Graph Algorithms

Graph Traversals and Connectivity

Michael Lampis

September 3, 2025

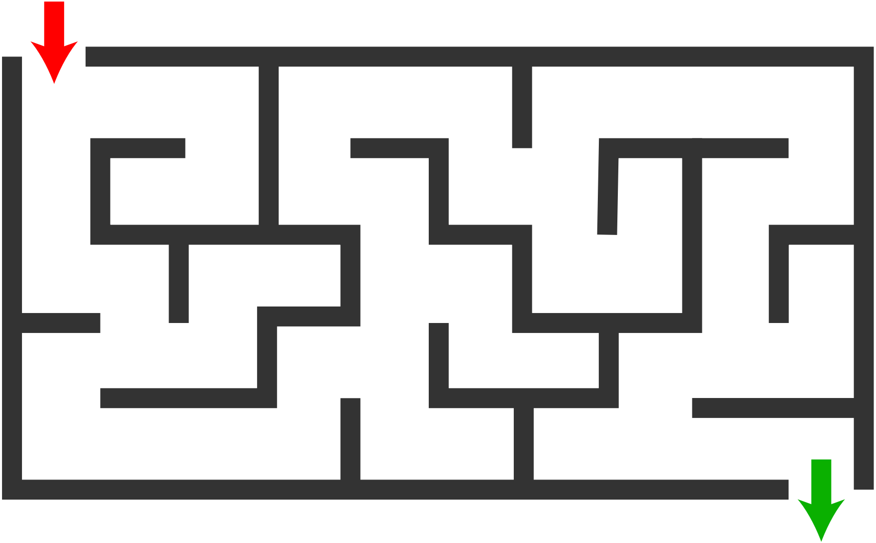
Graph Traversal

Problem

Given (di)graph G, determine connectivity properties:

- Is G (strongly) connected?
- What are the (strongly) connected components of G?
- Which vertices can be reached from a given source s?
- What is the shortest path distance from (given vertex) s to (given vertex) t?

Graph Traversal



Breadth-First Search

Breadth-first search (BFS) is a basic **graph traversal** algorithm.

- Input: G and specified source vertex s
- Output: Set of vertices reachable from s and tree of shortest paths from s to all such vertices.

Breadth-First Search

Breadth-first search (BFS) is a basic **graph traversal** algorithm.

- Input: G and specified source vertex s
- Output: Set of vertices reachable from s and tree of shortest paths from s to all such vertices.

Key properties:

- Linear time and space complexity $O(n + m)$.
- Works for both graphs and digraphs.

Breadth-First Search

Breadth-first search (BFS) is a basic **graph traversal** algorithm.

- Input: G and specified source vertex s
- Output: Set of vertices reachable from s and tree of shortest paths from s to all such vertices.

Key properties:

- Linear time and space complexity $O(n + m)$.
- Works for both graphs and digraphs.

Key idea:

- Explore vertices in order of increasing distance from s, using a queue.

How NOT to solve graph traversal!

How NOT to solve graph traversal!

Attempt to decide reachability from s to t:

```
procedure REACH(G, s, t)
    if s = t or st ∈ E then                ▷ Base Case
        Return Yes
    end if
    for u ∈ N(s) do                        ▷ Inductive Case
        if Reach(G, u, t) then
            Return Yes
        end if
    end for
    Return No
end procedure
```

How NOT to solve graph traversal!

Counterexample:

- Suppose $G = K_3 + K_1$, t is the isolated vertex, s is in the triangle.
- Algorithm goes into infinite loop!

How NOT to solve graph traversal!

Attempt to decide reachability from s to t in k steps:

```
procedure Reach(G, s, t, k)
    if k < 0 then
        Return No
    end if
    if s = t or st ∈ E then                         ▷ Base Case
        Return Yes
    end if
    for u ∈ N(s) do                                 ▷ Inductive Case
        if Reach(G, u, t, k − 1) then
            Return Yes
        end if
    end for
    Return No
end procedure
```

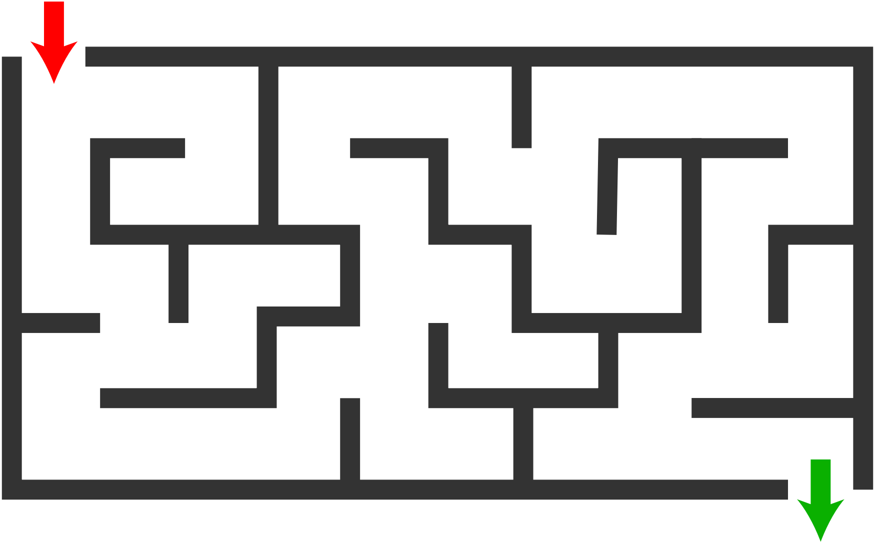
How NOT to solve graph traversal!

Counterexample:

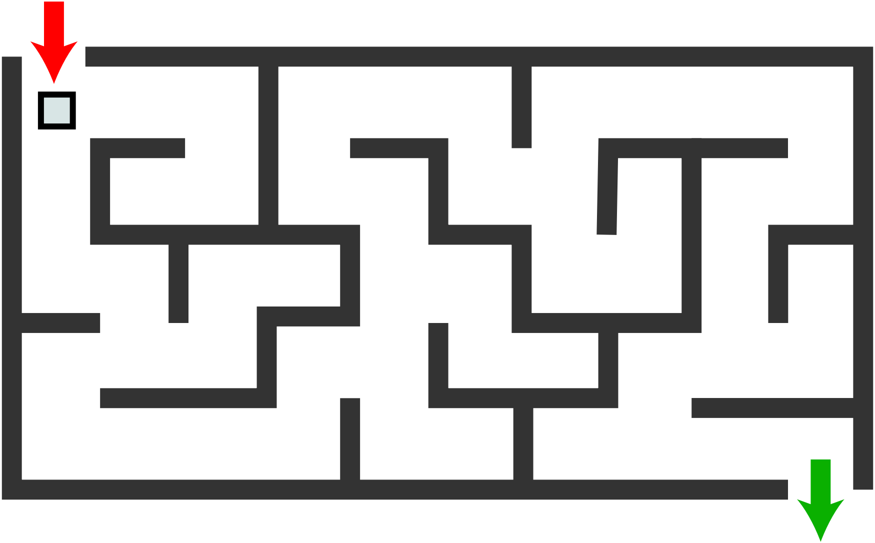
Algorithm is correct, but...

