“十二五”普通高等教育本科国家级规划教材

国家精品课程教材

计算机算法设计与分析习题解答

（第5版）

◎ 王晓东　编著

中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

“十二五”普通高等教育本科国家级规划教材
国家精品课程教材

计算机算法设计与分析习题解答

（第5版）

王晓东 编著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书是与“十二五”普通高等教育本科国家级规划教材《计算机算法设计与分析（第5版）》配套的辅助教材和国家精品课程教材，分别对主教材中的算法分析题和算法实现题给出了解答或解题思路提示。为了提高学生灵活运用算法设计策略解决实际问题的能力，本书还将主教材中的许多习题改造成算法实现题，要求学生设计出求解算法并上机实现。本书教学资料包含各章算法实现题、测试数据和答案，可在华信教育资源网免费注册下载。

本书内容丰富，理论联系实际，可作为高等学校计算机科学与技术、软件工程、信息安全、信息与计算科学等专业本科生和研究生学习计算机算法设计的辅助教材，也是工程技术人员和自学者的参考书。

图书在版编目（CIP）数据

计算机算法设计与分析习题解答/王晓东编著. —5版. —北京：电子工业出版社，2018.10

ISBN 978-7-121-34438-1

Ⅰ. ① 计… Ⅱ. ① 王… Ⅲ. ① 电子计算机－算法设计－高等学校－题解 ② 电子计算机－算法分析－高等学校－题解 Ⅳ. ① TP301.6-44

中国版本图书馆CIP数据核字（2018）第120711号

策划编辑：章海涛
责任编辑：章海涛
印　　刷：三河市良远印务有限公司
装　　订：三河市良远印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编　100036
开　　本：787×1092　1/16　　印张：22.75　　字数：580千字
版　　次：2005年8月第1版
　　2018年10月第5版
印　　次：2018年10月第1次印刷
定　　价：56.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式：192910558（QQ群）。

前　言

一些著名的计算机科学家在有关计算机科学教育的论述中认为，计算机科学是一种创造性思维活动，其教育必须面向设计。“计算机算法设计与分析”正是一门面向设计，且处于计算机学科核心地位的教育课程。通过对计算机算法系统的学习与研究，理解掌握算法设计的主要方法，培养对算法的计算复杂性正确分析的能力，为独立设计算法和对算法进行复杂性分析奠定坚实的理论基础，对每一位从事计算机系统结构、系统软件和应用软件研究与开发的科技工作者都是非常重要和必不可少的。课程结合我国高等学校教育工作的现状，追踪国际计算机科学技术的发展水平，更新了教学内容和教学方法，以算法设计策略为知识单元，在内容选材、深度把握、系统性和可用性方面进行了精心设计，力图适合高校本科生教学对学时数和知识结构的要求。

本书是“十二五”普通高等教育本科国家级规划教材《计算机算法设计与分析（第5版）》（ISBN 978-7-121-34439-8）配套的辅助教材，对《计算机算法设计与分析（第5版）》一书中的全部习题做了详尽的解答，旨在让使用该书的教师更容易教，学生更容易学。为了便于对照阅读，本书的章序与《计算机算法设计与分析（第5版）》一书的章序保持一致，且一一对应。

本书的内容是对《计算机算法设计与分析（第5版）》的较深入的扩展，许多教材中无法讲述的较深入的主题通过习题的形式展现出来。为了加强学生灵活运用算法设计策略解决实际问题的能力，本书将主教材中的许多习题改造成算法实现题，要求学生不仅设计出解决具体问题的算法，而且能上机实现。作者的教学实践反映出这类算法实现题的教学效果非常好。作者还结合国家精品课程建设，建立了“算法设计与分析”教学网站。国家精品资源共享课地址：http://www.icourses.cn/sCourse/course_2535.html。欢迎广大读者访问作者的教学网站并提出宝贵意见。

在本书编写过程中，福州大学“211工程”计算机与信息工程重点学科实验室为本书的写作提供了优良的设备与工作环境。电子工业出版社负责本书编辑出版工作的全体同仁为本书的出版付出了大量辛勤劳动，他们认真细致、一丝不苟的工作精神保证了本书的出版质量。在此，谨向每位曾经关心和支持本书编写工作的各方面人士表示衷心的谢意！

作　者

目　录

第 8 章　线性规划与网络流

算法分析题 8

8-1　线性规划可行区域无界的例子。

试给出一个线性规划的例子，使其可行区域是无界的，但其最优目标函数值却是有限的。

分析与解答：

$$\max z=-y$$

$$y-x\geq 1$$

$$x, y\geq 0$$

在$(x, y)=(0, 1)$处达到唯一的最大值−1。可行区域显然是无界的，如图 8-1 所示。

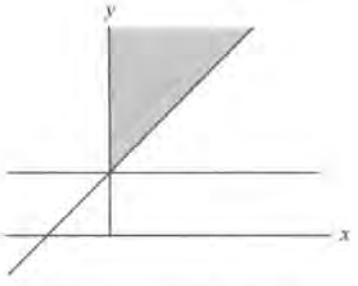

图 8-1　可行区域无界

8-2　单源最短路与线性规划。

试将单源最短路问题表示为一个线性规划问题。

分析与解答：

（1）单源单终点最短路问题

给定一个赋权有向图 $G=(V, E)$，其中每条边(i, j)的权是一个非负实数 w_{ij}。另外，给定 V 中的两个顶点 s 和 t。图 G 中以 s 为起点 t 为终点的有向路称为 s–t 有向路，它经过的所有边权之和称为该 s–t 有向路的长度。单源单终点最短路问题要求最短的 s–t 有向路。

设变量 x_{ij} 表示边(i, j)是否在 s–t 有向路上：当 $x_{ij}=1$ 时表示边(i, j)在 s–t 有向路上；当 $x_{ij}=0$ 时表示边(i, j)不在 s–t 有向路上。由此可将单源单终点最短路问题表示为如下线性规划问题：

$$\min \sum_{(i,j)\in E} w_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(i,j)\in E} x_{ji} = \begin{cases} 1 & i=s \\ -1 & i=t \\ 0 & i\neq s,t \end{cases}$$

$$x_{ij}\geq 0$$

从变量 x_{ij} 的定义可以看出，变量 x_{ij} 应为 0-1 变量，上述线性规划问题应表述为整数线性规划问题。但从约束条件可以看出，约数矩阵为全 1 模阵，因此 0-1 变量可以松弛为区间[0, 1]中的实数，用单纯形算法求解可得到 0-1 整数解。如果将变量 x_{ij} 松弛为所有非负实数，则约束条件还不足以保证所有非 0 变量所对应的边构成一条 s–t 有向路，可能含有向圈。由于每条边的权是一个非负实数，目标函数是非负的。任何正圈不可能使目标函数最小，因此找到的结构是一条 s–t 有向路，最多含有 0 圈。

最短路的最优性条件定理：对于赋权有向图 G 的每个顶点 $j\in V$，设 d_j 表示从源 s 到 j 的有向路的长度，则 d_j 是最短路的长度的充分必要条件是，对于任何$(i, j)\in E$ 有

$$d_j\leq d_i+w_{ij}$$

由最短路最优性条件定理，可将单源单终点最短路问题表述为如下的线性规划问题：

$$\min d_t$$

$$\text{s.t.} \quad \begin{cases} d_j \leqslant d_i + w_{ij} \\ d_s = 0 \end{cases}$$

（2）单源最短路问题

一般的单源最短路问题要求从源 s 到所有其他顶点的最短路。类似地，可将单源最短路问题表示为如下一个线性规划问题：

$$\min \sum_{(i,j)\in E} w_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(i,j)\in E} x_{ij} = \begin{cases} n-1 & i=s \\ -1 & i\neq s \end{cases}$$

$$x_{ij}\geqslant 0$$

由最短路最优性条件定理，也可将单源最短路问题表述为如下的线性规划问题：

$$\min \sum_{i\neq s} d_i$$

$$\text{s.t.} \quad \begin{cases} d_j \leqslant d_i + w_{ij} \\ d_s = 0 \end{cases}$$

8-3　网络最大流与线性规划。

试将网络最大流问题表示为一个线性规划问题。

分析与解答：对于给定的网络 $G=(V, E)$，源为 s，汇为 t。设边(i,j)的容量为 u_{ij}，边(i,j)的流量为 x_{ij}，可将网络最大流问题表示为如下线性规划问题：

$$\text{Max} f$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(i,j)\in E} x_{ji} = \begin{cases} f & i=s \\ -f & i=t \\ 0 & i\neq s,t \end{cases}$$

$$0\leqslant x_{ij}\leqslant u_{ij}$$

8-4　最小费用流与线性规划。

试将网络最小费用流问题表示为一个线性规划问题。

分析与解答：对于给定的网络 $G=(V, E)$，源为 s，汇为 t。设边(i,j)的容量为 u_{ij}，边(i,j)的流量为 x_{ij}，边(i, j)的单位流量费用为 c_{ij}，顶点 i 的流供需量为 d_i，可将网络最小费用流问题表示为如下线性规划问题：

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(i,j)\in E} x_{ji} = d_i$$

$$0\leqslant x_{ij}\leqslant u_{ij}$$

8-5　运输计划问题。

某集团公司拥有自己的产品运输网络。该公司现在生产 k 种产品，每种产品都需要从其生产地运输到销售地。假设第 i 种产品的产地为 s_i，销售地为 t_i，需要的运输量为 d_i。集团

公司需要规划其运输计划满足各种产品的运输需求。试建立该问题的线性规划模型。

分析与解答：设公司的产品运输网络为 $G=(V, E)$。边(u, v)的容量为 $c(u, v)$。产品 i 在边(u, v)上的运输量为 $f_i(u, v)$，则上述问题可以表示为如下线性规划问题：

$$\min 0$$

$$\text{s.t.}\begin{cases}\sum_{i=1}^{k} f_i(u,v) \leqslant c(u,v) & u,v \in V \\ f_i(u,v) = -f_i(v,u) & i=1,2,\cdots,k \\ \sum_{v \in V} f_i(u,v) = 0 & u \in V-\{s_i,t_i\} \\ \sum_{v \in V} f_i(s_i,v) = d_i & i=1,2,\cdots,k\end{cases}$$

8-6　单纯形算法。

试用单纯形算法解下面的线性规划问题：

$$\begin{aligned}&\min z = x_1 + x_2 + x_3 \\ \text{s.t.}\ &2x_1 + 7.5x_2 + 3x_2 \geqslant 10000 \\ &20x_1 + 5x_2 + 10x_3 \geqslant 30000 \\ &x_1,\ x_2,\ x_3 \geqslant 0\end{aligned}$$

分析与解答：用主教材中单纯形算法求解。输入格式为

```
-1 2 3
0 0 2
2 7.5 3 10000
20 5 10 30000
1 1 1
```

输出结果如下：

```
Minimize: 1.00 * x1+1.00 * x2+1.00 * x3
2.00 * x1+7.50 * x2+3.00 * x3>=10000.00
20.00 * x1+5.00 * x2+10.00 * x3 >=30000.00
```

初始单纯形表：

	0.00	−1.00	−1.00	−1.00	0.00	0.00
6	10000.00	2.00	7.50	3.00	−1.00	0.00
7	30000.00	20.00	5.00	10.00	0.00	−1.00
	0.00	22.00	12.50	13.00	−1.00	−1.00

单纯形算法求解：

	1500.00	0.05	−0.75	−0.50	0.00	−0.05
6	7000.00	−0.10	7.00	2.00	−1.00	0.10
1	1500.00	0.05	0.25	0.50	0.00	−0.05
	−33000.00	−1.10	7.00	2.00	−1.00	0.10

	2250.00	0.04	0.11	−0.29	−0.11	−0.04
2	1000.00	−0.01	0.14	0.29	−0.14	0.01
1	1250.00	0.05	−0.04	0.43	0.04	−0.05
	−40000.00	−1.00	−1.00	0.00	0.00	0.00

变量：1，2，3；松弛变量：4，5；人工变量：6，7。

最优值：2250.00。

最优解：x1=1250.00，x2=1000.00，x3=0.00。

8-7　边连通度问题。

无向图 $G=(V, E)$的边连通度为 k 是指最少需要移去 G 的 k 条边才能使 G 成为不连通图。例如，树的边连通度为 1；循环链的边连通度为 2。试用网络最大流算法求给定图 G 的边连通度。

分析与解答：可以通过解$|V|$个最大流问题求给定图 G 的边连通度，具体做法如下。

设(u, v)是 G 的一条边。将图 G 看作等价的有向图，即将图 G 的每条边看作 2 条有向边。设每条边的流容量为 1，并设源 s 为 u，汇 t 为 v。该网络的最大流量记为 $f(u, v)$，则由最大流和最小割定理可知，穿过任何割的边数等于割的容量，从而等于该网络的最大流量 $f(u, v)$。由此可见，图 G 的边连通度为 $\min\limits_{v\in V-\{u\}}\{f(u,v)\}$。

8-8　有向无环网络的最大流。

试证明有向无环网络的最大流问题等价于标准网络最大流问题。

分析与解答：有向无环网络的最大流问题是标准网络最大流问题的特殊情形，因此只要证明标准网络最大流问题可以变换为有向无环网络的最大流问题。给定一个标准网络 $G=(V, E)$，设 $n=|V|$，$m=|E|$，下面构造一个与之相应的有 $2n+2$ 个顶点和 $m+3n$ 条边的有向无环网络。

设标准网络 G 的顶点编号为 $0, 1, \cdots, n-1$。每个顶点 i 增加一个新顶点 $n+i$ 构成一个如下二分图：

① 原网络中的边(i, j)对应新网络中的边$(i, n+j)$，容量不变（$0\leqslant i, j<n$）。

② 设 X 是原网络中各边容量之和，新网络中增加边$(i, n+i)$（$0\leqslant i<n$），容量为 X。

③ 新增源 s 和汇 t。增加边(s, i)和$(n+i, t)$（$0\leqslant i<n$）。

新网络显然是一个有向无环网络。

进一步可以证明，对于原网络中每个大小为 c 的 $s-t$ 割都对应新网络中的一个大小为 $c+X$ 的 $s-t$ 割；新网络中每个大小为 $c+X$ 的最小 $s-t$ 割都对应原网络中的一个大小为 $c+X$ 的 $s-t$ 割。因此，新网络的最小割对应于原网络的最小割，从而新网络的最大流也对应原网络的最大流。

8-9　无向网络的最大流。

试将无向网络最大流问题变换为标准网络最大流问题。

分析与解答：无向网络的最大流问题的定义为：给定一个赋权无向图 G，每条边的边权解释为该边上流的容量。一条边上的流可以看作对流的定向，但必须满足容量约束和顶点的流量平衡条件。无向网络的最大流问题要求从 s 到 t 的最大流。

