目录

CONTENTS

4.1 知识图谱

4.2 本体知识表示

4.3 万维网知识表示

4.4 知识图谱构建

4.5 知识图谱的现状及发展

中国石油大学(华东)
CHINA UNIVERSITY OF PETROLEUM

计算机科学与技术学院
College of Computer Science & Technology

第五章 搜索技术

人工智能课题组

智能科学与技术系

目录 CONTENTS

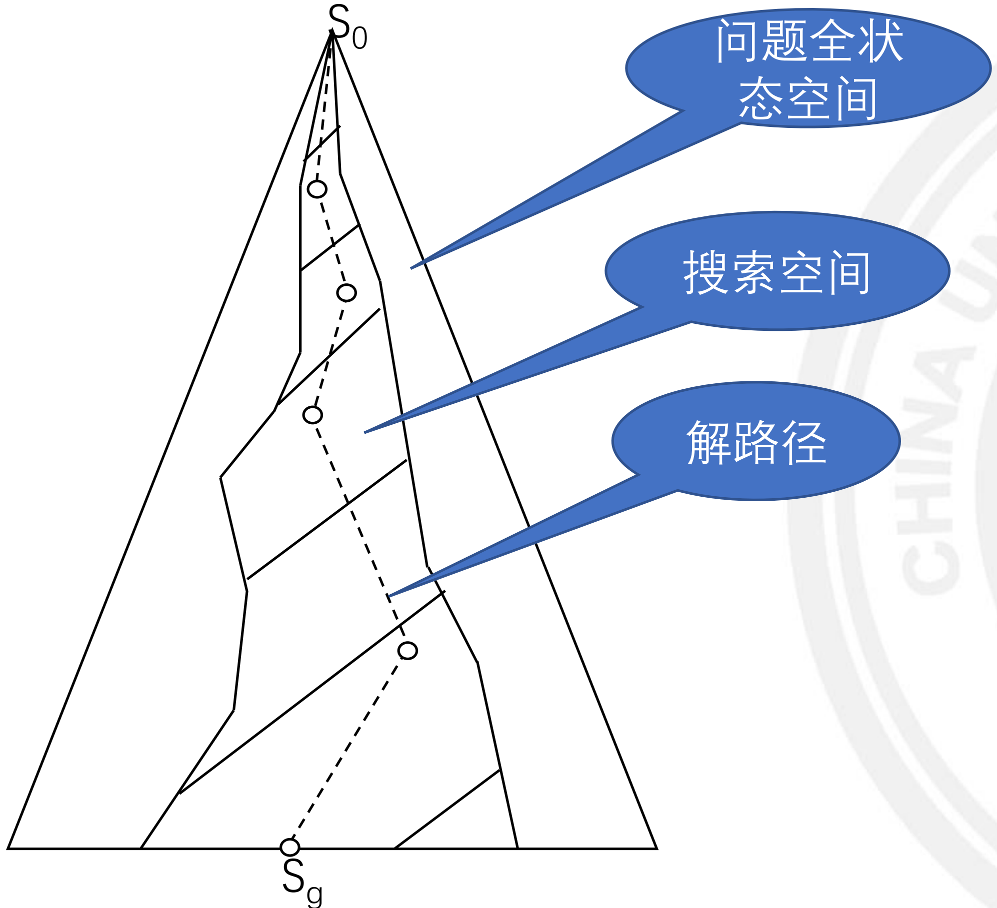
5.1 图搜索策略
5.2 盲目搜索
5.3 启发式搜索
5.4 博弈搜索

概述

■ 搜索的定义

- 搜索是人工智能中的一个基本问题，它解决**如何在一个比较大的问题空间中，只通过搜索比较小的范围就找到问题的解**的问题。

- 搜索空间示意图：



概述

■ 搜索算法的描述

- 搜索算法的**输入**是**给定的问题**，**输出**表示为**动作序列的方案**。

- **给定的问题**就是确定该问题的基本信息：
  1. **初始状态集合**：定义了问题的初始状态。
  2. **操作符集合**：把一个问题从一个状态变换为另一个状态的动作集合。
  3. **目标检测函数**：用来确定一个状态是不是目标。
  4. **路径费用函数**：对每条路径赋予一定费用的函数。

- 一旦有了**方案**，就可以执行该方案所给出的动作了。（执行阶段）

概述

■ 分类

1. 不考虑给定问题所具有的特定知识，系统根据事先确定好的某种固定排序，依次调用规则或随机调用规则，称为**无信息引导的搜索策略**。（***盲目搜索***）
2. 考虑问题领域可应用的知识，动态地确定规则的排序，优先调用比较合适的规则使用，称为**启发式搜索策略**或者**有信息导引的搜索策略**。

概述

■ 搜索问题实例：传教士和野人问题

3个传教士和3个野人来到河边准备渡河，河岸有一艘船，每次至多可供2人乘渡。为了安全起见，传教士应如何规划摆渡方案？

➢ 使任何时刻在河的两岸以及船上的野人数目总是**不超过**传教士的数目（但允许在河的某一岸或者船上只有野人或者只有传教士）

- 在每次渡河之后都会有几种渡河方案供你选择，究竟哪种方案才有利于在满足所有约束条件下顺利过河？

- 该问题的基本信息：
  - 用一个三元组(M,C,B)表示**状态**，其中M、C分别表示在左岸的传教士、野人人数，B表示船是否在左岸，1表示船在左岸。

  1、问题的初始状态为(3,3,1)

  2、目标状态为(0,0,0)。

5.1 图搜索策略

■ 很多搜索问题都可以转化为图搜索问题

例子：传教士与野人问题的图搜索策略解法：

- 用一个三元组(M,C,B)表示状态，其中M、C分别表示在左岸的传教士、野人人数，B表示船是否在船是否在左岸，1表示船在左岸。
- 问题的初始状态为(3,3,1)，目标状态为(0,0,0)。

- **如何通过图找到一条从初始状态到目标状态的路径**，就是图搜索问题。

- 所谓的路径就是给出一个状态序列，序列的第一个状态是初始状态，最后一个状态是目标状态，序列中任意两个相邻的状态之间，通过一个连线连接。

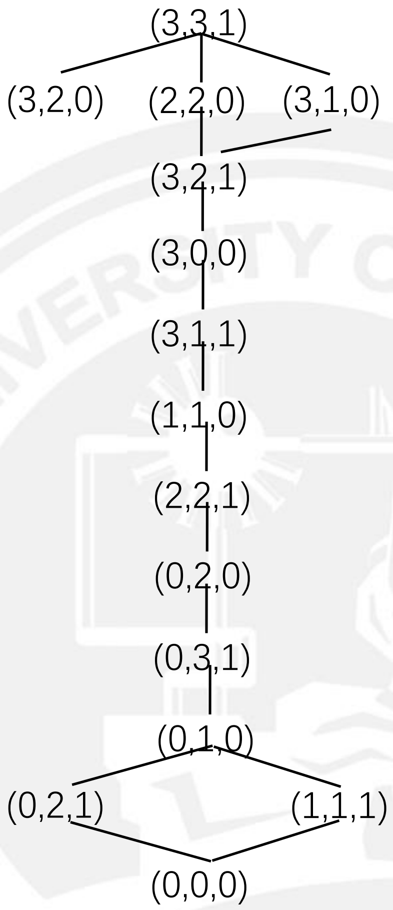
5.1 图搜索策略

■ 很多搜索问题都可以转化为图搜索问题

例子：传教士与野人问题的图搜索策略解法：

➢图中的**结点**代表**状态**

➢图中**结点之间的连线**代表状态之间可以相互转换（**操作**）



5.1 图搜索策略

- 为了提高搜索效率，图搜索并不是先生成所有状态的连接图再进行搜索，而是**边搜索边生成图**，直到找到一个符合条件的解，即路径为止。

- **图搜索算法的评价方法**：在搜索的过程中，生成的无用状态越少——即**非路径上的状态越少**，搜索的效率就越高，对应的搜索策略就越好。

- **图搜索算法的分类**：类似于搜索的分类，图搜索可以分成：
  - 5.2 盲目搜索
  - 5.3 启发式搜索

5.2 盲目搜索

- 如果在搜索过程中没有利用任何与问题有关的知识或启发信息，则称之为**盲目搜索**。

- 没有考虑到问题本身的特性，这种搜索具有很大的盲目性，效率不高，不便于复杂问题的求解。

- 一般是按预定的搜索策略进行搜索：

  5.2.1 深度优先搜索

  5.2.2 广度优先搜索

5.2.1 深度优先搜索

基本思想：优先扩展**深度最深**的节点。

- 如果有相同深度的多个节点，则按照事先的约定从中选择一个。
- 如果该节点没有子节点，则选择一个除了该节点以外的深度最深的节点进行扩展。

5.2.1 深度优先搜索

例子：N皇后问题

在一个$N \times N$的国际象棋棋盘上摆放N枚皇后棋子，摆好后要满足每行、每列和每个对角线上只允许出现一枚皇后，即棋子间不许相互俘获。以4皇后为例，下图给出了4皇后问题的一个解。

	Q		
			Q
Q			
		Q	

4皇后问题的一个解

5.2.1 深度优先搜索

例子：N皇后问题

- 为了求解该问题，我们用坐标表示一个皇后所在的位置，如上例中，1行2列有一个皇后，则可表示成(1,2)。
- 多个皇后则用所有皇后的位置组成的表表示。下图给出的解就可以表示为：((1,2),(2,4),(3,1),(4,3))。

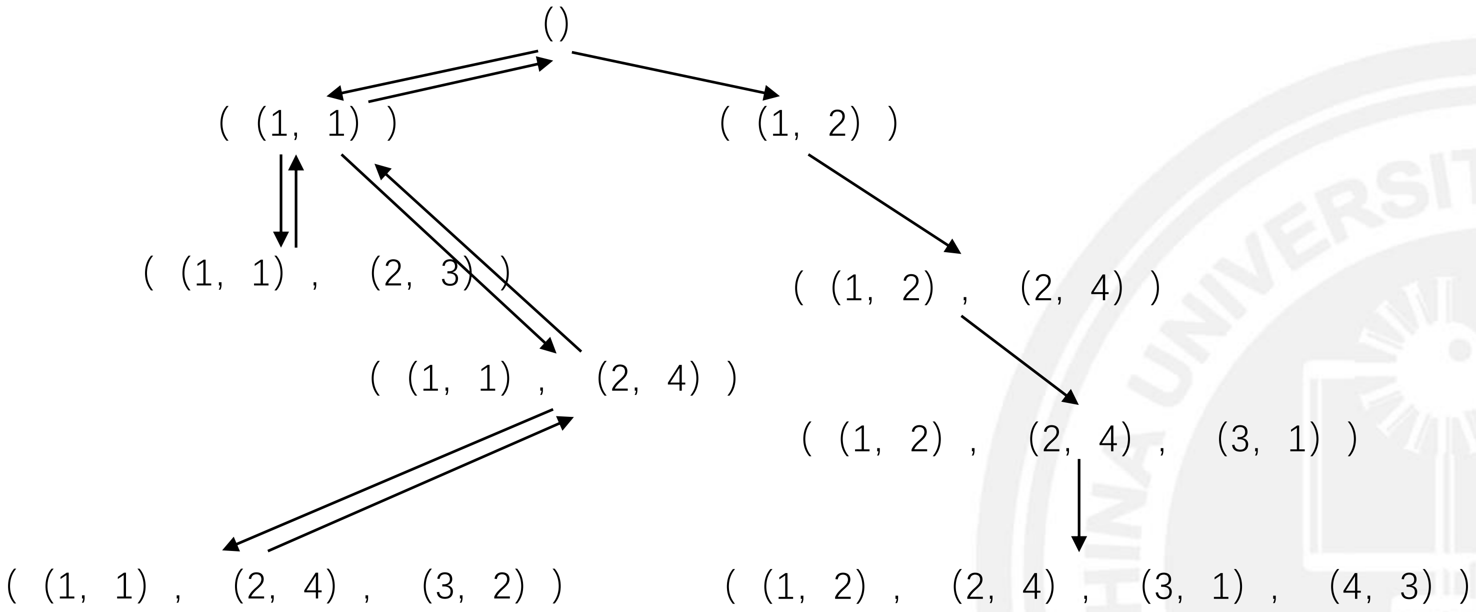
	Q		
			Q
Q			
		Q	

4皇后问题的一个解

- 假设搜索过程是从上向下按行进行，每一行从左到右按列排列，则深度优先搜索过程如图所示：

5.2.1 深度优先搜索

例子：N皇后问题



5.2.1 深度优先搜索

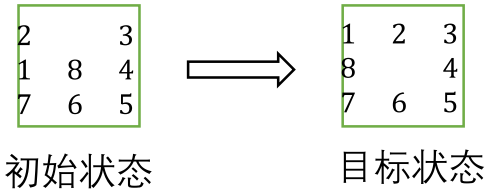
- 在深度优先搜索过程中，每当某一行不能摆放棋子时，就发生“回溯”返回到一个深度较浅的节点进行试探，否则就一直选择深度深的节点进行扩展。

