清华大学
Tsinghua University

程序设计基础

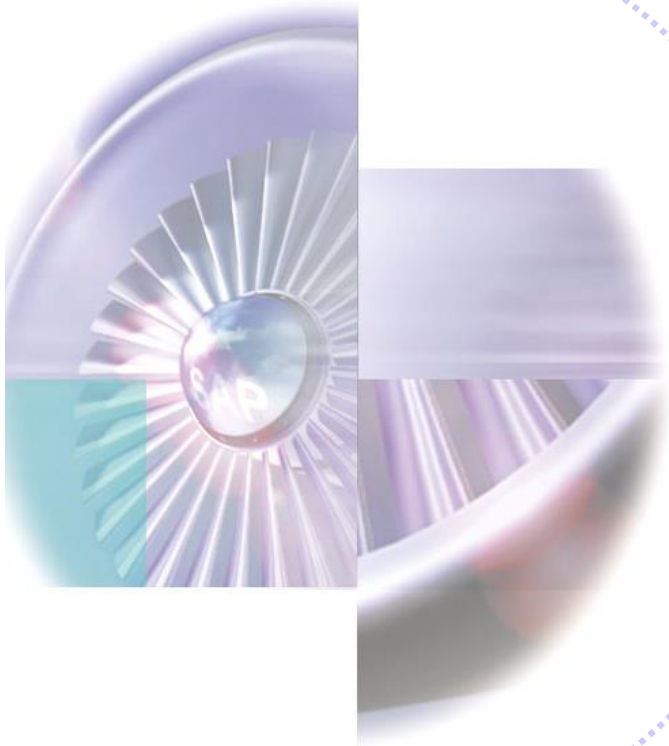
Fundamental of Programming

清华大学软件学院

刘玉身

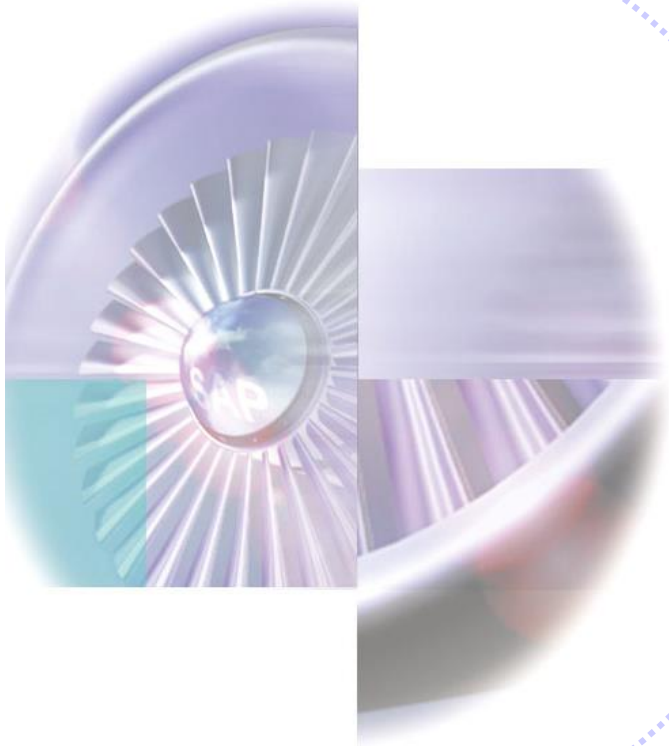
liuyushen@tsinghua.edu.cn

Lecture 8: 递归算法



1. 基本概念
2. 基于分治策略的递归
3. 基于回溯策略的递归

Lecture 8: 递归算法



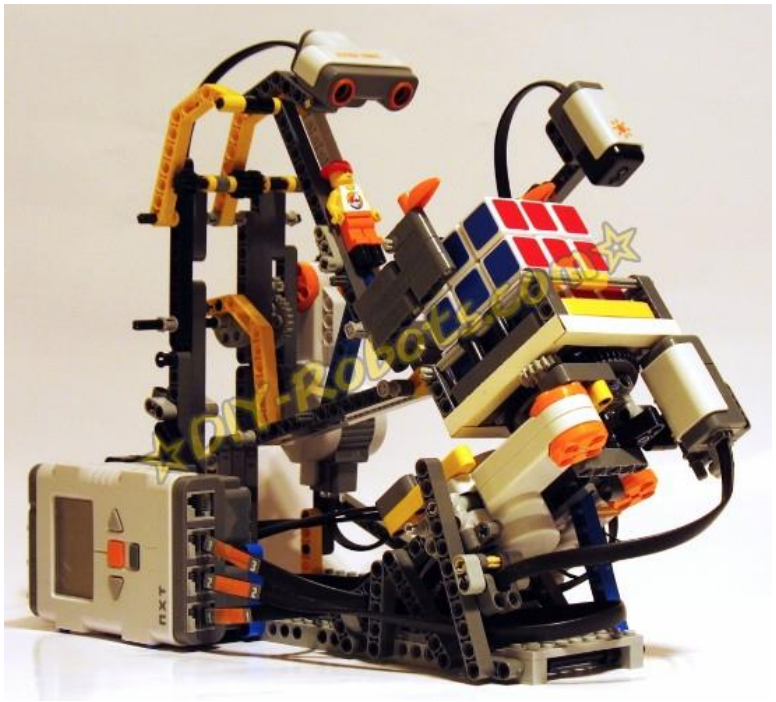
1. 基本概念
2. 基于分治策略的递归
3. 基于回溯策略的递归

递归与迭代

- 对于很多常用的递归都有简单、等价的迭代程序。究竟使用哪一种，凭你的经验选择。
- 迭代程序复杂，但效率高。
- 递归程序逻辑清晰，但往往效率较低。

在程序设计当中，有相当一类求一组解、或求全部解或求最优解的问题，不是根据某种确定的计算法则，而是利用试探和回溯（**Backtracking**）的搜索技术求解。回溯法也是设计递归算法的一种重要方法，它的求解过程实质上是一个先序遍历一棵“状态树”的过程，只不过这棵树不是预先建立的，而是隐含在遍历的过程当中。

