重庆邮电大学

计算机科学与技术学院

人工智能原理

搜索求解

状态空间法举例2

N×N阶数码问题(如3数码、8数码、15数码等)

8数码问题（3×3阶数码问题）：

将分别标有数字1，2，3，…，8的八块正方形数码牌任意地放在一块3×3的数码盘上。放牌时要求不能重叠。在3×3的数码盘上有一个空格。现在要求按照每次只能将与空格相邻的数码牌与空格交换的原则，将任意摆放的数码盘逐步摆成某种特殊的排列。

2	8	3
1		4
7	6	5

1	2	3
8		4
7	6	5

* 练习题：
* 1.画8数码的状态空间图

初始状态：

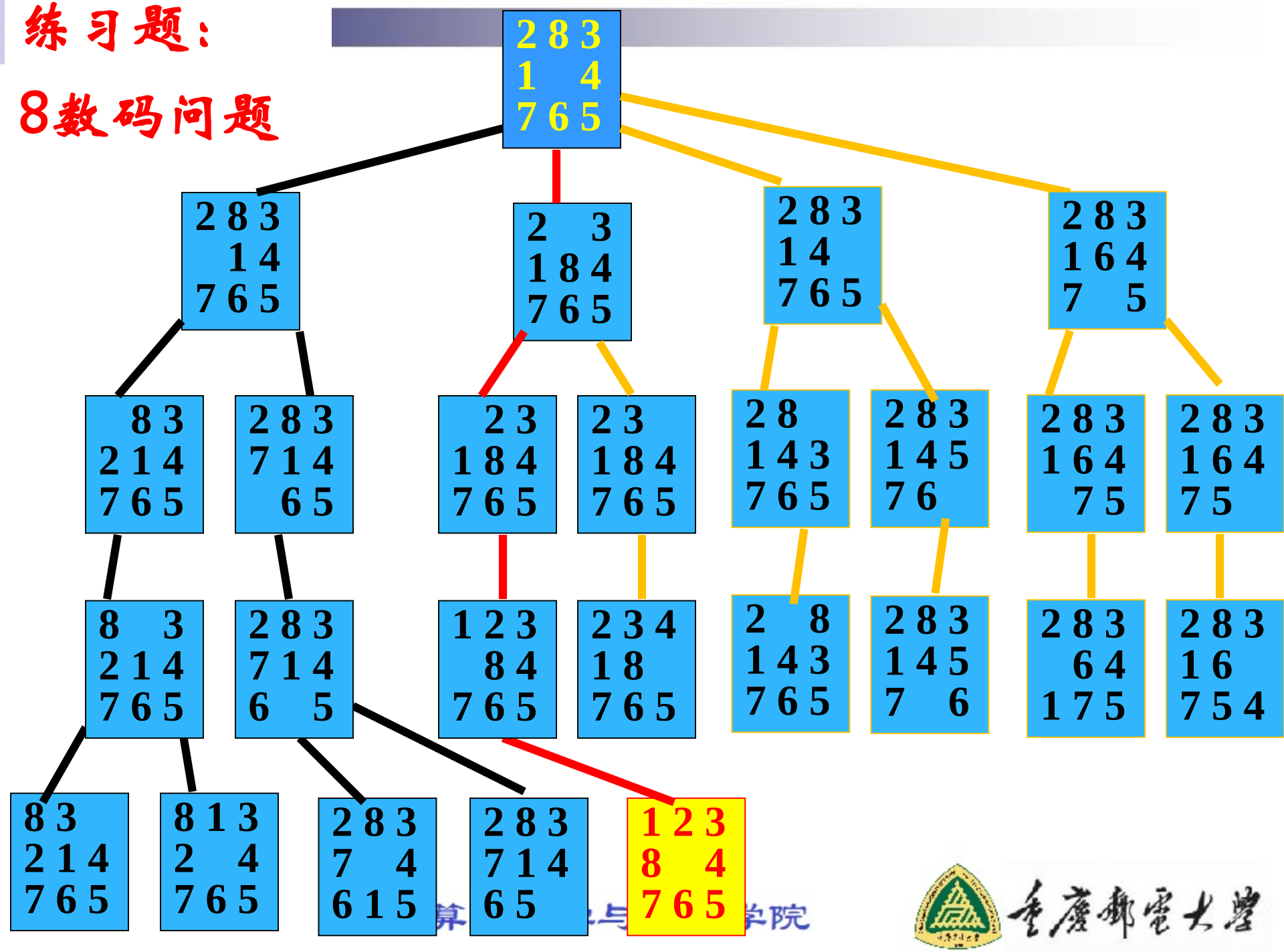
2	**8**	**3**
1		4
7	6	5

目标状态：

1	2	3
8		4
7	6	5

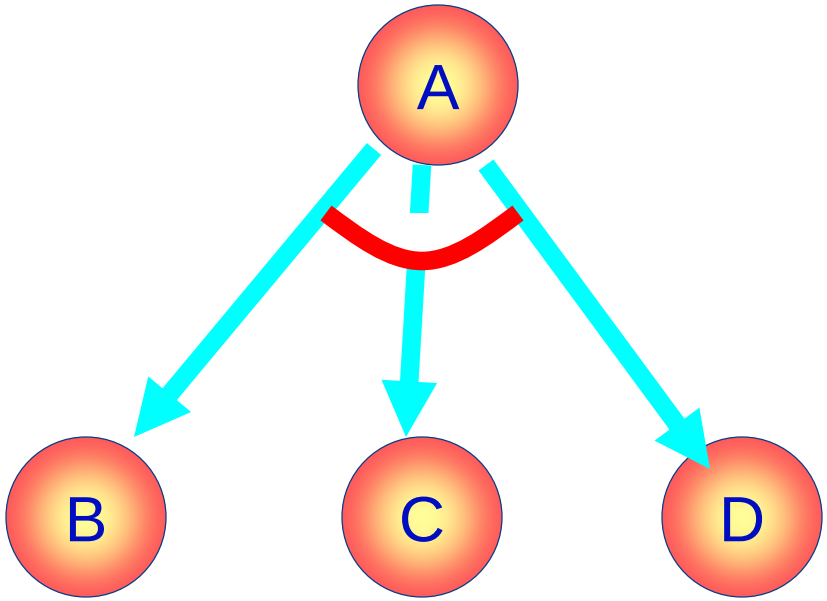
练习题：

8数码问题

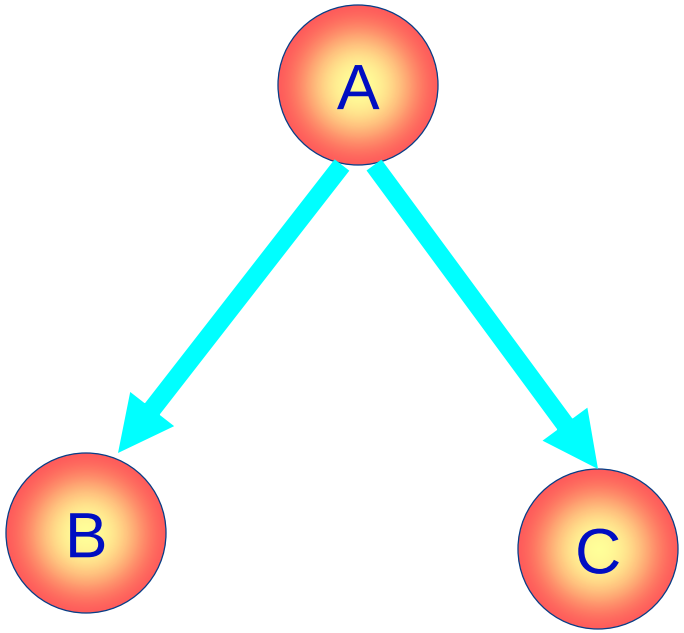

2 8 3
1 4
7 6 5
2 8 3
1 4
7 6 5
2 3
1 8 4
7 6 5
2 8 3
1 4
7 6 5
2 8 3
1 6 4
7 5
8 3
2 1 4
7 6 5
2 8 3
7 1 4
6 5
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
2 8
1 4 3
7 6 5
2 8 3
1 4 5
7 6
2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
8 3
2 1 4
7 6 5
2 8 3
7 1 4
6 5
1 2 3
8 4
7 6 5
2 3 4
1 8
7 6 5
2 8
1 4 3
7 6 5
2 8 3
1 4 5
7 6
2 8 3
6 4
1 7 5
2 8 3
1 6
7 5 4
8 3
2 1 4
7 6 5
8 1 3
2 4
7 6 5
2 8 3
7 4
6 1 5
2 8 3
7 1 4
6 5
1 2 3
8 4
7 6 5

