HW7, Dijkstra’s

CSE 331 – Section 7
02/21/2013
Slides by Kellen Donohue,
Modified by David Mailhot,
with much material from Dan Grossman
Homework 7

Modify your graph to use Generics
  • Change your hw5 code where it is now
  • Will have to update hw5, hw6 tests

Implement Dijkstra’s algorithm
  • Alternate search algorithm that uses edge weights
  • Apply to Marvel graph, with edge weights reciprocal to number of books in common
Note on folders

MarvelPaths2.java looks in src/hw7/data
HW7TestDriver.java looks in src/hw7/test
Shortest paths

Done: BFS to find the minimum path length from $v$ to $u$

Now: Weighted graphs

Given a weighted graph and node $v$, find the minimum-cost path from $v$ to every node

Unlike before, BFS will not work
Not as easy

Why BFS won’t work:
Smallest-cost path may not have the fewest edges

We will assume there are no negative weights
- Problem is ill-defined if there are negative-cost cycles
- Today’s algorithm is wrong if edges can be negative
Dijkstra’s Algorithm

Named after its inventor Edsger Dijkstra (1930-2002)

- Truly one of the “founders” of computer science; this is just one of his many contributions

The idea: reminiscent of BFS, but adapted to handle weights

- Grow the set of nodes whose shortest distance has been computed
- Nodes not in the set will have a “best distance so far”
- A priority queue will turn out to be useful for efficiency
Dijkstra’s Algorithm: Idea

Initially, start node has cost 0 and all other nodes have cost $\infty$

At each step:

- Pick closest unknown vertex $v$
- Add it to the “cloud” of known vertices
- Update distances for nodes with edges from $v$

That’s it!
Aside: weights for Marvel Data

The Marvel data doesn't really have a measure of 'weight' we can use:

- So for HW7 you'll be hacking your own!
Aside: weights for Marvel Data

The idea: the more well-connected two characters are, the lower the weight and the more likely that a path is taken through them.

- The weight of the edge between two characters is equal to the inverse of how many comic books those two characters are in together (the 'multiplicative inverse').
- For example, if Amazing Amoeba and Zany Zebra appeared in 5 comic books together, the weight of the edge between them would be 1/5.
- No duplicate edges: two characters will have at most one edge between them that is labeled with a cost.
Aside: weights for Marvel Data

You'll be placing your new Marvel application in hw7/MarvelPaths2.java.

Key: You will calculate edge costs when you read in the data and construct your graph using those calculated weights, all in MarvelPaths2.java.
Dijkstra’s Algorithm: Idea

Initially, start node has cost 0 and all other nodes have cost $\infty$.

At each step:

- Pick closest unknown vertex $v$
- Add it to the “cloud” of known vertices
- Update distances for nodes with edges from $v$

That’s it!
The Algorithm

1. For each node $v$, set $v.cost = \infty$ and $v.known = false$
2. Set $source.cost = 0$
3. While there are unknown nodes in the graph
   a. Select the unknown node $v$ with lowest cost
   b. Mark $v$ as known
   c. For each edge $(v,u)$ with weight $w$,
      \[
      c1 = v.cost + w \quad // \text{cost of best path through } v \text{ to } u
      \]
      \[
      c2 = u.cost \quad // \text{cost of best path to } u \text{ previously known}
      \]
      if($c1 < c2$) { // if the path through $v$ is better
        $u.cost = c1$
        $u.path = v$ // for computing actual paths
      }
Important features

When a vertex is marked known, the cost of the shortest path to that node is known

- The path is also known by following back-pointers

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

e: The “Order Added to Known Set” is not important

- A detail about how the algorithm works (client doesn’t care)
- Not used by the algorithm (implementation doesn’t care)
- It is sorted by path-cost, resolving ties in some way
Example #1

![Graph Diagram]

**Order Added to Known Set:**

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>0</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>
Example #1

Order Added to Known Set:

A

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>≤ 2</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>≤ 1</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>≤ 4</td>
<td>A</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>
Example #1

Order Added to Known Set:
A, C
**Example #1**

**Order Added to Known Set:**

A, C, B

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</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>≤ 4</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>≤ 12</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>≤ 4</td>
<td>B</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>
Example #1

