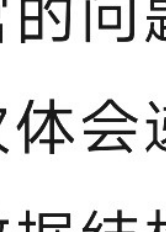


第172封信 | 递归算法的扩展以及它们和递归的关系




递归的逻辑，以及堆栈这种

一大类问题——

当然，在此

准确的数学表述，以及在工程上解决这些问题和理论上有何不同。

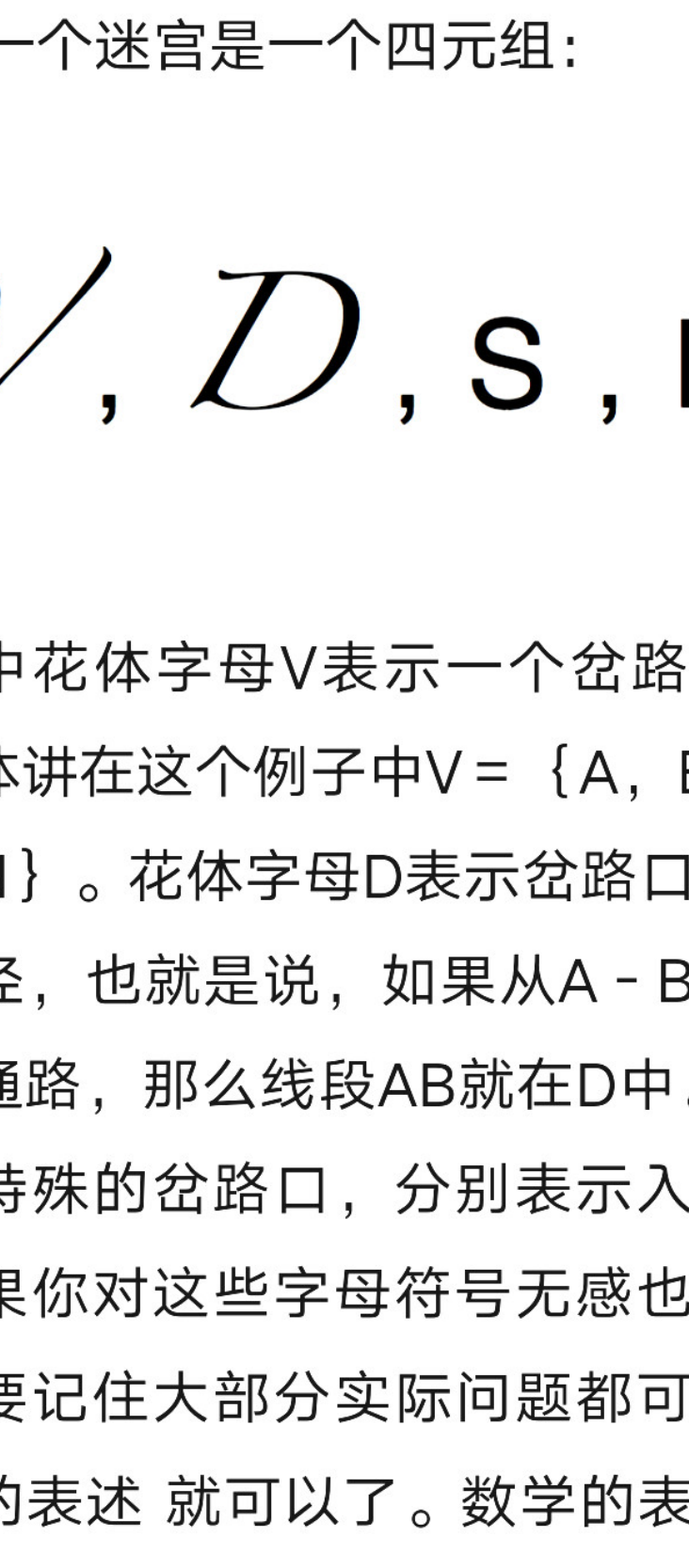
你可以看一眼下面这张图，就是前两天举例子用的迷宫图。



The diagram shows a maze with a green dot at the top left labeled '入口' (Entrance). Below the entrance, there are two paths labeled 'A' and 'B'. Path A leads to a dead end, and Path B leads to a dead end. The maze is composed of black lines on a white background.

A diagram of a rectangular room with a door on the left wall. The room is divided into several sections by lines. The sections are labeled with letters: G (top left), N (top middle), K (top right), M (middle left), L (middle right), D (bottom right), F (bottom middle), and E (bottom right). The door is on the left wall, between sections G and M.

