

CSP2020
野生Solution

Outline

- Junior
  - power
  - live
  - expr
  - number
- Senior
  - julian
  - Zoo
  - call
  - snakes

Junior - 优秀的拆分(power)

二进制签到

优秀的拆分(power)

【题目描述】

一般来说，一个正整数可以拆分成若干个正整数的和。例如，$1=1$，$10=1+2+3+4$ 等。

对于正整数 n 的一种特定拆分，我们称它为“优秀的”，当且仅当在这种拆分下，n 被分解为了若干个**不同**的 2 的**正整数**次幂。注意，一个数 x 能被表示成 2 的正整数次幂，当且仅当 x 能通过正整数个 2 相乘在一起得到。

例如，$10=8+2=2^3+2^1$ 是一个优秀的拆分。但是，$7=4+2+1=2^2+2^1+2^0$ 就不是一个优秀的拆分，因为 1 不是 2 的正整数次幂。

现在，给定正整数 n，你需要判断这个数的所有拆分中，是否存在优秀的拆分。若存在，请你给出具体的拆分方案。

优秀的拆分(power)

【输入格式】

输入文件名为 power.in。

输入文件只有一行，一个正整数 n，代表需要判断的数。

【输出格式】

输出文件名为 power.out。

如果这个数的所有拆分中，存在优秀的拆分。那么，你需要**从大到小**输出这个拆分中的每一个数，相邻两个数之间用一个空格隔开。可以证明，在规定了拆分数字的顺序后，该拆分方案是唯一的。

若不存在优秀的拆分，输出“-1”（不包含双引号）。

优秀的拆分(power)

【样例 1 输入】

6

【样例 1 输出】

4 2

【样例 1 解释】

$6 = 4 + 2 = 2^2 + 2^1$ 是一个优秀的拆分。注意，$6 = 2 + 2 + 2$ 不是一个优秀的拆分，因为拆分成的 3 个数不满足每个数互不相同。

优秀的拆分(power)

- 拿样例举几个例子吧

十进制	二进制	
10	1010	$10 = 2^3 + 2^1$
7	111	$7 = 2^2 + 2^1 + 2^0$
6	110	$6 = 2^2 + 2^1$

- 几个关键点:
- 1.注意拆分中一定是若干不同的正整数幂
- 2.输出时从第一项到最后一项是从大到小的
- 3.如果不存在题中要求的拆分，那么记得输出-1

https://en.wikipedia.org/wiki/Binary_number

优秀的拆分(power)

- 思考方式一：不能看出是转换二进制，直接按题意模拟
- 拆分的第一个正整数幂为$2^k<=n$的情况下k的最大值
- 当我们将n中拆分出第一个2^k后，n中仍未被拆分的部分就是$n- 2^k$
- 所以我们只要不断重复 拆分->$n- 2^k$ 直到n为0即可
- 大致代码结构

```
while(n!=0){
    s=1;
    while(s<=n) s=s*2;
    n=n-s;
    cout<<s<<" ";
}
```

https://en.wikipedia.org/wiki/Binary_number

优秀的拆分(power)

- 现在来考虑之前提到的关键点
- 1.拆分中是否会出现两个相同的正整数幂
- 如果存在两个相同的2^k+2^k，就相当于2^{k+1},和我们要求的$2^k<=n$中k取最大值相违背，所以一定不会出现两个相同的正整数幂
- 2.输出时从第一项到最后一项是从大到小的
- 我们每次都尽可能的拆分大的正整数幂，所以按照我们输出的顺序，一定是从大到小的，不需要刻意的进行排序或者是其他处理
- 即任何一个数在二进制下的表示是唯一的

https://en.wikipedia.org/wiki/Binary_number

优秀的拆分(power)

- 现在来考虑之前提到的关键点
- 3.如果不存在题中要求的拆分，那么记得输出-1
- 不存在的情况仅为拆分出了2^0，因为2^0 =1，2的正整数幂均为偶数，所以如果n是一个优秀拆分的情况，那么n必然是一些偶数相加的和，即n也为偶数。只有n中包含2^0 也就是n为奇数的时候，才是“不存在”的情况。所以只要在拆分之前特殊判断一下n是否为奇数即可

```
if(n%2==1){
    cout<<"-1";
    return 0;
}
```

https://en.wikipedia.org/wiki/Binary_number

优秀的拆分(power)

