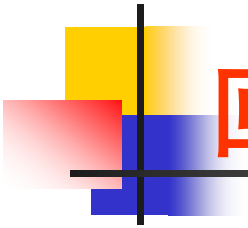
算法设计
——回溯法与分枝限界法

陈卫东

chenwd@scnu.edu.cn

华南师范大学计算机学院

2021-10



回溯法（Backtracking）

1 引言

2 回溯算法的基本框架

3 回溯法求解问题举例

4 回溯法编程举例

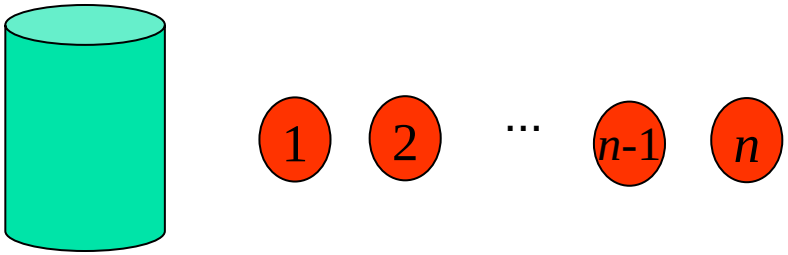
5 回溯法的效率分析

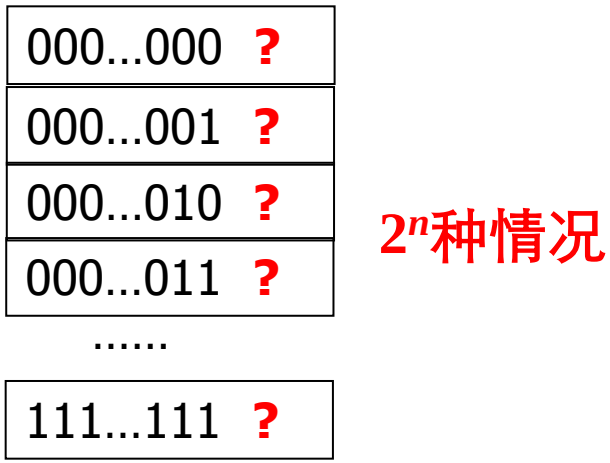
1 引言

【例】考虑用穷举法求解0-1背包问题

- **穷举法是很多问题最直接的求解方法。如何用穷举法求解该问题?**

——所有物品子集构成了问题解集。依次考虑每个物品子集，对那些放得下的物品子集记录下其价值，由此找到最大价值物品子集。

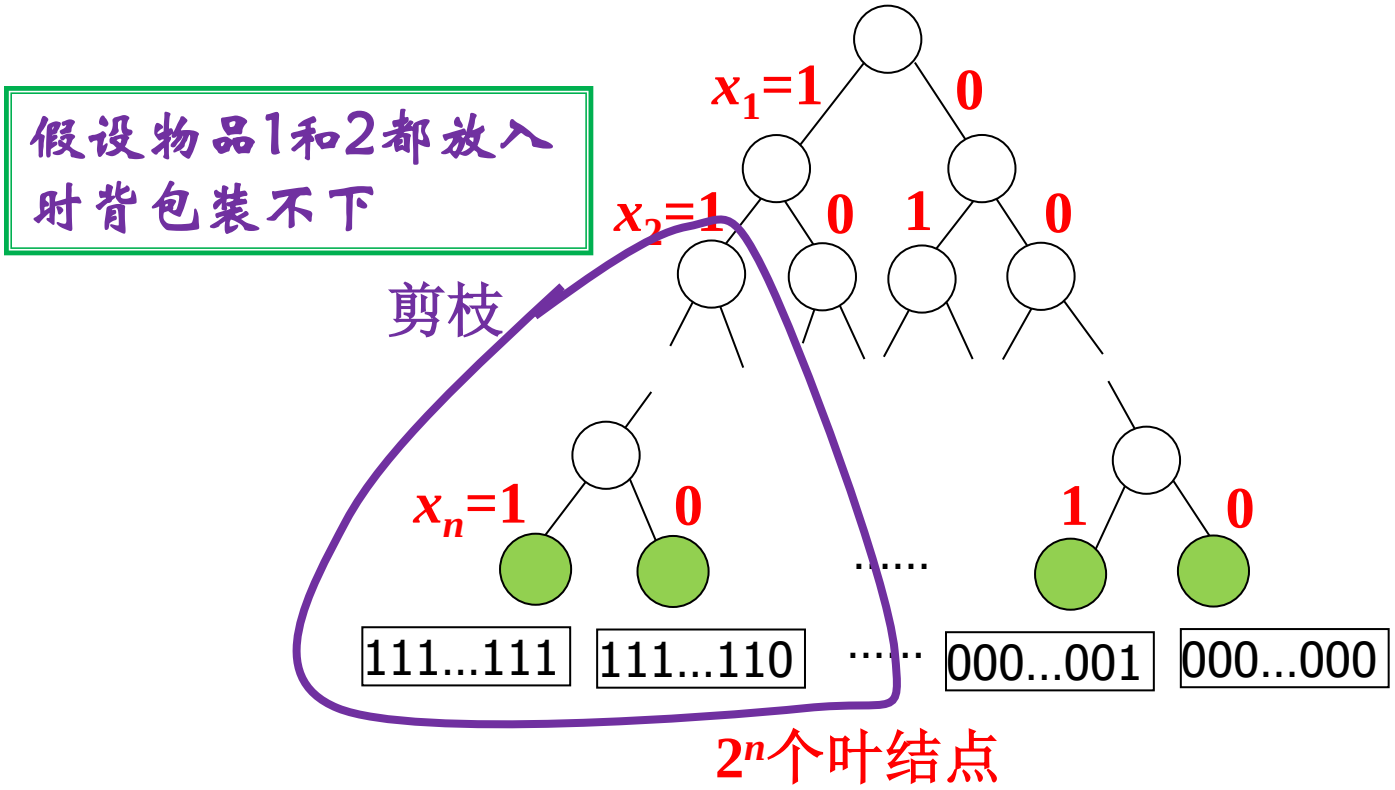




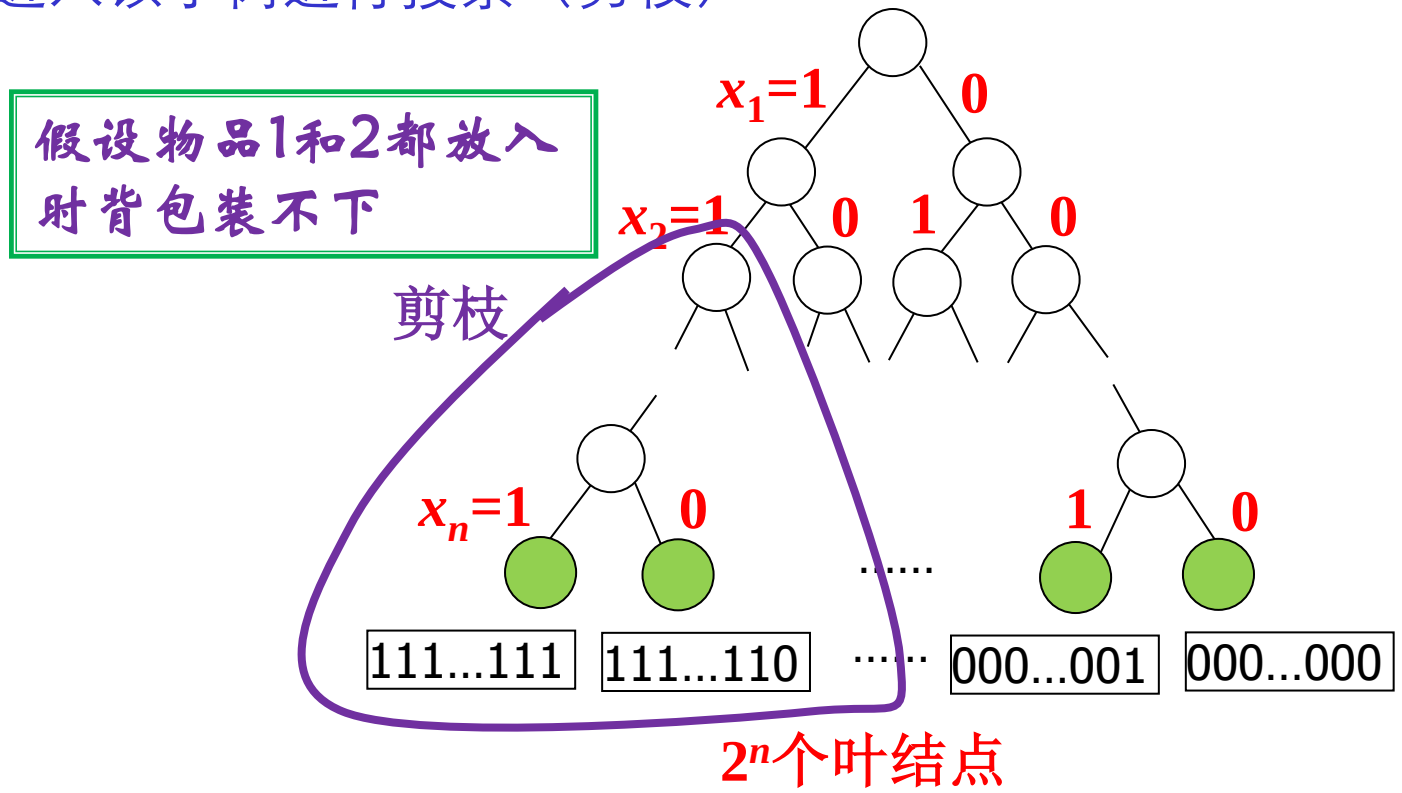
2^n种情况

- 如何提高穷举法的效率？比如当某些物品背包放不下时，包含这些物品的任何物品集也一定放不下，因此无需再考虑它们。如何实现这一点？

——将解集组织在树上，对树进行搜索，同时用剪枝操作来提高搜索效率



- 聪明人的穷举方法：回溯法（树的深度优先搜索+剪枝）
  - 共2^n个物品子集构成问题解集，将其组织在一棵高为n的二叉树上。（根节点表示求解的开始状态，树中每个其他节点表示问题求解的一个中间状态，对应一个部分解，而叶子节点表示一个完整的解）
  - 对解树进行深度优先搜索（从根节点开始）
  - **约束剪枝：**搜索到某个节点时，如果对应的部分解不可行，则无需进入该子树进行搜索（剪枝）
  - **优化剪枝：**搜索到某个节点时，该子树下面没有更好的解，则无需进入该子树进行搜索（剪枝）



