CS 341: ALGORITHMS

Trevor Brown

trevor.brown@uwaterloo.ca

DC 2338, Office hour M3-4pm
THIS TIME

• Breadth-first search (BFS)
  • Algorithm and example
  • Complexity
  • Application exercise: connected components via BFS
• Application: finding **shortest paths** in a graph
  • Proof of optimal distances
• Application: testing whether a graph is **bipartite**
Breadth-first Search

Algorithm: $BFS(G, s)$
for each $v \in V(G)$
do $\begin{cases} 
  \text{colour}[v] \leftarrow \text{white} \\
  \pi[v] \leftarrow \emptyset \\
  \text{colour}[s] \leftarrow \text{gray}
\end{cases}$
InitializeQueue($Q$)
Enqueue($Q, s$)
while $Q \neq \emptyset$
do $\begin{cases} 
  u \leftarrow \text{Dequeue}(Q) \\
  \text{for each } v \in \text{Adj}[u]
  \begin{cases} 
    \text{if } \text{colour}[v] = \text{white} \\
    \text{do } \begin{cases} 
      \text{if } \text{colour}[v] = \text{gray} \\
      \pi[v] \leftarrow u \\
      \text{Enqueue}(Q, v)
    \end{cases} \\
    \text{then } \begin{cases} 
      \text{colour}[u] \leftarrow \text{black}
    \end{cases}
  \end{cases}
\end{cases}$

Start processing node $u$

Discover (enqueue) starting node $s$

Discover (enqueue) node $v$

Finish processing node $u$
We run \textit{breadth-first search} with $s = 1$ on the previous graph:

$\text{colour}[1] \leftarrow \text{grey}, Q = [1]$

$u \leftarrow 1, Q = []$

$v \leftarrow 2, \text{colour}[2] \leftarrow \text{grey}, \pi[2] \leftarrow 1, Q = [2]$

$v \leftarrow 3, \text{colour}[3] \leftarrow \text{grey}, \pi[3] \leftarrow 1, Q = [2, 3]$

$\text{colour}[1] \leftarrow \text{black}$

$u \leftarrow 2, Q = [3]$

$v \leftarrow 4, \text{colour}[4] \leftarrow \text{grey}, \pi[4] \leftarrow 2, Q = [3, 4]$

$v \leftarrow 5, \text{colour}[5] \leftarrow \text{grey}, \pi[5] \leftarrow 2, Q = [3, 4, 5]$

$\text{colour}[2] \leftarrow \text{black}$

$u \leftarrow 3, Q = [4, 5]$

$v \leftarrow 7, \text{colour}[7] \leftarrow \text{grey}, \pi[7] \leftarrow 3, Q = [4, 5, 7]$

$v \leftarrow 8, \text{colour}[8] \leftarrow \text{grey}, \pi[8] \leftarrow 3, Q = [4, 5, 7, 8]$

$\text{colour}[3] \leftarrow \text{black}$
Example (cont.)

\[ u \leftarrow 4, Q = [5, 7, 8] \]
\[ \text{colour}[4] \leftarrow \text{black} \]

\[ u \leftarrow 5, Q = [7, 8] \]
\[ v \leftarrow 6, \text{colour}[6] \leftarrow \text{grey}, \pi[6] \leftarrow 5, Q = [7, 8, 6] \]
\[ \text{colour}[5] \leftarrow \text{black} \]

\[ u \leftarrow 7, Q = [8, 6] \]
\[ \text{colour}[7] \leftarrow \text{black} \]

\[ u \leftarrow 8, Q = [6] \]
\[ \text{colour}[8] \leftarrow \text{black} \]

\[ u \leftarrow 6, Q = [] \]
\[ \text{colour}[6] \leftarrow \text{black} \]

The tree edges are 12, 13, 24, 25, 37, 38, 56.
Breadth-first Search

Algorithm: $BFS(G, s)$
for each $v \in V(G)$
    do $\{$
        $\text{colour}[v] \leftarrow \text{white}$
        $\pi[v] \leftarrow \emptyset$
        $\text{colour}[s] \leftarrow \text{gray}$
    $\}$
InitializeQueue($Q$)
Enqueue($Q$, $s$)
while $Q \neq \emptyset$
    do $\{$
        $u \leftarrow \text{Dequeue}(Q)$
        for each $v \in \text{Adj}[u]$
            do $\{$
                if $\text{colour}[v] = \text{white}$
                    then $\{$
                        $\text{colour}[v] \leftarrow \text{gray}$
                        $\pi[v] \leftarrow u$
                        $\text{Enqueue}(Q, v)$
                    $\}$
                $\}$
        $\text{colour}[u] \leftarrow \text{black}$
    $\}$
Properties of Breadth-first Search

A vertex is **white** if it is **undiscovered**.

