算法设计与分析

实验指导书

温州大学

胡明晓　编

（2023 年 9 月）

实验 0 预备实验

实验目的：掌握时间函数的使用；掌握随机数的生成方法；掌握文本文件的读写方法；熟悉 C++类封装和 STL 编程技术。

实验内容和要求：

要求分别用日历时间函数和秒表时间函数获取系统时间戳和软件运行时长；要求使用 rand()、srand()函数生成伪随机数序列，并能做到多次运行同一程序产生的随机数序列都不相同；要求使用 C 语言的 FILE 指针、C++的文件流实现文本文件的读写。具体实验内容如下。

1. 日历时间

使用 C 函数读取日历时间，反复观察输出结果。

头文件：<time.h>

C 函数：time_t tm = time(NULL);

含义：自格林威治时间 1970 年 1 月 1 日零时以来的秒数，单位：秒。

2. 秒表时间

使用 C 函数读取秒表时间，反复观察输出结果。

头文件：<time.h>

C 函数：clock_t tm = clock();

含义：读取程序装载运行以来的时长，单位：毫秒。

3. 伪随机数生成

使用 C 函数生成一系列伪随机数。先用同一个随机数种子，运行多次，观察生成的伪随机数序列；再用日历时间作为种子，每隔一秒运行一次，观察生成的伪随机数序列。反复实验多次。

头文件：<stdlib.h>

C 函数：int a = rand();

含义：生成 0~32767（RAND_MAX）之间的一个伪随机数。

C 函数：srand(s);

含义：设置随机数种子 s。s 可取日历时间 time(NULL)。

4. 文本文件读写（C 语言）

使用 C 语言文件读写函数实现文本文件的写入、读取。对写操作检查文本文件的创建情况，打开文本文件观察内容的变化。反复实验多次。

头文件：<stdio.h>

4.1 利用 FILE 文件句柄

C 函数：FILE *fp = fopen("文件名", "r");

功能：打开文本文件，用于读。

C 函数：FILE *fp = fopen("文件名", "w+");

功能：打开文本文件，用于写。

C 函数：fclose(fp);

功能：关闭文件。

4.2 输入输出重定向

利用 freopen 函数可以将输入输出重定向到磁盘文件，优点是 scanf, printf, cin, cout 语句保持原来的写法，只需在开头部分加入如下两个 C 语言的 freopen 调用语句。

输入重定向语句：freopen("文件名", "r", stdin)

功能：将标准输入重定向，改为从指定文件读入。一般来说，输入内容很长，就有必要重定向。

输出重定向语句：freopen("文件名", "w", stdout)

功能：将标准输出重定向，改为保存到指定文件。如果输出文字很少，可以不写该语句，仍然输出到标准输出（显示器）。

5. 文本文件读写（C++语言）

使用 C++语言文件流实现文本文件的写入、读取。对写操作检查文本文件的创建情况，打开文本文件观察内容的变化。反复实验多次。

头文件：<fstream>

文件读入代码：

```
ifstream    fin("文件名");      // 创建文件输入流
fin >> a >> b .....          // 用>>操作从文件读入数据
fin.close();              // 使用后关闭
```

写到文件的代码：

```
ofstream    fout("文件名");    // 创建文件输出流
fout << a << b .....        // 用<<操作写数据到文件
fout.close();             // 使用后关闭
```

6. C++ STL

使用 C++ STL 的 queue、set、map 容器。使用 C++的排序算法、数值算法，所需头文件包括：

```
#include <queue>
#include <set>
#include <map>
#include <algorithm>
#include <numeric>
```

课外 OJ 题目：

HDU6433 Pow（C++类封装）

POJ1338 Ugly number（STL）

POJ2182 Lost Cows（STL）

实验 1 递归与分治（4 课时）

实验目的：掌握递归程序和分治法程序的设计步骤；能够用分治法解决非线性复杂度的计算机算法问题；能够区分不同子问题规模、个数和合并代码对分治法计算复杂性的影响；掌握算法执行时间的测定方法。

实验内容和要求：

要求用递归算法解决具有递推关系的序列和计数问题，用分治法高效解决具有非线性复杂度的实际问题，结合文件操作处理大规模数据输入的问题，用 clock_t tm=clock()测定算法执行时间，单位：毫秒。具体实验内容如下。

1. Catalan 数

问题描述：Catalan 数的定义如下，首先规定 C(0)为 1，然后按照下式定义 C(n)。

$$C(n)=\sum_{\substack{k_1\ge 0,k_2\ge 0\\ k_1+k_2=n-1}} C(k_1)C(k_2)=\sum_{k=0}^{n-1}C(k)C(n-1-k)\qquad (n\ge 1)$$

例如 n=5，$C(5)=C(0)C(4)+C(1)C(3)+C(2)C(2)+C(3)C(1)+C(4)C(0)$。

前 6 个 Catalan 数（C(1)~C(6)）是 1，2，5，14，42，132。要求计算各个 Catalan 数。

输入：n（n≤19）。

输出：C(n)。

输出（对比观察）：+运行时间(ms)。

样例输入	样例输出
6	132
19	1 767 263 190
35（仅对比方法）	3 116 285 494 907 301 262

实验方法 1：递归算法。用递归程序计算 C(n)，依次输入 n=1, 2, 3, ..., 20，输出 Catalan 数 C(n)，观察哪个 C(n)的计算时间开始大于 10 秒，记录能在 10 秒钟内计算出结果的最大 n 值、C(n)和计算时间。

按照上式设计的递归函数如下：

```
long long C(int n) {
    if (n == 0)
        return 1;
    long long sum(0);
    for (int i = 0; i < n; i++)
        sum += C(i) * C(n-1-i);
    return sum;
}
```

实验方法 2（对比方法）：非递归算法，为计算 C(n)，按顺序依次计算 C(1), C(2), ...，直至 C(n)。

用 n=18, ..., 35 测试其计算结果和计算时间。

2. 抽三计数

问题描述：n 个有区别的元素排成一行，编号依次为 1, 2, ..., n。对它的操作是：抽走奇数编号的元素或偶数编号的元素，然后将剩下的元素按原顺序重新编号。经过多次操作后，最后剩下 3 个，问有多少种可能的剩余结果？

输入：多个案例，每个案例输入一个整数 n（$n \leqslant 10^7$），直至输入结束（EOF）。

输出：每个案例的剩余结果种数。

输出（对比观察）：+运行时间(ms)。

样例输入	样例输出
10 2048 1536 10000000	2 0 512 1611392

实验方法 1：递归。设 n 长队列的结果有 A(n)种。只要 n 至少是 4，就可以二分，一半取上整数，另一半取下整数。得递推式如下：

$$A(n)=\begin{cases} A(\lfloor n/2 \rfloor)+A(\lceil n/2 \rceil) & (n \geq 4) \\ 1 & (n=3) \\ 0 & (n<3) \end{cases}$$

用递归函数实现。由于是一调二的递归，计算复杂性满足递推方程

$$T(n)=2T(n/2)+O(1)$$

解得 T(n)=O(n)。

优化：当 n 是偶数时，n/2 的上整数和下整数相等，只需一次递归调用，从而平均需 1.5 次调用，T(n)满足

$$T(n)=\frac{3}{2}T(n/2)+O(1)$$

计算复杂性可提高至 $T(n)=O(n^{\log 1.5})=O(n^{0.585})$。

实验方法 2：解析表达式。推出 A(n)的公式，计算复杂性 O(1)。

参考代码：

```
#include <iostream>
using namespace std;

int a(int n) {
    if (n <= 3)
        return (n < 3 ? 0 : 1);
    return a(n / 2) + a((n + 1) / 2);
}

int main() {
    int n;
    while (cin >> n)
```

```
        cout << a(n) << endl;
    return 0;
}
```

*3. 矩阵的幂

问题描述：对一个 t 阶方阵，要求计算 $\mathbf{A}^n \bmod p$，其中 n 是一个很大的整数，$n<=2\times10^9$。mod p 表示矩阵的每个元素对 p 取模。

输入：t（矩阵的阶，$t\leqslant100$），n，p，**A**（非负矩阵，按行，空隔输入）。

输出：$\mathbf{A}^n \bmod p$，分行空隔输出。

输出（对比观察）：+矩阵 **A**+指数 n+运行时间(ms)。

样例输入	样例输出
4 123456789 101 28 50 2 19 14 7 8 2 16 40 15 32 5 19 20 91	56 99 8 62 56 39 97 97 15 93 50 32 64 15 1 80
读取文件 QuickPower_in.txt	见文件 QuickPower_out.txt

