# CSE 531 Homework Assignment 3

Due in class on Thursday, Oct 11.

There are totally $6$ problems, $10$ points each. You should do them all. We will grade only $5$ problems chosen at my discretion. If it so happens that you don't do one of the problems we don't grade, then no points will be deducted.

**Example 1** (Sample Problem)**.** We want to make change for $n$ cents using the least number of coins. The coins are of denominations $1 = c^0, c^1 \dots, c^k$, for some integers $c > 1$, and $k \geq 0$.

Devise an efficient algorithm to solve this problem.

*Sample Solution.* **When you are asked to devise an algorithm, please conform to the following format:** (a) describe the idea, (b) write the pseudo code, (c) prove its correctness, (d) analyze its running time.

(a) **Idea**: our algorithm is a greedy one. We start by taking as many coins of denomination $c^k$ as possible, then as many of $c^{k-1}$ as possible, and so on. Since there's a coin of denomination 1 (coin $c^0$), this process is guaranteed to finish.

(b) **Pseudo code**: every solution $S$ to this problem is of the form

$$S = (f_0, f_1, \dots, f_k)$$

where the $f_i$ are all natural numbers, $f_i$ is the number of coins of denomination $c^i$ we took, and

$$f_0c^0 + f_1c^1 + \dots f_kc^k = n.$$

```
COIN-CHANGING($n, c, k$)
1: for $j \leftarrow k$ downto $1$ do
2:     $f_k \leftarrow \lfloor n/c^k \rfloor$
3:     $n \leftarrow n - f_kc^k$
4: end for
```

(c) **Proof of correctness**: We will use the "first type" of induction. Basically, our greedy choice is to set $f_k = \lfloor n/c^k \rfloor$, and then solve the subproblem with coins of denominations $c^0, \dots, c^{k-1}$, and the new number of cents $n' = n - f_kc^k$. The cost of a solution $S$ is the number of coins $S$ uses, i.e. cost$(S) = f_0 + \cdots + f_k$. We'd like to find an $S$ minimizing cost$(S)$.

**Lemma 2.** *There exists an optimal solution $O = (f_0, f_1, \dots, f_k)$ for which $f_k = \lfloor n/c^k \rfloor$.*

*Proof.* If $n < c^k$, $f_k = 0 = \lfloor n/c^k \rfloor$ for any feasible solution. Thus, we can assume $n \geq c^k$.

Let $O' = (f'_0, f'_1, \dots, f'_k)$ be any optimal solution. If $f'_k = \lfloor n/c^k \rfloor$, then we are done. Suppose $f'_k \leq \lfloor n/c^k \rfloor - 1$. We have

$$f'_0c^0 + f'_1c^1 + \cdots + f'_{k-1}c^{k-1} = n - f'_kc^k \geq n - \left(\lfloor n/c^k \rfloor - 1\right) c^k \geq c^k. \tag{1}$$

Now, if $f'_j \leq c-1, \forall j = 0, \ldots, k-1$, then

$$f'_0 c^0 + f'_1 c^1 + \cdots + f'_{k-1} c^{k-1} \leq (c-1)(c^0 + \cdots + c^{k-1}) = (c-1)\frac{c^k - 1}{c-1} = c^k - 1,$$

contradicting (1).

Hence, there must be some $j \in \{0, \ldots, k-1\}$ for which $f'_j \geq c$. However, if we reduce $f'_j$ by $c$, and increase $f'_{j+1}$ by 1, then we get another feasible solution where the total number of coins is $(c-1)$ less, contradicting the fact that $O'$ is optimal. □

**Lemma 3.** *Let $O = (f_0, \ldots, f_k)$ be an optimal solution for which $f_k = \lfloor n/c^k \rfloor$, then $O' = (f_0, \ldots, f_{k-1})$ is an optimal solution to the problem with the number of cents $n' = n - c^k \lfloor n/c^k \rfloor$ and coin denominations $c^0, \ldots, c^{k-1}$.*

