Count Distinct Subsequences

Difficulty Level : Hard • Last Updated : 05 Jul, 2022

Given a string, find the count of distinct subsequences of it.

Examples:

```
Input  : str = "gfg"
Output : 7
The seven distinct subsequences are "", "g", "f",
"gf", "fg", "gg" and "gfg"

Input  : str = "ggg"
Output : 4
The four distinct subsequences are "", "g", "gg"
and "ggg"
```

Recommended Problem

Number of distinct subsequences

Dynamic Programming Strings +2 more

Solve Problem

Submission count: 9.8K

The problem of counting distinct subsequences is easy if all characters of input string are distinct. The count is equal to ${}^nC_0 + {}^nC_1 + {}^nC_2 + \ldots {}^nC_n = 2^n$.
How to count distinct subsequences when there can be repetition in input string?
A Simple Solution to count distinct subsequences in a string with duplicates is to generate all subsequences. For every subsequence, store it in a hash table if it doesn't exist already. The time complexity of this solution is exponential and it requires exponential extra space.

ethod 1 (Naive Approach): Using a set (without Dynamic Programming)

Approach: Generate all the possible subsequences of a given string. The

1. Include a particular element(say ith) in the output array and recursively call the function for the rest of the input string. This results in the subsequences of a string having ith character.
2. Exclude a particular element(say ith) and recursively call the function for the rest of the input string. This contains all the subsequences which don't have the ith character.

   Once we have generated a subsequence, in the base case of the function we insert that generated subsequence in an unordered set. An unordered Set is a Data structure, that stores distinct elements in an unordered manner. This way we insert all the generated subsequences in the set and print the size of the set as our answer because at last, the set will contain only distinct subsequences.

Implementation:

C++

```
// C++ program to print distinct
// subsequences of a given string
#include <bits/stdc++.h>
using namespace std;

// Create an empty set to store the subsequences
unordered_set<string> sn;

// Function for generating the subsequences
void subsequences(char s[], char op[], int i, int j)
{

    // Base Case
    if (s[i] == '\0') {
        op[j] = '\0';

        // Insert each generated
    // Recursive Case
    else {
        // When a particular character is taken
        op[j] = s[i];
        subsequences(s, op, i + 1, j + 1);

        // When a particular character isn't taken
        subsequences(s, op, i + 1, j);
        return;
    }
}

// Driver Code
int main()
{
    char str[] = "ggg";
    int m = sizeof(str) / sizeof(char);
    int n = pow(2, m) + 1;

    // Output array for storing
    // the generating subsequences
    // in each call
    char op[m+1]; //extra one for having \0 at the end

    // Function Call
    subsequences(str, op, 0, 0);

    // Output will be the number
    // of elements in the set
    cout << sn.size();
    sn.clear();
    return 0;

    // This code is contributed by Kishan Mishra
}
```

Python3

```
# Python3 program to print
# distinct subsequences of
# a given string
import math

# Create an empty set
# to store the subsequences
sn = []
```

```
// This code is contributed by patel2127
```

Output

```
4
```

Time Complexity: O(2^n)
Auxiliary Space: O(n)
where n is the length of the string.

Method 2(Efficient Approach): Using Dynamic Programming

An Efficient Solution doesn't require the generation of subsequences.

```
Let countSub(n) be count of subsequences of
first n characters in input string. We can
recursively write it as below.

countSub(n) = 2*Count(n-1) - Repetition

If current character, i.e., str[n-1] of str has
not appeared before, then
   Repetition = 0

Else:
   Repetition  =  Count(m)
   Here m is index of previous occurrence of
   current character. We basically remove all
   counts ending with previous occurrence of
   current character.
```

How does this work?

If there are no repetitions, then count becomes double of count for n-1 because we get count(n-1) more subsequences by adding current character at the end of all subsequences possible with n-1 length.
there are repetitions, then we find a count of all distinct subsequences ending with ne previous occurrence. This count can b tained by recursively calling for an

Below is the implementation of the above idea.

