

目录

301. Remove Invalid Parentheses	4
302. Smallest Rectangle Enclosing Black Pixels	5
303. Range Sum Query - Immutable	6
304. Range Sum Query 2D - Immutable	8
305. Number of Islands II	10
306. Additive Number	12
307. Range Sum Query - Mutable	14
308. Range Sum Query 2D - Mutable	15
309. Best Time to Buy and Sell Stock with Cooldown	16
310. Minimum Height Trees(●~√~●).....	17
311. Sparse Matrix Multiplication	19
312. Burst Balloons	21
313. Super Ugly Number	23
314. Binary Tree Vertical Order Traversal	26
315. Count of Smaller Numbers After Self(●~√~●).....	30
316. Remove Duplicate Letters.....	32
317. Shortest Distance from All Buildings.....	34
318. Maximum Product of Word Lengths(●~√~●)	35
319. Bulb Switcher	38
320. Generalized Abbreviation	40
321. Create Maximum Number.....	41
322. Coin Change(●~√~●)	42
323. Number of Connected Components in an Undirected Graph.....	44
324. Wiggle Sort II	46
325. Maximum Size Subarray Sum Equals k.....	47
326. Power of Three	49
327. Count of Range Sum.....	50
328. Odd Even Linked List.....	51
329. Longest Increasing Path in a Matrix.....	53
330. Patching Array(●~√~●).....	56
331. Verify Preorder Serialization of a Binary Tree	58
332. Reconstruct Itinerary(●~√~●).....	60
333. Largest BST Subtree.....	62
334. Increasing Triplet Subsequence.....	64
335. Self Crossing.....	66
336. Palindrome Pairs.....	68
337. House Robber III(●~√~●).....	70
338. Counting Bits(●~√~●)	72
339. Nested List Weight Sum.....	74
340. Longest Substring with At Most K Distinct Characters	77
341. Flatten Nested List Iterator.....	79
342. Power of Four	82

343. Integer Break.....	83
344. Reverse String.....	84
345. Reverse Vowels of a String.....	85
346. Moving Average from Data Stream.....	86
347. Top K Frequent Elements(●~V~●).....	87
348. Design Tic-Tac-Toe.....	89
349. Intersection of Two Arrays.....	92
350. Intersection of Two Arrays II.....	95
352. Data Stream as Disjoint Intervals.....	98
354. Russian Doll Envelopes.....	100
355. Design Twitter.....	101
357. Count Numbers with Unique Digits.....	104
363. Max Sum of Rectangle No Larger Than K.....	105
365. Water and Jug Problem.....	107
367. Valid Perfect Square.....	108
368. Largest Divisible Subset.....	110
371. Sum of Two Integers.....	112
372. Super Pow.....	113
373. Find K Pairs with Smallest Sums.....	115
374. Guess Number Higher or Lower.....	117
375. Guess Number Higher or Lower II.....	119
376. Wiggle Subsequence.....	121
377. Combination Sum IV.....	123
378. Kth Smallest Element in a Sorted Matrix.....	125
380. Insert Delete GetRandom O(1).....	127
381. Insert Delete GetRandom O(1) - Duplicates allowed.....	129
382. Linked List Random Node.....	131
383. Ransom Note.....	133
384. Shuffle an Array.....	134
385. Mini Parser.....	136
386. Lexicographical Numbers.....	138
387. First Unique Character in a String.....	139
388. Longest Absolute File Path.....	140
389. Find the Difference.....	142
390. Elimination Game.....	143
391. Perfect Rectangle.....	145
392. Is Subsequence.....	147
393. UTF-8 Validation.....	149
394. Decode String.....	151
395. Longest Substring with At Least K Repeating Characters.....	153
396. Rotate Function.....	155
397. Integer Replacement.....	157
398. Random Pick Index.....	159
399. Evaluate Division.....	161

400. Nth Digit	164
----------------------	-----

301. Remove Invalid Parentheses

Hard

Remove the minimum number of invalid parentheses in order to make the input string valid.
Return all possible results.

Note: The input string may contain letters other than the parentheses (and).

Example 1:

Input: "()())"

Output: ["()()", "(())"]

Example 2:

Input: "(a)())"

Output: ["(a)()", "(a())"]

Example 3:

Input: ")("

Output: [""]

302. Smallest Rectangle Enclosing Black Pixels

An image is represented by a binary matrix with 0 as a white pixel and 1 as a black pixel. The black pixels are connected, i.e., there is only one black region. Pixels are connected horizontally and vertically. Given the location (x, y) of one of the black pixels, return the area of the smallest (axis-aligned) rectangle that encloses all black pixels.

Example:

Input:

```
[  
  "0010",  
  "0110",  
  "0100"  
]
```

and $x = 0, y = 2$

Output: 6

303. Range Sum Query - Immutable

Easy

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* ($i \leq j$), inclusive.

Example:

```
Given nums = [-2, 0, 3, -5, 2, -1]
```

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

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

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

Note:

1. You may assume that the array does not change.
2. There are many calls to *sumRange* function.

```
class NumArray {  
public:  
    NumArray(vector<int> nums) {  
  
    }  
};
```

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

    int sumRange(int i, int j) {
        return sum[j+1]-sum[i];
    }
private:
    vector<int> sum;
};
```

304. Range Sum Query 2D - Immutable

Medium

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row1*, *col1*) and lower right corner (*row2*, *col2*).

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

The above rectangle (with the red border) is defined by (*row1*, *col1*) = (2, 1) and (*row2*, *col2*) = (4, 3), which contains sum = 8.

Example:

Given 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

Note:

1. You may assume that the matrix does not change.
2. There are many calls to *sumRegion* function.
3. You may assume that $row1 \leq row2$ and $col1 \leq col2$.


```

class NumMatrix {
public:
    NumMatrix(vector<vector<int>> matrix) {
        if (matrix.empty()) return;
        int n = matrix.size(), m = matrix[0].size();
        sum.resize(n, vector<int>(m, 0));
        for (int i = 0; i < n; i++) {
            int rowSum = 0;
            for (int j = 0; j < m; j++) {
                rowSum += matrix[i][j];
                if (i == 0) sum[i][j] = rowSum;
                else sum[i][j] = sum[i-1][j]+rowSum;
            }
        }
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        if (sum.empty()) return 0;
        else return get_sum(row2, col2)+get_sum(row1-1, col1-1)
            -get_sum(row2, col1-1)-get_sum(row1-1, col2);
    }
private:
    vector<vector<int>> sum;
    int get_sum(int i, int j) {
        if (i < 0 || j < 0) return 0;
        else return sum[i][j];
    }
};

```

305. Number of Islands II

A 2d grid map of m rows and n columns is initially filled with water. We may perform an *addLand* operation which turns the water at position (row, col) into a land. Given a list of positions to operate, **count the number of islands after each *addLand* operation**. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example:

Input: $m = 3, n = 3, \text{positions} = [[0,0], [0,1], [1,2], [2,1]]$

Output: $[1,1,2,3]$

Explanation:

Initially, the 2d grid `grid` is filled with water. (Assume 0 represents water and 1 represents land).

```
0 0 0
0 0 0
0 0 0
```

Operation #1: `addLand(0, 0)` turns the water at `grid[0][0]` into a land.

```
1 0 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #2: `addLand(0, 1)` turns the water at `grid[0][1]` into a land.

```
1 1 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #3: `addLand(1, 2)` turns the water at `grid[1][2]` into a land.

```
1 1 0
0 0 1   Number of islands = 2
0 0 0
```

Operation #4: `addLand(2, 1)` turns the water at `grid[2][1]` into a land.

```
1 1 0
```

```
0 0 1   Number of islands = 3
0 1 0
```

Follow up:

Can you do it in time complexity $O(k \log mn)$, where k is the length of the `positions`?

```
class Solution {
public:
    vector<int> numIslands2(int m, int n, vector<vector<int>>& positions) {
        vector<vector<int>> dirs{{0, -1}, {-1, 0}, {0, 1}, {1, 0}};
        vector<int> fa(m*n, -1), res;
        // -1 water, other land 标签
        int cnt = 0;

        for (auto &v : positions) {
            int id = n*v[0] + v[1];
            if (fa[id] == -1) {
                fa[id] = id;
                ++cnt;
            }
            for (auto dir : dirs) {
                int x = v[0]+dir[0], y = v[1]+dir[1], cur_id = n*x + y;
                if (x < 0 || x >= m || y < 0 || y >= n || fa[cur_id] == -1)
                    continue;
                int p = find(fa, cur_id), q = find(fa, id);
                if (p != q) {
                    fa[p] = q;
                    --cnt;
                }
            }
            res.push_back(cnt);
        }
        return res;
    }

private:
    int find(vector<int> &fa, int x) {
        return (x == fa[x]) ? x : (fa[x] = find(fa, fa[x]));
    }
};
```

306. Additive Number

Medium

Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain **at least** three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

Given a string containing only digits `'0' - '9'`, write a function to determine if it's an additive number.

Note: Numbers in the additive sequence **cannot** have leading zeros, so sequence `1, 2, 03` or `1, 02, 3` is invalid.

Example 1:

Input: "112358"

Output: true

Explanation: The digits can form an additive sequence: 1, 1, 2, 3, 5, 8.

$$1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8$$

Example 2:

Input: "199100199"

Output: true

Explanation: The additive sequence is: 1, 99, 100, 199.

$$1 + 99 = 100, 99 + 100 = 199$$

Follow up:

How would you handle overflow for very large input integers?

```
class Solution {
public:
    bool isAdditiveNumber(string num) {

    }
};
```

```

class Solution {
public:
    bool isAdditiveNumber(string num) {
        int n = num.length();
        for (int i = 1; i <= n/2; i++) {
            string x = num.substr(0, i);
            for (int j = i+1; j < n; j++) {
                if (num[i] == '0' && j-i != 1) continue;
                string y = num.substr(i, j-i);
                if (dfs(x, y, j, num, 2)) return true;
            }
        }
        return false;
    }

private:
    bool dfs(string x, string y, int start, string &num, int cnt) {
        if (start == num.length() && cnt > 2) return true;
        string s = add(x, y);
        if (s != num.substr(start, s.length())) return false;
        return dfs(y, s, start+s.length(), num, cnt+1);
    }

    string add(string x, string y) {
        reverse(x.begin(), x.end());
        reverse(y.begin(), y.end());
        string ret;
        int c = 0, p = 0, q = 0, n = x.length(), m = y.length();
        while (p < n || q < m) {
            c += (p < n ? x[p++] - '0' : 0) + (q < m ? y[q++] - '0' : 0);
            ret += c % 10 + '0';
            c /= 10;
        }
        if (c) ret += '1';
        reverse(ret.begin(), ret.end());
        return ret;
    }
};

```

307. Range Sum Query - Mutable

Medium

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* ($i \leq j$), inclusive.

The *update(i, val)* function modifies *nums* by updating the element at index *i* to *val*.

Example:

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

```
sumRange(0, 2) -> 9
```

```
update(1, 2)
```

```
sumRange(0, 2) -> 8
```

Note:

1. The array is only modifiable by the *update* function.
2. You may assume the number of calls to *update* and *sumRange* function is distributed evenly.

308. Range Sum Query 2D - Mutable

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner *(row1, col1)* and lower right corner *(row2, col2)*.

The above rectangle (with the red border) is defined by *(row1, col1) = (2, 1)* and *(row2, col2) = (4, 3)*, which contains sum = 8.

Example:

```
Given 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  
  
update(3, 2, 2)  
  
sumRegion(2, 1, 4, 3) -> 10
```

Note:

1. The matrix is only modifiable by the *update* function.
2. You may assume the number of calls to *update* and *sumRegion* function is distributed evenly.
3. You may assume that $row1 \leq row2$ and $col1 \leq col2$.

309. Best Time to Buy and Sell Stock with Cooldown

Medium

Say you have an array for which the i^{th} element is the price of a given stock on day i .

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

- You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).
- After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

Example:

Input: [1,2,3,0,2]

Output: 3

Explanation: transactions = [buy, sell, cooldown, buy, sell]

310. Minimum Height Trees(●~▽~●)

Medium

For an undirected graph with tree characteristics, we can choose any node as the root. The result graph is then a rooted tree. Among all possible rooted trees, those with minimum height are called minimum height trees (MHTs). Given such a graph, write a function to find all the MHTs and return a list of their root labels.

Format

The graph contains `n` nodes which are labeled from `0` to `n - 1`. You will be given the number `n` and a list of undirected `edges` (each edge is a pair of labels).

You can assume that no duplicate edges will appear in `edges`. Since all edges are undirected, `[0, 1]` is the same as `[1, 0]` and thus will not appear together in `edges`.

Example 1 :

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

```

    0
    |
    1
  /\
 2  3
```

Output: `[1]`

Example 2 :

Input: `n = 6, edges = [[0, 3], [1, 3], [2, 3], [4, 3], [5, 4]]`

```

0  1  2
 \  |  /
  3
  |
  4
  |
  5
```

Output: `[3, 4]`

```

class Solution {
public:
    vector<int> findMinHeightTrees(int n, vector<pair<int, int>>& edges) {
        vector<unordered_set<int>> adj(n);
        for (auto &p : edges) {
            adj[p.first].insert(p.second);
            adj[p.second].insert(p.first);
        }
        vector<int> current;
        if (n == 1) {
            current.push_back(0);
            return current;
        }
        // Create first leaf layer
        for (int i = 0; i < adj.size(); ++i) {
            if (adj[i].size() == 1) {
                current.push_back(i);
            }
        }
        // BFS the graph
        while (1) {
            vector<int> next;
            for (int node : current) {
                for (int neighbor : adj[node]) {
                    adj[neighbor].erase(node);
                    if (adj[neighbor].size() == 1) next.push_back(neighbor);
                }
            }
            if (next.empty()) return current;
            current = next;
        }
    }
};

```

311. Sparse Matrix Multiplication

Given two [sparse matrices](#) **A** and **B**, return the result of **AB**.

You may assume that **A**'s column number is equal to **B**'s row number.

Example:

Input:

```
A = [  
  [ 1, 0, 0],  
  [-1, 0, 3]  
]
```

```
B = [  
  [ 7, 0, 0 ],  
  [ 0, 0, 0 ],  
  [ 0, 0, 1 ]  
]
```

Output:

```
      | 1 0 0 |   | 7 0 0 |   | 7 0 0 |  
AB = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |  
      | 0 0 1 |
```

