由图论问题浅析算法优化

武钢三中 贾由

【摘要】

论文以图论问题为对象、以算法优化为主题、以分类和举例为基本模式进行了一系列探讨。第一部分引言简单地介绍了图论与信息学竞赛的关系；第二部分分析了算法优化的根本途径：寻找特别之处；第三部分从算法的纠错入手，详细讨论其中的方法，进一步展示了发现问题的特殊点对算法优化的推动作用。

【关键字】

图论 算法优化 错误分析

【正文】

一、引言

图论是一个十分有趣而且与信息学竞赛联系紧密的数学分支。随着图论问题的日渐增多，一些经典图论模型与它们的相关算法已成为竞赛中不可或缺的知识。与此同时，题目也越来越注重模型的转换与算法的优化。这意味着在将知识转变为分数的过程中，我们需要做出更多的努力。本文以其中的算法优化为主题，尝试了一些相关的归纳与讨论。

另外，由于黑箱测试的缘故，我们所体验到的信息学可以说是一个以结果论成败的学科。这是很好的，因为结果是对历史的总结。但无论如何，对于一次以优化为主题的讨论来说，得到的最优算法仅仅是用来证明我们的优化过程是切实而有效的。

二、寻找特别之处——优化的根本途径

2.1 介绍

每一个让算法更加漂亮的改进都可以称为优化。不过在整体考虑一个问题时，优化的过程应该包括从原始算法到一个优秀算法当中的所有改进。这通常是一个逐步发现并利用问题的特殊之处、使算法更有针对性的过程。

做好优化的根本在于找出题目的特别之处。这是一个宽泛的想法，没什么步骤和诀窍可言。解决具体问题时，我们只能靠广泛的优化经验、充足的耐心以及一部分的灵感因素。关于经验，之前的几篇论文已经分别就一些有共同特征的题目介绍了深入挖掘信息的具体过程。这一章不再深入探讨某类问题，而是通过一个经典算法对“寻找特别之处”作出解释。

2.2 例题

[例] 二分图的最大匹配[1]

图的匹配指图中任何两条边都没有共同顶点的子图，二分图最大匹配问题旨在求出二分图中边数最多的一个匹配。求解这个问题最基本的方法是将它转换成一个网络流模型：

为了方便叙述，我们将二分图的两个顶点集合成为A和B。在图中加入源点和汇点，从源点向A中的每个点引一条边，容量为1；从B中的每个点向汇点引一条容量同样为1的边。然后将原图中的边作为有向边添加进来，由A指向B，容量为1。新图中用容量限制了每个点最多只能被一条边覆盖、每条边只能被记一次。容易看出，这个图的最大流与最大匹配中的边数相等。通过最大流算法，我们同样可以得到选边的具体方案。

最显然的一个优化是不记录容量，所有边的容量都是一。其次，这个网络中的可增广链很特别，它一定由源点开始，在点集A和点集B之间做若干次往返，再由B到达汇点的。搜索时可以不考虑第一条与源点连接的边和最后一条与汇点连接的边，直接从点集A中的一个未匹配点开始到点集B中的一个未匹配点结束。可以想象，在广搜的目标路径中减少两条边对于需要扩展的结点数的影响是巨大的。这就是匈牙利算法的基本思想。

基本的网络流算法与匈牙利算法的时间复杂度其实没有区别[2]，但是后者在所需空间、编写难度以及实际的运行时间上都拥有绝对的优势。

几乎可以肯定匈牙利算法最初不是由上面这样的优化得到的，但这两种优化手段在网络流算法的设计中是很实用的：根据图的特殊性简化存储方式、量身定制搜索可增广链的方法。

[例] 牧场规划[3]

小可可的好朋友Sealock最喜欢吃花生了，于是借用了小可可的牧场从事花生选种试验。他以网格的方式，非常规整地把牧场分割成M*N个矩形区域(M*N≤5000)，由于各个区域中地水面、沼泽面积各不相同，因此各区域地实际可种植面积也各不相同，已知区域(i,j)地可种面积使A(i,j)。

每个区域种最多只能种植一个品种地花生。可种植面积为零地区域不能被选择用来从事选种试验，同时为了防止花粉传播到相邻区域造成试验结果不正确，任何两个相邻的区域都不可以同时种植花生。这里说的相邻指的是两个区域有公共边，仅仅有公共点的两个区域不算做相邻。

小可可准备帮助Sealock规划一下如何选择种植区域，才能使得实际可种植面积总和最大。

对应选择方案与网络的割

建立方案与网络的割之间一一对应关系的方法在[4]中曾有所描述，我们根据这道题再回顾一遍。

---

[1] 题目来源：经典问题。

[2] 都是O(M*N)，更快的算法可以参考[2]。

[3] 题目来源：2003年安徽省省选。

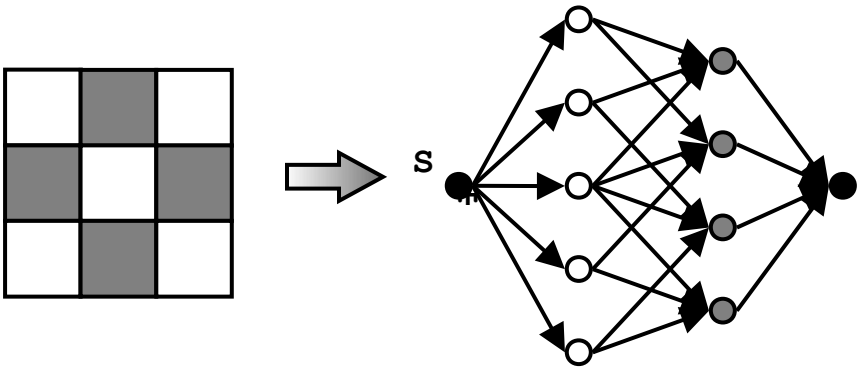


图1　建立网络

将试验田转化为点、并连接相邻的试验田后可以发现，我们得到的是一个二分图。通过对原图的黑白染色，可以把其中的一部分称为白点、另一部分称为黑点。由二分图建立网络：加入源点和汇点，从源点向每个白点引一条边，容量为白点对应试验田的面积；从每个黑点向汇点引边，容量为该黑点的对应面积。最后将相邻点之间的边改为网络中的边，由白点指向黑点，容量为正无穷。

