算法讨论班第 31 期-郭清沛

2016 年 05 月 15 日

76. Minimum Window Substring Difficulty: Hard

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).

For example,

S = "ADOBECODEBANC"

T = "ABC"

Minimum window is "BANC".

Note:

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

If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

思路：

由于题目中所说的包含是无序的，也就是数量包含，与位置无关，因此我们可以采用 2 个计数器，分别对已有字符的出现次数和目标出现次数进行比较

已有字符出现次数：当前 p1,p2 指针之间的字符，各自出现的次数

目标出现次数：目标字符串 t 中包含的各个字符出现的次数

如果已有字符的出现次数全部都>=目标中各个字符的出现次数就说明已经包含了目标

采用类似于滑动窗口的思想，用双指针[p1,p2]定义一个窗口，开始时,p2 不断向后滑动，同时统计[p1,p2]之间字符的出现次数 ,如果 p1,p2 中所有的字符出现的次数都>=目标字符中出现次数，那么[p1,p2]此时已经包含了目标；

由于题目要求最小窗口,当包含目标后，以后保持[p1,p2]窗口满足目标的前提下看能不能将 p1 往前移动仍然维持包含目标的性质，如果能缩减 p1 则对 p1

进行缩减，并在缩减后更新子串

代码：

```
class Solution {
public:
    string minWindow(string s, string t) {
        if (s.length() < t.length()) return "";
        int p1 = 0, p2 = 0;//two pointers
        int target[128]={0},current[128]={0};//target记录目标字符出现次数，current记录[p1,p2]之间字符的出现次数
        for (char c : t)
            target[c]++;
        int cnt = 0;//用cnt表示完全包含时的字母
        for (int i = 0; i < 128; i++)
            if (target[i] != 0) cnt++;
        string ret = "";
        while (p2 < s.length())
        {
            //移动p2到完全包含
            if(current[s[p2]]<target[s[p2]]&&current[s[p2]]+1==target[s[p2]])cnt--;//只有边沿触发时才会导致需要的字母个数减少1个
            current[s[p2]]++;
            if(cnt!=0)//cnt>0说明此时窗口还没有完全包含
            {
                p2++;
                continue;
            }
            //cnt==0说明此时窗口已经完全包含,看能不能收缩p1
            while (p1<=p2)
            {
                if (current[s[p1]] - 1 >= target[s[p1]])
                {
                    current[s[p1]]--;
                    p1++;
                }
                else break;
            }
            //收缩之后更新子串
            if(ret.empty()||(!ret.empty()&&p2 - p1 + 1 <= ret.length()))
                ret = s.substr(p1, p2 - p1 + 1);
            p2++;
        }
        return ret;
    }
};
```

79.Word Search

Difficulty: Medium

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example,

Given board =[

['A','B','C','E'],

['S','F','C','S'],

['A','D','E','E']

]

word = "ABCCED", -> returns true,

word = "SEE", -> returns true,

word = "ABCB", -> returns false.

思路：

回溯法遍历所有可能，当前节点的分支只要有一种可能返回为 true,当前节点就返回为 true.

用一个辅助数组 isVisited 保存当前节点是否被遍历过

```
8   bool exist(vector<vector<char>>& board, string word) {
9       if (board.empty()) return false;
10      int rows = board.size(), cols = board[0].size();
11      if (rows*cols < word.size()) return false;
12      vector<vector<bool>>  isVisited(rows, vector<bool>(cols, false));
13      for (int i = 0; i < rows; i++)
14      {
15          for (int j = 0; j < cols; j++)
16          {
17              if (board[i][j] == word[0])
18              {
19                  isVisited[i][j] = true;
20                  if (dfs(i, j, board, word, 1, isVisited))
21                      return true;
22                  isVisited[i][j] = false;
23
24              }
25
26          }
27      }
28      return false;
29  }
```

```
    bool dfs(int cur_row, int cur_col, vector<vector<char>>& board, string& word, int cur, vector<vector<bool>> & isVisited)
    {
        if (cur == word.length()) return true;
        int rows = board.size(), cols = board[0].size();
        int dx[4] = { 0, -1, 0, 1 }, dy[4] = { 1, 0, -1, 0 };//4个方向
        for (int i = 0; i < 4; i++)
        {
            int nxt_row = cur_row + dx[i], nxt_col = cur_col + dy[i];
            //4个方向中是否有满足要求的
            if (nxt_row >= 0 && nxt_row < rows&&nxt_col >= 0 && nxt_col < cols&&isVisited[nxt_row][nxt_col] == false && word[cur] ==
                board[nxt_row][nxt_col])
            {
                isVisited[nxt_row][nxt_col] = true;
                if (dfs(nxt_row, nxt_col, board, word, cur + 1, isVisited))//回溯法寻找
                    return true;
                isVisited[nxt_row][nxt_col] = false;
            }
        }
        return false;
    }
};
```

1. Word Search II

Difficulty: Hard

Given a 2D board and a list of words from the dictionary, find all words in the board.