与或图

* 1.与图、或图、与或图



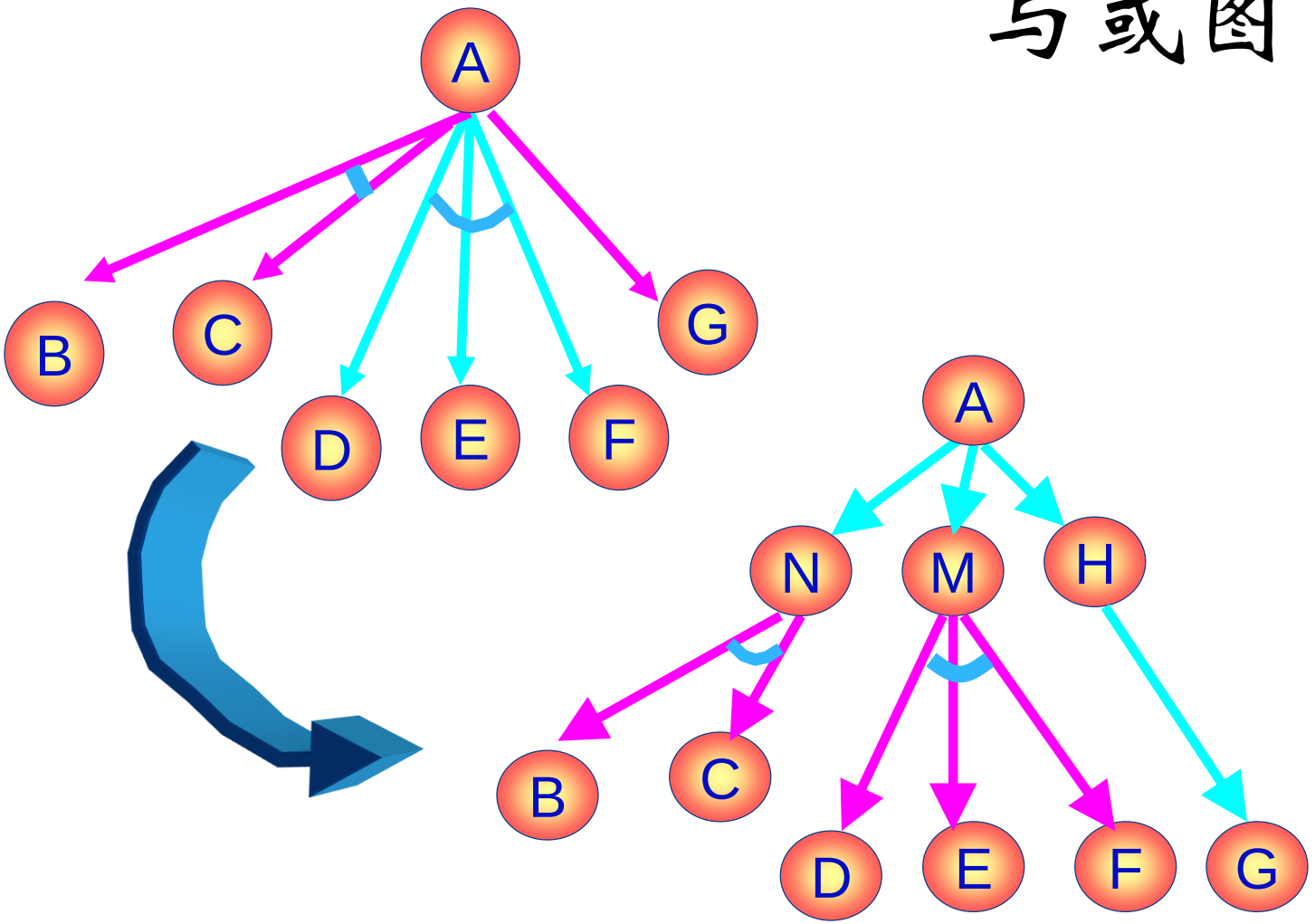
与图



或图

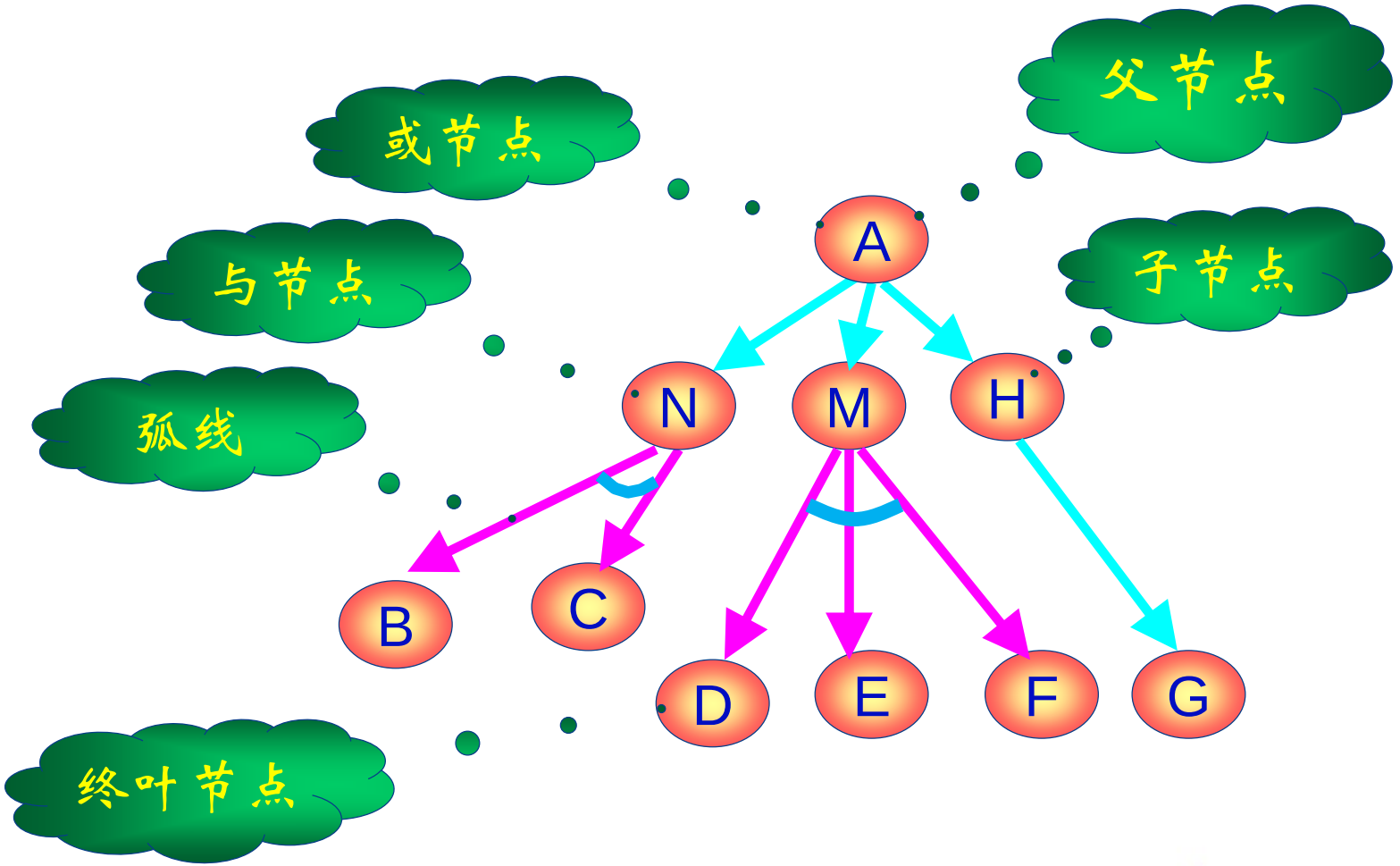
问题归约法

与或图



与或图

一些关于与或图的术语



终叶节点对应本原问题（有明显解答）

搜索策略

存放待搜索的
未扩展节点

搜索算法的几个重要名词

(1) OPEN表与CLOSE表

OPEN表

节点	父辈节点

CLOSED表

编号	节点	父辈节点

存放已搜索的
已扩展节点

搜索策略

搜索算法框架

(1) 建立一个只含有起始节点S的搜索图G，把S放到一个叫做OPEN的未扩展节点表中。

(2) 建立一个叫做CLOSED的已扩展节点表，其初始为空表。

(3) LOOP：若OPEN表是空表，则失败退出。

(4) 选择OPEN表上的第一个节点，把它从OPEN表移出并放进CLOSED表中。称此节点为节点n。

(5) 若n为一目标节点，则有解并成功退出，此解是追踪图G中沿着指针从n到S这条路径而得到的(指针将在第7步中设置)。

搜索策略

搜索算法框架

(6) 扩展节点n，同时生成不是n的祖先的那些后继节点的集合M。把M的这些成员作为n的后继节点添入图G中。

(7)对那些未曾在G中出现过的(既未曾在OPEN表上或CLOSED表上出现过的)M成员设置一个通向n的指针。把M的这些成员加进OPEN表。对已经在OPEN或CLOSED表上的每一个M成员，确定是否需要更改通到n的指针方向。对已在CLOSED表上的每个M成员，确定是否需要更改图G中通向它的每个后裔节点的指针方向。

(8) 按某一任意方式或按某个探试值，重排OPEN表。

(9) GO LOOP。

搜索策略

搜索方法分析：

搜索过程的第8步对OPEN表上的节点进行排序，以便能够从中选出一个“最好”的节点作为第4步扩展用。这种排序可以是任意的即盲目的(属于无信息搜索)，也可以用以后要讨论的各种启发思想或其它准则为依据(属于启发式搜索)。每当被选作扩展的节点为目标节点时，这一过程就宣告成功结束。这时，能够重现从起始节点到目标节点的这条成功路径，其办法是从目标节点按指针向S返回追溯。当搜索树不再剩有未被扩展的端节点时，过程就以失败告终(某些节点最终可能没有后继节点，所以OPEN表可能最后变成空表)。在失败终止的情况下，从起始节点出发，一定达不到目标节点。

无信息搜索

宽度优先搜索

定义 1 如果搜索是以接近起始节点的程度依次扩展节点的，那么这种搜索就叫做宽度优先搜索（breadth-first search)

深度优先搜索

定义 2 如果搜索是以扩展最新产生的节点，那么这种搜索就叫做深度优先搜索（depth-first search)

