ACM Summer Challenge 2020

BIT Manipulation

Editorial

Is Lower Bound Possible?

This question requires you to understand the basics of how a number can be represented using different radices. In context to this problem, you need to **represent a number with base 3**.

For example,
$13 = 1x3^0 + 1x3^1 + 1x3^2$
$14 = 2x3^0 + 1x3^1 + 1x3^2$
$15 = 0x3^0 + 2x3^1 + 1x3^2$
...and so on.

For every query, you are given a number **N**.
Represent the number in it's ternary form as described above (with base **3**). You need to find the **coefficients of powers of 3** in representation of **N** as shown in above example. This is left to you as an exercise.
If you don't encounter any coefficient equal to 2, then it means **N** can already be formed using distinct powers of 3. So, the required answer is **N**.
But what if it is not so?
Let's suppose that you get the coefficient of **3^i** as **2**. We need to then divide **N** by **3^i** so that this **2** is reduced to **1**. Now, we need the answer to be greater than **N**, because division results in a smaller number than **N**.
So, find **the smallest position j>i**, such that the coefficient of **3^j** in **N**'s ternary representation is **0**.
Multiply **N** with **3^j**. It can be proved easily that **N** is now greater than the given number.
Repeat the process until there is no **i** for which **3^i** has it's coefficient as **1** in **N**'s ternary representation.
This can be done in **O(QxLog(N))**.

Gogi

This question requires you to maintain the occurrence of any lowercase letter in the form of either BITSET or an array.
You can read more about BITSETS here.

For every given word W, maintain an array (or BITSET) of **26** values that keep the count of letters that occurred in the word.
Let's suppose your answer is stored in variable **ans**.
Now use 2 for-loops nested one in another and check for every two words $\mathbf{W_i}$ and $\mathbf{W_j}$ **1<=i,j<=N** and check whether they don't have any common letters.
If they don't have common letters, then,

$$\mathbf{ans = max(ans, len(W_i)*len(W_j))}$$

where $\mathbf{len(W_x)}$ denotes the length of word at index **X**
This can be done in $\mathbf{O(26xN^2)}$.

Luffy's Food Obsession

This question is a tricky one and requires you to understand the fact that any number can be represented as a sum of given numbers which can be halved provided that their sum is already equal or greater than the required number.

Therefore, the above statement provides you on how to approach the case where your answer will be **-1**.

Now, for the rest of the cases,
Maintain an array of size **61** (why 61? Maximum nutrition value of fruit is $10^{18} < 2^{60}$). Let's call this **pow2**.
For every given Fruit nutrition value **N**,
Find it's **log with base 2** and add them respectively to the **pow2** array.
For example, if **N = 16**, then **pow2[4] += 1.**

Let us store the answer (minimum number of times we need to halve a fruit) in the variable cnt.
Now, iterate this **pow2** array from **i = 0 to 61** and for every **i**, you need to repeat the following series of steps:

If the **i^{th} bit** in the binary representation of **N** is **not set**,

- **Floor divide pow2[i] by 2 and add it to pow2[i+1]** (meaning we will use these values to fill N later).

If the **i^{th} bit** in the binary representation of **N** is **set**,

- If **pow2[i] is at least 1**, then reduce it by **1**, **floor divide pow2[i] by 2 and add it to pow2[i+1]** (meaning we used a fruit value to fill **N** by **2^i** and added the remaining fruits that may be required to fill Luffy's belly later)
- If **pow2[i]** is **0**, then find smallest **j>i** such that **pow2[j] != 0**. This process signifies that you need to halve a fruit at the j^{th} index and while finding such a **j from position i, you can also fill Luffy's belly by 2^p where p belongs to (i,j) and p^{th} bit in N is set**. Thus, you need to **increase your answer stored in cnt by 1 every time you increase 1 from i to j** while finding such a **j**. Also, next time, you try filling Luffy's Belly with **i = j+1**. Also, **don't forget to reduce pow2[j] by 1 and floor divide it by 2 and add it to pow2[j+1]**.

Print your answer stored in cnt.

This can be done in **O(NLogN)**.