CSE 101 Homework 4 Solutions

Fall 2019

This homework is due on gradescope Friday November 15th at 11:59pm on gradescope. Remember to justify your work even if the problem does not explicitly say so. Writing your solutions in \LaTeX is recommended though not required.

**Question 1** (Change Making, 40 points). For each of the following sets $S$ determine whether or not the greedy change making algorithm is optimal. In particular, the algorithm attempts to write a positive integer $n$ as a sum of (not necessarily distinct) elements of $S$ by first taking the largest integer in $S$ that does not go over $n$, and then repeatedly adding the largest that will not bring the sum over the limit. For each set either prove that this algorithm always writes $n$ as a sum of the fewest possible number of elements of $S$ or find a counter-example.

(a) The set of odd numbers: $S = \{1, 3, 5, 7, \ldots\}$ [10 points]

(b) The set of squares: $S = \{1, 4, 9, 16, \ldots\}$ [10 points]

(c) The set of numbers whose base-10 representations have only 0s and 1s: $S = \{1, 10, 11, 100, \ldots\}$ [10 points]

(d) The set of powers of 2: $S = \{1, 2, 4, 8, 16, \ldots\}$ [10 points]

**Solution.**

(a) The greedy solution will always find the optimal solution. If $n$ is odd, the greedy solution will pick $n$ (since $S$ is the set of odd numbers). Otherwise, $n-1$ is odd (and hence will be in $S$), and using $(n-1)$ and 1 we can form $n$ i.e. just 2 elements. When $n$ is even, it is not possible to reach $n$ with just 1 element as all the elements in $S$ are odd. Hence, any solution must require greater than or equal to 2 elements from $S$. The greedy solution requires exactly 2 elements and so is optimal.

(b) Let $n$ be 18. The greedy solution would be to pick 16, 1 and 1 i.e. it requires 3 elements from $S$. However, 9, 9 is a better solution as $9+9=18$ and it only uses 2 elements from $S$.

(c) Let $n = 30$. The greedy solution would be to pick 11, 11, 1,1,1,1,1 and 1 i.e. it requires 10 elements from $S$. However, 10, 10 and 10 is a better solution as $10+10+10=30$ and it only uses 3 elements from $S$.

(d) The greedy solution will always find the optimal solution. We will prove this using the exchange argument.

Let $G = \{G_0, G_1, G_2, \ldots, G_{n_1}\}$ be the elements in $S$ selected by the greedy solution (in decreasing order of magnitude). There are $n_1$ elements in $G$.

Lemma: $G$ will have each element from $S$ at-most once. If a single element (say $x = 2^k$) occurred twice in $G$, it means that the greedy solution selected $x$ twice instead selecting $2x = 2^{k+1}$ once, which is a contradiction to the proposed greedy solution.

Let $A = \{A_0, A_1, A_2, \ldots, A_{n_2}\}$ be the elements in any arbitrary solution. There are $n_2$ elements in $A$.

$A_0$ and $G_0$ represent the initial state where no element from $S$ has been selected - so $A_0$ and $G_0$ are in agreement.

We will proceed with a proof by induction. We say that $G$ and $A$ agree till the $i$-th step if the first $i$ elements in $G$ and $A$ are the same.
Let us assume that G and A are in agreement till the i-th step. If \( A_{i+1} \) and \( G_{i+1} \) are equal then A and G agree till the \((i+1)\)-th step. If not, \( G_{i+1} \) must be greater than \( A_{i+1} \), because if \( A_{i+1} \) was bigger and could be selected then the greedy solution would have picked that before \( G_{i+1} \).

If \( n1 = i + 1 \) then G doesn’t require any more elements from S. However, since \( A_{i+1} < G_{i+1} \), we have \( n2 > i + 1 \) since A requires more elements to sum up-to \( n \). In this case, we can replace all elements in A from index \( i + 1 \) to \( n2 \) by \( G_{i+1} \) and get a better solution.

If \( n1 \neq i + 1, G_{i+2} + \ldots + G_{n1} \) must sum up-to \( y \) where \( y < G_{i+1} \). This follows from the lemma stated above. Since all elements following \( G_{i+1} \) are smaller powers of 2 than \( G_{i+1} \) and occur at-most once, they can sum up to a maximum of \( G_{i+1} - 1 \).

