第6章 回溯法

《人工智能算法》

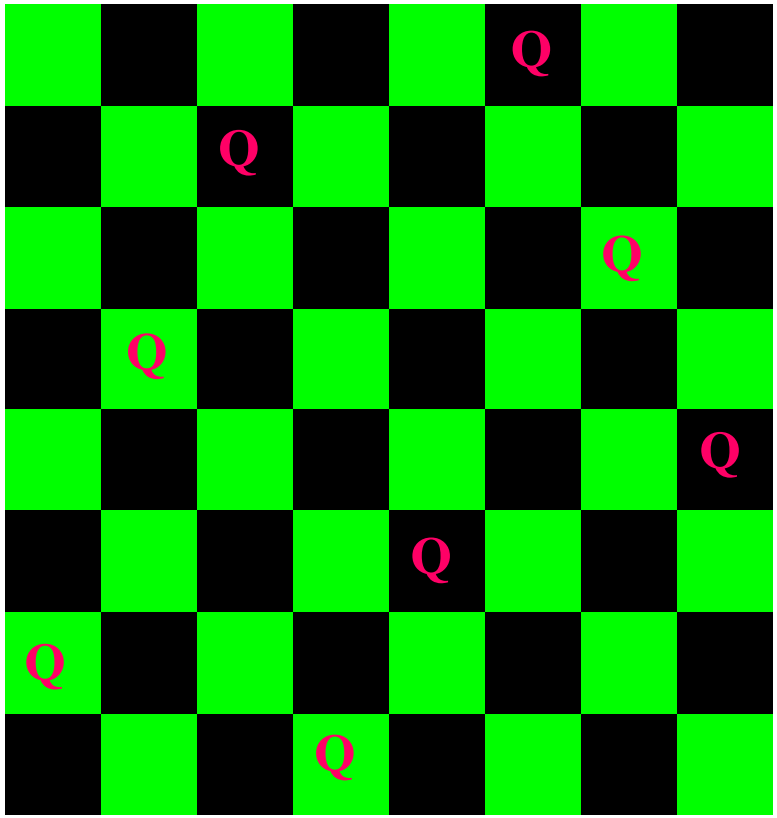
清华大学出版社
2022年7月

提纲

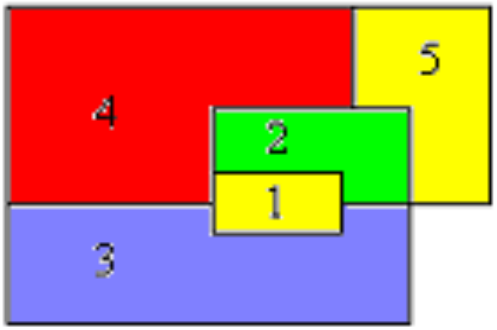
- 回溯法的基本思想
- n后问题
- 总结

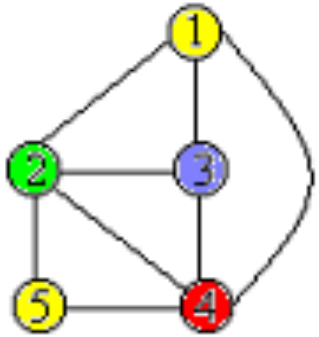
引例 (1)

