

# 算法设计与分析

Computer Algorithm Design & Analysis  
2020.10



群名称: 算法设计与分析2020  
群 号: 271069522

吕志鹏

[zhipeng.lv@hust.edu.cn](mailto:zhipeng.lv@hust.edu.cn)

群名称: 算法设计与分析  
群 号: 271069522



# 分支－限界法

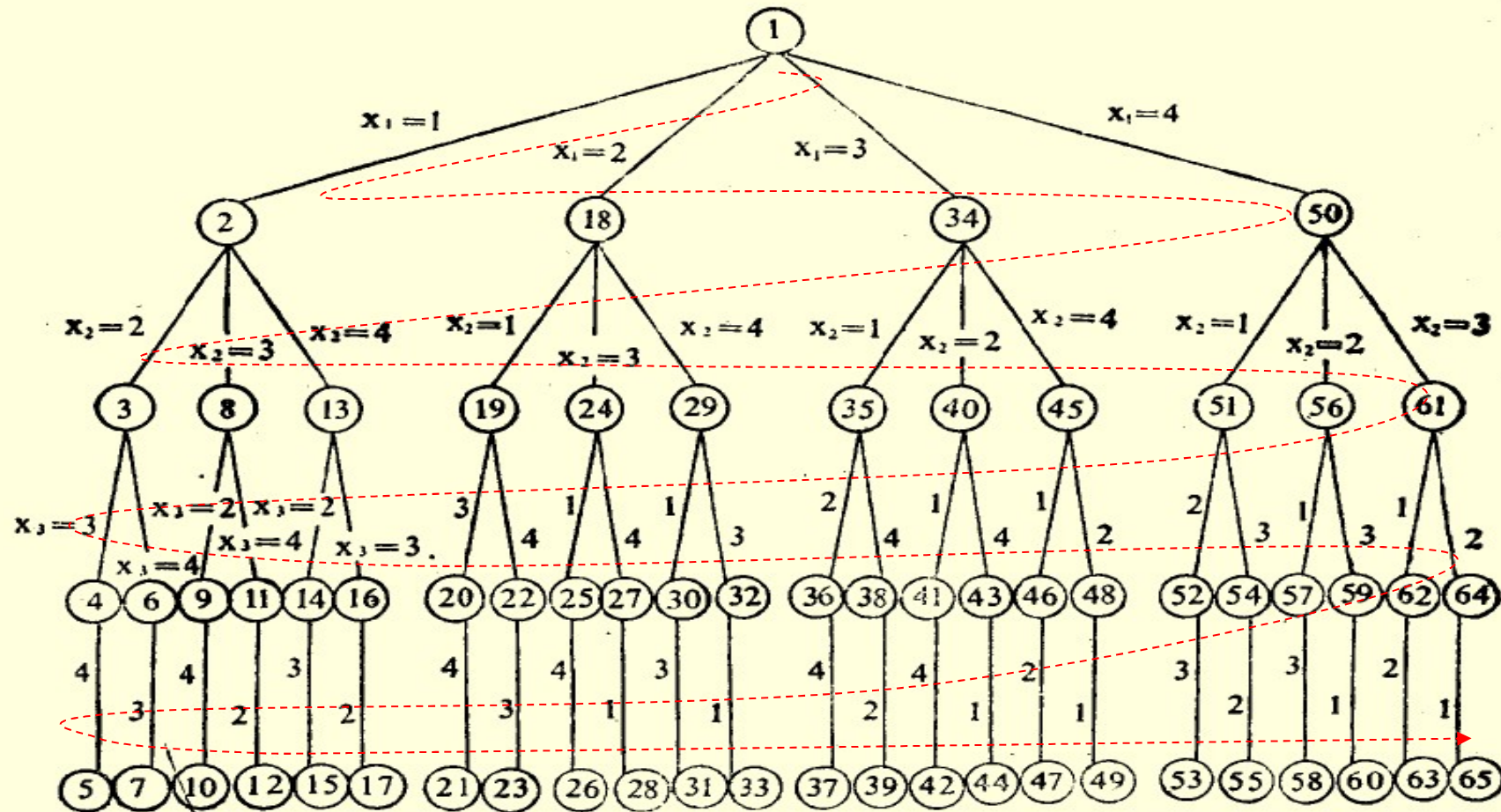
Branch and Bound



# 分支—限界法

- **分支—限界法**：采用宽度优先策略，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子树的状态空间的检索方法。
- **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）
- **分支—限界法的两种基本设计策略**：
  - FIFO检索：活结点表采用队列
  - LIFO检索：活结点表采用栈

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

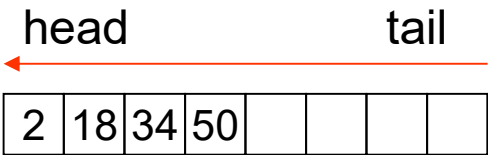
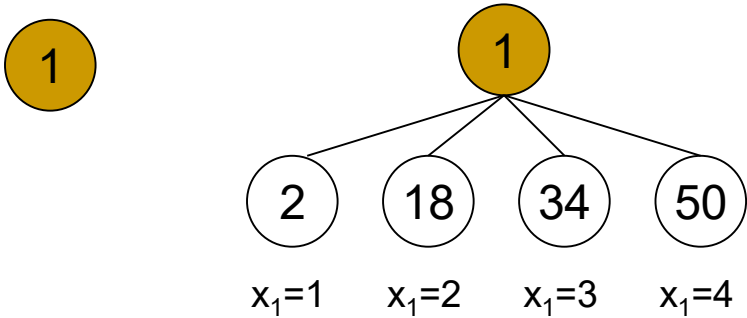


# 采用FIFO分支-限界法检索4-皇后问题的状态空间树：

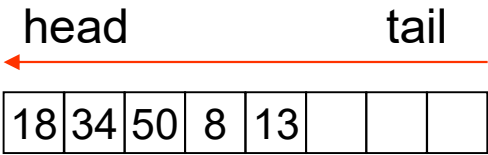
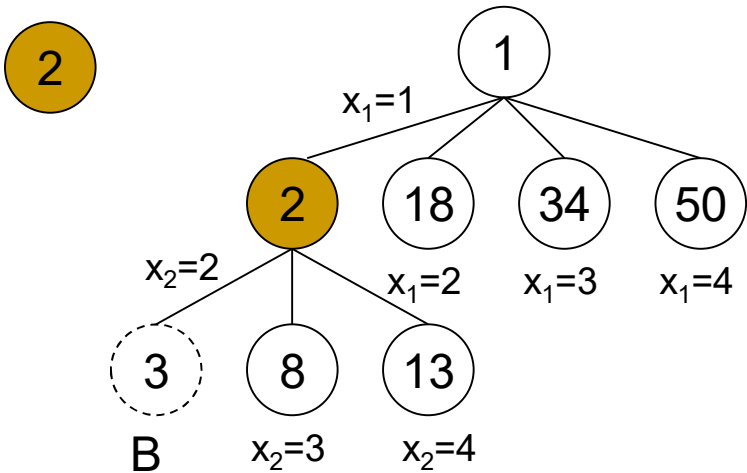
E结点

扩展E结点得到的状态空间树

活结点表（队列）



扩展结点1，得新结点2，18，34，50  
活结点2、18、34、50入队列



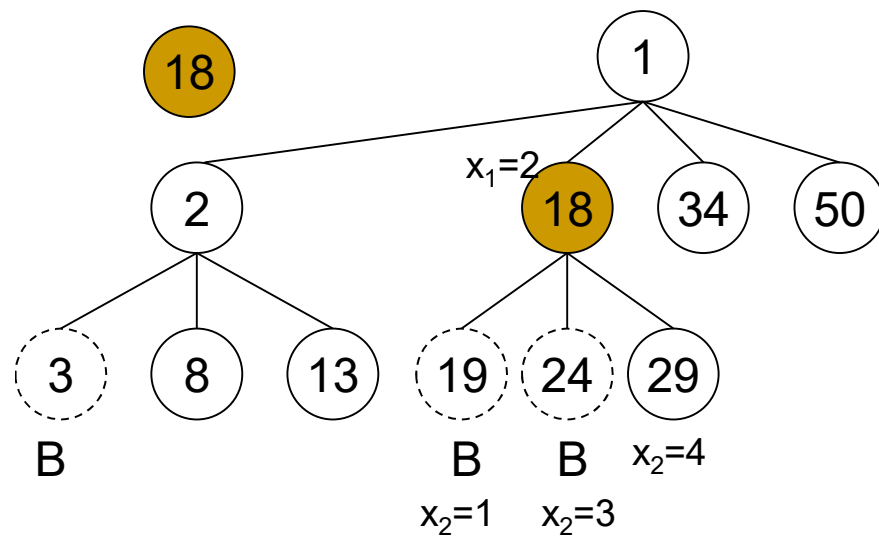
扩展结点2，得新结点3，8，13  
**利用限界函数杀死结点3**  
活结点8、13入队列



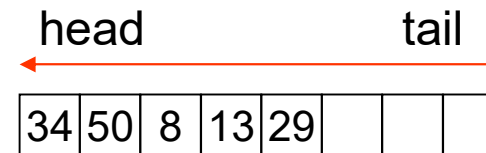
## 采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E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



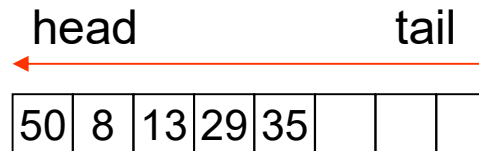
## 采用FIFO分支—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

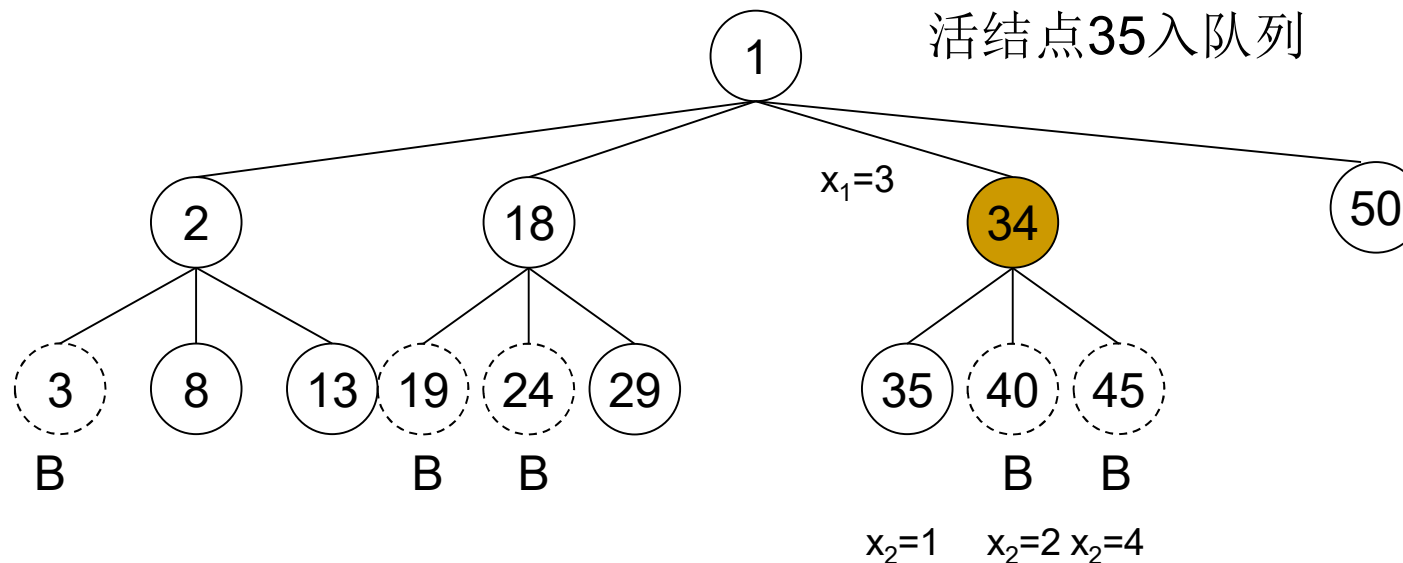
34



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分支—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

