電子工業出版社
Publishing House of Electronics Industry

第5章　回溯法

学习要点

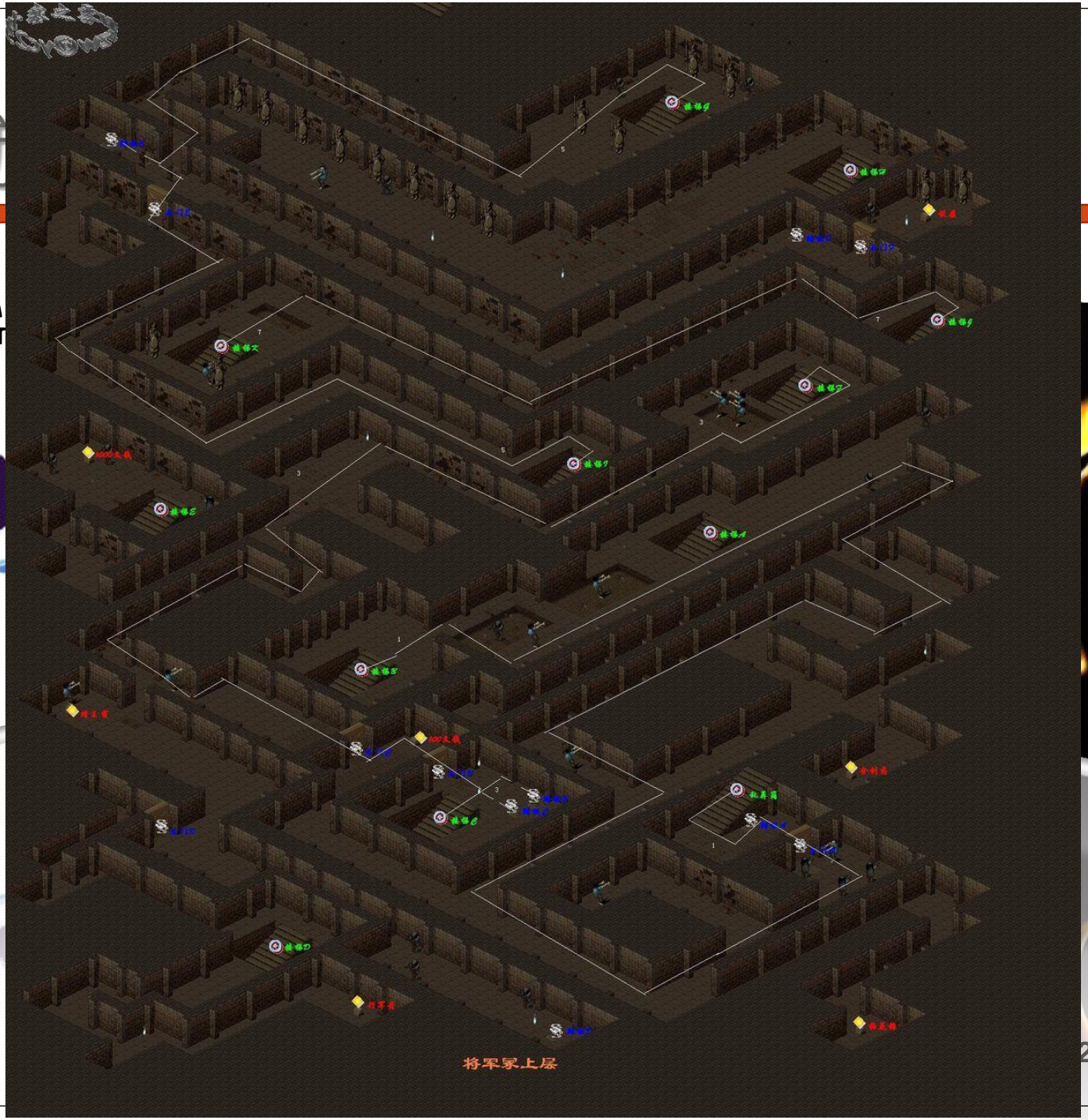
- 掌握用回溯法解题的算法框架
- 理解回溯法的深度优先搜索策略。
- 通过应用范例学习回溯法的设计策略。

学习要点:

- 通过应用范例学习回溯法设计策略
  - （1）n皇后问题;
  - （2）图的m着色问题
  - （3）迷宫问题
  - （4） 0-1背包问题
  - （5）子集和数问题
  - （6）装载问题

