

计算机算法基础

何琨 brooklet60@gmail.com

华中科技大学计算机学院

第九章 分枝—限界法

第九章 分枝限界法

9.1 一般方法

9.1.1 LC-检索

9.1.2 15谜问题

9.1.3 * - 9.1.6 * LC-检索抽象化控制，
特性，效率分析等

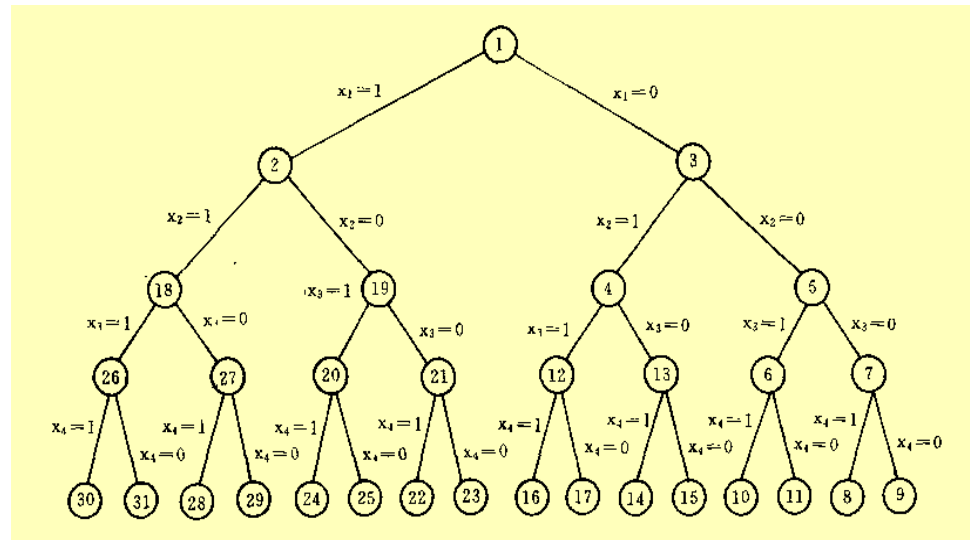
9.2* 0/1背包问题

9.3 * 货郎担问题

回顾：第八章中的基本概念

- 问题解的n元组表示
- 问题状态(树中所有结点)
- 解状态
- 状态空间
- 答案状态(解对应结点)
- 状态空间树
- 活结点
- E-结点
- 死结点

- 回溯法
- 分枝-限界方法
- 限界函数



子集和数问题的状态树

回顾:

- **状态空间树**: 解空间的树结构称为**状态空间树** (state space tree)
- **问题状态**: 树中的每一个结点确定问题的一个状态, 称为**问题状态**(problem state)。
- **状态空间**: 由根结点到其他结点的所有路径则确定了这个问题的**状态空间**(state space)。
- **解状态**: 是这样一些问题状态S, 对于这些问题状态, 由根到S的那条路径确定了这解空间中的一个元组(solution states)。
- **答案状态**: 是这样的一些解状态S, 对于这些解状态而言, 由根到S的这条路径确定了这问题的一个解 (满足隐式约束条件的解) (answer states)。

回顾:

在状态空间树生成的过程中，结点根据被检测情况分为三类：

- **活结点**：自己已经生成,但其儿子结点还没有全部生成并且有待生成的结点。
- **E-结点**（正在扩展的结点）：当前正在生成其儿子结点的活结点。
- **死结点**：不需要再进一步扩展或者其儿子结点已全部生成的结点。

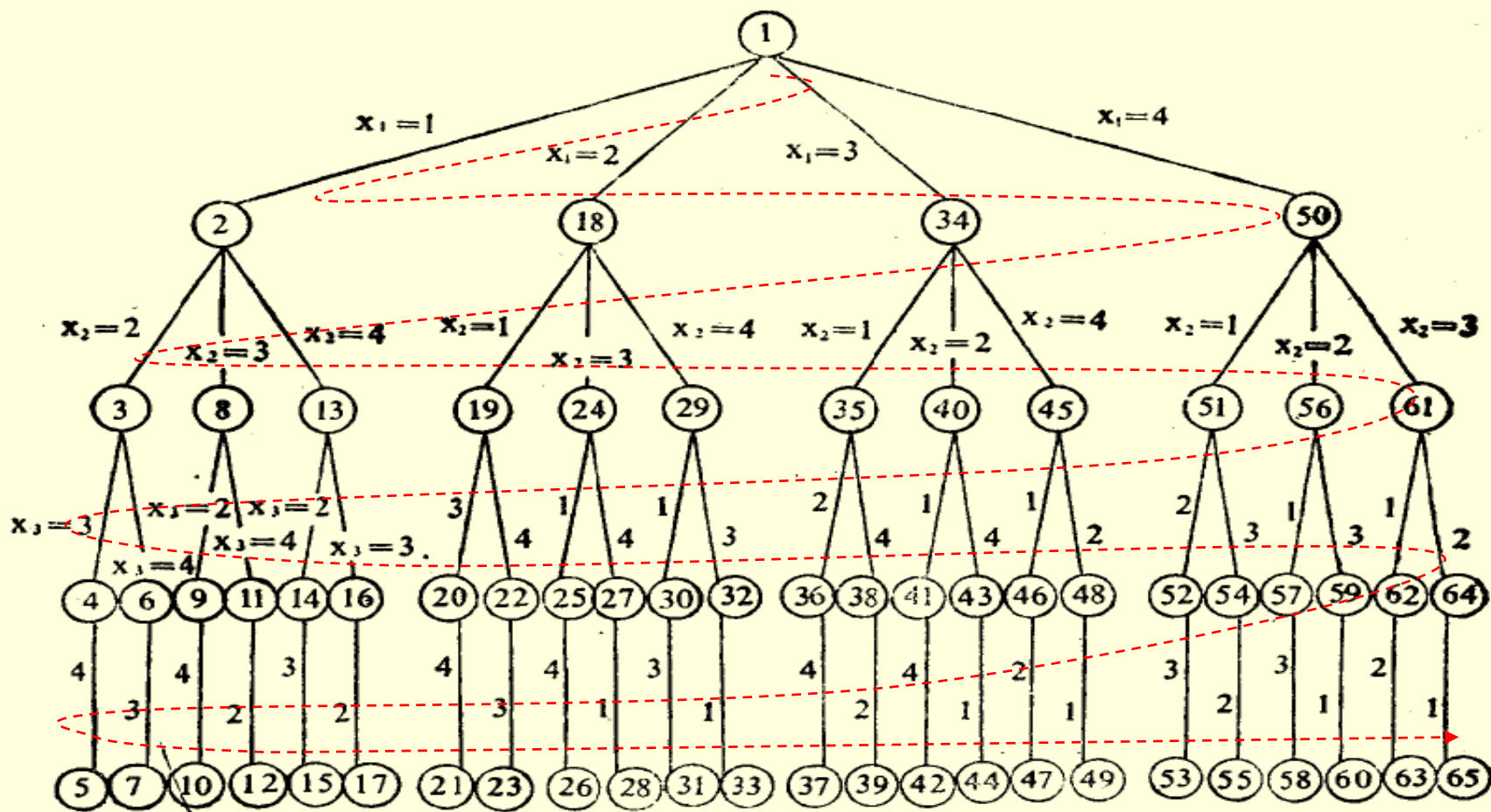
回顾:

- **回溯法**: 使用限界函数的深度优先结点生成方法称为**回溯法** (**backtracking**)
- **分枝-限界方法**: E结点一直保持到死为止的状态生成方法称为**分枝-限界方法** (**branch-and-bound**)
- **限界函数**: 判断当前活结点是否满足**限界函数**, 不满足的提前剪枝, 当前结点变为死结点

分枝—限界法

- 分枝—限界法：在生成当前E-结点全部儿子之后再生成其它活结点的儿子，且用限界函数帮助避免生成不包含答案结点的子树的状态空间的检索方法。(宽度优先)
- 活结点表：
活结点：自己已经被生成，但还没有被检测的结点。
存储结构：队列（First In First Out, BFS）、
栈（Last In First Out, D-Search）
- 分枝—限界法的两种设计策略：
FIFO检索：活结点表采用队列
LIFO检索：活结点表采用栈

例9.1 4-皇后问题的状态空间树。



4-皇后问题完整的状态空间树

限界函数：如果 (x_1, x_2, \dots, x_i) 是到当前E结点的路径，那么具有父—子标记 x_{i+1} 的所有儿子结点是一些这样的结点，它们使得 $(x_1, x_2, \dots, x_i, x_{i+1})$ 表示没有两个皇后正在相互攻击的一种棋盘格局。

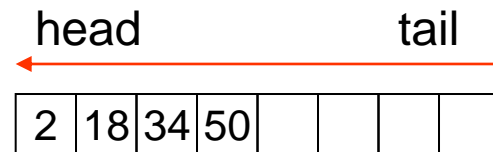
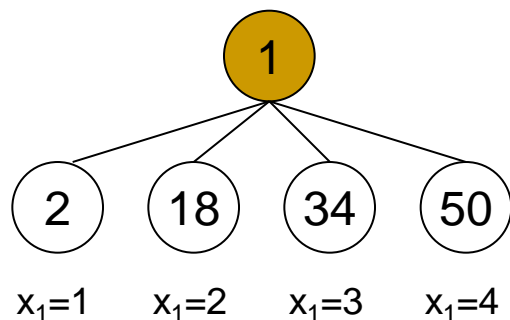
采用FIFO分枝—限界法检索4-皇后问题的状态空间树：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

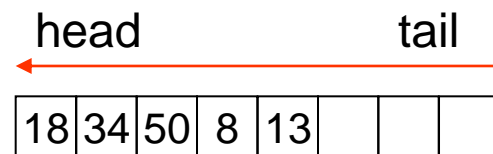
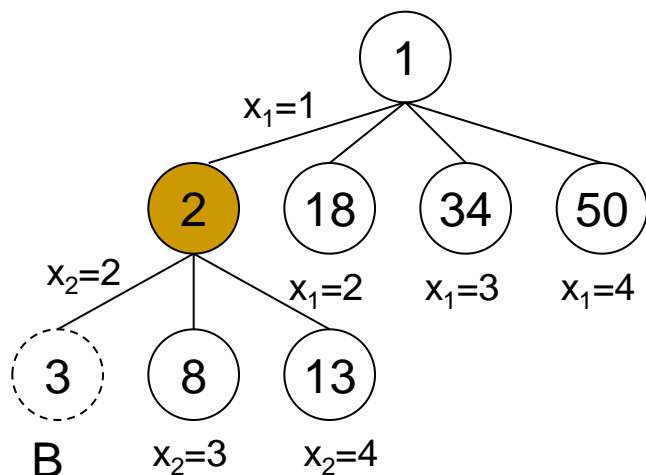
1



扩展结点1，得新结点2，18，34，50

活结点2、18、34、50入队列

2



扩展结点2，得新结点3，8，13

利用限界函数杀死结点3

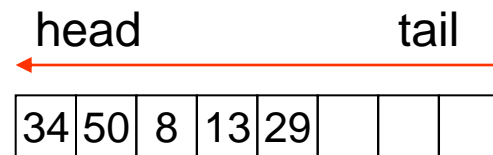
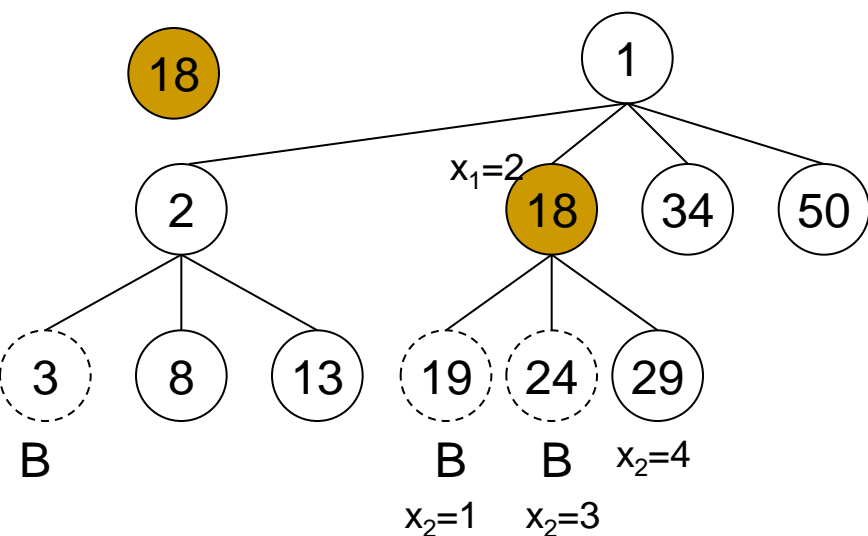
活结点8、13入队列

