Problem 1 (20 Points). You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \cdots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the *penalty* for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty — that is, the sum, over all travel days, of the daily penalties. Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

Solution: Let $T[i]$ be the cost of the optimal trip visiting only hotels $1, ..., i$, ending at hotel i. The optimal trip for the first i hotels consists of the optimal trip for the first j hotels for some $j < i$, plus an extra day covering all the hotels from $j + 1$ to i. The distance traveled in the last day is $a_i - a_j$, so the cost for the last day is $(200 - (a_i - a_j))^2$. To get the actual optimal cost, just take the minimum. Thus,

$$T[i] = \min_{j<i} \left(T[j] + (200 - a_i - a_j)^2 \right)$$

We can compute the $T[i]$ values in order of increasing i. $T[n]$ gives us the optimal cost.

To actually get the optimal sequence, keep track of which j caused the minimum, and call this $M[i]$. Then, the optimal trip will pass hotels $M[n] + 1$ through n on the last day. Let $k_1 = M[n]$. The optimal will visit $M[k_1] + 1$ through k_1 on the second to last day. Let $k_2 = M[k_1]$. The optimal will visit $M[k_2] + 1$ through k_2 on the third to last day, etc.

Problem 2 (20 Points). You are given a string of characters $(s_1, s_2, \cdots, s_n)$, which you believe to be a corrupted text document in which all punctuation has vanished (so that is lookes something like "itwasthebestoftimes..."). You wish to reconstruct the document using a dictionary, which is available in the form of a boolean function `dict`$(\cdot)$: for any string w,

$$\texttt{dict}(w) = \begin{cases} \texttt{true} & \text{if } w \text{ is a valid word} \\ \texttt{false} & \text{otherwise} \end{cases}$$

Give a dynamic programming algorithm that determines whether the string $(s_1, s_2, \cdots, s_n)$ can be reconstituted as a sequence of valid words. In the event that the string is valid, make your algorithm output the corresponding sequence of words. The running time should be at most $O(n^2)$, assuming calls to `dict` take constant time.

Solution: Modify `dict` so that instead of outputting `true`, it outputs 0, and instead of outputting `false`, it outputs ∞. Let $D[i] = 0$ if the first i characters can be parsed into words, ∞ otherwise. Any parsing of the dictionary into words will consist of parsing the first j characters

$(j < i)$ into words, plus a word consisting of characters $j+1$ through i. Let $S_{[a,b]}$ be the sequence $s_a ... s_b$. Then

$$D[i] = min_{j<i}\Big(D[i] + \texttt{dict}(S_{[j+1,i]})\Big)$$

We can compute D in order of increasing i. To check that the whole string is valid, check that $D[n] = 0$. To actually get the sequence, also keep track of $P[i]$, which is the value j that obtained the minimum. Then the last word will be $S_{[P[n]+1,n]}$, and the rest of the words will be the sequence of words for the first $P[n]$ characters.

Problem 3 (20 Points). We saw a greedy algorithm for the making change problem: Given coin denominations $d_1, ..., d_n$, and a target amount w, find a collections of coins whose value is w, using the fewest number of coins. That is, we wish to find a sequence of integers $x_1, ..., x_n$, such that

$$\sum_{i=1}^{n} x_i d_i = w$$

that minimizes

$$\sum_{i=1}^{n} x_i \ .$$

However, our greedy algorithm only worked for certain sets of denominations. Give an efficient dynamic programming algorithm to solve this problem.

Solution: Let $C[t]$ be the number of coins needed to make change for t cents. To make change for $t > 1$ cents, there is at least one coin with denomination d_i, and the optimal change consists of making optimal change for $t - d_i$, and then adding one coin of value d_i. Thus, we get the following recurrence:

$$C[t] = 1 + \min_i(C[t - d_i])$$

Where $C[0] = 0$. To get the actual coins, keep track of the d_i that caused the minimum. Output this coin. Then output the coins used to make change for $t - d_i$ cents. The solution corresponds to $C[w]$

Problem 4 (20 Points). In class, we say how to compute the edit distance where edits consist of insertions, deletions, and replacements. Suppose now we wish to also allow two adjacent characters to be swapped. For example, the following alignment for "COLOR" and "COOL" requires 2 edits:

```
C O (L O) R
C O (O L) -
```

Devise an efficient algorithm to compute the edit distance of two sequences, where insertions, deletions, replacements, and swaps each incur a cost of 1.

Solution: All we need to do is modify the update to allow for swaps. Let $\texttt{canSwap}(i, j)$ be 1 if we can swap the last two characters of $S_{[1,i]}$ to get the last two characters of $T_{[1,j]}$. That is, $\texttt{canSwap}(i, j) = 1$ if $S_i = T_{j-1}$ and $S_{i-1} = T_j$. Otherwise, $\texttt{canSwap}(i, j) = \infty$. The updates then look like:

$$E[i, j] = \min\left(\texttt{diff}(i, j) + E[i-1, j-1], 1 + E[i, j-1], 1 + E[i-1, j], \texttt{canSwap}(i, j) + E[i-2, j-2]\right)$$

Problem 5 (20 Points). Shortest path algorithms can be applied in currency trading. Suppose we have n different currencies $1, ..., n$. For any two currencies, there is an exchange rate $r_{i,j}$; this means that you can purchase $r_{i,j}$ units of currency j in exchange for one unit of currency i. These exchange rates satisfy the condition that $r_{i,j} \cdot r_{j,i} < 1$, so that if you start with a unit of currency i, change it to currency j, and then convert back to i, you end up with less that one unit of currency i. The difference is the cost of the transaction.

(a) Give an efficient algorithm for the following problem: Given a set of exchange rates $r_{i,j}$, and two currencies s and t, find the most advantageous sequence of currency exchanges for converting currency s into currency t. Toward this goal, you should represent the currencies and rates by a graph whose edge lengths are real numbers.

The exchange rates are updated frequently, reflecting the demand and supply of the various currencies. Occasionally the exchange rates satisfy the following property: there is a sequence of currencies $i_1, ..., i_k$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdots r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$. This means that by starting with a unit of currency i_1, and then successively converting it to currencies $i_2, i_3, ..., i_k$ and finally back to i_1, you would end up with more than one unit of currency i_1. Such anomalies last only a fraction of a minute on the currency exchange, but they provide an opportunity for risk-free profits.

(b) Give an efficient algorithm for detecting the presence of such an anomaly. Use the graph representation you found above.

Solution:

(a) Let the weight of the edge (i, j) be $-\log r_{i,j}$. Then, the cost of a path is the sum of the weights of these edges, which is the same as the negative log of the product of the $r_{i,j}$. Thus, to finish with the most money, we need to maximize the product of the $r_{i,j}$ along the path, which corresponds to finding the shortest path from s to t. We can accomplish this using the single-source shortest paths algorithm in the presence on negative edge weights.

(b) Such an anomaly corresponds to a negative cycle in the graph. Thus, run the negative-cycle detection algorithm from class.

Total points: 100