299. Bulls and Cows

Hash Table

Problem Description

Medium

The "Bulls and Cows" game is a brain teaser where one player, the codemaker, chooses a secret number, and another player, the codebreaker, tries to guess the number. The guess is made up of digits, and for each guess, the codemaker must provide a hint in two parts:

• The number of "cows", which are digits that are present in the secret number but are not in the correct position. Our goal is to create a function

The number of "bulls", which are digits that are correctly placed in the guess.

Counting

String

that takes the secret number and the guess as inputs and returns the hint according to the rules of the game. The hint should be a string formatted as "xAyB", where x represents the number of bulls and y represents the number of cows.

Intuition To solve the "Bulls and Cows" problem, we need to compare the secret and the guess strings character by character to count the bulls. A bull is when both have the same digit at the same index. When we encounter a bull, we increment our bulls count by one.

For cows, we need to consider the digits that are present in both secret and guess but are not bulls. To manage this, we keep two separate counts (arrays) of the digits (0 through 9) that appear in secret and guess but are not bulls. Once the bulls' count is complete, we can iterate over the counts array to calculate the cows by finding the minimum count of

each digit between the secret and the guess. The idea is that a cow can only happen if a digit is present in both secret and guess; the number of possible cows for a particular digit is the smaller count of that digit in both strings (since a digit can't be moved around more times than it appears). The overall intuition behind the solution is to:

2. Count the non-bull digits for both strings. 3. The cows count is the sum of the minimum counts of each digit between secret and guess.

1. Identify and count the bulls as we iterate through the strings.

guess at index i. This accounts for potential cows.

The implementation of the solution in Python involves iterating over the secret and guess strings and using two auxiliary arrays to keep track of non-bulls digits.

cows (y).

The Solution class has a method getHint with two parameters: secret and guess. Let's walk through the steps of the method:

Two lists, cnt1 and cnt2, each of size 10 (to account for digits 0-9), are initialized to count the occurrences of each digit in

A for loop goes through each index i of the secret and guess strings. If the characters at the current index match, we

We initialize x and y as integers to zero, where x will count the number of bulls and y will count the number of cows.

secret and guess, respectively, excluding those that are already bulls.

- increment the x counter by one, indicating a bull has been found. If the digits do not match (no bull is found), we increment the counts in cnt1 and cnt2 corresponding to the digit in secret and
- After we have completed the iteration for bulls, we loop through the digits from 0 to 9 to calculate the number of cows. For each digit, we take the minimum count found in cnt1 and cnt2. This minimum count represents how many times a non-bull

digit in guess could potentially be a bull if rearranged, hence a cow. We sum up these minimums to get the total number of

followed by 'B'. This is done using a formatted string (an f-string) f'{x}A{y}B' which is then returned by the method. The algorithm is efficient because it requires only a single pass to count bulls, and a second pass through a fixed-size list of

digits to count cows, making the time complexity linear in terms of the length of the secret string. The use of two counting arrays

Finally, we format the output as a string that shows the number of bulls (x) followed by 'A' and the number of cows (y)

also efficiently handles situations where digits appear multiple times in the secret and guess. Here's part of the code that handles the counting of bulls and updating the counts: for i in range(len(secret)):

x += 1else: cnt1[int(secret[i])] += 1 cnt2[int(guess[i])] += 1

y += min(cnt1[i], cnt2[i])

basic concepts of iteration and frequency counting.

to 9 in secret and guess, excluding bulls.

And here's the code for calculating cows from the counts:

```
The final result is formatted and returned as such:
return f'{x}A{y}B'
```

for i in range(10):

Example Walkthrough

if secret[i] == guess[i]:

"1807" and the codebreaker makes a guess "7810". We'll go through each step of the method to determine the number of bulls and cows.