采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）



扩展结点18，得新结点19，24，29

利用限界函数杀死结点19、24

活结点29入队列

采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

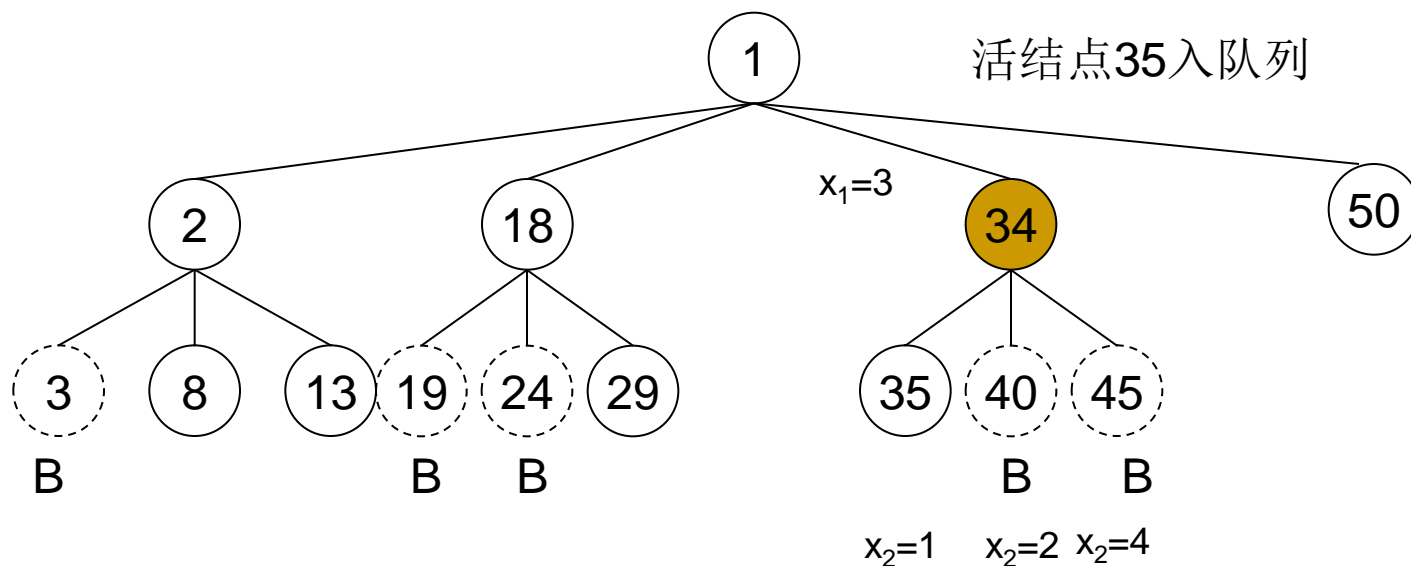
34

head		tail				
←		→				
50	8	13	29	35		

扩展结点34，得新结点35，40，45

利用限界函数杀死结点40、45

活结点35入队列



采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

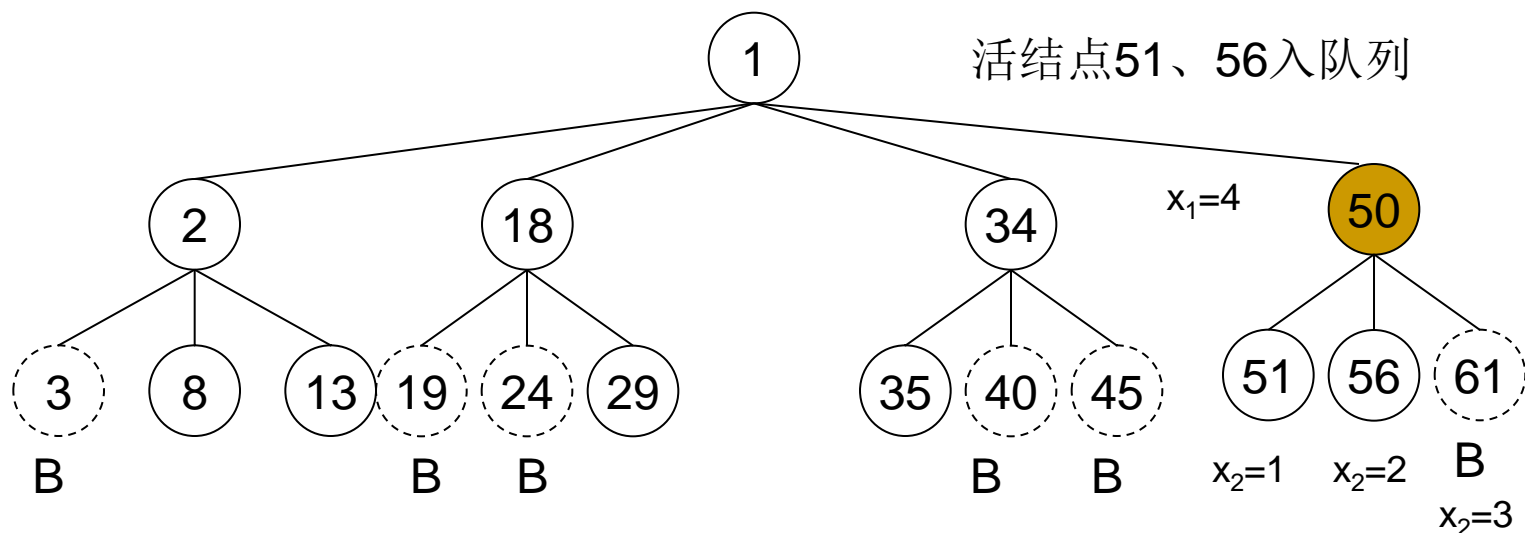
50

head		tail					
←							
8	13	29	35	51	56		

扩展结点50，得新结点51，56，61

利用限界函数杀死结点61

活结点51、56入队列



采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

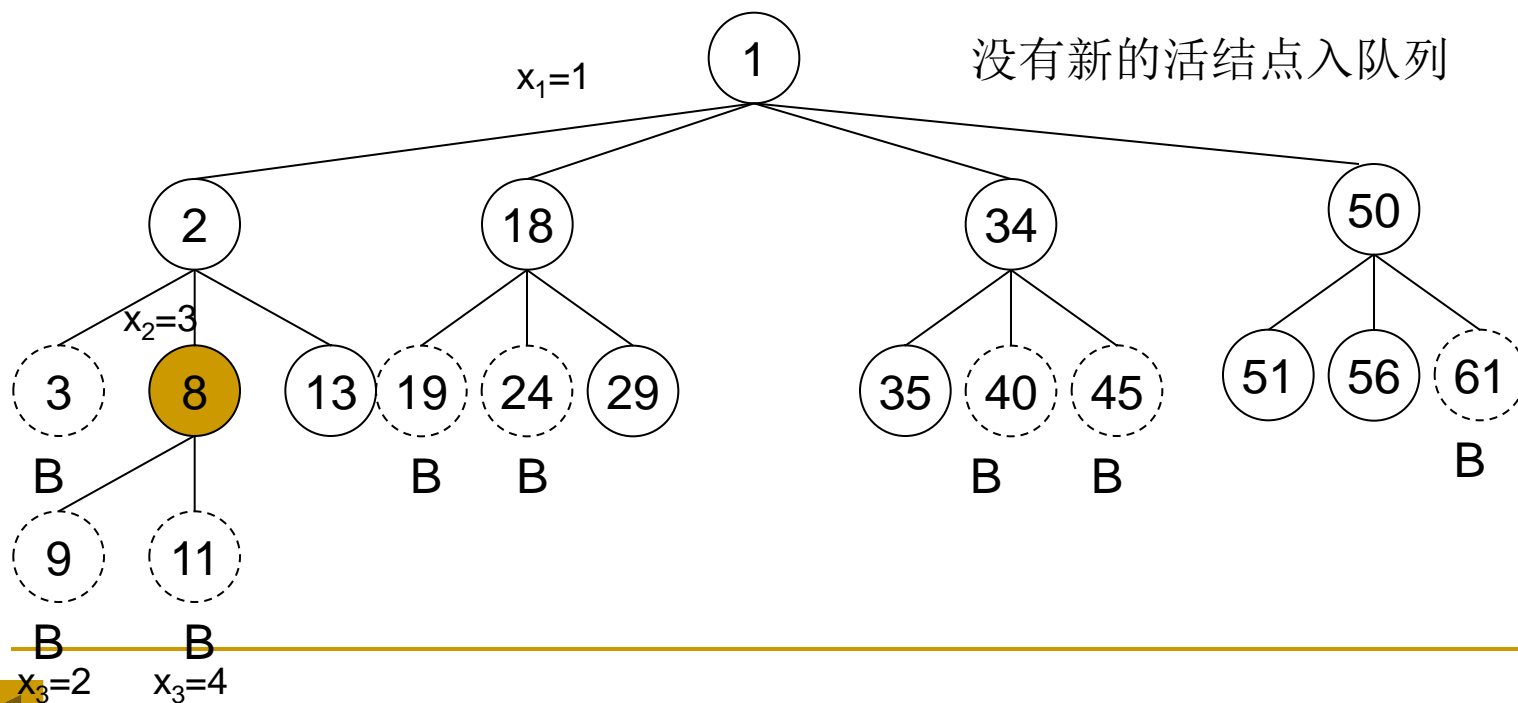
8

head		tail				
←						
13	29	35	51	56		

扩展结点8，得新结点9，11

利用限界函数杀死结点9、11

没有新的活结点入队列



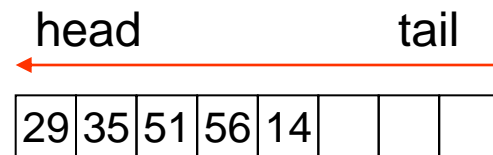
采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

