Problem 8-1 (Who Needs a Nickel? Greedy People!) 14 Points

Assume you have $q$ quarters, $d$ dimes, $n$ nickels and $p$ pennies. You just bought an amazing textbook on basic algorithms and paid $T > 0$ cents for it. You wish to provide the exact change of $T$ cents using the minimum number of coins $c$. For example, if $q = d = n = p = 2$ and $T = 27$, the optimal solution is $25 + 1 + 1 = 27$, using only $c = 3$ coins, since the other solution $10 + 10 + 5 + 1 + 1$ uses 5 coins. On the other hand, if $T = 23$, there is no solution at all, since you only have 2 pennies.

(a) (10 points) Devise an $O(T \cdot (q + d + n + p))$ dynamic-programming algorithm that computes the minimum number of coins $c$. If there is no solution, your algorithm should return $c = \infty$.

For full credit, use only $O(T)$ space. (For partial credit, your time may be much worse, e.g., $O(T \cdot q \cdot d \cdot n \cdot p)$. Note that you need not reconstruct the actual change!

(b) (4 points) Consider the following greedy algorithm: “Find the largest coin (among those you still have) whose value $v$ is less than $T$. Add this coin to your solution and recurse on $(T - v)$.” Give an example (i.e., a choice of $T, q, d, n, p$) where the greedy algorithm above fails to find the optimum solution. For extra credit, the greedy algorithm in your example should return $c < \infty$. (Hint: See the title of the problem!)

Problem 8-2 (Let’s paint a fence!) 12 (+3) Points

We just built a new fence which is made up of many boards, and our task is to paint the fence efficiently. To paint the fence, you have to paint $N$ boards that make up the fence. The lengths of the $N$ boards are $\{L_1, L_2, \ldots, L_N\}$. You have hired $K$ painters and you know that each painter takes 1 hour to paint 1 unit of board. If 3, 4, 5 are the boards painter $i$ paints, the total time he spends is $t_i = L_3 + L_4 + L_5$. Our goal is to assign each painter to boards so that the total painting time is minimized. Since the painters can work in parallel, the painting time is minimized when $\max(t_1, \ldots, t_K)$ is minimal. The painting task must be accomplished under the following constraints:

1. Two painters cannot share a board to paint. That is, a board cannot be painted partially by one painter, and partially by another. All $L_i$ units of board $i$ must be painted by one painter.

2. Any painter will only paint contiguous boards. For example, a configuration where painter 1 paints boards 1 and 3 but not 2 is not a valid solution.

You are given as input the following: $K$, the number of painters, and $L$, a list which will represent the length of each board, where $L_i$ is the length of the $i^{th}$ board. In the following problems, denote by $T[i, j]$ the minimum time to paint the first $i$ boards with $j$ painters. We will, successively in the following subtasks, come up with a procedure to minimize painting time.
(a) (6 points) Write a recurrence for $T[i, j]$ in terms of $T[* , j - 1]$.

(b) Note that the time required to fill table $T$ is $O(NK \cdot C)$, where $C$ is the maximum time needed to compute any of the entries $T[i, j]$. Provide the fastest algorithm you can conceive. You will obtain

- 2 points if your algorithm satisfies $C = O(N^2)$,
- an additional 4 points if $C = O(N)$,
- and 3 points of extra credit if you manage to get it down to $C = O(\log N)$.

Problem 8-3 (Word chaining) 12 Points

You are given a dictionary of $n$ words. Consider performing the following operation: take a word from the dictionary, remove one character from it, and check if the resulting word belongs to the dictionary. If yes, the two words are defined to form a word chain of length 2. More generally, words $u_1, u_2, \ldots, u_m$ are defined to be a word chain of length $m$ if each $u_i$ is in the dictionary, and removing one character from $u_{i+1}$ results in $u_i$. Our goal in this problem is to find the length of the longest possible word chain of a given dictionary.

(a) (5 points) Consider sorting the $n$ words in the dictionary by their length. Let $\text{words}$ be the sorted array such that all words of length 1 come first, then words of length 2 and so on. We define the array $C$ as follows: $C[i]$ is the length of the longest word chain ending with $\text{words}[i]$.

Show how to compute each $C[i]$ in time $k^2 n$.

(b) (5 points) Show how to compute each $C[i]$ in $k^2 \log n$. (Hint: Think of sorting each group of words of the same length in some useful order.)

(c) (2 points) What is the running time of the procedure that computes $C$ and uses it to determine the length of the longest word chain?

Problem 8-4 (Palindrome) 15 Points

A palindrome is a sequence that remains the same if written in the reverse order, e.g. DEED, DENNISSINNED, BORROWORROB. We are interested in the length of the longest palindrome subsequence of a string. For example for input string HOMEWORK, the longest palindromic subsequence is OEO, and so the answer is 3.

(a) (10 points) Given input string as an array of characters $A$, give an $O(n^2)$ algorithm to find the length of the longest palindrome subsequence. (Hint: Think of the Longest Common Subsequence done in class.)

(b) (5 points) The problem in part (a) above can be solved directly by reversing the given string and then finding the length of the longest common subsequence (using the algorithm studied in the lecture) of the string and its reverse. Formally, it is known that the length of the longest palindrome subsequence is equal to the length of the longest common subsequence.
of the string and its reverse. However, in general, not all longest common subsequences of a string and its reverse are necessarily palindromes. Give an example of a string where one of the longest common subsequences of the string and its reverse is not a palindrome.