CSCI 241

Lecture 22
Dijkstra’s Single-Source Shortest Paths Algorithm
Announcements

• Nick has office hours today 2-4 CF 163.

• hours.txt should contain a single integer. No more, no less.

• Nontrivial code sections (such as rebalance and rotations) should have comments!

• Test that your code compiles and runs on the command line:
  • git clone your_repo_url && cd your_repo_name
  • make test2
  • make test1 # yes, you have to run test2 before test1
  • make test3
Goals

• Know what a weighted graph is.

• Understand the intuition behind Dijkstra’s shortest paths algorithm.

• Be able to execute Dijkstra’s algorithm manually on a graph.
Weighted Graphs

• Like a normal graph, but edges have weights.

• Formally: a graph \((V,E)\) with an accompanying weight function \(w: E \rightarrow \mathbb{R}\)
  • may be directed or undirected.

• Informally: label edges with their weights

• Representation:
  • adjacency list - store weight of \((u,v)\) with \(v\) the node in \(u\)’s list
  • adjacency matrix - store weight in matrix entry for \((u,v)\)
Paths in Weighted Graphs

• The length (or weight) of a path in a weighted graph is the sum of the edge weights along that path.

• **ABCD**: What’s the length of the shortest path from 3 to 6?
  A. 7
  B. 8
  C. 9
  D. 10
Computing Shortest Paths in Unweighted Graphs

- Perform a breadth-first search (that’s it!)
- BFS visits nodes in order of “hop distance”, or path length!
- BFS(1):

```
0 1 2 3 4 5 6
1 2 3 3 2 1
```
Computing Shortest Paths in Weighted Graphs

BFS doesn’t visit nodes in order of shortest path length:

(edge weights)
(shortest path length from node 1)
Dijkstra’s Shortest Paths: History

- When Dijkstra designed the algorithm in 1956 (at age 26), most people were programming in assembly language.

- Fortran was the only high-level language, and it wasn’t quite finished at the time.

- Big-O analysis had not been thought of yet. In his paper, Dijkstra says, “my solution is preferred to another one … “the amount of work to be done seems considerably less.”
Dijkstra’s Shortest Paths: Subpaths

• Fact: **subpaths** of shortest paths are shortest paths

![Diagram showing a path from u to w through v]

• Example: if the shortest path from u to w goes through v, then
  • the part of that path from u to v is the shortest path from u to v.
  • if there were some better path u..v, that would also be part of a better way to get from u to w.
Dijkstra’s Shortest Paths: Subpaths

- Fact: subpaths of shortest paths are shortest paths

- Consequence: a candidate shortest path from start node $s$ to some node $v$’s neighbor $w$ is the shortest path from to $v$ + the edge weight from $v$ to $w$.

shortest path $u...v = v.d$ $w$t(v,w)
Dijkstra’s Shortest Paths: Intuition

• Intuition: explore nodes kinda like BFS.

• There are three kinds of nodes:
  • Settled - nodes for which we know the actual shortest path.
  • Frontier - nodes that have been visited but we don’t necessarily have their actual shortest path.
  • Unexplored - all other nodes.

• Each node n keeps track of \( n.d \), the length of the shortest known known path from start.

• We may discover a shorter path to a frontier node than the one we’ve found already - if so, update \( n.d \).
Before:

During:

After:
Dijkstra’s Shortest Paths: High-Level Algorithm

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:
    move the node f with smallest d from F to S
    For each neighbor w of f:
        if we’ve never seen w before:
            set its path length
            add it to frontier
        else if the path to w via f is shorter:
            update w’s shortest path length
Dijkstra’s Shortest Paths: High-Level Algorithm

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:
  move the node f with smallest d from F to S
  For each neighbor w of f:
    if we’ve never seen w before:
      set its path length
      add it to frontier
    else if the path to w via f is shorter:
      update w’s shortest path length
Dijkstra’s Shortest Paths: High-Level Algorithm

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:
    move the node \( f \) with smallest \( d \) from \( F \) to \( S \)
    For each neighbor \( w \) of \( f \):
        if we’ve never seen \( w \) before:
            set its path length
            add it to frontier
        else if the path to \( w \) via \( f \) is shorter:
            update \( w \)’s shortest path length

\[
\begin{align*}
\text{settled} & \quad \ldots \quad u \quad \ldots \quad f \quad \ldots \quad w \\
0.6 \quad 0.6 & \quad 0.6 \quad 0.6 & \quad 0.6 \quad 0.6 & \quad 0.6 \quad 0.6 & \quad 0.6 \quad 0.6 \quad 0.6 & \quad 0.6 \\
\end{align*}
\]

\[
\begin{align*}
w.d &= u.d + \text{wt}(u,w) \\
f.d \\
\end{align*}
\]
Dijkstra’s Shortest Paths: High-Level Algorithm

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:
    move the node \( f \) with smallest \( d \) from \( F \) to \( S \)
    For each neighbor \( w \) of \( f \):
        if we’ve never seen \( w \) before:
            set its path length
            add it to frontier
        else if the path to \( w \) via \( f \) is shorter:
            update \( w \)’s shortest path length

settled

\[ w.d = u.d + \text{wt}(u,w) \]

\[ f.d + \text{wt}(f,w) \]
Dijkstra’s Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>?</td>
</tr>
</tbody>
</table>

