产生式系统的搜索(2)

张文生

中国科学院自动化研究所

内容

- **A***算法的单调性
- 可分解产生式系统的搜索(AO*)
- 博弈树

- A*算法的单调性
- 可分解产生式系统的搜索(AO*)
- 博弈树

起因

- 在**A***算法中，扩展一个节点时，对已经在**OPEN**表或**CLOSED**表中的子节点，要调整指针，花时间和精力。
- 如果在扩展节点**n**时，就已经找到了从根节点开始到它的最优路径，则不必调整指针，可以大大提高效率。
- 如果满足单调性限制，则可实现此愿望。

单调性

- 如果对每一个节点n_i以及它的后继节点n_j，满足：

  $$h(n_i) - h(n_j) \leqslant k(n_i,n_j)$$

  则称启发式函数满足单调性限制。

  其中 $k(n_i,n_j)$： 连接n_i, n_j的弧上的费用(耗散值)

- 直观含义
  - $h(n_i) - h(n_j)$小
  - $h(n_j)$下降得慢

- $h(n_i) - h(n_j) \leqslant k(n_i,n_j)$
- 合理，但并不严格，很多函数都满足此点。

  - 例子：**eight-puzzle**
  - $f(n) = d(n) + h(n)$
  - **h(n):** 与目标相比, 错位的数字数目;
  - $k(n_i, n_j) \equiv 1$

	3	3

$h(n_i) - h(n_j)=1$

	3	3

$h(n_i) - h(n_j)=0$

		3
		3

$h(n_i) - h(n_j)=-1$

- 性质一：
  - 如果**A***满足单调性限制，则当它选择节点**n**扩展时，就已经发现了通向节点**n**的最佳路径，即**g(n)=g*(n)**。

- 证明
  - 如果**n**是初始状态，则结论显然成立。
  - 设**n**不是初始状态，令路径$n_0, n_1, \ldots\ldots, n_k$是从$n_0$到$n_k$的最佳路径，$n_k=n$。
  - 如果n_k的生成过程是n_0生成$n_1, \ldots\ldots, n_{k-1}$生成$n_k$，则结论显然成立。
  - 否则，在最佳路径中，一定存在一个节点n_{i+1}未被n_i生成.

- **n_i**或者在**OPEN**表中，或者还没有被生成（不在**OPEN**表和**CLOSED**表中）。

- 由于**n_0(S)**一定被扩展了，即，一定存在一个最佳路径中的节点**n_l**，它由最佳路径上的节点生成，并在**OPEN**表中，但是未被扩展，**$1 \le l < k$**（即，**n_{l+1}**不是由**n_l**生成）。

- 对任意最佳路径上的节点，根据单调性限制，$h(n_i) - h(n_{i+1}) \leqslant k(n_i,n_{i+1})$， 即, $h(n_i) \leqslant k(n_i,n_{i+1}) + h(n_{i+1})$，
- 两边同时加上$g^*(n_i)$，

  $$g^*(n_i)+h(n_i) \leqslant g^*(n_i)+k(n_i,n_{i+1}) + h(n_{i+1})$$
- 因为 $g^*(n_{i+1})= g^*(n_i)+ k(n_i,n_{i+1})$

  $$g^*(n_i) + h(n_i) \leqslant g^*(n_{i+1}) + h(n_{i+1})$$
- 取$i=l$, $g^*(n_l) + h(n_l) \leqslant g^*(n_{l+1}) + h(n_{l+1})$

- 利用传递性， $g^*(n_l) + h(n_l) \leqslant g^*(n_k) + h(n_k)$

- 由于n_l由最佳路径上的节点生成，所以

  $$f(n_l)= g(n_l) + h(n_l)= g^*(n_l) + h(n_l)$$

  即，$f(n_l) \leqslant g^*(n) + h(n)$

- 由于A*扩展了n，而不是n_l，根据性质 $f(n) \leqslant f(n_l)$

  即，$g(n) + h(n) \leqslant f(n_l) \leqslant g^*(n) + h(n)$,

- 则有 $g(n) \leqslant g^*(n)$

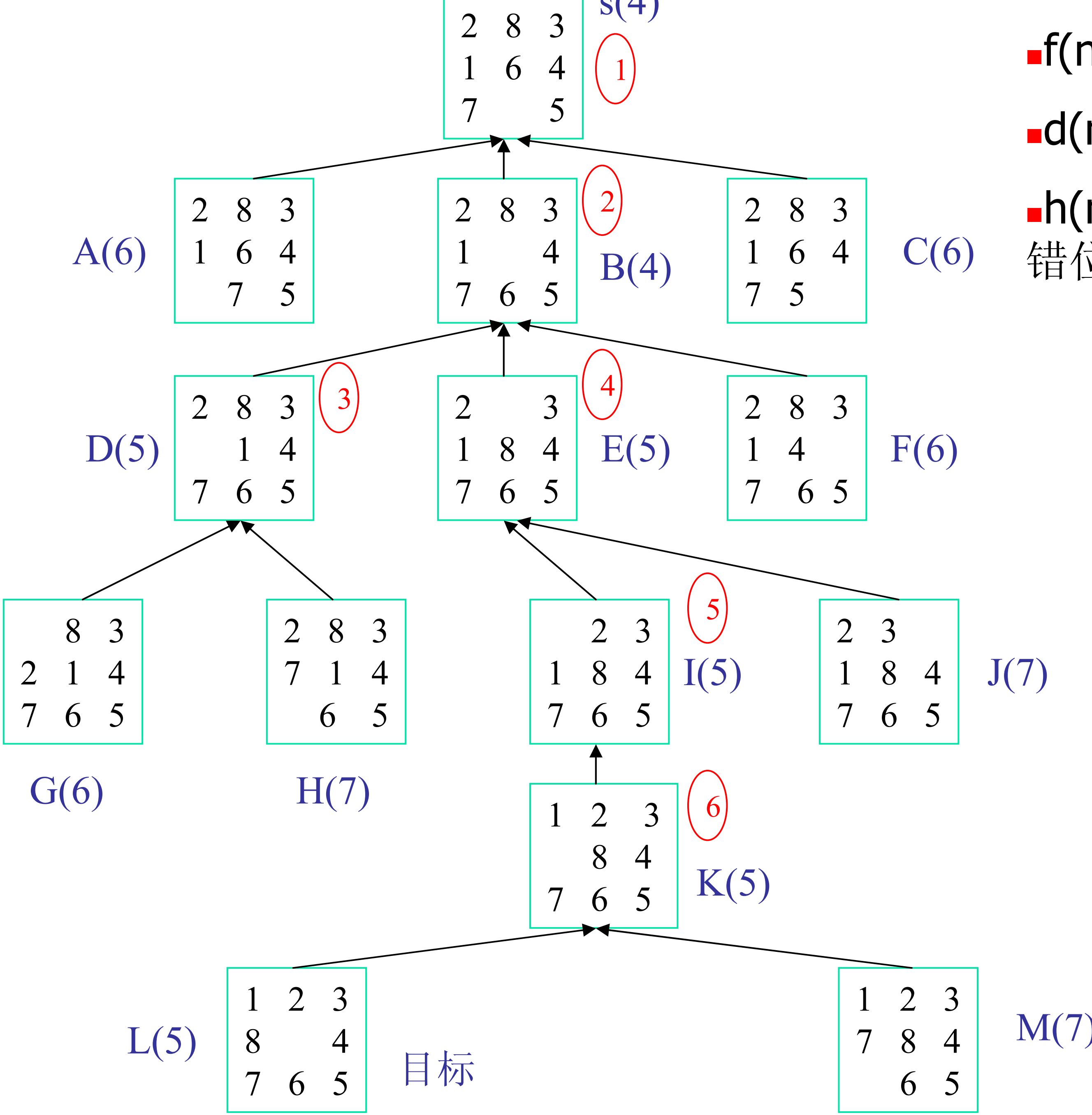
- 另一方面，根据A*的定义：$g(n) \geqslant g^*(n)$

- 因此，$g(n) = g^*(n)$

- 性质二：
  - 如果**A***满足单调性限制，则它扩展的节点序列的估值函数是单调上升的。

- **A***
  - 每次取**OPEN**表中**f(n)**最小的节点扩展**;**

{3, 4, 7, 9, 13}	{}
{8, 9, 4, 7, 9, 13}	{3}
{8, 9, 3, 6, 7, 9, 13}	{3,4}



- $f(n) = d(n) + h(n)$
- $d(n)$: 节点n的深度;
- $h(n)$:与目标相比，错位的数字数目;

- 证明:
  - 设A*扩展的节点序列是$\{n_0,n_1,\ldots\ldots,n_k\}$，要证对任意$0 \leqslant i<k$，$f(n_i) \leqslant f(n_{i+1})$。
  - 对任意$0 \leqslant i<k$，A*扩展了n_i后又扩展了n_{i+1}，
  - 如果A*扩展了n_i时n_{i+1}已经在OPEN表中，则$f(n_i) \leqslant f(n_{i+1})$。否则，A*扩展了$n_i$时$n_{i+1}$不在OPEN表中，由于$n_{i+1}$显然也不在CLOSED表中;

- 而**A***扩展了n_i后立刻扩展了n_{i+1}，所以n_{i+1}是作为n_i 的后继节点加入到**OPEN**表中，因此，当选择n_{i+1}扩展时，有:

$$f(n_{i+1}) = g(n_{i+1}) + h(n_{i+1})$$

$$= g^*(n_{i+1}) + h(n_{i+1}) \quad \text{（性质一）}$$

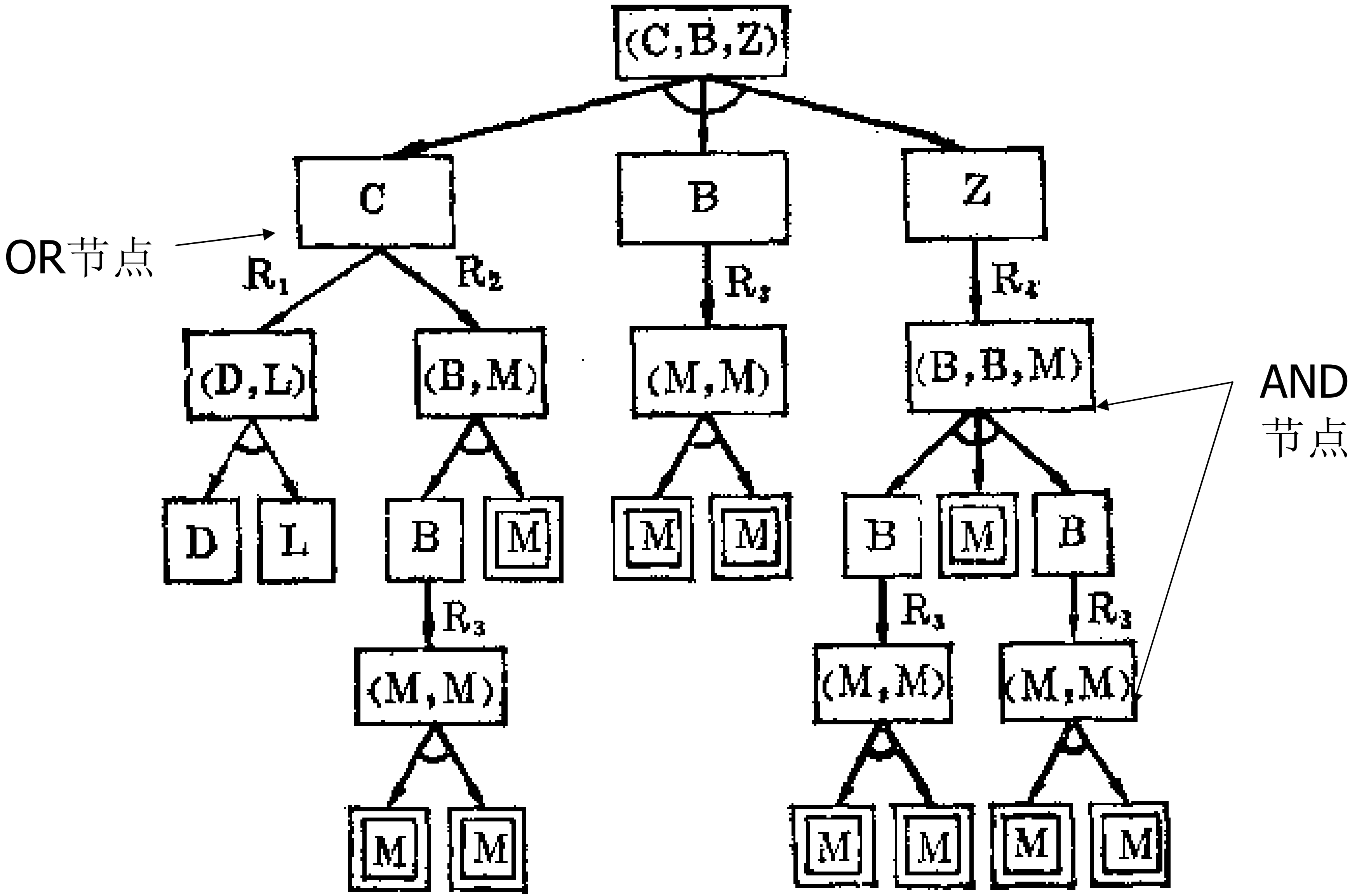
$$= g^*(n_i) + k(n_i,n_{i+1}) + h(n_{i+1})$$