宽度优先搜索

宽度优先搜索

宽度优先搜索算法

(1) 把起始节点放到OPEN表中(如果该起始节点为一目标节点，则求得一个解答)。

(2) 如果OPEN是个空表，则没有解，失败退出；否则继续。

(3) 把第一个节点(节点n)从OPEN表移出，并把它放入CLOSED的扩展节点表中。

(4) 扩展节点n。如果没有后继节点，则转向上述第(2)步。

(5) 把n的所有后继节点放到OPEN队列的末端，并提供从这些后继节点回到n的指针。

(6) 如果n的任一个后继节点是个目标节点，则找到一个解答，成功退出；否则转向第(2)步。

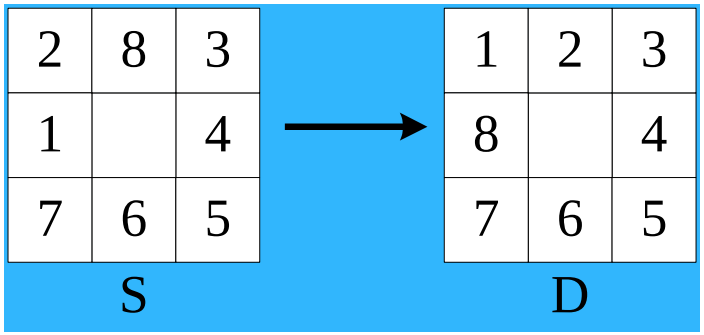
宽度优先搜索

宽度优先搜索

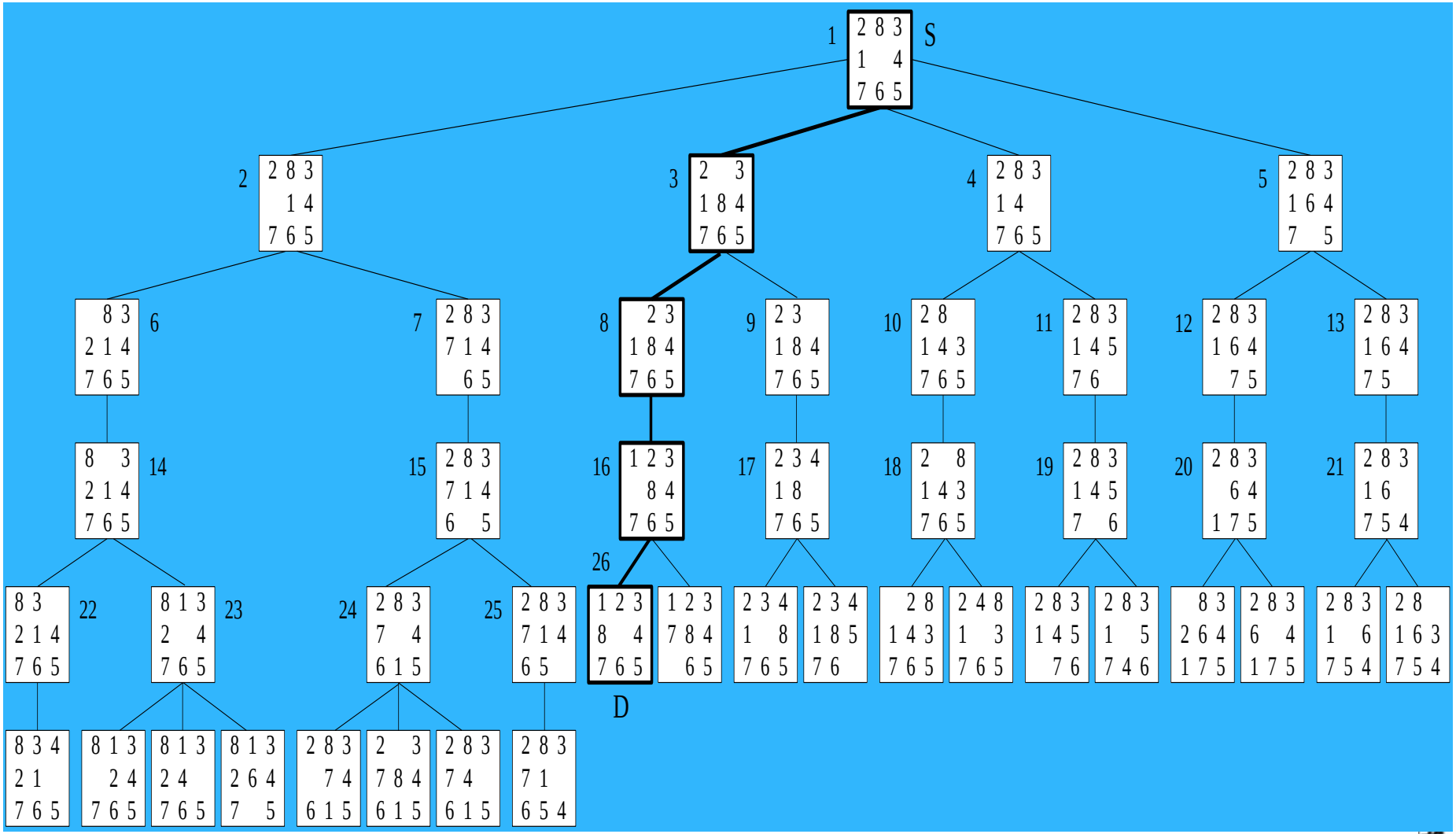
例 2 八数码问题

操作规定：允许空格四周上、下、左、右的数码块移入空格中，不许斜方向移动，不许返回先辈结点。

初始布局S和目标状态D如下图所示：



宽度优先搜索



节点展开按顺时针方向（开始：左）

宽度优先搜索： 搜索是以接近起始节点的程度依次扩展节点。

宽度优先搜索的特点：

- 无信息搜索（一种蛮力搜索算法）
- OPEN表为“先进先出”的队列实现
- 能够找到一条通向目标的最短路径
- 搜索树提供了所有可能的路径

深度优先搜索

深度优先搜索

深度优先算法步骤：

(1) 初始结点S放到未扩展节点OPEN中,CLOSE表置空；

(2) 若OPEN为空，则搜索失败，问题无解；

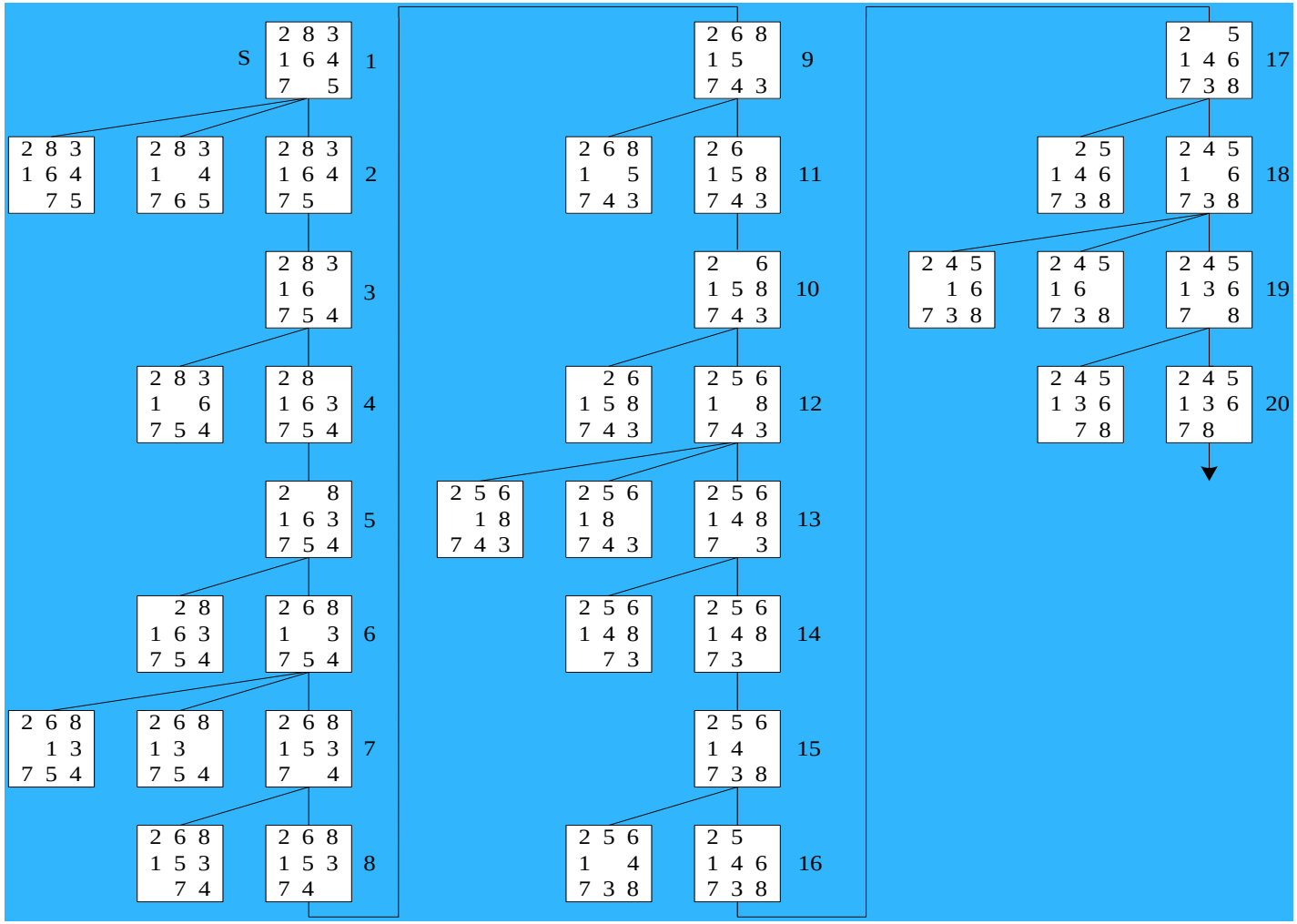
(3) 弹出OPEN表中最顶端结点放到CLOSE表中，并给出顺序编号n；

(4) 若n为目标结点D，则搜索成功，问题有解；

(5) 若n无子结点，转(2)；

(6) 扩展n结点，将其所有子结点配上返回n的指针，并按次序压入OPEN堆栈，转(2) 。

深度优先搜索


S
2 8 3
1 6 4
7 5
1
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 6 4
7 5
2
2 8 3
1 6
7 5 4
3
2 8 3
1 6
7 5 4
2 8
1 6 3
7 5 4
4
2 8
1 6 3
7 5 4
2 8
1 6 3
7 5 4
5
2 8
1 6 3
7 5 4
2 6 8
1 3
7 5 4
6
2 6 8
1 3
7 5 4
2 6 8
1 3
7 5 4
2 6 8
1 5 3
7 4
7
2 6 8
1 5 3
7 4
2 6 8
1 5 3
7 4
8
2 6 8
1 5
7 4 3
9
2 6 8
1 5
7 4 3
2 6 8
1 5
7 4 3
2 6
1 5 8
7 4 3
11
2 6
1 5 8
7 4 3
2 6
1 5 8
7 4 3
10
2 6
1 5 8
7 4 3
2 5 6
1 8
7 4 3
12
2 5 6
1 8
7 4 3
2 5 6
1 8
7 4 3
2 5 6
1 4 8
7 3
13
2 5 6
1 4 8
7 3
2 5 6
1 4 8
7 3
14
2 5 6
1 4
7 3 8
15
2 5 6
1 4
7 3 8
2 5
1 4 6
7 3 8
16
2 5
1 4 6
7 3 8
17
2 5
1 4 6
7 3 8
2 4 5
1 6
7 3 8
18
2 4 5
1 6
7 3 8
2 4 5
1 6
7 3 8
2 4 5
1 3 6
7 8
19
2 4 5
1 3 6
7 8
2 4 5
1 3 6
7 8
20

深度优先搜索

深度优先搜索

- 沿着某条单一路径搜索，只有没有后继才考虑替代；
- OPEN表作为“后进先出”的栈进行操作，最新节点先展开；
- 所求解的路径并一定是最短路径；
- 许多应用问题，搜索树可能无限深或超出搜索路径深度上限；
- **有界深度优先搜索**：

引入搜索深度限制值d，使深度优先搜索过程具有完备性 。
沿着一条路径搜索下去直到深度界限回溯到上一个节点。

有界深度优先搜索

深度优先搜索-有界深度优先算法步骤：

(1)初始结点S放入堆栈OPEN中；

(2)若OPEN为空，则搜索失败，问题无解；

(3)弹出OPEN中栈顶结点n，放入CLOSE表中，并给出顺序编号n；

(4)若n为目标结点D，则搜索成功，问题有解；

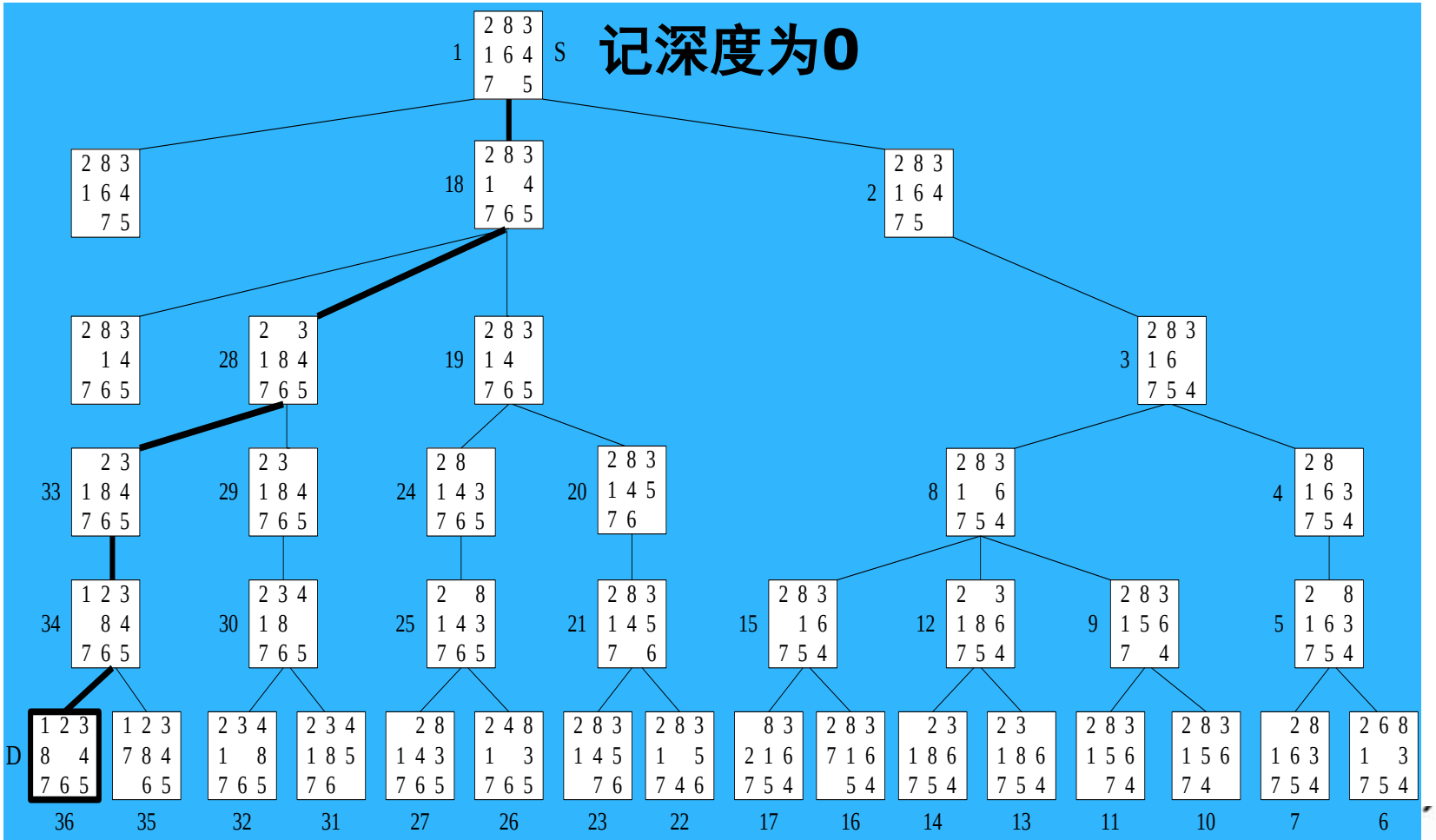
(5)若n的深度d(n)=d，则转(2)；

(6)若n无子结点，即不可扩展，转(2)；

(7)扩展结点n，将其所有子结点配上返回n的指针，并压入OPEN堆栈，转(2)。

深度优先搜索

有界深度优先搜索（下图设置深度界限5）



等代价搜索

等代价搜索

宽度优先搜索可被推广用来解决寻找从起始状态至目标状态的具有最小代价的路径问题，这种推广了的宽度优先搜索算法叫做等代价搜索算法。

等代价搜索：

起始节点记为S；

从节点i到它的后继节点j的连接弧线代价记为c(i，j)；

起始节点S到任一节点i的路径代价记为g(i)。

每次从OPEN表中选择代价g(i)最小节点进行扩展。

每次扩展放入OPEN表的后继节点j代价g(j)= g(i)+ c(i，j)

如果所有的连接弧线具有相等的代价，那么等代价算法就简化为宽度优先搜索算法。

启发式搜索(有信息搜索)

启发式搜索：利用问题域特性的信息（启发信息/辅助信息）进行搜索。

1 启发式搜索策略

启发式信息按用途分为三种：

(1) 选择最有希望的节点来确定要扩展的下一个节点，避免盲目扩展。

(2) 在扩展一个节点的过程中，用于确定要生成哪一个或哪几个后继节点，避免盲目生成所有可能节点。

(3) 用于确定某些应该从搜索树中抛弃或修剪的节点。

评价函数（evaluation function）：估算节点“希望”的量度。 通常用f(n) 表示节点n的估价函数值。

启发函数h(n)：从节点n到某目标节点的一条最佳路径的预期代价，依赖于有关问题的领域的启发信息。

建立估价函数的一般方法：（1）试图确定一个处在最佳路径上的节点的概率；（2）提出任意节点与目标集之间的距离量度或差别量度；（3）在棋盘式的博弈和难题中根据棋局的某些特点来决定棋局的得分数。

启发式搜索(有信息搜索)

辅助信息	所求解问题之外、与所求解问题相关的特定信息或知识
评价函数（evaluation function）**f(n)**	从当前节点n出发，根据评价函数来选择后续节点
启发函数（heuristic function）**h(n)**	计算从节点n到目标节点之间所形成路径的最小代价值。这里将两点之间的直线距离作为启发函数。

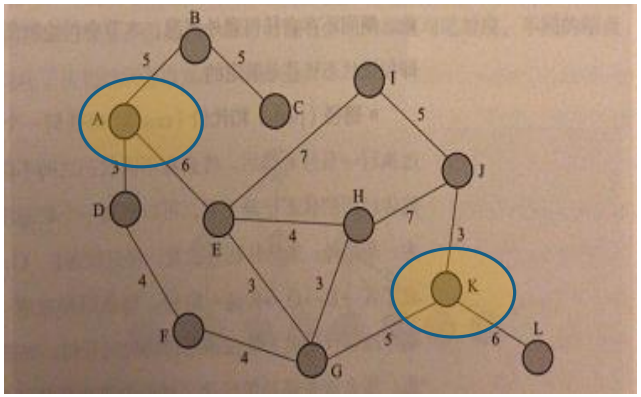


表3.2 每个状态（城市）所对应启发函数取值

状态	A	B	C	D	E	F	G	H	I	J	K	L
h(n)	13	10	6	12	7	8	5	3	6	3	0	6

辅助信息（启发函数）：
任意一个城市与终点城市K之间的直线距离

问题：寻找从城市A到城市K之间行驶时间最短路线？

启发式搜索（有信息搜索）

有序搜索

用评价函数f(n)来对图搜索策略循环最后步骤排列OPEN表上的节点。应用某个算法(例如等代价算法)选择OPEN表上具有最小f值的节点作为下一个要扩展的节点。这种搜索方法叫做**有序搜索**或**最佳优先搜索(best-first search)**，而其算法就叫做**有序搜索算法**或**最佳优先算法**。

有序搜索

有序状态空间搜索算法：

(1) 把起始节点S放到OPEN表中，计算f(S)并把其值与节点S联系起来。

(2) 如果OPEN是个空表，则失败退出，无解。

(3) 从OPEN表中选择一个f值最小的节点i。结果有几个节点合格，当其中有一个为目标节点时，则选择此目标节点，否则就选择其中任一个节点作为节点i。

(4) 把节点i从OPEN表中移出，并把它放入CLOSED的扩展节点表中。

(5) 如果i是个目标节点，则成功退出，求得一个解。

有序搜索

(6) 扩展节点i，生成其全部后继节点。对于i的每一个后继节点j：

(a) 计算f(j)。

(b) 如果j既不在OPEN表中，又不在CLOSED表中，则用评价函数f把它添入OPEN表。使j指向其父辈节点i的指针，以便一旦找到目标节点时记住一个解答路径。

(c) 如果j已在OPEN表上或CLOSED表上，则比较刚刚对j计算过的f值和前面计算过的该节点在表中的f值。如果新的f值较小，则

(i) 以此新值取代旧值。

(ii) 从j指向i，而不是指向它的父辈节点。

(iii) 如果节点j在CLOSED表中，则把它移回OPEN表。

(7) 转向(2)，即GO TO(2)。

有序搜索

宽度优先搜索、等代价搜索和深度优先搜索统统是有序搜索技术的特例。对于宽度优先搜索，选择f(i)作为节点i的深度。对于等代价搜索，f(i)是从起始节点至节点i这段路径的代价。

有序搜索的有效性直接取决于f(n)的选择，如果选择的f(n)不合适，有序搜索就可能失去一个最好的解甚至全部的解。如果没有适用的准确的希望量度，那么f的选择将涉及两个方面的内容：一方面是一个时间和空间之间的折衷方案；另一方面是保证有一个最优的解或任意解。

贪婪最佳优先搜索

贪婪最佳优先搜索：评价函数f(n)=启发函数h(n)

