KIRLAB-Codechef DEC16

DECEMBER 12, 2016 ~ ATULAC

The problem reduces to finding longest possible subsequence such that the gcd of every adjacent pair in the subsequence is greater than 1.

The Brute Force Way (Fetches 30 points)

Just scan through the elements in the array as we scan in case of Longest Increasing Subsequence. Replace the condition of comparing elements by checking for the condition if the gcd of two elements being compared is

Search …

Recent Posts

KIRLAB-Codechef DEC16

Recent Comments

Aditya kakliya on KIRLAB-Codechef

greater than 1 or not.

Step 1:

Declare an array with name length which stores the length of longest valid subsequence till each index. Initialise it to 1 for each index because a single element is always a valid subsequence of length 1.

Step2:

Now iterate through the array and for each index i (1 to n) , find all the indices j (0 to n-1) before that index which gives gcd (arr[i] , arr[j]) > 1 , and update the length by adding 1 to the index at j if that provides a larger length as compared to the previous value.

length(i)=max { length(j) } + 1 if gcd (arr[i],arr[j]) > 1

```
length = [1,1,1,1,1,1,1,1]

```

DEC16

Archives

December 2016 (1)

Categories

Uncategorized

```
for i = 1 to n-1 :          //Iterates thro
   for j = 0 to n-1 :        //Iterates thr

       if gcd(arr[i],arr[j]) is greater th
           if (ans[j] + 1) is greater than
               length [i]= length[j] + 1

return max(length)
```

The returned value is the maximum length of the subsequence. The time complexity of the solution is O(n^2) and therefore it passes only the first subtask. You can find the accepted code at atulac – Codechef Solution

KIRLAB (30 points)

The Optimal Way (Fetches 100 points)

This method uses dynamic programming and makes use of the fact that if gcd of two numbers is greater than 1, they must be sharing atleast one common prime factor. Now the constraint of each element in the array is 10^7 and such numbers can have atmost 9 prime factors because (2*3*5*7*11*13*17*19*23 = 223092870 with 9 prime factors and exceeds 10^7). This fact accounts for the huge improvement in time complexity of the approach as atmost 10 factors on single element means atmost 10^6 operations over 10^5 elements, if primes are preprocessed and stored.

Now if we have a list of prime factors occuring in the array elements in total i.e. prime factors known by factorizing each element and then grouping them in a set, the problem reduces to finding the maximum among the longest subsequences of elements with each prime in the list as a common factor.

The problem finally rests on how fast we compute prime factors of each element and how we apply dynamic programming to our advantage.

In order to efficiently prime factorize numbers we can use sieve method and can store the smallest factor of each element in an array or a map.

```
function sieve():

     sm_prime= smallest prime array filled
     max=maximum possible value of element

     for(i=2; i less than max; i++):

         if sm_prime[i] == 0:
             sm_prime[i]=i

             for(j=i*i; j less than max; j

                  if sm_prime[j] == 0:
                      sm_prime[j]=i

      return sm_prime
```

Now when we have smallest prime factor of each element in sm_prime, we can use it to factorize any element p through following method which repeatedly divides a number by its smallest prime factor and then does the same with the dividend produced util the number reduces to 1. Note that a number is a prime number if its smallest prime is the number itself. Thus, it needs to be added into the factor list in case it is found to be prime.

```
function factorize(element p):

      factors=[]              //an empty l

      if sm_prime[p]==p:
          factors.add(p)      //a prime nu

      while p is greater than 1:
```

```
                factors.add(sm_prime[p])
                p/=sm_prime[p]

        return factors
```

Now dynamic programming comes into play.

We take each element, find its factors and go to each of those factor indices in the length array marking them as 1. This implies that a subsequence of length 1 has been found for the factor in view. This step of marking 1 is actually a sequence of two steps, one of finding the largest subsequence for that factor till now (say of length len) and then updating each of the value at those factor indices in length array as len+1. Here len = 0 and therefore 0+1 is updated to the indices corresponding to the factors found till now.

For each successive elements in the array, we find out the factors using factorize() and then iterate over those factors to find which among those factors have the longest subsequence till now and of what length (len). Once we know this, we update the indices corresponding to factors in the factor list with the value len+1. This now implies that for the factor in test, we have found a subsequence of length len+1.

```
function dp():

      length array initialised to 0

      for each_element in array:

          factorize each_element and store

          len=0
          for each_factor in factors:
              len=max(len,length[each_fact

          for each_factor in factors:
              length[each_factor]=len+1

 return max(length)
```

After iterating through each element of the array, we end up with an array whose each index with value k states that longest subsequence of length k is present till the current index and therefore the maximum element in the length array is our answer.

The accepted code for 100 points can be found at atulac-Codechef Solution KIRLAB (100 points)

Clarifications of doubts or suggestions for improvements are highly welcomed.

Thank You.

atulac

Share this:

Twitter Facebook Google

Loading...

One blogger likes this.

2 thoughts on “KIRLAB-Codechef DEC16”

Aditya kakliya

DECEMBER 13, 2016 AT 6:25 AM

Could you please give your email id?

★ Like

Reply

atulac

DECEMBER 13, 2016 AT 6:59 AM

chaturvediatul05@gmail.com

Also you can connect me at http://www.facebook.com/atulac.11

★ Like

Reply

Leave a Reply

Enter your comment here...

POWERED BY WORDPRESS.COM.

Follow