程序设计实习

算法基础

张勤健
zqj@pku.edu.cn

北京大学信息科学技术学院

2024 年 5 月 15 日

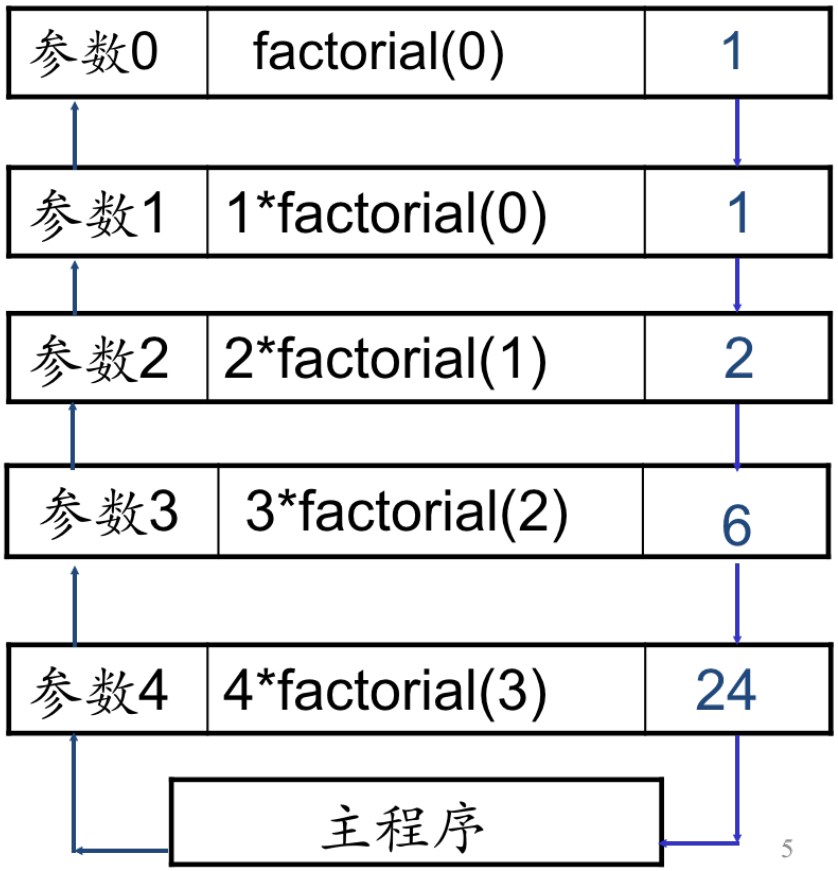
递归的基本概念

- 一个函数调用其自身，就是递归
- 求 $n!$ 的递归函数

```
int factorial(int n) {
  if (n == 0)  return 1;
  return  n * factorial(n - 1);
}
```

递归的基本概念

```
int factorial(int n) {
  if (n == 0)  return 1;
  return  n * factorial(n - 1);
}
```



递归的作用

1. 解决本来就是用递归形式定义的问题
2. 将问题分解为规模更小的子问题进行求解
3. 替代多重循环

用递归解决递归形式的问题

波兰表达式
波兰表达式是一种把运算符前置的算术表达式，例如普通的表达式 2 + 3 的波兰表示法为 + 2 3。波兰表达式的优点是运算符之间不必有优先级关系，也不必用括号改变运算次序，例如 (2 + 3) * 4 的波兰表示法为 * + 2 3 4。本题求解波兰表达式的值，其中运算符包括 + - * /四个。
波兰表达式的定义：

1. 一个数是一个波兰表达式，值为该数
2. " 运算符 波兰表达式 波兰表达式" 是波兰表达式，值为两个波兰表达式的值运算的结果

逆波兰表达式

- 样例输入
  * + 11.0 12.0 + 24.0 35.0
- 样例输出
  1357.000000
- 提示
  (11.0+12.0)*(24.0+35.0)

