CS 270 Homework 3

Neel Gupta

February 1, 2023

Problem 1. Suppose you want to drive from USC to Santa Monica. Your gas tank, when full, can hold enough gas to go p miles. Suppose there are n gas stations along the route at distances $d_1 \leq d_2 \leq ... \leq d_n$ from USC. Assume that all $d_i - d_j \leq p$ when $i = 2, 3, ..., n$ and $j = i - 1$. Assume you start with a full gas tank. Give the most efficient algorithm to determine which gas stations you should stop at and prove that your algorithm yields an optimal solution (i.e., the minimum number of gas stops). Give the time complexity of your algorithm as a function of n.

Answer: Starting from USC, choose the farthest gas station within p miles and eliminate any closer gas stations. Now starting from that gas station, go p more miles until reaching the new furthest gas station. Choose that gas station, eliminate closer stations, and repeat.

Algorithm 1 Determine which gas stations to stop at

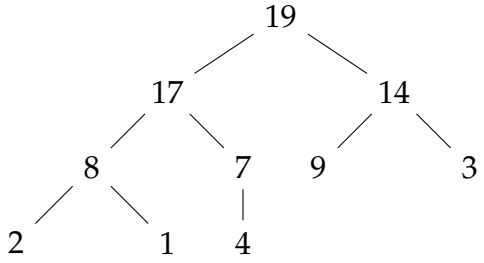
```
Let D be the set of gas station's distances that are assumed to be sorted
procedure MINGASSTOPS(D)
    Let S be the resulting set; S ← ∅
    Let g be the number of gas stops taken; g ← 0
    Let distanceTraveled ← 0
    for every station s_i at distance D_i when i = 1, 2, ..., n do
        if D_i − distanceTraveled > p then
            g ← g + 1
            distanceTraveled ← D_{i−1}
            S = S ∪ {s_{i−1}}
        end if
    end for
return g
end procedure
```

This algorithm runs in $O(n)$ and uses a greedy approach under the criteria of choosing the gas station that is at most p miles from the current distance traveled.
Consider our algorithm and an optimal algorithm that respectively return sets $S := \{s_1, s_2, ..., s_g\}$ and $O := \{t_1, t_2, ..., t_h\}$. Since our first stop is s_1, we

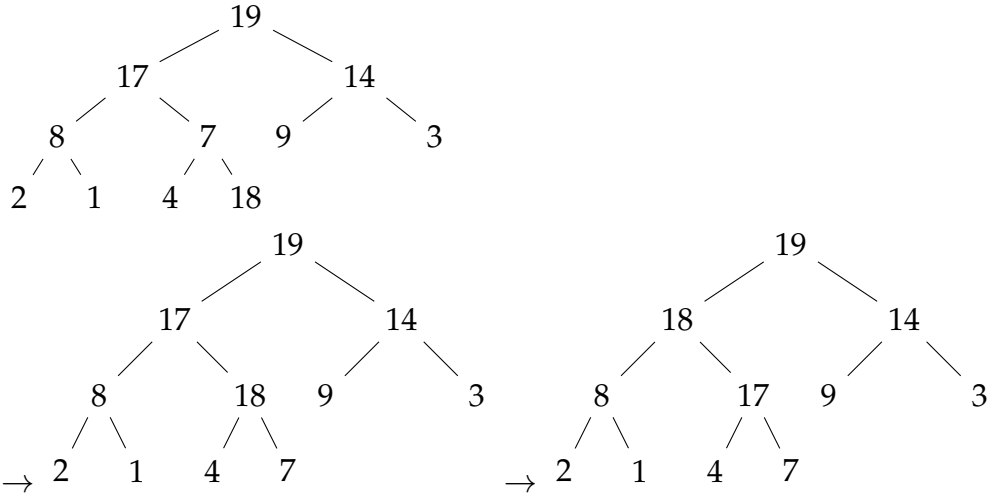
show that there is an optimal solution beginning at s_1. If $t_1 = s_1$, then O is a valid optimal solution under this criteria. If $t_1 \neq s_1$, then t_1 is before s_1 since our greedy algorithm takes the furthest station within our gas tank range. Now consider the solution set $O' := \{s_1, t_2, ..., t_h\}$, then $|O| = |O'|$. Second, we need to consider whether O' is valid. By acting greedily, we can reach s_1. Since O is optimal, there is at least enough gas to go from t_1 to t_2 and because t_1 is at most as far as s_1, then there is enough gas to go from s_1 to t_2. Therefore O' is just as optimal as O, then by exchange arguments, if you swap any $s_i \in S$ with any element in O, the solution will be just as optimal as any arbitrary optimal solution. Thus, S itself is optimal.

