第十届蓝桥杯省赛 B 组

方一舟

2021 年 3 月 17 日

西安财经大学信息学院

组队

A 组队

作为篮球队教练，你需要从以下名单中选出 1 号位至 5 号位各一名球员，组成球队的首发阵容。每位球员担任 1 号位至 5 号位时的评分如下表所示。请你计算首发阵容 1 号位至 5 号位的评分之和最大可能是多少？

编号	1 号位	2 号位	3 号位	4 号位	5 号位
1	97	90	0	0	0
2	92	85	96	0	0
3	0	0	0	0	93
4	0	0	0	80	86
5	89	83	97	0	0
6	82	86	0	0	0
7	0	0	0	87	90
8	0	97	96	0	0
9	0	0	89	0	0
10	95	99	0	0	0

编号	1 号位	2 号位	3 号位	4 号位	5 号位
11	0	0	96	97	0
12	0	0	0	93	98
13	94	91	0	0	0
14	0	83	87	0	0
15	0	0	98	97	98
16	0	0	0	93	86
17	98	83	99	98	81
18	93	87	92	96	98
19	0	0	0	89	92
20	0	99	96	95	81

```java
package Lanqiao.Provincial10th;

import java.util.List;

public class A_Group_Java_C {
  // 深度优先搜索dfs(回溯)

  private int res = 0; // 全局变量res记录最优组队的结果
  private boolean[] vis; //
      vis数组标记第i个球员是否被选为首发(每个队员只能被选择至多一次，所以被选择后将其
  private int m, n; // 记录数据的行数m = 20，列数n = 5
```

```
12  /**
13   * @param team 传入的二维数组 20×5 的数据(List<List<Integer>> team
         可以看作C中的 int[][] team)
14   * @return 返回评分之和的最大结果
15   */
16  public int group(List < List < Integer >> team) {
17    m = team.size(); // playerNum = 20 (数组下标为0 <= i <= 19)
18    n = team.get(0).size(); // quotaNum = 5 (数组下标为0 <= j <= 4)
19    vis = new boolean[m]; // m = 20组默认初始化为false
20    dfs(team, 0, 0); // 进行深度优先搜索(初始从第0 + 1号位开始, 初始评分为0)
21    return res;
22  }
```

```
24  /**
25   * 对于dfs类题目必须熟练运用模板
26   * @param team 传入的二维数组 20×5 的数据
27   * @param index 遍历到第 index + 1号位(总共有5号位且index从0开始)
28   * @param sum 遍历到第 index + 1号球员时的评分之和是多少
29   */
30  private void dfs(List < List < Integer >> team, int index, int sum) {
```

```
31    if (index == n) { // dfs函数的递归出口 index = 5时退出 (index =
          0..4已经遍历结束后)
32      if (sum > res) { // 如果当前的dfs函数遍历sum结果大于res
33        res = sum; // 更新当前的最大值res
34      }
35      return; // index = 5时当前的dfs就结束了, 直接退出
36    }
37    for (int i = 0; i < m; i++) { // 对第1到m=20号球员进行遍历
38      if (vis[i]) continue; // 回溯剪枝, vis[i] ==
            true的时候终止此次循环(即忽略i的情况, 因为此时队员已被选择)
39      int value = team.get(i).get(index); // 得到第 i+1 号球员如果在 index+1
            号位时
40      vis[i] = true; // 将 i+1 号球员的vis置为true, 代表第 i+1
            号球员已经被选择了
41      dfs(team, index + 1, sum + value); // dfs递归继续考虑index +
            2号位的情况
42      // 此时的评分之和要加上当前球员的评分value
43      vis[i] = false; // 回溯模板中需要将引用变量复位
```

```
48    public static void main(String[] args) {
49      List < List < Integer >> team =
            GetText.getText("D:\\Java\\Intellij\\Mycodes\\src\\Lanqiao\\Provincial10th
50      System.out.println(new A_Group_Java_C().group(team));
51    }
52  }
```

