E.C.: Graphing: Shortest Path
(45 pts, due SUNDAY, MAY 24)

Contents

Dijkstra’s algorithm .............................................................................................................................................. 1

Single-source shortest path: ............................................................................................................................ 1

Other Shortest Path Applications:.................................................................................................................. 2

Assumptions: ..................................................................................................................................................... 2

Directed: ........................................................................................................................................................... 2

Undirected: ........................................................................................................................................................ 3

Dijkstra’s algorithm: ......................................................................................................................................... 3

Greedy Approach ............................................................................................................................................ 3

Explanation with Examples: ............................................................................................................................ 4

Finding Minimum Path: .................................................................................................................................. 8

PROBLEMS: .................................................................................................................................................... 9

1. (4 pts) .......................................................................................................................................................... 9

(1 pt) ............................................................................................................................................................. 9

2. (4 pts) ........................................................................................................................................................ 9

(1 pt) ............................................................................................................................................................. 9

3. (2 pts) ........................................................................................................................................................ 9

4. (3 pts) ....................................................................................................................................................... 9

(2 pts) ........................................................................................................................................................... 9

5. (3 pts) ....................................................................................................................................................... 10

6. (8 pts) ....................................................................................................................................................... 10

7. (2 pts) ....................................................................................................................................................... 10

8. (12 pts) .................................................................................................................................................... 11

(2 pts) ........................................................................................................................................................... 11

Dijkstra’s algorithm

Single-source shortest path:

■ Finding the shortest path from one particular vertex to all other vertices on a graph

- Think of taking a plane from Philadelphia International Airport to any other airport – what is the shortest path?

“If debugging is the process of removing software bugs, then programming must be the process of putting them in.”

- Edsger Dijkstra
Dijkstra’s Algorithm

Other Shortest Path Applications:

- **Maps:** finding the shortest route, finding the fastest route
  - Vertices: intersections,
  - Edges: roads (cost: distance, speed, or something else)
- **Networks:** for routing packets (data) across a network or the internet
  - Vertices: routers
  - Edges: connections (cost: time for travel)
- **Epidemiology:** modeling the spread of infectious diseases (appropriate, anyone???)
  - Vertices: individuals with disease
  - Edges: contacts (cost: statistical likelihood of infection?)

"Computer Science is no more about computers than astronomy is about telescopes”

*Edsger Dijkstra*

Assumptions:

- **Costs of edges** are positive numbers or zero
- **Not all vertices** may be reachable from the single source
- **Weights are not necessarily distance** – could be time, cost, likelihood, etc.
- **Shortest paths may not be unique** – there may be more than one shortest path

"Progress is possible only if we train ourselves to think about programs without thinking of them as pieces of executable code."

*Edsger Dijkstra*

**Graphs** can be **Directed** or **undirected**:

**Directed:** the cost from a to b can be different than the cost of going from b to a

Think of it this way: the cost from a flight from Philly to New Haven is different than the cost of a flight from New Haven to Philly
Dijkstra’s Algorithm

**Undirected:** means the cost from vertex a to vertex b and vice versa is the same.

Think of it this way: if the distance is the cost, then the distance from Philly to New Haven is the same either way.

![Graph Diagram]

Dijkstra’s algorithm:

**Dijkstra:**
- Edsger Dijkstra – Dutch computer scientist
- Responsible for the concept of structured programming, which makes dealing with complex data management possible.

> “Simplicity and elegance are unpopular because they require hard work and discipline to achieve and education to be appreciated.”

>-Edsger Dijkstra

Dijkstra’s shortest path algorithm:

- solves single-source shortest path problem
- **Works on** both directed and undirected graphs.
  - all edges must have nonnegative weights.

**Greedy Approach**

- **Greedy means make a locally optimal choice in the hope that the ultimate result will be optimal**
  - *Always add the next shortest path*

**Input:** Weighted graph
- (weighted graph means edges have a weight assigned to them), E.g.,
  - distance between 2 places
Dijkstra’s Algorithm

- Cost of flight
- Length of time taken to travel between the 2 edges
- Etc.
  - $G = (E, V)$
    - Graph is a set of edges and vertices
  - source vertex $v \in V$,
    - this is the starting vertex in the graph
    - and the starting vertex has to be in the set of all the vertices
  - all edge weights are nonnegative
- Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $v \in V$ to all other vertices

“Program testing can be used to show the presence of bugs, but never to show their absence”

-Edsger Dijkstra

Explanation with Examples:
- We’ve got a graph with edges and distances between connected edges
  - (may be represented as a 2-dimensional array, aka Matrix):

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>inf</td>
<td>inf</td>
</tr>
<tr>
<td>$A$</td>
<td>inf</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$B$</td>
<td>inf</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>inf</td>
</tr>
<tr>
<td>$C$</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>$D$</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

- Start by initiating all shortest-path distances to infinity (or a really big number) except the distance from our starting vertex (0)
- Set visited to False for all vertices
- The Predecessor array gets set to NULL initially

<table>
<thead>
<tr>
<th>V</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D[v]$</td>
<td>0</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
</tr>
<tr>
<td>$Pred[v]$</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>Visited</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

- All vertexes go into a priority queue based on their distance from the initial vertex.

