Dijkstra’s Algorithm Overview:
- The overall logic is the same as Prim’s Algorithm
- We will modify the code in only two places – both involving the update to the distance metric.
- The result is a directed acyclic graph or DAG

**Pseudocode for Dijkstra’s SSSP Algorithm**

```
DijkstraSSSP(G, s):
    Input: G, Graph;
    s, vertex in G, starting vertex of algorithm
    Output: T, DAG with shortest paths (and distances) to s

    foreach (Vertex v : G):
        d[v] = +inf
        p[v] = NULL
        d[s] = 0
    PriorityQueue Q // min distance, defined by d[v]
    Q.buildHeap(G.vertices())
    Graph T // "labeled set"

    repeat n times:
        Vertex m = Q.removeMin()
        T.add(m)
        foreach (Vertex v : neighbors of m not in T):
            if cost(u, v) + d[u] < d[v]:
                d[v] = cost(u, v) + d[u]
                p[v] = m

    return T
```

**Backtracking in Dijkstra**
Dijkstra’s Algorithm gives us the shortest path from a single source to every connected vertex:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>NULL</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>F</td>
<td>A</td>
<td>F</td>
<td>C</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>10</td>
<td>17</td>
<td>15</td>
<td>12</td>
<td>7</td>
<td>11</td>
<td>21</td>
</tr>
</tbody>
</table>

**Examples:** How is a single heavy-weight path vs. many light-weight paths handled?

Ex 1:

Ex 2:

What about undirected graphs?
**Dijkstra:** What if we have a negative-weight cycle?

![Graph with negative-weight cycle](image)

**Dijkstra:** What if we have a minimum-weight edge, without having a negative-weight cycle?

![Graph with minimum-weight edge](image)

Dijkstra makes an assumption:

**Dijkstra:** What is the running time?

---

**Floyd-Warshall Algorithm**

Floyd-Warshall's Algorithm is an alternative to Dijkstra in the presence of negative-weight edges (but not negative weight cycles).

<table>
<thead>
<tr>
<th>Pseudocode for Floyd-Warshall’s Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. FloydWarshall(G):</td>
</tr>
<tr>
<td>2. Input: G, Graph;</td>
</tr>
<tr>
<td>3. Output: d, an adjacency matrix of distances between all vertex pairs</td>
</tr>
<tr>
<td>4. Let d be a adj. matrix initialized to +inf</td>
</tr>
<tr>
<td>5. foreach (Vertex v : G):</td>
</tr>
<tr>
<td>6. d[v][v] = 0</td>
</tr>
<tr>
<td>7. foreach (Edge (u, v) : G):</td>
</tr>
<tr>
<td>8. d[u][v] = cost(u, v)</td>
</tr>
<tr>
<td>9. foreach (Vertex u : G):</td>
</tr>
<tr>
<td>10. foreach (Vertex v : G):</td>
</tr>
<tr>
<td>11. foreach (Vertex w : G):</td>
</tr>
<tr>
<td>12. if d[u, v] &gt; d[u, w] + d[w, v]:</td>
</tr>
<tr>
<td>14. return d</td>
</tr>
</tbody>
</table>

---

**Running Floyd-Warshall’s Algorithm**

![Graph with Floyd-Warshall's algorithm](image)

---

**CS 225 – Things To Be Doing:**

1. Exam #13 (makeup exam) starts Monday
2. MP7 due Monday, Dec. 11 at 11:59pm
3. lab_ml due Sunday, Dec. 10 at 11:59pm
4. Multi-day “puzzle” POTDs available M/W/F