Problem 1

1. Possible Solutions for This Problem

a. Using Python's `Counter` from Collections

The `Counter` class in Python's `collections` module can automatically count the occurrences of each character in the string. This approach simplifies the code and can be more efficient for handling large datasets due to its optimized internal implementation.

b. Using Regular Expressions

Regular expressions can be used to find all occurrences of 'A' and 'B' in the string. This method can be more complex but useful if the input validation or processing of characters needs more flexibility and complexity.

2. Solution Explanation of This Program

Input Handling: The program starts by reading a string input from the user.

Initialization of Counters: Two variables, `a` and `b`, are initialized to zero. These serve as counters for the occurrences of 'A' and 'B' respectively.

Loop through Characters: The program iterates over each character in the string. If the character is 'A', the counter `a` is incremented. Similarly, if it's 'B', the counter `b` is incremented.

Decision Logic: After counting, the program compares the counters:

- If `a` is greater than `b`, it prints "A wins!".
- If `b` is greater than `a`, it prints "B wins!".
- If the counts are equal or both are zero, it prints "Draw!".

3. Advantages of the Chosen Solution

Efficiency: For the given problem, where the input is expected to be a relatively short string, the loop and conditional approach is efficient with a linear time complexity (O(n)), where n is the length of the string.

Directness: This approach directly addresses the problem without the overhead of additional libraries or more complex data structures, making it a good fit for lightweight applications or scripts where additional dependencies are undesirable.

Problem 2

1. Possible Solutions for This Problem

a. Pre-computation and Caching

For scenarios where input values might repeat or calculations are expensive, pre-computing results for possible input ranges and storing them in a lookup table could speed up the process. This technique trades memory for faster runtime performance.

b. Optimizing Mathematical Operations

Analyzing and potentially simplifying the mathematical expression to reduce the computational complexity or the number of function calls. For example, rearranging algebraic terms or using alternative mathematical identities could yield efficiency gains.

2. Solution Explanation of This Program

- Input Reading: The program first reads an integer `T`, which represents the number of test cases or input sets to process.
- Vector Initialization: A vector `result` is initialized to store the double precision results for each test case.
- Loop Over Test Cases: For each test case, the program reads three integers `r1`, `r2`, and `d`. It then calculates a double precision result using a complex formula involving cube roots and powers. The formula is calculated based on some arithmetic calculations of equations.
- Output Results: After processing all test cases, the program iterates over the `result` vector and prints each result with a precision of 10 decimal places.

3. Advantages of the Chosen Solution

- Precision and Accuracy: By utilizing `double` for calculations and controlling the output precision to 10 decimal places, the program ensures high numerical accuracy, which is crucial for scientific and engineering calculations.
- Performance: The use of direct mathematical computations with optimized C++ library functions (`cbrt` and `pow`) ensures that the solution is both efficient and fast for the given problem size.

Problem 3:

1. Possible Solutions for This Problem

a. Breadth-First Search (BFS) and Dynamic Programming

Another potential method could involve using BFS to level-order traverse the tree and dynamic programming to maintain and update state information for each node.

b. Segment Tree or Fenwick Tree (Binary Indexed Tree)

For certain tree problems where range queries and updates are frequent, segment trees or Fenwick trees can be used for efficient computation, although this might be more complex and less intuitive compared to DFS for this specific problem type.

2. Solution Explanation of This Program

- Initialization: Vectors and arrays are initialized to store node colors (`color`), answers (`ans`), and counts of black nodes below each node (`black_below`).

- DFS Traversal 1 (`dfs_1`):

- For each node, it counts the black nodes in its subtree.
- Accumulates a partial answer for `ans[1]` based on the number of black nodes below each node and the weight of the edge connecting it to its parent.

- DFS Traversal 2 (`dfs_2`):

- Adjusts the answers calculated in `dfs_1` for each node based on its parent's answer, thus propagating the influence of the entire tree's structure to each node.

- Corrects for any negative values due to modulo operations.

- Tree Construction: The tree is constructed from input where each edge is bidirectional, with nodes connected by weighted edges.

- Final Output: Each node' s calculated answer is printed, reflecting the tree' s structure and node properties.

3. Advantages of the Chosen Solution

- Efficiency: Each DFS runs in O(n) time complexity, making the total complexity for this solution O(n), which is optimal for tree operations involving all nodes.

- Modularity: The separation into two distinct DFS functions allows for clear logical separation of tasks (counting and adjusting), which aids in debugging and understanding the code.

- Flexibility in Update: The method allows for efficient updates to the answer if the tree structure or node properties change minimally, requiring only localized recalculations.

Some notes about this program:

when I fitst do this assignment, I implemented a single DFS, it acts well in the 1-6 testcases, but fails in 7-10. This is mainly because it has a time complexity of O(n^2) within the traversal of a DFS and then within it, there is another for loop. Actually I can' t figure out this 2-DFS approach before getting the hint from BB.

After finishing this assignment, I talked to my friends about this assignment. Then I found out there is also another method to do this assignment, which is using the Chain Forward Star, which is another method to store a graph that are not taught in class.