第三章 问题归约知识表示及其搜索技术

- 问题归约法及与或图
- 与或树的盲目式搜索
- 博弈与博弈树搜索

1

问题归约法

- 已知问题的描述，通过一系列变换把此问题最终变为一个子问题集合；这些子问题的解可以直接得到，从而解决了初始问题。
- 该方法也就是从目标(要解决的问题)出发逆向推理，建立子问题以及子问题的子问题，直至最后把初始问题归约为一个平凡的本原问题集合。这就是问题归约的实质。

2

问题归约法的组成部分

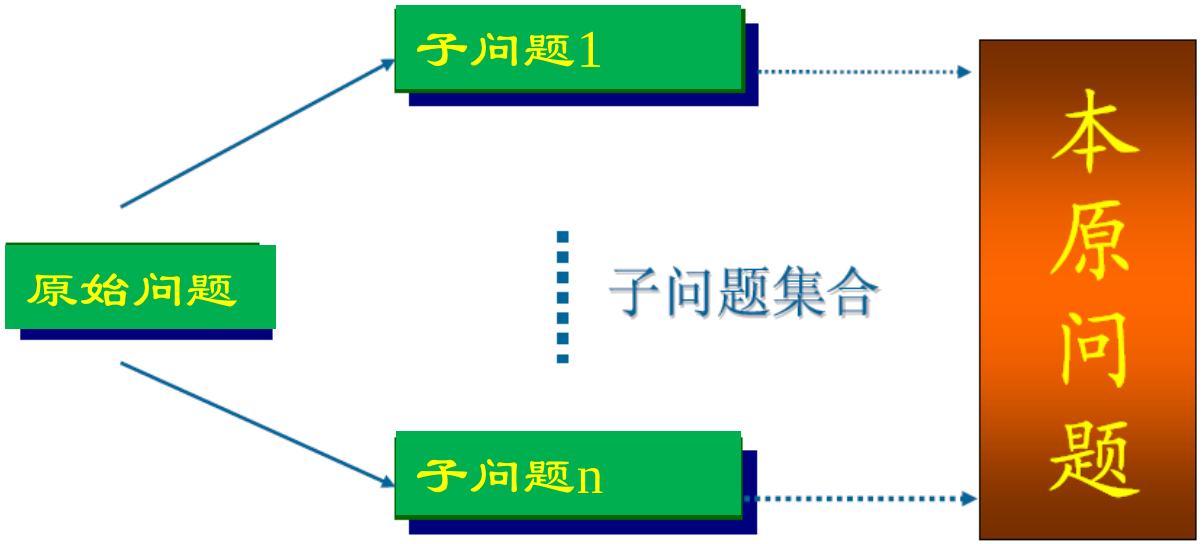
- (1) 一个初始问题描述；
- (2) 一套把问题变换为子问题的操作符；
- (3) 一套本原问题描述。

3

问题规约法图解

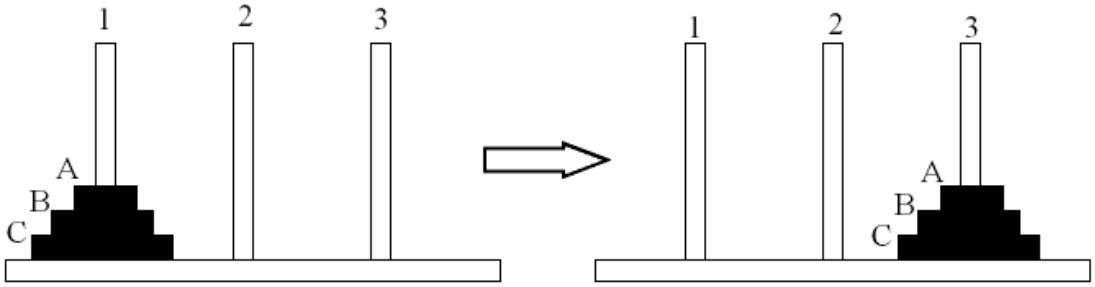
问题归约法

Problem Reduction Representation



4

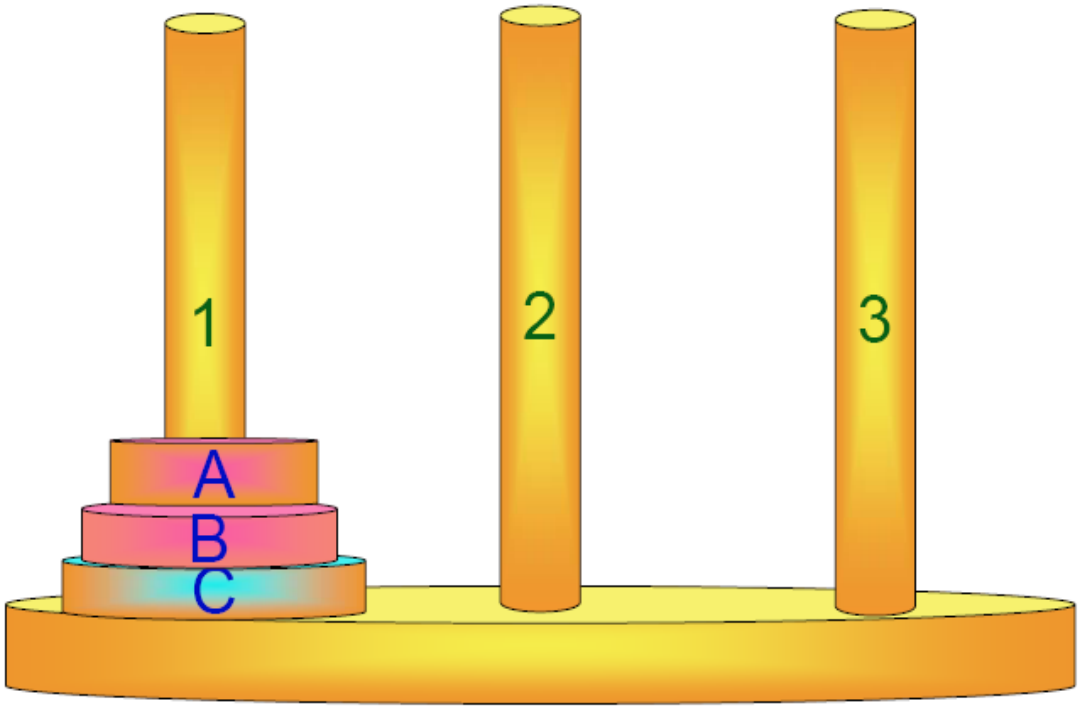
梵塔难题



- 有3个柱子(1，2，3)和3个不同尺寸的圆盘(A，B，C)。在每个圆盘的中心有个孔，所以圆盘可以堆叠在柱子上。最初，全部3个圆盘都堆在柱子1上：最大的圆盘C在底部，最小的圆盘A在顶部。要求把所有圆盘都移到柱子3上，每次只许移动一个，而且只能先搬动柱子顶部的圆盘，还不许把尺寸较大的圆盘堆放在尺寸较小的圆盘上。这个问题的初始配置和目标配置如图所示。

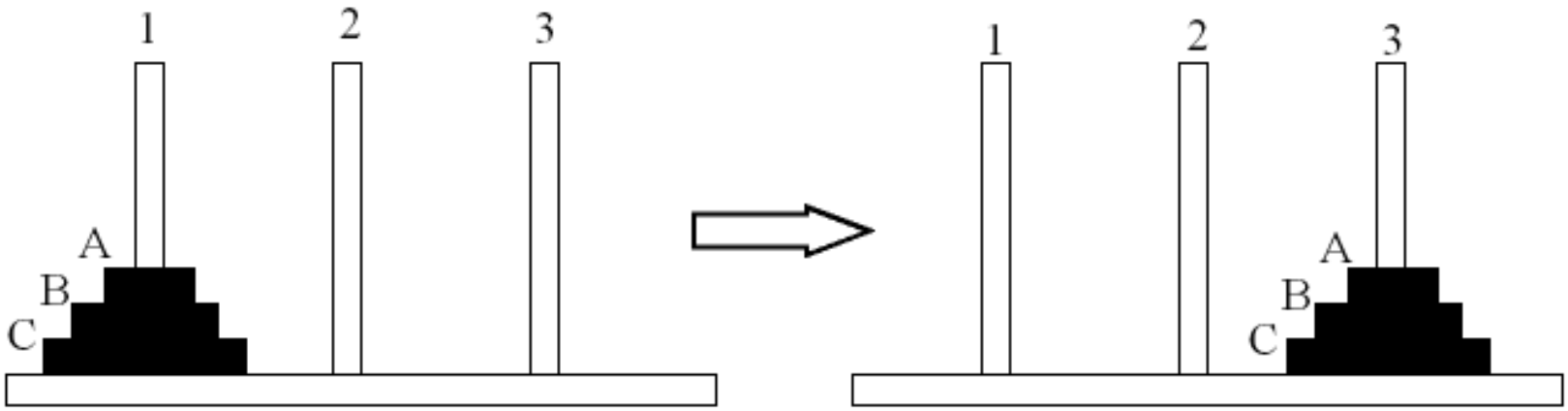
5

- 梵塔难题



6

梵塔难题



(a) 初始状态

(b) 目标状态

7

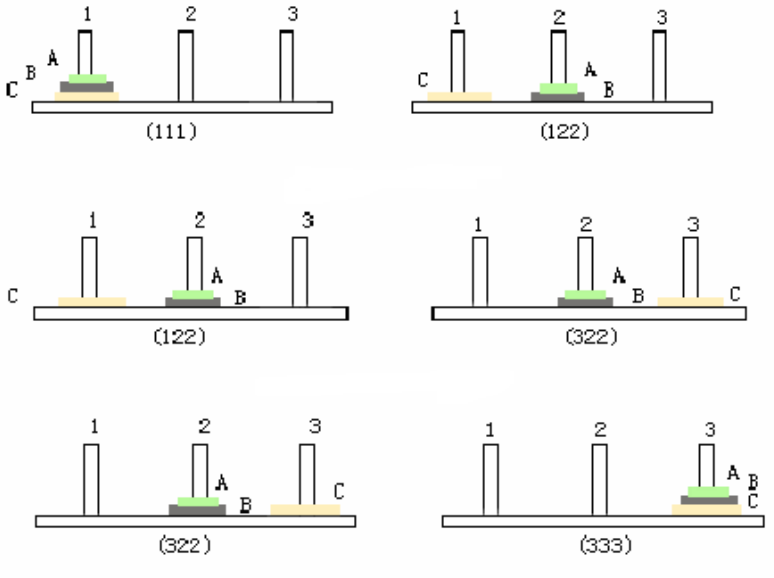
分析

- 原始问题归约（简化）为三个子问题

  1、移动A，B盘至柱子2的双圆盘难题

  2、移动圆盘C至柱子3的单圆盘问题

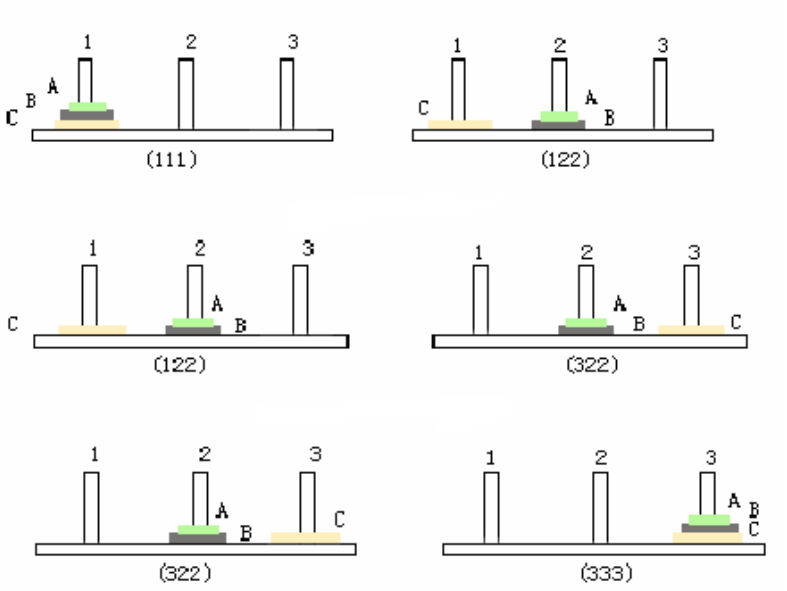
  3、移动A，B盘至柱子3的双圆盘难题



8

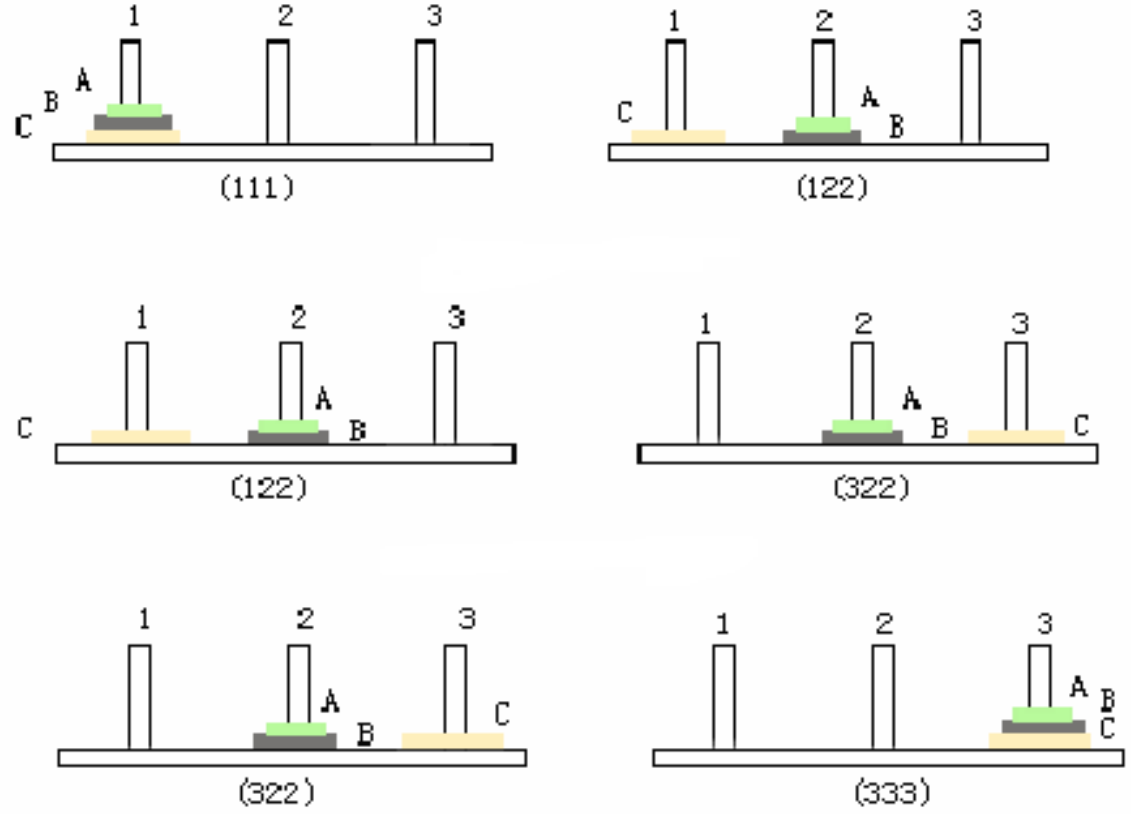
分析

- 对于梵塔问题，子问题[(111)→(122)]，[(122)→(322)]以及[(322)→(333)]规定了最后解答路径将要通过的脚踏石状态(122)和(322)。