实验方法：快速幂算法。先定义矩阵的乘法函数，再将结果矩阵 R 初始化为单位矩阵 I，矩阵的幂 M 初始化为 A，然后从低位向高位扫描 n 的二进制位，M 随扫描步骤不断平方（自乘），产生 A 的 2 的整数次幂的幂。每当扫描到 n 的二进制位 1，R 累乘以 M。如果为矩阵设计一个矩阵类，及其构造函数、乘法的运算符重载，可以用*实现矩阵的相乘。

参考代码：

```
#include <iostream>
using namespace std;

int t;// 矩阵的阶
int p;
int a[100][100], r[100][100], m[100][100];

void MatrixMultiply(int m1[][100], int m2[][100]) {
    // 矩阵乘法运算 M1×M2,  结果存回 M1
    int r[100][100];
    for (int i = 0; i < t; i++)
        for (int j = 0; j < t; j++) {
            r[i][j] = 0;
            for (int k = 0; k < t; k++)
                r[i][j] = (r[i][j] + m1[i][k] * m2[k][j]) % p;
        }
    for (int i = 0; i < t; i++)
        for (int j = 0; j < t; j++)
            m1[i][j] = r[i][j];
}
```

```
int main() {
    int n;
    cin >> t >> n >> p;
    for (int i = 0; i < t; i++)
        for (int j = 0; j < t; j++)
            cin >> a[i][j];
    for (int i = 0; i < t; i++)
        for (int j = 0; j < t; j++)
            r[i][j] = (i == j ? 1 : 0);   // r 是计算结果，初始化为单位矩阵
    if (n > 0) {
        for (int i = 0; i < t; i++)
            for (int j = 0; j < t; j++)
                m[i][j] = a[i][j];      // 保存 a 的 2 整数次幂的幂，初值 m=a
        while (n) {
            if (n & 1)
                MatrixMultiply(r, m);
            n >>= 1;
            MatrixMultiply(m, m);
        }
    }
    for (int i = 0; i < t; i++) {
        for (int j = 0; j < t; j++)
            cout << r[i][j] << ' ';
        cout << endl;
    }
    return 0;
}
```

*4. 大整数的乘法

问题：计算两个十进制大整数（位数达数千）的乘积。大整数用 char [10000]数组表示。

输入：两个大整数 a、b，十进制，最多 5000 位。

输出：a×b。

输出（对比观察）：+两个整数的位数+运行时间(ms)。

样例输入	样例输出
1024　10240	10485760
338849583746672139436839320467218152281583036860499304808492584055528117711658823406671259903148376558383270818131012258146392600439520994131344334162924536139	395058745832651445264197678006144819960207764603049364541393760515793556265294506836097278424682195350935443058704902519956553357102097992264849779494429556603
读取文件 BigInteger_in.txt	见文件 BigInteger_out.txt

实验方法：（参见教材 2.4）分治法，计算复杂性 $O(n^{1.59})$。

课外 OJ 题目：

POJ1664 放苹果 （递归**）
CFS559B EquivalentStrings （递归**）
POJ1505 Copying Books （二分***）
POJ2299 Ultra-QuickSort （**）
HDU1028 Ignatius and the Princess III （整数划分数问题*）
ZOJ2883 Shopaholic
ZOJ3129 Japan
ZOJ2107 Quoit Design （最接近点对***）

实验 2 动态规划（6 课时）

实验目的：掌握动态规划的阶段、状态、状态转移方程概念及动态规划设计步骤。能够选择适当的状态和状态指标，并构造出状态转移方程，最终能用动态规划算法高效解决具有最优子结构性质和子问题重叠性质的算法问题。

实验内容和要求：

要求识别出问题的最优子结构性质和重叠子问题特点，做适当的阶段划分，选择适当的状态指标并推导出状态转移方程的递推公式，然后编程实现动态规划算法，在很短时间内计算出大规模问题的结果。

1. 数字三角形问题（教材：算法实现题 3-4）

OJ 问题：Triangle（参见 http://poj.org/problem?id=1163，及校 ACM 训练题 WOJ1117）

题意：如下图，在数字三角形上寻找一条沿相邻顶点从顶到底走的路径，使路径上的数字和最大。

```
          7
        3   8
      8   1   0
    2   7   4   4
  4   5   2   6   5
```

输入：三角形高度 n，数字三角形数值 a_{ij}，a_{ij} 表示第 i 行第 j 个元素。i=1, …, n，j=1, …, i。

输出：最大数字和。

输出（对比观察）：+n+运行时间(ms)。

样例输入	样例输出
5 7 3 8 8 1 0 2 7 4 4 4 5 2 6 5	30

实验方法 1：动态规划，开设二维数组 L[n][n]，L[i][j]表示 a[1][1]至 a[i][j]的最大路径和。状态、评价值是：

状态(i,j)：考察从顶 a_{11} 到 a_{ij} 的路径。

评价值 L(i,j)：从顶 a_{11} 到 a_{ij} 的路径最大和。

状态转移方程：

$$L(i,j)=\begin{cases}\max\{L(i-1,j),L(i-1,j-1)\}+a_{ij} & (2\le j<i)\\ L(i-1,j)+a_{ij} & (j=1)\\ L(i-1,j-1)+a_{ij} & (j=i)\end{cases} \qquad (*)$$

初值：$L(1,1)=a_{11}$

计算顺序：i 从 1 到 n，j 从 1 到 i。

问题所求：$\max\limits_{1\le j\le n} L(n, j)$

实验方法 2：动态规划。d[i][j]表示在以 a[i][j]为顶的子三角形中的最大路径和。状态、评价值是：

状态(i,j)：考察以 a_{ij} 为顶到达底部的路径。显然，所有路径都限于以 a_{ij} 为顶的子三角形。

评价值 d(i,j)：以 a_{ij} 为顶到达底部的路径最大和。

状态转移方程：

$$d(i, j) = \max\{d(i+1, j), d(i+1, j+1)\} + a_{ij} \quad (1 \le j \le i) \qquad (**)$$

初值：$d(n,i)=a_{ni}$ (i=1, …, n)

计算顺序：d(n,1), d(n,2), ..., d(n,n)初值不用算，然后计算 d(n-1,1), ..., d(n-1,n-1)，再计算 d(n-2,1), ..., d(n-2,n-2)，……，最后计算 d(1,1)。

问题所求：d(1,1)。

实验方法 3（对比方法）：备忘录方法。开设二维数组 d[n][n]，初始化为-1。设计递归函数 int memoir(int i, int j)，计算 a[i][j]为顶的子三角形中的最大路径和后返回，计算时若 d[i][j]为-1，按(**)式递归计算出 d[i][j]后返回，否则直接返回 d[i][j]。主函数调用 memoir(1,1)，得到所求最大和。

参考代码：

```
#include <iostream>
using namespace std;

int n, a[110][110], d[110][110];

int memoir(int i, int j) {  // 备忘录方法
    if (d[i][j] >= 0)
        return d[i][j];
    if (i == n)
        return d[i][j] = a[i][j];
    return d[i][j] = max(memoir(i + 1, j), memoir(i + 1, j + 1)) + a[i][j];
}

int main() {
    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++) {
            cin >> a[i][j];
            d[i][j] = -1;
        }
    memoir(1, 1);
    cout << d[1][1] << endl;
    return 0;
}
```

2. 乡村邮局问题

OJ 问题：Post Office（参见 http://poj.org/problem?id=1160）

题意：在线性的乡村公路上设置若干邮局，使各乡村到最近邮局的总距离最小。

输入：乡村数 V，拟设邮局数 P。然后是 V 个乡村在直线上的坐标。

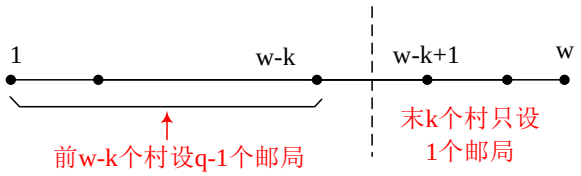
输出：最小总距离。

样例输入	样例输出
10 5 1 2 3 6 7 9 11 22 44 50	9

实验方法：动态规划。设置状态(w,q)：表示考虑前 w 个村，限设 q 个邮局的子问题。评价值 F(w,q)：表示前 w 个村限设 q 个邮局的最小总距离。按照最后一个邮局的服务乡村数 k 分类，则有状态转移方程

$$F(w,q)=\min_{1\le k\le w-q+1}\{F(w-k,q-1)+G(w-k+1,w)\} \qquad (w>q)$$

其中 G(x,y)为在 x~y 村仅设一个邮局的最小总距离。



初始化：F(w,w)=0，F(w,1)=G(1,w)。DP 计算顺序可以先按前缀村数 w 从 1 到 V 循环，再按邮局数 q 从 2 到 P 循环，或者先按邮局数 q 从 2 到 P 循环，再按前缀村数 w 从 1 到 V 循环。最终所求是 F(V, P)。