Problem 2. The array A holds a max-heap. What will be the order of elements in A after a new entry with value 18 is inserted into this heap? Show all your work. A = 19, 17, 14, 8, 7, 9, 3, 2, 1, 4.

Answer: A expressed as a binary max-heap:



After adding 18 to the heap, A = 19, 17, 14, 8, 7, 9, 3, 2, 1, 4, 18.



After trickling up, A = 19, 18, 14, 8, 17, 9, 3, 2, 1, 4, 7

Problem 3.

(a) Consider the problem of making change for n cents using the fewest number of coins. Describe a greedy algorithm to make change consisting of quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent). Prove that your algorithm yields an optimal solution. (Hint: consider how many pennies, nickels, dimes, and dimes plus nickels are taken by an optimal solution at most.)

(b) For the previous problem, give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Assume that each coin's value is an integer. Your set shold include a penny so that there is a solution for every value of n.

Answer: (a) A greedy algorithm to achieve this task would be to take the biggest possible coin denomination at each iteration and subtract this value from n until the smallest coin denomination does fit within the remaining balance. If this algorithm terminates with a balance equal to 0, then there exists a possible solution set S with denominations to make change of n cents.

```
procedure COINCHANGE(n)
    Let d be a list of coin denominations 25, 10, 5, 1
    sort(d) in ascending order
    res ← 0
    maxIndex ← len(d)-1
    while n > 0 do
        if n ≥ d[maxIndex] then
            n ← n − d[maxIndex]
            res ← res + 1
        else
            maxIndex ← maxIndex-1
            if maxIndex = −1 then return -1
            end if
        end if
    end while
    if n = 0 then return res
    else return -1
    end if
end procedure
```

If $n < 5$, use n pennies. If $5 \leq n < 10$, use 1 nickel and $n - 5$ pennies. If $10 \leq n < 25$, use $\lfloor n/10 \rfloor$ dimes. If $25 \leq n$, use $\lfloor n/25 \rfloor$ quarters and use the former 3 cases on $n - 25\lfloor n/25 \rfloor$.

Consider using pennies and nickels. At most, there can be 4 pennies because any 5 pennies can be replaced by a nickel, and this switch reduces the number of coins by 4. If $n \equiv 0 \mod 5$, then we would only use nickels and no pennies. We would use as many nickels as possible before going to pennies.If the remainder is more than 5, we could use both nickels and pennies.

Consider using pennies, nickels, and dimes. At most, there can be 1 nickel because any 2 nickels would be replaced by a dime which would reduce the coin count by 1.If $n \equiv 0 \mod 10$, we would only use dimes and no nickels or pennies. We would use as many dimes as possible before going to smaller coins. If the remainder is more than 10, we could use both pennies, nickels and dimes.

Consider using all coins. At most, there can be 2 dimes since 3 dimes would become a quarter and a nickel which reduces the coin count by 1. If the remainder is more than 30, we would use the combination of as many quarters and nickels as possible. Then the number of nickels would combined into dimes according to the second consideration, so if the remainder is greater than 30, take as many quarters as possible.

Consider a remainder of above 25 cents but below 30 cents without quarters. According to the first 3 rules, we would take 2 dimes, 1 nickel, and $n - 25$ pennies. We could reduce the coin count by replacing the 2 dimes and nickel with a quarter, reducing the total coin count by 2.

Since every case of using k different types of coins to then using $k + 1$ different types of coins reduces the total coin count, a greedy approach provides the optimal solution for this problem.

