第四章 图搜索策略

图搜索策略是一种在图中寻找解路径的方法。

（1）用树结构

优点：控制简单。

缺点：时空均需要较大的代价。

（2）用图结构

优点：节省大量空间和时间。

缺点：控制更复杂，判断也要占用时间。

图的分类

根据图对应的实际背景，可分为：

- **或图**：搜索扩展时，可在若干分支中选择其中之一。
- **与/或图**：搜索扩展时，有可能要同时搜索若干分支，也有可能在若干分支中选择其中之一。

4.1 图搜索策略

图搜索算法只记录状态空间那些被搜索过的状态，它们组成一个搜索图G。G由两张表内的节点组成：

Open表：用于存放已经生成，且已用评价函数作过估计或评价，但尚未产生它们的后继节点的那些结点，也称未考察结点。

Closed表：用于存放已经生成，且已考察过的结点。

一个结构Tree，它的节点为G的一个子集。Tree 用来存放当前已生成的搜索树，该树由G的反向边组成。

通用图搜索算法

设S_0：初态，S_g：目标状态

1. 产生一仅由S_0组成的open表；

2. 产生一空closed表；

3. 如果open为空，失败退出；

4. 在open表上按某一原则选出第一个优先结点，称为n，放n到closed表中，并从open表中去掉n；

通用图搜索算法

5. 若$n \in S_g$，则成功退出；

解为在Tree中沿指针从n到s_0的路径，或n本身。

(如八皇后问题给出n即可，八数码问题要给出路径）

6. 产生n的一切后继，将后继中不是n的前驱点的一切点构成集合M，将装入G作为n的后继，这就除掉了既是n的前驱又是n的后继的结点，避免了回路,节点之间有偏序关系存在。

通用图搜索算法

7. 对M中的元素P，分别作两类处理：

7.1 若P$\notin$G，即P不在open表中也不在closed表中，则P加入open表[注1]，同时加入搜索图G中，对P进行估计放入Tree中。

7.2 P$\in$G，则决定是否更改Tree中P到*n*的指针[注2]。

8. 转3。

通用图搜索算法

注1：

若生成的后继节点放于：

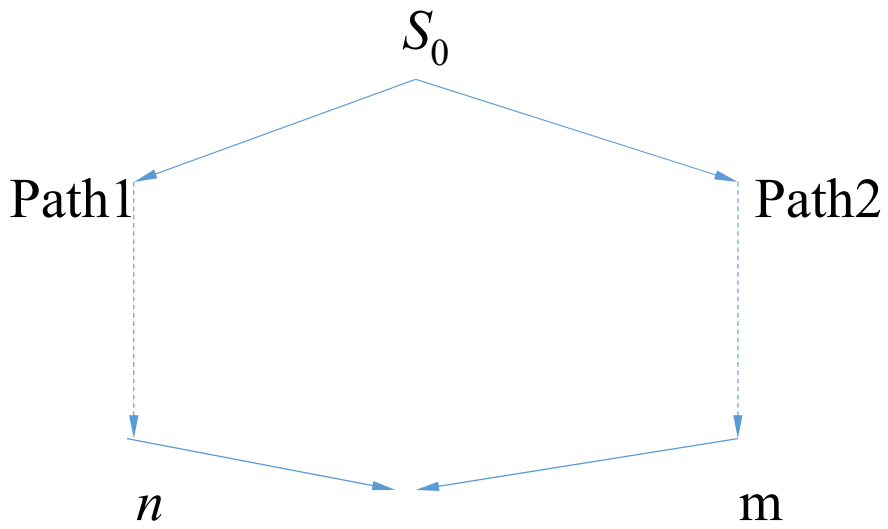
（1）Open表的尾部——相当于Breadth-first-search；

（2）Open表的首部——相当于Depth-first-search；

（3）根据评价函数f的值确定最佳者，放于Open表的首部——相当于Best-first-search。

说 明

(1) 若P∈M且在open表中，这说明P在*n*之前已是某一结点m的后继，但本身尚未被考察(未生成P的后继)。



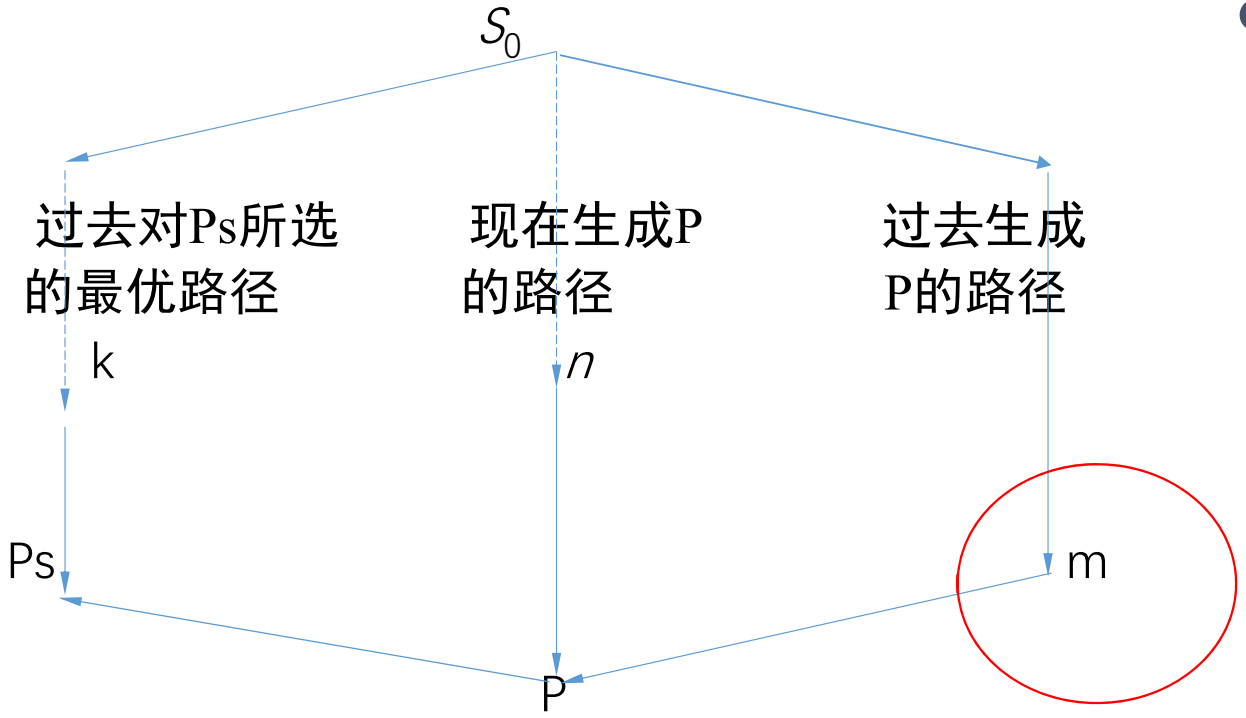
P在*n*之前已是某一结点m的后继

这说明从S_0→p至少有两条路径，这时有两种情况：

- 若Path1的代价<Path2的代价时，当前路径较好，要修改p的指针，使其指向*n*，即标出搜索之后的最好路径；
- 若Path1的代价≥Path2的代价时，原路径较好，不改变p的指针。

说 明

(2）若p∈M且在closed表中，这说明:

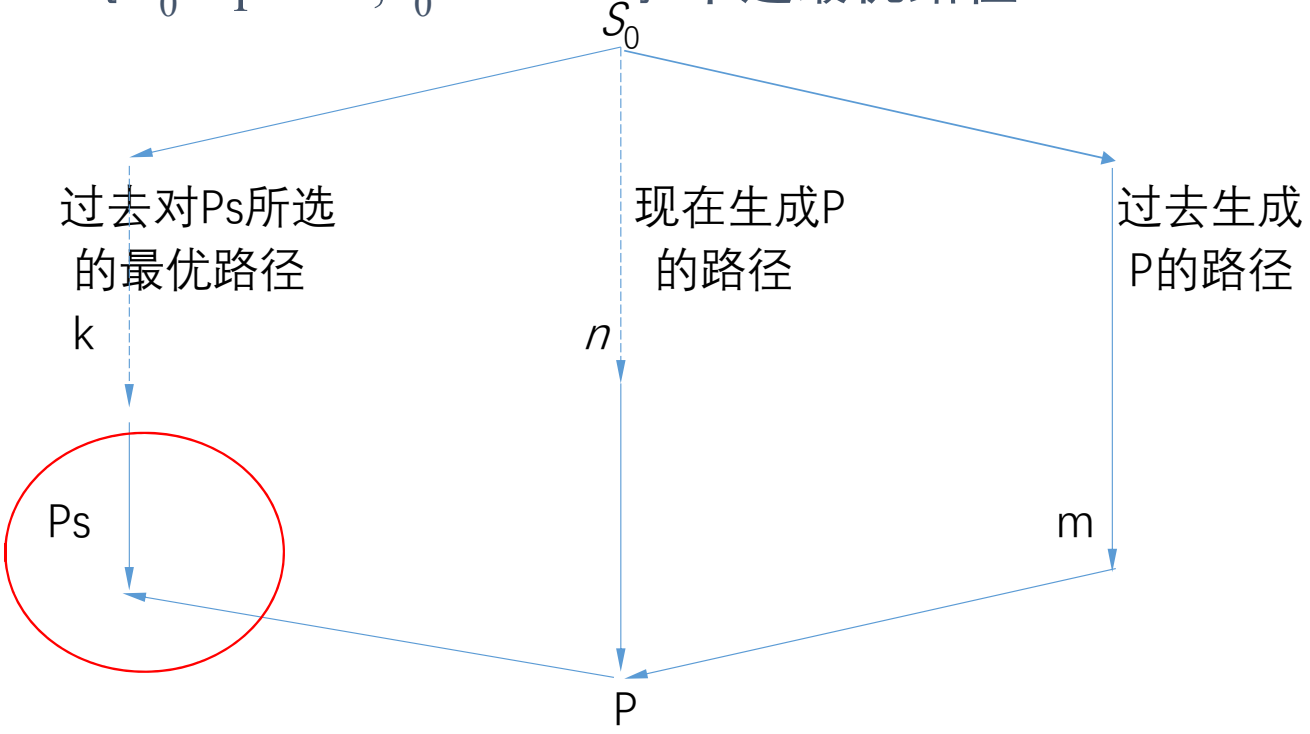


p∈M且在closed表中时不同的最优路径

- p在n之前已是某一节点m的后继，所以需要作如(1)同样的处理。
- 即过去对$S_0 \rightarrow P$而言的最优路径为$S_0 \rightarrow m \rightarrow p$，现在要从$S_0 \rightarrow m \rightarrow p$与$S_0 \rightarrow n \rightarrow p$中求最优路径。

说 明

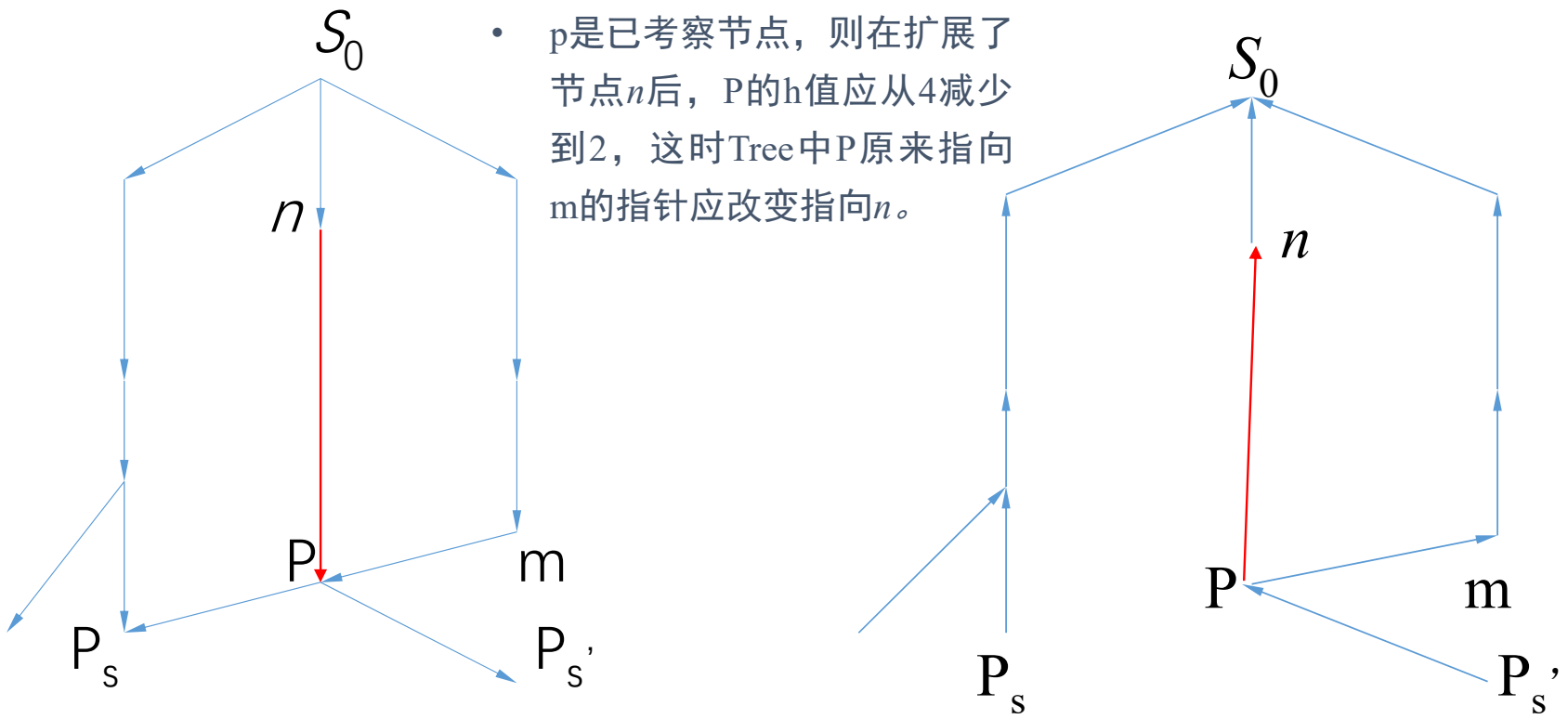
- 同理，若过去对$S_0 \rightarrow$Ps而言的最优路径为$S_0 \rightarrow$k$\rightarrow$Ps，现在要从$\{S_0 \rightarrow p \rightarrow Ps, S_0 \rightarrow k \rightarrow Ps\}$中选最优路径。



p∈M且在closed表中时不同的最优路径

- p在closed表中，说明p的后继也在n之前已生成，我们称为Ps，那么对Ps同样可能由于n→p这一路径的加入又必须比较多条路径代价后而取代价小的一条。

例4.1 设当前搜索图和搜索树分别为：P有两个后继结点



- p是已考察节点，则在扩展了节点*n*后，P的h值应从4减少到2，这时Tree中P原来指向m的指针应改变指向*n*。

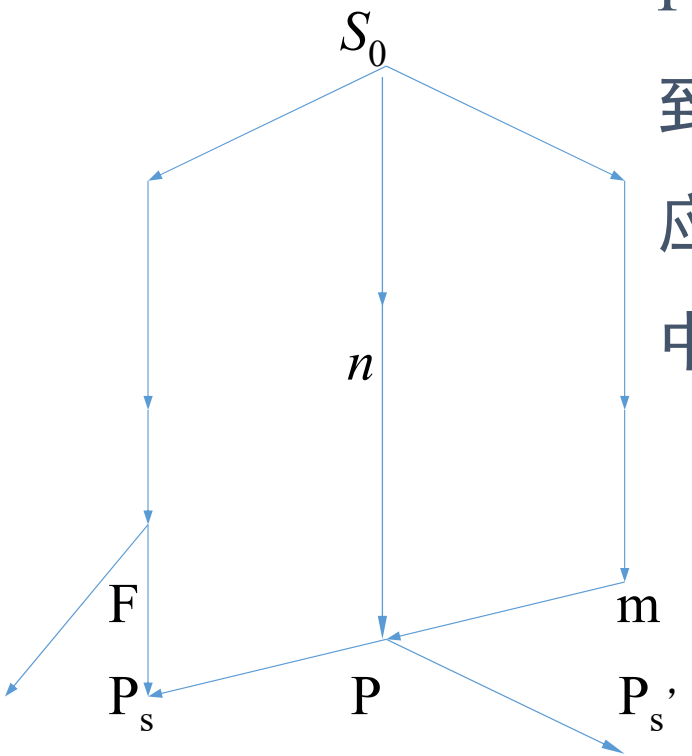
当前搜索图和搜索树

若启发式函数h(n)为从起点S_0到节点n的最短路径的长度，该长度用边的数目表示，当前扩展的节点是搜索图中的n，设p是n的后继。

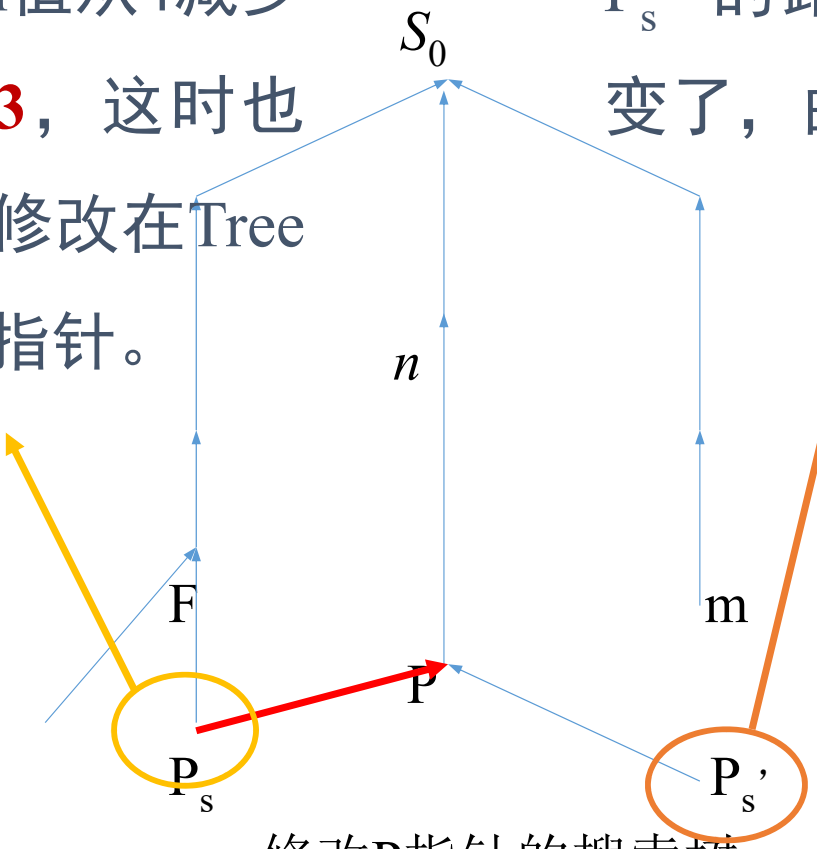
P的指针改变后，

P_s的h值从4减少到了**3**，这时也应该修改在Tree中的指针。

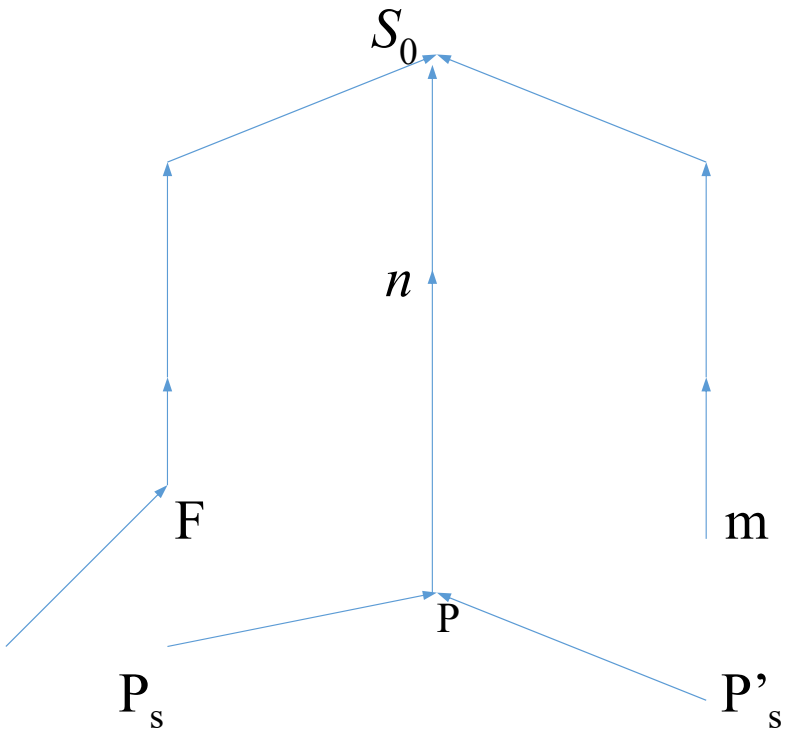
P_s' 的路径自动改变了，由5减少到**3**。



扩充n 的搜索图



修改P指针的搜索树



P的指针改变后，Tree中指针的修改

A 算法

通用图算法在采用如下形式的估计函数时, 称为A算法。

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

- $g(n)$表示从S_0到n点费用的估计，因为n为当前节点，搜索已达到n点，所以$g(n)$可计算出。
- $h(n)$表示从n到S_g接近程度的估计，因为尚未找到解路径，所以$h(n)$仅仅是估计值。
- **f(n)** 作用是用来评估OPEN表中各节点的重要性,决定其次序。

