Graphs – Breadth First Search

[Diagram of a graph with nodes SFO, ORD, LAX, and DFW connected by edges with weights labeled: SFO to LAX 337, LAX to DFW 1233, DFW to ORD 802, ORD to SFO 1843]
Outline

- BFS Algorithm
- BFS Application: Shortest Path on an unweighted graph
- Unweighted Shortest Path: Proof of Correctness
Outline

- BFS Algorithm
- BFS Application: Shortest Path on an unweighted graph
- Unweighted Shortest Path: Proof of Correctness
Breadth-First Search

- Breadth-first search (BFS) is a general technique for traversing a graph
- A BFS traversal of a graph $G$
  - Visits all the vertices and edges of $G$
  - Determines whether $G$ is connected
  - Computes the connected components of $G$
  - Computes a spanning forest of $G$
- BFS on a graph with $|V|$ vertices and $|E|$ edges takes $O(|V|+|E|)$ time
- BFS can be further extended to solve other graph problems
  - Cycle detection
  - Find and report a path with the minimum number of edges between two given vertices
BFS Algorithm Pattern

BFS(G,s)

Precondition: G is a graph, s is a vertex in G

Postcondition: all vertices in G reachable from s have been visited

for each vertex \( u \in V[G] \)
   
   \[ \text{color}[u] \leftarrow \text{BLACK} \quad // \text{initialize vertex} \]

\[ \text{colour}[s] \leftarrow \text{RED} \]

Q.enqueue(s)

while Q ≠ ∅

   \( u \leftarrow \text{Q.dequeue()} \)

   for each \( v \in \text{Adj}[u] \quad // \text{explore edge (u,v)} \)

      if \( \text{color}[v] = \text{BLACK} \)

         \[ \text{colour}[v] \leftarrow \text{RED} \]

         Q.enqueue(\( v \))

\[ \text{colour}[u] \leftarrow \text{GRAY} \]
BFS is a Level-Order Traversal

- Notice that in BFS exploration takes place on a wavefront consisting of nodes that are all the same distance from the source \( s \).

- We can label these successive wavefronts by their distance: \( L_0, L_1, \ldots \).
### BFS Example

- **A**: undiscovered
- **A**: discovered (on Queue)
- **A**: finished
- **---**: unexplored edge
- **→**: discovery edge
- **---**: cross edge

**L₀**

**L₁**

**A**

**B**

**C**

**D**

**E**

**F**
BFS Example (cont.)
BFS Example (cont.)
Properties

Notation

\( G_s \): connected component of \( s \)

Property 1

\( BFS(G, s) \) visits all the vertices and edges of \( G_s \)

Property 2

The discovery edges labeled by \( BFS(G, s) \) form a spanning tree \( T_s \) of \( G_s \)

Property 3

For each vertex \( v \) in \( L_i \)

- The path of \( T_s \) from \( s \) to \( v \) has \( i \) edges
- Every path from \( s \) to \( v \) in \( G_s \) has at least \( i \) edges
Analysis

- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled three times
  - once as BLACK (undiscovered)
  - once as RED (discovered, on queue)
  - once as GRAY (finished)
- Each edge is considered twice (for an undirected graph)
- Each vertex is placed on the queue once
- Thus BFS runs in $O(|V|+|E|)$ time provided the graph is represented by an adjacency list structure
END OF LECTURE
APRIL 1, 2014
Applications

BFS traversal can be specialized to solve the following problems in $O(|V|+|E|)$ time:

- Compute the connected components of $G$
- Compute a spanning forest of $G$
- Find a simple cycle in $G$, or report that $G$ is a forest
- Given two vertices of $G$, find a path in $G$ between them with the minimum number of edges, or report that no such path exists
Outline

- BFS Algorithm
- BFS Application: Shortest Path on an unweighted graph
- Unweighted Shortest Path: Proof of Correctness
Application: Shortest Paths on an Unweighted Graph

- **Goal:** To recover the shortest paths from a source node $s$ to all other reachable nodes $v$ in a graph.
  - The length of each path and the paths themselves are returned.

- **Notes:**
  - There are an exponential number of possible paths
  - Analogous to level order traversal for trees
  - This problem is harder for general graphs than trees because of cycles!
Breadth-First Search

Input: Graph $G = (V, E)$ (directed or undirected) and source vertex $s \in V$.

Output:

$d[v] = \text{shortest path distance } \delta(s, v) \text{ from } s \text{ to } v, \forall v \in V$. 

$\pi[v] = u \text{ such that } (u, v) \text{ is last edge on a shortest path from } s \text{ to } v$.

