2017 秋招算法题总结

网易:

> 1 . 问题描述:
> 小易有一些彩色的砖块。每种颜色由一个大写字母表示。各个颜色砖块看起来都完全一样。现在有一个给定的字符串s,s中每个字符代表小易的某个砖块的颜色。小易想把他所有的砖块排成一行。如果最多存在一对不同颜色的相邻砖块,那么这行砖块就很漂亮的。请你帮助小易计算有多少种方式将他所有砖块排成漂亮的一行。(如果两种方式所对应的砖块颜色序列是相同的,那么认为这两种方式是一样的。)
> 例如: s = "ABAB",那么小易有六种排列的结果:
> "AABB","ABAB","ABBA","BAAB","BABA","BBAA"
> 其中只有"AABB"和"BBAA"满足最多只有一对不同颜色的相邻砖块。
> 输入描述:
> 输入包括一个字符串s,字符串s的长度length(1 ≤ length ≤ 50),s中的每一个字符都为一个大写字母(A到Z)。
>
> 输出描述:
> 输出一个整数,表示小易可以有多少种方式。
>
> 输入例子1:
> ABAB
>
> 输出例子1:
> 2

设颜色种数为s,那么
if s > 2 ans = 0
else ans = s

```
#include <bits/stdc++.h>
using namespace std;
string s;
set<char> S;
int main() {
    cin >> s;
    for(int i = 0; i < s.size(); i++) S.insert(s[i]);
    int ans = S.size();
```

```
    if(ans > 2) ans = 0;
    cout << ans << endl;
    return 0;
}
```

2 . 问题描述:
如果一个数列S满足对于所有的合法的i,都有S[i + 1] = S[i] + d, 这里的d也可以是负数和零,我们就称数列S为等差数列。
小易现在有一个长度为n的数列x,小易想把x变为一个等差数列。小易允许在数列上做交换任意两个位置的数值的操作,并且交换操作允许交换多次。但是有些数列通过交换还是不能变成等差数列,小易需要判别一个数列是否能通过交换操作变成等差数列
输入描述:
输入包括两行,第一行包含整数n(2 ≤ n ≤ 50),即数列的长度。
第二行n个元素xi,即数列中的每个整数。

输出描述:
如果可以变成等差数列输出"Possible",否则输出"Impossible"。

输入例子1:
3
3 1 2

输出例子1:
Possible

对序列排序,然后比对一下是否等差即可。

```
#include <bits/stdc++.h>
using namespace std;
int n;
int x[55];
string solve() {
    sort(x, x + n);
    if(n <= 2) return "Possible";
    else {
        int d = x[1] - x[0];
        bool ok = 1;
        for(int i = 1; ok && i < n; i++) {
            if(d != x[i] - x[i - 1]) return "Impossible";
        }
        return "Possible";
    }
}
int main() {
```

```
    cin >> n;
    for(int i = 0; i < n; i++) cin >> x[i];
    cout << solve() << endl;
    return 0;
}
```

3 . 问题描述:
如果一个01串任意两个相邻位置的字符都是不一样的,我们就叫这个01串为交错01串。例如: "1","10101","0101010"都是交错01串。
小易现在有一个01串s,小易想找出一个最长的连续子串,并且这个子串是一个交错01串。小易需要你帮帮忙求出最长的这样的子串的长度是多少。

输入描述:
输入包括字符串s,s的长度length(1 ≤ length ≤ 50),字符串中只包含'0'和'1'

输出描述:
输出一个整数,表示最长的满足要求的子串长度。

输入例子1:
111101111

输出例子1:
3

```
#include <bits/stdc++.h>
using namespace std;
string s;
int main() {
    cin >> s;
    int ans = 1, cnt = 1;
    for(int i = 1; i < s.size(); i++) {
        if(s[i] != s[i - 1]) {
            cnt++;
        } else {
            cnt = 1;
        }
        ans = max(ans, cnt);
    }
    cout << ans << endl;
    return 0;
}
```

4 . 问题描述:

小易有一个长度为n的整数序列,a_1,...,a_n。然后考虑在一个空序列b上进行n次以下操作:
1、将a_i放入b序列的末尾
2、逆置b序列
小易需要你计算输出操作n次之后的b序列。
输入描述:
输入包括两行,第一行包括一个整数n($2 \le n \le 2*10^5$),即序列的长度。
第二行包括n个整数a_i($1 \le a_i \le 10^9$),即序列a中的每个整数,以空格分割。

