数据结构与算法(JAVA版)

一、动态数组(ArrayList)

```
import java.util.ArrayList;

public class Demo {
    public static void main(String[] args) {
        ArrayList<String> str = new ArrayList<String>();
        str.add("1");       //数组添加元素
        str.add("2");
        str.add("3");
        String s = str.get(1);    //得到数组中相应下标的元素
        System.out.println(s);
        boolean e = str.contains("1");    //判断数组中是否还有该元素
        boolean r = str.contains("0");
        System.out.println(e);
        System.out.println(r);
        int length = str.size();        //得到数组的长度
        System.out.println(length);
        boolean t = str.isEmpty();           //判断数组是否为空
        System.out.println(t);
        str.remove(1);          //清除掉数组中对应下标的元素
        str.add(1,"2");     //在数组的特定位置添加元素
        int y = str.indexOf("2");       //得到数组中对应元素的下标
        System.out.println(y);
```

```
        str.set(0,"2");          //改变数组中对应下标的元素值
        str.clear();             //清空数组内的所有元素
    }
}
```

二、链表（LinkedList）

```
import java.util.LinkedList;

public class Demo {
    public static void main(String[] args) {
        LinkedList<Integer> ll = new LinkedList<Integer>();
        ll.add(1);           //链表添加元素
        ll.add(2);
        ll.add(3);
        ll.addFirst(0);      //在链表的开头添加元素
        ll.addLast(4);       //在链表的结尾添加元素
        ll.add(1,2);     //在链表的特定位置插入元素
        Boolean  y = ll.contains(3);          //判断链表中是否含有该元素
        ll.remove(3);        //移除链表中特定下标的元素
        int r = ll.get(2);           //得到链表中特定下标的元素
        int u = ll.getFirst();       //得到链表中的第一个元素
        int i = ll.getLast();        //得到链表中的最后一个元素
        ll.offer(5);             //在链表的尾部添加元素
        ll.indexOf(1);               //返回一个元素在链表中第一次出现时的下标
        ll.lastIndexOf(2);            //返回一个元素在链表中第一次出现时的下标
        boolean p = ll.isEmpty();              //判断链表是否为空
        ll.offerFirst(6);            //在链表的开头添加元素
        ll.offerLast(8);             //在链表的结尾添加元素
        int k = ll.peek();               //得到链表的第一个元素
        ll.poll();                       //得到并清除链表的第一个元素
        ll.pop();                        //返回并清除链表的第一个元素
        ll.push(4);                  //将元素插入到链表的头部
        int e = ll.size();               //得到链表的长度
        System.out.println(e);
        ll.clear();      //清空链表
    }
}
```

三、栈和队列

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class Demo {
```

```
    public static void main(String[] args) {
        Stack<Integer> st = new Stack<Integer>();//建立一个栈
        st.push(1);     //入栈
        st.push(2);
        st.push(3);
        int r = st.pop();           //出栈
        int e = st.peek();          //得到栈顶元素
        boolean k =  st.isEmpty();      //判断栈是否为空
        int length = st.size();         //得到栈的长度
        Queue<Integer> que = new LinkedList<Integer>(); //建立一个队列
        que.offer(1);           //进队列
        que.offer(2);
        que.offer(3);
        int s = que.poll();             //出队列
    }
}
```

四、树与二叉树

五、图

六、数组排序和集合排序

```
import java.util.*;


public class Demo {
    public static void main(String[] args) {
        int n;
        Integer[] arr = {3,1,2,5,4};
        Arrays.sort(arr,new Comparator<Integer>(){          //默认是升序，这里将排序
函数重载为降序
            public int compare(Integer o1,Integer o2) {
                return o2 - o1;
            }
        });
        for(int e:arr) {
            System.out.print(e+" ");
        }
        System.out.println();

        TreeSet<Integer> set = new TreeSet<Integer>(new Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {        //默认是升序，这里将
排序函数重载为降序
```

