CMPSC 465: LECTURE XIV

Breath First Search
and
Shortest Paths

Ke Chen

October 01, 2025

Breadth First Search (BFS)

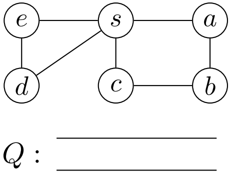
Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

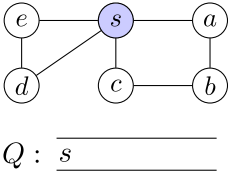
```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```



Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

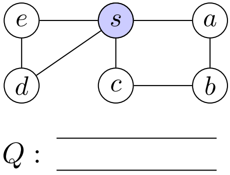
```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```



Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```



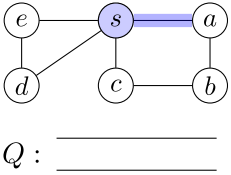
Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

`//` $visited$ `is an array of length` $|V|$`, filled with 0's`

BFS-Explore(G, s, $color$)

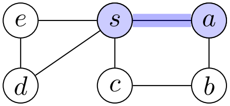
- Q.enqueue(s) `//` Q `is a queue`
- $visited[s] = color$
- **while** Q *is not empty* **do**
  - $v = Q$.dequeue()
  - **foreach** *edge* $\{v, w\} \in E$ **do**
    - **if** $visited[w] == 0$ **then**
      - Q.enqueue(w)
      - $visited[w] = color$



Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```

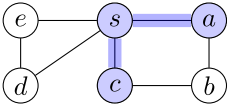


Q : a

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```



$Q:$ $a \; c$

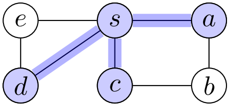
Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

// $visited$ is an array of length $|V|$, filled with 0's

BFS-Explore(G, s, $color$)

Q.enqueue(s) // Q is a queue
$visited[s] = color$
while Q *is not empty* **do**
  $v = Q$.dequeue()
  foreach *edge* $\{v, w\} \in E$ **do**
    if $visited[w] == 0$ **then**
      Q.enqueue(w)
      $visited[w] = color$

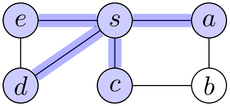


$Q:$ $a \ \ c \ \ d$

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```

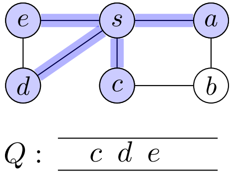


Q : $a \;\; c \;\; d \;\; e$

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

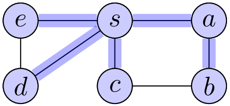
```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```



Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```

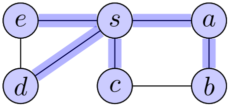


Q : $c \; d \; e \; b$

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```

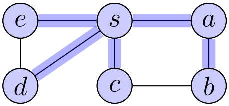


Q : d e b

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```

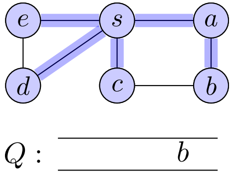


Q : e b

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

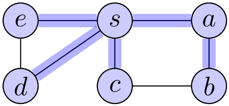
```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```



Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```

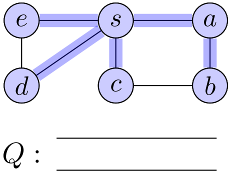


$Q :$

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```

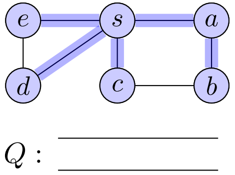


Correctness? Almost the same as Explore for DFS (Exercise).

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```

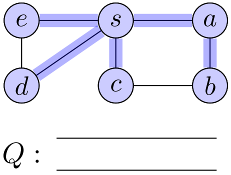


Time complexity?

Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
BFS-Explore(G, s, color)
    Q.enqueue(s) // Q is a queue
    visited[s] = color
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if visited[w] == 0 then
                Q.enqueue(w)
                visited[w] = color
```



Time complexity? Each vertex is enqueued exactly once.
Each edge is considered twice.

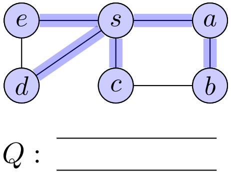
Breadth First Search (BFS)

Input: Graph $G = (V, E)$, a starting vertex s, an integer $color$
Output: All vertices reachable from s marked with $color$