输出描述:
在一行中输出操作n次之后的b序列,以空格分割,行末无空格。

输入例子1:
4
1 2 3 4

输出例子1:
4 2 1 3

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2e5 + 5;
int a[maxn];
int n;
int main() {
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) scanf("%d", &a[i]);
    if(n % 2) {
        for(int i = n; i >= 1; i -= 2) printf("%d ", a[i]);
        for(int i = 2; i <= n; i += 2) i == n - 1 ? printf("%d", a[i]) : printf("%d
 ", a[i]);
    } else {
        for(int i = n; i >= 1; i -= 2) printf("%d ", a[i]);
        for(int i = 1; i <= n; i += 2) i == n - 1 ? printf("%d", a[i]) : printf("%d
 ", a[i]);
    }
    printf("\n");
    return 0;
}
```

5 . 问题描述:
小易为了向他的父母表现他已经长大独立了,他决定搬出去自己居住一段时间。一个人生活

增加了许多花费: 小易每天必须吃一个水果并且需要每天支付x元的房屋租金。当前小易手中已经有f个水果和d元钱,小易也能去商店购买一些水果,商店每个水果售卖p元。小易为了表现他独立生活的能力,希望能独立生活的时间越长越好,小易希望你来帮他计算一下他最多能独立生活多少天。

输入描述:
输入包括一行,四个整数x, f, d, p(1 ≤ x,f,d,p ≤ $2 * 10^9$),以空格分割

输出描述:
输出一个整数, 表示小易最多能独立生活多少天。

输入例子1:
3 5 100 10

输出例子1:
11

```
import java.util.Scanner;

/**
 * designed by Steve Ke on 2017/8/12.
 *
 * @author Steve Ke
 *         e-mail  huangke7296@foxmail.com
 *         github  https://github.com/KoreHuang
 *         oschina https://git.oschina.net/steveKe
 * @version JDK 1.8.0_111
 * @since 2017/8/12
 */
public class Main {

    public static int solve(int x,int f,int d,int p){

        if(f<=0)return 0;

        int last = d-f*x;    //剩余可以完全用来支付水果和房租的钱

        int result = last/(p+x);


        if(result<0) return f;

        return f+result;
    }
```

```
    public static void main(String []args){
        Scanner scanner=new Scanner(System.in);

        int x,f,d,p;

        x=scanner.nextInt();    //每天支付 X 的租金
        f=scanner.nextInt();    //当前已有 F 个水果
        d=scanner.nextInt();    //当前已有的 D 元钱
        p=scanner.nextInt();    //商店每个水果 P 元钱

        System.out.println(solve(x,f,d,p));

    }
}
```

6 . 问题描述:
小易将n个棋子摆放在一张无限大的棋盘上。第i个棋子放在第x[i]行y[i]列。同一个格子允许放置多个棋子。每一次操作小易可以把一个棋子拿起并将其移动到原格子的上、下、左、右的任意一个格子中。小易想知道要让棋盘上出现有一个格子中至少有i(1 ≤ i ≤ n)个棋子所需要的最少操作次数.

输入描述:
输入包括三行,第一行一个整数n(1 ≤ n ≤ 50),表示棋子的个数
第二行为n个棋子的横坐标xi
第三行为n个棋子的纵坐标yi

输出描述:
输出n个整数,第i个表示棋盘上有一个格子至少有i个棋子所需要的操作数,以空格分割。行末无空格

如样例所示:
对于1个棋子: 不需要操作
对于2个棋子: 将前两个棋子放在(1, 1)中
对于3个棋子: 将前三个棋子放在(2, 1)中
对于4个棋子: 将所有棋子都放在(3, 1)中

输入例子1:
4
1 2 4 9
1 1 1 1

输出例子1:

0 1 3 10