A 算法

通用图搜索算法：

设S_0：初态，S_g：目标状态

1. 产生一仅由S_0组成的open表；

2. 产生一空closed表；

3. 如果open为空，失败退出；

4. 在open表上按某一原则选出第一个优先结点，称为n，放n到closed表中，并从open表中去掉n；

7. 对M中的元素P，分别作两类处理：

7.1 若P∉G，即P不在open表中也不在closed表中，则P加入open表[注1]，同时加入搜索图G中，对P进行估计放入Tree中。

7.2 P∈G，则决定是否更改Tree中P到n的指针[注2]。

8. 转3。

把通用图算法中的第（4）步按估计函数f的值的大小取出一个节点，或在（7）中的1步，以升序或降序排列open表然后根据f的值在某一位置加入一个节点。

例：用A算法解决水壶问题

每个节点的费用定义为：f(n)=g (n)+h (n)

其中g（n）是搜索树中搜索的深度

h(n) = 2　　如果0<x<4并且0<y<3

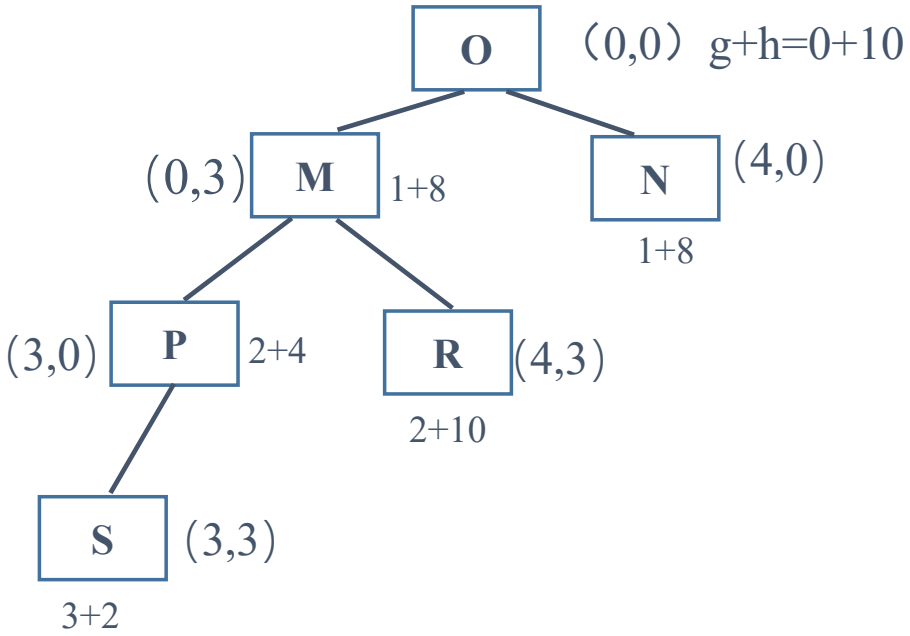
= 4　　如果0<x<4或者0<y<3

= 10　　如果x=0并且y=0

如果x=4并且y=3

= 8　　如果x=0并且y=3

如果x=4并且y=4



A*算法

在A算法的基础上，给出A*算法的定义：

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

- $g^*(n)$为S_0到n的实际最小费用
- $h^*(n)$为n到S_g的实际最小费用估计
- $f^*(n)$表示S_0经点n到S_g最优路径的费用，也有人将$f^*(n)$定义为实际最小费用。

f(n)和f(n*)进行比较

g（n）是g*（n）的估计，g（n）容易计算，但它不一定就是从起始节点S_0到n的真正最短路径的代价，很可能从初始节点S_0到节点n的真正最短路径还没有找到，所以一般都有g（n）≥g*（n）。

h（n）是对h*（n）的估计。

A*算法

有了g*（n）和h*（n）的定义，对A算法中的g（n）和h（n）做如下的限制：

（1） g（n）是g*（n）的估计，且g（n）>0.

（2） h（n）是h*（n）的下界，即对任意节点n均有h（n）≤h*（n）。

则称这样得到的算法为A*算法。

h（n）≤h*（n）的限制十分重要，它保证A*算法能够找到最优解。

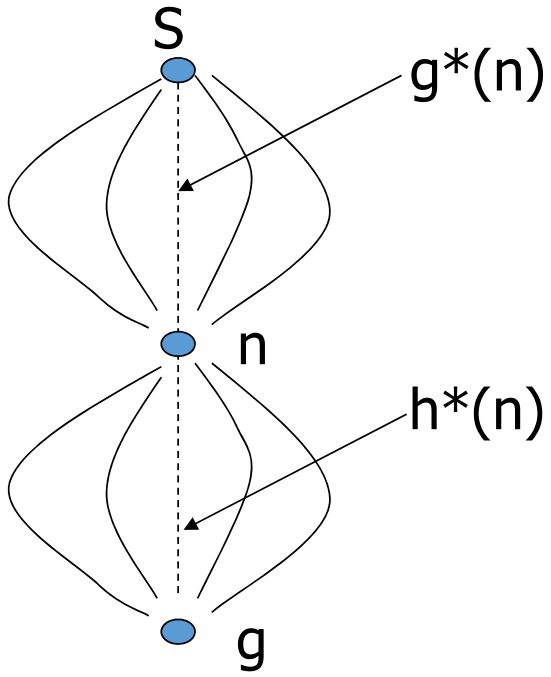
进一步的解释

h*是能计算出任意节点n到目标的最优代价的函数，称之为“完美启发式函数”，如果

∀n：h（n） ≤h*（n），则称h为可采纳（Admissible）的启发式函数。

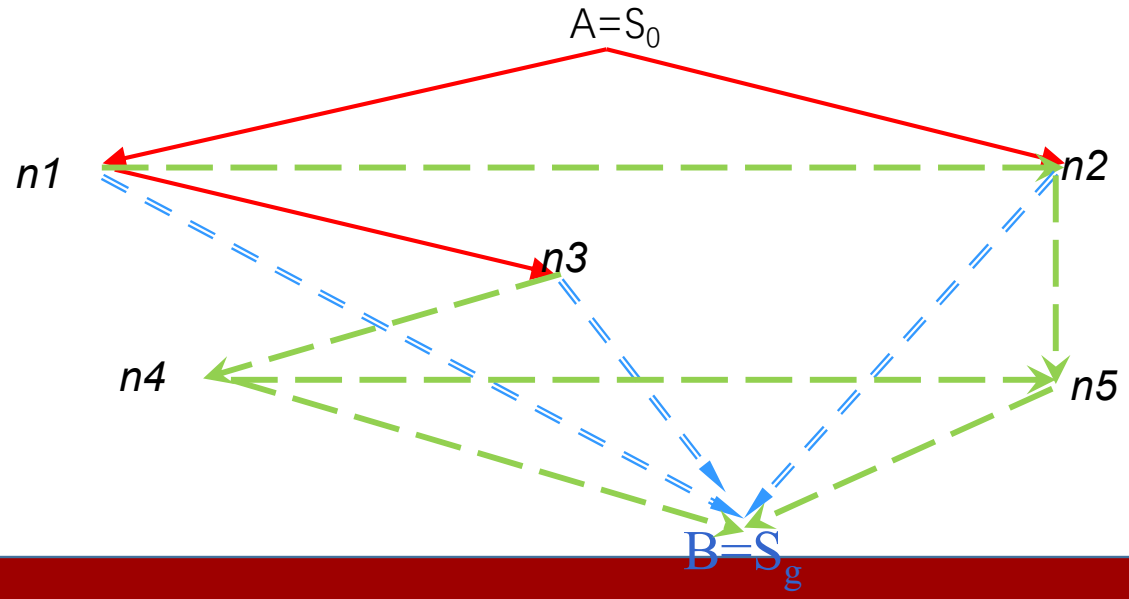
此外再引入函数g*，计算从开始节点到n的最优代价。

所以f*（n）就是从起始点出发经过节点n到达目标节点的最佳路径的总代价。



例：在地图上寻找城市A至B的最短路径

- 双虚线表示n_i与S_g的直线距离(可以从地图上量出），
- 虚线表示n_i与S_g的实际要走的路径，
- 实线表示从S_0出发已经走过的路径为$g(n)$;
- 则实线表示的路径为$g(n)$,虚线和双虚线表示都可作为h(n)。
- 以n3为例，g(n)={S_0->n1->n3}，h*(n)可以是{n3->n4->S_g}，{n3->n4->n5->S_g}，或{n3->S_g}（双虚线）



如果以双虚线表示路径定义为h(n)，显然有h(n）≤h*(n），g(n）≥g*(n）。所以使用这样的h(n)作为搜索算法的估计函数，这个算法就是A*算法。

进一步说明

在f(n)中若令：

- h(n)≡0，则A算法相当于一致代价搜索法。
- g(n)≡h(n)≡0，则相当于随机算法。
- g(*n*)≡0，则相当于贪婪最佳优先算法。
- h(n)≤h*(n)　就称为这种A算法为**A*算法**。

A*算法

设S_0：初态，S_g：目标状态

1. open=$\{S_0\}$；

2. closed=$\{\ \}$；

3. 如果open=$\{\}$，失败退出；

4. 在open表上取出f最小的结点n，n放到closed表中；其中：

$$f(n)=g(n)+h(n) \qquad h<=h^*$$

5. 若$n\in S_g$，则成功退出；

6. 产生n的一切后继，将后继中不是n的先辈点的一切点构成集合M

7. 对M中的元素P，分别作两类处理：

7.1 若P$\notin$G，则对P进行估计加入open表，记入G和Tree。

7.2 P$\in$G，则决定是否更改Tree中P到n的指针并且更改P的子节点n的指针和费用。

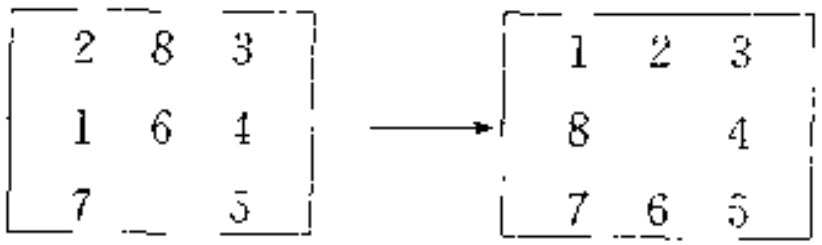
8. 转3。

A*算法的性质

A*算法与一般的最佳优先比较，有其特有的性质：

如果问题有解，即$S_0 \rightarrow S_g$存在一条路径，A*算法一定能找到最优解。这一性质称为**可采纳性**(admissibility）。

九宫格问题（同一个问题可以有多种设计方法）



采用A*算法，令$f(n) = d(n) + w(n)$。
其中$d(n)$为搜索树的深度。

假设h（n）= w(n)：放错位置数字个数

尽管我们不知道h*(n)具体为多少，但当采用单位代价时，通过对“不在目标状态中相应位置的数字个数”的估计，可以得出至少需要移动w（n）步才能够到达目标。显然，w（n）≤h*（n），因此它满足A*算法的要求。

九宫格问题

- 在搜索第二层中．若f值相等，需要再规定一下如何选择后继．如

  (1) 后生成的节点优先。

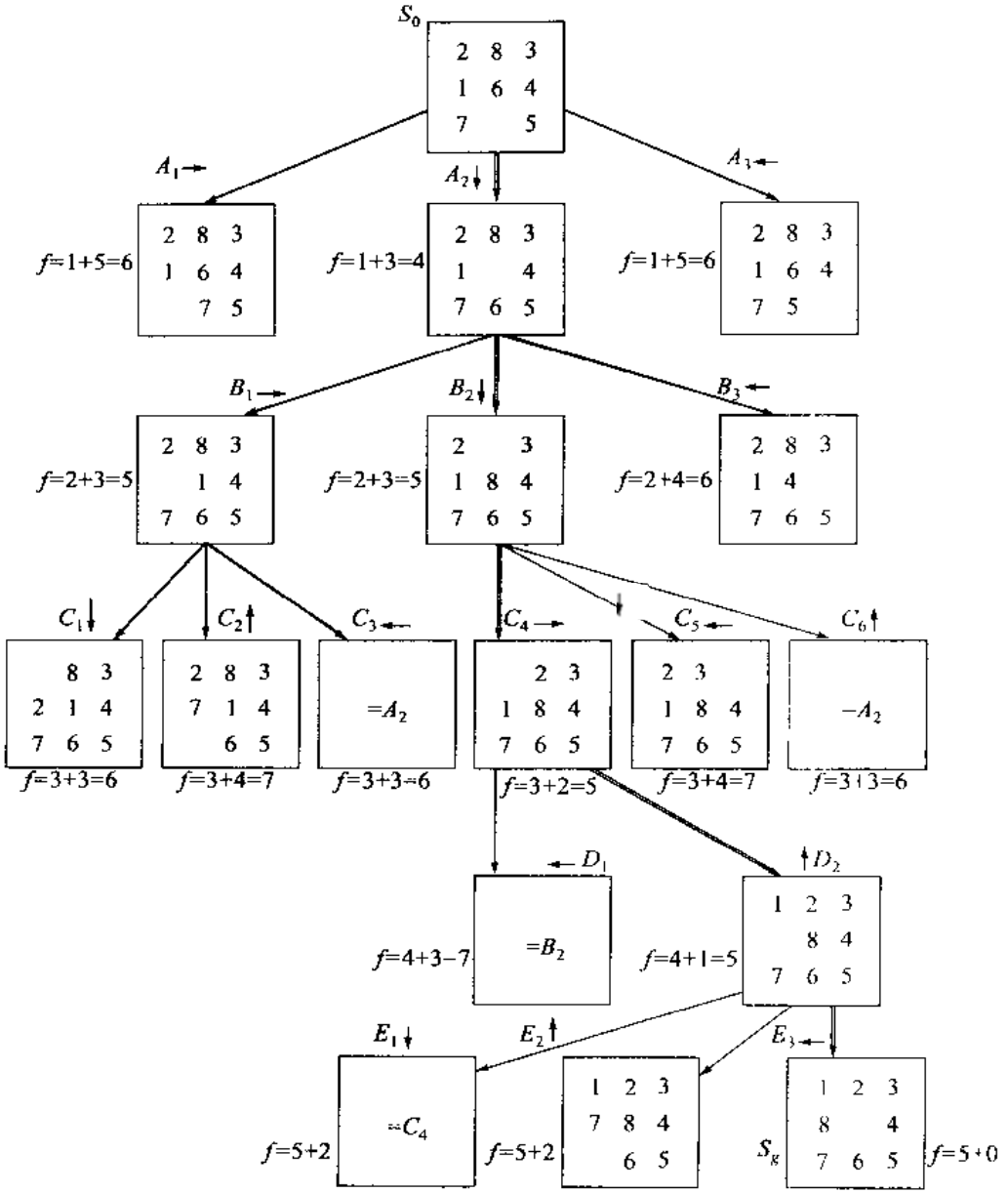
  (2) 先生成的节点优先。

但生成(C_1, C_2, C_3)之后，下次在算法的第4步仍会在Open表中找最小，即：

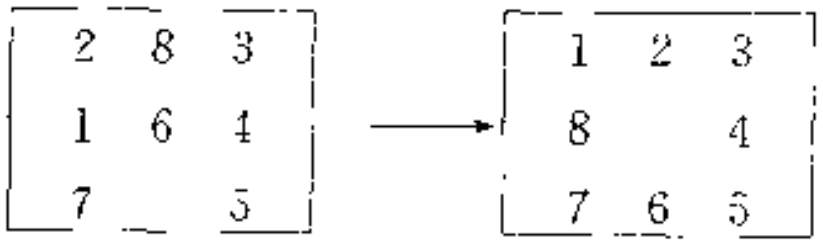
一层　（$A_1=6$, $A_3=6$）

二层　（$B_2=5$, $B_3=6$）

三层　（$C_1=6$, $C_2=7$, $C_3=6$）



九宫格问题（同一个问题可以有多种设计方法）



假设h（n）= p(*n*)：为节点n中每一个数字与其目标位置之间的距离总和。（不考虑有没有路走，能不能走）

显然这时，w（n）≤p（n）≤h*（n），因此它也满足A*算法的要求。但是p（n）比w（n）有更强的启发性信息。因此由h（n）= p(*n*)构造的启发式搜索树节点要更少。

A*算法的可采纳性

证明分两步：

（1）证若问题有解，A*一定终止，由如下命题4-1~4-3证出。

（2）证若问题有解，A*终止时一定找到最优解，由如下命题4-4证出。

A*算法的可采纳性证明

命题4.1 对有限图而言，A*一定终止

证：考察A*算法，算法终止只有二处：

- 第一处 在第5步，找到解时成功终止。

- 第二处 在第3步，open为空时失败退出。

设S_0：初态，S_g：目标状态

1. open={S_0}；
2. closed={ }；
3. 如果open={}，失败退出；
4. 在open表上取出f最小的结点n，n放到closed表中；其中：
   f(n)=g(n)+h(n)　　h<=h*
5. 若$n \in S_g$，则成功退出；
6. 产生n的一切后继，将后继中不是n的先辈点的一切点构成集合M
7. 对M中的元素P，分别作两类处理：
   7.1 若P∈G，则对P进行估计加入open表，记入G和Tree。
   7.2 P∈G，则决定是否更改Tree中P到n的指针并且更改P的子节点n的指针和费用。
8. 转3。

