

牛客网题单

学习网站

<https://www.programmcarl.com/>

```
//第一节
https://ac.nowcoder.com/acm/problem/collection/481
//第二节
https://ac.nowcoder.com/acm/problem/collection/509
//第三节
https://ac.nowcoder.com/acm/problem/collection/518
//第四节
https://ac.nowcoder.com/acm/problem/collection/558
//第五节
https://ac.nowcoder.com/acm/problem/collection/564
//第六节
https://ac.nowcoder.com/acm/problem/collection/576
//第七节
https://ac.nowcoder.com/acm/problem/collection/605
//第八节
https://ac.nowcoder.com/acm/problem/collection/614
//第九节
https://ac.nowcoder.com/acm/problem/collection/621
```

when you become a little better, you can get closer

- 对于1s的时间，能跑多少数据
 - O (logn) : 很大，longlong以内都行
 - O (n) : 10的7次方，也就是1000万的数据
 - O (nlogn) : 5×10^5 ，大约50万的数据
 - O (n^2) : 1000-5000左右
 - O (n^3) : 200-500左右
 - O (2^n) : 20-25
 - O (n!) : 12左右
- 对于256MB的空间，
 - 一个int，32位，4个字节。256= 2^{28} = 67,108,864个in
 - 也就是 6×10^7 的数据，如果是long long，那么少一半就可以了。

数字和字符串反转

数字进行反转 进行反转

```
int n = scanner.nextInt();

int sum = 0;

while(n > 0) {

    sum = sum * 10 + n % 10;

    System.out.println(sum);

    n /= 10;

}
```

字符串

String

将字符串转成字符数组

```
char []chars1 = scanner.next().toCharArray();
```

将字符数组转换成字符串

```
String s1 = new String(chars1); // 通过构造函数
```

对字符串进行排序

```
// 先转化成字符数组
char []chars1 = scanner.next().toCharArray();
// 对数组进行排序
Arrays.sort(chars1);
// 通过构造函数 将字符数组重新转换成 字符串
String s1 = new String(chars1);
```

获取第一个元素

```
String.charAt(int n) // 获取 第 n 个元素
```

字符串切割

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

String seq = scanner.nextLine();

// 分割函数 通过空格将字符串进行分割
String []words = seq.split(" ");

for(int i = 0 ; i < words.length; i++) {
    System.out.println(words[i]);
}
```

切割可能需要 字符可能需要进行转义

```
String idString = "192.168.1.1";

// 注意点是 . 在正则表达式中 有语义
// 需要进行转义
String []ip = idString.split("\\.");

for(int i = 0 ; i < ip.length; i++) {
    System.out.println(ip[i]);
}
```

StringBuffer

String.valueOf(n) 字符转成字符串

```
/**
 * String.valueOf(ch)
 * 将数字、字符、Boolean转成字符串 返回类型 String
 */

/**
 * Integer.valueOf(ch)
 * 将字符串转成Integer 返回类型 Integer
 * 将字符转成对应的ASCII值
 */

/**
 * Integer.parseInt(ch)
 * 只能将字符串转成Integer 返回类型 Integer
 */

/**
 * Integer.toString(ch)
 * 将字符转成字符串 返回类型 String
 */
```

stringBuffer.reverse(); 字符串反转函数

```
/**
 * stringBuffer.reverse();
 * 将字符串进行反转
 * 由于String 类中没有这个功能 于是 借用StringBuffer类来完成
 */
```

案例 将数字进行翻转

- 将数字转成字符串
- 利用StringBuffer进行反转

```
package 蓝桥杯习题;

import java.util.Scanner;

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

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        StringBuffer stringBuffer = new StringBuffer(String.valueOf(n));

        stringBuffer.reverse();

        System.out.println(stringBuffer);
    }
}
```

```
}  
}
```

Integer

Integer.toString(int,int)

```
/**  
 * Integer.toString(n,hex);  
 * n: 表示即将转换的数字  
 * hex: 表示要转化的进制数  
 * 将n 转换成hex 进制数 , 但是 对于负数而言转化不成功  
 */
```

```
String string = Integer.toString(n,2);  
String string = Integer.toString(n,8);  
String string = Integer.toString(n,10);  
String string = Integer.toString(n,16);
```

Integer.bitCount(int)

```
/**  
 * Integer.bitCount(n)  
 *  
 * n: 目标数  
 * 统计 n 转换成二进制 数 当中 含有1 的个数  
 */  
int count = Integer.bitCount(n);
```

Integer.toString(int)

```
String str = Integer.toString(int n) // 将 n 转成字符串
```

将数字转变成字符

```
// 1 将数字先转成字符串  
String str = Integer.toString(int n) // 将 n 转成字符串  
// 2 将 字符串转成 字符  
str.charAt(0);
```

将字符转成数字

```
char ch = '0';  
// 将字符转成字符串  
String first = String.valueOf(ch);  
// 将字符串转成数字  
int end = Integer.parseInt(first);
```

Integer.parseInt(String,Int)

```
//含义 是 将 string 字符串当成 n 进制 准换成 10进制  
int num = Integer.parseInt(String string,int n);
```

BigInteger

构造函数

```
BigInteger bigInteger = new BigInteger(String);    // 将 String 转成 BigInteger  
  
BigInteger bigInteger = new BigInteger("1");    // 将 字符串1 转成 大数类型 1
```

乘法API

```
bigInteger = bigInteger.multiply(BigInteger); // 将 bigInter 和 BigInteger 相乘  
  
// 先使用 new BigInteger(String) 将 String 转成 BigInteger  
bigInteger = bigInteger.multiply(new BigInteger(String));  
  
// 实例 将 bigInteger 和 2 相乘  
bigInteger = bigInteger.multiply(new BigInteger("2"));
```

加法API

```
bigInteger = bigInteger.add(BigInteger); // 将 bigInter 和 BigInteger 相加  
  
// 先使用 new BigInteger(String) 将 String 转成 BigInteger  
bigInteger = bigInteger.add(new BigInteger(String));  
  
// 实例 将 bigInteger 和 2 相加  
bigInteger = bigInteger.add(new BigInteger("2"));
```

快速读入和输出

能使用 快读和快速输出 尽量使用快速读入和输出 避免 卡时间

读入

```
static BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

```
// 再读入数据处理的时候 使用readLine() 函数
// 读取 一行字符串数据
String str = in.readLine();
// 当读入一行数据是 分别放入到数组当中去 使用split数组 进行分割
String[] strArray = str.split(" ");
```

输出

```
static PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
```

```
// 输出 使用 out.print语句输出 输出到缓冲区中
// 使用 out.flush 将缓冲区中的数据 显示
out.print(BigInteger);
out.flush();
```

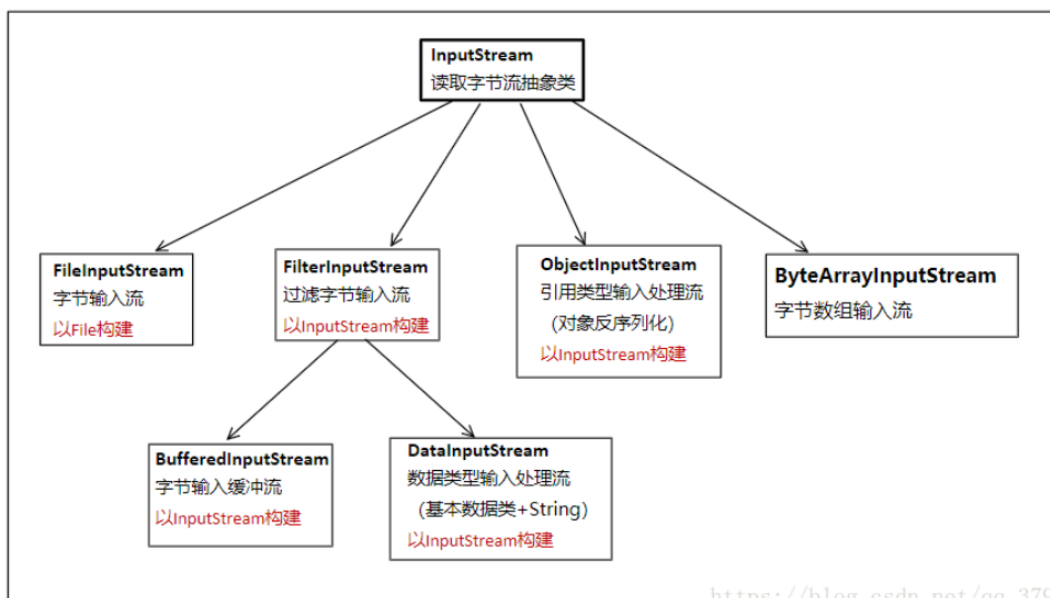
流/文件

InputStream

2018年4月9日, 星期一

22:07

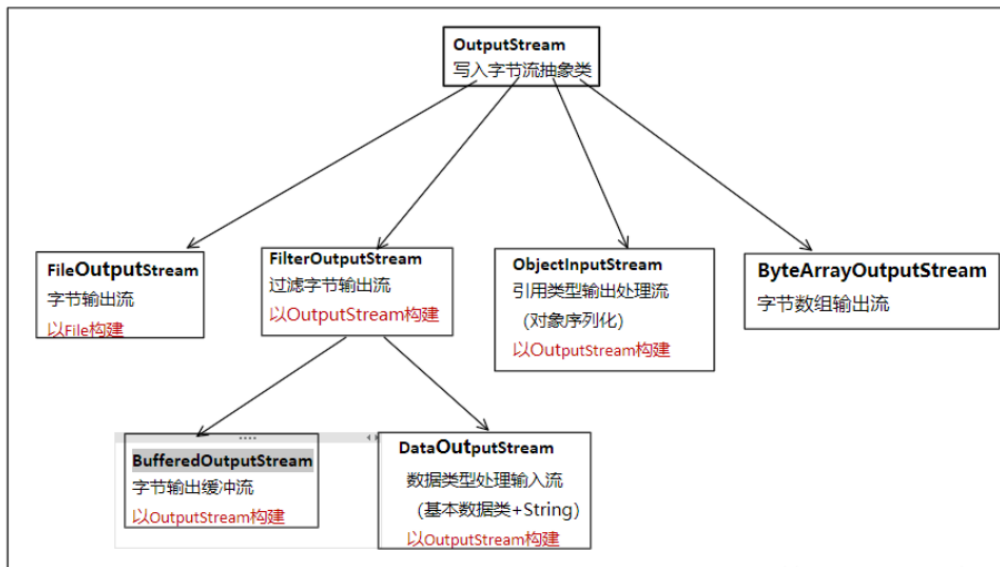
By sirm23333



https://blog.csdn.net/qq_37969431

OutputStream

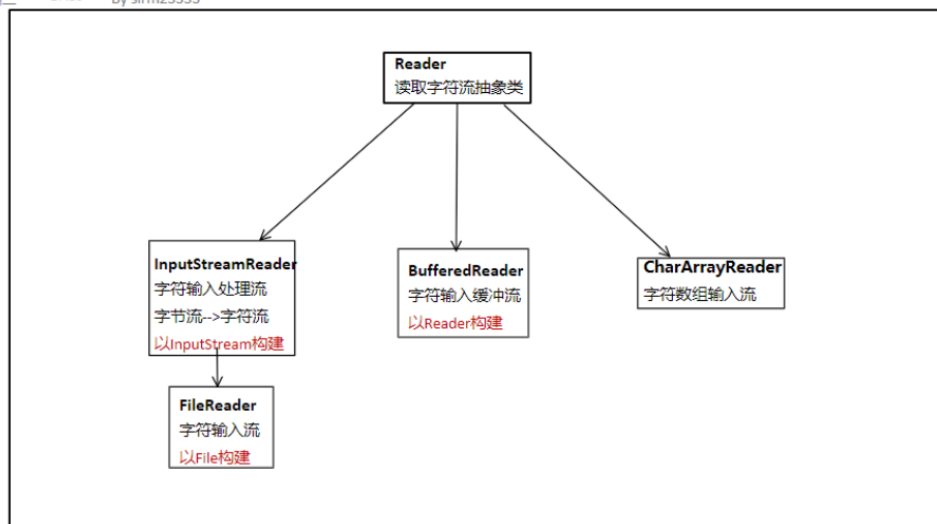
2018年4月10日, 星期二 17:58 By sirm23333



https://blog.csdn.net/qq_37969433

Reader

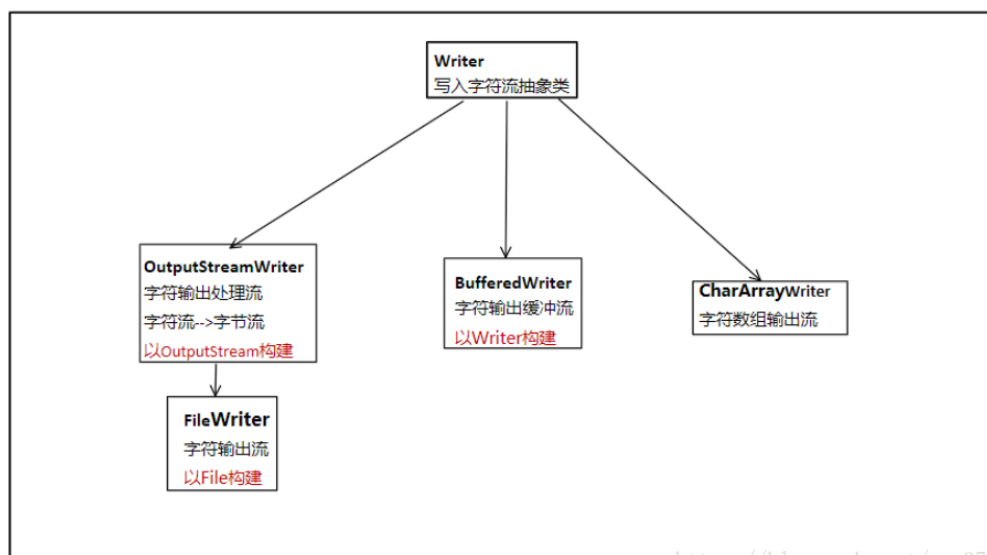
2018年4月10日, 星期二 17:59 By sirm23333



https://blog.csdn.net/qq_37969433

Writer

2018年4月10日, 星期二 19:06 By Sirm23333



https://blog.csdn.net/qq_37969433

文件操作 API

创建一个文件对象

```
String filename = "E:\\\\Eclipse\\\\demo1\\\\src\\\\第二章\\\\prime.txt";

File file = new File(filename);
```

获取文件属性

1 获取文件名称 file.getName();

```
String name = file.getName();

System.out.println(name);
```

2 获取文件目录file.getParent();

```
String pathString = file.getParent();

System.out.println(pathString);
```

3 获取文件对应路径

```
String path = file.getPath();

System.out.println(path);
```

4 判断当前是不是一个目录

```
String fileString = "E:\\Eclipse\\demo1\\src\\第二章";

File file2 = new File(fileString);

Boolean isDirectory = file2.isDirectory()
```

5 列举出当前目录中所有文件名称

```
if(file2.isDirectory()) {

    // 返回当前目录所有的文件名称
    String []nameStrings = file2.list();

    for (String string : nameStrings) {
        System.out.println(string);
    }
}
```

6 删除文件

```
// 删除文件操作
String deleteString = "E:\\Eclipse\\demo1\\src\\第二章\\delete.txt";

File deleFile = new File(deleteString);

// 判断文件时候存在
if(deleFile.exists()) {
    if(deleFile.delete()) {
        System.out.println("删除成功");
    }
}
}
```

7 创建文件

```
//创建一个文件
File createFile = new File("E:\\Eclipse\\demo1\\src\\第二章\\newFile.txt");

// 判断文件是不是存在 如果不存在就创建
if(createFile.exists() == false) {
    createFile.createNewFile();
    if(createFile.exists() == true) System.out.println("创建成功");
}
}
```

文件输入

1 字符流输入

```
// 创建一个文件对象
FileReader fileReader = new FileReader("E:\\Eclipse\\demo1\\src\\第二章\\show.txt");

int ch = 0;

String string = "";

// file.read() 以字符为单位读取 return int
while((ch = fileReader.read()) != -1) string += (char)ch;

System.out.println(string);
```

2 字节流输入

```
// 创建一个文件对象
FileInputStream in = new FileInputStream("E:\\Eclipse\\demo1\\src\\第二章\\newFile.txt");

// 按照一个字节形式 读入
int n = 0;

while(true) {
```

```

n = in.read();

if(n == -1) {
    break;
}
// 打印显示的是以Asca11值显示
System.out.print(n + " ");
}

in.close();

```

文件输出

```

// 创建一个文件对象
FileReader fileReader = new FileReader("E:\\Eclipse\\demo1\\src\\第二章\\show.txt");

int ch = 0;

String string = "";

// file.read() 以字符为单位读取 return int
while((ch = fileReader.read()) != -1) string += (char)ch;

System.out.println(string);

// 创建一个即将写入内容的文件
File wirteFile = new File("E:\\Eclipse\\demo1\\src\\第二章\\show1.txt");

// 如果文件不存在就创建一个新的文件
if(wirteFile.exists() == false) wirteFile.createNewFile();

// 创建一个写入流
FileWriter filewriter = new FileWriter(wirteFile);

// 将要写入的文件写入文件中
filewriter.write(string);

filewriter.write("\r\n");

// 关闭写入流
filewriter.close();

```

实例

```

package 第二章;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;

