



# 第5章 搜索策略

---

- 问题规约、与或图的概念和一般搜索过程
- 与或图的盲目搜索
- 与或图的启发式搜索
- 博弈树的启发式搜索

# 问题规约和与 / 或图搜索

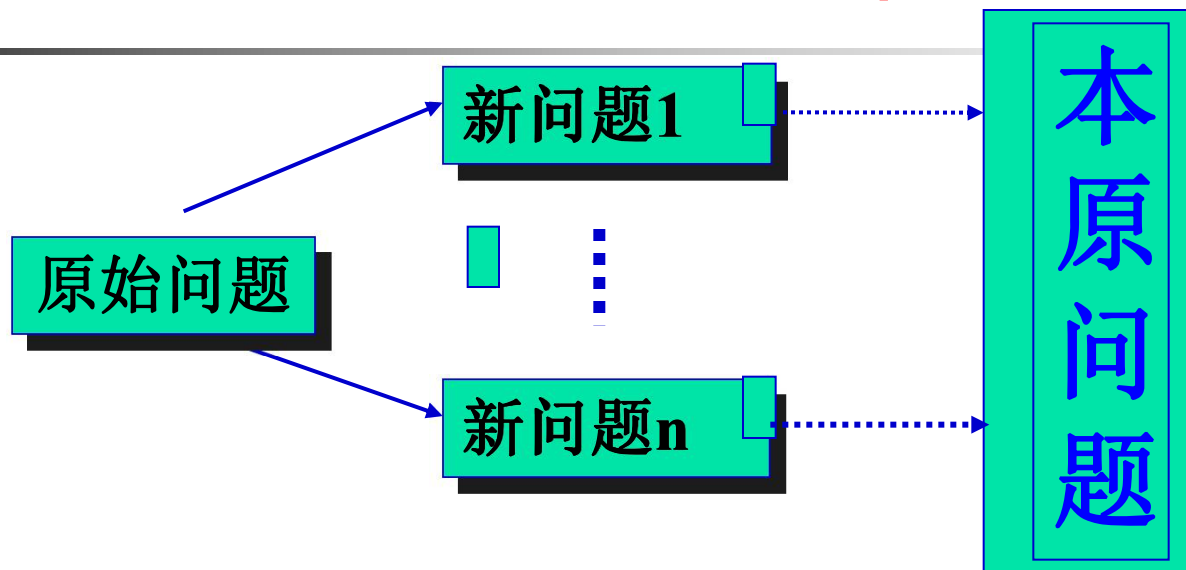
问题归约是人求解问题常用的策略：

把复杂的问题变换为较为简单的新问题后再求解

- 问题归约可以用三元组表示：  $(S_0, O, P)$  , 其中
  - $S_0$ 是初始问题，即要求解的问题；
  - $P$ 是本原问题集，其中的每一个问题是不用求解的或已求解过的问题；
  - $O$ 是操作算子集，它是一组变换规则把一个问题变换为若干个新问题。

# 问题归约法

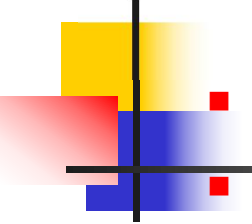
(Problem Reduction Representation)



## 问题归约过程：

由初始问题出发，运用操作算子产生一些关联的新问题，对新问题再运用操作算子产生与新问题关联的新问题，这样一直进行到产生的问题均为本原问题，则问题得解。

# 与/或图表示

- 
- 应用问题归约策略得到的问题空间图，也称为“与或图”
  - 逻辑“与”关系——用圆弧将几条节点间关联弧连接在一起
    - 将原问题P分解为新问题P1,P2,P3...,Pn;
    - 新问题全部解决才会导致原问题的解决;
    - 只要有一个新问题无解，则原问题就无解。
    - 新问题的解答联合起来构成原问题的解答。
  - 逻辑“或”关系：
    - 将原问题P变换为若干个较容易求解的新问题P1,P2,P3...,Pn;
    - 只要这些新问题中有一个有解，则原问题就有解;
    - 只有新问题都无解时，原问题才无解。
    - 其中一个新问题的解答构成原问题的解答

# 与或图的两种表示方式

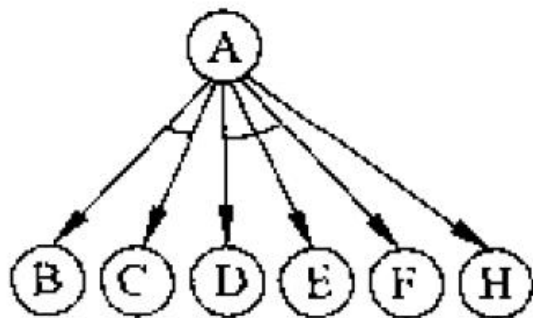


图 2.8 子问题替换集合结构图

与或图表示方式1

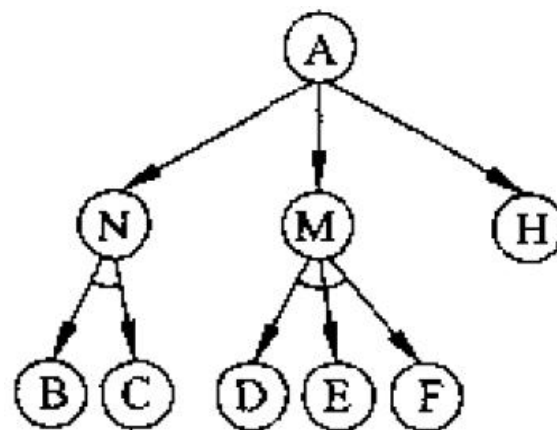


图 2.9 一个与或图

与或图表示方式2



# 与或图

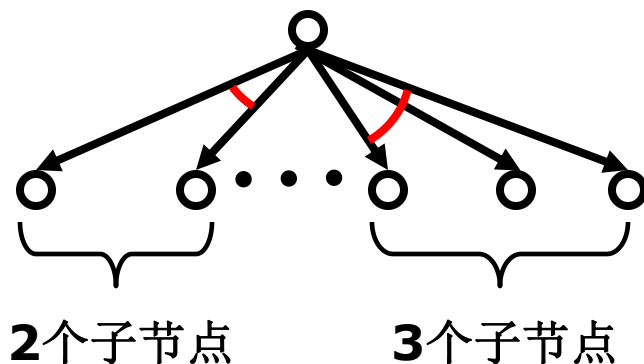
问题归约求解问题的过程  
表示为与或图搜索

- 与或图视为对一般图(或图)的扩展；
  - 引入K-连接
    - 子节点间可以存在“与”关系
  - 结果——解图。
    - 解答不再是路径形式，代之以广义的解路径——解图。

# 与或图

## ■ K-连接

- 从父节点到K个子节点的连接，子节点间有“与”关系；
- 以圆弧指示这些子节点间的“与”关系；



- 当所有的K都等于1时，与或图蜕化为一般图（或图）。



# 与或图

---

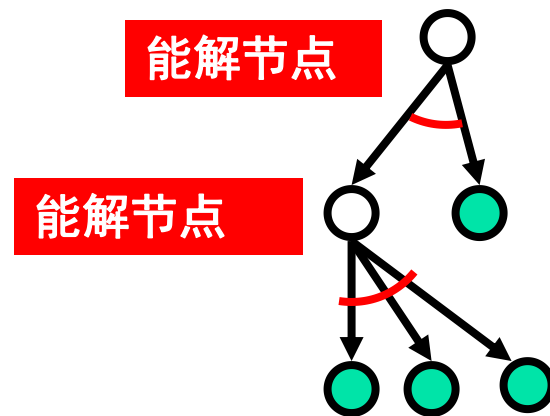
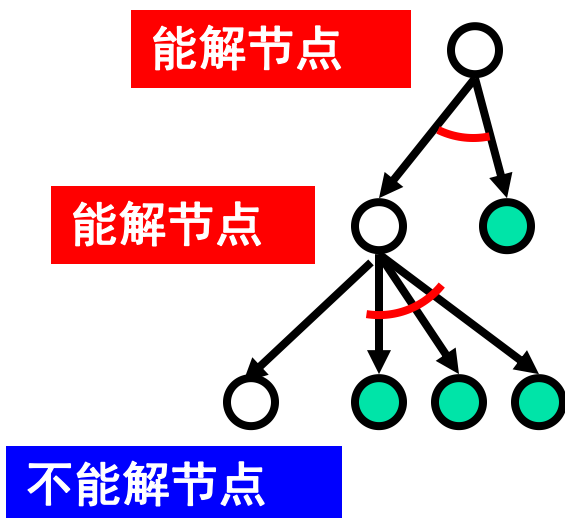
## ■ 根、叶节点