```
                return o2 - o1;
            }
        });
        set.add(10);
        set.add(6);
        set.add(8);
        set.add(4);
        set.add(9);
        set.add(5);
        set.add(2);
        set.add(7);
        for(int e:set) {
            System.out.print(e+" ");
        }
    }
}
```

七、进制转换

```
import java.util.Scanner;

public class Demo {
    public static void main(String[] args) {
        // 如果题目要进行转化的进制在2~36之间的话直接调用库函数就行了
        Scanner in = new Scanner(System.in);
        String s;
        s = in.next();
        int b = Integer.parseInt(s,16);// 将16进制的字符串转化十进制数
        //BigInteger a = new BigInteger(s, 16);// 高精度数
        String str = Integer.toString(b,8);//将十进制的数转换为8进制
        System.out.println(b);
        System.out.println(str);
    }
}
```

```
//任意进制转为十进制
    public static int Int(char c)
    {
        if(c >= '0' && c <= '9')
        {
            return c - '0';
        }else {
            return c >= 'a' && c <= 'z' ? c - 'a' + 10 : c - 'A' + 10;
        }

    }
```

```
    public static long ToTen(String x, int p) throws IOException {
        long y = 0, b = 1;
        for(int i = x.length() - 1; i >= 0; i--)
        {
            y += Int(x.charAt(i))*b;
            b *= p;
        }
        return y;
    }
```

```
//十进制转为任意进制
public static String f(int c)
{
    if(c >= 0 && c <= 9) return  String.valueOf(c);
    return c >= 0 && c <= 9 ? String.valueOf(c) : String.valueOf((char)('A' + c -
10));
}

public static String ToP(int x, int p) throws IOException{
        String y = "";

    while(true)
    {
        y = f(x % p) + y;
        x /= p;

                if(x == 0) break;
        }

        return y;
}
```

八、数字全排列（DFS算法）

```
import java.util.Scanner;

public class Main {
    static final int N = 9;
    static int num = 0;
    static int[] arr = new int[N];
    static boolean[] st = new boolean[N];
    public static void dfs(int n) {
        if(n==N) {
            for(int i=1;i<N;i++)
            {
                System.out.print(String.format("%d ",arr[i]));

            }
```

```
            num++;
            System.out.println();
        }
        for(int i = 1;i<N;i++) {
            if(!st[i]) {
                arr[n] = i;
                st[i] = true;
                dfs(n+1);
                st[i] = false;
                arr[n] = 0;
            }
        }
    }
    public static void main(String[] args) {
        dfs(1);
        System.out.println(num);
    }
}
```

九、欧拉筛素数

```
import java.util.Scanner;

public class Main{
        static final int N = 10000005;
    static int[] primes = new int[N];//存放每个素数的值
    static int[] st = new int[N];//

    static int cnt = 0;//计算素数的个数
    public static void ola(int n){

        for(int i = 2; i <= n; i++)
        {
            if(st[i] == 0)      primes[cnt++] = i;

            for(int j = 0; primes[j] <= n / i; j++)
            {
                st[primes[j]*i] = 1;
                if(i % primes[j] == 0) break;
            }
        }
    }
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                int a;
                a = in.nextInt();
                ola(a);
                System.out.println(cnt);
                for(int i=0;i<cnt;i++) {
                        System.out.print(String.format("%d ",primes[i]));
                }
        }
```

```
}
```

十、埃氏筛素数

```
import java.util.Scanner;

public class Main{
        static final int N = 10000005;
    static int[] primes = new int[N];//存放每个素数的值
    static boolean[] st = new boolean[N];//

    static int cnt = 0;//计算素数的个数
    public static void get_prime(int n){
      for(int i=2;i<=n;i++) {
          if(!st[i]) {
                  primes[cnt++] = i;
                  for(int j = i * i; j <= n; j += i)
                          st[j] = true; // j是i的一个倍数，j是合数，筛掉。
          }
    }
 }
        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                int a;
                a = in.nextInt();
                get_prime(a);
                System.out.println(cnt);
                for(int i=0;i<cnt;i++) {
                        System.out.print(String.format("%d ",primes[i]));
                }
        }
}
```

十一、判断素数(三种方法)