时间复杂性：$O(V^2(P+V))=O(n^3)$。空间复杂性 $O(VP)=O(n^2)$，利用滚动数组可以优化至 O(n)。

参考代码：

```
#include <iostream>
#include <algorithm>
using namespace std;

int V, P, X[301];     // 乡村坐标

int GetOnePost(int i1, int i2) {
    //乡村 i1~i2 间只设一个邮局的最小总距离
    int mid = (i1 + i2) / 2;    // 选中位点必定使总距离最小
    int result = 0;
    for (int i = i1; i <= i2; i++)
        result += abs(X[i]-X[mid]);
    return result;
}
```

```
int DP() {
    int dp[301][40];

    for (int i = 1; i <= V; i++)
        dp[i][1] = GetOnePost(1, i);
    // 换向计算，先按邮局数循环，空间未优化
    for (int j = 2; j <= P; j++) {    // 邮局数
        dp[j][j] = 0;    // 村数等于邮局数
        for (int i = j+1; i <= V; i++) { // 村数
            dp[i][j] = 0x7fffffff;
            for (int k = 1; k <= i-j+1; k++) {    // 最后一个邮局的服务村数
                int t = dp[i-k][j-1] + GetOnePost(i-k+1, i);
                if (t < dp[i][j])
                    dp[i][j] = t;
            }
        }
    }
    return dp[V][P];
}

int main() {
    cin >> V >> P;
    for (int i = 1; i <= V; i++)
        cin >> X[i];
    cout << DP() << endl;
    return 0;
}
```

3. 阿尔法码

OJ 问题：Alphacode（参见 http://poj.org/problem?id=2033）

题意：A 对应 1，B 对应 2，Z 对应 26。给定一个数字序列（非 0 开头），求对应到该字母序号序列（Alphacode）的编码数量。序列长度不超过 5000。

输入：字母序号序列。输入 0 结束处理。

输出：Alphacode 种数。

样例输入	样例输出
25114 1111111111 3333333333 0	6 89 1

实验方法：动态规划。设状态 k 表示考察 k-前缀（k=0,1,…,n）。评价值 C(k)：k-前缀的译码种数。则有状态转移方程

$$C(k)=('1'\le x_k\le'9')?C(k-1):0$$
$$+("10"\le x_{k-1}x_k\le"26")?C(k-2):0 \qquad (k\ge 2)$$

初始化：C(1)=1，C(0)=1。最终所求是 C(n)。

*4. 二维最大子段和问题

OJ 问题：To the Max（参见 http://poj.org/problem?id=1050）

题意：求二维最大子段和。

实验方法：（参见教材 3.4）动态规划。对第一维枚举，对第二维动态规划。

*5. m 处理器问题

OJ 问题：*m* processors（参见 http://acm.fzu.edu.cn/problem.php?pid=1442）

问题描述（教材：算法实现题 3-25）：在一个网络通信系统中，要将 n 个数据包依次分配给 m 个处理器进行数据处理，并要求处理器负载尽可能均衡。

设给定的数据包序列为$\{\sigma_0,\sigma_1,\cdots,\sigma_{n-1}\}$。

m 处理器问题要求的是 $r_0=0\leqslant r_1\leqslant\cdots\cdots\leqslant r_{m-1}\leqslant n=r_m$，将数据包序列划分为 m 段：$\{\sigma_0,\sigma_1,\cdots,\sigma_{r_1-1}\}$，$\{\sigma_{r_1},\cdots,\sigma_{r_2-1}\}$，…，$\{\sigma_{r_{m-1}},\cdots,\sigma_{n-1}\}$，使 $\max_{i=0}^{m-1}\{f(r_i,r_{i+1})\}$达到最小。式中，$f(i,j)=\sqrt{\sigma_i^2+\cdots+\sigma_j^2}$ 是序列$\{\sigma_i,\cdots,\sigma_j\}$的负载量。

$\max_{i=0}^{m-1}\{f(r_i,r_{i+1})\}$的最小值称为数据包序列$\{\sigma_0,\sigma_1,\cdots,\sigma_{n-1}\}$的均衡负载量。

算法设计：对于给定的数据包序列$\{\sigma_0,\sigma_1,\cdots,\sigma_{n-1}\}$，计算 m 个处理器的均衡负载量。

输入：第 1 行有 2 个正整数 n 和 m。n 表示数据包个数，m 表示处理器数。接下来的 1 行中有 n 个整数，表示 n 个数据包的大小。

输出：处理器均衡负载量，保留 2 位小数。

样例输入	样例输出
6 3 2 2 12 3 6 11	12.32

实验方法：二维 DP。令 dp[i][j]表示前 i 个数据包分给 j 个处理器的最优负载量。状态转移方程：

$$dp[i][j]=\min_{j-1\le k<i}\{\max(dp[k][j-1],f(k+1,i))\}$$

最终所求：dp[n][m]。

课外 OJ 题目：

CFS161D Distance in Tree （树形 DP**）

HDU3480 Division （四边形不等式加速**）

POJ1037 A decorative fence （双值 DP+区块链计数***）

HDU3555 Bomb （数位 DP***）

ZOJ4027 Sequence Swapping （**）

实验 3 贪心算法（4 课时）

实验目的：掌握贪心算法的解题步骤。能够判断问题是否具有贪心选择性质，识别贪心算法适用的指标量，并能用贪心算法解决具有最优子结构性质和贪心选择性质的计算机算法问题。

实验内容和要求：

要求判断出问题的贪心选择性质，识别出贪心算法适用的指标量，进一步设计关于该指标量的贪心算法，并编写贪心算法的代码，在很短时间内计算出大规模问题的结果。

1. 活动安排问题

OJ 问题：活动安排（参见 https://nanti.jisuanke.com/t/A2260）。仅要求输出最大活动数。

题意：给定 n 个活动的开始时间 s_i 和结束时间 f_i，每个活动的占用时间是$[s_i,f_i)$。要求在给定的活动中选出个数最多的相容活动子集合，所谓相容是指任意两个活动的时间没有重叠。

输入：n，s_1，f_1，……，s_n，f_n。

输出：c（最多活动个数）。

输出（对比观察）：+每个活动的选中标志 b[1..n]（1 表示安排 0 表示不安排）+运行时间(ms)。

样例输入	样例输出
11 0 6 1 4 2 13 3 5 3 8 5 7 5 9 6 10 8 11 8 12 12 14	4 0 1 0 0 0 1 0 0 1 0 1（答案不唯一）

实验方法：（参见教材 4.1）贪心算法，按活动结束时间从早到晚贪心扫描。

参考代码：

```
#include <iostream>
#include <algorithm>
using namespace std;

struct Program {     // 节目类
    int ts, te;
```

```
};
bool operator < (Program a, Program b) {    // 运算符重载
    return a.te < b.te;
}

int main() {
    int n, c;
    Program p[100];
    do {
        cin >> n;
        if (n == 0)
            break;
        for (int i = 0; i < n; i++)
            cin >> p[i].ts >> p[i].te;
        sort(p, p+n);
        c = 1;
        int j = 0;
        for (int i = 1; i < n; i++) {
            if (p[i].ts >= p[j].te) {
                c++;
                j = i;
            }
        }
        cout << c << endl;
    } while(1);
    return 0;
}
```

2. 有向图单源最短路径

OJ 问题：tgtsttg 去旅行（参见 https://nanti.jisuanke.com/t/A1115）

题意：给定一个带权有向图 G=(V, E)，其中每条边的权是非负实数。另外，还给定 V 中的一个顶点，称为源。要求计算源到其他各顶点的最短路径长度。路的长度是指路上各边的权值之和。

输入：n（顶点数），e[1..n][1..n]（权值矩阵，-1 表示无边）。v_1 为源。

输出：d[1..n]（v_1 到各个顶点的最短路径长度）。

输出（对比观察）：+最短路径（用顶点序号表示，共 n 条）+运行时间(ms)。

样例输入	样例输出
5 0 10 -1 30 100 -1 0 50 -1 -1 -1 -1 0 20 10 -1 -1 20 0 60 -1 -1 -1 -1 0	0 10 50 30 60 1 1 2 1 4 3 1 4 1 4 3 5

5 0 50 30 100 10 50 0 5 20 -1 30 5 0 50 -1 100 20 50 0 10 10 -1 -1 10 0	0 35 30 20 10 1 1 3 2 1 3 1 5 4 1 5

