. This is assuming the graph is provided using adjacency list.
- ▶ Initialization required $\Theta(|V|)$.
- ▶ Total running time is $O(|V| + |E|)$.
- ▶ **Note:** The colors can be omitted. Instead, check if $d[v] = \infty$

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, $d[v] = \delta(s, v)$ for all v .

We will first show (2).

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: $d[v] \geq \delta(s, v)$

Choose a *shortest* path from s to v .

Let u be the vertex immediately preceding 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 of correctness

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 2: If v was dequeued before u , then $d[v] \leq d[u]$.

Proof of correctness

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 2, $d[w] \leq d[u]$ since w was dequeued before u .

This gives: $d[v] = d[w] + 1 \leq d[u] + 1$.

Exercise

Show (1) using (2). That is, given that $d[v] = \delta(s, v)$, show that every vertex reachable from s gets discovered.

Proof of correctness



Claim 3

Let $(u, v) \in E$. Then we have:

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

Proof

If u is reachable from s , then:

Take the shortest path from s to u . Then take the edge (u, v) .

This gives a path from s to v .

The shortest path from s to v can only be shorter than the above path.



Proof of correctness

Claim 1

$$\forall v \in V, d[v] \geq \delta(s, v)$$

Proof

Induction on the number of enqueue operations.

Hypothesis: same as claim.

Base case: The time when the first vertex enqueued.

The first vertex enqueued is s . At this time we have:

- ▶ $\forall v \in V \setminus \{s\}, d[v] = \infty$
- ▶ $d[s] = \delta(s, s) = 0$.

Hence the claim holds for the base case.

Proof of correctness

Proof

Hypothesis: $\forall v \in V, d[v] \geq \delta(s, v)$

Step: A white (undiscovered) vertex v gets discovered while we are visiting a vertex u with $(u, v) \in E$.

From induction, we have: $d[u] \geq \delta(s, u)$.

The algorithm assigns $d[v] \leftarrow d[u] + 1$. So:

$$\begin{aligned} d[v] &= d[u] + 1 \\ &\geq \delta(s, u) + 1 \\ &\geq \delta(s, v) \end{aligned}$$

Last inequality follows from Claim 3.



Proof of correctness

Claim 2

If v was dequeued before u , then $d[v] \leq d[u]$.

We will show a stronger claim:



Claim 4

If at some point, the queue contained v_1, v_2, \dots, v_r where v_1 was the head. Then:

- (a) $d[v_1] \leq d[v_2] \leq \dots \leq d[v_r]$
- (b) $d[v_r] \leq d[v_1] + 1$

Proof of Claim 2:

Write down vertices in the order they went through the queue.

By claim 4 (a), the calculated d values for them are non-decreasing.

Vertex v will appear before u in this order.

Hence claim 2 follows. \square

Proof of correctness

Claim 4

If queue contains v_1, v_2, \dots, v_r where v_1 is the head. Then:

- (a) $d[v_1] \leq d[v_2] \leq \dots \leq d[v_r]$
- (b) $d[v_r] \leq d[v_1] + 1$

Proof

Induction on number of queue operations.

Hypothesis: Same as claim. We show that the claim holds after every enqueue and dequeue.

Base case: The first queue operation - enqueueing s .
The claim trivially holds.

Proof of correctness

Claim 4

If queue contains v_1, v_2, \dots, v_r where v_1 is the head. Then:

(a) $d[v_1] \leq d[v_2] \leq \dots \leq d[v_r]$

(b) $d[v_r] \leq d[v_1] + 1$

Proof

Step:

- **Dequeue:** After v_1 is dequeued, v_2 is the new head.

Part (a): From induction,

$$d[v_1] \leq d[v_2] \leq d[v_3] \leq \dots \leq d[v_r].$$

Hence (a) holds.

Part (b): From induction, $d[v_r] \leq d[v_1] + 1$. And so:

$$\begin{aligned} d[v_r] &\leq d[v_1] + 1 \\ &\leq d[v_2] + 1 \end{aligned}$$

Proof of correctness

$$d[v] = d[u] + 1 \leq d[v] + 1$$

Proof

- ▶ **Enqueue:** When a vertex v is enqueued:

It was enqueued because:

- ▶ it was undiscovered so far.
- ▶ it was present in the adjacency list of a vertex u that was just dequeued.

Since u was the previous head of the list, from induction we have:

- ▶ $d[u] \leq d[v_1] \leq d[v_2] \leq \dots \leq d[v_r]$.
- ▶ $d[v_r] \leq d[u] + 1$.

We assign $d[v] \leftarrow d[u] + 1$ and then enqueue v . Hence, we have:

- ▶ $d[v_r] \leq d[u] + 1 = d[v]$
- ▶ $d[v_1] \leq d[v_2] \leq \dots \leq d[v_r] \leq d[v]$.



Loop Invariant

Claim 4

If queue contains v_1, v_2, \dots, v_r where v_1 is the head. Then:

- (a) $d[v_1] \leq d[v_2] \leq \dots \leq d[v_r]$
- (b) $d[v_r] \leq d[v_1] + 1$

Claim 4 is actually a loop invariant!

Another loop invariant

The queue Q consists of the set of GRAY vertices.