欧拉回路性质与应用探究

湖南师大附中　仇荣琦

【摘要】

欧拉回路，又称“一笔画”，是图论中可行遍性问题的一种。本文首先介绍了欧拉回路的相关理论知识，以及求欧拉回路的算法。然后通过几个实例，介绍了与欧拉回路相关的几类典型问题。最后对欧拉回路的模型进行了总结，指出其特点和具备的优势。

【关键词】

欧拉回路　欧拉路径

【正文】

一　引言

欧拉回路问题是图论中最古老的问题之一。它诞生于十八世纪的欧洲古城哥尼斯堡。普瑞格尔河流经这座城市，人们在两岸以及河中间的两个小岛之间建了七座桥（如图 1）。

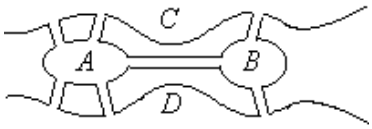

图 1

市民们喜欢在这里散步，于是产生了这样一个问题：是否可以找到一种方案，使得人们从自己家里出发，不重复地走遍每一座桥，然后回到家中？这个问题如果用数学语言来描述，就是在图 2 中找出一条回路，使得它不重复地经过每一条边。这便是著名的“哥尼斯堡七桥问题”。

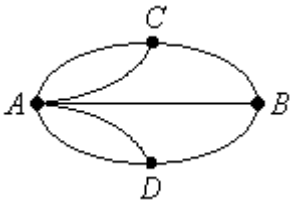

图 2

无数热衷于此的人试图解决这个问题，但均以失败告终。问题传到了欧拉（Leonhard

Euler, 1707-1783）那里，立即引起了这位大数学家的重视。经过悉心研究，欧拉终于在 1736 年发表了论文《哥尼斯堡的七座桥》，不但成功地证明了“七桥问题”无解，而且找到了对于一般图是否存在这类回路的充要条件。后人为了纪念欧拉这位伟大的数学家，便将这类回路称为欧拉回路。

欧拉回路问题在信息学竞赛中有着广泛的应用，近年来在各类比赛中出现了许多与之相关的试题。本文将介绍欧拉回路的相关理论知识，并通过几道例题分析欧拉回路的实际应用。

二　相关知识

首先介绍相关概念和定理。设 $G=(V,E)$ 是一个图。

欧拉回路　图 G 中经过每条边一次并且仅一次的回路称作欧拉回路。

欧拉路径　图 G 中经过每条边一次并且仅一次的路径称作欧拉路径。

欧拉图　存在欧拉回路的图称为欧拉图。

半欧拉图　存在欧拉路径但不存在欧拉回路的图称为半欧拉图。

在以下讨论中，假设图 G 不存在孤立点[1]；否则，先将所有孤立点从图中删除。显然，这样做并不会影响图 G 中欧拉回路的存在性。

我们经常需要判定一个图是否为欧拉图（或半欧拉图），并且找出一条欧拉回路（或欧拉路径）。对于无向图有如下结论：

定理 1　无向图 G 为欧拉图，当且仅当 G 为连通图且所有顶点的度为偶数。

证明　必要性。设图 G 的一条欧拉回路为 C。由于 C 经过图 G 的每一条边，而图 G 没有孤立点，所以 C 也经过图 G 的每一个顶点，G 为连通图成立。而对于图 G 的任意一个顶点 v，C 经过 v 时都是从一条边进入，从另一条边离开，因此 C 经过 v 的关联边的次数为偶数。又由于 C 不重复地经过了图 G 的每一条边，因此 v 的度为偶数。

充分性。假设图 G 中不存在回路，而 G 是连通图，故 G 一定是树，那么有 $|E|=|V|-1$。由于图 G 所有顶点的度为偶数而且不含孤立点，那么图 G 的每一个顶点的度至少为 2。由握手定理，有 $|E|=\frac{1}{2}\sum_{v\in V} d(v)\geq |V|$，与假设相矛盾。故图 G 中一定存在回路。设图 G 中边数最多的一条简单回路[2]为

$$C=\{e_1=(v_0,v_1),e_2=(v_1,v_2),\mathrm{K},e_m=(v_{m-1},v_0)\}$$

[1] 度为 0 的顶点称为孤立点。
[2] 边没有重复出现的回路称为简单回路。

下面证明回路 C 是图 G 的欧拉回路。

假设 C 不是欧拉回路，则 C 中至少含有一个点 v_k，该点的度大于 C 经过该点的关联边的次数。令 $v'_0 = v_k$，从 v'_0 出发有一条不属于 C 的边 $e'_1 = (v'_0, v'_1)$。若 $v'_1 = v'_0$，则顶点 v'_0 自身构成一个环，可以将其加入 C 中形成一个更大的回路；否则，若 $v'_1 \neq v'_0$，由于 v'_1 的度为偶数，而 C 中经过 v'_1 的关联边的次数也是偶数，所以必然存在一条不属于 C 的边 $e'_2 = (v'_1, v'_2)$。依此类推，存在不属于 C 的边 $e'_3 = (v'_2, v'_3), \mathrm{K}\ , e'_k = (v'_{k-1}, v'_0)$。故 $C' = \{e'_1, e'_2, \mathrm{K}\ , e'_k\}$ 是一条新的回路，将其加入 C 中可以形成一个更大的回路，这与 C 是图 G 的最大回路的假设相矛盾。故 C 是图 G 的欧拉回路。

由定理 1 可以立即得到一个用于判定半欧拉图的推论：

推论 1 无向图 G 为半欧拉图，当且仅当 G 为连通图且除了两个顶点的度为奇数之外，其它所有顶点的度为偶数。

证明 必要性。设图 G 的一条欧拉路径为

$$P = \{e_1 = (v_0, v_1), e_2 = (v_1, v_2), \mathrm{K}\ , e_m = (v_{m-1}, v_m)\}$$

