Determine If One String Is Another's Substring (Medium)

Determine if a small string is a substring of another large string.

Return the index of the first occurrence of the small string in the large string.

Return -1 if the small string is not a substring of the large string.

Assumptions

- Both large and small are not null
- If small is empty string, return 0

Examples

- “ab” is a substring of “bcabc”, return 2
- “bcd” is not a substring of “bcabc”, return -1
- "" is substring of "abc", return 0

```
#define h 26
int substrSearch(string txt, string pat){
    int patLen = pat.length();
    int txtLen = txt.length();
    int t = 0, p = 0, q = 1;

    // compute pow(h, patLen - 1)
    for(int i = 0; i < patLen - 1; i++)
        q = q * h;
    // compute the hash value of txt and pat window
    for(int i = 0; i < pat.length(); i++){
        t =  t * h + (txt[i] - 'a');
        p = p * h + (pat[i] - 'a');
    }
    // move the window from left to right and compute new hash value for txt
window
    for(int i = 0; i <= txtLen - patLen; i++){
        if(t == p)
            return i;
        t = (t - (txt[i] - 'a') * q) * h + (txt[i + patLen] - 'a');
    }
    return -1;
}
```

Time Complexity: O(n), since we traverse the string once.

//Ok, no problem.
//这个 hash 碰撞的概率会比较大，因为过于简单，建议去学习一下双 hash 的方法，比较保险。hash 这一块其实主要就是学习一些神奇的 hash 算法去避免碰撞，可以多学一点比较常用的。

Longest Common Prefix (Medium)

Write a function to find the longest common prefix string amongst an array of strings.

```
string commonPrefix(string a, string b){
    int index;
    while(index < a.length() && index < b.length()){
        if(a[index] != b[index])
            break;
        else
            index++;
    }
    return a.substr(0, index);
}

string longestCommonPrefix(vector<string> strs){
    string prefix = strs[0];
    for(int i = 1; i < strs.size(); i++)
        prefix = commonPrefix(prefix, strs[i]);
    return prefix;
}
```

Time Complexity: O(nm), where n is the number of strings, m is the length of the largest string.

//Ok, great!

One Edit Distance (Medium)

Determine if two given Strings are one edit distance.

One edit distance means you can only insert one character/delete one character/replace one character to another character in one of the two given Strings and they will become identical.

Assumptions:

- The two given Strings are not null

Examples:
s = "abc", t = "ab" are one edit distance since you can remove the trailing 'c' from s so that s and t are identical
s = "abc", t = "bcd" are not one edit distance

```
bool oneEditDistance(string str1, string str2){
    int m = str1.length();
    int n = str2.length();
    if(abs(m - n) > 1)
        return false;
    int i = 0, j = 0;
    int count = 0;
    while(i < m && j < n){
        if(str1[i] != str2[j]){
            count++;
            if(count == 2)
                return false;
            if(m > n)
                i++;
            else if(n > m)
                j++;
            else{
                i++;
                j++;
            }
        }
        else{
            i++;
            j++;
        }
    }
    while(i < m){
        i++;
        count++;
    }
    while(j < n){
        j++;
        count++;
    }
    return count == 1;
}
```

Time Complexity: O(n), where n is the length of the larger string.
//Ok, good!

Remove Certain Characters (Easy)

Remove given characters in input string, the relative order of other characters should be remained. Return the new string after deletion.

Assumptions

- The given input string is not null.
- The characters to be removed is given by another string, it is guaranteed to be not null.

Examples

- input = "abcd", t = "ab", delete all instances of 'a' and 'b', the result is "cd".

```
// remove a specific char from string
void removeChar(string& s, char c){
    int slow = 0, fast = 0;
    while(1){
        while(fast < s.size() && s[fast] == c)
            fast++;
        if(fast == s.size())
            break;
        while(fast < s.size() && s[fast] != c)
            s[slow++] = s[fast++];
    }
    s.resize(slow);
    return;
}

void remove(string&s, string t){
    for(int i = 0; i < t.length(); i++)
        removeChar(s, t[i]);
    return;
}
```

Time Complexity: O(nm), where m, n is the length of the two strings.
//Ok, no problem.