不同的子串

B 不同的子串

一个字符串的非空子串是指字符串中长度至少为 1 的连续的一段字符组成的串。例如，字符串 aaab 有非空子串 a, b, aa, ab, aaa, aab, aaab，一共 7 个。注意在计算时，只算本质不同的串的个数。请问，字符串 0100110001010001 有多少个不同的非空子串？

```java
package Lanqiao.Provincial10th;

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

public class B_DifferSubString_Java {

  private int count = 0; // 记录子串数目
  private final List < String > subStrings = new ArrayList < > (); //
      记录所有子串
```

```java
/**
 * @param s 给定子串s判断其有多少个不同空字串
 * @return 不同非空子串的个数
 */
private int differSubString(String s) {
  int len = s.length();
  for (int i = 0; i < len; i++) {
    for (int j = i; j < len; j++) {
      String sub = s.substring(i, j + 1); // substring方法获得下标为[i,
          j]区间的子串
      if (!subStrings.contains(sub)) { // 子串sub不在集合中就将其加入集合中
        count++;
        subStrings.add(sub);
      }
    }
  }
  return count;
}
```

```
29    public static void main(String[] args) {
30      String s = "0100110001010001";
31      System.out.println(new B_DifferSubString_Java().differSubString(s));
32    }
33  }
```

年号字串

B 年号字串

小明用字母 A 对应数字 1，B 对应 2，以此类推，用 Z 对应 26。对于 27 以上的数字，小明用两位或更长位的字符串来对应，例如 AA 对应 27，AB 对应 28，AZ 对应 52，LQ 对应 329。

请问 2019 对应的字符串是什么？

```java
package Lanqiao.Provincial10th;

public class B_YearString_C {

  /**
   * 本质上为求解10进制转为26进制的问题
   * @param year 年份year用于转换为26进制
   * @return year对应的26进制数
   */
  private String yearString(int year) {
```

```
11    char[] digits = new char[27]; //
          digits记录26进制所对应的26个字母A,B,...,Z
          (10进制对应十个符号0,1,...,9)
12    for (int i = 1; i <= 26; i++) {
13      digits[i] = (char)('A' + i - 1);
14    }
15    // digits[1~26] = {'A', 'B', 'C', ... , 'Y', 'Z'};
16    StringBuilder res = new StringBuilder(); //
          类似于C的String, 因为Java不定长String用StringBuilder
17    while (year != 0) {
18      res.append(digits[year % 26]);
19      year /= 26; // year为整数, 计算结果强制类型转换结果为下取整
20    }
21    // 此时 res = QYB
          因为StringBuilder的append是在字符串位追加, 结果为正常顺序的反向
22    return res.reverse().toString(); // 结果为res.reverse() == BYQ
23  }
```

```
24    // 2019 % 26 = 17 对应字母为Q
25    // 2019 / 26 = 77
26    // 77 % 26 = 25 对应字母为Y
27    // 77 / 26 = 2
28    // 2 % 26 = 2 对应字母为B
29    // 2 / 26 = 0 退出循环
30
31    public static void main(String[] args) {
32      int year = 2019;
33      System.out.println(new B_YearString_C().yearString(year));
34      // res = BYQ
35    }
36
37  }
```

数列求值

C 数列求值

给定数列 $1, 1, 1, 3, 5, 9, 17, \ldots$ 从第 4 项开始，每项都是前 3 项的和。求第 ‘20190324‘ 项的最后 ‘4‘ 位数字。

```java
package Lanqiao.Provincial10th;

public class C_SeriesValue_Java_C {

  /**
   * 给定项数index求出特定数列, 结果保留四位整数
   * f(n) = f(n - 1) + f(n - 2) + f(n - 3), n >= 4
   * f(1) = f(2) = f(3) = 1
   * 1, 1, 1, 3, 5, 9, 17, ..., ...
   * @param index 项数index
   * @return f(index) % 10000
   */
  private int getValue(int index) {
```

```
14    // 迭代实现，递归的空间消耗太大无法实现
15    if (index <= 3) { // f(1) = f(2) = f(3) = 1
16      return 1;
17    }
18    index -= 3; // 从第4项开始，将其视为第1项开始计算
19    int i1 = 1, i2 = 1, i3 = 1; // i1，i2，i3分别为下个元素的前3，2，1个元素
20    while (index-- != 0) {
21      //   3   5   9
22      int sum = (i1 + i2 + i3) % 10000; // sum = 3+5+9 = 17
23      i1 = i2; //   5
24      i2 = i3; //   9
25      i3 = sum; //   17
26    }
27    return i3;
28  }
```

```
30   public static void main(String[] args) {
31     int index = 20190324;
32     System.out.println(new C_SeriesValue_Java_C().getValue(index));
33   }
34
35 }
```