对于给定的无向网络，构造相应的有向网络如下：将原网络的每条边都扩展为 2 条有向边，这 2 条有向边的容量均为原无向边的边权。原无向网络的流显然也是新网络中的流。反

之，若新网络中边(u, v)上的流为f，边(v, u)上的流为g，则当$f \geq g$时，对应于原网络边流量为$f-g$的(u, v)定向，否则对应于原网络边流量为$g-f$的(v, u)定向。由此可见，新网络的最大流与原网络最大流对应。

8-10　最大流更新算法。

设$G=(V, E)$是源为s，汇为t，且容量均为整数的一个流网络。已知f是G的一个最大流。

① 假设一条边$(u, v) \in E$的容量增1，试设计在$O(|V|+|E|)$时间内更新最大流f的算法。

② 假设一条边$(u, v) \in E$的容量减1，试设计在$O(|V|+|E|)$时间内更新最大流f的算法。

分析与解答：

（1）将边$(u, v) \in E$的容量增1得到新的残留网络。在新的残留网络网络中找一条增广路并增流。更新最大流的算法显然只需$O(|V|+|E|)$时间。如果边$(u, v) \in E$的容量增加k，则在最坏情况下更新最大流的算法需要$O(k(|V|+|E|))$时间。

（2）将边$(u, v) \in E$的容量减1，如果边(u, v)的流量违反容量约束，则将flow(u, v)减1。此时顶点u处的存流为1，而顶点v处的存流为-1。用增广路算法找顶点u到顶点v的增广路，消去顶点u和顶点v处的存流。如果找不到顶点u到顶点v的增广路，则最大流必须减1。此时仍可用增广路算法找顶点u到顶点s的路和顶点t到顶点v的路，并沿这两条路减流得到新的最大流。更新最大流的算法显然只需$O(|V|+|E|)$时间。如果边$(u, v) \in E$的容量减少k，则在最坏情况下更新最大流的算法需要$O(k(|V|+|E|))$时间。

8-11　混合图欧拉回路问题。

试设计一个找混合图（既有无向边也有有向边的图）的欧拉回路的有效算法。

分析与解答：问题的关键是给每条无向边定向，使定向后的有向图有欧拉回路。为此构造网络G如下。

设原图有n个顶点和m条边。将原图中每个顶点u拆为两个顶点u和u_0，并增加边(u, u_0)，其容量为∞。原图中每条无向边(u, v)对应两条容量为1的有向边(u_0, v)和(v_0, u)。原图中每条有向边(u, v)对应一条容量为1的有向边(u_0, v)。网络中若存在一个可行流，使得每条新增加边(u, u_0)上的流量均为顶点u在原图中度数的一半，则原图有欧拉回路。为此目的，新增源s和汇t。从源s到每个顶点u_0有一条有向边(s, u_0)，其容量为deg$(u)/2$。从每个顶点u到汇t有一条有向边(u, t)，其容量为deg$(u)/2$。求所建立的网络中流量为m的可行流。

8-12　单源最短路与最小费用流。

试将单源最短路问题表示为一个最小费用流问题。

分析与解答：对于给定的源顶点为source的有向图G的单源最短路问题，构造相应网络如下。网络在原有向图G的基础上，给每条有向边的容量为∞，费用为原来的边权。新增源s和汇t。从s到源顶点source有一条有向边(s, source)，其容量为$n-1$，费用为0。从source外的其他$n-1$个顶点到汇t有一条有向边，其容量为1，费用为0。

求上述网络的最小费用可行流。得到的流支撑树就对应于原图G的最短路支撑树。

8-13　中国邮路问题。

试用最小费用流算法解中国邮路问题。

分析与解答：对于给定的有向图G的中国邮路问题，构造相应网络如下。网络在原有向图G的基础上，给每条有向边的容量为∞，费用为原来的边权。新增每条边的容量下界约束1，即要求每条边上的流量至少为1。求上述网络的最小费用循环流。根据流分解定理，

得到的循环流可以分解为若干个圈，由此容易构造出原图 G 的中国邮路。

算法实现题 8

8-1　飞行员配对方案问题。

问题描述：第二次世界大战时期，英国皇家空军从沦陷国征募了大量外籍飞行员。由皇家空军派出的每架飞机都需要配备在航行技能和语言上能互相配合的 2 名飞行员，其中一名是英国飞行员，另一名是外籍飞行员。在众多的飞行员中，每名外籍飞行员都可以与其他若干名英国飞行员很好地配合。如何选择配对飞行的飞行员才能使一次派出最多的飞机。

算法设计：对于给定的外籍飞行员与英国飞行员的配合情况，找出一个最佳飞行员配对方案，使皇家空军一次能派出最多的飞机。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有两个正整数 m 和 n。n 是皇家空军的飞行员总数（n<100）；m 是外籍飞行员数。外籍飞行员编号为 1～m；英国飞行员编号为 m+1～n。接下来每行有两个正整数 i 和 j，表示外籍飞行员 i 可以和英国飞行员 j 配合。文件最后以两个−1 结束。

结果输出：将最佳飞行员配对方案输出到文件 output.txt。第 1 行是最佳飞行员配对方案一次能派出的最多的飞机数 M。接下来的 M 行是最佳飞行员配对方案。每行有两个正整数 i 和 j，表示在最佳飞行员配对方案中，飞行员 i 和飞行员 j 配对。

如果所求的最佳飞行员配对方案不存在，则输出“No Solution!”。

输入文件示例	输出文件示例
input.txt	output.txt
5 10	4
1 7	1 7
1 8	2 9
2 6	3 8
2 9	5 10
2 10	
3 7	
3 8	
4 7	
4 8	
5 10	
−1 −1	

分析与解答：本题的数学模型是二分图的最大基数匹配问题，可将其变换为网络最大流问题。

如果图 G 是一个二分图，则可将图 G 的顶点集合分成两部分 X 和 Y，使得 G 中任一边所关联的两个顶点中，有一个顶点属于 X，而另一个顶点属于 Y。本题中外籍飞行员与英国飞行员的配合情况图就是一个二分图。外籍飞行员所对应的顶点为 X，英国飞行员所对应的顶点为 Y。

二分图的最大匹配问题就是在已知图 G 是一个二分图的前提下，求 G 的最大匹配。

给定一个二分图 G 和将图 G 的顶点集合 V 分成互不相交的两部分的顶点子集 X 和 Y，

构造与之相应的网络 F 如下。

（1）增加一个源 s 和一个汇 t；

（2）从 s 向 X 的每一个顶点都增加一条边；从 Y 的每一个顶点都向 t 增加一条边；

（3）原图 G 中的每一条边都改为相应的由 X 指向 Y 的有向边；

（4）置所有边的容量为 1。

求网络 F 的最大流。在从 X 指向 Y 的边集中，流量为 1 的边对应于二分图中的匹配边。最大流值对应于二分图 G 的最大匹配边数。

主教材中的类 BMATCHING 算法可用于解二分图的最大匹配问题。

实现算法的主函数如下。

```
int main() {
   int  s, t, N, n1;
   char  file[11] = "bm.txt";
   fin >> n1 >> N;
   fout << n1 << " "<< N << endl;
   fin >> s >> t;
   while(s >= 0) {
      fout << s-1 << " " << t-1 << endl;
      fin >> s >> t;
   }
   fout << -1 << -1 << endl;
   GRAPH<EDGE>G(N, 1);
   G.readbm(file);
   BMATCHING<GRAPH<EDGE>, EDGE>(G, n1);
   return 0;
}
```

8-2　太空飞行计划问题。

问题描述：W 教授正在为国家航天中心计划一系列的太空飞行。每次太空飞行可进行一系列商业性实验从而获取利润。现已确定了一个可供选择的实验集合 $E=\{E_1, E_2, \cdots, E_m\}$ 和进行这些实验需要使用的全部仪器的集合 $I=\{I_1, I_2, \cdots, I_n\}$。实验 E_j 需要用到的仪器是 I 的子集 $R_j \subseteq I$。配置仪器 I_k 的费用为 c_k 美元。实验 E_j 的赞助商已同意为该实验结果支付 p_j 美元。W 教授的任务是找出一个有效算法，确定在一次太空飞行中要进行哪些实验并因此而配置哪些仪器，才能使太空飞行的净收益最大。这里的净收益是指进行实验所获得的全部收入与配置仪器的全部费用的差额。

算法设计：对于给定的实验和仪器配置情况，找出净收益最大的实验计划。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有两个正整数 m 和 n。m 是实验数，n 是仪器数。接下来的 m 行，每行是一个实验的有关数据。第一个数是赞助商同意支付该实验的费用，然后是该实验需要用到的若干仪器的编号。最后一行的 n 个数是配置每个仪器的费用。

结果输出：将最佳实验方案输出到文件 output.txt。第 1 行是实验编号，第 2 行是仪器编号，最后一行是净收益。

输入文件示例	输出文件示例
input.txt	output.txt

2 3　　1 2
10 1 2　　1 2 3
25 2 3　　17
5 6 7

分析与解答：建立网络最大流模型解此问题。

（1）构造网络 G

网络 G 有 $m+n+2$ 个顶点。全部仪器 $I_1, I_2, \cdots, I_n$ 和可供选择的实验 $E_1, E_2, \cdots, E_m$ 分别对应 $m+n$ 个顶点。新增源 s 和汇 t，从源 s 到每个顶点 I_i 有一条有向边(s, I_i)，其容量为 c_i。从每个顶点 E_j 到汇 t 有一条有向边(E_j, t)，其容量为 p_j。若 $I_k \in R_j$，则从顶点 I_k 到顶点 E_j 有一条有向边(I_k, E_j)，其容量为∞。

（2）求网络 G 的最大流

设最大流量为 f，则所求的最大净收益为 $\sum_{j=1}^{m} p_j - f$。

事实上，网络 G 的任何一个割 cut(S, T)对应一个实验方案。如果顶点 $E_j \in T$，则任何 $I_k \in R_j$ 有 $I_k \in T$，否则割的容量将为∞。因此，实验 E_j 和 E_j 所需的仪器均在 T 中，T 中的所有实验构成一个实验方案。该实验方案的净收益为

$$\sum_{E_j \in T} p_j - \sum_{I_k \in T} c_k = \sum_{j=1}^{m} p_j - \sum_{E_j \in S} p_j - \sum_{I_k \in T} c_k$$

$$= \sum_{j=1}^{m} p_j - \text{cap}(S,T)$$

由此可见，要使净收益最大，必须使 cap(S, T)最小，即最优实验方案构成网络 G 的一个最小割。由最大流最小割定理即知，若求得网络 G 的最大流量 f，则所求的最大净收益为

$$\sum_{j=1}^{m} p_j - f$$

8-3　最小路径覆盖问题。

问题描述：给定有向图 $G=(V, E)$。设 P 是 G 的一个简单路（顶点不相交）的集合。如果 V 中每个顶点恰好在 P 的一条路上，则称 P 是 G 的一个路径覆盖。P 中路径可以从 V 的任何一个顶点开始，长度也是任意的，特别地，可以为 0。G 的最小路径覆盖是 G 的所含路径条数最少的路径覆盖。

设计一个有效算法求一个有向无环图 G 的最小路径覆盖。

〖提示：设 $V=\{1, 2, \cdots, n\}$，如下构造网络 $G_1=(V_1, E_1)$：

$$V_1 = \{x_0, x_1, \cdots, x_n\} \cup \{y_0, y_1, \cdots, y_n\}$$

$$E_1 = \{(x_0, x_i) \mid i \in V\} \cup \{(y_i, y_0) \mid i \in V\} \cup \{(x_i, x_j) \mid (i,j) \in E\}$$

每条边的容量均为 1。求网络 G_1 的(x_0, y_0)最大流。〗

算法设计：对于给定的有向无环图 G，找出 G 的一个最小路径覆盖。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有 2 个正整数 n 和 m。n 是给定有向无环图 G 的顶点数，m 是 G 的边数。接下来的 m 行，每行有 2 个正整数 i 和 j，表示一条有向边(i, j)。

结果输出：将最小路径覆盖输出到文件 output.txt。从第 1 行开始，每行输出一条路径。文件的最后一行是最少路径数。

输入文件示例	输出文件示例
input.txt	output.txt
11 12	1 4 7 10 11
1 2	2 5 8
1 3	3 6 9
1 4	3
2 5	
3 6	
4 7	
5 8	
6 9	
7 10	
8 11	
9 11	
10 11	

分析与解答：按照提示，建立网络最大流模型解此问题。

（1）将有向无环图 G 的每个顶点 i，拆成两个顶点 x_i 和 y_i。新增源 x_0 和汇 y_0。构造网络 $G_1=(V_1, E_1)$如下：

$$V_1=\{x_0, x_1, \cdots, x_n\} \cup \{y_0, y_1, \cdots, y_n\}$$

$$E_1=\{(x_0, x_i) \mid i \in V\} \cup \{(y_i, y_0) \mid i \in V\} \cup \{(x_i, y_j) \mid (i, j) \in E\}$$

每条边的容量均为 1。

（2）求网络 G_1 的(x_0, y_0)最大流。

（3）从 G_1 的(x_0, y_0)最大流构造出 G 的一个最小路径覆盖如下。

当 $\text{flow}(x_0, x_i)=\text{flow}(x_i, y_i)=\text{flow}(y_j, y_0)=1$ 时，边(i, j)属于最小路径覆盖。

当 $\text{flow}(x_0, x_i)=\text{flow}(y_i, y_0)=0$ 时，顶点 i 是最小路径覆盖中的一条长度为 0 的路径。网络 G_1 的(x_0, y_0)最大流量为 f，则最少路径数为 $n-f$。

8-4 魔术球问题。

问题描述：假设有 n 根柱子，现要按下述规则在这 n 根柱子中依次放入编号为 1, 2, 3, …的球。

① 每次只能在某根柱子的最上面放球。

② 在同一根柱子中，任何两个相邻球的编号之和为完全平方数。

试设计一个算法，计算出在 n 根柱子上最多能放多少个球。例如，在 4 根柱子上最多可放 11 个球。

算法设计：对于给定的 n，计算在 n 根柱子上最多能放多少个球。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有 1 个正整数 n，表示柱子数。

结果输出：将 n 根柱子上最多能放的球数及相应的放置方案输出到文件 output.txt。文件的第 1 行是球数。接下来的 n 行，每行是一根柱子上的球的编号。

输入文件示例	输出文件示例
input.txt	output.txt
4	11

1 8
2 7 9
3 6 10
4 5 11

分析与解答：借助有向无环图的最小路径覆盖问题，并建立网络最大流模型解此问题。

设给定编号为 1, 2, 3, …, n 的 n 个球，建立一个有 n 个顶点的有向无环图 G 如下。每个球对应图 G 中一个顶点。当两个不同的球 i 和 j 满足 $i<j$ 且 $i+j$ 是一个平方数时，G 中有一条边(i, j)。图 G 的最小路径覆盖确定了本题中的最少柱子数。进一步分析可以证明，n 个柱子能放入的最多球数为$\lfloor (n^2+2n-1)/2 \rfloor$。