- 由单调性限制，$k(n_i,n_{i+1}) + h(n_{i+1}) \geqslant h(n_i)$

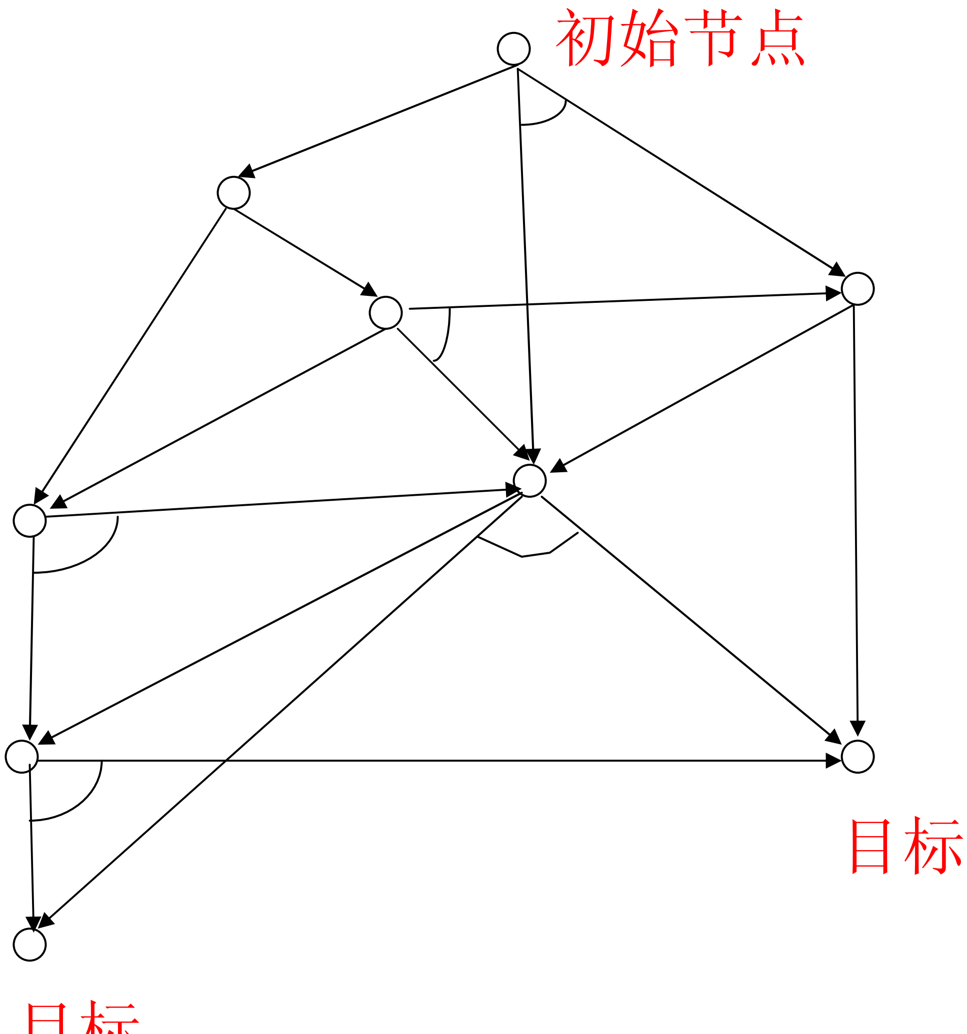
- 所以，$f(n_{i+1}) \geqslant g^*(n_i) + h(n_i) = f(n_i)$

