图论模型的建立与转化

安徽 徐静

关键字：图论模型、建立、转化

摘要

本文主要写图论模型的建立与转化，共分四部分：

第一部分引言说明了图论建模在整个信息学竞赛中的地位，以及图论模型与其它数学模型的异同，并指出很有研究总结图论建模的思想、方法及技巧的必要。

第二部分提出了图论模型**建立中**的两个要点：***对原型中的要素进行适当的取舍***和***选择合适的理论体系***，并分别举例加以详细分析，然后从中总结出了图论建模的总的原则：准确、清晰、简明。

第三部分主要讨论了在图论模型的**转化**中，应用得较为广泛的两种方法：***拆分转化***和***补集转化***，并着重分析了前者。文中把前者分为三类：点→边、点→点、边→边，其中详细分析了第二类。

第四部分总结了全文，并指出了进一步研究图论模型的必要性。

目录

正文

一. 引言

信息学竞赛以解题为主，整个解题过程中一个重要的步骤就是数学建模，本文要讨论的就是数学建模的一个分支——图论建模。

图论建模是指对一些客观事物进行抽象、化简，并用图[1]来描述事物特征及内在联系的过程。

建立图论模型的目的和建立其它的数学模型一样，都是为了简化问题，突出要点，以便更深入地研究问题的本质；它的求解目标可以是最优化问题，也可以是存在性或是构造性问题；并且，和几何模型、运筹学模型一样，在建立图论模型的过程中，也需要用到集合、映射、函数等基本的数学概念和工具；

但图论模型和其它模型在它们的研究方法上又有着很大的不同，例如我们可以运用典型的图论算法来对图论模型进行求解，或是根据图论的基本理论来分析图论模型的性质，这些特殊的算法和理论都是其它模型所不具备的，而且在其它模型中，能用类似于图这种直观的结构来描述的也很少。

我们学习图论，一般都是通过书籍，但书上介绍的往往只限于图论模型的基本要素、一些图论的相关理论和经典算法等，至于如何建立图论模型、如何运用这些理论和算法、如何研究图论问题，都只有靠自己来理解、来领会，并通过实践来验证这些理解，通过摸索总结来提高自己的能力。

在建立图论模型的过程中，我们常常会遇到一些困难，例如难以建立点、边、权关系，或是原型中的一些重要因素无法纳入现有模型，或是现有模型虽能表示原型，却无法求解等等。为了克服这些困难，就需要用到某些独特的思想、方法和技巧，本文要写的正是我在学习、实践中得出的这方面的一点认识。

二. 图论模型的建立

在建立模型之前，我们首先要对研究对象进行全面的调查，将原型理想化、简单化（对于竞赛题而言，这一步大部分已经由出题人完成了）；然后对原型进行初步的分析，分清其中的各个要素及求解目标，理出它们之间的联系；下一步就是用恰当的模型来描述这些要素及联系。

I. 要素的取舍

在用图论模型描述研究对象时，为了更突出与求解目标息息相关的要素，降低思考的复杂度，就不可避免地要舍去部分要素。下面我们就通过例 1 来分析一下。

【例1】 导线排布 Line[7]：

题目（文档附件：导线排布.doc）中蓝色的一段是问题描述的重点，其中涉及的要素有圆圈、N 根导线、2N 个端点、编号规则、导线的交叉等，求解目标是构造一种符合所给的导线交叉情况的导线排布方案。

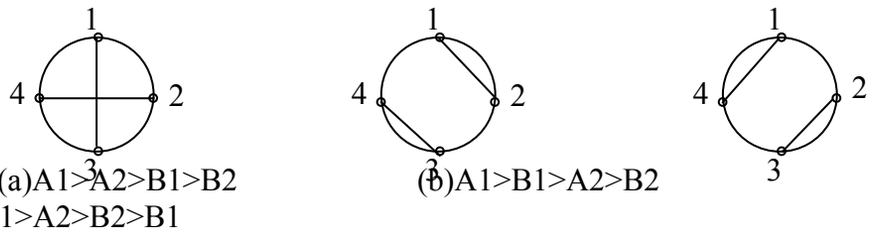
起先，我们对题目描述的导线排布并不熟悉，或许我们能够画出几个无解或是多解的例子，但竞赛时我们不可能花更多的时间在熟悉题目上了，这时只有尽快地把我们不熟悉

[1] 在本文中，“图”专指由若干不同顶点与连接其中某些顶点的边所组成的图形[6]，不包括一般的示意图。

的、难于思考的原型转化成我们熟知的、便于思考的模型。

先来分析求解目标：所谓的构造导线排布方案，也就是找出每根导线两个端点的编号；而编号要满足的条件就是导线交叉的情况。

那么下一步我们就来分析一下编号与导线交叉之间的关系。记第 i 根导线两端点的标号为 Ai 和 Bi（Ai<Bi），两根导线的交叉情况有 3 种，对应的编号情况如图一所示，它们的



(a)A1>A2>B1>B2　　(b)A1>B1>A2>B2

(c)A1>A2>B2>B1

图 一

共同点是 A1>B1，A2>B2，A1>A2（根据编号规则），不同的是(a)满足 A2>B1，B1>B2，(b)满足 B1>A2，而(c)满足 B2>B1。显然，这是一种偏序关系（有点不确切，它只满足反对称和传递性，但不是自反的），而我们的任务就是根据这种偏序关系求得全序关系，即拓扑排序。

我们用图中的有向边来表示偏序关系，若有向边构成环，则问题无解。以上三种情况对应的有向图如图二所示，若两导线交叉，则如(a)；若不交叉，则必是(b)、(c)其中之一，

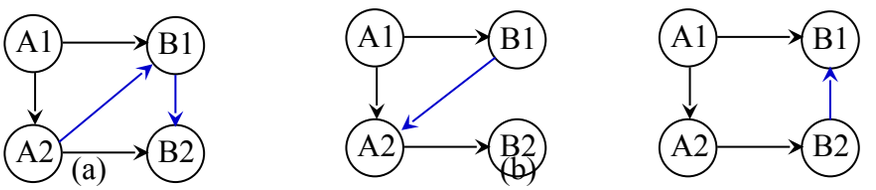


图 二

至于选择哪一个，就要看它们中哪一个不会导致无解。

若**(b)无解**，就表示图中除了 B1→A2 这条边之外，还存在着一条 A2→B1 的路径，可能的两种情况如图三(1)(2)中红色有向边所示：(1)表示有编号大于 2 的导线（(1)中的导线 3）与导线 1 交叉；(2)表示虽然它们都不与导线 1 交叉，但其中有选择图二(c)表示的。

显然，如果情况(1)出现，那么选择(b)必然无解；但如果(1)不出现，那么(2)就可以改成图三(3)（即改用(b)表示导线 1、3 不交叉，而不是用(c)），这样就有解了。也就是说，如果***导线 i 与导线 j(i<j)不交叉，***那么***是否有编号大于 j 的导线与导线 i 交叉***决定了***(b)是否无解。***