1 引言

回溯法素有“通用解题法”之称。使用它可以系统地搜索一个问题的所有解或者一个解。它是一种既带有系统性又带有跳跃性的搜索方法。

- **系统性**——深度优先搜索包含解空间中所有解的树；
- **跳跃性**——剪枝。

演示

2 回溯算法的基本框架

- 回溯法的基本思想

  树的深度优先搜索+限界（剪枝）

- 基本要素
  - 解空间的树形表示
  - 剪枝操作（约束剪枝、优化剪枝）

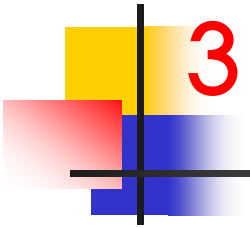
2 回溯算法的基本框架

- **设计回溯算法的基本步骤**
  - 将解空间表示成一棵树T（解空间树）。
  - 定义剪枝操作（需考虑约束条件和目标值优化两方面）。

- **回溯算法形式及终止条件**
  - **算法形式**：递归形式和非递归形式（迭代形式）
  - **算法终止的条件**：

    当求所有解时，回溯到根结点且根的所有子树都已被搜索遍才结束。

    当求一个解时，只要搜索至一个问题解即终止。



3 回溯法求解问题举例

[图的m着色问题]

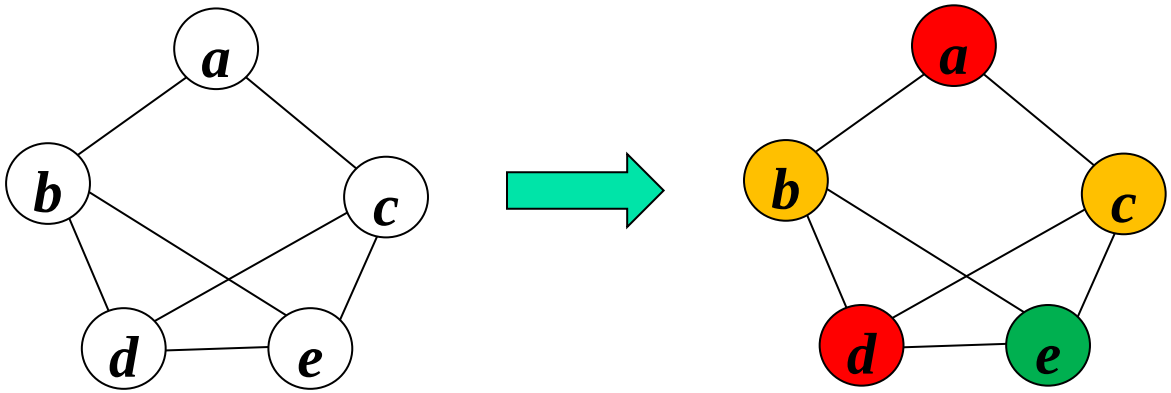
[n皇后问题]

[图的m着色问题]

图的m着色问题是指，任给简单无向图G及正整数m ，问是否存在对图G的顶点的合法m着色?

若一个图最少需要m种颜色才能使对图进行合法着色，则称m为该图的色数。

图的着色优化问题是指，求图的色数m的问题 。



[图的m着色问题]

- 回溯法的基本要素
  - 解的形式:

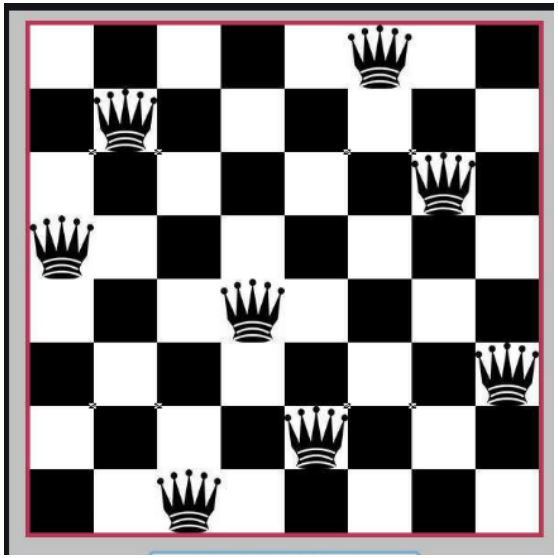
    $X=(x_1,x_2,\ldots,x_n)$，其中 $1\le x_i\le m$，$(1\le i\le n)$

    解的树形表示：满m叉树
  - 剪枝操作:

    第i个结点着第x_i号颜色可行⇔与第i个结点有边相连的任何结点j满足$x_i\neq x_j(1\le j<i)$
- 回溯算法
  - 递归形式，迭代形式
- 算法效率：$O(nm^n)$

[*n*皇后问题]

*n*皇后问题要求在一个*n*×*n*格棋盘上放置*n*个皇后，使得她们彼此不受到攻击。按照国际象棋规则，一个皇后可以攻击与之处在同一行或同一列或同一斜线上其它任何棋子。因此，*n*皇后问题等价于要求在一个*n*×*n*格棋盘上放置*n*个皇后，使得任何2个皇后不能被放在同一行、同一列、同一斜线上。



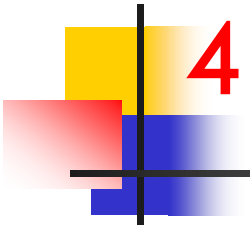
[n皇后问题]

- 回溯法的基本要素
  - 解的形式:

    $X=(x_1,x_2,\ldots,x_n)$，其中 $1\le x_i\le n$，($1\le i\le n$)

    解的树形表示：满n叉树
  - 剪枝操作：

    第i个皇后放在x_i列可行$\Leftrightarrow x_i\neq x_j$且$|i-j|\neq|x_i-x_j|$ $(1\le j<i)$
- 回溯算法
  - 迭代形式
- 算法效率：$O(nn^n)$



4 回溯法编程举例

[实例]

- 产生一个集合的所有的子集
- 产生n个元素的所有排列
- n皇后问题

5 回溯法的效率分析

- 回溯法的最坏时间和空间复杂度（理论分析）
  - **时间性能:**

    ——最坏情况要搜索到整个解空间，因此复杂度是指数型。若剪枝操作有力，则平均性能往往很好。

  - **空间性能**

    ——回溯法的附加空间为$\mathbf{O}(h)$,其中h为搜索树的高度。

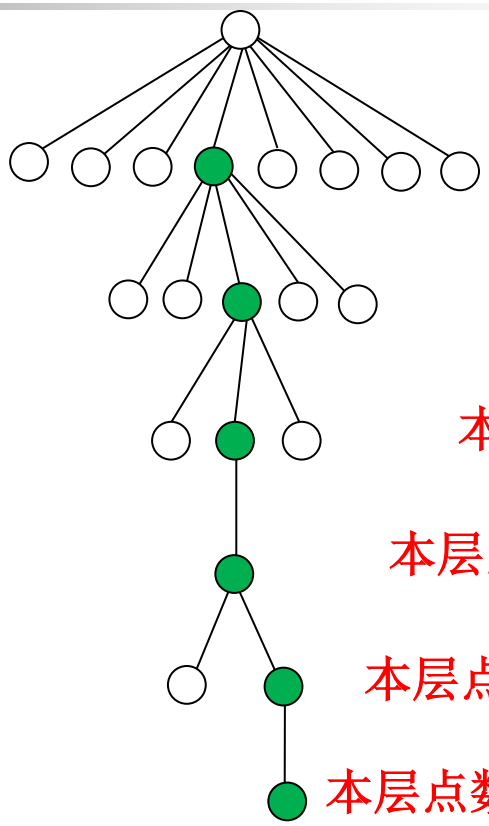
5 回溯法的效率分析

- 平均时间效率估计（实验分析）
  - 影响算法效率的因素
  - 蒙特卡洛法的基本思想

5 回溯法的效率分析

			1				
					2		
		3					
				4			
						5	
6							

(8,5,3,1,2,1)=769



$m_1=8$ 本层点数 m_1

$m_2=5$ 本层点数 m_1m_2

$m_3=3$ 本层点数 $m_1m_2m_3$

$m_4=1$ 本层点数 $m_1m_2m_3m_4$

$m_5=2$ 本层点数 $m_1m_2m_3m_4m_5$

$m_6=1$ 本层点数 $m_1m_2m_3m_4m_5m_6$

点数共计：

$1+m_1+m_1m_2+m_1m_2m_3+m_1m_2m_3m_4+\ldots=768$

5 回溯法的效率分析

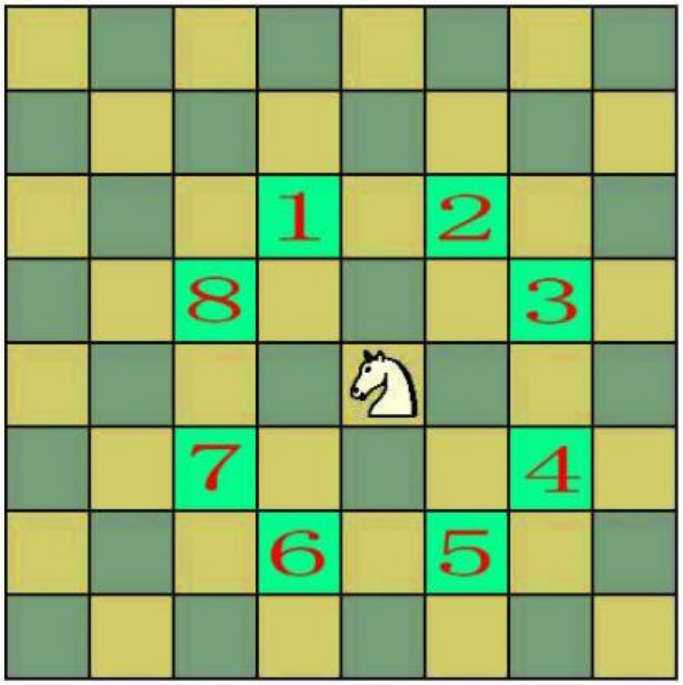
		1					
					2		
	3						
						4	
5							
			6				
							7
				8			

(8,5,3,2,2,1,1,1)=2329

【例】 骑士问题

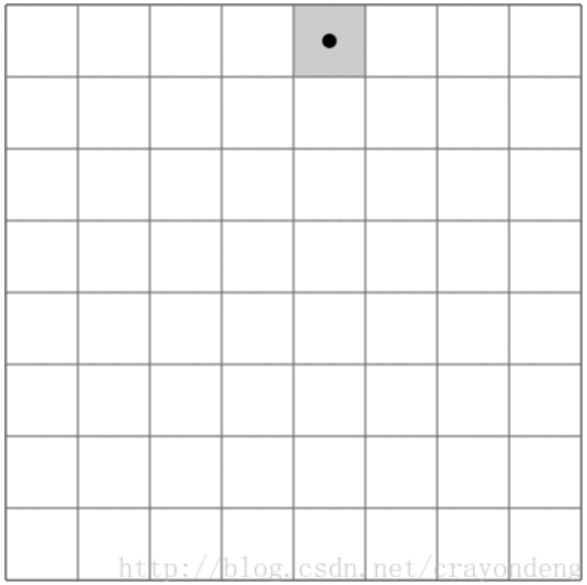
➢ 问题——国际象棋中的骑士问题

任意给定马的初始位置，试找出一种马能跳到每个格恰好一次且最后回到初始位置的周游棋步。



【例】 骑士问题

➢ 问题——国际象棋中的骑士问题



55	58	45	26	1	60	43	10
46	3	56	59	44	9	64	61
57	54	25	2	27	62	11	42
4	47	20	29	22	13	8	63
53	30	5	24	7	28	41	12
48	19	50	21	14	23	38	35
31	52	17	6	33	36	15	40
18	49	32	51	16	39	34	37

- 在8×8、20×20、100×100棋盘上简单搜索方法(回溯法)求解效率如何？
- 如何提高求解效率?