- 思考方式二：通过观察和思考，看出了题目本质是一个二进制拆分
- 那么可以根据进制转换，将输入的n转换为2进制存储在数组中，倒序输出。

https://en.wikipedia.org/wiki/Binary_number

优秀的拆分(power)

```
cin>>n;
if(n%2==1){
    cout<"-1";
    return 0;
}
k=0;s=1;
while(n!=0){
    if(n%2==1) a[k]=s;
    s=s*2;
    k++;
    n=n/2;
}
for(int i=k-1;i>=0;i--)
    if(a[i]!=0)
        cout<<a[i]<<" ";
return 0;
```

https://en.wikipedia.org/wiki/Binary_number

优秀的拆分(power)

- 如果你想看更优雅的代码

```
if (n & 1){
    puts("-1");
}else{
    vector <int> ans;
    for (int i = 25; i > 0; i--){
        if (n & (1<<i)){
            ans.push_back(i);
        }
    }

    for (int i = 0; i < ans.size(); i++){
        printf("%d ", (1<<ans[i]));
    }
    puts("");
}
```

https://en.wikipedia.org/wiki/Binary_number

Junior -直播获奖(live)

统计?

直播获奖(live)

NOI2130 即将举行。为了增加观赏性，CCF 决定逐一评出每个选手的成绩，并直播即时的获奖分数线。本次竞赛的获奖率为 $w\%$，即当前排名前 $w\%$ 的选手的最低成绩就是即时的分数线。

更具体地，若当前已评出了 p 个选手的成绩，则当前计划获奖人数为 $\max(1, \lfloor p \times w\% \rfloor)$，其中 w 是获奖百分比，$\lfloor x \rfloor$ 表示对 x 向下取整，$\max(x, y)$ 表示 x 和 y 中较大的数。如有选手成绩相同，则所有成绩并列的选手都能获奖，因此实际获奖人数可能比计划中多。

作为评测组的技术人员，请你帮 CCF 写一个直播程序。

直播获奖(live)

【样例 1 输入】

10 60
200 300 400 500 600 600 0 300 200 100

【样例 1 输出】

200 300 400 400 400 500 400 400 300 300

【样例 1 解释】

已评测选手人数	1	2	3	4	5	6	7	8	9	10
计划获奖人数	1	1	1	2	3	3	4	4	5	6
已评测选手的分数从高到低排列（其中，分数线用**粗体**标出）	**200**	**300** 200	**400** 300 200	500 **400** 300 200	600 500 **400** 300 200	600 600 **500** 400 300 200	600 600 500 **400** 300 200 0	600 600 500 **400** 300 300 200 0	600 600 500 400 **300** 300 200 200 0	600 600 500 400 300 **300** 200 200 100 0

注意，在第 9 名选手的成绩评出之后，计划获奖人数为 5 人，但由于有并列，因此实际会有 6 人获奖。

直播获奖(live)

- 先按照题目描述的内容模拟:
- 每读入一个人的成绩，重新给所有的学生排个序,算出前w%
- 那么大致的程序结构

```
for(i=1~n){
    1.cin>>a[i];
    2.将a[i]从大到小排序
    3.cnt=max(1,i*w/100);
    4.cout<<a[cnt]<<" ";
}
```

直播获奖(live)

- 注意事项：

> 在计算计划获奖人数时，如用浮点类型的变量（如 C/C++中的 float、double，Pascal 中的 real、double、extended 等）存储获奖比例 $w\%$，则计算 $5 \times 60\%$ 时的结果可能为 3.000001，也可能为 2.999999，向下取整后的结果不确定。因此，建议仅使用整型变量，以计算出准确值。

- 所以在计算时不要出现小数，用整数来计算。C++整数除法时会自动向下取整
- 时间复杂度：
- 1.如果你是选择冒泡之类的排序，那么复杂度为$O(n^3)$
- 2.如果你是sort之类的排序，那么复杂度为$O(n^2 \log_2 n)$

直播获奖(live)

- 优化的关键点

对于所有测试点，每个选手的成绩均为不超过 600 的非负整数，获奖百分比 w 是一个正整数且 $1 \leq w \leq 99$。

- 发现成绩的范围不会超出600，所以我们可以用空间换取时间的方式，开一个桶来存储。
- 即cnt[i]=j表示分数为i的学生一共有j个人
- 那么我们每次只要从当前学生的（可能的）最高分600开始数一数。如果从cnt[600]数到cnt[i]，那么意味着一共有sum这么多学生的分数大于等于i。如果发现sum刚好够了w%，那么i即为分数线。
- 复杂度$O(n * 600)$, 即 $6e7$