A*算法的可采纳性证明

命题4.2 若A*不终止，则搜索图中open表上的点的f值将会越来越大。

- 证：设n为open中任一节点，$d^*(n)$为从S到n中最短路径长度，由于从某一点求出其后继的费用不小于某个小的正数e，所以

$$g^*(n) \geq d^*(n) \cdot e$$

而 $g(n) \geq g^*(n) \geq d^*(n) \cdot e$

又因为 $h(n) \geq 0$

所以 $f(n) \geq g(n) \geq g^*(n) \geq d^*(n) \cdot e$ **(4-1)**

若A*不终止，Open表中总有后继节点加入，所以d*（n）会无限增大，因为那些小于当前f值的节点都会被考察（求出后继）后放入Closed表中。

A*算法的可采纳性证明

命题4.3 若问题有解，在A* 终止前，open表上必存在一点n'，n'位于从$S_0 \rightarrow S_g$的最优路径上，且有

$$f(n') \leq f^*(S_0) \qquad (4\text{-}2)$$

- $f^*(S_0)$ 表示从S_0到S_g的最优路的实际最小费用。
- $f(n')$表示从S_0经过n'到S_g的搜索费用的估计。
- $f^*(n)$表示从S_0经过n到S_g的最优路径的实际最小费用。

A*算法的可采纳性证明

证：令$S_0=n_0$，n_1，n_2，…，$n_k=S_g$为一条最优路径，设$n'\in$path(n_0，n_1，…，n_k）中最后一个出现在open表上的元素。显然n'一定存在，因为至少有$S_0=n_0$必然在open上，只考虑当n_k还未在closed表中时，因为若n_k已在closed表中时，则$n_k=S_g$，A*算法将终止于成功退出。

由定义有

$$f(n')=g(n')+h(n')=g^*(n')+h(n') \quad \text{(因为}n'\text{在最优路径上)}$$

$$\leq g^*(n')+h^*(n')=f^*(n')=f^*(S_0) \quad \text{(由于A}^*\text{的定义}h(n)\leq h^*(n)\text{)}$$

所以$f(n')\leq f^*(S_0)$成立。

A*算法的可采纳性证明

推论4.1 若问题有解，A*算法一定终止。

因为，若A*算法不终止，

则命题2的

$$f(n) \geq g(n) \geq g^*(n) \geq d^*(n) \cdot e \qquad (4\text{-}1)$$

与命题3的

$$f(n') \leq f^*(S_0) \qquad (4\text{-}2)$$

同时成立，这就产生矛盾。

A*算法的可采纳性

命题4.4 若问题有解，A*算法终止时一定找到最优解，即A*算法是可采纳的。

证：A*终止只有两种情况。

(1) 在第(3)步, 因open为空而失败退出

但由命题4.3可知：A*终止前，open表上必存在一点n’，满足

$$f(n') \le f^*(S_0)$$

即open表不会空，所以，不会终止于第3步。

A*算法的可采纳性

命题4.4 若问题有解，A*算法终止时一定找到最优解，即A*算法是可采纳的。

(2) 在第（5）步终止。若终止时找到的一条路径$S_0=n_0$，n_1，n_2，…，$n_k=S_g$不是最佳路径，即有 $f(n_k)>f^*(S_0)$

但从命题4.3知，存在n’∈Open，有$f(n')\le f^*(S_0)$

因此$f(n')\le f(n_k)$那么A*算法应该选n’，而不应该选n_k，所以产生矛盾。

所以A*终止时，找到的是一条最优路径。

A*算法的性质

推论4.2 凡open表中任一点n，若$f(n)< f^*(S_0)$，最终都将被A*算法挑选出来求后继，也即被挑选出来进行扩充。

证：用反证法，设$f(n) < f^*(S_0)$且n没有被选出来作后继。

由命题4.4, A*算法将找到一条路$S_0=n_0$，n_1，…，$n_k=Sg$,为最优路径

且$f(n_i) \le f(n)$对一切i=0,1,…,k成立，因为最优路径选择的是n_i而不是n。

又因为n_i在最优路上,由$f(n_i)=f^*(S_0)$,所以 $f^*(S_0) \le f(n)$

所以 $f^*(S_0) \le f(n)$与$f(n)<f^*(S_0)$同时成立,这是一个矛盾。

A*算法的性质

命题4.5 凡A*算法挑选出来求后继的点n必定满足：　$f(n) \le f^*(S_0)$　　(4-3)

证明: n不会是S_g

若$n \neq S_g$,由命题4.3可知:存在$n' \in$ open

且有　$f(n') \le f^*(S_0)$

而现在A*算法选中了n而不是n',所以必有

$f(n) \le f(n') \le f^*(S_0)$

即　$f(n) \le f^*(S_0)$

A*算法的性质

定义4.1 若A_1, A_2均是A^*算法,

A_1采用$f_1(x)=g_1(x)+h_1(x)$ 作为估计函数，

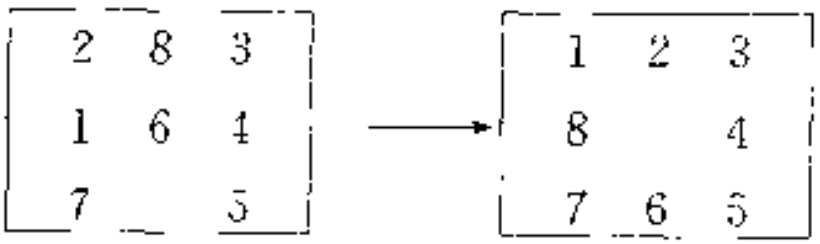
A_2采用$f_2(x)=g_2(x)+h_2(x)$ 作为估计函数。

$h_1(x)$,$h_2(x)$都满足: $h_i(x)\leq h_i^*(x)$,i=1,2 (4-4)

如果 $h_1(x) < h_2(x)$，则称A_2比A_1更具有信息(more informed)。

九宫格问题（同一个问题可以有多种设计方法）

采用A*算法，令$f(n) = d(n) + w(n)$。
其中$d(n)$为搜索树的深度。



假设h（n）= w(n)：放错位置数字个数

假设h（n）= p(n)：为节点n中每一个数字与其目标位置之间的距离总和。（不考虑有没有路走，能不能走）

显然这时，w（n）≤p（n）≤h*（n），p（n）比w（n）有更强的启发性信息。因此由h（n）= p(n)构造的启发式搜索树节点要更少。

A*算法的性质

- **命题6** 若A_2比A_1更具有信息,对任一图的搜索,只要从$S_0 \rightarrow Sg$存在一条路径,那么A_2所用来扩充的点也一定被A_1所扩充。

证明: 在证明之前需要说明，在图搜索过程中, 若某一点有几个先辈节点,则只保留最小费用的那条路,所以A_1 和A_2搜索的结果是树而不是图。

下面以A_2搜索树中节点的深度来归纳证明。

A*算法的性质

- 归纳基础：设A_2扩充的点n的深度d=0，即$n=S_0$，显然A_1也扩充点n，因为A_1、A_2都要从S_0开始。
- 归纳假设：假设A_1扩充了A_2搜索树中一切深度$d \leq k$的节点。
- 归纳证明：要证明A_2搜索树中深度d=k+1的任一节点n也必定为A_1所扩充。

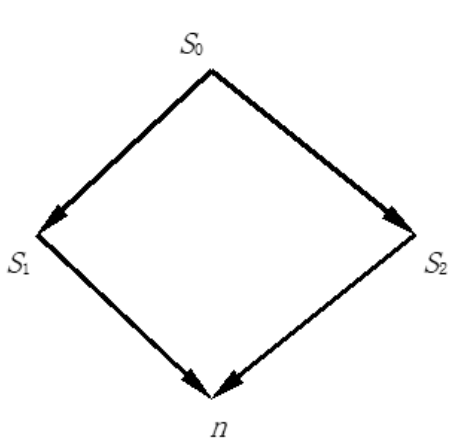
  用反证法：若A_2扩充了n，而A_1没有扩充n，将导出矛盾。

A*算法的性质

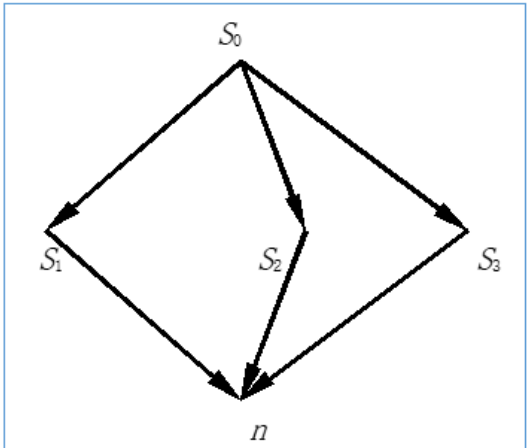
由归纳法假设可知A_1搜索树深度小于等于k的节点包含A_2搜索树深度小于等于k的节点，所以 如果存在路径$S_0 \rightarrow n$, d(n)=k+1, 则有

$$g_1(n) \leq g_2(n) \qquad (4\text{-}5)$$

因为A_1搜索树中从$S_0 \rightarrow n$路径多些.



(a) A_2算法搜索树



(b) A_1算法搜索树

A_1搜索树中从$S_0 \rightarrow n$路径比A_2多一些

A*算法的性质

又由命题4.5，如果A_2扩充n，必有

$$f_2(n) \le f_2^*(S_0) = f_1^*(S_0) \qquad (4\text{-}6)$$

其中$f_2^*(S_0) = f_1^*(S_0)$ 是因为最优路径的费用相同。

由命题4.4的推论和命题4.5，A_1不扩充n必有

$$f_1(n) \ge f_1^*(S_0) \qquad (4\text{-}7)$$

可推出

$$g_1(n)+h_1(n) \ge f_1^*(S_0) \qquad (4\text{-}8)$$

$g_1(n) \le g_2(n)$ (4-5)

A*算法的性

由命题4.4的推论和命题4.5，A_1不扩充n必有

$$f_1(n) \ge f_1^*(S_0) \quad (4\text{-}7)$$

由（4-7）和 (4-5)得

$$h_1(n) \ge f_1^*(S_0) - g_1(n) \ge f_1^*(S_0) - g_2(n) \quad (4\text{-}9)$$

由于A_2比A_1更具有信息：

$$h_2(n) \ge h_1(n) \ge f_1^*(S_0) - g_2(n)$$

即 $h_2(n) \ge f_1^*(S_0) - g_2(n)$ (4-10)

但从式（4-6）知：

$$f_2(n) \le f_2^*(S_0) = f_1^*(S_0) \quad (4\text{-}6)$$

$$h_2(n) \le f_1^*(S_0) - g_2(n) \quad (4\text{-}11)$$

式（4-10）与式（4-11）矛盾，所以本命题的结论得证。

h(n)的选择

搜索的费用与h(*n*)有一定的关系

A*算法要求h(*n*）≤h*(*n*）

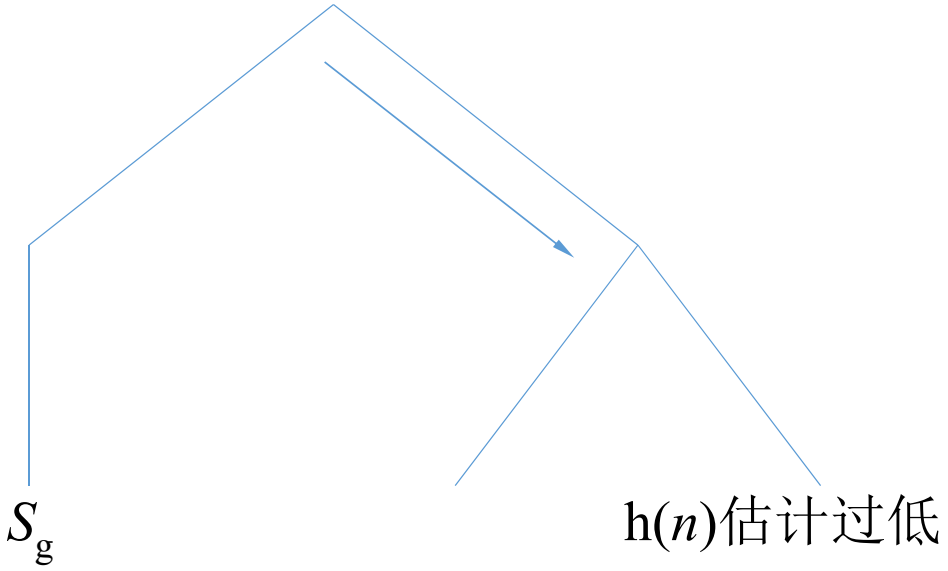
- 但并不是越小越好，
- 也并不是越大越好，

下面分两种情况讨论。

h(n)的选择

■ **若 h(*n*)估计过低，浪费过多**

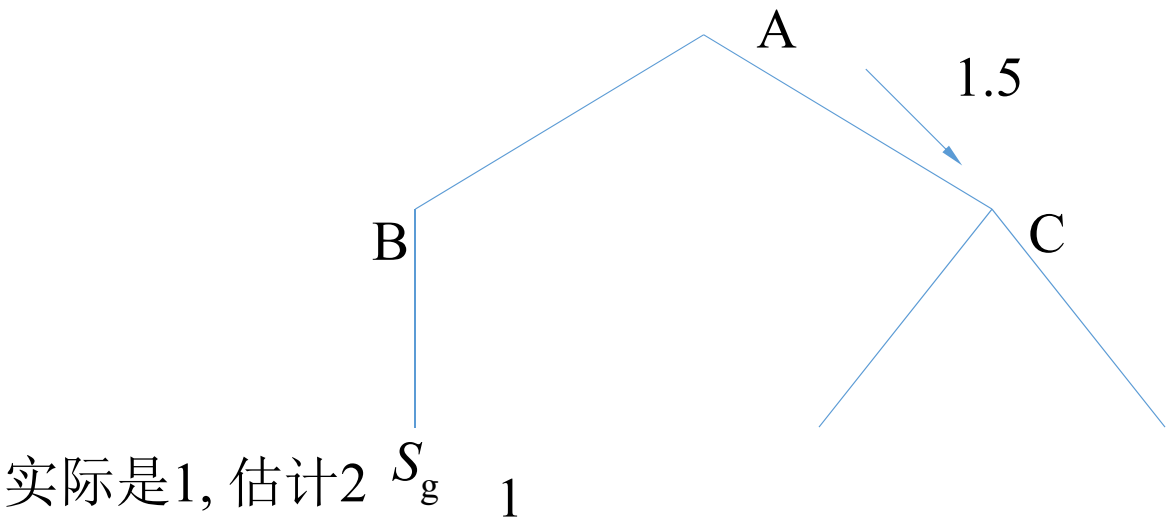
h(x)愈小，因为A*算法总是找具有小的f值的节点来扩充，将会造成一种误导，导致本不是通向解点也要搜索，并求后继。这样必定白白浪费时空。



h(n)的选择

■ **若h(n)估计过高，则可能错过目标。**

当h(x) 超过它的实际值时，则有可能错过本来可以到达的目标。在下图中，若h(B)>AC分枝中任一点的值，则永远不会走B这条路，这将导致可能找不到解。



h(n)的选择

- 另外搜索的费用并不完全由搜索节点数多少来确定，若$f_2(n)$计算远比$f_1(n)$复杂，则在比较两个搜索算法时，必须要考虑f的计算费用。

一般来说要权衡如下三个因素：

(1) 路径费用；

(2) 寻找路径时所搜索的节点数；

(3) 计算f所需的计算量。

A*算法的性质

定义4.2 一个启发式函数中的h(x)满足单调限制，可定义为：

如果对所有n_i与n_j，n_j是n_i的后继，有$h(n_i)\leq h(n_j)+c(n_i,n_j)$，并且$h(S_g)=0$，即$n_j$到目标的最佳费用估计不会大于$n_i$到目标的最佳费用估计加上$n_i$至$n_j$的费用。

A*算法的性质

命题4.7 估计函数若满足单调限制，那么A*所扩充的任一点(即用来求过后继的点）n必在最优路上（总是从祖先状态沿着最佳路径到达n）。

证明思路：令$g^*(n)$代表从$S_0 \rightarrow n$的最优路径费用，所以一般有:

$$g(n) \geq g^*(n) \qquad (4\text{-}12)$$

下面再利用单调限制能够证明：

$$g(n) \leq g^*(n) \qquad (4\text{-}13)$$

联立(4-12），(4-13），可得

$g(n) = g^*(n)$，也就说明了n位于最佳路径上。

A*算法的性质

证明:设n为open表中被A*算法当前准备扩充的任一点，又设P=(S_0=n_0, n_1, …, n_i, n_{i+1}, …, n_k = n)为从S_0到n的一条最优路径。那么．此时P中必有一点n_i在closed表中，且其后继n_{i+1}则在open表中。

由单调限制的定义：

$h(n_i) \le h(n_{i+1}) + c(n_i, n_{i+1})$

$g^*(n_i) + h(n_i) \le g^*(n_i) + h(n_{i+1}) + c(n_i, n_{i+1})$

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

$g^*(n_i) + h(n_i) \le g^*(n_{i+1}) + h(n_{i+1})$

$g^*(n_{i+1}) + h(n_{i+1}) \le g^*(n) + h(n)$

A*从open表中选择n而不是n_{i+1}来扩展，则说明必有:

$f(n) \le f(n_{i+1})$, 即：$g(n) + h(n) \le g(n_{i+1}) + h(n_{i+1})$

因n_i和n_{i+1}都在最优路径上，所以$g(n_{i+1}) = g^*(n_{i+1})$

$g(n) + h(n) \le g^*(n_{i+1}) + h(n_{i+1})$

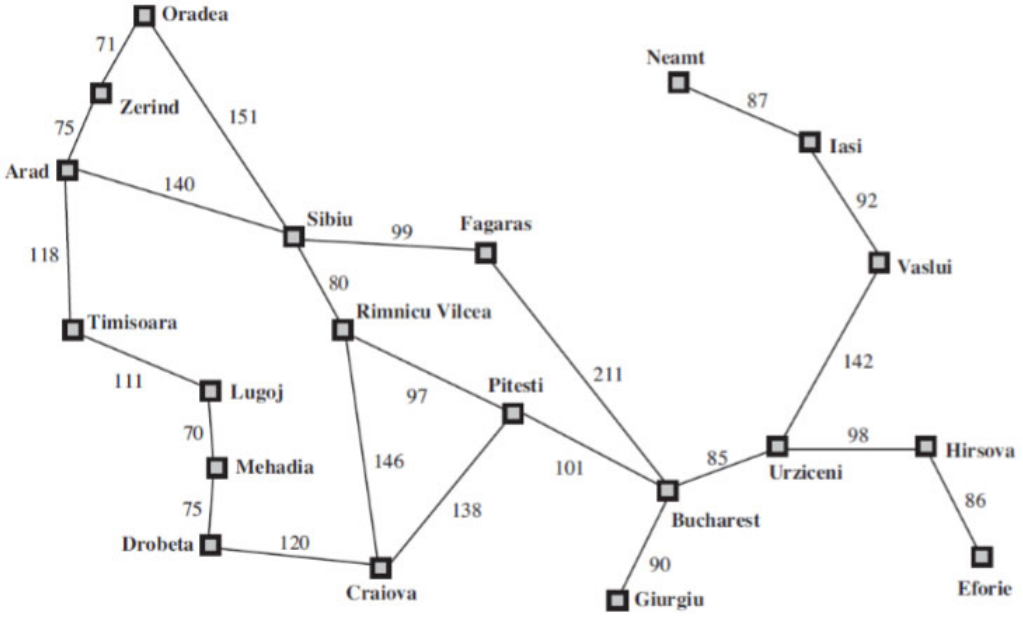
得到：$g(n) \le g^*(n)$

$g(n) = g^*(n)$

A*算法的性质

由于算法总是在第一次发现该点时就已经发现了到达该点状态的最短路径，所以当某一状态被重新搜索时，就无须检验新的路径是否更短，那是不可能的，这就意味着当某一状态被重新搜索时，可以将其立即从open或closed表中删除，而无需修改路径的信息。