我把这个迷宫问题用比较严格的数学方法表示如下：



示都是。所谓职业化，就是要用职业的语言表述问题。

- 比如P可以等于（B到C，C

3. 我们的问题是找一个特殊的路径，起点为S，终点为E，如果找到了，就输出这条路径，如果没有找到，就转

为什么要把这个问题数学化呢？一来把问题的已知条件和所需要完成的目标搞清楚，二来方便大家在同一个共识上交流和一起工作。学习数学主要的目的是能够抽象化问题，而不是做加减乘除。

今天加减乘除早交给了计算机去完成，而抽象化问题的能力依然属于人类自身。没有这个能力，当计算机出现后，会算简单算术的人就失业了，但是能够抽象化问题，并且解决问题的人就因为有了计算机就如虎添翼了。

明确了问题，接下来就要解决问题。

- 宫的想法用下面比较准确的语言表述一下。学过计算机的人其实很容易把这段迁

1. 在当前的岔路口，用左手原则，顺时针寻找下一个岔路口并由此路行。

如果能找到，则沿着下一个路径走到相邻的岔路口。比如在D这个位置，DE已经走过了，DL就是下一个还没有走的路径。沿着这个路径，就走到了L这个岔路口。

如果已经不存在还没有走过的路径，那么就在这个岔路口放一颗豆子，表示已经来过，此路不通，以后不要试了。然后返回到上一个路口。比如我们在D这个位

置，DE和DL都走过了，这时，就在D放一颗豆子，然后回到上一个路口，比如说是C。

2. 如果当前的岔路口是E，即出口，就成功了。打印相应的路径就好。如果当前的岔路口已经有了一颗豆子，说明走到了原来失败的地点，就不要再试验了，原

3. 当从某一条路径返回后，把这条路径打上叉叉，表示已经尝试过了，走不通，下次可以从紧挨着这条路径的右边一条再试验。比如从D回到了C，把CD打个叉子，下次就从CD右边的CJ接着往前试。

- 在计算机算法中，把上述1，2，3写成一个模块，在有的编程语言中这种模块被称为函数，有些被称为过程，反正都是一回事。这个模块可以来回来去地使用，

最后，只要从入口开始调用“顺时针探路”这个过程即可。如果一直没有走通，这个过程执行一遍会回到起点S，那就失败了。

把上面这个描述变成计算机的程序语言，这个程序就写好了。

上述算法整个的结构是递归的，也就是说，一开始在入口S调用“顺时针探路”这个过程，在这个过程里面，走到A这个岔口时，从A开始继续调用“顺时针探路”这个过程.....直到找到路径或者不断返回，直到失败。这就如同我们小时候听到的一个故事，从前有个山，山里有个庙，

- 事呢，“从前有个山……”。
- 所不同的是，上述算法比那个古老的故事多了三个条件。

1. 如果讲到某一次的时候，有一个出口能出去，故事就讲完了。

3. 如果返回到开头还没有看到出口，故事就失败了。

一些学习计算机的读者问，递归的做法是否效率低？其实递归更多地是一种思维方式，真到了需要实现的时候，其实有办法规避递归的低效率问题。

过来的岔路口记在一个本子上，从上到下，一行一个写清楚。如果走到死胡同，往回退，每退一步，就把相应的岔路口从

本子上划掉。就这样，遇到新的就写上，退回去就划掉，这其实就自动地实现了递归功能，并不复杂。

当然，在程序中并不要使用小本子，而是使用我写给你的[第98封来信中说过的堆栈这种数据结构](#)。在任何时候遇到一个难题，先想办法找到争取的解决方法，至于解决方案中涉及到效率的问题，第二步再想想有没有办法提高效率。

接下来，我们再用这个例子讲一次科

如果你在大学学计算机课程，老师出了这道题，只要你把上面我们讨论的想法

- 会得到A。
- 但是，如果你把同样的程序不作丝毫修改，作为产品的一部分提交给公司，相

- 难检查的，因此你需要增加很多冗余的信息。一方面帮助出错的时候查错，另一方面不断进行验证，以免出错。比如每一个字节的校验码只用在出错的，并且出错的概率

- 虽然我们总是力图将算法设计得无

不需要这些坐标信息。但是，绝对无错。

如果程序在递归迭代时出错了，把岔路口的次序搞乱了，通过坐标很容易发现这件事。

里面不仅包含执行相应功能的代码，而且包含很多测试和调试的代码，以免程序出错，找不到原因。通常，调试程序的时间是写程序的三四倍，甚至更多，因此，多花30%的时间写调试代码非常重要。类似地，商用的、性能可靠的半导体芯片设计，里面有大量的测试电路，它们不提供任何实用的功能，只是为了测试芯片的功