第1章　基础算法

本章介绍几个程序设计中最基础、最常见的算法，这些算法的思想简单明确，初学者首先应当掌握这些算法思想。大部分情况下，这些算法作为其他算法思想中的一部分，或者配合其他算法一起求解问题。

1.1　枚　　举

枚举算法是在程序设计中最为常见的一个算法，其核心在于枚举所有的可能性，找到可行解。枚举算法通常是将问题可能的答案一一列举，根据问题要求判断答案是否可行，保留可行解，舍弃不可行解，最终得出答案。

在实际生活中，枚举法十分常用。例如，询问一年中有 31 天的月份，就需要枚举 1~12 月每月的天数；询问考 90 分以上的科目，就需要枚举所有科目的分数；询问含热量低的食物，就需要枚举所有常见的食物进行比较……

在程序设计中，枚举算法也有着十分重要的作用。对于解题而言，经典的百钱买百鸡问题、解方程问题等，在数据范围较小时均可以使用枚举算法；对于很多数据范围较大的题目，自行枚举满足题意的、在较小范围内的所有答案，再通过假设、归纳，得出一般性的规律，能够使解题更加快速、高效。

注：在程序设计竞赛中有程序运行时间限制，所以需要合理选择枚举范围、方式，从而提高程序运行效率。

例题讲解

【例题 1.1】乘法表

题目描述

考虑一个 n 行 n 列的表，第 i 行第 j 列的元素的值为 $i \times j$，且行和列的序号从 1 开始。

给出一个正整数 x，计算该数 x 在表中出现的次数。

输入一行数据，包括两个数 n 和 x（$1 \leqslant n \leqslant 10^5, 1 \leqslant x \leqslant 10^9$），分别表示表格的行列数以及需要在表中查找的数。

输出一个数字，即 x 在表中出现的次数。

输入输出样例

Input	Output
6 12	4

样例解释

该表格如图1.1所示，出现 12 的位置标粗。

题目来源

Codeforces 577A　*https://codeforces.com/problemset/problem/577/A*

解题思路

首先考虑最简单的枚举思路：将表格每个位置上的值都计算出来，然后依次查找 x 是否出现。这样的枚举思路很清晰，但是其时间复杂度是 $O(n^2)$，显然这么做会超时，所以需要对该方法进行优化。

考虑到乘法表上某一行的每个数必定不同，故一行只可能最多出现一次 x，且 x 一定是该行的行号的倍数。于是可以枚举每一行，计算该行的行号 i 是否为 x 的因数，且其列数 x/i 小于或等于 n。如此一来，时间复杂度降为 $O(n)$，可以轻松解决该题。

1	2	3	4	5	6
2	4	6	8	10	**12**
3	6	9	**12**	15	18
4	8	**12**	16	20	24
5	10	15	20	25	30
6	**12**	18	24	30	36

图 1.1 样例解释

参考代码

```
#include <bits/stdc++.h>//万能头文件
using namespace std;
int n, x;
int main(){
    cin >> n >> x;
    int ans = 0;
    for (int i = 1; i <= n; i ++){
        if ((x % i == 0) && (x / i <= n)) ans ++;
    }
    cout << ans << endl;
    return 0;
}
```

【例题 1.2】猜数字

题目描述

猜数字游戏是 gameboy 最喜欢的游戏之一。游戏的规则是这样的：计算机随机产生一个四位数，然后玩家猜这个四位数是什么。每猜一个数，计算机都会告诉玩家猜对几个数字，其中有几个数字在正确的位置上。

比如计算机随机产生的数字为 1122。如果玩家猜 1234, 因为 1,2 这两个数字同时存在于这两个数中，而且 1 在这两个数中的位置是相同的，所以计算机会告诉玩家猜对了两个数字，其中一个在正确的位置。如果玩家猜 1111, 那么计算机会告诉他猜对了两个数字，其中有两个在正确的位置。