波兰表达式

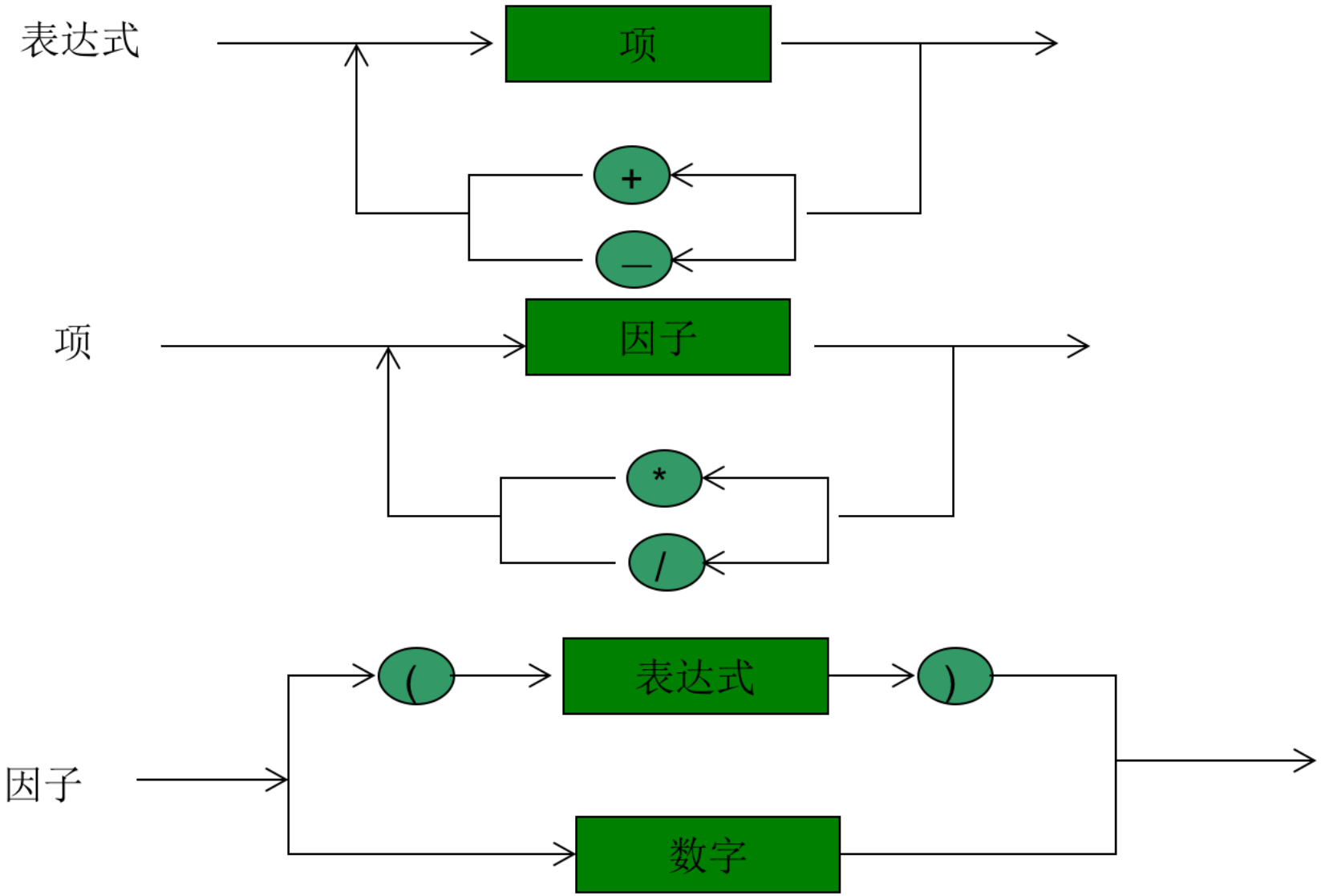
```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  double expression() {
6    //读入一个波兰表达式，并计算其值
7    string s;
8    cin >> s;
9    if (s == "+")  return expression() + expression();
10   else if (s == "-") return expression() - expression();
11   else if (s == "*") return expression() * expression();
12   else if (s == "/") return expression() / expression();
13   else  return atof(s.c_str());
14 }
15
16 int main() {
17   cout << expression() << endl;
18   return 0;
19 }
```

表达式求值

输入为四则运算表达式，仅由数字、+、−、*、/ 、(、) 组成，没有空格，要求求其值。假设运算符结果都是整数。"/" 结果也是整数

表达式求值

表达式是个递归的定义：



表达式求值

```
1  #include <iostream>
2  using namespace std;
3
4  int factor_value();
5  int term_value();
6  int expression_value();
7
8  int main() {
9    cout << "Enter an expression: " << endl;
10   cout << "The result is " << expression_value() << endl;
11   return 0;
12 }
```

表达式求值

```
11  int expression_value() {  //求一个表达式的值
12    int result = term_value(); //求第一项的值
13    bool more = true;
14    while (more) {
15      char op = cin.peek();
16      if (op == '+' || op == '-') {
17        cin.get();
18        int value = term_value();
19        if (op == '+')  result = result + value;
20        else  result = result - value;
21       } else {
22        more = false;
23      }
24    }
25    return result;
26  }
```

表达式求值

```
27  int term_value() { //求一个项的值
28    int result = factor_value(); //求第一个因子的值
29    bool more = true;
30    while (more) {
31      char op = cin.peek();
32      if( op == '*' || op == '/') {
33        cin.get();
34        int value = factor_value();
35        if( op == '*')  result = result * value;
36        else  result = result / value;
37      } else {
38        more = false;
39      }
40    }
41    return result;
42  }
```

表达式求值

```
43  int factor_value() {//求一个因子的值
44    int result = 0;
45    char c = cin.peek();
46    if (c == '(') {
47      cin.get();
48      result = expression_value();
49      cin.get();
50    } else {
51      cin >> result;
52    }
53    return result;
54  }
```