```

```
import java.io.Reader;

public class Ty22Io流 {

    public static void main(String[] args) throws Exception {

        // 使用字符串记录文件名称
        String filename = "E:\\Eclipse\\demo1\\src\\第二章\\prime.txt";

        // 实例化一个文件对象
        File file = new File(filename);

        // 实例化reader对象
        Reader reader = new FileReader(file);

        // 实例化 BufferedReader 对象 读写数据
        BufferedReader bufferedReader = new BufferedReader(reader);

        // 使用readLine() 读取数据
        String string = bufferedReader.readLine();

        while(string != null) {
            System.out.println(string);
            string = bufferedReader.readLine();
        }
    }
}
```

习题列表

01 查找两个总和为特定值的索引 - 蓝桥云课 (lanqiao.cn)

暴力求解

时间复杂度 $O(n^2)$

```
package 搜索;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class Ty09迷宫 {

    /*
    0 1 0 0 0 0
    0 0 0 1 0 0
    0 0 1 0 0 1
    1 1 0 0 0 0
        DRRURRDDDR
    */

    public static int step = 0;

    public static int map[][] = new int [30][50];

    public static int n = 4;
    public static int m = 6;

    static boolean visit[][] = new boolean [30][50];

    // 上下最有有一个优先级
    // 具体根据题目意思
    static int dx[] = {1, 0, 0, -1};
    static int dy[] = {0, -1, 1, 0};

    public static void BFS(int row,int col) {

        for(int i = 0 ; i < row; i++) {
            for(int j = 0 ; j < col; j++) {
                visit[i][j] = false;
            }
        }

        Queue<int []> queue = new LinkedList<>();
        // 具体的路径规则
        Queue<String> queue2 = new LinkedList<String>();

        queue.offer(new int [] {0,0});

        queue2.offer("");
    }
}
```

```

visit[0][0] = true;

while(!queue.isEmpty()) {

    int []place = queue.element();

    int x = place[0];
    int y = place[1];

    // 获取根节点
    String path = queue2.element();

//    System.out.println(path);
    // 对四个方向进行搜索
    for(int i = 0; i < 4; i++) {

        int newx = x + dx[i];
        int newy = y + dy[i];

        if(newx >= 0 && newx < row &&
            newy >= 0 && newy < col &&
            visit[newx][newy] == false &&
            map[newx][newy] == 0) {
            // 记录最终答案
            String tempString = "";
            queue.offer(new int[] {newx,newy});
            visit[newx][newy] = true;
            // path 不能被修改
            // 向四周搜索一次的时候path始终相同
            if(dx[i] == 0 && dy[i] == -1)
                tempString = path + "L";

            if(dx[i] == 0 && dy[i] == 1)
                tempString = path + "R";

            if(dx[i] == -1 && dy[i] == 0)
                tempString = path + "U";

            if(dx[i] == 1 && dy[i] == 0)
                tempString = path + "D";

            queue2.offer(tempString);

            if(newx == row - 1 && newy == col - 1) {
                System.out.println(tempString);
            }
        }
    }

    queue.poll();

    queue2.poll();
}
}

```

```

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    String []strings = new String[30];

    for(int i = 0 ; i < n; i++)
        strings[i] = scanner.nextLine();

    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            map[i][j] = strings[i].charAt(j) - '0';

    BFS(n,m);

}
}

```

哈希表

哈希表的好处是通过建立哈希表可以有效的消除了一层循环 将时间复杂度大大降低

```

HashMap<Integer, Integer> hashMap = new HashMap<>(); //哈希Map创建

```

```

if(hashMap.containsKey(ans-arr[i])) { // 判断是否存在这个键值对

    // 这里的键值顺序和数组的键值对顺序正好是反着的
    System.out.println(i + " " + hashMap.get(ans-arr[i]));

    return;

}

```

```

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
public class 查找两个总和为特定值的索引 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int []arr = new int [101];

        int n = scanner.nextInt();

        HashMap<Integer, Integer> hashMap = new HashMap<>();

        for(int i = 0 ; i < n; i++) {

            arr[i] = scanner.nextInt();

            hashMap.put(arr[i],i);

        }

        int ans = scanner.nextInt();
    }
}

```

```

        for(int i = 0 ; i < n; i++) {

            if(hashMap.containsKey(ans-arr[i])) {

                System.out.println(i + " " + hashMap.get(ans-arr[i]));

                return;
            }
        }
    }
}

```

02 寻找 3 个数的最大乘积 - 蓝桥云课 (lanqiao.cn)

暴力求解

错误示范：先对数组进行排序

选择最大的三个数据进行相乘，但是没有考虑到 负数 两个负数相乘之后就是一个正数

正确矫正

- 要么是最大三个正数相乘
- 最后两个负数相乘再乘上最大的正数

```

Arrays.sort(arr,0,n);

int ans = arr[n-1] * arr[n-2] * arr[n-3];

ans = Math.max(ans,arr[n-1] * arr[0] * arr[1]);

```

```

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int []arr = new int [n];

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

Arrays.sort(arr,0,n);

int ans = arr[n-1] * arr[n-2] * arr[n-3];

System.out.println(ans);

```

正确解法（暴力求解）

使用3个for循环进行遍历 复杂度 $O(n^3)$

```

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

```



```

int []arr = new int [n];

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

int ans = (int) -1e9;

for(int i = 0; i < n; i++) {
    for(int j = i + 1; j < n; j++) {
        for(int k = j + 1; k < n; k++) {
            ans = Math.max(ans, arr[i] * arr[j] * arr[k]);
        }
    }
}

System.out.println(ans);

```

```

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    int n = scanner.nextInt();

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

    for(int i = 0 ; i < n; i++) {
        int temp = scanner.nextInt();
        list.add(temp);
    }

    Collections.sort(list);

    int ans = list.get(n - 1) * list.get(n - 2) * list.get(n - 3);

    ans = Math.max(ans, list.get(n - 1) * list.get(0) * list.get(1));

    System.out.println(ans);

}

```

使用列表取存储 使用Collections类方法去调用 执行排序

03 字符统计 - 蓝桥云课 (lanqiao.cn)

暴力+快排

```

string.toCharArray();

```

```

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    String string = scanner.next();

```

多

```
char [] chars = string.toCharArray(); // 字符串转成字符数组排序处理起来方便很

Arrays.sort(chars);

int max = 0;

int cnt = 1;

for(int i = 0 ; i < string.length() - 1;i++) {

    if(chars[i] == chars[i+1]) {
        cnt++;
    }

    else {
        if(cnt >= max) {
            max = cnt;
        }

        cnt = 1;
    }
}

cnt = 1;

for(int i = 0 ; i < string.length() - 1;i++) {

    if(chars[i] == chars[i+1]) {
        cnt++;
    }

    else {

        if(cnt >= max) {
            System.out.print(chars[i]);
        }

        cnt = 1;
    }
}
}
```

数组存储

因为只有26个字母所以可以巧妙使用堆数组进行存储

```
package 蓝桥杯习题;

import java.awt.Taskbar.State;
import java.io.CharArrayReader;
import java.io.InterruptedIOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
```

```

import java.util.Scanner;

import javax.swing.UIClientPropertyKey;

public class 查找两个总和为特定值的索引 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String string = scanner.next();

        char []chars = string.toCharArray();

        int arr[] = new int [30];

        int max = 0;

        for(int i = 0 ; i < string.length(); i++) {
            arr[chars[i] - 'A']++;

            max = Math.max(max, arr[chars[i] - 'A']);
        }

        for(int i = 0 ; i < arr.length;i++) {
            if(arr[i] == max) {

                system.out.print((char)(i + 'A'));

            }
        }

    }
}

```

使用HashMap求解

```

hashMap.put(chars[i], hashMap.getOrDefault(chars[i], 0) + 1);

//先判断是否存在
if(hashMap.containsKey(chars[i]) == false) {

    hashMap.put(chars[i], 1);
}
else {

    // 如果不存在会报一个异常
    int count = hashMap.get(chars[i]);
    hashMap.put(chars[i], count+1);
}

```

```

package 蓝桥杯习题;

import java.awt.Taskbar.Feature;
import java.awt.Taskbar.State;
import java.io.CharArrayReader;

```

```

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;

import javax.security.auth.x500.X500Principal;
import javax.swing.UIClientPropertyKey;
import javax.swing.plaf.basic.BasicInternalFrameTitlePane.IconifyAction;

public class 查找两个总和为特定值的索引 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String string = scanner.next();

        char []chars = string.toCharArray();

        HashMap<Character, Integer>hashMap = new HashMap<>();

        for(int i = 0 ; i < string.length(); i++) {
            //
            hashMap.put(chars[i], hashMap.getOrDefault(chars[i], 0) +
1);

            //先判断是否存在
            if(hashMap.containsKey(chars[i]) == false) {
                hashMap.put(chars[i], 1);
            }
            else {
                // 如果不存在会报一个异常
                int count = hashMap.get(chars[i]);
                hashMap.put(chars[i], count+1);
            }
        }

        int cnt = 1;
        int max = 1;

        List<Character> list = new ArrayList<>();

        for (Character key : hashMap.keySet()) {

            cnt = hashMap.get(key);

            if(cnt > max) {
                list.clear();
                list.add(key);
                max = cnt;
            }
            else if(cnt == max) {
                list.add(key);
            }
        }
    }
}

```

```

    }

    for (Character character : list) {
        System.out.print(character);
    }
}
}

```

04 用杂志拼接信件 - 蓝桥云课 (lanqiao.cn)

```

import java.util.Scanner;

public class 字符串 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String str1 = scanner.next();
        String str2 = scanner.next();

        char []chars1 = str1.toCharArray();
        char []chars2 = str2.toCharArray();

        int []arr1 = new int [26];
        int []arr2 = new int [26];

        for(int i = 0 ; i < str1.length(); i++) {
            arr1[chars1[i] - 'a']++;
        }

        for(int i = 0 ; i < str2.length(); i++) {
            arr2[chars2[i] - 'a']++;
        }

        for(int i = 0 ; i < 26; i++) {
            if(arr1[i] < arr2[i]) {
                System.out.println("NO");
                return;
            }
        }

        System.out.println("YES");
    }
}

```

```

package 蓝桥杯习题;

import java.util.HashMap;
import java.util.Iterator;

```

```

import java.util.Scanner;

public class 字符串 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String str1 = scanner.next();
        String str2 = scanner.next();

        HashMap<Character, Integer> hashMap1 = new HashMap<>();
        HashMap<Character, Integer> hashMap2 = new HashMap<>();

        for(int i = 0 ; i < str1.length(); i++) {

            hashMap1.put(str1.charAt(i), hashMap1.getDefault(str1.charAt(i),
1) + 1);
        }

        for(int i = 0 ; i < str2.length(); i++) {

            hashMap2.put(str2.charAt(i), hashMap2.getDefault(str2.charAt(i),
1) + 1);
        }

        for (char ch : hashMap2.keySet()) {

            if(!hashMap1.containsKey(ch)) {
                System.out.println("NO");
                return;
            }

            else {
                if(hashMap1.get(ch) < hashMap2.get(ch)) {
                    System.out.println("NO");
                    return;
                }
            }
        }
        System.out.println("YES");
    }

}

```

05 小蓝吃糖果 - 蓝桥云课 (lanqiao.cn)

关键点是 不能重复吃糖果，最有可能重复吃糖果的是 最多的，如果最多的都不会重复吃 哪剩下的就基本就不会吃糖果

```

package 蓝桥杯习题;

import java.util.Scanner;

import javax.sql.rowset.spi.SyncFactory;

```

```

public class 小蓝吃糖果 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();
        int []arr = new int [n];

        int max = 0;

        long sum = 0;

        for(int i = 0 ; i < n; i++) {

            arr[i] = scanner.nextInt();

            sum += arr[i];

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

        if(sum - max <= max - 2) {
            System.out.println("No");
        }
        else {
            System.out.println("Yes");
        }
    }

}

```

暴力求解

直接排序 每次用自大的去剔除 时间复杂度达到了 $O(n^2 \log n)$ （只能过两个测试点）

```

package 蓝桥杯习题;

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

public class 查找两个总和为特定值的索引 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        int []arr = new int [n];

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

```

```

Arrays.sort(arr);

for(int i = n - 1; i > 0; i--) {
    arr[i - 1] = arr[i] - arr[i - 1];
    Arrays.sort(arr,0,i); //表示 从 0 到 i 排序
}

if(arr[0] > 1) {
    System.out.println("No");
}
else {
    System.out.println("Yes");
}
}
}

```

06 含 2 天数 - 蓝桥云课 (lanqiao.cn)

```

public class Main {

    public static Boolean isContain(int year) {

        if(year % 10 == 2) {

            return true;
        }

        else if(year / 10 % 10 == 2) {
            return true;
        }

        else if(year / 100 % 10 == 2) {
            return true;
        }

        else if(year / 1000 == 2) {
            return true;
        }

        else return false;
    }

    public static Boolean days(int year) {

        if((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
            return true;
        }
        else {
            return false;
        }
    }

    public static void main(String[] args) {

```



```

int sum = 0;

for(int i = 1900; i <= 9999; i++) {

    if(isContain(i) == true) {

        if(days(i) == true) {
            sum += 366;
        }
        else {
            sum += 365;
        }
    }

    else {
        if(days(i) == true) {
            sum += 12 * 10 + 29 + 31;
        }
        else {
            sum += 12 * 10 + 28 + 31;
        }
    }
}

System.out.println(sum);
}
}

```

07 完全日期 - 蓝桥云课 (lanqiao.cn)

```

public class Main {

    public static Boolean IsLear(int year) {

        if((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
            return true;
        }
        else {
            return false;
        }
    }

    public static Boolean isSqrt(int year,int month,int day){

        int sum = 0;

        while(day > 0) {
            sum += (day%10);
            day /= 10;
        }

        while(month > 0) {
            sum += (month%10);
            month /= 10;
        }

        while(year > 0) {
            sum += (year%10);

```

```

        year /= 10;
    }

    int ans = (int) Math.sqrt(sum);

    if(ans * ans == sum) {
        return true;
    }

    return false;
}

public static void main(String[] args) {

    int sum = 0;

    int days[] = {0,31,29,31,30,31,30,31,31,30,31,30,31};

    int d = 1;
    int m = 1;
    int year = 2001;

    while(year != 2022 || m != 1 || d != 1) {

        if(IsLear(year)) {
            days[2] = 29;
        }

        else {
            days[2] = 28;
        }

        if(isSqrt(year, m, d) == true) {
            sum++;
        }

        d++;

        if(d > days[m]) {
            m++;
            d = 1;
        }

        if(m > 12) {
            m = 1;
            year++;
        }
    }

    System.out.println(sum);
}
}

```

```

public class Main {

    public static Boolean IsLear(int year) {

        if((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
            return true;
        }
        else {
            return false;
        }
    }

    public static Boolean issqrt(int year,int month,int day){

        int sum = 0;

        while(day > 0) {
            sum += (day%10);
            day /= 10;
        }

        while(month > 0) {
            sum += (month%10);
            month /= 10;
        }

        while(year > 0) {
            sum += (year%10);
            year /= 10;
        }

        int ans = (int) Math.sqrt(sum);

        if(ans * ans == sum) {
            return true;
        }

        return false;
    }

    public static void main(String[] args) {

        int sum = 0;

        int days[] = {0,31,29,31,30,31,30,31,31,30,31,30,31};

        int d = 1;
        int m = 10;
        int year = 1949;
        int week = 6;

        while(year != 2012 || m != 10 || d != 2) {

            if(IsLear(year)) {
                days[2] = 29;
            }

            else {

```

```

        days[2] = 28;
    }

    if(week == 7 && m == 10 && d == 1) {
        sum++;
    }

    week++;

    if(week > 7) week = 1;

    d++;

    if(d > days[m]) {
        m++;
        d = 1;
    }

    if(m > 12) {
        m = 1;
        year++;
    }
}

System.out.println(sum);
}
}

```

09 二进制中 1 的个数『更新』 - 蓝桥云课 (lanqiao.cn)

进制转换常规转换

```

/**
 *
 * @param n
 * @author Typecoh
 * @name positive
 * @return
 * 将一个正数转换成二进制 这里采用的是分割法
 */

public static int positive(int n) {

    int count = 1;

    while(true) {

        if(Math.pow(2, count) <= n) {
            count++;
        }

        else {
            break;
        }
    }
}

```

```

    }

    int ans = count - 1;

    int res = 0;

    while(n>0) {

        if(n - Math.pow(2, ans) >= 0) {
            res++;
            n = (int) (n - Math.pow(2, ans));

        }

        ans--;
    }

    return res;

}

```

```

/**
 *
 * @param n
 * @author Typecoh
 * @name Minus
 * @return
 * 将一个负数转换成二进制 这里采用的是分割法
 * 用数组模拟存储1 和 0
 */
public static int Minus(int n) {

    n = Math.abs(n);

    int []arr = new int [32];

    int count = 1;

    while(true) {

        if(Math.pow(2, count) <= n) {
            count++;
        }

        else {
            break;
        }
    }

    int ans = count - 1;

    int res = 0;

    while(n>0) {

        if(n - Math.pow(2, ans) >= 0) {
            res++;
            arr[ans] = 1;

```

```

        n = (int) (n - Math.pow(2, ans));

    }
    ans--;
}

for(int i = 0 ; i < arr.length - 1; i++) {

    if(arr[i] == 0) arr[i] = 1;
    else arr[i] = 0;

}

arr[arr.length - 1] = 1;

int cnt = 1;

for(int i = 0 ; i < arr.length; i++) {

    if(arr[i] + cnt > 1) {
        arr[i] = 0;
        cnt = 1;
    }
    else {
        arr[i] = 1;
        break;
    }

}

int result = 0;

for (int i : arr) {
    if(i == 1) {
        result++;
    }
}

return result;
}