```cpp
#include <bits/stdc++.h>
using namespace std;
const int inf = 1e9;
int n, x[55], y[55];
int main() {
    int n;
    cin >> n;
    for(int i = 0; i < n; i++) cin >> x[i];
    for(int i = 0; i < n; i++) cin >> y[i];
    vector<int> res(n, inf);
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            for(int k = 0; k < n; k++) {
                vector<int> res2(n);
                for(int l = 0; l < n; l++) {
                    res2[l] = abs(x[l] - x[j]) + abs(y[l] - y[k]);
                }
                sort(res2.begin(), res2.end());
                int res3 = 0;
                for(int l = 0; l < i + 1; l++) res3 += res2[l];
                res[i] = min(res[i], res3);
            }
        }
    }
    for(int i = 0; i < n; i++) {
        i == 0 ? cout << res[i] : cout << " " << res[i];
    }
    return 0;
}
```

7 . 问题描述:
小易老师是非常严厉的,它会要求所有学生在进入教室前都排成一列,并且他要求学生按照身高不递减的顺序排列。有一次,n个学生在列队的时候,小易老师正好去卫生间了。学生们终于有机会反击了,于是学生们决定来一次疯狂的队列,他们定义一个队列的疯狂值为每对相邻排列学生身高差的绝对值总和。由于按照身高顺序排列的队列的疯狂值是最小的,他们当然决定按照疯狂值最大的顺序来进行列队。现在给出n个学生的身高,请计算出这些学生列队的最大可能的疯狂值。小易老师回来一定会气得半死。

输入描述:
输入包括两行,第一行一个整数n(1 ≤ n ≤ 50),表示学生的人数
第二行为n个整数hi,表示每个学生的身高

输出描述:
输出一个整数,表示n个学生列队可以获得的最大的疯狂值。

如样例所示:
当队列排列顺序是: 25-10-40-5-25, 身高差绝对值的总和为15+30+35+20=100。
这是最大的疯狂值了。

输入例子1:
5
5 10 25 40 25

输出例子1:
100

```
#include <bits/stdc++.h>
using namespace std;
int h[55];
int n;
int main() {
    cin >> n;
    for(int i = 0; i < n; i++) cin >> h[i];
    sort(h, h + n);
    int tmp, ans = 0, ans1 = 0, ans2 = 0;
    if(n % 2 == 0) {
        tmp = n / 2;
        for(int i = 0; i < tmp; i++) {
            ans += 2 * (h[tmp + i] - h[i]);
        }
        ans += h[tmp - 1] - h[tmp];
        cout << ans << endl;
        return 0;
    } else {
        tmp = n / 2;
        for(int i = 0; i < tmp; i++) {
            ans1 += 2 * (h[tmp + 1 + i] - h[i]);
            ans2 += 2 * (h[tmp + 1 + i] - h[i]);
        }
        ans1 += -h[tmp] + h[tmp - 1];
        ans2 += h[n] - h[n + 1];
        cout << max(ans1, ans2) << endl;
        return 0;
    }
    return 0;
}
```

8 . 问题描述:
小易非常喜欢拥有以下性质的数列:
1、数列的长度为n
2、数列中的每个数都在1到k之间(包括1和k)
3、对于位置相邻的两个数A和B(A在B前),都满足(A <= B)或(A mod B != 0)(满足其一即可)
例如,当n = 4, k = 7
那么{1,7,7,2},它的长度是4,所有数字也在1到7范围内,并且满足第三条性质,所以小易是喜欢这个数列的
但是小易不喜欢{4,4,4,2}这个数列。小易给出n和k,希望你能帮他求出有多少个是他会喜欢的数列。
输入描述:
输入包括两个整数n和k($1 \le n \le 10, 1 \le k \le 10^5$)

输出描述:
输出一个整数,即满足要求的数列个数,因为答案可能很大,输出对1,000,000,007取模的结果。

输入例子1:
2 2

输出例子1:
3

```
//dp[j][i]表示长度为i最后一个数是j的小易喜欢的数列的数量

#include <bits/stdc++.h>
using namespace std;
const int mod = 1e9 + 7;
const int maxn = 1e5 + 5;
int dp[maxn][15];
int n, k;
int main() {
    cin >> n >> k;
    dp[1][0] = 1;
    for(int i = 1; i <= n; i++) {
        int sum = 0;
        for(int j = 1; j <= k; j++) {
            sum += dp[j][i - 1];
            sum %= mod;
        }
        for(int j = 1; j <= k; j++) {
            int sum2 = 0;
```