最大公约数问题

给定两个正整数，求它们的最大公约数。
求最大公约数可以使用辗转相除法 (欧几里德算法)：
假设 a $>$ b $>$ 0，那么 a 和 b 的最大公约数等于 b 和 a%b 的最大公约数，然后把 b 和 a%b 作为新一轮的输入。由于这个过程会一直递减，直到 a%b 等于 0 的时候，b 的值就是所要求的最大公约数。比如：9 和 6 的最大公约数等于 6 和 9%6=3 的最大公约数。由于 6%3==0，所以最大公约数为 3。

最大公约数问题

```
1  #include <stdio.h>
2
3  int gcd(int a, int b) {
4    if (b == 0) return a;
5    return gcd(b, a % b);
6  }
7
8  int main() {
9    int a, b;
10   scanf("%d%d", &a, &b);
11   printf("%d", gcd(a, b));
12   return 0;
13 }
```

爬楼梯

- 描述
  树老师爬楼梯，他可以每次走 1 级或者 2 级，输入楼梯的级数，求不同的走法数例如：楼梯一共有 3 级，他可以每次都走一级，或者第一次走一级，第二次走两级也可以第一次走两级，第二次走一级，一共 3 种方法。
- 输入
  输入包含一行，一个正整数 N，代表楼梯级数，$1 <= N <= 30$
- 输出
  输出走法的数量
- 样例输入
  10
- 样例输出
  89

爬楼梯

n 级台阶的走法 =
先走一级后，n-1 级台阶的走法 +
先走两级后，n-2 级台阶的走法

爬楼梯

n 级台阶的走法 =
先走一级后，n-1 级台阶的走法 +
先走两级后，n-2 级台阶的走法
$f(n) = f(n-1) + f(n-2)$

爬楼梯

n 级台阶的走法 =
先走一级后，n-1 级台阶的走法 +
先走两级后，n-2 级台阶的走法
$f(n) = f(n-1) + f(n-2)$
边界条件：

爬楼梯

n 级台阶的走法 =
先走一级后，n-1 级台阶的走法 +
先走两级后，n-2 级台阶的走法
$f(n) = f(n-1) + f(n-2)$
边界条件：
n $<$ 0 0
n $=$ 0 1

爬楼梯

```
1  #include <stdio.h>
2
3  int stairs(int n) {
4    if (n < 0)  return 0;
5    if (n == 0) return 1;
6    return stairs(n-1) + stairs(n-2);
7  }
8
9  int main() {
10   int n;
11   scanf("%d", &n);
12   printf("%d", stairs(n));
13   return 0;
14 }
```

放苹果

- 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，问共有多少种不同的分法?（用 K 表示）5，1，1 和 1，5，1 是同一种分法。
- 输入
  第一行是测试数据的数目 t（0 <= t <= 20）。以下每行均包含二个整数 M 和 N，以空格分开。1<=M，N<=10。
- 输出
  对输入的每组数据 M 和 N，用一行输出相应的 K。
- 样例输入
  1
  7 3
- 样例输出
  8

放苹果

总放法 = 有盘子为空的放法 + 没盘子为空的放法

设 i 个苹果放在 k 个盘子里放法总数是 $f(i,k)$，则：

$$f(i,k) = f(i,k-1) + f(i-k,k)$$

边界条件?