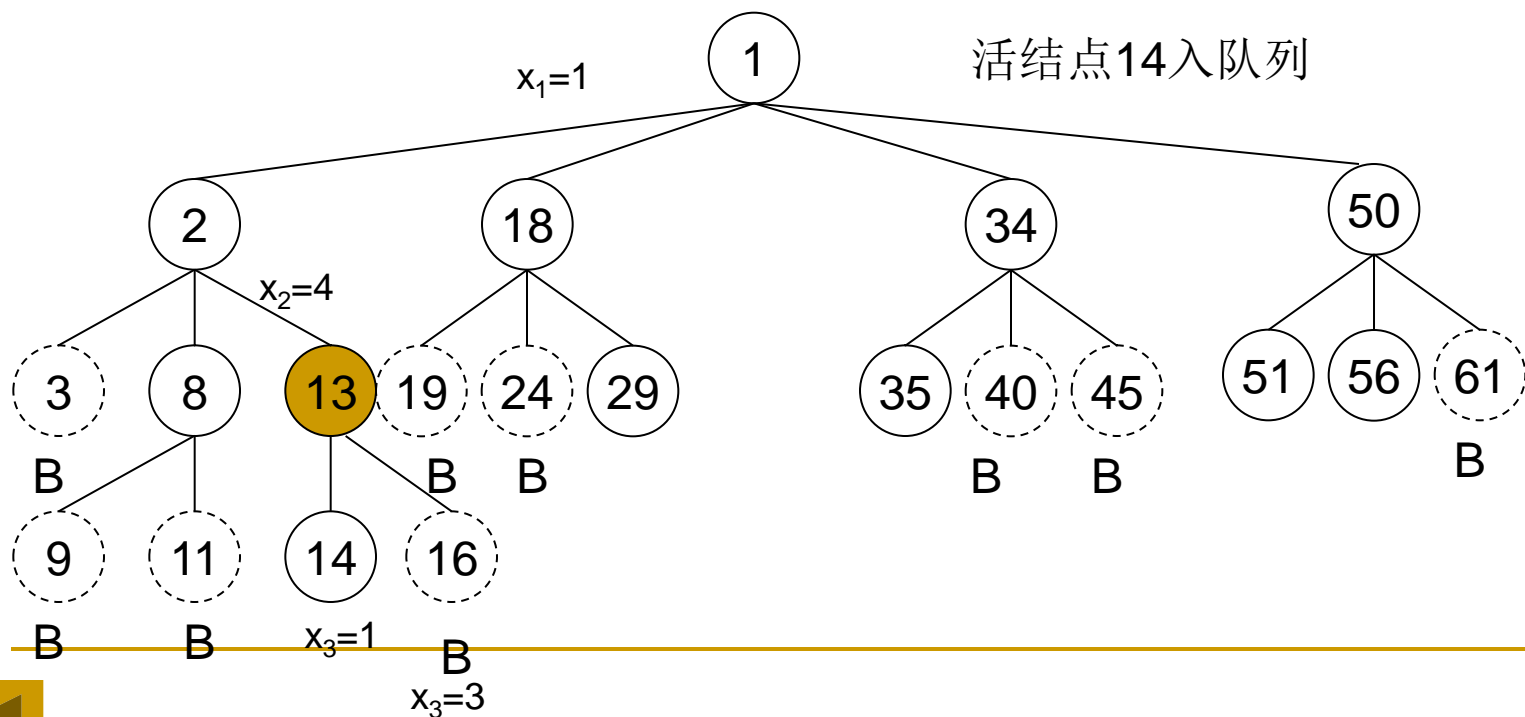
13



扩展结点13，得新结点14，16

利用限界函数杀死结点16

活结点14入队列



采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

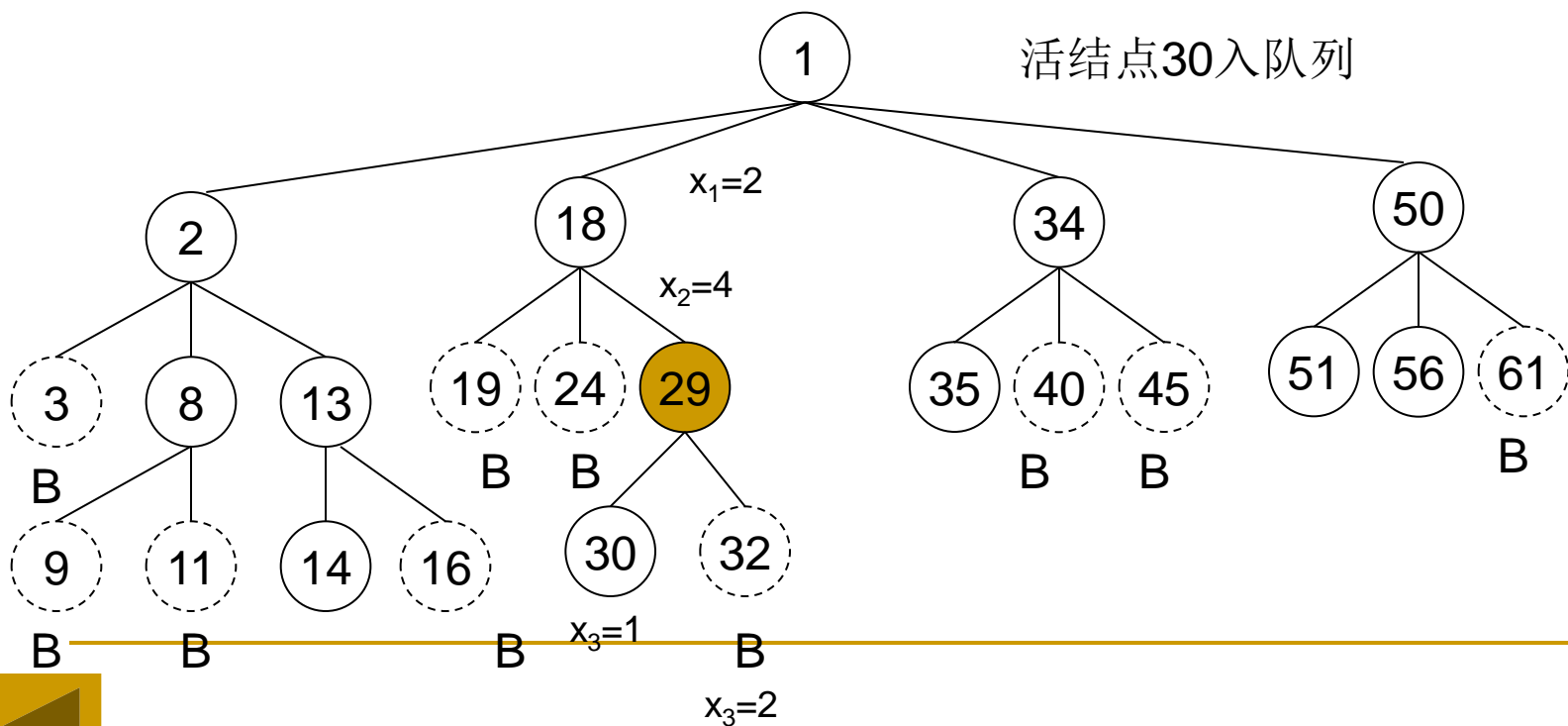
29

head		tail				
←		→				
35	51	56	14	30		

扩展结点29，得新结点30，32

利用限界函数杀死结点32

活结点30入队列

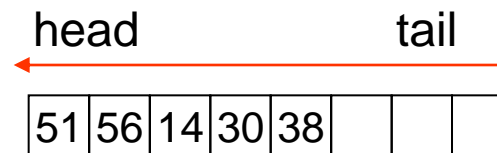


采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

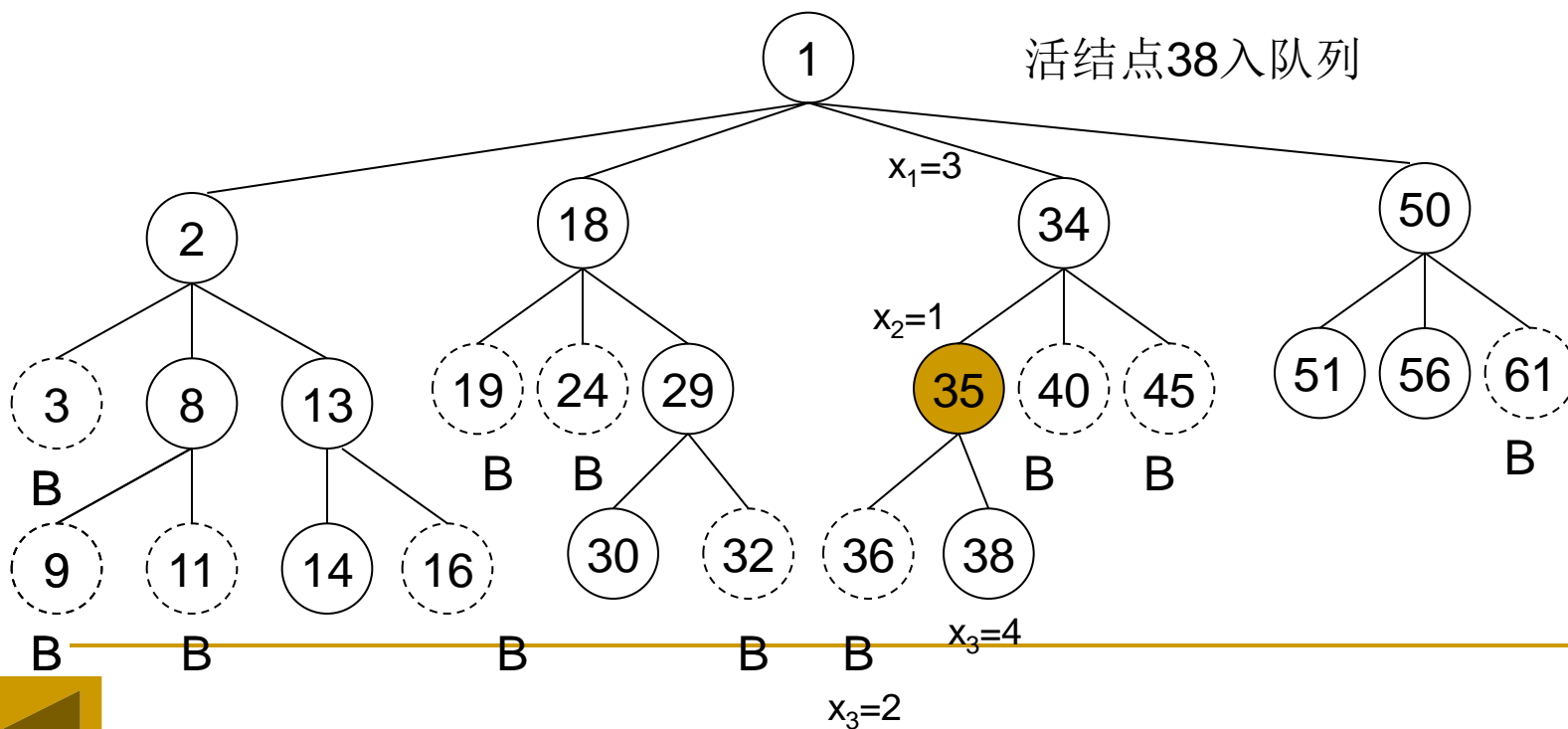


35

扩展结点35，得新结点36，38

利用限界函数杀死结点36

活结点38入队列

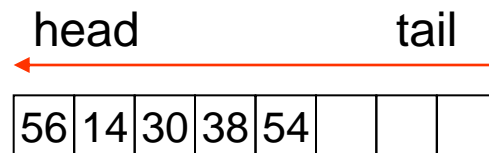


采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

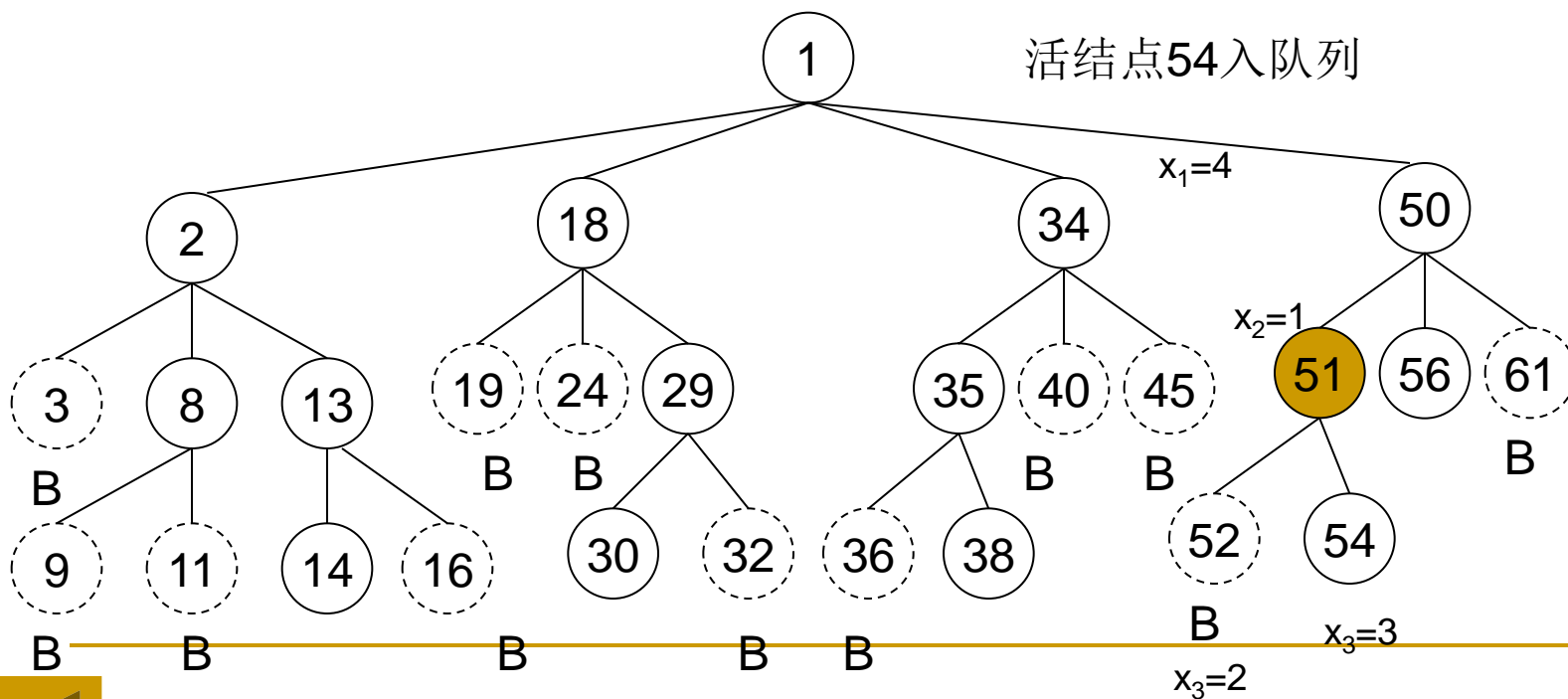


51

扩展结点51，得新结点52，54

利用限界函数杀死结点52

活结点54入队列

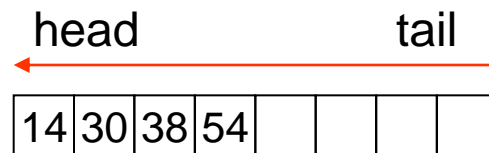


采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

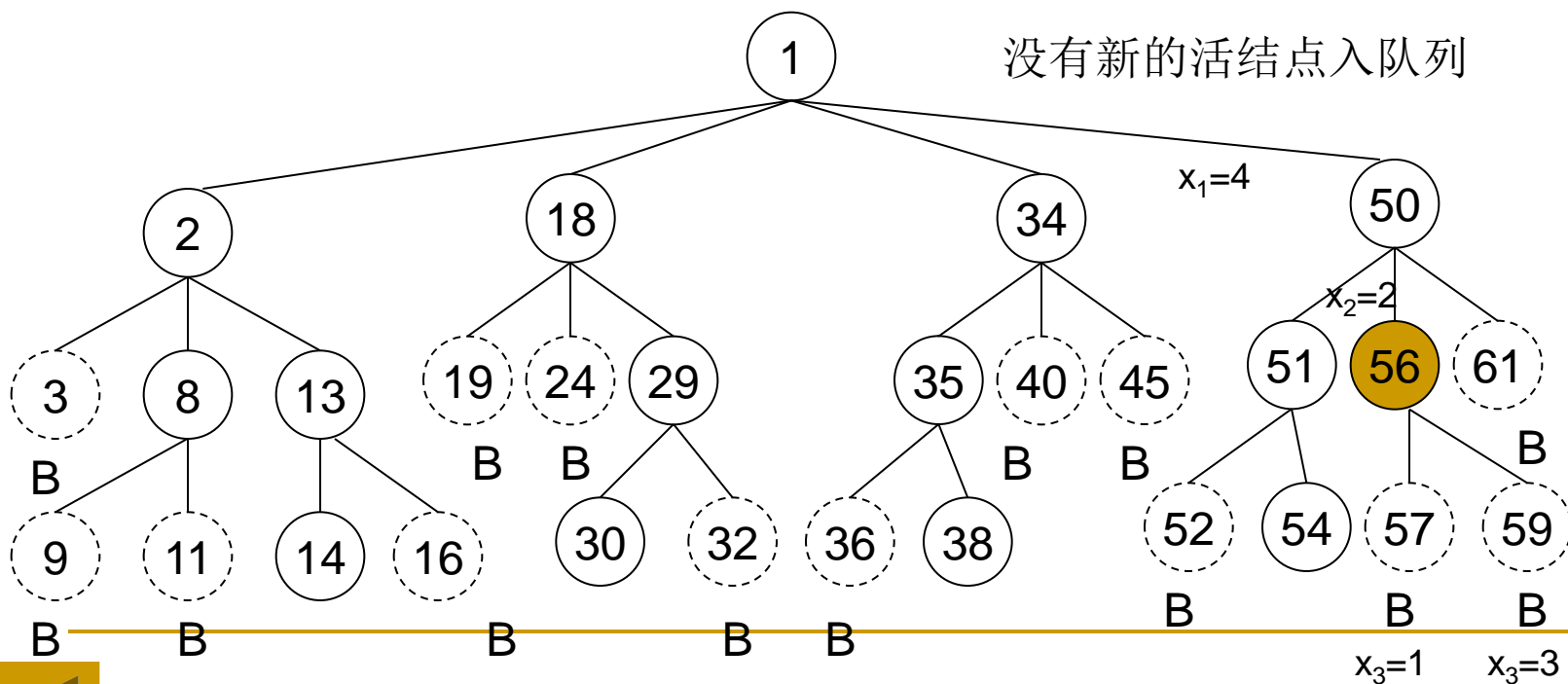


56

扩展结点56，得新结点57，59

利用限界函数杀死结点57、59

没有新的活结点入队列



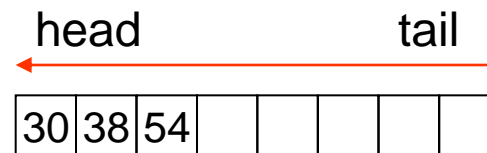
采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

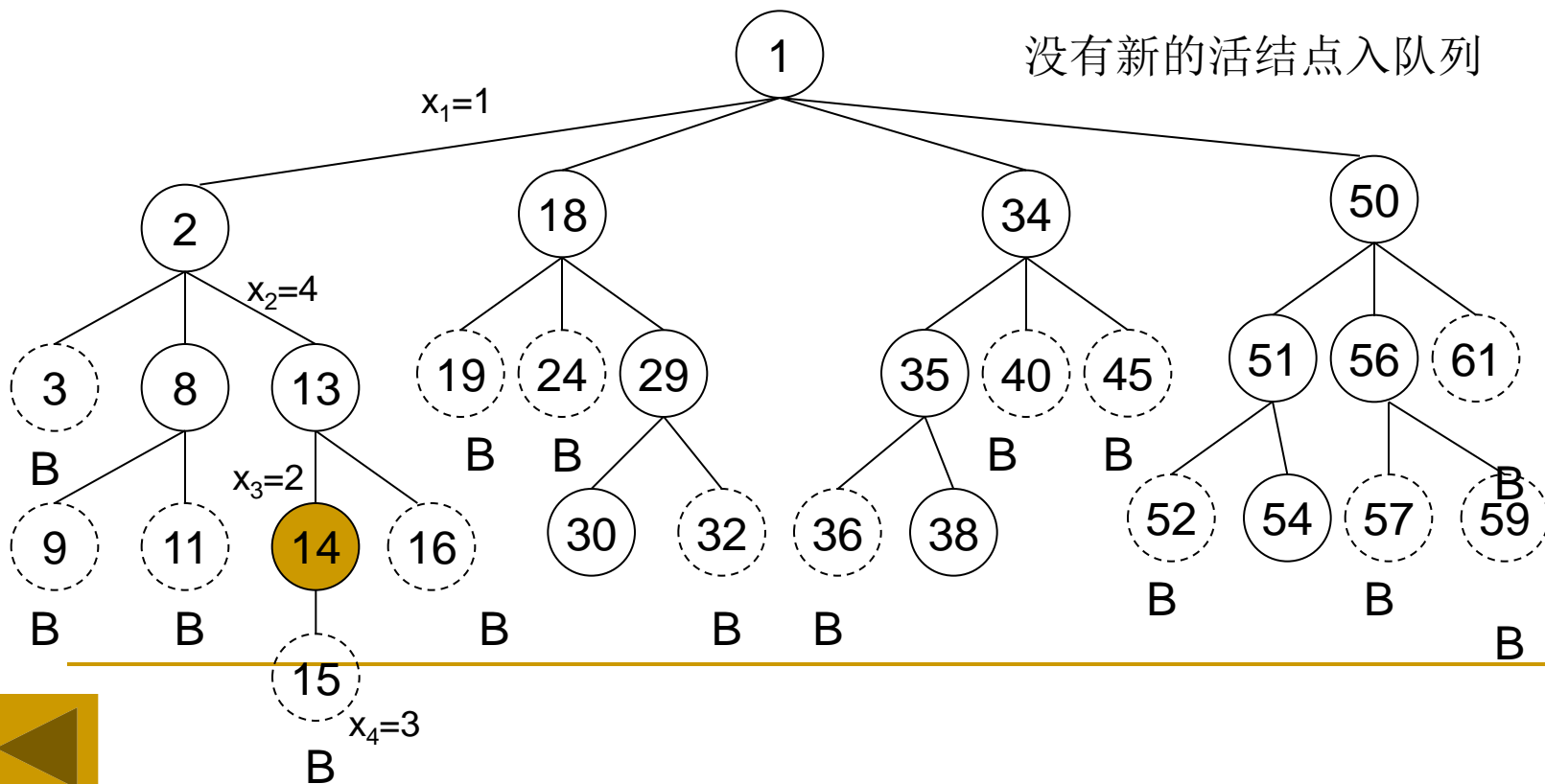
14



扩展结点14，得新结点15

利用限界函数杀死结点15

没有新的活结点入队列



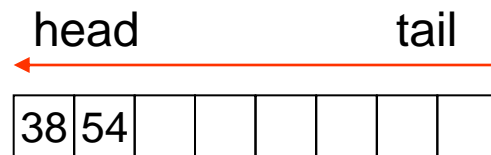
采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

活结点

扩展活结点得到的状态空间树

活结点表（队列）

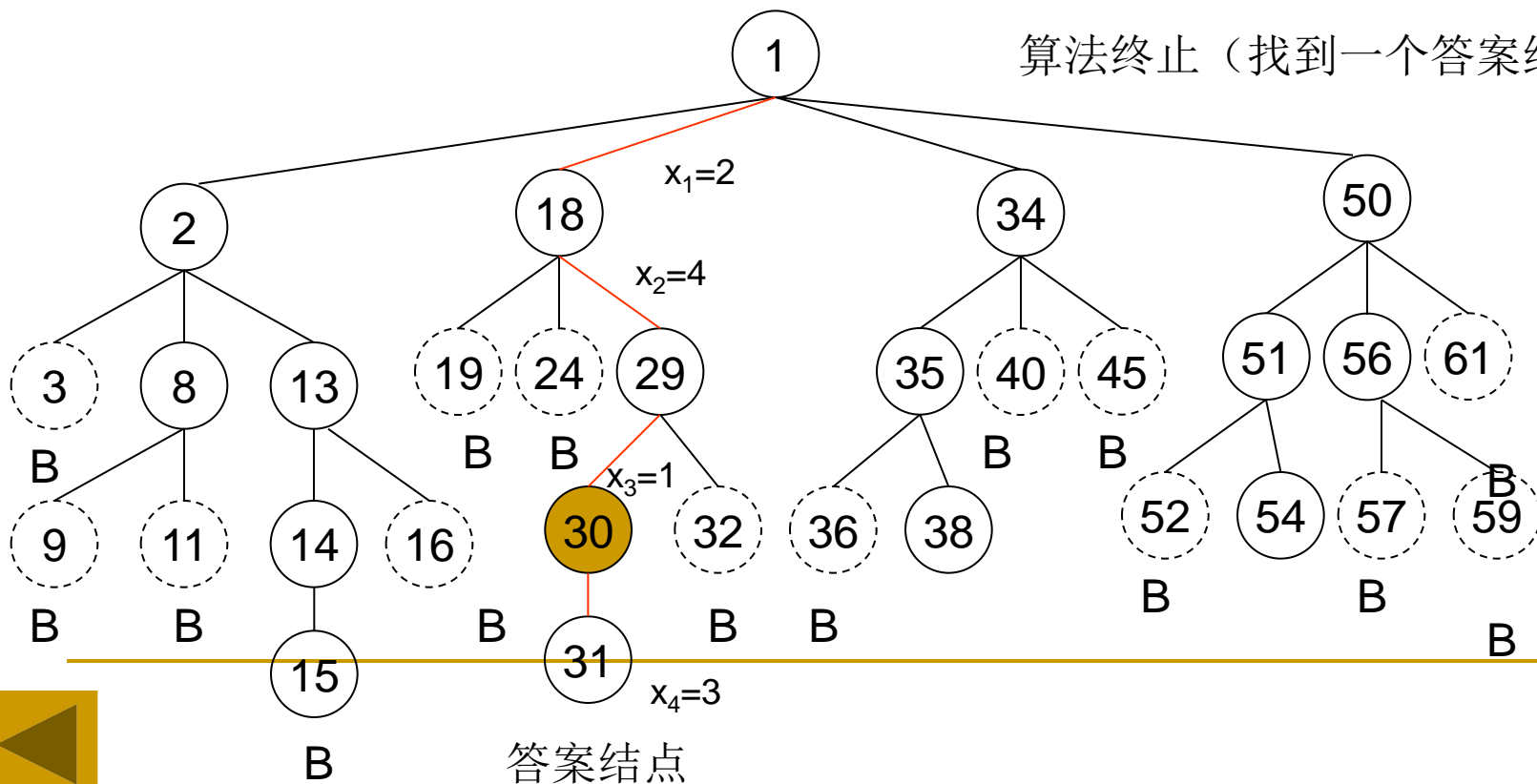
30



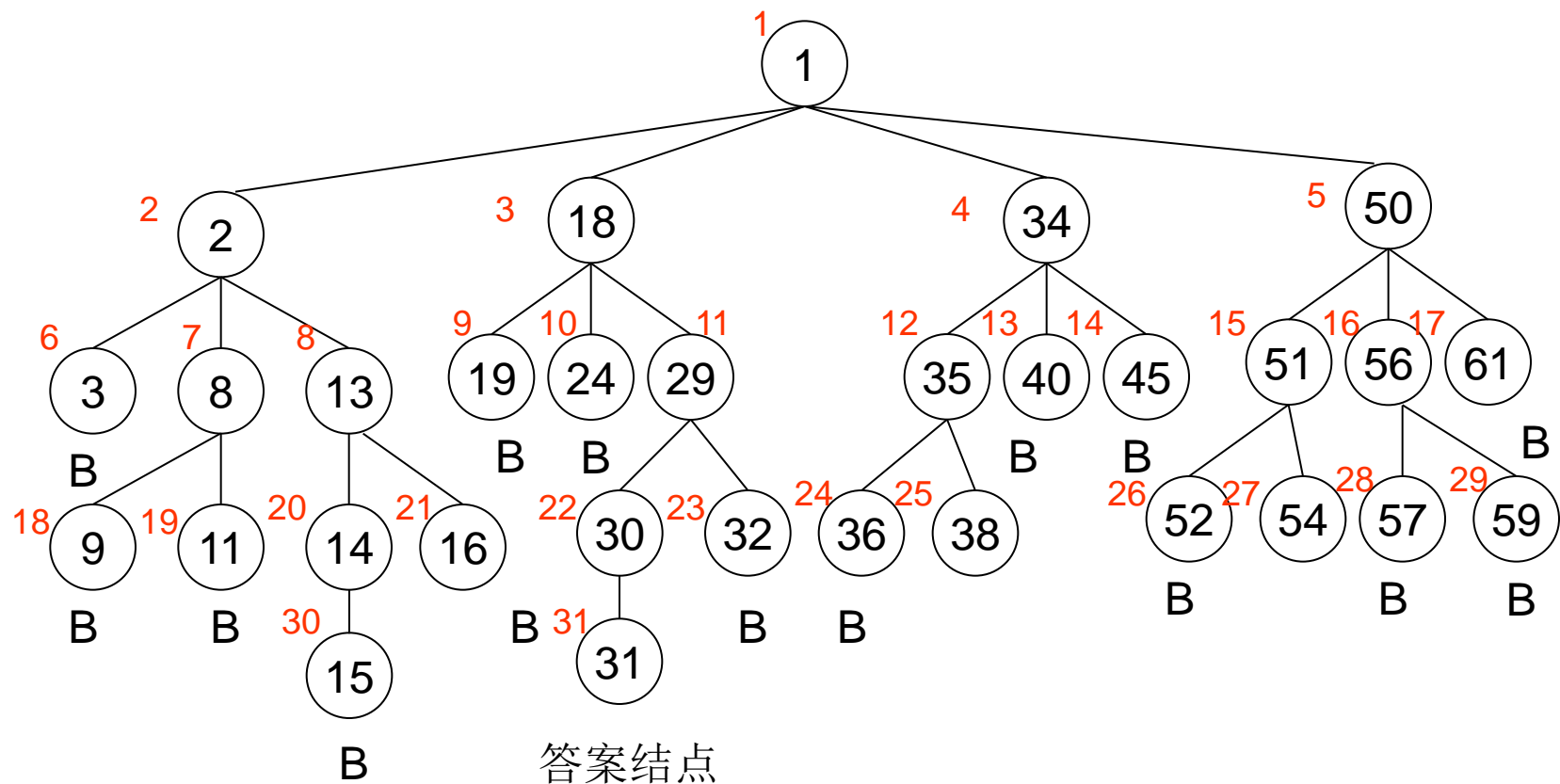
扩展结点30，得新结点31

结点31为答案结点

算法终止（找到一个答案结点）



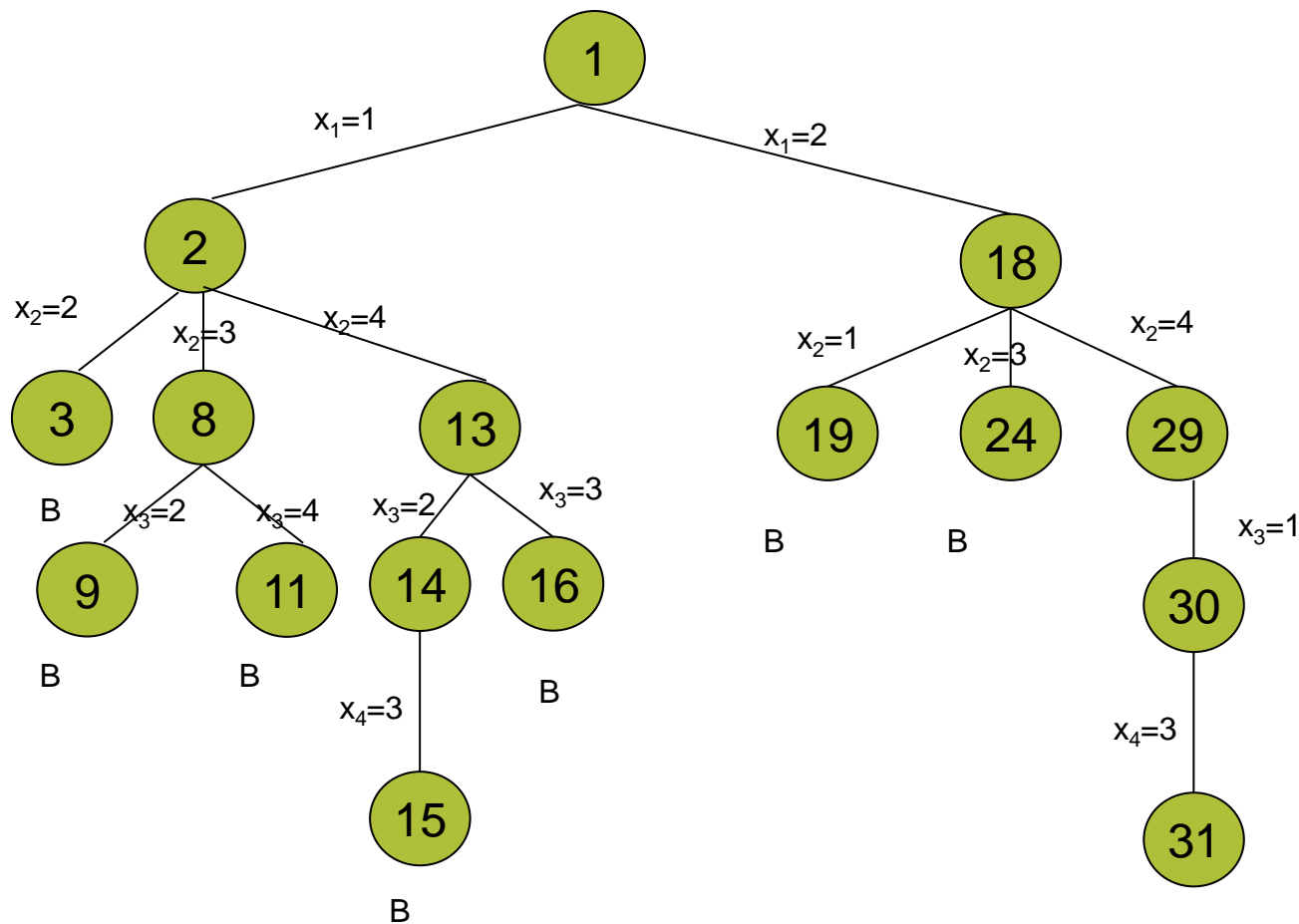
采用FIFO分枝—限界法检索4-皇后问题的状态空间树:



4-皇后问题— 回溯 vs FIFO分枝-限界

回溯

Win!



