# 《计算概论A》课程 程序设计部分

# 函数的递归调用 (4)


李　戈

北京大学 信息科学技术学院 软件研究所

**lige@sei.pku.edu.cn**


北京大学

# 递归问题解法小结

- 面对一个问题时：
  - ① 从简单情况开始分析问题，找出解决问题的规律；
  - ② 总结并抽取出解决方案中**反复做的事情**；
  - ③ 用一个**函数（原型）**来描述这件**反复做的事情**；
  - ④ 假设反复做的事情已经由上述函数实现，写出如何利用上述函数解决整体问题的步骤；
  - ⑤ 分析并写出边界条件；



# 题目 - 判断字符串是否为回文

## 描述

编程，输入一个字符串，输出该字符串是否回文。

## 关于输入

输入为一行字符串（字符串中没有空白字符，字符串长度不超过100）。

## 关于输出

如果字符串是回文，输出yes；否则，输出no。

## 例子输入

```
abcdedcba
```

## 例子输出

```
yes
```

## 提示

回文是指顺读和倒读都一样的字符串。

```
bool huiwen(int n)
{


    else if(str[n-1] == str[len-n]) return huiwen(n-2);
    else return false;
}
```



# 题目 - 布尔表达式

## 描述

输入一个布尔表达式，请你输出它的真假值。
比如：( V | V ) & F & ( F | V )
V表示true，F表示false，&表示与，|表示或，!表示非。
上式的结果是F

## 关于输入

输入包含多行，每行一个布尔表达式，表达式中可以有空格，总长度不超过1000

## 关于输出

对每行输入，如果表达式为真，输出"V",否则出来"F"

## 例子输入

```
( V | V ) & F & ( F| V)
!V | V & V & !F & (F | V ) & (!F | F | !V & V)
(F&F|V|!V&!F&!(F|F&V))
```

## 例子输出

```
F
V
V
```

```
#include <iostream>
using namespace std;
char str[1001] = {0};

int main()
{
    while(cin.getline(str,1000))
    {
        delblank();
        if(boolstr(0,strlen(str))-1) cout<<'F';
        else cout<<'V';
        cout<<endl;
    }
    return 0;
}
```

```
bool boolstr(int m, int n)
{
    int prebracket = n;
    for(int i = m; i < n; i++ )
    {


                    return (boolstr(m,i-1)||boolstr(i+1, n));
        }
    }
    for(int i = m; i < n; i++ )
    {


                    return boolstr(m,i-1)&&boolstr(i+1, n);
        }
    }




    if(str[m]=='F') return false;
    if(str[m]=='V') return true;
}
```

# 快速排序

# 快速排序

- 问题：
  - 设计程序对整数数组中的数进行由小到大的排序；
- 算法优劣的标准
  - 时间代价
  - 空间代价
- 思考：
  - 排序程序中，时间代价、空间代价花费在哪里？

| | i=1 | i=2 | i=3 | i=4 | i=5 | i=6 |
|---|---|---|---|---|---|---|
| | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] |
| 初始值 | 1 | 8 | 3 | 2 | 4 | 9 |
| 1<8；1,8互换 | 1 ⟷ | 8 | 3 | 2 | 4 | 9 |
| 1<3；1,3互换 | 8 | 1 ⟷ | 3 | 2 | 4 | 9 |
| 1<2；1,2互换 | 8 | 3 | 1 ⟷ | 2 | 4 | 9 |
| 1<4；1,4互换 | 8 | 3 | 2 | 1 ⟷ | 4 | 9 |
| 1<9；1,9互换 | 8 | 3 | 2 | 4 | 1 ⟷ | 9 |
| 1到达位置 | 8 | 3 | 2 | 4 | 9 | (1) |
| 8>3;顺序不动 | 8 | 3 | 2 | 4 | 9 | 1 |
| 3>2;顺序不动 | 8 | 3 | 2 | 4 | 9 | 1 |
| 2<4；2,4互换 | 8 | 3 | 2 ⟷ | 4 | 9 | 1 |
| 2<9；2,9互换 | 8 | 3 | 4 | 2 ⟷ | 9 | 1 |
| 2到达位置 | 8 | 3 | 4 | 9 | (2) | 1 |

# 快速排序

- 思路：
  - 假设你的手里有一把牌：要尽可能少移动牌的位置；尽可能少的进行比较；

| 3 | 2 | 4 | 7 | 5 | 1 | 6 |
|---|---|---|---|---|---|---|

# 快速排序

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 3 | 2 | 4 | 7 | 5 | 1 | 6 |
| 3 | | 2 | 4 | 7 | 5 | 1 | 6 |
| 3 | 1 | 2 | 4 | 7 | 5 | | 6 |
| 3 | 1 | 2 | | 7 | 5 | 4 | 6 |
| | 1 | 2 | 3 | 7 | 5 | 4 | 6 |