直播获奖(live)

```
memset(cnt, 0, sizeof cnt);
for (int s, i = 1; i <= n; i++){
    scanf("%d", &s);
    cnt[s]++;
    int num_awards = max(1, i * w / 100);
    int score_line = get_score_line(num_awards);
    printf("%d ", score_line);
}
puts("");
```

直播获奖(live)

```
memset(cnt, 0, sizeof cnt);
for (int s, i = 1; i <= n; i++){
    scanf("%d", &s);
    cnt[s]++;
    int num_awards = max(1, i * w / 100);
    int score_line = get_score_line(num_awards);
    printf("%d ", score_line);
}
puts("");
```

```
int get_score_line(int num_awards){
    int sum = 0;
    for (int i = 600; i >= 0; i--){
        sum += cnt[i];
        if (sum >= num_awards){
            return i;
        }
    }
    return -1;
}
```

Junior -表达式(expr)

表达式树

表达式(expr)

- 题意：

- 给个bool表达式和每个变量的初始值
- 每次将一个变量取反，问计算结果
- 询问独立

表达式(expr)

- 思路:
- 1.根据输入的后缀表达式可以建一棵表达式树

```
stack<int> stk;
while(cin >> atom) {
    if (atom[0] == 'x') {
        int x = 0,i = 1;
        while (i < atom.size()) {
            x = x * 10 + atom[i] - '0';
            i++;
        }
        id[x] = ++sz;
        op[sz] = -1;
        stk.push(sz);
    }
    else if (atom[0] == '!') {
        ++sz;
        op[sz] = 0;
        lson[sz] = stk.top();
        stk.pop();
        stk.push(sz);
    } else if (atom[0] == '&') {
        ++sz;
        op[sz] = 1;
        rson[sz] = stk.top();
        stk.pop();
        lson[sz] = stk.top();
        stk.pop();
        stk.push(sz);
    }
     else if (atom[0] == '|') {
        ++sz;
        op[sz] = 2;
        rson[sz] = stk.top();
        stk.pop();
        lson[sz] = stk.top();
        stk.pop();
        stk.push(sz);
    }
}
```

表达式(expr)

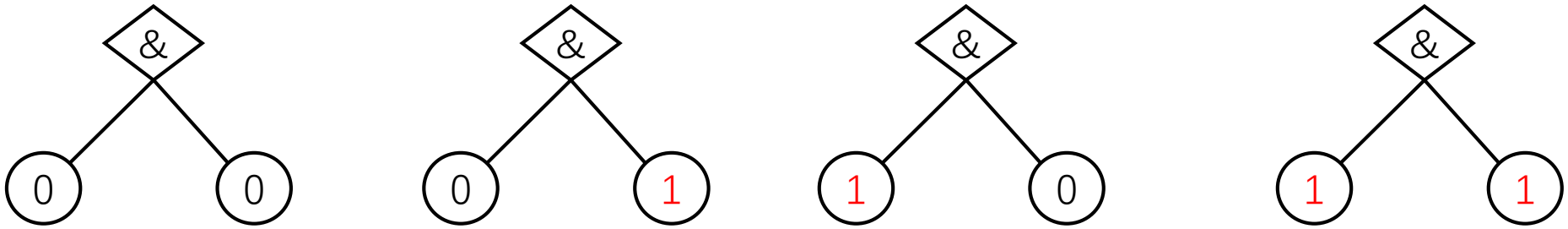
- 思路:
- 2.先dfs一次处理“未修改”时每个子树的数值

```
void update(int x) {
    if (op[x] == -1) {//x1 x2等
        return;
    }
    if (op[x] == 0) {//!
        res[x] = !res[lson[x]];
    } else if (op[x] == 1) {//&
        res[x] = (res[lson[x]] & res[rson[x]]);
    } else if (op[x] == 2) {//|
        res[x] = (res[lson[x]] | res[rson[x]]);
    }
}
void dfs(int o) {
    if (lson[o]) {
        dfs(lson[o]);
    }
    if (rson[o]) {
        dfs(rson[o]);
    }
    update(o);
}
```