- 根节点——无父节点的节点 用于指示原问题
- 叶节点——无子节点的节点（不再继续分解或变换）
- 终叶节点——对应于本原问题的节点
- 不可解叶节点——除终叶节点以外的其它叶节点



- **能解节点**（定义1，对应与或图表示方式1）
  - (1) **终叶节点**是能解节点（因为代表本原问题）；
  - (2) 若节点**n**有一**K-连接**指向子节点**n1, n2, ...nk**,且这些子节点都是能解节点，则**n**是能解节点；

- **能解节点**（定义2，对应与或图表示方式2）
  - (1) **终叶节点**是能解节点（因为代表本原问题）；
  - (2) 如果某个节点是**或节点**，那么只有当其后继节点**至少有一个**是可解的时候，此节点才是可解的。
  - (3) 如果某个节点是**与节点**，那么只要当其后继节点**全部为**可解时，此节点才是可解的。

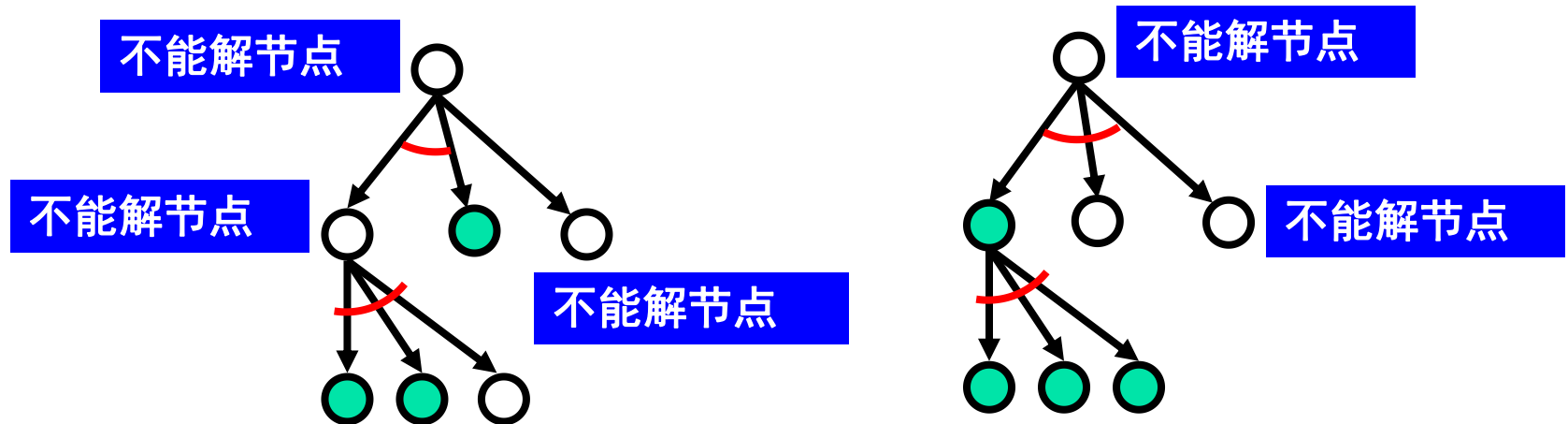


**不可解节点**（定义1，对应与或图表示方式1）

- **(1)不可解叶节点**是不能解节点；
- **(2)若节点n的每一个K-连接都至少指向一个不能解节点，则n是不能解节点。**

**不可解节点**（定义2，对应与或图表示方式2）

- **(1) 不可解叶节点**是不能解节点；
- **(2) 如果某个节点是或节点，那么只有当其后继节点全部不可解的时候，此节点才是不可解的。**
- **(3)如果某个节点是与节点，那么只要当其后继节点至少有一个不可解时，此节点才是不可解的。**





# 与或图

---

**解图：**能导致初始节点可解的那些可解节点及有关连线组成的子图称为该与或图的解图。

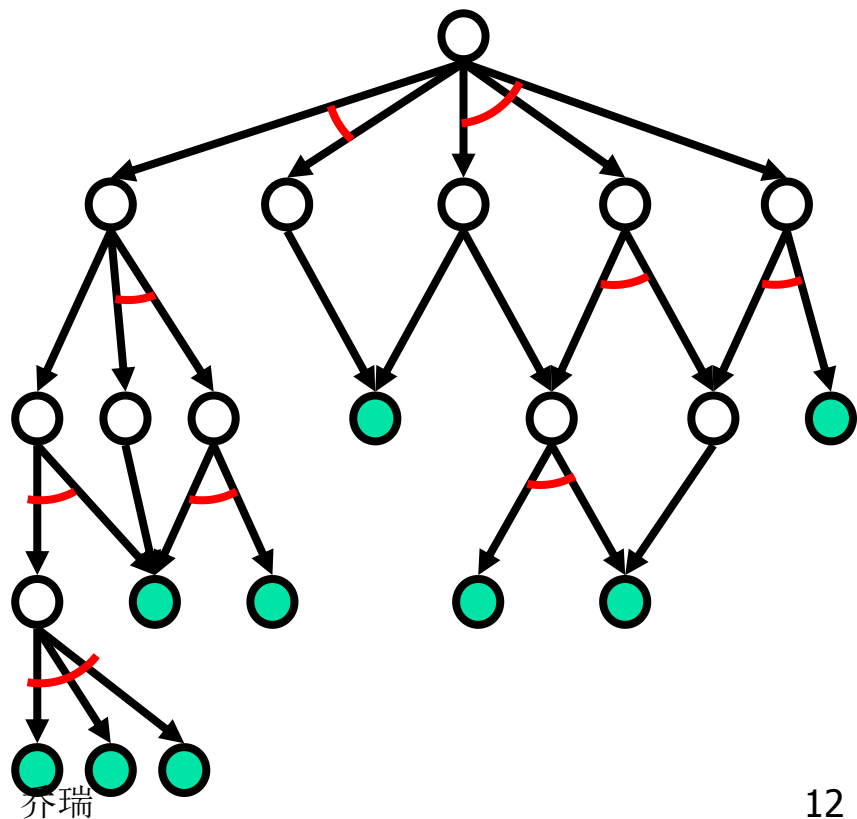
两种定义对应于与或图的两种不同的表示方法

- **定义1:** 一个与或图G中, 从节点n到终节点集合N的解图记为G', 是G的子图。

①若 $n$ 是 $N$ 的一个元素, 则 $G'$ 由单一节点组成;

②若 $n$ 有一个指向节点 $\{n_1, \dots, n_k\}$ 的外向连接符 $K$ , 使得从每一个 $n_i$ 到 $N$ 有一个解图 ( $i=1, \dots, k$ ), 则 $G'$ 由节点 $n$ , 连接符 $K$ , 及 $\{n_1, \dots, n_k\}$ 中的每一个节点到 $N$ 的解图所组成;

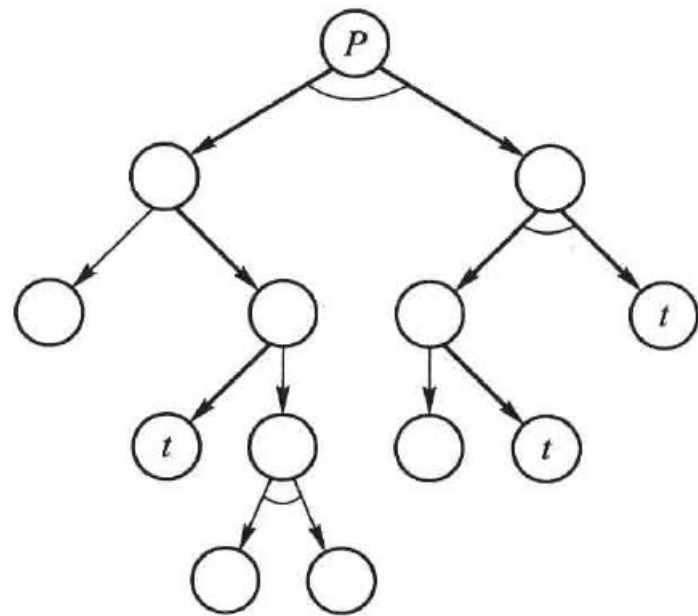
③ 否则  $n$  到  $N$  不存在解图。



## 对应于与或图表示方式2

定义2: 与或图 (记为 $G$ ) 任一节点 (记为 $n$ ) 到终节点集合的解图 (记为 $G'$ ) 是 $G$ 的子图。

- (1) 若 $n$ 是终节点, 则 $G'$ 就由单一节点 $n$ 构成;
- (2) 若 $n$ 为与节点,  $n$ 的每个子节点都有到终节点集合的解图, 则 $G'$ 由 $n$ 和所有这些相应于子节点的解图构成;
- (3) 若 $n$ 为或节点, 若存在一个子节点到终节点集合的解图, 则 $n$ 和该子节点的解图一起构成解图 $G'$ ;
- (4) 否则 $n$ 到 $N$ 不存在解图





