浅析信息学中的“分”与“合”

福建省福州第三中学　杨沐

目录

【摘要】

本文就“分”与“合”的思想在信息学中的应用进行讨论与研究。

“分”与“合”思想的精髓在于通过在“分”与“合”之间的转化中找到问题的本质从而解决问题，该思想的应用并非仅限于传统意义上的“分治法”。本文通过分析例题《牛奶模版》《树的重建》与《最优序列》来介绍“分”与“合”思想在解题中的具体应用。

【关键字】

“分”与“合”、转化、辨证关系

【正文】

一、引言

——“话说天下大势，合久必分，分久必合”，这富有辨证意味的第一句话可以说是《三国演义》的精髓。

浩浩中华几千年的历史，充分说明了“合久必分，分久必合”这一观点的正确性。“合久必分，分久必合”是从低级到高级，螺旋式发展的过程， 纵观天下事，无不是在“分”与“合”之间互相转化下发展的。

“分”与“合”的规律不仅在社会发展上得到应验，在信息学中的许多题目的解题过程中也有着充分的体现。

“分”的思想是将一个难以直接解决的大问题，分成一些规模较小或限制某些条件的子问题来思考，以求将问题解决。

“合”的思想与“分”相对，是将“分”出的一些零碎的小问题合并成一个大问题，从而理清关系，得出解法。

在当今信息学竞赛中，我们常常会碰到各种问题，由于规模过大而难以思考，无从入手，又或情况过于纷繁复杂导致思考难度骤增，这时不妨尝试使用“分”与“合”的思考方式来帮助分析问题。

二、例题分析

下面，作者通过介绍三个例题《牛奶模版》、《树的重建》和《最优序列》，结合各题自身特性利用“分”与“合”的思想解题，希望能起到抛砖引玉的作用。

【例一】牛奶模版[1]

问题简述

由于奶牛们的产奶品质波动不定，FJ 决定找出奶牛产奶品质波动的规律。FJ 用一个由 0 到 1000000 的整数组成的长度为 N $(1 \le N \le 20000)$ 的序列 $a1, a2, a3 \ldots a_N$ 记录了各天奶牛们的产奶品质，FJ 希望从中找出最长的一段模版串，使得该模版串在序列中出现至少 k 次，请输出最长模版串的长度。

（例如，12323231 中 2323 出现了 2 次，最长的模版长度为 4。）

问题分析

本题是一道与串匹配有关的问题，我们很自然的想到使用解决字符串问题的利器——后缀树，事实上，使用后缀树确实可以在线性时间内解决本题，但后缀树的高编程复杂度使我们很难在限定时间内完成解题任务。那么，是否存在简单而又行之有效的算法呢?

普通枚举法

解决此题最简单的方法莫过于枚举法了，我们可以枚举出原串的所有子串，互相比对并统计出它们各自出现的次数，从中选出出现次数不少于 k 次的最长子串。由于原串的子串数达到 N^2 的级别，而字符串比较的时间复杂度为 $O(N)$ 因此该枚举法的时间复杂度高达 $O(N^5)$，完全无法在限定时间内出解。

上述枚举法之所以低效，很重要的一个原因是重复比较过多。这时我们又想到了高效的串匹配算法——KMP 算法。

在枚举原串的子串后，我们可以使用 KMP 算法在 $O(N)$ 的时间内算出该串在原串中的出现次数，因而将总算法复杂度降为 $O(N^3)$。

但是，当我们试图使用串匹配算法继续优化算法时间复杂度的时候遇到了重重困难，在多次尝试无果后，不得不回到问题的起点，换一角度思考问题。

[1]选自 usaco contest DEC06 GOLD 《Milk Patterns》

枚举法改进

观察普通的枚举法，我们发现其实枚举所有子串的做法存在许多冗余，因为对于任意两个子串，相等的先决条件是它们的长度必须相同，这提示我们可以枚举子串的长度。

而题目中存在着一个很明显的单调性，即在原串中若存在一个长度为 len 的子串 S_{len} 出现了至少 k 次，则对于任意长度 $len'(len'<len)$，必然能找到一个长度为 len' 的子串 $S_{len'}$ 在原串中出现至少 k 次（只需取 $S_{len'}$ 为 S_{len} 的子串即可）。根据此单调性，我们可以二分枚举答案，之后只判断相同长度的子串是否相等，效率大大提升，时间复杂度优化为 $O(N^3 \log N)$。

