短代码速记

ACM模式

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        while (in.hasNextInt()) {
            int a = in.nextInt();
            int b = in.nextInt();
            System.out.println(a + b);
        }
    }
}
```

```
        Scanner in = new Scanner(System.in);
        String input = "";
        while (in.hasNext()) {
            input = in.next();
        }
```

```
        Scanner in = new Scanner(System.in);

        String inputStr =in.nextLine().toLowerCase();
        String inputSingle = in.nextLine();
```

```
  Scanner sc = new Scanner(System.in);
        //获取个数
        int num = sc.nextInt();

        //输入
        for(int i =0 ; i < num ;i++){
            set.add(sc.nextInt());
        }
```

```
4
0 1
0 2
1 2
3 4
   public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        int count = sc.nextInt();
        Map<Integer, Integer> map = new TreeMap<>();
        for (int i = 0; i < count; i++) {
            int key = sc.nextInt();
            int value = sc.nextInt();
        }
    }
```

多组样例输入

```
8
1 2 3 4 5 6 7 8
4
```

```
9
1 2 3 4 5 6 7 8 9
6
```

```
    Scanner in = new Scanner(System.in);
        while (in.hasNext()) {
            int input = in.nextInt();
            List<Integer> result = new ArrayList<Integer>();
            for (int i = 0; i < input; i++) {
                int num = in.nextInt();
                result.add(num);
            }
            int index = in.nextInt();
            // todo
        }
```

十六进制转十进制

```
        Scanner in = new Scanner(System.in);
        while (in.hasNext()) {

            String input = in.next();
            input = input.replaceAll("x","");
            BigInteger bigInteger = new BigInteger(input,16);

            System.out.println(bigInteger.intValue());

        }
```

十进制转二进制

```
Integer.toBinaryString(int i)
```

排序

```
 Collections.sort(list, new Comparator<Node>() {
                @Override
                public int compare(Node o1, Node o2) {
                    return o1.score - o2.score;
                }
            });
```

获取ascil值

```
int ascii = Integer.valueOf(input.charAt(i));
```

ascil转字符串

```
 int i = 97;
 System.out.println(Character.toString((char)i));
```

计算年月日差

```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
LocalDate start = LocalDate.of(year, 1, 1);
LocalDate end = LocalDate.of(year, month, day);
long daysDiff = ChronoUnit.DAYS.between(start, end);
```

有顺序的set

```
LinkedHashSet
```

四则运算

```
import javax.script.*;

   public static void main(String[] args) throws ScriptException {
        Scanner scan = new  Scanner(System.in);
        String input = scan.nextLine();
        input = input.replace("[","(");
        input = input.replace("{","(");
        input = input.replace("}",")");
        input = input.replace("]",")");
        ScriptEngine scriptEngine = new
ScriptEngineManager().getEngineByName("nashorn");
        System.out.println(scriptEngine.eval(input));
    }
```

单词接龙

```
        Map<String, Integer> visited = new HashMap<>();
        visited.put(beginWord, 1);
        Queue<String> queue = new LinkedList();
        queue.add(beginWord);
        while (!queue.isEmpty()) {
            String word = queue.poll();
            int level = visited.get(word);
            for (int i = 0; i < wordList.size(); i++) {
                String anotherWord = wordList.get(i);
                if (visited.get(anotherWord) == null) {
                    if (isOne(word, anotherWord)) {
                        if (anotherWord.equals(endWord)) {
                            queue.clear();
                            System.out.println(visited);
                            return level + 1;
                        }
                        visited.put(anotherWord, level + 1);
                        queue.add(anotherWord);
                    }
                }
            }
        }
```

最小公倍数

```
System.out.println(a*b/gcd(a,b));
private static int gcd(int a, int b) {
        return b==0?a:gcd(b,a%b);
}
```

立方根（二分法）

```
   double bottom = 0;
        double top = 0;
        while (top * top * top < num) {  // 比如 5 介于 1^3  和 2^3之间
            top++;
        }
        bottom = top - 1;
```

