《计算机算法设计与分析》

第七章 分枝—限界法

马丙鹏

2020年11月17日

中国科学院大学
University of Chinese Academy of Sciences

第七章 分枝－限界法

- 7.1 一般方法
- 7.2 LC-检索
- 7.3 15-谜问题
- 7.4 LC-检索(续)
- 7.5 分枝-限界算法
- 7.6 0/1背包问题
- 7.7 货郎担问题

7.3 15-谜问题

■ 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，要求通过一系列的合法移动将初始排列转换成目标排列。
- □合法移动：每次将一个邻接于空格的牌移动到空格位置。

(a) 初始排列

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

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

(b) 目标排列

7.3 15-谜问题

- 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)
  - 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。

7.3 15-谜问题

- 问题状态：
  - 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。
      - ✓并不是所有的初始状态都能变换成目标状态。
      - ✓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。

7.3 15-谜问题

- 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：按照目标状态中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

7.3 15-谜问题

- 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：

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。

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

POSITION(1:16) = (1, 5, 2, 3, 7, 10, 9, 13, 14, 15, 11, 8, 16, 12, 4, 6)

7.3 15-谜问题

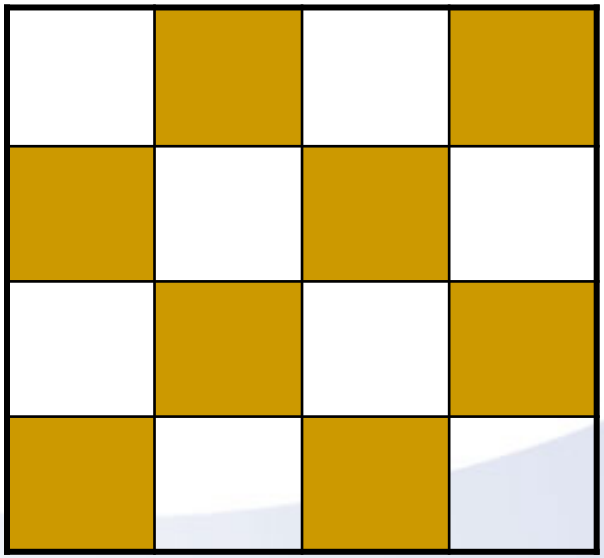
- 如何判定目标状态在初始状态的状态空间中？
  - 判定条件由以下4步给出：

③ 记LESS(i)是这样牌j的数目：
j<i，但POSITION(j) > POSITION(i)，
即：编号小于i但初始位置在i之后的牌的数目。
例：LESS(1)=0; LESS(4)=1; LESS(12)=6

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

7.3 15-谜问题

- 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：
    - 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。



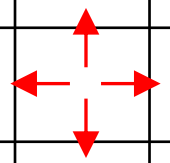
7.3 15-谜问题

- 如何判定目标状态在初始状态的状态空间中？
  - 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)