*Proof.* If there is a better solution $O'' = (f''_0, \ldots, f''_{k-1})$ for the subproblem, i.e.

$$\begin{aligned} f''_0 + \cdots + f''_{k-1} &< f_0 + \cdots + f_{k-1} \\ f''_0 c^0 + \ldots f''_{k-1} c^{k-1} &= n'. \end{aligned}$$

Then, obviously $(f''_0, f''_1, \ldots, f''_{k-1}, f_k)$ is a better solution for the original problem than $O$, contradiction! □

**Theorem 4.** *Algorithm* COIN-CHANGING *returns an optimal solution $S$.*

*Proof.* We show by induction on $k$ that cost$(S)$ = cost$(O)$, where $O$ is an optimal solution.

The base case when $k = 0$ is trivial.

Consider $k > 0$. Let $S = (f_0, \ldots, f_k)$, and $O$ be any optimal solution $O = (f'_0, \ldots, f'_k)$ for which $f'_k = f_k = \lfloor n/c^k \rfloor$. Such an optimal solution exists due to Lemma 1.

By induction hypothesis, $(f_0, \ldots, f_{k-1})$ is an optimal solution to the sub-problem. By Lemma 2, $(f'_0, \ldots, f'_{k-1})$ is an optimal solution to the sub-problem also. Thus,

$$f'_0 + \cdots + f'_{k-1} = f_0 + \cdots + f_{k-1},$$

which implies cost$(S)$ = cost$(O)$ as desired. □

(d) **Running Time:** There is a loop of $k$ iterations. The time in each iteration is dominated by computing $\lfloor n/c^k \rfloor$. We do not discuss numerical algorithms in this course, hence I will not analyze precisely the running time of this algorithm. (Refer to Knuth's TACP for numerical computation algorithms.) Let's just say $f(n,k)$ is the time it takes to compute $\lfloor n/c^k \rfloor$, then our algorithm runs in time $\Theta(kf(n,k))$.

□

**Problem 1.** Let's consider a long, quiet country road with houses scattered very sparsely along it. (We can picture the road as a long line segment, with an eastern endpoint and a western endpoint.) Further, let's suppose that despite the bucolic setting, the residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within four miles of one of the base stations.

Give an efficient algorithm that achieves this goal, using as few base stations as possible.

**Problem 2.** Suppose you have $n$ video streams that need to be sent, one after another, over a communication link. Stream $i$ consists of a total of $b_i$ bits that need to be sent, at a constant rate, over a period of $t_i$ seconds. (Thus, the rate for sending stream $i$ is $b_i/t_i$.) You cannot send two streams at the same time, so you need to determine a *schedule* for the streams: an order in which to send them. Whichever order you choose, there cannot be any delays between the end of one stream and the start of the next. Suppose your schedule starts at time 0 (and therefore ends at time $\sum_{i=1}^{n} t_i$, whichever order you choose). We assume that all the values $b_i$ and $t_i$ are positive integers.

Now, because you're just one user, the link does not want you taking up too much bandwidth, so it imposes the following constraint, using a fixed parameter $r$:

*(*) For each natural number $t > 0$, the total number of bits you send over the time interval from $0$ to $t$ cannot exceed $rt$.*

Note that this constraint is only imposed for time intervals that start at 0, *not* for time intervals that start at any other value. (This scheme is a variation of the so-called *leaky bucket* algorithm for rate limiting or traffic shaping in networking.)

We say that a schedule is *valid* if it satisfies the constraint (*) imposed by the link.

**The problem.** Given a set of $n$ streams, each specified by its number of bits $b_i$ and its time duration $t_i$, as well as the link parameter $r$, determine whether there exists a valid schedule.