- Execution can have complexity **exponential** in n!
- Notice that we are using very little space...

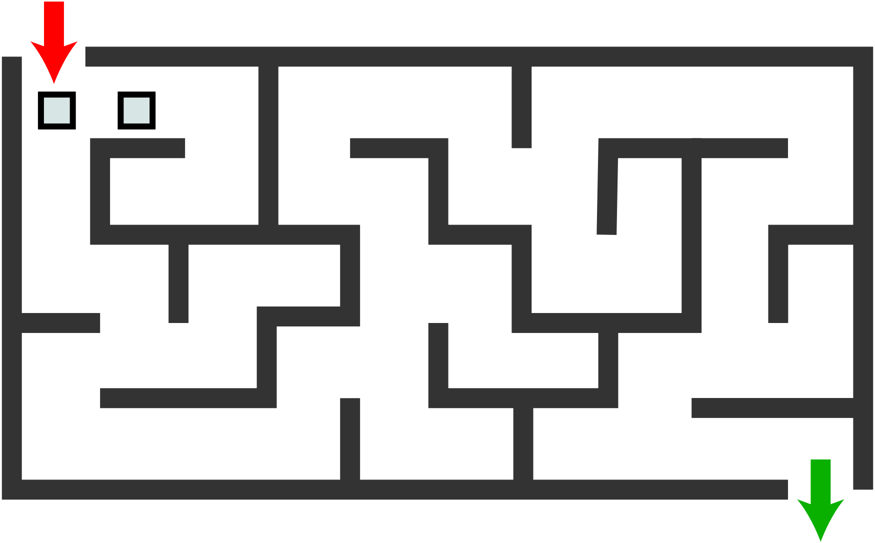
BFS (partial) example



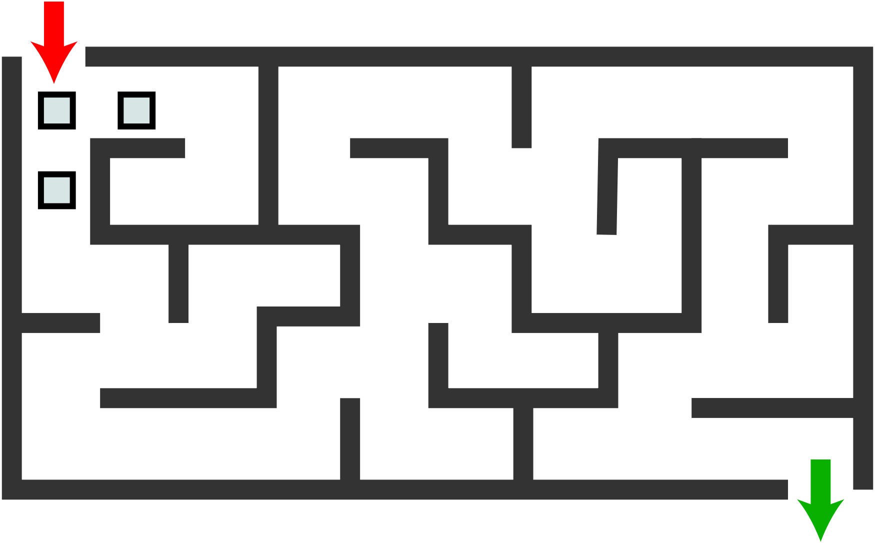
BFS (partial) example



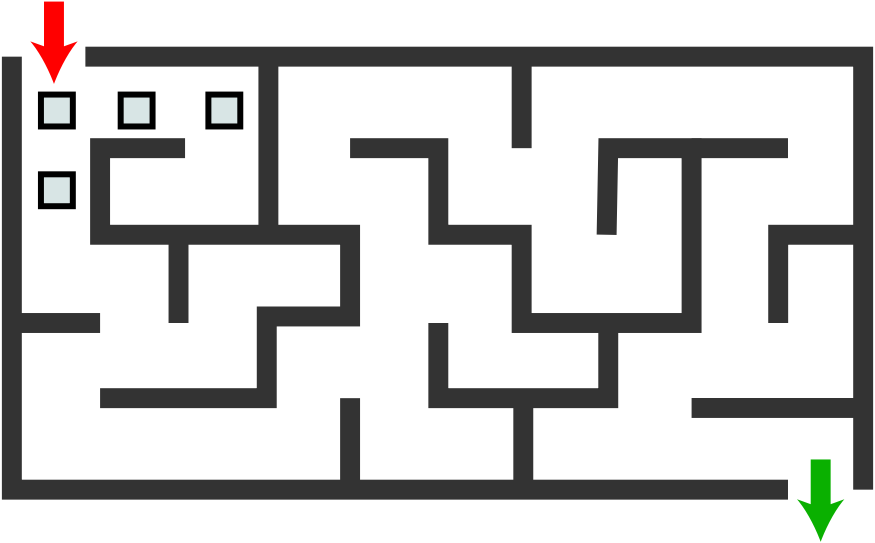
BFS (partial) example



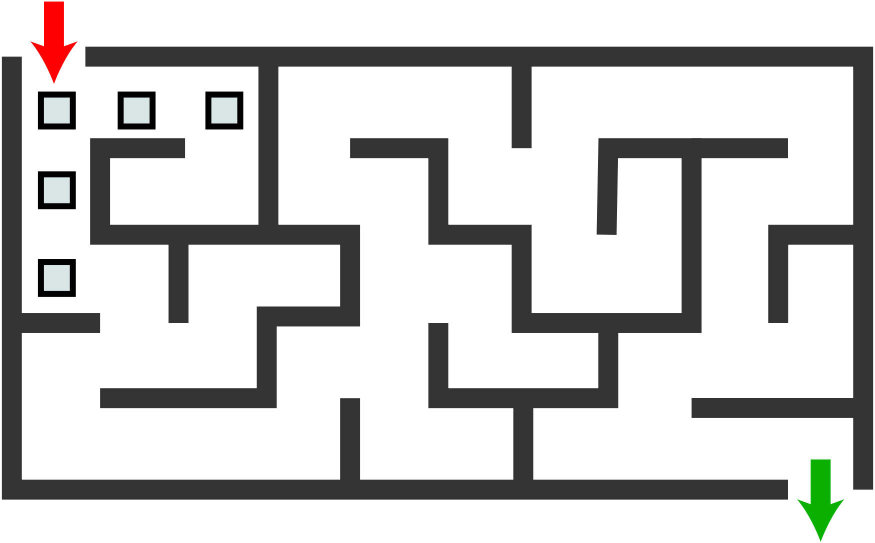
BFS (partial) example



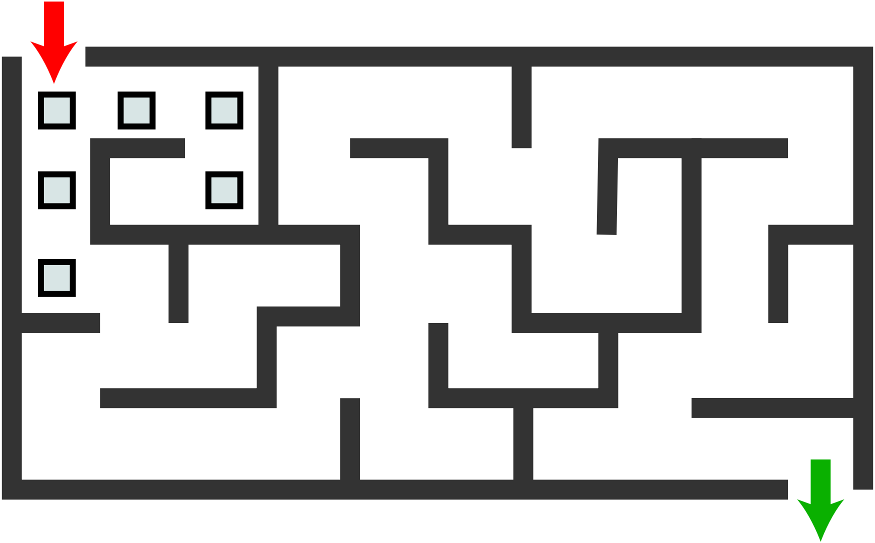
BFS (partial) example



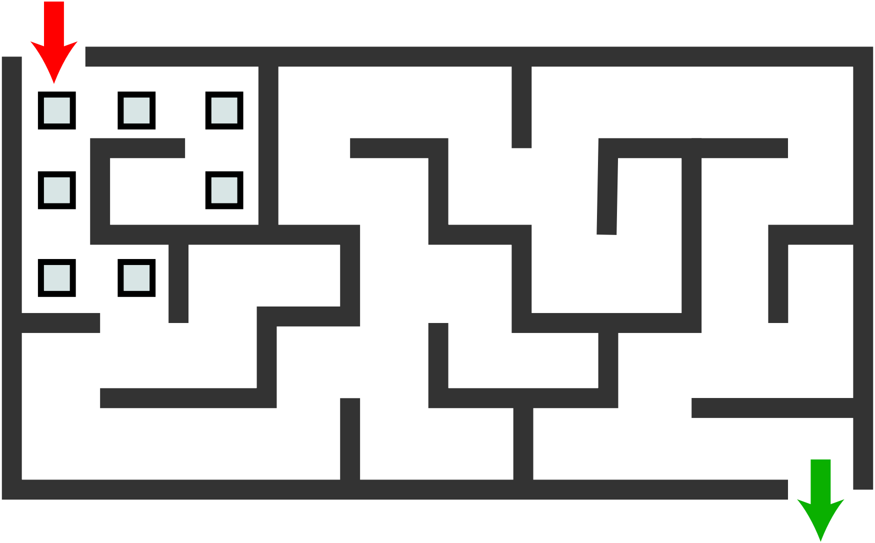
BFS (partial) example



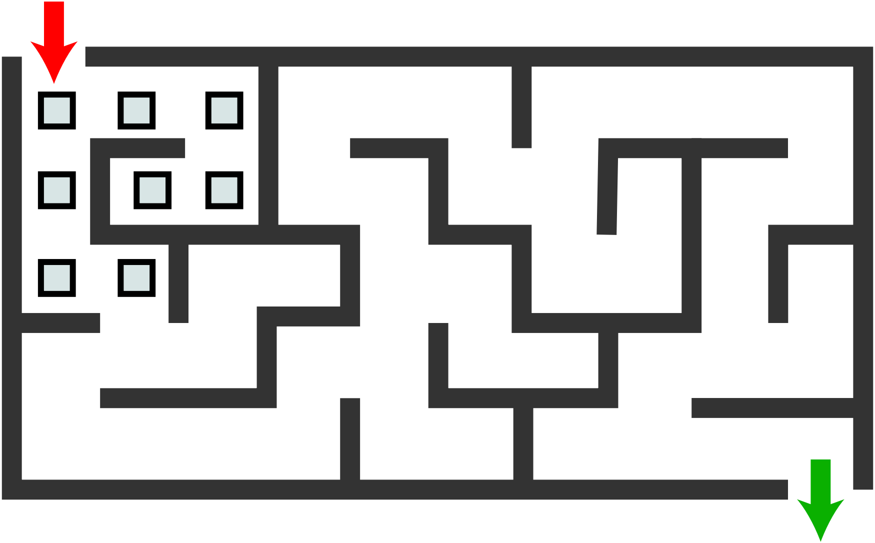
BFS (partial) example



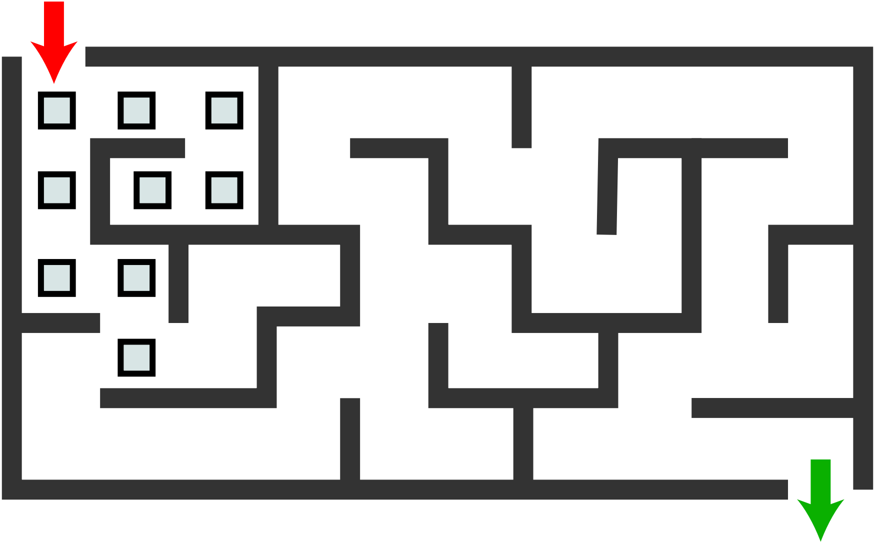
BFS (partial) example



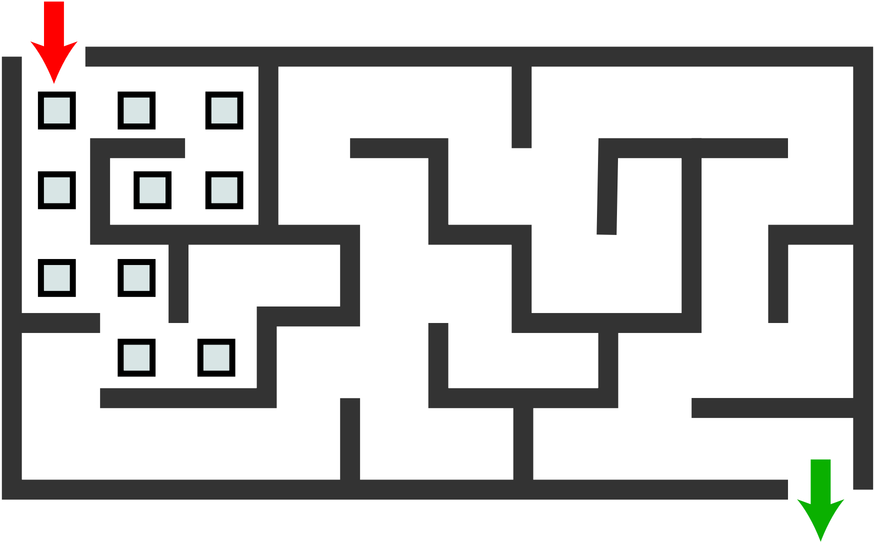
BFS (partial) example



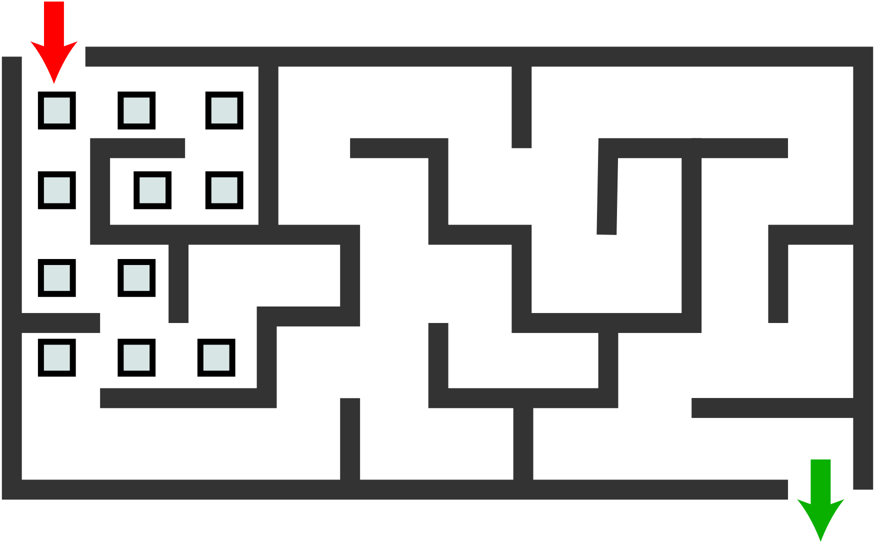
BFS (partial) example



BFS (partial) example



BFS (partial) example



Reminder: Queues

Queue (FIFO)

Queue specifications:

- Two operations: Enqueue and Dequeue.