- Idea: send out search ‘wave’ from $s$.
- Keep track of progress by colouring vertices:
  - Undiscovered vertices are coloured black
  - Just discovered vertices (on the wavefront) are coloured red.
  - Previously discovered vertices (behind wavefront) are coloured grey.
BFS Algorithm with Distances and Predecessors

BFS(G,s)

Precondition: G is a graph, s is a vertex in G
Postcondition: \( d[u] \) = shortest distance \( \delta[u] \) and 
\( \pi[u] = \) predecessor of \( u \) on shortest path from \( s \) to each vertex \( u \) in \( G \)

for each vertex \( u \in V[G] \)

\( d[u] \leftarrow \infty \)
\( \pi[u] \leftarrow \text{null} \)
\( \text{color}[u] = \text{BLACK} \) \//initialize vertex

\text{color}[s] \leftarrow \text{RED}

\( d[s] \leftarrow 0 \)

\text{Q.enqueue}(s)

while \( Q \neq \emptyset \)

\( u \leftarrow \text{Q.dequeue}() \)

for each \( v \in \text{Adj}[u] \) \//explore edge \((u,v)\)

\text{if} \ \text{color}[v] = \text{BLACK}

\( \text{color}[v] \leftarrow \text{RED} \)

\( d[v] \leftarrow d[u] + 1 \)

\( \pi[v] \leftarrow u \)

\text{Q.enqueue}(v)

\text{color}[u] \leftarrow \text{GRAY}
BFS

First-In First-Out (FIFO) queue
stores ‘just discovered’ vertices
BFS

Found
Not Handled
Queue

d=0

d=1

d=1

s

a

b

c

d

e

f
g

h

i

j

k

l

m

n

CSE 2011
Prof. J. Elder

Last Updated 2014-03-18 8:09 AM
BFS

Found
Not Handled
Queue

d=0
d=1
d=2

da=0
da=1
da=2

d=1
d=2

d=1
d=2

d=2
BFS

Found
Not Handled
Queue

d=0

d=1

d=2

d=1

d=2
BFS

Found
Not Handled
Queue

d=0

d=1

d=2

d=1

d=2

c  d=2

f

e  d=2

m

j

BFS

s

a

d

b

d

d

e

g

f

j

m

k

h

i

l

- 25 -

CSE 2011
Prof. J. Elder

Last Updated 2014-03-18 8:09 AM
BFS

Not Handled
Queue

d=0

d=1

d=2

d=2

Found

CSE 2011
Prof. J. Elder

Last Updated 2014-03-18 8:09 AM
BFS

Found
Not Handled
Queue

d=0
d=1
d=2
d=3

s
a
b
c
d
e
g
h
i
j
k
l
m

CSE 2011
Prof. J. Elder

Last Updated 2014-03-18 8:09 AM
BFS

Found
Not Handled
Queue

d=0

d=1

d=2

d=3

d=2

d=3

- 29 -

CSE 2011
Prof. J. Elder

Last Updated 2014-03-18 8:09 AM
BFS

Found
Not Handled
Queue

d=0

d=1

d=2

d=3

s

d=0

a

d=1

b

d=2

d=3

d=2

d=3

d=2

d=3

Not Handled

Queue

b

d=1

d=2

d=3

d=3
BFS

Found
Not Handled
Queue

d=0

d=1

d=2

d=3

d=2

d=3

d=2

h

i

j

k

l

m

ds=0

ds=1

ds=2

ds=3

CSE 2011
Prof. J. Elder
BFS

Found Not Handled Queue

s

d=0

d=1

d=2

d=3

d=4

d=4

a

d

c

d

f

d

h

k

i

j

m

b

g

l

k

d=4

d=3

d=2

d=3

CSE 2011
Prof. J. Elder

Last Updated 2014-03-18 8:09 AM
BFS

Found
Not Handled
Queue

s

d=0

d=1

d=3

d=4

d=2

d=4

d=3

d=4

a
b
c
f
d
e
g
j
h
i
m
k
l

CSE 2011
Prof. J. Elder

Last Updated 2014-03-18 8:09 AM
BFS

Found Not Handled Queue

d=0

d=1

d=2

d=3

d=4

Queue k
BFS

Found
Not Handled
Queue

s

d=0

d=1

d=2

d=3

d=4

d=5

a

d

d

d

e

g

b

f

c

h

j

i

l

m

k

CSE 2011
Prof. J. Elder

Last Updated 2014-03-18 8:09 AM
Breadth-First Search Algorithm: Properties

BFS(G, s)
Precondition: G is a graph, s is a vertex in G
Postcondition: \(d[u] = \) shortest distance \(\delta[u]\) and \(\pi[u] = \) predecessor of \(u\) on shortest paths from \(s\) to each vertex \(u\) in \(G\)

for each vertex \(u \in V[G]\)
  \(d[u] \leftarrow \infty\)
  \(\pi[u] \leftarrow \) null
  \(\text{color}[u] = \) BLACK  //initialize vertex