A vertex is **gray** if it has been **discovered**, but we are still processing its adjacent vertices.

A vertex becomes **black** when all the adjacent vertices have been processed.

If $G$ is **connected**, then every vertex eventually is coloured black and every vertex $v \neq s$ has a unique predecessor $\pi[v]$ in the BFS tree.

When we explore an edge $\{u, v\}$ starting from $u$:

- if $v$ is **white**, then $uv$ is a **tree edge** and $\pi[v] = u$ is the **predecessor** of $v$ in the BFS tree
- otherwise, $uv$ is a **cross edge**.

The BFS tree consists of all the tree edges.
APPLICATION: UNDIRECTED CONNECTED COMPONENTS
Can you think of a way to use BFS to count how many connected components there are?
APPLICATION:
FINDING SHORTEST PATHS
Game AI: path finding with waypoints

Divide game world into linear paths, then send game characters in straight lines between waypoints.

Use BFS to find shortest sequence of waypoints (with fewest waypoints).
User interfaces: shortest path to a mouse cursor around obstacles
Game AI: path finding in a grid-graph

Starting to get into the details

How to represent a grid graph?
Game AI: path finding in a grid-graph

Starting to get into the details

How to represent a grid graph?

BFS from here

Track optimal distance from start to each node
Quick preview: doing this quickly, and on a larger scale

Running BFS (plus some more stuff at the end...)
SHORTEST PATHS VIA BFS

Algorithm: BFS\((G, s)\)

\[
\text{for each } v \in V(G) \text{ do } \begin{cases} 
\text{colour}[v] \leftarrow \text{white} \\
\pi[v] \leftarrow 0
\end{cases}
\]

\[
\text{colour}[s] \leftarrow \text{gray} \\
\text{dist}[s] \leftarrow 0
\]

InitializeQueue\((Q)\)

Enqueue\((Q, s)\)

\text{while } Q \neq \emptyset \\
\text{do } \begin{cases} 
\text{u} \leftarrow \text{Dequeue}(Q) \\
\text{for each } v \in \text{Adj}[u] \\
\text{do } \begin{cases} 
\text{if colour}[v] = \text{white} \text{ then } \begin{cases} 
\text{colour}[v] = \text{gray} \\
\pi[v] \leftarrow u \\
\text{Enqueue}(Q, v) \\
\text{dist}[v] \leftarrow \text{dist}[u] + 1
\end{cases}
\end{cases}
\end{cases}
\]

\text{colour}[u] \leftarrow \text{black}
OUTPUTTING AN OPTIMAL PATH
HOW TO OUTPUT AN ACTUAL PATH

• Suppose you want to output a path from $s$ to $v$ with minimum distance (not just the distance to $v$)

• Algorithm (what do you think?)
  • Similar to extracting an answer from a DP array!
  • Work backwards through the predecessors
    • Start at $v$ (set $cur := v$)
    • Inside a loop, visit the predecessor node
      (by printing $cur$ and then setting $cur := \pi[cur]$)
    • Stop when you hit $s$ (when $cur = s$)

• Note: path is printed in reverse! Solution?
How to find the actual path
Each time you visit a predecessor, push it into a stack. I.e., push $v = 5$, then push $\pi[v] = 4$, then push $\pi[\pi[v]] = 3$, then 2, ... At the end, pop all off the stack. This gives 0, 1, 2, ..., 5 = the path!
BFS: PROOF OF OPTIMAL DISTANCES
**Algorithm: BFS(G, s)**

```plaintext
for each v ∈ V(G) do
  { colour[v] ← white
    π[v] ← 0
  }

colour[s] ← gray

dist[s] ← 0

InitializeQueue(Q)
Enqueue(Q, s)

while Q ≠ ∅
  do
    u ← Dequeue(Q)
    for each v ∈ Adj[u]
      do
        if colour[v] = white then
          { colour[v] = gray
            π[v] ← u
            Enqueue(Q, v)
            dist[v] ← dist[u] + 1
          }

    colour[u] ← black
```

**Discover (enqueue) node u**

**Start processing node u**

**Observe:** A node must be discovered before it can be processed!

**We use u < v to denote “u discovered before v”**

**Corollary:** if u < v then u is processed before v (and vice versa)

**Discover (enqueue) node v**

**Let’s use d_u as shorthand for dist[u]**

**Finish processing node u**

**Definitions: Discovering and Processing a Node**
Lemma 1: if \( u < v \) then \( d_u \leq d_v \)

Proof by contradiction: assume \( \exists u, v \) such that \( u < v \) but \( d_u > d_v \)

