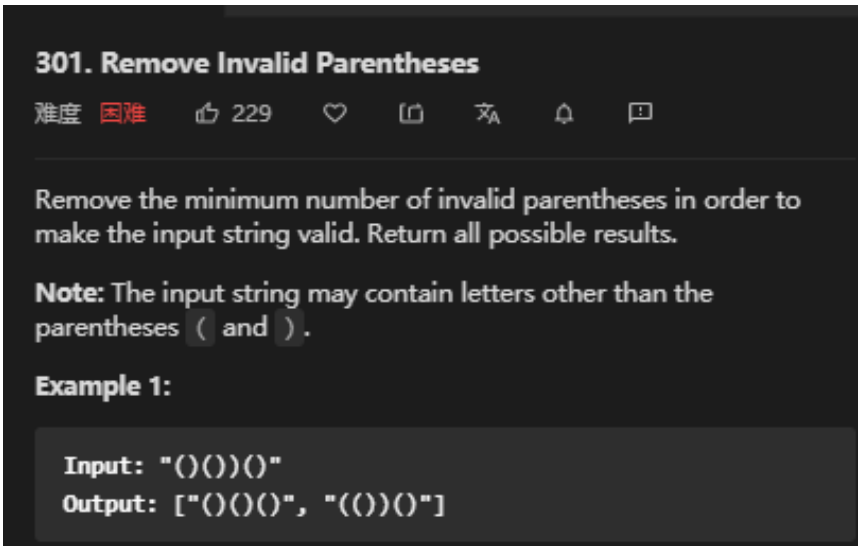
301-400

301 Remove Invalid Parentheses




```java
/*
  回溯
*/
List<String> res;
public List<String> removeInvalidParentheses(String s) {
    res  = new ArrayList<>();

    char[] chars = s.toCharArray();
    backtrack(chars, 0);

    int maxLen = 0;
    for(String str : res)
        maxLen = Math.max(maxLen, str.length());

    List<String> ans = new ArrayList<>();
    for(String str : res)
        if(str.length() == maxLen)
```

```
            ans.add(str);

    return ans;
}

private void backtrack(char[] chars, int start)
{
    if(isValid(chars) && !res.contains(toStr(chars)))
    {
        res.add(toStr(chars));
        return;
    }

    for(int i = start; i < chars.length; i++)
    {
        if((chars[i] <= 'z' && chars[i] >= 'a' ) || (chars[i] <= 'Z' && chars[i] >= 'A'))
            continue;
        char save = chars[i];
        chars[i] = '@';


        backtrack(chars, i+1);

        chars[i] = save;
    }

}

private boolean isValid(char[] chars)
{
    ArrayDeque<Character> stack = new ArrayDeque<>();

    for(int i = 0; i < chars.length; i++)
    {
        if(chars[i] == '@' ||
           (chars[i] <= 'z' && chars[i] >= 'a' ) || (chars[i] <= 'Z' && chars[i] >= 'A'))
            continue;

        if(chars[i] == '(')
            stack.push(chars[i]);
        else
        {
            if(stack.isEmpty())
                return false;
            else
                if(stack.pop() != '(')
                    return false;
```

```
        }
    }
    return stack.isEmpty();
}

private String toStr(char[] chars)
{
    StringBuilder sb = new StringBuilder();
    for(char ch : chars)
        if(ch != '@')
            sb.append(ch);
    return sb.toString();
}
```

执行用时： 63 ms ，在所有 Java 提交中击败了 34.32% 的用户

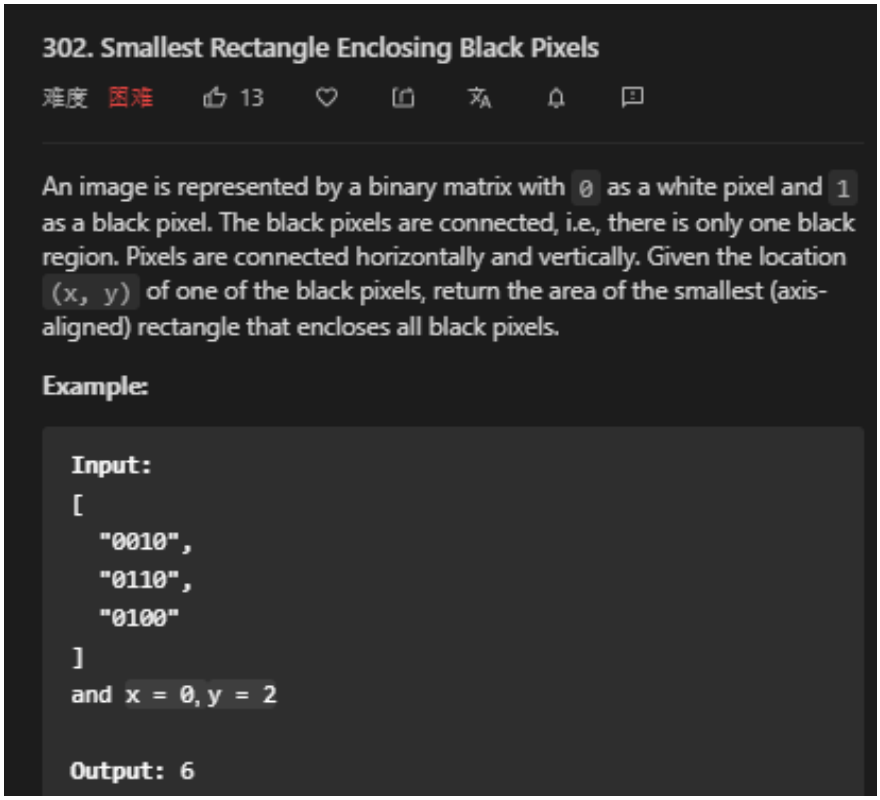
内存消耗： 39.5 MB ，在所有 Java 提交中击败了 17.56% 的用户

炫耀一下:

```
    public List<String> removeInvalidParentheses(String s) {
        HashSet<String> res = new HashSet<>();
        Deque<String> queue = new ArrayDeque<>();
        List<String> ans = new ArrayList<>();

        if(isValid(s)){
            ans.add(s);
            return ans;
        }

        HashSet<String> visited = new HashSet<>();
        queue.addLast(s);
        visited.add(s);
        boolean found = false;


        while(!queue.isEmpty()){
            int size = queue.size();

            for(int i = 0; i < size; i++){
                String curStr = queue.pollFirst();

                for(int j = 0; j < curStr.length(); j++){
                    if(curStr.charAt(j) != '(' && curStr.charAt(j) != ')')
                        continue;


                    String newStr = curStr.substring(0, j) + curStr.substring(j +
1);
```

```
                    if(visited.contains(newStr))
                        continue;

                    if(isValid(newStr)){
                        found = true;
                        res.add(newStr);
                    }

                    visited.add(newStr);
                    queue.addLast(newStr);

                }
            }

            if(found)
                break;
        }


        ans.addAll(res);
        if(ans.size() == 0)
            ans.add(s);
        return ans;
    }


    /*
        check the correctness of the str in the fomr of valid parentheses
    */
    private boolean isValid(String str){
        int leftPar = 0;
        int index = 0;

        while(index < str.length()){
            if(str.charAt(index) == '(')
                leftPar++;
            else if(str.charAt(index) == ')'){
                if(leftPar == 0)
                    return false;
                leftPar--;
            }

            index++;
        }

        return leftPar == 0;
    }

```

302 Smallest Rectangle Enclosing Black Pixels



302. Smallest Rectangle Enclosing Black Pixels

难度 困难 13

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
```

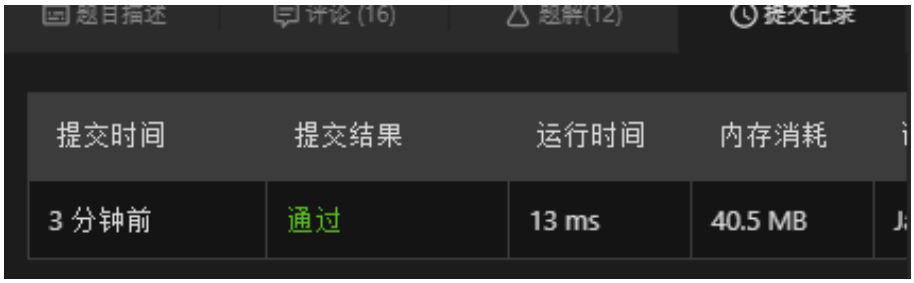
执行用时： 3 ms ，在所有 Java 提交中击败了 51.35% 的用户

内存消耗： 39.1 MB ，在所有 Java 提交中击败了 55.40% 的用户

炫耀一下：

```java
//二刷
public int minArea(char[][] image, int x, int y) {
        int row = image.length;
        int col = row == 0 ? 0 : image[0].length;
        if(col == 0)
            return 0;

        int up = row - 1;
        int down = 0;
        int left = col - 1;
        int right = 0;

        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(image[i][j] == '1'){
                    up = Math.min(up, i);
                    down = Math.max(down, i);

```

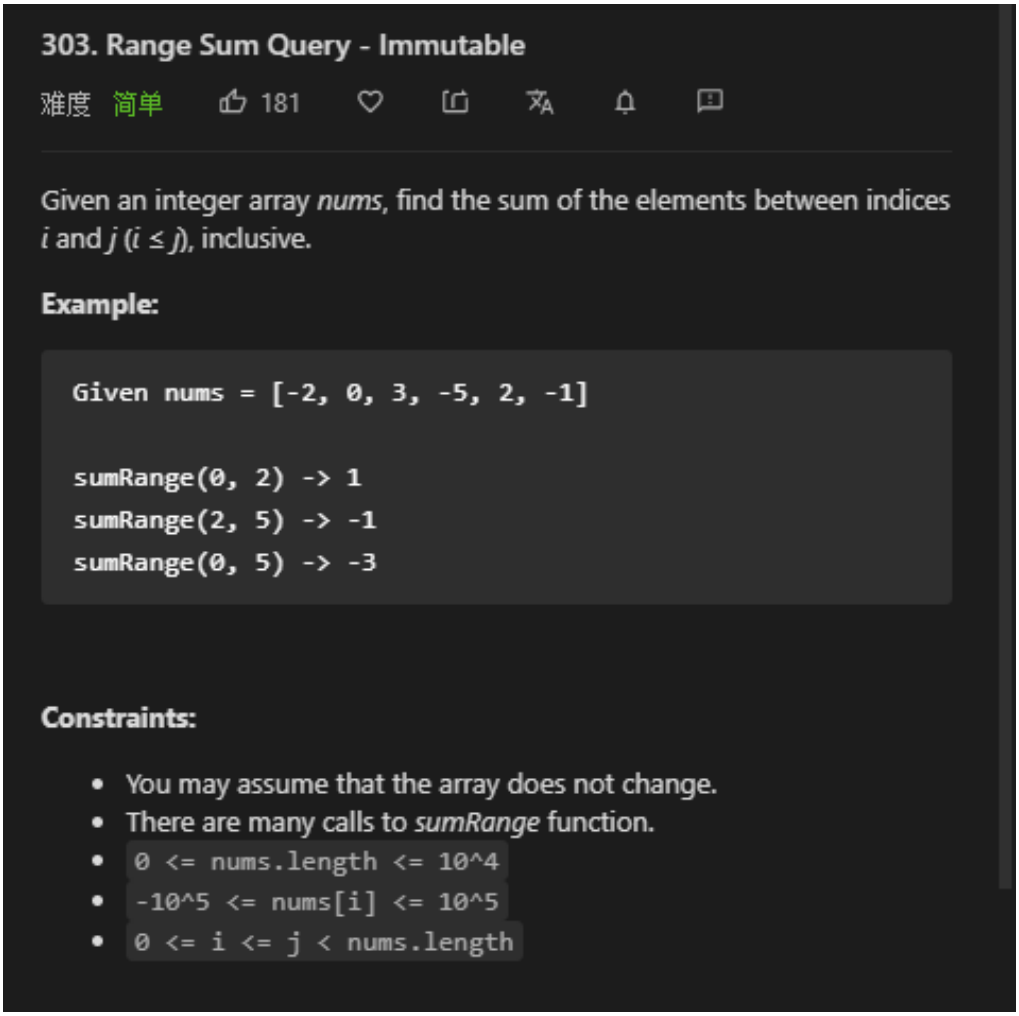
```
                    left = Math.min(left, j);
                    right = Math.max(right, j);
                }
            }
        }

        return (right - left + 1) * (down - up + 1);
    }
```




```
/*
  这个题谈不上hard,  就是个dfs
*/

class Solution {
    private List<Integer> collectionX;
    private List<Integer> collectionY;
    int row = 0;
    int column = 0;
    private boolean[][] visited;
    public int minArea(char[][] image, int x, int y) {
        collectionX = new ArrayList<>();
        collectionY = new ArrayList<>();

        row = image.length;
        if(row == 0)        return 0;
        column = image[0].length;
        visited = new boolean[row][column];

        dfs(image, x, y);

        int horizontalTop = row;
        int horizontalBot = 0;
        int verticalleft = column;
        int verticalRight = 0;

        for(int X : collectionX)
        {
            horizontalTop = Math.min(horizontalTop, X);
```

```
            horizontalBot = Math.max(horizontalBot, X);
        }

        for(int Y : collectionY)
        {
            verticalleft = Math.min(verticalleft, Y);
            verticalRight = Math.max(verticalRight, Y);
        }

        return (horizontalBot - horizontalTop + 1) * (verticalRight - verticalleft + 1);
    }

    private void dfs(char[][] image, int x, int y)
    {
        if(x < 0 || y < 0 || x >= row  || y >= column || image[x][y] == '0' || visited[x][y])
            return;

        visited[x][y] = true;
        collectionX.add(x);
        collectionY.add(y);

        dfs(image, x + 1, y);
        dfs(image, x - 1, y);
        dfs(image, x, y - 1);
        dfs(image, x, y + 1);
    }
}
```

303 Range Sum Query - Immutable



303. Range Sum Query - Immutable

难度 简单 181

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

Example:

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

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

Constraints:

- You may assume that the array does not change.
- There are many calls to *sumRange* function.
- `0 <= nums.length <= 10^4`
- `-10^5 <= nums[i] <= 10^5`
- `0 <= i <= j < nums.length`

```java
private int[] nums;
private HashMap<int[], Integer> map;
public NumArray(int[] nums) {
    this.nums = nums;
    map = new HashMap<>();
}

public int sumRange(int i, int j) {
    if(map.containsKey(new int[]{i, j}))
        return map.get(new int[]{i, j});

    int total = 0;
    for(int k = i; k <= j; k++)
        total += nums[k];
    map.put(new int[]{i, j}, total);
    return total;
}
```

```java
//更加简洁的方法
public class NumArray {
    int[] nums;
```

```
4
5       public NumArray(int[] nums) {
6           for(int i = 1; i < nums.length; i++)
7               nums[i] += nums[i - 1];
8
9           this.nums = nums;
10      }
11
12      public int sumRange(int i, int j) {
13          if(i == 0)
14              return nums[j];
15
16          return nums[j] - nums[i - 1];
17      }
18  }
```

304 Range Sum Query 2D --Immutable

304. Range Sum Query 2D - Immutable

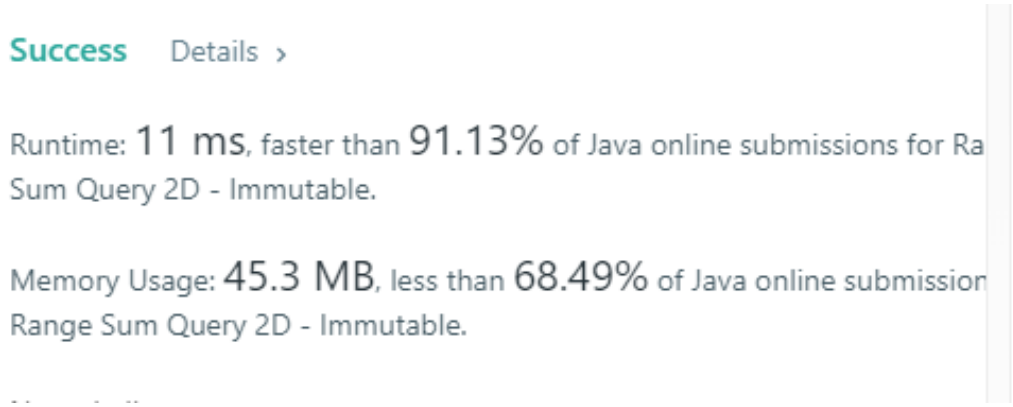
Medium 1053 181 Add to List Share

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).

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

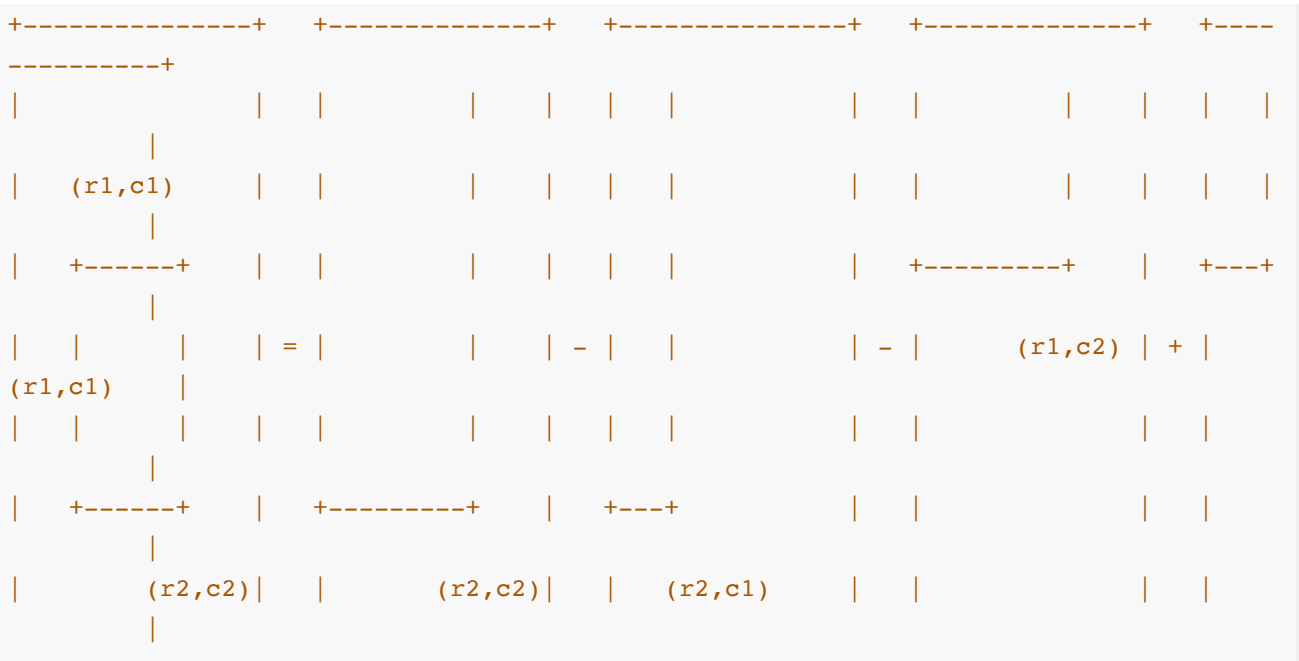
The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

Example:



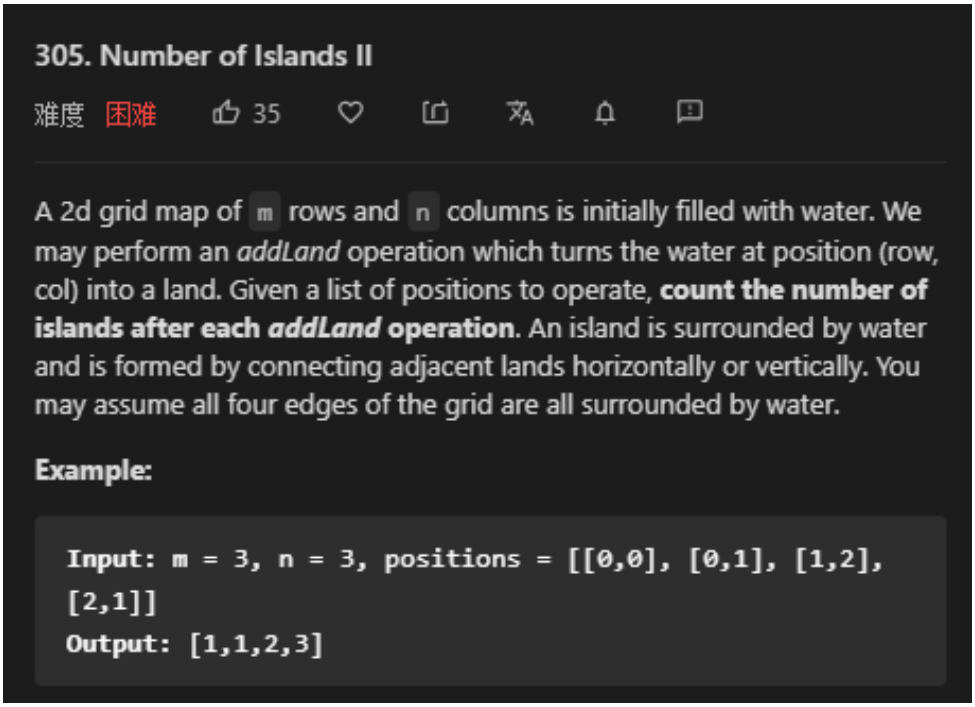

```java
class NumMatrix {
    private int[][] mm;
    public NumMatrix(int[][] matrix) {
        if(matrix.length == 0)      return;
        mm = new int[matrix.length][matrix[0].length + 1];

       //拷贝matrix
        for(int i = 0; i < matrix.length; i++)
            for(int j = 0; j < matrix[0].length; j++)
                mm[i][j+1] = matrix[i][j];

        //叠加
        for(int i = 0; i < matrix.length; i++)
            for(int j = 1; j <= matrix[0].length; j++)
                mm[i][j] += mm[i][j - 1];
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        if(mm[0].length == 1)      return 0;

        int total = 0;
        for(int i = row1; i <= row2; i++)
            total += mm[i][col2+1] - mm[i][col1];
        return total;
    }
}

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

```java
//更加牛逼的方法
/*

```




```
sum[i][j]      =    sums[i-1][j]    +     sums[i][j-1]    -   sums[i-1][j-1]   +

                        matrix[i-1][j-1]
*/
class NumMatrix {
private:
    int row, col;
    vector<vector<int>> sums;
public:
    NumMatrix(vector<vector<int>> &matrix) {
        row = matrix.size();
        col = row>0 ? matrix[0].size() : 0;

        sums = vector<vector<int>>(row+1, vector<int>(col+1, 0));
        for(int i=1; i<=row; i++)
            for(int j=1; j<=col; j++)
                sums[i][j] = matrix[i-1][j-1] + sums[i-1][j] + sums[i][j-1] -
sums[i-1][j-1] ;


    }

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

305 Number of Islands II 典型并查集



305. Number of Islands II

难度 困难 35

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, positions = [[0,0], [0,1], [1,2], [2,1]]
Output: [1,1,2,3]
```

```java
/*
  超时，通过159/162个案例
*/
private int[][] dirctions = {{1, 0},{-1, 0},{0, 1},{0, -1}};
private boolean[][] marked;
public List<Integer> numIslands2(int m, int n, int[][] positions) {
    int[][] grid = new int[m][n];


    List<Integer> res = new ArrayList<>();

    for(int i = 0; i < positions.length; i++)
    {
        grid[positions[i][0]][positions[i][1]] = 1;
        int count = 0;
        marked = new boolean[m][n];
        for(int a = 0; a < m; a++)
            for(int b = 0; b < n; b++)
            {
                if(!marked[a][b] && grid[a][b] == 1)
                {
                    marked[a][b] = true;
                    dfs(grid, a, b);
                    count++;
                }
            }
```

```
        res.add(count);
    }
    return res;
}

private void dfs(int[][] grid, int x, int y)
{
    if(x < 0 || y < 0 || x >= grid.length || y >= grid[0].length
       || marked[x][y] || grid[x][y] != 1)
        return;

    marked[x][y] = true;

    for(int k = 0; k < 4; k++)
    {
        int newX = x + dirctions[k][0];
        int newY = y + dirctions[k][1];

        dfs(grid, newX, newY);
    }
}
```

```
/*
    二刷 规范了并查集的使用

    具体实现细节做了小的改动,
      * count 初始值为 0
*/
public class Solution {
    int m;
    int n;
    public List<Integer> numIslands2(int m, int n, int[][] positions) {
        this.m = m;
        this.n = n;
        List<Integer> res = new ArrayList<>();
        int[][] matrix = new int[m][n];
        HashSet<String> visited = new HashSet<>();
        WeightedUnionFind wuf = new WeightedUnionFind(m * n);
        for(int i = 0; i < positions.length; i++){

          //去重操作
            String symbol = positions[i][0] + "@" + positions[i][1];
            if(visited.contains(symbol)){
                res.add(wuf.getCount());
                continue;
            }
            visited.add(symbol);
```

```
26
27              int[] pos = positions[i];
28              int x = pos[0];
29              int y = pos[1];
30              matrix[x][y] = 1;
31              wuf.count++;
32
33            //如果周围有可以连起来的
34              if(x > 0 && matrix[x - 1][y] == 1){
35                //如果之前已经连接起来了，我就不 --了
36                //如果没有，那么就要 --
37                  if(!wuf.isConnected(getCoor(x - 1, y), getCoor(x, y)))
38                      wuf.count--;
39                  wuf.union(getCoor(x - 1, y), getCoor(x, y));
40              }
41
42            //
43              if(x < m - 1 && matrix[x + 1][y] == 1){
44                  if(!wuf.isConnected(getCoor(x + 1, y), getCoor(x, y)))
45                      wuf.count--;
46                  wuf.union(getCoor(x + 1, y), getCoor(x, y));
47              }
48
49            //
50              if(y > 0 && matrix[x][y - 1] == 1){
51                  if(!wuf.isConnected(getCoor(x, y - 1), getCoor(x, y)))
52                      wuf.count--;
53                  wuf.union(getCoor(x, y - 1), getCoor(x, y));
54              }
55
56            //
57              if(y < n - 1 && matrix[x][y + 1] == 1){
58                  if(!wuf.isConnected(getCoor(x, y + 1), getCoor(x, y)))
59                      wuf.count--;
60                  wuf.union(getCoor(x, y + 1), getCoor(x, y));
61              }
62
63
64              res.add(wuf.getCount());
65          }
66
67          return res;
68      }
69
70    //helper function
71      private int getCoor(int x, int y){
72          return x * n + y;
73      }
74  }
```

```java

class WeightedUnionFind{
    private int[] id;
    public int count;
    private int[] sz;

    public WeightedUnionFind(int N){
        this.count = 0;
        this.id    = new int[N];
        this.sz  = new int[N];
        for(int i = 0; i < N; i++){
            id[i] = i;
            sz[i] = 1;
        }
    }

    public void union(int p, int q){
        int pRoot = find(p);
        int qRoot = find(q);

        if(pRoot == qRoot)
            return;

        if(sz[pRoot] > sz[qRoot]){
            sz[pRoot] += sz[qRoot];
            id[qRoot] = id[pRoot];
        }else{
            sz[qRoot] += sz[pRoot];
            id[pRoot] = id[qRoot];
        }

    }

    private int find(int p){
        while(p != id[p]){
            id[p] = id[id[p]];
            p = id[p];
        }

        return p;
    }

    public boolean isConnected(int p, int q){
        return find(p) == find(q);
    }

    public int getCount(){
        return count;
    }
```

```
124 }
```

306 Additive Number

306. Additive Number

Medium 380 413 Add to List Share

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.

执行结果: 通过 显示详情 >

执行用时: 3 ms , 在所有 Java 提交中击败了 17.29% 的用户

内存消耗: 38.2 MB , 在所有 Java 提交中击败了 11.30% 的用户

炫耀一下:

```java
/*
    二刷，思路采用回溯
*/
public boolean isAdditiveNumber(String num) {
        /*
            1. divide the string into different numbers in the LONG

            2. see if can add ?
                using List to keep track of the last one
            3. using backtrack to re-use the set
        */
        if(num.length() <= 2)
            return false;
        return backtrack(num, new ArrayList<>(), 0);
    }

    private boolean backtrack(String num, List<String> sequence, int start){
```

```
        if(start == num.length())
            return true;


        for(int i = start + 1; i <= num.length(); i++){
            String numStr = num.substring(start, i);
            if(numStr.charAt(0) == '0' && numStr.length() != 1)
                continue;

            if(sequence.size() < 2 ||
                    isAddable(sequence.get(sequence.size() - 1),
sequence.get(sequence.size() - 2), numStr))
                sequence.add(numStr);
            else
                continue;


            if(backtrack(num, sequence, i) && sequence.size() >= 3)
                return true;

            sequence.remove(sequence.size() - 1);


        }

        return false;
    }

    private boolean isAddable(String s1, String s2, String t){
        if(s2.length() > s1.length())
            return isAddable(s2, s1, t);

        //assert s1 > s2
        int[] chars1 = getArray(s1.toCharArray());
        int[] chars2 = getArray(s2.toCharArray());
        int[] chars3 = getArray(t.toCharArray());

        int[] res    = new int[Math.max(chars1.length, chars2.length) + 1];

        int index1 = s1.length() - 1;
        int index2 = s2.length() - 1;
        int index  = res.length - 1;
        while(index1 >= 0 && index2 >= 0){
            res[index--] = chars1[index1--] + chars2[index2--];
        }

        while(index1 >= 0){
            res[index--] = chars1[index1--];
        }
```

```
66
67          for(int j = res.length -1 ; j >= 1; j--){
68              if(res[j] >= 10){
69                  res[j] %= 10;
70                  res[j - 1] += 1;
71              }
72          }
73
74          if(index < 0)
75              index = 0;
76          if(res[index] == 0)
77              index++;
78
79          if(res.length - index != t.length())
80              return false;
81
82          for(int i = 0; i < t.length(); i++){
83              if(chars3[i] != res[i + index])
84                  return false;
85          }
86
87          return true;
88      }
89
90      private int[] getArray(char[] chars){
91          int[] res = new int[chars.length];
92          for(int i = 0; i < chars.length; i++)
93              res[i] = chars[i] - '0';
94          return res;
95      }
96
```

Runtime: 1043 ms, faster than 5.28% of Java online submissions for Additive Number.

Memory Usage: 39.1 MB, less than 30.06% of Java online submission Additive Number.

```
1   /*
2     回溯
3   */
4   class Solution {
5       boolean flag = false;
```

```java
    public boolean isAdditiveNumber(String num) {
        char[] chars = num.toCharArray();
        backtrack(num, new ArrayList<>(), 0);
        return flag;
    }

    private void backtrack(String num, List<String> path, int start)
    {
        if(!flag && start == num.length())
            if(path.size() >= 3)
            {
                for(int i = 2; i < path.size(); i++)
                    if(!isAddable(path.get(i - 2), path.get(i - 1), path.get(i)))
                        return;
                StringBuilder sb = new StringBuilder();
                for(String str : path)
                    sb.append(str);
                flag = sb.toString().equals(num);
            }

        for(int i = start; i < num.length(); i++)
        {
            String frac = num.substring(start, i + 1);
            if(isValidForm(frac))
            {
                path.add(frac);
                backtrack(num, path, i + 1);
                path.remove(path.size() - 1);
            }
        }
    }

    //判断前两个数字，是否可以相加等于第三个数字
    private boolean isAddable(String x, String y, String z)
    {
        char[] charsX = x.toCharArray();
        char[] charsY = y.toCharArray();
        char[] charsZ = z.toCharArray();
        int[] check   = new int[Math.max(charsX.length, charsY.length) + 1];

        int indexX     = charsX.length - 1;
        int indexY     = charsY.length - 1;
        int indexZ     = charsZ.length - 1;
        int indexCheck = check.length - 1;

        while(indexX >= 0 && indexY >= 0)
            check[indexCheck--] = charsX[indexX--] - '0' + charsY[indexY--] - '0';

        while(indexCheck >= 0 && indexX >= 0)
```

```
            check[indexCheck--] = charsX[indexX--] - '0';
        while(indexCheck >= 0 && indexY >= 0)
            check[indexCheck--] = charsY[indexY--] - '0';

        for(int i = check.length - 1; i >= 1; i--)
            if(check[i] >= 10)
            {
                check[i] -= 10;
                check[i-1] += 1;
            }

        StringBuilder original = new StringBuilder();
        StringBuilder che      = new StringBuilder();

        for(char ch : charsZ)
            original.append(ch);
        for(int i = 0; i < check.length; i++)
            if(i == 0 && check[i] == 0)
                continue;
            else
                che.append(check[i]);
        return original.toString().equals(che.toString());
    }

  //判断是否是有效数字格式， 比如"0123"就不是，因为以0开头
    private boolean isValidForm(String s)
    {
        if(s.length() == 0)     return true;
        if(s.charAt(0) == '0' && s.length() > 1)  return false;
        return true;
    }
}
```

307 Range Sum Query -- Mutable

307. Range Sum Query - Mutable

难度 中等 155

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

Constraints:

- The array is only modifiable by the *update* function.
- You may assume the number of calls to *update* and *sumRange* function is distributed evenly.
- `0 <= i <= j <= nums.length - 1`

307. Range Sum Query - Mutable

难度 中等 155

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* (*i* ≤ *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
```

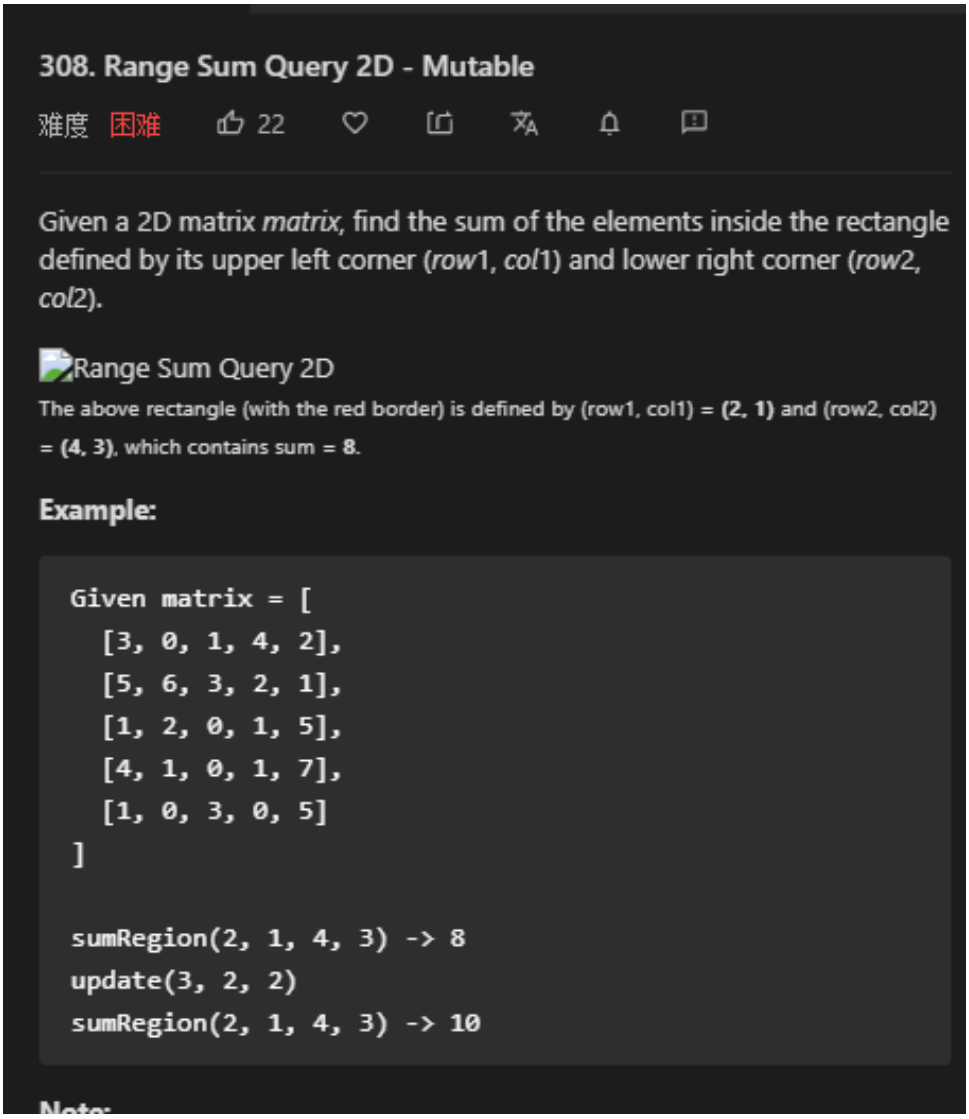
Constraints:

- The array is only modifiable by the *update* function.
- You may assume the number of calls to *update* and *sumRange* function is distributed evenly.
- `0 <= i <= j <= nums.length - 1`

```java
class NumArray {
    private int[] nums;
    private int[] records;

    public NumArray(int[] nums) {
        this.nums = nums;
        records   = new int[nums.length];
        for(int i = 0; i < nums.length; i++)
            records[i] = nums[i];
        getReadyToSum();
    }

    public void update(int i, int val) {
        records[i] = val;
        getReadyToSum();
    }

    public int sumRange(int i, int j) {
        if(i == 0)      return nums[j];
        return nums[j] - nums[i - 1];
    }

```

```
    private void getReadyToSum()
    {
        for(int i = 0; i < records.length; i++)
            nums[i] = records[i];
        for(int i = 1; i < nums.length; i++)
            nums[i] += nums[i - 1];
    }
}

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

308 Range Sum Query 2D -Mutable



308. Range Sum Query 2D - Mutable

难度 困难 22

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).

