Lecture 11 — Elementary Graph Algorithms (Part 1)

COMPSCI 308 — Design and Analysis of Algorithms

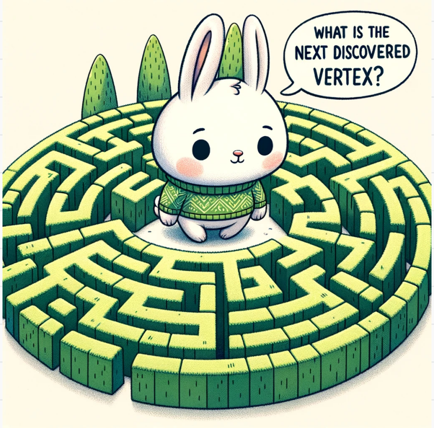
Xing Shi Cai https://newptcai.gitlab.io

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

Summary

Assignments[1]



Practice makes perfect!

Introduction to Algorithms (ITA)

📖 Required Readings:

- Section 20.1.
- Section 20.2.

✏️ Required Exercises:

- Exercises 20.1 — 1, 3–6.
- Exercises 20.2 — 1–6.

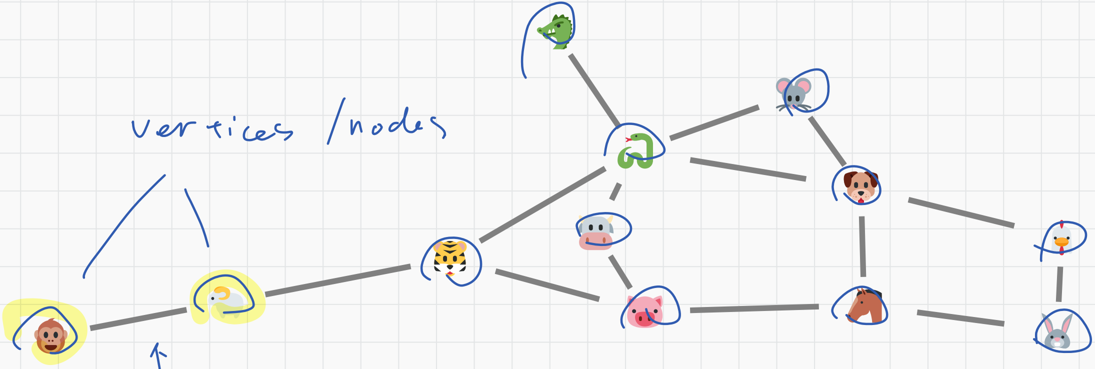
[1] 💣 Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

ITA 20.1 Representation of Graphs

Friendships

We can model friendships as a graph in which each vertex (node) represents a friend and each edge (line) represents a friendship.

vertices /nodes



edges

Figure 1: Friendships in COMPSCI 308

A metro network can also be modeled as a graph.

Figure 2: Mini Metro game screenshot. Source — Wikipedia.

Definition of Graph

A graph G is defined as a pair (V, E), where V is the vertex set, and E is the edge set.

An element in V is called a vertex or node.

An element in E is called an edge.

What is V and E of the following graph?

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

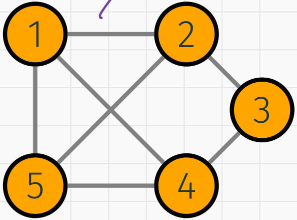


Figure 3: An undirected graph G

$E \subseteq V \times V$

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

📗 Definition of Graph

If $(x, y) \in E$, then x and y are adjacent, and the edge (x, y) is incident to both x and y.

If the edges are undirected, i.e., (x, y) and (y, x) are the same edge, the graph is said to be undirected.

🍰 What is the maximum number of edges a graph with $|V|$ vertices can have?

$$\binom{|V|}{2} = \frac{|V|(|V|-1)}{2}.$$

$$= \Theta(|V|^2)$$

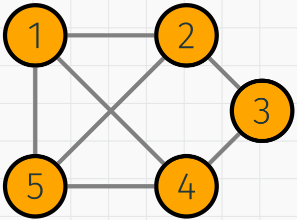


Figure 3: An undirected graph G

If the edges are directed, i.e., (x, y) and (y, x) are different edges, the graph is said to be directed or a digraph.

Parents and children.

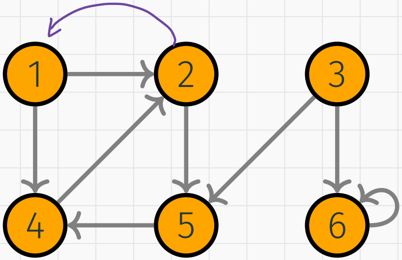


Figure 3: A directed graph G

Real world example:

one-way road network,

unrequited love,

Teacher student realtionships,

Boss and subordinates

If G is sparse, i.e., $|E| = o(|V|^2)$, then we often represent G in adjacency-list form.

(1,2), (1,5) are edges.