# 快速排序

- 完成一趟排序：
  - 将数组第一个元素取出，作为分界值存放到k里。此时数组第一个元素位置空闲，用一个左探针L指示。
  - 设右探针R，从右往左走，寻找小于k的数。找到则将该数存放进左探针L所指示的空闲位置，此时右探针R所指示的位置变为空闲位置。
  - 左探针L从左往右走，寻找大于k的数。找到则将该数放到右探针R所指示的空闲位置，此时左探针所指示的位置变为空闲位置。
  - 循环执行以上2步，直到左右探针相遇为止。左右探针相遇意味着找完所有的元素。它们相遇的地方就是k的位置。

| 3 | | 2 | 4 | 7 | 5 | 1 | 6 |
|---|---|---|---|---|---|---|---|

# 快速排序

## 快速排序(*QuickSort*)

Cooling

*本例演示只说明一次划分过程Partition。红色显示的元素表示待排序的无序区。*

R[ ]

Clear Start

*请输入待排序的记录数组R[low..high]*
*(数据之间用半角逗号隔开)*

# 快速排序

- 对数组是a[1]……a[n]，一趟快速排序的算法：
  1. 设置两个变量L、R，排序开始的时候L = 1，R = n；
  2. 以第一个数组元素作为关键数据，赋值给k，即k = a[1]；
  3. 从R开始向前搜索，即由后开始向前搜索（R = R-1），找到第一个小于k的值，两者交换；
  4. 从L开始向后搜索，即由前开始向后搜索（L = L+1），找到第一个大于k的值，两者交换；
  5. 重复第3、4步，直到L=R；
  6. 将选出的k归位，a[L] = k；（或a[R] = k；）

```
L = LP;        R = RP;        k = array[L];
                              // array[L]给了k，L处空缺；
do {
    while ((L < R) && (array[R] >= k))
            R = R - 1;      //找到右起第一个比k小的数
    if (L < R) {
            array[L] = array[R]; //array[R]送给array[L];
            L ++;
    }
    while ((L < R) && (array[L] <= k))
            L = L + 1;      //找到左起第一个比k大的数
    if (L < R) {
            array[R]=array[L];
            R --;
    }
} while (L != R);
array[L] = k;                       //将最初选出的数字归位
```

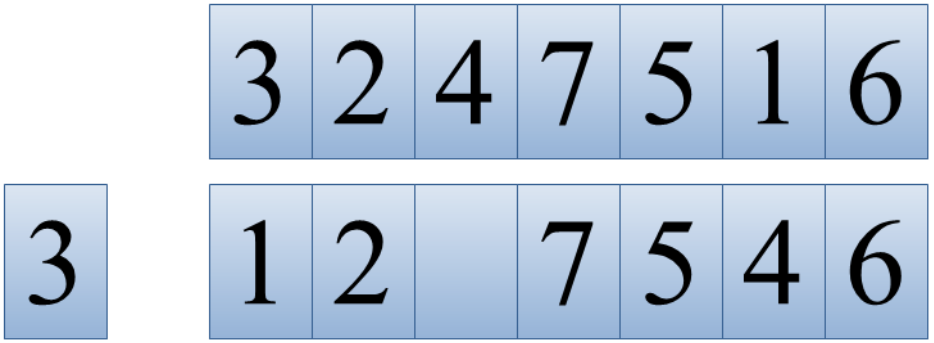
# 快速排序

- 如何完成全部排序？

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 3 | 2 | 4 | 7 | 5 | 1 | 6 |
| 3 | 1 | 2 | | 7 | 5 | 4 | 6 |

# 快速排序

- 如何完成全部排序？

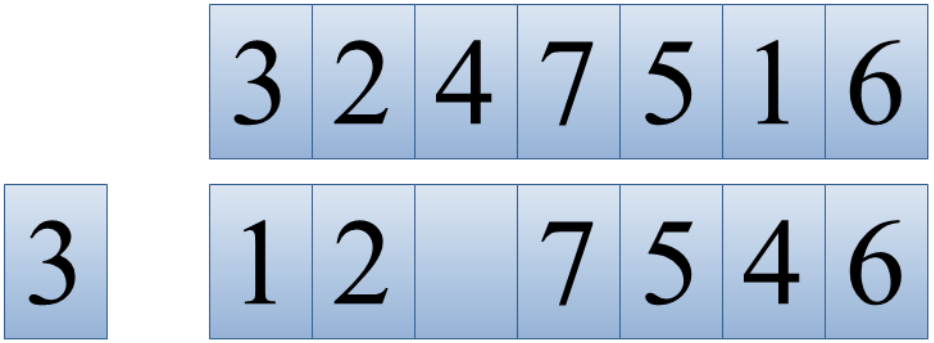



```
void sort(int LP, int RP)
{
        //排序
        sort(LP, L-1);
        sort(L + 1, RP);
}
```

