Words - K Length Words- 4

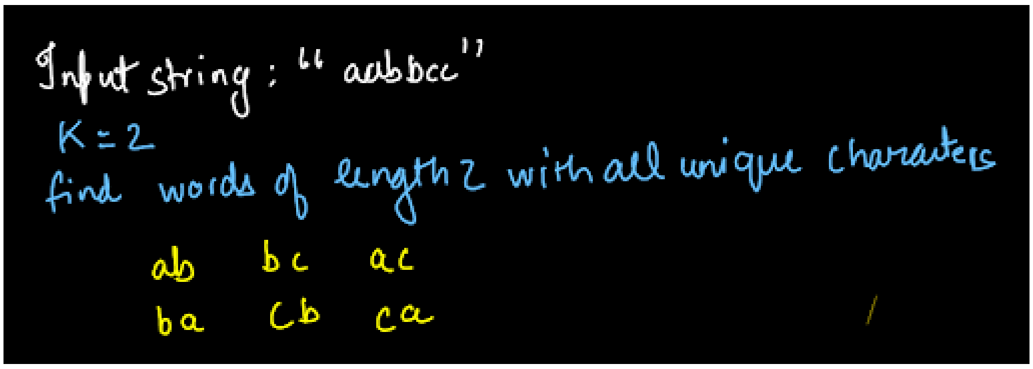
1. You are given a word (may have one character repeat more than once).
2. You are given an integer k.
2. You are required to generate and print all ways you can select k characters out of the word.
Note -> Use the code snippet and follow the algorithm discussed in question video. The judge can't force you but the intention is to teach a concept. Play in spirit of the question.

aabbbccdde
3
['aab', 'aac', 'aad', 'aae', 'aba', 'aca', 'ada', 'aea', 'baa', 'caa', 'daa', 'eaa', 'abb', 'abc', 'abd', 'abe', 'acb', 'adb', 'aeb', 'acc', 'acd', 'ace', 'adc', 'aec', 'add', 'ade', 'aed', 'bab', 'bac', 'bad', 'bae', 'cab', 'dab', 'eab', 'cac', 'cad', 'cae', 'dac', 'eac', 'dad', 'dae', 'ead', 'bba', 'bca', 'bda', 'bea', 'cba', 'dba', 'eba', 'cca', 'cda', 'cea', 'dca', 'eca', 'dda', 'dea', 'eda', 'bbb', 'bbc', 'bbd', 'bbe', 'bcb', 'bdb', 'beb', 'cbb', 'dbb', 'ebb', 'bcc', 'bcd', 'bce', 'bdc', 'bec', 'bdd', 'bde', 'bed', 'cbc', 'cbd', 'cbe', 'dbc', 'ebc', 'dbd', 'dbe', 'ebd', 'ccb', 'cdb', 'ceb', 'dcb', 'ecb', 'ddb', 'deb', 'edb', 'ccd', 'cce', 'cdc', 'cec', 'dcc', 'ecc', 'cdd', 'cde', 'ced', 'dcd', 'dce', 'ecd', 'ddc', 'dec', 'edc', 'dde', 'ded', 'edd']

```python
import collections
def wordsKLengthIV(s,k):
    fmap = collections.Counter(s)
    s = ''.join(list(set(s)))
    n = len(s)
    ans = []
    helper(s,k,fmap,ans,'')
    return ans

def helper(s,k,fmap,ans,ssf):
    if k==len(ssf):
        ans.append(ssf)
        return
    for i in range(len(s)):
        ch = s[i]
        if fmap[ch]!=0:
            fmap[ch]-=1
            helper(s,k,fmap,ans,ssf+ch)
            fmap[ch]+=1




print(wordsKLengthIV('aabbbccdde',3))
```

