第4章 搜索
for Problem Solving

4.1.1 搜索的含义

概念：

依靠经验，利用已有知识，根据问题的实际情况，不断寻找可利用知识，逐步摸索求解，从而构造一条代价最小的路线，使问题得以解决的过程称为搜索

适用情况：

不良结构或非结构化问题；难以获得求解所需的全部信息；没有现成的算法可供求解使用。

一些问题，结构良好，理论可解，实际无法付诸实用。

4.1.1 搜索的含义

搜索的类型

按是否使用启发式信息：

盲目搜索：按预定的控制策略进行搜索，在搜索过程中获得的中间信息并不改变控制策略。

启发式搜索：在搜索中加入了与问题有关的启发性信息，用于指导搜索朝着最有希望的方向前进，加速问题的求解过程并找到最优解。

按问题的表示方式：

状态空间搜索：用状态空间法来求解问题所进行的搜索

与或树搜索：用问题归约法来求解问题时所进行的搜索

……

4.1.2 状态空间问题求解方法

1. 状态空间问题表示

状态(State)

是表示问题求解过程中每一步问题状况的数据结构，它可形式地表示为：

$$S_k=\{S_{k0}, S_{k1}, \cdots\}$$

当对每一个分量都给以确定的值时，就得到了一个具体的状态。

操作(Operator)

也称为算符，它是把问题从一种状态变换为另一种状态的手段。它可理解为状态集合上的一个函数，它描述了状态之间的关系。

4.1.2 状态空间问题求解方法

1. 状态空间问题表示

状态空间(State space)

用来描述一个问题的全部状态以及这些状态之间的相互关系。常用一个三元组表示为：

(S, F, G)

其中，**S**为问题的所有初始状态集合；**F**为操作的集合；**G**为目标状态的集合。

状态空间也可用一个赋值的有向图来表示，该有向图称为状态空间图。在状态空间图中，节点表示问题的状态，有向边表示操作。

4.1.2 状态空间问题求解方法

2. 状态空间问题求解

状态空间法求解问题的基本过程：

首先，为问题选择适当的“状态”及“操作”的形式化描述方法；

然后，从某个初始状态出发，每次使用一个“操作”，递增地建立起操作序列，直到达到目标状态为止；

最后，由初始状态到目标状态所使用的算符序列就是该问题的一个解。

4.1.2 状态空间问题求解方法

3. 状态空间的例子(1/12)

例4.1 二阶梵塔问题

设有三根钢针，它们的编号分别是1号、2号和3号。在初始情况下，1号钢针上穿有A、B两个金片，A比B小，A位于B的上面。要求把这两个金片全部移到另一根钢针上，而且规定每次只能移动一个金片，任何时刻都不能使大的位于小的上面。

