NOIP 模拟赛

题目名称	**括号序列**	**和数检测**	**与**
输入文件名	bracket.in	check.in	and.in
输出文件名	bracket.out	check.out	and.out
每个测试点时限	1 sec	以评测机实际测试时间为准	1 sec
内存限制	128M	256M	128M
测试点数目	10	20	20
每个测试点分值	10	5	5
是否有部分分	无	无	无
题目类型	传统	传统	传统

提交源程序须加后缀

对于 Pascal 语言	bracket.pas	check.pas	and.pas
对于 C 语言	bracket.c	check.c	and.c
对于 C++ 语言	bracket.cpp	check.cpp	and.cpp

括号序列

【问题描述】

一个由小括号组成的字符串可以被称为一个括号序列。但一个括号序列可能并不满足括号匹配的要求。因此，我们可以进一步将满足括号匹配的括号序列成为“标准的括号序列。例如字符串")((())"是一个括号序列但不是标准的括号序列，而字符串"()(())"是一个标准的括号序列。

给定一个括号序列，你需要对求出：这个括号序列的所有不同的子串中，有多少个是标准的括号序列？

一个括号序列的子串指的是这个序列从某个位置起始、到某个位置截止的子字符串。如果两个子串拥有不同的起始位置或截止位置，那么它们就被认为是括号序列的不同的子串。

【数据规模和约定】

设输入字符串的长度为n。

对于100%的数据，满足$1 \le n \le 10^6$。

【简要题解】

把左括号当成$+1$，右括号当成-1，并求出变换后的序列的前缀和数组s。括号序列的第l到第r个字符构成的子串是标准的括号序列，当且仅当$s[l] = s[r]$，并且$s[l], s[l+1], \dots, s[r]$中，没有比$s[l]$更小的数。

如果只考虑$s[l] = s[r]$这一个判别条件，我们可以这么操作：用指针i从左到右扫描，并用数组f记录如下信息$f[x]$：表示截止到i之前，有多少j满足$s[j] = x$。当枚举到i时，只需要把$f[s[i]]$加入答案，再让$f[s[i]]$加一即可。

现在把“$s[l], s[l+1], \dots, s[r]$中，没有比$s[l]$更小的数”这一条件纳入考虑。事实上我们只需要在枚举到i后，把$f[s[i]+1], f[s[i]+2], \dots$全部清零即可。因为在$i$之后，如果有某个位置$k$满足$s[k] = s[i] + t, t > 0$，那么以$k$作为截止位置的标准的括号序列一定不能以一个小于$i$的位置作为起始位置。当然，我们不能暴力地进行清零操作。不过我们可以按如下方法进行考虑：考虑$s[i]$与$s[i-1]$，只会有两种不同的情况：如果$s[i] = s[i-1] + 1$，则在$s[i-1]$的时候，$f[s[i]], f[s[i]+1], f[s[i]+2], \dots$均被清零，所以不需要进行额外的操作；如果$s[i] = s[i-1] - 1$，则只有$f[s[i]+1]$需要被清零，其余部分都已经处理完毕。按照上述方法进行操作即可在$O(n)$的时间内解决问题。

和数检测

【问题描述】

给定n个正整数$d_1, d_2, \ldots, d_n$。如果取出其中的任意两个数（可以相同），则可以得到这两个数的和。对于n个数，则至多可以产生$\frac{n\times(n+1)}{2}$种不同的和。

给出正整数m，你需要判断：是否存在两个整数u, v，满足$d_u + d_v = m$。

【数据规模和约定】

对于20%的数据，满足$n \le 1000$，$m \le 10000$。

对于50%的数据，满足$n \le 10^5$。

对于另20%的数据，满足$m \le 10^7$。

对于100%的数据，满足$1 \le n \le 10^6$，$1 \le d_i \le m \le 10^9$，$1 \le T \le 20$。

【简要题解】

先考虑“另20%的数据”的做法：使用计数数组进行记录。

用$c[i]$表示是否存在u满足$d_u = i$。枚举d_v，判断$c[m-d_v]$是否为$true$即可。用$O(n)$的时间枚举所有d_i即可求出c数组，同样用$O(n)$的时间可以清空c数组，总时间复杂度为$O(n)$。

再来考虑100%的数据。可以发现，在上面的做法中，我们的时间复杂度和变量m并没有关系，唯一的问题在于长度为m的数组c开不下。因此，我们可以使用一种“分批处理”的方法来优化空间：

考虑设置一个数b，把每个d_i写成$d_i = a_i \times b + r_i$，并且把每个d_i按a_i的值分成若干组，a_i相同的数放在一组里。

一次性处理某一组的数。如果假设$d_u + d_v = m$的一个数在某一组中，那么另一个数的分布范围一定不超过两组。所以，如果要检测某一组的数是否可以作为一个加数，只需要把另外的两组放入计数数组即可。这样数组c的大小是$O(b)$的，组的数量是$O(m/b)$的，将两者调整到可以接受的空间范围（比如取$b=\sqrt{m}$）即可解决计数数组开不下的问题。

与

【问题描述】

你现在得到了n个非负整数$a_1, a_2, \ldots, a_n$，你需要求出有多少种方法可以将它们分成两部分，使得两部分都至少有一个数，并且两部分的数进行**按位与**操作后的结果相同。

按位与是一种对于二进制数的操作，它等价于C与$C++$里的运算&和$Pascal$里的运算and。即，将两个数写成二进制，较短的数补前导零使得两个数一样长。然后如果两个数在某一位上都是1，那么这一位运算的结果为1；否则这一位为0。例如两个整数14和11，它们按位与运算后的结果应为10。

【数据规模和约定】

对于100% 的数据，满足$1 \le n \le 60$，$0 \le a_i < 131072$。

【简要题解】

使用容斥原理来解决本题，则答案可以被写成如下形式：

$ans = \sum_{c=0}^{131071} (-1)^c \times f(c)$

其中，变量c枚举的是子集，代表需要限制为不符合要求的二进制位（即这些位一定得一边是0一边是1）。$f(c)$指在c的限制下的方案数。

由于有一些位一定得一边是0一边是1，所以这些位为1的元素一定得归一边，可以考虑把它们合并为一个元素。如果有多个位有限制，那就用并查集来合并元素。合并后，剩余的这些元素可以被任意分到左侧或右侧，则$f(c) = 2^k - 2$，其中k表示合并后的元素个数，减去2是为了避免有一侧没有分配到元素。

如果n个数的某个二进制位全部为0或为1，则可以直接把这一位去掉。这样可以避免在后续计算时的特殊讨论。