实验方法：Dijkstra 贪心算法，按最短特殊路径长度最小者贪心。

*3. Buildings

OJ 问题：Buildings（参见 http://acm.hdu.edu.cn/showproblem.php?pid=4296）

题意：有 n 个预制楼层模块，编号 1~n，建筑师通过堆叠成为一座 n 层的大楼。第 i 个模块的重量和强度分别是(w_i,s_i)，问按什么顺序堆叠这些模块能使大楼的最大 PDV_i 值达到最小？其中 PDV_i 定义为 $PDV_i = \sum_{j=1}^{i-1} w_{k_j} - s_{k_i}$，从顶到底的模块编号是 $k_1, k_2, ..., k_n$。

*4. Problem Buyer

OJ 问题：Problem Buyer（参见 http://acm.hdu.edu.cn/showproblem.php?pid=6003）

题意：有一个题目集，题目总数为 n，每道题有难度范围$[A_i, B_i]$，问至少要取多少道题（随机），才能保证覆盖 m 种指定的难度？

课外 OJ 题目：

POJ2209 The King （*）

ZOJ2883 Shopaholic （*）

HDU2570 迷瘴 （*）

POJ2054 Color a Tree （***）

POJ3614 Sunscreen （**）

实验 4 回溯法（4 课时）

实验目的：掌握回溯法的基本解题思路和剪枝技术。能够用深度优先搜索算法解决巨大解空间的搜索问题。

实验内容和要求：

要求识别出问题的解空间，并编写一个递归程序，在解空间树进行深度优先搜索，遍历解空间树，适当剪枝，搜索期间用一个全局 T 型变量不断更新当前的最优值或计数。

1. 最优装载问题

OJ 问题：（参见校 ACM 训练题 WOJ1130）

问题（一种特殊 0-1 背包问题）：给定 n 种物品和一个背包。物品 i 的重量是 w_i，背包的容量为 c。问应如何选择装入背包中的物品，使得装入背包中物品的总重量最大？

在选择装入背包的物品时，对每种物品 i 只有两种选择，即装入背包或不装入背包。不能将物品 i 装入背包多次，也不能只装入部分物品 i。

输入：物品数 n，装载限重 c，物品重量 $w[1…n]$。n≤30。

输出：最优装载重量 wmax。

输出（对比观察）：+最优装载方案 b[1…n]（b[i]=1 表示装，b[i]=0 表示不装）+运行时间(ms)。

样例输入	样例输出
5 50 20 15 31 11 23	49 0 1 0 1 1
30 800 39 50 58 74 58 32 65 14 45 26 41 97 67 80 66 74 48 19 9 10 20 77 64 60 30 49 36 45 41 39	800 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1（答案不唯一）

实验方法 1：（参见教材 5.2）回溯法。使用约束剪枝和限界剪枝。

实验方法 2（对比方法）：暴力搜索。以 n 位二进制数表示一种装载选择序列，然后循环 2^n 次，枚举出最优值和最优解。

参考代码：

```
#include <stdio.h>
#include <time.h>
#include <numeric>
using namespace std;

int cap, n, a[40];     // D 型数据，cap 最大装载限量，n 物品数(0 开始)，a 物品重量
int maxLoad;           // T 型数据，最大装载量（问题所求）
int curLoad;           // C 型数据，当前装载量
int cr;
int x[40], bestx[40];
```

```
void dfs(int t) {
    // 接口：计算第 0~t-1 件物品已决策完毕、第 t 件物品取遍各种决策的最优结果
    if (t == n) {           // 递归出口
        if (curLoad <= cap && curLoad > maxLoad) {
            maxLoad = curLoad;     // 收集或更新结果数据(T 型)
            for (int i = 0; i < n; i++)
                bestx[i] = x[i];
        }
        return;
    }
    // a[t]不装
    cr -= a[t];
    x[t] = 0;
    if (curLoad + cr > maxLoad)  // 剪枝
        dfs(t+1);
    // a[t]装
    curLoad += a[t];     // 更新当前数据(C 型)
    x[t] = 1;
    if (curLoad <= cap)// 剪枝
        dfs(t+1);
    curLoad -= a[t];     // 恢复当前数据(C 型)
    cr += a[t];
}

int main() {
    clock_t tm1, tm2;
    scanf("%d%d", &n, &cap);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    maxLoad = -1;// T 型初始化
    curLoad = 0;   // C 型初始化
    cr = accumulate(a, a + n, 0);
    tm1 = clock();
    dfs(0);
    tm2 = clock();
    printf("%d\n", maxLoad);
    for (int i = 0; i < n; i++)
        printf("%d%c", bestx[i], i == n - 1 ? '\n' : ' ');
    printf("%dms\n", tm2 - tm1);
    return 0;
}
```

2. 符号三角形问题

OJ 问题：符号三角形（参见 http://acm.hdu.edu.cn/showproblem.php?pid=2510，及校 ACM 训练题 WOJ1003）

题意：符号三角形的第一行有 n 个符号“＋”或“－”。从第二行开始，符号数逐行递减，在 2 个同号下面的都是“＋”号，2 个异号下面都是“－”号（如图）。要求对于给定的 n，计算有多少种不同的符号三角形，使其所含的“＋”和“－”的个数相同。

－ ＋ ＋ － ＋
 － ＋ － －
  － － ＋
   ＋ －
    －

输入：

第一行符号数 n（1≤n≤24）。

输出：“＋”和“－”的个数相同的符号三角形数。

输出（对比观察）：+n+运行时间(ms)。

样例输入	样例输出
3	4
12	410
24	822229

实验方法：（参见教材 5.4）回溯法。解空间树是子集树。结合离线打表法，先用程序 1 计算出所有 n=1, 2, ..., 24 的答案，以数组初值的格式存储答案，再用程序 2 直接读取答案。

*3. 最大团问题

OJ 问题：Maximum Clique （参见 http://acm.hdu.edu.cn/showproblem.php?pid=1530，及校 ACM 训练题 WOJ1084）

题意：给定一个无向图 G=(V, E)，求 G 的最大团，即点数最多的团。团是指一种完全子图，且不包含在更大的完全子图中。

输入：顶点数 n，邻接矩阵（n 行 n 列）。n≤30。

输出：最大团顶点数。

输出（对比观察）：+最大团的顶点 b[1…n]（1 表示在团中 0 表示不在团中）+运行时间(ms)。

样例输入	样例输出
5 0 1 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 1 1 1 1 0	3 1 0 0 1 1（答案不唯一）
5 0 1 1 0 1 1 0 1 1 1	4 1 1 1 0 1（答案不唯一）

1 1 0 1 1 0 1 1 0 1 1 1 1 1 0	
7 0 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 0 0 1 1 0 1 1 0 0 0 0 1 0 0 1 0 0	4 1 0 1 1 0 1 0（答案不唯一）

实验方法 1：（参见教材 5.7）回溯法。使用约束剪枝和限界剪枝。

实验方法 2（对比方法）：暴力搜索。以 n 位二进制数表示一种装载选择序列，然后循环 2^n 次，枚举出最优值和最优解。

参考代码：

```
#include <iostream>
#include <queue>
using namespace std;
#include <stdio.h>

#define N 50
int n;
int a[N][N];
int bestn, bestx[N];
int cn, x[N];

bool OK(int *x, int t) {
    int ok = true;
    for (int i = 1; i < t; i++)
        if (x[i] && a[i][t] == 0) {
            ok = false;
            break;
        }
    return ok;
}

void dfs(int t) {
    if (t > n) {
        if (cn > bestn) {
            bestn = cn;
            for (int i = 1; i <= n; i++)
                bestx[i] = x[i];
        }
```

```
            return;
        }
        if (OK(x, t)) {
            x[t] = 1;
            cn++;
            dfs(t + 1);
            cn--;
        }
        if (cn + (n - t) > bestn) {
            x[t] = 0;
            dfs(t + 1);
        }
}
int main() {
        do {
            cin >> n;
            if (n == 0)
                break;
            for (int i = 1; i <= n; i++)
                for (int j = 1; j <= n; j++) {
                    cin >> a[i][j];
                    if (i > j && a[i][j] != a[j][i])
                        a[i][j] = a[j][i] = 1;
                }
            bestn = 0;
            cn = 0;
            dfs(1);
            cout << bestn << endl;
            for (int i = 1; i <= n; i++)
                cout << (bestx[i] ? 1 : 0) << (i == n ? '\n' : ' ');
        } while(1);
        return 0;
}
```

