CSE 326: Data Structures

Dijkstra’s Algorithm

James Fogarty

Autumn 2009
Dijkstra’s Algorithm: Idea

Adapt BFS to handle weighted graphs

Two kinds of vertices:
- Finished or known vertices
  - Shortest distance has been computed
- Unknown vertices
  - Have tentative distance
Dijkstra’s Algorithm: Idea

At each step:
1) Pick closest unknown vertex
2) Add it to known vertices
3) Update distances
Dijkstra’s Algorithm: Pseudocode

Initialize the cost of each node to $\infty$

Initialize the cost of the source to 0

While there are unknown nodes left in the graph
   Select an unknown node $b$ with the lowest cost
   Mark $b$ as known
   For each node $a$ adjacent to $b$
      $a$’s cost = \( \min(a$’s old cost, \( b$’s cost + cost of \((b, a)\))
      $a$’s prev path node = $b$
Important Features

• Once a vertex is made known, the cost of the shortest path to that node is known

• While a vertex is still not known, another shorter path to it might still be found

• The shortest path itself can be found by following the backward pointers stored in node.path
Dijkstra’s Algorithm in action

Graph with weights and vertex status:

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Visited?</th>
<th>Cost</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>??</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm in action

Vertex | Visited? | Cost | Found by
--- | --- | --- | ---
A | Y | 0 | A
B | <=2 | A
C | <=1 | A
D | <=4 | A
E | ?? | |
F | ?? | |
G | ?? | |
H | ?? | |
Dijkstra’s Algorithm in action

Vertex | Visited? | Cost | Found by
--- | --- | --- | ---
A | Y | 0 | 
B | <=2 | A | 
C | Y | 1 | A | 
D | <=4 | A | 
E | <=12 | C | 
F | ?? | 
G | ?? | 
H | ?? |
Dijkstra’s Algorithm in action

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Visited?</th>
<th>Cost</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>&lt;=4</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>&lt;=12</td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>&lt;=4</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>??</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm in action

Vertex | Visited? | Cost | Found by
--- | --- | --- | ---
A | Y | 0 | 
B | Y | 2 | A
C | Y | 1 | A
D | Y | 4 | A
E | <=12 | C
F | <=4 | B
G | ?? | 
H | ?? | 

Graph with vertices A, B, C, D, E, F, G, H and edges with weights.
Dijkstra’s Algorithm in action

Vertex Visited? Cost Found by
A Y 0 A
B Y 2 A
C Y 1 A
D Y 4 A
E <=12 C
F Y 4 B
G ??
H <=7 F
Dijkstra’s Algorithm in action

Vertex | Visited? | Cost | Found by
-------|---------|------|---------
A       | Y       | 0    |         |
B       | Y       | 2    | A       |
C       | Y       | 1    | A       |
D       | Y       | 4    | A       |
E       |         | <=12 | C       |
F       | Y       | 4    | B       |
G       |         | <=8  | H       |
H       | Y       | 7    | F       |
Dijkstra’s Algorithm in action

Vertex | Visited? | Cost | Found by
---|---|---|---
A | Y | 0 | A
B | Y | 2 | A
C | Y | 1 | A
D | Y | 4 | A
E | <=11 | G
F | Y | 4 | B
G | Y | 8 | H
H | Y | 7 | F
Dijkstra’s Algorithm in action

**Graph Representation:**

- Vertices: A, B, C, D, E, F, G, H
- Edges with weights:
  - A to B: 2
  - B to C: 2
  - C to D: 2
  - C to E: 11
  - D to A: 4
  - D to C: 9
  - E to D: 7
  - E to G: 11
  - F to E: 3
  - F to G: 3
  - G to F: 2
  - G to H: 1
  - H to F: 7
  - H to G: 8

