滑动窗口算法是一种适用于数组的一种算法

> 滑动窗口的实质就是一个双指针的问题，右指针向右移动扩大窗口则移进字符，左指针向右移动则为缩小窗口,移出字符

滑动窗口算法的基本框架如下：

```
void slidingWindow(string s,string t){
 undered_map<char,int> need,window;
 int left=right=0;
 while(right<s.size()){
 //c是将移入窗口的字符
 char c=s[right];
 //扩大窗口
 right++;
 //进行窗口内数据的一系列更新
  ....
 /*******************debug*********************/
 printf("window:[%d,%d)\n",left,right);
 /*********************************************/
 while(window needs shrink){//窗口需要缩小
 //c为需要移除的字符
 char c=s[left];
 //减小窗口大小
 left++;
 //对窗口内数据的一系列更新
 ...
 }
 }
}

...部分就是具体逻辑的地方，前后两部分的...会发现是完全对称的
```

例题1：给你两个字符串S和T，请你在S中找到包含T中全部字母的最短子串

> 输入=“ADBECFEBANC” T="ABC" 算法应该返回"BANC"
> **滑动窗口算法的基本框架是：**
> **1：初始化left=right=0，把索引左闭右开区间[left,right)称为一个“窗口”**
> **2：我们要不断地增加right指针扩大[left,right)窗口，直到窗口内的字符串满足要求---这一步是找到可行解**
> **3:停止移动right指针，转而不断增加left指针缩小[left,right)，直到窗口内的字符串不符合要求---找到最优解,每次增加left，我们都要更新一轮结果**
> 4:重复2，3步骤，直到right到达字符串s的尽头.

```
需要两个数组
needs表示所要满足的条件
```

```
windows表示窗口内已经有什么字符了
valid变量表示窗口中满足need条件的字符个数,如果valid和need.size()的大小相同，则说明
窗口已满足条件，已经完全覆盖了子串
```

```
string minWindow(string s,string t){
 unordered_map<char,int> need,window;
 for(char c:t)need[c]++;
 int left=right=0;
 int valid=0;//判断是否更新最小覆盖子串起始索引和长度
 //记录最小覆盖子串的起始索引及长度
 int start=0,len=INT_MAX;

 while(right<s.szie()){
  char c=s[right];
  right++;//一定要先取出s[right]后再自加才满足左闭右开的索引
  if(need.count(c)){
  window[c]++;
  if(window[c]==need[c])
  valid++;
  }
  //倘若增加right扩大窗口找到可行解了
  while(valid==need.size()){//移动左指针，缩小滑动窗口，找到最优解
  if(right-left<len){
  start=left;
  len=right-left;
  }
  char c=s[left];
  left++;
  if(need.count(c)){
  if(window[c]==need[c]){
  valid--;
  }
  window[c]--;
  }
  }
  }
  return len==INT_MAX?
  “”: s.substr(start,len);
}
```

例题2输入两个字符串S和T，请你用算法判断S是否包含T的排列

```
bool checkIncludsion(string S,string T){
 unordered_map<char,int>need,window;
 for(char c:T) need[c]++;
 int valid=0;
 int left=right=0;
 while(right<S.size()){
 char c=S[right];
 right++;
 //进行窗口内数据的一系列更新
  if(need.count(c)){
  window[c]++;
  if(window[c]==need[c])
  valid++;
  }

  while((right-left)>=T.size()){
  if(valid==need.size()) return true;
  char c=S[left];
  left++;
  if(need.count(c)){
  if(window[c]==need[c]) valid--;
  window[c]--;
  }
  }
  }
  return false;
}
```

例题4:最长无重复子串

输入一个字符串S，请计算S中不包含重复字符的最长字符串长度

```
int lengthOfLongestSubString(string s){
 unordered_map<char,int> window;
 int left=right=0;
 int res=0;
 while(right<s.size()){
 char c=s[right];
 right++;
 window[c]++;
```

```
while(window[c]>1){//出现重复
 char a=s[left];
 left++;
 window[a]--;
 }
 res=max(res,right-left);
 }
 return res;
}
```