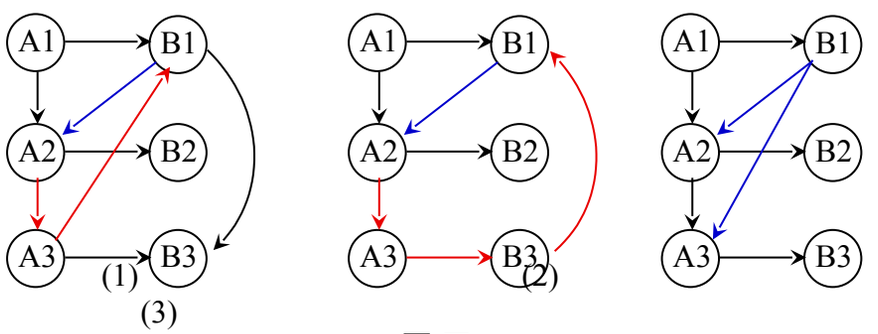
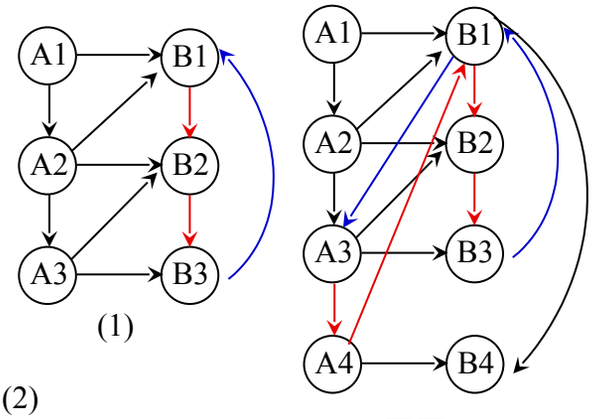


图 三

类似的，我们可以分析出：如果***导线 i 和 j (i<j) 不交叉***，那么***是否存在一个序列 $L_{1\sim m}$ (L_1=i，L_m=j)，使得导线 L_{k-1} 与导线 L_k(1<k≤m) 交叉***，决定了***(c) 是否无解***。

图四(1)画出了(c)无解的情况，此时导线 1 与 3 不交叉，m=3，L=(1,2,3)；图四(2)表示导线 1 与 3 不交叉，导线 1 与 2，2 与 3，1 与 4 交叉，此时用(b)或(c)表示导线 1 与 3 不交叉都无解，所以整个问题无解。



图四

分析到这里，图论模型和算法实际上已经出来，具体的实现就不多说了。

上面的分析看似冗长，但实际上，分析的过程如行云流水，十分自然，究其原因，就是我们舍去了原型中一些与求解目标无关的因素，而仅用一个有向图来表示交叉关系。比起紧紧抱住圆圈、导线这些细节不放，直接分析原型的方法来，分析模型的思维复杂度自然要小得多。

在建模过程中，忽略掉原型中与求解目标关系不大的要素，能够适当地简化问题。但事物都是一分为二的，如果简化得过了度，反而会使模型变得不够准确，甚至是不正确的，因此，只有在“简明”和“准确”之间找到平衡点，才能建立起具有最佳效果的模型。

II. 选择合适的理论体系

图由点、边、权三部分组成，根据这三部分的性质的不同，就有着不同的图论模型，有着不同的理论和算法，也就构成了不同的理论体系。

例如二部图把整个点集 V 分为两个子集，规定子集内部的点之间没有边，因此二部图就有着不同于一般图的特殊性质，而它的匹配算法也就比一般图的算法简单；此外还有树、有向无环图等，它们属于不同的理论体系，有着各自不同的性质，适于用不同的算法求解。

而权的加入则使图论模型和求解目标变得更加复杂多样了。

有的权表示长度或是时间等等，它们的运算特征是“串联求和，并联求最值”，即一条路径的权由这条路径上每条边的权相加得到，求解目标往往是求图中或是两点之间所有路径的权的最优值。

有的权则表示容量或是流量，它们的运算特征是“串联求最值，并联求和”，即一条路径上最大或是最小的权决定了整条路径的权，而求解目标则是求图中或是两点之间所有路径的权的加和。

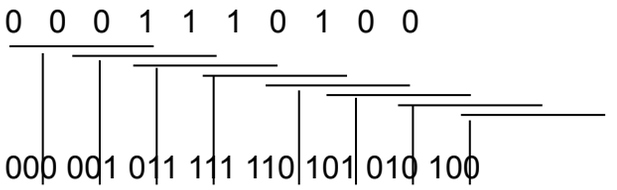
以上只是基本的几类权，它们还会产生一些变形，例如权的运算由简单的相加、求最值扩展到相乘，或是更复杂的函数计算等等。

还有的图不仅包含边权（边集 E 到实数集 R 的映射），还包含点权（点集 V 到实数集 R 的映射）；或是包含好几类不同性质的权。

以上这些差异形成了图论模型的多样化，使图论模型可以广泛地适应各类问题，但这些丰富的选择同时也增加了图论建模的难度。对于有些题目，我们可能很自然地就联想到某种图论模型，例如看到表达式，就会联想起表达式树；但对于另一些题目，分析的角度不同，就会得出不同的模型，产生不同的效果。

【例2】奶牛排队 Cows on Parade（文档附件：Finals95.htm）（ USACO'95 决赛）：

[问题简述]：求一个长度为 2^n+n-1 的 01 序列，要求序列中包含所有长度为 n 的 01 排列(共



个 2^n)。当 n=3 时，序列如左图所示。

与此类似的还有“鼓轮设计”一题（见参考书籍[9]P95），不同的是前者要求排成一线，而后者要求排成一圈，显然，只要后者解决了，前者也就解决了。所以这里也以排成一圈为求解目标。

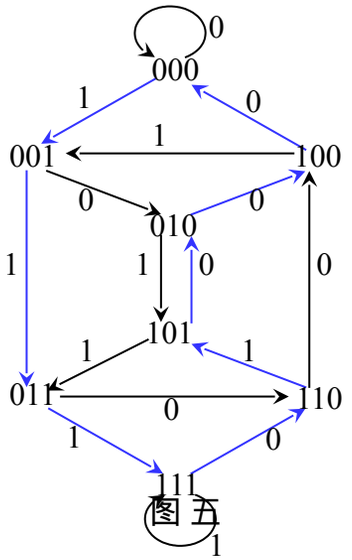


图 五

[模型 A]

如果以每个长度为 n 的 01 排列为点，以它们之间的重叠关系为边，就构成了一个有向图（如图五所示）。求解目标就是在这个图上找一个回路（图中蓝色部分），使得通过每点一次且仅一次——**Hamilton 回路**问题。

[模型 B]

如果以 01 排列之间的重叠部分为点，以 01 排列为边，就构成了一个如图六所示的有向图，求解目标就是在这个图上找一个回路，使得不重复地遍历每条边——**Euler 回路**问题。

