Algorithms and Data Structures for Data Science
Shortest Path Algorithms

CS 277
Brad Solomon

April 24, 2023
Exam 3 Signups Available

April 24 — April 27

Very limited window for makeup exams (since end of semester is near)

Covers content from week 10 — 14
Mini-Project 3

Average: 88%  Median: 99%

An optional final reflection form: Mini-Project Reflection Feedback Form
Please fill out ICES Evaluations

Feedback is important for the development of the class

This is especially important for this class as it is very new and growing
Learning Objectives

Introduce complexity of weights to shortest path / MST problem

Conceptualize Dijkstra's Shortest Path Algorithm

Code Dijkstra's in Python and see how to use it in Python
Shortest Path
Dijkstra's Shortest Path (Distances)

1) Create data structs to store distances
   Create data structs to store visited
2) Initialize all distances to $\infty$ (and source to 0)
3) While there exists an unvisited vertex:
   Visit the current nearest vertex
   Update distances based on current edges

A:  B:  C:  D:  E:  F:  G:  H:

Visited
Dijkstra's Shortest Path (Distances)

1) Create data structs to store distances

Create data structs to store visited

2) Initialize all distances to $\infty$ (and source to 0)

3) While there exists an unvisited vertex:
   - Visit the current nearest vertex
   - Update distances based on current edges

A: 0  B: 2  C: 3  D: 1  E: 3  F: 3  G: 5  H: 2

Visited: D, B, H, C, E, F, G
Dijkstra's Shortest Path (Full Paths)

**Graph Diagram**

A connected graph with vertices labeled A, B, C, D, E, F, G, and H. The edges are labeled with distances:
- A to B: 10
- A to D: 3
- B to C: 7
- C to E: 6
- C to H: 4
- D to E: 7
- D to F: 5
- E to F: 2
- E to G: 5
- F to G: 4
- G to H: 3

**Distance Table**

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Distance</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra's Shortest Path (Full Paths)

**Graph Representation:**

- **Vertices:** A, B, C, D, E, F, G, H
- **Edges:** A-B (5), A-D (3), A-F (7), B-C (7), B-D (10), B-G (5), C-D (6), C-E (3), D-E (2), E-F (5), F-G (4), G-H (2), H-C (6)

**Distance Table:**

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Distance</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>17</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>15</td>
<td>B</td>
</tr>
<tr>
<td>E</td>
<td>12</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>7</td>
<td>A</td>
</tr>
<tr>
<td>G</td>
<td>11</td>
<td>F</td>
</tr>
<tr>
<td>H</td>
<td>21</td>
<td>C</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm (SSSP)

When we will visit B in the following graph?
Dijkstra’s Algorithm (SSSP)

When we will visit B in the following graph?
Dijkstra’s Algorithm (SSSP)

When we will visit B in the following graph?
Dijkstra’s Algorithm (SSSP)

How does Dijkstra's handle a negative weight cycle?
Dijkstra’s Algorithm (SSSP)

How does Dijkstra's handle a negative weight cycle?

Shortest Path (A \rightarrow E): \[ A \rightarrow F \rightarrow E \rightarrow (C \rightarrow H \rightarrow G \rightarrow E)^* \]

Length: 12 \hspace{2cm} \text{Length: -5 (repeatable)}
Dijkstra’s Algorithm (SSSP)

How does Dijkstra's handle a single negative edge?
What if I wanted to get the shortest path from A to G but stopping at L along the way?
Floyd-Warshall’s Algorithm is an alternative to Dijkstra in the presence of **negative-weight edges** (not negative weight cycles).

```plaintext
1 FloydWarshall(G):
2   Let d be a adj. matrix initialized to +inf
3   foreach (Vertex v : G):
4       d[v][v] = 0
5   foreach (Edge (u, v) : G):
6       d[u][v] = cost(u, v)
7
8   foreach (Vertex u : G):
9       foreach (Vertex v : G):
10          foreach (Vertex w : G):
11            if (d[u, v] > d[u, w] + d[w, v])
12              d[u, v] = d[u, w] + d[w, v]
```
Floyd-Warshall Algorithm