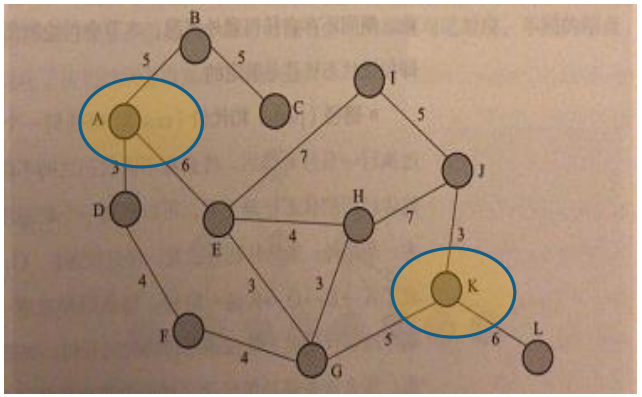
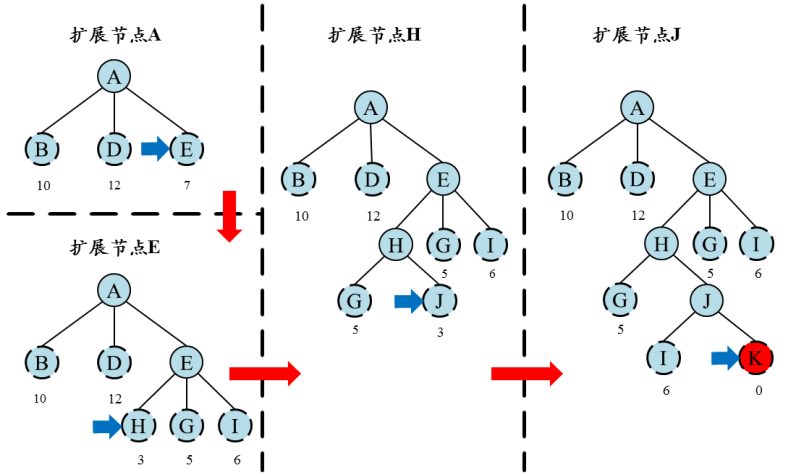


表3.2 每个状态（城市）所对应启发函数取值

状态	A	B	C	D	E	F	G	H	I	J	K	L
h(n)	13	10	6	12	7	8	5	3	6	3	0	6

问题：寻找从城市A到城市K之间行驶时间最短路线？

辅助信息（启发函数）：
任意一个城市与终点城市K之间的直线距离



算法找到了一条从起始节点到终点节点的路径A → E → H → J → K，但这条路径并不是最短路径，实际上最短路径为A → E → G → K。

A*算法

定义评价函数： $f(n) = g(n) + h(n)$

- $g(n)$**表示从起始节点到节点**n**的开销代价值，**$h(n)$**表示从节点**n**到目标节点路径中所估算的最小开销代价值。**
- $f(n)$**可视为经过节点**n**、具有最小开销代价值的路径。**

$$\underbrace{f(n)}_{\text{评价函数}} = \underbrace{g(n)}_{\substack{\text{起始节点到节点}n\text{代价} \\ \text{(当前最小代价)}}} + \underbrace{h(n)}_{\substack{\text{节点}n\text{到目标结点代价} \\ \text{(后续估计最小代价)}}}$$

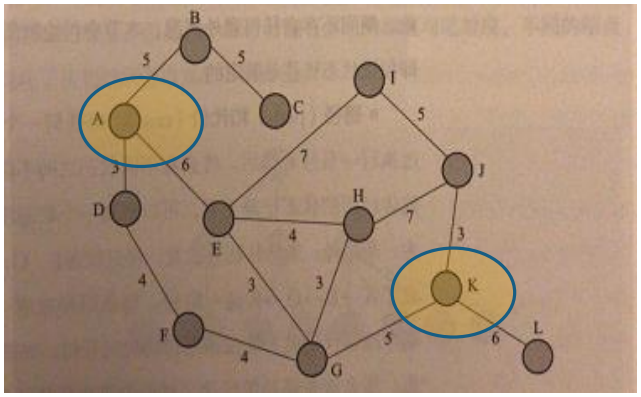


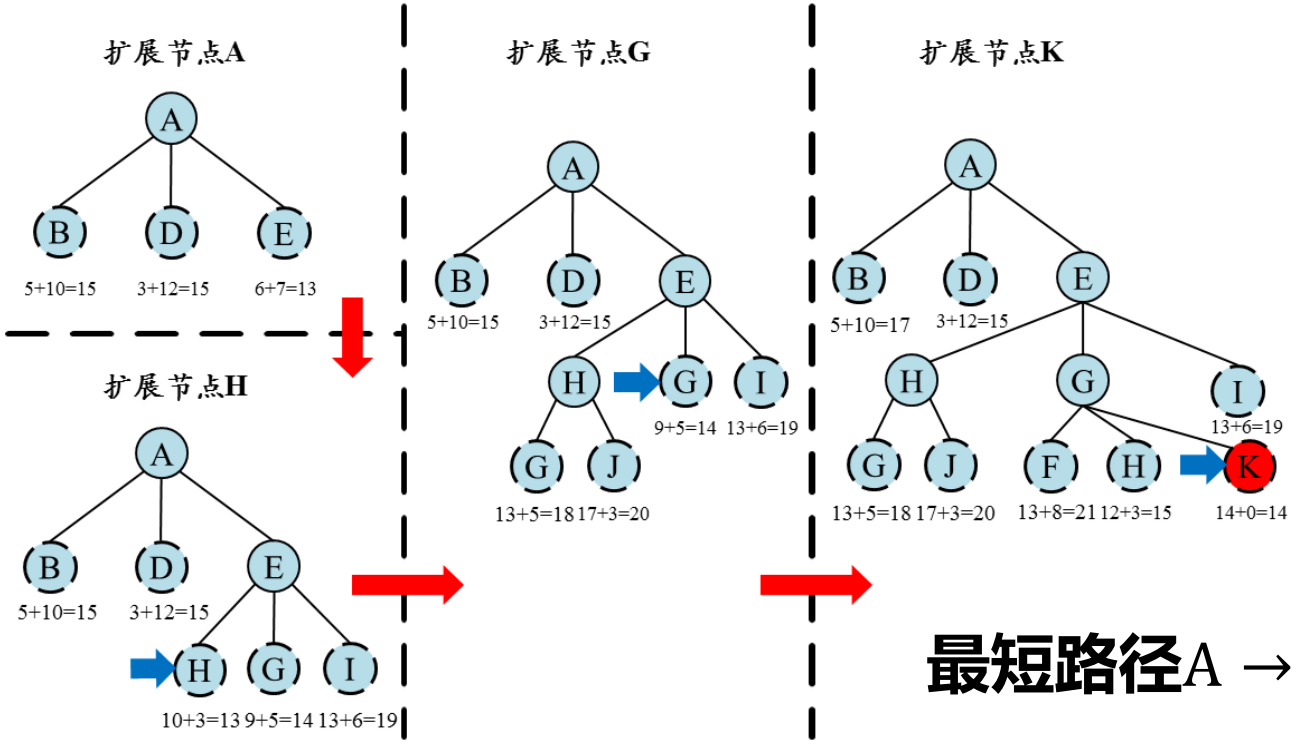
表3.2 每个状态（城市）所对应启发函数取值

状态	A	B	C	D	E	F	G	H	I	J	K	L
h(n)	13	10	6	12	7	8	5	3	6	3	0	6

辅助信息：任意一个城市与终点城市K之间的直线距离

问题：寻找从城市A到城市K之间行驶时间最短路线？

A*算法



最短路径$A \rightarrow E \rightarrow G \rightarrow K$

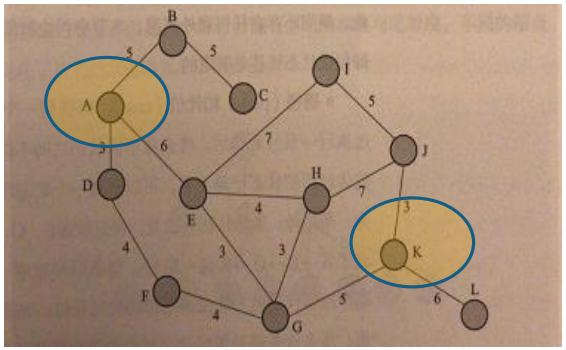


表3.2 每个状态（城市）所对应启发函数取值

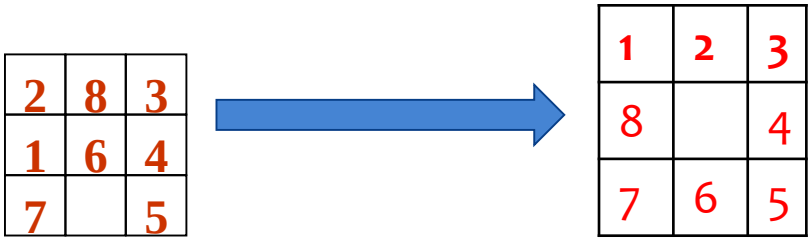
状态	A	B	C	D	E	F	G	H	I	J	K	L
h(n)	13	10	6	12	7	8	5	3	6	3	0	6

