# CS:3330 Spring 2017: Solutions to Homework 6

Shreyas Pai Richard Blair

## Problem 1

There is a very natural greedy algorithm for this problem. The idea is to first give the largest amount of quarters possible, then dimes, then nickles, and then pennies. Following pseudocode describes the algorithm. I am using the Python notation in the pseudocode where $a\%b$ gives you the remainder and $a//b$ gives you the quotient when you divide $a$ by $b$ when $a \geq 0, b > 0$ are integers –

**Algorithm 1** ChangeGreedy $(n)$

1: $numQuarters \leftarrow n//25$
2: $n \leftarrow n\%25$
3: $numDimes \leftarrow n//10$
4: $n \leftarrow n\%10$
5: $numNickels \leftarrow n//5$
6: $n \leftarrow n\%5$
7: $numCents \leftarrow n$
8: **return** $(numQuarters, numDimes, numNickels, numCents)$

It is a good exercise to write a more general procedure ChangeGreedy $(n, coins)$ which takes an arbitrary list of coin values *coins* and either returns a list of coin amounts or `failure` if that is not possible. Now we will analyse the above algorithm for correctness.

(a) First we answer the questions. The maximum number of pennies that $O$ can contain is 4 because if $O$ contains more than 4 pennies, we can exchange five of them out for one nickel. This exchange decreases the size of $O$ by 4 and is a valid solution which contradicts optimality of $O$. Using a similar argument, we can say that $O$ contains no more than one nickel (exchange two nickels with one dime) and no more than two dimes (exchange three dimes with a quarter and a nickel).

Also, if $O$ contains two dimes then $O$ cannot contain a nickel. Similarly, if $O$ contains one nickel, it cannot contain more than one dime. Therefore, the maximum value of dimes, nickles, and pennies in $O$ is given by two dimes and four pennies for a total of 24 cents. Now we are ready to prove the claim.

Note that $O$ cannot contain *more* quarters than the greedy solution. This is because the greedy algorithm picks as many quarters as possible that amounts to at most $n$ and if $O$ picks more than that then the change will add up to more than $n$. So let us assume for contradiction that $O$ contains less quarters than the greedy solution. Therefore we need to account for at least 25 cents in $O$ in terms of dimes, nickles, and pennies. But the maximum value we can obtain in $O$ with these coins is 24 according to our previous discussion. Thus, we have a contradicion.

(b) Now we use the fact that $O$ contains as many quarters as the greedy algorithm to prove that $O$ contains as many dimes as the greedy algorithm. Note again that $O$ cannot contain more

dimes than the greedy algorithm as the change would add up to more than $n$. So let us assume for contradiction that $O$ contains less dimes than the greedy solution. Therefore we have to account for at least 10 cents worth of change in $O$ in terms of nickels and pennies. But $O$ cannot contain more than four pennies and one nickel which accounts for only 9 cents worth of change. This is a contradiction.

Again using a similar argument, we can prove that the number of nickels in $O$ is the same as the greedy algorithm. The set $O$ cannot have more nickels than the greedy solution as it will add up to more than $n$ and also cannot have less nickels since we would have to account for at least 5 cents worth of change in terms of pennies in $O$ which is not possible.

(c) The denominations $\{1, 17, 30\}$ and $n = 34$ is one of the many examples where greedy algorithm gives a sub-optimal solution. Greedy solution is four 1's and one 30 for a total of five coins whereas optimal solution is two 17's.

## Problem 2

In this problem we consider the following algorithm. Let $x$ be the class with the earliest start time, and let $y$ be the class with the second earliest start time.

- If $x$ and $y$ are disjoint, choose $x$ and recurse on everything but $x$.
- If $x$ completely contains $y$ , discard $x$ and recurse.
- Otherwise, discard $y$ and recurse

We will prove by induction (on the number of intervals $n$ in the input) that the solution returned by this algorithm is the same as the solution returned by the earliest finish time first (EFF) algorithm.

**Base Case:** $n = 0$ and $n = 1$. The solutions returned by both the algorithms in these two cases are trivially the same.

**Induction Hypothesis:** Assume that both the algorithms have the same solution on all inputs with $j$ intervals where $2 \leq j \leq n$.

