第 2 部分 基本套路

第一章 模拟与高精度

恭喜大家完成了第一部分语言入门，相信大家已经可以使用 C++ 写出一些简单程序了。

各位读者有听说过“建模”一词吗？所谓“建模”，就是把事物进行抽象，根据实际问题来建立对应的数学模型。“抽象”并不意味着晦涩难懂；相反，它提供了大量的便利。计算机很难直接去解决实际问题，但是如果把实际问题建模成数学问题，就会大大地方便计算机来“理解”和“解决”。

举个生活中常见的例子：我们拿到了某次数学考试的成绩单，现在需要知道谁考得最好。当然不能把成绩单对着电脑晃一晃，然后问“谁考得最好？”。需要通过一种途径让计算机来理解这个问题。这个问题可以建模成：“给定数组 score[]，问数组内元素的最大值”。这样建模后，就能很方便的写程序解决问题了。对于这个问题，采用之前讨论过的“擂台法”，就可以给出答案。

如何把实际问题建模成数学问题，主要依靠我们的经验和直觉、当然还有你灵动的思维；而算法与数据结构，正是解决数学问题的两把利剑。从这一章开始会介绍一些程序设计竞赛中的一些常见套路算法，而下一部分会介绍基础的数据结构。如果已经认真学习完了第一部分，相信这一部分也不在话下。

这一章是语言部分的延伸，会介绍一些竞赛中会出现的“模拟题目”——这里的“模拟”不是指模拟某场比赛的模拟题，而是指让程序完整的按照题目叙述的方式执行运行得到最终答案。同时也会介绍可以计算很大整数的高精度运算方法。这一章对思维与算法设计的要求不高，但是会考验编程的基本功是否扎实。

第1节 模拟方法问题实例

在各类算法竞赛中，常常会出现各类“模拟题目”作为签到送分题。“模拟题目”并不是指某种算法，所以无法直接告诉大家如何去掌握模拟题目。本章中提供几种模拟题目的例题供读者举一反三。实际上，如果读者想要掌握模拟题目，就需要去多写题、多整理技术细节。

例 1：乒乓球（洛谷 P1042，NOIP 2003 普及组）

华华和朋友打球，得到了每一球的输赢记录：W 表示华华得一分，L 表示对手得一分，E 表示比赛信息结束。一局比赛刚开始时比分为 0:0。每一局中达到 11 分或者 21 分（赛制不同），且领先对方 2 分或以上的选手可以赢得这一局。现在要求输出在 11 分制和 21 分制两种情况下的每一局得分。输入数据每行至多有 25 个字母，最多有 2500 行。

分析：根据题意，只需要对读入的内容进行统计即可。使用数组 a 来记录下从最开始到结束的得分情况，如果是华华赢了就记为 1，反之记为 0。读到 E 的时候直接就不再继续读入，而读到换行符时也直接忽略。同时要记录他们一共打了几球（就是 n）。

然后分别对两种赛制进行计算。首先计分板双方都是 0。如果华华赢了，w 就增加 1，否则 l 就增加 1。如果发现计分板上得分高的一方达到了赛制要求的球数，而且分差也足够，就将计分板的得分输出，同时计分板清零开始下一局。到最后还要输出正在进行中的比赛的得分。

```
#include<iostream>
#include<cmath>
using namespace std;
int f[2] = {11, 21}; // 两种赛制的获胜得分
int a[25 * 2500 + 10], n = 0;
int main() {
    char tmp;
    while(1) {
        cin >> tmp; // 不断读入结果
        if(tmp == 'E') break;
        else if(tmp == 'W') a[n++] = 1; // 华华赢
        else if(tmp == 'L') a[n++] = 0; // 华华输
    }
    for(int k = 0; k < 2; k++){ // 两种赛制循环
        int w = 0, l = 0;
        for(int i = 0; i < n; i++ ){
            w += a[i]; l += 1 - a[i];
            if((max(w, l) >= f[k]) && abs(w - l) >= 2){ // 获胜者超过对应分数且超出
对手 2 分
                cout << w << ":" << l << endl;
                w = l = 0;
            }
        }
        cout << w << ":" << l << endl; // 未完成的比赛也要输出结果
        cout << endl;
    }
    return 0;
}
```