北京大学

# 快速排序

- 如何完成全部排序？




```
void sort(int array[ ], int LP, int RP)
{
        //排序
        sort(array, LP, L-1);
        sort(array, L + 1, RP);
}
```

# 探索型递归问题的解法

- 特点：
  - 每一步所做动作相同；
- 重点：
  - 考虑第n步时做什么！

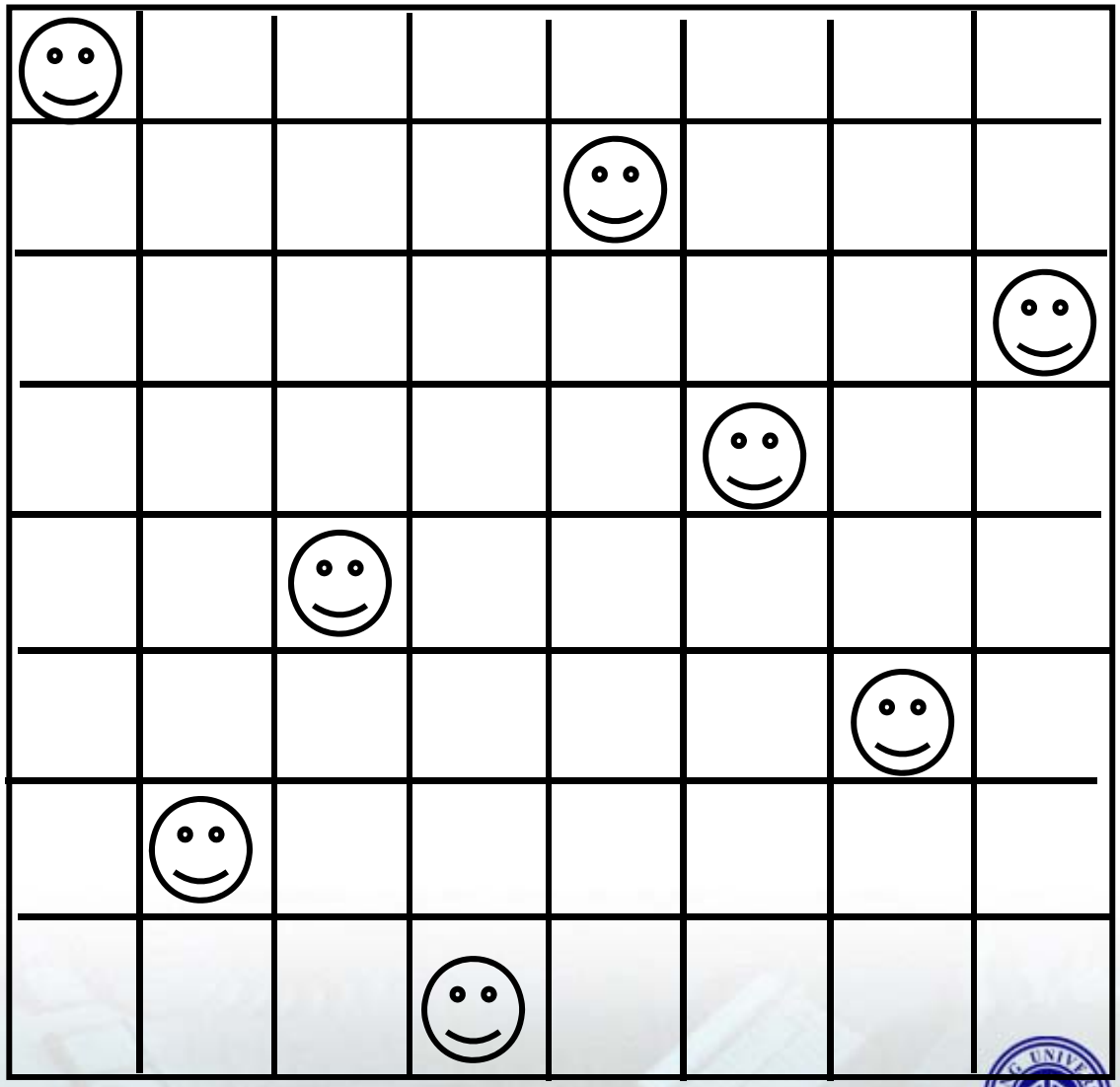
# 探索型递归问题的解法

- 第n步需要做什么？
  - 对于面前的每种选择
    - ① 把该做的事情做了！
    - ② 判定是否得到解！
    - ③ 递归（调用第n+1步）！
    - ④ 看是否需要回溯！