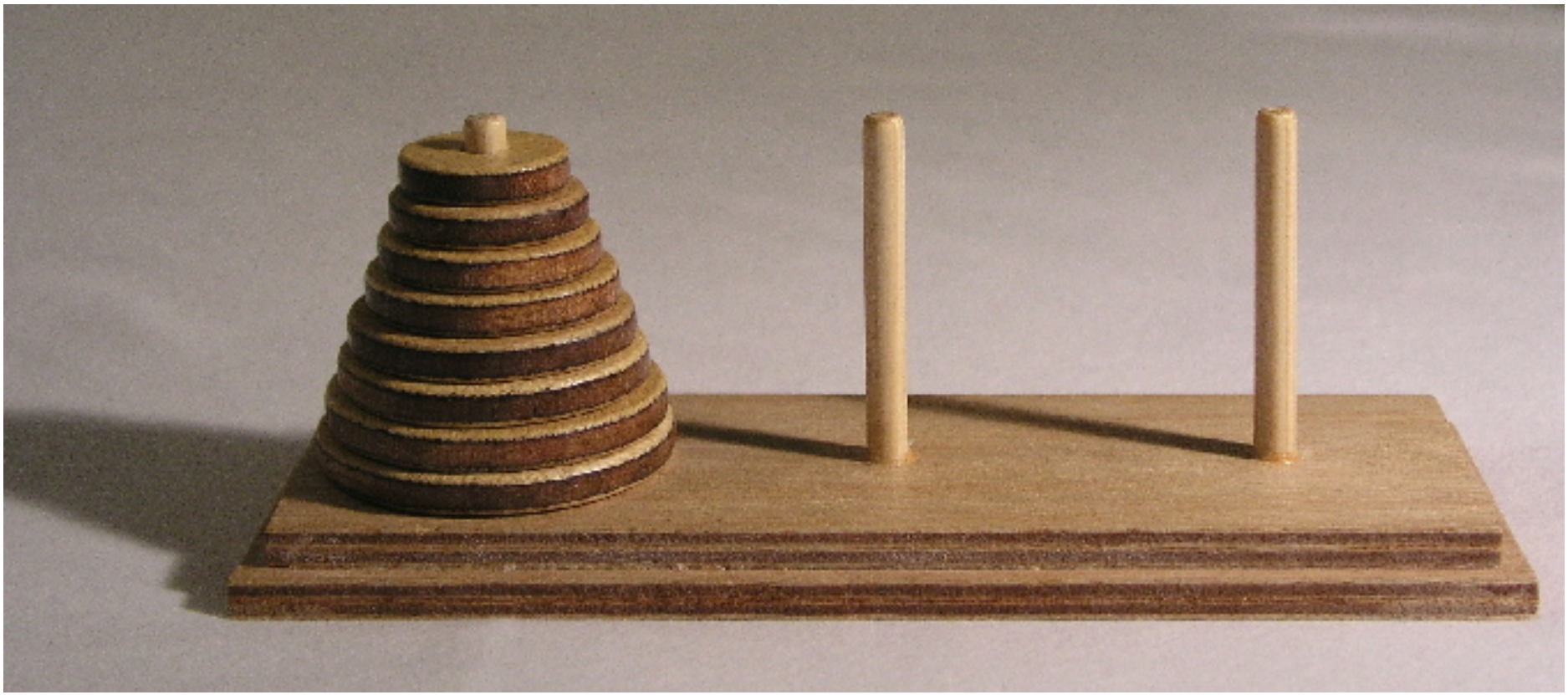
放苹果

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

int f(int m, int n) {
  if (m == 0) return 1;
  if (n <= 0) return 0;
  if (n > m)  return f(m, m);
  return f(m, n-1) + f(m-n, n);
}

int main() {
  int t,m,n;
  cin >> t;
  while (t--) {
    cin >> m >> n;
    cout << f(m, n) << endl;
  }
  return 0;
}
```

汉诺塔问题



汉诺塔问题

```c
#include <stdio.h>

//将编号为 numdisk 的盘子从 init 杆移至 desti 杆
void moveOne(int numDisk, char init, char desti)   {
  printf("Move disk No.%d from %c to %c.\n", numDisk, init, desti);
}
//将 numDisks 个盘子从 init 杆借助 temp 杆移至 desti 杆
void move(int numDisks, char init, char temp, char desti) {
  if (numDisks == 1) {
    moveOne(1, init, desti);
  } else {
    //首先将上面的 (numDisk-1) 个盘子从 init 杆借助 desti 杆移至 temp 杆
    move(numDisks-1, init, desti, temp);
    //然后将编号为 numDisks 的盘子从 init 杆移至 desti 杆
    moveOne(numDisks, init, desti);
    //最后将上面的 (numDisks-1) 个盘子从 temp 杆借助 init 杆移至 desti 杆
    move(numDisks-1, temp, init, desti);
  }
}

int main()  {
  move(3, 'A', 'B', 'C');
  return 0;
}
```

算 24

- 描述
  出 4 个小于 10 个正整数，你可以使用加减乘除 4 种运算以及括号把这 4 个数连接起来得到一个表达式。现在的问题是，是否存在一种方式使得得到的表达式的结果等于 24。
  这里加减乘除以及括号的运算结果和运算的优先级跟我们平常的定义一致(这里的除法定义是实数除法)。
  比如，对于 5，5，5，1，我们知道 5 * (5 −1 / 5) = 24，因此可以得到 24。又比如，对于 1，1，4，2，我们怎么都不能得到 24。

算 24

- 输入
  输入数据包括多行，每行给出一组测试数据，包括 4 个小于 10 个正整数。最后一组测试数据中包括 4 个 0，表示输入的结束，这组数据不用处理。
- 输出
  对于每一组测试数据，输出一行，如果可以得到 24，输出“YES”；否则，输出“NO”。
- 样例输入
  5 5 5 1
  1 1 4 2
  0 0 0 0
- 样例输出
  YES
  NO

算 24

n 个数算 24，必有两个数要先算。这两个数算的结果，和剩余 n-2 个数，就构成了 n-1 个数求 24 的问题

算 24

n 个数算 24，必有两个数要先算。这两个数算的结果，和剩余 n-2 个数，就构成了 n-1 个数求 24 的问题

枚举先算的两个数，以及这两个数的运算方式。

算 24

n 个数算 24，必有两个数要先算。这两个数算的结果，和剩余 n-2 个数，就构成了 n-1 个数求 24 的问题

枚举先算的两个数，以及这两个数的运算方式。

边界条件?

算 24

n 个数算 24，必有两个数要先算。这两个数算的结果，和剩余 n-2 个数，就构成了 n-1 个数求 24 的问题

枚举先算的两个数，以及这两个数的运算方式。

边界条件?

注意：浮点数比较是否相等，不能用 ==

算 24

