301. 删除无效的括号

301. 删除无效的括号

难度 困难 152 收藏 分享 切换为英文 关注 反馈

删除最小数量的无效括号，使得输入的字符串有效，返回所有可能的结果。

说明: 输入可能包含了除 `(` 和 `)` 以外的字符。

示例 1:

```
输入: "()())()"
输出: ["()()()", "(())()"]
```

示例 2:

```
输入: "(a)())()"
输出: ["(a)()()", "(a())()"]
```

示例 3:

```
输入: ")("
输出: [""]
```

303. 区域和检索 - 数组不可变

303. 区域和检索 - 数组不可变

难度 简单 154 收藏 分享 切换为英文 关注 反馈

给定一个整数数组 nums，求出数组从索引 i 到 j (i ≤ j) 范围内元素的总和，包含 i, j 两点。

示例:

```
给定 nums = [-2, 0, 3, -5, 2, -1]，求和函数为 sumRange()

sumRange(0, 2) -> 1
sumRange(2, 5) -> -1
sumRange(0, 5) -> -3
```

说明:

1. 你可以假设数组不可变。
2. 会多次调用 sumRange 方法。

```
class NumArray {
public:
    vector<int> f;
    NumArray(vector<int>& nums) {
        int n = nums.size();
        f = vector<int>(n + 1,0); //!!!!定义未初始化的vector!!!!
```

```
        for(int i = 1;i <= n;i ++)
            f[i] = f[i - 1] + nums[i - 1];
    }

    int sumRange(int i, int j) {
        return f[j + 1] - f[i];
    }
};

/**
 * Your NumArray object will be instantiated and called as such:
 * NumArray* obj = new NumArray(nums);
 * int param_1 = obj->sumRange(i,j);
 */
```

304. 二维区域和检索 - 矩阵不可变

304. 二维区域和检索 - 矩阵不可变

难度 中等 80 收藏 分享 切换为英文 关注 反馈

给定一个二维矩阵，计算其子矩形范围内元素的总和，该子矩阵的左上角为 (row1, col1) ，右下角为 (row2, col2)。

上图子矩阵左上角 (row1, col1) = **(2, 1)** ，右下角(row2, col2) = **(4, 3)**，该子矩形内元素的总和为 8。

示例:

```
给定 matrix = [
  [3, 0, 1, 4, 2],
  [5, 6, 3, 2, 1],
  [1, 2, 0, 1, 5],
  [4, 1, 0, 1, 7],
  [1, 0, 3, 0, 5]
]

sumRegion(2, 1, 4, 3) -> 8
sumRegion(1, 1, 2, 2) -> 11
sumRegion(1, 2, 2, 4) -> 12
```

说明:

1. 你可以假设矩阵不可变。
2. 会多次调用 sumRegion 方法。
3. 你可以假设 row1 ≤ row2 且 col1 ≤ col2。

```
class NumMatrix {
public:
    vector<vector<int>> f;
```

```
    NumMatrix(vector<vector<int>> matrix) {
        if (matrix.empty()) return;
        int n = matrix.size(), m = matrix[0].size();
        f = vector<vector<int>>(n + 1, vector<int>(m + 1, 0));
        for (int i = 1; i <= n; i ++ )
            for (int j = 1; j <= m; j ++ )
                f[i][j] = f[i - 1][j] + matrix[i - 1][j - 1];

        for (int i = 1; i <= n; i ++ )
            for (int j = 1; j <= m; j ++ )
                f[i][j] += f[i][j - 1];
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        return f[row2 + 1][col2 + 1] - f[row2 + 1][col1] - f[row1][col2 + 1] +
f[row1][col1];
    }
};

/**
 * Your NumMatrix object will be instantiated and called as such:
 * NumMatrix obj = new NumMatrix(matrix);
 * int param_1 = obj.sumRegion(row1,col1,row2,col2);
 */
```

306. 累加数

306. 累加数

难度 中等 64 收藏 分享 切换为英文 关注 反馈

累加数是一个字符串，组成它的数字可以形成累加序列。

一个有效的累加序列必须**至少**包含 3 个数。除了最开始的两个数以外，字符串中的其他数都等于它之前两个数相加的和。

给定一个只包含数字 `'0'-'9'` 的字符串，编写一个算法来判断给定输入是否是累加数。

说明: 累加序列里的数不会以 0 开头，所以不会出现 `1, 2, 03` 或者 `1, 02, 3` 的情况。

示例 1:

```
输入: "112358"
输出: true
解释: 累加序列为: 1, 1, 2, 3, 5, 8 。1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8
```

示例 2:

```
输入: "199100199"
输出: true
解释: 累加序列为: 1, 99, 100, 199。1 + 99 = 100, 99 + 100 = 199
```

进阶:
你如何处理一个溢出的过大的整数输入?

