Valid Anagram in C++

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

class ValidAnagrams {
public:
    static bool sol(const std::string& s1, const std::string& s2) {
        std::unordered_map<char, int> map;
        for (char ch : s1) {
            map[ch]++;
        }

        for (char ch : s2) {
            if (map.find(ch) == map.end()) {
                return false;
            } else if (map[ch] == 1) {
                map.erase(ch);
            } else {
                map[ch]--;
            }
        }
        return map.empty();
    }
};

int main() {
    std::string s1 = "abbcaad";
    std::string s2 = "babacda";
    std::cout << (ValidAnagrams::sol(s1, s2) ? "true" : "false") << std::endl;
    return 0;
}
```

Step-by-Step Breakdown:

1. **Input**:
   - `s1 = "abbcaad"`
   - `s2 = "babacda"`
2. **Core Logic**:
   - **`ValidAnagrams::sol` function**:
     - We create an **unordered_map** (`map`) to store the frequency of each character in `s1`.
     - Then, we iterate over the characters in `s2` and check if each character in `s2` is found in `map` (i.e., it should also exist in `s1` with the correct frequency).
     - If a character is found in `map`, we decrease its count. If the count reaches `1`, we remove that character from `map` entirely.
     - If `map` is empty at the end, it means that both strings are anagrams because they contain the same characters with the same frequencies.
     - If `map` is not empty at the end, it means the strings are not anagrams.
3. **Detailed Steps**:
   - **Input strings**: `s1 = "abbcaad"`, `s2 = "babacda"`
   - **First, we populate the frequency map** using `s1`:
     - For `s1`, the map will look like this:
       - `{'a': 3, 'b': 2, 'c': 1, 'd': 1}`
   - **Then, we iterate over the characters in `s2`**:
     - For `s2 = "babacda"`, the process proceeds as follows:
       - For `b`: map = `{'a': 3, 'b': 2, 'c': 1, 'd': 1}` → decrease `b` → map = `{'a': 3, 'b': 1, 'c':`

`1, 'd': 1}`

- For `a`: map = `{'a': 3, 'b': 1, 'c': 1, 'd': 1}` → decrease `a` → map = `{'a': 2, 'b': 1, 'c': 1, 'd': 1}`
- For `b`: map = `{'a': 2, 'b': 1, 'c': 1, 'd': 1}` → decrease `b` → map = `{'a': 2, 'b': 0, 'c': 1, 'd': 1}` → remove `b`
- For `a`: map = `{'a': 2, 'c': 1, 'd': 1}` → decrease `a` → map = `{'a': 1, 'c': 1, 'd': 1}`
- For `c`: map = `{'a': 1, 'c': 1, 'd': 1}` → decrease `c` → map = `{'a': 1, 'd': 1}`
- For `d`: map = `{'a': 1, 'd': 1}` → decrease `d` → map = `{'a': 0}` → remove `d`
- For `a`: map = `{'a': 0}` → decrease `a` → map = `{}` (empty)

4. **Conclusion**:
   - After processing all characters in `s2`, the map is empty, which indicates that the strings `s1` and `s2` are indeed anagrams of each other.

Output:

```
true
```

Output:-
true