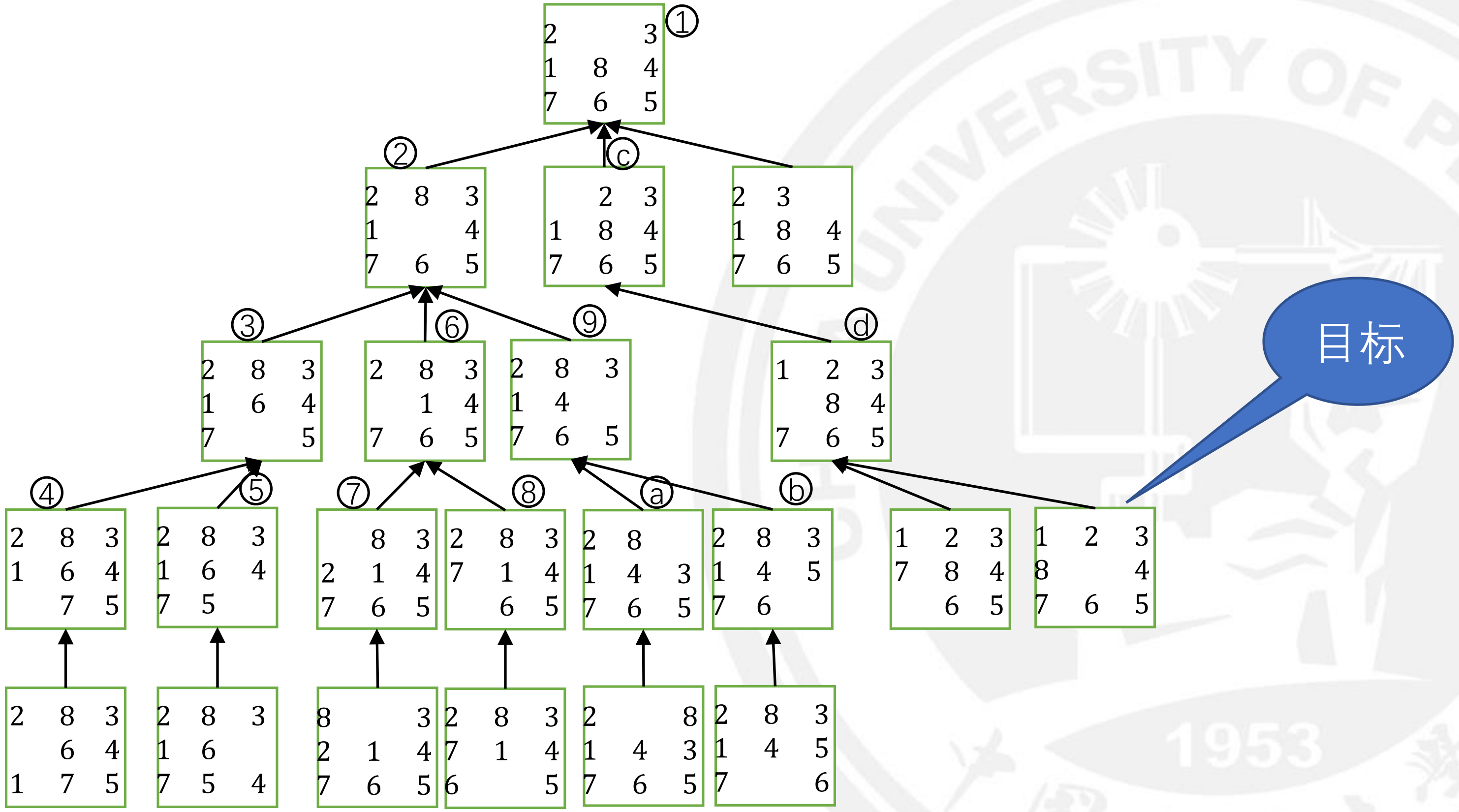
- 对于很多问题，这样做可能会导致“错误”的线路搜索下去而陷入“深渊”。为了防止这样的事情发生，在深度优先搜索中往往会加上一个深度限制，即在搜索过程中**如果一个节点的深度达到了深度限制**，无论该节点是否还有子节点，都**强制进行回溯**，选择一个稍浅的节点扩展，而不是沿着最深的节点继续扩展。

5.2.1 深度优先搜索

例子：八数码问题

- 带深度限制的深度优先策略求解八数码问题搜索图：





5.2.1 深度优先搜索

例子：八数码问题

- 八数码问题搜索图：

  - 深度限制为4。当达到深度限制之后，回溯到稍浅一层的节点继续搜索直到找到目标节点。需要根据经验设置一个合理值。如果深度限制过深，则影响求解效率；反之如果限制过浅，则可能导致找不到解。可以采取逐步加深的方法，先设置一个较小的值，然后再逐步加大。

5.2.1 深度优先搜索

- 深度优先搜索也有可能遇到“**死循环**”问题，也就是沿着一个环路一直搜索下去。

- 为了解决这个问题，可以在搜索过程中**记录从初始节点到当前节点的路径**，每扩展一个节点，就要检测该节点是否出现在这条路径上；**如果发现出现在该路径上，则强制回溯**，探索其他深度最深的节点。

5.2.2 宽度优先搜索

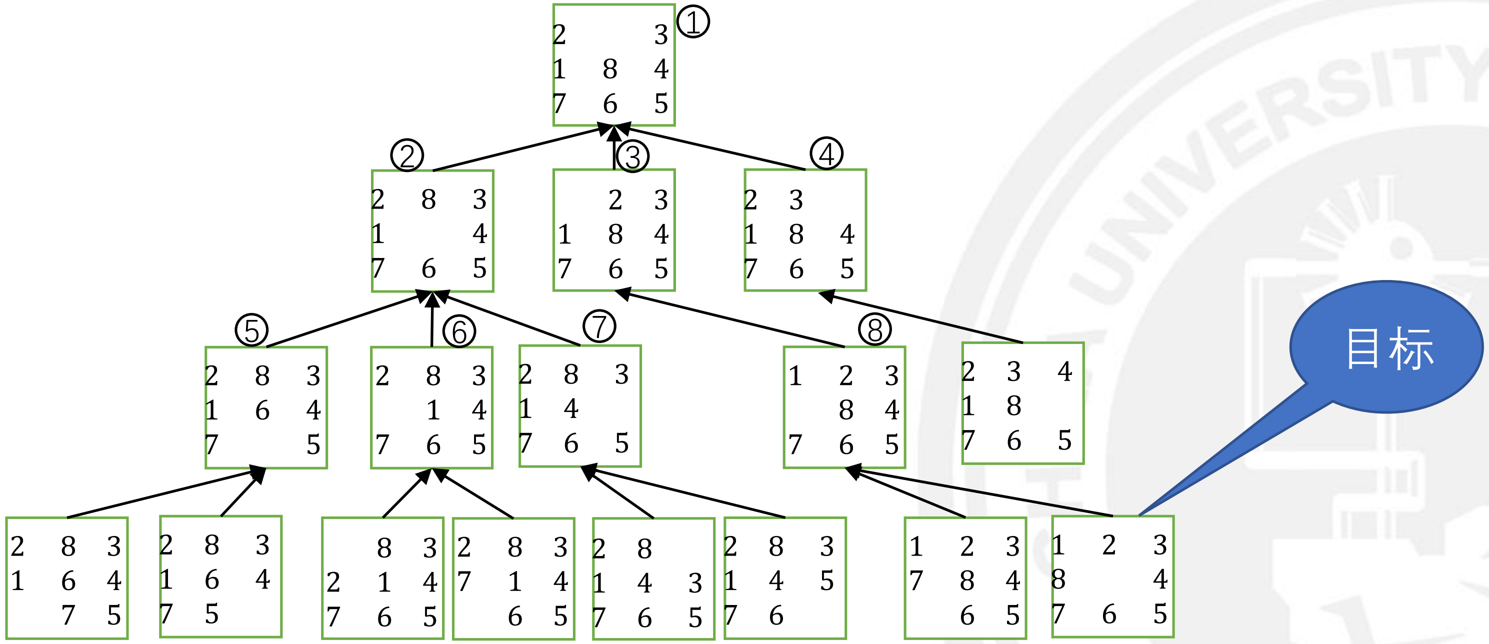
与深度优先策略刚好相反，宽度优先策略是优先搜索深度浅的节点。

- 每次选择**深度最浅**的叶节点进行扩展。
- 如果有深度相同的节点，则**按照事前约定**从深度最浅的几个节点中选择一个。

5.2.2 宽度优先搜索

例子：八数码问题

- 宽度优先策略求解八数码问题搜索图：



5.2.2 宽度优先搜索

宽度优先搜索与深度优先搜索的不同：

- 在问题有解的情况下，宽度优先搜索一定可以找到最优解。

- 由于**宽度优先搜索在搜索过程中需要保留已有的搜索结果**，需要占用比较大的搜索空间，而且会随着搜索深度的加深成几何级数增加。深度优先搜索虽然不能保证找到最优解，但是可以采用回溯的方法，只保留从初始节点到当前节点一条路径即可，可以大大节省存储空间，所需存储空间只与搜索深度成线性关系。

5.3 启发式搜索

- 如何在搜索过程中引入启发信息，减少搜索范围，以便尽快地找到解，这种搜索策略称为**启发式搜索。**

- 常用地启发式搜索算法
  - A算法
  - A*算法

5.3 启发式搜索

A算法：

为了尽快找到从初始节点到目标节点的一条耗散值比较小的路径，我们希望所选择的节点尽可能在最佳路径上。

- A算法评价一个节点在最佳路径上的可能性的评价函数定义为：

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

其中，n为待评价的节点；

$g(n)$为从初始节点s到节点n的最佳路径耗散值的估计值；

$h(n)$为从节点n到目标点t的最佳路径耗散值的估计值，称为启发函数；

$f(n)$为从初始节点s经过节点n到达目标节点t的最佳路径耗散值的估计值，称为评价函数。

- 这里的耗散值指的是路径的代价，根据求解问题的不同可以是**路径的长度**、**需要的时间**或**花费的费用**。

5.3 启发式搜索

A算法：

- 如果$f(n)$能够比较准确地估计出$\boldsymbol{s}-\boldsymbol{n}-\boldsymbol{t}$这条路径的耗散值的话，我们每次从叶节点中选择一个值最小地节点扩展。采用这种搜索策略的搜索算法，称之为A算法。

- $g(n)$很容易计算得到，启发函数$h(n)$则需要根据问题定义。

5.3 启发式搜索

A算法实现：

- 设置一个变量OPEN用于存放那些**搜索图上的叶节点**，也就是已经被生成出来但是还没有扩展的节点；变量CLOSED用于存放**搜索图中的非叶节点**，也就是那些不但被生成出来还被**扩展**的节点。
- OPEN中的节点按照f值从小到大排列。每次A算法从OPEN表中取出第一个元素（也就是f值最小的节点n）进行扩展。
  - 如果n是目标节点，**算法结束**；
  - 否则，扩展n。
    - 对于n的子节点m，如果m既不在OPEN中也不在CLOSED中，将m加入OPEN中；
    - 如果m在OPEN里，说明从初始点到m有两条路径，保留耗散值小的那条；
    - 如果m在CLOSED里，也说明从初始节点到m有两条路径，如果找到的新路径耗费较小，将m从CLOSED中取出放到OPEN中，对OPEN重新进行排序。
  - 重复上面的过程，直到**找到解**或者OPEN为空**算法失败**。