# 八皇后问题

- 背景：
  - 国际象棋的皇后可以走水平、垂直、斜线，若在一个皇后可以走动的范围内有其他棋子，则皇后可以吃掉这个棋子；
- 问题：
  - 如何在棋盘上摆放8个皇后，使得每个皇后都没有被吃掉的危险。
  - 换言之，当要摆放一个新的皇后时，摆放位置的行、列、左右对角线都不能有其它棋子。

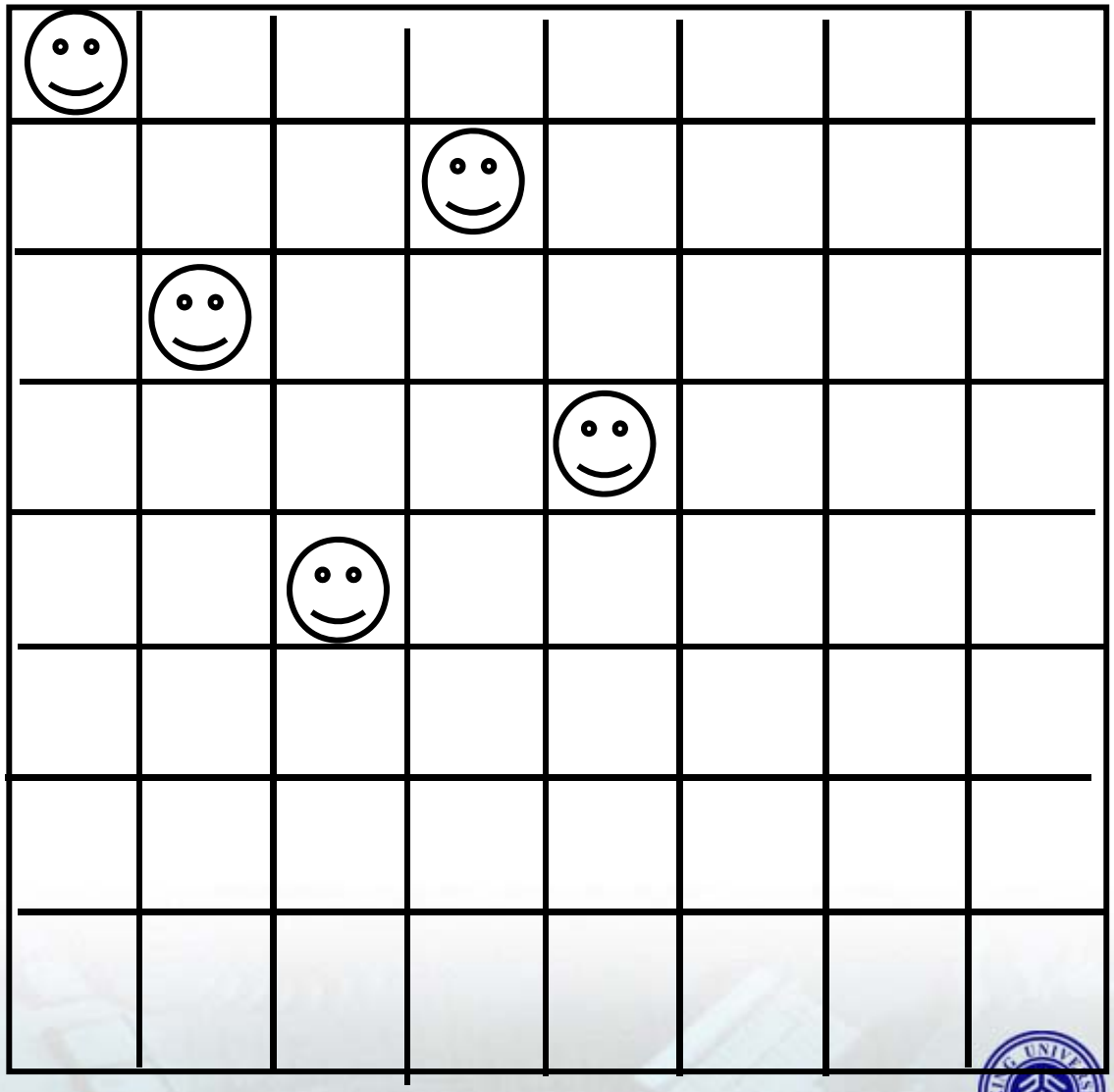
# 八皇后问题



成功

北京大学

# 八皇后问题



失败

北京大学

# 问题分析

- 递归函数要解决的问题
  - 在棋盘上放置棋子，
    - 第n个棋子放在第n列
  - 探测每一个可以放置皇后的位置
    - 若找到，进行放置的操作
      - 判定是否已经放置完毕；
      - 若完毕，打印输出；若未完，递归；
    - 递归回溯的操作；

