mBIT Standard Editorial

June 2020

These are the solutions to the standard problems. Each answer consists of a brief explanation of the solution followed by a link to our code in each language. Keep in mind that there are multiple ways to do each problem, and the given code may employ a different algorithm than the explanation.

Contents

§1 Ice Cream Truck

It is useful to buy cones in bundles if and only if the unit price of cones in bundles in lower than in individual units (i.e. $B < 5A$). In that case, buy as many cones in bundles as possible. Then, the remaining cones (or all the cones, if no bundles are used) should be bought at the A price.

Time Complexity: $O(1)$

Problem: Maxwell Zhang
Editorial: Claire Zhang
Solutions: C++, Java, Python

§2 Pirating Parrots

Loop through each character in the string and track the x- and y- coordinates of the robot after each move. Let the robot's position after the moves be (x_N, y_N). To reach (X, Y) with the minimum number of commands, clearly at least $|X - x_N|$ moves are needed to get the robot to the right x-coordinate and at least $|Y - y_N|$ moves are needed for the y-coordinate. This minimum can always be achieved, so a total of $|X - x_N| + |Y - y_N|$ commands are needed.

Time Complexity: $O(N)$

Problem: Aaron Mei
Editorial: Claire Zhang
Solutions: C++, Java, Python

§3 Monkey Signs

Loop through each character of the string, keeping count of the number of each letter (watch out for capitalization and spaces); we will call the counts *cnt*. While doing so, keep track of the letter (other than M) with the highest count; call this *maxCnt*. The maximum number of M's will be achieved by changing the most frequent letter other than M to M's, so the answer is $cnt['M'] + maxCnt$.

Time Complexity: $O(N)$

Problem: Gabriel Wu
Editorial: Claire Zhang
Solutions: C++, Java, Python

§4 Happy Bunnies

Loop from $i = A$ to $i = B$. At each iteration, keep a variable $curSum$ representing the amount the current i contributes to the overall sum. Initially, $curSum = i$. To add the digits, keeping adding the last digit of the current number to $curSum$ and removing that digit until the number is 0. If any of these digits are 7, skip this iteration completely. Otherwise, add $curSum$ to the overall sum, and output the overall sum.

Time Complexity: $O((B - A) \log B)$

Problem: Maxwell Zhang
Editorial: Claire Zhang
Solutions: C++, Java, Python

§5 Musical Bowings

Find the index of the first letter other than `B`; call this index i and the letter at that index L. If all the letters are `B`, define i as the last index and we may arbitrarily set L to `D`. Starting from i and moving forwards, keep alternating between setting the current letter to L and the opposite of L. If an already-marked bowing is encountered at any point, the current bowing must be set to that letter.* Run this algorithm from i in the backwards direction as well and the problem is complete.

* *Proof of correctness:* Note that if it is possible to mark all the bowings without any hooked bowings, this solution will find that marking. If not, the marking for the first new letter will not affect the number of hooked bowings present (ex. it is equally valid to mark `U B B B D` as both `U D U D D` and `U U D U D`), so this solution will still find one of the optimal markings.

Time Complexity: $O(N)$

Problem: Maxwell Zhang
Editorial: Jeffrey Tong
Solutions: C++, Java, Python

§6 Snake Moves

As the snake moves, store its current position x_{curr}, y_{curr}. For every tile (x_i, y_i) it must move to, it can reach that tile in 1 move if any of these conditions are satisfied: $x = x_{curr}$ (move horizontally), $y = y_{curr}$ (move vertically), or $|x - x_{curr}| = |y - y_{curr}|$ (move diagonally). Otherwise, two moves are needed (this can always be done by first moving to the right x-coordinate and then to the right y-coordinate.) Update x_{curr}, y_{curr}, and the total number of moves.

Time Complexity: $O(N)$

Problem: Jeffrey Tong
Editorial: Claire Zhang
Solutions: C++, Java, Python

§7 Egg Interception

Since Farmer Fred is required to catch all the eggs, he catches eggs in the order of the time they start falling. Sort the times and loop through the indices of the sorted times; at each iteration, add the distance needed to travel to the next egg to the final sum.