```
        double mid = bottom + (top - bottom) / 2;
        double mul = mid * mid * mid;
        while (top - bottom > 0.1) {  // 因为只保留一位小数
            if (mul > num) {
                top = mid;
            } else if (mul < num) {
                bottom = mid;
            }
            mid = bottom + (top - bottom) / 2;
            mul = mid * mid * mid;
        }
```

最大正方形

```
    public int maximalSquare(char[][] matrix) {
        int maxSide = 0;
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return maxSide;
        }
        int rows = matrix.length, columns = matrix[0].length;
        int[][] dp = new int[rows][columns];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                if (matrix[i][j] == '1') {
                    if (i == 0 || j == 0) {
                        dp[i][j] = 1;
                    } else {
                        dp[i][j] = Math.min(Math.min(dp[i - 1][j], dp[i][j - 
1]), dp[i - 1][j - 1]) + 1;
                    }
                    maxSide = Math.max(maxSide, dp[i][j]);
                }
            }
        }
        int maxSquare = maxSide * maxSide;
        return maxSquare;
    }
```

单词拆分

```
    public boolean wordBreak(String s, List<String> wordDict) {
        Set<String> wordDictSet = new HashSet(wordDict);
        boolean[] dp = new boolean[s.length() + 1];
        dp[0] = true;
        for (int i = 1; i <= s.length(); i++) {
            for (int j = 0; j < i; j++) {
                if (dp[j] && wordDictSet.contains(s.substring(j, i))) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[s.length()];
```

```
    }
```

爱吃香蕉的珂珂

```
    public int minEatingSpeed(int[] piles, int H) {
        int lo = 1;
        int hi = 1_000_000_000;
        while (lo < hi) {
            int mi = (lo + hi) / 2;
            if (!possible(piles, H, mi))
                lo = mi + 1;
            else
                hi = mi;
        }

        return lo;
    }

    public boolean possible(int[] piles, int H, int K) {
        int time = 0;
        for (int p: piles)
            time += (p-1) / K + 1;
        return time <= H;
    }
```

有效括号（栈）

```
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<Character>();
        for (char c : s.toCharArray()) {
            if (c == '(') stack.push(')');
            else if (c == '[') stack.push(']');
            else if (c == '{') stack.push('}');
            else if (stack.isEmpty() || c != stack.pop()) return false;
        }
        return stack.isEmpty();
    }
```

二叉树展开为链表

```
    public void flatten(TreeNode root) {
        List<TreeNode> list = new ArrayList<TreeNode>();
        zuzhuang(list, root);
        for (int i = 0; i < list.size() - 1; i++) {
            TreeNode first = list.get(i);
            TreeNode second = list.get(i + 1);
            first.left = null;
            first.right = second;
        }
    }
    private void zuzhuang(List<TreeNode> list, TreeNode root) {
        if (root != null) {
```

```
            list.add(root);
            zuzhuang(list, root.left);
            zuzhuang(list, root.right);
        }
    }
```

连接词

```
class Solution {
    Trie trie = new Trie();

    public List<String> findAllConcatenatedWordsInADict(String[] words) {
        List<String> ans = new ArrayList<String>();
        Arrays.sort(words, (a, b) -> a.length() - b.length());
        for (int i = 0; i < words.length; i++) {
            String word = words[i];
            if (word.length() == 0) {
                continue;
            }
            boolean[] visited = new boolean[word.length()];
            if (dfs(word, 0, visited)) {
                ans.add(word);
            } else {
                insert(word);
            }
        }
        return ans;
    }

    public boolean dfs(String word, int start, boolean[] visited) {
        if (word.length() == start) {
            return true;
        }
        if (visited[start]) {
            return false;
        }
        visited[start] = true;
        Trie node = trie;
        for (int i = start; i < word.length(); i++) {
            char ch = word.charAt(i);
            int index = ch - 'a';
            node = node.children[index];
            if (node == null) {
                return false;
            }
            if (node.isEnd) {
                if (dfs(word, i + 1, visited)) {
                    return true;
                }
            }
        }
        return false;
    }