华容道

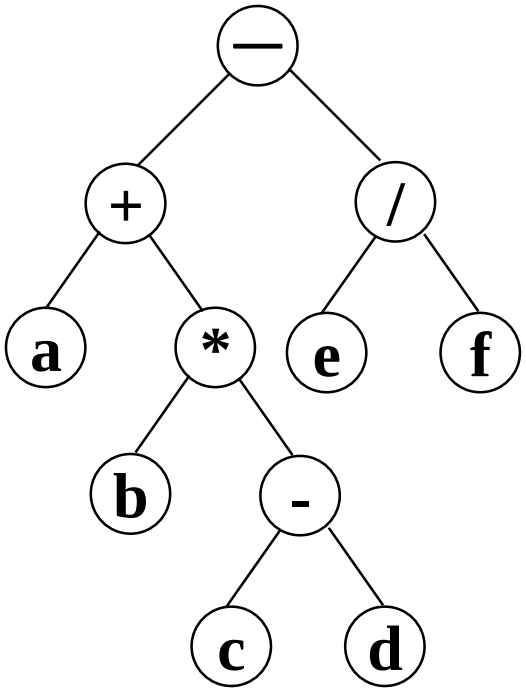


机器人解魔方

先序遍历

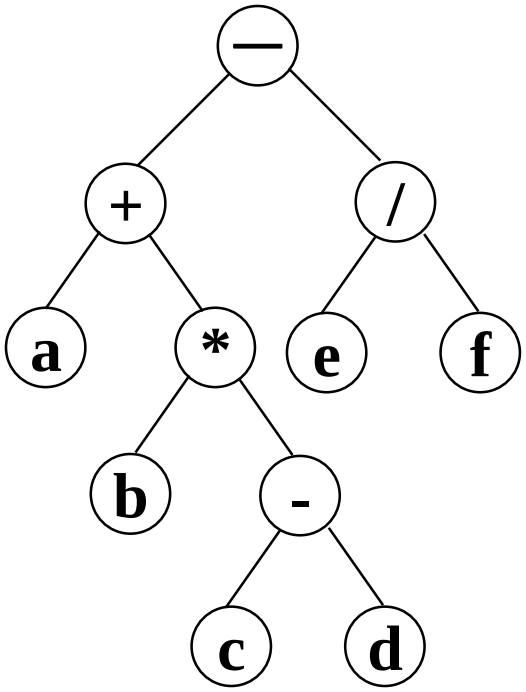
- 二叉树的遍历方式
  - 先序遍历(Preorder Traversal), or 先根/前序遍历
  - 中序遍历(Inorder)、后序遍历(Postorder)
- 先序遍历:
  - 先访问根节点，再访问左子树，然后是右子树
- 中序遍历:
  - 先访问左子树，然后是根节点，然后是右子树
- 后序遍历:
  - 先访问左子树，然后右子树，最后是根节点

- 先序序列(根-左-右)： - + a * b – c d / e f
- 中序序列(左-根-右)： a + b * c - d – e / f
- 后序序列(左-右-根)： a b c d - * + e f / -



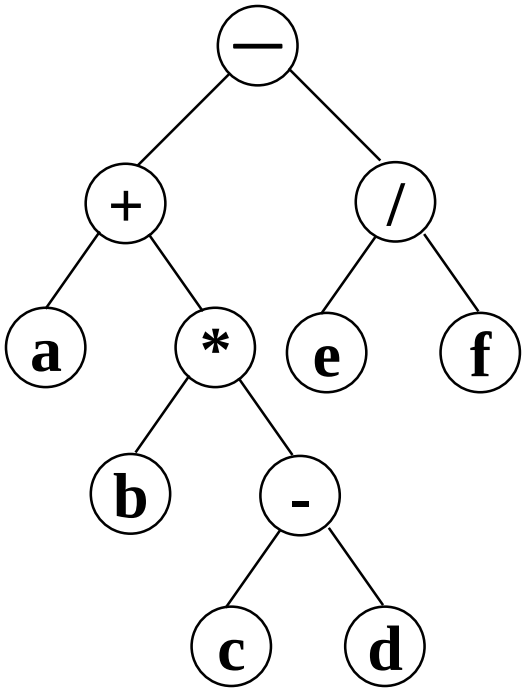
- 先序序列(根-左-右)： - + a * b – c d / e f
- 中序序列(左-根-右)： a + b * c - d – e / f
- 后序序列(左-右-根)： a b c d - * + e f / -

```
void pre_order_traversal(TreeNode *root) {
     // Do Something with root
     if (root->lchild != NULL)
         pre_order_traversal(root->lchild);
     if (root->rchild != NULL)
         pre_order_traversal(root->rchild);
```



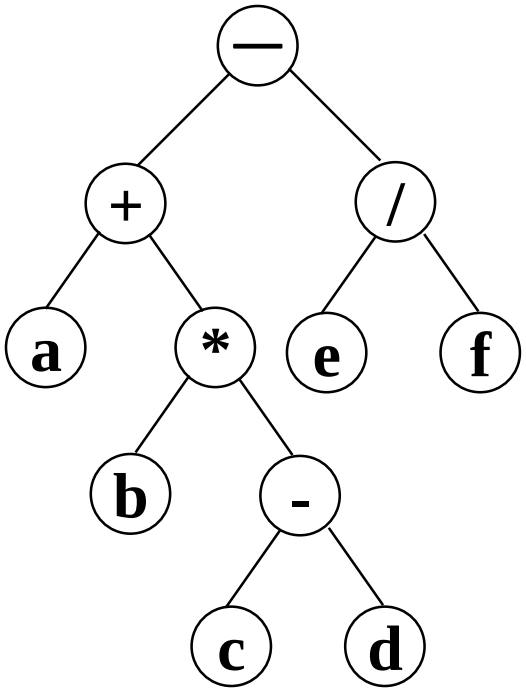
- 先序序列(根-左-右)： - + a * b – c d / e f
- 中序序列(左-根-右)： a + b * c - d – e / f
- 后序序列(左-右-根)： a b c d - * + e f / -

```
void in_order_traversal(TreeNode *root) {
     if (root->lchild != NULL)
         in_order_traversal(root->lchild);
     // Do Something with root
     if (root->rchild != NULL)
         in_order_traversal(root->rchild);
}
```



- 先序序列(根-左-右)： - + a * b – c d / e f
- 中序序列(左-根-右)： a + b * c - d – e / f
- 后序序列(左-右-根)： **a b c d - * + e f / -**

```
void post_order_traversal(TreeNode *root) {
     if (root->lchild != NULL)
         post_order_traversal(root->lchild);
     if (root->rchild != NULL)
         post_order_traversal(root->rchild);
     // Do Something with root
}
```



1. 分书问题

有五本书，它们的编号分别为1，2，3，4，5，现准备分给 A, B, C, D, E五个人，每个人的阅读兴趣用一个二维数组来加以描述：

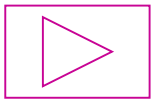
$$like[i][j] = \begin{cases} 1 & i\text{喜 欢 }j\text{书} \\ 0 & i\text{不 喜 欢 }j\text{书} \end{cases}$$

希望编写一个程序，输出所有的分书方案，让人人皆大欢喜。

假定这5个人对这5本书的阅读兴趣如下表：

书

人	1	2	3	4	5
A	0	0	1	1	0
B	1	1	0	0	1
C	0	1	1	0	1
D	0	0	0	1	0
E	0	1	0	0	1



不用递归？

一种方法：枚举法。

把所有可能出现的分书方案都枚举出来，然后逐一判断它们是否满足条件，即是否使得每个人都能够得到他所喜欢的书。

缺点：计算量大！

解题思路：

1、定义一个整型的二维数组，将上表中的阅读喜好用初始化的方法赋给这个二维数组。
可定义：

```
int Like[6][6] = {{0, 0, 0, 0, 0, 0},
                  {0, 0, 0, 1, 1, 0},
                  {0, 1, 1, 0, 0, 1},
                  {0, 0, 1, 1, 0, 1},
                  {0, 0, 0, 0, 1, 0},
                  {0, 0, 1, 0, 0, 1}};
```