【例】 骑士问题

- 回溯法的基本要素
  - 解空间的树形表示：

    解的形式：$(x_1,x_2,\ldots,x_n)$，其中 $1\le x_i\le 8$，$(1\le i\le n)$

    解空间树：满8叉树
  - 剪枝操作：

    第i步朝x_i方向走可行 $\Leftrightarrow$ x_i方向是未跳过的空格
  - (儿子)活结点X的成本函数（函数值越小优先级越高）

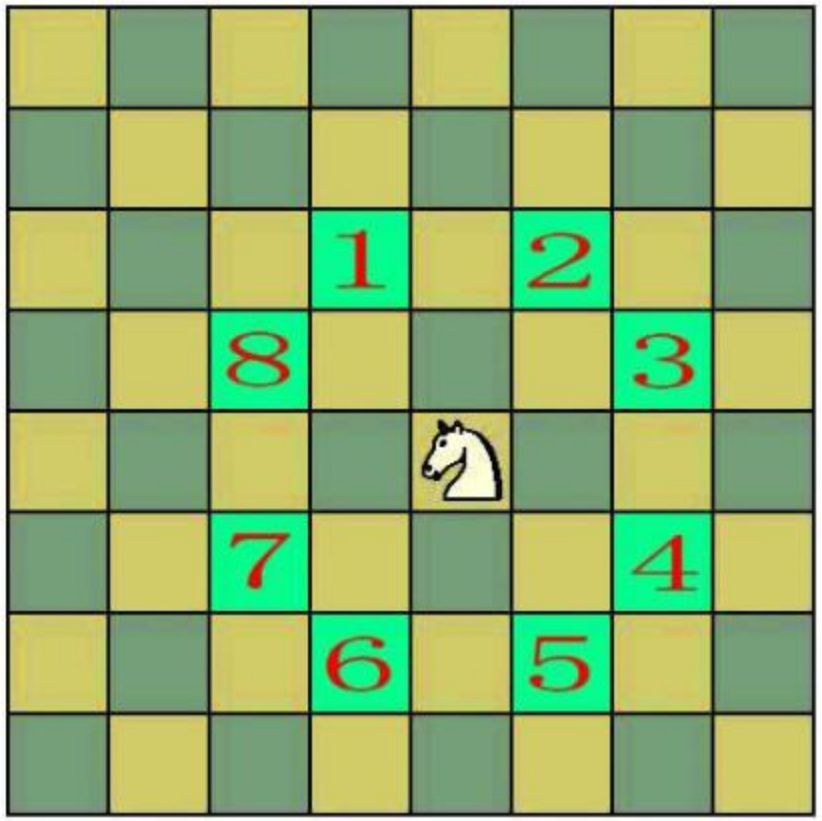
    $C(X)$= 此刻能跳到方向X对应格的机会次数

算法——骑士周游问题的智能搜索算法

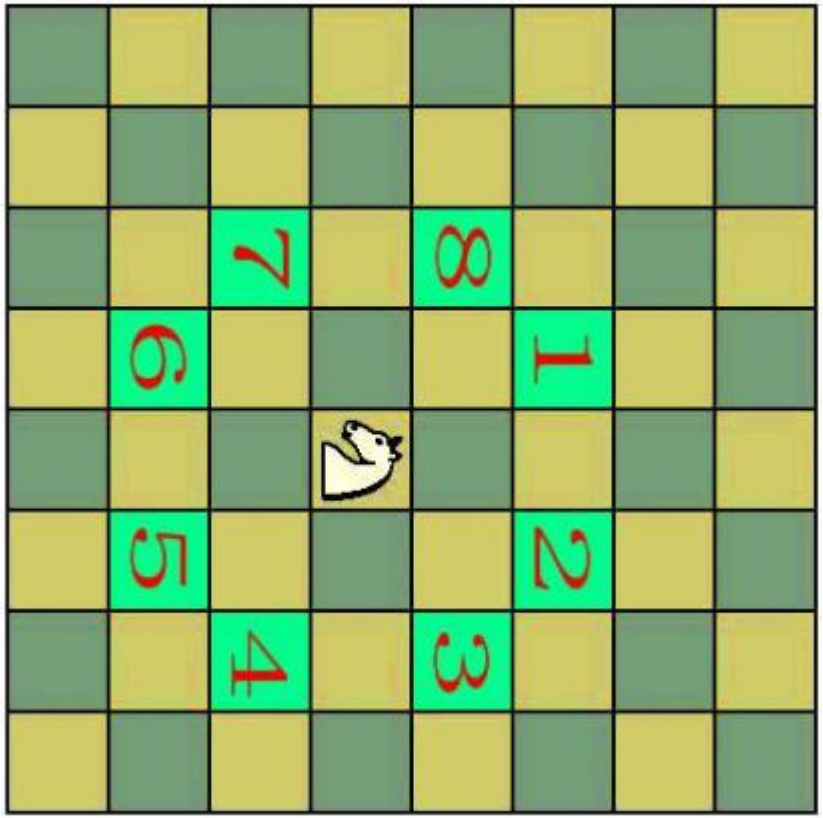
- **启发式策略：**
  - ✓ 平移策略：从中间位置开始容易得到结果，将该结果进行平移变换可得到任意位置开始的结果
  - ✓ 方向策略：机会少的方向优先
  - ✓ 离心策略：偏离中心位置的方向优先
- **实验效果：**
  - ✓ 简单搜索算法： 8×8的棋盘有时需要30分钟
  - ✓ 智能搜索算法： 3000×3000的棋盘能在瞬间出结果