- Enqueue: Adds an element to the data structure.
- Dequeue: Removes and returns the **oldest** element of the data structure.
- Complexity: $O(1)$ for both operations.

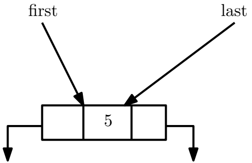
Queue example – Lists

1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\to$ 5
3. Enqueue(4)
4. Dequeue $\to$ 9

Queue example – Lists

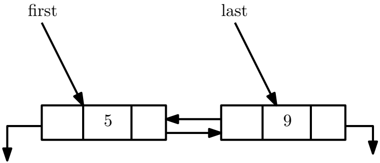
1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Lists



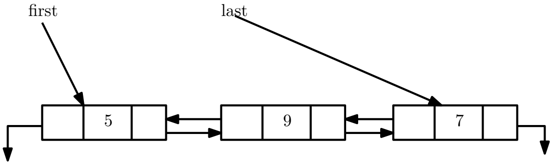
1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Lists



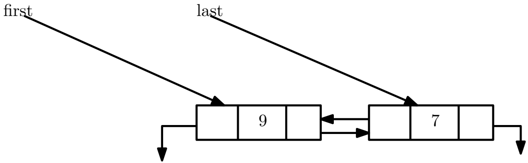
1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Lists



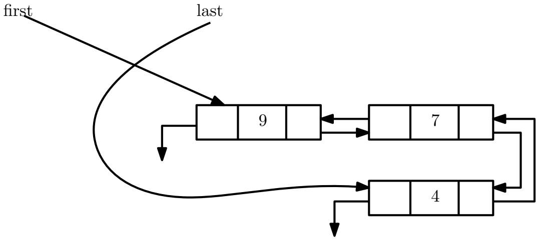
1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Lists



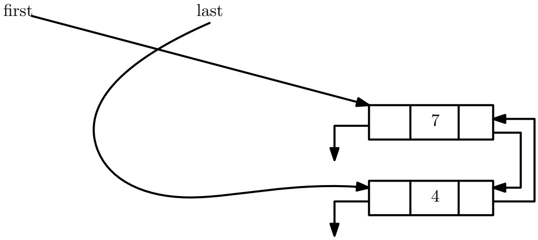
1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Lists



1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue → 5
3. Enqueue(4)
4. Dequeue → 9

Queue example – Lists



1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue → 5
3. Enqueue(4)
4. Dequeue → 9

Queue example – Arrays

1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Arrays

*	*	*	*	*	*	*	*	...

first=-1, last=-1

1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Arrays

5	*	*	*	*	*	*	*	...

first=1, last=1

1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Arrays

5	9	*	*	*	*	*	*	...

first=1, last=2

1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Arrays

5	9	7	*	*	*	*	*	...

first=1, last=3

1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue → 5
3. Enqueue(4)
4. Dequeue → 9

Queue example – Arrays

*	9	7	*	*	*	*	*	...

first=2, last=3

1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Arrays

*	9	7	4	*	*	*	*	...

first=2, last=4

1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

Queue example – Arrays

*	*	7	4	*	*	*	*	...