现在给你一段 gameboy 与计算机的对话过程，根据这段对话确定这个四位数是什么。

输入数据有多组。每组的第一行为一个正整数 $N(1 \leqslant N \leqslant 100)$，表示在这段对话中共有 N 次问答。在接下来的 N 行中，每行三个整数 A, B, C。gameboy 猜这个四位数为 A，然后计算机回答猜对了 B 个数字，其中 C 个在正确的位置上。当 $N = 0$ 时，输入数据结束。

每组输入数据对应一行输出。如果根据这段对话能确定这个四位数，则输出这个四位数；若不能，则输出 “Not sure”。

输入输出样例

Input	Output
6 4815 2 1 5716 1 0 7842 1 0 4901 0 0 8585 3 3 8555 3 2 2 4815 0 0 2999 3 3 0	3585 Not sure

题目来源

HDU 1172 *http://acm.hdu.edu.cn/showproblem.php?pid=1172*

解题思路

这道题如果从正面入手，分析每次猜测得到的信息对结果进行限制，显然是非常烦琐的。但想到四位数总共也不到 10^4 个，而数据最多有 100 组，所以可以采用枚举的方法。对每个枚举的四位数进行 N 次判断，如果这个数全部符合，则它就有可能是计算机产生的那个数。如果有多个这样的数，显然不能确定。

参考代码

```
#include <bits/stdc++.h>
using namespace std;
int n, a[110], b[110], c[110];
int ans, num;
bool check(int x, int y) {
    //四位数x是否符合第y次问答的结果
    int s[4], t[4], pa=a[y], pb=b[y], pc=c[y];
    for(int i = 3; i >= 0; i--)
        s[i] = x % 10, x /= 10, t[i] = pa % 10, pa /= 10;

    //检查过程
    int correct_num = 0, correct_pos = 0;
    for(int i = 0; i < 4; i++)
        if(s[i] == t[i]) correct_pos++;//在正确位置上的个数
    for(int i = 0; i < 4; i++)
        for(int j = 0; j < 4; j++)
            if(t[j] == s[i]) {
                correct_num++;//猜对的数字个数
                t[j] = -1;//防止同一个数字被使用多次
                break;
            }
    return correct_num == pb && correct_pos == pc;
}
int main(){
    while(cin >> n && n) {//该组有n次问答
        for(int i = 1; i <= n; i++)
            cin >> a[i] >> b[i] >> c[i];
        num = 0;
        //枚举范围内的每个值，判断是否符合条件
        for(int i = 1000; i < 10000 && num < 2; i++) {
            bool flag = true;
            for(int j = 1; j <= n; j++)
                if(!check(i, j)) {
                    flag = false;
                    break;
                }
            if(flag) num++, ans = i;
        }
        if(num == 1) cout << ans << endl; //有且只有一组符合
        else cout << "Not sure" << endl;
    }
    return 0;
}
```

习题推荐

- **Codeforces 724B** Batch Sort　*http://codeforces.com/problemset/problem/724/B*
- **HDU 4379** The More The Better　*http://acm.hdu.edu.cn/showproblem.php?pid=4379*
- **HDU 6463** 超级无敌简单题　*http://acm.hdu.edu.cn/showproblem.php?pid=6463*
- **Codeforces 1291C** Mind Control　*http://codeforces.com/problemset/problem/1291/C*
- **POJ 1753** Flip Game　*http://poj.org/problem?id=1753*

1.2 模　　拟

模拟就是用计算机来模拟题目中要求的操作，按照题目给出的十分明确的规则对输入数据处理，并按照输出规则输出。这类题目通常需要仔细阅读题面，标注出关键语句，避免出现信息遗漏。思考所有需要注意的细节，编写代码时需层次分明，并加上适当的注释辅助阅读代码。

模拟题有着以下非常鲜明的三点特征。

(1) 题目描述较长，主要是为了清晰地描述题目定义的规则（也有题目因使用公认的规则而描述较短，如大数的四则运算等）。

(2) 基本不涉及高深的算法，也不需要对题目本身进行较深入的思考。

(3) 代码量大，细节较多，出现错误不易排查。

例题讲解

【例题 1.3】幻方

题目描述

幻方是一个 3×3 的正方形，其中每个元素都是 1~9 中的一个数字，每个数字正好出现一次。在一个幻方中有 4 个不同的 2×2 的子正方形，从左上到右下按先行后列顺序分别标记为 1~4，且这些 2×2 的子正方形可以旋转。如果顺时针旋转该子正方形 90°，使用字母“C”表示；如果逆时针旋转该子正方形 90°，使用字母“R”表示。