**Example.** Suppose we have $n = 3$ streams, with

$$(b_1, t_1) = (2000, 1), \ \ (b_2, t_2) = (6000, 2), \ \ (b_3, t_3) = (2000, 1),$$

and suppose the link's parameter is $r = 5000$. Then the schedule that runs the streams in the order $1, 2, 3$ is valid, since the constraint (*) is satisfied:

- at $t = 1$: the whole first stream has been sent, and $2000 < 5000 \cdot 1$
- at $t = 2$: half the second stream has been sent, and $2000 + 3000 < 5000 \cdot 2$
- at $t = 3, 4$: similar calculations hold.

**Questions.**

(a) Consider the following claim:

*Claim: there exists a valid schedule if and only if each stream $i$ satisfies $b_i \leq rt_i$.*

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

(b) Given an algorithm that takes a set of $n$ streams, each specified by its number of bits $b_i$ and its time duration $t_i$, as well as the link parameter $r$, and determines whether there exists a valid schedule. The running time of your algorithm should be a polynomial in $n$.

**Problem 3.** You are given a minimum spanning tree $T$ of a graph $G = (V, E)$.

(a) Suppose we decrease the weight of some edge $e$ belonging to $T$. Prove that $T$ is still a minimum spanning tree of $G$ (with the new weight function).

(b) Suppose we decrease the weight of some edge $e$ not in $T$. How quickly can you find a new minimum spanning tree of $G$ (with the new weight function)?

**Problem 4.** Let $G = (V, E)$ be a connected graph. In general, $G$ will have more than one spanning trees. Given two different spanning trees $T_1$ and $T_2$ of $G$, $T_1$ and $T_2$ are said to be *friendly* if there is an edge $e_1 \in T_1$ and an edge $e_2 \in T_2$ such that $T_2$ is exactly $T_1 \cup \{e_2\} - \{e_1\}$.

Now, let $T_0$ and $T_k$ be two arbitrary spanning trees of $G$. Show that there exist spanning trees $T_1, \ldots, T_{k-1}$ such that $T_i$ and $T_{i+1}$ are friendly for all $i = 0..k-1$.

**Problem 5.** Timing circuits are a crucial component of VLSI chips. Here's a simple model of such a timing circuit. Consider a complete balanced binary tree with $n$ leaves, where $n$ is a power of 2. Each edge $e$ of the tree has an associated *length* $l_e$, which is a positive number. The *distance* from the root to a given leaf is the sum of the lengths of all the edges on the path from the root to the leaf.

The root generates a *clock signal* which is propagated along the edges to the leaves. We'll assume that the time it takes for the signal to reach a given leaf is proportional to the distance from the root to the leaf.

Now, if all leaves do not have the same distance from the root, then the signal will not reach the leaves at the same time, and this is a big problem. We want the leaves to be completely synchronized, and all to receive the signal at the same time. To make this happen, we will have to *increase* the lengths of certain edges, so that all root-to-leaf paths have the same length (we're not able to shrink edge lengths). If we achieve this, then the tree (with its new edge lengths) will be said to have *zero skew*. Our goal is to achieve zero skew in a way that keeps the sum of all the edge lengths as small as possible.

Devise an algorithm that increases the lengths of certain edges so that the resulting tree has zero skew and the total edge length is as small as possible.

**Problem 6.** The following problem comes up in control theory. I have rephrased it to avoid control theory terminologies. We are given two arrays $A = [a_1, \ldots, a_n]$ and $B = [b_1, \ldots, b_n]$ of real numbers where $a_i \geq 1$ and $b_i \geq 1$. We are to find a way to pair elements of $A$ and $B$ up in some one-to-one manner to minimize a certain objective function. Specifically, each permutation $\pi$ of $\{1, \ldots, n\}$ represents a pairing $(a_i, b_{\pi(i)})$. The cost of the pairing $\pi$, denoted by cost($\pi$) is defined as follows.

$$\text{cost}(\pi) = \sum_{i=1}^{n} \frac{1}{1 + a_i b_{\pi(i)}}.$$

Devise an algorithm to find a pairing $\pi$ with minimum cost.