Range Sum Query 2D

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:


```java
class NumMatrix {
    private int[][] matrix;

    private int[][] sum;
    public NumMatrix(int[][] matrix) {
        this.matrix = matrix;
        int row = matrix.length;
        int col = row > 0 ? matrix[0].length : 0;
        this.sum = new int[row + 1][col + 1];

        updateMatrix();
        getReadyToSumQuery();
    }

    public void update(int row, int col, int val) {
        matrix[row][col] = val;
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        updateMatrix();
        getReadyToSumQuery();

        return sum[row2+1][col2+1] - sum[row2+1][col1] - sum[row1][col2+1] + sum[row1][col1];

    }

    private void updateMatrix()
    {
        for(int i = 0; i < matrix.length; i++)
            for(int j = 0; j < matrix[0].length; j++)
                sum[i + 1][j + 1] = matrix[i][j];
```

```
32      }
33
34      private void getReadyToSumQuery()
35      {
36          for(int i = 1; i < sum.length; i++)
37              for(int j = 1; j < sum[0].length; j++)
38                  sum[i][j]  = sum[i-1][j] + sum[i][j-1] - sum[i-1][j-1] + matrix[i-
    1][j-1];
39
40      }
41  }
42
43  /**
44   * Your NumMatrix object will be instantiated and called as such:
45   * NumMatrix obj = new NumMatrix(matrix);
46   * obj.update(row,col,val);
47   * int param_2 = obj.sumRegion(row1,col1,row2,col2);
48   */
```

309 Best Time to Buy and Sell Stock

309. Best Time to Buy and Sell Stock with Cooldown

Medium 2705 88 Add to List Share

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]
```


```java
public int maxProfit(int[] prices) {
    int days =  prices.length;
    int max  = 0;
    if(days == 0)       return 0;
    //
    int[][] dp = new int[days + 1][2];
    dp[0][0]   = 0;
    dp[0][1]   = Integer.MIN_VALUE;
    dp[1][0]   = 0;
    dp[1][1]   = -prices[0];

    for(int i = 2; i < days + 1; i++)
    {
        dp[i][0] = Math.max(dp[i-1][0], dp[i-1][1] + prices[i - 1]);
        dp[i][1] = Math.max(dp[i-1][1], dp[i-2][0] - prices[i - 1]);

        max = Math.max(max, dp[i][0]);
    }

    return max;
}
```

310 Minimum Height Trees

310. Minimum Height Trees

难度 中等 178

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]
```

执行结果: 通过 显示详情 >

执行用时: 16 ms , 在所有 Java 提交中击败了 49.08% 的用户

内存消耗: 40.5 MB , 在所有 Java 提交中击败了 92.27% 的用户

炫耀一下:

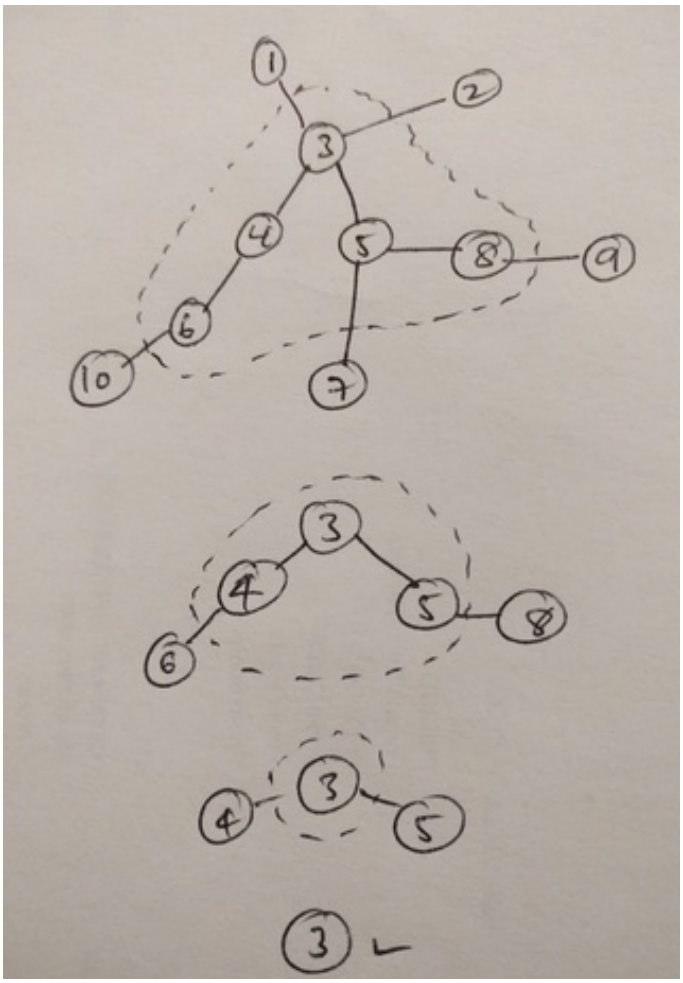
```java
/*
    二刷 03/07/2021
*/
public List<Integer> findMinHeightTrees(int n, int[][] edges) {
        if(n == 1){
            List<Integer>       res = new ArrayList<>();
            res.add(0);
```

```
            return res;
        }

        /*
            1. get the adjacent table for edges

            2. using queue to get all the end point

            3. to remove and to get the size-1 list, this will be next to inspect

            4. if say, we left 2 or 1 nodes, that will be our res
        */
        List<List<Integer>> adj = new ArrayList<List<Integer>>();
        List<Integer>       res = new ArrayList<>();
        Deque<Integer>    queue = new ArrayDeque<>();

        for(int i = 0; i < n; i++)
            adj.add(new ArrayList<>());

        for(int[] edge : edges){
            adj.get(edge[0]).add(edge[1]);
            adj.get(edge[1]).add(edge[0]);
        }


        for(int i = 0; i < n; i++){
            if(adj.get(i).size() == 1)
                queue.addLast(i);
        }


        while(n > 2){
            int size = queue.size();
            for(int i = 0; i < size; i++){
                Integer curNum = queue.pollFirst();
                for(Integer num : adj.get(curNum)){
                    adj.get(num).remove(curNum);

                    if(adj.get(num).size() == 1)
                        queue.addLast(num);
                }

                n--;
            }
        }

        while(!queue.isEmpty()){
            res.add(queue.pollFirst());
        }
```

```
        return res;
    }
```

```
/*
  超时， 49/61 主要思路：通过每一次的广度遍历，拿到高度
  最后做比较
*/
class Solution {
    int minHeight = Integer.MAX_VALUE;
    public List<Integer> findMinHeightTrees(int n, int[][] edges) {
        HashMap<Integer, List<Integer>> map = new HashMap<>();
        //times to expirment
        for(int i = 0; i < n; i++)
        {
            Deque<Integer> queue = new ArrayDeque<>();
            HashSet<Integer> visited = new HashSet<>();
            queue.addLast(i);
            visited.add(i);

            int height = 0;
            while(!queue.isEmpty())
            {
                int size = queue.size();
                for(int j = 0; j < size; j++)
                {
                    int curPoint = queue.removeFirst();
                    for(int[] edge : edges)
                    {
                        if(edge[0] == curPoint)
                        {
                            if(!visited.contains(edge[1]))
                            {
                                queue.addLast(edge[1]);
                                visited.add(edge[1]);
                            }
                        }
                        else if(edge[1] == curPoint)
                            if(!visited.contains(edge[0]))
                            {
                                queue.addLast(edge[0]);
                                visited.add(edge[0]);
                            }
                    }
                }
```

```
                height++;
            }

            if(!map.containsKey(height))
                map.put(height, new ArrayList<>());
            map.get(height).add(i);
        }

        for(Integer height : map.keySet())
            minHeight = Math.min(minHeight, height);

        return map.get(minHeight);
    }
}
```




```
/*
  牛逼方法，下附思路分析

  我们每次都会入队 入度为1的点， 原因是他们不可能为root， 参见上图
  因此比如 上图的情况，第一轮入队 1, 2 7,9,10

  之后把他们排除，然后减少对应点的入度， 比如 1 对应3， 就减少3的入度， 2 对应3 就减少3的入度
```

```
  7 对应5 就减少5的入度

  再进行下一轮的对入度为1的点的排查

  最后只剩1个点，或者两个点， 不可能是剩下3个点
  如果剩下三个点，一定还存在入度为2的点

  实在牛逼
*/
public List<Integer> findMinHeightTrees(int n, int[][] edges) {
    if (n == 1) return Collections.singletonList(0);

    //对入度情况的记录
    List<Set<Integer>> adj = new ArrayList<>(n);
    for (int i = 0; i < n; ++i) adj.add(new HashSet<>());
    for (int[] edge : edges) {
        adj.get(edge[0]).add(edge[1]);
        adj.get(edge[1]).add(edge[0]);
    }

    //拿到叶子节点， 这个变量命名很棒
    List<Integer> leaves = new ArrayList<>();
    for (int i = 0; i < n; ++i)
        if (adj.get(i).size() == 1) leaves.add(i);

    //开始循环
    while (n > 2) {
        n -= leaves.size();
        List<Integer> newLeaves = new ArrayList<>();
        for (int i : leaves) {
            int j = adj.get(i).iterator().next();
            adj.get(j).remove(i);
            if (adj.get(j).size() == 1) newLeaves.add(j);
        }
        leaves = newLeaves;
    }
    return leaves;
}

// Runtime: 53 ms
https://leetcode.com/problems/minimum-height-trees/discuss/76055/Share-some-thoughts
```

311 Sparse Matrix Multiplication

311. Sparse Matrix Multiplication

难度 中等 21

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 |
```

执行用时： 2 ms ，在所有 Java 提交中击败了 72.92% 的用户

内存消耗： 38.8 MB ，在所有 Java 提交中击败了 97.22% 的用户

炫耀一下:

```java
//二刷
class Solution {
    public int[][] multiply(int[][] A, int[][] B) {
        int row = A.length;
        int col = B[0].length;

        int[][] res = new int[row][col];

        HashSet<Integer> ARow = new HashSet<>();
        HashSet<Integer> BCol = new HashSet<>();


        for(int i = 0; i < row; i++){
            boolean zero = true;
```

```
            for(int j = 0; j < A[0].length; j++){
                if(A[i][j] != 0){
                    zero = false;
                    break;
                }
            }

            if(zero){
                ARow.add(i);
            }
        }


        for(int j = 0; j < B[0].length; j++){
            boolean zero = true;
            for(int i = 0; i < B.length; i++){
                if(B[i][j] != 0){
                    zero = false;
                    break;
                }
            }

            if(zero)
                BCol.add(j);
        }



        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(ARow.contains(i) || BCol.contains(j)){
                    res[i][j] = 0;
                    continue;
                }else{
                   int sum = 0;
                    for(int p = 0; p < A[0].length; p++)
                        sum += A[i][p] * B[p][j];

                    res[i][j] = sum;
                }
            }
        }

        return res;
    }
}
```

```java
public int[][] multiply(int[][] A, int[][] B) {
    int rowA = A.length;
    int colB = B[0].length;

    int[][] res = new int[rowA][colB];

    for(int i = 0; i < rowA; i++)
        for(int j = 0; j < colB; j++)
        {
            for(int t = 0; t < A[0].length; t++)
                res[i][j] += A[i][t] * B[t][j];
        }
    return res;
}
```

```java
/*
  非暴力解法

  碰到A的一行为0， 或者B的一列为0
  直接记录下来
  最后再遍历，碰到之前记录过的，就跳过

*/
public int[][] multiply(int[][] A, int[][] B) {
    int[][] res = new int[A.length][B[0].length];
    Set<Integer> aSet = new HashSet<>();
    Set<Integer> bSet = new HashSet<>();
    for(int i = 0;i < A.length;i++){
        boolean flag = true;
        for(int j = 0;j < A[0].length;j++){
            if(A[i][j] != 0){
                flag = false;
                break;
            }
        }
        if(flag) aSet.add(i);
    }

    for(int j = 0;j < B[0].length;j++){
        boolean flag = true;
        for(int i = 0;i < B.length;i++){
            if(B[i][j] != 0){
                flag = false;
                break;
            }
```

```
31          }
32          if(flag) bSet.add(j);
33      }
34
35      for (int row = 0; row < A.length; row++) {
36          if(aSet.contains(row)) continue;
37          for(int col = 0;col < B[0].length;col++){
38              if(bSet.contains(col)) continue;
39              int sum = 0;
40              for(int Brow = 0;Brow < B.length;Brow++){
41                  sum += A[row][Brow] * B[Brow][col];
42              }
43              res[row][col] = sum;
44          }
45      }
46      return res;
47  }
```

312 Burst Ballons

312. Burst Balloons

难度 困难 485

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.

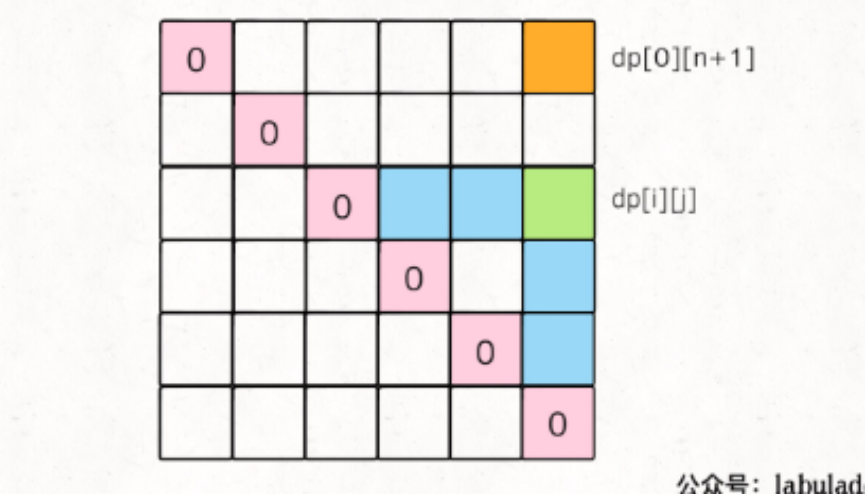
Find the maximum coins you can collect by bursting the balloons wisely.

Note:

- You may imagine `nums[-1] = nums[n] = 1`. They are not real therefore you can not burst them.
- 0 ≤ `n` ≤ 500, 0 ≤ `nums[i]` ≤ 100

Example:

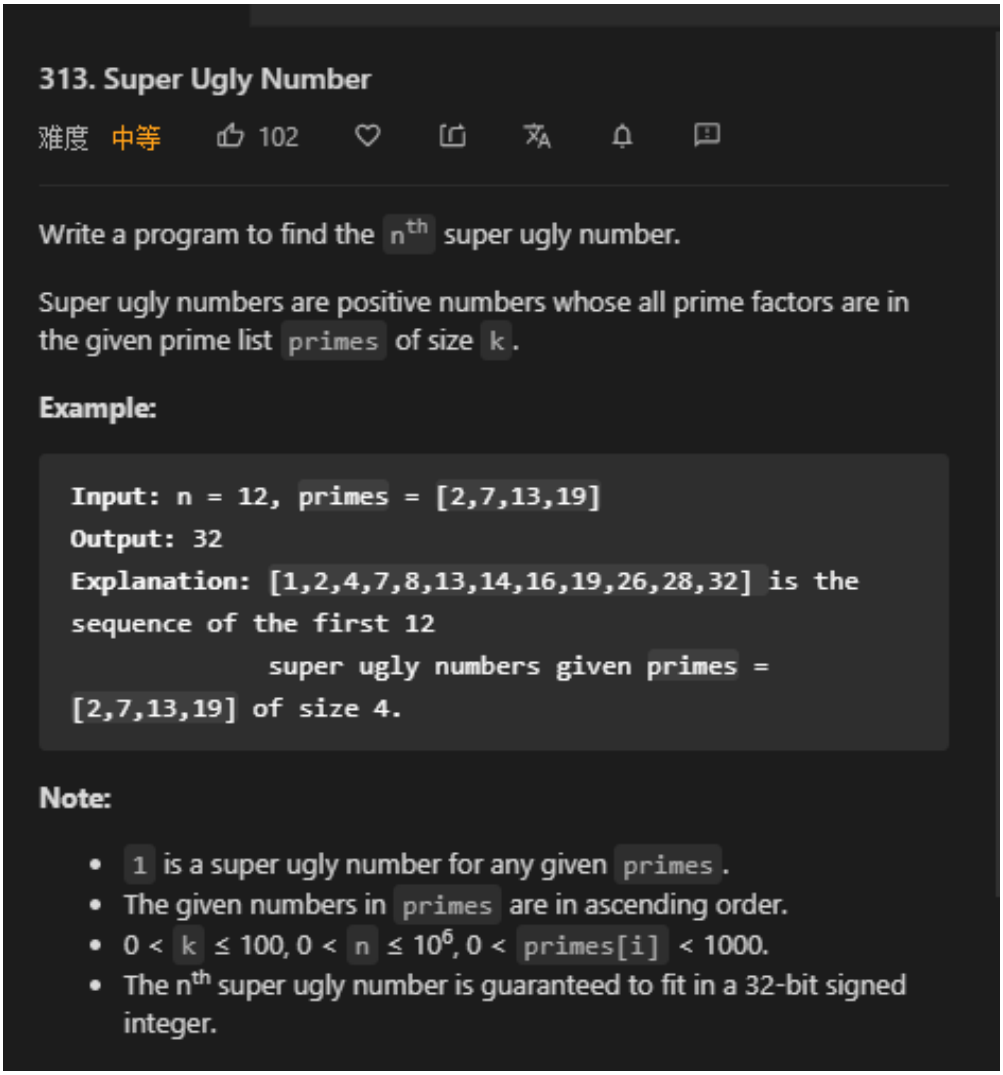
```
Input: [3,1,5,8]
Output: 167
Explanation: nums = [3,1,5,8] --> [3,5,8] -->   [3,8]
-->  [8]  --> []
             coins =  3*1*5      +  3*5*8    +  1*3*8
+ 1*8*1   = 167
```




```
/*
  在 两端加两个虚拟气球， 左边的值为1， 右边的值为1
  定义dp数组含义
  dp[i][j] = x 表示， 戳破气球i j 之间(开区间) 的所有气球，所能得到的最高分数x
  题目想求的就是 dp[0][n + 1];

  base case dp[i][i] = 0; 因为没气球可以戳

  状态转移方程
  k 为 假设最后一个戳破的是k， 那么dp[i][j] 的值应该为
  for(int k = i + 1; k < j; k++)
  {
    dp[i][j] = Math.max(dp[i][j], dp[i][k] + dp[k][j] + nums[k] * nums[i] *
nums[j]);
  }

  最后，拓扑顺序应该是什么样子的

  对于状态的穷举，最重要的一点，就是状态转移所依赖的状态，必须被提前计算出来
  见上图， 要想 dp[i][j] 推出来，我们必须得到dp[i][k] dp[k][j] 也就是所在的行，和列的下面

*/

public int maxCoins(int[] nums) {
    int[] points = new int[nums.length + 2];
    int[][] dp = new int[nums.length + 2][nums.length + 2];
    Arrays.fill(points, 1);
    for(int i = 0; i < nums.length; i++)
        points[i+1] = nums[i];

    for(int i = nums.length + 1; i >= 0; i--)
        for(int j = i + 1; j < nums.length + 2; j++)
            for(int k = i + 1; k < j ; k++)
                dp[i][j] = Math.max(dp[i][j],
                                    dp[i][k] + dp[k][j] + points[k] * points[i] *
points[j]);
```

```
35
36      return dp[0][nums.length + 1];
37  }
```

313 Super Ugly Number 未完成



313. Super Ugly Number

难度 中等 102

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` $\le 100, 0 <$ `n` $\le 10^6, 0 <$ `primes[i]` < 1000.
- The n^{th} super ugly number is guaranteed to fit in a 32-bit signed integer.

```
1   /*
2     source:
3       https://leetcode.com/problems/super-ugly-number/discuss/76291/Java-three-
    methods-23ms-36-ms-58ms(with-heap)-performance-explained
4   */
5   public int nthSuperUglyNumber(int n, int[] primes) {
6           int[] dp = new int[n];
7           int[] idx  = new int[primes.length];
8
9           dp[0] = 1;
```

```
        for(int i = 1; i < n; i++){
            dp[i] = Integer.MAX_VALUE;

          //find next one
            for(int j = 0; j < primes.length; j++) {
                dp[i] = Math.min(dp[i], primes[j] * dp[idx[j]]);
            }

          //avoid duplicates
            for(int j = 0; j < primes.length; j++){
                while(primes[j] * dp[idx[j]] <= dp[i])
                    idx[j]++;
            }
        }

        return ugly[n - 1];
    }

```

Nevsanev ★ 1686 April 19, 2019 12:05 AM 2.9K VIEWS

It is actually like how we merge k sorted list:

```
ugly number                    k sorted list
    1                         2     7    13   19     1 * [2,7,13,19]
    |                         |     |    |    |
    2                         4     14   26   38     2 * [2,7,13,19]
    |                         |     |    |    |
    4                         8     28   52   76     4 * [2,7,13,19]
    |                         |     |    |    |
    7                         14    49   91   133    7 * [2,7,13,19]
    |                         |     |    |    |
    8                         16    56   ...   ...   8 * [2,7,13,19]
    |                         |     |    |     |
    .                         .     .     .    .
    .                         .     .     .    .
    .                         .     .     .    .
```

```
/*
  source:
    https://leetcode.com/problems/super-ugly-number/discuss/277313/My-view-of-this-question-hope-it-can-help-you-understand!!!
*/
class Solution {
    public int nthSuperUglyNumber(int n, int[] primes) {
```

```
        PriorityQueue<int[]> queue=new PriorityQueue<>((a,b)->(a[0]-b[0]));
        for (int i=0;i<primes.length;i++)
            queue.offer(new int[]{primes[i], primes[i], 0});

        int[] nums=new int[n+1];
        nums[0]=1;

        int i=1;
        while (i<n){
            int[] entry=queue.poll();

          /*
            num -> the value of the node
            prime -> which sorted list this node is in
            index -> how fat we have gone?
          */
            int num=entry[0], prime=entry[1], index=entry[2];
            // remove duplicate
            if (num!=nums[i-1]){
                nums[i]=num;
                i++;
            }

            queue.offer(new int[]{prime*nums[index+1], prime, index+1});
        }
        return nums[n-1];
    }
}

时间复杂度 O(NK)
```

314 Binary Tree Vertical Order Traversal

314. Binary Tree Vertical Order Traversal

难度 中等 44

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]
```

执行用时： 5 ms ，在所有 Java 提交中击败了 57.67% 的用户

内存消耗： 38.8 MB ，在所有 Java 提交中击败了 34.92% 的用户

炫耀一下:

```java
/*
  二刷
*/
class Solution {
    List<List<Integer>> res = new ArrayList<>();
    public List<List<Integer>> verticalOrder(TreeNode root) {
        if(root == null)
            return res;

        TreeMap<Integer, List<Integer>> map = new TreeMap<>();
        Deque<MyNode> queue = new ArrayDeque<>();
        queue.add(new MyNode(0, root));
        while(!queue.isEmpty()){
```

```
            int size = queue.size();
            for(int i = 0; i < size; i++){
                MyNode curNode = queue.pollFirst();
                map.putIfAbsent(curNode.dis, new ArrayList<>());
                map.get(curNode.dis).add(curNode.node.val);

                if(curNode.node.left != null){
                    queue.addLast(new MyNode(curNode.dis - 1, curNode.node.left));
                }
                if(curNode.node.right != null){
                    queue.addLast(new MyNode(curNode.dis + 1, curNode.node.right));
                }
            }
        }

        for(Integer num : map.keySet()){
            res.add(map.get(num));
        }

        return res;
    }
}

class MyNode{
    public int dis;
    public TreeNode node;

    public MyNode(int dis, TreeNode node){
        this.dis  = dis;
        this.node = node;
    }
}
```

```
/*
  这个题很有意思，主要是关于给每个Root上ID
  水平遍历的方式，拿到垂直的结果
*/
public List<List<Integer>> verticalOrder(TreeNode root) {
    if(root == null)        return new ArrayList<>();

    //to find ID by TreeNode
    HashMap<TreeNode, Integer> seq = new HashMap<>();

    //to find ID-Container by ID
    TreeMap<Integer, List<Integer>> map = new TreeMap<>();
```

```
13
14      Deque<TreeNode> queue = new ArrayDeque<>();
15      queue.offer(root);
16      seq.put(root, 0);
17      map.put(0, new ArrayList<>());
18      map.get(0).add(root.val);
19
20      while(!queue.isEmpty())
21      {
22          TreeNode cur = queue.removeFirst();
23          int sequence = seq.get(cur);
24
25          if(cur.left != null)
26          {
27              queue.add(cur.left);
28              seq.put(cur.left, sequence - 1);
29
30              if(!map.containsKey(sequence - 1))
31                  map.put(sequence - 1, new ArrayList<>());
32              map.get(sequence - 1).add(cur.left.val);
33          }
34
35          if(cur.right != null)
36          {
37              queue.add(cur.right);
38              seq.put(cur.right, sequence + 1);
39
40              if(!map.containsKey(sequence + 1))
41                  map.put(sequence + 1, new ArrayList<>());
42              map.get(sequence + 1).add(cur.right.val);
43
44          }
45      }
46
47      List<List<Integer>> res = new ArrayList<>();
48
49      for(Integer i : map.keySet())
50          res.add(map.get(i));
51      return res;
52  }
```

```
1   /*
2     代码写的相当优雅
3   */
4   public:
```

```
    vector<vector<int>> verticalOrder(TreeNode* root) {
        //map本身是按key排好序的，所以输出的结果是从左到右的
        map<int,vector<int>> m;
        vector<vector<int>> res;
        if(root==NULL) return res;

        queue<pair<TreeNode*,int>> q;
        q.push(make_pair(root,0));

        //水平层次遍历得到每个节点的竖直方向的层次
        while(!q.empty()){
            TreeNode* t=q.front().first;
            int level=q.front().second;
            m[level].push_back(t->val);
            q.pop();
            if(t->left!=NULL)
                q.push(make_pair(t->left,level-1));
            if(t->right!=NULL)
                q.push(make_pair(t->right,level+1));
        }
        for(auto it=m.begin();it!=m.end();it++)
            res.push_back(it->second);
        return res;
    }
};

作者：ae2a
链接：https://leetcode-cn.com/problems/binary-tree-vertical-order-traversal/solution/jing-jing-de-bi-ji-314-by-ae2a/
```

315 Count of Smaller Numbers After Self 前缀和 FenwickTree

```
//暴力解，  通过15/16
public List<Integer> countSmaller(int[] nums) {
    List<Integer> counts = new ArrayList<>();

    for(int i = 0;  i< nums.length; i++)
    {
        int count = 0;
        for(int j = i + 1; j < nums.length; j++)
            if(nums[j] < nums[i])
```

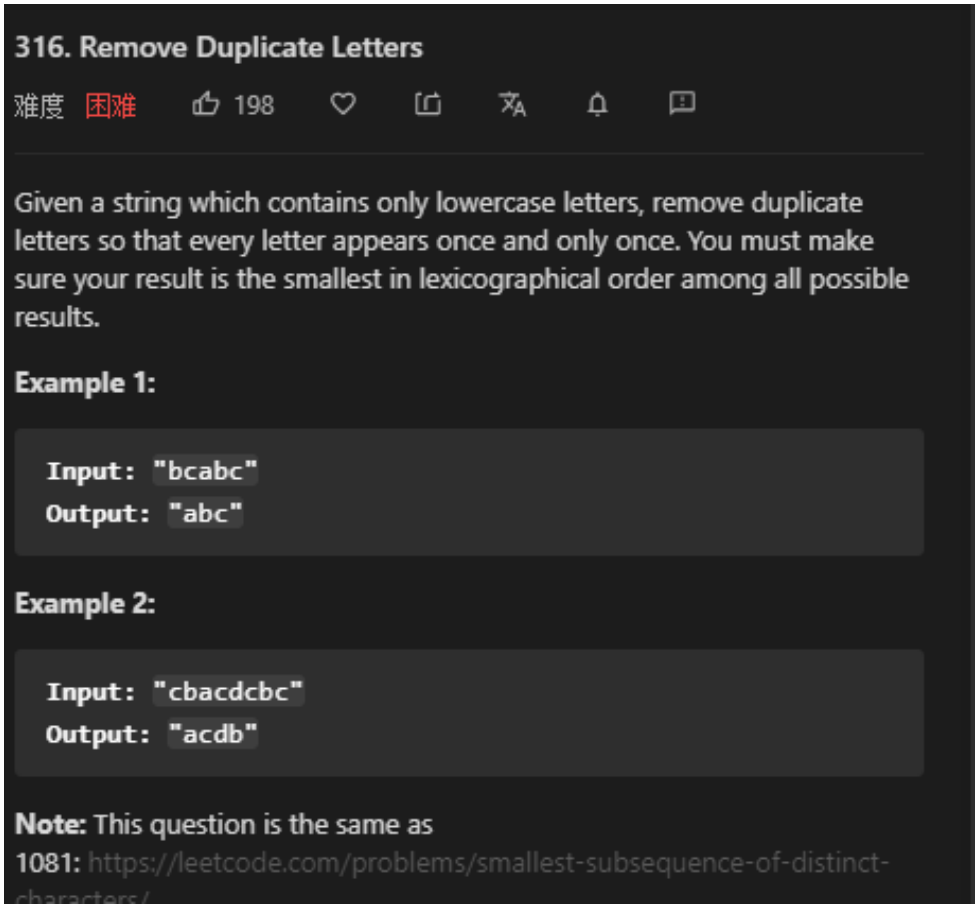
```
                count++;

            counts.add(count);
        }

        return counts;
    }
```

采用 前缀和 FenwickTree

```
/*
    本题核心思想
      将每个数字 rank
      出现的频率作为前缀和的权重

      然后采用 FenwickTree 计算前缀和

*/
class Solution {
    private static int lowbit(int x){
        return x & (-x);
    }

    class FenwickTree{
        private int[] sums;

        public FenwickTree(int n){
            sums = new int[n + 1];
        }

        public void update(int i, int delta){
            while(i < sums.length){
                sums[i] += delta;
                i += lowbit(i);
            }
        }

        public int query(int i){
            int sum = 0;
            while(i > 0){
                sum += sums[i];
                i -= lowbit(i);
            }

            return sum;
        }
```

```
    }

    public List<Integer> countSmaller(int[] nums) {
        int[] sorted = Arrays.copyOf(nums, nums.length);

        Arrays.sort(sorted);
        Map<Integer, Integer> ranks = new HashMap<>();
        int rank = 0;
        for(int i = 0; i < sorted.length; i++){
            if(i == 0 || sorted[i] != sorted[i - 1])
                ranks.put(sorted[i], ++rank);
        }

        FenwickTree tree  = new FenwickTree(ranks.size());
        List<Integer> res = new ArrayList<>();

        for(int i = nums.length - 1; i >= 0; i--){
            int sum = tree.query(ranks.get(nums[i]) - 1);
            res.add(tree.query(ranks.get(nums[i]) - 1));
            tree.update(ranks.get(nums[i]), 1);
        }

        Collections.reverse(res);
        return res;
    }
}

```

316 Remove Duplicate Letter



316. Remove Duplicate Letters

难度 困难 198

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appears 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"
```

Note: This question is the same as 1081: https://leetcode.com/problems/smallest-subsequence-of-distinct-characters/

```java
//超出时间限制 268/286
private HashSet<String> res;
private HashSet<Character> set;
private String str;
public String removeDuplicateLetters(String s) {
    res = new HashSet<>();
    set = new HashSet<>();
    str = s;
    for(int i = 0; i < s.length(); i++)
        set.add(s.charAt(i));

    backtrack(s.toCharArray(), new ArrayList<>(), 0);

    return str;

}

private void backtrack(char[] chars, List<Character> path, int start)
{
    if(path.size() == set.size())
    {
        StringBuilder sb = new StringBuilder();
        for(Character ch : path)
            sb.append(ch);

        String cc = sb.toString();
```

```
        if(str.length() > cc.length())
            str = cc;
        else
            if(isSmall(str, cc))
                str = cc;
        return;
    }

    for(int i = start; i < chars.length; i++)
    {
        if(!path.contains(chars[i]))
        {
            path.add(chars[i]);

            backtrack(chars, path, i + 1);

            path.remove(path.size() - 1);
        }
    }
}

private boolean isSmall(String s, String cc)
{
    for(int i = 0; i < s.length(); i++)
        if(s.charAt(i) < cc.charAt(i))
            return false;
        else if(s.charAt(i) == cc.charAt(i))
            continue;
        else
          return true;
    return true;
}
```