```
1  #include <stdio.h>
2  #include <math.h>
3  double a[5];
4  int count24(double a[], int);
5  int isZero(double x) {
6    return fabs(x) <= 1e-6;
7  }
8
9  int main() {
10   int i;
11   while(1) {
12     for (i = 0;i < 4; i++) scanf("%lf", &a[i]);
13     if (isZero(a[0])) break;
14     if (count24(a, 4))  printf("YES\n");
15     else  printf("NO\n");
16   }
17   return 0;
18 }
```

算 24

```
19  int count24(double a[], int n) {//用数组 a 里的 n 个数，计算 24
20    if (n == 1) {
21      if (isZero( a[0] - 24)) return 1;
22      else  return 0;
23    }
24    double b[5];
25    int i, j, k;
26    for(i = 0;i < n-1; i++) {
27      for(j = i+1; j < n; j++) { //枚举两个数的组合
28        int m = 0; //还剩下 m 个数，m = n - 2
29        for(k = 0; k < n; k++) {
30          if( k != i && k!= j)  b[m++] = a[k];//把其余数放入 b
31        }
```

算 24

```
32          b[m] = a[i]+a[j];
33          if(count24(b, m+1)) return 1;
34          b[m] = a[i]-a[j];
35          if(count24(b, m+1)) return 1;
36          b[m] = a[j]-a[i];
37          if(count24(b,m+1))  return 1;
38          b[m] = a[i]*a[j];
39          if(count24(b,m+1))  return 1;
40          if (!isZero(a[j])) {
41            b[m] = a[i]/a[j];
42            if(count24(b,m+1))  return 1;
43          }
44          if( !isZero(a[i])) {
45            b[m] = a[j]/a[i];
46            if(count24(b,m+1))  return 1;
47          }
48        }
49      }
50      return 0;
51    }
```

用递归替代多重循环

n 皇后问题：输入整数 n, 要求 n 个国际象棋的皇后，摆在 n*n 的棋盘上，互相不能攻击，输出全部方案。

用递归替代多重循环

n 皇后问题：输入整数 n, 要求 n 个国际象棋的皇后，摆在 n*n 的棋盘上，互相不能攻击，输出全部方案。
后的走法: 横、直、斜 45 度都可以走，步数不受限制

用递归替代多重循环

n 皇后问题：输入整数 n, 要求 n 个国际象棋的皇后，摆在 n*n 的棋盘上，互相不能攻击，输出全部方案。
后的走法: 横、直、斜 45 度都可以走，步数不受限制
互相不能攻击: 不能在同一行, 同一列, 斜 45 方向

直观的想法:8 重循环。

用递归替代多重循环

n 皇后问题：输入整数 n, 要求 n 个国际象棋的皇后，摆在 n*n 的棋盘上，互相不能攻击，输出全部方案。
后的走法: 横、直、斜 45 度都可以走，步数不受限制
互相不能攻击: 不能在同一行, 同一列, 斜 45 方向

直观的想法:8 重循环。
n 皇后，n 重循环?

N 皇后问题

- 输入一个正整数 N，则程序输出 N 皇后问题的全部摆法。
- 输出结果里的每一行都代表一种摆法。行里的第 i 个数字如果是 n，就代表第 i 行的皇后应该放在第 n 列。皇后的行、列编号都是从 1 开始算。
- 样例输入：
  4
- 样例输出:
  2 4 1 3
  3 1 4 2

N 皇后问题

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int N;
4  int queenPos[100]; //用来存放算好的皇后位置。最左上角是 (0,0)
5  void nQueen(int n);
6
7  int main(){
8    scanf("%d", &N);
9    nQueen(0); //从第 0 行开始摆皇后
10   return 0;
11 }
```

N 皇后问题

```
12  void nQueen(int n) { //在 0~n-1 行皇后已经摆好的情况下，摆 n 行及其后的皇后
13    int i, j;
14    if (n == N) { // N 个皇后已经摆好
15      for(i = 0; i < n; i++)
16        printf("%d ", queenPos[i] + 1);
17      printf("\n");
18      return ;
19    }
20    for (i = 0; i < N; i++) { //逐尝试第 n 个皇后的位置
21      for (j = 0; j < n; j++) {
22        //和已经摆好的 n 个皇后的位置比较，看是否冲突
23        if (queenPos[j] == i || abs(queenPos[j] - i) == abs(n - j)) {
24          break; //冲突，则试下一个位置
25        }
26      }
27      if(j == n) { //当前选的位置 i 不冲突
28        queenPos[n] = i; //将第 n 个皇后摆放在位置 i
29        nQueen(n+1);
30      }
31    }
32  }
```