50

head

tail

8

13

29

35

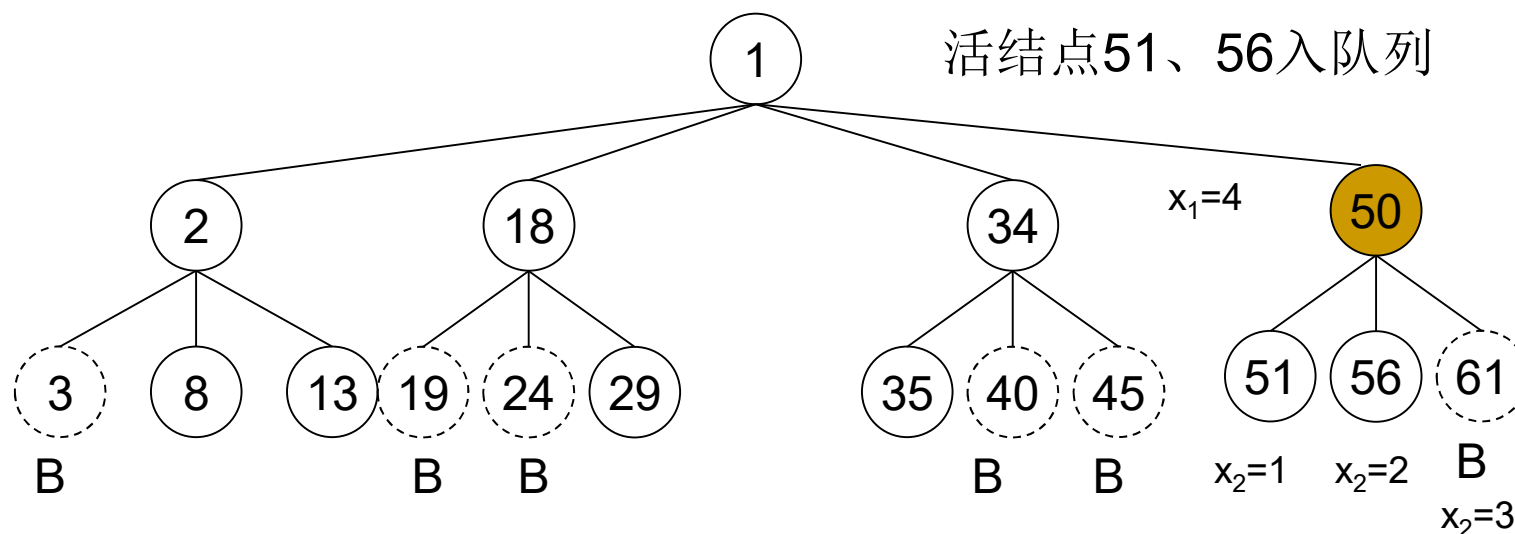
51

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







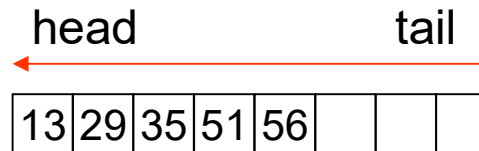
# 采用FIFO分支—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

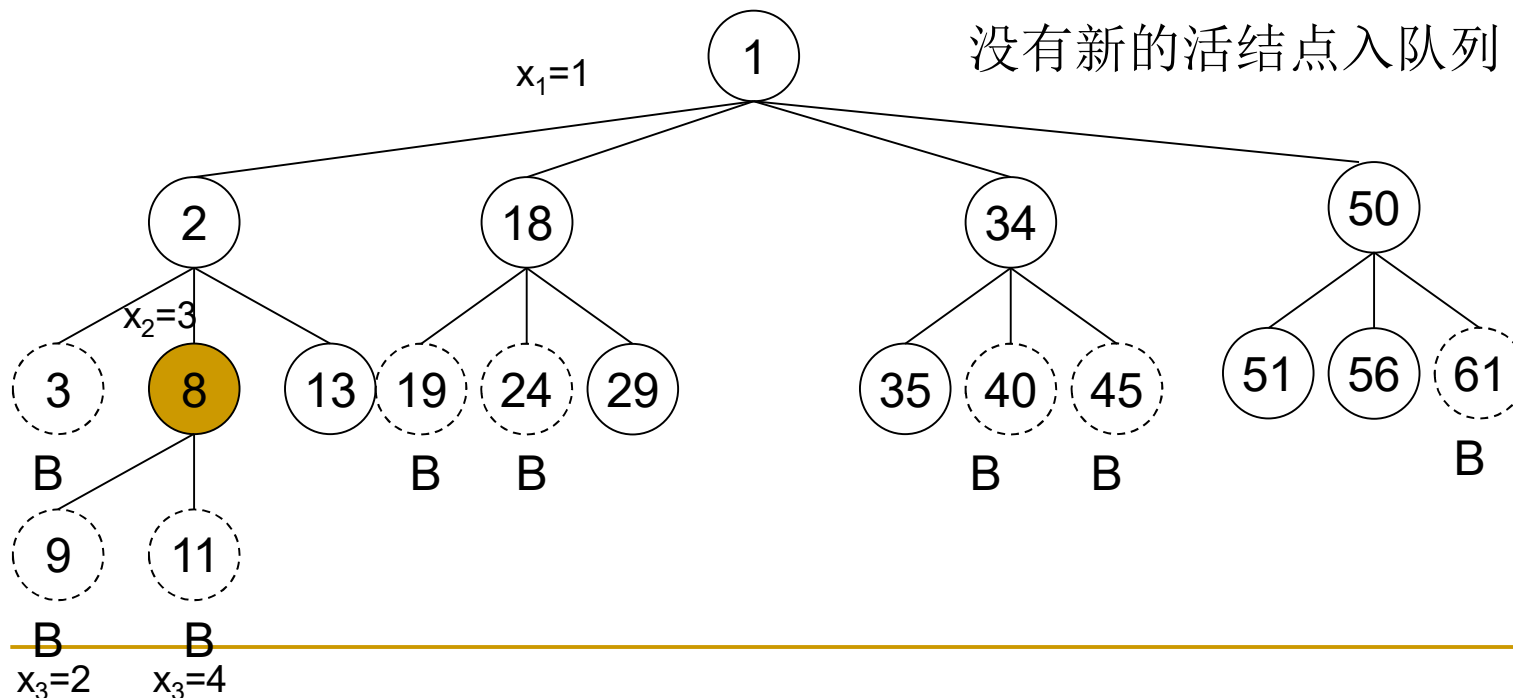
8



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分支—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

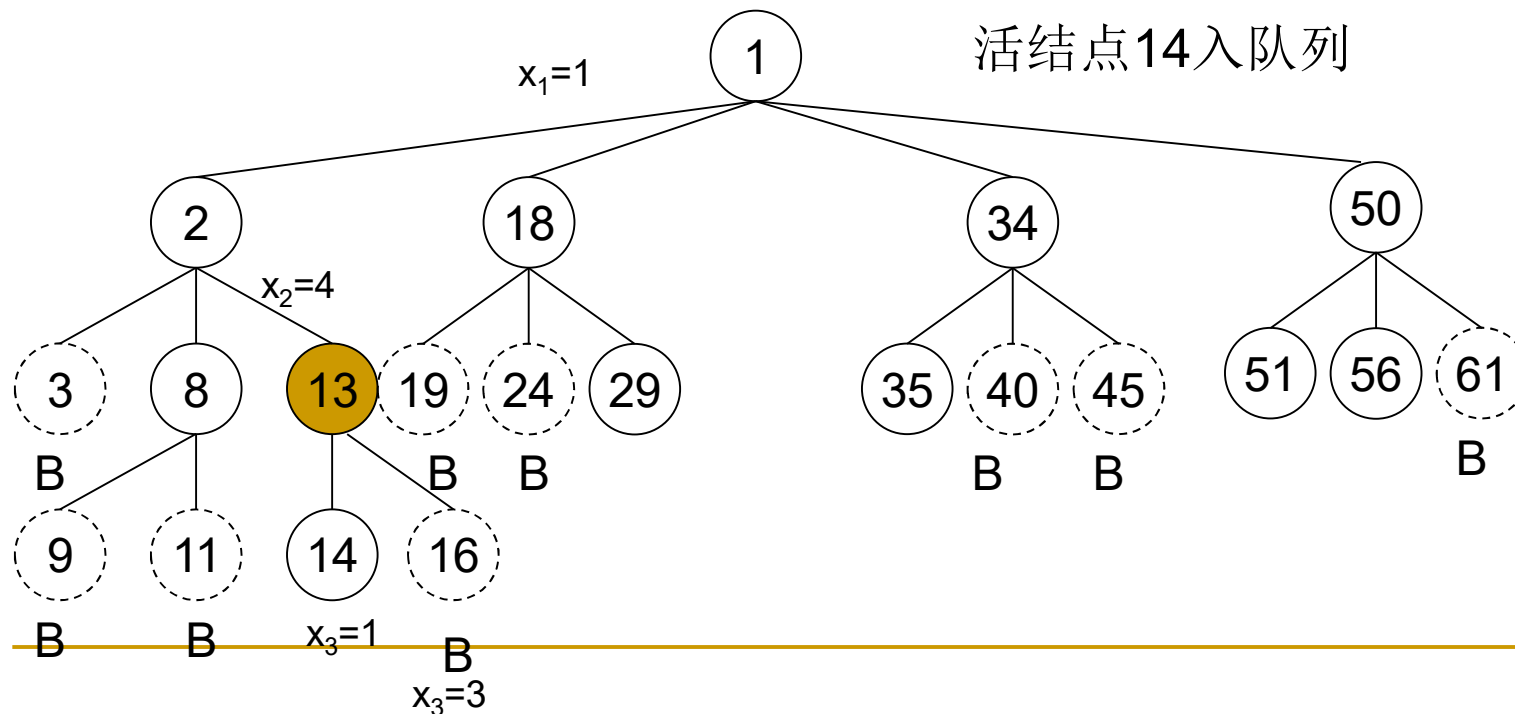
13

head		tail					
←							
29	35	51	56	14			

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





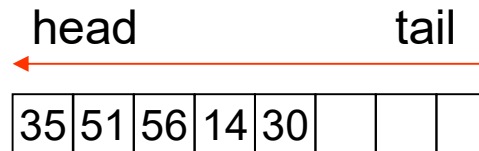
# 采用FIFO分支—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