5.3 启发式搜索

A算法实现：

例子：八数码问题

- 定义八数码问题的**启发函数**为：

$$h(n) = \text{不在位将牌的个数}$$

- 将下图所示的初始状态与目标状态进行比较， 也就是初始状态的h值等于4。

2	8	3
1	6	4
7		5

初始状态

1	2	3
8		4
7	6	5

目标状态

5.3 启发式搜索

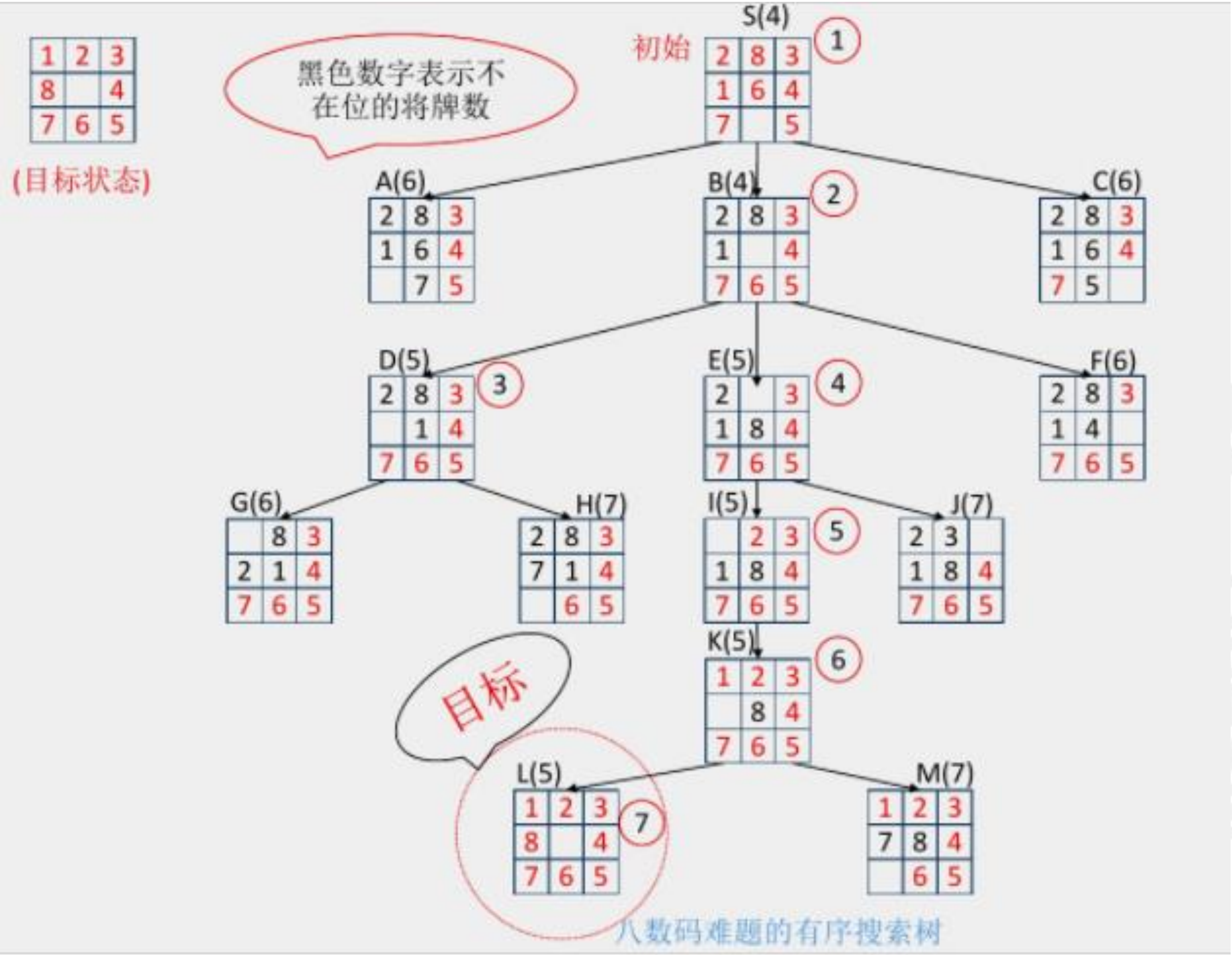
A算法实现：

例子：八数码问题

- 用A算法求解八数码问题的搜索图：

$f(n) = g(n) + h(n)$，$h(n)$ = 不在位将牌的个数

- OPEN和CLOSED表的变化



（括号内是路径总的耗散值f=g+h）

5.3 启发式搜索

A算法实现：

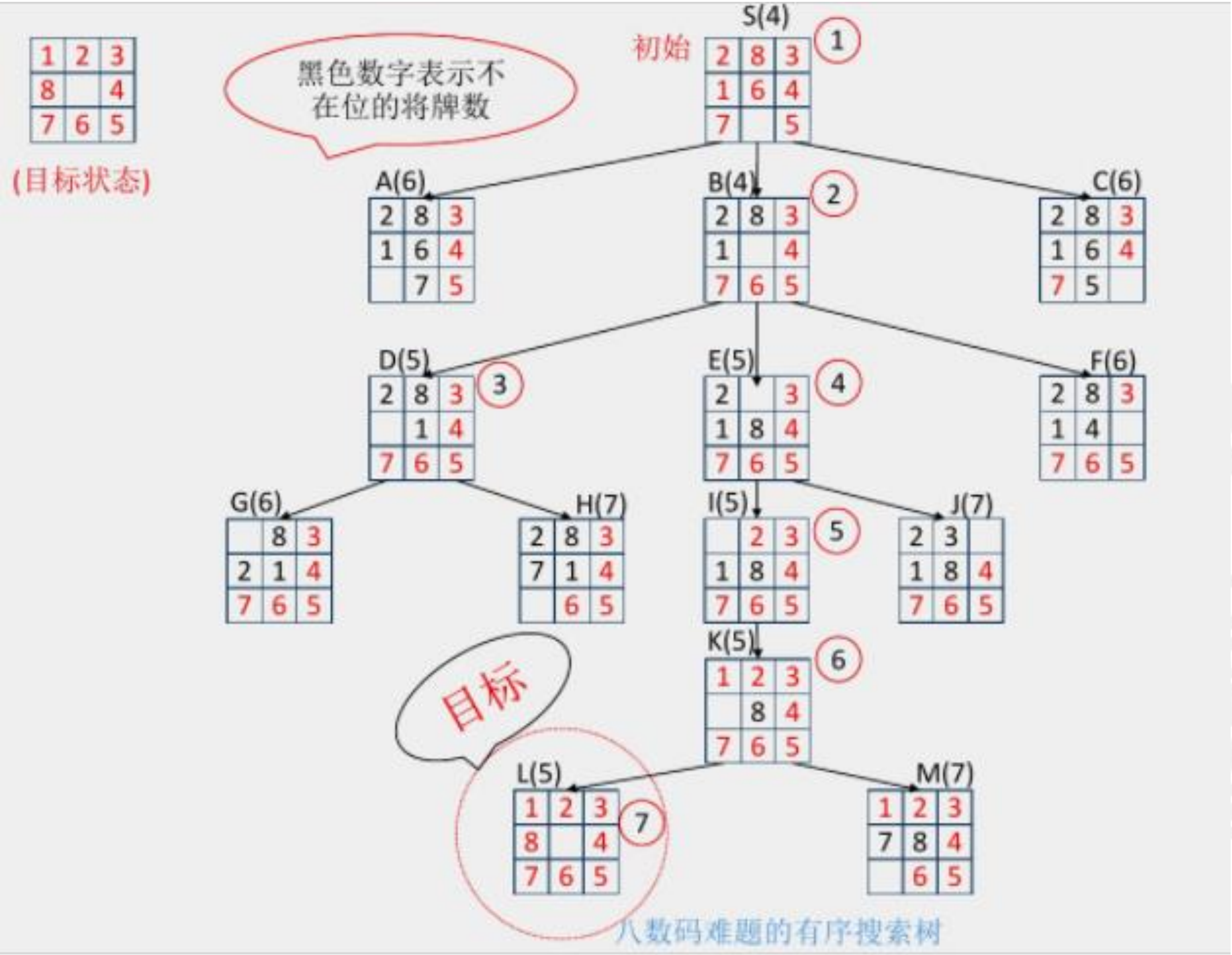
例子：八数码问题

- 用A算法求解八数码问题的搜索图：

$f(n) = g(n) + h(n)$，$h(n)$ = 不在位将牌的个数

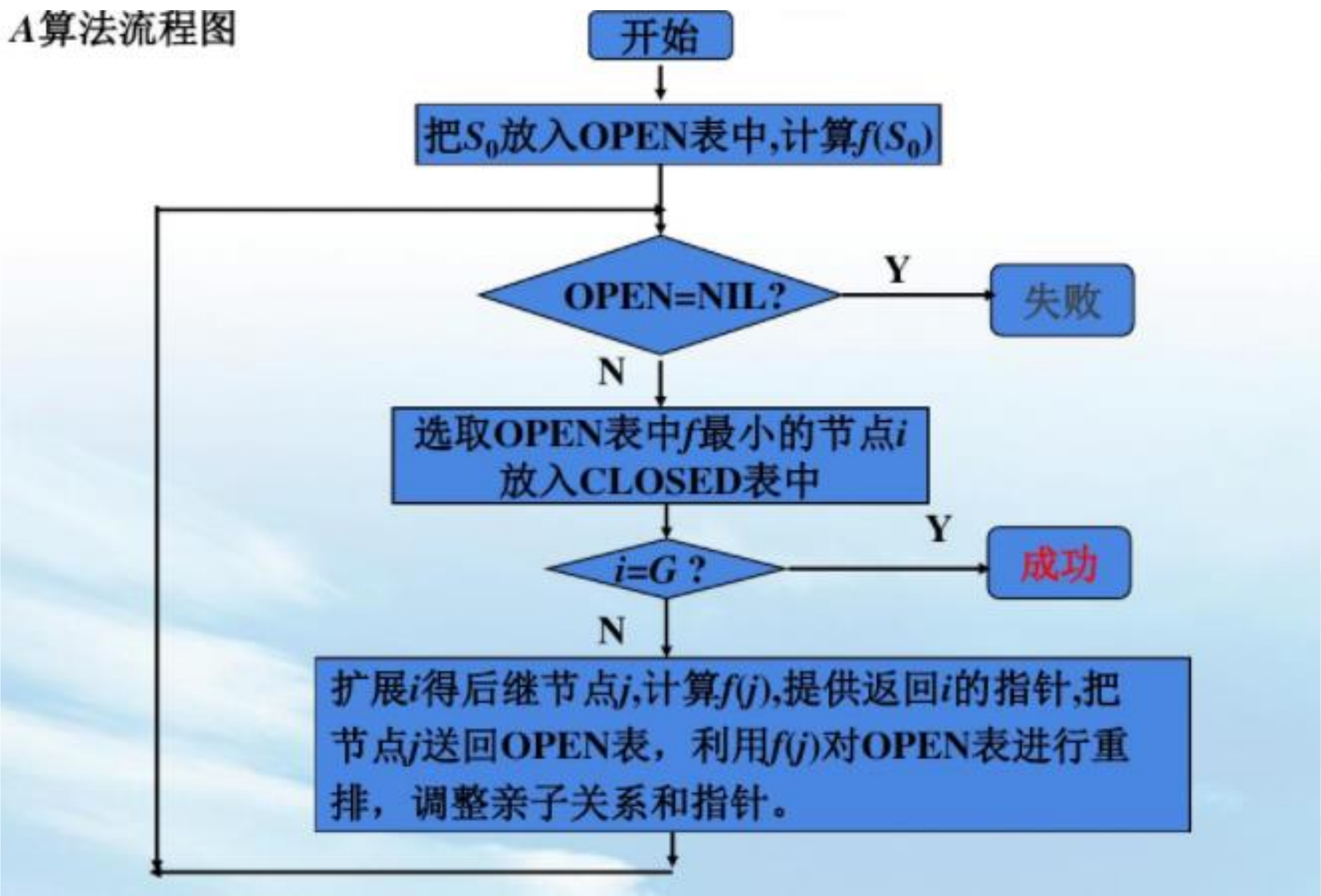
- OPEN和CLOSED表的变化

重复	OPEN表	CLOSE表
0	S(4)	[ ]
1	[B(4), A(6), C(6)]	[S(4)]
2	[D(5), E(5), A(6), C(6), F(6)]	[S(4), B(4)]
3	[E(5), A(6), C(6), F(6), G(6), H(7)]	[S(4), B(4), D(5)]
4	[I(5), A(6), C(6), F(6), G(6), H(7), J(7)]	[S(4), B(4), D(5), E(5)]
5	[K(5), A(6), C(6), F(6), G(6), H(7), J(7)]	[S(4), B(4), D(5), E(5), I(5)]
6	[L(5), A(6), C(6), F(6), G(6), H(7), J(7), M(7)]	[S(4), B(4), D(5), E(5), I(5), K(5)]
7	[A(6), C(6), F(6), G(6), H(7), J(7), M(7)]	[S(4), B(4), D(5), E(5), I(5), K(5), L(5)]
	L为目的状态，成功推出，结束搜索	



（括号内是路径总的耗散值f=g+h）

A算法的流程图


A算法流程图
开始
把S_0放入OPEN表中,计算$f(S_0)$
OPEN=NIL?
Y
失败
N
选取OPEN表中f最小的节点i放入CLOSED表中
i=G ?
Y
成功
N
扩展i得后继节点j,计算$f(j)$,提供返回i的指针,把节点j送回OPEN表，利用$f(j)$对OPEN表进行重排，调整亲子关系和指针。

A*算法（A星算法）

- 假设$f^*(n)$为从初始节点S_0出发，约束经过结点n到达目标节点的最小代价值。评价函数$f(n)$则是$f^*(n)$的估计值。
- $f^*(n)$由以下两部分所组成：
- 一部分是从初始结点S_0到结点n的最小代价，记为$g^*(n)$;
- 另一部分是从结点n到目标节点的最小代价，记为$h^*(n)$;
- 因此有$f*(n) = g^*(n) + h^*(n)$

A*算法

Open表中的节点按着估价函数$f(n)=g(n)+h(n)$的值从小到大排序。但是由于对启发函数h没有任何限制，A算法不能保证找到最优解。经研究发现，满足以下三个条件，能保证得到最优解。

1. $h(n)$是$h*(n)$的下界： $h(n) \le h^*(n)$（这样能满足启发函数的单调性，保证找到$f(n)$的最小值）。
2. 搜索空间中的每个节点都具有有限个后继。
3. 搜索空间的每个有向边的代价均为正值。