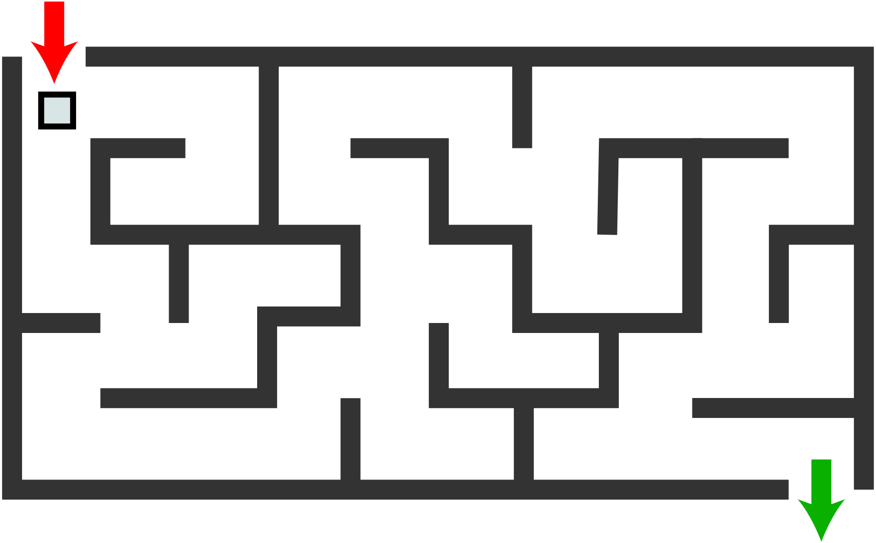
first=3, last=4

1. Enqueue(5), Enqueue(9), Enqueue(7)
2. Dequeue $\rightarrow$ 5
3. Enqueue(4)
4. Dequeue $\rightarrow$ 9

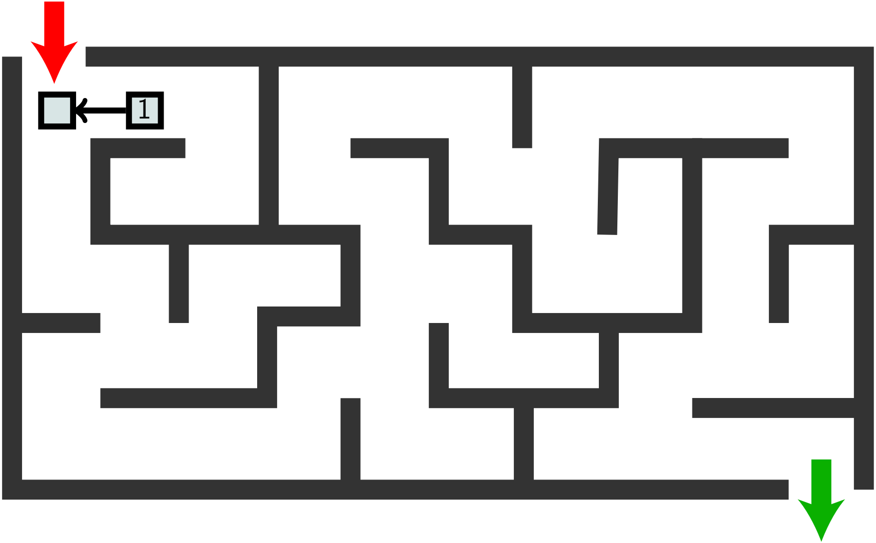
BFS Trees

- The output of BFS is a **tree** rooted at s.
- For each vertex v of the tree we calculate
  - The shortest-path distance from s to v.
  - The predecessor (parent) of v is a shortest path from s to v.