输入的第一行为一个整数 $T(1 \leqslant T \leqslant 100)$，表示测试用例的数量。

每个测试用例的第 1 行为一个整数 $n(1 \leqslant n \leqslant 100)$，表示旋转次数。紧接着的 3 行表示 3×3 的幻方，其中 1~9 的每个数字恰好出现一次，表示幻方的初始状态。

接下来的 n 行描述了旋转的顺序，例如，1C 表示第一个子正方形顺时针旋转 90°，测试数据保证输入有效。

对于每个测试用例，输出旋转 n 次后的 3×3 的幻方。

输入输出样例

Input	Output
1 2 123 456 789 1C 4R	413 569 728

题目来源

HDU 6401　*http://acm.hdu.edu.cn/showproblem.php?pid=6401*

解题思路

由于题目的数据范围很小，所以解决该题可以直接模拟 n 次旋转过程。

参考代码

```
#include <bits/stdc++.h>
using namespace std;
char a[3][3];
void rotate(char num,char direction){
    int x,y;//旋转的子矩阵左上角坐标
    switch (num){
        case '1':
            x = 0, y = 0;
            break;
        case '2':
            x = 0, y = 1;
            break;
```

```
        case '3':
            x = 1, y = 0;
            break;
        case '4':
            x = 1, y = 1;
            break;
    }
    if (direction == 'C'){
        char tmp = a[x][y];
        a[x][y] = a[x + 1][y];
        a[x + 1][y] = a[x + 1][y + 1];
        a[x + 1][y + 1] = a[x][y + 1];
        a[x][y + 1] = tmp;
    }else{
        char tmp = a[x][y];
        a[x][y] = a[x][y + 1];
        a[x][y + 1] = a[x + 1][y + 1];
        a[x + 1][y + 1] = a[x + 1][y];
        a[x + 1][y] = tmp;
    }
}
int main(){
    int T;
    cin >> T;
    while (T--){
        int q;
        cin >> q;
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                cin >> a[i][j];
        char num,dir;
        while (q--){
            cin >> num >> dir;
            rotate(num, dir);//执行旋转操作
        }
        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 3; j++)
                cout << a[i][j];
            cout<< endl;
        }
    }
    return 0;
}
```

【例题 1.4】糖豆人

题目描述

糖豆人是最近很火的一款游戏。糖豆人有一个项目叫作登山比拼，玩家将在穿越许多障碍后到达目的地。到达目的地后，他们需要夺取王冠。但是王冠在上下移动，只有当王冠不高于 h 米时，玩家才可以抓住它。

现在，有 n 名选手参加了登山比拼。比赛开始时，王冠的高度为 0 米，然后王冠以 1 米/秒的速度向上移动，当高度为 H 米时，王冠会立即以 1 米/秒的速度向下移动，直到高度降低到 0 米，然后反复向上移动和向下移动。

第 i 个玩家到达目的地的时间是 x_i。当某个玩家到达目的地时，如果王冠的高度大于 h 米，他会在原地等待，否则他将立即跳起来争抢王冠。但是，由于网络不好，每个玩家都有一个延迟时间 c_i。假设一个玩家在 t 秒抢到皇冠，系统将确定他抢到皇冠的时刻是 $(t + c_i)$ 秒。