```

class Solution {
public:
    vector<vector<int>> multiply(vector<vector<int>> &A, vector<vector<int>>
&B) {
        int n = A.size(), l = B.size(), m = B[0].size();
        vector<vector<int>> res(n, vector<int> (m, 0));
        vector<pair<int, int>> v[l];
        for (int i = 0; i < l; i++) {
            for (int j = 0; j < m; j++) if (B[i][j]) {
                v[i].push_back({j, B[i][j]});
            }
        }
        for (int i = 0; i < n; ++i) {
            for (int k = 0; k < l; ++k) if (A[i][k]) {
                for (auto &pii : v[k]) {
                    res[i][pii.first] += A[i][k] * pii.second;
                }
            }
        }
        return res;
    }
};

```

312. Burst Balloons

Hard

Given n balloons, indexed from 0 to $n-1$. Each balloon is painted with a number on it represented by array `nums`. You are asked to burst all the balloons. If the you burst balloon i you will get `nums[left] * nums[i] * nums[right]` coins. Here `left` and `right` are adjacent indices of i . After the burst, the `left` and `right` then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

Note:

- You may imagine `nums[-1] = nums[n] = 1`. They are not real therefore you can not burst them.
- $0 \leq n \leq 500, 0 \leq \text{nums}[i] \leq 100$

Example:

Input: [3,1,5,8]

Output: 167

Explanation: `nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []`

$$\text{coins} = 3*1*5 + 3*5*8 + 1*3*8 + 1*8*1 = 167$$

```

class Solution {
public:
    int maxCoins(vector<int>& nums) {
        int N = nums.size();
        nums.insert(nums.begin(), 1);
        nums.insert(nums.end(), 1);
        vector<vector<int>> dp(N+2, vector<int>(N+2, 0));
        for (int len = 1; len <= N; ++len) {
            for (int i = 1; i <= N - len + 1; ++i) {
                int k = i + len - 1;
                int bestCoins = 0;
                for (int j = i; j <= k; ++j) {
                    int coins = dp[i][j-1] + dp[j+1][k];
                    coins += nums[i-1] * nums[j] * nums[k+1];
                    bestCoins = max(bestCoins, coins);
                }
                dp[i][k] = bestCoins;
            }
        }
        return dp[1][N];
    }
};

```

313. Super Ugly Number

Medium

Write a program to find the n^{th} super ugly number.

Super ugly numbers are positive numbers whose all prime factors are in the given prime list `primes` of size `k`.

Example:

Input: `n = 12, primes = [2,7,13,19]`

Output: 32

Explanation: [1,2,4,7,8,13,14,16,19,26,28,32] is the sequence of the first 12 super ugly numbers given primes = [2,7,13,19] of size 4.

Note:

- 1 is a super ugly number for any given `primes`.
- The given numbers in `primes` are in ascending order.
- $0 < k \leq 100$, $0 < n \leq 10^6$, $0 < \text{primes}[i] < 1000$.
- The n^{th} super ugly number is guaranteed to fit in a 32-bit signed integer.

```
class Solution {
public:
    int nthSuperUglyNumber(int n, vector<int>& primes) {

    }
};
```

```

class Solution {
public:
    int nthSuperUglyNumber(int n, vector<int>& primes) {
        int sz = primes.size();
        vector<int> k(sz, 0), res(1, 1);
        while (--n) {
            int min_v = INT_MAX;
            for (int i = 0; i < sz; i++) {
                min_v = min(min_v, primes[i]*res[k[i]]);
            }
            res.push_back(min_v);
            for (int i = 0; i < sz; i++) {
                if (primes[i]*res[k[i]] == min_v)
                    k[i]++;
            }
        }
        return res.back();
    }
};

```



```

class Solution {
public:
    int nthSuperUglyNumber(int n, vector<int>& primes) {
        priority_queue<int, vector<int>, greater<int>> pq;
        unordered_set<int> My_set;
        pq.push(1);
        while (--n) {
            int t = pq.top();
            pq.pop();
            for (int i : primes) {
                if (INT_MAX / i < t) continue;
                i *= t;
                if (My_set.count(i)) continue;
                My_set.insert(i);
                pq.push(i);
            }
        }
        return pq.top();
    }
};

```

314. Binary Tree Vertical Order Traversal

Given a binary tree, return the *vertical order* traversal of its nodes' values. (ie, from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from **left to right**.

Examples 1:

Input: [3,9,20,null,null,15,7]

```

  3
 / \
9   20
 / \
15  7
```

Output:

```
[
  [9],
  [3,15],
  [20],
  [7]
]
```

Examples 2:

Input: [3,9,8,4,0,1,7]

```

      3
    /\
   /\ 
  9  8
 /\  /\
/\  \/\ 
4  01  7

```

Output:

```

[
  [4],
  [9],
  [3,0,1],
  [8],
  [7]
]

```

Examples 3:

Input: [3,9,8,4,0,1,7,null,null,null,2,5] (0's right child is 2 and 1's left child is 5)

```

      3
    /\
   /\ 
  9  8
 /\  /\
/\  \/\ 
4  01  7

```

```
  /\n /  \n5    2
```

Output:

```
[
  [4],
  [9,5],
  [3,0,1],
  [8,2],
  [7]
]
```

```

/**
 * 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:
    vector<vector<int>> verticalOrder(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        map<int, vector<int>> m;
        queue<pair<TreeNode*, int>> q;
        q.push({root, 0});
        while(!q.empty()) {
            auto t = q.front();
            q.pop();
            m[t.second].push_back(t.first->val);
            if (t.first->left) q.push({t.first->left, t.second-1});
            if (t.first->right) q.push({t.first->right, t.second+1});
        }
        for (auto &i : m) res.push_back(i.second);
        return res;
    }
};

```

315. Count of Smaller Numbers After Self(●~▽~●)

Hard

You are given an integer array *nums* and you have to return a new *counts* array. The *counts* array has the property where `counts[i]` is the number of smaller elements to the right of `nums[i]`.

Example:

Input: [5,2,6,1]

Output: [2,1,1,0]

Explanation:

To the right of 5 there are **2** smaller elements (2 and 1).

To the right of 2 there is only **1** smaller element (1).

To the right of 6 there is **1** smaller element (1).

To the right of 1 there is **0** smaller element.

```

class Solution {
public:
    vector<int> countSmaller(vector<int>& nums) {
        int n = nums.size();
        vector<pair<int, int>> v(n);
        vector<int> res(n, 0);
        for (int i = 0; i < n; i++) v[i] = {nums[i], i};
        merge_sort(v.begin(), v.end(), res);
        return res;
    }

private:
    void merge_sort(vector<pair<int, int>>::iterator l, vector<pair<int,
int>>::iterator r, vector<int> &res) {
        if (l+1 < r) {
            auto mid = l + (r-l)/2;
            merge_sort(l, mid, res);
            merge_sort(mid, r, res);
            merge(l, r, res);
        }
    }

    void merge(vector<pair<int, int>>::iterator l, vector<pair<int, int>>::
iterator r, vector<int> &res) {
        vector<pair<int, int>> t(r-l);
        int cnt = 0;
        auto mid = l + (r-l)/2, p = l, q = mid;
        while (p < mid || q < r) {
            if (p < mid && q < r) {
                if (q->first >= p->first) t[cnt++] = *q++;
                else {
                    res[p->second] += r-q;
                    t[cnt++] = *p++;
                }
            }
            else if (p < mid) t[cnt++] = *p++;
            else t[cnt++] = *q++;
        }
        int k = 0;
        for(auto i = l; i < r; i++) *i = t[k++];
    }
};

```

316. Remove Duplicate Letters

Hard

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appear once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

Example 1:

Input: "bcabc"

Output: "abc"

Example 2:

Input: "cbacdcbc"

Output: "acdb"


```

class Solution {
public:
    string removeDuplicateLetters(string s) {
        vector<int> dict(256, 0);
        vector<bool> visited(256, false);
        for(auto ch : s) dict[ch]++;
        string result = "";
        for(auto c : s) {
            dict[c]--;
            if (visited[c]) continue;
            while (c < result.back() && dict[result.back()] > 0) {
                visited[result.back()] = false;
                result.pop_back();
            }
            result += c;
            visited[c] = true;
        }
        return result.substr(1);
    }
};

```

317. Shortest Distance from All Buildings

You want to build a house on an *empty* land which reaches all buildings in the shortest amount of distance. You can only move up, down, left and right. You are given a 2D grid of values **0**, **1** or **2**, where:

- Each **0** marks an empty land which you can pass by freely.
- Each **1** marks a building which you cannot pass through.
- Each **2** marks an obstacle which you cannot pass through.

Example:

Input: `[[1,0,2,0,1],[0,0,0,0,0],[0,0,1,0,0]]`

```
1 - 0 - 2 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
```

Output: 7

Explanation: Given three buildings at (0,0), (0,4), (2,2), and an obstacle at (0,2),

the point (1,2) is an ideal empty land to build a house, as the total

travel distance of 3+3+1=7 is minimal. So return 7.

Note:

There will be at least one building. If it is not possible to build such house according to the above rules, return -1.

318. Maximum Product of Word Lengths(●~▽~●)

Medium

Given a string array `words`, find the maximum value of `length(word[i]) * length(word[j])` where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

Example 1:

Input: ["abcw","baz","foo","bar","xtfn","abcdef"]

Output: 16

Explanation: The two words can be "abcw", "xtfn".

Example 2:

Input: ["a","ab","abc","d","cd","bcd","abcd"]

Output: 4

Explanation: The two words can be "ab", "cd".

Example 3:

Input: ["a","aa","aaa","aaaa"]

Output: 0

Explanation: No such pair of words.

```
class Solution {
public:
    int maxProduct(vector<string> &words) {

    }
};
```

```
class Solution {
public:
    int maxProduct(vector<string> &words) {
        vector<int> mask(words.size());
        int res = 0;
        for (int i = 0; i < words.size(); ++i) {
            for (char c : words[i])
                mask[i] |= 1 << (c - 'a');
            for (int j = 0; j < i; ++j) if (!(mask[i] & mask[j])) {
                res = max(res, int(words[i].size()*words[j].size()));
            }
        }
        return res;
    }
};
```

```

class Solution {
public:
    int maxProduct(vector<string>& words) {
        unordered_map<int, int> maxlen;
        for (string word : words) {
            int mask = 0;
            for (char c : word) mask |= 1 << (c - 'a');
            maxlen[mask] = max(maxlen[mask], (int)word.size());
        }
        int res = 0;
        for (auto a : maxlen) {
            for (auto b : maxlen)
                if (!(a.first & b.first))
                    res = max(res, a.second * b.second);
        }
        return res;
    }
};

```

319. Bulb Switcher

Medium

There are n bulbs that are initially off. You first turn on all the bulbs. Then, you turn off every second bulb. On the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). For the i -th round, you toggle every i bulb. For the n -th round, you only toggle the last bulb. Find how many bulbs are on after n rounds.

Example:

Input: 3

Output: 1

Explanation:

At first, the three bulbs are [**off**, **off**, **off**].

After first round, the three bulbs are [**on**, **on**, **on**].

After second round, the three bulbs are [**on**, **off**, **on**].

After third round, the three bulbs are [**on**, **off**, **off**].

So you should return 1, because there is only one bulb is on.

```

class Solution {
public:
    int bulbSwitch(int n) {
        // return sqrt(n); /*这是标答， 以下答案超时*/
        bitset<100000000> bit;
        for (int i = 1; i <= n; i++) {
            for (int j = i; j <= n; j += i) {
                bit.flip(j);
            }
        }
        return bit.count();
    }
};
/*

```

对于一个叫做 bit 的 bitset:

```

bit.size()      返回大小（位数）
bit.count()     返回 1 的个数
bit.any()       返回是否有 1
bit.none()      返回是否没有 1
bit.set()       全都变成 1
bit.set(p)      将第 p + 1 位变成 1（bitset 是从第 0 位开始的！）
bit.set(p, x)   将第 p + 1 位变成 x
bit.reset()     全都变成 0
bit.reset(p)    将第 p + 1 位变成 0
bit.flip()      全都取反
bit.flip(p)     将第 p + 1 位取反
bit.to_ulong()  返回它转换为 unsigned long 的结果，如果超出范围则报错
bit.to_ullong() 返回它转换为 unsigned long long 的结果，如果超出范围则报错
bit.to_string() 返回它转换为 string 的结果
*/

```

320. Generalized Abbreviation

Write a function to generate the generalized abbreviations of a word.

Note: The order of the output does not matter.

Example:

Input: "word"

Output:

["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1", "1o2", "2r1", "3d", "w3", "4"]

```
class Solution {
public:
    vector<string> generateAbbreviations(string word) {
        if (word.empty()) return {" "};
        vector<string> pre{" "};
        for(char &c : word) {
            vector<string> cur;
            for (auto &s : pre) {
                cur.push_back(s+c);
                if (isdigit(s.back())) {
                    int i = s.size()-1;
                    while(i >= 1 && isdigit(s[i-1])) i--;
                    cur.push_back(s.substr(0, i) +
                                   to_string(stoi(s.substr(i))+1));
                }
                else cur.push_back(s+"1");
            }
            pre = cur;
        }
        return pre;
    }
};
```


321. Create Maximum Number

Hard

Given two arrays of length m and n with digits $0-9$ representing two numbers. Create the maximum number of length $k \leq m + n$ from digits of the two. The relative order of the digits from the same array must be preserved. Return an array of the k digits.

Note: You should try to optimize your time and space complexity.

Example 1:

Input:

```
nums1 = [3, 4, 6, 5]
```

```
nums2 = [9, 1, 2, 5, 8, 3]
```

```
k = 5
```

Output:

```
[9, 8, 6, 5, 3]
```

Example 2:

Input:

```
nums1 = [6, 7]
```

```
nums2 = [6, 0, 4]
```

```
k = 5
```

Output:

```
[6, 7, 6, 0, 4]
```

Example 3:

Input:

```
nums1 = [3, 9]
```

```
nums2 = [8, 9]
```

```
k = 3
```

Output:

```
[9, 8, 9]
```

322. Coin Change(●~▽~●)

Medium

You are given coins of different denominations and a total amount of money *amount*. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return `-1`.

Example 1:

Input: coins = [1, 2, 5], amount = 11

Output: 3

Explanation: 11 = 5 + 5 + 1

Example 2:

Input: coins = [2], amount = 3

Output: -1

Note:

You may assume that you have an infinite number of each kind of coin.

