Given an \( n \)-vertex directed weighted graph, find a shortest path from vertex \( i \) to vertex \( j \) for each of the \( n^2 \) vertex pairs \((i,j)\).

**Dijkstra’s Single Source Algorithm**
- Use Dijkstra’s algorithm \( n \) times, once with each of the \( n \) vertices as the source vertex.

**Dynamic Programming Solution**
- Time complexity is \( \Theta(n^3) \) time.
- Works so long as there is no cycle whose length is \( < 0 \).
- When there is a cycle whose length is \( < 0 \), some shortest paths aren’t finite.
  - If vertex 1 is on a cycle whose length is -2, each time you go around this cycle once you get a 1 to 1 path that is 2 units shorter than the previous one.
  - Simpler to code, smaller overheads.
  - Known as Floyd’s shortest paths algorithm.

**Performance**
- Time complexity is \( O(n^3) \) time.
- Works only when no edge has a cost \( < 0 \).

**Decision Sequence**
- First decide the highest intermediate vertex (i.e., largest vertex number) on the shortest path from \( i \) to \( j \).
- If the shortest path is \( i, 2, 6, 3, 8, 5, 7, j \) the first decision is that vertex 8 is an intermediate vertex on the shortest path and no intermediate vertex is larger than 8.
- Then decide the highest intermediate vertex on the path from \( i \) to 8, and so on.

**Problem State**
- \((i,j,k)\) denotes the problem of finding the shortest path from vertex \( i \) to vertex \( j \) that has no intermediate vertex larger than \( k \).
- \((i,j,n)\) denotes the problem of finding the shortest path from vertex \( i \) to vertex \( j \) (with no restrictions on intermediate vertices).
Cost Function

- Let $c(i,j,k)$ be the length of a shortest path from vertex $i$ to vertex $j$ that has no intermediate vertex larger than $k$.

$c(i,j,n)$
- $c(i,j,n)$ is the length of a shortest path from vertex $i$ to vertex $j$ that has no intermediate vertex larger than $n$.
- No vertex is larger than $n$.
- Therefore, $c(i,j,n)$ is the length of a shortest path from vertex $i$ to vertex $j$.

Recurrence For $c(i,j,k)$, $k > 0$
- The shortest path from vertex $i$ to vertex $j$ that has no intermediate vertex larger than $k$ may or may not go through vertex $k$.
- If this shortest path does not go through vertex $k$, the largest permissible intermediate vertex on $i$ to $k$ and $k$ to $j$ paths is $k-1$. So the path length is $c(i,j,k-1)$.

Recurrence For $c(i,j,k)$, $k > 0$
- Shortest path goes through vertex $k$.
- We may assume that vertex $k$ is not repeated because no cycle has negative length.
- Largest permissible intermediate vertex on $i$ to $k$ and $k$ to $j$ paths is $k-1$.

Recurrence For $c(i,j,k)$, $k > 0$
- $i$ to $k$ path must be a shortest $i$ to $k$ path that goes through no vertex larger than $k-1$.
- If not, replace current $i$ to $k$ path with a shorter $i$ to $k$ path to get an even shorter $i$ to $j$ path.
Recurrence For $c(i,j,k)$, $k > 0$

- Similarly, $k$ to $j$ path must be a shortest $k$ to $j$ path that goes through no vertex larger than $k-1$.
- Therefore, length of $i$ to $k$ path is $c(i,k,k-1)$, and length of $k$ to $j$ path is $c(k,j,k-1)$.
- So, $c(i,j,k) = c(i,k,k-1) + c(k,j,k-1)$.

Floyd’s Shortest Paths Algorithm

```java
for (int k = 1; k <= n; k++)
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      c(i,j,k) = min{c(i,j,k-1), c(i,k,k-1) + c(k,j,k-1)};
```

- Time complexity is $O(n^3)$.
- More precisely Theta($n^3$).
- Theta($n^3$) space is needed for $c(*,*,*)$.

Recurrence For $c(i,j,k)$, $k > 0$

- Combining the two equations for $c(i,j,k)$, we get $c(i,j,k) = min\{c(i,j,k-1), c(i,k,k-1) + c(k,j,k-1)\}$.
- We may compute the $c(i,j,k)$s in the order $k = 1, 2, 3, \ldots, n$.