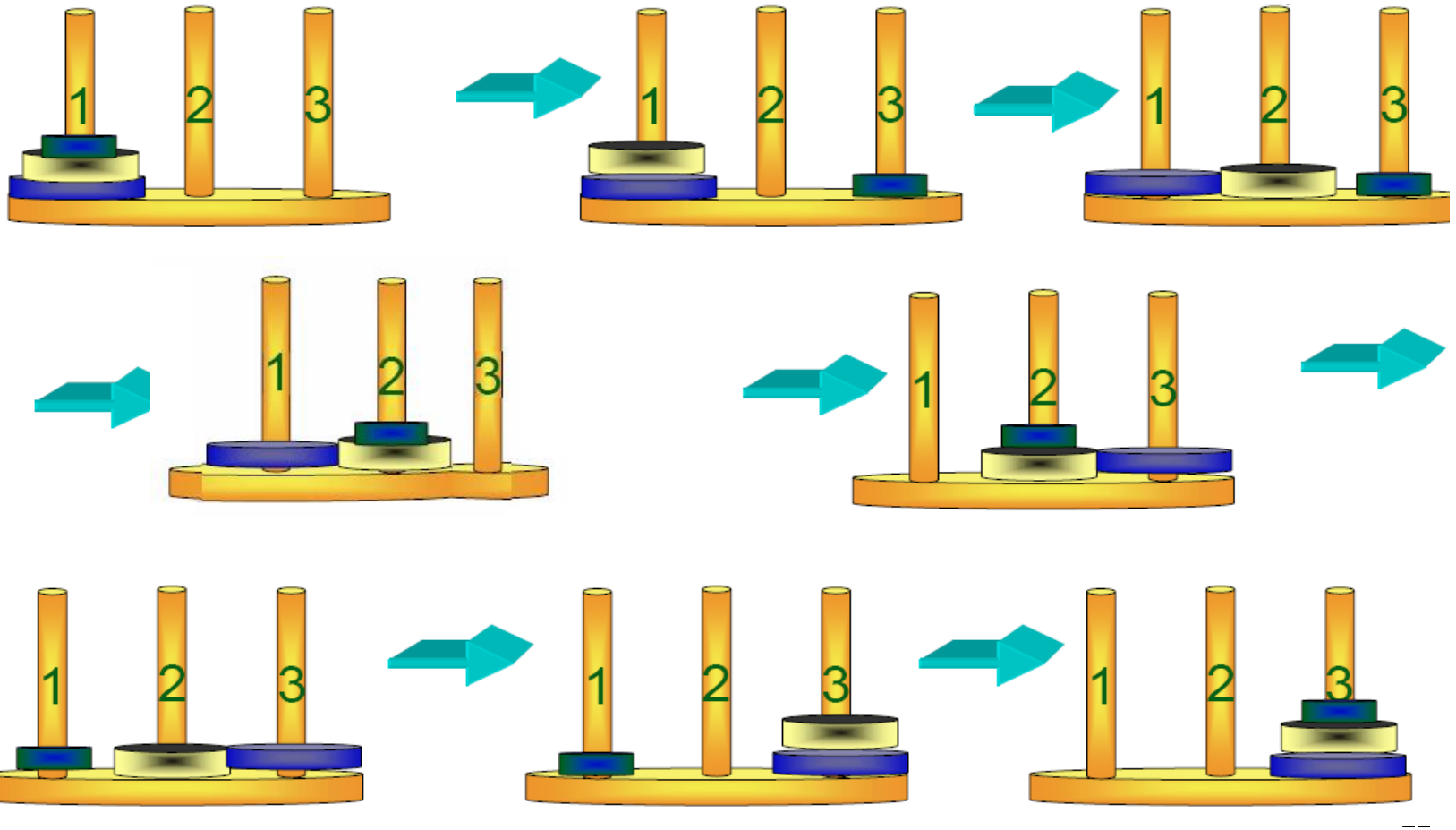
9

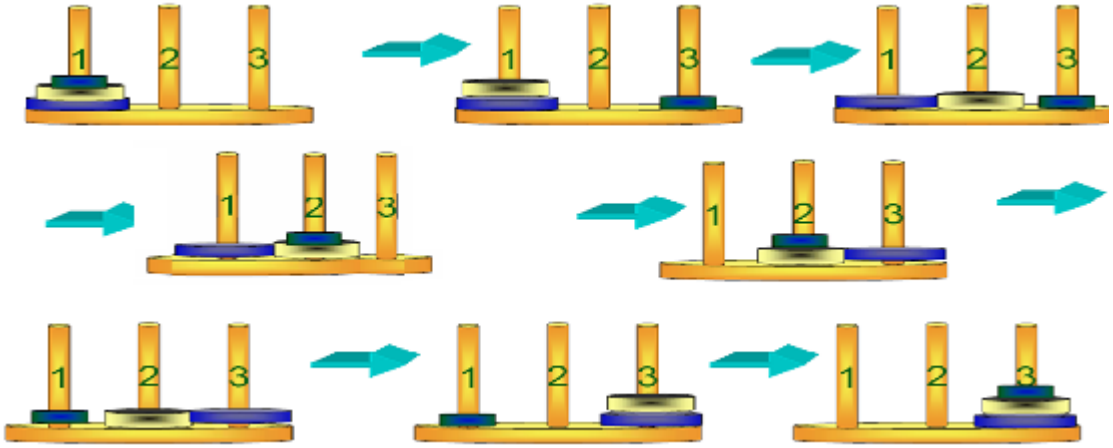
分析



10

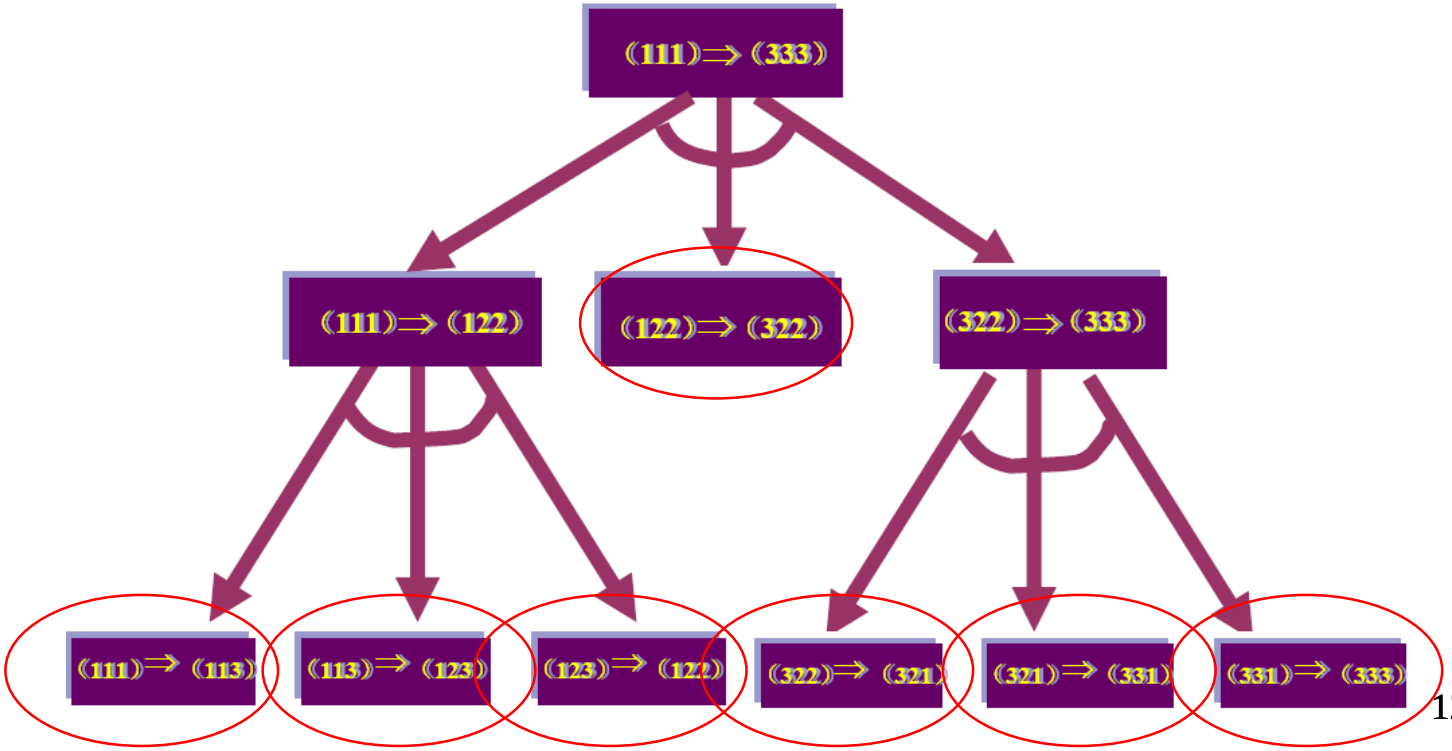
具体解题过程





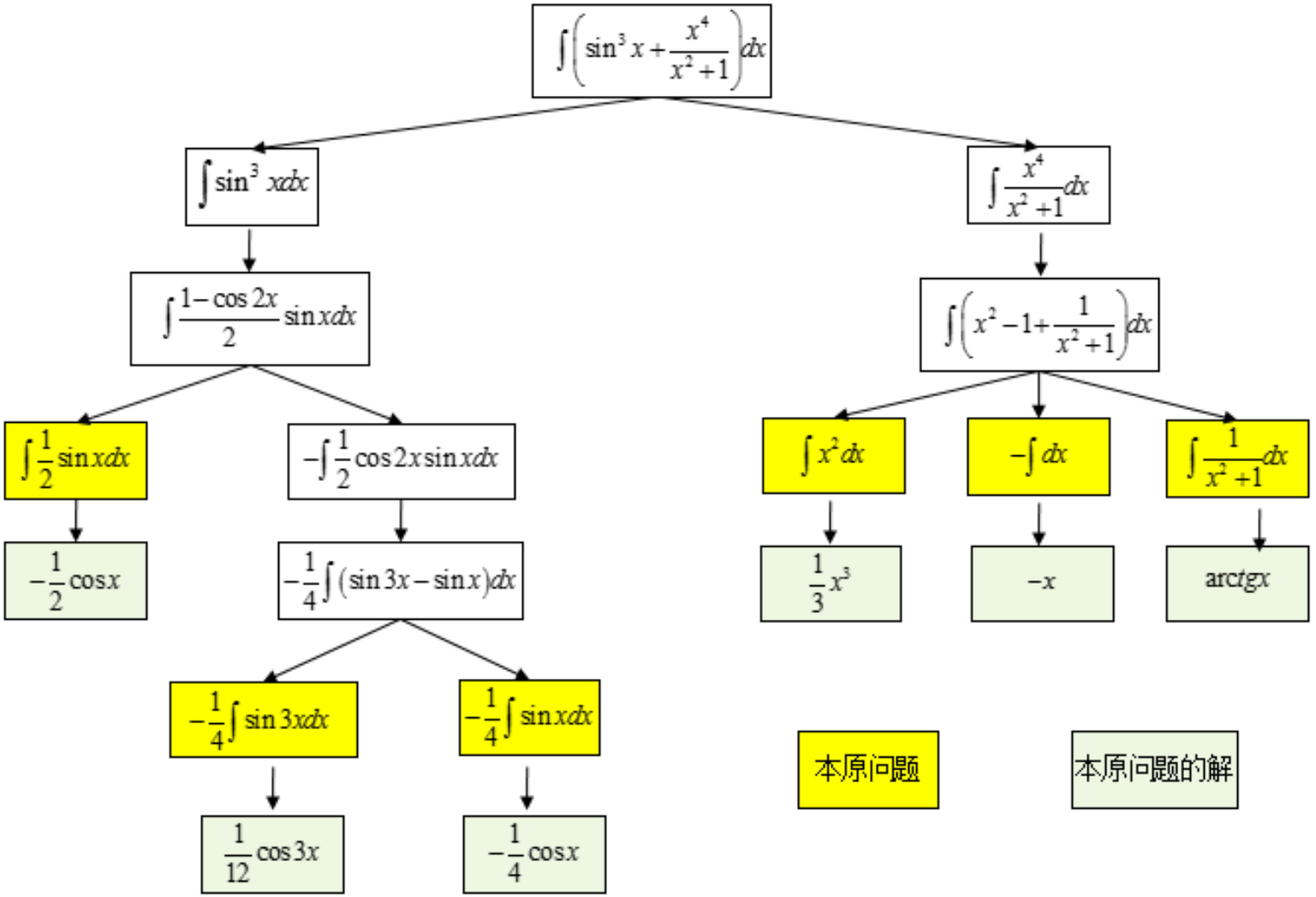
n=1?
n=2?
n=3?
……

梵塔问题归约图



12

不定积分的归约过程


$\int\left(\sin^3 x+\frac{x^4}{x^2+1}\right)dx$
$\int\sin^3 x dx$
$\int\frac{x^4}{x^2+1}dx$
$\int\frac{1-\cos 2x}{2}\sin x dx$
$\int\left(x^2-1+\frac{1}{x^2+1}\right)dx$
$\int\frac{1}{2}\sin x dx$
$-\int\frac{1}{2}\cos 2x\sin x dx$
$\int x^2 dx$
$-\int dx$
$\int\frac{1}{x^2+1}dx$
$-\frac{1}{2}\cos x$
$-\frac{1}{4}\int(\sin 3x-\sin x)dx$
$\frac{1}{3}x^3$
$-x$
arctgx
$-\frac{1}{4}\int\sin 3x dx$
$-\frac{1}{4}\int\sin x dx$
本原问题
本原问题的解
$\frac{1}{12}\cos 3x$
$-\frac{1}{4}\cos x$

由该问题归约的与或图，可读出该不定积分的解为：

$$\int\left(\sin^3 x+\frac{x^4}{x^2+1}\right)dx=-\frac{1}{2}\cos x+\frac{1}{12}\cos 3x-\frac{1}{4}\cos x+\frac{1}{3}x^3-x+\operatorname{arctg}x$$

$$=-\frac{3}{4}\cos x+\frac{1}{12}\cos 3x+\frac{1}{3}x^3-x+\operatorname{arctg}x$$

与或图表示

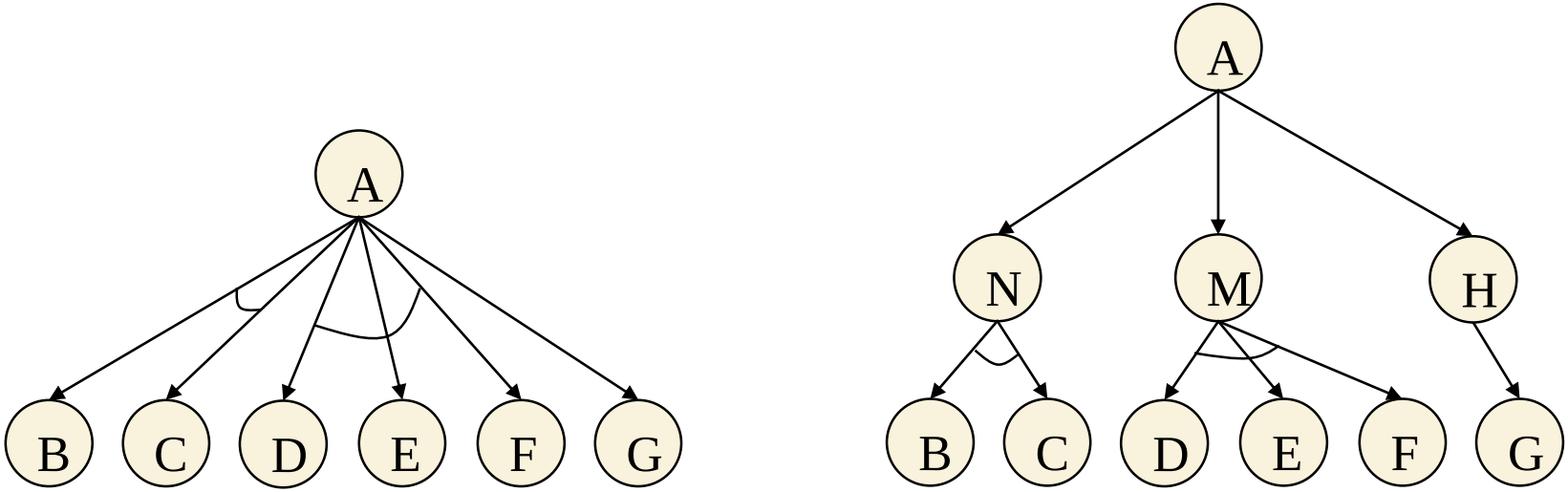
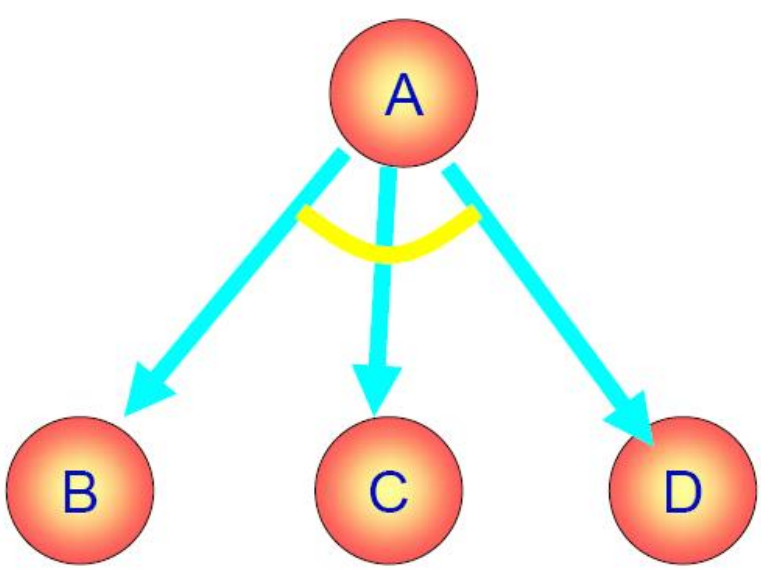