答案
结点

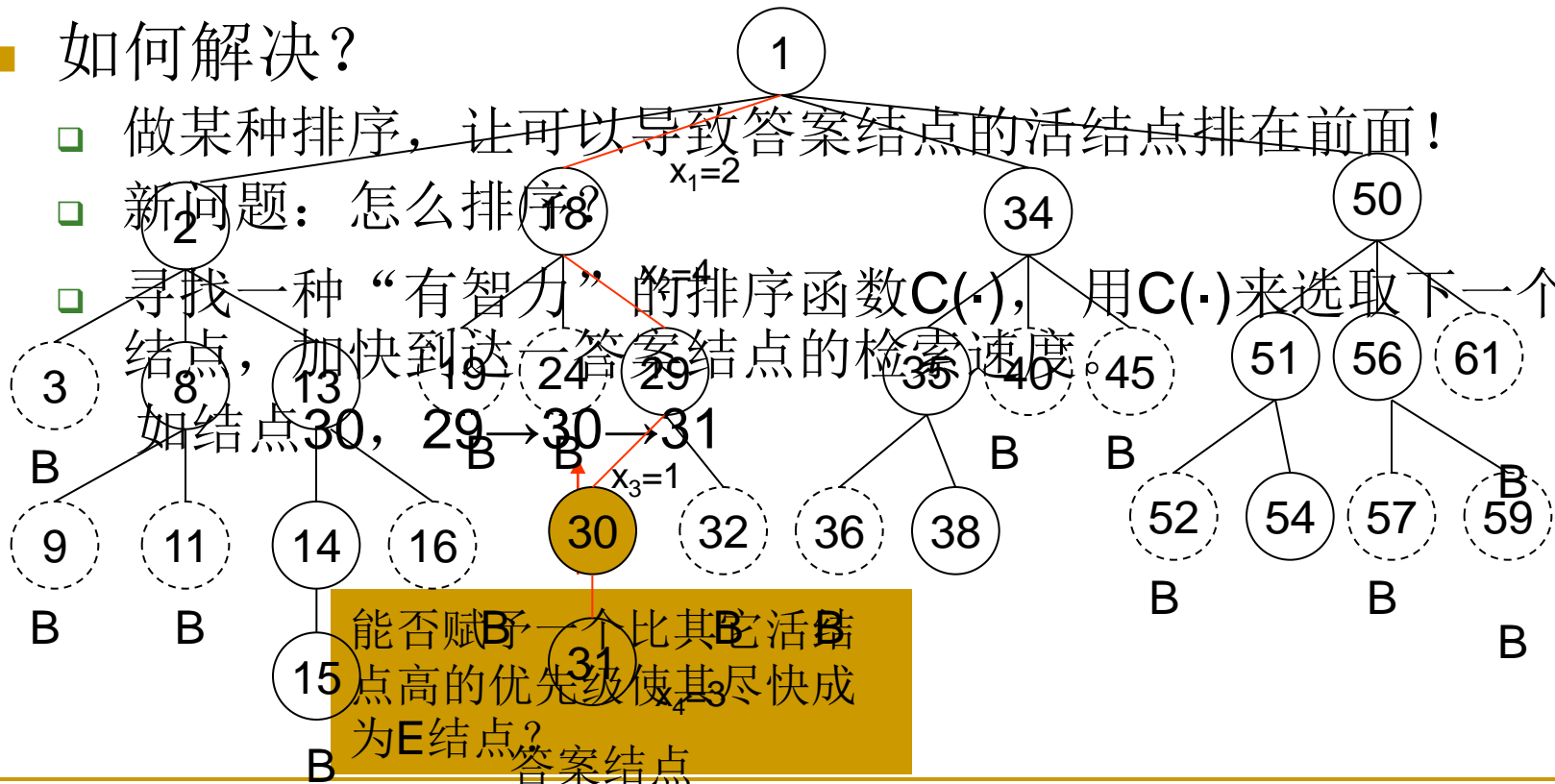
9.1.1 LC-检索 (Least Cost)

■ LIFO和FIFO分枝-限界法存在的问题

对下一个E-结点的选择规则过于死板、盲目。对于有可能快速检索到一个答案结点的结点没有给出任何优先权。如结点30。

■ 如何解决？

- 做某种排序，让可以导致答案结点的活结点排在前面！
- 新问题：怎么排序？
- 寻找一种“有智力”的排序函数 $C(\cdot)$ ，用 $C(\cdot)$ 来选取下一个E结点，加快到达答案结点的检索速度。



■ 如何衡量结点的优先等级？

- 对于任一结点，用该结点导致答案结点的成本（代价）来衡量该结点的优先级——成本越小越优先。
- 对任一结点X，可以用两种标准来衡量结点的代价：
 - 1) 在生成一个答案结点之前，子树X需要生成的结点数。
 - 2) 在子树X中离X最近的那个答案结点到X的路径长度。

■特点：

标准1：偏向于选择生成儿子结点数目最小的结点作为**E**结点。

标准2：偏向于选择由根到**最近**的那个答案结点路径上的结点作为**E**结点进行扩展。

□ 对任一结点X，可以用两种标准来衡量结点的代价：

1) 在生成一个答案结点之前，子树X需要生成的结点数。

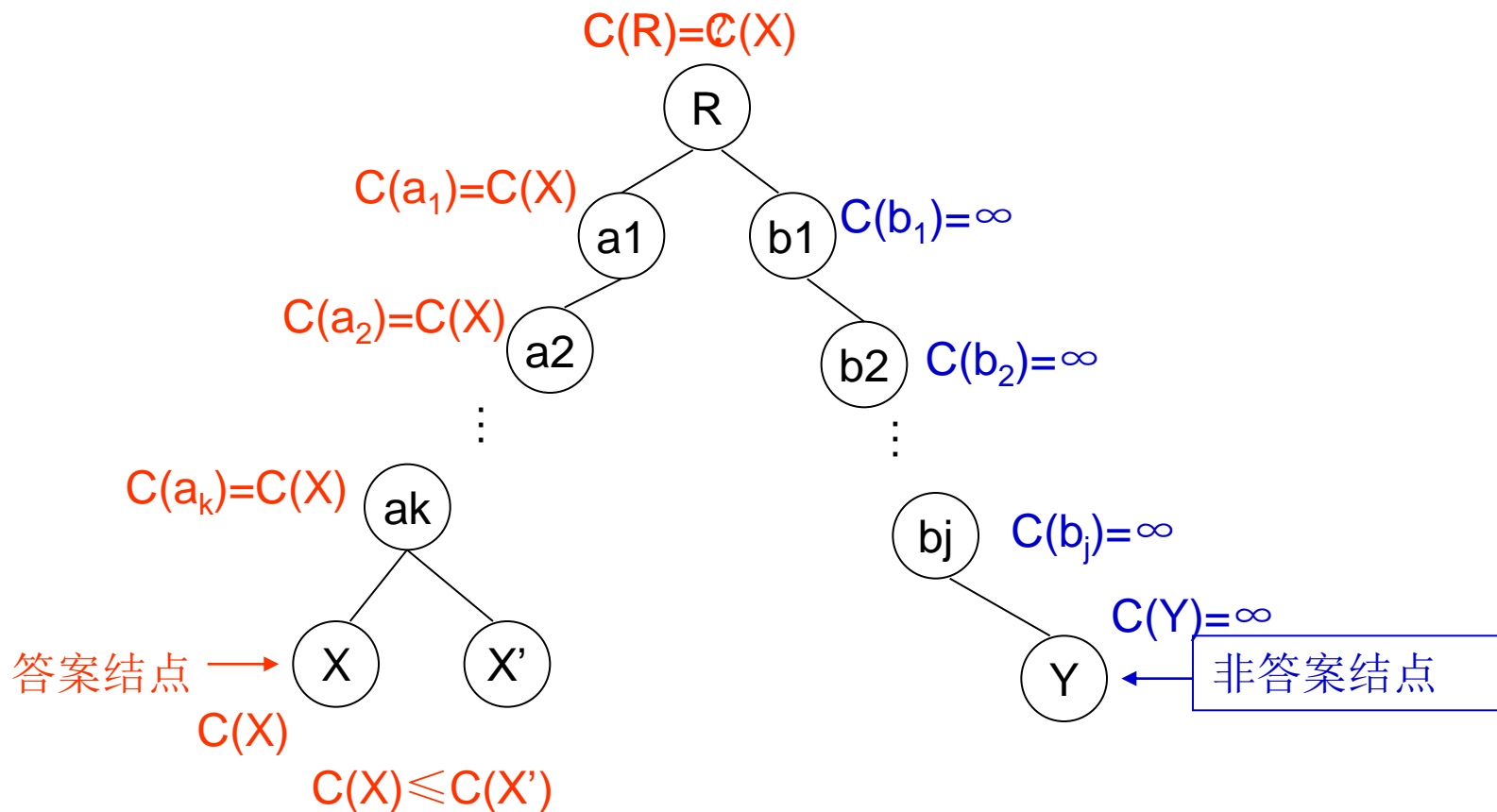
2) 在子树X中离X最近的那个答案结点到X的路径长度。

结点成本函数

- $C(\cdot)$: “有智力”的排序函数，依据成本排序，优先选择成本最小的活结点作为下一个扩展的E结点。

$C(\cdot)$ 又称为“**结点成本函数**”

- 结点成本函数 $C(X)$ 的定义:
 - 1) 如果 X 是答案结点，则 $C(X)$ 是由状态空间树的根结点到 X 的成本(即所用的代价，可以是级数、计算复杂度等)
 - 2) 如果 X 不是答案结点且子树 X 不包含任何答案结点，则 $C(X) = \infty$
 - 3) 如果 X 不是答案结点但子树 X 包含答案结点，则 $C(X)$ 等于子树 X 中具有最小成本的答案结点的成本



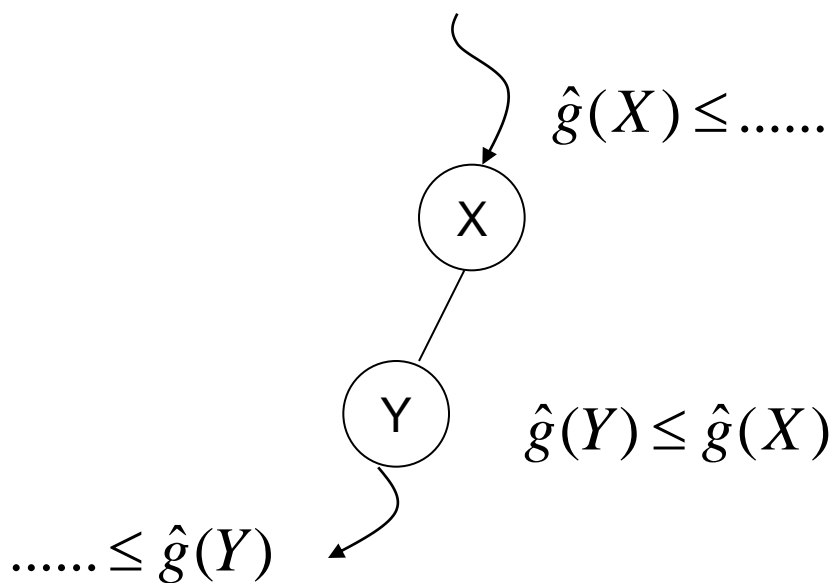
结点成本函数的计算

计算结点成本函数的困难

- 计算一个结点的代价通常要检索包含一个答案结点的子树 X 才能确定，因此计算 $C(X)$ 的工作量和复杂度与解原始问题是相同的。
- 计算结点成本的精确值是不现实的——相当于求解原始问题。
- 结点成本的估计函数
包括两部分： $\hat{g}(X)$ 和 $h(X)$

$\hat{g}(X)$: 设 $\hat{g}(X)$ 是由X到达一个答案结点所需成本的估计函数。

- 性质：单纯使用 $\hat{g}(X)$ 选择E结点会导致算法偏向纵深检查。



$\hat{g}(\bullet)$ 是X到答案结点的最小成本

纵深检索：直到子树X全部检索完才可能生成那些除了X子树以外的子树结点。

1) 如果 $\hat{g}(X) = C(X)$,
最理想！

2) 否则，可能导致不能很快地找到更靠近根的答案结点。

特例： $\hat{g}(W) < \hat{g}(Z)$

但Z比W更接近答案结点。

如何避免单纯考虑 $\hat{g}(X)$ 造成的纵深检查？

引进 $h(X)$ 改进成本估计函数。

- $h(X)$: 根结点到结点 X 的成本。

改进的结点成本估计函数 $\hat{c}(X)$

$$\hat{c}(X) = f(h(X)) + \hat{g}(X)$$

- $f(\cdot)$ 是一个非降函数。
- 非零的 $f(\cdot)$ 可以减少算法作偏向于纵深检查的可能性，它**强使**算法优先检索**更靠近答案结点**但又**离根较近**的结点。

LC-检索： 选择 $\hat{c}(\bullet)$ 值最小的活结点作为下一个**E-结点**。

特例：

□ **BFS：** 依据级数来生成结点，令

$$\hat{g}(X)=0; \quad f(h(X)) = X \text{的级数}$$

□ **D-Search：** 令 $f(h(X)) = 0$ ；而当 Y 是 X 的一个儿子时，
总有 $\hat{g}(X) \geq \hat{g}(Y)$ 。

LC分枝-限界检索： 伴之有**限界函数**的**LC-检索**

9.1.2 15-谜问题（问题描述）

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

(a) 一种初始排列



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(b) 目标排列

问题描述： 在一个分成16格的方形棋盘上放有15块编了号的牌。对于这些牌给定的一种**初始排列**（如图(a)），要求通过一系列的**合法移动**将初始排列转换成**目标排列**（图(b)）。

合法移动： 每次将一个邻接于空格的牌移动到空格位置。

- 问题状态：15块牌在棋盘上的任一种排列。
 - 初始状态：初始排列（任意给定的）
 - 目标状态：目标排列（确定的）
- 棋盘存在 $16!$ 种（约20万亿种）不同排列，而对于任一给定的初始状态，可到达的状态为这些排列中的一半。
- 若由初始状态到某状态存在一系列合法移动，则称该状态可由初始状态到达。
- 目标状态是否可由初始状态到达？

在求解问题前，首先需要判定目标状态是否在初始状态的状态空间中。

注：并不是所有的初始状态都能变换成目标状态的。

如何判定目标状态在初始状态的状态空间中？

判定条件由以下4步给出：

- 1) 给棋盘的方格位置编号：按目标状态各块牌在棋盘上的排列给对应方格编号，空格为16。

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

目标排列

2) 记 **POSITION(i)** 为编号为 i 的牌在初始状态中的位置;
POSITION(16) 表示空格的位置。

□ 例图(a)

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

$\text{POSITION}(1:16) = (1, 5, 2, 3, 7, 10, 9, 13, 14, 15, 11, 8, 16, 12, 4, \textcolor{red}{6})$

3) 记 **LESS(i)** 是这样牌 j 的数目:

$j < i$, 但 $\text{POSITION}(j) > \text{POSITION}(i)$,

即: 编号小于 i 但初始位置在 i 之后的牌的数目。

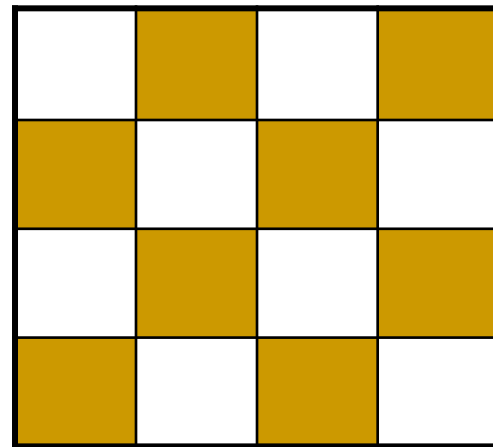
□ 例: $\text{LESS}(1)=0$; $\text{LESS}(4)=1$; $\text{LESS}(12)=6$

