Shortest Paths in Edge Weighted Graphs

**General input:** directed or undirected graph with weights on the edges.

**Example**

shortest path C to E: C,D,E, of weight 3.

Two ways to store edge-weighted graphs

<table>
<thead>
<tr>
<th>graph G</th>
<th>O(n²) size</th>
<th>V × V matrix of edge weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex number</td>
<td>(edge number, edge weight)</td>
<td></td>
</tr>
</tbody>
</table>

O(n+m) size

<table>
<thead>
<tr>
<th>Edges</th>
<th>List of edges incident to each vertex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights</td>
<td>Vertices</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>1,3 1,2 2,3 2,5 3,5 4,5 3,4</td>
</tr>
<tr>
<td>2 3 5 0 9</td>
<td>1 1 2</td>
</tr>
<tr>
<td>3 5</td>
<td>1 2 4 3</td>
</tr>
<tr>
<td>4 6</td>
<td>1 1 7 3 5</td>
</tr>
<tr>
<td>5</td>
<td>1 1 2 4 3</td>
</tr>
<tr>
<td>6</td>
<td>1 1 2 4 3</td>
</tr>
</tbody>
</table>

List of edges incident to each vertex
Shortest Paths in Edge Weighted Graphs

**General input:** directed or undirected graph with weights on the edges.

Versions of the problem:
1. Given vertices s, t, find shortest s-t path. This seems to involve solving 2.
2. “single source shortest path problem”. Given s, find shortest s-v path for all v.
3. “all pairs shortest path problem”. Find shortest u-v path for all u and v.

Recall that Dijkstra’s algorithm solves (2) for non-negative weight edges.

What happens with negative weight edges?

What is the shortest path from s to t?
2. single source shortest path problem for directed graphs.
   Given $s$, find shortest $s-v$ path for all $v$.
   
   a. no cycles. $O(n+m)$. this lecture
   b. no negative weights. $O(m \log n)$. Dijkstra’s algorithm. last lecture
   c. allow negative weights and allow cycles but NO negative weight cycles. $O(nm)$.
      Bellman-Ford algorithm — dynamic programming. this lecture

3. all pairs shortest path problem for directed graphs.
   Find shortest $u-v$ path for all $u$ and $v$.
   
   NO negative weight cycles. $O(n^3)$.
   Floyd-Warshall algorithm — dynamic programming this lecture

Note: what about undirected graphs?
Outline of this lecture.

2. single source shortest path problem for directed graphs.
   Given s, find shortest s-v path for all v.
   
   a. no cycles. $O(n+m)$. this lecture
   b. no negative weights. $O(m \log n)$. Dijkstra’s algorithm. last lecture
   c. allow negative weights and allow cycles but NO negative weight cycles. $O(nm)$.
   Bellman-Ford algorithm — dynamic programming. this lecture

3. all pairs shortest path problem for directed graphs.
   Find shortest u-v path for all u and v.
   NO negative weight cycles. $O(n^3)$.
   Floyd-Warshall algorithm — dynamic programming this lecture
Single source shortest paths in a directed acyclic graph (DAG)

Given s, find shortest s-v path for all v.

Use topological sort, to find vertex order 1, 2, . . ., n so every directed edge (i,j) has i < j. Recall (from Lecture 14) that DFS can find this in O(n+m) time.

If v < s then there is no path from s to v. So throw away vertices v with v < s. We can then assume that s=1.

Let d(i) be the distance from vertex 1 to vertex i.

```
initialize d(i) = infty for all i;  d(1) := 0
for i = 1..n
    for every edge (i,j)
        d(j) := min { d(j),  d(i) + w(i,j) }
```

Correctness. Prove by induction on i = 1 . . n that when we start the loop for i, then d(i) is the min distance from 1 to i.

Runtime. O(n+m)
Dynamic programming for shortest paths.

We will use dynamic programming for two shortest path algorithms in directed graphs.

2c. Single source shortest paths. NO negative weight cycles. O(nm).