(b) The greedy algorithm would not yield the optimal solution for the following set of coin denominations, $\{1, 90, 100\}$. Considering when $n = 380$, the greedy solution proposed would return 83 since after we take 3 of the 100 cent coins, we need 80 of the 1 cent coins. A far more optimal solution for this case would be to take 2 100 cent coins and then 2 80 cent coins to make the total.

Problem 4. You are given the position of N mice and N holes on a 1-dimensional number line. Each hole can accomodate at most 1 mouse. A mouse can stay in place, move one step right from location x to $x + 1$, or

move one step left from location x to $x-1$. Devise an algorithm to assign mice to holes so that the number of moves taken by the mice is minimized. Your algorithm should return the minimum number of moves taken to assign mice to holes and be in $O(N \log N)$ time.

Answer: This problem can be solved with a greedy approach. Moving every mouse to its nearest hole can minimize the time. After sorting both mouse positions and hole positions, pairing the ith mouse with the ith hole when $i = 1, 2, ..., N$ will give us the minimum number of steps by subtracting ith mouse's spot from how for they have to go to go get to the ith holes. Summing the absolute value of this difference will give us the minimum number of steps that all mice have to take.

```
Let M be mice positions
Let H be hole positions
procedure MOUSE2HOLES(M,H)
    res ← 0
    sort(M)
    sort(H)
    for i = 1, 2, ..., N do
        res ← res+|M_i − H_i|
    end for
return res
end procedure
```

This algorithm runs in $O(N \log N)$ which is the time taken to sort the mouse and house positions lists. Consider the following mice at positions M_1 and M_2 s.t. $M_1 < M_2$ and the following house positions H_1 and H_2 s.t. $H_1 < H_2$. By exchange arguments, it suffices to show that $|M_1 - H_1| + |M_2 - H_2| \leq |M_1 - H_2| + |M_2 - H_1|$, so any swap between mice and houses causes a worse travel time.

Problem 5. Farmer John has N cows numbered $1, 2, ..., N$ who are planning to escape and join the circus. His cows will be in a cow tower to try and escape where cow_i has weight W_i and strength S_i when $i = 1, 2, ..., N$. The risk value R_i is equal to the total weight of all cows on its back minus S_i. We want to design an algorithm to help cows find their positions in the tower s.t. we minimize the maximum risk value of all cows. For each of the following greedy algorithms either prove that the algorithm correctly solves this problem or provide a counterexample.

(a) Sort cows in ascending order of S_i from top to bottom.

(b) Sort cows in ascending order of $S_i + W_i$ from top to bottom.

Answer: (a) Consider the following counterexample.
$S := \{3, 2, 1\}$ and $W := \{3, 5, 2\}$, then the cow with the highest strength will be on the bottom which has strength equal to 3 but risk equal to 4. If cow_2 were on the bottom, the risk would then be equal to 3 which is less than 4, so sorting in ascending strength as our greedy criteria does not always yield the most optimal solution.

(b) Define an inversion in a solution that is $S_i + W_i$ for cow_i is more than $S_j + W_j$ for cow_j but cow_j is below cow_i in the cow tower. We can show that inversions can be removed without increasing the overall maximum risk. Given an optimal solution, we show that we can remove inversions without affecting the overall optimality of the solution. If an inversion exists between cow_i and cow_j, then either cow_i and cow_j are adjacent or there is a pair of cows between cow_i and cow_j that are adjacent and have an inversion. Assume cow_i and cow_j are adjacent and have an inversion. Now we can show that moving cow_i below cow_j will not increase the risk of either cow.

We can move cow_i below cow_j, but we will not increase R_i because our greedy assumption cow_i is stronger or heavier than cow_j.

We can then move cow_j above cow_i, and R_j is not decreased as a result of this because either cow_i can hold more or cow_i's heavier weight does not need to be held up by cow_j, and both of these situations reduce the overall risk.

Since we know that removing an inversion will not affect the overall risk negatively, if we are given an optimal solution with any inversions in it, we can remove these wihtout affecting the solution's optimality. When there are no more inversions, the solution will be the same as ours, that the cows are sorted in order of strength plus weight. Thus, our solution is optimal.