CodeKaze – Dec 2016

1. Can they fuse ?

Prerequisites : Strings , Hashing
Difficulty : Easy

Brute Way :
Let "Count" denotes the number of common elements in strings T and G. For each character in string T we search for similar character in string G, if it is not already taken we will increase the Count by one.

If Count >= K , then we print "YES"
Else we print "NO"

Time Complexity : O(N*M) , where N and M are lengths of string T and G respectively.

Brute Way gives us the polynomial complexity. Lets look at a faster and linear way to solve this problem.

Faster Way :

We see that there can be only 26 dictinct characters in any string. So we can make a hash of these 26 characters. Let **HashT [i]** denotes the **count of ith English alphabet in string T** and **HashG[i]** denotes the **count of ith English alphabet in string G .**

Now the count of common ith English alphabet in both the strings is min (HashT[i] , HashG[i]) . Iterate for all the 26 characters and find the total count of common characters.

Time Complexity : O(N + M) , where N and M are lengths of string T and G respectively.

2. Vegeta and Whis

Prerequisites : Dynamic Programming, Maths
Difficulty : Medium

For N number of stairs there can be at max 2*log(N) distinct steps that we can take, let that number be M. Let the value of ith step be Val [i] .

Let **dp (i , j)** denotes the **minimum number of steps we take to climb j stairs when we are considering only i distinct steps (out of all that are**

avialable). For every ith step we have two options, we can either take that step or not. So our recurrence relation for the dp becomes :

dp (i , j) = min {
dp (i , j – Val [i]) + 1 , *// when we take ith ste*
dp (i - 1, j) *// when we don't take the ith step*
}

dp (M , N) will give us the required answer.

Time Complexity : O(N*log(N)) , where N are the number of stairs.

3. Gohan and Modulo

Prerequisites : Segment Trees , Maths
Difficulty : Medium

Query 1 :

Query 1 suggests that we should build a segment tree to store sum of the range . But its difficult to figure out how we merge Query 2 with the segment tree because its not straight forward lazy propagation implementation .

Query 2 :

Lets look into a bit of maths. Suppose the query is x y z , there are two possible cases . Here i belongs to [x , y] .

Case 1: a[i]>=z .

Let A[i] =p*z+r , where r = A[i]%z and p is any integer greater than or equal to 1.
As 0<=r<=z-1 and z<= p*z .
In the worst case r=z-1 ,
A[i]=p*(r+1)+r.
A[i]=(p+1)*r+p.
so (nearly) r = a[i]/(p+1) where p>=1
So we can say that the value of A[i] reduces to half value even in worst case .
So we can say that each element will survives log2(A[i]) steps before it becomes 0 forever .

Case 2: A[i]<Z
Easily , we can see A[i]%z=A[i] . so no need to update this element.

so while updating , we follow the normal range update query similar to build tree . And then if a
range has a max value which is less than z . we can skip this update for the current range . so every element A[i] will be updated at max log(A[i]) times .

Runtime: q*log(n)*log(AMAX)

4. Help Goku

Prerequisites : Matrix Exponentiation, Maths
Difficulty : Hard

Brute Way :

For every subarray of the given array we will find its recurence term. Adding them will give us our answer.

Let M denotes the sum of subarray. The value of M can be as large as 10^14.

Time Complexity : O(N*N*M) .

This will surely give us Time Limit Error. Now lets look at a faster way.

Faster Way :

In the matrix form, we can write the recurrence in the following way :

$$\begin{bmatrix} T(n+1) \\ T(n) \\ T(n-1) \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} T(n) \\ T(n-1) \\ T(n-2) \end{bmatrix}$$

On solving furthur we get ,

$$\begin{bmatrix} T(n+1) \\ T(n) \\ T(n-1) \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^n \times \begin{bmatrix} T(1) \\ T(0) \\ T(-1) \end{bmatrix}$$

From the reccurence we can see that the value of T(-1) is 0. And therefore,

$$\begin{bmatrix} T(n+1) \\ T(n) \\ T(n-1) \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^n \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Here the matrix which is raised to the power "n" is known as objective matrix. Lets denote it by M. When we multiply M with itself n times , the element present at 2^{nd} **row** and 1^{st} **column** of the resulting matrix will give us the nth term of the reccurence. Or we can say M^n will give us the nth term of the recurrence. We can obtain M^n in O (log (n)) time.

Lets say S is the answer matrix and we have another matrix whose name is Prev. Initially both of them are Null matrices. Assume now we have an element in the array and its value is a1 . Therefore, we update the value of Prev in the following way,

$$\mathbf{Prev} = \mathbf{M}^{\mathbf{a1}} * (\mathbf{I} + \mathbf{Prev})$$

where I is the identity matrix of size 3x3 .

The value of Prev now becomes,

$$\mathbf{Prev} = \mathbf{M}^{\mathbf{a1}}$$

We will add Prev matrix to S matrix so that the value in S becomes

$$\mathbf{S} = \mathbf{M}^{\mathbf{a1}}$$

Lets say we get the second element in the array whose value is a2 therefore , we update the value in Prev in the following way ,

$$\mathbf{Prev} = \mathbf{M}^{\mathbf{a2}} * (\mathbf{I} + \mathbf{Prev})$$

The value of Prev now becomes,

$$\mathbf{Prev} = \mathbf{M}^{\mathbf{a2}} + \mathbf{M}^{\mathbf{a1+a2}}$$

Now we add Prev matrix to S matrix, which changes the value of S to

$$\mathbf{S} = \mathbf{M}^{\mathbf{a1}} + \mathbf{M}^{\mathbf{a1+a2}} + \mathbf{M}^{\mathbf{a2}}$$

We repeat the same procedure for the remaining elements in the array to get the result.

Time Complexity : O(d * N * log(MAX)) , where MAX is the value of maximum element in the array (can be as large as 10^9) and d is a constant for multiplying two matrices .

5. Krillin’s Last wish

Prerequisites : Miller Rabin primality test , Merge Sort Tree or MOS
Difficulty : NINJA

Let’s consider the given array to be A . Let’s try to build an array C where C[i] = count of distinct prime divisors of (A[i]) .

Brute Way

For every A[i] iterate over all primes till sqrt(A[i]) and keep incrementing the count if they divide A[i] .

Runtime : N*sqrt(A[i])

*This is not sufficient to pass the time limit * as A[i]<=10^9

Faster Way :

Statement 1:
Any number A[i]<=10^9 can be represented as S*X*Y . Where prime factorisation of S only contains primes less than equal to 10^3 and X,Y are greater than 10^3 which may or may not exist (this means thats X ,Y can be 1 too) . **There can not be more than two primes which are greater than 10^3 because then the number A[i] will be greater than 10^9 .**

Algorithm :

Divide A[i] with primes less than 10^3 keep incrementing the count . So after this there are three cases possible:
CASE 1: A[i]=1 , do nothing .
CASE 2: A[i] is a prime , check with miller rabin test . if A[i] is prime then count+=1 .Else move to case three .
CASE 3: A[i] is left with X,Y only . if (X==Y) count+=1 , else count+=2 .

At this position we have computed array C in O(N*10^3) better than previous.

So now the question turns to this : Find the count of elements in range [X,Y] which are greater than K . This is a standard problem which can be solved using Merge sort trees online or MOS offline method .

Runtime :

Using MOS : O(N*10^3 + Q*sqrt(N)*Log(Log(N))
Using Merge Sort Trees : O(N*10^3 + Q*Log(N)*Log(N))