- **A***算法的单调性
- 可分解产生式系统的搜索**(AO*)**
- 博弈树

可分解产生式

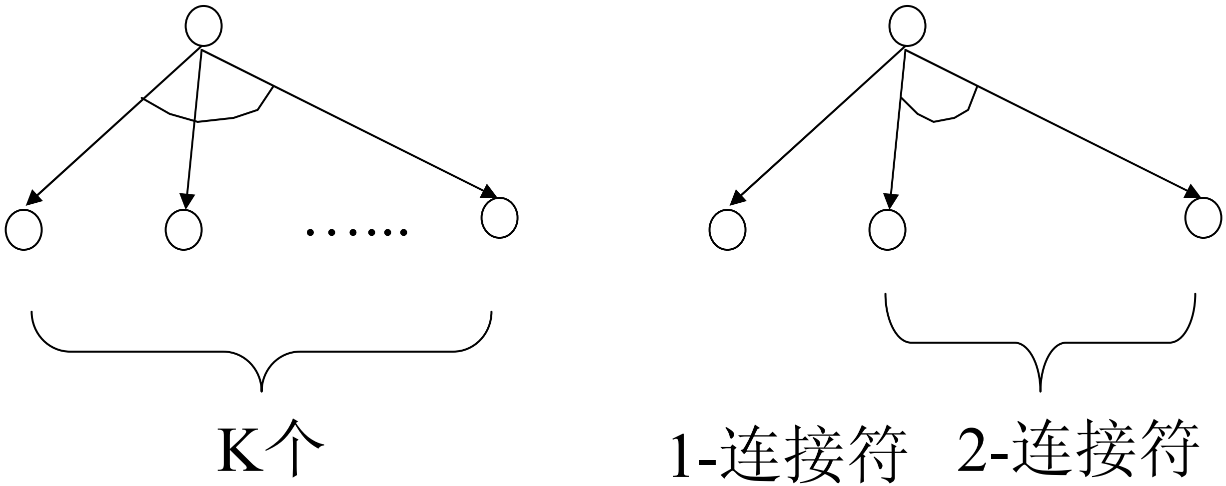

(C,B,Z)
C
B
Z
OR节点
R1
R2
R3
R4
(D,L)
(B,M)
(M,M)
(B,B,M)
AND
节点
D
L
B
M
M
M
B
M
B
R3
R3
R3
(M,M)
(M,M)
(M,M)
M
M
M
M
M
M

AND/OR图搜索

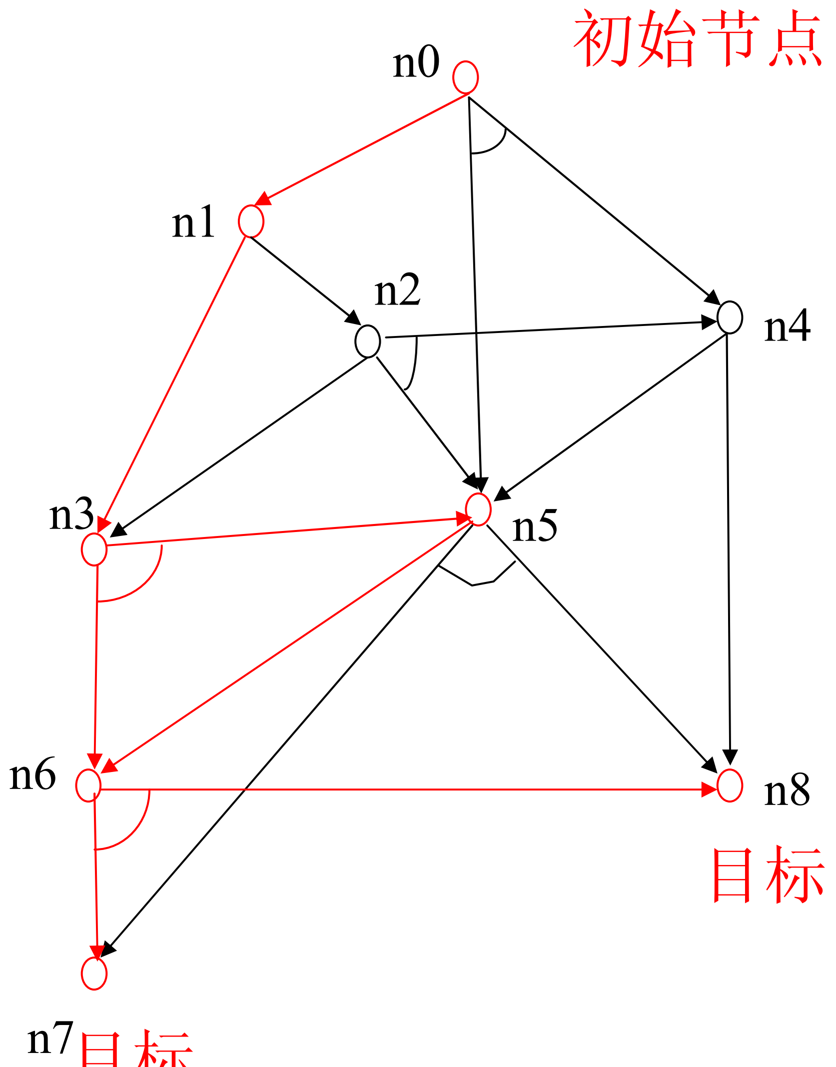


基本概念

- 与或图是一个超图，节点间通过连接符连接。
- K-连接符：
  - 从一个父亲节点指向k个后继节点的连接符



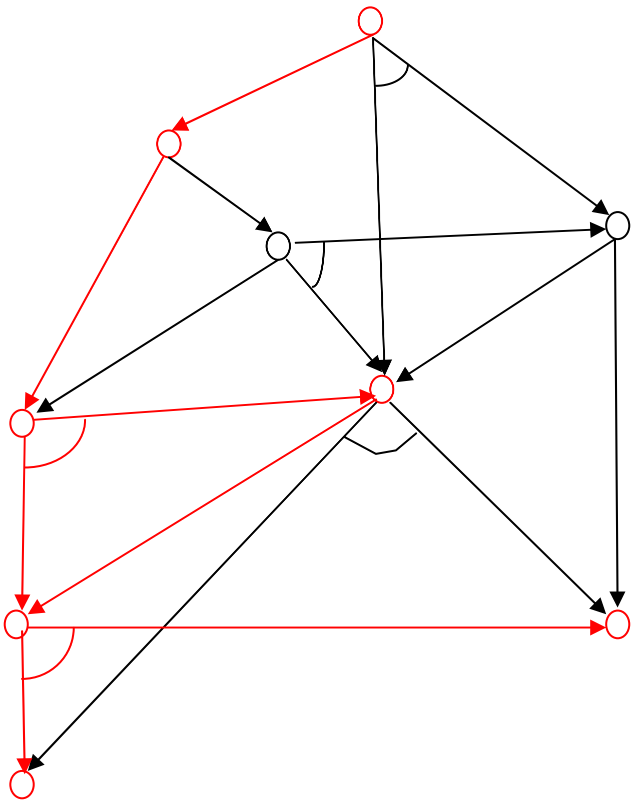
解图

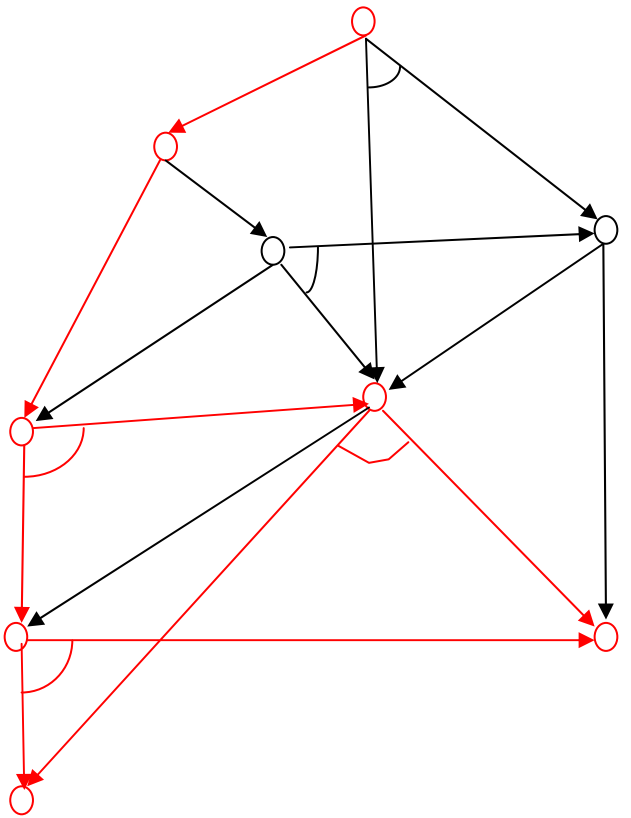


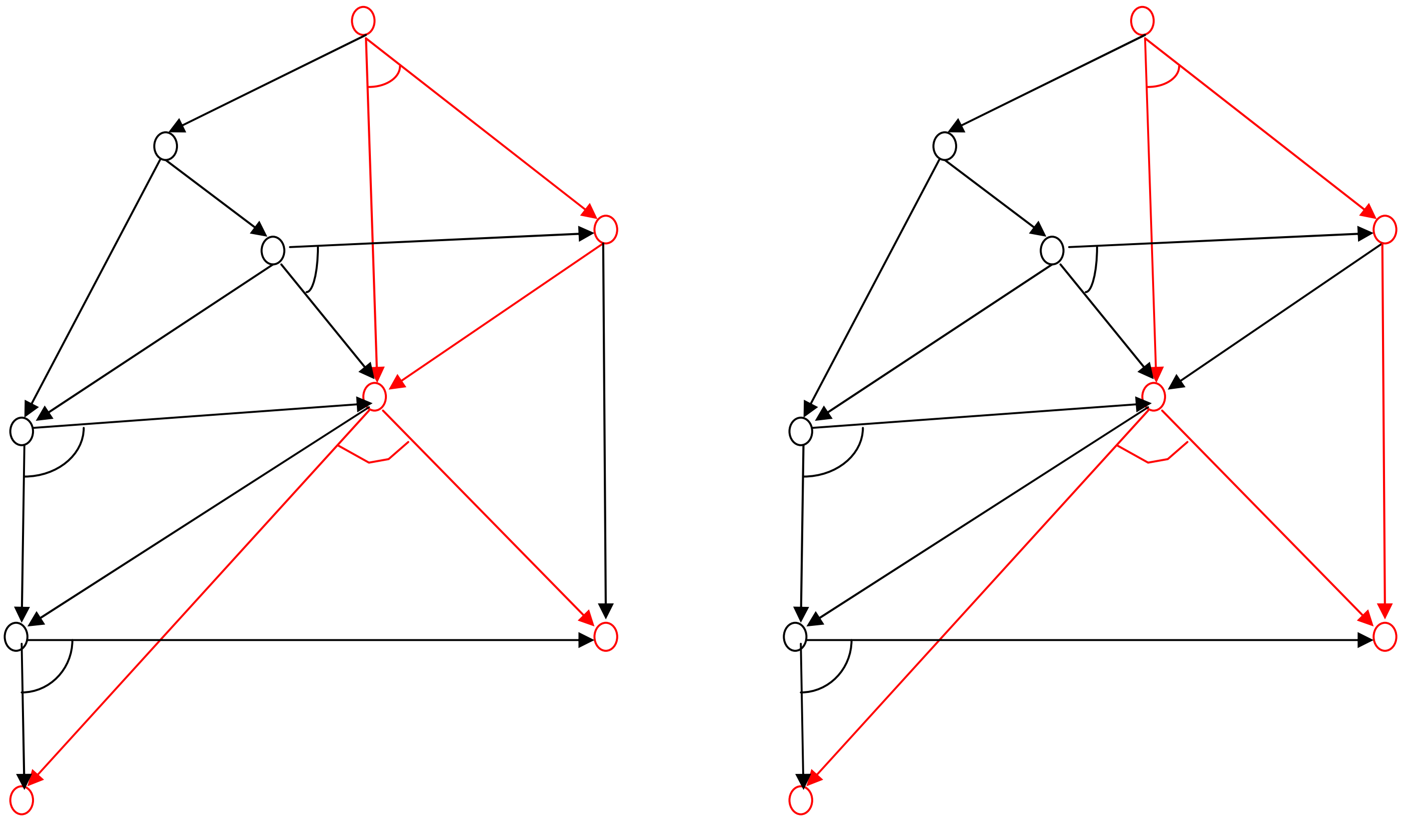
解图

- 定义（解图）：
  - 令**G**是一个**AND/OR**图，**N**是目标节点集合，则从**n**到**N**的解图，记为**G'**，是**G**的子图。
  - 如果**n**∈**N**，则**G'**只包括节点**n**。
  - 如果有一个从**n**出发的指向后继节点$\{n_1, n_2, \ldots\ldots n_k\}$的**k-**连接符，而每个$n_i$都有从$n_i$到**N**的解图，则**G'**由节点**n**，**k-**连接符，节点$\{n_1, n_2, \ldots\ldots n_k\}$，以及由每个$n_i$出发的解图构成。