Since \( A_{i+1} + \ldots + A_{2n2} = G_{i+1} + \ldots + G_{n1}, A_{i+1} + \ldots + A_{n2} \) must sum up to \( G_{i+1} + y \). Consider the elements in \( A_{i+1} + \ldots + A_{n2} \). If A contains duplicates of any element of S, we can add them together to get a larger power of 2. Keep doing this until we are left with at most one of each element of S. Now, we claim that one of these elements must be \( G_{i+1} \). None of them will be \( > G_{i+1} \) because if an element \( A' \) was larger than \( G_{i+1} \), we would have \( A' \geq 2G_{i+1} \), which is not possible since \( G_{i+1} + G_{i+2} + \ldots + G_{n1} = G_{i+1} + y < 2G_{i+1} \). All of them cannot be \( < G_{i+1} \) since if that were true, they would sum up to at most \( G_{i+1} - 1 \) which is \( < G_{i+1} + y \). Therefore, we can exchange the elements of A that added up to \( G_{i+1} \) with the element \( G_{i+1} \) and get a better solution. Thus at every step, the greedy solution is no worse than the arbitrary solution.

If we keep repeating this till \( n1 \) steps, we can convert any arbitrary solution to the greedy solution. Hence, the greedy solution is optimal.

**Question 2** (Bus Management, 60 points). Genevieve manages the bus system in her city. She has several stations and a scheduling of trips that she needs to manage. Each trip specifies that a bus needs to leave a particular station at one time and arrive at a different station at another. Any bus at the departing station can be used to take the trip. Also Genevieve can have the buses at her disposal start the day in any stations that she needs them. She wishes to know what the smallest possible number of busses she needs to fulfill all of the trips in her schedule and how to determine which busses are used when for it. Give an efficient algorithm to determine this.

**Solution.** We first define some notation. The input to our algorithm will be a list of trips, each with start/end times and departing/arriving stations. Let \( s_i \) and \( t_i \) be the starting and ending times of trip \( i \). Let \( X \) be the departing station of trip \( i \), and let \( Y \) be the arriving station.

**Algorithm.** Intuitively, our algorithm looks at each trip and checks whether it is possible to make the trip without increasing the number of buses used. This is possible if one of the buses already in use meets two conditions: its last trip ended at the departing station \( X \), and its last trip ended at a time earlier than \( s_i \). If there exists such a bus, our algorithm has it make the trip. If not, another bus is added.

To keep track of the start and end times, we add all of the trip start times \( s_i \) and end times \( t_i \) into a single priority queue, tripTimes. We will also keep track of the current bus schedule, which describes the buses that are available at each station at a given time. Initially, this schedule is empty.

To process each trip time, our algorithm does the following:

- Delete the minimum element of tripTimes.
- If the minimum element is a start time \( s_i \), get the departure station \( X \) of trip \( i \).
  - If there is a bus \( b \) available at \( X \), assign trip \( i \) to \( b \).
  - If there are no buses available at \( X \), create a new bus \( b \) and have trip \( i \) assigned to \( b \).
- If the minimum element is an end time \( t_i \), get the arrival station \( Y \) of trip \( i \). Add the bus \( b \) that took trip \( i \) to the arrival station \( Y \) (it is now available to take further trips).

Repeat the above until all trips are processed (tripTimes is empty). Return the total number of buses used.

**Correctness.** We first show that the output of our greedy algorithm will be a valid bus schedule. To show this, we show that whenever our algorithm assigns a trip \( i \) to a bus \( b \), the previous trip done by \( b \) always
ends before trip $i$ starts and ends at the same station that trip $i$ starts from. This is because our algorithm processes the trips in order from earliest to latest. If we assign trip $i$ to a bus $b$, $b$ must be available at the departing station $X$. The only way for $b$ to be available is if the end time of a previous trip ending at $X$ has already been processed, which means that $b$’s previous trip must have ended earlier than trip $i$’s start time. So it is valid to add trip $i$ to bus $b$. Since this is true for any arbitrary trip assignment, we conclude that the resulting bus schedule is valid.