*4. 旅行售货员问题

OJ 问题：（参见校 ACM 训练题 WOJ1069）

问题：旅行售货员问题（参见教材 5.9）。

题意：求最短哈密尔顿回路。

实验方法：（参见教材 5.9）排列树的深搜。

课外 OJ 题目：

HDU2307 Necklace Decomposition （*）

HDU2614 Beat （**）

POJ1980 Unit Fraction Partition （**）

POJ1011 Sticks （***）

实验 5 分支限界法（6 课时）

实验目的：掌握分支限界法的基本解题思路和剪枝技术。能够用广度优先搜索算法配合适当的搜索路径表记录技术和剪枝技术解决巨大解空间的搜索问题。

实验内容和要求：

要求编写一个基于队列或优先队列数据结构的程序，在解空间树进行广度优先搜索，充分利用控制剪枝、限界剪枝和约束剪枝，使用状态变量或其它路径记录技术跟踪队列中的搜索路径，搜索期间用一个全局 T 型变量不断更新当前最优值。

1. Red and Black（参见 http://acm.hdu.edu.cn/showproblem.php?pid=1312）

题意：给定一个格点矩形，有些是红格，有些是黑格，红格为禁入点。有一个出发格点@，只能向上下左右相邻格走。问：从@出发能到达的格点个数是多少？

输入：矩形宽 w，高 h，地图（# 红格，. 黑格，@ 出发点）。当 w=h=0 时结束处理。

输出：能到达的格点总数。

样例输入	样例输出
6 9 ....#. .....# .... #@...# .#..#. 0 0	45

实验方法：广搜。在一棵 4 叉树的解空间树上进行队列式广度优先搜索，解空间树的结点信息为二维坐标。开辟目标型的访问标记数组 visited[N][N]和计数 cnt，搜索过程不断 cnt++。剪枝：出界、禁入点、访问过。

参考代码：

```
#include <iostream>
#include <queue>
using namespace std;
#include <memory.h>

int w, h;
char map[21][21];
int vis[21][21];
int cnt;
```

```
int cx, cy;
int moves[4][2] = {{-1,0}, {1,0}, {0,-1}, {0,1} }; //方向数组
struct Point {   //类封装
    int x, y;
    Point(int x0 = 0, int y0 = 0) : x(x0), y(y0) { }
};

int main() {
    queue<Point> q;
    do {
        cin >> w >> h;
        if (w == 0 && h == 0)
            break;
        for (int i = 1; i <= h; i++)
            for (int j = 1; j <= w; j++) {
                cin >> map[i][j];
                if (map[i][j] == '@') {
                    cx = j;
                    cy = i;
                }
            }
        memset(vis, 0, sizeof(vis));
        q.push(Point(cx, cy));
        vis[cy][cx] = 1;
        cnt = 1;
        while (!q.empty()) {
            Point p = q.front();
            q.pop();
            for (int i = 0; i < 4; i++) {
                Point np(p.x+moves[i][0], p.y + moves[i][1]);
                if (np.x >= 1 && np.x <= w && np.y >= 1 && np.y <= h
                  && !vis[np.y][np.x] && map[np.y][np.x] == '.') {
                    q.push(np);
                    vis[np.y][np.x] = 1;
                    cnt++;
                }
            }
        }
        cout << cnt << endl;
    } while(1);
    return 0;
}
```

2. 最大团问题

OJ 问题：Maximum Clique （参见 http://acm.hdu.edu.cn/showproblem.php?pid=1530，教材 6.6）

题意：给定一个无向图 G=(V, E)，求 G 的最大团，即点数最多的团。团是指一种完全子图，且不包含在更大的完全子图中。

输入：顶点数 n，邻接矩阵（n 行 n 列）。n≤50。

输出：最大团顶点数。

输出（对比观察）：+最大团的顶点 b[1…n]（1 表示在团中 0 表示不在团中）+运行时间(ms)。

样例输入	样例输出
5 0 1 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 1 1 1 1 0	3 1 0 0 1 1

实验方法：（参见教材 6.6）优先队列式分支限界法。解空间树类型：子集树。结点数上界 un=cn+n-level+1 既作上界函数又作优先函数。

参考代码：

```
// 在 HDU1530 中会 TLE
#include <iostream>
#include <queue>
using namespace std;

#define N 50
int n;
int a[N+2][N+2];

int bestn;
int bestx;

struct Node {
    int level;
    int cn;
    char x[N+2];
    Node(int le0 = 0, int cn0 = 0) : level(le0), cn(cn0) { }
    bool operator < (const Node &b) const {
        return cn + n - level < b.cn + n - b.level;      // 顶点数上限越大越优先
    }
};

void bfs() {
    priority_queue<Node> q;
```

```
        q.push(Node(0, 0));
        while (!q.empty()) {
            Node qn = q.top();
            q.pop();
            if (qn.level == n) { // 叶结点
                if (qn.cn > bestn)
                    bestn = qn.cn;
                break;
            }
            int i = qn.level + 1;
            bool OK = true;
            for (int j = 1; j < i; j++)
                if (qn.x[j] == 1 && a[i][j] == 0) {
                    OK = false;
                    break;
                }
            Node newqn = qn;
            newqn.level = qn.level + 1;
            if (OK) { // 当前点符合完全图条件
                newqn.cn = qn.cn + 1;
                newqn.x[newqn.level] = 1;
                q.push(newqn);
                if (newqn.cn > bestn)
                    bestn = newqn.cn;
            }
            newqn.cn = qn.cn;
            newqn.x[newqn.level] = 0;
            if (newqn.cn + n - newqn.level > bestn)
                q.push(newqn);
        }
    }

    int main() {
        do {
            cin >> n;
            if (n == 0)
                break;
            for (int i = 1; i <= n; i++)
                for (int j = 1; j <= n; j++)
                    cin >> a[i][j];
            bestn = 0;
            bfs();
            cout << bestn << endl;
        } while (1);
```

```
    return 0;
}
```

*3. 单源最短路径问题

问题：MPI Maelstorm（参见 http://poj.org/problem?id=1502）

题意：给定一个无向图，边有非负权值。求固定源 s 到各个点的最短路径长度的最大值。顶点数 n≤100。

输入：顶点数 n，权值邻接矩阵的严格下三角部分。

输出：源 s 到各个点的最短路径长度的最大值。

输出（对比观察）：+源 s 到各个点的最短路径长度 L[1..n]+运行时间(ms)。

样例输入	样例输出
5 50 30 5 100 20 50 10 x x 10	35

实验方法 1：队列式分支限界法。解空间树：路径树。解空间树的节点信息有：编号 vi、最短长度 len，vi 表示无向图的结点编号，len 表示当前已知的从源 s 到 vi 的所有路径的最短长度。开辟一个 dist[1~n]的数组，记录当前 len[vi]，并利用 dist 进行控制剪枝。

实验方法 2（对比方法）：优先队列式分支限界法。解空间树、节点信息和剪枝同方法 1。优先函数：当前最短路径长度，越小越优先。

参考代码：

```
#include <iostream>
#include <queue>
#include <algorithm>
#include <iterator>
using namespace std;
#include <stdlib.h>

#define INF 0x7fffffff
int n;
int A[110][110];
int dist[110];
struct Node {
    int vi;
    int length;
    Node(int vi0 = 0, int len0 = 0) : vi(vi0), length(len0) {    }
    bool operator < (const Node &b) const  {
        return length > b.length;        // length 越小越优先
    }
};
```

```
void bfs() {
    priority_queue<Node> Q;
    for (int i = 1; i <= n; i++)
        dist[i] = INF;
    dist[1] = 0;
    Q.push(Node(1, 0));
    do {
        Node E = Q.top();
        Q.pop();
        for (int j = 1; j <= n; j++) {
            if (j != E.vi && A[E.vi][j] != INF && E.length + A[E.vi][j] < dist[j]) {
                dist[j] = E.length + A[E.vi][j];
                Q.push(Node(j, dist[j]));
            }
        }
    } while (!Q.empty());
}

int main() {
    char s[10];
    cin >> n;
    for (int i = 1; i <= n; i++)
        A[i][i] = 0;
    for (int i = 2; i <= n; i++) {
        for (int j = 1; j < i; j++) {
            cin >> s;
            A[i][j] = A[j][i] = (s[0] == 'x' ? INF : atoi(s));
        }
    }
    bfs();
    cout << *max_element(dist+2, dist+n+1) << endl;
    return 0;
}
```

*4. 最优装载问题

问题：（一种特殊 0-1 背包问题）给定 n 种物品和一个背包。物品 i 的重量是 w_i，背包的容量为 c。问应如何选择装入背包中的物品，使得装入背包中物品的总重量最大？