# 与或图的一般搜索算法

- 1. 把待求解的原始问题作为初始节点**S0**，并把它作为当前节点
- 2. 应用变换算符对当前节点进行扩展
- 3. 为每个子节点设置指向父节点的指针
- 4. 选择合适的子节点作为当前节点，反复执行第2步和第3步，在此期间将多次调用**可解标示过程**或**不可解标示过程**，直到初始节点**S0**被标示为可解节点或不可解节点为止。



## 5.4 与或图的盲目搜索

---

- 与或图的广度优先搜索
- 与或图的深度优先搜索

# 与或图的广度优先搜索

## 阶段一：初始化

- 1. 把初始节点 $S_0$ 放入OPEN表中。

## 阶段二：搜索循环

- 2. 依次把OPEN表中第一个节点 $n$ 取出放入CLOSED表中。
- 3. 如果节点 $n$ 是终止节点，则作如下工作：
  - 3.1 标识 $n$ 为可解节点。
  - 3.2 把 $n$ 的先辈节点中的可解节点都标识为可解节点。
  - 3.3 若初始节点 $S_0$ 能被标识为可解节点，就得到了解树，成功退出。
  - 3.4 否则，从OPEN表中删去具有可解先辈的所有节点。
- 4. 如果节点 $n$ 可扩展，则作下列工作：
  - 4.1 扩展节点 $n$ ，产生 $n$ 的所有子节点。
  - 4.2 把这些子节点放入OPEN表尾部，并为每个子节点配置指向父节点（节点 $n$ ）的指针。
- 5. 如果节点 $n$ 不是终止节点，且它不可扩展，则作下列工作：
  - 5.1 标识 $n$ 为不可解节点。
  - 5.2 把 $n$ 的先辈节点中的不可解节点都标识为不可解节点。
  - 5.3 若初始节点 $S_0$ 也被标识为不可解节点，则失败退出。
  - 5.4 否则，从OPEN表中删去具有不可解先辈的所有节点。
- 6. 转第2步。



# 与或图的深度优先搜索

## 阶段一：初始化

- 1. 把初始节点 $S_0$ 放入**OPEN**表中。

## 阶段二：搜索循环

- 2. 依次把**OPEN**表中第一个节点 $n$ 取出放入**CLOSED**表中。
- 3. 如果节点 $n$ 是终止节点，则作如下工作：
  - 3.1 标识 $n$ 为可解节点。
  - 3.2 把 $n$ 的先辈节点中的可解节点都标识为可解节点。
  - 3.3 若初始节点 $S_0$ 能被标识为可解节点，就得到了解树，成功退出。
  - 3.4 否则，从**OPEN**表中删去具有可解先辈的所有节点。
- 4. 如果节点 $n$ 可扩展，则作下列工作：
  - 4.1 扩展节点 $n$ ，产生 $n$ 的所有子节点。
  - 4.2 把这些子节点放入**OPEN**表前端，并为每个子节点配置指向父节点（节点 $n$ ）的指针。
- 5. 如果节点 $n$ 不是终止节点，且它不可扩展，则作下列工作：
  - 5.1 标识 $n$ 为不可解节点。
  - 5.2 把 $n$ 的先辈节点中的不可解节点都标识为不可解节点。
  - 5.3 若初始节点 $S_0$ 也被标识为不可解节点，则失败退出。
  - 5.4 否则，从**OPEN**表中删去具有不可解先辈的所有节点。
- 6. 转第2步。

# 与或树有界深度优先搜索算法

## 阶段一：初始化

- 1. 把初始节点 $S_0$ 放入**OPEN**表中。

## 阶段二：搜索循环

- 2. 依次把**OPEN**表中第一个节点 $N$ 取出放入**CLOSED**表中。
- 3. 如果节点 $n$ 的深度 $\geq dm$ ，则转第6.1点。
- 4. 如果节点 $n$ 是终止节点，则作如下工作：
  - 4.1 标识 $n$ 为可解节点。
  - 4.2 把 $n$ 的先辈节点中的可解节点都标识为可解节点。
  - 4.3 若初始节点 $S_0$ 能被标识为可解节点，就得到了解树，成功退出。
  - 4.4 否则，从**OPEN**表中删去具有可解先辈的所有节点。
- 5. 如果节点 $n$ 可扩展，则作下列工作：
  - 5.1 扩展节点 $n$ ，产生 $n$ 的所有子节点。
  - 5.2 把这些子节点放入**OPEN**表前端，并为每个子节点配置指向父节点（节点 $n$ ）的指针。
- 6. 如果节点 $n$ 不是终止节点，且它不可扩展，则作下列工作：
  - 6.1 标识 $n$ 为不可解节点。
  - 6.2 把 $n$ 的先辈节点中的不可解节点都标识为不可解节点。
  - 6.3 若初始节点 $S_0$ 也被标识为不可解节点，则失败退出。
  - 6.4 否则，从**OPEN**表中删去具有不可解先辈的所有节点。
- 7. 转第2步。