转换枚举方式后，低效的子串判等方法成为了算法新的瓶颈。但我们发现，确定了子串的长度 len 后，符合要求的子串只有 $N-len+1$ 个，可以使用 hash 表来存储各串，并用以判等。

根据题目特点，我们设计如下 hash 函数：

对于子串 $sub_k = a_k, a_{k+1} \ldots a_{k+len-1}$，

$$f_k = (a_k \times p1^{len-1} + a_{k+1} \times p1^{len-2} + a_{k+2} \times p1^{len-3} \ldots a_{k+len-1}) \bmod hashsize1$$

$$posi_k = (a_k \times p2^{len-1} + a_{k+1} \times p2^{len-2} + a_{k+2} \times p2^{len-3} \ldots a_{k+len-1}) \bmod hashsize2$$

$p1, p2$ 可以取任意素数，$hashsize1, hashsize2$ 为大素数，$posi_k$ 表示 sub_k 在 hash 表中的位置，使用 f_k 代替 sub_k 的内容（使得冲突时的比较时间复杂度降为 $O(1)$）。若二子串的 $posi$ 与 f 值分别相同，则认为这两个子串相同（由于设置了两个 $hash$ 函数，误判的概率大约仅为 $\frac{N}{hashsize1 \times hashsize2}$，可以忽略）。

由于 sub_k 与 sub_{k+1} 只有 1 个元素不同，因此 $f_{k+1} = f_k \times p1 - a^k \times p1^{len} + a_{k+len}$，$posi$ 同理。这样就能在 $O(N)$ 的时间内算出所有长度为 len 的子串所对应的 f 与 $posi$ 值，因此在理想情况下本方法的时间复杂度已经优化为了 $O(N \log N)$，并且正确率极高，足以通过所有数据。

小结

“分”的思想在很多场合都有应用，在本题的求解过程中，先后尝试了多种方法，最终利用“分”的思想二分答案，巧妙地将 N,k 限制下的最优化问题转化成 N,k,len 限制下的存在性问题，从而得出了一个十分简便而有效的可行算法。

[例二]树的重建[1]

问题简述

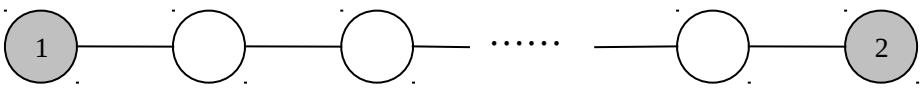
给出一棵树的叶子节点数 $l(2 \le l \le 200)$，以及任意两个叶子间的树中距离（不超过 200），树中任意边边权均为 1，要求还原原树，对于未知节点可任意标号，若无解输出 -1。