Initialize two lists, cnt1 and cnt2, each of size 10 and filled with zeros. These lists will count the occurrences of digits from 0

Let's walk through a small example to illustrate the solution approach. Assume the secret number chosen by the codemaker is

This approach is a simplistic yet effective way of solving the "Bulls and Cows" game problem systematically and efficiently using

Iterate through each index i of secret and guess: 3.

• At index 2, secret has 0 and guess has 1, these are not bulls, so we increase cnt1[0] by 1 and cnt2[1] by 1. • At index 3, secret has 7 and guess has 0, these are not bulls, so we increase cnt1[7] by 1 and cnt2[0] by 1. After looping through the strings, we now calculate the cows by iterating through the range of digits 0 to 9:

codebreaker would then be "1 bull, 2 cows."

Other digits cnt1 and cnt2 are 0, they don't contribute to y.

For 0, cnt1[0] is 1 and cnt2[0] is 1, so y increases by min(1, 1) which is 1. For 1, cnt1[1] is 1 and cnt2[1] is 1, so y increases by min(1, 1) which is 1. For 7, cnt1[7] is 1 and cnt2[7] is 0, so y does not increase because min(1, 0) is 0.

effectively computes the correct counts. This follows directly from the insight that a cow occurs only if a digit is present in both

secret and guess, regardless of its place, excluding the bulls that were identified earlier in the process. The game hints for the

This example demonstrates the effectiveness of the algorithm. Despite having the right digits in the wrong places (7 and 0), it

Iterate over both the secret and the guess strings

bulls += 1 # Increment bulls counter

Check if the current digits in both strings are the same (bulls)

for index in range(len(secret)):

if secret[index] == guess[index]:

Iterate over the digits from 0 to 9

for (int i = 0; i < 10; ++i) {

cows += Math.min(secretCount[i], guessCount[i]);

return String.format("%dA%dB", bulls, cows);

if (secretDigit == guessDigit) {

++countSecret[secretDigit];

// For digits that don't match, calculate cows

cows += min(countSecret[i], countGuess[i]);

// Return the result in the format "xBulls and yCows"

return to string(bulls) + "A" + to string(cows) + "B";

++countGuess[guessDigit];

++bulls;

for (int i = 0; i < 10; ++i) {

} else {

// Return the formatted string with bulls ("A") and cows ("B")

By summing all the minimums, we calculate the y as 2 (for the digits 0 and 1).

At index 1, secret has 8 and guess has 8, this is a bull, so we increase x by 1.

Initialize variables x and y to zero, where x will count bulls, and y will count cows.

• At index 0, secret has 1 and guess has 7, these are not bulls, so we increase cnt1[1] by 1 and cnt2[7] by 1.

Solution Implementation

Number of cows is the minimum of the same digit's count in both the secret and the guess

Finally, the x count is 1 (for the digit 8) and the y count is 2, so the formatted string output becomes "1A2B".

def getHint(self, secret: str, guess: str) -> str: # Initialize the counters for bulls ('A') and cows ('B') bulls = cows = 0# Initialize the counters for the numbers that are not bulls $secret_count = [0] * 10$ guess_count = [0] * 10

else: # If not a bull, increment the counts for the unmatched digits secret_count[int(secret[index])] += 1 guess_count[int(guess[index])] += 1

for i in range(10):

Python

class Solution:

```
cows += min(secret_count[i], guess_count[i])
       # Return the formatted hint string
        return f'{bulls}A{cows}B'
Java
class Solution {
    public String getHint(String secret, String guess) {
       // Initial count of bulls ("A") and cows ("B")
        int bulls = 0, cows = 0;
       // Arrays to keep count of numbers not matched (bulls) in secret and guess
        int[] secretCount = new int[10];
        int[] guessCount = new int[10];
       // Iterate through each character in secret and guess strings
        for (int i = 0; i < secret.length(); ++i) {</pre>
            int secretDigit = secret.charAt(i) - '0'; // Convert char to int
            int guessDigit = guess.charAt(i) - '0'; // Convert char to int
           // If digits match at the same index, it's a bull ("A")
            if (secretDigit == guessDigit) {
                ++bulls;
           } else {
                // If not a bull, tally in corresponding count arrays
                ++secretCount[secretDigit];
                ++guessCount[guessDigit];
```