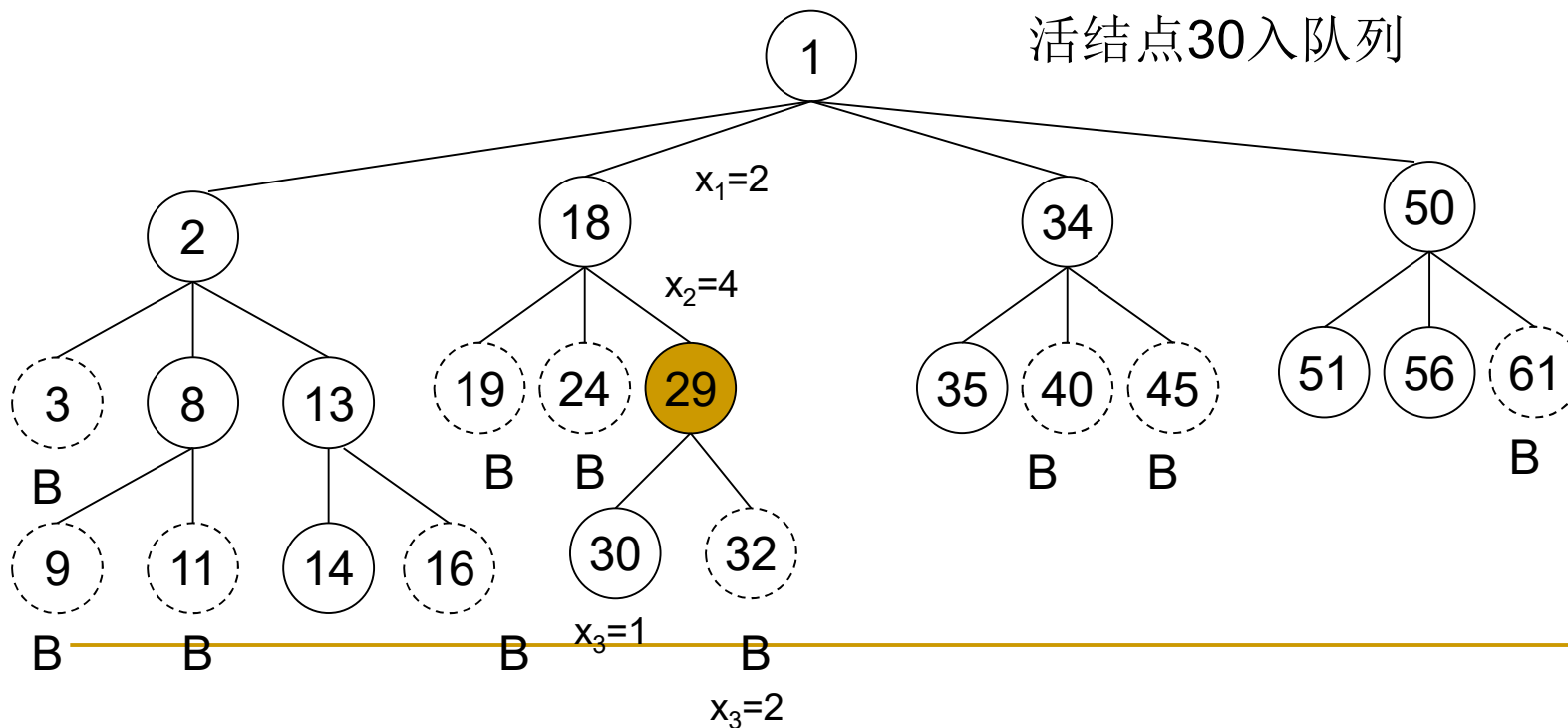
29



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





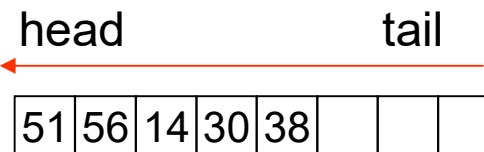
## 采用FIFO分支—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

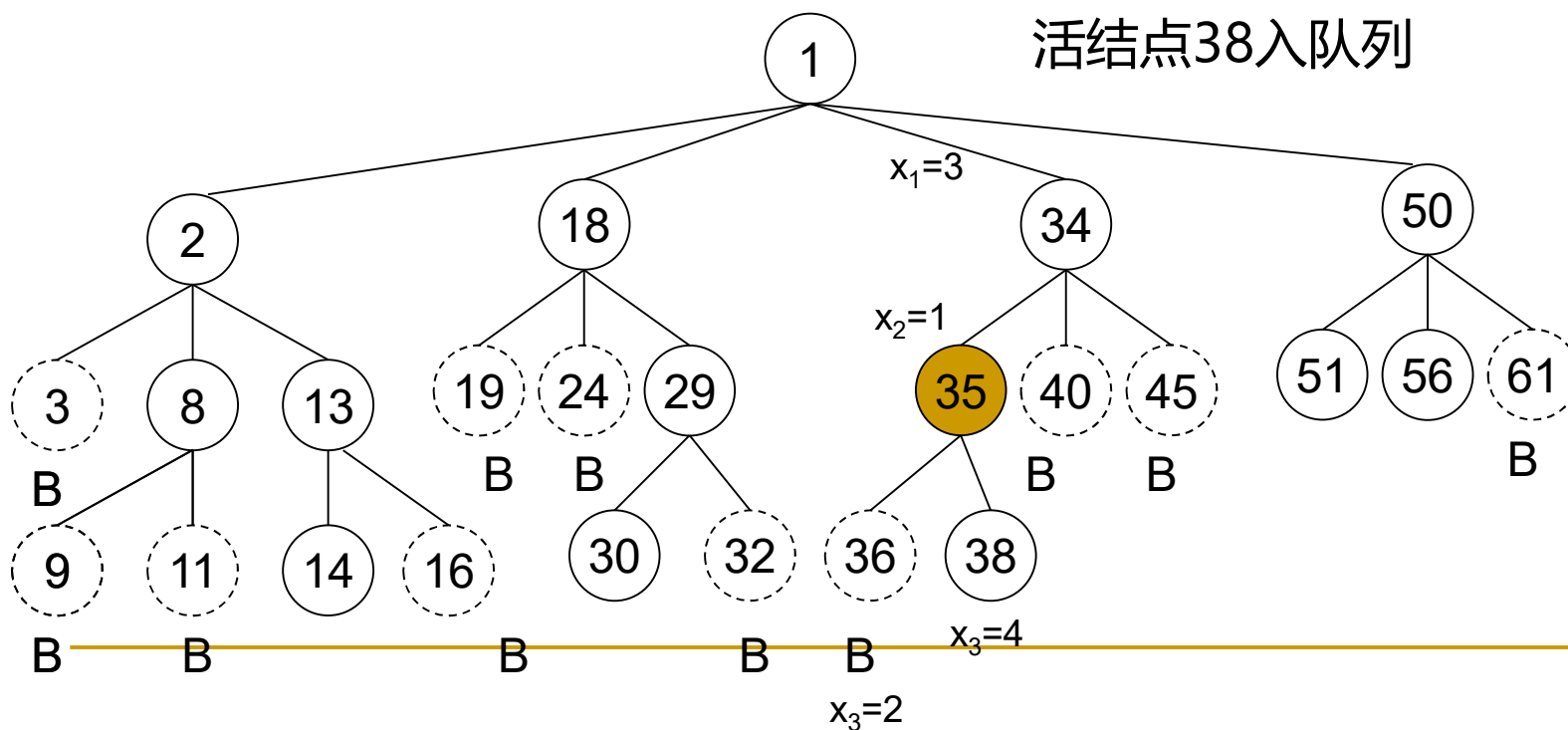
35



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





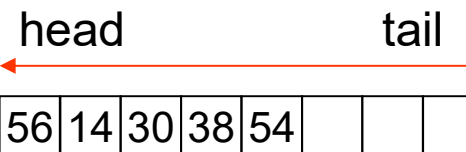
# 采用FIFO分枝—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

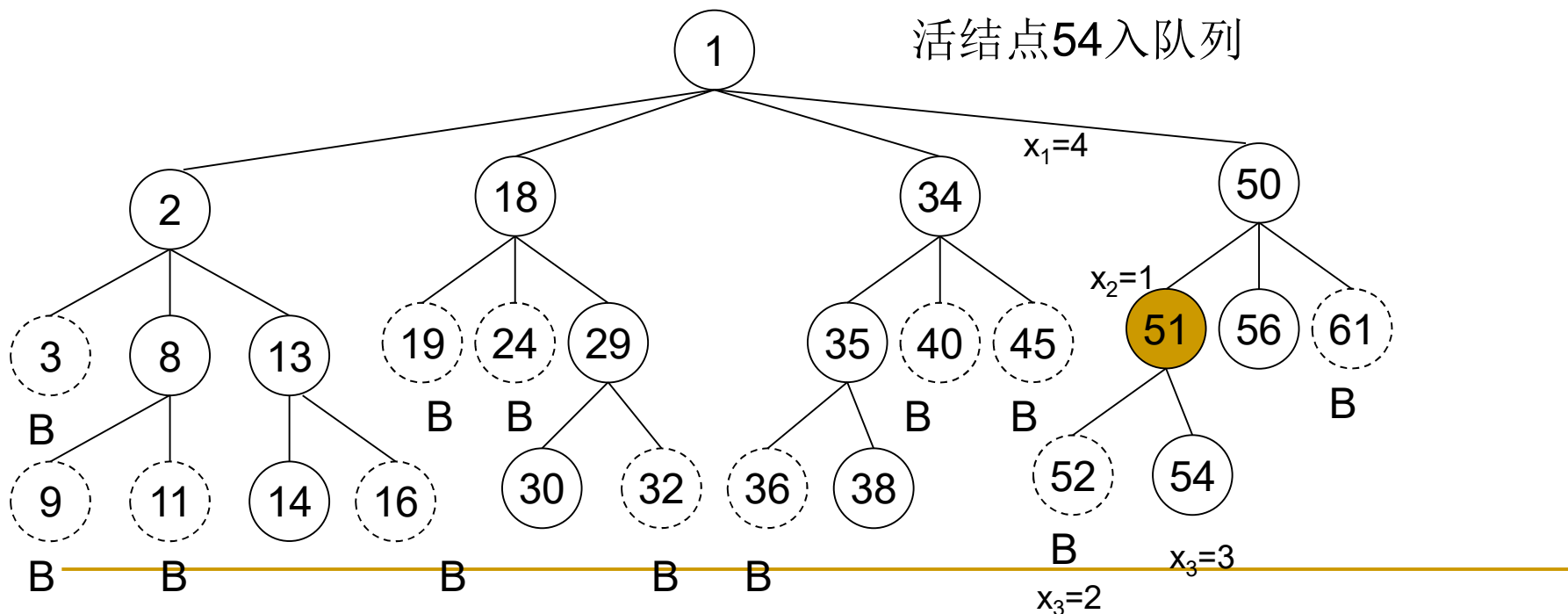
51



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





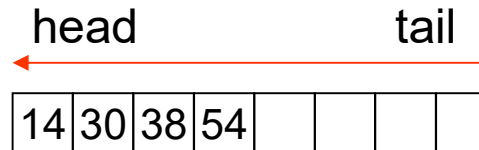
# 采用FIFO分支—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

