1160. 拼写单词（简单）

1. 题目描述

   给你一份『词汇表』（字符串数组） `words` 和一张『字母表』（字符串） `chars`。

   假如你可以用 `chars` 中的『字母』（字符）拼写出 `words` 中的某个『单词』（字符串），那么我们就认为你掌握了这个单词。

   注意：每次拼写（指拼写词汇表中的一个单词）时，`chars` 中的每个字母都只能用一次。

   返回词汇表 `words` 中你掌握的所有单词的 **长度之和**。

   示例 1：

   ```
   输入：words = ["cat","bt","hat","tree"], chars = "atach"
   输出：6
   解释：
   可以形成字符串 "cat" 和 "hat"，所以答案是 3 + 3 = 6。
   ```

   示例 2：

   ```
   输入：words = ["hello","world","leetcode"], chars = "welldonehoneyr"
   输出：10
   解释：
   可以形成字符串 "hello" 和 "world"，所以答案是 5 + 5 = 10。
   ```

   提示：

   1. `1 <= words.length <= 1000`
   2. `1 <= words[i].length, chars.length <= 100`
   3. 所有字符串中都仅包含小写英文字母

2. 比赛实现

```cpp
class Solution {
public:
    int countCharacters(vector<string>& words, string chars) {
        unordered_map<char, int> cnt;
        for(int i = 0; i < chars.size(); i++)
            cnt[chars[i]]++;
        int ans = 0;
        for(int i = 0; i < words.size(); i++){
            unordered_map<char, int> cur = cnt;
            int idx = 0;
            int len = words[i].size();
            for(; idx < len; idx++){
                if(cur[words[i][idx]] > 0)
                    cur[words[i][idx]]--;
                else
                    break;
```

```
                }
                if(idx == len)
                    ans += len;
            }
            return ans;
        }
};
```

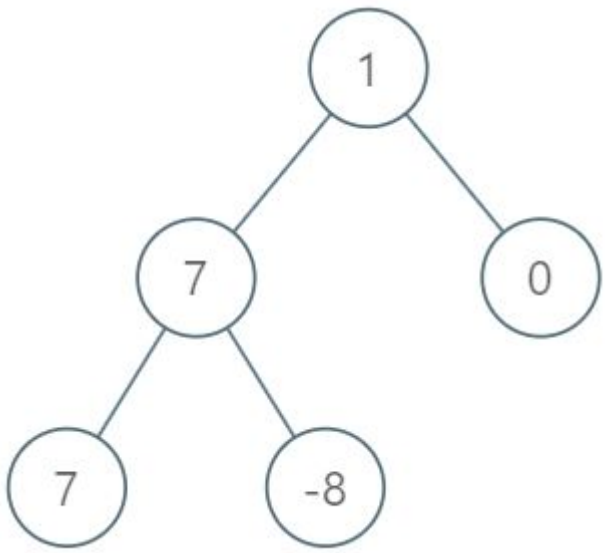
1161. 最大层内元素和（中等）

1. 题目描述

   给你一个二叉树的根节点 `root`。设根节点位于二叉树的第 `1` 层，而根节点的子节点位于第 `2` 层，依此类推。

   请你找出层内元素之和 **最大** 的那几层（可能只有一层）的层号，并返回其中 **最小** 的那个。

   示例：




```
输入：[1,7,0,7,-8,null,null]
输出：2
解释：
第 1 层各元素之和为 1，
第 2 层各元素之和为 7 + 0 = 7，
第 3 层各元素之和为 7 + -8 = -1，
所以我们返回第 2 层的层号，它的层内元素之和最大。
```

   提示：

   1. 树中的节点数介于 `1` 和 `10^4` 之间
   2. `-10^5 <= node.val <= 10^5`

2. 比赛实现

   层序遍历+求和

```
class Solution {
```

```
public:
    int maxLevelSum(TreeNode* root) {
        int ans = -1;
        int max_sum = INT_MIN;
        queue<TreeNode*> q;
        q.push(root);
        int level = 1;
        while(!q.empty()){
            int sum = 0;
            int size = q.size();
            for(int i = 0; i < size; i++){
                TreeNode* cur = q.front();
                q.pop();
                sum += cur->val;
                if(cur->left)
                    q.push(cur->left);
                if(cur->right)
                    q.push(cur->right);
            }
            if(sum > max_sum){
                max_sum = sum;
                ans = level;
            }
            level++;
        }
        return ans;
    }
};
```

1162. 地图分析（中等）

1. 题目描述

   你现在手里有一份大小为 N x N 的『地图』（网格） `grid`，上面的每个『区域』（单元格）都用 `0` 和 `1` 标记好了。其中 `0` 代表海洋，`1` 代表陆地，你知道距离陆地区域最远的海洋区域是是哪一个吗？请返回该海洋区域到离它最近的陆地区域的距离。

   我们这里说的距离是『曼哈顿距离』（ Manhattan Distance）：`(x0, y0)` 和 `(x1, y1)` 这两个区域之间的距离是 `|x0 - x1| + |y0 - y1|` 。