Example: Form Arad to Bucharest 举例：从Arad到Bucharest

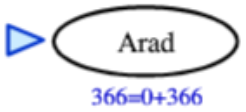


h_{SLD} Values	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

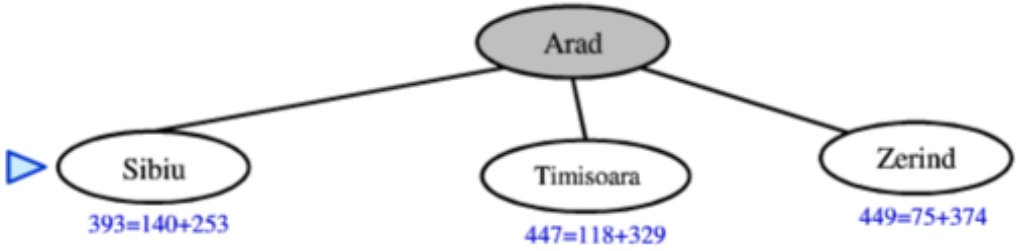
$f(n) = g(n) + h(n)$, which $g(n)$ = path cost, $h(n) = h_{SLD}$

Example: Form Arad to Bucharest 举例：从Arad到Bucharest

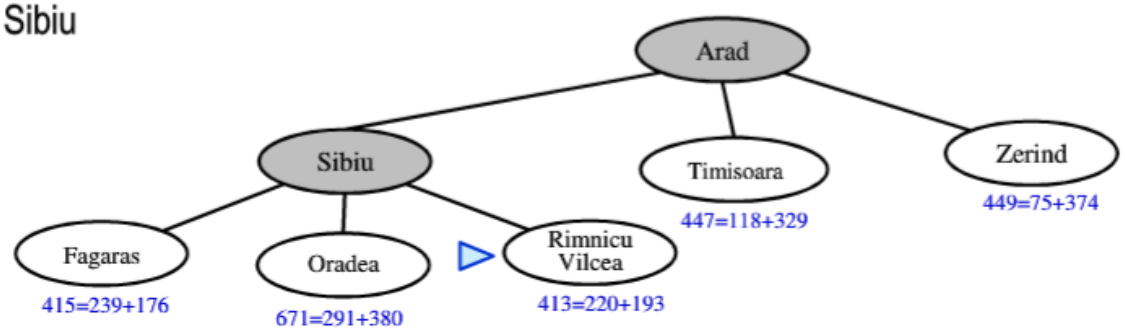
(a) The initial state



(b) After expanding Arad

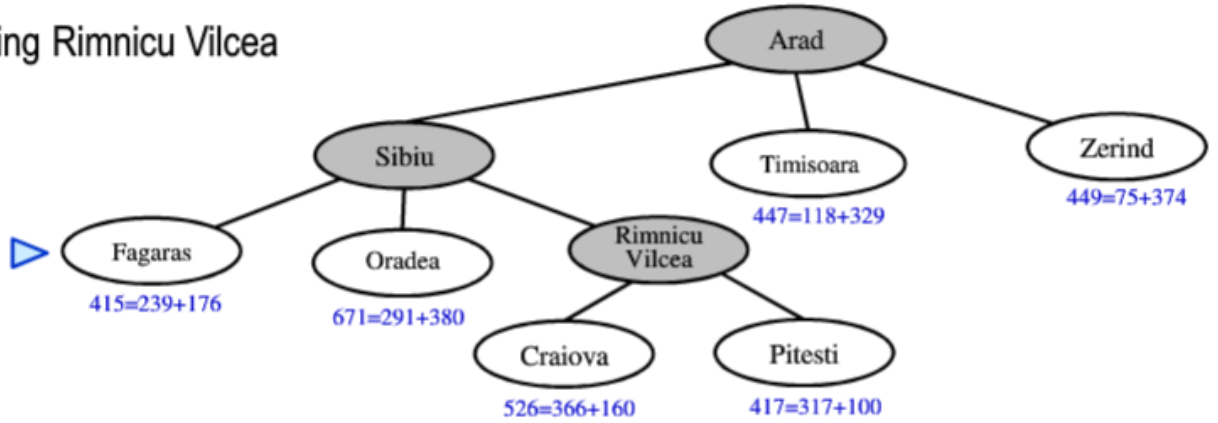


(c) After expanding Sibiu

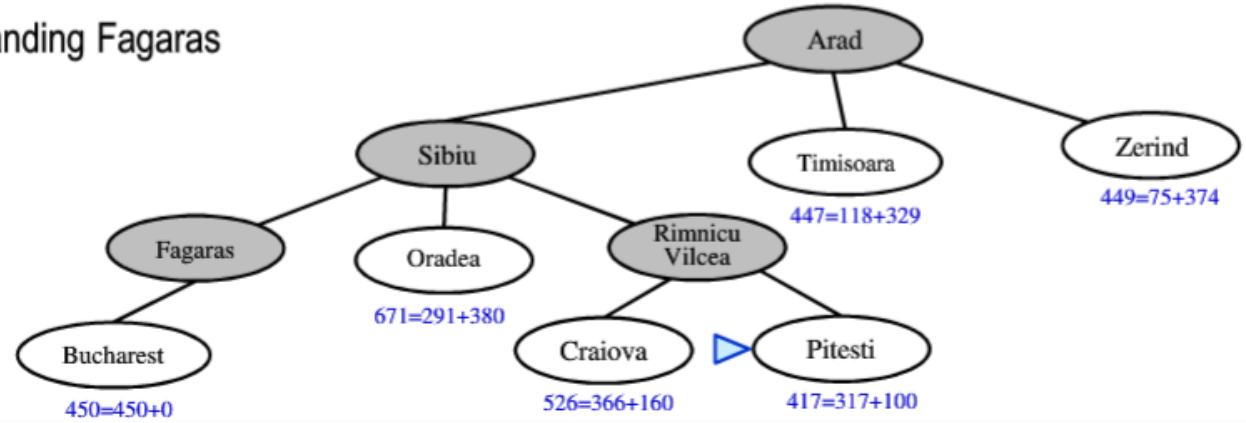


Example: Form Arad to Bucharest 举例：从Arad到Bucharest

(d) After expanding Rimnicu Vilcea

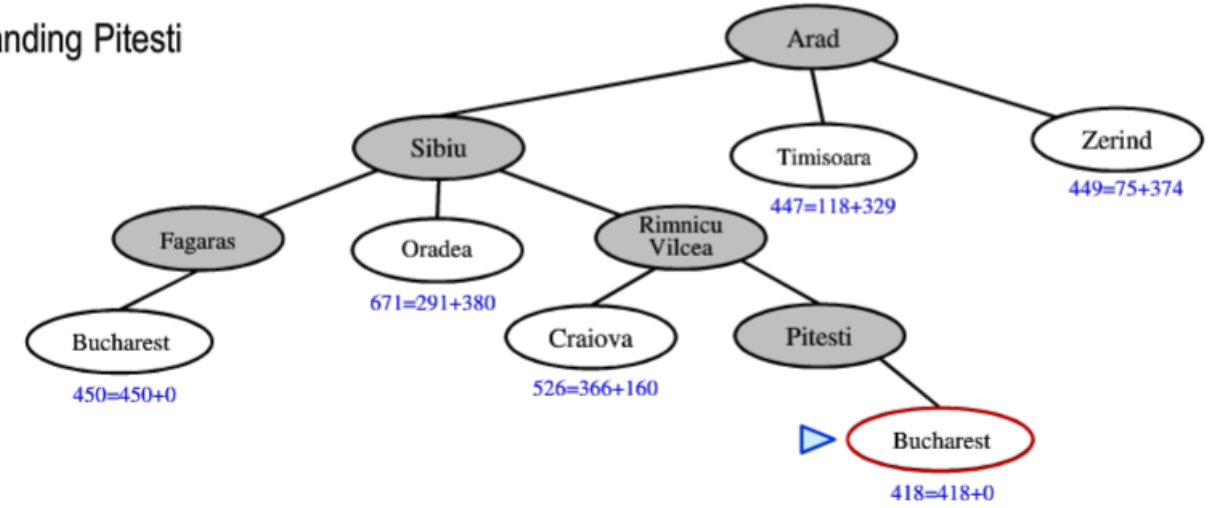


(e) After expanding Fagaras



Example: Form Arad to Bucharest 举例：从Arad到Bucharest

(f) After expanding Pitesti



A*算法的评价

A*算法的优点有：

（1）A*算法一定能保证找到最优解。

（2）若以搜索的节点数来估计它的效率，则当启发式函数h的值单调上升时，它的效率只会提高，不会降低。

（3）有比较合理的渐近性质。

A*算法的评价

A*算法的缺点是：

不仅考虑搜索节点的多少，而且还要考虑搜索节点被搜索的次数的时候，则当h(n）过低估计h*(n）时，有时会显出很高的复杂性。

A*算法的缺点举例

例4.4 设一个图G(m)由m个节点组成，

$n_m = S_0$, $n_0 = S_g$,且:

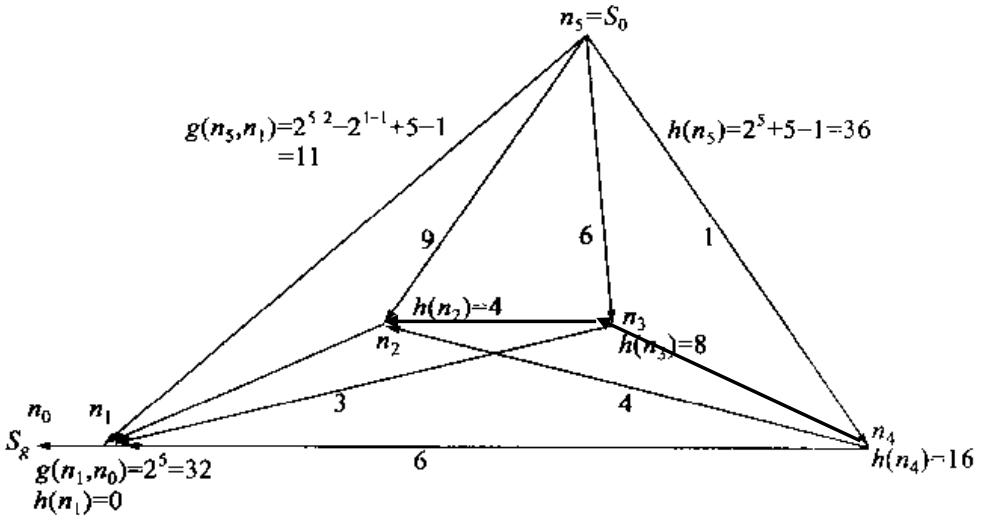
$g(n_i, n_j) = 2^{i-2} - 2^{j-1} + i - j$

$g(n_1, n_0) = 2^m$

$h(n_m) = 2^m + m - 1$

$h(n_0) = h(n_1) = 0$

$h(n_i) = 2^i$



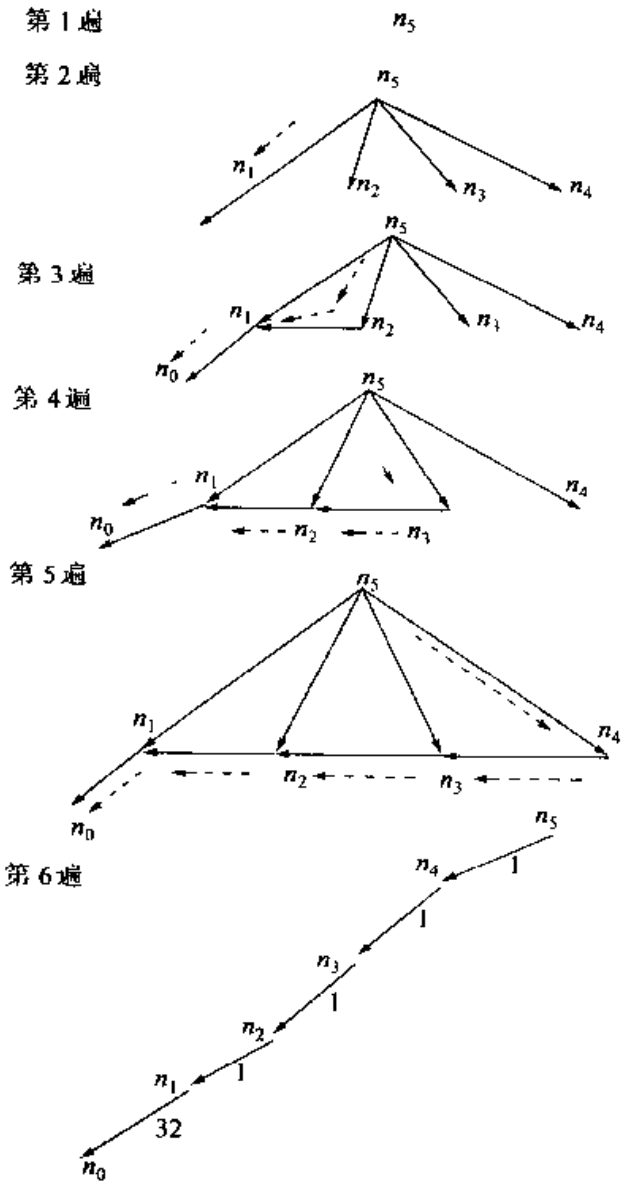
$f(n_1)=g(n_5,n_1)+h(n_1)=2^{5-2}-2^{1-1}+5-1+h(n_1)=11+0=11$

$f(n_2)=g(n_5,n_2)+h(n_2)=2^{5-2}-2^{2-1}+5-2+h(n_2)=9+4=13$

$f(n_3)=g(n_5,n_3)+h(n_3)=2^{5-2}-2^{3-1}+5-3+h(n_3)=6+8=14$

$f(n_4)=g(n_5,n_4)+h(n_4)=2^{5-2}-2^{4-1}+5-4+h(n_4)=1+16=17$

$f(n_5)=h(n_5)=2^5+5-1=36$



$f(n_1)=g(n_5,n_1)+h(n_1)=11+0=11$
$f(n_2)=g(n_5,n_2)+h(n_2)=9+4=13$
$f(n_3)=g(n_5,n_3)+h(n_3)=6+8=14$
$f(n_4)=g(n_5,n_4)+h(n_4)=1+16=17$
$f(n_5)=h(n_5)=36$

$f(n_0)=g(n_5,n_1)+g(n_1,n_0)=11+32=43$

因为n_2扩展了n_1，考虑从n_5到n_1会不会有更好的路径：
原来:g（n_1）=11
加入n_2后：
$g(n_1)=g(n_2)+g(n_2,n_1)=9+1=10$

因为n_3扩展了n_2，考虑从n_5到n_1和n_2会不会有更好的路径：
原来:g（n_2）=9
加入n_3后：
$g(n_2)=g(n_3)+g(n_3,n_2)=6+1=7$
$g(n_1)=g(n_2)+g(n_2,n_1)=7+1=8$

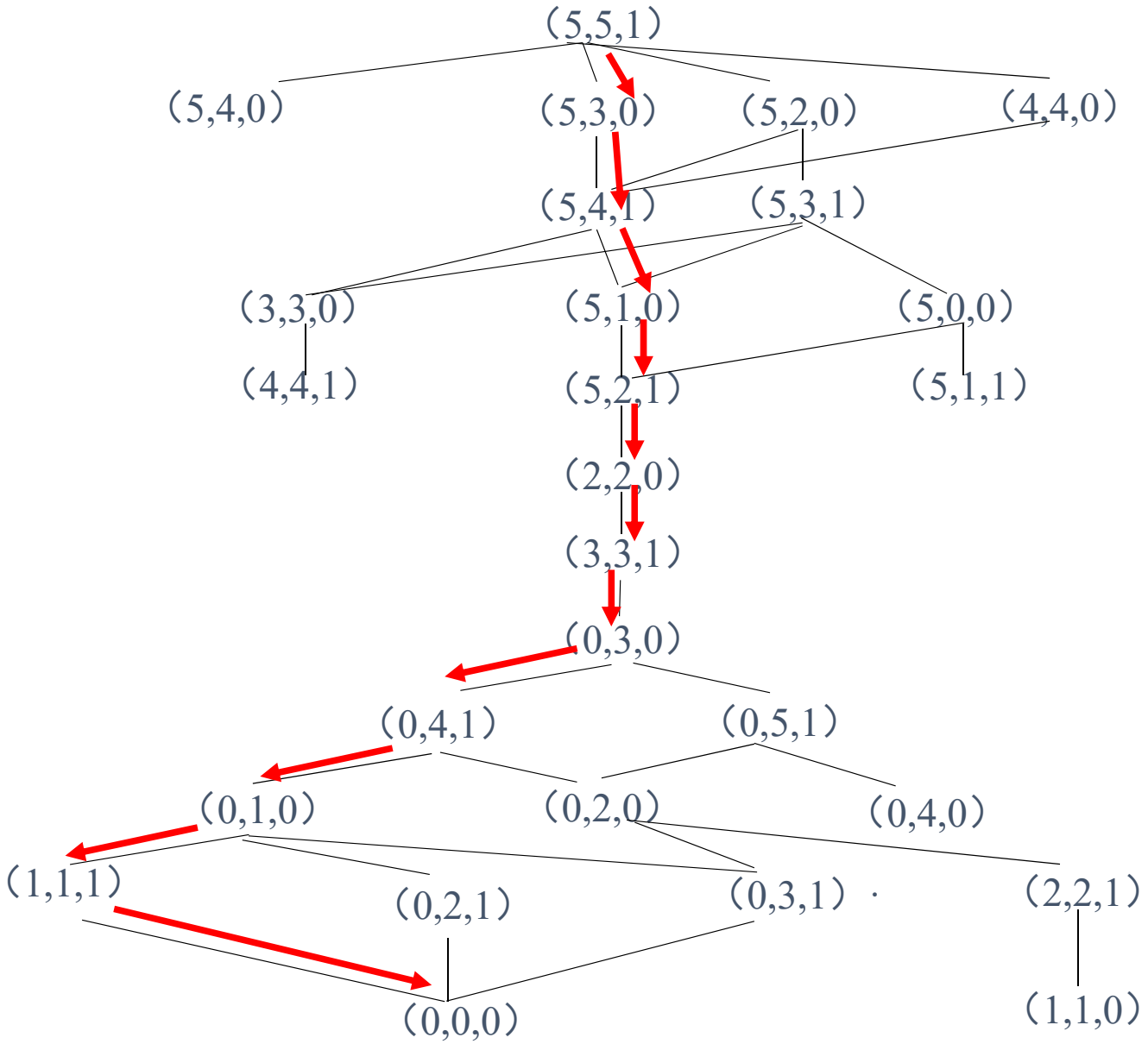
A*算法举例

传教士和野人问题（missionaries and cannibals, M-C问题）

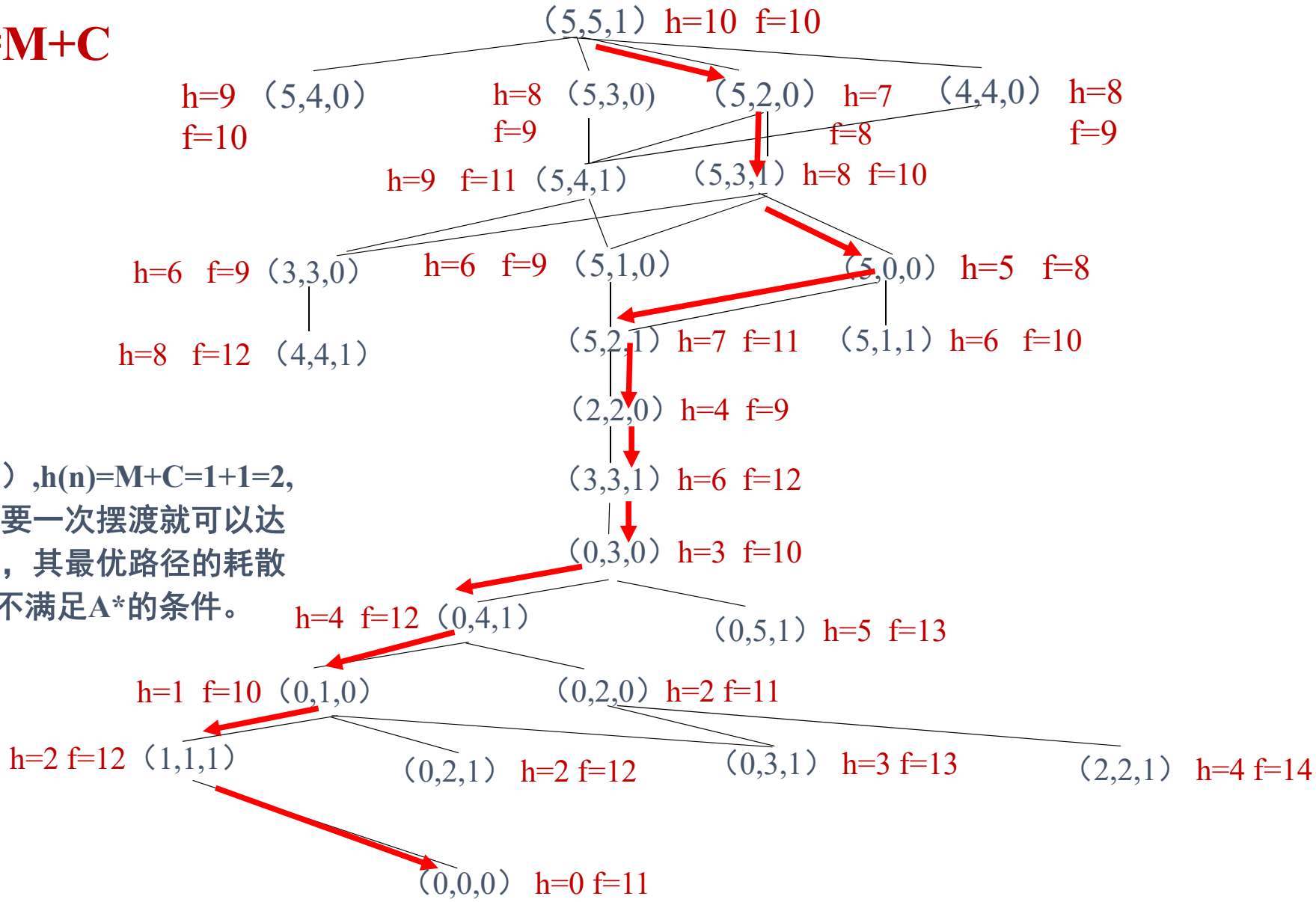
有N个传教士和N个野人来到河边准备渡河，河岸有一条船，每次至多可供K人乘渡。问传教士为了安全起见，应如何规划摆渡方案，使得任何时刻，在河的两岸以及船上的野人数目总是不超过传教士的数目（但允许在河的某一岸只有野人而没有传教士）。