```

完整代码

```

import java.util.Scanner;

public class Main {

    public static int positive(int n) {

        int count = 1;

        while(true) {

            if(Math.pow(2, count) <= n) {
                count++;
            }

}

```

```

        else {
            break;
        }
    }

    int ans = count - 1;

    int res = 0;

    while(n>0) {

        if(n - Math.pow(2, ans) >= 0) {
            res++;
            n = (int) (n - Math.pow(2, ans));
        }
        ans--;
    }

    return res;
}

public static int Minus(int n) {

    n = Math.abs(n);

    int []arr = new int [32];

    int count = 1;

    while(true) {

        if(Math.pow(2, count) <= n) {
            count++;
        }

        else {
            break;
        }
    }

    int ans = count - 1;

    int res = 0;

    while(n>0) {

        if(n - Math.pow(2, ans) >= 0) {
            res++;
            arr[ans] = 1;
            n = (int) (n - Math.pow(2, ans));
        }
        ans--;
    }

    for(int i = 0 ; i < arr.length - 1; i++) {

```

```

        if(arr[i] == 0) arr[i] = 1;
        else arr[i] = 0;
    }

    arr[arr.length - 1] = 1;

    int cnt = 1;

    for(int i = 0 ; i < arr.length; i++) {

        if(arr[i] + cnt > 1) {
            arr[i] = 0;
            cnt = 1;
        }
        else {
            arr[i] = 1;
            break;
        }
    }

    int result = 0;

    for (int i : arr) {
        if(i == 1) {
            result++;
        }
    }

    return result;
}

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    int n = scanner.nextInt();

    int res = 0;

    if(n > 0) {

        res = positive(n);
    }

    else {

        res = Minus(n);
    }

    System.out.println(res);
}
}

```


使用运算符

```
package 蓝桥杯习题;

import java.util.Scanner;

public class 巧用二进制 {

    /**
     * Title: 巧用二进制.java
     * Description: 将数字转换成二进制数 求解 二进制数当中 1 的个数
     *              数字在计算机中存储是按照反码进行存储的
     * @author Typecoh
     * @date 2022年12月6日
     * @version 1.0
     */

    public static int Chane(int n){

        int count = 0;

        int ans = 0;

        while(true) {

            int temp = n & 1;
            n >>= 1;

            if(temp == 1) ans += 1;

            count++;

            if(count >= 32) {
                break;
            }

        }

        return ans;
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        int res = Chane(n);

        System.out.println(res);
    }
}
```

```
}  
  
}
```

使用API

```
/**  
    bitCount(int n) 方法 统计 n 转换成二进制的时候 1 的个数  
*/  
int ans = Integer.bitCount(n);
```

10 乌托邦树 - 蓝桥云课 (lanqiao.cn)

典型的大数相加题型

数字太大 常规的类型存储不下 使用数据进行存储 每个数组元素 对应数字的一位数

利用数组模拟手写大数

```
package 蓝桥杯习题;  
  
import java.util.Scanner;  
  
import javax.swing.text.AbstractDocument.LeafElement;  
  
public class 乌托邦树 {  
    public static void main(String[] args) {  
  
        Scanner scanner = new Scanner(System.in);  
  
        int n = scanner.nextInt();  
  
        int []arr = new int [200];  
  
        arr[1] = 1;  
  
        for(int i = 1 ; i <= n; i++) {  
  
            // 当前次 加 1  
            if(i % 2 == 0) {  
  
                int cnt = 1;  
  
                arr[cnt] += 1;  
  
                while(true) {  
  
                    if(arr[cnt] >= 10) {  
                        arr[cnt] -= 10;  
                        cnt += 1;  
                        arr[cnt] += 1;  
                    }  
                }  
            }  
        }  
    }  
}
```

```

        if(arr[cnt] < 10) break;
    }

}

// 当前此 * 2
else {

    int cnt = 1;

    while(true) {

        if(cnt == arr.length - 1) break;

        arr[cnt] *= 2;

        cnt +=1 ;
    }

    cnt = 1;

    while(true) {

        if(cnt == arr.length - 1) break;

        if(arr[cnt] >= 10) {

            arr[cnt] -= 10;
            cnt++;
            arr[cnt] +=1;
        }
        else {
            cnt++;
        }
    }
}

}

Boolean isStar = false;

for(int i = arr.length - 1 ; i > 0; i--) {

    if(arr[i] != 0 && isStar == false) {
        isStar = true;
        System.out.print(arr[i]);
    }

    else if(isStar == true) {
        System.out.print(arr[i]);
    }
}

System.out.println();
}
}

```

利用BigInteger类

```
BigInteger bigInteger = new BigInteger("1"); // 将字符串转换成 BigInteger
bigInteger = bigInteger.multiply(new BigInteger("2")); // 将 大数乘以 2，首先将2 转
成 大数类型
bigInteger = bigInteger.add(new BigInteger("1")); // 将 大数加 1，首先将1 转成 大数
类型
```

```
package 蓝桥杯习题;

import java.math.BigInteger;
import java.util.Scanner;

public class 利用字符串进行计算大数 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        BigInteger bigInteger = new BigInteger("1");

        for(int i = 1 ; i <= n; i++) {

            if(i % 2 != 0) {

                bigInteger = bigInteger.multiply(new BigInteger("2"));

            }

            else {

                bigInteger = bigInteger.add(new BigInteger("1"));

            }

        }

        System.out.println(bigInteger);

    }

}
```

[11 最大化交易利润 - 蓝桥云课 \(lanqiao.cn\)](https://www.lanqiao.cn/)

暴力筛选

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
```

```

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int []arr = new int [n];

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

int max = (int)-1e5;

for(int i = 0 ;i < arr.length; i++) {
    for(int j = i+1 ; j < arr.length ; j++) {
        max = Math.max(max, arr[j] - arr[i]);
    }
}

System.out.println(max);
}
}

```

一点点贪心

为了找到最大利润

维护 $[1, x - 1]$ 的最小值 和 $x - 1$ 的最大利润

维护这个最大的利润 出现一个最大的利润 按照原方法查找 最大利润 进行更新维护

```

package 蓝桥杯习题;

import java.util.Scanner;

public class 最大化交易利润 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        int []arr = new int [n];

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

        int min = (int)1e5;

        int max = (int)-1e5;

        for(int i = 0 ; i < arr.length - 1; i++) {

            min = Math.min(min, arr[i]); // 找到 i 之前的最小值

```

```

        max = Math.max(max, arr[i + 1] - min);
    }

    System.out.println(max);
}
}

```

12 确定一个数字是否为 2 的幂 - 蓝桥云课 (lanqiao.cn)

```

package 蓝桥杯习题;

import java.awt.Checkbox;
import java.util.Scanner;

public class 判断时候是2的正数幂 {

    public static Boolean Check(int n) {

        int ans = Integer.bitCount(n);

        // System.out.println(ans);

        if(n == 1) return false;

        if(ans == 1) return true;

        return false;
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        if(Check(n) == true) {
            System.out.println("YES");
        }

        else {
            System.out.println("NO");
        }

    }
}

```

13 确定字符串是否是另一个的排列 - 蓝桥云课 (lanqiao.cn)

数组存储

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        String str1 = scanner.next();
        String str2 = scanner.next();

        int []chars1 = new int [100];
        int []chars2 = new int [100];

        for(int i = 0 ; i < str1.length(); i++) {
            chars1[str1.charAt(i) - 'A']++;
        }

        for(int i = 0 ; i < str2.length(); i++) {
            chars2[str2.charAt(i) - 'A']++;
        }

        for(int i = 0 ; i < chars1.length; i++) {

            if(chars1[i] < chars2[i]) {
                System.out.println("NO");
                return;
            }

        }

        System.out.println("YES");
    }
}
```

HashMap存储

```
//hashMap1.getOrDefault(chars1[i], 0) // 如果说 hashMap1 没有 chars1[i] 值 就返回 0
// 如果存在 就 回去 当前 已有的 值 再在原来的基础上 增加一个
hashMap1.put(chars2[i], hashMap1.getOrDefault(chars2[i], 0) + 1);
```

```
import java.util.HashMap;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

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

```

String str1 = scanner.next();
String str2 = scanner.next();

char []chars1 = str1.toCharArray();
char []chars2 = str2.toCharArray();

HashMap<Character, Integer> hashMap1 = new HashMap<>();
HashMap<Character, Integer> hashMap2 = new HashMap<>();

for(int i = 0 ; i < chars1.length; i++) {
    hashMap1.put(chars1[i], hashMap1.getOrDefault(chars1[i], 0) + 1);
}

for(int i = 0 ; i < chars2.length; i++) {
    hashMap2.put(chars2[i], hashMap2.getOrDefault(chars2[i], 0) + 1);
}

for(Character character : hashMap2.keySet()) {
    if(hashMap1.containsKey(character) == false) {
        System.out.println("NO");
        return;
    }
    else {
        if(hashMap2.get(character) > hashMap1.get(character)) {
            System.out.println("NO");
            return;
        }
    }
}

System.out.println("YES");
}
}

```

14 压缩字符串 - 蓝桥云课 (lanqiao.cn)

哈希表

```

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;

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

        Scanner scanner = new Scanner(System.in);

        char []chars1 = scanner.next().toCharArray();

        HashMap<Character, Integer> hashMap = new HashMap<>();

        List<Character> list = new ArrayList<>();
    }
}

```



```

        for(int i = 0 ; i < chars1.length; i++) {

            hashMap.put(chars1[i], hashMap.getOrDefault(chars1[i], 0) + 1);
        }

        for (Character character : hashMap.keySet()) {

            list.add(character);

            if(hashMap.get(character) > 1) {

                list.add(Integer.toString(hashMap.get(character)).charAt(0));
            }
        }

        if(list.size() < chars1.length) {
            for (Character character : list) {
                System.out.print(character);
            }
        }

        else {
            System.out.println("NO");
        }
    }
}

```

双指针

```

package 蓝桥杯习题;

import java.util.Iterator;
import java.util.Scanner;

public class 双指针压缩字符串 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        char []chars = scanner.next().toCharArray();

        Boolean istrue = false;

        // 判断 是否存结果
        // 如果 没有出现相同的 字母 不会出现压缩
        for (int i = 0; i < chars.length - 1; i++) {
            if(chars[i] == chars[i+1]) {
                istrue = true;
            }
        }

        if(istrue == false) {
            System.out.println("NO");
        }
    }
}

```

```

        return;
    }

    int len = 0;

    StringBuffer stringBuffer = new StringBuffer();

    for (int j = 1, i = 0; j < chars.length;) {

        while(true) {

            // 出界判断
            if(j > chars.length - 1) {

                len = j - i;

                stringBuffer.append(chars[i]);

                break;
            }

            // 相等 指针后移
            if(chars[j] == chars[i]) j++;

            // 如果 出现不相等 第一个指针 指向 第二个指针的位置
            // 求长度 第二个指针 后移动 一位
            else {

                len = j - i;

                stringBuffer.append(chars[i]);

                i = j;

                j++;

                break;
            }

        }

        if(len > 1) {
            stringBuffer.append(len);
        }
    }

    System.out.println(stringBuffer);

}
}

```

```

package 蓝桥杯习题;

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

public class 分发饼干 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int boy = scanner.nextInt();
        int num = scanner.nextInt();

        int []arr1 = new int [boy];
        int []arr2 = new int [num];

        for(int i = 0 ; i < arr1.length; i++) {
            arr1[i] = scanner.nextInt();
        }

        for(int i = 0; i < arr2.length; i++) {
            arr2[i] = scanner.nextInt();
        }

        // 常规排序
        Arrays.sort(arr1);
        Arrays.sort(arr2);

        int ans = 0;

        int star = 0;
        int end = arr1.length - 1;

        for(int i = 0 ; i < arr2.length; i++) {

            if(arr2[i] >= arr1[star]) {
                ans++;
                star++;
            }

            if(star > end) {
                break;
            }
        }

        System.out.println(ans);
    }
}

```

16 棋盘放麦子 - 蓝桥云课 (lanqiao.cn)

```

package 蓝桥杯习题;

import java.math.BigInteger;

public class Ty01棋盘放麦子 {

```

```

public static void main(String[] args) {

    String s1 = "1";

    BigInteger bigInteger = new BigInteger(s1);

    BigInteger bigInteger2 = new BigInteger(s1);

    for(int i = 2 ; i <= 64 ; i++) {
        bigInteger = bigInteger.multiply(new BigInteger("2"));
        bigInteger2 = bigInteger2.add(bigInteger);
    }

    System.out.println(bigInteger2);

}
}

```

17 等差数列 - 蓝桥云课 (lanqiao.cn)

```

package 蓝桥杯习题;

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

public class Ty02等差数列 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        int []arr = new int [n];

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

        int min = (int)1e10;

        Arrays.sort(arr);

        // 找到 最小的 那个 差值
        // d = [0,d]

        // 要对分母 做进一步的 判断
        // 当分母 是 0 的 时候要特判

        for(int i = 0 ; i < arr.length - 1; i++) {
            min = Math.min(min, arr[i+1] - arr[i]);
        }

        int num = 0;

        if(min == 0) {

```

```

        if(arr[arr.length - 1] == arr[0]) {
            num = n;
        }
        else {
            num = (arr[arr.length - 1] - arr[0]) + 1 ;
        }
    }

    else num = (arr[arr.length - 1] - arr[0]) / min + 1;

    System.out.println(num);

}
}

```

18 最小质因子之和 - 蓝桥云课 (lanqiao.cn)

// 快速输入输出 使用 `Scanner`类 特别慢会直接超时

```

static BufferedReader bReader = new BufferedReader(new
InputStreamReader(System.in));
static PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

```

```

public static long[] EulerSun(int n) {

    // 欧拉筛选素数
    int isprime[] = new int [n];
    Boolean[] isp = new Boolean [n + 5];

    // 存储当前下表的最小质因子
    int MQF[] = new int [n + 5];
    //记录素数的个数
    int count = 0;
    Arrays.fill(isp, true);
    Arrays.fill(MQF, 0);

    isp[0] = true;
    isp[1] = true;
    // 认为每个数都是素数
    for(int i = 2 ; i<= n; i++) MQF[i] = i;
    for(int i = 2; i <= n ; i++) {

        if(isp[i] == true) isprime[count++] = i;
        for(int j = 0 ; j < count && i * isprime[j] <= n; j++) {

            isp[i * isprime[j]] = false;
            // 更新 最小质因子
            MQF[i * isprime[j]] = isprime[j];
            if(i % isprime[j] == 0) break;
        }
    }

    long []anser = new long[n + 5];
    for(int i = 2 ; i <= n; i++)
        anser[i] = anser[i - 1] + MQF[i];

    return anser;
}

```

```
}
```

19 质因数个数 - 蓝桥云课 (lanqiao.cn)

```
import java.util.Scanner;

public class Main {

    // prime factor decomposition
    public static void PFD(long n) {

        // 先求解 质因数 在 2 - sqrt(n)
        // 用来标记答案的个数
        int ans = 0;
        for(long i = 2; i <= n / i ; i++) {

            if(n % i == 0) {

                ans++;
                while(n % i == 0) n /= i;
            }
        }

        if(n > 1) ans++;

        System.out.println(ans);
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        long n = scanner.nextLong();

        PFD(n);
    }
}
```

20 数数 - 蓝桥云课 (lanqiao.cn)

```
package 第二章;

public class Ty13数数 {

    public static int PFD(int n) {

        int ans = 0;

        for(int i = 2 ; i <= n / i ; i++) {

            if(n % i == 0) {
```

```

        while(n % i == 0) {
            n /=i;
            ans++;
        }
    }

    if(n > 1) ans++;
    return ans;
}

public static void main(String[] args) {

    int anser = 0;

    for(int i = 2333333; i <= 23333333; i++) {

        int res = PFD(i);

        if(res == 12) {
            anser++;
        }
    }
    System.out.println(anser);
}
}

```

21 求解质因子 - 蓝桥云课 (lanqiao.cn)

```

package 第二章;

import java.util.Scanner;

public class Ty14求解质因子 {

    //Solve for prime factors

    /**
     *
     * Title: Ty14求解质因子.java
     * Description: 求解质因数 其实就是 基本算数定理
     * @author Typecoh
     * @date 2022年12月10日
     * @version 1.0
     */
    public static void SPF(long n) {

        for(long i = 2 ; i <= n / i; i++) {

            if(n % i == 0) {

                System.out.print(i + " ");
            }
        }
    }
}

```

```

        while(n % i == 0) n /= i;

    }

}

if(n > 1) System.out.println(n);

}

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    long n = scanner.nextLong();

    SPF(n);

}
}

```

[22 小蓝做实验 - 蓝桥云课 \(lanqiao.cn\)](https://lanqiao.cn)

[23 求和 - 蓝桥云课 \(lanqiao.cn\)](https://lanqiao.cn)

```

package 第二章;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.math.BigInteger;
import java.util.Arrays;
import java.util.Scanner;

public class Ty16求和 {

    static BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
    static PrintWriter out = new PrintWriter(new
OutputStreamWriter(System.out));

    public static BigInteger Cal(int arr[]) {

        BigInteger []pre = new BigInteger[arr.length];

        Arrays.fill(pre, BigInteger.ZERO);
        // 使用前缀和 数组 pre 将 前缀和 进行输出 时间复杂度O(n)
        pre[0] = new BigInteger("0");

        pre[1] = new BigInteger(arr[0] + "");
    }
}

```



```

        for(int i = 2 ; i < pre.length; i++)
            pre[i] = pre[i-1].add(new BigInteger(arr[i - 1] + ""));