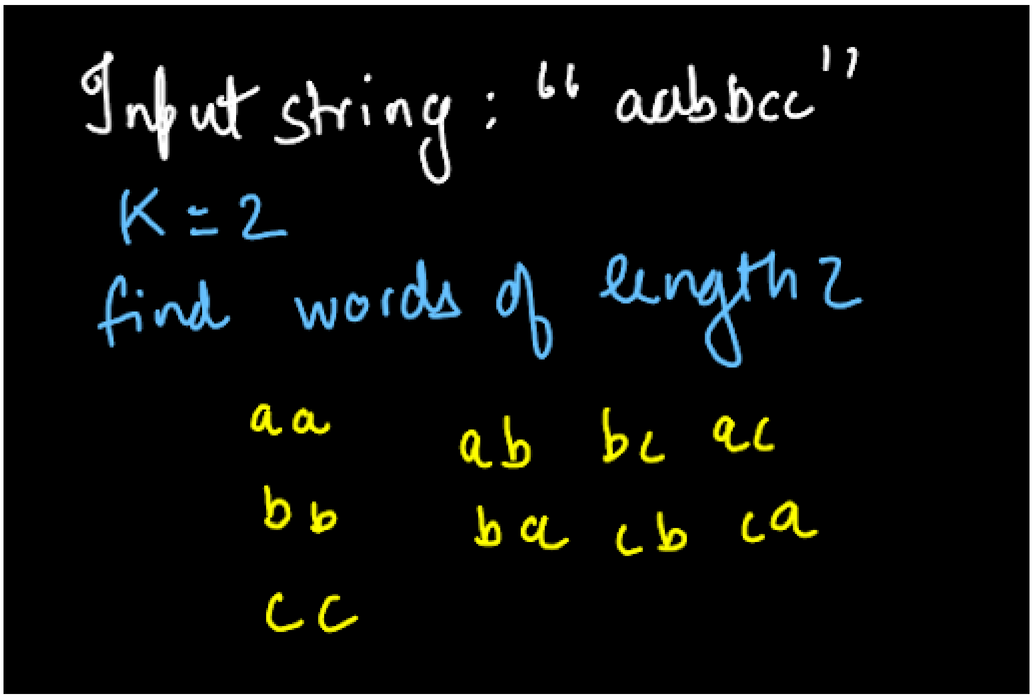
Welcome back, dear reader. So, how is it going? We hope you are doing well and learning the concepts and enjoying your coding journey. So, in this article, we will discuss the problem WORDS- K LENGTH WORDS-4. So, what does this problem say? Before we jump into this problem, do you remember the problem K-LENGTH WORDS-2 that we have solved previously. If you have not solved this problem, we recommend you try to solve this problem on your own first and refer to the solution video for K-LENGTH WORDS-2 problem because this is an extended version of the same problem. So, let us recall a little bit about that problem first and then we will tell you about this problem.

We will be given an input string which can have repeated characters also. In the k-length words-1 problem, we had to find the words of length k, which do not have any repeating character. For instance:



We were given the input string "aabbcc". The 2 letter words where both the characters are unique are shown in the image above. Now, there is a slight variation in this problem.

We still have to print k letter words but the words can have repeating characters. For instance:



So, we hope you have got the question and the difference between these two problems. You may refer to the K-LENGTH WORDS-4 video to understand the question if you have any doubts about the same. We recommend you try to solve this problem on your own first and then go to the solution.

Approach :

Time Complexity: O(n!) where n is the length of the input string

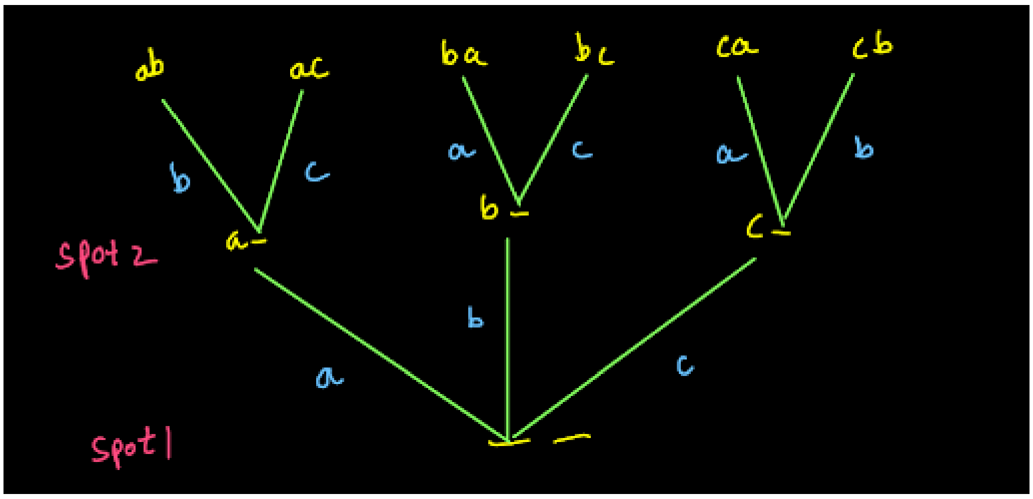
Space Complexity: O(h) where h is the height of the recursion stack.