**Induction Step:** Let $I$ be the set of input intervals such that $|I| = n+1$. Just like the algorithm, let $x$ be the interval with the earliest start time, and let $y$ be the interval with the second earliest start time. Let $S_x$ and $S_y$ denote the solutions of the EFF algorithm on inputs $I \setminus \{x\}$ and $I \setminus \{y\}$ respectively

- If $x$ and $y$ are disjoint, our algorithm chooses $x$ and recursively finds the solution $S'$ on $I \setminus \{x\}$. We know that $S' = S_x$ by the induction hypothesis. In this case, $x$ is the earliest finishing interval so $x$ will be part of the EFF solution. Moreover, no interval overlaps with $x$ so the solution of EFF algorithm will be exactly $\{x\} \cup S_x$.
- If $x$ completely contains $y$. In this case, we discard $x$ and find a solution $S'$ on $I \setminus \{x\}$ recursively. By induction hypothesis, we know that $S' = S_x$ and because $y$ is an interval that overlaps with $x$ and also finishes earlier than $x$, $x$ cannot be in the solution of the EFF algorithm. Thus, the solution of the EFF algorithm and our algorithm are the same solution $S_x$.

- Otherwise, we discard $y$ and recurse. Note that in this case, $x$ and $y$ overlap with each other and $x$ finishes before $y$. This means that the solution given by the EFF algorithm cannot contain $y$. So the solution of the EFF algorithm is $S_y$ which by the induction hypothesis, is the same as the solution of our algorithm on $I \setminus \{y\}$

Notice that the inductive proof closely follows the structure of our recursive algorithm. This is not a coincidence; recursion and induction are actually the same concept presented in different styles. Many of the greedy algorithms that we have studied can be written recursively and that is the reason why we use inductive arguments in the analysis.

## Problem 3

The proof in for this problem is very simple; we just need to provide a counterexample showing that the greedy algorithm is not optimal.

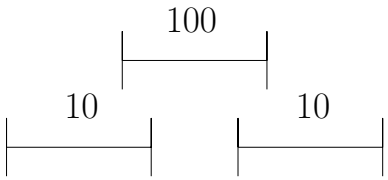


Figure 1: Counterexample showing greedy is not optimal

The greedy algorithm will pick the lower two intervals and give a solution of weight 20 whereas optimal solution has weight 100.

## Problem 4

The proof is by induction. We shall prove that the algorithm gives an optimal solution on a problem with one interval, and then prove, assuming the algorithm gives an optimal solution for all problems with $n$ intervals, that it gives an optimal solution for all problems with $n + 1$ intervals.

As for notation, let $I_1, \ldots, I_k$ denote the intervals. When referring to their endpoints (start and end times), let $I_k = (a_k, b_k)$.

The base case is a problem with a single interval. The algorithm will assign the color 1 to this problem, and then terminate. Exactly one color is used. This is clearly optimal, as any coloring of one interval will use at least one color.

For the inductive case, we assume the algorithm gives an optimal coloring for all problems of $n$ intervals, and we are given a problem with $n + 1$ intervals. Let $\mathcal{O}_n$ denote the solution given by the algorithm for the first $n$ intervals. $\mathcal{O}_n$ uses $k$ colors $\{1 \ldots k\}$ to color the intervals $I_1, \ldots, I_n$.

Now, interval $I_{n+1}$ intersects some number of intervals from the set $I_1 \ldots I_n$. I claim it cannot intersect more than $k$ intervals. Note that, since our intervals are in sorted order by start time, for an interval $I_p$ to intersect $I_q$ with $p < q$, we must have $a_q < b_p$. Now, if we have a set of intervals $I_{i_1} \ldots I_{i_m}$, each of which intersects $I_{n+1}$ individually, then the interval $(a_{n+1}, \min_{1 \le j \le m} b_{i_j})$ lies in $I_{i_1} \cap \cdots \cap I_{i_m}$. It can be seen that $m \le k$, as $m > k$ would imply that $I_1, \ldots, I_n$ could not be colored by $k$ colors.

Now, in the case that $I_{n+1}$ intersects fewer than $k$ intervals from the intervals $I_1, \ldots, I_n$, the algorithm assigns a color from $\{1, 2, \ldots, k\}$ not assigned to any of the intervals that $I_{n+1}$ intersects. The resulting assignment of a color to $I_{n+1}$, together with the assignment $\mathcal{O}_n$ of $k$ colors to $I_1, \ldots, I_n$, is optimal, as the assignment of $k$ colors to $I_1, \ldots, I_n$ was optimal for $I_1, \ldots, I_n$, and adding an interval to a set cannot decrease the number of colors required to color that set.

The other case is that $I_{n+1}$ intersects exactly $k$ intervals from $I_1, \ldots, I_n$. The algorithm will color $I_{n+1}$ with the color $k+1$, and this assignment together with $\mathcal{O}_n$ is an optimal coloring for $I_1, \ldots, I_{n+1}$, as no coloring can use fewer than $k+1$ colors, as the interval $(a_{n+1}, \min_{1 \le j \le k} b_{i_k})$ lies in $k+1$ intervals. ■

## Problem 5

To be posted.