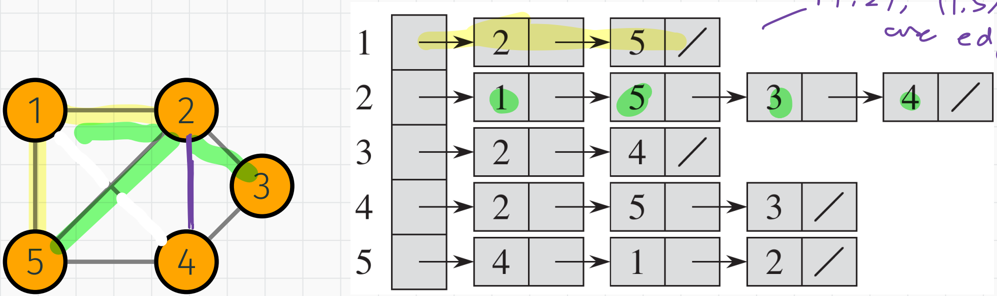
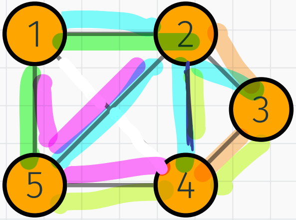


Figure 4: A graph and its adjacency-list representation

Adjacency Matrix Representation of a Graph

If G is dense, i.e., $|E| = \Theta(|V|^2)$, then we often represent G in adjacency-matrix form.



(1,2)   (1,5)

$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

Figure 5: A graph and its adjacency-matrix representation

🍰 Why is the adjacency-matrix for an undirected graph symmetric over the diagonal? this an undirected graph.

We can also represent a digraph in both adjacency-list and adjacency-matrix form.

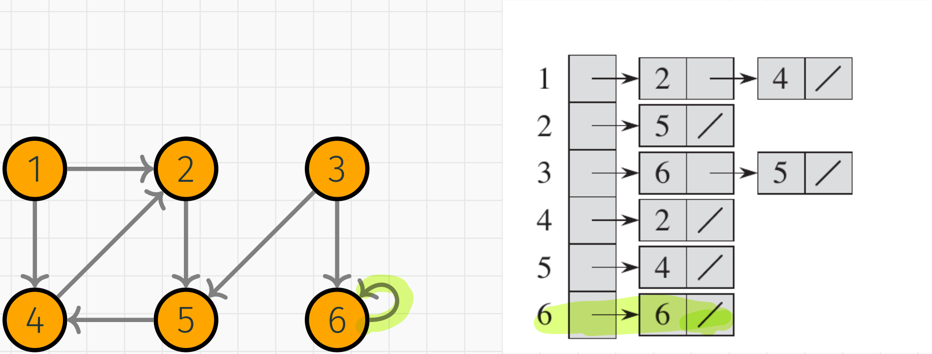
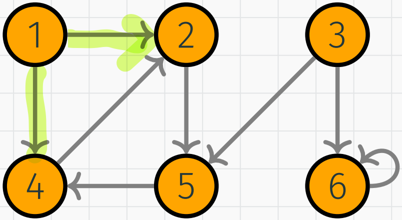


Figure 6: A digraph and its representations

Representation of Digraphs

We can also represent a digraph in both adjacency-list and adjacency-matrix form.



1 → 2   1 → 4.

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Figure 6: A digraph and its representations

💡 Note that the adjacency matrix is not necessarily symmetric for a digraph.

We use the notations $u.f$ and $(u, v).f$ to indicate the attributes of a vertex or an edge.

These attributes, such as edge weight, can also be stored in adjacency-list or adjacency-matrix form.

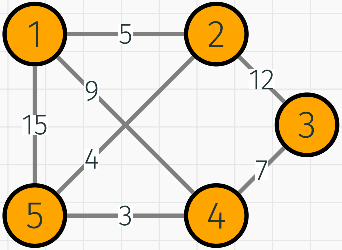


Figure 7: A weighted graph

$$\begin{bmatrix} 0 & 5 & 0 & 9 & 15 \\ 5 & 0 & 12 & 0 & 4 \\ 0 & 12 & 0 & 7 & 0 \\ 9 & 0 & 7 & 0 & 3 \\ 15 & 4 & 0 & 3 & 0 \end{bmatrix}$$

Real world example:

(1) Markov chain.

(2) length of roads.

(3) cost of time/money to travel

BFS.

ITA 20.2 Breadth-First Search

🍰 Can you design an algorithm which moves the tiles to their correct positions using the empty space?

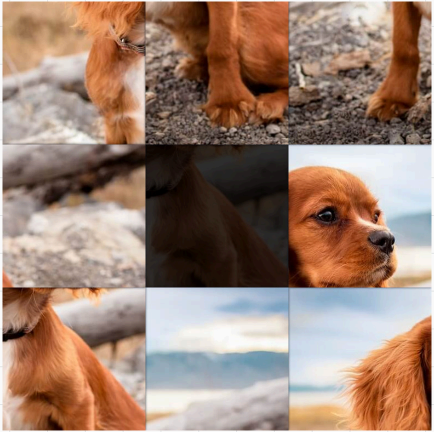


Figure 8: An 8-puzzle

Assume that a 😫 is spreading on a graph $G = (V, E)$.