# 与或图的广度和深度优先搜索例子

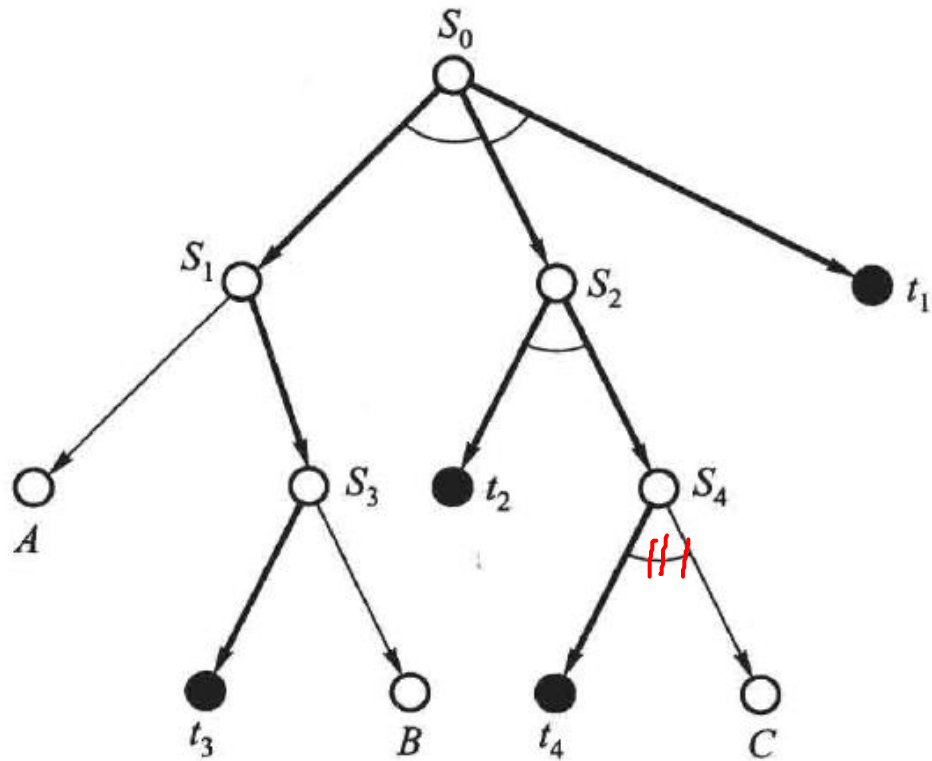


图 5.20 与/或树的广度优先搜索

## 5.5 与或图的启发式搜索

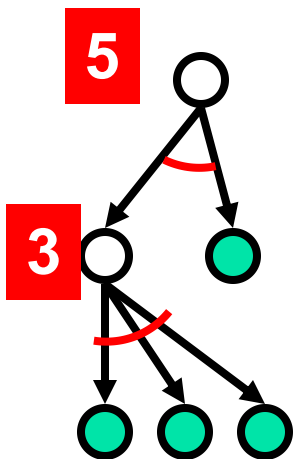
与或图中搜索的是解图，不是解答路径；

- 评价函数 $f(n)=h(n)$
- $h(n)$ 是对 $n$ 到终节点集合最小解图代价的估计；

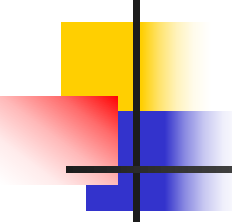
# 与或图搜索

解图代价 ——

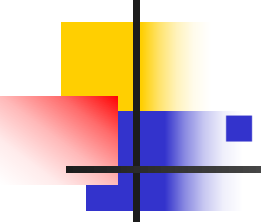
- 1)  $n$  为终止节点, 则代价  $h(n)=0$
- 2)  $n$  为或节点, 子节点为  $n_1, n_2, \dots, n_k$ , 则  $n$  的代价为  $h(n)=\min\{c(n, n_i)+h(n_i)\} \quad 1 \leq i \leq k$
- 3)  $n$  为与节点, 子节点为  $n_1, n_2, \dots, n_k$ , 则  $n$  的代价为:  
 $h(n)=\max\{c(n, n_i)+h(n_i)\} \quad 1 \leq i \leq k$  或者  
 $h(n)=\sum_{1 \leq i \leq k} \{c(n, n_i)+h(n_i)\}$
- 4)  $n$  是叶节点但不是终止节点, 则代价定义为:  
 $h(n)=\text{无穷大}$
- 5) 根节点的代价即为解图的代价



# 希望解树

- 
1. 初始节点 $S_0$ 在希望解树 $T$ 中;
  2. 如果节点 $n$ 在希望解树中:
    - (a) 如果 $n$ 是或节点, 且它有 $k$ 个子节点 $n_1, n_2, \dots, n_k$ , 则其某个子节点 $n_i$ 在希望解树 $T$ 中的充分必要条件是:  
 $n_i$  使  $c(n, n_i) + h(n_i)$  ( $1 \leq i \leq k$ ) 取值最小
    - (b) 如果 $n$ 是与节点, 则 $n$ 的全部子节点都在希望解树 $T$ 中。

# 与或图的最佳优先搜索算法思想

- 
- 与或图的最佳优先搜索是用来求取代价最小的解树的一种搜索方法，为了求得代价最小的解树，就要在每次确定欲扩展的节点时，先往前多看几步，计算以下扩展这个节点可能要付出的代价，并选择代价最小的节点进行扩展。是一个不断地选择、修正希望解树的过程。

## 最佳优先搜索过程：

自顶向下，图生成过程，扩展节点，从希望树中选择一个节点扩展。

自底向上，计算代价过程，修正代价估值，重新选择希望树



# 与或图的最佳优先搜索算法

## 阶段一：初始化

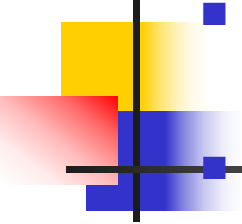
- 1. 把初始节点 $S_0$ 放入OPEN表中。

## 阶段二：搜索循环

- 2. 求出希望树T，即根据当前搜索树中节点的代价h求出以 $S_0$ 为根的希望树T。
- 3. 依次在OPEN表中选出T的端节点n选出放入CLOSED表中。
- 4. 如果节点n是终止节点，则作如下工作：
  - 4.1 标识n为可解节点。
  - 4.2 对T应用可解标识过程，把n的先辈节点中的可解节点都标识为可解节点。
  - 4.3 若初始节点 $S_0$ 能被标识为可解节点，则T就是最优解树，成功退出。
  - 4.4 否则，从OPEN表中删去具有可解先辈的所有节点。



# 与或图的最佳优先搜索算法

- 
- 5. 如果节点 $n$ 不是终止节点，但它可扩展，则作下列工作：
    - 5.1 扩展节点 $n$ ，产生 $n$ 的所有子节点。
    - 5.2 把这些子节点都放入OPEN表中，并为每个子节点配置指向父节点（节点 $n$ ）的指针。
    - 5.3 计算这些子节点的 $h$ 值及其先辈节点的 $h$ 值。
  - 6. 如果节点 $n$ 不是终止节点，且它不可扩展，则作下列工作：
    - 6.1 标识 $n$ 为不可解节点。
    - 6.2 对T应用不可解 标识过程，把 $n$ 的先辈节点中的不可解节点都标识为不可解节点。
    - 6.3 若初始节点 $S_0$ 也被标识为不可解节点，则失败退出。
    - 6.4 否则，从OPEN表中删去具有不可解先辈的所有节点。
  - 7. 转第2步。