解：设用$S_k=(S_{kA}, S_{kB})$表示问题的状态，其中，S_{kA}表示金片A所在的钢针号，S_{kB}表示金片B所在的钢针号。

全部可能的问题状态共有以下9种：

$S_0=(1, 1)$ $S_1=(1, 2)$ $S_2=(1, 3)$ $S_3=(2, 1)$ $S_4=(2, 2)$

$S_5=(2, 3)$ $S_6=(3, 1)$ $S_7=(3, 2)$ $S_8=(3, 3)$

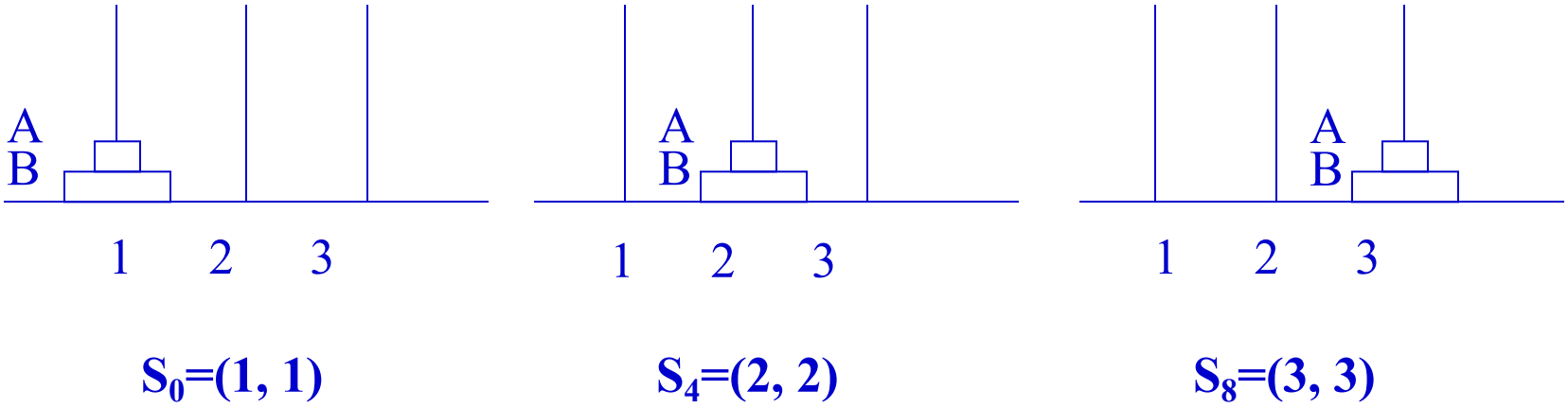
4.1.2 状态空间问题求解方法

3. 状态空间的例子(2/12)

初始状态集合　　$S=\{S_0\}$

目标状态集合　　$G=\{S_4, S_8\}$

初始状态S_0和目标状态S_4、S_8如下图



$S_0=(1, 1)$　　$S_4=(2, 2)$　　$S_8=(3, 3)$

图4.1 二阶梵塔问题的初始状态和目标状态

4.1.2 状态空间问题求解方法

3. 状态空间的例子(3/12)

操作

A_{ij} 表示把金片A从第i号钢针移到j号钢针上；

B_{ij} 表示把金片B从第i号钢针移到第j号钢针上。

共有12种操作，它们分别是：

A_{12} A_{13} A_{21} A_{23} A_{31} A_{32}

B_{12} B_{13} B_{21} B_{23} B_{31} B_{32}

根据上述9种可能的状态和12种操作，可构成二阶梵塔问题的状态空间图，如下图所示。

4.1.2 状态空间问题求解方法

3. 状态空间的例子(4/12)

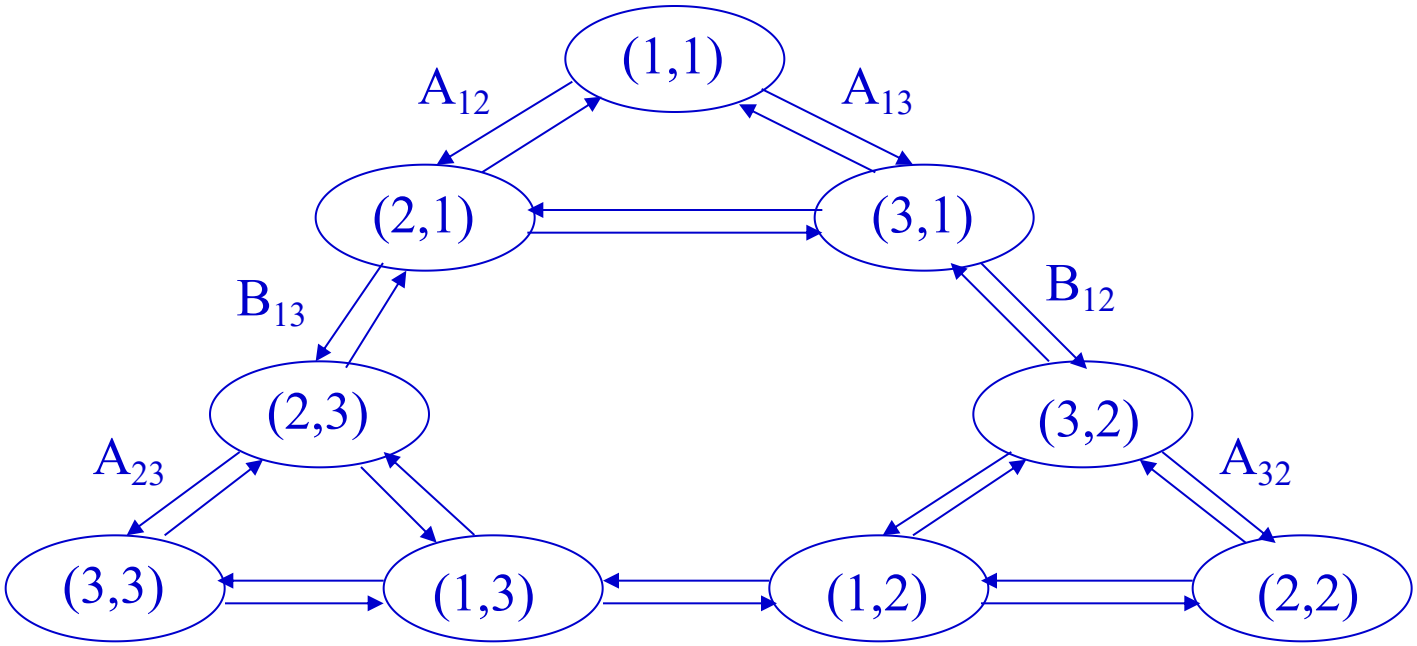


图4.2 二阶梵塔的状态空间图

从初始节点(1, 1)到目标节点(2, 2)及(3, 3)的任何一条路径都是问题的一个解。其中，最短的路径长度是3，它由3个操作组成。例如，从(1, 1)开始，通过使用操作A_{13}、B_{12}及A_{32}，可到达(2, 2)。