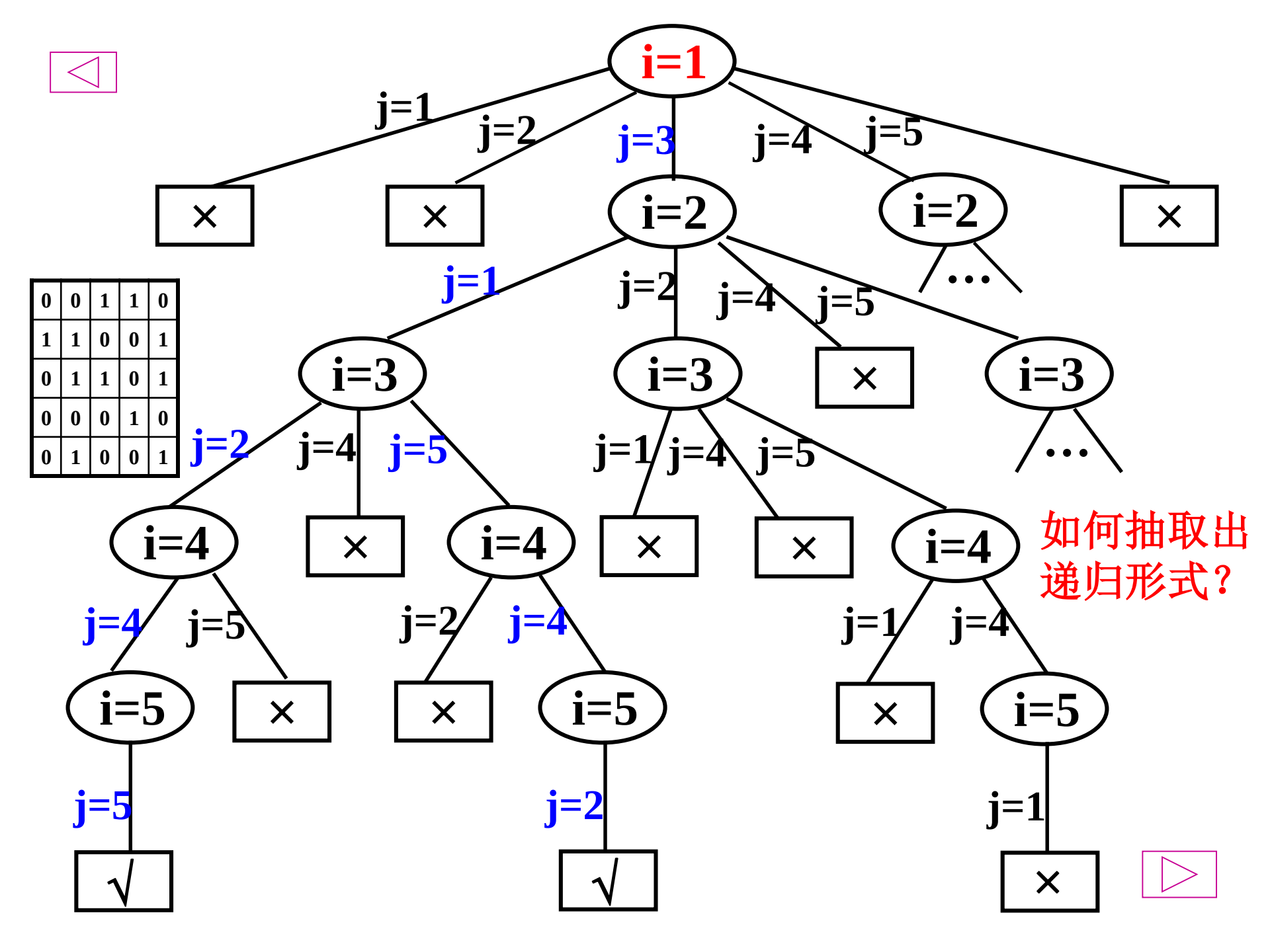
2、定义一个整型一维数组BookFlag[6]用来记录书是否已被选用。用后五个下标作为五本书的标号，被选用的元素值为1, 未被选用的值为0, 初始化皆为0.

```
int  BookFlag[6] = {0};
```

3、定义一个整型一维数组BookTaken[6]用来记录每一个人选用了哪一本书。用数组元素的下标来作为人的标号，用数组元素的值来表示书号。如果某个人还没有选好书，则相应的元素值为0。初始化时，所有的元素值均为0。

```
int  BookTaken[6] = {0};
```

4、循环变量 i 表示人， j 表示书， i, j ∈ {1, 2, 3, 4, 5}


i=1
j=1
j=2
j=3
j=4
j=5
×
×
i=2
i=2
×
…
j=1
j=2
j=4
j=5
i=3
i=3
×
i=3
…
j=2
j=4
j=5
j=1
j=4
j=5
i=4
×
i=4
×
×
i=4
j=4
j=5
j=2
j=4
j=1
j=4
i=5
×
×
i=5
×
i=5
j=5
j=2
j=1
√
√
×
0 0 1 1 0
1 1 0 0 1
0 1 1 0 1
0 0 0 1 0
0 1 0 0 1
如何抽取出
递归形式？

算法思路

```
void person (int   i)
{
    for (j = 1; j <= 5; j++)   //尝试把每本书分给第i个人
    {
        if (第j本书分给第i个人不可行) continue; //失败
        把第j本书分给第i个人;
        if（i == 5）
            输出解;
        else
            person (i + 1);  //给第i+1个人分书
        回溯，把这一次分得的书退回
    }
}
```

关键是数据定义！

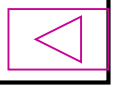
```
void   person( int  i );

int   Like[6][6]  =  {{0},   {0,   0,  0,  1,  1,  0},
                             {0,   1,  1,  0,  0,  1},
                             {0,   0,  1,  1,  0,  1},
                             {0,   0,  0,  0,  1,  0},
                             {0,   0,  1,  0,  0,  1}};
int   BookFlag[6]  =  {0};
int   BookTaken[6]  =  {0};

int  main( )
{
      person( 1 );
}
```

```
void   person(int  i)   // 尝试给第i个人分书
{   int   j,  k;
    for(j = 1;  j <= 5;  j++)      // 尝试把每本书分给第i个人
    {
        if((BookFlag[j] != 0) || (Like[i][j] == 0))   continue; // 失败
        BookTaken[i] = j;          // 把第j本书分给第i个人
        BookFlag[j] = 1;
        if(i == 5){                // 已找到一种分书方案
            for(k = 1; k <= 5; k++) printf("%d ", BookTaken[k]);
            printf("\n");
        }
        else{
             person(i + 1);        // 给第i+1个人分书
        }
        BookTaken[i] = 0;          // 回溯，把这一次分得的书退回
        BookFlag[j] = 0;
    }
}
```

```
3 1 2 4 5
3 1 5 4 2
```