数的分解

D 数的分解

把 2019 分解成 3 个各不相同的正整数之和，并且要求每个正整数都不包含数字 2 和 4，一共有多少种不同的分解方法？注意交换 3 个整数的顺序被视为同一种方法，例如 1000+1001+18 和 1001+1000+18 被视为同一种。

```java
package Lanqiao.Provincial10th;

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

public class D_NumDecompose_Java_C {

  // 记录不含数字2和4的数字
  List < Integer > nums = new ArrayList < > ();

  /**
   * 记录num是否不含数字2和数字4
   * @param num
   */
  private void notContain2And4(int num) {
```

```
16    int num0 = num; // 存取副本，若满足条件则加入集合nums
17    while (num != 0) { // 对num的每一位进行判断其是否包含数字2,4
18      int digit = num % 10; // 取num的个位数
19      if (digit == 2 || digit == 4) {
20        // 如果个位数为2或4，直接退出函数
21        return;
22      }
23      num /= 10; // 如num = 198 -> num = 19 相当于将十位9变为个位
24      // (本质是循环取num的个位进行判断)
25    }
26    // 若while遍历完毕仍未退出函数，则其不含数字2,4，将其副本加入集合nums
27    nums.add(num0);
28  }
```

```
30  /**
31   * 计算数num不包含数字2和数字4的 三个 数所有和的组合
32   * @param num
33   * @return
34   */
35  private int decompose(int num) {
```

```
36    for (int i = 1; i <= num; i++) {
37      // 将所有不大于num的所有不包含2和4的元素加入集合
38      notContain2And4(i);
39    }
40    int count = 0; // 记录结果
41    int x, y, z; // 记录三个因子
42    // 对集合nums进行x, y, z的三层遍历
43    first: for (int i = 0; i < nums.size(); i++) {
44      x = nums.get(i); // 遍历的为集合nums的下标,
            所以x的值为get(i), 相当于C语言中取数组下标x = nums[i]
45      second: for (int j = i + 1; j < nums.size(); j++) {
46        y = nums.get(j);
47        // 在第二层y的遍历中如果此时的x + y >= num的话
48        // 则必不可能存在 x + y + z == num (因为还没有加z(z > 0))
49        // 此时continue first (continue +
              label为Java的特殊语法, 跳转特定层的for)
50        // 正常的continue只能结束当前即second层的当前循环
51        // C语言实现此类跳转需要加设一个flag来记录是否满足条件
```

```
52        if (x + y >= num) continue first;
53        for (int k = j + 1; k < nums.size(); k++) {
54          z = nums.get(k);
55          // 同上
56          if (x + y + z > num) continue second;
57          if (x + y + z == num) { // 满足条件count++
58            count++;
59          }
60        }
61      }
62    }
63    return count;
64  }
65
66  public static void main(String[] args) {
67    int num = 2019;
68    System.out.println(new D_NumDecompose_Java_C().decompose(num));
69  }
```

迷宫

E 迷宫

图给出了一个迷宫的平面图，其中标记为 1 的为障碍，标记为 0 的为可以通行的地方。

迷宫的入口为左上角，出口为右下角，在迷宫中，只能从一个位置走到这个它的上、下、左、右四个方向之一。对于上面的迷宫，从入口开始，可以按 DRRURRDDDR 的顺序通过迷宫，一共 10 步。其中 D、U、L、R 分别表示向下、向上、向左、向右走。

