

241. 为运算表达式设计优先级

241. 为运算表达式设计优先级

难度 中等  156  收藏  分享  切换为英文  关注  反馈

给定一个含有数字和运算符的字符串，为表达式添加括号，改变其运算优先级以求出不同的结果。你需要给出所有可能的组合的结果。有效的运算符包含 `+`、`-` 以及 `*`。

示例 1:

输入: "2-1-1"

输出: [0, 2]

解释:

$((2-1)-1) = 0$

$(2-(1-1)) = 2$

示例 2:

输入: "2*3-4*5"

输出: [-34, -14, -10, -10, 10]

解释:

$(2*(3-(4*5))) = -34$

$((2*3)-(4*5)) = -14$

$((2*(3-4))*5) = -10$

$(2*((3-4)*5)) = -10$

$((((2*3)-4)*5)) = 10$

```
class Solution {
public:
    vector<int> solve(int l, int r, const vector<int>& input) {
        if (l > r)
            return vector<int>{};
        if (l == r)
            return vector<int>{input[l]};
        vector<int> sum;
        for (int i = l + 1; i < r; i++) {
            vector<int> left = solve(l, i - 1, input);
            vector<int> right = solve(i + 1, r, input);
            for (auto x : left)
                for (auto y : right) {
                    if (input[i] == 0)
                        sum.push_back(x + y);
                    else if (input[i] == 1)
                        sum.push_back(x - y);
                    else
                        sum.push_back(x * y);
                }
        }
        return sum;
    }

    vector<int> diffwaysToCompute(string input) {
        vector<int> cleaned_input;
```

```

int num = 0, n = 0;
for (auto c : input) {
    if (c >= '0' && c <= '9')
        num = num * 10 + c - '0';
    else {
        cleaned_input.push_back(num);
        num = 0;
        if (c == '+')
            cleaned_input.push_back(0);
        else if (c == '-')
            cleaned_input.push_back(1);
        else
            cleaned_input.push_back(2);
        n += 2;
    }
}

cleaned_input.push_back(num);
n++;

return solve(0, n - 1, cleaned_input);
}
};

```

242. 有效的字母异位词

242. 有效的字母异位词

难度 简单

185

收藏

分享

切换为英文

关注

反馈

给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。

示例 1:

输入: $s = \text{"anagram"}, t = \text{"nagaram"}$
 输出: true

示例 2:

输入: $s = \text{"rat"}, t = \text{"car"}$
 输出: false

说明:

你可以假设字符串只包含小写字母。

进阶:

如果输入字符串包含 unicode 字符怎么办？你能否调整你的解法来应对这种情况？

```

class Solution {
public:
    bool isAnagram(string s, string t) {
        vector<int> hash(26, 0);
        for (auto c : s)
            hash[c - 'a']++;
        for (auto c : t)
            hash[c - 'a']--;
    }
};

```

```

        for (int i = 0; i < 26; i++)
            if (hash[i] != 0)
                return false;
        return true;
    }
};

```

257. 二叉树的所有路径

257. 二叉树的所有路径

难度 **简单** 248 收藏 分享 切换为英文 关注 反馈

给定一个二叉树，返回所有从根节点到叶子节点的路径。

说明: 叶子节点是指没有子节点的节点。

示例:

输入:

```

      1
     / \
    2   3
     \
      5

```

输出: ["1->2->5", "1->3"]

解释: 所有根节点到叶子节点的路径为: 1->2->5, 1->3

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    void dfs(TreeNode *cur, string path, vector<string>& ans) {
        if (cur -> left == NULL && cur -> right == NULL) {
            ans.push_back(path);
            return;
        }
        if (cur -> left != NULL)
            dfs(cur -> left, path + "->" + to_string(cur -> left -> val), ans);







        if (cur -> right != NULL)
            dfs(cur -> right, path + "->" + to_string(cur -> right -> val),
ans);
    }
    vector<string> binaryTreePaths(TreeNode* root) {
        if (root == NULL)

```