  - 否则，从**n**到**N**没有解图。

例子







解图的费用

- 从节点**n**到目标节点集合**N**的解图的费用**k(n，N)**，递归定义如下：

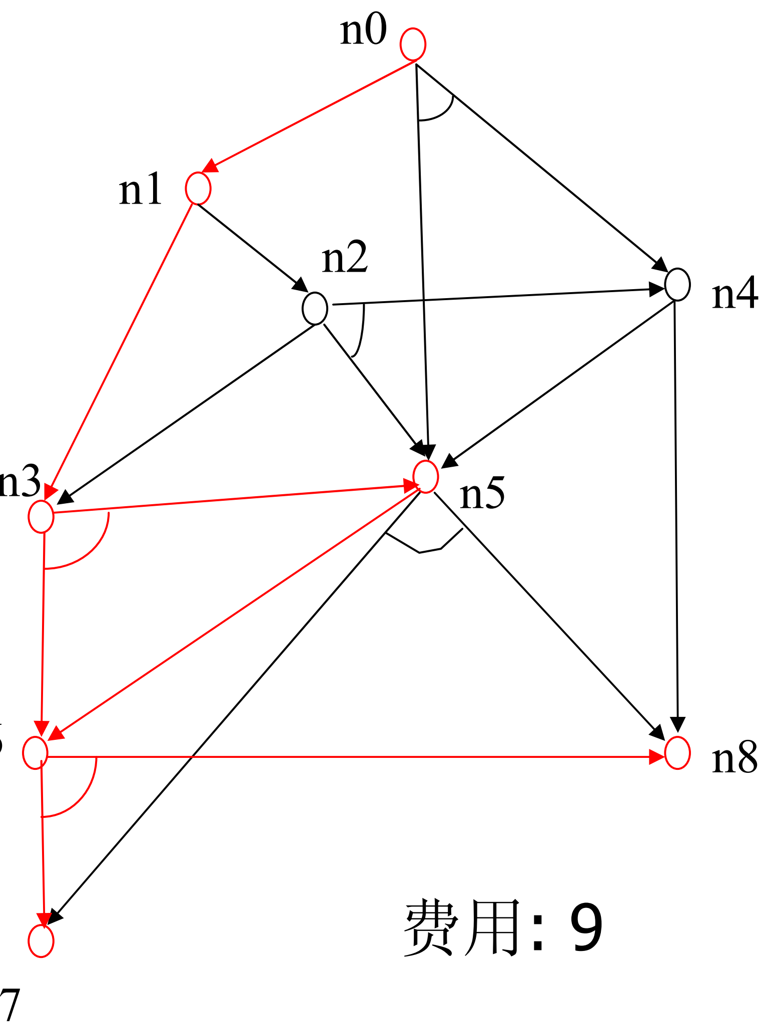
  如果**n**∈**N**，则**k**（**n**，**N**）**=0**；

  否则, 有从**n**出发的一个**m-**连接符指向**n**的后继节点$\{n_1, \ldots\ldots, n_m\}$，设此连接符的费用为$C_n$，则

$$k(n, N)= C_n + k(n_1, N)+\ldots\ldots+k(n_m, N)$$

例子: 解图的费用(1)

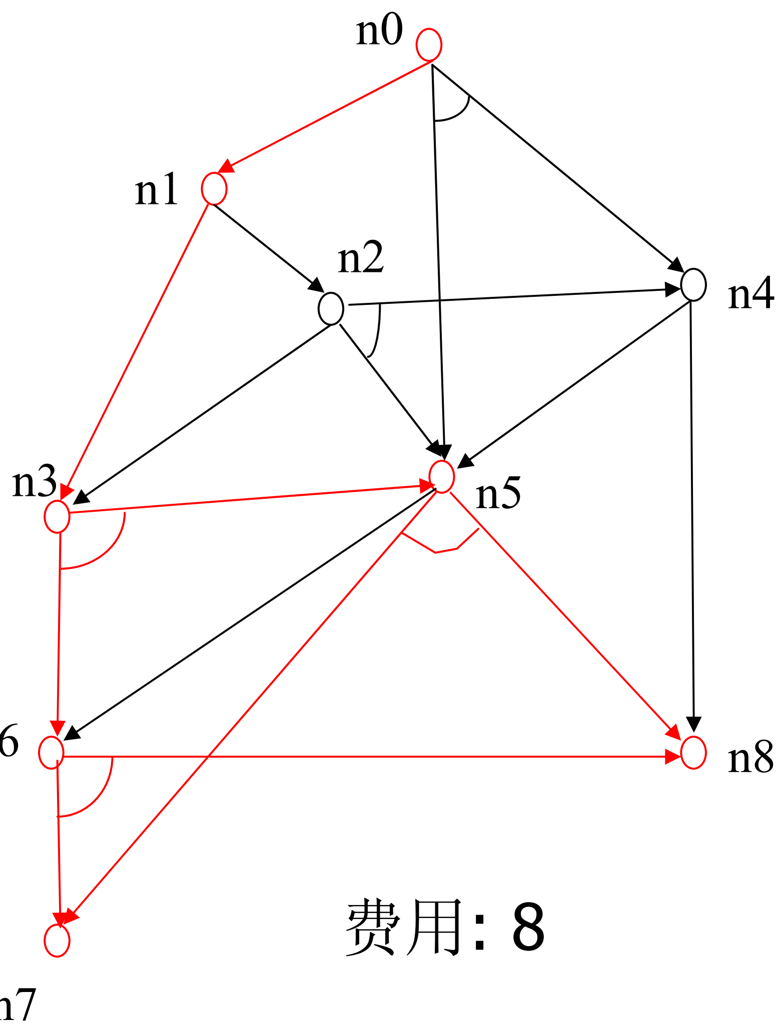
假定**k-**连接符的耗散值**C_n** =k
如果**n**∈**N**，则**k**（**n**，**N**）**=0**



费用: 9

k(n0, N)
=C(n0,n1)+K(n1,N)
=1+K(n1,N)
=1+C(n1,n3)+K(n3,N)
=1+1+K(n3,N)
=1+1+2+K(n5,N)+K(n6,N)
=1+1+2+1+K(n6,N)+K(n6,N)
=1+1+2+1+2+K(n7,N)+K(n8,N)
+K(n6,N)
=1+1+2+1+2+K(n6,N)
=1+1+2+1+2+2+K(n7,N)+K(n8,N
=1+1+2+1+2+2 = 9