方案对应的割：将方案中所选的白点和未选的黑点再加上源点划为一个集合，其它点划到另一个几何，就得到了一个割。直接把这个过程反过来，我们很容易由割得到一个方案。在这个对应中：一、不合法方案对应的割均为正无穷；二、在合法方案对应的割中，割上的点代表了方案没有取到的边。所以当割的容量最小时、方案选取的面积最大，而根据最大流最小割定理，我们可以通过求网络最大流得到它的最小割。

广搜可增广链

这同样是由二分图转换来的网络，但是边的容量一般化了。我们仍然可以不搜索首尾两条边，不同的是以某一个“新源点”（原可增广链上的第二个点）为起点的广搜可能要进行多次，次数最多等于源点到它的边的容量；同理，一个“新汇点”可以容纳多个可增广链。另外，为白点和黑点分别设计扩展过程也可以大大提高算法的效率。

三、改进错误算法——更灵活的优化

3.1 介绍

我们常说的算法优化有四个方向：时间复杂度的优化、空间复杂度的优化、编写难度的优化以及思维难度的优化。但是正如标题所表达的，这一部分的内容是如何提高算法的正确率。

纠错也算是优化吗？如果你也有同样的疑问，那你一定是想到代码的查错上去了。提高算法的正确率当然是对算法的优化。甚至，算法的错误常常也是由于对题目信息的不充分利用导致的，只不过除此之外还有很多别的原因，我们一会儿就会进一步分析到它们。

应对错误需要有一套方法。首先，我们总会希望错误不要出现，比如思维严谨一点、看问题全面一点。当问题不可避免地出现时，处理方法一定要视情况而定：如果考试的时间已

经不多，可以通过一些简单的处理适当地提高正确率；如果还不紧张，就应该仔细地分析分析错误；实在无能为力的话，放弃已有的算法另寻他解也是一个明智的选择。

应急时的简单处理虽是无奈之举，却也值得一提。最直接的办法是加入额外的判断过程，尽量把想到的反例都包括进去，但在问题比较复杂时，这通常是一件让人头疼而且收获不大的工作。另一方面，随机化加重复求解的处理操作简单、效果惊人，更能为人们所接受。对于最优化问题，取多次运算得到的最优解即可；判定性问题稍麻烦些，原算法还得满足下面两个前提中的至少一个：一、它的正确率高于50%，于是我们可以采用得出次数较多的那个结果；二、它可以准确判断“是”、“否”中的一个方面，假如被判断为“是”时一定正确，那么就把剩下得到的全是“否”的输入判为“否”。应对具体错误时还能找出许多小技巧，这儿就不再列举了。

下面我将通过分类举例来讲述我对如何仔细分析错误的理解。

导致我们设计出一个错误算法的原因大致有三类：

一、误解模型的性质。其中的模型指所有的数学模型，之前所说的信息利用不充分也属于这一类。这是一个主流原因，更深入的讨论见下一节中古老的《渔网》问题。

二、猜想错误。不知如何将模型与算法联系起来时，我们常提倡大胆去猜想。但猜想难免出错，如何处理这类错误至关重要。在《奶牛航班》的分析过程中出现了这个问题。

三、不了解算法细节。需要适当改进一个经典算法时，如果我们对这个算法的掌握不够透彻，很容易出现问题。下面的例子，《可疑的斑点》，不是图论问题，但很有代表性，它主要涉及KMP算法。

3.2 例题

[例] 奶牛航班[1]

约翰的奶牛们开通了一条飞机航线，专门为奶牛服务。每天早上，她们沿着密歇根湖的西岸，从线路的最北端出发飞到最南端，全程经过N个机场（包括头尾两个）。到了下午，她们又会沿着同样的路线飞回最北端。每天都会有数目不同的K群奶牛要求乘坐飞机，一群牛会在某一个机场等待，并希望飞到另外一个特定的机场。

飞机上只能同时容纳C头奶牛乘客，航班的负责牛希望知道在这一天中她们最多可以满足多少头奶牛的要求。飞机可以只将一群牛中的一部分带到目的地。

约定：1≤N≤10,000, 1≤K≤50,000, 1≤C≤100。

初步分析

飞机的一去一回很明显是两个相同的问题。而且如果一头奶牛非要绕一次折返点的话，它会在原来的基础上再多在飞机上坐一段路，这并不划算。于是这两个子问题互不干扰，可以单独处理，我们接下来只需考虑一个方向上的事情。还有一个问题是我们不必多考虑的，那就是飞机在途中的哪些机场着陆。我们可以认为它在每个机场都做停留（这并不耽误什么），或者干脆把它当作一辆长途汽车。

题目给人的第一感觉是动态规划：将机场看作点、牛的飞行路线看作边，那么算法将为每个点记录下从起点飞到这里最多能服务多少头奶牛，并通过边进行状态转移。

联系到这一题的实际情况，我们会很快发现这是很荒谬的。在这样的动态规划得到的方案中，肯定不会出现有重叠关系的边，但是我们的飞机完全有可能同时满足两头路线重叠的牛的飞行要求。