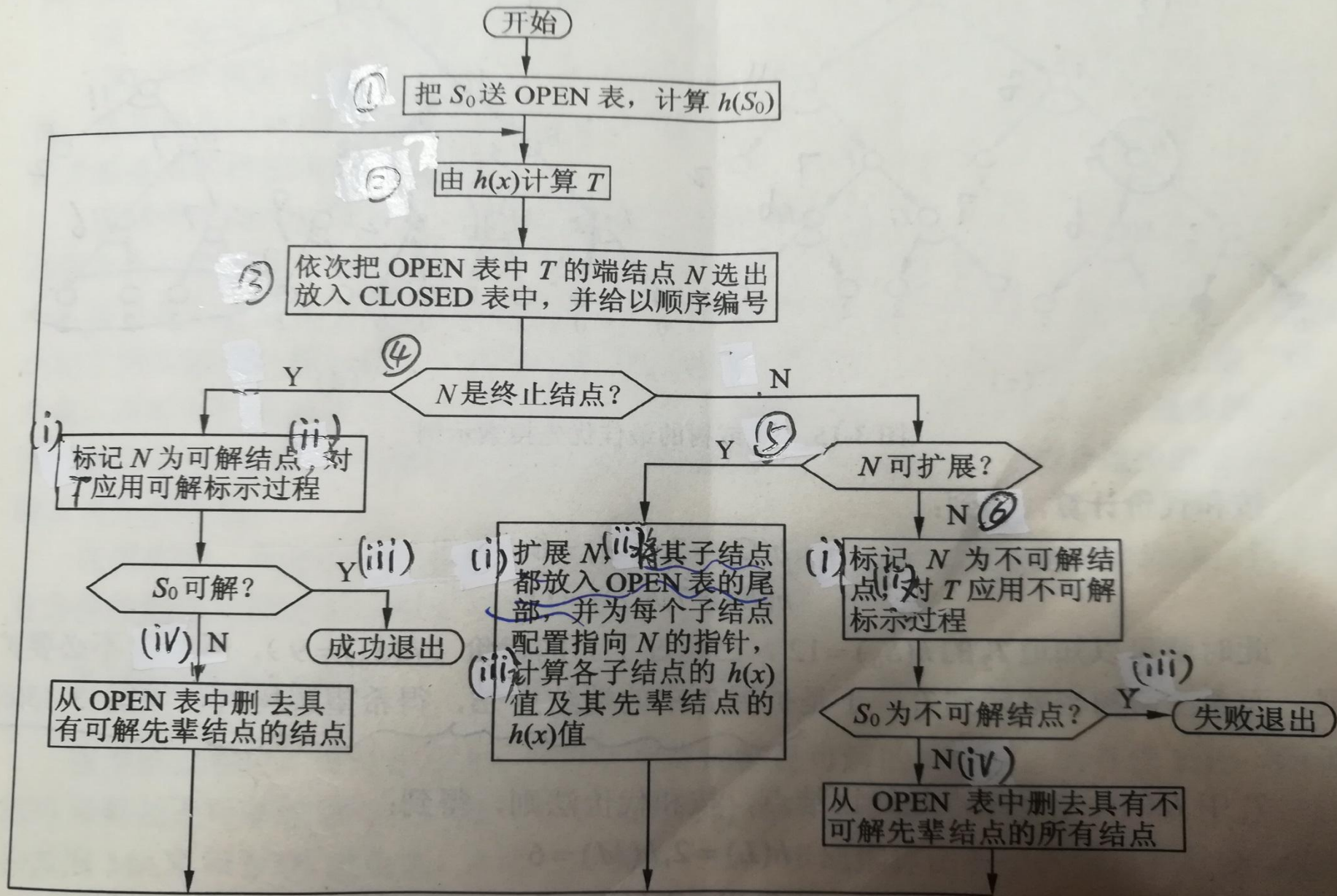


图 2-14 与或树的启发式搜索流程图

# 与或图的最佳优先搜索算法例子

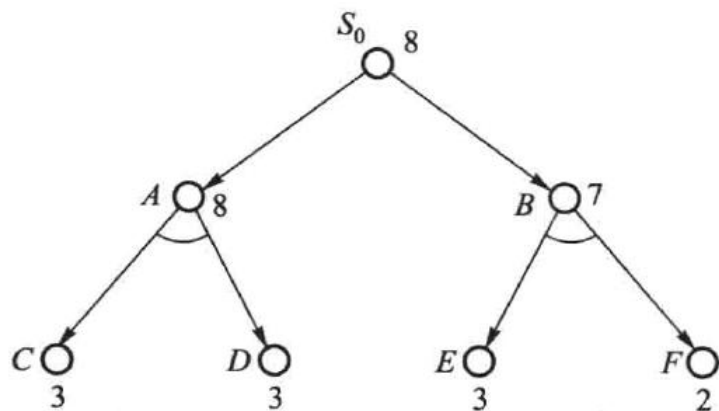


图 5.22  $S_0$  扩展两层后的与/或树

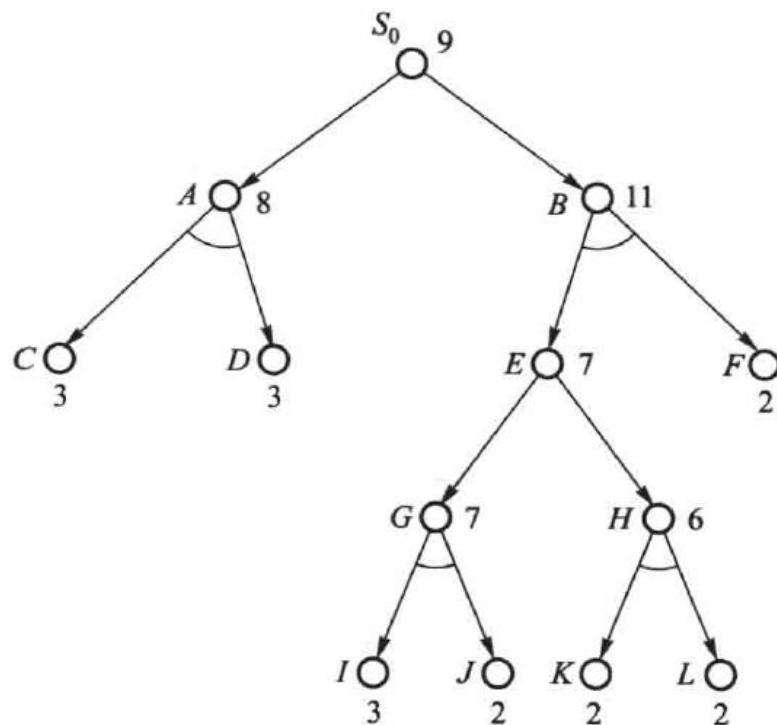


图 5.23 扩展端节点  $E$  后的与/或树

# 与或图的最佳优先搜索算法例子

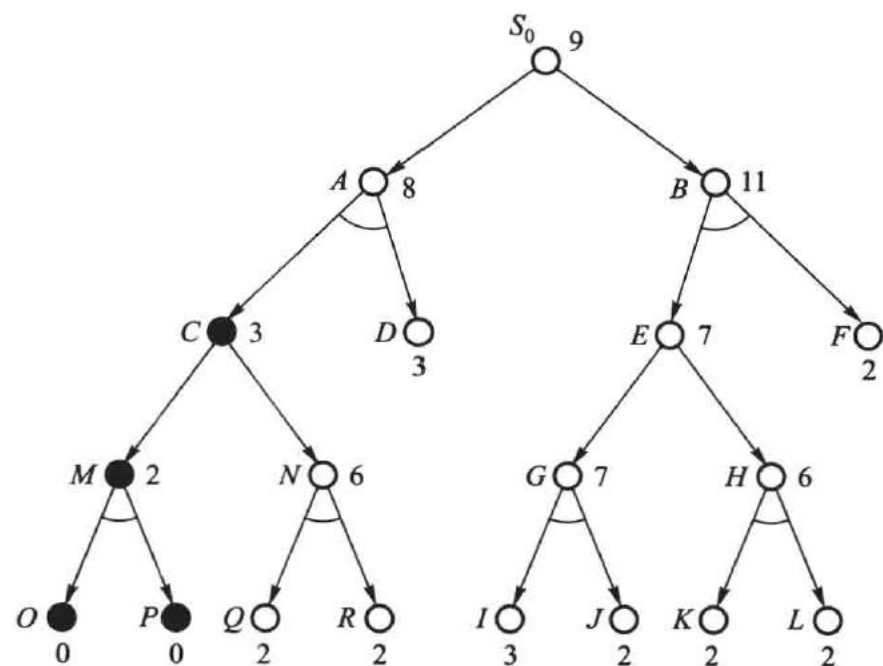


图 5.24 扩展 C 后的与/或树

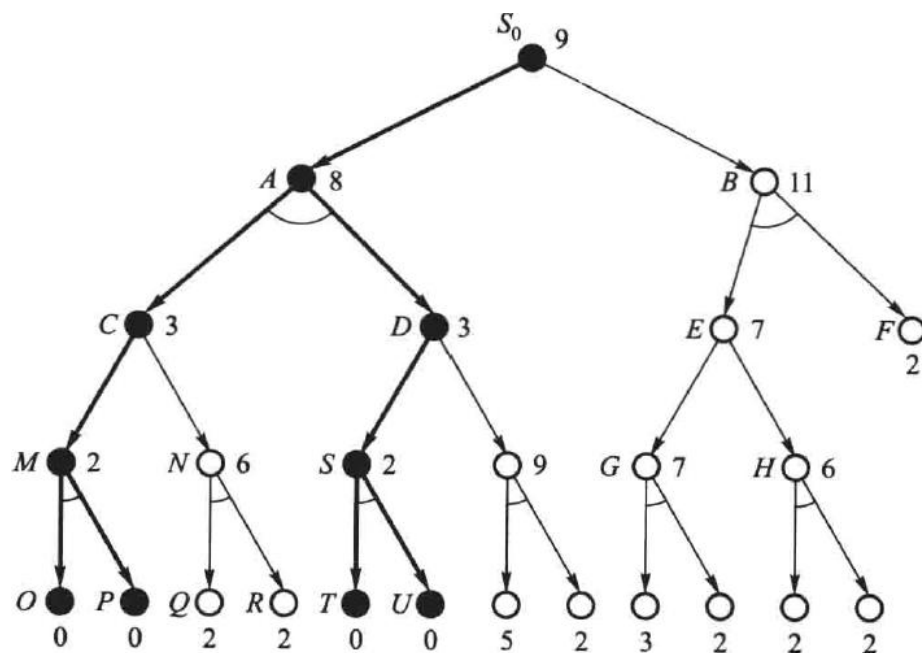


图 5.25 扩展 D 后的与/或树



## 5.6 博弈

- 博弈是一种具有竞争性的智能活动。在这个领域中，具有明确的胜利和失败；

- 博弈问题对人工智能研究提出了严峻的挑战。

- 这里讲的博弈是二人博弈，二人零和、全信息、非偶然博弈。

- “**二人零和**”，是指在博弈中只有“敌、我”二方。且双方的利益完全对立，其赢得函数之和为零，即

$$\phi_1 + \phi_2 = 0$$

式中， $\phi_1$ 为我方赢得(利益)； $\phi_2$ 为敌方赢得(利益)。

即：博弈的双方有三种结局：

(1)我胜： $\phi_1 > 0$ ；敌负： $\phi_2 = -\phi_1 < 0$ 。

(2)我负： $\phi_1 = -\phi_2 < 0$ ；敌胜： $\phi_2 > 0$ 。

(3)平局： $\phi_1 = 0$ ， $\phi_2 = 0$ 。

- “**全信息**”，是指博弈双方都了解当前的格局, 过去已经走过的棋步，还能估计未来对方可能的走步。
- “**非偶然**”，是指博弈双方都可根据得失大小进行分析，选取我方赢得最大，敌方赢得最小的对策，而不是偶然的随机对策。双方都希望自己获胜，因此当一方走步时，都是选择对自己最有利，而对对方最不利的走法。

