Longest Substring With At Most K Unique Characters in C++

```
#include <iostream>
#include <string>
#include <unordered_map>

class LongestSubstringWithAtMostKUniqueCharacters {
public:
    static int sol(const std::string& str, int k) {
        int ans = 0;
        int i = -1;
        int j = -1;
        std::unordered_map<char, int> map;

        while (true) {
            bool f1 = false;
            bool f2 = false;

            while (i < static_cast<int>(str.length()) - 1) {
                f1 = true;
                i++;
                char ch = str[i];
                map[ch]++;

                if (map.size() <= k) {
                    int len = i - j;
                    if (len > ans) {
                        ans = len;
                    }
                } else {
                    break;
                }
            }

            while (j < i) {
                f2 = true;
                j++;
                char ch = str[j];
                if (map[ch] == 1) {
                    map.erase(ch);
                } else {
                    map[ch]--;
                }

                if (map.size() > k) {
                    continue;
                } else {
                    int len = i - j;
                    if (len > ans) {
                        ans = len;
                    }
                    break;
                }
            }

            if (!f1 && !f2) {
                break;
            }
        }
        return ans;
    }
};
```

Step-by-Step Dry Run:

Initial state:

- str = "ddacbbaccdedacebb"
- k = 3
- ans = 0
- i = -1, j = -1
- map = {} (empty initially)

First iteration (expanding and contracting window):

Expand the window (while (i < str.length() - 1)):

1. i = 0, character is d, map = {d: 1} (1 unique character)
2. i = 1, character is d, map = {d: 2} (still 1 unique character)
3. i = 2, character is a, map = {d: 2, a: 1} (2 unique characters)
4. i = 3, character is c, map = {d: 2, a: 1, c: 1} (3 unique characters)
    - Window dacc has exactly 3 unique characters, so we calculate the length of this substring:
    - len = i - j = 3 - (-1) = 4. We update ans = 4.

At this point, the window size is valid (3 unique characters). Now we expand the window further.

5. i = 4, character is b, map = {d: 2, a: 1, c: 1, b: 1} (4 unique characters)
    - Since the number of unique characters exceeds k, we need to shrink the window from the left.

Shrink the window (while (j < i)):

1. j = 0, character is d, map = {d: 1, a: 1, c: 1, b: 1} (still 4 unique characters)
2. j = 1, character is d, map = {a: 1, c: 1, b: 1, d: 0} (map value for d becomes 0, we erase d)
    - Now, map = {a: 1, c: 1, b: 1} (3 unique characters)
    - The window acbb has 3 unique characters, and its length is i - j = 4 - 1 = 3. ans remains 4.

Now, we keep expanding again.

```
int main() {
    std::string str = "ddacbbaccdedacebb";
    int k = 3;
    std::cout << LongestSubstringWithAtMostKUniqueCharacters::sol(str, k) << std::endl;
    return 0;
}
```

Second pass of the window expansion:

1. i = 5, character is b, map = {a: 1, c: 1, b: 2} (3 unique characters)
2. i = 6, character is a, map = {a: 2, c: 1, b: 2} (3 unique characters)
3. i = 7, character is c, map = {a: 2, c: 2, b: 2} (3 unique characters)
    - Now we have a valid window of acb. Its length is i - j = 7 - 1 = 7, so we update ans = 7.

Output:-
7