

第八章 回溯法



概述



图问题中的回溯法



组合问题中的回溯法



小结

8.1 概述



8.1.1 问题的解空间

8.1.2 解空间树的动态搜索

8.1.3 回溯法的求解过程

8.1.4 回溯法的时间性能

8.1.1 问题的解空间树

用回溯法求解一个具有 n 个输入的问题，一般情况下，将其可能解表示为满足某个约束条件的等长向量 $X=(x_1, x_2, \dots, x_n)$ ，其中分量 x_i ($1 \leq i \leq n$)的取值范围是某个有限集合 $S_i=\{a_{i1}, a_{i2}, \dots, a_{iri}\}$

所有可能的解向量构成了问题的解空间

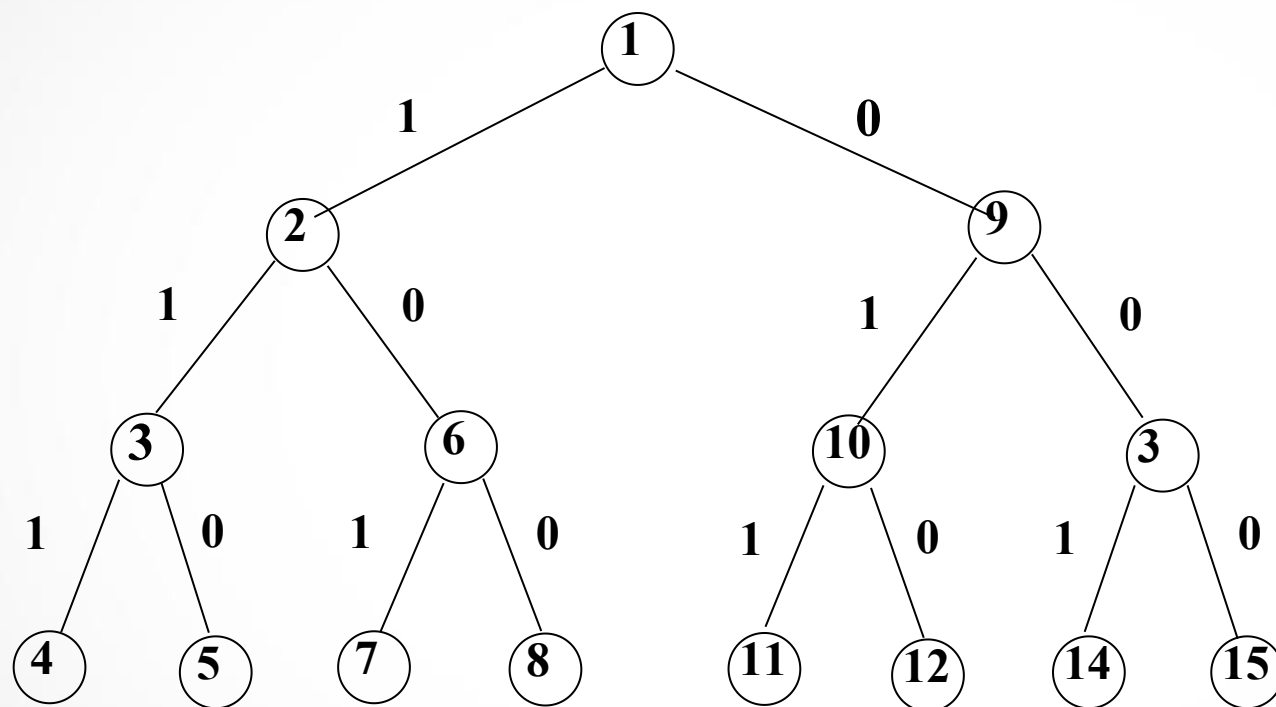
8.1.1 问题的解空间树

问题的解空间一般用解空间树（也称状态空间树）的方式组织：

树的根结点位于第1层，表示搜索的初始状态，第2层的结点表示对解向量的第一个分量做出选择后到达的状态，第1层到第2层的边上标出对第一个分量选择的结果

依此类推，从树的根结点到叶子结点的路径就构成了解空间的一个可能解。

