**All-Pairs Shortest Paths**

• 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.

**Performance**

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

**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.
**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 $k$.

• $c(i,j,n)$ 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 $j$. 
\( c(i,j,0) \)
- \( c(i,j,0) \) is the length of a shortest path from vertex \( i \) to vertex \( j \) that has no intermediate vertex larger than 0.
- Every vertex is larger than 0.
- Therefore, \( c(i,j,0) \) is the length of a single-edge 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 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 \( k-1 \).
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, $c(i,j,k) = c(i,k,k-1) + c(k,j,k-1)$.

Combining the two equations for $c(i,j,k)$, we get

$$c(i,j,k) = \min\{c(i,j,k-1), c(i,k,k-1) + c(k,j,k-1)\}.$$ 

When neither $i$ nor $j$ equals $k$, $c(i,j,k-1)$ is used only in the computation of $c(i,j,k)$. So $c(i,j,k)$ 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) = 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 $c(*,*,*)$.

**Space Reduction**

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

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

- 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 \( k(i,j) \) be the largest vertex on the shortest path from \( i \) to \( j \).
- Initially, \( k(i,j) = 0 \) (shortest path has no intermediate vertex).

```
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

```
      2
     / \\     \\
 5 ---- 9 ---- 1
    |     |     |
    7 --- 4 --- 1
```

Initial Cost Matrix

\[ c(*,*) = c(*,0) \]

```python
5  4  1
- -  2
- -  4
```
Final Cost Matrix $c(*,*) = c(*,*;n)$

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

kay Matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>4</th>
<th>0</th>
<th>4</th>
<th>8</th>
<th>8</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</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>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Shortest Path

Shortest path from 1 to 7.
Path length is 14.

Build A Shortest Path

04004885
80850885
70050065
80802885
84800880
77777007
04114800
77777060

- The path is 1 4 2 5 8 6 7.
- $kay(1,7) = 8$
- $kay(1,8) = 5$
- $kay(1,5) = 4$
Build A Shortest Path

<table>
<thead>
<tr>
<th></th>
<th>0 4 0 4 8 8 5</th>
<th>8 0 8 5 0 8 8 5</th>
<th>7 0 0 5 0 0 6 5</th>
<th>8 0 8 0 2 8 8 5</th>
<th>8 4 8 0 0 8 8 0</th>
<th>7 7 7 7 7 0 0 7</th>
<th>0 4 1 1 4 8 0 0</th>
<th>7 7 7 7 7 0 6 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 4 5 8 6 7</td>
<td>14 5 8 7</td>
<td>kay(1,4) = 0</td>
<td>kay(4,5) = 2</td>
<td>kay(4,2) = 0</td>
<td>kay(2,5) = 0</td>
<td>kay(5,8) = 0</td>
<td>kay(8,7) = 6</td>
</tr>
<tr>
<td></td>
<td>1 4 5 8 7</td>
<td>14 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td></td>
</tr>
</tbody>
</table>

Build A Shortest Path

<table>
<thead>
<tr>
<th></th>
<th>0 4 0 4 8 8 5</th>
<th>8 0 8 5 0 8 8 5</th>
<th>7 0 0 5 0 0 6 5</th>
<th>8 0 8 0 2 8 8 5</th>
<th>8 4 8 0 0 8 8 0</th>
<th>7 7 7 7 7 0 0 7</th>
<th>0 4 1 1 4 8 0 0</th>
<th>7 7 7 7 7 0 6 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 4 5 8 6 7</td>
<td>14 5 8 7</td>
<td>kay(1,4) = 0</td>
<td>kay(4,5) = 2</td>
<td>kay(4,2) = 0</td>
<td>kay(2,5) = 0</td>
<td>kay(5,8) = 0</td>
<td>kay(8,7) = 6</td>
</tr>
<tr>
<td></td>
<td>1 4 5 8 7</td>
<td>14 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td>1 4 2 5 8 7</td>
<td></td>
</tr>
</tbody>
</table>

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 if (kay[i][j] is an intermediate vertex on the path)
        outputPath(i, kay[i][j]);
        outputPath(kay[i][j], j);
}
```
Time Complexity Of `outputPath`:

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