由此可知，魔术球问题等价于有向无环图的最小路径覆盖问题。

8-5　圆桌问题。

问题描述：假设有来自 n 个不同单位的代表参加一次国际会议。每个单位的代表数分别为 r_i（i=1, 2, …, n）。会议餐厅共有 m 张餐桌，每张餐桌可容纳 c_i（i=1, 2, …, m）个代表就餐。为了使代表们充分交流，希望从同一个单位来的代表不在同一个餐桌就餐。试设计一个算法，给出满足要求的代表就餐方案。

算法设计：对于给定的代表数和餐桌数以及餐桌容量，计算满足要求的代表就餐方案。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有 2 个正整数 m 和 n，m 表示餐桌数，n 表示单位数（$1 \leqslant m \leqslant 150$，$1 \leqslant n \leqslant 270$）。文件第 2 行有 m 个正整数，分别表示每个单位的代表数。文件第 3 行有 n 个正整数，分别表示每个餐桌的容量。

结果输出：将代表就餐方案输出到文件 output.txt。如果问题有解，在文件第 1 行输出 1，否则输出 0。接下来的 m 行给出每个单位代表的就餐桌号。如果有多个满足要求的方案，只要输出一个方案。

输入文件示例	输出文件示例
input.txt	output.txt
4 5	1
4 5 3 5	1 2 4 5
3 5 2 6 4	1 2 3 4 5
	2 4 5
	1 2 3 4 5

分析与解答：建立网络最大流模型解此问题。

（1）构造网络 G

网络 G 有 $m+n+2$ 个顶点。n 个不同的单位和 m 张餐桌分别对应 $m+n$ 个顶点。新增源 s 和汇 t。从源 s 到每个单位顶点 I_i 有一条有向边(s, I_i)，其容量为 r_i。从每个餐桌顶点 E_j 到汇 t 有一条有向边(E_j, t)，其容量为 c_j。从每个单位顶点 I_i 到每个餐桌顶点 E_j 有一条有向边(I_i, E_j)，其容量为 1。

（2）求网络 G 的最大流

设最大流量为 f，当 $\sum_{j=1}^{n} r_j = f$ 时，有满足要求的代表就餐方案。

8-6　最长递增子序列问题。

问题描述：给定正整数序列 $x_1, x_2, \cdots, x_n$。要求：

① 计算其最长递增子序列的长度 s。

② 计算从给定的序列中最多可取出多少个长度为 s 的递增子序列。

③ 如果允许在取出的序列中多次使用 x_1 和 x_n，则从给定序列中最多可取出多少个长度为 s 的递增子序列。

算法设计：设计有效算法完成①、②、③提出的计算任务。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有 1 个正整数 n，表示给定序列的长度。接下来的 1 行有 n 个正整数 $x_1, x_2, \cdots, x_n$。

结果输出：将任务①、②、③的解答输出到文件 output.txt。第 1 行是最长递增子序列的长度 s。第 2 行是可取出的长度为 s 的递增子序列个数。第 3 行是允许在取出的序列中多次使用 x_1 和 x_n 时可取出的长度为 s 的递增子序列个数。

输入文件示例	输出文件示例
input.txt	output.txt
4	2
3 6 2 5	2
	3

分析与解答：先用动态规划算法计算出以 x_i 开头的最长递增子序列的长度 b[i]。然后构造网络 G 如下。

按照 b[i]的大小将 x_i 分层，b[i]=k 的数对应第 k 层顶点。当 x_i 是第 k 层顶点，x_j 是第 k−1 层顶点且 $x_i<x_j$ 时，x_i 和 x_j 相应的顶点之间增加一条有向边。由于每个数只能取 1 次，因此顶点的容量约束为 1。如果允许在取出的序列中多次使用 x_1 和 x_n，则 x_1 和 x_n 相应的顶点没有顶点的容量约束。求相应网络的最大流可得最多的最长递增子序列。

具体算法用类 LIS 实现如下。

```
class LIS {
public:
   LIS() { };
   ~LIS() { };
   void input(char *filename);
   void compute(int tt);
private:
   ofstream outFile;
   int  n, *a, *b, **c;
   int maxL(int n);
   int dynb();
};
```

其中，input()函数用于读入初始数据。

```
void LIS::input(char *filename) {
   ifstream inFile;
   inFile.open(filename);
   inFile >> n;
   a = new int[n];
   b = new int[n+1];
   Make2DArray(c, n+1, n+1);
   for(int i=0; i<n;i++)
      inFile >> a[i];
   inFile.close();
```

```
}
```

dynb()函数用动态规划算法计算出以 x_i 开头的最长递增子序列的长度 b[i]。

```
int LIS::dynb() {
   int  i, j, k;
   for(i=n-2, b[n-1]=1; i >= 0; i--) {
      for(j=n-1, k=0; j > i; j--)
         if(a[j] >= a[i] && k < b[j])
            k = b[j];
      b[i] = k+1;
   }
   for(i=0; i <= n; i++)
      for(j=0; j <= n; j++)
         c[i][j] = 0;
   for(i=0; i < n; i++) {
      c[b[i]][0]++;
      c[b[i]][c[b[i]][0]]=i;
   }
   return maxL(n);
}
```

maxL()函数计算最长递增子序列的长度 s。

```
int LIS::maxL(int n) {
   for(int i=0, temp=0; i < n; i++)
      if(b[i] > temp)
         temp = b[i];
   return temp;
}
```

compute()函数构造相应的网络 G，并计算最大流。

```
void LIS::compute(int tt) {
   GRAPH<EDGE>G(2*n+2, 1);
   int  s=0, t=2*n+1, maxflow=0;
   int len = dynb();
   if(tt == 0)
      cout<<len<<endl;
   for(int i=len; i>  1; i--)
      for(int j=1, kj=c[i][0];j  <= kj; j++)
         for(int k=1, kk=c[i-1][0]; k <= kk; k++)
            if(c[i][j] < c[i-1][k] && a[c[i][j]] <= a[c[i-1][k]])
               G.insert(new EDGE(2*c[i][j]+2, 2*c[i-1][k]+1, 1));
   int  sf = 1, tf = 1;
   if(tt > 0) {
      sf=n;
      tf=n;
   }
   for(i=1; i <= c[len][0]; i++)
      G.insert(new EDGE(s, 2*c[len][i]+1, sf));
   for(i=1; i <= c[1][0]; i++)
```

```
    G.insert(new EDGE(2*c[1][i]+2, t, 1));
  for(i=2; i < n; i++)
    G.insert(new EDGE(2*i-1, 2*i, 1));
  G.insert(new EDGE(1, 2, sf));
  G.insert(new EDGE(2*n-1, 2*n, tf));
  G.insert(new EDGE(2*n, t, tf));
  MAXFLOW<GRAPH<EDGE>, EDGE>(G, s, t, maxflow);
  cout << maxflow << endl;
}
```

实现算法的主函数如下。

```
int main(){
  char  *filein;
  filein = "alis.in";
  LIS X;
  X.input(filein);
 X.compute(0);
  X.compute(1);
  return 0;
}
```

8-7　试题库问题。

问题描述：假设一个试题库中有 n 道试题。每道试题都标明了所属类别。同一道题可能有多个类别属性。现要从题库中抽取 m 道题组成试卷。并要求试卷包含指定类型的试题。试设计一个满足要求的组卷算法。

算法设计：对于给定的组卷要求，计算满足要求的组卷方案。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有 2 个正整数 k 和 n（$2 \leqslant k \leqslant 20$，$k \leqslant n \leqslant 1000$），$k$ 表示题库中试题类型总数，n 表示题库中试题总数。第 2 行有 k 个正整数，第 i 个正整数表示要选出的类型 i 的题数。这 k 个数相加就是要选出的总题数 m。接下来的 n 行给出了题库中每个试题的类型信息。每行的第 1 个正整数 p 表明该题可以属于 p 类，接着的 p 个数是该题所属的类型号。

结果输出：将组卷方案输出到文件 output.txt。文件第 i 行输出“i:”后接类型 i 的题号。如果有多个满足要求的方案，只要输出 1 个方案。如果问题无解，则输出“No Solution!”。

输入文件示例	输出文件示例
input.txt	output.txt
3 15	1: 1 6 8
3 3 4	2: 7 9 10
2 1 2	3: 2 3 4 5
1 3	
1 3	
1 3	
1 3	
3 1 2 3	
2 2 3	
2 1 3	
1 2	

1 2
2 1 2
2 1 3
2 1 2
1 1
3 1 2 3

分析与解答：建立网络最大流模型解此问题。

（1）构造网络 G

网络 G 有 $k+n+2$ 个顶点。每道试题对应一个顶点 u_i；每个试题类型也对应一个顶点 v_i。新增源 s 和汇 t。设要选出类型 i 的题数为 k_i。从源 s 到每个试题类型顶点 v_i 各有一条有向边 (s, v_i)，其容量为 k_i。从每个试题顶点 u_i 到汇 t 有一条有向边(u_i, t)，其容量为 1。从每个试题类型顶点 v_i 到每个试题顶点 u_j 各有一条容量为 1 的有向边(v_i, u_j)的前提条件是试题 j 属于试题类型 i。

（2）求网络 G 的最大流

设最大流量为 f，当 $\sum_{j=1}^{k} k_j = f$ 时，有满足要求的组卷方案。

8-8　机器人路径规划问题。

问题描述：机器人 Rob 可在一个树状路径上自由移动。给定树状路径 T 上的起点 s 和终点 t，机器人要从 s 运动到 t。树状路径 T 上有若干可移动的障碍物。由于路径狭窄，任何时刻在路径的任何位置不能同时容纳 2 个物体。每步可以将障碍物或机器人移到相邻的空顶点上。设计一个有效算法用最少移动次数使机器人从 s 运动到 t。

算法设计：对于给定的树 T，以及障碍物在树 T 中的分布情况，计算机器人从起点 s 到终点 t 的最少移动次数。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行有 3 个正整数 n，s 和 t，分别表示树 T 的顶点数，起点 s 的编号和终点 t 的编号。

接下来的 n 行分别对应于树 T 中编号为 0, 1, …, $n-1$ 的顶点。每行的第 1 个整数 h 表示顶点的初始状态，当 h=1 时表示该顶点为空顶点，当 h=0 时表示该顶点为满顶点，其中已有一个障碍物。第 2 个数 k 表示有 k 个顶点与该顶点相连。接下来的 k 个数是与该顶点相连的顶点编号。

结果输出：将计算出的机器人最少移动次数输出到文件 output.txt。如果无法将机器人从起点 s 移动到终点 t，则输出“No Solution!”。

输入文件示例	输出文件示例
input.txt	output.txt
5 0 3	3
1 1 2	
1 1 2	
1 3 0 1 3	
0 2 2 4	
1 1 3	

分析与解答：用最小费用流算法求解。

根据给定条件构造网络 G 之前，需要对问题进行较深入的分析。

对于给定的树 T，设初始时已有障碍物的顶点集合为 O。机器人移动的起点为 s，终点为 t。二元组 conf=(O, s)构成机器人路径规划问题在给定树 T 上的一个布局；三元组 $S=(O, s, t)$构成机器人路径规划问题对于给定树 T 的一个实例。

设给定的树 T 中有 n 个顶点，从 s 到 t 的路径为 $P=(v_0, v_1, \cdots, v_d)$，起点为 $s=v_0$，终点为 $t=v_d$，路长为 d。路径 P 称为机器人路径规划问题中的关键路径。除了起点 s，路径 P 上有分叉的顶点称为路径 P 上的分叉顶点。对于路径 P 上每个顶点 v_j，将 v_j 的所有分支中的顶点记为 B_j。如果 B_j 非空，称 B_j 中与顶点 v_j 相邻的顶点为顶点 v_j 的旁路顶点。

容易看出，$P, B_0, B_1, \cdots, B_d$ 构成树 T 中顶点的一个划分。

另外，对于路径 P 上每个顶点 v_j 和每个分叉顶点 v_r，$0<r<j$，定义顶点集合 Ch(r, j)如下：

$$\mathrm{Ch}(r,j)=\{v_h \mid r\leqslant h<j\};\ \mathrm{Ch}(0,j)=B_0\cup\{v_h \mid r\leqslant h<j\}$$

对于树 T 中的顶点对(u, w)，定义偏序关系“$<$”如下：

$$u<w\Leftrightarrow\begin{cases}u=v_i, w=v_j & i<j\\ u=v_i, w\in B_j & i<j\\ u\in B_i, w=v_j & i\leqslant j\\ u\in B_i, w\in B_j & i<j\\ u, w\in B_0, w\in p(u,s)\end{cases}$$

设 A 是机器人从起点 s 到终点 t 的一个移动方案，依据关键路径上的每个顶点 $v_j(0\leqslant j\leqslant d)$，将 A 中的移动划分为左半部移动 $L(A, j)$和右半部移动 $R(A, j)$如下。

（1）机器人移动 $u\rightarrow w$

当 $w\leqslant v_j$ 时，$u\rightarrow w\in L(A, j)$，否则 $u\rightarrow w\in R(A, j)$。

（2）障碍物外移 $u\rightarrow w$，$w\in B_i$

当 $v_i\leqslant v_j$ 时，$u\rightarrow w\in L(A, j)$；

当 $v_j<v_i$ 且 $v_j<u$ 时，$u\rightarrow w\in R(A, j)$；

当 $u<v_j<v_i$ 时，$u\rightarrow v_j\in L(A, j)$，$v_j\rightarrow w\in R(A, j)$。

（3）障碍物后移 $u\rightarrow w$，且机器人位于顶点 v_l 的旁路顶点中

设 v_r 是机器人到达顶点 v_j 之前进入其旁路顶点的分叉顶点。如果机器人到达顶点 v_j 之前未进入任何旁路顶点，则 $v_r=s$。

当 $w<v_r$ 时，$u\rightarrow w\in L(A, j)$，否则 $u\rightarrow w\in R(A, j)$。

（4）障碍物前移 $u\rightarrow w$

设 v_r 是机器人到达顶点 v_j 之前进入其旁路顶点的分叉顶点。如果机器人到达顶点 v_j 之前未进入任何旁路顶点，则 $v_r=s$。

设 l 是满足下面条件的最大整数：$j\leqslant l$，$v_l\rightarrow z$ 是一个以 v_l 为初始起点的移动，且 $z<v_r$，或 $z\in B_h$，$h\leqslant j$。