4) 引入一个量X

如图所示，初始状态时，

若空格落在橙色方格上，则 $X=1$ ：

若空格落在白色方格上，则 $X=0$ 。



目标状态是否在初始状态的状态空间中的判别条件由定理7.1给出：

定理7.1 当且仅当 $\sum_{i=1}^{16} LESS(i) + X$ 是偶数时，

目标状态可由此初始状态到达。

证明 （略）

15 迷问题状态空间树的构造

从初始状态出发，每个结点的儿子结点表示由该状态通过一次合法的移动而到达的状态。

注：移动牌与移动空格是等效的。状态空间树中标注空格的一次合法移动。

空格的一次合法移动有四个可能的方向：

上、下、左、右。

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

15-谜问题实例

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

初始状态

剪枝策略：若P的儿子中有与P的父结点重复的，则剪去该分枝。

FIFO检索

结点编号: 1

初始状态

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

上

左

2

3

右

下

4

5

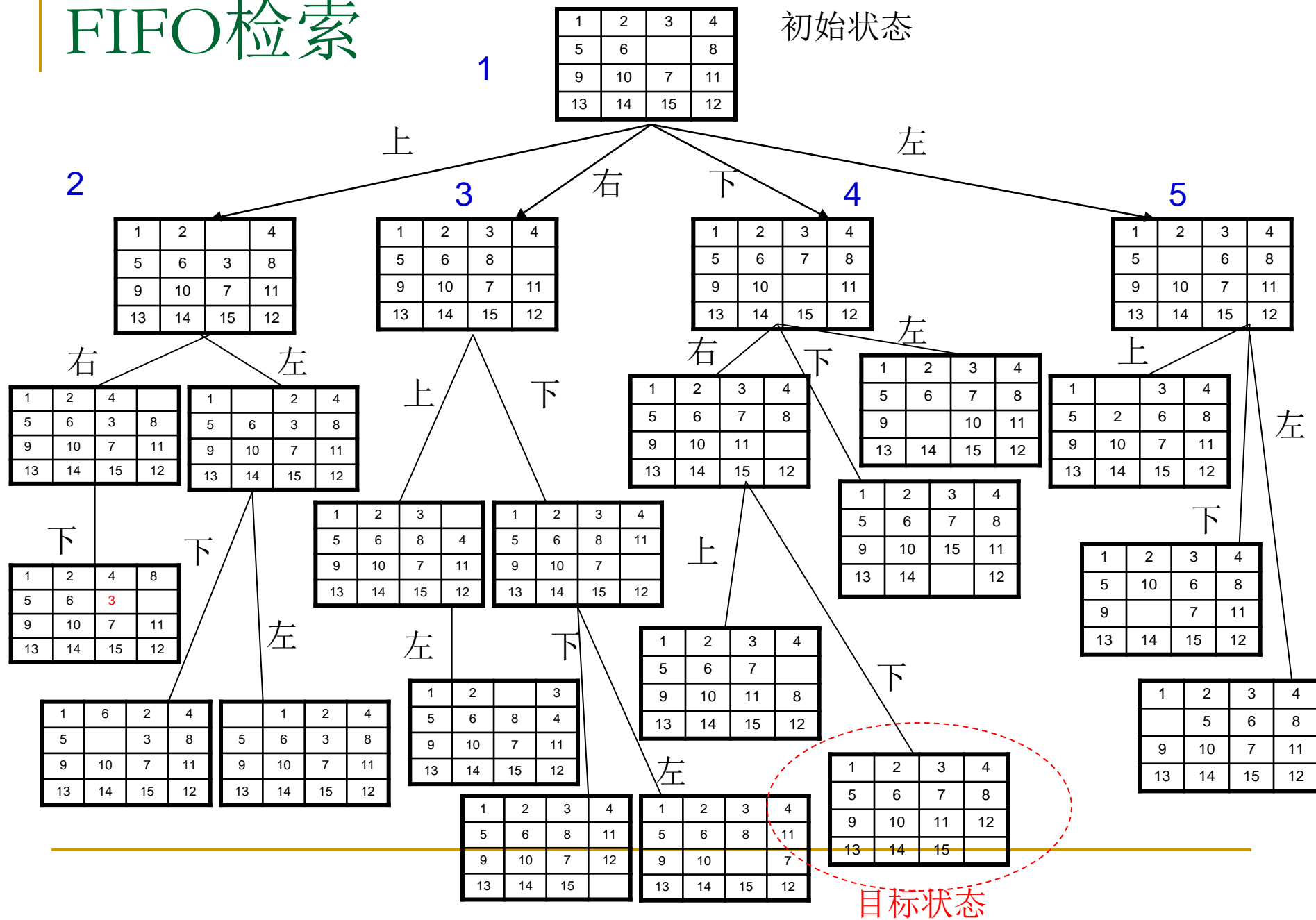
1	2		4
5	6	3	8
9	10	7	11
13	14	15	12

1	2	3	4
5	6	8	
9	10	7	11
13	14	15	12

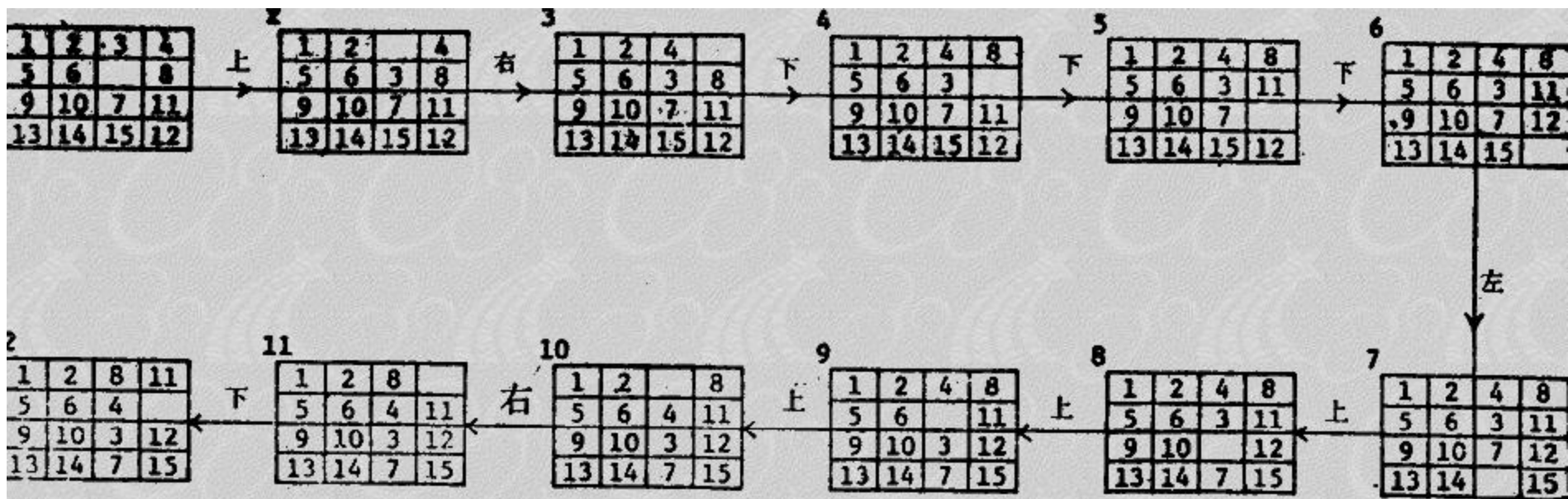
1	2	3	4
5	6	7	8
9	10		11
13	14	15	12

1	2	3	4
5		6	8
9	10	7	11
13	14	15	12

FIFO检索



15-谜问题深度优先检索



■ 简单检索存在的问题：呆板和盲目

是否能够找到一种“智能”方法，对不同的实例做不同的处理？

策略：给状态空间树中的每个结点赋予成本值 $c(X)$

如何定义 $c(X)$ ？

如果实例有解，则将由根出发到最近目标结点路径上的每个结点赋以这条路径的长度作为它们的成本。

例：

- $c(1)=c(4)=c(10)=c(23)=3$
- 其余结点的成本均为 ∞
- 检索时杀死成本为 ∞ 的结点

该方法实际上是不可操作的—— $c(X)$ 不可能通过简单的方法求出。精确计算 $c(X)$ 的工作量与求解原始问题相同

估算法：定义成本估计函数 $\hat{c}(X)$

$$\hat{c}(X) = f(X) + \hat{g}(X)$$

其中，

- 1) $f(X)$ 是由根到结点 X 的路径长度
- 2) $\hat{g}(X)$ 是以 X 为根的子树中由 X 到目标状态的一条最短路径长度的估计值—— $\hat{g}(X)$ 至少应是能把状态 X 转换成目标状态所需的最小移动数。故，令

$\hat{g}(X)$ = 不在其目标位置的非空白牌数目

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

例： $\hat{g}(X) = ?$

$\hat{c}(X)$ 是 $\mathbf{c}(X)$ 的下界

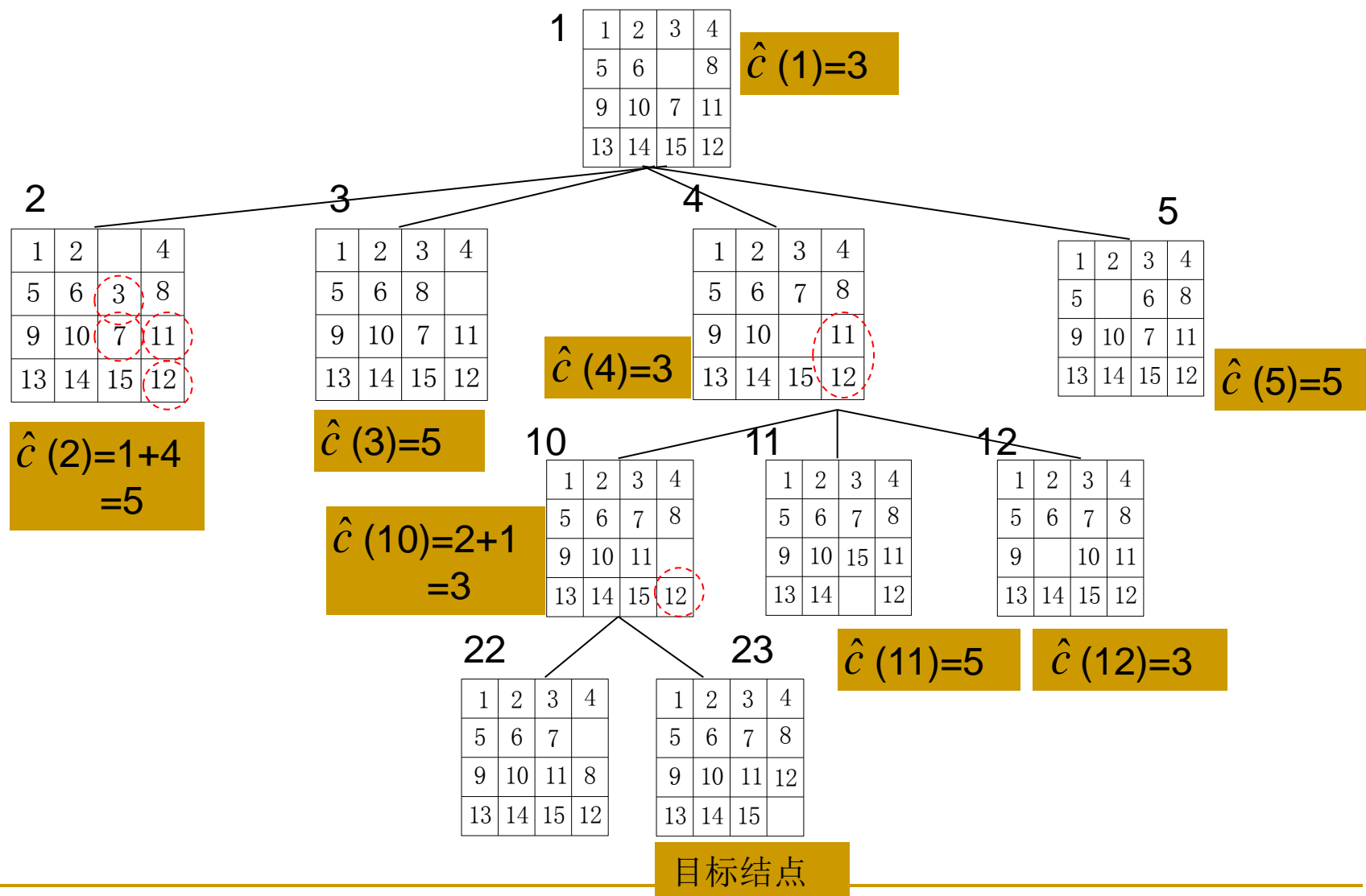
1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

例： 7、11、12号牌不在其位，故 $\hat{g}(X)=?$

注： 为达到目标状态所需的实际移动数 $\gg \hat{g}(X)$

$\hat{c}(X)$ 是 $c(X)$ 的下界

例：使用 $\hat{c}(X)$ 的LC-检索



使用 $\hat{c}(X)$ 的LC-检索可以使检索过程很快地定位到结点23。

9.1.3* LC-检索的抽象化控制

设： T 是一棵状态空间树

$c(\cdot)$ 是 T 的结点成本函数

$\hat{c}(X)$ 是成本估计函数。 $\hat{c}(X)$ 具备以下性质：

如果 X 是一个答案结点或者是一个叶结点，
则 $c(X) = \hat{c}(X)$ 。

LC-检索的抽象化控制：过程 LC 用 $\hat{c}(\bullet)$ 去寻找
一个答案结点

LC-检索的抽象化控制

procedure LC(T, \hat{c})

//为找答案结点检索 T, \hat{c} 为结点成本估计函数//

if T 是答案结点 then 输出 T ; return endif // T 为答案结点, 输出 T //

$E \leftarrow T$ //E—结点//

将活结点表初始化为空

loop

for E 的每个儿子 X do

if X 是答案结点 then 输出从 X 到 T 的路径; return endif

call ADD(X) // X 是新的活结点, ADD将 X 加入活结点表中//

PARENT(X) $\leftarrow E$ //指示到根的路径//

repeat

if 不再有活结点 then print("no answer code"); stop endif

call LEAST(E) //从活结点表中找 \hat{c} 最小的活结点, 赋给 X , 并从活结点表中删除//

repeat

end LC

找到答案结点,
输出到根的路径

说明：

LEAST(X): 在活结点表中找一个具有最小值的活结点，从活结点表中删除这个结点，并将此结点放在变量X中返回。

ADD(X): 将新的活结点X加到活结点表中。

活结点表：以min-堆结构存放。

LC的正确性证明

- 算法LC是正确的。

- 1) 满足确定性、能行性，有输入、输出。
- 2) 满足终止性——对于有限状态空间树，在找到一个答案结点或者在生成并检索了整棵状态空间树后算法终止。对于无限状态空间树，在找到一个答案结点或者对成本估计函数做出适当限制后也能使得算法终止。

因此LC算法正确。

（详细证明略）

LC-检索与FIFO-检索和D-检索的关系

1) LC算法与FIFO-检索及D-检索基本相同:

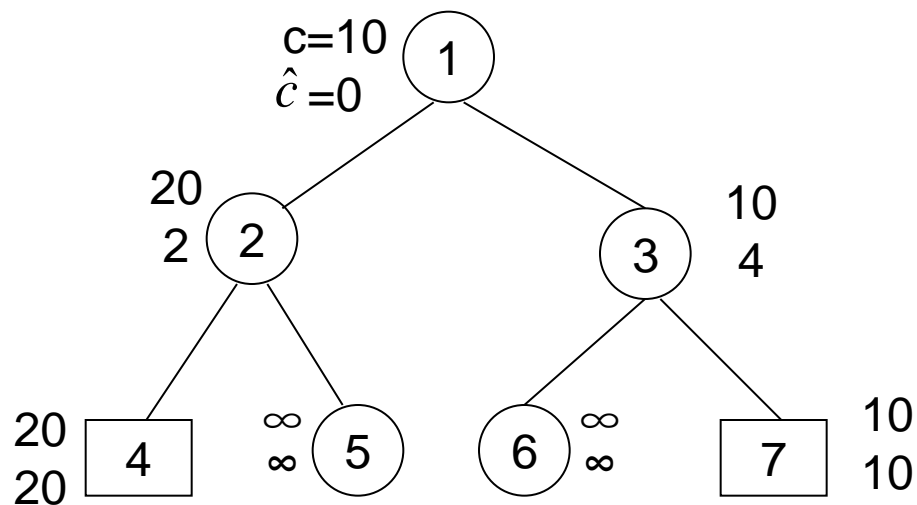
- ① 若活结点表采用**队列**，用**LEAST(X)**和**ADD(X)**从队列中删除或加入元素，则LC就变成了 FIFO-检索
- ② 若活节点表采用**栈**，用**LEAST(X)**和**ADD(X)**从栈中删除或加入元素，则LC就变成了 D-检索

2) 不同:

对下一个E-结点的选择规则不同。

LC-检索的特性

- 当有多个答案结点时，LC是否一定找得到具有最小成本的答案结点呢？ 否



首先扩展结点1，得结点2，3； $\hat{c}(2) = 2 < \hat{c}(3) = 4$ ；

然后扩展结点2，得答案结点4， $c(4)=20$ ；

实际最小成本的答案结点是7， $c(7)=10$

原因：

存在这样的结点X和Y：当 $c(X) > c(Y)$ 时， $\hat{c}(X) < \hat{c}(Y)$

改进策略1：

约定：对每一对 $c(X) < c(Y)$ 的结点X和Y，有 $\hat{c}(X) < \hat{c}(Y)$

目标：使得LC总会找到一个最小成本的答案结点
(如果状态空间树中有答案结点的话)。

定理7.2 在有限状态空间树T中，对于每一个结点X，令 $\hat{c}(X)$ 是 $c(X)$ 的估计值且具有以下性质：对于每一对结点Y、Z，当且仅当 $c(Y) < c(Z)$ 时，有 $\hat{c}(Y) < \hat{c}(Z)$ 。那么在使 $\hat{c}()$ 作为 $c()$ 的估计值时，算法LC到达一个最小的成本答案结点终止。

证明：（略）

改进策略2:

对结点成本函数做如下限定：对于每一个结点X有 $\hat{c}(X) \leq c(X)$ 且对于答案结点X有 $\hat{c}(X) = c(X)$ 。

算法LC1：找最小成本答案结点的改进算法，该算法可以找到成本最小的答案结点。

找最小成本答案结点的算法

procedure LC1(T, \hat{c})

//为找出最小成本答案结点检索 T , \hat{c} 为具有上述性质的结点成本估计函数//

$E \leftarrow T$ //第一个 E —结点//

置活结点表为空

loop

if E 是答案结点 then 输出从 E 到 T 的路径; return endif

for E 的每个儿子 X do

call ADD(X) // X 是新的活结点, ADD将 X 加入活结点表中//

PARENT(X) $\leftarrow E$ //指示到根的路径//

repeat

if 不再有活结点 then print(“no answer code”); stop endif

call LEAST(E) //从活结点表中找 \hat{c} 最小的活结点, 赋给 X , 并从活结点表中删除//

repeat

end LC1

关于算法LC1正确性的证明

- 定理9.3 令 $\hat{c}(\bullet)$ 是满足如下条件的函数，在状态空间树T中，对于每一个结点X，有 $\hat{c}(X) \leq c(X)$ ，而对于T中的每一个答案结点X，有 $\hat{c}(X) = c(X)$ 。如果算法在第一个return处终止，则所找到的答案结点是具有最小成本的答案结点。

证明：（略）

成本函数在分枝-限界算法中的应用

假定每个答案结点 X 有一个与其相联系的 $c(X)$ ，且找成本最小的答案结点。

1) 最小成本的下界： $\hat{c}(\bullet)$

$\hat{c}(X)$ 为 X 的成本估计函数。当 $\hat{c}(X) \leq c(X)$ 时， $\hat{c}(X)$ 给出由结点 X 求解的最小成本的下界——启发性函数，减少选取 E 结点的盲目性。

2) 最小成本的上界： U

能否定义最小成本的上界？

最小成本的上界

定义 U 为最小成本解的**成本上界**，则：

对具有 $\hat{c}(X) > U$ 的所有活结点可以被杀死，从而可以进一步使算法加速，减少求解的盲目性。

注：

根据 $c(X)$ 的定义，由那些 $\hat{c}(X) > U$ 的结点 X 可到达的所有答案结点必有 $c(X) \geq \hat{c}(X) \geq U$ 。故，当已经求得一个具有成本 U 的答案结点，那些有 $\hat{c}(X) > U$ 的所有活结点都可以被杀死——这些结点不可能具有更小的成本。

最小成本上界 U 的求取：

- 1) 初始值：利用启发性方法赋初值，或置为 ∞
- 2) 每找到一个新的答案结点后修正 U ， U 取当前最小成本值。

注：只要 U 的初始值**不小于**最小成本答案结点的成本，利用 U 不会杀死可以到达最小成本答案结点的活结点。

利用分枝-限界算法求解最优化问题

最优化问题：求能够使目标函数取最小（大）值的可行解——最优解。

如何把求最优解的过程表示成与分枝-限界相关联的检索过程？

- 可行解：类似于 n -元组的构造，把可行解可能的构造过程用“状态空间树”表示出来。
- 最优解：把对最优解的检索表示成对状态空间树答案结点的检索。
- 成本函数：每个结点赋予一个成本函数，并使得：代表最优解的答案结点的 $c(X)$ 是所有结点成本的最小值。

成本函数的定义：直接用目标函数作为成本函数 $c(\cdot)$ ：

1) 代表可行解结点的 $c(X)$ 就是该可行解的目标函数值。

2) 代表不可行解结点的 $c(X)=\infty$ ；

3) 代表部分解的结点的 $c(X)$ 是根为 X 的子树中最小成本结点的成本。

▶ 答案结点 \longleftrightarrow 可行解

▶ 成本最小的答案结点 \longleftrightarrow 最优解

▶ 成本估计函数 $\hat{c}(X)$ ，且要求有 $\hat{c}(X) \leq c(X)$

注： $\hat{c}(X)$ 根据目标函数进行估计分析。

实例：利用求解最优化问题的分枝-限界算法 求带限期的作业排序问题

1.问题描述：带限期的作业排序问题——更为一般化的定义

- 假定 n 个作业和一台处理机
- 每个作业 i 对应一个三元组 (p_i, d_i, t_i)