[1] 题目来源：USACO 2005年十月竞赛，Flying Right。

看来这不是一个常规的动态规划题，我们只好先把动态规划的思路搁在一边。

最小费用流模型

数据规模的约定提醒了我们要好好利用飞机上座位不多（最多 100 个）这个条件。一个座位一个座位地考虑，最直接的方法莫过于网络流了。

容量网络的建立并不复杂：仍然以机场为点，牛群的飞行线路为边。每条边的容量赋为牛群中的牛数 M_i，权值都是 1。代表这个牛群最多可以“容纳” M_i 个座位，每当一个座位选择了这个群，就相应的得到 1 点权值。再从每个点（除最后一个）向它的后一个点连一条容量无穷大、权值为 0 的边，代表座位在这一段可以是空闲的。总的来说，模型中的一个单位流就代表了一个座位。

为了避免最大费用流这个怪异的名词，在实现时把权值改为 -1 即可。最小费用流的算法中，用 Bellman-Ford 算法求一次最短路需要 O(K*N)，最大流量等于座位数 C，所以算法的总复杂度是 O(K*N*C)，无法承受题目给出的数据规模。

加入贪心思想

给这个特殊的图套上最小费用流算法的确有点浪费，时间复杂度高是必然的结果。那么如何利用这些特殊点优化算法呢？

费用流算法中，每次找到一条最短路径（权和最小的路径）进行扩展。扩展时为了可能的改动，我们会给扩展边的反向边扩大容量，这个处理解决了后效性的问题。仔细想想，在这个问题中或许根本就不存在后效性，我们能不能放弃对反向边的处理？

不处理反向边使算法有了本质上的改变。它所做的相当于：逐一安排每一个座位，使每个座位在当前情况下全程载牛数最大。对此，我们不必再用 B-F 算法，动态规划可以在 O(K) 时间内得出答案。这样，总时间复杂度降到了 O(K*C)，足以满足题目所给的数据规模。

但是这个算法存在反例：

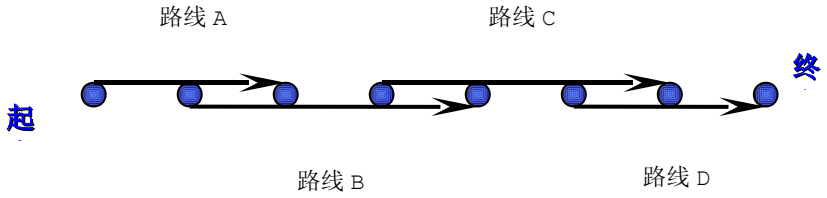


图 2　反例的图示

如图 2，假设这 4 条路线中都只有 1 头牛，那么一架配有两个空位的飞机就可以为所有的 4 头牛服务。但是以上的贪心算法很有可能会为第一个座位安排 A、D 中的两头牛，因为不存在一个 3 头牛的安排方案。这样一来，第二个座位无论如何都只能将一头牛带到目的地了。算法的确只能保证结果是很不错的，不一定最优。

即使这样，这仍然是一个正确率较高的算法。在实际测试中，它通过了官方 16 组测试数据中的 15 组。

深入判断边与边的关系

继续分析上一个算法。对于一个座位的方案而言，如果存在两条边，无论方案选择哪一条最终能座上这个座位的总牛数都一样，算法就会认为两条边是一样好的，但这时它们之间可能仍有优劣之分。所以接下来我们需要更深入地判断边与边之间的优劣关系。

边之间的优劣关系判断起来并不简单，因为每条边都有两个参数——起点和终点。处理这类问题的一个普遍方法是使一个参数有序化，但这里好像并不适用。不过换一个角度想，假如在某一个机场，两头牛抢一个位子，我们会把这个座位分给谁？当然是目的地较近的那个。如此的思维方式使我们很自然地丢掉了起点这个因素：它们无论是从哪儿上来的都不会对这个决定造成影响。

于是，我们得到了一个新的贪心模式：

一站一站地处理。飞到某个飞机场时，在要上机的牛与已经在飞机上的牛中选择目的地最近的C个，让它们坐上/留在飞机上一起前往下一站。有些牛可能会在中途被驱逐下机，有些不牛道，但这个算法无疑是正确而且简洁的。

用数组记录飞机上的牛，每一站进行一次O(C)的成员更新，可以得到一个O(N*C)的高效算法。一些实现上的技巧还可以将其优化到O(N)。

回顾与启示

回顾全过程，尝试动态规划失败，我们退回到一个低起点：网络流算法。在优化过程中，首先对网络流算法加入贪心思想。发现反例后，分析问题所在、调整贪心角度，从而得到了一个正确且更加高效的算法。最后，我们会发现这仍然是一个类似动态规划的算法（只是在选择最优状态时运用了贪心思想），虽然一开始就想到动态规划，但是我们对其中的很多细节不知所措。于是我们以标准的网络流算法为基础，进行大胆地猜想，发现猜想错误后分清问题所在并尝试改进，最终得到了动态规划的具体方法。

思路受阻时，我们需要这样的猜想。如果猜想正确，算法的设计得以继续；即使错误，往往也不用放弃已有的结果。一个错误的猜想至少会让问题的一部分更加明朗，加以分析、纠正后，同样可以得到令人满意的结果。

分析错误的能力为大胆猜想提供了保障。

[例] 渔网[1]

渔网的洞越小，捕到的鱼就越多。因此，渔民们在捕鱼回来后，都要检查一下渔网上有没有大的洞，以便在下次出海之前，将这些洞补好。

渔网被简单的看作一个由顶点和边构成的图。如果图中每一个长度大于3的圈，中间都至少有一段渔网（连接圈上两个顶点，且不在圈上的边）将其隔开，这个渔网就是完美的。请判断输入的渔网是否完美。

