Chapter 4

HEURISTIC SEARCH

（启发式搜索）

4.0 Introduction

- In state space search, heuristics are formalized as rules（规则）for choosing branches that are most likely to lead to an acceptable（可接受的）solution.
- AI problem solvers employ heuristics in two basic situations:

（1）A problem may not have an exact solution（精确解）because of inherent ambiguities（固有的含糊性）in the problem statement or available data.

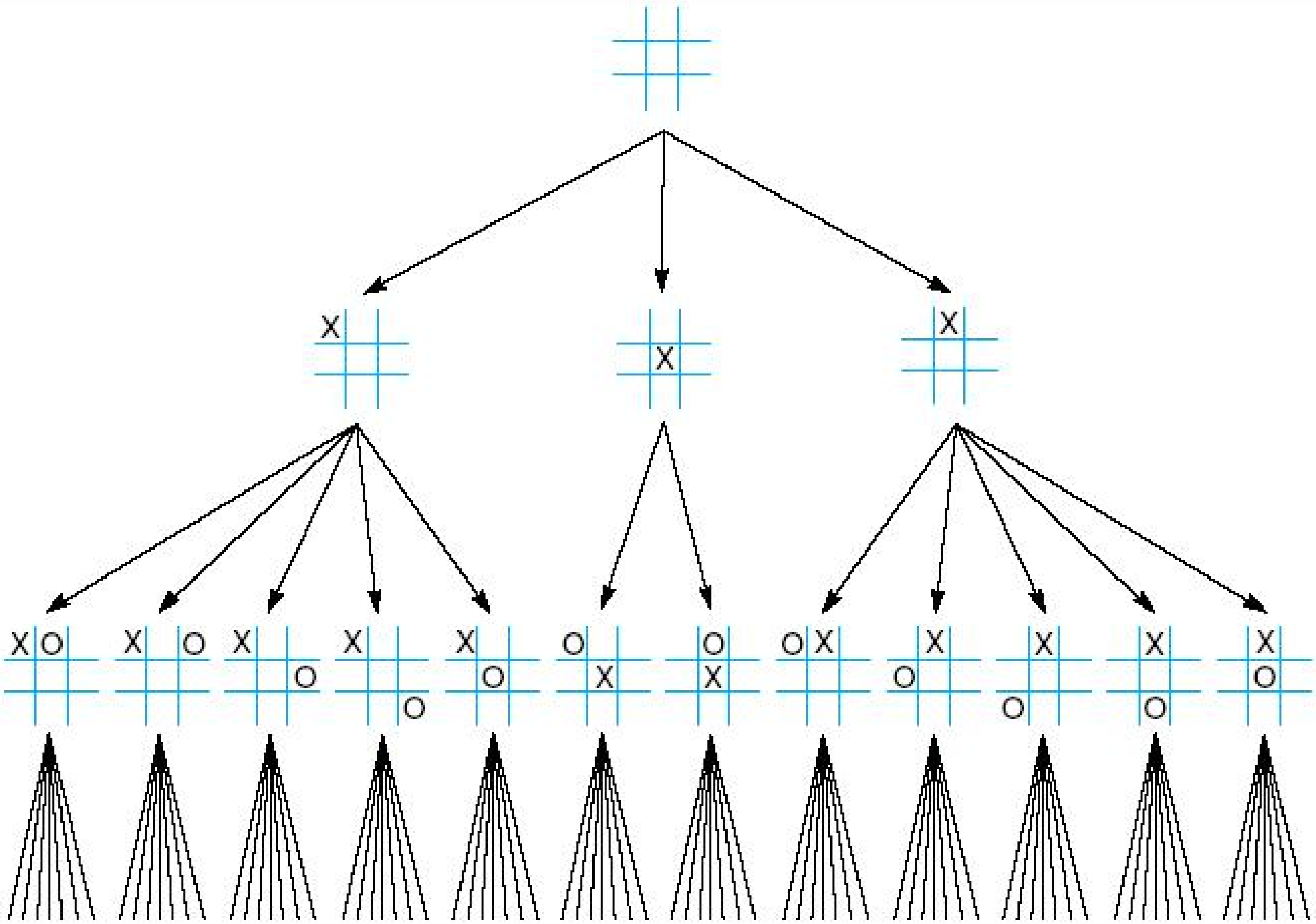
- Medical diagnosis is an example of this.

A given set of symptoms（症状）may have several possible causes（病因）;

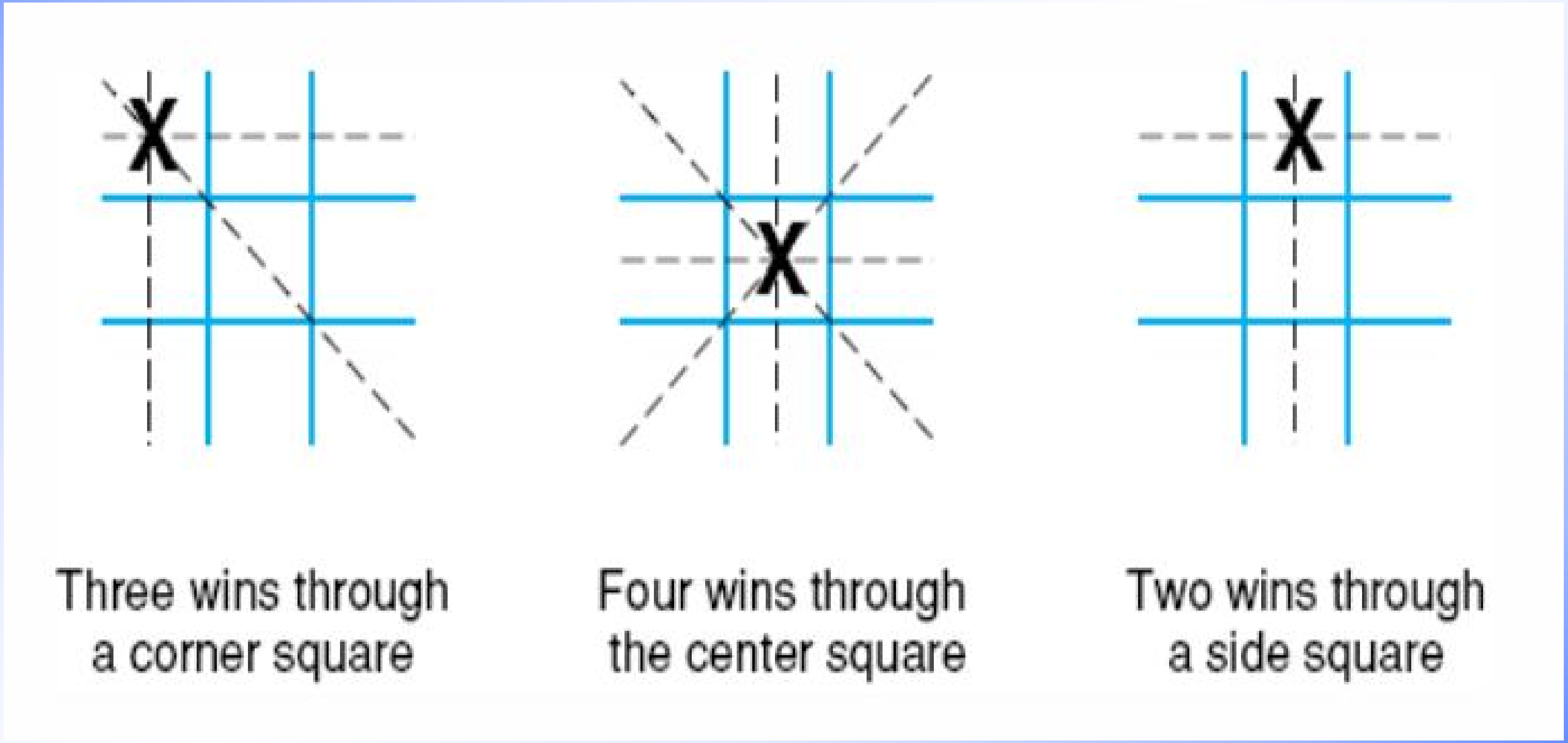
doctors use heuristics to choose the most likely diagnosis and formulate a plan of treatment.

（2） A problem, may have an exact solution, but the computational cost may be prohibitive（过高）. In many problems (such as chess), state space growth is combinatorially explosive（组合爆炸的）, with the number of possible states increasing exponentially（按指数方式） or factorially（按阶乘方式） with the depth of the search.

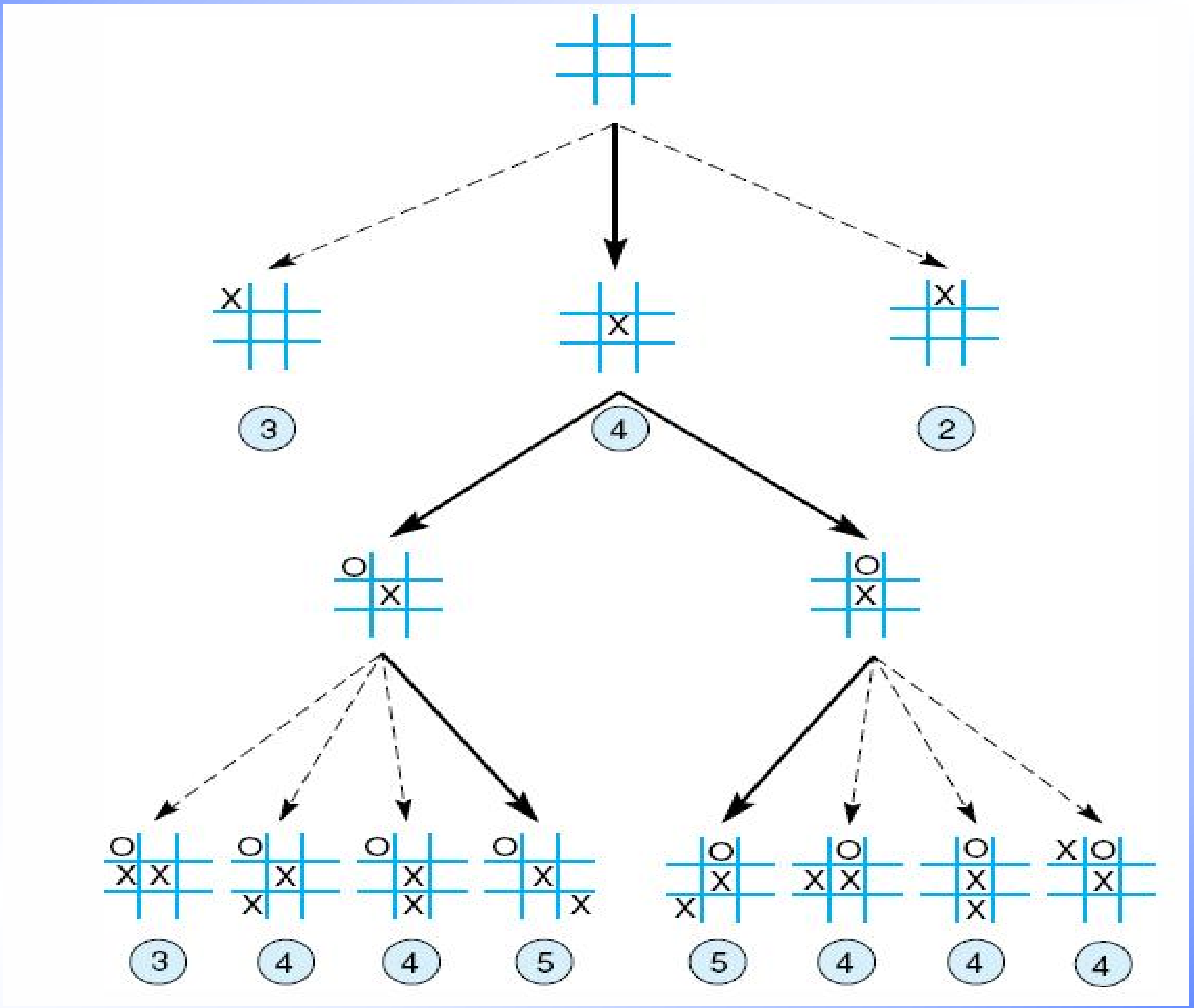
- Expert systems research has affirmed（肯定） the importance of heuristics.
- The “rules of thumb”（经验法则） that a human expert uses to solve problems efficiently are heuristic in nature.



The "most wins" heuristic applied to the first children in tic-tac-toe



Heuristically reduced state space for tic-tac-toe



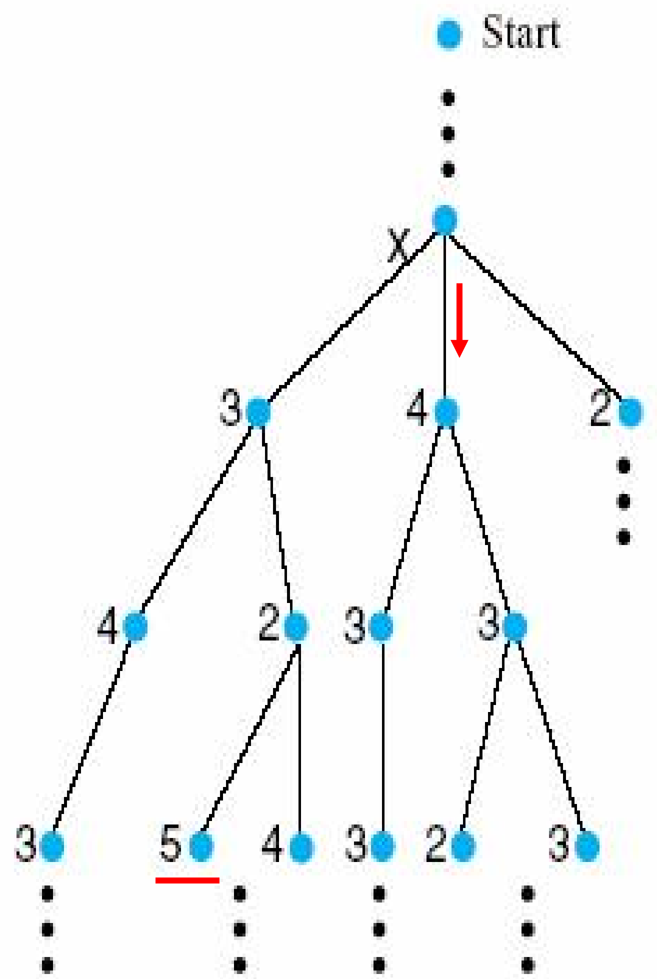
4.1 Hill-Climbing and Dynamic Programming

4.1.1 Hill-Climbing

- Hill-climbing strategies（爬山法）expand the current state and evaluate its children.
- The “best” child (better than parent) is selected for further expansion; neither its siblings nor its parent are retained（保留）.
- Hill climbing is named for the strategy that might be used by an eager but blind mountain climber : go uphill along the steepest（最陡）possible path until you can go no farther up.
- Because it keeps no history, it cannot recover（恢复）from failures .

- **A major problem** of hill-climbing strategy is the tendency to **become stuck at local maxima** （局部极大值）.

- In this case, it may fail to find the best solution.



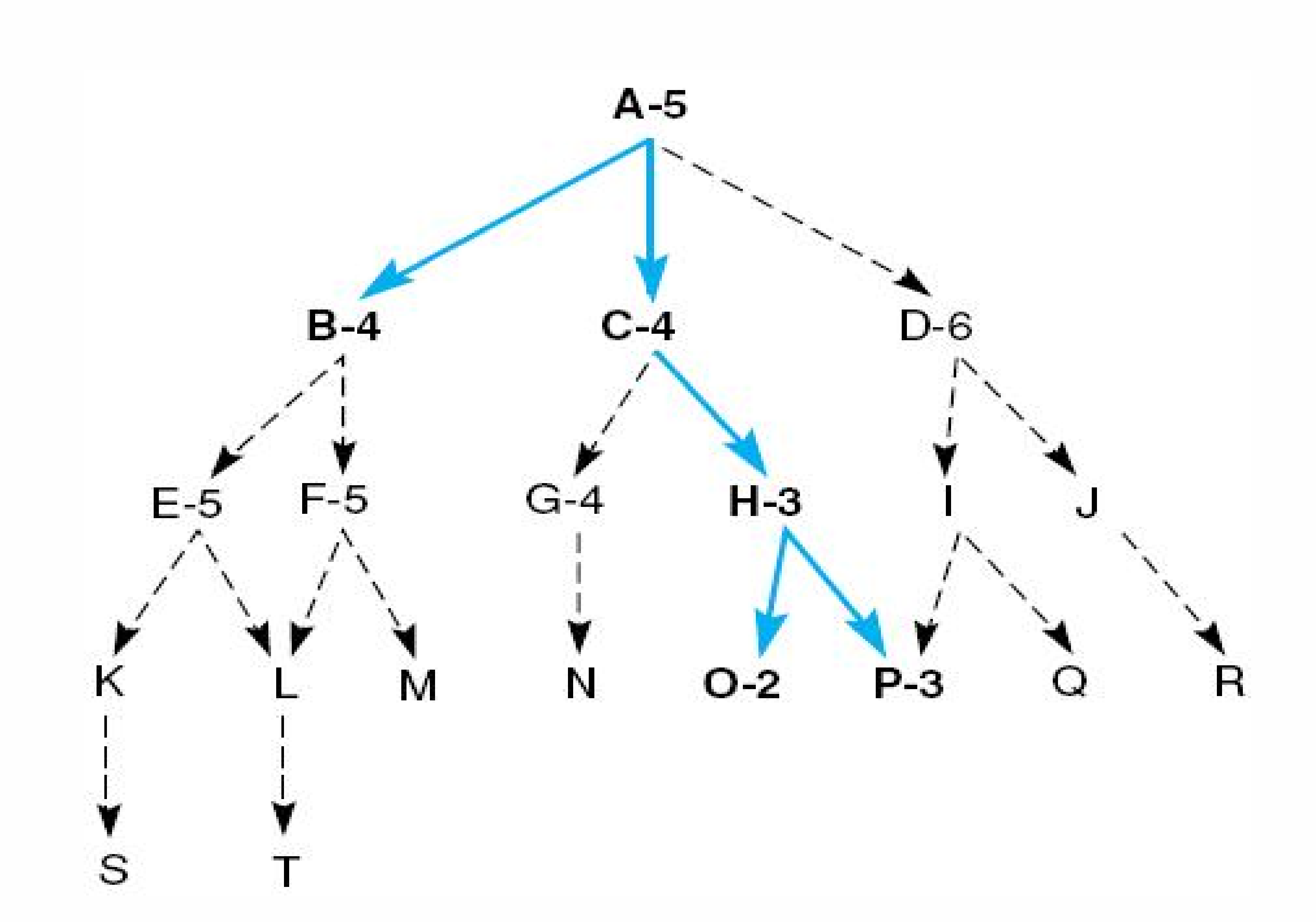
4.2 The Best-First Search Algorithm

4.2.1 Implementing Best-First Search

- open list : a priority queue（优先级队列） to keep track of the current fringe（边界结点）of the search
- closed list : to record states that already visited.
- An added step : orders（排序）the states on open according to some heuristic estimate（启发估值） of their “closeness” to a goal.
- Each iteration（迭代）considers the most “promising” state on the open list.

```
Function best-first-search;
  begin
     open := [Start];                    % initialize
     closed := [ ];
     while open ≠ [ ] do            % state remain
        begin
        remove the leftmost state from open, call it X;
        if X=goal then return the path from Start to X
        else begin
                 generate children of X;
                 for each child of X do
                 case
                    the child is not on open or closed:
                       begin
                          assign the child heuristic value;
                          add the child open
                       end;
                    the child is already on open:
                       if the child was reached by a shorter path
                       then give the state on open the shorter path
                    the child is already on closed:
                       if the child was reached by a shorter path then
                          begin
                             remove the state from closed;
                             add the child to open
                          end;
                 end;  % case
                 put X on closed;
                 re-order states on open by heuristic merit ( best leftmost )
              end
        end;
     return FAIL     % open is empty
  end
```

best-first-search of a hypothetical（假想的）state space



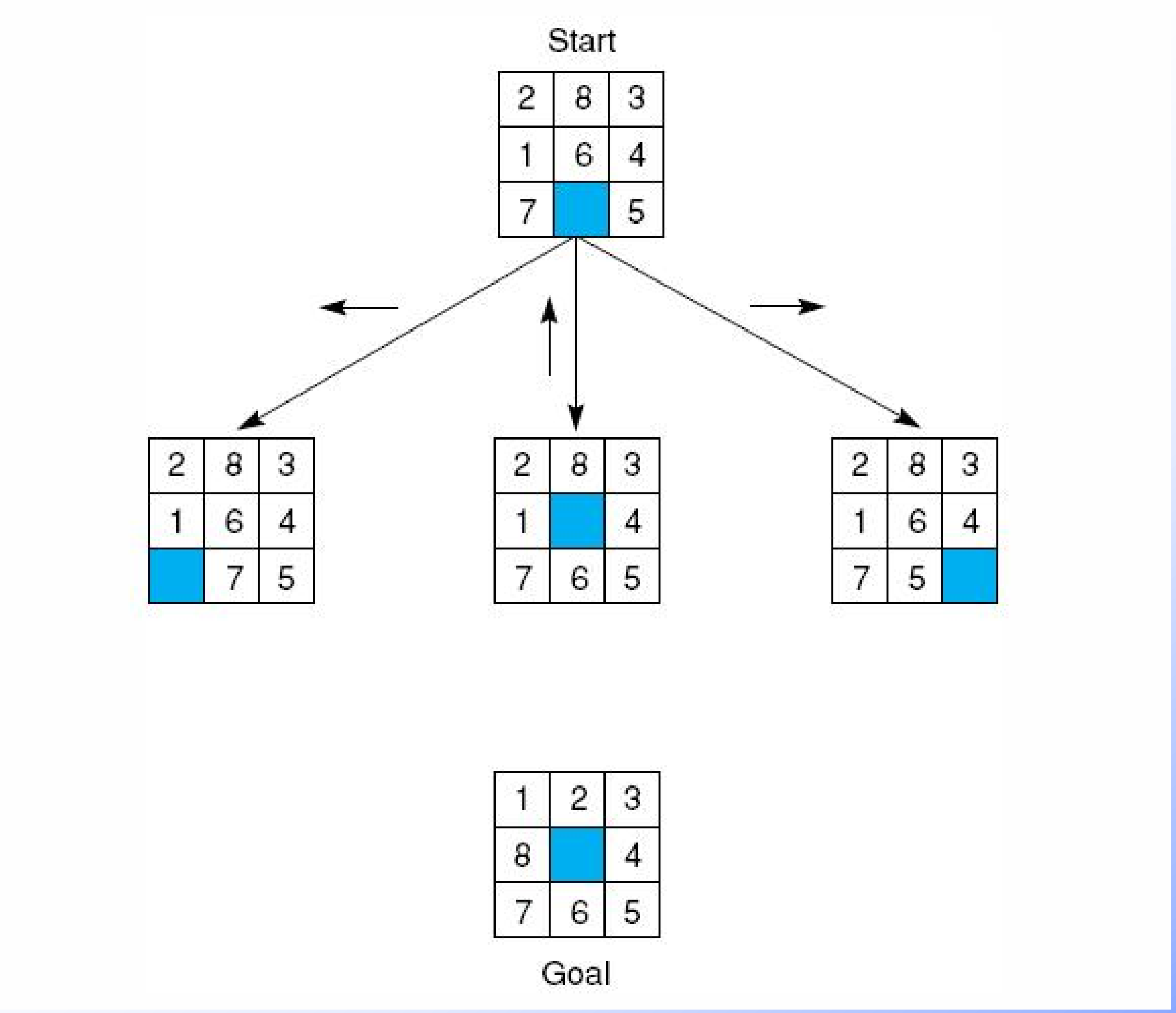
A trace of the execution of best-first-search

1. open=[A5]; closed=[ ]
2. evaluate A5; open=[B4,C4,D6]; closed=[A5]
3. evaluate B4; open=[C4,E5,F5,D6]; closed=[B4,A5]
4. evaluate C4; open=[H3,G4,E5,F5,D6]; closed=[C4,B4,A5]
5. evaluate H3; open=[O2,P3,G4,E5,F5,D6]; closed=[H3,C4,B4,A5]
6. evaluate O2; open=[P3,G4,E5,F5,D6]; closed=[O2,H3,C4,B4,A5]
7. evaluate P3; the solution is found!

4.2.2 Implementing Heuristic Evaluation Functions

- We next evaluate the performance（性能） of several different heuristics for solving the 8-puzzle.
- Figure 4.12 shows a start and goal state for the 8-puzzle, along with the first three states generated in the search.

The start state, first three moves, and goal state for an example - 8 puzzle



- The **simplest** heuristic **counts the tiles（将牌）out of place（错位）** in each state when **compared with** the goal.
- A "**better**" heuristic would **sum all the distances** by which **the tiles are out of place**, one for each square **a tile must be moved to reach its position** in the goal state.

- Both of these heuristics can be criticized for failing to acknowledge（反映）the difficulty of tile reversals（逆转）.
- That is , if two tiles are next to each other and the goal requires their being in opposite locations, it takes (several) more than two moves to put them back in place, as the tiles must “go around” each other (Figure 4.13)

- A heuristic that takes this into account multiplies a small number 2 :

$$2 \times \text{num}$$

num is the number of direct tile reversals.

- Figure 4.14 shows the result of applying each of these three heuristics to the three child states of Figure 4.12

<table>
<tr><td><table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td>6</td><td>4</td></tr><tr><td></td><td>7</td><td>5</td></tr></table></td><td>5</td><td>6</td><td>0</td></tr>
<tr><td><table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td></td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table></td><td>3</td><td>4</td><td>0</td></tr>
<tr><td><table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td>6</td><td>4</td></tr><tr><td>7</td><td>5</td><td></td></tr></table></td><td>5</td><td>6</td><td>0</td></tr>
<tr><td></td><td>Tiles out of place</td><td>Sum of distances out of place</td><td>2 x the number of direct tile reversals</td></tr>
</table>

1	2	3
8		4
7	6	5

Goal

- A forth heuristic, which may overcome the limitations of the tile reversal heuristic :
- adds the sum of the distances the tiles are out of place and 2 times the number of direct reversals.