对于 $n=3$ 的0/1背包问题解空间树



对物品1的选择

对物品2的选择

对物品3的选择

图8.2 0/1背包问题的解空间树

对于 $n=4$ 的TSP问题解空间树

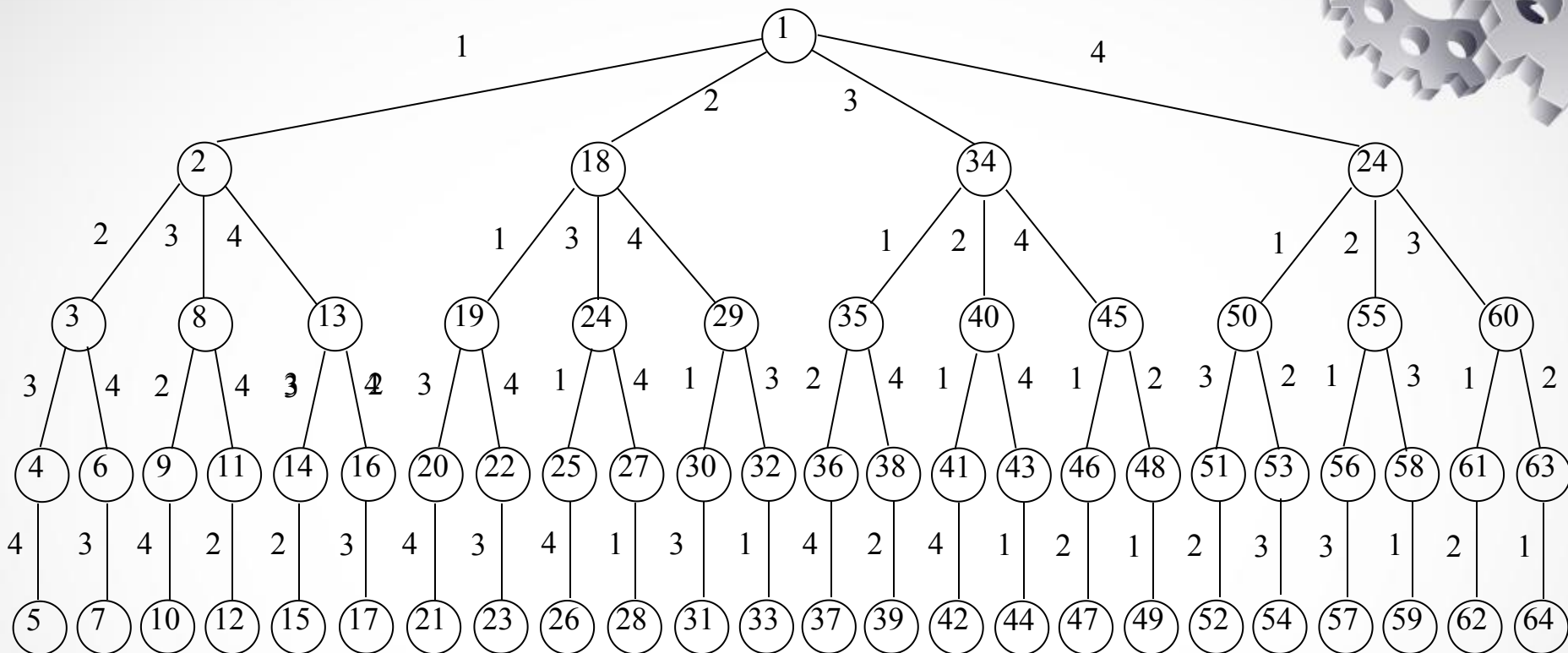


图8.3 $n=4$ 的TSP问题的解空间树


8.1.2 回溯法的设计思想



回溯法从根结点出发，按照**深度优先**策略遍历解空间树，搜索满足约束条件的解。

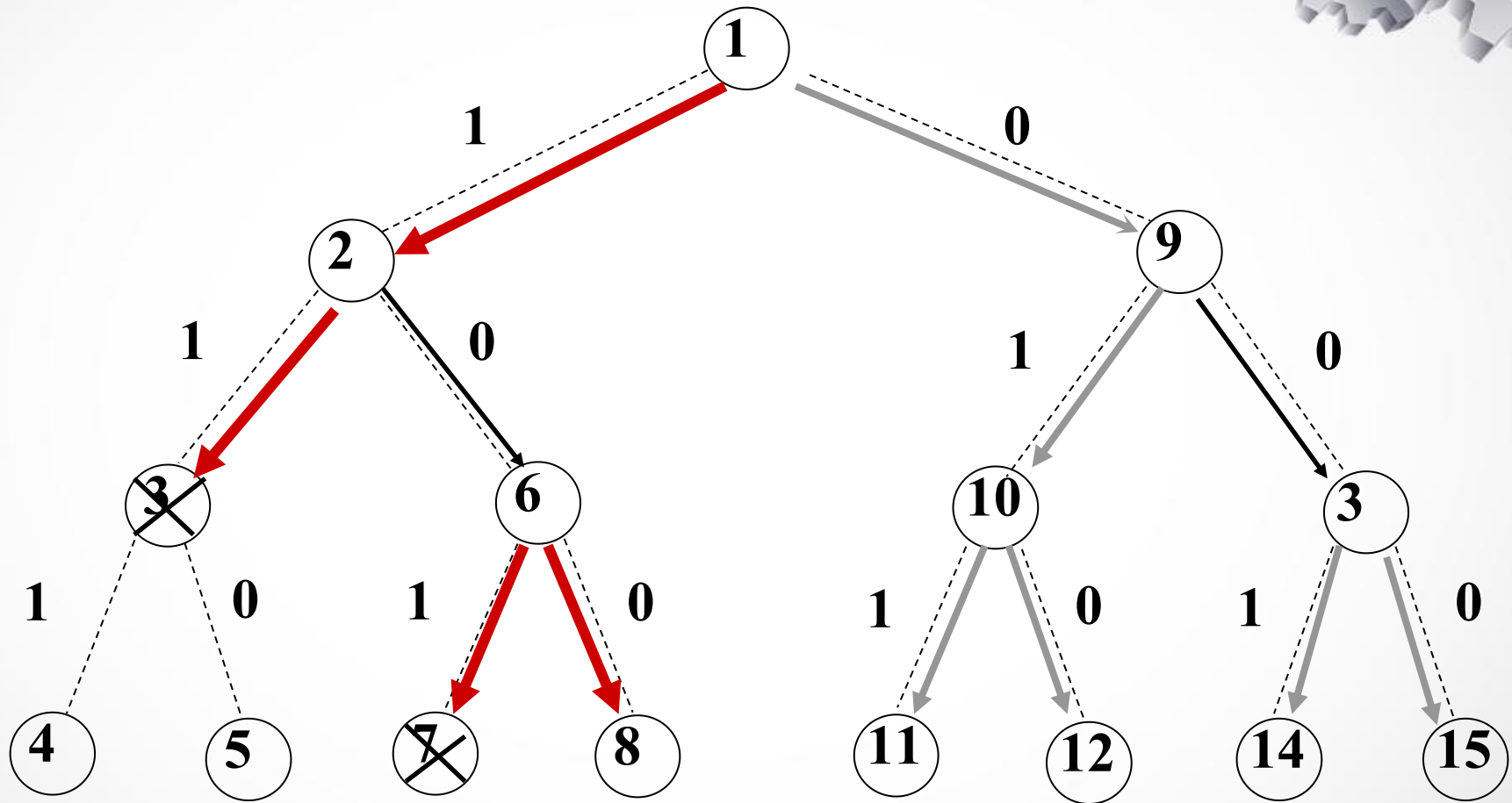
在搜索至树中任一结点时，先判断该结点对应的部分解是否满足约束条件，或者是否超出目标函数的界，也就是判断该结点是否包含问题的（最优）解，如果肯定不包含，则跳过对以该结点为根的子树的搜索，即所谓**剪枝 (Pruning)**；否则，进入以该结点为根的子树，继续按照深度优先策略搜索。

解空间树的动态搜索过程



例如，对于 $n=3$ 的0/1背包问题，三个物品的重量为 $\{20, 15, 10\}$ ，价值为 $\{20, 30, 25\}$ ，背包容量为25，从图8.2所示的解空间树的根结点开始搜索，搜索过程如下：

重量为{20, 15, 10}
价值为{20, 30, 25}
背包容量为25



小结

回溯法的搜索过程涉及的结点（称为搜索空间）只是整个解空间树的一部分，在搜索过程中，通常采用两种策略避免无效搜索：

- (1) 用约束条件剪去得不到可行解的子树；
- (2) 用目标函数剪去得不到最优解的子树。这两类函数统称为剪枝函数。

问题的解空间树是虚拟的，并不需要在算法运行时构造一棵真正的树结构，只需要存储从根结点到当前结点的路径。

8.1.3 回溯法的时间性能

回溯法的一般框架——递归形式

主算法

1. $X = \{ \}$;
2. $flag = false$;
3. $advance(1)$;
4. if ($flag$) 输出解 X ;
else 输出 “无解” ;

$advance(int\ k)$

1. 对每一个 $x \in S_k$ 循环执行下列操作
 - 1.1 $x_k = x$;
 - 1.2 将 x_k 加入 X ;
 - 1.3 if (X 是最终解) $flag = true$;
return;
 - 1.4 else if (X 是部分解)
 $advance(k+1)$;