\(\text{colour}[s] \leftarrow \text{RED}\)
\(d[s] \leftarrow 0\)
Q.enqueue(s)

while \(Q \neq \emptyset\)
  \(u \leftarrow \) Q.dequeue()
  for each \(v \in \text{Adj}[u]\)  //explore edge \((u, v)\)
    if \(\text{color}[v] = \) BLACK
      \(\text{colour}[v] \leftarrow \) RED
      \(d[v] \leftarrow d[u] + 1\)
      \(\pi[v] \leftarrow u\)
      Q.enqueue(v)

\(\text{colour}[u] \leftarrow \text{GRAY}\)

- Q is a FIFO queue.
- Each vertex assigned finite \(d\) value at most once.
- Q contains vertices with \(d\) values \(\{i, \ldots, i, i+1, \ldots, i+1\}\)
- \(d\) values assigned are monotonically increasing over time.
Breadth-First-Search is **Greedy**

- Vertices are handled (and finished):
  - in order of their discovery (FIFO queue)
  - Smallest $d$ values first
Outline

- BFS Algorithm
- BFS Application: Shortest Path on an unweighted graph
- Unweighted Shortest Path: Proof of Correctness
Correctness

Basic Steps:

The shortest path to $u$ has length $d$ & there is an edge from $u$ to $v$.

There is a path to $v$ with length $d + 1$. 
Correctness: Basic Intuition

- When we discover $v$, how do we know there is not a shorter path to $v$?
  - Because if there was, we would already have discovered it!
Correctness: More Complete Explanation

- Vertices are discovered in order of their distance from the source vertex $s$.

- Suppose that at time $t_1$ we have discovered the set $V_d$ of all vertices that are a distance of $d$ from $s$.

- Each vertex in the set $V_{d+1}$ of all vertices a distance of $d+1$ from $s$ must be adjacent to a vertex in $V_d$.

- Thus we can correctly label these vertices by visiting all vertices in the adjacency lists of vertices in $V_d$. 
Inductive Proof of BFS

Suppose at step $i$ that the set of nodes $S_i$ with distance $\delta(v) \leq d_i$ have been discovered and their distance values $d[v]$ have been correctly assigned.

Further suppose that the queue contains only nodes in $S_i$ with $d$ values of $d_i$.

Any node $v$ with $\delta(v) = d_i + 1$ must be adjacent to $S_i$.

Any node $v$ adjacent to $S_i$ but not in $S_i$ must have $\delta(v) = d_i + 1$.

At step $i + 1$, all nodes on the queue with $d$ values of $d_i$ are dequeued and processed. In so doing, all nodes adjacent to $S_i$ are discovered and assigned $d$ values of $d_i + 1$.

Thus after step $i + 1$, all nodes $v$ with distance $\delta(v) \leq d_i + 1$ have been discovered and their distance values $d[v]$ have been correctly assigned.

Furthermore, the queue contains only nodes in $S_i$ with $d$ values of $d_i + 1$. 
Correctness: Formal Proof

Input: Graph $G = (V, E)$ (directed or undirected) and source vertex $s \in V$.

Output:

$d[v] = \text{distance } \delta(v) \text{ from } s \text{ to } v, \forall v \in V.$

$\pi[v] = u \text{ such that } (u, v) \text{ is last edge on shortest path from } s \text{ to } v.$

Two-step proof:

On exit:

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

2. $d[v] \neq \delta(s, v) \forall v \in V$
Claim 1. $d$ is never too small: $d[v] \geq \delta(s,v) \\forall v \in V$

Proof: There exists a path from $s$ to $v$ of length $\leq d[v]$.

By Induction:

Suppose it is true for all vertices thus far discovered (red and grey). $v$ is discovered from some adjacent vertex $u$ being handled.

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

since each vertex $v$ is assigned a $d$ value exactly once, it follows that on exit, $d[v] \geq \delta(s,v) \forall v \in V$. 
Claim 1. $d$ is never too small: $d[v] \geq \delta(s,v) \forall v \in V$

Proof: There exists a path from $s$ to $v$ of length $\leq d[v]$.

