Question 1

Run the algorithm as given in the lecture slides. If you need additional help, please come to office hours or email the TA's.

Question 2

(a) The graph on nodes $v_1, \ldots, v_5$ with edges $(v_1, v_2), (v_1, v_3), (v_2, v_5), (v_3, v_4)$ and (v_4, v_5) is such an example. The algorithm will return 2 corresponding to the path of edges (v_1, v_2) and (v_2, v_5), while the optimum is 3 using the path $(v_1, v_3), (v_3, v_4)$ and (v_4, v_5).

(b) The idea is to use dynamic programming. The simplest version to think of uses the subproblems $OPT[i]$ for the length of the longest path from v_1 to v_i. One point to be careful of is that not all nodes v_i necessarily have a path from v_1 to v_i. We will use the value "$-\infty$" for the $OPT[i]$ value in this case. We use $OPT(1) = 0$ as the longest path from v_1 to v_1 has 0 edges.

```
Long-path(n)
  Array $M[1 \ldots n]$
  $M[1] = 0$
  For $i = 2, \ldots, n$
     $M = -\infty$
     For all edges $(j, i)$ then
        if $M[j] \neq -\infty$
             if $M < M[j] + 1$ then
                 $M = M[j] + 1$
             endif
        endif
     endfor
     $M[i] = M$
  endfor
  Return $M[n]$ as the length of the longest path.
```

The running time is $O(n^2)$ if you assume that all edges entering a node i can be listed in $O(n)$ time.

Question 3

The key observation to make in this problem is that if the segmentation $y_1 y_2 \dots y_n$ is an optimal one for the string y, then the segmentation $y_1 y_2 \dots y_{n-1}$ would be an optimal segmentation for the prefix of y that excludes y_n (because otherwise we could substitute the optimal solution for the prefix in the original problem and get a better solution).

Given this observation, we design the subproblems as follows. Let $Opt(i)$ be the score of the best segmentation of the prefix consisting of the first i characters of y. We claim that the recurrence

$$OPT(i) = max_{j \leq i}\{Opt(j-1) + quality(j, \dots, i)\}$$

would give us the correct optimal segmentation (where $Quality(\alpha \dots \beta)$ means the quality of the word that is formed by the characters starting from position α and ending in position β). Notice that the desired solution is $Opt(n)$.

We prove the correctness of the above formula by induction on the index i. The base case is trivial, since there is only one word with one letter.

For the inductive step, assume that we know that the Opt function as written above finds the optimum solution for the indices less than i, and we want to show that the value $Opt(i)$ is the optimum cost of any segmentation for the prefix of y up to the i-th character. We consider the last word in the optimal segmentation of this prefix. Let's assume it starts at index $j \leq i$. Then according to our key observation above, the prefix containing only the first $j-1$ characters must also be optimal. But according to our induction hypothesis, $Opt(j)$ will yield us the value of the aforementioned optimal segmentation. Therefore the optimal cost $Opt(i)$ would be equal to $Opt(j)$ plus the cost of the last word.

But notice that our above recurrence exactly does this calculation for each possibility of the last word. Therefore our recurrence will correctly find the cost of the optimal segmentation.

As for the running time, a simple implementation (direct evaluation of the above formula starting at index 1 until n, where n is the number of characters in the input string) will yield a quadratic algorithm.

Question 4

A useful way to analyze large products in cases like this is to take logarithms, which causes them to become sums. Thus, let us build a graph G with a node for each stock, and a directed edge (i, j) for each pair of stocks. We put a cost of $-\log r_{ij}$ on edge (i, j).

Now, a trading cycle C in G is an opportunity cycle if and only if

$$\prod_{(i,j)\in C} r_{ij} > 1,$$

in other words, taking logarithms of both sides, if and only if

$$\sum_{(i,j)\in C} \log r_{ij} > 0,$$

or

$$\sum_{(i,j)\in C} -\log r_{ij} < 0.$$

Thus, a trading cycle C in G is an opportunity cycle if and only if it is a negative cycle. Hence we can use our polynomial-time algorithm for negative-cycle detection to determine whether an opportunity cycle exists.

Question 5

First note that it is enough to maximize one's *total* grade over the n courses, since this differs from the average grade by the fixed factor of n. Let the (i, h)-*subproblem* be the problem in which one wants to maximize one's grade on the first i courses, using at most h hours.

Let $A[i, h]$ be the maximum total grade that can be achieved for this subproblem. Then $A[0, h] = 0$ for all h, and $A[i, 0] = \sum_{j=1}^{i} f_j(0)$. Now, in the optimal solution to the (i, h)-subproblem, one spends k hours on course i for some value of $k \in [0, h]$; thus

$$A[i, h] = \max_{0 \leq k \leq h} f_i(k) + A[i - 1, h - k].$$

We also record the value of k that produces this maximum. Finally, we output $A[n, H]$, and can trace-back through the entries using the recorded values to produce the optimal distribution of time. The total time to fill in each entry $A[i, h]$ is $O(H)$, and there are nH entries, for a total time of $O(nH^2)$.

Question 6

Let c_e denote the cost of the edge e and we will overload the notation and write c_{st} to denote the cost of the edge between the nodes s and t.

This problem is by its nature quite similar to the shortest path problem. Let us consider a two-parameter function $Opt(i, s)$ denoting the optimal cost of shortest path to s using *exactly* i edges, and let $N(i, s)$ denote the number of such paths.

We start by setting $Opt(i, v) = 0$ and $Opt(i, v') = \infty$ for all $v' \neq v$. Also set $N(i, v) = 1$ and $N(i, v') = 0$ for all $v' \neq v$. Intuitively this means that the source v is reachable with cost 0 and there is currently one path to achieve this.

Then we compute the following recurrence:

$$Opt(i, s) = min_{t,(t,s)\in E}\{Opt(i-1, t) + c_{ts}\}. \tag{1}$$

The above recurrence means that in order to travel to node s using exactly i edges, we must travel a predecessor node t using exactly $i - 1$ edges and then take the edge connecting t to s. Once of course the optimal cost value has been computed, the number of paths that achieve this optimum would be computed by the following recurrence:

$$N(i, s) = \sum_{t,(t,s)\in E \text{ and } Opt(i,s)=Opt(i-1,t)+c_{ts}} N(i-1, t). \tag{2}$$

In other words, we look at all the predecessors from which the optimal cost path may be achieved and add all the counters.

The above recurrences can be calculated by a double loop, where the outside loops over i and the inside loops over all the possible nodes s. Once the recurrences have been solved, our target optimal path to w is obtained by taking the minimum of all the paths of different lengths to w - that is:

$$Opt(w) = min_i\{Opt(i, w)\}. \tag{3}$$

And the number of such paths can be computed by adding up the counters of all the paths which achieve the minimal cost.

$$N(w) = \sum_{i,Opt(i,w)=Opt(w)} N(i, w). \tag{4}$$