Vertex 1 is infected first on day 0.

Each day, an infected vertex v infects all its neighbours.

How many days will it take for all vertices to become 😫?

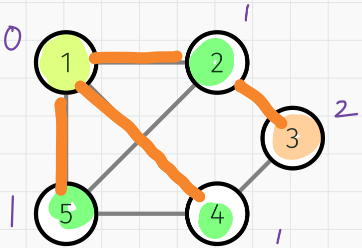
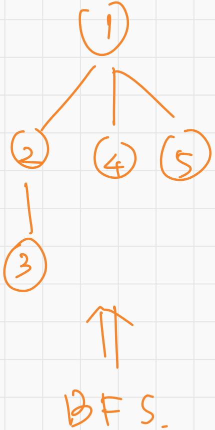


Figure 9: A graph G



Example of BFS

Breadth First Search (BFS) essentially works like the spread of 😧.

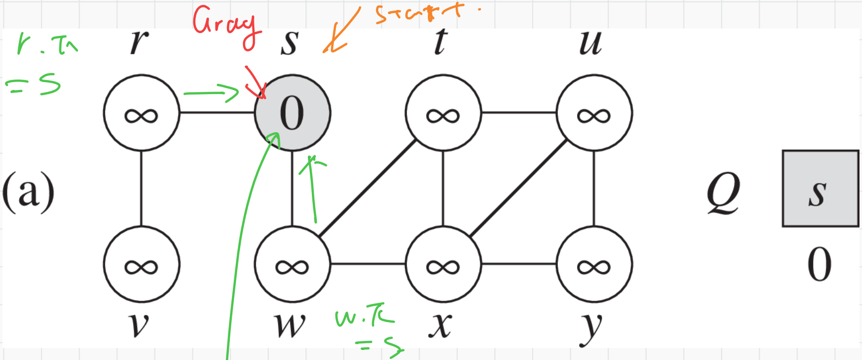


Figure 10: 📌 Run BFS on this graph

distance to s.

Breadth First Search (BFS) essentially works like the spread of 🤧.

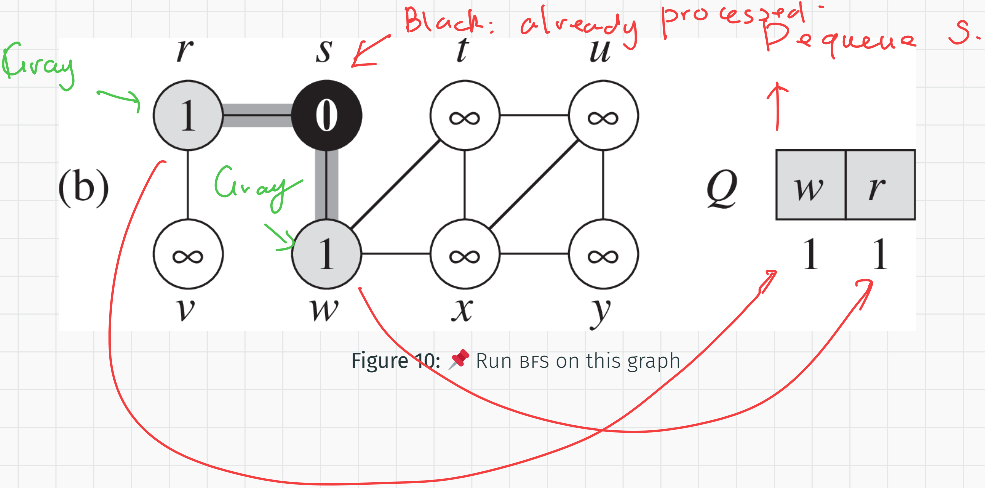


Figure 10: 📌 Run BFS on this graph

Breadth First Search (BFS) essentially works like the spread of 😧.

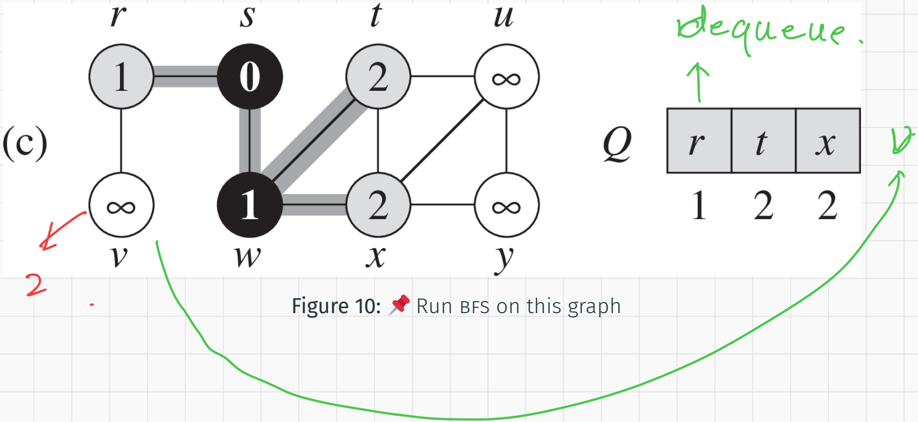


Figure 10: 📌 Run BFS on this graph

Breadth First Search (BFS) essentially works like the spread of 😷.