问题：寻找从城市A到城市K之间行驶时间最短路线？

辅助信息：任意一个城市与终点城市K之间的直线距离

采用A*算法求解八数码难题

其中：g(n)是搜索树中节点n的深度；h(n)用来计算对应于节点n的数据库中**错放的棋子个数**。因此，
起始节点棋局：

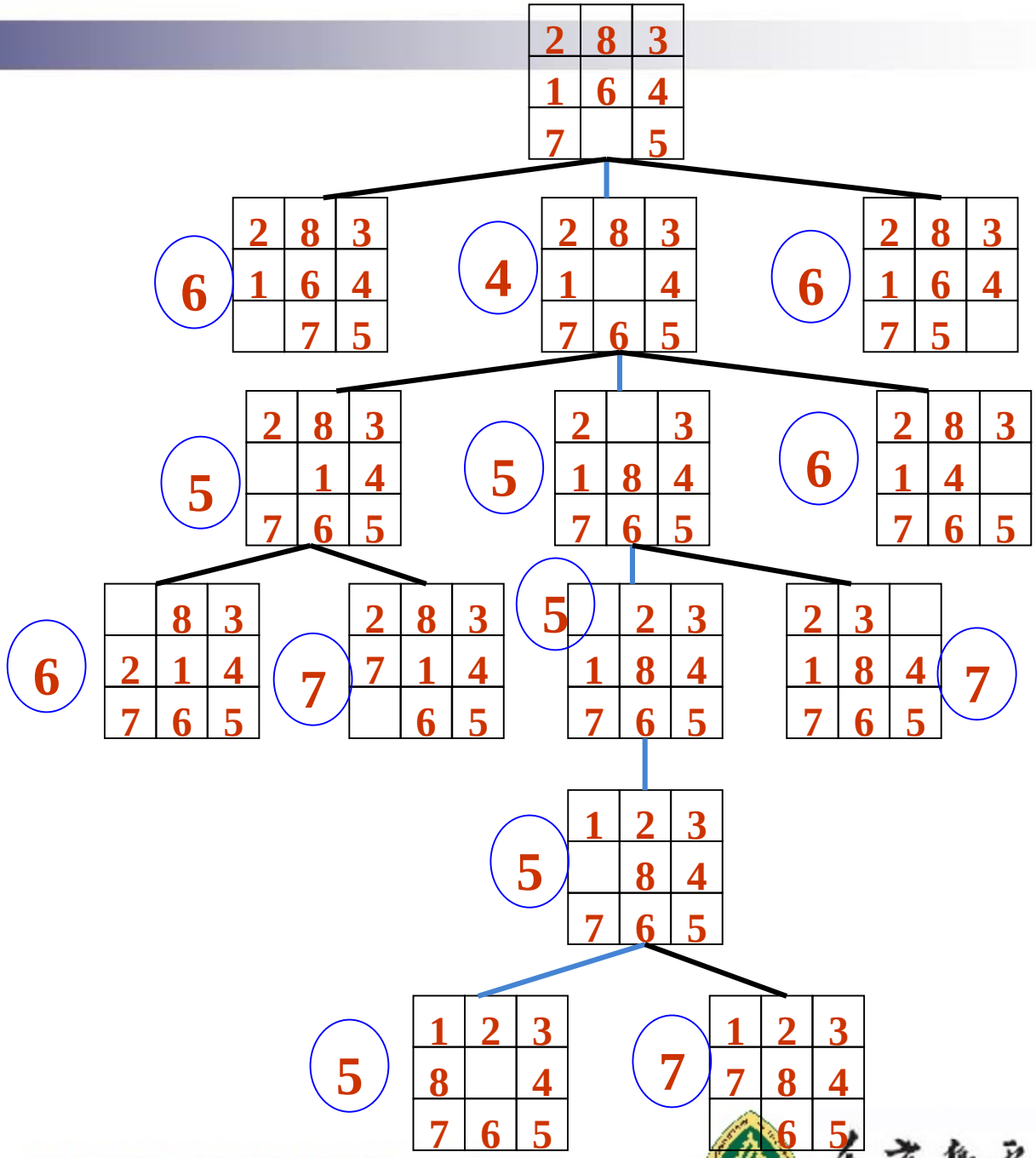


起始节点的f值等于0+4=4。

评价函数

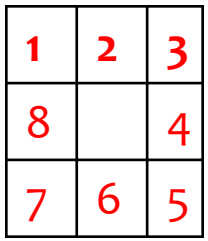
$f(n)=g(n)+h(n)$

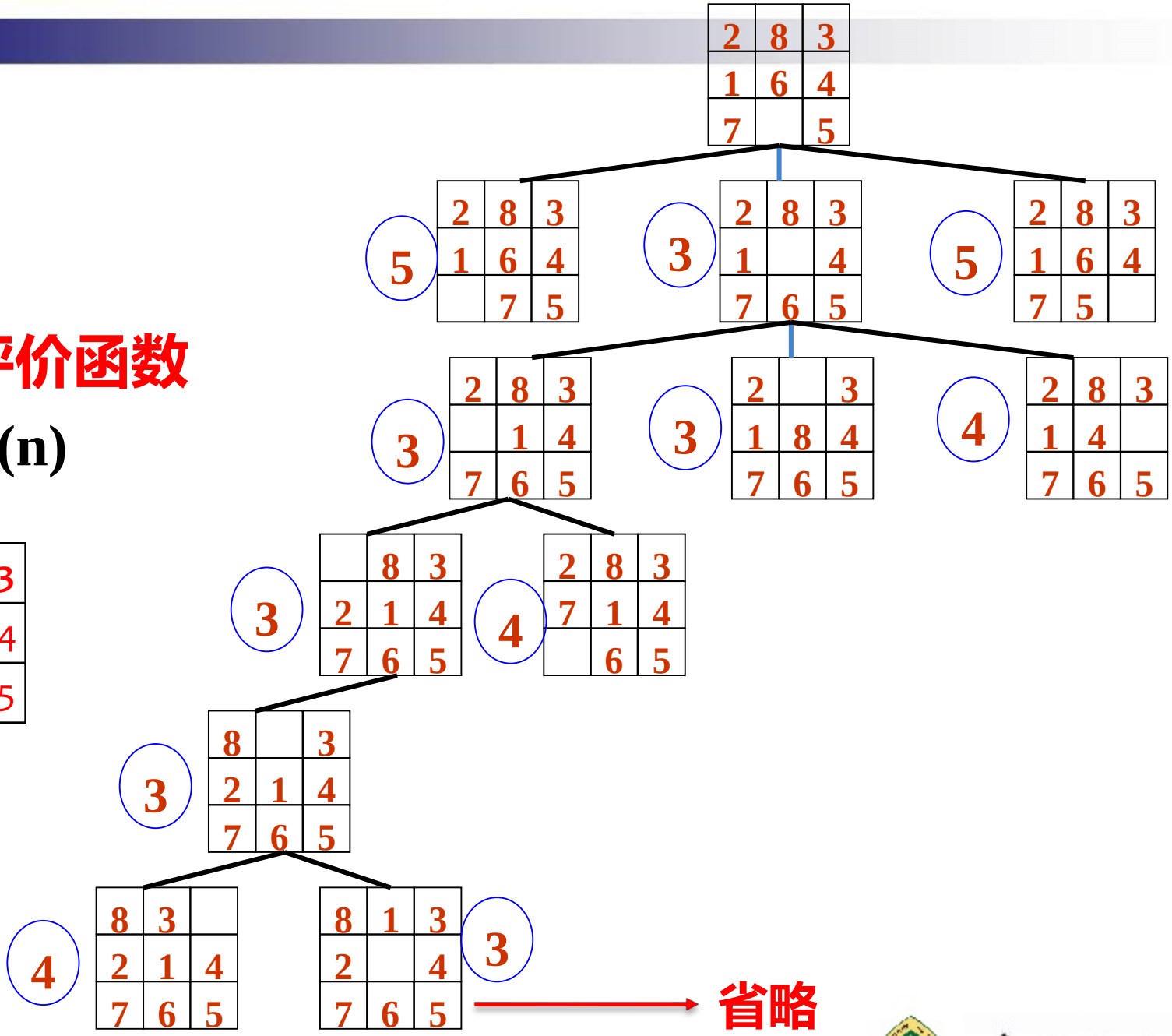
1	2	3
8		4
7	6	5



如果评价函数

f(n)=h(n)





省略

A算法和A*算法的定义

定义 在搜索过程中，如果重排OPEN表是依据f(x)=g(x)+h(x)进行的，则称该过程为A算法。

定义 在A算法中，如果对所有的x存在h(x)≤h*(x)（到目标节点最佳路径代价），则称h(x)为h*(x)的下界，它表示某种偏于保守的估计。

定义 采用h*(x)的下界h(x)为启发函数的A算法，称为A*算法。

A算法和A*搜索算法的目标有所不同： A搜索算法虽然希望能找到问题的最优解，但主要追求的是求解效率；而A*搜索算法直接目标就在于要找到问题的最优解及其解的路径，即便搜索效率有所降低也在所不惜。

例 4：八数码难题, 采用了简单的评价函数

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

其中：g(n)是搜索树中节点n的深度；h(n)用来计算对应于节点n的数据库中错放的棋子个数。因此，

起始节点棋局：

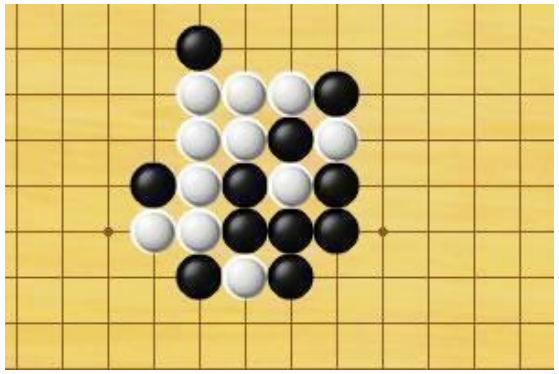
起始节点的f值等于0+4=4。

这是一个A*算法

对抗搜索(博弈树搜索)

1 博弈

何谓博弈？博弈就是下棋、打牌、竞技、战争等一类竞争性智能活动，从而产生另一类搜索问题，即博弈树搜索。

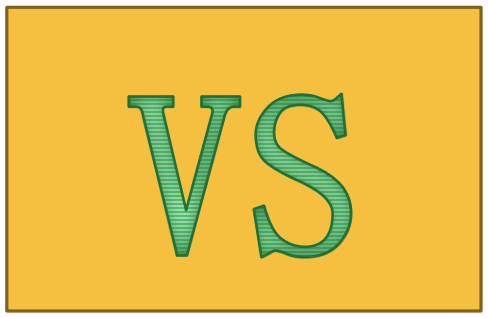


对抗搜索(博弈树搜索)

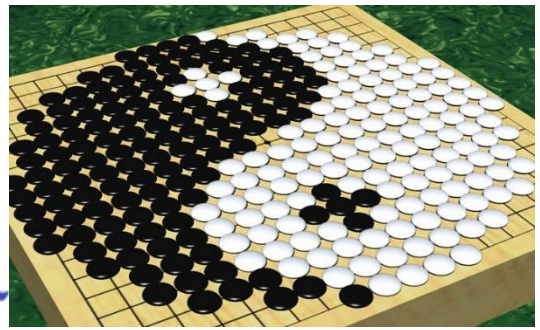
博弈过程

博弈的各方总是要挑选对自己最为有利而对对方最不利的那个行动方案。

- 游戏的两个玩家：MAX和MIN
- 站在玩家MAX的角度搜索赢的策略
- MAX方的目标：尽可能使自己方达到最大分数的行动方案，用“或”描述，称为MAX方节点
- MIN方的目标：尽可能使对方获得最小分数的行动方案，用“与”描述，是由MIN方自主进行控制的，称为MIN节点



计算机



对抗搜索(博弈树搜索)

3 博弈树

根据某一方（如MAX方取胜）把上述双方逐层交替的博弈过程用与/或树（图）描述表达出来，即一棵具有“与/或”节点交替出现的博弈树。

博弈树有如下特点：

（1）博弈的初始格局是初始节点。

（2）在博弈树中，由于双方轮流地扩展节点，“或”节点和“与”节点逐层交替出现。

（3）把本方获胜的终局定义为本原问题，相应最优搜索路径上的节点是可解节点，而所有使对方获胜的终局和属于对方最优搜索路径上的节点则是不可解节点。

对抗搜索(博弈树搜索)

极小极大（MiniMax）分析法

在二人博弈的常用分析方法就是极小极大化搜索法。

主要描述思想：

（1）设博弈的一方为MAX方，其目标是尽可能使自己得到最高分；另一方为MIN方，其目标是尽可能给MAX方送出最低分。

（2）考虑每一方案实施后对方可能采取的所有行动，并为其计算可能的得分；

（3）计算节点得分，需要根据问题的特性定义一个估价函数，用来估算当前博弈树所有端节点的得分。此时估算出来的得分称为的静态估值。

对抗搜索(博弈树搜索)

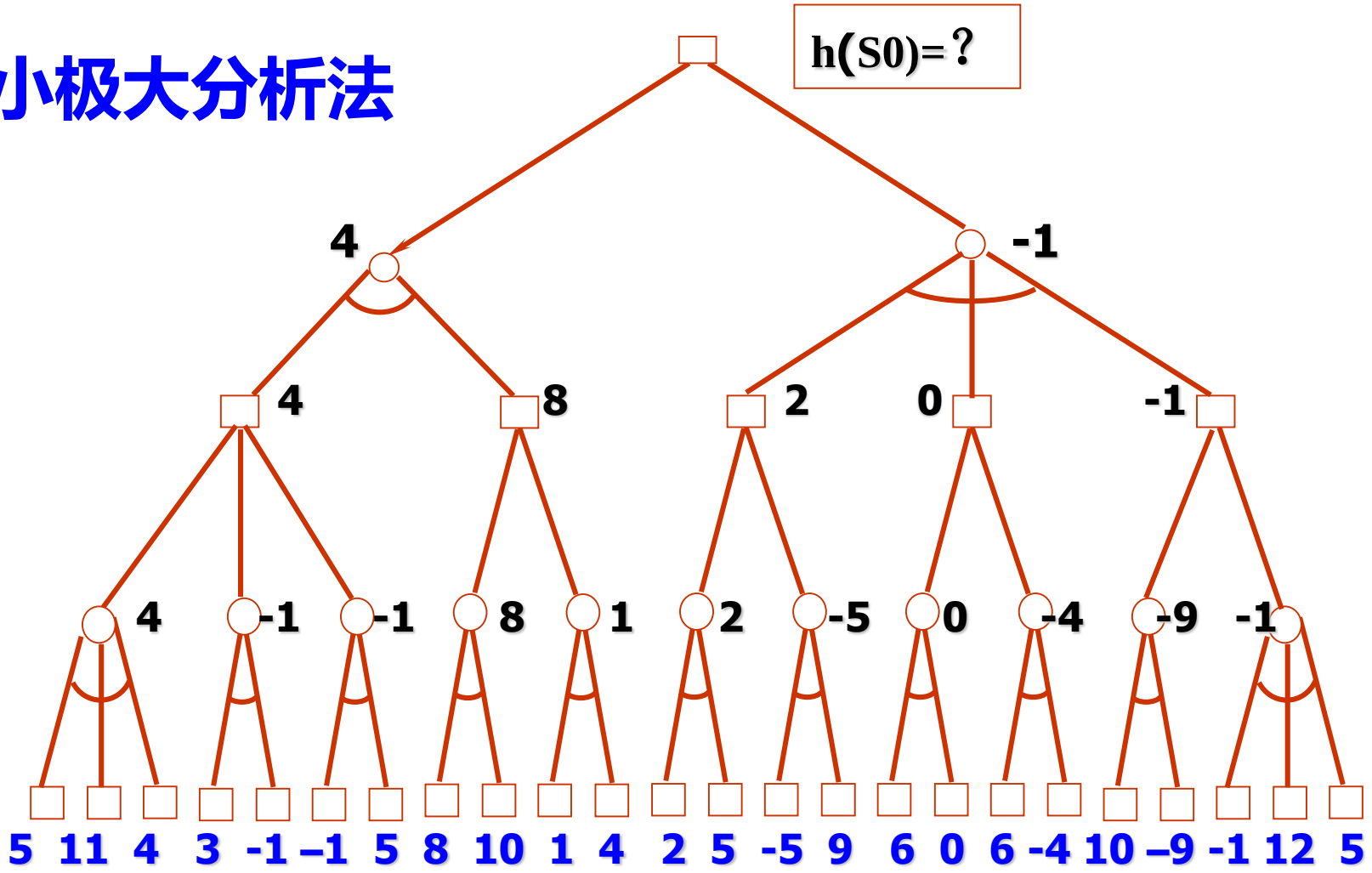
极小极大分析法

（4）父辈节点的得分（倒推值）：当端节点的估值计算出来后，再推算父辈节点的得分，推算方法是：对“或”节点，选择其子节点中最大的得分作为父辈节点的得分；对“与”节点，选其子节点中一个最小的得分作为作为父辈节点的得分（立足于最坏的情况）。

（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。

对抗搜索(博弈树搜索)