问题分析

本题与树有关，题目给出的条件仅为各叶子间的距离关系，对于树中其他节点没有任何描述，这使得直接构树困难重重。

既然直接考虑原问题过于复杂，就从简单情况入手。

考虑若只有 2 个叶节点，则答案显而易见，即一条链的情况。



若有 3 个叶节点，情况略显复杂，设 $dis[i,j]$ 表示叶节点 i 与叶节点 j 的距离。为了简单起见，不妨指定树的根为 1。

1. 当 $dis[2,3] > dis[1,2] + dis[1,3]$ 时

   根据树的性质，此情况必定无解。

   同理，当 $dis[1,2] > dis[1,3] + dis[2,3]$ 或 $dis[1,3] > dis[1,2] + dis[2,3]$ 时也

[1]选自 ZOJ2701 的《Restore the tree》

是无解的。

2. 当 $dis[2,3]=dis[1,2]+dis[1,3]$ 时

根据树的性质，节点 1 必然存在于节点 2 至节点 3 的路径之中，则节点 1 必然不是叶节点，这与题意矛盾，因此依然无解。

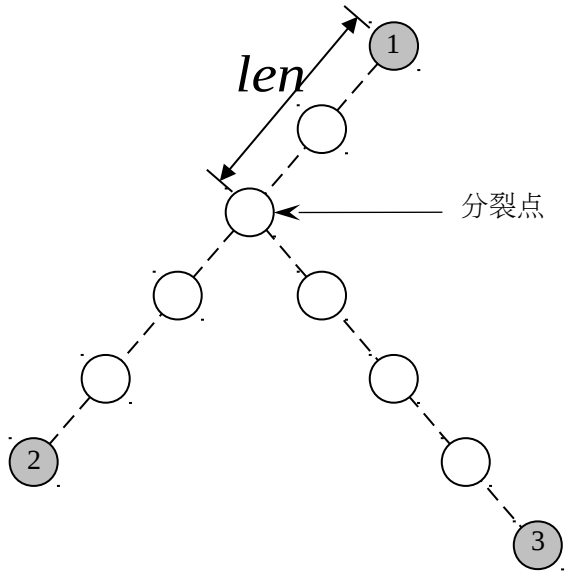
同理，当 $dis[1,2]=dis[1,3]+dis[2,3]$ 或 $dis[1,3]=dis[1,2]+dis[2,3]$ 时也无解。

3. 当 $dis[2,3]<dis[1,2]+dis[1,3]$ 时

① 当 $dis[1,3]=dis[1,2]+dis[2,3]$ 时

与 2.类似，此时节点 2 处于节点 1 至节点 3 的路径上，无解。

若不满足①，则树的形态可描绘如下。



因为 $len=\dfrac{dis[1,2]+dis[1,3]-dis[2,3]}{2}$，所以

② 当 $dis[1,2]+dis[1,3]-dis[2,3]\equiv 1(\mathrm{mod}\,2)$ 时

len 不可能是非整数，因此无解

③ 当 $dis[1,2]+dis[1,3]-dis[2,3]\equiv 0(\mathrm{mod}\,2)$ 时

得出了叶节点 $1,2,3$ 的关系，即节点 3 位于节点 1 至节点 2 的路径上第 len 个点（分裂点）分出的另一条长度为 $dis[1,3]-len$ 的子链的尾部。

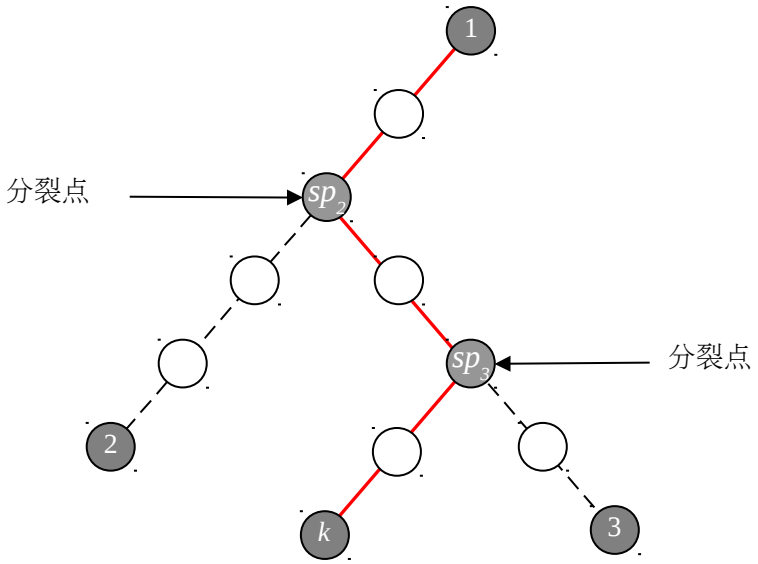
进一步分析

上述简单情况的分析讨论得出：任意两个叶节点必然被一条由非叶节点组成的链所连接，而剩下的所有叶节点与该两点之间的位置关系，都可由上述的讨论确定！

在这一发现的启发下，我们想到了利用“分”与“合”的思想解决该题。设包含 n 个叶节点的原问题为 $P(n)$，若能将一个新叶节点 i 并入一棵已知的树 T 上，则 $P(n)$ 就可以转化为 $P(n-1)$，下面介绍具体做法：