**Bellman-Ford algorithm.**

3. All pairs shortest paths. NO negative weight cycles. O(n^3).

**Floyd-Warshall algorithm.**

Idea of dynamic programming for shortest paths:

if the shortest u-v path goes through vertex x
(we can try all x) then it consists of:
shortest u-x path + shortest x-v path

these are subproblems

In what way are these subproblems “smaller”? Two possibilities:

1. they use fewer edges. This leads to dynamic programming where we try paths
   of ≤ 1 edge, paths of ≤ 2 edges, . . .

   Use this for single source shortest paths.

2. they don’t use the vertex x. This leads to dynamic programming where we try
   paths using only vertex 1, using only vertices 1, 2, using only, . . .

   Use this for all pairs shortest paths.
Bellman-Ford Algorithm. Single source.

Given an edge-weighted directed graph with no negative weight cycles and a source vertex $s$, find shortest paths from $s$ to all vertices $v$.

Let $d_i (v) =$ length of shortest path from $s$ to $v$ using $\leq i$ edges.
Then we want $d_{n-1}(v)$. Why?

Initialize: $d_0(v) := \text{infty}$ for all $v$; $d_0(s) := 0$

General formula $i = 1, 2, \ldots, n-1$:

$$d_i (v) = \min \left\{ d_{i-1}(v), \min_u \{d_{i-1}(u) + w(u,v)\} \right\}$$
(use $\leq i-1$ edges)
(use $i$ edges — try all choices for the last edge)
**Bellman-Ford Algorithm.** Single source.

General formula:

\[ d_i(v) = \min \begin{cases} 
  d_{i-1}(v) & \text{(use } \leq i-1 \text{ edges)} \\
  \min_{u} \{d_{i-1}(u) + w(u,v)\} & \text{(use } i \text{ edges — try all choices for the last edge)}
\end{cases} \]

**Correctness.** By induction. Initialization gives \(d_0\). Assuming \(d_{i-1}\) is correct, the above formula gives the correct \(d_i\) because we try all possibilities.

**Bellman-Ford Algorithm.**

\begin{align*}
  d_0(v) &:= \infty \text{ for all } v; \quad d_0(s) := 0 \\
  \text{for } i = 2 \ldots n-1 \\
  & \quad \text{for each vertex } v \\
  & \quad \quad d_i(v) := d_{i-1}(v) \\
  & \quad \quad \text{for each edge } (u,v) \\
  & \quad \quad \quad \text{Note: from } v \text{ we want edges IN to } v \\
  & \quad \quad \quad d_i(v) := \min \{d_i(v), d_{i-1}(u) + w(u,v)\}
\end{align*}

**Runtime:**
**Bellman-Ford Algorithm.** Single source.

We can save space — don’t use $d_i(v)$, just reuse the same $d(v)$. We can also simplify the code, and avoid the need for in-edges.

\[
\begin{align*}
\text{Bellman-Ford Algorithm.} \\
& d(v) := \text{infty for all } v; \; d(s) := 0 \\
& \text{for } i = 1 \ldots n-1 \\
& \quad \text{for each edge } (u,v) \\
& \quad \quad d(v) := \min \{ d(v), d(u) + w(u,v) \}
\end{align*}
\]

Note the curious fact that $i$ does not appear inside the loop!

**Exercise.** Convince yourself that this code does the same thing. (In this form, it’s more mysterious why it works.)

**Exercise.** Show that we can exit the top-level loop early after an iteration in which no $d$ value changes.
Bellman-Ford Algorithm. Single source.
The algorithm finds the lengths of shortest paths from source s.

Finding the actual shortest paths.

Store a parent pointer with each vertex v. When we update
\[ d(v) := d(u) + w(u,v) \]
then update
\[ \text{parent}(v) := u \]

Then the path from s to v can be recovered by following parent pointers backwards from v.
Bellman-Ford Algorithm. Single source.
The algorithm is for directed graphs with no negative cycles.

Testing if a directed graph has a negative cycle. \(O(mn)\) time.

1. Testing for a negative cycle reachable from \(s\)
   Run one more iteration of Bellman-Ford and see if any \(d\) value changes.

   **Exercise.** See why this works.

   **Example.**

2. Testing for a negative cycle anywhere in the graph
Dynamic programming for shortest paths.

We will use dynamic programming for two shortest path algorithms in directed graphs.

2c. Single source shortest paths. NO negative weight cycles. $O(nm)$.

Bellman-Ford algorithm.