图 2.14 与或图

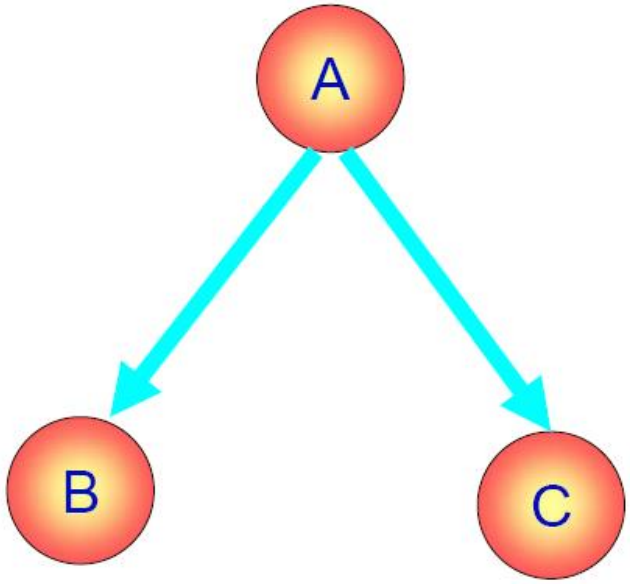
15

与或图表示

- 与图、或图、与或图



与图

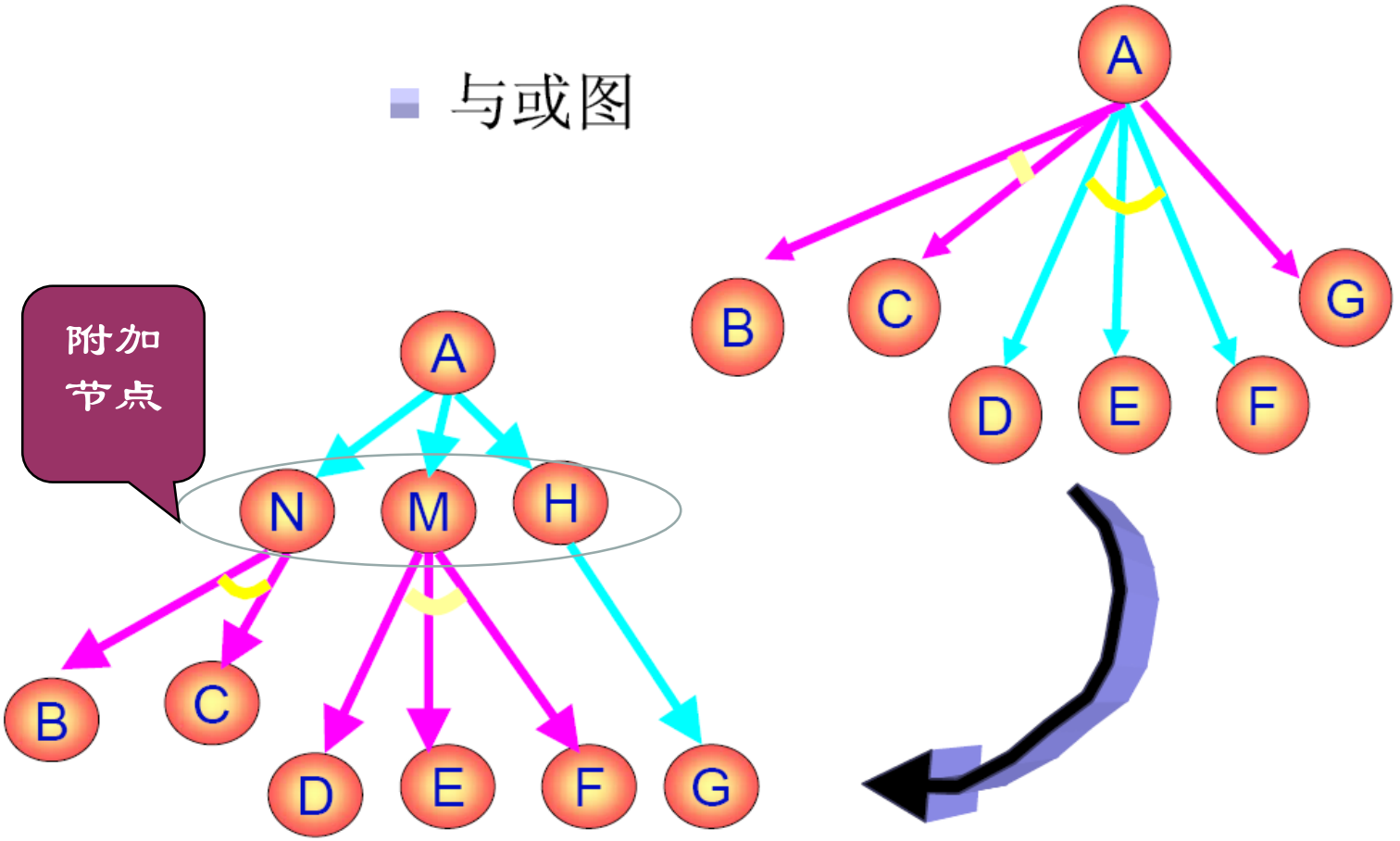


或图

16

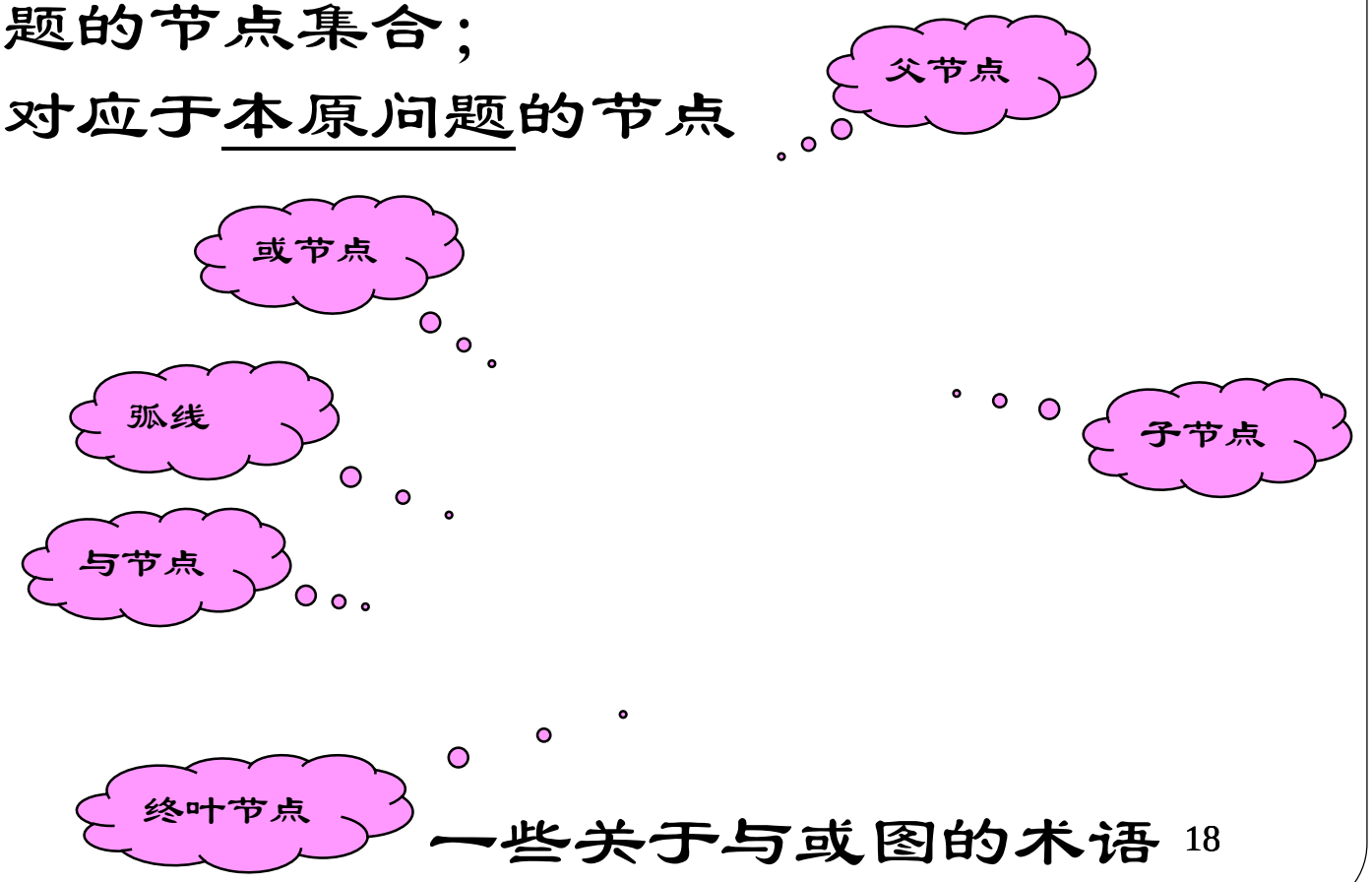
与或图表示

- 与或图



17

- 如果某条弧线从节点a指向节点b，那么节点a叫做节点b的**父辈节点**；节点b叫做节点a的**后继节点**或**后裔**；
- **或节点**，只要解决某个问题就可解决其父辈问题的节点集合；
- **与节点**，只有解决所有子问题，才能解决其父辈问题的节点集合；
- **终叶节点**，是对应于本原问题的节点



一些关于与或图的术语

18

- **可解节点**：与或图中一个可解节点的一般定义可以归纳如下：

1、 终叶节点是可解节点(因为它们与本原问题相关连)。

2、如果某个非终叶节点含有**或后继节点**，那么只有当其后继节点至少有一个是可解的时，此非终叶节点才是可解的。

3、如果某个非终叶节点含有**与后继节点**，那么只要当其后继节点全部为可解时，此非终叶节点才是可解的。

19

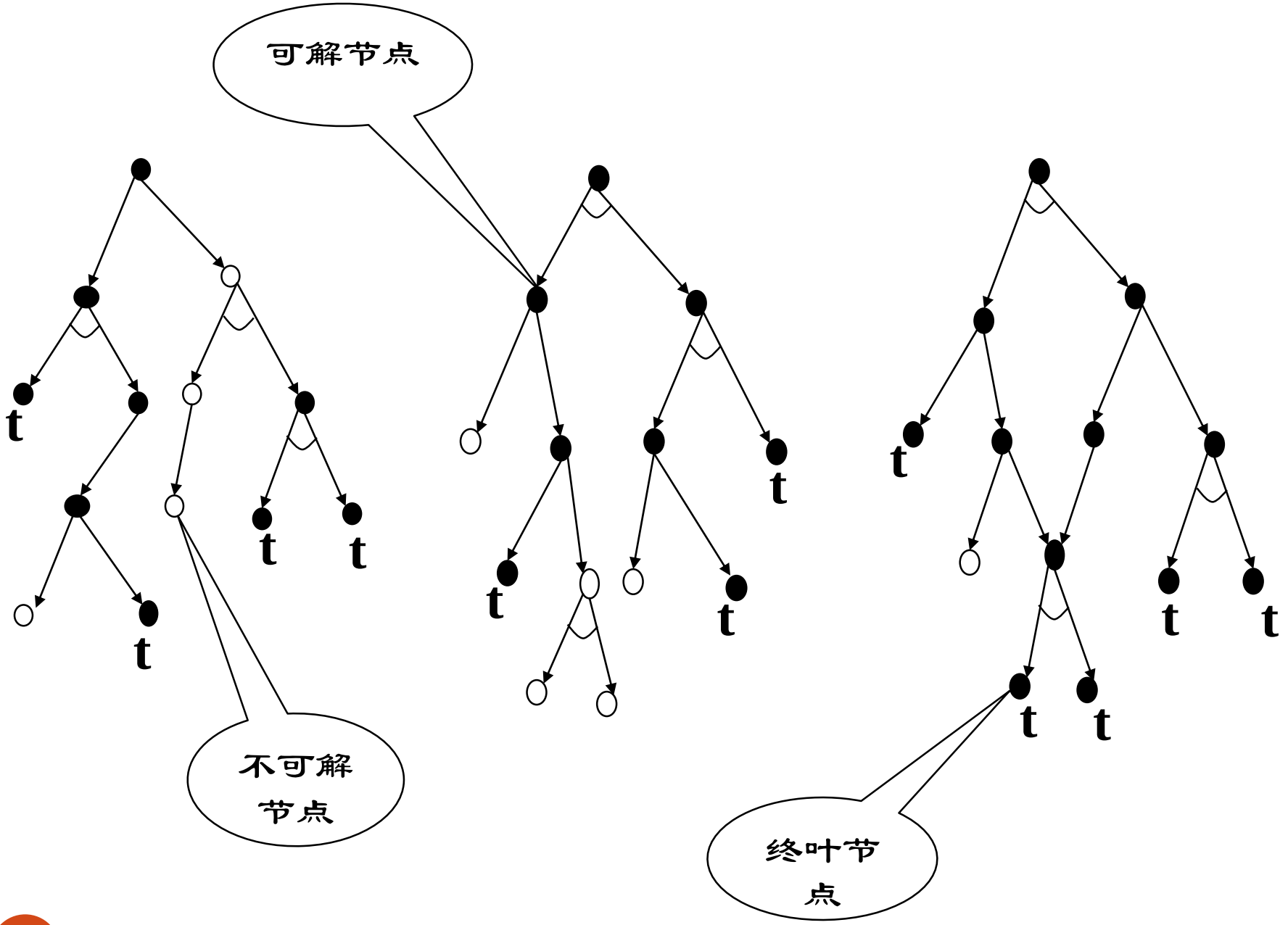
- **不可解节点** 不可解节点的一般定义归纳于下：

1、没有后裔的非终叶节点为不可解节点。

2、如果某个非终叶节点含有**或后继节点**，那么只有当其全部后裔为不可解时，此非终叶节点才是不可解的。

3、如果某个非终叶节点含有**与后继节点**，那么只要当其后裔至少有一个为不可解时，此非终叶节点才是不可解的。

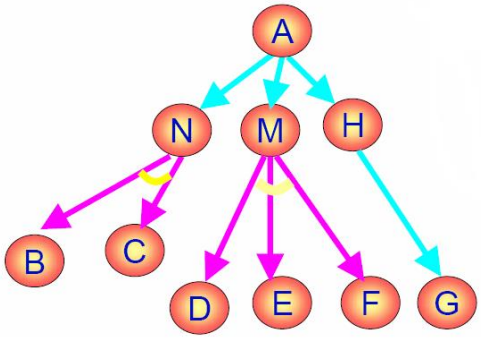
20

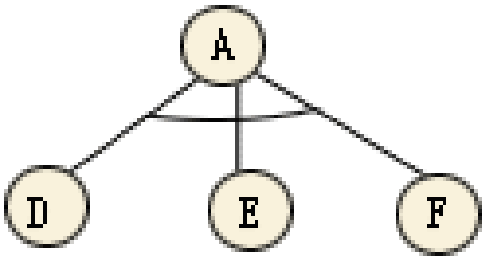


21

与或图构成规则

(1) 与或图中的每个节点代表一个要解决的单一问题或问题集合。图中所含起始节点对应于原始问题。(2) 对应于本原问题的节点，叫做终叶节点，它没有后裔。(3) 对于把算符应用于问题A的每种可能情况，都把问题变换为一个子问题集合；有向弧线自A指向后继节点，表示所求得的子问题集合。（或节点）(4) 一般对于代表两个或两个以上子问题集合的每个节点，有向弧线从此节点指向此子问题集合中的各个节点。由于只有当集合中所有的项都有解时，这个子问题的集合才能获得解答，所以这些子问题节点叫做与节点。(5) 在特殊情况下，当只有一个算符可应用于问题A，而且这个算符产生具有一个以上子问题的某个集合时，由上述规则3和规则4所产生的图可以得到简化。因此，代表子问题集合的中间或节点可以被略去。





