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

- BFS this graph from $s = 1$
Breadth-First Search Implementation

\[\text{BFS}(G = (V,E), s) :\]
\begin{align*}
\text{Let } & \text{explored}[v] \leftarrow \text{false} \quad \forall v, \text{explored}[s] \leftarrow \text{true} \\
\text{Let } & \text{layer}[v] \leftarrow \infty \quad \forall v, \text{layer}[s] \leftarrow 0 \\
\text{Let } & \text{parent}[v] \leftarrow \bot \quad \forall v \\
\text{Let } & i \leftarrow 0, \text{L}_{0} = \{s\}, \text{T} \leftarrow \emptyset \\
\end{align*}

While (\text{L}_i \text{ is not empty}):
\begin{align*}
& \text{Initialize new layer } \text{L}_{i+1} \\
& \text{For (u in L}_i\text{):} \\
& \quad \text{For ((u,v) in E):} \\
& \quad \quad \text{If (explored}[v] = \text{false}): \\
& \quad \quad \quad \text{explored}[v] \leftarrow \text{true}, \\
& \quad \quad \quad \text{layer}[v] \leftarrow i+1 \\
& \quad \quad \quad \text{parent}[v] \leftarrow u \\
& \quad \quad \text{Add v to } \text{L}_{i+1} \\
& \quad i \leftarrow i+1
\end{align*}
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 ← Ø

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
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 $s$
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 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.

- 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 \(s \leadsto v\)-path ending with \((u, v)\)
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>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

$S = \{\}$
Dijkstra’s Algorithm: Demo

Explore A

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>

$S = \{A\}$
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>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>$d_1(u)$</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>∞</td>
<td>∞</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

\[
S = \{A, C, E\}
\]
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

\[
\begin{align*}
S &= \{A, C, E, B, D\}
\end{align*}
\]
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>$d_0(u)$</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>
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
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 1\textsuperscript{st} and 2\textsuperscript{nd} nodes
Correctness of Dijkstra

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

• **Proof:**
Implementing Dijkstra

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

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

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

Return (d, parent)
Implementing Dijkstra (Naïvely)

• Need to explore all $n$ nodes
• Each exploration requires:
  • Finding the unexplored node $u$ with smallest distance
  • Updating the distance for each neighbor of $u$
Priority Queues / Heaps
Priority Queues

• Need a data structure $Q$ to hold key-value pairs

• Need to support the following operations
  • $\text{Insert}(Q,k,v)$: add a new key-value pair
  • $\text{Lookup}(Q,k)$: return the value of some key
  • $\text{ExtractMin}(Q)$: identify the key with the smallest value
  • $\text{DecreaseKey}(Q,k,v)$: reduce the value of some key
Priority Queues

• **Naïve approach:** linked lists

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>11</td>
</tr>
<tr>
<td>c</td>
<td>12</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
</tr>
<tr>
<td>h</td>
<td>36</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td>g</td>
<td>20</td>
</tr>
<tr>
<td>k</td>
<td>42</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>f</td>
<td>8</td>
</tr>
</tbody>
</table>

  • Insert takes $O(1)$ time
  • ExtractMin, DecreaseKey take $O(n)$ time

• **Binary Heaps:** implement all operations in $O(\log n)$ time where $n$ is the number of keys
Heaps

- Organize key-value pairs as a binary tree
  - Later we’ll see how to store pairs in an array
- **Heap Order:** If $a$ is the parent of $b$, then $v(a) \leq v(b)$

Each node represents a key-value pair
Implementing ExtractMin
Implementing ExtractMin
Implementing ExtractMin

Fails to be a heap in one place
Implementing ExtractMin
Implementing ExtractMin
Implementing ExtractMin

• Three steps:
  • Pull the minimum from the root
  • Move the last element to the root
  • Repair the heap-order (heapify down)
Implementing DecreaseKey
Implementing DecreaseKey

Fails to be a heap
Implementing DecreaseKey

• Two steps:
  • Change the key
  • Repair the heap-order (heapify up)
Implementing Insert

Fails to be a heap
Implementing Insert

Insert fails to be a heap.
Implementing Insert

- Two steps:
  - Put the new key in the last location
  - Repair the heap-order (heapify up)
Implementation Using Arrays

- Maintain an array \( V \) holding the values
- Maintain an array \( K \) mapping keys to values
  - Can find the value for a given key in \( O(1) \) time
- For any node \( i \)
  - \( \text{LeftChild}(i) = 2i \)
  - \( \text{RightChild}(i) = 2i+1 \)
  - \( \text{Parent}(i) = \lfloor i/2 \rfloor \)
Binary Heaps

• **Heapify:**
  - O(1) time to fix a single triple
  - With n keys, might have to fix O(log n) triples
  - Total time to heapify is O(log n)

• **Lookup** takes O(1) time

• **ExtractMin** takes O(log n) time

• **DecreaseKey** takes O(log n) time

• **Insert** takes O(log n) time
Implementing Dijkstra with Heaps

\[ \text{Dijkstra}(G = (V,E,\{\ell(e)\}, s):} \]
- Let \( Q \) be a new heap
- Let \( \text{parent}[u] \leftarrow \bot \) for every \( u \)
- Insert(\( Q,s,0 \)), Insert(\( Q,u,\infty \)) for every \( u \neq s \)

While (\( Q \) is not empty):
- \((u,d[u]) \leftarrow \text{ExtractMin}(Q)\)

For \(((u,v) \text{ in } E)\):
- \(d[v] \leftarrow \text{Lookup}(Q,v)\)
- If \(d[v] > d[u] + \ell(u,v)\):
  - DecreaseKey(\( Q,v,d[u] + \ell(u,v) \))
  - \(\text{parent}[v] \leftarrow u\)

Return \((d, \text{parent})\)
Dijkstra Summary:

- **Dijkstra's Algorithm** solves **single-source shortest paths** in non-negatively weighted graphs
  - Algorithm can fail if edge weights are negative!

- **Implementation:**
  - A **priority queue** supports all necessary operations
  - Implement priority queues using **binary heaps**
  - Overall running time of Dijkstra: $O(m \log n)$

- Compare to BFS