8.1.3 回溯法的时间性能

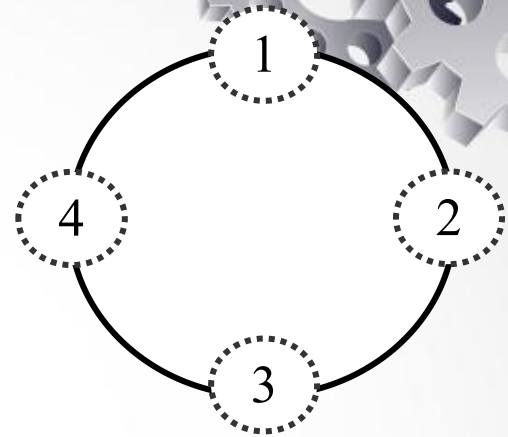
在问题的解向量 $X=(x_1, x_2, \dots, x_n)$ 中，分量 $x_i (1 \leq i \leq n)$ 的取值范围为某个有限集合 $S_i=\{a_{i1}, a_{i2}, \dots, a_{iri}\}$ ，因此，问题的解空间由笛卡儿积 $A=S_1 \times S_2 \times \dots \times S_n$ 构成。

在用回溯法求解问题时，常常遇到两种典型的解空间树：

(1) **子集树**：当所给问题是从 n 个元素的集合中找出满足某种性质的子集时，相应的解空间树称为子集树。在子集树中， $|S_1|=|S_2|=\dots=|S_n|=c$ ，即每个结点有相同数目的子树，通常情况下 $c=2$ ，因此，子集树中共有 2^n 个。

(2) **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。在排列树中，通常情况下， $|S_1|=n$ ， $|S_2|=n-1$ ， \dots ， $|S_n|=1$ ，所以，排列树中共有 $n!$ 个叶子结点，因此，遍历排列树需要 $\Omega(n!)$ 时间。

8.1.4 一个简单的例子—素数环问题



【问题】 把整数 $\{1, 2, \dots, 20\}$ 填写到一个环中，要求每个整数只填写一次，并且相邻的两个整数之和是一个素数。

8.1.4 一个简单的例子—素数环问题

【想法】这个素数环有20个位置，每个位置可以填写的整数有1~20共20种可能，可以对每个位置从1开始进行试探，约束条件是正在试探的数满足如下条件：

- (1) 与已经填写到素数环中的整数不重复；
- (2) 与前面相邻的整数之和是一个素数；
- (3) 最后一个填写到素数环中的整数与第一个填写的整数之和是一个素数。

在填写第 k 个位置时，如果满足上述约束条件，则继续填写第 $k+1$ 个位置；如果1~20个数都无法填写到第 k 个位置，则取消对第 k 个位置的填写，回溯到第 $k-1$ 个位置。

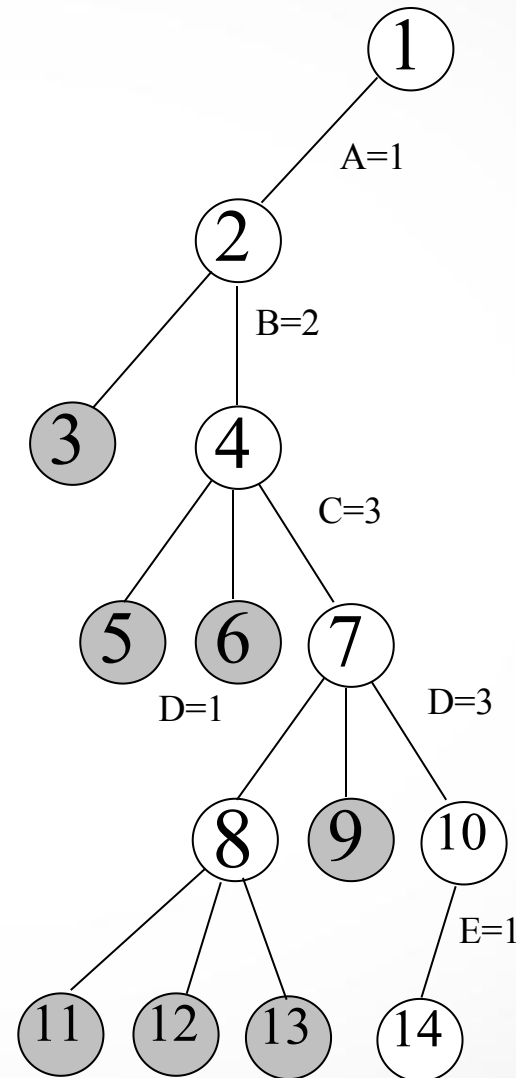
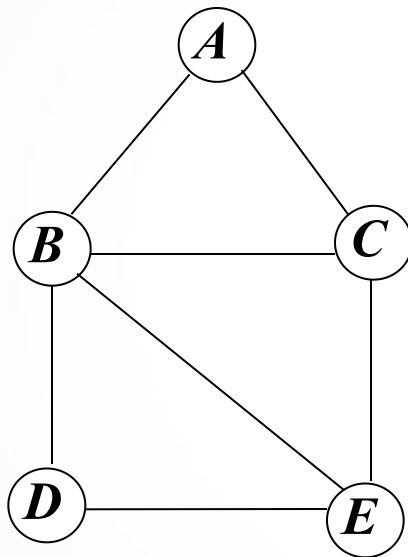
图问题中的回溯法——图着色问题