对于下面这个更复杂的迷宫（30 行 50 列），请找出一种通过迷宫的方式，其使用的步数最少，在步数最少的前提下，请找出字典序最小的一个作为答案。请注意在字典序中 D<L<R<U。（如果你把以下文字复制到文本文件中，请务必检查复制的内容是否与文档中的一致。在试题目录下有一个文件 maze.txt，内容与下面的文本相同）

```
010000
000100
001001
110000
```

```java
package Lanqiao.Provincial10th;

import java.io.*;
import java.util.LinkedList;
import java.util.Queue;

public class E_Maze_Java_C {
  // 广度优先搜索bfs

  /**
   * node类记录迷宫每个坐标的信息
   */
  static class node {
    int x; // 迷宫的x坐标
    int y; // 迷宫的y坐标
    int len; // 到(x, y)时已走步长
    StringBuilder path; // 记录走到(x, y)时其之前的路径
```

```
18    node(int x, int y, int len, StringBuilder path) {
19      // 构造函数
20      this.x = x;
21      this.y = y;
22      this.len = len;
23      this.path = path;
24    }
25  }
26  char[][] maze; // 全局变量maze记录迷宫信息
27  boolean[][] vis; // 记录迷宫的结点(x, y)是否被访问
28  int m, n; // 迷宫的长m和宽n
```

```
30  /**
31   * @param maze char[][]迷宫数组
32   */
33  private void getPath(char[][] maze) {
34    // 将传入参数赋为全局变量(避免函数调用时的复杂传参)
35    this.maze = maze;
36    this.m = maze.length;
37    this.n = maze[0].length;
38    this.vis = new boolean[m][n];
39    bfs();
40  }
```

```
42  private boolean satisfy(int x, int y) {
43    // 判断当前结点是否可以遍历
44    // 横坐标 0 <= x < m, 纵坐标 0 <= y < n
45    // !vis[x][y] 等价于 vis[x][y] == false 即坐标(x, y)未被访问过
46    // maze[x][y] != '1' 等价于 maze[x][y] == '0' 即坐标(x, y)可以通行
47    // 满足上述所有条件就证明当前结点可以满足通行
48    return x >= 0 && x < m && y >= 0 && y < n && !vis[x][y] && maze[x][y]
          != '1';
49  }
```

```
51  /**
52   * bfs 广度优先遍历，遍历迷宫找到最短路
53   * 因为此题需要求最短路径，所以不能使用深度优先遍历
54   * 其中遍历路径的优先级为 DLRU
55   * 对于bfs类题目必须熟练运用模板
56   */
57  private void bfs() {
```

```
58    Queue < node > queue = new LinkedList < > (); //
          队列Queue, 数据元素为node类
59    queue.offer(new node(0, 0, 0, new StringBuilder()));
60    // 从结点(0, 0)开始, 所以将其入队类, node(0, 0, 0, "")
61    // 对于第一个结点其坐标为(0, 0), 已走步长为0, 已走路径为""(空字符串)
62    vis[0][0] = true; // 且第一个结点已经被访问
63    while (!queue.isEmpty()) {
64      // 此处开始套bfs模板
65      node curr = queue.poll(); // 获取当前结点curr(第一次循环即为node(0, 0,
            0, ""))
66      if (curr.x == m - 1 && curr.y == n - 1) { // 此处为退出条件
67        // 当当前结点坐标(x, y)为m - 1, n - 1时, 即迷宫最右下角时
68        System.out.println(curr.path); // 输出当前结点记录的路径path并结束程序
69        return;
70      }
```

特别数的和

F 特别数的和

小明对数位中含有 2、0、1、9 的数字很感兴趣(不包括前导 0)，在 1 到 40 中这样的数包括 1、2、9、10 至 32、39 和 40，共 28 个，他们的和是 574。请问，在 1 到 n 中，所有这样的数的和是多少？

输入一行包含两个整数 n。

输出一行，包含一个整数，表示满足条件的数的和。

```
1 40
```

```
1 574
```

```java
package Lanqiao.Provincial10th;

public class F_SpecialNum_Java_C {

  /**
   * 求小于n的所有特殊数的和
   * @param n
   * @return  n的特殊数的和
   */
  private int getSpecialNum(int n) {
    int sum = 0; // 求和结果
    for (int i = 1; i <= n; i++) {
      if (isSpecial(i)) { // i是特殊数则求和
        sum += i;
      }
    }
    return sum;
  }
```

```java
/**
 * 判断是否为特殊数
 * @param n
 * @return 包含2或0或1或9 return true
 */
private boolean isSpecial(int n) {
  while (n != 0) {
    int digit = n % 10;
    if (digit == 2 || digit == 0 || digit == 1 || digit == 9) {
      return true;
    }
    n /= 10;
  }
  return false;
}
```

```
36    public static void main(String[] args) {
37      int n = 40;
38      System.out.println("求和为: " + new
            F_SpecialNum_Java_C().getSpecialNum(n));
39    }
40
41  }
```