        BigInteger ans = BigInteger.ZERO;

        for(int i = 1 ; i < arr.length; i++)
            ans = ans.add(pre[i].multiply(new BigInteger(arr[i] + "")));

        return ans;
    }

    public static void main(String[] args) throws IOException {

        int n = Integer.parseInt(in.readLine());

        int []arr = new int [n];

        String str = in.readLine();

        String[] strArray = str.split(" ");

        for(int i = 0 ; i < arr.length; i++)
            arr[i] = Integer.parseInt(strArray[i]);

        if(n == 1) {

            out.println(arr[0]);

            out.flush();

            return;
        }

        BigInteger bigInteger = Cal(arr);

        out.print(bigInteger);

        out.flush();
    }
}

```

24 星期计算 - 蓝桥云课 (lanqiao.cn)

```

package 第二章;

import java.math.BigInteger;

public class Ty17星期计算 {

    public static void main(String[] args) {

        BigInteger bigInteger = BigInteger.ONE;
    }
}

```

```

        for(int i = 1 ; i <= 22 ; i++) {
            bigInteger = bigInteger.multiply(new BigInteger("20"));
        }

        System.out.println(bigInteger);

        bigInteger = bigInteger.mod(new BigInteger("7"));

        System.out.println(bigInteger);

        System.out.println(7);
    }
}

```

25 求阶乘 - 蓝桥云课 (lanqiao.cn)

```

package 第二章;

import java.awt.Checkbox;
import java.math.BigInteger;
import java.util.Scanner;

import javax.print.attribute.DateTimeSyntax;
import javax.swing.JTabbedPane;

public class Ty19求阶乘 {

    public static BigInteger check(BigInteger n) {

        BigInteger ans = BigInteger.ZERO;

        while(n.compareTo(new BigInteger("0")) == 1) {

            ans = ans.add(n.divide(new BigInteger("5")));

            n = n.divide(new BigInteger("5"));

        }

        return ans;
    }

    public static void find(BigInteger integer) {

        BigInteger l = BigInteger.ONE;
        BigInteger r = new BigInteger("400000000000000020");

        BigInteger mid = l.add(r).divide(new BigInteger("2"));

        Boolean isfind = false;

        while(true) {
            // 如果 mid 的 0的个数小了
            if(check(mid).compareTo(integer) == -1) {

```

```

        l = mid.add(new BigInteger("1"));
    }

    // 找到了
    else if(check(mid).compareTo(integer) == 0
        && mid.mod(new BigInteger("5")).compareTo(BigInteger.ZERO)
== 0 ) {
        isfind = true;
        break;
    }

    // 如果大了
    else
        r = mid.subtract(new BigInteger("1"));

    mid = l.add(r).divide(new BigInteger("2"));
    // 无解
    if(l.compareTo(r) == 1) {
        break;
    }
}

if(isfind == true) {
    System.out.println(mid.toString());
}
else System.out.println("-1");
}

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    String string = scanner.next();

    BigInteger integer = new BigInteger(string);

    find(integer);

}
}

```

```

package 第二章;

import java.util.Scanner;

public class Ty20求阶乘 {

    public static long check(long n) {

        long ans = 0;

        while(n > 0) {
            ans += n / 5;
            n /= 5;
        }
    }
}

```

```

        return ans;
    }

    public static void isfind(long n) {

        long l = 1;
        long r = (long)1e19;

        Boolean isFind = false;

        while(l < r) {

            long mid = l + (r - l) / 2;

            if(check(mid) >= n) r = mid;
            else l = mid + 1;

        }

        if(check(r) == n) System.out.println(r);
        else System.out.println("-1");
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        long K = scanner.nextLong();

        isfind(K);

    }
}

```

```

package 第二章;
import java.util.Scanner;

public class Ty34分巧克力 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int N = scanner.nextInt();
        int K = scanner.nextInt();

        int [][]arr = new int [N][2];

        int max = -1;

        for(int i = 0 ; i < N; i++) {
            for(int j = 0 ; j < 2 ; j++) {
                arr[i][j] = scanner.nextInt();
                max = Math.max(max, arr[i][j]);
            }
        }
    }
}

```

```

    }

    int l = 1;
    int r = max;

    int mid = (l + r) / 2;
    int ans = 1;

    while(l <= r) {

        int cnt = 0;

        for(int i = 0 ; i < N; i++) {
            // 如果长宽都满足
            if(arr[i][0] >= mid && arr[i][1] >= mid) {
                cnt += ((arr[i][0] / mid) * (arr[i][1] / mid));
            }
        }

        if(cnt > K) {
            ans = Math.max(ans, mid);
            l = mid + 1;
            mid = (l + r) / 2;
        }
        else if(cnt == K) {
            ans = Math.max(ans, mid);
            l = mid + 1;
            mid = (l + r) / 2;
        }
        else {
            r = mid - 1;
            mid = (l + r) / 2;
        }
    }
    System.out.println(ans);
}
}

```

```
package 第二章;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Scanner;

public class Ty10最小质因子之和 {

    static BufferedReader bReader = new BufferedReader(new
InputStreamReader(System.in));
    static PrintWriter out = new PrintWriter(new
OutputStreamWriter(System.out));
```

```

public static long[] EulerSun(int n) {

    int isprime[] = new int [n];

    Boolean[] isp = new Boolean [n + 5];

    // 这个用来存储 我 是 被谁筛出的
    int MQF[] = new int [n + 5];
    int count = 0;
    Arrays.fill(isp, true);
    Arrays.fill(MQF, 0);

    isp[0] = true;
    isp[1] = true;

    // 认为每个数都是素数
    for(int i = 2 ; i<= n; i++) MQF[i] = i;
    for(int i = 2; i <= n ; i++) {

        if(isp[i] == true) isprime[count++] = i;
        for(int j = 0 ; j < count && i * isprime[j] <= n; j++) {

            isp[i * isprime[j]] = false;
            // 更新 最小质因子
            MQF[i * isprime[j]] = isprime[j];
            if(i % isprime[j] == 0) break;
        }
    }
    long []anser = new long[n + 5];

    for(int i = 2 ; i <= n; i++)
        anser[i] = anser[i - 1] + MQF[i];

    return anser;
}

public static void main(String[] args) throws NumberFormatException,
IOException {

    // 打表 把 所有素数 先全部求出来
    int T = (int)3e6;

    // index 中存入的是 每个下标 对应的 最小的 质因子
    long index[] = EulerSun(T);

    int n = Integer.parseInt(bReader.readLine());

    int arr[] = new int[n];

    for(int i = 0 ; i < n; i++) {

        arr[i] = Integer.parseInt(bReader.readLine());

        out.println(index[arr[i]]);
    }
    out.flush();
}
}

```


算法

二分

```

# 闭区间写法
def lower_bound(nums: List[int], target: int) -> int:
    left, right = 0, len(nums) - 1 # 闭区间 [left, right]
    while left <= right: # 区间不为空
        # 循环不变量:
        # nums[left-1] < target
        # nums[right+1] >= target
        mid = (left + right) // 2
        if nums[mid] < target:
            left = mid + 1 # 范围缩小到 [mid+1, right]
        else:
            right = mid - 1 # 范围缩小到 [left, mid-1]
    return left # 或者 right+1

```

```

# 左闭右开区间写法
def lower_bound2(nums: List[int], target: int) -> int:
    left, right = 0, len(nums) # 左闭右开区间 [left, right)
    while left < right: # 区间不为空
        # 循环不变量:
        # nums[left-1] < target
        # nums[right] >= target
        mid = (left + right) // 2
        if nums[mid] < target:
            left = mid + 1 # 范围缩小到 [mid+1, right)
        else:
            right = mid # 范围缩小到 [left, mid)
    return left # 或者 right

```

```

# 开区间写法
def lower_bound3(nums: List[int], target: int) -> int:
    left, right = -1, len(nums) # 开区间 (left, right)
    while left + 1 < right: # 区间不为空
        mid = (left + right) // 2
        # 循环不变量:
        # nums[left] < target
        # nums[right] >= target
        if nums[mid] < target:
            left = mid # 范围缩小到 (mid, right)
        else:
            right = mid # 范围缩小到 (left, mid)
    return right # 或者 left+1

```

作者: 灵茶山艾府

链接: <https://leetcode.cn/problems/binary-search/solutions/2023397/er-fen-cha-zhao-zong-shi-xie-bu-dui-yi-g-eplk/>

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。

进制

统计各进制中各位数

```
/**
 *
 * Title: Sky数.java
 * Description: 获取 n 为进制 R的 各个数位的和
                可以 计算 将n 转换成 R进制 之后 每一位的数字 是多少
                缺点是 这个负数并不适用
 * @author Typecoh
 * @date 2022年12月7日
 * @version 1.0
 */
public static int getSky(int n ,int R) {

    int sum = 0;
    while( n > 0) {
        sum += n % R;
        System.out.println("各个位数" + n % R);
        n /= R;
    }

    return sum;

}
```

利用进制输出子集问题

```
package 蓝桥杯习题;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class 集合子集 {
    /**
     *
     * Title: 集合子集.java
     * Description: 通过进制数 表示 输出子集问题
     *              当前进制 位数 为1的时候 就输出相应的 集合中的 字母
     *              集合元素 {A,B,C,D} => 分别对应 1 1 1 1
     * @author Typecoh
     * @date 2022年12月7日
     * @version 1.0
     */

    public static void main(String[] args) {
```

```

char []arr = {'A','B','C','D'};

for (int i = 0; i < Math.pow(2, arr.length); i++) {

    int temp = i;

    String string = Integer.toString(temp,2);

    List<Character> list = new ArrayList<>();

    for(int j = 0 ; j < 4 - string.length(); j++) {
        list.add('0');
    }

    for(int j = 0; j < string.length(); j++) {
        list.add(string.charAt(j));
    }

    int index = 0;

    System.out.print("{");

    for (Character character : list) {
        if(character == '1') {
            System.out.print(arr[index]);
        }
        index++;
    }

    System.out.print("}");

    System.out.println();

}
}
}

```

位运算

与

&

或

|

异或

异或用来判断是否是整数幂

```
package 第二章;

import java.util.Scanner;

public class Ty01判断是否是2的整数幂 {

    /**
     *
     * Title: Ty01判断是否是2的整数幂.java
     * Description: 方式1 将 数字 转成 二进制 当 二进制中之后一个1 说明就是 二 的整数次
    幂
     * @author Typecoh
     * @date 2022年12月7日
     * @version 1.0
     */
    public static Boolean JudegPower1(int n) {

        String string = Integer.toString(n,2);

        int count = 0;

        for(int i = 0 ; i < string.length() ; i++ ) {
            if(string.charAt(i) == '1') {
                count++;
            }
        }

        if(count == 1) return true;

        return false;
    }

    /**
     *
     * Title: Ty01判断是否是2的整数幂.java
     * Description: 使用Integer 中 bitCount 统计 1 的个数 如果 个数 为 1
     *              就是 2 的整数幂 else 不是
     * @author Typecoh
     * @date 2022年12月7日
     * @version 1.0
     */
    public static Boolean JudegPower2(int n) {

        int count = 0;
```

```

        count = Integer.bitCount(n);

        return count == 1 ? true : false;
    }
    /**
     *
     * Title: Ty01判断是否是2的整数幂.java
     *
     * Description:利用位运算 n & n - 1 如果n 只有 一位 是 1 那个 n & n - 1必为0
     *              如果n 只有 多位 是 n & n - 1 最后一为1 到最后一位 都为零
     *              但是 之前的必为1
     *
     * @author Typecoh
     *
     * @date 2022年12月7日
     *
     * @version 1.0
     */
    public static Boolean JudegPower3(int n) {

        return (n & (n - 1)) == 0 ? true : false;
    }
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        for(int i = 0 ; i < 100 ; i++) {

            System.out.println("当前 " + " " + i);

            System.out.print(JudegPower1(i));

            System.out.print(" ");

            System.out.print(JudegPower2(i));

            System.out.print(" ");

            System.out.println(JudegPower3(i));

        }
    }
}

```

小性质

- 和 0 异或 等于本身

$$A \wedge 0 = A$$

- 和 自己异或 等于 0

$$A \wedge A = 0$$

移位

左移

- 每次左移一次 都是相当于 乘2

<<

右移

- 每次右移一次 并不是 除 2

>>

大整数

大整数乘法模拟

使用API处理

```
public static BigInteger BigNumberApi(String s1,String s2) {  
  
    BigInteger bigInteger = new BigInteger(s1);  
  
    bigInteger = bigInteger.multiply(new BigInteger(s2));  
  
    return bigInteger;  
}
```

模拟大数相乘

```
public static String BigNumerArr(String s1,String s2) {  
  
    int []arr3 = new int [s1.length() + s2.length()];  
  
    // 将字符串进行反转  
    String string1 = new StringBuilder(s1).reverse().toString();  
    String string2 = new StringBuilder(s2).reverse().toString();  
  
    for(int i = 0; i < string1.length(); i++) {  
        for(int j = 0 ; j < string2.length(); j++) {  
  
            /**  
             * 获取各位数字 进行 模拟 相乘  
             */  
            int ai = string1.charAt(i) - '0';
```

```

        int bi = string2.charAt(j) - '0';

        int ans = ai * bi;

        arr3[i + j] += ans;
    }
}

/*
 * 对每位 进行 进位操作
 * 对每位进行处理
 */

for(int i = 0 ; i < arr3.length - 1; i++) {

    arr3[i+1] += arr3[i] / 10;

    arr3[i] = arr3[i] % 10;

}

int index = arr3.length - 1;
while(arr3[index] == 0) {
    index--;

    if(index == -1) {
        System.out.println(0);
        return new String();
    }
}

for(int i = index ; i >= 0; i--) {
    System.out.print(arr3[i]);
}

return new String();
}

```

大整数加法

API

```

public static BigInteger BigNumAdd(String s1,String s2) {

    BigInteger bigInteger = new BigInteger(s1);

    bigInteger = bigInteger.add(new BigInteger(s2));

    return bigInteger;
}

```


模拟运算

```
public static String BigNumberAdd1(String s1,String s2){

    int len = Math.max(s1.length(), s2.length());

    int []arr = new int [len + 1];

    s1 = new StringBuilder(s1).reverse().toString();
    s2 = new StringBuilder(s2).reverse().toString();

    int ai = 0;
    int bi = 0;

    for(int i = 0 ; i < len; i++) {

        // 分别 获取 字符串的 数字
        if(s1.length() <= i ) {
            ai = 0;
        }

        else {
            ai = s1.charAt(i) - '0';
        }

        if(s2.length() <= i ) {
            bi = 0;
        }

        else {
            bi = s2.charAt(i) - '0';
        }

        arr[i] = ai + bi;
    }

    /**
     * 对数据进行 进位 处理
     */

    for(int i = 0 ; i < arr.length - 1; i++) {

        arr[i+1] += arr[i]/10;
        arr[i] = arr[i] % 10;
    }

    StringBuilder sBuilder = new StringBuilder();

    // 将数字 字符串处理
    for (int i : arr) {
        sBuilder.append(i);
    }

    sBuilder = sBuilder.reverse();

    // 判断最高位是否有进位
```

```

        if(sBuilder.charAt(0) == '0') {
            sBuilder.deleteCharAt(0);
        }

        return sBuilder.toString();
    }
}

```

```

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    String string1 = scanner.next();
    String string2 = scanner.next();

    System.out.println(BigNumAdd(string1, string2));
    System.out.println(BigNumberAdd1(string1, string2));

}

```

素数

常规素数 sqrt(n)

线性复杂度 $O(n * \sqrt{n})$

```

package 第二章;

import java.util.Scanner;

public class Ty06素数判定 {

    /**
     *
     * Title: Ty06素数判定.java
     *
     * Description: 判断 n 是否是一个 素数
     *              当 x 是 n 的一个约数的时候
     *               $n \% x = b \implies x * b = n$ 
     *              1, 2, ..... x b ..... n
     *              如果说 2 ~ x 中存在 一个数是 n的约数 那么 x~n中必定存在 是 另外一个约数
     *              如果 sqrt(n) 能被 n 整除 那么 n / sqrt(n) 也能被 整除
     *
     * @author Typecoh
     *
     * @date 2022年12月8日
     *
     * @version 1.0
     */

    public static Boolean isPrime(int n) {

        if(n <= 1) return false;
    }
}

```

```

        for(int i = 2; i * i <= n; i++)

            if( n % i == 0) return false;

        return true;
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        System.out.println(isPrime(n));

    }
}

```

埃式素数筛

复杂度 $O(N\log n)$

```

package 第二章;

import java.util.Arrays;

public class Ty07埃式素数筛 {
    /**
     *
     * Title: Ty07埃式素数筛.java
     *
     * Description: 埃式筛
     *             思想: 将素数的倍数筛掉
     *             用 isprime 这个数组 存储当前的 数字 是不是 素数
     *             i 表示 当前这个 素数 j 表示 当前那个素数的 倍数 筛出
     *             isprime[i*j] = false;
     *
     * @author Typecoh
     *
     * @date 2022年12月8日
     *
     * @version 1.0
     */
    public static int StrombergSieve(int n) {

        Boolean isprime[] = new Boolean[n + 1];

        Arrays.fill(isprime, true);
    }
}

```

```

// 初始化 将0 和 1 标记为 false
isprime[0] = false;
isprime[1] = false;

for(int i = 2; i <= n; i++) {

    // 筛掉 素数的倍数
    if(isprime[i] == true) {
        // 从2的倍数 开始 筛除
        for(int j = 2; i * j<=n ; j++) {
            isprime[i*j] = false;
        }
    }
}

for(int i = 0 ; i < isprime.length; i++) {
    if(isprime[i] == true) {
        System.out.println(i);
    }
}

return 0;
}

public static void main(String[] args) {

    int n = 100;

    StrombergSieve(n);

}
}

```

欧拉筛

欧拉筛 是 数学家欧拉 发明的
复杂度 $O(n)$
欧拉筛!!!