假设N=5，K<=3，传教士和野人从左岸向右岸过河。在某时刻，在河左岸的传教士数用M表示，野人数用C表示。B=1表示船在左岸，B=0表示船在右岸。考虑用M和C的组合作为启发函数的基本分量。

h(n)=0

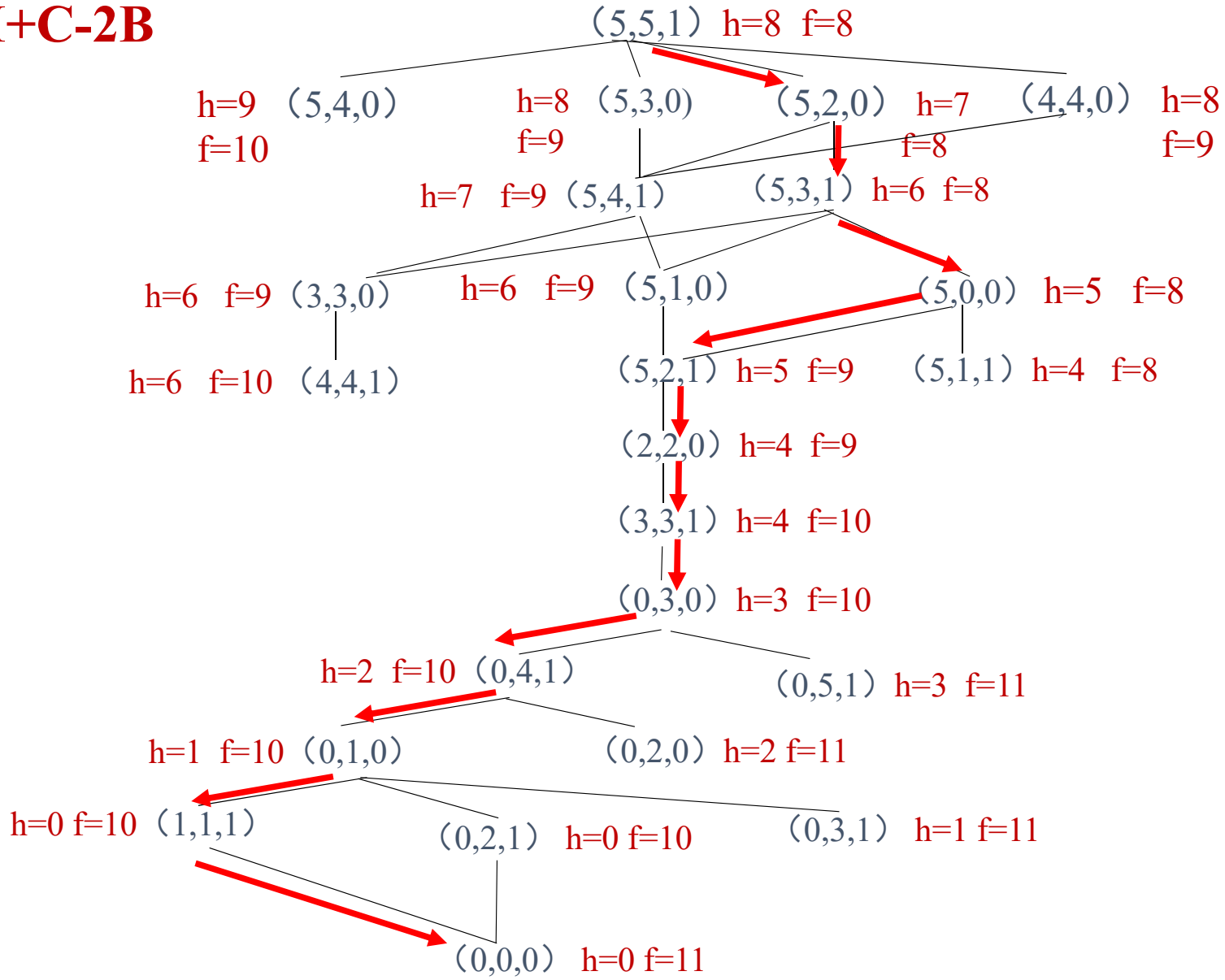

(5,5,1)
(5,4,0)
(5,3,0)
(5,2,0)
(4,4,0)
(5,4,1)
(5,3,1)
(3,3,0)
(5,1,0)
(5,0,0)
(4,4,1)
(5,2,1)
(5,1,1)
(2,2,0)
(3,3,1)
(0,3,0)
(0,4,1)
(0,5,1)
(0,1,0)
(0,2,0)
(0,4,0)
(1,1,1)
(0,2,1)
(0,3,1)
(2,2,1)
(1,1,0)
(0,0,0)

h(n)=M+C



状态（1,1,1）,h(n)=M+C=1+1=2,而实际上只要一次摆渡就可以达到目标状态，其最优路径的耗散值为1.所以不满足A*的条件。

h(n)=M+C-2B


(5,5,1) h=8 f=8
h=9 (5,4,0) f=10
h=8 (5,3,0) f=9
(5,2,0) h=7 f=8
(4,4,0) h=8 f=9
h=7 f=9 (5,4,1)
(5,3,1) h=6 f=8
h=6 f=9 (3,3,0)
h=6 f=9 (5,1,0)
(5,0,0) h=5 f=8
h=6 f=10 (4,4,1)
(5,2,1) h=5 f=9
(5,1,1) h=4 f=8
(2,2,0) h=4 f=9
(3,3,1) h=4 f=10
(0,3,0) h=3 f=10
h=2 f=10 (0,4,1)
(0,5,1) h=3 f=11
h=1 f=10 (0,1,0)
(0,2,0) h=2 f=11
h=0 f=10 (1,1,1)
(0,2,1) h=0 f=10
(0,3,1) h=1 f=11
(0,0,0) h=0 f=11

证明h(n)=M+C-2B满足A*条件

- 船在左岸时

不考虑限制条件，船一次可以将3人从左岸运到右岸，然后再有一个人将船送回来。这样，船一个来回可以运过河2人，而船仍在左岸。而最后剩下的3个人可以一次将他们全部从左岸运到右岸。

至少需要摆渡 $\lceil \frac{M+C-3}{2} \rceil \times 2 + 1$次，化简后：

$$\lceil \frac{M+C-3}{2} \rceil \times 2 + 1 \geq \frac{M+C-3}{2} \times 2 + 1 = M + C - 3 + 1 = M + C - 2$$

证明h(n)=M+C-2B满足A*条件

- 船在右岸时

不考虑限制条件，船在右岸，需要一个人将船运到左岸。因此对状态（M，C，0）来说，其所需要的最少摆渡数，相当于船在左岸时状态（M+1，C，1）或（M，C+1，1）所需要的最少摆渡数，再加上一次将船从右岸送到左岸的一次摆渡数。

至少需要摆渡 （M+C+1）-2+1次，化简后为M+C

综合以上两种情况：M+C-2B 其中B=1表示船在左岸，B=0表示船在右岸

4.2 问题归约与AO*算法

在问题求解过程中，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大问题的解。问题归约法不同于状态空间法，是另一种问题描述和求解的方法。

问题归约求解方法

问题归约中，问题分解为子问题后，对于求解原问题有3种可能（与/或图概念的由来）：

1. 解决其中一个子问题，原问题得解（or）
2. 解决全部问题，原问题得解（and）
3. 解决其中一些子问题，原问题得解（and，or）

从初始问题出发, 建立子问题以及子问题的子问题, 直至把初始问题规约为一个本原问题的集合, 这就是问题规约的实质。

问题归约求解方法

待求解的原问题，被称为**初始问题**

- 可直接求解的子问题，被称为**本原问题**

问题归约求解方法

- 问题归约可以用三元组表示：（S_0，O，P）

  S_0：初始问题，即要求解的问题；

  P：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；

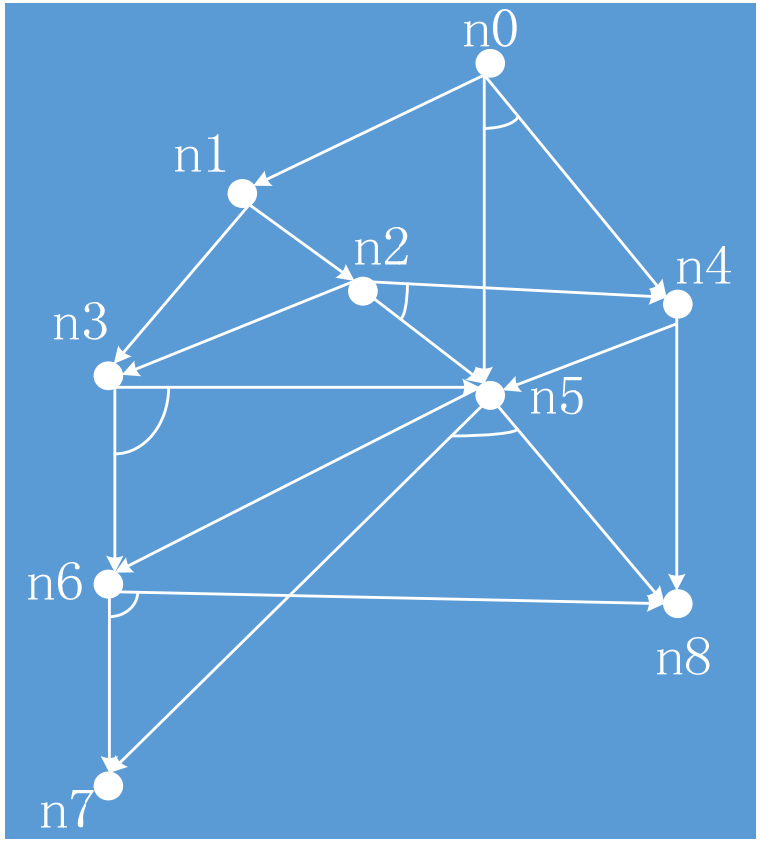
  O：操作算子集，它是一组变换规则，通过一个操作算子把一个问题化成若干个子问题。

- 这样，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。

与/或图搜索

- 与节点：把单个问题分解为几个子问题来求解。只有当所有子问题都有解，该父辈节点才有解。表示一种 “与” 关系。
- 或节点：同一问题被转换为几种不同的后继问题。只要有一个后继问题有解，则原问题有解。表示一种 “或” 关系。
- 与节点由与运算连接（超弧）.
- 或节点由或运算连接.

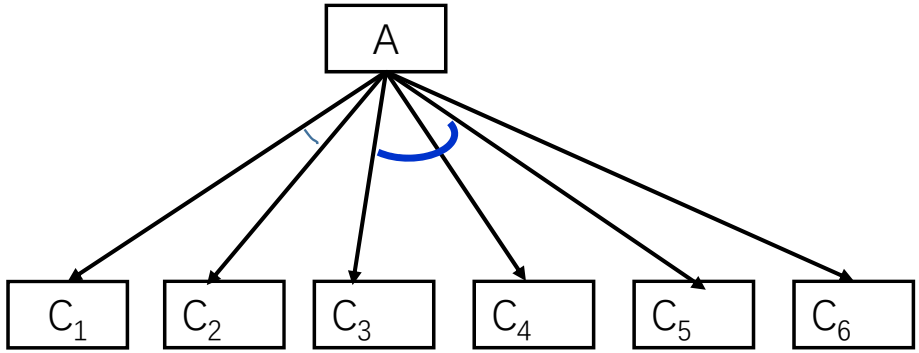
与或图

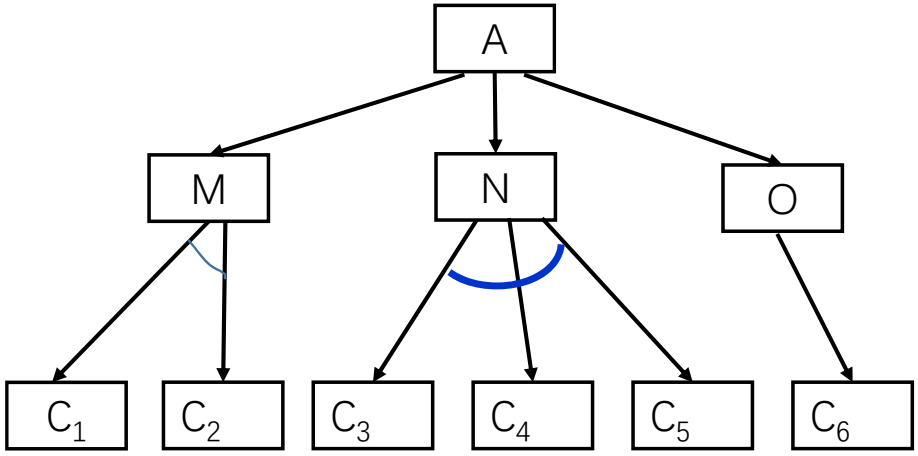


与或图

- 在与或图中，节点 n_0 有两个连接符：1-连接符指向节点 n_1；2-连接符指向节点集合{n_4、n_5}；
- 对于节点 n_0 来讲，n_1 可称为或节点，n_4、n_5 可称为与节点。

与或图





与/或图搜索

将问题求解归约为与/或图的时，作如下规定：

1. **根节点**为原始问题描述，本原问题的节点为**叶节点**。

2. **可解节点**为：

2.1 终叶节点是可解节点；

2.2 若n为一非终叶节点，且含有“或”后继节点，则只有当后继节点中至少有一个是可解节点时，n才可解；

2.3 若n为非终叶节点，且含“与”后继节点，则只有当后继节点全部可解时，n才可解。

与/或图搜索

3． 不可解节点：

3.1 没有后继节点的非终叶节点为不可解；

3.2 若n为一非叶节点含有“或”后继节点，则仅当全部后继节点为不可解时，n不可解。

3.3 若n为一非叶节点含有“与”后继节点，则只要有一个后继节点为不可解时，n为不可解。

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**。

与/或图搜索

- **4．图中搜索费用的计算**

设从当前节点n到目标集S_g费用估计为$h(n)$.

4.1 若$n \in S_g$, 则$h(n)=0$;

4.2 若n有一组由“与”弧连接的后继节点$\{n_1,n_2,\ldots n_i\}$ 则:

$$h(n)=c_1 + c_2 \ldots + c_i + \ldots + h(n_1) + h(n_2) + \ldots + h(n_i)$$

4.3 若n既有“与”又有“或”弧，则“与”弧算作一个“或”后继，再取各or弧后继中费用最小者为n的费用。

与/或图搜索的特点

1. 与/或图搜索费用的估计

其费用计算的规则是：

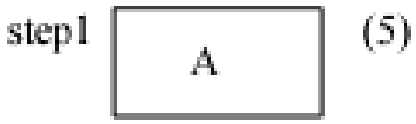
- n未生成后继节点时，费用由n本身决定;
- n已生成后继节点时，费用由n的后继节点的费用决定。