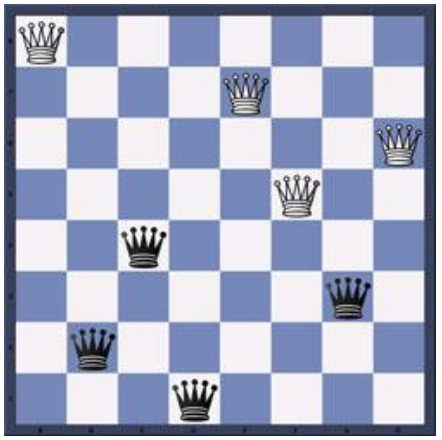


一般回溯策略

1. 将问题的候选解按某种顺序逐一枚举和检验。
2. 当发现候选解不可能是解时，就选择下一候选解。
3. 如果当前候选解除了不满足规模要求外，满足其他所有要求时，继续扩大当前候选解的规模，并继续试探。
4. 如果当前的候选解满足包括问题规模在内的所有要求时，该候选解就是问题的一个解。

2. 八皇后问题

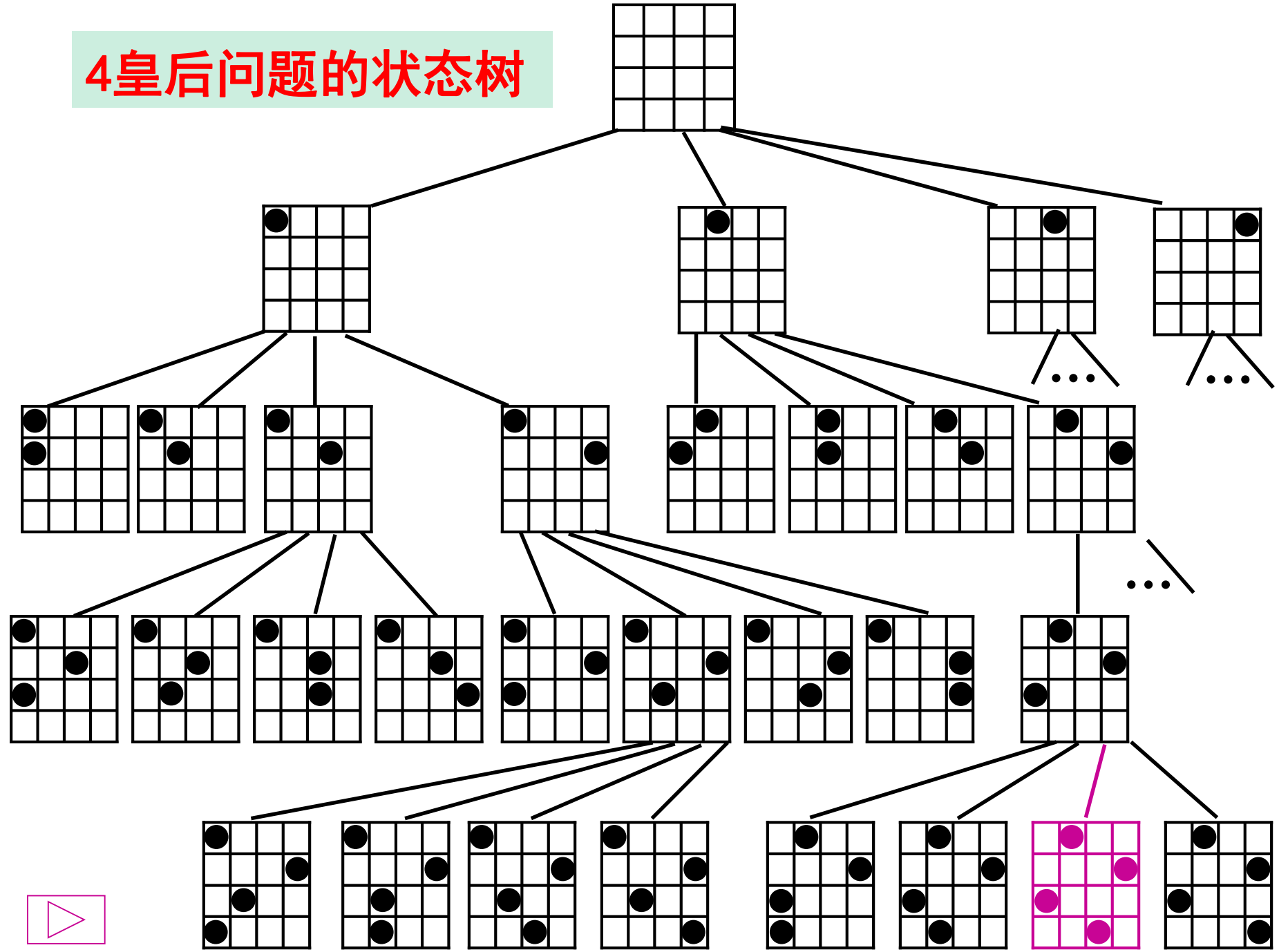
在8×8的棋盘上，放置8个皇后（棋子），使两两之间互不攻击。所谓互不攻击是说任何两个皇后都要满足：