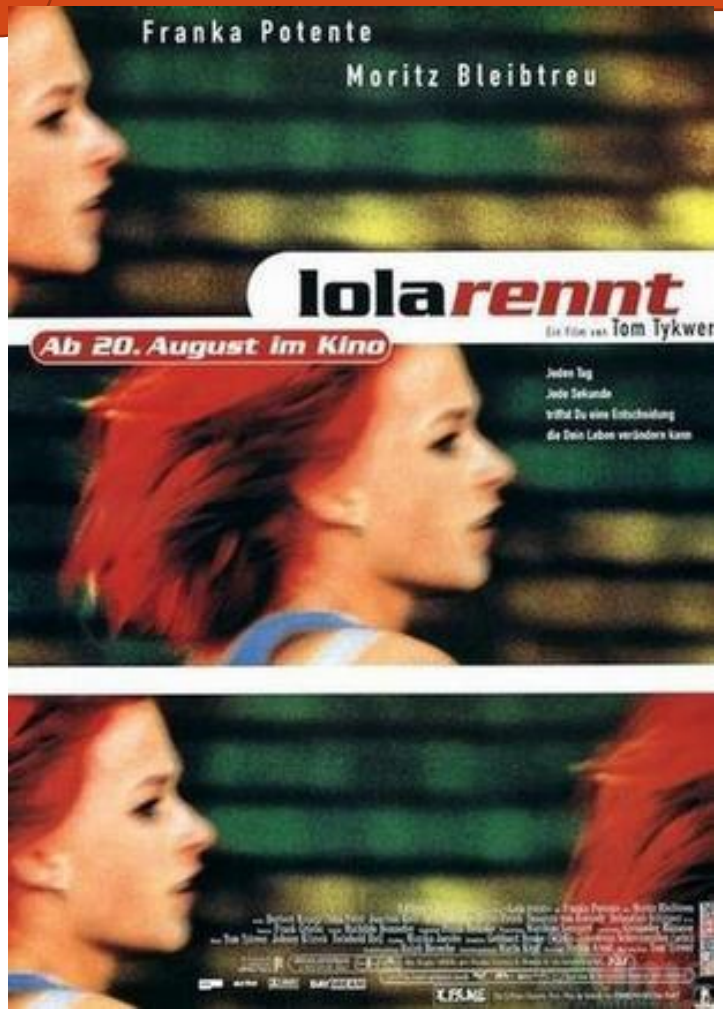
流

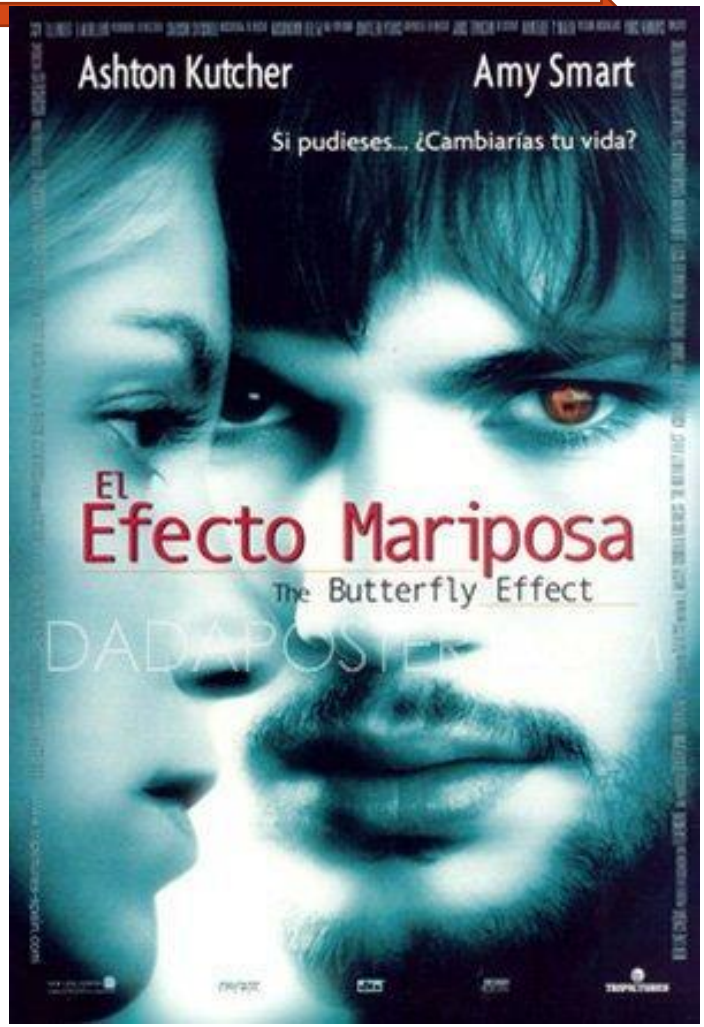
- RPG



圆明园黄花阵迷宫

电影中的回溯


Franka Potente
Moritz Bleibtreu
lola rennt
Tom Tykwer
Ab 20. August im Kino


Ashton Kutcher
Amy Smart
Si pudieses... ¿Cambiarías tu vida?
El Efecto Mariposa
The Butterfly Effect

路漫漫其修远兮 吾将上下而求索

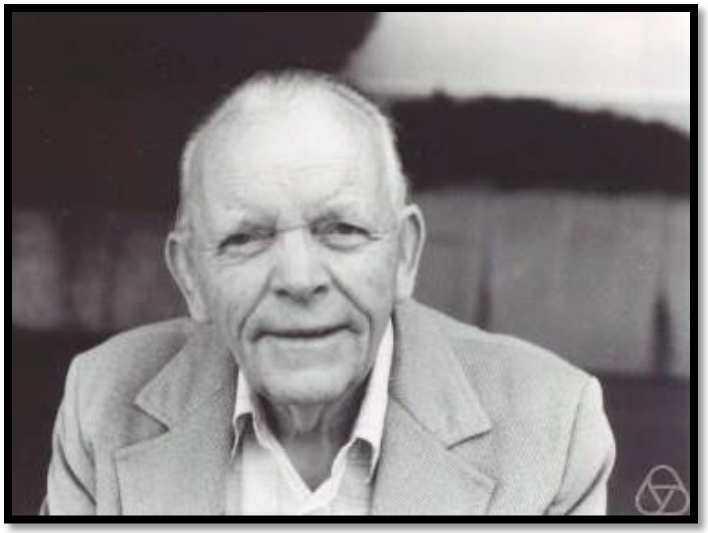
- **回溯法（也称试探法）：将问题候选解按某一顺序逐一枚举和试探的过程。**
- **三种可能的情况：**

回溯：发现当前候选解不可能是可行解或最优解，则直接选下一个候选解

试探：当前候选解除了不满足问题规模的要求外，满足所有其他要求，则继续扩大当前候选解规模

解：满足包括问题规模在内的所有要求，则该候选解就是问题的一个可行解或最优解

回溯法的历史



Derrick Henry Lehmer

(1905/02/23~1991/05/22)

- 上世纪50年代由美国数学家Derrick Henry Lehmer 提出的一种算法，属于一种“穷尽搜索算法”（Brute-force search）
- Mersenne素数的Lucas－Lehmer测试

5.1 回溯法概述

- 回溯法在问题的解空间树中，按深度优先策略，从根结点出发搜索解空间树。
- 算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解。
- 如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向其祖先结点回溯；否则，进入该子树，继续按深度优先策略搜索。

问题的解空间

- **可行解**：满足约束条件的解。解空间中的一个子集 。
- **最优解**：使目标函数取极值（极大或极小）的可行解，一个或少数几个 。
- 例如：
  - TSP问题:有n!种可能解，有些是可行解，只有一个或几个是最优解。
  - 八后问题和图的着色问题:只要可行解，不需要最优解。