当 $w\leqslant v_l$（从而 $u\leqslant v_l$）时，$u\rightarrow w\in L(A, j)$；

当 $v_l<u$（从而 $v_l<w$）时，$u\rightarrow w\in R(A, j)$；

当 $u<v_j$ 且 $v_l<w$ 时，$u\rightarrow v_j\in L(A, j)$，$v_j\rightarrow w\in R(A, j)$。

对于机器人从起点 s 到终点 t 的所有移动方案 A，将左半部移动 $L(A, j)$依 5 个参数分类为 $C(j, k, l, f, r)$如下。

① 参数 j。$L(A, j)$是关于顶点 v_j 的左半部移动。

② 参数 k。有 k 个障碍物移动到顶点 v_j。

③ 参数 l。l 是满足下面条件的最大整数：$j \leqslant l$，$v_l \rightarrow z$ 是一个以 v_l 为初始起点的移动，且 $z<v_r$，或 $z \in B_h$，$h \leqslant j$。如果没有这样的移动，则 $l=j$。

④ 参数 f。f 是满足下面条件的最大整数：$j<f \leqslant l$，$u \rightarrow v_f$ 是 $L(A, j)$的一个障碍物前移。如果没有这样的障碍物前移，则 $f=j$。

⑤ 参数 r。r 是满足下面条件的最大整数：$0<r<j$，v_r 是机器人到达顶点 v_j 之前进入其旁路顶点的分叉顶点。如果机器人到达顶点 v_j 之前未进入任何旁路顶点，则 $r=0$。

$C(j, k, l, f, r)$可以看作将机器人从 s 移动到 v_j，但在顶点 v_j 处还有 k 个障碍物待确定终点。这 k 个障碍物称为悬挂障碍物，相应的移动称为悬挂移动。若一个悬挂障碍物的起点为 u，则接着的移动只有两种方式：

① 障碍物外移 $u \rightarrow w$，$w \in B_i$，$i>j$。

② 先障碍物前移 $u \rightarrow v_z$，$z>l$，然后障碍物后移 $v_z \rightarrow w$，$v_r \leqslant w$。

设 $L_1 \in C(j, k_1, l_1, f_1, r_1)$，$L_2 \in C(j+1, k_2, l_2, f_2, r_2)$，定义它们的差 $M=\Delta(L_1, L_2)$如下：

对于每个移动 $u \rightarrow v \in L_2$，且移动 $u \rightarrow v$ 分解为 $u \rightarrow v'$和 $v' \rightarrow v$，$u \rightarrow v' \in L_1$，则 $v' \rightarrow v \in M$，否则 $u \rightarrow v \in M$。M 也称为 L_1 到 L_2 的一个扩展。

M 中的移动只有下面 4 种类型：

（1）机器人移动 $v_j \rightarrow v_j+1$；

（2）障碍物前移 $v_j \rightarrow v_j+1$；

（3）障碍物外移 $u \rightarrow w$，$w \in B_j+1$；

（4）如果 v_j 是一个分叉顶点，则 M 可能包含机器人从 v_j 到其旁路顶点，然后从旁路顶点返回的移动。

① 如果发生上述移动，则 $r_2=j$，M 包含所有机器人在 v_j 的旁路顶点中时的障碍物后移 $u \rightarrow w$，$v_{l_1+1} \leqslant u \leqslant v_{l_2}$，$v_r \leqslant w<v_j$；

② 如果顶点序列 $v_{f_1+1}, \cdots, v_{l_1}$ 中没有空顶点，则对 v_{l_1} 和 v_{f_2} 中的每个空顶点 w，M 中包含移动 $v_j \rightarrow w$。

如果 M 中含有 h_1 个障碍物外移，h_2 个障碍物前移和 h_3 个障碍物后移，则称 M 是 L_1 到 L_2 的一个(h_1, h_2, h_3)扩展。

如果 M 是 L_1 到 L_2 的一个(h_1, h_2, h_3)扩展，则：

① 如果存在 $f_1<m \leqslant l_1$，使 v_m 是一个空顶点，则 $h_2=0$；

② 设 v_m 是最靠近 v_{f_1} 的空顶点，如果 v_{j+1} 是 v_j 和 v_m 间唯一的分叉顶点，则 $h_1=k$；

③ 如果 $j+1 \leqslant f_1$，则所有障碍物外移均为悬挂移动。

④ 如果 $h_2>0$ 且 v_f 是 $v_{l_1+1}, \cdots, v_d$ 中的第 h_2 个空顶点，则在关键路径 P 上顶点 v_{l_1} 和 v_f 之间最多有 h_3-h_2 个 O 中的顶点。

对于机器人路径规划问题的一个实例 S，构造网络 $G(S)$如下。

① 每个非空移动类 $C(j, k, l, f, r)$对应网络 $G(S)$中一个顶点 $w(j, k, l, f, r)$。顶点 $w(j, k, l, f, r)$到顶点 $w(j', k', l', f', r')$之间有一条有向边，当且仅当 $C(j', k', l', f', r')$是 $C(j, k, l, f, r)$的一个(h_1, h_2, h_3)扩展，该有向边的容量为 1，费用为最小(h_1, h_2, h_3)扩展费用。

② 新增源顶点 w_s 和汇顶点 w_t。从源顶点 w_s 到顶点 $w(0, 0, 0, 0, 0)$之间有一条有向边，容量为 1，费用为 0。所有顶点 $w(d, 0, d, d, r)$到汇顶点 w_t 之间有一条有向边，容量为 1，费

用为 0。

网络 $G(S)$的最小费用流对应于 S 的一个解。

下面讨论如何计算 $C(j, k, l, f, r)$的最小(h_1, h_2, h_3)扩展。

设 out(j, k, l, f, h)是在 $C(j, k, l, f, r)$的扩展中执行 h 个障碍物外移所需的最少移动次数，back$(j, k, l, f, r, h_1, h_2)$是在 $C(j, k, l, f, r)$的扩展中执行 h_1 个障碍物前移和 h_2 个障碍物后移所需的最少移动次数。

上述两个函数除了返回所需的最少移动次数，还计算出 k^*、f^*和 l^*的值。k^*是执行移动后剩余的悬挂障碍物数，f^*对应于悬挂障碍物最远前移位置 v_{f^*}，l^*是障碍物从 v_{l^*}移动到 $u<v_{l^*+1}$的最大下标。

计算上述两个函数要用到障碍物在 S 中的位置信息，这可以在 $O(n)$时间内预计算。

out(j, k, l, f, h)可按照如下方式计算。

初始时 out$(j, k, l, f, 0)=0$。out$(j, k, l, f, h+1)$的值可以用 out(j, k, l, f, h)的值，以及障碍物到 B_{j+1} 中第 $h+1$ 近的空顶点的距离来计算。

由此可见，对所有可能的 h_1，out(j, k, l, f, h_1)可以在 $O\{\min\{d, |B_{j+1}|\}\}$时间内完成。

类似地，对所有可能的 h_2 和 h_3，back$(j, k, l, f, r, h_2, h_3)$可以在 $O((\min\{d, |\mathrm{Ch}(r, j)|\})^2)$时间内完成。

有了这两个函数，容易计算 $C(j, k, l, f, r)$的最小(h_1, h_2, h_3)扩展如下。

```
Extend(j, k, l, f, r, h1, h2, h3) {
    (c_{h1}, k_{h1}, l_{h1}, f_{h1}) ← out(j, k, l, f, h1);
    (c_{h2,h3}, k_{h2,h3}, l_{h2,h3}, f_{h2,h3}) ← back(j, k_{h1}, l_{h1}, f_{h1}, r, h2, h3);
    c=c_{h1}+c_{h2,h3}+k-k_{h2,h3}+1;
    if(h3>0)
        r_{h2,h3}=j;
    else
        r_{h2,h3}=r;
    return c;
}
```

由此得到，$C(j+1, k_{h_2,h_3}, l_{h_2,h_3}, f_{h_2,h_3}, r_{h_2,h_3})$。

8-9 方格取数问题。

问题描述：在一个有 $m\times n$ 个方格的棋盘中，每个方格中有一个正整数。现要从方格中取数，使任意两个数所在方格没有公共边，且取出的数的总和最大。试设计一个满足要求的取数算法。

算法设计：对于给定的方格棋盘，按照取数要求找出总和最大的数。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有 2 个正整数 m 和 n，分别表示棋盘的行数和列数。接下来的 m 行，每行有 n 个正整数，表示棋盘方格中的数。

结果输出：将取数的最大总和输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
3 3	11
1 2 3	

3 2 3
2 3 1

分析与解答：

（1）**解法 1**：用线性规划算法求解

设第 i 行第 j 列中的正整数为 c_{ij}，相应的变量为 x_{ij}。变量 x_{ij} 的含义是取了 x_{ij} 个(i,j)方格中放置的正整数 c_{ij}，$x_{ij}\in\{0,1\}$。

方格取数问题的求解目标是 $\sum_{i=1}^{m}\sum_{j=1}^{n}c_{ij}x_{ij}$ 达到最大。

约束条件是，取出的任意两个数所在方格没有公共边。

一般情况下，有 4 个方格与方格(i,j)有公共边。因此，一般情况下，方格(i,j)对应下面 4 个不等式约束：

$$x_{ij}+x_{ij-1}\leqslant 1$$
$$x_{ij}+x_{ij+1}\leqslant 1$$
$$x_{ij}+x_{i-1j}\leqslant 1$$
$$x_{ij}+x_{i+1j}\leqslant 1$$

由此可见，方格取数问题可以变换为如下的整数线性规划问题：

$$\max\sum_{i=1}^{m}\sum_{j=1}^{n}c_{ij}x_{ij}$$

$$\text{s.t.}\quad x_{ij}+x_{ij-1}\leqslant 1$$
$$x_{ij}+x_{ij+1}\leqslant 1$$
$$x_{ij}+x_{i-1j}\leqslant 1$$
$$x_{ij}+x_{i+1j}\leqslant 1$$
$$x_{ij}\in\{0,1\}$$

如果将相邻方格用黑白 2 种颜色着色，每个方格对应图 G 的一个顶点，相邻方格顶点之间连一条边，则得到的图 G 是一个二分图。上述整数线性规划的约束矩阵恰好是二分图 G 的关联矩阵。由已知结论，任何一个二分图的关联矩阵是一个全 1 模阵，可知上述整数线性规划的约束矩阵是一个全 1 模阵，因此可以将非线性约束条件 $x_{ij}\in\{0,1\}$松弛为线性约束条件 $x_{ij}\geqslant 0$，从而将整数线性规划问题转化为线性规划问题，可用单纯形算法求解。

具体算法实现如下。

read()函数读入初始数据。

```
void read() {
   cin>>m>>n;
   Make2DArray(c, m, n);
   for(int i=0; i < m; i++)
      for(int j=0; j < n; j++)
         cin>>c[i][j];
}
```

用线性规划算法的成员函数 lpin()建立相应的初始单纯形表。

```
void LinearProgram::lpin(int mima, int mm, int nn, int**cc) {
   int  i, j, k, t;
   double  value;
```

```
minmax = mima;
m = (mm-1)*nn+(nn-1)*mm;
n = mm*nn;
m1=  m;
m2 = 0;
m3 = 0;
n1=  n;
n2 = n+m1;
Make2DArray(a, m+2, n1+1);
basic = new int[m+2];
nonbasic = new int[n1+1];
for(i=0; i <= m+1; i++)
   for(j=0; j <= n1; j++)
      a[i][j] = 0.0;
for(j=0; j <= n1; j++)
   nonbasic[j] = j;
for(i=1, j=n1+1; i <= m; i++, j++)
   basic[i] = j;
for(i=m-m3+1, j=n+1; i <= m; i++, j++) {
   a[i][j] = -1.0;
   a[m+1][j] = -1.0;
}
for(k=1, t=1; k <= mm; k++) {
   for(i=1; i < nn; i++, t++) {
      j = (k-1)*nn + i;
      a[t][j] = 1.0;
      a[t][j+1] = 1.0;
   }
}
for(k=-1; k < mm; k++) {
   for(i=1; i <= nn; i++, t++) {
      j = (k-1)*nn + i;
      a[t][j] = 1.0;
      a[t][j+nn] = 1.0;
   }
}
for(j=1; j <= m; j++)
   a[j][0] = 1;
for(i=0; i < mm; i++) {
   for(j=0; j < nn; j++) {
      int  jj = i*nn+j+1;
      a[0][jj] = minmax*cc[i][j];
   }
}
for(j=1; j <= n; j++) {
   for(i=m1+1, value=0.0; i<=m;i++)
      value += a[i][j];
   a[m+1][j] = value;
}
```

```
}
```

算法的主函数调用单纯形算法求解。

```
int main() {
   read();
   LinearProgram X;
   X.lpin(1, m, n, c);
   X.solve();
   return 0;
}
```

（2）**解法 2**：用最大流算法求解

将相邻方格用黑白两种颜色着色，每个方格对应图 G 的一个顶点，相邻方格顶点之间连一条边，则所得到的图 G 是一个二分图。接着将每个方格顶点按照方格中的正整数 x 拆成 x 个顶点。原来连接方格顶点 x 和 y 的边增加为 $x \times y$ 条边。由此构成的图 G'仍是一个二分图。所求的最大取数方案对应该二分图的一个最大独立集。

设图 $G=(V, E)$的最大独立集中顶点数为 $\alpha(G)$，图 G 的最大匹配中边数为 $\alpha'(G)$，图 G 的最小顶点覆盖中顶点数为 $\beta(G)$，图 G 的最小边覆盖中边数为 $\beta'(G)$，则：