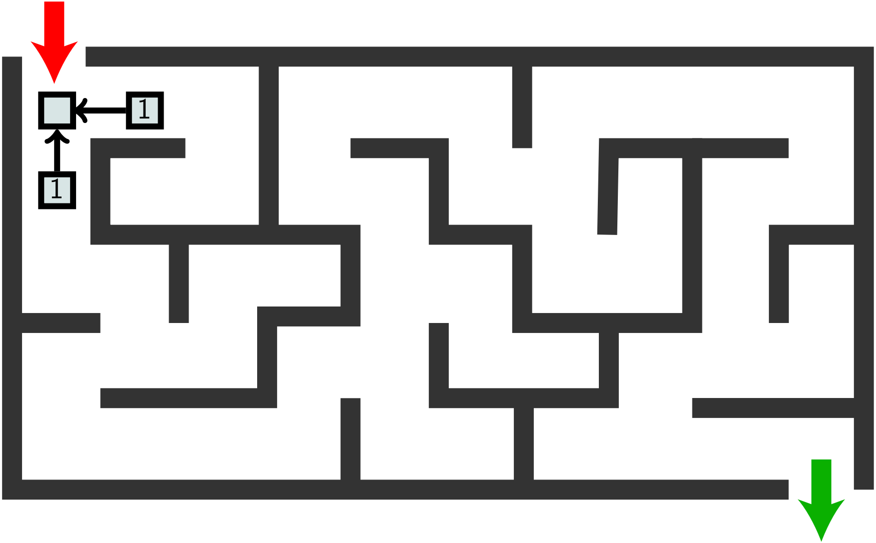
BFS Trees



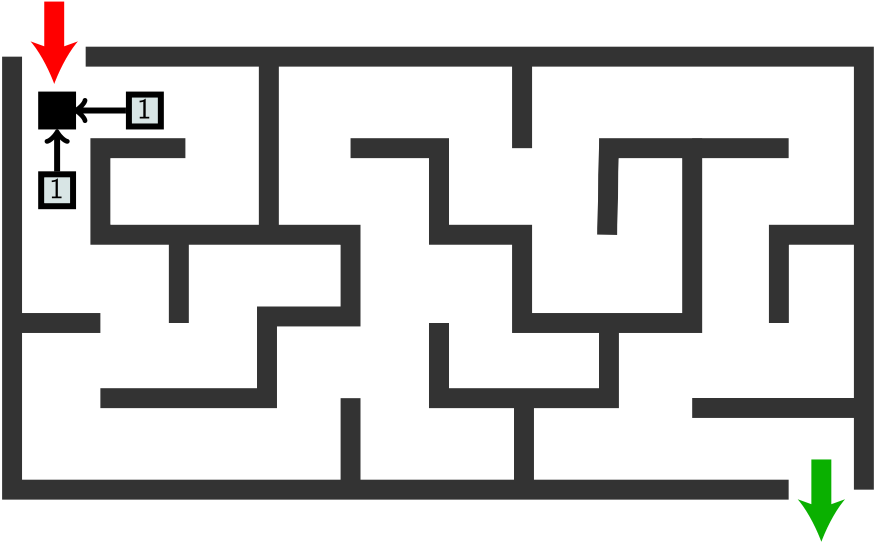
BFS Trees



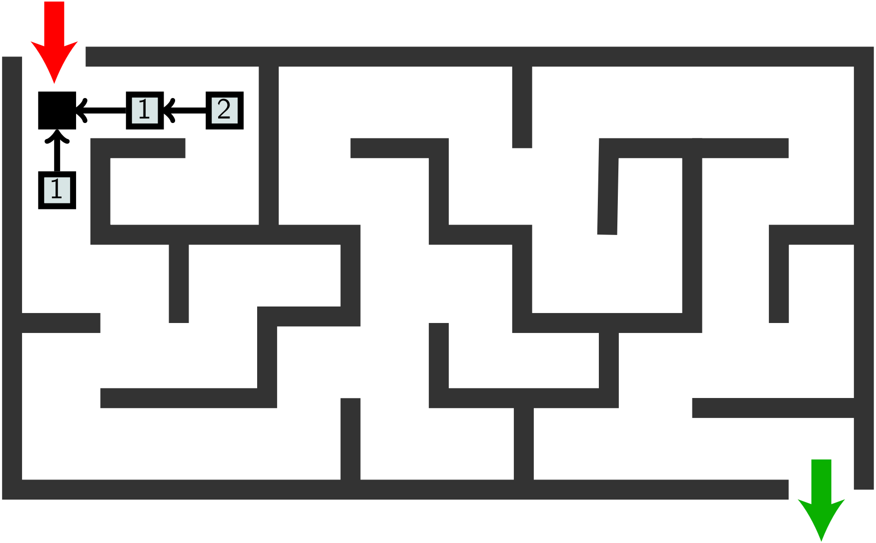
BFS Trees



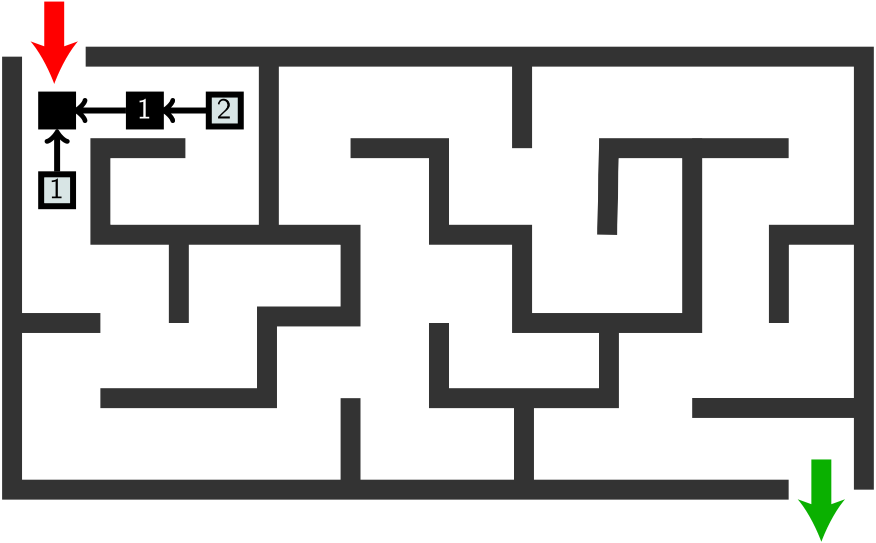
BFS Trees



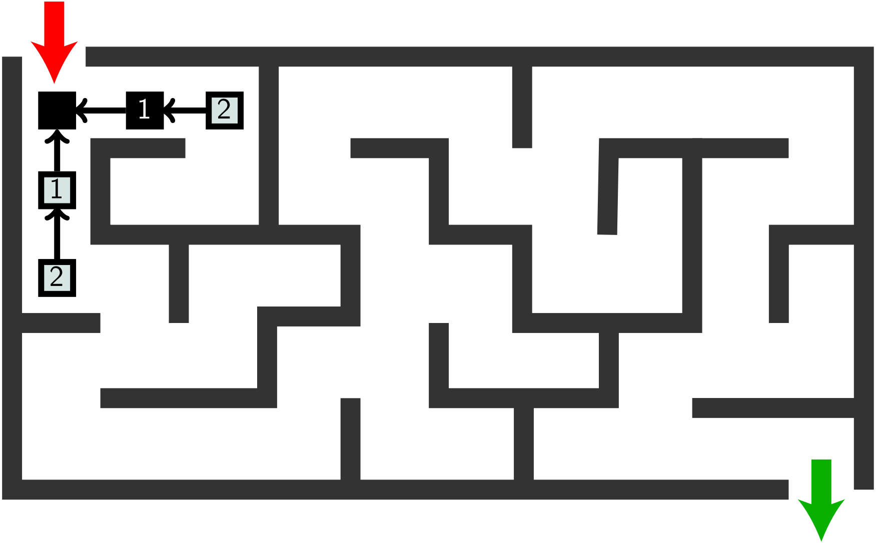
BFS Trees



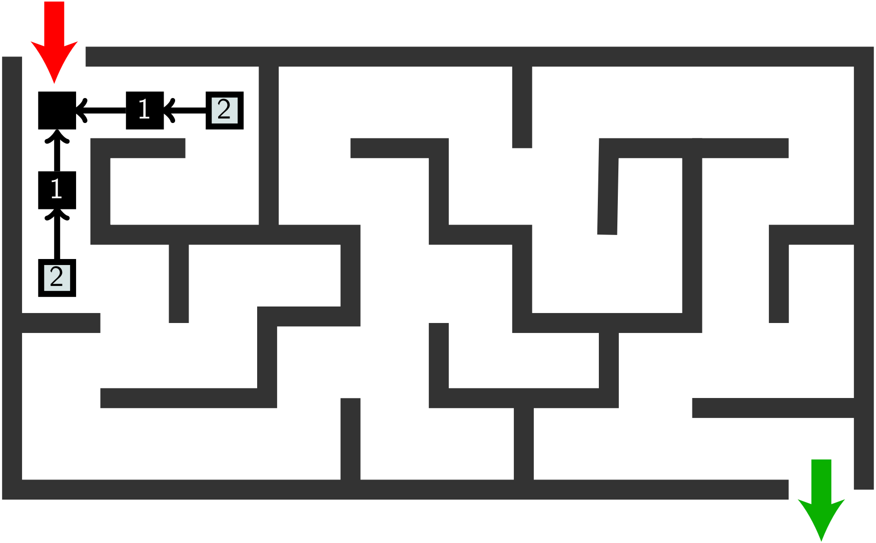
BFS Trees



BFS Trees



BFS Trees



The algorithm

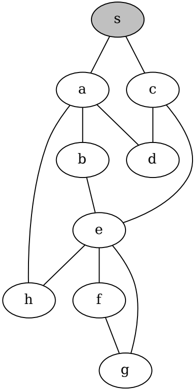
BFS

```
for $v \in V \setminus \{s\}$ do                                        ▷ initialize
    Color $v$ White, Parent of $v \leftarrow$ NULL, $\mathrm{dist}(s, v) \leftarrow \infty$
end for
$\mathrm{dist}(s, s) \leftarrow 0$, Parent of $s \leftarrow$ NULL, Color $s$ Gray
Enqueue($s$)                                                            ▷ end of initialization
while Queue not empty do                                                ▷ main loop
    $v \leftarrow$ Dequeue
    for each $u \in N(v)$ do
        if $u$ is White then
            $\mathrm{dist}(s, u) \leftarrow \mathrm{dist}(s, v) + 1$, $v$ becomes parent of $u$
            Color $u$ Gray
            Enqueue($u$)
        end if
    end for
    Color $v$ Black
end while
```

BFS High-level ideas

- Vertices are colored White, Gray, or Black
  - White: undiscovered
  - Gray: discovered, not processed yet
  - Black: finished
- As vertices are discovered, they are added to the queue
- FIFO$\rightarrow$ vertices further away are processed later (proof?)

Example



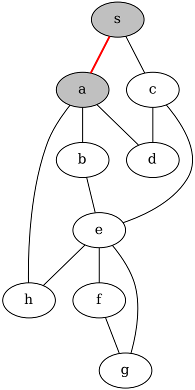
Distances:

a	b	c	d	e	f	g	h
∞	∞	∞	∞	∞	∞	∞	∞

Queue:

s

Example