```
// visited is an array of length |V|, filled with 0's
```

BFS-Explore(G, s, $color$)

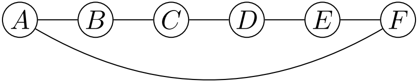
```
Q.enqueue(s) // Q is a queue
visited[s] = color
while Q is not empty do
    v = Q.dequeue()
    foreach edge {v, w} ∈ E do
        if visited[w] == 0 then
            Q.enqueue(w)
            visited[w] = color
```



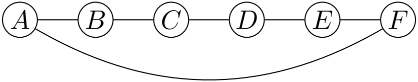
Time complexity? Each vertex is enqueued exactly once.
Each edge is considered twice.

Adjacency list: $O(|V| + |E|)$ Adjacency matrix: $O(|V|^2)$

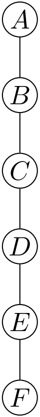
BFS vs DFS



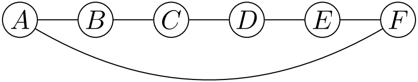
BFS vs DFS



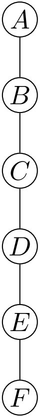
DFS tree:



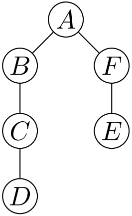
BFS vs DFS



DFS tree:



BFS tree:



BFS vs DFS

- Just like DFS, the same BFS algorithm works for both directed and undirected graphs.

BFS vs DFS

- Just like DFS, the same BFS algorithm works for both directed and undirected graphs.
- Same complexity

BFS vs DFS

- Just like DFS, the same BFS algorithm works for both directed and undirected graphs.
- Same complexity (how about MicroMouse?)

BFS vs DFS

- Just like DFS, the same BFS algorithm works for both directed and undirected graphs.
- Same complexity (how about MicroMouse?)
- Both can answer queries “Can I go from A to B?”

BFS vs DFS

- Just like DFS, the same BFS algorithm works for both directed and undirected graphs.
- Same complexity (how about MicroMouse?)
- Both can answer queries "Can I go from A to B?"
- DFS finds some path, BFS finds the shortest path

BFS vs DFS

- Just like DFS, the same BFS algorithm works for both directed and undirected graphs.
- Same complexity (how about MicroMouse?)
- Both can answer queries "Can I go from A to B?"
- DFS finds some path, BFS finds the shortest path (sometimes).

The shortest path problem

Given a graph G, what is the shortest path between two vertices?

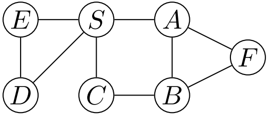
First scenario G is unweighted, i.e., all edges have distance one.

The shortest path problem

Given a graph G, what is the shortest path between two vertices?

First scenario G is unweighted, i.e., all edges have distance one.

Example



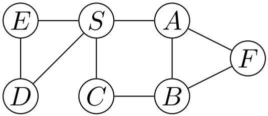
- Shortest path between D and F?

The shortest path problem

Given a graph G, what is the shortest path between two vertices?

First scenario G is unweighted, i.e., all edges have distance one.

Example



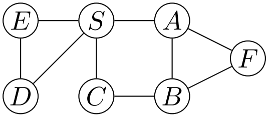
- Shortest path between D and F? $D \rightarrow S \rightarrow A \rightarrow F$, distance=3

The shortest path problem

Given a graph G, what is the shortest path between two vertices?

First scenario G is unweighted, i.e., all edges have distance one.

Example



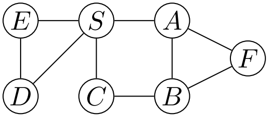
- Shortest path between D and F? $D \to S \to A \to F$, distance=3
- Shortest path between C and A?

The shortest path problem

Given a graph G, what is the shortest path between two vertices?

First scenario G is unweighted, i.e., all edges have distance one.

Example



- Shortest path between D and F? $D \to S \to A \to F$, distance=3
- Shortest path between C and A? $C \to S(\text{or } B) \to A$, distance=2

Unweighted shortest path with BFS

Idea We can modify BFS to keep track of distances.

Unweighted shortest path with BFS

Idea We can modify BFS to keep track of distances.

Input: Graph $G = (V, E)$, a starting vertex s
Output: Shortest distances to all vertices reachable from s