① 若 $S \subseteq V$ 是图 G 的独立集，则 $V-S$ 是图 G 的顶点覆盖，因此 $\alpha(G)+\beta(G)=n$。

② 若图 G 无孤立顶点，则 $\alpha'(G)+\beta'(G)=n$。

③ 若图 G 无孤立顶点的二分图，则 $\alpha'(G)=\beta'(G)$，从而 $\alpha(G)=n-\alpha'(G)$。

由此可见，所求问题可以转化为二分图的最大匹配问题。

另一方面，容易看出，拆点后二分图 G'的最大匹配对应于未拆点前二分图 G 的最大流，因此方格取数问题又转化为二分图 G 的最大流问题。

具体算法实现如下。

input()函数读入初始数据。

```
void GRID::input() {
   cin >> m >> n;
   Make2DArray(c, m+1, n+1);
   Make2DArray(d, m+1, n+1);
   Make2DArray(e, m+1, n+1);
   for(i=1; i<=m; i++)
      for(int j=1; j <= n; j++)
         cin >> c[i][j];
   d[0][0] = 0;
   for(int j=1; j <= m; j++)
      d[j][0] != d[j-1][0];
   for(i=1; i <= m; i++)
      for(j=1; j<=  n; j++)
         d[i][j] != d[i][j-1];
   for(i=1, nn=0; i <= m; i++)
      for(j=1; j <= n; j++)
         nn += c[i][j];
}
```

comp 构造相应的二分图 G，并通过计算网络最大流找出最大取数方案。

```
int GRID::comp() {
   int  i, j, s, t, maxflow = 0;
   GRAPH<EDGE>G(m*n+2, 1);
   for(i=1; i <= m; i++) {
      for(j=1; j < n; j++) {
         if(!d[i][j])
            adde(G, i, j, i, j+1);
         else
            adde(G, i, j+1, i, j);
        }
   }
   for(j=1; j <= n; j++) {
      for(i=1; i < m; i++) {
         if(!d[i][j])
            adde(G, i, j, i+1, j);
         else
            adde(G, i+1, j, i, j);
      }
   }
   s = m*n;
   t = m*n+1;
   for(i=1; i <= m; i++) {
      for(j=1; j <= n; j++) {
         int  x = (i-1)*n+j1;
         if(d[i][j])
            G.insert(new EDGE(x, t, c[i][j]));
         else
            G.insert(new EDGE(s, x, c[i][j]));
      }
   }
   MAXFLOW<GRAPH<EDGE>, EDGE>(G, s, t, maxflow);
   return nn-maxflow;
}
void GRID::adde(GRAPH<EDGE>&G, int x1,i nt y1, int x2, int y2) {
   int  i = (x1-1)*n + y1-1, j = (x2-1)*n + y2-1;
   G.insert(new EDGE(i, j, c[x1][y1]));
}
```

8-10　餐巾计划问题。

问题描述：一个餐厅在相继的 N 天里，每天需用的餐巾数不尽相同。假设第 i 天需要 r_i 块餐巾（i=1, 2, …, N）。餐厅可以购买新的餐巾，每块餐巾的费用为 p 分；或者把旧餐巾送到快洗部，洗一块需 m 天，其费用为 f 分；或者送到慢洗部，洗一块需 n 天（n>m），其费用为 s 分（s<f）。每天结束时，餐厅必须决定将多少块脏的餐巾送到快洗部，多少块餐巾送到慢洗部，以及多少块保存起来延期送洗。但是每天洗好的餐巾和购买的新餐巾数之和要满足当天的需求量。试设计一个算法，为餐厅合理地安排好 N 天中餐巾使用计划，使总的花费最小。

算法设计：编程找出一个最佳餐巾使用计划。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有 6 个正整数 N、p、m、f、n、s。N 是要安排餐巾使用计划的天数，p 是每块新餐巾的费用，m 是快洗部洗一块餐巾需用天数，f 是快洗部洗一块餐巾需要的费用，n 是慢洗部洗一块餐巾需用天数，s 是慢洗部洗一块餐巾需要的费用。接下来的 N 行是餐厅在相继的 N 天里，每天需用的餐巾数。

结果输出：将餐厅在相继的 N 天里使用餐巾的最小总花费输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
3 10 2 3 3 2	145
5	
6	
7	

分析与解答：用最小费用流算法求解。

对于给定条件构造网络 G 如下。每天对应图 G 的两个顶点 X_i 和 Y_i。

① 顶点 X_i 和 X_{i+1} 之间有一条有向边，容量为∞，费用为 0，表示第 i 天可以保存旧餐巾到第 i+1 天，延期送洗。

② 当 $i+m \leqslant N$ 时，顶点 X_i 和 Y_{i+m} 之间有一条有向边，容量为∞，费用为 f，表示第 i 天可以把旧餐巾送到快洗部。

③ 当 $i+n \leqslant N$ 时，顶点 X_i 和 Y_{i+n} 之间有一条有向边，容量为∞，费用为 s，表示第 i 天可以把旧餐巾送到慢洗部。新增源 s 和汇 t，以及附加顶点 k。

④ 从源 s 到每个顶点 X_i 有一条有向边，其容量为第 i 天餐巾需求量 r_i，费用为 0。

⑤ 从附加顶点 k 到每个顶点 Y_i 有一条有向边，其容量为∞，费用为 p，表示第 i 天可以购买新餐巾。

⑥ 从每个顶点 Y_i 到汇 t 有一条有向边，其容量为第 i 天餐巾需求量 r_i，费用为 0。

然后求网络 G 的最小费用流。

8-11　航空路线问题。

问题描述：给定一张航空图，图中顶点代表城市，边代表两个城市间的直通航线。现要求找出一条满足下述限制条件且途经城市最多的旅行路线：

① 从最西端城市出发，单向从西向东途经若干城市到达最东端城市，再单向从东向西飞回起点（可途经若干城市）。

② 除起点城市外，任何城市只能访问 1 次。

算法设计：对于给定的航空图，试设计一个算法，找出一条满足要求的最佳航空旅行路线。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有两个正整数 N 和 V，N 表示城市数（N<100），V 表示直飞航线数。接下来的 N 行中的每行是一个城市名，可乘飞机访问这些城市。城市名出现的顺序是从西向东。也就是说，设 i、j 是城市表列中城市出现的顺序，当 $i>j$ 时，表示城市 i 在城市 j 的东边，而且不会有两个城市在同一条经线上。城市名是一个长度不超过 15 的字符串，串中的字符可以是字母或阿拉伯数字，如 AGR34 或 BEL4。

再接下来的 V 行中，每行有 2 个城市名，中间用空格隔开，如 city1 city2 表示 city1 到 city2 有一条直通航线，从 city2 到 city1 也有一条直通航线。

结果输出：将最佳航空旅行路线输出到文件 output.txt。文件第 1 行是旅行路线中所访问

的城市总数 M。接下来的 M+1 行是旅行路线的城市名，每行写一个城市名。首先是起点城市名，然后按访问顺序列出其他城市名。注意，最后一行（终点城市）的城市名必然是起点城市名。如果问题无解，则输出“No Solution!”。

输入文件示例	输出文件示例
input.txt	output.txt
8 9	7
Vancouver	Vancouver
Yellowknife	Edmonton
Edmonton	Montreal
Calgary	Halifax
Winnipeg	Toronto
Toronto	Winnipeg
Montreal	Calgary
Halifax	Vancouver
Vancouver Edmonton	
Vancouver Calgary	
Calgary Winnipeg	
Winnipeg Toronto	
Toronto Halifax	
Montreal Halifax	
Edmonton Montreal	
Edmonton Yellowknife	
Edmonton Calgary	

分析与解答：对于给定的航空图构造网络 G 如下。

① 每个城市 i 对应图 G 的两个顶点 i 和 i'，并在两个顶点之间连接一条由 i 至 i'的有向边，边的容量为 1，表示该市最多只能被访问一次。为了使该城市尽可能被访问，单位流量费用设为 0。

② 若城市 i 到城市 j 有直通航线（$i<j$），则在顶点 i'与顶点 j 之间连接一条边，方向由顶点 i'至顶点 j。边的容量为 1'，表示这条航线最多只能通过一次。单位流量费用设为 $j-i+1$，即城市 j 位于城市 i 的东面，中间间隔的城市数为 $j-i+1$。

③ 顶点 1 与顶点 2 之间的边容量改为 2，表示最西端的城市 1 被访问 2 次。顶点 n 与顶点 n'的边容量也改为 2。因为如果将往返航线看作两条不同路线，则最东端的城市 n 可以看作被访问 2 次。

④ 顶点 1 作为源，顶点 n'作为汇。

然后对上述网络求最小费用最大流，其结果是：一条由顶点 1 至顶点 n，边容量为 1 且互相连接的若干条边组成的流，表示往程航线；另一条由顶点 n 至顶点 1，边容量为 1 且互相连接的若干条边组成的流，表示返程航线。这两条流上的(顶点序号+1)/2 即为往返航线上被访问的城市序号。显然最佳航空路线上经过的城市数=$2\times(n-1)$−流量费用。

8-12　软件补丁问题。

问题描述：T 公司发现其研制的一个软件中有 n 个错误，随即为该软件发放了一批共 m 个补丁程序。每个补丁程序都有其特定的适用环境，某补丁只有在软件中包含某些错误而同时又不包含另一些错误时才可以使用。一个补丁在排除某些错误的同时，往往会加入另一些

错误。换句话说，对于每个补丁 i，都有两个与之相应的错误集合 $B1[i]$和 $B2[i]$，使得仅当软件包含 $B1[i]$中的所有错误，而不包含 $B2[i]$中的任何错误时，才可以使用补丁 i。补丁 i 将修复软件中的某些错误 $F1[i]$，同时加入另一些错误 $F2[i]$。另外，每个补丁都耗费一定的时间。

试设计一个算法，利用 T 公司提供的 m 个补丁程序，将原软件修复成一个没有错误的软件，并使修复后的软件耗时最少。

算法设计：对于给定的 n 个错误和 m 个补丁程序，找到总耗时最少的软件修复方案。

数据输入：由文件 input.txt 提供输入数据。文件第 1 行有 2 个正整数 n 和 m，n 表示错误总数，m 表示补丁总数（$1 \leqslant n \leqslant 20$，$1 \leqslant m \leqslant 100$）。接下来 m 行给出了 m 个补丁的信息。每行包括一个正整数，表示运行补丁程序 i 所需时间以及 2 个长度为 n 的字符串，中间用一个空格符隔开。在第 1 个字符串中，如果第 k 个字符 bk 为“+”，则表示第 k 个错误属于 $B1[i]$，若为“-”，则表示第 k 个错误属于 $B2[i]$，若为“0”，则第 k 个错误既不属于 $B1[i]$也不属于 $B2[i]$，即软件中是否包含第 k 个错误并不影响补丁 i 的可用性。在第 2 个字符串中，如果第 k 个字符 bk 为“+”，则表示第 k 个错误属于 $F1[i]$，若为“-”，则表示第 k 个错误属于 $F2[i]$，若为“0”，则第 k 个错误既不属于 $F1[i]$也不属于 $F2[i]$，即软件中是否包含第 k 个错误不会因使用补丁 i 而改变。

结果输出：将总耗时数输出到文件 output.txt。如果问题无解，则输出 0。

输入文件示例	输出文件示例
input.txt	output.txt
3 3	8
1 000 00-	
1 00- 0-+	
2 0- - -++	

分析与解答：用 n 位串 a 来表示用补丁软件过程中 n 个错误的状态，a[i]=1 表示软件中存在第 i 个错误，a[i]=0 表示软件中第 i 个错误已修正。对于第 i 个补丁软件 mend[i]，用两个 n 位串 b1 和 b2 来表示错误集合 $B1[i]$和 $B2[i]$；用两个 n 位串 f1 和 f2 来表示错误集合 $F1[i]$和 $F2[i]$。在修复软件的任何状态 a，补丁软件 mend[i]可运行的条件是

$$a \ \& \ mend[i].b1 = mend[i].b1$$

$$a \ \& \ mend[i].b2 = 0$$

运行补丁软件 mend[i]后，软件状态 a 转换为软件状态 b=(a |mend[i].f1) & (~mend[i].f2)。

软件修复状态构成网络 $G=(V, E)$如下：每个软件状态对应于网络中的一个顶点。从软件状态 a=(11…1)出发，当补丁软件 mend[i]可用于当前顶点 a，并产生新的状态顶点 b 时，增加一条网络边(a, b)，其流量为 1，费用为 time[i]（软件 mend[i]的耗时）。另外增加源 s 和汇 t。从源 s 到初始状态 a=(11…1)有一条有向边，其容量为 1，费用为 0。从目标状态 b=(00…0)到汇 t 有一条有向边，其容量为 1，费用为 0。

求相应网络 G 的最小费用流。

8-13　星际转移问题。

问题描述：由于人类对自然资源的消耗，人们意识到大约在 2300 年后，地球就不能再居住了。于是在月球上建立了新的绿地，以便在需要时移民。令人意想不到的是，2177 年冬由于未知的原因，地球环境发生了连锁崩溃，人类必须在最短的时间内迁往月球。现有 n 个太空站位于地球与月球之间，且有 m 艘公共交通太空船在其间来回穿梭。每个太空站可

容纳无限多的人，而每艘太空船 i 只可容纳 $H[i]$个人。每艘太空船将周期性地停靠一系列的太空站，例如，(1, 3, 4)表示该太空船将周期性地停靠太空站 134134134…每艘太空船从一个太空站驶往任一太空站耗时均为 1。人们只能在太空船停靠太空站（或月球、地球）时上下船。初始时，所有人全在地球上，太空船全在初始站。试设计一个算法，找出让所有人尽快全部转移到月球上的运输方案。

算法设计：对于给定的太空船的信息，找到让所有人尽快全部转移到月球上的运输方案。

数据输入：由文件 *in*put.txt 提供输入数据。文件第 1 行有 3 个正整数 n（太空站个数）、m（太空船个数）和 k（需要运送的地球上的人数）。其中，$1\leqslant m\leqslant 13$，$1\leqslant n\leqslant 20$，$1\leqslant k\leqslant 50$。

接下来的 m 行给出太空船的信息。第 i+1 行说明太空船 pi。第 1 个数表示 pi 可容纳的人数 Hpi；第 2 个数表示 pi 一个周期停靠的太空站个数 r（$1\leqslant r\leqslant n+2$）；随后 r 个数是停靠的太空站的编号 $Si1, Si2, \cdots, Sir$，地球用 0 表示，月球用−1 表示。时刻 0 时，所有太空船都在初始站，然后开始运行。在时刻 1、2、3、…等正点时刻各艘太空船停靠相应的太空站。人只有在 0、1、2、…等正点时刻才能上、下太空船。

结果输出：将全部人员安全转移所需的时间输出到文件 output.txt。如果问题无解，则输出 0。

输入文件示例	输出文件示例
input.txt	output.txt
2 2 1	5
1 3 0 1 2	
1 3 1 2 −1	

分析与解答：首先考察下面的问题：在已知时间 T 内，能转移到月球上的最多人数 maxp(T)是多少。如果这个问题得到解决，则原问题变成这个问题的反问题。

将地球、太空站和月球编号为 0, 1, …, n, n+1。其中 0 表示地球，n+1 表示月球，1, 2, …, n 分别表示太空站 1, 2, …, n。

建立网络 $G=(V, E)$如下：$V=\{u(i, j)|\ i=0, 1, \cdots, T;\ j=0, 1, \cdots, n, n+1\}$。式中，$u(i, j)$对应太空站 j 在时刻 i 的状态。从时刻 i 到时刻 i+1 太空站 j 的人流可流向太空站 0, 1, …, n, n+1。因此，从顶点 $u(i, j)$到顶点 $u(i+1, k)$有一条边$(u(i, j), u(i+1, k))$，其容量 $r(u(i, j), u(i+1, k))$为

$$r(u(i,j),u(i+1,k))=\begin{cases}\sum t(s) & j\neq k\\ \infty & j=k\end{cases}$$