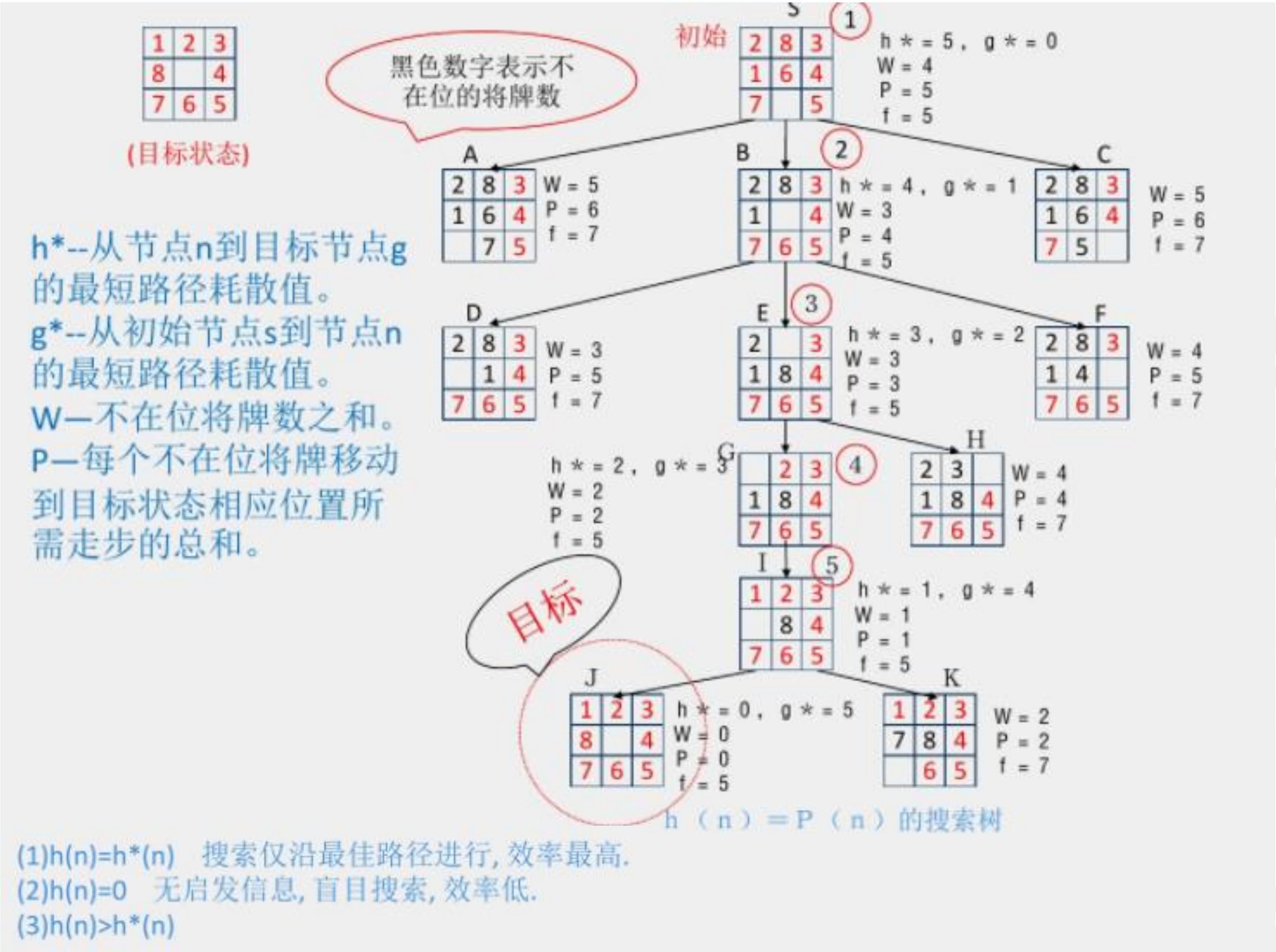
$g^*(n)$：初始结点到结点n的最小代价

$h^*(n)$：节点n到目标节点的最小代价；（一定存在）

一般来说$h^*(n)$并不知道，那么如何判断$h(n) \le h^*(n)$是否成立呢？这就要具体问题具体分析。例如，如果要找一条从A点到B点的距离最短的路径，我们可以用当前节点到目标节点的欧氏距离作为启发函数。虽然不知道$h^*(n)$，但由于两点间距离最近，肯定就满足$h(n) \le h^*(n)$。

A*算法

- *p(n)*为不在位的棋子与其目标位置的距离之和，因为两点间距离最短，则有 *p(n)* ≤ $h*(n)$，满足限制条件。
- w(n)是不在目标状态中相应位置的数码个数，w(n)也小于h*(n)，但是p(n)比w(n)更有启发性信息，因为前者构造的树比后者结点少，效率更高。
- 所以在这里选择 $h(n)=p(n)$



5.4 博弈搜索

- 博弈一向被认为是富有挑战力的活动，如下棋、打牌、作战、游戏等等。这里讲的博弈是二人博弈，博弈双方的利益是完全对立的。

（1）对垒的双方MAX和MIN轮流采取行动，博弈的结果只能有3种情况：MAX胜、MIN败；MAX败，MIN胜；和局。如果胜利记为+1分，失败为-1分，平局为0分，则双方在博弈结束时的总分为0，称此类博弈为**零和博弈**。

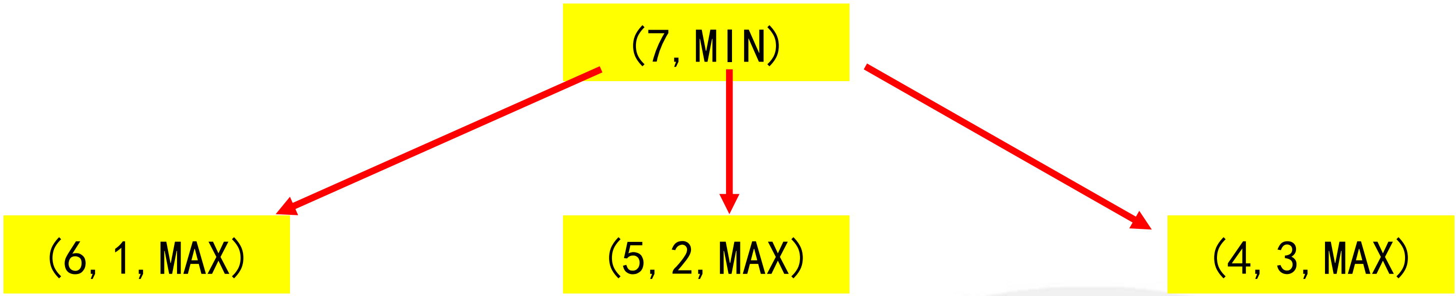
（2）在对垒过程中，任何一方都了解当前的格局和过去的历史。

（3）任何一方在采取行动前都要根据当前的实际情况，进行得失分析，选择对自己最为有利而对对方最不利的对策，不存在“碰运气”的偶然因素，即双方都很理智地决定自己的行动。

5.4 博弈搜索

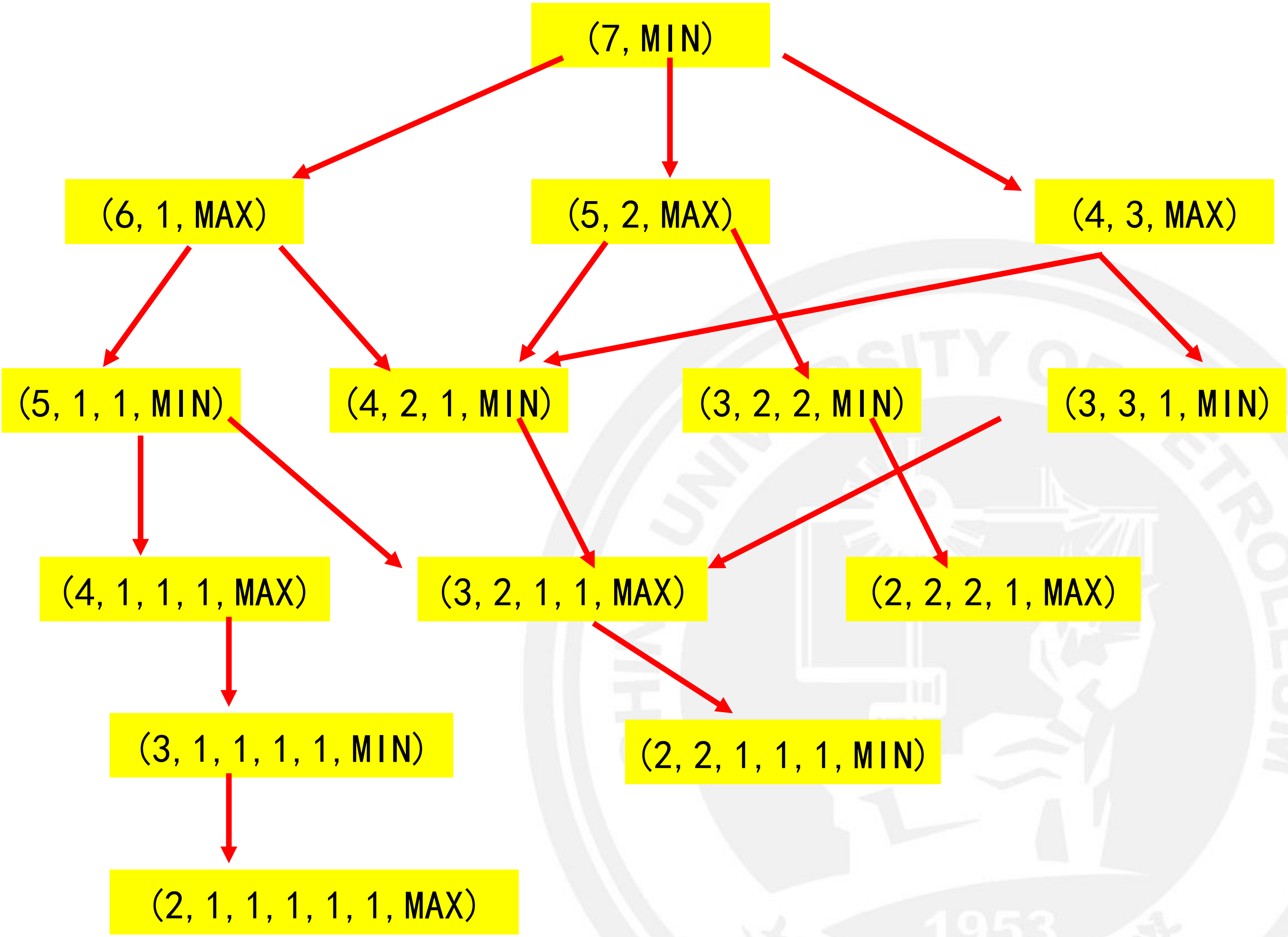
- **博弈例子——分钱币**：假设有7枚钱币，任一选手只能将已选好的一堆钱币分成两堆个数不等的，两位选手轮流进行，直到每一堆都只有一个或两个钱币不能再分为止，哪个选手遇到不能再分的情况则输。
- 用数字序列加上一个说明表示一个状态，其中数字表示不同堆中钱币的个数，说明表示下一步由谁来分，
- 如（7，MIN）表示只有一个由七枚钱币组成的堆，由MIN走，
- MIN有3种可供选择的分法，即（6，1，MAX），（5，2，MAX），（4，3，MAX），其中MAX表示另一选手，不论哪一种方法，MAX在它的基础上再作符合要求的划分。

5.4 博弈搜索



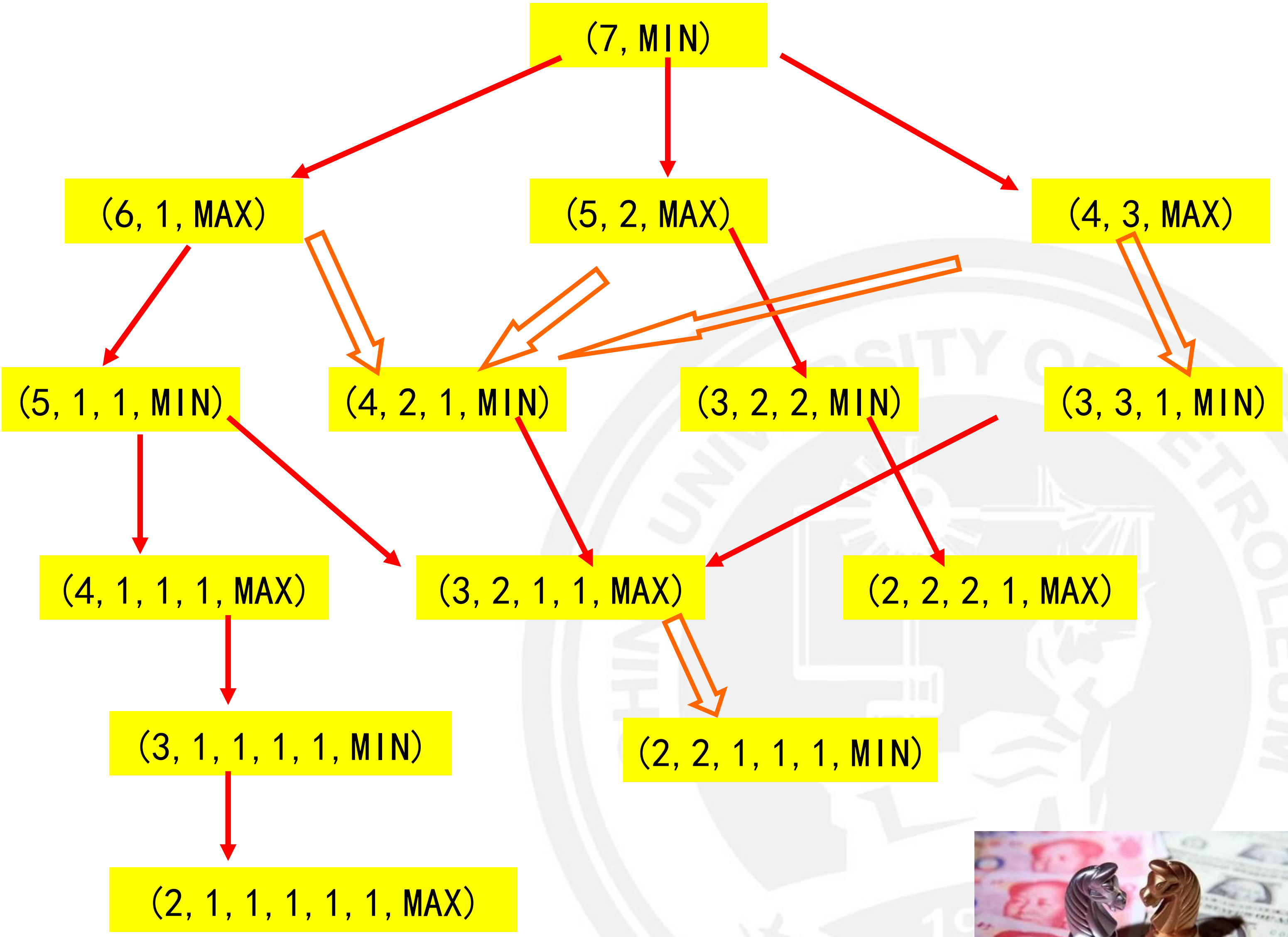
5.4 博弈搜索

- 图中已经将双方可能的方案全部表示出来了，而且从图中可以看出，无论MIN开始时怎么走，MAX总可以获胜。



5.4 博弈搜索

- 图中已经将双方可能的方案全部表示出来了，而且从图中可以看出，无论MIN开始时怎么走，MAX总可以获胜，取胜的策略就是粗箭头表示。
- 实际情况，对于任何一种棋，我们都不可能枚举出全部情况，因此，只能模拟人“向前看几步”，然后做决策，决定是否对自己最有利，然后按照一定的估算方法决定走哪一步棋。