由于 P 经过图 G 的每一条边，而图 G 没有孤立点，所以 P 也经过图 G 的每一个顶点，G 为连通图成立。对于顶点 v_0，P 进入 v_0 的次数比离开 v_0 的次数少 1；对于顶点 v_m，P 进入 v_m 的次数比离开 v_m 的次数多 1：故 v_0 和 v_m 的度为奇数。而对于其它任意一个顶点 $v_k (v_k \neq v_0, v_k \neq v_m)$，$P$ 进入 v_k 的次数等于离开 v_k 的次数，故 v_k 的度为偶数。

充分性。设 v_1, v_2 是图 G 中唯一的两个度为奇数的顶点。给图 G 加上一条虚拟边 $e_0 = (v_1, v_2)$ 得到图 G'，则图 G' 的每一个顶点度均为偶数，故图 G' 中存在欧拉回路 C'。从 C' 中删去 e_0 得到一条从 v_1 到 v_2 的路径 P，P 即为图 G 的欧拉路径。

对于有向图，可以得到类似的结论：

定理 2 有向图 G 为欧拉图，当且仅当 G 的基图[3]连通，且所有顶点的入度等于出度。

推论 2 有向图 G 为半欧拉图，当且仅当 G 的基图连通，且存在顶点 u 的入度比出度大 1、v 的入度比出度小 1，其它所有顶点的入度等于出度。

[3] 忽略有向图所有边的方向，得到的无向图称为该有向图的基图。

这两个结论的证明与定理 1 和推论 1 的证明方法类似，这里不再赘述。

注意到定理 1 的证明是构造性的，可以利用它来寻找欧拉回路。下面以无向图为例，介绍求欧拉回路的算法。

首先给出以下两个性质：

性质 1 设 C 是欧拉图 G 中的一个简单回路，将 C 中的边从图 G 中删去得到一个新的图 G'，则 G' 的每一个极大连通子图都有一条欧拉回路。

证明 若 G 为无向图，则图 G' 的各顶点的度为偶数；若 G 为有向图，则图 G' 的各顶点的入度等于出度。

性质 2 设 C_1、C_2 是图 G 的两个没有公共边，但有至少一个公共顶点的简单回路，我们可以将它们合并成一个新的简单回路 C'。

证明 只需按如图 3 所示的方式合并。

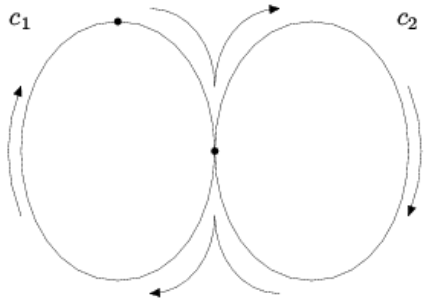

图 3

由此可以得到以下求欧拉图 G 的欧拉回路的算法：

1 在图 G 中任意找一个回路 C；

2 将图 G 中属于回路 C 的边删除；

3 在残留图的各极大连通子图中分别寻找欧拉回路；

4 将各极大连通子图的欧拉回路合并到 C 中得到图 G 的欧拉回路。

该算法的伪代码如下：

```
Procedure Euler-circuit ( start);
Begin
  For 顶点 start 的每个邻接点 v  Do
  If 边 (start, v) 未被标记  Then Begin
    将边 (start, v) 作上标记;
```

将边 $(v,start)$ 作上标记; //1

Euler-circuit (v);

将边 $(start,v)$ 加入栈 S;

End;

End;

最后依次取出栈 S 每一条边而得到图 G 的欧拉回路。

由于该算法执行过程中每条边最多访问两次，因此该算法的时间复杂度为 $O(|E|)$ 。

在实际应用中，以上的实现可能导致堆栈溢出（递归的深度最多可以达到 $|E|$ 层），因此常常需要将其改造成非递归的形式。完整的 Pascal 代码见附录 1（递归形式）和附录 2（非递归形式）。

如果图 G 是有向图，我们仍然可以使用以上算法，只需将标记有//1 的行删去即可。

以上介绍了欧拉回路的相关知识和算法。然而实际问题中，更灵活、更具挑战性的部分则是模型的建立。下面通过剖析若干实例，探究建立欧拉回路模型的方法，使读者对欧拉回路算法有更深入的认识。

三 例题

例题一 单词游戏[4]

题目描述 有 N 个盘子，每个盘子上写着一个仅由小写字母组成的英文单词。你需要给这些盘子安排一个合适的顺序，使得相邻两个盘子中，前一个盘子上面单词的末字母等于后一个盘子上面单词的首字母。请你编写一个程序，判断是否能达到这一要求。如果能，请给出一个合适的顺序。

数据规模 $1 \le N \le 100000$

分析 通过对题目条件的一些初步分析，我们很容易得到下面的模型。

模型 1：以 N 个盘子作为顶点；如果盘子 A 的末字母等于盘子 B 的首字母，那么从 A 向 B 连一条有向边。对于样例我们可以按图 4 所示的方式构图。这样，问题转化为在图中寻找一条不重复地经过每一个顶点的路径，即哈密尔顿路。然而，求哈密尔顿路是一个十分困难的问题，这样的模型没有给我们的解题带来任何便利。因此，我们必须另辟蹊径。

