Problem 1 Do the following tracing exercises in the "A4_P1.xlxs" file. Each part below has its own tab.

1a) (10 pts) Trace through the dynamic programming algorithm to compute number of ways to mak change for 30 cents using coins of denomination 1c, 2c, 4c, 7c, and 9c.

1b) (10 pts) You are given D4 and P4 while in the middle of running Floyd's algorithm. Fill in the highlighted boxes with the correct values for D5 and P5.

1c) (10 pts) Trace through the printPath algorithm to find the shortest paths requested.

Problem 2 (20 pts) First, copy the body of your recursive `numWays` method from Assignment 2 into the method `numWaysREC`. If you did not have a base case for n = 1 in Assignment 2, then add one here. Then use Dynamic Programming to write the `numWaysDP` method, which determines the number of ways to make change for *n* cents, using coins of denominations $1 \leq D[1] < D[2] < D[3] < \cdots < D[k]$. Your method is ***not*** allowed to make ***any*** method calls (so ***no recursion***), otherwise 0 points.

```
long numWaysDP(int n, int k, int [] D)
```

Problem 3 (15 pts) Implement Floyd's alorithm:

```
int Floyd(int n, double [][] W, double [][] D, int [][] P)
```

All three 2-d arrays are indexed from 1 to *n* (they are actually indexed from 0 to *n*, but you will ignore all of row 0 and all of column 0). Array *W* contains the weights of the graph (on *n* vertices). Once Floyd's algorithm returns, $D[i][j]$ contains the cost of a shortest path from *i* to *j*, and $P[i][j]$ contains the index of a vertex along the optimal path. You may not alter array *W*.

The return value of the method is the number of times the array *D* is updated, ***not including*** the copying of values from *W* into *D*.

Extra Credit (up to **+3** points) Change the last (4 x 4) input W in an attempt to maximize the number of updates. +3 points for whoever gets the most, +2 for second, +1 for 3rd. Ties will split the points.

Problem 4 (15 pts) Write the printPath method for Floyd's algorithm:

```
void printPathWithEndpoints(int s, int d, int [][] P)
```

This prints the the vertices along the path (including the endpoints *s* and *d*) of the least-cost path from vertex *s* to vertex *d*.

Problem 5 (20 pts) Write the method below, which takes only *n* and the *P* matrix (passed back from Floyd's algorithm) and returns the largest number of edges in any single optimal cost path.

```
int maxEdgesInOneOptPath(int n, int [][] P)
```

Hint: It will help to first write a method to find the number of edges in the optimal path from *s* to *d* (for any vertices *s* and *d*).