C++

```
// C++ program to count number of distinct
// subsequences of a given string.
#include <bits/stdc++.h>
using namespace std;
const int MAX_CHAR = 256;

// Returns count of distinct subsequences of str.
int countSub(string str)
{
    // Create an array to store index
    // of last
    vector<int> last(MAX_CHAR, -1);

    // Length of input string
    int n = str.length();

    // dp[i] is going to store count of distinct
    // subsequences of length i.
    int dp[n + 1];

    // Empty substring has only one subsequence
    dp[0] = 1;

    // Traverse through all lengths from 1 to n.
    for (int i = 1; i <= n; i++) {
        // Number of subsequences with substring
        // str[0..i-1]
        dp[i] = 2 * dp[i - 1];

        // If current character has appeared
        // before, then remove all subsequences
        // ending with previous occurrence.
        if (last[str[i - 1]] != -1)
            dp[i] = dp[i] - dp[last[str[i - 1]]];

        // Mark occurrence of current character
        last[str[i - 1]] = (i - 1);
    }

    return dp[n];
}

// Driver code
```

```
}

// Driver code
document.write(countSub("gfg"));

// This code is contributed by mukesh07
```

Output

```
7
```

Time Complexity: O(n)
Auxiliary Space: O(n)

Method 3: Without extra space

Let's say we have 2 variables : `allCount` which adds up total distinct subsequence count and `levelCount` which stores the count of subsequences ending at index i. To find repetitions we will store the most recent levelCount for each character. Finally we will see how we can determine `allCount` using the `levelCount` variable.

Here is a representation of above statement:

Say s = "abab"

Let's initialize all string characters in a map with -1 . The value in this map represents how many distinct subsequences ended at the last occurrence of this character.

map = {a:-1,b:-1}

Also initially we have

levelCount=0;

allCount = 0;

Now iterating through each character

- **1st iteration 'a'**
  - Distinct Subsequences ending with 'a' are : "a" . So we make levelCount = 1, also allCount is 1 now.

  - Distinct Subsequences ending with 'b' are "ab","b". So the levelCount = 2.
  - Also total subsequences we found till now are 3. allCount = 3.
  - Here we can notice that levelCount can be determined from last value of allCount variable by adding 1 to it
  - levelCount = allCount+1 (levelCount= 1+1=2)
  - In case, if this is a distinct character then current allCount can also be determined easily as
  - allCount = allCount + levelCount; (allCount= 1+ 2 = 3)
  - We also update the map with the current character's levelCount. map{a:1,b:2}
- **3rd iteration 'a'**
  - Now we have a repetition.
  - Distinct Subsequences ending with 'a' are now "aa","ba","aba","a". So our levelCount is now 4: which is determined as previously stated allCount+1 = 3+1 = 4.
  - if this was a distinct character allcount would have been 7 (allCount = allCount+levelCount = 3+4) but we will have remove the repetition, which is map.get(`a`) which is 1 , so now allCount is 7-1 = 6
  - Here note that we have essentially remove the results of our 1st iteration which was duplicate subsequence "a". This simply means that we can form the same subsequences ending with our new founded `a` that old `a` would have formed, so we subtract the older subsequences.
  - map is now updated with new levelCount of a to {a:4,b:2}
  - In case of repetition out allCount calculation changes as
  - allCount = allCount + levelCount – map.get(currentCharacter);
  - allCount = 3+4-1 = 6
- **4th iteration 'b'**
  - Again a repetition.
  - Subsequences ending with 'b' are now "abb","bb","aab","bab","abab","ab" ,"b" which count is same as levelCount = allCount+1 = 6+1 = 7 .
  - allCount will be = allCount+levelCount – map.get('b') = 6+7-2 = 11
  - Total number of distinct subsequences is allCount.
  - If empty String is also included then our answer is allCount+1.
  - Below is the implementation of the above approach.