Each word must be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

For example,

Given words = ["oath","pea","eat","rain"] and board =

[

['o','a','a','n'],

['e','t','a','e'],

['i','h','k','r'],

['i','f','l','v']

]

Return ["eat","oath"].

Note:

You may assume that all inputs are consist of lowercase letters a-z.

思路：

intuition:暴力搜索解,每当来一个字符，就需要遍历 words,看是哪些 word 的前缀

用前缀树来对待查找的 words 进行归纳，每当移动一个新的位置，看加上新的位置所构成的单词能否在前缀树中找到，

如果能找到，就可以继续往下找；知道找到加上新位置的字符为前缀树的叶子节点(单词节点)

如果不能找到，继续往下找已经不可能构成已有单词，就剪断分支。

```
75  class Solution {
76  public:
77      vector<string> findWords(vector<vector<char>>& board, vector<string>& words) {
78          vector<string> ret;
79          if (board.empty()) return ret;
80          int rows = board.size(), cols = board[0].size();
81          //插入待查找的words到trieTree中
82          TriTree prefixTree = TriTree();
83          for (string s : words)
84              prefixTree.insertWord(s);
85          vector<vector<bool>> isVisit(rows, vector<bool>(cols, false));
86          set<string> res;
87          for (int i = 0; i < rows; i++)
88          {
89              for (int j = 0; j < cols; j++)
90              {
91                  if (prefixTree.isPrefix(string(1,board[i][j])))
92                  {
93                      isVisit[i][j]= true;
94                      dfs(i, j, board, string(1,board[i][j]), prefixTree, res,isVisit);
95                      isVisit[i][j]= false;
96                  }
97
98              }
99          }
100         for (string s : res)
101             ret.push_back(s);
102         return ret;
103     }
```

```
105 |   void dfs(int cur_row, int cur_col, vector<vector<char>>& board, string tmp, TriTree  & prefixTree, set<string>& res,vector<vector
        <bool>>& isVisit)
106     {
107         if (prefixTree.isWord(tmp)) res.insert(tmp);
108         int dx[4] = { 0, -1, 0, 1 }, dy[4] = { 1, 0, -1, 0 };
109         for (int i = 0; i < 4; i++)
110         {
111             int next_row = cur_row + dx[i], next_col = cur_col + dy[i];
112             if (next_row >= 0 && next_row < board.size() && next_col >= 0 && next_col < board[0].size() &&
113             isVisit[next_row][next_col]==false&&prefixTree.isPrefix(tmp + board[next_row][next_col]))
114             {
115                 isVisit[next_row][next_col]=true;
116                 dfs(next_row, next_col, board, tmp + board[next_row][next_col], prefixTree, res,isVisit);
117                 isVisit[next_row][next_col]=false;
118             }
119
120         }
121     }
```

154.

Find Minimum in Rotated Sorted Array II

Difficulty: Hard

Follow up for "Find Minimum in Rotated Sorted Array":

What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

The array may contain duplicates.

33. Search in Rotated Sorted Array ★

Total Accepted: **99425** Total Submissions: **328646** Difficulty: **Hard**

> Suppose a sorted array is rotated at some pivot unknown to you beforehand.
>
> (i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).
>
> You are given a target value to search. If found in the array return its index, otherwise return -1.
>
> You may assume no duplicate exists in the array.
>
> Subscribe to see which companies asked this question

81. Search in Rotated Sorted Array II ★

Total Accepted: **61442** Total Submissions: **192505** Difficulty: **Medium**

> Follow up for "Search in Rotated Sorted Array":
>
> What if *duplicates* are allowed?
>
> Would this affect the run-time complexity? How and why?
>
> Write a function to determine if a given target is in the array.

