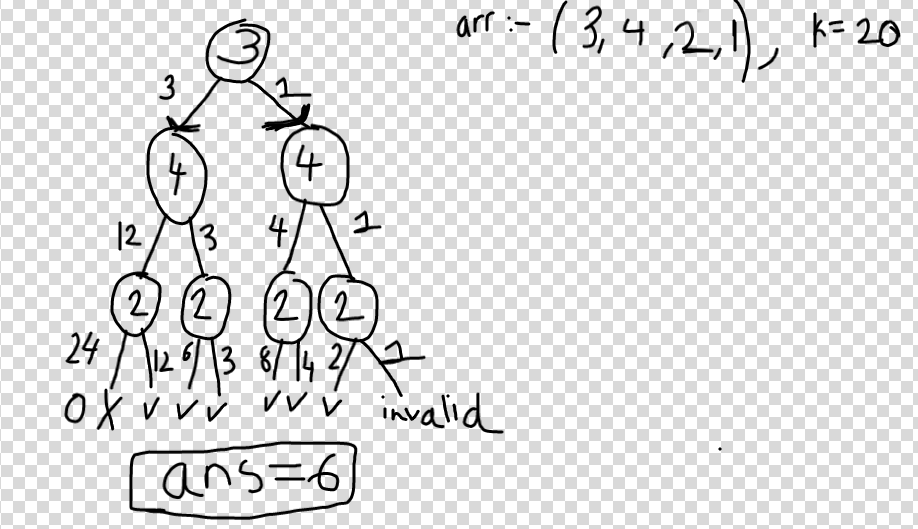
**Problem :** [**https://www.geeksforgeeks.org/count-subsequences-product-less-k/**](https://www.geeksforgeeks.org/count-subsequences-product-less-k/)

**Constraints :**

**n<=62**

**k<=100**

**Easy top down approach : Take or not take approach.**



State = dp[i][prod] , means by including some elements in a valid subsequence from 0...index-1 we have got some product “prod”,now decide whether to include the element at current index “i” or not into the subsequence in the current path or not.

**if arr[i]\*prod>k**,then dont go ahead return 0,as current element cant be added to subsequence,go ahead without including current element,

Else **just try both , by including and not including it.**

**//Psudo Code :**

**if(prod\*v[cur] < k) {**

**func(v,k,n,cur+1,prod\*v[cur]);**

**func(v,k,n,cur+1,prod);**

**dp[cur][prod] = dp[cur+1][prod] + dp[cur+1][prod\*v[cur]];**

**}**

**else**

**{**

**func(v,k,n,cur+1,prod);**

**dp[cur][prod]=dp[cur+1][prod];**

**}**

=> No of unique states = n\*k=62\*100 and transition time = 2(constant)

**So time = O(n\*k)**

**=> Code :** [**https://ideone.com/XyLpCW**](https://ideone.com/XyLpCW)

**Hard Approach :**

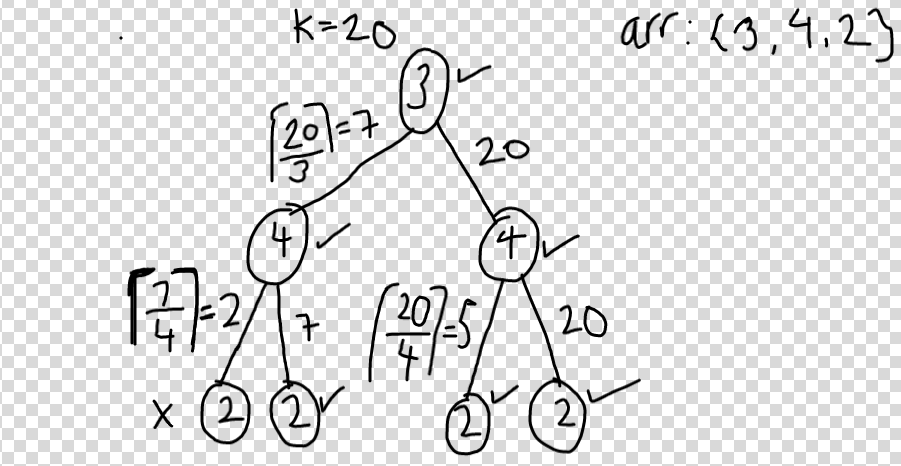
->Let’s say the product until now is ‘m’ ,then any other element ‘x’ which comes further in array has to obey :

m\*x<k

So x<(k/m)

-> It can be actually written as **x<ceil(k/m) ,** so if the next element ‘x’ obeys the property it can be in the current subsequence. Now ‘m’ becomes m = m\*x, and we check for next element ‘x’ in the array whether it’s lesser than ceil(k/m), and so on…

-> So it’s easy to see that , we can start from behind, and for each element at index ‘i’,we can have dp[i][k],where **dp[i][k] indicates number of subsequences (anywhere between index i till last) and having product lesser than k.**

****

**E.g :** if you take 3 in subsequence,then next number should be <ceil(20/3) means <7,

If dont take 3 ,then nxt\_num< 20,

So 4<7,so if we take 4 , the next num should be <ceil(7/4) means <2,and so on...

So

**if(arr[i]<k) {**

**dp[i][k] = 1 + dp[i+1][((k-1)/m + 1)] + dp[i+1][k];**

//1=>the element itself is a subsequence

// dp[i+1][((k-1)/m + 1)] = no of subsequences behind which have prod<ceil(k/m),so if we add the current element to all those subsequences product will still be lesser than ‘k’.

// dp[i+1][k]=no of subsequences behind which have prod<k,means if we don’t take this element than all the subsequences behind which have product<k will also have to be added.

**}**

**else{**

**dp[i][k] - dp[i+1][k]**

**}**

//also if arr[i]=0,then to all other subsequences behind if we also add arr[i]=0 to the set the product becomes 0.

So in that case dp[i][k] = 1 + 2^(count of remaining elements) + dp[i-1][k]

**Code :** [**https://ideone.com/j0sNvv**](https://ideone.com/j0sNvv)