```
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {

    }
};
```

```

class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        int Max = amount + 1;
        vector<int> dp(amount + 1, Max);
        dp[0] = 0;
        for (int i = 1; i <= amount; i++) {
            for (int j = 0; j < coins.size(); j++) {
                if (coins[j] <= i) {
                    dp[i] = min(dp[i], dp[i-coins[j]]+1);
                }
            }
        }
        return dp[amount] > amount ? -1 : dp[amount];
    }
};

```

```

class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        int Max = amount + 1;
        vector<int> dp(amount+1, Max);
        dp[0] = 0;
        for(int i = 0; i < coins.size(); i++) {
            for(int j = coins[i]; j <= amount; j++)
                dp[j] = min(dp[j], dp[j-coins[i]] + 1);
        }
        return dp[amount] == Max ? -1 : dp[amount];
    }
};

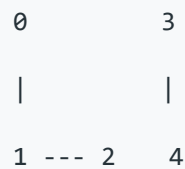
```

323. Number of Connected Components in an Undirected Graph

Given n nodes labeled from 0 to $n - 1$ and a list of undirected edges (each edge is a pair of nodes), write a function to find the number of connected components in an undirected graph.

Example 1:

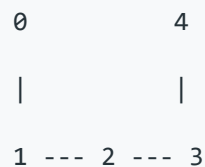
Input: $n = 5$ and $edges = [[0, 1], [1, 2], [3, 4]]$



Output: 2

Example 2:

Input: $n = 5$ and $edges = [[0, 1], [1, 2], [2, 3], [3, 4]]$



Output: 1

Note:

You can assume that no duplicate edges will appear in `edges`. Since all edges are undirected, `[0, 1]` is the same as `[1, 0]` and thus will not appear together in `edges`.

```

class Solution {
public:
    int countComponents(int n, vector<vector<int>>& edges) {
        vector<vector<int>> g(n);
        vector<bool> v(n, false);
        for (auto &i : edges) {
            g[i[0]].push_back(i[1]);
            g[i[1]].push_back(i[0]);
        }
        for (int i = 0; i < n; i++) if (!v[i]) {
            dfs(i, g, v);
            res++;
        }
        return res;
    }
private:
    int res = 0;
    void dfs(int i, vector<vector<int>> &g, vector<bool> &v) {
        v[i] = true;
        for (auto &j : g[i]) if (!v[j]) {
            dfs(j, g, v);
        }
    }
};

```

324. Wiggle Sort II

Medium

Given an unsorted array `nums`, reorder it such that `nums[0] < nums[1] > nums[2] < nums[3] ...`.

Example 1:

Input: `nums = [1, 5, 1, 1, 6, 4]`

Output: One possible answer is `[1, 4, 1, 5, 1, 6]`.

Example 2:

Input: `nums = [1, 3, 2, 2, 3, 1]`

Output: One possible answer is `[2, 3, 1, 3, 1, 2]`.

Note:

You may assume all input has valid answer.

Follow Up:

Can you do it in $O(n)$ time and/or in-place with $O(1)$ extra space?

325. Maximum Size Subarray Sum Equals k

Given an array *nums* and a target value *k*, find the maximum length of a subarray that sums to *k*. If there isn't one, return 0 instead.

Note:

The sum of the entire *nums* array is guaranteed to fit within the 32-bit signed integer range.

Example 1:

Input: *nums* = [1, -1, 5, -2, 3], *k* = 3

Output: 4

Explanation: The subarray [1, -1, 5, -2] sums to 3 and is the longest.

Example 2:

Input: *nums* = [-2, -1, 2, 1], *k* = 1

Output: 2

Explanation: The subarray [-1, 2] sums to 1 and is the longest.

Follow Up:

Can you do it in $O(n)$ time?

```
class Solution {
public:
    int maxSubArrayLen(vector<int> &nums, int k) {

    }
};
```

```
class Solution {
public:
    int maxSubArrayLen(vector<int> &nums, int k) {
        int sum = 0, res = 0, n = nums.size();
        unordered_map<int, int> m;
        for (int i = 0; i < n; ++i) {
            sum += nums[i];
            if (sum == k) res = i + 1;
            else if (m.count(sum - k)) res = max(res, i - m[sum - k]);
            if (!m.count(sum)) m[sum] = i;
        }
        return res;
    }
};
```


326. Power of Three

Easy

Given an integer, write a function to determine if it is a power of three.

Example 1:

Input: 27

Output: true

Example 2:

Input: 0

Output: false

Example 3:

Input: 9

Output: true

Example 4:

Input: 45

Output: false

Follow up:

Could you do it without using any loop / recursion?

```
class Solution {
public:
    bool isPowerOfThree(int n) {
        if(n <= 0) return false;
        return pow(3, (int)(log10(n) / log10(3)+0.5)) == n;
        // return n > 0 && (1162261467 % n == 0);
    }
};
```

327. Count of Range Sum

Hard

Given an integer array `nums`, return the number of range sums that lie in `[lower, upper]` inclusive.

Range sum $S(i, j)$ is defined as the sum of the elements in `nums` between indices `i` and `j` ($i \leq j$), inclusive.

Note:

A naive algorithm of $O(n^2)$ is trivial. You MUST do better than that.

Example:

Input: `nums = [-2,5,-1]`, `lower = -2`, `upper = 2`,

Output: 3

Explanation: The three ranges are : `[0,0]`, `[2,2]`, `[0,2]` and their respective sums are: -2, -1, 2.

328. Odd Even Linked List

Medium

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in $O(1)$ space complexity and $O(\text{nodes})$ time complexity.

Example 1:

Input: 1->2->3->4->5->NULL

Output: 1->3->5->2->4->NULL

Example 2:

Input: 2->1->3->5->6->4->7->NULL

Output: 2->3->6->7->1->5->4->NULL

Note:

- The relative order inside both the even and odd groups should remain as it was in the input.
- The first node is considered odd, the second node even and so on ...

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {

    }
};
```

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        ListNode *even = new ListNode(-1), *odd = new ListNode(-1);
        ListNode *q = even, *p = odd, *t = head;
        while (t) {
            p->next = t;
            p = t;
            t = t->next;
            q->next = t;
            q = t;
            if(t) t = t->next;
        }
        p->next = even->next;
        return odd->next;
    }
};

```

329. Longest Increasing Path in a Matrix

Hard

85417FavoriteShare

Given an integer matrix, find the length of the longest increasing path.

From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

Example 1:

Input: nums =

```
[
  [9,9,4],
  [6,6,8],
  [2,1,1]
]
```

Output: 4

Explanation: The longest increasing path is [1, 2, 6, 9].

Example 2:

Input: nums =

```
[
  [3,4,5],
  [3,2,6],
  [2,2,1]
]
```

Output: 4

Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.

```

class Solution {
public:
    int longestIncreasingPath(vector<vector<int>>& mat) {
        if (mat.empty()) return 0;
        int n = mat.size(), m = mat[0].size();

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (i != 0 && mat[i][j] > mat[i-1][j])
                    mp[i*m+j].push_back((i-1)*m+j);
                else if (i != 0 && mat[i][j] < mat[i-1][j])
                    mp[(i-1)*m+j].push_back(i*m+j);
                if (j != 0 && mat[i][j] > mat[i][j-1])
                    mp[i*m+j].push_back(i*m+j-1);
                else if (j != 0 && mat[i][j] < mat[i][j-1])
                    mp[i*m+j-1].push_back(i*m+j);
            }
        }
        int res = 0;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                res = max(res, dfs(i*m+j));
            }
        }
        return res;
    }
private:
    unordered_map<int, vector<int>> mp;
    unordered_map<int, int> dp;

    int dfs(int i) {
        if (dp.count(i)) return dp[i];
        int ret = 1;
        for (auto j : mp[i]) {
            ret = max(ret, 1 + dfs(j));
        }
        return dp[i] = ret;
    }
};

```

```

class Solution {
public:
    int longestIncreasingPath(vector<vector<int>>& matrix) {
        if (matrix.empty()) return 0;
        int n = matrix.size(), m = matrix[0].size();

        vector<vector<int>> dp(n, vector<int>(m, 0));
        std::function<int(int, int)> dfs = [&] (int x, int y) {
            if (dp[x][y]) return dp[x][y];
            vector<vector<int>> dirs = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}};
            for (auto &dir : dirs) {
                int xx = x + dir[0], yy = y + dir[1];
                if (xx < 0 || xx >= n || yy < 0 || yy >= m) continue;
                if (matrix[xx][yy] <= matrix[x][y]) continue;
                dp[x][y] = std::max(dp[x][y], dfs(xx, yy));
            }
            return ++dp[x][y];
        };

        int ret = 0;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                ret = std::max(ret, dfs(i, j));
            }
        }

        return ret;
    }
};

```

330. Patching Array(●~▽~●)

Hard

Given a sorted positive integer array *nums* and an integer *n*, add/patch elements to the array such that any number in range $[1, n]$ inclusive can be formed by the sum of some elements in the array. Return the minimum number of patches required.

Example 1:

Input: *nums* = [1,3], *n* = 6

Output: 1

Explanation:

Combinations of *nums* are [1], [3], [1,3], which form possible sums of: 1, 3, 4.

Now if we add/patch 2 to *nums*, the combinations are: [1], [2], [3], [1,3], [2,3], [1,2,3].

Possible sums are 1, 2, 3, 4, 5, 6, which now covers the range [1, 6].

So we only need 1 patch.

Example 2:

Input: *nums* = [1,5,10], *n* = 20

Output: 2

Explanation: The two patches can be [2, 4].

Example 3:

Input: *nums* = [1,2,2], *n* = 5

Output: 0

```
class Solution {
public:
    int minPatches(vector<int>& nums, int n) {

    }
};
```

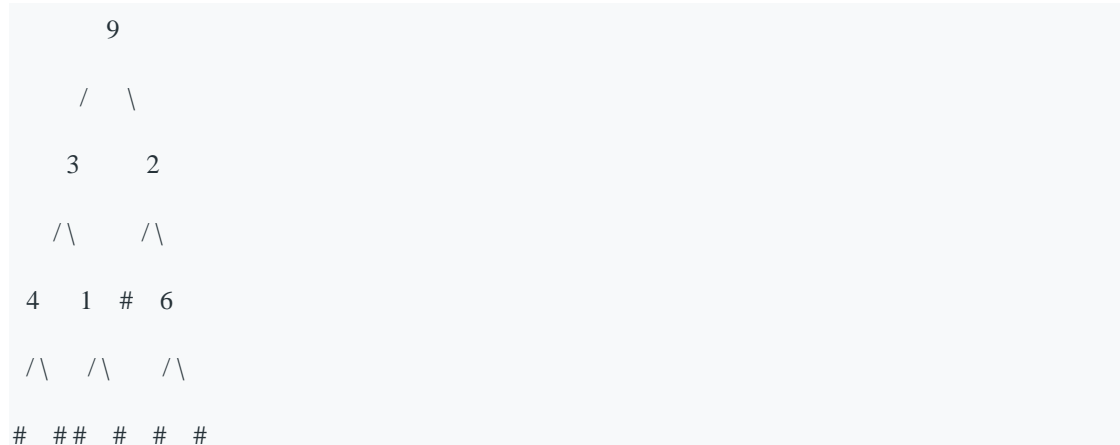


```
class Solution {
public:
    int minPatches(vector<int>& nums, int n) {
        int cnt = 0, i = 0;
        long long maxNum = 1;
        while (maxNum <= n) {
            if (i < nums.size() && nums[i] <= maxNum)
                maxNum += nums[i++];
            else {
                maxNum *= 2;
                cnt++;
            }
        }
        return cnt;
    }
};
```

331. Verify Preorder Serialization of a Binary Tree

Medium

One way to serialize a binary tree is to use pre-order traversal. When we encounter a non-null node, we record the node's value. If it is a null node, we record using a sentinel value such as #.



For example, the above binary tree can be serialized to the string "9,3,4,#,#,1,#,#,2,#,6,#,#", where # represents a null node.

Given a string of comma separated values, verify whether it is a correct preorder traversal serialization of a binary tree. Find an algorithm without reconstructing the tree.

Each comma separated value in the string must be either an integer or a character '#' representing `null` pointer.

You may assume that the input format is always valid, for example it could never contain two consecutive commas such as "1,,3".

Example 1:

Input: "9,3,4,#,#,1,#,#,2,#,6,#,#"

Output: true

Example 2:

Input: "1,#"

Output: false

Example 3:

Input: "9,#,#,1"

Output: false

```

class Solution {
public:
    bool isValidSerialization(string preorder) {
        stack<bool> stk;
        stringstream ss(preorder+',');
        char c;
        while (ss >> c) {
            bool Isdigit = isdigit(c);
            while(c != ',') ss >> c;
            if (!insert(stk, Isdigit)) return false;
        }
        return stk.size() == 1 && stk.top() == false;
    }

private:
    bool insert(stack<bool> &stk, bool Isdigit) {
        if (Isdigit) stk.push(true);
        else if (!stk.empty() && stk.top() == false) {
            stk.pop();
            if (stk.empty() || stk.top() != true) return false;
            stk.pop();
            return insert(stk, Isdigit);
        }
        else stk.push(false);
        return true;
    }
};

```

332. Reconstruct Itinerary(●~▽~●)

Medium

Given a list of airline tickets represented by pairs of departure and arrival airports `[from, to]`, reconstruct the itinerary in order. All of the tickets belong to a man who departs from `JFK`. Thus, the itinerary must begin with `JFK`.

Note:

1. If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order when read as a single string. For example, the itinerary `["JFK", "LGA"]` has a smaller lexical order than `["JFK", "LGB"]`.
2. All airports are represented by three capital letters (IATA code).
3. You may assume all tickets form at least one valid itinerary.

Example 1:

Input: `[["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]`

Output: `["JFK", "MUC", "LHR", "SFO", "SJC"]`

Example 2:

Input: `[["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]`

Output: `["JFK","ATL","JFK","SFO","ATL","SFO"]`

Explanation: Another possible reconstruction is `["JFK","SFO","ATL","JFK","ATL","SFO"]`.

But it is larger in lexical order.