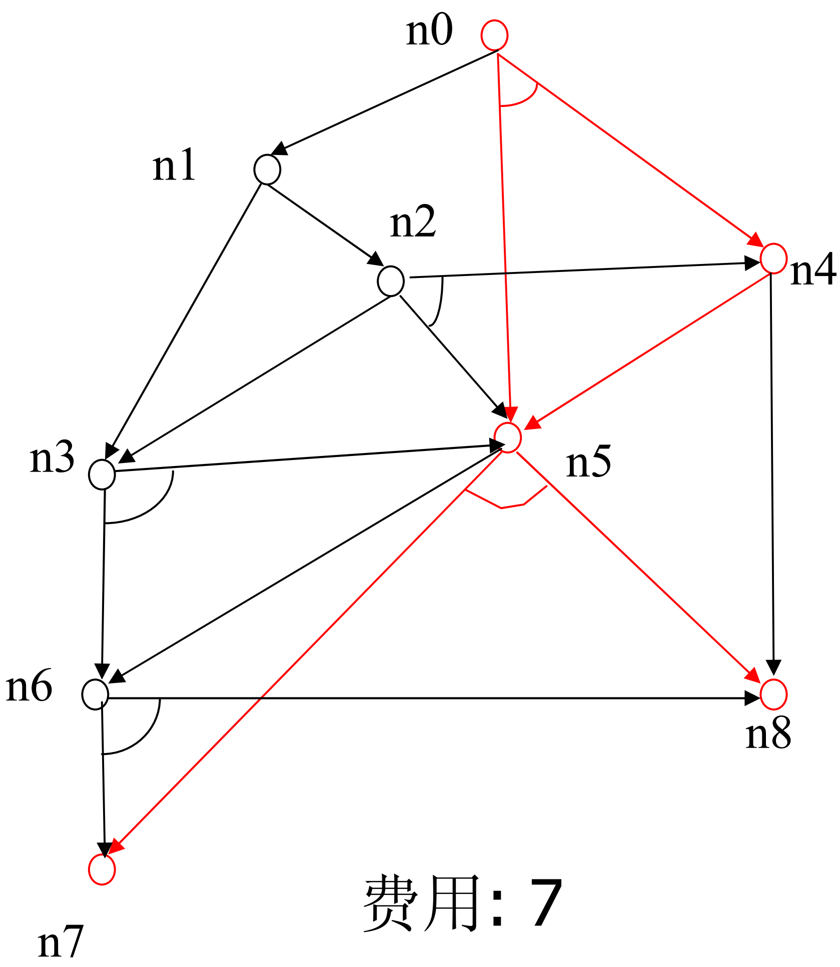
例子: 解图的费用(2)



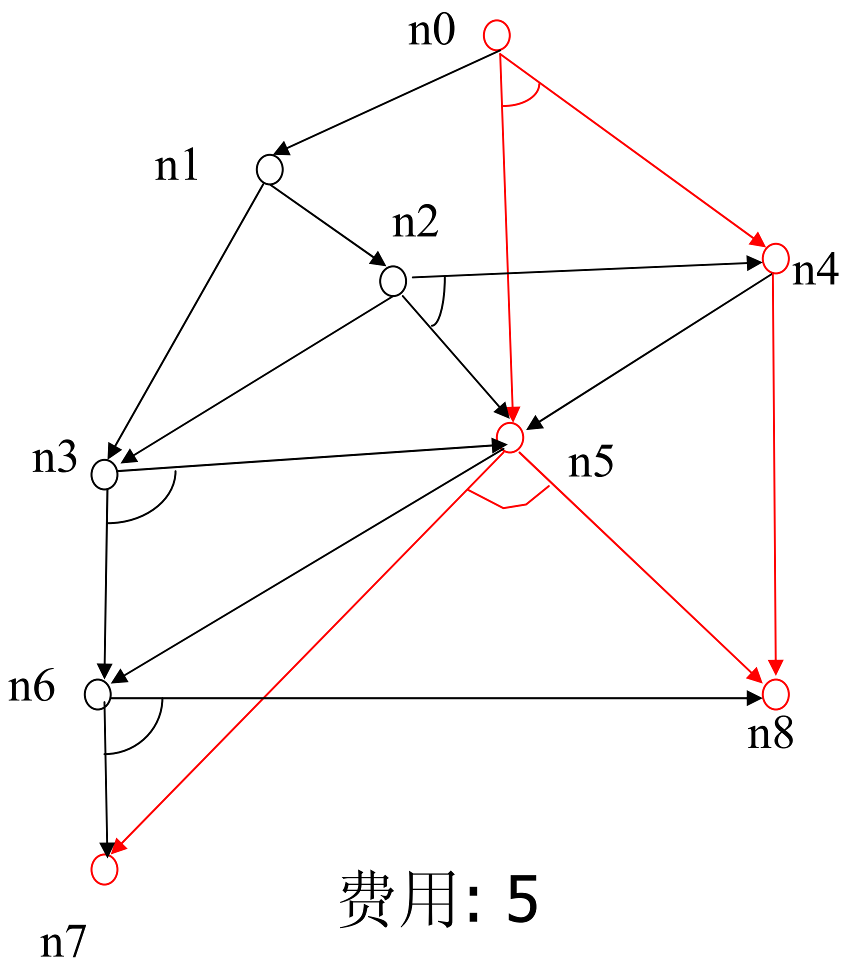
费用: 8

k(n0, N)
=C(n0,n1)+K(n1,N)
=1+K(n1,N)
=1+C(n1,n3)+K(n3,N)
=1+1+K(n3,N)
=1+1+2+K(n5,N)+K(n6,N)
=1+1+2+ 2+K(n7,N)+K(n8,N)
+K(n6,N)
=1+1+2+2+K(n6,N)
=1+1+2+2+2+K(n7,N)+K(n8,N)
=1+1+2+2+2
=8

例子: 解图的费用(3)



费用: 7



费用: 5

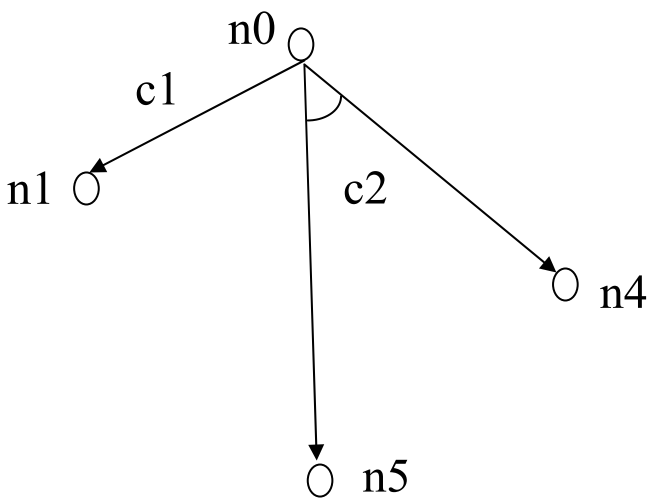
最佳解图

- 最佳解图
  - 从**n**到**N**的解图中，费用最小的解图称为最佳解图。

AO*算法

- 利用**h(n)**探索求解

  启发式的与或图搜索过程和通常的图搜索类似，通过评价函数来引导搜索过程。由于搜索的是一个解图，不考虑**g**（**n**），只考虑**h(n)**，并企图通过**h(n)**对**h*(n)**进行估计。



$$k(n0, N) = C1+K(n1,N)$$

$$=1+K(n1,N)$$

$$k(n0, N) = C2+K(n4,N)+K(n5,N)$$

$$=2+K(n4,N)+K(n5,N)$$

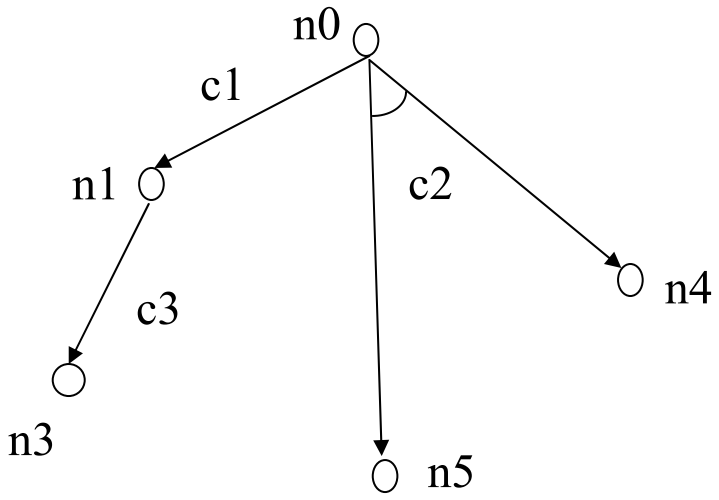
$$k(n0, N) = C1+K(n1,N)$$

$$\approx 1+h(n1)$$

$$k(n0, N) = C2+K(n4,N)+K(n5,N)$$

$$\approx 2+h(n4)+h(n5)$$

- 扩展一个部分解图后, 要重新计算这个解图费用的估计值;



$$k(n0, N) = C1+K(n1,N)$$

$$\approx 1+h(n1)$$

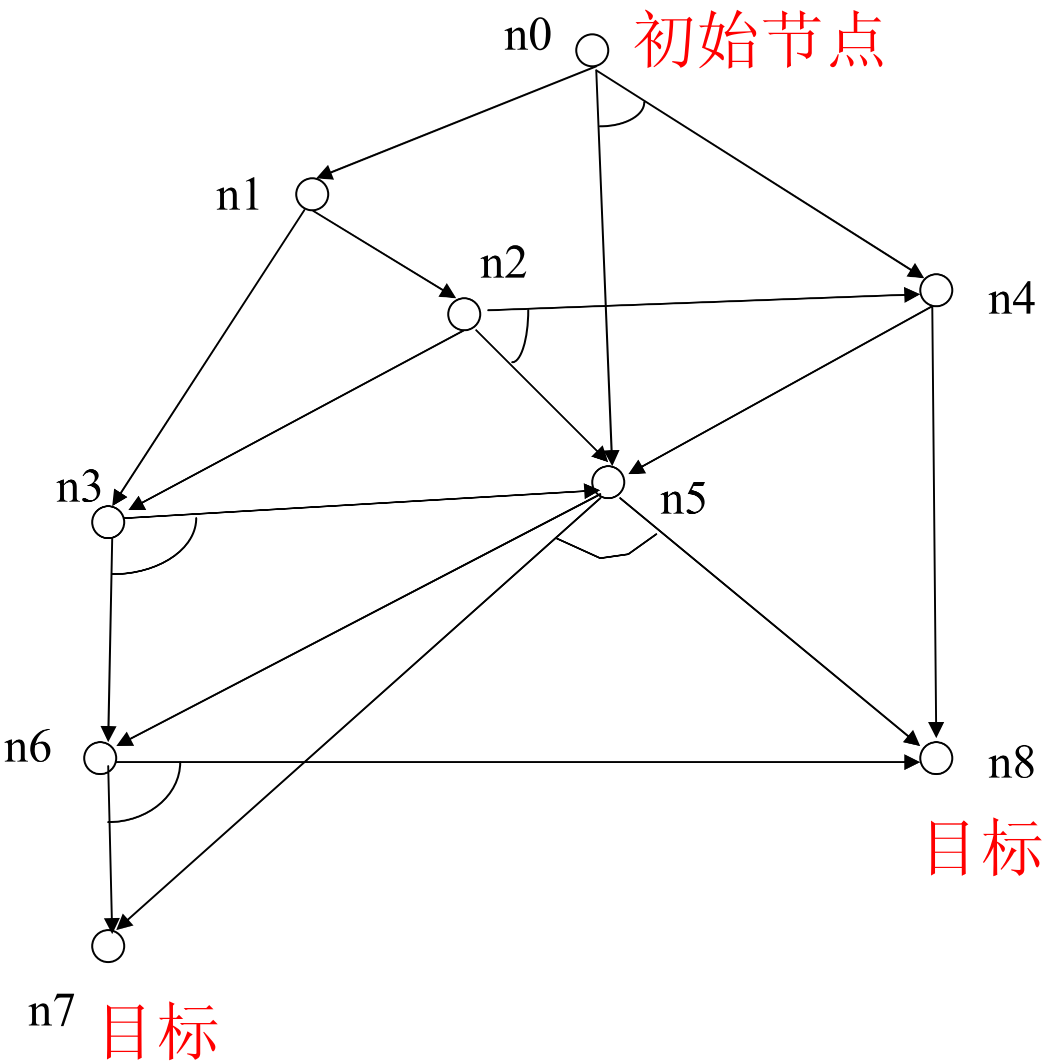
$$k(n0, N) = C1+ C3+ K(n3,N)$$

$$\approx 2+h(n3)$$

- 两个过程
  - 图生成过程，即扩展节点
  - 计算解图费用的过程
- 特点
  - 用标记来追踪解图

1. 建立一个只有初始节点**S**构成的搜索图**G**，设**S**的费用为$q(S)= h(S)$。如果**S**是目标节点，则标记为**SOLVED**。
2. 直到**S**被标记为**SOLVED**，
   a) 通过跟踪从**S**出发的有标记的连接符计算部分解图**G'**；
   b) 选择**G'**中的一个非目标节点**n**；
   c) 扩展**n**，生成**n**的所有后继，并把他们加到图**G**中；对于每一个未曾出现在**G**中的后继n_j，设其费用$q(n_j) = h(n_j)$。如果这些节点中有目标节点，则用**SOLVED**标记；
   d) 建立一个只有一个节点**n**的节点集合**W**；
   e) 直到**W**为空；
      i. 从**W**中删除节点**m**，并保证**m**的后裔不出现在**W**中；
      ii. 按以下的步骤修改**m**的费用**q**（**m**）：
         对于每一个从**m**出发的指向节点集合$\{n_{1i}, \ldots\ldots, n_{ki}\}$的连接符，计算$q_i(m)=c_i+q(n_{1i})+\ldots\ldots+q(n_{ki})$，这里$q(n_{ji})$或者是本循环内部计算出来的值，或者是**c**）中指定的值。
         设$q(m)$是$q_i(m)$中的最小者，标记实现这个最小值的连接符。如果本次标记与以前的不同，抹去以前的标记；如果这个连接符指向的所有后继节点都标记了**SOLVED**，则把**m**标记为**SOLVED**。
      iii. 如果**m**标记为**SOLVED**或者**m**的**q**值被修改，则把**m**的通过标记连接的所有父亲加到**W**中。

