745. Prefix and Suffix Search

DescriptionHintsSubmissionsDiscussSolution

Given many `words`, `words[i]` has weight `i`.

Design a class `WordFilter` that supports one function, `WordFilter.f(String prefix, String suffix)`. It will return the word with given `prefix` and `suffix` with maximum weight. If no word exists, return -1.

Examples:

```
Input:
WordFilter(["apple"])
WordFilter.f("a", "e") // returns 0
WordFilter.f("b", "") // returns -1
```

Note:

1. `words` has length in range `[1, 15000]`.
2. For each test case, up to `words.length` queries `WordFilter.f` may be made.
3. `words[i]` has length in range `[1, 10]`.
4. `prefix, suffix` have lengths in range `[0, 10]`.
5. `words[i]` and `prefix, suffix` queries consist of lowercase letters only.

Seen this question in a real interview before?

- Difficulty:Hard
- Total Accepted:957
- Total Submissions:4.9K
- Contributor:zestypanda
-
- Subscribe to see which companies asked this question.

  Related Topics Trie Tree

very easy java solution but get time limited exceeded

```
class WordFilter {
```

```
    String[] input;
    public WordFilter(String[] words) {
        input = words;
    }
    public int f(String prefix, String suffix) {
        for(int i = input.length-1; i >= 0; i--){
            if(input[i].startsWith(prefix) && input[i].endsWith(suffix)) return
i;
        }
        return -1;
    }
}
```

Approach #3: Trie of Suffix Wrapped Words [Accepted]

Intuition and Algorithm

Consider the word `'apple'`. For each suffix of the word, we could insert that suffix, followed by `'#'`, followed by the word, all into the trie.

For example, we will insert `'#apple', 'e#apple', 'le#apple', 'ple#apple', 'pple#apple', 'apple#apple'` into the trie. Then for a query like `prefix = "ap", suffix = "le"`, we can find it by querying our trie for `le#ap`.

```
class WordFilter {
    HashMap<String, Integer> map = new HashMap<>();

    public WordFilter(String[] words) {
        for(int w = 0; w < words.length; w++){
            for(int i = 0; i <= 10 && i <= words[w].length(); i++){
                for(int j = 0; j <= 10 && j <= words[w].length(); j++){
                    map.put(words[w].substring(0, i) + "#" +
words[w].substring(words[w].length()-j), w);
                }
            }
        }
    }

    public int f(String prefix, String suffix) {
        return (map.containsKey(prefix + "#" + suffix))? map.get(prefix + "#" +
suffix) : -1;
    }
}
```

```
// C++

#include<stdio.h>

#include<iostream>

#include<sstream>
```

```
#include<vector>
#include<algorithm>
#include<unordered_map>
#include<limits.h>
#include<queue>
#include<unordered_map>
#include<unordered_set>
#include<stack>
#include<string.h>
using namespace std;
typedef struct Trie
{
     int mx;
     unordered_map<char,struct Trie*> next;
     Trie()
     {
          mx=-1;
     }
}Trie;

void insert(Trie *trie, string word, int x)
{
     Trie *p = trie;
     for(int i=0;i<(int)word.size();i++)
     {
          char tmp = word[i];
          if(p->next.count(tmp)==0)
          {
               p->next[tmp]=new Trie();
          }
          p = p->next[tmp];
          p->mx = max(p->mx,x);
     }
}

int find(Trie *trie, string word)
```

```
{
     Trie *p = trie;
     for(int i=0;i<(int)word.size();++i)
     {
          char tmp = word[i];
          if(p->next.count(tmp)==0) return -1;
          p=p->next[tmp];
     }
     return p->mx;
}

Trie *root;

void WordFilter(vector<string> words) {
     root = new Trie();
     for(int i=0;i<(int)words.size();++i)
     {
          int m = min((int)words[i].size(),10);
          string r = words[i].substr(words[i].size()-m) + "#";
          reverse(r.begin(),r.end());
          string t = "";

          for(int j=0;j<m;++j)
          {
               insert(root,t+r,i);
               t+=words[i][j];
               cout << t+r << endl;
          }
          insert(root,t+r,i);
     }
}

int f(string prefix, string suffix) {
     reverse(suffix.begin(), suffix.end());
     return find(root,prefix + "#" + suffix);
}
```