Solution to Problem 16-1

Before we go into the various parts of this problem, let us first prove once and for all that the coin-changing problem has optimal substructure.

Suppose we have an optimal solution for a problem of making change for n cents, and we know that this optimal solution uses a coin whose value is c cents; let this optimal solution use k coins. We claim that this optimal solution for the problem of n cents must contain within it an optimal solution for the problem of $n - c$ cents. We use the usual cut-and-paste argument. Clearly, there are $k - 1$ coins in the solution to the $n - c$ cents problem used within our optimal solution to the n cents problem. If we had a solution to the $n - c$ cents problem that used fewer than $k - 1$ coins, then we could use this solution to produce a solution to the n cents problem that uses fewer than k coins, which contradicts the optimality of our solution.

a. A greedy algorithm to make change using quarters, dimes, nickels, and pennies works as follows:

- Give $q = \lfloor n/25 \rfloor$ quarters. That leaves $n_q = n \bmod 25$ cents to make change.
- Then give $d = \lfloor n_q/10 \rfloor$ dimes. That leaves $n_d = n_q \bmod 10$ cents to make change.
- Then give $k = \lfloor n_d/5 \rfloor$ nickels. That leaves $n_k = n_d \bmod 5$ cents to make change.
- Finally, give $p = n_k$ pennies.

An equivalent formulation is the following. The problem we wish to solve is making change for n cents. If $n = 0$, the optimal solution is to give no coins. If $n > 0$, determine the largest coin whose value is less than or equal to n. Let this coin have value c. Give one such coin, and then recursively solve the subproblem of making change for $n - c$ cents.

To prove that this algorithm yields an optimal solution, we first need to show that the greedy-choice property holds, that is, that some optimal solution to making change for n cents includes one coin of value c, where c is the largest coin value such that $c \leq n$. Consider some optimal solution. If this optimal solution includes a coin of value c, then we are done. Otherwise, this optimal solution does not include a coin of value c. We have four cases to consider:

- If $1 \leq n < 5$, then $c = 1$. A solution may consist only of pennies, and so it must contain the greedy choice.
- If $5 \leq n < 10$, then $c = 5$. By supposition, this optimal solution does not contain a nickel, and so it consists of only pennies. Replace five pennies by one nickel to give a solution with four fewer coins.

- If $10 \le n < 25$, then $c = 10$. By supposition, this optimal solution does not contain a dime, and so it contains only nickels and pennies. Some subset of the nickels and pennies in this solution adds up to 10 cents, and so we can replace these nickels and pennies by a dime to give a solution with (between 1 and 9) fewer coins.
- If $25 \le n$, then $c = 25$. By supposition, this optimal solution does not contain a quarter, and so it contains only dimes, nickels, and pennies. If it contains three dimes, we can replace these three dimes by a quarter and a nickel, giving a solution with one fewer coin. If it contains at most two dimes, then some subset of the dimes, nickels, and pennies adds up to 25 cents, and so we can replace these coins by one quarter to give a solution with fewer coins.

Thus, we have shown that there is always an optimal solution that includes the greedy choice, and that we can combine the greedy choice with an optimal solution to the remaining subproblem to produce an optimal solution to our original problem. Therefore, the greedy algorithm produces an optimal solution.

For the algorithm that chooses one coin at a time and then recurses on subproblems, the running time is $\Theta(k)$, where k is the number of coins used in an optimal solution. Since $k \le n$, the running time is $O(n)$. For our first description of the algorithm, we perform a constant number of calculations (since there are only 4 coin types), and the running time is $O(1)$.

b. When the coin denominations are $c^0, c^1, \ldots, c^k$, the greedy algorithm to make change for n cents works by finding the denomination c^j such that $j = \max\{0 \le i \le k : c^i \le n\}$, giving one coin of denomination c^j, and recursing on the subproblem of making change for $n - c^j$ cents. (An equivalent, but more efficient, algorithm is to give $\lfloor n/c^k \rfloor$ coins of denomination c^k and $\lfloor (n \bmod c^{i+1})/c^i \rfloor$ coins of denomination c^i for $i = 0, 1, \ldots, k-1$.)

To show that the greedy algorithm produces an optimal solution, we start by proving the following lemma:

Lemma

For $i = 0, 1, \ldots, k$, let a_i be the number of coins of denomination c^i used in an optimal solution to the problem of making change for n cents. Then for $i = 0, 1, \ldots, k-1$, we have $a_i < c$.

Proof If $a_i \ge c$ for some $0 \le i < k$, then we can improve the solution by using one more coin of denomination c^{i+1} and c fewer coins of denomination c^i. The amount for which we make change remains the same, but we use $c - 1 > 0$ fewer coins. ■ (lemma)

To show that the greedy solution is optimal, we show that any non-greedy solution is not optimal. As above, let $j = \max\{0 \le i \le k : c^i \le n\}$, so that the greedy solution uses at least one coin of denomination c^j. Consider a non-greedy solution, which must use no coins of denomination c^j or higher. Let the non-greedy solution use a_i coins of denomination c^i, for $i = 0, 1, \ldots, j-1$; thus we have $\sum_{i=0}^{j-1} a_i c^i = n$. Since $n \ge c^j$, we have that $\sum_{i=0}^{j-1} a_i c^i \ge c^j$.

Now suppose that the non-greedy solution is optimal. By the above lemma, $a_i \le c - 1$ for $i = 0, 1, \ldots, j - 1$. Thus,

$$\begin{aligned}
\sum_{i=0}^{j-1} a_i c^i &\le \sum_{i=0}^{j-1} (c-1)c^i \\
&= (c-1) \sum_{i=0}^{j-1} c^i \\
&= (c-1) \frac{c^j - 1}{c-1} \\
&= c^j - 1 \\
&< c^j ,
\end{aligned}$$

which contradicts our earlier assertion that $\sum_{i=0}^{j-1} a_i c^i \ge c^j$. We conclude that the non-greedy solution is not optimal.