假设当前树 T 已包含了 $k-1$ 个叶节点（不妨设其标号为 $1\ldots k-1$），指定树 T 的根为叶节点 1，当前需要新加入一个叶节点 k。

枚举已在树 T 中的叶节点 $i(i\neq 1)$，由上述对 3 个叶节点的情况讨论可得叶节点 k 关于链 $1\to i$ 的分裂点 sp_i 及其深度 $deep[sp_i]$。



观察树 T 中各分裂点的位置关系。我们发现所有分裂点都处于同一条链上，事实上这是显然的——它们必然在 $1\to k$ 的路径上。而对于每次并入新叶节点，在树 T 中出现的新链仅为深度最深的分裂点→叶节点 k 这一段。

因此，我们只需从所有分裂点中深度最深的分裂点 sp_{max} 上分裂出一条长度为 $dis[1,k]-deep[sp_{max}]$ 新链，链尾即为叶节点 k，这样一来就将叶节点 k 并入了树 T。

同时，在合并新点的过程中，我们发现无解情况实在太多了！这使得思维复杂度与编程复杂度大大提高。但我们发现，若原题存在一组可行解，则上述做法必能将新点并入。因此，我们大可抛弃繁杂的讨论，只在算法结束时判断解的

可行性。

算法框架

至此可以确定算法如下：

1. 建立基树 T，初始时只包含叶节点 $1,2$ 与若干用于连接叶节点 1 与 2 的非叶节点。

2. 取一未被 T 包含的叶节点 i 按上述合并方法将其并入 T。

3. 若 T 未包含所有叶节点，则跳转 2.；否则则得到了一组未检查解 T，转 4.。

4. 检查解 T 中各叶节点是否满足题意。即若叶节点个数与原题给出条件不等，或存在某对叶节点 (i, j) 距离与原题给定距离不同，则无解；否则得到一组可行解 T。

该算法时空复杂度均为 $O(n^2)$，都达到了理论下界，是个十分优秀的算法。

小结

在解决本题的过程中，从简单情况开始讨论，进而发现了题目的重要性质，使用“分”与“合”的方法，将规模为 n 的问题转化为规模为 $n-1$ 的问题，从而得出解法。

【例三】最优序列

问题简述

给出一个长度为 $n(n <= 1000)$ 的正整数序列，求一个子序列，使得原序列中任意长度为 m 的子串中被选出的元素不超过 $k(k, m <= 100)$ 个，并且选出的元素之和最大。

问题分析

初看此题，我们首先考虑动态规划，它是解决这类最优化问题的一把利刃，然而，由于题目条件过于苛刻，使得普通动态规划完全无法胜任，而状态压缩

的动态规划时间复杂度甚至高达 $O(2^M N)$，根本无法在时限内出解。

既然动态规划无法解决此题，我们又很自然的想到了对于一维区间操作十分适用的线段树，可是，经过多次尝试最终发现这种思路也是行不通的。

初步分析

在尝试不同方法解决此题的过程中，我们发现本题的主要困难在于 k 和 m 的范围过大，使得情况过于复杂，难以下手。这时，我们就想到了使用“分”的方法来限制问题的规模，并试图了解问题的本质。

“分”繁为简

现在面对的主要问题是 k 和 m 的范围过大，先抛开这个困扰，让我们来看看这题的一个子问题：

给出一个长度为 $n(n <= 1000)$ 的序列，求一个子序列，使得原序列中任意长度为 m 的子串中被选出的元素不超过 1 个，并且选出的元素之和最大。

对于这个子问题，由于没有了 k 的阻碍，思路豁然开朗，我们可以很轻松的列出动态转移方程来解决这个问题

设 $dp[i]$ 表示区间 $[1, i]$ 所能取到的最大和，则状态转移方程为：

$$\begin{cases} dp[i] = \max(dp[i-1], dp[i-m] + value[i]) & \{i \geq m\} \\ dp[i] = \max(dp[i-1], value[i]) & \{0 < i < m\} \\ dp[0] = 0 \end{cases}$$

那么，这个子问题与原问题究竟有什么关联呢?

