3042. Count Prefix and Suffix Pairs I

You are given a **0-indexed** string array `words`.

Let's define a **boolean** function `isPrefixAndSuffix` that takes two strings, `str1` and `str2`:

- `isPrefixAndSuffix(str1, str2)` returns `true` if `str1` is **both** a prefix and a suffix of `str2`, and `false` otherwise.

For example, `isPrefixAndSuffix("aba", "ababa")` is `true` because `"aba"` is a prefix of `"ababa"` and also a suffix, but `isPrefixAndSuffix("abc", "abcd")` is `false`.

Return *an integer denoting the* ***number*** *of index pairs* `(i, j)` *such that* `i < j`, *and* `isPrefixAndSuffix(words[i], words[j])` *is* `true`.

Example 1:

```
Input: words = ["a","aba","ababa","aa"]
Output: 4
Explanation: In this example, the counted index pairs are:
i = 0 and j = 1 because isPrefixAndSuffix("a", "aba") is true.
i = 0 and j = 2 because isPrefixAndSuffix("a", "ababa") is true.
i = 0 and j = 3 because isPrefixAndSuffix("a", "aa") is true.
i = 1 and j = 2 because isPrefixAndSuffix("aba", "ababa") is true.
Therefore, the answer is 4.
```

Example 2:

```
Input: words = ["pa","papa","ma","mama"]
Output: 2
Explanation: In this example, the counted index pairs are:
i = 0 and j = 1 because isPrefixAndSuffix("pa", "papa") is true.
i = 2 and j = 3 because isPrefixAndSuffix("ma", "mama") is true.
Therefore, the answer is 2.
```

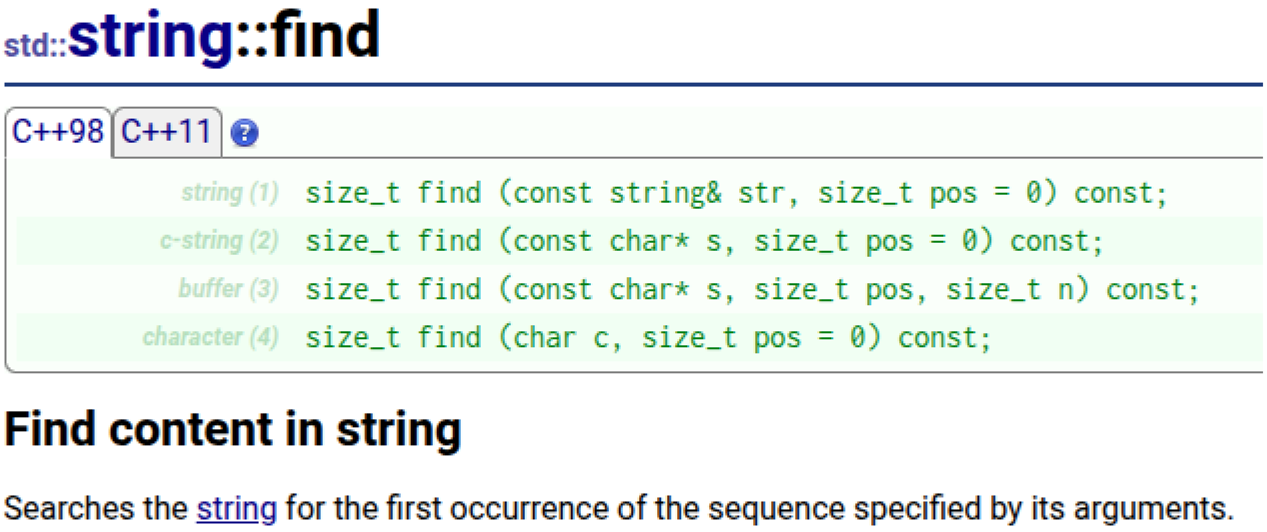
Example 3:

```
Input: words = ["abab","ab"]
Output: 0
Explanation: In this example, the only valid index pair is i = 0 and j = 1, and
isPrefixAndSuffix("abab", "ab") is false.
Therefore, the answer is 0.
```

Constraints:

- `1 <= words.length <= 50`
- `1 <= words[i].length <= 10`
- `words[i]` consists only of lowercase English letters.