在选择装入背包的物品时，对每种物品 i 只有两种选择，即装入背包或不装入背包。不能将物品 i 装入背包多次，也不能只装入部分物品 i。

输入：物品数 n，装载限重 c，物品重量 $w[1…n]$。n≤24。

输出：最优装载重量 wmax。

输出（对比观察）：+最优装载方案 b[1..n]（0 表示不装 1 表示装）+运行时间(ms)。

样例输入	样例输出
5 50 20 15 31 11 23	49
10 100 54 9 20 59 50 33 40 56 92 89	99
24 800 39 50 58 74 58 32 65 14 45 26 41 97 67 80 66 74 48 19 9 10 20 77 64 60	800

实验方法：优先队列式分支限界法。解空间树：子集树。约束函数：当前部分重量和不能超限。限界函数兼优先函数：装载重量上界 ub = cw + cr，cw 为当前重量，cr 为当前余重，ub 越大越优先。算法在第一扩展叶结点时即可结束。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;

#define N 40
int bestw;
int c, n;
int w[N+2];

struct Node {
    int level;
    int cw;
    int cr;
    Node(int le0 = 0, int w0 = 0, int r0 = 0) : level(le0), cw(w0), cr(r0) { }
    bool operator < (const Node &b) const  // ub = cw+cr  越大越优先
    {   return cw + cr < b.cw + b.cr;  }
};
priority_queue<Node> q;

void bfs() {
    int r = accumulate(w + 1, w + n + 1, 0);
    q.push(Node(0, 0, r));
    while (!q.empty()) {
        Node cn = q.top();
        q.pop();
        int t = cn.level;
        if (t == n) {    // 要扩展叶结点
            if (cn.cw > bestw)
                bestw = cn.cw;
            break;
        }
```

```
            t++;
            if (cn.cw + w[t] <= c)     // w[t]装
                q.push(Node(t, cn.cw + w[t], cn.cr - w[t]));
            if (cn.cw + cn.cr - w[t] > bestw)     // w[t]不装
                q.push(Node(t, cn.cw, cn.cr - w[t]));
        }
    }

    int main() {
        cin >> n >> c;
        for (int i = 1; i <= n; i++)
            cin >> w[i];
        bestw = 0;
        bfs();
        cout << bestw << endl;
        return 0;
    }
```

*5. 旅行售货员问题

问题：（参见教材 6.7）

输入：顶点数 n；严格下三角部分的边权。

输出：最短 Hamilton 回路长度。

输出（对比观察）：+最短回路顶点序列（从顶点 1 开始）+运行时间(ms)。

样例输入	样例输出
5 10 200 50 30 300 20 100 400 10 60	160 1 2 3 5 4
15 10 200 50 30 300 20 100 400 10 60 20 300 100 20 80 80 10 10 10 100 70 30 600 90 80 50 60 700 10 60 20 800 200 20 100 90 90 900 200 100 10 40 30 100 400 20 300 100 20 80 10 10 10 100 70 100 120 30 50 20 60 110 200 500 100 90 20 300 100 20 80 200 660 80 20 10 100 70 70 30 510 320 20 330 80 200 50 200 160 50 80	270 1 2 15 4 7 3 12 14 5 10 13 9 6 11 8（答案不唯一）

50 10 320 20 330 80 200 50 200 160 50 80 170 30	

实验方法：优先队列式分支限界法。解空间树类型：排列树。
优先函数：最短路径下界 LB=当前路长+剩余顶点的最小出边和。
参考代码：

```
#include <queue>
#include <iostream>
#include <algorithm>
#include <iterator>
using namespace std;

#define N 30
int n;
int a[N+2][N+2];
int minlen;
int minedge[N+2];
int bestperm[N+2];
struct Node {
	int len;
	int perm[N+2];
	int cl;
	int lbound;
	bool operator < (const Node &b) const {
		return cl + lbound > b.cl + b.lbound;
	}	// 下界越小越优先
};

void swap(int &a, int &b)
{	int t = a;
	a = b;
	b = t;
}

void bfs() {
	priority_queue<Node> q;
	Node cn;
	cn.len = 1;
	cn.cl = 0;
	cn.lbound = 0;
	for (int i = 1; i <= n; i++) {
		cn.perm[i] = i;
		cn.lbound += minedge[i];
	}
	q.push(cn);
```

```
        while (!q.empty()) {
            cn = q.top();
            q.pop();
            Node nn = cn;
            nn.len = cn.len + 1;
            if (cn.len >= n) {
                if (cn.len > n)
                    break;
                nn.cl = cn.cl + a[cn.perm[n]][1];
                nn.len++;
                nn.lbound = 0;
                q.push(nn);
                if (cn.cl + a[cn.perm[n]][1] < minlen) {
                    minlen = cn.cl + a[cn.perm[n]][1];
                    copy(cn.perm+1, cn.perm+n+1, bestperm+1);
                }
                continue;
            }
            for (int i = cn.len + 1; i <= n; i++) {
                int k = cn.perm[i];
                swap(nn.perm[cn.len+1], nn.perm[i]);
                nn.cl = cn.cl + a[cn.perm[cn.len]][k];
                nn.lbound -= minedge[k];
                q.push(nn);
                swap(nn.perm[cn.len+1], nn.perm[i]);
                nn.lbound += minedge[k];
            }
        }
    }

    int main() {
        cin >> n;
        for (int i = 2; i <= n; i++)
            for (int j = 1; j < i; j++) {
                cin >> a[i][j];
                a[j][i] = a[i][j];
            }
        for (int i = 1; i <= n; i++) {
            minedge[i] = 0x7fffffff;
            for (int j = 1; j <= n; j++)
                if (j != i && a[i][j] < minedge[i])
                    minedge[i] = a[i][j];
        }
        minlen = 0x7fffffff;
```

```
        bfs();
        cout << minlen << endl;
        copy(bestperm+1, bestperm+n+1, ostream_iterator<int>(cout, " "));
        cout << endl;
        return 0;
}
```

课外 OJ 题目：

HDU2717 Catch that cow （*）

POJ1915 Knight moves （*）

POJ2362 Square （**）

POJ2415 Hike on a Graph （***）

POJ3328 Cliff Climbing （***）

实验 6 随机化算法（4 课时）

实验目的：掌握随机化算法的种类、特点、原理和设计步骤。能够使用随机化数值算法求解非线性方程组的数值解，能够根据问题的偏性和有限正确率设计高正确率的随机化算法，在很小时间复杂度内解决或近似解决高复杂度的算法问题。

实验内容和要求：

要求利用判定问题的必要条件或充分条件，设计偏真或偏假的 Monte Carlo 算法，利用随机数 C 函数 rand()、srand()编写随机化算法，并通过多次重复调用大幅提高结果的正确率。

1. 矩阵等式判断

OJ 问题：Matrix Multiplication（参见 http://poj.org/problem?id=3318，及校 ACM 训练题 WOJ1026）

题意：判断矩阵等式 AB=C 是否成立。

输入：n，表示方阵的阶，n≤1000。

A（n 行 n 列矩阵，按行空隔输入）。

B（n 行 n 列矩阵，按行空隔输入）。

C（n 行 n 列矩阵，按行空隔输入）。

输出：YES（相等）或 NO（不相等）。

输出（对比观察）：+n+运行时间(ms)。

样例输入	样例输出
2 1 5 0 0 10 20 20 30 110 170 0 0	YES
读取文件 MatrixABC1.txt	YES
读取文件 MatrixABC2.txt	NO

实验方法：（参见教材：算法分析题 7-16）Monte Carlo 算法。如果先将矩阵乘积 AB 计算出来，其计算复杂性是 $O(n^3)$。运用随机化算法可在 $O(n^2)$时间内解决。对任意一个由 1 与-1 组成的随机列向量 x，ABx=Cx 是 AB=C 的必要条件。利用该必要条件可以快速判断矩阵等式。

参考代码：

```
#include <iostream>
#include <fstream>
using namespace std;
#include <stdlib.h>
#include <stdio.h>
```

```
#define NN 1000
int p1, p2, p3;
int a[NN+2][NN+2], b[NN+2][NN+2], c[NN+2][NN+2];

bool MC() {
	int i, j;
	int x[NN], d[NN], e[NN];
	for (i = 1; i <= p3; i++)
		x[i] = rand()%2 * 2 - 1;
	for (i = 1; i <= p2; i++)
		for (d[i] = 0, j = 1; j <= p3; j++)
			d[i] += b[i][j] * x[j];
	for (i = 1; i <= p1; i++)
		for (e[i] = 0, j = 1; j <= p2; j++)
			e[i] += a[i][j] * d[j];		// e = abx
	for (i = 1; i <= p1; i++)
		for (d[i] = 0, j = 1; j <= p3; j++)
			d[i] += c[i][j] * x[j];		// d = cx
	for (i = 1; i <= p1; i++)
		if (e[i] != d[i])	// abx = cx?
			return false;
	return true;
}

int main() {
	int n;
	int i, j;