式中，$t(s)$表示太空船 s 的容量；求和是对所有满足 stay(s, i)=j 且 stay(s, i+1)=k 的 s 进行。stay(s, i)表示太空船 s 在时刻 i 所处的位置。

设源 $s=u(0, 0)$和汇 $t=(T, n+1)$，网络 G 的从 s 到 t 的最大流量即为 maxp(T)。

有了求 maxp(T)的算法，就可以逐步增加 T 值找出 maxp(T)≥k 的最小 T 值。在逐步增加 T 值的过程中，应充分利用已找出的最大流来计算 T 值增加后的最大流。

8-14 孤岛营救问题。

问题描述：1944 年，特种兵麦克接到美国国防部的命令，要求立即赶赴太平洋上的一个孤岛，营救被敌军俘虏的大兵瑞恩。瑞恩被关押在一个迷宫里，迷宫地形复杂，但幸好麦克得到了迷宫的地形图。迷宫的外形是一个长方形，其南北方向被划分为 N 行，东西方向被划分为 M 列，于是整个迷宫被划分为 $N\times M$ 个单元。每个单元的位置可用一个有序数对(单

元的行号，单元的列号)来表示。南北或东西方向相邻的两个单元之间可能互通，也可能有一扇锁着的门，或者是一堵不可逾越的墙。迷宫中有一些单元存放着钥匙，并且所有的门被分成 P 类，打开同一类的门的钥匙相同，不同类门的钥匙不同。

大兵瑞恩被关押在迷宫的东南角，即(N, M)单元里，并已经昏迷。迷宫只有一个入口，在西北角。也就是说，麦克可以直接进入$(1, 1)$单元。另外，麦克从一个单元移动到另一个相邻单元的时间为 1，拿取所在单元钥匙的时间及用钥匙开门的时间可忽略不计。

算法设计：试设计一个算法，帮助麦克以最快的方式到达瑞恩所在单元，营救大兵瑞恩。

数据输入：由文件 input.txt 提供输入数据。第 1 行有 3 个整数，分别表示 N、M、P 的值。第 2 行是 1 个整数 K，表示迷宫中门和墙的总数。第 I+2 行（$1 \leqslant I \leqslant K$），有 5 个整数，依次为 $Xi1$、$Yi1$、$Xi2$、$Yi2$、Gi：

当 $Gi \geqslant 1$ 时，表示$(Xi1, Yi1)$单元与$(Xi2, Yi2)$单元之间有一扇第 Gi 类的门；当 $Gi=0$ 时，表示$(Xi1, Yi1)$单元与$(Xi2, Yi2)$单元之间有一堵不可逾越的墙（其中，$|Xi1-Xi2|+|Yi1-Yi2|=1$，$0 \leqslant Gi \leqslant P$）。

第 K+3 行是一个整数 S，表示迷宫中存放的钥匙总数。

第 K+3+J 行（$1 \leqslant J \leqslant S$）有 3 个整数，依次为 $Xi1$、$Yi1$、Qi：表示第 J 把钥匙存放在（$Xi1$、$Yi1$）单元里，并且第 J 把钥匙是用来开启第 Qi 类门的（其中 $1 \leqslant Qi \leqslant P$）。

输入数据中同一行各相邻整数之间用一个空格分隔。

结果输出：将麦克营救到大兵瑞恩的最短时间值输出到文件 output.txt。如果问题无解，则输出-1。

输入文件示例	输出文件示例
input.txt	output.txt
4 4 9	14
9	
1 2 1 3 2	
1 2 2 2 0	
2 1 2 2 0	
2 1 3 1 0	
2 3 3 3 0	
2 4 3 4 1	
3 2 3 3 0	
3 3 4 3 0	
4 3 4 4 0	
2	
2 1 2	
4 2 1	

分析与解答：用 p 位串 a 来表示钥匙的状态，a[i]=1 表示存在打开 i 类门的钥匙，a[i]=0 表示不存在打开 i 类门的钥匙。特种兵麦克的状态可以用三元组(x, y, z)来表示，(x, y)是当前位置，z 是当前持有钥匙的状态。从状态(x, y, z)变换到状态$(x1, y1, z1)$的条件是，对于 i=1, 2, 3, 4（东，南，西，北）：

$$x1=x+\mathrm{dir}(i, 1)$$
$$y1=y+\mathrm{dir}(i, 2)$$
$$z1=z|\mathrm{put}(x1, y1)$$

$$z \,\&\, \mathrm{map}(x, y, i) = \mathrm{map}(x, y, i)$$

式中，dir(i, 1)，dir(i, 2)表示第 i 个方向上的坐标增量；put(x, y)表示在位置(x, y)处存放钥匙情况的 p 位串；map(x, y, i)也是 p 位串，表示在位置(x, y)的第 i 个方向上门的类型，仅当门的类型为 j 时，map(x, y, i)的第 j 位为 1。要求(x, y)与($x1$, $y1$)之间没有墙。

特种兵状态构成网络 G=(V, E)如下：每个状态对应网络中的一个顶点，从软件状态 a=(1, 1, put(1, 1))出发，从当前顶点(x, y, z)变换到状态($x1$, $y1$, $z1$)时，增加一条网络边((x, y, z), ($x1$, $y1$, $z1$))，其流量为 1，费用为 1。新增源 s 和汇 t。从源 s 到初始状态 a=(1, 1, put(1, 1))有一条有向边，其容量为 1，费用为 0。从目标状态 b=(m, n, s)到汇 t 有一条有向边，其容量为 1，费用为 0。

求相应网络 G 的最小费用流。

8-15 汽车加油行驶问题。

问题描述：给定一个 N×N 的交通方形网格，设其左上角为起点◎，坐标为(1, 1)，X 轴向右为正，Y 轴向下为正，每个方格边长为 1，汽车加油行驶问题的交通方形网格如图 8-2 所示。一辆汽车从起点◎出发驶向右下角终点▲，其坐标为(N, N)。在若干个网格交叉点处，设置了油库，可供汽车在行驶途中加油。汽车在行驶过程中应遵守如下规则：

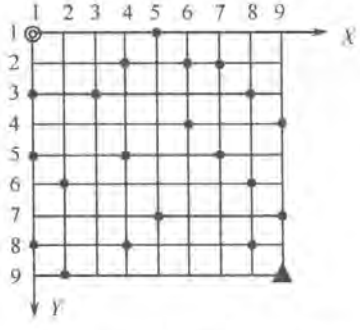

图 8-2 汽车加油行驶问题的交通方形网格

① 汽车只能沿网格边行驶，装满油后能行驶 K 条网格边。出发时汽车已装满油，在起点与终点处不设油库。

② 汽车经过一条网格边时，若其 X 坐标或 Y 坐标减小，则应付费用 B，否则免付费用。

③ 汽车在行驶过程中遇油库，应加满油并付加油费用 A。

④ 在需要时可在网格点处增设油库，并付增设油库费用 C（不含加油费用 A）。

⑤ ①～④中的各数 N、K、A、B、C 均为正整数，且满足约束：$2 \leqslant N \leqslant 100$，$2 \leqslant K \leqslant 10$。

设计一个算法，求出汽车从起点出发到达终点的一条所付费用最少的行驶路线。

算法设计：对于给定的交通网格，计算汽车从起点出发到达终点的一条所付费用最少的行驶路线。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行是 N、K、A、B、C 的值。第 2 行起是一个 N×N 的 0-1 方阵，每行 N 个值，至 N+1 行结束。方阵的第 i 行第 j 列处的值为 1 表示在网格交叉点(i, j)处设置了一个油库，为 0 时表示未设油库。各行相邻两个数以空格分隔。

结果输出：将最小费用输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
9 3 2 3 6	12
0 0 0 0 1 0 0 0 0	
0 0 0 1 0 1 1 0 0	
1 0 1 0 0 0 0 1 0	
0 0 0 0 0 1 0 0 1	
1 0 0 1 0 0 1 0 0	
0 1 0 0 0 0 0 1 0	

0 0 0 0 1 0 0 0 1
1 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0

分析与解答：汽车行驶的状态可以用三元组(x, y, z)来表示，(x, y)是当前位置，z 表示汽车还可行驶 z 条网格边。对于 i=1, 2, 3, 4（东，南，西，北），$x1$=x+dir(i, 1)；$y1$=y+dir(i, 2)；从状态(x, y, z)可以变换到状态$(x1, y1, z-1)$。其中，dir(i, 1)和 dir(i, 2)表示第 i 个方向上的坐标增量。

汽车行驶状态构成网络 $G=(V, E)$如下：

每个网格点(x, y)对应于网络中的 K+2 个顶点(x, y, z)（$0 \leqslant z \leqslant K+1$）。当 $0 \leqslant z \leqslant K$ 时，网络顶点(x, y, z)对应汽车行驶的状态；当 $z=K+1$ 时，网络顶点$(x, y, K+1)$对应于加油站。

设 $x1=x+\text{dir}(i, 1)$，$y1=y+\text{dir}(i, 2)$，i=1, 2, 3, 4。

① 当 $1 \leqslant z \leqslant K$，且$(x1, y1)$不是加油站时，从顶点$(x, y, z)$到顶点$(x1, y1, z-1)$有一条有向边，容量为 1；当 $x1<x$ 或 $y1<y$ 时费用为 B，否则费用为 0。

② 当 $1 \leqslant z \leqslant K$，且$(x1, y1)$是加油站时，从顶点$(x, y, z)$到顶点$(x1, y1, K+1)$有一条有向边，容量为 1，费用为 A。

③ 当 $z=0$ 且(x, y)不是加油站时，从顶点$(x, y, 0)$到顶点$(x, y, K+1)$有一条有向边，容量为 1，费用为 $A+C$。

④ 从顶点$(x, y, K+1)$到顶点(x, y, K)有一条有向边，容量为∞，费用为 0。新增源 s 和汇 t。

⑤ 从源 s 到初始状态$(1, 1, K)$有一条有向边，其容量为 1，费用为 0。

⑥ 从目标状态(n, n, z)（$0 \leqslant z < K$）到汇 t 有一条有向边，其容量为 1，费用为 0。

求相应网络 G 的最小费用流。

具体算法实现如下。

read()函数读入初始数据。

```
int  n, k, a, b, c, s, t, f, nn, **map;
int  dir[4][2];
void read() {
   dir[0][0] = 0;
   dir[0][1] = 1;
   dir[1][0] = 1;
   dir[1][1] = 0;
   dir[2][0] = 0;
   dir[2][1] = -1;
   dir[3][0] = -1;
   dir[3][1] = 0;
   cin >> n >> k >> a >> b >> c;
   Make2DArray(map, n+1, n+1);
   for(int i=1; i <= n; i++)
      for(int j=1; j <= n; j++)
         cin >> map[i][j];
}
```

construct()函数构造相应的网络。

```
int num(int i, int j) {
   return((i-1)*n+j-1)*(k+2)+2;
```

```
}
void constructG(GRAPH<EDGE>&G) {
   s = 0;
   t = 1;
   f = 1;
   int  i, j, d, e, cost, i1, j1, k1, k2;
   for(i=1; i <= n; i++) {
      for(j=1; j <= n; j++) {
         k1=num(i, j);
         for(e=1; e <= k; e++) {
            for(d=0; d < 4; d++) {
               i1 = i+dir[d][0];
               j1 = j+dir[d][1];
               if(i1 > 0 && i1 <= n && j1 > 0 && j1 <= n) {
                  cost = (d>1) ? b : 0;
                  k2 = num(i1, j1);
                  if(map[i1][j1])
                     G.insert(new EDGE(k1+e, k2+k+1, 1, cost+a));
                  else
                     G.insert(new EDGE(k1+e, k2+e-1, 1, cost));
               }
            }
         }
         if(!map[i][j])
            G.insert(new EDGE(k1, k1+k+1, 1, a+c));
         G.insert(new EDGE(k1+k+1, k1+k, 5*(k+2), 0));
      }
   G.insert(new EDGE(s, k+2, 1, 0));
   k1=num(n, n);
   for(e=0; e <= k; e++)
      G.insert(new EDGE(k1+e, t, 1, 0));
}
```

实现算法的主函数如下。

```
int main() {
   read();
   nn = n*n*(k+2) + 1;
   GRAPH<EDGE>G(nn, 1);
   constructG(G);
   MINCOST<GRAPH<EDGE>, EDGE>(G, s, t, f);
   output(G);
   return 0;
}
```

output()函数输出最小费用。

```
void output(GRAPH<EDGE>&G) {
   int  sum = 0;
   for(int i=2; i <= nn; i++) {
      adjIterator<EDGE>A(G, i);
```

```
    for(EDGE*e=A.beg(); !A.end(); e=A.nxt())
      if(e->from(i) && e->flow()>0 && e->cost() > 0)
        sum += e->cost()*e->flow();
  }
  cout << sum << endl;
}
```

8-16 数字梯形问题。

问题描述：给定一个由 n 行数字组成的数字梯形，如图 8-3 所示。梯形的第 1 行有 m 个数字。从梯形的顶部的 m 个数字开始，在每个数字处可以沿左下或右下方向移动，形成一条从梯形的顶至底的路径。

规则 1：从梯形的顶至底的 m 条路径互不相交。

规则 2：从梯形的顶至底的 m 条路径仅在数字结点处相交。

规则 3：从梯形的顶至底的 m 条路径允许在数字结点处相交或在边处相交。

```
      2  3
    3  4  5
  9  10  9  1
 1  1  10  1  1
1  1  10  12  1  1
```

图 8-3 数字梯形

算法设计：对于给定的数字梯形，分别按照规则 1、规则 2 和规则 3 计算出从梯形的顶至底的 m 条路径，使这 m 条路径经过的数字总和最大。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行中有 2 个正整数 m 和 n（$m, n \leq 20$），分别表示数字梯形的第 1 行有 m 个数字，共有 n 行。接下来的 n 行是数字梯形中各行的数字。第 1 行有 m 个数字，第 2 行有 $m+1$ 个数字……

结果输出：将按照规则 1、规则 2 和规则 3 计算出的最大数字总和输出到文件 output.txt。每行一个最大总和。

输入文件示例	输出文件示例
input.txt	output.txt
2 5	66
2 3	75
3 4 5	77
9 10 9 1	
1 1 10 1 1	
1 1 10 12 1 1	
2 3	
3 4 5	
9 10 9 1	
11 10 11	
111012 1 1	