```

class Solution {
public:
    // fleury's algorithm or Hierholzer algorithm
    vector<string> findItinerary(vector<vector<string>>> &tickets) {
        for (auto e : tickets)
            graph[e[0]].push(e[1]);
        dfs("JFK");
        reverse(res.begin(), res.end());
        return res;
    }

private:
    unordered_map<string, priority_queue<string, vector<string>,
greater<string>>>> graph;
    vector<string> res;

    void dfs(string from) {
        auto &edges = graph[from];
        while (!edges.empty()) {
            string to = edges.top();
            edges.pop();
            dfs(to);
        }
        res.push_back(from);
    }
};

```

333. Largest BST Subtree

Given a binary tree, find the largest subtree which is a Binary Search Tree (BST), where largest means subtree with largest number of nodes in it.

Note:

A subtree must include all of its descendants.

Example:

Input: [10,5,15,1,8,null,7]



Output: 3

Explanation: The Largest BST Subtree in this case is the highlighted one.

The return value is the subtree's size, which is 3.

Follow up:

Can you figure out ways to solve it with $O(n)$ time complexity?

```

/**
 * 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:
    int largestBSTSubtree(TreeNode* root) {
        int sum = 0, Min = INT_MAX, Max = INT_MIN;
        dfs(root, Min, Max, sum);
        return res;
    }

private:
    int res = 0;
    bool dfs(TreeNode *root, int &Min, int &Max, int &sum) {
        if (root == nullptr) {
            Min = INT_MAX, Max = INT_MIN, sum = 0;
            return true;
        }
        int l_sum = 0, r_sum = 0;
        int l_min = INT_MAX, r_min = INT_MAX, l_max = INT_MIN, r_max =
INT_MIN;
        bool l_tree = dfs(root->left, l_min, l_max, l_sum);
        bool r_tree = dfs(root->right, r_min, r_max, r_sum);
        Min = min(Min, min(root->val, l_min));
        Max = max(Max, max(root->val, r_max));
        if (l_tree && r_tree && l_max < root->val && r_min > root->val) {
            res = max(res, sum = l_sum + r_sum+1);
            return true;
        }
        return false;
    }
};

```

334. Increasing Triplet Subsequence

Medium

Given an unsorted array return whether an increasing subsequence of length 3 exists or not in the array.

Formally the function should:

Return true if there exists i, j, k

such that $arr[i] < arr[j] < arr[k]$ given $0 \leq i < j < k \leq n-1$ else return false.

Note: Your algorithm should run in $O(n)$ time complexity and $O(1)$ space complexity.

Example 1:

Input: [1,2,3,4,5]

Output: true

Example 2:

Input: [5,4,3,2,1]

Output: false

```
class Solution {
public:
    bool increasingTriplet(vector<int>& nums) {

    }
};
```



```
class Solution {
public:
    bool increasingTriplet(vector<int>& nums) {
        int i = INT_MAX, j = INT_MAX;
        for (auto &a : nums) {
            if (a <= i) i = a;
            else if (a <= j) j = a;
            else return true;
        }
        return false;
    }
};
```

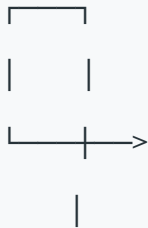
335. Self Crossing

Hard

You are given an array x of n positive numbers. You start at point $(0, 0)$ and moves $x[0]$ metres to the north, then $x[1]$ metres to the west, $x[2]$ metres to the south, $x[3]$ metres to the east and so on. In other words, after each move your direction changes counter-clockwise.

Write a one-pass algorithm with $O(1)$ extra space to determine, if your path crosses itself, or not.

Example 1:



Input: [2,1,1,2]

Output: true

Example 2:



Input: [1,2,3,4]

Output: false

Example 3:



Input: [1,1,1,1]

Output: true

```
class Solution {
public:
    bool isSelfCrossing(vector<int>& x) {
        x.insert(x.begin(), 4, 0);
        int i = 4, n = x.size();

        for (; i < n && x[i] > x[i - 2]; i++); //逐渐变大

        if (i == n) return false;

        if (x[i] >= x[i-2] - x[i-4]) x[i-1] -= x[i-3];

        for (i++; i < n && x[i] < x[i-2]; i++); //逐渐变小

        return i != n;
    }
};
```

336. Palindrome Pairs

Hard

Given a list of **unique** words, find all pairs of *distinct* indices (i, j) in the given list, so that the concatenation of the two words, i.e. `words[i] + words[j]` is a palindrome.

Example 1:

Input: ["abcd","dcba","lls","s","ssll"]

Output: [[0,1],[1,0],[3,2],[2,4]]

Explanation: The palindromes are ["dcbaabcd","abcdcba","slls","llssll"]

Example 2:

Input: ["bat","tab","cat"]

Output: [[0,1],[1,0]]

Explanation: The palindromes are ["battab","tabbat"]

```

class Solution {
public:
    vector<vector<int>> palindromePairs(vector<string>& words) {
        vector<vector<int>> res;
        unordered_map<string, int> dict;
        int i, j, n = words.size();
        for(i = 0; i < n; i++) {
            string t = words[i];
            reverse(t.begin(), t.end());
            dict[t] = i;
        }

        for(i = 0; i < n; i++) {
            for(j = 0; j < words[i].size(); j++) {
                string left = words[i].substr(0, j);
                string right = words[i].substr(j);
                if(dict.count(left) && dict[left] != i && isPalindrome(right)){
                    res.push_back({i, dict[left]});
                    if (left.empty()) res.push_back({dict[left], i});
                }
                if (dict.count(right) && dict[right] != i
                    && isPalindrome(left)) {
                    res.push_back({dict[right], i});
                }
            }
        }
        return res;
    }

private:
    bool isPalindrome(string s) {
        int start, end, n = s.size();
        for(start = 0, end = n - 1; start < end; start++, end--) {
            if(s[start] != s[end])
                return false;
        }
        return true;
    }
};

```

337. House Robber III(●~▽~●)

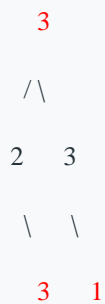
Medium

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

Example 1:

Input: [3,2,3,null,3,null,1]



Output: 7

Explanation: Maximum amount of money the thief can rob = 3 + 3 + 1 = 7.

Example 2:

Input: [3,4,5,1,3,null,1]



Output: 9

Explanation: Maximum amount of money the thief can rob = 4 + 5 = 9.

```
class Solution {
public:
    int rob(TreeNode* root) {
        int l, r;
        return tryRob(root, l, r);
    }

private:
    int tryRob(TreeNode *root, int &l, int &r) {
        // l 为左子树最大值（不考虑父亲节点偷不偷）
        if (!root) return 0;
        int ll = 0, lr = 0, rl = 0, rr = 0;
        l = tryRob(root->left, ll, lr);
        r = tryRob(root->right, rl, rr);
        return max(root->val + ll + lr + rl + rr, l + r);
    }
};
```

338. Counting Bits(●~▽~●)

Medium

Given a non negative integer number **num**. For every numbers **i** in the range $0 \leq i \leq \text{num}$ calculate the number of 1's in their binary representation and return them as an array.

Example 1:

Input: 2

Output: [0,1,1]

Example 2:

Input: 5

Output: [0,1,1,2,1,2]

Follow up:

- It is very easy to come up with a solution with run time $O(n * \text{sizeof}(\text{integer}))$. But can you do it in linear time $O(n)$ /possibly in a single pass?
- Space complexity should be $O(n)$.
- Can you do it like a boss? Do it without using any builtin function like `__builtin_popcount` in c++ or in any other language.

```
class Solution {
public:
    vector<int> countBits(int num) {

    }
};
```



```
class Solution {
public:
    vector<int> countBits(int num) {
        vector<int> res(num + 1, 0);
        for (int i = 1; i <= num; ++i) {
            res[i] = res[i >> 1] + (i & 1);
        }
        return res;
    }
};
```

339. Nested List Weight Sum

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

Input: `[[1,1],2,[1,1]]`

Output: 10

Explanation: Four 1's at depth 2, one 2 at depth 1.

Example 2:

Input: `[1,[4,[6]]]`

Output: 27

Explanation: One 1 at depth 1, one 4 at depth 2, and one 6 at depth 3; $1 + 4*2 + 6*3 = 27$.

```

/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * class NestedInteger {
 *     public:
 *         // Constructor initializes an empty nested list.
 *         NestedInteger();
 *
 *         // Constructor initializes a single integer.
 *         NestedInteger(int value);
 *
 *         // Return true if this NestedInteger holds a single integer, rather
than a nested list.
 *         bool isInteger() const;
 *
 *         // Return the single integer that this NestedInteger holds, if it
holds a single integer
 *         // The result is undefined if this NestedInteger holds a nested list
 *         int getInteger() const;
 *
 *         // Set this NestedInteger to hold a single integer.
 *         void setInteger(int value);
 *
 *         // Set this NestedInteger to hold a nested list and adds a nested
integer to it.
 *         void add(const NestedInteger &ni);
 *
 *         // Return the nested list that this NestedInteger holds, if it holds a
nested list
 *         // The result is undefined if this NestedInteger holds a single
integer
 *         const vector<NestedInteger> &getList() const;
 * };
 */

class Solution {
public:
    int depthSum(vector<NestedInteger>& nestedList) {

    }
};

```

```
class Solution {
public:
    int depthSum(vector<NestedInteger>& nestedList) {
        return dfs(1, nestedList);
    }

private:
    int dfs(int depth, vector<NestedInteger>& nestedList) {
        int ret = 0;
        for (auto &nest : nestedList) {
            if (nest.isInteger()) ret += depth*nest.getInteger();
            else ret += dfs(depth+1, nest.getList());
        }
        return ret;
    }
};
```

340. Longest Substring with At Most K Distinct Characters

Given a string, find the length of the longest substring T that contains at most k distinct characters.

Example 1:

Input: s = "eceba", k = 2

Output: 3

Explanation: T is "ece" which its length is 3.

Example 2:

Input: s = "aa", k = 1

Output: 2

Explanation: T is "aa" which its length is 2.

```
class Solution {
public:
    int lengthOfLongestSubstringKDistinct(string s, int k) {

    }
};
```

```

class Solution {
public:
    int lengthOfLongestSubstringKDistinct(string s, int k) {
        int res = 0, left = 0, n = s.size();
        unordered_map<char, int> m;
        for (int i = 0; i < n; ++i) {
            ++m[s[i]];
            while (m.size() > k) {
                if (--m[s[left]] == 0) m.erase(s[left]);
                ++left;
            }
            res = max(res, i - left + 1);
        }
        return res;
    }
};

```

341. Flatten Nested List Iterator

Medium

Given a nested list of integers, implement an iterator to flatten it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

Input: [[1,1],2,[1,1]]

Output: [1,1,2,1,1]

Explanation: By calling *next* repeatedly until *hasNext* returns false,
the order of elements returned by *next* should be: [1,1,2,1,1].

Example 2:

Input: [1,[4,[6]]]

Output: [1,4,6]

Explanation: By calling *next* repeatedly until *hasNext* returns false,
the order of elements returned by *next* should be: [1,4,6].

```
/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * class NestedInteger {
 *   public:
 *     // Return true if this NestedInteger holds a single integer, rather
 *     // than a nested list.
 *     bool isInteger() const;
 *
 *     // Return the single integer that this NestedInteger holds, if it
 *     // holds a single integer
 *     // The result is undefined if this NestedInteger holds a nested list
 *     int getInteger() const;
 *
 *     // Return the nested list that this NestedInteger holds, if it holds a
 *     // nested list
 *     // The result is undefined if this NestedInteger holds a single
 *     // integer
 *     const vector<NestedInteger> &getList() const;
 * };
 */
```

```

class NestedIterator {
public:
    NestedIterator(vector<NestedInteger> &nestedList) {
        begins.push(nestedList.begin());
        ends.push(nestedList.end());
    }

    int next() {
        hasNext();
        return (begins.top()++)->getInteger();
    }

    bool hasNext() {
        while (!begins.empty()) {
            if (begins.top() == ends.top()) {
                begins.pop();
                ends.pop();
            } else {
                auto x = begins.top();
                if (x->isInteger()) return true;
                begins.top()++;
                begins.push(x->getList().begin());
                ends.push(x->getList().end());
            }
        }
        return false;
    }

private:
    stack<vector<NestedInteger>::iterator> begins, ends;
};

```



```

class NestedIterator {
public:
    NestedIterator(vector<NestedInteger> &nestedList) {
        int n = nestedList.size();
        for(int i = n-1; i >= 0; --i) {
            stk.push(nestedList[i]);
        }
    }

    int next() {
        int result = stk.top().getInteger();
        stk.pop();
        return result;
    }

    bool hasNext() {
        while (!stk.empty()) {
            auto cur = stk.top();
            if (cur.isInteger()) return true;
            stk.pop();
            auto &adjs = cur.getList();
            int n = adjs.size();
            for(int i = n-1; i >= 0; --i) stk.push(adjs[i]);
        }
        return false;
    }
private:
    stack<NestedInteger> stk;
};

```

342. Power of Four

Easy

Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

Example 1:

Input: 16

Output: true

Example 2:

Input: 5

Output: false

Follow up: Could you solve it without loops/recursion?

```
class Solution {
public:
    bool isPowerOfFour(int num) {
        return num > 0 && (num & (num - 1)) == 0 && (num - 1) % 3 == 0;
        //return (num > 0) && ((num & (num - 1)) == 0) && ((num & 0x55555555)
        == num);
    }
};
```

343. Integer Break

Medium

Given a positive integer n , break it into the sum of **at least** two positive integers and maximize the product of those integers. Return the maximum product you can get.

Example 1:

Input: 2

Output: 1

Explanation: $2 = 1 + 1$, $1 \times 1 = 1$.

Example 2:

Input: 10

Output: 36

Explanation: $10 = 3 + 3 + 4$, $3 \times 3 \times 4 = 36$.

Note: You may assume that n is not less than 2 and not larger than 58.

344. Reverse String

Easy

Write a function that reverses a string. The input string is given as an array of characters `char[]`.

Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.

You may assume all the characters consist of printable ascii characters.

Example 1:

Input: ["h","e","l","l","o"]

Output: ["o","l","l","e","h"]

Example 2:

Input: ["H","a","n","n","a","h"]

Output: ["h","a","n","n","a","H"]

```
class Solution {
public:
    void reverseString(vector<char>& s) {
        int i = 0, j = s.size()-1;
        while (i < j) {
            swap(s[i++], s[j--]);
        }
    }
};
```

345. Reverse Vowels of a String

Easy

Write a function that takes a string as input and reverse only the vowels of a string.

Example 1:

Input: "hello"

Output: "holle"

Example 2:

Input: "leetcode"

Output: "leotcede"

Note:

The vowels does not include the letter "y".