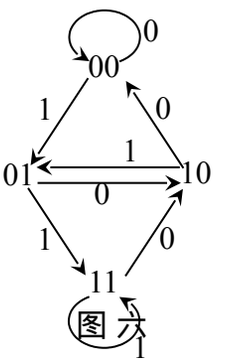


图 六

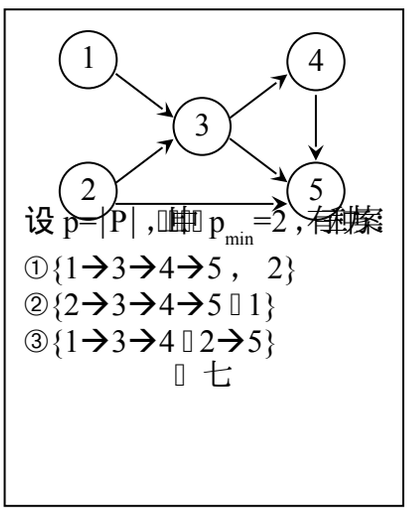
从图六中不难看出，图上每点都是连通的，而且入度和出度都等于 2，因此图中存在 Euler 回路，也并不难求（算法及相关理论见参考书籍[6]P152 或[8]P246 或[9]P95）。相较之下，Hamilton 回路问题是 NP 完全问题，也就是说，模型 A 的建立并没有给问题的解决带来任何便利。

由此可见，不同理论体系的图论模型很可能会产生完全不同的效果。

例 2 中不同模型的建立是由于分析角度的不同，而在下面的例 3 中，则是因为对原型的分析深浅不同的缘故。

【例3】 最少路径覆盖（题目见参考书籍[1]P274 或[3]P429）：

有向无环图 G(V,E)的一个路径覆盖是指一个路径集合 P，满足图中的每点属于且仅属于集合中的一条路径。求一个包含路径数最少的路径覆盖（如图七所示）。



七

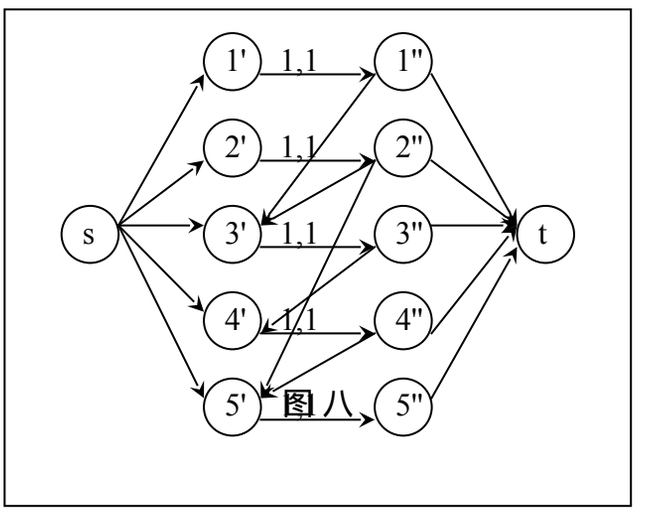


图 八

[解法A] **网络流模型**（见图八）：

1． 把点 i(i∈V)拆成两点 i'和 i"，添加弧（即有向边）(i', i")，容量上界 $C_{i',i''}=1$，下界 $B_{i',i''}=1$；

2． 加源点 s 和汇点 t，添加弧(s, i')和(i", t)，$C_{s,i'}=C_{i'',t}=+\infty$，$B_{s,i'}=B_{i'',t}=0$；

3． 把边集 E 中的弧(i, j)改为弧(i", j')，$C_{i'',j}=+\infty$，$B_{i'',j}=0$。

建立了新图 G'之后，原图 G 中的一条路径就相当与 G'中的一条流路径。因为我们规定 G'中 $C_{i',i''}=B_{i',i''}=1$，所以原图 G 中的点 i 属于且仅属于一条路径，因此 G 中包含路径最少的一个路径覆盖方案就相当于 G'中的一个最小流方案。

[解法B] 二部图模型

设路径覆盖 P={P_1，P_2，.....，P_{p-1}，P_p}，$P_i(1\le i\le p)$表示集合 P 中的一条路径；记 $V_i=P_i\cap V$，$E_i=P_i\cap E$，即 V_i 为 P_i 中的点，E_i 为 P_i 中的边，显然$|\mathbf{V_i}|=|\mathbf{E_i}|+1$。

例如在图七中，第①种方案为 P={1→3→4→5，2}，其中 P_1=1→3→4→5，所以 V_1={1，3，4，5}，E_1={(1,3)，(3,4)，(4,5)}。

因为 P 是一个路径覆盖，所以 $\mathbf{V_1\cup V_2\cup\cdots\cdots\cup V_{p-1}\cup V_p=V}$，且 $\mathbf{V_i\cap V_j=\varnothing}(1\le i,j\le p，i\ne j)$。这些条件加上一点集合方面的知识，我们不难有以下的推导：

$$\because V_1\cup V_2\cup\cdots\cdots\cup V_p=V,$$

且$V_i\cap V_j=\varnothing\ (1\le i,j\le p，\ i\ne j)$，

$$\therefore |V|=\sum_{i=1}^{p}|V_i|$$

$$=\sum_{i=1}^{p}(|E_i|+1)$$

$$=p+\sum_{i=1}^{p}|E_i|,$$

$$\therefore p=|V|-\sum_{i=1}^{p}|E_i|$$

又∵$E_i\cap E_j=\varnothing(1\le i,j\le p，i\ne j)$,

$$\therefore \sum_{i=1}^{p}|E_i|=\left|\sum_{i=1}^{p}E_i\right|，$$

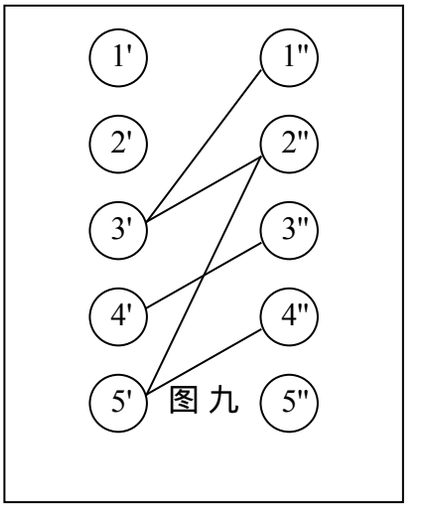


图 九

记 M=$\sum_{i=1}^{p}E_i$，则 M 需要满足"***V 中的每一点最多只与 M 中的两条弧相连：一条入弧和一条出弧***"，并且只要 M 满足这个条件，它与路径覆盖就是一一对应的。

有了以上的分析之后，我们就可以建立起一个**二部图模型**（见图九）了：

1． 把点 i(i∈V)拆成两点 i'和 i";

2． 把边集 E 中的弧(i, j)改为无向边(i", j')。

在新图 G'中, 点 i'为入点，原图 G 中所有以点 i 为弧头的弧都改成了与 i'相连的无向边；同样的，点 i"为出点，G 中所有以点 i 为弧尾的弧都改成了与 i"相连的无向边。这时，原图 G 上的 M 就相当于二部图 G'上的一个匹配。