外卖店优先级

G 外卖店优先级

“饱了么”外卖系统中维护着 N 家外卖店，编号 1 N。每家外卖店都有一个优先级，初始时 (0 时刻) 优先级都为 0。每经过 1 个时间单位，如果外卖店没有订单，则优先级会减少 1，最低减到 0；而如果外卖店有订单，则优先级不减反加，每有一单优先级加 2。如果某家外卖店某时刻优先级大于 5，则会被系统加入优先缓存中；如果优先级小于等于 3，则会被清除出优先缓存。给定 T 时刻以内的 M 条订单信息，请你计算 T 时刻时有多少外卖店在优先缓存中。

第一行包含 3 个整数 N、M 和 T。以下 M 行每行包含两个整数 ts 和 id，表示 ts 时刻编号 id 的外卖店收到一个订单。输出一个整数代表答案。

```
2 6 6
1 1
5 2
3 1
6 2
2 1
6 2
```

```
1
```

```java
package Lanqiao.Provincial10th;

import java.util.ArrayList;
import java.util.Scanner;

public class G_TakeOut_Java {

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int res = 0;
    int N = sc.nextInt();
    int M = sc.nextInt();
    int T = sc.nextInt();
    int[] level = new int[N + 1]; //存放优先级
    boolean[] judge = new boolean[N + 1]; //存放是否在缓存队列,初始值为false
    ArrayList < Integer > [] array = new ArrayList[T + 1];
        //存放每个时刻对应的店铺
```

```
17    for (int i = 0; i <= T; i++) {
18      array[i] = new ArrayList < > ();
19    }
20    for (int i = 1; i <= M; i++) {
21      int ts = sc.nextInt();
22      int id = sc.nextInt();
23      array[ts].add(id);
24    }
```

```
25    for (int i = 1; i <= T; i++) { //枚举array
26      for (int j = 1; j <= N; j++) { //枚举店铺
27        int flag = 0;
28        Object k = j;
29        while (array[i].contains(k)) {
              //使用while是因为存在一个时刻一个店铺存在多个订单
30          flag = 1;
31          array[i].remove(k);
32          level[j] += 2;
33          if (level[j] > 5) {
34            judge[j] = true;
35            res++;
36          }
37        }
```

```
38        if (flag == 0) { //此时刻无店铺j的订单
39          level[j] = level[j] == 0 ? 0 : level[j] - 1;
40          if (judge[j]) { //原来在缓存队列
41            if (level[j] <= 3) { //优先级下降
42              judge[j] = false;
43              res--;
44            }
45          }
46        }
47      }
48    }
49    System.out.println(res);
50  }
```

完全二叉树的权值

G 完全二叉树的权值

给定一棵包含 N 个节点的完全二叉树，树上每个节点都有一个权值，按从上到下、从左到右的顺序依次是 $A_1, A_2, \ldots A_N$ ，
现在小明要把相同深度的节点的权值加在一起，他想知道哪个深度的节点权值之和最大？如果有多个深度的权值和同为最大，请你输出其中最小的深度。
注：根的深度是 1。
第一行包含一个整数 N。第二行包含 N 个整数 $A_1, A_2, \ldots, A_N$ 。
输出一个整数代表答案。