Order Added to Known Set:
A, C, B, D
**Example #1**

```
<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</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></td>
<td>≤ 12</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>≤ 7</td>
<td>F</td>
</tr>
</tbody>
</table>
```

**Order Added to Known Set:**

A, C, B, D, F
Example #1

Order Added to Known Set:
A, C, B, D, F, H

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</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></td>
<td>≤ 12</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>≤ 8</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>Y</td>
<td>7</td>
<td>F</td>
</tr>
</tbody>
</table>
Example #1

Order Added to Known Set:

A, C, B, D, F, H, G

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</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></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>
Example #1

Order Added to Known Set:
A, C, B, D, F, H, G, E

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</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>
Features

When a vertex is marked known, the cost of the shortest path to that node is known
  • The path is also known by following back-pointers
While a vertex is still not known, another shorter path to it might still be found

Note: The “Order Added to Known Set” is not important
  • A detail about how the algorithm works (client doesn’t care)
  • Not used by the algorithm (implementation doesn’t care)
  • It is sorted by path-cost, resolving ties in some way
Interpreting the Results

Now that we’re done, how do we get the path from, say, A to E?

Order Added to Known Set:
A, C, B, D, F, H, G, E
Order Added to Known Set:
Example #2

Order Added to Known Set:

A, D, C, E, B, F, G

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</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>3</td>
<td>E</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>Y</td>
<td>2</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>4</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td>Y</td>
<td>6</td>
<td>D</td>
</tr>
</tbody>
</table>
Efficiency, first approach

Use pseudocode to determine asymptotic run-time

Notice each edge is processed only once

dijkstra(Graph G, Node start) {
    for each node: x.cost=inf, x.known=false
    start.cost = 0
    while (not all nodes are known) {
        b = dequeue
        b.known = true
        for each edge (b,a) in G
            if (!a.known)
                if (b.cost + weight((b,a)) < a.cost){
                    a.cost = b.cost + weight((b,a))
                    a.path = b
                }
    }
}
Priority Queue

- Increase efficiency by considering lowest cost unknown vertex with sorting instead of looking at all vertices
- PriorityQueue is like a queue, but returns elements by **lowest value** instead of insertion time
Priority Queue

Two different ways to define 'lowest value' for a priority queue:

1. Inserted elements must implement the java Comparable interface.
   a. class Node implements Comparable<Node>
   b. public int compareTo(other)

2. Define a Comparator object and hand it to your priority queue on construction.
   a. class NodeComparator extends Comparator<Node>
   b. new PriorityQueue(new NodeComparator())
Efficiency, second approach

Use pseudo code to determine asymptotic run-time

dijkstra(Graph G, Node start) {
   for each node: x.cost=infinity, x.known=false
   start.cost = 0
   build-heap with all nodes
   while(heap is not empty) {
      b = deleteMin()
      if (b.known) continue;
      b.known = true
      for each edge (b,a) in G
         if(!a.known) {
            add(b.cost + weight((b,a)) )
         }
   }
}
Correctness: Intuition

Rough intuition:
All the “known” vertices have the correct shortest path
  • True initially: shortest path to start node has cost 0
  • If it stays true every time we mark a node “known”, then by induction this holds and eventually everything is “known”

Key fact we need: When we mark a vertex “known” we won’t discover a shorter path later!
  • This holds only because Dijkstra’s algorithm picks the node with the next shortest path-so-far
  • The proof is by contradiction…
Correctness: The Cloud (Rough Sketch)

Suppose v is the next node to be marked known (“added to the cloud”)

- The best-known path to v must have only nodes “in the cloud”
  - Else we would have picked a node closer to the cloud than v
- Suppose the actual shortest path to v is different
  - It won’t use only cloud nodes, or we would know about it
  - So it must use non-cloud nodes. Let w be the first non-cloud node on this path. The part of the path up to w is already known and must be shorter than the best-known path to v. So v would not have been picked. Contradiction.
Use in HW

- Will use in HW7 to find paths between characters, weighted so characters that commonly appear together have short paths between them

- Will use in HW8/9 to map distances across campus