Lecture 13:
• Depth-First Search
• Start Shortest Paths

Oct 23, 2018
Midterm Stats
Midterm Grade Distribution

This midterm is only 15% of your course grade!
Midterm Grade Distribution

• Lots of points missed for asymptotics, induction:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 Asymptotics</td>
<td>76%</td>
</tr>
<tr>
<td>#2 Recurrences</td>
<td>93%</td>
</tr>
<tr>
<td>#3 Induction</td>
<td>65%</td>
</tr>
<tr>
<td>#4 Divide-and-Conquer</td>
<td>72%</td>
</tr>
<tr>
<td>#5 Stable Matching</td>
<td>90%</td>
</tr>
<tr>
<td>#6 Dynamic Programming</td>
<td>63%</td>
</tr>
</tbody>
</table>

• These topics will come up on future exams!
Homework Grade Distribution

Homework is 45% of your course grade!
I have not dropped your lowest grade yet
Depth-First Search (DFS)
Exploring a Graph

• **Problem:** Is there a path from \( s \) to \( t \)?
• **Idea:** Explore all nodes reachable from \( s \).

• Two different search techniques:
  • **Breadth-First Search:** explore nearby nodes before moving on to farther away nodes
  • **Depth-First Search:** follow a path until you get stuck, then go back
Depth-First Search

\[ G = (V, E) \] is a graph
\[ \text{explored}[u] = 0 \ \forall u \]

DFS(u):
\[ \text{explored}[u] = 1 \]

\[ \text{for } ((u, v) \text{ in } E): \]
\[ \text{if } (\text{explored}[v] = 0): \]
\[ \text{parent}[v] = u \]
\[ \text{DFS}(v) \]
Depth-First Search

- **Fact:** The parent-child edges form a (directed) tree

- **Each edge has a type:**
  - **Tree edges:** \((u, a), (u, c), (c, b)\)
    - These are the edges that explore new nodes
  - **Forward edges:** \((u, b)\)
    - Ancestor to descendant
  - **Backward edges:** \((a, u)\)
    - Descendant to ancestor
  - **Cross edges:** \((c, a)\)
    - No ancestral relation

Backward edges give directed cycles
Ask the Audience

- DFS this graph starting from node \( a \)
  - Search in alphabetical order
  - Label edges as \{ tree, forward, backward, cross \}
Post-Ordering

$G = (V,E)$ is a graph
explored$[u] = 0 \ \forall u$

DFS$\text{(}u\text{)}$:
  explored$[u] = 1$
  for ((u,v) in E):
    if (explored[v]=0):
      parent$[v] = u$
      DFS$\text{(}v\text{)}$
  post-visit$\text{(}u\text{)}$

• Maintain a counter $\text{clock}$, initially set $\text{clock} = 1$
• post-visit$\text{(}u\text{)}$:
  set postorder$[u]$=$\text{clock}$, clock=clock+1
Pre-Ordering

\[ G = (V,E) \text{ is a graph} \]

\[ \text{explored}[u] = 0 \quad \forall u \]

\[ \text{DFS}(u): \]
\[ \quad \text{explored}[u] = 1 \]

\[ \text{pre-visit}(u) \]

\[ \text{for } ((u,v) \text{ in } E): \]
\[ \quad \text{if } (\text{explored}[v]=0): \]
\[ \quad \quad \text{parent}[v] = u \]
\[ \quad \text{DFS}(v) \]

- Maintain a counter \textbf{clock}, initially set \textbf{clock} = 1
- \textbf{pre-visit}(u):
  - set \text{preorder}[u]=\text{clock}, \text{ clock}=\text{clock}+1
Ask the Audience

• Compute the **post-order** of this graph
  • DFS from \( a \), search in alphabetical order

<table>
<thead>
<tr>
<th>Vertex</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-Order</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
• **Observation:** if \text{postorder}[u] < \text{postorder}[v] then (u,v) is a backward edge
• **Observation**: if $\text{postorder}[u] < \text{postorder}[v]$ then $(u,v)$ is a backward edge

  • DFS($u$) can’t finish until its children are finished
    • If $(u,v)$ is a tree edge, then $\text{postorder}[u] > \text{postorder}[v]$ 
    • If $(u,v)$ is a forward edge, then $\text{postorder}[u] > \text{postorder}[v]$

  • If $\text{postorder}[u] < \text{postorder}[v]$, then DFS($u$) finishes before DFS($v$), thus DFS($v$) is not called by DFS($u$)

  • When we ran DFS($u$), we must have had $\text{explored}[v] = 1$
    • Thus, DFS($v$) started before DFS($u$)

  • DFS($v$) started before DFS($u$) but finished after
    • Can only happen for a backward edge
Fast Topological Ordering
Topological Ordering (TO)