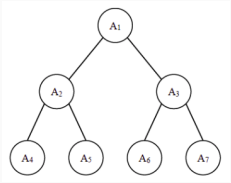

图 1: 一颗完全二叉树

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

```
2
```

```java
package Lanqiao.Provincial10th;

import java.util.LinkedList;
import java.util.Queue;

public class G_CompleteBinaryTree_C {

  private int getMaxFloorWeight(int[] tree, int len) {
    Queue < Integer > queue = new LinkedList < > ();
    queue.offer(0); // bfs 第一个结点入队

    int maxWeight = tree[0]; // 初始最大权值为根节点权值
    int maxHeight = 1; // 初始最大高度为根节点高度

    int currWeight = 0;
    int currHeight = 1; // 根节点从1层开始

    int size = 1; // 队列的初始大小为1
```

```
19    while (!queue.isEmpty()) { // bfs模板
20      //<editor-fold desc="对每层的结点进行计算，得到当前权值">
21      currHeight += 1; // 每层都为上一层数+1
22      while (size-- != 0) { //
            size的大小即为当前queue的大小，此时queue所包含元素必为同一层元素
23        int node = queue.remove(); // 求得当前结点坐标
24        int left = node * 2 + 1, right = node * 2 + 2; //
              对于顺序存储的树，左孩子和右孩子坐标
25        if (left < len) { // 左孩子存在
26          queue.offer(left); // 左孩子入队
27          currWeight += tree[left]; // 当前层的权值累加
28        }
29        if (right < len) { // 右孩子存在
30          queue.offer(right);
31          currWeight += tree[right];
32        }
33      }
```

```
34      //</editor-fold>
35      if (currWeight > maxWeight) { // 当前层权值大于以往权值，进行记录
36        maxWeight = currWeight; // 更改当前最大权值
37        maxHeight = currHeight; // 记录最大权值层数
38      }
39      currWeight = 0; // 下层开始时currWeight应为0
40      size = queue.size(); // 记录队列中结点个数
41    }
42    return maxHeight;
43  }
```

```
45   public static void main(String[] args) {
46     int[] tree = new int[] {
47       1, 6, 5, 4, 3, 2, 1
48     };
49     int len = 7;
50     //<editor-fold desc="get initiate data">
51     //       Scanner sc = new Scanner(System.in);
52     //       int len = sc.nextInt();
53     //       int[] tree = new int[len];
54     //       for (int i = 0; i < len; i++) {
55     //           tree[i] = sc.nextInt();
56     //       }
57     //</editor-fold>
58     System.out.println(new G_CompleteBinaryTree_C().getMaxFloorWeight(tree,
           len));
59   }
60
61 }
```

人物相关性分析

H 人物相关性分析

小明正在分析一本小说中的人物相关性。他想知道在小说中 Alice 和 Bob 有多少次同时出现。更准确的说，小明定义 Alice 和 Bob “同时出现” 的意思是：在小说文本中 Alice 和 Bob 之间不超过 K 个字符。例如以下文本：This is a story about Alice and Bob. Alice wants to send a private message to Bob. 假设 $K = 20$，则 Alice 和 Bob 同时出现了 2 次，分别是” Alice and Bob” 和” Bob. Alice”。前者 Alice 和 Bob 之间有 5 个字符，后者有 2 个字符。

H 人物相关性分析

注意:
Alice 和 Bob 是大小写敏感的，alice 或 bob 等并不计算在内。
Alice 和 Bob 应为单独的单词，前后可以有标点符号和空格，但是不能有字母。例如 Bobbi 并不算出现了 Bob。
第一行包含一个整数 K。第二行包含一行字符串，只包含大小写字母、标点符号和空格。长度不超过 1000000。　　输出一个整数，表示 Alice 和 Bob 同时出现的次数。

```
20
This is a story about
    Alice and Bob.
    Alice wants to
    send a private
    message to Bob.
```

```
2
```

```java
package Lanqiao.Provincial10th;

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

public class H_CharacterRelativity_Java {

  private int relativity(int k, String str) {
    int len = str.length();
    int i = 0, j = 0;
    List < Integer > aliceIndexes = new ArrayList < > (); //
        记录Alice串出现的所有的首位置
    List < Integer > bobIndexes = new ArrayList < > (); //
        记录Bob串出现的所有首位置
```

```
13    while (i < len) {
14      //
            indexOf方法为字符串匹配算法，从下标i开始在str中找到字符串的下标位置(需要熟练
15      int index = str.indexOf("Alice", i); // C/C++中为string库中的find方法
16      if (index == -1) { //
            indexOf算法若无匹配时，返回-1，此时该串不存在原串中
17        break;
18      }
19      aliceIndexes.add(index); // 将当前下标加入集合
20      i = index + 1; // 使下次的搜索位置从下一个字符开始
21
22      // 如果对于字符串AliceAliceBobAlice开始搜索"Alice":
23      // 第一次while循环从i = 0开始搜索，得到下标0(第一次出现在0)加入集合，
24      // 第二次while循环从i = 1开始搜索，得到下标5(第二次出现在5)加入集合，
25      // 第三次while循环从i = 6开始搜索，得到下标13(第三次出现在13)加入集合
26    }
```

```
26    }
27    while (j < len) {
28      // Bob同理
29      int index = str.indexOf("Bob", j);
30      if (index == -1) {
31        break;
32      }
33      bobIndexes.add(index);
34      j = index + 1;
35    }
36    int count = 0;
```

```
37    // 遍历aliceIndex和bobIndex
38    for (int aliceIndex: aliceIndexes) {
39      for (int bobIndex: bobIndexes) {
40        if (aliceIndex < bobIndex && aliceIndex + 5 + k < bobIndex) break;
41        if (bobIndex < aliceIndex && bobIndex + 3 + k >= aliceIndex) {
42          // 如果bobIndex < aliceIndex(前提)时, 满足bobIndex + 3 + k >=
                aliceIndex(判断条件)
43          // 计数器+1
44          count++;
45        }
46        if (aliceIndex < bobIndex && aliceIndex + 5 + k >= bobIndex) {
47          // 如果aliceIndex < bobIndex(前提)时, 满足aliceIndex + 5 + k >=
                bobIndex(判断条件)
48          // 计数器+1
49          count++;
50        }
51      }
52    }
```

```
56    public static void main(String[] args) {
57      int k = 20;
58      String str = "This is a story about Alice and Bob. Alice wants to send
            a private message to Bob.";
59      System.out.println(new H_CharacterRelativity_Java().relativity(k, str));
60    }
61
62  }
```