其中， t_i ：表示作业 i 需要 t_i 个单位处理时间——允许不同作业有不同的处理时间；

d_i ：表示完成期限；

p_i ：表示若 i 在期限内未完成招致的罚款。

求解目标：从 n 个作业的集合中选取子集 J ，要求 J 中所有作业都能在各自期限内完成并且使得不在 J 中的作业招致的罚款总额最小——最优解。

实例：

$$n = 4;$$

$$(p_1, d_1, t_1) = (5, 1, 1); \quad (p_2, d_2, t_2) = (10, 3, 2);$$

$$(p_3, d_3, t_3) = (6, 2, 1); \quad (p_4, d_4, t_4) = (3, 1, 1);$$

状态空间：问题的解空间由作业指标集 $(1, 2, 3, 4)$ 的所有可能子集合组成。

状态空间树：两种表示形式，

1) 元组大小可变的表示：

表示选了哪些作业，用作业编号表示。

2) 元组大小固定的表示：

表示是否选中作业，每个作业对应一个0/1值

$(p_1, d_1, t_1) = (5, 1, 1);$
 $(p_2, d_2, t_2) = (10, 3, 2);$
 $(p_3, d_3, t_3) = (6, 2, 1);$
 $(p_4, d_4, t_4) = (3, 1, 1);$

圆形结点都是答案结点(可行解)
方形结点是不可行的子集合

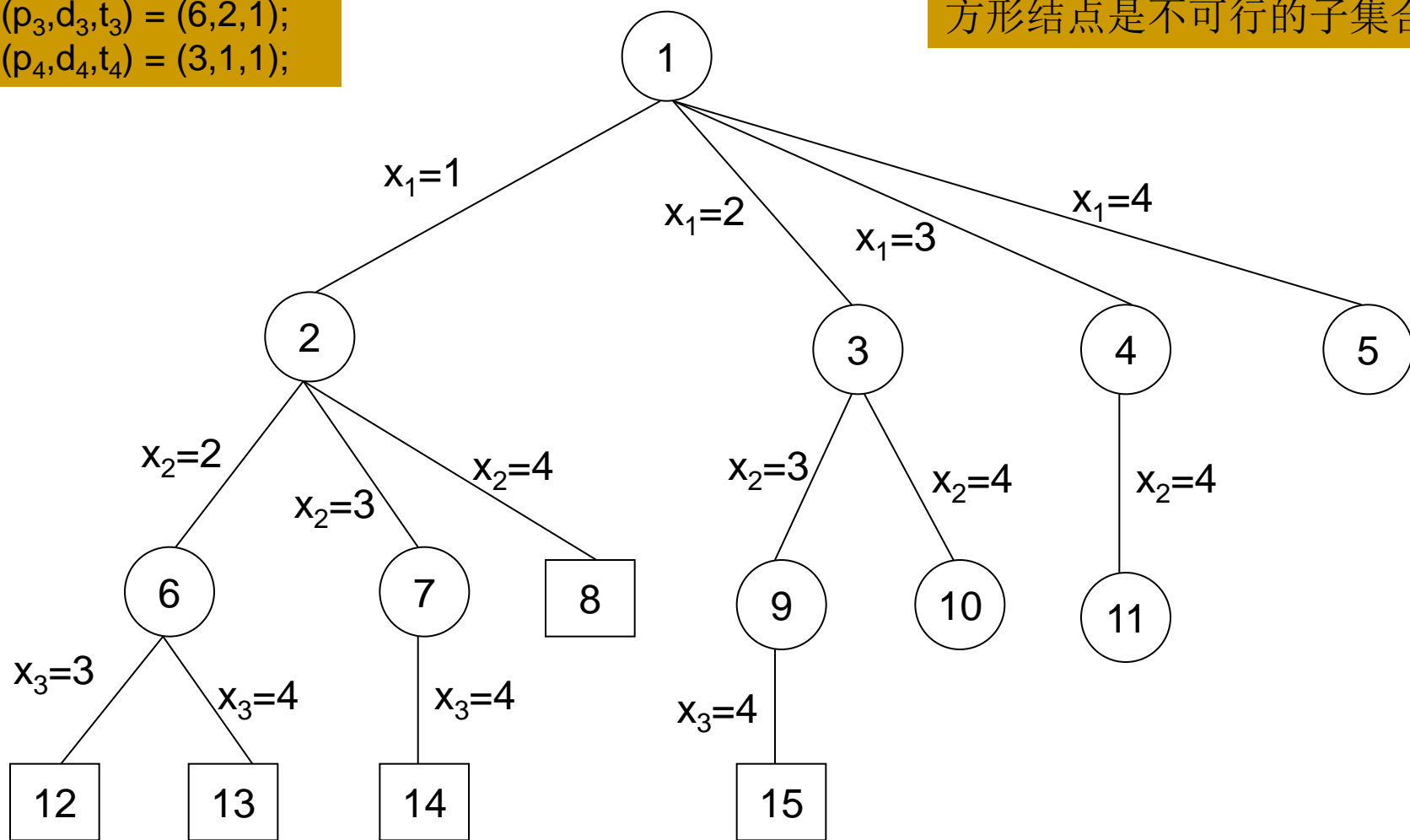
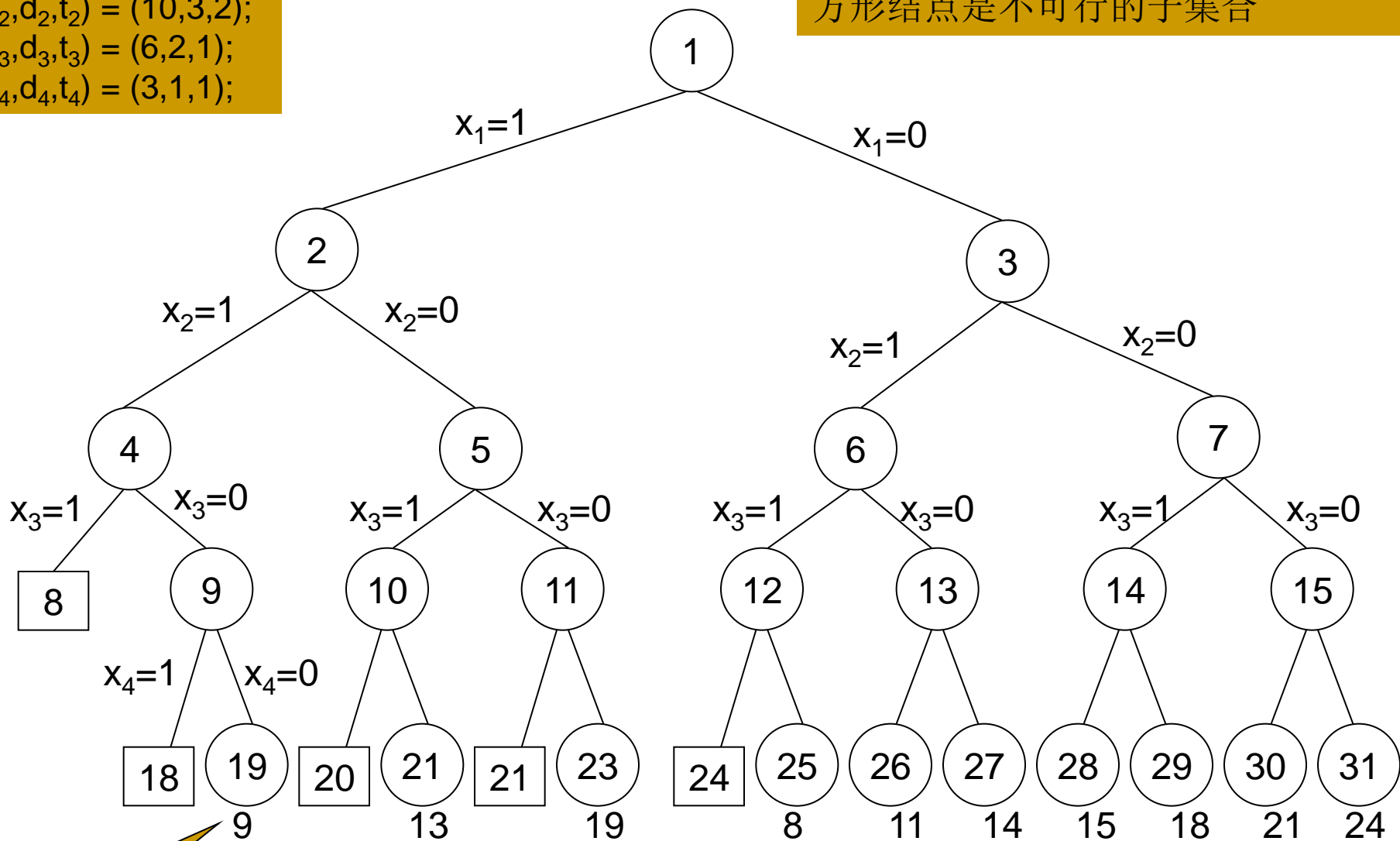


图9.7 采用大小可变的元组表示的状态空间树

$(p_1, d_1, t_1) = (5, 1, 1);$
 $(p_2, d_2, t_2) = (10, 3, 2);$
 $(p_3, d_3, t_3) = (6, 2, 1);$
 $(p_4, d_4, t_4) = (3, 1, 1);$

只有圆形叶结点都是答案结点
 方形结点是不可行的子集合



罚款值

图9.8 采用大小固定的元组表示的状态空间树

成本函数 $c(\cdot)$ 定义为:

- ▶ 对于圆形结点 X , $c(X)$ 是根为 X 的子树中结点的最小罚款;
- ▶ 对于方形结点, $c(X)=\infty$ 。

例: 如图

图9.7, $c(3)=8$, $c(2)=9$, $c(1)=8$

图9.8, $c(2)=9$, $c(5)=13$, $c(6)=8$, $c(1)=8$

$c(1)$ =最优解 J 对应的罚款

$(p_1, d_1, t_1) = (5, 1, 1);$
 $(p_2, d_2, t_2) = (10, 3, 2);$
 $(p_3, d_3, t_3) = (6, 2, 1);$
 $(p_4, d_4, t_4) = (3, 1, 1);$

最优解的罚款

$$c(X) = \sum_{i \notin J} p_i$$

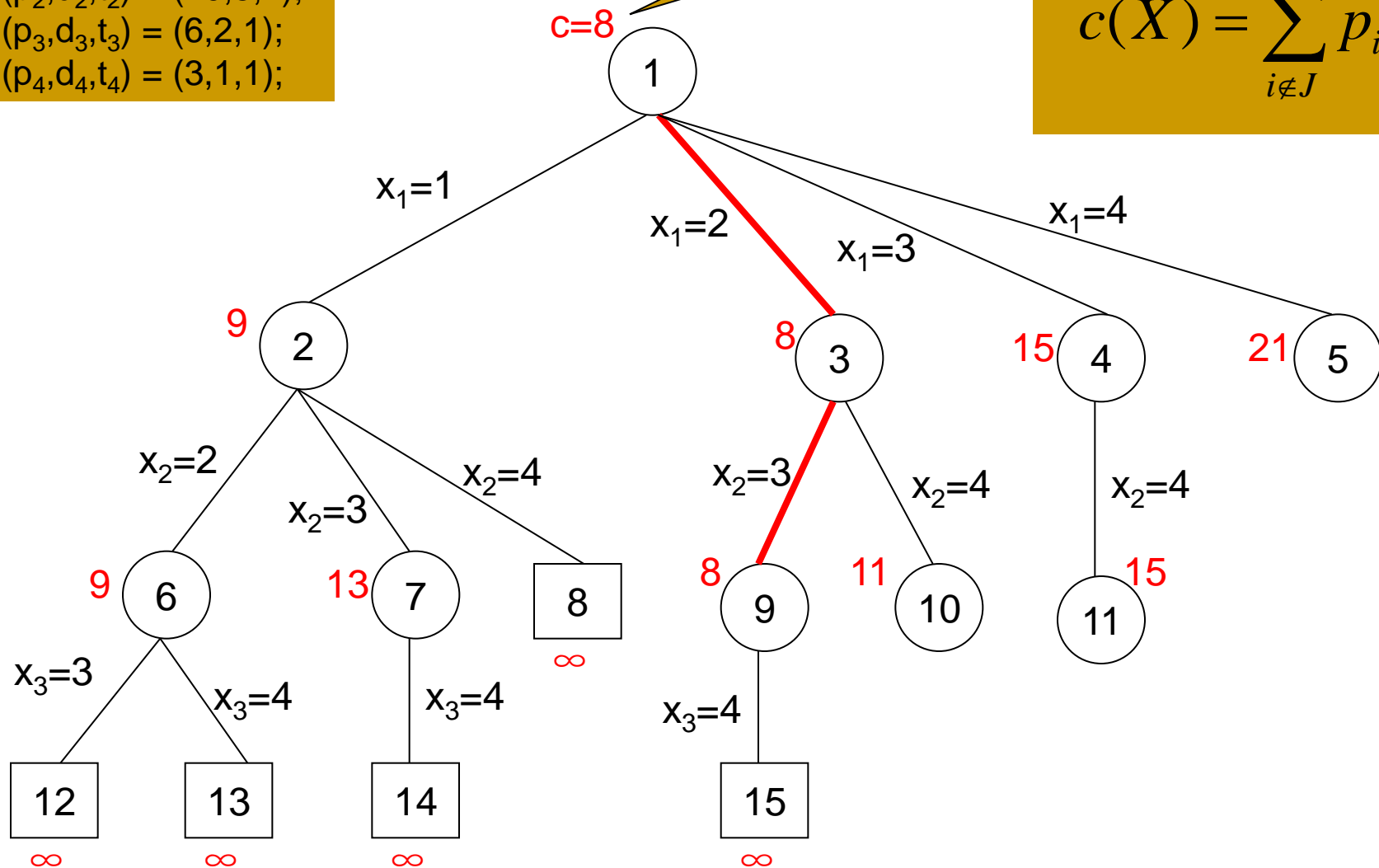
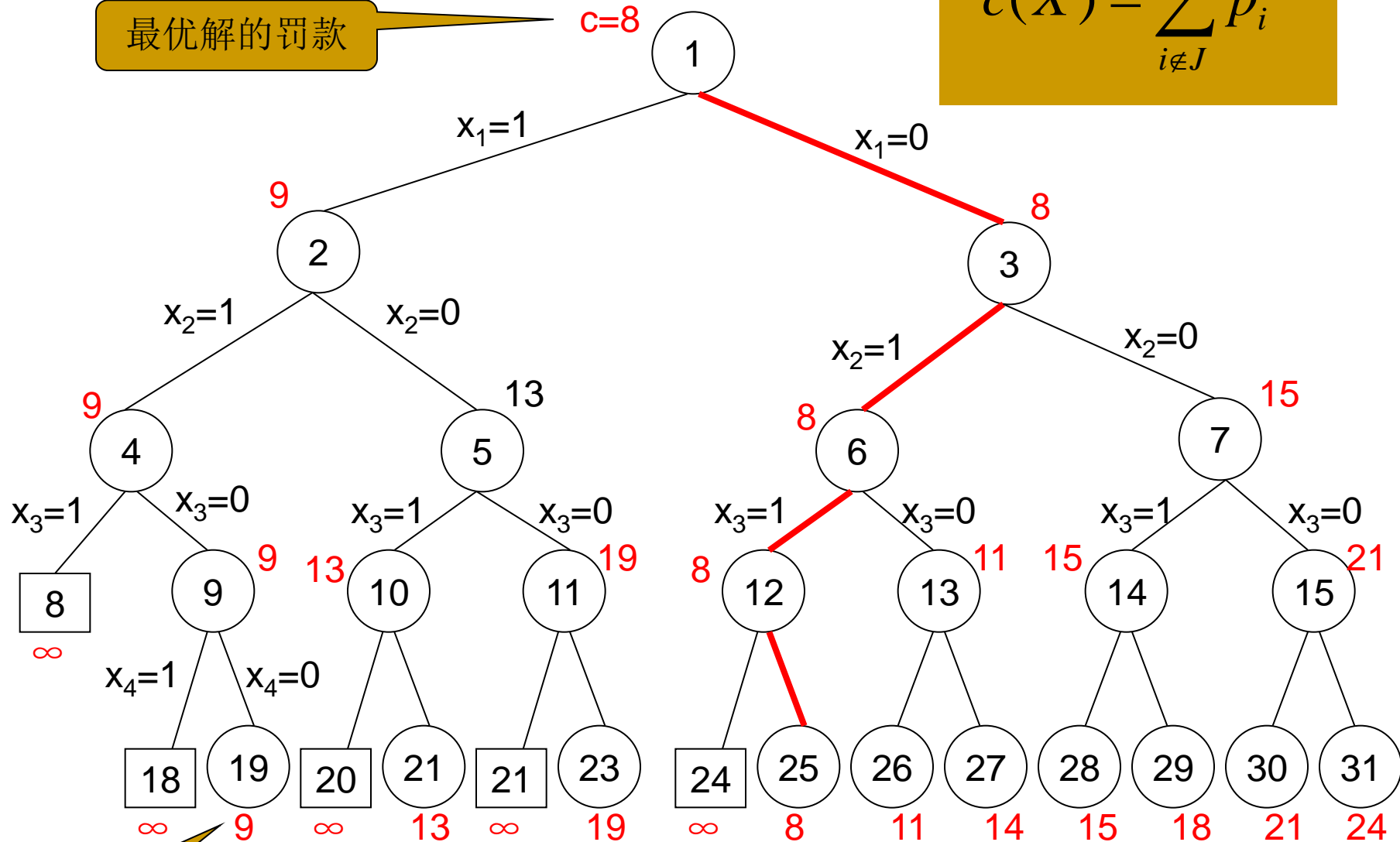