Time Complexity: $O(N \log N)$

Problem: Ayush Varshney
Editorial: Claire Zhang
Solutions: C++, Java, Python

§8 Icebergs

We can keep track of which penguins are still standing with a boolean array (call it isStanding), and loop through each penguin. In each iteration, if the current penguin is still alive, loop through each penguin again and eliminate the first penguin with the lowest distance. Update the isStanding array as you go. Once the end is reached, loop back to the beginning. Once only one penguin is standing, output its index.*

* *Proof of time complexity:* Note that this brute-force solution always works because since at least one penguin is eliminated in each cycle through all the penguins, there will be at most N cycles, giving a time complexity of $O(N^2)$ for looping through the turns. We will only execute an inner loop to find the nearest penguin if the current penguin is still standing, which will happen exactly $N-1$ times, again giving a time complexity of $O(N^2)$. The total time complexity is thus $O(N^2)$.

Time Complexity: $O(N^2)$

Problem: Gabriel Wu
Editorial: Jeffrey Tong
Solutions: C++, Java, Python

§9 Bracelets

Because rotating the first bracelet by N beads will return it to its original orientation, we only need to consider offsets of k beads where $0 \leq k \leq N-1$.

For convenience, subtract 1 from each number so that the numbers are from 0 to $N-1$. For each number, we now compute the offset by which the first bracelet should be rotated so that the number is in the same position in both bracelets. To do this, construct arrays a and b so that for each $0 \leq i \leq N-1$, $a[i]$ and $b[i]$ are the indices of i in the first and second bracelets, respectively. Since we want $0 \leq k \leq N-1$, the offset for each i is then $(b[i]-a[i]+N)\%N$ (the nonnegative value of $b[i]-a[i] \bmod N$).

The maximum number of matching beads is then the frequency of the most common offset between the two bracelets.

Time Complexity: $O(N)$

Problem: Gabriel Wu
Editorial: Jeffrey Tong
Solutions: C++, Java, Python

§10 Zoo Tour

The key idea to this problem is that there are two ways to go around the circular neighborhood to get from one habitat to another. Thus, we can use prefix sums to solve this problem. We will index the habitats from 0 to $N-1$. Before taking the queries, preprocess the data by constructing a prefix array pre such that for $0 \leq i < N$, $pre[i]$ is the distance from habitat 0 to habitat i traveling clockwise and $pre[N]$ is the length of the entire loop, using all the paths.

For each query, arrange u and v such that $u < v$. Then, if the veterinarian travels clockwise, the distance traveled (call this d) is $pre[v] - pre[u]$. If he travels counterclockwise, he will travel through every path except the ones in the clockwise route, so the distance will be $pre[N] - d$. The answer to that query is the minimum of those two distances.

Time Complexity: $O(N+Q)$

Problem: Aaron Mei
Editorial: Aaron Mei
Solutions: C++, Java, Python

§11 Raging Rhinos

We see that each interaction between rhinos uses exactly two rhinos. We may choose to look at each interaction from the perspective of either rhino, however it turns out that it is advantageous to only observe these interactions through either the right or left initial rhino for all interactions. For the sake of explanation, we shall look at the right initial rhino. This crucial observation reveals that we are able to simulate these interactions using a stack approach iterating over the given list from left to right.

In this stack, we apply casework before inserting the current rhino. If the rhino on the top of this stack faces right and our current rhino faces left, we have an interaction. While our current rhino can defeat the rhino at the top of the stack, we decrease our rhino's stamina and remove the rhino at the top of the stack. Once the current rhino is unable to defeat more rhinos (the stack is empty/the rhino at the top of the stack is facing left or the rhino at the top of the stack has more stamina than the current rhino) we either insert or do not insert the current rhino with its remaining stamina into the stack. After iterating over the whole list using this process, we can simply output the resultant stack.

Time Complexity: $O(N)$

Problem: Maxwell Zhang
Editorial: Ayush Varshney
Solutions: C++, Java, Python

§12 Hen Hackers

Since we are given the information that all characters that appear in the string only appear once, we may simply query each of the 62 possible characters one at a time to determine the characters the password consists of.

All we need now is the order of these characters within the password. Though this may seem strange at first, we can simply apply a sort operation on the list of known characters within the string. This requires a custom comparator: a function that determines whether one character should go in front of or behind another. In this case, the comparator would be a function that sends queries of two characters to the grader. Depending on the output from the grader, the comparator returns the respective value for the sorting algorithm. For example, if you want to know whether 'a' comes before or after 'b' in the password, you simply query 'ab'. After all the characters are sorted, we query the correctly ordered string of characters and end the program.

Additionally, it is important to remember that the password size could be 1 or 2 characters long, so any time we are sorting with the comparator and receive a response of 'C' instead of 'Y' or 'N', we must immediately end the program.

Query Complexity: $O(N \log N)$, where N is the password length.

Problem: Gabriel Wu
Editorial: Ayush Varshney
Solutions: C++, Java, Python