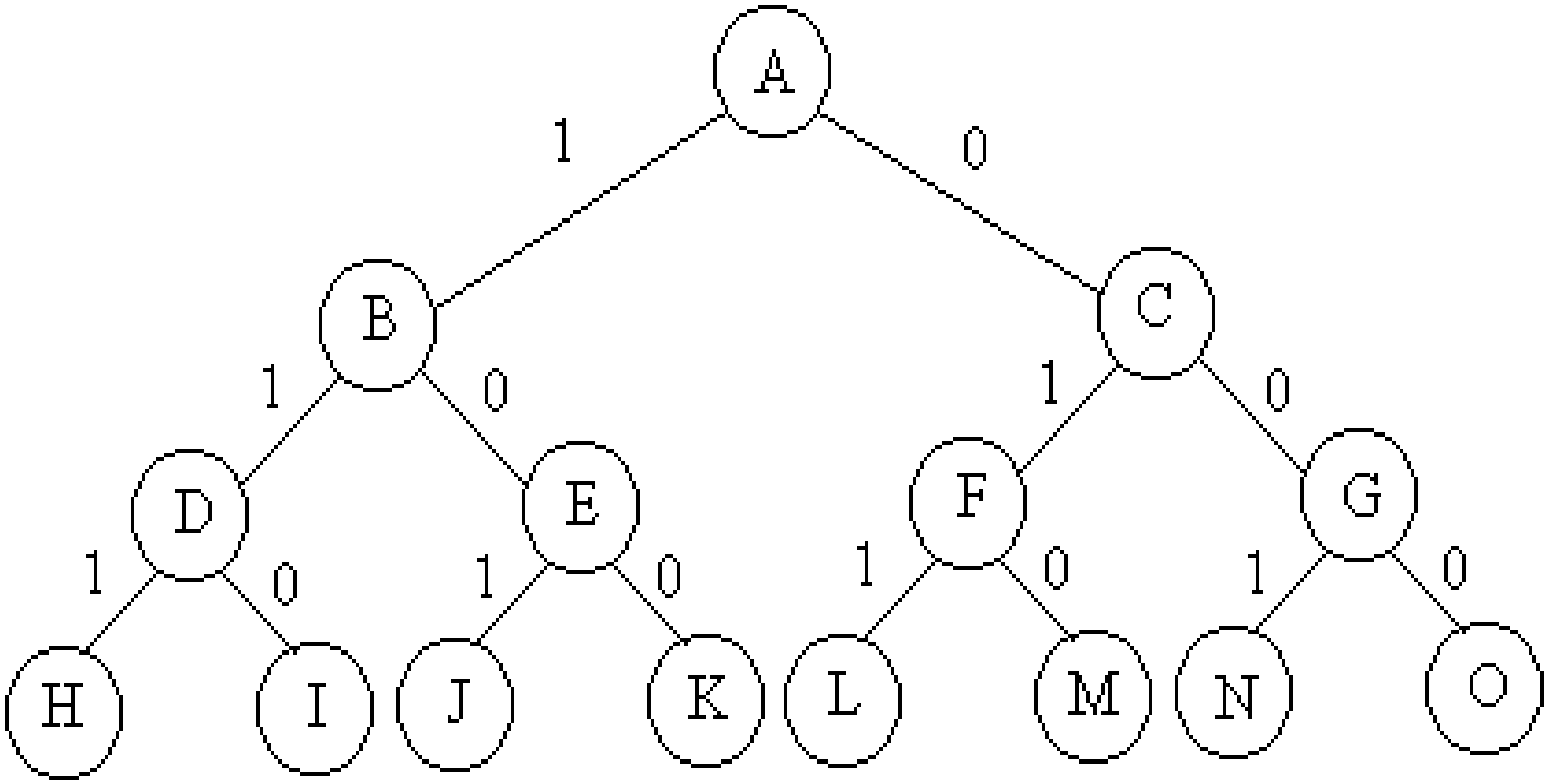
5.2 回溯法的算法框架

一、问题的解空间

- 定义问题的解空间：
  - 对于问题的一个实例，解向量满足显式约束条件的所有多元组，构成了该实例的一个解空间。
  - 例：n=3的0-1背包问题

n=3时的0-1背包问题

用完全二叉树表示的解空间



5.2 回溯法的算法框架

二、回溯法的基本思想

- 确定了解空间的组织结构后，回溯法就从开始结点(根节点)出发，以深度优先的方式搜索整个解空间。

- 深度优先的问题状态生成法：
  - 对一个扩展结点R，一旦产生了它的一个儿子C，就把C当做新的扩展结点。
  - 完成对子树C（以C为根的子树）的穷尽搜索之后，将R重新变成扩展结点，继续生成R的下一个儿子(若有)
  - 在一个扩展结点变成死结点之前，它一直是扩展结点