初步分析

说到渔网，很容易让人认为所给的图是一个平面图。但仔细看看题目中对渔网的说明就会发现，由渔网得到的图其实是任意的。

这是一道标准的弦图判定问题（判断一个图是不是弦图）。题目中已经对弦图做了形象的解释，这里再给出一个更明确的定义：

[1] 题目来源：ICPC 2001上海赛区，Fishing Net。

弦　图

如果在一个图中，每一个长度大于三的环都至少有一条弦，那么这个图是一个弦图（chordal graph）。弦是连接环上两个不相邻顶点的一条边。

为了方便说明，有弦的环将被简称为合法环，有弦的路径将被简称为合法路径。

弦图判定已有一个形式优美、时间复杂度为O(M)的算法，这个算法足以用来解决这道题目。但是在信息学竞赛中，有关弦图的问题极其罕见，而这个算法以巧妙的数学变换为基础，针对性很强，所以对我们来说价值并不大。它的具体思路与实现在[1]中有详细描述，本文不再关注。下面我们将讨论的是另一个更加直观的算法。

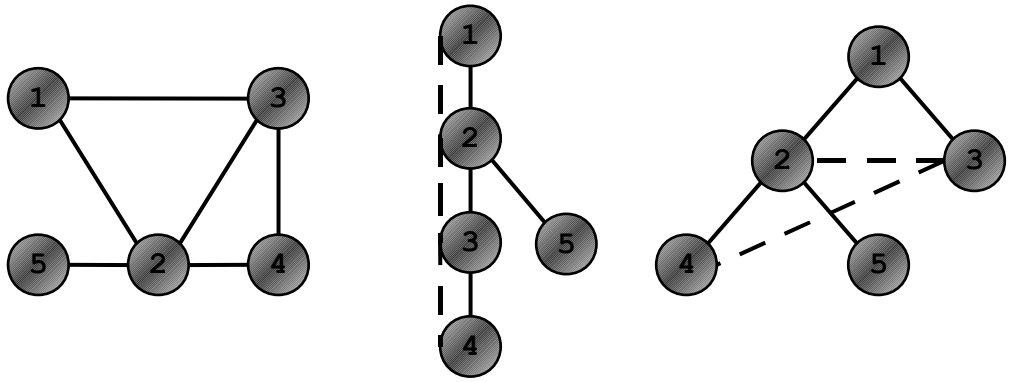
使用搜索枚举环

出于对弦图的定义最直接的理解，我们应该先设法找到图中的环并加以判断。任意的图并没有什么优秀的性质可言，直接在其中寻找环将是盲目而低效的（例如枚举长度然后枚举这一长度的顶点序列）。我们应该先将数据的结构简化。

基本搜索算法就可以做到这一点，它将图有序化从而得到搜索树。树是无环的，而环恰恰是我们所找的。这一点提醒我们将目光转向树枝（前向边）以外的边。

在进一步的思考之前，让我们先来解决另外一个问题——深搜还是广搜？这个选择直接影响着以后的判断方法。

不妨先用手工的方法试试：



原图　　深搜树　　广搜树

图3　生成树的对比

前面已经提到了接下来的焦点将是树外的边。由图3可以看出，这些边在深搜树中以后向边的形式出现，在广搜树中以横叉边的形式出现（这两点也可由各自的搜索顺序经过简单的推理得到）。这个差别很难让我们做出一个很肯定的结论，但至少从表面上看后向边所连接的祖孙关系要比横叉边连接的兄弟关系简单。所以为了讨论的进一步展开，我们选择深度优先搜索。

至此，我们有了一个算法的轮廓：在图上进行深度优先遍历，在发现后向边时检查与它相关的环。

通过递推检查环

如果我们把对应着合法环的后向边称为合法后向边的话，那么剩下的问题就是如何判断后向边是否合法。很自然的，我们需要知道后向边的两端在加入后向边之前的距离是多少。如果距离大于二，那么就找到了一个不合法的圈。事实上我们根本不用求出具体的距离，只要知道一个节点的祖先节点中，哪些离它一条边远、哪些离它两条边远。这需要用两个二维布尔型数组来记录（第一个其实就是邻接矩阵的一部分）。

算法实现时，我们可以通过递推快速地维护第二个数组。

整理主体算法

梳理一下得到的整个算法：

在图上进行深度优先遍历。每到一个待扩展节点时，首先根据它的父节点将它到所有祖先节点的距离初始化，然后逐一检查从它连出的所有后向边，并更新与它祖先节点之间的距离。距离确定后，再次检查所有后向边。如果某一条后向边所连接的两点距离大于二，则判定图不是弦图，退出搜索。否则开始扩展当前节点的后代。

由于检查后向边时要更新当前节点到祖先节点的距离，是 O(N) 的。而后向边的个数是 O(M) 级别的。所以这个算法的总时间复杂度为 O(M*N)，对于题目中 (N≤1000, $M=O(N^2)$) 来说，这个复杂度是不可接受的。但考虑到内层循环操作的是布尔型数组，我们可以进行适当的空间压缩：将 32 个布尔型变量存储到一个 32 位整型变量中。这不会影响到算法的时间复杂度，但可以很实际地将运行时间减少到将近十分之一。