```

public class Ty08欧拉筛 {

    public static void EulerSun(int n) {

        // 设置一个数组 进行存储 当前的素数
        int primes[] = new int [n];

        // 设置一个数组 说明当前元素是不是素数
        Boolean isprimes[] = new Boolean [n + 5];

        // 对数组 进行初始化设置
        Arrays.fill(ips, true);

        isprimes[0] = false;
        isprimes[1] = false;
    }
}

```

```

// 设置一个 素数个数 统计
int count = 0;

for(int i = 2 ; i <= n; i++) {

    // 判断 当前这个元素 是否是素数
    if(ips[i] == true) {
        // 如果是 素数 就将这个数 存储到数组当中去
        primes[count++] = i;
    }

    // 遍历所用的 素数 将 当前 所有的 素数 的 i倍 筛出掉
    for(int j = 0 ; j < count && i * primes[j] <= n; j++) {
        // 将当前的数 设置为 false
        isprimes[i * primes[j]] = false;

        /*

        欧拉筛的原则 是 若果这个数不是 素数
        那么这个数要被自己最小的那个 素因子 给筛出掉
        i 如果是 isprime[j] 的 倍数 i = isprime[j] * x
        isprime[j + 1] * i = isprime[j] * x * isprime[j + 1] = y
        这个 y 的 最小素因子 明显是 isprime[j]
        但是被筛选的是 isprime[j + 1] 于是 和 欧拉筛的 思想不符合
        if( i % isprime[j] == 0) break; 最为 核心的一句
        */
        if( i % primes[j] == 0) break;
    }

}

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

}

public static void main(String[] args) {

    int n = 100;

    EulerSun(n);

}
}

```

算术基本定理

每个大于1的自然数，要么本身就是质数，要么可以写为2个或以上的质数的积

$n = p_1^{x_1} * p_2^{x_2} * p_3^{x_3} * p_4^{x_4} \dots p_n^{x_n}$ p_i 均为质数

任意一个正整数n最多只有一个质因数大于根号n

$$1 \ A > \sqrt{n}, B > \sqrt{n} \implies A * B > n$$

1.

$$2 \ n \geq A * B > n$$

1 和 2 自相矛盾|

```
package 第二章;

import java.util.Scanner;

public class Ty11质因数分解 {

    // prime factor decomposition
    public static void PFD(long n) {

        // 先求解 质因数 在 2 - sqrt(n)
        // 用来标记答案的个数
        int ans = 0;
        for(long i = 2; i <= n / i ;i++) {

            // 如果 n % i == 0 说明 i是 n的展开式中的一项 ==> i 是 n的一个质因子
            if(n % i == 0) {
                ans++;
                // 求这个质因子在n中的个数
                // 求除去这些质因子剩下的那个数的质因子有哪些
                // 注意这里的n变化了 变小了!!! 因此 时间复杂发是 o(n)
                while(n % i == 0) n /= i;
            }
        }

        // 剩下 最后的一项如果不是1
        // 那这个数 本身就是一个质数
        if(n > 1) ans++;

        System.out.println(ans);
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        long n = scanner.nextLong();

        PFD(n);
    }
}
```

阶乘小知识

问题描述:

$N!$ 的末尾恰好有 K 个 0

怎么考虑这个问题? 10 的由来 是 $2 * 5$ 那就可以转成 1 - N 中 这些数 可以 提供多少个5的倍数问题 (为什么不是2的个数 因为 2的个数大于 5的个数 我们统计的是小的那个)

比如 10 可以 提供两个分别是 5 10 各自提供一个

20 可以提供四个 2 10 15 20

25 可以 提供 $5 + 1$ 分别 是 5 10 15 20 25 (提供2个)

```
public static long check(long n) {  
  
    long ans = 0;  
  
    while(n > 0) {  
        ans += n / 5;  
        n /= 5;  
    }  
    return ans;  
}
```

十大排序

冒泡排序

```
/**  
 *  
 * Title: 冒泡排序.java  
 * Description: 冒泡排序核心思想就是: 每次比较两个相邻元素, 每次将相邻元素进行对比  
 *             如果升序排序, 就每一趟将最大的元素放置在本次序列的最后一个  
 * @author Typecoh  
 * @date 2022年12月25日  
 * @version 1.0  
 */  
public static int[] BubbleSort(int arr[]) {  
  
    int []BubbleSort = arr;  
  
    for(int i = 0 ; i < BubbleSort.length; i++) {  
        for(int j = 0 ; j < BubbleSort.length - i - 1 ; j++) {  
            if(BubbleSort[j] > BubbleSort[j + 1]) {  
                int temp = BubbleSort[j+1];  
                BubbleSort[j+1] = BubbleSort[j];  
                BubbleSort[j] = temp;  
            }  
        }  
    }  
  
    return BubbleSort;  
}
```

插入排序

```
/**
 *
 * Title: temp.java
 *
 * Description: 插入排序：核心思想是：
 *              将当前元素 插入到 前面一个有序的序列当中去
 *              每次初始化的位置 index = i
 *              记录当前元素因该插入的位置index
 *              将[index,i]的元素集体向后移
 *
 * @author Typecoh
 *
 * @date 2022年12月25日
 *
 * @version 1.0
 */

public static void InsertSort(int []arr,int len) {

    // 将当前元素插入到之前 有序队伍中
    for(int i = 1 ; i < len; i++) {

        int value = arr[i];
        int index = i;
        for(int j = 1 ; j <= i ; j++) {
            if(value < arr[j]) {
                index = j;
                break;
            }
        }
        System.out.println("index = " + index + " " + "i = " + i);
        // 需要 插入在 下标为 index 位置
        // 需要将 [index,i] 的位置全部移动 j - 1 = index j - 1 >= index
        for(int j = i; j - 1 >= index; j--) {
            arr[j] = arr[j - 1];
        }

        arr[index] = value;

        for(int j = 1; j < len; j++)
            System.out.print(arr[j] + " ");
        System.out.println();
    }
}

public static void main(String[] args) {

    int []arr = {0,28,34,30,26,36,27,28,20,22};

    int len = arr.length - 1;
    for(int i = 1 ; i <= len; i++) System.out.print(arr[i] + " ");
    System.out.println();

    InsertSort(arr,len);
}
```



```
}
```

选择排序

```
/**
 *
 * Title: temp.java
 * Description: 选择排序:
 * 核心就是 每次找出[i,n] 区间的最值 将这个最值放入到 arr[i] 中
 * @author Typecoh
 * @date 2022年12月25日
 * @version 1.0
 */

public static void SelectSort(int []arr) {

    int minvalue = arr[0];

    for(int i = 0; i < arr.length; i++) {
        minvalue = arr[i];
        for(int j = i; j < arr.length ; j++) {
            // 每一次选出最小值
            if(minvalue > arr[j]) {
                int temp = minvalue;
                minvalue = arr[j];
                arr[j] = temp;
            }
        }
        // 这个arr[i] 如果是这一趟的最值 就相当于没变
        // 如果不是 最值 那么 这个值 至少出现过两次
        arr[i] = minvalue;

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

希尔排序

```
/**
 *
 * Title: temp.java
 *
 * Description: 希尔排序:
 * 核心是增量减半
 * 对于 gap = len / 2 gap = gap / 2
 * 从 gap -- len 进行 计算
 * 每次计算 j - gap j - 2gap j - 3gap
 * 出界判断 j - gap > 0 交换判断 arr[j - gap] > arr[j]
```

```

* @author Typecoh

* @date 2022年12月25日

* @version 1.0
*/

public static void ShellSort(int []arr,int len) {

    for(int gap = len / 2; gap > 0; gap /= 2) {

        // 对每一个 分组 进行插入排序
        for(int i = gap; i <= len; i++) {

            int j = i; // 存放的是 当前要处理的 那个元素 从 j - gap
            while( j - gap >= 0 && arr[j - gap] > arr[j]) {
                int temp = arr[j];
                arr[j] = arr[j - gap];
                arr[j - gap] = temp;
                j = j - gap;
            }
        }

        for(int i = 1; i <= len; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}

public static void main(String[] args) {

    int []arr = {0,28,34,30,26,36,27,28,20,22};

    int len = arr.length - 1;

    shellSort(arr,len);

}

```

计数排序

```

public static void GetValue(int []arr) {

    maxvalue = arr[0];
    minvalue = arr[0];

    for(int i = 0 ; i < arr.length; i++) {
        if(maxvalue < arr[i]) maxvalue = arr[i];
        if(minvalue > arr[i]) minvalue = arr[i];
    }
}

```

```

static int maxvalue = 0;
static int minvalue = 0;

public static void main(String[] args) {

    // 原始数组
    int []arr = {28,34,30,26,36,27,28,20,22};

    GetValue(arr);

    int dis = maxvalue - minvalue + 1;

    // 存放数组
    // c数组用来存放 原始数组的大小和个数
    int []c = new int[dis];

    for(int i = 0 ; i < arr.length; i++) {

        // arr[i] - minvalue 表示这个数的下标数
        c[arr[i] - minvalue]++;
    }

    // 输出 c 中元素
    for(int i = 0 ; i < c.length; i++) System.out.printf("%3d",c[i]);
    System.out.println();
    // 存放最终答案
    int []ans = new int [arr.length];
    // 定义一个前缀和数组标记 当前元素有几个小于等于他
    int []pre= new int[dis];

    pre[0] = c[0];
    for(int i = 1; i < c.length; i++) pre[i] = c[i] + pre[i - 1];
    for(int i = 0 ; i < c.length; i++) System.out.printf("%3d",i + minvalue
);

    System.out.println();
    // 输出 ans中应该存放的位置
    for(int i = 0 ; i < pre.length; i++) System.out.printf("%3d",pre[i]);

    // c数组中有 某个元素可能有多
    for(int i = arr.length - 1; i >= 0; i--) {

        // 求出 小于等于 arr[i]中的数是多少个
        // pre[arr[i] - minvalue - 1] 表示 arr[i] 的下标 表示因该在b中存放的位置
        // arr[i] - minvalue 表示 前缀和数组的下标
        ans[pre[arr[i] - minvalue] - 1] = arr[i];
        pre[arr[i] - minvalue]--;
    }
    System.out.println();
    System.out.println("-----");
    for(int i = 0 ; i < ans.length; i++) System.out.print(ans[i] + " ");
}

```

桶排序

```
public static int[] BucketSort(int []arr) {

    for (int i : arr) {
        System.out.print(i + " ");
    }
    System.out.println();
    // 找出 数组 最大最小值
    int max = arr[0];
    int min = arr[0];

    for (int i : arr) {
        max = Math.max(max, i);
        min = Math.min(min, i);
    }

    int len = arr.length;
    // 存放答案
    int []ans = new int [len];

    // 创建一个桶里面 有多个元素
    int [][]bucket = new int [len + 1][];

    for(int i = 0; i < len; i++) {

        int index = (int) ((arr[i] - min) * 1.0 / (max - min) * len);
        // 给这个桶里面放入 当前元素

        bucket[index] = append(bucket[index], arr[i]);
    }

    // 记录 最终ans
    int count = 0;
    for(int i = 0; i <= len; i++) {

        if(bucket[i] == null) continue;
        for(int j = 0; j < bucket[i].length; j++) {
            ans[count++] = bucket[i][j];
        }
    }

    return ans;
}

public static int [] append (int []bucket,int value) {

    if(bucket == null) return new int[]{value};

    else {

        // 赋值给新数组
        int []arr = Arrays.copyOf(bucket, bucket.length + 1);

        // 将 value 和 所有的数组进行遍历
        // 如果 是 按照升序排序 就 从后往前比较
        int index;
```

```

        for( index = arr.length - 2 ; index >= 0 && value < arr[index];
index--)
            arr[index + 1] = arr[index];
        // 插入在 index 前面那个位置
        arr[index + 1] = value;

        return arr;
    }
}

public static void main(String[] args) {

    // 原始数组
    int []arr = {28,34,30,26,36,27,28,20,22};

    int [] ans = BucketSort(arr);

    for (int i : ans) {
        System.out.print(i + " ");
    }
}

```

堆排序

```

public static void HeapSort(int[] tree, int n) {

    BuildHeap(tree, n);
    for(int i = n - 1; i >= 0; i--) {
        Swap(tree, i, 0);
        BuildHeap(tree, i);
    }
}

/**
 * name heapfiy
 * tree 所有树的节点 n 所有节点数量 point 当前节点
 * 形成一个根堆
 */
public static void heapify(int []tree, int n, int point) {

    if(point >= n) return;
    // 左右子树
    int leftchild = 2 * point + 1;
    int rightchild = 2 * point + 2;
    // 定义当前最大节点是 位于 root 节点
    int max = point;
    // 找到 根节点 左子树 右子树 三者最大值
    if(leftchild < n && tree[max] < tree[leftchild]) max = leftchild;
    if(rightchild < n && tree[max] < tree[rightchild]) max = rightchild;

    // 交换 root 和 max
    if(max != point) {
        Swap(tree,max,point);
        // 为什么 这里需要 heapify?
    }
}

```

```

        * 当 自己的 左右子树中的某一个 和 root 进行交换之后
        * 那个被交换的 子树 可能不满足堆 需要从新排列
        */
        heapify(tree, n, max);
    }
}

public static void Swap(int []tree,int max,int root) {
    int temp = tree[max];
    tree[max] = tree[root];
    tree[root] = temp;
}

public static void BuildHeap(int []tree,int n) {

    int LastNode = n - 1;
    int Parent = (LastNode - 1) / 2;

    for(int i = Parent; i >= 0; i--) heapify(tree, n, i);
}

public static void main(String[] args) {

    int []tree = {28,34,30,26,36,27,28,20,22};

    int n = tree.length;

    HeapSort(tree, n);

    for (int i : tree) System.out.print(i + " ");
}

```

归并排序

```

package 第二章;

import java.lang.reflect.Array;
import java.util.Arrays;

public class Ty59归并排序 {

    public static void mergeSort(int a[], int left, int right) {

        int mid = ( left + right ) / 2;
        System.out.println("left == > " + left + " right == >" + right);
        if(left < right) {
            // 使用递归进行 减小划分
            mergeSort(a, left, mid);
            mergeSort(a, mid+1, right);
            // 合并操作
            merge(a, left, mid, right);
        }
    }

    public static void merge(int a[],int left,int mid,int right) {

```

```

        int l = left;
        int r = mid+1;
        int index = 0;
        int []temp = new int [right - left + 1];
        while(l <= mid && r <= right) {
            if(a[l] < a[r]) temp[index++] = a[l++];
            else temp[index++] = a[r++];
        }
        // 当 一端结束了 剩下的 加入到数组中去
        while(l <= mid) temp[index++] = a[l++];
        while(r <= right) temp[index++] = a[r++];
        // 返回原数组的值
        for(int i = 0 ; i < temp.length; i++) a[left + i] = temp[i];
    }

    public static void main(String[] args) {

        int a[] = { 51, 46, 20, 18, 95, 67, 82, 30};

        mergeSort(a, 0, a.length - 1);

        System.out.println(Arrays.toString(a));
    }
}

```

基数排序

快速排序

```

package 第二章;

import java.util.Arrays;

public class Ty60快速排序 {

    public static void quickSort(int a[], int left, int right) {

        if(left >= right) return;

        int l = left;
        int r = right;

        int temp = a[l];

        while(l != r) {
            while(a[r] >= temp && r > l) r--;
            if(r > l) a[l++] = a[r];

            while(a[l] < temp && l < r) l++;
            if(l < r) a[r--] = a[l];
        }
    }
}

```

```

        System.out.println("l == " + l + " " + "r == " + r);
        // 通过一些列交换找到了 第一个哨兵的位置 并放在了正确的位置
        // 左边总是比哨兵小 右边总是比哨兵大
        // l 和 r 永远相等的 所以 a[l] = temp === a[r] = temp
        a[l] = temp;

        quickSort(a, left, l - 1);
        quickSort(a, l + 1, right);
    }

    public static void main(String[] args) {

        int a[] = {49, 46, 20, 18, 95, 67, 82, 49};

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

        System.out.println(Arrays.toString(a));
    }
}

```

对象排序

compare排序原理

- compare返回值是Int类型，三种情况正数、0、负数。
- compare如果比较的是Int、Float、Double类型的话，直接以值进行比较。
- compare如果比较的是String、char类型的数据，它会从两个数据的第一个字符开始比较，如果不同，就返回两个字符的ASCII差值
- 如果字符都相同，那么再比较两个字符串的长度，最后返回两个字符串长度的差值。

compare比较如果返回的是正数，那么两个数要进行交换，如果结果为负数和0，两个数都不会进行交换。

对于compare(Object o1,Object o2)而言

- 如果要按照对象的某一个属性进行升序排序，那么直接返回o1.属性值-o2.属性值（因为如果第一个数比第二个数大，会返回正值，两个对象要进行交换）。
- 如果要按照对象的某一个属性进行降序排序，那么直接返回o2.属性值-o1.属性值（因为如果第二个数比第一个数大，会返回正值，两个对象要进行交换）。

以如下代码分析

```

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.Map.Entry;

public class Main {

    public static void main(String[] args) {