- 8-皇后问题



- 图的*m*-着色问题

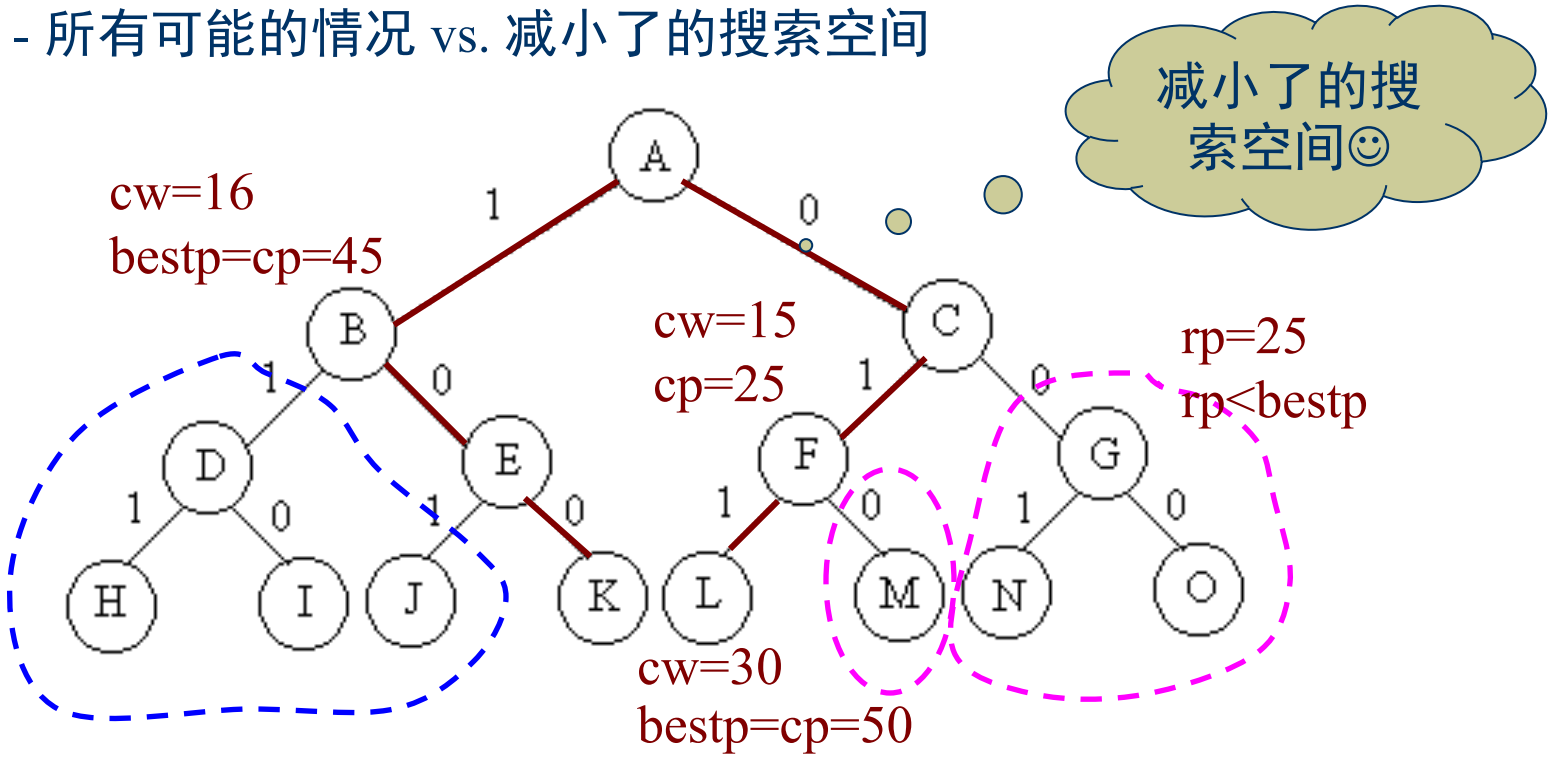




不存在用穷举搜索之外的方法来解决问题

引例 (2)

- **0-1背包问题的回溯分析**
  - n=3, w={16, 15, 15}, p={45, 25, 25}, c=30
  - 所有可能的情况 vs. 减小了的搜索空间



回溯法的基本思想 (1)

