Lectures 33-35

- Depth-first Search
- Topological Sorting

Adam Smith
DFS pseudocode

- Maintain a global counter `time`
- Maintain for each vertex `v`
  - Two timestamps:
    - `v.d` = time first discovered
    - `v.f` = time when finished
  - “color”: `v.color`
    - `white` = unexplored
    - `gray` = in process
    - `black` = finished
- Parent `v.\pi` in DFS tree
As DFS progresses, every vertex has a color:

- **WHITE**: undiscovered
- **GRAY**: discovered, but not finished (not done exploring from it)
- **BLACK**: finished (have found everything reachable from it)

**Discovery and finishing times:** Unique integers from 1 to \(2^{|V|}\).

For all \(u, v \in V\): \(d < f\).

In other words, \(1 < d < f < 2^{|V|}\).

**Pseudocode**

Uses a global timestamp \(time\).

\[
\text{DFS}(G) \\
\text{for each } u \in G.V \quad \begin{align*}
  & u.color = \text{WHITE} \\
  & time = 0 \\
  & \text{for each } u \in G.V \quad \begin{align*}
    & \text{if } u.color == \text{WHITE} \\
    & \quad \text{DFS-Visit}(G, u)
  \end{align*}
\]\

\[
\text{DFS-Visit}(G, u) \\
\begin{align*}
  & time = time + 1 \\
  & u.d = time \\
  & u.color = \text{GRAY} \quad \text{// discover } u \\
  & \text{for each } v \in G.Adj[u] \quad \text{// explore } (u, v) \\
  & \quad \text{if } v.color == \text{WHITE} \\
  & \quad \quad \text{DFS-Visit}(v) \\
  & u.color = \text{BLACK} \\
  & time = time + 1 \\
  & u.f = time \quad \text{// finish } u
\end{align*}
\]

**Note:** recursive function different from first call…

**Exercise:** change code to set “parent” pointer?
DFS example

- **T** = tree edge (drawn in red)
- **F** = forward edge (to a descendant in DFS forest)
- **B** = back edge (to an ancestor in DFS forest)
- **C** = cross edge (goes to a vertex that is neither ancestor nor descendant)
DFS with adjacency lists

**Outer code** runs once, takes time $O(n)$ (not counting time to execute recursive calls)

**Recursive calls:**
- Run once per vertex
- time = $O(\text{degree}(v))$

**Total:** $O(m+n)$

---

DFS($G$)

```
for each $u \in G.V$
    $u.color = \text{WHITE}$
    time = 0
for each $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT($G, u$)
```

DFS-VISIT($G, u$)

```
time = time + 1
$u.d = time$
$u.color = \text{GRAY}$  // discover $u$
for each $v \in G.Adj[u]$  // explore ($u, v$)
    if $v.color == \text{WHITE}$
        DFS-VISIT($v$)
$u.color = \text{BLACK}$
time = time + 1
$u.f = time$  // finish $u$
```
DFS Edge Types

• Tree
• Forward
• Backward
• Cross

• Classifying edges according to type gives info about graph structure
Review Question

• Give an algorithm that takes a directed graph $G$ and finds a cycle if one exists. (Hint: use DFS)
Topological Sort
Directed Acyclic Graphs

**Def.** An **DAG** is a directed graph that contains no directed cycles.

**Ex.** Precedence constraints: edge \((v_i, v_j)\) means \(v_i\) must precede \(v_j\).

**Def.** A **topological order** of a directed graph \(G = (V, E)\) is an ordering of its nodes as \(v_1, v_2, ..., v_n\) so that for every edge \((v_i, v_j)\) we have \(i < j\).
Precedence constraints. Edge \((v_i, v_j)\) means task \(v_i\) must occur before \(v_j\).

Applications.
- Course prerequisite graph: course \(v_i\) must be taken before \(v_j\).
- Compilation: module \(v_i\) must be compiled before \(v_j\). Pipeline of computing jobs: output of job \(v_i\) needed to determine input of job \(v_j\).
- Getting dressed
Recall from book

- Every DAG has a topological order

- If G graph has a topological order, then G is a DAG.
Review

- Suppose your run DFS on a DAG $G=(V,E)$
- True or false?
  - Sorting by **discovery** time gives a topological order
  - Sorting by **finish** time gives a topological order
Shortest Paths
Shortest Path Problem

• **Input:**
  – Directed graph $G = (V, E)$.
  – Source node $s$, destination node $t$.
  – for each edge $e$, length $\ell(e) = \text{length of } e$.
  – length path = sum of edge lengths

• **Find:** shortest directed path from $s$ to $t$.