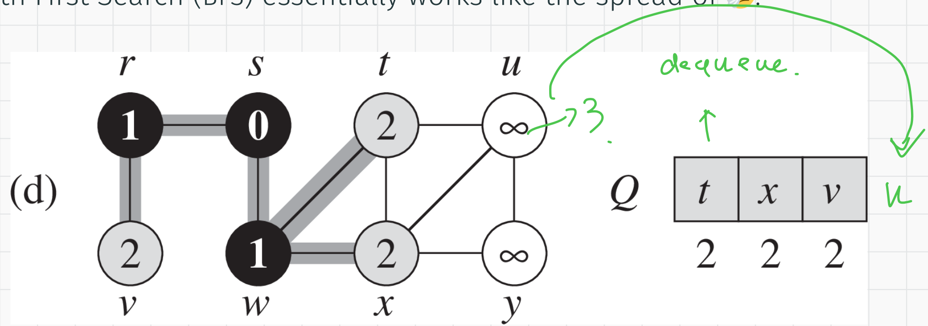


Figure 10: 📌 Run BFS on this graph

Breadth First Search (BFS) essentially works like the spread of 😤.

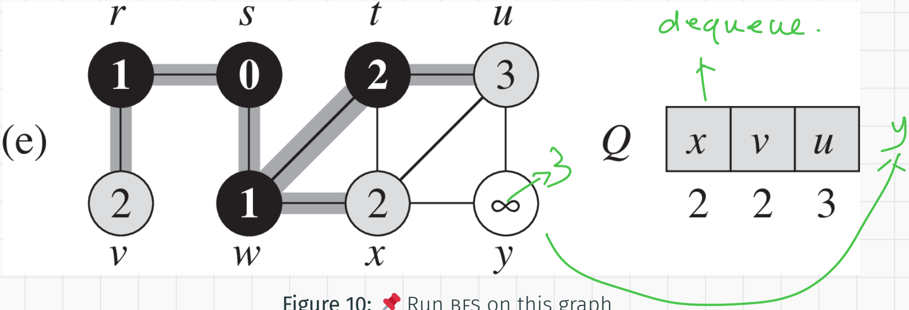


Figure 10: 📌 Run BFS on this graph

Breadth First Search (BFS) essentially works like the spread of 🤧.

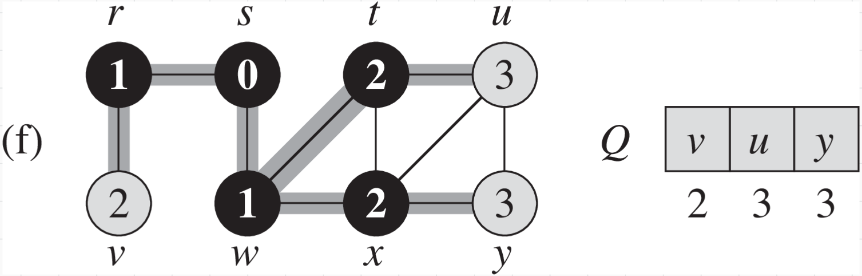


Figure 10: 📌 Run BFS on this graph

Example of BFS

Breadth First Search (BFS) essentially works like the spread of 🤧.

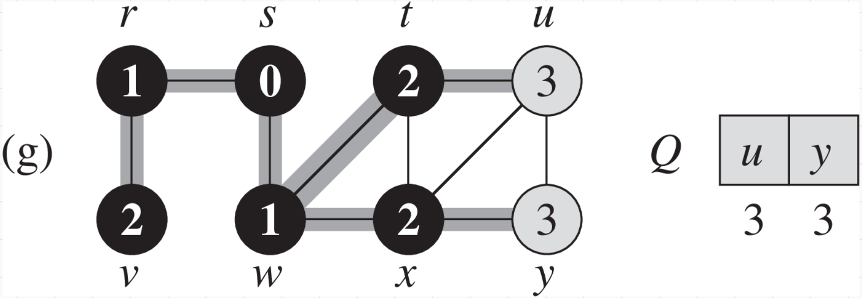


Figure 10: 📌 Run BFS on this graph

Example of BFS

Breadth First Search (BFS) essentially works like the spread of 🤧.