AO*算法举例



其中：

$h(n0)=3$

$h(n1)=2$

$h(n2)=4$

$h(n3)=4$

$h(n4)=1$

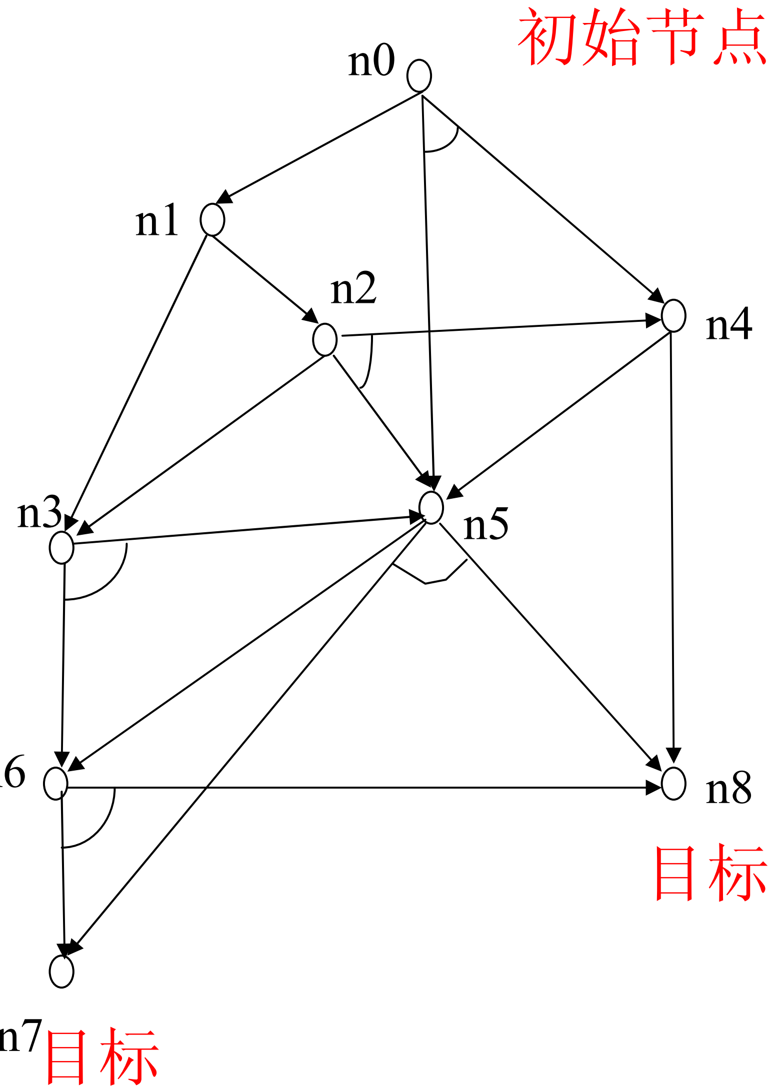
$h(n5)=1$

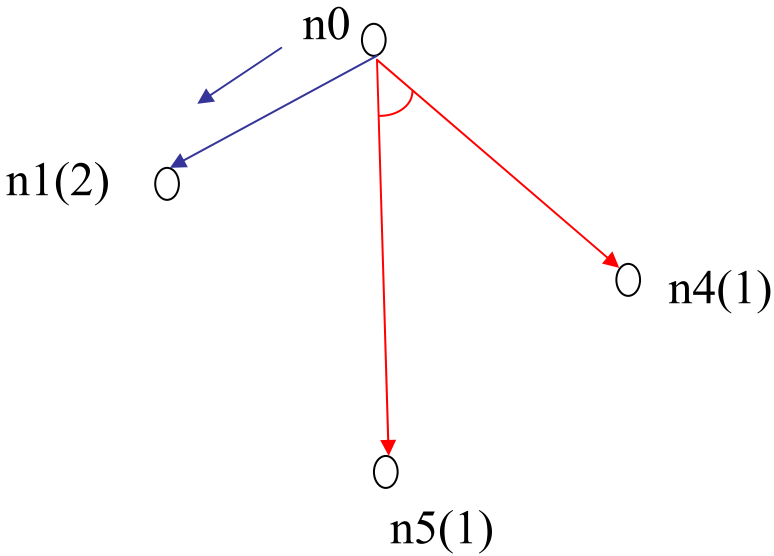
$h(n6)=2$

$h(n7)=0$

$h(n8)=0$

设：K连接符的耗散值为K

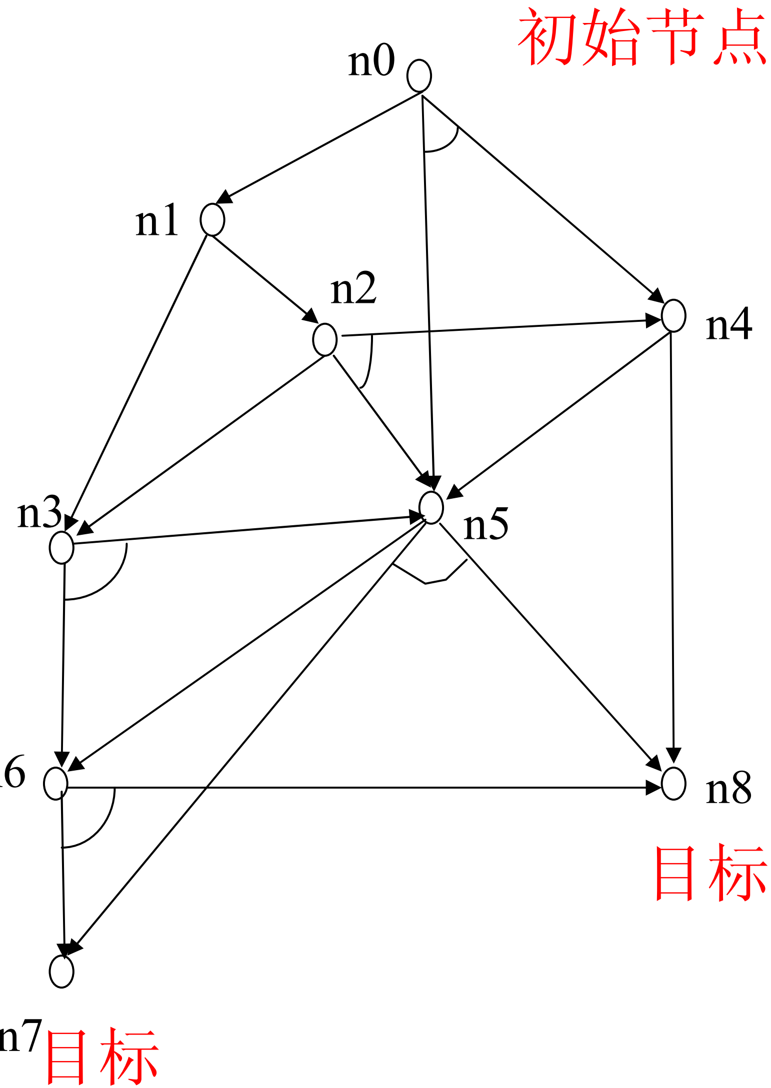


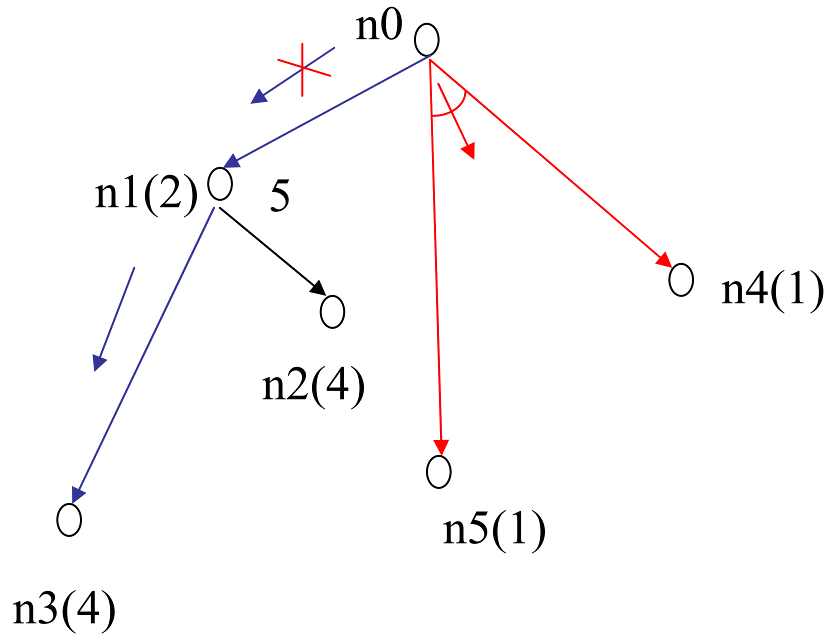


$W=\{n0\}$

$q_1(n0)=1+q(n1)=1+2=3$

$q_2(n0)=2+q(n4)+q(n5) =2+1+1=4$





W={n1}

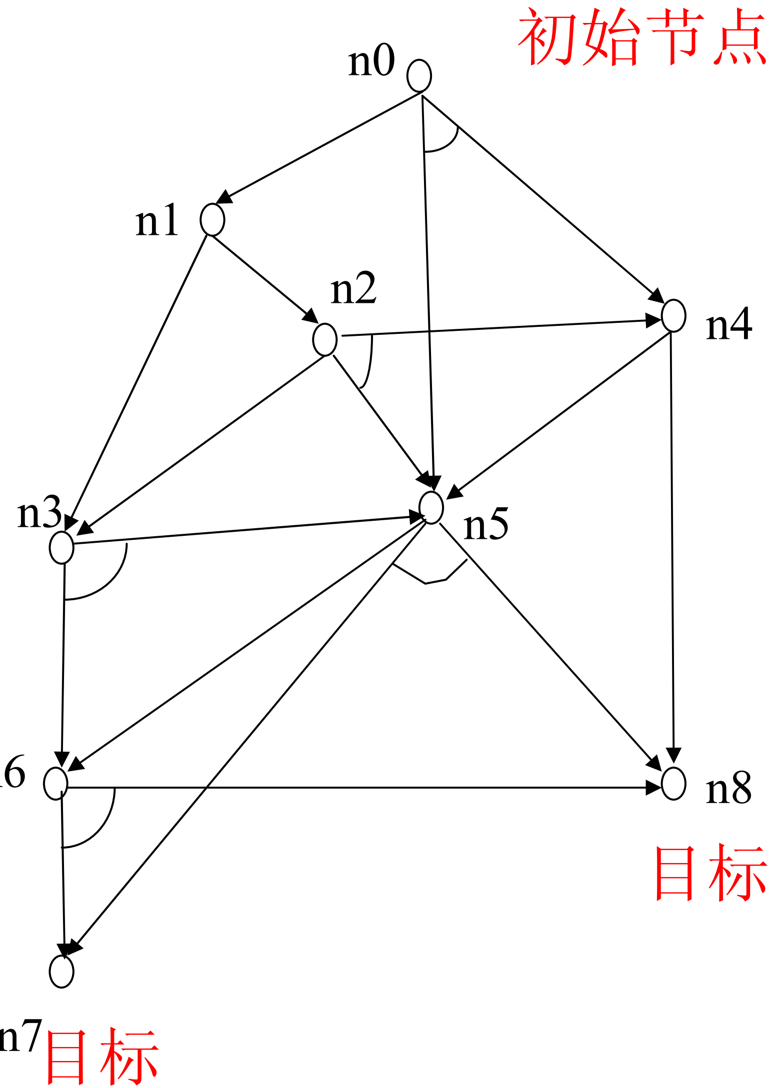
$q_1(n1)=1+q(n3)=1+4=5$

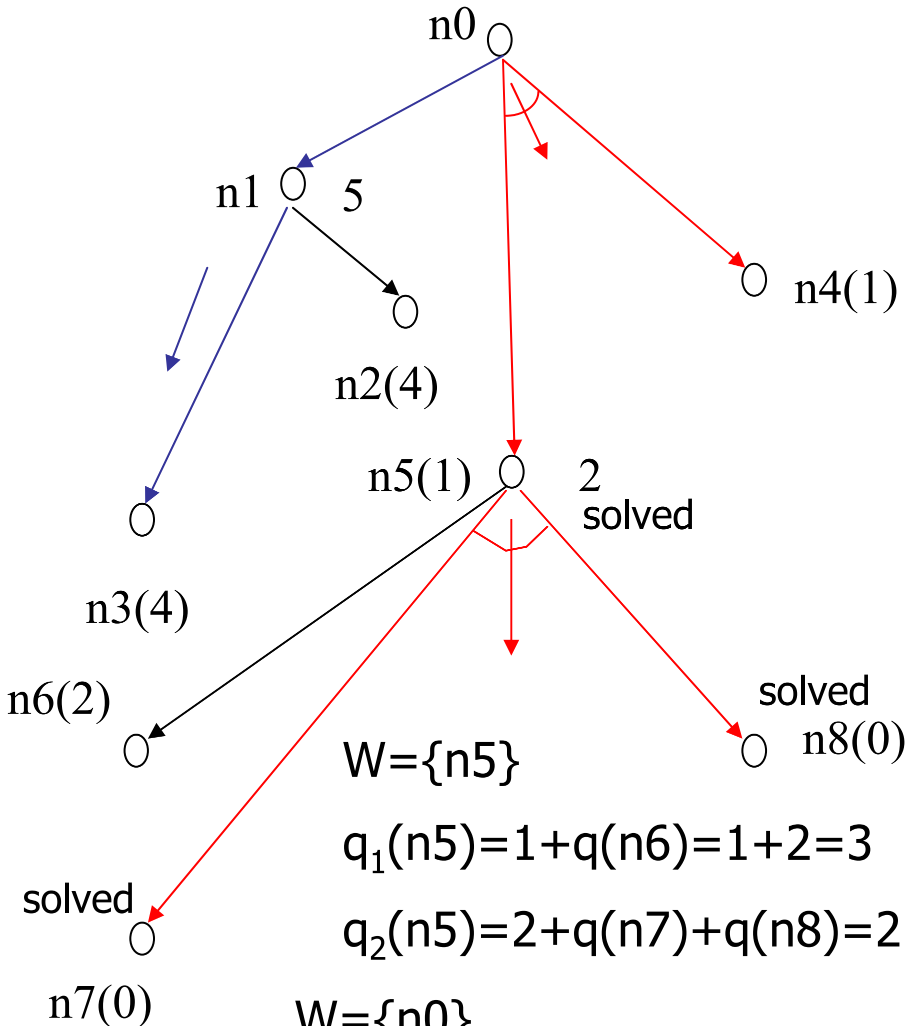
$q_2(n1)=1+q(n2)=1+4=5$

W={n0}

$q_1(n0)=1+q(n1)=1+5=6$

$q_2(n0)=2+q(n4)+q(n5)=2+1+1=4$





W={n5}

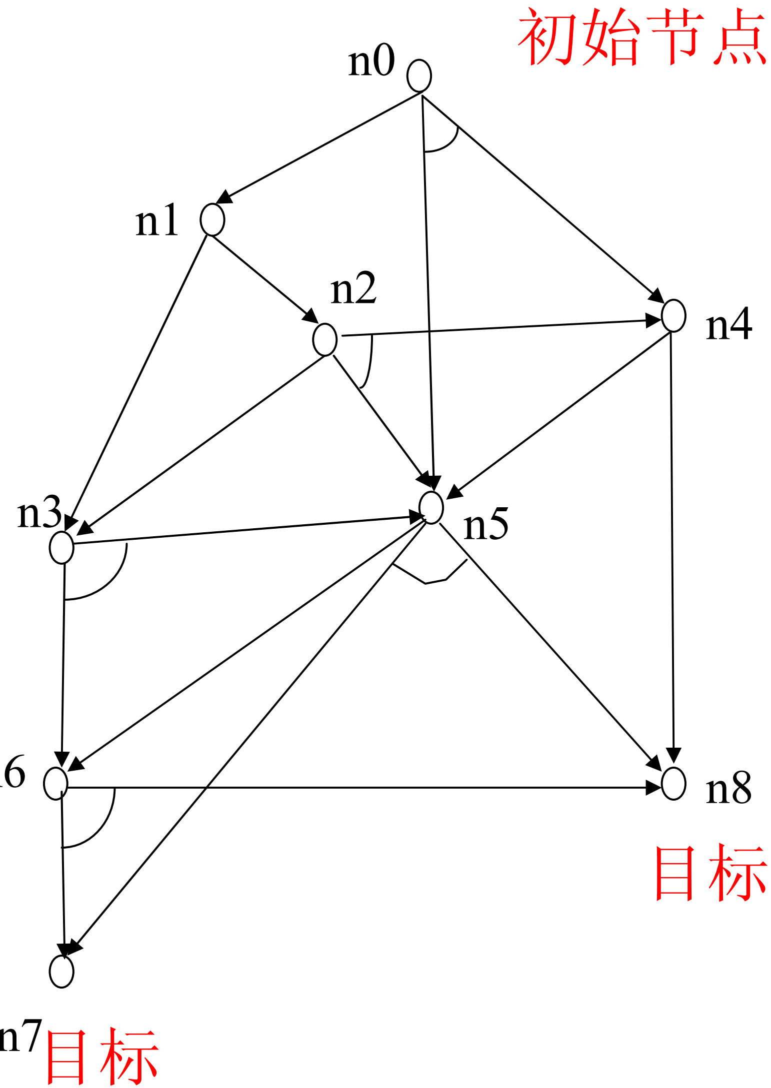
$q_1(n5)=1+q(n6)=1+2=3$

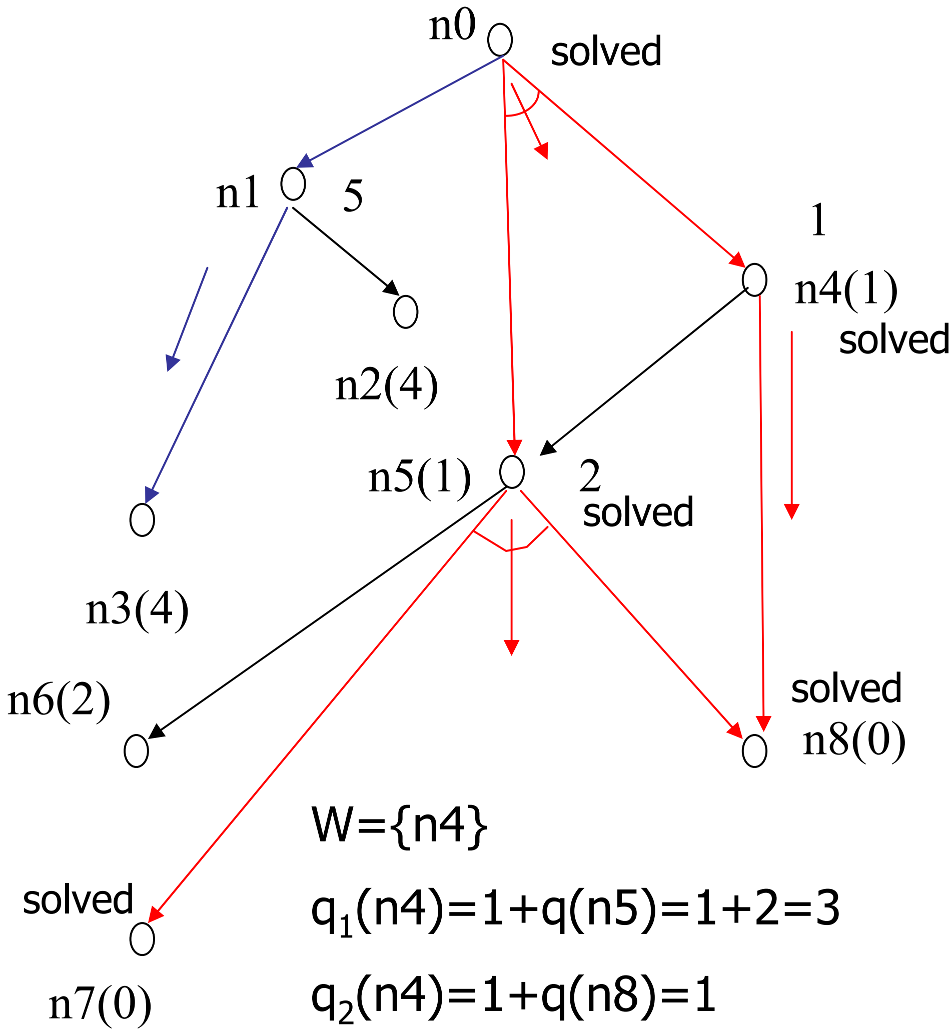
$q_2(n5)=2+q(n7)+q(n8)=2$

W={n0}

$q_1(n0)=1+q(n1)=1+5=6$

$q_2(n0)=2+q(n4)+q(n5)=2+1+2=5$





$W=\{n4\}$

$q_1(n4)=1+q(n5)=1+2=3$

$q_2(n4)=1+q(n8)=1$

$W=\{n0\}$

$q_1(n0)=1+q(n1)=1+5=6$

$q_2(n0)=2+q(n4)+q(n5)=2+1+2=$

- 如果一个**AND/OR**图存在解图，并且对所有节点都有**h(n)≤h*(n)**，并且**h**满足单调性限制，则 **AO***是可采纳的。

- 当**h(n)=0**时， **AO***就是宽度搜索。

- **A***算法的单调性
- 可分解产生式系统的搜索**(AO*)**
- 博弈树

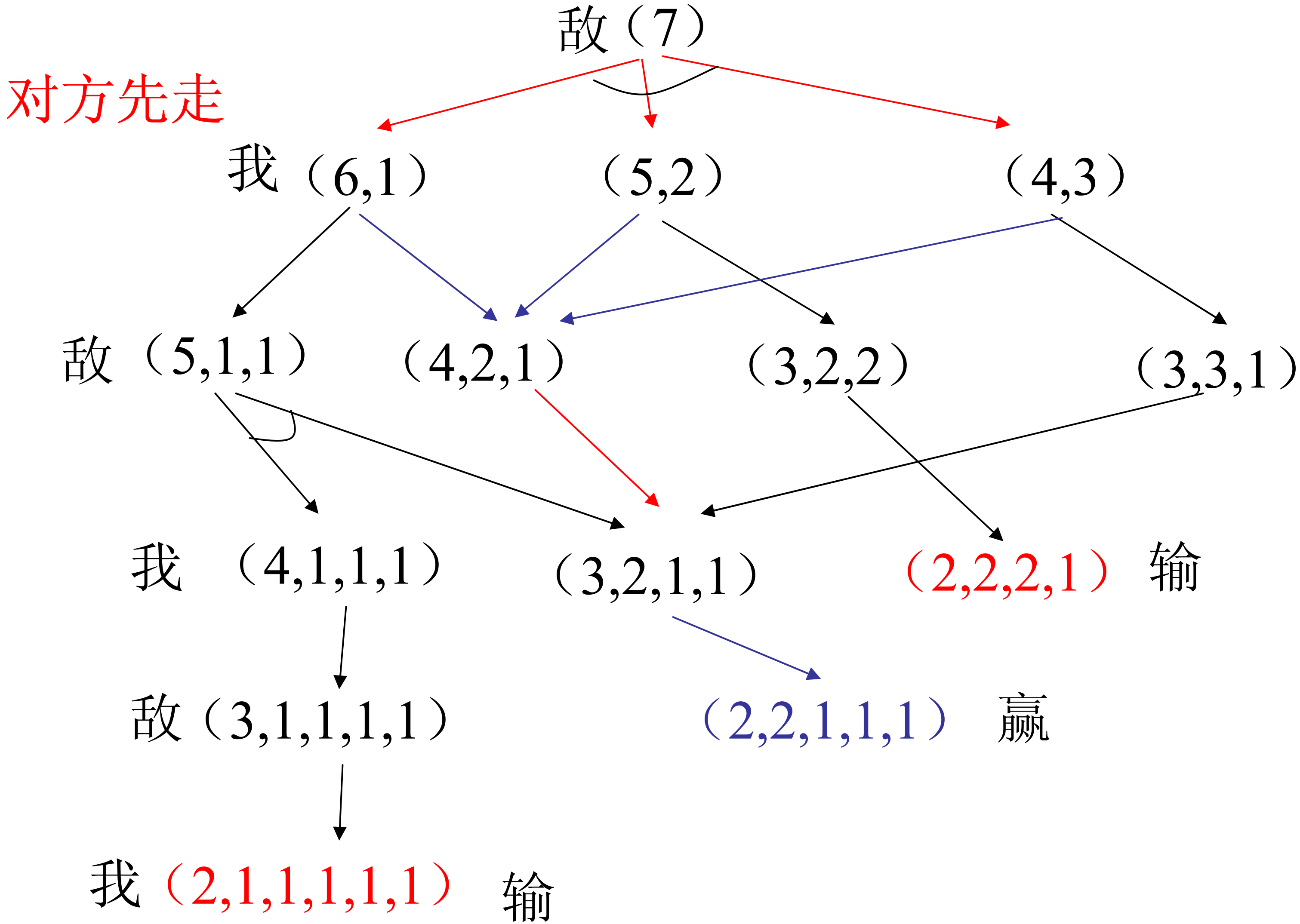
博弈树(game tree)搜索

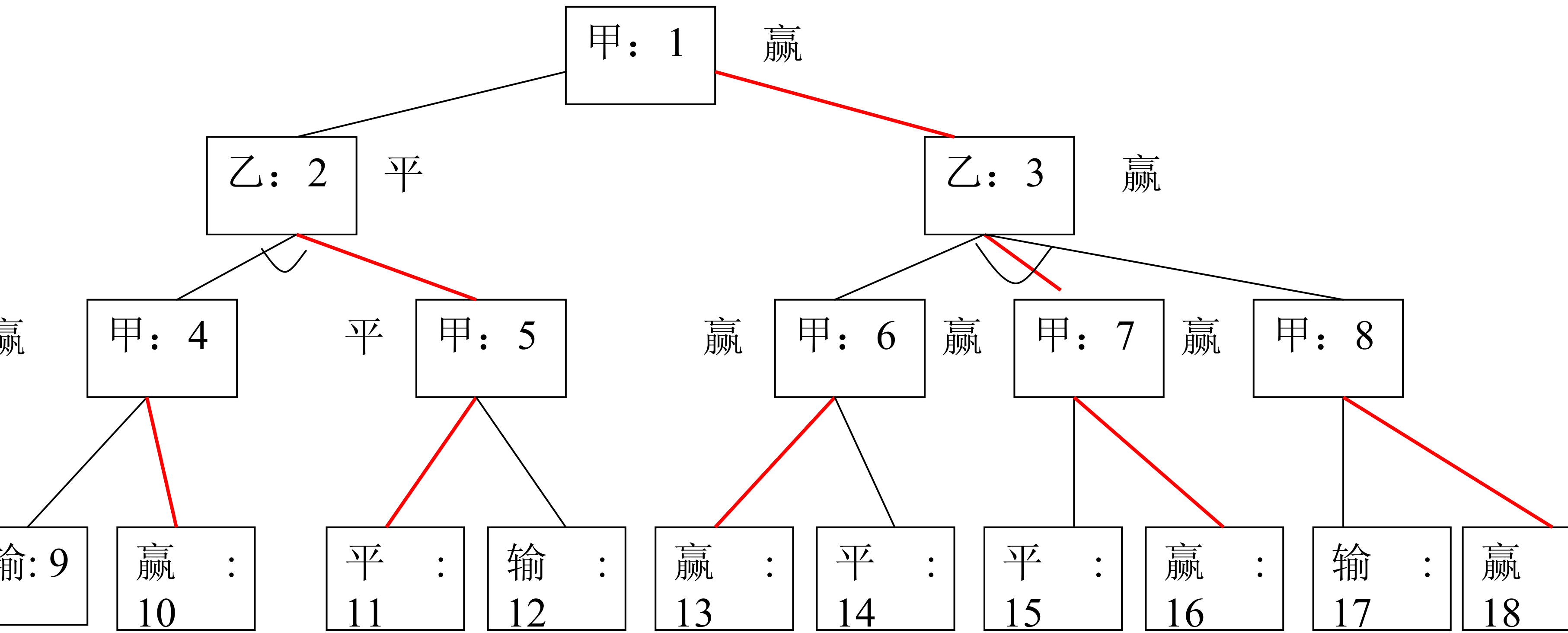
