第三章 基本搜索

- 第3.1节 搜索问题概述
- 第3.2节 状态空间搜索
- 第3.3节 问题空间搜索
- 第3.4节 博弈空间搜索

第三章 基本搜索

- 问题求解系统划分为两大类
  - 知识贫乏系统
    - 依靠搜索技术解决问题
    - 知识贫乏、缺乏针对性
    - 效率低
  - 知识丰富系统
    - 依靠推理技术解决问题
    - 基于丰富知识的推理技术，直截了当
    - 效率高

第3.1节 搜索问题概述

- **3.1.1** 搜索的概念
- **3.1.2** 搜索的分类
- **3.1.3** 各种搜索算法的不同特点及其评价

3.1.1 搜索的概念

- 给定的问题，智能系统的行为一般是找到能够达到所希望目标的动作序列，并使其所付出的代价最小、性能最好。
- 基于给定的问题，搜索就是找到智能系统的动作序列的过程。
- 搜索算法的输入是给定的问题，输出是表示为动作序列的方案。用搜索方法求解问题就是依据给定问题的信息，寻找从初始状态到某个目标状态的具有最优费用的动作序列。一旦有了方案，就可以执行该方案所给出的动作。

3.1.1 搜索的概念

- 搜索问题主要关心下述问题：
  - (1) 在哪里搜索（搜索空间）；
  - (2) 从哪里开始搜索（初始状态）；
  - (3) 搜索什么（目标）；
  - (4) 怎样从一个状态到达另一个状态（动作）；
  - (5) 怎样评价动作序列的优劣（费用计算）。

- 搜索问题可以表示为一个五元组(S, O, I, G, C)，其中：
  - (1) **状态集合 S**：描述问题所有可能的状态；
  - (2) **操作符集合 O**：把一个问题从一个状态变换为另一个状态的动作集合；
  - (3) **初始状态 I**：定义问题的初始状态，一般地，初始状态是唯一的；
  - (4) **目标检测函数G**：用来确定一个状态是不是目标状态；
  - (5) **费用计算函数C**：对每个从初始状态到某个结点的动作序列赋予一定费用的函数。
- 状态集合也称为**搜索空间**。和通常的搜索空间不同，人工智能中大多数问题的搜索空间在问题求解之前不是全部知道的，搜索空间常常是无限的或者虽然在理论上有限但实际上太大从而可以被认为是无限的，往往需要在搜索的过程中动态地逐步进行生成。

3.1.2 搜索的分类

- 搜索可以按不同的观点进行分类。
- 按搜索空间的表示方式或应用目的可以将搜索分成：
  - (1) **状态空间搜索；**
  - (2) **问题空间搜索；**
  - (3) **博弈空间搜索。**
- 按搜索是否使用启发式信息可以将搜索分成：
  - (1) **盲目搜索（无信息搜索）；**
  - (2) **启发式搜索。**

3.1.2 搜索的分类

- 按搜索实现的控制方式可以将搜索分成：
  - (1) **宽度优先搜索**；
  - (2) **深度优先搜索**；
  - (3) **有序搜索**。
- 按搜索所使用的数据结构可以将搜索分成：
  - (1) **树搜索**；
  - (2) **一般图搜索**；
  - (3) **与或图搜索**。

- 在本章中，我们主要关心状态空间、问题空间和博弈空间中问题求解的启发式搜索。问题空间搜索和博弈空间搜索是特殊的状态空间搜索方法。

- **状态空间搜索**首先将问题的全部可能状态及其关系表示出来，在搜索过程中，从初始状态出发，逐步遍历其状态空间，并通过识别当前状态是否为目标状态达到求解问题的目的。
- **问题空间搜索**用问题归约的方法进行搜索，在问题求解过程中，将问题逐步进行分解，变换成若干个子问题，通过求解子问题完成对原问题的求解。
- **博弈空间搜索**主要是利用已有的搜索技术用于求解博弈问题，并根据博弈问题的基本特点对搜索的控制进行优化。

3.1.3 各种搜索算法的不同特点及其评价

盲目搜索的主要特点：

- (1) 只是可以区分出哪个是目标状态；
- (2) 一般是按预定的搜索策略进行搜索；
- (3) 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。

启发式搜索的主要特点：

- (1) 使用启发式信息指导搜索过程，可以在较大的程度上提高搜索算法的时间效率和空间效率；
- (2) 启发式搜索的效率在于启发式函数的优劣，在启发式函数构造不好的情况下，甚至在存在解的情形下也可能导致解的丢失现象或者找不到最优解。

搜索策略评价标准：

- 完备性：
  - 如果存在一个解答，该策略是否保证能够找到？
- 时间复杂性：
  - 需要多长时间可以找到解答？
- 空间复杂性：
  - 执行搜索需要多少存储空间？
- 最优性：
  - 如果存在不同的几个解答，该策略是否可以发现最高质量的解答？

第3.2节 状态空间搜索

- 3.2.1 状态空间描述
- 3.2.2 一般图的盲目搜索
- 3.2.3 一般图的启发式搜索
- 3.2.4 状态空间抽象和生成- 测试法

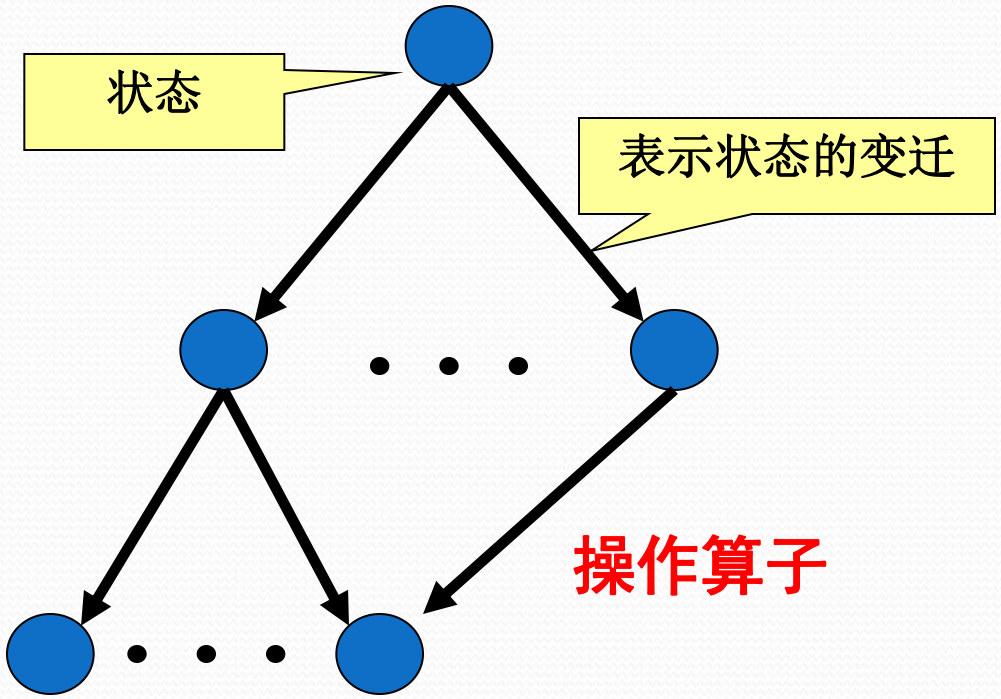
3.2.1 状态空间描述

1. 状态空间的表示

- 状态空间记为SP，可表示为二元组SP=(S,O):
  - S——问题求解（即搜索）过程中所有可能到达的合法状态构成的集合；
  - O——操作算子的集合，操作算子的执行会导致问题状态的变迁；
  - 状态——用于记载问题求解（即搜索）过程中某一时刻问题现状的快照；
    - 抽象为矢量形式 $Q=[q_0,q_1,\cdots,q_n]^T$
    - 每个元素q_i称为状态分量
    - 给定每个状态分量q_i的值就得到一个具体的状态