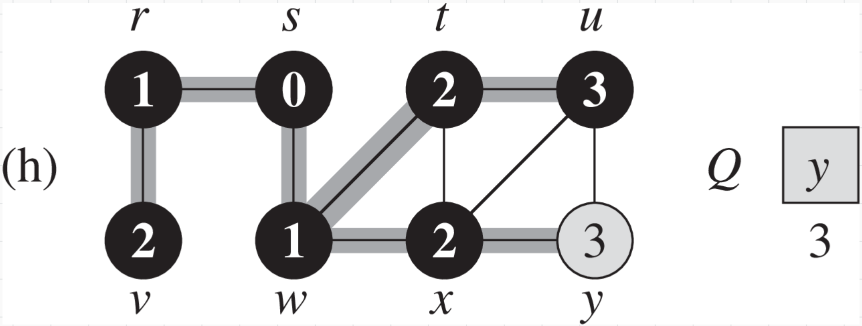


Figure 10: 📌 Run BFS on this graph

Example of BFS

Breadth First Search (BFS) essentially works like the spread of 🤧.

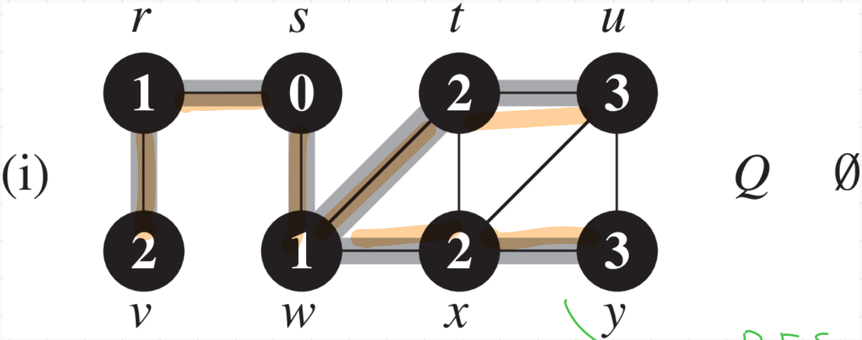


Figure 10: 📌 Run BFS on this graph

BFS tree.

What Is BFS

BFS operates on a graph $G = (V, E)$ with a source vertex s.

The algorithm explores edges of G to find all vertices reachable from s.

BFS computes the distance from s to each vertex, constructing a breadth-first tree.

Paths in the tree represent the shortest paths in G. from s.

What Is BFS

BFS operates on a graph $G = (V, E)$ with a source vertex s.

The algorithm explores edges of G to find all vertices reachable from s.

BFS computes the distance from s to each vertex, constructing a breadth-first tree.

Paths in the tree represent the shortest paths in G.

Dequeued.

Vertices are initially white, becoming grey upon discovery, and black once fully explored.

Enqueue.

Discovery occurs upon first encounter in the search.

A grey vertex has been discovered but not fully explored. Inside Queue.

A black vertex and all its adjacent vertices have been fully explored.

Out Queue.

The Queue Data Structure

Definition — A queue is a linear data structure that follows the First-In, First-Out (FIFO) principle.

Operations —

- Enqueue — Add an element to the end of the queue.
- Dequeue — Remove an element from the front of the queue.

Real-World Analogy — A line of customers waiting their turn.

Figure 11: A line of patients waiting to see a doctor is a queue

run $|V| - 1$ times
runtime: $\Theta(|V|)$

source.

BFS(G, s)
for each vertex $u \in G.V - \{s\}$ **do**
    $u.color$ = WHITE
    $u.d = \infty$
    $u.\pi$ = NIL
$s.color$ = GRAY
$s.d = 0$
$s.\pi$ = NIL
$Q = \emptyset$
ENQUEUE(Q, s)

Not discovered.
initial estimate of distance.
From where it is been discovered.
s to itself has distance 0.
s is the root in BFS tree.
Empty
s is the first in the Queue.

Run time: $\Theta(|V|)$

BFS(G, s)
for each vertex $u \in G.V - \{s\}$ do
  $u.color$ = WHITE
  $u.d = \infty$
  $u.\pi$ = NIL
$s.color$ = GRAY
$s.d = 0$
$s.\pi$ = NIL
$Q = \emptyset$
ENQUEUE(Q, s)

At head of Q

Run 2|E| times

while $Q \neq \emptyset$ do
  u = DEQUEUE(Q)
  for each $v \in G.Adj[u]$ do
    if $v.color$ == WHITE then
      $v.color$ = GRAY
      $v.d = u.d + 1$
      $v.\pi = u$
      ENQUEUE(Q, v)
  $u.color$ = BLACK → processed.

we have processed all edges

v is neighbors of u.

Distance

Not discovered

discovered

🍰 What's the ⏱ complexity?

Runtime: $\Theta(|E|)$

parent

child

$s \rightsquigarrow u \xrightarrow{+1} v$

$\underbrace{\qquad\qquad}_{}$ u.d

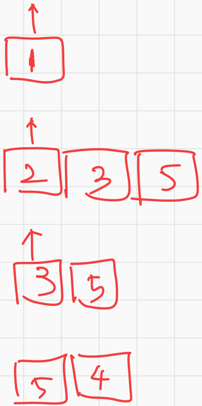
Total Runtime: $\Theta(|V|+|E|)$ $v.d = u.d + 1$

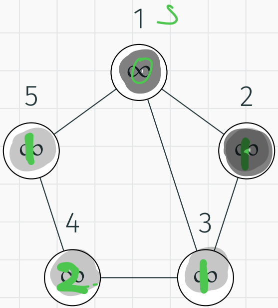
🍰 Run BFS on the following graph with $s = 1$ and stops after three vertices become black. What is $4.d$ at this moment?