上面我们已经分析出 M 与 P 是一一对应的，且|P|=|V|-|M|，而|V|又是定值，所以只要求出二部图 G'上的最大匹配，也就求得原图 G 的最小路径覆盖了。

比较上面两个解法的时空效率和编程复杂度，不难看出解法 B 要明显优于解法 A。这是因为解法 B 对原型进行更深入的分析，只抽取了原型中较关键的要素来建立模型，所以求解起来更为简单、方便。

另外值得注意的一点是：解法 A 中用到了网络流算法。由于这类算法一般编程复杂度大，易错，而且算法的系数较大，所以它在竞赛中的表现往往不十分出色。但是网络流模型可以容纳的要素很多，特别是权的类型众多：不仅有表示容量和流量的权，还有表示费用的权；容量不仅有上界，还可以有下界.....有了这些多变的因素，再加上网络流算法一般都属有效算法，所以网络流模型在现实问题的解决和图论问题的研究中有着十分广泛的应用。在下面的例题中，也还有不少用到了网络流模型。

从上面的例子中，我们已经了解到了选择合适的理论体系的重要性。有时，可能会有多个看似都正确的模型可供选择，只有仔细分辨出它们之间细微但又关键的差别，才能够选出适于求解的模型。这就要求我们既要熟悉图论的算法和理论，也要对原型有着清楚的认识。

无论是要决定要素的取舍，还是面对着不同理论体系的选择，总的原则都是：**准确、清晰、简明**。

“准确”：指模型应当忠于原型，不可扭曲原型本来的性质，也不可默认原型具有一些假设的、未经证明的性质；

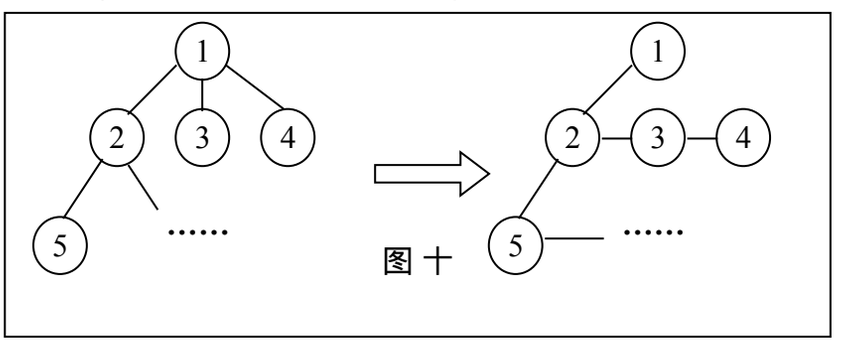
“清晰”：就像程序具有可读性一样，图论模型也具有清晰度，以结构清晰的模型为分析对象，思考复杂度就会降低很多；

“简明”：指模型中含有的要素简单明了，不繁杂，模型也是便于分析求解的，有这一要求主要是因为图论中很多算法和理论都相当高深和复杂，很难运用和变通，因此尽量不要采用。

三. 图论模型的转化

在建立图论模型解决问题时，我们往往会遇到这样的情况：在很轻松地建立起一个模型之后，却不知该如何分析，求解，或是发现建立起的模型无法表现原型的一个很重要的性质。这时，我们必须要放弃现有的模型吗？不，如果运用一些方法和技巧的话，现有的模型很可能就会转化为能够准确描述原型，而又易于求解的新模型。

在这些方法和技巧中，有一些只适用于特定的图论模型，例如把多叉树转化为二叉树（见图十）处理的技巧。此外，还有另一些作用范围更广的方法和技巧，下面将讨论的“拆分转化”就是其中之一。



图十

“拆分转化”可分为三类：点→边、点→点、边→边。其中第一类很常用，但拆法较简单；第二类不仅常用，而且灵活多变；第三类的拆法和作用则与第二类很相似。所以下面将详述第二类，简述第一、三类。

1．点→边：

“拆点为边”是“拆分转化”中十分常用的一类，当要求现有模型中的**点具有边的性质**时，“拆点为边”就是一个好办法。

在例 3 中建立模型 A 时，我们把每个点拆成了一条有上下界容量的边，以满足“每个点经过且只经过一次”的要求。另外，我们在求解点有容量的网络流时，为了得到标准的网络流模型，往往也把点拆成边，把点上的容量限制转移到对应的边上。

在实际应用中，“拆点为边”的例子很多，“拆法”也大同小异：把每点 i 拆成两个点 i'和 i"；以 i 为弧头的弧都改为以 i'为弧头，以 i 为弧尾的弧都改为以 i"为弧尾（这些弧上的权以及原有属性是保留还是更改，则视情况而定）；并且添加一条 i'指向 i"的弧，把点 i 具有的

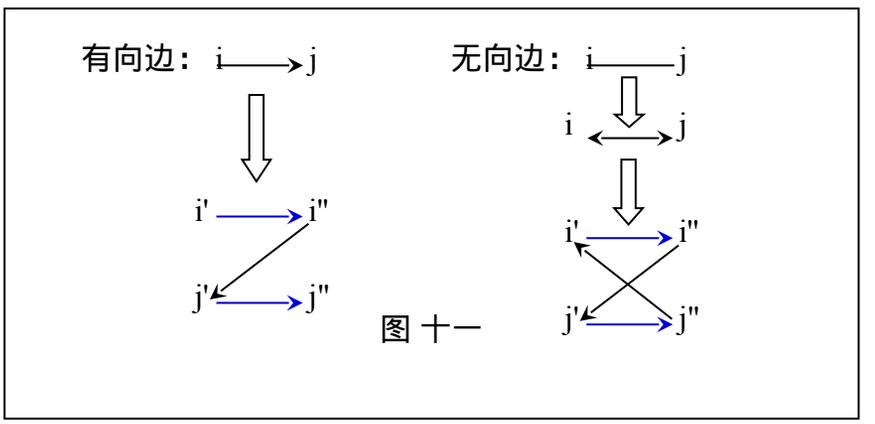


图 十一

属性赋给这条弧（参见图十一，其中蓝色的弧为添加的弧）。

2．点→点：

这是“拆分转化”的第 2 类：把一个点拆成若干个点。它主要用于把一个点拥有的好几个不同的性质分离开，分别赋给几个新的点，使每个点的性质单一，再通过一些改造来统一处理它们。

例如在例 3 中建立模型 B 时，为了表示任意一点 i 最多允许与一条入弧和一条出弧相连，我们把 i 拆成了一个入点 i'和一个出点 i"，并且通过把所有的入弧都连到 i'上，所有的出弧都连到 i"上的措施，使每点都只与一条边相连，这样，用匹配算法就可以统一处理了。