[4] 题目来源：ACM/ICPC Central Europe Regional Contest 1999/2000，有改动

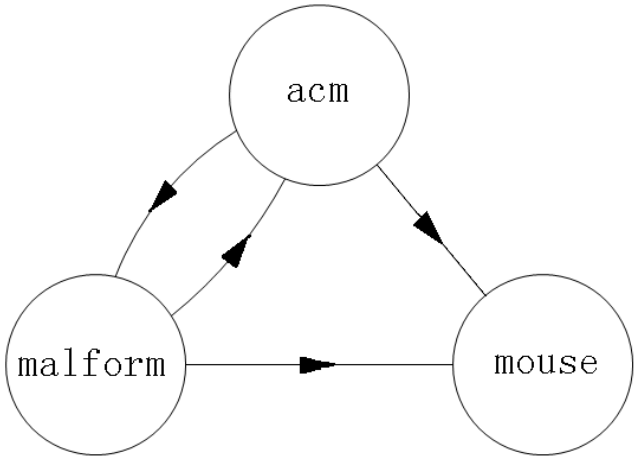

图 4

模型 2：经过分析，我们发现模型 1 的失败之处在于，图中需要遍历的信息——也就是每一个盘子——表示在顶点上，而顶点的遍历问题不易解决。能否将遍历信息表示在边上呢？考虑如下的构图方法：以26 个字母作为顶点；对于每一个盘子，如果它的首字母为 c_1，末字母为 c_2，那么从 c_1 向 c_2 连一条有向边。对于样例我们可以按图 5 所示的方式构图，图中未表示出的顶点均为孤立点，可以事先将其删去。这样，问题转化为在图中寻找一条不重复地经过每一条边的路径，即欧拉路径。这个问题能够在 $O(|E|=N)$ 时间内解决。

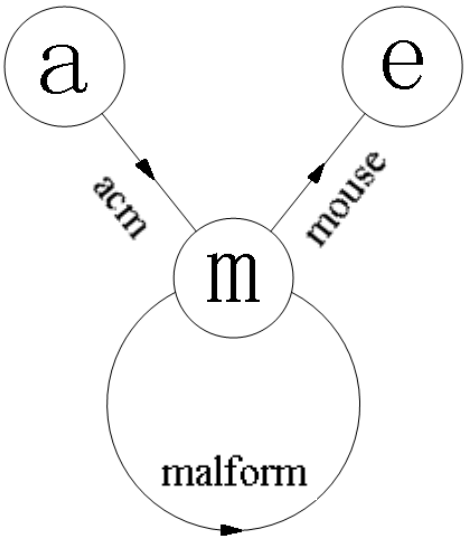

图 5

小结　比较以上两个模型，模型 1 非常直观，模型 2 的建立则需要一点逆向思维：我们已经习惯于“顶点表示元素，边表示元素之间的关系”这种先入为主的思想，而模型 2 则是反其

道而行之——将元素表示在边上，而顶点则起到连接各个元素的作用。这说明，我们考虑问题时，必须将算法进行反复地推敲、改进，甚至打破旧的思维模式，大胆创新，才能找到解决问题的最佳方法。

例题二　仓库管理[5]

题目描述　一个公司有 N 家商店，每家商店销售相同的 M 种商品。公司有一个很大的仓库，商品在运往商店之前首先在这里进行整理、装箱。公司将一定数量的某种商品放到一个箱子中，并贴上商品标签，用 $1\sim M$ 来标识。这样，一共有 $N\times M$ 个箱子，每家商店都将得到标签为 $1\sim M$ 的箱子各一个。仓库是在一个狭窄的建筑里，所以箱子只好排成一列。为了加快分发，管理员决定对箱子重新排列。一个好的排列顺序应该满足以下条件：前 M 个箱子具有不同的标识，以便运往 1 号商店；接下来 M 个箱子具有不同的标识，以便运往 2 号商店……依此类推。另外，初始状态时只有箱子队列的末尾才有唯一的一个空位，每一次移动都只能将一个箱子移动到当前的空位，并且要求所有移动结束后，空位必须回到队尾。请你编写程序，计算重新排列所需的最少移动次数和相应的移动方式。

数据规模　$1\le N,M\le 400$

分析　构造二分图 G，其顶点集 $V=\{p_1,p_2,\mathrm{K},p_n\}\mathrm{Y}\{q_1,q_2,\mathrm{K},q_m\}$，其中 $p_1\sim p_n$ 代表 n 家商店，$q_1\sim q_m$ 代表 m 种商品。设仓库中商店 i 的箱子（即从第 $(i-1)m+1$ 个到第 $i\times m$ 个箱子）中商品 j 的数量为 $t_{i,j}$，我们需要通过重新排列箱子使得所有 $t_{i,j}=1$。对任意 $i,j(1\le i\le n,1\le j\le m)$，若 $t_{i,j}>1$，那么从 p_i 向 q_j 连 $t_{i,j}-1$ 条有向边，表示商店 i 多了 $t_{i,j}-1$ 个商品 j；若 $t_{i,j}<1$，那么从 q_j 向 p_i 连 $1-t_{i,j}$ 条有向边，表示商店 i 少了 $1-t_{i,j}$ 个商品 j。

例如，样例 $N=5$，$M=6$ 的情况，各个箱子的排列情况如下表：

4	1	3	1	6	5	2	3	2	3	5	6	2	1	4	5	6	4	1	3	2	4	5	5	1	2	3	4	6	6	

按上述方法构造图 G 如下：

[5] 题目来源：Central-European Olympiad in Informatics 2005 Day 1 Depot Rearrangement

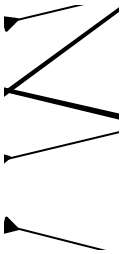

图 6

通过分析整个重排过程中空位的位置，不难发现：初始和结束状态中，空位处于队尾的位置；另外还可能有若干个中间状态，空位也处于队尾的位置。我们将空位处于队尾时的相邻两个状态之间的若干次移动称为一个阶段。

如此定义阶段有什么好处呢？其实，一个阶段的移动与图 G 中的简单回路一一对应。例如，上图中回路 $\{(p_4,q_5),(q_5,p_5),(p_5,q_6),(q_6,p_4)\}$ 对应的操作序列为：