    public void insert(String word) {
```

```
        Trie node = trie;
        for (int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            int index = ch - 'a';
            if (node.children[index] == null) {
                node.children[index] = new Trie();
            }
            node = node.children[index];
        }
        node.isEnd = true;
    }
}

class Trie {
    Trie[] children;
    boolean isEnd;

    public Trie() {
        children = new Trie[26];
        isEnd = false;
    }
}
```

二叉树最小深度（广式搜索）

```
 class MyNode {
        TreeNode treeNode;
        int depth;
        public MyNode(TreeNode treeNode, int depth) {
            this.treeNode = treeNode;
            this.depth = depth;
        }
    }
    public int minDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        Queue<MyNode> queue = new LinkedList();
        int count = 1;
        MyNode myNode = new MyNode(root, 1);
        queue.add(myNode);
        while (!queue.isEmpty()) {
            MyNode node = queue.poll();
            int depth = node.depth;
            if (node.treeNode.left == null && node.treeNode.right == null) {
                queue.clear();
                return depth;
            }

            if (node.treeNode.left != null) {
                queue.add(new MyNode(node.treeNode.left, depth + 1));
            }
            if (node.treeNode.right != null) {
                queue.add(new MyNode(node.treeNode.right, depth + 1));
            }
        }
```

```
        return count;
    }
```

零钱兑换(动态规划)

```
Map<Integer, Integer> resultMap  = null;

    public int coinChange(int[] coins, int amount) {
        resultMap = new HashMap<>(amount);
        if (amount < 1) {
            return 0;
        }
        for (int i = 0; i < coins.length; i++) {
            resultMap.put(coins[i], 1);
        }
        int result = -1;
        for (int i = 1; i <= amount; i++) {
            result = getMinResult(i, coins);
            if (i == amount) {
                return result;
            }
        }
        return result;
    }


    public Integer getMinResult(int num, int[] coins) {
        Integer ret = resultMap.get(num);
        if (ret != null) {
            return ret;
        }
        resultMap.put(num, -1);
        for (int i = 0; i < coins.length; i++) {
            if (num < coins[i]) {
                continue;
            }
            Integer result = resultMap.get(num - coins[i]);
            if (!result.equals(-1)) {
                Integer min = resultMap.get(num);
                if (min.equals(-1)) {
                    resultMap.put(num, result + 1);
                } else {
                    if (min >= (result + 1)) {
                        resultMap.put(num, result + 1);
                    }
                }
            }
        }
        return resultMap.get(num);
    }
```

爬楼梯(动态规划)

```
    Map<Integer, Integer> resultMap = new HashMap<>();

    public int climbStairs(int n) {
        resultMap.put(1, 1);
        resultMap.put(2, 2);
        for (int i = 1; i <= n; i++) {
            int result = getMaxResult(i);
            if (i == n) {
                return result;
            }
        }
        return 0;
    }

    public int getMaxResult(int n) {
        if (n == 1 || n == 2) {
            return resultMap.get(n);
        }
        if (resultMap.get(n) == null) {
            resultMap.put(n, resultMap.get(n - 1) + resultMap.get(n - 2));
        }
        return resultMap.get(n);
    }
```

对称二叉树

```
 public boolean isSymmetric(TreeNode root) {
        return check(root, root);
    }

    public boolean check(TreeNode p, TreeNode q) {
        if (p == null && q == null) {
            return true;
        }
        if (p == null || q == null) {
            return false;
        }
        return p.val == q.val && check(p.left, q.right) && check(p.right,
q.left);
    }
```

四舍五入取整