进一步分析

随着上述子问题的解决，一个很诱人的想法随之产生：若原问题的解等价于 k 个子问题的解的并，我们就可以通过解决子问题从而将原问题解决。

在这一思路的启发下，我们尝试证明如下命题。

命题：

原问题的解集等价于由 k 组互不相交的子问题的解组成的解集。

定理一 原问题的任意一组解都可以由 k 组不相交的子问题的解组成。

证明：

对于原问题的任意一解 $P=\{a_1,a_2,a_3\ldots a_l\}$，$a_1<a_2<a_3\ldots<a_l$

设 $sum[i]$ 表示该解在区间 $[1,i]$ 内取出的元素个数，则根据题意 sum 满足不等式：

$$sum[i]-sum[i-m]\le k \quad \{i\ge m\}$$

以下，我们给出一种构造法使之能产生一组与该解等价的 k 个子问题的解。

设 k 个子问题的解分别为 $P_0,P_1,P_2\ldots P_{k-1}$，

令 $P_i=\{a_j \mid j\equiv i(\bmod k)\}$

$\because sum[i]-sum[i-m]\le k$

$\therefore a_i-a_{i-k}\ge m$

$\therefore P_0,P_1,P_2\ldots P_{k-1}$ 均为合法的子问题的解

又因为 $P_0\cup P_1\cup P_2\ldots\cup P_{k-1}=P$，因此我们成功的构造出了子问题的解。

定理二 任意 k 组不相交的子问题的解的并均为原问题的解。
证明：

设 k 个子问题的解分别为 $P_0,P_1,P_2\ldots P_{k-1}$，

$P_i=\{a_{i1},a_{i2},a_{i3}\ldots a_{il_i}\}$，$a_{i1}<a_{i2}<a_{i3}\ldots<a_{il_i}$

$\because a_{ix}-a_{i(x-1)}\ge m$

$\therefore$对于任意长度为 m 的区间 $[y,y+m)$，P_i 至多只有一个元素在其内部

设 $P=P_0\cup P_1\cup P_2\ldots\cup P_{k-1}$，则对于任意长度为 m 的区间 $[y,y+m)$，P 在其内部选出的元素个数不超过 k 个
$\therefore$任意 k 组互不相交的子问题的解的并都是原问题的合法解

定理一与定理二分别证明了该命题的充要性，因此该命题成立。

整体分析

随着命题的证明，我们已经成功地将“一个最优解”转化为“K 个子问题的最优并”，接下来的任务就是找到一种方法，使之能产生出最优的不重叠的 k 子

问题解的并。

首先尝试之前所述的朴素动态规划，可惜事与愿违，由于子问题间有重叠，若直接套用 k 次动态规划来求解，有可能导致某个元素被取多次。

动态规划无法解决，那么贪心是否能解决这个问题呢？不能。我们注意到取 K 次局部最优解，它们的并，不一定是全局的最优解。

既然从局部分析已经走入了一个死胡同，我们不妨使用“合”的思想从整体入手，提升自己的思想高度。

考虑动态规划之所以不能得到正确解，其原因在于题目中每个元素只能被取一次的限制，而对于这种限制各点被选取次数的题目，我们通常使用网络流来解决，那么这道题是否也能通过转化图论模型来使用网络流解决呢？答案是肯定的。