```
class Solution {
public:
    string reverseVowels(string s) {
        int i = 0, j = s.size()-1;
        while (i < j) {
            i = s.find_first_of("aeiouAEIOU", i);
            j = s.find_last_of("aeiouAEIOU", j);
            if (i < j) swap(s[i++], s[j--]);
        }
        return s;
    }
};
```

346. Moving Average from Data Stream

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

Example:

```
MovingAverage m = new MovingAverage(3);

m.next(1) = 1

m.next(10) = (1 + 10) / 2

m.next(3) = (1 + 10 + 3) / 3

m.next(5) = (10 + 3 + 5) / 3
```

```
class MovingAverage {
public:
    /** Initialize your data structure here. */
    MovingAverage(int size) {
        this->size = size;
    }

    double next(int val) {
        q.push(val);
        if (q.size() > size) {
            sum -= q.front();
            q.pop();
        }
        return (sum += val)/q.size();
    }

private:
    int size;
    double sum = 0;
    queue<int> q;
};
```

347. Top K Frequent Elements(●~▽~●)

Medium

Given a non-empty array of integers, return the k most frequent elements.

Example 1:

Input: nums = [1,1,1,2,2,3], k = 2

Output: [1,2]

Example 2:

Input: nums = [1], k = 1

Output: [1]

Note:

- You may assume k is always valid, $1 \leq k \leq$ number of unique elements.
- Your algorithm's time complexity **must be** better than $O(n \log n)$, where n is the array's size.

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {

    }
};
```

```

class Solution {
public:
    vector<int> topKFrequent(vector<int> &nums, int k) {
        unordered_map<int, int> counts;
        for(auto i : nums) ++counts[i];
        vector<vector<int>> buckets(nums.size() + 1);
        for(auto &k : counts) buckets[k.second].push_back(k.first);
        vector<int> res;
        for(int t = buckets.size()-1; t >= 0; t--) {
            for(auto i : buckets[t]) {
                res.push_back(i);
                if (res.size() == k) return res;
            }
        }
        return res;
    }
};

```


348. Design Tic-Tac-Toe

Design a Tic-tac-toe game that is played between two players on a $n \times n$ grid.

You may assume the following rules:

1. A move is guaranteed to be valid and is placed on an empty block.
2. Once a winning condition is reached, no more moves is allowed.
3. A player who succeeds in placing n of their marks in a horizontal, vertical, or diagonal row wins the game.

Example:

Given $n = 3$, assume that player 1 is "X" and player 2 is "O" in the board.

```
TicTacToe toe = new TicTacToe(3);
```

```
toe.move(0, 0, 1); -> Returns 0 (no one wins)
```

```
|X| | |
```

```
| | | | // Player 1 makes a move at (0, 0).
```

```
| | | |
```

```
toe.move(0, 2, 2); -> Returns 0 (no one wins)
```

```
|X| |O|
```

```
| | | | // Player 2 makes a move at (0, 2).
```

```
| | | |
```

```
toe.move(2, 2, 1); -> Returns 0 (no one wins)
```

```
|X| |O|
```

```
| | | | // Player 1 makes a move at (2, 2).
```

```
| | |X|
```

```
toe.move(1, 1, 2); -> Returns 0 (no one wins)
```

```
|X| |O|
```

```
| |O| |    // Player 2 makes a move at (1, 1).
```

```
| | |X|
```

```
toe.move(2, 0, 1); -> Returns 0 (no one wins)
```

```
|X| |O|
```

```
| |O| |    // Player 1 makes a move at (2, 0).
```

```
|X| |X|
```

```
toe.move(1, 0, 2); -> Returns 0 (no one wins)
```

```
|X| |O|
```

```
|O|O| |    // Player 2 makes a move at (1, 0).
```

```
|X| |X|
```

```
toe.move(2, 1, 1); -> Returns 1 (player 1 wins)
```

```
|X| |O|
```

```
|O|O| |    // Player 1 makes a move at (2, 1).
```

```
|X|X|X|
```

Follow up:

Could you do better than $O(n^2)$ per `move()` operation?

```

class TicTacToe {
public:
    /** Initialize your data structure here. */
    TicTacToe(int n) {
        this->n = n;
        R.resize(n, 0);
        C.resize(n, 0);
    }

    int move(int row, int col, int player) {
        int add = (player == 2) ? -1 : 1;
        if (judge(row, add, R) || judge(col, add, C)) return player;
        else if (row == col && judge_diag(0, add)) return player;
        else if (row + col == n-1 && judge_diag(1, add)) return player;
        else return 0;
    }

private:
    int n;
    vector<int> R, C, diag{0, 0};

    bool judge(int row, int add, vector<int> &R) {
        if (abs(R[row] += add) == n) return true;
        else return false;
    }

    bool judge_diag(int type, int add) {
        if (abs(diag[type] += add) == n) return true;
        else return false;
    }
};

```

349. Intersection of Two Arrays

Easy

Given two arrays, write a function to compute their intersection.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [9,4]

Note:

- Each element in the result must be unique.
- The result can be in any order.

```
class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {

    }
};
```

```

class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
        sort(nums1.begin(), nums1.end());
        sort(nums2.begin(), nums2.end());
        int n1 = (int)nums1.size(), n2 = (int)nums2.size();
        int i1 = 0, i2 = 0;
        vector<int> res;
        while (i1 < n1 && i2 < n2){
            if (nums1[i1] == nums2[i2]) {
                res.push_back(nums1[i1]);
                i1++; i2++;
                while (i1 < n1 && nums1[i1] == nums1[i1-1]) i1++;
                while (i2 < n2 && nums2[i2] == nums2[i2-1]) i2++;
            }
            else if (nums1[i1] > nums2[i2]) {
                i2++;
                while (i2 < n2 && nums2[i2] == nums2[i2-1]) i2++;
            }
            else {
                i1++;
                while (i1 < n1 && nums1[i1] == nums1[i1-1]) i1++;
            }
        }
        return res;
    }
};

```

```
class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
        vector<int> res;
        unordered_map<int, bool> m;
        for (auto i : nums1) m[i] = true;
        for (auto i : nums2) if (m[i]){
            res.push_back(i);
            m[i] = false;
        }
        return res;
    }
};
```

350. Intersection of Two Arrays II

Easy

Given two arrays, write a function to compute their intersection.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2,2]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [4,9]

Note:

- Each element in the result should appear as many times as it shows in both arrays.
- The result can be in any order.

Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?
- What if *nums1*'s size is small compared to *nums2*'s size? Which algorithm is better?
- What if elements of *nums2* are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

```
class Solution {  
public:  
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {  
  
    }  
};
```

```
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
        sort(nums1.begin(), nums1.end());
        sort(nums2.begin(), nums2.end());
        int n1 = (int)nums1.size(), n2 = (int)nums2.size();
        int i1 = 0, i2 = 0;
        vector<int> res;
        while (i1 < n1 && i2 < n2){
            if(nums1[i1] == nums2[i2]) {
                res.push_back(nums1[i1]);
                i1++;
                i2++;
            }
            else if(nums1[i1] > nums2[i2]) i2++;
            else i1++;
        }
        return res;
    }
};
```



```
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
        vector<int> res;
        unordered_map<int, int> m;
        for (auto i : nums1) m[i]++;
        for (auto i : nums2) if (m[i] > 0){
            res.push_back(i);
            m[i]--;
        }
        return res;
    }
};
```

352. Data Stream as Disjoint Intervals

Hard

Given a data stream input of non-negative integers $a_1, a_2, \dots, a_n, \dots$, summarize the numbers seen so far as a list of disjoint intervals.

For example, suppose the integers from the data stream are 1, 3, 7, 2, 6, ..., then the summary will be:

[1, 1]

[1, 1], [3, 3]

[1, 1], [3, 3], [7, 7]

[1, 3], [7, 7]

[1, 3], [6, 7]

Follow up:

What if there are lots of merges and the number of disjoint intervals are small compared to the data stream's size?

```

class SummaryRanges {
public:
    /** Initialize your data structure here. */
    SummaryRanges() {}

    void addNum(int val) {
        if (visited.count(val)) return;
        visited.insert(val);
        auto pos = lower_bound(v.begin(), v.end(), vector<int>{val, val});
        bool ok = false;
        if (pos != v.end() && (*pos)[0] == val+1) {
            (*pos)[0] = val;
            ok = true;
        }
        if (pos != v.begin() && (*prev(pos))[1] == val-1) {
            (*prev(pos))[1] = val;
            ok = true;
        }
        if (!ok) v.insert(pos, vector<int>{val, val});
        else if (pos != v.begin() && pos != v.end()) {
            if ((*prev(pos))[1] == (*pos)[0]) {
                (*prev(pos))[1] = (*pos)[1];
                v.erase(pos);
            }
        }
    }

    vector<vector<int>> getIntervals() {
        return v;
    }
private:
    unordered_set<int> visited;
    vector<vector<int>> v;
};

/**
 * Your SummaryRanges object will be instantiated and called as such:
 * SummaryRanges* obj = new SummaryRanges();
 * obj->addNum(val);
 * vector<vector<int>> param_2 = obj->getIntervals();
 */

```

354. Russian Doll Envelopes

Hard

You have a number of envelopes with widths and heights given as a pair of integers (w, h) . One envelope can fit into another if and only if both the width and height of one envelope is greater than the width and height of the other envelope.

What is the maximum number of envelopes can you Russian doll? (put one inside other)

Note:

Rotation is not allowed.

Example:

Input: `[[5,4],[6,4],[6,7],[2,3]]`

Output: 3

Explanation: The maximum number of envelopes you can Russian doll is 3 (`[2,3] => [5,4] => [6,7]`)

```
class Solution {
public:
    int maxEnvelopes(vector<vector<int>>& envelopes) {
        sort(envelopes.begin(), envelopes.end(), [](vector<int> &a,
vector<int> &b){
            return a[0] < b[0] || (a[0] == b[0] && a[1] > b[1]);
        });
        vector<int> dp;
        for (auto &v : envelopes) {
            auto iter = lower_bound(dp.begin(), dp.end(), v[1]);
            if (iter == dp.end()) dp.push_back(v[1]);
            else if (v[1] < *iter) *iter = v[1];
        }
        return dp.size();
    }
};
```

355. Design Twitter

Medium

583148FavoriteShare

Design a simplified version of Twitter where users can post tweets, follow/unfollow another user and is able to see the 10 most recent tweets in the user's news feed. Your design should support the following methods:

1. **postTweet(userId, tweetId):** Compose a new tweet.
2. **getNewsFeed(userId):** Retrieve the 10 most recent tweet ids in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user herself. Tweets must be ordered from most recent to least recent.
3. **follow(followerId, followeeId):** Follower follows a followee.
4. **unfollow(followerId, followeeId):** Follower unfollows a followee.

Example:

```
Twitter twitter = new Twitter();

// User 1 posts a new tweet (id = 5).

twitter.postTweet(1, 5);

// User 1's news feed should return a list with 1 tweet id -> [5].

twitter.getNewsFeed(1);

// User 1 follows user 2.

twitter.follow(1, 2);

// User 2 posts a new tweet (id = 6).

twitter.postTweet(2, 6);

// User 1's news feed should return a list with 2 tweet ids -> [6, 5].

// Tweet id 6 should precede tweet id 5 because it is posted after tweet id 5.

twitter.getNewsFeed(1);

// User 1 unfollows user 2.

twitter.unfollow(1, 2);

// User 1's news feed should return a list with 1 tweet id -> [5],
```

```
// since user 1 is no longer following user 2.
```

```
twitter.getNewsFeed(1);
```

```
class Twitter {
public:
    struct cmp {
        template <typename T>
        bool operator() (T const &lhs, T const &rhs) {
            return lhs.first > rhs.first;
        }
    };
    Twitter() {}

    void postTweet(int userId, int tweetId) {
        follow(userId, userId);
        twitter[userId].push_back(make_pair(timestamp++, tweetId));
    }

    vector<int> getNewsFeed(int userId) {
        priority_queue<pair<int, int>, vector<pair<int, int>>, cmp> pq;
        for (auto &i : user[userId]) {
            for (auto &pii : twitter[i]) {
                if (!pq.empty() && pq.top().first > pii.first && pq.size() > 10)
                    break;
                pq.push(pii);
                if (pq.size() > 10) pq.pop();
            }
        }
        vector<int> res;
        while (!pq.empty()) {
            res.push_back(pq.top().second);
            pq.pop();
        }
        reverse(res.begin(), res.end());
        return res;
    }
};
```

```
void follow(int followerId, int followeeId) {
    user[followerId].insert(followeeId);
}

void unfollow(int followerId, int followeeId) {
    if (followerId != followeeId) {
        user[followerId].erase(followeeId);
    }
}

private:
    int timestamp = 0;
    unordered_map<int, unordered_set<int>> user;
    unordered_map<int, vector<pair<int, int>>> twitter;
};
```

357. Count Numbers with Unique Digits

Medium

Given a **non-negative** integer n , count all numbers with unique digits, x , where $0 \leq x < 10^n$.

Example:

Input: 2

Output: 91

Explanation: The answer should be the total numbers in the range of $0 \leq x < 100$,

excluding 11,22,33,44,55,66,77,88,99

```
class Solution {
public:
    int countNumbersWithUniqueDigits(int n) {
        int res = 1;
        while (n-- >= 1) {
            int t = 9;
            for (int i = 9; i >= 10-n; --i) {
                t *= i;
            }
            res += t;
        }
        return res;
    }
};
```


363. Max Sum of Rectangle No Larger Than K

Hard

Given a non-empty 2D matrix *matrix* and an integer *k*, find the max sum of a rectangle in the *matrix* such that its sum is no larger than *k*.

Example:

Input: matrix = [[1,0,1],[0,-2,3]], k = 2

Output: 2

Explanation: Because the sum of rectangle [[0, 1], [-2, 3]] is 2,

and 2 is the max number no larger than k (k = 2).

Note:

1. The rectangle inside the matrix must have an area > 0.
2. What if the number of rows is much larger than the number of columns?

```

class Solution {
public:
    int maxSumSubmatrix(vector<vector<int>>& matrix, int k) {
        if (matrix.empty()) return 0;
        int row = matrix.size(), col = matrix[0].size(), res = INT_MIN;
        for (int l = 0; l < col; ++l) {
            vector<int> sums(row, 0);
            for (int r = l; r < col; ++r) {
                for (int i = 0; i < row; ++i) {
                    sums[i] += matrix[i][r];
                }

                // Find the max subarray no more than K
                set<int> mySet;
                mySet.insert(0);
                int curSum = 0, curMax = INT_MIN;
                for (int sum : sums) {
                    curSum += sum;
                    auto it = mySet.lower_bound(curSum - k);
                    if (it != mySet.end()) curMax = std::max(curMax, curSum - *it);
                    mySet.insert(curSum);
                }
                res = std::max(res, curMax);
            }
        }
        return res;
    }
};

```

365. Water and Jug Problem

Medium

You are given two jugs with capacities x and y litres. There is an infinite amount of water supply available. You need to determine whether it is possible to measure exactly z litres using these two jugs.