极小极大分析法

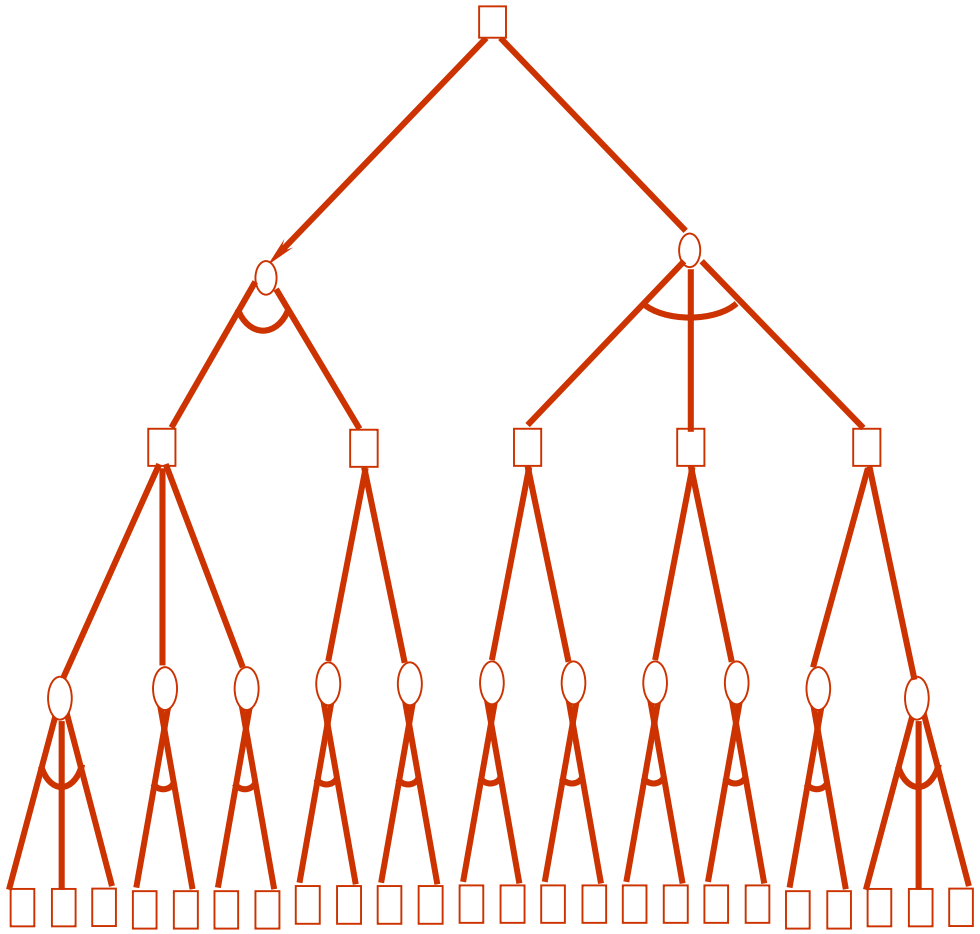


MAX-MIN博弈树的倒推值计算

博弈树的极大极小分析的局限

博弈问题中，博弈树很庞大，计算倒推值效率低，例如西洋跳棋的完整博弈树约10^{40}

- 一种解决方案是生成一定深度的博弈树，找出当前最好方案。
- 一种解决方案是减枝技术



对抗搜索(博弈树搜索)

α-β剪枝技术

α-β剪枝算法基本思想：**边生成博弈树边估算各节点的倒推值**，并且根据评估出的倒推值范围，及时停止扩展那些已无必要再扩展的子节点。通常都要使用某种**深度优先的搜索**方法

具体剪枝方法：

（1） 对于一个“与”节点MIN，**若能估计出其倒推值上界β，并且这个β值不大于MIN的父辈节点（一定是“或”节点）的估计倒推值的下界α**，即α≥ β，则就不必要再扩展该MIN节点的其余子节点了。这一过程称为α剪枝。

（2）对于一个“或”节点MAX，**若能估计出其倒推值下界α，并且这个α 值不小于MAX的父辈节点（一定是“与”节点）的估计倒推值的上界β**，即α≥ β，则就不必要再扩展该MAX节点的其余子节点了。这一过程称为β 剪枝。

对抗搜索(博弈树搜索)

α-β剪枝技术

从算法中看到:

(1) MAX节点(包括起始节点)的α值永不减少。

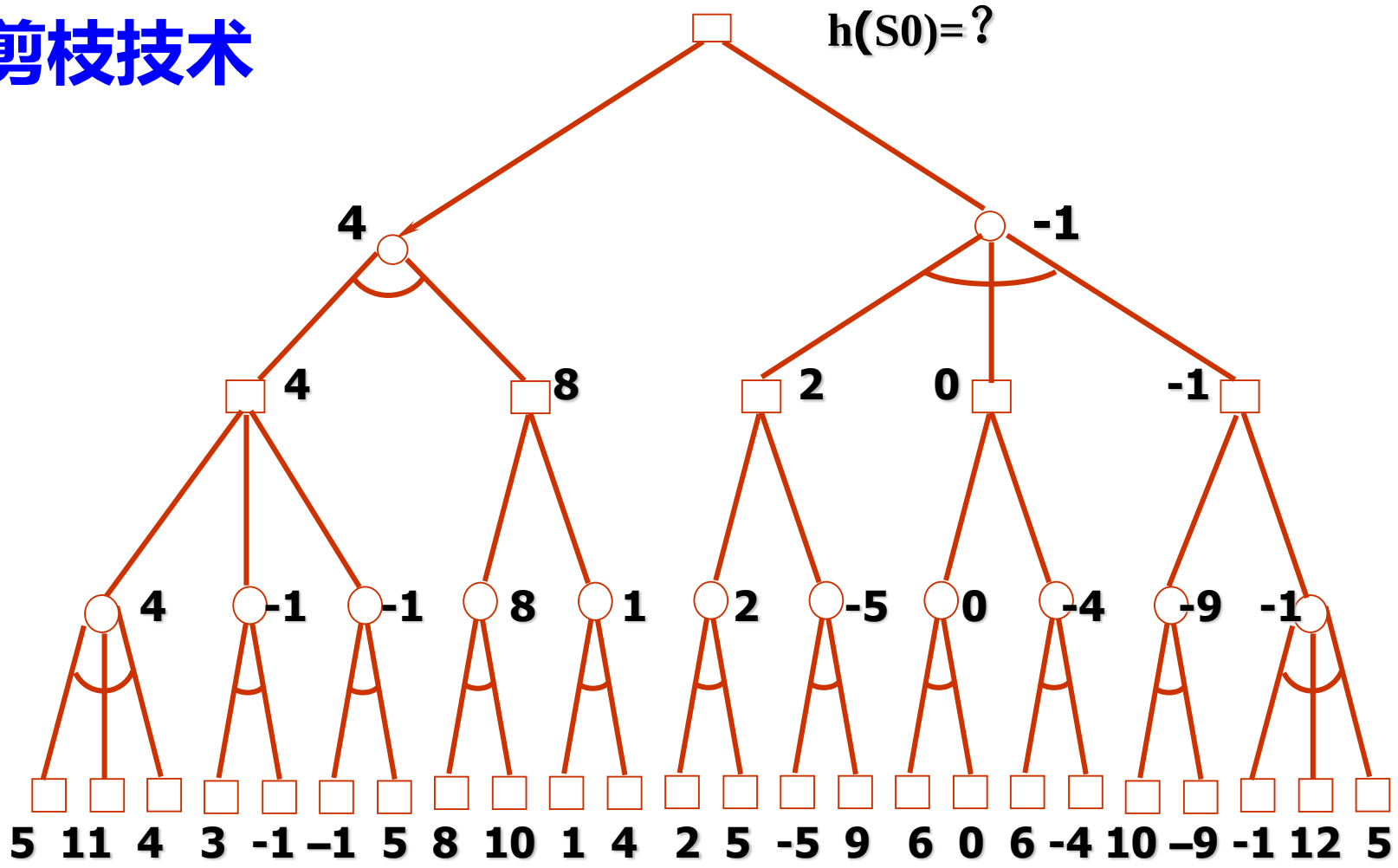
(2) MIN节点(包括起始节点)的β值永不增加。

在搜索期间, α和 β值的计算如下:

- 一个MAX节点的α值等于其后继节点当前最大的最终倒推值。
- 一个MIN节点的β 值等于其后继节点当前最小的最终倒推值。

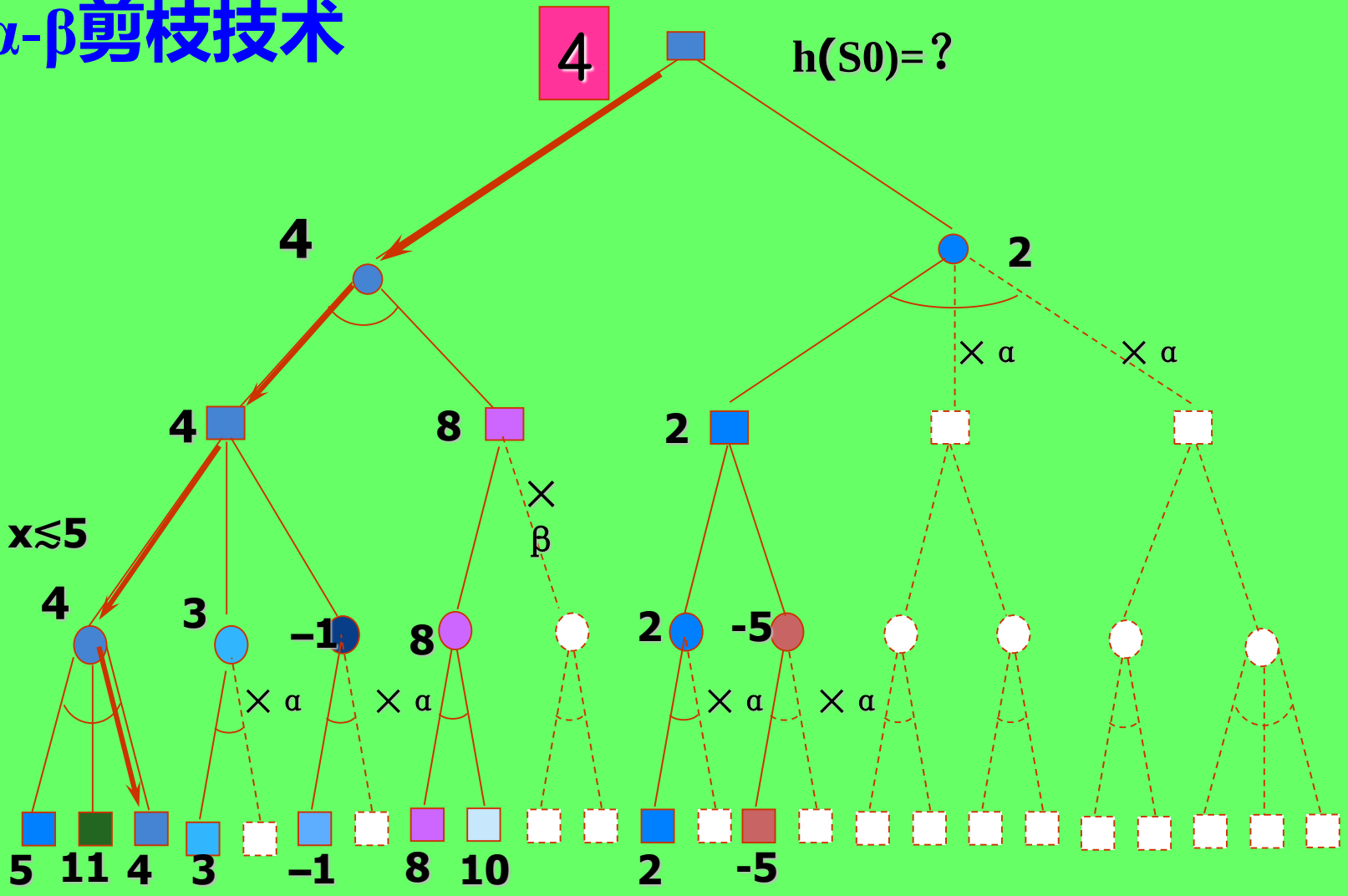
对抗搜索(博弈树搜索)

α-β剪枝技术



MAX-MIN博弈树的倒推值计算

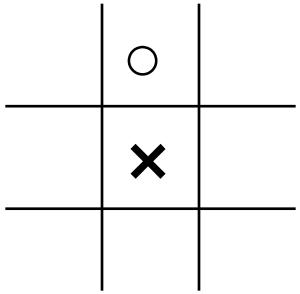
α-β剪枝技术



博弈树的 α－β 剪枝实现过程

对抗搜索（博弈树搜索）

α-β剪枝技术



例 5 一字棋搜索树α和β值计算

估价函数g(p)定义如下：

（1）若当前棋局对任何一方都不是获胜的，则

g(p)=(所有空格都放上MAX的棋子之后3个棋子所组成的行列及对角线的总数）—（所有空格都放上MIN的棋子之后3个棋子所组成的行列及对角线的总数)