7.3 15-谜问题

- 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



7.3 15-谜问题

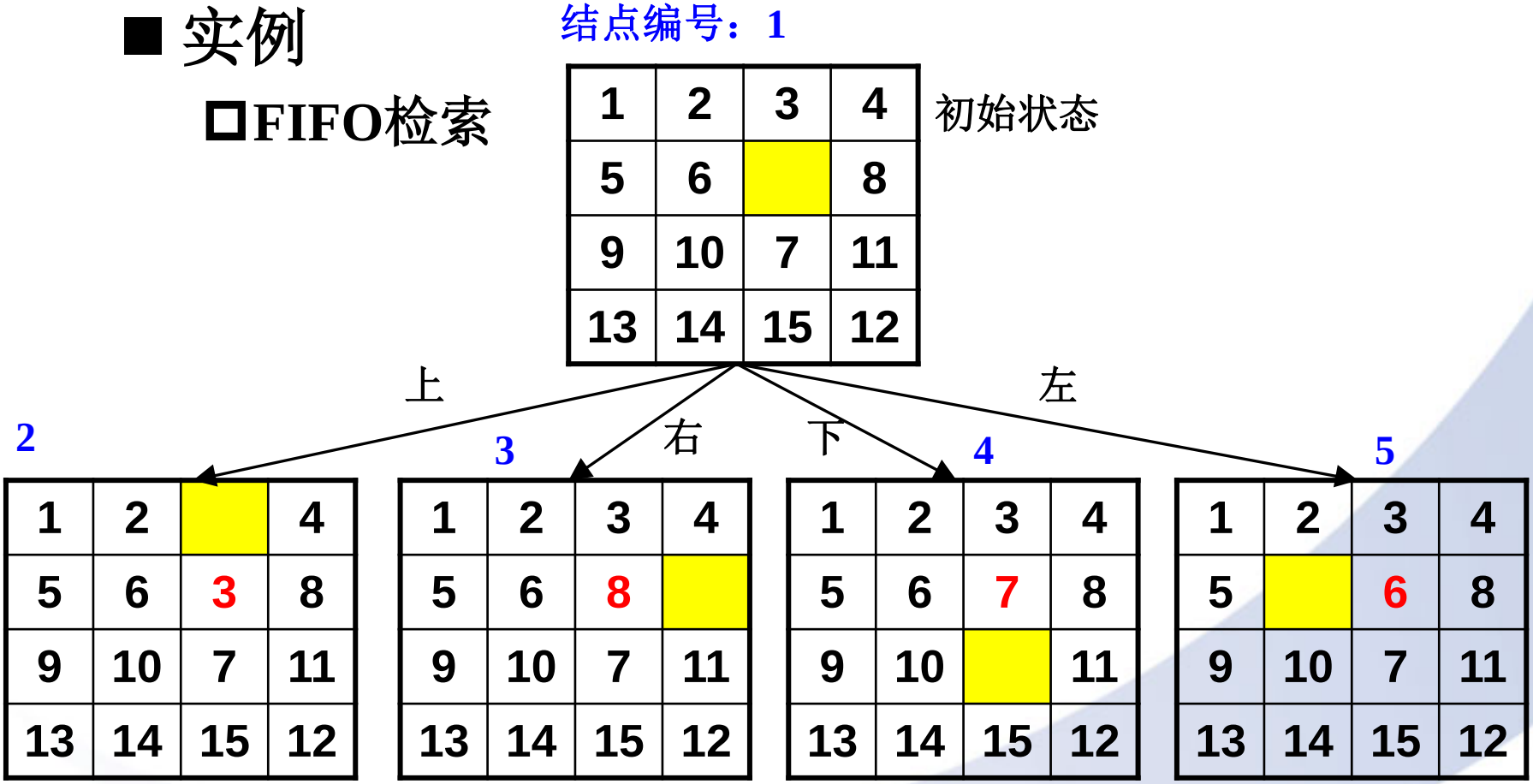
- 实例
  - 初始状态

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

