信息学竞赛中搜索问题的常见优化技巧

重庆一中　黄晓愉

【**摘要**】结合例题分析归纳了信息学竞赛中解决搜索问题所常用的思考方法与解题方法，从深度优先搜索和广度优先搜索两个方面探讨了提高程序效率的适用技巧。

【**关键词**】1 信息学； 2 搜索顺序； 3 搜索对象； 4 Hash 表 5 剪枝。

在信息学竞赛中解决搜索问题通常采用两种方法进行，即：深度优先搜索和广度优先搜索。

一、深度优先搜索的优化技巧

我们在做题的时候，经常遇到这类题目——给出约束条件，求一种满足约束条件的方案，这类问题我们叫它“约束满足”问题。对于约束满足问题，我们通常可以从搜索的顺序和搜索的对象入手，进而提高程序的效率。

搜索的顺序及对象：

在解决约束满足问题的时候，题目给出的约束条件越强，对于搜索中的剪枝就越有利。之所以深度优先搜索的效率在很大程度上优于穷举，就是因为它在搜索过程中很好的利用了题目中的约束条件进行剪枝，达到提高程序效率的目的。

显然，在同样的一棵搜索树中，越在接近根接点的位置利用约束条件剪枝效果就越好。如何在搜索中最大化的利用题目的约束条件为我们提供剪枝的依据，是提高深度优先搜索效率的一个很重要的地方。而不同的搜索顺序和搜索对象就直接影响到我们对于题目约束条件的运用。

下面，我们就从搜索的顺序和搜索的对象两方面来探讨一下不同的方法对程序效率的影响。

（1）搜索顺序的选择：

我们先来看一道比较简单的题目： **(zju1937)**

已知一个数列 a0,a1……am 其中

a0 = 1

am = n

a0 < a1 < a2 < … < am-1 < am

对于每个 k(1<=k<=m),ak=ai+aj (0 <= i, j <= k-1)，这里 i 与 j 可以相等。

现给定 n 的值，要求 m 的最小值（并不要求输出）,及这个数列的值(可能存在多个数列,只输出任一个满足条件的就可以了)。

分析 由于 $a_k=a_i+a_j(0<=i,j<k)$,所以我们在搜索的过程中可以采用由小到大搜索数列的每一项的搜索顺序进行试算。在一般搜索的时候我们习惯于从小到大依次搜索每个数的取值，但是在这到题目中按照这样的顺序搜索编程运算其结果（效率）十分不理想：

N	10	20	30	40	50	60	70	80	90	100	200	300	400	500
用时	0.03	0.01	0.03	0.05	0.20	0.34	1.80	1.80	8.91	10.1	Too long	Too long	Too long	Too long

由于题目要求的是 m 的最小值，也就是需要我们尽快得到数 n，所以每次构造的数应当是尽可能大的数，根据题目的这个特性，我们将搜索顺序改为从大到小搜索每个数，新程序的效率如下：

N	10	20	30	40	50	60	70	80	90	100	200	300	400	500
用时	0.01	0.01	0.01	0.01	0.01	0.01	0.03	0.01	0.03	0.03	0.13	1.48	1.5	22.88

显然，后一种搜索顺序得到的程序效率大大地优于第一种搜索顺序得到的程序。

当然，这道题还有很大的优化余地，但是搜索顺序这种思想在搜索的题目中是广泛运用的。也许大家会觉得这种单一的运用搜索顺序来优化程序的方法很普通，但是

这种看似简单的方法在考试中出现得也不少，例如 IOI2000 中的 BLOCK，只要将木块从大到小经过旋转和反转后，依次放入进行搜索，对于比赛中的数据就可以得到满分。最近的一次出现是 NOI2005 中的智慧珠，同样的只是将珠子从大到小进行搜索，不加任何其他剪枝就可以在比赛中获得 90 分。

可见，选择合适的搜索顺序对于提高程序的效率是编程设计最有效的技巧之一，运用良好的搜索顺序来对搜索题目进行优化是一个性价比很高的算法。

（2）搜索对象的选择：

让我们再来看看下面一道题：**(USACO－weight)**

已知原数列 a_1，a_2……a_n 中前 1 项，前 2 项，前 3 项……前 n 项的和,以及后 1 项,后 2 项,后 3 项……后 n 项的和，但是所有的数据都已经被打乱了顺序，还知道数列中的数存在集合 S 中，求原数列。当存在多组可能数列的时候求左边的数最小的数列。

其中 n<=1000,S∈{1..500}

例如，假如原数列为 1　1　5　2　5，S={1，2，4，5}那么知道的值就是 (1　2　7　9　14　5　7　12　13　14)