本题思路很简单，直接根据题意和生活常识模拟运算。但是还是有一些需要注意的地方：

1. 数组要开够，至少需要容纳 25*2500 条得分记录。
2. 读到 E 就停止读入了，后面的都忽略掉。同时遇到换行符等也要忽略。
3. 注意要分差 2 分以上才能结算一局的结果。
4. 最后还要输出正在进行中的比赛，就算是刚刚完成一局也要输出 0:0。

题目几乎每一句话都很关键，所以一定要认真审题。不过有些题目可能存在一些不严谨的地方造成歧义，所以在比赛中如果发现字面意思不清楚，可以找比赛组织者（监考老师、志愿者、答疑帖等）明确题意。

例 2：扫雷游戏（洛谷 P2670，NOIP 2015 普及组）

给出一个 $n \times m$ 的网格，有些格子埋有地雷。求问这个棋盘上每个没有地雷的格子周围（上下左右和斜对角）的地雷数。

输入第一行两个整数 n 和 $m $表示网格的行数和列数，接下来 n 行，每行 m 个字符，描述了雷区中的地雷分布情况。字符*表示相应格子是地雷格，字符?表示相应格子是非地雷格。m 和 n 不超过 100。

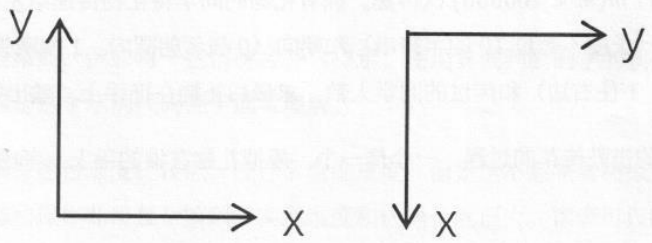
输出包含 n 行，每行 m 个字符，描述整个雷区。用*表示地雷格，用周围的地雷个数表示非地雷格。

分析：根据题意，对于每个非地雷的格子，只需要统计其上、下、左、右、左上、右上、左下、右下八个方向的格子中地雷的数量。如果需要统计的格子的坐标是 (x,y)，那么它左上角的坐标是 (x-1,y-1)。其他相邻的格子的坐标如下图：

(x-1,y-1)	(x-1,y)	(x-1,y+1)
(x,y-1)	(x,y)	(x,y+1)
(x+1,y-1)	(x+1,y)	(x+1,y+1)

【表：相对坐标偏移】

和大家熟悉的直角坐标系不一样，在这里上方是 x-1 而右方是 y+1。一般情况下，算法竞赛涉及的坐标系表示方法（下图右）相对于我们日常接触的直角坐标系（下图左）的表示方法是右转 90 度的，原点在左上角，如图。这样就能很方便的将行数和 x 对应起来，列数与 y 对应起来（比如第 1 行第 2 列可以表示成 (1,2)），便于存储和查询。



【图：直角坐标系方向】

没有必要列举出 8 个 if 语句逐一判定这个格子的 8 个方向是否是雷，只要将预先定义好这 8 个方向的格子相对于这个格子的偏移量，然后简单地循环就行了。

还有一点需要注意边界问题。在枚举某些边界上格子时，它的某些方向会超出了这个阵列，如果不经过特殊处理就可能造成数组越界。有两种方法可以解决这个问题，一是对需要查询的坐标进行范围判断，二是可以在这个阵列外围加一圈空白“虚拟的”非雷区参与统计。

至于怎么读入，怎么输出，是第一部分字符串中介绍过的，这里不再赘述。可以得到这样的代码：

```
#include <iostream>
using namespace std;
const int dx[] =  {1, 1, 1, 0, 0, -1, -1, -1};
const int dy[] =  {-1, 0, 1, -1, 1, -1, 0, 1};
// (i,j) 分别加上 (dx,dy) 就可以得到该点八个方向相邻的新坐标
const int maxn = 105;
char g[maxn][maxn]; // 函数外定义的变量默认初始化为 0，反正不是'*'。
int n, m;
int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> g[i][j];
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++)
        // 这里用到了一些偷懒的技巧，没有专门加上一圈“虚拟边框”，但不影响判断
            if (g[i][j] != '*') {
                int cnt = 0;
                for (int k = 0; k < 8; k++)
                    if (g[i + dx[k]][j + dy[k]] == '*') cnt++;
                cout << cnt;
            }
            else cout << "*";
        cout << endl; // 行末空格
    }
    return 0;
}
```