环在搜索树中的多种形态

上一节中，我们得到了一个在题目数据规模内效率接近 O(M) 的新算法。可是稍加揣摩就会发现，算法的构建中有几处并不严密。

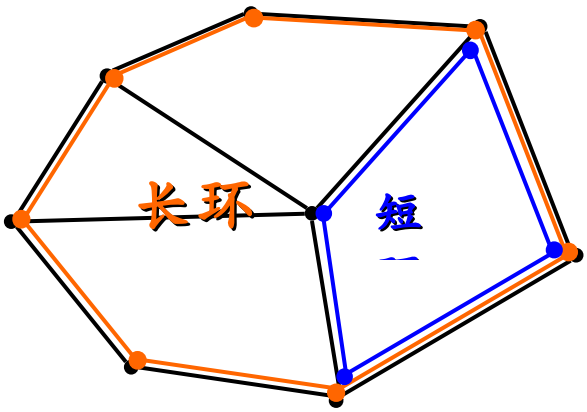


图 4　主要考虑短环的一个原因

先说明一个小问题。对于一个较长的环，如果还有一些点和边间接地连着环上不相邻的点，就会出现长度更小的环（如图 4）；否则，它本身会被算法找出并判断为不合法（从下面的分析中可以看出）。所以我们暂时放开长环，以长度最短的不合法环 C_4（长度为 4 的无弦环）为例讨论改进办法。

C_4 在深搜树中出现的三种形态：

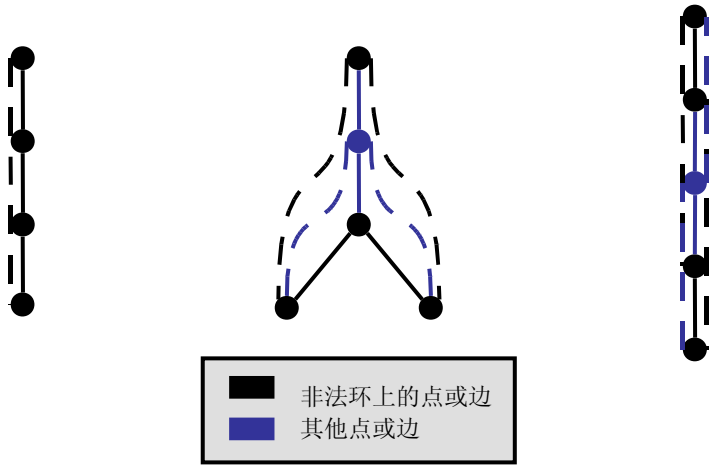


图 5　C_4 出现在深搜树中的三种可能形态

目前的算法所能判断到的环只有第一种。这是由于在另外两种情况下，环上的边都有一条以上不在搜索树上。

解决办法有两种：一、加入另外的判断过程；二、设法将三种情况统一后处理。

经过简单的尝试发现，逐一处理另外两种情况有一定困难，而且这有可能使算法难以实现。恰好又有一个简单的方法可以把各种情况统一，它就是随机化。

三种情况的出现可以认为是等概率的，利用排列组合的知识容易得到下面这样一个表：

随机化次数	1	2	3	4	5	6
漏掉某个特定 C_4 的概率	66.7%	44.4%	29.6%	19.8%	13.2%	8.8%

不难看出，经过几次反复求解后，即使图中只有一个不合法环，它被遗漏的概率也已经十分小了。具体实现中，选择在 5 次以上就完全可以满足题目的要求了。

环与环之间的屏蔽

以上的算法成功地通过了 ZOJ[1] 上的 9 组数据，它对随机图的判断几乎不会出错。但遗憾的是，反例仍然存在。在之前的讨论中，我们忽略了：一条后向边对应的环可能不止一个。

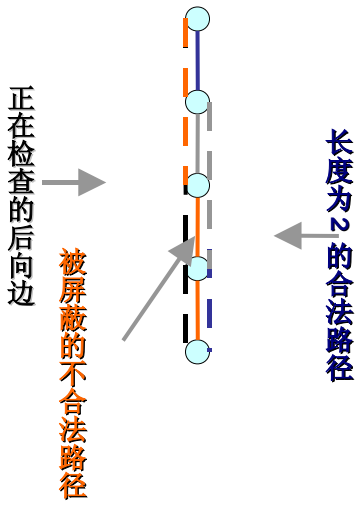


图 6　反例的图示

求距离时，我们得到的是其中最短的一个环减去后向边后的长度。即使这个环合法，也还有可能存在其它更长的环。所以实际的效果是当算法判定一条后向边不合法时，图中一定有不合法环；然而当算法判定一条后向边合法时，它仍有可能不合法。我们所判定的只是后向边合法的必要条件。

随机化加上多次求解的改进对这个漏洞也有一定效果，但是当图中边的密度接近 1 时，算法可能要经过很多次随机才能找到那仅有的几个不合法环。

这个漏洞的存在是因为我们需要的并不是后向边两端点之间的最短路径长度，我们需要

[1] 浙江大学在线题库，http://acm.zju.edu.cn。

的是这“两个端点之间无弦路径中最长的一条的长度”。如果这个长度小于等于2，那么与这条反向边对应的所有环均合法，否则至少有一个不合法。

不幸的是，进一步的思考证明，用树形递推的方法根本不可能在深搜树中求出最长无弦路径的长度。如何转换思路、求出这个长度的方法还在探索中……

不能彻底解决问题的一个改进

起初，为了避免带弦路径的出现，我们直接采用了寻找最短路径的方法。而这其实是无弦路径中我们最不需要的一条（最好是最长无弦路径的长度）。与其辛辛苦苦去找这么一条最没用的路径，我们倒不如任意地找一条，只要无弦就行。

其实，只要在搜索到一个点时，做一次由下向上的扫描，就可以得到它到每个祖先的某一条无弦路径的长度。具体方法并不复杂，我会将程序放在附件中，这里不再深入说明。这个改进在提高算法正确率的同时，把时间复杂度降到了$O(N^2+M)$。