图9.7 采用大小可变的元组表示的状态空间树各结点的 c 值

最优解的罚款

$c=8$

$$c(X) = \sum_{i \notin J} p_i$$



罚款值

图9.8 采用大小固定的元组表示的状态空间树各结点的 c 值

成本估计函数 $\hat{c}(\bullet)$ 的定义

设 S_x 是在结点 X 时，已计入 J 中的作业的集合。

令 $m = \max \{i \mid i \in S_x\}$ ，则

按结点编号从小到大的顺序选择作业

$$\hat{c}(X) = \sum_{\substack{i < m \\ i \notin S_x}} p_i$$

是使 $c(X)$ 有 $\hat{c}(X) \leq c(X)$ 的估计值(下界)。

$\hat{c}(X)$ 的含义：已经被考虑过但没有被计入 J 中的作业的罚款合计——已确定的罚款数，目前罚款的底线。

例：见图

$$\hat{c}(X) = \sum_{\substack{i < m \\ i \notin S_X}} p_i$$

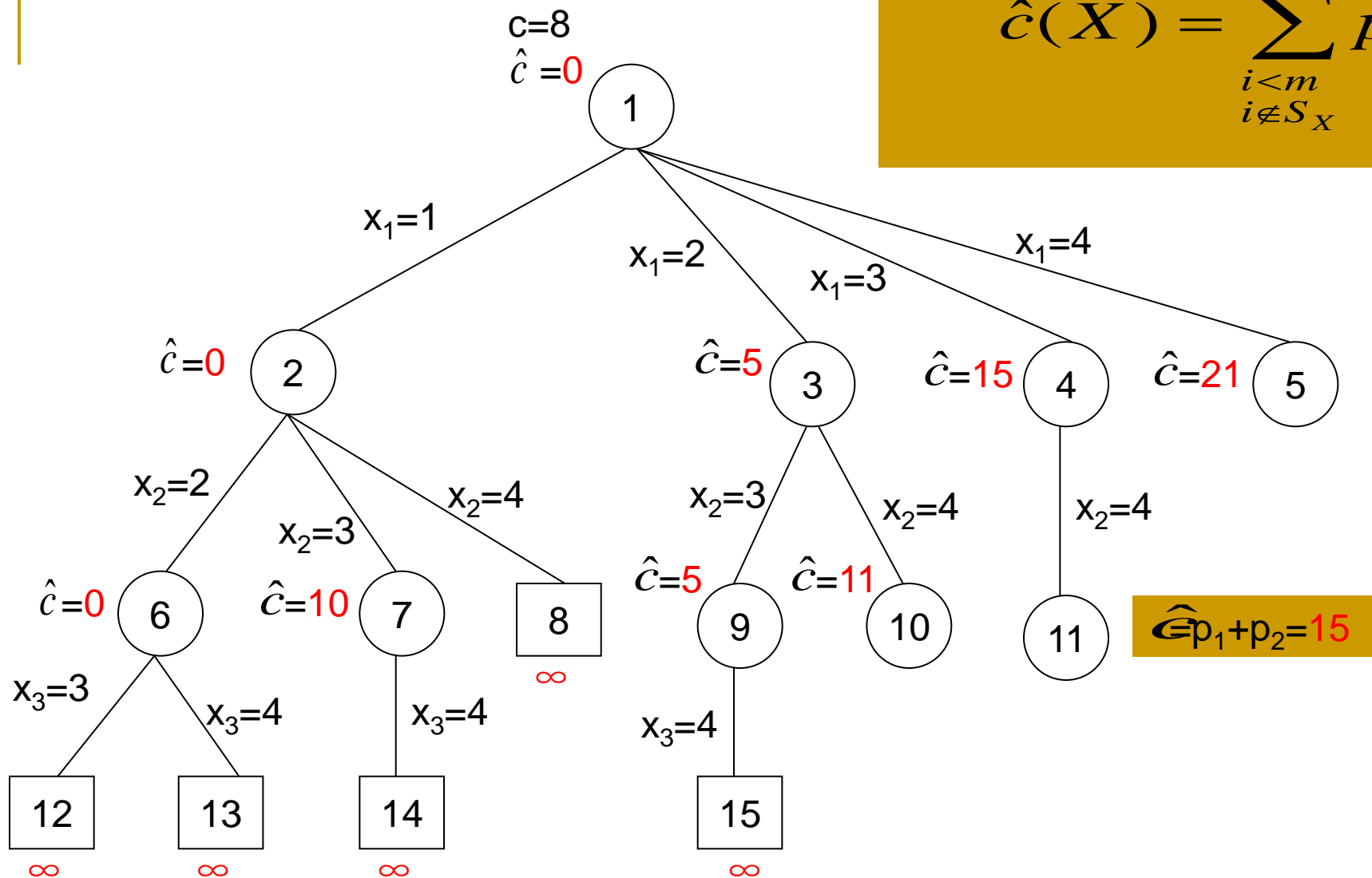


图9.7 采用大小可变的元组表示的状态空间树的成本估计值

$$\hat{c}(X) = \sum_{\substack{i < m \\ i \notin S_X}} p_i$$

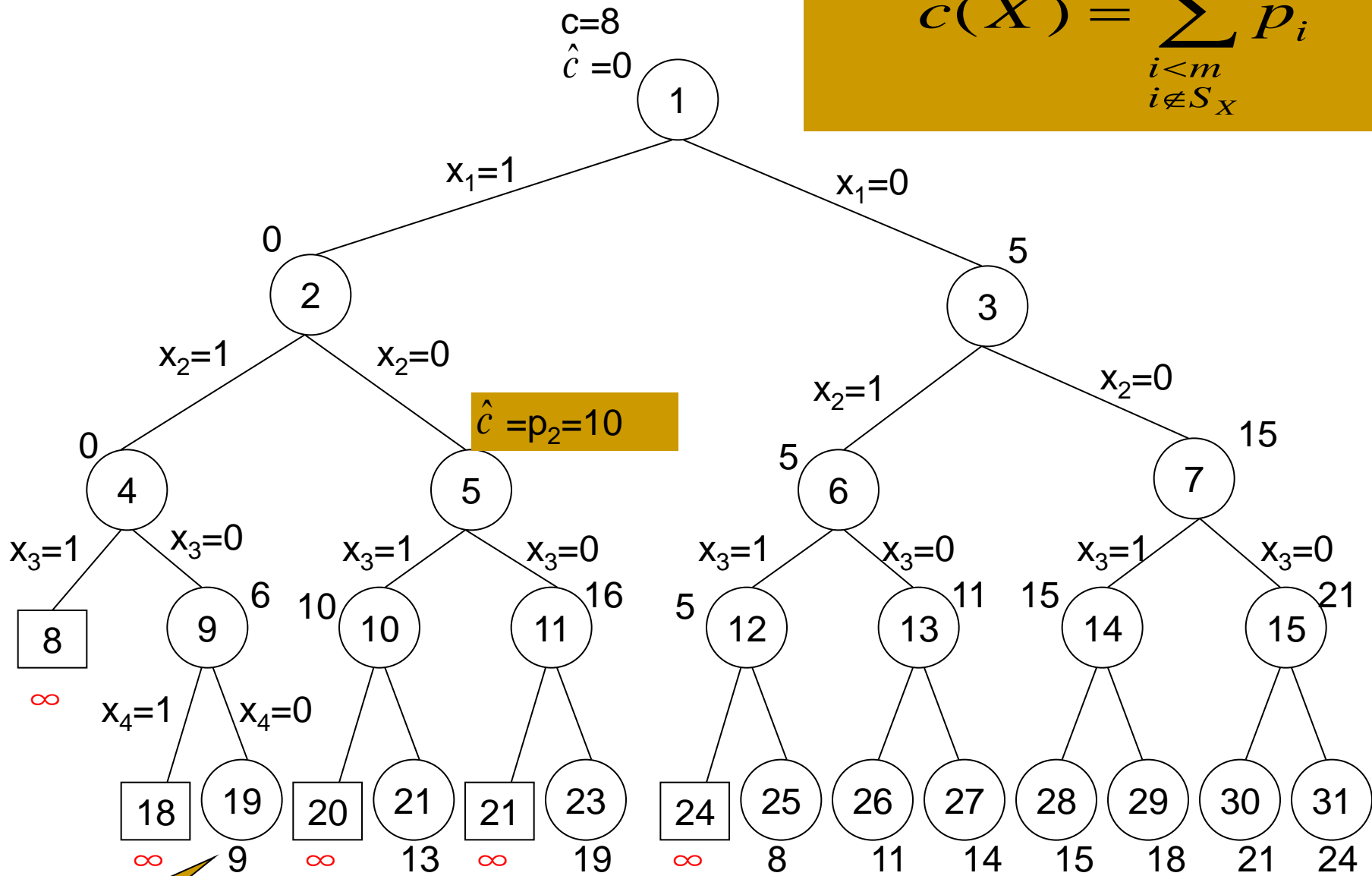


图9.8 采用大小固定的元组表示的状态空间树的成本估计值

罚款值

成本估计函数上界的定义

$$U(X) = \sum_{i \notin S_X} p_i$$

是 $c(X)$ 的一个“简单”上界。

$U(X)$ 的含义：目前没有被计入到 J 中的作业的罚款合计——可能的最多罚款，目前罚款的上线。

作业排序问题的FIFO分枝-限界算法图例分析

$$\hat{c}(X) = \sum_{\substack{i < m \\ i \notin S_X}} p_i$$

$$u(1) = \sum_{1 \leq i \leq n} p_i$$

$$u(X) = \sum_{i \notin S_X} p_i$$

$$U = \min(u(X))$$

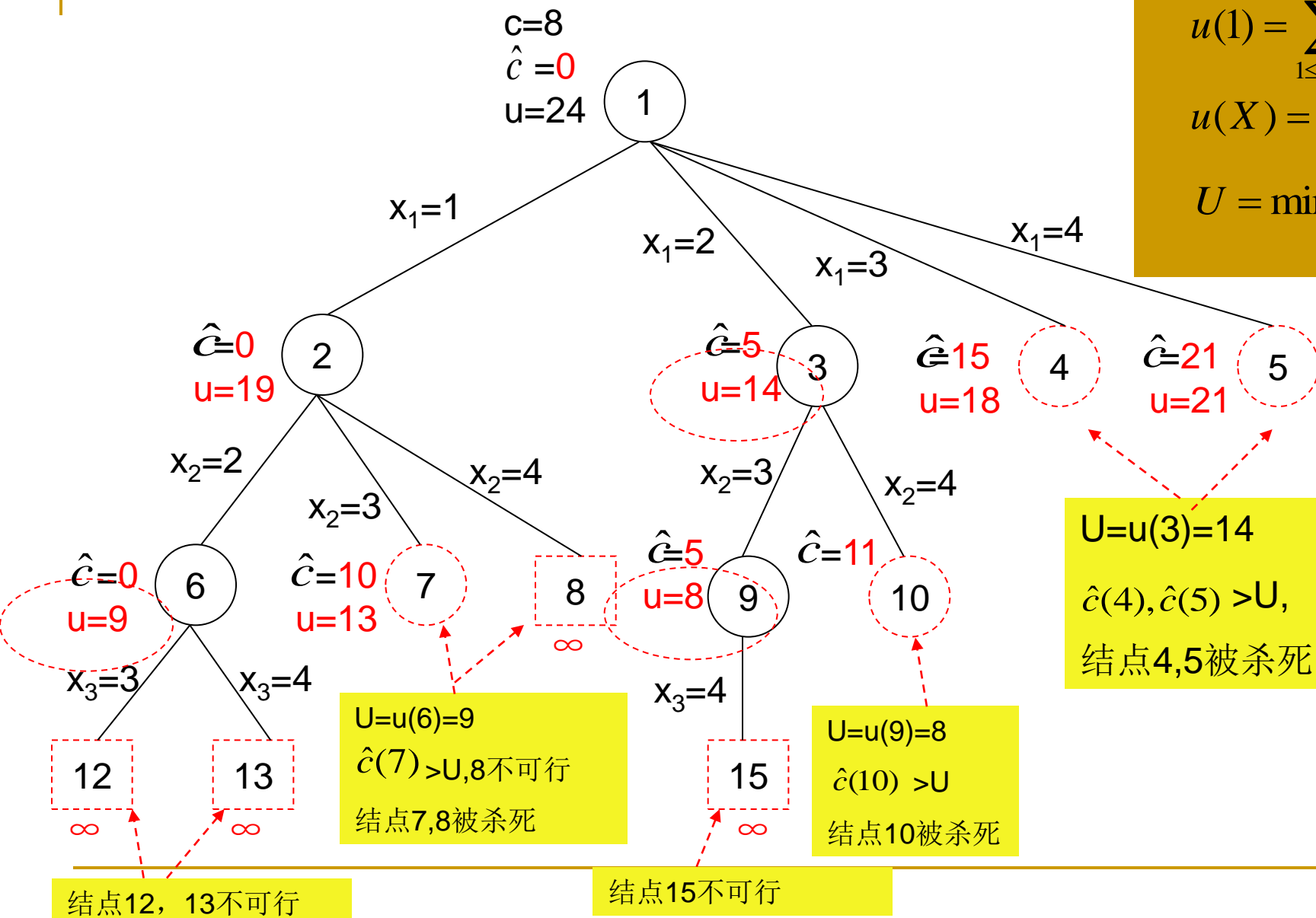


图9.7 采用大小可变的元组表示的状态空间树的成本估计值

作业排序问题的FIFO分枝-限界算法的实现

算设计说明

- 1) 每有更小的 $u(X)$ 计算出来时修正 U 值。
- 2) 当结点 X 从活结点表出来将变成 E 结点时,

● 若 $\hat{c}(X) > U$

X 被立即杀死。继续考虑活结点表中的其它活结点

● 若 $\hat{c}(X) = U$, 根据 U 的得来作如下处理:

① U 是一个已找到的解的成本: 杀死 X

注: 至少已经有了一个成本等于 U 的解, 这个 X 对最优解没有任何意义。

② U 是一个单纯的上界: X 成为 E 结点, 继续扩展

注: 当前的最好解还没找到, X 是有可能导致成本等于 U 的解的结点的。

(续)

3) 引进一个很小的正常数 ε , ε 足够小, 使得对于任意两个可行的结点X和Y, 若 $u(X) < u(Y)$, 则

$$u(X) < u(X) + \varepsilon < u(Y)$$

4) 在扩展E结点得到一个儿子结点X时若有 $\hat{c}(X) < U$:

① 若X是答案结点且 $c(X) < U$ 时,

$$U = \min(c(X), u(X) + \varepsilon)$$

注: 若 $u(X) < c(X)$, 则根为X的子树中有更优的解。

② 否则, $U = \min(U, u(X) + \varepsilon)$

$\hat{c}(X) \geq U$ 的儿子结点全部被杀死。

注: 引入 ε 的目的: 使U略大于 $u(x)$, 便于比较计算。

(续)

- 5) **ADDQ、DELETEQ**过程：将结点加入队列或从队列中删除。
- 6) **cost(X)**：若X是答案结点，则**cost(X)**是结点X对应的解的成本。
- 7) 可行结点的估计值 $\hat{c}(X) \leq c(X) \leq u(X)$
- 8) 不可行结点的 $\hat{c}(X) = \infty$

作业排序问题的FIFO分枝-限界算法FIFOBB

line procedure FIFOBB($T, \hat{c}, u, \varepsilon, \text{cost}$)

// 为找出最小成本答案结点检索 T , 假定 T 至少包含一个解结点且 $\hat{c}(X) \leq c(X) \leq u(X)$ //

- 1 $E \leftarrow T; \text{PARENT}(E) \leftarrow 0;$
- 2 if T 是解结点 then
 - $U \leftarrow \min(\text{cost}(T), u(T) + \varepsilon); \text{ans} \leftarrow T$
- 3 else $U \leftarrow u(T) + \varepsilon; \text{ans} \leftarrow 0$
- 4 endif
- 5 将队列置初值为空



ans指向最新解

(Continue...)

(continue)

```
6  loop
7      for E的每个儿子X do
8          if  $\hat{c}(X) < U$  then call ADDQ(X); PARENT(X)  $\leftarrow$  E
9              case
10                 :X是解结点 and  $\text{cost}(X) < U$ :
11                      $U \leftarrow \min(\text{cost}(X), u(X) + \epsilon)$ ;
12                      $\text{ans} \leftarrow X$ 
13                 :  $u(X) + \epsilon < U$ :  $U \leftarrow u(X) + \epsilon$ 
14             endcase
15         endif
16     repeat
```

(Continue...)

判定X的可解性
修正U

(continue)

```
17      loop //从队列中取下一个E-结点
18          if 队列为空 then print('least cost=', U)
19              while ans <> 0 do
20                  print(ans)
21                  ans ← PARENT(ans)
22              repeat
23              return
24          endif
25          call DELETEQ(X)
26          if  $\hat{c}(X) < U$  then exit
27      repeat
28  repeat
29end FIFOB
```

$\hat{c}(X) \geq U$ 的结点被从队列中清除（被杀死）

作业排序问题的LC分枝-限界算法LCBB

设计说明:

- 1) 采用min-堆保存活结点表
- 2) ADD、LEAST过程: 将结点加入到min-堆或从min-堆中删除。
- 3) 其余约定同于FIFO检索
- 4) 当min-堆中没有活结点或下一个E结点有 $\hat{c}(E) \geq U$ 情况下算法终止。

算法描述:

line procedure LCBB($T, \hat{c}, u, \epsilon, \text{cost}$)

// 为找出最小成本答案结点检索T, 假定T至少包含一个解结点且 $\hat{c}(X) \leq c(X) \leq u(X)$ //

- 1 $E \leftarrow T$; PARENT(E) $\leftarrow 0$;
- 2 if T是解结点 then
 - $U \leftarrow \min(\text{cost}(T), u(T) + \epsilon)$; ans $\leftarrow T$
- 3 else $U \leftarrow u(T) + \epsilon$; ans $\leftarrow 0$
- 4 endif
- 5 将队列置初值为空

(Continue...)


(continue)

```
6  loop
7      for E的每个儿子X do
8          if  $\hat{c}(X) < U$  then call ADDQ(X);
9              PARENT(X)  $\leftarrow$  E
10             case
11                 :X是解结点 and  $\text{cost}(X) < U$ :
12                      $U \leftarrow \min(\text{cost}(X), u(X) + \epsilon)$ ;
13                      $\text{ans} \leftarrow X$ 
14                 :  $u(X) + \epsilon < U$ :  $U \leftarrow u(X) + \epsilon$ 
15             endcase
16         endif
17     repeat
```

(Continue...)