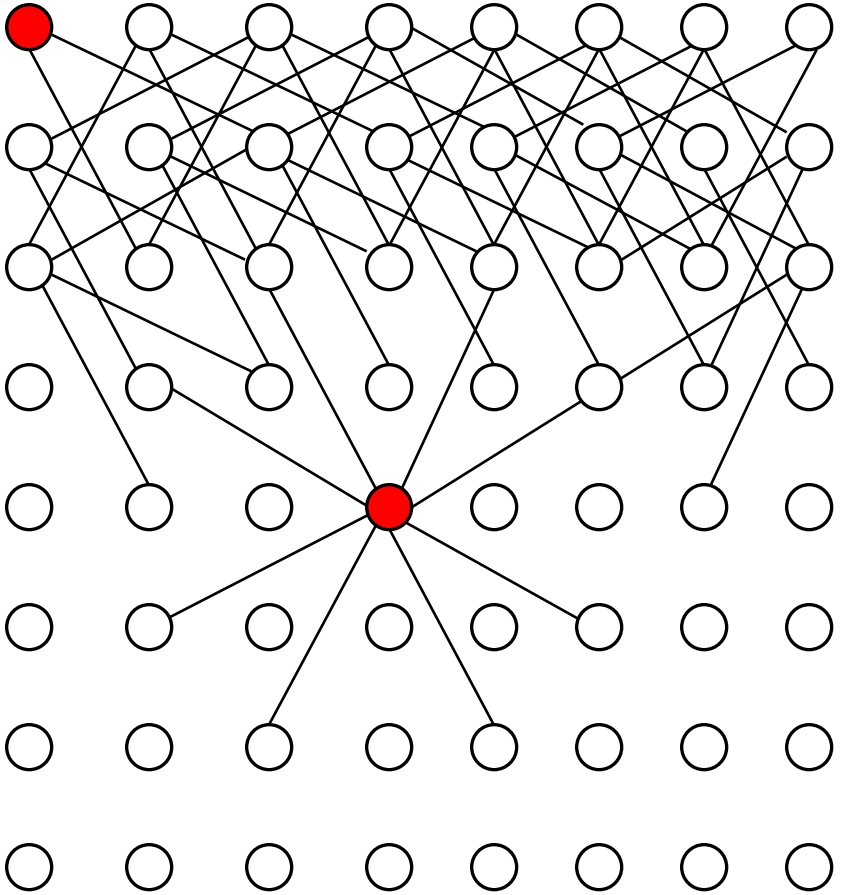
算法——骑士周游问题的智能搜索算法

- 偶数阶棋盘，为何如果从一个位置出发有解，则从任何位置出发都有解?
- 奇数阶棋盘，为何无解？



数学模型——无向图（二部图）的Hamilton圈问题





问题思考

问题1：回溯法是一种怎样的搜索方法？有什么特点？

回溯法可以系统地搜索一个问题的解空间，是一种系统性又有跳跃性的搜索方法。

- 系统性——使用深度优先搜索法系统搜索解空间树
- 跳跃性——剪枝

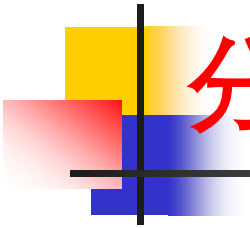
问题2：将深度优先搜索换为宽度优先搜索会有什么特点？

演示

分枝限界法的特点

分枝限界法可以系统地搜索一个问题的解空间，它也是一种既带有系统性又带有跳跃性的搜索方法。

- 系统性——宽度优先或优先队列搜索法系统地搜索解空间树
- 跳跃性——剪枝



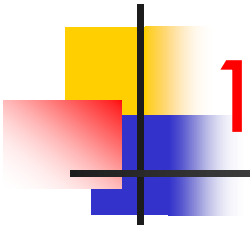
分枝限界法（Branch and Bound）

1 分枝限界法的基本思想

2 分枝限界法的基本框架

3 应用举例

4 小结



1 分枝限界法的基本思想

- 分枝限界法的基本思想

E-结点保持到变为死结点的搜索法 + 剪枝操作

1 分枝限界法的基本思想

- 分枝限界法的种类
  - **先进先出（FIFO）分枝限界法**

    宽度优先搜索+剪枝
  - **先进后出（FILO）分枝限界法**

    D-搜索+剪枝
  - **最小成本（LC）分枝限界法**

    LC-搜索+剪枝

    （这种搜索的目标是要搜索一个具有最小成本的解）

1 分枝限界法的基本思想

- 分枝限界法求解的问题类型
  - 存在性问题
  - 最优化问题

2 分枝限界算法的基本框架

- **基本要素**
  - **解空间的树形表示**

    0/1背包问题：子集树

    旅行商问题：排列树

    n皇后问题：满n叉树
  - **剪枝操作**

    约束剪枝，优化标剪枝
  - **活结点成本函数（即定义结点的优先级别）**

    决定了挑选活结点的优先级

2 分枝限界算法的基本框架

- 设计分枝限界法的基本步骤
  - **将解空间表示成一棵树T（解空间树）**

    求解问题转化为在树T中搜索解对应的树结点
  - **定义剪枝操作**

    考虑约束条件和目标值优化两方面
  - **定义每个活动结点成本函数**

    决定结点的优先级

2 分枝限界算法的基本框架

- **分枝限界算法形式及终止条件**

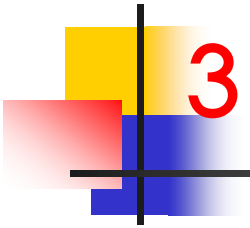
  从树T的根结点开始，用宽度优先搜索或者优先队列搜索方法来搜索该树，并跳过肯定不包含问题解对应的结点的子树的搜索（剪枝），以提高效率。

  - **算法形式：** 一般是迭代形式。
  - **算法终止的条件：**

  求存在性问题时，找到问题的一个解即可结束；在求优化问题时，一般要求活结点表为空时才结束，但是在满足一定条件时可在找到一个解即可结束（该解即为最优解）。

2 分枝限界算法的基本框架

- 分枝限界法的特性
  - **时间性能：** 最坏的情况要搜索整个解空间，是复杂度是指数型。但如果启发式信息强且剪枝操作有力的话，平均性能往往很好。
  - **空间性能：** 为了保存活结点往往需要较大的空间开销。

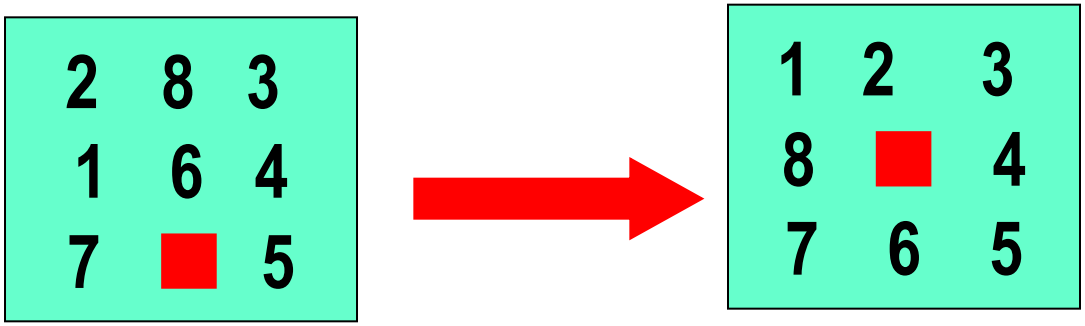


3 应用举例

[九宫重排问题]

[旅行商问题]

[九宫重排问题]



