AOI 2022 Solution Write-Up

Anderson Secondary School - Robotics Club Coding Group

22 and 26 August 2022

Task A: Equality

Authored and prepared by: Kiew Tian Le, Ernest

Subtask 1

Limits: $0 \leq N \leq 10^2, 1 \leq K \leq 10$
Firstly, we must make the observation that the optimal way to distribute the food pieces to all the ducks is to do so as equally as possible, such that the difference between the most-fed and least-fed duck is less than or equal to 1.

It is always possible to do this. We can then create an array of size K to represent the amount of food each ducks receives, and set each of them to N/K, which rounds down in C++.

We can subtract each of the values of the array from a counter that starts at N to tell us how much food we have left. We then distribute 1 piece of food to any duck in any order repeatedly until no more food is left. Ensure you do not give more than 1 extra piece of food to any duck.

Then, we can simply calculate the total jealousy of the ducks as described by the question statement.

We achieve an O(K) solution, which suffices for this subtask. This is an overcomplicated solution however, and subtask 2 is actually easier.

Subtask 2

Limits: No additional restrictions ($0 \leq N \leq 10^{15}$, $1 \leq K \leq 10^{14}$)
For this subtask, our time complexity cannot be O(N) or even O(K). We should try to find a solution in constant time.

Firstly, we note that if N is divisible by K, we are able to distribute an equal amount of food to all ducks, and our answer is 0.

Otherwise, we note that in an optimal distribution when the difference between the most-fed and least-fed duck is 1, the total jealousy corresponds to the number of ducks that are the least well-fed.

Aside from the situation where N is divisible by K, we can compute the number of ducks that are least well-fed with the formula, D as $D = N - (N\%K)$.

Thus, the solution is to print 0 if N is divisible by K, and otherwise, print $N - (N\%K)$.

Task B: GAY's Project

Authored and prepared by: Josiah Huang

Subtask 1

Limits: First string is all vowels, second string is all consonants

Since the first string is all vowels, simply output the first string.

Time complexity: $\mathcal{O}(1)$

Subtask 2

Limits: The first letter of a string is vowel, first letter of another string is consonant

Since the they differ at the first letter, and the first letter of the first string is a vowel, and the first letter of the second string is consonant. Check for the string with a vowel as the first letter and output it.

Time complexity: $\mathcal{O}(1)$

Subtask 3

Limits: No additional restrictions

Loop through the two strings. If the letter at that position of 1st string is a vowel but the letter at that position of the 2nd string is a consonant, output the 2nd string and vice-versa.

Time complexity: $\mathcal{O}(N)$, where N is length of the strings.

Task C: Toilet Disruptions

Authored and prepared by: Hayden How

Subtask 1

$N = 1$, so there is only 1 day left. We can make a variable, called current_time, to keep track of previous time that Doo the dog is avaliable (Basically the ending time of each toilet period + 1 cuz it is inclusive), initially set equal to 0.

Then as we loop through all the toilet periods, we find the time that he can work for which is $S_{i,j}$ - current_time. We keep a running max to find the max of all time periods in between toilet periods which he can work for. Then set current_time equal to the ending time of the toilet session, $E_{i,j} + 1$. (+1 because it is inclusive, we need it to be the period where he CAN work).

After looping through every toilet period, we still have 1 edge case which is the very last time period that he can work at. (A lot of people missed this and died). So u have to check 1440 - current_time to get last time period and print the running max.

Time complexity: $\mathcal{O}(TN)$.

Subtask 2

$T = 1$, meaning there is only 1 time period. I only added this subtask so that there will be less edge cases so that it will be easier to ac.

The same solution from subtask 1 can be extended to account for multiple days. Notice that $S_{i,j}$ and $E_{i,j}$ are called from 00:00 on the i-th day, not the very first day. This means that $S_{i,j}$ and $E_{i,j}$ will not be able to fit into the other days so easily. In order to get all of them to count from 00:00 on the FIRST day, we can increase $S_{i,j}$ and $E_{i,j}$ by $1440 \cdot i$. The same solution from subtask 1 can be trivially applied to this.

Time complexity: $\mathcal{O}(TN)$.

Subtask 3

Literally sane solution from subtask 1 and 2 but with more edge cases you have to be careful about.

Time complexity: $\mathcal{O}(TN)$.

Task D: Embarrassment

Authored and prepared by: Hayden How

Subtask 1

All values of S_i are equal, meaning everyone has the same skill level. This means that everyone has a embarrassment level of 0 no matter what the arrangement is.

So we just print 0.

Time complexity: $\mathcal{O}(1)$.

Subtask 2

N is very small, so this means that we can generate ALL arrangements ($N!$ different arrangements) of the students and find the minimum sum of the embarrassment levels and print it.