Explanation :

Let us have a quick recap of the WORDS-K LENGTH WORDS-2 problem and see how we used to solve it.

So, let us say that the input string is "abcabc" and we have to find the words of length 2. So, in this question, since we had to find the words with every character unique, we selected the unique string "ustr" first from the complete input string. So, we have "abc" as the string now. After this, we used to keep the spots at the levels and the characters used to be our options (Remember we compared this with the box and items problem?).

So, at every level, we had a spot and it had n options i.e. n different characters to fit in. The entire tree for the same is shown below:

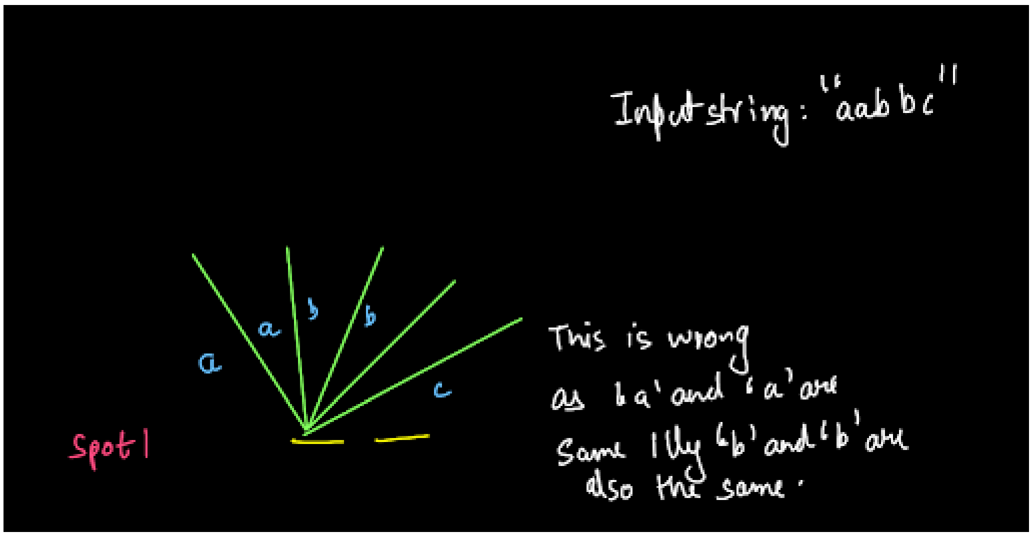


So, at every level, we used to have options for the spots to fit in a character. Here, we are having a little bit of a different situation.

Here we can have multiple occurrences of the same character in the word. So, at least we know where to start now,right? So, let us begin the procedure for our question.

Constructing Recursion Tree

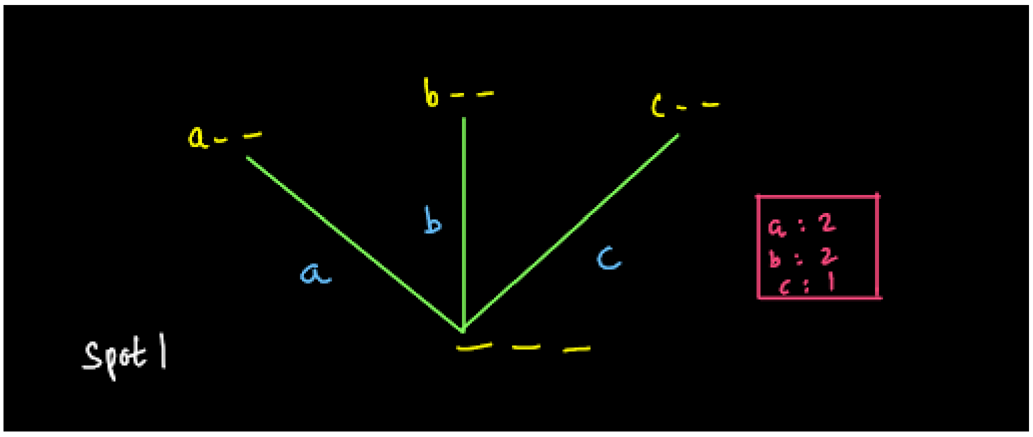
Let us take the input string "aabbc". Now, we will start by placing the spots at the levels and we will see what all options we have at each spot. Have a look at the image shown below:


Input string: "aabbc"
a
a
b
b
c
Spot 1
This is wrong as 'a' and 'a' are same Illy 'b' and 'b' are also the same.

