CS 341: ALGORITHMS

Trevor Brown

trevor.brown@uwaterloo.ca

DC 2338, Office hour M3-4pm
THIS TIME

• Breadth-first search (BFS)
  • Finish proof of optimal distances
  • Application: testing whether a graph is \textit{bipartite}

• Depth-first search (DFS)
  • \textbf{Classifying} graph edges
  • Application: testing whether a graph is \textit{acyclic}
  • Application: topological sort (\textit{time permitting})
BFS: PROOF OF OPTIMAL DISTANCES
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 $\text{dist}[u]$

### Algorithm: $\text{BFS}(G, s)$

- **for each** $v \in V(G)$ **do**
  - $\text{colour}[v] \leftarrow \text{white}$
  - $\pi[v] \leftarrow \emptyset$
  - $\text{colour}[s] \leftarrow \text{gray}$
  - $\text{dist}[s] \leftarrow 0$

- **InitializeQueue**$(Q)$
- $\text{Enqueue}(Q, s)$
- **while** $Q \neq \emptyset$
  - $u \leftarrow \text{Dequeue}(Q)$
  - **for each** $v \in \text{Adj}[u]$
    - **do**
      - **if** $\text{colour}[v] = \text{white}$ **then**
        - $\text{colour}[v] \leftarrow \text{gray}$
        - $\pi[v] \leftarrow u$
        - $\text{Enqueue}(Q, v)$
        - $\text{dist}[v] \leftarrow \text{dist}[u] + 1$
      - $\text{colour}[u] \leftarrow \text{black}$

Finish processing node $u$
PROOF OF OPTIMAL BFS DISTANCES: CLAIM 1 OF 3

Lemma 1: if \( u < v \) then \( d_u \leq d_v \)

Proved last time.
Lemma 2: if there is an edge \( \{u, v\} \), then \( |d_u - d_v| \leq 1 \)

Proof: 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 \( d_u \leq d_{v'} \) 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 $v \rightarrow \pi[v] \rightarrow \pi[\pi[v]] \rightarrow \cdots \rightarrow s$ 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:** "if $\delta_v = k - 1$ then $\delta_v \geq d_v$.”
    - We prove: "if $\delta_v = k$ then $\delta_v \geq d_v$.” (So, suppose $d_v = k$)
    - Then $\exists$ a shortest path $s \rightarrow v_1 \rightarrow \cdots \rightarrow v_{k-1} \rightarrow v_k = v$ 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
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 $dist[u] = dist[v]$, then $G$ is not bipartite, whereas
- If no such edge is found, then define $X = \{u : dist[u] \text{ is even}\}$ and $Y = \{u : dist[u] \text{ is odd}\}$; then $X, Y$ forms a bipartition.

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

Complexity?
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 $\boxed{\text{?}}$
**Algorithm: DFS(G)**

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

\[\text{time} \leftarrow 0\]

\[
\text{for each } v \in V(G) \quad \text{do} \quad \{
\begin{align*}
\text{if } \text{colour}[v] & = \text{white} \quad \text{then} \quad \text{DFSvisit}(v)
\end{align*}
\}
\]

**Algorithm: DFSvisit(v)**

\[
\begin{align*}
\text{colour}[v] & \leftarrow \text{gray} \\
\text{time} & \leftarrow \text{time} + 1 \\
\text{d}[v] & \leftarrow \text{time}
\end{align*}
\]

comment: \(\text{d}[v]\) is the discovery time for vertex \(v\)

\[
\text{for each } w \in \text{Adj}[v] \quad \text{do} \quad \{
\begin{align*}
\text{if } \text{colour}[w] & = \text{white} \\
\text{then} \quad \{
\begin{align*}
\pi[w] & \leftarrow v \\
\text{DFSvisit}(w)
\end{align*}
\}
\end{align*}
\}
\]

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

\[\text{time} \leftarrow \text{time} + 1\]

\[\text{f}[v] \leftarrow \text{time}\]

comment: \(\text{f}[v]\) is the finishing time for vertex \(v\)
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:

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

Initial call: $DFSvisit(5)$, recursive call: $DFSvisit(6)$.

The depth-first forest consists of two trees. One tree has arcs 12, 23, 34 (initial call from $DFSvisit(1)$) and the other tree has arc 56 (initial call from $DFSvisit(5)$).
BASIC DFS PROPERTIES TO REMEMBER

- Nodes start **white**
- A node $v$ turns **gray** when it is **discovered**, which is when the first call to $DFSVisit(v)$ happens
- **After** $v$ is turned **gray**, we recurse on its neighbours
- After recursing on **all neighbours**, we turn $v$ **black**
  - Recursive calls on neighbours end first, so neighbours turn black first

Also gets a **discovery time** $d[v]$ at this point

Also gets a **finish time** $f[v]$ at this point
Algorithm: $DFS(G)$
for each $v \in V(G)$
do $\{$
  colour$[v] \leftarrow$ white
  $\pi[v] \leftarrow \emptyset$
  $time \leftarrow 0$
do $\{$
  if colour$[v] = $ white
  then $DFSvisit(v)$
for each $v \in V(G)$
do $\{$
  if colour$[v] = $ white
  then $\{$
    $\pi[w] \leftarrow v$
    $DFSvisit(w)$
  $\}$
  colour$[v] \leftarrow$ black
  $time \leftarrow time + 1$
  $f[v] \leftarrow$ time
  comment: $f[v]$ is the finishing time for vertex $v$
$\}$

Complexity with adjacency lists is $O(n + m)$

Home exercise: complexity with adjacency matrix?
Classification of Edges in Depth-first Search

What are the edge types in the example graph?