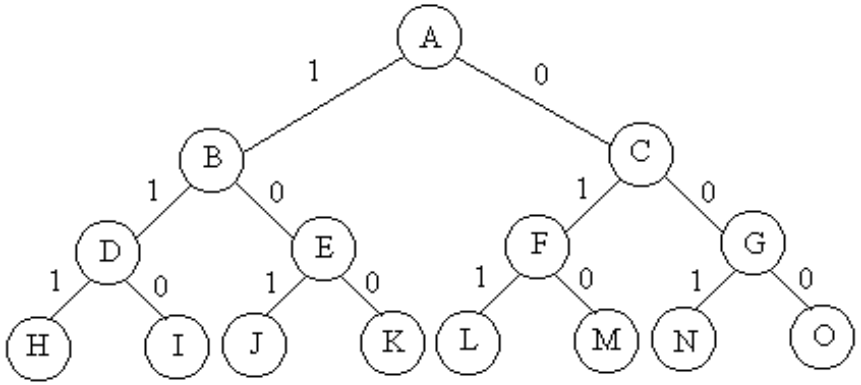
- **问题的提出**
  - 很多问题通过穷举搜索数量巨大但有限多个可能性可以获得问题的解
  - 很多问题不存在用穷举搜索之外的方法来解决问题的算法
  - 找出问题的解集、回答什么是满足约束条件的最佳解、……
- **回溯法概述**
  - 系统化的搜索，并且希望能将搜索空间尽可能减少
  - 有组织的搜索，常常可以避免搜索所有的可能性
  - 适用于解一些组合数（解空间）相当大的问题
  - 问题的解向量：回溯法希望一个问题的解能够表示成一个n元式$(x_1, x_2, \ldots, x_n)$的形式

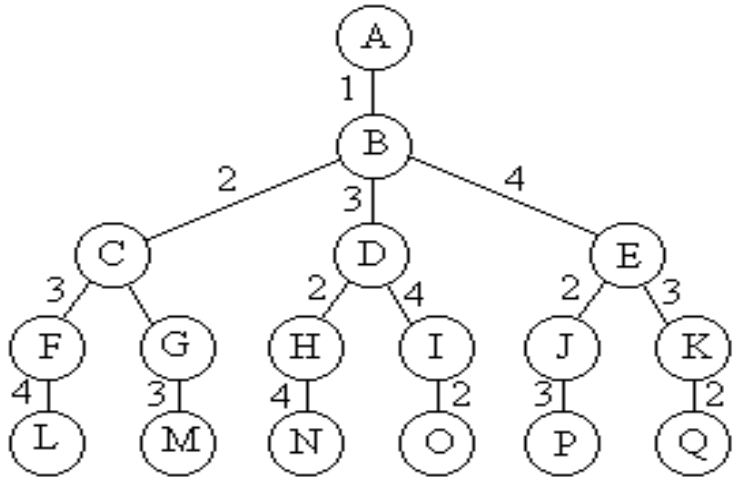
回溯法的基本思想 (2)

- **问题的解空间**
  - 显约束：对分量x_i的取值限定
  - 隐约束：为满足问题的解而对不同分量之间施加的约束
  - 解空间：解向量满足显式约束条件的所有元组；将解空间组织为树
- **问题状态的生成**
  - 扩展节点、活节点、死节点
  - 深度优先的问题状态生成
- **回溯法提出**
  - 避免无效搜索、提高效率——利用**约束函数**和**限界函数**来处死那些实际上不可能产生所需解的活节点，以减少问题的计算量
  - **具有限界函数的深度优先生成法——回溯法**

回溯法的基本思想 (3)

- 子集树与排列树





遍历子集树需$O(2^n)$计算时间（最坏）遍历排列树需要$O(n!)$计算时间（最坏）

```
backtrack (int t)
  if t>n then output(x)
  else
    for i←0 to 1 do
      x[t]←i
      if (legal(t))
      backtrack(t+1)
    end for
```

```
backtrack (t)
  if t>n then output(x)
  else
    for i←t  to n do
      swap(x[t], x[i])
      if(legal(t))
      backtrack(t+1)
      swap(x[t], x[i])
    end for
```

回溯法的基本思想 (4)

- **求解思路**
  - 确定解空间结构
  - 深度优先搜索解空间+剪枝函数
- **常用剪枝函数**
  - 用**约束函数**在扩展节点处剪去不满足约束的子树（问题本身的约束）
  - 用**限界函数**剪去得不到最优解的子树（相对于已得到的解）
- **主要特征**
  - 不需要存储整棵搜索树，只需存储根到当前扩展节点的路径
  - 设$h(n)$为从根到叶的最长路径长度
  - 对于子集树解空间 —— $O(2^{h(n)})$

    对于排列树解空间 —— $O((h(n))!)$

提纲

- 回溯法的基本思想
- n后问题
- 总结

n后问题 (1)