```
/*
1、遍历字符串里的字符，如果读到的字符的 ASCII 值是升序，依次存到一个栈中；
2、如果读到的字符在栈中已经存在，这个字符我们不需要；
3、如果读到的 ASCII 值比栈顶元素严格小，看看栈顶元素在后面是否还会出现，如果还会出现，则舍弃栈顶元素，而选择后出现的那个字符，这样得到的字典序更小。

*/
public String removeDuplicateLetters(String s) {
    int[] lastIndex = new int[26];
    for(int i = 0; i < s.length(); i++)
        lastIndex[s.charAt(i) - 'a'] = i;

    boolean[] exist = new boolean[26];
    Deque<Character> stack = new ArrayDeque<>();
```

```
    stack.addLast('a');

    for(int i = 0; i < s.length(); i++)
    {
        char ch = s.charAt(i);
        if(exist[ch - 'a'])       continue;
        while(stack.peekLast() > ch && lastIndex[stack.peekLast() - 'a'] > i)
        {
            char c = stack.removeLast();
            exist[c - 'a'] = false;
        }

        stack.addLast(ch);
        exist[ch - 'a'] = true;
    }

    StringBuilder sb = new StringBuilder();
    stack.removeFirst();
    int size = stack.size();
    for(int i = 0; i < size; i++)
        sb.append(stack.removeFirst());
    return sb.toString();
}

作者：liweiwei1419
    链接：https://leetcode-cn.com/problems/remove-duplicate-letters/solution/zhan-by-liweiwei1419/
```

317 Shortest Distance from All Buildings 未完成

317. Shortest Distance from All Buildings

难度 困难 35

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
```

```java
public int shortestDistance(int[][] grid) {
    int m = grid.length;
    int n = grid[0].length;
    int res = Integer.MAX_VALUE;
    int val = 0;

    int[][] sum     = new int[m][n];
    int[][] dist    = new int[m][n];
    copy(sum, dist, grid);

    int[][] dir      = new int[][]{{1, 0},{0, 1},{0, -1},{-1, 0}};
    Deque<int[]> queue = new ArrayDeque<>();

    for(int i = 0; i < m; i++)
        for(int j = 0; j < n; j++)
        {
            if(grid[i][j] != 1)
                continue;
            res = Integer.MAX_VALUE;
            queue.add(new int[]{i, j});

            while(!queue.isEmpty())
            {
                int[] point = queue.removeFirst();
                int a = point[0]; int b = point[1];
                for(int k = 0; k < 4; k++)
```

```
                {
                    int newX = a + dir[k][0];
                    int newY = b + dir[k][1];
                    if(newX >= 0 & newY >= 0 && newX < m && newY < n
                       && grid[newX][newY] == val)
                    {
                        --grid[newX][newY]; //这里就不用再开访问数组了，直接减1就ok
                        dist[newX][newY] = dist[a][b] + 1;
                        sum[newX][newY]  += dist[newX][newY] - 1;
                        queue.add(new int[]{newX, newY});
                        res = Math.min(res, sum[newX][newY]);
                    }
                }

            }
            --val;
        }
    return res == Integer.MAX_VALUE ? -1 : res;
}

private void copy(int[][] sum, int[][] dist, int[][] grid)
{
    for(int i = 0; i < grid.length; i++)
        for(int j = 0; j < grid[0].length; j++)
        {
            sum[i][j] = grid[i][j];
            dist[i][j] = grid[i][j];
        }
}
```

```
public:
int shortestDistance(vector<vector<int>>& grid) {
    //从每个建筑物出发去找空地，空地对每个建筑物的距离累积
    vector<vector<int>> dir = {{1,0},{0,1},{0,-1},{-1,0}};
    vector<vector<int>> build;
    int i, j, k, x, y, mindis;
    int m = grid.size(), n = grid[0].size();
    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
            if(grid[i][j]==1)
                build.push_back({i,j});
    vector<vector<int>> dis(m, vector<int>(n, 0));
    //记录单次遍历1个房屋到空地的距离
    vector<vector<int>> totaldis(m, vector<int>(n, 0));
    //记录所有房屋到空地的距离
    int emptyPlace = 0;//空地的标记数字
    for(auto& pos : build)//遍历房子
```

```
    {
        queue<vector<int>> q;
        // vector<vector<bool>> visited(m, vector<bool>(n,false));
        q.push({pos[0], pos[1]});//x,y
        mindis = INT_MAX;
        while(!q.empty())
        {
            x = q.front()[0];
            y = q.front()[1];
            q.pop();
            for(k = 0; k < 4; ++k)
            {
                i = x + dir[k][0];
                j = y + dir[k][1];
                if(i>=0 && i<m && j>=0 && j<n &&
                   grid[i][j] == emptyPlace)//空地
                {
                    dis[i][j] = dis[x][y]+1;//该房子到ij空地的距离
                    totaldis[i][j] += dis[i][j];//之前所有房子到这的距离和
                    mindis = min(mindis, totaldis[i][j]);
                    // visited[i][j] = true;
                    grid[i][j]--;//从lx_lx大佬那学的，不用开访问数组了
                    q.push({i,j});
                }
            }
        }
        if(mindis == INT_MAX)
            return -1;//该房屋不能到达任何空地
        emptyPlace--;//更新空地的标记，免开访问标记数组
    }
    return mindis==INT_MAX ? -1 : mindis;
}
```

318 Maximum Product of Word Lengths

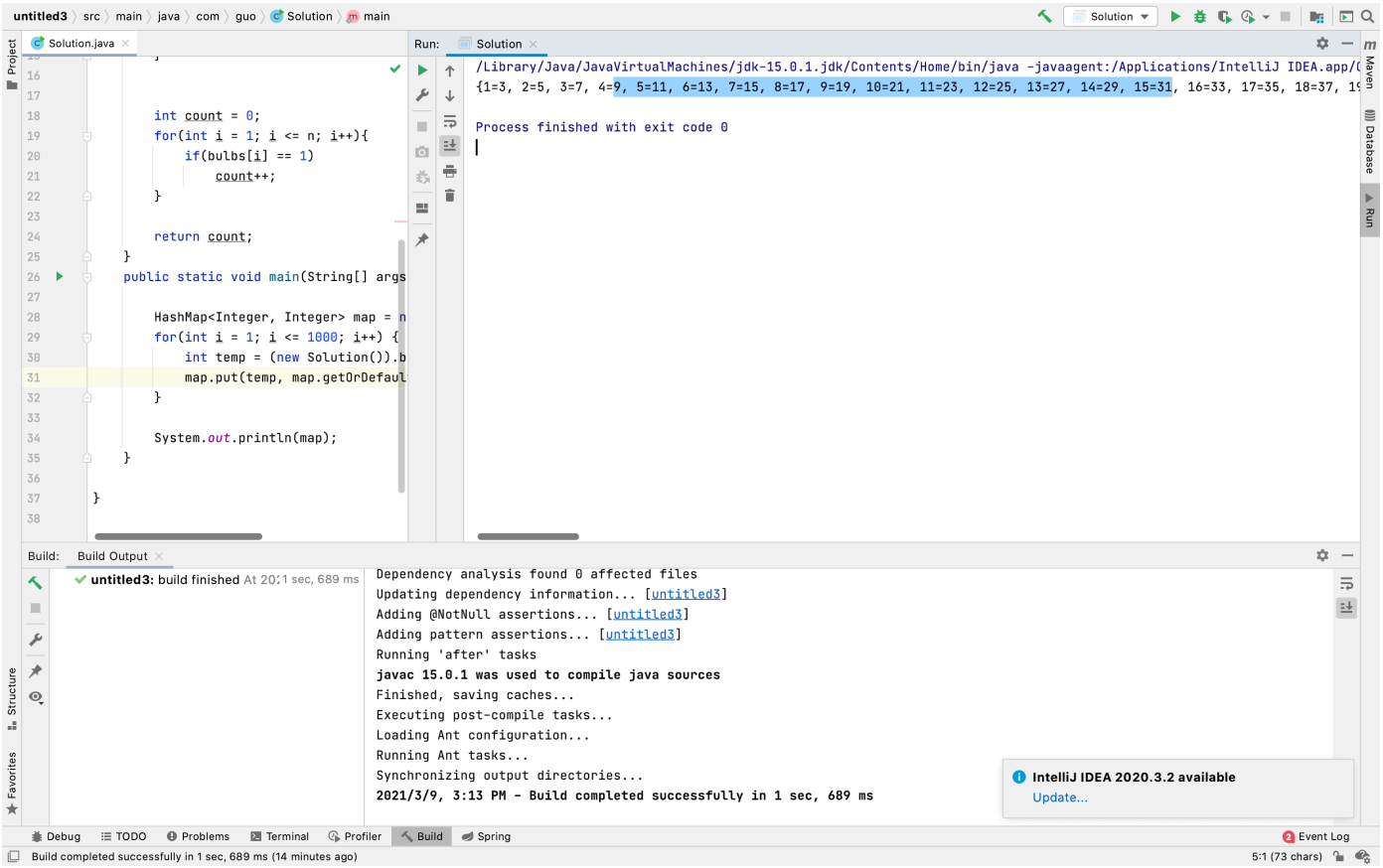

```
public int maxProduct(String[] words) {
```

```
    int res = 0;
    List<HashSet<Character>> list = new ArrayList<>();
    for(int i = 0; i < words.length; i++)
    {
        list.add(new HashSet<>());
        for(int j = 0; j < words[i].length(); j++)
            list.get(i).add(words[i].charAt(j));
    }

    for(int i = 0; i < words.length; i++)
        for(int j = i + 1; j < words.length; j++)
        {
            boolean dup = false;
            for(Character ch : list.get(i))
                if(list.get(j).contains(ch))
                {
                    dup = true;
                    break;
                }
            if(!dup)
                res = Math.max(res, words[i].length() * words[j].length());
        }


    return res;
}
```

```
//更加简单的方法
/*
  这个相当于给每一个单词，根据他们字母的特性，对应生成一个ID
  value[i] |= 1 << (tmp.charAt(j) - 'a');
  比如
    a -> 1
    b -> 10
    c -> 100
    ab -> 11
    az -> 10000000000000000000000001  //一共 26位
*/
public int maxProduct(String[] words) {
    int[] values = new int[words.length];

    for(int i = 0; i < words.length; i++)
        for(int j = 0; j < words[i].length(); j++)
            values[i] |= 1 << (words[i].charAt(j) - 'a');

    int max = 0;
    for(int i = 0; i < words.length; i++)
```

```
        for(int j = i + 1; j < words.length; j++)
            if((values[i] & values[j]) == 0)
                max = Math.max(max, words[i].length() * words[j].length());
    return max;
}
```

319 Bulb Switcher




```
    //二刷，找规律得到灵感，
public int bulbSwitch(int n) {
        if(n == 0)
            return 0;
        else if(n <= 3)
            return 1;

```

```
        int left = 1, right = n;

        while(true){
            long mid = left + (right - left) / 2;

            if(mid * mid <= n && n <= mid * mid + 2 * mid)
                return (int)mid;
            else if(mid * mid > n)
                right = (int)mid - 1;
            else
                left = (int)mid + 1;
        }

    }
```


```
public int bulbSwitch(int n) {
    return (int)Math.sqrt((double)n);
}
```

320 Generalized Abbreviation

提交时间	提交结果	运行时间	内存消耗
12 分钟前	通过	19 ms	44.9 MB

```
private List<String> res;
public List<String> generateAbbreviations(String word) {
    res = new ArrayList<>();

```

```
    char[] chars = word.toCharArray();
    backtrack(chars, 0);
    res.add(word);
    return res;
}

private void backtrack(char[] chars, int start)
{
    if(start >= chars.length)       return;

    for(int i = start; i < chars.length; i++)
    {
        char ch = chars[i];
        chars[i] = '@';
        res.add(toStr(chars));

        backtrack(chars, i + 1);

        chars[i] = ch;
    }
}

private String toStr(char[] chars)
{
    StringBuilder sb = new StringBuilder();
    for(int i = 0; i < chars.length;)
    {
        int j = i;
        int count = 0;
        if(chars[j] == '@')
        {
            while(j < chars.length && chars[j] == '@')
                j++;
            sb.append(j - i);
            i = j;
        }
        else
        {
            sb.append(chars[j]);
            i++;
        }
    }
    return sb.toString();
}
```

321 Create Maximum Number

321. Create Maximum Number

Hard 724 232 Add to List Share

Given two arrays of length `m` and `n` with digits `0-9` representing two numbers. Create the maximum number of length `k <= 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]
```

执行用时： 34 ms ，在所有 Java 提交中击败了 13.71% 的用户

内存消耗： 39 MB ，在所有 Java 提交中击败了 77.92% 的用户

炫耀一下:

```java
/*
    二刷
*/
public int[] maxNumber(int[] nums1, int[] nums2, int k) {
        int len1 = nums1.length;
        int len2 = nums2.length;

        if(len1 < len2)
            return maxNumber(nums2, nums1, k);

        int[] candidate = null;
        //assume nums1.length > nums2.length
        for(int i = 0; i <=  k; i++) {
            if(i > nums1.length || k - i > nums2.length)
                continue;
            int[] max1 = getMax(nums1, i);
            int[] max2 = getMax(nums2, k - i);

            int[] combination = getMaxArray(max1, max2);
            if(greater(combination, 0 , candidate, 0))
                candidate = combination;
```

```
        }

        return candidate;
    }

    // A > B ?
    private boolean greater(int[] A, int index1, int[] B, int index2){
        if(B == null)
            return true;

        while(index1 < A.length && index2 < B.length && A[index1] == B[index2]){
            index1++;
            index2++;
        }

        return index2 == B.length || (index1 < A.length && A[index1] > B[index2]);
    }

    private int[] getMaxArray(int[] nums1, int[] nums2){
        int index1 = 0, index2 = 0;
        int len1 = nums1.length, len2 = nums2.length;
        int[] res = new int[len1 + len2];
        int index = 0;
        while(index1 < len1 || index2 < len2){
            if(index1 == len1)
                res[index++] = nums2[index2++];
            else if(index2 == len2)
                res[index++] = nums1[index1++];
            else if(nums1[index1] > nums2[index2])
                res[index++] = nums1[index1++];
            else if(greater(nums1, index1, nums2, index2))
                res[index++] = nums1[index1++];
            else
               res[index++] = nums2[index2++];
        }

        return res;
    }
//[7, 3, 8, 2, 5, 6, 4, 4, 0, 6, 5, 7, 6, 2, 0]
//[7,3,8,2,5,6,4,4,0,6,5,7,6,2,0]
    private int[] getMax(int[] nums, int num){
        int[] res = new int[num];

        Deque<Integer> stack = new ArrayDeque<>();

        stack.addLast(Integer.MAX_VALUE);
        int index = 0;
        while(index < nums.length){
            while(index < nums.length && stack.peekLast() > nums[index]){
```

```
                stack.addLast(nums[index++]);
            }

            if(index == nums.length)
                break;

            int curNum = nums[index];
            while(stack.size() - 2 + nums.length - index >=  num &&
stack.peekLast() < curNum){
                stack.removeLast();
            }
            stack.addLast(curNum);
            index++;
        }

        index = 0;
        stack.removeFirst();
        for(int i = 0; i < num; i++)
            res[i] = stack.removeFirst();
        return res;
    }


```

```
public int[] maxNumber(int[] nums1, int[] nums2, int k) {
    int n = nums1.length;
    int m = nums2.length;
    int[] ans = new int[k];
    for (int i = Math.max(0, k - m); i <= k && i <= n; ++i)
    {
        int[] candidate = merge(maxArray(nums1, i), maxArray(nums2, k - i), k);
        if (greater(candidate, 0, ans, 0))
            ans = candidate;
    }
    return ans;
}
private int[] merge(int[] nums1, int[] nums2, int k) {
    int[] ans = new int[k];
    for (int i = 0, j = 0, r = 0; r < k; ++r)
        ans[r] = greater(nums1, i, nums2, j) ? nums1[i++] : nums2[j++];
    return ans;
}
public boolean greater(int[] nums1, int i, int[] nums2, int j) {
    while (i < nums1.length && j < nums2.length && nums1[i] == nums2[j])
    {
```

```
        i++;
        j++;
    }
    return j == nums2.length || (i < nums1.length && nums1[i] > nums2[j]);
}
private int[] maxArray(int[] nums, int len)
{
    Stack<Integer> stack = new Stack<Integer>();
    for (int i = 0; i < nums.length; i++)
    {
        while (stack.size() + nums.length - i > len && !stack.empty() &&
stack.peek() < nums[i])
            stack.pop();

        if (stack.size() < len)
            stack.push(nums[i]);

    }
    int[] result = new int[len];
    for (int i = len - 1; i >= 0; i--) {
        result[i] = stack.pop();
    }
    return result;
}
```

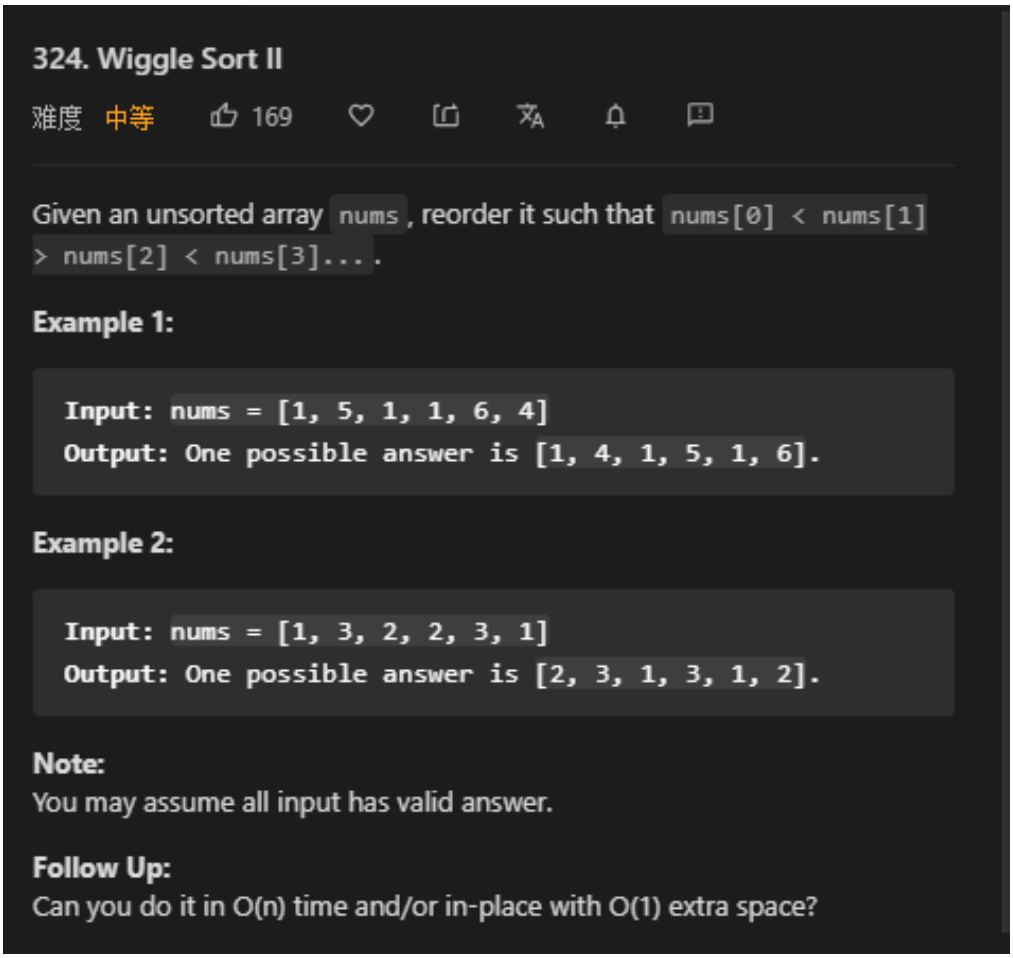
323 Number of Connected Componenets in an Undirected


```
class WeightedUnionFind{
    private int[] id;
    private int[] size;
    private int count;

    public WeightedUnionFind(int n){
        id   = new int[n];
```

```
        size = new int[n];
        for(int i = 0; i < n; i++)
            id[i] = i;
        Arrays.fill(size, 1);
        count = n;
    }

    private boolean connected(int p, int q)
    {   return find(p) == find(q);  }

    private int find(int p)
    {
        while(id[p] != p){
            id[p] = id[id[p]];
            p = id[p];

        }
        return p;
    }

    private void union(int p, int q)
    {
        int pRoot = find(p);
        int qRoot = find(q);

        if(pRoot == qRoot)      return;

        if(size(pRoot) > size(qRoot))
        {
            id[qRoot] = id[pRoot];
            size[pRoot] += size[qRoot];
        }
        else
        {
            id[pRoot] = id[qRoot];
            size[qRoot] += size[pRoot];
        }
        count--;
    }

    private int count()
    {   return count;}

    private int size(int p)
    {   return size[p]; }
}
public int countComponents(int n, int[][] edges) {
    WeightedUnionFind wuf = new WeightedUnionFind(n);
    for(int[] edge : edges)
```

```
57          wuf.union(edge[0], edge[1]);
58
59
60      return wuf.count();
61  }
```

324 Wiggle Sort II



324. Wiggle Sort II

难度 中等 169

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?

```
1   /*
2       注意这个必须倒序插入
3
4       比如 [4,5,5,6] 这个案例
5       否则就会有问题
6       不能顺序插入
7
8       因为 如果我们分成 两个数组
9       A 【4, 5】
10      B  【5, 6】
11
12      在顺序划分的时候， 会发现 5 会插入错的位置
13      而为了避免这个问题，我们会采用倒叙
14      也就是将相同的元素R In A and B to
```

```
    R弄到A的前面， R弄到B的的后面
    就可以尽可能避免这个问题
*/
public void wiggleSort(int[] nums) {
        int[] helper = nums.clone();

        Arrays.sort(helper);
        int len = nums.length;

        int firstHalf = (nums.length + 1) / 2 - 1;
        int secondHalf = (nums.length - 1);
        int flag = firstHalf;

        int index = 0;
        while(secondHalf != flag && firstHalf >= 0){
            nums[index++] = helper[firstHalf--];
            nums[index++] = helper[secondHalf--];
        }

        while(secondHalf != flag)
            nums[index++] = helper[secondHalf--];
        while(firstHalf >= 0)
            nums[index++] = helper[firstHalf--];


    }
```

```
/*
郭郭：
  整体思路，本质上仍然和第一种思路类似
  只是，首先通过 快速选择 O(N) 的算法，拿到该数组的中位数
  之后， 将数组通过O(N) 遍历，分割成三部分
    小于中位数的
    等于中位数的
    大于中位数的

  最后，按照倒序插入

  其中
    快速选择在最坏情况下是 O(N ^ 2) 时间复杂度
    平均状态下是 O(N)
*/
public void wiggleSort(int[] nums) {
        int len      = nums.length;

```

```
        int mid = getMiddleElement(nums);

        int i = 0, j = 0, k = nums.length - 1;

        while(j < k){
            if(nums[j] == mid)
                j++;
            else if(nums[j] > mid)
                exch(nums, j, k--);
            else
                exch(nums, i++, j++);
        }

        int divideIndex = len / 2;
        if(len % 2 == 1)
            divideIndex++;

        int[] firstHalf = Arrays.copyOfRange(nums, 0, divideIndex);
        int[] secondHalf = Arrays.copyOfRange(nums, divideIndex, len);

        for(int m = 0; m < firstHalf.length; m++)
            nums[2 * m] = firstHalf[firstHalf.length - 1 - m];
        for(int n = 0; n < secondHalf.length; n++)
            nums[2 * n + 1] = secondHalf[secondHalf.length - 1 - n];
    }

    private int getMiddleElement(int[] nums){
        int target = nums.length / 2;

        int lo = 0, hi = nums.length - 1;
        while(true){
            int onePartition = partition(nums, lo, hi);

            if(onePartition == target)
                return nums[target];
            else if(onePartition > target)
                hi = onePartition - 1;
            else
                lo = onePartition + 1;
        }
    }

    private int partition(int[] nums, int lo, int hi){
        if(hi == lo)
            return lo;

        int i = lo, j = hi + 1;
        int cmp = nums[lo];
        while(true){
```

```
68              while(nums[++i] < cmp){
69                  if(i == hi)
70                      break;
71              }
72
73              while(nums[--j] > cmp){
74                  if(j == lo)
75                      break;
76              }
77
78              if(i >= j)
79                  break;
80              exch(nums, i, j);
81          }
82
83          exch(nums, j, lo);
84          return j;
85      }
86
87      private void exch(int[] nums, int i, int j){
88          int temp = nums[i];
89          nums[i] = nums[j];
90          nums[j] = temp;
91      }
92
```

325 Maximum Size SubArray Sum Equals K

```java
public int maxSubArrayLen(int[] nums, int k) {
    for(int i = 1; i < nums.length; i++)
        nums[i] += nums[i-1];

    int maxLen = 0;
    for(int i = 0; i < nums.length; i++)
        for(int j = i; j < nums.length; j++)
        {
            if(i == 0)
            {
                if(nums[j] == k)
                    maxLen = Math.max(maxLen, j + 1);
            }
            else
                if(nums[j] - nums[i - 1] == k)
                    maxLen = Math.max(maxLen, j - i + 1);
        }
    return maxLen;
}
```

```java
/*
  大致思路类似于
  两数和
  总结下来前缀和 + HashMap
*/
public int maxSubArrayLen(int[] nums, int k) {
        int len = nums.length;
        int max = 0;

        // 求前缀和，并利用 Map<前缀和，对应索引> 储存
        int[] sum = new int[len + 1];
        Map<Integer, Integer> map = new HashMap<>();
        map.put(0, 0);

        for (int i = 1; i <= len; i ++) {
            sum[i] = sum[i - 1] + nums[i - 1];
            if (!map.containsKey(sum[i]))
                map.put(sum[i], i);
        }

        // 从后向前遍历数组，i为子数组的结尾，寻找符合条件的前缀和及其索引
        for (int i = len; i > max; i --)
            if (map.containsKey(sum[i] - k))
                max = Math.max(max, i - map.get(sum[i] - k));


```

```
        return max;
    }

作者：iame
链接：https://leetcode-cn.com/problems/maximum-size-subarray-sum-equals-k/solution/325-he-deng-yu-k-de-zui-chang-zi-shu-zu-chang-du-j/
```

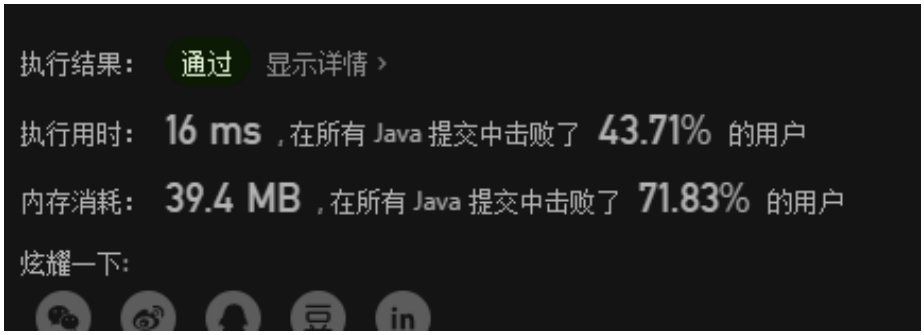
326 Power of Three

执行结果： 通过 显示详情 >

执行用时： 19 ms ，在所有 Java 提交中击败了 24.79% 的用户

内存消耗： 38.5 MB ，在所有 Java 提交中击败了 13.25% 的用户

```java
/*
    二刷
*/
class Solution {
    public boolean isPowerOfThree(int n) {
        if(n == 0)
            return false;

        while(true){
            if(n == 1)
                return true;

            if(n % 3 == 0)
                n /= 3;
            else
                return false;
        }
    }
}
```




```java
//牛逼方法
public boolean isPowerOfThree(int n) {
    // 1162261467 is 3^19,  3^20 is bigger than int
    return ( n>0 &&  1162261467%n==0);
}
```

327 Count of Range Sum 未完成树状数组/分治

执行结果： 通过 显示详情 ›

执行用时： 277 ms ，在所有 Java 提交中击败了 5.15% 的用户

内存消耗： 38.4 MB ，在所有 Java 提交中击败了 82.63% 的用户

```java
/*
    二刷
*/
class Solution {
    public int countRangeSum(int[] nums, int lower, int upper) {
        int len = nums.length;
        long[] prefixSum = new long[len];

        for(int i = 0; i < len; i++){
            if(i == 0)
                prefixSum[i] = nums[i];
            else
                prefixSum[i] += prefixSum[i - 1] + nums[i];
        }

        int count = 0;

        for(int i = 0; i < len; i++){
            for(int j = i; j < len; j++){
                long sum;
                if(i == j)
                    sum = nums[i];
```

```
                else if(i == 0)
                    sum = prefixSum[j];
                else
                    sum = prefixSum[j] - prefixSum[i - 1];

                if(lower <= sum && sum <= upper)
                    count++;
            }
        }

        return count;
    }
}
```

327. Count of Range Sum

难度 困难 109

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` ≤ `j`), inclusive.

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

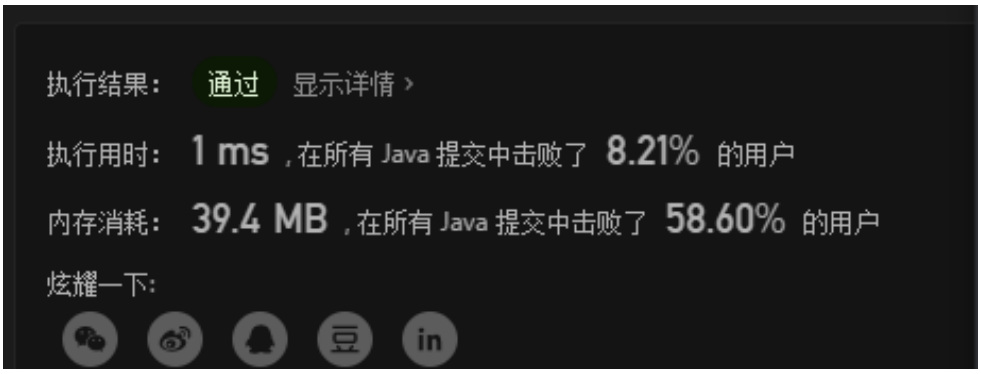
Example:

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`.

