CS 341: ALGORITHMS
Trevor Brown
trevor.brown@uwaterloo.ca
DC 2338, Office hour M3-4pm
THIS TIME

• BFS
  • Complexity
  • Proof of optimal distances
• Depth-first search (DFS)
BFS: TIME AND SPACE COMPLEXITY
Algorithm: \( \text{BFS}(G, s) \)

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

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

\text{InitializeQueue}(Q) \\
\text{Enqueue}(Q, s) \\
\text{while } Q \neq \emptyset \\
\text{do } \begin{cases} 
  u &\leftarrow \text{Dequeue}(Q) \\
  \text{for each } v \in \text{Adj}[u] \text{ do } \begin{cases} 
    \text{if } \text{colour}[v] = \text{white} \text{ then } \begin{cases} 
      \text{colour}[v] &\leftarrow \text{gray} \\
      \pi[v] &\leftarrow u \\
      \text{Enqueue}(Q, v) \\
      \text{dist}[v] &\leftarrow \text{dist}[u] + 1
    \end{cases} \\
    \text{colour}[u] &\leftarrow \text{black}
  \end{cases}
\end{cases}
\]
BFS: PROOF OF OPTIMAL DISTANCES
**Definitions: Discovering and Processing a Node**

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

1. For each \( v \in V(G) \) do:
   - \( \text{color}[v] \leftarrow \text{white} \)
   - \( \pi[v] \leftarrow 0 \)
2. \( \text{colour}[s] \leftarrow \text{gray} \)
3. \( \text{dist}[s] \leftarrow 0 \)
4. InitializeQueue\((Q)\)
5. Enqueue\((Q, s)\)
6. While \( Q \neq \emptyset \):
   - \( u \leftarrow \text{Dequeue}(Q) \)
   - For each \( v \in \text{Adj}[u] \):
     - If \( \text{color}[v] = \text{white} \) then:
       - \( \text{color}[v] = \text{gray} \)
       - \( \pi[v] \leftarrow u \)
       - Enqueue\((Q, v)\)
       - \( \text{dist}[v] \leftarrow \text{dist}[u] + 1 \)
   - \( \text{colour}[u] \leftarrow \text{black} \)
7. Finish processing node \( u \)

We use \( u < v \) to denote “\( u \) discovered before \( v \)”

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

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

Let’s use \( d_u \) as shorthand for \( \text{dist}[u] \)

Discover (enqueue) node \( u \)

Finish processing node \( u \)
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 $v < u$!
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
PROOF OF OPTIMAL BFS DISTANCES: CLAIM 2 OF 3

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: \( \text{"if } \delta_v = k - 1 \text{ then } \delta_v \geq d_v.\text{"} \)
  - We prove: \( \text{"if } \delta_v = k \text{ then } \delta_v \geq d_v.\text{"} \) (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 \( d_{v_k} \leq d_{v_{k-1}} + 1 \), 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
OUTPUTTING AN OPTIMAL PATH
Game AI: path finding in a grid-graph

How to represent a grid graph?

BFS from here

SCORE: 0
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
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?
doing this quickly, on a large scale
Recall the Pac-man example

Shortest path to here?

BFS from here

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!
DEPTH FIRST SEARCH (DFS)
Depth-first Search of a Directed Graph

A depth-first search uses a stack (or recursion) instead of a queue. We define predecessors and colour vertices as in BFS. It is also useful to specify a discovery time $d[v]$ and a finishing time $f[v]$ for every vertex $v$.

We increment a time counter every time a value $d[v]$ or $f[v]$ is assigned. We eventually visit all the vertices, and the algorithm constructs a depth-first forest.

The complexity of depth-first search is ?
Algorithm: $DFS(G)$

for each $v \in V(G)$
  do $egin{cases} 
  \text{colour}[v] \leftarrow \text{white} \\
  \pi[v] \leftarrow \emptyset 
  \end{cases}$
  time $\leftarrow 0$

for each $v \in V(G)$
  do \begin{cases} 
  \text{if colour}[v] = \text{white} \\
  \text{then } DFSvisit(v)
  \end{cases}

Algorithm: $DFSvisit(v)$

\begin{align*}
\text{colour}[v] & \leftarrow \text{gray} \\
\text{time} & \leftarrow \text{time} + 1 \\
\text{d}[v] & \leftarrow \text{time} \\
\text{comment: } d[v] \text{ is the discovery time for vertex } v \\
\text{for each } w \in \text{Adj}[v] \\
  \text{do } \begin{cases} 
  \text{if colour}[w] = \text{white} \\
  \text{then } \begin{cases} 
  \pi[w] \leftarrow v \\
  DFSvisit(w)
  \end{cases}
  \end{cases}
\text{colour}[v] \leftarrow \text{black} \\
\text{time} \leftarrow \text{time} + 1 \\
\text{f}[v] \leftarrow \text{time} \\
\text{comment: } f[v] \text{ is the finishing time for vertex } v
\end{align*}
PREVIEW: VISUALIZING DFS

Example of Depth-first Search

Consider the directed graph on vertex set \{1, 2, 3, 4, 5, 6\} with the following adjacency lists:

\[
\begin{align*}
\text{Adj}[1] & : 2 \rightarrow 3 \\
\text{Adj}[2] & : 3 \\
\text{Adj}[3] & : 4 \\
\text{Adj}[4] & : 2 \\
\text{Adj}[5] & : 4 \rightarrow 6 \\
\text{Adj}[6] & : \\
\end{align*}
\]
Initial call: \textit{DFSvisit}(1), recursive calls: \textit{DFSvisit}(2), \textit{DFSvisit}(3), \textit{DFSvisit}(4).

Initial call: \textit{DFSvisit}(5), recursive call: \textit{DFSvisit}(6).

The depth-first forest consists of two trees. One tree has arcs 12, 23, 34 (initial call from \textit{DFSvisit}(1)) and the other tree has arc 56 (initial call from \textit{DFSvisit}(5)).