图着色问题描述为：给定无向连通图 $G=(V, E)$ 和正整数 m ，求最小的整数 m ，使得用 m 种颜色对 G 中的顶点着色，使得任意两个相邻顶点着色不同。

用一个 n 元组 $C=(c_1, c_2, \dots, c_n)$ 来描述图的一种可能着色。其中， $c_i \in \{1, 2, \dots, m\}$ 表示赋予顶点 i 的颜色。

例如，5元组 $(1, 2, 2, 3, 1)$ 表示对于5个顶点的无向图，顶点1着色1，顶点2着色2，顶点3着色2，顶点4着色3，顶点5着色1。

图问题中的回溯法——图着色问题



图问题中的回溯法——图着色问题

算法：回溯法求解图着色问题

1、数组color[n]初始化为0;

2、 $k=0$;

3、while ($k \geq 0$)

3.1、依次考察每一种颜色，若顶点k的着色与其他顶点的着色不冲突，则转3.2；否则，搜索下一个颜色；

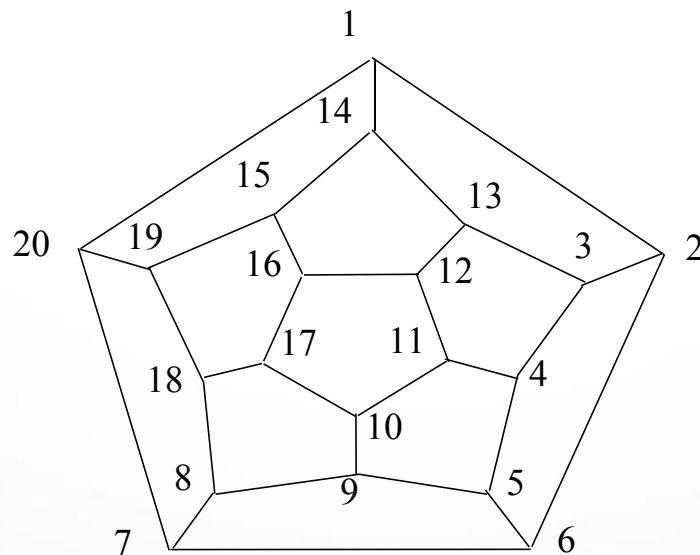
3.2、若顶点已全部着色，则输出答案color[n]，结束；

3.3、若顶点k是一个合法着色，则 $k=k+1$ ，转3，处理下一个顶点；

3.4、否则，重置顶点k的着色情况， $k=k-1$ ，转3，回溯；

图问题中的回溯法——哈密顿回路问题

问题描述：著名的爱尔兰数学家哈密顿提出了著名的周游世界问题。正十二面体的20个顶点代表20个城市，要求从一个城市出发，经过每个城市恰好一次，然后回到出发城市。



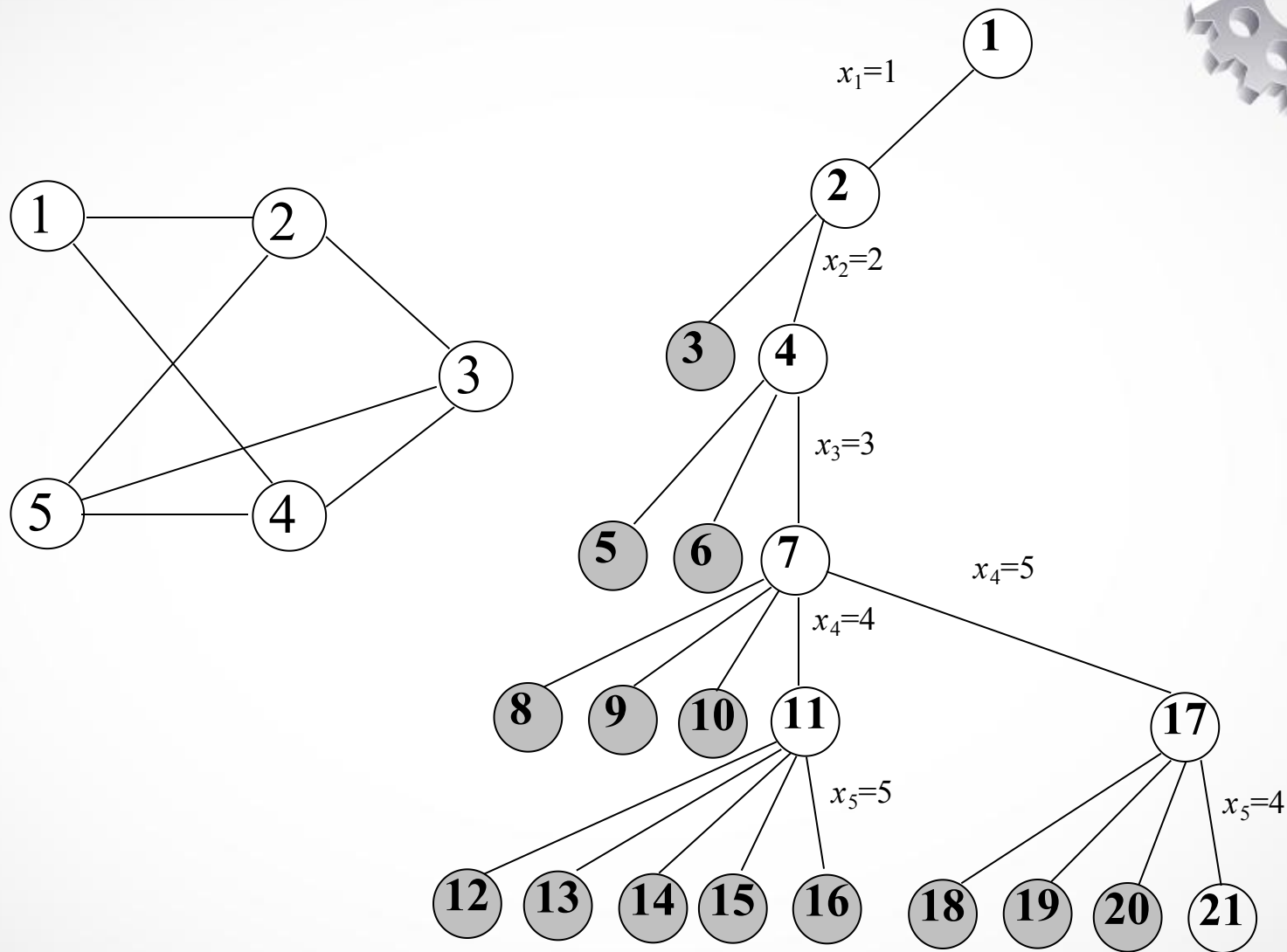
哈密顿回路问题数学模型

假定图 $G=(V, E)$ 的顶点集为 $V=\{1, 2, \dots, n\}$, 则哈密顿回路的可能解表示为 n 元组 $X=(x_1, x_2, \dots, x_n)$, 其中, $x_i \in \{1, 2, \dots, n\}$ 。根据题意, 有如下约束条件:

$$\left\{ \begin{array}{l} (x_i, x_{i+1}) \in E \quad (1 \leq i \leq n-1) \\ (x_n, x_1) \in E \\ x_i \neq x_j \quad (1 \leq i, j \leq n, \quad i \neq j) \end{array} \right.$$

在哈密顿回路的可能解中, 考虑到约束条件 $x_i \neq x_j$ ($1 \leq i, j \leq n, \quad i \neq j$), 则可能解应该是 $(1, 2, \dots, n)$ 的一个排列, 对应的解空间树中有 $n!$ 个叶子结点。

图问题中的回溯法——哈密顿回路问题



图问题中的回溯法——哈密顿回路问题

设数组 $x[n]$ 存储哈密顿回路上的顶点，数组 $visited[n]$ 存储顶点的访问标志， $visited[i]=1$ 表示哈密顿回路经过顶点 i ，算法如下：

1. 将顶点数组 $x[n]$ 初始化为0，标志数组 $visited[n]$ 初始化为0;
2. $visited[1]=1$; $x[1]=1$; $k=2$ 从顶点1出发构造哈密顿回路;
3. while ($k \geq 1$)
 - 3.1 $x[k]=x[k]+1$, 搜索下一个顶点;
 - 3.2 若(n 个顶点没有被穷举) 执行下列操作
 - 3.2.1 若(顶点 $x[k]$ 不在哈密顿回路上 $\&\&(x[k-1], x[k]) \in E$), 转步骤3.3;
 - 3.2.2 否则, $x[k]=x[k]+1$, 搜索下一个顶点;
 - 3.3 若数组 $x[n]$ 已形成哈密顿路径, 则输出 $x[n]$, 算法结束;
 - 3.4 否则,
 - 3.4.1 若 $x[n]$ 构成部分解, 则 $k=k+1$, 转步骤3;
 - 3.4.2 否则, 重置 $x[k]$, $k=k-1$, 取消顶点 $x[k]$ 的访问标志, 转步骤3;