①30→31；

4	1	3	1	6	5	2	3	2	3	5	6	2	1	4	5	6	4	1	3	2	4	5	5	1	2	3	4	6		6

②24→30；

4	1	3	1	6	5	2	3	2	3	5	6	2	1	4	5	6	4	1	3	2	4	5		1	2	3	4	6	5	6

③31→24。

4	1	3	1	6	5	2	3	2	3	5	6	2	1	4	5	6	4	1	3	2	4	5	6	1	2	3	4	6	5	

（图中蓝色格子为移动的起始位置，绿色格子为移动的终止位置。）

现在问题转化为：如何在图 G 中找出若干个没有公共边的回路，使得它们覆盖图 G 中所有的边。注意到一个长度为 $tot\times 2$ 的回路对应的移动次数为 $tot+1$，如果用 c 个回路覆

盖图 G，则总移动次数 $sum=\frac{|E|}{2}+c$。由于题目要求 sum 尽量小，所以 c 要尽量小。而图 G 中，$p_i(1\le i\le n)$ 的入度与出度相等，$q_j(1\le j\le m)$ 的入度与出度相等，因此 G 的每一个极大强连通子图都有欧拉回路。显然，用各极大强连通子图的欧拉回路来覆盖图 G 能够使得回路数最少。

完整的算法流程如下：

1 建立二分图 G；

2 求出 G 的所有极大强连通子图；

3 对于 G 的每一个极大强连通子图，求出一条欧拉回路；

4 根据各极大强连通子图的欧拉回路构造出对应的移动方案。

小结 一些看似与欧拉回路，甚至与图没有太大关系的问题，经过巧妙的转化，就能构图并且转化成在图中求欧拉回路，从而高效地解决问题。这也说明欧拉回路应用的灵活性。

例题三 中国邮路问题（版本 1）[6]

题目描述 A 城市的交通系统由若干个路口和街道组成，每条街道都连接着两个路口。所有街道都是双向通行的，且每条街道都有一个长度值。一名邮递员传送报纸和信件，要从邮局出发经过他所管辖的每一条街道最后返回邮局（每条街道可以经过不止一次）。请问他应该如何安排自己的路线，使得走过的总长度最短呢？

分析 这道题目看起来比较复杂，不好下手。先来分析简单一点的情况。将路口看成顶点，街道看成无向边，得到一个无向图 G。如果 G 不是连通图，那么一定无解；否则，如果 G 是欧拉图，显然 G 的任意一条欧拉回路都是最优线路。如果 G 中奇点[7]个数为 2（设这两个顶点为 v_1，v_2），那么 G 中一定存在一条从 v_1 到 v_2 的欧拉路径 P。然后，找一条从 v_2 到 v_1 的最短路径 R，将其连接到 P 之后得到一条完整的回路。可以证明，这样的路线一定是最优的。

如果 G 中奇点个数大于 2，怎样确定最优路线呢？

注意到前面关于奇点个数为 2 的情况讨论中，我们实际是将从 v_2 到 v_1 的最短路径 R 所经过的边加入到了 G 中，即得到的新图 $G'=G\oplus R$，而欧拉图 G' 的一条欧拉回路即是最优路线。经过分析，我们发现加入任意一条路径 R 的结果是路径 R 的两个端点的度增加了 1，

[6] 题目来源：经典问题

[7] 度为奇数的顶点称为奇点。

奇偶性改变；而路径 R 中其它结点的度增加了 2，奇偶性不变。因此，增加的每一条路径的两个端点都应该是奇点，这样每次增加一条路径就能减少 2 个奇点。我们知道，任意图中奇点个数为偶数[8]。设图 G 中奇点个数为 k，那么一定能通过增加 $\frac{k}{2}$ 条路径使得所有顶点的度为偶数。

更进一步地，考虑如何使得增加的 $\frac{k}{2}$ 条路径的总长度最短。显然，增加的每一条路径都必须是两点间的最短路径；另外，需要给 k 个奇点找到一种合理的配对方式，使得总长度最小。以图 G 的 k 个奇点作为顶点集构造无向完全图 H，边的权值为两点间的最短路径长度，则 H 的最小权完备匹配对应最优的方案。无向完全的最小权匹配可以用 Edmonds 提出的算法[9]在多项式时间内解决。最后按照匹配的方案在图 G 中增边得到图 G'，并在图 G' 中求欧拉回路即可。

完整的算法流程如下：

1　如果 G 是连通图，转 2，否则返回无解并结束；

2　检查 G 中的奇点，构成图 H 的顶点集；

3　求出 G 中每对奇点之间的最短路径长度，作为图 H 对应顶点间的边权；

4　对 H 进行最小权匹配；

5　把最小权匹配里的每一条匹配边代表的路径，加入到图 G 中得到图 G'；

6　在 G' 中求欧拉回路，即所求的最优路线。

例题四　中国邮路问题（版本 2）[10]

题目描述　A 城市的交通系统由若干个路口和街道组成，每条街道都连接着两个路口。**所有街道都只能单向通行**，且每条街道都有一个长度值。一名邮递员传送报纸和信件，要从邮局出发经过他所管辖的每一条街道最后返回邮局（每条街道可以经过不止一次）。请问他应该如何安排自己的路线，使得走过的总长度最短呢？

分析　本题条件与例题三的唯一区别是：所有街道只能单向通行。显然，如果 G 的基图不连通，那么一定无解。另外，如果存在某个顶点，它的入度或者出度为 0，那么不可能存在经过该点的回路。因此，这种情况也是无解的。