第一个夺得王冠的选手将获胜。如果多个玩家同时抢冠，赢家是序号较小的玩家。

给你所有玩家的到达时间 x_i 和延迟时间 c_i，请计算出谁将是最终的赢家。

输入包含多组样例。

第一行为一个整数 $T(1 \leqslant T \leqslant 20)$，表示测试数据的组数，每组测试数据格式如下。

第一行包括三个整数 $n,h,H(1 \leqslant n \leqslant 2\times10^5,1 \leqslant h \leqslant H \leqslant 300)$。第二行包括 n 个数 $x_1,x_2,\cdots,x_n(1 \leqslant x_i \leqslant 2\times10^5)$，表示第 i 个玩家的到达时间。第三行包括 n 个数 $c_1,c_2,\cdots,c_n(1 \leqslant c_i \leqslant 2\times10^5)$，表示第 i 个玩家的延迟时间。

对于每组测试数据，打印出获胜者的序号。

输入输出样例

Input	Output
2 4 2 5 3 6 1 9 6 4 2 3 3 1 2 1 2 3 4 5 6	3 1

题目来源

Codeforces Gym 102801D *https://codeforces.com/gym/102801/problem/D*

解题思路

首先确定题目的关键信息：

(1) 王冠在 $0 \sim H$ 米上下来回移动，移动速度为 1 米/秒。

(2) 王冠只有在小于或等于 h 米时才能被抓取。

(3) 玩家抓取成功后有网络延迟时间。

(4) 如果到达时间相同，则序号小的玩家获胜。

这样，就可以将每个玩家的最终结束时间 (由三部分时间组成：到达时间、等待时间和延迟时间) 计算出来，最终模拟该过程即可。

注：需要注意王冠运动方向的判断。此外，由于输入数据量大，使用 cin 和 cout 会超时，需要使用 scanf 和 printf。

参考代码

```
#include <bits/stdc++.h>
using namespace std;
int n, h, H;
struct node{
    int x, c;
};
node a[200010];
int main(){
    int T;
    scanf("%d", &T);
    while (T --){
        scanf("%d%d%d", &n, &h, &H);
        for (int i = 1; i <= n; i ++)
            scanf("%d", &a[i].x);
        for (int i = 1; i <= n; i ++)
            scanf("%d", &a[i].c);
        int min_t = 2e6 + 10, min_pos = 2e6 + 10;
        for (int i = 1; i <= n; i ++){
            int f = (a[i].x / H) % 2;//0-up
            int d = 0;
            if (f == 0)
                d = a[i].x % H;
            else
                d = H - a[i].x % H;
            int jump_t = a[i].x + a[i].c;
            if (d > h){
                if (f == 1)
                    jump_t += d - h;
                else
                    jump_t += (H - d) + (H - h);
```

```
            }
            if (jump_t == min_t && i < min_pos)
                min_pos = i;
            if (jump_t < min_t){
                min_pos = i;
                min_t = jump_t;
            }
        }
        printf("%d\n", min_pos);
    }
    return 0;
}
```

习题推荐

- **Codeforces 268A** Game　*https://codeforces.com/problemset/problem/268/A*
- **Codeforces 1216A** Prefixes　*https://codeforces.com/problemset/problem/1216/A*
- **Codeforces 1200A** Hotelier　*https://codeforces.com/problemset/problem/1200/A*
- **POJ 1029** False coin　*http://poj.org/problem?id=1029*
- **POJ 1083** Moving Tables　*http://poj.org/problem?id=1083*
- **POJ 2271** HTML　*http://poj.org/problem?id=2271*
- **POJ 1002** 487-3279　*http://poj.org/problem?id=1002*

1.3　递　　归

递归就是在程序运行过程中，某个函数或过程自己调用自己，通过重复将问题分解为同类子问题并解决问题的一种方法。

例如，假设你坐在教室的最后一排，你需要知道第一排同学的姓名且不能走动，那么你可以询问前一排的同学："第一排的同学姓名是什么?"；随后，前一排的同学又继续重复该过程，直到第一排的同学回答："我叫小明"，并将此信息逐个往回传递，最终传到你的耳中。这个询问的过程就是在不断递归调用"询问"函数。