- **Full evaluation function:**

$$f(n) = g(n) + h(n)$$

- **g(n) is the length of the current shortest path from the start state to state n**

- **h(n) is a heuristic estimate of the distance from state n to a goal.**

- In the 8-puzzle, we can define h(n) as follows:

  h(n) = the number of tiles out of place

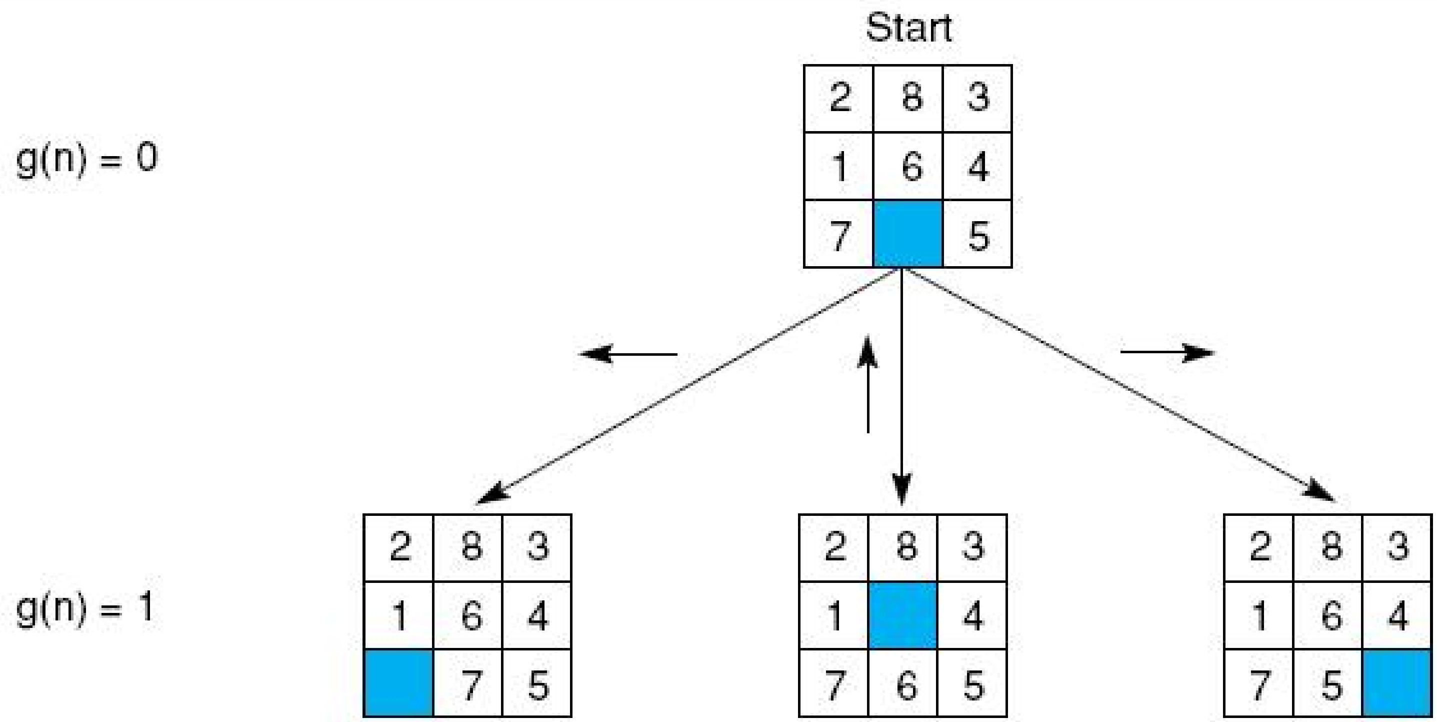
- We can define g(n) as follows:

  g(n) = the depth of n

- The full evaluation function is :

  f(n) = g(n) + h(n)

- When this evaluation is applied to each of the child states in Figure 4.12, their f values are 6, 4, and 6, respectively.



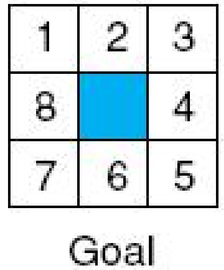
Values of f(n) for each state, **6** **4** **6**

where:

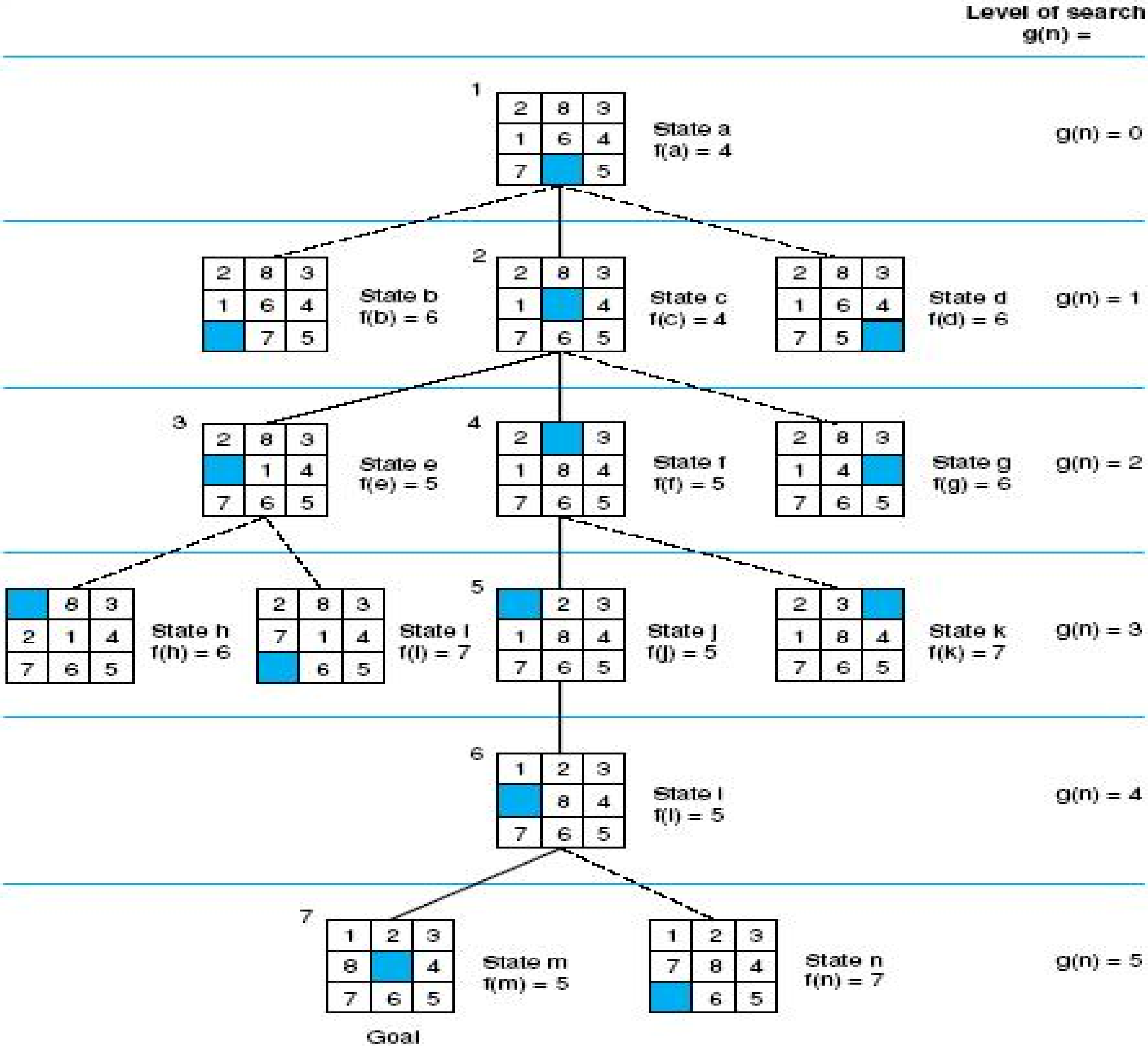
f(n) = g(n) + h(n),

g(n) = actual distance from n to the start state, and

h(n) = number of tiles out of place.



- The full best-first search of the 8-puzzle graph, using f as defined above, appears in Figure 4.16.
- Each state is labeled with a letter and its heuristic weight, $f(n)=g(n)+h(n)$.
- The number at the top of each state indicates the order in which it was taken off the open list.



Level of search
g(n) =
State a
f(a) = 4
g(n) = 0
State b
f(b) = 6
State c
f(c) = 4
State d
f(d) = 6
g(n) = 1
State e
f(e) = 5
State f
f(f) = 5
State g
f(g) = 6
g(n) = 2
State h
f(h) = 6
State i
f(i) = 7
State j
f(j) = 5
State k
f(k) = 7
g(n) = 3
State l
f(l) = 5
g(n) = 4
State m
f(m) = 5
State n
f(n) = 7
g(n) = 5
Goal

- The **successive stages** of open and closed that generate this graph are:

1. open=[a4] closed=[ ]
2. open=[c4,b6,d6] closed=[a4]
3. open=[e5,f5,b6,d6,g6] closed=[a4,c4]
4. open=[f5,h6,b6,d6,g6,i7] closed=[a4,c4,e5]
5. open=[j5,h6,b6,d6,g6,k7,i7] closed=[a4,c4,e5,f5]
6. open=[l5,h6,b6,d6,g6,k7,i7] closed=[a4,c4,e5,f5,j5]
7. open=[m5,h6,b6,d6,g6,n7,k7,i7]
   closed=[a4,c4,e5,f5,j5,l5]
8. m=goal ! success

- To summarize:

① Operations on a state generate its children.

② Each new state is checked to see whether it has occurred before, thereby preventing loops.

③ Each state n is given an f value equal to the sum of its depth in the search space g(n) and a heuristic estimate of its distance to a goal h(n).

- The h value guides search toward promising states
- the g value can prevent search from persisting on a fruitless path.

④ States on open are sorted by their f values. By keeping all states on open until they are examined or a goal is found, the algorithm recovers from dead ends.

⑤ As an implementation point, the algorithm’s efficiency can be improved through maintenance of the open as *heaps*（堆） or *leftist trees*（左偏树）.

4.2.3 Heuristic Search and Expert Systems

EXAMPLE 4.2.1

THE FINANCIAL ADVISOR, REVISITED

savings-account(adequate) $\wedge$ income(adequate)

$$\Downarrow$$

investment(stocks)

with confidence = 0.8

4.3 Admissibility, Monotonicity and Informedness

- **Admissibility（可采纳性）: heuristics that find the shortest path to a goal whenever it exists are said to be *admissible*.**

- **Informedness（信息性）: In what sense is one heuristic "better" than another?**

- **Monotonicity（单调性）: When a state is discovered (examined), is there any guarantee that the same state won't be found later at a cheaper cost (with a shorter path)?**

4.3.1 Admissibility Measures

- A search algorithm is *admissible*（可采纳的）if it is guaranteed to find a minimal path to a goal whenever such a path exists.

- define an evaluation function f^*：

$f^*(n)=g^*(n)+h^*(n)$, where $g^*(n)$ is the cost of the shortest path from the start to node n and h^* returns the cost of the shortest path from n to goal.

- It follows that $f^*(n)$ is the cost of the optimal path from a start node to a goal node that passes through node n（经过 n）.

- **DEFINITION**

(algorithm A, admissibility, algorithm A*)

- **Consider the evaluation function**

$$f(n) = g(n) + h(n)$$

- **if f(n) is used with the best_first_search algorithm, the result is called *algorithm A***
- **If h(n) is less than or equal to h* (n), the algorithm is called *algorithm* A***
- **It is now possible to state a property of A* algorithms:**

All A* algorithms are admissible.

4.3.2 Monotonicity

- **DEFINITION (MONOTONICITY)**

A heuristic function h is monotone（单调的）if

1. **For all states n_i and n_j, where n_j is a decedent of n_i,**

$$h(n_i) - h(n_j) \le cost(n_i, n_j)$$

2. **h(Goal)=0**

- If the heuristics function is monotone, the algorithm will be "locally admissible"

- That is, it will consistently find the minimal path to each state x when it is expanded,

- i.e. $g(x) = g^*(x)$

- This means that when finding a new path to a node x in CLOSED, we don't need to update the value of $g(x)$.

- A simple argument can show that any monotonic heuristic is admissible.
- Considers any solution path in the space as a sequence of states $s_1, s_2, \ldots, s_g$, where s_1 is the start state and s_g is the goal.
- For the sequence of moves in this path:

s_1 to s_2 $h(s_1) - h(s_2) \le cost(s_1, s_2)$

s_2 to s_3 $h(s_2) - h(s_3) \le cost(s_2, s_3)$

s_3 to s_4 $h(s_3) - h(s_4) \le cost(s_3, s_4)$

....

s_{g-1} to s_g $h(s_{g-1}) - h(s_g) \le cost(s_{g-1}, s_g)$

- Summing each column
- $h(s_g) = 0$

  we get : $h(s_1) \leq cost(s_1, s_g)$

- When $s_1, s_2, \ldots s_g$ is the best solution path,

  there is : $cost(s_1, s_g) = h^*(s_1)$

  that is : $h(s_1) \leq h^*(s_1)$

- Let s_1 be any state x, then :

  $$h(x) \leq h^*(x)$$

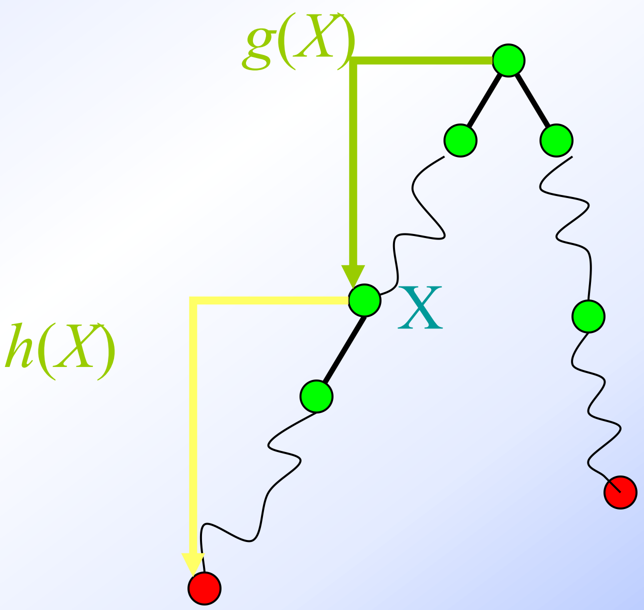
- This means that monotone heuristic h is A^* and admissible.

4.3.3 When One Heuristic Is Better : More Informed Heuristics

- **DEFINITION：INFORMEDNESS （信息度）**
  - For two A^* heuristics h_1 and h_2
  - if $h_1(n) \leq h_2(n)$, for all states n in the search space,
  - heuristic h_2 is said to be *more informed* than h_1

估价函数与择优搜索

- 估价函数概念：用于估价节点重要性的函数称为估价函数。
- 一般形式：$f(x)=g(x)+h(x)$
  - $g(x)$为从初始节点S_0到节点x已经实际付出的代价；
  - $h(x)$是从节点x到目标节点的最优路径的估计代价，它体现了问题的启发性信息，其形式要根据问题的特性确定。



- 八数码难题的启发函数h（x），可以定义为节点x中数码位置与目标节点相比不同的个数。

A*算法小结

- A*算法引入了估价函数f(n)，它是对价值函数f*(n)估计。
- 价值函数f*(n)表示从初始结点经过结点n而到达目标结点最小花费的代价。
- f*(n)它分为两部份，一是从初始结点到结点n的最小代价，记为g*(n)，另一部份是从结点n到目标结点的最小代价，记为h*(n)；所以f*(n)=g*(n)+h*(n)。
- f(n)是对函数f*(n)估计。

- **f(n)=g(n)+h(n)**，其中**f(n)**是节点**n**的估价函数。**g(n)**是在状态空间中从初始节点到**n**节点的实际代价，**h(n)**是从**n**到目标节点最佳路径的估计代价。
- 条件： **h(n)<= h*(n),g(n)>= g*(n)**
- 宽度优先搜索算法就是**A***算法的特例。**f(n)=g(n)+h(n)**，其中**g(n)**是节点所在的层数，**h(n)=0**。
- **A***算法找到的是初始状态到目标状态的最优路径，而并非在路径上每个状态都最优。而如果**h(n)**满足单调性，则在这个路径上的每一处都是可纳的 。

A*算法应用——搜索最短路径

（一）

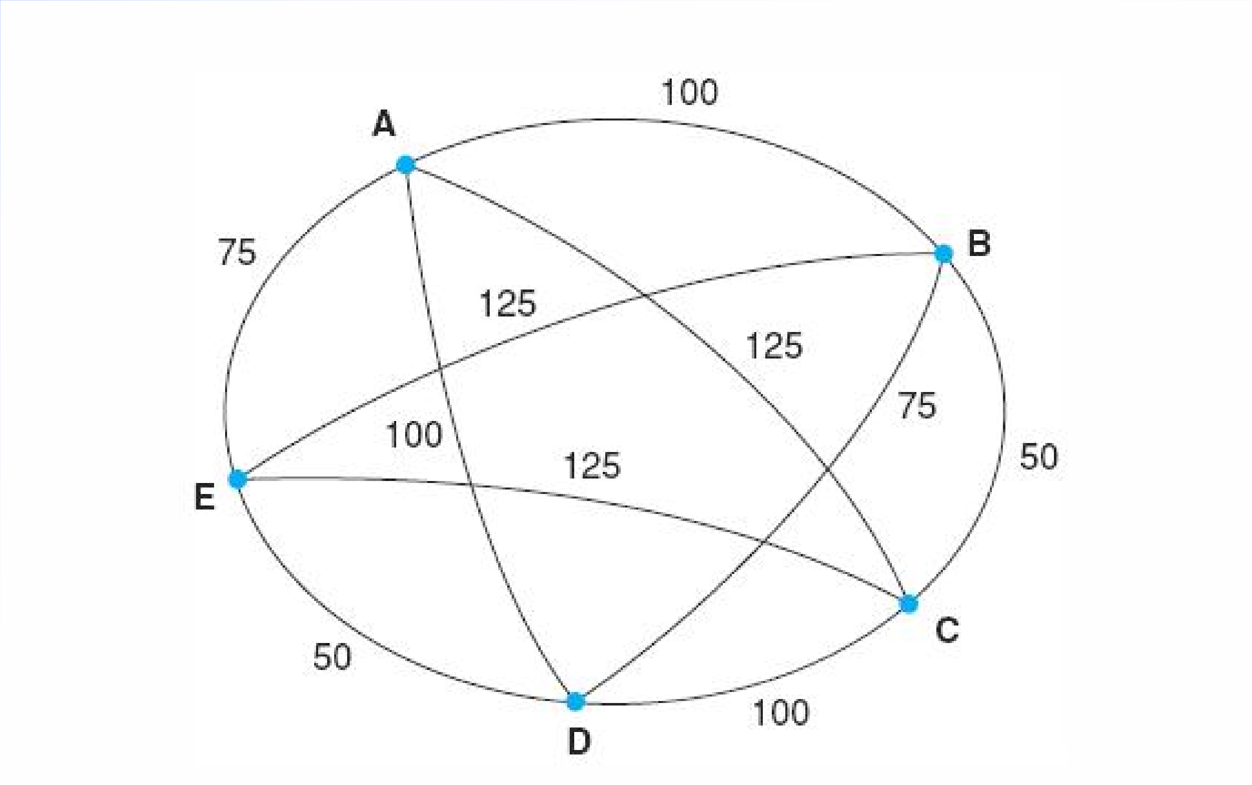
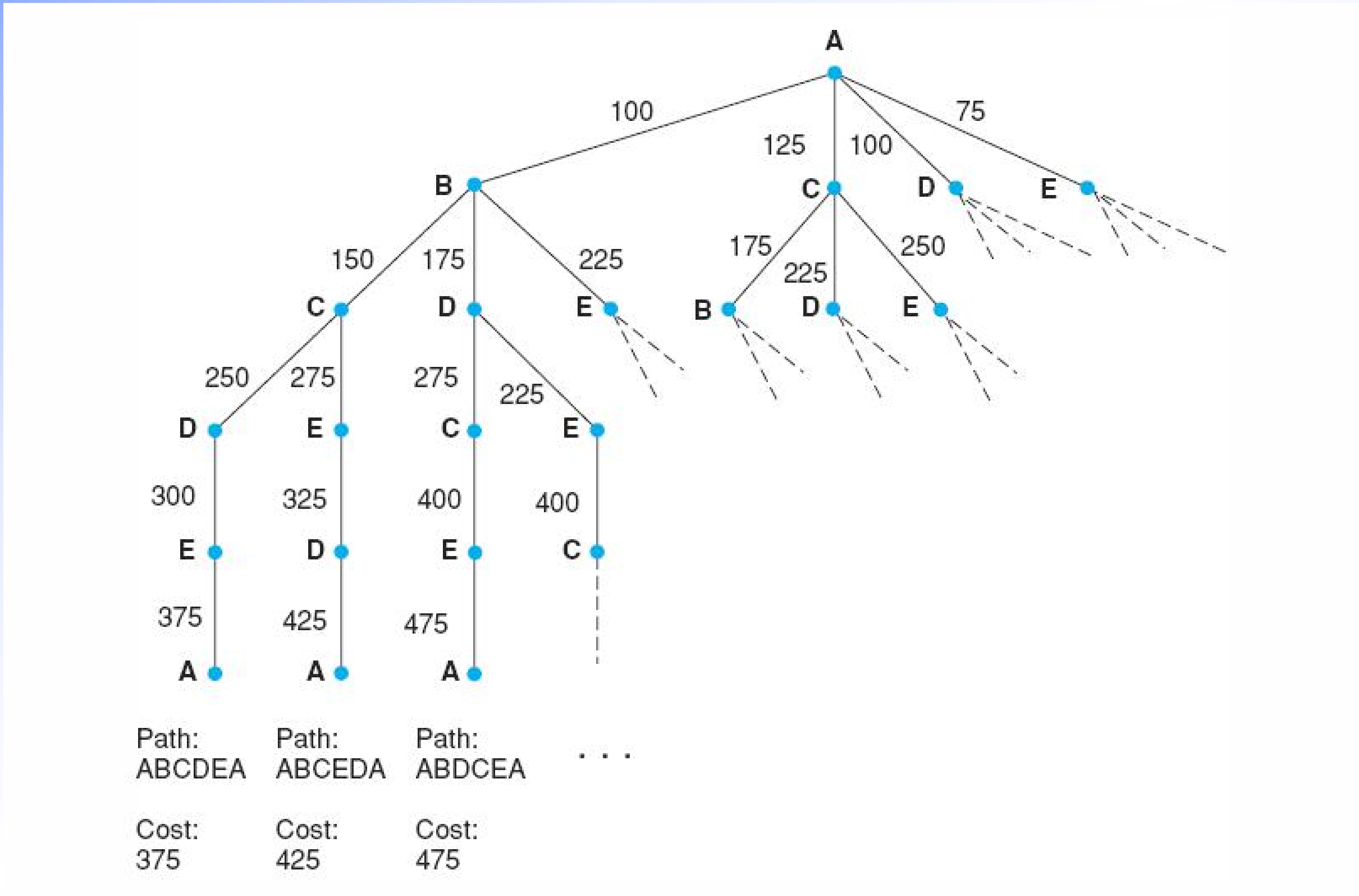


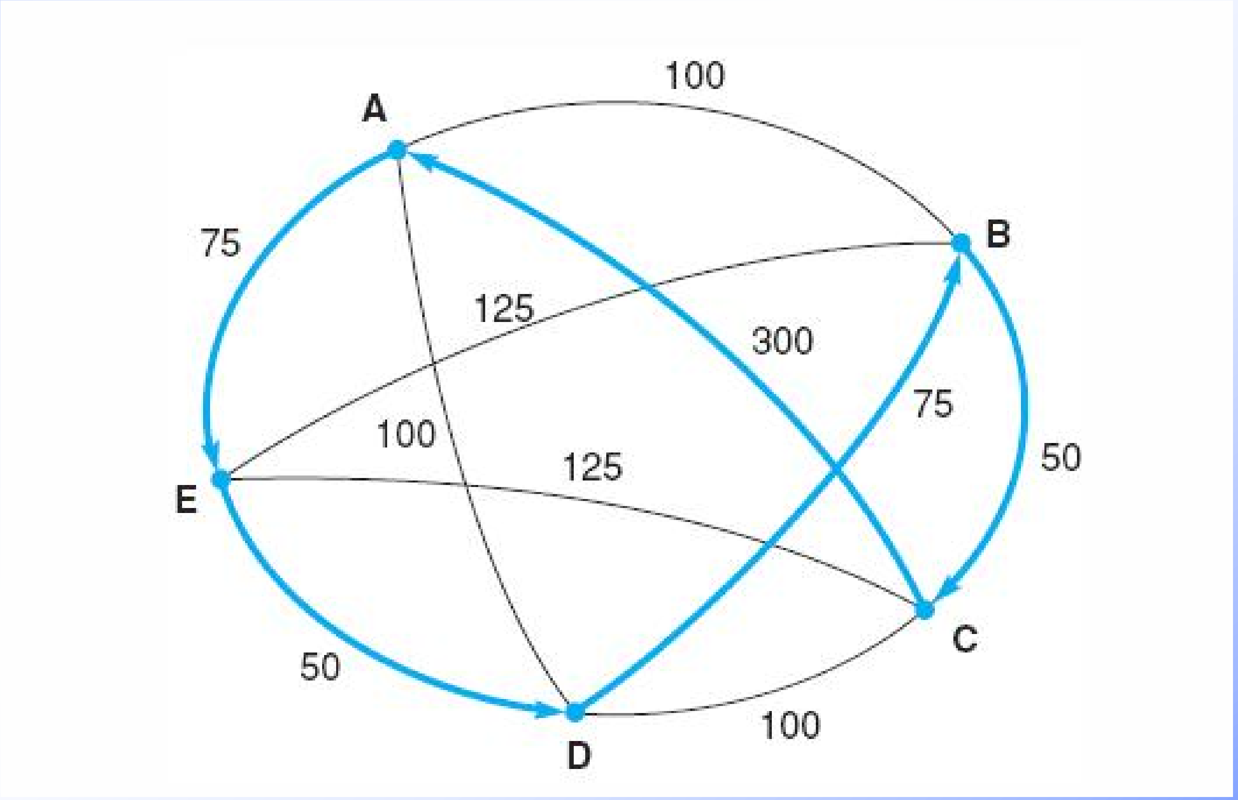
Fig 3.9 An instance of the travelling salesperson problem

Fig 3.10.Search for the travelling salesperson problem. Each arc is marked with the total weight of all paths from the start node (A) to its endpoint.



