Below is an example of a graph in which Dijkstra’s Algorithm fails.

![Graph Diagram]

The shortest path from $A$ to $D$ is $ACBD$, which has weight $-734$. Dijkstra’s algorithm instead reports that the weight of the shortest path is $2$.

This happens because Dijkstra’s assumes that no distance, when modified, has yet been used to find other distances. By the time $ACB$ is explored, the path $AB$ has already been extended to $ABD$. This results in the incorrect shortest distance from $A$ to $D$ of $2$ with predecessor $B$ even though the shortest distance from $A$ to $B$ is reported as $-735$.

We could instead follow the predecessors back to $A$. This will give us the shortest path $ACBD$ from $D$ back to $A$. We could then calculate the shortest distance by adding the edge weights along this path. But this is not always the case!

**Problem 1.**

![Graph Diagram]

Give (integer) values for $x$ and $y$ which cause Dijkstra’s to fail to find the shortest path to $F$ even when the shortest path is constructed via following $F$’s predecessors back to the start vertex $A$ (i.e. Dijkstra’s incorrectly reports that $F$’s predecessor is $E$). In addition, give an intuitive explanation of why your choice of $x$ and $y$ causes this failure.

**Solution 1.**
Problem 2. To fix this problem, we give up our ability to know when a vertex is “done”. Instead, we seek shortest paths using a maximum number of intermediate edges.

Let \( G = (V, E) \) be a graph with edge weights \( w : E \rightarrow \mathbb{Q} \). Define \( D_{s,i}(u) \) to be the shortest distance from the start vertex \( s \) to vertex \( u \) using at most \( i \geq 0 \) intermediate edges. The trick to computing \( D_{s,i} \) is that there is always a last edge on a path, so we can obtain

\[
D_{s,i+1}(u) = \min \left( D_{s,i}(u), \min_{(v,u) \in E} (D_{s,i}(v) + w(v,u)) \right).
\]

Moreover, any path in a graph requires at most \(|V| - 1\) edges.

Explain why this recursive definition is correct (you need not prove it) by examining its cases.

Solution 2. The shortest path to \( u \) from \( s \) using at most \( i + 1 \) edges either

- uses less than \( i + 1 \) edges, in which case the shortest path is contained within \( D_{s,i} \), or
- it must have a last edge \((v, u)\) from some vertex \( v \neq u \).

In the latter case, this last edge leaves us with only \( i \) edges additional available to form our shortest path. We already have a path from \( v \) to \( u \), so we need to augment it with the shortest path from \( s \) to \( v \). Fortunately, we have this value in \( D_{s,i}(v) \)!

Problem 3. Fill in the chart of values for \( D_{A,i}(u) \) below to calculate the shortest paths in 1 (for your values of \( x \) and \( y \)). Unreachable vertices have distance \( \infty \), and the only vertex reachable from \( A \) via no intermediate edges is \( A \) itself.

<table>
<thead>
<tr>
<th>( i )</th>
<th>( u )</th>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
<th>( D )</th>
<th>( E )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution 3.
Problem 4. Propose a (psuedocode) algorithm that, on input graph $G = (V, E)$, weighting $w : E \to \mathbb{Q}$, and start vertex $s \in V$, calculates the shortest path from $s$ to every vertex in $V$ using the previous two problems as a guide. You do not need to keep predecessor data, though you may if you wish.

Solution 4.

Algorithm 1: Bellman-Ford($G = (V, E), s, w$

// Initialize the map $D$
// Only the last two rows of $D$ are needed
// Previous rows can be reused/discarded for space
Let $D$ be a map from $V \times \mathbb{Z}_{|V|}$ to $\mathbb{Q}$
Initialize $D(u, 0) = 0$ for all $u \in V$
Assign $D(s, 0) = 0$

for $i = 1$ to $|V| - 1$ do
  foreach $(u, v) \in E$ do
    if $D(v, i - 1) < D(u, i - 1) + w(u, v)$ then
      Assign $D(v, i) = D(u, i - 1) + w(u, v)$
    else
      Assign $D(v, i) = D(v, i - 1)$
  return $D$'s last row

Problem 5. What is the worst-case runtime of your algorithm? Why?

Solution 5. The worst-case runtime is $O(VE)$ regardless of input. The outer loop executes the inner loop $V$ times, and each inner loop takes $O(E)$ time.

Problem 6. The algorithm you gave in the previous problem (probably) has a scenario in which it fails. The shortest path is not always the shortest walk. Under what circumstances can this occur? In other words, when is there not a shortest path?

Solution 6. When there is a negative weight cycle in the graph.

Problem 7. Detecting this situation is actually quite easy. If $D_{s,|V|}(u) < D_{s,|V|-1}(u)$ for any $u \in V$, then you have an unsolvable problem (interestingly, this is also an efficient way to detect this situation). If necessary, amend your algorithm to detect this and report it/throw an error if necessary.

Solution 7.
Algorithm 2: Bellman-Ford($G = (V,E), s, w$

// Initialize the map $D$
// Only the last two rows of $D$ are needed
// Previous rows can be reused/discarded for space
Let $D$ be a map from $V \times \mathbb{Z}_{|V|}$ to $\mathbb{Q}$
Initialize $D(u,0) = 0$ for all $u \in V$
Assign $D(s,0) = 0$

for $i = 1$ to $|V| - 1$ do
  foreach $(u,v) \in E$ do
    if $D(v,i - 1) < D(u,i - 1) + w(u,v)$ then
      Assign $D(v,i) = D(u,i - 1) + w(u,v)$
    else
      Assign $D(v,i) = D(v,i - 1)$
  
foreach $(u,v) \in E$ do
  if $D(v,i - 1) < D(u,i - 1) + w(u,v)$ then

return $D$’s last row