```
            for(int z = j + j; z <= k; z += j) {
                sum2 += dp[z][i - 1];
                sum2 %= mod;
            }
            dp[j][i] = (sum - sum2 + mod) % mod;
        }
    }
    int ans = 0;
    for(int j = 1; j <= k; j++) {
        ans += dp[j][n];
        ans %= mod;
    }
    cout << ans << endl;
    return 0;
}
```

滴滴

1 . 问题描述:
一个单链表，其中除了next指针外，还有一个random指针，指向链表中的任意某个元素。
如何复制这样一个链表
通过next来复制一条链是很容易的，问题的难点在于如何恰当地设置新链表中的random指针。

解法:
（1）使用Hash表的做法，先依次遍历原链表，每经过一个节点X，开辟一个新节点Y，然后（key=X的地址，value=Y的地址）存入哈希表。第二次再遍历原链表，根据拓扑结构设置新的链表。需要O(n)的空间，时间也是O(n)。

如果不使用额外的空间，那么要想在旧链表和新链表的对应节点之间建立联系。就要利用链表中多余的指针。

```
public class CopySpecialLinkedList {

    /**
     * 题目：有一个特殊的链表，其中每个节点不但有指向下一个节点的指针pNext，还有一个指向链表中任意节点的指针pRand，如何拷贝这个特殊链表?
拷贝pNext指针非常容易，所以题目的难点是如何拷贝pRand指针。
假设原来链表为A1 -> A2 ->... -> An，新拷贝链表是B1 -> B2 ->...-> Bn。
为了能够快速的找到pRand指向的节点，并把对应的关系拷贝到B中。我们可以将两个链表合并成
A1 -> B1 -> A2 -> B2 -> ... -> An -> Bn。
从A1节点出发，很容易找到A1的pRand指向的节点Ax,然后也就找到了Bx，将B1的pRand指向Bx也就完成了B1节点pRand的拷贝。依次类推。
```

```
当所有节点的pRand都拷贝完成后，再将合并链表分成两个链表就可以了。
     */

 public static void main(String[] args) {
        int[] data = { 1, 2, 3, 4, 5 };
        SpecialLinkList sList = new SpecialLinkList();
        SpecialLinkList.Node source = sList.create(data);
        sList.pRand(0, 4);//add 'pRand'
        sList.pRand(1, 3);
        sList.pRand(2, 0);
        sList.pRand(3, 2);
        sList.pRand(4, 1);
        sList.print(source);
        SpecialLinkList.Node dest = sList.copyPNext(source);
        sList.copyPRand(dest, source);
        sList.print(dest);
    }

}

class SpecialLinkList {

    private Node head;

    class Node {
        int val;
        Node pNext;
        Node pRand;

        Node(int val) {
            this.val = val;
        }
    }

    public Node copyPNext(Node source) {
        Node pre = null;// previous node
        Node dest = null;// destination node
        while (source != null) {
            int val = source.val;
            Node cur = new Node(val);// current node
            if (pre == null) {
                pre = cur;
                dest = pre;
            } else {
                pre.pNext = cur;
                pre = cur;
            }
            source = source.pNext;
```

```
        }
        return dest;
    }

    public Node copyPRand(Node dest, Node source) {
        Node a = source;
        Node b = dest;
        // create a1-b1-a2-b2...
        while (a != null && b != null) {
            Node tmp = a.pNext;
            a.pNext = b;
            a = b;
            b = tmp;
        }
        // copy pRand
        a = source;
        b = a.pNext;
        while (a.pNext != null) {
            Node tmp = a.pNext.pNext;
            b.pRand = a.pRand.pNext;
            if (tmp == null) {
                break;
            }
            a = tmp;
            b = a.pNext;
        }
        // split a1-b1-a2-b2... to a1-a2...,b1-b2....
        a = source;
        b = source.pNext;
        dest = b;
        while (a.pNext.pNext != null) {
            Node tmp = a.pNext.pNext;
            a.pNext = tmp;
            a = tmp;

            tmp = b.pNext.pNext;
            b.pNext = tmp;
            b = tmp;
        }
        a.pNext = null;
        b.pNext = null;
        return dest;
    }

    // create a linked list.Insert from tail.
    public Node create(int[] data) {
        if (data == null || data.length == 0) {
            return null;
```