等差数列

H 等差数列

数学老师给小明出了一道等差数列求和的题目。但是粗心的小明忘记了一部分的数列，只记得其中 N 个整数。现在给出这 N 个整数，小明想知道包含这 N 个整数的最短的等差数列有几项?

输入的第一行包含一个整数 N。第二行包含 N 个整数 $A_1, A_2, \ldots, A_n$。(注意 $A_1 \sim A_n$ 并不一定是按等差数列中的顺序给出)

输出一个整数表示答案。

```
5
2 6 4 10 20
```

```
10
```

包含 2、6、4、10、20 的最短的等差数列是 2、4、6、8、10、12、14、16、18、20。

```java
package Lanqiao.Provincial10th;

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

public class H_ArithmeticalSequence_C {

  /*
     5
     2 6 4 10 20
  */
```

```
13  public static void main(String[] args) {
14    Scanner sc = new Scanner(System.in);
15    // 输入整数N和数组nums
16    int N = sc.nextInt();
17    int[] nums = new int[N];
18    for (int i = 0; i < N; i++) {
19      nums[i] = sc.nextInt();
20    }
21    // 调用排序算法(算法比赛直接调用接口即可)
22    Arrays.sort(nums); // 升序
23    // getDeltaFactor方法求相邻元素差的最大公因子
24    // 例: 2, 6, 12, 差为(4, 6) 最大公因子为2
25    // 例: 2, 4, 7, 差为(2, 3) 最大公因子为1
26    // 例: 2, 4, 6, 10, 20, 差为(2, 4, 10) 最大公因子为2
27    int deltaFactor = new H_ArithmeticalSequence_C().getDeltaFactor(nums);
28    // 等差数列公式 an = a1 + (n - 1) * d
29    // 元素个数 n = (an - a1) / d + 1
30    int count = (nums[N - 1] - nums[0]) / deltaFactor + 1;
```

```
31      System.out.println(count);
32  }
33
34  /**
35   * @param nums 数组nums记录所有的元素
36   * @return 数组相邻元素的差的最大共因子
37   */
38  private int getDeltaFactor(int[] nums) {
```

```
39    TreeSet < Integer > deltaSet = new TreeSet < > (); //
          TreeSet存储数列元素的差
40    // (set的特点是元素不重复)
          TreeSet为排序集合，每次添加元素会自动维持顺序插入新元素
41
42    for (int i = 1; i < nums.length; i++) { // 找到任意相邻元素的间隔delta
43      int currDelta = nums[i] - nums[i - 1];
44      deltaSet.add(currDelta); //
            所有相邻元素的差添加进deltaSet，若重复Set会自动保留一个
45    }
46    int minDelta = deltaSet.first(); // TreeSet特性，第一个元素为集合中最小值
47    // C++的类似接口如果不了解，也可以直接用Set后，再单独遍历一次找最小值
```

```
49      outer: while (minDelta > 1) { //
50        for (int delta: deltaSet) { // for-each对delta集合里面的元素进行遍历
51          if (delta % minDelta != 0) {
52            // 若minDelta不是任何一个元素delta的因子，则就不是公因子
53            minDelta--; // minDelta-- 后继续外层循环继续搜寻minDelta—1是否满足
54            continue outer;
55          }
56        }
57        //
              如果当前的minDelta顺利完成上述for循环，则当前minDelta为所有差的最大公因子
58        return minDelta;
59      }
60      // 上述均不满足，则最大公因子就为1
61      return 1;
62    }
63
64  }
```