Length of path $(s, 2, 3, 5, t)$ is $9 + 23 + 2 + 16 = 50$. 

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Rough algorithm (Dijkstra)

• Maintain a set of **explored nodes** $S$ whose shortest path distance $d(u)$ from $s$ to $u$ is known.
• Initialize $S = \{ s \}$, $d(s) = 0$.
• Repeatedly choose unexplored node $v$ which minimizes
  \[
  \pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell(e),
  \]
• add $v$ to $S$, and set $d(v) = \pi(v)$.
Rough algorithm (Dijkstra)

• Maintain a set of explored nodes $S$ whose shortest path distance $d(u)$ from $s$ to $u$ is known.
• Initialize $S = \{ s \}$, $d(s) = 0$.
• Repeatedly choose unexplored node $v$ which minimizes
  $$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell(e),$$
• add $v$ to $S$, and set $d(v) = \pi(v)$.

Invariant: $d(u)$ is known for all vertices in $S$

BFS with weighted edges
Demo of Dijkstra’s Algorithm

Graph with nonnegative edge lengths:
Demo of Dijkstra’s Algorithm

Initialize:

\[ Q: \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty
\end{array} \]

\[ S: \{\} \]

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Demo of Dijkstra’s Algorithm

“$A$” $\leftarrow$ \textbf{Extract-Min}(Q):

$Q$: $A$
$B$
$C$
$D$
$E$

0
$\infty$
$\infty$
$\infty$
$\infty$

$S$: $\{A\}$
Demo of Dijkstra’s Algorithm

Explore edges leaving $A$:

$Q: \begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty \\
\end{array}$

$S: \{ A \}$
Demo of Dijkstra’s Algorithm

“C” ← \text{\textsc{Extract-Min}}(Q):

\begin{itemize}
  \item \text{Q:} \quad A & B & C & D & E \\
    0 & \infty & \infty & \infty & \infty \\
    10 & 3 & \infty & \infty & \infty \\
\end{itemize}

\begin{itemize}
  \item \text{S:} \quad \{A, \ C\}
\end{itemize}
Demo of Dijkstra’s Algorithm

Explore edges leaving $C$:

$Q$: \[
\begin{array}{cccccc}
A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty \\
10 & 3 & \infty & \infty & \infty \\
7 & 11 & 5 & & \\
\end{array}
\]

$S$: \{ $A$, $C$ \}

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Demo of Dijkstra’s Algorithm

“E” ← \textbf{\textsc{Extract-Min}}(Q):

\[ Q: \begin{array}{cccccc}
& A & B & C & D & E \\
0 & \infty & \infty & \infty & \infty & \infty \\
10 & 10 & 3 & \infty & \infty & \infty \\
7 & 7 & 11 & 5 &
\end{array} \]

\[ S: \{ A, C, E \} \]
Demo of Dijkstra’s Algorithm

Explore edges leaving $E$:

$Q$: $A$ $B$ $C$ $D$ $E$

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>3</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>11</td>
<td>11</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

$S$: $\{A, C, E\}$
Demo of Dijkstra’s Algorithm

“B” ← \textbf{EXTRACT-MIN}(Q):

\begin{tabular}{lllll}
\textbf{Q:} & A & B & C & D & E \\
\hline
0 & ∞ & ∞ & ∞ & ∞ & ∞ \\
10 & 3 & ∞ & ∞ & ∞ & ∞ \\
7 & 7 & 11 & 5 & \\
7 & 7 & 11 & 5 & \\
\end{tabular}

\textbf{S:} \{ A, C, E, B \}
Demo of Dijkstra’s Algorithm

Explore edges leaving B:

Q:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>3</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>11</td>
<td>5</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

S: \{ A, C, E, B \}

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Demo of Dijkstra’s Algorithm