- 回溯法即以这种工作方式递归地在解空间中搜索，直至找到所要求的解或解空间中已无活结点时为止。

5.2 回溯法的算法框架

二、回溯法的基本思想

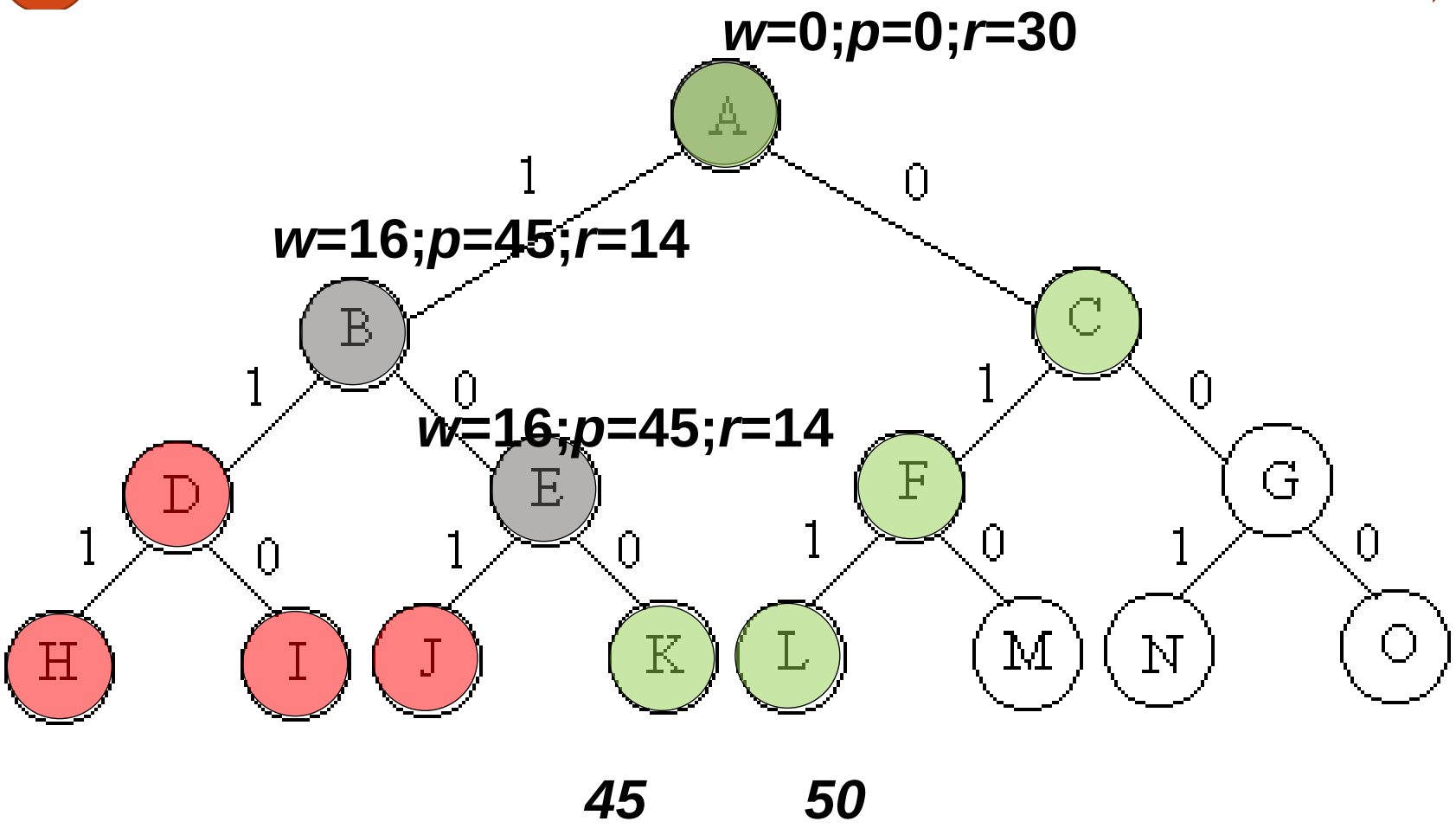
□基本概念：

试探

回溯

- 扩展结点：一个正在产生儿子的结点称为扩展结点
- 活结点：一个自身已生成但其儿子还没有全部生成的节点称做活结点
- 死结点：一个所有儿子已经产生的结点称做死结点

$w=[16, 15, 15]; p=[45, 25, 25]; c=30$



旅行售货员问题

- 旅行售货员问题又称货郎担问题，是指某售货员要到n个城市去推销商品，已知各城市之间的路程（或旅费）。

- 售货员要选定一条从驻地出发经过每个城市一次，最后回到驻地的路线，使总的路程（总旅费）最短（最小）。

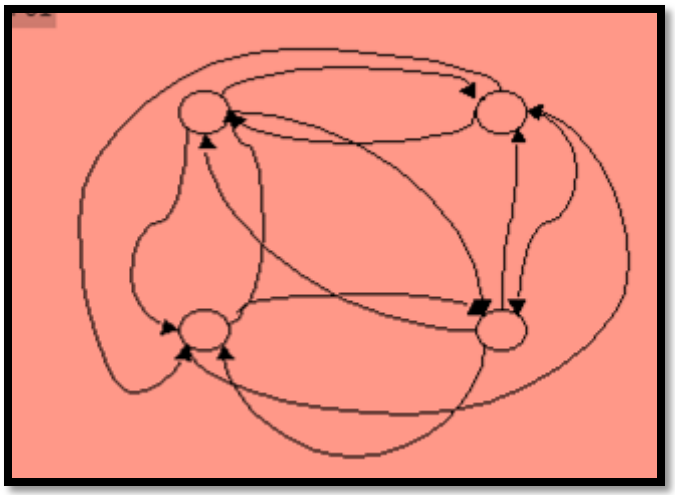
旅行售货员问题
(Traveling Saleman Problem)

- 使用图论语言描述：
  - 设图G=（V，E）是一个加权连通图
  - 要求：
    - |V|=n,
    - 边（i,j）(i不等于j)的费用c[]i[j]为正数。

- 图G的一条周游路线是经过V中的每个顶点恰好一次的一条回路。
- 周游路线的费用是这条回路上的所有边的费用之和。
- TSP问题就是要在图G中找出费用最小的周游路线。