The options shown above are incorrect. This is because 'a' and 'a' are the same characters. So, we should have only three options, 'a', 'b' and 'c'. So, we will need to make a unique string "ustr" as we made in the k-length words-2 problem to have the options available for every spot.

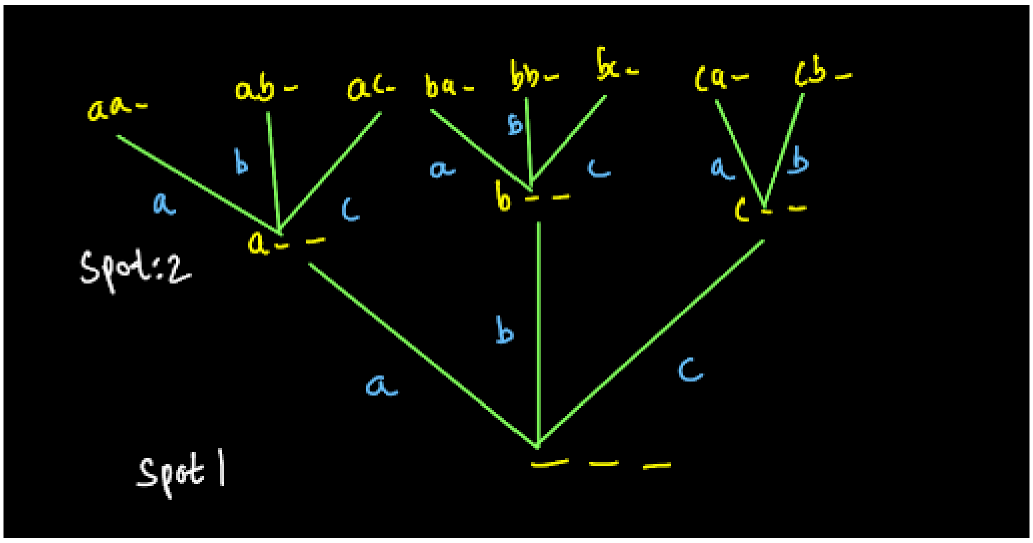
But, there are 2 occurrences of the character 'a' in our string. This also means that if it has taken one spot, it may still take the other. So, we will maintain a hashmap, which will tell us about the frequencies of each character. As we put these characters into the spots, we will reduce their frequencies in the hashmap and if for a spot, if the frequency of a character is 0, it will not have the option to put that character into it.

Now, let us start the procedure again. Let us also increase the number of spots to 3 so that your understanding becomes better. So, we have now the input string "aabbc" and we have to make words of length 3 from it.

