Assignment 4

Ayman Shahriar UCID: 10180260 Tutorial: T02

November 2, 2020

1)

The greedy algorithm that solves this problem will use this greedy rule: pick the largest denomination while the total remains less than or equal to N (where N is the amount of change needed). Once the total is equal to N, the algorithm returns the sequence of coins it chose at each iteration and terminates.

We will use the stays ahead argument to prove that the greedy algorithm always yields an optimal solution.

Suppose that the greedy algorithm is not optimal.

Let $A = (n1, \ldots, nk)$ be the sequence of coins returned by our algorithm. The coins in S are in the order in which they were added by the algorithm, so A is in decreasing order.

Now suppose $O = (n'_1, \ldots, n'_m)$ is the optimal solution to this problem.
Let O be ordered in decreasing order.

Since the greedy algorithm is not optimal, it must be that $k > m$, because an optimal solution must contain less coins than a non-optimal solution.

Now, we will prove using induction that $n_i \geq n'_i$ for all $1 \geq i \geq m$

Proof by Induction:

Base case: We know that when selecting the first coin, the greedy algorithm will select the largest denomination that is less than or equal to N. Then n'_1 cannot be a denomination that is larger than n_1, so it must be that $n_1 \geq n'_1$.

Inductive Step:

Suppose $n_i \geq n'_i$ for $1 \leq i < m$ (Inductive Hypothesis).
We will use our inductive hypothesis to prove that $n_{i+1} \geq n'_{i+1}$.

Note that the greedy algorithm returns the sequence of coins S in decreasing order, so $n_i \geq n_{i+1}$. Also note that O is in decreasing order, so $n'_i \geq n'_{i+1}$.

Also, when choosing the coin for n_{i+1}, our algorithm will choose the largest denomination such that the sum of selected coins will be less than or equal to N.

This means that since $n_i \geq n'_i$ (according to the inductive hypothesis), then the denomination chosen for n'_{i+1} cannot be a coin that is larger than n_{i+1}.

Then $n_{i+1} \geq n'_{i+1}$, as required.

Thus, $n_i \geq n'_i$ for all $1 \geq i \geq m$

Back to out main proof:

Using induction, we have proved that $n_i \geq n'_i$ for all $1 \leq i \leq m$.

And since $k > m$, this means that the sum of the values of all coins in S is greater than the sum of the values of all coins in S'.

We know that our greedy algorithm always returns a sequence whose sum is N (where N is the amount of change specified by the problem), so then O returns a sequence that is actually smaller than the amount specified by the problem. This contradicts the assumption that S' is a correct, optimal solution.

Thus, using contradiction we have proved that the greedy algorithm always returns an optimal solution.

2)

a) Preconditions:

- An integer $N \geq 1$ that represents the length of the road in miles
- A non-empty set $H = \{h_1, \ldots, h_i\}$ $(1 \geq i \geq N)$ of house locations along the road.
- The house locations increase as you go from the east to west along the road. So the location at 1 represents the easternmost location, and the location at N represents the westernmost location.

Postcondtions:

- Return a set of tower locations $T = \{t_1, \ldots, t_i\}$ $(1 \geq i \geq N)$ such that:
- $\forall h_i \in H$, $\exists t_i \in T$ such that $|t_i - h_i| \leq 4$. In other words, every house in H has to be within 4 miles of a tower in T.
- The length of T is minimized. In other words, there does not exist a set T' such that $|T'| < |T|$ and $\forall h_i \in H$, $\exists t'_i \in T'$ such that $|t'_i - h_i| \leq 4$.

b)

