Leetcode\_76\_MinimumWindowSubstring\_最小窗口子串\_Hard

# Leetcode\_76\_MinimumWindowSubstring\_最小窗口子串\_Hard

## 题目描述

\* 难度：Hard

\* https://leetcode.com/problems/minimum-window-substring/description/

\* Given a string S and a string T, find the minimum window in S

\* which will contain all the characters in T in complexity O(n).

\* Example:

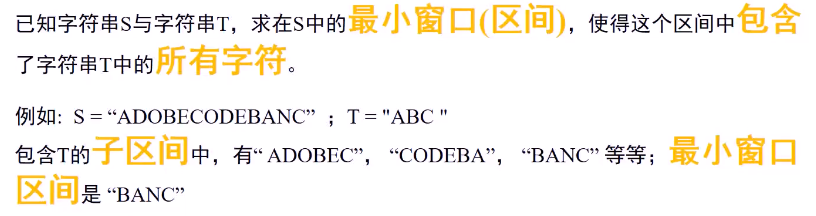
\* Input: S = "ADOBECODEBANC", T = "ABC"

\* Output: "BANC"

**\* Note:**

\* If there is no such window in S that covers all characters in T,return the empty string "".

\* If there is such window, you are guaranteed that there will always be only one unique minimum window in S.



## 思路分析

\* 思路分析：双int[]字符哈希表和双指针最小窗口。需要两个int[]对S和T进行字符次数统计。

\*首先统计T字符串的字符数目；然后利用双指针构造滑动窗口；

\*right右移一个字符，首先hashMap\_S对该字符数目加1，然后

\*尽可能地移动left：通过while实现，移动条件有两个：hashMap\_T中不存在left字符或者hashMap\_S中该字符数目大于hashMap\_T；

\*然后再判断当前窗口是否满足要求且是否比上次更短，若是，则更新，否则continue。

## Java代码

public String **minWindow**(String s, String t) {

if(s == null||t==null||s.length()==0||t.length() == 0||s.length() < t.length()) return "";

int[] hashMap\_T = new int[128];

int[] hashMap\_S = new int[128];

char[] chars\_t = t.toCharArray();

for(char c:chars\_t) hashMap\_T[c]++;//对T字符串进行hash统计计算

char[] chars\_s = s.toCharArray();

int left = 0,right = 0;//双指针，表示窗口的左和右端点

int windowLen = chars\_s.length;

String result = "";

for(;right < chars\_s.length;right++){//遍历S

hashMap\_S[chars\_s[right]]++;

//尽可能让left向右移动

//两种情况移动：left字符不在T中和left字符出现次数大于T中该字符次数

while((left < right) && (hashMap\_T[chars\_s[left]] == 0|| hashMap\_S[chars\_s[left]]>hashMap\_T[chars\_s[left]])){

hashMap\_S[chars\_s[left]]--;

left++;

}

int currentLen = right-left+1;

if(currentLen < chars\_t.length) continue;

if(windowLen >= currentLen && doesScoversT(hashMap\_S,hashMap\_T)){//一定包括等号，因为若S与T相等情况

windowLen = currentLen;

result = new String(chars\_s,left,right-left+1);//保存

}

}

return result;

}

/\*\*

\* 判断此时窗口是否已经包含T的所有字符

\*/

public boolean **doesScoversT**(int[] hashMap\_S,int[] hashMap\_T){

for(int key = 0;key < hashMap\_T.length;key++)

if(hashMap\_T[key] > hashMap\_S[key])

return false;

return true;

}