An instance of the travelling salesperson problem with the nearest neighbor path in bold.

Note this path (A, E, D, B, C, A), at a cost of 550, is not the shortest path. The comparatively high cost of arc (C, A) defeated the heuristic.



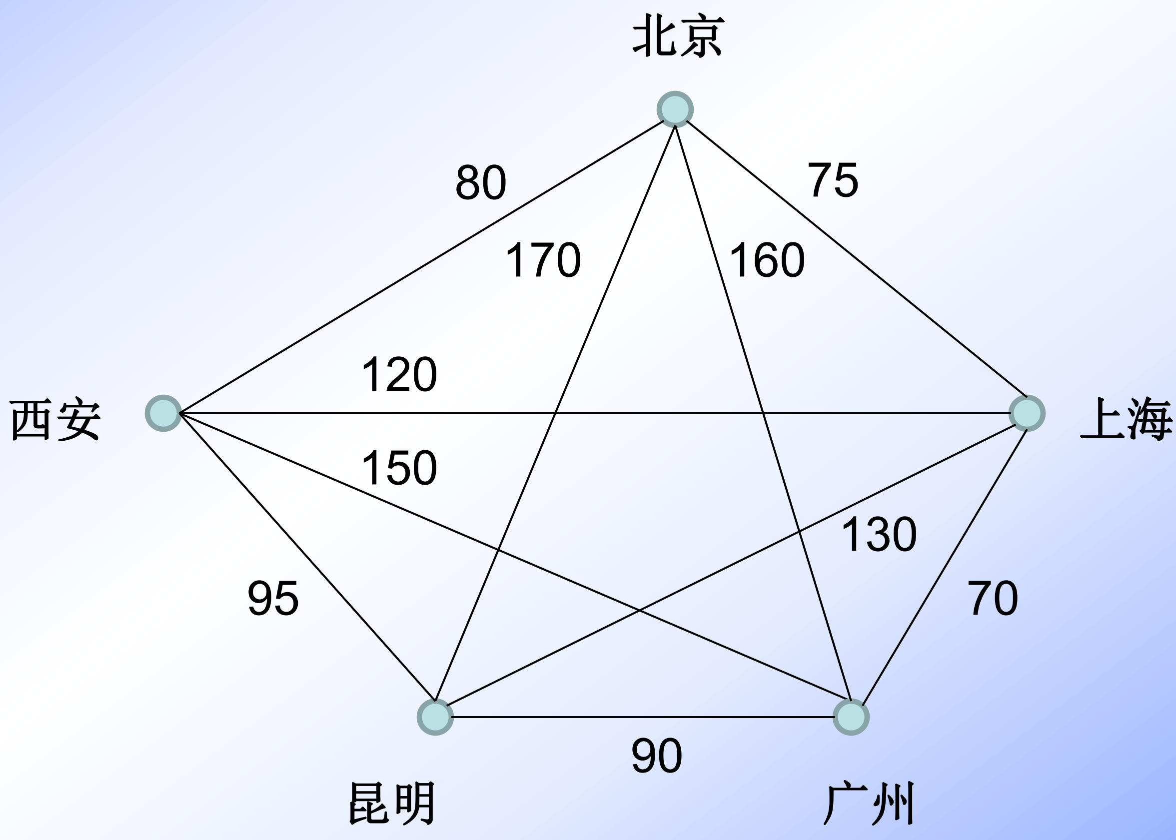
A*算法应用——搜索最短路径

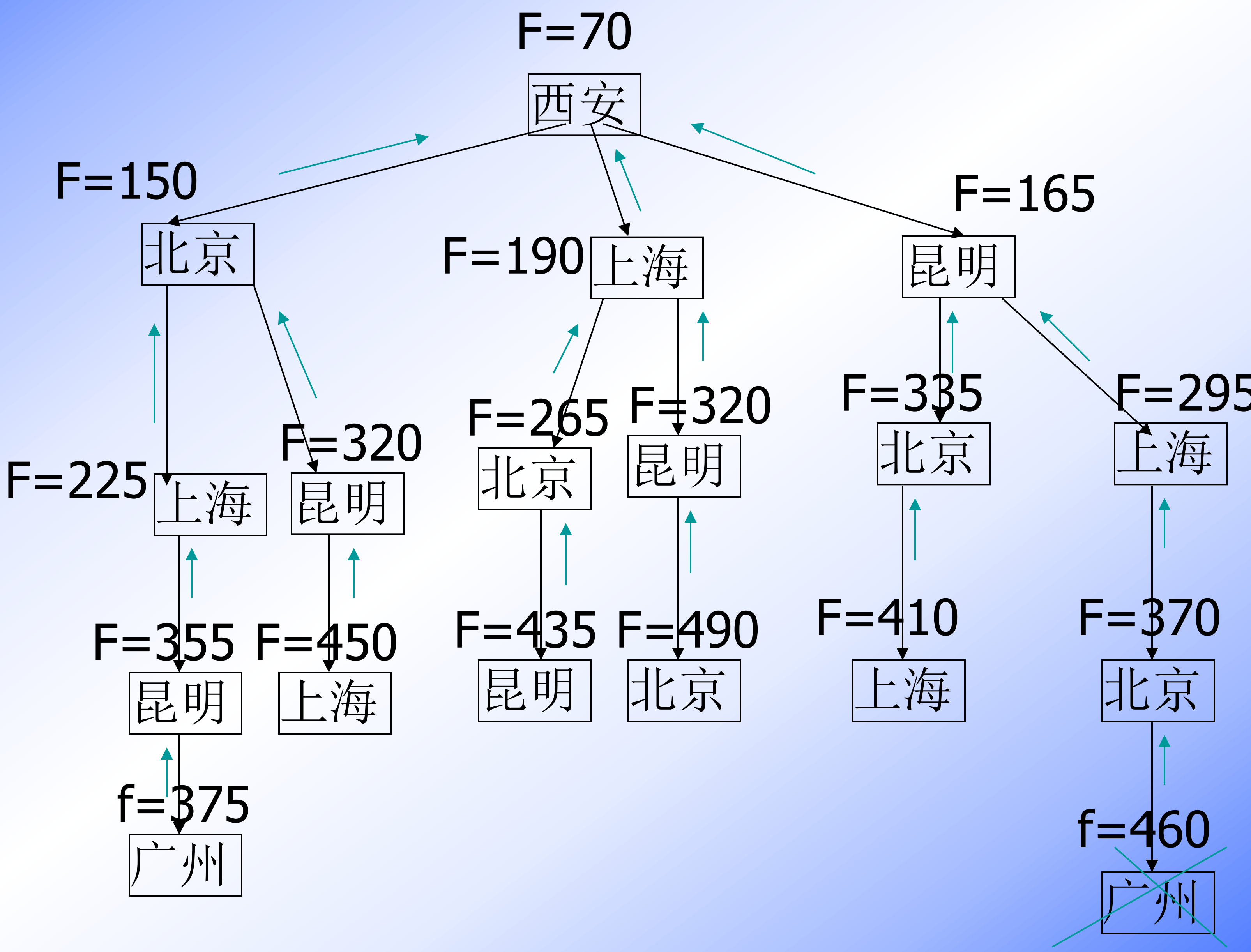
（二）

- 假定:
- h(x)=70;x属于集合{西安，北京，昆明，上海}
- H(广州)=0

- 西安→北京 →上海 →昆明 →广州，
- 代价为375

练习：五个城市之间的交通费用如图所示，若从西安出发，经过每个城市一次且一次，最后到达广州，请找出一条交通费用最少的路线。画出搜索树，并给出问题的解。

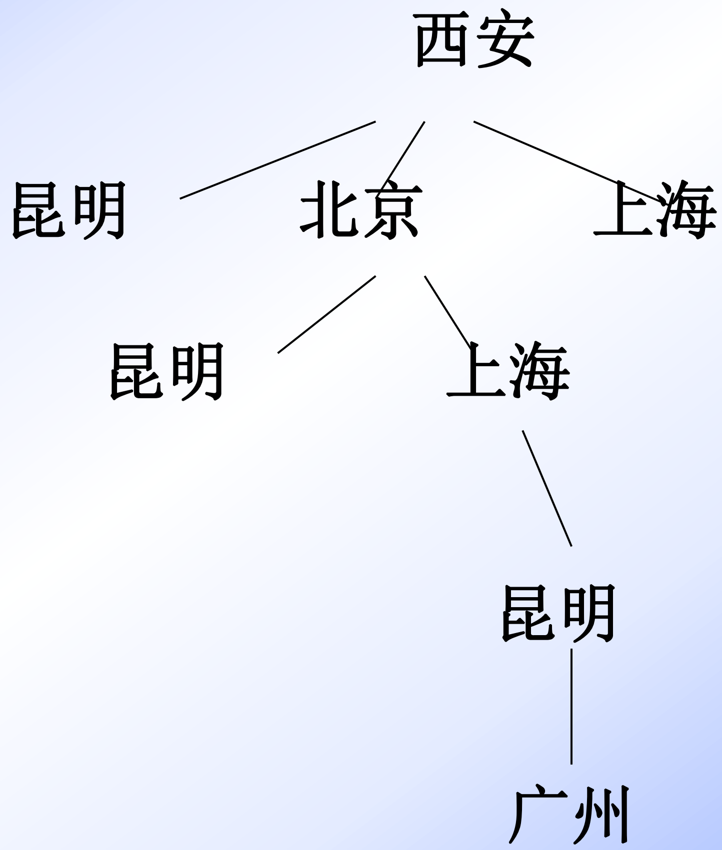




F=70
西安
F=150
北京
F=190
上海
F=165
昆明
F=225
上海
F=320
昆明
F=265
北京
F=320
昆明
F=335
北京
F=295
上海
F=355
昆明
F=450
上海
F=435
昆明
F=490
北京
F=410
上海
F=370
北京
f=375
广州
f=460
广州

路径：西安→北京 →上海 →昆明 →广州

最小代价为：375



A*算法应用——搜索最短路径

（三）

- 一个问题的状态空间图是客观存在的。
- 应用时先给出问题的状态空间图。
- 注：如果是80个城市，在有限时间内给出全部状态空间图则不可能。
- 同一问题启发函数可以有多种设计方法

对本章的旅行商问题，定义两个h函数（非零），使其都满足A*算法的条件。（右图为h(n)=0）

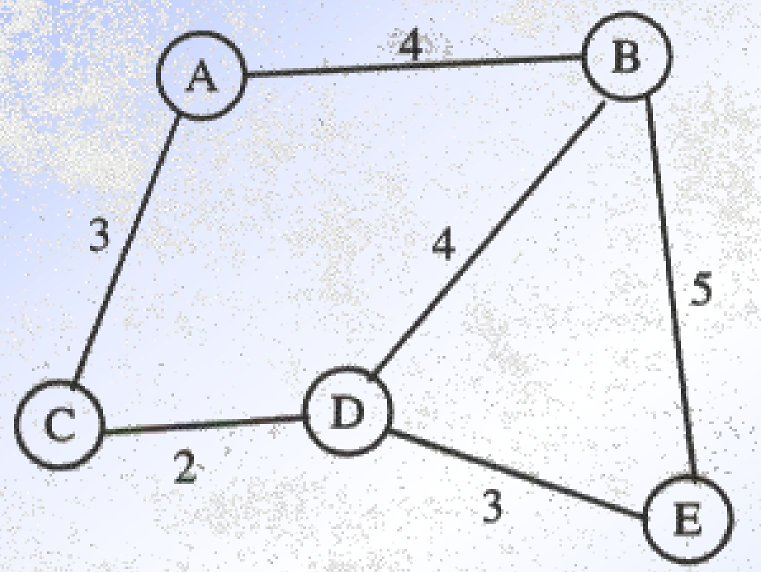


图 4.14　交通图

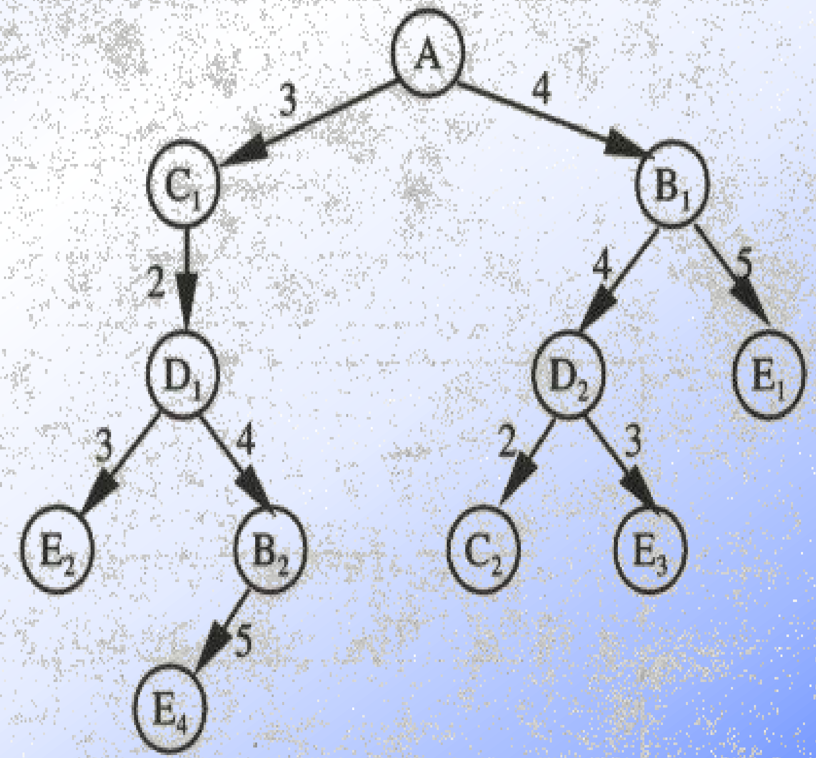


图 4.15　交通图的代价树

参考答案：

定义：$(1) h_1 = n \times k$

其中**n**是还未走过的城市数，**k**是还未走过的城市间距离的最小值。

$$(2) h_2 = \sum_{i=1}^{n} k_i$$

其中**n**是还未走过的城市数，**ki**是还未走过的城市间距离中**n**个最小的距离。

A*算法应用——八数码游戏

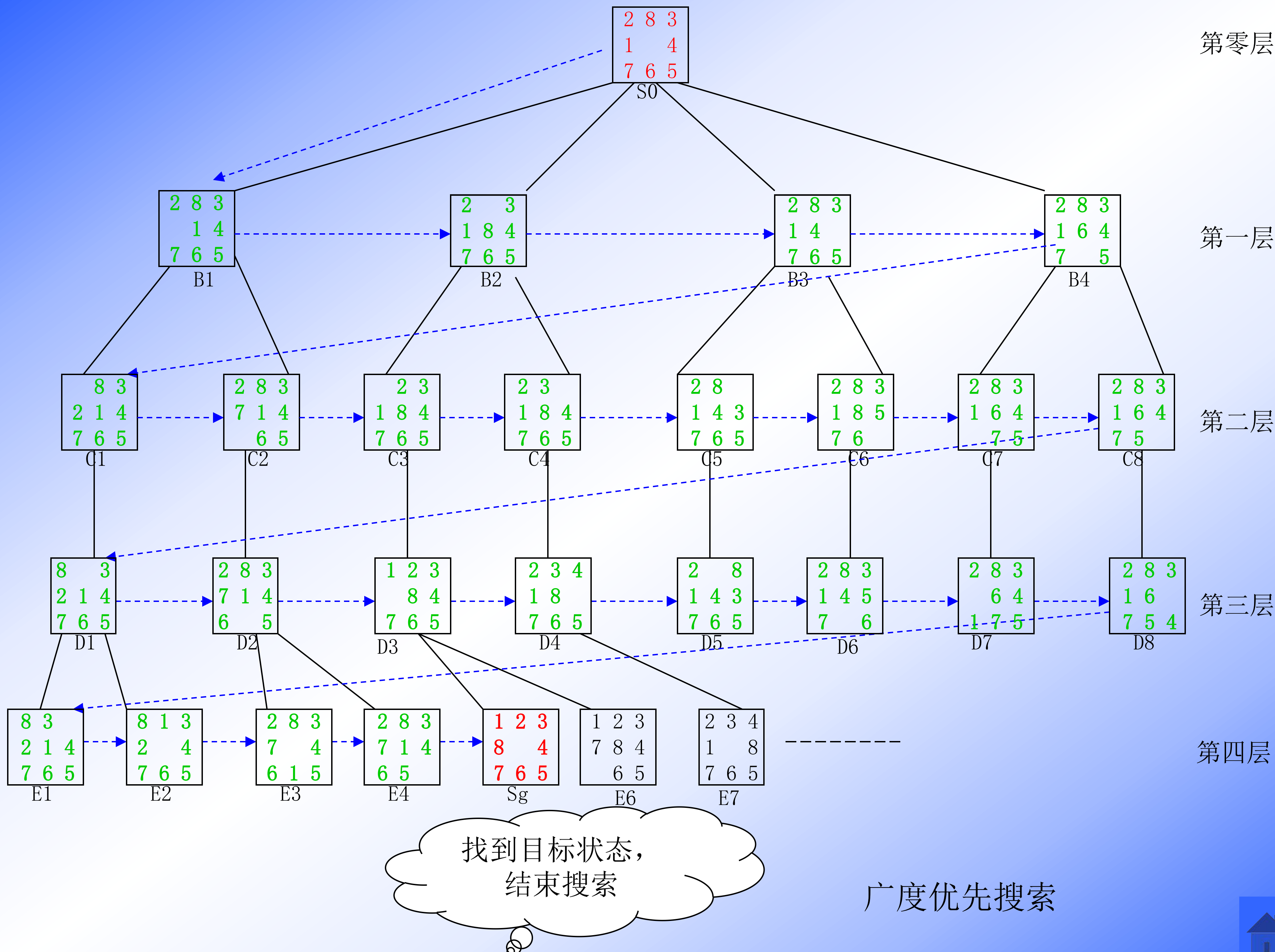
$$S_0 = \begin{bmatrix} 2 & 8 & 3 \\ 1 & & 4 \\ 7 & 6 & 5 \end{bmatrix} \qquad Sg = \begin{bmatrix} 1 & 2 & 3 \\ 8 & & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

- 在八数码问题中，启发式函数的选取如下：**g(x)**表示节点**x**在搜索树中的深度，**w(x)**表示节点**x**中不在目标状态中相应位置的数码个数，显然，**h(x)<=h*(x)**，因此它满足**A***算法的要求，所以找到的是最短路径。
- 还可以定义启发函数**h(n)=p(n)**为节点**n**的每一数码与其目标位置之间的距离总合。显然，相应的搜索算法也是**A***算法。

- p(n)比w(n)有更强的启发性信息，由h(n)=p(n)构造的启发式搜索树，比h(n)=w(n)构造的启发式搜索树节点树要少。

- 以下是三种不同启发函数下，扩展节点情况比较结果。

启发函数	h(n)=0	h(n)=w(n)	h(n)=p(n)
扩展节点	26	6	5
生成节点	46	12	10



第零层
S0
第一层
B1
B2
B3
B4
第二层
C1
C2
C3
C4
C5
C6
C7
C8
第三层
D1
D2
D3
D4
D5
D6
D7
D8
第四层
E1
E2
E3
E4
Sg
E6
E7
找到目标状态，
结束搜索

广度优先搜索

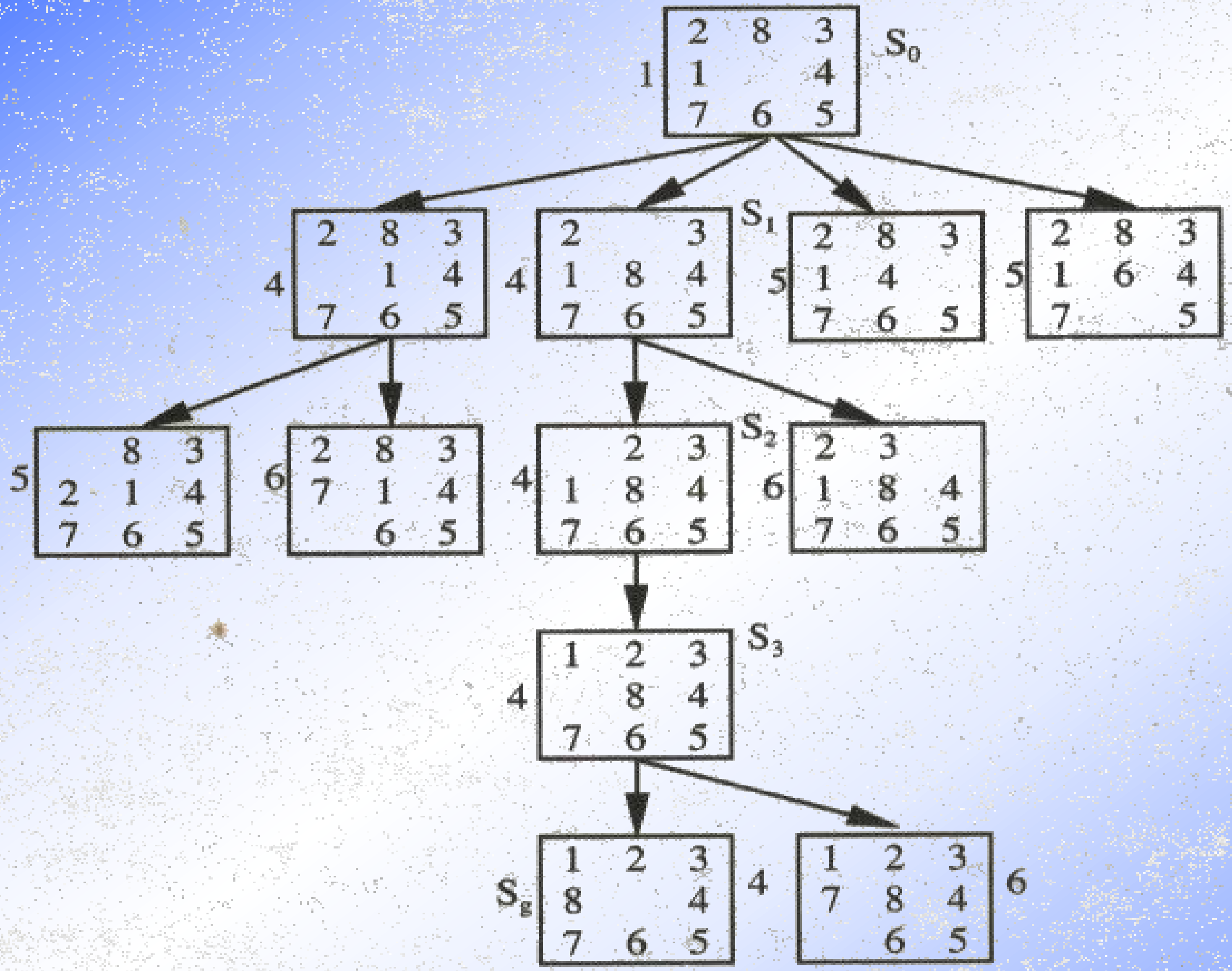
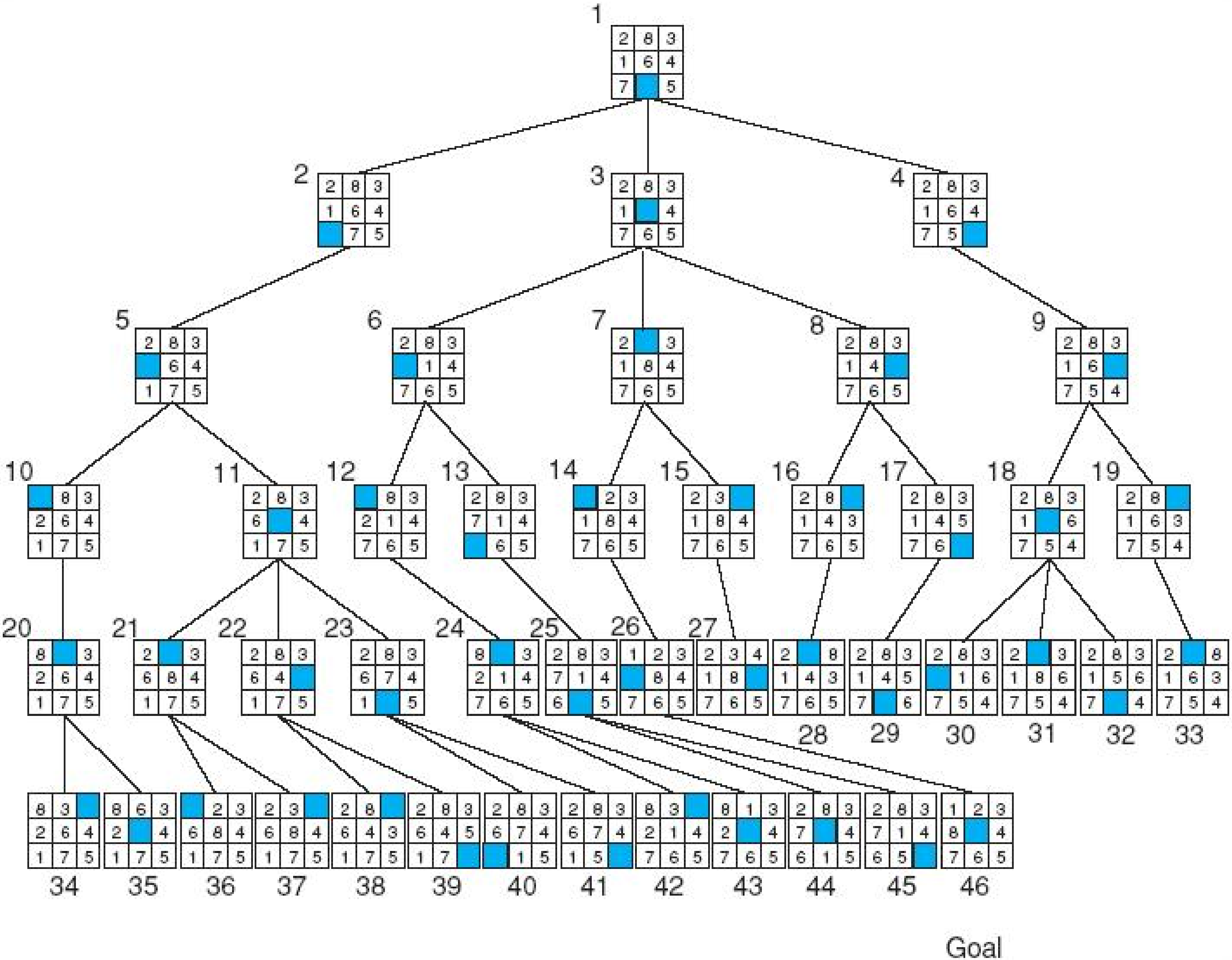
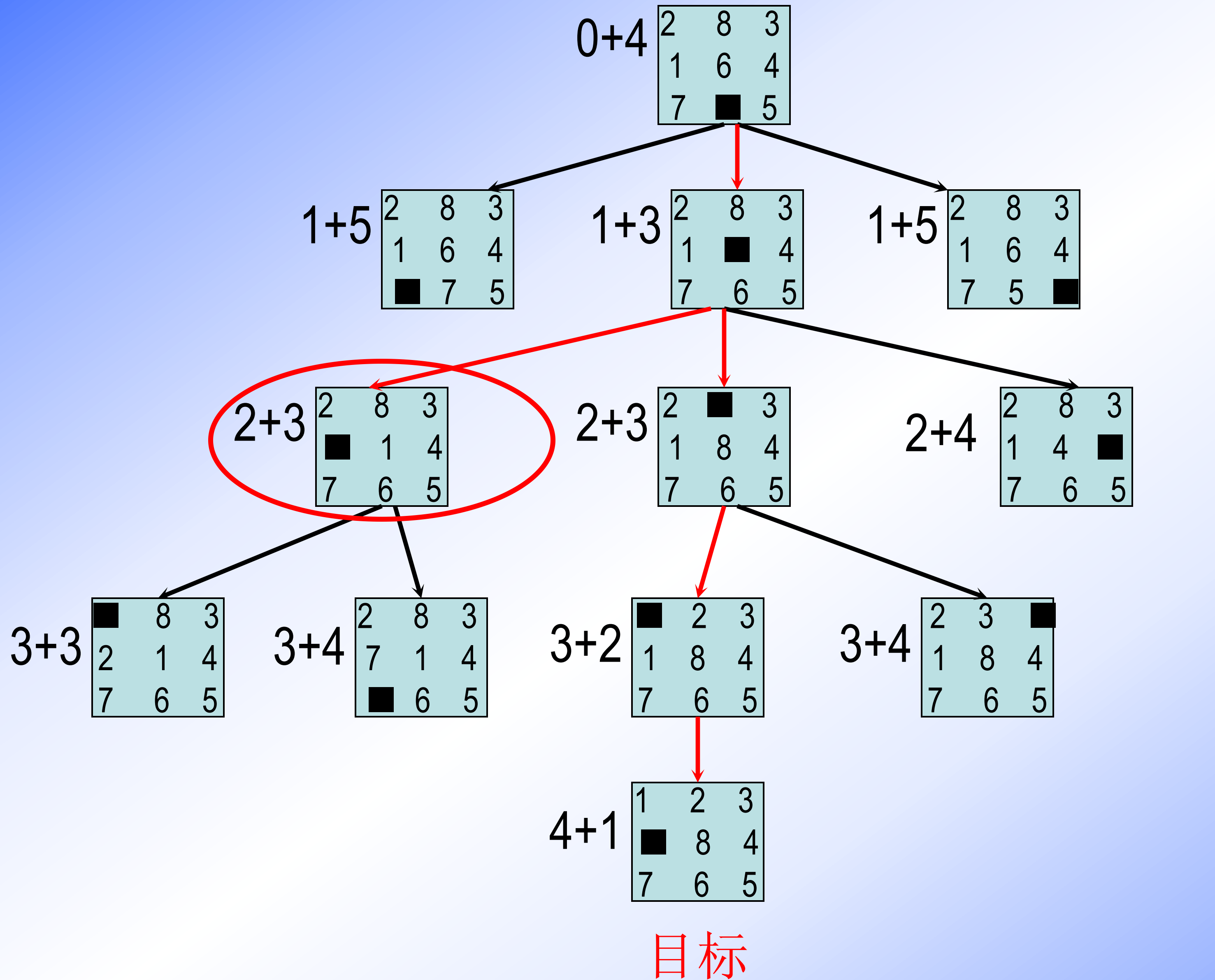