Floyd’s Shortest Paths Algorithm

```java
for (int k = 1; k <= n; k++)
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      c(i,j,k) = c(i,j,k-1) + c(k,j,k-1);
```

- $c(i,j,k) = min\{c(i,j,k-1), c(i,k,k-1) + c(k,j,k-1)\}$
- When neither $i$ nor $j$ equals $k$, $c(i,j,k-1)$ is used only in the computation of $c(i,j,k)$.

Space Reduction

- So $c(i,j,k)$ can overwrite $c(i,j,k-1)$.

Space Reduction

- $c(i,j,k) = min\{c(i,j,k-1), c(i,k,k-1) + c(k,j,k-1)\}$
- When $i$ equals $k$, $c(i,j,k-1)$ equals $c(i,j,k)$.
- $c(k,j,k) = min\{c(k,j,k-1), c(k,k,k-1) + c(k,j,k-1)\}$
  \[= min\{c(k,j,k-1), 0 + c(k,j,k-1)\}\]
  \[= c(k,j,k-1)\]
- So, when $i$ equals $k$, $c(i,j,k)$ can overwrite $c(i,j,k-1)$.
- Similarly when $j$ equals $k$, $c(i,j,k)$ can overwrite $c(i,j,k-1)$.
- So, in all cases $c(i,j,k)$ can overwrite $c(i,j,k-1)$.

Floyd’s Shortest Paths Algorithm

```java
for (int k = 1; k <= n; k++)
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      c(i,j,k) = c(i,j,k-1) + c(k,j,k-1);
```

- Initially, $c(i,j) = c(i,j,0)$.
- Upon termination, $c(i,j) = c(i,j,n)$.
- Time complexity is Theta($n^3$).
- Theta($n^2$) space is needed for $c(*,*)$. 
Building The Shortest Paths
• Let $kay(i,j)$ be the largest vertex on the shortest path from $i$ to $j$.
• Initially, $kay(i,j) = 0$ (shortest path has no intermediate vertex).

```c
for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (c(i,j) > c(i,k) + c(k,j))
                {kay(i,j) = k; c(i,j) = c(i,k) + c(k,j);}
```

Example

Initial Cost Matrix $c(*,*) = c(*,*,0)$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>13</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0</td>
<td>15</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>7</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>5</td>
<td>20</td>
<td>0</td>
<td>9</td>
<td>12</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>9</td>
<td>11</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>4</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td>12</td>
<td>6</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>11</td>
<td>10</td>
<td>6</td>
<td>15</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Final Cost Matrix $c(*,*) = c(*,*,n)$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

kay Matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Shortest Path

The path is $1 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 7$.

Build A Shortest Path

The path is 1425867.

• $kay(1,7) = 8$
  $1 \rightarrow 8 \rightarrow 7$
  • $kay(1,8) = 5$
  $1 \rightarrow 5 \rightarrow 8 \rightarrow 7$
  • $kay(1,5) = 4$
  $1 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 7$
Build A Shortest Path

<table>
<thead>
<tr>
<th>Node</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>404885</td>
</tr>
<tr>
<td>1</td>
<td>8050885</td>
</tr>
<tr>
<td>2</td>
<td>70050065</td>
</tr>
<tr>
<td>3</td>
<td>80802885</td>
</tr>
<tr>
<td>4</td>
<td>84800880</td>
</tr>
<tr>
<td>5</td>
<td>77777007</td>
</tr>
<tr>
<td>6</td>
<td>04114800</td>
</tr>
<tr>
<td>7</td>
<td>77777060</td>
</tr>
</tbody>
</table>

- The path is **1 4 2 5 8 6 7**.
- \(k_{ay}(1,4) = 0\)
- \(k_{ay}(4,5) = 2\)
- \(k_{ay}(4,2) = 0\)

Output A Shortest Path

```java
public static void outputPath(int i, int j)
{
    // does not output first vertex (i) on path
    if (i == j)
        return;
    if (kay[i][j] == 0)
        // no intermediate vertices on path
        System.out.print(j + " ");
    else
    { // kay[i][j] is an intermediate vertex on the path
        outputPath(i, kay[i][j]);
        outputPath(kay[i][j], j);
    }
}
```

Time Complexity Of `outputPath`

\(O(\text{number of vertices on shortest path})\)