算法常用解题技巧

赵耀

算法常用技巧

- 空间换时间
- 二分搜索的妙用
- 规律类问题分析方法
- 大事化小
- 运算注意事项
- 对拍
- C++编码注意事项

空间换时间

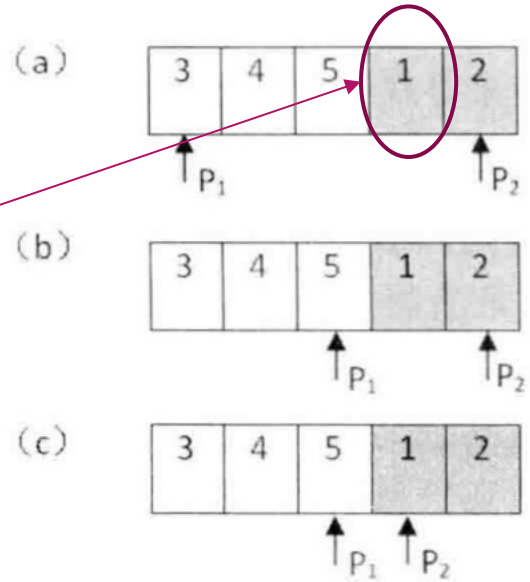
- **Memoization(记忆化)**
- **动态规划:**后续课程将学习这类算法

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?
Mid < p2?
什么时候循环结束？

规律类问题的分析方法

- 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

例题

- 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。

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。

大事化小

- 贪心、分治和动态规划都体现了将大的问题分解或转化成小问题的分析思路
- 这也是本学期学习的重点内容

运算注意事项

- 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]

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]

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。

出现负数，是否仍有解题思路？

更通用的解题思路：

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

```
#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;
}
```

模运算常见错误

```
#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

模运算除法

```
#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;
}
```

对拍

- 为了验证某解题程序是否正确，另外写一个暴力求解该题目的程序，然后生成一些测试数据，看同样的数据，两个程序输出的结果是否相同，不同意味着被对拍的程序有问题。
- 鼓励大家自己写暴力程序、数据生成器、对拍脚本，自行发现算法问题

以下博客讲得挺详细的，可供大家参考：

https://blog.csdn.net/Njhemu/article/details/99539576

https://blog.csdn.net/weixin_30835649/article/details/98410133

C++编码注意事项

- 调试
- 常数优化
- STL库
- C++参考资料

调试

- 使用GDB打断点单步调试和监控变量
- 打印中间变量
- 小黄鸭调试法（实测非常有用）

常数优化

- 卡常数，又称底层常数优化，特指在OI/ACM-ICPC等算法竞赛中针对程序基本操作进行的底层优化，一般在对程序性能要求较为严苛的题目或是在算法已经达到理论最优时间复杂度时使用。
- 算法课不会出现必须卡常的题目，但以防万一还是需要知道一些比较有常用的优化方法。
- 用cin与cout记得关闭同步 std::ios::sync_with_stdio(false)
- 多用inline函数
- 在熟练的前提下用指针，会比用数组少一次寻址

STL库

- STL 即标准模板库（Standard Template Library），是 C++ 标准库的一部分，里面包含了一些模板化的通用的数据结构和算法
- 常用STL容器：vector：变长数组，(unordered_)set：集合，(unordered_)map：映射，priority_queue：优先队列（堆）…
- 常用STL算法：find：顺序查找，reverse：反转数组字符串；unique：数组去重；binary_search(lower_bound)：二分查找；next_permutation：将当前排列更改为 全排列中的下一个排列…
- 只要题目没有说明不能用，用STL库可以大大提高编码效率和减少错误概率。

C++参考资料

- https://en.cppreference.com/ c++语法，标准库参考资料

补充

- https://oi-wiki.org/ 介绍算法竞赛中会用到的算法，工具等