结果

很可惜，最后的结果仍然不算完美。相信在继续的思考与尝试中，这个算法可以被做得更快、更好。

[例] 可疑的斑点[1]

约翰的奶牛中有K(1≤K≤25,000)头格外喜欢闹事儿，排队时她们总是站在一起。为了找出她们，约翰让他所有的N(1≤N≤100,000)头奶牛排成一列进入畜棚，并希望你能告诉他队列中的长度为K的可疑群体。

约翰通过斑点数S(1≤S≤25)来辨别奶牛。他已经忘了那些爱闹事的奶牛身上具体有多少个斑点，但仍然记得在这些牛中谁的斑点更多、或者哪些牛的斑点数一样。于是他用一个由斑点数排名构成的序列描述出了这些奶牛排成的队伍。比如说”1,4,4,3,2,1”表示：第一头与最后一头牛斑点数相同，并且比所有其它牛少；第五头牛身上的斑点稍多一些；而第二头和第三头牛拥有最多的斑点。

请你帮助约翰在队伍中找出所有符合描述的子序列。

初步分析

这是一个模式串可以浮动的模式匹配问题。规模不小，很容易猜到最终的算法会与KMP算法有关。

直接套用肯定不行：KMP的高效在于模式串向后推移后，仍然保存着以前的匹配结果。在这个问题中，模式串位置的推移很有可能会导致模式串本身的变化，以前的结果显然就无法再用了。

从整体枚举到逐一枚举

上面的分析告诉了我们一个简单的道理：KMP中的模式串必须始终不变，想用KMP就得先把模式串确定。这会让人想到预先枚举并一一匹配，但即使题目中的最大斑点数只有25，这也是不能接受的。最坏的情况下：在1..25这25个斑点数中选择12个或13个对应排名，具体的模式串会有500多万个，再来一一匹配，1秒的时间限制是远远不够的。

这个枚举可以剪枝！因为在大多数情况下，只是确定几个排名就会发现这个模式串已经不可能在原串中匹配成功了。新的算法改动较大（因为再用递归就显得有些累赘了）：每次为一个排名枚举出对应的斑点数，用KMP算法得出这个不完整的模式串能匹配到原串的

[1] 题目来源：USACO 2005年十二月竞赛，Cow Patterns。

哪些位置（如题目所说，用起点的位置代表一个子串）。注意到，这个模式串虽然不完整，但它是确定的：如果正在被枚举的排名对应的斑点数是5，那么其它排名在模式串中占的位置始终匹配除了5以外的所有斑点数，所以可以使用KMP算法。做完这些工作后，检查原串中的每个位置，如果一个位置成功地匹配了所有排名，并且这些匹配中各个排名所对应的斑点数正好与排名一致，那么从这个位置开始的长度为K的子串就是一个要找的可疑子串了。

比如，原串是”45858”、可疑子串的斑点排名是”121”。首先考虑排名为1的：当它对应5时，”5?5”可以匹配到原串的第2个位置；当对应8时，”8?8”可匹配到第3个位置；对应其它斑点数时皆无处匹配。然后考虑排名2：对应5时，”?5?”可匹配到1、3两个位置；对应8时，”?8?”只匹配第2个位置；其它无匹配。总结果如下表：

原串	4	5	8	5	8
匹配第一个排名	-	5	8	-	-
匹配第二个排名	5	8	5	-	-

2、3两个位置都成功地匹配了两个排名，但是第3个位置不满足排名的大小顺序要求，所以最后确定的可疑子串只有一个。

跳过已被排除的位置

上面所说的算法有了运行时间上质的飞跃，只是仍然无法完全达到题目的要求。

它的确还会做许多无用功：就上面的例子来说，第一个排名不能匹配1、4、5这三个位置，那么考虑第二个排名时应该直接跳过它们。进一步看，如果按大小顺序处理每个排名，大小关系的矛盾也可以让我们放心地跳过一些位置。

但是在具体的操作过程中如何“跳过”一个位置呢？匹配完一个位置后，找到下一个仍有意义的位置再开始匹配？这可就是朴素的模式匹配了！我们还是得利用KMP算法的思想：如果当前位置已经没有意义，就继续利用后缀函数把模式串向后推。

KMP算法的小优化

优化到这一步，我们已经得到了一个高效的算法。让人奇怪的是，它在测试中出现了错误。分析错误之前，让我们回忆一下KMP算法中对后缀函数的一个小优化。

小 优 化

用一个例子来说明。假设模式串是” aaab”， next() 表示后缀函数。那么当第三个a匹配失败时，无优化的KMP算法会把模式串向后推一位，尝试匹配第二个 a ，即 next(3)=2 。但是这个匹配很明显也会失败，所以在这种情况下，可以直接把后缀函数值再向前赋一步，即 next(3)=next(2) ，以避免无畏的比较。

注意到，这个优化有一个潜在的前提：每一次模式串后推都是由匹配失败导致的。在这个前提下，我们才能确定，如果推移后模式串的待匹配字符没有变化则匹配仍会失败。但是在上面的算法中，我们还会用这种后推方式来跳过已被排除的位置——这时候匹配可能还可以继续。

去掉优化之后，问题终于得到了圆满的解决。

启示

如果把算法中这样的细节出成问答题，稍加揣摩，相信谁也不会答错。但在紧张的考试

中，一个之前从未被考虑过的细节却极有可能被忽略。只有经过了几次错误、投入了足够的思考后，我们才能真正地掌握一个算法。

3.3 小结

与时间复杂度的优化相比，寻找并纠正算法错误是一种更灵活的优化：对错误的敏感使我们在这方面有更多的灵感。广泛积累、总结纠错经验会使算法的设计过程更加流畅，让我们不至于总是因为一点小问题无计可施、甚至从头再来，让算法的世界中条条大路通罗马。