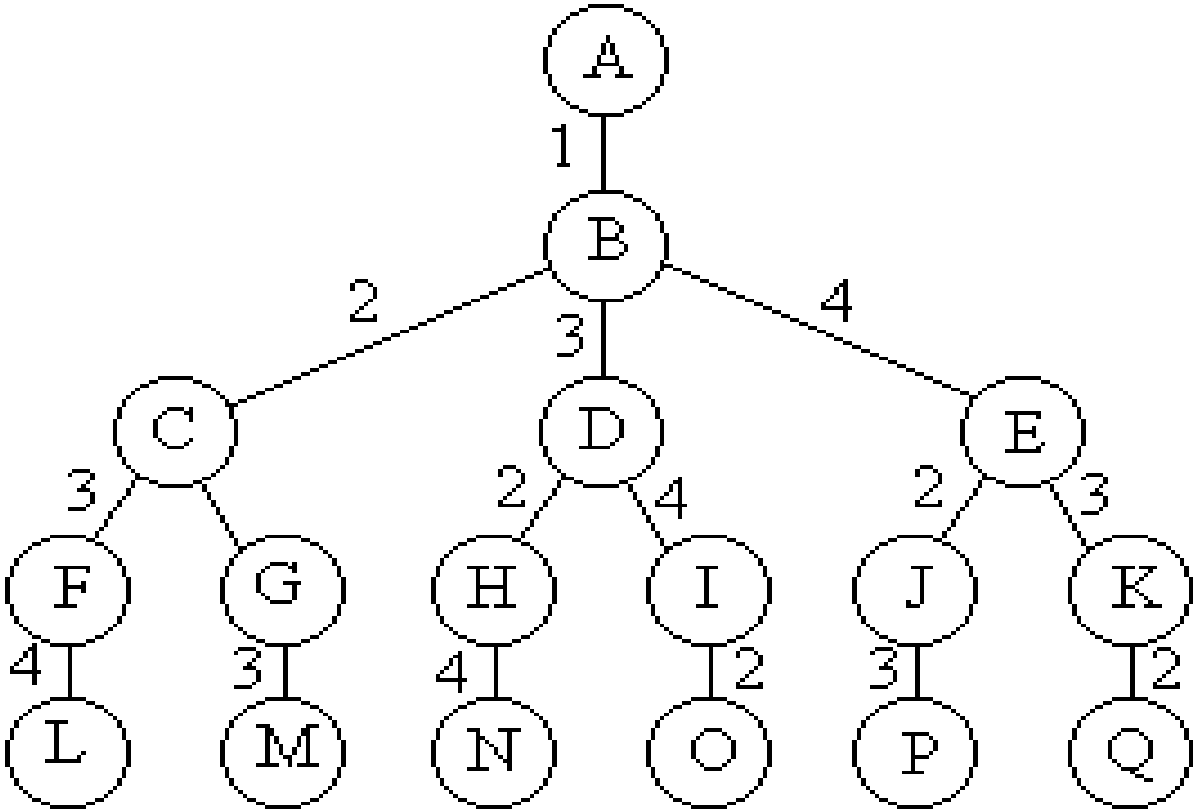
旅行售货员问题
(Traveling Saleman Problem)

- 方法一：穷举法
  - 由起始点出发的周游路线一共有(n-1)!条，即等于除始结点外的n-1个结点的排列数。
  - 旅行售货员问题是一个排列问题。
  - 穷举法的时间复杂度：O(n!)



$$C = \begin{bmatrix} \infty & 2 & 1 & 3 \\ 1 & \infty & 6 & 1 \\ 3 & 4 & \infty & 4 \\ 7 & 1 & 5 & \infty \end{bmatrix}$$

|V|=4旅行售货员问题的解空间



5.2 回溯法的算法框架

回溯法：

- 为了避免生成那些不可能产生最佳解的问题状态，要不断地利用剪枝函数来剪掉那些实际上不可能产生所需解的活结点，以减少问题的计算量。

- 具有剪枝函数的深度优先生成法称为回溯法。

◆常用剪枝函数：背包问题中大于背包容量

➢用约束函数在扩展结点处剪去不满足约束的子树；

➢用限界函数剪去得不到最优解的子树。

最大价值

回溯法的步骤

- 针对所给问题，定义问题的解空间；
- 确定易于搜索的解空间结构；
- 以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

（4）递归回溯

- 回溯法对解空间作深度优先搜索，因此，在一般情况下用递

当前扩展结点在解空间树中的深度

索过的子树的起始、

当前扩展结点处的约束函数和限界函数

```
void backtrack (int t)
{
    if (t>n )output(x);
     else
       for (int i=f(n,t);i<=g(n,t);i++) {
         x[t]=h(i);          //h(i)表示当前扩展结点处的第i个可选值
         if (constraint(t)&&bound(t))
             backtrack(t+1);
       }
}
```

以调用递归函数方式试探

以递归函数返回方式回溯

（5）迭代回溯

- 采用树的非递归深度优先遍历算法，可将回溯法表示为一个非递归迭代过程。

```
void iterativeBacktrack (){
  int t=1;
  while (t>0) {
    if (f(n,t)<=g(n,t))
      for (int i=f(n,t);i<=g(n,t);i++) {
        x[t]=h(i); //当前扩展结点处的第i个可选值
        if (constraint(t)&&bound(t)) {
          if (solution(t)) output(x);
          else t++;}
      }
    else t--;
  }
}
```

当前扩展结点处未搜索过的子树的起始、终止编号

试探

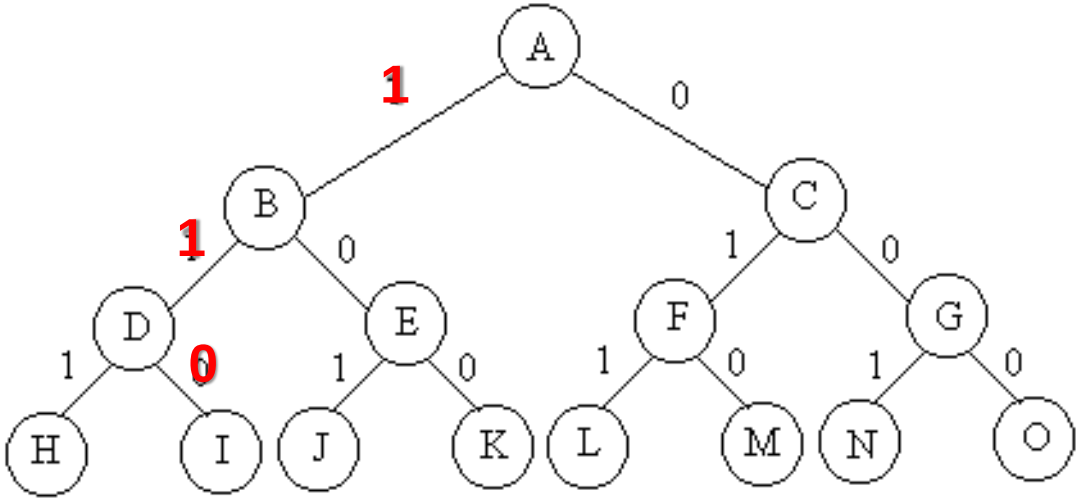
回溯

当前扩展结点处的约束函数和限界函数

(6) 子集树与排列树

- 子集树：当所给问题是从n个元素的集合S中找出S满足某种性质的子集时，相应的解空间树称为子集树。如：n个物品的0-1背包问题所相应的解空间树。
- 遍历子集树需$O(2^n)$计算时间。