分析与解答：数字梯形构成网络 $G=(V, E)$如下：

数字梯形中的每个数字对应网络 G 的一个顶点，每个顶点处沿左下和右下方向有一条有向边。新增源 s 和汇 t，从源 s 到数字梯形每个顶层顶点有一条有向边，其容量为 1，费用为 0。从数字梯形每个底层顶点到汇 t 有一条有向边，其容量为 1，费用为 0。

按照规则 1，在每个顶点处有一个顶点容量约束，即经过数字 x 所在顶点的容量为 1，费用为 x。

按照规则 2，在每条边处有边容量约束，即以数字 x 所在顶点为始发边的容量为 1，费用为 x。

按照规则3，在每条边处没有边容量约束，即以数字x所在顶点为始发边的容量为∞，费用为x。

按照以上3种规则，构造3种不同的网络，求相应网络的最大费用流。

具体算法用类DIGIT实现如下。

```
class DIGIT {
  int  n, mm, nn, f, s, t, maxc, **a;
  int num(int i, int j) {
    return ((i-1)*mm+(i-1)*(i-2)/2+j)*2;
  }
  int num1(int i, int j) {
    return (i-1)*mm+(i-1)*(i-2)/2+j+1;
  }
  void constructG(GRAPH<EDGE>&G) {
    f = mm;
    s = 0;
    t = 1;
    for(int i=1, kk=mm; i < nn; i++, kk++) {
      for(int j=1; j <= kk; j++) {
        int  k = num(i,j), k1 = num(i+1, j), k2 = num(i+1, j+1);
        G.insert(new EDGE(k, k+1, 1, a[i][j]));
        G.insert(new EDGE(k+1, k1, 1, 0));
        G.insert(new EDGE(k+1, k2, 1, 0));
        if(i == 1)
          G.insert(new EDGE(s, k, 1, 0));
    }
    for(int j=1; j < mm+nn; j++) {
      int  k = num(nn, j);
       G.insert(new EDGE(k, k+1, 1, a[nn][j]));
       G.insert(new EDGE(k+1, t, 1, 0));
    }
  }
  void constructG1(GRAPH<EDGE>&G) {
    f = mm;
    s = 0;
    t = 1;
    for(int i=1, kk=mm; i < nn; i++, kk++) {
      for(int j=1; j <= kk; j++) {
        int  k = num1(i, j), k1 = num1(i+1, j), k2 = num1(i+1, j+1);
        G.insert(new EDGE(k, k1, 1, a[i+1][j]));
        G.insert(new EDGE(k, k2, 1, a[i+1][j+1]));
        if(i == 1)
          G.insert(new EDGE(s, k, 1, a[i][j]));
      }
      for(int j=1; j < mm+nn; j++) {
        int  k = num1(nn, j);
        G.insert(new EDGE(k, t, mm, 0));
      }
    }
```

```
  void constructG2(GRAPH<EDGE>&G) {
    f = mm;
    s = 0;
    t = 1;
    for(int i=1, kk=mm; i < nn; i++, kk++) {
      for(int j=1; j <= kk; j++) {
        int  k = num1(i, j), k1 = num1(i+1, j), k2 = num1(i+1, j+1);
        G.insert(new EDGE(k, k1, mm, a[i+1][j]));
        G.insert(new EDGE(k, k2, mm, a[i+1][j+1]));
        if(i == 1)
          G.insert(new EDGE(s, k, 1, a[i][j]));
      }
      for(int j=1; j < mm+nn; j++) {
        int  k = num1(nn, j);
        G.insert(new EDGE(k, t, mm, 0));
      }
      Delete2DArray(a, nn+mm+1);
    }
  }
  void read(char *filename) {
    int  i, j, k;
    ifstream  inFile;
    inFile.open(filename);
    inFile>>mm>>nn;
    Make2DArray(a, nn+mm+1 ,nn+mm+1);
    for (i=1, maxc=0, k=mm; i <= nn; i++, k++) {
      for(j=1; j <= k; j++) {
        inFile >> a[i][j];
        if(maxc < a[i][j])
          maxc = a[i][j];
      }
    }
  }
  inFile.close();
  for(i=1, k=  mm;i <= nn; i++, k++)
    for(j=1; j <= k; j++)
      a[i][j] = maxca[i][j];
}
void trans(int i, int &u, int &v) {
  u = 0;
  v = i;
  if(i < 2)
    return;
  int  k = i/2;
  for(int j=1; j <= nn; j++) {
    int  ij = j*mm+j*(j1)/2;
    if(k <= ij) {
      u = j;
      v = k-(j-1)*mm-(j-1)*(j-2)/2;
```

```
            break;
        }
    }
}
void trans1(int i, int &u, int &v) {
    u = 0;
    v= i;
    if(i < 2)
        return;
    i--;
    for(int j=1; j<=  nn; j++) {
        int  ij = j*mm+j*(j-1)/2;
        if(i<= ij) {
            u = j;
            v = i-(j-1)*mm-(j-1)*(j-2)/2;
            break;
        }
    }
}
void output(GRAPH<EDGE>&G) {
    int  u1, v1, u2, v2, sum = 0;
    adjIterator<EDGE>A(G, s);
    for(EDGE*e=A.beg(); !A.end(); e=A.nxt())
        if(e->from(s) && e->flow()>0)
            sum++;
    if(sum<f) {
        cout << "No Solution!" << sum << endl;
        return;
    }
    sum = 0;
    for(int i=2; i <= n; i++) {
        adjIterator<EDGE>A(G, i);
        for(EDGE*e=A.beg(); !A.end(); e=A.nxt()) {
            if(e->from(i) && e->flow()>0) {
                trans(i, u1, v1);
                trans(e->w(), u2, v2);
                sum += e->cost()*e->flow();
            }
        }
    }
    cout << mm*nn*maxc-sum << endl;
}
void output1(GRAPH<EDGE>&G) {
    int  u1, v1, u2, v2, sum = 0;
    adjIterator<EDGE>A(G, s);
    for(EDGE*e=A.beg(); !A.end(); e=A.nxt())
        if(e->from(s) && e->flow()>0)
            sum++;
    if(sum<f) {
```

```
            cout << "No Solution!" << sum << endl;
            return;
        }
        sum = 0;
        for(int i=0; i <= n; i++) {
            adjIterator<EDGE>A(G,i);
            for(EDGE*e=A.beg(); !A.end(); e=A.nxt()) {
                if(e->from(i) && e->flow()>0) {
                    trans1(i, u1, v1);
                    trans1(e->w(), u2, v2);
                    sum += e->cost()*e->flow();
                }
            }
        }
        cout<<  mm*nn*maxc-sum << endl;
    }

public:
    DIGIT(char *filename) {
        read(filename);
        n = 2*mm*nn + (nn-1)*nn + 1;
        GRAPH<EDGE>G(n, 1);
        constructG(G);
        MINCOST<GRAPH<EDGE>, EDGE>(G, s, t, f);
        output(G);
        n = mm*nn + (nn-1)*nn/2 + 3;
        GRAPH<EDGE>G1(n, 1);
        constructG1(G1);
        MINCOST<GRAPH<EDGE>, EDGE>(G1, s, t, f);
        output1(G1);
        constructG2(G1);
        MINCOST<GRAPH<EDGE>, EDGE>(G1, s, t, f);
        output1(G1);
    }
};
```

其中，constructG()函数按照规则 1 构造相应网络，constructG1()函数按照规则 2 构造相应网络，constructG2()函数按照规则 3 构造相应网络。

实现具体计算的主函数如下。

```
int main(){
    create();
    DIGIT("digit.in");
    return 0;
}
```

8-17 运输问题。

问题描述：W 公司有 m 个仓库和 n 个零售商店。第 i 个仓库有 a_i 个单位的货物；第 j

个零售商店需要 b_j 个单位的货物。货物供需平衡，即 $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$。从第 i 个仓库运送每单位货物到第 j 个零售商店的费用为 c_{ij}。试分别设计一个将仓库中所有货物运送到零售商店的最优和最差运输方案，即使总运输费用最少或最多。

算法设计：对于给定的 m 个仓库和 n 个零售商店间运送货物的费用，计算最优运输方案和最差运输方案。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行有 2 个正整数 m 和 n，分别表示仓库数和零售商店数。接下来的一行中有 m 个正整数 a_i（$1 \leqslant i \leqslant m$），表示第 i 个仓库有 a_i 个单位的货物。再接下来的一行中有 n 个正整数 b_j（$1 \leqslant j \leqslant n$），表示第 j 个零售商店需要 b_j 个单位的货物。接下来的 m 行，每行有 n 个整数，表示从第 i 个仓库运送每单位货物到第 j 个零售商店的费用 c_{ij}。

结果输出：将计算的最少运输费用和最多运输费用输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
2 3	48500
220 280	69140
170 120 210	
77 39 105	
150 186 122	

分析与解答：设最优运输方案从第 i 个仓库运送 x_{ij} 单位货物到第 j 个零售商店。运输问题可以表述为如下的线性规划问题。

$$\min \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} x_{ij} = b_j \quad j = 1,2,\cdots,n$$

$$\sum_{j=1}^{n} x_{ij} = a_i \quad i = 1,2,\cdots,m$$

$$x_{ij} \geqslant 0, i = 1,2,\cdots,m, j = 1,2,\cdots,n$$

用单纯形算法求解，最差运输方案相应于求最大值的同一线性规划问题。具体算法实现如下。

read()函数读入初始数据。

```
int  m, n, *a, *b, **c;
void read() {
   cin >> m >> n;
   a = new int[m];
   b = new int[n];
   Make2DArray(c, m, n);
   for(int i=0; i < m; i++)
      cin >> a[i];
   for(int j=0; j < n; j++)
      cin >> b[j];
```

```
    for(i=0; i < m; i++)
      for(j=0; j < n; j++)
        cin >> c[i][j];
}
```

用线性规划算法的成员函数 lpin()建立相应的初始单纯形表。

```
void LinearProgram::lpin(int mima, int mm, int nn, int *aa, int *bb, int **cc) {
   ifstream  inFile;
   int  i, j, k;
   double  value;
   minmax = mima;
   m = mm+nn, n = mm*nn;
   m1 = 0;
   m2 = m;
   m3 = 0;
   n1 = n;
   n2 = n;
   Make2DArray(a, m+2, n1+1);
   basic = new int[m+2];
   nonbasic = new int[n1+1];
   for(i=0; i <= m+1; i++)
      for(j=0; j <= n1; j++)
         a[i][j] = 0.0;
   for(j=0; j <= n1 ; j++)
      nonbasic[j] = j;
   for(i=1, j=n1+1; i <= m; i++, j++)
      basic[i] = j;
   for(i=m-m3+1, j=n+1; i <= m;i++, j++) {
      a[i][j] = -1.0;
      a[m+1][j] = -1.0;
   }
   for(k=0; k < mm; k++)
      for(i=0, j=k*nn; i < nn; i++)
         a[i+1][j+i+1] = 1;
   for(k=0; k < mm ; k++)
      for(i=0, j=k*nn; i < nn ;i++)
         a[nn+k+1][j+i+1] = 1;
   for(j=0; j < nn; j++)
      a[j+1][0] = bb[j];
   for(i=0; i < mm; i++)
      a[nn+i+1][0] = aa[i];
   for(i=0; i < mm; i++) {
      for(j=0; j < nn; j++) {
         int  jj = i*nn+j+1;
         a[0][jj] = minmax*cc[i][j];
       }
   }
   for(j=1; j <= n; j++) {
      for(i=m1+1, value=0.0; i <= m; i++)
```

```
            value += a[i][j];
        a[m+1][j] = value;
    }
}
```

算法的主函数调用单纯形算法求解。

```
int main() {
    read();
    LinearProgram X;
    X.lpin(-1, m, n, a, b, c);
    X.solve();
    X.lpin(1, m, n, a, b, c);
    X.solve();
    return 0;
}
```

本题显然也可以用最小费用流模型求解。

8-18 分配工作问题。

问题描述：有 n 件工作要分配给 n 个人做。第 i 个人做第 j 件工作产生的效益为 c_{ij}。试设计一个将 n 件工作分配给 n 个人做的最优和最差分配方案，使产生的总效益最大或最小。

算法设计：对于给定的 n 件工作和 n 个人，计算最优分配方案和最差分配方案。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行有 1 个正整数 n，表示有 n 件工作要分配给 n 个人做。接下来的 n 行中，每行有 n 个整数 c_{ij}（$1 \leqslant i \leqslant n$，$1 \leqslant j \leqslant n$），表示第 i 个人做第 j 件工作产生的效益为 c_{ij}。

结果输出：将计算的最小总效益和最大总效益输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
5	5
2 2 2 1 2	14
2 3 1 2 4	
2 0 1 1 1	
2 3 4 3 3	
3 2 1 2 1	

分析与解答：设变量 x_{ij} 表示将第 j 件工作分配给第 i 个人做。分配问题可以表述为如下的线性规划问题。

$$\max \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} x_{ij} = 1 \quad j = 1,2,\cdots,n$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,2,\cdots,n$$

$$x_{ij} \in \{0, 1\} \quad (i=1, 2, \cdots, m;\ j=1, 2, \cdots, n)$$

上述线性规划问题实际上是整数线性规划问题。但从约束条件可以看出，约束矩阵为全 1 模阵，因此 0-1 变量可以松弛为区间[0, 1]中的实数，用单纯形算法求解可得到 0-1 整数解。

进一步还可将变量 x_{ij} 松弛为所有非负实数。因此，上述问题转化为一个特殊的运输问题。

具体算法实现如下。

read()函数读入初始数据。

```
int  n, *a, *b, **c;
void read() {
   cin >> n;
   a=new int[n];
   b=new int[n];
   Make2DArray(c, n, n);
   for(int i=0; i < n; i++) {
      a[i]=1;
      b[i]=1;
   }
   for(i=0;i<n;i++)
      for(int j=0;j<n;j++)
         cin>>c[i][j];
}
```

用线性规划算法的成员函数 lpin()建立相应的初始单纯形表。算法的主函数调用单纯形法求解。

```
int main() {
   read();
   LinearProgram X;
   X.lpin(-1, n, n, a, b, c);
   X.solve();
   X.lpin(1, n, n, a, b, c);
   X.solve();
   return 0;
}
```

本题显然也可以用最小费用流模型求解。

8-19　负载平衡问题。

问题描述：G 公司有 n 个沿铁路运输线环形排列的仓库，每个仓库存储的货物数量不等。如何用最少搬运量可以使 n 个仓库的库存数量相同。搬运货物时，只能在相邻仓库之间搬运。