例 3：玩具谜题（洛谷 P1563，NOIP 2016 提高组）

$n(n < 100000)$ 名同学依次逆时针方向围成一个圈，有些同学面向圈内，有些面向圈外。第一个同学手里有礼物。从这名同学开始进行 $m(m < 100000)$ 次传递。拥有礼物的同学将礼物传递给左手/右手边的第 $s(s < n)$ 个人。已知每名同学的姓名（长度不超过 10 的字符串）和朝向（0 表示朝圈内，1 朝圈外），以及每次传递的方向（0 表示往这名同学左边，1 往右边）和传过的同学人数，求最后礼物在谁手上，输出姓名。

分析：如果模拟现实的击鼓传花的过程，一个传一个，模拟礼物在谁的手上，考虑到数据范围很大，两重循环的做法会超时。

把手里拥有礼物的第一个同学称为 0 号同学（因为数组习惯从 0 开始，最后一名同学是 n-1 号同学）。当他面朝内时，往右边传递 3 名同学，就会变成 0+3=3 号同学拥有礼物。如果 0 号同学往左边传递 3 名同学，就会变成 0-3=-3 号同学拥有礼物，显然没有 -3 号同学。考虑到 0 号同学和 7 号同学等价（因为 6 号同学的下一个就是 0 号同学），1 号同学和 8 号同学等价……可以得到猜出：-3 号同学和 -3+n=4 号同学等价。面朝外的同学往右边数 s 个就是编号增加 s，往左边数就是编号减少 s。如果得到的编号不在 0 到 n-1 的范围内则需要通过增减 n 调整到这个范围内，为了达到这个目的，可以使用取余数的方式。

如果这名同学面朝外，那么方向和加减的关系就和上面一种情况反过来，但是依然需要调整到 0 到 n-1 的范围内。

枚举这 4 种情况，可以得到这样的代码：

```
#include <iostream>
#include <string>
using namespace std;
const int MAXN = 1e6 + 5;
struct node { // 使用结构体存储同学的姓名和方向
    int head;
    string name;
}a[MAXN];
int n, m, x, y;
int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++)
        cin >> a[i].head >> a[i].name;
    int now = 0;
    for (int i = 1; i <= m; i++) {
        cin >> x >> y;
        if (a[now].head == 0 && x == 0) // 情况 1
            now = (now + n - y) % n;
        else if (a[now].head == 0 && x == 1) // 情况 2
            now = (now + y) % n;
        else if (a[now].head == 1 && x == 0) // 情况 3
            now = (now + y) % n;
        else if (a[now].head == 1 && x == 1) // 情况 4
            now = (now + n - y) % n;
    }
    cout << a[now].name << endl;
    return 0;
}
```

这个代码还能进行一些精简，例如将一些情况合并（提示：使用异或判断同学的朝向和传递方向甚至可以压缩成唯一的一种情况），但是这个示例代码便于读者理解。

虽然现在接触的模拟题目难度比较低，没什么思维难度，但是这不意味着模拟题目就一定是作为签到题。很多模拟题目状态和操作非常复杂繁琐，实现难度和代码长度很大。读者可以围观一下这些高难度模拟题，看看就好：- 猪国杀（洛谷 P2482，山东省选 2010）- 立体图（洛谷 P3326，山东省选 2015）- 杀蚂蚁（洛谷 P2586，浙江省选 2008）- 琪露诺的冰雪小屋（洛谷 P3693，By orangebird）

第2节 高精度运算

在语言入门部分已经介绍了几种数据类型，并且已经知道每种数据类型能够容纳的数字范围是有限的。一般情况下使用 int 类型，如果数字大一点还能使用 long long 类型。如果需要存储或者使用更大的整数该怎么办呢？不能使用浮点数，因为浮点数的有效数字位数也是有限的。有些编译器允许提供__int128 类型，但是不仅大