仿照例题三的思路，考虑能否通过在图 G 中增加若干条路径得到欧拉图 G'，然后在

[8] 由握手定理，对任意图 G 有 $\sum_{v\in V} d(v) = 2|E|$，故奇点个数必为偶数。

[9] 详见参考文献 4。

[10] 题目来源：经典问题

G' 中求欧拉回路。

首先计算每个顶点 v 的入度与出度之差 $d''(v)$。如果 G 中所有的 $d''(v)=0$，那么 G 中已经存在欧拉回路。否则，由于 $\sum_{v\in V} d''(v)=0$，$d''(v)$ 的值一定是有正有负。对于 $d''(v)>0$ 的顶点 v，需要增加 $d''(v)$ 条从 v 出发的路径；对于 $d''(v)<0$ 的顶点 v，需要增加 $d''(v)$ 条到 v 结束的路径；对于 $d''(v)=0$ 的顶点 v，要求 v 不作为增加的任何一条路径的端点（或者是，以 v 为终止点的路径数等于以 v 为出发点的路径数，但在那种情况下，可以把后者连接到前者之后而合并成一条路径）。这个模型与网络流模型有着惊人的相似！ $d''(v)>0$ 的顶点 v 对应于网络流模型中的源点，它发出 $d''(v)$ 个单位的流；$d''(v)<0$ 的顶点 v 对应于网络流模型中的汇点，它接收 $-d''(v)$ 个单位的流；而 $d''(v)=0$ 的顶点 v 则对应于网络流模型中的中间结点，它接收的流量等于发出的流量。在原问题中还要求增加的路径总长度最小，我们可以给网络中每条边的费用值 $w(e)$ 设为图 G 中对应边的长度。这样，在网络中求最小费用最大流，即可使总费用 $\sum_{e\in E} f(e)w(e)$ 最小。具体来说，可以这样构造网络 N：

1 其顶点集为图 G 的所有顶点，以及附加的超级源 s 和超级汇 t；

2 对于图 G 中每一条边 (u,v)，在 N 中连边 (u,v)，容量为 ∞，费用为该边的长度；

3 从源点 s 向所有 $d''(v)>0$ 的顶点 v 连边 (s,v)，容量为 $d''(v)$，费用为 0；

4 从所有 $d''(v)<0$ 的顶点 u 向汇点 t 连边 (u,t)，容量为 $-d''(v)$，费用为 0。

完整的算法流程如下：

1 如果 G 的基图连通且所有顶点的入、出度均不为 0，转 2，否则返回无解并结束；

2 计算所有顶点 v 的 $d''(v)$ 值；

3 构造网络 N；

4 在网络 N 中求最小费用最大流；

5 对 N 中每一条流量 $f(u,v)$ 的边 (u,v)，在图 G 中增加 $f(u,v)$ 次得到 G'；

6 在 G' 中求欧拉回路，即为所求的最优路线。

拓展 如果将条件改成：部分街道能够双向通行，部分街道只能单向通行，该如何解决？[11]

[11] 本问题已被证明是一个 NPC 问题。详见参考文献 5。

小结 例题三、例题四以及拓展的问题，虽然条件上只有几字之差，但解法却迥然不同。这说明我们在做题时必须仔细审题，紧紧围绕题中的关键条件进行思考。否则，可能会使思维误入歧途。另外，我们平时要注意发散思维、举一反三：如果将题目改动一个条件该如何解决？这样，通过思考和解决一系列具有诸多相似点的问题，我们的思维能力和思维深度得到了锻炼。

例题五 赌博机[12]

题目描述 一台赌博机由 n 个整数发生器 $T_1, T_2, \mathrm{K}\ T_n$ 组成。T_i 能够产生的整数集合为 S_i，并且 S_i 是集合 $\{1,2,\mathrm{K}\ n\}$ 的一个子集。S_i 可以为空集。在赌博机运转的任意时刻，n 个发生器中有且仅有一个发生器处于活动状态。游戏开始时只有 T_1 是活动的。设当前的活动发生器为 T_i，若 $S_i \neq \varnothing$，游戏者可以在 S_i 中选择一个数 r，然后机器将 r 从 S_i 中删除，并将活动状态转移到 T_r；否则，$S_i = \varnothing$，那么游戏结束。如果最后一个活动发生器为 T_1，并且游戏结束时所有 $S_i = \varnothing$，那么游戏者失败，否则获胜。

现在你正站在一台赌博机面前。在开始游戏之前，你决定先编写一个程序，判断你能否获胜。如果能，程序将告诉你每次如何选择合适的数才能获胜。

数据规模 $1 \le n \le 1000$，$\sum_{i=1}^{n} |S_i| \le 12000$

分析 我们将问题抽象成图结构。构造图 G，其顶点集 $V = \{v_1, v_2, \mathrm{K}\ , v_n\}$ 表示 n 个发生器。如果 T_i 可以产生 j，那么从 v_i 向 v_j 连一条有向边。在本问题中，任意一次游戏过程在图 G 中都对应一条从 v_1 出发的简单路径。特别地，一次失败的游戏过程对应一条从 v_1 出发的欧拉回路。如果游戏者无论如何都将失败，那么对应的图 G 满足：从 v_1 出发，不管怎么走，只要不刻意地重复走一条边，一定能走出一条欧拉回路而不会在途中无法继续。我们把这类图称为**随机欧拉图**。显然，随机欧拉图属于欧拉图。根据欧拉图的相关结论，对于基图不连通，或者不满足所有顶点 $v_i (1 \le i \le n)$ 的入度等于出度的情况，游戏者无论如何都不会失败。以下的讨论中，我们只考虑图 G 为欧拉图的情况。