Our greedy algorithm selects the earliest (ie, easternmost) available house, and places a tower 4 miles after that house (ie, 4 miles west of that house). Then the algorithm iterates over each remaining house h in order of increasing house location, and places a tower 4 miles after h if h is not within the range of the last added tower. At any point in the algorithm, if the location $(h+4) \geq N$, then the algorithm will place a tower at N (which is the westernmost point of the road) instead.

```
Function Compute-Tower-Locations($H = \{h_1, \ldots, h_i\}$, $N$)

    Sort house locations in increasing order and re-label so that
      $h_1 \leq h_2 \leq \ldots \leq h_i$
    Let $H'$ be this sorted set of house locations
    Let $T = \emptyset$
    Initialize integer variable $t$

    // Calculate first tower location, add to T
    $t = \min(h_1 + 4, N)$
    $T = T \cup \{t\}$

    // Calculate rest of the tower locations
    for $j = 2$ to $i$: do
        if $|h_j - t| \geq 4$ then
            t = $\min(h_j + 4, N)$
            $T = T \cup \{t\}$

    return T
```

c) Our algorithm returns an optimal solution because it places each tower at the farthest location that the tower could possibly be placed in order for all the houses before it to be within range of a tower.

That is because at each iteration, the algorithm finds the earliest (easternmost) house that is out of range, and then it places a tower 4 miles after it.

So it would be impossible to reduce the towers because if we removed a tower, there would be at least one house located 4 miles before it that would become out of range.

Also note that when we add a tower t, the house h that is 4 miles before that tower does not have any towers 4 miles before it (otherwise h would have been within range and we would not have needed to add the new tower t).

Then this means that each tower is spaced at least 8 miles apart, so there is no way to remove a tower t_1 and relocate a tower t_2 that was adjacent to t_2 in such a way that t_2 will simultaneously cover all the houses that t_2 already covers along with the houses that t_1 covered before.

So our algorithm returns an optimal solution because there is no way we can reduce the number of towers in the solution and still maintain full coverage of all the houses.

d) No, the solution returned by the algorithm is not the only possible solution for all possible inputs.

In our algorithm we go from east to west, selecting the earliest (ie, easternmost) house and placing a cellphone tower 4 miles west from that house. Then the algorithm iterates over each remaining house h in order of increasing house location, and places a tower 4 west after h if h is not within the range of the last added tower. So our algorithm processes houses from east to west.

However, we could do the exact opposite and still get an optimal solution. We could go from west to east, selecting the westernmost available house and placing a cellphone tower 4 miles east of that house. And then we would process the other houses similarly to how our algorithm does, except that we process the houses in order of decreasing house location (ie, from west to east).

This algorithm will still return an optimal solution because we just ”flipped” the order of the houses being processed. Even though the location of the towers will be different (due to the reverse processing order), the number of towers needed to cover all the houses will be the same because the spacing of each house relative to each other in the ”flipped” order of processing is still the same. So our modified algorithm will behave just like our original algorithm and calculate the tower locations needed to return an optimal solution.

Thus, the solution returned by our algorithm is not the only possible solution for all inputs.

e)

Our algorithm is correct if: 1) Each house is within 4 miles of a tower 2) There is no smaller set of towers where each house is still within 4 miles of a cellphone tower.

Proof of 1)

In our algorithm, we select the easternmost house that is not covered by any tower, and add a tower 4 miles after that house. So that house is now within 4 miles of a tower. We then skip over all houses (in order from east to west) that are within range of that tower, and then keep on repeating these two steps until there are no more unprocessed houses left. This means that every house h was either skipped over because it is already covered, or a tower was added 4 miles after h to bring it coverage.

Therefore, it means that in our solution, every house is within 4 miles of a tower.

Proof of 2) We will use the "stays ahead" method to prove that there is no smaller set of towers than our solution where each house is still within 4 miles of a tower.

Suppose the solution returned by our algorithm is not optimal.
Let $O = \{o_1, \ldots, o_m\}$ be an optimal set of tower locations that is ordered by increasing location (so from east to west)
Let $T = \{t_1, \ldots, t_k\}$ be the set of tower locations returned by our greedy algorithm.

Since T is not optimal, then it must be that $k > m$ because the optimal set must have less towers.