```
        }
        Node tail = null;
        for (int len = data.length, i = len - 1; i >= 0; i--) {
            Node tmp = new Node(data[i]);
            tmp.pNext = tail;
            tail = tmp;
        }
        head = tail;
        return head;
    }

    // create 'pRand' between posX and posY
    public void pRand(int posX, int posY) {
        if (posX < 0 || posY < 0) {
            return;
        }
        Node nodeX = getNodeAt(posX);
        Node nodeY = getNodeAt(posY);
        if (nodeX != null && nodeY != null) {
            nodeX.pRand = nodeY;
        }
    }

    // get the node at the specific position.The position starts from 0.

    public Node getNodeAt(int pos) {
        if (pos < 0) {
            return null;
        }
        if (head == null) {
            return null;
        }
        Node node = head;
        while (node != null && pos > 0) {
            node = node.pNext;
            pos--;
        }
        return node;
    }
    //print the special linked list,like 1(5)-2(4)-3(1)-4(3)-5(2).'5' is the pRand
of '1',and so on.

    public void print(Node node) {
        while (node != null) {
            System.out.print(node.val + "");
            if (node.pRand != null) {
                System.out.print("(" + node.pRand.val + ")-");
            }
```

```
            node = node.pNext;
        }
        System.out.println();
    }

}
```

2 . 问题描述:
题意：从给定的数组中找三个数，让它们的和为0。输出所有可能。
如[1,3,-1,0,-3],那么输出[1,-1,0],[3,0,-3]。

```
public class Main {
    public static void main(String[] args){
        int[] array = {-1, 0,1,2,-1,-4};
        print3Num(array, 0);
    }
    public static void print3Num(int[] array,int k){
        Arrays.sort(array);
        int length = array.length;
        if(array == null||array.length<=0){
             return ;
        }
        for(int i = 0;i < length;i++){
            if(i<length-1&&array[i] == array[i+1]){
                continue;
            }
            int num = k-array[i];

            printK(array,i,length-1,array[i],num);
        }


    }
    public static void printK(int[] array,int start,int end, int num,int k){

        while(start < end){
            while(array[start] == array[start+1]){
                start++;
            }
            while(array[end] == array[end-1]){
                end--;
            }
            if(array[start] + array[end] == k){
                System.out.println(num+" "+array[start]+" "+array[end]);
                start ++;
            }
```

```
            if(array[start]+array[end] < k){
                start++;
            }
            if(array[start]+array[end] > k){
                end--;
            }
        }
    }
}
```

3 . 问题描述:
最长递增子序列（longest increasing subsequence）

```
/**
 * designed by Steve Ke on 2017/8/12.
 *
 * @author Steve Ke
 *         e-mail  huangke7296@foxmail.com
 *         github  https://github.com/KoreHuang
 *         oschina https://git.oschina.net/steveKe
 * @version JDK 1.8.0_111
 * @since 2017/9/12
 */
public class LIS {
    public static int lis(int[] arr){
        if(arr == null || arr.length == 0)
            return 0;
        return lis(arr, arr.length);
    }

    private static int lis(int[] arr, int length){
        int lis[] = new int[length];

        //init
        for(int i = 0; i < length; i++)
            lis[i] = 1;

        for(int i = 1; i < length; i++)
        {
            for(int j = 0; j < i; j++)
            {
                if(arr[i] > arr[j] && lis[j] + 1 > lis[i])
                    lis[i] = lis[j] + 1;
            }
        }

        int max = lis[0];
```

```
        for(int i = 1; i < length; i++)
            if(max < lis[i])
                max = lis[i];
        return max;
    }

    public static void main(String[] args) {
        int[] arr = {3,1,4,1,5,9,2,6,5};
        int result = lis(arr);
        System.out.println(result);
    }
```

腾讯

> 1 . 问题描述:
> 有n级楼梯，你一次可以爬一级或两级，问爬上n级楼梯有多少种爬法