（2）若p是MAX获胜，则

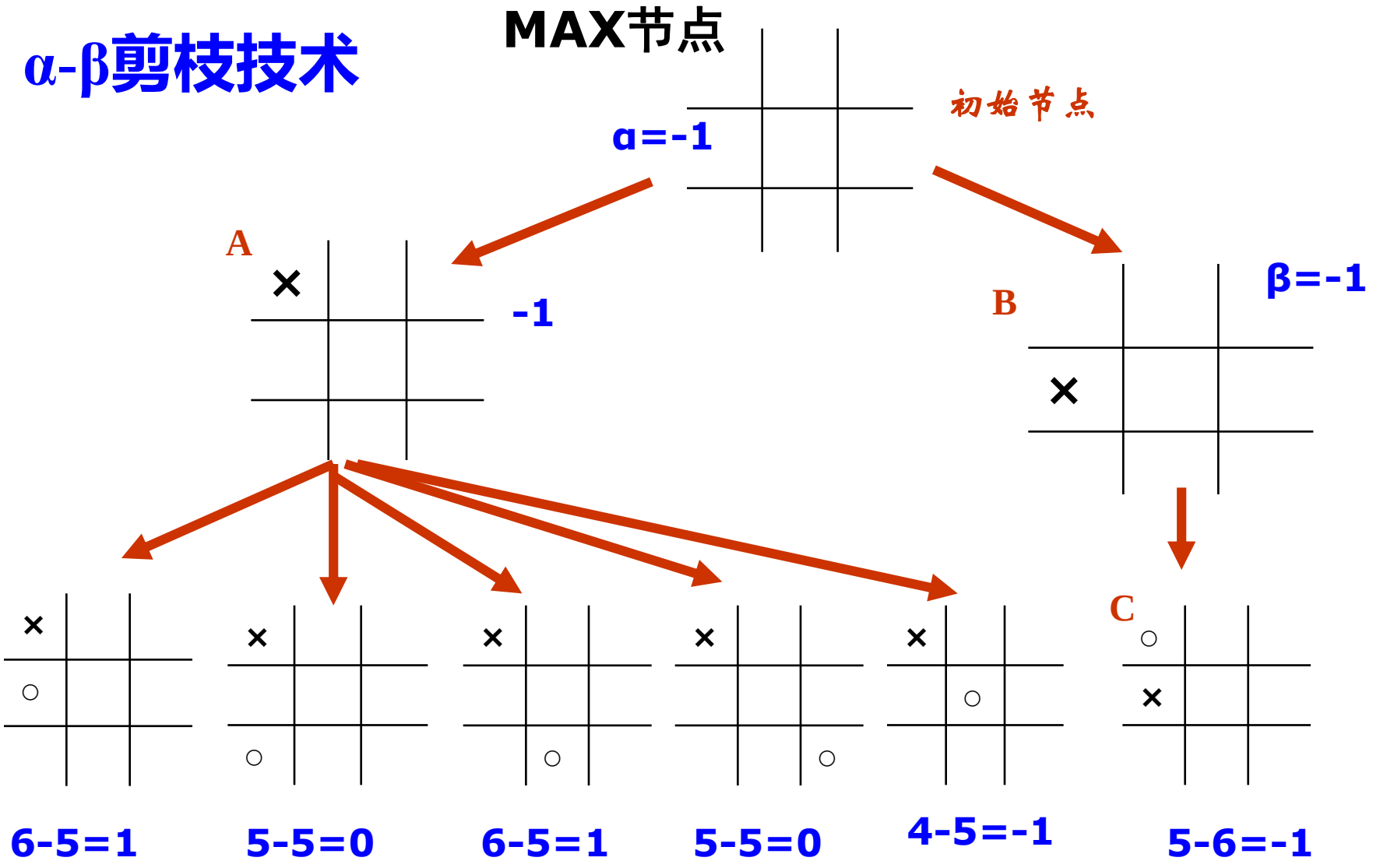
$g(p)=+\infty$

（3）若p是MIN获胜，则

$g(p)=-\infty$

上图中，g(p)=6-4=2，其中×表示MAX方，○表示MIN方

α-β剪枝技术



对抗搜索(博弈树搜索)

α-β剪枝技术

- 要进行α-β剪枝，必须至少使某一部分的搜索树生长到最大深度，因为α和β值必须以端节点的静态估值为依据。

- 采用α-β剪枝技术通常都要使用某种深度优先的搜索方法。

对抗搜索之蒙特卡洛树搜索

蒙特卡洛（模拟）方法

蒙特卡洛方法主要是用在评价（估值）函数上，双方在某个局面下「随机」走子，注意是「随机」走，走到终局或者残局为止。随机很多次（比如一万盘）之后，计算平均得分，平均得分越高的局面就越好。

对抗搜索之蒙特卡洛树搜索

先考虑一个简化问题：假设智能体面前有K个赌博机，每个赌博机有一个臂膀。每次转动一个赌博机臂膀，赌博机则会随机吐出一些硬币或不吐出硬币，将所吐出的硬币的币值用收益分数来表示。现在假设给智能体$\tau(\tau > \mathrm{K})$次转动臂膀的机会，那么智能体如何选择赌博机、转动τ次赌博机臂膀，能够获得更多的收益分数呢?

方法：让智能体先把K个赌博机的臂膀依次转动一遍，观察在摇动每个赌博机臂膀时的收益分数，然后去转动那些收益分数高的赌博机臂膀。但是，由于从每个赌博机获得的收益分数是随机的，显然一个刚刚给用户带来可观收益分数的赌博机在下一次摇动其臂膀时就可能不会获得可观收益分数了，因此这一方法不可取。

对抗搜索之蒙特卡洛树搜索

- **状态。每个被摇动的臂膀即为一个状态，记K个状态分别为$\{s_1, s_2, \cdots, s_K\}$，没有摇动任何臂膀的初始状态记为$s_0$。**
- **动作。动作对应着摇动一个赌博机的臂膀，在多臂赌博机问题中，任意状态下的动作集合都为$\{a_1, a_2 \cdots, a_K\}$，分别对应摇动某个赌博机的臂膀。**
- **状态转移。选择动作$a_i(1 \leq i \leq K)$后，将相应的改变为s_i。**

对抗搜索之蒙特卡洛树搜索

- 奖励（reward）。假设从第i个赌博机获得收益分数的分布为D_i，其均值为μ_i。如果智能体在第t次行动中选择转动了第l_t个赌博机臂膀，那么智能体在第t次行动中所得收益分数$\hat{r}_t$服从分布D_{l_t}，$\hat{r}_t$被称为第t次行动的奖励。为了方便对多臂赌博机问题的理论研究，一般假定奖励是有界的，进一步可假设奖励的取值范围为$[0,1]$。

- 悔值（regret）函数。根据智能体前T次动作，可以如下定义悔值函数：

$$\rho_T = T\mu^* - \sum_{t=1}^{T} \hat{r}_t \quad (3.4.1)$$

其中$\mu^* = \max_{i=1,\cdots,K} \mu_i$。显然为了尽量减少悔恨，在每次操作时，智能体应该总是转动能够给出最大期望奖励的赌博机臂膀，但是这是不现实的，因为智能体并不知道哪个臂膀的奖励期望最大。公式（3.4.1）告诉我们，将T次操作中最优策略的期望得分减去智能体的实际得分，就是悔值函数的结果。显然，问题求解的目标为最小化悔值函数的期望，该悔值函数的取值取决于智能体所采取的策略。

对抗搜索之蒙特卡洛树搜索

- 贪心算法策略：智能体记录下每次摇动的赌博机臂膀和获得的相应收益分数。给定第$i(1 \leq i \leq K)$个赌博机，记在过去$t-1$次摇动赌博机臂膀的行动中，一共摇动第i个赌博机臂膀的次数为第$T_{(i,t-1)}$。于是，可以计算得到第i个赌博机在过去$T_{(i,t-1)}$次被摇动过程中的收益分数平均值$\bar{x}_{i,T_{(i,t-1)}}$。这样，智能体在第t步，只要选择$\bar{x}_{i,T_{(i,t-1)}}$值最大的赌博机臂膀进行摇动，这是贪心算法的思路。

- 不足：忽略了其他从未摇动或很少摇动的赌博机，而失去了可能的机会

- 上述困境体现了探索（exploration）和利用（exploitation）之间存在对立关系。贪心算法基本上是利用从已有尝试结果中所得估计来指导后续动作，但问题是所得估计往往不能准确反映未被（大量）探索过的动作。因此，需要在贪心算法中增加一个能够改变其“惯性”的内在动力，以使得贪心算法能够访问那些尚未被（充分）访问过的空间。

对抗搜索之蒙特卡洛树搜索

ϵ-贪心算法：在探索与利用之间进行平衡的搜索算法。在第t步，ϵ-贪心算法按照如下机制来选择摇动赌博机：$l_t =$

$$\begin{cases} \text{argmax}_i \ \bar{x}_{i,T_{(i,t-1)}}, & \text{以} 1-\epsilon \text{的概率} \\ \text{随机的} i \in \{1,2,\cdots,K\}, & \text{以} \epsilon \text{的概率} \end{cases}$$

即以$1-\epsilon$的概率选择在过去$t-1$次摇动赌博机臂膀行动中所得平均收益分数最高的赌博机进行摇动；以ϵ的概率随机选择一个赌博机进行摇动。

不足：与被探索的次数无关。可能存在一个给出更好奖励期望的动作，但因为智能体对其探索次数少而认为其期望奖励小。因此，需要对那些探索次数少或几乎没有被探索过的动作赋予更高的优先级。

对抗搜索之蒙特卡洛树搜索

上限置信区间算法（Upper Confidence Bounds, UCB1）：

UCB1算法的策略可以描述为，在第t次时选择使得（3.4.3）式子取值最大的动作a_{l_t}，其中l_t由如下式子计算得到：

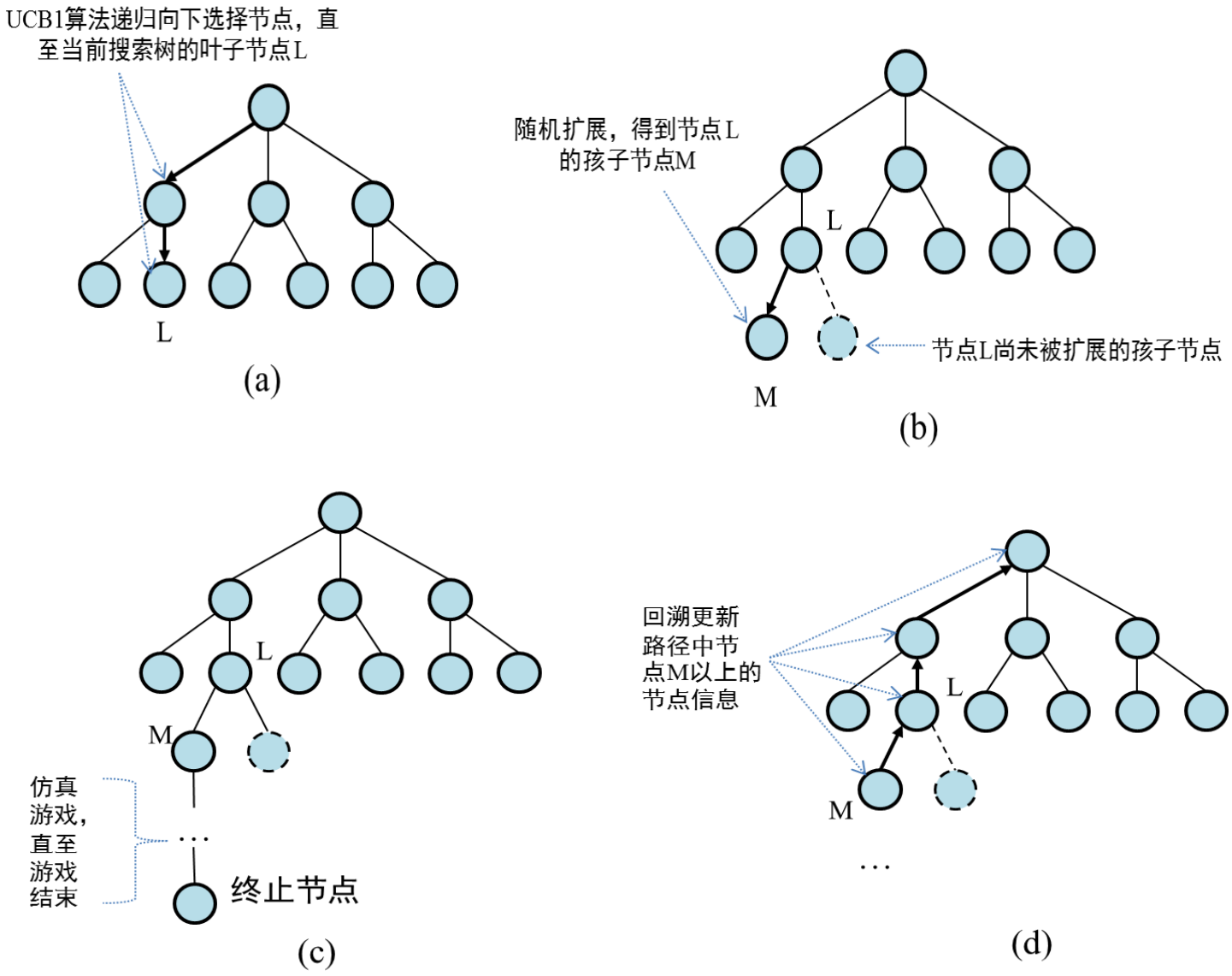
$$l_t = \text{argmax}_i \, \bar{x}_{i,T_{(i,t-1)}} + C\sqrt{\frac{2\ln t}{T_{(i,t-1)}}} \qquad (3.4.3)$$

在过去第t次已经对动作a_i探索了$T_{(i,t-1)}$次，在当前问题中对应摇动了第i个赌博机的臂膀$T_{(i,t-1)}$次，执行动作a_i所收到收益分数的均值为$\bar{x}_{i,T_{(i,t-1)}}$