```
//暴力解，虽然通过了，但是这个题不算
public int countRangeSum(int[] nums, int lower, int upper) {
    long[] prefix_sum = new long[nums.length + 1];

    for(int i = 1; i < prefix_sum.length; i++)
        prefix_sum[i] = prefix_sum[i-1] + nums[i - 1];


    int count = 0;
```

```
    for(int i = 1; i < prefix_sum.length; i++)
        for(int j = i; j < prefix_sum.length; j++)
            if(prefix_sum[j] - prefix_sum[i-1] >= lower
               && prefix_sum[j] - prefix_sum[i - 1] <= upper)
                count++;

    return count;
}
```

328 Odd Even Linked List




```
public ListNode oddEvenList(ListNode head) {
    if(head == null || head.next == null)
        return head;
    ListNode oddHead = new ListNode(0);
    ListNode evenHead = new ListNode(0);

    ListNode cur = head;    ListNode curOdd = oddHead;  ListNode curEven =
evenHead;
    boolean flag = false;
    while(cur != null)
    {
        if(!flag)
        {
            curOdd.next = cur;
            cur         = cur.next;
            curOdd      = curOdd.next;
            curOdd.next = null;
        }
        else
        {
            curEven.next = cur;
            cur         = cur.next;
            curEven      = curEven.next;
            curEven.next = null;
```

```
        }
        flag = !flag;
    }

    evenHead = evenHead.next;

    while(evenHead != null)
    {
        curOdd.next = evenHead;
        evenHead    = evenHead.next;
        curOdd      = curOdd.next;
    }

    return oddHead.next;
}
```

```
//牛逼方法
public ListNode oddEvenList(ListNode head) {
    if(head == null || head.next == null)       return head;

    ListNode odd = head, even = head.next, evenHead = even;

    while(even != null && even.next != null)
    {
        odd.next  = odd.next.next;
        even.next = even.next.next;
        odd       = odd.next;
        even       = even.next;
    }

    odd.next = evenHead;
    return head;
}
```

329 Longest Increasing Path in a Matrix

329. Longest Increasing Path in a Matrix

难度 困难 306

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].
```

执行结果： 通过 显示详情 >

执行用时： 79 ms , 在所有 Java 提交中击败了 5.35% 的用户

内存消耗： 39.7 MB , 在所有 Java 提交中击败了 5.00% 的用户

炫耀一下:

```java
/*
    郭郭版本的 dfs 二刷

*/
int res =  0;

    /* to memorize the result for each coordinate
        key will be x-coor + "@" + y-coor
        value will be the longest path starting from this point[x, y]
    */
    HashMap<String, Integer> map;
    int row;
    int col;
    int[][] dir = {{0, 1},{1,0},{-1, 0},{0, -1}};
    int[][] max;
    public int longestIncreasingPath(int[][] matrix) {
        map = new HashMap<>();
        row = matrix.length;
```

```java
        col = row == 0 ? 0 : matrix[0].length;
        max = new int[row][col];

        if(col == 0)
            return 0;

        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                String symbol = i + "@" + j;

                if(!map.containsKey(symbol))
                    dfs(matrix, i, j);
            }
        }

        return res;
    }

    private int dfs(int[][] matrix, int i, int j){
        String symbol = i + "@" + j;
        if(map.containsKey(symbol))
            return map.get(symbol);

        int tempMax = 1;

        for(int k = 0; k < dir.length; k++){
            int newX = i + dir[k][0];
            int newY = j + dir[k][1];


            if(isInRange(newX, newY) && matrix[newX][newY] > matrix[i][j]){
                tempMax = Math.max(tempMax, dfs(matrix, newX, newY) + 1);
            }
        }

        res = Math.max(res, tempMax);
        map.put(symbol, tempMax);
        return tempMax;
    }

    private boolean isInRange(int i, int j){
        return i >= 0 && j >= 0 && i < row && j < col;
    }
```

```java
/*
  最直观的想法， DFS， 但是通过135/138

*/
private int m, n;
private int[][] dir = {{1, 0}, {0, 1} ,{-1, 0}, {0, -1}};
private int maxLen = 0;
private int[][] matrix;
private HashMap<int[], Integer> map;
public int longestIncreasingPath(int[][] matrix) {
    m = matrix.length;  n = m == 0 ? 0 : matrix[0].length;
    this.matrix  = matrix;
    map = new HashMap<>();

    for(int i = 0; i < m; i++)
        for(int j = 0; j < n; j++)
            if(map.containsKey(new int[]{i, j}))
                continue;
    else
        dfs(i, j, 1);

    return maxLen;
}

private void dfs(int x, int y, int increSeq)
{
    if(!isInRange(x, y))
        return;

    map.put(new int[]{x, y}, increSeq);
    maxLen = Math.max(maxLen, increSeq);

    for(int k = 0; k < 4; k++)
    {
        int newX = x + dir[k][0];
        int newY = y + dir[k][1];

        if(isInRange(newX, newY) && matrix[newX][newY] > matrix[x][y])
        {
            if(map.containsKey(new int[]{newX, newY}))
                maxLen = Math.max(maxLen, increSeq + map.get(new int[]{newX,
newY}));

            else
                dfs(newX, newY, increSeq + 1);
        }

    }
}
```

```
49
50 private boolean isInRange(int x, int y)
51 {   return x >= 0 && y >= 0 && x < m && y < n;   }
```

```
1  private int m, n;
2  private int[][] matrix;
3  private int[][] max;
4  private int maxLen = 0;
5  private int[][] dir = {{0, 1},{1, 0},{0, -1},{-1, 0}};
6  public int longestIncreasingPath(int[][] matrix) {
7      m = matrix.length;  n = m == 0 ? 0 : matrix[0].length;
8      this.matrix = matrix;
9      max         = new int[m][n];
10
11     for(int i = 0; i < m; i++)
12         for(int j = 0; j < n; j++)
13             if(max[i][j] == 0)
14                 longestIncreasingPath(i, j);
15     return maxLen;
16 }
17
18 //返回以x, y 出发能拿到的最大路径数
19 /*
20   EG: matrix
21   3 4 5
22   3 2 6
23   2 2 1
24
25   对应MAX
26   4 3 2
27   1 4 1
28   2 1 2
29 */
30 private int longestIncreasingPath(int x, int y)
31 {
32     if(max[x][y] != 0)
33         return max[x][y];
34
35     max[x][y] = 1;
36
37     for(int k = 0; k < 4; k++)
38     {
39         int newX = x + dir[k][0];
40         int newY = y + dir[k][1];
41         if(isInRange(newX, newY) && matrix[x][y] < matrix[newX][newY])
42             max[x][y] = Math.max(max[x][y], longestIncreasingPath(newX, newY) + 1);
43
```

```
    }
    maxLen = Math.max(maxLen, max[x][y]);
    return max[x][y];
}

private boolean isInRange(int x, int y)
{   return x >= 0 && y >= 0 && x < m && y < n;  }

作者：gousiqi
链接：https://leetcode-cn.com/problems/longest-increasing-path-in-a-matrix/solution/pu-pu-tong-tong-de-shen-du-you-xian-ji-yi-hua-de-m/
```

330 Patching Array 太难了 未完成

```

```

331 Verify Preorder Serialization of a Binary Tree 栈的应用没理解

```
/*
  关键思想：
    每一个点，可以带来两个slots
    看看最后slots 消了没
*/
class Solution {
  public boolean isValidSerialization(String preorder) {
    // number of available slots
    int slots = 1;

    for(String node : preorder.split(",")) {
      // one node takes one slot
      --slots;

      // no more slots available
      if (slots < 0) return false;

      // non-empty node creates two children slots
```

```
      if (!node.equals("#")) slots += 2;
    }

    // all slots should be used up
    return slots == 0;
  }
}

作者：LeetCode
链接：https://leetcode-cn.com/problems/verify-preorder-serialization-of-a-binary-tree/solution/yan-zheng-er-cha-shu-de-qian-xu-xu-lie-hua-by-leet/
```

```
//使用栈的解法

/*

case 1: 如果是数字，直接压入就完事了，说明我们准备开始一颗新的子树了

case 2.1: 看到 #,  如果栈顶是数字，说明我们得到#是左子树，那么我们要做个标记，下一个来的就是右子树了.

  5

    /\

   #  whatever

case 2.2: you see a #, while top of stack is #, you know you meet this # as right null child, you now cancel the sub tree (rooted as t, for example) with these two-# children. But wait, after the cancellation, you continue to check top of stack is whether # or a number:

---- if a number, say u, you know you just cancelled a node t which is left child of u. You need to leave a # mark to the top of stack. So that the next node know it is a right child.
  u

   /  \

  t    m            当前进度

 /\                ↓

#  # <- 当前进度   | 栈内顺序是 u t # # m

---- if a #, you know you just cancelled a tree whose root, t, is the right child of u. So you continue to cancel sub tree of u, and the process goes on and on.
```

```
  u

    /\

   #  t                当前进度

      /\                  ↓

      # # <- 当前进度 |  栈内顺序 u # t # #

*/

public class Solution {

    public boolean isValidSerialization(String preorder) {
        if (preorder == null)
            return false;

        Stack<String> st = new Stack<>();
        String[] strs = preorder.split(",");

        for (int pos = 0; pos < strs.length; pos++) {
            String curr = strs[pos];
            while (curr.equals("#") && !st.isEmpty() && st.peek().equals(curr))
            {
                st.pop();
                if (st.isEmpty())
                    return false;
                st.pop();

            }
            st.push(curr);
        }
        return st.size() == 1 && st.peek().equals("#");
    }
}

https://leetcode.com/problems/verify-preorder-serialization-of-a-binary-tree/discuss/78566/Java-intuitive-22ms-solution-with-stack
```

332 Reconstruct Itinerary 贪心算法 未完成

执行用时： 13 ms ，在所有 Java 提交中击败了 15.15% 的用户

内存消耗： 39.3 MB ，在所有 Java 提交中击败了 22.66% 的用户

炫耀一下：

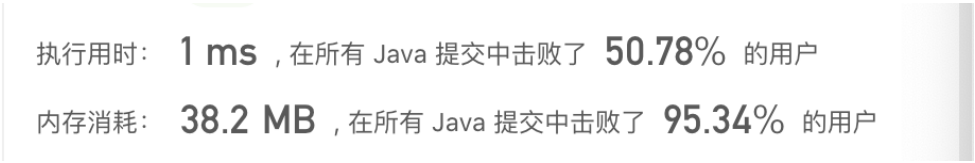
```java
    List<String> res = new ArrayList<>();

    public List<String> findItinerary(List<List<String>> tickets) {
        HashMap<String, TreeMap<String, Integer>> map = new HashMap<>();
        for(List<String> ticket : tickets){
            String from = ticket.get(0);
            String to   = ticket.get(1);

            map.putIfAbsent(from, new TreeMap<>());
            TreeMap<String, Integer> treeMap = map.get(from);
            treeMap.put(to, treeMap.getOrDefault(to, 0) + 1);
        }

        res.add("JFK");
        backtrack(tickets, map, 0);

        return res;
    }

    private boolean backtrack(List<List<String>> tickets, HashMap<String, TreeMap<String, Integer>> map,int progress){
        if(progress == tickets.size()){
            return true;
        }

        TreeMap<String, Integer> tos = map.get(res.get(res.size() - 1));
        if(tos == null || tos.isEmpty() || tos.size() == 0)
            return false;

        for(String str : tos.keySet()){
            if(tos.get(str) == 0)
                continue;

            res.add(str);
            tos.put(str, tos.get(str) - 1);

            if(backtrack(tickets, map, progress + 1))
                return true;

            res.remove(res.size() - 1);
            tos.put(str, tos.get(str) + 1);
        }

        return false;
```

```
    }


```

```
/*
  方法没问题，  但是超时了
  11/80
*/
List<String> res = new ArrayList<>();

    public List<String> findItinerary(List<List<String>> tickets) {
        List<String> path = new ArrayList<>();
        path.add("JFK");
        backtrack(tickets, path, new HashSet<>()) ;

        Collections.sort(res);

        List<String> ans = new ArrayList<>();
        for(String str : res.get(0).split(","))
            ans.add(str);

        return ans;
    }

    private void backtrack(List<List<String>> tickets, List<String> path,
Set<Integer> used){
        if(used.size() == tickets.size()){
            res.add(toStr(path));
            return;
        }

        for(int i = 0; i < tickets.size(); i++){
            if(used.contains(i))
                continue;

            List<String> cur = tickets.get(i);
            String from      = cur.get(0);
            String to        = cur.get(1);

            if(from.equals(path.get(path.size() - 1))){
                path.add(to);
                used.add(i);

                backtrack(tickets, path, used);

```

```
                used.remove(i);
                path.remove(path.size() - 1);
            }

        }
    }

    private String toStr(List<String> path){
        StringBuilder sb = new StringBuilder();

        for(int i = 0; i < path.size(); i++){
            sb.append(path.get(i));

            if(i != path.size() - 1)
                sb.append(",");
        }

        return sb.toString();
    }
```

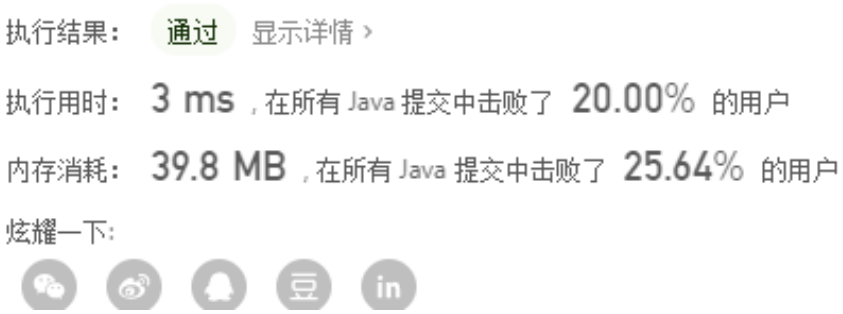
333 Largest BST Subtree




```
/*
    二刷：
    本质就是包装node
    然后记住是postorder
*/
int res = 0;

    class MyNode{
        TreeNode node;
        int max;
        int min;

        /* if this node can be made to be the root of BST
```

```
14          *  how many nodes itself contains?
15          */
16         int size;
17 
18         public MyNode(TreeNode node){
19             this.node = node;
20             max = node.val;
21             min = node.val;
22         }
23 
24         public MyNode(TreeNode node, int max, int min){
25             this.node = node;
26             this.max = max;
27             this.min = min;
28         }
29     }
30     public int largestBSTSubtree(TreeNode root) {
31         if(root == null)
32             return 0;
33         postorder(new MyNode(root));
34 
35         return res;
36     }
37 
38     private void postorder(MyNode root) {
39         if(root == null)
40             return;
41 
42         root.size = 1;
43         MyNode left = null;
44         MyNode right = null;
45 
46         if(root.node.left != null)
47             left = new MyNode(root.node.left, root.node.left.val,
root.node.left.val);
48         if(root.node.right != null)
49             right = new MyNode(root.node.right, root.node.right.val,
root.node.right.val);
50 
51         postorder(left);
52         postorder(right);
53 
54         boolean lValid = left == null || (left.max < root.node.val && left.size >=
1);
55         boolean rValid = right == null || (right.min > root.node.val && right.size
>= 1);
56 
57         if(lValid && rValid) {
```

```
            root.size = (left == null ? 0 : left.size) + (right == null ? 0 :
right.size) + 1;
            if(left != null)
                root.min = left.min;
            if(right != null)
                root.max = right.max;
        }else
            root.size = 0;
        res = Math.max(root.size, res);
    }
```



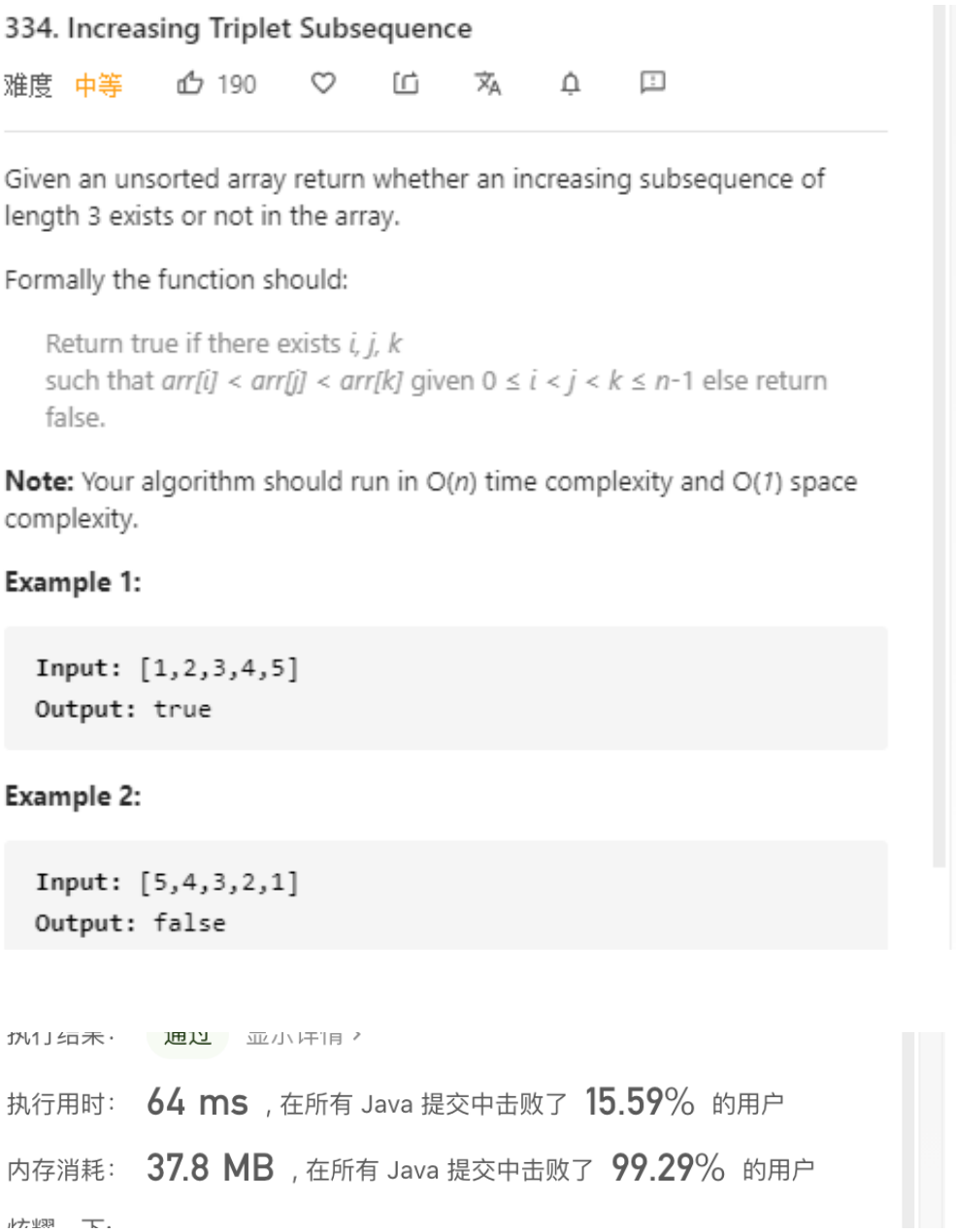

```
//暴力遍历，只不过拿map和set记忆化
int maxLen = 0;
//记录，如果当前节点是BST的root， 记录个数
HashMap<TreeNode, Integer> map;

//记录，如果当前节点是BST， 那么放进set中
HashSet<TreeNode> set;
public int largestBSTSubtree(TreeNode root) {
    map = new HashMap<>();
    set = new HashSet<>();
    return helper(root);
}

//返回该棵树的最大个数
private int helper(TreeNode root)
{
    if(map.containsKey(root))   return map.get(root);
    if(root == null)        return 0;

    if(isValidBST(root))
    {
```

```
        int size = countSize(root);
        map.put(root, size);
        maxLen = Math.max(maxLen, size);
        return map.get(root);
    }

    return Math.max(helper(root.left), helper(root.right));
}

private boolean isValidBST(TreeNode root)
{
    if(root == null)        return true;
    if(set.contains(root))  return true;
    List<Integer> list = new ArrayList<>();
    flatten(root, list);
    for(int i = 1; i < list.size(); i++)
        if(list.get(i - 1) >= list.get(i))
            return false;

    set.add(root);
    return true;
}

private void flatten(TreeNode root, List<Integer> list)
{
    if(root == null)        return;

    flatten(root.left, list);
    list.add(root.val);
    flatten(root.right, list);
}

private int countSize(TreeNode root)
{
    if(root == null)        return 0;
    return 1 + countSize(root.left) + countSize(root.right);
}

//另一种Validate  BST
preVal = Integer.MIN_VALUE;
private boolean isBST(TreeNode root) {
    if (root == null) return true;
    boolean leftFlag = isBST(root.left);
    if (preVal >= root.val) return false;
    preVal = root.val;
    boolean rightFlag = isBST(root.right);
    return leftFlag && rightFlag;
}
```

```java
/*
    左右子树均为BST，且满足 左子树max < node < 右子树min，则当前树也是BST
    左右子树中都搜索到了BST，则返回size更大的
    左右子树之一搜索到了BST，则直接返回
    左右子树都没搜索到BST，则返回null
*/
class Solution {
    class Result {
        TreeNode node; // BST根节点
        int size; // BST的size
        int max; // BST的最大值
        int min; // BST的最小值
    }

    public int largestBSTSubtree(TreeNode root) {
        Result r = visit(root);
        return r == null ? 0 : r.size;
    }

    //一次遍历， visit 方法返回从该节点找到的BST信息，全部包含在result类中
    public Result visit(TreeNode node) {
        if (node == null) return null;

        Result l = null, r = null;

        if (node.left  != null) l = visit(node.left);
        if (node.right != null) r = visit(node.right);

        // 当前树为BST
        boolean lValid = (l == null || (l.node == node.left && l.max < node.val));
        boolean rValid = (r == null || (r.node == node.right && r.min > node.val));

        if (lValid && rValid) {
            Result result = new Result();
            result.node = node;
            result.max = r == null ? node.val : r.max;
            result.min = l == null ? node.val : l.min;
            result.size = (l == null ? 0 : l.size) + (r == null ? 0 : r.size) + 1;
            return result;
        }

        // 左右子树中找到了BST
        if (l != null && r != null)
            return l.size > r.size ? l : r;

        if (l != null) return l;
        if (r != null) return r;
```

```
48
49          return null;
50      }
51  }
52
53  作者：jzj1993
54  链接：https://leetcode-cn.com/problems/largest-bst-subtree/solution/si-lu-qing-xi-de-javadai-ma-onfu-za-du-1ms-by-jzj1/
```

334 Increasing Triplet Subsequence 贪心算法



334. Increasing Triplet Subsequence

难度 中等 190

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 $0 \le i < j < k \le n-1$ 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
```

执行结果： 通过 显示详情

执行用时： 64 ms ，在所有 Java 提交中击败了 15.59% 的用户

内存消耗： 37.8 MB ，在所有 Java 提交中击败了 99.29% 的用户

收藏 一下

```
1  /*
2    二刷：  灵感来源是 leetcode 548 split Array into equal sum
```

```
*/
    public boolean increasingTriplet(int[] nums) {
        int len = nums.length;

        for(int j = 1; j < len - 1; j++){
            boolean first = false;
            for(int i = 0; i < j; i++){
                if(nums[i] < nums[j]){
                    first = true;
                    break;
                }
            }

            if(first == false)
                continue;

            boolean second = false;
            for(int k = j + 1; k < len; k++){
                if(nums[j] < nums[k]){
                    second = true;
                    break;
                }
            }

            if(first && second)
                return true;
        }

        return false;
    }
```

```
/*
  举例[1, 0, 2, 0, -1, 3]
  初始化first = Integer.MAX_VALUE  second = Integer.MAX_VALUE;
  遍历顺序
  遍历 index= 0 -> first = 1, second = Integer.MAX_VALUE;
  遍历 index= 1 -> first = 0, second = Integer.MAX_VALUE;
  遍历 index= 2 -> first = 0, second = 2;
  遍历 index= 3 -> first = 0, second = 2;
  遍历 index= 4 -> (重点)first = -1, second = 2;
  遍历 index= 5 -> first - 01, second = 2 (找到啦！！)
```

```
  在index = 4 ， 为什么当first 要更新为-1， 当我们index = 5时， 可以看到second仍然是前面的2
        那么，当我们第三个数字发现是比second 大，说明在second 之前，还有小的，因此可以成立

  关注这样的案例
  [1, 0, 2, 0, -1, 0, 1]
  当first  = -1, second 自动被更新为0
  在这个案例中，体现了更新first 为最小， second 尽可能为最小的重要性

  个人感悟：从这两个案例中，我们可以看到，实际上对于这个问题的first 和 Second
  只要赋值过了 first 和 Second， 那么也就是随时准备好了两个递增数字， 随时等待第三个

  尽管可能， second 的更新会比first的更新慢，但是更新的慢，证明我们仍然在关注是否以second为第二个数字
  的三元递增序列的存在的可能性， 而如果恰好遍历到最后，second 还是没被更新，说明不行了，找不到
  比如[1, 0, 2, 0, -1, 0]
             ↑
  first = -1 second = 2
  遍历到0的时候，说明以2 的递增序列不用找了， 那么重新复制Secon
  在这个案例中就没有
*/
public boolean increasingTriplet(int[] nums) {
    int first = Integer.MAX_VALUE, second = Integer.MAX_VALUE;

    for(int i = 0; i < nums.length; i++)
    {
        if(nums[i] <= first)
            first = nums[i];
        else if (nums[i] <= second)
            second = nums[i];
        else
            return true;
    }
    return false;
}

```

335 Self Crossing

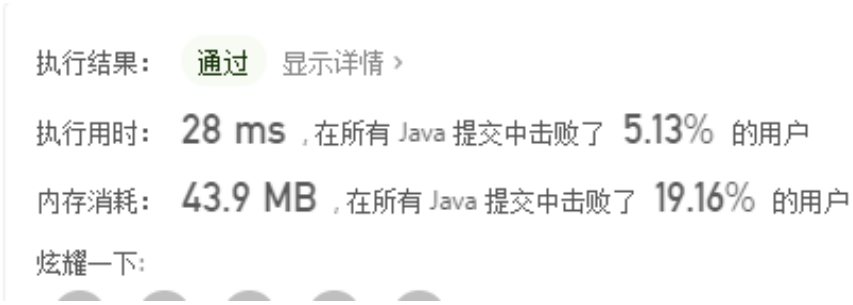
```
//对于图形的分析能力，没别的技巧
public boolean isSelfCrossing(int[] x) {
    if(x.length <= 3)       return false;

```

```
    for(int i = 3; i < x.length; i++)
    {
        if(i >= 3 && x[i-1] <= x[i-3] && x[i] >= x[i-2])
            return true;
        if(i >= 4 && x[i-1] == x[i-3] && x[i-2] <= x[i-4] + x[i])
            return true;
        if(i >= 5 && x[i-2] <= x[i-4] + x[i] && x[i-1] >= x[i-3] - x[i-5]
           && x[i-5] >= 0 && x[i-2] > x[i-4] && x[i-3] >= x[i-1])
            return true;
    }

    return false;
}
https://leetcode-cn.com/problems/self-crossing/solution/yi-bu-yi-bu-fen-xi-by-lzh_yves/
```

336 Palindrome Pairs 马拉车 + 前缀树 未完成

337 House Robber III

338 Counting Bits




```
public int[] countBits(int num) {
    int[] res = new int[num + 1];
    for(int i = 0; i <= num; i++)
        res[i] = count(i);
    return res;
}

private int count(int num)
{
    int count = 0;
```

```
    for(int i = 0; i < 32; i++)
    {
        count += ((num & 1) == 1 ? 1 : 0);
        num /= 2;
    }

    return count;
}
```

```
//优化 奇数永远比相邻偶数多一个1
// 偶数除2相同， 因为 2 = 10  4 = 100
public int[] countBits(int num) {
    int[] res = new int[num + 1];
    for(int i = 1; i <= num; i++)
    {
        if(i % 2 == 0)
            res[i] = res[i / 2];
        else
            res[i] = res[i-1] + 1;
    }
    return res;
}
```

339 Nested List Weight Sum

执行结果： 通过 显示详情 >

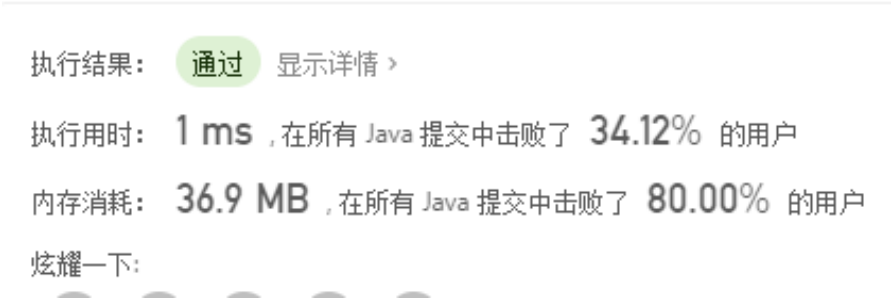
执行用时： 0 ms , 在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 35.9 MB , 在所有 Java 提交中击败了 46.88% 的用户

炫耀一下:

```
/*
    二刷
*/
class Solution {
    int sum = 0;
    public int depthSum(List<NestedInteger> nestedList) {
        if(nestedList == null || nestedList.size() == 0)
            return 0;

        depthSum(nestedList, 1);
        return sum;
    }

```

```
    private void depthSum(List<NestedInteger> nestedList, int level){
        if(nestedList == null || nestedList.size() == 0)
            return;

        for(NestedInteger ni : nestedList){
            if(ni.isInteger())
                sum += ni.getInteger() * level;
            else
                depthSum(ni.getList(), level + 1);
        }
    }
}
```




```
    public int depthSum(List<NestedInteger> nestedList) {
         int res= 0;
         return depthSum(nestedList, 1);
     }

     private int depthSum(List<NestedInteger> nestedList, int level)
     {
         int res = 0;
         for(int i = 0; i < nestedList.size(); i++)
         {
             if(nestedList.get(i).isInteger())
                 res += (nestedList.get(i).getInteger() * level);
             else
                 res += depthSum(nestedList.get(i).getList(), level + 1)  ;
         }

         return res;
     }
     public int lengthOfLongestSubstringKDistinct(String s, int k) {
         if(k == 0 || s.length() == 0)       return 0;
         int left = 0, right = 0;
         HashMap<Character, Integer> map = new HashMap<>();
         int count = 0;
         int maxLen = 0;
```

```
25
26          while(right < s.length())
27          {
28              while(right < s.length() && count <= k)
29              {
30                  char ch = s.charAt(right);
31                  map.put(ch, map.getOrDefault(ch, 0) + 1);
32                  if(map.get(ch) == 1)
33                      count++;
34                  if(count <= k)
35                      maxLen = Math.max(maxLen, right - left + 1);
36                  right++;
37              }
38
39              while(count > k)
40              {
41                  char ch = s.charAt(left);
42                  map.put(ch, map.get(ch) - 1);
43                  if(map.get(ch).equals(0))
44                      count--;
45                  left++;
46              }
47
48          }
49          return maxLen;
50      }
51  /*
52    与上面339题类似，  如果不能使用递归，那么我们就使用栈
53  */
54  public class NestedIterator implements Iterator<Integer> {
55      Deque<NestedInteger> stack = new ArrayDeque<>();
56      public NestedIterator(List<NestedInteger> nestedList) {
57          prepareStack(nestedList);
58      }
59
60      @Override
61      public Integer next() {
62          if(!hasNext())
63              return null;
64          return stack.pop().getInteger();
65      }
66
67      @Override
68      public boolean hasNext() {
69          while(!stack.isEmpty())
70          {
71              if(stack.peek().isInteger())
72                  return true;
73              else
```

```
                prepareStack(stack.pop().getList());
        }
        return false;
    }

    private void prepareStack(List<NestedInteger> nestedList)
    {
        for(int i = nestedList.size() - 1; i >= 0; i--)
            stack.push(nestedList.get(i));
    }
}
    public boolean isPowerOfFour(int num) {
        if(num > 1073741824)        return false;

        int res = 1;
        while(res <= num)
        {
            if(res == num)
                return true;
            res <<= 2;

        }
        return false;
    }
/*
  试了回溯和递归，下面是递归方法，还是不太可以
  只能通过25/50 个案例

  使用了记忆，就可以了
  自顶向下
*/
    private int n;
    private HashMap<Integer, Integer> map;
    public int integerBreak(int n) {
        if(n == 1)  return 1;
        map = new HashMap<>();
        this.n = n;

        return helper(n, 1);
    }
    //返回， 给定remains 的最大结果
    private int helper(int remains, int start)
    {
        if(map.containsKey(remains))    return map.get(remains);
        int res = 1;

        if(remains <= 0)   return res;

        for(int i = start; i < n; i++)
```

```
        {
            if(i > remains) continue;

            res = Math.max(res, helper(remains - i, start) * i);
        }

        map.put(remains, res);
        return res;
    }
//dp的方法，蛮好的
//自底向上
/*
  动归方程
  将n 分解成两个数 F(n) = i * (n - i);
  将n 分解为多个数
    F(n) = i * F(n - i);
    F(n) = F(i) * (n - i);
    F(n) = F(i) * F(n - i);

*/
public int integerBreak3(int n) {
    memory[2] = 1;
    for (int i = 3; i <= n; i++) {
        for ( int j = 1; j <= i - 1; j++) {
            memory[i] = Math.max(memory[i], Math.max(j * memory[i - j], j * (i - j)));
        }
    }
    return memory[n];
}

作者：97wgl
链接：https://leetcode-cn.com/problems/integer-break/solution/bao-li-sou-suo-ji-yi-hua-sou-suo-dong-tai-gui-hua-/
public String reverseVowels(String s) {
    int left = 0, right = s.length() - 1;
    char[] chars = s.toCharArray();
    HashSet<Character> set = new HashSet<>();
    set.add('a');set.add('e');set.add('i');set.add('o');set.add('u');
    set.add('A');set.add('E');set.add('I');set.add('O');set.add('U');

    while(left < right)
    {
        while(left < right && !set.contains(chars[left]))
            left++;
        while(left < right && !set.contains(chars[right]))
            right--;

        char ch = chars[left];
```

```
        chars[left] = chars[right];
        chars[right] = ch;
        left++; right--;
    }

    StringBuilder res = new StringBuilder();
    for(char ch : chars)
        res.append(ch);
    return res.toString();
}
class MovingAverage {
    private double sum;
    private Deque<Integer> queue;
    private int size;

    /** Initialize your data structure here. */
    public MovingAverage(int size) {
        this.sum = 0.0;
        this.queue = new ArrayDeque<>();
        this.size = size;
    }

    public double next(int val) {
        if(queue.size() == size)
        {
            int drop = queue.removeFirst();
            sum -= (double)drop;
            sum += (double)val;
            queue.addLast(val);
            return (sum / queue.size());
        }
        else
        {
            queue.addLast(val);
            sum += (double)val;
            return sum / queue.size();
        }

    }
}

public int[] intersect(int[] nums1, int[] nums2) {
    HashMap<Integer, Integer> map1 = new HashMap<>();
    HashMap<Integer, Integer> map2 = new HashMap<>();

    for(int num : nums1)
        map1.put(num, map1.getOrDefault(num, 0) + 1);
    for(int num : nums2)
        map2.put(num, map2.getOrDefault(num, 0) + 1);
```

```
219
220     List<Integer> list = new ArrayList<>();
221     for(Integer num : map1.keySet())
222         if(map2.containsKey(num))
223         {
224             int freq = Math.min(map1.get(num), map2.get(num));
225             for(int i = 0; i < freq; i++)
226                 list.add(num);
227         }
228
229     int[] res = new int[list.size()];
230     int index = 0;
231     for(Integer num : list)
232         res[index++] = num;
233     return res;
234 }
```

340 Longest Substring with At Most K Distinct Characters

执行结果: 通过 显示详情 >

执行用时: 9 ms , 在所有 Java 提交中击败了 53.56% 的用户

内存消耗: 38.3 MB , 在所有 Java 提交中击败了 93.47% 的用户

```
1  /*
2      二刷
3  */
4  class Solution {
5      public int lengthOfLongestSubstringKDistinct(String s, int k) {
6          Map<Character, Integer> map = new HashMap<>();
7
8          int left = 0, right = 0;
9          int len = s.length();
10
11         int res = 0;
12         while(right < len){
13             while(right < len && map.size() <= k){
14                 char ch = s.charAt(right);
15                 map.put(ch, map.getOrDefault(ch, 0) + 1);
16                 if(map.size() <= k)
```

```
                    res = Math.max(res, right - left + 1);

                right++;
            }

            while(left < right && map.size() > k){
                char ch = s.charAt(left);
                map.put(ch, map.getOrDefault(ch, 0) - 1);

                if(map.getOrDefault(ch, 0) == 0)
                    map.remove(ch);

                left++;
            }
        }

        return res;
    }
}
```

341 Flatten Nested List Iterator

3 分钟前	通过	3 ms	40.9 MB	J

```
//二刷
public class NestedIterator implements Iterator<Integer> {

    private List<Integer> list;
    private int progress;
    private List<NestedInteger> nestedList;
    public NestedIterator(List<NestedInteger> nestedList) {
        list = new ArrayList<>();
        progress = 0;
        this.nestedList = nestedList;

        flatten(nestedList);
    }

    private void flatten(List<NestedInteger> nestedList){
        for(NestedInteger ni : nestedList){
            if(ni.isInteger())
```

```
                list.add(ni.getInteger());
            else
                flatten(ni.getList());
        }
    }


    @Override
    public Integer next() {
        if(hasNext()){
            return list.get(progress++);
        }else{
            return null;
        }
    }

    @Override
    public boolean hasNext() {
        return progress != list.size();
    }
}
```

342 Power of Four

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].
```

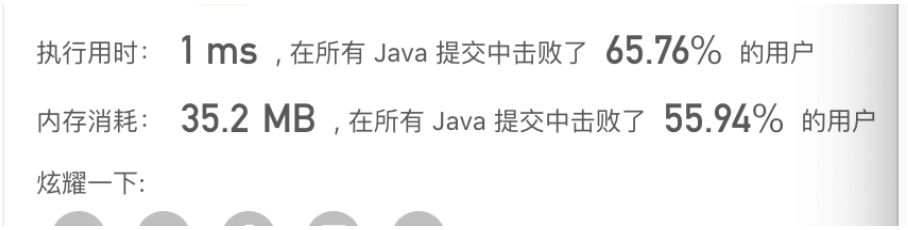
执行用时： 1 ms , 在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 35.8 MB , 在所有 Java 提交中击败了 10.72% 的用户

```java
/*
  二刷， 打表
*/
class Solution {
    public boolean isPowerOfFour(int n) {
        int[] nums ={1,4,16,64,256,1024,4096,16384,65536,262144,1048576,
                     4194304,16777216,67108864,268435456,1073741824};

        for(int i = 0; i < nums.length; i++)
            if(nums[i] == n)
                return true;


        return false;
    }
}
```

```java
//一刷
class Solution {
    public boolean isPowerOfFour(int num) {
        if(num > 1073741824)        return false;
        int res = 1;
        while(res <= num)
        {
            if(res == num)
                return true;
            res <<= 2;

        }
        return false;
    }
}

```

343 Integer Break




```java
//二刷
class Solution {
    /* this will store the max product for each interger
       used to memorize the result during middle procee, speed it up

       key: int to be splited
       value: max product corresponding to the integer

    */
    Map<Integer, Integer> map = new HashMap<>();
    public int integerBreak(int n) {
        map.put(1, 1);
        map.put(2, 1);
        backtrack(n);
        return map.get(n);
    }

   // return the  max result for this n
    private int backtrack(int n){
```

```
        if(map.containsKey(n))
            return map.get(n);

        int res = 0;
        for(int i = 1; i < n; i++){
            int first = i;
            int second = n - i;

            res = Math.max(res, Math.max(i * (n - i), Math.max(backtrack(i) * (n - i), i * backtrack(n - i))));
        }

        map.put(n, res);

        return res;
    }
}
```