8.3 组合问题中的回溯法

8.3.1 八皇后问题

8.3.2 批处理作业调度问题



组合问题中的回溯法——八皇后问题

问题描述：八皇后问题是十九世纪著名的数学家高斯于1850年提出的。问题是：在 8×8 的棋盘上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上。可以把八皇后问题扩展到 n 皇后问题，即在 $n \times n$ 的棋盘上摆放 n 个皇后，使任意两个皇后都不能处于同一行、同一列或同一斜线上。

显然，棋盘的每一行上可以而且必须摆放一个皇后，所以， n 皇后问题的可能解用一个 n 元向量 $X=(x_1, x_2, \dots, x_n)$ 表示，其中， $1 \leq i \leq n$ 并且 $1 \leq x_i \leq n$ ，即第 i 个皇后放在第 i 行第 x_i 列上。

组合问题中的回溯法——八皇后问题

由于两个皇后不能位于同一列上，所以，解向量 X 必须满足约束条件：

$$x_i \neq x_j \quad (\text{式8.1})$$

若两个皇后摆放的位置分别是 (i, x_i) 和 (j, x_j) ，在棋盘上斜率为-1的斜线上，满足条件 $i - j = x_i - x_j$ ，在棋盘上斜率为1的斜线上，满足条件 $i + j = x_i + x_j$ ，综合两种情况，由于两个皇后不能位于同一斜线上，所以，解向量 X 必须满足约束条件：

$$|i - x_i| \neq |j - x_j| \quad (\text{式8.2})$$

为了简化问题，下面讨论四皇后问题。四皇后问题的解空间树是一个完全4叉树，树的根结点表示搜索的初始状态，从根结点到第2层结点对应皇后1在棋盘中第1行的可能摆放位置，从第2层结点到第3层结点对应皇后2在棋盘中第2行的可能摆放位置，依此类推。

组合问题中的回溯法——八皇后问题

Q			

(a)

Q			
×	×	Q	

(b)

Q			
×	×	Q	
×	×	×	×

(c)

Q			
			Q

(d)

Q			
			Q
×	Q		

(e)

Q			
			Q
×	Q		
×	×	×	×

(f)

	Q		

(g)

	Q		
×	×	×	Q

(h)

	Q		
			Q
Q			

(i)

	Q		
			Q
Q			
×	×	Q	

(j)

组合问题中的回溯法——八皇后问题

算法：回溯法求解n皇后问题

1、初始化：解向量 $x[n]=\{-1\}$ ， $k=1$ ；

2、while ($k \geq 1$)

2.1、把皇后k摆放在下一列的位置，即 $x[k]++$ ；

2.2、从 $x[k]$ 开始依次考察每一列。如果皇后k摆放在 $x[k]$ 位置上不冲突，则转2.3；否则 $x[k]++$ ，试探下一列；

2.3、若n个皇后已经全部摆放，则输出一个解，结束；

2.4、若尚有皇后没摆放，则 $k++$ ，转2；摆放下一个皇后

2.5、若 $x[k]$ 出界，则回溯， $x[k]=-1$ ， $k--$ ，转2；重新摆放皇后k

3、退出循环，说明n皇后问题无解；

批处理作业调度问题



问题描述: n 个作业 $\{1, 2, \dots, n\}$ 要在两台机器上处理, 每个作业必须先由机器1处理, 然后再由机器2处理, 机器1处理作业 i 所需时间为 a_i , 机器2处理作业 i 所需时间为 b_i ($1 \leq i \leq n$), 批处理作业调度问题要求确定这 n 个作业的最优处理顺序, 使得从第1个作业在机器1上处理开始, 到最后一个作业在机器2上处理结束所需时间最少。

