Minimum Repeat To Make Substring

solved by	Senan
Platform	GeeksForGeeks
difficulty	Medium
tags	String Manipulation
language	C++
solved on	@08/11/2024
link	https://www.geeksforgeeks.org/problems/minimum-times-a-has-to-be-repeated-such-that-b-is-a-substring-of-it--170645/1
Completion	✓

Intuition

To find how many times we need to repeat `s1` so that `s2` becomes a substring of the repeated `s1`, we can observe that if `s2` is not initially a substring, repeating `s1` enough times will cover all characters in `s2`.

Approach

1. **Check Initial Condition**: If `s2` is already a substring of `s1`, then one repetition of `s1` is enough, and we return `1`.
2. **Calculate Minimum Repeats**: To ensure that the length of the repeated `s1` covers `s2`, calculate the minimum number of times we need to repeat `s1`:
   - Let `n1` and `n2` be the lengths of `s1` and `s2`, respectively.
   - We need to repeat `s1` at least `atleast = n2 / n1` times. If there's a remainder, increment `atleast` by `1`.
3. **Construct and Check**: Build a `larger` string by repeating `s1` `atleast` times and check if `s2` is now a substring of `larger`.
4. **One Additional Check**: If not found, repeat `s1` once more and check again.
5. **Return -1 if Not Found**: If `s2` is still not found, return `1`.

Complexity

Time Complexity:

- Checking if `s2` is a substring in the repeated string has a complexity of **O(NM)**, where **N** is the length of `s1` and **M** is the length of `s2`.
- Since we are repeating `s1` up to `atleast + 1` times, the total length of `larger` will be close to **O(M)**, making the substring check efficient given the limit of two checks.

So, the time complexity can be approximated as **O(NM)** in the worst case.

Space Complexity:

- **O(N + M)** for storing the `larger` string and `s2`.

Code

```
class Solution {
  public:
    int minRepeats(string& s1, string& s2) {
        if(s1.find(s2)!=-1) return 1;
        int n1 = s1.size();
        int n2 = s2.size();

        int atleast = n2/n1;
        if(n2%n1!=0) atleast++;
        string larger;
        larger.reserve((2 + atleast)*s1.size());

        int i = 0;
        while(i < atleast){
            larger += s1;
            i++;
        }

        if(larger.find(s2)!=-1) return atleast;
        larger += s1;

        if(larger.find(s2)!=-1) return atleast+1;
        return -1;
    }
};
```