5.4.1博弈—极大极小博弈

- 最常用的分析方法是极大极小分析法，其基本思想是：

① 设博弈的双方一方为MAX，另一方为MIN。然后设计算法为其中一方（例如MAX）寻找一个最优行动方案。

② 为了找到当前的最优行动方案，需要对各个可能的方案所产生的的后果进行比较并计算可能的得分。

③ 为了计算得分，需要根据问题的特性信息定义一个评价函数，用来估算当前博弈树末端结点的得分。

④ 当末端结点得分估算出来后，以博弈交替取MAX和MIN向上传递，直至计算出顶端结点得分。

⑤ 如果一个行动方案能获得较大的倒推值，则它就是当前最好的行动方案。

5.4.1博弈—极大极小博弈

- 极大极小过程是考虑双方对弈若干步之后，从可能的走法中选一步相对好的走法来走，即在有限的搜索深度范围内进行求解。

- 需要定义一个静态估价函数f,以便对棋局的态势做出评估。 这个函数可以根据棋局的态势特征进行定义。假定对弈双方分别为MAX和MIN，规定：
  - 有利于MAX方的态势：f(p)取正值
  - 有利于MIN方的态势：f(p)取负值
  - 态势均衡的时候：f(p)取零

  其中p代表棋局。

5.4.1博弈—极大极小博弈

- **MINMAX基本思想：**

(1)当轮到MIN走步的节点时，MAX应考虑最坏的情况（即f(p)取极小值）。

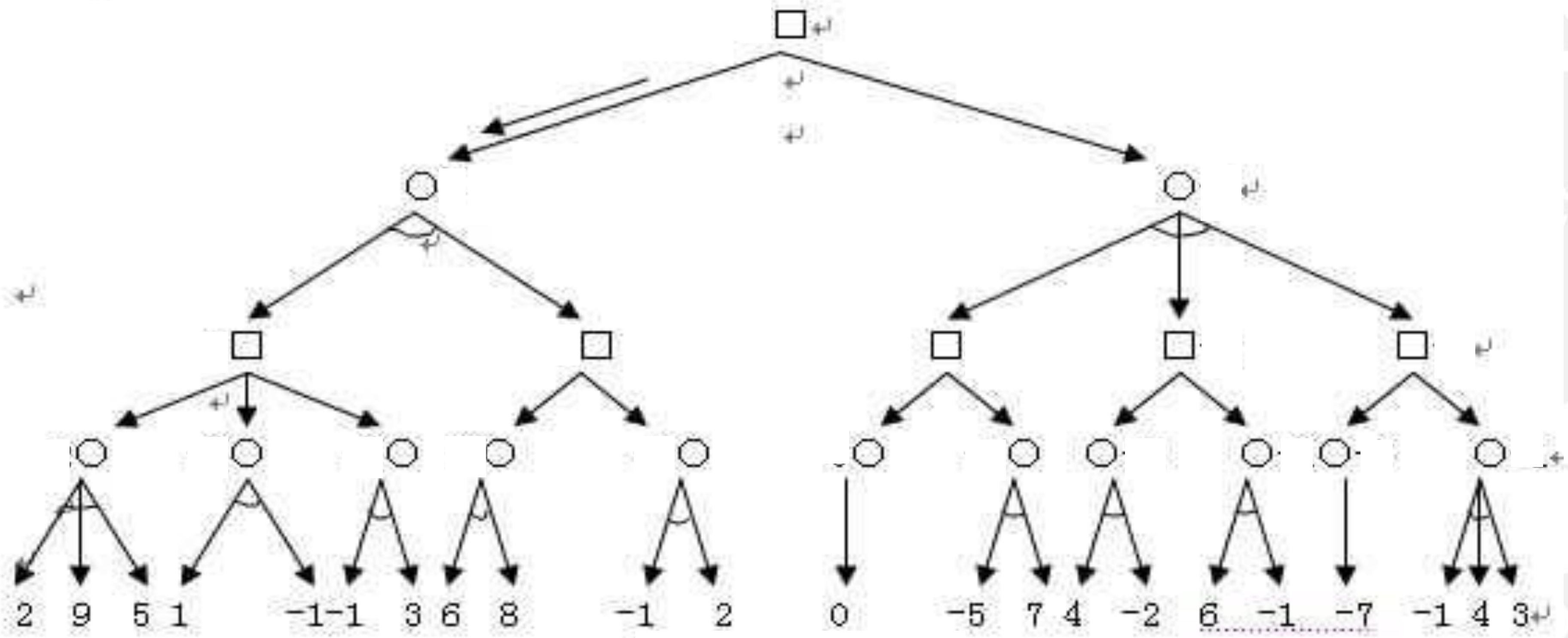
(2)当轮到MAX走步的节点时，MAX应考虑最好的情况（即f(p)取极大值）。

(3)评价往回倒推时，相应于两位棋手的对抗策略，交替使用（1）和（2）两种方法传递倒推值。

- 所以这种方法称为极大极小过程。

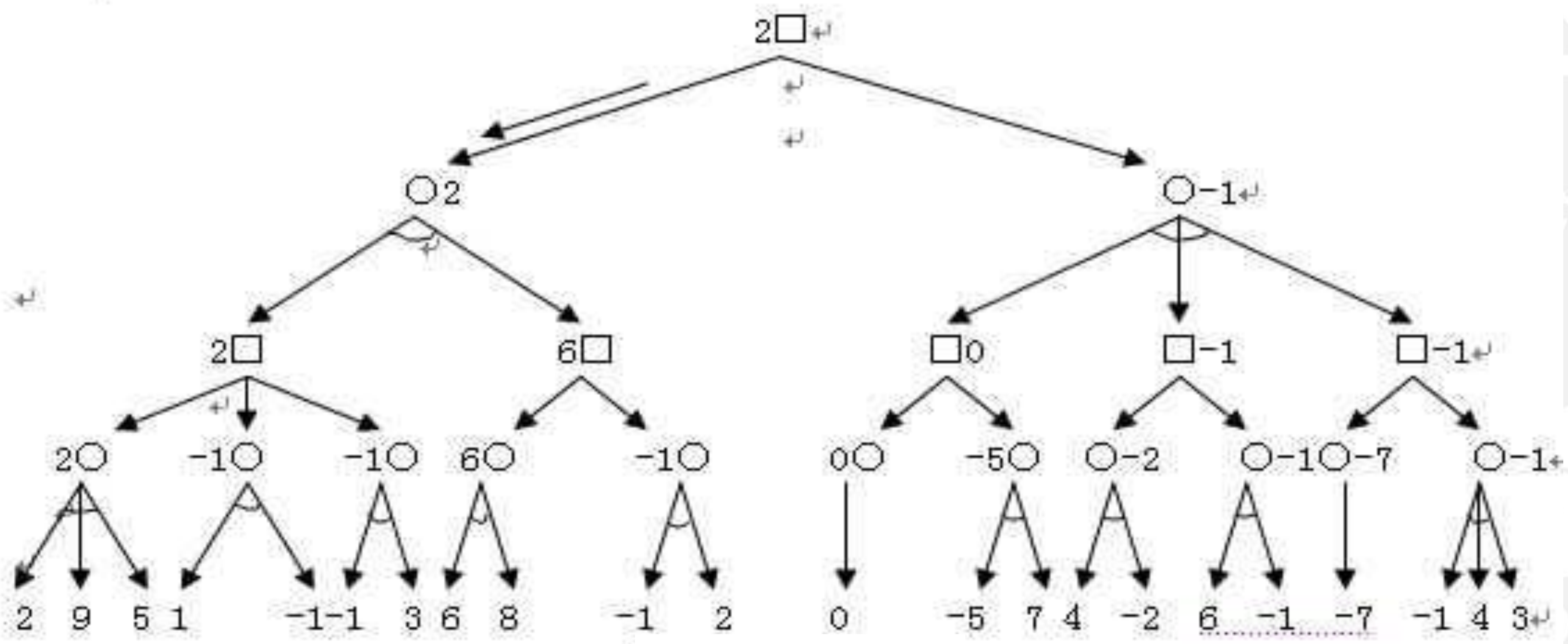
5.4.1 博弈—极大极小博弈

右图所示是向前看两步，共4层的博弈树，用□表示MAX，用○ 表示MIN，端结点上的数字表示它对应的评价函数的值。在MIN处用圆弧连接。



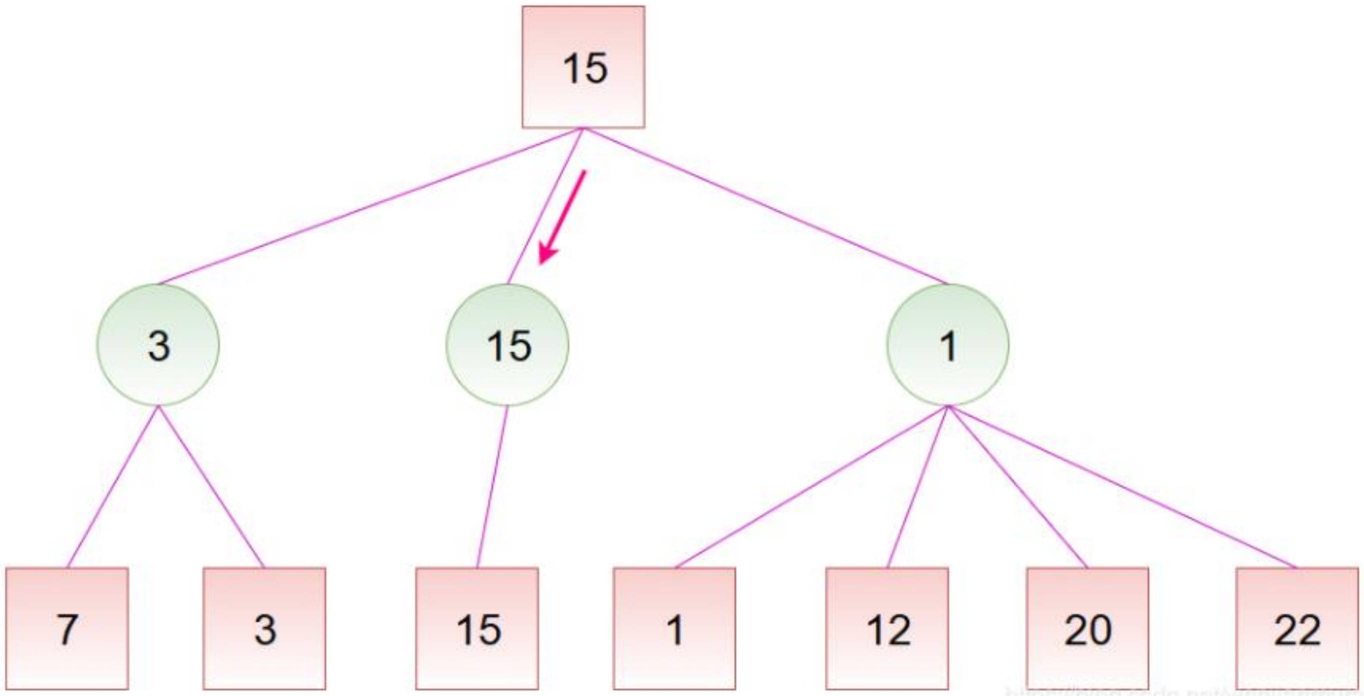
5.4.1 博弈—极大极小博弈

右图所示是向前看两步，共4层的博弈树，用□表示MAX，用○ 表示MIN，端结点上的数字表示它对应的评价函数的值。在MIN处用圆弧连接。



5.4.1 博弈—极大极小博弈

- 实际上，再看右边的路线时，当发现赢面可能为1就不必再去看赢面为12、20、22的分支了，因为已经可以确定右边的路线不是最好的。这个过程就是**剪枝**，可以避免不必要的计算。

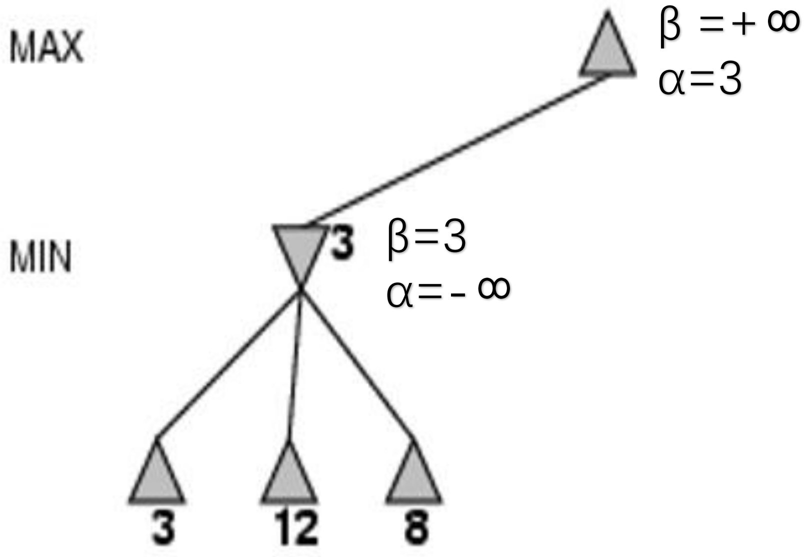


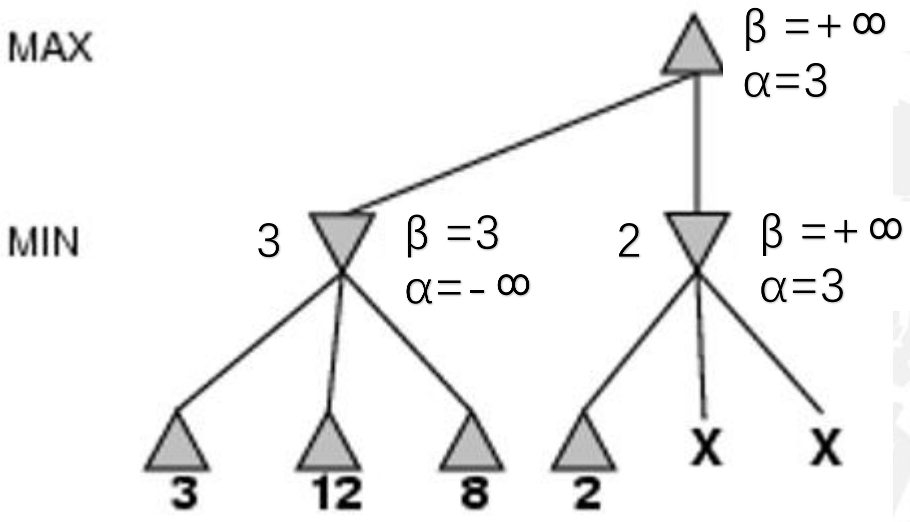
5.4.2 α–β剪枝算法

- 极大极小方法是把搜索树的生成和估值这两个过程完全分开. 只有在已经生成树之后才开始进行估值, 这一分离导致了低效率的策略
  - 游戏状态随游戏招数成指数增长
- α–β剪枝技术是极大极小方法的改进, 是一种提高博弈树搜索效率的方法
- α–β剪枝技术是一种边生成节点, 边计算估值和倒推值的方法, 从而剪去某些分枝.

