Lecture 11:
• Shortest Paths: BFS, Start Dijkstra

Feb 24, 2020
Shortest Paths: Breadth-First Search
Exploring a Graph

- **Problem:** Is there a path from $s$ to $t$?
- **Idea:** Explore all nodes reachable from $s$.

- Two different search techniques:
  - **Depth-First Search:** follow a path until you get stuck, then go back
  - **Breadth-First Search:** explore all nearby nodes before moving on to farther away nodes
    - Finds the shortest path from $s$ to $t$!
Breadth-First Search (BFS)

- **Informal Description**: start at $s$, find neighbors of $s$, find neighbors of neighbors of $s$, and so on...

- BFS Tree:
  - $L_0 = \{s\}$
  - $L_1 = \text{all neighbors of } L_0$
  - $L_2 = \text{all neighbors of } L_1 \text{ that are not in } L_0, L_1$
  - $L_3 = \text{all neighbors of } L_2 \text{ that are not in } L_0, L_1, L_2$
  - ...
  - $L_d = \text{all neighbors of } L_{d-1} \text{ that are not in } L_0, \ldots, L_{d-1}$
  - Stop when $L_{d+1}$ is empty
Example

- BFS this graph from $s = 1$

- Red edges are "tree edges"
- Red edges give paths from $s$ to $t$
- Blue edges are either $L_i \leftrightarrow L_i$ or $L_i \leftrightarrow L_{i+1}$
Breadth-First Search Implementation

BFS(G = (V,E), s):

Let explored[v] ← false ∀v, explored[s] ← true
Let layer[v] ← ∞ ∀v, layer[s] ← 0
Let parent[v] ← ⊥ ∀v
Let i ← 0, L₀ = {s}, T ← ∅

While (Lᵢ is not empty):
    Initialize new layer Lᵢ₊₁
    For (u in Lᵢ):
        For ((u,v) in E):
            If (explored[v] = false):
                explored[v] ← true,
                layer[v] ← i+1
                parent[v] ← u (Add (u,v) to T)
                Add v to Lᵢ₊₁
    i ← i+1
BFS Running Time (Adjacency List)

BFS(G = (V,E), s):
Let explored[v] ← false ∀v, explored[s] ← true
Let layer[v] ← ∞ ∀v, layer[s] ← 0
Let parent[v] ← ⊥ ∀v
Let i ← 0, L₀ = {s}, T ← ∅

While (Lᵢ is not empty):
  Initialize new layer Lᵢ₊₁
  For (u in Lᵢ):
    For ((u,v) in E):
      If (explored[v] = false):
        explored[v] ← true,
        layer[v] ← i+1
        parent[v] ← u
        Add v to Lᵢ₊₁
  i ← i+1

O(n)
Shortest Paths via BFS

- **Definition:** the distance between \( s, t \) is the number of edges on the shortest path from \( s \) to \( t \).
- **Thm:** BFS finds distances from \( s \) to other nodes.
  - \( L_i \) contains all nodes at distance \( i \) from \( s \).
**Shortest Paths via BFS**

- **Definition:** the distance between \( s, t \) is the number of edges on the shortest path from \( s \) to \( t \)
- **Thm:** BFS finds distances from \( s \) to other nodes
  - \( L_i \) contains all nodes at distance \( i \) from \( s \)

**Base Cases:**
- \( L_0 \) is obvious
- \( L_1 \) is obvious (\( L_1 \) contains all neighbors of \( s \))

**Induction:** If true for \( L_0, L_1, \ldots, L_i \), then true for \( L_{i+1} \)

Suppose \( u \) is such that \( d(s, u) = i + 1 \)

\[
\begin{array}{c}
\text{\( \circ \) } \\
\text{\( S \) } \xrightarrow{i \text{ hops}} \text{\( \square \) } \xrightarrow{} \text{\( u \) }
\end{array}
\]

By induction, \( v \) is in \( L_i \). Therefore \( u \) is in \( L_{i+1} \).
Shortest Paths via BFS

- **Definition:** the distance between $s, t$ is the number of edges on the shortest path from $s$ to $t$.
- **Thm:** BFS finds distances from $s$ to other nodes and the tree edges give the shortest $s$ to $t$ path.
  - Can find distances and shortest path tree in time $O(n + m)$... then can find a shortest path in time $O(n)$.