小依然局限（最多可以表示接近 40 位的十进制数），而且使用范围也很局限。不过可以使用数组来模拟非常长的整数。

例 4：A+B Problem 高精（洛谷 P1601）

分别在两行内输入两个 500 位以内的十进制非负整数，求他们的和。

分析：

用数组来模拟非常长的整数，这意味着可以用数组的每一位记录那个数字上的每一位。也就是说，可以用 n 位数组来记录一个 n 位数字。

解决了存储问题之后，再来回顾一下竖式加法：

```
  514
+ 495
------
 1009
```

竖式加法的实质就是模拟每一位的加法与进位。对于十进制加法来说，某一位上的和超过 9 时会产生进位现象，进位就是保留那一处数字的个位数，然后把十位上的数字加给下一位去，详见下表。

数	第四位	第三位	第二位	第一位
a		5	1	4
b		4	9	5
中间产物		9	10	9
处理进位 1		10	0	9
处理进位 2	1	0	0	9
结果	1	0	0	9

可以看到，必须从低位到高位处理进位，否则可能会发生顺序上的错误。如上表，只有先处理了第二位处的进位，才发现第三位处也需要进位。那么可以让处理进位和处理加法从低位到高位同时进行了。

```
#include <cstdio>
#include <string>
#define maxn 520
using namespace std;
int a[maxn], b[maxn], c[maxn];
int main() {
    string A, B;
    cin >> A >> B;
    int len = max(A.length(), B.length());
    for (int i = A.length() - 1, j = 1; i >= 0; i--, j++)
        a[j] = A[i] - '0';
    for (int i = B.length() - 1, j = 1; i >= 0; i--, j++)
        b[j] = B[i] - '0';
    for (int i = 1; i <= len; i++) {
        c[i] += a[i] + b[i];
```

```
        c[i + 1] = c[i] / 10;//模拟进位
        c[i] %= 10;
    }
    if (c[len + 1])//最后进位可能会导致位数增加
        len++;
    for (int i = len; i >= 1; i--)
        printf("%d", c[i]);
}
```

例 5：A*B Problem（洛谷 P1303）

分别在两行内输入两个 2000 位以内的十进制非负整数，求他们的积。

让先来回顾一下竖式乘法

```
   514
×  495
--------
  2570
 4626
2056
--------
254430
```

鉴于这样的形式十分不直观，再换个表示方式来观察它的实质

数	第六位	第五位	第四位	第三位	第二位	第一位
a				5	1	4
b				4	9	5
a*b[1]				25	5	20
a*b[2]			45	9	36	
a*b[3]		20	4	16		
中间产物		20	49	50	41	20
处理进位	2	5	4	4	3	0
结果	2	5	4	4	3	0

仔细观察，可以发现 a[i]*b[j] 的贡献全都在中间产物的第 i+j-1 位上，这个性质提供了一个简便的写法：我们可以把所有贡献算出来，最后一口气处理所有进位问题。这样可以避免处理多次进位事件，优化效率——计算机中取模的效率远低于加法和乘法。

这个进位模拟过程与加法过程中大同小异，都是把个位数留下，把个位以外的数字贡献给下一位，具体可见代码。

```
#include <cstdio>
#include <string>
#define maxn 5010
```

```
using namespace std;
int a[maxn], b[maxn], c[maxn];
int main() {
    string A, B;
    cin >> A >> B;
    int lena = A.length(), lenb = B.length();
    for (int i = lena - 1; i >= 0; i--)a[lena - i] = A[i] - '0';
    for (int i = lenb - 1; i >= 0; i--)b[lenb - i] = B[i] - '0';
    for (int i = 1; i <= lena; i++)
        for (int j = 1; j <= lenb; j++)
            c[i + j - 1] += a[i] * b[j];//计算贡献
    int len = lena + lenb;//乘积的位数不超过两数的位数之和
    for (int i = 1; i <= len; i++) {
        c[i + 1] += c[i] / 10;//处理进位
        c[i] %= 10;
    }
    for (; !c[len];)
        len--;//去掉前导零
    for (int i = max(1, len); i >= 1; i--)
        printf("%d", c[i]);
}
```