（1）不在棋盘的同一行；
（2）不在棋盘的同一列；
（3）不在棋盘的同一对角线上。

因此可以推论出，棋盘共有8行，故至多有8个皇后，即每一行有且仅有一个皇后。这8个皇后中的每一个应该摆放在哪一列上是解该题的任务。

4皇后问题的状态树



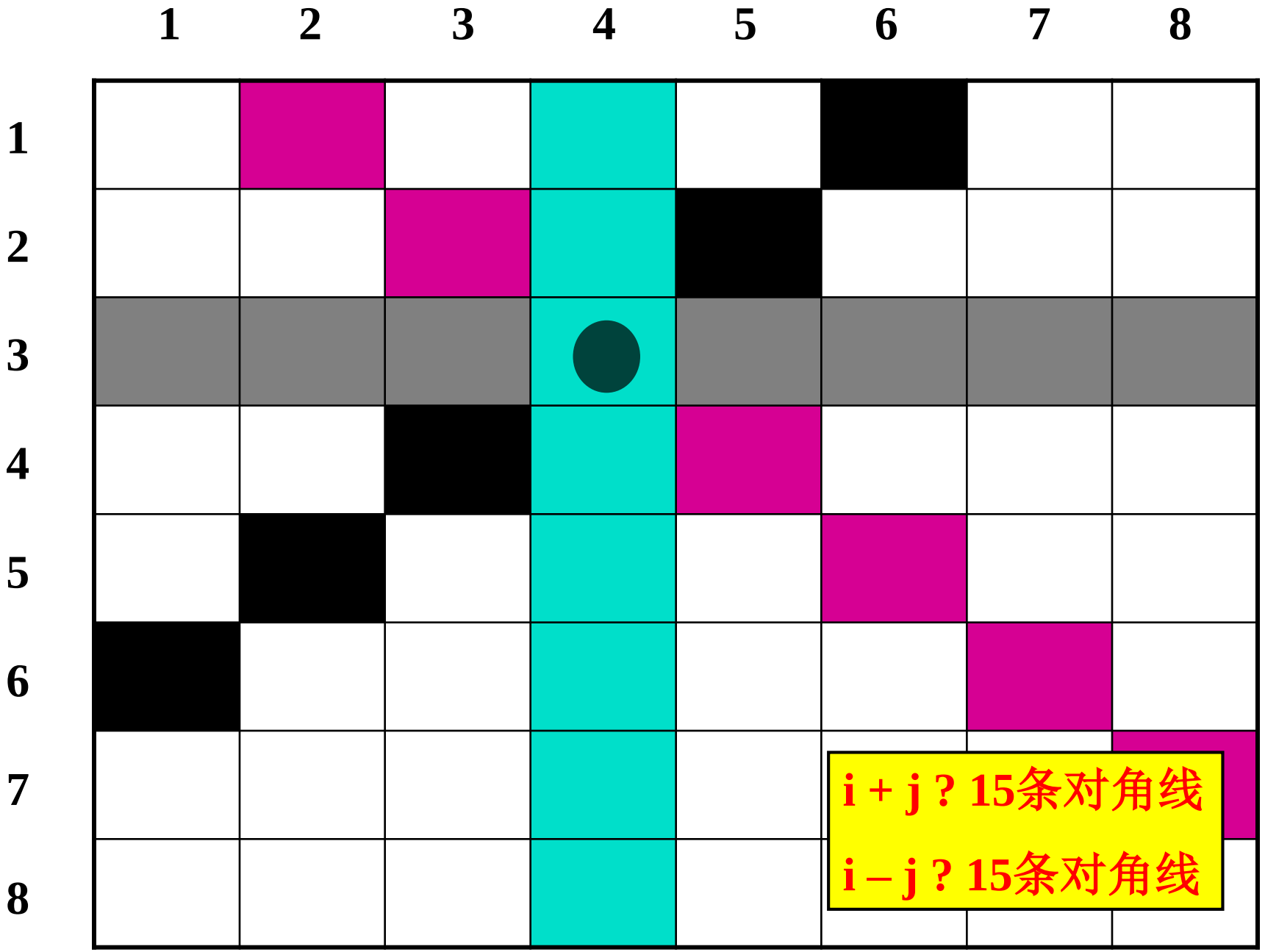
算法思路

```
void  TryQueen (int   i)
{
    for (j = 1; j <= 8; j++)
    {
        if (皇后i放在第j列是不可行的)  continue; //失败
        在第j列放入皇后;
        if（i== 8）
            输出解;
        else
            TryQueen (i + 1);  //摆放第i+1行的皇后
        回溯，把该皇后从第j列拿起
    }
}
```

关键还是数据定义！

数据的定义（1）：

- **i**—— 第i行（个）皇后，1 ≤ i ≤ 8；
- **j**—— 第j列，1 ≤ j ≤ 8；
- ***Queen[i]*** —— 第i行皇后所在的列；
- **Column[j]**—— 第j列是否安全，{0，1}；


1 2 3 4 5 6 7 8
1
2
3
4
5
6
7
8
i + j ? 15条对角线
i – j ? 15条对角线

数据的定义（2）：

- **Down[-7..7]**——记录每一条从上到下的对角线，是否安全，{0,1}
  - (i - j)，取值范围
- Up[2..16]——记录每一条从下到上的对角线，是否安全，{0,1}
  - (i + j)，取值范围

利用以上的数据定义：

- 当我们需要在棋盘的(i, j) 位置摆放一个皇后的时候，可以通过Column数组、Down数组和Up数组的相应元素，来判断该位置是否安全；
- 当我们已经在棋盘的(i, j) 位置摆放了一个皇后以后，就应该去修改Column数组、Down数组和Up数组的相应元素，把相应的列和对角线设置为不安全。

```
void   TryQueen( int   i );

int   Queen[9]    =  { 0 };
int   Column[9]  =  { 0 };
int   Down[15]   =  { 0 };
int   Up[15]      =  { 0 };

int  main( )
{
      TryQueen( 1 );
}
```

```
void   TryQueen(int  i)          // 摆放第 i 行的皇后
{
   int   j,  k;
   for(j = 1;  j <= 8;  j++)      // 尝试把该皇后放在每一列
   {
      if(Column[j] || Down[i-j+7] || Up[i+j-2])   continue; // 失败
      Queen[i] = j;  // 把该皇后放在第j列上
      Column[j] = 1;    Down[i-j+7] = 1;    Up[i+j-2] = 1;
      if(i  ==  8)          // 已找到一种解决方案
      {
         for(k = 1; k <= 8; k++)   printf("%d  ",   Queen[k]);
         printf("\n");
      }
      else    TryQueen(i + 1); // 摆放第i+1行的皇后
      Queen[i] = 0;     // 回溯，把该皇后从第j列拿起
      Column[j] = 0;    Down[i-j+7] = 0;    Up[i+j-2] = 0;
   }
}
```

共92组解，部分答案如下：

方案1：1 5 8 6 3 7 2 4

方案2：1 6 8 3 7 4 2 5

方案3：1 7 4 6 8 2 5 3

方案4：1 7 5 8 2 4 6 3

方案5：2 4 6 8 3 1 7 5

方案6：2 5 7 1 3 8 6 4

方案7：2 5 7 4 1 8 6 3

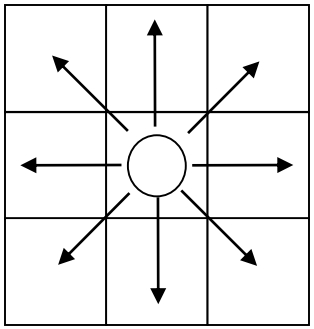
方案8：2 6 1 7 4 8 3 5

方案9：2 6 8 3 1 4 7 5

方案10：2 7 3 6 8 5 1 4

思考题：对题目做如下的修改

1. 假设在棋盘上事先已经摆放了一个国王，位置为<X, Y>，即第X行第Y列，在摆放八个皇后时，要求皇后间不能互相攻击且不能被国王攻击。国王的攻击范围如下图所示：



2. 先输入某一个国王所在的位置<X, Y>，即第X行第Y列，在此情形下，如何摆放其余的8个皇后，要求找到所有解决方案。

```
#include <math.h>
void   TryQueen(int  i)          // 摆放第 i 行的皇后
{
   int   j,  k;
   for(j = 1;  j <= 8;  j++)      // 尝试把该皇后放在每一列
   {
      if(Column[j] || Down[i-j+7] || Up[i+j-2])   continue; // 失败
      if (abs(i - x) <= 1 && abs(j - y) <= 1)  continue;  // 国王周围
      Queen[i] = j;  // 把该皇后放在第j列上
      Column[j] = 1;    Down[i-j+7] = 1;    Up[i+j-2] = 1;
      ……
   }
}
```