拆点的方式多种多样，远不止上面一种，下面的例子中用到的就与例 3 不同。

【例4】 障碍物探测器 Mars explorer（文档附件：Mars.html）（ IOI’97）：

SOOOOOOT 图十二

[问题简述]：

在 P×Q 的矩形中，每个格子的地形可能是下列三种之一：

1． 障碍：无法通过；
2． 平地：平坦无物，可通过；
3． 石块：可通过，第一次通过时可采到一块岩石；

求 M 条的路径（M 给定），使得路径能够覆盖到石块数总和最大，路径要满足的条件是：

1． 每条路径都是从(1,1)到(P,Q)，途中每步只能向东或向南走一格；
2． 路径不通过障碍；

[建立网络流模型]：

我们按照常规思路把每条路径看作流量为 1 的流路径，并根据路径要满足的两个条件，建立如下模型：

1． 把每个格子看作点，并引两条弧，分别指向其东侧和南侧相邻的点（如果有的话），弧的流量表示通过这条弧的路径数，所以容量为+∞（≥m 就可以了）；

2． 删去与障碍点相连的弧。

但这只是一个初步的模型，它很显然没能区分“平地”和“石块”。难点主要就在于：岩石只有第一次通过时可以采到，之后这个格子就变成平地了。因此，就需要把含有岩石的点 i 拆成两个点：i'和 i"，分别表示岩石和平地。具体改造步骤（如图十三所示）如下：

1． i"继承与 i 相连的所有入弧和出弧，且容量不变，收益为 0；

2． 从 i"引出一条弧，指向 i'，令其容量为 1，收益为 1；

3． i'继承与 i 相连的所有出弧，容量改为 1（>1 也可以），收益为 0；

经过上述转化之后，原问题的求解目标就对应于新模型上流量为 M 的最大收益流[2]。

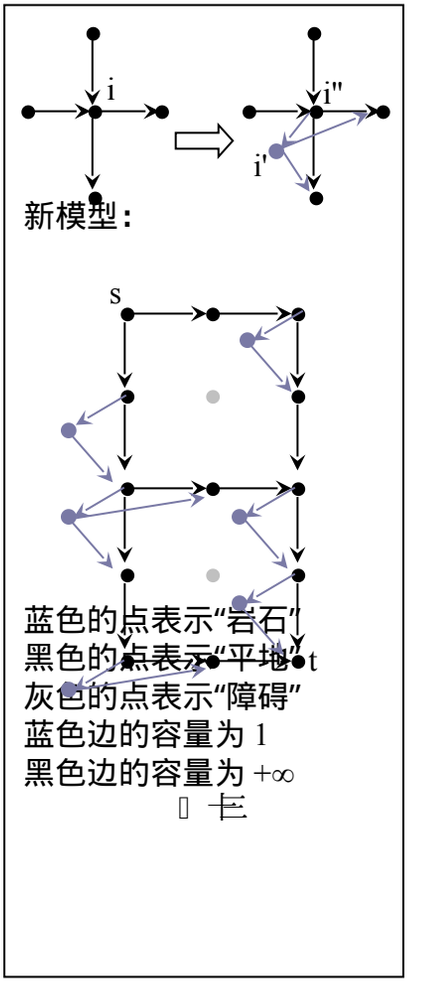


图 十三

上面举的两个例子都是把一个点拆成两个点，但这与“拆点为边”是截然不同的。这里是为了把一个点具有的各种性质分离开，有几种性质需要分离，就要拆成几个点；而“拆点为边”是为了使点具有边的性质，所以总是固定地把一个点拆成两个，并在两个新点之间添边。

3．边→边：

这是“拆分转化”的第 3 类：把一条边拆成若干条边。它和上面的第 2 类一样，也是用于因素分离。下面就只简单地举例分析一下。

【例5】 容量有上下界的网络流：

这是一个标准的网络流问题，许多书上都有该问题的解法和相关证明，所以这里就只简述一下把原图转化为一般的最大流模型的过程[3]：

1． 加两个新顶点：附加源 s'和附加汇 t'，作为新的源和汇；

2． 把原图上的弧(i,j)拆成三条新弧：(i,j)、(s',j)、(i,t')，令 $C'_{i,j}=C_{i,j}-B_{i,j}$，$C'_{s',j}=C'_{i,t'}=B_{i,j}$（$C_{i,j}$、$B_{i,j}$ 分别为原图上弧(i,j)的容量上、下界，$C'_{i,j}$、$C'_{s',j}$ 和 $C'_{i,t'}$ 为新弧的容量上界）；

3． 在原来的源和汇之间添加两条容量为+∞的新弧：(s,t)和(t,s)。

在以上三步中，第二步最为关键，它实现了把原图上的容量上界和下界分离开的目的（如图十四(a)所示）。图十四(b)和(c)分别是转化前的原图和转化后的新图。

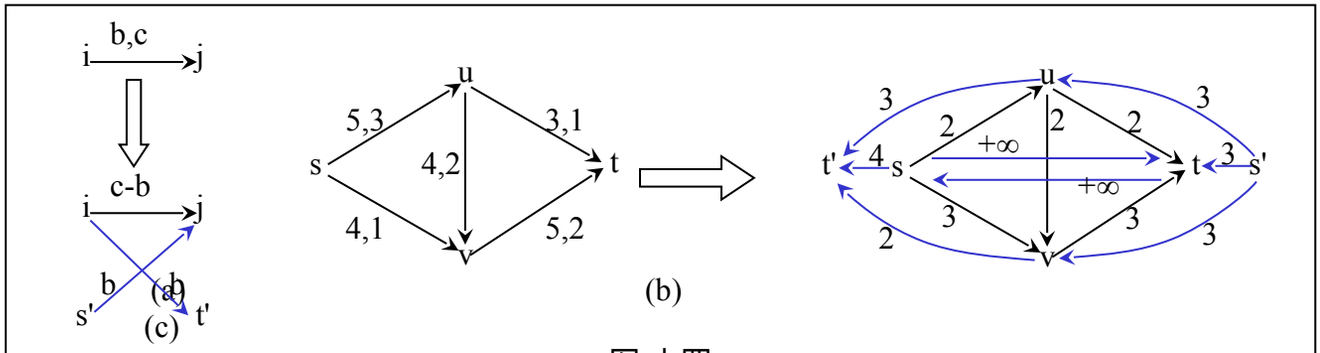


图 十四

[2] 后面我们将通过“补集转化”把最大收益流转化为最小费用流求解。

[3] 至于如何根据新模型上的最大流求得原图的可行流，请见参考书籍[6]P68 或[8]P165。

更进一步地，如果点（或边）的性质随着时间的推移而变化着，我们也可以用类似于第 2 类或第 3 类的方式把它们拆成许多点（或边），以表示不同时刻不同性质的点。下面我们就来看一看这类“拆分转化”是如何应用到例 6 上的。

【例6】家园 Homeland（CTSC '99，下面只简述建模的过程，具体算法分析请见文档附件：homeland..doc）：

仔细分析了题意之后，不难想到建立网络流模型来解决这题：

1． 以太空站为点；
2． 以往返于太空站之间太空船为边；
3． 以船的最大载客量为容量上界；

求解目标就是在最短时间内把固定流量（人）从源（地球）送到汇（月球）。但这里的时间不同于费用，而且边随着时间 t 的变化存在于不同的点对之间。因此按时间拆点就很有必要：把每个点都拆成 t 个点；把边按照时刻的不同分配在相应的新点对间。