- **问题**
  - 在$n \times n$格的棋盘上放置彼此不受攻击的n个皇后
  - 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
- **算法思想**
  - 解空间：完全n叉树
  - 解向量：$(x_1, x_2, \ldots, x_n)$
  - 每行放一个皇后，$x[i]$表示皇后i被放在第i行$x[i]$列，约束：

(1) $x[i] \neq x[j], j \in [1, i-1]$

(2) $|i-j| \neq |x[i]-x[j]|, j \in [1, i-1]$

n后问题 (2)

n皇后问题的回溯算法

```
boolean place (k)
  for j=1 to k do
    if |k−j|=|(x[j]−x[k]|) or |x[j]=x[k]| then
      return false
    return true
  end for

void backtrack (t)
  if t>n then sum←sum+1
  else
    for i=1 to n do
      x[t]←i
      if place(t) then backtrack(t+1)
    end for
```

- 初始化：
  for i←0 to n do
    $x[i]$←0
- 调用：
  backtrack(1)

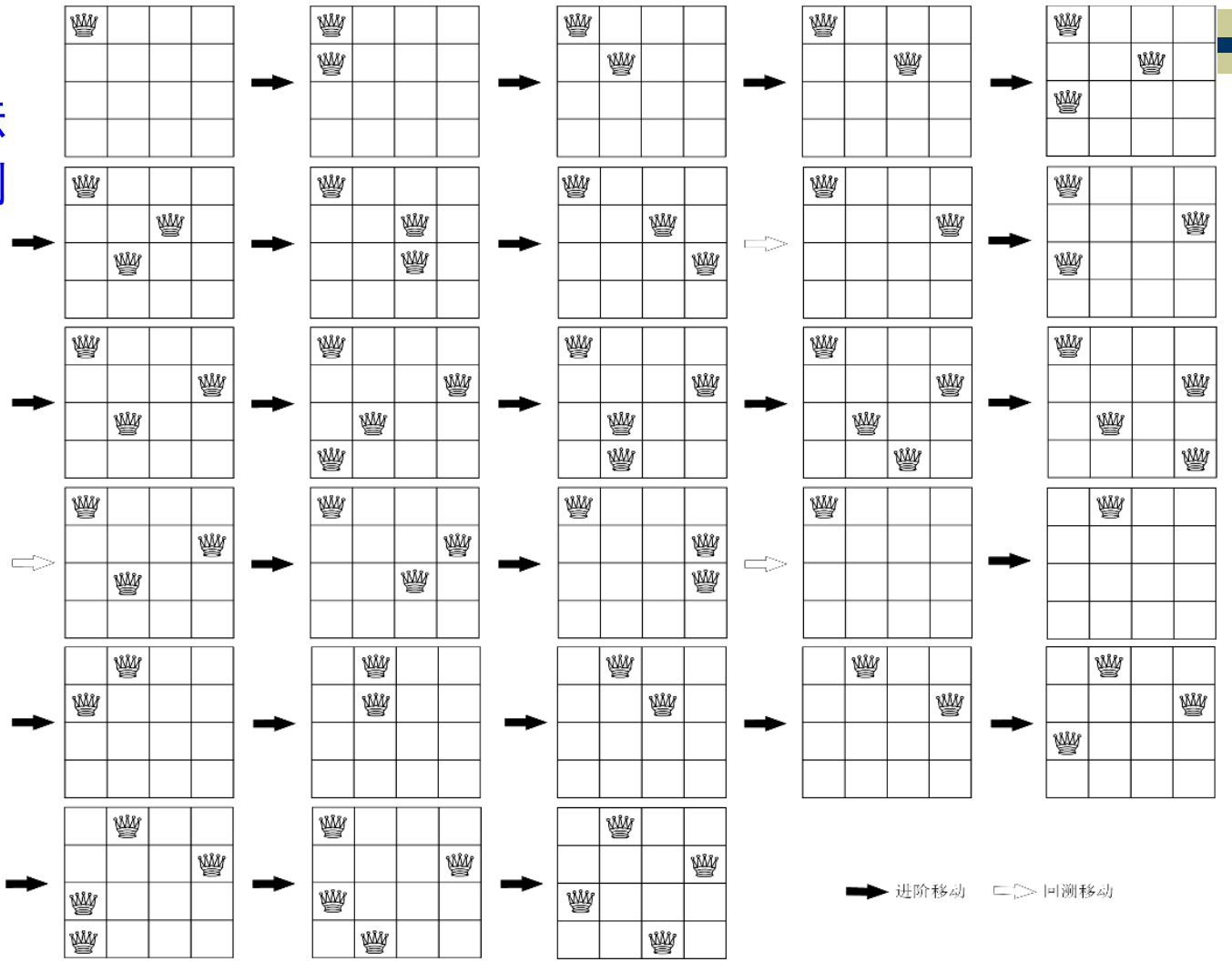
时间复杂度：$O(n^n)$

和蛮力法相比？

*n*后问题 (3)

- 4-后问题的回溯法求解示例

生成子集树中的27个节点

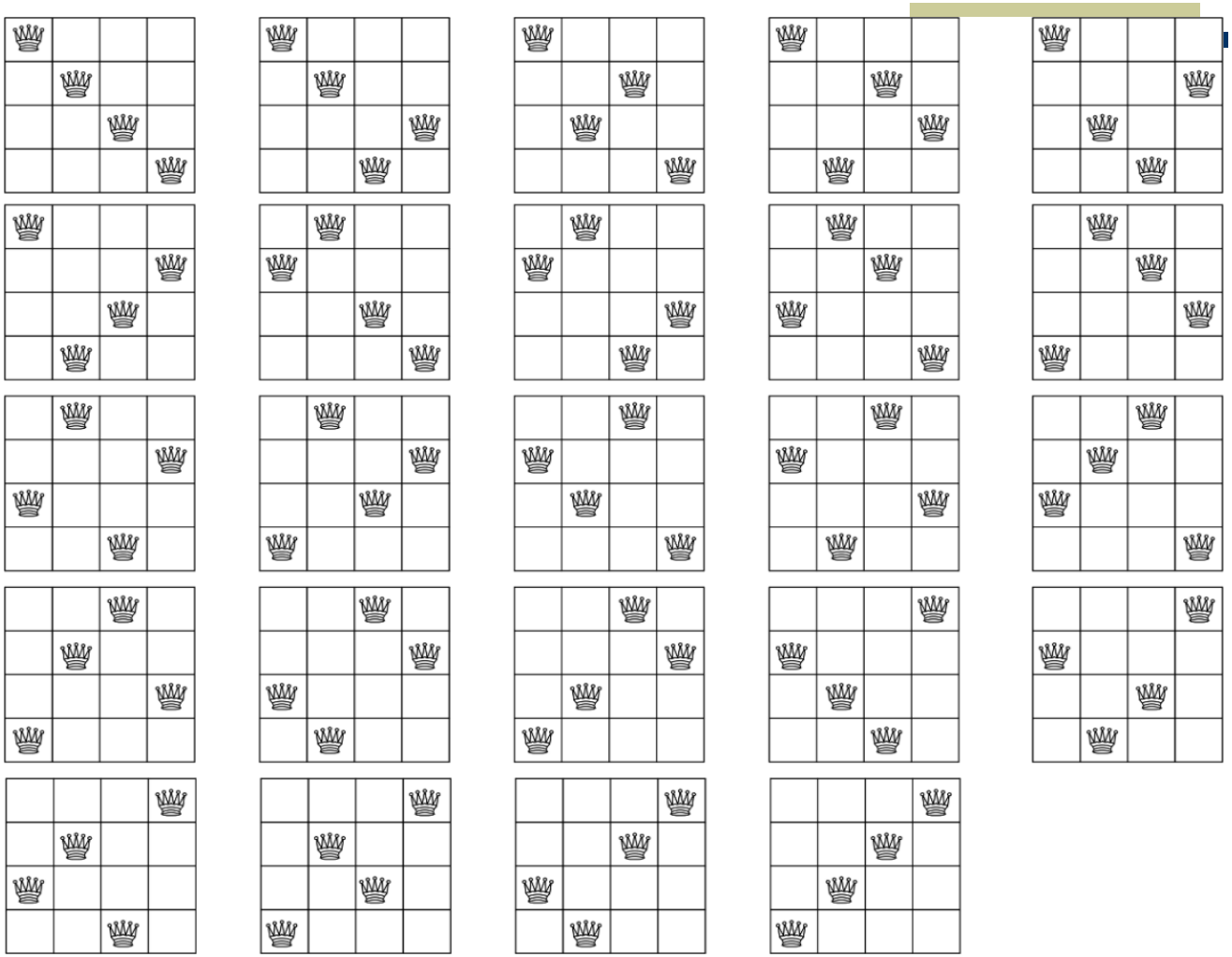


*n*后问题 (4)