表达式(expr)

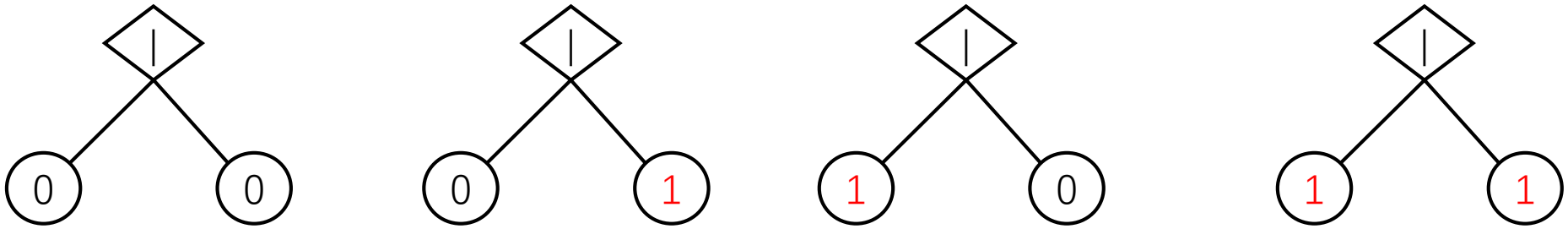
- 思路:
- 3.再预处理一下每个点的数值发生改变是否会影响最终答案
- 每个点的数值发生修改，只会影响根结点到当前结点这部分的数值。
- 所以如果当前结点(xi)的数值发生改变，只要能够对“根结点到当前结点的路径”产生影响，那么最终结果则会改变

表达式(expr)



- 情况1：原结果为0，无论改变左还是右，结果仍然为0
- 情况2：原结果为0，如果改变左，结果为1;如果改变右，结果为0
- 情况3：原结果为0，如果改变左，结果为0;如果改变右，结果为1
- 情况4：原结果为1，无论改变左还是有，结果都变为0
- 结论：情况1一定不变，其余情况（左右中至少含有一个1）均可能发生改变

表达式(expr)



- 情况1：原结果为0，无论改变左还是右，结果都变为1
- 情况2：原结果为1，如果改变左，结果为1;如果改变右，结果为0
- 情况3：原结果为1，如果改变左，结果为0;如果改变右，结果为1
- 情况4：原结果为1，无论改变左还是有，结果都变为1
- 结论：情况4一定不变，其余情况（左右中至少含有一个0）均可能发生改变

表达式(expr)

- 情况1：原结果为0，改变左，结果为1
- 情况2：原结果为1，改变左，结果为0
- 结论：两种情况均会发生改变

表达式(expr)

```
void go(int o) {
    if (op[o] == -1) {
        tag[o] = true;
        return;
    }
    if (op[o] == 0) { // !
        go(lson[o]);
    }
    if (op[o] == 1) { // &
        if (res[lson[o]] == 1 && res[rson[o]] == 1) {
            go(lson[o]);
            go(rson[o]);
        }
        if (res[lson[o]] == 0 && res[rson[o]] == 1) {
            go(lson[o]);
        }
        if (res[lson[o]] == 1 && res[rson[o]] == 0) {
            go(rson[o]);
        }
    }
```

表达式(expr)

```
if (op[o] == 2) { // |
    if (res[lson[o]] == 0 && res[rson[o]] == 0) {
        go(lson[o]);
        go(rson[o]);
    }
    if (res[lson[o]] == 1 && res[rson[o]] == 0) {
        go(lson[o]);
    }
    if (res[lson[o]] == 0 && res[rson[o]] == 1) {
        go(rson[o]);
    }

}
```

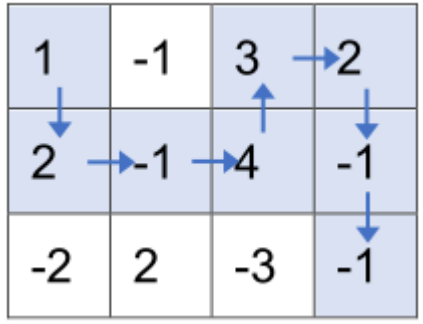
Junior -方格取数(number)