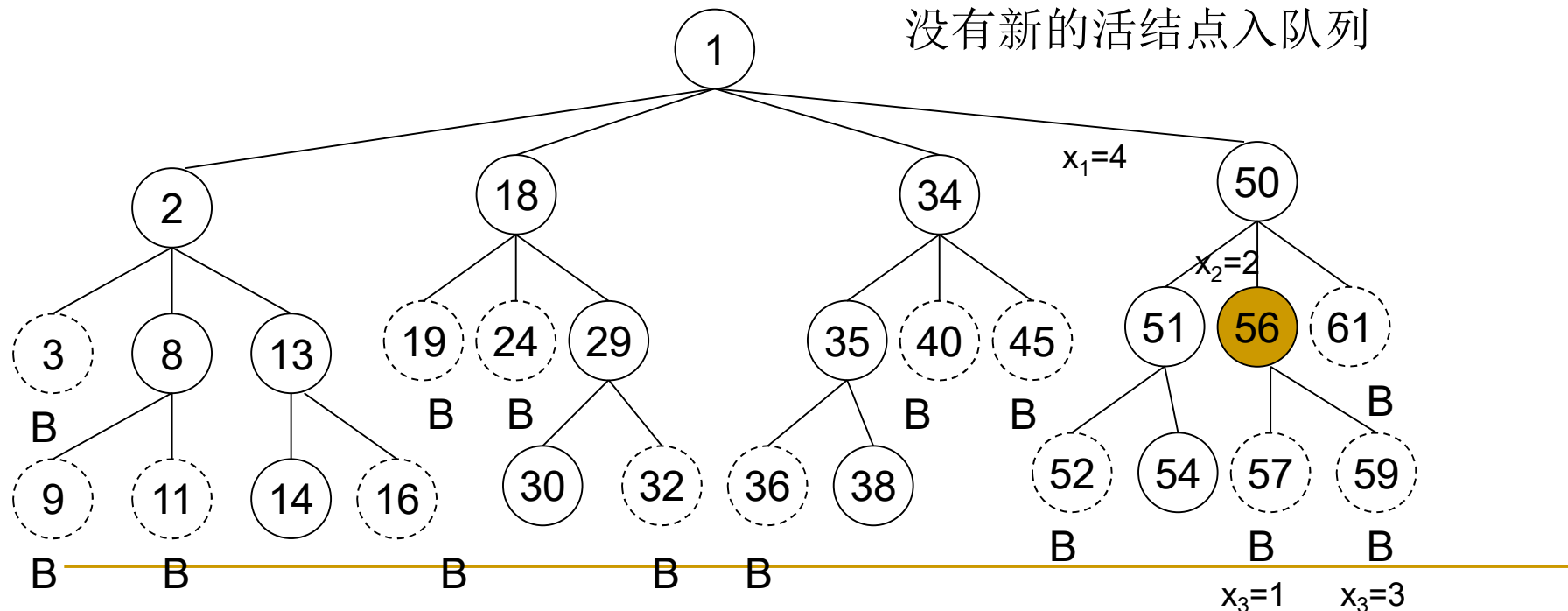
56



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





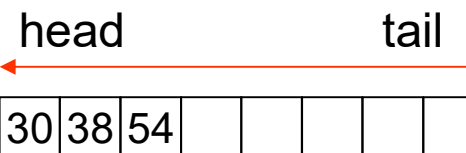
# 采用FIFO分支—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