- **DAG**: A directed graph with no directed cycles
- Any DAG can be **topologically ordered**
  - Label nodes $v_1, \ldots, v_n$ so that $(v_i, v_j) \in E \implies j > i$
• **Claim**: ordering nodes by decreasing postorder gives a topological ordering

• **Proof**:
  
  • A DAG has no backward edges
  
  • Suppose this is not a topological ordering
    
    • That means there exists an edge \((u, v)\) such that \(\text{postorder}[u] < \text{postorder}[v]\)
    
    • We showed that any such \((u, v)\) is a backward edge
    
    • But there are no backward edges, contradiction!
Topological Ordering (TO)

- **DAG**: A directed graph with no directed cycles
- Any DAG can be **topologically ordered**
  - Label nodes $v_1, \ldots, v_n$ so that $(v_i, v_j) \in E \Rightarrow j > i$
- Can compute a TO in $O(n + m)$ time using DFS
  - Reverse of post-order is a topological order
Shortest Paths
Navigation
Weighted Graphs

• **Definition:** A weighted graph $G = (V, E, \{w(e)\})$
  - $V$ is the set of vertices
  - $E \subseteq V \times V$ is the set of edges
  - $w_e \in \mathbb{R}$ are edge weights/lengths/capacities
  - Can be directed or undirected

• **Today:**
  - Directed graphs (one-way streets)
  - Strongly connected (there is always some path)
  - Non-negative edge lengths ($\ell(e) \geq 0$)
Shortest Paths

- The **length** of a path \( P = v_1 - v_2 - \cdots - v_k \) is the sum of the edge lengths
  \[
  \ell(P) = \sum_{e \in P} \ell(e)
  \]

- The **distance** \( d(s, t) \) is the length of the shortest path from \( s \) to \( t \).

- **Shortest Path**: given nodes \( s, t \in V \), find the shortest path from \( s \) to \( t \).

- **Single-Source Shortest Paths**: given a node \( s \in V \), find the shortest paths from \( s \) to **every** \( t \in V \).
Structure of Shortest Paths

• If $(u, v) \in E$, then $d(s, v) \leq d(s, u) + \ell(u, v)$ for every node $s \in V$

• If $(u, v) \in E$, and $d(s, v) = d(s, u) + \ell(u, v)$ then there is a shortest $s \sim v$-path ending with $(u, v)$
Compare to BFS

• **Thm:** BFS finds distances from \( s \) to other nodes
  • \( L_i \) contains all nodes at distance \( i \) from \( s \)
  • Nodes not in any layer are not reachable from \( s \)
Compare to BFS

 Vertex  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
 Parent  | - | 1 | 1 | 2 | 2 | 5 | 3 | 3
Dijkstra’s Algorithm

- Dijkstra’s Shortest Path Algorithm is a modification of BFS for non-negatively weighted graphs

- Informal Version:
  - Maintain a set $S$ of explored nodes
  - Maintain an upper bound on distance
    - If $u$ is explored, then we know $d(u)$ (Key Invariant)
    - If $u$ is explored, and $(u, v)$ is an edge, then we know $d(v) \leq d(u) + \ell(u, v)$
  - Explore the “closest” unexplored node
  - Repeat until we’re done

$s$ is the source
Dijkstra’s Algorithm: Demo

Diagram of a weighted graph with labeled edges and nodes.
Dijkstra’s Algorithm: Demo

Initialize

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} \\
\hline
\text{d}_0(u) & 0 & \infty & \infty & \infty & \infty \\
\hline
\end{array}
\]

\[S = \{\}\]
Dijkstra’s Algorithm: Demo

Explore A

<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>$d_0(u)$</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$d_1(u)$</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

$S = \{A\}$
• First node we explore is $S$

• Second node we explore is just the neighbor of $S$ with the smallest distance

\[ \Rightarrow \ell(p') > 10 \]
Dijkstra’s Algorithm: Demo

Explore C

<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>$d_0(u)$</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$d_1(u)$</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$d_2(u)$</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>11</td>
<td>5</td>
</tr>
</tbody>
</table>

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

Explore E

\[
\begin{array}{cccccc}
 & A & B & C & D & E \\
\hline
d_0(u) & 0 & \infty & \infty & \infty & \infty \\
d_1(u) & 0 & 10 & 3 & \infty & \infty \\
d_2(u) & 0 & 7 & 3 & 11 & 5 \\
d_3(u) & 0 & 7 & 3 & 11 & 5 \\
\end{array}
\]

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

Explore B

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

Don’t need to explore D

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

Maintain parent pointers so we can find the shortest paths

<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>d₀(u)</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>d₁(u)</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>d₂(u)</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>d₃(u)</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>d₄(u)</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>