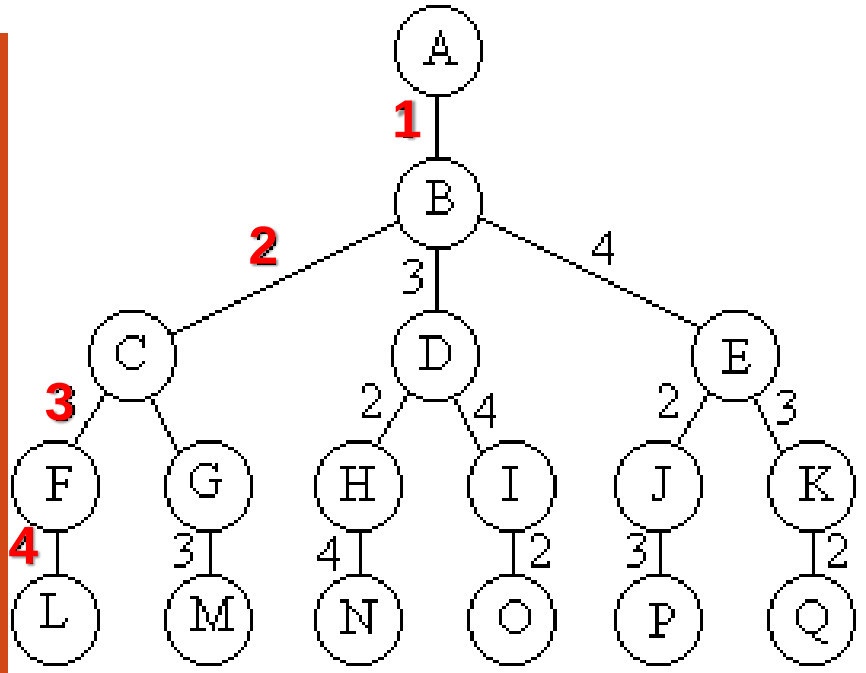
```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=0;i<=1;i++) {
        x[t]=i;
        if (legal(t)) backtrack(t+1);
      }
}
```



(6) 子集树与排列树

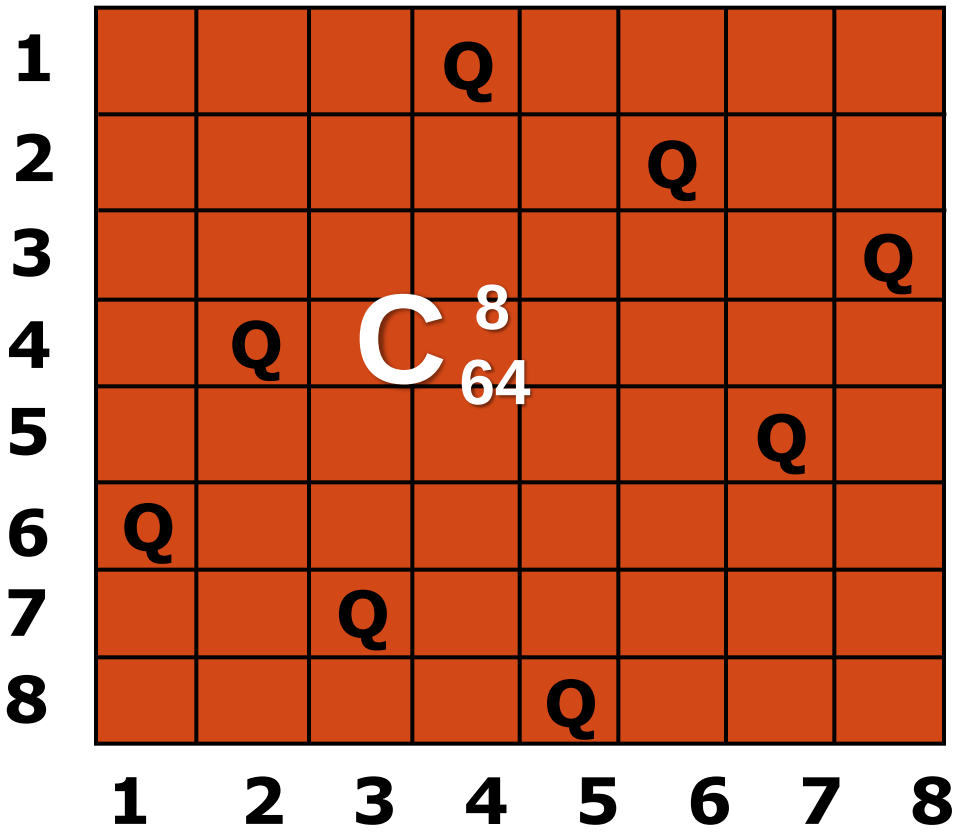
- 排列树：当所给问题是确定n个元素满足某种性质的排列时，相应的解空间树称为排列树。如：旅行售货员问题的解空间树。
- 遍历排列树需要O(n!)计算时间。

```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=t;i<=n;i++) {
        swap(x[t], x[i]);
        if (legal(t)) backtrack(t+1);
        swap(x[t], x[i]);
      }
}
```