```
import java.util.Scanner;

public class Main{
    //第一种方法(基础)
    public static boolean isPrime_1(long number){
        long i;
        boolean flag = false;
```

```
        if(number == 1){
            flag = false;
        }
        else{
            for(i = 2; i < number; i++){
                if(number%i==0){
                    flag = false;
                    break;
                }
            }
            if(i==number){
                flag = true;
            }
        }
        return flag;
    }
    //第二种方法(改进)
    public static boolean isPrime_2(long number){
        long i;
        boolean flag = false;
        if(number == 1){
            flag = false;
        }
        else{
            for(i = 2; i <= number/2; i++){
                if(number%i==0){
                    flag = false;
                    break;
                }
            }
            if(i>number/2){
                flag = true;
            }
        }
        return flag;
    }

    //第三种方法(高效率)
    public static boolean isPrime_3(long number){
        long i;
        boolean flag = false;
        if(number==1){
            return false;
        }else{
            for (i = 2; i <= Math.sqrt(number); i++) {
                if(number%i==0){
                    flag = false;
                    break;
                }
            }
            if(i>Math.sqrt(number)){
                flag = true;
            }
        }
        return flag;
    }
    public static void main(String[] args){
```

```
        Scanner in = new Scanner(System.in);
        long number = in.nextLong();
        System.out.println(isPrime_1(number));
        System.out.println(isPrime_2(number));
        System.out.println(isPrime_3(number));
    }
}
```

十二、数字三角形(动态规划算法)

```
import java.util.Scanner;


public class Demo {
//    public static int unique(int n,int m) {
//
//    }
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        int[][] arr = new int[N][N];
        int[][] dp = new int[N][N];

        for(int i=0;i<N;i++) {
            for(int j=0;j<=i;j++) {
                arr[i][j] = sc.nextInt();
            }
        }
        sc.close();

        dp[0][0] = arr[0][0];       //初始化dp数组

        for(int i=1;i<N;i++) {
            dp[i][0] = dp[i-1][0] + arr[i][0];
        }

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

        if(N%2!=0) {
            System.out.println(dp[N-1][N/2]);
        }else {
            System.out.println(Math.max(dp[N-1][N/2], dp[N-1][N/2-1]));
        }
//        Scanner in = new Scanner(System.in);
//        int m,n;
//        n = in.nextInt();
//        m = in.nextInt();
```

```
    }
}
```

十三、冒泡排序(升序)

```
import java.util.Arrays;
import java.util.Scanner;

public class Main() {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        int[] arr = new int[7];
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

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

        for (int k = 0; k < arr.length; k++) {
            System.out.print(arr[k] + "");
        }
    }
}
```

十四、快速排序(升序)

```
import java.util.Arrays;
import java.util.Scanner;

public class Main(){
    public static void main(String[] args){

        Scanner in = new Scanner(System.in);
        int[] arr = new int[7];
```

```
        for (int i = 0; i < arr.length; i++) {
            arr[i] = scanner.nextInt();
        }
        scanner.close();

        quickSort(arr,0,arr.length-1);

        for(int i = 0;i< arr.length;i++) {
            System.out.print(arr[i]+"");
        }
    }

    public void quickSort(int[] arr,int start,int end){
        if(start < end){
            // 把首位数字作为基准数
            int stand = arr[start];
            //记录需要排序的下标
            int low = start;
            int high = end;
            // 循环找到比基准数大的数和比基准数小的数
            while(low < high) {
                // 右边的数字比基准数大
                while(low < high && arr[high] >= stand){
                    high--;
                }
                //使用右边的数替换左边的数
                arr[low] = arr[high];
                // 左边的数字比基准数小
                while(low < high && arr[low] <= stand){
                    low++;
                }
                //使用左边的数替换右边的数
                arr[high] = arr[low];
            }
            // 把标准值赋给下标重合的位置
            arr[low] = stard;
            // 处理所有小的数字
            quickSort(arr, start, low);
            // 处理所有大的数字
            quickSort(arr, low + 1, end);
        }
    }
}
```