This can be done with the help of std::sort and std::next_permutation().

Time complexity: $\mathcal{O}(N \cdot N! + N \log N)$

Subtask 3

I left this out for any sort of bruteforce ($\mathcal{O}(N^2)$ or faster) or unoptimized version of full solution to pass.

Subtask 4

Let's take a look at the array [3, 1, 4, 9]. Index 0, the number 3, we want to minimise its difference with its neighbors. In order to minimise the difference, we can put the first number that is $\leq$ to 3 and $\geq$ to 3 next to 3 (Which is 1 and 4). This applies for all other numbers 1, 4 and 9. Notice that this arrangement of putting the next higher or lower number is just sorting the array.

So the optimal arrangement for all cases is to sort them in ascending or descending order (They produce same results). In the case of [3, 1, 4, 9], it's [1, 3, 4, 9] or [9, 4, 3, 1].

We can just sort the array and find the sum of all embarrassments from there and print it.

Time complexity: $\mathcal{O}(N \log N)$.

Task E: Art of War

Authored and prepared by: Lim Chang Jun

Subtask 1

$A_i \leq 1$, exactly one tank is sufficient to capture the city. Therefore we will have a variable tank_cnt to keep track of the number of tanks left, and another variable last_city to keep track of the last city we can capture. After that, we run a loop from the first city, if the enemy strength is 0 we will continue the iteration, else if the strength is 1 we will deploy on tank, if the tank is already zero we would break the loop, else (tank_cnt–). Throughout the loop last_city would constantly be updated as the largest index. Lastly output the last_city

Time complexity: $\mathcal{O}(N)$.

Subtask 2

A_i is divislbe by 3. Therefore by taking the enemy strength divide by 3, we would get the amount of tank needed to be deployed.

The same solution from subtask 1 can use. The only edge case is to check if you even have enough tanks before you deploy them (tank_cnt - enemy/3 ≥ 0) , else break the loop.

Time complexity: $\mathcal{O}(N)$.

Subtask 3

Same solution as subtask 1, if the enemy have 0 strength then continue the iteration. Else deploy a tank if able to. Lastly output the last city you can capture.

Time complexity: $\mathcal{O}(N)$.

Subtask 4

Realise that we can know the number of tanks needed by dividing enemy strength by 3 and if there's still remainder left we will deploy one additional tank (for example enemy with a strength of 3 would need 1 tank, 4 would need 2 tank, 8 would need 2 + 1 tank). By knowing this, we can use the same solution we did in Subtask 3 and apply this new concept to solve the entire question

Time complexity: $\mathcal{O}(N)$.

Task F: Physics go brrr

Authored and prepared by: Hayden How

Subtask 1

Since $X_i = 10^7$ and $1 \leq A \leq 10^7$, all the safe squares where Mr Nathan can throw the robot will be to the right of the target or on the same x coordinate.

Case 1:

When the squares are to the right, we can simply print NO as they only can move to the right.

Case 2:

When they have the same x-coordinates, we can simply check for all the y coordinates. If the squares have same y-coordinates, it means that we can directly hit the target on this square. If none have same y-coordinates, then we cannot and print NO.

Time complexity: $\mathcal{O}(TS)$.

Subtask 2

$M = 0$, meaning all lines have gradient of 0 and thus will travel in a horizontal line where the y coordinates stay the same. This means that for a robot to be able to hit the target from the square, its x-coordinates must be $\leq A$ and its y-coordinates must be equal to B (Since the y-coordinates won't change).

We can check every single square if $X_i \leq A$ and $Y_i = B$. If so, print YES. Else, print NO.

Time complexity: $\mathcal{O}(TS)$.

Subtask 3

Since A is rather small, we can bruteforce all integer coordinate points on the paths of each robot if it were thrown at a square. For example, the robot will first reach the coordinates of a square, then start moving to the right and the y-coordinates will increase by M (The gradient) as x-coordinates increases by 1.

We can simulate each robot throw (Starting at X_i and slowly increasing up to A) and increase the y-coordinates by M and check what the y-coordinate is when the x-coordinate reaches A. If they are the same, we managed to hit the target. Else, we have not yet.

Be aware of 1 edge case where the squares are to the right of the target ($X_i > A$), Mr Nathan will not be able to hit the target from this square as they only travel to the right.

Time complexity: $\mathcal{O}(TS \cdot (A - X_i))$.

Subtask 4

We can extend the solution from subtask 3. We do not need to check every x-coordinate and its y-coordinate between X_i and A, we only care about the robot's y-coordinate when the x-coordinate is A. We can do some simple math to find this in constant time.