四、结束语

每一次算法优化都是一次思想的旅程，细心与经验是思想的双脚、特殊的信息是思想的路标，无论是否走到了终点，思想都会有所收获。

【参考文献】

[1] Chordal Graph Isomorphism，作者不详。
[2] 算法艺术与信息学竞赛，作者：刘汝佳、黄亮。
[3] USACO关于Flying Right的分析，作者：Bruce Merry。
[4] 从一道题目的解法试谈网络流的构造与算法，作者：江鹏。

【感谢】

衷心感谢吴亚妮老师对我的指导和帮助。
衷心感谢曾呈、吴斌对论文提出的宝贵意见。

【附录】

一、原题

牧场规划

小可可的好朋友Sealock最喜欢吃花生了，于是借用了小可可的牧场从事花生选种试验。他以网格的方式，非常规整地把牧场分割成M*N个矩形区域，由于各个区域中地水面、沼泽面积各不相同，因此各区域地实际可种植面积也各不相同，已知区域(i,j)地可种面积使A(i,j)。

每个区域种最多只能种植一个品种地花生。可种植面积为零地区域不能被选择用来从事选种试验，同时为了防止花粉传播到相邻区域造成试验结果不正确，任何两个相邻的区域都不可以同时种植花生。这里说的相邻指的是两个区域有公共边，仅仅有公共点的两个区域不算做相邻。

小可可准备帮助Sealock规划一下如何选择种植区域，才能使得实际可种植面积总和P最大。

输入：

文件第一行有两个正整数，一次是M、N(M*N≤5000)，用空白字符隔开。随后的M行中，每行有N个不大于100的非负整数，都由空白字符隔开，分别表示各区域的实际可种植面积A(i,j)。

输出：

输出用于选种试验的各区域的实际可种植面积总和的最大值P_{max}。

样例：

输入

```
2 1
1
2
```

输出

```
2
```

输入

```
2 2
10 50
2 3
```

输出

```
52
```

Flying Right

Figuring that they cannot do worse than the humans have, Farmer John's cows have decided to start an airline. Being cows, they decide to cater to the heretofore-untapped market of cows as passengers. They plan to serve the cows who live along the western coast of Lake Michigan. Each morning, they will fly from the northern-most point of the coast southward towards Chicowgo, making many stops along the way. Each evening, they will fly back north to the northern-most point. They need your help to decide which passengers to carry each day. Each of N (1 <= N <= 10,000) farms numbered 1..N along the coast contains an airport (Farm 1 is northern-most; farm N is southern-most). On this day, K (1 <= K <= 50,000) groups of cows wish to travel. Each group of cows wants to fly from a particular farm to another particular farm. The airline, if it wishes, is allowed to stop and pick up only part of a group. Cows that start a flight, however, must stay on the plane until they reach their destination.

Given the capacity C (1 <= C <= 100) of the airplane and the groups of cows that want to travel, determine the maximum number of cows that the airline can fly to their destination.

PROBLEM NAME: flight

INPUT FORMAT:
* Line 1: Three space-separated integers: K, N, and C
* Lines 2..K+1: Each line contains three space-separated integers S,
E, and M that specify a group of cows that wishes to travel.
The M (1 <= M <= C) cows are currently at farm S and want to
travel to farm E (S != E).

SAMPLE INPUT (file flight.in):
4 8 3
1 3 2
2 8 3
4 7 1
8 3 2

INPUT DETAILS:
Four groups of cows, eight farms, and three seats on the
plane.

OUTPUT FORMAT:
* Line 1: The maximum number of cows that can be flown to their
destination. This is the sum of the number of cows flown to
their destination on the flight southward in the morning plus
the number of cows flown to their destination on the flight
northward in the evening.

SAMPLE OUTPUT (file flight.out):
6

OUTPUT DETAILS:
In the morning, the flight takes 2 cows from 1->3, 1 cow from 2->8,
and 1 cow from 4->7. In the evening, the flight takes 2 cows from
8->3.

Fishing Net

In a highly modernized fishing village, inhabitants there make a living on fishery. Their major tools, fishing nets, are produced and fixed by computer. After catching fishes each time, together with plenty of fishes, they will bring back the shabby fishing nets, which might be full of leaks. Then they have to inspect those nets. If there exist large leaks, they have to repair them before launching out again.

Obviously, the smaller the leaks in the fishing nets are, the more fishes they will catch. So after coming back, those fishermen will input the information of the fishing nets into the computer to check whether the nets have leaks.

The checking principle is very simple: The computer regards each fishing net as a simple graph constructed by nodes and edges. In the graph, if any circle whose length (the number of edges) is larger than 3 must has at least one chord, the computer will output "Perfect" indicating that the fishnet has no leaks. Otherwise, "Imperfect" will be displayed and the computer will try to repair the net.

Note: A circle is a closed loop, which starts from one node, passes through other distinct nodes and back to the starting node. A chord is an edge, which connects two different nodes on the circle, but it does not belong to the set of edges on the circle.

INPUT:
The input file contains several test cases representing different fishing nets. The last test case in the input file is followed by a line containing 0 0.

The first line of each test case contains two integers, n and m, indicating the number of nodes and edges on the net respectively, 1 <= n <= 1000. It is followed by m lines accounting for the details of the edges. Each line consists of two integers xi and yi, indicating there is an edge between node xi and node yi.

OUTPUT:
For each test case, display its checking results. The word "Imperfect" suggests that the corresponding fishing net is leaking, while the word "Perfect" stands for a fishing net in

good condition. Follow the output for each net with a blank line.