4.1.2 状态空间问题求解方法

3. 状态空间的例子(5/12)

例4.2 修道士(Missionaries)和野人(Cannibals)问题(简称M-C问题)

设在河的一岸有3个野人、3个修道士和1条船，修道士想用这条船把所有的人运到河对岸，但受以下条件的约束：

第一，修道士和野人都会划船，但每次船上至多可载2个人；

第二，在河的任一岸，如果野人数目超过修道士数，修道士会被野人吃掉。

如果野人会服从任何一次过河安排，请规划一个确保修道士和野人都能过河，且没有修道士被野人吃掉的安全过河计划。

4.1.2 状态空间问题求解方法

3. 状态空间的例子(6/12)

解：先选取描述问题状态的方法。这里，需要考虑两岸的修道士人数和野人数，还需要考虑船在左岸还是在右岸，故可用如下三元组来表示状态

S=(m, c, b)

其中，**m**表示左岸的修道士人数，**c**表示左岸的野人数，**b**表示左岸的船数。而右岸的状态可由下式确定：

右岸修道士数 **m'=3-m**

右岸野人数 **c'=3-c**

右岸船数 **b'=1-b**

在这种表示方式下，**m**和**c**都可取**0**、**1**、**2**、**3**中之一，**b**可取**0**和**1**中之一。因此，共有**4×4×2=32**种状态。

4.1.2 状态空间问题求解方法

3. 状态空间的例子(7/12)

初始状态

目标状态

(3,3,1)

(0,0,0)

4.1.2 状态空间问题求解方法

3. 状态空间的例子(8/12)

有效状态

在32种状中，除去不合法和修道士被野人吃掉的状态，有效状态只16种：

$S_0=(3, 3, 1)$ $S_1=(3, 2, 1)$ $S_2=(3, 1, 1)$ $S_3=(2, 2, 1)$

$S_4=(1, 1, 1)$ $S_5=(0, 3, 1)$ $S_6=(0, 2, 1)$ $S_7=(0, 1, 1)$

$S_8=(3, 2, 0)$ $S_9=(3, 1, 0)$ $S_{10}=(3, 0, 0)$ $S_{11}=(2, 2, 0)$

$S_{12}=(1, 1,0)$ $S_{13}=(0, 2, 0)$ $S_{14}=(0, 1, 0)$ $S_{15}=(0, 0, 0)$

过河操作

过河操作是指用船把修道士或野人从河的左岸运到右岸，或从右岸运到左岸的动作。每个操作都应当满足如下条件：

第一，船上至少有一个人（m或c）操作，离开岸边的m和c的减少数目应该等于到达岸边的m和c的增加数目；

第二，每次操作船上人数不得超过2个；

第三，操作应保证不产生非法状态。

4.1.2 状态空间问题求解方法

3. 状态空间的例子(9/12)

操作的表示

L_{ij} 表示有i个修道士和j个野人，从左岸到右岸的操作

R_{ij} 表示有i个修道士和j个野人，从右岸到左岸的操作

操作集

本问题有10种操作可供选择，他们的集合称为操作集，即

$A=\{L_{01}, L_{10}, L_{11}, L_{02}, L_{20}, R_{01}, R_{10}, R_{11}, R_{02}, R_{20}\}$

操作的例子

下面以L_{01}和R_{01}为例来说明这些操作的条件和动作。

操作符号	条件	动作
L_{01}	b=1, m=0或3, c≥1	b=0, c=c-1
R_{01}	b=0, m=0或3，c≤2	b=1, c=c+1

4.1.2 状态空间问题求解方法

3. 状态空间的例子(10/12)