If gradient is M. When x-coordinate increases by 1, y-coordinate increases by M. When x-coordinate increases by 2, y-coordinate increases by $2M$. When x-coordinate increases by 3, y-coordinate increases by $3M$. Therefore, when x-coordinate increases by ΔX, y-coordinate increases by $\Delta X \cdot M$.

The change in x-value is just equal to $A - X_i$ which will be our ΔX. To find change in y-value, it's simply $\Delta X \cdot M$ which is $(A - X_i) \cdot M$. We look at all squares and find the y-coordinate when the x-coordinate is at A and see if it's equal to B.

Be aware of same edge case from subtask 3.

Time complexity: $\mathcal{O}(TS)$.

Task G: Duck Snacks

Authored and prepared by: Kiew Tian Le, Ernest

Subtask 1

Limits: $B >= N$
This is a free subtask. As the protection range is greater than the size of the entire field, we can deduce that all piles can be protected at the same time. As such, simply print out the sum of the values of all piles.

Subtask 2

Limits: X_i is presorted by distance from the left end in increasing order, $1 \leq N \leq 2 \cdot 10^3$
As N is small enough to allow an $O(N^2)$ solution to pass, we should look for such a solution. For this subtask, we will have an $O(NK)$ solution, which will pass as $K \leq N$.

For this subtask, a pure bruteforce solution will pass. Run every possible range, 0 to B-1, 1 to B, and so on until either N-B+1 to N or N to N+B-1. (Both work, but the latter handles unnecessary cases.)

We can simply run through the entire range of K, checking if its x-position falls within our current range of protection, and adding to a counter if it does. We keep track of the maximum value of the counter, reset the counter to 0, and repeat for all other ranges.

The maximum value of this counter throughout the whole process is our final answer.

Subtask 3 and 4

Limits of ST3: X_i is presorted by distance from the left end in increasing order, $1 \leq N \leq 2 \cdot 10^6$
Limits of ST4: $1 \leq N \leq 2 \cdot 10^6$

Subtask 3 and 4 are very similar, with the exception that Subtask 4's input is not presorted, leading to possible issues with other alternative solutions. There are two main techniques that both work.

We can simply form an array of size N, and when a pile exists at X_i, we can add its value to the element at index X_i. This will be used for both techniques, transforming it into an array problem.

Technique 1:
Sliding window is a technique we can use to obtain the maximum value of elements in a range. We start by fully calculating the sum of the first B elements, then keeping a counter to keep track of the maximum sums of the ranges. We "slide" the window to the right, subtracting the first element in the range and adding the next element after the end of the range.

Technique 2:
We can create another array to store the prefix sum. See https://usaco.guide/silver/prefix-sums?lang=cpp.
We can then perform up to N different range sum queries for each range, taking the maximum value as our answer.

Both techniques result in an O(N) solution.

Subtask 5

Limits: No additional restrictions. ($1 \leq N \leq 10^{15}$, $1 \leq K \leq 10^6$)
With N being such a massive number, O(N) solutions will fail. We should seek out an O(K) time complexity solution.

For this, we can modify our sliding window solution from before, turning it into a two-pointer technique. The two-pointer technique is very similar to the sliding window, with the exception that the window size defined by the two pointers will not be constant. See https://usaco.guide/silver/two-pointers?lang=cpp.

We push the right pointer rightwards if the difference between our first element of the window and new element's x-position is strictly less than B. Otherwise, we push the left pointer rightwards.

If we keep track of the maximum sum of the range defined by the two pointers, we get the final solution to this task.

Task H: What The Quack

Authored and prepared by: Kiew Tian Le, Ernest

Note:

The time complexity of the while loop given in the question is actually O(B), where B is the number of bits in x with leading zeroes removed. This is extra information that is not essential to solving the question.

Subtask 1

Limits: $K_i = 0$
This is a free subtask. Print 0.

Subtask 2

Limits: $0 \leq K_i \leq 10^2$, $0 \leq Q \leq 10^2$
There are many different solutions that all pass due to the extremely relaxed time complexity requirements. Here is just one sample solution:

For each query, we try all $f(x)$ for all values of x from 0 to N, and stop when $f(x) \geq K_i$.
Alternatively, basic linear search on a precomputed array of values of $f(x)$ also works (and is slightly more efficient, but unable to solve further tasks)

Subtask 3

Limits: $M = 2 \cdot 10^5$

M here stands for the maximum value of the answer. See the last sentence of the task description, just above Input Format. As such, an O(MQ) solution will fail as it exceeds the time limit.
Instead, we can extend Subtask 2's alternative solution with a precomputed array of values, but instead of linear search, we use binary search with lower_bound on vector or array.