If z liters of water is measurable, you must have z liters of water contained within **one or both buckets** by the end.

Operations allowed:

- Fill any of the jugs completely with water.
- Empty any of the jugs.
- Pour water from one jug into another till the other jug is completely full or the first jug itself is empty.

Example 1: (From the famous *"Die Hard"* example)

Input: $x = 3, y = 5, z = 4$

Output: True

Example 2:

Input: $x = 2, y = 6, z = 5$

Output: False

```
class Solution {
public:
    bool canMeasureWater(int x, int y, int z) {
        return z == 0 || (z - x <= y && z % gcd(x, y) == 0);
    }

private:
    int gcd(int x, int y) {
        return y == 0 ? x : gcd(y, x % y);
    }
};
```

367. Valid Perfect Square

Easy

Given a positive integer *num*, write a function which returns True if *num* is a perfect square else False.

Note: Do not use any built-in library function such as `sqrt`.

Example 1:

Input: 16

Output: true

Example 2:

Input: 14

Output: false

```

class Solution {
public:
    bool isPerfectSquare(int num) {
        long r = num;
        while (r*r > num)
            r = (r + num/r) / 2;
        return r*r == num;
    }
};

```

```

class Solution {
public:
    bool isPerfectSquare(int num) {
        long long l = 0, r = num;
        while (l <= r) {
            auto mid = l + (r-l)/2;
            auto sqmid = mid * mid;
            if (sqmid == num) return true;
            else if (sqmid < num) l = mid + 1;
            else r = mid - 1;
        }
        return false;
    }
};

```

368. Largest Divisible Subset

Medium

Given a set of **distinct** positive integers, find the largest subset such that every pair (S_i, S_j) of elements in this subset satisfies:

$$S_i \% S_j = 0 \text{ or } S_j \% S_i = 0.$$

If there are multiple solutions, return any subset is fine.

Example 1:

Input: [1,2,3]

Output: [1,2] (of course, [1,3] will also be ok)

Example 2:

Input: [1,2,4,8]

Output: [1,2,4,8]

```

class Solution {
public:
    vector<int> largestDivisibleSubset(vector<int>& nums) {
        vector<int> res;
        if (nums.empty()) return res;
        int n = nums.size(), p = 0;
        sort(nums.begin(), nums.end());
        vector<pair<int, int>> dp(n);
        for (int i = 0; i < n; i++) {
            dp[i] = {1, -1};
            for (int j = 0; j < i; j++) {
                if (nums[i] % nums[j] == 0) {
                    if (dp[j].first+1 > dp[i].first) {
                        dp[i] = {dp[j].first+1, j};
                    }
                }
            }
            if (dp[i].first > dp[p].first) p = i;
        }
        while (p != -1) {
            res.push_back(nums[p]);
            p = dp[p].second;
        }
        return res;
    }
};

```

371. Sum of Two Integers

Easy

Calculate the sum of two integers a and b , but you are **not allowed** to use the operator `+` and `-`.

Example 1:

Input: $a = 1, b = 2$

Output: 3

Example 2:

Input: $a = -2, b = 3$

Output: 1

```
class Solution {
public:
    int getSum(int a, int b) {
        while (b != 0) {
            int carry = a & b;
            a = a ^ b;
            b = (carry & 0xffffffff) << 1;
        }
        return a;
    }
};
```


372. Super Pow

Medium

Your task is to calculate $a^b \bmod 1337$ where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Example 1:

Input: $a = 2, b = [3]$

Output: 8

Example 2:

Input: $a = 2, b = [1,0]$

Output: 1024

```
class Solution {
public:
    int superPow(int a, vector<int>& b) {
        int pre = 1;
        a %= 1337;
        for (int i = b.size()-1; i >= 0; i--) {
            int cur = POW(a, b[i]);
            pre = pre*cur % 1337;
            a = POW(a, 10);
        }
        return pre;
    }

private:
    int POW(int x, int n) {
        int ret = 1;
        while (n-->0) {
            ret = ret*x % 1337;
        }
        return ret;
    }
};
```

373. Find K Pairs with Smallest Sums

Medium

You are given two integer arrays **nums1** and **nums2** sorted in ascending order and an integer **k**.

Define a pair **(u,v)** which consists of one element from the first array and one element from the second array.

Find the **k** pairs **(u₁,v₁),(u₂,v₂) ... (u_k,v_k)** with the smallest sums.

Example 1:

Input: nums1 = [1,7,11], nums2 = [2,4,6], k = 3

Output: [[1,2],[1,4],[1,6]]

Explanation: The first 3 pairs are returned from the sequence:

[1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]

Example 2:

Input: nums1 = [1,1,2], nums2 = [1,2,3], k = 2

Output: [1,1],[1,1]

Explanation: The first 2 pairs are returned from the sequence:

[1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]

Example 3:

Input: nums1 = [1,2], nums2 = [3], k = 3

Output: [1,3],[2,3]

Explanation: All possible pairs are returned from the sequence: [1,3],[2,3]

```

class Solution {
public:
    struct node{
        int x, y, val;
        bool operator < (const node& rhs) const {
            return val > rhs.val;
        }
    };

    vector<vector<int>> kSmallestPairs(vector<int>& nums1, vector<int>& nums2,
int k) {
        if (nums1.empty() || nums2.empty()) return {};
        int n = nums1.size(), m = nums2.size();
        priority_queue<node> pq;
        pq.push({0, 0, nums1[0] + nums2[0]});
        vector<vector<int>> res;
        while (!pq.empty() && res.size() < k) {
            auto t = pq.top();
            res.push_back({nums1[t.x], nums2[t.y]});
            pq.pop();
            if (t.x+1 < n) pq.push({t.x+1, t.y, nums1[t.x+1] + nums2[t.y]});
            if (t.x == 0 && t.y+1 < m) pq.push({t.x, t.y+1, nums1[t.x] +
nums2[t.y+1]});
        }
        return res;
    }
};

```

374. Guess Number Higher or Lower

Easy

We are playing the Guess Game. The game is as follows:

I pick a number from **1** to ***n***. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

You call a pre-defined API `guess(int num)` which returns 3 possible results (`-1`, `1`, or `0`):

`-1` : My number is lower

`1` : My number is higher

`0` : Congrats! You got it!

Example :

Input: `n = 10`, `pick = 6`

Output: `6`

```
class Solution {
public:
    int guessNumber(int n) {
        int l = 1, r = n;
        while (l <= n) {
            int mid = l + (r-l)/2;
            switch(guess(mid)) {
                case 0: return mid; break;
                case -1: r = mid-1; break;
                case 1: l = mid+1; break;
            }
        }
        return -1;
    }
};
```

375. Guess Number Higher or Lower II

Medium

We are playing the Guess Game. The game is as follows:

I pick a number from **1** to **n**. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number I picked is higher or lower.

However, when you guess a particular number x , and you guess wrong, you pay $\$x$. You win the game when you guess the number I picked.

Example:

$n = 10$, I pick 8.

First round: You guess 5, I tell you that it's higher. You pay \$5.

Second round: You guess 7, I tell you that it's higher. You pay \$7.

Third round: You guess 9, I tell you that it's lower. You pay \$9.

Game over. 8 is the number I picked.

You end up paying $\$5 + \$7 + \$9 = \21 .

Given a particular $n \geq 1$, find out how much money you need to have to guarantee a **win**.

//存在 $O(n^2)$ algorithm, 待补充

```
class Solution {
public:
    int getMoneyAmount(int n) {
        dp.resize(n+1, vector<int>(n+1));
        return f(0, n);
    }
private:
    vector<vector<int>> dp;

    int f(int i, int j) {
        if (i >= j) return 0;
        else if (dp[i][j] > 0) return dp[i][j];
        else if (i == j) return dp[i][j] = i;
        dp[i][j] = INT_MAX;
        for (int k = i; k <= j; ++k) {
            dp[i][j] = min(dp[i][j], k + max(f(i, k-1), f(k+1, j)));
        }
        return dp[i][j];
    }
};
```


376. Wiggle Subsequence

Medium

A sequence of numbers is called a **wiggle sequence** if the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with fewer than two elements is trivially a wiggle sequence.

For example, `[1, 7, 4, 9, 2, 5]` is a wiggle sequence because the differences `(6, -3, 5, -7, 3)` are alternately positive and negative. In contrast, `[1, 4, 7, 2, 5]` and `[1, 7, 4, 5, 5]` are not wiggle sequences, the first because its first two differences are positive and the second because its last difference is zero.

Given a sequence of integers, return the length of the longest subsequence that is a wiggle sequence. A subsequence is obtained by deleting some number of elements (eventually, also zero) from the original sequence, leaving the remaining elements in their original order.

Example 1:

Input: `[1,7,4,9,2,5]`

Output: 6

Explanation: The entire sequence is a wiggle sequence.

Example 2:

Input: `[1,17,5,10,13,15,10,5,16,8]`

Output: 7

Explanation: There are several subsequences that achieve this length. One is `[1,17,10,13,10,16,8]`.

Example 3:

Input: `[1,2,3,4,5,6,7,8,9]`

Output: 2

Follow up:

Can you do it in $O(n)$ time?

```
class Solution {
public:
    int wiggleMaxLength(vector<int>& nums) {
        int n = nums.size();
        if (n < 2) return n;
        int down = 1, up = 1;
        for (int i = 1; i < n; i++) {
            if (nums[i] > nums[i-1])
                up = down + 1;
            else if (nums[i] < nums[i-1])
                down = up + 1;
        }
        return max(down, up);
    }
};
```

377. Combination Sum IV

Medium

Given an integer array with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer target.

Example:

```
nums = [1, 2, 3]
```

```
target = 4
```

The possible combination ways are:

(1, 1, 1, 1)

(1, 1, 2)

(1, 2, 1)

(1, 3)

(2, 1, 1)

(2, 2)

(3, 1)

Note that different sequences are counted as different combinations.

Therefore the output is 7.

Follow up:

What if negative numbers are allowed in the given array?

How does it change the problem?

What limitation we need to add to the question to allow negative numbers?

Credits:

Special thanks to [@pbrother](#) for adding this problem and creating all test cases.

```
class Solution {
public:
    int combinationSum4(vector<int>& nums, int target) {
        vector<unsigned int> dp(target + 1, 0);
        dp[0] = 1;
        for (int i = 1; i <= target; ++i) {
            for (int x : nums) {
                if (i >= x) dp[i] += dp[i - x];
            }
        }
        return dp[target];
    }
};
```

378. Kth Smallest Element in a Sorted Matrix

Medium

147692FavoriteShare

Given a $n \times n$ matrix where each of the rows and columns are sorted in ascending order, find the k th smallest element in the matrix.

Note that it is the k th smallest element in the sorted order, not the k th distinct element.

Example:

```
matrix = [  
  [ 1,  5,  9],  
  [10, 11, 13],  
  [12, 13, 15]  
],  
k = 8,  
  
return 13.
```

Note:

You may assume k is always valid, $1 \leq k \leq n^2$.

```

class Solution {
public:
    struct node{
        int x, y, value;
        node(int x, int y, int value):x(x), y(y), value(value){}
        bool operator < (const node &rhs) const{
            return value > rhs.value;
        }
    };

    int kthSmallest(vector<vector<int>>& matrix, int k) {
        const int dx[] = {1, 0};
        const int dy[] = {0, 1};
        int n = matrix.size();
        priority_queue<node> pq;
        pq.push({0, 0, matrix[0][0]});
        matrix[0][0] = INT_MIN;
        while(--k) {
            node t = pq.top();
            pq.pop();
            for (int i = 0; i < 2; i++) {
                int x = t.x+dx[i], y = t.y+dy[i];
                if (x < n && y < n && matrix[x][y] != INT_MIN) {
                    pq.push(node{x, y, matrix[x][y]});
                    matrix[x][y] = INT_MIN;
                }
            }
        }
        return pq.top().value;
    }
};

```

380. Insert Delete GetRandom O(1)

Medium

Design a data structure that supports all following operations in *average* **O(1)** time.

1. `insert(val)`: Inserts an item `val` to the set if not already present.
2. `remove(val)`: Removes an item `val` from the set if present.
3. `getRandom`: Returns a random element from current set of elements. Each element must have the **same probability** of being returned.

Example:

```
// Init an empty set.

RandomizedSet randomSet = new RandomizedSet();

// Inserts 1 to the set. Returns true as 1 was inserted successfully.
randomSet.insert(1);

// Returns false as 2 does not exist in the set.
randomSet.remove(2);

// Inserts 2 to the set, returns true. Set now contains [1,2].
randomSet.insert(2);

// getRandom should return either 1 or 2 randomly.
randomSet.getRandom();

// Removes 1 from the set, returns true. Set now contains [2].
randomSet.remove(1);

// 2 was already in the set, so return false.
randomSet.insert(2);

// Since 2 is the only number in the set, getRandom always return 2.
randomSet.getRandom();
```

```

class RandomizedSet {
public:
    /** Initialize your data structure here. */
    RandomizedSet(): sz(0) {}

    /** Inserts a value to the set. Returns true if the set did not already
    contain the specified element. */
    bool insert(int val) {
        if (m.count(val)) return false;
        m[val] = sz++;
        nums.push_back(val);
        return true;
    }

    /** Removes a value from the set. Returns true if the set contained the
    specified element. */
    bool remove(int val) {
        if (!m.count(val)) return false;
        int pos = m[val];
        m[nums[--sz]] = pos;
        nums[pos] = nums[sz];
        m.erase(val);
        nums.pop_back();
        return true;
    }

    /** Get a random element from the set. */
    int getRandom() {
        return nums[rand() % sz];
    }

private:
    unordered_map<int, int> m; // <number, pos>
    int sz;                  // the size of nums
    vector<int> nums;
};

/**
 * Your RandomizedSet object will be instantiated and called as such:
 * RandomizedSet* obj = new RandomizedSet();
 * bool param_1 = obj->insert(val);
 * bool param_2 = obj->remove(val);
 * int param_3 = obj->getRandom();
 */

```


381. Insert Delete GetRandom O(1) - Duplicates allowed

Hard

Design a data structure that supports all following operations in *average* **O(1)** time.