例4.3 猴子摘香蕉问题。

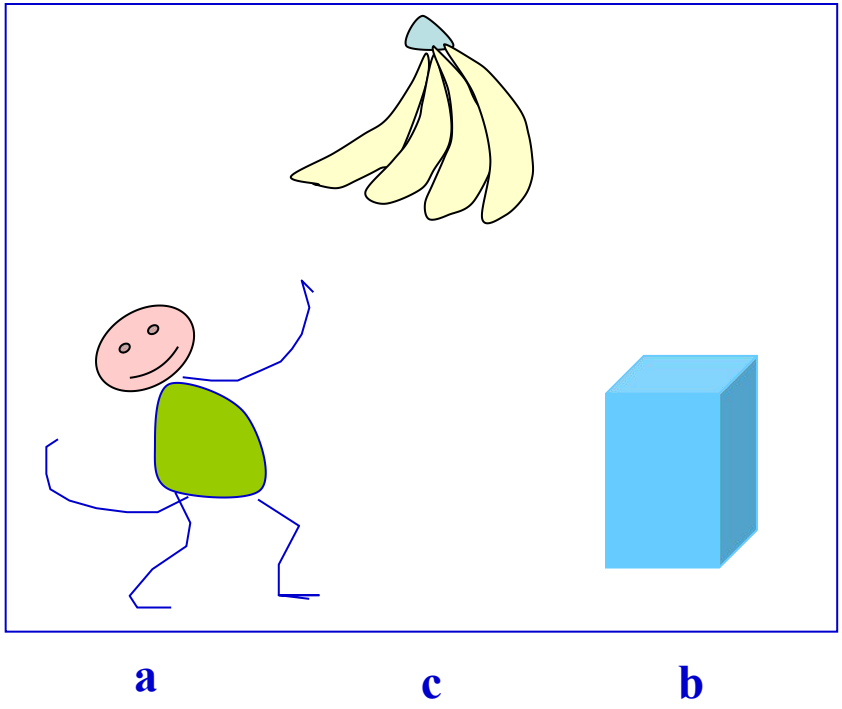
解：问题的状态可用4元组（w, x, y, z）表示。其中：

w 表示猴子的水平位置；

x 表示箱子的水平位置；

y 表示猴子是否在箱子上，当猴子在箱子上时，y取1；否则y取0；

z 表示猴子是否拿到香蕉，当拿到香蕉时z取1，否则z取0。



4.1.2 状态空间问题求解方法

3. 状态空间的例子(11/12)

所有可能的状态

S_0: (a, b, 0, 0)　初始状态

S_1: (b, b, 0, 0)　　(b, b, 1, 0)　(a, a, 0, 0)　(b, a, 0, 0) .....

S_2: (c, c, 0, 0)

S_3: (c, c, 1, 0)

S_4: (c, c, 1, 1)　目标状态

允许的操作为

Goto(u)：猴子走到位置u，即

(w, x, 0, 0)→(u, x, 0, 0)

Pushbox(v): 猴子推着箱子到水平位置v，即

(x, x, 0, 0)→(v, v, 0, 0)

Climbbox: 猴子爬上箱子，即

(x, x, 0, 0)→(x, x, 1, 0)

Grasp；猴子拿到香蕉，即

(c, c, 1, 0)→(c, c, 1, 1)

问题的状态空间图

如下图所示。可见，由初始状态到目标状态的操作序列为：

{Goto(b), Pushbox(c), Climbbox, Grasp}

4.1.2 状态空间问题求解方法

3. 状态空间的例子(12/12)