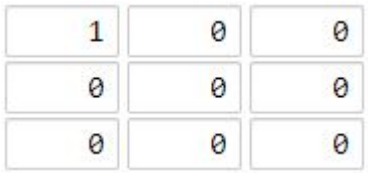
   如果我们的地图上只有陆地或者海洋，请返回 `-1`。

   示例 1：

1	0	1
0	0	0
1	0	1

```
输入：[[1,0,1],[0,0,0],[1,0,1]]
输出：2
解释：
海洋区域（1，1）和所有陆地区域之间的距离都达到最大，最大距离为 2。
```

示例 2：



1	0	0
0	0	0
0	0	0

```
输入：[[1,0,0],[0,0,0],[0,0,0]]
输出：4
解释：
海洋区域（2，2）和所有陆地区域之间的距离都达到最大，最大距离为 4。
```

提示：

1. `1 <= grid.length == grid[0].length <= 100`
2. `grid[i][j]` 不是 `0` 就是 `1`

2. 比赛实现

从陆地开始BFS找海洋，最后遍历到的就是距离陆地最远的海洋

```
class Solution {
public:
    vector<vector<int>> dirs = {{-1,0},{1,0},{0,-1},{0,1}};//四个方向
    int maxDistance(vector<vector<int>>& grid) {
        int m = grid.size();
        int n = grid[0].size();
        queue<pair<int, int>> q;
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
                if(grid[i][j] == 1)//所有陆地入队列
                    q.push(make_pair(i,j));
        if(q.size() == 0 || q.size() == m*n)//全陆地/全海洋
            return -1;
        int ans = -1;
        while(!q.empty()){
            int size = q.size();
            ans++;
            for(int i = 0; i < size; i++){
                int x = q.front().first;
                int y = q.front().second;
                q.pop();
                for(int j = 0; j < 4; j++){
                    int xx = x + dirs[j][0];
                    int yy = y + dirs[j][1];
                    if(xx >= 0 && xx < m && yy >= 0 && yy < n && grid[xx][yy] == 0)
{
                        grid[xx][yy] = 1;//标记为陆地，表示vsited，省内存
```

```
                        q.push(make_pair(xx, yy));
                    }
                }
            }
        }
        return ans;
    }
};
```

1163. 按字典序排在最后的字串（困难）

1. 题目描述

   给你一个字符串 `s`，找出它的所有子串并按字典序排列，返回排在最后的那个子串。

   示例 1：

   ```
   输入：“abab”
   输出：“bab”
   解释：我们可以找出 7 个子串 [“a”, “ab”, “aba”, “abab”, “b”, “ba”, “bab”]。按字典序排在最后的子串是 “bab”。
   ```

   示例 2：

   ```
   输入：“leetcode”
   输出：“tcode”
   ```

   提示：

   1. `1 <= s.length <= 4 * 10^5`
   2. s 仅含有小写英文字符。

2. 比赛实现

   字典序大的：前缀中的字母足够靠后，前缀相同时越长越好，因此，可以

   - 先找到s中字典序最大的字母head
     - 如果该字母head在s中只出现一次，则直接返回从head开始到s结尾构成的字符串
     - 如果head出现多次，则需要分别以这些head为开头字符，比较它们的下一个字符，并去除字典序小的字符所在的head，直到只剩下一个head，为了字典序最大，还要把其后面剩余的所有字符加上

```
class Solution {
public:
    string lastSubstring(string s) {
        unordered_map<char, vector<int>> idxs;//记录s中各个字符出现的idx
        int len = s.size();
        if(len == 1) return s;
        for(int i = 0; i < len; i++)
            idxs[s[i]].push_back(i);
        if(idxs.size() == 1)//只有一个字符，实际上时看了超时测试用例'aaa....aaaa'之后加上的trick
            return s;
```

```
        char head = 'z';
        while(idxs.find(head) == idxs.end())//找到字典序最大的字符
            head--;
        string ans = "";
        if(idxs[head].size() == 1)
            ans = s.substr(idxs[head][0], len-idxs[head][0]);
        else{
            vector<int> old = idxs[head];//当前遍历到的以各个head为开头的子串的结尾索引
            vector<int> cur;
            while(old.size() > 1){//剩余字串数量大于一
                char cur_max = 'a';
                for(int i = 0; i < old.size(); i++){
                    if(old[i] >= len) continue;
                    if(s[old[i]] == cur_max)//字典序相同
                        cur.push_back(old[i]+1);
                    else if(s[old[i]] > cur_max){//有字典序更大的出现
                        cur_max = s[old[i]];
                        cur.clear();
                        cur.push_back(old[i]+1);
                    }
                }
                ans += cur_max;
                old = cur;
                cur.clear();
            }
            ans += s.substr(old[0], len-old[0]);//把结尾剩余的字符也都加上
        }
        return ans;
    }
};
```

3. 最优解法

从我前面的分析来看，可以得到结论：最终的答案一定是s的一个后缀的，所以直接找后缀

只有满足s[i]>s[i-1]的那些s[i]才有可能作为答案子串的首字母

```
char * lastSubstring(char * s){
    int len = strlen(s);
    int ans = 0;
    for (int i = 1; i < len; i++) {
        if (s[i] <= s[i - 1])
            continue;
        if (strcmp(&s[i], &s[ans]) > 0)
            ans = i;
    }
    return &s[ans];
}
```