合理分配时间

在 ACM 赛制（GXCPC）中，赛题的难易度是不明确的，应该尽量避免在难题之中花费大量时间

在 OI 赛制（NOIP、CSP 等）中，赛题的难度一般由低到高，应该尽快分辨题目的难易程度，选用合适的方法进行解答

跟榜（ACM 赛制）

在 ACM 赛制中，会有一个**排行榜**，榜上有着其他选手的做题进度，我们可以参考其他选手的做题顺序和题目的总通过数来对题目的难易度进行一个大致的判断

- 提交次数多并且正确率高的题一般比较好想到解答方法，是简单题
- 提交次数多的题但是正确率很低的题可能是有坑的题
- 观察和自己相同水平的队伍的写题顺序

及时止损

在比赛之中，很可能会因为赛时的紧张等原因，使得你并不是处于最佳状态。
往往体现在：

1. 大家都在做的题目，自己一点思路都没有
2. 有了思路，但是代码难以实现，反复进行大量的删改
3. 代码写到一半，感觉自己的思路完全不正确，又重新思考，重新写代码
4. （ACM 赛制）自己认为题目解法正确，但是多次提交后都没办法通过题目

此时就需要及时止损，我们需要限制一个题目上花费的**连续思考**时间。我们在同一个题目上花费大量的时间缺毫无进展，我们称之为 `卡题`

比赛过程中，我们需要尽量的避免 `卡题` 的出现，不需要在同一个题目上死磕太长时间，将注意力转移到下一个可能做出来的题目上，因为说不定再回头看这一题的时候，就能把正解直接想出来。

骗分（OI 赛制）

题目从开始卡到结束的情况是我们都不想看到的。

所以我建议，如果当前时间，所有的题目都被卡住了，可以选择其中一题，使用暴力或者模拟的方法去写代码。

在写暴力和模拟时，思考的东西并不算多（只需要按照题目的要求去做就行），而且还能拿到不少的分数，写的过程中还可以一边思考正确的解法。

往往在通过代码去一步步实现的时候，还能找到题目的隐藏条件，也就是之后说到的**性质**，或者能够发现这个题的规律。

> 反正不要完全放弃某一个题目，至少把样例输出，如果是判断题直接输出 YES 或者 NO，能拿到多少分算多少

如何对一个题目进行分析

时间复杂度

通过时间复杂度对题目进行分析

什么是时间复杂度

我们衡量一个算法的效率时，最重要的不是看它在某个数据规模下的用时，而是看它的用时随数据规模而增长的趋势，即 **时间复杂度**。

用来衡量你的代码的运行速度的。

通常，我们认为计算机在一秒钟能够执行 $5 * 10^8$ 次运算，如果题目給出的时间限制为1s,那么你选择的算法执行的汁算次数最多应该在 10^8 量级才有可能解决这个题目。

时间复杂度有什么用?

1. 可以知道自己写的代码能拿到多少分
    i. OI 赛制之中，题目的数据规模一般是从小到大，所以我们可以根据时间复杂度来计算有多少个数据点是能够稳拿分的
2. 可以通过时间限制和数据规模来得到解法的提示
    i. 通过这两个信息来推断出应该用何种算法，例如：
        a. 能够用几重循环等

如何计算时间复杂度

一般来说，我们只计算最坏时间复杂度

1. 最坏时间复杂度，即每个数据规模下用时最长的输入对应的时间复杂度。在算法竞赛中，由于输入可以在给定的数据范围内任意给定，我们为保证算法能够通过某个数据范围内的任何数据，一般考虑最坏时间复杂度。

在比赛之中，不需要计算绝对正确的复杂度，我们只需要快速的估算即可

只需要一行代码就能够得到结果的，记作 $O(1)$

一个类似于这样的循环 `for(int i = 0; i < n; i++)`，记作 $O(n)$

循环的嵌套用乘算

顺序结构 (线性)取时间复杂度最高的

举例:

```
for(i=0;i<=n;i++)
    for(j=0;j<=i;j++)
        for(k=0;k<j;k++)
```

$O(n^3)$

```
count = 0;
for (k=1;k<=n;k*=2)
    for(j=1;j<=n;j++)
        count ++;
```

$O(nlog_2n)$

空间复杂度

通过空间复杂度对自己的想法进行验证

什么是空间复杂度

空间复杂度是对一个算法在运行过程中临时占用存储空间大小的一个量度，同样反映的是一个趋势，我们用 S(n) 来定义。

```
int a;//S(1)

int a[n];//S(n)

int a[n][n];//S(n*n)
```

