Chapter 20. Elementary Graph Algorithms

Joon Soo Yoo

May 29, 2025

Assignment

- Read §20.2, §20.3
- Problems
  - §20.2 - 1

Chapter 20: Elementary Graph Algorithms

- Chapter 20.1: Representations of Graphs
- **Chapter 20.2: Breadth-First Search**
- Chapter 20.3: Depth-First Search
- Chapter 20.4: Topological Sort
- Chapter 20.5: Strongly Connected Components

Breadth-First Search (BFS): Introduction

- Breadth-First Search (BFS) is a fundamental graph traversal algorithm.
- It serves as the foundation for more advanced algorithms, such as:
  - Prim's Minimum Spanning Tree algorithm (§21.2)
  - Dijkstra's Single-Source Shortest Paths algorithm (§22.3)

Purpose of BFS

- Given a graph $G = (V, E)$ and a source vertex s, BFS performs the following:
  - Systematically explores all vertices reachable from s.
  - Computes the shortest path (in edge count) from s to each reachable vertex v.
  - Builds a **breadth-first tree** rooted at s containing all reachable vertices.
- For each reachable vertex v, the path from s to v in the BFS tree is a shortest path in G.

Why It's Called Breadth-First

- BFS explores the graph level by level, or in "waves":
  - First it discovers all vertices at distance 1 from s.
  - Then all vertices at distance 2, and so on.
- This layered discovery is why it's called "breadth-first".

Queue-Based Exploration

- BFS uses a single **first-in, first-out (FIFO) queue** to manage the frontier of exploration.
- At any point, the queue contains some vertices at distance k and possibly some at distance $k + 1$.
- This queue maintains the order of discovery and ensures vertices are processed by increasing distance.

Coloring Scheme and Tree Structure

- BFS uses a color scheme to track the state of each vertex:
  - **WHITE**: undiscovered
  - **GRAY**: discovered but not fully explored (in queue)
  - **BLACK**: fully explored
- Initially, all vertices are white; unreachable vertices remain white.
- Each time a white vertex v is discovered from a gray vertex u, BFS:
  - Marks v gray and adds it to the queue
  - Sets $v.d = u.d + 1$
  - Sets $v.\pi = u$ and adds edge (u, v) to the BFS tree
- Once all neighbors of u are explored, u is marked black.

BFS: Inputs and Vertex Attributes

- BFS operates on a graph $G = (V, E)$ represented as an **adjacency list**.
- Each vertex $v \in V$ maintains three attributes:
  - $\texttt{v.color} \in \{\text{WHITE, GRAY, BLACK}\}$
  - $\texttt{v.d}$: distance from source vertex s
  - $\texttt{v.}\pi$: predecessor (or parent) in the BFS tree
- A **queue** Q is used to manage discovered (GRAY) vertices.