1 = 1　　5 = 5

2 = 1+1　　7 = 2+5

7 = 1+1+5　　12 = 5+2+5

9 = 1+1+5+2　　13 = 1+5+2+5

14 = 1+1+5+2+5　　14 = 1+1+5+2+5

分析　因为题目中的 S ∈{1..500}，最坏的情况下每个数可以取到的值有 500 种，从数学方面很难找到有较好方法予以解决，而采用搜索方法却是一种很好的解决办法，根据数列从左往右依次搜索原数列每个数可能的值，然后与所知道的值进行比较。这样，我们得到了一个最简单的**搜索方法 A**。

但是搜索方法 A 的这个算法最坏的情况下扩展的节点为 500^{1000}，运算速度太慢了。

在这个算法中，我们对数列中的每个数分别进行了 500 次搜索，由此导致了搜索量如此之大。如何有效的减少搜索量是提高本题算法效率的关键。而前面提到的运用搜索顺序的方法在本题中由于规定了左边的数最小而无法运用。让我们换个角度对这个问题进行思考：

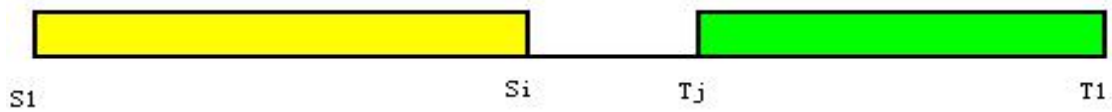
搜索方法 B：回过头来看看题目提供给我们的约束条件，我们用 S_i 表示前 I 项的和，用 T_i 表示后 I 项的和。

根据题目，我们得到的数据应该是数列中的 $S_1,S_2,S_3\cdots\cdots S_n$，以及 $T_1,T_2,T_3\cdots\cdots T_n$。其中的任意 $S_{i+1}-S_i$ 和 $T_{i+1}-T_i$ 都属于集合 S。另一个比较容易发现的约束条件是对于任意的 I，有 Sn=Tn=Si+Tn-I。同样的，在搜索的过程中最大化这些约束条件是提高程序效率的关键。

那么当我们任意从已知的数据中取出两个数的时候，只会出现两种情况：

1、两个数同属于 S_i,或者 T_i

2、两数分别属于 T_i 和 S_i。



当两数同属于 S_i 或者 T_i 时，两个数之差，就是图中 Sj-Si 那一段，而当 j=I+1 时，Sj-Si 必然属于题目给出的集合 S。由此，当每次得到一个数 Si 或者 Ti 时，如果我们已知 S_{i-1} 或者 T_{i-1}，便能够判断出此时的 S_i 或者 T_i 是否合法。所以我们在搜索中尽可能利用 S_{i-1} 和 T_{i-1} 推得 S_i 和 T_i 的可能，便能尽可能利用题目的约束条件。

因为题目的约束条件集中在 S_i 和 T_i 中，我们改变搜索的对象，不再搜索原数列中每个数的值，而是搜索给出的数中出现在 S_i 或者 T_i 中的位置。又由于约束条件中得出的 S_{i+1} 与 S_i 的约束关系，提示我们在搜索中按照 S_i 中 i 递增或者递减的顺序进行

搜索。

例如，对于数据组：1 1 5 2 5，由它得到的值为

1 2 7 9 14 5 7 12 13 14

排序后为：

1 2 5 7 7 9 12 13 14 14

由于最大的两个数为所有数的和，在搜索中不用考虑它们，去掉 14：

1 2 5 7 7 9 12 13

观察发现，数列中的最小数 1，只可能出现在所求数列的头部或者尾部。再假设 1 的位置已经得到了，去掉它以后，我们再观察剩下的数中最小的数 2，显然也只可能在当前状态的头部或者尾部加上一个数得到 2。这样，每搜索一个数，都只会将它放在头部和尾部，也就是放入 S_i 中或者 T_i 中。

推而广之，我们由小到大对排序的数进行搜索，判断每个数是出现在原数列头部还是尾部。此时我们由原数列的两头向中间搜索，而不是先前的从一头搜向另一头。由之前的分析已经知道，每个数只可能属于 S_i 和 T_i 中。当我们已经搜索出原数列的 $S_1,S_2…S_i$ 和 $T_1,T_2…T_j$，此时对于正在搜索的数 K,只可能有两种存在的可能：S_{i+1} 和 T_{j+1}，分别依次搜索这两个可能，即判断 $K-S_i$ 和 $K-T_j$ 是否属于已知集合 S。并且在每搜索出一个数 K 的时候，我们将排序后的数列中 S_n-k 去掉。这样，当 $K-S_i(T_i)$不属于集合 S 或者 S_n-k 不存在与排序后的数列时，就回溯。