判断该题是否能写

主要用于判断是否会超出内存限制，即自己提交的代码会不会直接超出内存限制（0分）

一般而言，我们能够定义的变量数量不超过 $1e9$ 就差不多
就是需要避免定义太大的数组就行

我们使用的是 c++语言，可以使用动态数组 `vector` 或者定义数组时使用变量来规定长度来避免内存超限

如何找到题目的解法

寻找题目的性质

题目的解法一定是由性质的堆叠而推导出来的

就像证明一个形状是三角形一样：

1. 因为有三条边
2. 因为有三个角
3. 因为形成一个封闭图形
4. 所以这是一个三角形

1、2、3 点我们可以认为是由题目而来的性质，结论是正解

有一些性质是题目会直接给出的，有一些性质是题目隐藏的

暴力

什么是暴力

暴力法（也称穷举法、枚举法或蛮力法）是指采用遍历（扫描）技术，即采用一定的策略将待求解问题的所有元素依次处理一次，从而找出问题的解。

什么时候写暴力

1. 数据规模小的时候
2. 时间不够的时候

怎么写暴力

将每种可能的情况都列举出来

例题

```
/*
鸡兔同笼问题

有n只脚，鸡有两只脚，兔有四只脚，问有几只鸡几只兔?
 (n<=1e8)
我们枚举其中一种动物的数量，计算出另一种动物的数量，判断是否合法
*/
int main(){
        int n;
        cin>>n;

        for(int i = 0; i * 2 <= n; i++){
                if((n - (i * 2)) % 4 == 0){
                        找到答案
                }
        }

        return 0;
}
```

「广西中小学生程序设计挑战赛 2023-J组」区间筛数 (screen)

回文数很漂亮，它非常对称，形如：$1, 11, 121, 1331$ 等，给人的感觉很舒服；质数很重要，贯穿数学 2000 多年，始终在数学发展的主干道上。今天我们将两者相结合，请你筛选出区间$[a, b]$中所有的既是回文数又是质数的数。

输入格式

共一行，第一行共两个正整数 a, b。

输出格式

若干行，每行一个数，从小到大输出 $[a, b]$ 中所有的回文质数。

提示

数据范围

对于 100% 的数据，均有$1 <= a < b <= 10^5$。

```c
//参考代码
int is_prime(int x){
    if (x == 2 || x == 3 || x == 5 || x == 7)
        return 1;
    if (x == 1)
        return 0;
    for (int i = 2; i * i <= x; i++){
        if (x % i == 0)
            return 0;
    }
    return 1;
}

int is_hw(int x){
    int k = x;
    int tmp = 0;
    while (x)    {
        tmp *= 10;
        tmp = tmp + x % 10;
        x /= 10;
    }
    return tmp == k;
}
```

```
int main(){
    int a, b;
    cin >> a >> b;

    for (int num = a; num <= b; num++)
    {
        if (is_prime(num) && is_hw(num))
            cout << num << endl;
    }
}
```

模拟

模拟就是用计算机来模拟题目中要求的操作。用一句老话说，就是“照着葫芦画瓢”；官方化的诠释则是：根据题目表述进行筛选提取关键要素，按需求书写代码解决实际问题。

简单粗暴的模拟

2023年GXCPC A题

抽牌游戏，有三种牌αβΩ，抽到一张α就往牌堆里放置一张β，抽到一张β就往牌堆里放置一张Ω，抽到Ω时获胜，每回合抽k张卡

$n_a, n_b, n_o, k(0 \leq n_a, n_b, n_o \leq 3e8, 1 \leq k \leq n_a + n_b + n_o)$

问：最差情况下，需要几个回合结束?

题目中提问，最差情况下，所以可以得到性质

Ω最难抽到

又可以得到

需要把α和β全部抽完，才能开始抽Ω

即

每一次抽卡，一定是先抽出α，如果没有α就是抽出β，如果没有β才能抽出Ω

结合题目中提到的：抽到Ω时获胜

看第一张Ω是第几个回合被抽到的

在这题之中，我们发现，如果真的去一张一张卡的抽取，会导致运行超时

如果抽一张α，相当于 $n_a - 1$，但是同时会导致 $n_b + 1$

当 n_a、n_b、n_o 都取最大值 `3e8` 时

抽完 n_a 需要 `3e8` 次操作，产生 `3e8` 个 n_b

抽完 n_b 需要 `3e8+3e8` 次操作，产生 `3e8` 个 n_o

这样已经使用了 $3e8 + 3e8 + 3e8 = 9e8$ 次操作了!!（超出时间复杂度限制）
而我们一秒最多能接受的操作次数只有 $5e8$ 次

所以就需要进行一些优化

优化一些的模拟

在对模拟进行优化时，我们一般往 `批量操作` 的方向思考

把能够在一次操作就能做完的事情，一次性做完

我们知道，会先抽 na，再抽 nb，最后才抽 no

一个回合抽 k 张卡

那么我们把 n_a 全部抽完需要 n_a/k 个回合

当然不能够保证一定能够整除，即这个回合会把 n_a 全部抽完，并且还能继续抽，那么剩下的 n_a 我们可以单独处理，计算还能抽多少张牌，批量抽取 n_b

处理完 n_a，我们用同样的方法处理 n_b

一遍处理一遍统计使用了多少个回合

算法

前缀和

二分

二分搜索

二分答案