当有重复元素时，这时题目只会要求你判断 target 是否存在，这时只用将 nums[mid]与 nums[right]进行比较，既可以确定 target 落在哪个区间，

若 nums[mid]==nums[right],将 right 往后移动

比如：{ 1，3，1，1，1，1 }

如果 A[right] <= A[mid] 条件就不能确定[start mid]区间为递增有序序列，我们就把该条件分成两个子条件：

A[right] < A[mid] 则 [start mid]区间为递增有序序列

A[right] = A[mid] 则[right mid]区间不能确定，那就 right--，往下一步看看即可

以上 2 题代码完全相同！！

```cpp
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int left=0,right=nums.size()-1;
        int mid = left+(right-left)/2;
        while(left<=right)
        {
            if(nums[mid]==target)return mid;
            else if(nums[mid]>nums[right])//中间元素>左边元素，[left,mid]是一个递增区间
            {
                if(target>=nums[left]&&target<=nums[mid])
                    right = mid-1;//因为前面已经确定mid也不是target,所以right可以跳过mid
                else
                    left = mid+1;
            }
            else if(nums[mid]<nums[right])//中间元素<左边元素，[mid,right]是一个递增区间
            {
                if(target>=nums[mid]&&target<=nums[right])
                    left = mid+1;
                else
                    right = mid-1;
            }
            else//nums[mid]==nums[right]
                right--;
            mid = left+(right-left)/2;
        }
        return -1;
    }
};
```

153. Find Minimum in Rotated Sorted Array ★

Total Accepted: **89094** Total Submissions: **246488** Difficulty: **Medium**

> Suppose a sorted array is rotated at some pivot unknown to you beforehand.
>
> (i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).
>
> Find the minimum element.
>
> You may assume no duplicate exists in the array.

154. Find Minimum in Rotated Sorted Array II ★

Total Accepted: **49817** Total Submissions: **145328** Difficulty: **Hard**

> *Follow up* for "Find Minimum in Rotated Sorted Array":
> What if *duplicates* are allowed?
>
> Would this affect the run-time complexity? How and why?

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).

Find the minimum element.

The array may contain duplicates.

注意：以上两题都是完全相同的代码：

```
class Solution {
public:
    int findMin(vector<int>& nums) {
        if(nums[nums.size()-1]>nums[0])//not rotated
            return nums[0];
        int l = 0,r =nums.size()-1,mid = l+(r-l)/2;
        //rotated
        while(l<=r)
        {
            if(nums[mid]>nums[r])//一定在左侧上升沿,因为nums[r]<nums[mid],
                l = mid+1;
            else if(nums[mid]<nums[r])//一定在右侧上升沿
                r = mid;
            else//no idea
                r--;//mid==r,可以排除r
            mid = l+(r-l)/2;
        }
        return nums[l];

    }
};
```

233.Number of Digit One

Difficulty: Hard

Given an integer n, count the total number of digit 1 appearing in all non-negative integers less than or equal to n.

For example:

Given n = 13,

Return 6, because digit 1 occurred in the following numbers: 1, 10, 11, 12, 13.

思路：

比如 8012

从低位到高位依次确定相应位为 1 的数字个数：每位为 1 的数个数，与高位数字，低位数字和相应位本身有关

步骤如下:

第 1 位：2　高位：802 低位：0 相应位:2　因为 2 >1: 0001-8011 共有 802*1 =　802 个 1(相应位>1 时，1 的个数只与高位有关)

第 2 位: 1　高位: 80　低位: 2 相应位:1　因为 1==1: 考虑高位:80(0-79) 1 2(0-2)

80*10+3 = 803 个 1(相应位=1 时，1 的个数与高位和低位有关)

第 3 位 :0　高位: 8　低位:12 相应位:0　因为 0==0 :相应位为 1 的个数：(0-7) 1 (00-99)

8*100 = 800(相应位=1 时，1 的个数只与高位有关)

第 4 位：8　高位: 0　低位:012 相应位:8　因为 8>1: 1(000-999) 1*1000 = 1000 个

因此总共有: 802+803+800+1000 = 3405 个 1

```
case 0:totalNum+=highNum*ifactor;break;

case 1:totalNum += (highNum*ifactor+lowNum+1);break;
```

default:totalNum+=(highNum+1)*ifactor;break;

1）注意提取高位，低位和当前位的方法：

2）需要注意溢出的问题：在所有的数都用 long long 型表示

```
class Solution {
public:
    int countDigitOne(int n) {
        if(n<=0) return 0;
        int cur_digit = 0;
        long long highNum = 0,lowNum =0,curNum=0;
        long long ifactor = 1;
        long long totalNum = 0;
        while(n/ifactor)//循环提取每个位上的数字，以及高位和低位数字
        {
            curNum = (n/ifactor)%10;
            lowNum = n%ifactor;
            highNum = n/10/ifactor;
            switch(curNum)
            {
                case 0:totalNum+=highNum*ifactor;break;
                case 1:totalNum += (highNum*ifactor+lowNum+1);break;
                default:totalNum+=(highNum+1)*ifactor;break;
            }
            ifactor*=10;
        }
        return totalNum;
    }
};
```

下周题目：

135 ,138, 146,273,301

主讲人：李耀宗