We will now prove by induction that $o_i \leq t_i$ for all $1 \leq i \leq m$

Proof by Induction:

Base Case: t_1 is the earliest (easternmost) tower in our returned set T, which means that it is located 4 miles after the easternmost house in the street.

Since each house has to be within 4 miles of a tower, it means t_1 is the farthest (westernmost) location where we can put the first cellphone tower and the first house in the street will still be within 4 miles of one.

So it follows that the earliest cellphone tower in the set O cannot be located after t_1, so $t_1 \geq o_1$

Inductive Step

Suppose $t_i \geq o_i$ for $1 \leq i < m$ (Inductive hypothesis).
We will use the inductive hypothesis to prove that $t_{i+1} \geq o_{i+1}$.

Since our algorithm selects the easternmost house that is not covered and puts a tower 4 miles after that house, this means that there is a house h that is not covered by t_i, but is covered by and 4 miles before t_{i+1}.

From the inductive hypothesis, we know that $t_i \geq o_i$, so putting a cellphone tower at o_i will not cover the house h (so will not be within 4 miles of h).

So in order to cover h, we need to place an $(i+1)^{th}$ cellphone tower at location o_{i+1}. And since o_{i+1} has to be within 4 miles of h, then this means that o_{i+1} has to be located at or before the location t_{i+1}.

So then $o_{i+1} \leq t_{i+1}$, as required.

Thus, we have shown that $o_i \leq t_i$ for all $1 \leq i \leq m$

Back to our main proof:

According to our greedy algorithm, there is a house h that is 4 miles behind the tower located at t_k.

This house is not covered by t_{k-1} because if it was, then the algorithm would not have placed t_k 4 miles after h.

Note that since $k > m$, then it must be that $m \leq k - 1$.
Then since $o_i \leq t_i$ for all $1 \leq i \leq m$ (which we proved by induction), it must be that $o_m \leq t_{k-1}$.

So this means that o_m, which is the the farthest (ie, westernmost) tower in O, is located before t_{k-1}.
So then the farthest tower in O does not cover (ie, is not within 4 miles) of h, which contradicts the assumption that the solution O is optimal.

Thus, our greedy algorithm always returns an optimal set T.

f)

The running time of our algorithm is in $\Theta(nlogn)$

Sorting the input house locations will take $\Theta(nlogn)$ time (if we use heapsort or mergesort), where n is the number of houses given as input.

Initializing T and t will take constant time $\Theta(1)$.

We can calculate the first tower location by selecting the first house among all the sorted house locations, add 4 to it, check if it's smaller than N, and add whatever is smaller to T. Each of these steps take constant time, so the runtime of adding the first tower location to T is $\Theta(1)$

The for loop will always iterate from 2 to n (where n is the number of houses given as input), so the loop always iterates $n - 1$ times.

Since checking if $|h_j - t| \geq 4$, calculating $\min(h_j + 4, N)$ and adding a tower location to T all take constant time, it means that the runtime of the entire for loop is in $\Theta(n)$.

Since $n \in O(nlogn)$, this means that sorting the house locations is the slowest step in our algorithm.

Thus, the runtime of our algorithm is in $\Theta(nlogn)$

3)

a) Preconditions:

- A sequence of positive integers $B = (t_1, \ldots, t_n)$ where t_i is the number of days required to read book i.

Postconditions:

- Return a sequence $S = (t_1, \ldots, t_n)$ such that if the books were read one at a time in the order of S, and immediately returned, then the sum of all the daily late fees would be minimized.
- The late fee for a day is defined by the number of unreturned books you currently have left at that day. So the total late fees is the sum of the late fees from day 1 (when you first start reading the first book) to the day you finish reading the last book
- S is a permutation of B and does not contain any additional elements.

b)

The optimal solution is: (t_2, t_3, t_1)
Late fee of optimal solution: $(5 \times 3) + (8 \times 2) + (10 \times 1) = \$41$

c) The greedy alrogithm solves the problem by using this greedy rule: pick the smallest read-time that has not been added to S, and add this read time to the end of S

Here is the pseudocode of our greedy algorithm:

```
Function Read-Books($B = (t_1, \ldots, t_n)$)

    Sort $B$ in increasing order and re-label so that $t_1 \leq t_2 \leq \ldots \leq t_n$
    Initialize empty sequence $S$

    for $i = 1$ to n: do
        Add $t_i$ to end of $S$

    return $S$
```

d)

The higher the number of books you have at a certain day, the higher the library fees will be for that day.
So the slower it takes to decrease the number of unreturned books, the higher the total library fee will be.