```
public class Main {
    public static void main(String[] args){
        System.out.print(climbStair(5));
    }
    //递归
    public static int climbStairs(int n) {
        if(n == 1){
            return 1;
        }
        if(n == 2){
            return 2;
        }
        return climbStairs(n-1)+climbStairs(n-2);
    }
    //非递归
    public static int climbStair(int n) {
        int num1 = 1;
        int num2 = 2;
        int sum = 0;
        if(n == 1){
            return 1;
        }
        if(n == 2){
            return 2;
        }
        for(int i = 2;i < n;i++){
            sum = num1+num2;
            num1 = num2;
```

```
            num2 = sum;
        }
        return sum;
    }

}
```

> 2 . 问题描述:
> 验证括号是否能正常闭合，如“()[]{}”是可以正常闭合的，但“(]”就是不正常的。

```
public class Main {
    public static void main(String[] args){
        System.out.println(valid("["));
    }
    public static boolean valid(String str){
        if(str.length() <= 0){
            return false;
        }
        Stack<Character> stack = new Stack<Character>();
        int length = str.length();
        for(int i = 0;i < length;i++){
            if(str.charAt(i) == '{'||str.charAt(i) == '['||str.charAt(i)=='('){
                stack.push(str.charAt(i));
            }
            if(str.charAt(i)==')'){
                if(!stack.isEmpty()){
                char c = stack.pop();
                if(c != '('){
                    return false;
                }
                }else{
                    return false;
                }
            }
            if(str.charAt(i)=='}'){
                if(!stack.isEmpty()){
                char c = stack.pop();
                if(c != '{'){
                    return false;
                }
                }else{
                    return false;
                }
            }
            if(str.charAt(i)==']'){

                if(!stack.isEmpty()){
```

```
                    char c = stack.pop();
                if(c != '['){
                    return false;
                }
            }else{
                    return false;
                }
            }
        }
        if(stack.isEmpty())
        return true;
        else{
            return false;
        }
    }
}
```

美团

> 1 . 问题描述:
> 堆排序

```
public class HeapSortTest {

    public static void main(String[] args) {
        int[] data5 = new int[] { 5, 3, 6, 2, 1, 9, 4, 8, 7 };
        print(data5);
        heapSort(data5);
        System.out.println("排序后的数组：");
        print(data5);
    }

    public static void swap(int[] data, int i, int j) {
        if (i == j) {
            return;
        }
        data[i] = data[i] + data[j];
        data[j] = data[i] - data[j];
        data[i] = data[i] - data[j];
    }

    public static void heapSort(int[] data) {
        for (int i = 0; i < data.length; i++) {
            createMaxdHeap(data, data.length - 1 - i);
            swap(data, 0, data.length - 1 - i);
```

```
            print(data);
        }
    }

    public static void createMaxdHeap(int[] data, int lastIndex) {
        for (int i = (lastIndex - 1) / 2; i >= 0; i--) {
            // 保存当前正在判断的节点
            int k = i;
            // 若当前节点的子节点存在
            while (2 * k + 1 <= lastIndex) {
                // biggerIndex总是记录较大节点的值,先赋值为当前判断节点的左子节点
                int biggerIndex = 2 * k + 1;
                if (biggerIndex < lastIndex) {
                    // 若右子节点存在，否则此时biggerIndex应该等于 lastIndex
                    if (data[biggerIndex] < data[biggerIndex + 1]) {
                        // 若右子节点值比左子节点值大，则biggerIndex记录的是右子节点的值

                        biggerIndex++;
                    }
                }
                if (data[k] < data[biggerIndex]) {
                    // 若当前节点值比子节点最大值小，则交换2者得值，交换后将biggerIndex
值赋值给k
                    swap(data, k, biggerIndex);
                    k = biggerIndex;
                } else {
                    break;
                }
            }
        }
    }

    public static void print(int[] data) {
        for (int i = 0; i < data.length; i++) {
            System.out.print(data[i] + "\t");
        }
        System.out.println();
    }