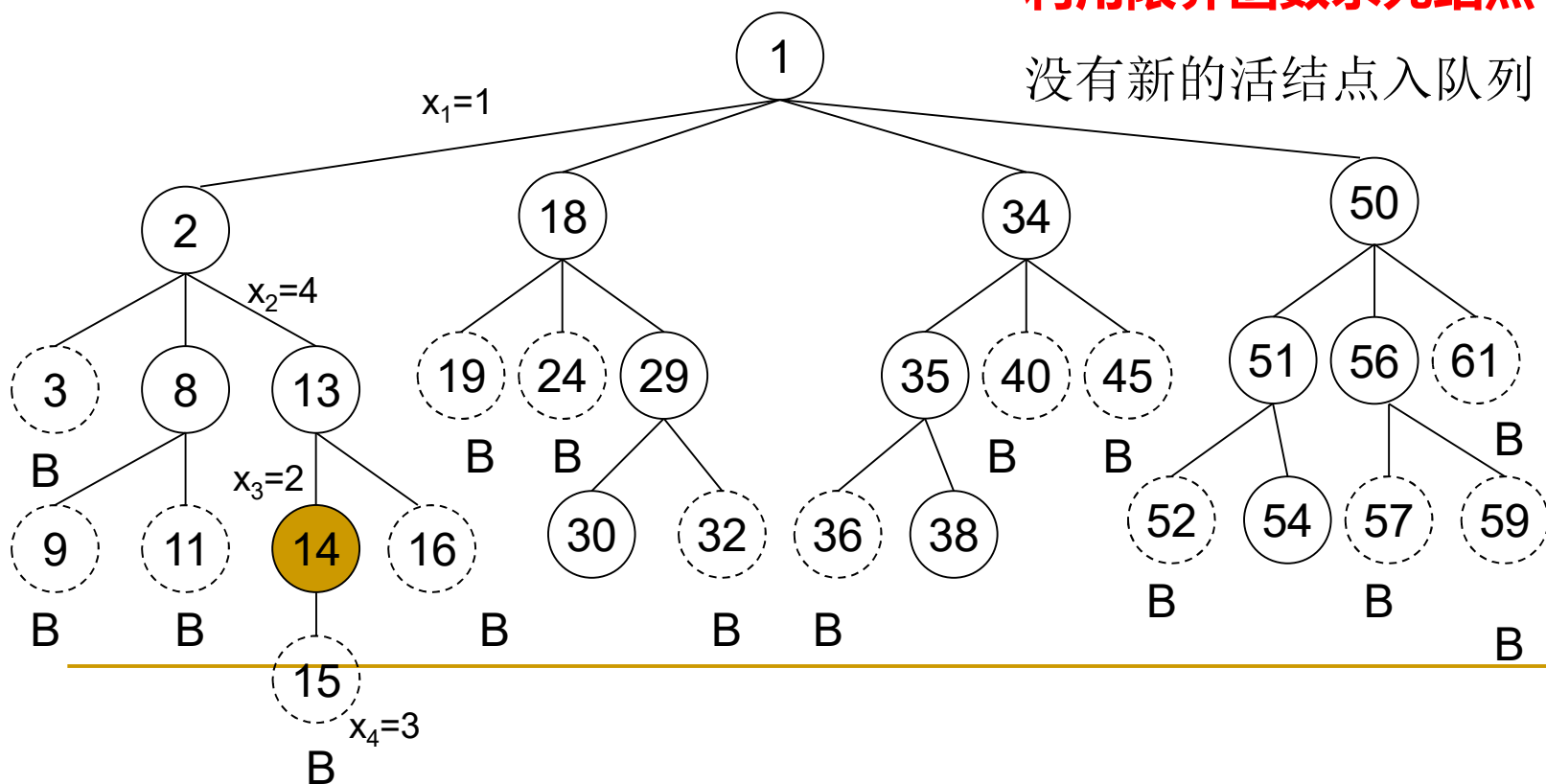
14



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分支—限界法检索4-皇后问题的状态空间树（续）：

E结点

扩展E结点得到的状态空间树

活结点表（队列）

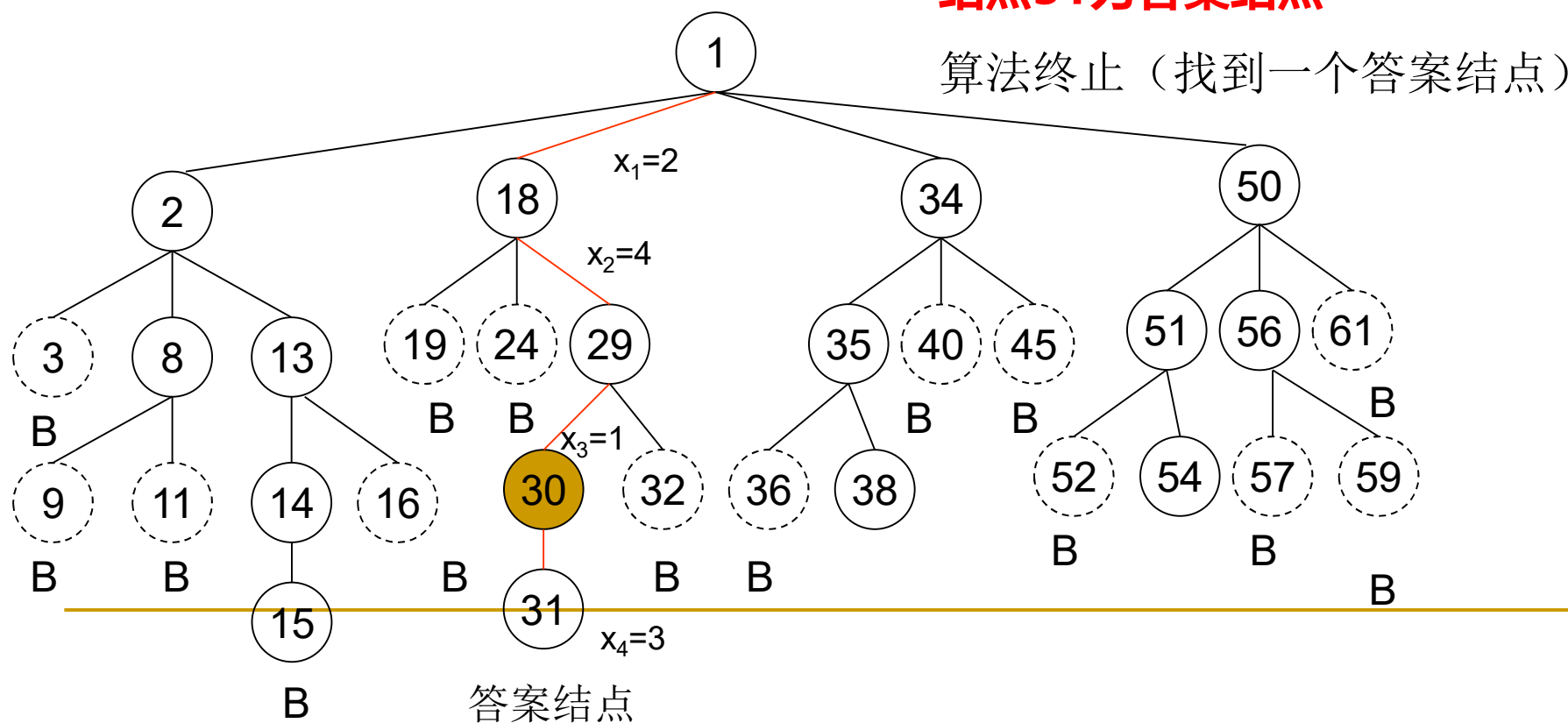
30



扩展结点30，得新结点31

**结点31为答案结点**

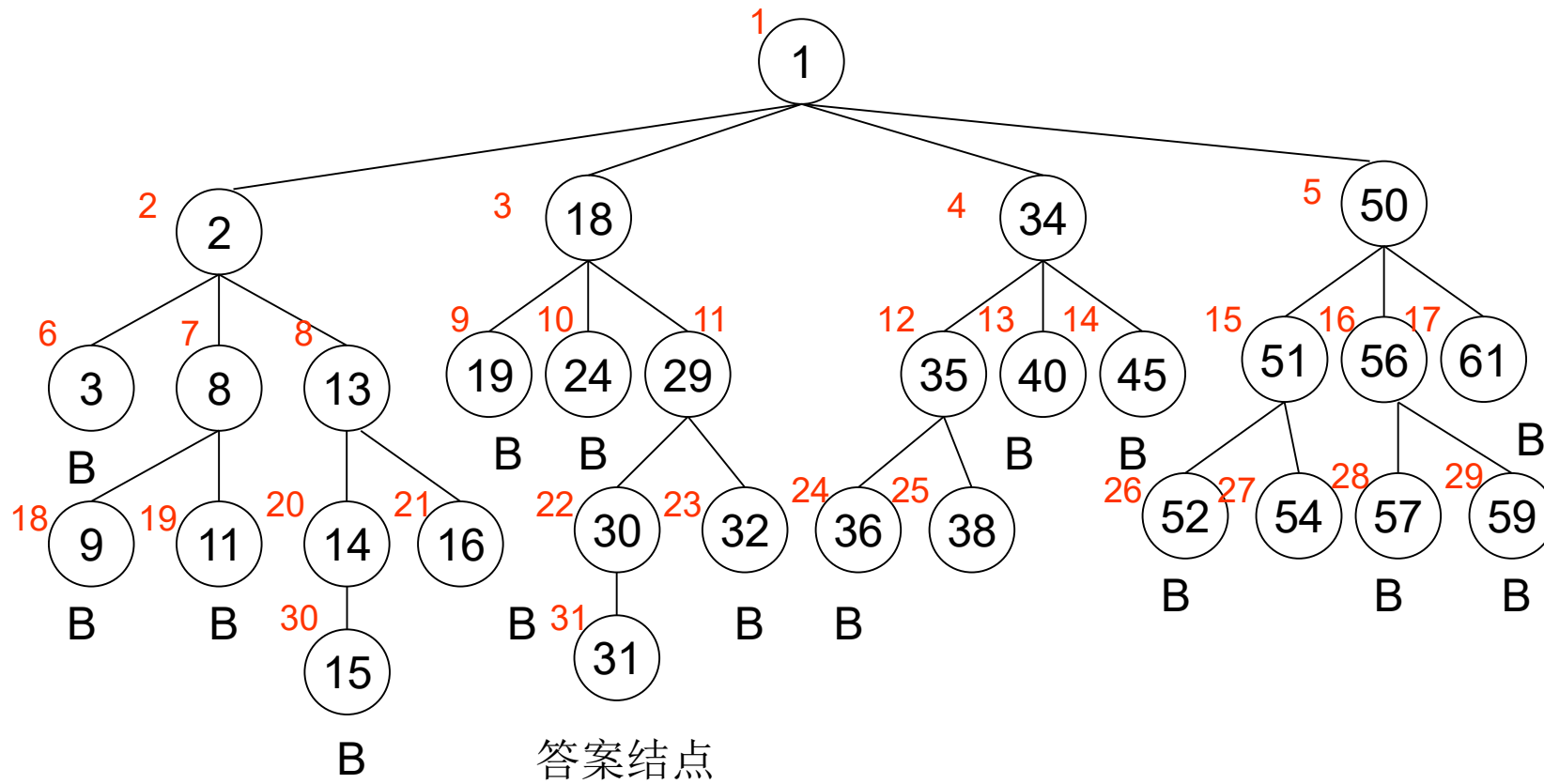
算法终止（找到一个答案结点）







# 采用FIFO分支—限界法检索4-皇后问题的状态空间树：

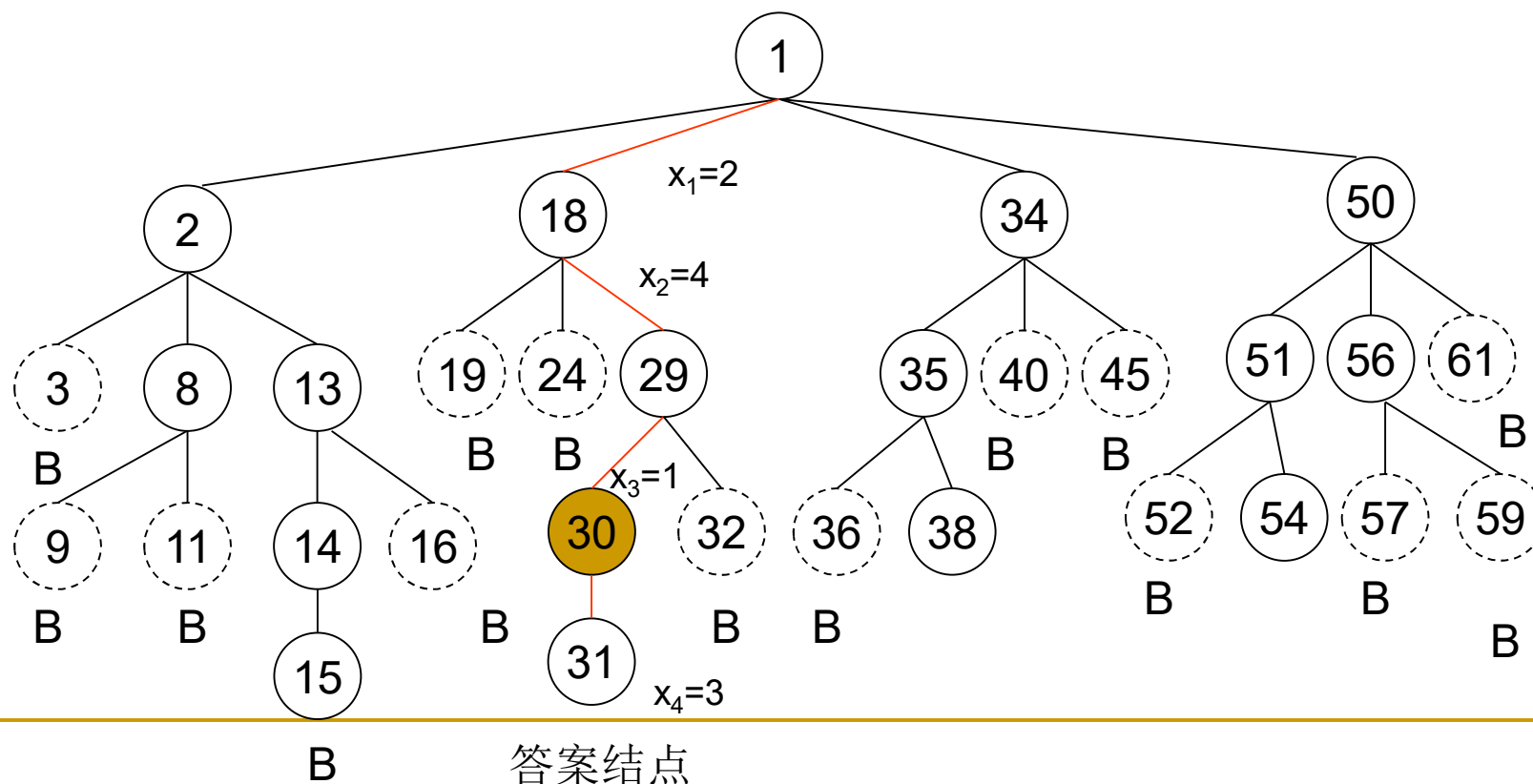




# LC-检索 (Least Cost, A\*算法)

## ■ 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。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。





# LC-检索 (Least Cost, A\*算法)

## ■ 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。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？

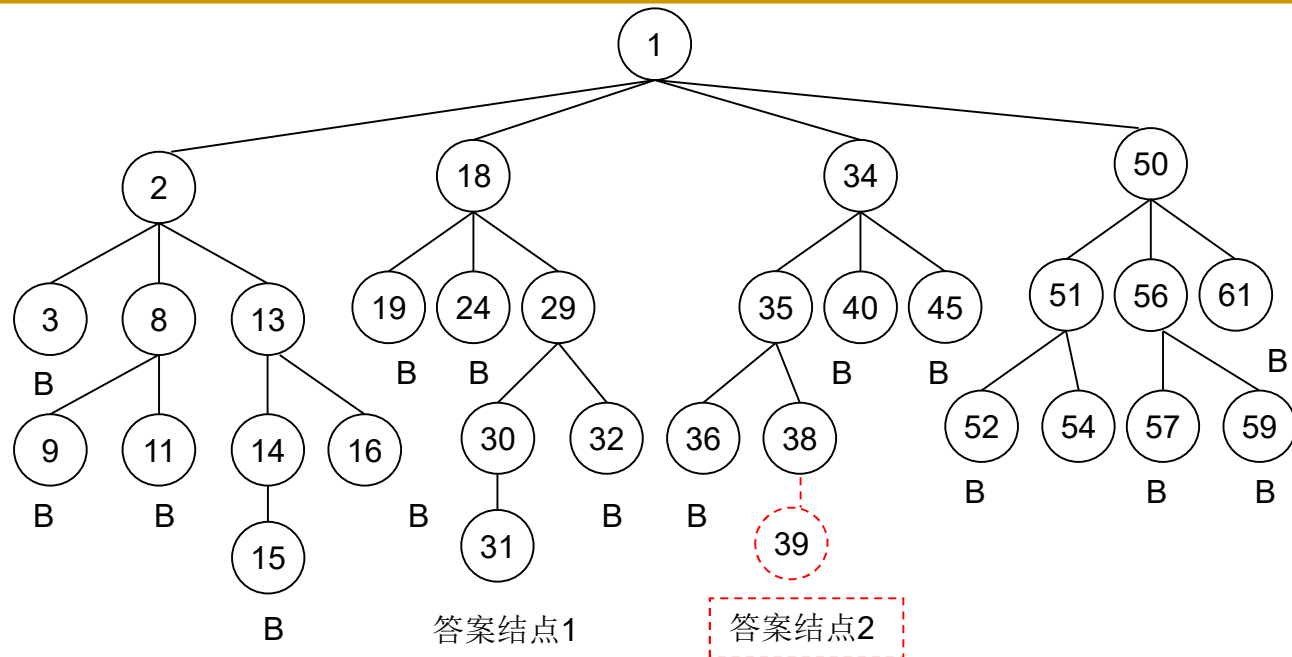
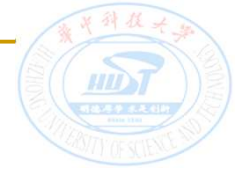
- 做某种排序，让可以导致答案结点的活结点排在前面！
- 新问题：怎么排序？
- 寻找一种“有智力”的排序函数 $C(\cdot)$ ，用 $C(\cdot)$ 来选取下一个E结点，加快到达一答案结点的检索速度。

如结点30， $29 \rightarrow 30 \rightarrow 31$



## ■ 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。



例：在量度2)下各结点的代价：

结点	代价
□ 1	4
□ 18, 34	3
□ 29, 35	2
□ 30, 38	1
□ 其余结点(除31、39)	$\geq 3, 2, 1$

量度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E结点进行扩展。

$C(\cdot)$ 又称为“**结点成本函数**”

- 结点成本函数 $C(X)$ 的取值:

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 $C(X)$ 应等于子树 $X$ 中具有最小成本的答案结点的成本