  - 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。

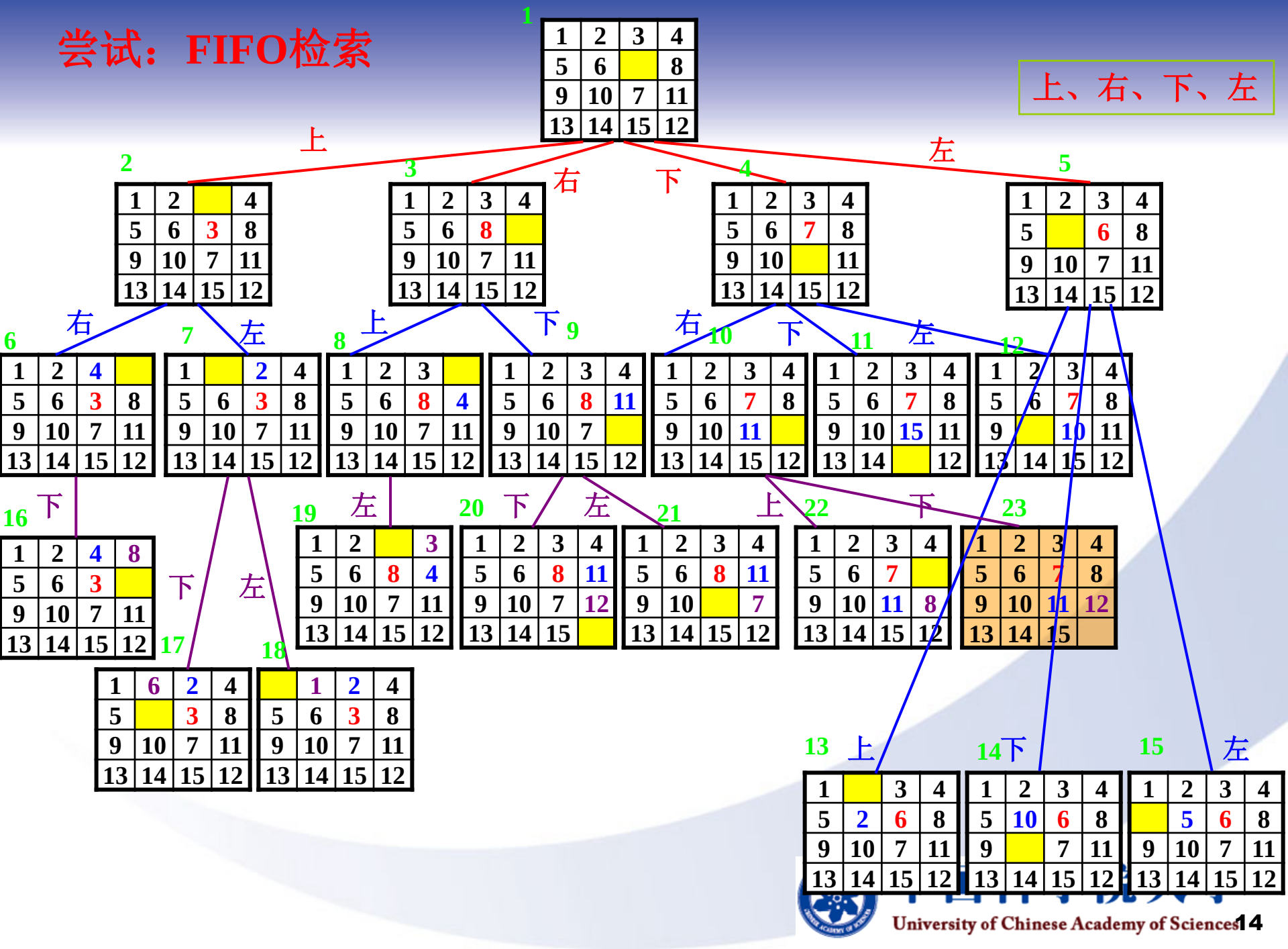
7.3 15-谜问题

- 实例
  - FIFO检索



尝试：FIFO检索

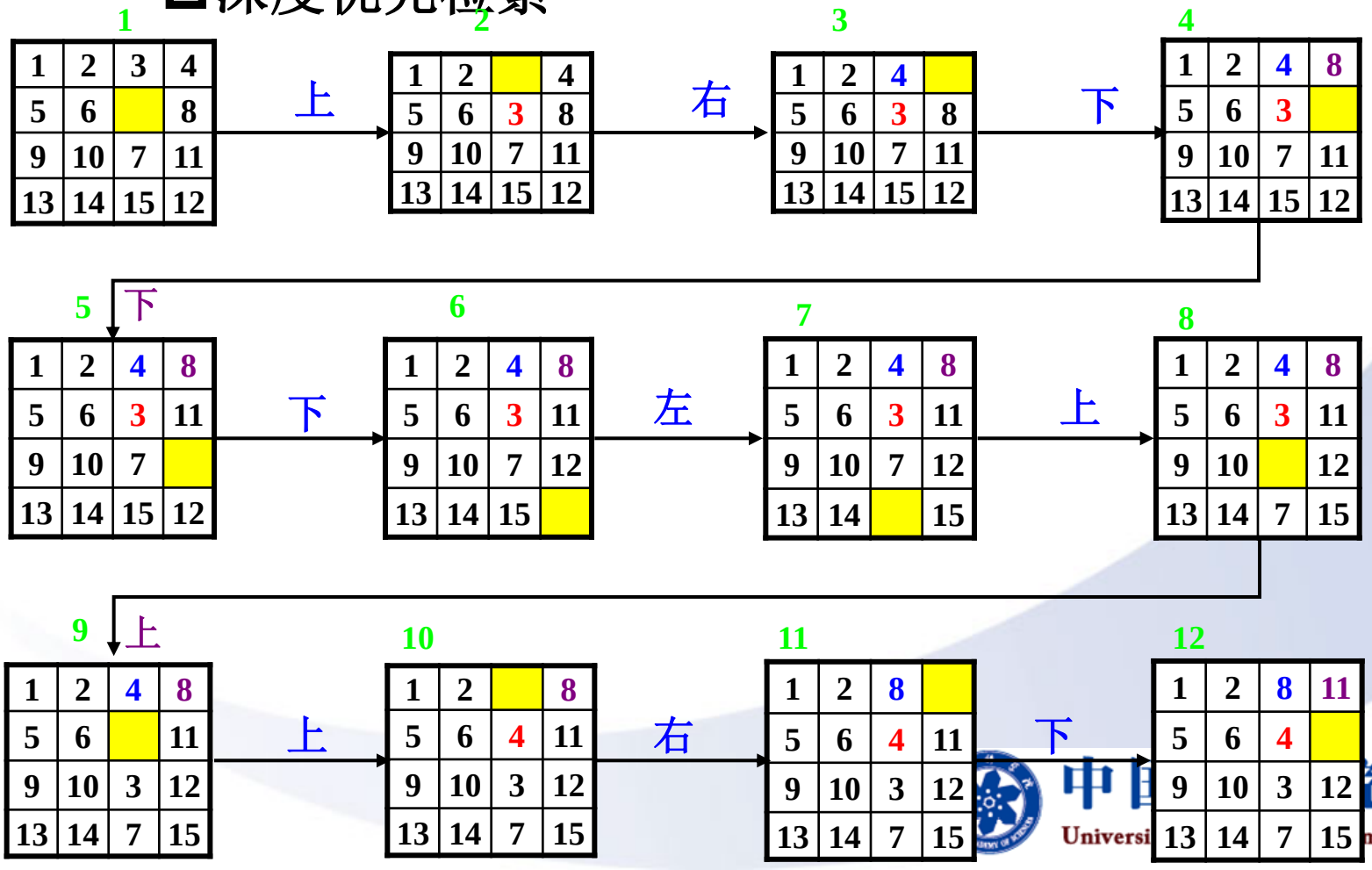
上、右、下、左



7.3 15-谜问题

■ 实例

□深度优先检索



7.3 15-谜问题

- 实例
  - 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)

7.3 15-谜问题

- 实例
  - 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。

7.3 15-谜问题

■ 实例

□如何定义c(X)?