图 4.17　重排九宫问题的全局择优搜索树



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
Goal


0+4
2 8 3
1 6 4
7 5
1+5
2 8 3
1 6 4
7 5
1+3
2 8 3
1 4
7 6 5
1+5
2 8 3
1 6 4
7 5
2+3
2 8 3
1 4
7 6 5
2+3
2 3
1 8 4
7 6 5
2+4
2 8 3
1 4
7 6 5
3+3
8 3
2 1 4
7 6 5
3+4
2 8 3
7 1 4
6 5
3+2
2 3
1 8 4
7 6 5
3+4
2 3
1 8 4
7 6 5
4+1
1 2 3
8 4
7 6 5
目标

启发式搜索应用

- 给定4升和3升的水壶各一个。水壶上没有刻度。可以向水壶中加水。如何在4升的壶中准确地得到2升水？

提示：在这里用（x，y）表示4升壶里的水有x升和3升壶里的水有y升，n表示搜索空间中的任一节点，则给出下面的启发式函数：

h(n)=2，　如果$0<x<4$并且$0<y<3$

　　= 4，　如果$0<x<4$或者$0<y<3$

　　= 10，如果(1)x=0并且y=0

　　　　　或者(2)x=4并且y=3

　　= 8，　如果(1)x=0并且y=3

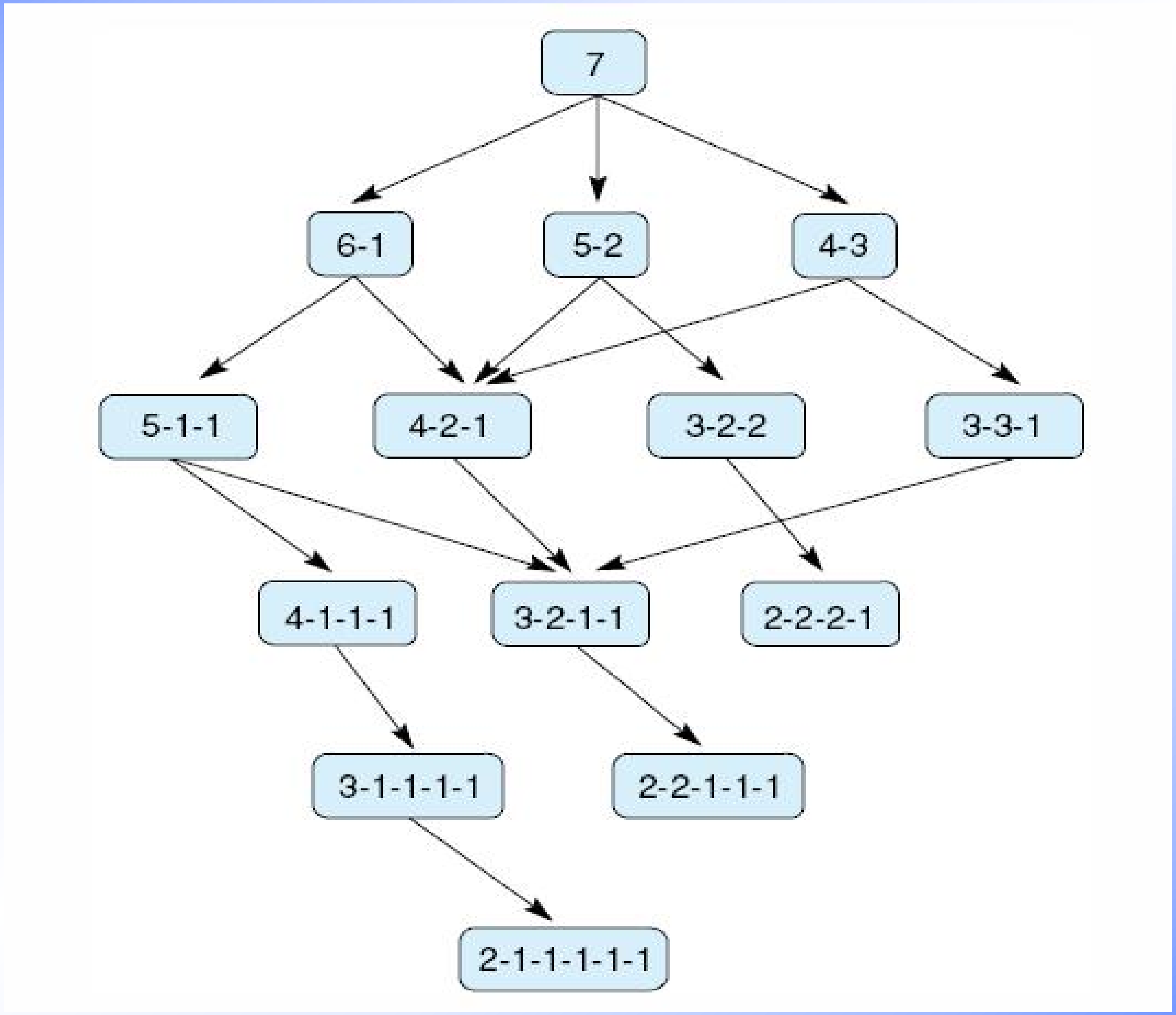
　　　　　或者(2)x=4并且y=0））

4.4 Using Heuristics in Games

4.4.1 The Minimax Procedure（极大极小过程） on Exhaustively Searchable Graphs

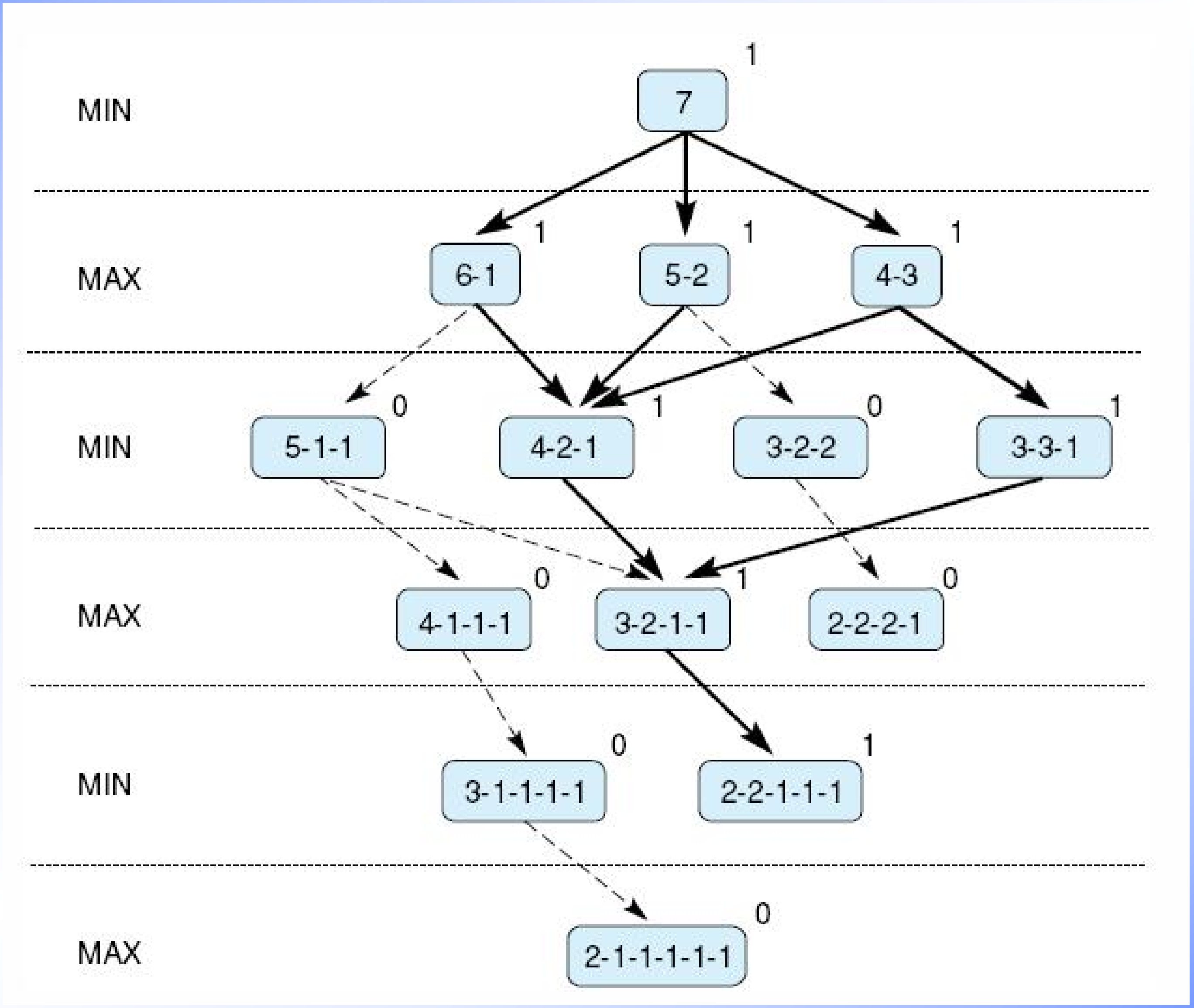
- we consider games whose state space is small enough to be exhaustively searched.
- We consider a variant（变体）of the game *nim*（余一棋，取物游戏）, at each move, the player must divide a pile of tokens（筹码） into nonempty piles of different sizes.
- The first player who can no longer make a move loses the game.

State space for a variant of nim



- Each leaf node is given a value of 1 or 0, depending on whether it is a win for MAX or for MIN.
- Minimax propagates（传播）these values up the graph through successive（连续的） parent nodes according to the rule:
  - if the parent is a MAX node（极大结点）, give it the maximum value among its children.
  - if the parent is a MIN node（极小结点）, give it the minimum value of its children.

Exhaustive minimax for the game of nim



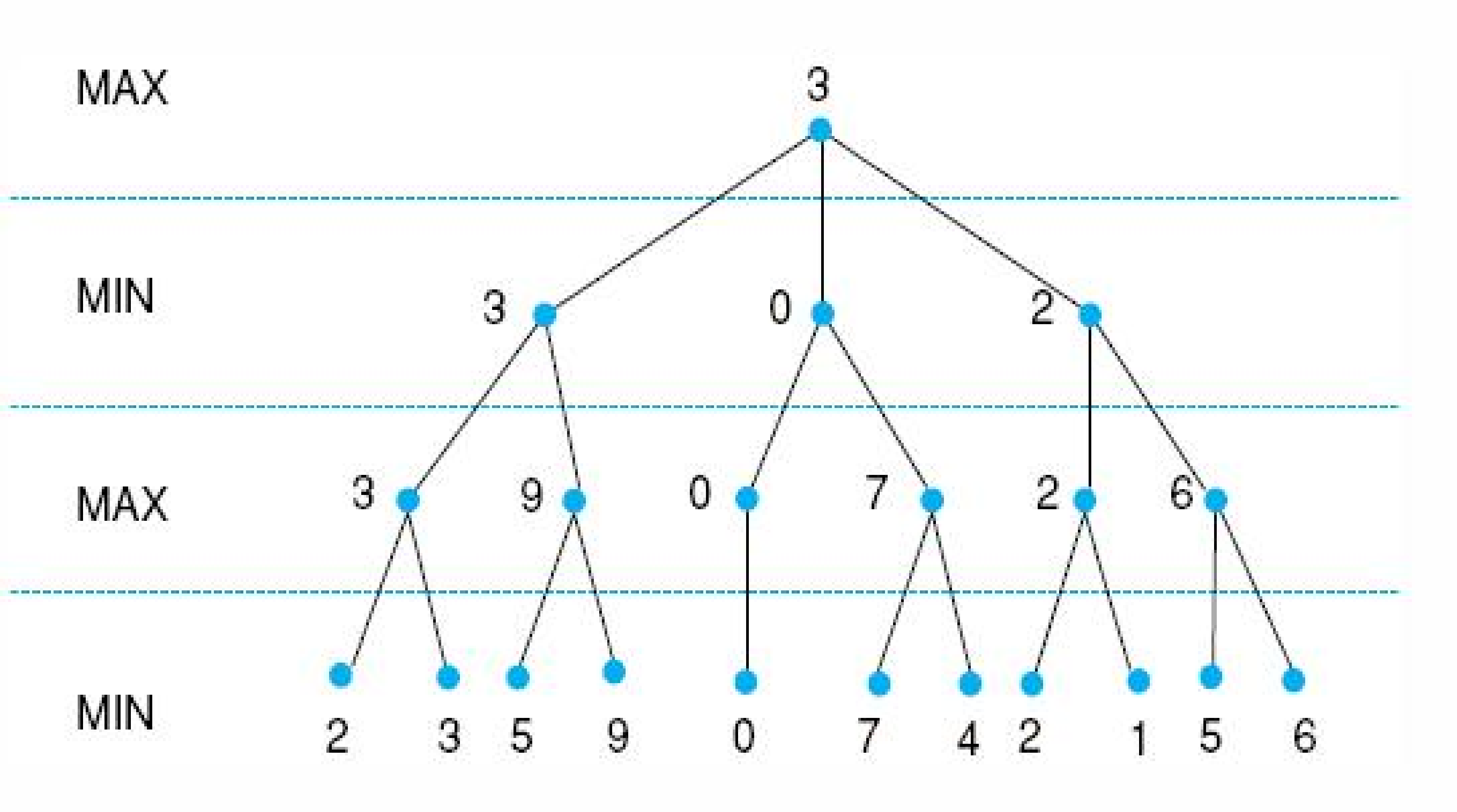
4.4.2 Minimaxing to Fixed Ply Depth

- In applying minimax to complicated games, it is impossible to expand the state space graph out to the final leaf nodes.
- Instead, the state space is searched to a predefined number of levels.
- This strategy is called an *n-ply look-ahead*（n 层预判）.
- Each node on level n (current leaf node) is given a value according to some heuristic evaluation function.
- The value is the estimation of the goodness of the state.

Minimax to a hypothetical state space

Leaf states show heuristic values

internal states show backed-up values



举例：下图给出了计算博弈树倒推值的示例

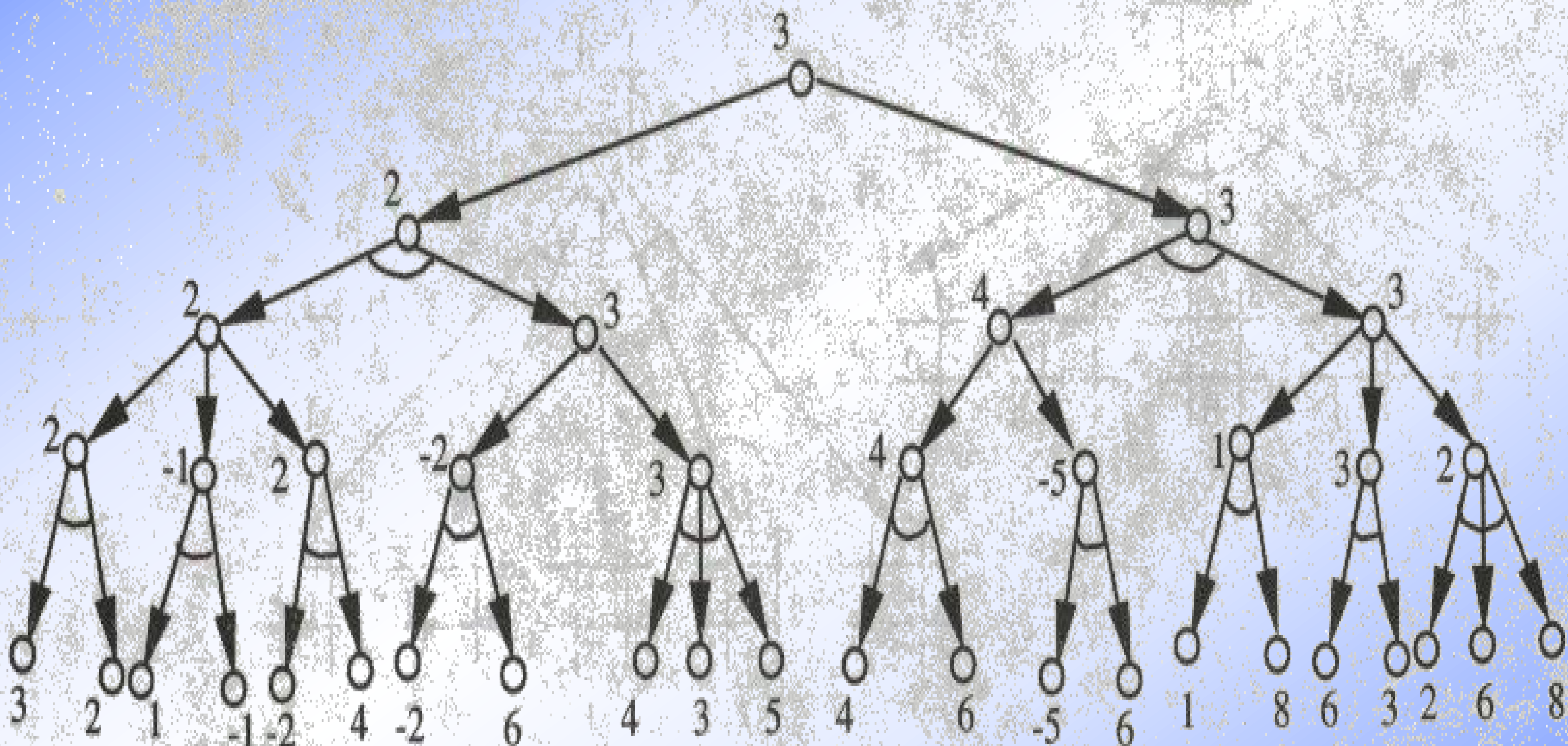
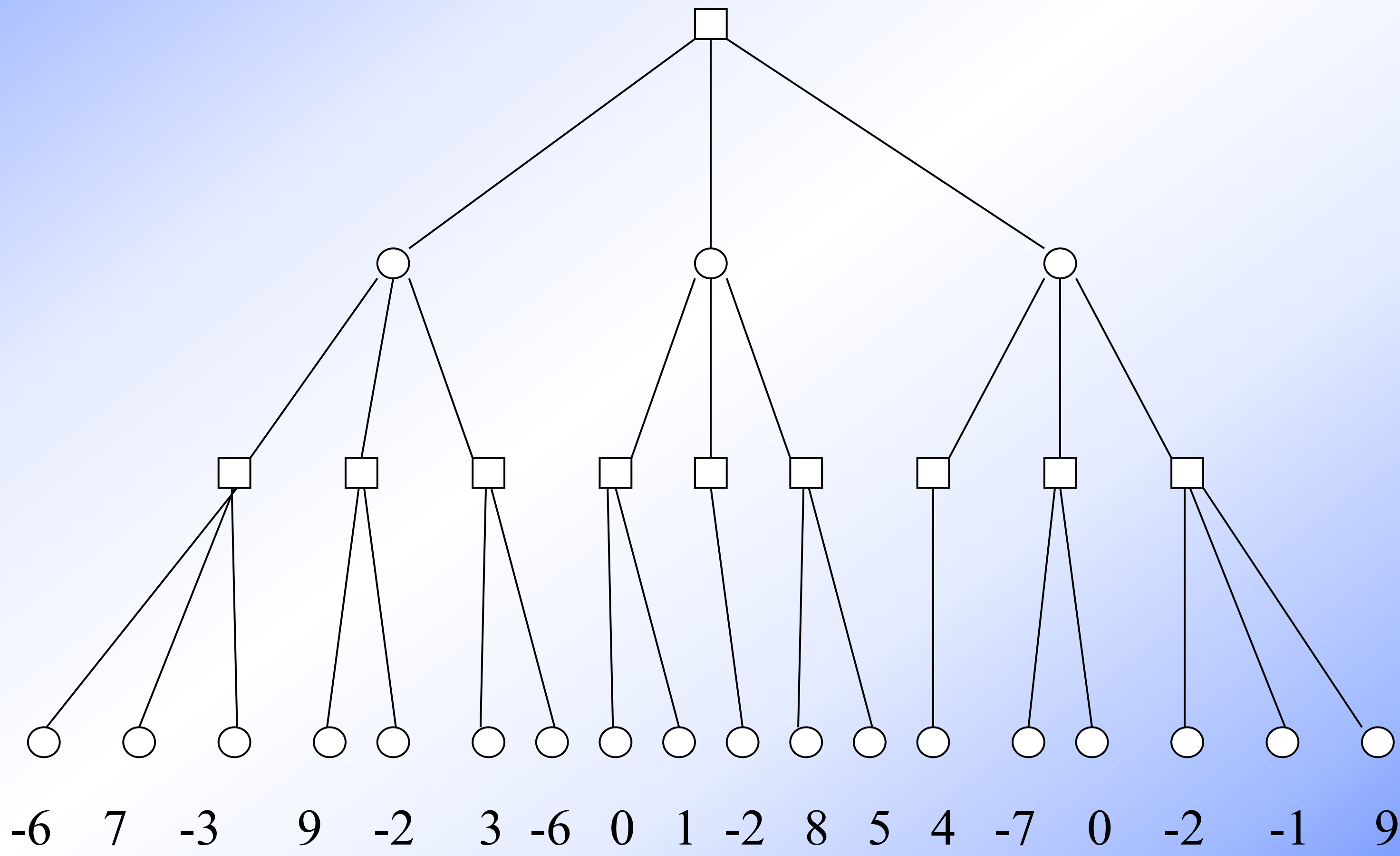