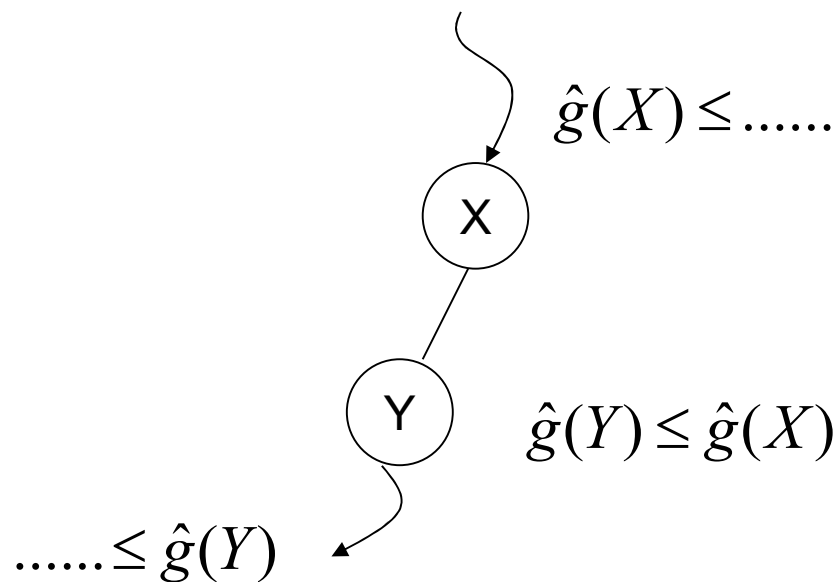
# 计算结点成本函数的困难

- 计算结点 $X$ 的代价通常要检索子树 $X$ 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。怎么办？
- 结点成本的估计函数 $\hat{c}(X)$   
包括两部分： $\hat{g}(X)$ 和  $h(X)$



$\hat{g}(X)$  : 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：

- 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: Z比W更接近答案结点,  
但  $\hat{g}(W) < \hat{g}(Z)$  。





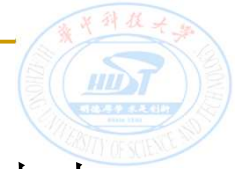
如何避免单纯考虑  $\hat{g}(X)$  造成的纵深检查？

- 引进 **$h(X)$** 改进成本估计函数。
- $h(X)$ : 根结点到结点X的成本——已发生成本。

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-结点的状态空间树检索方法。

(Least Cost Search)

特例：

■ BFS： 依据级数来生成结点， 令

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

■ 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分支-限界检索：** 带有**限界函数**的LC-检索



## LC-检索的抽象化控制

设：  $T$  是一棵状态空间树

- ◆  $c(X)$  是  $T$  的结点成本函数
- ◆  $\hat{c}(X)$  是成本估计函数
- ◆ 如果  $X$  是一个答案结点或者是一个叶结点，则
$$c(X) = \hat{c}(X)。$$

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 $E$ , 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(E):** 在活结点表中找一个具有最小成本估计值的活结点，从活结点表中删除这个结点，并将此结点放在变量 **E** 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（优先队列）存放。



# LC-检索与FIFO-检索和D-检索的关系

## 1) FIFO-检索及D-检索是LC算法的特例:

- 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，并**依据级数**来生成结点，即令 $\hat{g}(X)=0$ ； $f(h(X))=X$ 的级数，则LC-检索就变成了 FIFO-检索。
- 若活节点表采用**栈**，用LEAST(X)和ADD(X)从栈中删除或加入元素，并**令 $f(h(X))=0$** ；而当Y是X的一个儿子时，总有 $\hat{g}(X) \geq \hat{g}(Y)$ ，则LC就变成了 D-检索

## 2) 算法的不同之处在于：对下一个E-结点的选择规则不同。

---



# 不同估算函数对于结果的影响

估计函数选择不同，对寻路结果有哪些影响呢？

- 1、**当估算的距离  $\hat{g}(X)$  完全等于实际距离时**，也就是每次扩展的那个点都准确的知道选它以后，路径距离是多少，这样就不用乱选了，每次都选最小的那个，一路下去，肯定就是最优的解，而且基本不用扩展其它的点。
- 2、**如果估算距离  $\hat{g}(X)$  小于实际距离时**，则到最后一定能找到一条最短路径，但是有可能会经过很多无效的点。
- 3、**如果估算距离  $\hat{g}(X)$  大于实际距离时**，有可能就很快找到一条通往目的地的路径，但是却不一定是最优的解。



# 成本函数在分支-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

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

## 2) 最小成本的上界

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：

作用：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  $c(X) \geq \hat{c}(X) \geq U$ ，不可能具有更小的成本。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。



## 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 $\infty$
- 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就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极值的最优解。如何把求最优解的过程表示成与分支-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 $c(X)$ ，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。



# 实例：利用求解最优化问题的分支-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

假定有 $n$ 个作业和一台处理机，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；

$d_i$ ：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集(1,2,3,4)的所有可能的子集组成, 共有 $2^4$ 个元组。

状态空间树: 两种表示形式,

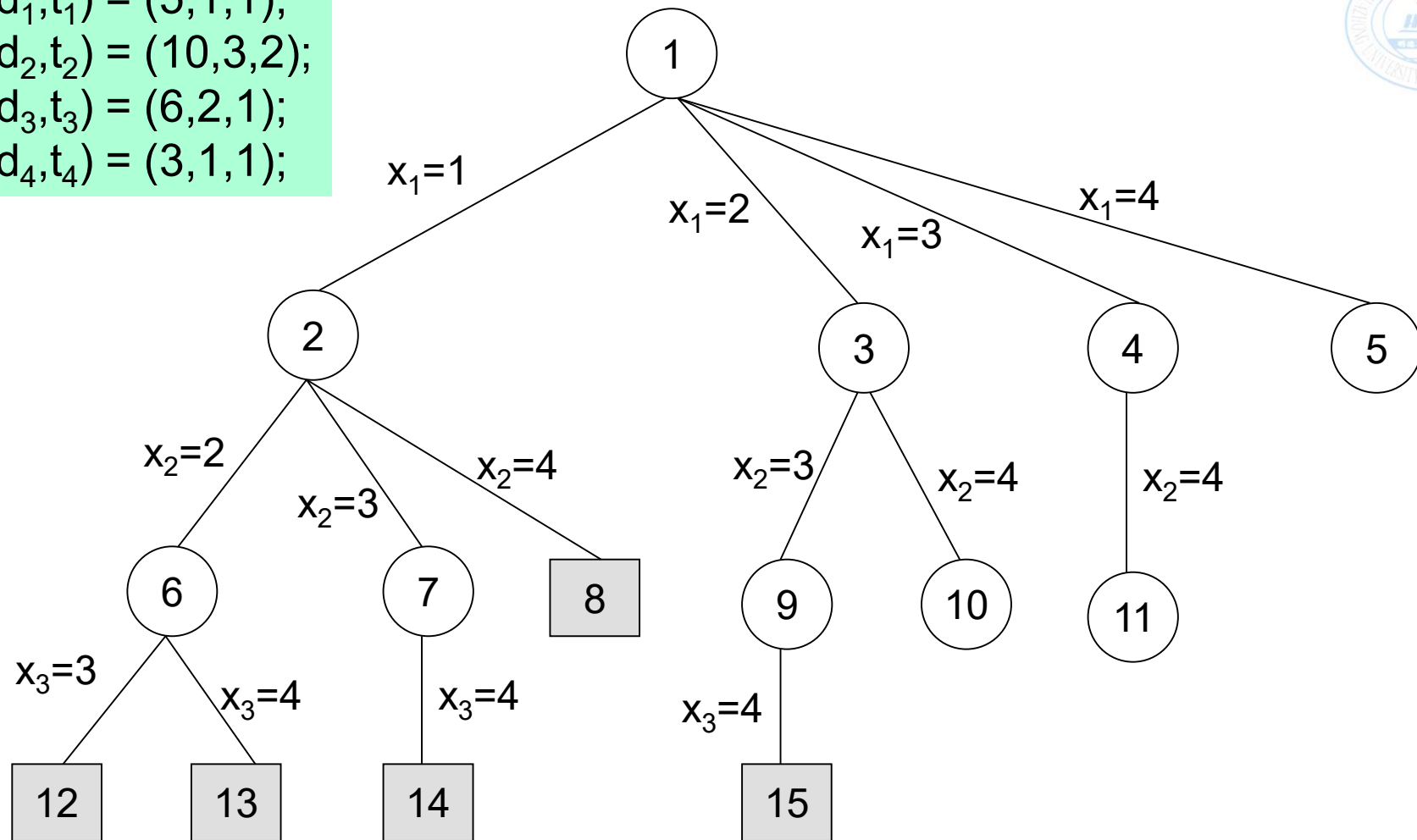
1) 元组大小可变的表示

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

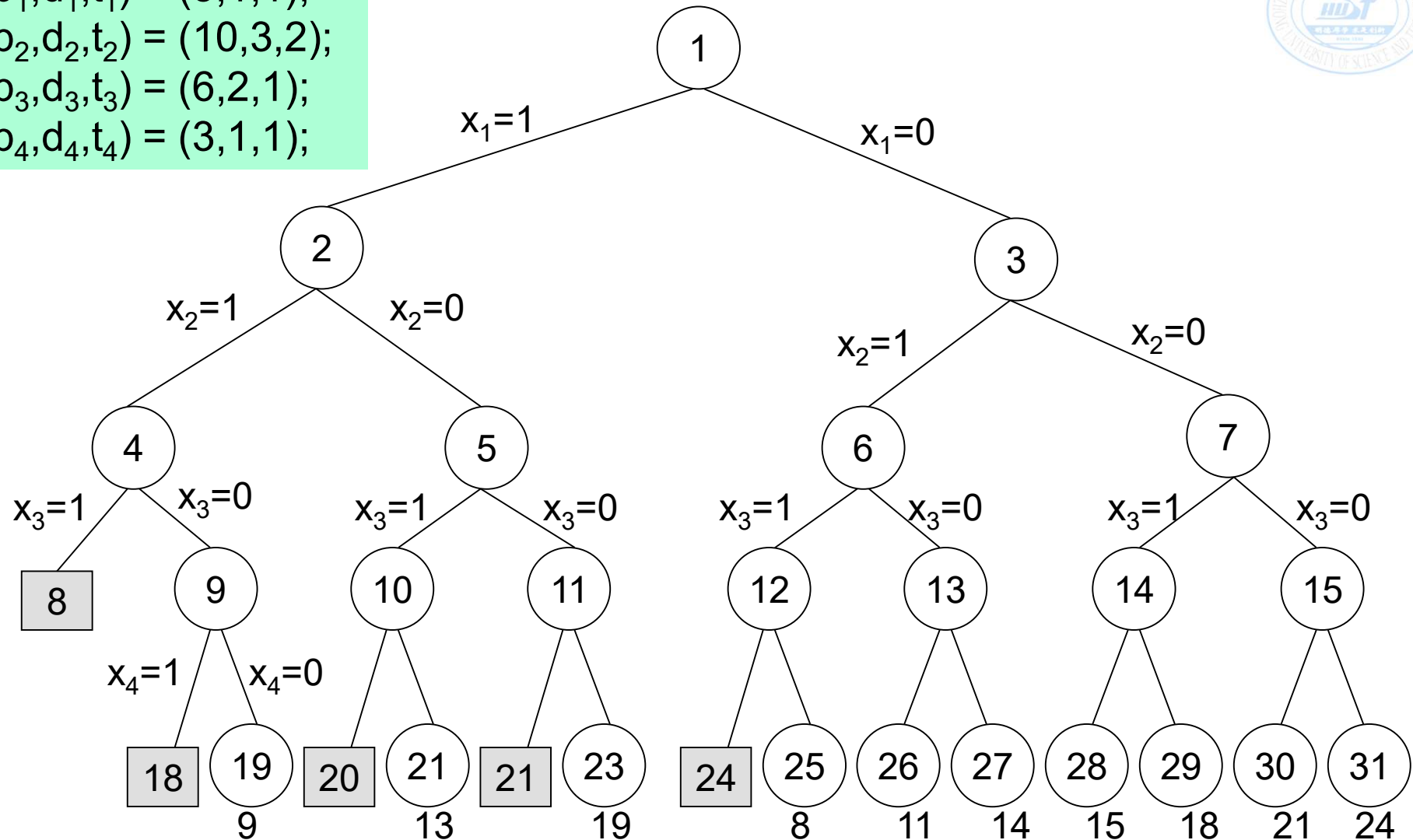
$(p_1, d_1, t_1) = (5, 1, 1);$   
 $(p_2, d_2, t_2) = (10, 3, 2);$   
 $(p_3, d_3, t_3) = (6, 2, 1);$   
 $(p_4, d_4, t_4) = (3, 1, 1);$