输出整数 1-N 的全排列

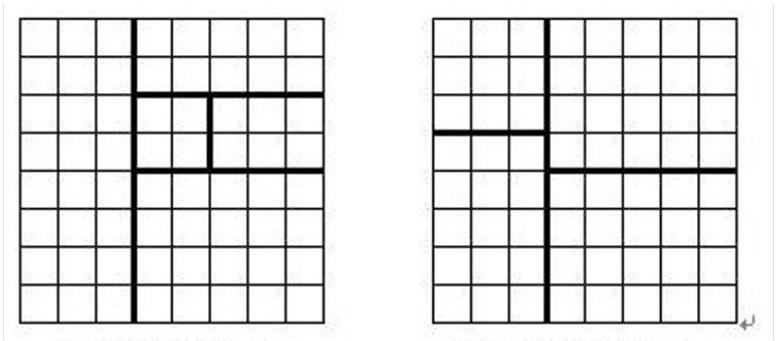
```cpp
#include <iostream>
using namespace std;
int result[200];
int N;
int used[200] = {0};
void Permutation(int k) {
  if (k == N) {
    for( int i = 0;i < N; ++ i)
    cout << result[i] << " " ;
    cout << endl;
    return ;
  }
  for (int i = 1;i <= N; ++ i) {
    if (!used[i]) {
      result[k] = i;
      used[i] = 1;
      Permutation(k+1);
      used[i] = 0;
    }
  }
}
int main() {
  cin >> N;
  Permutation(0);
  return 0;
}
```

求全排列

```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4  template <class T>
5  void Perm(T s, T e, T k) {//输出 [s,k) 固定情况下的全排列
6    if (k == e) { // [s,e) 的一个全排列已经生成
7      for(s; s != e; ++s)
8        cout << *s << " ";
9      cout << endl;
10     return;
11   }
12   for (T i = k ; i != e; ++i) {
13     swap(*k, *i);
14     Perm(s, e, k+1);
15     swap(*k, *i);
16   }
17 }
18 int main() {
19   char a[] = "abcd";
20   Perm(a, a+4, a);
21   return 0;
22 }
23
```

例题: 棋盘分割

将一个 8*8 的棋盘进行如下分割：将原棋盘割下一块矩形小棋盘并使剩下部分也是矩形，再将剩下的部分继续如此分割，这样割了 (n-1) 次后，连同最后剩下的矩形小棋盘共有 n 块矩形小棋盘。(每次切割都只能沿着棋盘格子的边进行)



允许的分割方案　　不允许的分割方案

原棋盘上每一格有一个分值，一块矩形小棋盘的总分为其所含各格分值之和。现在需要把原棋盘按上述规则分割成 n 块矩形小棋盘，并使各矩形小棋盘总分的均方差最小。

均方差 $\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\overline{x})^2}{n}}$，其中平均值 $\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$，$x_i$ 为第 i 块矩形小棋盘的总分.

例题: 棋盘分割

输入
第 1 行为一个整数 n $(1 < n < 15)$
第 2 行至第 9 行每行为 8 个小于 100 的非负整数，表示棋盘上相应格子的分值。每行相邻两数之间用一个空格分隔

输出
仅一个数，为 σ（四舍五入精确到小数点后三位）

样例输入

```
3
1 1 1 1 1 1 1 3
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 0 3
```

样例输出

```
1.633
```

例题: 棋盘分割

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}}$$

其中,

$$\begin{aligned}
\sum_{i=1}^{n}(x_i - \overline{x})^2 &= \sum_{i=1}^{n}({x_i}^2 - 2x_i\overline{x} + \overline{x}^2) \\
&= \sum_{i=1}^{n}{x_i}^2 - 2\sum_{i=1}^{n}x_i\overline{x} + n\overline{x}^2 \\
&= \sum_{i=1}^{n}{x_i}^2 - 2n\overline{x}^2 + n\overline{x}^2 \\
&= \sum_{i=1}^{n}{x_i}^2 - n\overline{x}^2
\end{aligned}$$

若要求出最小均方差，只需要求出最小的 $\sum_{i=1}^{n}{x_i}^2$

例题: 棋盘分割

设 $fun(n, x_1, y_1, x_2, y_2)$ 为以 (x_1, y_1) 为左上角，(x_2, y_2) 为右下角的棋盘分割成 n 份后的最小平方和。
那么，

$$
\begin{aligned}
&fun(n, x_1, y_1, x_2, y_2) \\
&\qquad = \min \{ \\
&\qquad\qquad \min_{i=x_1}^{x_2-1} \{fun(n-1, x_1, y_1, i, y_2) + fun(1, i+1, y_1, x_2, y_2)\}, \\
&\qquad\qquad \min_{i=x_1}^{x_2-1} \{fun(1, x_1, y_1, i, y_2) + fun(n-1, i+1, y_1, x_2, y_2)\}, \\
&\qquad\qquad \min_{i=y_1}^{y_2-1} \{fun(n-1, x_1, y_1, x_2, i) + fun(1, x_1, i+1, x_2, y_2)\}, \\
&\qquad\qquad \min_{i=y_1}^{y_2-1} \{fun(1, x_1, y_1, x_2, i) + fun(n-1, x_1, i+1, x_2, y_2)\} \\
&\qquad \}
\end{aligned}
$$

其中 $fun(1, x_1, y_1, x_2, y_2)$ 等于该棋盘内分数和的平方
且当 $x_2 - x_1 + y_2 - y_1 < n - 1$ 时，$fun(n, x_1, y_1, x_2, y_2) = +\infty$

例题: 棋盘分割

当 $x_2 - x_1 + y_2 - y_1 < n - 1$ 时，$fun(n, x_1, y_1, x_2, y_2) = +\infty$ 为什么?