307. 区域和检索 - 数组可修改

307. 区域和检索 - 数组可修改

难度 中等 126 收藏 分享 切换为英文 关注 反馈

给定一个整数数组 nums，求出数组从索引 i 到 j (i ≤ j) 范围内元素的总和，包含 i, j 两点。

update(i, val) 函数可以通过将下标为 i 的数值更新为 val，从而对数列进行修改。

示例:

```
Given nums = [1, 3, 5]

sumRange(0, 2) -> 9
update(1, 2)
sumRange(0, 2) -> 8
```

说明:

1. 数组仅可以在 update 函数下进行修改。
2. 你可以假设 update 函数与 sumRange 函数的调用次数是均匀分布的。

```
class NumArray {
public:
    vector<int> a, f, sum;
    int n;
    NumArray(vector<int> nums) {
        n = nums.size();
        a = nums;
        f = vector<int>(n + 1, 0);
        sum = vector<int>(n + 1, 0);
        for (int i = 1; i <= n; i++)
            sum[i] = sum[i - 1] + nums[i - 1];
    }

    void update(int i, int val) {
        int x = i;
        for (x++; x <= n; x += x & -x)
            f[x] += val - a[i];
        a[i] = val;
    }

    int prefixSum(int x) {
        int res = 0;
        for (; x; x -= x & -x)
            res += f[x];
        return res;
    }

    int sumRange(int i, int j) {
        return prefixSum(j + 1) - prefixSum(i) + sum[j + 1] - sum[i];
    }
};

/**
 * Your NumArray object will be instantiated and called as such:
 * NumArray obj = new NumArray(nums);
 * obj.update(i,val);
 * int param_2 = obj.sumRange(i,j);
```

```
 */
```

309. 最佳买卖股票时机含冷冻期

309. 最佳买卖股票时机含冷冻期

难度 中等 302 收藏 分享 切换为英文 关注 反馈

给定一个整数数组，其中第 i 个元素代表了第 i 天的股票价格 。

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）：

- 你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。
- 卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

示例:

```
输入: [1,2,3,0,2]
输出: 3
解释: 对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出]
```

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        if (n == 0)
            return 0;

        vector<int> f(n); // 当天不持有
        vector<int> g(n); // 当天持有

        f[0] = 0;
        g[0] = -prices[0];
        for (int i = 1; i < n; i++) {
            f[i] = max(f[i - 1], g[i - 1] + prices[i]);
            if (i >= 2)
                g[i] = max(g[i - 1], f[i - 2] - prices[i]);
            else
                g[i] = max(g[i - 1], -prices[i]);
        }
        return f[n - 1];
    }
};
```

310. 最小高度树

310. 最小高度树

难度 中等 130 收藏 分享 切换为英文 关注 反馈

对于一个具有树特征的无向图，我们可选择任何一个节点作为根。图因此可以成为树，在所有可能的树中，具有最小高度的树被称为最小高度树。给出这样的一个图，写出一个函数找到所有的最小高度树并返回他们的根节点。

格式

该图包含 `n` 个节点，标记为 `0` 到 `n - 1`。给定数字 `n` 和一个无向边 `edges` 列表（每一个边都是一对标签）。

你可以假设没有重复的边会出现在 `edges` 中。由于所有的边都是无向边，`[0, 1]` 和 `[1, 0]` 是相同的，因此不会同时出现在 `edges` 里。

示例 1:

```
输入: n = 4, edges = [[1, 0], [1, 2], [1, 3]]

        0
        |
        1
       / \
      2   3

输出: [1]
```

313. 超级丑数

313. 超级丑数

难度 中等 74 收藏 分享 切换为英文 关注 反馈

编写一段程序来查找第 `n` 个超级丑数。

超级丑数是指其所有质因数都是长度为 `k` 的质数列表 `primes` 中的正整数。

示例:

```
输入: n = 12, primes = [2,7,13,19]
输出: 32
解释: 给定长度为 4 的质数列表 primes = [2,7,13,19]，前 12 个超级丑数序列为:
[1,2,4,7,8,13,14,16,19,26,28,32] 。
```

说明:

- `1` 是任何给定 `primes` 的超级丑数。
- 给定 `primes` 中的数字以升序排列。
- $0 < k \le 100, 0 < n \le 10^6, 0 < primes[i] < 1000$。
- 第 `n` 个超级丑数确保在 32 位有符整数范围内。

```
class Solution {
public:
    int nthSuperUglyNumber(int n, vector<int>& primes) {
        int idx = 1;
        vector<int> res;
        vector<int> index;
        index.resize(primes.size());
```