// Calculate cows ("B") by finding the minimum count of each digit in both secret and guess that were not bulls

class Solution { public: string getHint(string secret, string guess) { // Initialize bull and cow counters int bulls = 0, cows = 0; // Arrays to count the occurrence of each digit in secret and guess vector<int> countSecret(10, 0); vector<int> countGuess(10, 0); // Loop through the strings to find bulls and to count numbers for (int i = 0; i < secret.length(); ++i) {</pre> int secretDigit = secret[i] - '0', guessDigit = guess[i] - '0'; // If digits match, increment bulls

```
TypeScript
// Function to calculate the hint based on the secret and guess strings
function getHint(secret: string, guess: string): string {
   // Initialize bull and cow counters
    let bulls: number = 0;
    let cows: number = 0;
   // Arrays to count the occurrence of each digit in secret and guess
    let countSecret: number[] = new Array(10).fill(0);
    let countGuess: number[] = new Array(10).fill(0);
   // Loop through the strings to find bulls and to count digits
   for (let i = 0; i < secret.length; i++) {</pre>
        let secretDigit = parseInt(secret[i], 10);
        let guessDigit = parseInt(guess[i], 10);
       // If digits match, increment bulls
       if (secretDigit === guessDigit) {
           bulls++;
       } else {
           // Otherwise, increment the count of the digit for each string
```

// Otherwise, increment the count of the digit for each string

// The number of cows is the minimum of the same digit in both strings

```
countSecret[secretDigit]++;
              countGuess[guessDigit]++;
      // For digits that don't match, calculate cows
      for (let i = 0; i < 10; i++) {
          // The number of cows is the minimum of the same digit in both strings
          cows += Math.min(countSecret[i], countGuess[i]);
      // Return the result in the format "xBulls and yCows"
      return `${bulls}A${cows}B`;
class Solution:
   def getHint(self, secret: str, guess: str) -> str:
       # Initialize the counters for bulls ('A') and cows ('B')
       bulls = cows = 0
       # Initialize the counters for the numbers that are not bulls
        secret_count = [0] * 10
       guess_count = [0] * 10
       # Iterate over both the secret and the guess strings
        for index in range(len(secret)):
           # Check if the current digits in both strings are the same (bulls)
            if secret[index] == guess[index]:
                bulls += 1 # Increment bulls counter
           else:
               # If not a bull, increment the counts for the unmatched digits
               secret_count[int(secret[index])] += 1
               guess_count[int(guess[index])] += 1
       # Iterate over the digits from 0 to 9
        for i in range(10):
            # Number of cows is the minimum of the same digit's count in both the secret and the guess
            cows += min(secret_count[i], guess_count[i])
```

The provided code involves iterating over the length of the secret and guess strings exactly once. The for loop runs for len(secret) times, and the operations inside the loop have a constant time complexity. After that, there is another loop that iterates a fixed number of times (10), as the possible digits are 0-9 for any guessing game. This loop also performs operations

Time Complexity

Return the formatted hint string

return f'{bulls}A{cows}B'

Time and Space Complexity

with a constant time complexity.

Therefore, the time complexity is determined by the length of the secret string. Let's denote the length of the secret string as n. The time complexity of the code is O(n) + O(10), which simplifies to O(n).

Space Complexity Two arrays, cnt1 and cnt2, each with a size of 10, are used to store the counts of digits from 0 to 9 in secret and guess,

respectively. The size of these arrays is constant and does not depend on the input size. No additional space that scales with the

size of the input is being used. Thus, the space complexity of the code is 0(1) for the constant space used by the counts of the

digits.