Intuition:

To solve this equation, we can just find for each element, how many substrings are there in which it is unique.

For example, think about the following string:

X A X X X A X X A L

For the middle A, for it to be unique in a contiguous substring, then we need to avoid including the previous and the next occurrence of ‘A’

Thus, for the left boundary, there are 4 options. For the right boundary, there are 3 options. In total, we can form 4*3 substrings in which ‘A’ is unique. We can apply the same logic to all the element, then the problem is solved.

We will use a vector (size of 26) of vectors. For each english character, we have a vector to keep track of the indices of occurrences.

Then we loop over the vector, and sum up all the distances in the vectors.

The edge case will be the first and the last occurrence, which we can handle by adding extra elements in the beginning and end.

```
class Solution {
public:
    int uniqueLetterString(string s) {
        int n = s.size();
        //Initialize the first position to be -1 to account for the first
occurrence.
        vector<vector<int>>pos(26,{-1});

        // Adding the index for each character appearance
        for (int i=0; i<n; i++)  pos[s[i]-'A'].emplace_back(i);

        // Adding (N+1) to the vector to account for the last occurrence
        for (int i=0; i<n; i++)  pos[s[i]-'A'].emplace_back(n);

        int ret = 0;
        for (int k=0; k<26; k++)
            for (int i=1; i<pos[k].size()-1; i++)
            {
                ret += (pos[k][i]-pos[k][i-1])*(pos[k][i+1]-pos[k][i]);
            }

        return ret;
    }
};
```