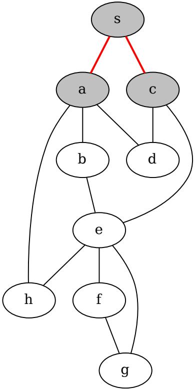
Distances:

a	b	c	d	e	f	g	h
1	∞	∞	∞	∞	∞	∞	∞

Queue:

a

Example



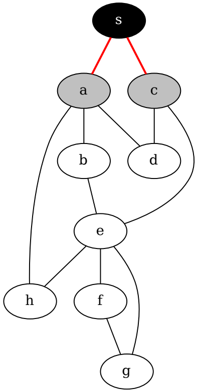
Distances:

a	b	c	d	e	f	g	h
1	∞	1	∞	∞	∞	∞	∞

Queue:

a, c

Example



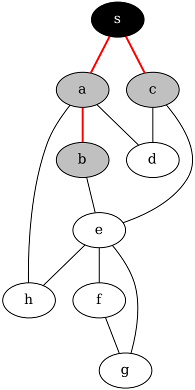
Distances:

a	b	c	d	e	f	g	h
1	∞	1	∞	∞	∞	∞	∞

Queue:

a, c

Example



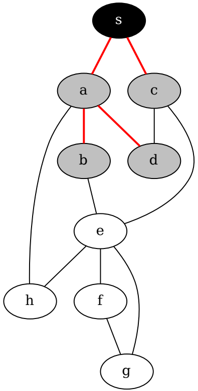
Distances:

a	b	c	d	e	f	g	h
1	2	1	∞	∞	∞	∞	∞

Queue:

c, b

Example



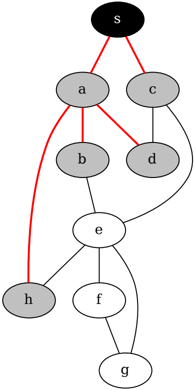
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	∞	∞	∞	∞

Queue:

c, b, d

Example



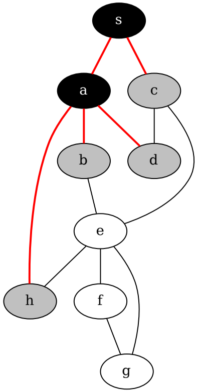
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	∞	∞	∞	2

Queue:

c, b, d, h

Example



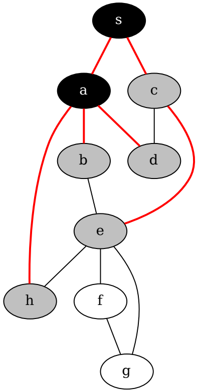
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	∞	∞	∞	2

Queue:

c, b, d, h

Example



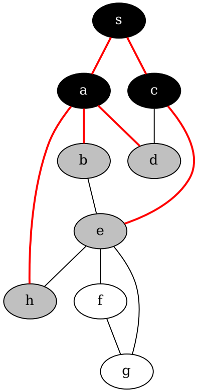
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	∞	∞	2

Queue:

b, d, h, e

Example



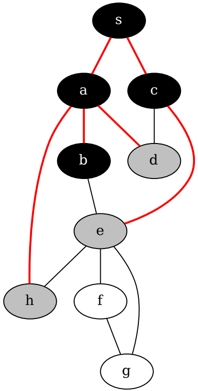
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	∞	∞	2

Queue:

b, d, h, e

Example



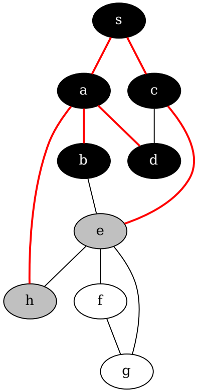
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	∞	∞	2

Queue:

d, h, e

Example



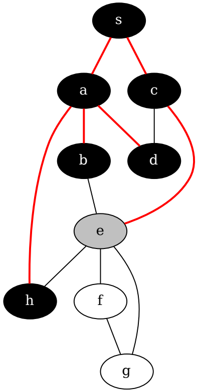
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	∞	∞	2

Queue:

h, e

Example



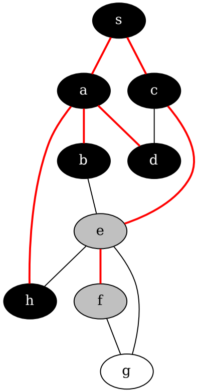
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	∞	∞	2

Queue:

e

Example



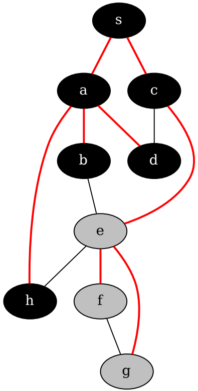
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	3	∞	2