判定X的可解性
修正U

```
18      if 不再有活结点 or 下一个E结点有  $\hat{c}(X) \geq U$ 
19          then  print('least cost=', U)
20                  while ans<>0 do
21                      print(ans)
22                      ans  $\leftarrow$  PARENT(ans)
23                  repeat
24                  return
25      endif
26      call LEAST(E)
27  repeat
28end LCBB
```



LC-检索选取最小 $\hat{c}(X)$ 的结点作为下一个E结点。

效率分析（略）

2005级课程结束

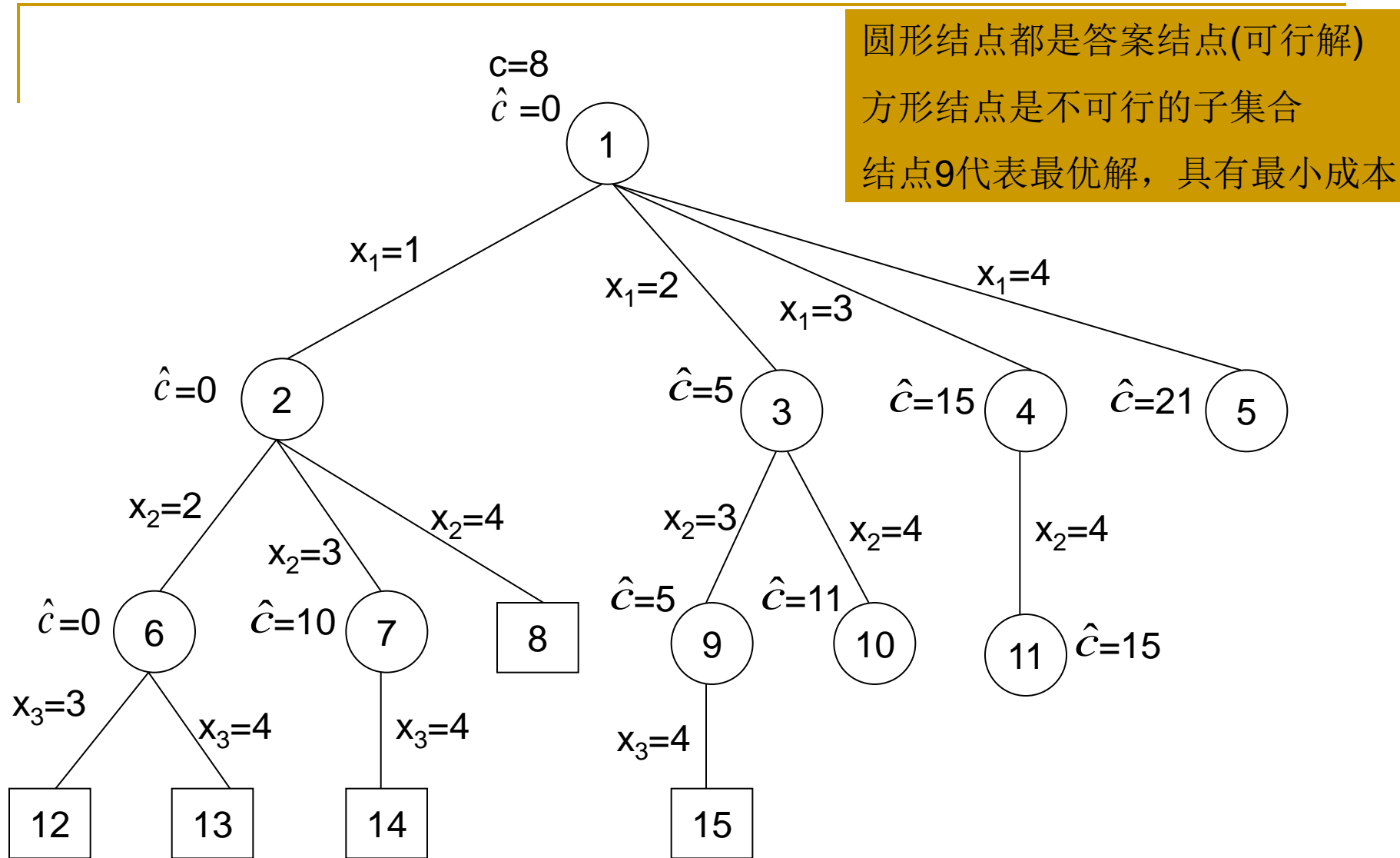


图9.7 采用大小可变的元组表示的状态空间树

只有圆形叶结点都是答案结点
 方形结点是不可行的子集合
 结点25代表最优解，具有最小成本

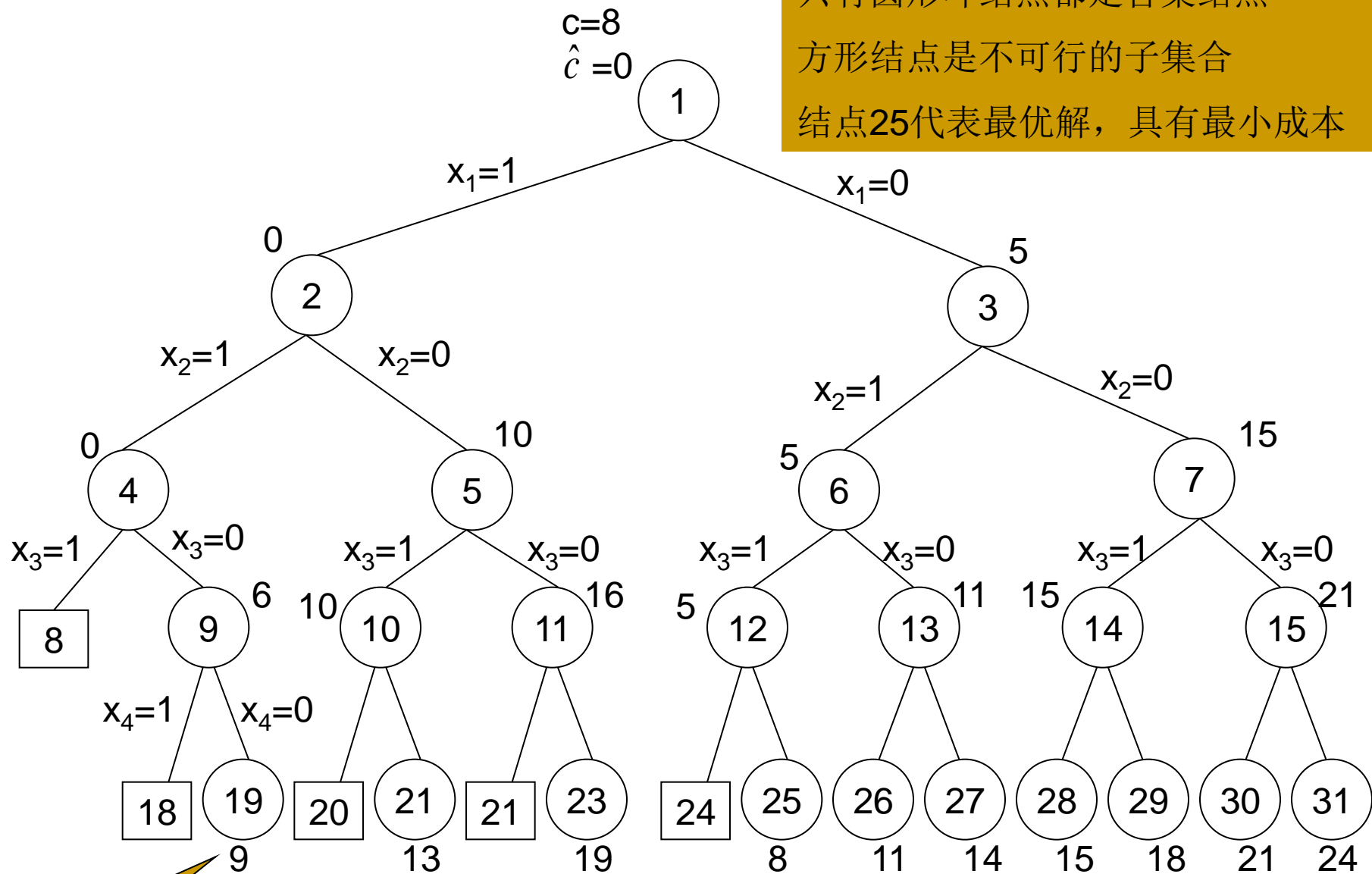


图9.8 采用大小固定的元组表示的状态空间树

罚款值

效率分析

- 上下界函数的选择是决定分枝-限界算法效率的主要因素
- 对 U 选择一个更好的初值是否能减少所生成的结点数？（否，根据定理7.4）
- 扩展一些 $c^*(\cdot) > U$ 的结点是否能减少所生成的结点数？（否，根据定理7.5）
- 假定有两个成本估计函数 $c_1^*(\cdot)$ 和 $c_2^*(\cdot)$ ，对于状态空间树的每一个结点 X ，若有 $c_1^*(\cdot) \leq c_2^*(\cdot) \leq c(X)$ ，则称 $c_2^*(\cdot)$ 比 $c_1^*(\cdot)$ 好。是否用较好的成本估计函数生成的结点数要少呢？（否，根据定理7.6和定理7.7）

0/1背包问题描述

- 极小化

$$-\sum_{i=1}^n p_i x_i$$

- 约束条件

$$\sum_{i=1}^n w_i x_i \leq M$$

- $x_i=0$ 或 $x_i=1$, $1 \leq i \leq n$

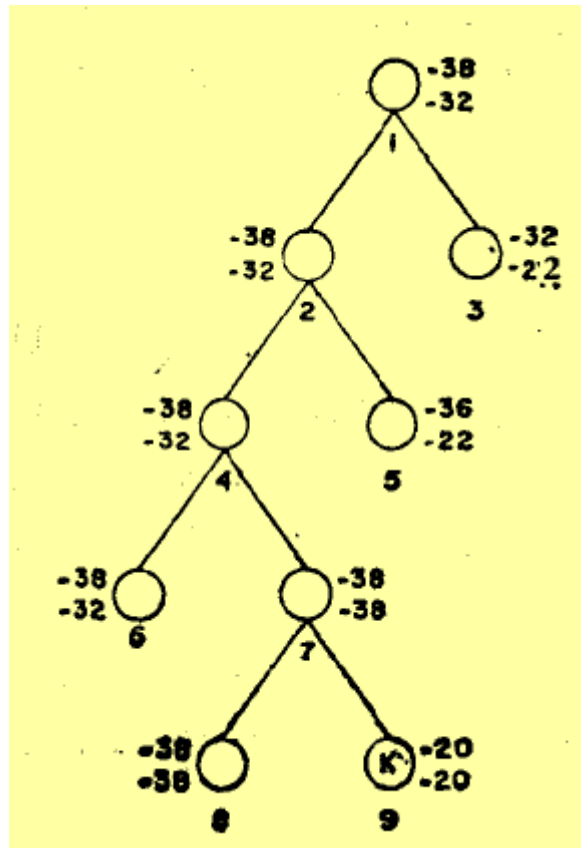
0/1背包问题的函数定义

- $c(X) = -\sum_{i=1}^n p_i x_i$ （答案结点）
- $c(X) = \infty$ （不可行的结点）
- $c(X) = \min\{c(\text{LCHILD}(X)), c(\text{RCHILD}(X))\}$
- $c^{\wedge}(X) = -\text{Bound}\left(\sum_{i=1}^{j-1} p_i x_i, \sum_{i=1}^{j-1} w_i x_i, j-1, M\right)$
- $U(X) = -\text{Bound}\left(\sum_{i=1}^{j-1} p_i x_i, \sum_{i=1}^{j-1} w_i x_i, j-1, M\right)$
 - 其中j是结点X所在的层级

例9.2

- $n=4, M=15$
- $(p_1, p_2, p_3, p_4) = (10, 10, 12, 18)$
- $(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$

例9.2的LC分枝-限界树



上面的数= c^u

下面的数= u

大小固定元组

LCBB求解背包问题分析

- 状态空间树中结点的结构
- 如何生成一给定结点的儿子
- 如何识别答案结点
- 如何表示活结点表

状态空间树中结点的结构

- PARENT
 - 父结点链接指针
- LEVEL
 - 状态空间树中的级数
- TAG
 - X_i 的取值
- CU
 - 背包的剩余空间
- PE
 - 已装入物品的效益值的和
- UB
 - $c^{\wedge}(X)$

如何生成一给定结点的儿子

■ 左儿子生成

- $PARENT(Y) = X$
- $LEVEL(Y) = LEVEL(X) + 1$
- $CU(Y) = CU(X) - W_{LEVEL(X)}$
- $PE(Y) = PE(X) + P_{LEVEL(X)}$
- $TAG = 1$
- $UB(Y) = UB(X)$

如何识别答案结点

- 当且仅当 $\text{LEVEL}(X) = n + 1$
- X 是答案结点

如何表示活结点表

- Min-堆
- 测试活结点表是否为空
 - 常量时间
- 加结点到活结点表
 - $\log(n)$
- 删除最小UB值的结点
 - $\Theta(\log(n))$



计算上界和下界的算法

```
line procedure LUBOUND(P, W, rw, cp, N, k, LBB, UBB)
1  LBB  $\leftarrow$  cp; c  $\leftarrow$  rw;
2  for i  $\leftarrow$  k to N do
3      if c < W(i) then UBB  $\leftarrow$  LBB + c * P(i) / W(i)
4          for j  $\leftarrow$  i + 1 to N do
5              if c  $\geq$  W(j) then c  $\leftarrow$  c - W(j)
6                  LBB  $\leftarrow$  LBB + P(j)
7              endif
8          repeat
9          return
10     endif
11     c  $\leftarrow$  c - W(i); LBB  $\leftarrow$  LBB + P(i)
12 repeat
13 UBB  $\leftarrow$  LBB
14 end LUBOUND
```

生成一个新结点

```
line procedure NEWNODE(par, lev, t, cap, prof, ub)
1  call GETNODE(I)
2  PARENT(I)  $\leftarrow$  par; LEVEL(i)  $\leftarrow$  lev; TAG(I)  $\leftarrow$  t
3  CU(I)  $\leftarrow$  cap; PE(I)  $\leftarrow$  prof; UB(I)  $\leftarrow$  ub
4  call ADD(I)
5  end NEWNODE
```

背包问题的LC分枝-限界算法

```
line procedure LCKNAP(P, W, M, N,  $\epsilon$ )
// 大小固定元组表示状态空间树
// 假设 $P(1)/W(1) \geq P(2)/W(2) \geq \dots \geq P(N)/W(N)$ 
  real P(N), W(N), M, L, LBB, UBB, cap, prof
  int ANS, X, N
1  call INIT
2  call GETNODE(E)
3  PARENT(E)  $\leftarrow$  0; LEVEL(e)  $\leftarrow$  1; CU(E)  $\leftarrow$  M; PE(E)  $\leftarrow$  0
4  call LUBOUND(P, W, M, N, 0, 1, LBB, UBB)

5  L  $\leftarrow$  LBB -  $\epsilon$ ; UB(E)  $\leftarrow$  UBB
6  loop
7    i  $\leftarrow$  LEVEL(E); cap  $\leftarrow$  CU(E); prof  $\leftarrow$  PE(E)
```

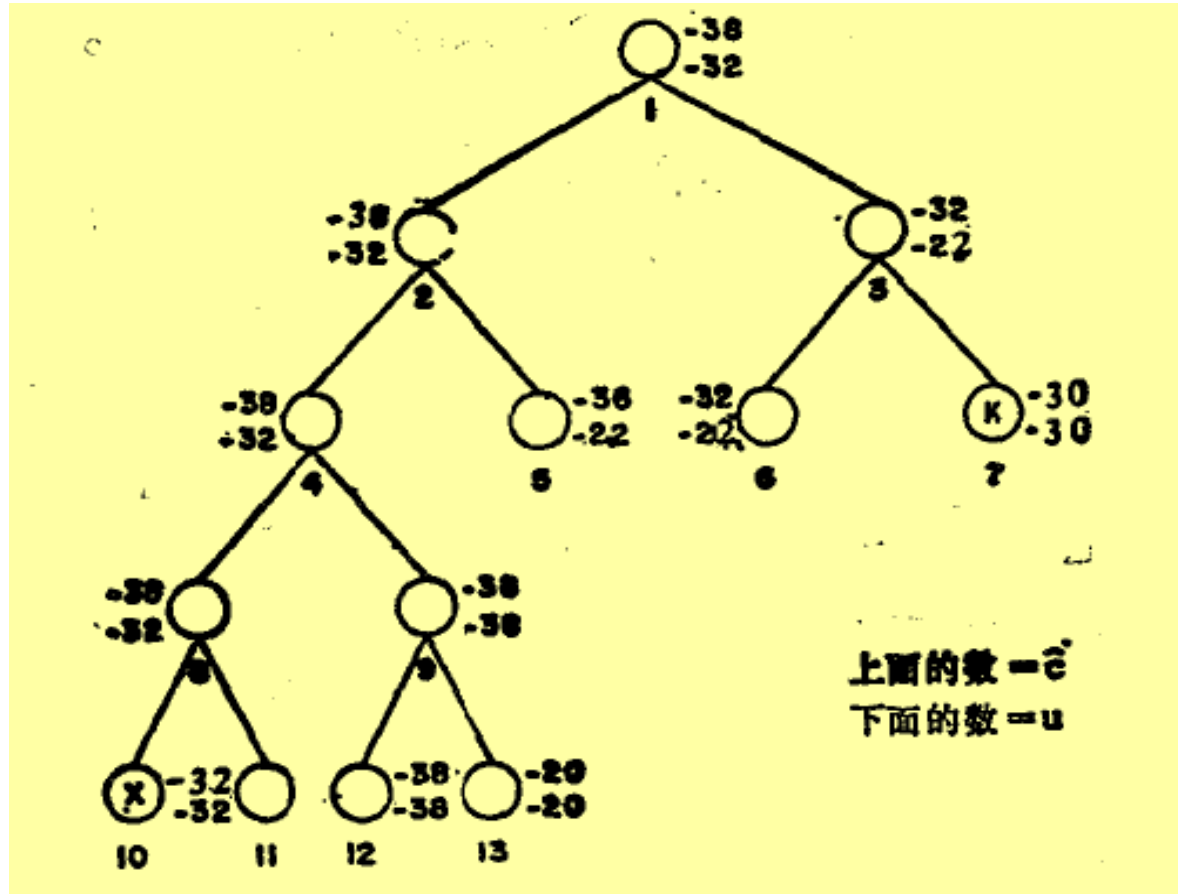
背包问题的LC分枝-限界算法

```
8      case
9      :i=N+1:
10         if prof>L then L ← prof; ANS ← E
11         endif
12      :else:
13         if cap>=W(i) then
14             call NEWNODE(E,i+1,1,cap-W(i),
15                             prof+P(1)UB(E))
16         endif
17         call LUBOUND(P,W,cap,prof,N,i+1, LBB,UBB)
18         if UBB>L then
19             call NEWNODE(E,i+1,0,cap,prof,UBB)
20             L ← max(L,LBB-  $\epsilon$ )
21         endif
22     endcase
```

背包问题的LC分枝-限界算法

```
22      if 不再有活结点 then exit endif
23      call LARGEST(E)
24      until  $UB(E) \leq L$  repeat
25      call FINISH(L,ANS,N)
26 end LCKNAP
```

FIFO分枝限界树



背包问题FIFO分枝-限界算法

```
line procedure FIFOKNAP(P, W, M, N)
// 大小固定元组表示状态空间树
// 假设 $P(1)/W(1) \geq P(2)/W(2) \geq \dots \geq P(N)/W(N)$ 
    real P(N), W(N), M, L, LBB, UBB, E, cap, prof
    int ANS, X, N
1   call INIT;  $i \leftarrow 1$ 
2   call LUBOUND(P,W,M,0,N,1,L,UBB)
3   call NNODE(0,0,M,0,UBB)
4   call ADDQ('#')
5   while  $i \leq N$  do
6       loop
7       call DELETEQ(E)
```

背包问题FIFO分枝-限界算法

```
8      case
9        :E='#': exit
10       :UB(E)>=L:
11         cap  $\leftarrow$  CU(E); prof  $\leftarrow$  PE(E)
12         if cap  $\geq$  W(i) then
13           call NNODE(E,1,cap-W(i), prof+P(i),UB(E))
14         endif
15         call LUBOUND(P,W,cap,prof,N,i+1, LBB,UBB)
16         if UBB  $\geq$  L then
17           call NNODE(E,0,cap,prof,UBB)
18           L  $\leftarrow$  max(L,LBB)
19         endif
20       endcase
21     repeat
22       call ADDQ('#')
23       i  $\leftarrow$  i + 1
24     repeat
25     ANS  $\leftarrow$  PE(X)=L的活结点X
26     call FINISH(L,ANS,N)
27   end FIFOKNAP
```