初始状态　　目标状态

问题要求：从初始格局经过尽可能少的棋步到达目标格局。

分枝限界法的基本要素

- 解空间的树形表示：

  $(x_1,x_2,\ldots)$，其中 $1\le x_i\le 4$，$(1\le i\le n)$

  解空间树：满4叉树

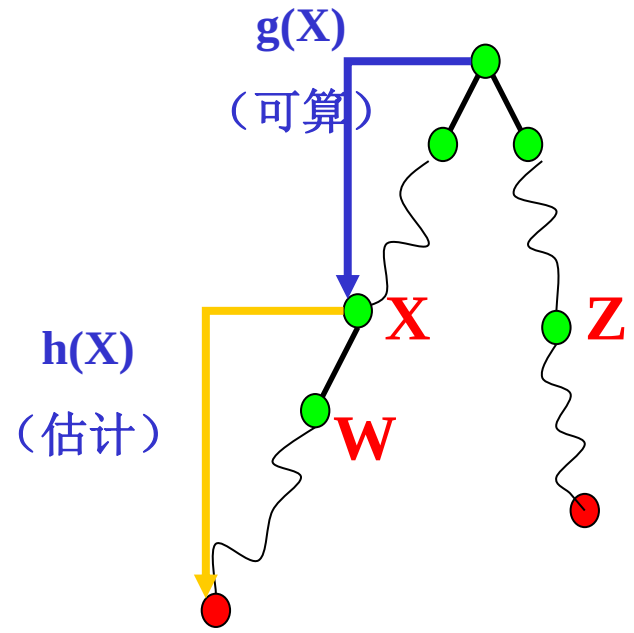
- 剪枝操作：

  第i步空格朝x_i 方向走是可行的

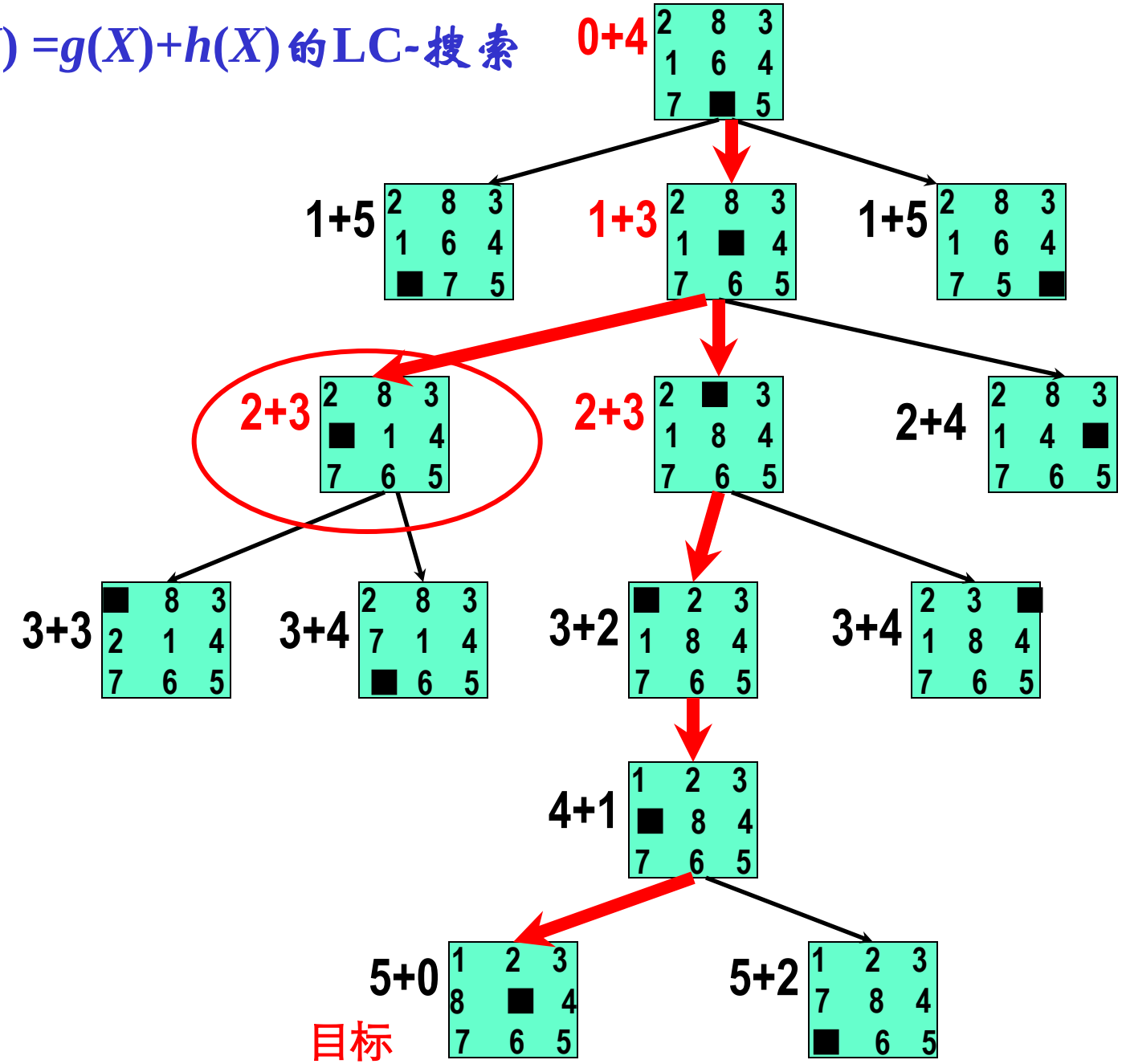
  $\Leftrightarrow$ 空格的x_i 方向不是边界，且x_i 方向不是第i-1步的逆方向

- 优先队列（值越小优先级越高）

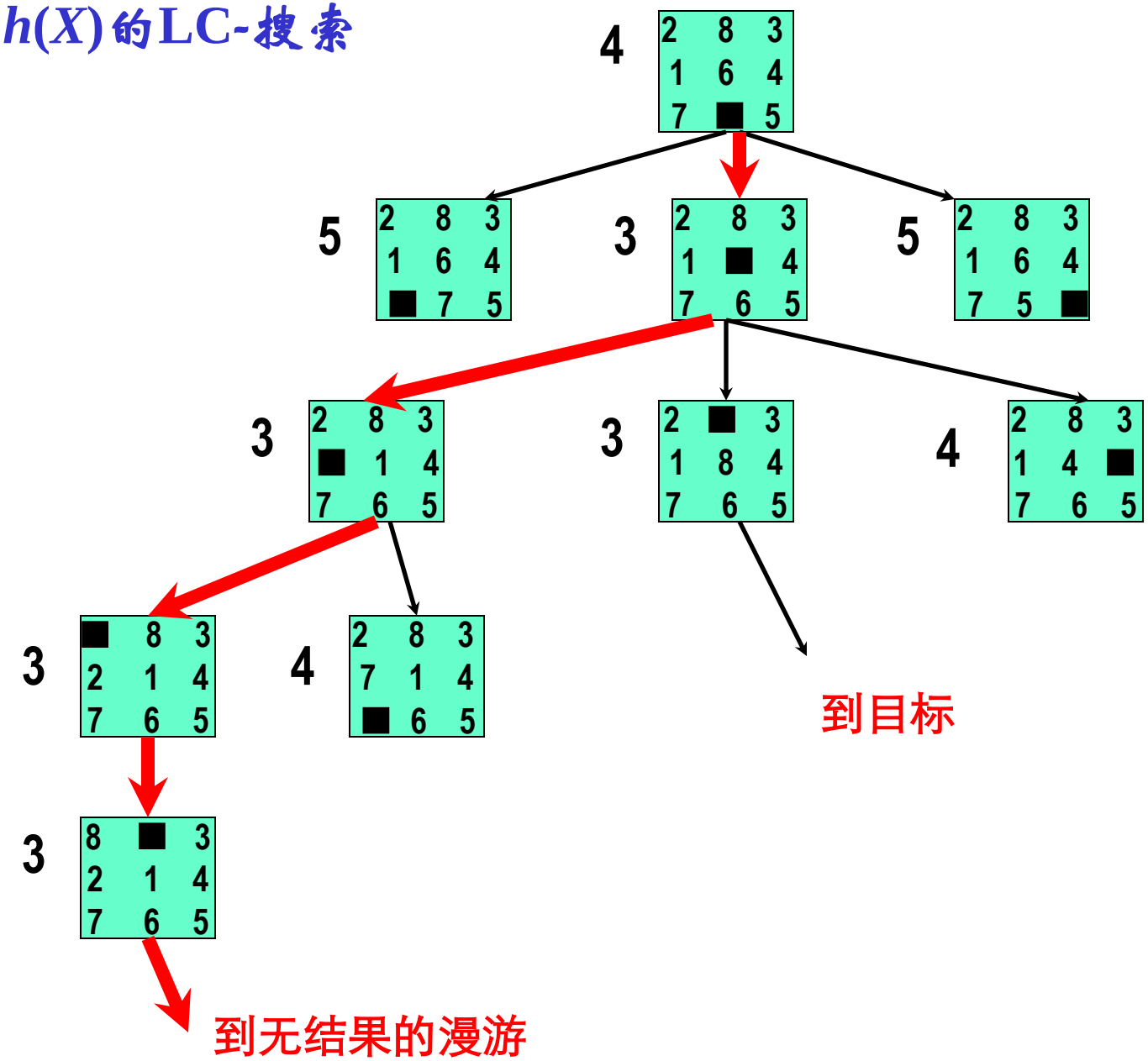
  $C(X)= g(X) + h(X)$



使用$C(X)=g(X)+h(X)$的LC-搜索



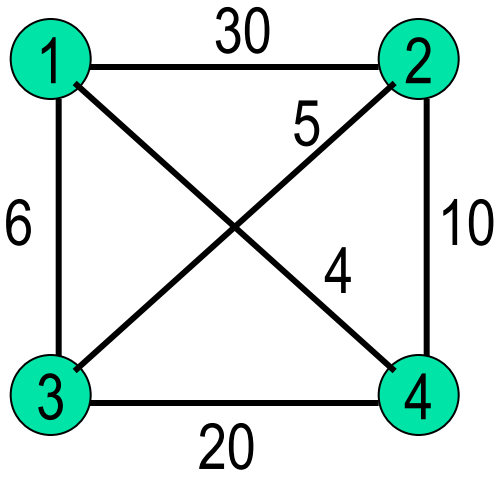
使用$C(X)=h(X)$的LC-搜索



[旅行商问题]