如此一来，原本动态变化着的模型就被转化成了静态不变的新模型了。

从上面举的几个例子可以看出，要分离的性质不同，拆点（边）的方式也就随之不同，只有准确地分析出点（边）要表示的性质，才能转化得到合适的模型。

通过对这三类点和边的“拆分转化”的分析和总结不难发现，它们的目的和应用范围各异，但方法都是一个“拆”字。在它们的应用上，我们完全不必拘泥于具体形式，只要是建模的需要、解题的需要，就可以按需“拆分”。只要“拆分”得当，上面的三类“拆分转化”就可以应用得更广。

除了“拆分转化”之外，“补集转化”也是一种不依赖于特定模型的转化方法。

前面在例 4 的分析中，我们留下了一个问题：把流量固定的最大收益流转化为最小费用流求解。这就要求对边权进行一定的修改：改所有表示“平地”的点对之间的弧（图十三中黑色的弧）的费用为 1；改所有指向“岩石”或从这些点指出的弧（蓝色的弧）的费用为 0。这样，原模型上每条流路径的收益就等于 P+Q-2（该路径的长度）减去路径上的费用的差。如此也就实现了把最大收益转化为最小费用的目的。

这种用某个特定值减去原权值的转化技巧，我们就称作***权的“补集转化”***。

既然有权的“补集转化”，自然也有**点**和**边**的“补集转化”。点和边的“补集转化”是指用全集（如图的点集、边集或是其它集合）减去某个特定的集合（如题目给定的集合或求解目标集合等）的转化技巧。下面我们就通过具体例子来分析一下。

【例7】最大独立集问题及其等价问题：

最大独立集的问题描述如下：给定无向图 G(V,E)，其中 A⊆V，且 E∩{(i,j)|i,j∈A}=∅，求 A，使得|A|最大。它还有两个与之等价的问题：

1． 最小覆盖问题：

一般地，我们都把最大独立集转化为最小覆盖集，然后根据逻辑运算定律求解（见参考书籍[6]P57）：令 B=V－A，则 B⊆V，且对于任意一条边(i,j)∈E，有 i∈B 或 j∈B，所以 B 是图 G(V,E)的最小覆盖集。在这个转化过程中就用到了点的“补集转化”——用点集 V 减去求解目标集合 A，以得到新的目标集合 B。

2． 最大完全子图问题：

[问题描述] 给定无向图 G(V,E)，其中 C⊆V，对于任意两个点 i,j∈C，且 i≠j，有(i,j)∈E，求 C，使得|C|最大。

当我们已经知道并且能够证明这个问题是 NP 完全问题（见参考书籍[3]P658）时，只要能够把它转化为最大独立集问题求解，也就证明了后者是 NP 难度的，而这一步转化中就要用到边的“补集转化”：令全集 U={(i,j)∣i,j∈V}，E'=U－E，则在无向图 G'(V,E')中，有 C⊆V，且 E∩{(i,j)∣ i,j∈C}=∅，所以 C 也是图 G'的最大独立集。

例如图十五 G'中，黑色的点构成了最大独立集，而白色的点就是最小覆盖集；图 G 的点集与 G'一样，而边集 E 则是 E'的补集，图中黑色的点构成了该图的最大完全子图的点集。

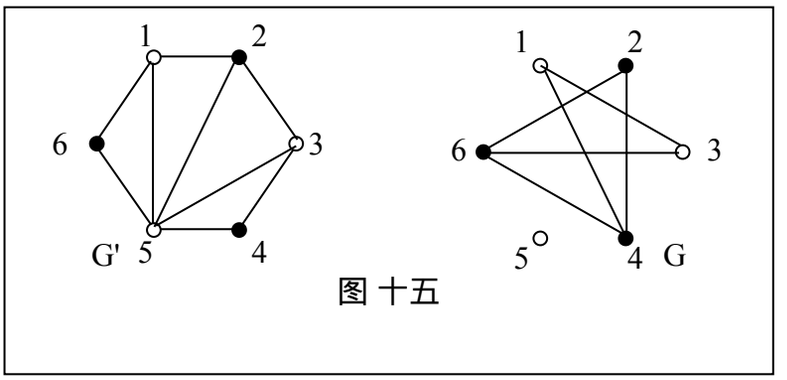


图 十五

从上面的例子可以看出，“补集转化”往往都是等价转化，而且往往都会使求解目标产生一定的变化。无论是点、边还是权的“补集转化”，目的都是使模型向着更便于思考和求解的方向转化。

除了上面分析的“拆分转化”和“补集转化”外，图论模型的转化方法和技巧还有很多，而且它们还可以综合起来运用，就像在例 4 中一样，经过了几次模型的转化，最终得到了需要的图论模型。

四. 结语

本文主要分析了图论模型建立中的两个要点和图论模型转化的几种方法技巧。在实际的建模过程中，它们是密不可分的：正确提取原型中的要素；对应到特定理论体系中相应的元素上；建立起初步的模型；然后根据需要进行适当的转化。至此，一个适于求解的图论模型才建立成功。在这其中，无论是对建模原则的把握，还是模型转化方法的运用，都遵循着一点：原型本身的性质决定了模型。如果硬要把原型套到不合适的模型上去，往往反而会破坏原型的关键性质，这时，即使建立的模型再怎么巧妙、经典，也是经不住考验的。

图论算法和理论十分独特精妙，然而难于随心所欲地运用；图论模型的建立和转化十分灵活，因而难于掌握。因此，对图论模型的研究并非一朝一夕的事，需要持之以恒。本文只是我个人对图论建模的一点浅显认识，也只是一个开始。我相信，随着认识的进一步加深，集思广益，图论模型一定有更广阔、更精彩的应用。

【附录】

本文的重点在于图论建模，而模型的解答、解释和检验等不在本文讨论范围之内。文中设计到的图论算法和理论在下列书籍中都有详细介绍，因此文中也只加以简要的分析。

【参考书目】

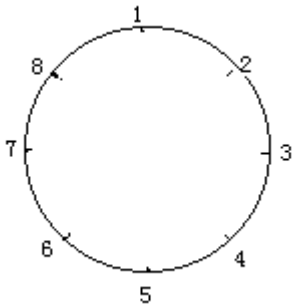
[1] 吴文虎、王建德，实用算法的分析与程序设计，电子工业出版社，1998
[2] 任善强、雷鸣，数学模型，重庆大学出版社，1998
[3] 潘金贵、顾铁成等，现代计算机常用数据结构和算法，南京大学出版社，1994
[4] 吴文虎、王建德，国际国内青少年信息学（计算机）竞赛试题解析（1994 ~ 1995），清华大学出版社，1997
[5] 吴文虎、赵鹏，1993 ~ 1996 美国计算机程序设计竞赛试题与解析，清华大学出版社，1999
[6] 吴文虎、王建德，青少年国际和全国信息学（计算机）奥林匹克竞赛指导——图论的算法与程序设计，清华大学出版社，1997
[7] 1999 国家集训队资料
[8] 谢政、李建平，网络算法与复杂性理论，国防科技大学出版社，1995
[9] 倪兆中，集训队作业论文选编，1998