```
        return vector<string>{};
        string path = to_string(root -> val);
        vector<string> ans;
        dfs(root, path, ans);
        return ans;
    }
};
```

258. 各位相加

258. 各位相加

难度 **简单**  238  收藏  分享  切换为英文  关注  反馈

给定一个非负整数 `num`，反复将各个位上的数字相加，直到结果为一位数。

示例:

输入: 38

输出: 2

解释: 各位相加的过程为: $3 + 8 = 11$, $1 + 1 = 2$ 。由于 2 是一位数，所以返回 2。

进阶:

你可以不使用循环或者递归，且在 $O(1)$ 时间复杂度内解决这个问题吗？

```
class Solution {
public:
    int addDigits(int num) {
        while (num >= 10) {
            int tot = 0;
            for (; num > 0; num /= 10)
                tot += num % 10;
            num = tot;
        }
        return num;
    }
};
```

260. 只出现一次的数字 III

260. 只出现一次的数字 III

难度 中等

👍 210

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个整数数组 `nums`，其中恰好有两个元素只出现一次，其余所有元素均出现两次。找出只出现一次的那两个元素。

示例：

输入：[1,2,1,3,2,5]

输出：[3,5]

注意：

1. 结果输出的顺序并不重要，对于上面的例子，[5, 3] 也是正确答案。
2. 你的算法应该具有线性时间复杂度。你能否仅使用常数空间复杂度来实现？

```
class Solution {
public:
    vector<int> singleNumber(vector<int>& nums) {
        int sum = 0;
        for (int i = 0; i < nums.size(); i++)
            sum ^= nums[i];
        int pos = 0;
        for (int i = 0; i < 32; i++)
            if ((sum >> i) & 1) {
                pos = i;
                break;
            }
        int s1 = 0, s2 = 0;
        for (int i = 0; i < nums.size(); i++)
            if ((nums[i] >> pos) & 1)
                s1 ^= nums[i];
            else
                s2 ^= nums[i];

        return vector<int>{s1, s2};
    }
};
```

263. 丑数

263. 丑数

难度 简单

120

收藏

分享

切换为英文

关注

反馈

编写一个程序判断给定的数是否为丑数。

丑数就是只包含质因数 2, 3, 5 的**正整数**。

示例 1:

输入: 6
输出: true
解释: $6 = 2 \times 3$

示例 2:

输入: 8
输出: true
解释: $8 = 2 \times 2 \times 2$

示例 3:

输入: 14
输出: false
解释: 14 不是丑数，因为它包含了另外一个质因数 7。

说明:

- 1 是丑数。
- 输入不会超过 32 位有符号整数的范围: $[-2^{31}, 2^{31} - 1]$ 。

```
class Solution {
public:
    bool isUgly(int num) {
        if (num <= 0)
            return false;
        while (num % 5 == 0)
            num = num / 5;
        while (num % 3 == 0)
            num = num / 3;
        while (num % 2 == 0)
            num = num / 2;
        return num == 1;
    }
};
```

264. 丑数 II

264. 丑数 II

难度 中等

264

收藏

分享

切换为英文

关注

反馈

编写一个程序，找出第 n 个丑数。

丑数就是只包含质因数 2, 3, 5 的**正整数**。

示例:

输入: $n = 10$

输出: 12

解释: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 是前 10 个丑数。

说明:

- 1 是丑数。
- n 不超过 1690。

```
class Solution {
public:
    long long nthUglyNumber(int n) {
        if(n <= 0)
            return 0;
        priority_queue<long long> pq;
        pq.push(-1);
        //由于priority queue会把最大的值放在顶端，为了实现最小堆，我把数字取负数放进去，最后
        //取结果时再变为正数即可。
        long long lastnum = 0; //标记前一个数字
        for(int i = 1; i <= n; i++){
            long long top = pq.top(); //取最小堆堆顶
            pq.pop();
            if(top == lastnum){
                //注意到不能重复，如果第i个丑数和第i-1个丑数重复了，那么跳过该丑数
                i--;
                continue;
            }
            lastnum = top;
            pq.push(top*2); //将堆顶乘上2、3、5并放入堆中
            pq.push(top*3);
            pq.push(top*5);
        }
        return lastnum*-1; //注意将结果取相反数
    }
};
/*
class Solution {
public:
    int nthUglyNumber(int n) {
        vector<int> uglyNumbers;
        uglyNumbers.push_back(1);
        int index2 = 0; //2 3 5三个指针
        int index3 = 0;
        int index5 = 0;
        for(int i = 1; i < n; i++){
            int curMaxNum2 = uglyNumbers[index2]*2; //找出2 3 5指针指向的当前最大丑数
            int curMaxNum3 = uglyNumbers[index3]*3;
            int curMaxNum5 = uglyNumbers[index5]*5;
```