通过分析，我们不难发现这样一个结论：对于任何一次游戏过程，最后一个活动发生器

[12] 题目来源：Polish Olympiad in Informatics 1996 Stage II Problem 3 Gambling，有改动

一定是 T_1。否则，如果最后一个活动发生器为 $T_i(1<i\le n)$，那么在对应的路径中，进入 v_i 的次数比离开 v_i 的次数大 1。而 v_i 的入度等于出度，所以此时一定存在一条从 v_i 出发的边还没有被访问过，这不符合游戏结束的条件。也就是说，任何一次游戏过程在图中对应一个简单回路。

假设某次游戏过程在图 G 中对应的回路为 C。将回路 C 经过的边从图 G 中删去，得到残留图 G'（即 $G'=G-C$），那么 G' 中所有顶点的入度与出度相等。这里分为两种情况：若 G' 为零图[13]，那么这是一次失败的游戏过程；否则，G' 的每一个极大强连通子图都存在欧拉回路。注意到 G' 中 v_1 一定是孤立点，否则游戏不会结束。因此，G' 中一定存在不经过 v_1 的回路。总之，游戏者能够获胜，当且仅当图 G 存在一个子图 G'，使得 G' 由不经过 v_1 的回路组成，并且获胜的游戏过程对应 G' 的补图 G_0 中的欧拉回路。

最后还有一个问题：如何寻找图 G 中不经过 v_1 的回路？其实，只需将图 G 中的顶点 v_1 删除，然后在残留图中进行深度优先遍历（DFS）即可。如果在 DFS 遍历过程中，发现某个顶点 v 存在一条回边指向它的祖先结点 u，那么 DFS 树中从 u 到 v 的路径以及边 (v,u) 构成一个回路。

完整的算法流程如下：

1、 建立有向图 G；

2、 判断其中是否存在欧拉回路。如果存在，转 3，否则任意产生并返回一次游戏过程并结束；

3、 检查图 G 中是否存在不经过 v_1 的回路 C。如果存在，令 $G_0=G-C$，返回图 G_0 中的欧拉回路并结束，否则转 4；

4、 返回游戏失败。

【总结】

欧拉回路是指不重复地经过图中所有边的回路。欧拉回路模型简洁、灵活，容易实现且算法效率高，因而得到广泛的应用。在实际问题中，如果能将主要信息集中在边上，并且转化成遍历图中每一条边，就能很好地运用欧拉回路的相关性质和算法高效地解决。因此，在解题过程中，模型的建立起到了至关重要的作用。而问题的模型往往不是显而易见的，我们

[13] 不含任何边的图称为零图。

必须在仔细分析、研究题目的基础上，挖掘问题的本质，拓展思路，大胆创新，才能建立合适的模型从而高效地解决问题。

【致谢】

本论文的撰写得到湖南师大附中李淑平老师的精心指导，在此表示衷心的感谢！同时感谢班上信息组同学和集训队队友的帮助与支持！

【参考文献】

1 刘汝佳 黄亮 《算法艺术与信息学竞赛》

2 卢开澄 卢华明 《图论及其应用》

3 戴一奇 胡冠章 陈卫 《图论与代数结构》

4 J. Edmonds, E. Johnson 《Matching, Euler tours, and the Chinese postman》

5 C. Papadimitriou 《The complexity of edge traversing》

6 Baltic Olympiad in Informatics 2001 官方解答

【附录】

附录 1 求无向图的欧拉回路（递归实现）

```
program euler;
  const maxn=10000;{顶点数上限}
        maxm=100000;{边数上限}
  type tnode=^tr;
       tr=record
            f,t:longint;{边的起始点和终止点}
            al:boolean;{访问标记}
            rev,next:tnode;{反向边和邻接表中的下一条边}
          end;
  var n,m,bl:longint;{顶点数，边数，基图的极大连通子图个数}
      tot:longint;
      g:array[1..maxn] of tnode;
      d:array[1..maxn] of longint;{顶点的度}
      fa,rank:array[1..maxn] of longint;{并查集中元素父结点和启发函数值}
      list:array[1..maxm] of tnode;{最终找到的欧拉回路}
      o:boolean;{原图中是否存在欧拉回路}
  procedure build(ta,tb:longint);{在邻接表中建立边(ta, tb)}
    var t1,t2:tnode;
    begin
      t1:=new(tnode);
      t2:=new(tnode);
      t1^.f:=ta;
```

```
    t1^.t:=tb;
    t1^.al:=false;
    t1^.rev:=t2;
    t1^.next:=g[ta];
    g[ta]:=t1;
    t2^.f:=tb;
    t2^.t:=ta;
    t2^.al:=false;
    t2^.rev:=t1;
    t2^.next:=g[tb];
    g[tb]:=t2;
  end;
procedure merge(a,b:longint);{在并查集中将 a, b 两元素合并}
  var oa,ob:longint;
  begin
    oa:=a;
    while fa[a]<>a do a:=fa[a];
    fa[oa]:=a;
    ob:=b;
    while fa[b]<>b do b:=fa[b];
    fa[ob]:=b;
    if a<>b then begin
      dec(bl);{合并后，基图的极大连通子图个数减少 1}
      if rank[a]=rank[b] then inc(rank[a]);
      if rank[a]>rank[b] then fa[b]:=a else fa[a]:=b;
    end;
  end;
procedure init;{初始化}
  var i,ta,tb:longint;
  begin
    fillchar(fa,sizeof(fa),0);
    fillchar(rank,sizeof(rank),0);
    fillchar(d,sizeof(d),0);
    readln(n,m);
    for i:=1 to n do fa[i]:=i;
    bl:=n;
    for i:=1 to m do begin
      readln(ta,tb);
      build(ta,tb);
      inc(d[tb]);
      inc(d[ta]);
      merge(ta,tb);
    end;
  end;
```

```
  procedure search(i:longint);{以 i 为出发点寻找欧拉回路}
    var te:tnode;
    begin
      te:=g[i];
      while te<>nil do begin
        if not te^.al then begin
          te^.al:=true;
          te^.rev^.al:=true;
          search(te^.t);
          list[tot]:=te;
          dec(tot);
        end;
        te:=te^.next;
      end;
    end;
  procedure main;{主过程}
    var i:longint;
    begin
      o:=false;
      for i:=1 to n do
        if d[i]=0 then dec(bl);{排除孤立点的影响}
      if bl<>1 then exit;{原图不连通，无解}
      for i:=1 to n do
        if odd(d[i]) then exit;{存在奇点，无解}
      o:=true;
      for i:=1 to n do
        if d[i]<>0 then break;
      tot:=m;
      search(i);{从一个非孤立点开始寻找欧拉回路}
    end;
  procedure print;{输出结果}
    var i:longint;
    begin
      if not o then writeln('No solution.') else begin
        writeln(list[1]^.f);
        for i:=1 to m do writeln(list[i]^.t);
      end;
    end;
begin
  init;
  main;
  print;
end.
```