安徽 徐静
2000.1.15

导线排布(Line)

问题描述：

ACM 实验室最近正在试验一种新型的导线转接装置，这种导线转接装置是一个圆形的盒子，在圆周上平均分布着 2N 个转接点。现在有 N 根导线需要连接在这个装置上。由于特殊的需要，某些导线必须交叉。由于导线的数量过多，情况太复杂，人力难以胜任，所以请你编一个程序，帮助工作人员排布这些导线。



一个圆圈上共有 2N 个点（在上图中，N=4），分别编号 1,2,3…..2N-1,2N。以这 2N 个点为端点可连接 N 条导线，每个端点只能够被一个导线所连结。每条导线均有自己的编号，线的编号是按端点中编号较小的点的编号排序的。已经知道了各个线之间的交叉情况，请你帮助工作人员找出一种可能的方法去安排这些导线，使其符合要求。

输入输出要求：

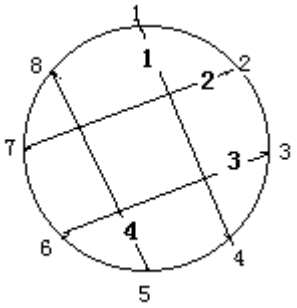
输入数据：
文件名为 Input.txt
第 1 行为总共的线数 N (1<=n<=400)
第 2 行为总共的交叉点 x 数量 M (1<=M<=N *(N-1))
第 3…M+2 行为每个交叉的两个线的编号，前面的那根线的编号小于后边那根线的编号。这些交叉已经排序(第一条线编号小的交叉在前，第一条线编号一样的则第二条线编号小的在前)。
输出数据：
文件名为 Output.txt
输出数据共有 N 行，每行两个数，分别表示每根导线的两个端点的编号。

样例数据：

输入数据样例：
4
4
1 2
1 3
2 4
3 4

输出数据样例
1 4
2 7
3 6
5 8

下图就是以上输入数据对应的图。



评分标准：

在规定的时间内，如果程序能够正常退出，并且得到正确的解答，则能够得到此测试点的分值。所有测试点规定时间最长不超过 15 秒。

家园
Homeland

题意及算法分析

仔细分析题意，我们不难看出本题具有一个流网络应有的各个要素：

- 源──地球：0
- 汇──月球：-1
- 中间点──太空站：1~ n
- 边──往返于各个太空站之间的太空船
- 边的容量──太空船最多可容纳的人数。

所不同的是，本题还多出一个时间因素，并且是给定流量，要求最短时间。

我们面临着难题：一般的流网络无法表示两点间不同时刻的不同容量和流量。

因此，就需要对流网络进行改造：拆点──

如果最短时间为 T，我们就把每个点 V_i（包括源和汇）拆成 $V_{i,0}$、$V_{i,1}$、.....、$V_{i,T}$这 T+1 个点：这样，太空船在 t-1 时刻从 V_i出发($t \le T$)，t 时刻到达 V_j的动作就可以用边 $V_{i,t-1} \rightarrow V_{j,t}$来表示，如果 $C(V_{i,t-1}, V_{j,t})=a$，$f(V_{i,t-1}, V_{j,t})=b$，那么就表示太空船可以容纳 a 个人，而此时正载有 b 个人；又因为每个太空站可以容纳无限的人停留，所以对于每个 $V_{i,t-1}$，都应有一条容量不限的边指向 $V_{i,t}$。

这样改造之后，源就是 $V_{0,0}$点，汇就是 $V_{-1,t}$点，从源到汇的流量就表示到时刻 T 为止，从地球到月球最多可以输送的人数。

我们从小到大地尝试每个 T 值，第一个能够把 k 个人全部送达月球的时刻就是最优解。

在算法的实现上，我们采用宽度搜索找增广路径的算法（Edmonds-Karp 算法）来求最大流，并且，每当 T 值加 1 时，我们只需在原有流量的基础上进行增广，而无须重新求解。

算法的空间复杂度为 $O(T \times n^2)$。由于 T 最大可达到 1000 以上（目前找到的最大值为 1159），所以在 TP7 下只保存得下流量，容量网络需要在求解过程中重建（好在并不太麻烦）。

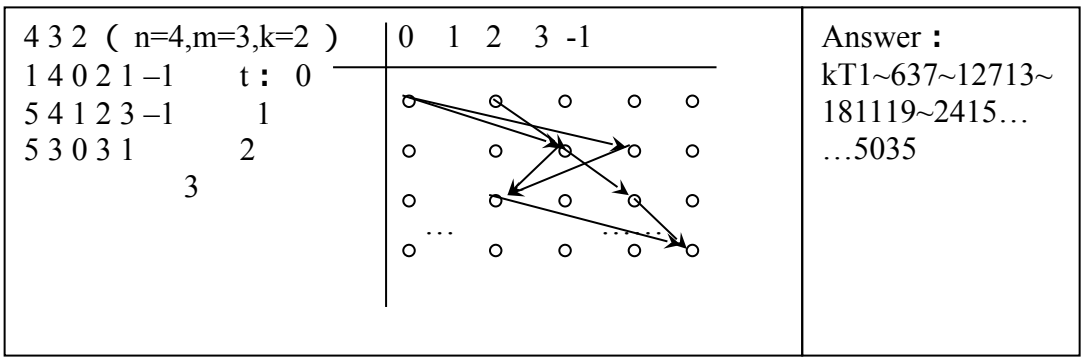
算法的时间复杂度为 $O((n \times T)^3)$：求解的全过程中找不到增广路径的宽度搜索次数为 O(T)（因为一旦找不到增广路径， T 值就会加 1），找到增广路径的为 O(|V|×|E|)，一次宽度搜索的复杂度为 O(|E|)，把它们综合起来，并代入|E|=(m+n)×T── m 表示 m 艘太空船某一时刻必然正从一个太空站驶往另一个，n 表示 $V_{i,t-1}$指向 $V_{i,t}$的边──和|V|=n×T，即得 O(T

$^3 \times n \times (m^2+n^2))$，从题目所给范围（n≤20,m≤13）看，我们也可以把 m 视为 O(n)，所以时间复杂度可简写为 $O((n \times T)^3)$。

一个猜想

在构造测试数据的过程中，我发现最优解 T 随着人数 k 的增加而有规律地增加，即当 k 增加一个特定的值 d 时，T 增加一个特定的值 t，写成表达式即：$T = t \times \left\lfloor \frac{k+a}{d} \right\rfloor + b$（t>0,d>0,a∈[0,d),b 无限制），其中 t、d、a、b 由流网络的结构和容量决定。

例如右图所示的数据，图中给出了对应的改造后的流网络（略去了边上的容量和所有 $V_{i,t-1}$ 指向 $V_{i,t}$ 的边），数据对应的答案列于右表，从中可以看出这个流网络对应的 t=4、d=6 、a=5、b= -1，即 $T = 4 \times \left\lfloor \frac{k+5}{6} \right\rfloor - 1$。