```

        int uglynum = min(min(curMaxNum2,curMaxNum3),curMaxNum5); //从当前最大
        丑数中选一个最小的
        if(uglynum==curMaxNum2)//更新指针
            index2++;
        if(uglynum==curMaxNum3)
            index3++;
        if(uglynum==curMaxNum5)
            index5++;
        uglyNumbers.push_back(uglynum);
    }
    return uglyNumbers[n-1];
}
};
*/

```

268. 缺失数字

268. 缺失数字

难度 **简单**  249  收藏  分享  切换为英文  关注  反馈

给定一个包含 $0, 1, 2, \dots, n$ 中 n 个数的序列，找出 $0 \dots n$ 中没有出现在序列中的那个数。

示例 1:

输入: $[3, 0, 1]$
输出: 2

示例 2:

输入: $[9, 6, 4, 2, 3, 5, 7, 0, 1]$
输出: 8

说明:

你的算法应具有线性时间复杂度。你能否仅使用额外常数空间来实现？

```

class Solution {
public:
    int missingNumber(vector<int>& nums) {
        long long sum = 0;
        int n = nums.size();
        for (auto x : nums)
            sum += x;
        return (int)((long long)(n) * (n + 1) / 2 - sum);
    }
};

```

273. 整数转换英文表示

273. 整数转换英文表示

难度 **困难** 66 收藏 分享 切换为英文 关注 反馈

将非负整数转换为其对应的英文表示。可以保证给定输入小于 $2^{31} - 1$ 。

示例 1:

输入: 123
输出: "One Hundred Twenty Three"

示例 2:

输入: 12345
输出: "Twelve Thousand Three Hundred Forty Five"

示例 3:

输入: 1234567
输出: "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"

示例 4:

输入: 1234567891
输出: "One Billion Two Hundred Thirty Four Million Five Hundred Sixty Seven Thousand Eight Hundred Ninety One"

274. H 指数

274. H 指数

难度 **中等** 66 收藏 分享 切换为英文 关注 反馈

给定一位研究者论文被引用次数的数组（被引用次数是非负整数）。编写一个方法，计算出研究者的 h 指数。

h 指数的定义: h 代表“高引用次数”（high citations），一名科研人员的 h 指数是指他（她）的（ N 篇论文中）**至多**有 h 篇论文分别被引用了**至少** h 次。（其余的 $N - h$ 篇论文每篇被引用次数 **不超过** h 次。）

例如：某人的 h 指数是 20，这表示他已发表的论文中，每篇被引用了至少 20 次的论文总共有 20 篇。

示例:

输入: citations = [3,0,6,1,5]
输出: 3
解释: 给定数组表示研究者总共有 5 篇论文，每篇论文相应的被引用了 3, 0, 6, 1, 5 次。
由于研究者有 3 篇论文每篇 **至少** 被引用了 3 次，其余两篇论文每篇被引用 **不多于** 3 次，所以她的 h 指数是 3。