$$Q_k=[q_{0k},q_{1k},\cdots,q_{nk}]^T$$

用状态空间搜索来求解问题的系统均定义一个状态空间，并通过适当的搜索算法在状态空间中搜索解答路径。



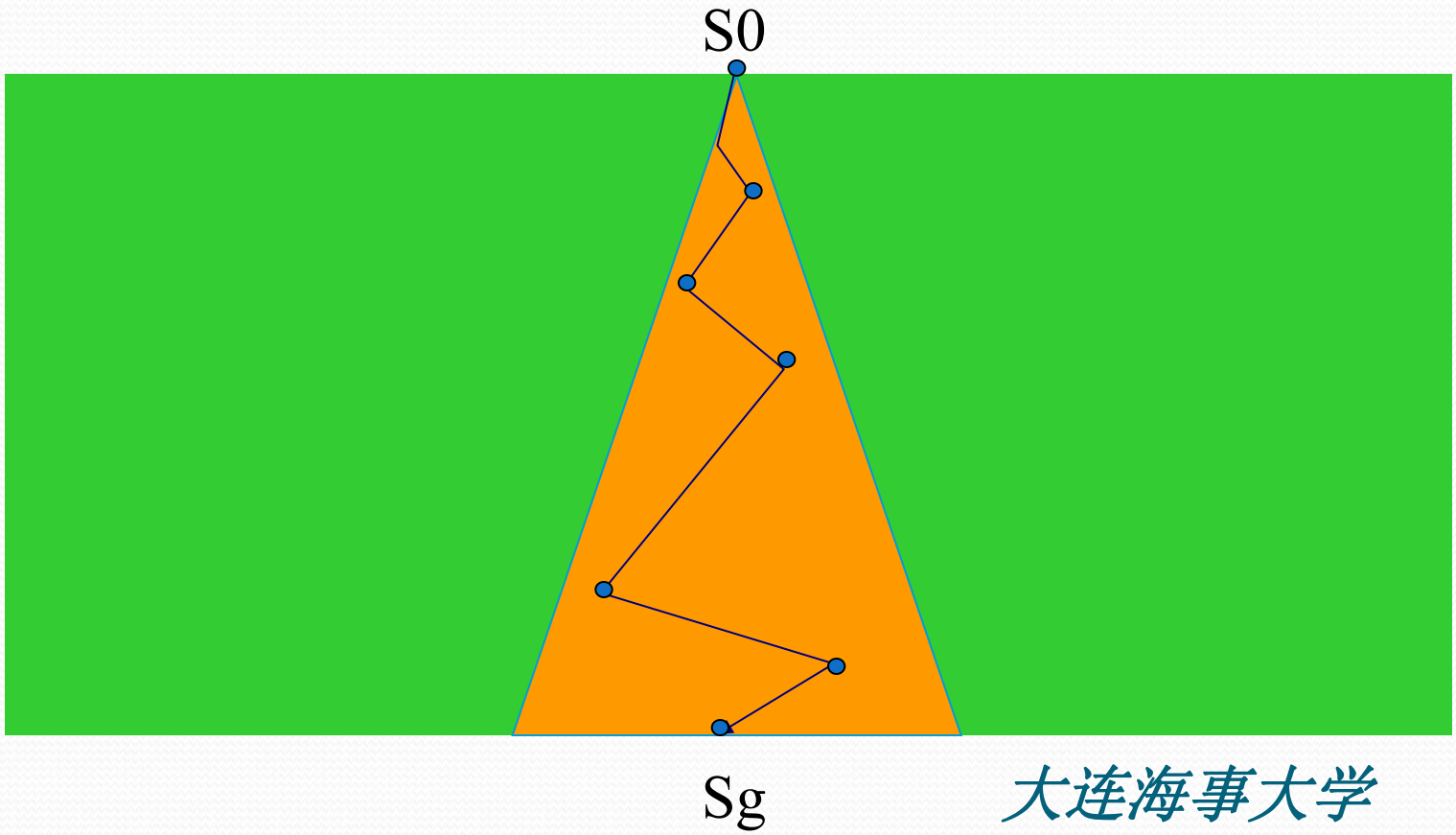
- 节点
  - 状态
- 有向弧
  - 状态的变迁
- 弧上的标签
  - 导致状态变迁的操作算子

问题的状态空间是一个表示该问题的全部可能状态及其变迁的有向图。

2.状态空间的搜索

- 状态空间的搜索记为SE，可表示为五元组：
  - SE=(S,O,E,I,G)；
  - E——搜索引擎；
  - I——问题的初始状态，$I \in S$；
  - G——问题的目标状态集合，$G \subset S$。
- 搜索引擎E——可以设计为实现任何搜索算法的控制系统。
- 基本思想——通过搜索引擎E寻找一个操作算子的调用序列，使问题从初始状态I变迁到目标状态G之一。
- 解答路径——初-目变迁过程中的状态序列或相应的操作算子调用序列。

- 状态空间、搜索图和解答路径之间的关系:
  - 状态空间一般都表示为或图（一般图）
  - 搜索图——在搜索解答路径的过程中画出搜索时涉及到的节点和弧线，构成所谓的搜索图。



3.2.2 一般图的盲目搜索

1. 搜索术语

- 节点深度
- 节点扩展
- 路径
- 路径代价——相邻节点n_i和n_{i+1}间的路径代价记为$C(n_i, n_{i+1})$

$C(n_i, n_{i+1})=1$

$C(n_i, n_g) = C(n_i, n_k) + C(n_k, n_g)$

2. 一般图搜索算法

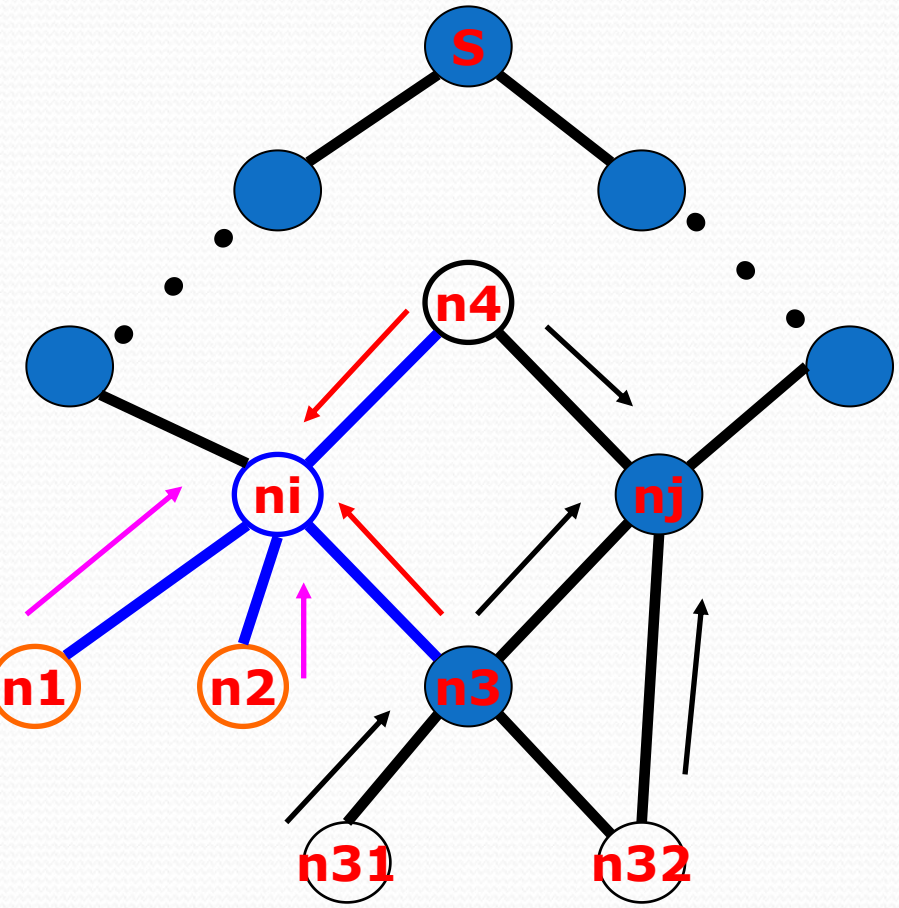
- 符号说明：
  - s-初始状态节点
  - G-搜索图
  - OPEN-存放待扩展节点的表
  - CLOSE-存放已被扩展的节点的表
  - MOVE-FIRST(OPEN)-取OPEN表首的节点作为当前要被扩展的节点n，同时将节点n移至CLOSE表
- 一般图搜索算法划分为二个阶段：
  - 1、初始化
  - 2、搜索循环