3. All pairs shortest paths. NO negative weight cycles. $O(n^3)$.

Floyd-Warshall algorithm.

Idea of dynamic programming for shortest paths:

if the shortest $u-v$ path goes through vertex $x$
(we can try all $x$) then it consists of:
shortest $u-x$ path + shortest $x-v$ path

these are subproblems

In what way are these subproblems “smaller”? Two possibilities:

1. they use fewer edges. This leads to dynamic programming where we try paths of $\leq 1$ edge, paths of $\leq 2$ edges, . . .

   Use this for single source shortest paths.

2. they don’t use the vertex $x$. This leads to dynamic programming where we try paths using only vertex 1, using only vertices 1, 2, using only, . . .

   Use this for all pairs shortest paths.
**Floyd-Warshall Algorithm.** All pairs.

Given an edge-weighted directed graph with no negative weight cycles find shortest paths from u to v for all pairs of vertices u,v.

We can output the distances as an nxn matrix D[u,v].

What if we use Bellman-Ford with each vertex u as a source?

We will get a faster algorithm using dynamic programming where intermediate paths use only a subset of the vertices. Recall that V = \{1, 2, \ldots, n\}.

Let \(D_i[u,v] = \text{length of shortest path from } u \text{ to } v \text{ using intermediate vertices from } \{1, 2, \ldots, i\} \).

Solve subproblems \(D_i[u,v] \) for all \(u,v\) as \(i\) goes from 0 to \(n\).

The final answers are \(D_n[u,v] \).

Initialize:

\[
D_0[u,v] = \begin{cases} 
0 & \text{if } u = v \\
\infty & \text{otherwise} \\
w(u,v) & \text{if } (u,v) \text{ is an edge}
\end{cases}
\]

General formula \(i = 1, 2, \ldots n:\)

\[
D_i[u,v] = \min \left\{ \begin{array}{ll}
D_{i-1}[u,v] & \text{(don’t use vertex } i) \\
D_{i-1}[u,i] + D_{i-1}[i,v] & \text{(use vertex } i) 
\end{array} \right.
\]
Floyd-Warshall Algorithm. All pairs.

General formula $i = 1, 2, \ldots, n$:

$$D_i[u,v] = \min \left\{ D_{i-1}[u,v], D_{i-1}[u,i] + D_{i-1}[i,v] \right\}$$

Correctness. By induction. Initialization gives $D_0$. Assuming $D_{i-1}$ is correct, the above formula gives the correct $D_i$ because we try all possibilities.

Floyd-Warshall Algorithm.

$D_0[u,v] := \infty$ for all $u,v$

for every vertex $u$, $D_0[u,u] := 0$

for every edge $[u,v]$, $D_0[u,v] := w(u,v)$

for $i = 1 \ldots n$
  
  for $u = 1 \ldots n$
    
    for $v = 1 \ldots n$
      
      $D_i[u,v] := \min \{ D_{i-1}[u,v], D_{i-1}[u,i] + D_{i-1}[i,v] \}$

Runtime:  
Space:
Floyd-Warshall Algorithm. All pairs.

Reduce space to $O(n^2)$ by reusing the same $D(u,v)$:

$$D[u,v] := \min \{ D[u,v], D[u,i] + D[i,v] \}$$

The algorithm finds the **lengths** of shortest paths.

Finding the actual shortest paths.

Compute $\text{Next}[u,v] =$ the first vertex (after $u$) on a shortest $u$ to $v$ path.

If we update

$$D[u,v] := D[u,i] + D[i,v]$$

then update

$$\text{Next}[u,v] := \text{Next}[u,i]$$

Note that we recover the path going forward (whereas Bellman-Ford used parent pointers going from back to front).
Summary of Lecture 17

- Bellman-Ford algorithm, single source shortest paths
- Floyd-Warshall algorithm, all pairs shortest paths

What you should know from Lecture 17:

- issues (wrt shortest paths) with negative weight cycles, undirected graphs
- dynamic programming for shortest paths
- how to detect negative cycles in directed graphs
- Bellman-Ford and Floyd-Warshall algorithms, runtimes
- how to find the actual paths

Next:

- NP-completeness, e.g. why researchers think there is no polynomial time algorithm to find shortest simple paths