又例如，斐波那契数列：$1, 1, 2, 3, 5, 8, \cdots$，如果使用 Fib(n) 表示数列第 n 项，那么除了基本情况 $\mathrm{Fib}(1) = 1, \mathrm{Fib}(2) = 1$ 以外，当 $n \geqslant 3$ 时，有 $\mathrm{Fib(n)} = \mathrm{Fib(n-1)} + \mathrm{Fib(n-2)}$ 成立，这个式子就是递归式，int Fib(int) 即为递归函数。

必备条件

递归算法的必备条件有以下两点。

(1) 子问题与原始问题内容相同，且更为简单。

(2) 不能无限制地调用自身，需要有出口结束递归过程。

算法流程

以直接递归为例，大部分递归程序采用以下算法形式实现。

算法 1.1　直接递归

```
function Recursion( )
    ···
    if ··· then
        ···
        return
    end if
    ···
    Recursion( )
    ···
end function
```

例题讲解

【例题 1.5】汉诺塔问题

题目描述

汉诺塔（Hanoi Tower），又称河内塔，源于印度的一个古老传说。

大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着 64 片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上，并且规定任何时候，在小圆盘上都不能放大圆盘，且在三根柱子之间一次只能移动一个圆盘。

现在假设三个柱子标号为 A、B、C，在第一个柱子上有 n 个圆盘，从上到下的第 i 个圆盘的半径为 i 且标号为 i。现在输入 n，请打印出移动的过程。

输入输出样例

Input	Output
3	Move 1 From A to C Move 2 From A to B Move 1 From C to B Move 3 From A to C Move 1 From B to A Move 2 From B to C Move 1 From A to C

解题思路

首先，由于最终的目标是将 A 柱上的圆盘全部转移到 C 柱上，那么 B 柱相当于一个中转站。定义函数 hanoi(n, x, y, z) 表示将 x 柱上前 n 个盘子经 y 柱中转后移动到 z 柱上。那么答案自然是 hanoi(n, 'A', 'B', 'C')。

现在需要考虑这个函数具体的实现方法。首先，由于移动的最下面的圆盘无法放在比它小的圆盘上，所以需要把其上面的所有圆盘全部移动至中转盘后（hanoi(n − 1, x, z, y)），再将其移动到目的盘（move 操作），最后将中转盘上的圆盘移动至目的盘上（hanoi(n − 1, y, x, z)），才能完成任务。

此时，关于递归函数的实现过程就完成了。注意，需要设置递归的结束条件。

参考代码

```
#include <bits/stdc++.h>
using namespace std;
void move(int id, char from, char to){
    cout << "Move " << id << " From " << from << " to " << to << endl;
}
void hanoi(int n, char x, char y, char z){
    if (n == 0) return;
    hanoi(n - 1, x, z, y);
    move(n, x, z);
    hanoi(n - 1, y, x, z);
}

int main(){
    int n;
    cin >> n;
    hanoi(n, 'A', 'B', 'C');
    return 0;
}
```

【例题 1.6】放苹果

题目描述

把 M 个同样的苹果放在 N 个同样的盘子里，允许有的盘子空着不放，问共有多少种（用 K 表示）不同的分法？应当注意，5，1，1 和 1，5，1 是同一种分法。

输入的第一行是测试数据的数目 $t(0 \leqslant t \leqslant 20)$。以下每行均包含两个整数 M 和 N，以空格分开，$1 \leqslant M, N \leqslant 10$。

对输入的每组数据 M 和 N，用一行输出相应的方案数 K。

输入输出样例

Input	Output
1 7 3	8

题目来源

POJ 1664　*http://poj.org/problem?id=1664*

解题思路

定义 $f(m,n)$ 表示有 m 个苹果，n 个盘子的方案数。

首先，当 $m<n$ 时，一定有 $f(m,n)=f(m,m)$ 成立，因为此时无论如何放置，一定有 $n-m$ 个盘子未放苹果。

其次，当 $m\geqslant n$ 时，$f(m,n)$ 由两部分组成：$f(m,n-1)$ 表示至少有一个盘子为空；$f(m-n,n)$ 表示每个盘子都至少有一个苹果，则 $f(m,n)=f(m,n-1)+f(m-n,n)$。