```



```

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();
int m = scanner.nextInt();

int []arr = new int [n + 1];
int []dp = new int [n + 1];

HashMap<Integer, Integer> hashMap = new HashMap<>();

for(int i = 1 ; i <= n; i++) {
    int temp = i;
    int ans = 0;
    while(temp > 0) {
        ans += temp % 10;
        temp /= 10;
    }
    hashMap.put(i, ans);
}

List<Map.Entry<Integer, Integer>> list = new ArrayList<>();

list.addAll(hashMap.entrySet());

Collections.sort(list, new Comparator<Map.Entry<Integer, Integer>>() {

    @Override
    public int compare(Map.Entry<Integer, Integer> o1,
Map.Entry<Integer, Integer> o2) {
        // 如果 o1.getvalue > o2.getvalue 就交换 o1, o2
        if(o1.getValue() != o2.getValue()) return o1.getValue() -
o2.getValue();
        else return o1.getKey() - o2.getKey();
    }
});

int value = 0;
int count = 0;
for (Entry<Integer, Integer> entry : list) {
    count++;
    if(count == m) {
        value = entry.getKey();
    }
}
System.out.println(value);
}
}

```

首先说一下这个Collections这个类，自带sort方法，这个sort只能对list进行排序

```

Collections.sort(list, new Comparator<Map.Entry<Integer, Integer>>() {

    @Override
    public int compare(Map.Entry<Integer, Integer> o1, Map.Entry<Integer, Integer> o2) {
        // 如果 o1.getValue() > o2.getValue() 就交换 o1, o2
        if(o1.getValue() != o2.getValue()) return o1.getValue() - o2.getValue();
        else return o1.getKey() - o2.getKey();
    }
});

```

当我们需要按照自定义方式进行排序，需要创建一个Comparator对象，在里面重写compare方法

Comparator这个构造器中传入的参数是 Map.Entry<Integer, Integer>

和 list 中的对象List<Map.Entry<Integer, Integer>> list = new ArrayList<>(); 是一致的

那这个 Map.Entry<Integer, Integer> 到底是个什么东西呢?

```

public void test02(){
    Map<Integer,String> map = new HashMap<>();
    map.put(1, "莫德里奇");
    map.put(2, "罗纳尔多");
    map.put(3, "马拉多纳");
    map.put(4, "克鲁伊维特");
    Set<Map.Entry<Integer,String>> entrySet = map.entrySet();
    for (Map.Entry<Integer, String> entry : entrySet) {
        System.out.println(entry.getKey() + "==" + entry.getValue());
    }
}

```

- Map是一个接口，Entry是Map内部接口 所以可以使用Map.Entry
- Entry<T,T>是一个泛型
- entrySet(); 是Map中的一个方法 Set<Map.Entry<K, V>> entrySet();得到一个set对象 这个对象是存在key-value 因此 Entry
中提供了getValue() 和 getKey()方法获取值
- Comparator参数就是Entry<T,T>形式

有了Comparator这个对象，那里面的compare方法是怎么原理是什么?

```

@Override
public int compare(Map.Entry<Integer, Integer> o1, Map.Entry<Integer, Integer> o2) {
    // 如果 o1.getValue() > o2.getValue() 就交换 o1, o2
    if(o1.getValue() != o2.getValue()) return o1.getValue() - o2.getValue();
    else return o1.getKey() - o2.getKey();
}

```

自定义排序方式 同时需要考虑到 key-value

compare比较如果返回的是正数，那么两个数要进行交换，如果结果为负数和0，两个数都不会进行交换。

Arrays.sort重写

```
Integer [] pre = new Integer [100];

Arrays.sort(pre,new Comparator<Integer>() {

    @Override
    public int compare(Integer o1, Integer o2) {
        // TODO Auto-generated method stub
        return 0;
    }
});
```

快速幂

快速幂的核心思想就是“降幂增底”

```
while(b > 0) {

    if(b % 2 != 0)
        ans = ans * a % p;
    b = b / 2;
    a = a * a % p ;
}
```

对某个数取余 连续取余不会影响结果

累加求末n尾数

在相加的时候对某n尾取余数是没有影响的

```
#include <iostream>

using namespace std;

int main()
{
    long n = 0;

    cin >> n;

    int ans = 0;
    for (long i = 1; i <= n; i++)
        ans = (ans + i) % 100;

    cout << ans;
    return 0;
}
```

并查集

用来判断两个点，是不是在同一个连通分量中。

Kruskal算法生成最小树就是使用并查集进行合并操作

并查集三大操作：

- 初始化

```
public static void init() {  
    for(int i = 0; i < parent.length; i++)  
        parent[i] = i;  
}
```

- 合并

```
public static void merge(int x, int y) {  
  
    int LeftParent = find(x);  
    int RightParent = find(y);  
  
    parent[LeftParent] = RightParent;  
}
```

- 查找
 - 不剪枝

```
public static int find(int point) {  
  
    if(parent[point] == point) return point;  
    else return find(parent[point]);  
}
```

- 剪枝

```
public static int find(int point) {  
  
    if(parent[point] == point) return point;  
  
    else parent[point] = find(parent[point]);  
  
    return parent[point];  
}
```

```
package 第二章;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.HashMap;  
import java.util.Iterator;  
import java.util.Map;
```

```

import java.util.Map.Entry;
import java.util.Scanner;

/*
 * Long 最大能到 9e18
 */
public class Ty75并查集 {

    /**
     *
     * Title: Ty75并查集.java
     * Description:
     * @author Typecoh
     * @date 2023年2月28日
     * @version 1.0
     *
     * 并查集主要分为 3个操作
     *     1 初始化
     *     2 查找
     *     3 合并
     */

    static int parent[] = new int [(int)2e5+1];

    /**
     * 元素初始化操作
     * 一开始自己的指向自己 自己的祖先就是自己
     */
    public static void init() {

        for(int i = 0 ; i < parent.length; i++) parent[i] = i;
    }

    /**
     * 查找 point 的祖先
     * 没有剪枝
     */
    // public static int find(int point) {
    //     if(parent[point] == point) return point;
    //     else return find(parent[point]);
    // }
    /**
     * 进行剪枝操作
     */
    public static int find(int point) {

        if(parent[point] == point) return point;

        else parent[point] = find(parent[point]);
    }

```

```

        return parent[point];
    }

    /*
     * 将 x 和 y 进行合并
     */
    public static void merge(int x, int y) {

        int LeftParent = find(x);
        int RightParent = find(y);

        parent[LeftParent] = RightParent;
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();
        int m = scanner.nextInt();

        int arr[] = new int [n];

        init();

        for(int i = 0; i < m; i++) {

            int option = scanner.nextInt();
            int x = scanner.nextInt();
            int y = scanner.nextInt();

            if(option == 1) merge(x, y);
            else {
                if(find(x) == find(y)) System.out.println("YES");
                else System.out.println("NO");
            }
        }
    }
}

```

种类/扩展域并查集

一般的并查集，维护的是具有连通性、传递性的关系，例如**亲戚的亲戚是亲戚**。但是，有时候，我们要维护另一种关系：**敌人的敌人是朋友**。种类并查集就是为了解决这个问题而诞生的。

对于AB互斥这种 因该是有两个并查集的集合分别放置A和B

数组设置为原来的两倍 a[2N] 前n个放置正常的数据 后n个放置 相反（对立的）的数据

```
static int parent[] = new int [(int)5e5 * 2 + 1]; // 二维举例 数组翻倍
```

初始化 合并 查找三个函数是不变的

```
public static void init() {
```

```

        for(int i = 0; i < parent.length; i++) parent[i] = i;
    }

    public static int find(int point) {

        if(parent[point] == point) return point;

        else parent[point] = find(parent[point]);

        return parent[point];
    }

    public static void merge(int x, int y) {

        int LeftParent = find(x);
        int RightParent = find(y);

        parent[LeftParent] = RightParent;
    }

```

```

// 本义是 x 和 y 不相同
if(find(x) == find(y)) {
    // 如果 x 和 y 是相同的 和本义相反 因此有问题
    System.out.println(x);
    break;
}
// 何合并 和 自己不相同元素的 对立面
// x 和 y 不相同 则 x 合并 y 的对立面 y 合并 x 的对立面
else {
    merge(x, y + N);
    merge(x + N, y);
}

```

带权并查集

板子

```

class UF{

    int []parent,dist,value;

    public UF(int n) {

        parent = new int [n];
        dist = new int [n];
        value = new int [n];

        for(int i = 1; i < n; i++) {
            parent[i] = i;
        }

        Arrays.fill(value, 1);
    }
}

```

```

}

public int find(int x) {
    if(x == parent[x]) return parent[x];
    // 先记录 祖宗
    int root = find(parent[x]);
    // 加上父亲的距离
    dist[x] += dist[parent[x]];
    // 指向祖宗
    return parent[x] = root;
}

public boolean same(int x, int y) {
    return find(x) == find(y);
}

public boolean merge(int x, int y) {

    x = find(x);
    y = find(y);

    if(x == y) return false;
    dist[y] += value[x];
    value[x] += value[y];

    parent[y] = x;
    return true;
}

public int size(int x) {
    return value[find(x)];
}

public int dist(int x, int y) {

    if(!same(x, y)) return -1;
    return Math.abs(dist[x] - dist[y]) - 1;
}
}

```

差分和前缀和

第14届蓝桥杯 Java 组省赛 X | 第14届蓝桥杯 Java 组省赛 X | 蓝桥幼儿园 - 蓝桥云课 X | 蓝

→ oi-wiki.org/ds/dsu/

(18条消息) 执梗的... 哔哩哔哩 (゜-゜)つ... Codeforces AtCoder 首页 - 洛谷 | 计算...

并查集

数据结构
数据结构部分简介

的父亲设为自己。

实现

a数组	0	9	5	2	7	5	2	1	1	3	1	4
前缀和b	0	9	14	16	23	28	30	31	32	35	36	40
下标	0	1	2	3	4	5	6	7	8	9	10	11

数组 a 是输入的数据， b数组是前缀和数组

b是a的前缀和数组，同时a也是b的差分数组

$$b[i] = b[i - 1] + a[i]$$

$$a[i] = b[i] - b[i - 1]$$

④ $b[i \rightarrow j]$ 全部加上一个差值 k \Rightarrow $\left. \begin{array}{l} \text{for } (i \rightarrow j) \quad b[i] + k \\ a[i] + k \\ a[j+1] - k \end{array} \right\}$

当 $a[i]$ 加上一个 k 之后

$$b[i] = a[1] + a[2] + \dots + a[i] + k + \dots + a[n]$$

$$b[i + 1] = a[1] + a[2] + \dots + a[i] + k + \dots + a[n] + a[n + 1]$$

$$b[i + 2] = a[1] + a[2] + \dots + a[i] + k + \dots + a[n] + a[n + 1] + a[n + 2]$$

证明：差分数组 $a[i] + k$ ，看 b 数组变化

$$\left. \begin{array}{l} b[i] = a[1] + a[2] + \dots + a[i] + k \\ b[i+1] = a[1] + a[2] + \dots + a[i] + k + a[i+1] \\ \dots \\ b[j] = a[1] + a[2] + \dots + a[i] + k + a[i+1] + \dots + a[j] \\ b[j+1] = a[1] + \dots + a[i] + k + \dots + a[j] + a[j+1] \\ \vdots \\ b[n] = a[1] + \dots + a[i] + k + \dots + a[j] + a[j+1] + \dots + a[n] \end{array} \right\}$$

如果能让 前缀和数组 $[left, right]$ 中的每一个元素加一个 k ：只需要 让 $a[left] + k$ && $a[right+1] - k$

对于每一个数组都有自己的前缀和数组也有自己差分数组

逆元

搜索

剪枝

将整数 n 分成 k 份，且每份不能为空，任意两种划分方案不能相同(不考虑顺序)。例如： $n=7$ ， $k=3$ ，下面三种划分方案被认为是相同的。

```
1 1 5
1 5 1
5 1 1
```

```
package 搜索;

import java.util.Scanner;

public class Ty05DFS剪枝 {

    public static int ans = 0;
    public static int count = 0;
    /**
     *
     * Title: Ty05DFS剪枝.java
     *
     * n 总数 k 份数 min 当前这个数最小不能小于的数
     * Description: dfs(int n, int k, int min, String femshu)
     * @author Typecoh
     * @date 2023年3月1日
     * @version 1.0
     */

    public static void dfs(int n, int k, int min, String femshu) {

        count++;

        if(k == 1 && min <= n) {

            ans++;
            System.out.println(femshu + n + " ");

            return ;
        }

        /**
         * k 份 每一份 最小是 min 那么 总额最小就是 k * min <= n
         */
        if(k * min > n) return;

        for(int i = min; i <= n; i++) {

            dfs(n - i, k - 1, i, femshu + i + " ");
        }
    }
}
```

```

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();
        int k = scanner.nextInt();

        // 将 n 分成 k份 每一份都是 从最小的那个数开始划分
        // 当确定了 第 i 份数之后 在 搜索 第i+1个数
        dfs(n,k,1,"");

        System.out.println(ans);
        System.out.println(count);

    }
}

```

一个整数可以划分成若干个不超过自己的整数之和的形式。例如：

4

4=1+1+1+1

4=1+1+2

4=1+3

4=2+2

4=4

总共有5种划分形式，为了方便判题，我们约定

1) 这些加数必须遵循从小到大的原则。

2) 4=1+3 和4=3+1 当做一种划分

```

package 搜索;

import java.util.Scanner;

public class Ty06DFS实战整数划分 {

    public static int ans = 0;

    /**
     *
     * Title: Ty06DFS实战整数划分.java
     * Description:
     * @author Typecoh
     * @date 2023年3月1日
     *
     * dfs(int n, int nowget, int maxuse, String fenshu)
     *

```

```

    * n 表示最终的这个数 nowget 表示现在得到的数 maxuse 表示 使用的最大的这个数
    * 为了 控制是升序传递 fenshu 表示 由那几个构成
    * 4 = 1 1 1 1

    * @version 1.0
    */
    public static void dfs(int n, int nowget, int maxuse, String fenshu) {

        if(nowget == n) {
            ans++;
            System.out.println(fenshu);
            return;
        }

        for(int i = 1; i <= n - nowget; i++) {

            if(i >= maxuse) {
                dfs(n, nowget + i, i, i + "+" + fenshu);
            }
        }
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        dfs(n, 0, 0, "");

        System.out.println(ans);
    }
}

```

回溯

```

if(b[i] == 0) { // 表示这个数字是否使用过
    a[step] = i; // 将这个数字放在这个位置 没有过
    b[i] = 1; // 标记这个数字使用过了
    dfs(step + 1); // 进行下一步搜索
    // 回溯 将自己这一次 记录取消
    /**
        * 标记这个数没有被使用过，并且这个数
        * 不会在出现在这个位置上
        */
    b[i] = 0; // 这一步回溯
}

```

```

package 搜索;

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

```

```

public class Ty07全排列回溯剖析 {

    public static int n = 5;
    public static int cnt = 0;
    public static int a[] = new int[10];
    public static int b[] = new int[10];
    public static int count = 0;
    // step 表示 当前在第几个位置
    public static void dfs(int step) {

        cnt++;
        // 如果 step = n + 1 说明 前n不 已经排好了
        if(step == n + 1) {
            count++;
            System.out.println("---");
            for(int i = 1; i <= n; i++) System.out.print(a[i]);
            System.out.println();
            // Arrays.toString(a);
            return;
        }
        for(int i = 1; i <= n; i++) {
            if(b[i] == 0) { // 表示这个数字是否使用过
                a[step] = i; // 将这个数字放在这个位置 没有过
                b[i] = 1; // 标记这个数字使用过了
                dfs(step + 1); // 进行下一步搜索
                // 回溯 将自己这一次 记录取消
                /**
                 * 标记这个数没有被使用过，并且这个数
                 * 不会在出现在这个位置上
                 */
                b[i] = 0; // 这一步回溯
            }
        }
        return;
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        dfs(1);

        System.out.println(count);

        System.out.println(cnt);
    }
}

```

BFS

```

package 搜索;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;

```

```

import java.util.Queue;
import java.util.Scanner;

import javax.lang.model.type.IntersectionType;

public class Ty09迷宫2 {

    public static int count = 0;
    public static int cnt = 0;

    public static void BFS(int n) {

        Queue<String> queue = new LinkedList<>();

        for(int i = 1; i <= n; i++) {
            queue.add(i + "");
        }

        while(true) {

            if(queue.size() == 0) {
                System.out.println(count);
                System.out.println(cnt);
                break;
            }

            String string = queue.element();

            if(string.length() == n) {
                System.out.println(string);
                count++;
            }

            for(int i = 1; i <= n; i++) {
                cnt++;
                // 队头元素
                if(string.contains(i + "")) continue;
                queue.add(string + i);
            }
            queue.poll();
        }
    }

    public static void main(String[] args) {

        BFS(5);
    }
}

```

[迷宫 - 蓝桥云课\(lanqiao.cn\)](http://lanqiao.cn)

```

package 搜索;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;