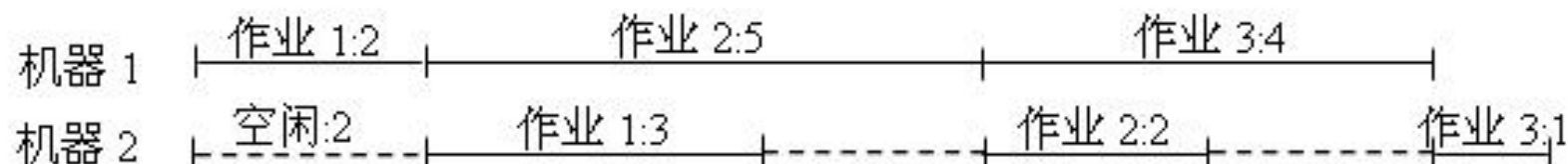
批处理作业调度问题



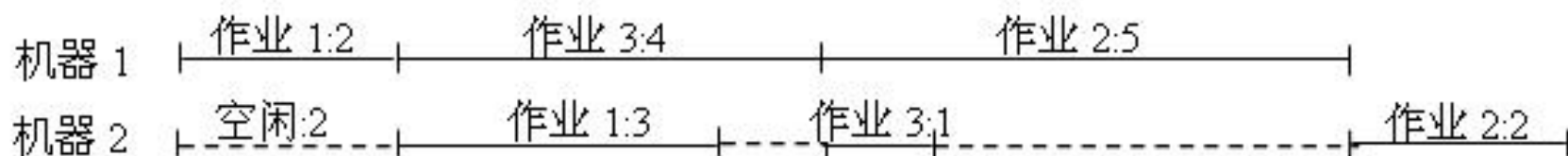
显然，批处理作业的一个最优调度应使机器1没有空闲时间，且机器2的空闲时间最小。可以证明，存在一个最优作业调度使得在机器1和机器2上作业以相同次序完成。

例如，有三个作业 $\{1, 2, 3\}$ ，这三个作业在机器1上所需的处理时间为 $(2, 3, 2)$ ，在机器2上所需的处理时间为 $(1, 1, 3)$ ，则这三个作业存在6种可能的调度方案： $(1, 2, 3)$ 、 $(1, 3, 2)$ 、 $(2, 1, 3)$ 、 $(2, 3, 1)$ 、 $(3, 1, 2)$ 、 $(3, 2, 1)$ ，相应的完成时间为10, 8, 10, 9, 8, 8。

批处理作业调度问题



(a) 调度方案(1, 2, 3), 最后完成时间为 12

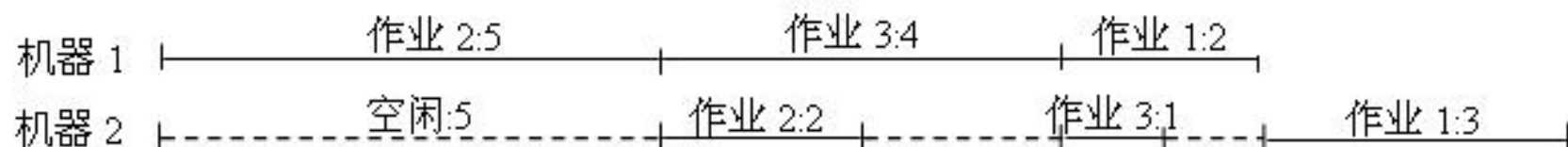


(b) 调度方案(1, 3, 2), 最后完成时间为 13

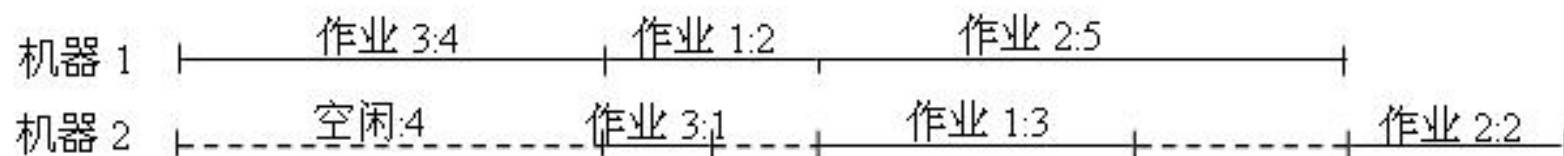


(c) 调度方案(2, 1, 3), 最后完成时间为 12

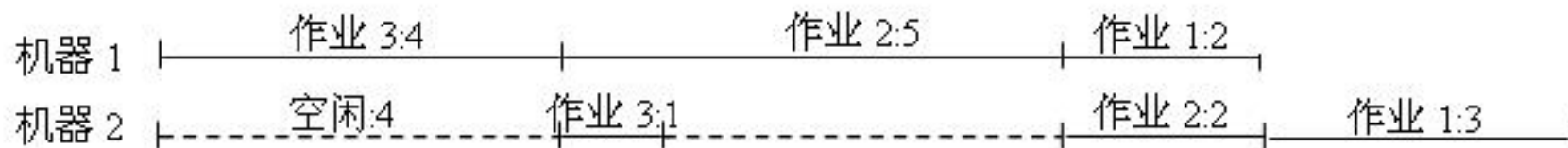
批处理作业调度问题



(d) 调度方案(2, 3, 1), 最后完成时间为 14



(e) 调度方案(3, 1, 2), 最后完成时间为 13



(f) 调度方案(3, 2, 1), 最后完成时间为 14