这样的高精度算法的复杂度是$O(n^2)$，其中 n 是数字的位数。如果位数过大的话计算速度还是比较慢。可以利用快速傅里叶变换的方式加速高精度乘法，将其复杂度优化到$O(n\log n)$，但是相关内容超出了本书的范围，且思维和实现难度都比较大，现阶段可以不必深入了解。

例 6：阶乘之和（洛谷 1009，NOIP 普及组 1998）

用高精度计算出$S = 1! + 2! + 3! + \cdots + n!\ (n \leq 50)$

分析：曾经在循环一章中介绍过本题，但是本题数据范围使得阶乘的数字非常大，即使是使用 long long 类型也会超过限度。因此需要使用高精度来计算。

这里涉及到了高精度加高精度与高精度乘低精度两个问题，可以采用封装大整数类并重载运算符的方式来编写这道题。封装结构体有一个好处是使得代码更加模块化，易于调试，并且使用时简洁干净，不易在接口上使用出错。

首先需要一个结构体来模拟大整数类。

```
#define maxn 100
struct Bigint {
    int len, a[maxn];    // 为了兼顾效率与代码复杂度，用 len 记录位数，a 记录每个数位
    Bigint(int x = 0) { //通过初始化使得这个大整数能够表示整形 x，默认为 0
        memset(a, 0, sizeof(a));
        for (len = 1; x; len++)
            a[len] = x % 10, x /= 10;
        len--;
    }
    int &operator[](int i) {
```

```
        return a[i];     //重载[]，可以直接用 x[i]代表 x.a[i]，编写时更加自然
    }
    void flatten(int L) { //一口气处理 1 到 L 范围内的进位并重置长度。需要保证 L 不小于有效长度。
        //因为相当于把不是一位数的位都处理成一位数，故取名为“展平”
        len = L;
        for (int i = 1; i <= len; i++)
            a[i + 1] += a[i] / 10, a[i] %= 10;
        for (; !a[len];)//重置长度成为有效长度
            len--;
    }
    void print() { //输出
        for (int i = max(len, 1); i >= 1; i--)
            printf("%d", a[i]);
    }
};
```

下一步是重载运算符 + 和 *，这样就可以像计算 int 类型一样直接对两个高精度对象加减了。

```
Bigint operator+(Bigint a, Bigint b) { // 表示两个 Bigint 类相加，返回一个 Bigint 类
    Bigint c;
    int len = max(a.len, b.len);
    for (int i = 1; i <= len; i++)
        c[i] += a[i] + b[i]; // 计算贡献
    c.flatten(len + 1); // 答案不超过 len+1 位，所以用 len+1 做一遍“展平”处理进位。
    return c;
}
Bigint operator*(Bigint a, int b) { //表示 Bigint 类乘整形变量，返回一个 Bigint 类
    Bigint c;
    int len = a.len;
    for (int i = 1; i <= len; i++)
        c[i] = a[i] * b;// 计算贡献
    c.flatten(len + 12);// int 位数有限，所以可以这样做一遍“展平”处理进位。
    return c;
}
```

前面千辛万苦封装结构体，就是为了最后能让大整数类型像整形那样轻松使用。计算阶乘的时候不需要每次循环中都要从 1 乘到 i，而是使用变量，每次乘 i 并记录阶乘的值即可，这样可以减少一重循环。

```
int main() {
    Bigint ans(0), fac(1);// 分别用 0 和 1 初始化 ans 与 fac，
    // 如果要将常数赋值给大整数，可以使用类似于 ans=Bigint(233) 的办法
    int m;
    cin >> m;
    for (int i = 1; i <= m; i++) {
        fac = fac * i; // 模拟题意
        ans = ans + fac;
    }
```

```
    ans.print(); // 输出答案
}
```