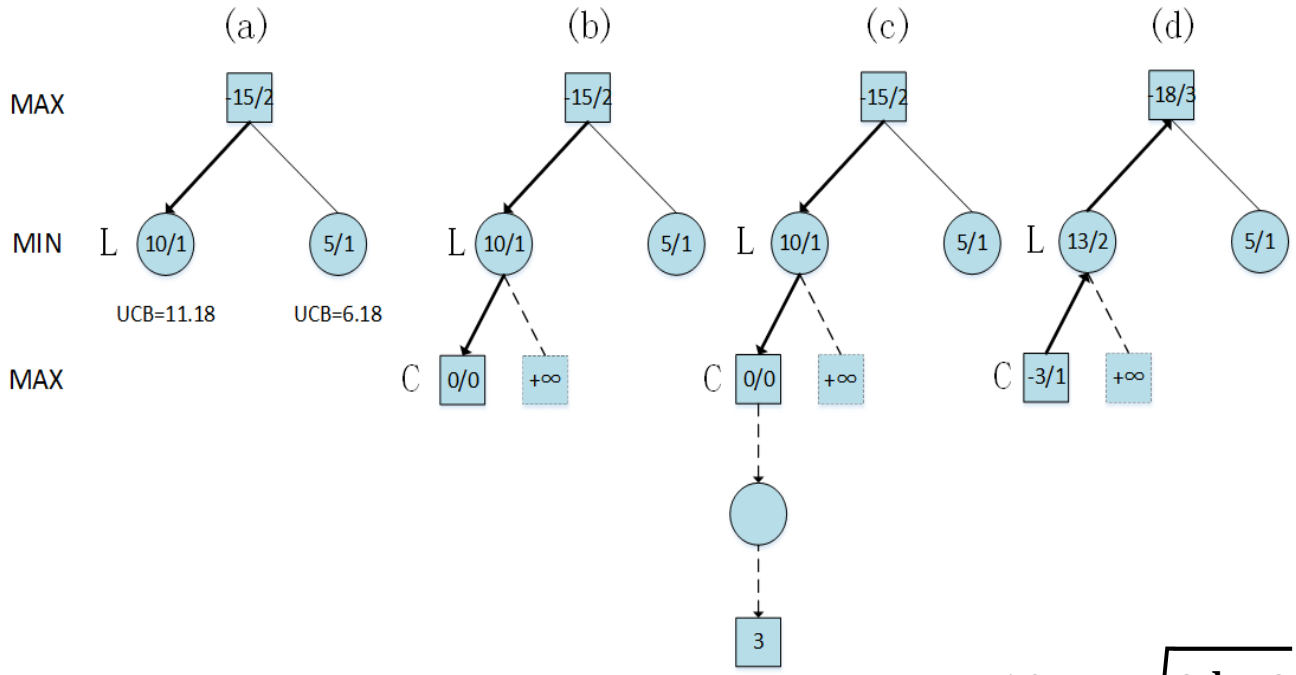
对抗搜索之蒙特卡洛树搜索

- 选择（selection）：选择指算法从搜索树的根节点开始，向下递归选择子节点，直至到达叶子节点或者到达具有还未被扩展过的子节点的节点L。这个向下递归选择过程可由UCB1算法来实现，在递归选择过程中记录下每个节点被选择次数和每个节点得到的奖励均值。
- 扩展（expansion）：如果节点 L 不是一个终止节点（或对抗搜索的终局节点），则随机扩展它的一个未被扩展过的后继边缘节点M。
- 模拟（simulation）：从节点M出发，模拟扩展搜索树，直到找到一个终止节点。模拟过程使用的策略和采用UCB1算法实现的选择过程并不相同，前者通常会使用比较简单的策略，例如使用随机策略。
- 反向传播（Back Propagation）：用模拟所得结果（终止节点的代价或游戏终局分数）回溯更新模拟路径中M以上（含M）节点的奖励均值和被访问次数。

对抗搜索之蒙特卡洛树搜索

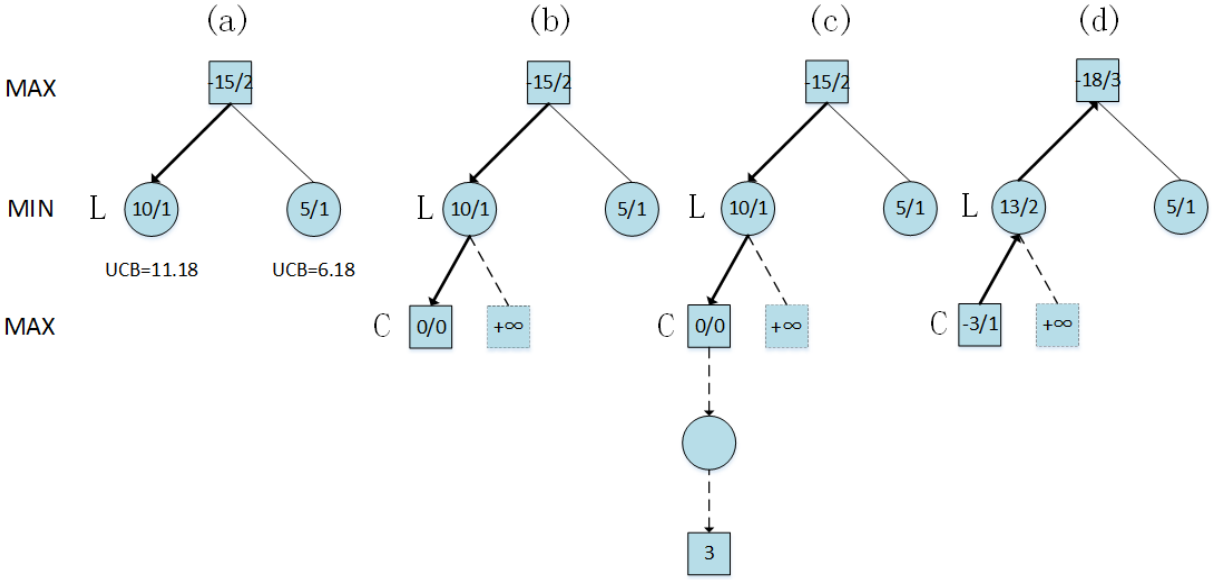


对抗搜索之蒙特卡洛树搜索



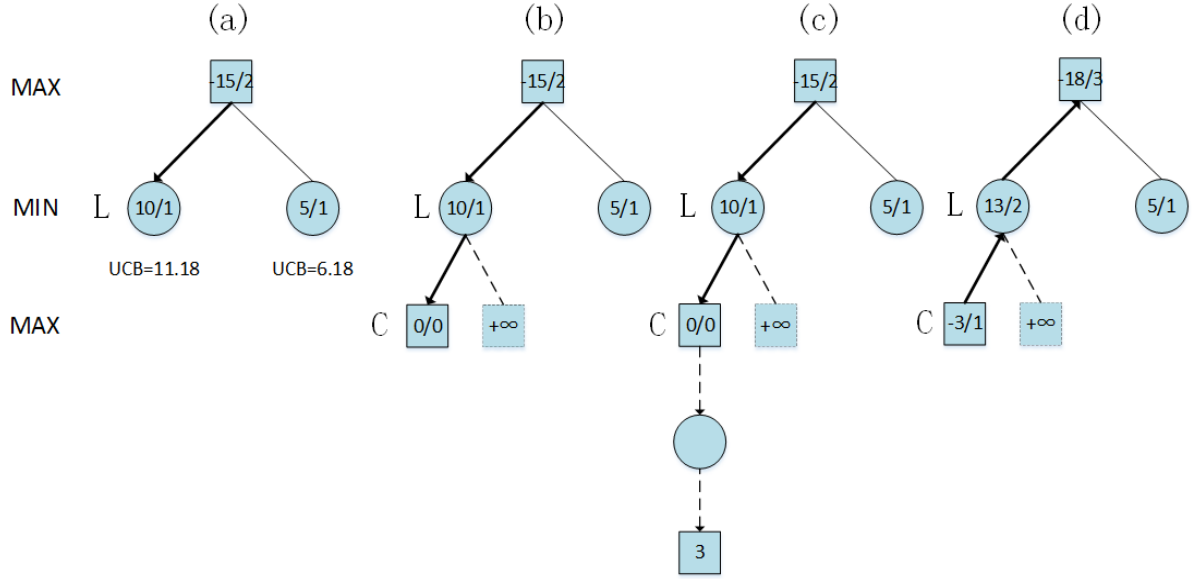
计算第二层节点的UCB值：左侧节点为 $\frac{10}{1}+\sqrt{\frac{2\ln 2}{1}}=11.18$，**右侧节点为** $\frac{5}{1}+\sqrt{\frac{2\ln 2}{1}}=6.18$，**因此算法选择第二层左侧的节点L，由于该节点有尚未扩展的子节点，因此选择阶段结束。**

对抗搜索之蒙特卡洛树搜索



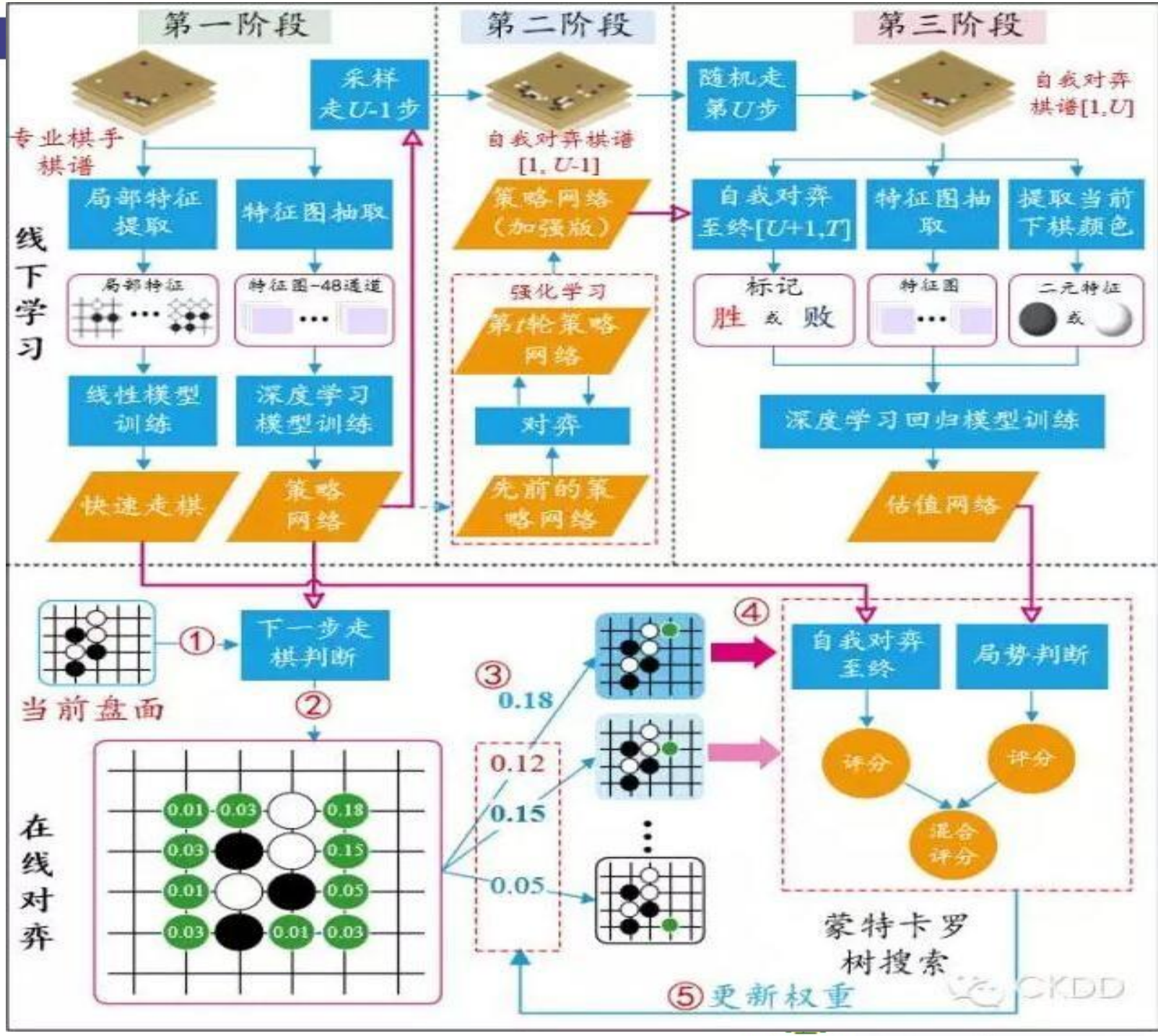
在图3.4.6(b)中，算法随机扩展了L的子节点C，将其总分数和被访问次数均初始化为0。注意，为了清晰地展示算法选择扩展的节点，图3.4.6(b)画出了L的其他未被扩展的子节点，并标记其UCB值为正无穷大，以表示算法下次访问到L时必然扩展这些未被扩展的节点。图3.4.6(c)中采用随机策略模拟游戏直至完成游戏。当游戏完成时，终局得分为3。

对抗搜索之蒙特卡洛树搜索



在图3.4.6(d)中C节点的总分被更新为-3，被访问次数被更新为1；L节点的总分被更新为13，被访问次数被更新为2；根节点的总分被更新为-18，被访问次数被更新为3。在更新时，会将MIN层节点现有总分加上终局得分分数， MAX层节点现有总分减去终局得分分数。这是因为在对抗搜索中，玩家MIN总是期望最小化终局得分，因此在MIN层选择其子节点时，其目标并非选取奖励最大化的子节点，而是选择奖励最小化的节点，为了统一使用UCB1算法求解，算法将MIN层的子节点（即MAX层节点）的总分记为其相反数。

AlphaGo


第一阶段
第二阶段
第三阶段
专业棋手棋谱
采样走U-1步
自我对弈棋谱[1, U-1]
随机走第U步
自我对弈棋谱[1,U]
线下学习
局部特征提取
特征图抽取
局部特征
特征图-48通道
线性模型训练
深度学习模型训练
快速走棋
策略网络
策略网络（加强版）
强化学习
第t轮策略网络
对弈
先前的策略网络
自我对弈至终[U+1,T]
特征图抽取
提取当前下棋颜色
标记
胜 或 败
特征图
二元特征
或
深度学习回归模型训练
估值网络
当前盘面
①
下一步走棋判断
②
在线对弈
0.01 0.03 0.18
0.03 0.15
0.01 0.05
0.03 0.01 0.03
③
0.18
0.12
0.15
0.05
④
自我对弈至终
局势判断
评分
评分
混合评分
蒙特卡罗树搜索
⑤更新权重

Alpha Go的改进

Alpha Go

- 以一个估值网络取代“模拟”操作
- 更改MCTS的选点公式为

$$\mathbb{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + cP(v, v_i)\sqrt{\frac{N(v)}{1+N(v_i)}}$$

其中，$P(v, v_i)$由一个策略网络给出。

- 用强化学习进行自我对弈训练估值网络。