附录 2 求无向图的欧拉回路（非递归实现）

```
program euler;
  const maxn=10000;{顶点数上限}
  type tnode=^tr;
       tr=record
            f,t:longint;{边的起始点和终止点}
            rev,prev,next:tnode;{反向边和邻接表中的上一条边，下一条边}
          end;
  var n,m,bl:longint;{顶点数，边数，基图的极大连通子图个数}
      g:array[1..maxn] of tnode;
      d:array[1..maxn] of longint;{顶点的度}
      fa,rank:array[1..maxn] of longint;{并查集中元素父结点和启发函数值}
      o:boolean;{原图中是否存在欧拉回路}
      f,link:tnode;{用链表保存最终找到的欧拉回路}
  procedure build(ta,tb:longint);{在邻接表中建立边(ta, tb)}
    var t1,t2:tnode;
    begin
      t1:=new(tnode);
      t2:=new(tnode);
      t1^.f:=ta;
      t1^.t:=tb;
      t1^.rev:=t2;
      t1^.prev:=nil;
      t1^.next:=g[ta];
      if g[ta]<>nil then g[ta]^.prev:=t1;
      g[ta]:=t1;
      t2^.f:=tb;
      t2^.t:=ta;
      t2^.rev:=t1;
      t2^.prev:=nil;
      t2^.next:=g[tb];
      if g[tb]<>nil then g[tb]^.prev:=t2;
      g[tb]:=t2;
    end;
  procedure merge(a,b:longint);{在并查集中将 a, b 两元素合并}
    var oa,ob:longint;
    begin
      oa:=a;
      while fa[a]<>a do a:=fa[a];
      fa[oa]:=a;
      ob:=b;
      while fa[b]<>b do b:=fa[b];
      fa[ob]:=b;
      if a<>b then begin
```

```
      dec(bl);{合并后，基图的极大连通子图个数减少 1}
      if rank[a]=rank[b] then inc(rank[a]);
      if rank[a]>rank[b] then fa[b]:=a else fa[a]:=b;
    end;
  end;
procedure init;{初始化}
  var i,ta,tb:longint;
  begin
    fillchar(fa,sizeof(fa),0);
    fillchar(rank,sizeof(rank),0);
    fillchar(d,sizeof(d),0);
    readln(n,m);
    for i:=1 to n do fa[i]:=i;
    bl:=n;
    for i:=1 to m do begin
      readln(ta,tb);
      build(ta,tb);
      inc(d[tb]);
      inc(d[ta]);
      merge(ta,tb);
    end;
  end;
procedure delm(t:tnode);{在邻接表中删除边 t}
  var no:longint;
  begin
    no:=t^.f;
    if t^.prev<>nil
      then t^.prev^.next:=t^.next
      else g[no]:=t^.next;
    if t^.next<>nil then t^.next^.prev:=t^.prev;
  end;
procedure findcircle(no:longint;var head,tail:tnode);{以 no 为起点找一条回路，保存在以 head 为头，tail 为尾的链表中}
  var i:longint;
  begin
    i:=no;
    head:=g[i];
    tail:=head;
    g[i]:=g[i]^.next;
    if g[i]<>nil then g[i]^.prev:=nil;
    tail^.prev:=nil;
    tail^.next:=nil;
    delm(tail^.rev);
    i:=tail^.t;
```

```
    while i<>no do begin
      tail^.next:=g[i];
      g[i]:=g[i]^.next;
      if g[i]<>nil then g[i]^.prev:=nil;
      tail:=tail^.next;
      tail^.prev:=nil;
      tail^.next:=nil;
      delm(tail^.rev);
      i:=tail^.t;
    end;
  end;
procedure main;{主过程}
  var i:longint;
      te,head,tail:tnode;
  begin
    o:=false;
    for i:=1 to n do
      if d[i]=0 then dec(bl);{排除孤立点的影响}
    if bl<>1 then exit;{原图不连通，无解}
    for i:=1 to n do
      if odd(d[i]) then exit;{存在奇点，无解}
    o:=true;
    for i:=1 to n do
      if d[i]<>0 then break;
    findcircle(i,head,tail);{首先任意找一条回路}
    link:=head;
    f:=link;
    while link<>nil do
      if g[link^.t]<>nil then begin
        findcircle(link^.t,head,tail);
        tail^.next:=link^.next;
        link^.next:=head;{将两个回路合并}
      end else link:=link^.next;
  end;
procedure print;{输出结果}
  begin
    if not o then writeln('No solution.') else begin
      link:=f;
      writeln(link^.f);
      while link<>nil do begin
        writeln(link^.t);
        link:=link^.next;
      end;
    end;
```

```
    end;
begin
  init;
  main;
  print;
end.
```

附录3 例题一的英文原题

Play on Words

Some of the secret doors contain a very interesting word puzzle. The team of archaeologists has to solve it to open that doors. Because there is no other way to open the doors, the puzzle is very important for us.