## 例:

- 假设有七枚钱币，任一选手只能将已分好的一堆钱币分成两堆个数不等的钱币，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，则为输。
- 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。

- 下图已将双方可能的方案完全表示出来了，无论MIN开始时怎么走法，MAX总可以获胜，取胜的策略用双线箭头表示。

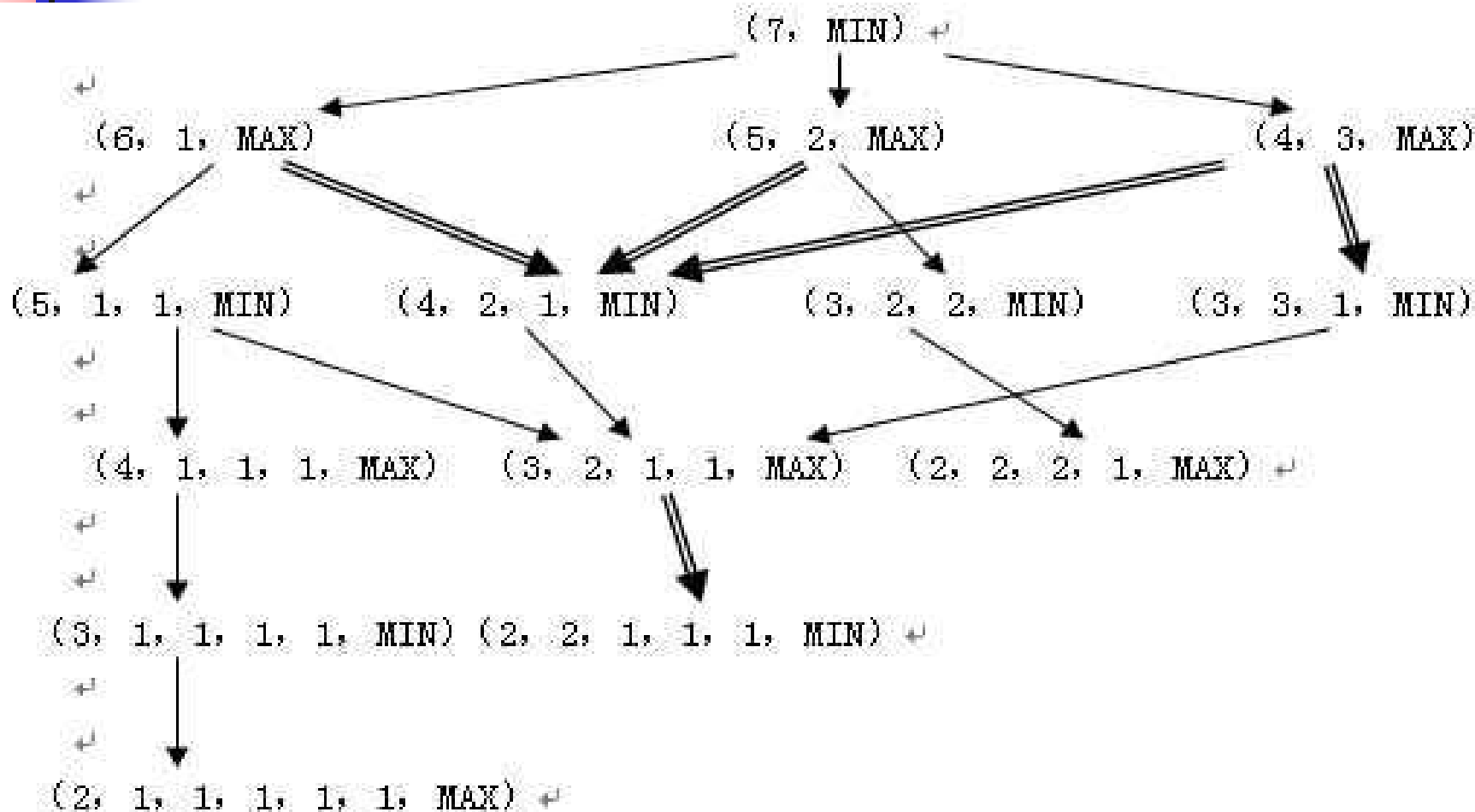


图 3-17 分枝树的博弈



- 双人博弈可用与或图表示。上图是把双人博弈过程用图的形式表示出来，这样就可以得到一棵AND-OR树，这种AND-OR树称为博弈树。

博弈的初始状态是初始节点；

- 博弈树的与节点和或节点是逐层交替出现的；

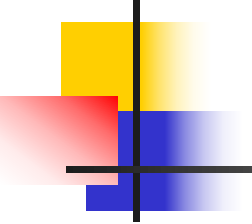
在博弈树中，那些下一步该MAX走的节点称为MAX节点（或节点），而下一步该MIN走的节点称为MIN节点（与节点）。

- 假设博弈双方为MAX和MIN。在博弈的每一步，可供他们选择的方案都有很多种。

从MAX的观点看，

可供自己选择的方案之间是“或”的关系，原因是主动权在自己手里，选择哪个方案完全由自己决定。

而对那些可供MIN选择的方案之间是“与”的关系，这是因为主动权在MIN手中，任何一个方案都可能被MIN选中，MAX必须防止那种对自己最不利的情況出现。



整个博弈过程始终站在MAX一方的立场上，所以能使自己一方获胜的终局都是本原问题，相应的节点也是可解节点，所有使对方获胜的节点都是不可解节点。

- 实际的情况没有这么简单，任何一种棋都不可能将所有情况列尽，因此，只能模拟人“向前看几步”，然后做出决策，决定自己走哪一步最有利。即只能给出几层走法，然后按照一定的估算方法，决定走哪一步棋。等对手回应后再继续搜索。
- MINMAX就是按这种思想建立的过程，而  $\alpha - \beta$  过程是MINMAX过程的改进，可提高效率。



# 博弈树的搜索

---

在人工智能中可以采用搜索方法来求解博弈问题，下面就来讨论博弈中最基本的搜索方法。

# 极大极小过程

- 人是如何下棋的呢？人实际上采用的是一种试探性的方法。首先假定走了一步棋，看对方会有那些应法，然后再根据对方的每一种应法，看我方是否有好的回应.....这一过程一直进行下去，直到若干步以后，找到了一个满意的走法为止。初学者可能只能看一、两个轮次，而高手则可以看几个，甚至十几个轮次。

极小极大搜索方法，模拟的就是人的这样一种思维过程。当轮到我方走棋时，

- 首先按照一定的搜索深度生成出给定深度 $d$ 以内的所有状态，计算所有叶节点的评价函数值。
- 然后从 $d-1$ 层节点开始逆向计算：对于我方要走的节点（用MAX标记，称为极大节点）取其子节点中的最大值为该节点的值（因为我方总是选择对我方有利的棋）；对于对方要走的节点（用MIN标记，称为极小节点）取其子节点中的最小值为该节点的值（对方总是选择对我方不利的棋）。一直到计算出根节点的值为止。获得根节点取值的那一分枝，即为所选择的最佳走步。



# 极大极小过程

---

- 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 $f$ 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

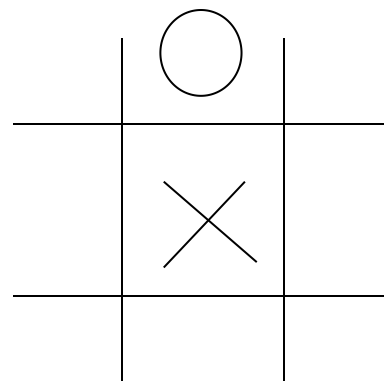
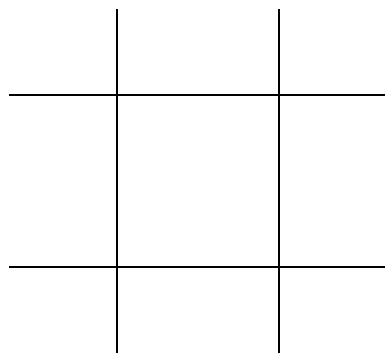
其中 $p$ 代表棋局。