Tree edges give shortest paths.
Shortest Paths via BFS

• **Definition:** the distance between $s, t$ is the number of edges on the shortest path from $s$ to $t$

• **Thm:** BFS finds distances from $s$ to other nodes and the tree edges give the shortest $s$ to $t$ path
  - Can find distances and shortest path tree in time $O(n + m)$... then can find a shortest path in time $O(n)$
Shortest Paths:
Dijkstra
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:

$$l(P) = \sum_{e \in P} l(e)$$

• 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 \(s \rightsquigarrow v\)-path ending with \((u, v)\)
**Dijkstra's Algorithm**

- Maintain an upper bound on \( d(s, t) \) \( \forall t \)
  
  \[
  d[s] = 0 \quad d[t] = \infty \text{ for } t \neq s
  \]

- Explore neighbors of \( s \)

- Find another node [with the smallest \( d[u] \) of all unexplored nodes]
  
  Explore neighbors of that node

- Repeat until all nodes are explored
Dijkstra’s Algorithm: Demo
Dijkstra’s Algorithm: Demo

Initialize

<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>
</tbody>
</table>

$S = \{\}$

Set of explored nodes
Dijkstra’s Algorithm: Demo

Explore A

\[ S = \{A\} \]
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

<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>
<tr>
<td>$d_3(u)$</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>11</td>
<td>5</td>
</tr>
</tbody>
</table>

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

Explore B

<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>
<tr>
<td>$d_3(u)$</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>$d_4(u)$</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

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

Don’t need to explore D

\[ S = \{A, C, E, B, D\} \]
Maintain parent pointers so we can find the shortest paths
Correctness of Dijkstra

- **Warmup 0:** initially, $d_0(s)$ is the correct distance

- **Warmup 1:** after exploring the first node $v$, $d_1(v)$ is the correct distance

  If $(s,v)$ is the shortest edge starting at $s$. Then $d(s,v) = \ell(s,v)$

  Any other $s \rightarrow v$ path has length $> S$, so it is not a shorter path
Correctness of Dijkstra

- **Invariant**: after we explore the i-th node, $d_i(v)$ is correct for every $v \in S$

- We just argued the invariant holds after we’ve explored the 1st and 2nd nodes
Correctness of Dijkstra

- **Invariant**: after we explore the i-th node, \( d_i(v) \) is correct for every \( v \in S \)

- **Proof**:

  Want to show that \( d_i(v) = d_i(u) + l(u,v) \) is the shortest path

\[
\begin{align*}
l(P') &= l(P_s, x) + l(x \rightarrow y) + l(P_y, v) \\
&\geq l(P_s, x) + l(x \rightarrow y) \\
&\geq d_i(x) + l(x \rightarrow y) \\
&\geq d_i(y) \\
&\geq d_i(v) \\
&= l(P)
\end{align*}
\]

\( [l(e) > 0] \)
\( [x \text{ is explored}] \)
\( [x \text{ is explored}] \)
\( [\text{I chose } v, \text{ not } y] \)
Implementing Dijkstra

\[ \text{Dijkstra}(G = (V,E,\{\ell(e)\}, s)) : \]
\[ d[s] \leftarrow 0, \ d[u] \leftarrow \infty \text{ for every } u \neq s \]
\[ \text{parent}[u] \leftarrow \perp \text{ for every } u \]
\[ Q \leftarrow V \quad // Q \text{ holds the unexplored nodes} \]

\text{While (Q is not empty)}:
\[ u \leftarrow \text{argmin}_{w \in Q} d[w] \quad // \text{Find closest unexplored} \]
\[ \text{Remove } u \text{ from } Q \]

\[ // \text{Update the neighbors of } u \]
\[ \text{For } ((u,v) \text{ in } E) : \]
\[ \text{If } (d[v] > d[u] + \ell(u,v)) : \]
\[ d[v] \leftarrow d[u] + \ell(u,v) \]
\[ \text{parent}[v] \leftarrow u \]

\text{Return } (d, \text{parent})
Implementing Dijkstra (Naïvely)

1. Need to explore all $n$ nodes
2. Each exploration requires:
   2a. Finding the unexplored node $u$ with smallest distance
   2b. Updating the distance for each neighbor of $u$

\[
\sum_{u \in V} O(n + \deg(u) + 1) = O(n^2 + m)
\]