猴子摘香蕉问题的状态空间图

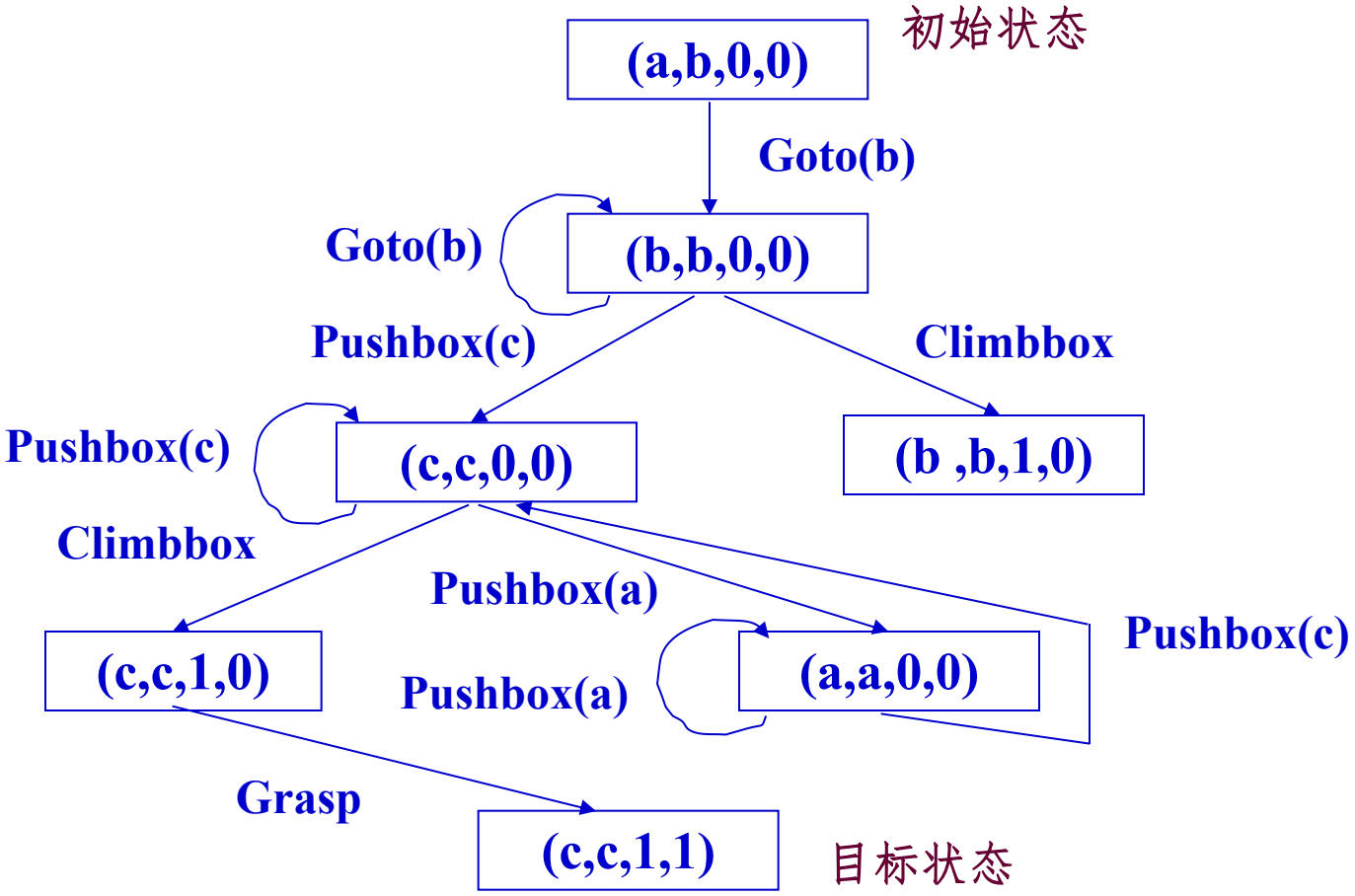


图4.3 猴子摘香蕉问题的状态空间图

4.1.3 问题归约求解方法

1. 问题的分解与等价变换

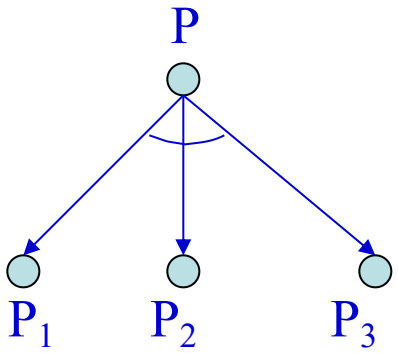
基本思想

当一问题较复杂时，可通过分解或变换，将其转化为一系列较简单的问题，然后通过对这些简单问题的求解来实现对原问题的求解。

分解

如果一个问题**P**可以归约为一组子问题$P_1,P_2,\ldots,P_n$，并且只有当所有子问题P_i都有解时原问题**P**才有解，任何一个子问题P_i无解都会导致原问题**P**无解，则称此种归约为问题的分解。

即分解所得到的子问题的“与”与原问题**P**等价。



与树 分解

4.1.3 问题归约求解方法

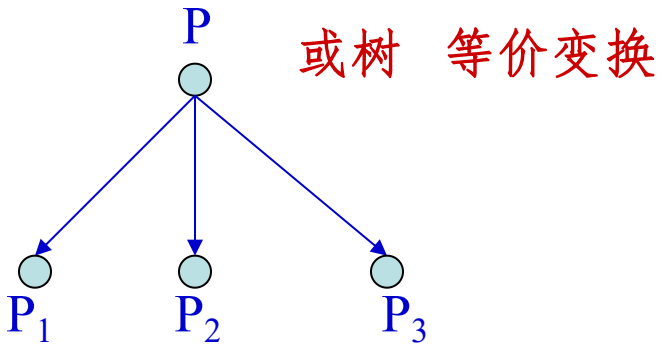
1. 问题的分解与等价变换

等价变换

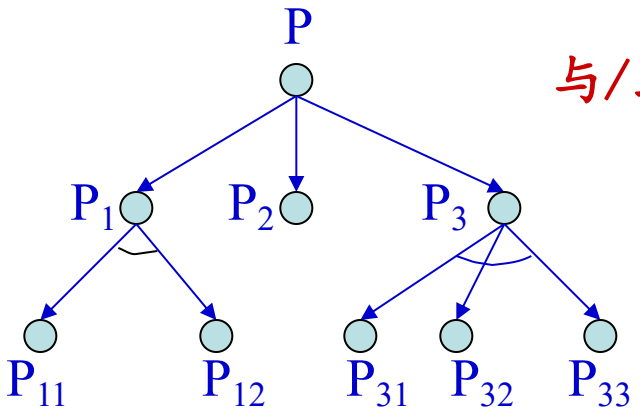
如果一个问题**P**可以归约为一组子问题$P_1,P_2,\ldots,P_n$，并且子问题P_i中只要有一个有解则原问题**P**就有解，只有当所有子问题P_i都无解时原问题**P**才无解，称此种归约为问题的等价变换，简称变换。

