Problem Set Editorial

YCPC by Phitron | 2nd Round : Contest 1 | 2023

Problem A: Can You Multiply

Author: Rahat Khan Pathan
Approach: The key to solving this problem efficiently is to recognize that you don't need to calculate the full product A*B. Instead, you can focus on the last two digits of the product, which are determined by the last two digits of A and B so that the product of A*B doesn't overflow integer.. As you need to remove the leading zero, that means 05 will be 5 which will make the question more easier to apply.
Comment: This was the easiest problem on the problem set.

C++ Solution: Can You Multiply - C++
Python Solution: Can You Multiply - Python
Java Solution: Can You Multiply - Java

Problem B: Jingle Bells

Author: Rahat Khan Pathan
Approach: This was an implementation and observation problem. You need to observe that the number of lines for stars starts from 6 and grows for the next odd number by one. Also the number of stars in each line starts from one and increases by two each line. The next observation is the number of lines in the root of the tree is fixed which is 5. But the number of stars in each line is equal to the input.

C++ Solution: Jingle Bells - C++
Python Solution: Jingle Bells - Python
Java Solution: Jingle Bells - Java

Problem C: Marbles

Author: Rahat Khan Pathan
Approach: To solve this problem efficiently, we can use the concept of a **Difference Array**. A difference array stores the difference between consecutive elements of the original array. In this problem, the difference array will allow us to handle the updates for each query without modifying the original array directly. Here's a step-by-step approach:

1. Read the input values: N (the number of baskets), the initial number of marbles in each basket, Q (the number of queries), and for each query, read A (starting basket index), B (ending basket index), and X (number of marbles to add).
2. Initialize two arrays: baskets to store the initial number of marbles in each basket and diffArray to store the differences between consecutive baskets. diffArray should be of size N+1, initialised with zeros.
3. For each query (A, B, X), update the diffArray as follows:
4. Add X to the A-th element of the diffArray.
5. Subtract X from the (B+1)-th element of the diffArray. This ensures that the update is cancelled out after the ending basket.
6. Compute the cumulative sum of the diffArray to get the final values for each basket. Initialise a variable cumulative_sum to 0.
7. For each index i (0 to N-1):
    a. Add diffArray[i] to cumulative_sum.
    b. Update the number of marbles in the i-th basket as baskets[i] += cumulative_sum.
    c. Print the updated values of each basket.

C++ Solution: Marbles - C++
Python Solution: Marbles - Python
Java Solution: Marbles - Java

Problem D: Tourists

Author: Rahat Khan Pathan
Approach: To solve this problem efficiently, we can use a **Binary Search** approach to find the largest minimum distance between any two rooms' floors. Here are the steps:

1. Read the input values: T (the number of test cases), N (the number of available rooms), floor numbers of available rooms, and C (the number of tourists).
2. Sort the floor numbers of the available rooms in ascending order.
3. Perform a binary search to find the largest minimum distance that allows at least C tourists to be accommodated with privacy. Initialise the left pointer to 0 and the right pointer to the maximum possible distance between the first and last room.
4. In each binary search iteration, calculate the midpoint as (left + right) / 2 and check if it's possible to accommodate at least C tourists with this minimum distance.
5. To check this, iterate through the sorted room floor numbers and count how many tourists can be accommodated with the current minimum distance. If the count is at least C, update the result and move the left pointer to mid + 1. Otherwise, move the right pointer to mid - 1.
6. Repeat the binary search until the left pointer is less than or equal to the right pointer.
7. Output the result, which is the largest minimum distance they can achieve for privacy.

C++ Solution: Tourist - C++
Python Solution: Tourists - Python (There is a chance of getting TLE as Python is comparatively slow.)
Java Solution: Tourists - Java

Problem E: King’s Order

Author: Rahat Khan Pathan
Approach: This problem can be efficiently solved using **Kruskal's Algorithm**, which finds the minimum spanning tree (MST) of a graph. The MST connects all cities with the minimum total edge weight. Here's a step-by-step approach:

1. Read the input values: N (the number of cities), E (the number of roads), and details of each road (A, B, and W, where A and B are the cities connected by the road, and W is the construction cost).
2. Sort the roads by their construction cost in non-decreasing order.
3. Initialise a disjoint-set data structure (DSU) to keep track of connected cities. Initially, each city is in its own set.
4. Iterate through the sorted roads:
5. For each road, check if its endpoints (cities) are in different sets. If they are, it means adding this road to the MST does not create a cycle, so add it to the MST and unite the sets of its endpoints.
6. After constructing the MST, check if it contains exactly N - 1 edges. If not, it's not possible to connect all cities, and you should output "Not Possible."
7. Calculate the total cost of all roads and subtract the cost of the MST from it to find the cost saved by removing some roads.
8. Output the number of roads that can be removed (E - N + 1) and the minimum construction cost (cost of the MST).

C++ Solution: King's Order - C++
Python Solution: King's Order - Python
Java Solution: King's Order - Java

Problem F: Bank Robbery

Author: Rahat Khan Pathan
Approach: To solve this problem efficiently, you can use **Dynamic Programming** to keep track of the maximum number of gold bars that can be obtained up to each locker. Here's a step-by-step approach:

1. Read the number of lockers, N, and initialise a vector goldBars to store the number of gold bars in each locker.
2. Initialise a dynamic programming array dp of size N to keep track of the maximum number of gold bars that can be obtained up to each locker.
3. Handle the first two lockers separately:
4. Set dp[0] to be the maximum of goldBars[0] and 0 (ensuring negative values, which represent trackers, are not considered).
5. Set dp[1] to be the maximum of goldBars[1] and 0, and also consider dp[0].
6. Iterate through the lockers starting from the third locker (index 2):
7. If the current locker has a tracker (indicated by a value of -1), skip it and set dp[i] to be equal to dp[i - 1]. The thief can't open this locker or any consecutive ones.
8. If the current locker does not have a tracker:
    a. Calculate dp[i] as the maximum of two possibilities:
        i. dp[i - 1]: The thief skips the current locker.
        ii. dp[i - 2] + goldBars[i]: The thief opens the current locker, adding its gold bars to the total. The dp[i - 2] term ensures that consecutive lockers with trackers are not triggered.
9. The maximum number of gold bars the thief can steal without triggering any trackers is stored in dp[N - 1].
10. Output dp[N - 1] as the result.

C++ Solution: Bank Robbery - C++
Python Solution: Bank Robbery - Python
Java Solution: Bank Robbery - Java