2. 一般图搜索算法

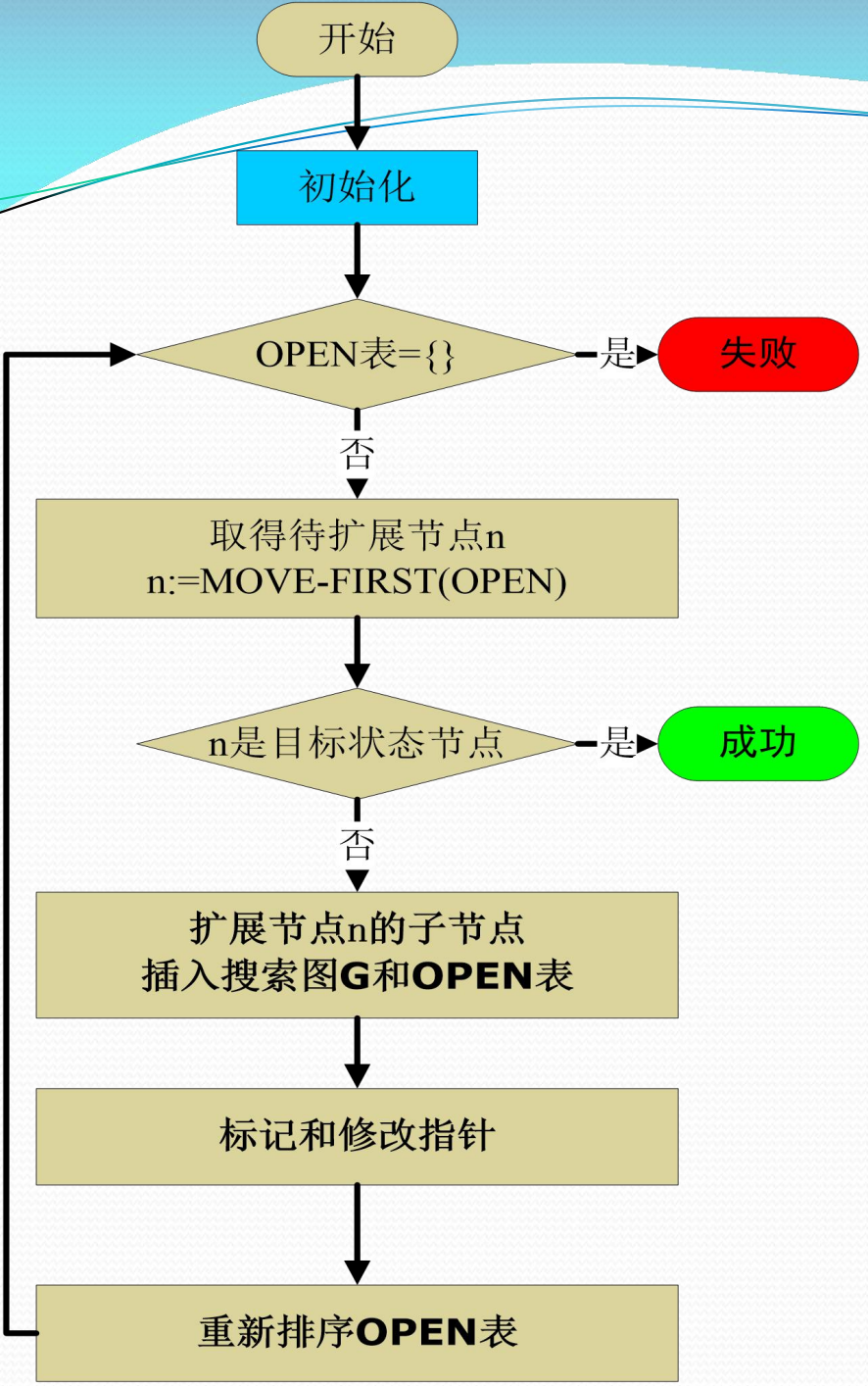
- 算法划分为二个阶段：
  - 1、初始化
    - 建立只包含初始状态节点s的搜索图G:={s}
    - OPEN:={s}
    - CLOSE:={}
  - 2、搜索循环
    - MOVE-FIRST(OPEN)-取出OPEN表首的节点n作为扩展的节点，同时将其移到close表
    - 扩展出n的子节点,插入搜索图G和OPEN表
    - 适当的标记和修改指针
    - 排序OPEN表
  - 通过循环地执行该算法，搜索图G会因不断有新节点加入而逐步长大，直到搜索到目标节点。

2. 一般图搜索算法——搜索过程中的指针修改

- 节点n扩展后的子节点分为3类：
  - (i)全新节点
  - (ii)已出现在OPEN表中的节点
  - (iii)已出现的CLOSE表中的节点
- 指针标记和修改的方法：
  - (i)类节点：加入OPEN表，建立从子节点到父节点n的指针
  - (ii)类节点、 (iii)类节点
    - 比较经由老父节点、新父节点n到达初始状态节点的路径代价
    - 经由新父节点n的代价比较小，则将原子节点指向老父节点的指针，修改为指向新父节点n
    - (iii)类节点还得从CLOSE表中移出，重新加入OPEN表。



- 节点ni是当前扩展的节点；
- 扩展出4个后续节点；
- n1、n2是新节点，只需建立指向父节点的指针，并加入OPEN表；
- n4已经存在于OPEN表，并且已有父节点nj
  - n4经nj的路径代价大
  - 取消n4指向nj的指针
  - 改为建立n4指向新父节点ni的指针
- n3已经存在于CLOSE表，并且已有父节点nj
  - 需要做和n4同样的比较和指针修改工作。并且要重新加入open表。



例：八数码

1		3
7	2	4
6	8	5

初始布局

1	2	3
8		4
7	6	5

目标布局

3.2.2 一般图的盲目搜索

- **OPEN表是一个有序表，结点加入 表的方式决定了搜索策略的不同**。
- 如果费用函数 的计算方法仅依赖于预先对算符代价的假定而不依赖于新结点的产生所带来的信息，使得 中结点的排列方式变得无启发式信息可以利用，这种搜索策略称为**盲目搜索或无信息搜索**。

- 在盲目搜索的情形下，费用计算几乎完全失去了意义，在扩展结点时，只要将没有在OPEN表和CLOSED表中出现的后继结点加入OPEN 表中就可以了。最典型的盲目搜索方法有以下两种：

- (1) 如果OPEN表为堆栈，则搜索算法首先扩展最新产生的（即最深的）结点。深度相等的结点可以任意排列。这种盲目搜索称为**深度优先搜索**。深度优先搜索的特点是，扩展最深的结点的结果使得搜索沿着状态空间某条单一的路径从初始结点向下进行下去，只有当搜索到达一个没有后裔的状态时，它才考虑另一条替代的路径。
- (2) 如果 OPEN表为队列，则搜索算法以接近初始结点的程度依次扩展结点，这种盲目搜索称为**宽度优先搜索**。宽度优先搜索的特点是，逐层进行扩展结点，在对下一层的任一结点进行搜索之前，必须搜索完本层的所有结点。

- (1) 宽度优先搜索是完备的，
  - 即在目标结点存在的情形下，无论搜索空间是否有限，一定能够找到从初始结点到目标结点的费用最小的路径而成功结束。
  - 当然，由于宽度优先搜索可能需要扩展的结点太多，效率极差，理论上可行并非意味着在实际中也是可行的。
- (2) 深度优先搜索不是完备的。
  - 如果搜索方向正确，它找到目标结点的速度是很快的，但在搜索空间无限的情形下，深度优先搜索不能保证可以找到目标结点。

- (3) 在深度优先搜索方法中，为了防止搜索过程沿着无益的路径扩展下去，往往将深度优先法与**回溯法**结合使用，即给出一个结点扩展的最大深度—**深度界限**。
  - 任何结点如果达到了深度界限，那么都将把它们作为没有后继结点处理，并向上回溯。
  - 这种有限深度的深度优先搜索当有目标结点在深度界限内时在理论上可以找到某个目标结点，但未必是最最优的。
- (4) 为了保证深度优先搜索的完备性以及解的最优性，可以将使用**迭代加深控制策略**，
  - 从深度界限为1 开始，使用深度优先搜索，当搜索失败时将深度限制加1，迭代进行深度优先搜索。

3.2.3 一般图的启发式搜索

- 盲目搜索效率低，耗费过多的计算空间与时间。
- 进行搜索技术一般需要某些有关具体问题领域的特性的信息，称为**启发信息**。利用启发信息的搜索方法叫做**启发式搜索**方法。
- 有关具体问题领域的信息常常可以用来简化搜索。
- 一个比较灵活（但代价也较大）的利用启发信息的方法是**应用某些准则来重新排列每一步OPEN 表中所有结点的顺序**。
- 然后，搜索就可能沿着某个被认为是最有希望的边缘区段向外扩展。
- 应用这种排序过程，需要某些估算结点“希望”的量度，这种量度叫做**估价函数**（evaluation function）。