**Priority Queue:**

<table>
<thead>
<tr>
<th>V</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D[v]</td>
<td>0</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
</tr>
</tbody>
</table>

- Loop:
  - pick the unvisited vertex with the shortest distance
    - In the above priority queue, it would be S (yeah, I know it’s the starting vertex, but that’s how we start)
  - Add S to the set of visited nodes. (Set it to True in the visited array)
  - Remove it from the priority queue.
  - Recalculate all distances from the original vertex to all other vertices not visited yet by taking the minimum of the distance from the original vertex to vx or the distance from ther original vertex to the current vertex + the distance from the current vertex to vx

<table>
<thead>
<tr>
<th>V</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D[v]</td>
<td>0</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
</tr>
<tr>
<td>Pred[v]</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>Visited</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

- We update the priority queue with the cost to visit all nodes now that we’ve visited S as follows:
  - We can get these costs by looking at our graph matrix – if the cost of going to s plus the cost of going from s to any other node is less than the cost in the priority queue, we update the cost in the priority queue

### Graph:

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>inf</td>
<td>inf</td>
</tr>
<tr>
<td>A</td>
<td>inf</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>inf</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>inf</td>
</tr>
<tr>
<td>C</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

1. So cost to S is 0
2. Cost from s to a is 2
3. $0+2 < \text{infinity}$, so we update the cost in the priority queue to be 2
   AND Pred of A is S

### Again:

4. Cost to S is 0
5. Cost from s to b is 7
6. $0 + 7 < \text{infinity}$, so we update the cost in the priority queue to be 7
   AND pred of B is S
**Dijkstra’s Algorithm**

**Resulting Vectors (arrays):**

<table>
<thead>
<tr>
<th>V</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D[v]</td>
<td>0</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
</tr>
<tr>
<td>Pred[v]</td>
<td>null</td>
<td>S</td>
<td>S</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>Visited</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Priority Queue (updated with the visited S removed and the new costs with going through node S):

<table>
<thead>
<tr>
<th>V</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D[v]</td>
<td>2</td>
<td>7</td>
<td>Inf</td>
<td>Inf</td>
</tr>
</tbody>
</table>

**Repeat:**

- pick the unvisited vertex with the shortest distance
  - In the above priority queue, it would now be A (with a cost of 2)
- Add A to the set of visited nodes. (Set it to True in the visited array)
- Remove it from the priority queue.
- Recalculate all distances from the original vertex to all other vertices not visited yet by taking the minimum of the distance from the original vertex to vx or the distance from the original vertex to the current vertex + the distance from the current vertex to vx
  - We update the priority queue with the cost to visit all nodes now that we’ve visited A as follows:
    - We can get these costs by looking at our graph matrix – if the cost of going to A plus the cost of going from A to any other node is less than the cost in the priority queue, we update the cost in the priority queue

**Graph:**

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>inf</td>
<td>inf</td>
</tr>
<tr>
<td>A</td>
<td>inf</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>inf</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>inf</td>
</tr>
<tr>
<td>C</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

1. So cost to A is 2
2. Cost from A to B is 3
3. $2 + 3 < 7$, so we update the cost in the priority queue to be 5
   AND we update the pred of B to be A
   (because we went through A to get the shortest path so far)

Again:

1. Cost to A is 2
2. Cost from A to C is 8
3. $2 + 8 < \text{infinity}$, so we update the cost in the priority queue to be 10
   AND we update the pred of C to be A

Again:

4. Cost to A is 2
5. Cost from A to D is 5
6. $2 + 5 < \text{infinity}$, so we update the cost in the priority queue to be 7
   AND we update the pred of C to be A
**Dijkstra’s Algorithm**

**Resulting Vectors:**

<table>
<thead>
<tr>
<th>V</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D[v]</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>Pred[v]</td>
<td>null</td>
<td>s</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>Visited</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Priority Queue (note that a has been removed from the priority queue because it has been visited):

<table>
<thead>
<tr>
<th>V</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D[v]</td>
<td>5</td>
<td>10</td>
<td>7</td>
</tr>
</tbody>
</table>