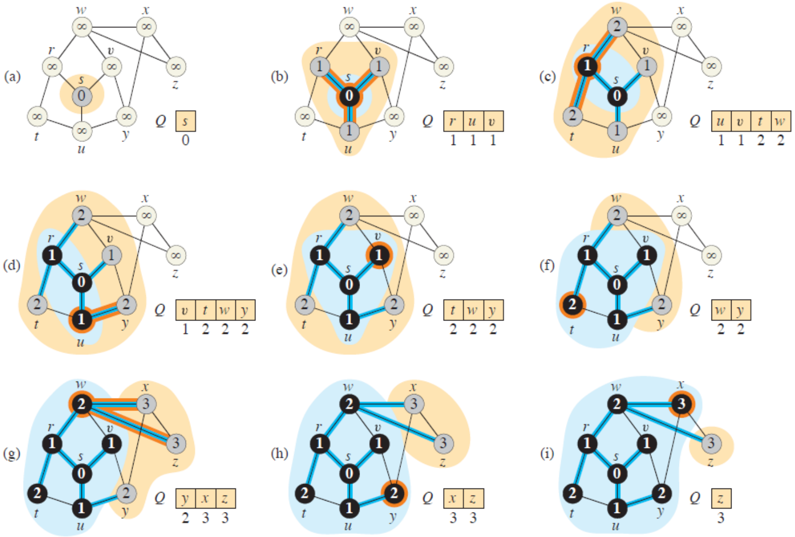
BFS Algorithm

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each vertex v in G.Adj[u]      // search the neighbors of u
        if v.color == WHITE            // is v being discovered now?
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)              // v is now on the frontier
    u.color = BLACK                    // u is now behind the frontier
```

Understanding the BFS Queue

- The queue Q maintains the **frontier** — vertices discovered but not fully explored.
- Invariant: at any time, Q contains only GRAY vertices.
- Vertices are dequeued in the order they were discovered, ensuring:
  - Vertices at distance k are fully explored before those at $k + 1$
  - This leads to the correct computation of shortest paths

BFS Diagram


(a)
Q s
0
(b)
Q r u v
1 1 1
(c)
Q u v t w
1 1 2 2
(d)
Q v t w y
1 2 2 2
(e)
Q t w y
2 2 2
(f)
Q w y
2 2
(g)
Q y x z
2 3 3
(h)
Q x z
3 3
(i)
Q z
3

BFS: Time Complexity Analysis

- Let the input graph be $G = (V, E)$ and represented as an adjacency list.
- BFS performs the following:
  - **Initialization:** each vertex is colored and initialized once $\Rightarrow O(V)$
  - **Queue operations:**
    - Each vertex is enqueued and dequeued at most once
    - Each enqueue/dequeue takes $O(1)$ time
    - Total queue time: $O(V)$
  - **Adjacency list scanning:**
    - Each vertex's adjacency list is scanned once when it is dequeued
    - Total length of all adjacency lists: $\Theta(E)$
    - Total scanning time: $\Theta(E)$
- **Total time complexity:** $\boxed{O(V + E)}$
- BFS runs in linear time relative to the size of the input graph.

Shortest Paths and BFS: Lemma 20.1

Lemma

Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$. Define $\delta(s, v)$ as the **shortest-path distance** *from s to v, i.e., the minimum number of edges in any path from s to v. If v is unreachable from s, then $\delta(s, v) = \infty$. Then, for any edge $(u, v) \in E$, we have:*

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

Proof of Lemma 20.1

- **Case 1:** u is reachable from s
  - Then there is a shortest path from s to u of length $\delta(s, u)$
  - Since $(u, v) \in E$, we can extend this path by one edge to reach v
  - So, there exists a path from s to v of length $\delta(s, u) + 1$
  - Therefore, $\delta(s, v) \leq \delta(s, u) + 1$
- **Case 2:** u is not reachable from s
  - Then $\delta(s, u) = \infty$
  - So $\delta(s, v) \leq \infty + 1 = \infty$
  - Inequality still holds

Lemma 20.2

Lemma

Let $G = (V, E)$ be a directed or undirected graph, and suppose BFS is run from a source vertex $s \in V$. Then for every vertex $v \in V$, the value $v.d$ computed by BFS satisfies:

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

This holds at all times during the execution of BFS, including at termination.

Proof of Lemma 20.2

- **Proof by induction on the number of `ENQUEUE` operations.**
- **Base case:** after enqueuing s (line 9 of BFS)
  - $s.d = 0 = \delta(s, s)$
  - All other vertices v have $v.d = \infty \geq \delta(s, v)$
- **Inductive step:** suppose a white vertex v is discovered from u
  - By inductive hypothesis: $u.d \geq \delta(s, u)$
  - BFS sets $v.d = u.d + 1$
  - By Lemma 20.1: $\delta(s, v) \leq \delta(s, u) + 1$
  - Therefore:

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

- Once set, $v.d$ never changes again.
- Hence, the inductive hypothesis is maintained.

Lemma 20.3: Structure of the BFS Queue

Lemma

Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $<v_1, v_2, \ldots, v_r>$, where v_1 is at the head and v_r is at the tail. Then:

$$v_r.d \leq v_1.d + 1$$
$$v_i.d \leq v_{i+1}.d \quad \textit{for } i = 1, 2, \ldots, r-1$$

- That is, the d-values in the queue are either all equal or increase from k to $k+1$.

Proof Sketch of Lemma 20.3

- **Proof by induction** on the number of queue operations (enqueue/dequeue).
- **Base case:** Queue starts as $Q = [s]$. Then $s.d = 0$, and the lemma trivially holds.
- **Dequeue step:**
  - Removing v_1 shifts the queue to $\langle v_2, \ldots, v_r \rangle$.
  - By induction: $v_r.d \leq v_1.d + 1 \leq v_2.d + 1$, so the inequality is maintained.
- **Enqueue step:**
  - Suppose v is discovered via v_1.
  - Then $v.d = v_1.d + 1$, and v is enqueued at the end.
  - At that time, $v_r.d \leq v_1.d + 1 = v.d$, so the non-decreasing order is preserved.
- In both enqueue and dequeue steps, the d-value ordering in the queue remains valid.

Corollary 20.4

