Lab 19
Shortest Path

- Giving an matrix of \( m \) by \( n \) integers, compute a path of minimal weight from left to right across the matrix.
- The matrix wraps around as to represent a horizontal cylinder.
- The traversable path is represented by as follows...
Traversal
Wrap Around Traversal

<table>
<thead>
<tr>
<th>8</th>
<th>4</th>
<th>1</th>
<th>2</th>
<th>8</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
- We want to determine what is the shortest path from one side to the other.
- We need to calculate the shortest path starting from any of the given row, then the shortest of those path is the absolute shortest possible path through the matrix.

//rows and cols are predefine size of matrix  
//length is an array with size of rows  
//call the cost function to calculate the length on all rows  
for ( int r = 0; r < rows; r++)
    length[r] = cost(r, cols – 1);
//minimum of all lengths is the shortest path  
int min = length[0];
for ( int r = 1; r < rows; r++)
    if ( min > length[r] )
        min = length[r];
Cost Matrix

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>4</th>
<th>1</th>
<th>2</th>
<th>8</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>7</th>
<th>5</th>
<th>7</th>
<th>15</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>12</td>
<td>7</td>
<td>14</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>7</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>9</td>
<td>13</td>
<td>13</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Cost Function

We determine the shortest path from one side to the next column, then gradually builds up from column to column.

To solve the cost for the entire matrix, solve a smaller shortest problem.
- Base Case
- Iterative Step
/pseudo code
int weight[ rows ][ cols ];
int cost( int r, int c)
{
    //at the first column of the matrix
    //the cost is the weight of block
    if ( c == 0 )
        return weight[ r ][ 0 ];
Cost – Iterative Step

- For all other columns, need to consider the weight of the current box, plus the minimum weight of ... - Upper left, direct left and lower left

//pseudo code
int left = cost ( r, c - 1);
int uleft = cost ( ( r - 1 + rows ) % rows, c - 1);
int dleft = cost ( ( r + 1 ) % rows, c - 1);
int min;
//find min
return min + weight[ r ][ c ];
}
Optimization

However, there are lots of repeated calculations. Cost through block(2, 2) is calculated three times by cost through block(3, 1), (3, 2), (3, 3), not to mention each block that goes through (3, 1), (3, 2) and (3, 3) Build a memoize 2-D array to eliminate this!

//pseudo code
int memo[rows][cols];
//if cost value is already in memo array,
return memo[ r ][ c ];
//if not, do the regular computation and store it to memo before return the result.
return memo[ r ][ c ] = min + weight[ r ][ c ];