即变换所得到的子问题的“或”与原问题**P**等价。

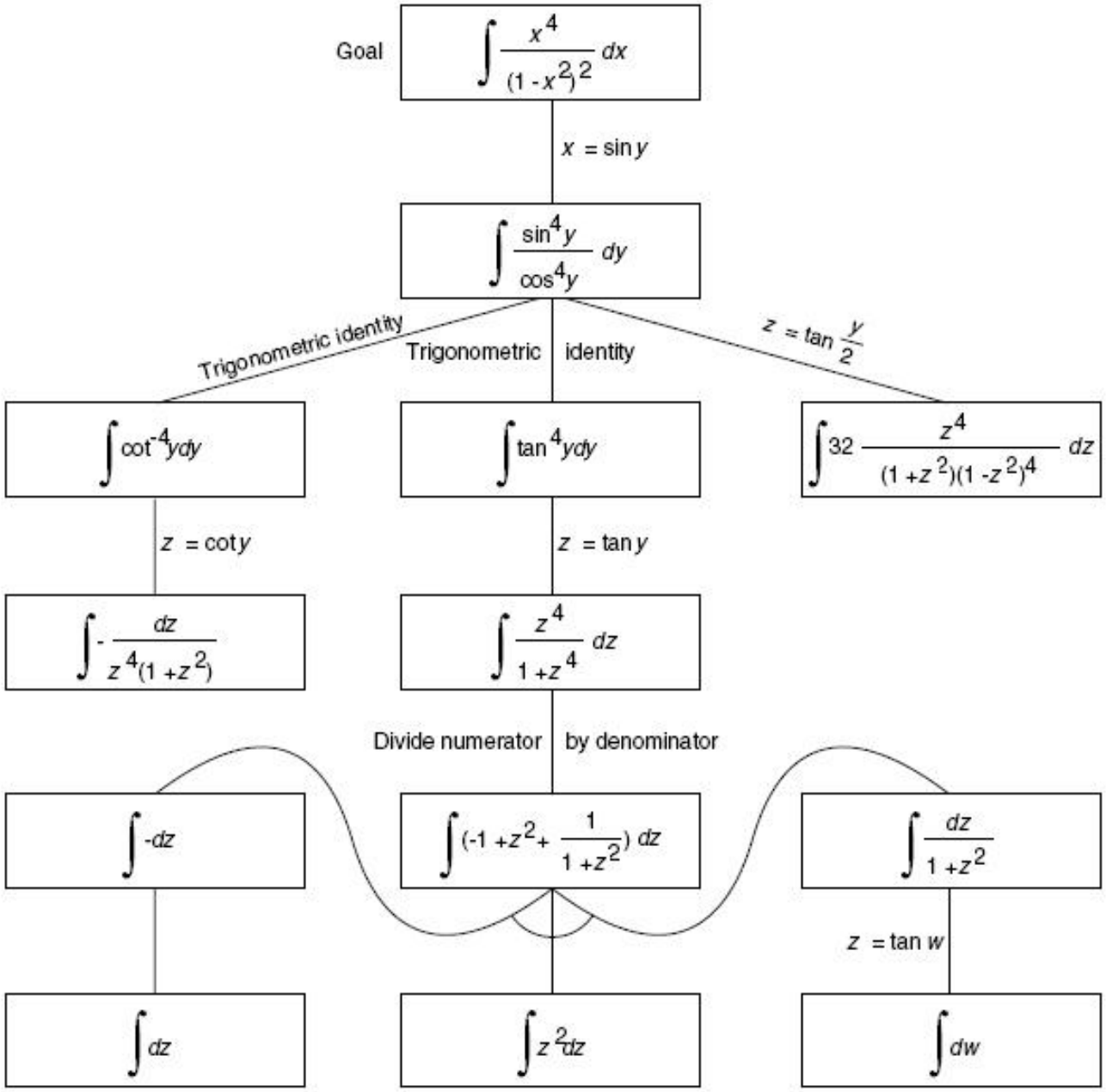
实际问题归约过程中，可能要同时采用变换和分解。将原问题归约为一系列本原问题，本原问题指那些不能或不需要再进行分解或变换，且可以直接解答的问题。