- 博弈一向被认为是富有智能行为的游戏，因而很做就收到**AI**界的重视，**1960s**年代就出现了若干博弈程序。
- 对于单人博弈，可用一般的搜索技术进行求解。对于双人完备信息博弈问题，可以用与或图来描述，因而搜索过程就是寻找最佳解图的问题。

- 博弈问题
  - 参与搜索的不只是一个主体，而包括对抗的多方(对弈)
  - 任何一方在选择自己的行为时，都要将对方可能采取的反应考虑在内
  - 由此而产生的搜索树称为博弈树(博弈图)

- 博弈问题可以用产生式系统的形式描述，综合数据库（各种布局)、规则（棋子的合法走步）、目标（将被吃）。规则作用与数据库的结果便生成博弈树。
- 下棋，军事仿真，**Agent**交互;
- 双人**(two players)**
- 一人一步
- 双方信息完备
  - 每一方不仅都知道对方过去已经走过的棋步，而且还能估计出对方未来可能的走步.
- 零和

分钱币问题（**Grundy**博弈）：有一堆树数目为**N**的钱币，两个选手轮流进行分堆，每一个选手每次只能把其中某一堆分成数目不等的两小堆，无法把钱分成不相等的两堆者认输。





甲：1
赢
乙：2
平
乙：3
赢
赢
甲：4
平
甲：5
赢
甲：6
赢
甲：7
赢
甲：8
输: 9
赢：10
平：11
输：12
赢：13
平：14
平：15
赢：16
输：17
赢：18

极小极大原则(minimaxing)

- 如果把”赢”标记为**+1**, ”平”标记为**0**, ”输”标记为**-1**，则
  - 在甲方节点，即或节点，选择极大值；
  - 而在乙方节点，即与节点，在假定乙方一定选择正确的着法的情况下，选择极小值；
- 这称为极小极大原则（即，在极小中取极大）。

穷举法

- 中国象棋
  - 一盘棋平均双方各走**40**步，一盘棋平均双方各走**50**步，收缩的位置共有**:** $(40^2)^{50}$，深度达**100**层，总节点数约为10^{161}。
  - 假设**1**毫微秒生成一个节点，约需10^{145}年。
- 结论：不可能穷举。

- 搜索必须适可而止，到达一定深度就不往下走.
- 不是根据实际的输赢来评分，而是对每个节点给出估计值, 根据估计值选择节点.
- 静态估值函数

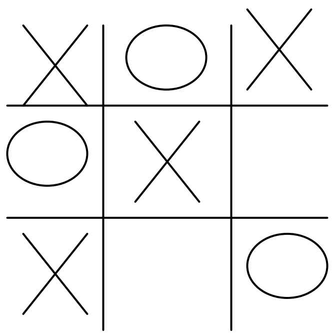
如何设计静态估值函数

- 对叶节点:
  - ”赢”: **+**∞
  - ”平”: **0**
  - ”输”: **-**∞

- 对其他节点
  - 定义参数，根据参数决定估计值;
  - 参数：例如子力强弱，子力分布，子力配合
  - **Samuel**的跳棋程序
    - **20-30**个参数。