22

作业-2

- 试用四元数列结构表示四圆盘梵塔问题，并画出求解该问题的与或图。

23

问题归约搜索推理技术

- 主要内容

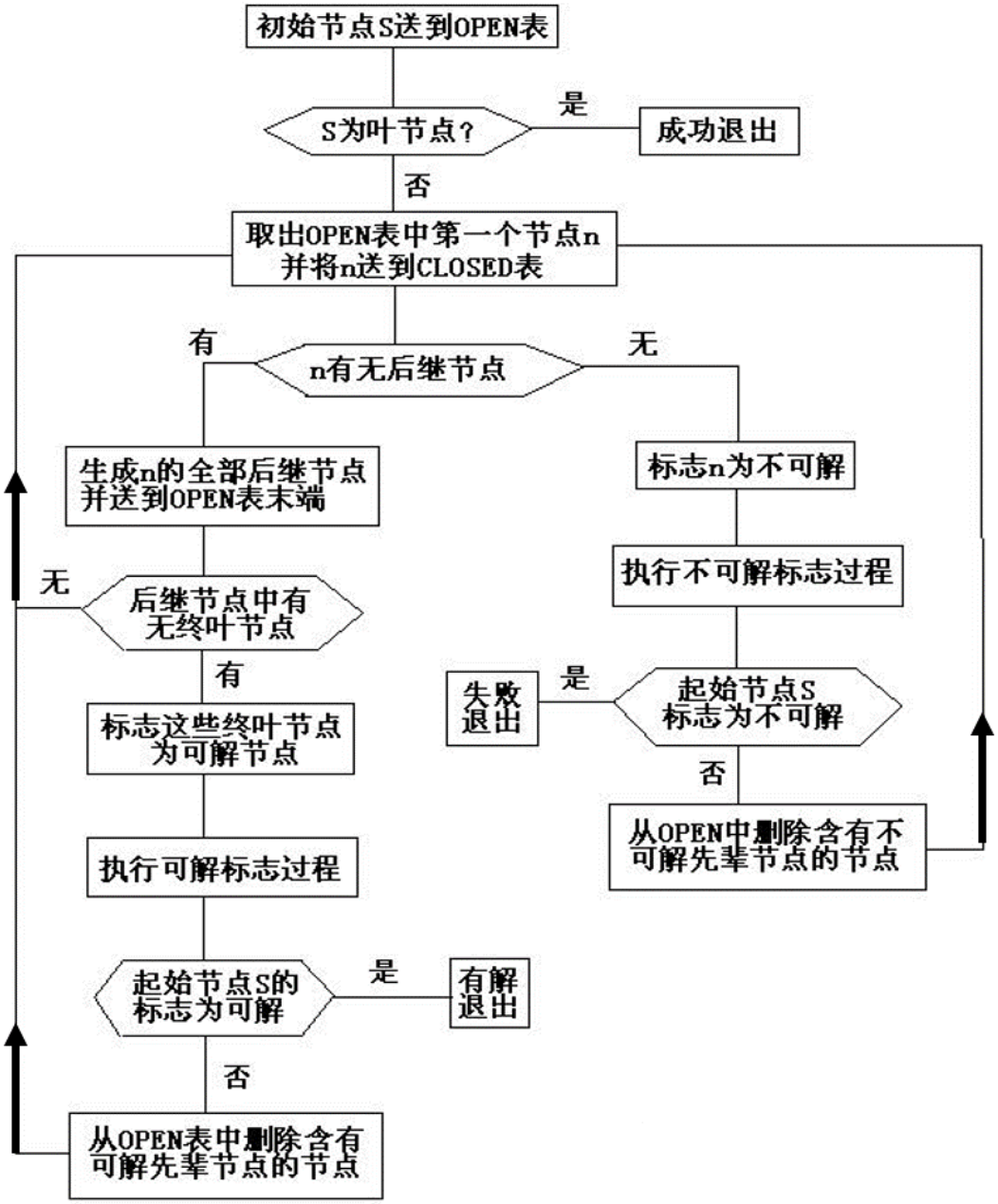
- 与/或树搜索
- 机器博弈

} 问题归约法

与/或图搜索

- 与/或图的搜索策略
  - 盲目式搜索
    - 与/或图的一般搜索
    - 宽度优先搜索
    - 深度优先搜索
  - 启发式搜索
    - 希望树
    - 机器博弈
      - Max-Min搜索
      - α–β剪枝

与或树的宽度优先搜索


初始节点S送到OPEN表
S为叶节点？
是
成功退出
否
取出OPEN表中第一个节点n
并将n送到CLOSED表
有
n有无后继节点
无
生成n的全部后继节点
并送到OPEN表末端
标志n为不可解
执行不可解标志过程
无
后继节点中有
无终叶节点
有
失败
退出
是
起始节点S
标志为不可解
否
标志这些终叶节点
为可解节点
从OPEN中删除含有不
可解先辈节点的节点
执行可解标志过程
起始节点S的
标志为可解
是
有解
退出
否
从OPEN表中删除含有
可解先辈节点的节点

宽度优先搜索算法流程

(1) 起始节点S送OPEN表

(2) 若S为叶节点，则成功结束，否则，继续

(3) 取出OPEN表的第一个节点（记作n），并送到CLOSED表

(4) 扩展节点n，生成其全部后继节点，送OPEN表末端，并设置指向n的指针
说明：此时可能出现三种情况
- 节点 n 无后继节点
- 节点 n 有后继节点、并有叶节点
- 节点 n 有后继节点、但无叶节点

(5)
- 若 n 无后继节点，标志 n 为不可解，并转9（10、11）；
- 若后继节点中有叶节点，则标志这些叶节点为可解节点，并继续6、7、8
- 否则转第3步

(6) 实行可解标志过程

(7) 若起始节点S标志为可解，则找到解而结束，否则继续

(8) 从OPEN表中删去含有可解先辈节点的节点，并转第3步

(9) 实行不可解标志过程

(10) 若起始节点S标志为不可解，则失败而结束，否则继续

(11) 从OPEN表中删去含有不可解先辈节点的节点

(12) 转第3步

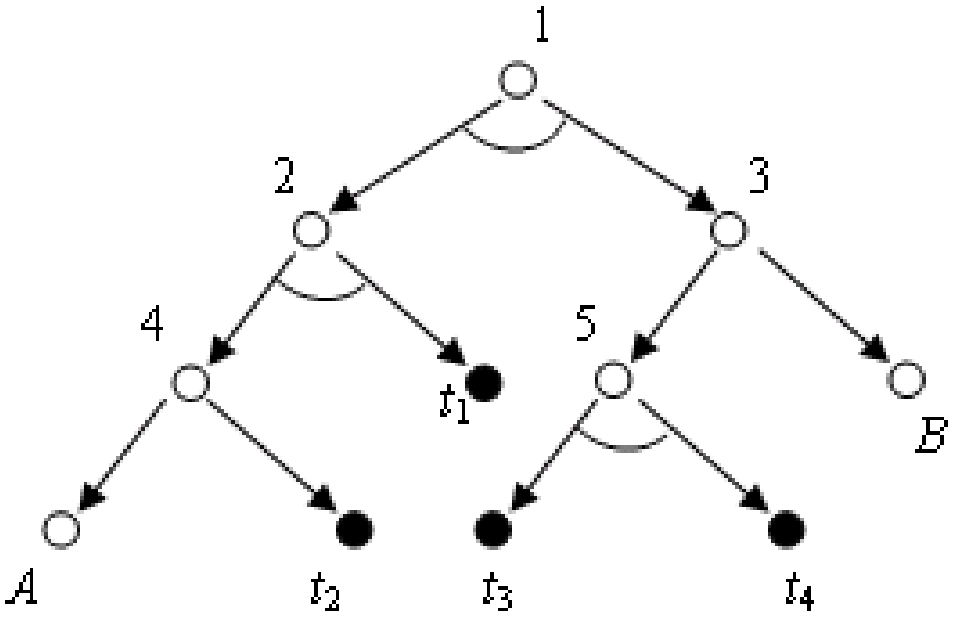
算法结束的条件

- 若初始节点被标示为可解节点，算法成功结束（有解）
- 若初始节点被标示为不可解结点，则搜索失败结束（无解）

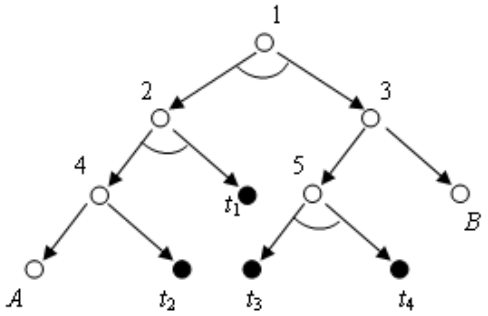
与/或树的宽度优先搜索

例1：设有如图所示的与/或树，其中t_1, t_2, t_3, t_4均为终叶节点，A和B是不可解的端节点。

用与/或树的宽度优先搜索法对该图进行搜索。



例：与/或树的宽度优先搜索



初始化：

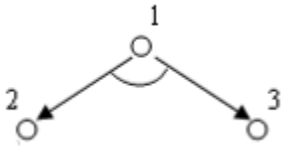
节点1送open表，且不为叶节点

open	close
1	[]

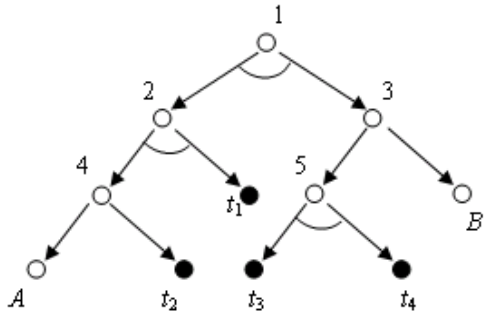
STEP1：

- 扩展节点1，得到节点2、3；
- 节点2，3都不是终叶节点，接着扩展节点2，此时open表只剩节点3.

open	close
1	
2,3	1
3	1,2



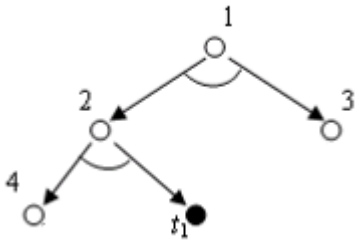
例：与/或树的宽度优先搜索



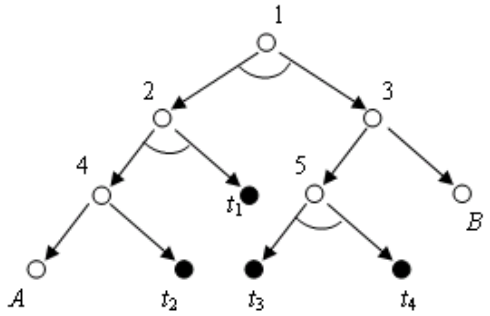
STEP2：

- 扩展节点2后，得到节点4、t1，此时open表中的节点有3，4，t1；
- 节点t1是终叶节点且为可解节点，对其先辈节点进行标示；
- t1的父节点是“与”节点，无法判断节点2是否可解，接着扩展节点3.

open	close
1	
2,3	1
3,4,t1	1,2
4,t1	1,2,3



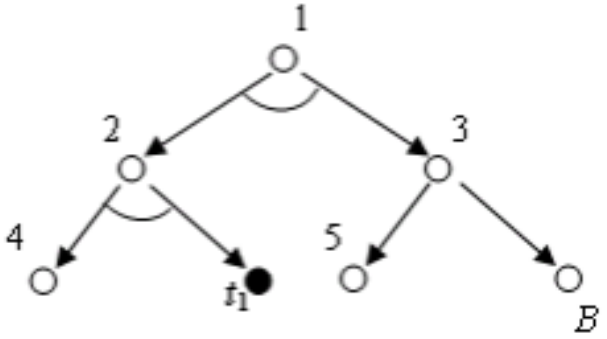
例：与/或树的宽度优先搜索



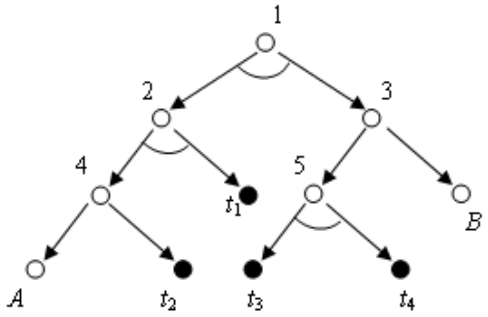
STEP3：

- 扩展节点3后，得到节点5、B；
- 节点B都是不可解节点，执行反标注过程，没有先辈节点被标为不可解节点，接着扩展节点4.

open	close
1	
2,3	1
3,4,t1	1,2
4,t1,5,B	1,2,3
t1,5,B	1,2,3,4



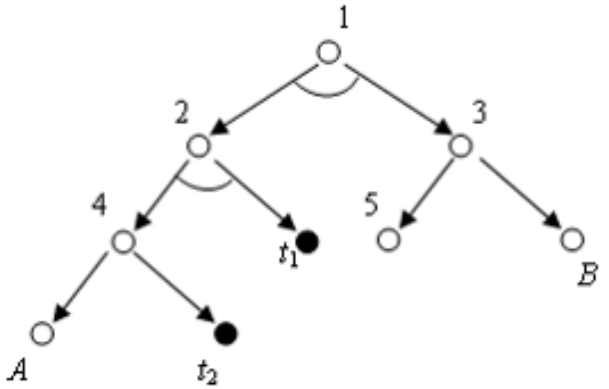
例：与/或树的宽度优先搜索



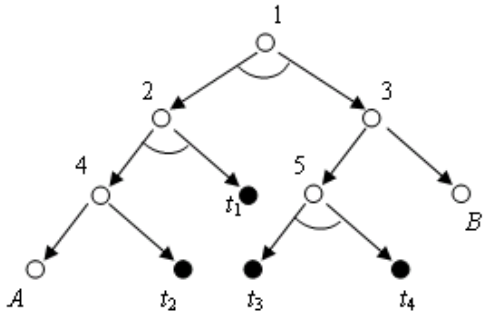
STEP4:

- 扩展节点4后，得到节点A、t2；
- 节点t2是终叶节点且为可解节点，对其先辈节点进行标示；
- t2的父节点是“或”节点，推出节点4可解，推出节点2可解，但不能确定1是否可解；
- 此时节点5是open表中第一个待考察的节点，下一步将扩展节点5.

open	close
1	
2,3	1
3,4,t1	1,2
4,t1,5,B	1,2,3
t1,5,B,**A**,**t2**	1,2,3,4
5,B	1,2,3,4