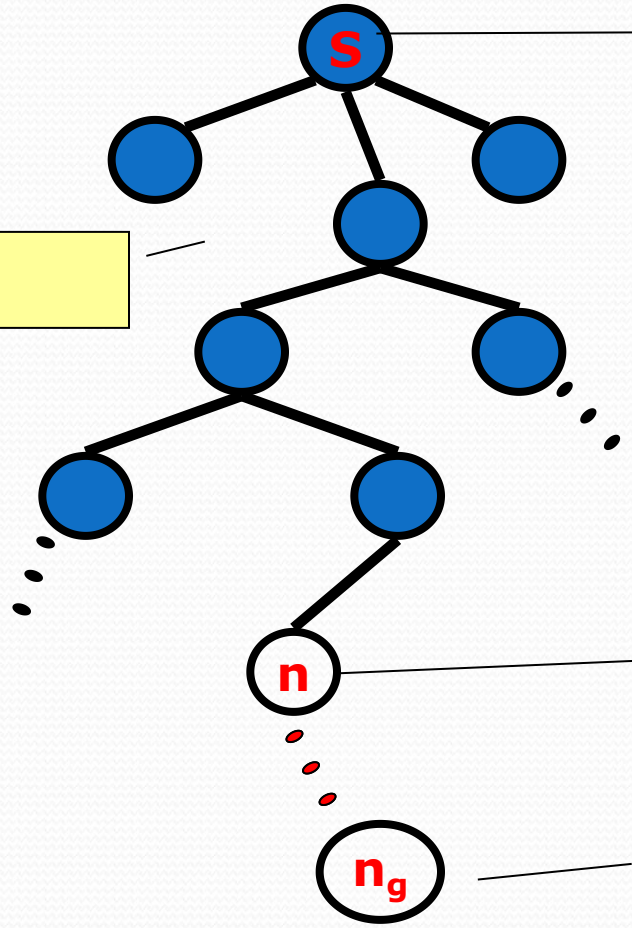
3.2.3 一般图的启发式搜索

- 启发式知识指导OPEN表排序的一般图搜索：
  - 全局排序——对OPEN表中的所有节点排序，使最有希望的节点排在表首。
    - ✓A算法， A*算法
  - 局部排序——仅对新扩展出来的子节点排序，使这些新节点中最有希望者能优先取出考察和扩展；
    - ✓爬山法

一、A算法

- 【基本思想】
  - 设计体现启发式知识的评价函数f(n)；
  - 指导一般图搜索中OPEN表待扩展节点的排序：
- 【评价函数f(n)=g(n)+h(n) 】
  - n-搜索图G中的节点；
  - f(n)- G中从初始状态节点s，经由节点n到达目标节点n_g，估计的最小路径代价；
  - g(n)- G中从s到n，目前实际的路径代价；
  - h(n)-从n到n_g，估计的最小路径代价；
    - h(n)值-依赖于启发式知识加以计算；
    - h(n)称为启发式函数。

一、A算法



搜索图G

初始状态节点s

f(n):s-n-n_g的估计最小路径代价

= g(n):s-n的实际路径代价

+ h(n): n-n_g的估计最小路径代价

节点n

目标状态节点n_g

- **A算法**的设计与一般图搜索相同，划分为二个阶段：

1、初始化

- 建立只包含初始状态节点s的搜索图G:={s}
- OPEN:={s}　　CLOSE:={}

2、搜索循环

- MOVE-FIRST(OPEN)-取出OPEN表首的节点n
- ⑥扩展出n的子节点,插入搜索图G和OPEN表
  - 对每个子节点ni,计算f(n,ni)=g(n,ni)+h(ni)
- ⑦适当的标记和修改指针（子节点→父节点）
  - (i)全新节点：f(ni)=f(n,ni)
  - (ii)已出现在OPEN表中的节点
  - (iii)已出现的CLOSE表中的节点

    IF f(ni)>f(n,ni) THEN　修改指针指向新父结点n，　f(ni)=f(n,ni)
- ⑧排序OPEN表（评价函数f(n)的值排序）

通过循环地执行该算法，搜索图会因不断有新节点加入而逐步长大，直到搜索到目标节点。

A算法实例——八数码游戏

- 设计评价函数f(n)
  - f(n)=d(n)+w(n),其中
  - d(n)-节点n在搜索图中的节点深度，对g(n)的度量；
  - w(n)-代表启发式函数h(n),其值是节点n与目标状态节点n_g相比较，不考虑空格，错位的棋牌个数；

1		3
7	2	4
6	8	5

初始布局

移动数码

1	2	3
8		4
7	6	5

目标布局

1. 搜索算法的可采纳性(Admissibility)

- 1)定义
  - 在存在从初始状态节点到目标状态节点解答路径的情况下，若一个搜索法总能找到最短（代价最小）的解答路径，则称该状态空间中的搜索算法具有可采纳性，也叫最优性。
  - 如，宽度优先的搜索算法是可采纳的，只是搜索效率不高。
- 2) A算法的可采纳性——定义$f^*(n)=g^*(n)+h^*(n)$
  - n-搜索图G中最短解答路径的节点；
  - $f^*(n)$- s经节点n到n_g的最短解答路径的路径代价；
  - $g^*(n)$-该路径前段（从s到n）的路径代价；
  - $h^*(n)$-该路径后段（从n到n_g）的路径代价；

1. 搜索算法的可采纳性(Admissibility)

- 3) 评价函数f与f*的比价
  - f(n)、g(n)、h(n)分别是$f^*(n)$、$g^*(n)$、$h^*(n)$的近似值（估计值）
  - 理想情况下：
    - 若$g(n)=g^*(n)$、$h(n)=h^*(n)$ 则搜索过程中，每次都正确选择，不扩展无关的节点
  - 实际情况：
    - 设计接近f*的f是很困难的
      - g(n)容易从已经生成的搜索树中计算出来，比如就以节点深度d(n)当做g(n),且有$g(n)>=g^*(n)$
    - h(n)尽可能靠近$h^*(n)$ ——A算法的关键。

1. 搜索算法的可采纳性(Admissibility)

- 4)改进启发式函数——八数码游戏
  - f(n)=d(n)+w(n),其中
  - w(n)-表示错位的棋牌个数，不够贴切，错误的扩展了节点d;
  - p(n)-节点n与目标状态节点比较，错位棋牌在不受阻拦的情况下，移动到目标状态相应位置所需走步（移动次数）的总和;
  - p(n)比w(n)更接近于h*(n)-p(n)不仅考虑了棋牌的错位因素，还考虑了错位的距离（移动距离）

