Homework 3 Solutions

2 February 2023

GRADED

1. Suppose you want to drive from USC to Santa Monica. Your gas tank, when full, holds enough gas to go p miles. Suppose there are n gas stations along the route at distances $d1 \leq d2 \leq ... \leq dn$ from USC. Assume that the distance between any neighboring gas stations, and the distance between USC and the first gas station, as well as the distance between the last gas station and Santa Monica, are all at most p miles. Assume you start from USC with the tank full. Your goal is to make as few gas stops as possible along the way. 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. (15 points)

   Solution:
   Greedy algorithm: The greedy strategy we adopt is to go as far as possible before stopping for gas. That is when you are at the i^{th} gas station, if you have enough gas to go to the $(i+1)^{th}$ gas station, then skip the i^{th} gas station. Otherwise, stop at the i^{th} station and fill up the tank.
   Proof of optimality:
   The proof is similar to that for the interval scheduling solution we did in the lecture. We first show (using mathematical induction) that our gas stations are always to the right of (or not to the left of) the corresponding base stations in any optimal solution. Using this fact, we can then easily show that our solution is optimal using proof by induction.

   (a) First, we show our gas stations are never 'earlier' than the corresponding gas station in any optimal solution: Let $g_1, g_2, \ldots, g_m$ be the set of gas stations at which our algorithm made us refuel. Let $h_1, h_2, \ldots, h_k$ be an optimal solution. We first prove that for any indices i $<$ m, $h_i \leq g_i$ Base case: Since it is not possible to get to the $(g_1+1)^{th}$ gas station without stopping, any solution should stop at either g1 or a gas station before g_1, thus h1 $\leq$ g1. Induction hypothesis: Assume that for the greedy strategy taken by our algorithm, $h_c \leq g_c$ Inductive step: We want to show that $h_{c+1} \leq g_{c+1}$. It follows from the same reasoning as above. If we start from h_c, we first get to g_c (IH) and , when leaving g_c, we now have at most as much fuel as we did if we had refilled at g_c. Since it is not possible to get to $g_{(c+1)+1}$ without any stopping, any solution should stop at either g_{c+1} or a gas station before g_{c+1} , thus $h_{c+1} \leq g_{c+1}$
   (b) Now assume that our solution requires m gas stations and the optimal solution requires fewer gas stations. We now look at our last gas station. The reason we needed this gas station in our solution was that there is a point on I-10 after this gas station that cannot be reached with the amount of gas when we left gas station m-1. Therefore we would not have enough gas if we left gas station m-1 in any optimal solution. Therefore, any optimal solution also would require another gas station.

   The running time is O(n) since we at most make one computation/decision at each gas station.
   Rubrics:
   - Greedy algorithm: 6 pts
   - "Stay-ahead argument" of the proof (part a): 8 pts
   - Induction base: 2 pts
   - Induction hypothesis: 3pts
   - Induction step: 3 pts
   - Completing the argument of optimality in the proof (part b): 2 points

2. The array A holds a max-heap. What will be the order of elements in array 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 (8 points)

Solution:
Initial Array:

Index	1	2	3	4	5	6	7	8	9	10
Element	19	17	14	8	7	9	3	2	1	4

After inserting 18:

Index	1	2	3	4	5	6	7	8	9	10	11
Element	19	17	14	8	7	9	3	2	1	4	18

18 is greater than 7 (the element at index 11/2=5), so swap:

Index	1	2	3	4	5	6	7	8	9	10	11
Element	19	17	14	8	18	9	3	2	1	4	7

18 is greater than 17 (the element at index 5/2=2), so swap:

Index	1	2	3	4	5	6	7	8	9	10	11
Element	19	18	14	8	17	9	3	2	1	4	7

Rubric:

- 2 points for showing every swap correctly

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. (Hints: 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 should include a penny so that there is a solution for every value of n. (15+5 points)

Solution:

(a) Denote the coins values as c1 = 1, c2 = 5, c3 = 10, c4 = 25.

i. if n = 0, do nothing but return.
ii. Otherwise, find the largest coin ck , $1 \leq k \leq 4$, such that ck $\leq$ n. Add the coin into the solution coin set S.
iii. Subtract x from n, and repeat the steps 1) and 2) for n ck.