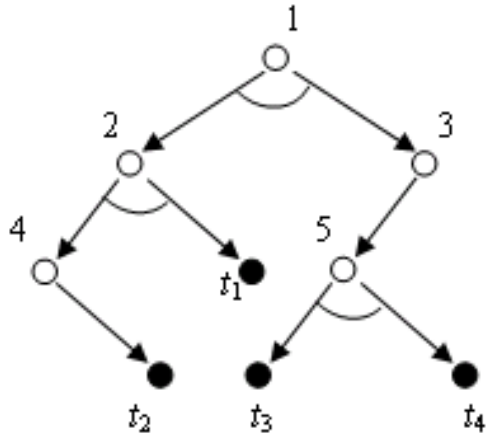
例：与/或树的宽度优先搜索



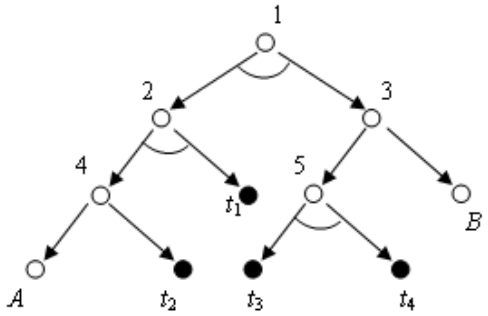
STEP5：

- 扩展节点5后，得到节点t3、t4；
- 节点t3、t4都是终叶节点，且为可解节点，对其先辈节点进行标示；
- 节点5可解，接着推出节点3可解，节点1可解.

open	close
1	
2,3	1
3,4,t1	1,2
4,t1,5,B	1,2,3
5,B	1,2,3,4
B,t3,t4	1,2,3,4,5

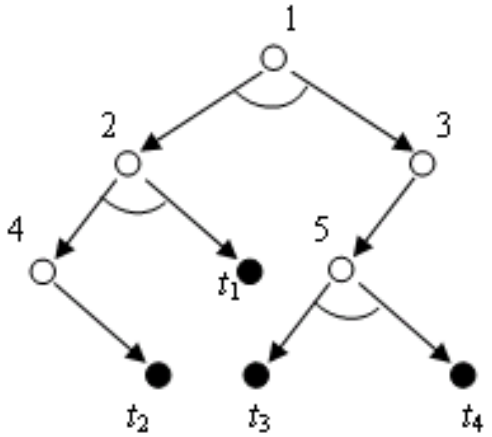


例：与/或树的宽度优先搜索

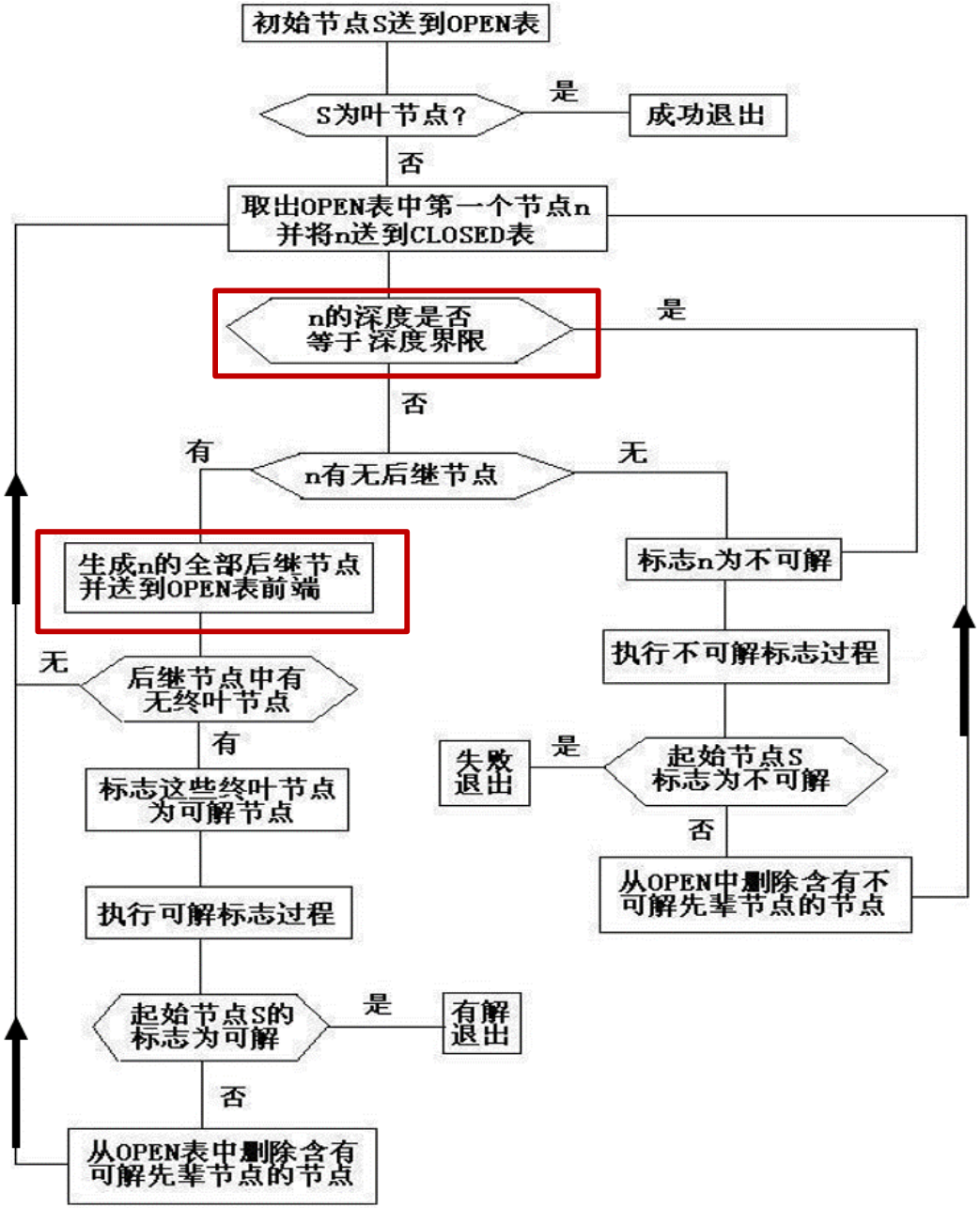


得到节点1可解，算法终止

open	close
1	[]
2,3	1
3,4,t1	1,2
4,t1,5,B	1,2,3
5,B	1,2,3,4
B,t3,t4	1,2,3,4,5



与或树的深度优先搜索

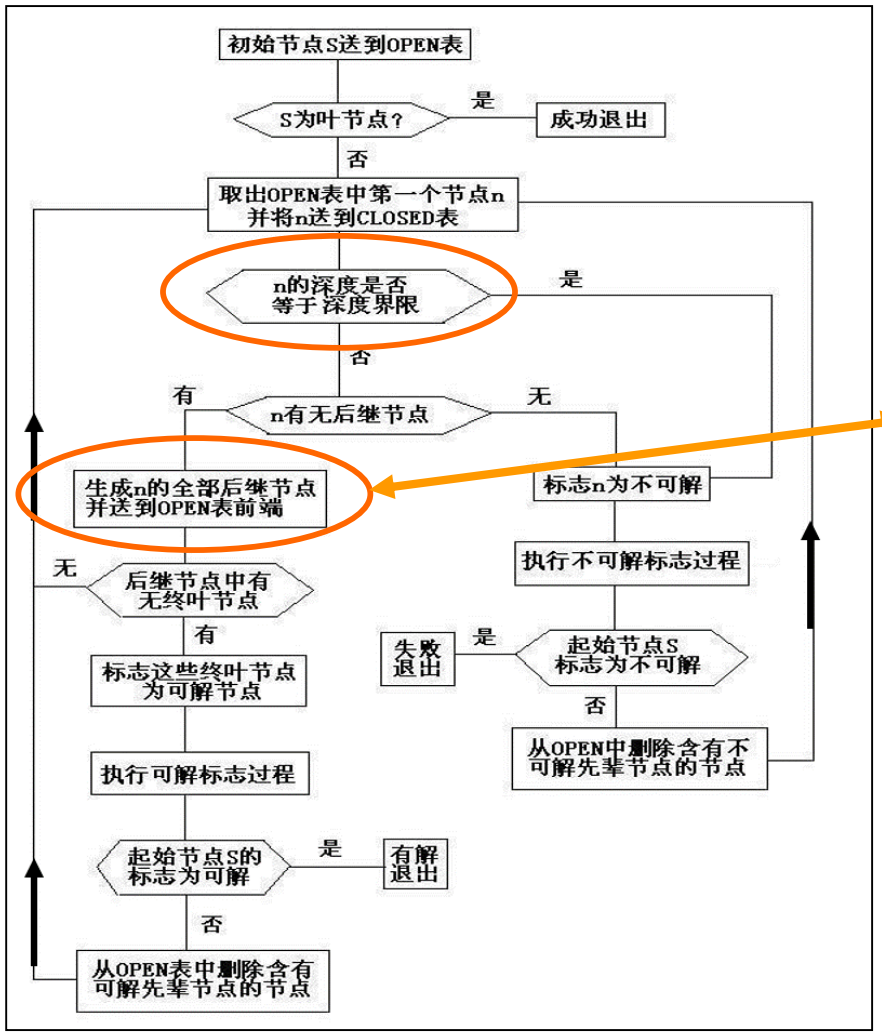

初始节点S送到OPEN表
S为叶节点？
是
成功退出
否
取出OPEN表中第一个节点n
并将n送到CLOSED表
n的深度是否
等于深度界限
是
否
有
n有无后继节点
无
生成n的全部后继节点
并送到OPEN表前端
标志n为不可解
执行不可解标志过程
无
后继节点中有
无终叶节点
有
失败
退出
是
起始节点S
标志为不可解
否
标志这些终叶节点
为可解节点
从OPEN中删除含有不
可解先辈节点的节点
执行可解标志过程
起始节点S的
标志为可解
是
有解
退出
否
从OPEN表中删除含有
可解先辈节点的节点

与或树深度优先搜索算法

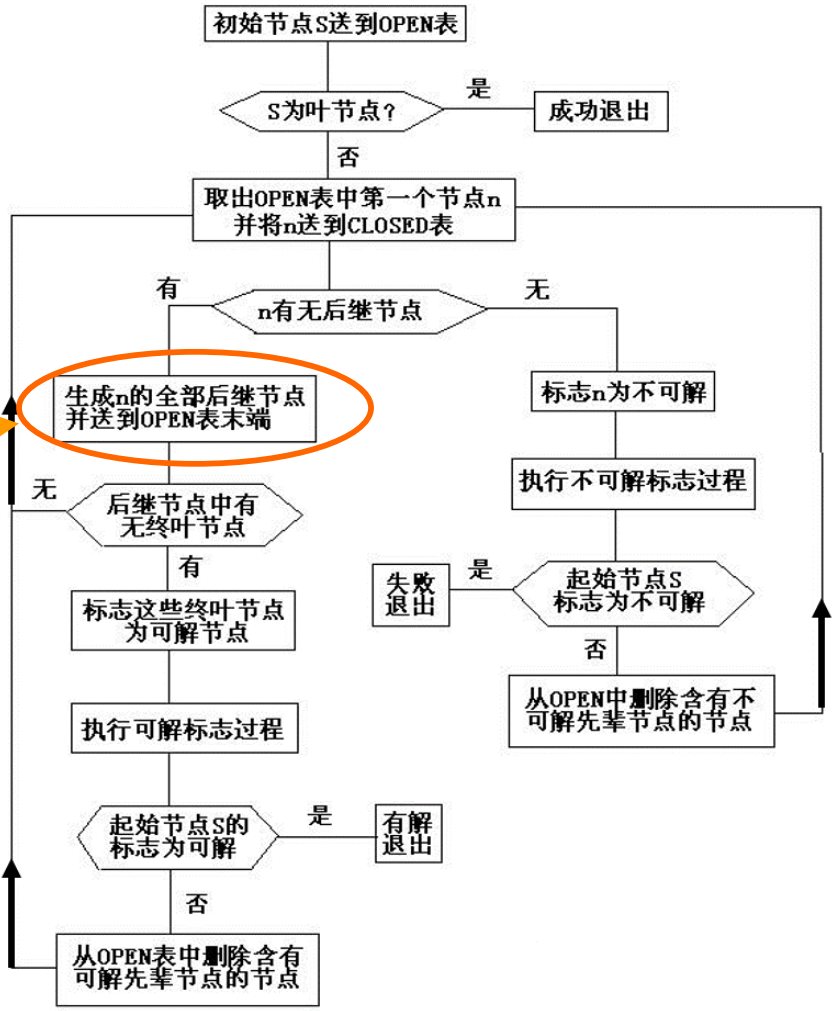
与宽度优先算法相比，深度优先算法的特殊之处：

- 第4步要判断从open表取出来的节点的深度。如果等于深度界限，认定它为不可解节点

- 第5步将扩展出来的节点放到open的前端，即open是堆栈



与或树深度优先搜索算法



与或树宽度优先搜索算法

深度优先搜索算法流程

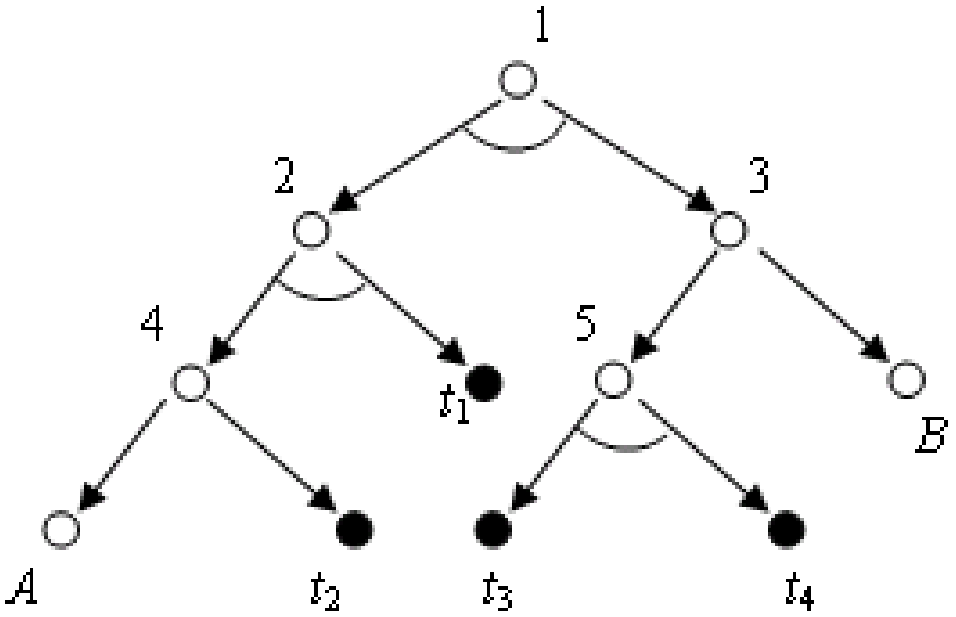
(1) 起始节点S送OPEN表
(2) 若S为叶节点，则成功结束，否则继续
(3) 取出OPEN表第一个节点（记作n），送到CLOSED表
(4) 若节点n的深度等于深度界限，则将n标志为不可解节点，并转10；否则继续
(5) 扩展节点n，生成全部后继节点，置于OPEN表前面，并设置指向n的指针
(6) （分三种情况）
- 如果 n 无后继节点，则标志为不可解节点，并转10，否则继续
- 若有后继节点为叶节点，则将这些叶节点标志为可解节点，并继续；
- 否则转3

(7) 实行可解标志过程
(8) 若起始节点为可解节点，则算法成功结束；否则，继续下一步
(9) 从OPEN表中删除含有可解先辈节点的节点，并转3
(10) 实行不可解标志过程
(11) 若起始节点为不可解，则失败结束，否则，继续下一步
(12) 从OPEN表中删去含有不可解先辈节点的节点
(13) 转3

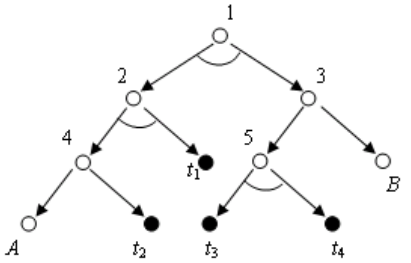
与/或树的深度优先搜索

例2：设有如图所示的与/或树，其中t_1, t_2, t_3, t_4均为终叶节点，A和B是不可解的端节点。

采用与/或树的深度优先搜索法对该图进行搜索。（规定深度界限为4）



与/或树的深度优先搜索



初始化：

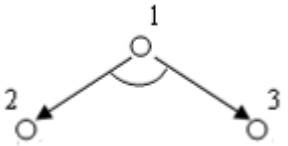
节点 1 送open表，且不为叶节点

open	close
1	[]