“D” ← $\text{EXTRACT-MIN}(Q)$:

\[ Q:\begin{array}{cccccc}
  & A & B & C & D & E \\
  A & 0 & \infty & \infty & \infty & \infty \\
  B & 10 & 3 & \infty & \infty & \infty \\
  C & 7 & 11 & 5 & 3 & 2 \\
  D & 7 & 11 & 9 & 2 & 2 \\
  E & 7 & 11 & 9 & 2 & 2 \\
\end{array} \]

\[ S: \{ A, C, E, B, D \} \]
Implementation

• For unexplored nodes, maintain $\pi(v) = \min_{e=(u,v): u \in S} d(u) + \ell(e)$.
  – Next node to explore = node with minimum $\pi(v)$.
  – When exploring $v$, for each edge $e = (v,w)$, update
    $$\pi(w) = \min \{ \pi(w), \pi(v) + \ell(e) \}.$$  

• Efficient implementation: Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.  

Pseudocode for Dijkstra($G, \ell$)

\[
\begin{align*}
    d[s] & \leftarrow 0 \\
    \text{for each } v \in V - \{s\} & \text{ do } d[v] \leftarrow \infty; \pi[v] \leftarrow \infty \\
    S & \leftarrow \emptyset \\
    Q & \leftarrow V \quad \triangleright Q \text{ is a priority queue maintaining } V - S, \\
    & \text{keyed on } \pi[v] \\
\text{while } Q \neq \emptyset & \text{ do } u \leftarrow \text{Extract-Min}(Q) \\
    S & \leftarrow S \cup \{u\}; d[u] \leftarrow \pi[u] \\
    \text{for each } v \in \text{Adjacency-list}[u] & \text{ do if } \pi[v] > \pi[u] + \ell(u, v) \\
    & \text{ then } \pi[v] \leftarrow d[u] + \ell(u, v) \\
\end{align*}
\]

explore edges leaving $v$

Implicit Decrease-Key
Analysis of Dijkstra

while $Q \neq \emptyset$
  do $u \leftarrow \text{Extract-Min}(Q)$
     $S \leftarrow S \cup \{u\}$
     for each $v \in \text{Adj}[u]$
       do if $d[v] > d[u] + w(u, v)$
          then $d[v] \leftarrow d[u] + w(u, v)$

Handshaking Lemma $\Rightarrow \leq m$ implicit Decrease-Key’s.

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Dijkstra</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap $\dagger$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExtractMin</td>
<td>$n$</td>
<td>$n$</td>
<td>$\log n$</td>
<td>HW3</td>
<td>$\log n$</td>
</tr>
<tr>
<td>DecreaseKey</td>
<td>$m$</td>
<td>$1$</td>
<td>$\log n$</td>
<td>HW3</td>
<td>$1$</td>
</tr>
<tr>
<td>Total</td>
<td>$n^2$</td>
<td>$n \log n$</td>
<td>$m \log \frac{m}{n} \cdot n$</td>
<td>$m + n \log n$</td>
<td></td>
</tr>
</tbody>
</table>

$\dagger$ Individual ops are amortized bounds

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Invariant. For each node u ∈ S, d(u) is the length of the shortest path from s to u.

Proof: (by induction on |S|)

• Base case: |S| = 1 is trivial.

• Inductive hypothesis: Assume for |S| = k ≥ 1.
  – Let v be next node added to S, and let (u,v) be the chosen edge.
  – The shortest s-u path plus (u,v) is an s-v path of length π(v).
  – Consider any s-v path P. We'll see that it's no shorter than π(v).
  – Let (x,y) be the first edge in P that leaves S, and let P' be the subpath to x.
  – P + (x,y) has length ≤ d(x)+ ℓ(x,y)≤ π(y)≤ π(v)
Proof of Correctness

• More formally, we are proving two statements.

• Invariant: At the end of each phase of the algorithm (i.e. after each execution of while loop):
  – For all vertices reachable in one step from S: \( \pi(v) = \min_{u \in S} \pi(u) + \ell(u,v) \)
  – For all vertices in S: \( \pi(v) = d(s,v) \).
Review Question

• Is Dijsktra’s algorithm correct with negative edge weights?