Answer: 2

Assume that the neighbours of a vertex are stored in adjacency-list form, ordered by their labels in increasing order.

1 → 2 → 3 → 5





Distance in a Graph

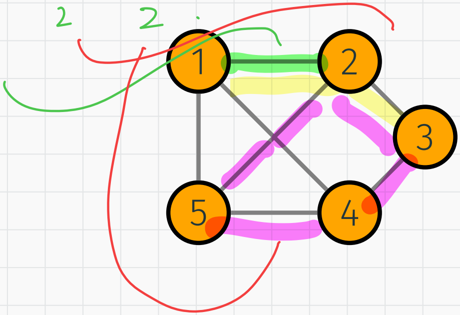
For two vertices u, v in G, the distance from u to v is the length of the shortest path from u to v.

We denote this distance by $\delta(u, v)$.

If there is no path from u to v, then $\delta(u, v) = \infty$.

🍰 What are $\delta(1,2), \delta(1,3), \delta(5,3)$?

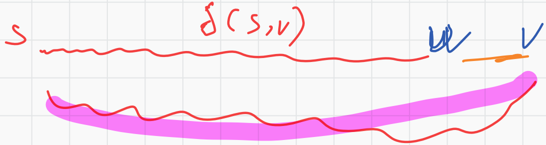
1 2 2



Let $G = (V, E)$ be a graph. Let $s \in V$ be an arbitrary vertex.

Then for every edge $(u, v) \in E$,

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



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

Proof: Since we can go from s to v by first go from s to u, then u to v, there is a path from s to v of length $\delta(s,u)+1$.

Lemma 20.2

Upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$.

Lemma 20.2

Upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$.

Proof Sketch —

We use induction on the number of `ENQUEUE` operations.

Base Case — Initially, only the source s is enqueued with $s.d = 0 = \delta(s, s)$ and $v.d = \infty \geq \delta(s, v)$ for all other v.

Lemma 20.2

Upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$.

Proof Sketch —

We use induction on the number of **ENQUEUE** operations.

Base Case — Initially, only the source s is enqueued with $s.d = 0 = \delta(s, s)$ and $v.d = \infty \geq \delta(s, v)$ for all other v.

For the inductive step, when a white vertex v is discovered from u, we set

$$v.d = u.d + 1 \geq \delta(s, u) + 1 \geq \delta(s, v)$$

Since $v.d$ is set only once, this maintains the inductive hypothesis.

Suppose that during the execution of BFS 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,

$$v_r.d \le v_1.d + 1$$

and

$$v_i.d \le v_{i+1}.d \qquad \text{for } i \in \{1, \ldots, r-1\}.$$

Suppose that during the execution of BFS the queue Q contains the vertices

$$\langle v_1, v_2, \dots, v_r \rangle,$$

where v_1 is the head of Q and v_r is the tail. Then,

$$v_r.d \le v_1.d + 1$$

and

推论

$$v_i.d \le v_{i+1}.d \qquad \text{for } i \in \{1, \dots, r-1\}.$$

Corollary 20.4

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 $v_i.d \le v_j.d$ at the time that v_j is enqueued.

Proof Outline of Lemma 20.3 (by Induction)

Base Case —

- Initially, only the source s is in Q, so the lemma holds.

Proof Outline of Lemma 20.3 (by Induction)

Base Case —

- Initially, only the source s is in Q, so the lemma holds.

Inductive Step —

- *Dequeue Operation —*
  - If v_1 is dequeued, v_2 becomes the new head.
  - By induction, $v_1.d \leq v_2.d$, and $v_r.d \leq v_1.d + 1 \leq v_2.d + 1$, preserving the inequalities.

Tail.

New head.

Base Case —

- Initially, only the source s is in Q, so the lemma holds.

Inductive Step —

$u \mid v_1 \mid v_2 \cdots v_r.$

- *Dequeue Operation —*
  - If v_1 is dequeued, v_2 becomes the new head.
  - By induction, $v_1.d \le v_2.d$, and $v_r.d \le v_1.d + 1 \le v_2.d + 1$, preserving the inequalities.
- *Enqueue Operation —*

  New tail

  Just been dequeued

  - New vertex v becomes v_{r+1} after u is processed.
  - By induction, $v_{r+1}.d = v.d = u.d + 1 \le v_1.d + 1$.
  - This maintains $v_r.d \le u.d + 1 = v_{r+1}.d$.

New tail.

New head.