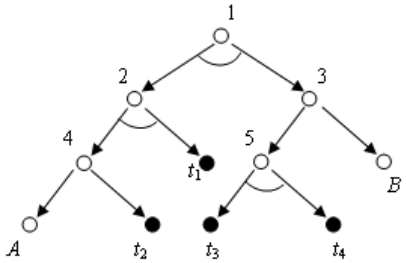
STEP1：

- 扩展节点1，得到节点2、3；
- 节点2，3都不是终叶节点，接着扩展节点2，此时open表只剩节点3.

open	close
1	
2,3	1
3	1,2



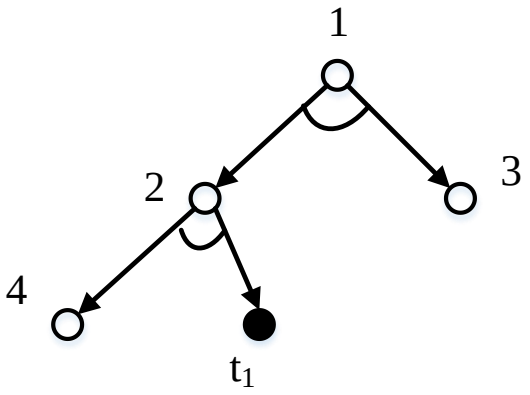
与/或树的深度优先搜索



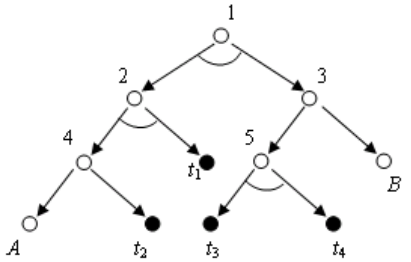
STEP2：

- 扩展节点2后，得到节点4, t1；
- 节点t1是终叶节点，对其先辈节点进行标示；
- 没有先辈节点被标记成可解节点，不需要调整OPEN表。

open	close
1	
2,3	1
4,t1,3	1,2
t1,3	1,2,4



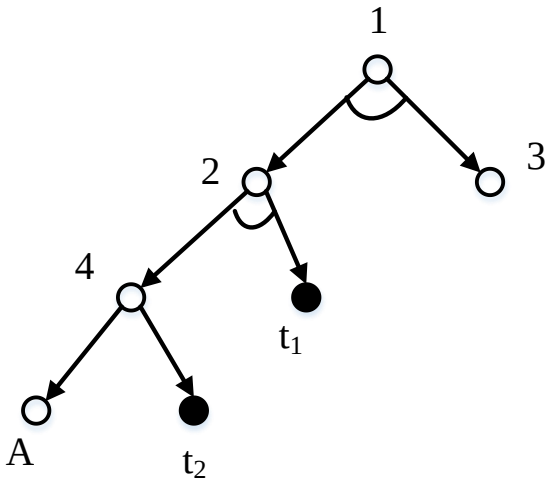
与/或树的深度优先搜索



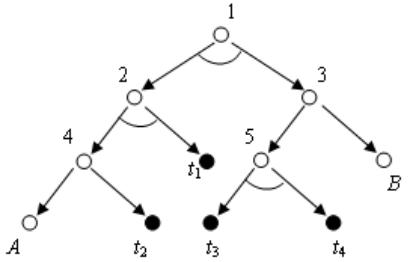
STEP3：

- 扩展节点4后，得到节点A, t2；
- 节点t2是终叶节点，A节点为不可解点，对其先辈节点进行标示；
- 先辈节点2被标记为可解节点，删除OPEN表中的A, t2, t1；

open	close
1	
2,3	1
4,t1,3	1,2
A,t2,t1,3	1,2,4
3	1,2,4
	1,2,4,3



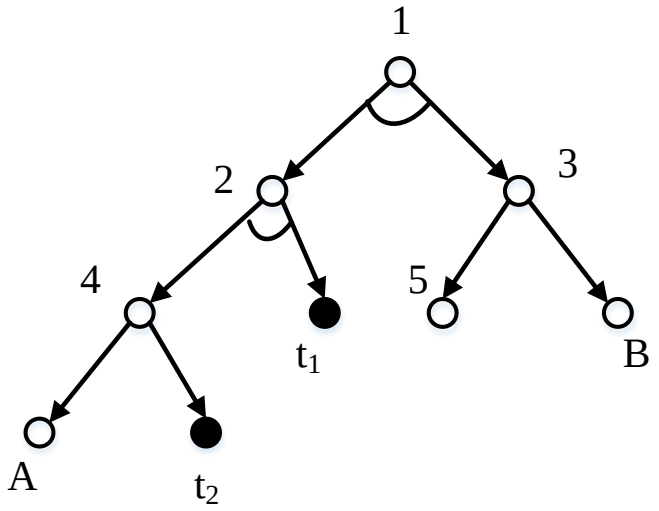
与/或树的深度优先搜索



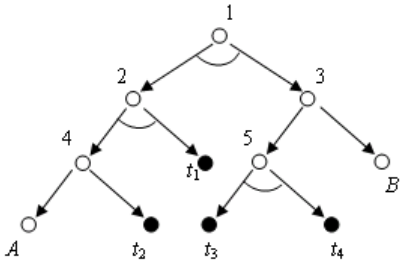
STEP4：

- 扩展节点3后，得到节点5、B；
- 节点B是不可解节点，对其先辈节点进行标示；

open	close
1	
2,3	1
4,t1,3	1,2
3	1,2,4
5,B	1,2,4,3
B	1,2,4,3,5



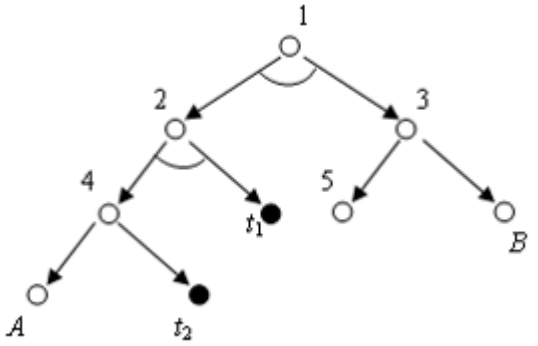
与/或树的深度优先搜索



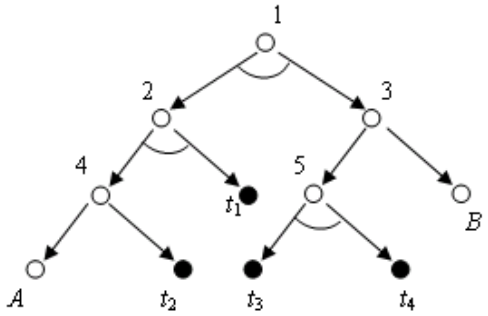
STEP5：

- 扩展节点5后，得到节点t3、t4；
- 节点t3, t4是终叶节点且标示为可解节点，对其先辈节点进行标示；
- 节点1被标记为可解节点

open	close
1	
2,3	1
4,t1,3	1,2
3	1,2,4
5,B	1,2,4,3
t3,t4,B	1,2,4,3,5

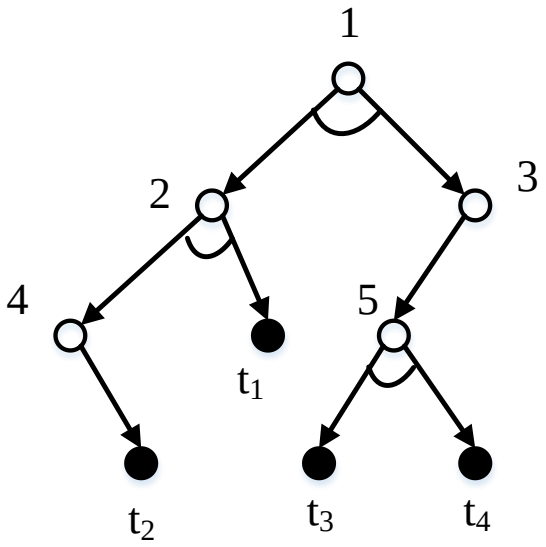


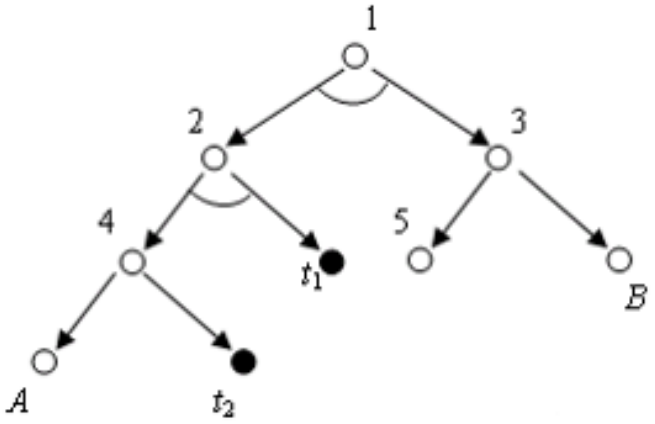
与/或树的深度优先搜索



得到节点1可解，算法终止

open	close
1	
2,3	1
4,t1,3	1,2
3	1,2,4
5,B	1,2,4,3
t3,t4,B	1,2,4,3,5





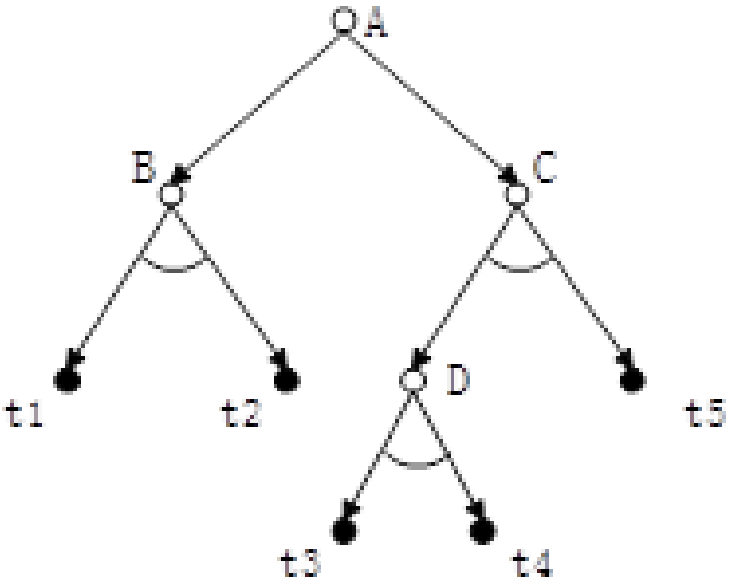
open	close
1	[]
2,3	1
3,4,t1	1,2
4,t1,5,B	1,2,3
5,B	1,2,3,4
B,t3,t4	1,2,3,4,5

open	close
1	
2,3	1
4,t1,3	1,2
3	1,2,4
5,B	1,2,4,3
t3,t4,B	1,2,4,3,5

与/或树的宽/深度优先搜索

作业

设有如下所示的与或树，请分别用与或树的宽度优先搜索和深度优先搜索求解树。（课堂）



机器博弈

你可曾听说过“深蓝”？

1997年5月11日，IBM开发的“深蓝”击败了国际象棋冠军卡斯帕罗夫。

卡氏何许人也？

- 6岁下棋
- 13岁获得全苏青年赛冠军
- 16岁获世界青年赛第一名
- 1980年他获得世界少年组冠军(17岁)

电脑棋手：永不停歇的挑战！

- 1988年“深思”击败了丹麦特级大师拉森。
- 1993年“深思”第二代击败了丹麦世界优秀女棋手小波尔加。

- 2001年“更弗里茨” 击败了除了克拉姆尼克之外的所有排名世界前十位的棋手。
- 2002年10月“更弗里茨”与世界棋王克拉姆尼克在巴林交手，双方以4比4战平。
- 2003年1至2月“更年少者”与卡斯帕罗夫在纽约较量，3比3战平。

许多人在努力

机器博弈

- 20世纪50年代，有人设想利用机器智能来实现机器与人的对弈。
- 1997年IBM的“深蓝”战胜了国际象棋世界冠军卡斯帕罗夫，惊动了世界。
- 加拿大阿尔伯塔大学的奥赛罗程序Logistello和西洋跳棋程序Chinook也相继成为确定的、二人、零和、完备信息游戏世界冠军
- 西洋双陆棋这样的存在非确定因素的棋类也有了美国卡内基梅隆大学的西洋双陆琪程序BKG这样的世界冠军。
- 2017年5月，在中国乌镇围棋峰会上，AlphaGo与排名世界第一的世界围棋冠军柯洁对战，以3比0的总比分获胜。
- 中国象棋、桥牌、扑克等许多种其它种类游戏博弈的研究也正在进行中。

博弈搜索

- 特征：智力竞技

机器博弈，意味着机器参与博弈，参与智力竞技。

我们这里的博弈只涉及双方博弈，常见的是棋类游戏，如：中国象棋，军旗，围棋等。

- 目标：取胜

取胜的棋局如同状态空间法中的目标状态。

与八数码游戏一样，游戏者需要对棋局进行操作，以改变棋局，使其向目标棋局转移。

然而，八数码游戏只涉及一个主体，不是博弈。

博弈涉及多个主体，他们按规则，依次对棋局进行操作，并且目标是击败对手。

- 方法：Max-Min搜索、α–β剪枝

博弈问题为何可以使用与或图来表示？

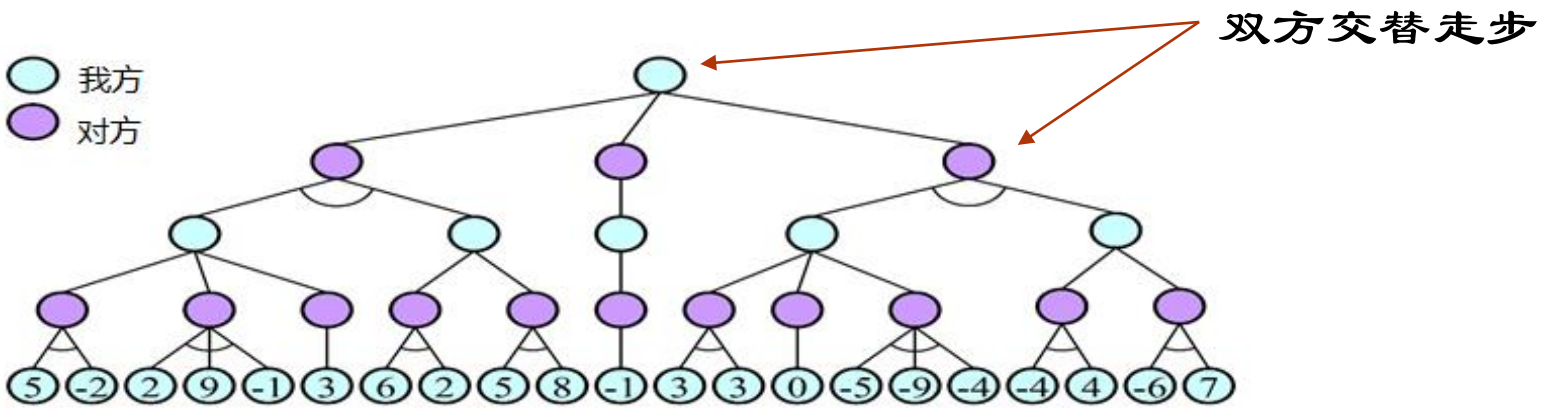
- 当轮到我方走棋时，只需要从若干个可以走的棋中选择一个棋走就可以，从这个意义上来说，这若干个可以走的棋是“或”的关系。
- 而轮到对手走棋时，对于我方来说，必须能够应付对手的每一步走棋，相当于这些棋是“与”的关系。

因此，博弈问题可以看作是一个与或图，但是与一般的与或图不同，是一种具有特殊结构的与或图。

博弈树

博弈树

- 如何根据当前的棋局，选择对自己最有利的一步棋 ？
- 博弈的问题表示：博弈树表示
  - 一种特殊的与或树。
  - 节点：博弈的格局（即棋局），相当于状态空间中的状态，反映了博弈的信息， 并且与节点、或隔层交替出现。