```

```

import java.util.Map;
import java.util.Queue;
import java.util.Scanner;

public class Ty10 {

    // N*N M个传送门
    static int N = 2010;
    // 表示 四个方位
    static int dx[] = {0,0,-1,1};
    static int dy[] = {1,-1,0,0};

    // 是否被访问过
    static boolean visit[][] = new boolean[N][N];

    // 表示传送门之间的关系
    // 压缩二维数组 成为 一维数组 Integer 就是 压缩之后的 坐标
    // List 列表 存的是一个 int[] 数组 每一个 int[] 数组 分别表示 x y
    static Map<Integer, List<int []>> map = new HashMap<>();

    // 最终的答案
    static int ans = 0;

    // n 行 n 列
    static int n;
    // m 表示 传送门的个数
    static int m;

    // add 将 (x1,y1) 和 (x2,y2) 连接起来
    public static void add(int x1, int y1, int x2, int y2) {

        if(!map.containsKey(x1*n + y1))
            map.put(x1*n + y1, new ArrayList<>());

        map.get(x1*n + y1).add(new int[] {x2,y2});
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        for(int i = 0 ; i < n; i++) {
            for(int j = 0; j < n; j++) {
                visit[i][j] = false;
            }
        }

        n = scanner.nextInt();
        m = scanner.nextInt();

        for(int i = 0; i < m; i++) {

            int x1 = scanner.nextInt() - 1;
            int y1 = scanner.nextInt() - 1;

            int x2 = scanner.nextInt() - 1;
            int y2 = scanner.nextInt() - 1;

```

```

        add(x1, y1, x2, y2);
        add(x2, y2, x1, y1);
    }

    Queue<int []> queue = new LinkedList<>();

    // 将 (n-1,n-1) 加入到队列中去
    queue.offer(new int[] {n-1,n-1});

    // 表示 (n-1,n-1) 已经被访问过
    visit[n-1][n-1] = true;

    // 累计层数
    int floor = 0;

    while(queue.isEmpty() == false) {

        // 求出 size()
        int size = queue.size();

        while(size-- > 0) {

            // 取出对头元素
            int []head = queue.poll();

            int x = head[0];
            int y = head[1];

            ans += floor;

            if(map.containsKey(x * n + y)) {

                List<int []> list = map.get(x * n + y);

                for (int[] js : list) {
                    // System.out.println(x * n + y);
                    // System.out.println(visit[js[0]][js[1]]);
                    if(visit[js[0]][js[1]] == false) {
                        queue.offer(js);
                        visit[js[0]][js[1]] = true;
                    }
                }
            }

            for(int j = 0; j < 4; j++) {

                int newx = x + dx[j];
                int newy = y + dy[j];

                if(newx >= 0 && newx < n && newy >= 0 &&
                    newy < n && visit[newx][newy] == false) {

                    // System.out.println("newx = " + newx + "newy = " + newy);
                    queue.offer(new int[] {newx,newy});
                    visit[newx][newy] = true;
                }
            }
        }
    }
}

```



```

//          for (int [] q: queue) {
//              System.out.println(Arrays.toString(q));
//          }
//      }

      floor++;
  }

  System.out.printf("%.2f",ans * 1.0 / (n * n));
}
}

```

如下代码 就是 将当前节点 只通过一步就能到达的顶点加入到队列中去

- 通过传送门到达
- 通过上下左右搜索到达

```

if(map.containsKey(x * n + y)) {

    List<int []> list = map.get(x * n + y);

    for (int[] js : list) {
        //          System.out.println(x * n + y);
        //          System.out.println(visit[js[0]][js[1]]);
        if(visit[js[0]][js[1]] == false) {
            queue.offer(js);
            visit[js[0]][js[1]] = true;
        }
    }
}

// 板子
for(int j = 0; j < 4; j++) {

    int newx = x + dx[j];
    int newy = y + dy[j];

    if(newx >= 0 && newx < n && newy >= 0 &&
        newy < n && visit[newx][newy] == false) {

        //          System.out.println("newx = " + newx + "newy = "
+ newy);
        queue.offer(new int[] {newx,newy});
        visit[newx][newy] = true;
    }
}
}

```

迷宫搜索

BFS有一个自己独特的有点就是自己找到的最终答案一定是最优的

```
package 搜索;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class Ty09迷宫 {

    /*
        0 1 0 0 0 0
        0 0 0 1 0 0
        0 0 1 0 0 1
        1 1 0 0 0 0
        DRRURRDDR
    */

    public static int step = 0;

    public static int map[][] = new int [30][50];

    public static int n = 30;

    static boolean visit[][] = new boolean [30][50];

    // 上下最有有一个优先级
    // 具体根据题目意思
    static int dx[] = {1, 0, 0, -1};
    static int dy[] = {0, -1, 1, 0};

    public static void BFS(int row,int col) {

        for(int i = 0 ; i < row; i++) {
            for(int j = 0 ; j < col; j++) {
                visit[i][j] = false;
            }
        }

        Queue<int []> queue = new LinkedList<>();
        // 具体的路径规则
        Queue<String> queue2 = new LinkedList<String>();

        queue.offer(new int [] {0,0});
        queue2.offer("");

        visit[0][0] = true;

        while(queue.isEmpty()) {

            int []place = queue.element();

            int x = place[0];
```

```

        int y = place[1];

        // 获取根节点
        String path = queue2.element();

        // 对四个方向进行搜索
        for(int i = 0; i < 4; i++) {

            int newx = x + dx[i];
            int newy = y + dy[i];

            if(newx >= 0 && newx < row &&
               newy >= 0 && newy < col &&
               visit[newx][newy] == false &&
               map[newx][newy] == 0) {
                // 记录最终答案
                String tempString = "";
                queue.offer(new int[] {newx,newy});
                visit[newx][newy] = true;
                if(dx[i] == 0 && dy[i] == -1) {
                    tempString = path + "L";
                    queue2.add(path + "L");
                }
                if(dx[i] == 0 && dy[i] == 1) {
                    tempString = path + "R";
                    queue2.add(path + "R");
                }
                if(dx[i] == -1 && dy[i] == 0) {
                    tempString = path + "U";
                    queue2.add(path + "U");
                }
                if(dx[i] == 1 && dy[i] == 0) {
                    tempString = path + "D";
                    queue2.add(path + "D");
                }

                if(newx == row - 1 && newy == col - 1)
                    System.out.println(tempString);
            }
        }

        queue.poll();
        // System.out.println(queue2.element());
        queue2.poll();
    }
}

public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    String []strings = new String[30];

    String []strings1 = new String[2];

    strings1 = scanner.nextLine().split(" ");

    int n = Integer.parseInt(strings1[0]) ;

```

```

        int m = Integer.parseInt(strings1[1]) ;

        System.out.println(n + " " + m);

        for(int i = 0 ; i < n; i++)
            strings[i] = scanner.nextLine();

        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
                map[i][j] = strings[i].charAt(j) - '0';

        BFS(n,m);
    }
}

```

贪心

板子 使用优先级队列

将数据存储优先级队列中，再将数据取出来进行操作

优先队列模板

```

PriorityBlockingQueue<Long> q = new PriorityBlockingQueue<>();

    for(int i = 0 ; i < n; i++) {

        long num = scanner.nextLong();

        q.add(num);
    }

    long ans = 0;

    while(q.size() > 1) {

        long one = q.poll();
        long two = q.poll();

        ans += (one + two);

        q.add(one + two);
    }

```

小明的衣服

题目描述

小明买了 n 件白色的衣服，他觉得所有衣服都是一种颜色太单调，希望对这些衣服进行染色，每次染色时，他会将某种颜色的**所有**衣服寄去染色厂，第 i 件衣服的邮费为 a_i 元，染色厂会按照小明的要求将其中一部分衣服染成同一种任意的颜色，之后将衣服寄给小明，请问小明要将 n 件衣服染成不同颜色的最小代价是多少？

输入描述

第一行为一个整数 n ，表示衣服的数量。

第二行包括 n 个整数 $a_1, a_2 \dots a_n$ 表示第 i 件衣服的邮费为 a_i 元。

$$(1 \leq n \leq 10^5, 1 \leq a_i \leq 10^9)$$

输出描述

输出一个整数表示小明所要花费的最小代价。

将同一个点分解成不同的点 == 讲不同的点合并成一个根节点 == 哈夫曼树的生成

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Arrays;
import java.util.Scanner;
import java.util.concurrent.PriorityBlockingQueue;

public class Main {

    static BufferedReader bReader = new BufferedReader(new
InputStreamReader(System.in));
    public static void main(String[] args) throws Exception {

        int n = Integer.parseInt(bReader.readLine());

        PriorityBlockingQueue<Long> p = new PriorityBlockingQueue<>();

        String []strings = bReader.readLine().split(" ");

        for(int i = 0 ; i < strings.length; i++) {
```

```

        p.add(Long.parseLong(strings[i]));
    }

    long ans = 0;

    while(p.size() > 1) {

        long a = p.poll();
        long b = p.poll();

        ans += (a + b);
        p.add(a + b);
    }
    System.out.println(ans);
}
}

```

二叉树

创建类

```

class BtNode{

    int data;
    BtNode left;
    BtNode right;

    public BtNode() {

    }

    public BtNode(int data, BtNode left, BtNode right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}

```

创建二叉树

```

public static BtNode CreateTree() {

    BtNode root = null;

```

```
int value = scanner.nextInt();

if(value == 0) return null;

else {

    root = new BtNode();

    root.data = value;

    root.left = CreateTree();

    root.right = CreateTree();

}

return root;

}
```

遍历

前序遍历

```
public static void PreOrder(BtNode tree) {

    if(tree == null) return ;

    else {

        System.out.print(tree.data + " ");

        PreOrder(tree.left);

        PreOrder(tree.right);

    }

}
```

中序遍历

```
public static void InOrder(BtNode tree) {

    if(tree == null) return;

    else {

        InOrder(tree.left);

        System.out.print(tree.data + " ");

        InOrder(tree.right);

    }

}
```

后序遍历

```
public static void PostOrder(BtNode tree) {  
  
    if(tree == null) return;  
  
    else {  
        PostOrder(tree.left);  
        PostOrder(tree.right);  
        System.out.print(tree.data + " ");  
    }  
}
```

层次遍历

```
public static void LevelOrder(BtNode tree) {  
  
    Queue<BtNode> queue = new LinkedList<>();  
  
    if(tree == null) return;  
  
    else queue.add(tree);  
  
    while(queue.size() > 0) {  
  
        BtNode tBtNode = queue.poll();  
  
        System.out.print(tBtNode.data + " ");  
  
        if(tBtNode.left != null) queue.add(tBtNode.left);  
  
        if(tBtNode.right != null) queue.add(tBtNode.right);  
    }  
}
```

叶子节点

```
/**  
 * 统计叶子节点的个数  
 */  
  
public static int Leaf(BtNode tree) {  
  
    int leaf = 0;  
  
    if(tree == null) return 0;  
  
    else {  
  
        if(tree.left == null && tree.right == null) return 1;
```



```

        leaf += Leaf(tree.left);

        leaf += Leaf(tree.right);
    }

    return leaf;
}

```

树的深度

```

/*
 * 数的深度
 */
public static int depth(BtNode tree){

    int depth1 = 0;
    int depth2 = 0;

    if(tree == null) return 0;

    else {

        depth1 = depth(tree.left) + 1;

        depth2 = depth(tree.right) + 1;
    }

    return Math.max(depth1, depth2);
}

```

树的重建

前序+中序

```

/*
 * 已知前序和中序
 */
public static BtNode createNewTree
(int pre[],int prestar,int preend, int in[], int instart, int inend) {

    if(instart > inend || prestar > preend) return null;

    BtNode root = new BtNode();

    root.data = pre[prestar];

    int index = -1;

    for(int i = instart; i <= inend; i++) {

        if(in[i] == pre[prestar]) {
            index = i;
            break;
        }
    }
}

```

```

        root.left = createNewTree(pre, prestar + 1, prestar + index - instart, in,
instart, index - 1);
        root.right = createNewTree(pre, prestar + index - instart + 1, preend, in,
index + 1, inend);

        return root;
    }

```

中序+后序

```

public static BtNode createNewTree2
(int post[],int poststar,int postend, int in[], int instart, int inend) {

    if(instart > inend || poststar > postend) return null;

    BtNode root = new BtNode();

    root.data = post[postend];

    int index = -1;

    for(int i = instart; i <= inend; i++) {

        if(in[i] == post[postend]) {
            index = i;
            break;
        }
    }

    root.left = createNewTree2(post, poststar, poststar + index - instart - 1,
in, instart, index - 1);
    root.right = createNewTree2(post, poststar + index - instart, postend - 1,
in, index + 1, inend);

    return root;
}

```

字典树

```
11 public static void main(String[] args) throws IOException {
12     int n=Integer.parseInt(br.readLine());
13     String[] s = br.readLine().split(" ");
14     for (int i = 0; i < n; i++) {
15         a[i] = Integer.parseInt(s[i]);
16         cnt[a[i]]++;
17     }
18     // 获得前缀和数组
19     for (int i = 1; i <= 100000; ++i) {
20         cnt[i] += cnt[i - 1];
21     }
22
23     for (int i = 0; i < n; ++i) {
24         if (cnt[100000] - cnt[a[i]] <= cnt[Math.max(0, a[i]-1)]) {
25             out.print(0 + " ");
26             continue;
27         }
28         // 去二分找到我们应该刷多少题才符合条件
29         int l = a[i] + 1, r = 100000;
30         while (l < r) {
31             int mid = l + r >> 1;
32             if (cnt[100000] - cnt[mid] <= cnt[mid - 1] - 1) r = mid;
33             else l = mid + 1;
34         }
35         out.print((r - a[i]) + " ");
36     }
37     out.flush();
38 }
39 }
```

图论

最小生成树：最小权重

prim

Kruskal

使用的就是并查集操作

```
void init()
{
    for (int i = 0; i < n; i++) parent[i] = i;
}

int find(int x)
{
    if (parent[x] == x)
        return x;
    else
        parent[x] = find(parent[x]);
    return parent[x];
}

void merge(int a, int b)
{
    int r1 = find(a);
    int r2 = find(b);
```

```

    parent[r1] = r2;
}

```

将边的权值按大小进行排序

当所有点都只有一个公共祖先的时候 就结束

```

void kruskal()
{
    int Sumweight = 0;
    int u, v;

    init();

    for (int i = 0; i < m; i++)
    {
        u = Edge[i].u;
        v = Edge[i].v;

        if (find(u) != find(v))
        {
            cout << Edge[i].u << " " << Edge[i].v << " " << Edge[i].weight << endl;
            Sumweight += Edge[i].weight;
            merge(u, v);
        }
    }

    cout << Sumweight << endl;
}

```

```

public class Main {

    class Edge{
        int u;
        int v;
        int weight;
    }

    public static int parent[] = new int [100];

    public static void init() {
        for(int i = 0; i < 100; i++) parent[i] = i;
    }

    public static int find(int x) {
        if(parent[x] == x) return x;
        else parent[x] = find(parent[x]);
        return parent[x];
    }

    public static void merge(int x,int y) {
        int l = find(x);
        int r = find(y);
        parent[l] = r;
    }
}

```

```

    }

    public static Edge edge[] = new Edge[100];

    public static int point = 0;
    public static int m = 0;

    public static void kruskal() {

        int sum = 0;
        int u = 0;
        int v = 0;

        init();

        for(int i = 0; i < m; i++) {

            u = edge[i].u;
            v = edge[i].v;

            // 如果不是 公共的祖先 将两个节点进行合并操作
            if(find(u) != find(v)) {
                sum += edge[i].weight;
                merge(u, v);
            }
        }
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("输入顶点个数");

        point = scanner.nextInt();

        System.out.println("输入边长个数");

        m = scanner.nextInt();

        for(int i = 0; i < m; i++) {
            edge[i].u = scanner.nextInt();
            edge[i].v = scanner.nextInt();
            edge[i].weight = scanner.nextInt();
        }

        // 根据权值 进行排序
        Arrays.sort(edge, new Comparator<Edge>() {
            @Override
            public int compare(Edge o1, Edge o2) {
                // TODO Auto-generated method stub
                return o1.weight - o2.weight;
            }
        });

        kruskal();
    }
}

```

Djakarta

单源点最短路径问题

```
/*
 * @Description: my project
 * @version: 1.0
 * @Author: Typecoh
 * @Date: 2023-04-02 15:01:05
 * @LastEditors: Typecoh
 * @LastEditTime: 2023-04-03 21:04:24
 */

#include <iostream>
#include <vector>
using namespace std;

int Edge[1000][1000];
int n;
int m;
int dist[10000];
int MAX = 999999;

void Dijkstra()
{
    vector<int> U;

    U.push_back(0);

    // 加入初始节点
    for (int i = 0; i < n; i++)
    {
        if (Edge[0][i] != MAX) U.push_back(i);
        dist[i] = Edge[0][i];
    }

    // 看是否 通过 一个中间节点 是 到目标节点距离变短
    for (int i = 0; i < U.size(); i++)
    {
        for (int j = 0; j < n; j++)
        {
            // 如果存在这个节点
            if (dist[j] > dist[U[i]] + Edge[U[i]][j])
            {
                // 新最短距离
                dist[j] = dist[U[i]] + Edge[U[i]][j];
                U.push_back(j);
            }
        }
    }
}
```

Java版本

```
package 冲刺;
```

```

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class test {

    // 顶点个数
    static int n;
    // 变长的条数
    static int m;

    static int star;
    static int end;
    // 最终距离的计算
    static int dist[];
    // 边长的表示
    static int edge[][];
    // 不可达的标志
    static int MAX = 99999999;

    public static void Dijkstra() {

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

        dist = new int [n + 1];

        for(int i = 1; i <= n; i++) {
            if(edge[star][i] != MAX) list.add(i);
            dist[i] = edge[star][i];
        }

        for(int i = 0; i < list.size(); i++) {
            for(int j = 1; j <= n; j++) {
                if(dist[j] > dist[list.get(i)] + edge[list.get(i)][j]) {
                    dist[j] = dist[list.get(i)] + edge[list.get(i)][j];
                    list.add(j);
                }
            }
        }
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        n = scanner.nextInt();

        m = scanner.nextInt();

        star = scanner.nextInt();
        end = scanner.nextInt();

        edge = new int [n + 1][n + 1];

