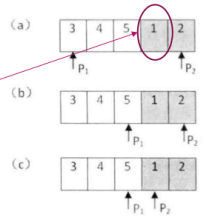
二分搜索的妙用.

- 通常的应用：在递增或递减序列中找目标值
- 还可用于最优解：比如经典的割绳子问题 （**LeetCode 1891. 割绳子**），有n条绳子，长度不等，需要将所给的绳子能切割成k条长度均为L的绳子，求L最大能达到多少。绳子长度 L变长，那么得到的绳子的根数 k 不会变多，具有单调性。

扩展应用：
把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。输入一个递增排序的数组的一个旋转，输出旋转数组的最小元素。例如，数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。

旋转之后的数组实际上可以划分为两个排序的子数组，而且前面子数组的元素都大于或者等于后面子数组的元素。最小的元素刚好是这两个子数组的分界线。



Mid > p1? → 最小值在Mid右边：令 p1 := Mid
Mid < p2? → 最小值在Mid左边：令 p2 := Mid+1
什么时候循环结束？
当p2=p1+1时结束，最大值为p1，最小值为p2.

规律类问题的分析.

- 可以先从小规模样例入手，或者先去掉一个限制简化问题，之后再去解决原问题

- Lanran想要举办一场**最多**有3个问题的编程比赛。他有一个题目库，每个问题都有自己的难度等级。竞赛选题有一些要求:所选题目的难度不能相互被对方整除，不能选2个同等级难度的问题，而Lanran希望所选问题的难度之和尽可能大。请输出最大可能的难度和。

在O(n)内解决

解题思路

观察以下非递增序列:

30 29 16 15:**三个最大的数都不能被对方整除**，返回a[0]+a[1]+a[2]

30 16 15 2:**a[0]不能被a[1]整除，但a[0] 能被a[2]整除: a[2]/a[0] = 1/2**(或小于1/2)，选择a[0]和a[1]，然后遍历序列的其余部分，直到找到不是[0]和[1]的因子的最大的元素。

a[0]能被a[1]整除:

30 15 10 6:**a[1] /a[0] = 1/2 a[2] /a[0] = 1/3** a[3] /a[0] = 1/5 观察到:1/2 + 1/3 + 1/5 > 1

30 15 6 5:**a[1] /a[0] = 1/2 a[2] /a[0] = 1/5** a[3] /a[0] = 1/6观察到:1/2 + 1/5 + 1/6 < 1

当a[1]/a[0] = 1/2时，

选择a[0]，然后遍历序列的其余部分，以找到不是a[0]因子且满足约束的最大的其他2个元素。

选择a[1]，然后遍历序列的其余部分，以找到不是a[1]因子满足约束的最大的其他2个元素。

30 10 9 8: **a[1]/a[0] = 1/3(或小于1/3)** 由此可推断 a[2]/a[0] < 1/3 a[3]/a[0] < 1/3(a[1]和后面任何数字的组合都不可能超过a[0])

选择a[0]，然后遍历序列的其余部分，以找到不属于a[0]的因子且满足约束的最大的其他2个元素。

运算注意

- 需要快速得到数组中某一连续段的和（乘积，异或），都可以用前缀和（类似）方法
- 乘法和加法，要考虑会不会可能有整数溢出；取模要考虑负数情况和除法情况；除法要考虑除数是否为0

前缀和、积、异或

- 前缀和

Sum(i,j) = sum[j]-sum[i-1]

- 前缀积

Product(i,j) = Product[j] / Product[i-1]

- 前缀异或

xor(i,j) = xor[j]^xor[i-1]

Sum数组中sum(i)即原数组中前i个元素的和.
要获得(i,j)的和，只需 Sum[j]- Sum[i-1]

例题

给定一个整数数组(元素>0)和一个正整数 k，找到该数组中和为 k 的连续的子数组的个数。

输入： a = [1,2,2,0,1]， k = 3
输出：2 (a1+a2=3\a3+a4+a5=3)

解题思路：

符合条件的子数组[ai…aj]要满足连续元素和等于k，即sum[j]-sum[i-1] ==k 。

a = [1,2,2,0,1]
sum = [0,1,3,5,5,6]
初始值，前缀和。

在算Sum数组时即可构造 初始次数均为1.