$uv$ is a **tree edge** if $u = \pi[v]$

$uv$ is a **forward edge** if it is not a tree edge, and $v$ is a descendant of $u$ in a tree in the depth-first forest

$uv$ is a **back edge** if $u$ is a descendant of $v$ in a tree in the depth-first forest

any other edge is a **cross edge**.

Can we classify edges **without** inspecting the DFS forest? Perhaps using $d[\ldots]$, $f[\ldots]$ and/or $colour[\ldots]$?
DEFINITIONS

• **Definition:** we use $I_u$ to denote $(d[u], f[u])$, which we call the **interval of** $u$

• **Definition:** $v$ is **white-reachable from** $u$ if there is a path from $u$ to $v$ containing **only white nodes**
EXPLORING D[], F[] AND COLOUR[]

- **Observe:** every node \( v \) that is **white-reachable** from \( u \) when we call \( DFSVisit(u) \) (discover \( u \)) becomes **gray after \( u \)** and **black before \( u \)** (so \( I_v \) is **nested inside** \( I_u \))

Start \( DFSVisit(u) \) and colour \( u \) grey

Perform \( DFSVisit \) calls recursively...

Colour \( u \) black and return from \( DFSVisit(u) \)

Note: \( v \) is discovered during the recursive \( DFSVisit \) calls, which is equivalent to \( v \) being a **descendent of \( u \)** in the DFS forest
MORE GENERALLY

• **Theorem:** Let $u, v$ be any nodes. The following statements are all equivalent.
  • $(v$ is discovered during $DFSVisit(u)$)
  • $(v$ is **white-reachable** from $u$ when we call $DFSVisit(u)$)
  • $(v$ is a **descendant of** $u$ in the DFS forest)
  • $(v$ turns grey after $u$ and black before $u$)
  • $(I_v$ nested inside $I_u$)
DFS inspects **every edge** in the graph. When DFS inspects an edge \{u, v\}, the colour of v and relationship between the intervals of u and v determine the **edge type**.

<table>
<thead>
<tr>
<th>edge type</th>
<th>colour of v</th>
<th>discovery/finish times</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree</td>
<td>Q1?</td>
<td>Q2?</td>
</tr>
<tr>
<td>forward</td>
<td>Q4?</td>
<td>Q3?</td>
</tr>
<tr>
<td>back</td>
<td>Q6?</td>
<td>Q5?</td>
</tr>
<tr>
<td>cross</td>
<td>Q8?</td>
<td>Q7?</td>
</tr>
</tbody>
</table>

Recall: (v is discovered during DFSVisit(u)) ⇔ (v is **white-reachable** from u when we call DFSVisit(u)) ⇔ (v is a **descendant of** u in the DFS forest) ⇔ (v turns grey after u and black before u) ⇔ (I_v nested inside I_u)

\( v \) discovered **during** DFSVisit(u) but **not** at the first level of recursion (else \{u, v\} is a tree edge).

So the recursive call in DFSVisit(u) that discovered v must **finish** (and set v black) **before** DFSVisit(u) inspects \{u, v\}.
WE STOPPED HERE
**USEFUL FACT: PARENTHESIS THEOREM**

- **Theorem**: for each pair of nodes $u, v$ the intervals of $u$ and $v$ are either **disjoint** or **nested**.

- **Proof**: Suppose the intervals are **not disjoint**.
  - Then either $d[v] \in I_u$ or $d[u] \in I_v$.
  - WLOG suppose $d[v] \in I_u$.
  - Then $v$ is discovered during $DFSVisit(u)$, so $v$ is white-reachable from $u$ when $DFSVisit(u)$ starts.
  - So, $v$ must turn black before $u$, so $f[v] < f[u]$.
  - So the intervals are nested. QED.
DFS inspects **every edge** in the graph. When DFS inspects an edge \( \{u, v\} \), the colour of \( v \) and relationship between the intervals of \( u \) and \( v \) determine the **edge type**.

<table>
<thead>
<tr>
<th>edge type</th>
<th>colour of ( v )</th>
<th>discovery/finish times</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree</td>
<td>white</td>
<td>( d[u] &lt; d[v] &lt; f[v] &lt; f[u] )</td>
</tr>
<tr>
<td>forward</td>
<td>black</td>
<td>( d[u] &lt; d[v] &lt; f[v] &lt; f[u] )</td>
</tr>
<tr>
<td>back</td>
<td>gray</td>
<td>( d[v] &lt; d[u] &lt; f[u] &lt; f[v] )</td>
</tr>
<tr>
<td>cross</td>
<td>Q8?</td>
<td>Q7?</td>
</tr>
</tbody>
</table>

**Recall:** \( v \) is discovered during \( DFSVisit(u) \)

\[ \iff \ (v \text{ is white-reachable from } u \text{ when we call } DFSVisit(u)) \]

\[ \iff \ (v \text{ is a descendant of } u \text{ in the DFS forest}) \]

\[ \iff \ (I_v \text{ nested inside } I_u) \]

If \( I_u \) were earlier, then \( v \) would be **discovered before \( u \) finishes** (because of edge \( \{u, v\} \)), so intervals would not be disjoint!

So, \( I_v \) must be earlier.

Intervals \( I_u \) and \( I_v \) must be **disjoint**. But which is **earlier**?

\( v \) is **not** a descendent, and **not** an ancestor.