```
Math.round(input)
```

list反转

```
Collections.reverse(list);
```

合唱团（动态规划）

```
 public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int count = in.nextInt();
        int[] peoples = new int[count];
        for (int i = 0; i < count; i++) {
            peoples[i] = in.nextInt();
        }
        int[] left = getLeftSize(peoples);
        int[] right = getRightSize(peoples);
        List<Integer> list = new ArrayList<>();
        for (int i = 0; i < peoples.length; i++) {
            list.add(left[i] + right[i] - 1);
        }
        Collections.sort(list);
        System.out.println(count  - list.get(list.size() - 1));
    }

private static int[] getRightSize(int[] peoples) {
        int[] right = new int[peoples.length];
        for (int i = peoples.length - 1; i > 0; i--) {
            right[i] = 1;
            for (int j = peoples.length - 1; j > i; j--) {
                if (peoples[j] < peoples[i]) {
                    right[i] = Math.max(right[i], right[j] + 1);
                }
            }
        }
        return right;
    }

    private static int[] getLeftSize(int[] peoples) {
        int ret[] = new int[peoples.length];
        Arrays.fill(ret, 1);
        ret[0] = 1;

        for (int i = 0; i < peoples.length; i++) {
            int num = peoples[i];
            for (int j = 0; j < i; j++) {
                if (num > peoples[j]) {
                    ret[i] = Math.max(ret[i], ret[j] + 1);
                }
            }
        }
        return ret;

    }
```

汽水瓶

```
import java.util.Scanner;
public class Main{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        while(sc.hasNextInt()){
            int bottle = sc.nextInt();
            if(bottle==0){
                break;
            }
            System.out.println(bottle/2);
        }
    }
}
```

表示数字

```
  public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (scanner.hasNext()) {
            String input = scanner.next();
            System.out.println(input.replaceAll("([0-9]+)","*$1*")); //把所有的数字段提取出来，前后加上星号再放回去
        }
    }
```

双指针遍历

```
for (int i = 0; i < input.length(); i++) {
            for (int j = i + 1; j <= input.length(); j++) {
                String str = input.substring(i, j);
            }
}
```

滑动窗口遍历

```
        for (int i = 0; i < input.length; i++) {
            int[] str = Arrays.copyOfRange(input, i, input.length);
            int count = isLiu(str) ;
            if (count > num) {
                num = count;
            }
        }
```

最长覆盖字串（滑动窗口）

```
        int right = 0;
        for (int left = 0; left < s.length(); left++) {
            if (left > 0) {
                // 每当left向右边滑动的时候，删除上一个元素
                removeWord(s.charAt(left - 1));
            }
            boolean isContains = false;
            while (!isContains) {
                isContains = isContains();
                // 如果符合条件，说明找到了最短距离，且可以退出while循环
                if (isContains) {
                    String okStr = s.substring(left, right);
                    if (okStr.length() <= min) {
                        min = okStr.length();
                        minResult = okStr;
                    }
                    break;
                } else {
                    // 如果不符合条件，则窗口范围扩增，也就是right+1
                    if (right != s.length()) {
                        addWord(s.charAt(right));
                        right = right + 1;
                    } else {
                        // 如果窗口范围已经扩张到极限，也退出
                        break;
                    }
                }

            }
        }
```

滑动窗口

239

二分查找模板

```
        int left = 0;
        int right = nums.length - 1;

        int retPos = 0;
        while (left < right) {
            int mid = left + right >> 1;
            if (target <= nums[mid]) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }
```

深度遍历模板

```
        Stack<TreeNode> stack = new Stack<>();
        stack.add(root);
        while (!stack.isEmpty()) {
            TreeNode node = stack.pop();
            list.add(node.val);
            if (node.right != null) {
                stack.push(node.right);
            }
            if (node.left != null) {
                stack.push(node.left);
            }

        }
```

广度遍历模板

```
Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {

            int size = queue.size();
            List<Integer> item = new ArrayList<>();
            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                if (node.left != null) {
                    queue.add(node.left);
                }
                if (node.right != null) {
                    queue.add(node.right);
                }

                item.add(node.val);
            }
            ret.add(item);

        }
```