Complete Search

The Idea

Solving a problem using complete search is based on the ``Keep It Simple, Stupid'' principle. The goal of solving contest problems is to write programs that work in the time allowed, whether or not there is a faster algorithm.

Complete search exploits the brute force, straight-forward, try-them-all method of finding the answer. This method should almost always be the first algorithm/solution you consider. If this works within time and space constraints, then do it: it's easy to code and usually easy to debug. This means you'll have more time to work on all the hard problems, where brute force doesn't work quickly enough.

In the case of a problem with only fewer than a couple million possibilities, iterate through each one of them, and see if the answer works.

Careful, Careful

Sometimes, it's not obvious that you use this methodology.

Problem: Party Lamps [IOI 98]

You are given N lamps and four switches. The first switch toggles all lamps, the second the even lamps, the third the odd lamps, and last switch toggles lamps 1, 4, 7, 10, ...

Given the number of lamps, *N*, the number of button presses made (up to 10,000), and the state of some of the lamps (e.g., lamp 7 is off), output all the possible states the lamps could be in.

Naively, for each button press, you have to try 4 possibilities, for a total of 4^{10000} (about 10^{6020}), which means there's no way you could do complete search (this particular algorithm would exploit recursion).

Noticing that the order of the button presses does not matter gets this number down to about 10000^4 (about 10^{16}), still too big to completely search (but certainly closer by a factor of over 10^{6000}).

However, pressing a button twice is the same as pressing the button no times, so all you really have to check is pressing each button either 0 or 1 times. That's only $2^4 = 16$ possibilities, surely a number of iterations solvable within the time limit.

Problem 3: The Clocks [IOI 94]

A group of nine clocks inhabits a 3 x 3 grid; each is set to 12:00, 3:00, 6:00, or 9:00. Your goal is to manipulate them all to read 12:00. Unfortunately, the only way you can manipulate the clocks is by one of nine different types of move, each one of which rotates a certain subset of the clocks 90 degrees clockwise.

Find the shortest sequence of moves which returns all the clocks to 12:00.

The ``obvious'' thing to do is a recursive solution, which checks to see if there is a solution of 1 move, 2 moves, etc. until it finds a solution. This would take 9^k time, where k is the number of moves. Since k might be fairly large, this is not going to run with reasonable time constraints.

Note that the order of the moves does not matter. This reduces the time down to k^9 , which isn't enough of an improvement.

However, since doing each move 4 times is the same as doing it no times, you know that no move will be done more than 3 times. Thus, there are only 4^9 possibilities, which is only 262,072, which, given the rule of thumb for run-time of more than 10,000,000 operations in a second, should work in time. The brute-force solution, given this insight, is perfectly adequate.

Sample Problems

Milking Cows [USACO 1996 Competition Round]

Given a cow milking schedule (Farmer A milks from time 300 to time 1000, Farmer B from 700 to 1200, etc.), calculate

- The longest time interval in which at least one cow was being milked
- The longest time interval in which no cow is being milked

Perfect Cows & Perfect Cow Cousins [USACO 1995 Final Round]

A perfect number is one in which the sum of the proper divisors add up to the number. For example, 28 = 1 + 2 + 4 + 7 + 14. A perfect pair is a pair of numbers such that the sum of the proper divisor of each one adds up to the other. There are, of course, longer perfect sets, such that the sum of the divisors of the first add up to the second, the second's divisors to the third, etc., until the sum of the last's proper divisors add up to the first number.

Each cow in Farmer John's ranch is assigned a serial number. from 1 to 32000. A perfect cow is one which has a perfect number as its serial. A group of cows is a set of perfect cow cousins if their serial numbers form a perfect set. Find all perfect cows and perfect cow cousins.