第3节 课后习题与实验

下面的题目经过了一些简化概括，如果对于细节还有不明确的地方，请登录题库查看原题。有些模拟题的题面较长，细节繁杂，读题时一定要认真仔细，实现代码的时候要耐心。

习题 1：魔法少女小 Scarlet（洛谷 P4924，By Scarlet）

Scarlet 首先把 1 到 n^2 的正整数按照从左往右，从上至下的顺序填入 $n \times n (n \leq 500)$ 的二维数组中，并进行 $m (m \leq 500)$ 次操作，使二维数组上将一个奇数阶方阵按照顺时针或者逆时针旋转 90°。每次操作给出 4 个整数 x, y, r, z，表示在这次魔法中，Scarlet 会把以第 x 行第 y 列为中心的 $2r+1$ 阶矩阵按照某种时针方向旋转，其中 $z=0$ 表示顺时针，$z=1$表示逆时针。要求输出最终所得的矩阵。

习题 2：字符串的展开（洛谷 P1098，NOIP2007 提高组）

给出一个长度不超过 100 的字符串和 3 个参数，要求对字符串展开，约定如下：

1. 遇到下面的情况需要做字符串的展开：在输入的字符串中，出现了减号“-”，减号两侧同为小写字母或同为数字，且按照 ASCII 码的顺序，减号右边的字符严格大于左边的字符。
2. 参数p_1：展开方式。$p_1=1$时，对于字母子串，填充小写字母；$p_1=2$时，对于字母子串，填充大写字母。这两种情况下数字子串的填充方式相同。$p_1=3$时，不论是字母子串还是数字字串，都用与要填充的字母个数相同的星号“*”来填充。
3. 参数p_2：填充字符的重复个数。$p_2=k$表示同一个字符要连续填充 k 个。例如，当$p_2=3$时，子串“d-h”应扩展为“deeefffgggh”。减号两边的字符不变。
4. 参数p_3：是否改为逆序：$p3=1$表示维持原来顺序，$p_3=2$表示采用逆序输出，注意这时候仍然不包括减号两端的字符。例如当$p_1=1$、$p_2=2$、$p_3=2$时，子串“d-h”应扩展为“dggffeeh”。
5. 如果减号右边的字符恰好是左边字符的后继，只删除中间的减号，例如：“d-e”应输出为“de”，“3-4”应输出为“34”。如果减号右边的字符按照 ASCII 码的顺序小于或等于左边字符，输出时，要保留中间的减号，例如：“d-d”应输出为“d-d”，“3-1”应输出为“3-1”。

习题 3：多项式输出（洛谷 P1067，NOIP2009 普及组）

一元n次多项式可表示为 $f(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0, a_n \neq 0$；给出一个一元多项式各项的次数和系数，请按照如下规定的格式要求输出该多项式：

1. 多项式中自变量为x，从左到右按照次数递减顺序给出多项式。
2. 多项式中只包含系数不为0的项。
3. 如果多项式n次项系数为正，则多项式开头不出现“+”号，如果多项式n次项系数为负，则多项式以“-”号开头。
4. 对于不是最高次的项，以“+”号或者“-”号连接此项与前一项，分别表示此项系数为正或者系数为负。紧跟一个正整数，表示此项系数的绝对值（如果一个高于0次的项，其系数的绝对值为1，则无需输出 1）。

如果x的指数大于1，则接下来紧跟的指数部分的形式为 x^b，其中 $b $为 $x $的指数；如果$x$的指数为 $1，则接下来紧跟的指数部分形式为"x$"；如果 x 的指数为$ 0$，则仅需输出系数即可。

5. 多项式中，多项式的开头、结尾不含多余的空格。

例如，当输入的系数为 `100 -1 1 -3 0 10`，则应当输出 `100x^5-x^4+x^3-3x^2+10`。

习题 4：作业调度方案（洛谷 P1065，NOIP2006 提高组）

用$m(m \leq 20)$台机器加工$n(n \leq 20)$个工件，每个工件都有m道工序，每道工序都在不同的指定的机器上完成。每个工件的每道工序都有指定的加工时间。每个工件的每个工序称为一个操作，用记号 j-k 表示一个操作，例如 2-4 表示第2个工件第4道工序的这个操作。