Queue:

f

Example



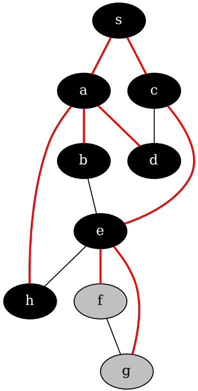
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	3	3	2

Queue:

f, g

Example



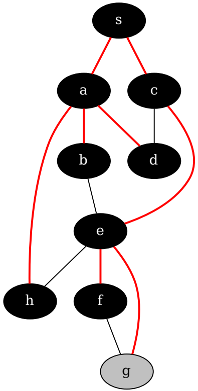
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	3	3	2

Queue:

f, g

Example



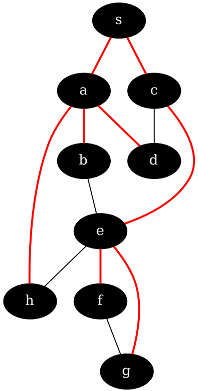
Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	3	3	2

Queue:

g

Example



Distances:

a	b	c	d	e	f	g	h
1	2	1	2	2	3	3	2

Queue:

$\emptyset$

Analysis of BFS

Running Time

- Initialization takes $O(n)$ time (and space).

Running Time

- Initialization takes $O(n)$ time (and space).
- Vertex colors go from White$\rightarrow$Gray$\rightarrow$Black
  - $O(n)$ color changes.

Running Time

- Initialization takes $O(n)$ time (and space).
- Vertex colors go from White$\rightarrow$Gray$\rightarrow$Black
  - $O(n)$ color changes.
- $O(n)$ queue operations:
  - Only enqueue White vertices, which we turn Gray $\rightarrow$ each vertex enqueued at most once.
  - Therefore, $\leq n$ dequeue operations.

Running Time

- Initialization takes $O(n)$ time (and space).
- Vertex colors go from White$\rightarrow$Gray$\rightarrow$Black
    - $O(n)$ color changes.
- $O(n)$ queue operations:
    - Only enqueue White vertices, which we turn Gray $\rightarrow$ each vertex enqueued at most once.
    - Therefore, $\leq n$ dequeue operations.
- $O(\deg(v))$ operations when v is dequeued.
    - Recall $\sum_{v \in V} \deg(v) = 2m$.

Running Time

- Initialization takes $O(n)$ time (and space).
- Vertex colors go from White$\rightarrow$Gray$\rightarrow$Black
  - $O(n)$ color changes.
- $O(n)$ queue operations:
  - Only enqueue White vertices, which we turn Gray $\rightarrow$ each vertex enqueued at most once.
  - Therefore, $\leq n$ dequeue operations.
- $O(\deg(v))$ operations when v is dequeued.
  - Recall $\sum_{v \in V} \deg(v) = 2m$.
- $\Rightarrow$ Total space: $O(n)$ and total time $O(n+m)$.

Correctness

Would like to establish that:

- BFS computes correct shortest-path distances from s.
- For all $v \in V$, the path from s to v in the BFS tree is a shortest $s \to v$ path in G.

Correctness

Would like to establish that:

- BFS computes correct shortest-path distances from s.
- For all $v \in V$, the path from s to v in the BFS tree is a shortest $s \to v$ path in G.

Define:

- $\mathrm{dist}(s, v)$: the (correct) shortest-path distance from s to v.
- $d_{BFS}(v)$: the distance from s to v computed by BFS.

We want:

$$\forall v \in V: \ \mathrm{dist}(s, v) = d_{BFS}(v)$$

The easy part

Lemma

For all $v \in V: \ \mathrm{dist}(s,v) \leq d_{BFS}(v)$

The easy part

Lemma

For all $v \in V$: $\mathrm{dist}(s, v) \leq d_{BFS}(v)$

Proof.

Induction on $d_{BFS}(v)$:

- Base case: $d_{BFS}(v) = 0$ only applies to s and is clearly correct.
- Suppose lemma correct if $d_{BFS}(v) \leq k$ and we have a vertex v for which $d_{BFS}(v) = k + 1$.
  - For some u (dequeued before v) we have $d_{BFS}(v) = d_{BFS}(u) + 1 \Rightarrow d_{BFS}(u) = k$.
  - Inductive hypothesis: $\mathrm{dist}(s, u) \leq d_{BFS}(u) = k$.
  - Therefore, $\mathrm{dist}(s, v) \leq \mathrm{dist}(s, u) + 1 \leq k + 1$.

□

Analyzing the queue

Lemma

Vertices are placed into the queue in non-decreasing order of d_{BFS}. *Furthermore, for any* u, v *which are simultaneously in the queue we have* $|d_{BFS}(u) - d_{BFS}(v)| \leq 1$.

Analyzing the queue

Lemma

Vertices are placed into the queue in non-decreasing order of d_{BFS}. Furthermore, for any u, v which are simultaneously in the queue we have $|d_{BFS}(u) - d_{BFS}(v)| \leq 1$.

Proof.

By induction on number of queue operations:

- Base case: empty queue, vacuously true
- General case: statement is true, then d_{BFS} values in the queue are of the form $k, k, \ldots, k, k+1, \ldots, k+1$ (or all vertices have same value).
- Dequeue operation $\rightarrow$ property still true.
- Enqueue operation $\rightarrow$ we add a neighbor of a just removed vertex and increase its distance by 1 $\rightarrow$ property still true.

□

The other part of the inequality

Lemma

For all $v \in V: \ \mathrm{dist}(s, v) \geq d_{BFS}(v)$

The other part of the inequality

Lemma

For all $v \in V: \ \mathrm{dist}(s, v) \geq d_{BFS}(v)$

Proof.

Suppose for $v \in V$ we have $\mathrm{dist}(s, v) < d_{BFS}(v)$. Among all such v, pick one with **minimum** $\mathrm{dist}(s, v)$.

- Let u be the last vertex before v in a shortest $s \to v$ path.
  - $\mathrm{dist}(s, u) = \mathrm{dist}(s, v) - 1 \Rightarrow \mathrm{dist}(s, u) = d_{BFS}(u)$.
- When u was dequeued, v was:
  - White: then $d_{BFS}(v) = d_{BFS}(u) + 1$, which is correct!
  - Gray: then $|d_{BFS}(v) - d_{BFS}(u)| \leq 1$, contradiction!
  - Black: then $d_{BFS}(v) \leq d_{BFS}(u)$, contradiction!

□

Implementation details

- BFS trees are not unique and depend on order in adjacency lists
  - Can the edge *cd* be added to the BFS tree in our example? What about *be*?
  - In most examples, we will assume that adjacency lists are sorted alphabetically.
- BFS can be executed also when graph is given in adjacency matrix form
  - Complexity $O(n^2)$ (why?)
- BFS also works for directed graphs
  - In line 8 we look at list of outneighbors.