```
class Solution {  
public:
```

```

int hIndex(vector<int>& citations) {
    sort(citations.begin(), citations.end());
    int res = 0;
    for (int i = 0; i < citations.size(); i++)
        if (citations.size() - i <= citations[i])
        {
            res = citations.size() - i;
            break;
        }
    return res;
}
};

```

275. H指数 II

275. H指数 II

难度 **中等** 42 收藏 分享 切换为英文 关注 反馈

给定一位研究者论文被引用次数的数组（被引用次数是非负整数），数组已经按照**升序排列**。编写一个方法，计算出研究者的 h 指数。

h 指数的定义：“ h 代表“高引用次数”（high citations），一名科研人员的 h 指数是指他（她）的（ N 篇论文中）**至多有** h 篇论文分别被引用了**至少** h 次。（其余的 $N - h$ 篇论文每篇被引用次数**不多于** h 次。）”

示例:

输入: citations = [0,1,3,5,6]

输出: 3

解释: 给定数组表示研究者总共有 5 篇论文，每篇论文相应的被引用了 0, 1, 3, 5, 6 次。

由于研究者有 3 篇论文每篇至少被引用了 3 次，其余两篇论文每篇被引用不多于 3 次，所以她的 h 指数是 3。

说明:

如果 h 有多有几种可能的值， h 指数是其中最大的那个。

进阶:

- 这是 H 指数的延伸题目，本题中的 citations 数组是保证有序的。
- 你可以优化你的算法到对数时间复杂度吗？

```

class Solution {
public:
    int hIndex(vector<int>& citations) {
        if(citations.empty()) return 0;
        int l = 0, r = citations.size();
        while(l < r)
        {
            int mid = l + r + 1 >> 1;

```

```

        if(citations[citations.size() - mid] >= mid) l = mid;
        else r = mid - 1;
    }
    return r;
}
};

```

278. 第一个错误的版本

278. 第一个错误的版本

难度 **简单**  159  收藏  分享  切换为英文  关注  反馈

你是产品经理，目前正在带领一个团队开发新的产品。不幸的是，你的产品的最新版本没有通过质量检测。由于每个版本都是基于之前的版本开发的，所以错误的版本之后的所有版本都是错的。

假设你有 n 个版本 $[1, 2, \dots, n]$ ，你想找出导致之后所有版本出错的第一个错误的版本。

你可以通过调用 `bool isBadVersion(version)` 接口来判断版本号 `version` 是否在单元测试中出错。实现一个函数来查找第一个错误的版本。你应该尽量减少对调用 API 的次数。

示例:

给定 $n = 5$ ，并且 `version = 4` 是第一个错误的版本。

调用 `isBadVersion(3)` -> false

调用 `isBadVersion(5)` -> true

调用 `isBadVersion(4)` -> true

所以，4 是第一个错误的版本。

```

// Forward declaration of isBadVersion API.
bool isBadVersion(int version);

```

```

class Solution {
public:
    int firstBadVersion(int n) {
        long long l = 0, r = n;
        while(l < r)
        {
            int mid = l + r + 011 >> 1;
            if(isBadVersion(mid)) r = mid;
            else l = mid + 1;
        }
        return r;
    }
};

```

279. 完全平方数

279. 完全平方数

难度 中等

404

收藏

分享

切换为英文

关注

反馈

给定正整数 n ，找到若干个完全平方数（比如 1, 4, 9, 16, ...）使得它们的和等于 n 。你需要让组成和的完全平方数的个数最少。

示例 1:

输入: $n = 12$

输出: 3

解释: $12 = 4 + 4 + 4$.

示例 2:

输入: $n = 13$

输出: 2

解释: $13 = 4 + 9$.

```
class Solution {
public:
    int numSquares(int n) {
        vector<int> f(n + 1, n);
        f[0] = 0;
        for (int i = 1; i <= n; i++)
            for (int j = 1; j * j <= i; j++)
                f[i] = min(f[i], f[i - j * j] + 1);

        return f[n];
    }
};
/*
class Solution {
public:
    int numSquares(int n) {
        vector<int> f(n + 1, n);
        queue<int> q;
        f[0] = 0;
        q.push(0);
        while (!q.empty()) {
            int s = q.front();
            if (s == n)
                break;
            q.pop();
            for (int i = 1; s + i * i <= n; i++)
                if (f[s + i * i] > f[s] + 1) {
                    f[s + i * i] = f[s] + 1;
                    q.push(s + i * i);
                }
        }
        return f[n];
    }
};
*/
```