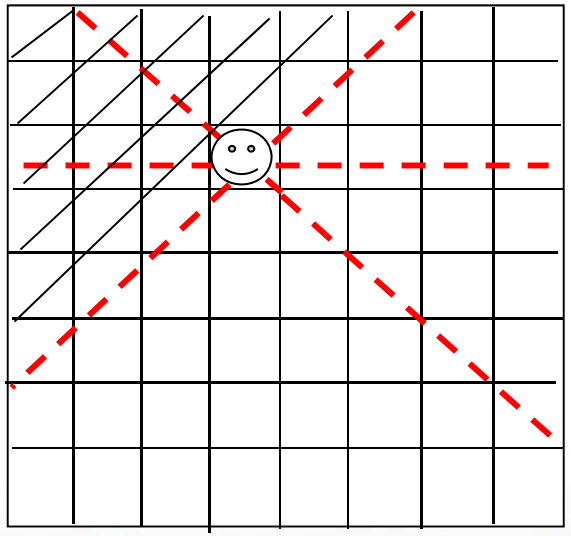
# 面临问题

- 如何表示棋盘上所有可以选择的放置点
  - 列已知，按行判断；
  - 如何表达对角线？

行列与对角线关系：

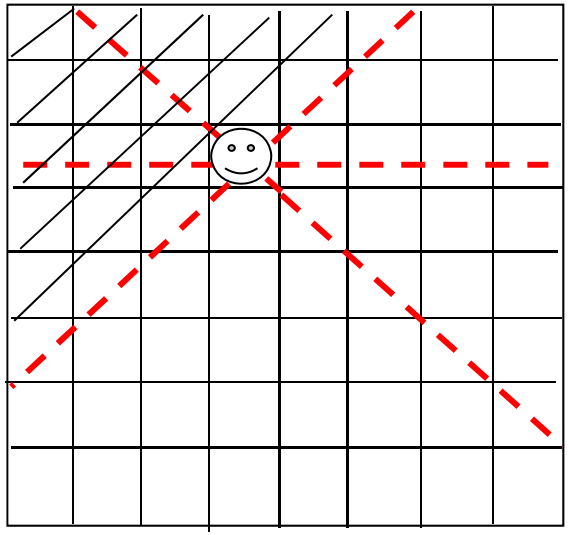
board[row][col]→left[row+col-1]

board[row][col]→right[8+row-col]



# 问题分析

- 每当一个位置可以满足摆放皇后条件时,令:
  - board[row][col]=1;
  - row[row]=0;
  - left[row+col-1]=0
  - right[8+row-col]=0;



# 题目 - 1090 分解因数

## 描述

给出一个正整数a，要求分解成若干个正整数的乘积，即a = a1 * a2 * a3 * ... * an，并且1 < a1 <= a2 <= a3 <= ... <= an，问这样的分解的种数有多少。注意到a = a也是一种分解。

## 关于输入

第1行是测试数据的组数n，后面跟着n行输入。每组测试数据占1行，包括一个正整数a (1 < a < 32768)

## 关于输出

n行，每行输出对应一个输入。输出应是一个正整数，指明满足要求的分解的种数

## 例子输入

```
2
2
20
```

## 例子输出

```
1
4
```

# 题目 - 习题(15-6) 走出迷宫

## 描述

当你站在一个迷宫里的时候，往往会被错综复杂的道路弄得失去方向感，如果你能得到迷宫地图，事情就会变得非常简单。
假设你已经得到了一个n*m的迷宫的图纸，请你找出从起点到出口的最短路。

## 关于输入

第一行是两个整数n和m(1<=n,m<=100)，表示迷宫的行数和列数。
接下来n行，每行一个长为m的字符串，表示整个迷宫的布局。字符'.'表示空地，'#'表示墙，'S'表示起点,'T'表示出口。

## 关于输出

输出从起点到出口最少需要走的步数。(你不能起出迷宫外)

## 例子输入

```
3 3
S#T
.#.
...
```

## 例子输出

```
6
```

# 好好想想,有没有问题?

# 谢　谢　！

北京大学