博弈树

➢ **描述博弈过程的与或树为博弈树，特点为：**

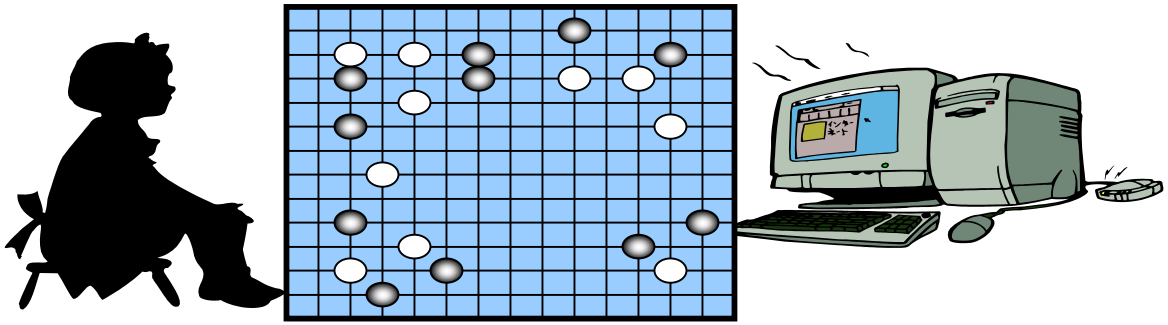
- 博弈的初始格局是初始节点；
- 博弈树中，或节点与与节点逐层交替出现，自己一方扩展节点是“或”关系，对方扩展节点是“与” 关系；
- 所有能使自己一方获胜的终局是本原问题，相应节点为可解节点，所有使对方获胜的终局都是不可解节点

➢ 博弈树搜索

- 极大极小（Max-Min）搜索
- α–β剪枝

双方博弈实例

以围棋为例，竞技的双方分为黑方和白方，由黑方开棋，双方轮流行棋，最终，谁占据的地盘大，谁就成为获胜方。



Max-Min搜索

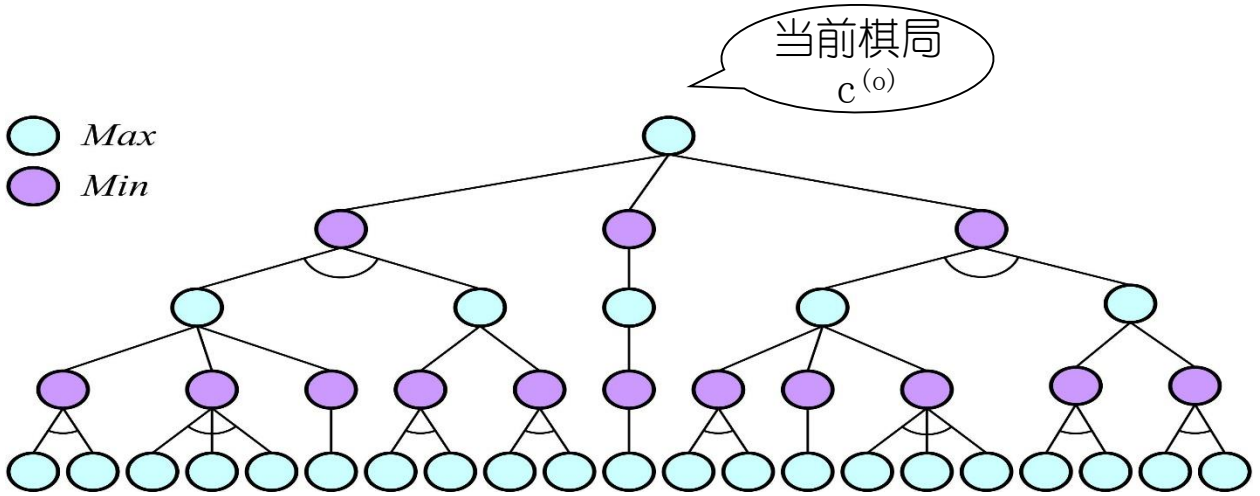
基本思想：

- 目的是为博弈的双方中的一方寻找一个最优行动方案；
- 要寻找这个最优方案，就要通过计算当前所有可能的方案来进行比较；
- 方案的比较是根据问题的特征来定义一个估价函数，用来估算当前博弈树端节点的得分；
- 当计算出端节点的估值后，再推算出父节点的得分（即计算倒推值）；
  - 对或节点，选其子节点中一个最大得分作为父节点的得分
  - 对与节点，选其子节点中一个最小得分作为父节点的得分
- 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。

Max-Min搜索

Step1.生成k-步博弈树

- Max 代表机器一方 / Min 代表敌方
- 设 Max 面对的当前棋局为 $c^{(o)}$，以 $c^{(o)}$为根，生成 k-步博弈树：



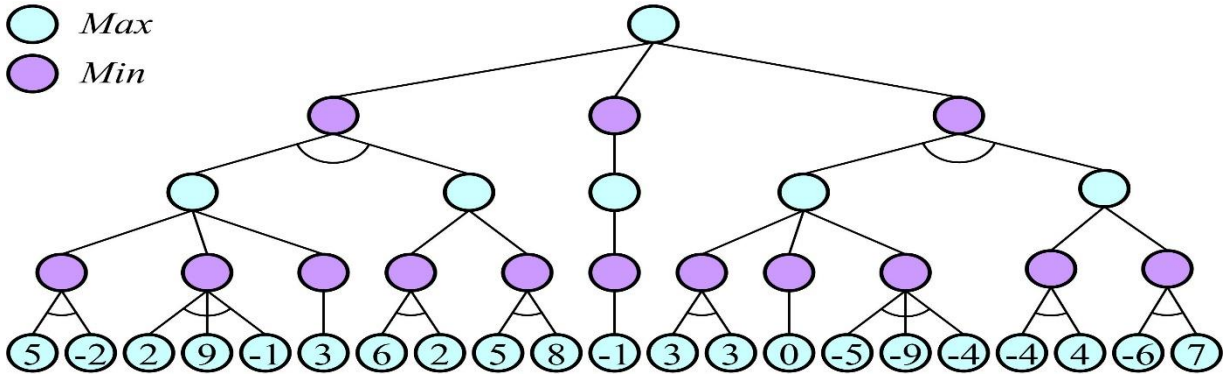
Max-Min搜索

Step2. 评估棋局(博弈状态)

- 估价函数

✓ 为特定的博弈问题定义一个估价函数est(c)，用以评估k-步博弈树叶节点对应的棋局c

✓ est(c)的值越大，意味着棋局c对Max越有利。

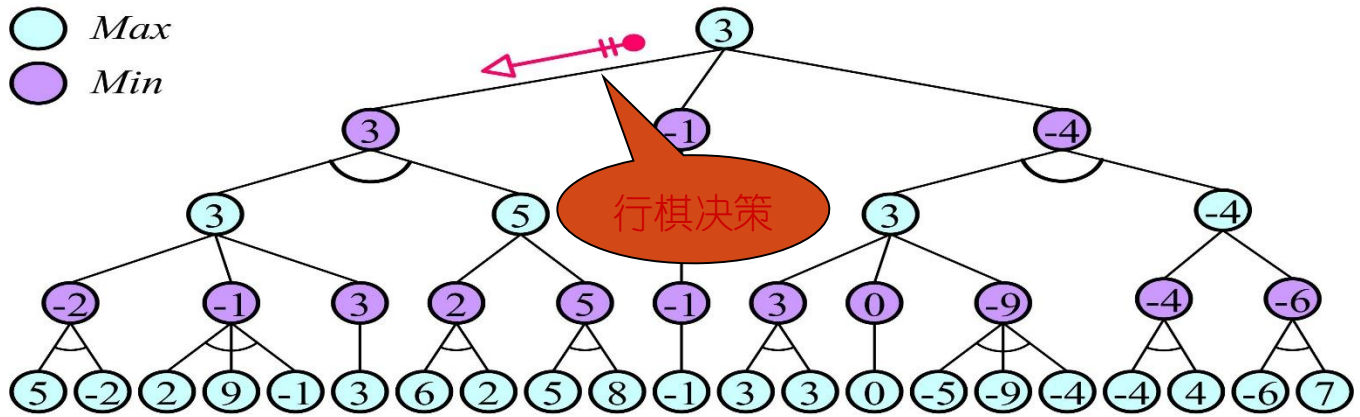


Max-Min搜索

Step3. 回溯评估

- 极大极小运算

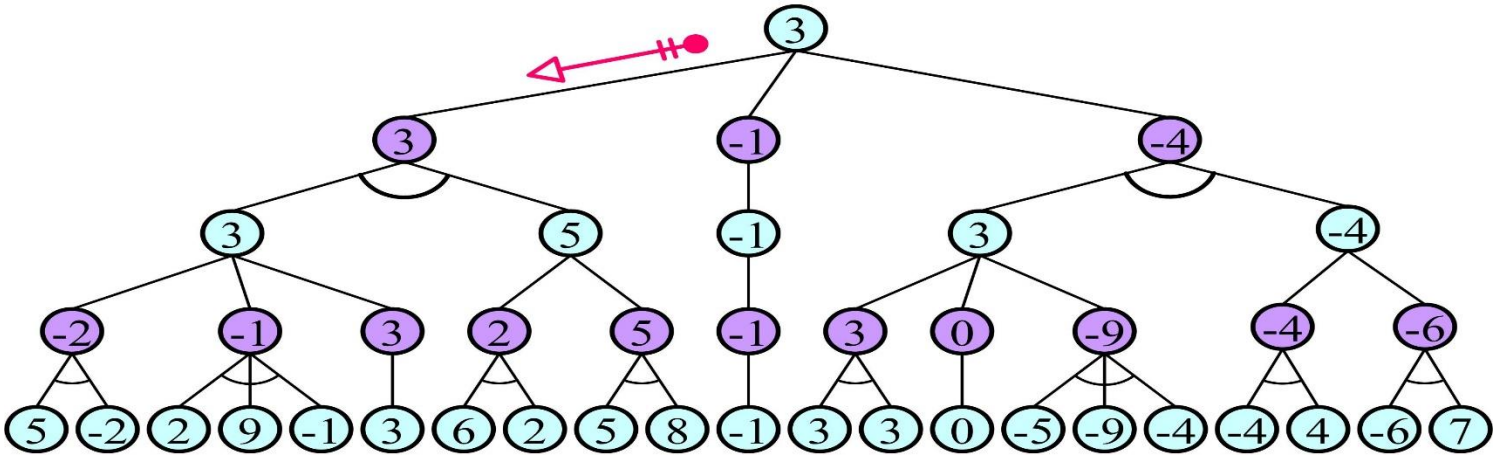
由叶节点向根节点方向回溯评估，在Max处取最大评估值(或运算)，在Min处取最小评估值(与运算)。



Max-Min搜索

Step3. 回溯评估

- 极大极小运算

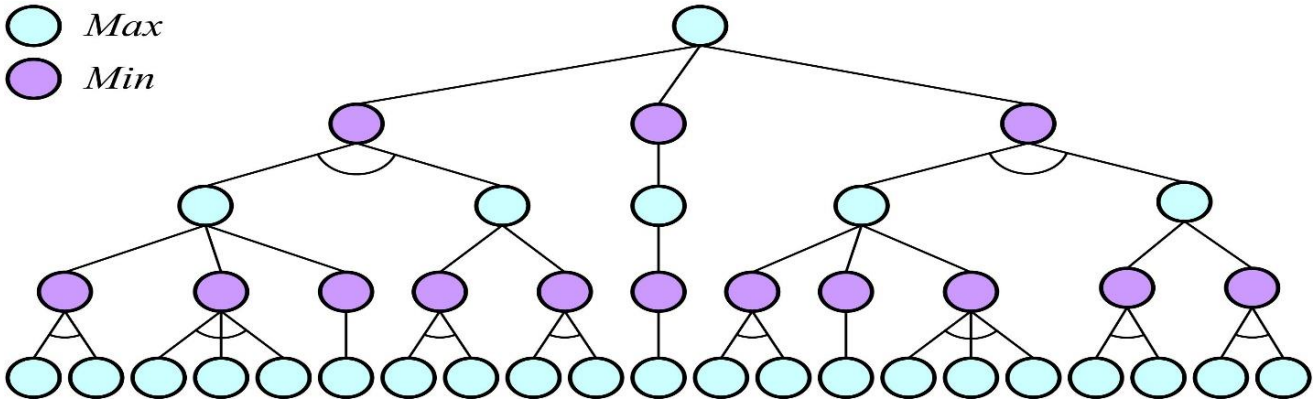


- Max 按取最大评估值的方向行棋

Max-Min搜索

Step4. 递归循环

- Max 行棋后，等待 Min 行棋;
- Min 行棋后，即产生对于 Max 而言新的当前棋局 $c^{(o)}$;
- 返回 Step1，开始下一轮博弈。



Max-Min搜索

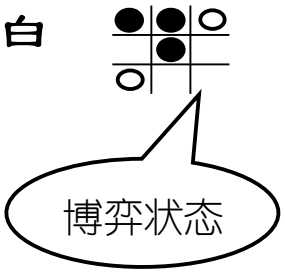
Max-Min搜索流程总结：

- Step1：以 c(o) 为根，生成 k-步博弈树；
- Step2：评估博弈树叶节点对应的博弈状态(棋局)；
- Step3：进行极大极小运算 (Max-Min 运算)；
- Step4：等待 Min 行棋，产生新的 c(o)，返回 step1.

Max-Min搜索

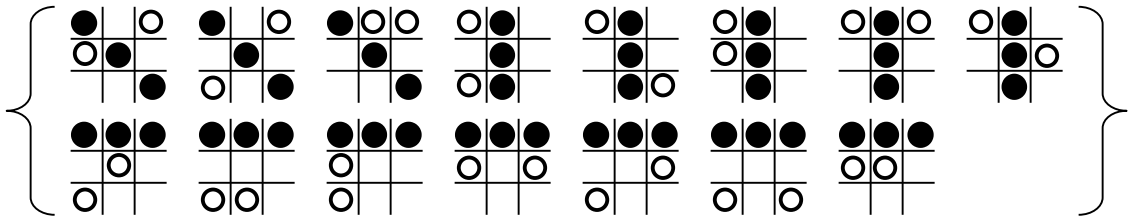
一字棋：

设有 3×3 棋格，Max 与 Min 轮流行棋，黑先白后，先将 3 颗棋子连成一线的一方获胜。



- 一字棋博弈空间：共有 9！种可能的博弈状态
- 一字棋算子空间：博弈规则集合
- 一字棋博弈目标集合 (对 Max而言)：

$\sum c^{(g)} =$



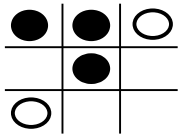
Max-Min搜索

一字棋：

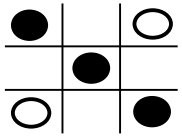
- 定义估价函数：*est*(c)