```
// dist is an array of length |V|, filled with ∞'s
BFS-ShortestPath(G, s)
    Q.enqueue(s) // Q is a queue
    dist[s] = 0
    while Q is not empty do
        v = Q.dequeue()
        foreach edge {v, w} ∈ E do
            if dist[w] == ∞ then
                Q.enqueue(w)
                dist[w] = dist[v] + 1
```

Unweighted shortest path with BFS

Time complexity? Same as BFS.

Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

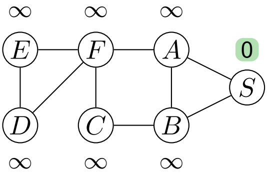
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



Q : S

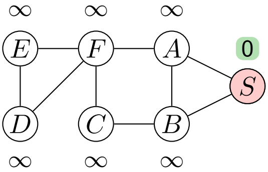
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



Q :

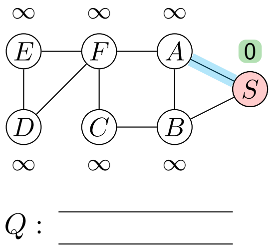
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



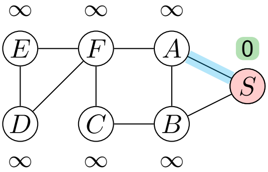
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



Q : A

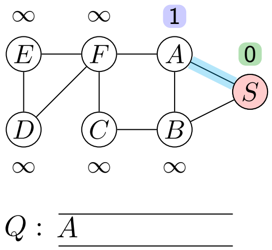
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



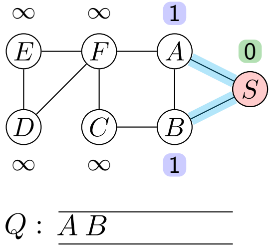
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



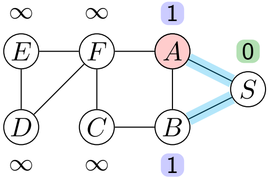
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



$Q :$ B

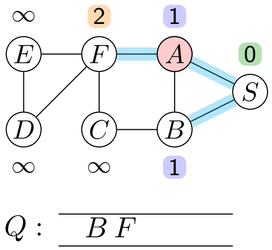
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



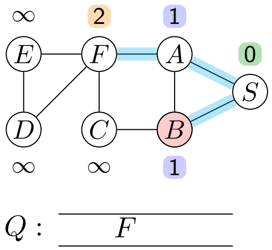
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



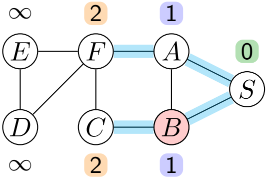
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



$Q : \; F \; C$

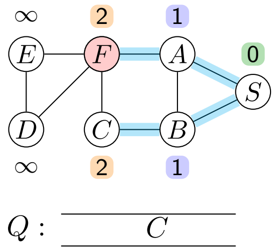
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



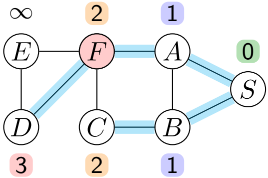
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



Q : $C\ D$

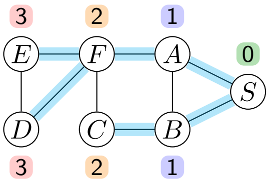
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



Q : $C\ D\ E$

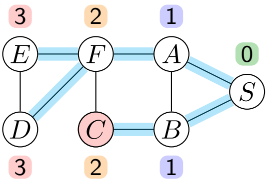
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



Q : $D\ E$

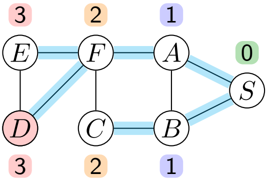
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores “layer by layer”: for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



Q : E

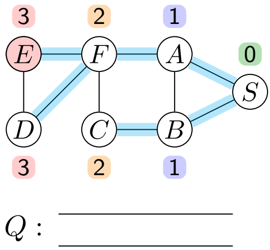
Unweighted shortest path with BFS

Time complexity? Same as BFS.

Correctness? Prove by induction that BFS explores "layer by layer": for $d = 0, 1, 2, \ldots$, there is a moment at which

- All nodes with distance $\leq d$ from s have the correct distances.
- All other nodes have distances ∞.
- The queue contains exactly all nodes at distance d.

Example



Unweighted shortest path with BFS

Remarks We find all shortest distances from s, this is called the single-source shortest path problem.

- No need to call Explore on unvisited vertices.
- Have to make a fresh start if want shortest distances from a different node.

Shortest path on weighted graphs

In many applications, having weights on edges is useful.

The edge weights could represent distances, cost, time, etc.

Shortest path on weighted graphs

In many applications, having weights on edges is useful.

The edge weights could represent distances, cost, time, etc.

Second scenario Find shortest paths on weighted graphs.

Shortest path on weighted graphs

In many applications, having weights on edges is useful.

The edge weights could represent distances, cost, time, etc.

Second scenario Find shortest paths on weighted graphs.

Idea Suppose all the weights are positive integers, we can add dummy nodes to represent edge weights.

Shortest path on weighted graphs

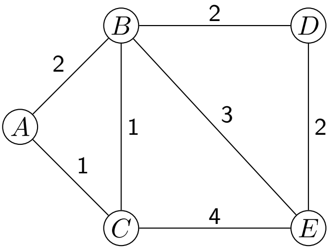
In many applications, having weights on edges is useful.

The edge weights could represent distances, cost, time, etc.

Second scenario Find shortest paths on weighted graphs.

Idea Suppose all the weights are positive integers, we can add dummy nodes to represent edge weights.

Example



Shortest path on weighted graphs

In many applications, having weights on edges is useful.

The edge weights could represent distances, cost, time, etc.

Second scenario Find shortest paths on weighted graphs.

Idea Suppose all the weights are positive integers, we can add dummy nodes to represent edge weights.

Example