Count Distinct Subsequence C++

```cpp
#include <iostream>
#include <unordered_map>
using namespace std;

int countDistinctSubsequences(const string& str) {
    int n = str.length();
    int dp[n + 1];
    dp[0] = 1; // Empty subsequence

    unordered_map<char, int> lastOccurrence;

    for (int i = 1; i <= n; i++) {
        dp[i] = 2 * dp[i - 1];
        char ch = str[i - 1];
        if (lastOccurrence.find(ch) !=
lastOccurrence.end()) {
            int j = lastOccurrence[ch];
            dp[i] -= dp[j - 1];
        }
        lastOccurrence[ch] = i;
    }
    return dp[n] - 1;
}

int main() {
    string str = "abc";
    cout << countDistinctSubsequences(str) << endl;
    return 0;
}
```

Step-by-Step Dry Run:

Input:

```cpp
string str = "abc";
```

- Length of the string n = 3.

Initialization:

```cpp
dp[0] = 1; // Empty subsequence
unordered_map<char, int> lastOccurrence;
```

- Initially, dp = [1, 0, 0, 0] (the first element is 1 for the empty subsequence).
- lastOccurrence is empty.

Iteration 1 (i = 1, character = 'a'):

- dp[1] = 2 * dp[0] = 2 * 1 = 2 (considering subsequences from previous).
- 'a' has not been seen before, so no need to subtract.
- lastOccurrence['a'] = 1.
- After this iteration, dp = [1, 2, 0, 0].

Iteration 2 (i = 2, character = 'b'):

- dp[2] = 2 * dp[1] = 2 * 2 = 4.
- 'b' has not been seen before, so no need to subtract.
- lastOccurrence['b'] = 2.
- After this iteration, dp = [1, 2, 4, 0].

Iteration 3 (i = 3, character = 'c'):

- dp[3] = 2 * dp[2] = 2 * 4 = 8.
- 'c' has not been seen before, so no need to subtract.
- lastOccurrence['c'] = 3.
- After this iteration, dp = [1, 2, 4, 8].

Final Result:

- dp[n] = dp[3] = 8.
- Subtract 1 to exclude the empty subsequence: 8 - 1 = 7.

Output:

```
7
```

Explanation of Output:

The distinct subsequences of "abc" are:

- "" (empty subsequence)
- "a"

	• "b" • "c" • "ab" • "ac" • "bc" • "abc" Thus, there are **7 distinct subsequences**, excluding the empty subsequence.
Output:- 7	