构造带权网络 $D=(V,A,C)$

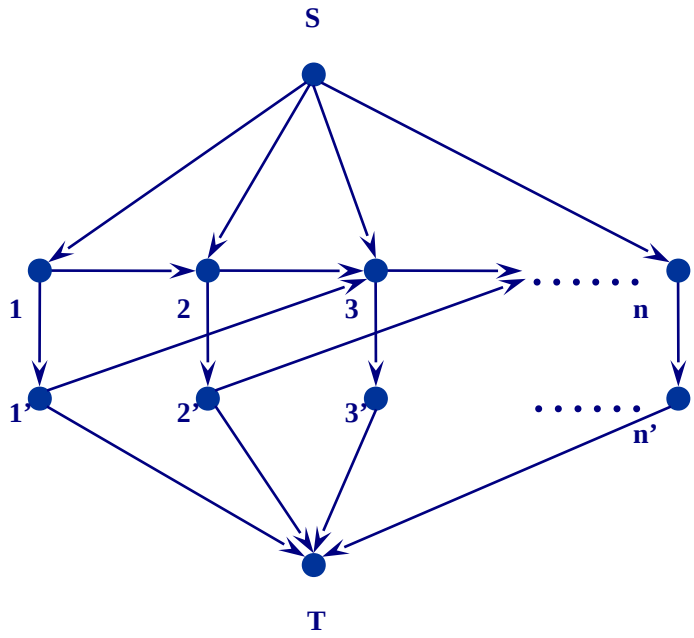
◎序列中的每个元素用顶点 i 与 i' 表示，$i \to i'$ 连边，容量为 1，费用为该元素的数值 $value[i]$，图中还包含源 S 与汇 T 。

◎所有点 i 向点 $i+1$ 连边，容量为 $+\infty$，费用为 0

◎源点 S 向所有点 i 各连一条边，容量为 $+\infty$，费用为 0

◎所有点 i' 向汇点 T 连各连一条边，容量为 $+\infty$，费用为 0

◎所有点 i' 向点 $i+m$ 连边，容量为 $+\infty$，费用为 0



构图完毕之后，网络流量每增加一表示多完成了一个子问题，因此，我们只需要在网络中寻找 k 次最大费用增广路即可得到答案。

由于这张图的边数与顶点数同阶，若使用 SPFA 算法求增广轨，则期望时间复杂度仅为 $O(kN)$，是个十分优秀的算法。

小结

本题在难以入手的情况下通过“分”的思想简化题目，从而理清思路，发现题目本质并转化目标，而后通过“合”的思想采用网络流将所有子问题综合考虑，从而解决了问题。本题充分说明了“分”与“合”的思想在帮助解题时的巨大作用。

三、总结

本文简要介绍了“分”与“合”的几个应用——在题目满足一些单调性时采取二分的做法以优化算法，在发现题目可化归的性质之后从解决小范围问题开始逐步扩大求得正解以及在题目无从下手的情况下利用该思想化简题目并找出算法。

前两个例题皆使用了基于“分”与“合”思想的算法。

对于例题一，其实存在着许多其他精彩的解法，但与其相比，利用“分”的思想入手的思维复杂度极低，易于上手，且总结出的算法编程复杂度亦极低，在争分夺秒的竞赛赛场上可以说是不二的选择。

而在例题二中，化归的方法则成了解题的唯一出路。在前两个例题中，我们发现“分”与“合”虽然对立，却没有明显的分界。一道问题若使用“分”的方法，则必然有“合”的操作，正所谓“分中有合，合中有分”，这两者相互对立，各有优势，却又相互补充，“分”的思想帮助我们迅速地切入问题核心，但在若过分细化则会使问题太过凌乱，失去求解的方向；而“合”的思想则以线串珠，使各种纷杂无序的问题具有了整体性，这正体现了两者之间的辨证关系。

事实上，本文所介绍到的应用仅仅是“分”与“合”思想的皮毛，目的只是希望能起到抛砖引玉的作用。“分”与“合”的思想有着更十分广泛的应用。分治、补集转化等方法就是“分”与“合”思想的经典应用，并已经有了深入地研究，而各类高级数据结构也大量的采用了“分”的方式来提升自身各种操作的效率，其思想的无穷魅力可见一斑。