So in order to minimize the total fee, we want our algorithm to return a sequence that decreases the number of unread books as quickly as possible. This will be possible if we define our greedy algorithm to choose the books in order of increasing read times and add to S.

That way, we will read the books with the smallest read times first, which will decrease our number of unreturned books at the fastest possible rate. This in turn will minimize our library fees.

This is why our alrogithm returns an optimal solution

e) Our algorithm always returns a sequence where the books are read in the order of increasing read-times. A solution to the library books problem is optimal if the total library fees incurred is minimized. Using the "Exchange Argument" method, we will prove that our algorithm always returns an optimal solution.

Consider an arbitrary and optimal scheduling (ie. sequence) S'.
Suppose book b is read right before book a in S', but $t_a \leq t_b$.
Let S be a sequence created by swapping a and b.

Now, suppose that the total number of books in each sequence is n.
Note that the late fees incurred when reading other books besides a and b are the same for both sequences.
So the only difference between the two sequences is that the late fees incurred when reading a and b will be different.

This means that the difference between the total late fees of S' and S solely depend on the difference between the late fees accumulated during reading a and b

Now let us compare the late fees accumulated when reading a and b in the two sequences (remember that n is the number of books in each sequence):

In S', the late fees incurred during reading a and b is:
$(n \cdot t_b) + (n-1) \cdot t_a = (n \cdot t_b) + (n \cdot t_a) - t_a$.

In S, the late fees incurred during reading a and b is:
$(n \cdot t_a) + (n-1) \cdot t_b = (n \cdot t_a) + (n \cdot t_b) - t_b$.

Since $t_b \geq t_a$, we get:
$-t_b \leq -t_a$
$= (n \cdot t_a) + (n \cdot t_b) - t_b \leq (n \cdot t_b) + (n \cdot t_a) - a$

So the late fees accumulated when reading a and b in S will be less than or equal to that of S'.

And since the difference between the total late fees of S' and S solely depend on the difference between the late fees accumulated during reading a and b, it means that the sum of all daily late fees in the swapped sequence S will be less than or equal to that of S'

Since a and b (where $t_a \leq t_b$) are arbitrary books that are consecutively located besides each other in S', this means that we can continue swapping consecutive pairs of books (where the read-time of the book that comes before is greater of equal to the read-time of the book that comes after) until all the books are ordered in order of increasing read-time, and the total late fee of this ordered sequence will be less than or equal to that of the optimal sequence S'.

And since our algorithm returns such a sequence where the books are ordered in increasing read-time, it means that our algorithm will always return an optimal sequence.

Thus, we have proved that our algorithm is correct.

f) The running time of our algorithm is in $\Theta(nlogn)$

Sorting the input read times in decreasing order will take $\Theta(nlogn)$ (if we use heapsort of mergesort), where n is the number of read times given as input.

Initializing and returning S at the end of the algorithm both take constant time $\Theta(1)$

The for loop will always iterate from 1 to n (where n is the number of houses given as input), so the loop always iterates n times.

In each iteration of the loop, we add a read-time of a book into S, which takes constant time $\Theta(1)$. Then since the loop iterates n times, the runtime of the entire for loop is $\Theta(n)$

Since $n \in O(nlogn)$, this means that sorting the read-times is the slowest step in our algorithm.

Thus, the runtime of our algorithm is in $\Theta(nlogn)$