//	freopen("MatrixABC1.txt", "r", stdin);
	scanf("%d", &n);
	p1 = p2 = p3 = n;
	for (i = 1; i <= p1; i++)
		for (j = 1; j <= p2; j++)
			scanf("%d", &a[i][j]);
	for (i = 1; i <= p2; i++)
		for (j = 1; j <= p3; j++)
			scanf("%d", &b[i][j]);
	for (i = 1; i <= p1; i++)
		for (j = 1; j <= p3; j++)
			scanf("%d", &c[i][j]);
	bool ans = true;
	srand(12374);
	for (int t = 0; t < 10; t++) {
```

```
            if (!(ans = MC()))
                break;
        }
        printf("%s\n", ans ? "YES" : "NO");
        return 0;
    }
```

*2. 非线性方程组数值解

问题：求解如下非线性方程组

$$\begin{cases} 2x_1 + x_2 = 1 \\ 0.1 + x_3 = x_4 \\ \dfrac{x_1 + x_2 + x_3 + x_4}{4} = 0.5 \\ x_2^{\,2} + x_4^{\,2} = 1 \end{cases} \quad \text{或} \quad \begin{cases} 2x_1 + x_2 = 1 \\ 3\arcsin x_3 = 2\arctan \dfrac{x_3 + x_4}{3x_1 + x_2} \\ \dfrac{x_1 + x_2 + x_3 + x_4}{4} = 0.5 \\ x_2^{\,2} + x_4^{\,2} = 1 \end{cases}$$

已知其精确解是$(x_1, x_2, x_3, x_4)=(0.1, 0.8, 0.5, 0.6)$。[0.5, 0.5, 0.5, 0.5]作为搜索起点，搜索范围是超立方体$[0,1]^4$。要求精度分别达到 10^{-3}，10^{-4}，10^{-6}，观察需要的迭代次数。

输入：无。

输出：输出数值解，精确到小数 8 位。

输出（对比观察）：+迭代次数+误差评价值+运行时间(ms)。

实验方法：（参见教材 7.2.3）数值随机化算法。从初始解开始，反复沿随机化方向位移，若解的误差变小，视为成功，移动，同时步长扩大，否则视为失败，重选随机化方向。若连续多次失败，退回至上次成功移动过的起点，同时步长缩小。

解的误差视为步长 s 小于容许误差，同时方程组的均方误差也小于容许误差，记

$$e = \sqrt{\frac{1}{4}\sum_{i=1}^{4}(f_{iL}(x) - f_{iR}(x))^2}\ ,$$

方程组表示为 $f_{iL}=f_{iR}(i=1,2,3,4)$。当 s<epsilon AND e<epsilon 时迭代结束。

改进技术：

1. 采用相对误差。将每条方程改写成 $f_L=f_R$ 的形式，f_L, f_R 均为非零表达式，取相对误差为

$$e = \frac{(f_L(x) - f_R(x))^2}{f_L(x)^2 + f_R(x)^2}\ ,$$

再对 e 求和作为误差评价值。用此评价值代表解的精确程度能够消除方程式左右同时乘以任意倍数带来的不合理影响。

2. 随机化位移方向均匀化、归一化。均匀化：将 $\|\mathbf{d}\|>1$ 的随机位移舍弃，重新随机生成，直至 $0<\|\mathbf{d}\|\le 1$。归一化：将位移 **d** 转换为 $\dfrac{\mathbf{d}}{\|\mathbf{d}\|}$。

参考代码：

```
#include <math.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <algorithm>
using namespace std;

const int EquNum(4);

// 已知准确解是 x[0~3]=(0.1, 0.8, 0.5, 0.6)
double fL1(double *x) {
	double l = x[0]*2 + x[1];
	double r = 1;
	return (l-r) * (l-r) / (l*l + r*r);
}
/*
double fL2(double *x) {
	double l = x[3];
	double r = x[2] + 0.1;
	return (l-r) * (l-r) / (l*l + r*r);
}
*/
double fL2(double *x) {
	double l = 3.0 * asin(x[2]);
	double r = 2.0 * atan((x[2] + x[3]) / (x[0]*3 + x[1]));
	return (l-r) * (l-r) / (l*l + r*r);
}

double fL3(double *x) {
	double l = (x[0] + x[1] + x[2] + x[3]) / 4;
	double r = 0.5;
	return (l-r) * (l-r) / (l*l + r*r);
}
double fL4(double *x) {
	double l = x[1]*x[1] + x[3]*x[3];
	double r = 1.0;
	return (l-r) * (l-r) / (l*l + r*r);
}

double Evaluate(double *xx) {
	return sqrt((fL1(xx) + fL2(xx) + fL3(xx) + fL4(xx)) / 4.0);
}

void Output(double *x, int sn, double step) {
	printf("sn=%d,step=%f, x[0~3]=%f,%f,%f,%f, ", sn, step, x[0], x[1], x[2], x[3]);
	printf("Err=%.7f\n", Evaluate(x));
}
```

```
int SA() {
    double err = 1e10;
    double step = 0.01;
    double dx[EquNum], xprev[EquNum], xnext[EquNum];
    int sn = 0, mm = 0, TotalSteps = 1000;  // 迭代步数，失败搜索上限，步数上限
    double epsilon = 1e-6;

    srand(1);
    for (int i = 0; i < EquNum; i++)
        x[i] = xprev[i] = 0.5;
    while ((err > epsilon || step > epsilon) && sn < TotalSteps) {
        int rr = 0;
        for (int i = 0; i < EquNum; i++) {
            dx[i] = rand() % 32768 - 16384;
            rr += dx[i] * dx[i];
        }
        rr = sqrt((double)rr) * (1.0 + rand() / 32768.0);
        for (int i = 0; i < EquNum; i++) {
            dx[i] *= step / rr;    // 归一化到[0.5,1]step 半径的球环
            xnext[i] = x[i] + dx[i];
        }
        double e = Evaluate(xnext);
        if (e < err) {
            err = e;
            step *= 2;// 成功，步长扩大
            if (step > 0.1)
                step = 0.1;
            for (int i = 0; i < EquNum; i++) {
                xprev[i] = x[i];
                x[i] = xnext[i];
            }
            mm = 0;
            sn++;
//printf("sn=%d, err=%.9lf, (%.7lf,%.7lf,%.7lf)\n", sn, err, x[0], x[1], x[2]);
        } else {
            mm++;
            if (mm > 50) {
                for (int i = 0; i < EquNum; i++)    // 退回到前一点继续搜索
                    x[i] = xprev[i];
                err = Evaluate(x);
                step /= 2.5;    // 连续失败 50 次，步长缩小
                if (step < epsilon)
                    step = epsilon;
```