(1) 对于非终局的博弈状态c，估价函数为：

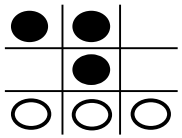
$est(c)$=(所有空格都放上黑色棋子之后，3颗黑色棋子连成的直线总数)－(所有空格都放上白色棋子之后，3颗白色棋子连成的直线总数)

est(c)=3–2=1　　例如：c= 

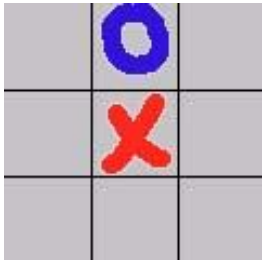
(2) 若 c 是 Max 的胜局，则：

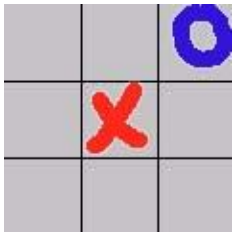
est(c) = +∞　　例如：c= 

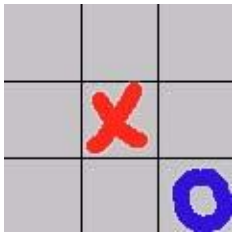
(3) 若 c 是 *Min* 的胜局，则：

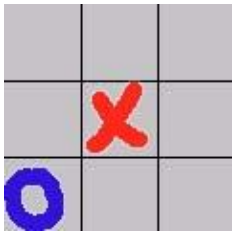
est(c) = –∞　　例如：c= 

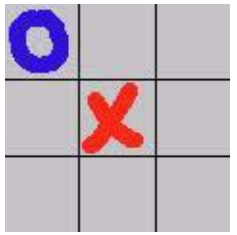
需要说明的是，等价的 (如具有对称性的) 棋局被视为相同棋局。





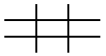


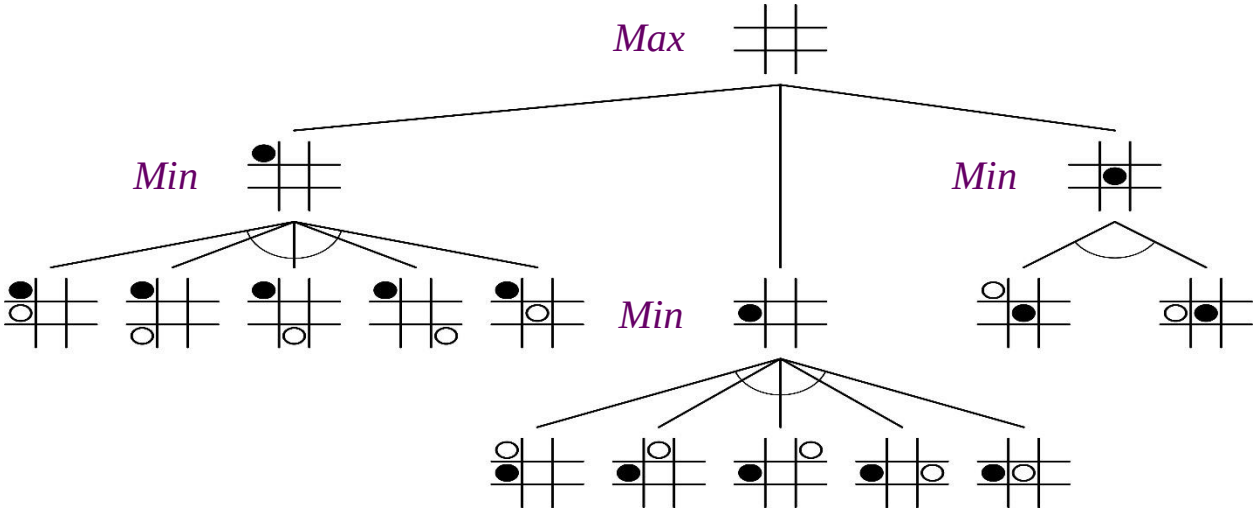




用叉号表示MAX，用圆圈代表MIN。

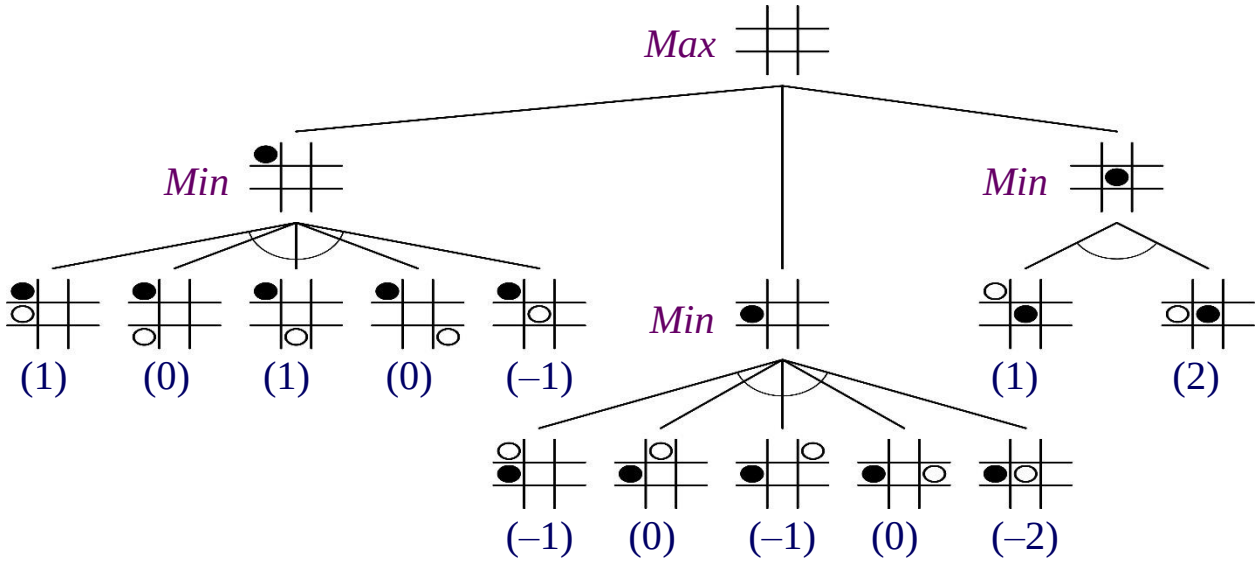
Max-Min搜索

step1. 以 $c^{(o)}$ =  为根，生成2-步博弈树：



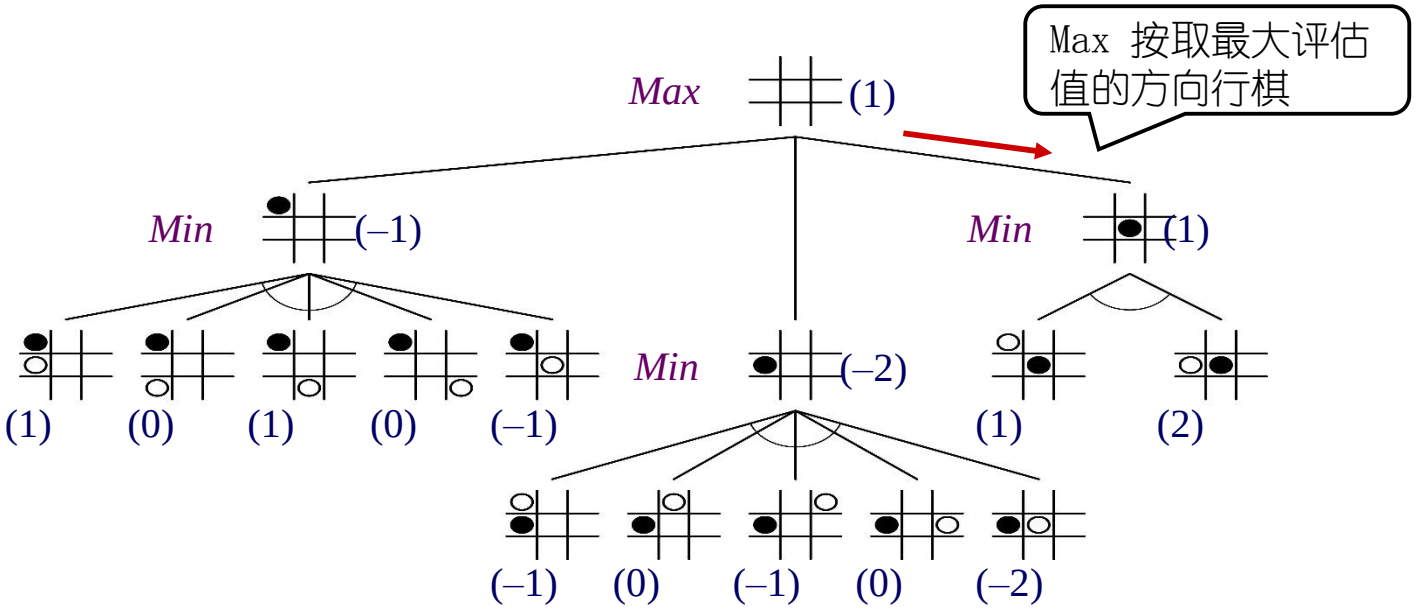
Max-Min搜索

step2. 评估博弈树叶节点对应的博弈状态



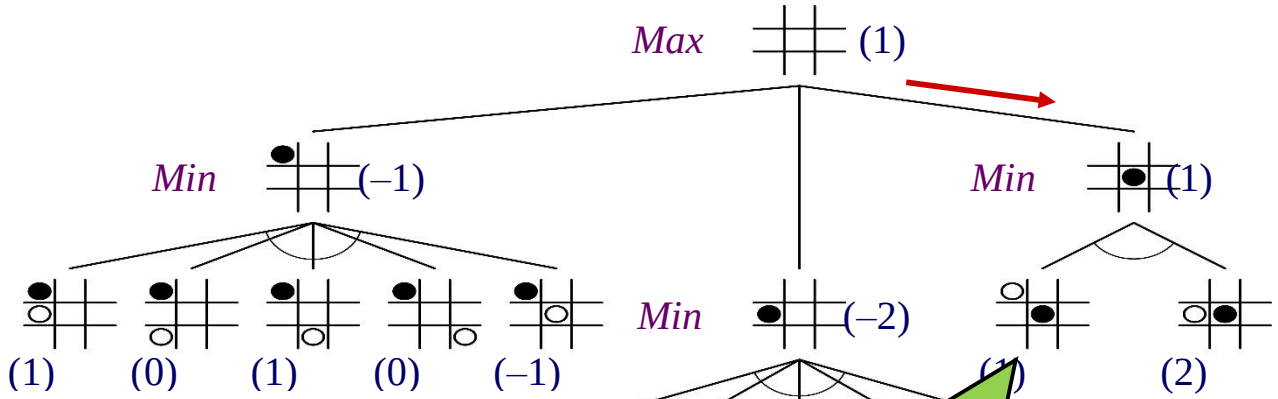
Max-Min搜索

step3. 进行极大极小运算(Max-Min运算)

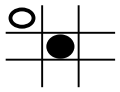


Max-Min搜索

step4. 等待 Min 行棋，产生新的 $c^{(o)}$，返回 step1.



则下一轮：$c^{(o)}$ =



α–β 剪枝

首先分析极小极大分析法效率，上述的极小极大分析法，实际是先生成一棵博弈树，然后再计算其倒推值，至使极小极大分析法效率较低。于是在极小极大分析法的基础上提出了α-β剪枝技术。

α-β剪枝技术的基本思想，边生成博弈树边计算评估各节点的倒推值，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，即相当于剪去了博弈树上的一些分枝，从而节约了机器开销，提高了搜索效率。

α–β 剪枝

- 实际上，就博弈而言，人类棋手的思维模式更多地表现出深度优先的特征，而不是宽度优先。
- 因此，采用深度优先搜索策略进行k-步博弈搜索，更符合AI模拟人类智能的原则，这里的k是深度优先搜索的一个自然的深度界限。
- 深度优先搜索策略产生的k-步博弈树是可以剪枝的，因此，搜索空间较小。重要的是，这正是人类棋手约束搜索空间的特征。

α–β 算法的剪枝规则

对于一个**与**节点来说，它取当前子节点中的最**小**倒推值作为它倒推值的**上界**，称此为β值；（β <= 最小值 ）

对于一个**或**节点来说，它取当前子节点中的最**大**倒推值作为它倒推值的**下界**，称此为α值．（α >= 最大值）

规则一（α 剪枝规则）：

任何**与**节点x的β值如果不能升高其父节点的α值，则对节点x以下的分支可停止搜索，并使x的倒推值为β

规则二（β 剪枝规则）：

任何**或**节点x的α值如果不能降低其父节点的β值，则对节点x以下的分支可停止搜索，并使x的倒推值为α

由规则一形成的剪枝被称为 **“α 剪枝”**，而由规则二形成的剪枝被称为 **“β 剪枝”**。

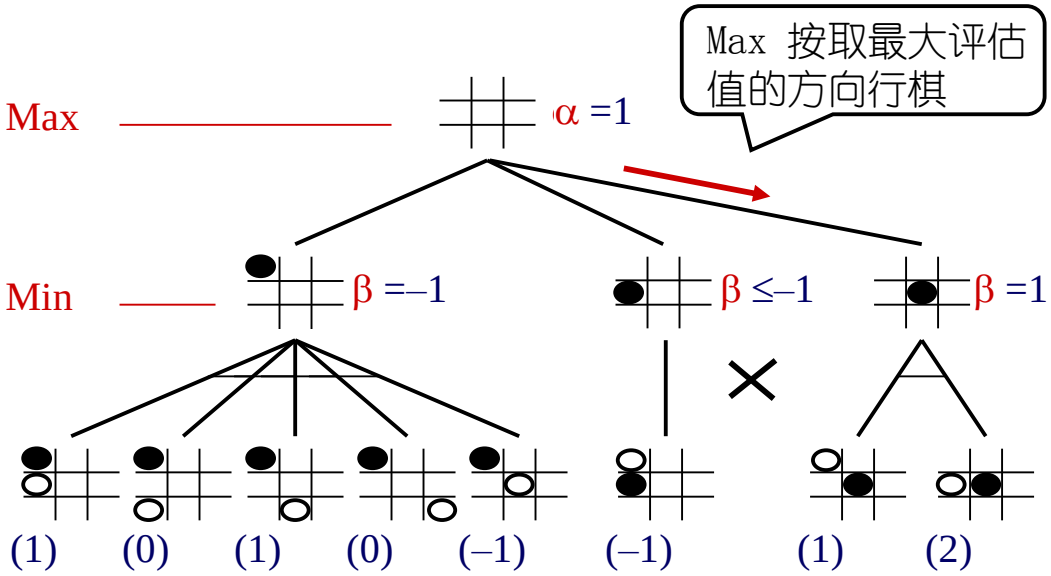
α–β 剪枝

一字棋：

- 搜索策略：k-步博弈；深度优先；每次扩展一个节点；一边扩展一边评估。
- 在 α -β 算法中：

  α： Max 节点评估值的下界

  β： Min 节点评估值的上界



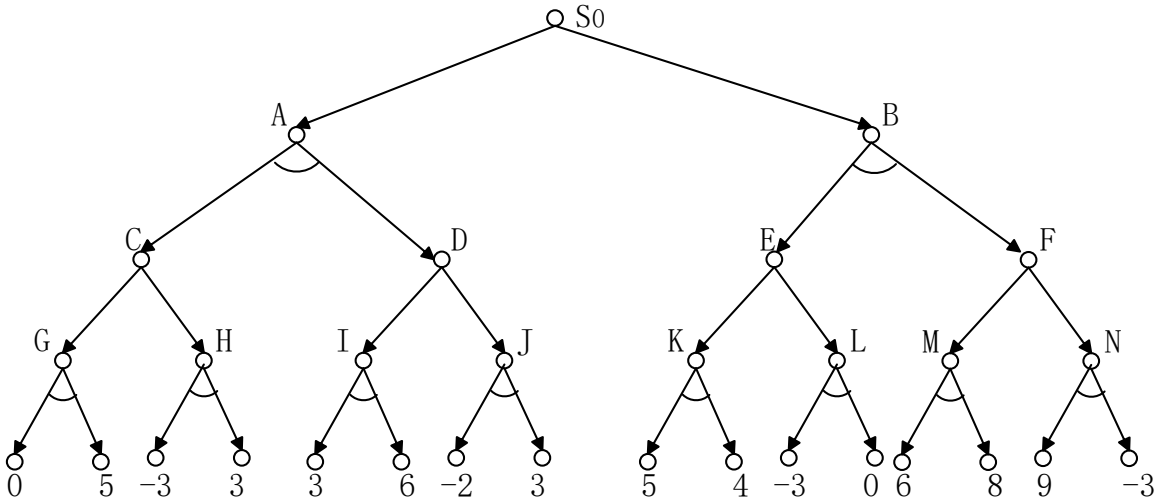
一字棋的 α–β 搜索过程

作 业

- 设有如下图所示的博弈树，其中最下面的数字是假设的估值。该博弈树做如下工作：

①计算各节点的倒推值。

②利用 $\alpha-\beta$ 剪枝技术剪去不必要的分枝。



小结

（1）问题归约法从目标（要解决的问题）出发，逆向推理，通过一系列变换把初始问题变换为子问题集合和子-子问题集合，直至最后归约为一个平凡的本原问题集合。这些本原问题的解可以直接得到从而解决了初始问题，用与或图来有效地说明问题归约法的求解途径。

（2）问题的分解过程转化为了与或图的搜索过程，不断分解，根据得到的子问题的解的情况不断去反向判断原问题的可解性，直至确定原问题可解或不可解。

（3）博弈问题是特殊的与或图结构，其关键为节点得分的估计和得分值的反推操作。

77