5.3 n皇后问题

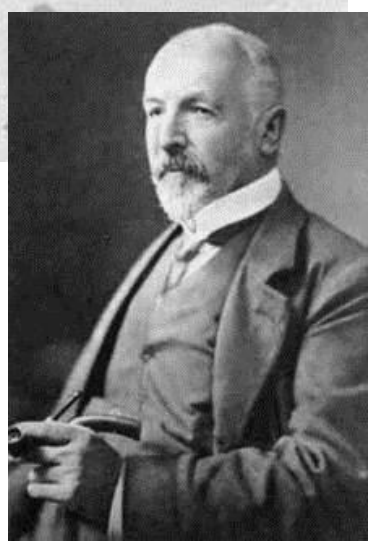
- 在n×n格的棋盘上放置彼此不受攻击的n个皇后。按照国际象棋的规则，皇后可以攻击与之处在同一行或同一列或同一斜线上的棋子。
- n后问题等价于在n×n格的棋盘上放置n个皇后，任何2个皇后不放在同一行或同一列或同一斜线上。



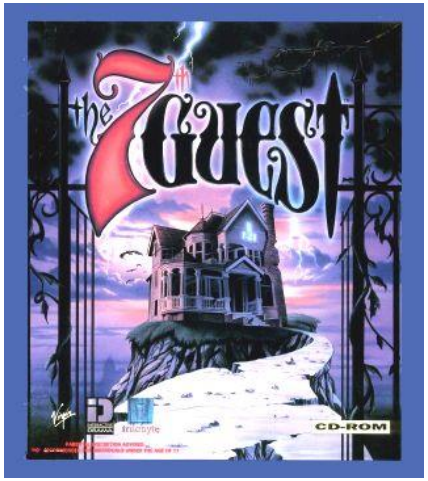
About Eight / n Queens Puzzle

- 1848年，德国棋手Max Bezzel 提出
- Gauss和Georg Cantor 都研究过8 / n
- 1850年，Franz Nauck 等

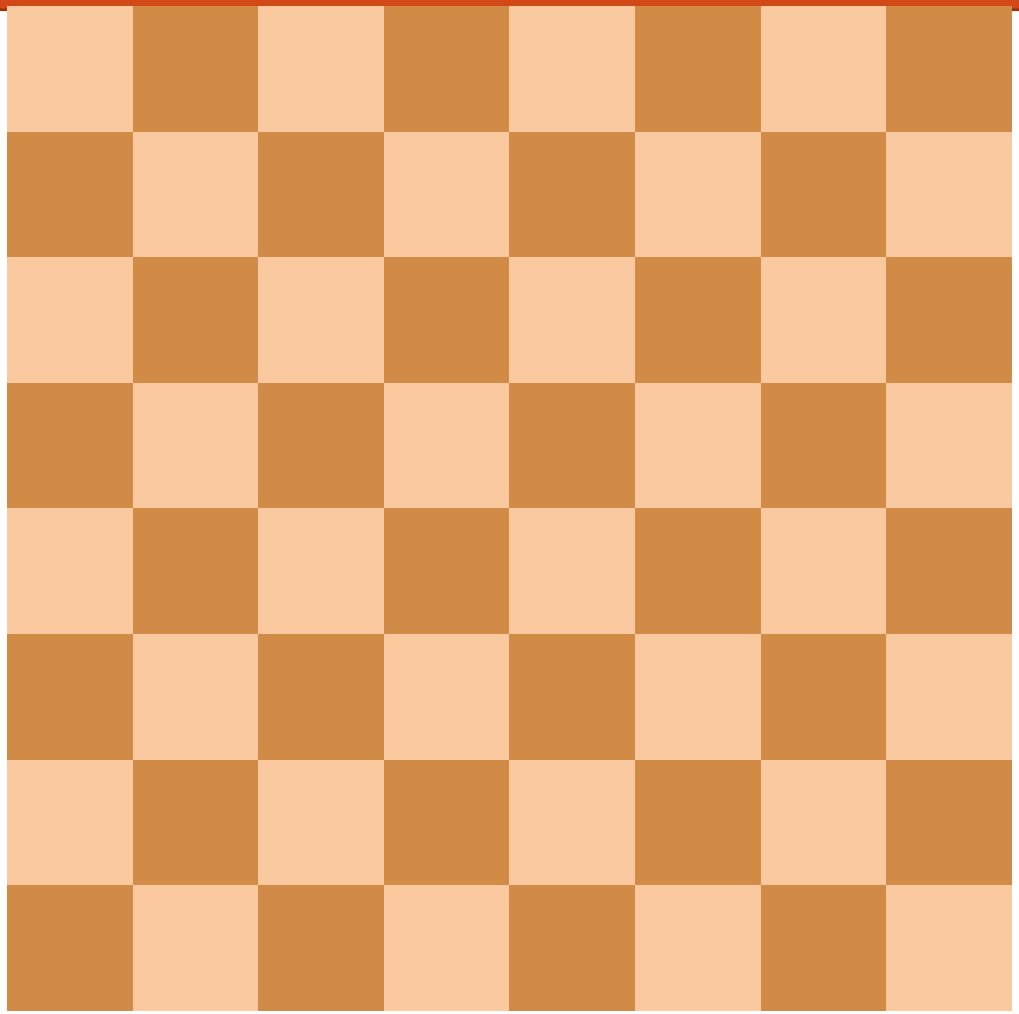
猜测有96个解，未证明；实际只有92个解，且经过旋转或转秩后只有12个不同



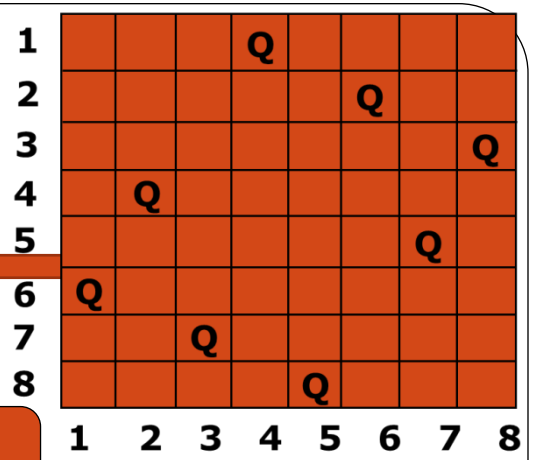
- 1874年， S. Gunther用行列式求解，J.W.L. Glaisher 进行了细化
- 1972年，Edsger Dijkstra 用该问题来说明结构化编程的能力。撰文说明如何编写深度优先的回溯算法
- 基于启发式的解法…
- 1993年，出现于流行的计算机游戏：*The 7th Guest.*