BFS(G,s)
Precondition: $G$ is a graph, $s$ is a vertex in $G$
Postcondition: $d[u] =$ shortest distance $\delta[u]$ and
$\pi[u] =$ predecessor of $u$ on shortest paths from $s$ to each vertex $u$ in $G$

for each vertex $u \in V[G]$

\begin{align*}
  d[u] &\leftarrow \infty \\
  \pi[u] &\leftarrow \text{null} \\
  \text{color}[u] &\leftarrow \text{BLACK} \quad \text{//initialize vertex}
\end{align*}

color[s] $\leftarrow$ RED

d[s] $\leftarrow$ 0

Q.enqueue(s)

while Q $\neq \emptyset$

\begin{align*}
  u &\leftarrow Q\text{.dequeue}() \\
  \text{for each } v \in \text{Adj}[u] \quad \text{//explore edge (u,v)} \\
  \quad \text{if color}[v] = \text{BLACK} \\
  \quad \quad \text{color}[v] \leftarrow \text{RED} \\
  \quad \quad d[v] &\leftarrow d[u] + 1 \\
  \quad \quad \pi[v] &\leftarrow u \\
  \quad \quad Q\text{.enqueue}(v)
\end{align*}

color[u] $\leftarrow$ GRAY

\[ \delta(s,v) \geq \delta(s,u) + 1 \geq \delta(s,v) \]
Claim 2. \( d \) is never too big: \( d[v] \leq \delta(s,v) \forall v \in V \)

Proof by contradiction:

Suppose one or more vertices receive a \( d \) value greater than \( \delta \).

Let \( v \) be the vertex with minimum \( \delta(s,v) \) that receives such a \( d \) value.

Suppose that \( v \) is discovered and assigned this \( d \) value when vertex \( x \) is dequeued.

Let \( u \) be \( v \)'s predecessor on a shortest path from \( s \) to \( v \).

Then
\[
\delta(s,v) < d[v] \\
\rightarrow \delta(s,v) - 1 < d[v] - 1 \\
\rightarrow d[u] < d[x]
\]

Recall: vertices are dequeued in increasing order of \( d \) value.

\( \rightarrow \) \( u \) was dequeued before \( x \).

\( \rightarrow d[v] = d[u] + 1 = \delta(s,v) \) \hspace{1em} Contradiction!
Correctness

Claim 1. $d$ is never too small: $d[v] \geq \delta(s,v) \forall v \in V$

Claim 2. $d$ is never too big: $d[v] \leq \delta(s,v) \forall v \in V$

$\Rightarrow d$ is just right: $d[v] = \delta(s,v) \forall v \in V$
Progress?  ➚ On every iteration one vertex is processed (turns gray).

BFS(G,s)

Precondition: G is a graph, s is a vertex in G

Postcondition: d[u] = shortest distance δ[u] and

π[u] = predecessor of u on shortest paths from s to each vertex u in G

for each vertex u ∈ V[G]

    d[u] ← ∞
    π[u] ← null
    color[u] = BLACK //initialize vertex

colour[s] ← RED

q[d[s] ← 0

Q.enqueue(s)

while Q ≠ ∅

    u ← Q.dequeue()

    for each v ∈ Adj[u] //explore edge (u,v)
        if color[v] = BLACK
            colour[v] ← RED
            d[v] ← d[u] + 1
            π[v] ← u
            Q.enqueue(v)

    colour[u] ← GRAY
The shortest path problem has the optimal substructure property:

- Every subpath of a shortest path is a shortest path.

The optimal substructure property is a hallmark of both greedy and dynamic programming algorithms.

- allows us to compute both shortest path distance and the shortest paths themselves by storing only one $d$ value and one predecessor value per vertex.
Recovering the Shortest Path

For each node $v$, store predecessor of $v$ in $\pi(v)$.

Predecessor of $v$ is $\pi(v) = u$. 

\[ s = \pi(\pi(\pi(\pi( v)))) \]
Recovering the Shortest Path

PRINT-PATH(G, s, v)
Precondition: s and v are vertices of graph G
Postcondition: the vertices on the shortest path from s to v have been printed in order
if v = s then
    print s
else if π[v] = NIL then
    print "no path from" s "to" v "exists"
else
    PRINT-PATH(G, s, π[v])
    print v
BFS Algorithm without Colours

BFS(G,s)

Precondition: G is a graph, s is a vertex in G

Postcondition: predecessors $\pi[u]$ and shortest distance $d[u]$ from s to each vertex $u$ in G has been computed

for each vertex $u \in V[G]$
  
  $d[u] \leftarrow \infty$

  $\pi[u] \leftarrow \text{null}$

$d[s] \leftarrow 0$

Q.enqueue(s)

while Q $\neq \emptyset$

  $u \leftarrow \text{Q.dequeue()}$

  for each $v \in \text{Adj}[u]$  //explore edge $(u,v)$

    if $d[v] = \infty$

      $d[v] \leftarrow d[u] + 1$

      $\pi[v] \leftarrow u$

      Q.enqueue(v)
Outline

- BFS Algorithm
- BFS Application: Shortest Path on an unweighted graph
- Unweighted Shortest Path: Proof of Correctness