```
        res.push_back(1);

        while(idx<n){
            int minval = INT_MAX;
            int minindex = -1;
            for(int i = 0;i<primes.size();++i){
                if (primes[i]*res[index[i]] < minval){
                    minval = res[index[i]] * primes[i];
                    minindex = i;
                }
            }
            index[minindex]++;
            if (minval == res.back()) continue;
            res.push_back(minval);
            idx = idx + 1;
        }
        return res.back();
    }
};
```

315. 计算右侧小于当前元素的个数

315. 计算右侧小于当前元素的个数

难度 困难 218 收藏 分享 切换为英文 关注 反馈

给定一个整数数组 nums，按要求返回一个新数组 counts。数组 counts 有该性质： `counts[i]` 的值是 `nums[i]` 右侧小于 `nums[i]` 的元素的数量。

示例:

```
输入：[5,2,6,1]
输出：[2,1,1,0]
解释：
5 的右侧有 2 个更小的元素 (2 和 1).
2 的右侧仅有 1 个更小的元素 (1).
6 的右侧有 1 个更小的元素 (1).
1 的右侧有 0 个更小的元素.
```

316. 去除重复字母

316. 去除重复字母

难度 困难 143 收藏 分享 切换为英文 关注 反馈

给你一个仅包含小写字母的字符串，请你去除字符串中重复的字母，使得每个字母只出现一次。需保证返回结果的字典序最小（要求不能打乱其他字符的相对位置）。

示例 1:

```
输入：“bcabc”
输出：“abc”
```

示例 2:

```
输入：“cbacdcbc”
输出：“acdb”
```

```
class Solution {
public:
    string removeDuplicateLetters(string s) {
        vector<bool> visit(26,false);
        vector<int> cnt;
        cnt.resize(26);

        for (auto c: s){
            cnt[c-'a']++;
        }
        string result = "";
        for(auto c:s){
            cnt[c-'a']--;
            if (visit[c-'a']) continue;
            while(!result.empty()&&result.back()>c&&cnt[result.back()-'a']>0){
                visit[result.back()-'a'] = false;
                result.pop_back();

            }
            result+=c;
            visit[c-'a'] =true;
        }

        return result;

    }
};
```

318. 最大单词长度乘积

318. 最大单词长度乘积

难度 中等 85 收藏 分享 切换为英文 关注 反馈

给定一个字符串数组 `words`，找到 `length(word[i]) * length(word[j])` 的最大值，并且这两个单词不含有公共字母。你可以认为每个单词只包含小写字母。如果不存在这样的两个单词，返回 0。

示例 1:

```
输入: ["abcw","baz","foo","bar","xtfn","abcdef"]
输出: 16
解释: 这两个单词为 "abcw", "xtfn"。
```

示例 2:

```
输入: ["a","ab","abc","d","cd","bcd","abcd"]
输出: 4
解释: 这两个单词为 "ab", "cd"。
```

示例 3:

```
输入: ["a","aa","aaa","aaaa"]
输出: 0
解释: 不存在这样的两个单词。
```

```cpp
class Solution {
public:
    int maxProduct(vector<string>& words) {
        vector<pair<int, int>> states;
        for (auto word : words)
        {
            unordered_set<char> S;
            for (auto c : word) S.insert(c);
            int state = 0;
            for (int i = 0; i < 26; i ++ )
                if (S.count(i + 'a'))
                    state += 1 << i;
            states.push_back(make_pair(state, word.size()));
        }
        int res = 0;
        for (int i = 0; i < states.size(); i ++ )
            for (int j = 0; j < i; j ++ )
            {
                if (states[i].first & states[j].first) continue;
                res = max(res, states[i].second * states[j].second);
            }
        return res;
    }
};
```

319. 灯泡开关

319. 灯泡开关

难度 中等 98 收藏 分享 切换为英文 关注 反馈

初始时有 n 个灯泡关闭。 第 1 轮，你打开所有的灯泡。 第 2 轮，每两个灯泡你关闭一次。 第 3 轮，每三个灯泡切换一次开关（如果关闭则开启，如果开启则关闭）。第 i 轮，每 i 个灯泡切换一次开关。 对于第 n 轮，你只切换最后一个灯泡的开关。 找出 n 轮后有多少个亮着的灯泡。

示例:

```
输入：3
输出：1
解释：
初始时，灯泡状态 [关闭，关闭，关闭].
第一轮后，灯泡状态 [开启，开启，开启].
第二轮后，灯泡状态 [开启，关闭，开启].
第三轮后，灯泡状态 [开启，关闭，关闭].

你应该返回 1，因为只有一个灯泡还亮着。
```

```cpp
class Solution {
public:
    int bulbSwitch(int n) {
        return sqrt(n);
    }
};
```