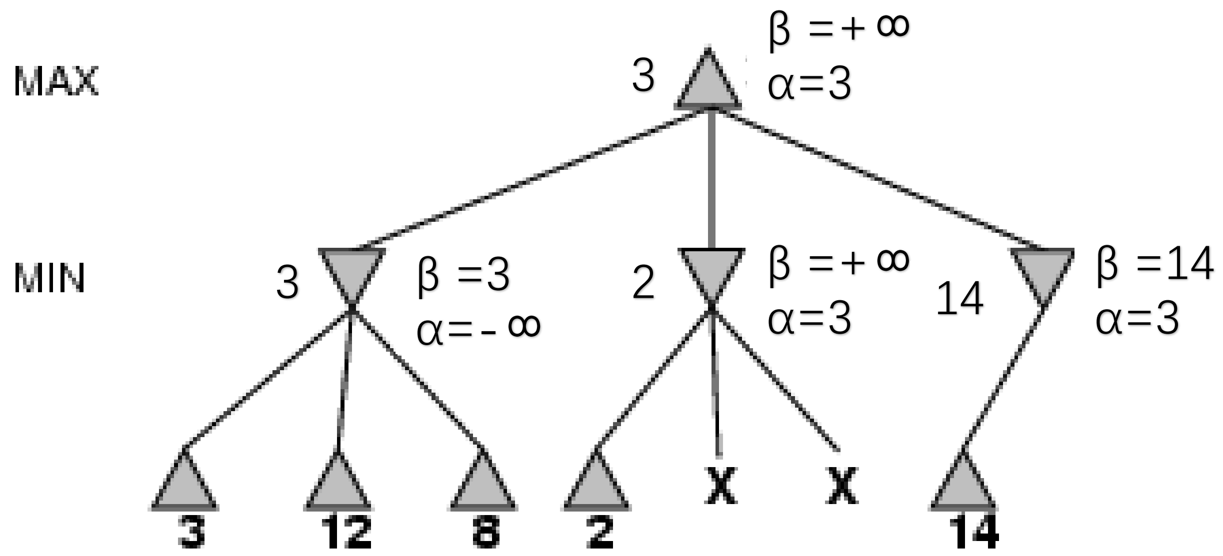
5.4.2 α-β剪枝算法

- 执行深度优先搜索直到第一个叶节点

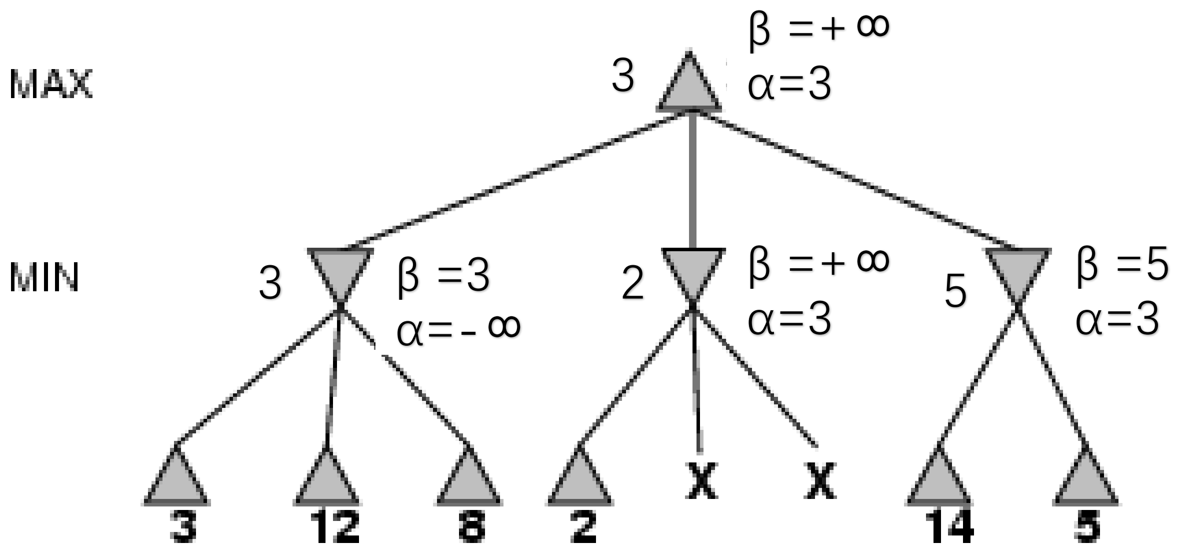




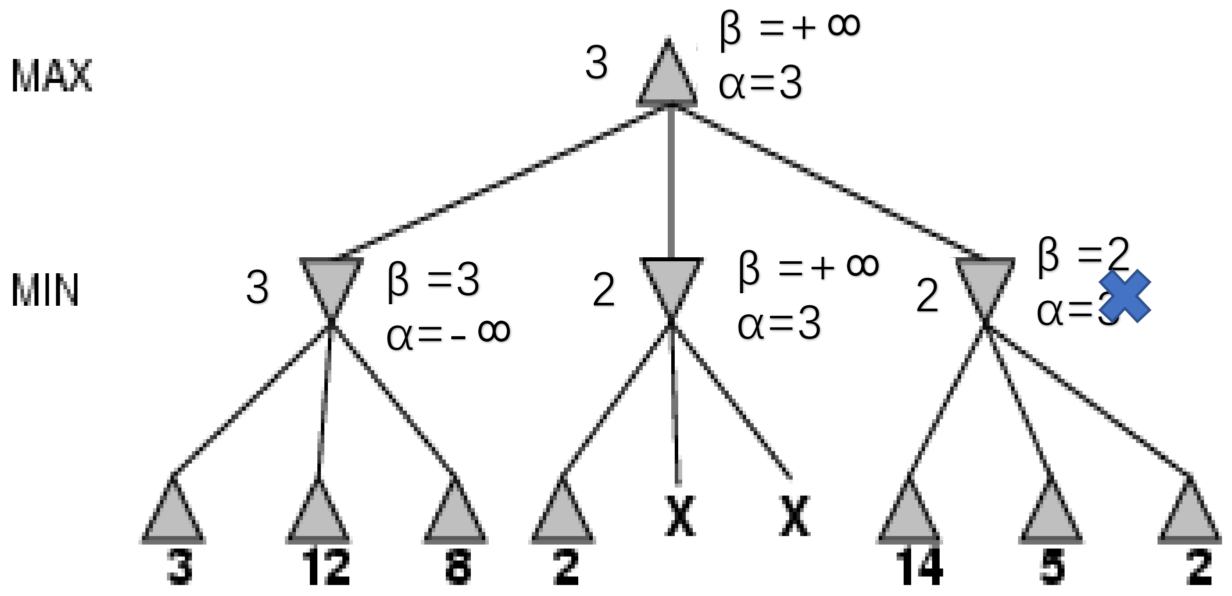
5.4.2 α−β剪枝算法



5.4.2 α−β剪枝算法

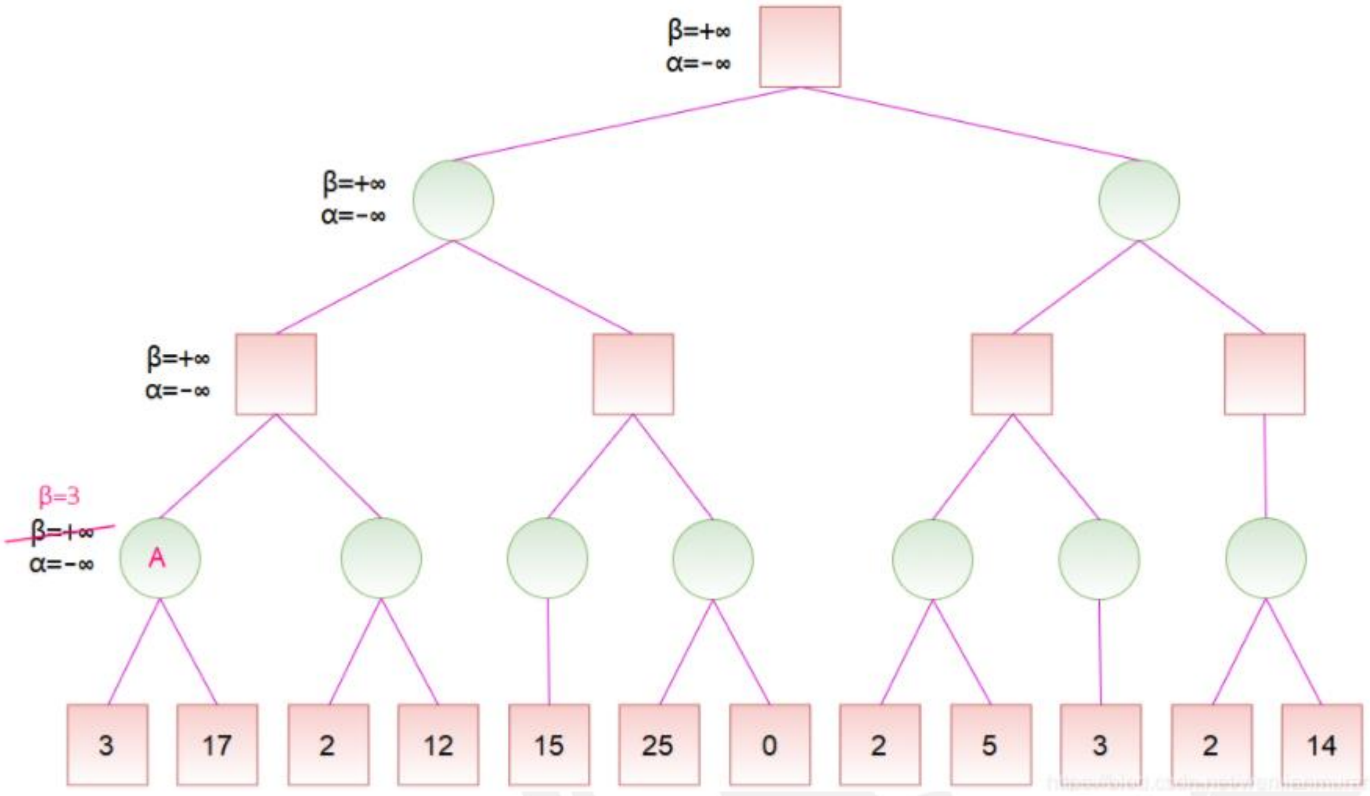


5.4.2 α-β剪枝算法



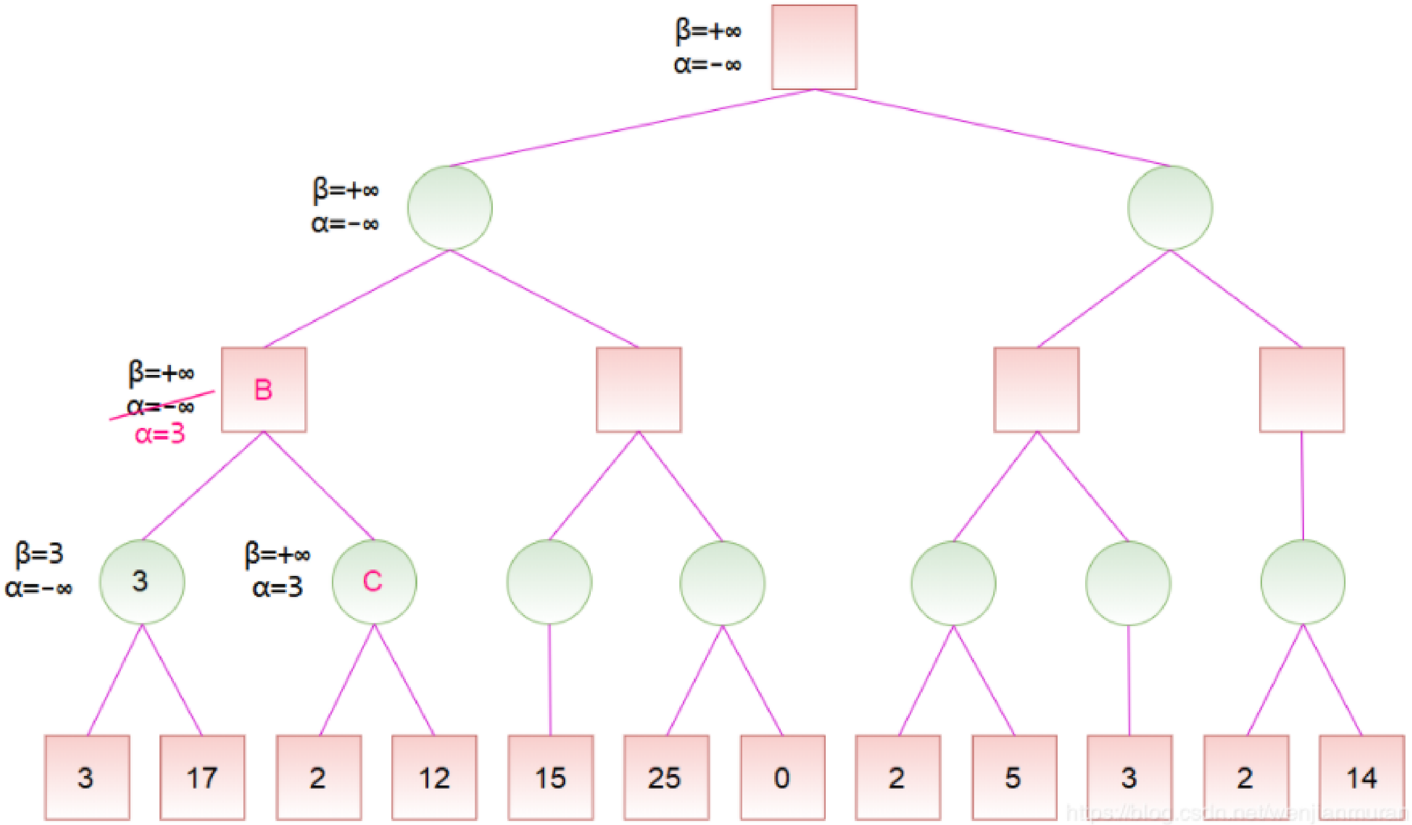
例题：

- 初始化时，令 α=−∞，β=+∞，也就是 −∞≤v≤+∞。到节点A时，由于左子节点的倒推值为3，而节点A是MIN节点，试图找倒推值小的走法，于是将 β值修改为3，这是因为3小于当前的β值（β=+∞）。然后节点A的右子节点的倒推值为17，此时不修改节点A的 β值，这是因为17大于当前的β值（β=3）。之后，节点A的所有子节点搜索完毕，即可计算出节点A的倒推值为3。



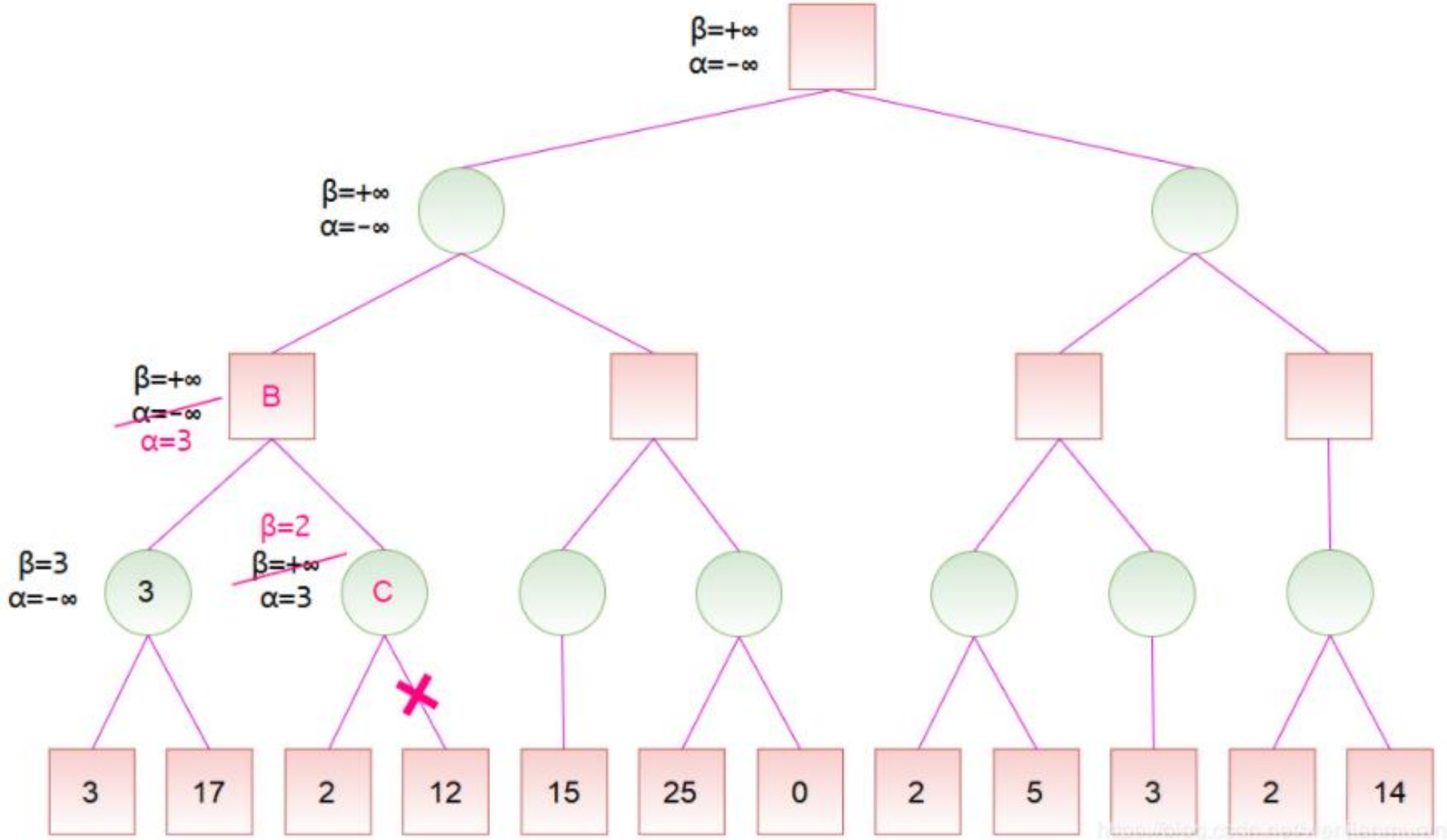
例题：