**Table:**

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Visited?</th>
<th>Cost</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>Y</td>
<td>11</td>
<td>G</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td>Y</td>
<td>8</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>Y</td>
<td>7</td>
<td>F</td>
</tr>
</tbody>
</table>
Another

\[
\begin{array}{|c|c|c|c|}
\hline
V & Visited? & Cost & Found by \\
\hline
v0 & & & \\
v1 & & & \\
v2 & & & \\
v3 & & & \\
v4 & & & \\
v5 & & & \\
v6 & & & \\
\hline
\end{array}
\]

\[
\begin{align*}
\text{V Visited? Cost Found by} \\
v_0 & \quad \text{ } & \quad \text{ } & \quad \text{ } \\
v_1 & \quad \text{ } & \quad \text{ } & \quad \text{ } \\
v_2 & \quad \text{ } & \quad \text{ } & \quad \text{ } \\
v_3 & \quad \text{ } & \quad \text{ } & \quad \text{ } \\
v_4 & \quad \text{ } & \quad \text{ } & \quad \text{ } \\
v_5 & \quad \text{ } & \quad \text{ } & \quad \text{ } \\
v_6 & \quad \text{ } & \quad \text{ } & \quad \text{ } \\
\end{align*}
\]
V | Visited? | Cost | Found by
---|---|---|---
v0 | Y | 0 |
v1 |
v2 | <= 2 | V0 |
v3 | <= 1 | V0 |
v4 |
v5 |
v6 |
Another

<table>
<thead>
<tr>
<th>V</th>
<th>Visited?</th>
<th>Cost</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>&lt;= 6</td>
<td>V3</td>
<td></td>
</tr>
<tr>
<td>v2</td>
<td>Y</td>
<td>2</td>
<td>V0</td>
</tr>
<tr>
<td>v3</td>
<td>Y</td>
<td>1</td>
<td>V0</td>
</tr>
<tr>
<td>v4</td>
<td>&lt;= 2</td>
<td>V3</td>
<td></td>
</tr>
<tr>
<td>v5</td>
<td>&lt;= 4</td>
<td>V2</td>
<td></td>
</tr>
<tr>
<td>v6</td>
<td>&lt;= 6</td>
<td>V3</td>
<td></td>
</tr>
</tbody>
</table>
Another

<table>
<thead>
<tr>
<th>V</th>
<th>Visited?</th>
<th>Cost</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td></td>
<td>&lt;= 3</td>
<td>V4</td>
</tr>
<tr>
<td>v2</td>
<td>Y</td>
<td>2</td>
<td>V0</td>
</tr>
<tr>
<td>v3</td>
<td>Y</td>
<td>1</td>
<td>V0</td>
</tr>
<tr>
<td>v4</td>
<td>Y</td>
<td>2</td>
<td>V3</td>
</tr>
<tr>
<td>v5</td>
<td></td>
<td>&lt;= 4</td>
<td>V2</td>
</tr>
<tr>
<td>v6</td>
<td></td>
<td>&lt;= 6</td>
<td>V3</td>
</tr>
</tbody>
</table>
Another

<table>
<thead>
<tr>
<th>V</th>
<th>Visited?</th>
<th>Cost</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>Y</td>
<td>3</td>
<td>V4</td>
</tr>
<tr>
<td>v2</td>
<td>Y</td>
<td>2</td>
<td>V0</td>
</tr>
<tr>
<td>v3</td>
<td>Y</td>
<td>1</td>
<td>V0</td>
</tr>
<tr>
<td>v4</td>
<td>Y</td>
<td>2</td>
<td>V3</td>
</tr>
<tr>
<td>v5</td>
<td>&lt;= 4</td>
<td></td>
<td>V2</td>
</tr>
<tr>
<td>v6</td>
<td>&lt;= 6</td>
<td></td>
<td>V3</td>
</tr>
</tbody>
</table>
Another