        // 对边长进行初始化操作
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= n; j++) {

```

```

        if(i == j) edge[i][j] = edge[j][i] = 0;
        else edge[i][j] = edge[j][i] = MAX;
    }
}

// 读入变长数据
for(int i = 1; i <= m; i++) {
    int u = scanner.nextInt();
    int v = scanner.nextInt();
    // 无向边
    edge[v][u] = edge[u][v] = scanner.nextInt();
}

Dijkstra();

System.out.println(dist[end]);
}
}

```

Floyd

任意两点之间的最短路径

```

void Floyd()
{
    int i, j, k, dist[100][100];

    // dist 表示 两点之间的距离
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            dist[i][j] = Edge[i][j];

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            cout << dist[i][j] << " ";
        cout << endl;
    }

    // 求任意两点之间的最短路径
    for (int k = 0; k < n; k++)
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (dist[i][j] > dist[i][k] + dist[k][j])
                    dist[i][j] = dist[i][k] + dist[k][j];

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            cout << dist[i][j] << " ";
        cout << endl;
    }
}
}

```


1551: [蓝桥杯2021初赛] 直线

内存限制: 256 MB

时间限制: 1 S

标准输入输出

题目类型: 传统

评测方式: 文本比较

上传者: 外部导入

提交: 4366

通过: 1553

提交

提交记录

统计

讨论版

题目描述

在平面直角坐标系中，两点可以确定一条直线。

如果有多点在一条直线上，那么这些点中任意两点确定的直线是同一条。

给定平面上 2×3 个整点 $\{(x, y) | 0 \leq x < 2, 0 \leq y < 3, x \in \mathbb{Z}, y \in \mathbb{Z}\}$,

即横坐标是0到1 (包含0和1) 之间的整数、纵坐标是0到2 (包含0和2) 之间的整数的点。

这些点一共确定了11条不同的直线。

给定平面上 20×21 个整点 $\{(x, y) | 0 \leq x < 20, 0 \leq y < 21, x \in \mathbb{Z}, y \in \mathbb{Z}\}$,

即横坐标是0到19 (包含0和19) 之间的整数、纵坐标是0到20 (包含0和20) 之间的整数的点。

请问这些点一共确定了多少条不同的直线。

对 K 和 d 在计算的时候主要是 精度损失 最好不要使用除法

```
package 蓝桥杯省赛真题;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map.Entry;
import java.util.Scanner;
import java.util.Set;

public class Ty08直线 {

    public static int gcd(int a, int b) {

        if(b == 0) return a;

        return gcd(b, a % b);
    }

    public static void main(String[] args) {

//        System.out.println(gcd(20,8));

        Scanner scanner = new Scanner(System.in);

        Set<List<Integer>> set = new HashSet<>();

        int vist[][] = new int [20][21];

        int n = 20;

        for(int i = 0 ; i < n; i++) {
            for(int j = 0; j < n + 1; j++) {
                vist[i][j] = 0;
            }
        }
    }
}
```

```

int count = 0;

for(int x1 = 0; x1 < n; x1++) {
    for(int y1 = 0; y1 < n + 1; y1++) {

        vist[x1][y1] = 1;

        for(int x2 = 0; x2 < n; x2++) {
            for(int y2 = 0; y2 < n + 1; y2++) {

                if(vist[x2][y2] == 1) continue;

                if(x1 == x2 && y1 == y2) continue;

                else {

                    if(x1 == x2 || y1 == y2) continue;

                    else {

                        int k = gcd((y2 - y1), (x2 - x1));

                        int up = (y2 - y1) / k;

                        int down = (x2 - x1) / k;

                        int b = (y1 * x2 - y2 * x1) / k;

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

                        list.add(up);
                        list.add(down);
                        list.add(b);

                        set.add(list);
                    }
                }
            }
        }
    }

    System.out.println(set.size() + n + n + 1);
}

}

```

1552: [蓝桥杯2021初赛] 货物摆放

内存限制: 256 MB

时间限制: 1 S

标准输入输出

题目类型: 传统

评测方式: 文本比较

上传者: 外部导入

提交: 3785

通过: 1725

提交

提交记录

统计

讨论版

题目描述

小蓝有一个超大的仓库，可以摆放很多货物。

现在，小蓝有 n 箱货物要摆放在仓库，每箱货物都是规则的正方体。

小蓝规定了长、宽、高三个互相垂直的方向，每箱货物的边都必须严格平行于长、宽、高。

小蓝希望所有的货物最终摆成一个大的立方体。即在长、宽、高的方向上分别堆 L 、 W 、 H 的货物，满足 $n = L \times W \times H$ 。

给定 n ，请问有多少种堆放货物的方案满足要求。

例如，当 $n = 4$ 时，有以下 6 种方案： $1 \times 1 \times 4$ 、 $1 \times 2 \times 2$ 、 $1 \times 4 \times 1$ 、 $2 \times 1 \times 2$ 、 $2 \times 2 \times 1$ 、 $4 \times 1 \times 1$ 。

请问，当 $n = 2021041820210418$ （注意有 16 位数字）时，总共有多少种

方案？

提示：建议使用计算机编程解决问题。

16位数可以使用 Long 存储不需要使用大数

将 n 以内所有的 因子求出来 将这些因子逐一试探

```
package 蓝桥杯省赛真题;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map.Entry;

import org.w3c.dom.ls.LSOutput;

import java.util.Scanner;
import java.util.Set;

public class Ty09货物摆放 {

    public static void main(String[] args) {

        int ans = 0;

        long n = 2021041820210418L;

        List<Long> list = new ArrayList<>();

        for(long i = 1; i <= Math.sqrt(n); i++) {
            if(n % i == 0) {

                if(i * i != n) {
                    list.add(i);
                    list.add(n / i);
                }
                else {
```

```

        list.add(i);
    }
}

int count = 0;

for(long i = 0; i < list.size(); i++) {
    for(long j = 0; j < list.size(); j++) {
        for(long k = 0; k < list.size(); k++) {
            if(list.get((int) i) * list.get((int) j) * list.get((int) k)
== n) {
                count++;
            }
        }
    }
}

System.out.println(count);
}
}

```

蓝桥杯冲刺

将字符串转成字符数组

将字符数组转成字符串

将字符串进行分割处理 需要注意是不是需要进行转义处理 (统一做转义处理)

字符串反转如何处理

字符串和数字之间的转换处理

进制转换！！！！

10 ==》 某个进制

某个进制 ==》 10

九进制正整数 (2022)9(2022)9 转换成十进制等于多少？

大数 BigInteger

快读

读文件操作

素数

欧拉晒

线性筛

算数基本定理

```
// prime factor decomposition
public static void PFD(long n) {

    // 先求解 质因数 在 2 - sqrt(n)
    // 用来标记答案的个数
    int ans = 0;
    for(long i = 2; i <= n / i ;i++) {

        // 如果 n % i == 0 说明 i是 n的展开式中的一项 ==> i 是 n的一个质因子
        if(n % i == 0) {
            ans++;
            // 求这个质因子在n中的个数
            // 求除去这些质因子剩下的那个数的质因子有哪些
            // 注意这里的n变化了 变小了!!! 因此 时间复杂发是 o(n)
            while(n % i == 0) n /= i;
        }
    }

    // 剩下 最后的一项如果不是1
    // 那这个数 本身就是一个质数
    if(n > 1) ans++;

    System.out.println(ans);
}
```

阶乘/末尾0的个数

求5的个数

二分板子

```
int binarySearch(int left, int right) {
    while (left < right) {
        int mid = (left + right) / 2; // 注意防止溢出
        if (check(mid)) // 判断 mid 是否满足查找条件
            left = mid + 1; // 结果落在 [mid+1, right] 区间
        else
            right = mid; // 结果落在 [left, mid] 区间
    }
    return left;
}
```

```
int binarySearch(vector<int> &nums, int target) {
```

```

int left = 0, right = nums.size() - 1;
while (left < right) {
    // 防止溢出, 结果等同于 (left + right + 1)/2
    int mid = left + ((right - left + 1) / 2);
    // 检查 mid, 比较 nums[mid] 和 target
    if (nums[mid] > target)
        // target 落在 [left, mid-1]
        right = mid - 1;
    else
        // target 落在 [mid, right]
        left = mid;
}
return nums[left] == target ? left : -1;
}

```

最少刷题数

```

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

public class Main {

    static int N = 100005;
    static int arr[] = new int [N];
    static int pre[] = new int [N];

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        for(int i = 0 ; i < n; i++) {

            arr[i] = scanner.nextInt();
            pre[arr[i]]++;
        }

        //      for(int i = 0 ; i < n; i++) System.out.println(arr[i]);
        // pre[arr[i]] 表示 arr[i] 相等的有多少

        for(int i = 1; i < 100005; i++) {

            pre[i] = pre[i - 1] + pre[i];
        }

        //      System.out.println(pre[arr[1]]);
        // 小于 arr[i] 就是 pre[arr[i] - 1] 的数量
        // 和小于等于 arr[i] 的就是 pre[arr[i]]
        // 和等于 arr[i] 的就是 pre[arr[i]] - pre[arr[i]-1]

        for(int i = 0 ; i < n; i++) {

            if(pre[100000] - pre[arr[i]] <= pre[Math.max(0, arr[i] - 1)]) {
                System.out.print(0 + " ");
            }
        }
    }
}

```

```

        continue;
    }

    int l = arr[i] + 1;
    int r = 100000;

    while(l < r) {
        int mid = (l + r) / 2;
        if(pre[100000] - pre[mid] <= pre[mid - 1] - 1) {
            r = mid;
        }
        else {
            l = mid + 1;
        }
    }

    System.out.print(r - arr[i] + " ");
}
}
}

```

BFS DFS

```

package 冲刺;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class test {

    public static int dx [] = {0,1,0,0,-1};
    public static int dy [] = {0,0,-1,1,0};

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();
        int m = scanner.nextInt();

        int map[][] = new int [n + 1][m + 1];

        boolean visit[][] = new boolean [n + 1][m + 1];

        String string[] = new String[31];
        for(int i = 1; i <= n; i++) {
            string[i] = scanner.next();
        }

        for(int i = 1; i <= n; i++) {
            for(int j = 0; j < string[i].length(); j++) {
                map[i][j + 1] = string[i].charAt(j) - '0';
                visit[i][j+1] = false;
            }
        }
    }
}

```

```

//      for(int i = 1; i <= n; i++) {
//          for(int j = 1; j <= m; j++) {
//              System.out.print(map[i][j]);
//          }
//          System.out.println();
//      }
//
Queue<int[]> queue = new LinkedList<>();
Queue<String> queue2 = new LinkedList<String>();
queue.offer(new int[]{1,1});
queue2.offer("");
visit[1][1] = true;

while(queue.size() != 0) {

    int []place = queue.element();

    String path = queue2.element();

//      System.out.println(path);

    int x = place[0];
    int y = place[1];

    for(int i = 1; i <= 4; i++) {

        int newx = x + dx[i];
        int newy = y + dy[i];

        if(newx >= 1 && newx <= n &&
            newy >= 1 && newy <= m &&
            visit[newx][newy] == false &&
            map[newx][newy] == 0) {
            String tempString = "";
            queue.offer(new int[] {newx,newy});
            visit[newx][newy] = true;

            if(dx[i] == 0 && dy[i] == -1) {
                tempString = path + "L";
                queue2.add(path + "L");
            }
            if(dx[i] == 0 && dy[i] == 1) {
                tempString = path + "R";
                queue2.add(path + "R");
            }
            if(dx[i] == -1 && dy[i] == 0) {
                tempString = path + "U";
                queue2.add(path + "U");
            }
            if(dx[i] == 1 && dy[i] == 0) {
                tempString = path + "D";
                queue2.add(path + "D");
            }

            if(newx == n && newy == m) {
                System.out.println(tempString);
                break;
            }
        }
    }
}

```



```

        }

        }

        queue.poll();

        queue2.poll();
    }
}

```

排序 对象 collections

```

Map<Integer, Integer> map = new HashMap<>();

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

List<Map.Entry<Integer,Integer>> list2 = new ArrayList<>();

// 对List 使用 Collections
Collections.sort(list,new Comparator<Integer>() {

    @Override
    public int compare(Integer o1, Integer o2) {
        // TODO Auto-generated method stub
        return 0;
    }
});

// 对 Map使用 Collections
Collections.sort(list2, new Comparator<Map.Entry<Integer,Integer>>() {

    @Override
    public int compare(Map.Entry<Integer, Integer> o1,Map.Entry<Integer,
Integer> o2) {
        // TODO Auto-generated method stub
        return 0;
    }
});

```

快速幂

```

while(y > 0) {

    if(y % 2 != 0) ans = ans * x;

    y /= 2;
    x = x * x;
}
System.out.println(ans);

```

并查集基本操作

```
public static void init() {
    for(int i = 0; i < parent.length; i++) parent[i] = i;
}

public static int find(int point) {

    if(parent[point] == point) return point;

    else parent[point] = find(parent[point]);

    return parent[point];
}

public static void merge(int x, int y) {

    int LeftParent = find(x);
    int RightParent = find(y);

    parent[LeftParent] = RightParent;
}
```

去除Set

对单一List去重

```
List<Integer> list = new ArrayList<>();

list.add(1);
list.add(2);
list.add(1);

for (Integer integer : list) System.out.print(integer + " ");

System.out.println();

Set<Integer> set = new HashSet();

// 将所有元素加进去
set.addAll(list);

for (Integer integer : set) {
    System.out.print(integer + " ");
}
```

TreeSet排好序

空格分开 split函数

```

Scanner scanner = new Scanner(System.in);

String string = scanner.nextLine();

// 使用的是 正则表达式 根据空格的位置 将 单词分开
String []strings = string.split(" ");

for (String string2 : strings) {

    System.out.println(string2);
}

```

```

public static boolean visit[] = new boolean[7];

    public static int ans = 0;

    public static int map [][] = {
        {0,1,0,0,0,1,0},
        {1,0,1,0,0,0,1},
        {0,1,0,1,0,0,1},
        {0,0,1,0,1,0,0},
        {0,0,0,1,0,1,1},
        {1,0,0,0,1,0,1},
        {0,1,1,0,1,1,0},
    };

    public static int parent[] = new int [7];

    public static void init() {
        for(int i = 0; i < parent.length; i++) {
            parent[i] = i;
        }
    }

    public static void merge(int x, int y) {

        int LeftParent = find(x);
        int RightParent = find(y);

        parent[LeftParent] = RightParent;
    }

    public static void dfs(int a) {
        if(a==7)
        {
            for(int i=0;i<7;i++) parent[i]=i;
            for(int i=0;i<7;i++)
            {
                for(int j=0;j<7;j++)
                    if(map[i][j]==1&&visit[i]==true&&visit[j]==true)
                        merge(i,j);
            }
        }

        int root_num=0; // 所有亮着的节点的根节点数
    }

```

```

        for(int i=0;i<7;i++)
            if(parent[i]==i&&visit[i]) root_num++;
        if(root_num==1) ans++;
        return;
    }

    System.out.println(a);
    visit[a]= true; // 把这个二极管点亮，看这种情况下的树
    dfs(a+1);
    visit[a]=false; // 把这个二极管熄灭，看这种情况下的树
    dfs(a+1);
}

public static int find(int x) {

    if(parent[x] == x) return x;
    else parent[x] = find(parent[x]);
    return parent[x];
}

public static void main(String[] args) {

    dfs(0);

    System.out.println(ans);

}

```

时间格式设置


```

Scanner scanner = new Scanner(System.in);

long t = scanner.nextLong();

Date data = new Date(t);
// 将时间 设置为 HH:MM:SS
SimpleDateFormat ft = new SimpleDateFormat ("HH:mm:ss");
// 设置时区 为 US模式
ft.setTimeZone(TimeZone.getTimeZone("US"));
// 将时间 格式化 输出
System.out.println(ft.format(data));

```

 中心主题

JS

封装Stack

```

class Stack {
    constructor() {
        this.items = []
    }
}

```

```

}
// 新增元素
add(e1) {
  this.items.push(e1)
}
// 删除栈顶的元素并返回其值
pop() {
  return this.items.pop()
}
// 返回栈顶的元素
peek() {
  return this.items[this.items.length - 1]
}
// 清空栈
clear() {
  this.items = []
}
// 栈的大小
size() {
  return this.items.length
}
// 栈是否为空
isEmpty() {
  return this.items.length === 0
}
}

```

优先队列封装

```

class PriorityQueue {
  constructor(compare) {
    if (typeof compare !== 'function') {
      throw new Error('compare function required!')
    }

    this.data = []
    this.compare = compare
  }
  //二分查找 寻找插入位置
  search(target) {
    let low = 0, high = this.data.length
    while (low < high) {
      let mid = low + ((high - low) >> 1)
      if (this.compare(this.data[mid], target) > 0) {
        high = mid
      }
      else {
        low = mid + 1
      }
    }
    return low;
  }
  //添加
  add(elem) {
    let index = this.search(elem)

```

```
        this.data.splice(index, 0, elem)
        return this.data.length
    }
    //取出最优元素
    poll() {
        return this.data.pop()
    }
    //查看最优元素
    peek() {
        return this.data[this.data.length - 1];
    }
}
```

重写比较器

```
var openTable = new PriorityQueue((B, A) => ((A.value + A.depth) - (B.value + B.depth)))
```