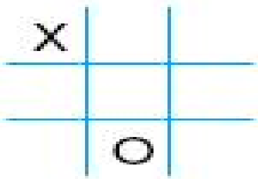
图 4.27 博弈树倒推值的计算

举例：已知博弈树如图所示，其中方形结点为极大结点，圆形结点为极小结点。如何找出当前最佳棋步。

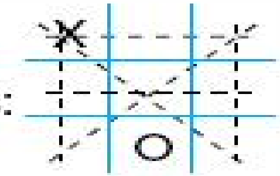


An application of minimax to tic-tac-toe（一字棋）

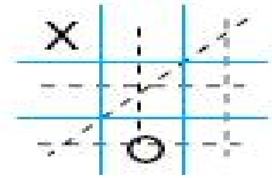
- A more complex heuristic is used here.
- The heuristic counts the total winning lines（所有取胜线路）open to MAX, and then subtracts the total number of winning lines open to MIN
- The search attempts to maximize this difference.
- If a state is a forced win（必胜）for MAX, it is evaluated as $+\infty$
- If a state is a forced win（必胜）for MIN, it is evaluated as $-\infty$



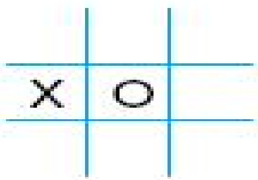
X has 6 possible win paths:



O has 5 possible wins:

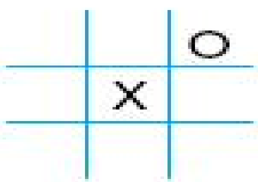


E(n) = 6 – 5 = 1



X has 4 possible win paths;
O has 6 possible wins

E(n) = 4 – 6 = –2



X has 5 possible win paths;
O has 4 possible wins

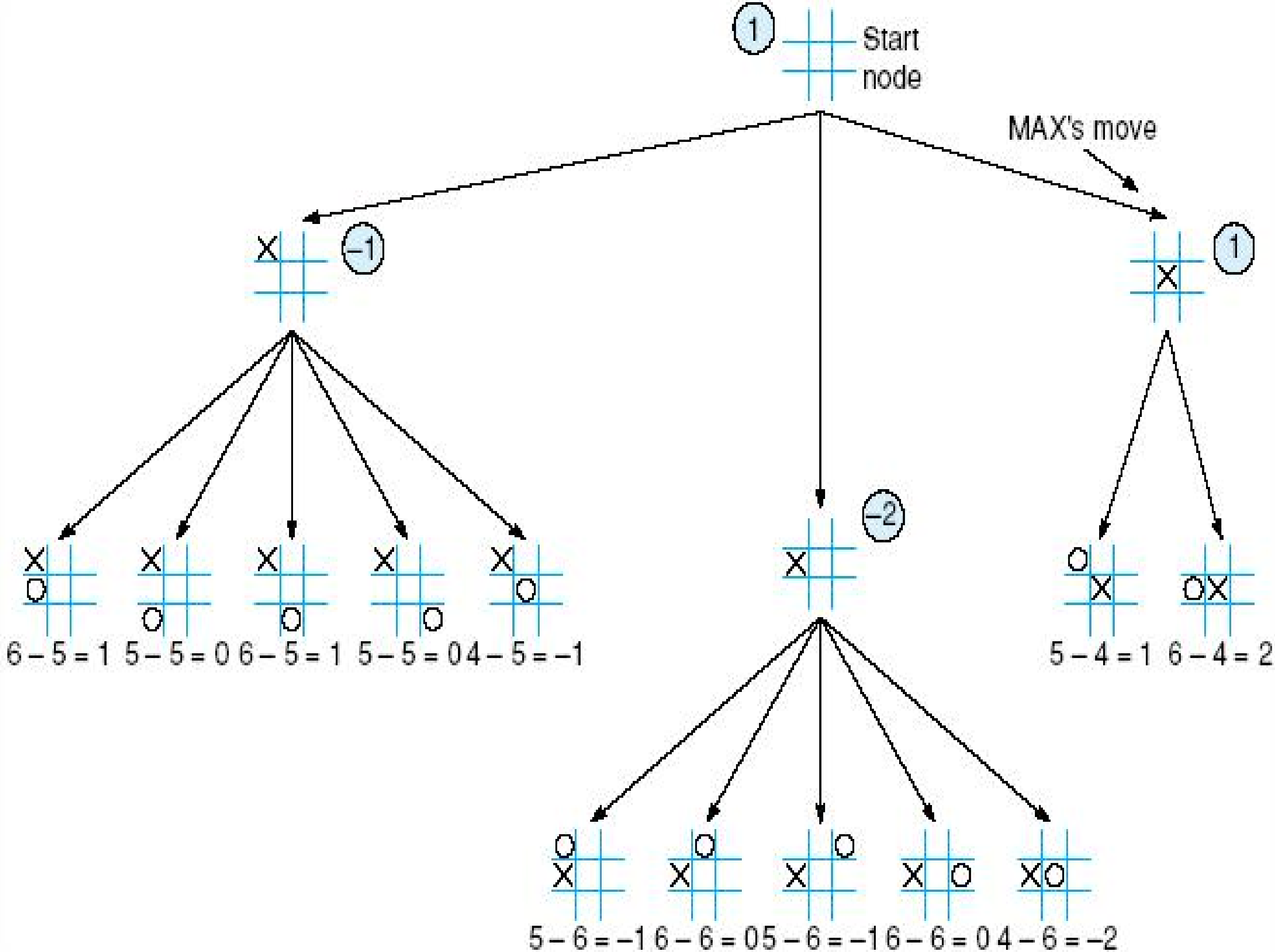
E(n) = 5 – 4 = 1

Heuristic is E(n) = M(n) – O(n)

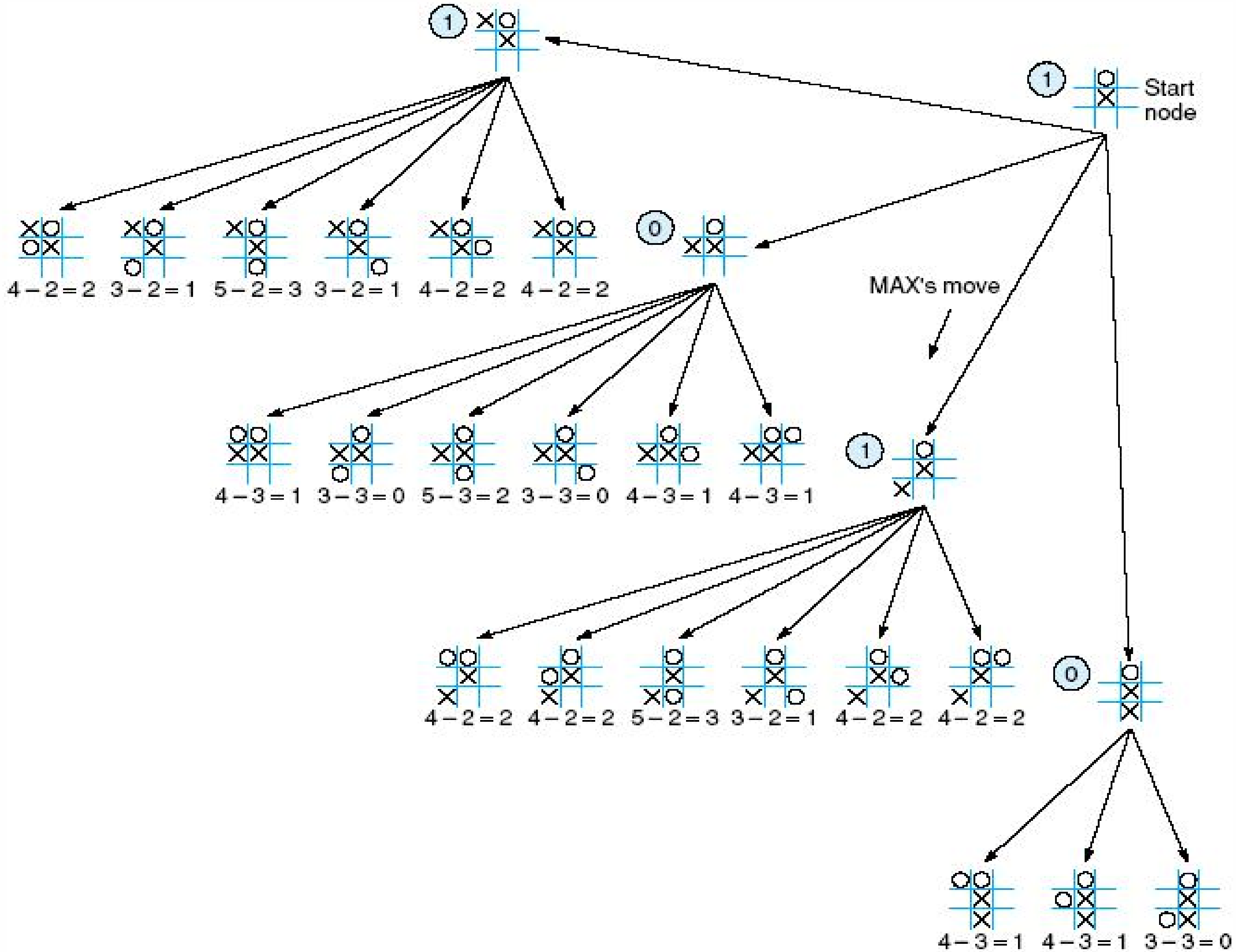
where M(n) is the total of My possible winning lines

O(n) is total of Opponent's possible winning lines

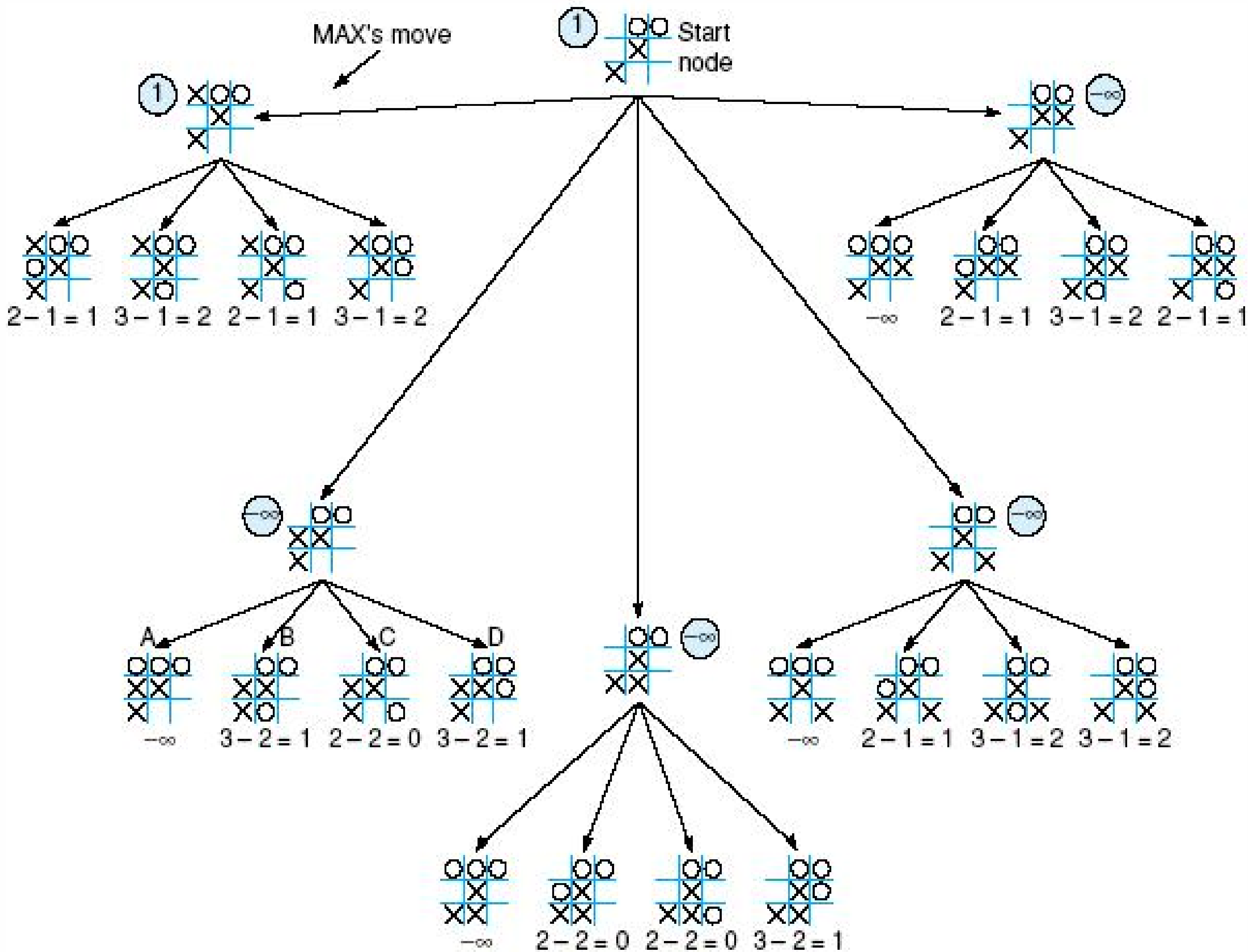
E(n) is the total Evaluation for state n



1
Start
node
MAX's move
−1
1
−2
6 − 5 = 1
5 − 5 = 0
6 − 5 = 1
5 − 5 = 0
4 − 5 = −1
5 − 4 = 1
6 − 4 = 2
5 − 6 = −1
6 − 6 = 0
5 − 6 = −1
6 − 6 = 0
4 − 6 = −2



Start node
MAX's move
1
1
0
1
0
4 − 2 = 2
3 − 2 = 1
5 − 2 = 3
3 − 2 = 1
4 − 2 = 2
4 − 2 = 2
4 − 3 = 1
3 − 3 = 0
5 − 3 = 2
3 − 3 = 0
4 − 3 = 1
4 − 3 = 1
4 − 2 = 2
4 − 2 = 2
5 − 2 = 3
3 − 2 = 1
4 − 2 = 2
4 − 2 = 2
4 − 3 = 1
4 − 3 = 1
3 − 3 = 0



MAX's move
1
Start node
1
−∞
2 − 1 = 1
3 − 1 = 2
2 − 1 = 1
3 − 1 = 2
−∞
2 − 1 = 1
3 − 1 = 2
2 − 1 = 1
−∞
−∞
A
B
C
D
−∞
3 − 2 = 1
2 − 2 = 0
3 − 2 = 1
−∞
−∞
2 − 1 = 1
3 − 1 = 2
3 − 1 = 2
−∞
2 − 2 = 0
2 − 2 = 0
3 − 2 = 1

4.4.3 The Alpha-Beta Procedure

- Straight minimax requires a two-pass（两阶段）analysis of the search space,
  - the first to descend（向下）to the ply depth（预定层深）and there apply the heuristic
  - the second to propagate values back up the tree.
- Minimax pursues all branches in the space, including many that could be ignored or pruned（剪枝）.
- Researchers developed a class of search techniques called *alpha-beta* pruning（alpha-beta 剪枝）.

- The idea for alpha-beta search : rather than searching the entire space to the ply depth （预定层深）, search proceeds in a depth-first fashion.

- Two values, called *alpha* and *beta*, are created during the search.

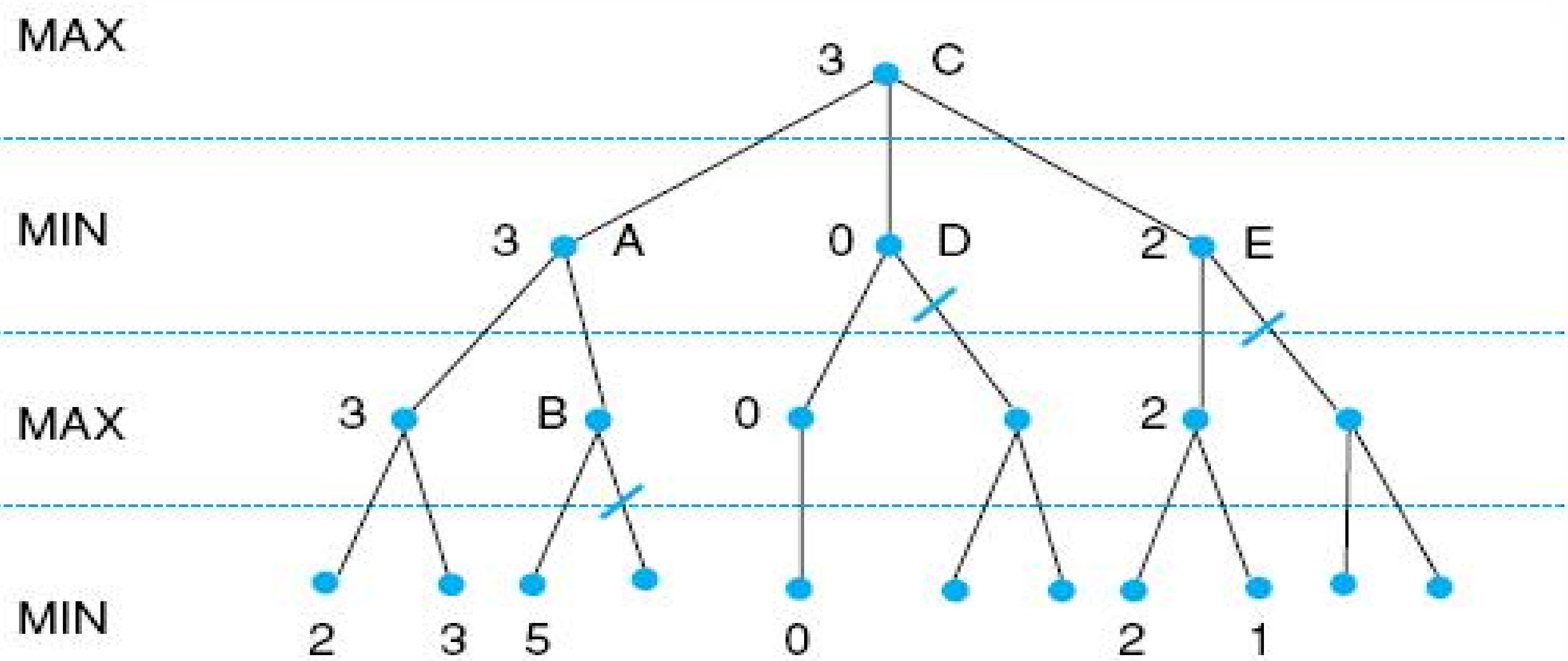
- The alpha value（alpha 值）, associated with MAX nodes, can never decrease（减小）, and the beta value（beta 值）, associated with MIN nodes, can never increase（增大）.

- Suppose a MAX node’s alpha value is 6. Then MAX need not consider any backed-up value（倒推值）less than or equal to 6 that is associated with any MIN node below it（子孙）.
- Similarly, if MIN has beta value 6, it dose not need to consider any MAX node below（子孙）that has a value of 6 or more.

- Two rules for terminating search（终止搜索）, based on alpha and beta values, are:

1. Search can be stopped below any MIN node having a beta value less than or equal to the alpha value of any of its MAX node ancestors（MAX 祖先）. ($\beta \leq \alpha$)

2. Search can be stopped below any MAX node having an alpha value greater than or equal to the beta value of any of its MIN node ancestors（MIN 祖先）. ($\alpha \geq \beta$)



A has $\beta = 3$ (A will be no larger than 3)

B is β pruned, since 5 > 3

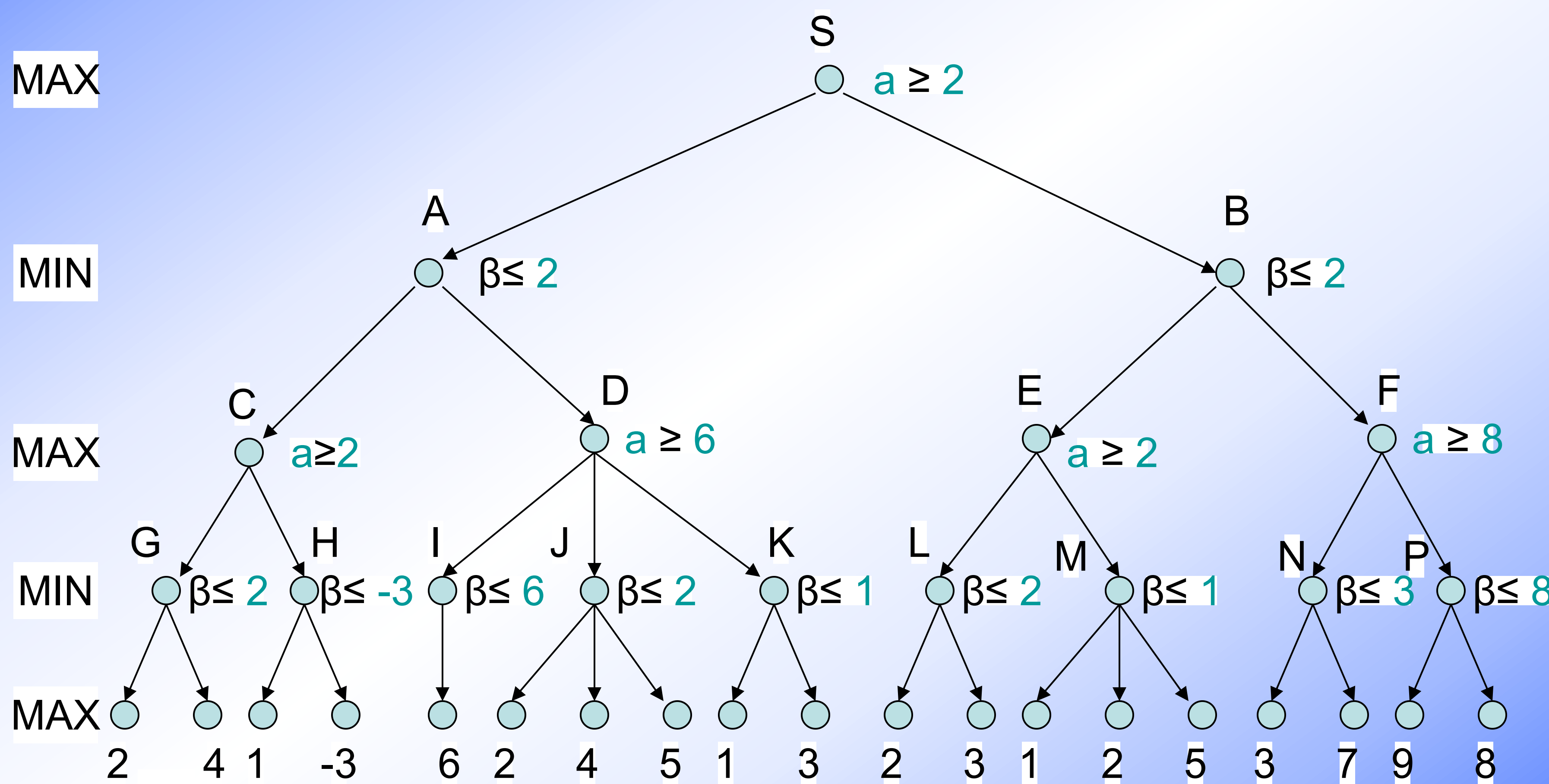
C has $\alpha = 3$ (C will be no smaller than 3)