这样得到的算法在最坏情况下扩展的节点为 2^{1000}(实际中远远小于这个数)，并且由于在搜索过程中充分利用了题目约束条件，其程序运行结果如下：

在这道题目中，原始的搜索方法搜索量巨大，我们通过分析，选择适当的搜索对象，在搜索量减少的同时充分利用了题目的约束条件，成为了程序的一个有利的剪枝，使题目得到较好的解决。

二、广度优先搜索的优化技巧

相对于深度优先搜索的另外一类题目——给出起始和目标状态，以及状态转移的规则，要求找到一条到达目标状态的的路径或者方法。这类问题我们叫它路径寻找问

题(例如走迷宫问题)。解决这类问题最有效的手段是选取合适的构造 Hash 表的方法。

Hash 表的一般构造方法有：

状态压缩-------运用 2 进制来记录状态。

直接取余法-----选取一个素数 M 作为除数。

平方取中法-----计算关键值平方，再取中间 r 位形成一个大小为 2^r 的表。

折叠法---------把所有字符的 ASCII 码加起来。

路径寻找问题中，经常会遇到走回头路的问题，所以在搜索的过程中都必须做一件事，就是判重。判重是决定程序效率的关键，而如何构造一个优秀的 Hash 表决定着这一切。一个好的 Hash 函数可以很大程度上提高程序的整体时间效率和空间效率。

(zju1301)：

黑先生新买了一栋别墅，可是里面的电灯线路的连接是很混乱的(每个房间的开关可能控制其他房间,房间数<=10)，有一天晚上他回家时发现所有的灯（除了他出发的房间）都是关闭的，而他想回卧室去休息。可是很不幸，他十分怕黑，因此他不会走入任何关着灯的房间，于是请你帮他找出一条路使他既能回到卧室又能关闭除卧室以外的所有灯。如果同时有好几条路线的话，请输出最短的路线。

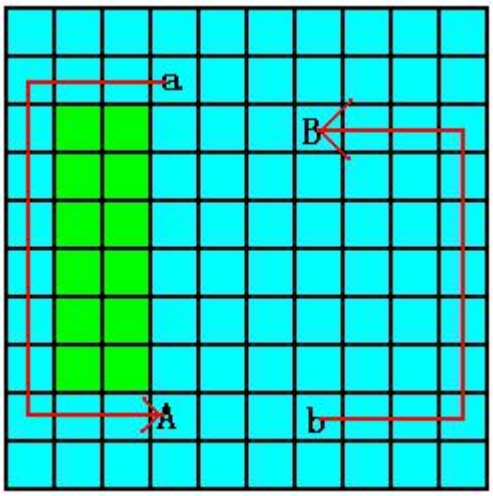
分析　这是一道比较简单的搜索题目，题目要求是一条路径，所以我们用广度优先搜索来解决。广度优先搜索不能避免的是重复状态，而用循环判断重复是得不偿失的，在状态多的情况下，循环法甚至比深度优先搜索的效率更低，而且低得多。而题目的难点在于 Hash 表的构造，经过分析发现，对于状态有影响的便是房间内电灯的开关与否，还有当时所在的房间。由于电灯只有开和关两种情况，我们考虑用 2 进制来储存状态，也就是大家熟悉的状态压缩。

将每个房间中电灯的状态用 0 和 1 来表示，然后将 10 个房间的状态排列起来就成了 1000100101 这样的形式。然后将他转换成 10 进制$(1000100101)_2=(549)_{10}$，这样一来就可以为唯一的表示出一个电灯开关的状态,再用一个数记录下黑先生当时所

在的房间，就成功地构造出了所需的 Hash 表。总共的状态数为 $2^{10}*10=10240$。

同时，在搜索中可以用位运算来判断某个房间的状态，使得 Hash 表的填充和查找变得简单。例如，假设当前状态为 K，现在要判断第 I 个房间的状态。只需(2^{i-1} and K)是否为 0 就行了。这样一来，这道题就已经解决了。

(pku1729)



在一个 N*N(N<=30)的地图上，有 A 和 B 两个人，地图上的一些地方为空地，一些地方有障碍不能通过。在每一个时刻 A 和 B 必须向四个方向移动（‘N’,’E’,’W’,’S’），并且 AB 两人彼此特别讨厌对方，他们希望在移动的时候尽可能的离对方远，现在知道两个人分别的起点和终点。求出一条使 AB 到达终点的路径，并且在途中 AB 间最近的距离最远，在此基础上使 AB 尽快到达终点。如图为 N=10 时的一种情况。

分析：本题是求路径的一道题，所以是一道很明显的广度优先搜索题目，题目的条件很多：首先是要 AB 都到达终点，然后是要路径中 AB 离得尽可能的远，同时 AB 要尽快到达。