在这个问题中，给出一个n个顶点（有向或无向）网络G=<V,E>，要求找出一个包含所有n个顶点的具有最小耗费的基本回路。任何一个包含网络中所有的n个顶点的基本回路被称作一个周游（tour）。在旅行商问题中，要求找到一条最小耗费的周游。

右图给出了一个四个顶点的网络。这里顶点少，容易观察到最小耗费周游是1，3，2，4，1，最小耗费是25。求解该问题的一般的方法有，动态规划、回溯法和分枝限界法等等。

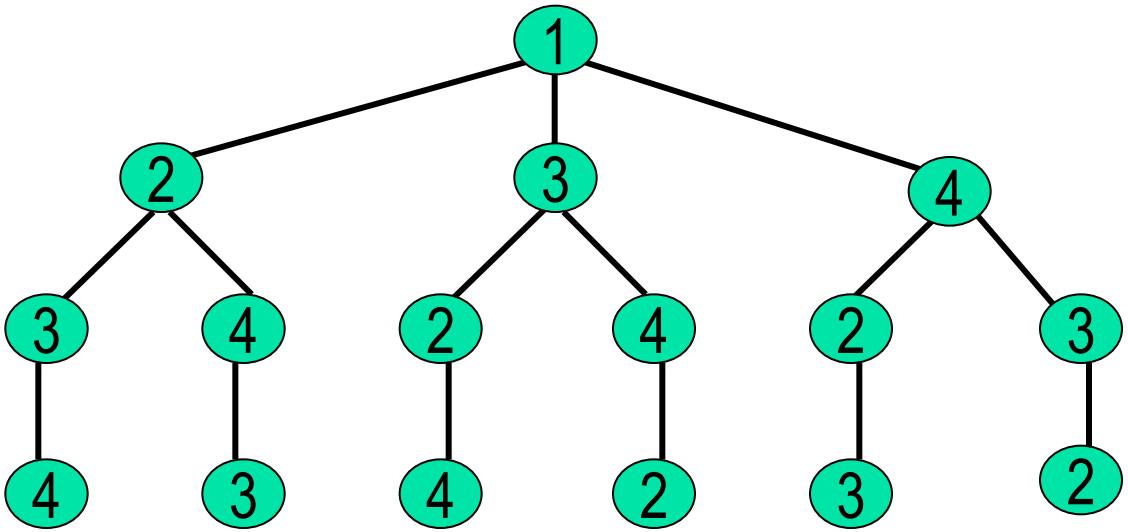


分枝限界法的基本要素

- 解空间的树形表示:

  $X=(x_2,x_3,\ldots,x_n)$为2,3,…,n的一个排列

  解空间树：排列树



分枝限界法的基本要素

- 约束剪枝:

  $x_i=m$不可行⇔ x_i与x_{i-1}之间没有边

- 活结点成本函数—目标值下界（值越小优先级越高）

  $C(X)$=从根到结点X那条部分路径的耗费+往前走的最低耗费

- 目标值剪枝:

  子树X不可能包含有更好解 ⇔$C(X)$≥当前最好解的值

分枝限界算法

- 算法终止条件：E-结点是叶结点(可行解)即可终止
- 举例说明

对成本矩阵进行归约的过程如下：

∞	20	30	10	11
15	∞	16	4	2
3	5	∞	2	4
19	6	18	∞	3
16	4	7	16	∞

	V1	V2	V3	V4	V5	
V1	∞	20	30	<u>10</u>	11	10
V2	15	∞	16	4	<u>2</u>	2
V3	3	5	∞	<u>2</u>	4	2
V4	19	6	18	∞	<u>3</u>	3
V5	16	<u>4</u>	7	16	∞	4

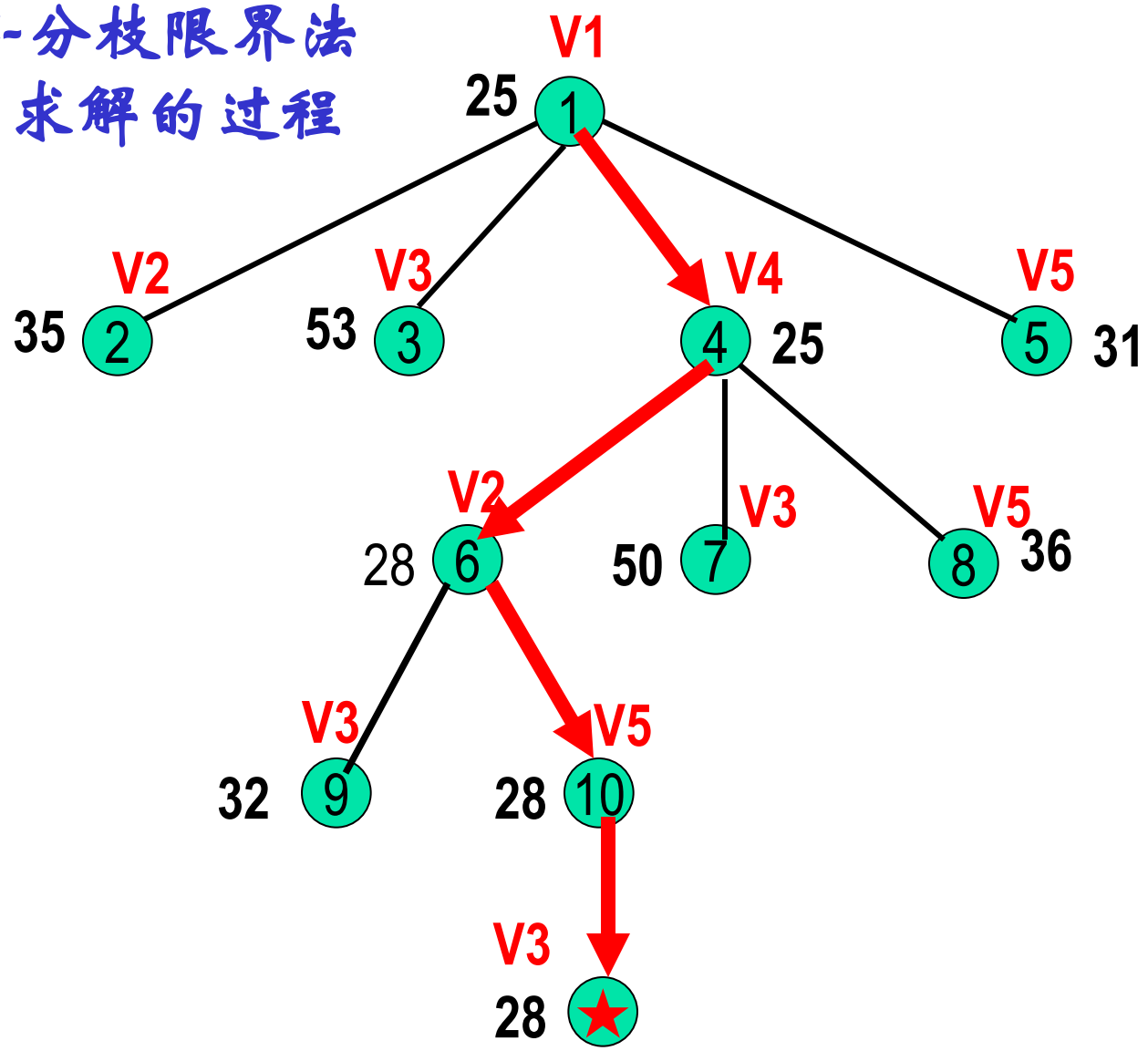
	1		3	
∞	10	20	0	1
13	∞	14	2	0
<u>1</u>	3	∞	0	2
16	3	15	∞	0
12	0	<u>3</u>	12	∞