- 节点A是节点B的子节点，计算出节点A的倒推值后，可以更新节点B的倒推值范围（也就是α和β值）。由于节点B是MAX节点，试图找倒推值大的走法，于是将α值修改为3，这是因为3大于当前的α值（α=+∞）。之后搜索节点B的右子节点C，并将节点B的α和β值传递给节点C。



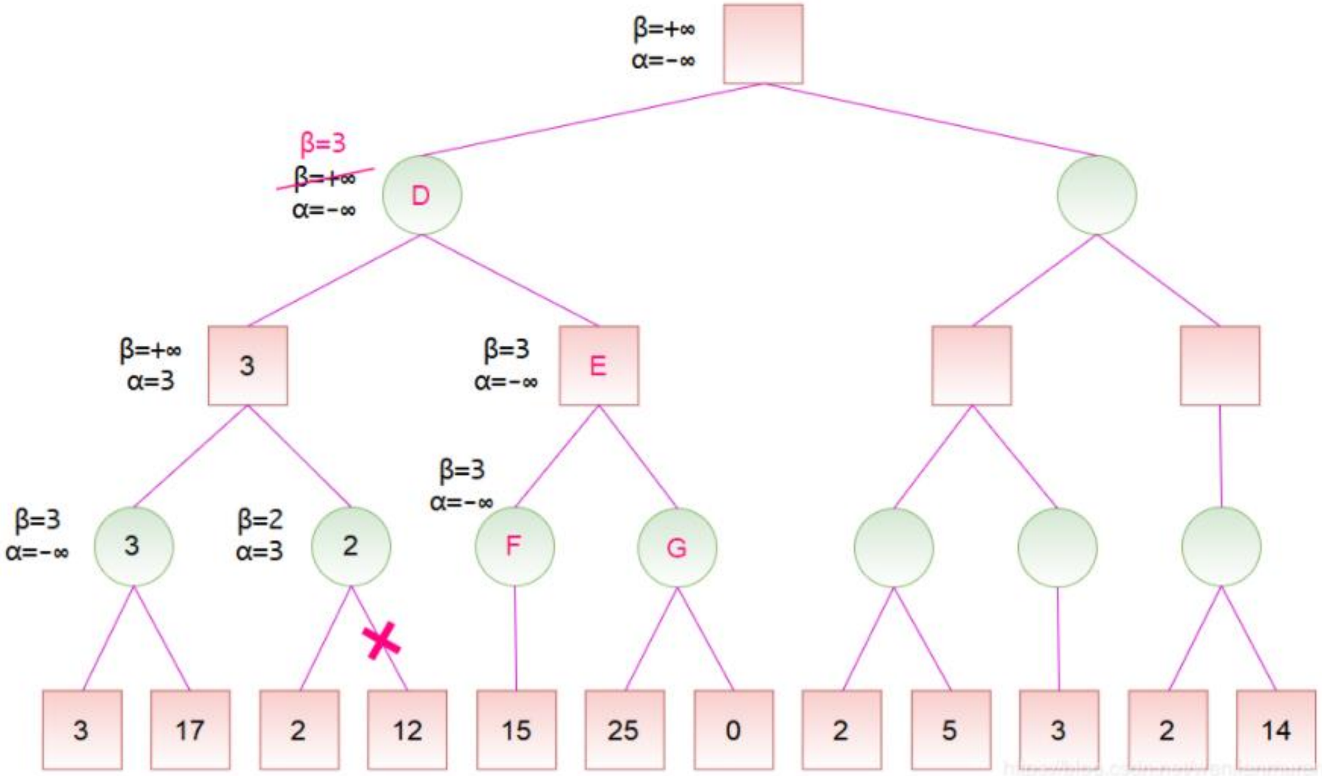
例题：

- 对于节点C，由于左子节点的倒推值为2，而节点C是MIN节点，于是将β值修改为2。此时α≥β，故节点C的剩余子节点就不必搜索了，因为可以确定，通过节点C并没有更好的走法。然后，节点C是MIN节点，将节点C的倒推值设为β，也就是2。由于节点B的所有子节点搜索完毕，即可计算出节点B的倒推值为3。



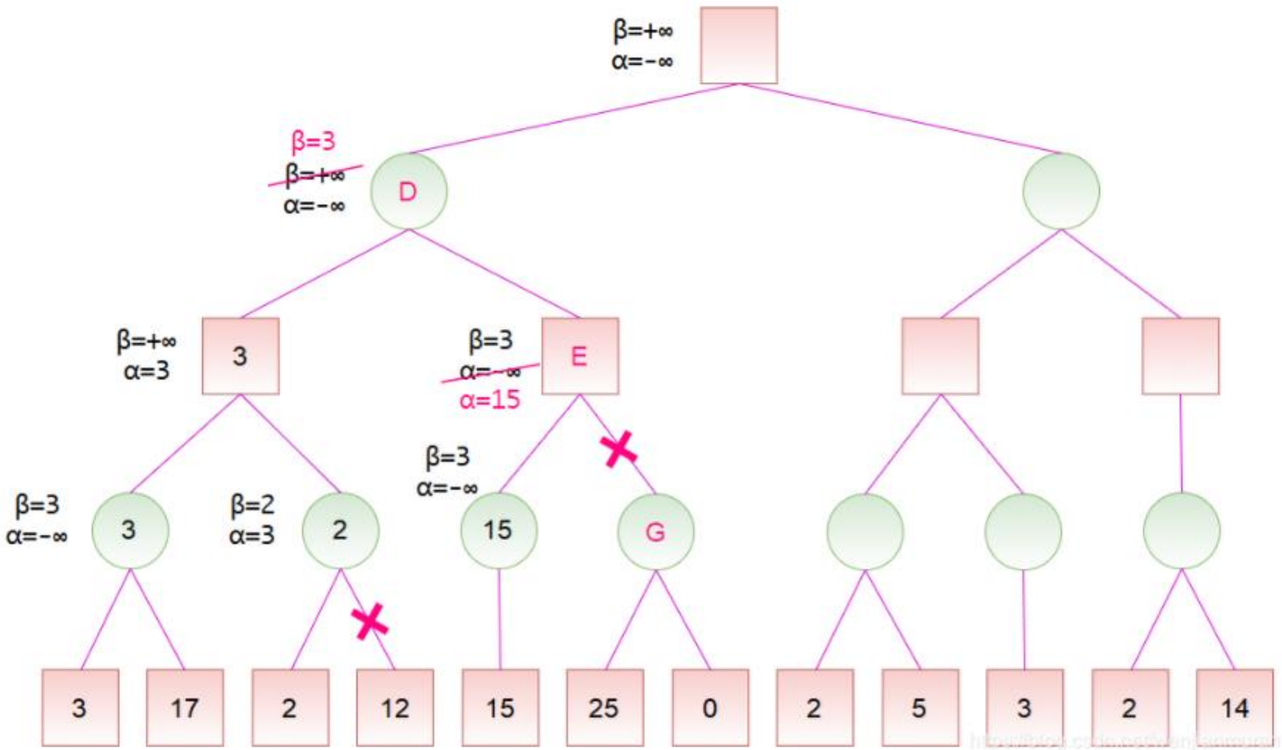
例题：

- 计算出节点B的倒推值后，节点B是节点D的一个子节点，故可以更新节点D的倒推值范围。由于节点D是MIN节点，于是将β值修改为3。然后节点D将α和β值传递给节点E，节点E又传递给节点F。对于节点F，它只有一个倒推值为15的子节点，由于15大于当前的β值，而节点F为MIN节点，所以不更新其β值，然后可以计算出节点F的倒推值为15。