在本题中，还给定对于各操作的一个安排顺序。每个操作的安排都要满足以下的两个约束条件，且在安排后面的操作时，不能改动前面已安排的操作的工作状态。

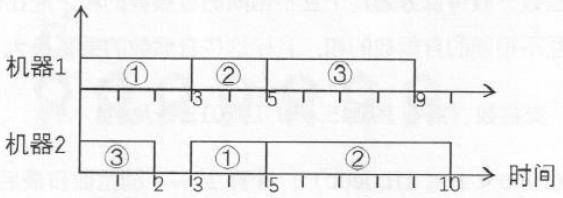
1. 对同一个工件，每道工序必须在它前面的工序完成后才能开始；
2. 同一时刻每一台机器至多只能加工一个工件。

还要注意，“安排顺序”只要求按照给定的顺序安排每个操作。不一定是各机器上的实际操作顺序。在具体实施时，有可能排在后面的某个操作比前面的某个操作先完成。

当一个操作插入到某台机器的某个空档时，保证约束条件 1 和 2 的条件下，尽量插入到最前面的一个空档，且靠前插入。例如，当 $n=2, m=3$，安排顺序为 1,1,2,3,3,2，各工序信息如下图 (a)，得到安排工序方案如图 (b)。

工件号	工序1	工序2
1	1/3	2/2
2	1/2	2/5
3	2/2	1/5

(a)



(b)

【图：工序信息和安排方案】

显然，在这些约定下，对于给定的安排顺序，符合该安排顺序的实施方案是唯一的，请你计算出该方案完成全部任务所需的总时间。

习题 5：帮贡排序[18]（洛谷 P1786，By absi2011）

[18] 建议学习完本书《排序》一章后再尝试完成本题。

absi2011 在玩一款在线帮派游戏。帮派内共最多有一位帮主，两位副帮主，两位护法，四位长老，七位堂主，二十五名精英，帮众若干。absi2011 作为帮派的副帮主，要对帮派内几乎所有人的职位全部调整一番。他给你每个用户的数据：他的名字、原来职位、帮派贡献（简称帮贡）、等级。

absi2011 要把护法的职位给帮贡最多的用户，其次长老，以此类推。但是 absi2011 无权调整帮主、副帮主的职位，包括他自己。

游戏中帮派成员列表是按照等级从高往低排序的，所以输入文件顺序也是按照职位第一关键字，等级第二关键字从高往低排序的。他想请你帮忙按照要求对这些成员进行帮贡排序、重新分配职位，然后输出调整职位后的成员列表，职位第一关键字，等级第二关键字。对于同职位同等级的用户，原先排名在前的，输出的排名还是在前面。对于同帮贡的用户，原列表在前的可以优先分配职位。

此外输入文件还遵循以下规则：

1. 保证职位必定为 BangZhu,FuBangZhu,HuFa,ZhangLao,TangZhu,JingYing,BangZhong 之中一个；
2. 保证有一名帮主，保证有两名副帮主，保证有一名副帮主叫 absi2011；
3. 不保证一开始帮派里所有职位都是满人的，但排序后分配职务请先分配高级职位；
4. 总人数不多于 110，名字长度不会超过 30 且不会重复，帮贡不大于 10^9，等级不大于 150。

习题 6：阶乘数码（洛谷 P1591）

求 $n!(n \leq 1000)$中某个数码 $a(0 \leq a \leq 9)$ 出现的次数。例如 $10! = 3628800$ 中，8 出现的次数是 2 次。

习题 7：最大乘积（洛谷 P1249）

一个正整数一般可以分为几个互不相同的自然数的和，现在你的任务是将指定的正整数 $n(3 \leq n \leq 10000)$ 分解成若干个互不相同的自然数的和，且使这些自然数的乘积最大。输出分解方案和最大乘积。

习题 8：麦森数（洛谷 P1045，NOIP2013 普及组）

输入P（$1000 < P < 3100000$），计算 $2^P - 1$ 的位数和最后 500 位数字，用十进制高精度数表示。（提示：判断位数可以列出方程$2^p = 10^q$。计算结果时，每次乘 2 速度很慢，所以可以每次乘 2^{30}，提升 30 倍效率。）