Complete Search 枚举搜索

译 by Lucky Crazy

思想：

写枚举搜索时应遵循 KISS 原则（Keep it simple stupid，译为“写最单纯愚蠢的程序”，意思是应把程序写得尽量简洁），竞赛时写程序的最终目标就是在限制时间内求出解，而不需太在意否还有更快的算法。

枚举搜索具有强大的力量，他用直接面向答案并尝试所有方案的方法发现答案。这种算法几乎总是解题时你第一个想到的方法。如果它能在规定的时间与空间限制内找出解，那么它常常很容易编写与调试。这就意味着你可以有时间去解答其他难题，即那些不能显示枚举算法强大的题目。

如果你面对一道可能状态小于两百万的题目，那么你可以考虑使用枚举搜索。尝试所有的状态，看看它们是否可行。

小心！小心！

有时，题目不会直接要求你使用枚举算法。

例题 1：派对灯 [IOI 98]

在一次 IOI 派对上有 N 个灯和 4 个灯开光,第一个开关可以使所有灯改变状态(关上开着的灯，开启关着的灯），第二个开关可以改变所有偶数位上灯的状态，第三个开关可以改变所有奇数位上灯的状态，第四个开关控制着灯 1、4、7、10……（3n+1）。

告诉你 N 的值和所有按开关的次数（小于 10,000 次）。并告诉你某些灯的状态（例如：7 号灯是关着的，10 号灯是开着的）请编程输出所有灯可能的最后状态。

很明显，每次按开关你都要偿试 4 种可能。那么总共要试 410000 次（大约 106020），那意味着你没有足够的时间去使用枚举搜索，但是我们将枚举方法改进一下，那就可以使用枚举了。因为无论有多少个灯，由于开关控制的特殊性，都会出现 6 个灯一次循环的情况，即 1 号灯的状态永远与 7 号灯，13 号灯，19 号灯……相同，2 号灯的状态也永远与 8 号灯，14 号灯，20 号灯……相同。同样，无论你按了多少次开关，按同一个开关两次就相当于没有按该开关，那么每一个开关就只需要考虑按一次或没有按，那么这题的枚举量就很小了。

例题 2：时钟调整 [IOI 94]

有九个钟被摆放在一个 3 X 3 的矩阵中，它们各自指向 12：00，9：00，6：00，3：00 中的一种，你的目的是将它们的指针全部调向 12：00。很遗憾，每一次调整你都只能从九种调整方案中选择一种执行（九种方案已被从 1 到 9 编号），每一种方案可以改变固定钟的状态（例如：方案 1 控制钟 1，2，3，方案 2 控制钟 1，4，7，方案 3 控制钟 5，6，8，9……），即将方案指定的所有钟向前拨快 3 小时（使时针向顺时针方向旋转 90 度），请你输出一个数列，使得按该数列表示方案执行后，所有钟都指向 12：00。并且如果把整个序列看作一个数，要求该数最小。

最容易想到的方法是用递归枚举 1 到 9 的方案在该步使用。很可怕，由于递归的层数在此没有限定，所以将用掉 9k 的时间（k 为层数），那可能是相当巨大的。其实，不用紧张，细心的你一定会发现：当一个方案执行 4 次后，就相当于没有执行，又因为题目要求输出最优解，那么任意一个方案都没有必要 4 次以上执行。也就是说，只需要枚举每一个方案的 4 种情况（没执行，执行 1，2，3 次）就可以了。他仅有 49 ，约 262,072 此枚举，我们的计算机 1s 内就可以算完，应该算是极快了。

类似问题：

挤牛奶 [USACO 1996 初赛]

给出一个挤牛奶的顺序（农夫 A 在 300 秒到 1000 秒时挤牛奶，农夫 B 从 700 秒到 1200 秒），要求输出：

最长的有人挤牛奶的时间。

最长的没人挤牛奶的时间。

完全数牛与完全数牛群 [USACO 1995 决赛]

如果一个数可以由它的某几个约数相加得到，那么我们叫它完全数，如 28 = 1 + 2 + 4 + 7 + 14。而一对完全对数就是指两个数都可以由对方的约数相加得出。同样一个完全数组就是一个数组的第一个数可以由第二个数的约数加和得到，第二个数也可以由第三个数的约数相加得到……最后一个数可以由第一个数的约数加和得到。

现在 Farmer John 已经将它的牛儿们编好了号（1 到 32000）请找出其中所有的完全数牛与完全数牛群。