Since any algorithm that does not produce the greedy solution fails to be optimal, only the greedy algorithm produces the optimal solution.

The problem did not ask for the running time, but for the more efficient greedy-algorithm formulation, it is easy to see that the running time is $O(k)$, since we have to perform at most k each of the division, floor, and mod operations.

c. With actual U.S. coins, we can use coins of denomination 1, 10, and 25. When $n = 30$ cents, the greedy solution gives one quarter and five pennies, for a total of six coins. The non-greedy solution of three dimes is better.

The smallest integer numbers we can use are 1, 3, and 4. When $n = 6$ cents, the greedy solution gives one 4-cent coin and two 1-cent coins, for a total of three coins. The non-greedy solution of two 3-cent coins is better.

Introduction to Algorithms November 14, 2001
Massachusetts Institute of Technology 6.046J/18.410J
Singapore-MIT Alliance SMA5503
Professors Erik Demaine, Lee Wee Sun, and Charles E. Leiserson Handout 28

Scheduling to minimize average completion time

Suppose you are given a set $S = \{a_1, a_2, \ldots, a_n\}$ of tasks, where task a_i requires p_i units of processing time to complete, once it has started. You have one computer on which to run these tasks, and the computer can run only one task at a time. Let c_i be the ***completion time*** of task a_i, that is, the time at which task a_i completes processing. Your goal is to minimize the average completion time, that is, to minimize $(1/n)\sum_{i=1}^{n} c_i$. For example, suppose there are two tasks, a_1 and a_2, with $p_1 = 3$ and $p_2 = 5$, and consider the schedule in which a_2 runs first, followed by a_1. Then $c_2 = 5$, $c_1 = 8$, and the average completion time is $(5 + 8)/2 = 6.5$.

(a) Suppose there are two tasks with $p_1 = 3$ and $p_2 = 5$. Consider (1) the schedule in which task 1 runs first, followed by task 2 and (2) the schedule in which task 2 runs first, followed by task 1. In each case, state the values of c_1 and c_2 and compute the average completion time.

Solution:

For (1) the schedule in which task 1 runs first, followed by task 2:
$c_1 = p_1 = 3$ and $c_2 = p_1 + p_2 = 8$
average completion time $= (c_1 + c_2)/2 = 11/2 = 5.5$
For (2) the scedule in which task 2 runs first, followed by task 1:
$c_2 = p_2 = 5$ and $c_1 = p_2 + p_1 = 8$
average completion time $= (c_1 + c_2)/2 = 13/2 = 6.5$

(b) Give an algorithm that schedules the tasks so as to minimize the average completion time. Each task must run nonpreemptively, that is, once task i is started, it must run continuously for p_i units of time. Prove that your algorithm minimizes the average completion time, and state the running time of your algorithm.

Solution:

Run tasks in shortest processing time order. This can be done by sorting the elements using heap sort or merge sort and then scheduling them in the order of increasing scheduling times. This algorithm takes $O(n \lg n)$.

This algorithm uses a greedy strategy. It is shown to be optimal as follows: $c_{avg} = \frac{c_1+c_2+c_3+\cdots+c_n}{n}$ This cost can also be expressed as $[p_1 + (p_1 + p_2) + (p_1 + p_2 + p_3) + \cdots + (p_1 + p_2 + \cdots + p_n)]\frac{1}{n}$. p_1 is added in the most times, then p_2, etc. As a result, p_1 should have the shortest processing time, then p_2, etc. otherwise, you could cut and paste in a shorter processing time and produce a faster algorithm. As a result, the greedy property holds and our algorithm is correct.

Suppose now that the tasks are not all available at once. That is, each task has a ***release time*** r_i before which it is not available to be processed. Suppose also that we allow ***preemption***, so that a task can be suspended and restarted at a later time. For example, a task i with processing time $p_i = 6$ may start running at time 1 and be preempted at time 4. It can then resume at time 10 but be preempted at time 11 and finally resume at time 13 and complete at time 15. Task i has run for a total of 6 time units, but its running time has been divided into three pieces.

(c) Give an algorithm that schedules the tasks so as to minimize the average completion time in this new scenario. Prove that your algorithm minimizes the average completion time, and state the running time of your algorithm.

Solution:

This problem also exhibits the greedy property which can be exploited by running tasks in shortest remaining processing time order. Use a priority queue which prioritizes based on the task with the shortest time remaining. Each time a new task comes up, insert it into the queue and if it would take less time to do that task then the one you are on, do the shorter task. Each time you finish a task, the next task you should do is the one with the least remaining time until completion. The priority queue can be maintained in $O(n \lg n)$ time.

This algorithm minimizes the average completion time and the proof is similar to part b. If we do not schedule using the greedy algorithm based on remaining processing time, then we will be able to swap two time slots which would then improve the sum of the completion times and thus result in a contradiction. For example, assume you have two tasks at time t, where task i has x processing time remaining and j has y processing time remaining where $x > y$. Assume for the purposes of contradicction that the optimal answer has task i running before task j. If i is done before j then $c_i = t + x$ and $c_j = t + x + y$. The average completion time is $\frac{2t+2x+y}{2}$. However if j were done before i, then $c_i = t + y + x$ and $c_j = t + y$. The average completion time is now $\frac{2t+2y+x}{2}$ which is less the average completion time for the "optimal" solution since $x > y$. As a result, the task with the lowest time remaining should be done first.