例题: 棋盘分割

当 $x_2 - x_1 + y_2 - y_1 < n - 1$ 时，$fun(n, x_1, y_1, x_2, y_2) = +\infty$ 为什么?
要分成 n 块，即要切 $n - 1$ 刀，

例题: 棋盘分割

当 $x_2 - x_1 + y_2 - y_1 < n - 1$ 时，$fun(n, x_1, y_1, x_2, y_2) = +\infty$ 为什么?
要分成 n 块，即要切 $n - 1$ 刀，
X 方向最多可切 $x_2 - x_1$ 刀，Y 方向最多可切 $y_2 - y_1$ 刀

例题: 棋盘分割

当 $x_2 - x_1 + y_2 - y_1 < n - 1$ 时，$fun(n, x_1, y_1, x_2, y_2) = +\infty$ 为什么?
要分成 n 块，即要切 $n - 1$ 刀，
X 方向最多可切 $x_2 - x_1$ 刀，Y 方向最多可切 $y_2 - y_1$ 刀
则必须 $x_2 - x_1 + y_2 - y_1 \geq n - 1$

例题: 棋盘分割

当 $x_2 - x_1 + y_2 - y_1 < n - 1$ 时，$fun(n, x_1, y_1, x_2, y_2) = +\infty$ 为什么?
要分成 n 块，即要切 $n - 1$ 刀，
X 方向最多可切 $x_2 - x_1$ 刀，Y 方向最多可切 $y_2 - y_1$ 刀
则必须 $x_2 - x_1 + y_2 - y_1 \geq n - 1$

那么 $x_2 - x_1 + y_2 - y_1 < n - 1$ 时无法再分

例题: 棋盘分割

对于对于某个 $fun(n, x_1, y_1, x_2, y_2)$ 来说，可能使用多次这个值，所以每次都计算太消耗时间！

例题: 棋盘分割

对于对于某个 $fun(n, x_1, y_1, x_2, y_2)$ 来说，可能使用多次这个值，所以每次都计算太消耗时间!
解决办法：记录表

- 用 $val[n][x_1][y_1][x_2][y_2]$ 来记录 $fun(n, x_1, y_1, x_2, y_2)$
- val 初始值统一为 -1
- 当需要使用 $fun(n, x_1, y_1, x_2, y_2)$ 时，查看 $val[n][x_1][y_1][x_2][y_2]$
  - 如果为 -1，那么计算 $fun(n, x_1, y_1, x_2, y_2)$，并保存于 $val[n][x_1][y_1][x_2][y_2]$
  - 如果不为 -1，直接返回 $val[n][x_1][y_1][x_2][y_2]$

例题: 棋盘分割

```cpp
#include <iostream>
#include <cmath>
#include <cstring>
#include <iomanip>
using namespace std;
int sum[9][9]; //sum[i][j] 是从 (1,1) 到 (i,j) 的总分
int result[16][9][9][9][9];
int smallSum(int x1,int y1,int x2,int y2) { //求 (x1,y1) 到 (x2,y2) 的总分
  return sum[x2][y2] - sum[x1-1][y2] - sum[x2][y1-1] + sum[x1-1][y1-1];
}
int fun(int n,int x1,int y1,int x2,int y2);
int main() {
  int n;
  memset(sum,0,sizeof(sum));
  memset(result,0xff,sizeof(result));
  cin >> n;
  for(int i = 1;i <= 8; ++i) {
    int rowSum = 0;
    for(int j = 1; j <= 8; ++j) {
      int s;
      cin >> s;
      rowSum += s;
      sum[i][j] = sum[i-1][j] + rowSum;
    }
  }
  int squareSum = fun(n,1,1,8,8);
  double result = n * squareSum - sum[8][8] * sum[8][8];
  cout << setiosflags(ios::fixed) << setprecision(3) << sqrt(result/(n*n));
  return 0;
}
```

例题: 棋盘分割

```
31
32  int fun(int n,int x1,int y1,int x2,int y2){
33  //求将 (x1,y1)-(x2,y2) 分割成 n 块, 所能的到的 n 个块的分数的平方之和的最小值
34    int v = 1 << 30;
35    if( n == 1) {
36      int t =  smallSum(x1,y1,x2,y2);
37      return t * t;
38    }
39    if (x2-x1+y2-y1 < n-1) return v;
40    if (result[n][x1][y1][x2][y2] != -1) return result[n][x1][y1][x2][y2];
41    for (int x = x1; x < x2; ++x) {
42      int left = smallSum(x1,y1,x,y2);
43      int right = smallSum(x+1,y1,x2,y2);
44      int t = min(fun(n-1,x+1,y1,x2,y2) + left*left,
45                  fun(n-1,x1,y1,x,y2) + right*right);
46      v = min(t,v);
47    }
48    for (int y = y1; y < y2; ++y) {
49      int up = smallSum(x1,y1,x2,y);
50      int down = smallSum(x1,y+1,x2,y2);
51      int t = min(fun(n-1,x1,y+1,x2,y2) + up*up,
52                  fun(n-1,x1,y1,x2,y) + down*down);
53      v = min(t,v);
54    }
55    result[n][x1][y1][x2][y2] = v;
56    return v;
57  }
```

