2.Medium\05.Count_the_number_of_substrings_with_k_unique_characters.cpp

```
/*Given a string of lowercase alphabets, count all possible substrings (not necessarily distinct) that have exactly k distinct characters.

Example 1:

Input:
S = "aba", K = 2
Output:
3
Explanation:
The substrings are:
"ab", "ba" and "aba".

Example 2:

Input:
S = "abaaca", K = 1
Output:
7
Explanation:
The substrings are:
"a", "b", "a", "aa", "a", "c", "a".

Approach:
1. We can solve this problem using the sliding window technique.
2. Initialize a variable ans to keep track of the count of substrings with exactly k distinct characters.
3. Create an unordered_map mp to store the count of characters in the current window.
4. Initialize two pointers i and j to mark the start and end of the window, both initially pointing to the start of the string.
5. Iterate j from the start to the end of the string:
   - Increment the count of the current character s[j] in mp.
   - If the number of distinct characters in mp exceeds k, move the start pointer i towards the right until the number of distinct characters becomes k again.
   - Update the ans by adding the length of the current window (j - i + 1) to it.
6. Return the value of ans.

Code:*/

long long int substrAtmostK(string s, int k) {
    long long int ans = 0;
    unordered_map<char, int> mp;
    int i = 0;
    for (int j = 0; j < s.size(); j++) {
        mp[s[j]]++;

        while (mp.size() > k) {
            mp[s[i]]--;
            if (mp[s[i]] == 0)
                mp.erase(s[i]);
            i++;
        }
```

```
49
50          ans += j - i + 1;
51      }
52      return ans;
53  }
54
55  long long int substrCount(string s, int k) {
56      long long int atmostk = substrAtmostK(s, k);
57      long long int atmostk_1 = substrAtmostK(s, k - 1);
58      return atmostk - atmostk_1;
59  }
60
61  /*Time Complexity: O(N), where N is the length of the input string.
62  Space Complexity: O(K), where K is the number of distinct characters.*/
63
64
```