1		3
7	2	4
6	8	5

初始布局

移动数码

1	2	3
8		4
7	6	5

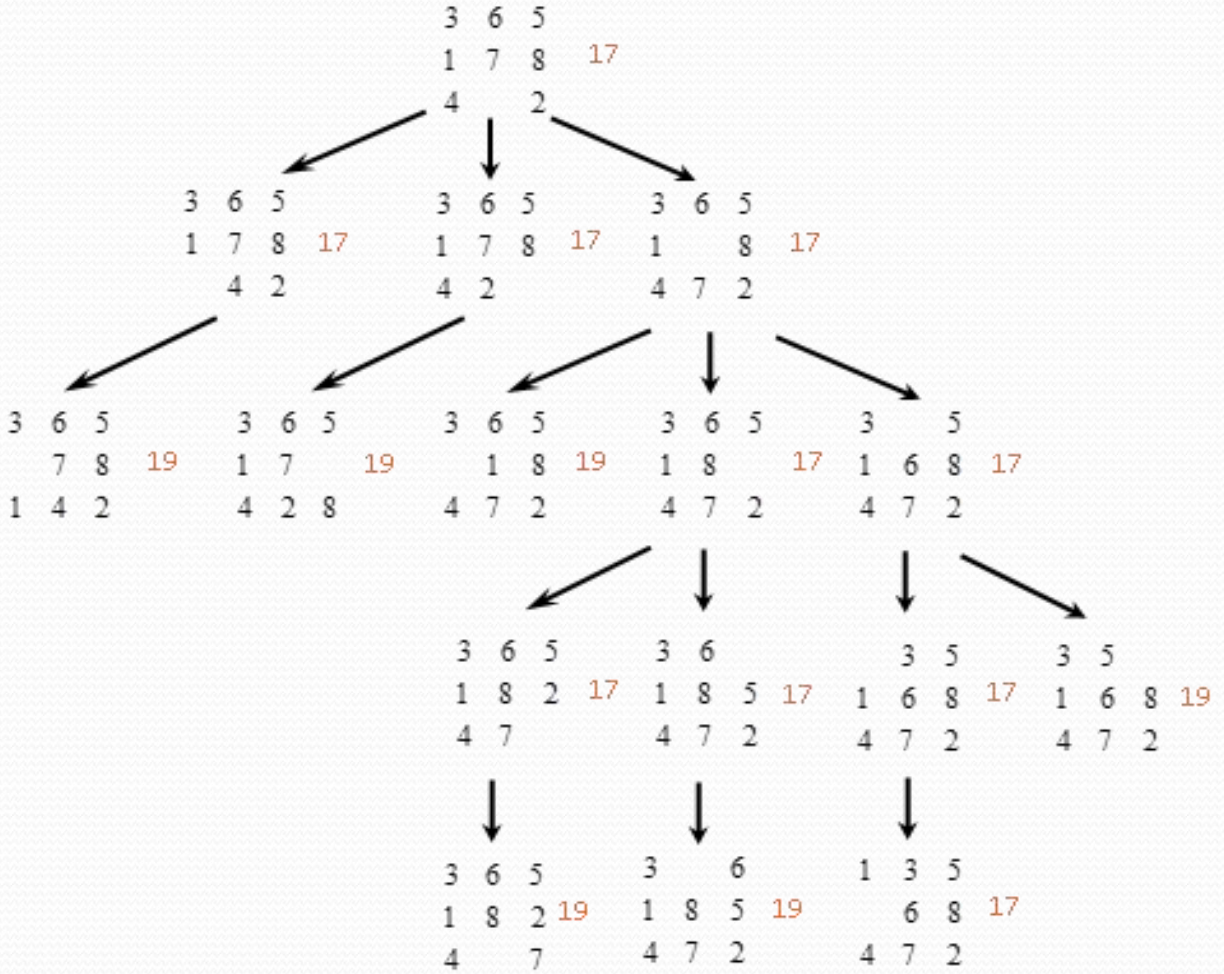
目标布局

二、A*算法

- A*算法定义：
  - 1、在A算法中，规定h(n)≤h*(n);
  - 2、经如此限制以后的A算法就是A*算法。
- A*算法是可采纳的，即总能搜索到最短解答路径
- 八数码：w(n) ≤ p(n) ≤h*(n)， A*算法
  - p(n)比有w(n)更强的启发性信息，因为由h(n)=p(n) 构造的启发式搜索图，比h(n)=w(n)构造的启发式搜索图的结点数要少。

- 例 启发式函数为f(n)=d(n)+p(n),

按照空格左移、下移、右移、上移的顺序考虑各个结点的扩展，则问题求解过程搜索图的前5层情况如右图所示。



对在启发式 算法的估价函数f(n)=g(n)+h(n)，下述几点值得说明。

- (1) h(n)接近h*(n)的程度——衡量启发式函数的强弱
  - h(n)<h*(n)且差距较大时，OPEN表中节点排序的误差较大，h(n)过弱，产生较大的搜索图；
  - h(n)>h*(n)，h(n)过强，A算法失去可采纳性，不能确保找到最短解答路径；
  - h(n)=h*(n)是最理想的， OPEN表中节点排序没有误差，可以确保产生最小的搜索图，搜索到最短解答路径；
    - 无法设计
  - A*算法搜索问题解答的关键
    - h(n)在满足h(n) ≤ h*(n)的条件下，越大越好！

- (2) 在不要求求得最优解的情况下，通过牺牲算法的可采纳性来换取好h(n)设计的简化和减少h(n)的计算量有时是可行的。
- (3) 在f(n)=g(n)+h(n)中，g(n)较大时使得算法更接近于宽度优先； h(n)较大时使得算法更接近于深度优先。为了更有效地搜索解答，可以使用形如f(n)=g(n)+wh(n) 的估价函数，其中w是随深度增加而逐渐减少的权。
- (4) 在 g(n)=o的情况下，只用h(n)对OPEN表中的待扩展结点进行排序，可以实现较为简单的搜索策略：爬山法和回溯法。

3.2.4 状态空间抽象和生成- 测试法

- 状态空间抽象是减少搜索量的重要技术，其本质是将搜索的注意力集中于问题求解的重要因素。
- 对于一个复杂的状态空间，常用的方式是将其按子问题进行划分，子问题构成抽象空间。由于抽象空间一般比原有的状态空间小得多，因此搜索效率可能会得到大幅度的提高。对于复杂的问题，可以采用多级抽象。

- 状态空间搜索有时能表示为**生成- 测试法**。搜索过程由两个部件合作完成，可能解的**生成器**和修剪不正确解答的**测试器**。生成器的性能取决于完备性和无冗余性，知识丰富的生成器常常会导致较高的搜索效率。
- 生成- 测试法可以描述如下：

```
PROCEDURE  Generate & Test
  BEGIN
    REPEAT
      生成一个新的状态，称为当前状态;
    UNTIL 当前状态=目标;
  END.
```

- 例: 从 n个不同元素的集合S中选取 r个不同元素作成不允许重复的 r-组合的一种生成算法。

算法 集合$\{1,2,\cdots,n\}$的所有r-组合的生成算法

1. [初始化]

   i 将要生成的r-组合的第一个位置预置成0

   ii 令$k=1$

2. [逐步生成所有的r-组合]

   反复执行下述步骤直到$k=0$

   i 将第k个位置的数改写成比当前的数大的最小整数

   ii 若第k个位置的数大于n

   则 将k减1

   否则 若$k=r$

   则 输出已经得到了的一个r-排列

   否则 ① 将k加1

   ② 在第k个位置预置上第$k-1$个位置的整数

3. [算法结束]

第3.3节 问题空间搜索

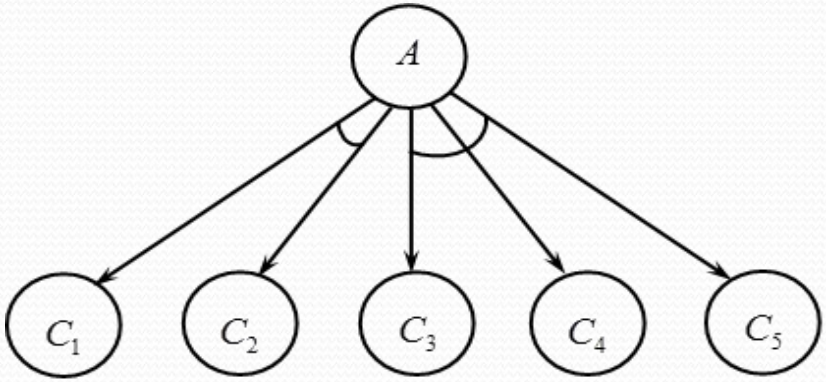
- 3.3.1 问题归约描述
- 3.3.2 与或图的盲目搜索
- 3.3.3 与或图的启发式搜索

3.3.1 问题归约描述

- 启发式搜索可以应用的第二个问题是AND-OR图（与或图）的反向推理问题。
- AND-OR图的反向推理过程可以看作是一个问题归约过程，问题归约是不同于状态空间法的另一种问题描述和求解的方法，所有问题归约的最终目的是产生本原问题。
- 在问题求解过程中，问题归约将一个大的问题变换成若干个子问题，子问题又可以分解成更小的子问题，这样一直分解到可以直接求解为止，全部子问题的解就是原问题的解；称原问题为**初始问题**，可直接求解的问题为**本原问题**。

- **问题归约**可以用三元组（S_0，O，P） 表示，其中
  - (1) S_0是初始问题，即要求解的问题；
  - (2) O是本原问题集，其中的每一个问题是不用证明的，自然成立的，如公理、已知事实等，或已证明过的问题；
  - (3) P是操作算子集，它是一组变换规则，通过一个操作算子把一个问题化成若干个子问题。
- 问题归约表示方法就是由初始问题出发，运用操作算子产生一些子问题，对子问题再运用操作算子产生子问题的子问题，这样一直进行到产生的问题均为本原问题，则问题得解。

- 问题归约过程可以用一个AND-OR图进行表示。在AND-OR图中，结点用于表示问题或子问题，如果问题P的解决可以归约为求解k个问题P1 ,...,Pk ，则从表示问题 P的结点用一条 **k- 连接弧**连接到表示问题 P1 ,...,Pk 的结点。
- 例如，假设问题 A既可通过问题C1 与C2 ，也可通过问题 C3、 C4和C5 来解决。