用栈替代递归-放苹果

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

int f(int m, int n) {
  if (m == 0) return 1;
  if (n <= 0) return 0;
  if (n > m)  return f(m, m);//1
  int tmp = f(m, n-1);//2
  return tmp + f(m-n, n);//3
}

int main() {
  int t,m,n;
  cin >> t;
  while (t--) {
    cin >> m >> n;
    cout << f(m, n) << endl;
  }
  return 0;
}
```

用栈替代递归-放苹果

```
1  struct Node {  //栈中要放的东西
2    int m,n;
3    int tmp;  //局部变量 tmp
4    int retAdr; //返回地址，为 0 则说明还没开始算
5    Node() { }
6    Node(int m_,int n_,int adr):m(m_),n(n_),retAdr(adr){ }
7  };
8  int ff(int m,int n) {
9    Node nd;
10   int retVal; //某层 StkWays(i,j) 的最终结果
11   stack<Node> stk;
12   stk.push(Node(m,n,0)); //要计算 Ways(m,n)
13   while(!stk.empty()) {
14     nd = stk.top();
15     if (nd.m == 0) {
16       retVal = 1;
17       stk.pop();
18       continue;
19     }
20     if (nd.n <= 0) {
21       retVal = 0;
22       stk.pop();
23       continue;
24     }
```

用栈替代递归-放苹果

```
25      if( nd.retAdr == 0) { //还没开始算
26        if( nd.n > nd.m) {
27          stk.top().retAdr = 1;
28          stk.push(Node(nd.m,nd.m,0));
29          continue;
30        }
31        stk.top().retAdr = 2;
32        stk.push(Node(nd.m,nd.n-1,0));
33      } else if (nd.retAdr == 1) {
34        stk.pop();
35      } else if(nd.retAdr == 2) {
36        stk.top().tmp = retVal;
37        stk.top().retAdr = 3;
38        stk.push(Node(nd.m-nd.n,nd.n,0));
39      } else if(nd.retAdr == 3) {
40        retVal += nd.tmp;
41        stk.pop();
42      }
43    }
44    return retVal;
45  }
46
```

用栈替代递归-汉诺塔问题

```c
#include <stdio.h>

//将编号为 numdisk 的盘子从 init 杆移至 desti 杆
void moveOne(char init, char desti)   {
  printf("Move disk from %c to %c.\n", init, desti);
}
//将 numDisks 个盘子从 init 杆借助 temp 杆移至 desti 杆
void move(int numDisks, char init, char temp, char desti) {
  if (numDisks == 1) {
    moveOne(init, desti);
  } else {
    //首先将上面的 (numDisk-1) 个盘子从 init 杆借助 desti 杆移至 temp 杆
    move(numDisks-1, init, desti, temp);
    //然后将编号为 numDisks 的盘子从 init 杆移至 desti 杆
    moveOne(init, desti);
    //最后将上面的 (numDisks-1) 个盘子从 temp 杆借助 init 杆移至 desti 杆
    move(numDisks-1, temp, init, desti);
  }
}

int main()  {
  move(3, 'A', 'B', 'C');
  return 0;
}
```

用栈替代递归-汉诺塔问题

```cpp
#include <iostream>
#include <stack>
using namespace std;
struct Problem   {
  int numDisks;
  char init,temp,desti;
  Problem(int n, char i,char t,char d) :numDisks(n),init(i),temp(t),desti(d)  {   }
};
//将编号为 numdisk 的盘子从 init 杆移至 desti 杆
void moveOne(char init, char desti)   {
  printf("Move disk from %c to %c.\n", init, desti);
}
stack<Problem> stk;//用来模拟信封堆的栈，一个元素代表一个信封
int main()   {
  stk.push(Problem(3,'A','B','C')); //初始化了第一个信封
  while (!stk.empty()) { //只要还有信封，就继续处理
    Problem curPrb = stk.top(); //取最上面的信封，即当前问题
    stk.pop();  // 丢弃最上面的信封
    if (curPrb.numDisks == 1)   moveOne(curPrb.init, curPrb.desti);
    else {  //分解子问题
      //先把分解得到的第 3 个子问题放入栈中
      stk.push(Problem(curPrb.numDisks - 1,curPrb.temp,curPrb.init,curPrb.desti));
      //再把第 2 个子问题放入栈中
      stk.push(Problem(1,curPrb.init,curPrb.temp,curPrb.desti));
      //最后放第 1 个子问题，后放入栈的子问题先被处理
      stk.push(Problem(curPrb.numDisks - 1,curPrb.init,curPrb.desti,curPrb.temp));
    }
  }
  return 0;
}

```