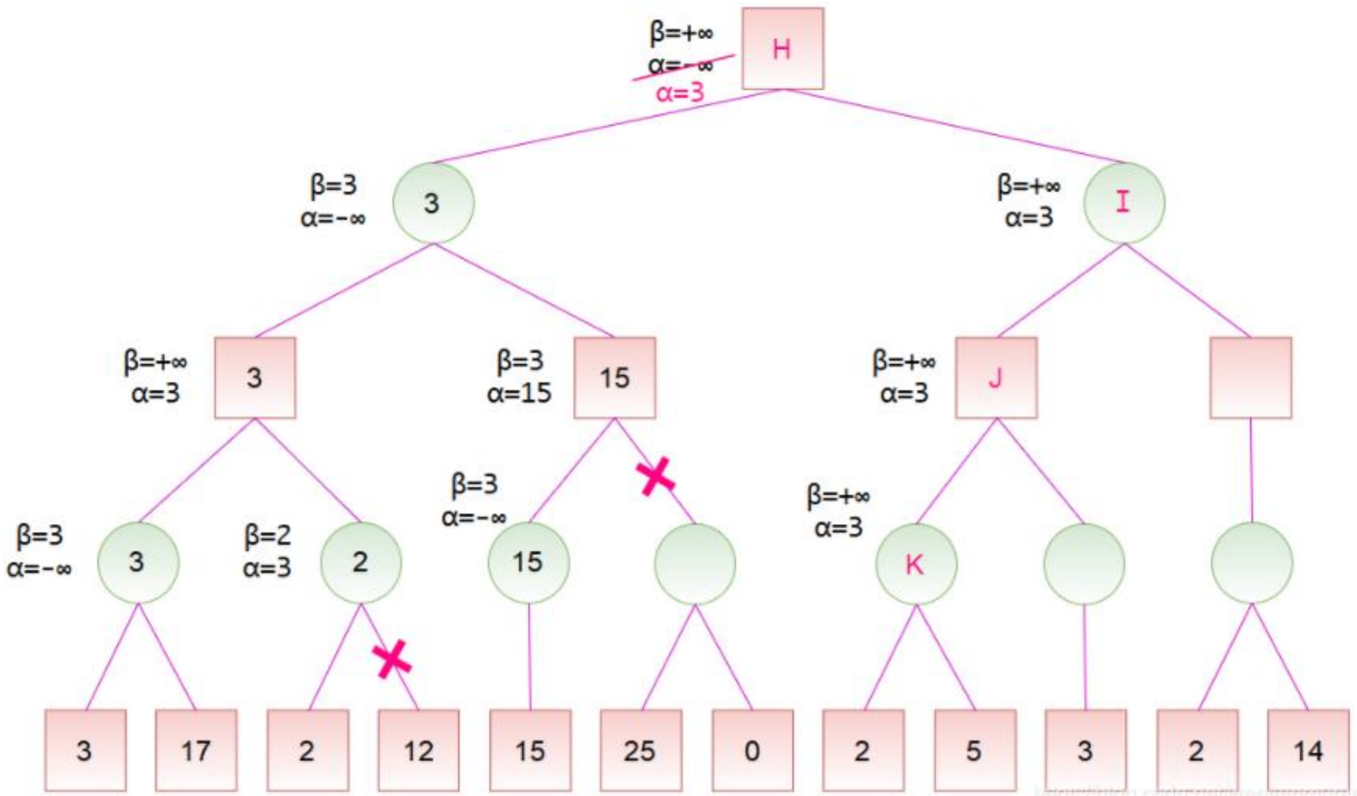
例题：

- 计算出节点F的倒推值后，节点F是节点E的一个子节点，故可以更新节点E的倒推值范围。节点E是MAX节点，更新α，此时α≥β，故可以剪去节点E的余下分支。然后，节点E是MAX节点，将节点E的倒推值设为α，也就是15。此时，节点D的所有子节点搜索完毕，即可计算出节点D的倒推值为3。



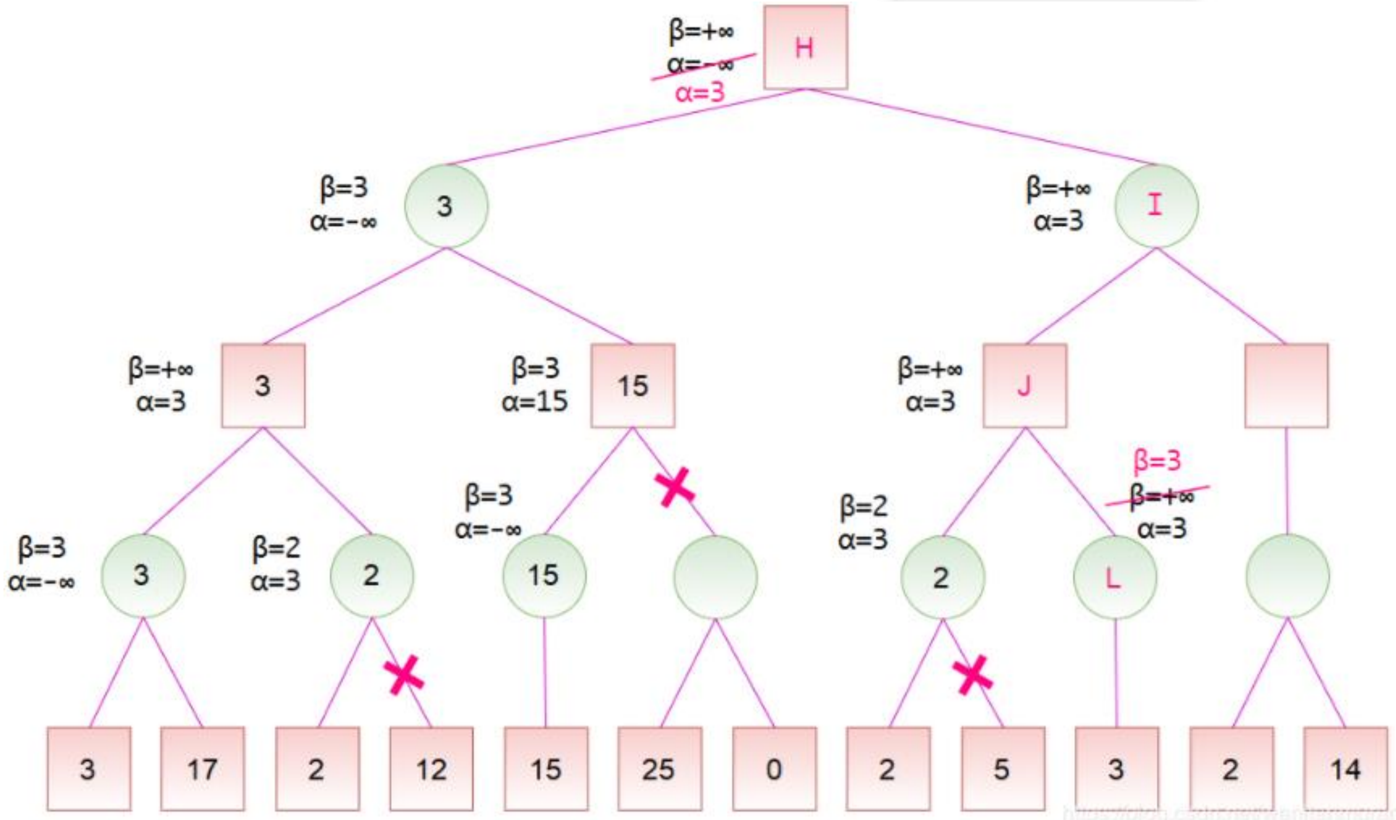
例题：

- 计算出节点D的倒推值后，节点D是节点H的一个子节点，故可以更新节点H的倒推值范围。节点H是MAX节点，更新α。然后，按搜索顺序，将节点H的α和β值依次传递给节点I、J、K。对于节点K，其左子节点的倒推值为2，而节点K是MIN节点，更新β，此时 α≥β，故可以剪去节点K的余下分支。然后，将节点K的倒推值设为2。



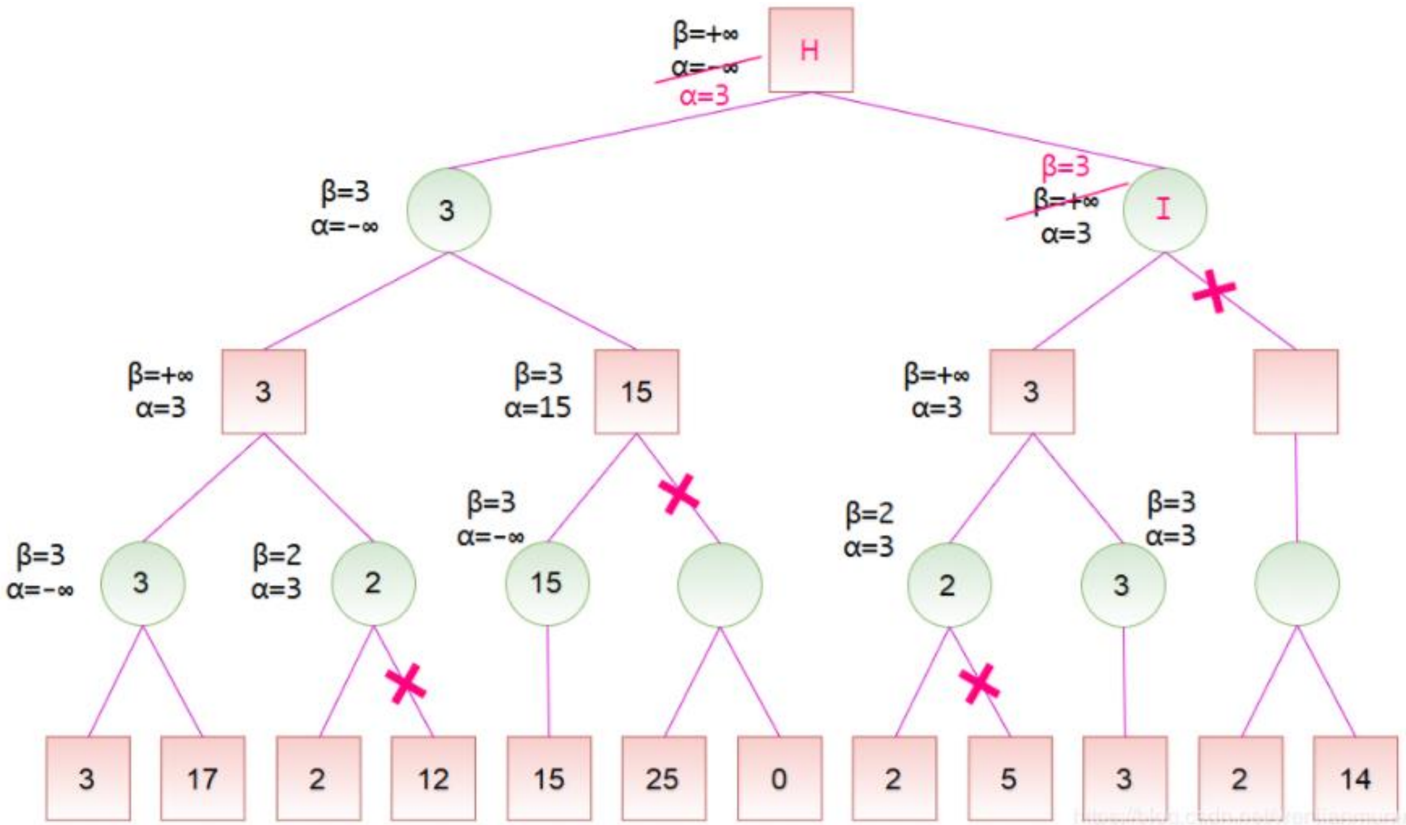
例题:

- 计算出节点K的倒推值后,节点K是节点J的一个子节点,故可以更新节点J的倒推值范围。节点J是MAX节点,更新α,
- 但是,由于节点K的倒推值小于α,所以节点J的α值维持3保持不变。然后,将节点J的α和β值传递给节点L。由于节点L是MIN节点,更新β=3,此时α≥β,故可以剪去节点L的余下分支,由于节点L没有余下分支,所以此处并没有实际剪枝。然后,将节点L的倒推值设为3。



例题：

- 此时，节点J的搜索子节点搜索完毕，计算出节点J的倒推值为3。由于节点J是节点I的子节点，故可以更新节点I的倒推值范围。节点I是MIN节点，更新 β=3，此时，α ≥ β，故可以剪去节点I的余下分支。然后，将节点I的倒推值设为3。
  至此，剪枝完成。



5.4 博弈搜索

- 20世纪40年代中期，S.M.乌拉姆和J.冯·诺伊曼提出**蒙特卡洛方法**。它是一类**随机模拟方法**的总称。

- 2006年，蒙特卡洛树搜索方法使围棋程序性能有了质的提高，在9路围棋（9×9大小的棋盘）上战胜了人类职业棋手。

5.4 博弈搜索

- **蒙特卡洛随机模拟方法**
  - 基本思想：对于当前棋局，随机地模拟双方走步直到分出胜负为止。通过多次模拟，计算每个可下棋点的获胜概率，选取获胜概率最大的点走棋。
  - 蒙特卡洛树搜索方法：边模拟边建立一个搜索树，父节点可以共享子节点的模拟结果，以提高搜索效率。过程分为：
    - 选择：以当前棋局为根节点，自上而下地选择一个落子点；
    - 扩展：向选定地节点添加一个或多个子节点；
    - 模拟：对扩展出的节点用蒙特卡洛方法进行模拟；
    - 回传：根据模拟结果依次向上更新祖先节点的估计值。

THANKS