FloydWarshall(G):
Let d be a adj. matrix initialized to +inf
foreach (Vertex v : G):
    d[v][v] = 0
foreach (Edge (u, v) : G):
    d[u][v] = cost(u, v)
foreach (Vertex u : G):
    foreach (Vertex v : G):
        foreach (Vertex w : G):
            if (d[u, v] > d[u, w] + d[w, v])
                d[u, v] = d[u, w] + d[w, v]

def floydwarshall(inGraph):
    dist = {}
    for u in inGraph.edges.keys():
        dist[u] = {}  # Corrected to use {} instead of {} for consistency
        for v in inGraph.edges.keys():
            dist[u][v] = float('inf')
    for u in inGraph.edges.keys():
        dist[u][u] = 0
    for u in inGraph.edges.keys():
        for e in inGraph.edges[u]:
            v = e[0]
            w = e[1]
            dist[u][v] = w
    for u in inGraph.edges.keys():
        for v in inGraph.edges.keys():
            for w in inGraph.edges.keys():
                if dist[u][v] > dist[u][w] + dist[w][v]:
                    dist[u][v] = dist[u][w] + dist[w][v]
Floyd-Warshall Algorithm

1. FloydWarshall(G):
2.   Let d be a adj. matrix initialized to +inf
3.   foreach (Vertex v : G):
4.       d[v][v] = 0
5.   foreach (Edge (u, v) : G):
6.       d[u][v] = cost(u, v)
Floyd-Warshall Algorithm

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>-1</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>

Let us consider \( w = A \):

|   |   |   |   |   |
|---|---|---|---|
| 8 | foreach (Vertex \( u \) : G):
| 9  |    |   |   |
| 10 |  foreach (Vertex \( v \) : G):
| 11 |      |   |   |
| 12 | if (\( d[u, v] > d[u, w] + d[w, v] \))
|    | \( d[u, v] = d[u, w] + d[w, v] \) |   |   |   |
Floyd-Warshall Algorithm

Let us consider \( w = A \):

\[
\begin{align*}
&\text{B} \rightarrow \text{C} \quad 4 \quad \text{vs.} \quad \text{B} \rightarrow \text{A} \rightarrow \text{C} +\infty \\
&\text{B} \rightarrow \text{D} \quad 3 \quad \text{vs.} \quad \text{B} \rightarrow \text{A} \rightarrow \text{D} +\infty \\
&\text{C} \rightarrow \text{B} \quad +\infty \quad \text{vs.} \quad \text{C} \rightarrow \text{A} \rightarrow \text{B} +\infty \\
&\text{C} \rightarrow \text{D} \quad -2 \quad \text{vs.} \quad \text{C} \rightarrow \text{A} \rightarrow \text{D} +\infty \\
&\text{D} \rightarrow \text{B} \quad +\infty \quad \text{vs.} \quad \text{D} \rightarrow \text{A} \rightarrow \text{B} \\
&\text{D} \rightarrow \text{C} \quad +\infty \quad \text{vs.} \quad \text{D} \rightarrow \text{A} \rightarrow \text{C}
\end{align*}
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & D \\
\hline
A & 0 & -1 & \infty & \infty \\
B & \infty & 0 & 4 & 3 \\
C & \infty & \infty & 0 & -2 \\
D & 2 & \infty & \infty & 0 \\
\hline
\end{array}
\]
Floyd-Warshall Algorithm

\[
\begin{align*}
8 & \quad \text{foreach (Vertex } u : G) : \\
9 & \quad \text{foreach (Vertex } v : G) : \\
10 & \quad \text{foreach (Vertex } w : G) : \\
11 & \quad \text{if (} d[u, v] > d[u, w] + d[w, v] \text{)} \\
12 & \quad \quad d[u, v] = d[u, w] + d[w, v]
\end{align*}
\]

Let us consider \( k = A \):

- \( B \rightarrow C \): 4 vs. \( B \rightarrow A \rightarrow C \): +\( \infty \)
- \( B \rightarrow D \): 3 vs. \( B \rightarrow A \rightarrow D \): +\( \infty \)
- \( C \rightarrow B \): +\( \infty \) vs. \( C \rightarrow A \rightarrow B \): +\( \infty \)
- \( C \rightarrow D \): -2 vs. \( C \rightarrow A \rightarrow D \): +\( \infty \)
- \( D \rightarrow B \): +\( \infty \) vs. \( D \rightarrow A \rightarrow B \): 1
- \( D \rightarrow C \): +\( \infty \) vs. \( D \rightarrow A \rightarrow C \): +\( \infty \)
Floyd-Warshall Algorithm

FloydWarshall(G):
Let d be a adj. matrix initialized to +inf
foreach (Vertex v : G):
d[v][v] = 0
foreach (Edge (u, v) : G):
d[u][v] = cost(u, v)
foreach (Vertex u : G):
foreach (Vertex v : G):
foreach (Vertex w : G):
if (d[u, v] > d[u, w] + d[w, v])
d[u, v] = d[u, w] + d[w, v]

A | B | C | D
---|---|---|---
A | 0 | -1 | 3 | 1
B | 5 | 0 | 4 | 2
C | 0 | -1 | 0 | -2
D | 2 | 1 | 5 | 0

![Graph with distances](image)