- Let \( v \) be the earliest discovered node such that, for some \( u \), we have \( u < v \) and \( d_u > d_v \)
- Consider predecessors \( u' \) of \( u \) and \( v' \) of \( v \)
- Note \( d_{u'} = d_u - 1 \), and \( d_u > d_v \), so \( d_{u'} \geq d_v \)
- Also, \( d_{v'} = d_v - 1 \), so \( d_{v'} < d_u \)
- It turns out \( d_{v'} < d_{u'} \) implies \( v' < u' \). Why?
- Since \( v' < u' \), we know \( v' \) is processed before \( u' \)
- And, \( v \) is discovered while processing \( v' \), which is before \( u' \) is processed, which is when \( u \) is discovered. So \( v < u \)!
WE STOPPED HERE
Lemma 2: if there is an edge \( \{u, v\} \), then \( |d_u - d_v| \leq 1 \)

Proof by cases: WLOG suppose \( u < v \)

- Case 1: \( v \) is white when we process \( u \)
  - Then \( d_v = d_u + 1 \). QED
Lemma 2: if there is an edge \{u, v\}, then \( |d_u - d_v| \leq 1 \)

Proof by cases: WLOG suppose \( u < v \)

• Case 2: \( v \) is grey when we process \( u \)
  • Since \( v \) is not white, we did not discover it from \( u \)
  • We discovered \( v \) earlier when processing some \( v' \neq u \)
  • Since \( v' \) was processed before \( u \), we have \( v' < u \)
  • So, by Lemma 1 we have \( d_{v'} \leq d_u \).
  • Also note \( d_v = d_{v'} + 1 \). Rearrange to get \( d_{v'} = d_v - 1 \).
  • Substituting \( d_{v'} \) into \( d_{v'} \leq d_u \) we get \( d_u \geq d_v - 1 \)
  • Also, since \( u < v \), Lemma 1 implies \( d_u \leq d_v \)
  • So, \( d_v - 1 \leq d_u \leq d_v \). QED
Lemma 2: if there is an edge \{u, v\}, then |d_u - d_v| \leq 1

Proof by cases: WLOG suppose u < v

• Case 3: \(v\) is **black** when we process \(u\)
  • Then \(v\) is finished processing before \(u\)
  • Therefore \(v < u\)
  • This contradicts our assumption that \(u < v\). QED
Theorem: $d_v$ is the length of the shortest path from $s$ to $v$

- Let $\delta_v$ denote the length of the shortest path from $s$ to $v$.
- The path $v \to \pi[v] \to \pi[\pi[v]] \to \cdots \to s$ has distance $d_v$. So $\delta_v \leq d_v$.
- We show $\delta_v \geq d_v$ by induction on the value of distance $\delta_v$.
- Base case: suppose $\delta_v = 0$. Then $v = s$ and $d_v = 0$, so $\delta_v \geq d_v$.
- Inductive hypothesis: 
  \begin{align*}
  \text{if } \delta_v = k - 1 & \text{ then } \delta_v \geq d_v. \end{align*}
  (So, suppose $d_v = k$)
- Then $\exists$ a shortest path $s \to v_1 \to \cdots \to v_{k-1} \to v_k = v$ with length $\delta_v = k$.
- By Lemma 2, we have $d_v \leq d_{v_{k-1}} + 1$, so $d_v \leq d_{v_{k-1}} + 1$.
- By the inductive hypothesis, we have $\delta_{v_{k-1}} \geq d_{v_{k-1}}$ so $d_{v_{k-1}} = \delta_{v_{k-1}} = k - 1$.
- So $d_v \leq d_{v_{k-1}} + 1$ becomes $d_v \leq (k - 1) + 1 = k$.
- Rearranging to $k \geq d_v$ and substituting $\delta_v = k$ we get $\delta_v \geq d_v$. QED
APPLICATION: TESTING WHETHER A GRAPH IS BIPARTITE
BIPARTITE GRAPHS AND BFS

A graph is **bipartite** if the vertex set can be partitioned as $V = X \cup Y$, in such a way that all edges have one endpoint in $X$ and one endpoint in $Y$.

A graph is bipartite if and only if it does not contain an **odd cycle**.

**BFS** can be used to test if a graph is bipartite:

if we encounter an edge $\{u, v\}$ with $\text{dist}[u] = \text{dist}[v]$, then $G$ is not bipartite, whereas

if no such edge is found, then define $X = \{u : \text{dist}[u] \text{ is even}\}$ and $Y = \{u : \text{dist}[u] \text{ is odd}\}$; then $X, Y$ forms a bipartition.

Complexity?

Done on **black board only**. Take notes!