D is α pruned, since 0 < 3

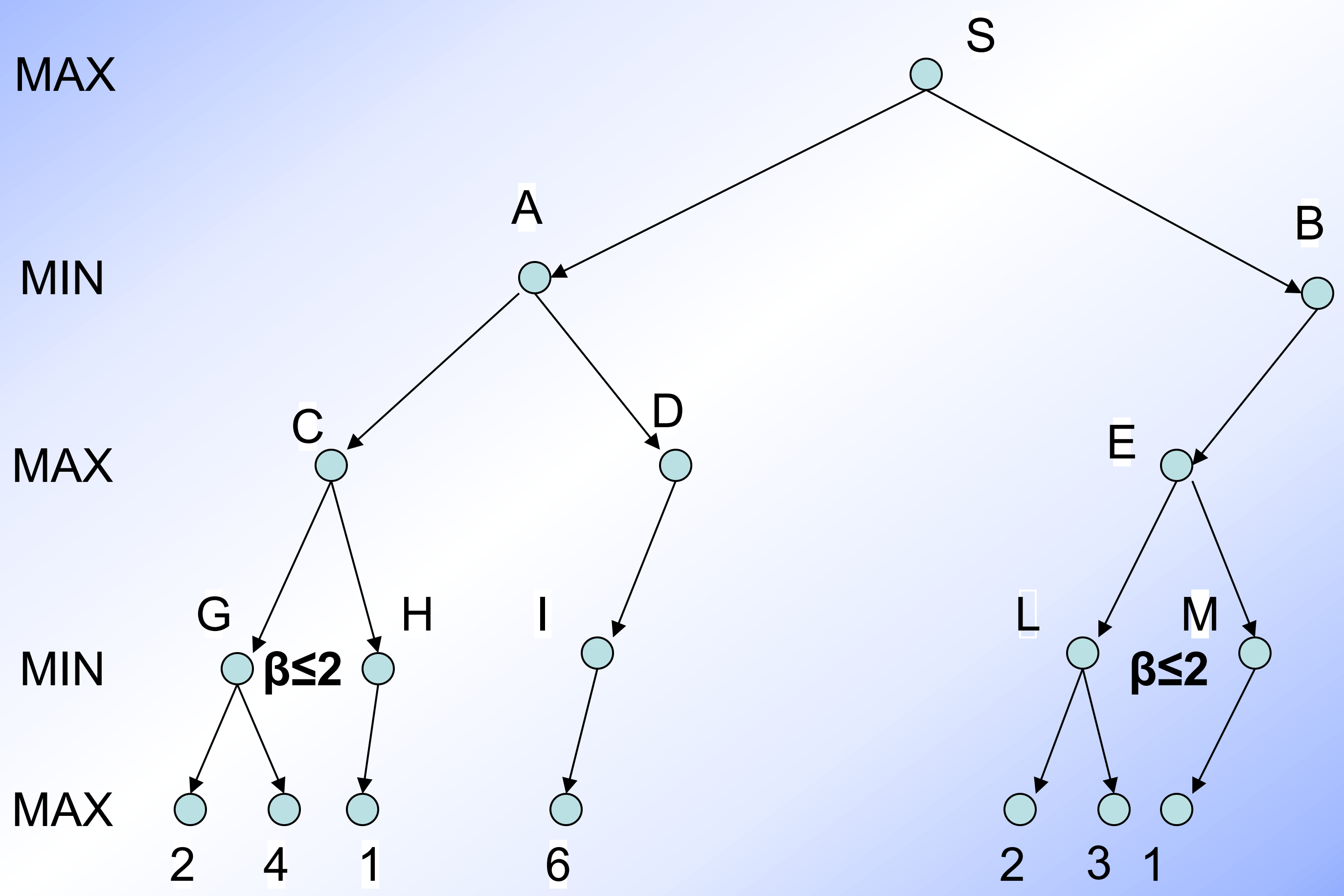
E is α pruned, since 2 < 3

C is 3

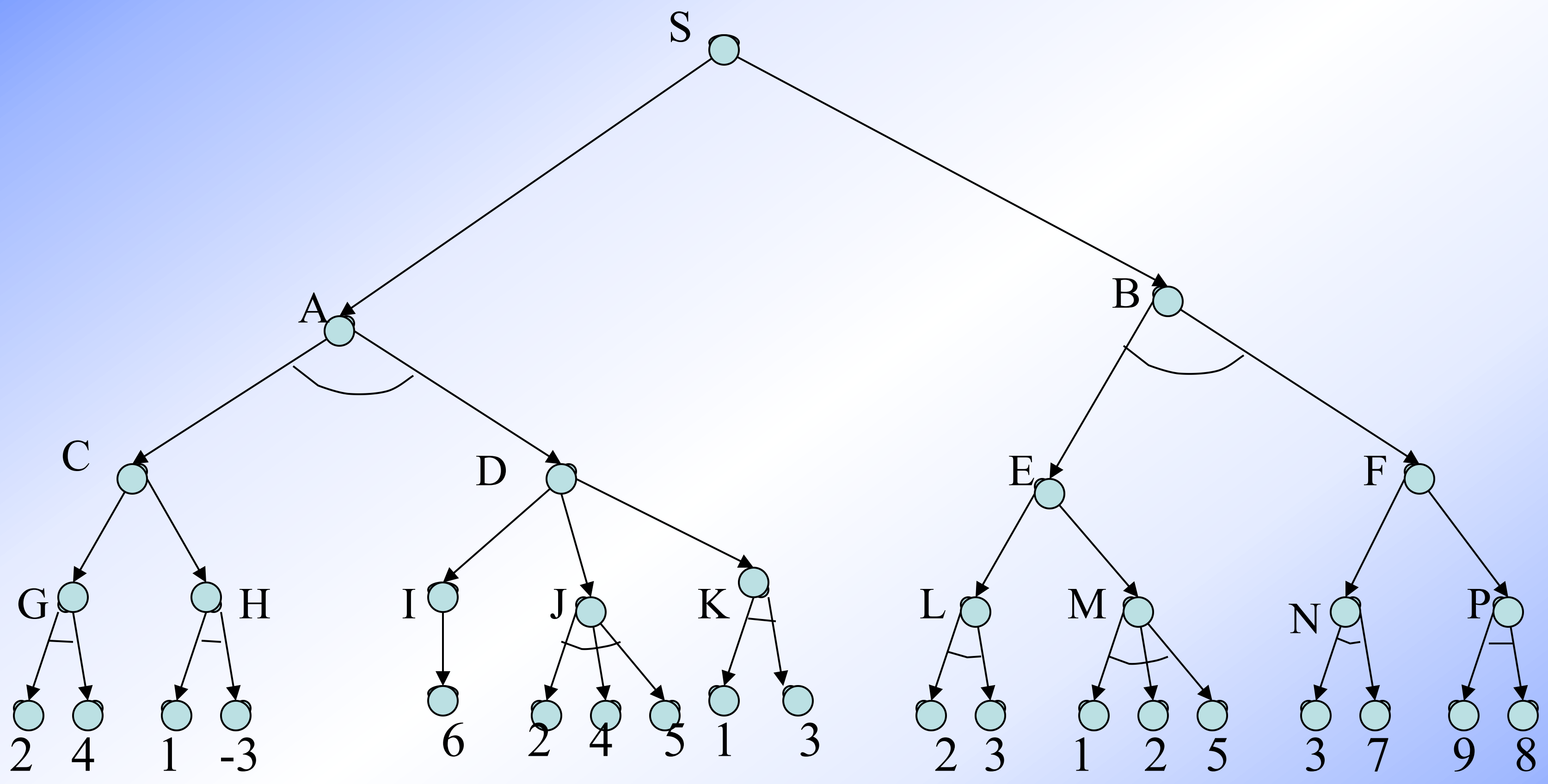
An example of alpha-beta search

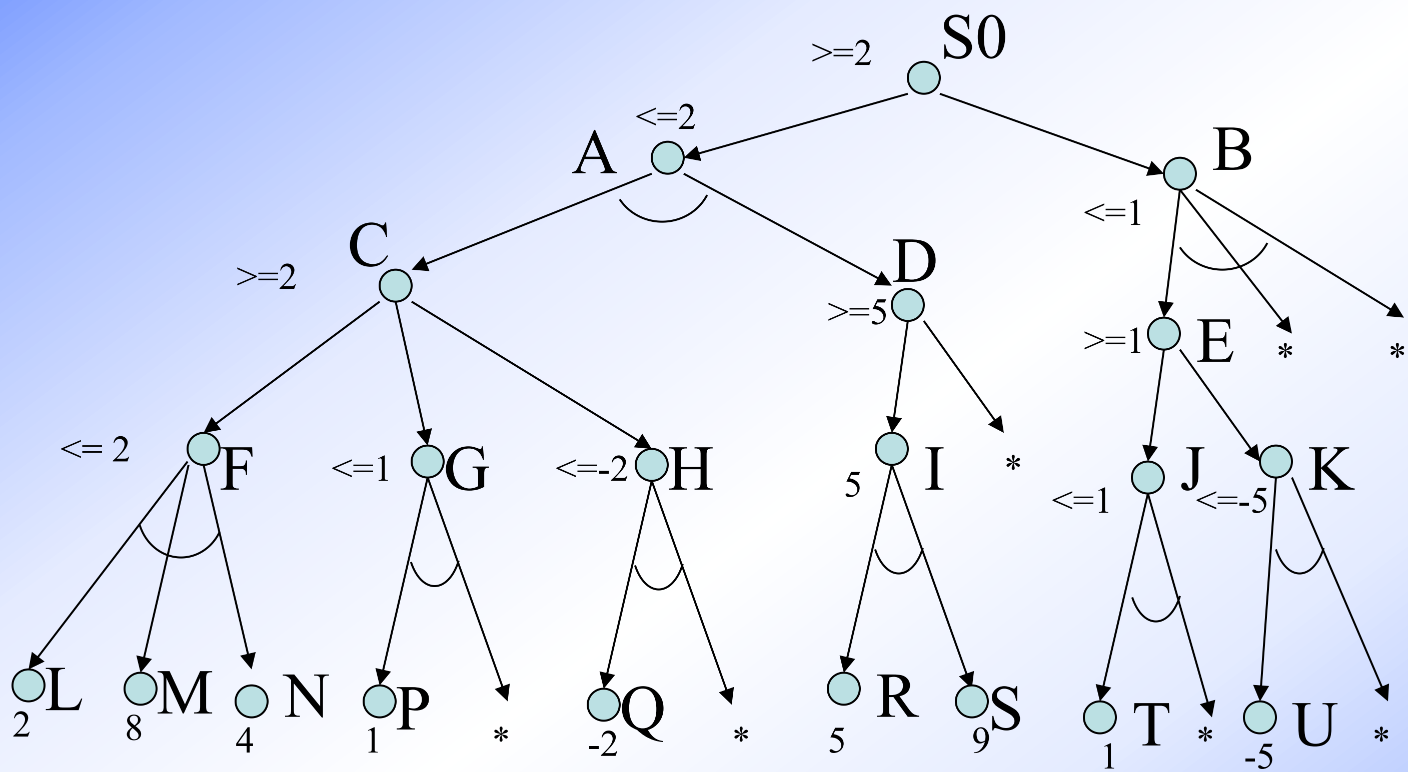


An example of alpha-beta search



计算各节点倒推值



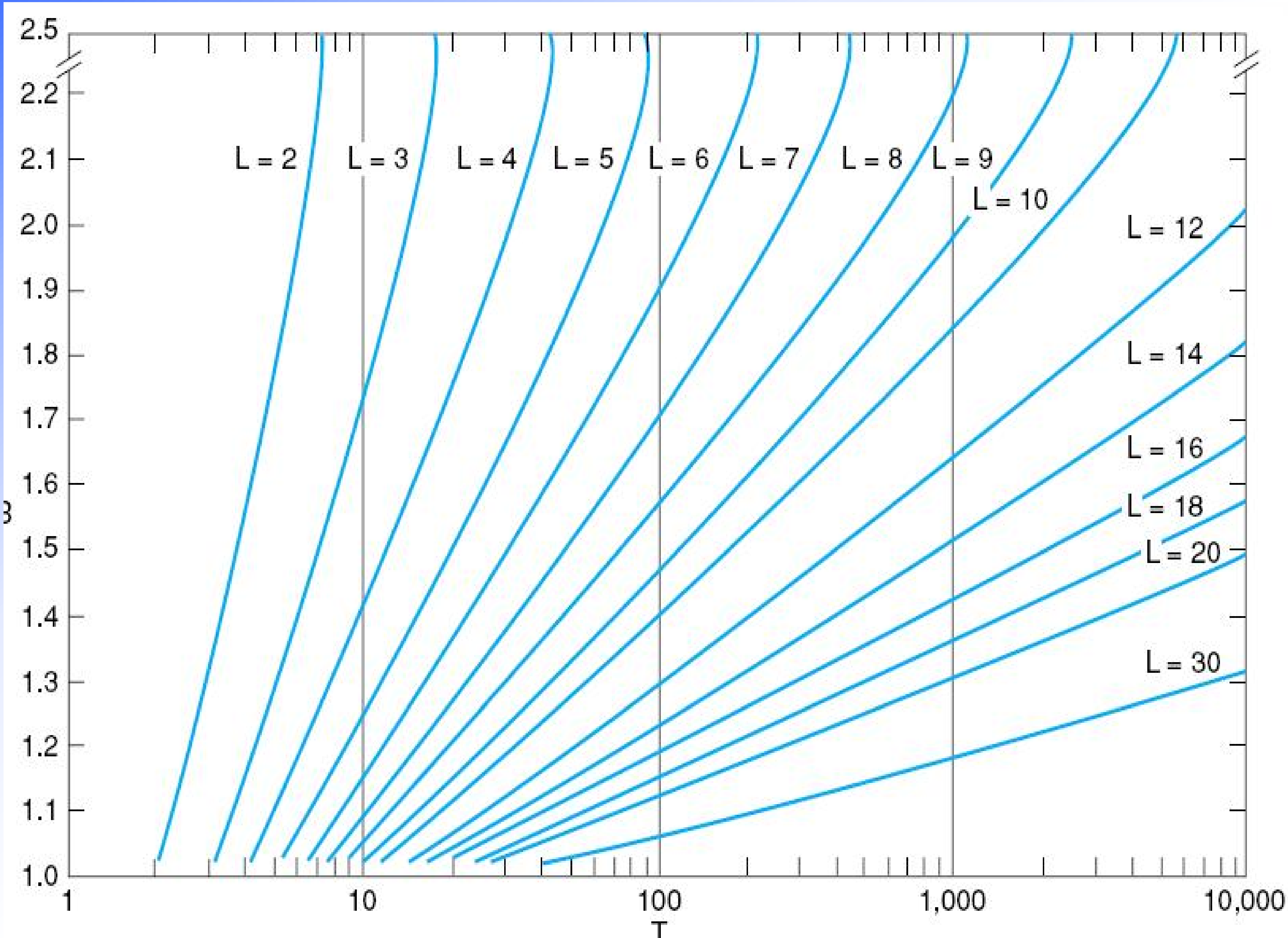


S0
>=2
A
<=2
B
<=1
C
>=2
D
>=5
E
>=1
*
*
F
<= 2
G
<=1
H
<=-2
I
5
*
J
<=1
K
<=-5
L
2
M
8
N
4
P
1
*
Q
-2
*
R
5
S
9
T
1
*
U
-5
*

4.5 Complexity issues

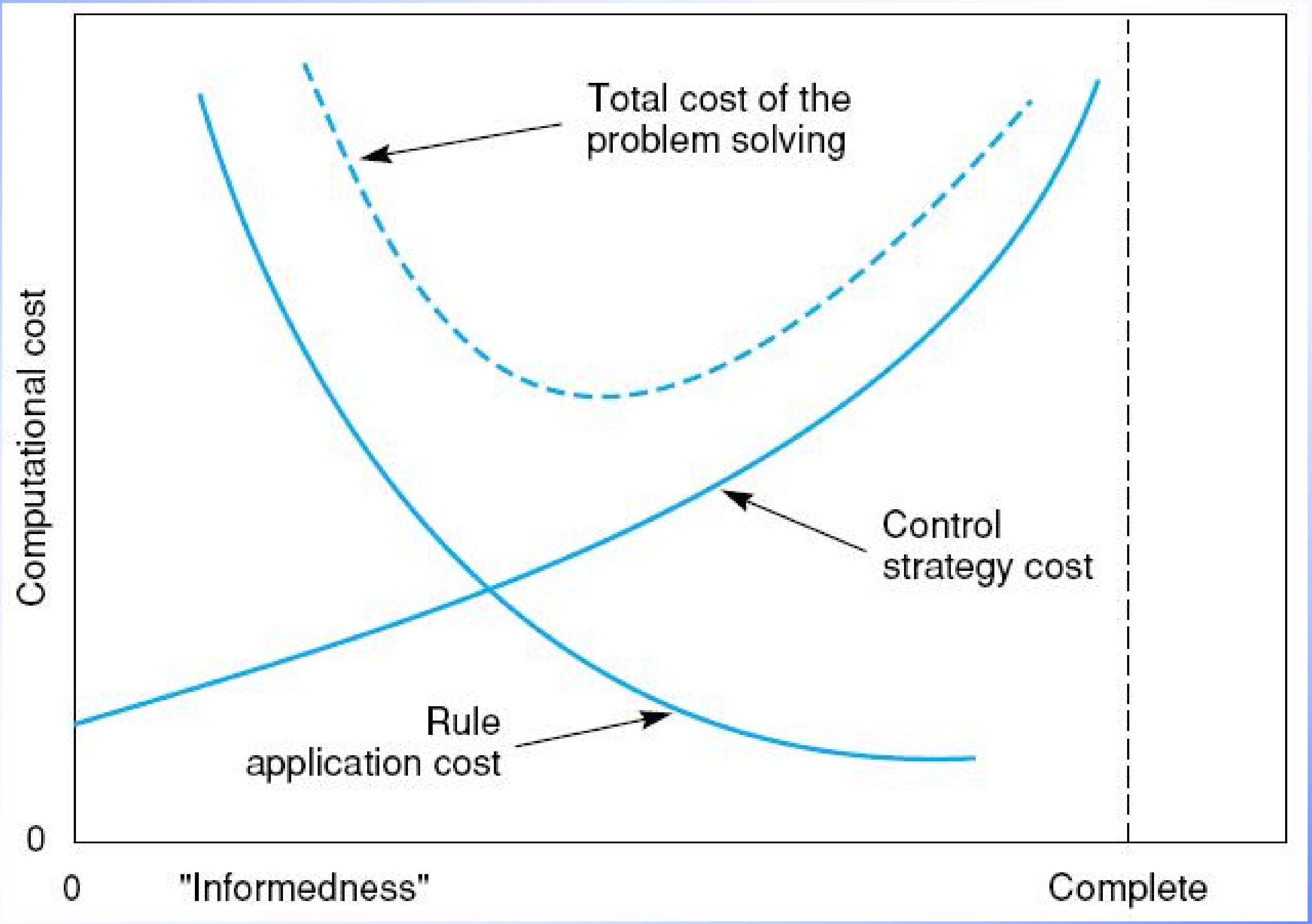
- The **branching factor** : B
- **Depth** or search length : L
- **Total number** of states **generated** : T
- $T = B + B^2 + B^3 + \ldots\ldots + B^L$
- The relating equation is :

$$T = B (B^L - 1) / (B - 1)$$



2.5
2.2
2.1
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
B
L = 2
L = 3
L = 4
L = 5
L = 6
L = 7
L = 8
L = 9
L = 10
L = 12
L = 14
L = 16
L = 18
L = 20
L = 30
1
10
100
1,000
10,000
T

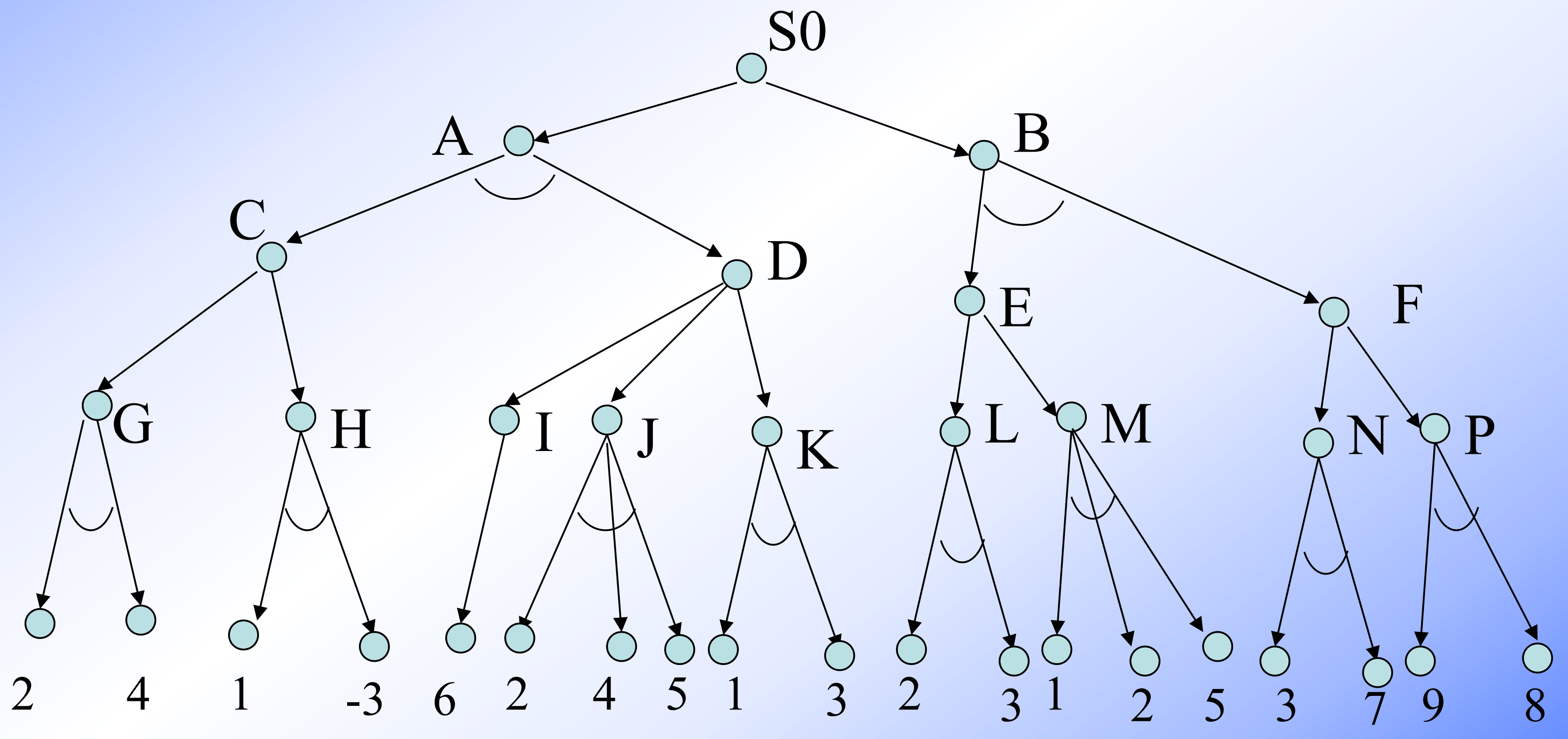
Cost of searching and cost of computing heuristic evaluation

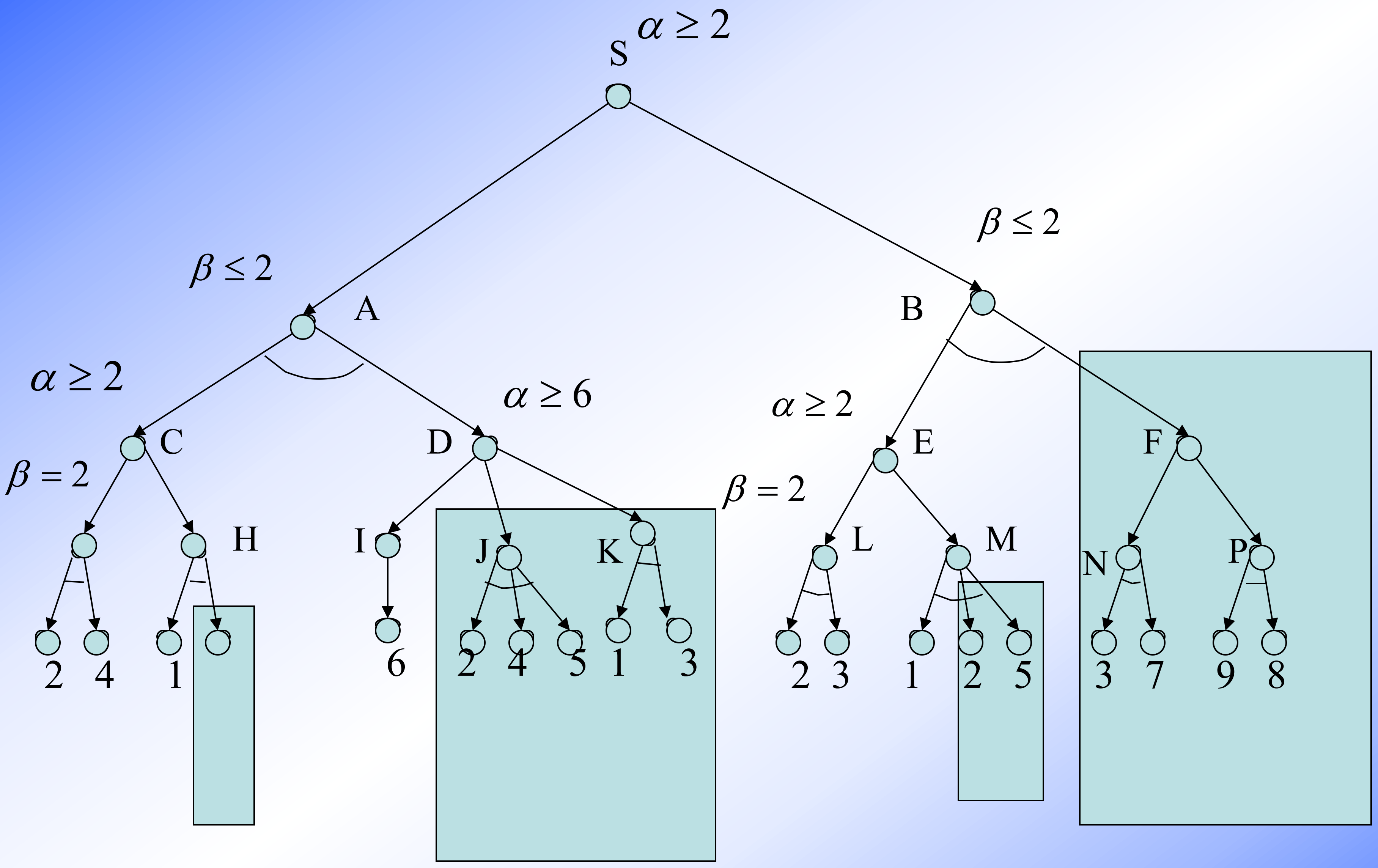


习题1.如图所示的博弈树,其中末一行的数字为假设的估值,

1)计算各节点的倒推值。

2)利用α-β剪枝技术剪去不必要的分枝。



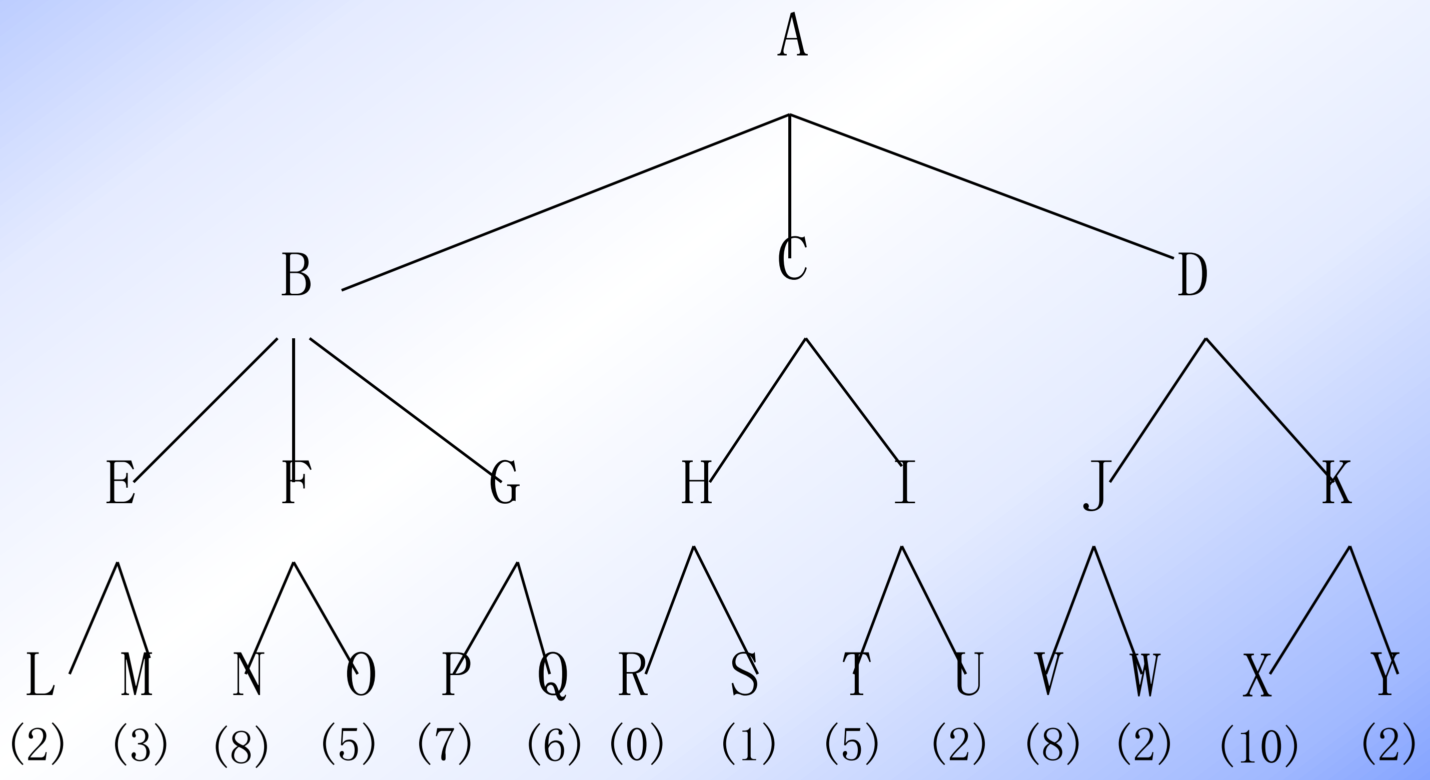


$\alpha \geq 2$
S
$\beta \leq 2$
A
$\beta \leq 2$
B
$\alpha \geq 2$
C
$\beta = 2$
D
$\alpha \geq 6$
$\alpha \geq 2$
E
F
$\beta = 2$
H
I
J
K
L
M
N
P
2 4 1 6 2 4 5 1 3 2 3 1 2 5 3 7 9 8