执行用时： 3 ms , 在所有 Java 提交中击败了 13.25% 的用户

内存消耗： 35.6 MB , 在所有 Java 提交中击败了 5.13% 的用户

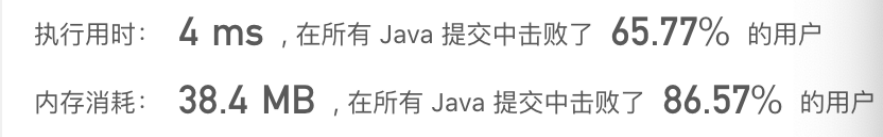
```
//二刷 DP
class Solution {
    public int integerBreak(int n) {
        int[] dp = new int[n + 1];
        dp[1] = 1;
        dp[2] = 1;

        for(int i = 3; i <= n; i++){
            for(int j = 1; j < i; j++){
                dp[i] = Math.max(dp[i], Math.max(j * (i - j), Math.max(dp[j] * (i - j), Math.max(j * dp[i - j], dp[j] * dp[i - j]))));
            }
        }

        return dp[n];
    }
}
```

344 Reverse String

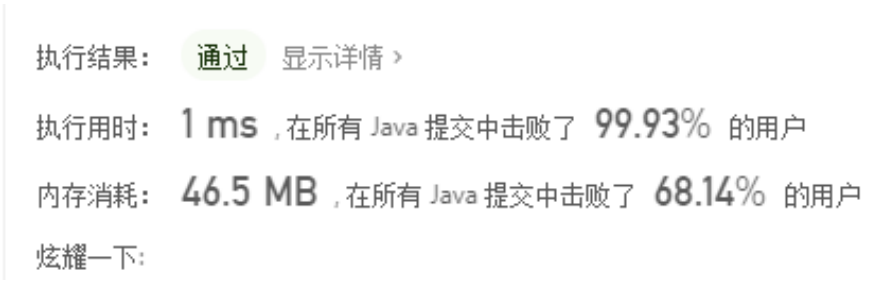
提交时间	提交结果	运行时间	内存消耗	语
37 分钟前	通过	1 ms	36.3 MB	Ja

```java
public void reverseString(char[] s) {
    int left = 0, right = s.length - 1;

    while(left < right){
        char temp = s[left];
        s[left] = s[right];
        s[right] = temp;
        left++;
        right--;
    }

}
```

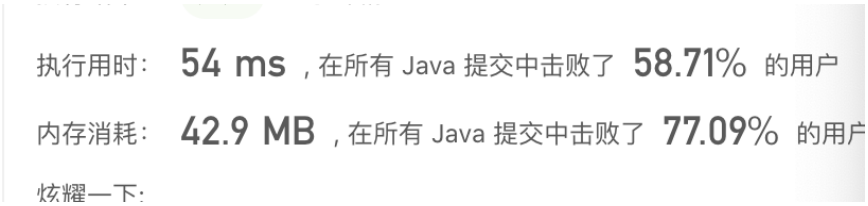
345 Reverse Vowels of a String



```java
//二刷
class Solution {
    public String reverseVowels(String s) {
        char[] chars = s.toCharArray();

        int left = 0, right = chars.length - 1;


        while(left < right){
            while(left < right && !isVowel(chars[left]))
                left++;

            if(left >= right)
                break;

            while(left < right && !isVowel(chars[right]))
                right--;

            if(left >= right)
                break;

            char ch = chars[left];
            chars[left] = chars[right];
            chars[right] = ch;

            left++;
            right--;
        }


        StringBuilder sb = new StringBuilder();
        for(char ch : chars)
            sb.append(ch);
        return sb.toString();
    }

    private boolean isVowel(char ch){
        return ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
            ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U';
    }
}
```

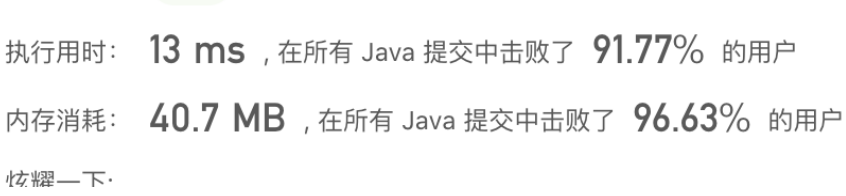


346 Moving Average from Data Stream




```java
//二刷
class MovingAverage {

    private Deque<Integer> queue;
    private int cap;
    private int sum;
    /** Initialize your data structure here. */
    public MovingAverage(int size) {
        queue    = new ArrayDeque<>();
        this.cap = size;
        this.sum  = 0;
    }

    public double next(int val) {
        if(queue.size() == cap)    {
            sum -= queue.removeFirst();
            sum += val;
            queue.addLast(val);
        }else{
            sum += val;
            queue.addLast(val);
        }

        return sum / (queue.size() * 1.0);

    }
}
```

347 Top K Frequent Elements




```java
//继续二刷，改进算法，成为快速选择
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();

        for(int num : nums)
            map.put(num, map.getOrDefault(num, 0) + 1);

        Node[] quickSelect = new Node[map.size()];
        int index = 0;
        for(Integer num : map.keySet())
            quickSelect[index++] = new Node(num, map.get(num));

        int targetIndex = map.size() - k;
        int lo = 0, hi = map.size() - 1;
        while(true){
            int tempIndex = quickSort(quickSelect, lo, hi);

            if(tempIndex == targetIndex)
                return copyRes(quickSelect, targetIndex);
            else if(tempIndex > targetIndex)
                hi = tempIndex - 1;
            else
                lo = tempIndex + 1;
        }
    }

    private int quickSort(Node[] quickSelect, int lo, int hi){
```

```
        if(lo == hi)
            return lo;

        int i = lo, j = hi + 1;
        int cmp = quickSelect[i].freq;

        while(true){
            while(quickSelect[++i].freq < cmp){
                if(i == hi)
                    break;
            }

            while(quickSelect[--j].freq > cmp){
                if(j == lo)
                    break;
            }

            if(i >= j)
                break;

            exch(quickSelect, i, j);
        }

        exch(quickSelect, j, lo);
        return j;
    }

    private void exch(Node[] quickSelect, int i, int j){
        Node temp      = quickSelect[i];
        quickSelect[i] = quickSelect[j];
        quickSelect[j] = temp;
    }

    private int[] copyRes(Node[] quickSelect, int index){
        int[] res = new int[quickSelect.length - index];

        for(int i = 0; i < res.length; i++)
            res[i] = quickSelect[index++].val;

        return res;
    }
}

class Node{
    public int val;
    public int freq;

    public Node(int val, int freq){
        this.val = val;
```

```
        this.freq = freq;
    }
}
```

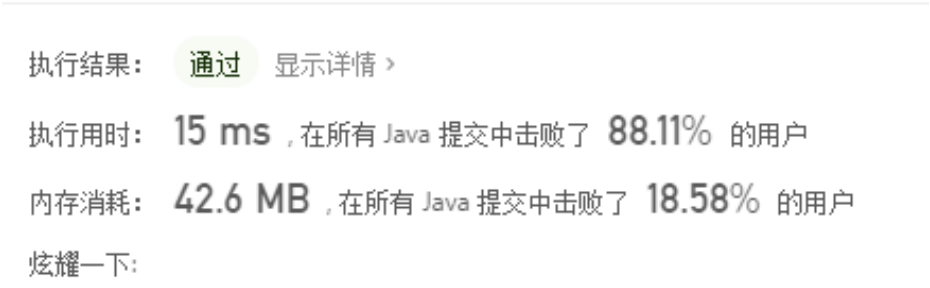
执行用时： 15 ms ，在所有 Java 提交中击败了 69.37% 的用户

内存消耗： 41.3 MB ，在所有 Java 提交中击败了 10.63% 的用户

```
//二刷

class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();

        for(int num : nums)
            map.put(num, map.getOrDefault(num, 0) + 1);

        PriorityQueue<Node> pq = new PriorityQueue<Node>((o1, o2) -> (o2.freq - o1.freq));
        for(Integer num : map.keySet())   //(Onlgn)
            pq.add(new Node(num, map.get(num)));

        int[] res = new int[k];
        for(int i = 0; i < res.length; i++){
            Node node = pq.poll();  //lgn
            res[i] = node.val;
        }

        return res;
    }
}

class Node{
    public int freq;
    public int val;

    public Node(int val, int freq){
        this.val = val;
        this.freq = freq;
    }
}
```

```
public int[] topKFrequent(int[] nums, int k) {
```

```
    HashMap<Integer, Integer> map = new HashMap<>();
    PriorityQueue<Integer> pq     = new PriorityQueue<>((o1, o2) -> o2 - o1);
    for(int num : nums)
        map.put(num, map.getOrDefault(num, 0) + 1);

    for(Integer freq : map.values())
        pq.offer(freq);

    HashSet<Integer> set = new HashSet<>();
    for(int i = 0; i < k; i++)
    {
        set.add(pq.peek());
        pq.remove();
    }

    int[] res = new int[k];
    int index = 0;
    for(Integer num : map.keySet())
        if(set.contains(map.get(num)))
            res[index++] = num;
    return res;
}
```



```java
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        HashMap<Integer, Integer> map = new HashMap<>();
        PriorityQueue<Integer> pq     = new PriorityQueue<>((o1, o2) -> o2 - o1);
        for(int num : nums)
            map.put(num, map.getOrDefault(num, 0) + 1);

        for(Integer freq : map.values())
            pq.offer(freq);
        HashSet<Integer> set = new HashSet<>();
        for(int i = 0; i < k; i++)
        {
            set.add(pq.peek());
            pq.remove();
        }

        int[] res = new int[k];
        int index = 0;
        for(Integer num : map.keySet())
            if(set.contains(map.get(num)))
                res[index++] = num;
        return res;
    }
}
```

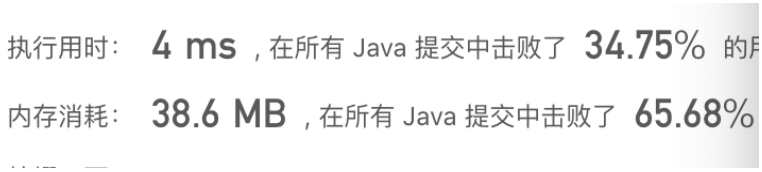
348 Tic Tac Toe


```java
//二刷
class TicTacToe {
    int[][] board;
    /** Initialize your data structure here. */
```

```java
    public TicTacToe(int n) {
        board = new int[n][n];
    }

    /** Player {player} makes a move at ({row}, {col}).
        @param row The row of the board.
        @param col The column of the board.
        @param player The player, can be either 1 or 2.
        @return The current winning condition, can be either:
                0: No one wins.
                1: Player 1 wins.
                2: Player 2 wins. */
    public int move(int row, int col, int player) {
        int token = player == 1 ? 1 : 2;

        board[row][col] = token;
        for(int i = 0; i <= board.length; i++){
            if(i == board.length)
                return token;
            if(board[i][col] != token){
                break;
            }
        }

        for(int j = 0; j <= board[0].length; j++){
            if(j == board[0].length)
                return token;
            if(board[row][j] != token){
                break;
            }
        }

        int r = board.length - 1;
        int c = 0;
        while(board.length - row - 1 == col){
            if(board[r][c] != token){
                break;
            }

            r--; c++;
            if(c == board[0].length || r < 0)
                return token;
        }

        r = 0;
        c = 0;
        while(row == col){
            if(board[r][c] != token){
                break;
```

```
            }

            r++;
            c++;
            if(c == board[0].length)
                return token;
        }

        return 0;

    }
}

```

349 Intersection to Two Arrays




```
//二刷
class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        HashSet<Integer> set1 = new HashSet<>();
        HashSet<Integer> set2 = new HashSet<>();

        for(int num : nums1)
            set1.add(num);
        for(int num : nums2)
            set2.add(num);

        List<Integer> res = new ArrayList<>();
        for(Integer num : set1){
            if(set2.contains(num))
                res.add(num);
        }

        int[] ans = new int[res.size()];
        for(int i = 0; i < res.size(); i++)
            ans[i] = res.get(i);

        return ans;
    }
}
```

350 Intersection to Two Arrays

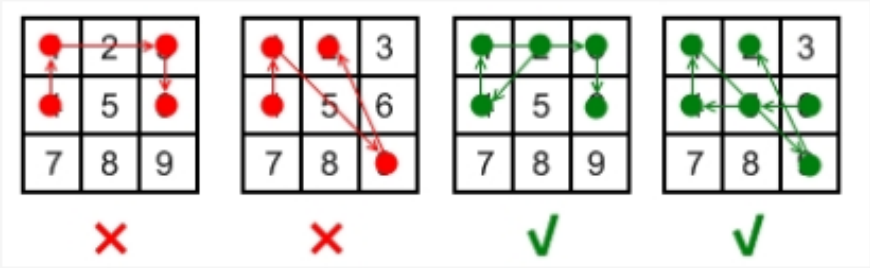

```java
//二刷
class Solution {
    public int[] intersect(int[] nums1, int[] nums2) {

        HashMap<Integer, Integer> map1 = new HashMap<>();
        HashMap<Integer, Integer> map2 = new HashMap<>();

        for(int num : nums1)
            map1.put(num, map1.getOrDefault(num, 0) + 1);
        for(int num : nums2)
            map2.put(num, map2.getOrDefault(num, 0) + 1);

        List<Integer> res = new ArrayList<>();
        for(Integer num : map1.keySet()){
            if(map2.containsKey(num)){
                for(int i = 0; i < Math.min(map1.get(num), map2.get(num)); i++)
                    res.add(num);
            }
        }

        int[] ans = new int[res.size()];
        for(int i = 0; i < res.size(); i++)
            ans[i] = res.get(i);

        return ans;
    }
}
```

351 Android Unlock Patterns 回溯 + 并查集的典型应用

Given an Android **3x3** key lock screen and two integers **m** and **n**, where 1 ≤ m ≤ n ≤ 9, count the total number of unlock patterns of the Android lock screen, which consist of minimum of **m** keys and maximum **n** keys.

Rules for a valid pattern:

1. Each pattern must connect at least **m** keys and at most **n** keys.
2. All the keys must be distinct.
3. If the line connecting two consecutive keys in the pattern passes through any other keys, the other keys must have previously selected in the pattern. No jumps through non selected key is allowed.
4. The order of keys used matters.




```java
//二刷
class Solution {
    private int res = 0;
    WeightedUnionFind wuf = new WeightedUnionFind(10);

    public int numberOfPatterns(int m, int n) {
        wuf.union(1, 3);
        wuf.union(1, 7);
        wuf.union(3, 9);
        wuf.union(2, 8);
        wuf.union(4, 6);

        boolean[] visited = new boolean[10];
        for(int i = 1; i <= 9; i++)
            dfs(i, 1, visited, m, n);
        return res;
    }

    private void dfs(int currentKey, int keyNum, boolean[] visited, int m, int n){
```

```
        if(keyNum >= m && keyNum <= n)
            res++;
        else if(keyNum  > n)
            return;

        visited[currentKey] = true;

        for(int i = 1; i <= 9; i++){
            boolean cross = wuf.isConnected(i, currentKey);

            if(!visited[i] && (!cross || visited[(i + currentKey) / 2]))
                dfs(i, keyNum + 1, visited, m, n);
        }

        visited[currentKey] = false;
    }
}

class WeightedUnionFind{
    private int[] id;
    private int[] sz;

    public WeightedUnionFind(int N){
        id = new int[N];
        sz = new int[N];
        for(int i = 0; i < N; i++)
            id[i] = i;
        for(int i = 0; i < N; i++)
            sz[i] = 1;
    }

    private int find(int p){
        while(p != id[p]){
            id[p] = id[id[p]];
            p = id[p];
        }

        return p;
    }
    public void union(int p, int q){
        int pRoot = find(p);
        int qRoot = find(q);

        if(pRoot == qRoot){
            return ;
        }

        if(sz[pRoot] > sz[qRoot]){
            sz[pRoot] += sz[qRoot];
```

```
            id[qRoot] = id[pRoot];
        }else{
            sz[qRoot] += sz[pRoot];
            id[pRoot] = id[qRoot];
        }
    }

    public boolean isConnected(int p, int q){
        return find(p) == find(q);
    }
}


```

```
class Solution {
    private int res;
    WeightedUnionFind wuf = new WeightedUnionFind(10);
    public int numberOfPatterns(int m, int n) {

        init(wuf);
        res = 0;
        boolean[] visited = new boolean[10];

        for(int i = 1; i <= 9; i++)
            dfs(i, 1, visited, m, n);
        return res;
    }

    private void dfs(int currentKey, int keyNum, boolean[] visited, int lo, int hi)
    {
        if(keyNum >= lo && keyNum <= hi)
            res++;
        else if(keyNum > hi)
            return;

        visited[currentKey] = true;
        for(int i = 1; i <= 9; i++)
        {
            boolean crossNumber = wuf.connected(currentKey, i);
            if(!visited[i] && (!crossNumber || visited[(currentKey + i) / 2]))
                dfs(i, keyNum + 1, visited, lo, hi);
        }
```

```
29
30          visited[currentKey] = false;
31      }
32
33      private void init(WeightedUnionFind wuf)
34      {
35          wuf.union(1,3);
36          wuf.union(1,7);
37          wuf.union(1,9);
38          wuf.union(2,8);
39          wuf.union(6,4);
40      }
41  }
42
43  class WeightedUnionFind
44  {
45      private int[] id;
46      private int[] size;
47
48      public WeightedUnionFind(int count)
49      {
50          id = new int[count];
51          for(int i = 0; i < id.length; i++)
52              id[i] = i;
53          size = new int[count];
54          Arrays.fill(size, 1);
55      }
56
57      public int find(int p)
58      {
59          while(p != id[p])
60          {
61              id[p] = id[id[p]];
62              p = id[p];
63          }
64          return p;
65      }
66
67      public boolean connected(int p, int q)
68      {   return find(p) == find(q);  }
69
70      public void union(int p, int q)
71      {
72          int rootP = find(p);
73          int rootQ = find(q);
74
75          if(rootP == rootQ)  return;
76
77          if(size[p] > size[q])
```

```
        {
            id[rootQ] = id[rootP];
            size[p] += size[q];
        }
        else
        {
            id[rootP] = id[rootQ];
            size[q] += size[p];
        }
    }
}

作者：wu-ai-4
链接：https://leetcode-cn.com/problems/android-unlock-patterns/solution/351-an-zhuo-xi-tong-shou-shi-jie-suo-bing-cha-ji-h/
```

```
class Solution {
    public int numberOfPatterns(int m, int n) {
        int [][]skip = new int[10][10];
        //这个skip数组是为了记录跳跃的点数，比如说从1到3，就跳跃2
        //而且因为是对称的操作，所以3到1也是如此
        skip[1][3] = skip[3][1] = 2;
        skip[1][7] = skip[7][1] = 4;
        skip[3][9] = skip[9][3] = 6;
        skip[4][6] = skip[6][4] = skip[2][8] = skip[8][2] = 5;
        skip[1][9] = skip[9][1] = skip[3][7] = skip[7][3] = 5;
        skip[7][9] = skip[9][7] = 8;

        int result = 0;
        boolean[] visited = new boolean[10];

        //深度遍历，遍历每一个点到点的次数
        for(int i = m; i<=n; i++){
            //因为从1,3,7,9出发都是对称的，为什么i要减一呢，因为我们是从1出发，先天少了一个节点
            result += DFS(1,visited,skip,i-1)*4;

            //2,4,6,8对称
            result += DFS(2,visited,skip,i-1)*4;

            //唯独5独立
            result += DFS(5,visited,skip,i-1);
        }
        return result;
    }

    //深度遍历
```

```
    public int DFS(int current, boolean[] visited, int[][] skip,int remainKeyCount)
{
        if(remainKeyCount == 0)
            return 1;

        int result = 0;
        //深度遍历都是渣男，做了又要反悔找下家
        visited[current] = true;

        for(int i = 1; i <= 9; i++)
        {
            //看当前的节点到i节点的路径中有没有其他节点在中间
              int crossThroughNumber = skip[current][i];

              //如果这一次我们的i节点没有被读过
              //那么就判断有没有路过中间节点(visited[crossThroughNumber])或者
              //这两个节点相邻没有中间节点（currentThrough=0）
              if(!visited[i] && (crossThroughNumber == 0 ||
visited[crossThroughNumber]))
                  result += DFS(i,visited,skip,remainKeyCount-1);

        }


        //渣男行径开始了
        visited[current] = false;
        return result;
    }
}

作者：a-le-m
链接：https://leetcode-cn.com/problems/android-unlock-patterns/solution/vivode-bi-
shi-ti-by-wei-zhi-meng/
```

352 Data Stream as Disjoint Intervals

352. Data Stream as Disjoint Intervals

难度 困难 58

Given a data stream input of non-negative integers a_1, a_2, ..., a_n, ..., 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?

通过次数 3,374 | 提交次数 5,850

执行用时： 91 ms , 在所有 Java 提交中击败了 32.67% 的用户

内存消耗： 43.3 MB , 在所有 Java 提交中击败了 96.67% 的用户

炫耀一下:

```java
// 二刷
class SummaryRanges {
  List<int[]> data;
    /** Initialize your data structure here. */
    public SummaryRanges() {
        data = new ArrayList<>();
    }

    public void addNum(int val) {
        data.add(new int[]{val, val});
    }

    public int[][] getIntervals() {
        int[][] res = merge(data);
        data.clear();
        for(int[] r : res)
            data.add(r);
```

```
        return res;
    }

    private int[][] merge(List<int[]> intervals) {
        Collections.sort(intervals, (o1, o2) -> (o1[0] == o2[0] ? o1[1] - o2[1] : o1[0] - o2[0]));

        List<int[]> res = new ArrayList<>();
        int left = 0, right = 0;

        while(right < intervals.size()){
            int leftBound  = intervals.get(left)[0];
            int rightBound = intervals.get(left)[1];

            while(right < intervals.size() && rightBound + 1 >= intervals.get(right)[0]){
                rightBound = Math.max(rightBound, intervals.get(right)[1]);
                right++;
            }

            res.add(new int[]{leftBound, rightBound});
            left = right;
        }

        int[][] ans = new int[res.size()][2];
        for(int i = 0; i < res.size(); i++){
            ans[i][0] = res.get(i)[0];
            ans[i][1] = res.get(i)[1];
        }

        return ans;
    }

}
```

```
/*
  source https://leetcode.com/problems/data-stream-as-disjoint-intervals/discuss/82553/Java-solution-using-TreeMap-real-O(logN)-per-adding.
*/
public class SummaryRanges {
    TreeMap<Integer, Interval> tree;

    public SummaryRanges() {
        tree = new TreeMap<>();
```

```
    }

    public void addNum(int val) {
        if(tree.containsKey(val)) return;
        Integer l = tree.lowerKey(val);
        Integer h = tree.higherKey(val);
        if(l != null && h != null && tree.get(l).end + 1 == val && h == val + 1) {
            tree.get(l).end = tree.get(h).end;
            tree.remove(h);
        } else if(l != null && tree.get(l).end + 1 >= val) {
            tree.get(l).end = Math.max(tree.get(l).end, val);
        } else if(h != null && h == val + 1) {
            tree.put(val, new Interval(val, tree.get(h).end));
            tree.remove(h);
        } else {
            tree.put(val, new Interval(val, val));
        }
    }

    public List<Interval> getIntervals() {
        return new ArrayList<>(tree.values());
    }
}
```

353 Design Snake Game 典型 双端队列的应用

353. Design Snake Game

难度 中等 21

Design a Snake game that is played on a device with screen size = *width* x *height*. Play the game online if you are not familiar with the game.

The snake is initially positioned at the top left corner (0,0) with length = 1 unit.

You are given a list of food's positions in row-column order. When a snake eats the food, its length and the game's score both increase by 1.

Each food appears one by one on the screen. For example, the second food will not appear until the first food was eaten by the snake.

When a food does appear on the screen, it is guaranteed that it will not appear on a block occupied by the snake.

执行结果： 通过 显示详情 >

执行用时： 73 ms ，在所有 Java 提交中击败了 25.74% 的用户

内存消耗： 44.3 MB ，在所有 Java 提交中击败了 88.12% 的用户

炫耀一下:

```java
/*
    注意两个点
      1. 如果将要碰的点，是我们身体的最后一个格子，那其实无所谓，不需要返回 -1
*/
class SnakeGame {

    private int progress;
    private int row;
    private int col;
    private Deque<int[]> queue;
    private int score;
    private int[][] food;
    private HashSet<String> myBody;

    /** Initialize your data structure here.
        @param width - screen width
        @param height - screen height
        @param food - A list of food positions
        E.g food = [[1,1], [1,0]] means the first food is positioned at [1,1], the second is at [1,0]. */
    public SnakeGame(int width, int height, int[][] food) {
        row = height;
        col = width;
        score = 0;
        this.food = food;
        myBody    = new HashSet<>();
        myBody.add(0 + "@" + 0);

        progress = 0;
        queue    = new ArrayDeque<>();
        queue.addLast(new int[]{0, 0});
    }

    /** Moves the snake.
        @param direction - 'U' = Up, 'L' = Left, 'R' = Right, 'D' = Down
        @return The game's score after the move. Return -1 if game over.
        Game over when snake crosses the screen boundary or bites its body. */
    public int move(String direction) {
        int[] curPos = queue.peekLast();

        int newX = curPos[0];
        int newY = curPos[1];
```

```
42
43          if(direction.equals("U")){
44              newX -= 1;
45          }else if(direction.equals("L")){
46              newY -= 1;
47          }else if(direction.equals("R")){
48              newY += 1;
49          }else if(direction.equals("D")){
50              newX += 1;
51          }
52
53          String key = newX + "@" + newY;
54          //System.out.println(newX + " ->" + newY);
55
56          if(!isInRange(newX, newY) || (myBody.contains(key) &&
    !key.equals(queue.peekFirst()[0] + "@" + queue.peekFirst()[1])))
57              return -1;
58
59
60
61          if(progress < food.length &&
62             food[progress][0] == newX && food[progress][1] == newY){
63              score++;
64              queue.addLast(new int[]{newX, newY});
65              progress++;
66          }else{
67              int[] temp = queue.removeFirst();
68              myBody.remove(temp[0] + "@" + temp[1]);
69              queue.addLast(new int[]{newX, newY});
70          }
71
72          myBody.add(key);
73          return score;
74      }
75
76      private boolean isInRange(int i, int j){
77          return i >= 0 && j >= 0 && i < row && j < col;
78      }
79  }
```

```
1   public class SnakeGame {
2       private int width;
3       private int height;
4       private int[][] food;
5       private int foodIndex;
```

```
    private ArrayDeque<int[]> queue;

    public SnakeGame(int width, int height, int[][] food) {
        this.width  = width;
        this.height = height;
        this.food   = food;
        foodIndex   = 0;
        queue       = new ArrayDeque<>();
        queue.addLast(new int[]{0, 0});
    }


    public int move(String direction) {
        int[] curPos = queue.peekFirst();

        int   curX   = curPos[0];   int curY = curPos[1];

        if      (direction.equals("R"))    curY += 1;
        else if (direction.equals("L"))    curY -= 1;
        else if (direction.equals("U"))    curX -= 1;
        else                               curX += 1;


        if(outOfRange(curX, curY) || isTouchBody(curX, curY))
            return -1;

        else if(foodIndex < food.length
            && curX == food[foodIndex][0] && curY == food[foodIndex][1])
        {
            foodIndex++;
            queue.addFirst(new int[]{curX, curY});
        }

        else
        {
            queue.removeLast();
            queue.addFirst(new int[]{curX, curY});
        }

        return queue.size() - 1;
    }

    private boolean outOfRange(int x, int y)
    {   return x < 0 || y < 0 || x >= height || y >= width;}

    private boolean isTouchBody(int x ,int y)
    {
        for(int[] pos : queue)
            if(x == pos[0] && y == pos[1])
```

```
                if(x != queue.peekLast()[0] || y != queue.peekLast()[1])
                    return true;
        return false;
    }
}
```

354 Russian Doll Envelopes 最长递增子序列扩展到二维情况

354. Russian Doll Envelopes

难度 困难 182

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]).
```

执行结果： 通过 显示详情

执行用时： 2022 ms , 在所有 Java 提交中击败了 5.03% 的用户

内存消耗： 41 MB , 在所有 Java 提交中击败了 5.20% 的用户

```
/*
  二刷，记忆化回溯
*/
HashMap<Integer, Integer> map = new HashMap<>();
    public int maxEnvelopes(int[][] envelopes) {
        Arrays.sort(envelopes, (o1, o2) -> (o1[0] == o2[0] ? o1[1] - o2[1] : o1[0]
- o2[0]));

```

```
        return backtrack(envelopes, 0, new ArrayList<>());
    }

    /*
        get the max res of envp from this point
    */
    private int backtrack(int[][] envelopes, int start, List<int[]> path){
        if(map.containsKey(start))
            return map.get(start);
        if(start == envelopes.length)
            return 0;

        int maxRes = 0;
        int pathSize = path.size();
        for(int i = start; i < envelopes.length; i++){
            int[] curEnv = envelopes[i];

            if(path.size() == 0 ||
               (path.get(path.size() - 1)[0] < curEnv[0] && path.get(path.size() - 1)[1] < curEnv[1])){

                path.add(curEnv);

                maxRes = Math.max(maxRes, backtrack(envelopes, i + 1, path) + 1);

                path.remove(path.size() - 1);
            }
        }

        map.put(start, maxRes);

        return maxRes;
    }
```

```
/*
  先对宽度w进行升序排列
    w如果相同，就按照高度h降序排序
    之后把所有的h作为一个数组，在这个数组上计算LIS的长度

  为什么是这样的呢?
  首先，按照宽度w，  也就是数组第一个元素进行排序
  使得宽度有序，那么下面就只需要关注高度是否递增就好

  转为  最长递增子序列的问题了(300)
```

```
11
12    而如果宽度由底到高，就会出现重复问题，比如
13    [3,6] [4,7],[4,8]
14
15    我们就会算两边
16
17    如果是从高到低
18    [3,6],[4,8],[4,7]
19  */
20  //by vk
21  public int maxEnvelopes(int[][] envelopes) {
22      if(envelopes.length == 0)        return 0;
23      Arrays.sort(envelopes, new Comparator<int[]>(){
24          @Override
25          public int compare(int[] o1, int[] o2)
26          { return o1[0] == o2[0] ? o2[1] - o1[1] : o1[0] - o2[0]; }
27      });
28
29      int[] dp = new int[envelopes.length];
30      int maxLen = 1;
31      Arrays.fill(dp, 1);
32      for(int i = 1; i < envelopes.length; i++)
33      {
34          for(int j = 0; j < i; j++)
35              if(envelopes[j][1] < envelopes[i][1])
36                  dp[i] = Math.max(dp[j] + 1, dp[i]);
37
38          maxLen = Math.max(maxLen, dp[i]);
39      }
40
41      return maxLen;
42  }
```

355 Design Twitter

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.
```

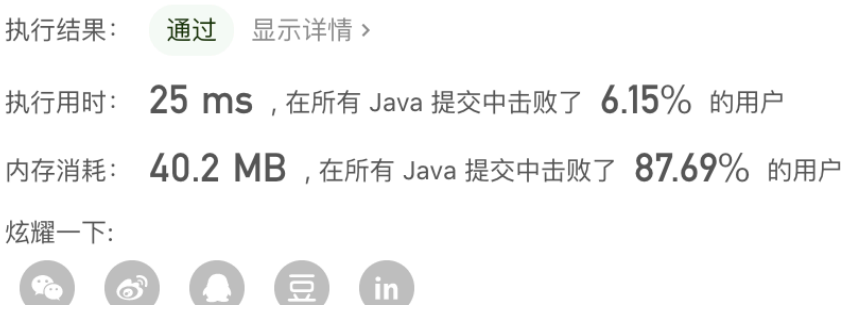
```
1  class Twitter {
2      private static int timestamp = 0;
3
4      private static class  Tweet
5      {
6          private int id;
7          private int time;
8          private Tweet next;
9
10         // 需要传入推文内容（id）和发文时间
11         public Tweet(int id, int time)
12         {
13             this.id = id;
14             this.time = time;
15             this.next = null;
16         }
17     }
18
19     private static class User
20     {
21         private int id;
22         public Set<Integer> followed;
```

```
        // 用户发表的推文链表头结点
        public Tweet head;

        public User(int userId) {
            followed = new HashSet<>();
            this.id = userId;
            this.head = null;
            // 关注一下自己
            follow(id);
        }

        public void follow(int userId) {
            followed.add(userId);
        }

        public void unfollow(int userId) {
            // 不可以取关自己
            if (userId != this.id)
                followed.remove(userId);
        }

        public void post(int tweetId) {
            Tweet twt = new Tweet(tweetId, timestamp);
            timestamp++;
            // 将新建的推文插入链表头
            // 越靠前的推文 time 值越大
            twt.next = head;
            head = twt;
        }
    }

    // 我们需要一个映射将 userId 和 User 对象对应起来
    private HashMap<Integer, User> userMap = new HashMap<>();

    /** user 发表一条 tweet 动态 */
    public void postTweet(int userId, int tweetId) {
        // 若 userId 不存在，则新建
        if (!userMap.containsKey(userId))
            userMap.put(userId, new User(userId));
        User u = userMap.get(userId);
        u.post(tweetId);
    }

    /** follower 关注 followee */
    public void follow(int followerId, int followeeId) {
        // 若 follower 不存在，则新建
    if(!userMap.containsKey(followerId)){
      User u = new User(followerId);
      userMap.put(followerId, u);
```

```
    }
        // 若 followee 不存在，则新建
    if(!userMap.containsKey(followeeId)){
      User u = new User(followeeId);
      userMap.put(followeeId, u);
    }
    userMap.get(followerId).follow(followeeId);
    }

    /** follower 取关 followee，如果 Id 不存在则什么都不做 */
    public void unfollow(int followerId, int followeeId) {
        if (userMap.containsKey(followerId)) {
            User flwer = userMap.get(followerId);
            flwer.unfollow(followeeId);
        }
    }

    /** 返回该 user 关注的人（包括他自己）最近的动态 id，
    最多 10 条，而且这些动态必须按从新到旧的时间线顺序排列。*/
    public List<Integer> getNewsFeed(int userId) {
        // 需要理解算法，见下文
    }
}

public List<Integer> getNewsFeed(int userId) {
    List<Integer> res = new ArrayList<>();
    if (!userMap.containsKey(userId)) return res;
    // 关注列表的用户 Id
    Set<Integer> users = userMap.get(userId).followed;
    // 自动通过 time 属性从大到小排序，容量为 users 的大小
    PriorityQueue<Tweet> pq =
        new PriorityQueue<>(users.size(), (a, b)->(b.time - a.time));

    // 先将所有链表头节点插入优先级队列
    for (int id : users) {
        Tweet twt = userMap.get(id).head;
        if (twt == null) continue;
        pq.add(twt);
    }

    while (!pq.isEmpty()) {
        // 最多返回 10 条就够了
        if (res.size() == 10) break;

        // 弹出 time 值最大的（最近发表的）
        Tweet twt = pq.poll();
        res.add(twt.id);

        // 将下一篇 Tweet 插入进行排序
```

```
            if (twt.next != null)
                pq.add(twt.next);
        }
        return res;
    }
```