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	0	12	∞

矩阵归约

归约成本= 10+2+2+3+4+1+3 =25

LC-分枝限界法的求解的过程



4 小结

- 回溯法与分枝限界法

  - **回溯法**

    基本思想：深度优先搜索 + 剪枝操作

    性能特点：空间开销小；搜索缺乏灵活性、跳跃性小

  - **分枝限界法**

    基本思想：E-结点保持到变为死结点的搜索法+剪枝操作

    性能特点：空间开销大；搜索更灵活、跳跃性大

习题

【子集和问题】给定n个正整数的集合$\boldsymbol{A}=\{a_1,a_2,\ldots,a_n\}$和正整数$y$，是否存在一个子集$\boldsymbol{Y}\subseteq\boldsymbol{A}$使得$\boldsymbol{Y}$的元素之和恰为$y$?

考虑用回溯算法求解**子集和问题**实例：给定5个正整数的集合$\boldsymbol{A}=\{3, 5, 8, 12, 15\}$，求元素和恰为20的$\boldsymbol{A}$的所有子集。

（1）给出解向量的形式，指出搜索树的类型，描述剪枝操作。

（2）画出找到所有解所访问到的搜索树，并指出所求的解。

考虑用回溯算法求解**子集和问题**实例：给定5个正整数的集合A={3, 5, 8, 12, 15}，求元素和恰为20的A的所有子集。

（1）给出解向量的形式，指出搜索树的类型，描述剪枝操作。

（2）画出找到所有解所访问到的搜索树，并指出所求的解。

解：（1）假设A中元素已经由小到大有序了。

- 解向量形式为$(x_1,x_2,\ldots,x_n)$，其中x_i=0,1表示集合中第i个元素是否选中为子集中元素$(1\le i\le n)$。
- 搜索树为高度为n的满2叉树。
- 剪枝操作如下： 当搜索到结点$X=(x_1,x_2,\ldots,x_i)$时，如果1,2,…,i元素中选中的元素之和$s\ge 20$（其中s=20说明找到一个解答，输出该解答），则剪枝（即搜索跳过对子树X的搜索）；否则不剪枝，继续往下搜索。

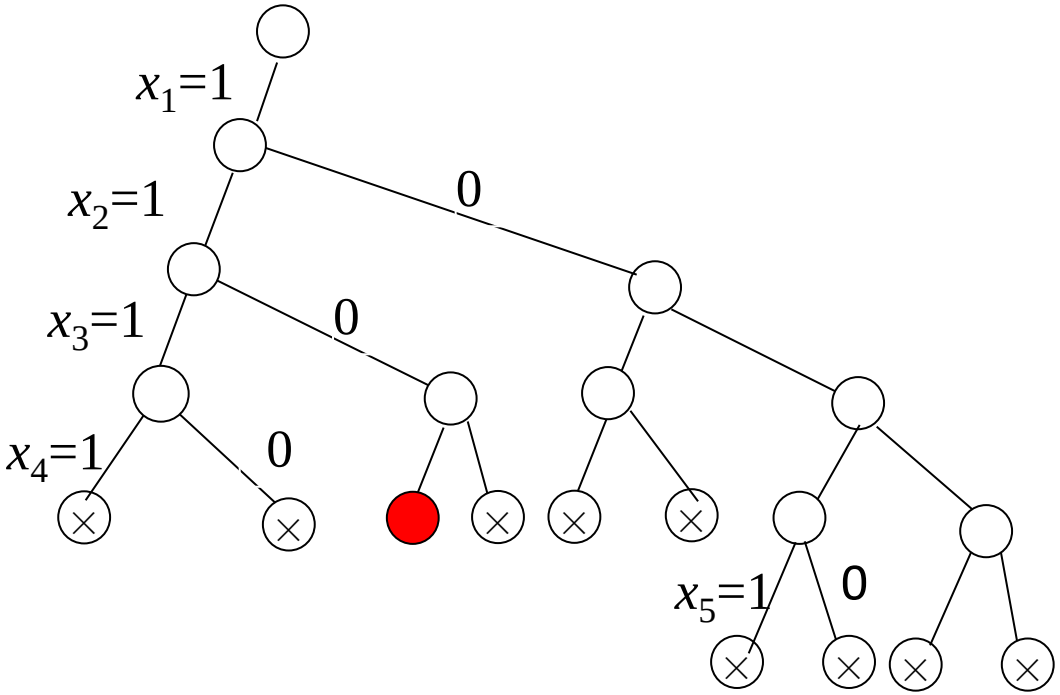
考虑用回溯算法求解**子集和问题**实例：给定5个正整数的集合A={3, 5, 8, 12, 15}，求元素和恰为20的A的所有子集。

（1）给出解向量的形式，指出搜索树的类型，描述剪枝操作。

（2）画出找到所有解所访问到的搜索树，并指出所求的解。

解： （2）找到所有解所生成的（部分）搜索树如下图。

所得解为 3+5+12，5+15， 8+12。



习题

【钱币兑换问题】 设某货币系统有n种硬币，面值由小到大分别为$1=v_1, v_2,\ldots,v_n$。我们需要用最少个数的硬币兑换价值为y的钱。更形式地，我们要让下面的量

$$\sum_{1\le i\le n} x_i$$

在约束条件

$$\sum_{1\le i\le n} (x_i \times v_i) = y$$

下达到最小，其中 $x_1, x_2, \ldots, x_n$ 是非负整数。

考虑用回溯算法求解**钱币兑换问题**实例：v_1=1，v_2=5，v_3=7，v_4=11和 y=20。

（1）给出解向量的形式，指出搜索树的类型，描述剪枝操作。

（2）画出找到所有最优解所访问的搜索树，给出所求最优解。

解：（1）

- 解向量形式为$(x_n, \ldots, x_2, x_1)$，其中$x_i=\lfloor j/v_i \rfloor$, ...,1,0表示第$i$种硬币的个数$(1\le i\le n)$，其中$j=y-(x_nv_n+..+x_iv_i)$。
- 搜索树为高度为n的多叉树。
- 剪枝操作如下：当搜索到结点$X=(x_n, \ldots x_i)$时，如果所用第$n, \ldots, i$种硬币的累计价值$s\ge 20$（其中s=20说明找到一个解答，输出该解答），则剪枝（**约束剪枝**）；如果所用硬币个数超过了当前最好解的硬币个数，则剪枝（**优化剪枝**）；否则不剪枝，继续往下搜索。

考虑用回溯算法求解**钱币兑换问题**实例：v_1=1，v_2=5，v_3=7，v_4=11和 y=20。

（1）给出解向量的形式，指出搜索树的类型，描述剪枝操作。

（2）画出找到所有最优解所访问的搜索树，给出所求最优解。

解： （2）找到所有最优解所生成的搜索树如下图。

所得解为 1×11+1×7+2×1，2×7+1×5+1×1，4×5。