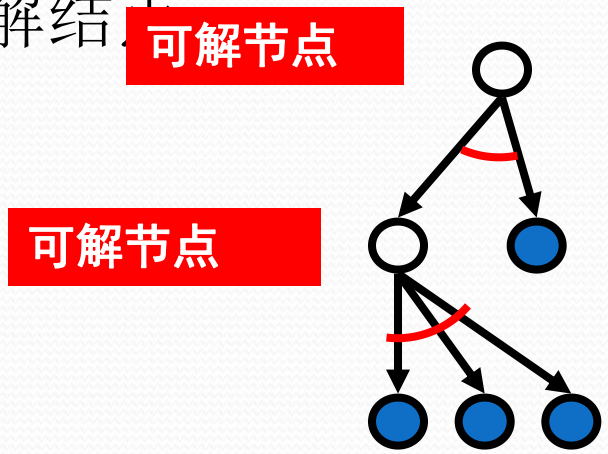
与或图的例子

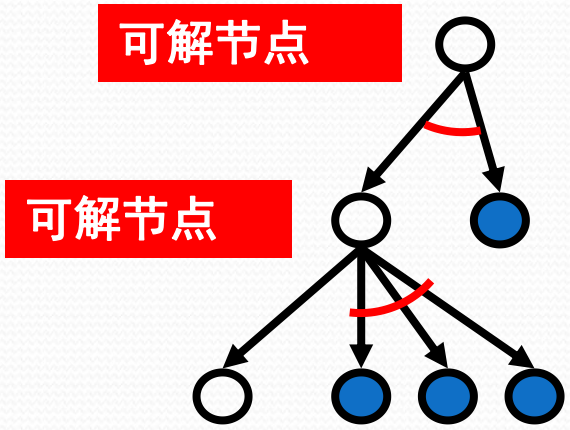
- 问题 C1和 C2构成了问题A 的一个后继问题的集合，问题 C3 、 C4和C5 构成了A 的另一个后继问题集合。

- 由结点及 k- 连接弧组成的图，称为AND-OR图，当所有k均为1时，就变为普通的 OR图。
- k>1的连接弧连接的子结点称为**与结点**；
- k=1的连接弧连接的子结点称为**或结点**。
- 当然，可以对AND-OR图进行变换，引进某些附加结点，以便使含有一个以上后继问题的每个集合能够聚集在它们各自不同的父辈结点之下。
- 将问题求解归约为AND-OR图搜索时，将初始结点表示初始问题描述，对应于本原问题的结点称为**叶结点**。

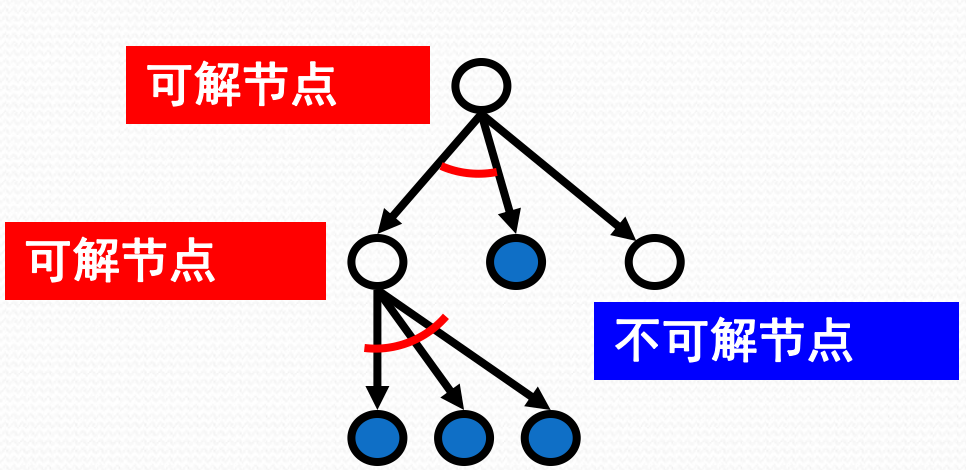
3.3.2 与或图的盲目搜索

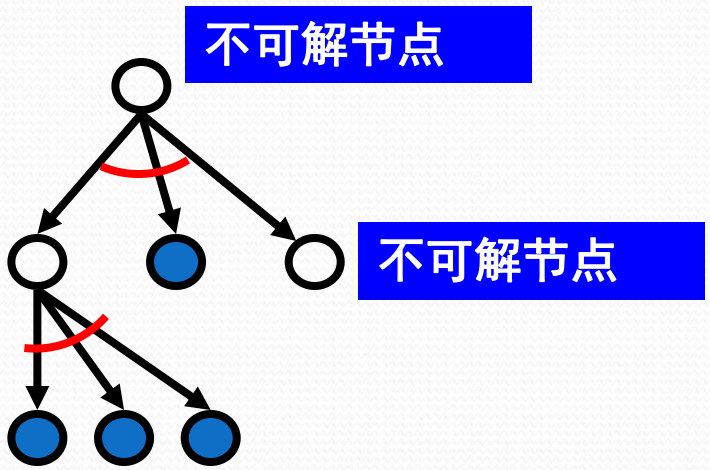
- 在AND-OR图上执行搜索过程，其目的在于表明初始结点是有解的。
- 1. AND-OR图中的一个**可解结点**可递归地定义如下：
  - (1) 叶结点是可解结点；
  - (2) 如果结点 n有一条向外发出的k - 连接弧，则当该k - 连接弧所指向的所有 k个后继结点都是可解结点时， 为n可解结点。





- **2. 不可解结点**可递归定义如下：
  - (1) 没有后裔结点的非叶结点是不可解结点；
  - (2) 若结点n 的每一个向外发出的连接弧都至少指向一个不可解结点，则 n是不可解结点。
- 也就是说，一个结点如果不是可解的，那么它就是不可解的。



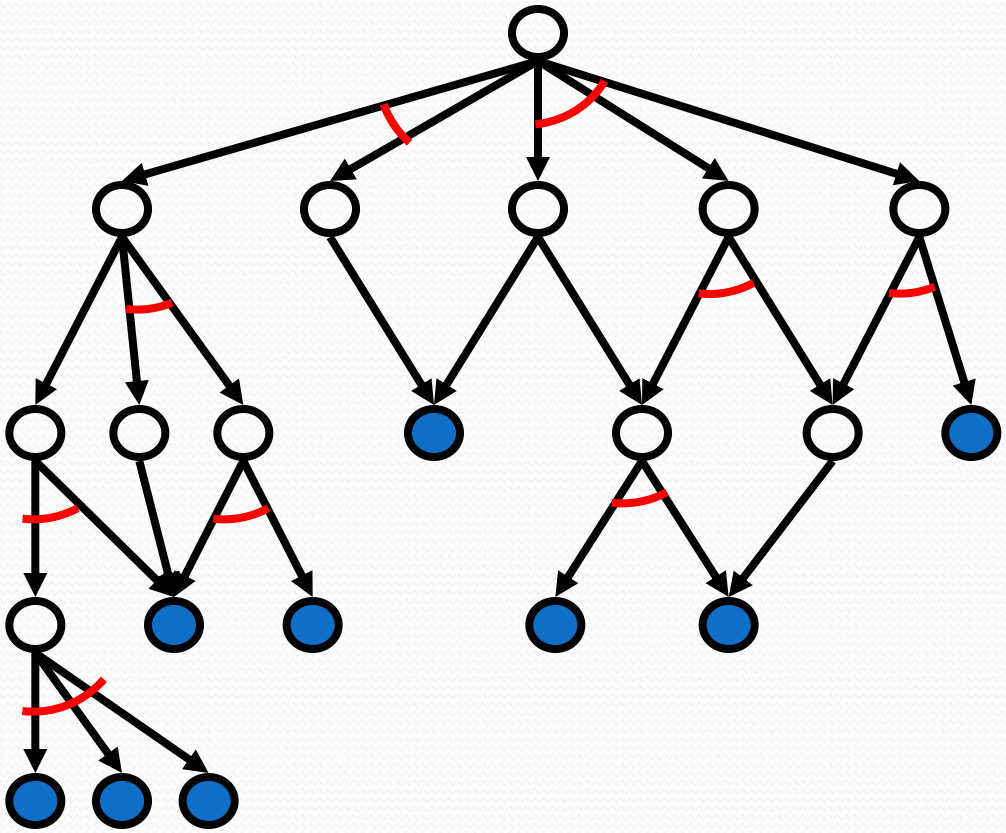


3. 解图

能导致初始结点可解的那些可解结点及有关连接弧所组成的子图称为该AND-OR 图的**解图**。

4. 解图的生成

- 自根节点开始选K-连接;
- 从该K-连接指向的每个子节点出发，再选一K-连接;
- 如此反复进行，直到所有K-连接都指向终节点为止.