Repeat:

- Next shortest path: B (with a cost of 5)
  - Add B to the set of visited nodes. (Set it to True in the visited array)
  - Remove it from the priority queue.
  - Recalculate all distances:

**Graph:**

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>inf</td>
<td>inf</td>
</tr>
<tr>
<td>A</td>
<td>inf</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>inf</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>inf</td>
</tr>
<tr>
<td>C</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>Inf</td>
<td>Inf</td>
<td>Inf</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

4. So cost to B is 5
5. Cost from B to C is 1
6. $5 + 1 < 10$, so we update the cost in the priority queue to be 6
   AND we update the pred of C to be B
   (because we went through A to get the shortest path so far)

Again:

4. Cost to B is 5
5. Cost from B to D is inf
6. $5 + inf > 7$, so we DO NOT update the cost in the priority queue

**Resulting Vectors:**

<table>
<thead>
<tr>
<th>V</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D[v]</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Pred[v]</td>
<td>null</td>
<td>s</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>Visited</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Priority Queue (note that a has been removed from the priority queue because it has been visited):

<table>
<thead>
<tr>
<th>V</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D[v]</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Repeat (One last time!):
- Next shortest path: C (with a cost of 6)
  - Add C to the set of visited nodes. (Set it to True in the visited array)
  - Remove it from the priority queue.
  - Recalculate all distances:

<table>
<thead>
<tr>
<th>Graph:</th>
<th>Resulting Vectors:</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>A</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>inf</td>
</tr>
<tr>
<td>B</td>
<td>inf</td>
</tr>
<tr>
<td>C</td>
<td>Inf</td>
</tr>
<tr>
<td>D</td>
<td>Inf</td>
</tr>
</tbody>
</table>

7. So cost to C is 6
8. Cost from C to D is 4
9. $6 + 4 > 7$, so we DO NOT update the cost in the priority queue

Now update D to be true:

<table>
<thead>
<tr>
<th>V</th>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D[v]</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Pred[v]</td>
<td>null</td>
<td>s</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>Visited</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

“*Aim for brevity while avoiding jargon.*”

-Edsger Dijkstra

Finding Minimum Path:
To find the minimum path from any vertex to the starting vertex, START AT THE END VERTEX (the node you want to find the path to) and follow the prev nodes back to the start node (S)

Min Path from S to C:
- In the Pred array, go to C’s index (each vertex is assigned a number). In the C index of Pred is B, so B is C’s predecessor.
- Now repeat. Go to B’s index, and in the Pred vertex A is B’s predecessor
- Go to A. S is A’s predecessor.

So the path from S to C is S->A->B->C
PROBLEMS:

1. (4 pts) Represent the following Graph as a Matrix:

   ![Graph](image)

   (1 pt) 1b. Is the above graph directed or undirected? ______________________________________

2. (4 pts) Represent the following Graph as a Matrix:

   ![Graph](image)

   (1 pt) 1b. Is the above graph directed or undirected? ______________________________________

3. (2 pts) Given the priority queue, how do you choose the next vertex to add to the visited list?

   4. (3 pts) In a graph with vertices A,B,C,D,E, and S. You’ve visited S, and the latest vertex to be visited is B. Now what is the formula you’d use to update the Distance Vector?

   (2 pts) 4b. If the path is shorter through B, what other vector must you update?
5. **(3 pts)** Given the Pred array, show the path formed by Dijkstra’s algorithm between w and g:

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>a</th>
<th>r</th>
<th>f</th>
<th>l</th>
<th>t</th>
<th>h</th>
<th>v</th>
<th>e</th>
<th>o</th>
<th>y</th>
<th>g</th>
<th>d</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>w</td>
<td>a</td>
<td>a</td>
<td>i</td>
<td>r</td>
<td>t</td>
<td>p</td>
<td>l</td>
<td>h</td>
<td>u</td>
<td>o</td>
<td>s</td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>

6. **(8 pts)** Write in pseudocode (logical C++ code that should be readable as code but doesn’t necessarily need to compile) the method for finding the shortest path between some vertex x and another vertex y using an array that is the predecessor array. You may assume the following:
   - The predecessor array stores predecessors as integers (that act as indices)
   - You are creating the path as a linked list:
     - You may assume you have the typical methods associated with a linked list (push, pop, insert, addAtFirst, etc.)

7. **(2 pts)** Give me your favorite Dijkstra quote:
8. (12 pts) Given the following Graph, show all the steps (visited, distance, and pred vectors and priority queues) to find the shortest path from L to every other vertex in the graph:

(2 pts) 8b. Find the shortest path from L to T: