滑动窗口

> 链表子串数组题，用双指针别犹豫；
> 双指针家三兄弟，个个都是万人迷；
> 快慢指针最神奇，链表操作无压力；
> 归并排序找中点，链表成环搞判断；
> 左右指针最常见，左右两端相向行；
> 反转数组要靠他，二分搜索是弟弟；
> 滑动窗口老孟男，子串问题全靠他；
> 左右指针滑窗口，一前一后齐头进。

算法框架：

```
int left = 0, right = 0;
while(right < s.size()) {
  // 增大窗口
  windows.push_back(s[right]);
  right++;

  while(windows need shrink) {
    // 缩小窗口
    windows.pop_back();
    left++;
  }
}
```

整理算法框架如下：

```
void slideWindows(std::string s, std::string t){
  std::unordered_map<char, int> need, window;
  for(auto c : t){
    need[c]++;
  }

  int left =0, right = 0;
  int valid = 0;
  while(right < s.size()){
    // c 是将移入窗口的字符串
    char c = s[right];
    // 右移窗口
    right++;
    // 将窗口内数据进行一系列更新
    ...

    // 判断左侧窗口是否要收缩
    while(window need shirk){
      // d 是将移出窗口的字符
```

```
        char d = s[left];
        // 左侧收缩
        left++;
        // 进行窗口内数据更新
        ...
      }
    }
}
```

最小覆盖字符串

给定字符串s和t，求解s中包含t所有字符的最短子串，不存在返回空。

```
class Solution {
public:
  std::string minWindow(std::string s, std::string t) {
    std::unordered_map<char, int> need, windows;
    for (auto c : t) {
      need[c]++;
    }

    int left = 0, right = 0;
    int valid = 0;
    int start = 0;
    int len   = INT_MAX;
    while (right < s.size()) {
      char c = s[right++];
      if (need.count(c)) {
        windows[c]++;

        if (windows[c] == need[c]) {
          valid++;
        }
      }

      // 是否进行收缩，未限定子串长度
      while (valid == need.size()) {
        // 更新最小长度
        if (right - left < len) {
          start = left;
          len   = right - left;
        }
        char d = s[left++];
        if (windows.count(d)) {
          if (windows[d] == need[d]) {
            valid--;
          }
          windows[d]--;
        }
```

```
      }
    }

    return len == INT_MAX ? "" : s.substr(start, len);
  }
};
```

字符串排列

给定字符串s1,s2; 判断s2中是否包含s1的一个排列。

```
class Solution {
public:
  bool checkInclusion(std::string s1, std::string s2) {
    std::unordered_map<char, int> need, window;
    for (auto c : s1) {
      need[c]++;
    }
    int left = 0, right = 0;
    int valid = 0;
    while (right < s2.size()) {
      char c = s2[right++];
      if (need.count(c)) {
        window[c]++;
        if (window[c] == need[c]) {
          valid++;
        }
      }

      // 保证了每次s1.size() == right - left
      while (s1.size() <= right - left) {
        if (valid == need.size()) { // 出现含全部字符的子串
          return true;
        }

        char d = s2[left++];
        if (window.count(d)) {
          if (window[d] == need[d]) {
            valid--;
          }
          window[d]--;
        }
      }
    }

    return false;
  }
};
```

求字符串中包含排列的全部起始索引位置

给定字符串s1,s2; 判断s2中包含s1的一个排列, 输出该排列在s2中的全部开始索引位置。

```
class Solution {
public:
  std::vector<int> findAnagrams(std::string s, std::string p) {
    std::unordered_map<char, int> need, window;
    std::vector<int>              res;
    for (auto c : p) {
      need[c]++;
    }
    int right = 0, left = 0;
    int valid = 0;
    while (right < s.size()) {
      char c = s[right++];
      if (need.count(c)) {
        window[c]++;
        if (window[c] == need[c]) {
          valid++;
        }
      }

      while (right - left >= p.size()) {
        if (valid == need.size()) {
          res.push_back(left);
        }

        char d = s[left++];
        if (window.count(d)) {
          if (window[d] == need[d]) {
            valid--;
          }
          window[d]--;
        }
      }
    }

    return res;
  }
};
```

最长无重复子串

给定一个字符串，找出其中不含重复字符的最长子串长度。

```cpp
class Solution {
public:
  int lengthOfLongestSubstring(std::string s) {
    std::unordered_map<char, int> window;
    int                           res   = 0;
    int                           right = 0, left = 0;

    while (right < s.size()) {
      char c = s[right++];
      window[c]++;

      while (window[c] > 1) {
        char d = s[left++];
        window[d]--;
      }

      res = res < right - left ? right - left : res;
    }

    return res;
  }
};
```