3.3.3 与或图的启发式搜索

- AND-OR图的搜索与一般的OR 图搜索类似，但在代价的计算上有很大的差别。对OR图进行搜索，若搜索到某个结点n 时，不论 n是否生成了后继结点，n 的费用是由其本身状态决定的，但AND-OR图不同，其费用的计算依赖于n的后继结点的费用。
- 因为一个结点的后继结点代表了分解的子问题，子问题的难易程度决定了原问题求解的难易程度，所以不再考虑 n本身的难易程度。也就是说，在使用形如

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

的估价函数时，第一分量g(n)已经没有意义。而第二分量h(n)也不再是对最小路径代价的估计，而是对最小解图代价的估计。

- 在AND-OR图中，结点 n的代价可以按下面的方法计算：
  - (1) 如果 n是叶结点，则$h(n) = 0$；
  - (2) 若 n有一组由与弧连接的后继结点 {n1，n2，...nk}，则

    $$h(n) = c + h(n1) + h(n2) + ... + h(nk)$$

    其中c 为与弧的代价。
  - (3) 若n有一组由或弧连接的后继结点{n1，n2，...nk}，则

    $$h(n) = \min\{ c1 + h(n1) , c2 + h(n2) , ... , ck + h(nk)\}$$

    其中ci 表示第 i个或弧的代价。
  - (4) 若 n是既有与弧又有或弧连接的后继结点，则整个与弧作为一个或弧后继来考虑。

- **算法** 求AND-OR图 的解图的AO*算法

(1) [初始化]

- 建立一个搜索图 G’，使其仅仅包含初始结点I，如果 I为叶结点，则标记计算I为SOLVED 。 I的费用为 h(I)。

(2) **当 I没有被标记为 SOLVED或者 h(I)<FUTILITY时反复执行下述步骤：**

- i [计算局部解图]
  - 在搜索图G’ 中从初始结点 I开始，通过向下跟踪带有标记的连接弧，计算出G 的一个局部解图G”。
- ii [向下生长搜索图]
  - 选择局部解图 G”的任意一个非叶结点n ，扩展 n，生成它的全部后继结点，并对 n的每个后继结点ni ，将其加入搜索图 G’的同时计算ni 的费用 h(ni)。当 ni是叶结点时，标记 ni为SOLVED 。

◆ iii [向上修正结点的费用、连接弧标记和可解结点标记]

➢ 令集合S={n}。当S非空时反复执行下述步骤

✓ (a) 从S中删除节点m，满足m在G中的后裔不出现在S中。

✓ (b)按以下步骤修改m的费用h(m)：

对于每一从m出发的指向节点集合{n_{1i}，…，n_{ki}}的连接符，计算$h_j(m)=c_i+h(n_{1i})+...+h(n_{ki})$，h(m):＝min {$h_j(m)$},标记实现此最小值的连接弧，如果以前的标记与此不同，则删除以前的标记；如果这个连接符指向的所有后继节点都标记了SOLVED,则将m标记为SOLVED。

✓ (c) 如果m已被标记为SOLVED，或者h(m)的修正费用不同于它的前一次计算结果，则将m的所有通过有标记的连接弧将m作为后继结点之一的父结点添加到S中。

- (3) [算法结束]
- 在搜索图G'中从初始结点I出发通过向下跟踪带有标记的连接弧便可以得到G的解图。

- (1) 在上述 AO*算法的第 ii 步中，我们没有讨论结点n 是没有后裔的非叶结点的情形。实际上，这时 n是不可解结点，对不可解结点的标记及不可解性的向上传递的处理方法是比较容易的，请自行加以考虑。
- (2) 在上述AO*算法的第 ii 步中，如果有多个可以被用于扩展的结点n ，那么首先扩展其中的哪个结点是值得讨论的。

✓一般地，如果估计最终将改变到某个更接近于最佳解图的情况，那么 算法越早进行这种改变为好。在通常情况下，扩展具有最高h值的那个结点，最有可能产生一个变化的估计。

- (3) 如果对所有结点n 均满足单调限制 $h(n) \le h^*(n)$，则 AO*算法是可采纳的，也就是说，如果从初始结点到一组终结点存在一个解图，那么AO*算法算法最终总能得到一个最佳的解图。

第3.4节 博弈空间搜索

- **3.4.1** 极大极小过程
- **3.4.2** α－β 过程

- **博弈**问题是人工智能研究的一个重要问题。以下仅考虑**非偶然性全信息零和对弈**：

➢(1) 博弈的双方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。

➢(2) 在博弈过程中，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，理智地决定自己的行动。

3.4.1 极大极小过程

- 以下假定博弈双方为MAX和MIN，MAX先行。
- 在博弈的每一步，可供他们选择的方案都有很多种。
- 如果从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；
  - 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况出现

- 把双人博弈过程用图的形式表示出来，就可以得到一棵AND-OR树，这种AND-OR树称为**博弈树**。
- 在博弈树中，那些下一步该MAX走的结点称为**MAX结点**，而下一步该MIN走的结点称为 **MIN结点**。
- 博弈树特点主要如下：

➢(1) 博弈的初始状态是初始结点；

➢(2) 博弈树的“与”结点和“或”结点是逐层交替出现的；

➢(3) 整个博弈过程始终站在某一方的立场上，所以能使自己一方获胜的终局都是本原问题，相应的结点也是可解结点，所有使对方获胜的结点都是不可解结点。

- 在人工智能中可以采用搜索方法来求解博弈问题，这个方法称为**极大极小（MINMAX）过程**。
- 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，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：
  - (1) 有利于MAX方的态势： f(p)取正值；
  - (2) 有利于MIN方的态势： f(p) 取负值；
  - (3) 态势均衡的时候： f(p) 取零，

  其中p 代表棋局。

- MINMAX过程的基本思想是：
  - (1) 当轮到MIN走步的结点时，MAX应考虑最坏的情况（即应使f(p) 取极小值）。
  - (2) 当轮到MAX走步的结点时，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。

- 以下用一字棋说明极大极小过程，设只考虑两层，即每方只走一步。在生成后继结点时，利用棋盘的对称性，我们省略了从对称上看是相同的格局。
- 格局 p估价函数 e(p)的定义如下：

➢(1) 若格局p 对任何一方都不是获胜的，则

e(p) = (所有空格都放上MAX的棋子后三子成一线的总数)