Subtask 4

Limits: No additional restrictions ($M = 10^8$)

M here stands for the maximum value of the answer. See the last sentence of the task description, just above Input Format. With subtask 4's value of M being so high, an O(M) solution will barely scrape past, but due to the added time complexity of computing the values, we cannot use the same solution as subtask 3.
Instead of precomputing the array, we must now self-implement binary search without any container. This yields the final solution to Task H.

Special Graph Theory Task: Nice Ducks

Authored by Josiah Huang, prepared by Kiew Tian Le, Ernest

Subtask 1

Limits: As all the houses have niceness level of 0, all the neighbourhoods will have total niceness level of 0. Hence, output 0.

Time complexity: $\mathcal{O}(1)$

Subtask 2

Limits: As all the houses have niceness level of 1, the total niceness level of a neighbourhood is equal to the number of houses in that neighbourhood. Hence, using dfs / bfs, find the size of each neighbourhood and output the largest size of all the neighbourhood.

Time complexity: $\mathcal{O}(N)$, where N is the number of houses

Subtask 3

Limits: No further restrictions

This question is similar to subgraphs (codebreaker.xyz), the only difference being that when you loop through the subgraph (which represents the neighburhood) through dfs / bfs, you will need to add a counter to count the total niceness level of the current neighbourhood. An unordered set or boolean array will also needed to prevent double counting.

Time complexity: $\mathcal{O}(N)$, where N is number of houses.

Special DP Task: Royal Guards

Authored and prepared by: Kiew Tian Le, Ernest

Subtask 1

Limits: $E_i = -(10^3), N \leq 10$
This is also a free subtask, hidden in the final question :) Simply print 0 as none of the guards are ever worth taking.

Subtask 2

Limits: $N \leq 10$
This is a very tough bruteforce question. Despite looking extremely simple, it is really not easy. The expected time complexity is $O(2^N \cdot N)$. (I think the other programming heads might not be able to solve either lol)

We can create 11 different vectors of 1s and 0s, each with a different number of 1s. We then iterate through each permutation of each vector, with next_permutation(). We'll define 1 as taking the guard, and 0 as not taking the guard. We need $O(N)$ time to compute the value of each permutation, and the total number of permutations throughout all 11 vectors is $O(2^N)$. Thus, this solution takes $O(2^N \cdot N)$ time.

Subtask 3

Limits: $N \leq 10^2$
This subtask is reserved for $O(N^3)$ DP (Dynamic Programming) solutions. See subtask 4 for a better solution.

Subtask 4

Limits: $N \leq 2 * 10^3$
For this subtask, we must make use of Dynamic Programming. We'll define the states and transition as such:
Let dp_x be the maximum value of all permutations of guard selection such that the following conditions hold:
1. The guard of index x must be chosen.
2. No guards of index greater than x can be chosen.
In that case, we can then define the transition between states as follows:
$dp_x = \max(dp_i + (x-i)^2/k)$ over all values of i such that $i < x$.

After computing the full DP table, we can produce our final answer by taking the maximum value between 0 and all values of our DP table.

Since our transition occurs in O(N) time, and we have O(N) states, our final time complexity is $O(N^2)$.

Subtask 5 (NOT A SERIOUS SUBTASK)

Limits: $N \leq 10^6$
Turns out, we actually don't know if there is a possible solution for this. I (and maybe Hayden) do know of a solution for the case where $K = 1$ however, and if you're willing to lose your mind over this problem feel free to contact me (Ernest) with your solutions. I will not release the solution here or anywhere else. Good luck, and thanks for reading the editorial to this point!

Setters' comments and feedback

Ernest:

The questions I set seem to have been a bit too hard, maybe I'll tone it down next time. Aside from that, I tried to test common techniques that are helpful, but it seems that most do not have a good understanding of this yet. Do try working on some of them, as they can be quite useful in some situations! I was also surprised that not many seemed to be able to grab the points of the subtasks that we set to be free. I recommend you fully digest the meaning of each subtask for each question when out of options to continue. Good luck for all future programming contests!

Hayden:

Maybe I added too many edge cases to my questions. I realised many people don't test their code even on the sample testcases and miss out such edge cases or even random things like printing new lines or spaces. **PLEASE** test your code, it is very important. Also plz learn how to indent it's very cancer to look at code without proper indentation.

Chang Jun:

I made a bit of a typo while setting the question. But still, im glad to see that a lot of the members were able to solve the question. The only thing to point out is that the data will automatically round down when we are doing division, therefore it is recommended to use float for division. It is the same for the function ceil which would require ceil. Lastly, take note that subtask is very important as they allow you to grab points (if you cannot fully ac) or even give you hint on how to solve the full question.