There is a large number of magnetic plates on every door. Every plate has one word written on it. The plates must be arranged into a sequence in such a way that every word begins with the same letter as the previous word ends. For example, the word “acm” can be followed by the word “Motorola”. Your task is to write a computer program that will read the list of words and determine whether it is possible to arrange all of the plates in a sequence (according to the given rule) and consequently to open the door.

Input Specification

The input consists of T test cases. The number of them (T) is given on the first line of the input file. Each test case begins with a line containing a si ngle integer number N that indicates the number of plates ($1 \leq N \leq 100000$). Then exactly N lines follow, each containing a single word. Each word contains at least two and at most 1000 lowercase characters, that means only letters ’a’ through ’z’ will appear in the word. The same word may appear several times in the list.

Output Specification

Your program has to determine whether it is possible to arrange all the plates in a sequence such that the first letter of each word is equal to the last letter of the previous word. All the plates from the list must be used, each exactly once. The words mentioned several times must be used that number of

times.

If there exists such an ordering of plates, your program should print the sente nce "Ordering is possible.". Otherwise, output the sentence "The door cannot be opened.".

Sample Input

3

2

acm

ibm

3

acm

malform

mouse

2

ok

ok

Output for the Sample Input

The door cannot be opened.

Ordering is possible.

The door cannot be opened.

附录4 例题二的英文原题

Depot Rearrangement

A company operates N shops, selling M different products in each shop. The company has a large depot where the products are packed before delivering to shops. Each shop receives the same number of items of each product. Hence the company packs a certain number of items of a given product into a container, and labels that container with the product identifier. Products are identified by the numbers from 1 to M. Thus, at the end of packing, there are N*M

containers in the depot, and exactly N containers are labeled with a given product label for each product. Because the depot is in a narrow building, the containers are arranged in a single row. In order to speed-up distribution, the manager of the depot wants to rearrange the containers. Since the product delivery to the shops occurs by sending exactly one truck to each shop, and each truck carries one container of each product, a suitable arrangement is the following. The first M containers in the row must be labeled with different product labels, the second M containers in the row must be labeled with different product labels, and so on. Unfortunately, there is only one free place at the end of the row to hold a container. Therefore the rearrangement must be performed by successively picking up a container and moving it to the free place. After the rearrangement the free place must be at the end of the row.

The goal is to achieve the required rearrangement by a minimal number of moves.

Task

You are to write a program that computes a rearrangement which needs the minimal number of moves.

Input

The first line of the text file depot.in contains two integers, N and M. N ($1 \leq N \leq 400$) is the number of shops and M ($1 \leq M \leq 400$) is the number of products. The second line contains N*M integers, the labels of the containers in their initial order. Each product identifier x ($1 \leq x \leq M$) occurs exactly N times in the line.

Output

The first line of the text file depot.out contains one integer S, the minimal number of moves that are necessary to obtain a required order of the container row (Subtask A). The following S lines describe a rearrangement (Subtask B). Each line contains a pair of integers x y. The pair x y describes a move: the container at position x is to move to position y. Positions are identified by

the numbers from 1 to N*M+1; initially the position N*M+1 is free (holds no container). A move from x to y is legal only if position y is free prior to the move. After a move from x to y the position x will be free. It is enough to output only the first line if you solve only Subtask A.

If there are multiple possibilities, your program should output only one; it does not matter which one.

Example

depot.in

5 6

4 1 3 1 6 5 2 3 2 3 5 6 2 1 4 5 6 4 1 3 2 4 5 5 1 2 3 4 6 6

depot.out

8

9 31

18 9

10 18

4 10

31 4

30 31

24 30

31 24

附录5 例题五的英文原题

Gambling

A gambling machine consists of n generators of integers: G_1, ..., G_n, where 1 <= n <= 1000. The generator G_i can generate integers only from the certain set S_i included in the interval {1,..., n}, or 0 which means that the game is over. The S_i can be an empty set. Let k_i be the number of elements of the set S_i. The sum of all the integers k_i, for i = 1,..., n, cannot exceed 12000.

While G_i is activated for the first time it generates an integer from the set

S_i. The next activating ends up in generating an integer from the set S_i, which was not chosen before. If there is no such an integer, G_i generates zero.

The machine starts with activating G_1. Then the generators are activated according to the following rule :

· In case when a generator generated a positive integer r, the next activated generator is G_r.

· If zero is generated the machine stops.

The machine looses if the generator G_n generates zero and the rest of the generators had exhausted their sets of integers.

The machine is well constructed if it may generate a sequence of integers ending with zero, but not leading to a defeat (i.e. a sequence shorter then 1 + (the sum of integers ki for i = 1,...,n)).

Task

Write a program that:

· Reads from the text file HAZ.IN the description of the machine, i.e. the number of generators n, integers k_i and sets S_i,

· Verifies, whether the machine described in the input file is well constructed,

· if it is well constructed, writes a sequence of integers, which may be generated by the machine, is ended with zero, and does not lead to defeat,

· if not, writes one word NIE ("no" in Polish) in the text file HAZ.OUT.

Input

In the first line of the text file HAZ.IN there is written one positive integer n <= 1000. This is the number of generators. In the (i + 1)-st line (for i = 1,..., n) there is written an integer ki followed by all the elements of the set S_i (written in arbitrary order). The integers in each line are separated by single spaces.

Output

In the text file HAZ.OUT there should be written one word NIE (if the machine isn't well constructed) or one line containing an appropriate finite series of integers separated by single spaces.

Example

For the input file HAZ.IN:

2

2 1 2

1 2

the correct answer is the output file HAZ.OUT:

2 2 0

For the input file HAZ.IN:

2

1 2

0

the correct answer is the output file HAZ.OUT:

NIE