$s \quad u \quad +1 \quad v.$

Theorem 20.5 — Correctness of BFS

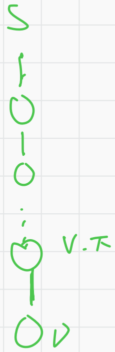
During its execution, BFS discovers every vertex $v \in V$ that is reachable from the source s, and upon termination, $v.d = \delta(s, v)$ for all $v \in V$.

Theorem 20.5 — Correctness of BFS

During its execution, BFS discovers every vertex $v \in V$ that is reachable from the source s, and upon termination, $v.d = \delta(s, v)$ 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 $v.\pi$ followed by the edge $(v.\pi, v)$.

💡 In short, BFS is correct.



Assume, for contradiction, that some vertex v has a d-value different from its shortest-path distance $\delta(s,v)$.

Let v be the vertex with minimum $\delta(s,v)$ that receives an incorrect d value; $v \neq s$.

No mistake
in first
step.

Proof of BFS Correctness

Assume, for contradiction, that some vertex v has a d-value different from its shortest-path distance $\delta(s, v)$.

Let v be the vertex with minimum $\delta(s, v)$ that receives an incorrect d value; $v \neq s$.

By Lemma 20.2, $v.d \geq \delta(s, v)$, so

$$v.d > \delta(s, v).$$

can not have =

Proof of BFS Correctness

Assume, for contradiction, that some vertex v has a d-value different from its shortest-path distance $\delta(s,v)$.

Let v be the vertex with minimum $\delta(s,v)$ that receives an incorrect d value; $v \neq s$.

By Lemma 20.2, $v.d \geq \delta(s,v)$, so

$$v.d > \delta(s,v).$$

Let u be the vertex immediately before v on a shortest path from s to v, meaning

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

No mistake $u.d = \delta(s,u)$.

$$\underbrace{s \xrightarrow{\ \delta(s,u)\ } u \xrightarrow{+1} v.}_{\delta(s,v).}$$

Assume, for contradiction, that some vertex v has a d-value different from its shortest-path distance $\delta(s, v)$.

Let v be the vertex with minimum $\delta(s, v)$ that receives an incorrect d value; $v \neq s$.

By Lemma 20.2, $v.d \geq \delta(s, v)$, so

$$v.d > \delta(s, v).$$

Let u be the vertex immediately before v on a shortest path from s to v, meaning

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

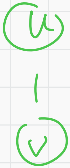
Thus, $u.d = \delta(s, u)$, so we have —

$$v.d > \delta(s, v) = \boxed{\delta(s, u) + 1} = u.d + 1.$$

Proof of BFS Correctness

Thus, $u.d = \delta(s, u)$, so we have —

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



When BFS dequeues u, v is either white, gray, or black —

- If v is white, then $v.d = u.d + 1$, contradicting $v.d > u.d + 1$.
- If v is black, then $v.d \leq u.d$ by Corollary 20.4, again a contradiction.
- If v is gray, it was painted gray when dequeuing some vertex w with $w.d \leq u.d$. So

  $$v.d = w.d + 1 \leq u.d + 1,$$

  which also contradicts $v.d > u.d + 1$.

Proof of BFS Correctness

Thus, $u.d = \delta(s, u)$, so we have —

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

When BFS dequeues u, v is either white, gray, or black —

- If v is white, then $v.d = u.d + 1$, contradicting $v.d > u.d + 1$.
- If v is black, then $v.d \leq u.d$ by Corollary 20.4, again a contradiction.
- If v is gray, it was painted gray when dequeuing some vertex w with $w.d \leq u.d$. So

  $$v.d = w.d + 1 \leq u.d + 1,$$

  which also contradicts $v.d > u.d + 1$.

Therefore, we have a contradiction and BFS is correct.

Predecessor Subgraph

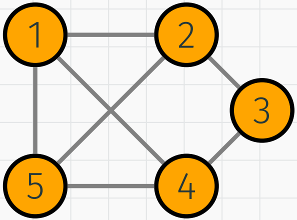
For a graph $G = (V, E)$ with source s, we define the predecessor subgraph of a BFS as $G_\pi = (V_\pi, E_\pi)$, where

$$V_\pi = \{v \in V : v.\pi \neq \mathrm{NIL}\} \cup \{s\}$$

and

$$E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}.$$

🍰 What is G_π for the following graph if $s = 1$?

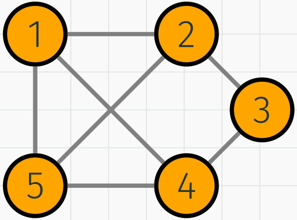


Breadth-First Tree

A subgraph of G is called a breadth-first tree if

- it contains all vertices reachable from s,
- every path in it is also the shortest path between the two endpoints,
- it is a tree.

🍰 What is a breadth-first tree for the following graph if $s = 1$?

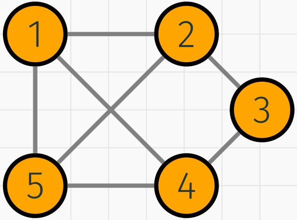


Breadth-First Tree

A subgraph of G is called a breadth-first tree if

- it contains all vertices reachable from s,
- every path in it is also the shortest path between the two endpoints,
- it is a tree.