图(a) 采用大小可变的元组表示的状态空间树

圆形结点都是可行结点，方形结点是不可行的子集合

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



图(b) 采用大小固定的元组表示的状态空间树

只有圆形叶结点是答案结点，方形结点是不可行的子集合





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: 下图(c),  $c(3)=8$ ,  $c(2)=9$ ,  $c(1)=8$

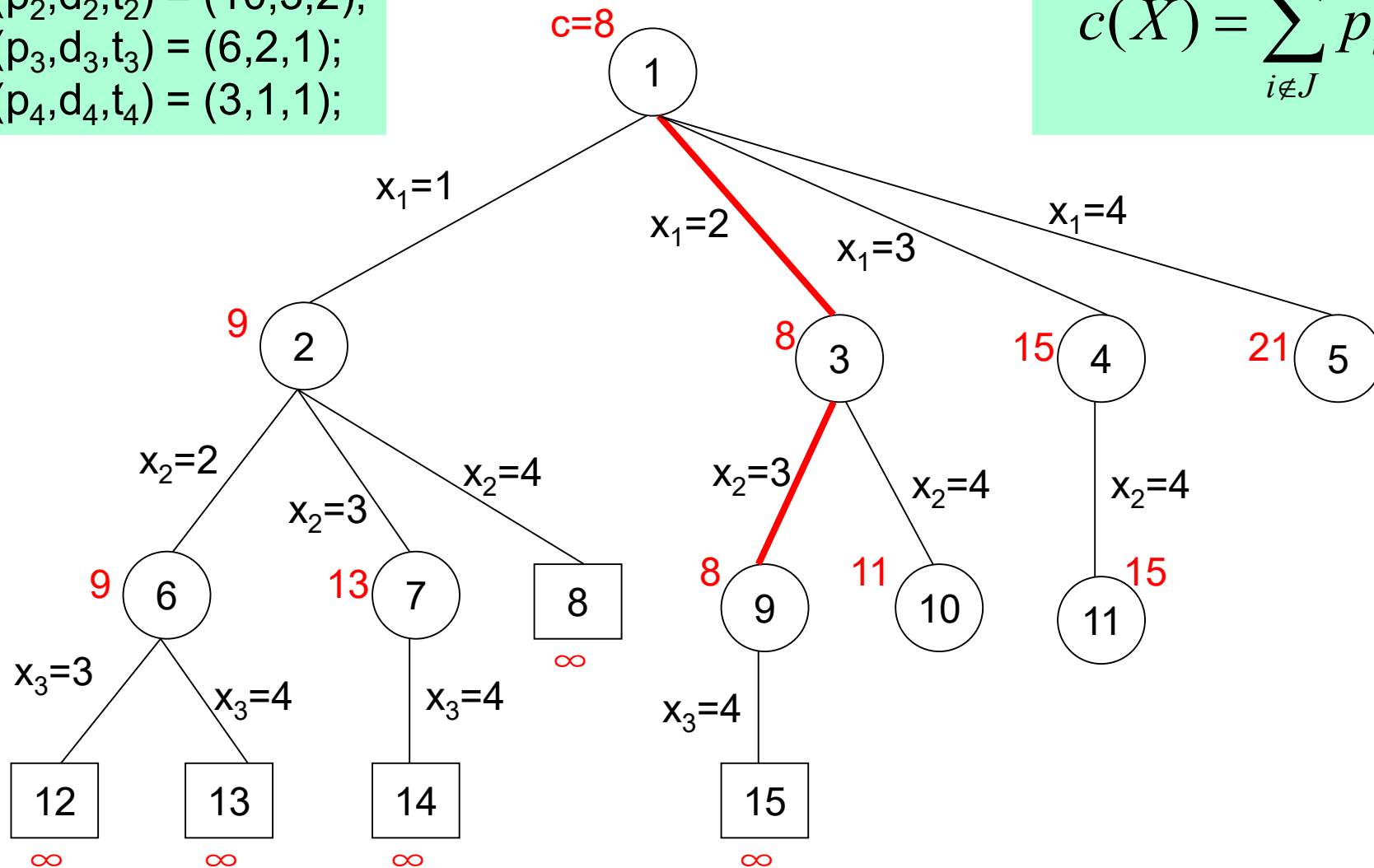
下图(d),  $c(2)=9$ ,  $c(5)=13$ ,  $c(6)=8$ ,  $c(1)=8$

$c(1)$ =最优解 $J$ 对应的罚款



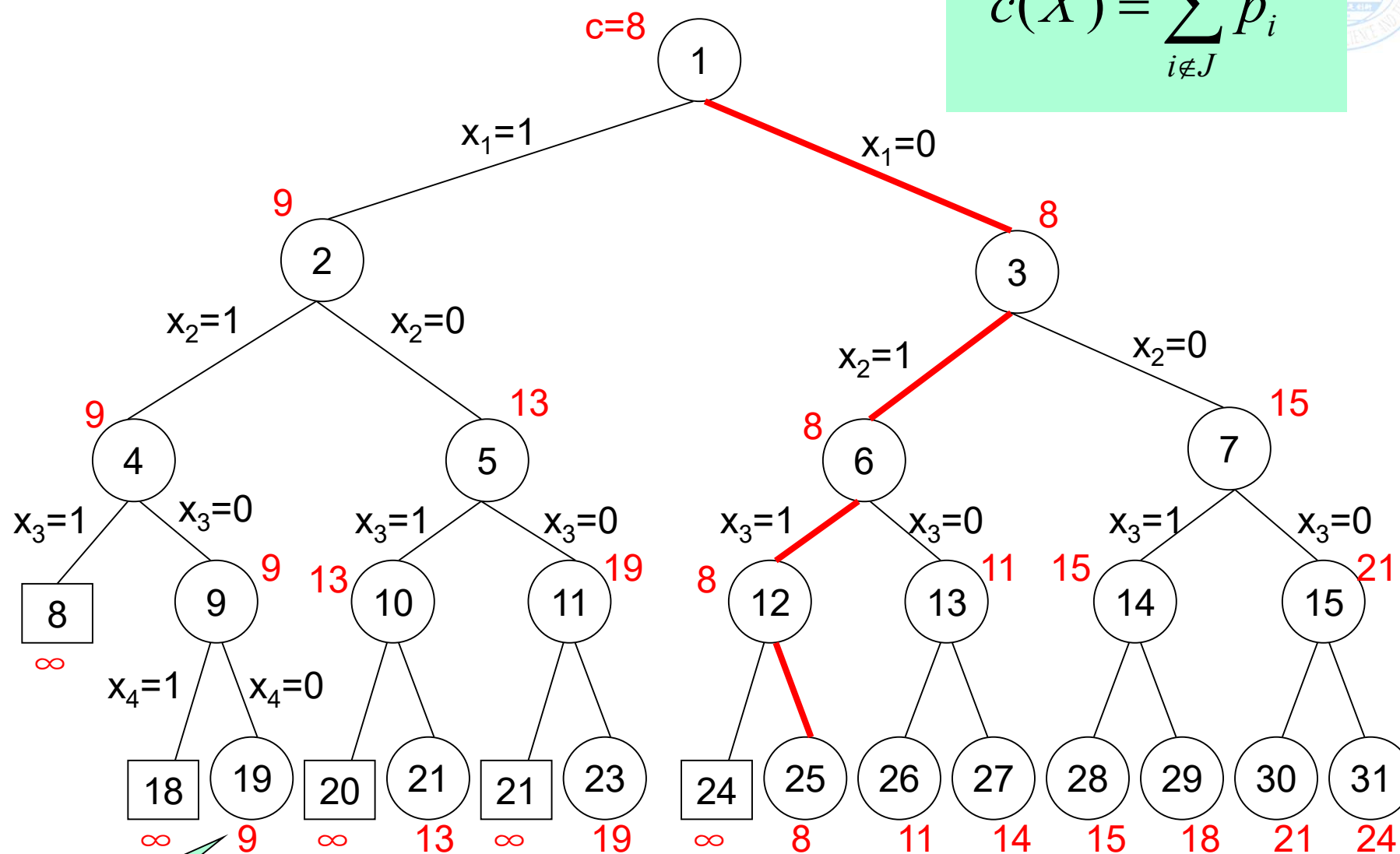
$(p_1, d_1, t_1) = (5, 1, 1);$   
 $(p_2, d_2, t_2) = (10, 3, 2);$   
 $(p_3, d_3, t_3) = (6, 2, 1);$   
 $(p_4, d_4, t_4) = (3, 1, 1);$