```
                    mm = 0;
                    sn++;
                }
            }
        }
        return err < epsilon;
    }
    int main() {
        int t;
        if (SA())
            printf("%.8lf %.8lf %.8lf %.8lf\n", x[0], x[1], x[2], x[3]);
        else
            printf("=== Fail: err=%.10lf ===\n", Evaluate(x));
        return 0;
    }
```

*3. 素数判定

问题：判断 n（long long 类型）是否为素数。

输入：整数 n（$1<n<2^{64}$）

输出：YES（n 是素数）或 NO（n 不是素数）

输出（对比观察）：+n+运行时间(ms)。

样例输入	样例输出
1729	NO
2017	YES
1000000007	YES
999101	YES

实验方法：（参见教材 7.5.3）Miller-Rabin 算法。利用费尔马小定理和二次探测定理所描述的素数必要条件，设计素数判定的随机化算法。该算法属于 Monte Carlo 算法。

课外 OJ 题目：

HDU4712 Hamming Distance （*）

HDU2138 HowManyPrimeNumbers

ZOJ1450 Minimal Circle （***）

POJ2420 A Star not a Tree （**数值随机化，模拟退火）

POJ2454 Jersey Politics （**Las Vegas）

实验 7 串与序列的算法（2 课时）

实验目的：掌握子串搜索算法、后缀数组算法和序列比较算法。

实验内容和要求：

要求用 KMP 算法解子串搜索问题，用倍增算法求后缀自动机，用 DP 求编辑距离。

1. 子序列查找问题

OJ 问题：Number Sequence（参见 http://acm.hdu.edu.cn/showproblem.php?pid=1711）

题意：在一个整数序列中查找整数子序列的首次出现位置。

输入：案例数 t。每个案例输入 n, m，分别表示序列 a 的长度和查找子序列 b 的长度。n ≤10^6，m≤10^4。

输出：b 在 a 中首次出现的位置。若没有出现，输出-1。

样例输入	样例输出
2 13 5 1 2 1 2 3 1 2 3 1 3 2 1 2 1 2 3 1 3 13 5 1 2 1 2 3 1 2 3 1 3 2 1 2 1 2 3 2 1	6 -1

实验方法：（参见教材 9.1.2）KMP 算法。

2. 编辑距离

OJ 问题：Edit Distance（参见 http://acm.fzu.edu.cn/problem.php?pid=1434，及校 ACM 训练题 WOJ1011）

题意：编辑距离是用编辑操作将一个串 s1[1...m]变为另一个串 s2[1...n]的最少操作次数，编辑操作包括三种：删除一个字符、插入一个字符和更改一个字符。输入两个串 s1、s2，计算它们的编辑距离。

输入：分两行输入两个字符串 X,Y，X 和 Y 都不含空格，长度最大 6000。

输出：编辑距离 d(X,Y)。

输出（对比观察）：+两个串的长度+运行时间(ms)。

样例输入	样例输出
smile simple	2
读取文件 EditDistance_in.txt	7
读取文件 EditDistance_in.txt 中两个串的前 1000 个字符	3

实验方法 1：（参见教材 9.3.1）动态规划。设立二维数组 d[m+1][n+1]，存储 s1 的 i-前

缀与 s2 的 j-前缀的编辑距离。最后所求是 d[m][n]。利用滚动数组进行空间优化，二维数组 d 的空间可以优化至 2max(m,n)。

实验方法 2（对比方法）：递归。设计递归函数 int recur(int i, int j)，主函数调用 recur(m, n)得到所求编辑距离。

实验方法 3（对比方法）：备忘录方法。设立二维数组 d[m+1][n+1]，存储 s1 的 i-前缀与 s2 的 j-前缀的编辑距离，初值全置-1。设计递归函数 int memoir(int i, int j)，主函数调用 memoir (m,n)得到所求编辑距离。

参考代码：

```
#include <string.h>
#include <iostream>
#include <fstream>
using namespace std;

#define NMAX 6000
int d[2][NMAX+2];       // d[i%2][j]：s1 的 i-前缀与 s2 的 j-前缀的编辑距离
char s1[NMAX+2], s2[NMAX+2];

int min(int a, int b, int c) {
    int t = a;
    if (b < t)    t = b;
    if (c < t)    t = c;
    return t;
}

int EditDistance(char *s1, char *s2) {
    // 使用动态规划算法计算串 s1 与 s2 的编辑距离，s1, s2 从下标 0 开始存储字符串。
    int n1, n2;
    int i, j;

    n1 = strlen(s1);
    n2 = strlen(s2);
    for (j = 0; j <= n2; j++)
        d[0][j] = j;
    for (i = 1; i <= n1; i++) {
        d[i%2][0] = i;
        for (j = 1; j <= n2; j++) {
            if (s1[i-1] == s2[j-1])
                d[i%2][j] = d[(i-1)%2][j-1];
            else
                d[i%2][j] = min(d[(i-1)%2][j], d[i%2][j-1], d[(i-1)%2][j-1]) + 1;
        }
    }
    return d[n1%2][n2];
}
```

```
int main() {
	cin >> s1 >> s2;
//	ifstream ifs("D:\\EditDistance_in.txt");
//	ifs >> s1 >> s2;	//	s1[1000] = s2[1000] = 0;
//	ifs.close();
	cout << EditDistance(s1, s2) << endl;
	return 0;
}
```

课外 OJ 题目：

HDU1358 Period（KMP next 函数应用）

HDU3068 最长回文（Manacher 算法）

POJ1699 Best Sequence（扩展 KMP）

实验 8 网络最大流（2 课时）

实验目的：掌握网络最大流问题的增广路算法。

实验内容和要求：

要求运用增广路算法及 Dinic 算法解决网络流的最大流量问题。

1. 最大流问题

OJ 问题：Drainage Ditches（参见 http://acm.hdu.edu.cn/showproblem.php?pid=1532）

题意：给定一个单源单汇的无向带权图 G=<V, E>，权值 w 表示流量上限，计算从源到汇的可行流的最大流。可行流是指满足容量约束和平衡约束的有向图<V, F>。

输入：边数 N，顶点数 M（1 为源，n 为汇）。接着输入 N 行描述容量网络的数据，每行 3 个数，分别是起止顶点编号、容量。

输出：最大流流量。

输出（对比观察）：+运行时间(ms)。

样例输入	样例输出
5 4 1 2 40 1 4 20 2 4 20 2 3 30 3 4 10	50

实验方法 1：（参见教材 8.2.2）Ford-Fulkerson 增广路算法。首先定义两组 map：

```
map<int, int> cap[N+2];      // 容量网络, size=出度
map<int, int> rflow[N+2];    // 残流网络, size=出度+入度
```

cap 为容量网络，cap[v]记录 v 的各条出边的容量。rflow 为残流网络，rflow[v]记录 v 的各条出边的残流流量（正值）或 v 的入边的残流流量（负值）。

然后设计一个先进先出队列式广搜函数 bfs，队列结点信息仅需包含一个网络结点编号。bfs 在网络流中广搜，一旦找到一条从源到汇的可增广路，返回可增广量。更新流量后继续广搜，直至找不到新的可增广路。最后的流量即最大流流量。

实验方法 2（对比方法）：Dinic 算法。一次广搜得到多条增广路。

参考代码：

```
#include <queue>
#include <map>
#include <algorithm>
#include <iostream>
using namespace std;

#define N 210
```

```
#define Source 1
#define Sink n

int n, m;
map<int, int> cap[N+2];// 容量网络, size=出度
map<int, int> rflow[N+2];    // 残流网络, size=出度+入度
int maxd[N+2];      // 每个节点当前可增广量的最大值, 控制剪枝用
int Prev[N+2];// 可增广路的前驱节点

int bfs() {
    map<int, int>::iterator it;
    for (int i = 1; i <= n; i++)
        maxd[i] = 0;
    queue<int> q;
    maxd[Source] = 0x7fffffff;    // 防止回到 s
    q.push(Source);
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        if (v == Sink)
            return maxd[v];      // 返回增广量
        for (it = rflow[v].begin(); it != rflow[v].end(); it++) {
            if (it->second == 0)// 跳过残流网络的不存在边
                continue;
            int u = it->first;
            if (min(maxd[v], it->second) > maxd[u]) {   // 控制剪枝
                maxd[u] = min(maxd[v], it->second);   // u 遇到可增广量更大的路
                Prev[u] = v;
                q.push(u);
            }
        }
    }
    return 0;
}

int main() {
    int inc;

    while (cin >> m >> n) {
        for (int i = 1; i <= n; i++) {
            cap[i].clear();
            rflow[i].clear();
        }
        for (int j = 0; j < m; j++) {
```

```
                int u, v, c;
                cin >> u >> v >> c;
                cap[u][v] += c;      // 合并平行边
                rflow[u][v] += c;    // 同步构建残流网络，已考虑存在对向容量边的情况
            }
            int maxflow = 0;
            while (inc = bfs()) {      // 反复广搜可增广路
                for (int v = Sink; v != Source; v = Prev[v]) { // 调整当前流，其实只需对残流网络调整
                    rflow[Prev[v]][v] -= inc;
                    rflow[v][Prev[v]] += inc;
                }
                maxflow += inc;
            }
            cout << maxflow << endl;
        }
        return 0;
    }
```

课外 OJ 题目：

HDU1532 Drainage Ditches（最大流）

FZU1844 Earthquake Damage（最大流）

参考文献：

[1] 王晓东：计算机算法设计与分析（第 5 版），电子工业出版社，2018
[2] 郑莉、董渊等：C++语言程序设计（第 4 版），清华大学出版社，2010
[3] 刘汝佳：算法竞赛入门经典（第 2 版），清华大学出版社，2014
[4] POJ（北大 OJ）：http://poj.org
[5] HDU（杭电 OJ）：http://acm.hdu.edu.cn
[6] ZOJ（浙大 OJ）：https://zoj.pintia.cn
[7] FZU（福州大学 OJ）：http://acm.fzu.edu.cn
[8] CFS（Codeforces）：http://codeforces.com
[9] BZOJ（黑暗爆炸）：http://www.lydsy.com/JudgeOnline/
[10] JSK（计蒜客）：https://nanti.jisuanke.com