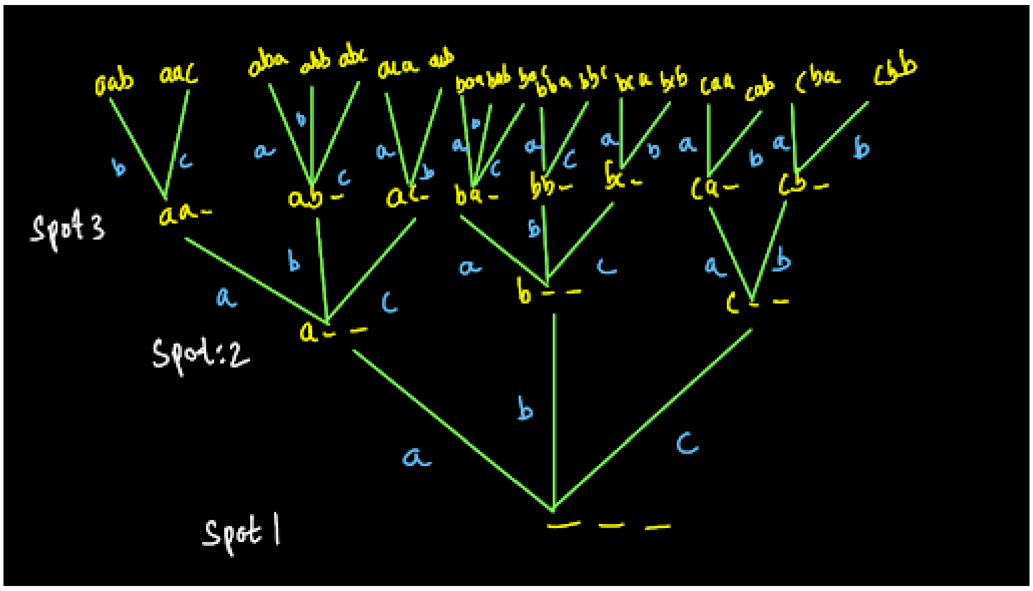


As you can see, at the 0th level, we try to fill the first spot. So, here we will be using the string "ustr" which shows that there are only three options 'a', 'b' and 'c'. The hashmap is also present with the frequency mappings of each character. We will not be able to show a hashmap for each case in the diagram so, carefully look at the diagram when we decrease an option.

Now, let us try to fill the second spot.



So, look at img-6 carefully. If we had selected 'c' at the 0th level, we would not have an option to select 'c' at the next level because it had already been selected and its frequency was only 1. Now its frequency is 0. So, we have only two options for that case. Let us fill the third and the final spot.



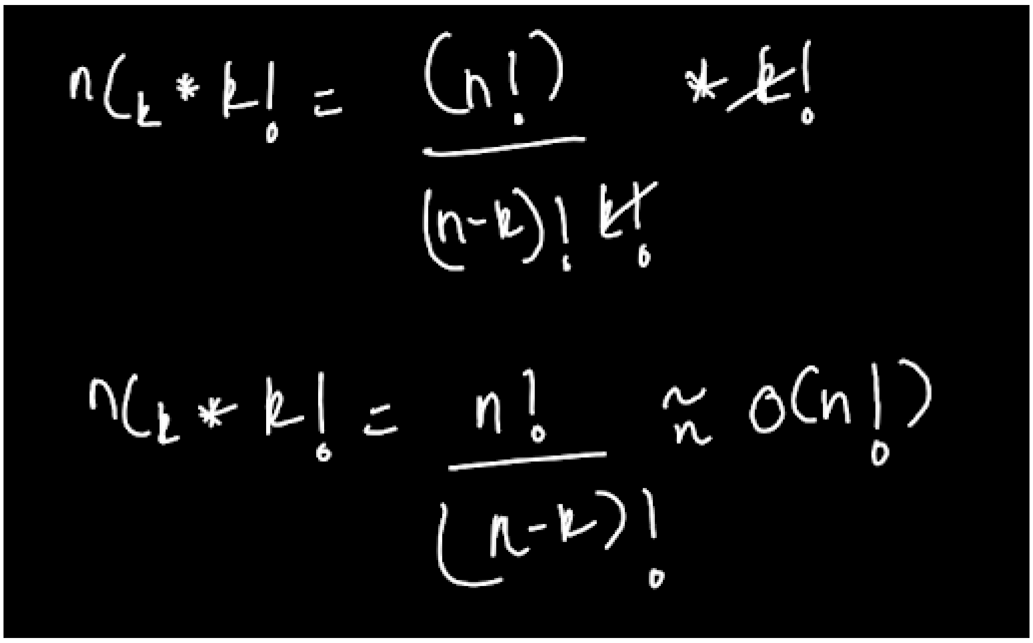
So, we have found the third spot also and these are the words that we can form from the input string "aabbc". This process is simpler in comparison with the process that you studied in K-LENGTH WORDS-3. We hope that you have understood this procedure.

You may refer to the solution video to understand the above procedure completely and clear all your doubts about the above procedure.

Analysis

Time Complexity:

If we take the above example, we were to select 3 letters from 4 as we were making 3 letter words. Then, we had 3! Arrangements of those letters also. So, first we select the k letters from n i.e. ^nC_k and then we are arranging them in k! ways. So, the time complexity should be ^nC_k x k!



$${}^{n}C_{k} * k! = \frac{(n!)}{(n-k)!\,\cancel{k!}} * \cancel{k!}$$

$${}^{n}C_{k} * k! = \frac{n!}{(n-k)!} \approx O(n!)$$

Space Complexity:

The space complexity without considering recursion will be O(1) and if we consider recursion it will be O(h) where h is the height of the tree. Also, what is the height of the tree? Since at every level, we have spots i.e. the length of the words that we want, we will have the height of the tree as O(k). So, the max height of the recursion stack will also be O(k) and so will be the space complexity.

So, dear reader, we hope that you understood the time and space complexity analysis also. If you have any doubts regarding the procedure or the code, refer to the complete solution video to clear all your doubts. With this, we have completed this problem.