我们首先尝试用 Hash 表将所有的信息存下，然后进行搜索，我首先想到了两种构造 Hash 表的方法：

1、 用 Hash[x1,y1,x2,y2,t]表示经过了 t 个时间点，A 到达坐标(x1,y1)，B 到达(x2,y2)，它们在途中的最近距离。

2、 用 Hash[x1,y1,x2,y2,k]当 A 到达坐标(x1,y1)，B 到达坐标(x2,y2)，它们在途中的最近距离为 k 时的最少时间。

这两种方法构造方法共同的特点是 Hash 表中储存了大量的信息，并且在编程实现中比较困难。

由于大量的条件在 Hash 表中堆积，我们尝试将其中的一个条件从 Hash 表中去掉，用其他的方法来分担。

假设我们已经知道了两人在途中的最近距离，那么剩下的将会简单许多。但是怎样才能知道两人在途中的最短距离呢？我们想到了二分。

在两个人在地图上的距离差最多只可能有 $30^4 = 810000$ 种可能，如果我们采用 2 分法，最多只需要 $\log_2 810000 < 20$ 次，然后在用广度优先搜索来解决剩下的问题，那么最坏的情况下扩展的节点数为 $20*30^4=16200000$。完全可以在规定的时间内得出结果。

考虑到在 AB 移动的过程中，同一个状态(x1,y1,x2,y2)不可能出现两次，那么可以考虑直接用 Hash 表将状态(x1,y1,x2,y2)的情况存下，于是 Hash 表中应该储存下 A 到达坐标(x1,y1)，B 到达坐标(x2,y2)，它们途中的最近距离并且在此基础上的最短时间。

小结 Hash 表是非常重要的广度优先搜索优化方式之一，它能够把搜索算法的效率从大指数级提高到小指数级、多项式级甚至常数级。

三、深度优先搜索和广度优先搜索的有力工具----剪枝

USACO-Cryptcowgraphy(解密牛语)

农民 Brown 和 John 的牛们计划协同逃出它们各自的农场。它们设计了一种加密方法用来保护它们的通讯不被他人知道。

如果一头牛有信息要加密，比如"International Olympiad in Informatics"，它会随机地把 C，O，W 三个字母插到到信息中（其中 C 在 O 前面，O 在 W 前面），然后它把 C 与 O 之间的文字和 O 与 W 之间的文字的位置换过来。这里是两个例子：

International Olympiad in Informatics->CnOIWternational Olympiad in Informatics

International Olympiad in Informatics-> International Cin InformaticsOOlympiad W

为了使解密更复杂，牛们会在一条消息里多次采用这个加密方法（把上次加密的结果再进行加密）。一天夜里，John 的牛们收到了一条经过多次加密的信息。请你写一个程序判断它是不是这条信息经过加密（或没有加密）而得到的：

Begin the Escape execution at the Break of Dawn

分析 基于密码编译规则，我们很容易地可以想出一个非常简单的 dfs 方法，当然，那是明显要超时的，而分析题目我们可以发现，题目要求的是一种得到信息的加密方法，也就是求的一种加密的路径。于是我们不得不采用宽度优先搜索算法。

对于 Hash 表的构造方法，可以采用 ELFhash 函数或者 SDBMhash(参见 05 年李羽修论文)。这里跳过。

题目规定加密信息不超过 75 个字符，而原始的信息一共有 47 个字符，那么按照正规的方式在信息中最多可能加入 9 组'C','O','W'字符。如果使用原始的算法进行搜索，将最多扩展$(9!)^3$个接点，大大超过了时间的允许，所以剪枝是必要的行为。

题目已经将原始的信息提供给了我们，所以如何利用好已知的信息对搜索进行剪枝将直接影响程序的效率。

1、密文的每次加密都会加入'C','O','W'三个字母，所以输入的密文长度必须是 47+3n，这样可以迅速的排除一些数据。

2、已知密文中的每个字符的个数除了'C',"O,'W'外必定都是固定的，而且'C','O','W'三个字符的个数也必定在原始信息的基础上增加了(n-47)/3 (n 为输入信息长度)。

3、原文中出现的'C','O','W'均为小写，那么在加密后的信息里面，连续的'C','O'和"O,'W'之间的字符必定是原文的序列。

4、第一个字符和加密字符('C')之间必须是原文，同样的，最后一个字符和最后一个加密字符('W')之间也必须是原文。

在程序的实现中，为了更快了搜索，将把字符串里面的加密字符，按索引存储。这样每次扩展节点的时间大大减少，最坏情况为 9^3。否则每次扩展节点将花去 $(length(s)^3)$。

使用了这些优化以后，程序的效率将大大的提高。

小结 剪枝是所有搜索中必不可少的一个重点，充分的利用题目的特性进行剪枝可以很大程度的提高程序的效率。

参考文献：

[1] 刘汝佳，黄亮. 《算法艺术与信息学竞赛》(2004)

[2] AngleForYou 《搜索算法的通用优化方法》

[3] USACO 《解密牛语解题报告》