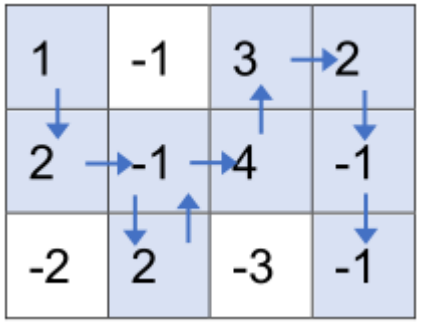
dp

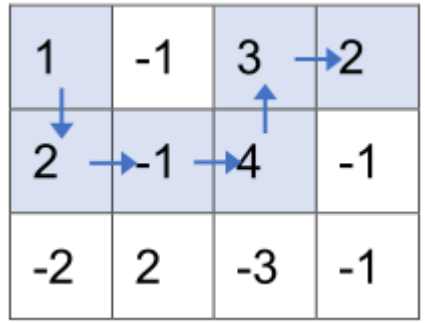
方格取数(number)

- 题意

- 一个n*m的方格，每次只能向下向上向右走，不能重复
- 问从左上走到右下，经过的格子的总和最大是多少







方格取数(number)

- 解法：

- 状态定义：令dp[i][j][0/1]表示在(i，j)位置下一步向上/下的能获得的最大值

- 根据题目要求，每次移动时列不是+1就是不变，行可能+1可能-1，可能不变。
- 所以我们在枚举时要将列放在外层循环，行放在里层循环

方格取数(number)

- 解法:

- 根据题意，状态转移方程有:
- `dp[i][j][0] + a[i + 1][j] -> dp[i + 1][j][0]`
- `dp[i][j][1] + a[i - 1][j] -> dp[i - 1][j][1]`
- `dp[i][j][0] + a[i][j + 1] -> dp[i][j + 1][0/1]`
- `dp[i][j][1] + a[i][j + 1] -> dp[i][j + 1][0/1]`

方格取数(number)

```
for (int j = 0; j < m; j++) {
    for (int i = 0; i < n; i++) {
        if (i + 1 < n) {
            upmax(dp[i + 1][j][0], dp[i][j][0] + a[i + 1][j]);
        }
        if (j + 1 < m) {
            upmax(dp[i][j + 1][0], dp[i][j][0] + a[i][j + 1]);
            upmax(dp[i][j + 1][1], dp[i][j][0] + a[i][j + 1]);
        }

    }
    for (int i = n - 1; ~i; i--) {
        if (i - 1 >= 0) {
            upmax(dp[i - 1][j][1], dp[i][j][1] + a[i - 1][j]);
        }
        if (j + 1 < m) {
            upmax(dp[i][j + 1][0], dp[i][j][1] + a[i][j + 1]);
            upmax(dp[i][j + 1][1], dp[i][j][1] + a[i][j + 1]);
        }
    }
}
```

```
template<class T>
inline void upmax(T& x, T y) {
    x = std::max(x, y);
}
```

Senior –儒略日(julian)

数数

儒略日(julian)

- 题意:

- 给定历法
- 计算从 公元前 4713 年 1 月 1 日 向后的 r 天的具体日期

儒略日(julian)

- 思路：

- 做法有很多
- But边界条件太多，容易翻车

- 从 公元前 4713 到 2000年，大约3e6天，day by day
- 实现一个 next_day()函数即可，特判题目中的各种条件

儒略日(julian)

- 观察到： 大约从21世纪(2000年)开始，每400年可以看作一个周期，那么超出400年的部分(+month+day)可以直接取模，只需要计算模剩下的部分

- 那么模剩下的部分如何计算：day by day
- 再次调用next_day()函数

儒略日(julian)

```
struct Date{
    // ...
};

Date next_day(Date){
    // ...
    // 考虑题中描述的各种条件计算
    return  next_day;
}
```

儒略日(julian)

```
struct Date{
    // ...
};

Date next_day(Date){
    // ...
    // 考虑题中描述的各种条件计算
    return  next_day;
}
```

```
// 预处理
day = -4713.1.1
while (day < 2400.1.1) {
    days[i++] = next_day(day)
    day = days[i - 1]
}
```

儒略日(julian)

```
// 计算
if (input_number < length1){
    output_directly();
}else{
    cycle = (input_number - lengh1) / length_400years;
    remain = (input_number - lengh1) % length_400years;
    ans = compute_y_m_d(cycle, remain);
    output << ans << endl;
}
```

Senior -动物园(zoo)

?