356 Line Reflection 哈希表的应用




```
/*
    二刷 哈希表的应用
*/
public boolean isReflected(int[][] points) {
        int minX = Integer.MAX_VALUE;
        int maxX = Integer.MIN_VALUE;
        HashMap<String, Integer> map = new HashMap<>();

        for(int[] point : points){
            minX = Math.min(minX, point[0]);
            maxX = Math.max(maxX, point[0]);

            String symbol = point[0] + "@" + point[1];
            map.put(symbol, map.getOrDefault(symbol, 0) + 1);
        }

        double y_axis = (minX + maxX) / 2.0;

        int ops = 0;

        for(String str : map.keySet()){
            if(map.get(str) == 0)
                continue;

            String[] strs = str.split("@");
            int x = Integer.parseInt(strs[0]);
            int y = Integer.parseInt(strs[1]);

            int counterPartX = (int)(y_axis * 2) - x;
            int counterPartY = y;
```

```
31
32              String symbol = counterPartX + "@" + counterPartY;
33
34              if(x == y_axis && (symbol.equals(str) || map.getOrDefault(symbol, 0) ==
    0)){
35                  ops += map.get(str);
36              }else if(!map.containsKey(symbol)){
37                  return false;
38              }else{
39                  ops += map.get(str) + map.get(symbol);
40                  map.put(symbol, 0);
41                  map.put(str, 0);
42
43              }
44          }
45
46          return ops == points.length;
47      }
```

```
1   import java.util.HashSet;
2
3   class Solution {
4       public boolean isReflected(int[][] points) {
5           HashSet<String> set = new HashSet<>();
6           for(int[] point : points)
7                   set.add(point[0] + "@" + point[1]);
8
9           int[][] newPoints = new int[set.size()][2];
10          int index = 0;
11          for(String s : set)
12          {
13              String[] strs = s.split("@");
14              newPoints[index][0] = Integer.parseInt(strs[0]);
15              newPoints[index][1] = Integer.parseInt(strs[1]);
16              index++;
17          }
18
19          int minX = Integer.MAX_VALUE, maxX = Integer.MIN_VALUE;
20          for(int[] point : newPoints)
21          {
22              if(point[0] < minX)
23                  minX = point[0];
24              if(point[0] > maxX)
25                  maxX = point[0];
26          }
```

```
27
28          int mid = minX + maxX;
29          for(int[] point : newPoints)
30              if(!set.contains((mid - point[0]) + "@" + point[1]))
31                      return false;
32
33          return true;
34      }
35  }
```

357 Count Number with Unique Digits

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

Example:

```
Input: 2
Output: 91
Explanation: The answer should be the total numbers in
the range of 0 ≤ x < 100,
             excluding 11,22,33,44,55,66,77,88,99
```

执行结果： 通过 显示详情 >

执行用时： 0 ms , 在所有 Java 提交中击败了 100.00% 的用户

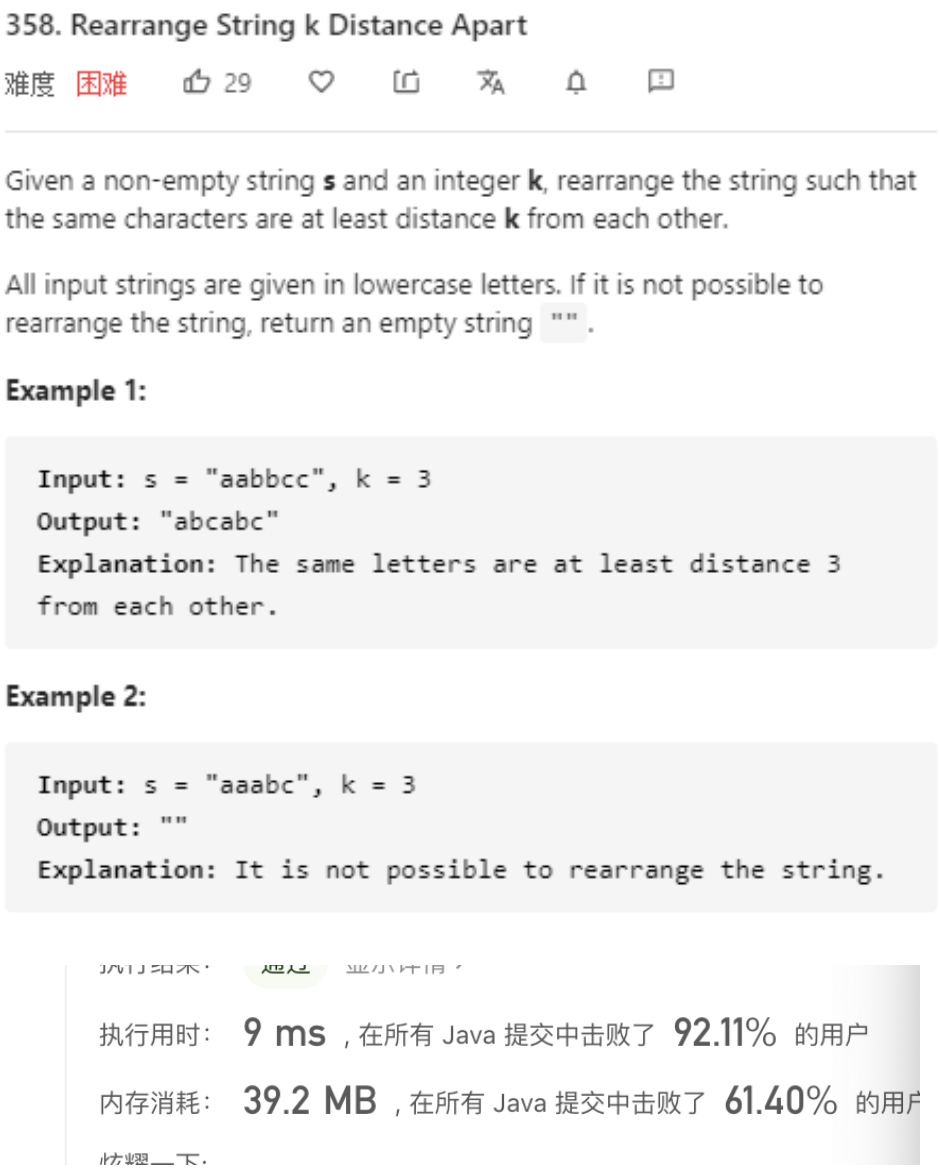
内存消耗： 36.4 MB , 在所有 Java 提交中击败了 41.33% 的用户

炫耀一下:

```
1   public int countNumbersWithUniqueDigits(int n) {
2       if(n == 0)      return 1;
3       int res = 10;
4
5       //bit for pos
6       for(int i = 2; i <= n; i++)
7       {
8           int count = 1;
9           for(int j = 1; j <= i; j++)
10          {
11              if(j >= 3)  count *= (11 - j);
12              else        count *= 9;
13          }
```

```
14
15          res += count;
16      }
17
18      return res;
19  }
```

358 Rearrange String k Distance Apart 贪心算法



358. Rearrange String k Distance Apart

难度 困难 29

Given a non-empty string **s** and an integer **k**, rearrange the string such that the same characters are at least distance **k** from each other.

All input strings are given in lowercase letters. If it is not possible to rearrange the string, return an empty string `""`.

Example 1:

```
Input: s = "aabbcc", k = 3
Output: "abcabc"
Explanation: The same letters are at least distance 3
from each other.
```

Example 2:

```
Input: s = "aaabc", k = 3
Output: ""
Explanation: It is not possible to rearrange the string.
```

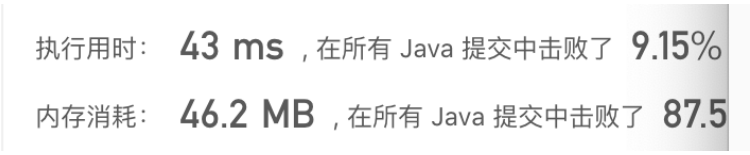
执行用时： 9 ms , 在所有 Java 提交中击败了 92.11% 的用户

内存消耗： 39.2 MB , 在所有 Java 提交中击败了 61.40% 的用户

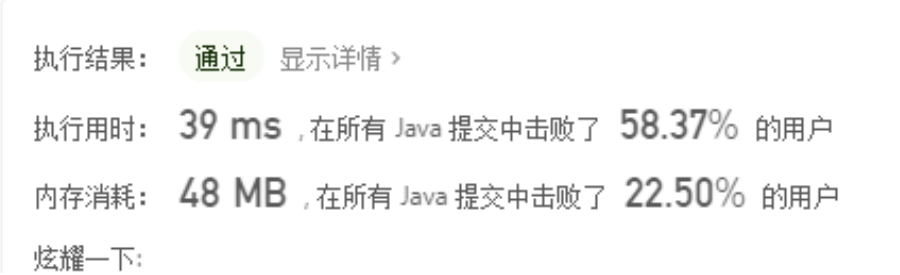
```
1   /*
2   直觉：    尽量把高频的放在前面出现
3   */
4       public String rearrangeString(String s, int k) {
```

```
        if(k <= 1)
            return s;

        int[] map = new int[26];
        PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> (a[1] == b[1] ? a[0] - b[0] : b[1] - a[1]));

        for(char ch : s.toCharArray())
            map[ch - 'a']++;
        for(int i = 0 ; i < 26; i++) {
            if(map[i] != 0)
            pq.add(new int[]{i, map[i]});
        }

      //主要描述 sb 中已有元素
        Deque<Character> queue = new ArrayDeque<>();
        StringBuilder sb = new StringBuilder();

        while(!pq.isEmpty()){
            int[] curAlpha = pq.poll();

            char ch = (char)(curAlpha[0] + 'a');
            queue.addLast(ch);
            sb.append(ch);
            map[ch - 'a']--;

            if(queue.size() == k){
                char curChar = queue.pollFirst();
                if(map[curChar - 'a'] > 0){
                    pq.add(new int[]{curChar - 'a', map[curChar - 'a']});
                }
            }
        }

        return sb.length() == s.length() ? sb.toString() : "";
    }
```

359 Logger Rate Limiter




```java
//二刷，没啥好说的
class Logger {
    HashMap<String, Integer> map;
    /** Initialize your data structure here. */
    public Logger() {
        map        = new HashMap<>();
    }

    /** Returns true if the message should be printed in the given timestamp, otherwise returns false.
        If this method returns false, the message will not be printed.
        The timestamp is in seconds granularity. */
    public boolean shouldPrintMessage(int timestamp, String message) {
        if(!map.containsKey(message)){
            map.put(message, timestamp);
            return true;
        }else{
            int before = map.get(message);

            if(timestamp - before < 10){
                return false;
            }else{
                map.put(message, timestamp);
                return true;
            }
        }
    }
}

```




```java
class Logger {
    private HashMap<String, Integer> map;
    /** Initialize your data structure here. */
```

```
    public Logger() {
        map = new HashMap<>();
    }

    /** Returns true if the message should be printed in the given timestamp, otherwise returns false.
        If this method returns false, the message will not be printed.
        The timestamp is in seconds granularity. */
    public boolean shouldPrintMessage(int timestamp, String message) {
        if(!map.containsKey(message))
        {
            map.put(message, timestamp);
            return true;
        }
        else
        {
            int lastTime = map.get(message);
            if(timestamp - lastTime < 10)  return false;

            map.put(message, timestamp);
            return true;
        }
    }
}

/**
 * Your Logger object will be instantiated and called as such:
 * Logger obj = new Logger();
 * boolean param_1 = obj.shouldPrintMessage(timestamp,message);
 */
```

360 Sort Transformed Array 数学知识 + 双指针

Given a **sorted** array of integers *nums* and integer values *a*, *b* and *c*. Apply a quadratic function of the form $f(x) = ax^2 + bx + c$ to each element *x* in the array.

The returned array must be in **sorted order**.

Expected time complexity: **O(*n*)**

Example 1:

```
Input: nums = [-4,-2,2,4], a = 1, b = 3, c = 5
Output: [3,9,15,33]
```

Example 2:

```
Input: nums = [-4,-2,2,4], a = -1, b = 3, c = 5
Output: [-23,-5,1,7]
```

```java
public int[] sortTransformedArray(int[] nums, int a, int b, int c) {
    if(nums.length == 0)        return new int[]{};
    int[] res = new int[nums.length];

    if(a == 0)
    {
        int i = b > 0 ? 0 : nums.length - 1;
        for(int num : nums)
        {
            res[i] = num * b + c;
            i = b > 0 ? i + 1 : i - 1;
        }

        return res;
    }

    double mid = -b * 1.0 / (2 * a);
    int left = 0, right = nums.length - 1;

    int i = a < 0 ? 0 : nums.length - 1;
    while(left <= right)
    {
        double ld = Math.abs(mid - nums[left]);
        double rd = Math.abs(nums[right] - mid);

        if(ld >= rd)
        {
            res[i] = a * nums[left] * nums[left] + b * nums[left] + c;
            i = a < 0 ? i + 1 : i-1;
```

```
            left++;
        }
        else
        {
            res[i] = a * nums[right] * nums[right] + b * nums[right] + c;
            i = a < 0 ? i + 1 : i - 1;
            right--;
        }
    }

    return res;
}
```

361 Bomb Enemy 暴力解

Given a 2D grid, each cell is either a wall `'W'`, an enemy `'E'` or empty `'0'` (the number zero), return the maximum enemies you can kill using one bomb.
The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since the wall is too strong to be destroyed.
Note: You can only put the bomb at an empty cell.

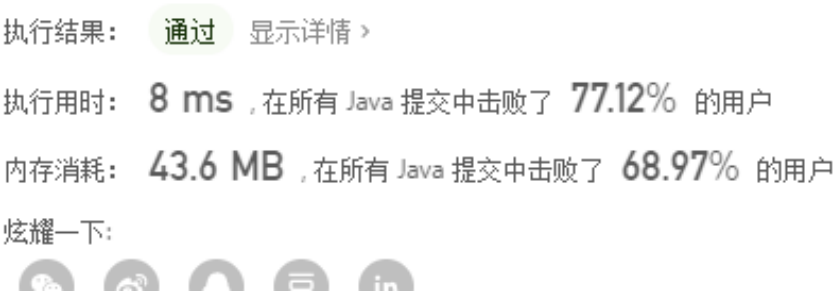
Example:

```
Input: [["0","E","0","0"],["E","0","W","E"],
["0","E","0","0"]]
Output: 3
Explanation: For the given grid,

0 E 0 0
E 0 W E
0 E 0 0

Placing a bomb at (1,1) kills 3 enemies.
```

```
/*
  E
*/
int row = 0;
    int col = 0;

    public int maxKilledEnemies(char[][] grid) {
        row = grid.length;
```

```
        col = row == 0 ? 0 : grid[0].length;
        if(col == 0)
            return 0;

        int[][] dpUp    = new int[row][col];
        int[][] dpLeft  = new int[row][col];

        int[][] dpDown  = new int[row][col];
        int[][] dpRight = new int[row][col];
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(grid[i][j] == 'W'){
                    dpUp[i][j] = 0;
                    dpLeft[i][j] = 0;
                }else{
                    if(grid[i][j] == 'E') {
                        dpLeft[i][j] = 1;
                        dpUp[i][j]   = 1;
                    }

                    if(isInRange(i - 1, j))
                        dpUp[i][j] += dpUp[i - 1][j];
                    if(isInRange(i, j - 1))
                        dpLeft[i][j] += dpLeft[i][j - 1];
                }
            }
        }

        for(int i = row - 1; i >= 0; i--){
            for(int j = col - 1; j >= 0; j--){
                if(grid[i][j] == 'W'){
                    dpDown[i][j] = 0;
                    dpRight[i][j] = 0;
                }else{
                    if(grid[i][j] == 'E'){
                        dpDown[i][j] = 1;
                        dpRight[i][j] = 1;
                    }

                    if(isInRange(i + 1, j))
                        dpDown[i][j] += dpDown[i + 1][j];
                    if(isInRange(i, j + 1))
                        dpRight[i][j] += dpRight[i][j + 1];
                }
            }
        }

        int res = 0;
        for(int i = 0; i < row; i++){
```

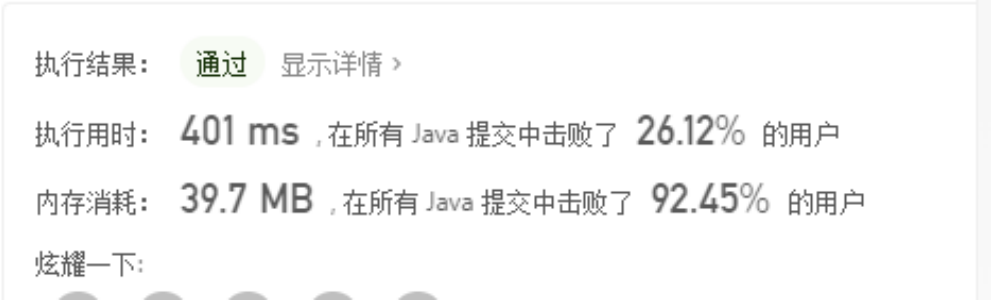
```
            for(int j = 0; j < col; j++){
                if(grid[i][j] == '0'){
                    res = Math.max(res, dpUp[i][j] + dpDown[i][j] + dpLeft[i][j] +
dpRight[i][j]);
                }
            }
        }

        return res;
    }

    private boolean isInRange(int i, int j){
        return i >= 0 && j >= 0 && i < row && j < col;
    }
```




```
/*
  暴力解法
*/
class Solution {
    public int maxKilledEnemies(char[][] grid) {
        int max = 0, row = grid.length, column = row == 0 ? 0 : grid[0].length;

        for(int i = 0; i < row; i++)
            for(int j = 0; j < column; j++)
                if(grid[i][j] == '0')
                {
                    int count = 0;
                    int x = i, y =j;

                    while(x >= 0)
                    {
                        if(grid[x][j] == 'W')
                            break;
                        if(grid[x][j] == 'E')
                            count++;
                        x--;
                    }
                    x = i;
```

```
                    while(x < row)
                    {
                        if(grid[x][j] == 'W')
                            break;
                        if(grid[x][j] == 'E')
                            count++;
                        x++;
                    }
                    while(y < column)
                    {
                        if(grid[i][y] == 'W')
                            break;
                        if(grid[i][y] == 'E')
                            count++;
                        y++;
                    }
                    y = j;
                    while(y >= 0)
                    {
                        if(grid[i][y] == 'W')
                            break;
                        if(grid[i][y] == 'E')
                            count++;
                        y--;
                    }

                    max = Math.max(max, count);
                }
        return max;
    }
}
```

363 Max Sum of Rectangle No Larger Than K




```
/*
  思路就是预处理，  DP方案
*/
```

```java
public int maxSumSubmatrix(int[][] matrix, int k) {
    int row = matrix.length; int column = row == 0 ? 0 : matrix[0].length;
    if(column == 0)     return 0;

    //DESIGN a new Matrix to pre-calculate
    int[][] expandMatrix = new int[row + 1][column + 1];

    for(int i = 1; i < row + 1; i++)
        expandMatrix[i][1] = matrix[i-1][0] + expandMatrix[i-1][1];
    for(int j = 1; j < column + 1; j++)
        expandMatrix[1][j] = matrix[0][j-1] + expandMatrix[1][j-1];

    for(int i = 2; i < row + 1; i++)
        for(int j = 2; j < column + 1; j++)
            expandMatrix[i][j] = expandMatrix[i-1][j]
            + expandMatrix[i][j-1] - expandMatrix[i-1][j-1] + matrix[i-1][j-1];

    //对任意两点进行查找
    int res = Integer.MIN_VALUE;
    for(int i = 1; i < row + 1; i++)
        for(int j = 1; j < column + 1; j++)
            for(int m = i; m < row + 1; m++)
                for(int n = j; n < column + 1; n++)
                {
                    int testExpr = expandMatrix[m][n] - expandMatrix[m][j-1]
                        -  expandMatrix[i-1][n] + expandMatrix[i-1][j-1];
                    if(testExpr <= k)
                        res = Math.max(res, testExpr);
                }
    return res;
}
```

```java
/*
An improved version takes O(n^3logn). It borrows the idea to find max subarray with sum <= k in 1D array, and apply here: we find all rectangles bounded between r1 & r2, with columns from 0 to end. Pick a pair from tree.
*/
public int maxSumSubmatrix(int[][] matrix, int k) {
    if (matrix == null || matrix.length == 0 || matrix[0].length == 0)
        return 0;
    int rows = matrix.length, cols = matrix[0].length;
    int[][] areas = new int[rows][cols];

    //to calculate(0,0) -> (r,c)'s sum area
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            int area = matrix[r][c];
```

```
            if (r >= 1)       area += areas[r-1][c];
            if (c >= 1)       area += areas[r][c-1];
            if (r >= 1 && c >= 1)  area -= areas[r-1][c-1];

            areas[r][c] = area;
        }
    }

    int max = Integer.MIN_VALUE;
    for (int r1 = 0; r1 < rows; r1++) {
        for (int r2 = r1; r2 < rows; r2++) {
            TreeSet<Integer> tree = new TreeSet<>();
            tree.add(0);    // padding

            for (int c = 0; c < cols; c++)
            {
                int area = areas[r2][c];
                if (r1 >= 1)
                    area -= areas[r1-1][c];
                //ceiling(E e) 方法返回在这个集合中大于或者等于给定元素的最小元素
                Integer ceiling = tree.ceiling(area - k);
                if (ceiling != null)
                    max = Math.max(max, area - ceiling);
                tree.add(area);
            }
        }
    }
    return max;
}
//ref: https://leetcode.com/problems/max-sum-of-rectangle-no-larger-than-k/discuss/83618/2-Accepted-Java-Solution
```

364 Nested List Weight Sum II

执行结果: 通过 显示详情 >

执行用时: 1 ms , 在所有 Java 提交中击败了 97.53% 的用户

内存消耗: 36 MB , 在所有 Java 提交中击败了 28.39% 的用户

炫耀一下:

```
//二刷 遍历两次
class Solution {
    int maxDepth = 1;
    int res = 0;
    public int depthSumInverse(List<NestedInteger> nestedList) {
        int depth = 1;
```

```
        dfs(nestedList, depth);

        //System.out.println(maxDepth);
        getRes(nestedList, 1);
        return res;
    }

    private void getRes(List<NestedInteger> nestedList, int level){
        for(NestedInteger ni : nestedList){
            if(ni.isInteger())
                res += ni.getInteger() * (maxDepth - level + 1);
            else
                getRes(ni.getList(), level + 1);
        }
    }

    private void dfs(List<NestedInteger> nestedList, int depth){
        maxDepth = Math.max(depth, maxDepth);
        if(nestedList == null){
            return;
        }

        for(NestedInteger ni : nestedList){
            if(!ni.isInteger()){
                dfs(ni.getList(), depth + 1);
            }
        }
    }
}
```


```
  //     weighted,    Integer
HashMap<Integer, List<Integer>> map = new HashMap<>();
int maxLevel = 0;
public int depthSumInverse(List<NestedInteger> nestedList) {
    depthSumInverse(nestedList, 1);

    int res = 0;
    for(int i = maxLevel; i >= 1; i--)
    {
```

```
        if(!map.containsKey(i))     continue;
        for(Integer num : map.get(i))
            res += num * (maxLevel - i + 1);
    }
    return res;
}

private void  depthSumInverse(List<NestedInteger> nestedList, int level){
    if(nestedList.size() == 0)      return;
    maxLevel = Math.max(level, maxLevel);
    for(NestedInteger ni : nestedList)
    {
        if(ni.isInteger())
        {
            if(!map.containsKey(level))
                map.put(level, new ArrayList<>());
            map.get(level).add(ni.getInteger());
        }

        else
            depthSumInverse(ni.getList(), level + 1);
    }

```

365 Water and Jug Problem 脑经急转弯

365. Water and Jug Problem

难度 中等 249

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
```

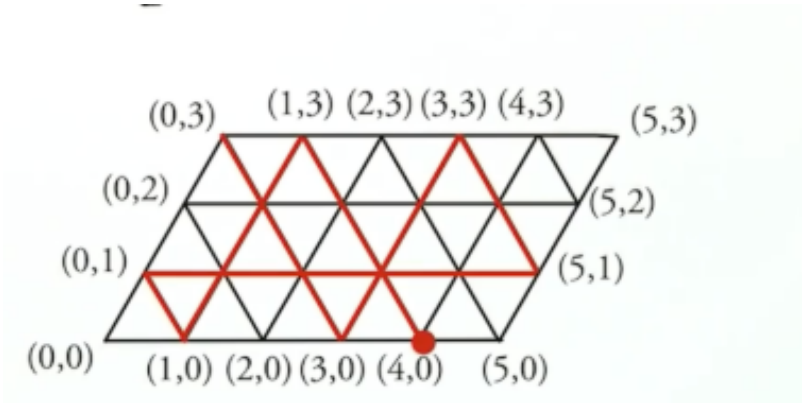
Example 2:

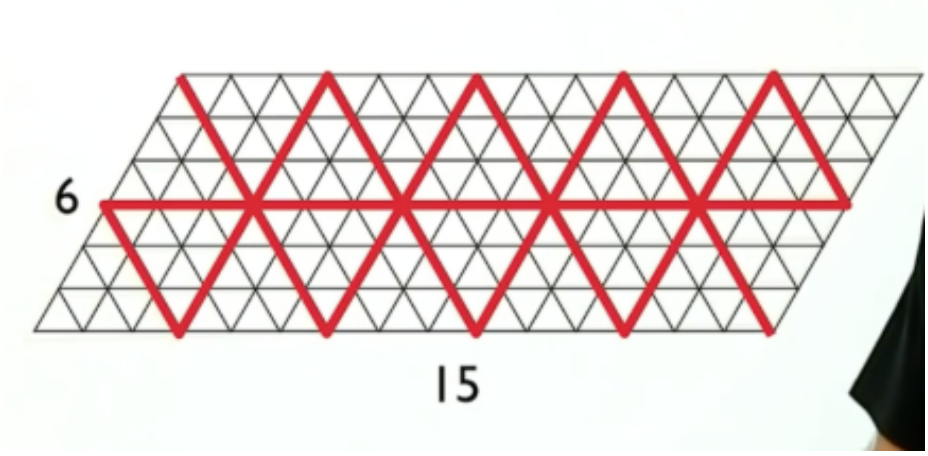
```
Input: x = 2, y = 6, z = 5
Output: False
```

```java
/*
    二刷 正经 BFS 方法
    ref : https://leetcode-cn.com/problems/water-and-jug-problem/solution/javade-bfsxie-fa-by-sweetiee/
    思路
      1,  不可能存在两个都半满的情况
          如果某一个水壶是半满的,  那么另外一个肯定是满的 / 空的

      2. 如果某个水壶半满，那么就不能直接倒掉，因为这样会回到原始状态
*/
import java.util.*;
class Solution {
    public boolean canMeasureWater(int x, int y, int z) {
        if(z == 0)
            return true;
        else if(x + y < z)
            return false;

        Deque<Node> queue = new ArrayDeque<>();
        Set<Node> visited = new HashSet<>();
        Node start = new Node(0, 0);
```

```
        queue.add(start);
        visited.add(start);

        while(!queue.isEmpty()){
            Node cur = queue.removeFirst();
            int curX = cur.x;
            int curY = cur.y;

            if(curX == z || curY == z || curX + curY == z){
                return true;
            }

            if(curX == 0){
                add(queue, visited, new Node(x, curY));
            }

            if(curY == 0){
                add(queue, visited, new Node(curX, y));
            }

            if(curY < y){
                add(queue, visited, new Node(0, curY));
            }

            if(curX < x){
                add(queue, visited, new Node(curX, 0));
            }

            int moveSize = Math.min(curX, y - curY);
            add(queue, visited, new Node(curX - moveSize, curY + moveSize));
            moveSize     = Math.min(curY, x - curX);
            add(queue, visited, new Node(curX + moveSize, curY - moveSize));
        }

        return false;
    }

    private void add(Deque<Node> queue, Set<Node> visited, Node node){
        if(!visited.contains(node)){
            queue.addLast(node);
            visited.add(node);
        }
    }
}

class Node{
    public int x;
    public int y;
    public Node(int x, int y){
```

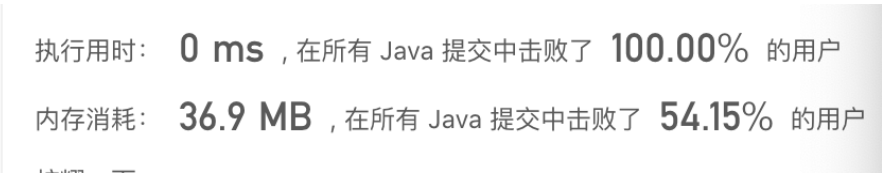
```
70          this.x = x;
71          this.y = y;
72      }
73
74      @Override
75      public boolean equals(Object o) {
76          if (this == o) return true;
77          if (o == null || getClass() != o.getClass()) return false;
78          Node node = (Node) o;
79          return x == node.x && y == node.y;
80      }
81
82      @Override
83      public int hashCode() {
84          return toString().hashCode();
85      }
86
87      @Override
88      public String toString(){
89          return x + "@" + y;
90      }
91  }
```





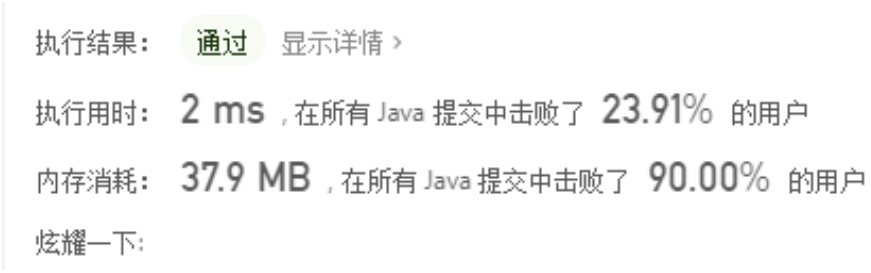
```java
//https://www.youtube.com/watch?v=0Oef3MHYEC0
public boolean canMeasureWater(int x, int y, int z) {
    if (z == 0) return true;
    if (x + y < z) return false;

    int big = Math.max(x, y);
    int small = x + y - big;

    if (small == 0) return big == z;


    while (big % small != 0) {
        int temp = small;
        small = big % small;
        big = temp;
    }
    return z % small == 0;
}

作者：antonzhao
链接：https://leetcode-cn.com/problems/water-and-jug-problem/solution/hu-dan-long-wei-liang-zhang-you-yi-si-de-tu-by-ant/
```

366 Find Leaves of Binary Tree




```java
//二刷
class Solution {

    public List<List<Integer>> findLeaves(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        while(true){
            List<Integer> path = new ArrayList<>();
            root = dfs(root, path);
            res.add(path);
            if(root == null)
                break;
        }

        return res;
```

```
    }

    private TreeNode dfs(TreeNode root, List<Integer> path){
        if(root == null)
            return null;;

        if(root.left == null && root.right == null){
            path.add(root.val);
            return null;
        }

        root.left = dfs(root.left, path);
        root.right = dfs(root.right, path);
        return root;
    }
}
```



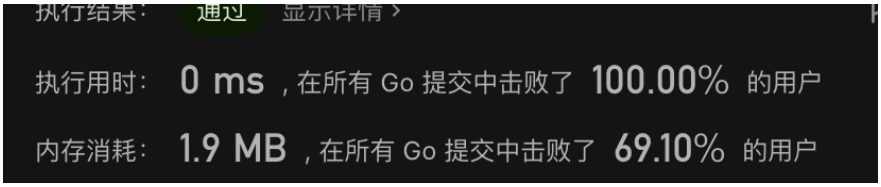

```
    public List<List<Integer>> findLeaves(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if(root == null)        return res;

        List<TreeNode> path = BFS(root);
        HashSet<TreeNode> visited = new HashSet<>();
        int count = 0;
        while(count != path.size())
        {
            List<Integer> oneLevel = new ArrayList<>();
            for(TreeNode tn : path)
                if(isLeaves(tn) && !visited.contains(tn))
                {
                    oneLevel.add(tn.val);
                    count++;
                    tn.val = Integer.MIN_VALUE;
                    visited.add(tn);
                }
            res.add(oneLevel);
        }

```

```
        return res;
    }

    private boolean isLeaves(TreeNode tn)
    {
        if(tn.left == null && tn.right == null)
return true;
        if(tn.left == null && tn.right != null && tn.right.val ==
Integer.MIN_VALUE)                                  return true;
        if(tn.left != null && tn.left.val == Integer.MIN_VALUE && tn.right == null)
                                  return true;
        if(tn.left != null && tn.right != null && tn.left.val == Integer.MIN_VALUE
&& tn.right.val == Integer.MIN_VALUE)
            return true;

        return false;
    }

    private List<TreeNode> BFS(TreeNode root)
    {
        List<TreeNode> path = new ArrayList<>();
        if(root == null)        return path;
        Deque<TreeNode> queue = new ArrayDeque<>();
        queue.offer(root);

        while(!queue.isEmpty())
        {
            int size = queue.size();
            for(int i = 0; i < size; i++)
            {
                TreeNode cur = queue.poll();
                if(cur.left != null)
                    queue.add(cur.left);

                if(cur.right != null)
                    queue.add(cur.right);

                path.add(cur);
            }

        }
        return path;
    }
```

```
class Solution {
    public List<List<Integer>> findLeaves(TreeNode root) {
        List<List<Integer>> resList = new ArrayList<>();
```

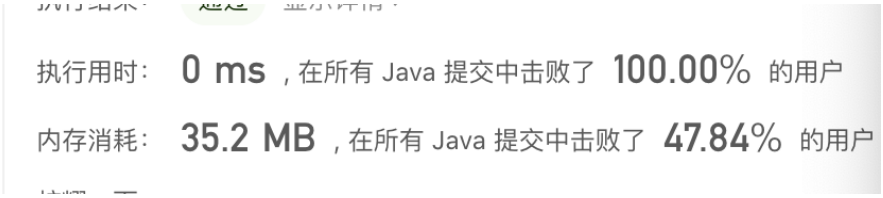
```
        while (root != null) {
            List list = new ArrayList<>();
            root = recur(root, list);
            resList.add(list);
        }
        return resList;
    }

    /*
     * 如果 root 是叶子节点，则把它装入到 list 中，并且给上一级返回 null，表示自己被删除
     * 如果 root 不是叶子节点，则递归 root 的左子节点和右子节点，并返回 root 给上一级，表明自己没有被删除
     */
    private TreeNode recur(TreeNode root, List<Integer> list) {
        if (root == null)
            return null;

        if (root.left == null && root.right == null) {
            list.add(root.val);
            return null;
        }
        root.left = recur(root.left, list);
        root.right = recur(root.right, list);
        return root;
    }
}

作者：klb
链接：https://leetcode-cn.com/problems/find-leaves-of-binary-tree/solution/366-xun-zhao-er-cha-shu-de-xie-zi-jie-dian-by-klb/
```

367 valid perfect square