习题2.假设第一个博弈者为MAX，如图所示。

（1）第一个博弈者将选择什么移动？

（2）假如采用α－β剪枝算法，哪些节点无须检验（假设节点按从左到右的顺序检验）？



习题3. A*算法12分（共3小题）

1. A*算法采用怎样的启发函数？(3分)

2. A*算法有哪些特性？（3分）

3. 请用A*算法求解重排九宫问题。（6分）

问题的初始状态S_0和目标状态S_g分别为：

$$S_0 = \begin{bmatrix} 2 & 8 & 3 \\ 1 & & 4 \\ 7 & 6 & 5 \end{bmatrix} \qquad Sg = \begin{bmatrix} 1 & 2 & 3 \\ 8 & & 4 \\ 7 & 6 & 5 \end{bmatrix}$$

[解答]：

1. 如果有序搜索算法的估价函数f满足如下限制，则称为A*算法：节点的估价函数为： $f(x)=g(x)+h(x)$ ，其中$h(x)\leqslant h^*(x)$。

2. A*算法的有关特性：可纳性、最优性、h(x) 的单调性限制。

3. 用A*算法重新求解重排九宫问题：

评价函数：$f(x)= d(x) + h(x)$

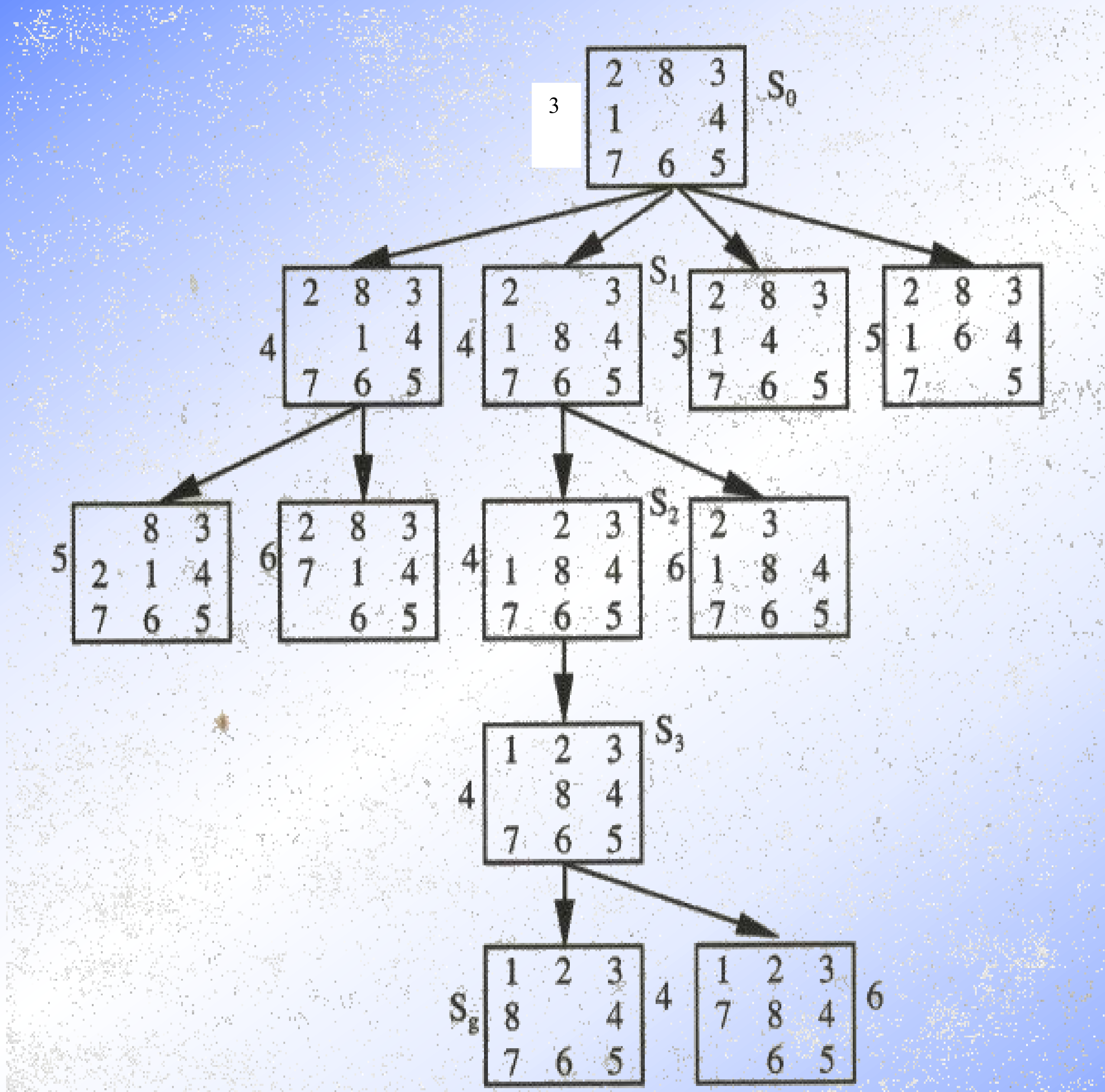
其中：h(x)为节点x与目标节点S_g 不相同的数码个数，d(x)为节点x的深度。

此评价函数f(x)满足A*算法的限制。

其搜索的状态空间图如下图。

则路径为：$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_g$

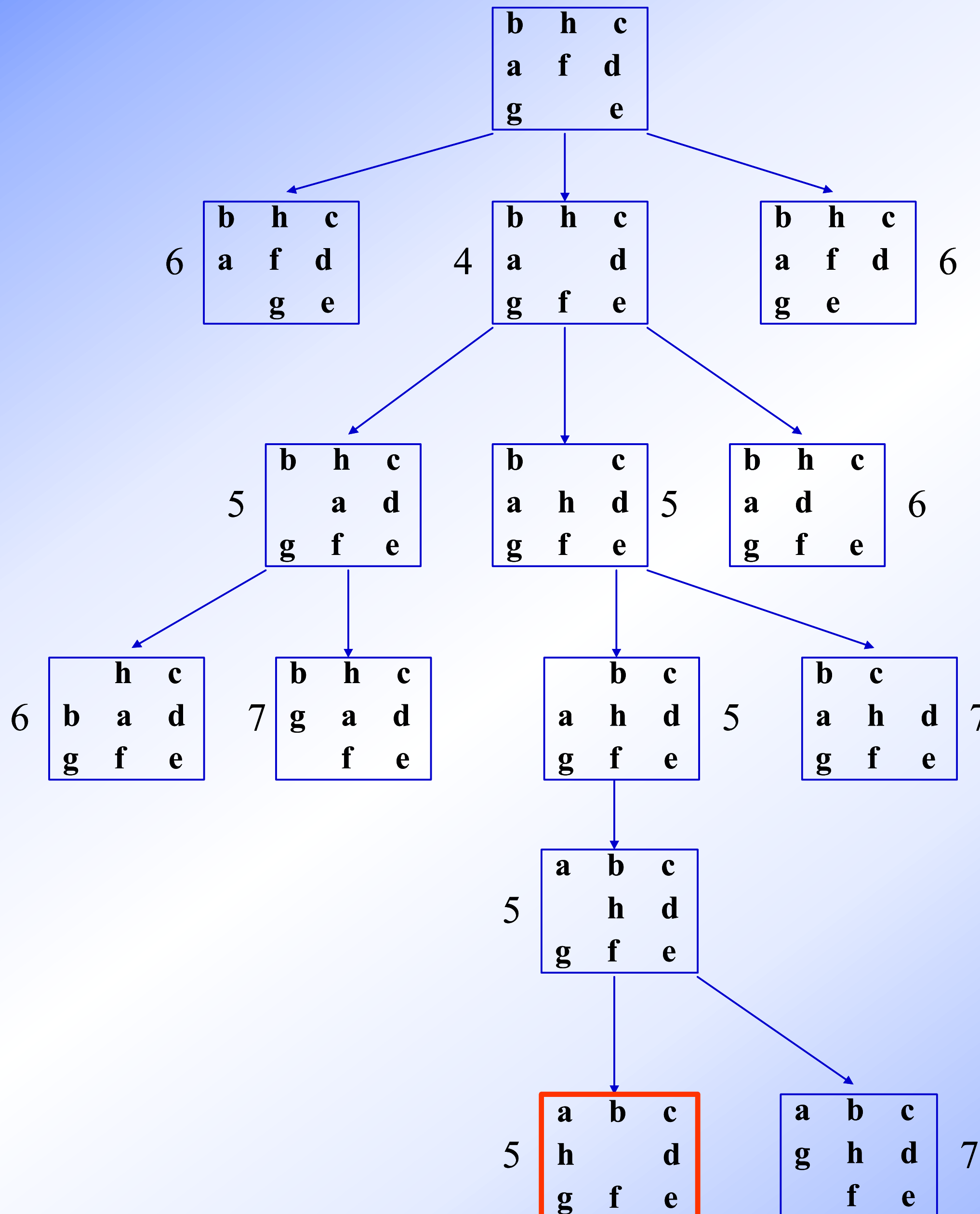
- **[评分标准]**：定义占3分，性质占3分，搜索图占6分。


3
S0
S1
S2
S3
Sg

习题4. 用A*算法求解九宫重排问题。问题的初始状态S_0和目标状态S_g分别为：

$$S_0 = \boxed{\begin{array}{ccc} b & h & c \\ a & f & d \\ g & & e \end{array}} \qquad S_g = \boxed{\begin{array}{ccc} a & b & c \\ h & & d \\ g & f & e \end{array}}$$

可使用的算符有：空格左移、空格上移、空格右移、空格下移。设估价函数为：**f(x)=d(x)+h(x)**，其中**d(x)**为结点**x**的深度，**h(x)**为结点**x**中还没有到位的字母个数。要求画出搜索图示，标出每个结点的估值，并指出解路径。



b h c
a f d
g e
6
4
6
5
5
6
6
7
5
7
5
5
7

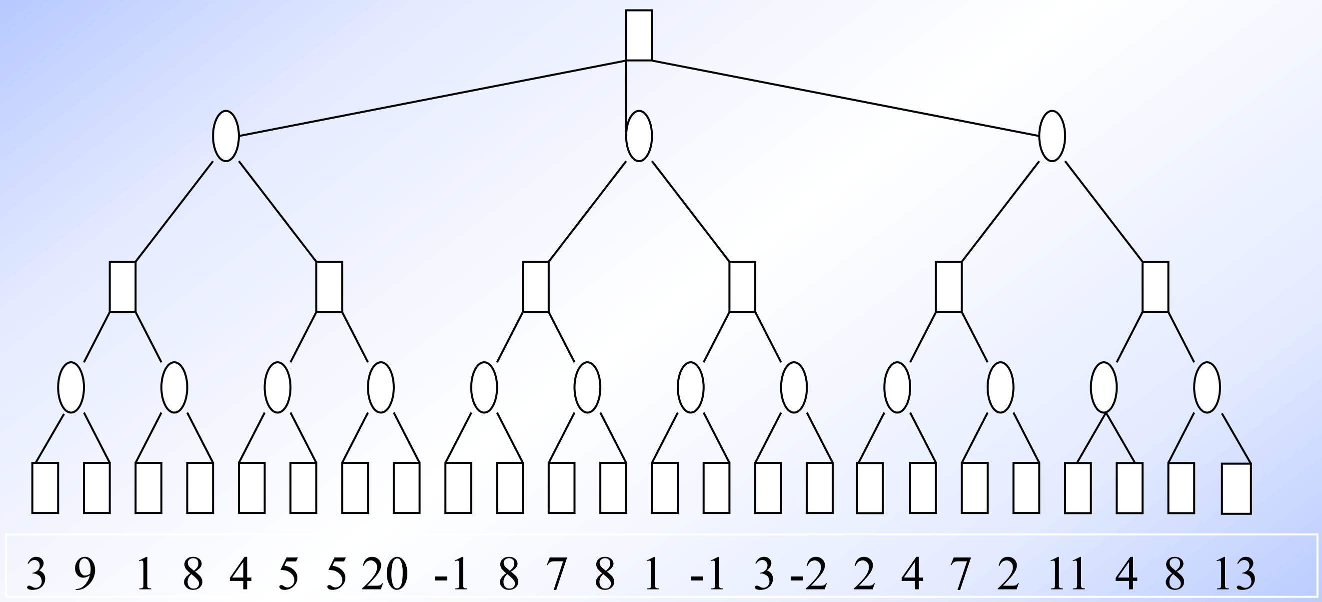
习题5. 用有界深度优先搜索法求解九宫重排问题。问题的初始状态S_0和目标状态S_g分别为：

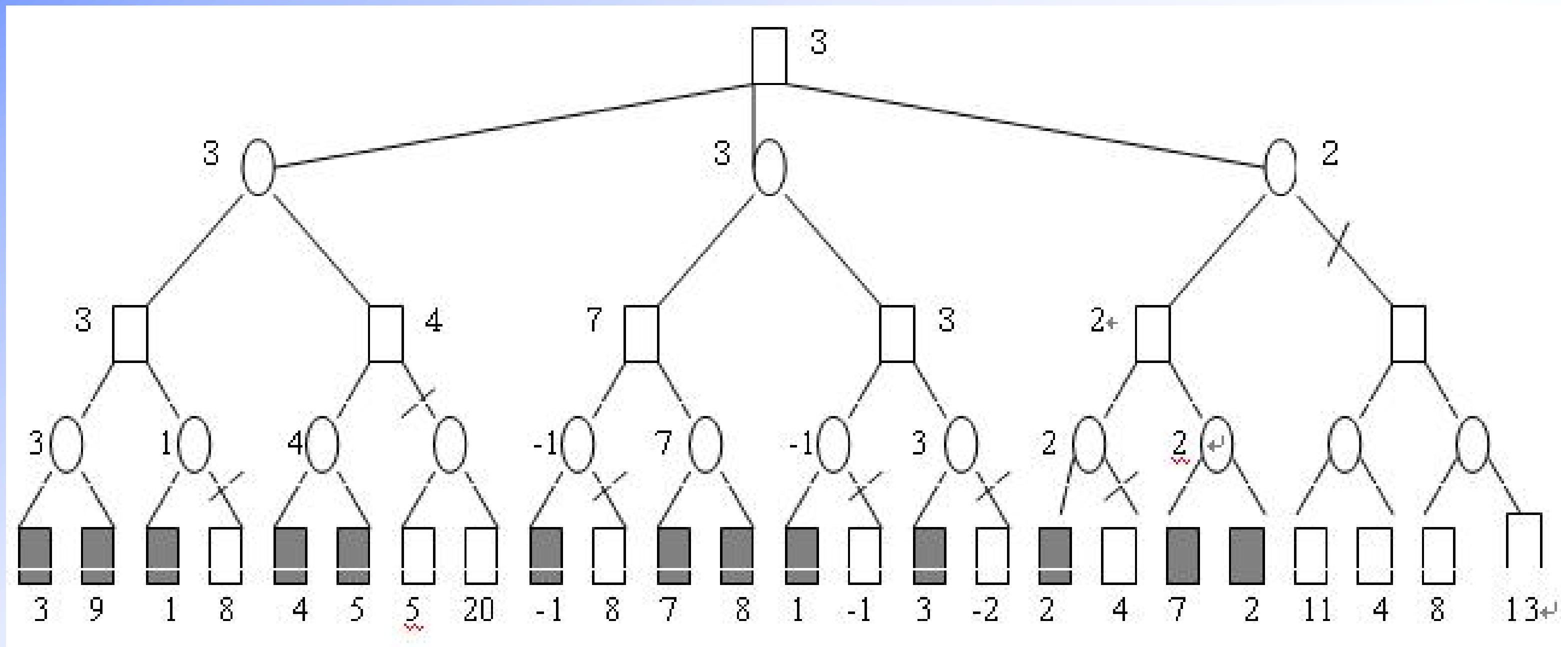
$$S_0 = \boxed{\begin{array}{ccc} b & h & c \\ a & f & d \\ g & & e \end{array}} \qquad S_g = \boxed{\begin{array}{ccc} a & b & c \\ h & & d \\ g & f & e \end{array}}$$

可使用的算符有：空格左移、空格上移、空格右移、空格下移。

设搜索深度界限d_m=5，要求画出搜索图示，并指出解路径。

习题6. 在如下所示博弈树中，按从左到右的顺序进行剪枝搜索，试标明各生成节点的倒退值，何处发生剪枝，及当前应选择的走步。





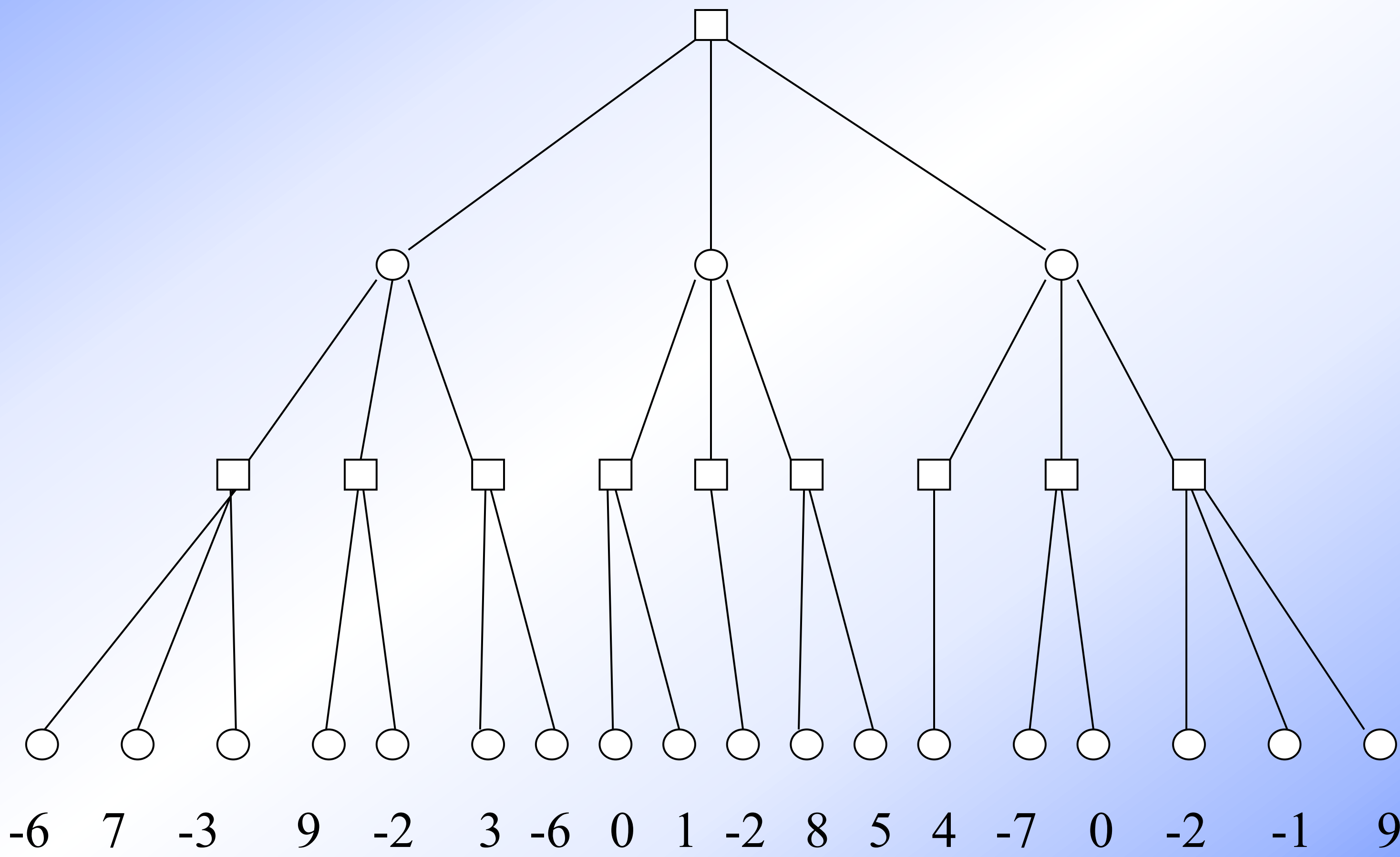
[评分标准]:

标明各生成节点的倒退值占2分；

标出发生剪枝位置占6分；

选出当前应选择的走步占2分。

习题7. 已知博弈树如图所示，其中方形结点为极大结点，圆形结点为极小结点。用α-β剪枝法找出当前最佳棋步。（要求标明α、β值和剪枝位置）



Exercises P. 162

1.

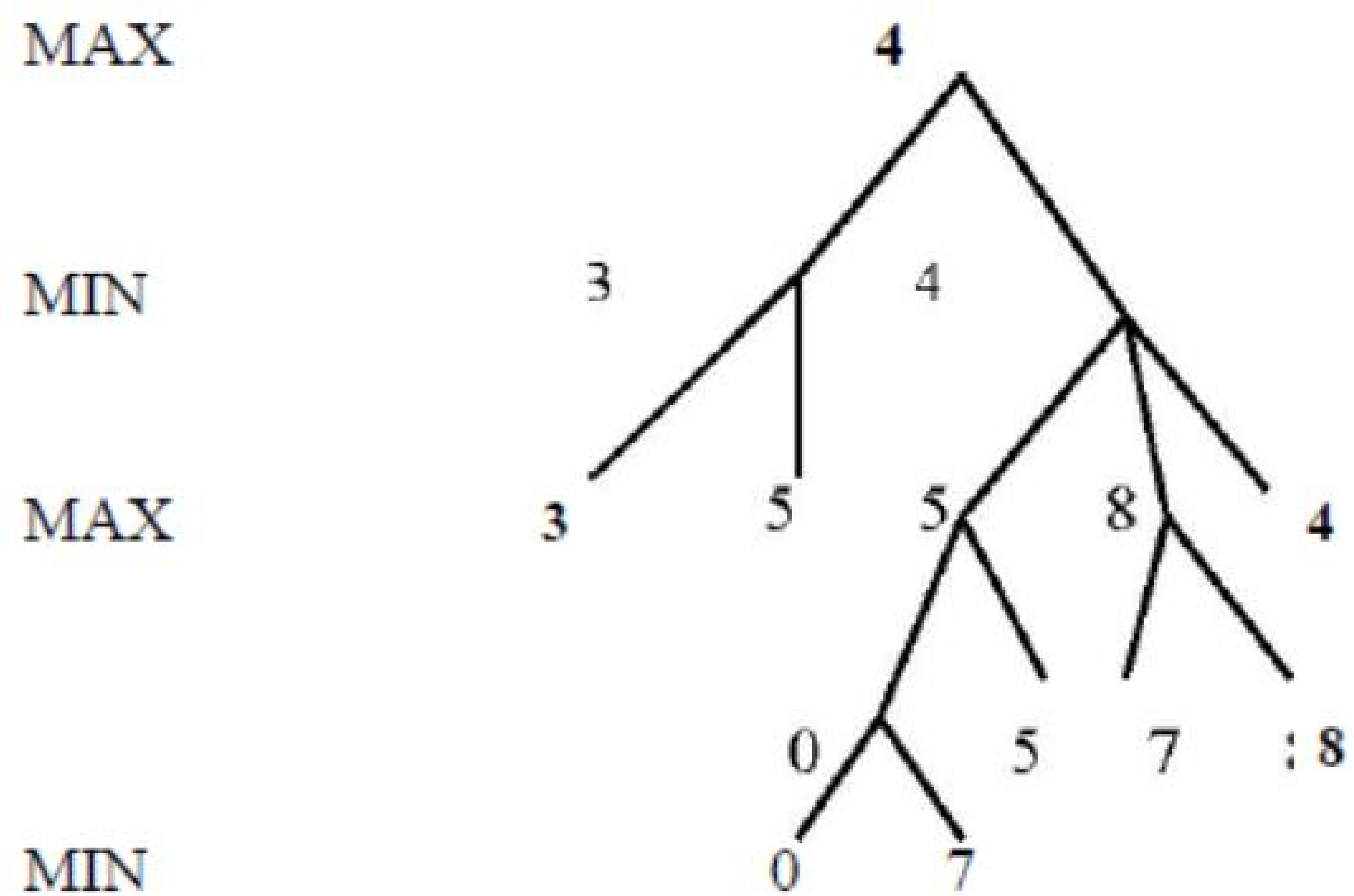
6.