核心: (i，j) 与(x，y)的位置关系

3. 下楼问题

从楼上走到楼下共有 h 个台阶，每一步有三种走法：

- 走一个台阶；
- 走二个台阶；
- 走三个台阶。

问可以走出多少种方案。

采用回溯递归思想来编程实现。

数据的定义：

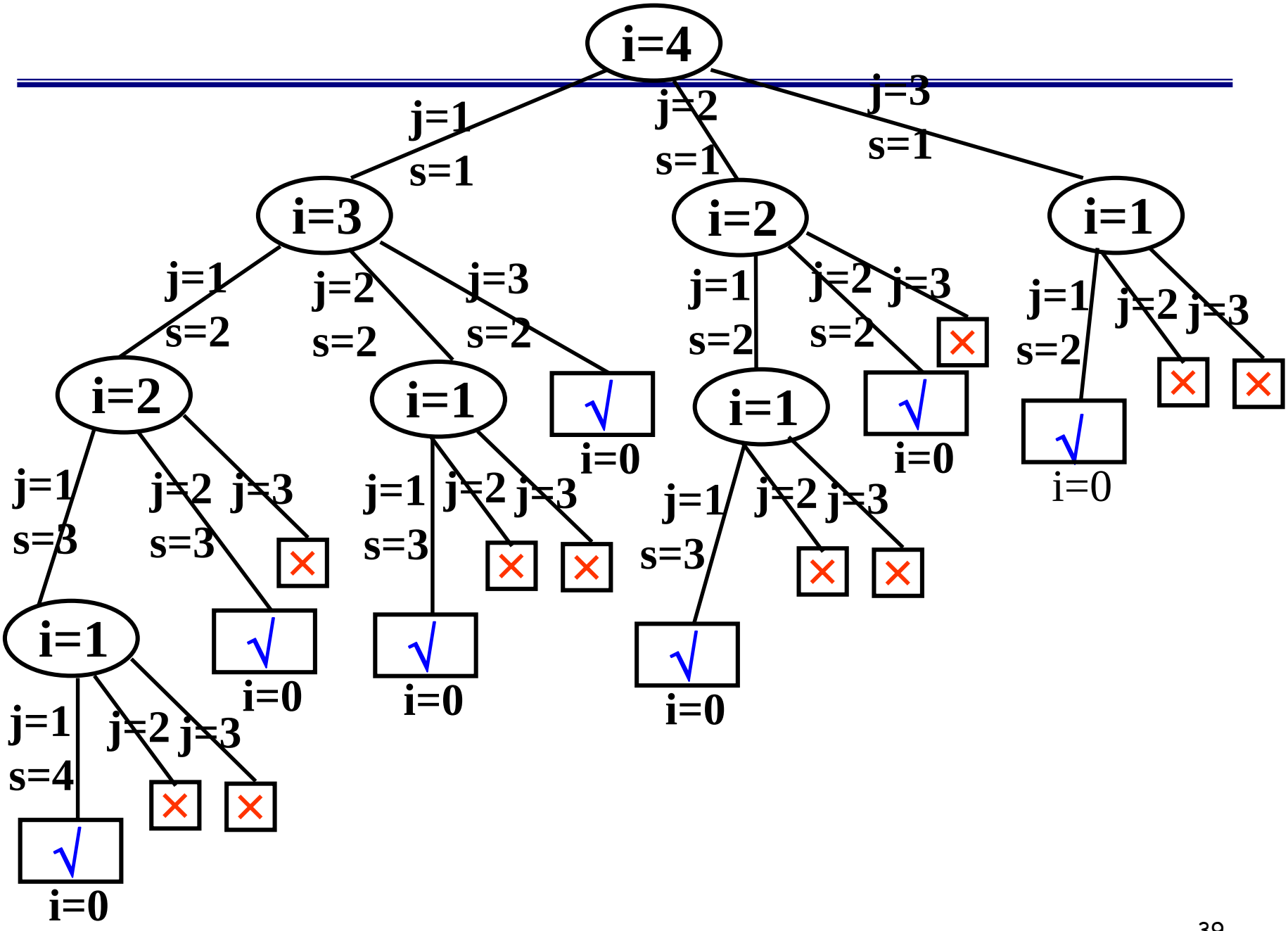
- **j = 1, 2, 3 ——** 表示在每一步可以试着走的台阶数
- **i ——** 表示当前的高度；
- **s ——** 表示步数
- **pace[s] ——** 保存第s步走过的台阶数

基本思路：

1. 用枚举的方法，试着一步一步地走，从高到低。让 i 先取 h 值，然后在下楼时，每走一步 i 的值就会减去这一步所走的台阶数 j，即 i = h（初值），以后 i = i-j，(j=1, 2, 3)，当 i = 0 时，说明已走到楼下。

2. 枚举时，每一步都要试 j 的三种不同取值，即或者为1，或者为2，或者为3。这时可用for循环结构。

3. 每一步走法都用相同的策略，故可以用递归算法。


i=4
j=1
s=1
j=2
s=1
j=3
s=1
i=3
i=2
i=1
j=1
s=2
j=2
s=2
j=3
s=2
j=1
s=2
j=2
s=2
j=3
j=1
s=2
j=2
j=3
i=2
i=1
√
i=0
i=1
√
i=0
√
i=0
√
i=0
j=1
s=3
j=2
s=3
j=3
j=1
s=3
j=2
j=3
j=1
s=3
j=2
j=3
i=1
√
i=0
√
i=0
√
i=0
j=1
s=4
j=2
j=3
√
i=0

定义 TryStep(i, s) —— 站在第i级台阶上往下试走第 s步的过程

如何实现？

第一步：j = 1;

第二步：如果 j > i, 表明这一步不可能走 j 级台阶, 函数返回；否则，转第三步；

第三步：这一步走 j 级台阶，即 pace[s] = j;
如果 i - j = 0，说明已经到达地面了，也就是已经找到一种方案了，把它显示出来；否则的话，接着走下一步，TryStep(i-j，s+1);

第四步：j = j +1，如果 j ≤ 3，转第二步；否则函数结束。

也可用分治策略！

```
void  TryStep(int i,  int s);

int  pace[100] = {0};
int  num = 0;
int  main( )
{
        int h;
        scanf("%d", &h);
        TryStep(h,  1);
        return 0;
}
```

```
void TryStep(int i,  int s)
{
    int j,  k;
    for(j = 1;  j <= 3;  j++)
    {
        if(j > i)  continue;   //试着走的台阶数j  > 剩余台阶数i
        pace[s] = j;   //记录第s步走j个台阶
        if(i == j) {        // 已经到达地面了
            num ++;  printf("方案%d: ", num);
            for(k = 1; k <= s; k++)
                printf("%d ", pace[k]);
            printf("\n");
          }
          else   {   // 尚未走到楼下
             TryStep(i - j, s + 1); //再试剩下的台阶(递归调用)
           }
    }
}
```

```
3
方案1: 1 1 1
方案2: 1 2
方案3: 2 1
方案4: 3
```

```
4
方案1: 1 1 1 1
方案2: 1 1 2
方案3: 1 2 1
方案4: 1 3
方案5: 2 1 1
方案6: 2 2
方案7: 3 1
```

```
5
方案1: 1 1 1 1 1
方案2: 1 1 1 2
方案3: 1 1 2 1
方案4: 1 1 3
方案5: 1 2 1 1
方案6: 1 2 2
方案7: 1 3 1
方案8: 2 1 1 1
方案9: 2 1 2
方案10: 2 2 1
方案11: 2 3
方案12: 3 1 1
方案13: 3 2
```