Note: Duplicate elements are allowed.

1. `insert(val)`: Inserts an item `val` to the collection.
2. `remove(val)`: Removes an item `val` from the collection if present.
3. `getRandom`: Returns a random element from current collection of elements. The probability of each element being returned is **linearly related** to the number of same value the collection contains.

Example:

```
// Init an empty collection.

RandomizedCollection collection = new RandomizedCollection();

// Inserts 1 to the collection. Returns true as the collection did not contain 1.

collection.insert(1);

// Inserts another 1 to the collection. Returns false as the collection contained 1. Collection now
contains [1,1].

collection.insert(1);

// Inserts 2 to the collection, returns true. Collection now contains [1,1,2].

collection.insert(2);

// getRandom should return 1 with the probability 2/3, and returns 2 with the probability 1/3.

collection.getRandom();

// Removes 1 from the collection, returns true. Collection now contains [1,2].

collection.remove(1);

// getRandom should return 1 and 2 both equally likely.

collection.getRandom();
```

//可能还有更好思路

```
class RandomizedCollection {
public:
    RandomizedCollection() {}

    bool insert(int val) {
        bool ret = !mp.count(val);
        mp[val].push_back(nums.size());
        nums.push_back({val, mp[val].size()-1});
        return ret;
    }

    bool remove(int val) {
        if (!mp.count(val)) return false;
        auto last = nums.back();
        auto &delete_ps = mp[val];
        int i = rand() % (int)delete_ps.size();
        int delete_p = mp[val][i];
        if (i != delete_ps.size()-1) {
            delete_ps[i] = delete_ps.back();
        }
        mp[last.first][last.second] = delete_p;
        nums[delete_p] = last;
        delete_ps.pop_back();
        if (delete_ps.empty()) mp.erase(val);
        nums.pop_back();
        return true;
    }

    /** Get a random element from the collection. */
    int getRandom() {
        return nums[rand() % nums.size()].first;
    }
private:
    vector<pair<int, int>> nums;
    unordered_map<int, vector<int>> mp;
};
```

382. Linked List Random Node

Medium

Given a singly linked list, return a random node's value from the linked list. Each node must have the **same probability** of being chosen.

Follow up:

What if the linked list is extremely large and its length is unknown to you? Could you solve this efficiently without using extra space?

Example:

```
// Init a singly linked list [1,2,3].

ListNode head = new ListNode(1);

head.next = new ListNode(2);

head.next.next = new ListNode(3);

Solution solution = new Solution(head);

// getRandom() should return either 1, 2, or 3 randomly. Each element should have equal probability of
returning.

solution.getRandom();
```

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    /** @param head The linked list's head.
     * Note that the head is guaranteed to be not null, so it
     contains at least one node. */
    Solution(ListNode* head) : head(head) {}

    /** Returns a random node's value. */
    int getRandom() {
        //在此我扩展为 k 个随机取样 (水塘抽样问题)
        int k = 1, cnt = 0;
        vector<int> ret;
        ListNode *cur = head;
        while (cur != nullptr) {
            if (cnt++ < k) ret.push_back(cur->val);
            else {
                int t = rand() % cnt;
                if (t < k) ret[t] = cur->val;
            }
            cur = cur->next;
        }
        return ret[0];
    }

private:
    ListNode *head;
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(head);
 * int param_1 = obj->getRandom();
 */

```

383. Ransom Note

Easy

Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.

Each letter in the magazine string can only be used once in your ransom note.

Note:

You may assume that both strings contain only lowercase letters.

```
canConstruct("a", "b") -> false  
  
canConstruct("aa", "ab") -> false  
  
canConstruct("aa", "aab") -> true
```

```
class Solution {  
public:  
    bool canConstruct(string ransomNote, string magazine) {  
        vector<int> vec(256, 0);  
        for (const auto &c : magazine) vec[c]++;  
        for (const auto &c : ransomNote) {  
            if (--vec[c] < 0) return false;  
        }  
        return true;  
    }  
};
```

384. Shuffle an Array

Medium

Shuffle a set of numbers without duplicates.

Example:

```
// Init an array with set 1, 2, and 3.

int[] nums = {1,2,3};

Solution solution = new Solution(nums);

// Shuffle the array [1,2,3] and return its result. Any permutation of [1,2,3] must equally likely to be
// returned.

solution.shuffle();

// Resets the array back to its original configuration [1,2,3].

solution.reset();

// Returns the random shuffling of array [1,2,3].

solution.shuffle();
```

```
class Solution {
public:
    Solution(vector<int> nums) : arr(nums), nums(nums) {};

    /** Resets the array to its original configuration and return it. */
    vector<int> reset() {
        return arr = nums;
    }

    /** Returns a random shuffling of the array. */
    vector<int> shuffle() {
        int n = nums.size();
        for (int i = 0; i < n; ++i) {
            int j = rand() % (i + 1);
            swap(arr[i], arr[j]);
        }
        return arr;
    }
private:
    vector<int> arr, nums;
};
```

385. Mini Parser

Medium

Given a nested list of integers represented as a string, implement a parser to deserialize it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Note: You may assume that the string is well-formed:

- String is non-empty.
- String does not contain white spaces.
- String contains only digits 0-9, [, -, ,,].

Example 1:

Given s = "324",

You should return a NestedInteger object which contains a single integer 324.

Example 2:

Given s = "[123,[456,[789]]]",

Return a NestedInteger object containing a nested list with 2 elements:

1. An integer containing value 123.
2. A nested list containing two elements:
 - i. An integer containing value 456.
 - ii. A nested list with one element:
 - a. An integer containing value 789.


```

* class NestedInteger {
*     public:
*         NestedInteger();
*         NestedInteger(int value);
*         // Return true if this NestedInteger holds a single integer, rather than
a nested list.
*         bool isInteger() const;
*         // Return the single integer that this NestedInteger holds, if it holds
a single integer
*         // The result is undefined if this NestedInteger holds a nested list
*         int getInteger() const;
*         // Set this NestedInteger to hold a single integer.
*         void setInteger(int value);
*         // Set this NestedInteger to hold a nested list and adds a nested
integer to it.
*         void add(const NestedInteger &ni);
*         // Return the nested list that this NestedInteger holds, if it holds a
nested list
*         // The result is undefined if this NestedInteger holds a single integer
*         const vector<NestedInteger> &getList() const;
* };
class Solution {
public:
    NestedInteger deserialize(string s) {
        istringstream in(s);
        return deserializel(in);
    }
private:
    NestedInteger deserializel(istringstream &in) {
        int number;
        if (in >> number)
            return NestedInteger(number);
        in.clear(); //not clear all the message, bit clear error state flags
        in.get();
        NestedInteger list;
        while (in.peek() != ']') {
            list.add(deserializel(in));
            if (in.peek() == ',')
                in.get();
        }
        in.get();
        return list;
    }
};

```

386. Lexicographical Numbers

Medium

Given an integer n , return 1 - n in lexicographical order.

For example, given 13, return: [1,10,11,12,13,2,3,4,5,6,7,8,9].

Please optimize your algorithm to use less time and space. The input size may be as large as 5,000,000.

```
class Solution {
public:
    vector<int> lexicalOrder(int n) {
        dfs(0, n);
        return res;
    }

private:
    vector<int> res;
    void dfs(int i, int n) {
        if (i > n) return;
        for (int j = (i == 0 ? 1 : 0); j <= 9; ++j) {
            int k = i*10 + j;
            if (k <= n) {
                res.push_back(k);
                dfs(k, n);
            }
        }
    }
};
```

387. First Unique Character in a String

Easy

Given a string, find the first non-repeating character in it and return its index. If it doesn't exist, return -1.

Examples:

```
s = "leetcode"
```

```
return 0.
```

```
s = "loveleetcode",
```

```
return 2.
```

Note: You may assume the string contains only lowercase letters.

```
class Solution {
public:
    int firstUniqChar(string s) {
        unordered_map<char, int> m;
        for (const auto &c : s) m[c]++;
        for (int i = 0; i < s.length(); i++) {
            if (m[s[i]] == 1) return i;
        }
        return -1;
    }
};
```

388. Longest Absolute File Path

Medium

Suppose we abstract our file system by a string in the following manner:

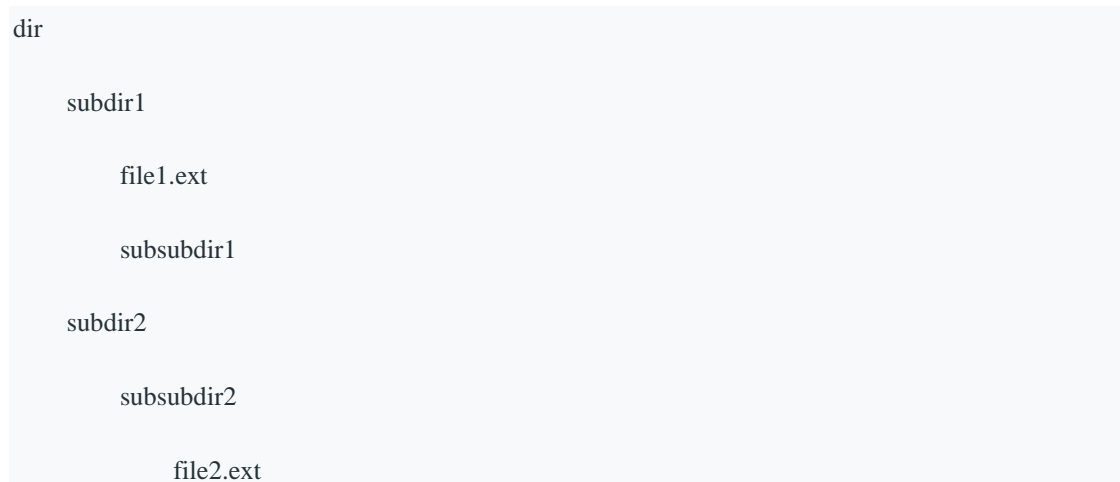
The string `"dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext"` represents:



The directory `dir` contains an empty sub-directory `subdir1` and a sub-directory `subdir2` containing a file `file.ext`.

The string

`"dir\n\tsubdir1\n\t\tfile1.ext\n\t\t\tsubsubdir1\n\tsubdir2\n\t\t\tsubsubdir2\n\t\t\t\tfile2.ext"` represents:



The directory `dir` contains two sub-directories `subdir1` and `subdir2`. `subdir1` contains a file `file1.ext` and an empty second-level sub-directory `subsubdir1`. `subdir2` contains a second-level sub-directory `subsubdir2` containing a file `file2.ext`.

We are interested in finding the longest (number of characters) absolute path to a file within our file system. For example, in the second example above, the longest absolute path is `"dir/subdir2/subsubdir2/file2.ext"`, and its length is 32 (not including the double quotes).

Given a string representing the file system in the above format, return the length of the longest absolute path to file in the abstracted file system. If there is no file in the system, return 0.

Note:

- The name of a file contains at least a `.` and an extension.
- The name of a directory or sub-directory will not contain a `..`.

Time complexity required: $O(n)$ where n is the size of the input string.

Notice that `a/aa/aaa/file1.txt` is not the longest file path, if there is another path `aaaaaaaaaaaaaaaaaaaaa/sth.png`.

```
class Solution {
public:
    int lengthLongestPath(string input) {
        input += "\n";
        stack<int> stk;
        stk.push(0);
        int res = 0, left = 0, pre = -1;
        while (1) {
            auto pos0 = input.find('\n', left);
            auto pos1 = input.find('.', left);
            if (pos0 == string::npos) break;
            int cur = 0;
            while (input[left] == '\t') {
                cur++;
                left += 1;
            }
            for (int i = cur; i <= pre; i++) stk.pop();
            stk.push((int)pos0-left+stk.top()+1);
            if (pos1 != string::npos && pos1 < pos0)
                res = max(res, stk.top()-1);
            pre = cur;
            left = pos0 + 1;
        }
        return res;
    }
};
```

389. Find the Difference

Easy

Given two strings s and t which consist of only lowercase letters.

String t is generated by random shuffling string s and then add one more letter at a random position.

Find the letter that was added in t .

Example:

Input:

$s = \text{"abcd"}$

$t = \text{"abcde"}$

Output:

e

Explanation:

'e' is the letter that was added.

```
class Solution {
public:
    char findTheDifference(string s, string t) {
        char res = 0;
        for (auto &c : s) res ^= c;
        for (auto &c : t) res ^= c;
        return res;
    }
};
```

390. Elimination Game

Medium

There is a list of sorted integers from 1 to n . Starting from left to right, remove the first number and every other number afterward until you reach the end of the list.

Repeat the previous step again, but this time from right to left, remove the right most number and every other number from the remaining numbers.

We keep repeating the steps again, alternating left to right and right to left, until a single number remains.

Find the last number that remains starting with a list of length n .

Example:

Input:

$n = 9$,

1 2 3 4 5 6 7 8 9

2 4 6 8

2 6

6

Output:

6