}
```

2 . 问题描述:
单链表的逆置

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
```

```
using namespace std;

struct ListNode
{
    int m_nKey;
    ListNode* m_pNext;
};

//构造链表
void CreateList(ListNode *&pHead)
{
    fstream fin("list.txt");
    ListNode *pNode = NULL;
    ListNode *pTmp = NULL;
    int data;
    fin>>data;
    while (data)
    {
        pNode = new ListNode;
        pNode->m_nKey = data;
        pNode->m_pNext = NULL;
        if (NULL == pHead)
        {
            pHead = pNode;
            pTmp = pNode;
        }
        else
        {
            pTmp->m_pNext = pNode;
            pTmp = pNode;
        }

        fin>>data;
    }
}

//翻转单链表
void ReverseLink(ListNode *&pHead)
{
    if (NULL == pHead)
    {
        return;
    }
    ListNode *pNode = pHead;
    ListNode *Prev = NULL;
    ListNode *pNext = NULL;
    while (NULL != pNode)
```

```
    {
        pNext = pNode->m_pNext;
        if (NULL == pNext)
        {
            pHead = pNode;
        }
        pNode->m_pNext = Prev;
        Prev = pNode;
        pNode = pNext;
    }
}

void PrintList(ListNode *pHead)
{
    if (NULL == pHead)
    {
        return;
    }
    ListNode *pNode = pHead;
    while (NULL != pNode)
    {
        cout<<pNode->m_nKey<<"  ";
        pNode = pNode->m_pNext;
    }
    cout<<endl;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ListNode *pHead = NULL;
    cout<<"原来的链表：";
    CreateList(pHead);
    PrintList(pHead);
    ReverseLink(pHead);
    cout<<"翻转的链表：";
    PrintList(pHead);

    return 0;
}
```

3．问题描述:
打印蛇形矩阵

```
import java.util.Scanner;

public class mysnakematrix {
    private int n; //
```

```
    private int a[][]; // 声明一个矩阵
    private int value = 1;// 矩阵里数字的值

    public mysnakematrix(int i) {
        this.n = i;
        a = new int[n][n];
    }

    // 计算第m层左上角的数字
    private int getcorner(int m) {
        int corner = 1;
        int o = n - 1;
        for (int i = 0; i < m - 1; ++i) {
            corner += 4 * o;
            o = o - 2;
        }
        return corner;
    }

    // 生成矩阵的每一层的每一边的数
    // s表示4个方向，分别取值1,2,3,4，表示4个不同的方向。
    // o表示这条边的起始值。
    // x表示第m层每条边的数字个数
    private void side(int s, int o, int x, int m) {
        int i = 0;
        int j = 0;
        switch (s) {
        case 1:
            i = m - 1;
            j = m - 1;
            for (int k = 0; k < x; ++k) {
                a[i][j + k] = value;
                ++value;
            }

            break;
        case 2:
            i = m - 1;
            j = m - 1 + x;
            for (int k = 0; k < x; ++k) {
                a[i + k][j] = value;
                ++value;
            }
            break;
        case 3:
            i = m - 1 + x;
            j = m - 1 + x;
            for (int k = 0; k < x; ++k) {
```

```
                a[i][j - k] = value;
                ++value;
            }
            break;
        case 4:
            i = m - 1 + x;
            j = m - 1;
            for (int k = 0; k < x; ++k) {
                a[i - k][j] = value;
                ++value;
            }
            break;
        }
    }

    // 生成蛇形矩阵的第m层
    private void shell(int m)// m表示第m层
    {
        int x = n - 1 - (m - 1) * 2; // x表示第m层每条边的数字个数
        int o = getcorner(m);
        int o1 = o;
        int o2 = o1 + x;
        int o3 = o2 + x;
        int o4 = o3 + x;
        // System.out.println(o4);

        side(1, o, x, m);
        side(2, o, x, m);
        side(3, o, x, m);
        side(4, o, x, m);
    }

    // 生成蛇形矩阵
    public void snakeMatrix() {
        int m = (n + 1) / 2;// 计算一共有多少层
        for (int i = 1; i <= m; ++i) {

            shell(i);
        }
        if (n % 2 == 1) {
            a[n / 2][n / 2] = n * n;
        }

    }

    // 打印矩阵
    public void print() {
        for (int i = 0; i < n; ++i) {
```

```
            for (int j = 0; j < n; ++j) {
                if (a[i][j] < 10) {
                    System.out.print(a[i][j] + "  ");
                } else {
                    System.out.print(a[i][j] + " ");
                }

            }
            System.out.println();
        }
    }

    public static void main(String args[]) {
        mysnakematrix my = new mysnakematrix(new Scanner(System.in).nextInt());  //
利用Scanner获取控制台输入
        my.snakeMatrix();
        my.print();
    }
}
```