如果ai的取值范围比较大，而a的长度相对小，可用Hashmap存储sum数组中每个数值出现的次数：

数值	次数
0	1
1	1
3	1
5	2
6	1

遍历sum中的元素si，判断si+k是否存在于hashmap中，如果存在，则可将次数累计到答案中。

例如s0=0, 在hashmap尝试找元素3，找到了，3对应的次数为1，则result+=1，此时 result 等于1；
S2=3，找到元素6， 3对应的次数为1， result+=1，此时 result 等于2，后续均无累计，返回result ，值为2。

如果出现负数，上述解题思路不适用。
反例：
a = [4,-1,-3,3]
sum = [0,4, 3, 0, 3] k = 3
s0 = 0 3出现2次
s3 = 0 3出现2次
总计4次
但实际上，[4,-1], [3],[4,-1,-3,3] 符合条件的子数组只有3个。
原因：当 s3 = 0时， 不应该统计s2 = 3 的 这个3。

更通用的解题思路：

在构建Hashmap时处理，避免了重复的可能.

a = [4,-1,-3,3] k = 3

可边计算sum，边构造hashmap，边统计子数组数量。关键在于每次累计sum[i]-k对应的次数。
具体操作如下：

sum[0] = 0 Hashmap:

数值	次数
0	1

result: 0

sum[0] – 3 = -3 hashmap中是否有-3?否

Hashmap:
sum[1] = sum[0]+a1=4

数值	次数
0	1
4	1

result: 0

sum[1] – 3 = 1 hashmap中是否有1?否

Hashmap:
sum[2] = sum[1]+a2=3

数值	次数
0	1
4	1
3	1

result: 1

sum[2] – 3 = 0 hashmap中是否有0?有
result +=1

Hashmap:
sum[3] = sum[2]+a3=0

数值	次数
0	2
4	1
3	1

result: 1

sum[3] – 3 = -3 hashmap中是否有-3?否

Hashmap:
sum[4] = sum[3]+a3=3

数值	次数
0	2
4	1
3	2

result: 3

sum[4] – 3 = 0 hashmap中是否有0?有
result +=2

模运算

在算法题目中，有时候我们会遇到超过long(C++: long long)范围的数据，这个时候题目往往会要求我们输出答案对某个质数取模的结果。这个时候就涉及到模运算。

模运算与基本四则运算比较类似（除法除外）。

$$(a + b) \% n = (a \% n + b \% n) \% n$$

$$(a - b) \% n = (a \% n - b \% n) \% n$$

$$(a * b) \% n = (a \% n * b \% n) \% n$$

$$a \hat{} b \% p = ((a \% p) \hat{} b) \% p$$

模运算常见错误

```cpp
#include <iostream>
using namespace std;

using ll = long long;
const ll mod = 998244353;

int main() {

    ll a = 1e10, b = 1e10, c;
    c = a * b; //可能溢出
    c %= mod;
    // 正确写法：c = (a % mod) * (b%mod) % mod;

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

using ll = long long;
const ll mod = 998244353;

int main() {

    ll a = 2, b = 9, c;
    c = (a - b) % mod;//可能为负
    // 正确写法：c = (a - b + mod) % mod;

    return 0;
}
```

模运算除法

在模运算中是不能直接使用除法的。所以，我们引入乘法逆元的概念：

在（mod p）意义下（ p 是素数），如果 a * a' = 1 (mod p)，那么我们就说 a' 是 a 的逆元。当然，反过来， a 也是 a' 的逆元。

乘法逆元的性质：

存在唯一性

a * inv[b] = a / b (mod p)

求解乘法逆元的方法：

费马小定理　扩展欧几里得

欧拉筛　　线性递推

```cpp
#include <iostream>
using namespace std;

using ll = long long;
const ll mod = 998244353;

int inv[20];

int main() {

    ll a = 10, b = 5, c;
    c = (a / b) % mod
    // 正确写法：c = a * inv[b] % mod;

    return 0;
}
```

ST表(Sparse Table 稀疏表)：不支持修改，$\Theta(n\log n)$处理，$\Theta(1)$查询.