因为后继节点代表分解的子问题, 子问题的难易程度决定原问题求解的难易程度，所以不再考虑n本身的难易程度。因此当决定了某个路径时，要将后继节点的估计值往回传送。

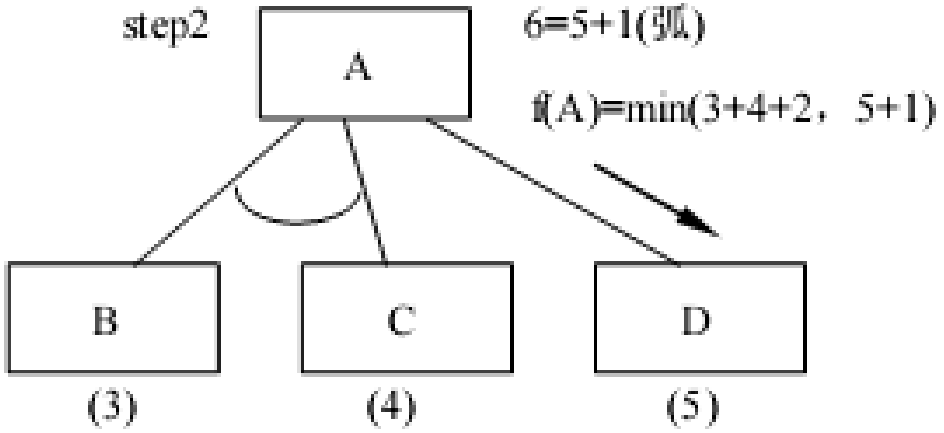
与/或图搜索的特点

例4.6 一个与/或图的搜索过程。

- 第一步，A是唯一节点；

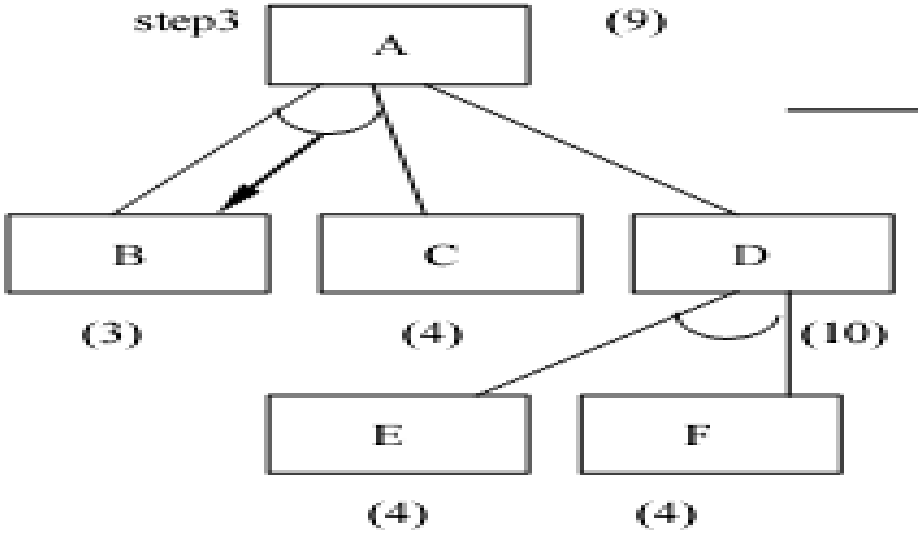


- 第二步，扩展A后，得到节点B,C和D, 因为B，C的耗费为9，D的耗费为6，所以把列D的弧标志为出自A最有希望的弧；

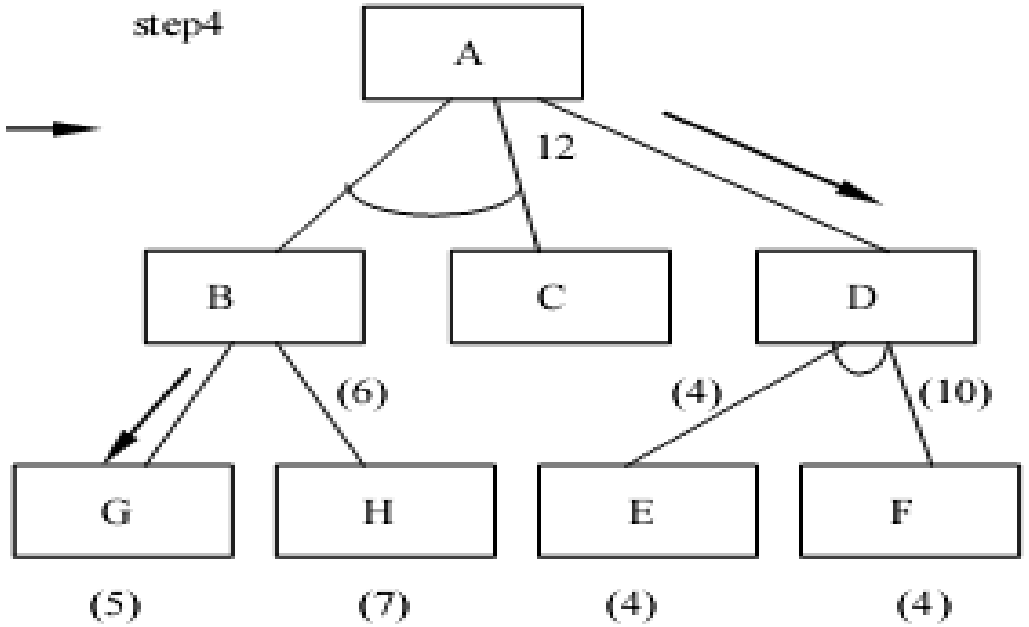


与/或图搜索的特点

第三步，选择对D的扩展，得到E和F的与弧，其耗费估计值为10。此时回退一步后，发现与弧BC比D更好，所以将弧BC标志为目前最佳路径；



第四步，在扩展B后,再回传值发现弧BC的耗费为12（6+4+2），所以D再次成为当前最佳路径。



与/或图搜索的特点

最后求得的耗费为:　f(A)=min(12,4+4+2+1)=11。

以上搜索过程由两大步组成：

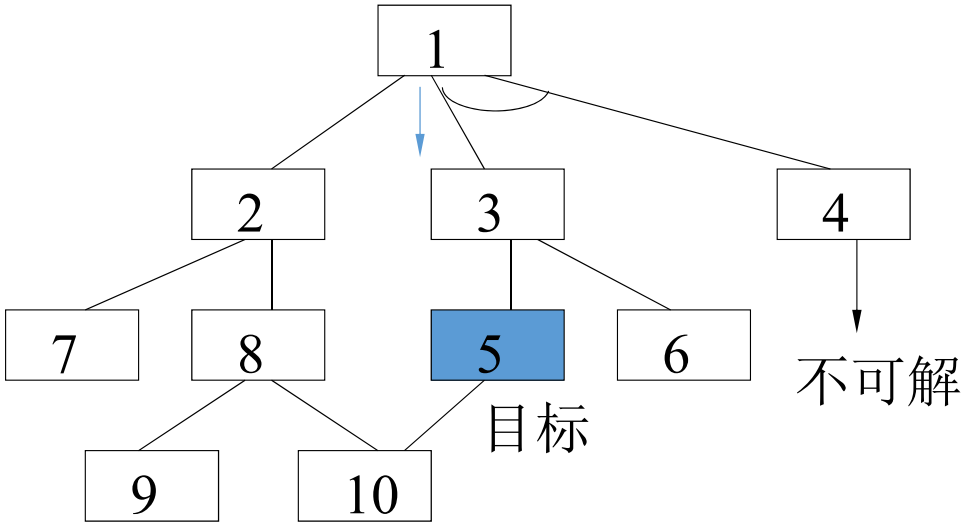
（1） 自顶向下，沿当前最优路产生后继节点。

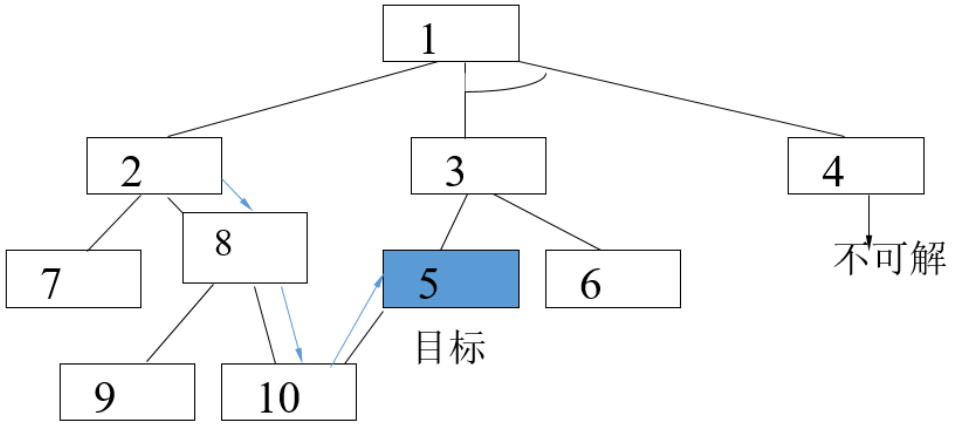
（2）由底向上，作估计值修正，再重新选择最优路径。

与/或图搜索的特点

2. 与/或图搜索路径的选择

由于有“与”连接弧，所以不能像“或”弧那样只看从节点到节点的个别路径。有时路径长反而好一些。如：





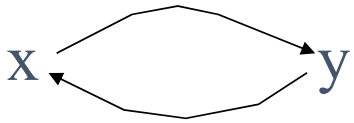
- 搜索虽然从①→②→⑧→⑩→⑤比①→③→⑤更长，但由于走③分枝的同时还必须走④，而④不可能通向解，所以有时走长点的路径比短路径要好一些。

与/或图搜索的特点

3. 与/或图搜索的限制

与/或图搜索仅对不含回路的图进行操作。

例如：



表示求了x就可以求y.，求了y就可以求x，两者都不可能求解。

与/或图搜索算法AO*

1. 令G=Init,计算h’(Init)。

2. 在Init标志solved之前或h’(Init)变成大于Futility之前，执行以下步骤：

2.1沿始于Init的已带标志的弧，选出当前沿标志路上未扩展的节点之一扩展（即求后继节点），此节点称为node。

2.2 生成node的后继节点。

若无后继节点，则令h’(node)=Futility,说明该节点不可解；

若有后继节点，称为successor,对每个不是node祖先的后继节点（避免回路），执行下述步骤：

2.2.1 将successor加入G。

2.2.2 若successor∈Sg,则标志successor为solved，且令h’(successor)=0。

2.2.3 若successor∉Sg,则求h’(successor)

与/或图搜索算法AO*

2.3 由底向上作评价值修正，重新挑选最优路径。令S为一节点集。

S＝｛已标志为solved的点，或h’值已改变, 需回传至其先辈节点的节点｝

令S初值＝｛node｝,重复下述过程，直到S为空时停止。

2.3.1 从S中挑选一节点,该节点的后辈点均不在S中(保证每一正在处理的点都在其先辈节点之前作处理),此节点称为current,并从S中删除;

2.3.2 计算始于current的每条弧的费用,即每条弧本身的费用加上弧末端节点h’ 的值(注意区分与或弧的计算方法),并从中选出极小费用的弧作为h’（current）的新值。

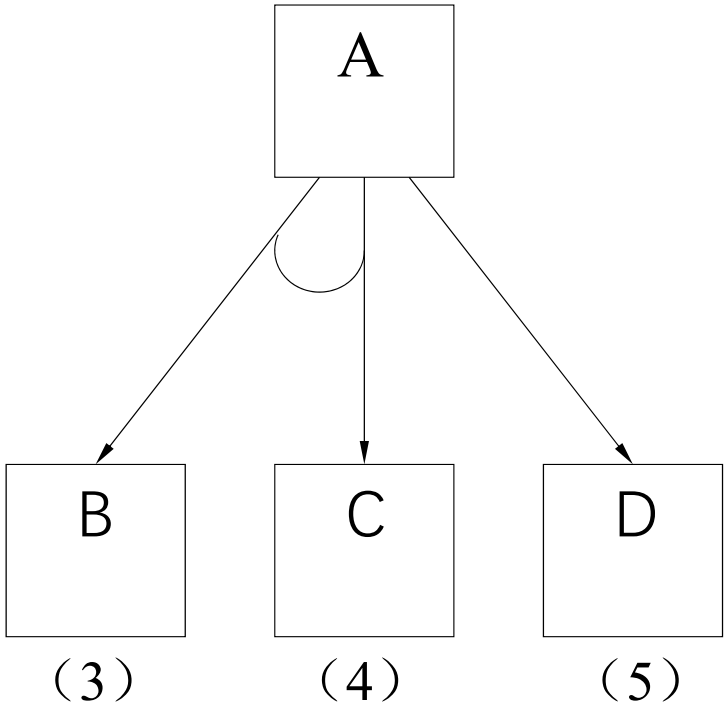
2.3.3 将费用最小弧标志为出自current的最优路径。

2.3.4 若current与新的带标志的弧所连接的点均标志solved,则current标志solved.。

2.3.5 若current已标志为solved或current的费用已改变，则需要往回传，因此要把current的所有先辈节点加入S中。

G= {A}， node=A, successor={ B and C, D}

G= { A, B and C, D }



step2.3.1 S={A}

step2.3.2 current：A

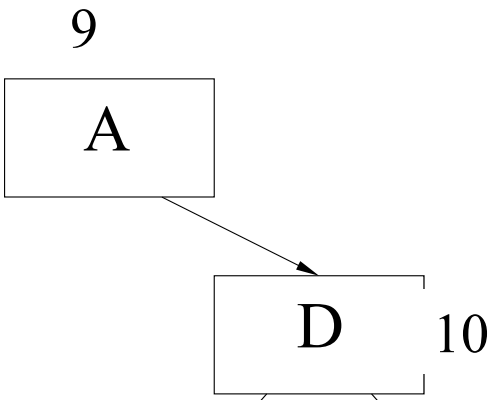
由于有A→B and C的弧，current的费用

=1+1+h’(B)+h’(C)=9

由于有A→D的弧，current的费用=1+5=6

A的费用=min(9,6)=6;