- 4-后问题的蛮力法求解示例

需4!次搜索便可找到问题的一个解

回溯法与蛮力法相比，优势何在？



n后问题 (5)

回溯法与蛮力法的时间复杂度比较：

- 使用递归回溯法求解n-后问题的时间复杂度为$O(n^n)$，使用蛮力法求解n-后问题的时间复杂度为$O(n!)$，但与蛮力法相比，递归的回溯法可通过剪枝函数将不需遍历的子树删减。
- 当n值不同时，蛮力法和回溯法的比较有不同的结论。例如，当$n = 4$时子集树包含341个可能节点，采用回溯法在生成子集树中的27个节点后，便可得到问题的一个解，如图6.6所示。蛮力法的时间开销低于回溯法，仅需4!次搜索便可找到问题的一个解。
- 当n较大时，回溯法相对于蛮力法的优势才显现出来，例如，$n = 8$时，回溯法仅需114次搜索就可找到问题的一个解，而蛮力法需要8!次搜索。

结论：

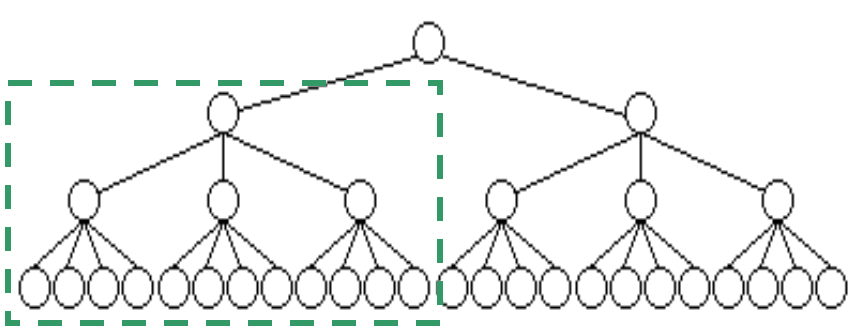
- 当n较小时，蛮力法优于回溯法；
- 当n较大时，回溯法优于蛮力法；
- n越大，回溯法的优势越显著

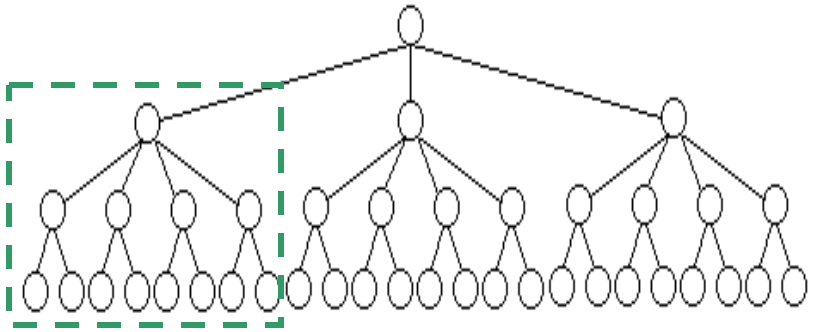
提纲

- 回溯法的基本思想
- n后问题
- 总结

总结 (1)

- **回溯法效率分析**
  - 好的约束函数能显著地减少所生成的节点数☺，往往计算量较大☹
  - 考虑生成节点数与约束函数计算量之间的折衷
- **重排原理**
  - 尽可能减小搜索空间
  - 对于许多问题而言，在搜索试探时选取$x[i]$的值顺序是任意的
  - 在其他条件相当的前提下，让可取值最少的$x[i]$优先





总结 (2)

- 回溯法的基本思想
  - 解空间
  - 深度优先搜索+剪枝函数
  - 时间复杂度分析
- 回溯法的重要算法实例：n后问题

结语

谢谢！