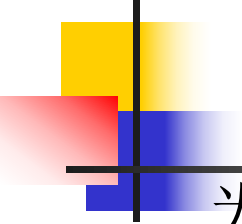


## MINMAX基本思想:

- (1) 当轮到MIN走步的（与）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（或）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。

- 用一字棋说明极大极小过程，设只进行两层，即每方只走一步。
- 一字棋游戏规则如下：设有一个三行三列的棋盘，如图所示，两个棋手轮流走步，每个棋手走步时往空格上摆一个自己的棋子，谁先使自己的棋子成三子一线为赢。设MAX方的棋子用×标记，MIN方的棋子用○标记，并规定MAX方先走步。





为了不致于生成太大的博弈树，假设每次仅扩展两层。估价函数定义如下：

设棋局为 $P$ ，估价函数为 $e(P)$ 。

- (1) 若格局 $p$ 对任何一方都不是获胜的，则

$$e(p) = e(+p) - e(-p)$$

$e(+p)$  : 所有空格都放上MAX的棋子之后，MAX的三子成线（行、列、对角）的总数

$e(-p)$  : 所有空格都放上MIN的棋子之后，MIN的三子成线（行、列、对角）的总数


- (2) 若 $p$ 是MAX获胜，则

$$e(p) = +\infty$$

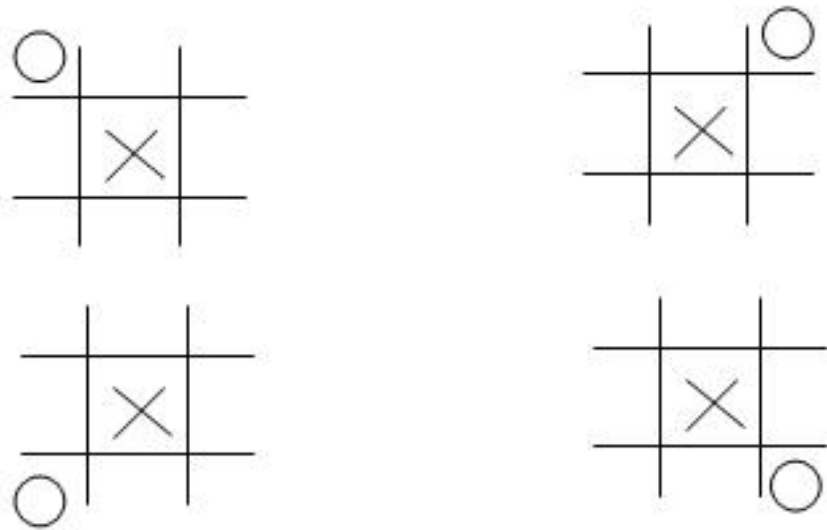
- (3) 若 $p$ 是MIN获胜，则

$$e(p) = -\infty$$





例：若p为下图所示：



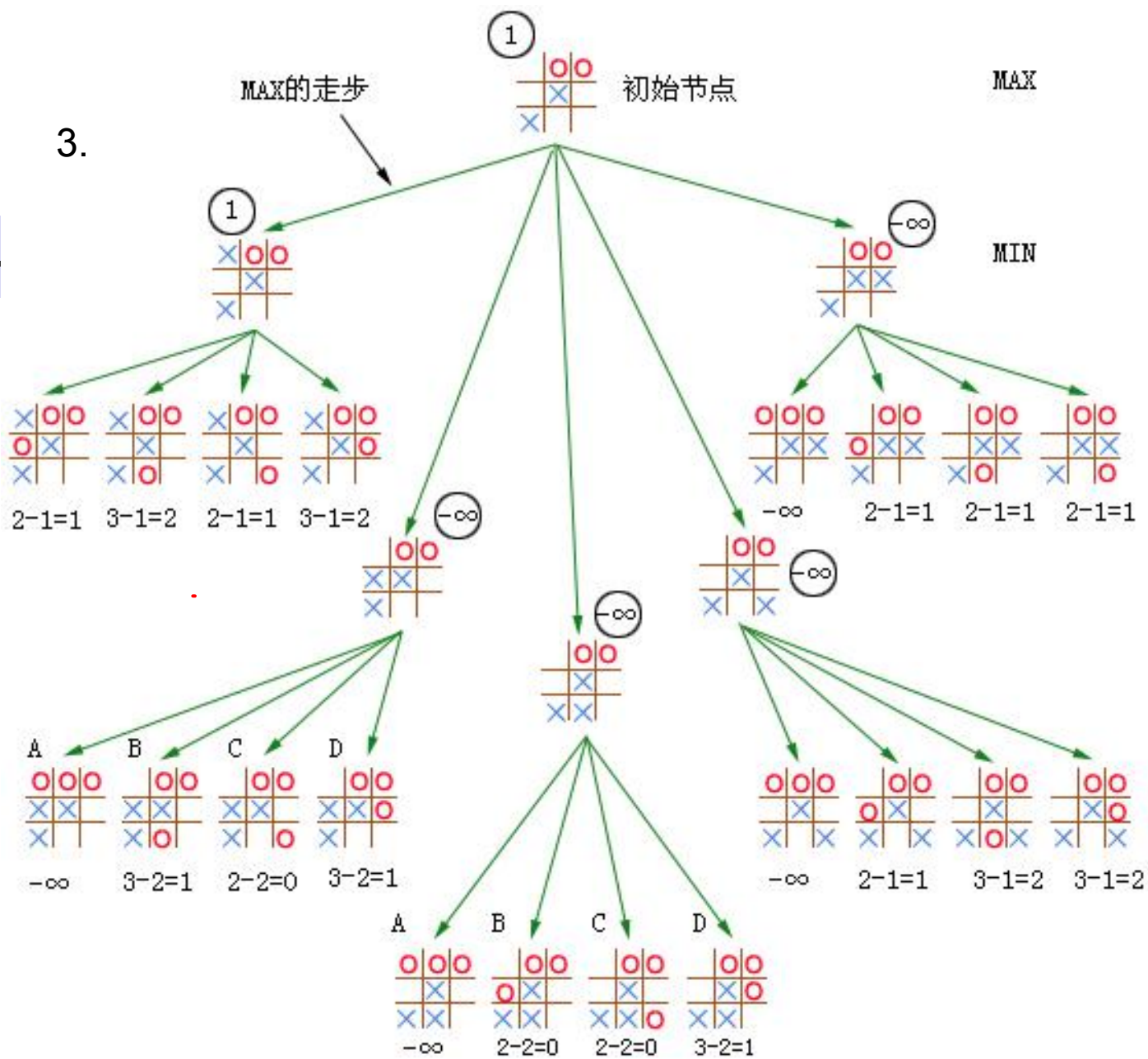
■  $e(P) = e(+P) - e(-P) = 5 - 4 = 1$

- 1.





3.



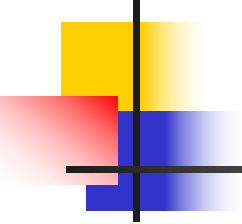
乔瑞

# $\alpha - \beta$ 过程

上面讨论的极大极小过程先生成一棵博弈搜索树，而且会生成规定深度内的所有节点，其节点数将随着搜索深度的增加呈指数增长，然后再进行估值的倒推计算，这样使得生成博弈树和估计值的倒推计算两个过程完全分离，因此搜索效率较低。

- 如图 3 中，其中一个MIN节点要全部生成A、B、C、D四个节点，然后还要逐个计算其静态估值，最后在求倒推值阶段，才赋给这个MIN节点的倒推值 $-\infty$ 。

如果生成节点A后，马上进行静态估值，得知 $f(A) = -\infty$ 之后，就可以断定再生成其余节点及进行静态计算是多余的，可以马上对MIN节点赋倒推值 $-\infty$ ，而丝毫不会影响MAX的最好优先走步的选择。这是一种极端的情况。