G= { A, B and C, D }　　**node=D, successor=**{E and F}



node=D，由step2.2.3，扩展D得

successor={E andF}，S={D} current：D

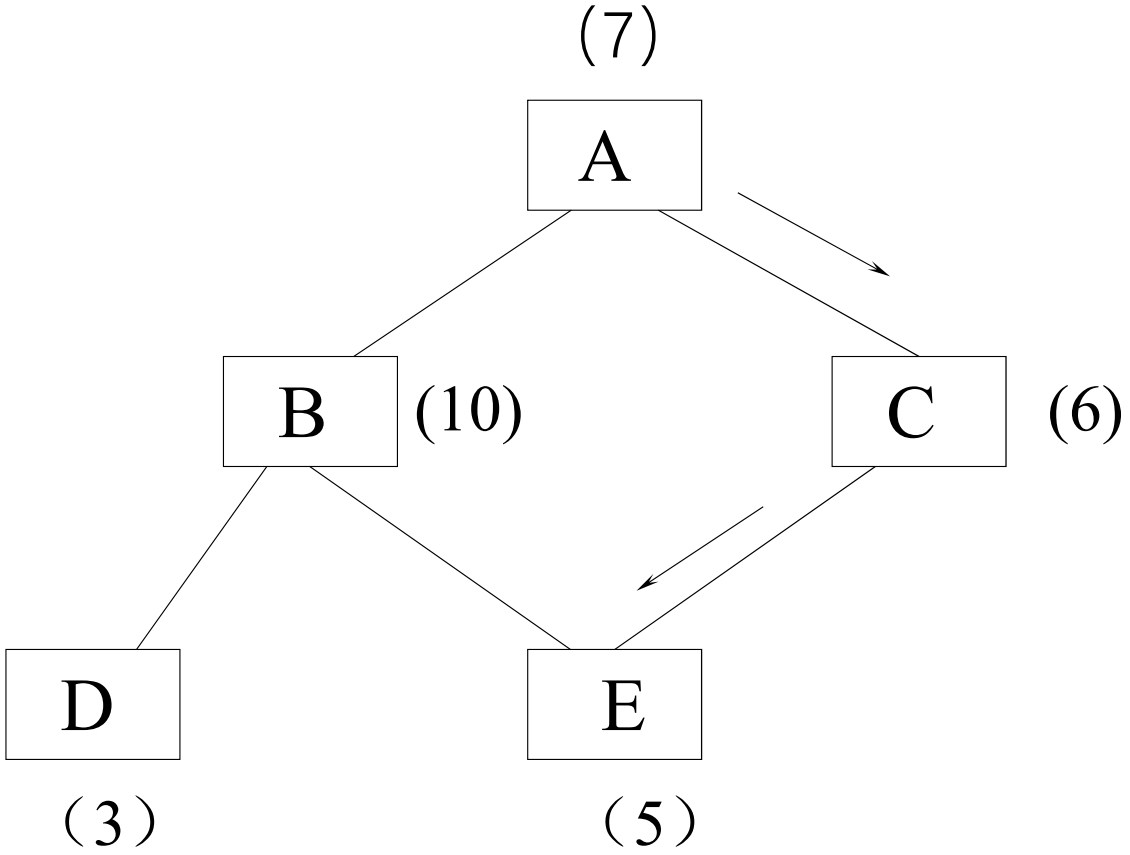
D的耗费估计已经改变，h’(D)=10

向上回传，S={A}

导致A的耗费为min(9, 11) = 9，所以，最优路径为A→BC弧。

对AO*算法的进一步观察

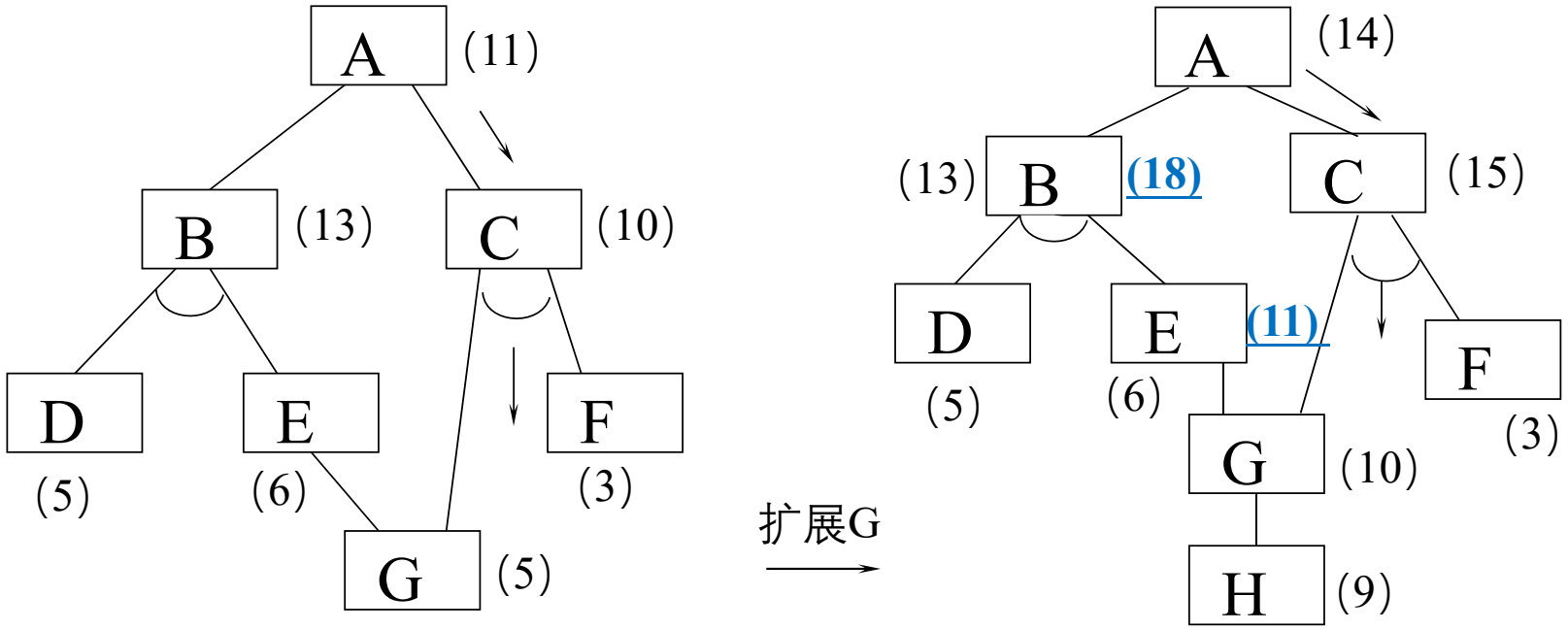
1.算法中2.3.5步中可能造成无用的回传



当从E往回传时，若实际上走A→B这条路总是不好时，则从E→B回传是浪费，是无用的回传，所以完全回传至一切祖先的费用很大。

对AO*算法的进一步观察

若仅沿标志往回传，又可能找不到最优解。

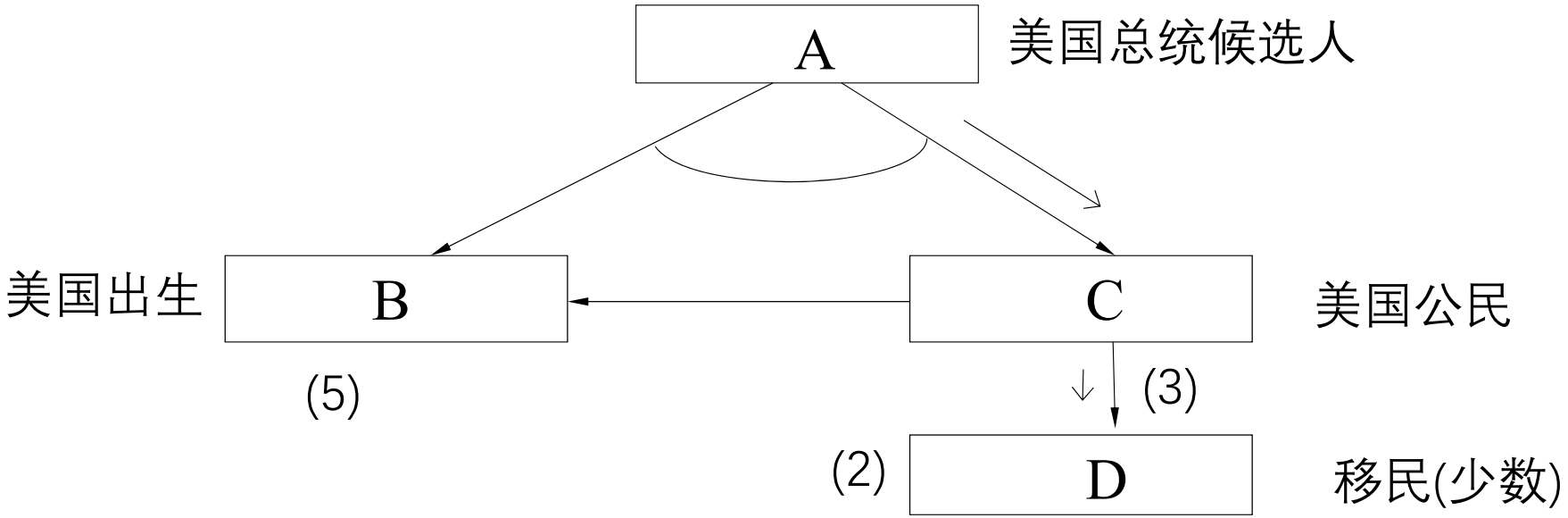


对AO*算法的进一步观察

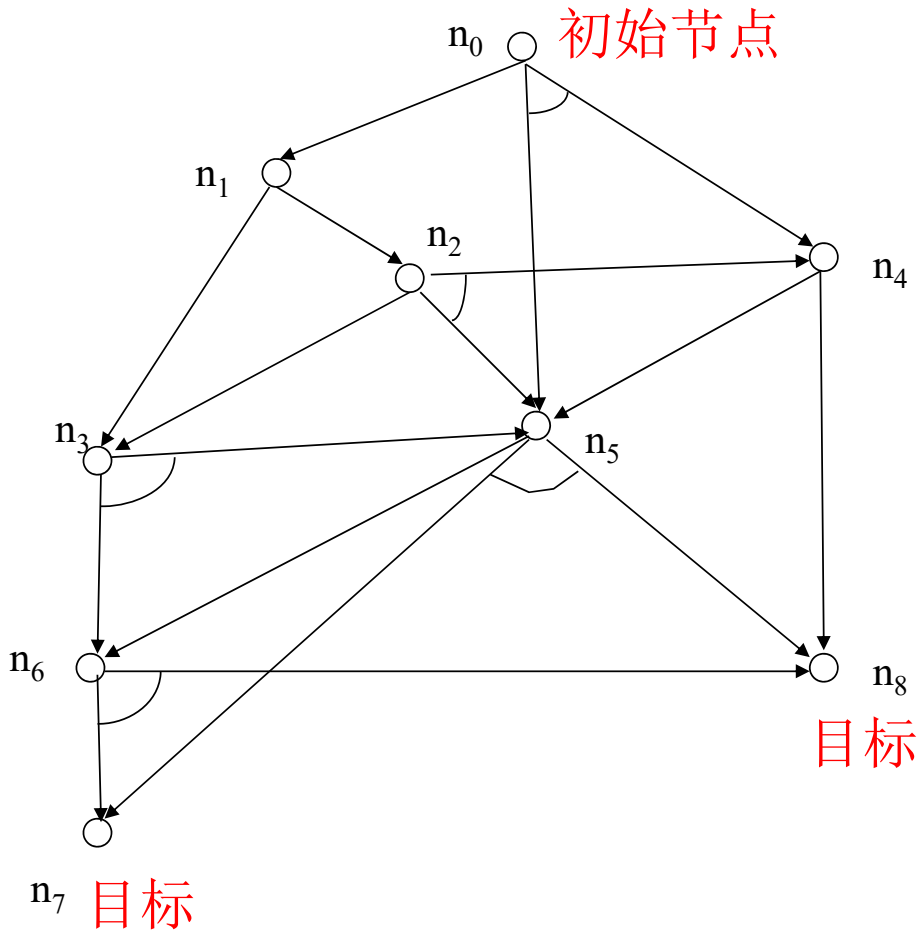
- 由H→G回传时,若只沿“→”标志往上传至C，再回至E, 则E仍为（6）而实际上应为（11），B仍为（13）而实际上应为（18）。这样导致A将又选择A→B这条路，实际这条路径比A→C更差。
- 所以,只顺路径标志往上传递修正的AO*算法不一定保证能找到最优解。

对AO*算法的进一步观察

2. 算法没有考虑子目标之间的相互依赖关系。



AO*算法举例



其中：

$h(n_0)=3$

$h(n_1)=2$

$h(n_2)=4$

$h(n_3)=4$

$h(n_4)=1$

$h(n_5)=1$

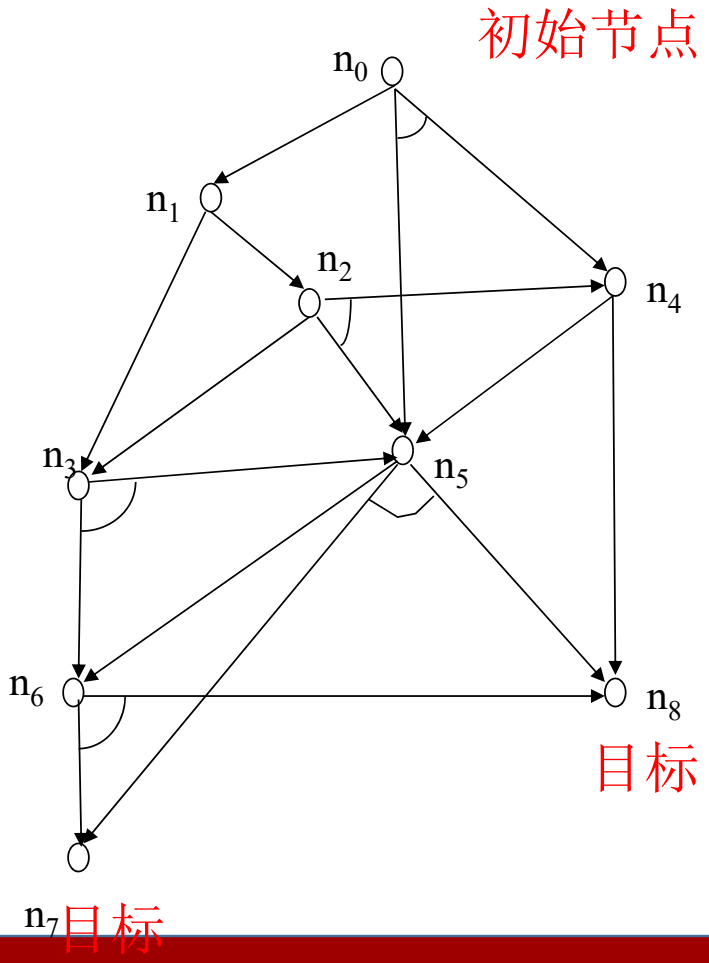
$h(n_6)=2$

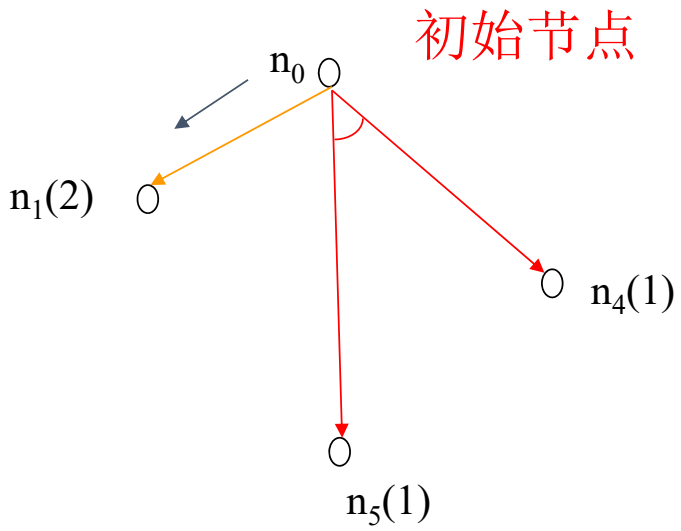
$h(n_7)=0$

$h(n_8)=0$

设：K连接符的耗散值为K

AO*算法举例

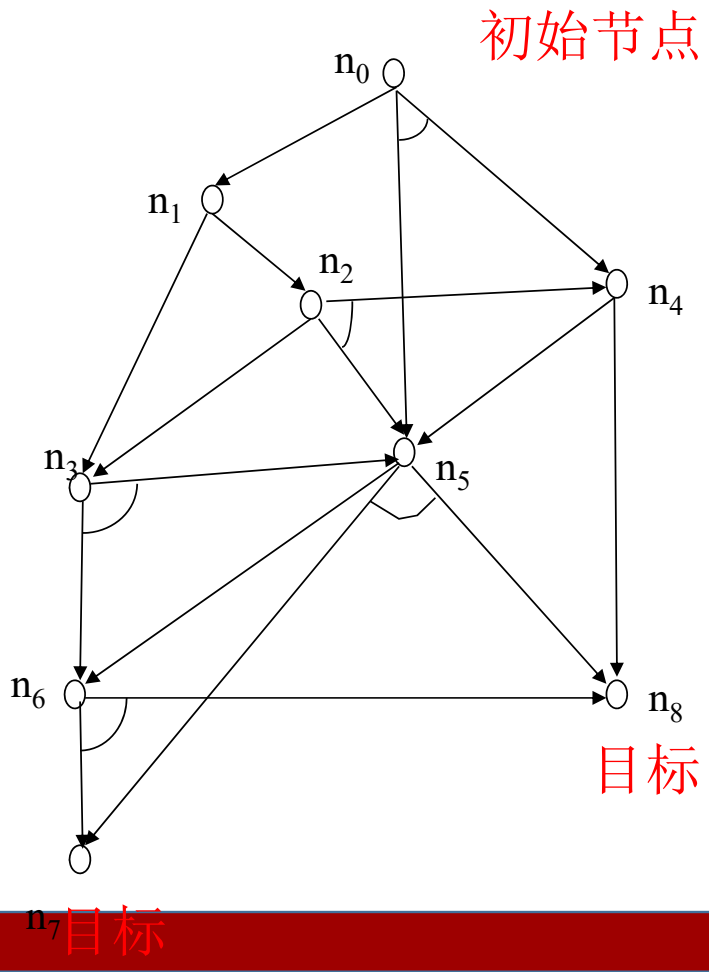


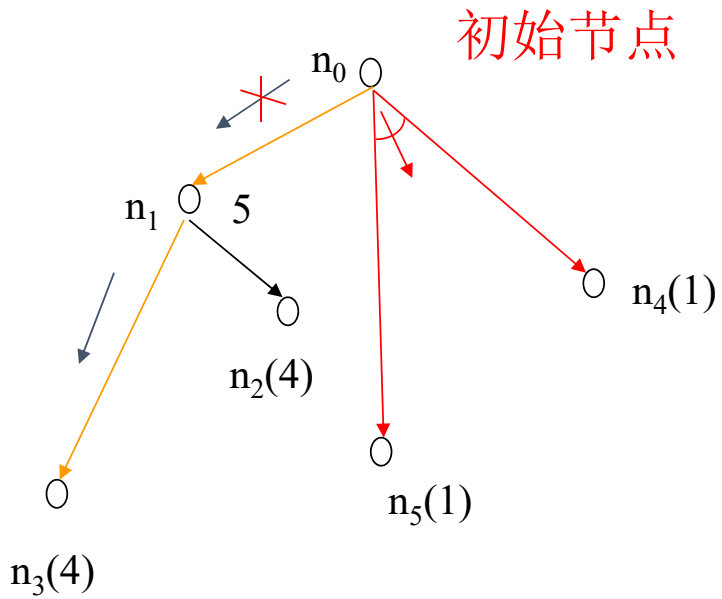


红色：4

黄色：3

AO*算法举例

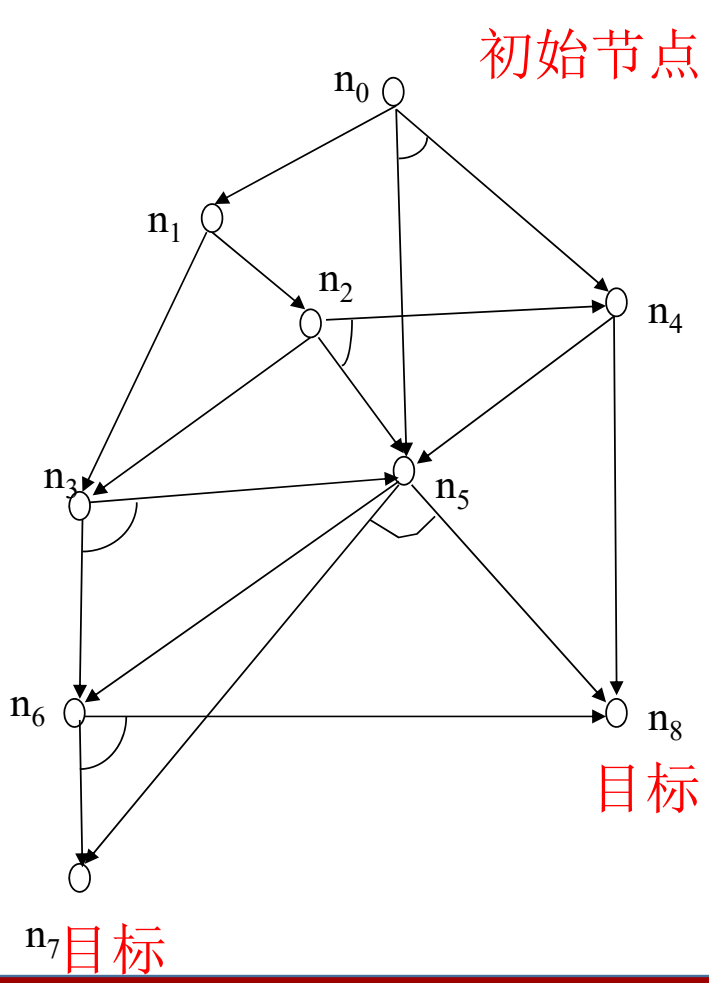


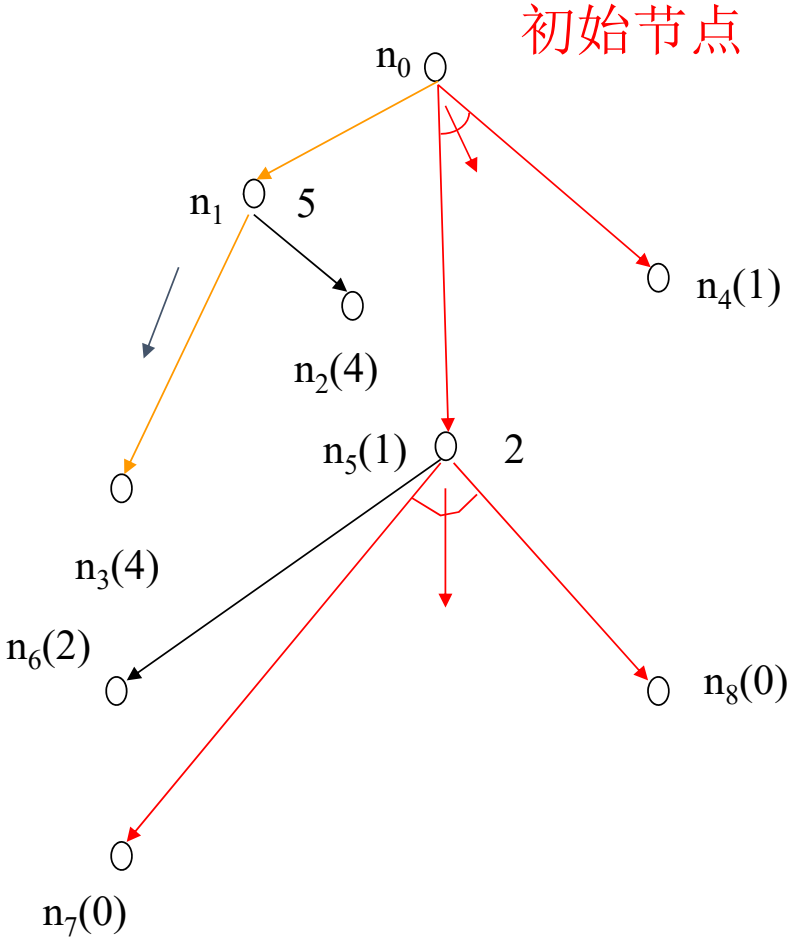


红色：4

黄色：6

AO*算法举例

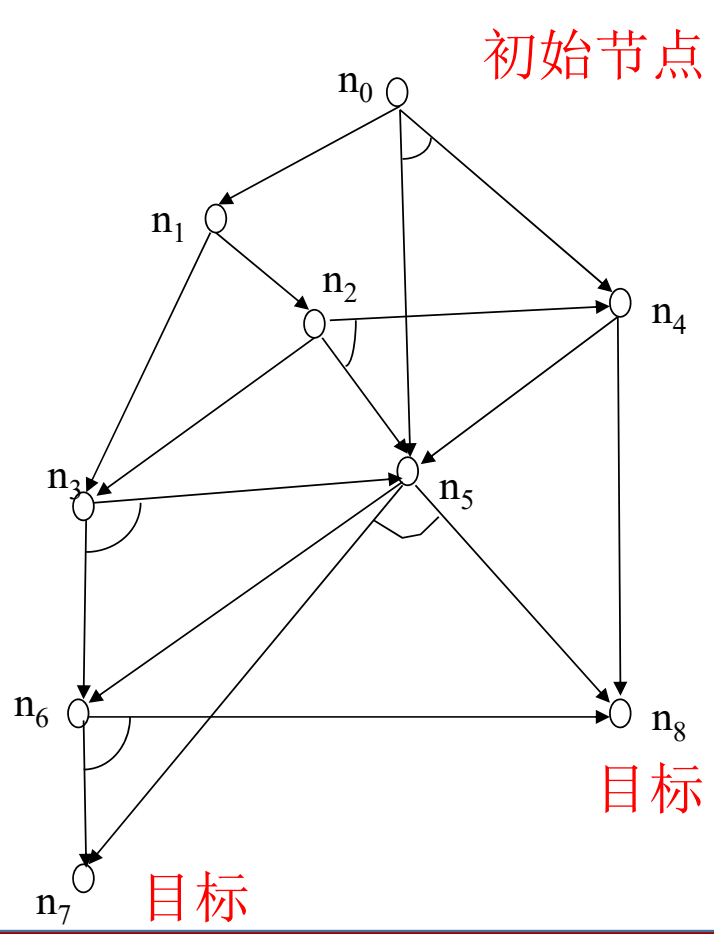


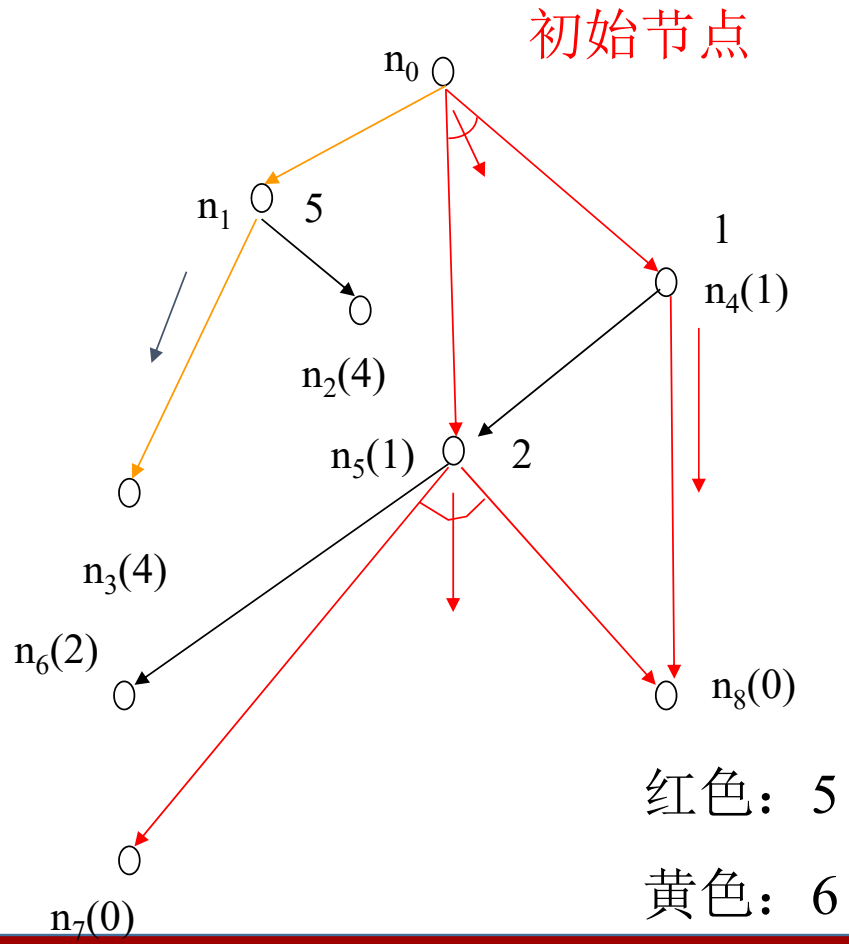


红色：5

黄色：6

AO*算法举例


初始节点
n_0
n_1
n_2
n_4
n_3
n_5
n_6
n_8
目标
n_7
目标


初始节点
n_0
n_1
5
1
$n_4(1)$
$n_2(4)$
$n_5(1)$
2
$n_3(4)$
$n_6(2)$
$n_8(0)$
$n_7(0)$
红色：5
黄色：6