```
func isPerfectSquare(num int) bool {
  if num <= 4{
    return num == 1 || num == 4
  }

  left, right := 1, num

  for ;left <= right;{
```

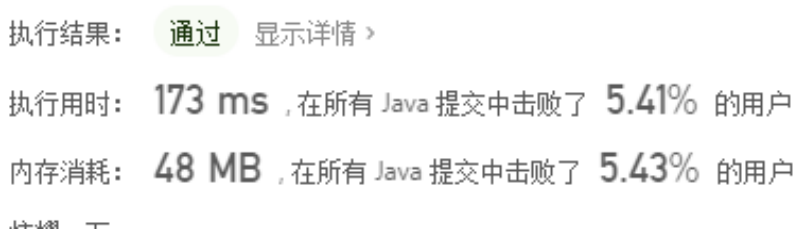
```
    mid := (left + right) / 2
    if mid * mid == num{
      return true
    }else if mid * mid > num{
      right = mid - 1
    }else{
      left = mid + 1
    }
  }

  return false
}

```




```
//继续二刷
class Solution {
    public boolean isPerfectSquare(int num) {
        int left = 1, right = 46340;

        while(left <= right){
            int mid = (left + right) / 2;

            int res = mid * mid;
            if(res== num)
                return true;
            else if(res < num)
                left = mid + 1;
            else
                right = mid - 1;
        }

        return false;
    }
}
```

```
//二刷 暴力求解
class Solution {
    public static Set<Integer> set = new HashSet<>();
```

```
    static{
        for(int i = 1; i <= Integer.MAX_VALUE; i++){
            int ii = i * i;
            if(ii < 0)
                break;

            set.add(ii);
        }
    }

    public boolean isPerfectSquare(int num) {
        return set.contains(num);
    }
}
```




```
public boolean isPerfectSquare(int num) {
    //46340
    HashSet<Integer> set = new HashSet<>();
    for(int i = 1; i <= 46340; i++)
        set.add(i * i);
    return set.contains(num);
}
```

```java
public boolean isPerfectSquare(int num) {
    long left = 1, right = num / 2;
    while(left < right)
    {
        long mid = left + (right - left) / 2;
        // System.out.println("left " + left + "right " + right);
        if(mid * mid == (long)num)        return true;
        else if(mid * mid > num)    right = mid - 1;
        else                        left = mid + 1;
    }

    return left * left == num;
}
```

368 Largest Divisiable Subset 不错的DP 题目

```java
//二刷 采用 dp
class Solution {
    public List<Integer> largestDivisibleSubset(int[] nums) {
        List<Integer>[] dp = new ArrayList[nums.length + 1];

        Arrays.sort(nums);
        List<Integer> res = null;
        for(int i = 1; i < dp.length; i++){
            dp[i] = new ArrayList<>();
            int sumMax = -1;
            int index = -1;
            for(int j = 1; j < i; j++){
                if(nums[i - 1] % nums[j - 1] == 0 && dp[j].size() != 0 &&
dp[j].size() > sumMax){
                    sumMax = dp[j].size();
                    index = j;
                }
            }

            if(index != -1)
                dp[i].addAll(dp[index]);

            dp[i].add(nums[i - 1]);

            if(res == null || dp[i].size() > res.size())
                res = dp[i];
```

```
        }

        return res;
    }
}s
```

```
//二刷采用回溯，  超时
//44 / 45
//2 ^ n
class Solution {
    List<Integer> res;
    public List<Integer> largestDivisibleSubset(int[] nums) {
        res = new ArrayList<>();
        Arrays.sort(nums);

        backtrack(nums, 0, new ArrayList<>());
        return res;
    }

    public void backtrack(int[] nums, int start, List<Integer> path){
        if(start == nums.length)
            return;

        for(int i = start; i < nums.length; i++){
            if(isAddable(path, nums[i])){
                path.add(nums[i]);
                if(res.size() < path.size())
                    res = new ArrayList<>(path);

                backtrack(nums, i + 1, path);

                path.remove(path.size() - 1);
            }
        }
    }

    private boolean isAddable(List<Integer> path, int num)  {
        if(path.size() == 0)
            return true;

        for(Integer in : path){
            if(num % in != 0)
                return false;
        }

        return true;
    }
```

```
}
```

368. Largest Divisible Subset

难度 中等 149

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 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]
```

执行结果: 通过 显示详情 >

执行用时: 22 ms , 在所有 Java 提交中击败了 69.69% 的用户

内存消耗: 38.9 MB , 在所有 Java 提交中击败了 72.23% 的用户

炫耀一下:

```java
/*
    思路:  在有序数组中,  比如[1,2,4,8,16,32]

    如果拿到了 数字8的结果,  那么只要能被 8 整除的,  比如 16,  32 都可以将 8 的结果集添加到自己的结果集

    注意去重操作,  和拿到最大的个数

*/
class Solution {
    public List<Integer> largestDivisibleSubset(int[] nums) {
        if(nums.length == 0)            return new ArrayList<>();

        List<Integer>[] dp = new ArrayList[nums.length];
        Arrays.sort(nums);
        int maxNum = 0;
```

```
16
17          for(int i = 0; i < nums.length; i++){
18            //创建自己的结果集
19              dp[i] = new ArrayList<>();
20              dp[i].add(nums[i]);
21
22              int index = -1;
23              int subMax = -1;
24            //从索引0-j的结果集中寻找自己想要的
25              for(int j = i - 1; j >= 0; j--){
26                  if(nums[i] % nums[j] == 0){
27                     if(subMax < dp[j].size()){
28                         subMax = dp[j].size();
29                         index  = j;
30                     }
31                  }
32              }
33              //如果找到了，  就添加到自己
34              if(index != -1)
35                  for(Integer num : dp[index])
36                      dp[i].add(num);
37
38              maxNum = Math.max(maxNum, dp[i].size());
39          }
40
41          for(int i = 0; i < nums.length; i++){
42              if(dp[i].size() == maxNum)
43                  return dp[i];
44          }
45
46          return null;
47      }
48  }
```

369 Plus One List

执行用时： 0 ms ，在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 36.3 MB ，在所有 Java 提交中击败了 10.35% 的用户

```
1  /*
2      二刷 递归 方法
3  */
4  class Solution {
```

```
    int carryBit = 0;
    public ListNode plusOne(ListNode head) {
        head = plusOneHelper(head);

        if(carryBit == 1){
            ListNode newHead = new ListNode(1);
            newHead.next = head;
            return newHead;
        }else{
            return head;
        }
    }

    private ListNode plusOneHelper(ListNode head){
        if(head == null)
            return head;

        if(head.next == null){
            head.val += 1;
            if(head.val >= 10){
                head.val -= 10;
                carryBit = 1;
            }

            return head;
        }

        head.next = plusOneHelper(head.next);
        head.val += carryBit;
        carryBit = 0;

        if(head.val >= 10){
            head.val -= 10;
            carryBit = 1;
        }

        return head;
    }

}
```

执行结果： 通过 显示详情 >

执行用时： 1 ms ，在所有 Java 提交中击败了 24.35% 的用户

内存消耗： 36.6 MB ，在所有 Java 提交中击败了 67.50% 的用户

炫耀一下:

```java
    public ListNode plusOne(ListNode head) {
        if(head == null)        return head;

        Deque<ListNode> stack = new ArrayDeque<>();
        while(head != null){
            stack.push(head);
            head = head.next;
        }

        stack.peek().val += 1;

        int carryFlag = 0;
        ListNode cur = null;
        while(!stack.isEmpty()){
            cur = stack.pop();
            cur.val += carryFlag;
            if(cur.val >= 10){
                cur.val -= 10;
                carryFlag = 1;
            }else{
                carryFlag = 0;
            }
        }

        if(carryFlag == 1){
            ListNode newHead = new ListNode(1);
            newHead.next = cur;
            return newHead;
        }

        return cur;
    }
```

```java
/*
    值得学习的方法的，  给链表加个头节点

    算法核心是  找到最后边不是  9的数字，  给它加一，  然后该节点右边其他的全部置0
```

```
*/
class Solution {
  public ListNode plusOne(ListNode head) {
    // sentinel head
    ListNode sentinel = new ListNode(0);
    sentinel.next = head;
    ListNode notNine = sentinel;

    // find the rightmost not-nine digit
    while (head != null) {
      if (head.val != 9) notNine = head;
      head = head.next;
    }

    // increase this rightmost not-nine digit by 1
    notNine.val++;
    notNine = notNine.next;

    // set all the following nines to zeros
    while (notNine != null) {
      notNine.val = 0;
      notNine = notNine.next;
    }

    return sentinel.val != 0 ? sentinel : sentinel.next;
  }
}

作者：LeetCode
链接：https://leetcode-cn.com/problems/plus-one-linked-list/solution/gei-dan-lian-biao-jia-yi-by-leetcode/
```

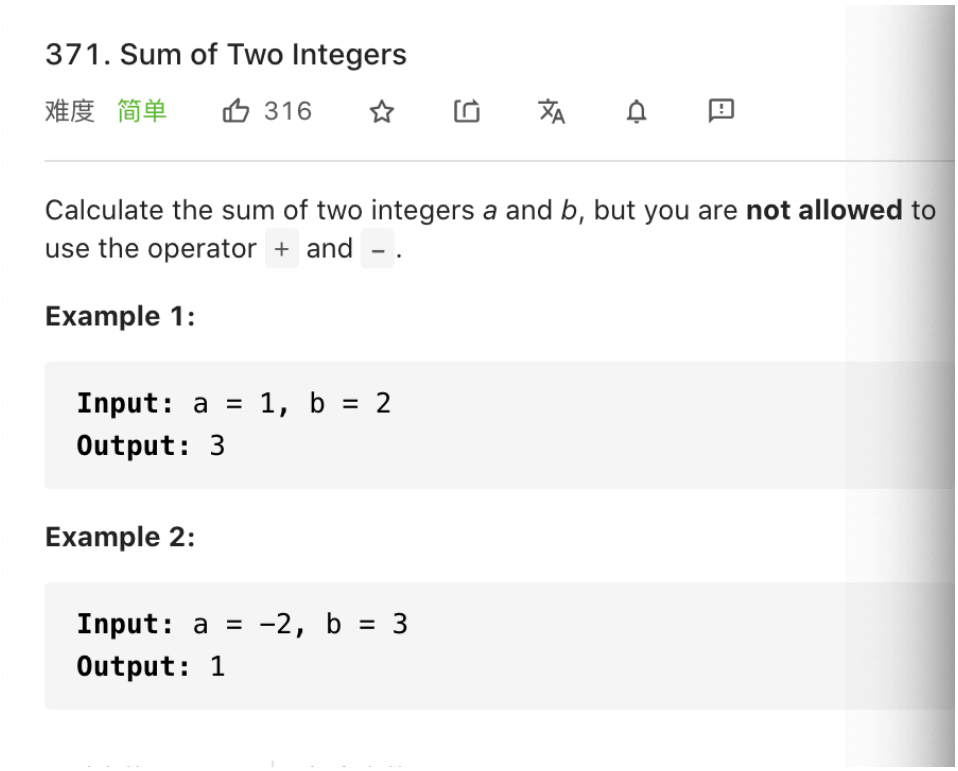
370 Region Addition 挺取巧的一个题

```java
/*
    简单粗暴的方法
*/
public int[] getModifiedArray(int length, int[][] updates) {
  int[] res = new int[length];

  for(int i = 0; i < updates.length; i++){
    for(int j = updates[i][0]; j <= updates[i][1]; j++)
      res[j] += updates[i][2];
  }

  return res;
}
```

```java
/*
    很简洁的方法，  算法是这样的
    updata 包含 startIndex, endIndex, val
    那么我在 [stratIndex, length -1] + val;
    [endIndex, length - 1] - val 即可

    算法表现为
    ans[startIndex] += val;
    ans[endIndex + 1] -= val;

    然后对
    [startIndex, length - 1] 遍历进行ans[i] += ans[i-1]
*/
class Solution {
   public int[] getModifiedArray(int length, int[][] updates) {
        int[] ans = new int[length];
        int start, end, val;
        for (int[] update : updates) {
            start = update[0];
            end = update[1];
            val = update[2];
            ans[start] += val;
            if (end < length - 1) {
                ans[end + 1] -= val;
            }
        }
        for (int i = 1; i < length; i++) {
            ans[i] += ans[i - 1];
        }
        return ans;
    }
```

```
}

作者：klb
链接：https://leetcode-cn.com/problems/range-addition/solution/370-qu-jian-jia-fa-by-klb/
```

371 Sum or Two Integers



371. Sum of Two Integers

难度 简单 316

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
```

```java
/*
  这个主要是当进位（b） == 0 的时候， 停止加法
  原理很简单， 就是550 上课讲的 XOR + AND
*/
public int getSum(int a, int b) {
        while(b != 0){
            int temp = (a ^ b);
            b = (a & b) << 1;
            a = temp;
        }
        return a;
    }
```

372 Super Power

372. Super Pow

难度 中等 84

Your task is to calculate a^b mod `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
```

Example 3:

```
Input: a = 1, b = [4,3,3,8,5,2]
Output: 1
```

Example 4:

```
Input: a = 2147483647, b = [2,0,0]
Output: 1198
```

题目... ... 上一题 372/1822 下一题

```java
int base = 1337;
public int superPow(int a, int[] b) {
    Deque<Integer> queue = new ArrayDeque<>();

    for(int num : b)
        queue.addLast(num);
    return superPow(a, queue);
}

private int superPow(int a, Deque<Integer> queue){
```

```
        if(queue.isEmpty())
            return 1;

        int lastBit = queue.removeLast();

        int part1 = myPow(a, lastBit);
        int part2 = myPow(superPow(a, queue), 10);

        return (part1 * part2) % base;
    }

    // (a * b) % k = [(a % k) * (b % k)] % k
    /*
        assume 这个函数 已经 返回 mod k 的结果
    */
    private int myPow(int a, int b){
        if(b == 0)
            return 1;


        if(b % 2 == 0)
            return myPow( ((a % base) * (a % base)) % base, b / 2);
        else
            return (  (a % base) * myPow(a, b - 1)  ) % base;
    }
```

```
//先对于一下leetcode 50
class Solution {
    public double myPow(double x, int n) {
        if(n == Integer.MIN_VALUE){
            if(x == -1 || x == 1)
                return 1;
            return 0;
        }
        if(n < 0)
            return myPow(1 / x, -n);
        else if(n == 1)
            return x;
        else if(n == 0)
            return 1;

        if(n % 2 == 0){
            return myPow(x * x, n / 2);
        }else{
            return myPow(x * x, n / 2) * x;
        }
    }
```

```
}
```

```java
//二刷 郭郭版本
    int base = 1337;
    public int superPow(int a, int[] b) {
        Deque<Integer> queue = new ArrayDeque<>();
        for(int num : b)
            queue.addLast(num);

        return superPow(a, queue);


    }

    private int superPow(int a, Deque<Integer> queue){
        if(queue.isEmpty())
            return 1;

        int lastBit = queue.removeLast();

        int part1 = myPow(a, lastBit);
        int part2 = myPow(superPow(a, queue), 10);

        return (part1 * part2) % base;
    }



    int myPow(int a, int k){
        if(k == 0)    return 1;
        a %= base;

        if(k % 2 == 1)
            return (a * myPow(a, k-1)) % base;
        else{
            int sub = myPow(a, k / 2);
            return (sub * sub) % base;
        }
    }

```

```python
class Solution:
    def superPow(self, a: int, b: List[int]) -> int:
```

```
        if not b:
            return 1
        last = b.pop()
        part1 = self.mypow(a, last)
        part2 = self.mypow(self.superPow(a, b), 10)
        return part1 * part2 % 1337

    # def mypow(self, a, k, base=1337):
    #     # a^k % base
    #     if k == 0:
    #         return 1
    #     res = 1
    #     a = a % base
    #     for _ in range(k):
    #         res = res * a
    #         res = res % base
    #     return res

    def mypow(self, a, k, base=1337):
        if k == 0:
            return 1
        if k % 2 == 1:
            return (a * self.mypow(a, k-1)) % base
        if k % 2 == 0:
            return (self.mypow(a, k // 2))**2 % base

作者：jue-qiang-zha-zha
链接：https://leetcode-cn.com/problems/super-pow/solution/372-chao-ji-ci-fang-by-jue-qiang-zha-zha-m0lj/
```

$$
\begin{aligned}
& a^{[1,5,6,4]} \\
=& a^4 \times a^{[1,5,6,0]} \\
=& a^4 \times \left(a^{[1,5,6]}\right)^{10}
\end{aligned}
$$

```
/*
    由上图可以， 由于问题规模缩小， 因此 可以采用递归的方式
*/
int base = 1337;
//大体框架就出来了
int myPow(int a, int k);

int superPow(int a, vector<int> & b){
  //递归的base case
  if(b.empty()) return 1;

```

```
  int last = b.back();
  b.pop_back();

  int part1 = myPow(a, last);
  int part2 = myPow(superPow(a, b), 10);

  return part1 + part2;
}

//同时注意如何处理模运算，能够不溢出
/*
    公式：(a * b) % k = ((a % k) * (b % k)) % k
    对乘法结果求模， 可以等价于先对每个因子都求模， 然后对因子的相乘结果求模
*/
int myPow(int a, int k){
  a %= base;
  int res = 1 % base;
  //到这里， a 和 b 都去模k， 之后只需要结果去模就行

  for(expression){
    //这里存在乘法， 是潜在溢出点
    res *= a;
    res % = base;   //对乘法结果求模
  }

  return res;
}

//同时修正上面的算法
int superPow(int a, vector<int> & b){
  //...
  return (part1 + part2) % base;
}
```

运算的性质，我们可以写出这样一个递归式：

$$a^b = \begin{cases} a \times a^{b-1}, b \text{ 为奇数} \\ (a^{b/2})^2, b \text{ 为偶数} \end{cases}$$

```
/*
    拓展， 对于050 题目的联系
*/
int base = 1337;

int myPow(int a, int k){
```

```
  if(k == 0)    return 1;
  a %= base;

  if(k % 2 == 1)
    return (a * myPow(a, k-1)) % base;
  else{
    int sub = myPow(a, k / 2);
    return (sub * sub) % base;
  }
}
```

373 Find K Pairs with Smallest Sums 优先队列的应用

```
//二刷
public List<List<Integer>> kSmallestPairs(int[] nums1, int[] nums2, int k) {
        List<List<Integer>> res = new ArrayList<>();
        if(nums1.length == 0 || nums2.length == 0)
            return res;
        //store index1 -> nums1,  index2 -> nums2
        PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> (nums1[a[0]] + nums2[a[1]] - nums1[b[0]] - nums2[b[1]]));

        for(int i = 0; i < nums1.length; i++)
            pq.add(new int[]{i, 0});

        while(res.size() != k && !pq.isEmpty()){
            int[] cur = pq.poll();

            int index1 = cur[0];
            int index2 = cur[1];
            List<Integer> path = new ArrayList<>();
            path.add(nums1[index1]);
            path.add(nums2[index2]);

            if(index2 < nums2.length - 1){
                pq.add(new int[]{index1, index2 + 1});
            }
```

```

            res.add(path);
        }

        return res;
    }
```

```java
/*
    思路有些笨重， 就是全部遍历
    然后全部加入优先队列， 之后一个一个拔
*/
class Solution {
    class Unit{
        int sum;
        List<Integer> pair;

        public Unit(){
            sum = 0;
            pair = new ArrayList<>();
        }
    }

    public List<List<Integer>> kSmallestPairs(int[] nums1, int[] nums2, int k) {
        List<List<Integer>> res = new ArrayList<>();
        if(k == 0 || nums1.length == 0 || nums2.length == 0)     return res;

        if(k > nums1.length * nums2.length)     k = nums1.length * nums2.length;
        PriorityQueue<Unit> pq = new PriorityQueue<>((o1, o2) -> o1.sum - o2.sum);

        for(int i = 0; i < nums1.length; i++)
            for(int j = 0; j < nums2.length; j++){
                Unit u = new Unit();
                u.sum = nums1[i] + nums2[j];
                u.pair.add(nums1[i]); u.pair.add(nums2[j]);
                pq.add(u);
            }

        while(k != 0){
            Unit cur = pq.poll();
            res.add(cur.pair);
            k--;
        }

        return res;
    }
```

```
}
```

```
/*
    ref:
    https://leetcode.com/problems/find-k-pairs-with-smallest-sums/discuss/84551/simple-Java-O(KlogK)-solution-with-explanation

这个题 十分类似"merge k sorted list",  下面对其的解析
每次我们只拿头部元素,  一开始必然拿到的是 (nums1[0], nums2[0])
之后每次对剩下元素进行比较
(1,2) -> (1,9) -> (1,10) -> (1,15)
(7,2) -> (7,9) -> (7,10) -> (7,15)
(11,2) -> (11,9) -> (11,10) -> (11,15)
(16,2) -> (16,9) -> (16,10) -> (16,15)

⬇️


  -----
|(1,2)  | -> (1,9) -> (1,10) -> (1,15)
|(7,2)  |-> (7,9) -> (7,10) -> (7,15)
|(11,2) |-> (11,9) -> (11,10) -> (11,15)
|(16,2) |-> (16,9) -> (16,10) -> (16,15)
 --------
 ⬇️
   -----
 | (1,9) |-> (1,10) -> (1,15)
|(7,2)  |-> (7,9) -> (7,10) -> (7,15)
|(11,2) |-> (11,9) -> (11,10) -> (11,15)
|(16,2) |-> (16,9) -> (16,10) -> (16,15)
 --------
*/
public List<int[]> kSmallestPairs(int[] nums1, int[] nums2, int k) {
        // min queue, sorted by pair sum
        PriorityQueue<int[]> q = new PriorityQueue<>((a, b) -> (a[0] + a[1]) - (b[0] + b[1]));
        List<int[]> res = new ArrayList();
        int N1 = nums1.length, N2 = nums2.length;
        if (N1 == 0 || N2 == 0) return res; // no pairs to form, just return an empty res list

        // offer initial pairs {num1, num2, index_of_num2}
        for (int i = 0; i < Math.min(N1, k); i++)
          q.offer(new int[]{nums1[i], nums2[0], 0});
        // get 1st k elem into result, each time, offer potential better pairs into queue
        // if there r not enough pair, just return all pairs

```

```
        for (int i = 0; i < Math.min(N1 * N2, k); i++) {
            // get the best pair and put into res
            int[] cur = q.poll();
            res.add(new int[]{cur[0], cur[1]});
            // next better pair could with be A: {after(num1), num2} or B: {num1.after(num2)}
            // for A, we've already added top possible k into queue, so A is either in the queue already, or not qualified
            // for B, it might be a better choice, so we offer it into queue
            if (cur[2] < N2 - 1 ) { // still at least one elem after num2 in array nums2
                int idx = cur[2] + 1;
                q.offer(new int[]{cur[0], nums2[idx], idx});
            }
        }
        return res;
    }
```

374 Guess Number Higher or Lower

```
/*
    典型二分， 没啥好说的
*/
public int guessNumber(int n) {
  int left = 1, right = n;

  while(left <= right){
    int mid = left + (right - left ) / 2;
    int res = guess(mid);

    if(res == 0)
      return mid;
    else if (res < 0)
      right = mid - 1;
    else
      left = mid + 1 ;
  }

  return -1;
}
```

375 Guess Number Higher or Lower II 典型 DP

```java
/*

*/
    int[][] dp;
    public int getMoneyAmount(int n) {
        dp = new int[n + 1][n + 1];

        return dfs(1, n);
    }

    private int dfs(int left, int right){
        if(left >= right)
            return 0;

        if(dp[left][right] != 0)
            return dp[left][right];

        int res = Integer.MAX_VALUE;
        for(int i = left; i <= right; i++){
            int temp  = i + Math.max(dfs(left, i - 1), dfs(i + 1, right));

            res = Math.min(res, temp);
        }

        dp[left][right] = res;
        return res;
    }
```

376 Wiggle Subsequence

```java
//二刷 dp
public int wiggleMaxLength(int[] nums) {
        int len = nums.length;

        if(len == 1)
            return 1;

        /*
            以 i 结尾
            dpPos ->和前面的差值为正   的wiggle sequence 最大元素个数
            dpNeg ->和前面的差值为负数 的wiggle sequence 最大元素个数
        */
        int[] dpPos = new int[len];
        int[] dpNeg = new int[len];
        Arrays.fill(dpPos, 1);
        Arrays.fill(dpNeg, 1);
        if(nums[1] - nums[0] > 0)
            dpPos[1] = 2;
        else if(nums[1] - nums[0] < 0)
            dpNeg[1] = 2;


        int maxLen = 1;
        for(int i = 2; i < len; i++){
            for(int j = i - 1; j >= 0; j--){
                if(nums[i] - nums[j] > 0)
                    dpPos[i] = Math.max(dpPos[i], dpNeg[j] + 1);
                else if(nums[i] - nums[j] < 0)
                    dpNeg[i] = Math.max(dpNeg[i], dpPos[j] + 1);

            }

            maxLen = Math.max(dpPos[i], dpNeg[i]);
        }

        return maxLen;
    }
```

```java
public int wiggleMaxLength(int[] nums) {
  if(nums.length < 2)     return nums.length;
```

```
3
4     //dpPos: 以nums[i] 为结尾，  与前面的单位差值为正，的最大元素数
5     //dpNeg: 以nums[i] 为结尾，  与前面单位差值为负，   的最大元素数
6     int[] dpPos = new int[nums.length];
7     int[] dpNeg = new int[nums.length];
8
9     int maxLen = 1;
10    Arrays.fill(dpPos, 1);
11    Arrays.fill(dpNeg, 1);
12
13    if(nums[1] - nums[0] > 0)
14      dpPos[1] = 2;
15    else if(nums[1] - nums[0] < 0)
16      dpNeg[1] = 2;
17    maxLen = Math.max(maxLen, Math.max(dpPos[1], dpNeg[1]));
18
19    for(int i = 2; i < nums.length; i++){
20      for(int j = i - 1; j >= 1; j--){
21        if(nums[i] - nums[j] > 0)
22          dpPos[i] = Math.max(dpPos[i], 1 + dpNeg[j]);
23        else if(nums[i] - nums[j] < 0)
24          dpNeg[i] = Math.max(dpNeg[i], 1 + dpPos[j]);
25      }
26
27      maxLen = Math.max(maxLen, Math.max(dpPos[i], dpNeg[i]));
28    }
29
30    return maxLen;
31  }
```

```
1   /*
2       介绍一个 O(n) 时间复杂度的方法
3       up 和 down 都只取决于前一个状态，  在未优化的情况下是这样
4   */
5   public class Solution {
6       public int wiggleMaxLength(int[] nums) {
7           if (nums.length < 2)
8               return nums.length;
9           int[] up = new int[nums.length];
10          int[] down = new int[nums.length];
11          up[0] = down[0] = 1;
12          for (int i = 1; i < nums.length; i++) {
13              if (nums[i] > nums[i - 1]) {
14                  up[i] = down[i - 1] + 1;
15                  down[i] = down[i - 1];
16              } else if (nums[i] < nums[i - 1]) {
17                  down[i] = up[i - 1] + 1;
```

```
                up[i] = up[i - 1];
            } else {
                down[i] = down[i - 1];
                up[i] = up[i - 1];
            }
        }
        return Math.max(down[nums.length - 1], up[nums.length - 1]);
    }
}

public int wiggleMaxLength(int[] nums) {
    int down = 1, up = 1;
    for (int i = 1; i < nums.length; i++) {
        if (nums[i] > nums[i - 1])
            up = down + 1;
        else if (nums[i] < nums[i - 1])
            down = up + 1;
    }
    return nums.length == 0 ? 0 : Math.max(down, up);
}

作者：lgh18
链接：https://leetcode-cn.com/problems/wiggle-subsequence/solution/tan-xin-si-lu-qing-xi-er-zheng-que-de-ti-jie-by-lg/
```

377 Combination Sum IV 背包问题

```
//没有记忆的回溯是过不去
    int total = 0;
    public int combinationSum4(int[] nums, int target) {
        backtrack(nums, target);

        return total;
    }

    private void backtrack(int[] nums, int remains){
        if(remains == 0)
            total++;

        for(int i = 0; i < nums.length; i++){
            if(nums[i] <= remains){
                backtrack(nums, remains - nums[i]);
```

```
            }
        }
    }
```

```
//记忆化回溯，  勉强能够
HashMap<Integer, Integer> map;
    public int combinationSum4(int[] nums, int target) {
        map = new HashMap<>();
        helper(nums, target);

        return map.get(target);
    }
    //this functioon can return the number of combination for remains
    private int helper(int[] nums, int remains){
        if(remains == 0)
            return 0;
        if(map.containsKey(remains))
            return map.get(remains);

        int numOfCombination = 0;

        for(int i = 0; i < nums.length; i++)
            if(remains - nums[i] > 0)
                numOfCombination += helper(nums, remains - nums[i]);
            else if(remains - nums[i] == 0)
                numOfCombination += 1;
        map.put(remains, numOfCombination);

        return map.get(remains);
    }
```

```
public class Solution {
    /*
     * 这里状态定义就是题目要求的，并不难，状态转移方程要动点脑子，也不难：
     * 状态转移方程：dp[i]= dp[i - nums[0]] + dp[i - nums[1]] + dp[i - nums[2]] + ...（当 [] 里面的数 >= 0）
     * 特别注意：dp[0] = 1，表示，如果那个硬币的面值刚刚好等于需要凑出的价值，这个就成为 1 种组合方案
     * 再举一个具体的例子：nums=[1, 3, 4], target=7;
     * dp[7] = dp[6] + dp[4] + dp[3]
     * 即：7 的组合数可以由三部分组成，1 和 dp[6]，3 和 dp[4]，4 和dp[3];
*/
    public int combinationSum4(int[] nums, int target) {
        int[] dp = new int[target + 1];
```

```
        // 这个值被其它状态参考，设置为 1 是合理的
        dp[0] = 1;

        for (int i = 1; i <= target; i++) {
            for (int num : nums) {
                if (num <= i) {
                    dp[i] += dp[i - num];
                }
            }
        }
        return dp[target];
    }
}

作者：liweiwei1419
链接：https://leetcode-cn.com/problems/combination-sum-iv/solution/dong-tai-gui-hua-python-dai-ma-by-liweiwei1419/
```

378 Kth Smallest Element In a Sorted Matrix

```
//二刷
public int kthSmallest(int[][] matrix, int k) {
        int row = matrix.length;
        int col = matrix[0].length;

        /*
            int[] cur
                cur[0] -> row
                cur[1] -> col index
                cur[2] -> val
        */
        PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> (a[2] - b[2]));
        List<Integer> list = new ArrayList<>();
        for(int i = 0; i < row; i++){
            pq.add(new int[]{i, 0, matrix[i][0]});
        }

        int counter = 0;

        while(list.size() < k){
            int[] cur = pq.poll();
            int curColIndex = cur[1];
            list.add(cur[2]);
            if(list.size() == k)
```

```
                return cur[2];

            if(curColIndex != col - 1)
                pq.add(new int[]{cur[0], curColIndex + 1, matrix[cur[0]]
[curColIndex + 1]});
        }

        return pq.poll()[2];
    }
```

```
//与373 极为类似
public int kthSmallest(int[][] matrix, int k) {
        int row = matrix.length;
        int column = row == 0 ? 0 : matrix[0].length;

        //int[] a,b
        //a represents the number ,b is the index for (row, col)
        PriorityQueue<int[]> pq = new PriorityQueue<>((o1, o2) -> (o1[0] - o2[0]));

        for(int i = 0; i < row; i++)
            pq.add(new int[]{matrix[i][0], i, 0});

        while(k != 0){
            int[] cur = pq.poll();
            k--;

            int curNumber = cur[0];
            if(k == 0){
                return curNumber;
            }

            if(cur[2] < column - 1){
                int index = cur[2] + 1;
                pq.add(new int[]{matrix[cur[1]][index], cur[1], index});
            }
        }

        return -1;
    }
```

379 Design Phone Directory

```java
class PhoneDirectory {
  boolean[] sys;
  int size = 0;

  /**
   * Initialize your data structure here
   *
   * @param maxNumbers - The maximum numbers that can be stored in the phone
   *                   directory.
   */
  public PhoneDirectory(int maxNumbers) {
    size = maxNumbers;
    sys = new boolean[size];
    Arrays.fill(sys, true);
  }

  /**
   * Provide a number which is not assigned to anyone.
   *
   * @return - Return an available number. Return -1 if none is available.
   */
  public int get() {
    for (int i = 0; i < size; i++) {
      if (sys[i]) {
        sys[i] = false;
        return i;
      }
    }
    return -1;
  }

  /** Check if a number is available or not. */
  public boolean check(int number) {
    return sys[number];
  }

  /** Recycle or release a number. */
  public void release(int number) {
    sys[number] = true;
  }
}

作者：songhouhou
链接：https://leetcode-cn.com/problems/design-phone-directory/solution/java-listboolean-jian-dan-yi-dong-by-songhouhou/
```

```java
/*
    我自己写的代码，  认为没什么问题，  但是过不去

*/
class Node{
        int val;
        Node next;
        public Node(int val){
            this.val = val;
        }
    }

    int maxNumbers;
    Node head;

    /** Initialize your data structure here
        @param maxNumbers - The maximum numbers that can be stored in the phone directory. */
    public PhoneDirectory(int maxNumbers) {
        maxNumbers = maxNumbers;
        head = new Node(-1);
        int index = 0;
        Node cur = head;

        while(index != maxNumbers){
            Node newNode = new Node(index);
            cur.next = newNode;
            cur = cur.next;

            index++;
        }
    }

    /** Provide a number which is not assigned to anyone.
        @return - Return an available number. Return -1 if none is available. */
    public int get() {
        if(head.next == null){
            return -1;
        }

        Node cur = head.next;
        head.next = head.next.next;

        return cur.val;
    }

    /** Check if a number is available or not. */
```

```
    public boolean check(int number) {
        if(head.next == null)
            return false;

        Node cur = head.next;
        while(cur != null && cur.val <= number){
            if(cur.val == number)
                return true;
            else
                cur = cur.next;
        }

        return false;
    }

    /** Recycle or release a number. */
    public void release(int number) {
        Node newNode = new Node(number);

        Node cur = head;
        if(head.next == null){
            cur.next = newNode;
            return;
        }

        while(cur.next != null){
            if(cur.val < number && cur.next.val > number){
                newNode.next = cur.next;
                cur.next = newNode;
                return;
            }

            cur = cur.next;
        }
    }
```

380 Insert Delete GetRandom O(1) 很棒的随机类题目

```
/*
    这个题指明了,  如何利用hashmap 辅助 数组进行O(1)的插入删除

```

```
    因为我们知道， 数组对于删除操作来说， O(N)的时间复杂度
    通过map <val, index> 从而实现上述操作

*/

class RandomizedSet {
  Map<Integer, Integer> dict;
  List<Integer> list;
  Random rand = new Random();

  /** Initialize your data structure here. */
  public RandomizedSet() {
    dict = new HashMap();
    list = new ArrayList();
  }

  /** Inserts a value to the set. Returns true if the set did not already contain the specified element. */
  public boolean insert(int val) {
    if (dict.containsKey(val)) return false;

    dict.put(val, list.size());
    list.add(list.size(), val);
    return true;
  }

  /** Removes a value from the set. Returns true if the set contained the specified element. */
  public boolean remove(int val) {
    if (! dict.containsKey(val)) return false;

    // move the last element to the place idx of the element to delete
    int lastElement = list.get(list.size() - 1);
    int idx = dict.get(val);
    list.set(idx, lastElement);
    dict.put(lastElement, idx);
    // delete the last element
    list.remove(list.size() - 1);
    dict.remove(val);
    return true;
  }

  /** Get a random element from the set. */
  public int getRandom() {
    return list.get(rand.nextInt(list.size()));
  }
}

作者：LeetCode
```

```
链接：https://leetcode-cn.com/problems/insert-delete-getrandom-o1/solution/chang-shu-shi-jian-cha-ru-shan-chu-he-huo-qu-sui-j/
```