Corollary

Suppose vertices v_i and v_j are enqueued during BFS and v_i is enqueued before v_j. Then at the moment v_j is enqueued,

$$v_i.d \leq v_j.d.$$

- Follows directly from Lemma 20.3: the queue maintains non-decreasing d-values.
- Since vertices are enqueued only once, the distance of v_j must be at least that of v_i.

Theorem 20.5: Correctness of BFS (Part 1)

Theorem (Correctness of Breadth-First Search)

Let $G = (V, E)$ be a directed or undirected graph. Suppose BFS is run from a source vertex $s \in V$. Then:

- *Every vertex v reachable from s is discovered by BFS.*
- *Upon termination, $v.d = \delta(s, v)$ for all $v \in V$.*
- *Moreover, for any reachable $v \neq s$, one of the shortest paths from s to v is formed by taking a shortest path to $v.\pi$ and following the edge $(v.\pi, v)$.*

Proof idea: Use contradiction and properties from Lemmas 20.1–20.3.

Correctness of BFS (Part 2): Assume a Contradiction

- Suppose **some vertex** v receives $v.d \neq \delta(s, v)$.
- Let v be a vertex with **minimum** $\delta(s, v)$ among such incorrect vertices.
- By Lemma 20.2: $v.d \geq \delta(s, v)$. So if incorrect, $v.d > \delta(s, v)$.
- $v \neq s$ (since $s.d = 0 = \delta(s, s)$).
- Let u be the predecessor of v on a shortest path from s.
- Then $\delta(s, v) = \delta(s, u) + 1$.
- By minimality of v, $u.d = \delta(s, u)$.

Correctness of BFS (Part 3): Analyze When u is Dequeued

Now consider the moment BFS dequeues u from Q:

- Vertex v must be white, gray, or black at this point.
- **Case 1: v is white**:
  - Then BFS sets $v.d = u.d + 1$.
  - But $u.d = \delta(s, u)$, so $v.d = \delta(s, v)$ **(contradiction)**.
- **Case 2: v is black**:
  - Then v has already been removed from Q.
  - By Corollary 20.4: $v.d \leq u.d$.
  - So $v.d \leq \delta(s, u) < \delta(s, v)$ **(contradiction)**.

Correctness of BFS (Part 4): Remaining Case v is Gray

- **Case 3: v is gray**:
  - Then v was discovered while dequeuing some earlier vertex w.
  - So $v.d = w.d + 1$.
  - By Corollary 20.4: $w.d \leq u.d = \delta(s, u)$.
  - So $v.d = w.d + 1 \leq u.d + 1 = \delta(s, v)$ **(contradiction)**.
- In all cases, we reached a contradiction.
- Therefore, $v.d = \delta(s, v)$ for all v.

Breadth-First Trees

- As BFS explores the graph from a source s, it builds a tree structure rooted at s.
- Each vertex v reachable from s gets a parent $v.\pi$ from which it was discovered.
- These parent pointers form the **predecessor subgraph**, which encodes shortest paths from s.

Visual: Blue edges in Figure 20.3 represent this tree structure.

Definition: Predecessor Subgraph

Let $G = (V, E)$ be a graph with source vertex s. The **predecessor subgraph** $G_\pi = (V_\pi, E_\pi)$ is defined as:

$$V_\pi = \{v \in V \mid v.\pi \neq \text{NIL}\} \cup \{s\}$$
$$E_\pi = \{(v.\pi, v) \mid v \in V_\pi \setminus \{s\}\}$$

- This forms the tree structure discovered by BFS.
- Each edge $(v.\pi, v)$ is called a **tree edge**.

Lemma 20.6: BFS Builds a Breadth-First Tree

Lemma

When BFS is applied to a graph $G = (V, E)$ *from a source vertex* $s \in V$*, the resulting predecessor subgraph* $G_\pi = (V_\pi, E_\pi)$ *forms a breadth-first tree.*

Proof of Lemma 20.6

- Line 16 of BFS sets $v.\pi = u$ only if $(u, v) \in E$ and v is undiscovered.
- Therefore, v is reachable from s, and so V_π consists of all vertices reachable from s.
- The set $E_\pi = \{(v.\pi, v) \mid v \in V_\pi \setminus \{s\}\}$ defines edges from each discovered node to its predecessor.
- By Theorem B.2, since G_π is connected and has $|V_\pi| - 1$ edges, it forms a tree.
- From Theorem 20.5, $v.d = \delta(s, v)$, and each $(v.\pi, v)$ lies on a shortest path.
- Thus, G_π is a tree of shortest paths: a breadth-first tree.

Exercise: BFS Tree Construction

Graph Description:

Consider the undirected graph $G = (V, E)$ with vertex set $V = \{1, 2, 3, 4, 5\}$. The graph is given in adjacency list form:

- $\text{Adj}[1] = [2, 3]$
- $\text{Adj}[2] = [1, 4, 5]$
- $\text{Adj}[3] = [1, 4]$
- $\text{Adj}[4] = [2, 3, 5]$
- $\text{Adj}[5] = [2, 4]$

Tasks:

(a) Draw the original undirected graph based on the adjacency list.

(b) Run **BFS** starting from the source vertex $s = 1$. For each vertex, record:

- Its distance $v.d$ from the source.
- Its predecessor $v.\pi$ in the BFS tree.

(c) Draw the **breadth-first tree** formed by BFS. Include only the tree edges determined by the π values.

BFS Print Path

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

Question?