➢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

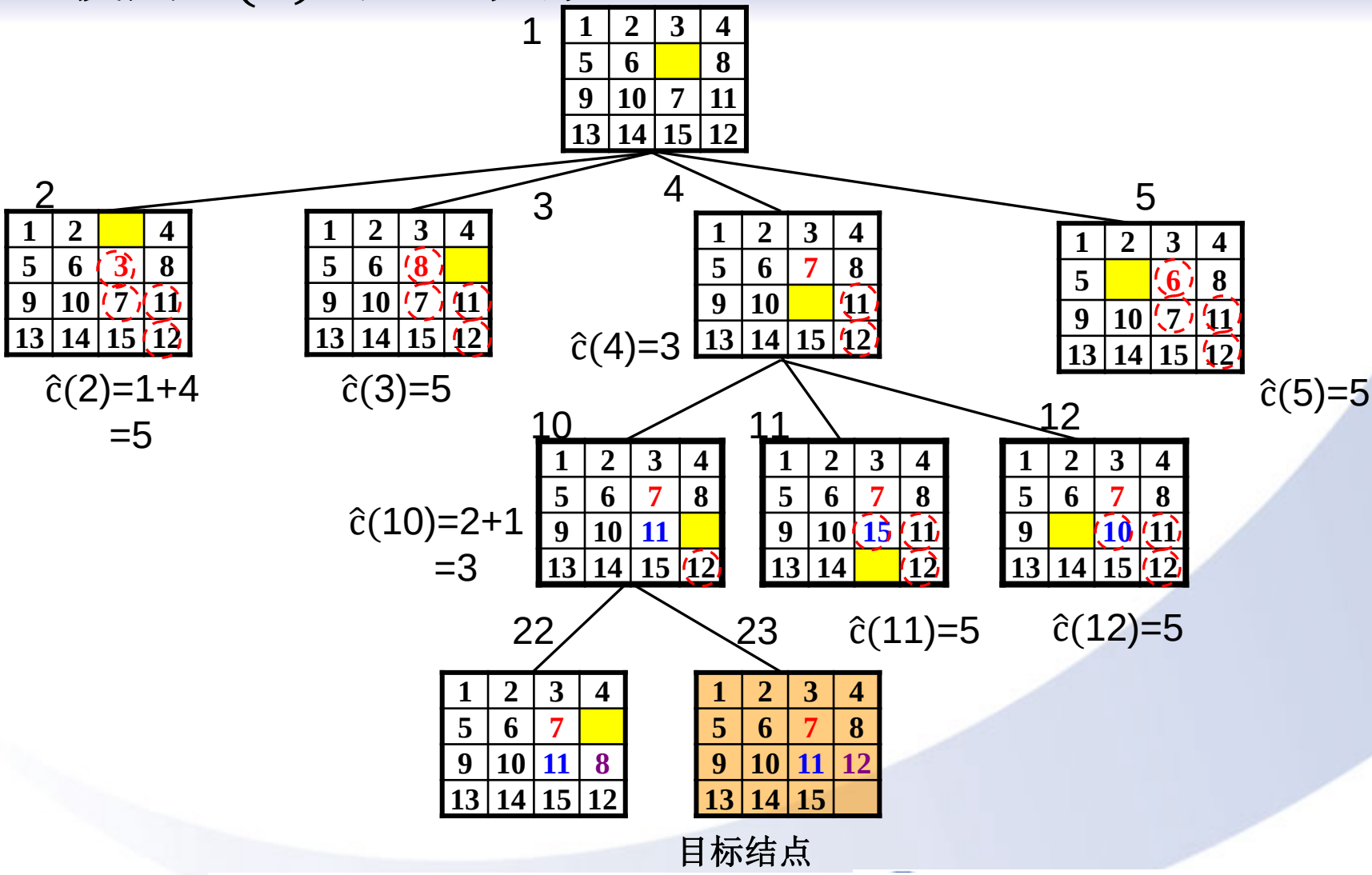
$\hat{g}(X)$ = 不在其目标位置的非空白牌数目

7.3 15-谜问题

- 实例
  - 如何定义c(X)?
    - 估算法：
      - ✓例：$\hat{g}(X) = 3$
        注：为达到目标状态所需的实际移动数>>$\hat{g}(X)$
      - ✓$\hat{c}(X)$是c(X)的下界

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

例：使用 ĉ(X)的LC-检索



使用ĉ(X)的LC-检索可以使检索过程很快地定位到结点23

第七章 分枝－限界法

- 7.1 一般方法
- 7.2 LC-检索
- 7.3 15-谜问题
- 7.4 LC-检索(续)
- 7.5 分枝-限界算法
- 7.6 0/1背包问题
- 7.7 货郎担问题

