CSE 101- Winter ‘16
Discussion Section
January 25th 2016
Shortest Paths

• Breadth-first search finds the shortest path in graphs whose edges have unit length.

• How do we adapt breadth-first search to find the shortest path in a more general graph $G = (V,E)$ whose edge lengths $l_e$ are positive integers.

• We could break $G$’s long edges into unit-length pieces by introducing “dummy nodes”. Breadth-first search on this new graph should give the shortest path between the nodes in the original graph.

• For a more detailed explanation of the above procedure, please refer to slides 10-11 of lecture 6.
Dijkstra’s Algorithm

• The usage of the concept of alarm clock in the breadth-first search performed on the graph with dummy nodes results in Dijsktra’s algorithm.

procedure dijkstra(G, l, s)
for all u ∈ V:
    \( \text{dist}(u) = \infty \)
    \( \text{prev}(u) = \text{nil} \)
dist(s) = 0

H = makequeue(V) (using dist-values as keys)
While H is not empty:
    u = deletemin(H)
    for all edges (u, v) ∈ E:
        if(dist(v)) > dist(u) + l(u,v) :
            dist(v) = dist(u) + l(u,v)
            prev(v) = u
            decreasekey(H,v)
Dijkstra’s Algorithm

• In the graph below, we are finding the shortest paths to all the nodes from the source node A.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Distance from node A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
</tr>
<tr>
<td>E</td>
<td>∞</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Initially only node A is part of the known region.

Nodes not in the known region are maintained in a priority queue with their current distance from node A as the key.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Distance from node A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
</tr>
<tr>
<td>E</td>
<td>∞</td>
</tr>
</tbody>
</table>

Update the distances of the neighbors of A.
Dijkstra’s Algorithm

Delete-min – Returns C as the node with the shortest distance from the known region. C is added to the known region and deleted from the priority queue.

### Nodes

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Distance from node A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
</tr>
</tbody>
</table>

Update the distances of the neighbors of C
Dijkstra’s Algorithm

Nodes | Distance from node A
--- | ---
A | 0
B | 3
C | 2
D | 5
E | 6

Update the distances of the neighbors of B

Delete-min – Returns B as the node with the shortest distance from the known region. B is added to the known region and deleted from the priority queue.
Dijkstra’s Algorithm

Since D has no outgoing edges, we have nothing to update.

Nodes | Distance from node A
--- | ---
A | 0
B | 3
C | 2
D | 5
E | 6

Delete-min – Returns D as the node with the shortest distance from the known region. D is added to the known region and deleted from the priority queue.
Dijkstra’s Algorithm

We have the shortest path from A to all the other nodes

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Distance from node A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm
Dijkstra’s Algorithm: Running time

procedure dijkstra(G, l, s)
for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil
dist(s) = 0

H = makequeue(V) (using dist-values as keys)
While H is not empty:
    u = deletemin(H)
    for all edges (u, v) ∈ E:
        if(dist(v)) > dist(u) + l(u,v) :
            dist(v) = dist(u) + l(u,v)
            prev(v) = u
            decreasekey(H,v)
Dijkstra’s Algorithm: Running time

procedure dijkstra(G, l, s)
for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil
dist(s) = 0

H = makequeue(V) (using dist-values as keys) |V|
While H is not empty:
    u = deletemin(H) |V|
    for all edges (u, v) ∈ E:
        if(dist(v)) > dist(u) + l(u,v) :
            dist(v) = dist(u) + l(u,v)
            prev(v) = u
            decreasekey(H,v) |E|

Priority queue operations = |V|+|E| insert/decreasekey + |V| deletemin
Priority queue implementation

• Running time = (|V| + |E|) \* insert/decreasekey + |V| * deletemin

• Binary heap
  – Insert/decreaseKey: O(log |V|)
  – deletemin: O(log |V|)
  
  – Running time = O((|V| + |E|) \log |V|)

• Array
  – Insert/decreasekey: O(1)
  – deletemin: O(|V|)
  
  – Running time = O(|V|^2)
Shortest Path (negative edges)
Shortest Path (negative edges)

Dijkstra’s algorithm fails

Nodes | Distance from node S
--- | ---
S | 0
A | 3
B | 4
Shortest Path (negative edges)

- Dijkstra’s algorithm works in part because the shortest path from the starting node S to any node V must pass exclusively through nodes that are closer than V.
- This no longer holds when the edge lengths can be negative.
- In the figure below, shortest path from S to A passes through B, a node that is further away.
Bellman-Ford Algorithm

• Idea: Successive Approximation
  – Find SP using ≤ 1 edges
  – Find SP using ≤ 2 edges
  – Find SP using ≤ |V|-1 edges → shortest paths (why?)
Bellman-Ford Algorithm

• Idea: Successive Approximation
  – Find SP using ≤ 1 edges
  – Find SP using ≤ 2 edges
  – Find SP using ≤ |V|-1 edges → shortest paths (why?)

• Consider the following update operation performed in Dijkstra’s:

  \[
  \text{procedure } \text{update}((u,v) \in E) \\
  \text{Dist}_{k+1}(v) = \min\{\text{dist}_k(v), \text{dist}_k(u) + l(u,v)\}
  \]
Bellman-Ford Algorithm

• Idea: Successive Approximation
  – Find SP using ≤ 1 edges
  – Find SP using ≤ 2 edges
  – Find SP using ≤ |V|-1 edges → shortest paths (why?)

• Consider the following update operation performed in Dijkstra’s:

  procedure update((u,v) ∈ E)
  Dist_{k+1}(v) = min{dist_k(v), dist_k(u)+l(u,v)}

• “Update” all the edges |V| - 1 times. This should ensure we get the shortest path even when the graph has negative edges.

• Why do we need to do this |V| -1 times?
  – The shortest path from the source vertex S to a vertex V has at most V-1 edges.
procedure update((u, v) ∈ E)
\text{dist}(v) = \min\{\text{dist}(v), \text{dist}(u) + l(u,v)\}

procedure shortest-paths(G, l, s)

for all u ∈ V:
\text{dist}(u) = \infty
\text{prev}(u) = \text{nil}

\text{dist}(s) = 0
Repeat |V| − 1 times:
\text{for all } e ∈ E:
update(e)  \quad (Ref: Algorithms, Fig 4.8)
Previous example

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Distance from node S</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
</tbody>
</table>

Dijkstra

Bellman-Ford
Previous example

Dijkstra

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Distance from node S</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
</tbody>
</table>

Bellman-Ford

<table>
<thead>
<tr>
<th>Node</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
</tr>
</tbody>
</table>
Previous example

Dijkstra

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Distance from node S</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
</tbody>
</table>

Bellman-Ford

<table>
<thead>
<tr>
<th>Node</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>∞</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Previous example

Dijkstra

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Distance from node S</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
</tr>
</tbody>
</table>

Bellman-Ford

<table>
<thead>
<tr>
<th>Node</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
</tr>
</tbody>
</table>