SAMPLE INPUT:

```
4 4
1 2
2 3
3 4
4 1
3 3
1 2
2 3
3 1
0 0
```

SAMPLE OUTPUT:

```
Imperfect
Perfect
```

Cow Patterns

A particular subgroup of K (1 <= K <= 25,000) of Farmer John's cows likes to make trouble. When placed in a line, these troublemakers stand together in a particular order. In order to locate these troublemakers, FJ has lined up his N (1 <= N <= 100,000) cows. The cows will file past FJ into the barn, staying in order. FJ needs your help to locate suspicious blocks of K cows within this line that might potentially be the troublemaking cows.

FJ distinguishes his cows by the number of spots 1..S on each cow's coat (1 <= S <= 25). While not a perfect method, it serves his purposes. FJ does not remember the exact number of spots on each cow in the subgroup of troublemakers. He can, however, remember which cows in the group have the same number of spots, and which of any pair of cows has more spots (if the spot counts differ). He describes such a pattern with a sequence of K ranks in the range 1..S. For example, consider this sequence:

```
   1 4 4 3 2 1
```

In this example, FJ is seeking a consecutive sequence of 6 cows from among his N cows in a line. Cows #1 and #6 in this sequence have the same number of spots (although this number is not necessarily 1) and they have the smallest number of spots of cows #1..#6 (since they are labeled as '1'). Cow #5 has the second-smallest number of spots, different from all the other cows

#1..#6. Cows #2 and #3 have the same number of spots, and this number is the largest of all cows #1..#6.

If the true count of spots for some sequence of cows is:
   5 6 2 10 10 7 3 2 9
then only the subsequence 2 10 10 7 3 2 matches FJ's pattern above.

Please help FJ locate all the length-K subsequences in his line of cows that match his specified pattern.

PROBLEM NAME: cpattern

INPUT FORMAT:
* Line 1: Three space-separated integers: N, K, and S
* Lines 2..N+1: Line i+1 describes the number of spots on cow i.
* Lines N+2..N+K+1: Line i+N+1 describes pattern-rank slot i.

SAMPLE INPUT (file cpattern.in):
```
9 6 10
5
6
2
10
10
7
3
2
9
1
4
4
3
2
1
```

INPUT DETAILS:
The sample input corresponds to the example given in the problem statement.

OUTPUT FORMAT:
* Line 1: The number of indices, B, at which the pattern matches
* Lines 2..B+1: An index (in the range 1..N) of the starting location where the pattern matches.

```
SAMPLE OUTPUT (file cpattern.out):
1
3

OUTPUT DETAILS:
There is only one match, at position 3 within FJ's sequence of N
cows.
```

二、代码

《Fishing Net》的分析中最后一个算法的Pascal实现：

```
program net;

const
  maxn = 1100;
  maxm = 1000000;
  rantimes = 5;

var
  net: array [1..maxn,1..maxn] of longint;
  deg: array [1..maxn] of longint;
  map: array [1..maxn,1..maxn] of boolean;
  n, m: longint;

procedure ReadIn();
var
  i, a, b: longint;
begin
  fillchar(deg,sizeof(deg),0);
  fillchar(map,sizeof(map),0);
  for i:=1 to m do
  begin
    read(a,b);
    inc(deg[a]);
    net[a,deg[a]]:=b;
    map[a,b]:=true;
    inc(deg[b]);
    net[b,deg[b]]:=a;
    map[b,a]:=true;
  end;
end;
```

```
procedure RandomSwap();
var
  i, j, loca, locb, t: longint;
begin
  for i:=1 to n do
    if (deg[i]>1) then
    begin
      loca:=1;
      t:=net[i,loca];
      for j:=1 to deg[i] do
      begin
        locb:=random(deg[i])+1;
        net[i,loca]:=net[i,locb];
        loca:=locb;
      end;
      net[i,loca]:=t;
    end;
end;

var
  dis: array [0..maxn,0..maxn] of longint;
  vis: array [1..maxn] of boolean;
  row: array [1..maxn] of longint;

function Search(dep,a:longint): boolean;
var
  i, b: longint;
begin
  result:=false;
  vis[a]:=true;
  row[dep]:=a;
  b:=0;
  for i:=dep-1 downto 1 do
  begin
    dis[a,row[i]]:=1+dis[b,row[i]];
    if map[a,row[i]] then
    begin
      if dis[a,row[i]]>2 then exit;
      dis[a,row[i]]:=1;
      b:=row[i];
    end;
  end;
  for i:=1 to deg[a] do
```

```
    if (not vis[net[a,i]]) then
      if (not Search(dep+1,net[a,i])) then
        exit;
  result:=true;
end;

function MaybePerfect(): boolean;
var
  i: longint;
begin
  fillchar(dis,sizeof(dis),0);
  fillchar(vis,sizeof(vis),0);
  fillchar(row,sizeof(row),0);
  result:=false;
  for i:=1 to n do
    if (not vis[i]) then
      if (not Search(1,i)) then
        exit;
  result:=true;
end;

procedure Solve();
var
  perfect: boolean;
  i: longint;
begin
  ReadIn;
  perfect:=true;
  for i:=1 to rantimes do
  begin
    if (not MaybePerfect) then
    begin
      perfect:=false;
      break;
    end;
    RandomSwap;
  end;
  if perfect then
    writeln('Perfect')
  else
    writeln('Imperfect');
  writeln;
end;
```

```
begin
  assignfile(input,'net.in');
  reset(input);
  assignfile(output,'net.out');
  rewrite(output);
  repeat
    read(n,m);
    if (n>0) then
      Solve;
  until (n=0);
  closefile(input);
  closefile(output);
end.
```