后缀表达式

给定 N 个加号、M 个减号以及 $N+M+1$ 个整数 $A_1, A_2, \ldots, A_{N+M+1}$，小明想知道在所有由这 N 个加号、M 个减号以及 $N+M+1$ 个整数凑出的合法的后缀表达式中，结果最大的是哪一个？请你输出这个最大的结果。例如使用 1 2 3 + -，则“2 3 + 1 -”这个后缀表达式结果是 4，是最大的。

第一行包含两个整数 N 和 M。第二行包含 $N+M+1$ 个整数 $A_1, A_2, \ldots, A_{N+M+1}$。

输出一个整数表示答案。

```
1 1
1 2 3
```

```
4
```

```java
package Lanqiao.Provincial10th;

import java.util.*;

public class I_PostFixExpression_Java_C {

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int N = sc.nextInt(), M = sc.nextInt();
    long absSum = 0, noSignSum = 0;
    int posNum = 0, negNum = 0;
    int minPos = Integer.MAX_VALUE, maxNeg = Integer.MIN_VALUE;
```

```
14    for (int i = 0; i < N + M + 1; i++) {
15      int val = sc.nextInt();
16      noSignSum += val;
17      absSum += Math.abs(val);
18      if (val >= 0) {
19        posNum += 1;
20        minPos = Math.min(val, minPos);
21      } else {
22        negNum += 1;
23        maxNeg = Math.max(val, maxNeg);
24      }
25    }
```

```
27      if (M == 0) {
28        System.out.println(noSignSum);
29      } else if (posNum == 0 && M < negNum) {
30        System.out.println(absSum + 2 L * maxNeg);
31      } else if (negNum == 0 && M > negNum) {
32        System.out.println(absSum - 2 L * minPos);
33      } else {
34        System.out.println(absSum);
35      }
36    }
37
38  }
```

```
/*
   (1)没有负号运算符: 此时将所有数都加了,例如:
      5+ 0-
      1 2 3 -4 -5 -6
      -> 1 + 2 + 3 + (-4) + (-5) + (-6)
      -> -9
```

(2)一旦负号存在，负号位置就会对运算结果产生影响，那么：

1.在无正数并且负号运算符不足参与负数运算的情况下，

就有一个负数会单独出来和一个其它数求出来的一个绝对值和整体做减运算，负数为被减数

负数取最大值时值最大.

也就是所求的值等于所有绝对值的值和除去这个负数两次的值。

例如：

```
2+ 3-
-1 -2 -3 -4 -5 -6
-> (-1) - ((-2) + (-3)) - ((-4) + (-5)) - (-6)
-> 19
```

```
55      2.在无负数并且负号运算符大于0，所有正整数参与运算的情况下，
56      就有一个正数会单独出来和一个其它数求出来的一个绝对值和整体做减运算，正数为被减数
57      正数取最小值时值最大.
            也就是所求的值等于所有绝对值的值和除去这个正数两次的值。
58      例如：
59      2+ 3-
60      1 2 3 4 5 6
61      ->  6 + 5 + 4 - (1 - 2 - 3)
62      ->  19
63      3.
            正数和负数都在的情况下，并且负号运算存在，只要存在一个负号，就可以通过加括号
64  */
```

完整题解

References

 陈伯硕. 蓝桥杯 *2019* 省赛 *B* 组.
https://github.com/chenboshuo/programming_practice/tree/master/lanqiaobei/provincial_level/2019B. 2020-08-12.