To prove the correctness of this algorithm, we will use the exchange argument. Let $G$ be the trip schedule obtained by the greedy algorithm. We want to show that if there is an arbitrary solution different from $G$, we can exchange trips in the arbitrary solution to get a new solution that has a trip schedule closer to $G$.

Assume we are looking at the trips in increasing order of start time. Let $A_i$ be an arbitrary solution that assigns the first $i$ trips to the same buses that $G$ does. Consider trip $i + 1$ - if $A_i$ assigns trip $i + 1$ to a different bus than $G$ does, then trip $i + 1$ was supposed to be assigned to a bus $g$, but was instead assigned to another bus $a$ in solution $A_i$. We can split this into two cases: either $g$ is a new bus and trip $i + 1$ is its first trip, or $g$ is an already existing bus.

If $g$ is a new bus, then our greedy algorithm must have found that there were no existing buses available at the departure station of trip $i + 1$. Since $A_i$ has an identical bus schedule to $G$ for the first $i$ trips, there must have also been no existing buses available at the departure station when $A_i$ assigned trip $i + 1$. This means $A_i$ would have also assigned trip $i + 1$ to a new bus, so bus $a$ must be a new bus. In this case, we can construct our new solution $A_{i+1}$ by just renaming bus $a$ to bus $g$. This only changes the labels of the buses, not the actual trips, so $A_{i+1}$ is still a valid solution, and since trip $i + 1$ is now assigned to bus $g$, $A_{i+1}$ has the first $i + 1$ trips assigned to the same buses as the greedy solution.

If $g$ is an already existing bus, consider bus $a$ and bus $g$ in solution $A_i$. We claim that in solution $A_i$, we can exchange all of the trips in bus $a$ from $i + 1$ onwards, with all of the trips in bus $g$ that are different from the greedy solution. Let this be our new solution $A_{i+1}$, which has the first $i + 1$ trips assigned to the same buses as the greedy solution.

This exchange does not increase the total number of buses used, so we only need to show that $A_{i+1}$ is still a valid solution. All of the buses other than buses $a$ and $g$ are the same as they were in $A_i$ (so are valid because $A_i$ is valid) so we only need to check that $a$ and $g$ remain valid after the exchange.

Since trip $i + 1$ was assigned to bus $g$ in the greedy solution, bus $g$ must have been available at the departure station of trip $i + 1$ when trip $i + 1$ was assigned. Since $A_i$ and $G$ have identical bus schedules for the first $i$ trips, and since $A_i$ originally assigned trip $i + 1$ to bus $a$, both bus $a$ and bus $g$ must have been available at trip $i + 1$’s departure station when solution $A_i$ assigned trip $i + 1$. This means we can assign trip $i + 1$ to bus $g$. We are then just taking all of the trips in bus $a$ after trip $i + 1$ and assigning them to bus $g$ instead without changing their order, so our new sequence of trips for bus $g$ is valid.

In the case of bus $a$, trip $i + 1$ is being replaced by the trip that $A_i$ had originally assigned to $g$. This trip must depart from the same station as trip $i + 1$ since bus $g$ and bus $a$ were both at $i + 1$’s departure station, and since both buses were available when trip $i + 1$ was processed, bus $a$ should be able to take the trip $A_i$ had originally assigned to $g$. So, since we can assign this trip to bus $a$, and since we are then taking all of the subsequent trips in bus $g$ and assigning them to bus $a$ instead without changing their order, we will get a valid sequence of trips for bus $a$.

We conclude that $A_{i+1}$ must be a valid solution, and since $A_{i+1}$ has $i + 1$ trips in common with the greedy solution, any arbitrary valid schedule of buses can be changed to another schedule that contains all the trips in the same order as $G$, using the same or fewer number of buses. We conclude that $G$ is a valid schedule that uses the minimum number of buses.

**Runtime.** Let $n$ be the total number of trips. We store the start and end times in `tripTimes` individually, so there are $2n$ elements in `tripTimes` and we process each element exactly once. Processing a single element involves updating a priority queue so takes $O(\log n)$ time. We process $O(n)$ elements, so the runtime of our algorithm is therefore $O(n \log n)$.

**Question 3** (Extra credit, 1 point). *Approximately how much time did you spend working on this homework?*