用AO*算法求解一个智力问题

- 有这样一个智力问题：有12枚硬币，凡轻于或重于真币者，即为假币(只有一枚假币)，要设计一个搜索算法来识别假币并指出它是轻于还是重于真币，且利用天平的次数不多于3次。 演示
- 该问题的困难之处在于问题要求只称三次就要找到假币，否则就承认失败。如果称法不得当，使得留下的未知币太多，就不可能在3次内称出假币。 因此，每称一次，我们希望尽可能地得到关于假币的信息。

用AO*算法求解一个智力问题

- 问题的表示
  - 硬币可能有哪些状态
  - 每称一次之后，状态会发生何种变化
  - 每称一次后，必须保留所剩的使用天平的次数

- 硬币的重量

标准型 (Standard)　S

轻标准型（Light or Standard）LS

重标准型 (Heavy or Standard) HS

轻重标准型（Light or Heavy or Standard）LHS

用AO*算法求解一个智力问题

问题的表示

五元组：（lhs, ls, hs, s, t）

4种类型硬币的个数

所剩称硬币的次数

初始状态: (12,0,0,0,3）

目标状态: sg1: (0,1,0,11,0)

sg2: (0,0,1,11,0)

用AO*算法求解一个智力问题

- 利用AO*算法求解

  初始问题的描述: (12,0,0,0,3）

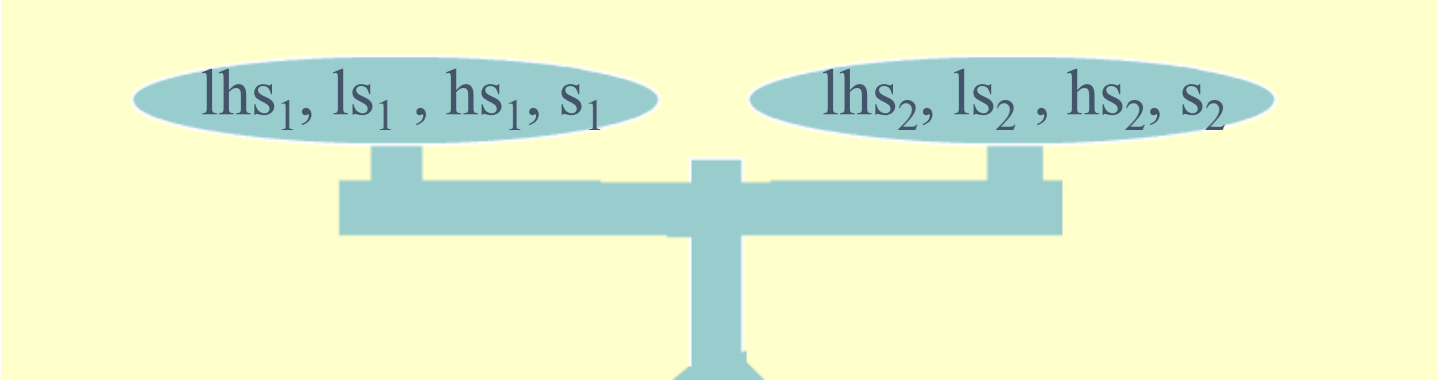
  一组将问题变换成子问题的变换规则

  一组本原问题描述: sg1: (0,1,0,11,0)

  sg2: (0,0,1,11,0)

用AO*算法求解一个智力问题

- 设当前的状态为 (lhs，ls，hs，s，t)
- 函数PICKUP([lhs_1, ls_1 , hs_1, s_1],[lhs_2, ls_2 , hs_2, s_2])
- 令PICKUP()等于-1,0,1 分别表示天平左倾斜、平衡和右倾斜



- $0<lhs_1+ls_1+hs_1+s_1=lhs_2+ls_2+hs_2+s_2<=6$

  $lhs_1+lhs_2<lhs \wedge ls_1+ls_2<=ls \wedge hs_1+hs_2<=hs \wedge s_1+s_2<=S$

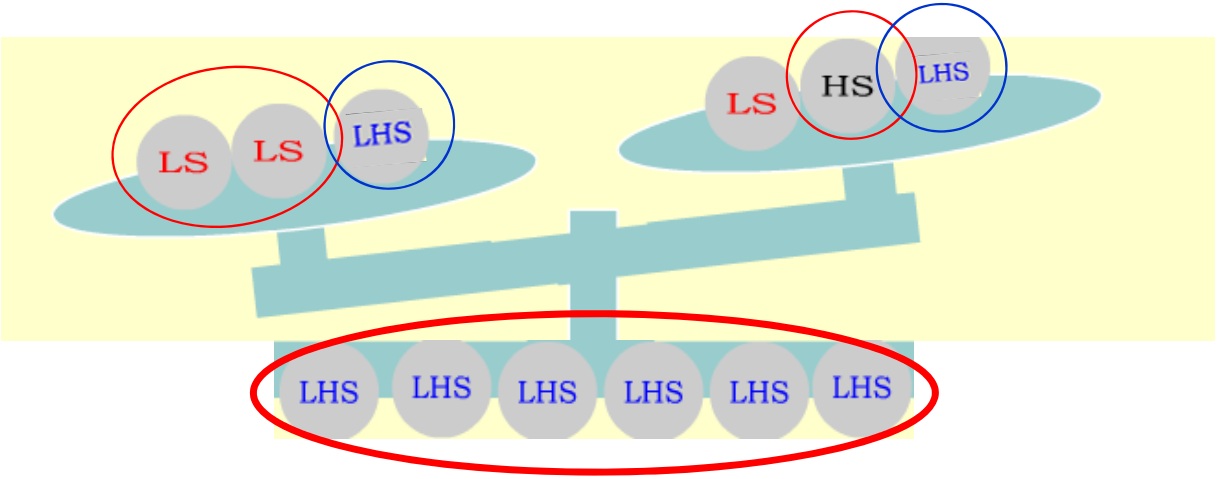
用AO*算法求解一个智力问题

左倾斜规则：

- if PICKUP ([lhs_1 , ls_1 , hs_1 , s_1],[lhs_2 , ls_2 , hs_2 , s_2]) = - 1 ∧ (lhs, ls , hs, s,t)
- then (lhs′, ls′ , hs′, s′, t-1)；
- 其中 s′= s+ **ls_1+hs_2+(lhs - lhs_1 -lhs_2)+(ls-ls_1-ls_2)+(hs-hs_1-hs_2)**

  =ls-ls_2 + hs-hs_1 + lhs - (lhs_1 +lhs_2) ；

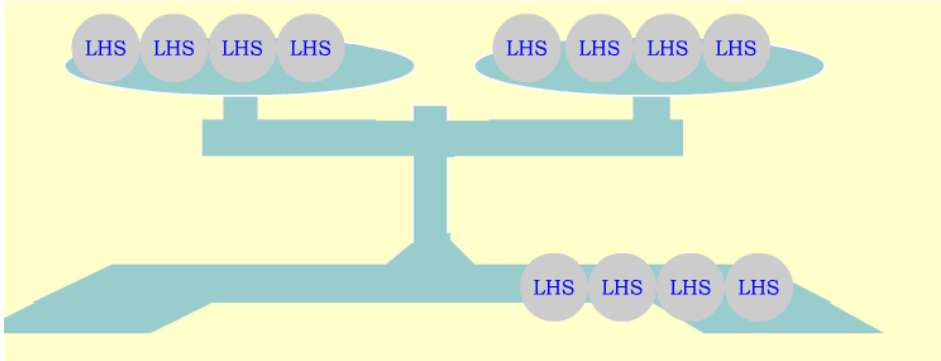
  ls′=ls_2 +**lhs_2** ； hs′=hs_1 +**lhs_1**； lhs′=0



用AO*算法求解一个智力问题

- 平衡规则：
  - if PICKUP（[lhs_1 , ls_1 , hs_1 , s_1],[lhs_2 , ls_2 , hs_2 , s_2])=0 ∧ (lhs, ls , hs, s,t)then (lhs′, ls′ , hs′, s′, t-1)；
  - 其中 s′= s+ls_1 +ls_2 +hs_1 +hs_2 + lhs_1 +lhs_2 ；

    ls′=ls-ls_1 -ls_2 ； hs′=hs-hs_1 -hs_2 ； lhs′=lhs-lhs_1 -lhs_2



用AO*算法求解一个智力问题

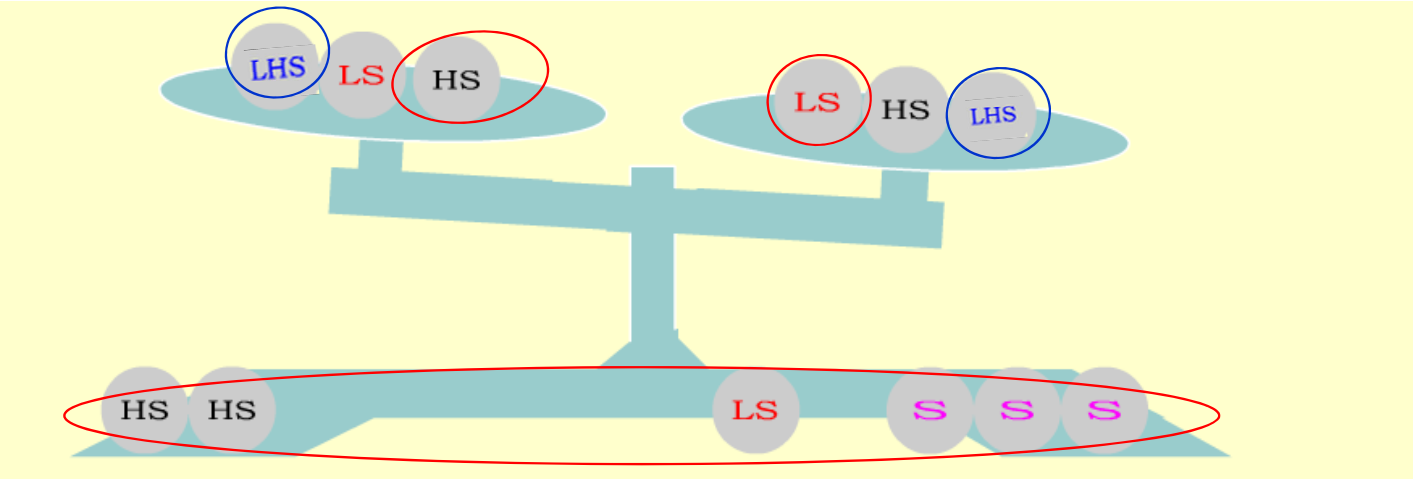
- **右倾斜规则：**

if PICKUP（[lhs_1, ls_1 , hs_1, s_1],[lhs_2, ls_2 , hs_2, s_2])=1∧ (lhs, ls , hs, s,t)　then (lhs′, ls′ , hs′, s′, t-1)；

其中 s′= s+ $\mathbf{hs_1+ls_2}$++$\mathbf{(lhs - lhs_1 -lhs_2)+(ls-ls_1-ls_2)+(hs-hs_1-hs_2)}$

=s+ ls-ls_1 + hs-hs_2 + lhs - (lhs_1+lhs_2)；

ls′=ls_1+lhs_1 ； hs′=lhs_2+hs_2； lhs′=0

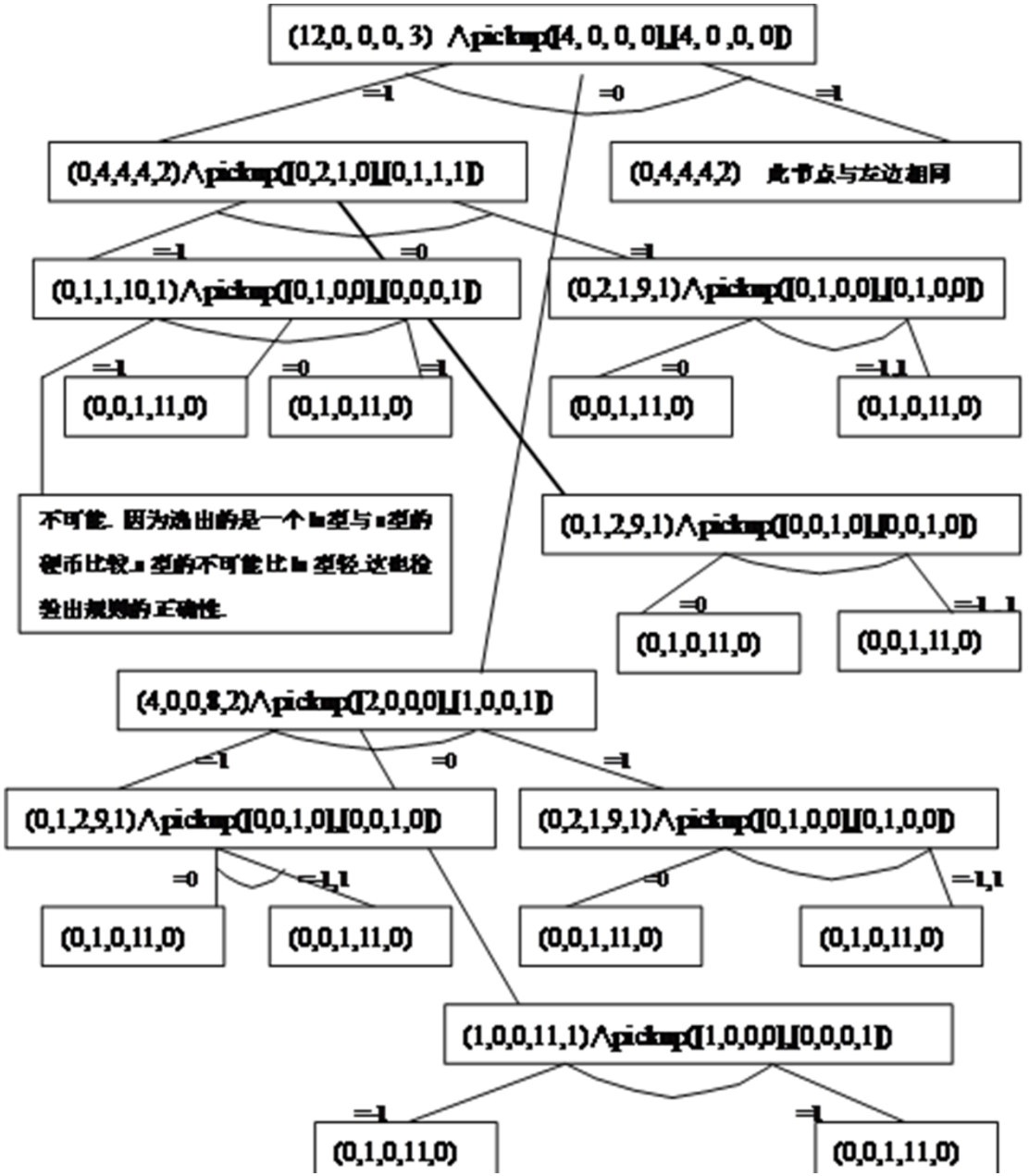


用AO*算法求解一个智力问题

搜索

用PICKUP（）填入不同的参数表示一种选取方法，不同的选取方法之间是或的关系，当选定一组PICKUP的参数后，就必须考虑它的值为-1,0和1时下层的结点都可解，则这三种情况之间的关系为“与”关系。这说明该搜索图为与或图。

- 评价函数h ((lhs,ls,hs,s,t))=ls+hs+lhs-1
  显然：　h ((0,1,0,11,0))=0
  　　h ((0,0,1,11,0))=0
- 即：h（ sg_1 ）=0；h（sg_2）=0


(12,0, 0,0, 3) Λpickup([4, 0, 0, 0]⊔[4, 0 ,0, 0])
=1
=0
=1
(0,4,4,4,2)Λpickup([0,2,1,0]⊔[0,1,1,1])
(0,4,4,4,2) 此节点与左边相同
=1
=0
=1
(0,1,1,10,1)Λpickup([0,1,0,0]⊔[0,0,0,1])
(0,2,1,9,1)Λpickup([0,1,0,0]⊔[0,1,0,0])
=1
=0
=1
=0
=1,1
(0,0,1,11,0)
(0,1,0,11,0)
(0,0,1,11,0)
(0,1,0,11,0)
不可能. 因为选出的是一个b型与a型的硬币比较.a型的不可能比b型轻.这也检验出规则的正确性.
(0,1,2,9,1)Λpickup([0,0,1,0]⊔[0,0,1,0])
=0
=1,1
(0,1,0,11,0)
(0,0,1,11,0)
(4,0,0,8,2)Λpickup([2,0,0,0]⊔[1,0,0,1])
=1
=0
=1
(0,1,2,9,1)Λpickup([0,0,1,0]⊔[0,0,1,0])
(0,2,1,9,1)Λpickup([0,1,0,0]⊔[0,1,0,0])
=0
=1,1
=0
=1,1
(0,1,0,11,0)
(0,0,1,11,0)
(0,0,1,11,0)
(0,1,0,11,0)
(1,0,0,11,1)Λpickup([1,0,0,0]⊔[0,0,0,1])
=1
=1
(0,1,0,11,0)
(0,0,1,11,0)

比较与分析

- L. Wos等人曾采用的方法是用子句描述了14个转换公理，然后在推理系统上使用超归结推理规则求解。所用的机器是IBM3033，运行时间22秒，求得40多种不同的解.
- 采用AO*搜索方法，只用了三个转换规则，算法在586微机上实现，运行时间1秒，求得410种不同的解，这410种解不包括对称的情况.
- 比较之下，用AO*算法求解这个智能问题解法比较简洁，且效率比较高.
- AO*算法之所以可以很快的找出所有解，首先是因为该问题用与或图表示比较恰当；其次是AO*算法可以利用与或图的特点和启发式方法可以避免许多无用路径的搜索.