上面只是我在实践中发现的一点现象，但限于理论水平不够，还无法解释或是证明这一点。不过，如果它成立的话，我们至少可以先解 k 值较小的情况（用时很少），求出 t、d、a、b 的值，再计算出 k 值很大时的解，或许我们仅根据输入的数据就可以直接计算了。

源程序及注释：

```
{$A+,B-,D+,E-,F-,G+,I+,L+,N+,O-,P-,Q-,R-,S-,T+,V-,X+}
{$M 16384,0,655360}
program Escape;
const
    Infns='Homeland.in';
    Outfns='Homeland.out';
    LimitM=13;
    LimitN=20;
    LimitK=50;
    LimitTime=1300;  { 时间最大值 }
```

```
    LimitNode=500;  { 用于求最大流的宽度搜索算法 }
    LImitDanger=LimitNode-50;
type
    TMap=array[0..LimitN,-1..LimitN] of byte;
    TTimeMap=array[0..LimitTime] of ^TMap;
    TStationBool=array[-1..LimitN] of boolean;
    TShip=record
            content,moor:byte;
            mooring:array[1..LimitN+2] of shortint;
          end;
    TSpaceShip=array[1..LimitM] of TShip;
    TNode=record
            delta,station:shortint;
            level:word;
          end;
    TLinearNode=array[1..LimitNode] of TNode;
    TStationInt=array[0..LimitTime,-1..LimitN] of shortint;
var
    Ship     :TSpaceShip;     { 太空船 }
    Flow         :TTimeMap;       { 记录每一时刻每条边的流量 }
    Father   :TStationInt; { 求最大流时用于记录每点在增广路径上的父结点 }
    OpenList:^TLinearNode;   { 求最大流时记录宽度搜索的待扩展结点 }
    n,m,k    :byte;
    pre,now,
    head,tail :word;         { 待扩展队列的头、尾指针 }

procedure Init;    {读入}
  var
    f:Text;
    i,j:byte;
  begin
    assign(f,Infns); reset(f);
    readln(f,n,m,k);
    for i:=1 to m do
      with Ship[i] do
        begin
          read(f,content,moor);
          for j:=1 to moor do    { 读入太空船信息 }
            read(f,mooring[j]);
          readln(f);
        end;
    close(f);
  end;

procedure Out(totaltime:word); { 输出 }
  var
    f:text;
  begin
    assign(f,Outfns);rewrite(f);
    writeln(f,totaltime);
```

```
    close(F);halt;
  end;

function NoAnswer:boolean;      { 判断无解情况 }
  var
    i,j:byte;
    flag:boolean;
    Touch:TStationBool;
  begin
    fillchar(Touch,sizeof(Touch),false);
    Touch[0]:=true;
    repeat
      flag:=true;
      for i:=1 to m do
        with Ship[i] do
          for j:=1 to moor do
            if Touch[mooring[j]] and (not Touch[mooring[j mod moor +1]])
              then begin
                     Touch[mooring[j mod moor +1]]:=true;
                     flag:=false;
                   end;
    until flag;
    NoAnswer:=not Touch[-1];
  end;

function min(a,b:byte):byte;       {求 a,b 中的较小者}
  begin
    if a<=b then min:=a
            else min:=b;
  end;

procedure IncFlow(delta:shortint;level:word);
{ 根据已找到的一条增广路径进行增广：delta 为增加的流量 }
  var
    i,fa:shortint;
  begin
    i:=-1;      dec(k,delta);  { 剩余的 k 人中又有 delta 人得到安排 }
    repeat
      fa:=Father[level,i];
      if fa>=0
        then begin     { 对正向弧进行增广 }
               inc(Flow[level-1]^[fa,i],delta);
               i:=fa;       dec(level);
             end
        else begin     { 对反向弧进行增广 }
               dec(Flow[level]^[i,-fa],delta);
               i:=-fa;      inc(level);
             end;
    until i=0;
  end;
```

```
function Open:boolean;     { 扩展结点 head，open=true 表示未找到增广路径，需要继续扩展结点 }
  var
     Map:array[-1..LimitN] of shortint;
     i,which,t:shortint;
     nextlevel:word;
  begin
  with OpenList^[head] do
     begin
       fillchar(Map,sizeof(Map),0);
       Map[station]:=LimitK;
       for i:=1 to m do   { 求得从(level,station)点出发的所有弧的容量 }
          with Ship[i] do
            begin
               which:=level mod moor +1;
               if mooring[which]=station
                 then inc(Map[mooring[which mod moor +1]],content);
            end;
       if Map[-1]>Flow[level]^[station,-1] then   { 能够增广 }
          begin
            Father[level+1,-1]:=station;
            IncFlow(min(delta,Map[-1]-Flow[level]^[station,-1]),level+1);
            Open:=false;        exit;
          end;
       Open:=true;
       nextlevel:=level+1;
       if level<pre then
          for i:=0 to n do            { 找正向弧 }
            if (Father[nextlevel,i]=100) and (Map[i]>Flow[level]^[station,i]) then
               begin
                 Father[nextlevel,i]:=station;
                 inc(tail);
                 OpenList^[tail].delta:=min(delta,Map[i]-Flow[level]^[station,i]);
                 OpenList^[tail].level:=nextlevel;
                 OpenList^[tail].station:=i;
               end;
       if level=0 then exit;
       nextlevel:=level-1;
       for i:=1 to n do    { 找反向弧 }
          if (Father[nextlevel,i]=100) and (Flow[nextlevel]^[i,station]>0) then
            begin
               Father[nextlevel,i]:=-station;
               inc(tail);
               OpenList^[tail].delta:=min(delta,Flow[nextlevel]^[i,station]);
               OpenList^[tail].level:=nextlevel;
               OpenList^[tail].station:=i;
            end;
     end;
  end;
```

```
procedure SearchWay;      { 宽度搜索 }
  var
      i,j:word;
      flag:boolean;
  begin
      repeat
        fillchar(Father,sizeof(Father),100);
        head:=1; tail:=1;
        with OpenList^[1] do  { 从源出发 }
            begin
              delta:=k; level:=0; station:=0;
            end;
        flag:=true;
        while (head<=tail) and flag do
            begin
              flag:=Open;  inc(head);
              if tail>LimitDanger then    { 队列平移：充分利用队列空间 }
                  begin
                    move(OpenList^[head],OpenList^[1],(tail-head+1)*sizeof(TNode));
                    dec(tail,head-1); head:=1;
                  end;
            end;
      until flag or (k=0);       { 找不到增广路径或是已找到最优时间 }
  end;

begin     {主程序}
      Init;
      if NoAnswer then Out(0);
      New(OpenList);
      for now:=1 to LimitTime do {累试每个时刻}
        begin
            pre:=now-1;
            new(Flow[pre]);
            fillchar(Flow[pre]^,sizeof(Flow[pre]^),0); {为流量赋初值}
            SearchWay;
            if k=0 then Out(now);
        end;
end.
```