或树 等价变换



与/或树


Goal
$\int \frac{x^4}{(1-x^2)^2}\,dx$
$x = \sin y$
$\int \frac{\sin^4 y}{\cos^4 y}\,dy$
Trigonometric identity
Trigonometric identity
$z = \tan\frac{y}{2}$
$\int \cot^{-4} y\,dy$
$\int \tan^4 y\,dy$
$\int 32\,\frac{z^4}{(1+z^2)(1-z^2)^4}\,dz$
$z = \cot y$
$z = \tan y$
$\int -\frac{dz}{z^4(1+z^2)}$
$\int \frac{z^4}{1+z^4}\,dz$
Divide numerator by denominator
$\int -dz$
$\int (-1+z^2+\frac{1}{1+z^2})\,dz$
$\int \frac{dz}{1+z^2}$
$z = \tan w$
$\int dz$
$\int z^2 dz$
$\int dw$

4.1.3 问题归约求解方法

2. 问题归约的与/或树表示

(4) 端节点与终止节点

在与/或树中，没有子节点的节点称为端节点；本原问题所对应的节点称为终止节点。可见，终止节点一定是端节点，但端节点却不一定是终止节点。

4.1.3 问题归约求解方法

2. 问题归约的与/或树表示

(5) 可解节点与不可解节点

在与/或树中，满足以下三个条件之一的节点为可解节点：

①任何终止节点都是可解节点。

②对“或”节点，当其子节点中至少有一个为可解节点时，则该或节点就是可解节点。

③对“与”节点，只有当其子节点全部为可解节点时，该与节点才是可解节点。

同样，可用类似的方法定义不可解节点：

①不为终止节点的端节点是不可解节点。

②对“或”节点，若其全部子节点都为不可解节点，则该或节点是不可解节点。

③对“与”节点，只要其子节点中有一个为不可解节点，则该与节点是不可解节点。

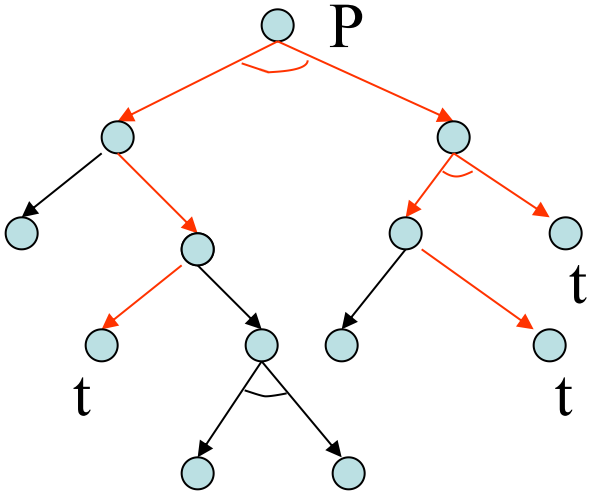
4.1.3 问题归约求解方法

2. 问题归约的与/或树表示

(6) 解树

由可解节点构成，并且由这些可解节点可以推出初始节点（它对应着原始问题）为可解节点的子树为解树。在解树中一定包含初始节点。

例如，右图给出的与或树中，用红 线表示的子树是一个解树。在该图中，节点**P**为原始问题节点，用**t**标出的节点是终止节点。根据可解节点的定义，很容易推出原始问题**P**为可解节点。



问题归约求解过程就实际上就是生成解树，即证明原始节点是可解节点的过程。这一过程涉及到搜索的问题，对于与/或树的搜索将在后面详细讨论。

4.1.3 问题归约求解方法

3. 问题归约的例子

例4.4 三阶梵塔问题。要求把1号钢针上的3个金片全部移到3号钢针上，如下图所示。

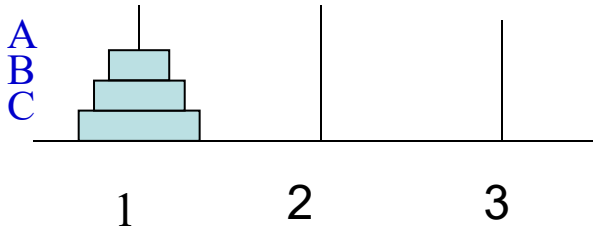
解： 这个问题也可用状态空间法来解，不过本例主要用它来说明如何用归约法来解决问题。