“分”与“合”不仅仅在算法实现上有着重要的效用，更在思维过程中起着举足轻重的作用，例题三恰好体现了这一点。运用“分”与“合”的思想，对于不同的题目需要不同的分析，其精髓就在于“转化”。无论是“分”还是“合”都是朝着将问题转化为更加便于思考的方向前进，而在这路途中，又需要我们善于归纳总结。只有将已有的知识与“分合”思想有机地结合起来，同时勇于创新，不断积累经验，我们才能从千变万化的题目找寻出本质，从而更快更有效地解决实际问题。

【感谢】

感谢魏丽真老师在我写这篇论文时对我的指导和帮助。

感谢王知昆前辈仔细阅读本文并提出宝贵的修改意见。

感谢袁昕颢同学给我提供例题三。
感谢王航同学在我写论文时的无私帮助。

【参考文献】

[A] 《算法艺术与信息学竞赛》（刘汝佳黄亮著）
[B] 《最短路算法及其应用》余远铭同学 2006 年冬令营论文

【附录】

例一原题:
Problem 3: Milk Patterns [Coaches, 2004]

Farmer John has noticed that the quality of milk given by his cows varies from day to day. On further investigation, he discovered that although he can't predict the quality of milk from one day to the next, there are some regular patterns in the daily milk quality.

To perform a rigorous study, he has invented a complex classification
scheme by which each milk sample is recorded as an integer between
0 and 1,000,000 inclusive, and has recorded data from a single cow over N (1 <= N <= 20,000) days. He wishes to find the longest pattern of samples which repeats identically at least K (2 <= K <= N) times.
This may include overlapping patterns -- 1 2 3 2 3 2 3 1 repeats 2 3 2 3 twice, for example.

Help Farmer John by finding the longest repeating subsequence in the sequence of samples. It is guaranteed that at least one subsequence
is repeated at least K times.

PROBLEM NAME: patterns

INPUT FORMAT:

* Line 1: Two space-separated integers: N and K

* Lines 2..N+1: N integers, one per line, the quality of the milk on
        day i appears on the ith line.

SAMPLE INPUT (file patterns.in):

```
8 2
1
2
3
2
3
2
3
1
```

OUTPUT FORMAT:

* Line 1: One integer, the length of the longest pattern which occurs
        at least K times

SAMPLE OUTPUT (file patterns.out):

```
4
```

例二原题:

Restore the Tree

Time limit: 10 Seconds Memory limit: 32768K Special Judge
Total Submit: 206 Accepted Submit: 26

An undirected connected graph with no cycles is called a *tree*. A vertex with degree equal to one in a tree is called a *leaf*.

Consider a tree. For each pair of leaves one can determine the distance between these leaves --- the number of edges one needs to walk to get from one leaf to another. Given these distances for all pairs of leaves, you need to restore the tree.

Input

There are mutilple cases in the input file.

The first line of each case contains l --- the number of leaves in the tree ($2 <= l <= 200$). Next l lines contain l integer numbers each --- distances between leaves. It is guaranteed that no distance exceeds 200 , all distances are non-negative, distance from a leaf to itself is zero, and the distance from leaf i to leaf j is equal to the distance from leaf j to leaf i .

There is an empty line after each case.

Output

If it is impossible to restore the tree because no such tree exists, output -1 on the first line of the output file.

In the other case first output n --- the number of vertices in the tree. After that output $n-1$ pairs of numbers --- edges of the tree, each specified with two vertices it connects. If there are several solutions, output any. First l vertices must correspond to tree leaves as they are described in the input file, other vertices may be numbered in arbitrary order.

There should be an empty line after each case.

Sample Input

```
3
0 2 3
2 0 3
3 3 0
```

Sample Output

```
5
1 4
2 4
3 5
4 5
```

Problem Source: *Andrew Stankevich's Contest #9*