For the proof of optimality, we first prove the following claim: Any optimal solution must take the largest ck, such that ck $\leq$ n .Here we have the following observations for an optimal solution:

i. Must have at most 2 dimes; otherwise we can replace 3 dimes with quarter and nickel.
ii. If 2 dimes, no nickels; otherwise we can replace 2 dimes and 1 nickel with a quarter.
iii. At most 1 nickel; otherwise we can replace 2 nickels with a dime. 4. At most 4 pennies; otherwise can replace 5 pennies with a nickel. Correspondingly, an optimal solution must have
   - Total value of pennies: $\leq$ 4 cents.
   - Total value of pennies and nickels: $\leq 4 + 5 = 9$ cents.
   - Total value of pennies, nickels and dimes: $\leq 2\times10+4 = 24$ cents. Therefore,
   - If $1 \leq n < 5$, the optimal solution must take a penny.
   - If $5 \leq n < 10$, the optimal solution must take a nickel; otherwise, the total value of pennies exceeds 4 cents.
   - If $10 \leq n < 25$, the optimal solution must take a dime; otherwise, the total value of pennies and nickels exceeds 9 cents.
   - If $n \geq 25$, the optimal solution must take a quarter; otherwise, the total value of pennies, nickels and dimes exceeds 24 cents.

Compared with the greedy algorithm and the optimal algorithm, since both algorithms take the largest value coin ck from n cents, then the problem reduces to the coin changing of n - ck cents, which, by induction, is optimally solved by the greedy algorithm.

Rubrics

- Algorithm: 4 pts
- Claim and proof: 3 pts
- Final proof with induction: 3 pts

(b) Coin combinations = 1, 15, 20 cents coins.Consider this example n = 30 cents. According to the greedy algorithm, we need 11 coins: $30 = 20 + 10$1; but the optimal solution is 2 coins $30 = 215$.

4. You are given positions of N Mice and positions of N holes on a 1-dimensional number line. Each hole can accommodate at most 1 mouse. A mouse can stay in place, move one step right from x to $x + 1$, or move one step left from 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. (10 points).

Solution:
Sort the N mice in order of non-decreasing position and the N holes in order of non-decreasing position. Loop from i = $1 \ldots n$ and sum up the absolute value of distances between the i^{th} mouse position and the i^{th} hold position.
More formally:

```
getMinimumMouseMoves(n, [x_mouse], [x_hole])
    int result;
    sort([x_mouse]);
    sort([x_hole]);
    for (i...n) {
        result += | x_mouse[i] - x_hole[i] | ;
    }
    return result;
```

PROOF (Students do not need to submit proof): We can prove this greedy algorithm is optimal using an exchange argument. We define an inversion as follows: Mouse i and Mouse j form an inversion if Mouse i is assigned to hole k and Mouse j is assigned to Hole m with $i < j$ and $m < k$.
We consider 6 cases here:

- $m \leq i$ and $k \leq i$
  In this case, the number of movements in the optimal solution is $d = i-k+j-m$. But, if we interchange and assign Mouse i to Hole m and Mouse j to Hole k then $d' = i-m+j-k$. We observe that both d and d' values are equal and hence we can safely interchange assignments for mouse i and mouse j.
- $m \leq i$ and $i \leq k \leq j$
  In this case, the number of movements in the optimal solution is $d = k-i+j-m$. But, if we interchange and assign Mouse i to Hole m and Mouse j to Hole k then $d' = i-m+j-k$. We observe that $d' \leq d$ (following these steps:
  $i-k+jhttps://www.overleaf.com/download/project/63d20e6d298bf853382b33b7/build/186140654a0-24c75e6d2e9daffb/output/output.pdf?compileGroup = standardclsiserverid = clsi - pre - emp - e2 - c - f - ql60enable_pdf_caching = truepopupDownload = true - m \leq k - i + j - m$
  $i - k \leq k - i$
  $i \leq k$ (Given as the condition)). Hence we can safely interchange assignments for mouse i and mouse j.
- $m \leq i$ and $j \leq k$
  In this case, the number of movements in the optimal solution is $d = k-i+j-m$. But, if we interchange and assign Mouse i to Hole m and Mouse j to Hole k then $d' = i-m+k-j$. We observe that $d' \leq d$ (following these steps:
  $i - m + k - j \leq k - i + j - m$
  $i - j \leq j - i$

$i \leq j$ (Given as the condition)). Hence we can safely interchange assignments for mouse i and mouse j.

- $i \leq m \leq j$ and $i \leq k \leq j$ In this case, the number of movements in the optimal solution is $d = k - i + j - m$. But, if we interchange and assign Mouse i to Hole m and Mouse j to Hole k then $d' = m - i + j - k$. We observe that both d and d' values are equal and hence we can safely interchange assignments for mouse i and mouse j.
- $i \leq m \leq j$ and j In this case, the number of movements in the optimal solution is $d = k - i + j - m$. But, if we interchange and assign Mouse i to Hole m and Mouse j to Hole k then $d' = m - i + k - j$. We observe that $d' \leq d$ (following these steps:
  $m - i + k - j \leq k - i + j - m$
  $m - j \leq j - m$
  $m \leq j$ (Given as the condition)). Hence we can safely interchange assignments for mouse i and mouse j.
- $j \leq m$ and $j \leq k$ In this case, the number of movements in the optimal solution is $d = k - i + m - j$. But, if we interchange and assign Mouse i to Hole m and Mouse j to Hole k then $d' = m - i + k - j$. We observe that both d and d' values are equal and hence we can safely interchange assignments for mouse i and mouse j.