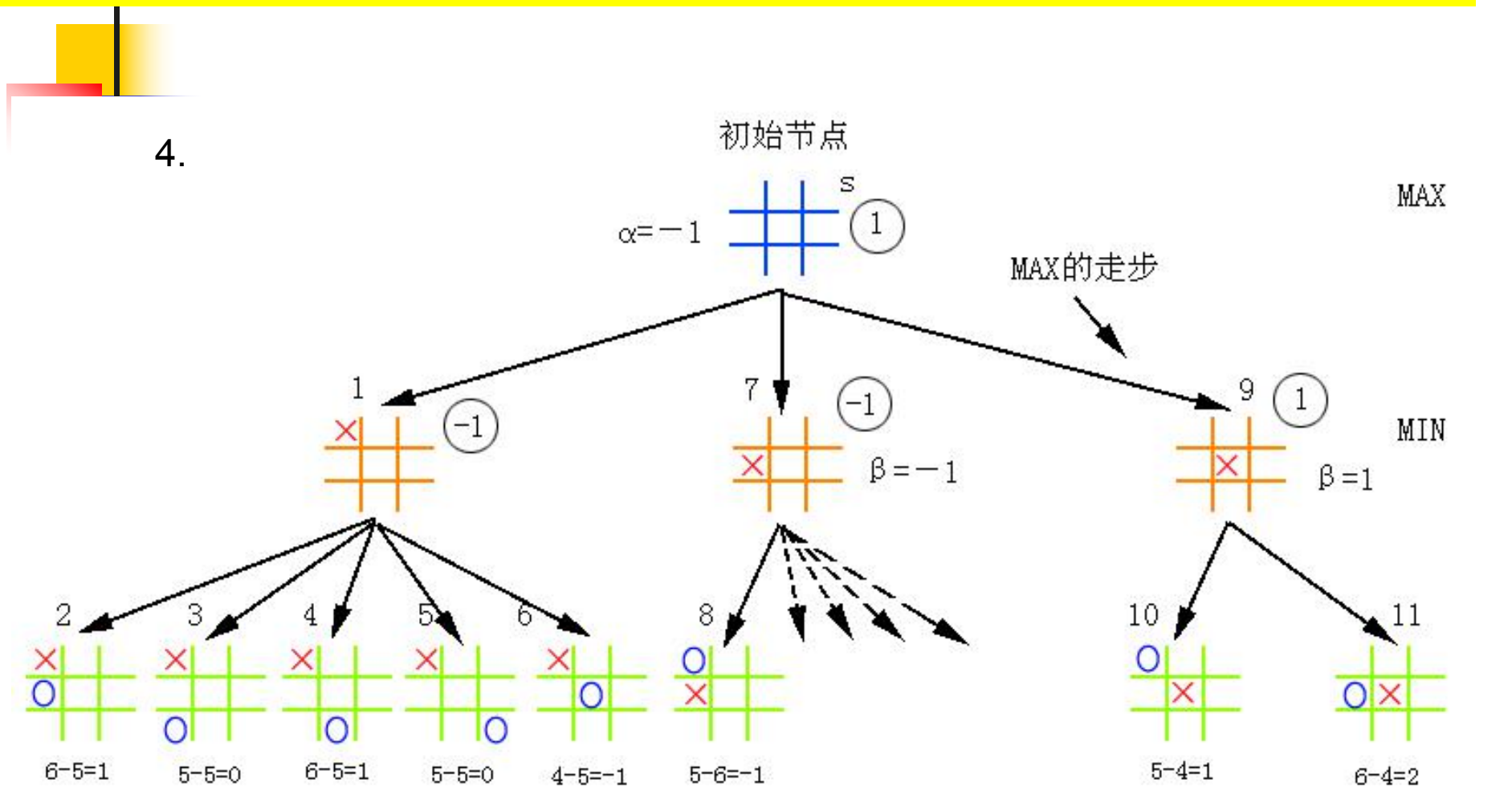


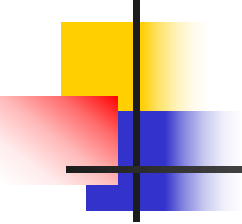
推广：能否在搜索深度不变的情况下，利用已有的搜索信息减少生成的节点数呢？如果能边生成博弈树，边进行估值的计算，则可能不必生成规定深度内的所有节点，以减少搜索的次数，这就是下面要讨论的  $\alpha - \beta$  过程。

- $\alpha$  值：针对或节点。当前子节点中的最大倒推值（或估计值）。
- $\beta$  值：针对与节点。当前子节点中的最小倒推值（或估计值）。



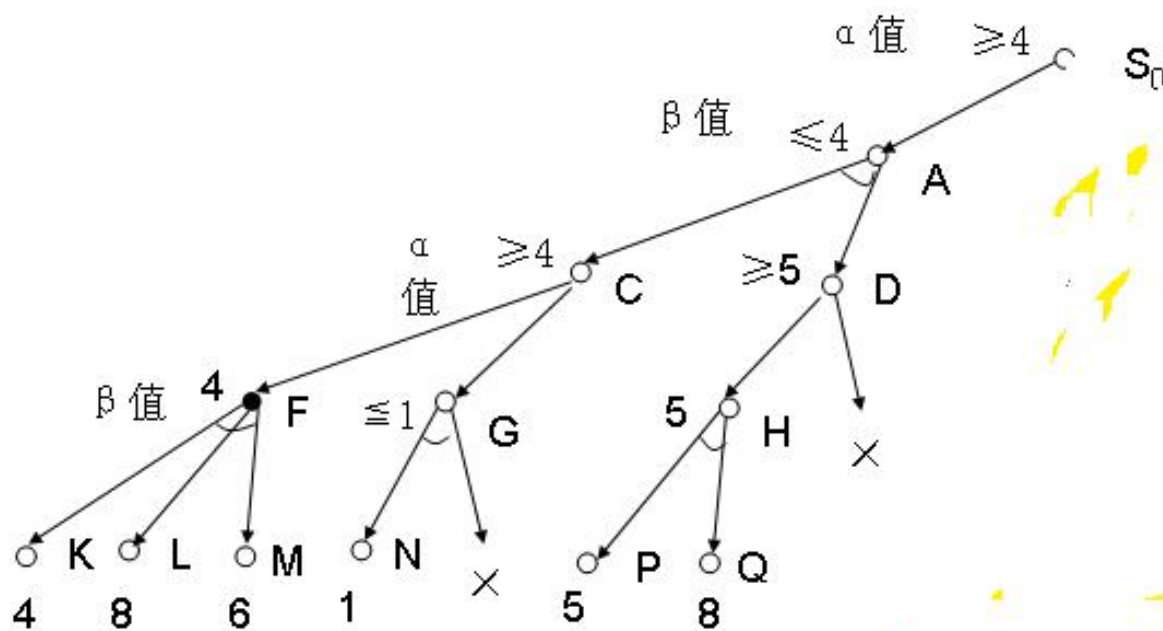
- $\alpha - \beta$  过程就是把生成后继和倒推值估计结合起来，及时剪掉一些无用分支，以此来提高算法的效率。
- 下面仍然用一字棋进行说明。现将原图左边所示的一部分重画在图中。



- 
- 比如在节点A处，若已生成5个子节点，并且A处的倒推值等于-1，我们将此下界叫做MAX节点的  $\alpha$  值，即  $\alpha = -1$ 。
  - 现在轮到节点B，产生它的第一后继节点C，C的静态值为-1，可知B处的倒推值  $\leq -1$ ，此为上界MIN节点的  $\beta$  值，即B处  $\beta = -1$ ，这样B节点最终的倒推值可能小于-1，但绝不可能大于-1，因此，B节点的其他后继节点的静态值不必计算，自然不必再生成，反正B决不会比A好，所以通过倒推值的比较，就可以减少搜索的工作量



再看一个例子，如下图所示。其中最下面一层端节点旁边的数字是假设的估值。



# $\alpha$ 剪枝和 $\beta$ 剪枝

因此可以利用上述**规律**进行剪枝，一般可以停止对某个节点搜索，即**剪枝的规则**表述如下：

- (1) 若任何MIN节点的  $\beta$  值小于或等于任何它的先辈MAX节点的  $\alpha$  值, 即 $\alpha$  (先辈层)  $\geq \beta$  (后继层), 则可停止该MIN节点以下的搜索, 然后这个MIN节点的最终倒推值即为它已得到的  $\beta$  值。
- (2) 若任何MAX节点的  $\alpha$  值大于或等于它的MIN先辈节点的  $\beta$  值, 即 $\alpha$  (后继层)  $\geq \beta$  (先辈层), 则可以停止该MAX节点以下的搜索, 然后这个MAX节点处的倒推值即为它已得到的  $\alpha$  值。

当满足规则(1)而减少了搜索时, 进行了  $\alpha$  剪枝;  
当满足规则(2)而减少了搜索时, 进行了  $\beta$  剪枝。

确定  $\alpha$  和  $\beta$  值, 并且一旦可能就进行剪枝的整个过程通常称为  $\alpha - \beta$  过程,