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:

  move the node f with smallest d from F to S
  For each neighbor w of f:
    if we’ve never seen w before:
      set its path length to f.d + wt(f,w)
      add w to the frontier
    else if the path to w via f is shorter:
      update w’s shortest path length

Settled set:

Frontier set:

shortest-paths(4)
Dijkstra’s Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Initialize Settled to empty
Initialize Frontier to the start node

While the frontier isn’t empty:
  move the node f with smallest d from F to S
  For each neighbor w of f:
    if we’ve never seen w before:
      set its path length to f.d + wt(f,w)
      add w to the frontier
    else if the path to w via f is shorter:
      update w’s shortest path length

Settled set: {}

Frontier set: {4}
Dijkstra’s Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Settled set: {4}

Frontier set: {}
Dijkstra’s Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Initialize Settled to empty
Initialize Frontier to the start node

While the frontier isn’t empty:

- move the node \( f \) with smallest \( d \) from \( F \) to \( S \)

For each neighbor \( w \) of \( f \):

- if we’ve never seen \( w \) before:
  - set its path length to \( f.d + wt(f,w) \)
  - add \( w \) to the frontier

- else if the path to \( w \) via \( f \) is shorter:
  - update \( w \)’s shortest path length

Settled set: \{4\}

Frontier set: \{0\}

shortest-paths(4)
Dijkstra’s Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:

- move the node f with smallest d from F to S
- For each neighbor w of f:
  - if we’ve never seen w before:
    - set its path length to f.d + wt(f,w)
    - add w to the frontier
  - else if the path to w via f is shorter:
    - update w’s shortest path length

Settled set: {4, 0}
Frontier set: {}

shortest-paths(4)
Dijkstra’s Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:
  move the node f with smallest d from F to S
  For each neighbor w of f:
    if we’ve never seen w before:
      set its path length to f.d + wt(f,w)
      add w to the frontier
    else if the path to w via f is shorter:
      update w’s shortest path length

Settled set: \{4, 0\}
Frontier set: \{1\}
Dijkstra’s Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Settled set: \{4, 0\}

Frontier set: \{1, 2\}

Initialize Settled to empty

Initialize Frontier to the start node

While the frontier isn’t empty:

- move the node \( f \) with smallest \( d \) from \( F \) to \( S \)

For each neighbor \( w \) of \( f \):

- if we’ve never seen \( w \) before:
  - set its path length to \( f.d + wt(f,w) \)
  - add \( w \) to the frontier

- else if the path to \( w \) via \( f \) is shorter:
  - update \( w \)’s shortest path length

shortest-paths(4)
Dijkstra's Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:

- move the node f with smallest d from F to S
- For each neighbor w of f:
  - if we’ve never seen w before:
    - set its path length to f.d + wt(f,w)
    - add w to the frontier
  - else if the path to w via f is shorter:
    - update w’s shortest path length

Settled set: {4, 0, 1}
Frontier set: {2}

shortest-paths(4)
Dijkstra’s Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Settled set: {4, 0, 1}
Frontier set: {2, 3}

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:
  move the node f with smallest d from F to S
  For each neighbor w of f:
    if we’ve never seen w before:
      set its path length to f.d + wt(f,w)
      add w to the frontier
    else if the path to w via f is shorter:
      update w’s shortest path length

shortest-paths(4)
Dijkstra’s Shortest Path Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:

move the node f with smallest d from F to S
For each neighbor w of f:
if we’ve never seen w before:
  set its path length to f.d + wt(f,w)
  add w to the frontier
else if the path to w via f is shorter:
  update w’s shortest path length

Settled set: {4, 0, 1, 2}
Frontier set: {3}
Dijkstra's Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Settled set: \{4, 0, 1, 2\}

Frontier set: \{3\}

Initialize Settled to empty
Initialize Frontier to the start node

While the frontier isn't empty:
  move the node $f$ with smallest $d$ from $F$ to $S$
  For each neighbor $w$ of $f$:
    if we've never seen $w$ before:
      set its path length to $f.d + wt(f,w)$
      add $w$ to the frontier
    else if the path to $w$ via $f$ is shorter:
      update $w$'s shortest path length

$2.d + wt(2,3) < 3.d$

$7 < 8$

shortest-paths(4)
Dijkstra's Shortest Paths: Execution

Best known distances:

<table>
<thead>
<tr>
<th>Node</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Initialize Settled to empty
Initialize Frontier to the start node
While the frontier isn’t empty:

- move the node f with smallest d from F to S
- For each neighbor w of f:
  - if we’ve never seen w before:
    - set its path length to f.d + wt(f,w)
    - add w to the frontier
  - else if the path to w via f is shorter:
    - update w’s shortest path length

Settled set: \{4, 0, 1, 2, 3\}

Frontier set: {}  Empty => done!
Let’s Dijkstra

• Half get the algorithm, half get the graphs.

• Run the algorithm on each graph:
  • (first) 5-node graph: start at node S
  • (second) Other graph: start at node D
Unanswereded Questions

• Does this always work?

• How do you get the path, not just its length?

• How do you implement it efficiently?

• What’s the runtime?
Sometimes it’s not about finding the shortest path.

Have a great Thanksgiving!