动物园(zoo)

动物园里饲养了很多动物，饲养员小 A 会根据饲养动物的情况，按照《饲养指南》购买不同种类的饲料，并将购买清单发给采购员小 B。

具体而言，动物世界里存在 2^k 种不同的动物，它们被编号为 $0 \sim 2^k - 1$。动物园里饲养了其中的 n 种，其中第 i 种动物的编号为 a_i。

《饲养指南》中共有 m 条要求，第 j 条要求形如“如果动物园中饲养着某种动物，满足其编号的二进制表示的第 p_j 位为 1，则必须购买第 q_j 种饲料”。其中饲料共有 c 种，它们从 $1 \sim c$ 编号。本题中我们将动物编号的二进制表示视为一个 k 位 01 串，第 0 位是最低位，第 $k-1$ 位是最高位。

根据《饲养指南》，小 A 将会制定饲料清单交给小 B，由小 B 购买饲料。清单形如一个 c 位 01 串，第 i 位为 1 时，表示需要购买第 i 种饲料；第 i 位为 0 时，表示不需要购买第 i 种饲料。

实际上根据购买到的饲料，动物园可能可以饲养更多的动物。更具体地，如果将当前未被饲养的编号为 x 的动物加入动物园饲养后，饲料清单没有变化，那么我们认为动物园当前还能饲养编号为 x 的动物。

现在小 B 想请你帮忙算算，动物园目前还能饲养多少种动物。

动物园(zoo)

【样例 1 输入】

```
3 3 5 4
1 4 6
0 3
2 4
2 5
```

【样例 1 输出】

```
13
```

【样例 1 解释】

动物园里饲养了编号为 1、4、6 的三种动物，《饲养指南》上 3 条要求为：

1. 若饲养的某种动物的编号的第 0 个二进制位为 1，则需购买第 3 种饲料。
2. 若饲养的某种动物的编号的第 2 个二进制位为 1，则需购买第 4 种饲料。
3. 若饲养的某种动物的编号的第 2 个二进制位为 1，则需购买第 5 种饲料。

饲料购买情况为：

1. 编号为 1 的动物的第 0 个二进制位为 1，因此需要购买第 3 种饲料；
2. 编号为 4、6 的动物的第 2 个二进制位为 1，因此需要购买第 4、5 种饲料。

由于在当前动物园中加入一种编号为 0,2,3,5,7,8,…,15 之一的动物，购物清单都不会改变，因此答案为 13。

动物园(zoo)

- 根据现有动物和饲养规则计算出已经购买的饲料

- 查看是否有:
- 对于没有买的饲料，如果他出现在了饲养规则中
- 其对应的二级制位不能为1，然后乘法原理计数

Senior - 函数调用(call)

?

函数调用(call)

函数调用(call)

Senior - 贪吃蛇(snakes)

?

贪吃蛇(snakes)

草原上有 n 条蛇，编号分别为 $1,2,\cdots,n$。初始时每条蛇有一个体力值 a_i，我们称编号为 x 的蛇实力比编号为 y 的蛇强当且仅当它们当前的体力值满足 $a_x > a_y$，或者 $a_x = a_y$ 且 $x > y$。

接下来这些蛇将进行决斗，决斗将持续若干轮，每一轮实力最强的蛇拥有选择权，可以选择吃或者不吃掉实力最弱的蛇：

1. 如果选择吃，那么实力最强的蛇的体力值将减去实力最弱的蛇的体力值，实力最弱的蛇被吃掉，退出接下来的决斗。之后开始下一轮决斗。
2. 如果选择不吃，决斗立刻结束。

每条蛇希望在自己不被吃的前提下在决斗中尽可能多吃别的蛇（显然，蛇不会选择吃自己）。

现在假设每条蛇都足够聪明，请你求出决斗结束后会剩几条蛇。

本题有多组数据，对于第一组数据，每条蛇体力会全部由输入给出，之后的每一组数据，会相对于上一组的数据，修改一部分蛇的体力作为新的输入。

贪吃蛇(snakes)

- “现在假设每条蛇都足够聪明”
- ->决定权在体力最高的一条蛇上
- ->只要最后一条蛇不吃蛇吃到自己变成最小的

- 即 if max - min < min2

我用心感受了一下 确实不会被吃到

贪吃蛇(snakes)

- 实现：

- set 直接取begin() end()

QA