7.4 LC-检索(Least Cost)

- LC-检索的抽象化控制
  - 成本估计函数的动机
    - 设T是一棵状态空间树，c(·)是T的结点成本函数。
    - c(X)是根为X的子树中任一答案结点的最小成本。
    - c(T)是T中最小成本答案结点。
    - c(·)难以找到 => 成本估计函数ĉ(X)
  - 成本估计函数的性质
    - 如果X是一个答案结点或者是一个叶结点，则c(X) = ĉ(X)。
  - LC-检索的抽象化控制：
    - 过程LC用ĉ(•)去寻找一个答案结点。

```
procedure LC(T, ĉ)
 //为找答案结点检索T, ĉ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 ← 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) ← 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ĉ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

7.4 LC-检索(Least Cost)

- LC-检索的抽象化控制
  - 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。

7.4 LC-检索(Least Cost)

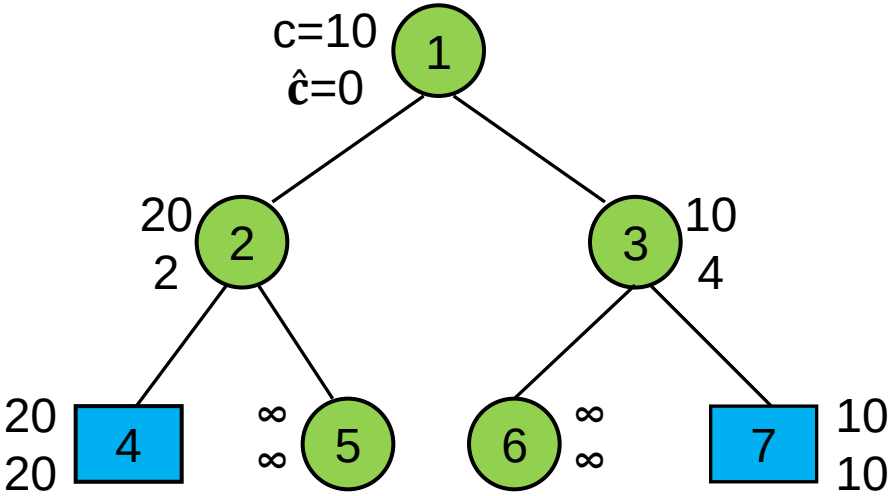
- LC的正确性证明
  - 算法LC是正确的。
    - 满足确定性、能行性、有输入、输出。
    - 满足终止性
      - ✓对于有限状态空间树，在找到一个答案结点或者在生成并检索了整棵状态空间树后算法终止。
      - ✓对于无限状态空间树，在找到一个答案结点或者对成本估计函数做出适当限制后也能使得算法终止。
    - 因此LC算法正确。
  - (详细证明略)

7.4 LC-检索(Least Cost)

- LC-检索与FIFO-检索和D-检索的关系
  - LC算法与FIFO-检索及D-检索基本相同：
    - 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
    - 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
  - 不同：
    - 对下一个E-结点的选择规则不同。

7.4 LC-检索(Least Cost)

- 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，$c(7) = 10$。

7.4 LC-检索(Least Cost)

- 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
  - 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)。
    - 找到这样的$\hat{c}(X)$通常是不可能的。

7.4 LC-检索(Least Cost)

- LC-检索的特性
  - 定理7.2 在有限状态空间树T中，对于每一个结点X，令是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$\hat{c}(\cdot)$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)

7.4 LC-检索(Least Cost)

- LC-检索的特性
  - 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LC：
    - 仍不能保证算法LC找到最小成本答案结点
  - 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。

7.4 LC-检索(Least Cost)

```
procedure LC1(T, ĉ)
 //为找出最小成本答案结点检索T, ĉ为具有上述性质的结点成本估计函数//
  E ← 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) ← 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ĉ 最小的活结点，赋给X，并从活结点表
      中删除//
  repeat
end LC1
```

生成结点后立即加入活结点表

7.4 LC-检索(Least Cost)

- 算法LC1正确性的证明
  - 定理7.3 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$\hat{c}(X) \le c(X)$，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：设此时E-结点是答案结点，L为活结点表中的任意一个结点。

① $\hat{c}(E) \le \hat{c}(L)$

② $\hat{c}(E) = c(E)$ ⟹ $c(E) \le c(L)$

③ $\hat{c}(L) \le c(L)$

End