4. 排列问题

n个对象的一个排列，就是把这 n 个不同的对象放在同一行上的一种安排。例如，对于三个对象 a, b, c，总共有6个排列：

a b c
a c b
b a c
b c a
c a b
c b a

n 个对象的排列个数就是 n!。

如何生成排列？

基于分治策略的递归算法：

- 假设这 n 个对象为 $1, 2, 3, \ldots, n$；
- 对于前n-1个元素的每一个排列 $a_1 a_2 \ldots a_{n\text{-}1}$，$1 \le a_i \le n\text{-}1$，通过在所有可能的位置上插入数字 n，来形成 n 个所求的排列，即：

$$n\, a_1 a_2 \ldots a_{n\text{-}1}$$
$$a_1\, n\, a_2 \ldots a_{n\text{-}1}$$
$$\ldots\ldots$$
$$a_1 a_2 \ldots n\, a_{n\text{-}1}$$
$$a_1 a_2 \ldots a_{n\text{-}1}\, n$$

例如：生成1,2,3的所有排列

permutation(3) → permutation(2) → permutation(1)

permutation(1)：1

permutation(2)：2 1，1 2

permutation(3)：3 2 1，2 3 1，2 1 3，

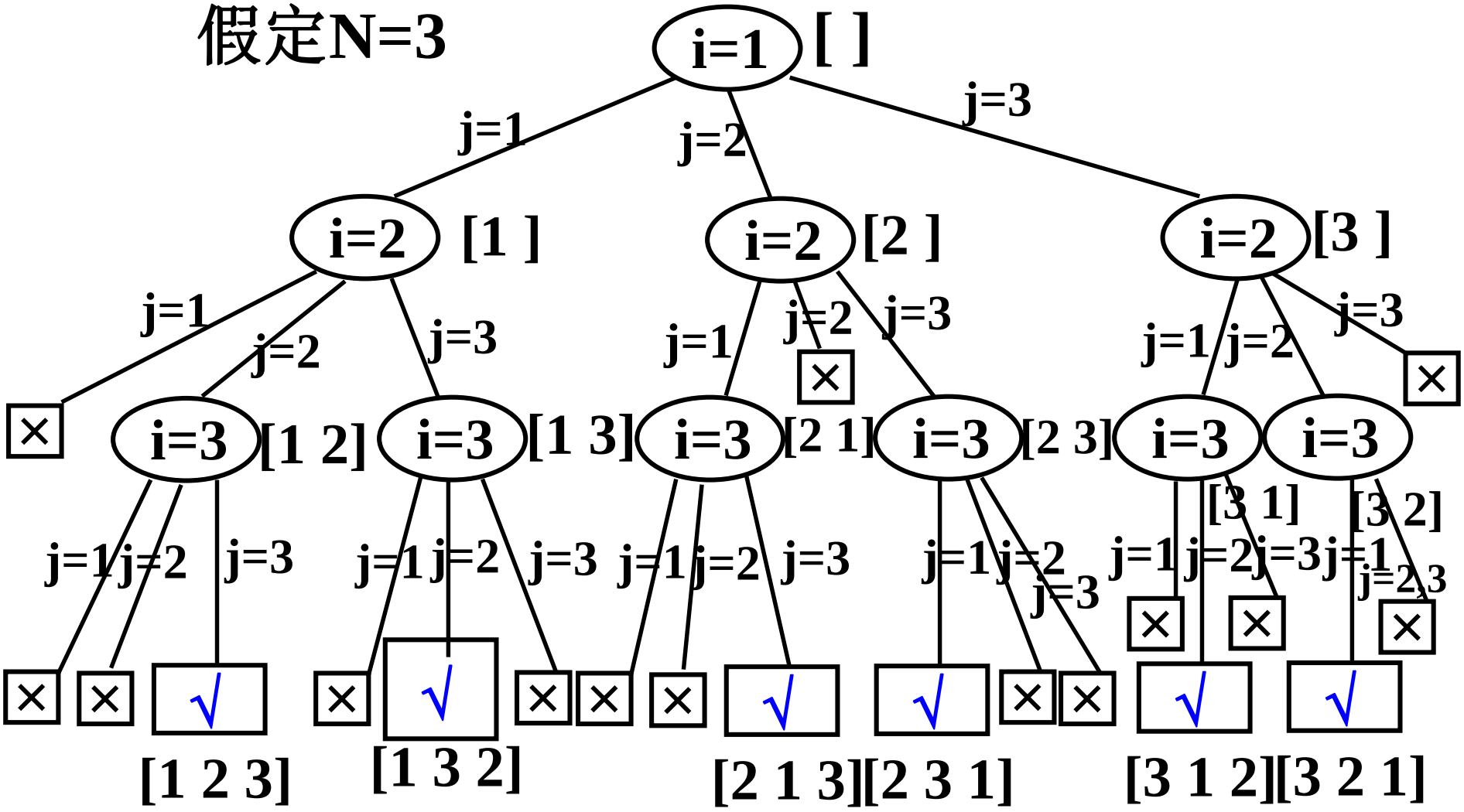
3 1 2，1 3 2，1 2 3

基于回溯策略的递归算法：

- 基本思路：每一个排列的长度为 N，对这N个不同的位置，按照顺序逐一地枚举所有可能出现的数字。
- 定义一维数组NumFlag[N+1]用来记录1-N之间的每一个数字是否已被使用，1表示已使用，0表示尚未被使用，初始化皆为0；

- 定义一维数组NumTaken[N+1]，用来记录每一个位置上使用的是哪一个数字。如果在某个位置上还没有选好数字，则相应的数组元素值为0。初始化时，所有元素值均为0；
- 循环变量 i 表示第 i 个位置，j 表示整数 j，$i, j \in \{1, 2, \ldots, N\}$。

循环变量 i 表示第 i 个位置，j 表示第 j个数字，i, j ∈ {1, 2, 3}



```
#include <stdio.h>
#define      N      3

void   TryNumber( int  i );

int   NumFlag[N+1]  =  {0};
int   NumTaken[N+1]  =  {0};

int  main( )
{
      TryNumber( 1 );
}
```

```
void  TryNumber(int  i)
{
      int  j,  k;
      for(j = 1;  j <= N;  j++) // 枚举法，尝试第i个位置放数字j
     {
         if(NumFlag[j]  !=  0)   continue;  // 该数已被占用
         NumTaken[i] = j;      // 成功，在第i个位置放数字j
         NumFlag[j] = 1;       // 修改状态标志，数字j被占用
         if(i == N)                 // 递归边界：已找到一个排列
         {
            for(k = 1; k <= N; k++) printf("%d ", NumTaken[k]);
            printf("\n");
         }
         else   TryNumber(i + 1);     //未到边界，尝试第i+1位置
         NumTaken[i] = 0;      // 回溯，把本次分配的数退回
         NumFlag[j] = 0;
   }
}
```