3042. Count Prefix and Suffix Pairs I



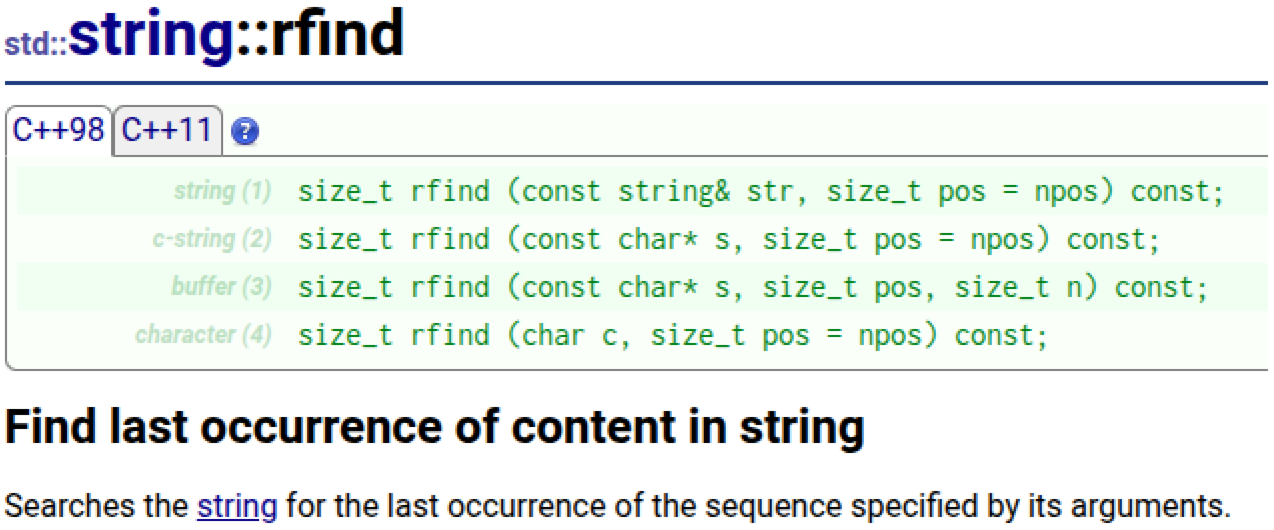
std::string::find

C++98 C++11

string (1)	`size_t find (const string& str, size_t pos = 0) const;`
c-string (2)	`size_t find (const char* s, size_t pos = 0) const;`
buffer (3)	`size_t find (const char* s, size_t pos, size_t n) const;`
character (4)	`size_t find (char c, size_t pos = 0) const;`

Find content in string

Searches the string for the first occurrence of the sequence specified by its arguments.



std::string::rfind

C++98 C++11

string (1)	`size_t rfind (const string& str, size_t pos = npos) const;`
c-string (2)	`size_t rfind (const char* s, size_t pos = npos) const;`
buffer (3)	`size_t rfind (const char* s, size_t pos, size_t n) const;`
character (4)	`size_t rfind (char c, size_t pos = npos) const;`

Find last occurrence of content in string

Searches the string for the last occurrence of the sequence specified by its arguments.

3042. Count Prefix and Suffix Pairs I

```
/*
   STLs find and rfind
   Time compelxity: $O(n^2 . 2m^2)$
   Space compelxity: O(1)
   n: size of the words's list
   m: size of a word in the list of words

*/
class Solution {
   public:
      int countPrefixSuffixPairs(std::vector<std::string>& words) {
         int n=words.size();

         auto is_prefix_and_suffix=[&](std::string& needle,std::string& haystack)->bool{
            int n=haystack.size();
            int m=needle.size();
            if(n<m) return false;
            auto it1=haystack.find(needle);
            auto it2=haystack.rfind(needle);
            return it1==0 && it2==n-m;
         };

         int ans=0;
         for(int i=0;i<n-1;++i){
            for(int j=i+1;j<n;++j){
               if(is_prefix_and_suffix(words[i],words[j])) ans++;
            }
         }

         return ans;
      }
};
```

3042. Count Prefix and Suffix Pairs I

/*

Three pointers

Time compelxity: $O\left(n^{2}.m\right)$

Space compelxity: O(1)

n: size of the words's list

m: size of a word in the list of words

*/

```
class Solution {
   public:
      int countPrefixSuffixPairs(std::vector<std::string>& words) {
         int n=words.size();

         auto is_prefix_and_suffix=[&](std::string& needle,std::string& haystack)->bool{
            int n=haystack.size();
            int m=needle.size();
            if(n<m) return false;
            int i=0,j=n-m,k=0;
            while(k<m && haystack[i]==needle[k] && haystack[j]==needle[k]){
               i++;
               j++;
               k++;
            }
            return k==m;
         };

         int ans=0;
         for(int i=0;i<n-1;++i){
            for(int j=i+1;j<n;++j){
               if(is_prefix_and_suffix(words[i],words[j])) ans++;
            }
         }

         return ans;
      }
};
```

3045. Count Prefix and Suffix Pairs II

Constraints:

- `1 <= words.length <=` 10^5
- `1 <= words[i].length <=` 10^5
- `words[i]` consists only of lowercase English letters.
- The sum of the lengths of all `words[i]` does not exceed `5 * 105`.

3045. Count Prefix and Suffix Pairs II

```
/*
    Prefix tree (Trie)
    Time compelxity: O(n.m)
    Space compelxity: O(n.m)
*/
class Trie{
    public:
        class TrieNode{
            public:
                std::unordered_map<int,TrieNode*> children;
                long long count=0;
        };
        TrieNode* root;
    public:
        Trie(){
            root=new TrieNode();
        }

        ~Trie(){delete_trie(root);}

        // Delete the try to avoid memory leaks
        void delete_trie(TrieNode* root){
            if(!root) return;
            for(auto& [_,node]: root->children){
                delete_trie(node);
            }
            delete root;
        }
long long solve(std::vector<std::string>& words){
            int n=words.size();
            long long ans=0;
            for(auto& word: words){
                TrieNode* cur=root;
                int m=word.size();
                for(int i=0;i<m;++i){
                    int hash=(word[i]-'a')*26+word[m-i-1]-'a';
                    TrieNode* node=cur->children[hash];
                    if(!node){
                        node=new TrieNode();
                        cur->children[hash]=node;
                    }
                    ans+=cur->children[hash]->count;
                    cur=node;
                }
                // The word is completely processed, it could be a prefix and a suffix for another word,
                // so increment its count
                cur->count++;
            }
            return ans;
        }
};
```

```
class Solution {
public:
    long long countPrefixSuffixPairs(std::vector<std::string>& words) {
        Trie trie=Trie();
        return trie.solve(words);
    }
};
```