<table>
<thead>
<tr>
<th>V</th>
<th>Visited?</th>
<th>Cost</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>Y</td>
<td>3</td>
<td>V4</td>
</tr>
<tr>
<td>v2</td>
<td>Y</td>
<td>2</td>
<td>V0</td>
</tr>
<tr>
<td>v3</td>
<td>Y</td>
<td>1</td>
<td>V0</td>
</tr>
<tr>
<td>v4</td>
<td>Y</td>
<td>2</td>
<td>V3</td>
</tr>
<tr>
<td>v5</td>
<td>Y</td>
<td>4</td>
<td>V2</td>
</tr>
<tr>
<td>v6</td>
<td>&lt;= 6</td>
<td></td>
<td>V3</td>
</tr>
</tbody>
</table>
Another

<table>
<thead>
<tr>
<th>V</th>
<th>Visited?</th>
<th>Cost</th>
<th>Found by</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>v1</td>
<td>Y</td>
<td>3</td>
<td>V4</td>
</tr>
<tr>
<td>v2</td>
<td>Y</td>
<td>2</td>
<td>V0</td>
</tr>
<tr>
<td>v3</td>
<td>Y</td>
<td>1</td>
<td>V0</td>
</tr>
<tr>
<td>v4</td>
<td>Y</td>
<td>2</td>
<td>V3</td>
</tr>
<tr>
<td>v5</td>
<td>Y</td>
<td>4</td>
<td>V2</td>
</tr>
<tr>
<td>v6</td>
<td>Y</td>
<td>6</td>
<td>V3</td>
</tr>
</tbody>
</table>
Dijkstra’s Alg: Implementation

Initialize the cost of each node to $\infty$
Initialize the cost of the source to 0
While there are unknown nodes left in the graph
  Select the unknown node $b$ with the lowest cost
  Mark $b$ as known
  For each node $a$ adjacent to $b$
    $a$’s cost = min($a$’s old cost, $b$’s cost + cost of $(b, a)$)
    $a$’s prev path node = $b$ (if we updated $a$’s cost)

What data structures should we use?

Running time?
void Graph::dijkstra(Vertex s) {
    Vertex v, w;

    Initialize s.dist = 0 and set dist of all other vertices to infinity

    while (there exist unknown vertices, find the one b with the smallest distance)
        b.known = true;

        for each a adjacent to b
            if (!a.known)
                if (b.dist + weight(b, a) < a.dist) {
                    a.dist = (b.dist + weight(b, a));
                    a.path = b;
                }
    }
}

Running time: O(|E| log |V|) – there are |E| edges to examine, and each one causes a heap operation of time O(log |V|)
Dijkstra’s Algorithm: Summary

• Classic algorithm for solving SSSP in weighted graphs without negative weights

• A greedy algorithm (irrevocably makes decisions without considering future consequences)

• Intuition for correctness:
  – shortest path from source vertex to itself is 0
  – cost of going to adjacent nodes is at most edge weights
  – cheapest of these must be shortest path to that node
  – update paths for new node and continue picking cheapest path
How does Dijkstra’s decide which vertex to add to the Known set next?

- If path to $V$ is shortest, path to $W$ must be \textit{at least as long}
  \textit{(or else we would have picked $W$ as the next vertex)}
- So the path through $W$ to $V$ cannot be any shorter!
Correctness: Inside the Cloud

Prove by induction on # of nodes in the cloud:

Initial cloud is just the source with shortest path 0

Assume: Everything inside the cloud has the correct shortest path

Inductive step: Only when we prove the shortest path to some node \( v \) (which is \textit{not} in the cloud) is correct, we add it to the cloud

When does Dijkstra’s algorithm not work?
The Trouble with Negative Weight Cycles

What’s the shortest path from A to E?

Problem?
Dijkstra’s vs BFS

At each step:
1) Pick closest unknown vertex
2) Add it to finished vertices
3) Update distances

Dijkstra’s Algorithm

At each step:
1) Pick vertex from queue
2) Add it to visited vertices
3) Update queue with neighbors

Breadth-first Search
Two Questions

• What if I had multiple potential start points, and need to know the minimum cost of reaching each node from any start point?

• What if I want to know the minimum cost between every pair of nodes in the graph?