381 Insert Delete GetRandom O(1) Duplicated

```java
/*
    使用HashSet 进行contain duplicates

    同时采用iterator 去拿到set里面的一个元素， 这个方法值得学习
*/
public class RandomizedCollection {
    ArrayList<Integer> lst;
    HashMap<Integer, Set<Integer>> idx;
    java.util.Random rand = new java.util.Random();
    /** Initialize your data structure here. */

    public RandomizedCollection() {
        lst = new ArrayList<Integer>();
        idx = new HashMap<Integer, Set<Integer>>();
    }

    /** Inserts a value to the collection. Returns true if the collection did not already contain the specified element. */
    public boolean insert(int val) {
        if (!idx.containsKey(val))
          idx.put(val, new LinkedHashSet<Integer>());

        idx.get(val).add(lst.size());
        lst.add(val);
        return idx.get(val).size() == 1;
    }

    /** Removes a value from the collection. Returns true if the collection contained the specified element. */
    public boolean remove(int val) {
        if (!idx.containsKey(val) || idx.get(val).size() == 0)
          return false;

        int remove_idx = idx.get(val).iterator().next();
        idx.get(val).remove(remove_idx);

```

```
        int last = lst.get(lst.size() - 1);
        lst.set(remove_idx, last);
        idx.get(last).add(remove_idx);
        idx.get(last).remove(lst.size() - 1);

        lst.remove(lst.size() - 1);
        return true;
    }

    /** Get a random element from the collection. */
    public int getRandom() {
        return lst.get(rand.nextInt(lst.size()));
    }
}

作者：LeetCode
链接：https://leetcode-cn.com/problems/insert-delete-getrandom-o1-duplicates-allowed/solution/o1-shi-jian-cha-ru-shan-chu-he-huo-qu-sui-ji-yua-3/
```

382 Linked List Random Node 随机类题目

```
//二刷
class Solution {
    List<Integer> list;
    /** @param head The linked list's head.
        Note that the head is guaranteed to be not null, so it contains at least one node. */
    public Solution(ListNode head) {
        list = new ArrayList<>();
        traversal(head);
    }

    private void traversal(ListNode head){
        if(head == null)
            return;
        list.add(head.val);
        traversal(head.next);
    }

    /** Returns a random node's value. */
    public int getRandom() {
        int randIndex = (int)(Math.random() * list.size());
        return list.get(randIndex);
    }
```

```
}
```

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    List<Integer> list;
    Random rand;
    /** @param head The linked list's head.
        Note that the head is guaranteed to be not null, so it contains at least one node. */
    public Solution(ListNode head) {
        list  = new ArrayList<>();
        while(head != null){
            list.add(head.val);
            head = head.next;
        }

        rand = new Random();
    }

    /** Returns a random node's value. */
    public int getRandom() {
        return list.get(rand.nextInt(list.size()));
    }
}

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

```
/*
    正常写法，  这里的题目，是假设数据以数据流的形式传送过来，而并非全部一下子来
    因为下面的while循环并非遍历式写法
*/
```

```

class Solution {
    ListNode head;
    Random rand;
    /** @param head The linked list's head.
        Note that the head is guaranteed to be not null, so it contains at least one node. */
    public Solution(ListNode head) {
        this.head = head;
        rand = new Random();
    }

    /** Returns a random node's value. */
    public int getRandom() {
        ListNode temp = head;
        int res = temp.val;
        int i = 1;
        while(temp.next != null){
            temp = temp.next;
            i++;
            if(rand.nextInt(i) == 0)
                res = temp.val;
        }

        return res;
    }
}

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

作者：Jasion_han
链接：https://leetcode-cn.com/problems/linked-list-random-node/solution/382-lian-biao-sui-ji-jie-dian-sui-ji-deng-gai-lu-f/
```

383 Ransom Note

```
    public boolean canConstruct(String ransomNote, String magazine) {
        int[] r = new int[26];
        int[] m = new int[26];

```

```
        for(char ch : ransomNote.toCharArray())
            r[ch - 'a']++;
        for(char ch : magazine.toCharArray())
            m[ch - 'a']++;

        for(int i = 0; i < 26; i++){
            if(r[i] > m[i])
                return false;
        }

        return true;
    }
```

```

//二刷
public boolean canConstruct(String ransomNote, String magazine) {
        Map<Character, Integer> mapR = new HashMap<>();
        Map<Character, Integer> mapM = new HashMap<>();

        for(char ch : ransomNote.toCharArray())
            mapR.put(ch, mapR.getOrDefault(ch, 0) + 1);

        for(char ch : magazine.toCharArray())
            mapM.put(ch, mapM.getOrDefault(ch, 0) + 1);


        for(Character ch : mapR.keySet()){
            if(mapM.getOrDefault(ch, 0) < mapR.get(ch))
                return false;
        }

        return true;
    }
```

```
public boolean canConstruct(String ransomNote, String magazine) {
  HashMap<Character, Integer> map1, map2;

  map1 = getAllCharFreq(ransomNote);
  map2 = getAllCharFreq(magazine);

  for(Character ch : map1.keySet()){
    if(!map2.containsKey(ch) ||map1.get(ch) > map2.get(ch))
```

```
      return false;
    }

    return true;
}

private HashMap<Character, Integer> getAllCharFreq(String s){
    HashMap<Character, Integer> map = new HashMap<>();

    for(int i = 0; i < s.length(); i++)
      map.put(s.charAt(i), map.getOrDefault(s.charAt(i), 0) + 1);

    return map;
}
```

384 Shuffle an Array 随机类题目

```
class Solution {
public:
    vector<int> nums;
    Solution(vector<int>& nums) {
        this->nums = nums;
    }

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

    /** Returns a random shuffling of the array. */
    vector<int> shuffle() {
        vector<int> res = vector<int>(nums);
        for(int i = 0; i < nums.size(); i++){
            if(i == nums.size() - 1)
                continue;
            int pos = std::rand() % (nums.size() - i) + i;
            swap(res, i, pos);
        }

        return res;
    }

    void swap(vector<int>& nums, int i, int j){
        int temp = nums[i];
```

```
        nums[i] = nums[j];
        nums[j] = temp;
    }
};

```

```
/*
It's interesting to know the function clone()


for position 0, we choose a random number nums[i] from nums[0 : n-1], put it to nums[0](actually swap it with nums[0] such that we can follow the following rules);

for position 1, we choose a random number nums[i] from nums[1 : n-1], put it to nums[1], (...);

for position 2, we choose a random number nums[i] from nums[2 : n-1], put it to nums[2], (...);
...
*/

class Solution {
  int[] nums;
  public Solution(int[] nums) {
    this.nums = nums;
  }

  /** Resets the array to its original configuration and return it. */
  public int[] reset() {
    return nums;
  }

  /** Returns a random shuffling of the array. */
  public int[] shuffle() {
    int[] res = new int[nums.length];
    res = nums.clone();

    for(int i = 0; i < nums.length; i++){
      //注意这里的 + i,  它保证了我们每次的随机数范围是[i --> nums.length);
      //这样就不会把前面已经洗好的牌打乱
      int rand = (int)(Math.random()*(nums.length - i)) + i;
      swap(res, i, rand);
    }
    return res;
  }
```

```
37
38    public void swap(int[] nums, int i, int j){
39      int temp = nums[i];
40      nums[i] = nums[j];
41      nums[j] = temp;
42    }
43  }
44  //https://leetcode.com/problems/shuffle-an-array/discuss/85958/First-Accepted-Solution-Java
```

385 Mini Parser 栈的很棒的应用

```
1  /*
2      二刷:  优化变量名字
3  */
4  public NestedInteger deserialize(String s) {
5          if(!s.startsWith("["))
6              return new NestedInteger(Integer.valueOf(s));
7
8          int left = 1;
9          Deque<NestedInteger> stack = new ArrayDeque<>();
10         NestedInteger res = new NestedInteger();
11         stack.push(res);
12
13         for(int right = 1; right < s.length(); right++){
14             char ch = s.charAt(right);
15             if(ch == '['){
16                 NestedInteger ni = new NestedInteger();
17                 stack.peek().add(ni);
18                 stack.push(ni);
19                 left = right + 1;
20             }else if(ch == ']' || ch == ','){
21                 if(right > left){
22                     Integer val = Integer.valueOf(s.substring(left, right));
23                     stack.peek().add(new NestedInteger(val));
24                 }
25
26                 left = right + 1;
27                 if(ch == ']')
28                     stack.pop();
29             }
```

```
        }

        return res;
    }
```

```
/*
    牛逼解法
    栈的应用

    算法核心
    1. 如果是左括号， 就新建一个Nestlist， 然后入站
    2. 如果是右括号 或者 ，
        首先将前面的数字入nestlist

        如果是右括号， 就弹出当前的栈， 因为所有的对于当前list 已经处理完毕
*/
public NestedInteger deserialize(String s) {
        if (!s.startsWith("[")) {
            return new NestedInteger(Integer.valueOf(s));
        }
        Stack<NestedInteger> stack = new Stack<>();
        NestedInteger res = new NestedInteger();
        stack.push(res);
        int start = 1;
        for (int i = 1; i < s.length(); i ++) {
            char c = s.charAt(i);
            if (c == '[') {
                NestedInteger ni = new NestedInteger();
                stack.peek().add(ni);
                stack.push(ni);
                start = i + 1;
            } else if (c == ',' || c == ']') {
                if (i > start) {
                    Integer val = Integer.valueOf(s.substring(start, i));
                    stack.peek().add(new NestedInteger(val));
                }
                start = i + 1;

                if (c == ']') { //如果碰到右括号， 说明当前nestlist 处理完毕！， 弹栈
                    stack.pop();
                }
            }
        }
        return res;
    }
```

386 Lexicographical Numbers 典型DFS 题目

```
/*
The idea is pretty simple. If we look at the order we can find out we just keep adding digit from 0 to 9 to every digit and make it a tree.
Then we visit every node in pre-order.
       1         2         3    ...
      /\        /\        /\
   10 ...19  20...29  30...39   ....

山谷里有座千年古刹，一日，方丈收到一个任务，将1-n的字典排序进行输出；
思绪良久，方丈找来9个大法师，对第一个大法师说：“大弟子，我现在给你一个任务，我给你一个数字1，你负责把这个数字开头的，并且不大于n的所有数字按照字典排序交付于我。”
接着又对第二个大法师说：“二弟子，我也给你一个任务，我给你一个数字2，你负责把这个数字开头的，并且不大于n的所有数字按照字典排序交付于我。”
如是依次对剩下弟子说了一遍，各大法师领命依次离去；
大法师归于禅室，思虑良久：方觉方丈之策可复行之，乃唤来座下大弟子十人，依次要求将10,11,12...19开头的并且不大于n的所有数字交付于己。众大弟子离去，效法以行。

作者：cgq_dlut
链接：https://leetcode-cn.com/problems/lexicographical-numbers/solution/di-gui-jie-fa-by-cgq_dlut/

*/
public class Solution {
    public List<Integer> lexicalOrder(int n) {
        List<Integer> res = new ArrayList<>();
        for(int i=1;i<10;++i){
          dfs(i, n, res);
        }
        return res;
    }

    public void dfs(int cur, int n, List<Integer> res){
        if(cur>n)
            return;
        else{
            res.add(cur);
            for(int i=0;i<10;++i){
                if(10*cur+i>n)
                    return;
                dfs(10*cur+i, n, res);
            }
        }
```

```
    }
}
```

387 First Unique Character in a String

```
//虐菜题目
public int firstUniqChar(String s) {
        HashMap<Character, Integer> map = new HashMap<>();
        for(int i = 0; i < s.length(); i++)
            map.put(s.charAt(i), map.getOrDefault(s.charAt(i), 0) + 1);

        for(int i = 0; i < s.length(); i++)
            if(map.get(s.charAt(i)) == 1)
                return i;
        return -1;
    }
```

388 Longest Absolute File Path

```
import java.util.*;

public class Solution {
    class Node{
        public int level;

        public int sum;

        public Node(int level, int sum){
            this.level   = level;
            this.sum     = sum;
        }

    }

    public int lengthLongestPath(String input) {
        String[] splits = input.split("\n");
        Deque<Node> stack = new ArrayDeque<>();
```

```java

        int res = 0;
        for(int i = 0; i < splits.length;){
            int howManyTabs = countTab(splits[i]);

            if(stack.isEmpty()){
                if(splits[i].indexOf('.') != -1){
                    res = Math.max(res, splits[i].length());
                }else{
                    Node newNode = new Node(howManyTabs, splits[i].length() - howManyTabs);
                    stack.push(newNode);
                }
                i++;
            }else{
                Node peek = stack.peek();
                //上下级关系
                if(peek.level + 1 == howManyTabs){
                    Node newNode = new Node(howManyTabs, splits[i].length() + peek.sum - howManyTabs + 1);
                    if(splits[i].indexOf('.') != -1){
                        res = Math.max(res, newNode.sum);
                    }else{
                        stack.push(newNode);
                    }
                    i++;
                }else{
                  stack.pop();
                }
            }
        }

        return res;
    }

    public int countTab(String s){
        StringBuilder sb = new StringBuilder();
        int count = 0;
        for(int i = 0;; i++){
            sb.append("\t");
            if(s.startsWith(sb.toString()))
                count++;
            else
                break;
        }

        return count;
    }
}
```

389 Find the difference

```java
    public char findTheDifference(String s, String t) {
        int[] str1 = new int[26];
        int[] str2 = new int[26];

        for(char ch : s.toCharArray())
            str1[ch - 'a']++;
        for(char ch : t.toCharArray())
            str2[ch - 'a']++;

        for(int i = 0; i < 26; i++){
            if(str1[i] < str2[i])
                return (char)(i + 'a');
        }

        return '@';
    }
```

```java
/*
  虐菜题

*/
public char findTheDifference(String s, String t) {
        int[] charArray = new int[26];

        for(int i = 0; i < s.length(); i++){
            int indexS = s.charAt(i) - 'a';
            int indexT = t.charAt(i) - 'a';

            charArray[indexS]++;
            charArray[indexT]--;
        }

        charArray[t.charAt(s.length()) - 'a']--;

        for(int i = 0; i < 26; i++)
            if(charArray[i] == -1)
                return (char)(i + 'a');

```

```
        return 'a';
    }
```

390 Elimination Game 脑经急转弯

```
/*
    数学手段分析这个题
    假设偶数个
      第一轮 1 2 3 4 ..... 2K
      第二轮 2 4 ..... 2k
      编号   k k-1 ......1
找规律
 对于所有的第二轮数字来说
   第二轮所有数字 / 2 + 编号永远等于 k + 1

   也就是说比如有2k 个数字 这个时候求出下一轮的方法
   f(2k) 表示最后剩下元素在该轮的位置,  那么
     f(k) + f(2k) / 2 = k + 1
     f(2k) = 2(k + 1 - f(k));
     f(k) =  2(k / 2 + 1 - f(k / 2))
     也就是
     f(n) = 2(n / 2 + 1 - f(n / 2))

*/
public:
    int lastRemaining(int n) {
        return n == 1 ? 1 : 2 * (n / 2 + 1 - lastRemaining(n / 2));
    }
};
```

391 Perfect Rectangle

```
//二刷 java 版本
class Solution {
   class Node{
        public int x;
        public int y;

        public Node(int x, int y){
            this.x = x;
            this.y = y;
        }
```

```java

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            Node node = (Node) o;
            return x == node.x && y == node.y;
        }

        @Override
        public int hashCode() {
            return toString().hashCode();
        }

        @Override
        public String toString() {
            return x + "@" + y;
        }
    }

    public boolean isRectangleCover(int[][] rectangles) {
        int left   = Integer.MAX_VALUE, right = Integer.MIN_VALUE;
        int bottom = Integer.MAX_VALUE, top   = Integer.MIN_VALUE;

        int area = 0;
        Map<Node, Integer> map = new HashMap<>();

        for(int[] rect : rectangles){
            left   = Math.min(left, rect[0]);
            right  = Math.max(right, rect[2]);
            bottom = Math.min(bottom, rect[1]);
            top    = Math.max(top, rect[3]);

            area += (rect[2] - rect[0]) * (rect[3] - rect[1]);
            addNodeToMap(map, rect[0], rect[2], rect[3], rect[1]);
        }

        if(area != (right - left) * (top - bottom))
            return false;

        addNodeToMap(map, left, right, top, bottom);
        for(Node node : map.keySet()){
            if(map.get(node) % 2 != 0)
                return false;
        }

        return true;
    }

```

```
    private void addNodeToMap(Map<Node, Integer> map, int left, int right, int top, int bottom){
        Node leftBtn  = new Node(left, bottom);
        Node rightTop = new Node(right, top);
        Node leftTop  = new Node(left, top);
        Node rightBtn = new Node(right, bottom);

        map.put(leftBtn, map.getOrDefault(leftBtn, 0) + 1);
        map.put(rightTop, map.getOrDefault(rightTop, 0) + 1);
        map.put(leftTop, map.getOrDefault(leftTop, 0) + 1);
        map.put(rightBtn, map.getOrDefault(rightBtn, 0) + 1);
    }
}
```

```
/*
    解题思路:
    首先所有的小块面积要等于大块的
    其次除了矩形的四个角，其他各顶点必须出现偶数次
*/
class Solution {
public:
    bool isRectangleCover(vector<vector<int>>& rectangles) {
        int left=INT_MAX,right=INT_MIN;
        int bottom=INT_MAX,top=INT_MIN;
        int area=0;
        map<pair<int,int>,int> m;   //保存每个顶点数量

        for(vector<int>& a:rectangles){
            left = min(left,a[0]);  //找最大矩形
            right= max(right,a[2]);
            bottom=min(bottom,a[1]);
            top  = max(top,a[3]);
            area += (a[2]-a[0])*(a[3]-a[1]);
            m[{a[0],a[1]}]++;   //保存4个顶点
            m[{a[2],a[3]}]++;
            m[{a[0],a[3]}]++;
            m[{a[2],a[1]}]++;
        }

        if(area != (right-left)*(top-bottom))
            return false;

        m[{left,bottom}]++; //把大矩形有4个角放入后,所有点都应该是偶数了
        m[{left,top}]++;
        m[{right,bottom}]++;
        m[{right,top}]++;

```

```
        for(auto it=m.begin();it != m.end(); it++)
          if((*it).second %2 ==1)
            return false;
        return true;
    }
};

作者：xiu-xi-e
链接：https://leetcode-cn.com/problems/perfect-rectangle/solution/bi-jiao-rong-yi-li-jie-de-fang-fa-by-xiu-xi-e/
```

392 isSubsequence 很好的方法

```
//二刷
class Solution {
    public boolean isSubsequence(String s, String t) {
        int up = 0;
        int down = 0;

        int len1 = s.length();
        int len2 = t.length();

        while(up < len1 && down < len2){
            if(s.charAt(up) == t.charAt(down)){
                up++;
                down++;
            }else{
                down++;
            }
        }

        return up == len1;
    }
```

```
/*
    简单dp
*/
bool isSubsequence(string s, string t) {
  /*
                dp[i][j] mean
                s[0...i] can be formed from t[0...j]
```

```
                return dp[s.length][t.length]
        */
  vector<vector<bool>> dp(s.length() + 1, vector<bool>(t.length()+1, false));
  for(int j = 0; j <= t.length(); j++)        //base case when s = "", t = ""
    dp[0][j] = true;
  for(int i = 1; i <= s.length(); i++)
    for(int j = i; j <= t.length(); j++)
      if(s[i-1] == t[j-1])
        dp[i][j] = dp[i-1][j-1];
    else
      dp[i][j] = dp[i][j-1];



  return dp[s.length()][t.length()];
}
```

```
/*
    算法思路，  举例比如
    “ahbgdca”

    记录每一个字符出现的最后位置
    比如以a为例，从后往前遍历，记录第一个a的位置 s.length() - 1
    当遍历到 0 的时候，  nextPos 的值正好等于s.length() - 1
    那么我就可以记录到 对应的下一个字符的位置

    因为最终是要有很多待判断字符串，那么这个时候就要想到预先处理的思路
    KMP
    本质：贪心算法
*/
class Solution {
public:
  bool isSubsequence(string s, string t) {
    t.insert(t.begin(), ' ');
    int len1 = s.size(), len2 = t.size();

    vector<vector<int> > dp(len2 , vector<int>(26, 0));
/*
        0 1 2 3 4 5
        a b c c q a
                  |

        dp[0]["a"] = 0
        dp[0]["b"] = 1
        dp[0]["c"] = 2

        dp[0]["a"] = 0
```

```
        dp[2]["c"] = 3
        target: bqa

        "abcaderqfgda"
*/
    for (char c = 'a'; c <= 'z'; c++) {
      int nextPos = -1; //表示接下来再不会出现该字符

      for (int i = len2 - 1; i>= 0; i--) {  //为了获得下一个字符的位置，要从后往前
        dp[i][c - 'a'] = nextPos;
        if (t[i] == c)
          nextPos = i;
      }
    }

    int index = 0;
    for (char c : s) {
      index = dp[index][c - 'a'];
      if (index == -1)
        return false;
    }
    return true;

  }
};


作者：dongzengjie
链接：https://leetcode-cn.com/problems/is-subsequence/solution/dui-hou-xu-tiao-zhan-de-yi-xie-si-kao-ru-he-kuai-s/
```

394 Decode String 不错的递归类型题目

```
//二刷
class Solution {
    public String decodeString(String s) {
        if(onlyString(s)){
            return s;
        }

        int index = 0;
        int len = s.length();

```

```
        StringBuilder res = new StringBuilder();
        while(index < len){
            char ch = s.charAt(index);
            if('a' <= ch && ch <= 'z'){
                res.append(ch);
                index++;
            }else{
                int num = 0;
                int left = index;
                int right = index;
                while(right < len && s.charAt(right) >= '0' && s.charAt(right) <= '9')
                    right++;
                num = Integer.parseInt(s.substring(left, right));
                left = right;
                right++;
                int leftP = 1;
                while(leftP != 0){
                    char temp = s.charAt(right);
                    if(temp == '['){
                        leftP++;
                    }else if(temp == ']'){
                        leftP--;
                    }

                    if(leftP == 0)
                        break;
                    right++;
                }

                String smallFrac = decodeString(s.substring(left + 1, right));
                for(int i = 0; i < num; i++)
                    res.append(smallFrac);

                index = right + 1;
            }
        }

        return res.toString();
    }

    public boolean onlyString(String s){
        int index = 0;
        while(index != s.length()){
            char ch = s.charAt(index);

            if(ch == '[' || ch == ']')
                return false;
            index++;
```

```
        }

        return true;
    }
}
```

```
/*
    题目本身有也有一定难度
    比如如何能够把递归写的更加简洁?

    本题可以参照使用引用的index,  可以使得我们不需要传递地址 也可以实现 【 】 左右括号索引的移动
*/
class Solution {
public:
string helper(string s, int& index) {
    string res;
    int num = 0;
    string temp;
    while (index < s.size()) {
        if (s[index] >= '0' && s[index] <= '9')
            num = 10 * num + s[index] - '0';

        else if (s[index] == '[') {
            temp = helper(s, ++index);//碰到'[',开始递归
            while(num-- > 0)
              res += temp;
            num = 0;//num置零
        }
        else if (s[index] == ']')
          break;//碰到']',结束递归
        else
          res += s[index];

        index++;
    }
    return res;
}
string decodeString(string s) {
    int index = 0;
    return helper(s, index);
}
};

作者: yexiso
```

```
链接：https://leetcode-cn.com/problems/decode-string/solution/c-di-gui-fen-zhi-shuang-bai-by-avphn4vwuo/
来源：力扣（LeetCode）
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

395 Longest Substring With At Least K Repeating Characters 不错的分治算法

```go
func longestSubstring(s string, k int) (ans int) {
  if s == ""{
    return 0
  }

  alpha := [26]int{}
  for _, ch := range s{
    alpha[ch - 'a']++
  }

  var split byte
  for i, c := range alpha[:]{
    if 0 < c && c < k{
      split = 'a' + byte(i)
      break
    }
  }

  if split == 0{
    return len(s)
  }

  for _, substr := range strings.Split(s, string(split)){
    ans = max(ans, longestSubstring( substr, k))
  }

  return
}


func max(num1 int, num2 int) int{
  if num1 > num2 {
```

```
    return num1
  } else {
    return num2
  }
}
```

```
    int res = 0;
    public int longestSubstring(String s, int k) {
        if(s.length() < k)
            return 0;

        int[] alpha = new int[26];

        for(char ch : s.toCharArray())
            alpha[ch - 'a']++;

        List<Integer> splitPoint = new ArrayList<>();
        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);
            if(alpha[ch - 'a'] < k){
                splitPoint.add(i);
            }
        }

        if(splitPoint.size() == 0){
            res = Math.max(res, s.length());
        }else{
            List<String> strs = new ArrayList<>();

            int index = 0;
            for(int i = 0;  i< splitPoint.size(); i++){
                int nextIndex = splitPoint.get(i);

                if(index == 0)
                    strs.add(s.substring(0, nextIndex));
                else
                    strs.add(s.substring(index + 1, nextIndex));
                index = nextIndex;

                if(i == splitPoint.size() - 1)
                    strs.add(s.substring(nextIndex + 1));
            }



            for(String str : strs)
```

```
                longestSubstring(str, k);
        }

        return res;
    }
```

```
https://leetcode-cn.com/problems/longest-substring-with-at-least-k-repeating-characters/solution/jie-ben-ti-bang-zhu-da-jia-li-jie-di-gui-obla/


```

```
//二刷， 暴力 过了 30 / 31
public int longestSubstring(String s, int k) {
        int len = s.length();
        int res = 0;
        for(int i = 0; i < len; i++){
            for(int j = i + 1; j <= len; j++){
                String frac = s.substring(i, j);

                int tempRes = getRes(frac, k);
                res = Math.max(tempRes, res);
            }
        }

        return res;
    }

    private int getRes(String s, int k){
        int[] map = new int[26];
        for(char ch : s.toCharArray())
            map[ch - 'a']++;

        for(int i = 0; i < 26; i++){
            if(map[i] == 0)
                continue;

            if(map[i] < k)
                return 0;
        }

        return s.length();
    }
```

```cpp
/*
    算法思路:
      与之前一些类似题目不同的是， 采用分治算法， 分割可能的子字符串，之后进行计算
*/
class Solution {
public:
    int longestSubstring(string s, int k) {
        int res = 0;
        unordered_map<char, int> map;
        for(char ch : s)    map[ch]++;

        vector<int> split;
        for(int i = 0; i < s.length(); i++)
            if(map[s[i]] < k)
                split.push_back(i);


        if(split.size() == 0)   return s.length();
        split.push_back(s.length());  //必须添加， 否则比如如下案例 “bbaaacbd” 会有问题
                                      //当判断 bbaaa,时， split 没有存储 s.length 导致无法计算aaa的长度

        int left = 0;
        for(int i = 0; i < split.size(); i++){
            int len = split[i] - left;

            if(len > res)
                res = max(res, longestSubstring(s.substr(left, len), k)); //分治算法
            left = split[i] + 1;
        }
        return res;
    }
};
```

396 Rotate Function

```cpp
//二刷
public int maxRotateFunction(int[] nums) {
```

```
        /*
            1. get F(0)

            2. to get F(N)
        */

        int func = 0;
        int sum = 0;
        int res = 0;
        for(int i = 0; i < nums.length; i++){
            func += i * nums[i];
            sum += nums[i];
        }

        res = func;
        for(int i = 1; i < nums.length; i++){


            func = func + sum - nums.length * nums[nums.length - i];
            res = Math.max(func, res);
        }

        return res;
    }
```

```
/*
    找规律题目
     (1) F(k) = 0 * Bk[0] + 1 * Bk[1] + ... + (n-2) * Bk[n-2] + (n-1) * Bk[n-1]
   (2) F(k+1) = 0 * Bk[n-1] + 1 * Bk[0] + 2 * Bk[2] + ... + (n-1) * Bk[n-2]
   (2) - (1) 得: F(k+1) - F(k) = (Bk[0] + Bk[1] + ... + Bk[n-2]) - (n-1)*Bk[n-1]
  可得: F(k+1) - F(k) = (Bk[0] + Bk[1] + ... + Bk[n-2] + Bk[n-1]) - n*Bk[n-1]
  令S=Sum{Bk}
  有: F(k+1) = F(k) + S - n * Bk[n-1]

*/
int maxRotateFunction(vector<int>& A) {
        long sum  = getSum(A, false);
        long f_pre   = getSum(A, true);
        long f_cur = 0;
        long res = f_pre;

        for(int i = 1; i < A.size(); i++){
            f_cur = f_pre + sum - (long)A.size() * (long)A[A.size() - i];
            f_pre = f_cur;
            res = max(res, f_cur);
        }
```

```
22
23          return res;
24      }
25
26      long getSum(vector<int> &A, bool weighted){
27          long total = 0;
28          for(int i = 0; i < A.size(); i++)
29              if(weighted)
30                  total += i * A[i];
31              else
32                  total +=  A[i];
33          return total;
34      }
```

397 Integer Replacement

```
1  //二刷
2  Map<Integer, Integer> map = new HashMap<>();
3      public int integerReplacement(int n) {
4          if(n == 0 || n == 1)
5              return 0;
6          else if(n == Integer.MAX_VALUE)
7              return 32;
8          else if(map.containsKey(n))
9              return map.get(n);
10
11
12         if(n % 2 == 0){
13             int res = integerReplacement(n / 2) + 1;
14             map.put(n, res);
15             return res;
16         }else{
17             int res = Math.min(integerReplacement(n - 1), integerReplacement(n +
   1)) + 1;
18             map.put(n, res);
19             return res;
20         }
21     }
```

```java
//二刷
public int integerReplacement(int n) {
        if(n == 0 || n == 1)
            return 0;
        else if(n == Integer.MAX_VALUE)
            return 32;

        if(n % 2 == 0){
            return integerReplacement(n / 2) + 1;
        }else{
            return Math.min(integerReplacement(n - 1), integerReplacement(n + 1)) + 1;
        }
    }
```

```java
public:
    int integerReplacement(int n) {
        if(n == 2147483647) return 32;
        if(n <= 3)          return n-1;

        if(n % 2 == 0)
            return integerReplacement(n / 2) + 1;
        else
            return min(integerReplacement(n + 1), integerReplacement(n - 1)) + 1;

        return -1;
    }
```

```java
/*
    TLE
*/
int integerReplacement(int n) {
  vector<int> dp(n + 5, n);
  dp[1] = 0;
  dp[2] = 1;
  dp[3] = 2;
  dp[4] = 2;

  if(n <= 4)      return dp[n];

  for(int i = 5; i <= n; i++){
    int num = i;
```

```
    if(num % 2 == 0)
      dp[num] = dp[num / 2] + 1;
    else
      dp[num] = min(dp[(num+1) / 2] + 2, dp[(num - 1) / 2] + 2);
  }

  // for(int i = 1; i <= n; i++)
  //     cout << i << "is times"<<dp[i] << endl;

  return dp[n];
}
```

398 Random Pick Index

```
class Solution {
    unordered_map<int, vector<int>> map;
public:
    Solution(vector<int>& nums) {
        for(int i = 0; i < nums.size(); i++){
            if(map.find(nums[i]) == map.end())
                map.emplace(nums[i], vector<int>());
            map[nums[i]].push_back(i);
        }
    }

    int pick(int target) {
        if(map.find(target) == map.end())
            return -1;

        return map[target][rand() % map[target].size()];
    }
};
```

```
/*
    蓄水池原理
    每次来，都以当前个数作为选择余地，判断是否进行选择
*/
class Solution {
public:
```

```
    Solution(vector<int>& nums) {
        pvec = &nums;
    }

    int pick(int target) {
        int count = 0;
        int res = 0;
        for (int i=0;i<(*pvec).size();++i){
            if ((*pvec)[i] == target){
                ++count;
                if (rand() % count == count-1)  //以count 分之1的概率留下该元素
                    res = i;
            }
        }
        return res;
    }
private:
    vector<int>* pvec;
};
```

399 Evaluate Division

```
/*
  reference: huahua
*/
private Map<String, HashMap<String, Double>> g = new HashMap<>();
    public double[] calcEquation(List<List<String>> equations, double[] values, List<List<String>> queries) {
        for(int i = 0; i < equations.size(); i++){
            String x = equations.get(i).get(0);
            String y = equations.get(i).get(1);
            double k = values[i];

            g.computeIfAbsent(x, l -> new HashMap<String, Double>()).put(y, k);
            g.computeIfAbsent(y, m -> new HashMap<String, Double>()).put(x, 1.0 / k);
        }


        double[] ans = new double[queries.size()];

```

```
        for(int i = 0; i < queries.size(); i++){
            String x = queries.get(i).get(0);
            String y = queries.get(i).get(1);
            if(!g.containsKey(x) || !g.containsKey(y)){
                ans[i] = -1.0;
            }else{
                ans[i] = divide(x, y, new HashSet<String>());
            }
        }

        return ans;
    }

    private double divide(String x, String y, Set<String> visited){
        if(x.equals(y))
            return 1.0;

        visited.add(x);
        if(!g.containsKey(x))
            return -1.0;

        for(String n : g.get(x).keySet()){
            if(visited.contains(n))
                continue;
            visited.add(n);
            double d = divide(n, y, visited);
            if(d > 0)
                return d * g.get(x).get(n);
        }

        return -1.0;
    }
```

400 Nth Digit

```
//二刷
public int findNthDigit(int n) {
        if (n <= 9)
            return n;

        long ln = n - 9;
        long digits = 2;
```

```
8
9           while(ln > 0){
10              long roundDigits = getRoundDigits(digits);
11
12              if(ln < roundDigits)
13                  break;
14              digits++;
15              ln -= roundDigits;
16          }
17
18          long whichNumber = (ln - 1) / (digits);
19          long whichPos    = (ln - 1) % (digits);
20          long startNumber = getStartNumber(digits);
21
22          String res = startNumber + whichNumber + "";
23          return res.charAt((int)whichPos) - '0';
24      }
25
26      private long getStartNumber(long digits) {
27          StringBuilder start = new StringBuilder();
28          start.append("1");
29          for(int i = 0; i < digits - 1; i++)
30              start.append("0");
31
32          return Long.parseLong(start.toString());
33      }
34
35      private long getRoundDigits(long digits) {
36          StringBuilder end   = new StringBuilder();
37          end.append("9");
38
39          for(int i = 0; i < digits - 1; i++)
40              end.append("9");
41
42          return (Long.parseLong(end.toString()) - getStartNumber(digits) + 1) *
    digits;
43      }
44
```