8皇后回溯演示

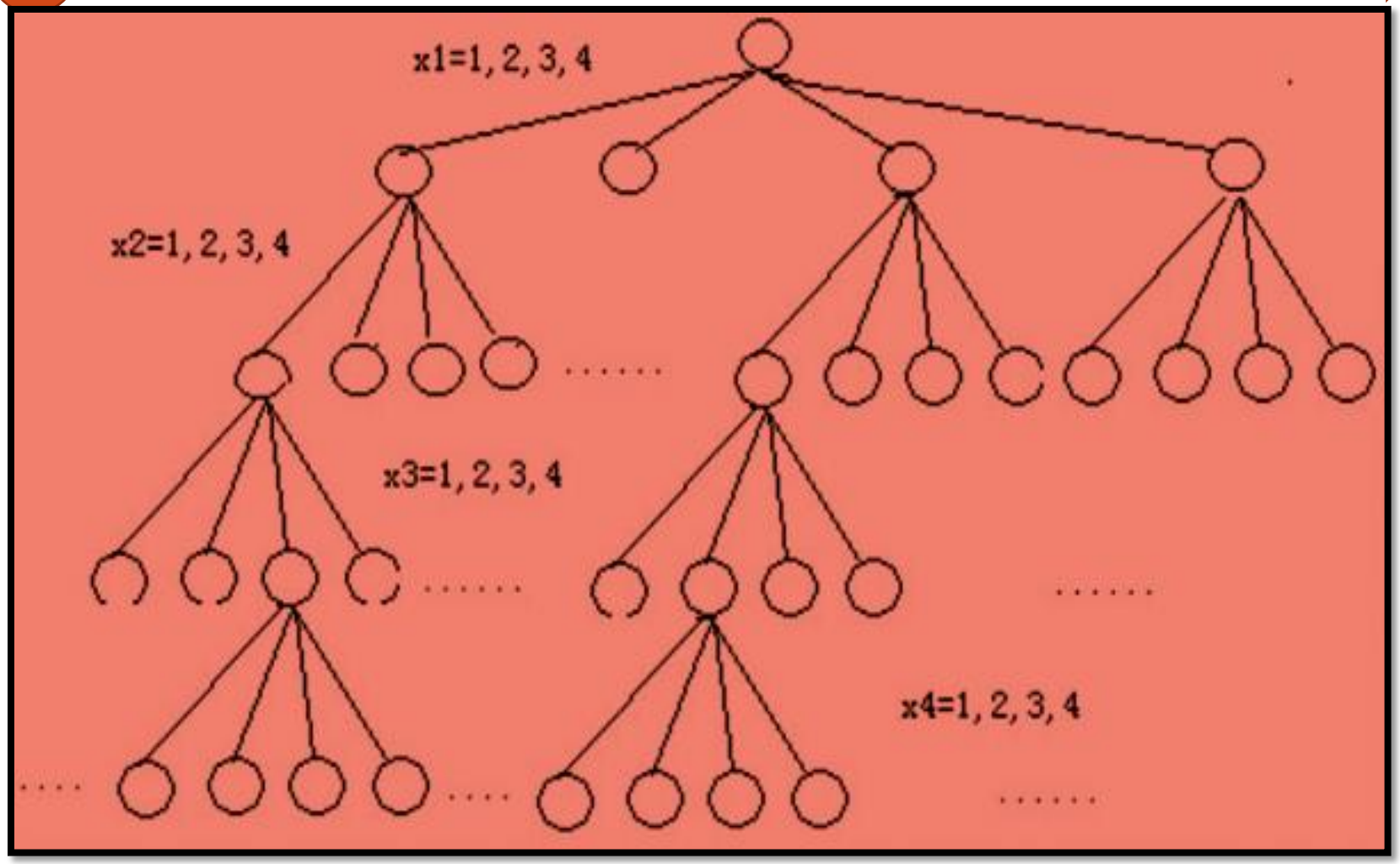


n后问题分析



- 解向量：$(x_1, x_2, \cdots, x_n)$
- 显约束：$x_i=1, 2, \cdots, n$
- 隐约束：
  - 不同列：
    - $x_i \neq x_j$
  - 不处于同一正、反对角线：
    - $|i-j| \neq |x_i-x_j|$ （如何推算？）

4皇后问题的状态空间树



递归算法

检测当前行为k，列为x[k]的皇后的位置是否合理（与前k-1行比较）

当前行为k，当前列为x[k]

```
bool Queen::Place(int k)
{
  for (int j=1;j<k;j++)
    if ((abs(k-j)==abs(x[j]-x[k]))||(x[j]==x[k])) return false;
  return true;
}
```

对角线

同列

```
void Queen::Backtrack(int  t)
{
  if (t>n) sum++;输出解;
    else
      for (int i=1;i<=n;i++) {
        x[t]=i;
        if (Place(t)) Backtrack(t+1);
      }
 }
```

行

设定列值，选取扩展结点

试探第t+1行

非递归算法：

K为当前行，当前列为x[k]

```
backtrack()
{ X[1]=0;  int k=1;
  While(k>0){
  x[k]+=1;
while((x[k]<=n)&&!(place(k)))
        x[k]+=1;
    if(x[k]<=n)
      if(k==n)   sum++;输出解;
    else{  k++;x[k]=0;}
  else  {x[k]=0; k--;}
}}
```

```
Place(int
{
  for(int j=1;j<k;j++)
    if((abs(k-j)==aba(x[j]-x[k]))
      ||(x[j]==x[k]))
  return false;
  return  true;
}
```

请给出4皇后的解，画出解空间树的搜索过程

没有回溯到树根

在k行的各列搜索

找到可行解

向下一层试探

回溯

4皇后问题的部分状态空间树