Roti-Prata Problem [CodeStudio]

You are hosting a cooking competition where participants have different ranks representing their cooking speed. The rank represents the number of time units required for a cook to make a single prata. The higher the rank, the slower the cook.

You have a target number of pratas that need to be cooked. Your task is to determine the minimum amount of time required to cook the target number of pratas using the given ranks.

Inputs:

Example 1

Target: 10 pratas

Rank: 4 1 2 3

Output: 11

The ranks are sorted in ascending order: 1 2 3 4 4

Let 'N' = 4, 'ranks' = [1, 2, 3, 4] and 'M' = 11. Then the minimum time required to cook 11 dishes will be 12 minutes. The cooks should prepare dishes in the following manner -:

Cook-0 prepare 4 dishes in 10 minutes i.e (1 dish in 1 minute, 1 more dish in next 2 minutes, 1 more dish in next 3 minutes, 1 more dish in next 4 minutes).

Cook-1 prepare 3 dishes in 12 minutes i.e (1 dish in 2 minutes, 1 more dish in 4 minutes, 1 more dish in 6 minutes).

Cook-2 prepare 2 dishes in 9 minutes i.e (1 dish in 3 minutes, 1 more dish in the next 6 minutes).

Cook-3 prepare 2 dishes in 12 minutes i.e (1 dish in 4 minutes, 1 more dish in the next 8 minutes).

If all four cooks work simultaneously then they can prepare(4 + 3 + 2 + 2 = 11) dishes in 12 minutes. And it is the minimum possible time.

Example 2

Target: 8 pratas

Rank: 1 1

Output: 36

Example 3

Target: 8 pratas

Rank: 8 1 1 1 1 1 1 1 1

Output: 1

Approach 1: Using Binary Search Algorithm to find the minimum time required to cook the pratas.

We start with the isPossibleSolution function. It checks if a certain time mid is a possible solution by simulating the cooking process.

Inside the isPossibleSolution function, we iterate over the ranks of the cooks.

For each cook, we initialize the cooking time time to their rank and set the multiplier multiplier to 2.

We enter a while loop that continues as long as the cooking time time is less than or equal to mid. This loop simulates cooking pratas until the cooking time exceeds mid or until the target number of pratas is reached.

Inside the loop, we increment the cookedDishes count, representing the number of pratas cooked so far.

If the number of cooked pratas equals the targetDishes, we have found a possible solution, so we return true.

We update the cooking time time by adding the product of the cook's rank and the multiplier. The multiplier is incremented for subsequent dishes, representing the increased cooking time for each prata.

If the while loop completes without finding a possible solution, we return false.

Moving on to the minCookTime function:

We sort the rank array in ascending order to ensure the cooks with lower ranks are considered first.

We initialize low to 0, ans to -1, and calculate high as the maximum possible time. The maximum possible time is obtained by multiplying the maximum rank by the sum of the target dishes. This ensures that each dish can be cooked individually by the cook with the maximum rank.

We enter a binary search loop, where we look for the minimum cooking time within the range low to high.

In each iteration, we calculate the mid time as the average of low and high.

We use the isPossibleSolution function to check if the mid time is a possible solution. If it is, we update ans to mid and continue searching for a smaller time by updating high to mid - 1.

If the mid time is not a possible solution, we continue searching for a larger time by updating low to mid + 1.

The binary search loop continues until low becomes greater than high, indicating that the search space has been exhausted.

Finally, we return the minimum cooking time ans.

Time and Space Complexity:

The time complexity of this solution is O(N * log M), where N is the size of the rank vector and M is the maximum possible time. The binary search performs log M iterations, and in each iteration, the isPossibleSolution function has a linear time complexity of O(N). Sorting the rank vector takes O(N * log N) time. Therefore, the overall time complexity is dominated by the binary search.

The space complexity is O(1) since the algorithm uses a constant amount of extra space for variables regardless of the input size.