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



图(c) 采用大小可变的元组表示的状态空间树各结点的c值

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



图(d) 采用大小固定的元组表示的状态空间树各结点的c值

罚款值



## 成本估计函数 $\hat{c}(\bullet)$ 的定义

设  $S_x$  是考察结点  $X$  时, 已计入  $J$  中的作业的集合。

令  $m = \max \{i \mid i \in S_x\}$ ,

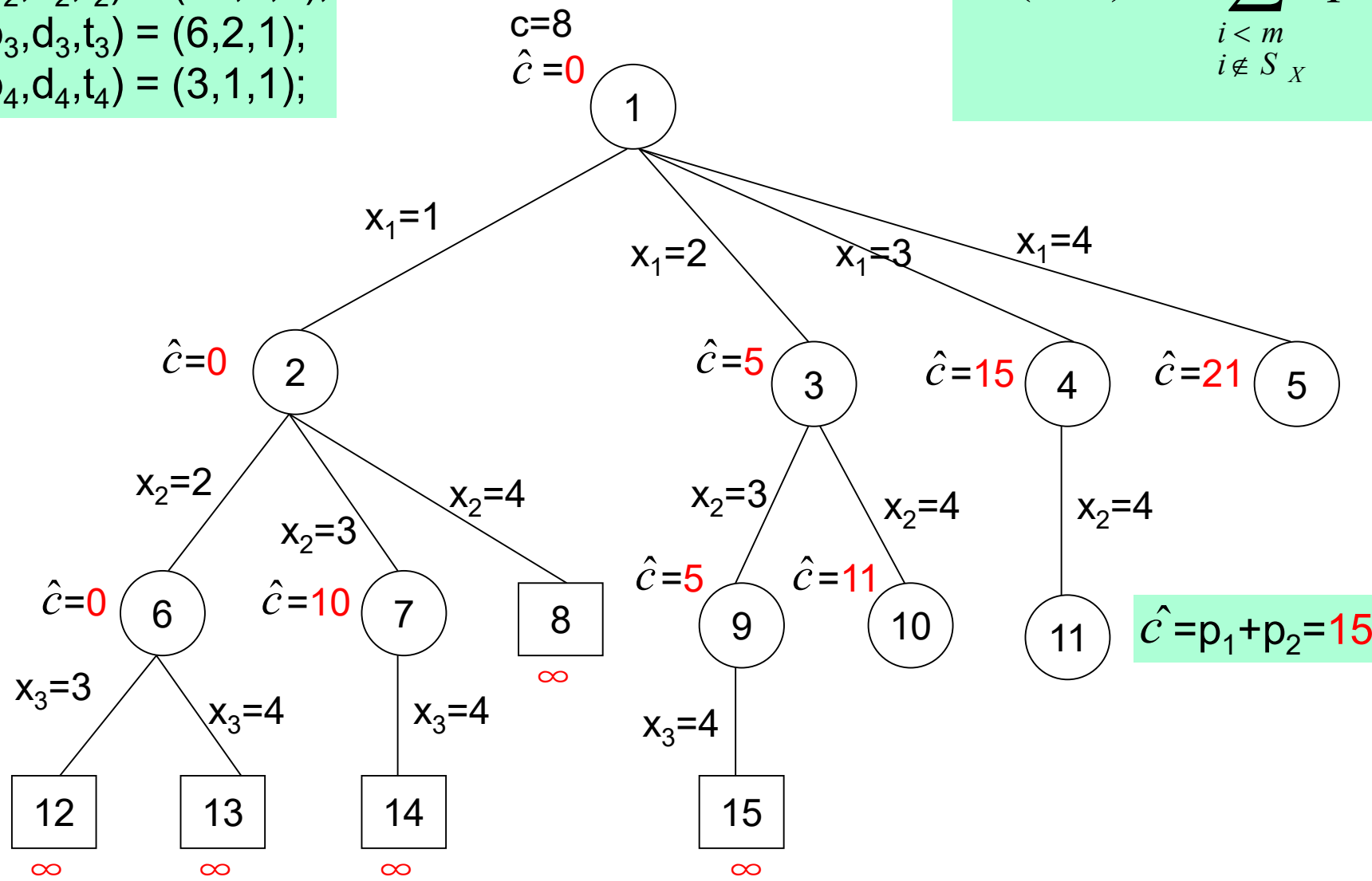
$$\hat{c}(X) = \sum_{\substack{i < m \\ i \notin S_x}} p_i$$

即  $\hat{c}(X)$  代表已经被考虑过但没有被计入  $J$  中的作业的罚款合计。这是已确定的罚款数, 因此  $\hat{c}(X) \leq c(X)$ , 是  $\hat{c}(X)$  可以作为  $c(X)$  的估计值(下界)。

例: 见图

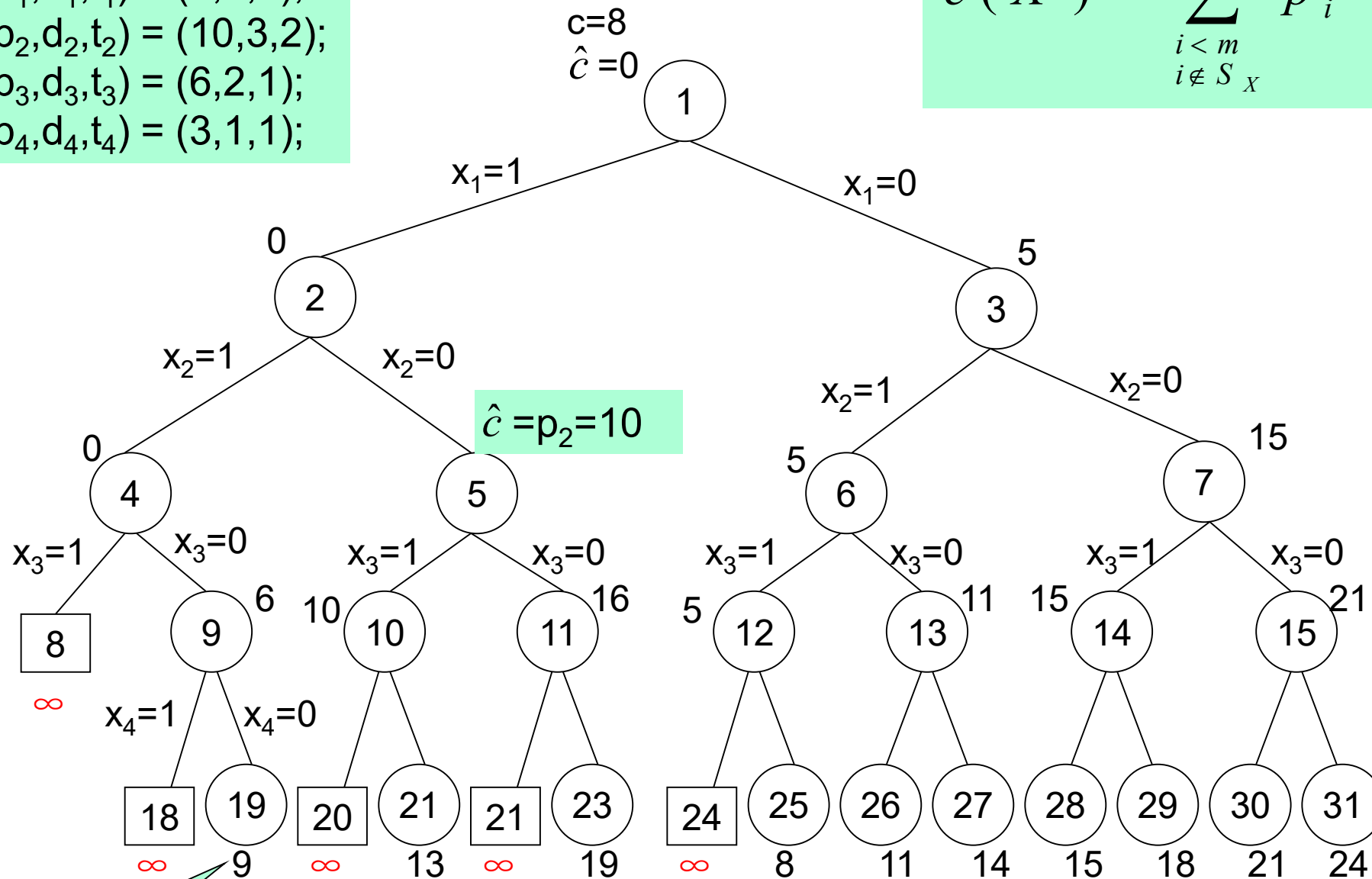
$(p_1, d_1, t_1) = (5, 1, 1);$   
 $(p_2, d_2, t_2) = (10, 3, 2);$   
 $(p_3, d_3, t_3) = (6, 2, 1);$   
 $(p_4, d_4, t_4) = (3, 1, 1);$

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



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

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



罚款值



## 成本估计函数上界的定义

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

$U(X)$  是  $c(X)$  的一个“简单”上界，代表目前没有被计入到  $J$  中的作业的罚款合计——可能的最多罚款，因此是目前罚款的上限。

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

$$\hat{c}(X) = \sum_{\substack{i < m \\ i \notin S_X}} p_i \quad u(1) = \sum_{1 \leq i \leq n} p_i$$

$$u(X) = \sum_{i \in S_X} p_i \quad U = \min(u(X))$$

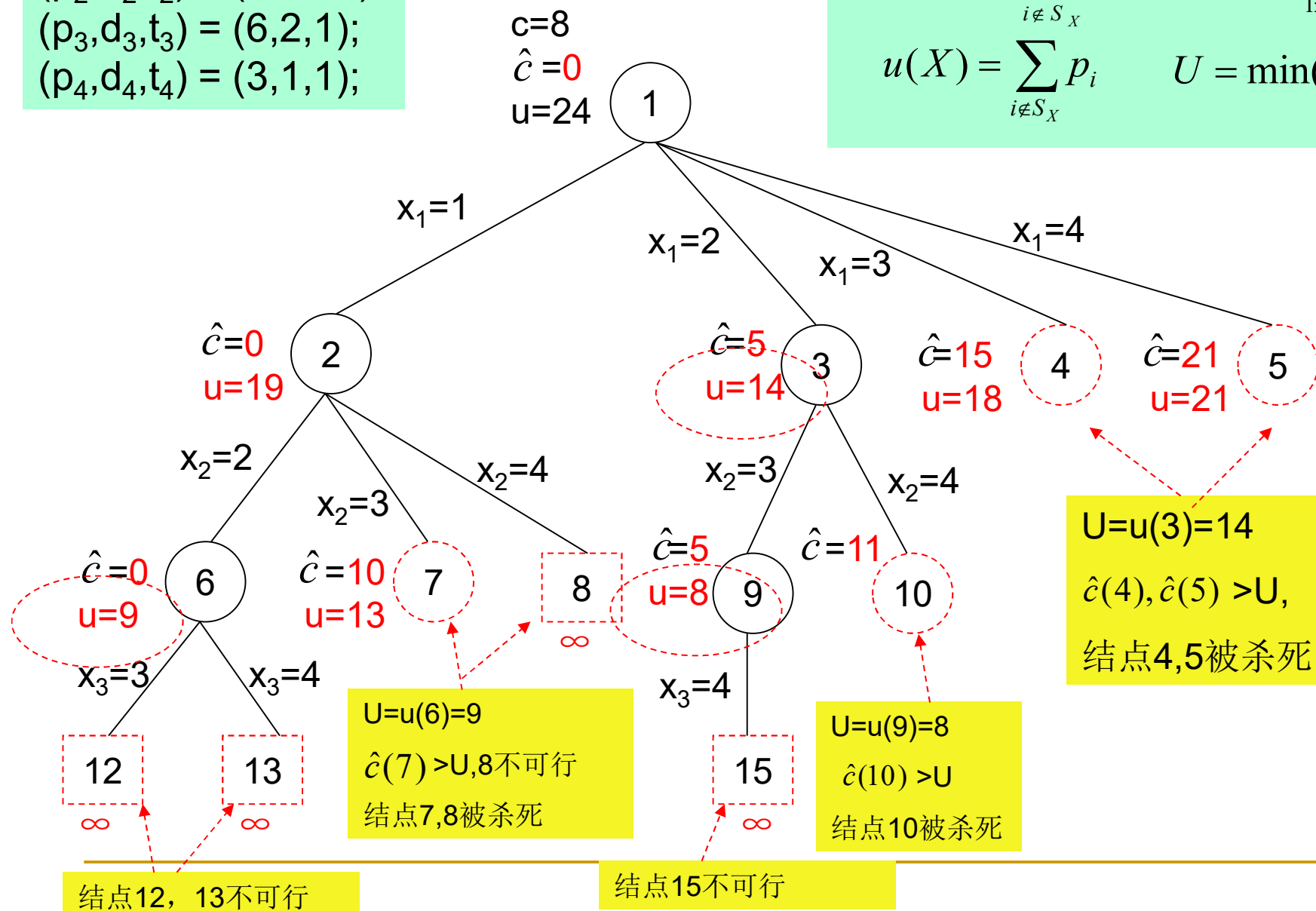


图9.7 采用大小可变的元组表示的状态空间树的成本估计值