```
class Solution {
public:
    int lastRemaining(int n) {
        int A = 1, B = 0;
        int flag = 0;
        while (n != 1) {
            if (n % 2 == 0 && flag) B -= A;
            A *= 2;
            n /= 2;
            flag ^= 1;
        }
        return A + B;
    }
};
```


391. Perfect Rectangle

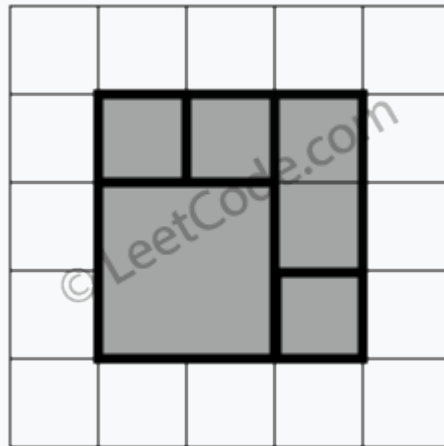
Hard

Given N axis-aligned rectangles where $N > 0$, determine if they all together form an exact cover of a rectangular region.

Each rectangle is represented as a bottom-left point and a top-right point. For example, a unit square is represented as $[1,1,2,2]$. (coordinate of bottom-left point is $(1, 1)$ and top-right point is $(2, 2)$).

Example 1:

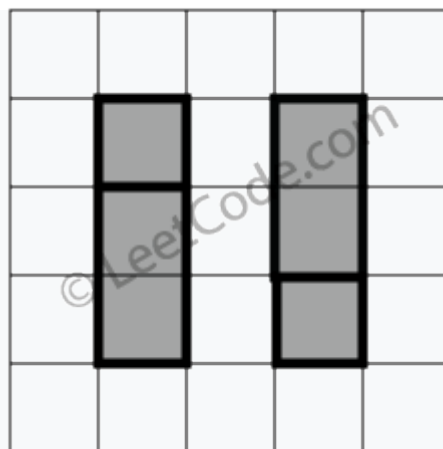
```
rectangles = [  
    [1,1,3,3],  
    [3,1,4,2],  
    [3,2,4,4],  
    [1,3,2,4],  
    [2,3,3,4]  
]
```



Return true. All 5 rectangles together form an exact cover of a rectangular region.

Example 2:

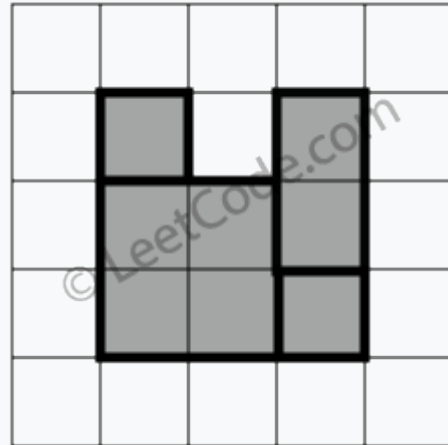
```
rectangles = [  
    [1,1,2,3],  
    [1,3,2,4],  
    [3,1,4,2],  
    [3,2,4,4]  
]
```



Return false. Because there is a gap between the two rectangular regions.

Example 3:

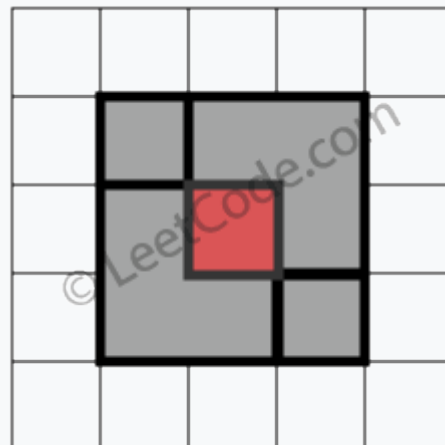
```
rectangles = [  
  [1,1,3,3],  
  [3,1,4,2],  
  [1,3,2,4],  
  [3,2,4,4]  
]
```



Return false. Because there is a gap in the top center.

Example 4:

```
rectangles = [  
  [1,1,3,3],  
  [3,1,4,2],  
  [1,3,2,4],  
  [2,2,4,4]  
]
```



Return false. Because two of the rectangles overlap with each other.

392. Is Subsequence

Easy

Given a string **s** and a string **t**, check if **s** is subsequence of **t**.

You may assume that there is only lower case English letters in both **s** and **t**. **t** is potentially a very long (length $\sim 500,000$) string, and **s** is a short string (≤ 100).

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ace" is a subsequence of "abcde" while "aec" is not).

Example 1:

s = "abc", **t** = "ahbgdc"

Return `true`.

Example 2:

s = "axc", **t** = "ahbgdc"

Return `false`.

Follow up:

If there are lots of incoming **S**, say **S**₁, **S**₂, ... , **S**_k where $k \geq 1B$, and you want to check one by one to see if **T** has its subsequence. In this scenario, how would you change your code?

Credits:

Special thanks to [@pbrother](#) for adding this problem and creating all test cases.

```
class Solution {
public:
    bool isSubsequence(string s, string t) {
        int n = s.length(), m = t.length(), j = 0;
        for (int i = 0; i < n; i++) {
            while (j < m && t[j] != s[i]) j++;
            if (j++ == m) return false;
            if (i == n-1) return true;
        }
        return true;
    }
};
```

393. UTF-8 Validation

Medium

A character in UTF8 can be from **1 to 4 bytes** long, subjected to the following rules:

1. For 1-byte character, the first bit is a 0, followed by its unicode code.
2. For n-bytes character, the first n-bits are all one's, the n+1 bit is 0, followed by n-1 bytes with most significant 2 bits being 10.

This is how the UTF-8 encoding would work:

Char. number range	UTF-8 octet sequence
(hexadecimal)	(binary)
-----+-----	
0000 0000-0000 007F	0xxxxxxx
0000 0080-0000 07FF	110xxxxx 10xxxxxx
0000 0800-0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000-0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Given an array of integers representing the data, return whether it is a valid utf-8 encoding.

Note:

The input is an array of integers. Only the **least significant 8 bits** of each integer is used to store the data. This means each integer represents only 1 byte of data.

Example 1:

data = [197, 130, 1], which represents the octet sequence: **11000101 10000010 00000001**.

Return **true**.

It is a valid utf-8 encoding for a 2-bytes character followed by a 1-byte character.

Example 2:

data = [235, 140, 4], which represented the octet sequence: **11101011 10001100 00000100**.

Return **false**.

The first 3 bits are all one's and the 4th bit is 0 means it is a 3-bytes character.

The next byte is a continuation byte which starts with 10 and that's correct.

But the second continuation byte does not start with 10, so it is invalid.

```
class Solution {
public:
    bool validUtf8(vector<int>& data) {
        int cnt = 0;
        for (auto c : data) {
            if (cnt == 0) {
                if ((c >> 5) == 0b110) cnt = 1;
                else if ((c >> 4) == 0b1110) cnt = 2;
                else if ((c >> 3) == 0b11110) cnt = 3;
                else if ((c >> 7) == 0b1) return false;
            } else {
                if ((c >> 6) != 0b10) return false;
                cnt--;
            }
        }
        return cnt == 0;
    }
};
```

394. Decode String

Medium

Given an encoded string, return its decoded string.

The encoding rule is: `k[encoded_string]`, where the *encoded_string* inside the square brackets is being repeated exactly *k* times. Note that *k* is guaranteed to be a positive integer.

You may assume that the input string is always valid; No extra white spaces, square brackets are well-formed, etc.

Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, *k*. For example, there won't be input like `3a` or `2[4]`.

Examples:

`s = "3[a]2[bc]"`, return `"aaabcbc"`.

`s = "3[a2[c]]"`, return `"accaccacc"`.

`s = "2[abc]3[cd]ef"`, return `"abcabccdcdcddef"`.

```

class Solution {
public:
    string decodeString(string s) {
        stack<string> chars;
        stack<int> nums;
        string res;
        int num = 0;
        for(auto &c : s) {
            if (isdigit(c)) num = num*10 + (c-'0');
            else if (isalpha(c)) res.push_back(c);
            else if (c == '[') {
                chars.push(res);
                nums.push(num);
                res = "";
                num = 0;
            }
            else if (c == ']') {
                string tmp = res;
                int k = nums.top();
                nums.pop();
                res = chars.top();
                chars.pop();
                while (k--) res += tmp;
            }
        }
        return res;
    }
};

```


395. Longest Substring with At Least K Repeating Characters

Medium

Find the length of the longest substring T of a given string (consists of lowercase letters only) such that every character in T appears no less than k times.

Example 1:

Input:

$s = \text{"aaabb"}, k = 3$

Output:

3

The longest substring is "aaa", as 'a' is repeated 3 times.

Example 2:

Input:

$s = \text{"ababbc"}, k = 2$

Output:

5

The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3 times.

```

class Solution {
public:
    int longestSubstring(string s, int k) {
        int cnt[256] = {0};
        for (auto &c : s) cnt[c]++;
        bool ok = true;
        for (auto &c : s) if (cnt[c] < k) {
            c = ' ';
            ok = false;
        }
        if (ok) return s.length();
        stringstream ss(s);
        int res = 0;
        while (ss >> s) {
            res = max(res, longestSubstring(s, k));
        }
        return res;
    }
};

```

396. Rotate Function

Medium

Given an array of integers A and let n to be its length.

Assume B_k to be an array obtained by rotating the array A k positions clock-wise, we define a "rotation function" F on A as follow:

$$F(k) = 0 * B_k[0] + 1 * B_k[1] + \dots + (n-1) * B_k[n-1].$$

Calculate the maximum value of $F(0), F(1), \dots, F(n-1)$.

Note:

n is guaranteed to be less than 10^5 .

Example:

$A = [4, 3, 2, 6]$

$$F(0) = (0 * 4) + (1 * 3) + (2 * 2) + (3 * 6) = 0 + 3 + 4 + 18 = 25$$

$$F(1) = (0 * 6) + (1 * 4) + (2 * 3) + (3 * 2) = 0 + 4 + 6 + 6 = 16$$

$$F(2) = (0 * 2) + (1 * 6) + (2 * 4) + (3 * 3) = 0 + 6 + 8 + 9 = 23$$

$$F(3) = (0 * 3) + (1 * 2) + (2 * 6) + (3 * 4) = 0 + 2 + 12 + 12 = 26$$

So the maximum value of $F(0), F(1), F(2), F(3)$ is $F(3) = 26$.

```
class Solution {
public:
    int maxRotateFunction(vector<int>& A) {
        long long n = A.size(), sum = 0, res = 0;
        for (int i = 0; i < n; ++i) {
            sum += A[i];
            res += i * A[i];
        }
        long long temp = res;
        for (int i = 0; i < n; ++i) {
            temp += n*A[i] - sum;
            res = max(temp, res);
        }
        return res;
    }
};
```

397. Integer Replacement

Medium

Given a positive integer n and you can do operations as follow:

1. If n is even, replace n with $n/2$.
2. If n is odd, you can replace n with either $n + 1$ or $n - 1$.

What is the minimum number of replacements needed for n to become 1?

Example 1:

Input:

8

Output:

3

Explanation:

8 -> 4 -> 2 -> 1

Example 2:

Input:

7

Output:

4

Explanation:

7 -> 8 -> 4 -> 2 -> 1

or

7 -> 6 -> 3 -> 2 -> 1

```
class Solution {
public:
    int integerReplacement(int n) {
        if (n == 1) return 0;
        else if (n == INT_MAX) return 32;
        if (m.find(n) != m.end()) return m[n];
        if (n % 2 == 0) {
            return m[n] = 1 + integerReplacement(n/2);
        }
        int a = 1 + integerReplacement(n-1);
        int b = 1 + integerReplacement(n+1);
        return m[n] = min(a, b);
    }

private:
    unordered_map<int, int> m;
};
```

398. Random Pick Index

Medium

Given an array of integers with possible duplicates, randomly output the index of a given target number. You can assume that the given target number must exist in the array.

Note:

The array size can be very large. Solution that uses too much extra space will not pass the judge.

Example:

```
int[] nums = new int[] {1,2,3,3,3};
```

```
Solution solution = new Solution(nums);
```

```
// pick(3) should return either index 2, 3, or 4 randomly. Each index should have equal probability of returning.
```

```
solution.pick(3);
```

```
// pick(1) should return 0. Since in the array only nums[0] is equal to 1.
```

```
solution.pick(1);
```

```

class Solution {
public:
    Solution(vector<int>& nums) {
        int n = nums.size();
        for (int i = 0; i < n; i++) {
            m[nums[i]].push(i);
        }
    }

    int pick(int target) {
        queue<int> &q = m[target];
        int ret = q.front();
        q.pop();
        q.push(ret);
        return ret;
    }

private:
    unordered_map<int, queue<int>> m;
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution* obj = new Solution(nums);
 * int param_1 = obj->pick(target);
 */

```


399. Evaluate Division

Medium

Equations are given in the format $A / B = k$, where A and B are variables represented as strings, and k is a real number (floating point number). Given some queries, return the answers. If the answer does not exist, return -1.0 .

Example:

Given $a / b = 2.0$, $b / c = 3.0$.

queries are: $a / c = ?$, $b / a = ?$, $a / e = ?$, $a / a = ?$, $x / x = ?$.

return $[6.0, 0.5, -1.0, 1.0, -1.0]$.

The input is: `vector<pair<string, string>> equations, vector<double>& values, vector<pair<string, string>> queries`, where `equations.size() == values.size()`, and the values are positive. This represents the equations. Return `vector<double>`.

According to the example above:

```
equations = [ ["a", "b"], ["b", "c"] ],
```

```
values = [2.0, 3.0],
```

```
queries = [ ["a", "c"], ["b", "a"], ["a", "e"], ["a", "a"], ["x", "x"] ].
```

The input is always valid. You may assume that evaluating the queries will result in no division by zero and there is no contradiction.

```

class Solution {
public:
    vector<double> calcEquation(vector<vector<string>> &es, vector<double> &vs,
vector<vector<string>> &qs) {
        for (int i = 0; i < es.size(); i++) {
            string a = es[i][0], b = es[i][1];
            if (!id.count(a)) id[a] = node(a, 1);
            if (!id.count(b)) id[b] = node(b, 1);
            Union(a, b, vs[i]);
        }
        vector<double> res;
        for (auto q : qs) {
            if (!id.count(q[0]) || !id.count(q[1]))
                res.push_back(-1);
            else {
                double va = 1, vb = 1;
                string fa_a = find(q[0], va);
                string fa_b = find(q[1], vb);
                if (fa_a != fa_b) res.push_back(-1);
                else res.push_back(va / vb);
            }
        }
        return res;
    }

private:
    struct node {
        node(string p = "", double v = 0) : parent(p), val(v){};
        string parent;
        double val;
    };

    unordered_map<string, node> id;

    string find(string s, double &v) {
        while (id[s].parent != s) {
            v = v * id[s].val;
            s = id[s].parent;
        }
        return s;
    }
}

```

```
void Union(string a, string b, double v) {  
    double va = 1.0, vb = 1.0;  
    string fa_a = find(a, va);  
    string fa_b = find(b, vb);  
    id[fa_a] = node(fa_b, v*vb/va);  
}  
};
```

400. Nth Digit

Medium

Find the n^{th} digit of the infinite integer sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

Note:

n is positive and will fit within the range of a 32-bit signed integer ($n < 2^{31}$).

Example 1:

Input:

3

Output:

3

Example 2:

Input:

11

Output:

0

Explanation:

The 11th digit of the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... is a 0, which is part of the number 10.

```
class Solution {
public:
    int findNthDigit(int n) {
        int k = 1;
        // k 位数第 n 个
        while (n > 9*pow(10, k-1)*k) {
            n -= 9*pow(10, k-1)*k;
            k++;
        }
        int m = (n-1)/k, num = pow(10, k-1)+m;
        n -= m*k;
        // num 的第 n 位, 即从右往左数第 k+1-n 位
        return (num / (int)pow(10, k-n)) % 10;
    }
};
```