N = 3, 共6种组合

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

N = 4, 共24种组合

1 2 3 4

1 2 4 3

1 3 2 4

1 3 4 2

1 4 2 3

1 4 3 2

2 1 3 4

2 1 4 3

2 3 1 4

2 3 4 1

2 4 1 3

……

问题描述：

编写一个程序，它接受用户输入的一个英文单词（长度不超过20个字符），然后输出由这个单词的各个字母所组成的所有排列。有两个条件：

1. 这个单词的各个字母允许有相同的；
2. 不能输出重复的排列。

例如：

如何检测重复排列？

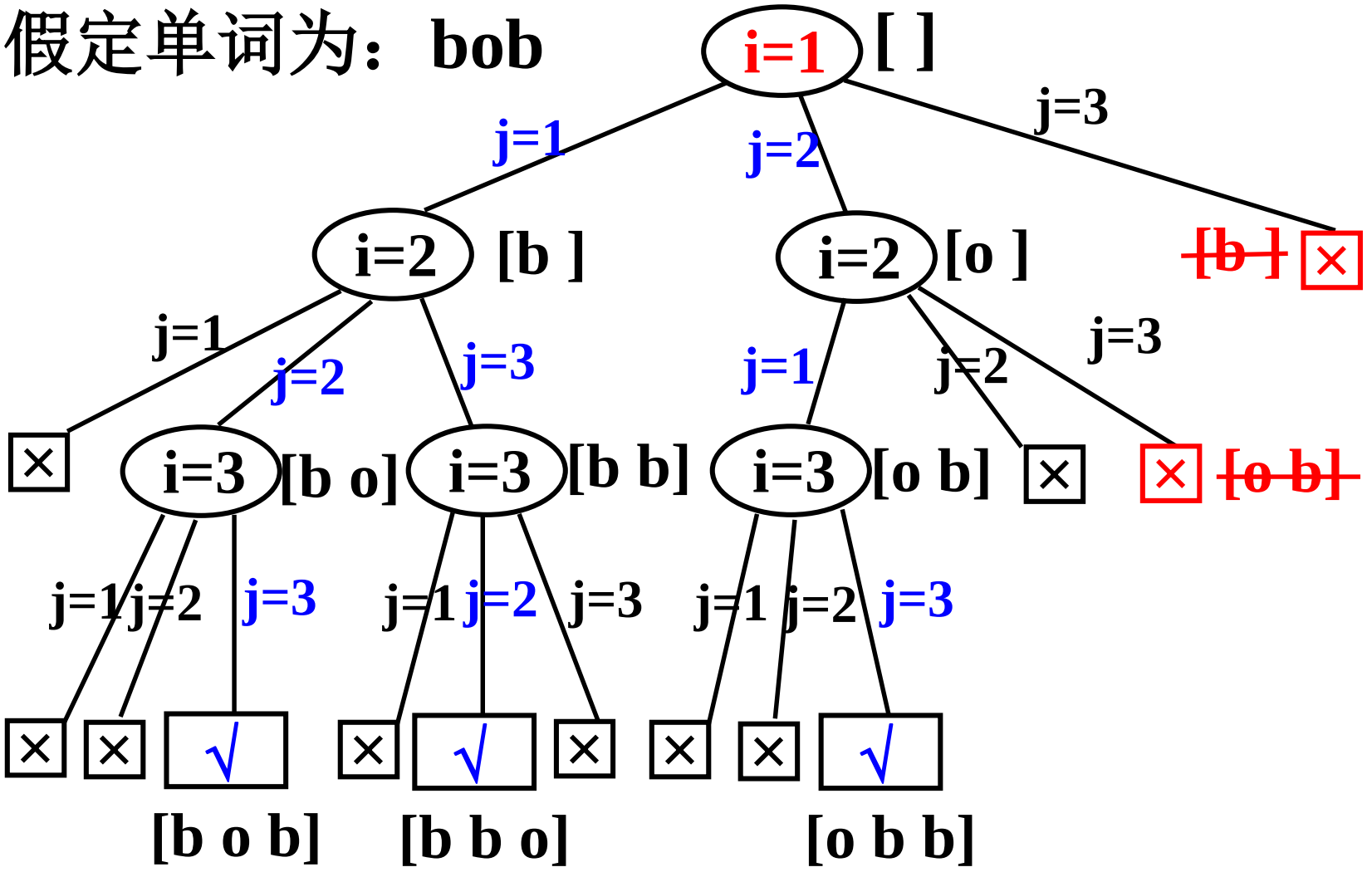
```
请输入一个英文单词：boy

boy
byo
oby
oyb
ybo
yob
```

```
请输入一个英文单词：bob

bob
bbo
obb
```

循环变量 i 表示第 i 个位置，j 表示第 j个字符，i, j ∈ {1, 2, 3}

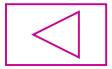


```
#include <stdio.h>
#include <string.h>
#define   MAXSIZE   20

void  TryChar(int  i);
int  CharFlag[MAXSIZE]  =  {0};
int  CharTaken[MAXSIZE]  =  {0};
char  word[MAXSIZE];
int  length;

int  main( )
{
      printf("请输入一个单词：");
      scanf("%s",  word);
      length  =  strlen(word);
      TryChar(1);
}
```

```
void   TryChar(int  i)
{
      int   j,  k;
      for(j = 1;  j <= length;  j++) // 尝试第i个位置放第j个字符
     {
           if (CharFlag[j] == 1)   continue; // 该字符已被占用
           for(k = 1;  k < j;  k++) //对已测试过的前j-1个字符遍历
           {
                 if(CharFlag[k] == 0)  //若第k个字符未被占用
                 {
                       // 在同一结点不能测试相同的两个字符，
                       // 减1是为了对齐下标。
                       if(word[k - 1] == word[j - 1])   break;
                 }
           }
           if(k  <  j)   continue;   //j字符与之前测试过的k字符一致
```



```
            CharTaken[i] = j;  // 成功，在第i个位置放第j个字符
            CharFlag[j] = 1;   // 修改状态标志，第j个字符被占用
            if(i == length)       // 递归边界：已找到一个排列
            {
                for(k = 1; k <= length; k++)
                {
                    printf("%c ", word[CharTaken[k] - 1]);
                }
                printf("\n");
            }
            else
                TryChar(i + 1);  //未到边界，尝试第i+1位置
            CharTaken[i] = 0;    // 回溯，把本次分配的数退回
            CharFlag[j] = 0;
        }
}
```

```
请输入一个单词：bob
b o b
b b o
o b b
```

```
请输入一个单词：food
f o o d
f o d o
f d o o
o f o d
o f d o
o o f d
o o d f
o d f o
o d o f
d f o o
d o f o
d o o f
```

递归算法总结

- “从简到繁”，先考虑简单的情形，再逐渐考察复杂的情形，用一个简单的、具体的例子走一遍，从而理清解题的思路。
- 在理清思路之后，从中发现共性，抛除具体的参数，抽象出相同的步骤，也就是递归的形式。
- 把递归的形式整理为递归函数。

Lecture 8 - Summary

- **Topics covered:**
  - **Recursive programming**
  - **Divide-and-conquer algorithms**
  - **Backtracking algorithms**
  - **Factorial, Binary Search, Towers of Hanoi, Eight Queens**
  - **……**