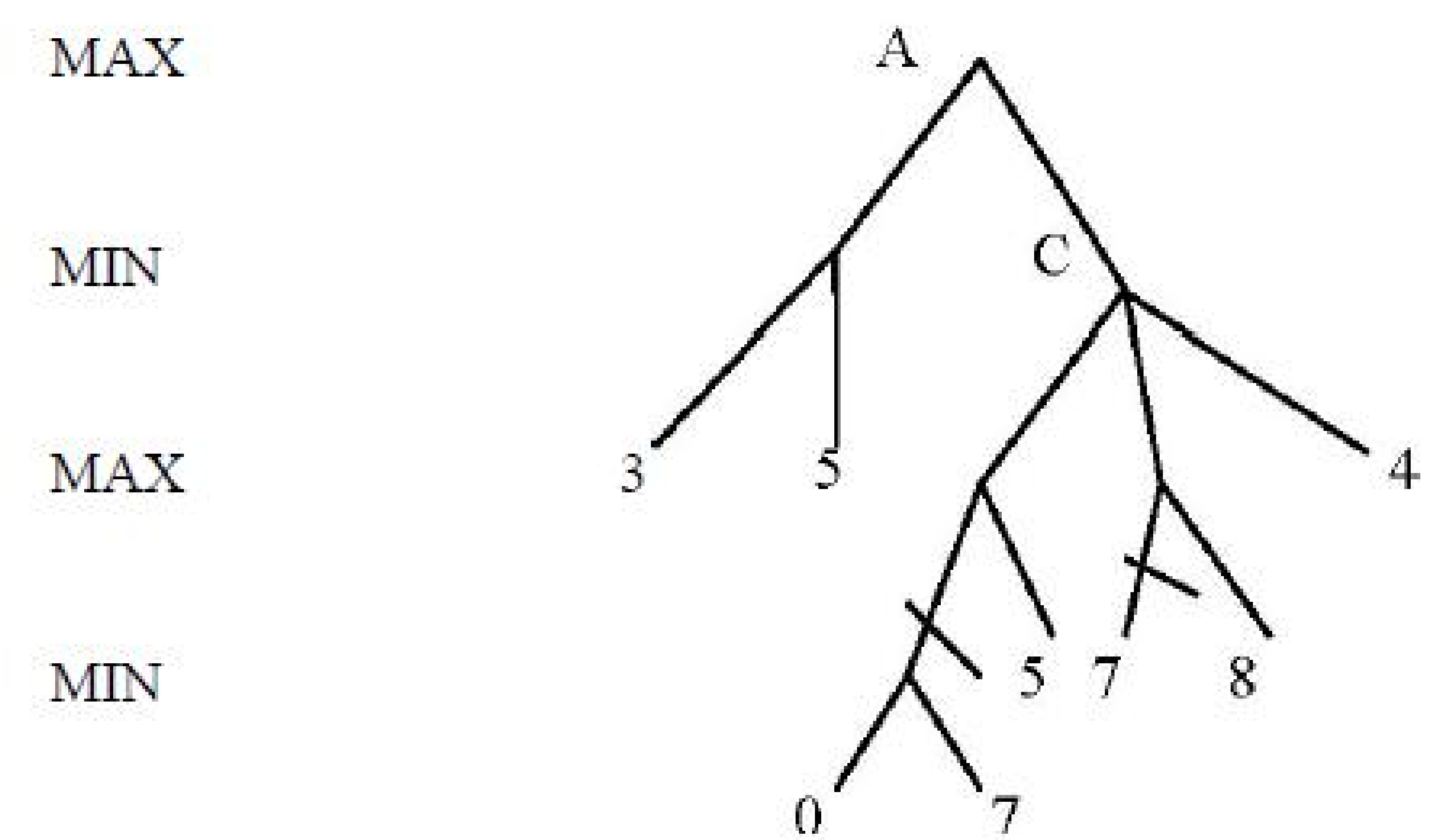
13.

14.

16. Figure 4.23、4.25

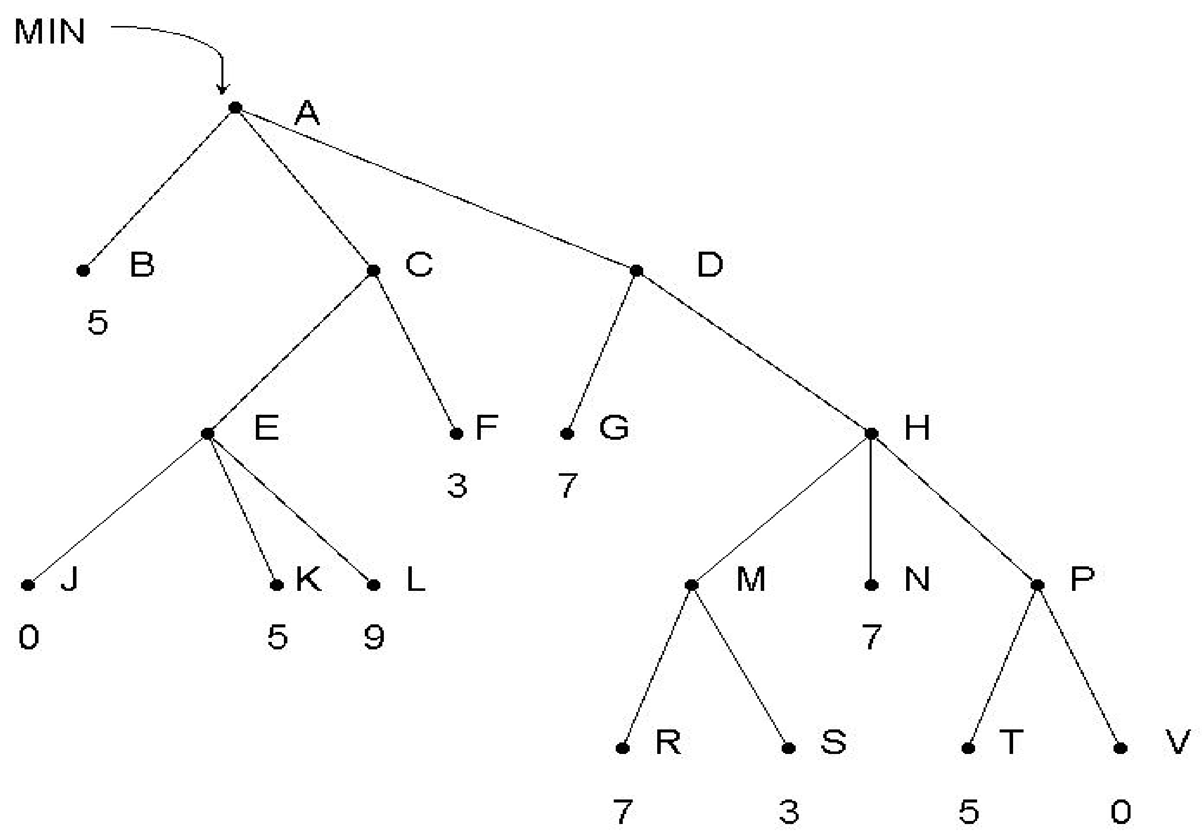
13. Perform MINIMAX on the tree shown in Figure 4.30.



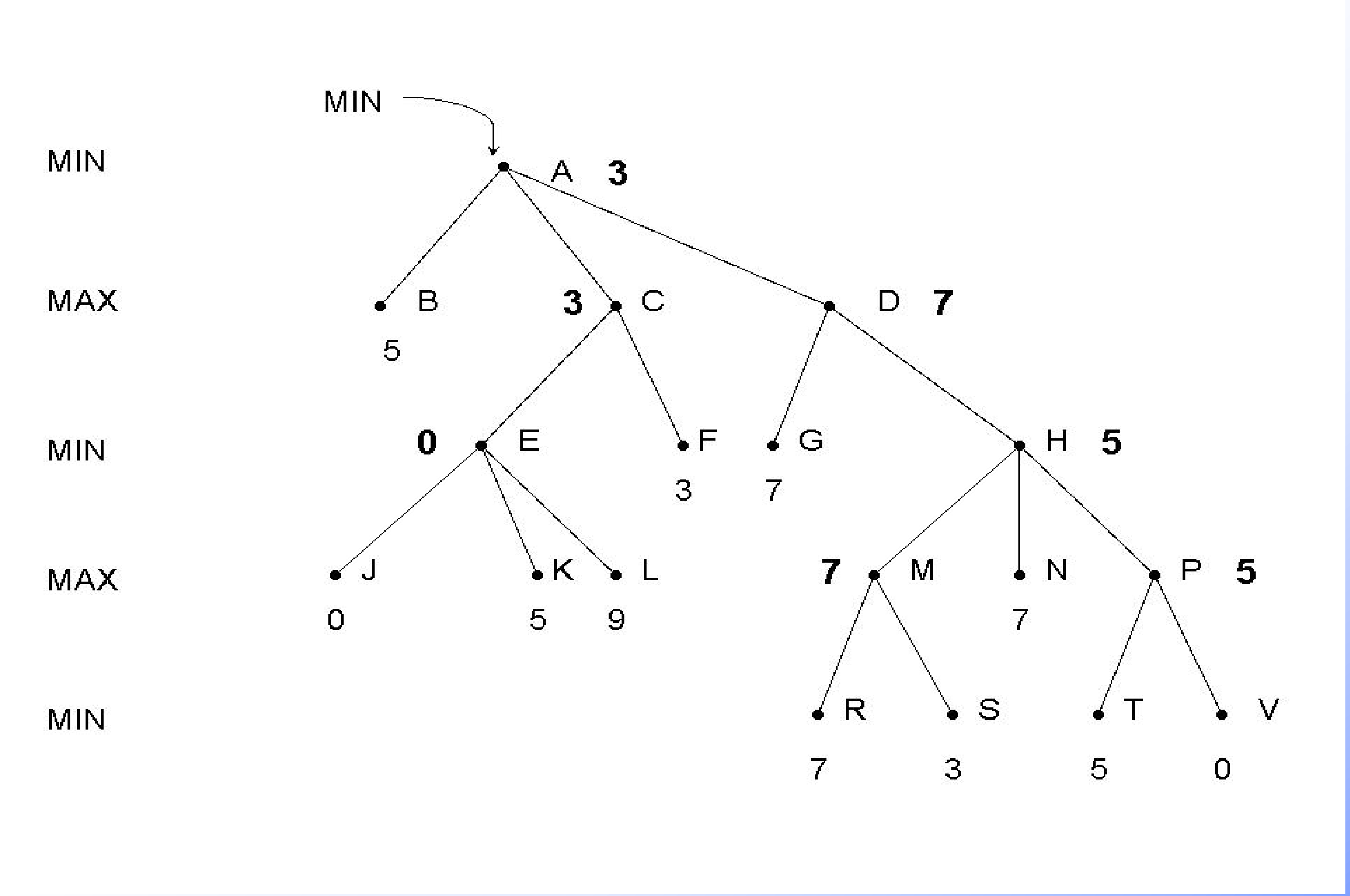


MAX
MIN
MAX
MIN
A
C
3
5
4
5
7
8
0
7

13. Perform minimax on the following tree.

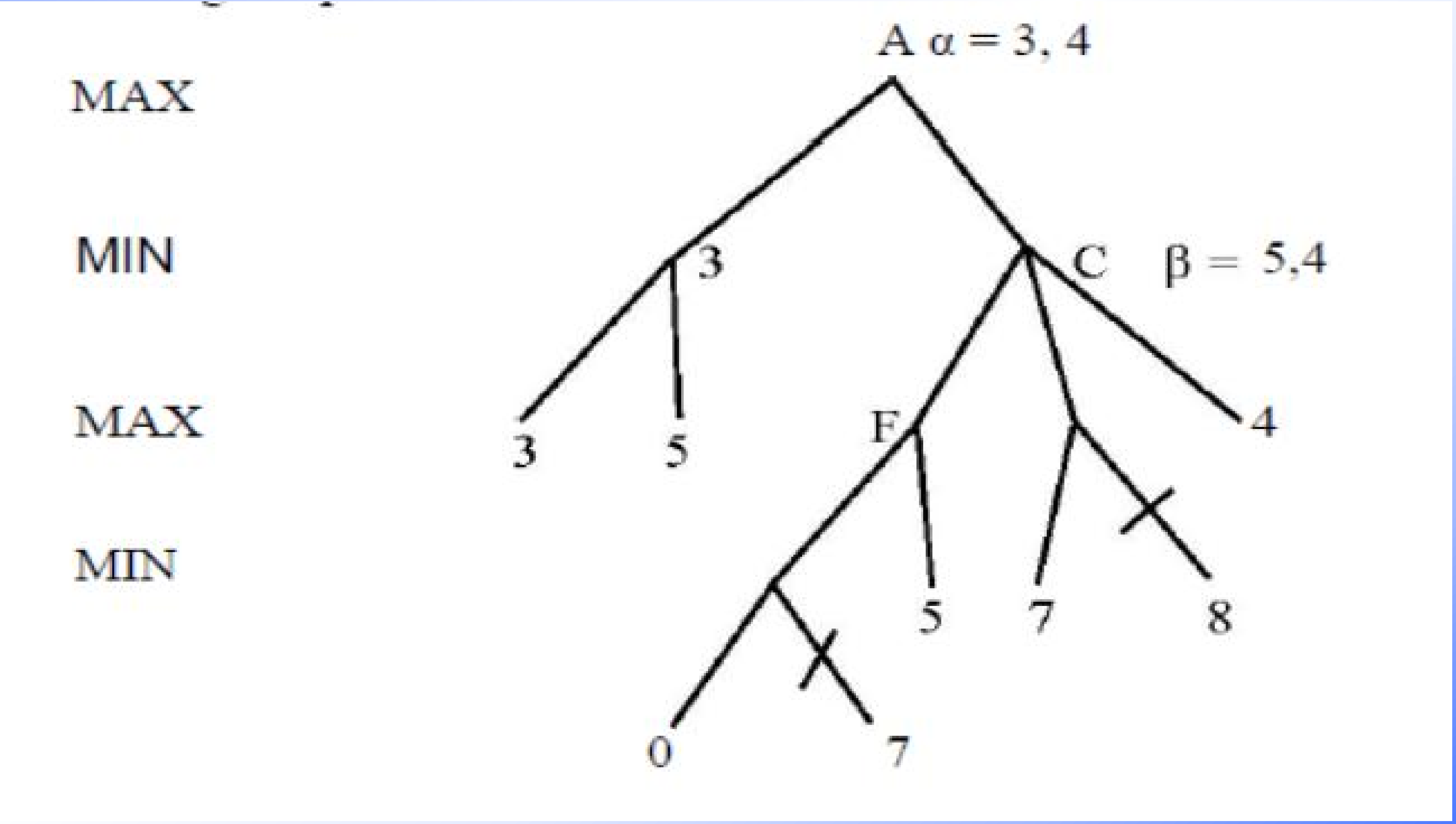


Solution:


MIN
MIN
MAX
MIN
MAX
MIN
A 3
B
5
3 C
D 7
0 E
F
3
G
7
H 5
J
0
K
5
L
9
7 M
N
7
P 5
R
7
S
3
T
5
V
0

14. Perform a left-to-right alpha-beta prune on the tree of Exercise 13. Perform a right-toleftprune on the same tree. Discuss why different pruning occurs.

Left-to-right alpha-beta:



MIN on left-most deepest subtree (3,5) to get α at A = 3.

From next left-most deepest subtree (0,7), MIN of 0, prune.

F is now 5 and β at C = 5.

Subtree (7,8) is MAX, so with 7, 8 is β pruned.

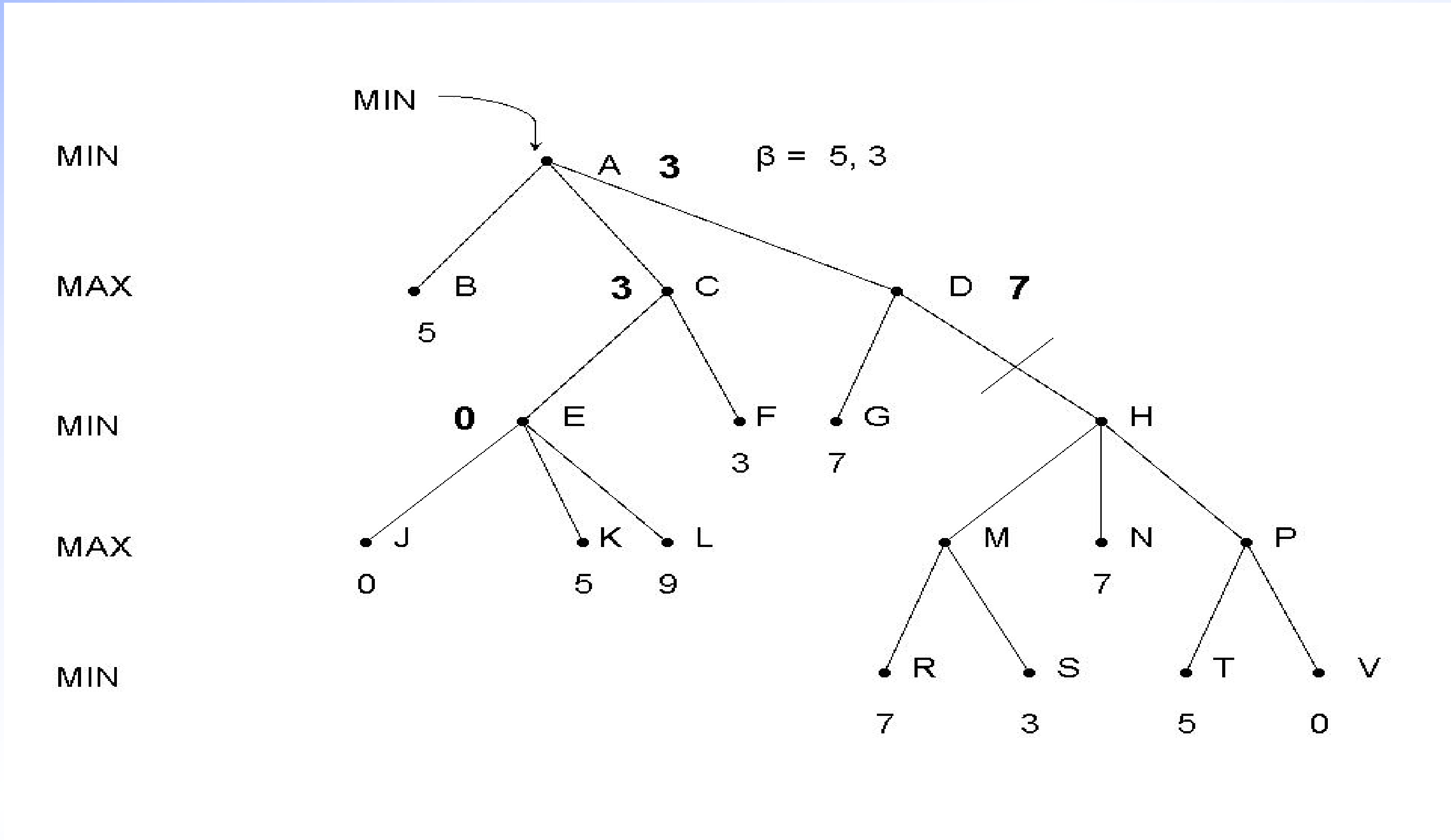
Seeing the right-most 4, β at C = 4, and α at A = 4, the final backed up value.

Right-to-left alpha-beta:

Starting at the right-most deepest subtree, 4 gives β of C = 4.
Seeing 8 in subtree (7,8), 7 is β-pruned.
Seeing 5 next the entire subtree (7,8) is β-pruned.
A takes on the α value of 4.
5 and then 3 are examined (MIN) in the left-most subtree.
The final value of A = 4.

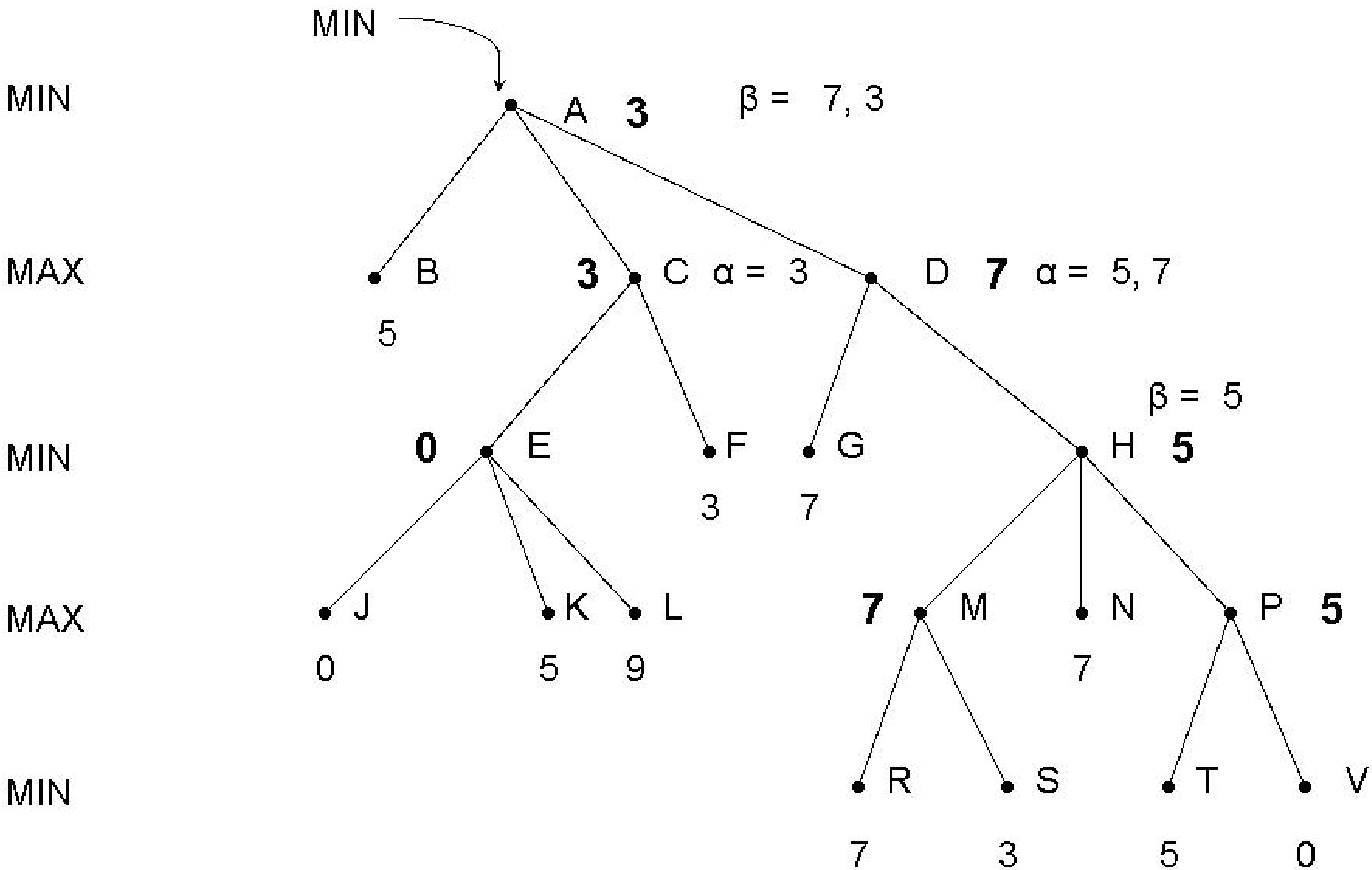
14. Perform a left-to-right alpha-beta prune on the tree from Exercise 13. Perform a right-to-left prune on the same tree. Discuss why a different pruning occurs.

Solution:(Left-to-right)



The tree is searched depth-first from left to right. The value of 5 at node B is reported, and β set to 5. Next the min of (0,5,9) is evaluated and 0 passed to E. C takes the value 3 which is the max of (0,3) and 3 is assigned to β, since 3 is less than 5. When the 7 is found at node G, the right subtree below D can be β-pruned because the value of D will be at least as large as 7, which is greater than the most recent β value of 3.

Right-to-left:


MIN
MIN
A 3
β = 7, 3
MAX
B
5
3 C α = 3
D 7 α = 5, 7
β = 5
MIN
0 E
F
3
G
7
H 5
MAX
J
0
K
5
L
9
7 M
N
7
P 5
MIN
R
7
S
3
T
5
V
0

The tree is searched depth first, right to left. (0,5) are compared, and P is assigned 5. The β-value associated with node H is set to 5, so no subtree below H with a value greater than 5 need be considered. We look next at node N, which is 7. We could not prune N because we had to look at it to determine its value. Next, node S is searched, which gives a value of 3. Since this will potentially change the value of β at H, we have to search R, which gives us 7 at M and 5 at H. We set the α value at node D to 5 because we do not need to consider any min grandchild of D whose value is less than 5. D only has one other grandchild, and its value is 7, so the α associated with D is set to 7.

The β value of the root node is set to 7, and no child of the root evaluating to more than 7 need to be considered further. The next node evaluated is F, which has a value of 3, and the α value associated with C is set to 3, and no node whose value is greater than 3 need be considered. However, L, K, and J have to be evaluated because L, and K both have values greater than 3. Thus 0 is assigned to E, and 3 is assigned to the β value associated with the root. The final node, B, cannot be pruned because although its value is greater than the last β value of the root, it has to be examined to determine this.