最终确定递归出口。当 $n=1$ 时，盘子数为 1，方案数只能是 1；当 $m=0$ 时，无苹果可放置，无论 n 取多少全部盘子均为空，方案数为 1。

参考代码

```
#include <iostream> //POJ不支持万能头文件
using namespace std;
int m, n;
int f(int m, int n) {//递归函数
    if(n == 1 || m == 0){
        return 1;
    }
    if(m < n) {
        return f(m, m);
    }else {
        return f(m, n - 1) + f(m - n, n);
    }
}
int main(){
    int T;
    cin >> T;
    while (T --){
        cin >> m >> n;
        cout << f(m, n) << endl;
    }
    return 0;
}
```

习题推荐

- **Codeforces 743B** Chloe and the sequence　*https://codeforces.com/problemset/problem/743/B*
- **Codeforces 897C** Nephren gives a riddle　*https://codeforces.com/problemset/problem/897/C*
- **POJ 2663** Tri Tiling　*http://poj.org/problem?id=2663*

1.4　分治基础

分治，即“分而治之”。它的核心思想是将一个复杂的问题分成若干个相同或相似的子问题，最终子问题可以简单地求解，再将子问题的解合并，得到最终的解。

对于一个规模为 n 的问题，若该问题可容易地解决（如 n 较小）则直接解决，否则将其分解为 k 个规模较小的子问题，这些子问题互相独立且与原问题形式相同，递归地解这些子问题，然后将各子问题的解合并得到原问题的解。这种算法设计策略叫作分治法。

如果原问题可分割成为 k 个子问题，$1 < k \leqslant n$ 且这些子问题都可解，并可利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求解。

特征

能用分治算法解决的问题一般具有如下特征。

(1) 该问题规模缩小到一定程度后可以较容易解决。

(2) 该问题可以分解为若干规模较小的相同问题，子问题的解合并即为该问题的解。

(3) 子问题的解是独立的，即子问题之间不存在公共的子问题。

应用

分治算法有许多应用，例如，一些排序算法（归并排序、快速排序），树分治（点分治、边分治），快速傅里叶变换，Strassen 矩阵乘法等。

注：本章介绍较为基础的分治算法，进阶的分治算法详见第 8 章。

复杂度

分治法通过将问题划分为规模最小的子问题，递归地解决划分后的子问题，再将结果合并，从而高效地解决问题。复杂度一般为 $O(\log n)$。

基本步骤

分治法的基本步骤如下。

(1) （分解）把一个大问题分解成多个小问题。

(2) （解决）分别解决每个小问题。

(3) （合并）把小问题的解组合起来。

例题讲解

【例题 1.7】蜘蛛牌

题目描述

蜘蛛牌是 Windows XP 操作系统自带的一款纸牌游戏，游戏规则是：只能将牌拖到比它大 1 的牌上面（A 最小，K 最大），如果拖动的牌上有按顺序排好的牌时，那么这些牌也跟着一起移动，游戏的目的是将所有的牌按同一花色从小到大排好，简单起见，我们的游戏只有同一花色的 10 张牌，从 A 到 10，且随机地在一行上展开，编号为 1~10，把第 i 号上的牌移到第 j 号牌上，移动距离为 $\mathrm{abs}(i-j)$，现在你要做的是求出完成游戏的最小移动距离。

第一个输入数据是 T，表示数据的组数。每组数据有一行，10 个输入数据，数据的范围是 $[1,10]$，分别表示 $A \sim 10$，且保证每组数据都是合法的。

对应每组数据输出最小移动距离。

输入输出样例

Input	Output
1 1 2 3 4 5 6 7 8 9 10	9

题目来源

HDU 1584 *http://acm.hdu.edu.cn/showproblem.php?pid=1584*

解题思路

本题需要计算将 1（A）~ 10 叠放到一起的最小移动距离，该问题可以分解成两部分：计算将数字 1~k 叠放到一起的最小移动距离、计算将数字 $(k+1)$~10 叠放到一起的最小移动距离。最终答案即为这两部分的答案加上将数字 k 叠放至数字 10 的移动距离，答案的最小值即为最终结果。