THE WEIGHTED SHORTEST PATH PROBLEM
Weighted Shortest Path Problem

Single-source shortest-path problem:
Given as input a weighted graph, $G = (V, E)$, and a distinguished starting vertex, $s$, find the shortest weighted path from $s$ to every other vertex in $G$.

Dijkstra’s algorithm (also called uniform cost search)
- Use a priority queue in general search/traversal
- Keep tentative distance for each vertex giving shortest path length using vertices visited so far.
- Record vertex visited before this vertex (to allow printing of path).
- At each step choose the vertex with smallest distance among the unvisited vertices (greedy algorithm).
Example Network
The pseudo code for Dijkstra’s algorithm assumes the following structure for a Vertex object

class Vertex
{
    public List adj;         //Adjacency list
    public boolean known;
    public DisType dist;     //DistType is probably int
    public Vertex path;
    //Other fields and methods as needed
}
Dijkstra’s Algorithm

```java
void dijkstra(Vertex start) {
    for each Vertex v in V {
        v.dist = Integer.MAX_VALUE;
        v.known = false;
        v.path = null;
    }

    start.distance = 0;

    while there are unknown vertices {
        v = unknown vertex with smallest distance
        v.known = true;
        for each Vertex w adjacent to v
            if (!w.known)
                if (v.dist + weight(v, w) < w.distance) {
                    decrease(w.dist to v.dist+weight(v, w))
                    w.path = v;
                }
    }
}
```
Correctness of Dijkstra’s Algorithm

- The algorithm is correct because of a property of shortest paths:
- If $P_k = v_1, v_2, ..., v_j, v_k$, is a shortest path from $v_1$ to $v_k$, then $P_j = v_1, v_2, ..., v_j$, must be a shortest path from $v_1$ to $v_j$. Otherwise $P_k$ would not be as short as possible since $P_k$ extends $P_j$ by just one edge (from $v_j$ to $v_k$).
- $P_j$ must be shorter than $P_k$ (assuming that all edges have positive weights). So the algorithm must have found $P_j$ on an earlier iteration than when it found $P_k$.
- i.e. Shortest paths can be found by extending earlier known shortest paths by single edges, which is what the algorithm does.
The running time depends on how the vertices are manipulated.

The main ‘while’ loop runs $O(|V|)$ time (once per vertex).

Finding the “unknown vertex with smallest distance” (inside the while loop) can be a simple linear scan of the vertices and so is also $O(|V|)$. With this method the total running time is $O(|V|^2)$. This is acceptable (and perhaps optimal) if the graph is dense ($|E| = O(|V|^2)$) since it runs in linear time on the number of edges.

If the graph is sparse, ($|E| = O(|V|)$), we can use a priority queue to select the unknown vertex with smallest distance, using the deleteMin operation ($O(\lg |V|)$). We must also decrease the path lengths of some unknown vertices, which is also $O(\lg |V|)$. The deleteMin operation is performed for every vertex, and the “decrease path length” is performed for every edge, so the running time is $O(|E| \lg |V| + |V| \lg |V|) = O((|V| + |E|) \lg |V|) = O(|E| \lg |V|)$ if all vertices are reachable from the starting vertex.
Dijkstra and Negative Edges

- Note in the previous discussion, we made the assumption that all edges have positive weight. If any edge has a negative weight, then Dijkstra’s algorithm fails. Why is this so?
- Suppose a vertex, u, is marked as “known”. This means that the shortest path from the starting vertex, s, to u has been found.
- However, it’s possible that there is negatively weighted edge from an unknown vertex, v, back to u. In that case, taking the path from s to v to u is actually shorter than the path from s to u without going through v.
- Other algorithms exist that handle edges with negative weights for weighted shortest-path problem.
All-pairs shortest paths...

“Floyd-Warshall algorithm”

Matrix representation

D^0

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>8</td>
<td>13</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>C</td>
<td>-</td>
<td>9</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>E</td>
<td>-</td>
<td>-</td>
<td>11</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>
All-pairs shortest paths...

\[ D^0 = (d^0_{ij}) \]

\[
\begin{pmatrix}
A & 0 & 8 & 13 & - & 1 \\
B & - & 0 & - & 6 & 12 \\
C & - & 9 & 0 & - & - \\
D & 7 & - & 0 & 0 & - \\
E & - & - & - & 11 & 0
\end{pmatrix}
\]

\[ D^1 = (d^1_{ij}) \]

\[
\begin{pmatrix}
A & 0 & 8 & 13 & - & 1 \\
B & - & 0 & - & 6 & 12 \\
C & - & 9 & 0 & - & - \\
D & 7 & - & 15 & 0 & 8 \\
E & - & - & - & 11 & 0
\end{pmatrix}
\]

\[ d^k_{ij} = \text{shortest distance from } i \text{ to } j \text{ through } \{1, \ldots, k\} \]
All-pairs shortest paths...

\[
D^2 = (d^2_{ij}) \\
\begin{pmatrix}
0 & 8 & 13 & |14| & 1 \\
- & 0 & - & 6 & 12 \\
- & 9 & 0 & |15| & 21 \\
7 & 15 & 0 & 0 & 8 \\
- & - & - & 11 & 0 \\
\end{pmatrix}
\]

\[
D^3 = (d^3_{ij}) \\
\begin{pmatrix}
0 & 8 & 13 & 14 & 1 \\
- & 0 & - & 6 & 12 \\
- & 9 & 0 & |15| & 21 \\
7 & 9 & 0 & 0 & 8 \\
- & - & - & 11 & 0 \\
\end{pmatrix}
\]

\[
D^4 = (d^4_{ij}) \\
\begin{pmatrix}
0 & 8 & 13 & 14 & 1 \\
13 & 0 & |6| & 6 & 12 \\
22 & 9 & 0 & |15| & 21 \\
7 & 9 & 0 & 0 & 8 \\
18 & 20 & 11 & 11 & 0 \\
\end{pmatrix}
\]

\[
D^5 = (d^5_{ij}) \\
\begin{pmatrix}
0 & 8 & |12| & 12 & 1 \\
13 & 0 & 6 & 6 & 12 \\
22 & 9 & 0 & |15| & 21 \\
7 & 9 & 0 & 0 & 8 \\
18 & 20 & 11 & 11 & 0 \\
\end{pmatrix}
\]

to store the path, another matrix can track the last intermediate vertex.
Floyd-Warshall Pseudocode

Input: \( D^0 = (d_{ij}^0) \) (the initial edge-cost matrix)
Output: \( D^n = (d_{ij}^n) \) (the final path-cost matrix)

for \( k = 1 \) to \( n \) // intermediate vertices considered
  for \( i = 1 \) to \( n \) // the “from” vertex
    for \( j = 1 \) to \( n \) // the “to” vertex
      \[ d_{ij}^k = \min\{ d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1} \} \]
      best, ignoring vertex \( k \)
      best, including vertex \( k \)