🍰 Can you think of another one?



Lemma 20.6

The predecessor subgraph G_π of a graph G, as produced by BFS, is a breadth-first tree.

Proof Sketch —

- G_π is connected because it contains all vertices reachable from s, by Theorem 20.5.

Lemma 20.6

The predecessor subgraph G_π of a graph G, as produced by BFS, is a breadth-first tree.

Proof Sketch —

- G_π is connected because it contains all vertices reachable from s, by Theorem 20.5.
- The number of edges in G_π is one less than the number of vertices, as each vertex (except s) has a unique predecessor. Thus, G_π forms a tree.

The predecessor subgraph G_π of a graph G, as produced by BFS, is a breadth-first tree.

Proof Sketch —

- G_π is connected because it contains all vertices reachable from s, by Theorem 20.5.
- The number of edges in G_π is one less than the number of vertices, as each vertex (except s) has a unique predecessor. Thus, G_π forms a tree.
- The distance from s to any vertex v in G_π is $v.d = \delta(s, v)$, by Theorem 20.5. Therefore, the unique path from s to v in G_π represents a shortest path from s to v in G.

Given a graph G whose breadth-first tree has been computed by BFS, we can print the shortest path from s to v using the following algorithm —

```
Print-Path(G, s, v)
if v == s then
    print s
else if v.π == NIL then
    print "no path from" s "to" v
else
    Print-Path(G, s, v.π)
    print v
```

🍰 How can we solve this maze using BFS?

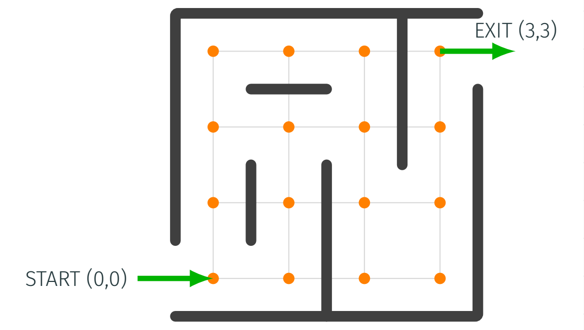


Figure 12: 4x4 Maze for BFS Visualization

Try it out at `https://brkwok.github.io/Maze-Solver/`.

How can we solve the 8-puzzle using BFS?

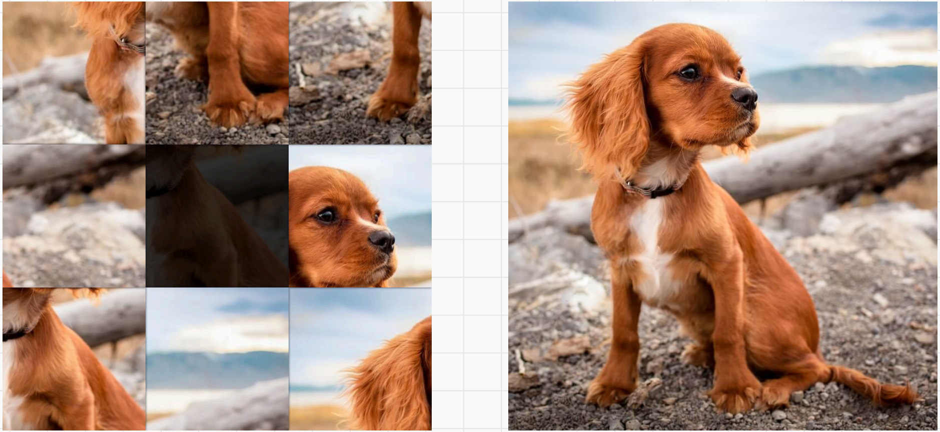


Figure 13: An 8-puzzle

8	5	4
1		3
6	7	2

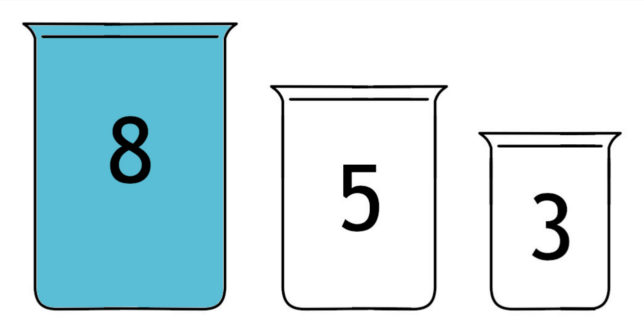
8	5	4
	1	3
6	7	2

8	5	4
1	3	
6	7	2

We have a jug filled with 8 units of water, and two empty jugs of sizes 5 and 3.

How can we make the first and second jugs both contain 4 units, and the third is empty, without using any extra tools?

🧩 How to solve this puzzle using BFS?



What are your main takeaways today? Any questions?