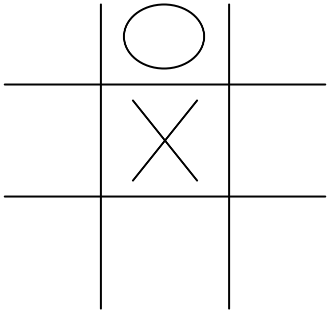
例子

- 一字棋(九格棋)
  - 两选手轮流下棋，每次摆各自一个旗帜，谁先取得三子一线的结果就取胜。

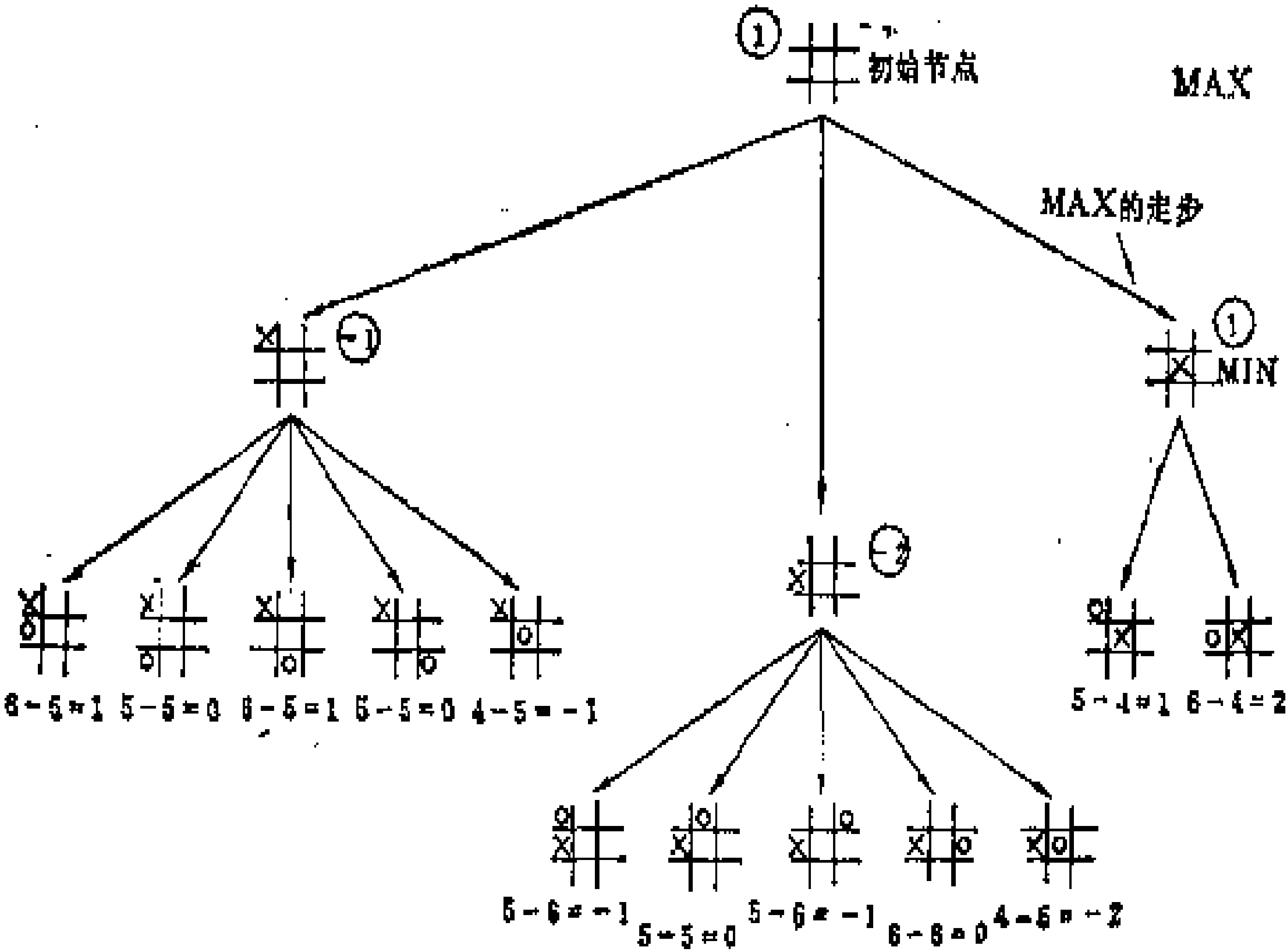


- **f(n):**
  - 甲胜 **: f(n)= +∞**
  - 乙胜 **: f(n)= -∞**
  - 都不胜**:**

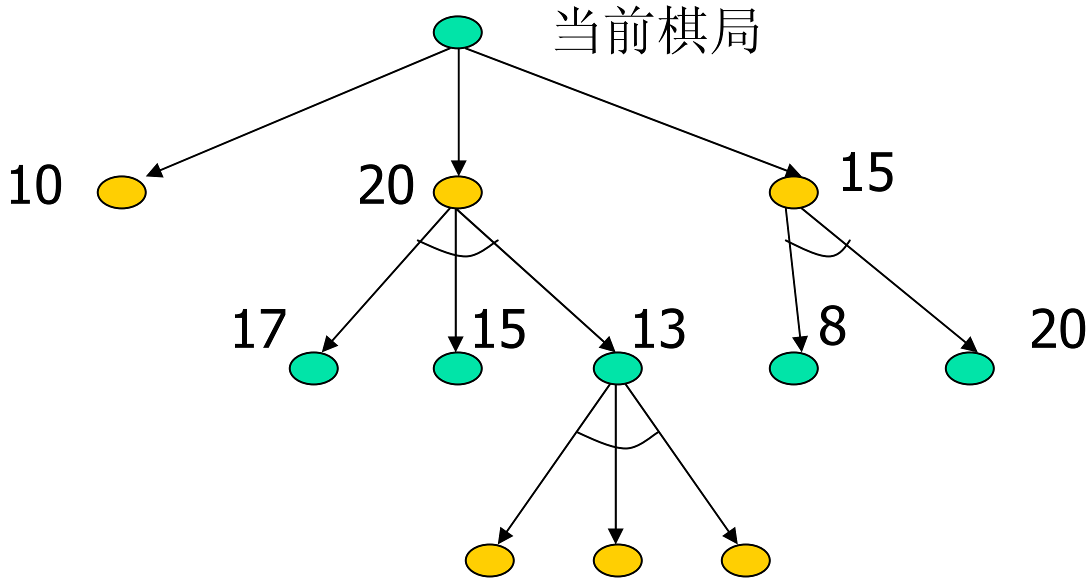
f(n) = 甲方可能成一字的数目 – 乙方可能成一字的数目



f(n) = 6 - 4 = 2



1
初始节点
MAX
MAX的走步
-1
1
MIN
-2
6-5=1
5-5=0
6-5=1
5-5=0
4-5=-1
5-4=1
6-4=2
5-6=-1
5-5=0
5-6=-1
6-6=0
4-6=-2


当前棋局
10
20
15
17
15
13
8
20

α 剪枝和 β 剪枝

- **MINMAX**过程是把搜索树的生成和格局估计值这两个过程分开进行，先生成全部搜索树，然后再进行端点静态估计值和倒退值计算。

- 为避免节点生成过多，有两种博奕树特有的减少节点的方法：α 剪枝和 β 剪枝。
  - α 剪枝：（针对甲方而言）
  - β 剪枝：（针对乙方而言）

- α 剪枝：若任一极小层节点的β值小于或等于它任一先辈极大值层节点的α值，α**(**先辈层**)>=** β **(**后继层**)**，这可以中止该极小值层中这个**MIN**节点以下的搜索过程，这个**MIN**节点最终的倒推值就确定为这个值。

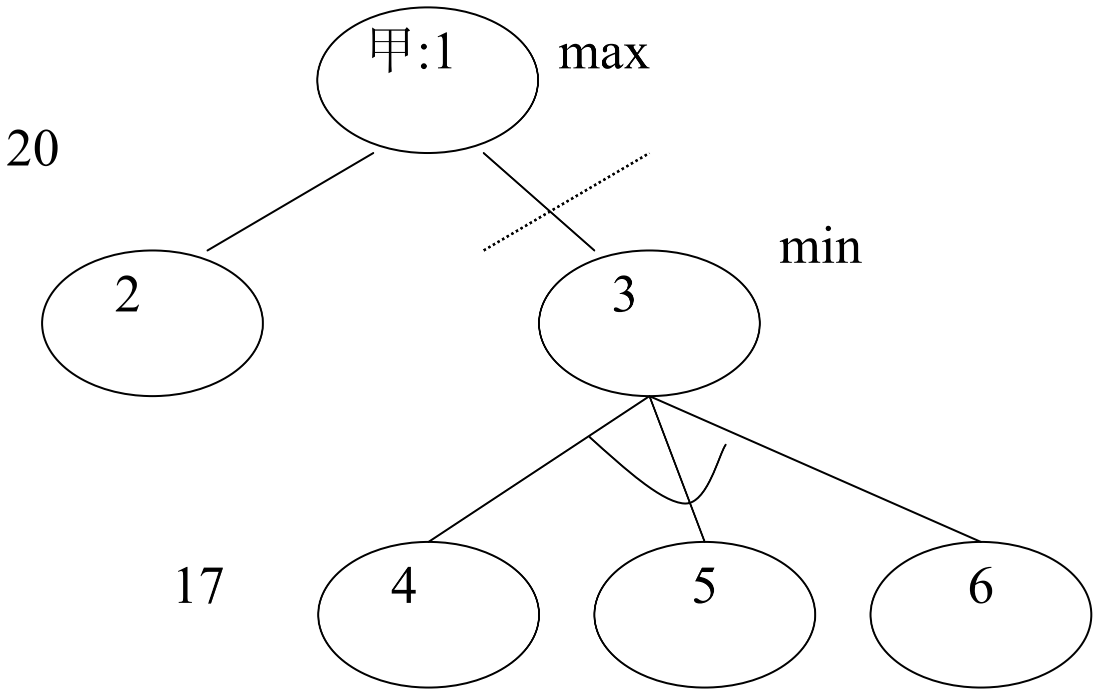
- β 剪枝：若任一极大层节点的α 值大于或等于它任一先辈极小值层节点的 β值，α**(**后继层**)>=** β **(**先辈层**)**，这可以中止该极大值层中这个**MAX**节点以下的搜索过程，这个**MAX**节点最终的倒推值就确定为这个值。

- 只要在搜索过程记住倒退值的上下界并进行比较，就可以实现修剪操作，这种操作为α剪枝。
- 极大层取下界α ，倒推至决不会比β更小。

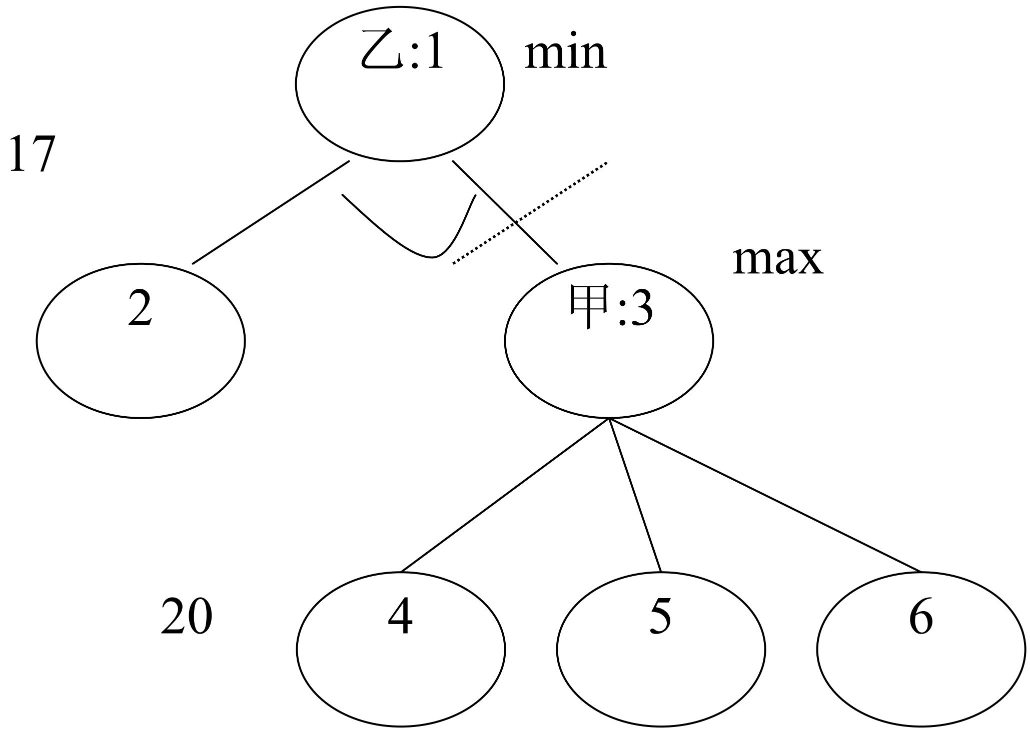
- 类似地，可以定义β剪枝
- 极小层取上界β ，倒推至决不会比α更大。

- 在实际修剪过程中， α 、 β还可随时修正，但是极大层的倒推值下界α永远不下降，实际的倒推值取其后继节点最终确定的倒推值中最大的一个倒推值。

α 剪枝

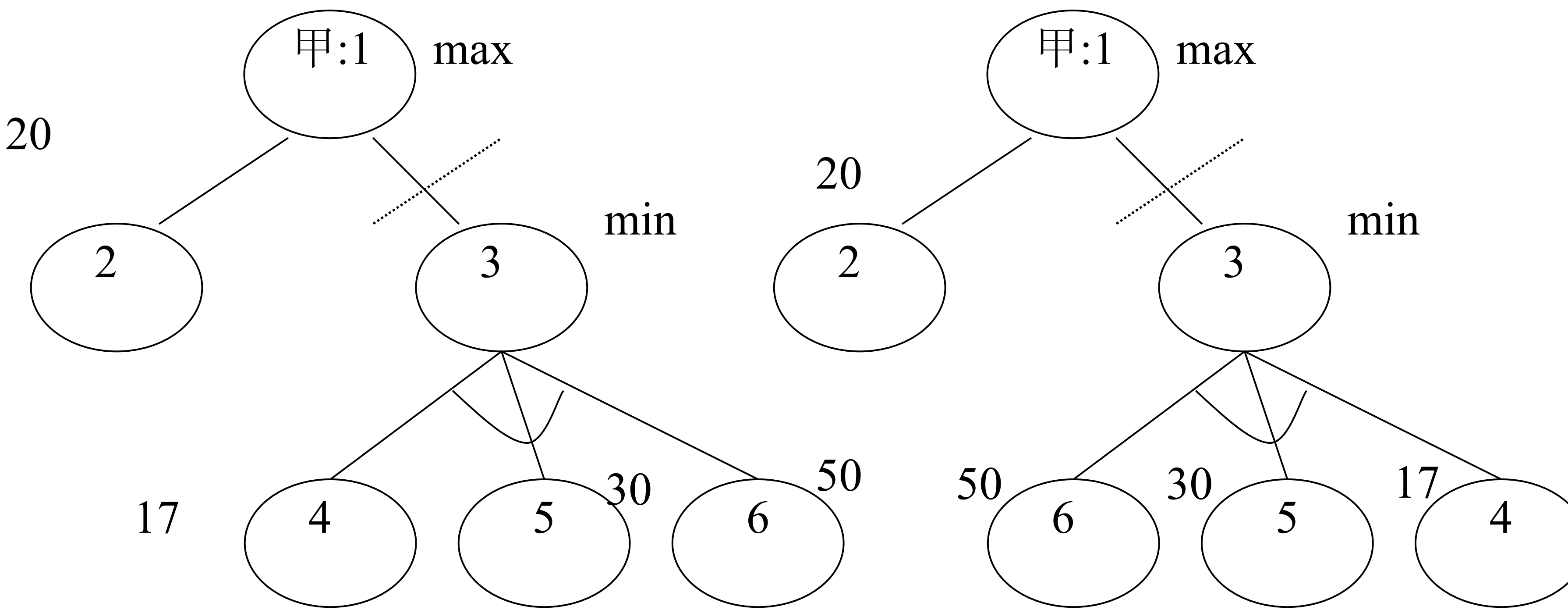


β 剪枝



博弈树算法的缺点

- 博弈树算法的缺点：
  - α / β 剪枝的效率与节点出现的顺序有很大关系。



- 计算估计值时，假设双方使用同一评估函数是不合适的（甲重视实，不等于乙也重视实）。

- 由于不能穷举所有节点，在算法中为了保证搜索效率，一般设置一个搜索深度，达到这个深度就停止搜索了, 这会很不合理。
- 典型的例子是对子。
  - 水平负效应: 当前有利, 真正不利
  - 水平正效应: 当前不利, 真正有利
  - A型程序和B型程序

- **A**型程序
  - 为每个节点配备一个静态估值函数，并规定搜索的最大深度**,** 每走一步即重新搜索一次**.**
- **B**型程序
  - 搜索深度不固定**,** 要找到一个合理的终止点才暂停搜索**.**
  - 静止期**:** 当搜索前沿的节点的估计值在搜索向前延伸时有激烈变化现象时，不应停止搜索，应坚持把搜索进行到一个相对静止的时期.