算法设计：对于给定的 n 个环形排列的仓库的库存量，计算使 n 个仓库的库存数量相同的最少搬运量。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行中有 1 个正整数 n，表示有 n 个仓库。第 2 行中有 n 个正整数，表示 n 个仓库的库存量。

结果输出：将计算的最少搬运量输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
5	11
17 9 14 16 4	

分析与解答：设每个仓库的货物数量为 x_i，最终要使 n 个仓库的库存数量相同，即每个

仓库的库存数量变成 $\sum_{i=1}^{n} x_i / n$，因此有些仓库要搬出货物，有些仓库要搬入货物。容易看出，本题实际上是算法实现题 8-17 运输问题的特殊情形，要搬出货物的仓库对应运输问题中的仓库，要搬入货物的仓库对应运输问题中的零售商店。从第 i 个仓库搬运单位数量货物到第 j 个仓库的费用为 $\min\{|j-i|, n-|j-i|\}$。

具体算法实现如下。

read()函数读入初始数据并转化为相应的运输问题。

```
bool read() {
   int  i, k=0, ii=0, jj=0, nn, *data, *ia, *ib;
   cin >> nn;
   data = new int[nn+1];
   for(i=1; i <= nn; i++) {
      cin >> data[i];
      k += data[i];
   }
   if(k%nn != 0){
      cout << "No Solution!" << endl;
      return false;
   }
   k /= nn;
   m = 0;
   n = 0;
   for(i=1; i <= nn; i++) {
      if(data[i]>k)
         m++;
      if(data[i]<k)
         n++;
   }
   a = new int[m];
   b = new int[n];
   ia = new int[m];
   ib = new int[n];
   Make2DArray(c, m, n);
   for(i=1; i <= nn; i++) {
      if(data[i] > k) {
         ia[ii] = i;
         a[ii++] = data[i]-k;
      }
      if(data[i] < k) {
         ib[jj] = i;
         b[jj++] = kdata[i];
      }
   }
   for(i=0; i < m; i++) {
      for(int j=0; j < n; j++) {
         c[i][j] = abs(ia[i]-ib[j]);
         if(nn-c[i][j] < c[i][j])
```

```
            c[i][j] = nn-c[i][j];
        }
    }
    return true;
}
```

用线性规划算法的成员函数 lpin()函数建立相应的初始单纯形表。算法的主函数调用单纯形算法求解。

```
int main() {
    if(!read())
        return 1;
    LinearProgram X;
    X.lpin(-1, m, n, a, b, c);
    X.solve();
    return 0;
}
```

本题也可以直接用最小费用流模型求解。

8-20 最长 k 可重区间集问题。

问题描述：给定实直线 L 上 n 个开区间组成的集合 I 和一个正整数 k，试设计一个算法，从开区间集合 I 中选取出开区间集合 $S \subseteq I$，使得在实直线 L 的任何一点 x，S 中包含点 x 的开区间个数不超过 k，且 $\sum_{z \in S}|z|$ 达到最大。这样的集合 S 被称为开区间集合 I 的最长 k 可重区间集。$\sum_{z \in S}|z|$ 称为最长 k 可重区间集的长度。

算法设计：对于给定的开区间集合 I 和正整数 k，计算开区间集合 I 的最长 k 可重区间集的长度。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行有 2 个正整数 n 和 k，分别表示开区间的个数和开区间的可重叠数。接下来的 n 行，每行有 2 个整数，表示开区间的左、右端点坐标。

结果输出：将计算的最长 k 可重区间集的长度输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
4 2	15
1 7	
6 8	
7 10	
9 13	

分析与解答：设给定的开区间集合共有 m 个不同的端点，将它们从小到大排序为 $x_1, x_2, \cdots, x_m$。构造网络 $G=(V, E)$如下。

① 每个端点 x_i 对应于网络中的一个顶点，新增源顶点 x_0 和汇顶点 x_{m+1}。

② 设给定的开区间集合中与顶点对 $x_i<x_j$ 相应的开区间数为 c_{ij}。每对顶点(x_i, x_j)之间有一条有向边，容量为 c_{ij}，费用为 x_j-x_i。

③ 每对顶点(x_i, x_{i+1})（$1 \leqslant i<m$），之间如果没有边，则增加一条有向边，容量为 k，费用为 0。

顶点对(x_0, x_1)和(x_m, x_{m+1})之间分别有一条有向边，容量为 k，费用为 0。

求网络 G 的最大费用流，最大费用流的总费用即为所求最长 k 可重区间集的长度。

具体算法用类 INTERV 实现如下。

```
class INTERV {
  int  m, n, k, f, s, t, **a, *b, **c;
  void read() {
    int  i, j;
    cin >> n >> k;
    Make2DArray(a, n+1, 2);
    b = new int[2*n];
    b[0] = 0;
    for(i=1; i <= n; i++) {
      cin >> a[i][0] >> a[i][1];
      if(a[i][0] > a[i][1])
        Swap(a[i][0], a[i][1]);
      b[2*i-2] = a[i][0];
      b[2*i-1] = a[i][1];
    }
    MergeSort(b, 2*n);
    m = dis(b, 2*n);
    Make2DArray(c, m, m);
    for(i=0; i < m; i++)
      for(j=0; j < m; j++)
        c[i][j] = 0;
    for(i=1; i <= n; i++) {
      int  p = BinarySearch(b, a[i][0], m+1), q=BinarySearch(b, a[i][1], m+1);
      c[p][q]++;
    }
  }
  void constructG(GRAPH<EDGE>&G) {
    int  i, j;
    s = 0;
    t = m+1;
    for(i=0; i < m; i++)
      for(j=i+1; j < m; j++)
        if(c[i][j])
          G.insert(new EDGE(i+1, j+1, c[i][j], b[i]b[j]));
    for(i=0; i < m-1; i++)
      if(c[i][i+1]==0)
        G.insert(new EDGE(i+1, i+2, k, 0));
    G.insert(new EDGE(0, 1, k, 0));
    G.insert(new EDGE(m, m+1, k, 0));
  }
  void output(GRAPH<EDGE>&G) {
    int  sum = 0;
    adjIterator<EDGE>A(G, s);
    for(EDGE*e=A.beg(); !A.end(); e=A.nxt())
      if(e->from(s)&&e->flow()>0)
```

```
            sum+=e->flow();
        if(sum<k)
            cout << "No Maxflow! " << sum << endl;
        sum=0;
        for(int i=1; i <= m; i++) {
            adjIterator<EDGE>A(G, i);
            for(EDGE*e=A.beg(); !A.end(); e=A.nxt())
                if(e->from(i) && e->flow()>0 && e->cost()<0)
                    sum -= e->cost()*e->flow();
        }
        cout << sum << endl;
    }
public:
    INTERV() {
        read();
        GRAPH<EDGE>G(m+1, 1);
        constructG(G);
        MINCOST<GRAPH<EDGE>, EDGE>(G, s, t, k);
        output(G);
    }
};
```

其中，constructG()函数构造相应网络，并将最大费用流问题转换为最小费用流问题，由MINCOST()函数求相应的最小费用流。

函数 dis()删去数组 b 中的重复端点。

```
int dis(int *b, int sz) {
    int  *e = new int[sz+1];
    for(int i=1, j=0; i <= sz; i++)
        e[i] = b[i];
    for(i=1,j=1; i <= sz; i++)
        if(e[i] != b[j1])
            b[j++] = e[i];
    delete []e;
    return j-1;
}
```

8-21　最长 k 可重线段集问题。

问题描述：给定平面 XOY 上 n 个开线段组成的集合 I 和一个正整数 k，试设计一个算法，从开线段集合 I 中选取出开线段集合 $S \subseteq I$，使得在 X 轴上的任何一点 p，S 中与直线 $x=p$ 相交的开线段个数不超过 k，且 $\sum_{z \in S}|z|$ 达到最大。这样的集合 S 称为开线段集合 I 的最长 k 可重线段集，$\sum_{z \in S}$ 称为最长 k 可重线段集的长度。

对于任何开线段 z，设其端点坐标为(x_0, y_0)和(x_1, y_1)，则开线段 z 的长度定义为

$$|z| = \left\lfloor \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \right\rfloor$$

算法设计：对于给定的开线段集合 I 和正整数 k，计算开线段集合 I 的最长 k 可重线段集的长度。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行有 2 个正整数 n 和 k，分别表示开线段的个数和开线段的可重叠数。接下来的 n 行，每行有 4 个整数，表示开线段的 2 个端点坐标。

结果输出：将计算的最长 k 可重线段集的长度输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
4 2	17
1 2 7 3	
6 5 8 3	
7 8 10 5	
9 6 13 9	

分析与解答：设给定的开线段集合共有 m 个不同的端点，将它们按照字典序从小到大排序为 $p_1, p_2, \cdots, p_m$。构造网络 $G=(V, E)$如下。

① 每个端点 p_i 对应于网络中的一个顶点，新增源顶点 p_0 和汇顶点 p_{m+1}。

② 设给定的开线段集合中与顶点对 $p_i<p_j$ 相应的开线段数为 c_{ij}。每对顶点(p_i, p_j)之间有一条有向边，容量为 c_{ij}，费用为$|(p_i, p_j)|$。

③ 每对顶点(p_i, p_{i+1})（$1\leqslant i<m$）之间如果没有边，则增加一条有向边，容量为 k，费用为 0。

顶点对(p_0, p_1)和(p_m, p_{m+1})之间分别有一条有向边，容量为 k，费用为 0。

求网络 G 的最大费用流，最大费用流的总费用即为所求最长 k 可重线段集的长度。

与算法实现题 8-21 最长 k 可重区间集问题不同的是，线段的端点是二维的，用类 ppair 表示二维端点如下。

```
class ppair{
public:
   bool operator<=(ppair a) const {
      if(x == a.x)
         return (y <= a.y);
      else
         return(x < a.x);
   }
   bool operator<(ppair a) const {
      if(x<a.x || (x == a.x && y < a.y))
         return true;
      return false;
   }
   bool operator>(ppair a) const {
      if(x>a.x || (x==a.x && y>a.y))
         return true;
      return false;
   }
   bool operator==(ppair a) const {
      return(x == a.x && y == a.y);
   }
   bool operator!=(ppair a) const {
      return !(*this == a);
```

```
  }
  int  x, y;
};
ostream& operator<<(ostream& ostr, const ppair& a) {
  return ostr << a.x << " " << a.y;
}
istream& operator>>(istream& istr, ppair& a) {
  istr >> a.x >> a.y;
  return istr;
}
```

具体算法用类 LINE 实现如下。

```
class LINE {
  int  m, n, k, f, s, t, **c;
  ppair  **a, *b;
  void read() {
    int  i, j;
    cin >> n >> k;
    Make2DArray(a, n+1, 2);
    b = new ppair[2*n];
    b[0].x = b[0].y = 0;
    for(i=1; i <= n; i++) {
      cin >> a[i][0] >> a[i][1];
      if(a[i][1] <= a[i][0])
        Swap(a[i][0], a[i][1]);
      b[2*i-2] = a[i][0];
      b[2*i-1] = a[i][1];
    }
    MergeSort(b, 2*n);
    m=dis(b, 2*n);
    Make2DArray(c, m, m);
    for(i=0; i < m; i++)
      for(j=0; j < m; j++)
        c[i][j] = 0;
      for(i=1; i <= n; i++) {
        int  p = BinarySearch(b, a[i][0], m+1), q = BinarySearch(b, a[i][1], m+1);
        c[p][q]++;
      }
    }
    void constructG(GRAPH<EDGE>&G) {
      int  i, j;
      s = 0;
      t = m+1;
      for(i=0; i < m; i++)
        for(j=i+1; j < m; j++)
          if(c[i][j])
            G.insert(new EDGE(i+1, j+1, c[i][j], len(b[j], b[i])));
      for(i=0;i < m-1; i++)
        if(c[i][i+1]==0)
```

```
            G.insert(new EDGE(i+1, i+2, k, 0));
         G.insert(new EDGE(0, 1, k, 0));
         G.insert(new EDGE(m, m+1, k, 0));
      }
      void output(GRAPH<EDGE>&G) {
         int sum=0;
         adjIterator<EDGE>A(G, s);
         for(EDGE*e=A.beg(); !A.end(); e=A.nxt())
            if(e->from(s) && e->flow()>0)
               sum += e->flow();
         if(sum < k)
            cout << "No Maxflow! " << sum << endl;
         sum = 0;
         for(int i=1; i <= m; i++) {
            adjIterator<EDGE>A(G, i);
            for(EDGE*e=A.beg(); !A.end(); e=A.nxt())
               if(e->from(i) && e->flow()>0 && e->cost()<0)
                  sum -= e->cost()*e->flow();
         }
         cout << sum << endl;
      }
   }
public:
   LINE() {
      read();
      GRAPH<EDGE>G(m+1, 1);
      constructG(G);
      MINCOST<GRAPH<EDGE>, EDGE>(G, s, t, k);
      output(G);
   }
};
```

其中，constructG()函数构造相应网络，并将最大费用流问题转换为最小费用流问题，由MINCOST()函数求相应的最小费用流。

dis()函数删去数组 b 中的重复端点。len()函数计算线段的长度。

```
int dis(ppair *b, int sz) {
   ppair  *e = new ppair[sz+1];
   for(int i=1, j=0; i <= sz; i++)
      e[i] = b[i];
   for(i=1, j=1; i <= sz; i++)
      if(e[i] != b[j1])
         b[j++] = e[i];
    delete []e;
    return j1;
}
int len(ppair u, ppair v) {
   double  z = sqrt((u.xv.x)*(u.xv.x) + (u.yv.y)*(u.yv.y));
   return int(z);
}
```

计算机算法设计与分析习题解答（第5版）

本书是与“十二五”普通高等教育本科国家级规划教材《计算机算法设计与分析（第5版）》配套的辅助教材和国家精品课程教材，分别对主教材中的算法分析题和算法实现题给出了解答或解题思路提示。为了提高学生灵活运用算法设计策略解决实际问题的能力，本书还将主教材中的许多习题改造成算法实现题，要求学生设计出求解算法并上机实现。本书教学资料包含各章算法实现题、测试数据和答案，可在华信教育资源网免费注册下载。

本书内容丰富，理论联系实际，可作为高等学校计算机科学与技术、软件工程、信息安全、信息与计算科学等专业本科生和研究生学习计算机算法设计的辅助教材，也是工程技术人员和自学者的参考书。

提升学生“知识—能力—素质”	体现“基础—技术—应用”内容
把握教学“难度—深度—强度”	提供“教材—教辅—课件”支持

相关图书：《计算机算法设计与分析（第5版）》 ISBN 978-7-121-34439-8

策划编辑：章海涛
责任编辑：章海涛
封面设计：张 昱

ISBN 978-7-121-34438-1

定价：56.00 元