Thus, if we are given any optimal solution that has inversions, we can swap them one by one without affecting the optimality of the solution. When there are no inversions, the solution becomes the same as ours, and hence the total number of moves of our solution ¡= total number of moves by optimal solution.

Rubric:

- 10 points for the correct algorithm.
- −3 points for not taking absolute value.
- −7 points for not sorting the positions of mice and holes correctly (either not sorting in the same direction or not at all)
- −5 if algorithm is not in O(nlogn).

5. Farmer John has N cows (1, 2,. . . , N) who are planning to escape to join the circus. His cows generally lack creativity. The only performance they came up with is the "cow tower". A "cow tower" is a type of stunt in which every cow (except for the bottom one) stands on another cow's back and supports all cows above in a column. The cows are trying to find their position in the tower. Cow I (i = 1, 2,. . . , N) has weight W_i and strength S_i. The "risk value" of cow i failing (R_i) is equal to the total weight of all cows on its back minus Si. We want to design an algorithm to help cows find their positions in the tower such that 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 counter-example. Hint: One of the two solutions is correct and the other is not.
   (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. (15 points total)

   Solution:

   (a) Counterexample: $W_1 = 5$, $S_1 = 20$, $W_2 = 500$, $S_2 = 1$. As per the algorithm, cow 2 will be above cow1. The risk value in this algorithm would be 480 ($500 - 20 = 480$) whereas if we interchange them and put cow 1 above cow 2, we will have a risk value of 4 ($5 - 1 = 4$). Thus, we have a counterexample at hand for the given algorithm.
   (b) The proof is similar to the proof we did for the scheduling problem to minimize maximum lateness. We first define an inversion as a cow i with higher (W_i+S_i) being higher in the tower compared to cow j with lower (W_j+S_j). We can then show that inversions can be removed without increasing the maximum risk value time. We then show that given an optimal solution with inversions, we can remove inversions one by one without affecting the optimality of the solution until the solution turns into our solution.
      - Inversions can be removed without increasing the risk value.
        Remember that if there is an inversion between two items a and b, we can always find two adjacent items somewhere between a and b so that they have an inversion between them. Now we focus on two adjacent cows (one standing on top of the other) who have an inversion between them, e.g. cow i with higher (W_i+S_i) is on top of cow j with lower

(W_j+S_j). Now we show that scheduling cow i before cow j is not going to increase the maximum risk value of the two cows i and j. We do this one cow at a time:

- By moving cow j higher we cannot increase the risk value of cow j
- By moving cow i lower (below cow j) we will increase the risk value of cow j but since cow i has a higher total weight and strength than cow j, the risk value of cow i (after removing the inversion) will not be worse than the risk value for cow j prior to removing the inversion:
- Risk value of cow j before removing the inversion: W_i + (weight of remaining cows above cows i and j) - S_j
- Risk value of cow i after removing the inversion: W_j + (weight of remaining cows above cows i and j) - S_i
- Since W_i+$S_i \geq W_j$+S_j , if we move S_i and S_j to the opposite sides we get: W_i-$S_j \geq W_j$-S_i. (weight of the remaining cows above the two cows i and j does not change). In other words, the risk value of cow j before removing the inversion is higher than that of cow i after removing the inversion

– Since we know that removing inversions will not affect the maximum risk value negatively, if we are given an optimal solution that has any inversions in it, we can remove these inversions one by one without affecting the optimality of the solution. When there are no more inversions, this solution will be the same as ours, i.e. cows sorted from top to bottom in ascending order of weight+strength. So our solution is also optimal.

Rubric:

- 5 points for correct counterexample
- 2 points for attempted proof or invalid counter-example
- 10 points for correct proof
- Partial credit up to 7 points for proof that omits section 2 or is otherwise incomplete.
- Partial credit up to 5 points for proof without a complete mathematical argument in section 1
- Partial credit up to 2 points for incorrect proof 0 points for counter-example

UNGRADED

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

Solution:
Initial Array:

Index	1	2	3	4	5	6	7	8	9	10	11
Element	19	18	14	8	17	9	3	2	1	4	7

After deleting 14 and replacing with last element:

Index	1	2	3	4	5	6	7	8	9	10
Element	19	18	7	8	17	9	3	2	1	4

As 7 is greater than its child, we swap 7 with the largest of its two children that is 9:

Index	1	2	3	4	5	6	7	8	9	10
Element	19	18	9	8	17	7	3	2	1	4

Rubric:

- 3 points for showing that last element is to be replaced at deleted element
- 2 points for every next swap