－（所有空格都放上MIN的棋子后三子成一线的总数）

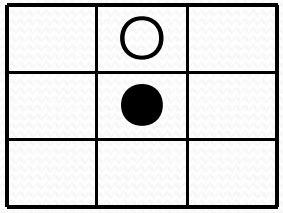
➢(2) 若 p是MAX获胜，则

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

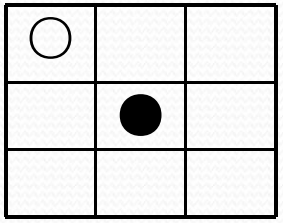
➢(3) 若 p是MIN获胜，则

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

- 若p 为下图所示,其中●表示MAX 方，○表示MIN方。

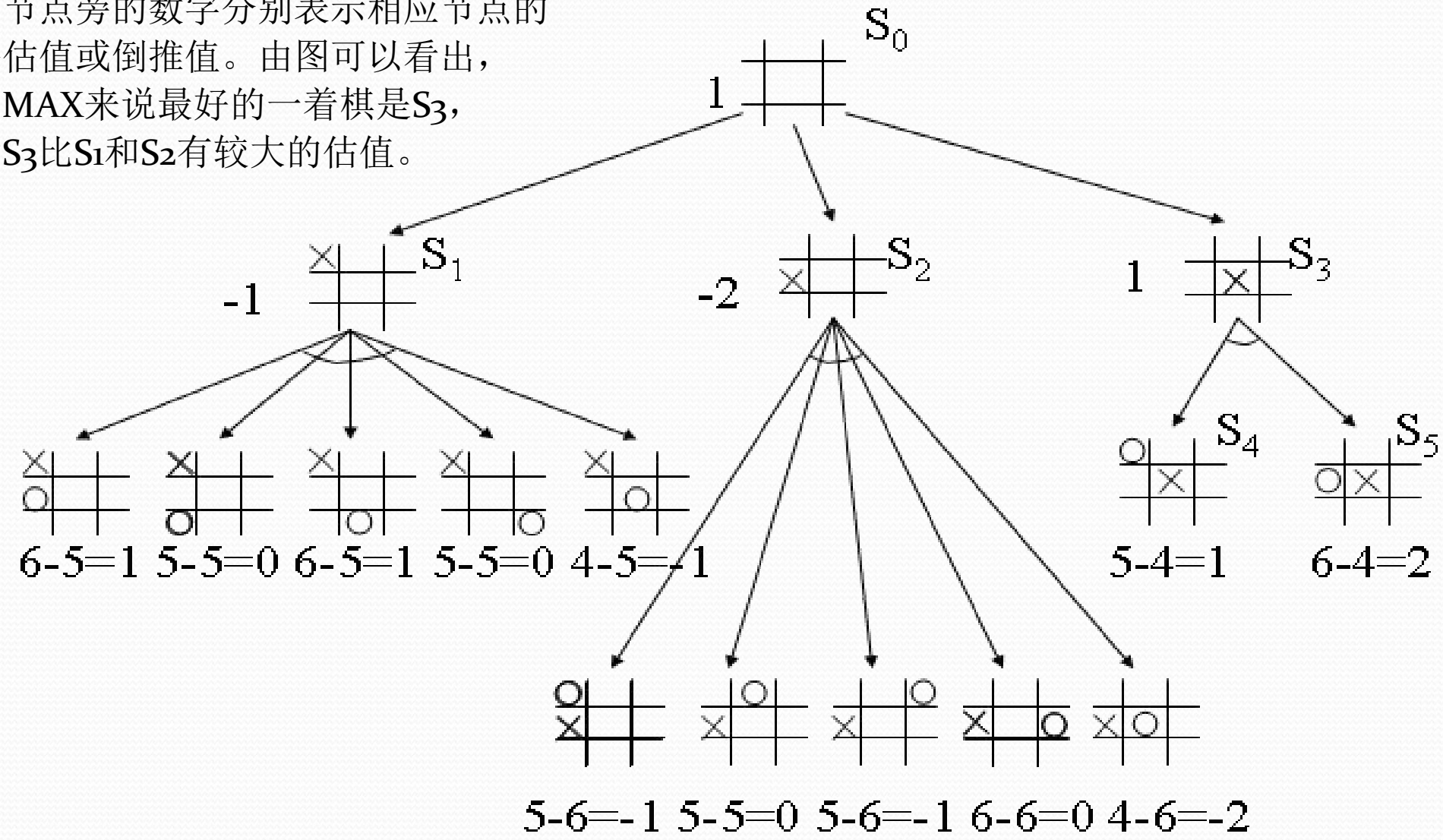


或



- 则
- e(p)=6-4=2
- e(p)=5-4=1

假设由MAX先走棋，且我们站在MAX立场上。
下图给出了MAX的第一着走棋生成的博弈树。
图中节点旁的数字分别表示相应节点的
静态估值或倒推值。由图可以看出，
对于MAX来说最好的一着棋是S_3，
因为S_3比S_1和S_2有较大的估值。



3.4.2 α－β 过程

- 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结点，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结点，以减少搜索的次数，这就是下面要讨论的 α－β 过程。
- α－β 过程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。

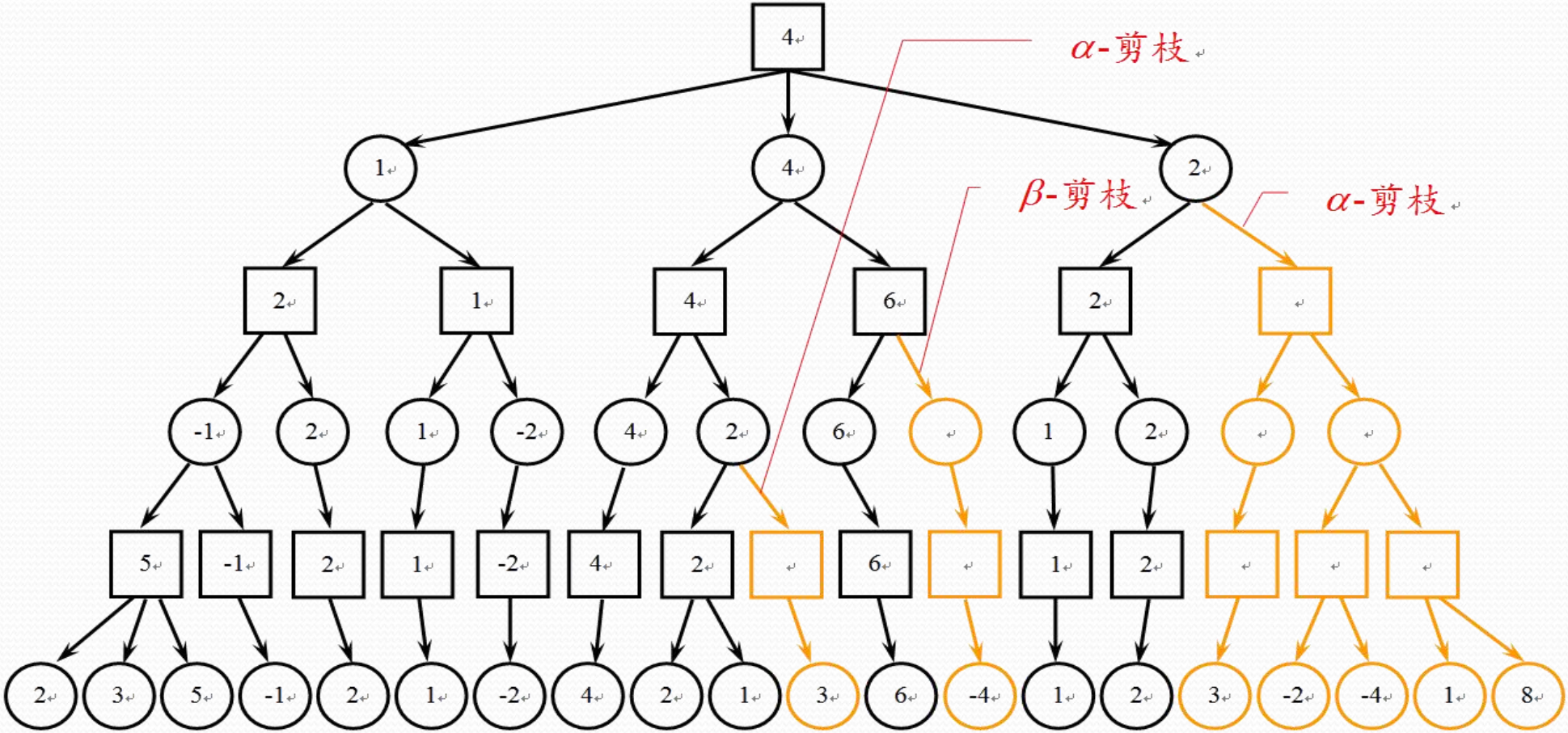
- 在 α－β 过程中，一个结点的倒推值是动态进行计算的。
  - 首先使搜索树的某一部分达到最大深度，计算出某些 MAX 结点的暂时的 α 值，或者是某些MIN结点的暂时的 β 值；
  - 随着搜索的继续，不断修正个别结点的 α 或 β 值。
  - 当初始结点的倒推值确定以后，终止整个 α－β 过程。
- 注意， MAX结点的 α 值和MIN 结点的 β 值的修改有如下规律：**MAX结点的 α 值永不下降；MIN结点的 β 值永不增加。**

- 剪枝的规则：

➢(1) 若任何MIN结点的 β 值小于或等于任何它的先辈MAX结点（不一定是父结点）的 α 值，则可终止该MIN结点以下的搜索，然后这个MIN结点的最终倒推值即为它已得到的 β 值。

➢(2) 若任何 MAX结点的 α 值大于或等于它的 MIN先辈结点（不一定是父结点）的 β 值，则可以终止该 MAX 结点以下的搜索，然后这个MAX结点处的倒推值即为它已得到的 α 值。

- 当满足规则(1)而减少了搜索时，进行了 α - 剪枝；而当满足规则(2)而减少了搜索时，进行了 β - 剪枝。


α-剪枝
β-剪枝
α-剪枝

- 在发生了剪枝的情况下，一个 MAX结点的 α 值或者一个MIN结点的 β 值未必与真正的极大极小值的搜索结果的倒推值相同
- 初始结点的倒推值必然是相同的，使用它选择的走步也是相同的。
- 动态地计算并保存 α 和 β 值，并且一旦可能就进行剪枝的过程称为 **α－β 过程**，当初始结点的全体后继结点的最终倒推值全部给出时，上述过程便结束。
- 在搜索深度相同的条件下，采用这个过程所获得的走步总跟简单的极大极小过程的结果是相同的，区别只在于 α－β 过程通常只用少得多的搜索便可以找到一个理想的走步。