为了能够解决这一问题，首先需要定义该问题的形式化表示方法。设用三元组 **(i, j, k)** 表示问题在任一时刻的状态，用“→”表示状态的转换。上述三元组中

i 代表金片**A**所在的钢针号

j 代表金片**B**所在的钢针号

k 代表金片**C**所在的钢针号



利用问题归约方法，原问题可分解为以下三个子问题：

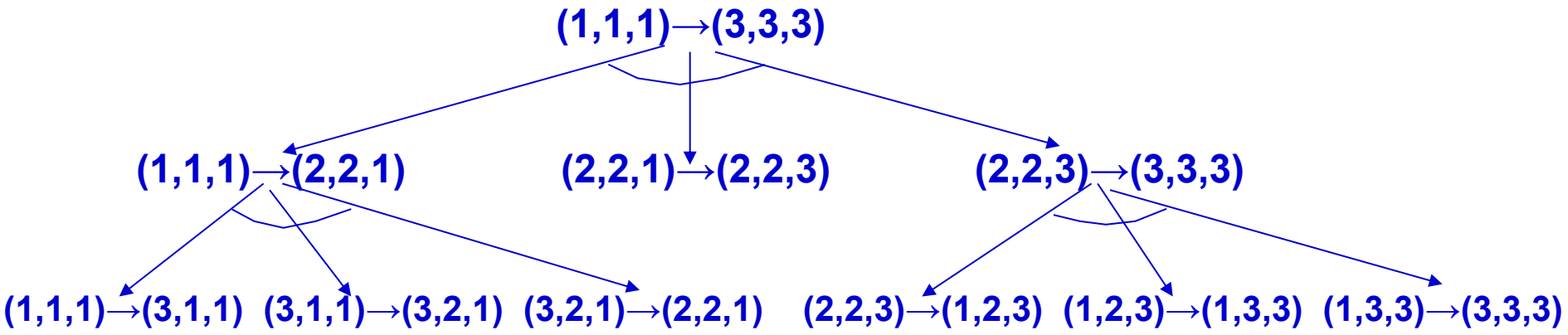
(1) 把金片A及B移到2号钢针上的双金片移动问题。即(1, 1, 1)→(2, 2, 1)

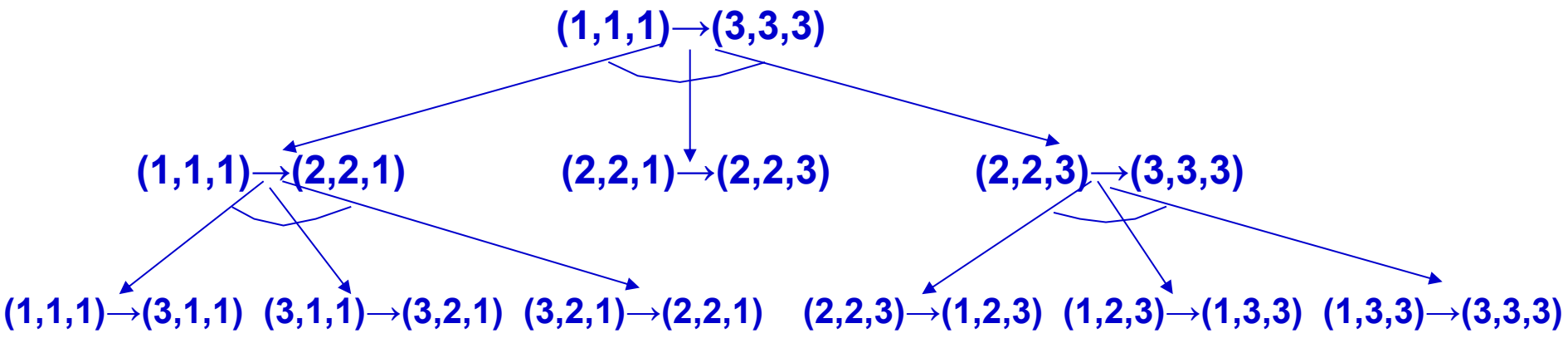
(2) 把金片C移到3号钢针上的单金片移动问题。即 (2, 2, 1)→(2, 2, 3)

(3) 把金片A及B移到3号钢针的双金片移动问题。即(2, 2, 3)→((3, 3, 3)

其中，子问题(1)和(3)都是一个二阶梵塔问题，它们都还可以再继续进行分解；子问题(2)是本原问题，它已不需要再分解。

三阶梵塔问题的分解过程可用如下图与/或树来表示





在该与/或树中，有7个终止节点，它们分别对应着7个本原问题。如果把这些本原问题从左至右排列起来，即得到了原始问题的解：

(1, 1, 1)→(3, 1, 1)　(3, 1, 1)→(3, 2, 1)　(3, 2, 1)→(2, 2, 1)

(2, 2, 1)→(2, 2, 3)　(2, 2, 3)→(1, 2, 3)　(1, 2, 3)→(1, 3, 3)

(1, 3, 3)→(3, 3, 3)

第4章 搜索策略

4.1 搜索概述

4.x 无信息(盲目)搜索

1.状态空间的盲目搜索

2.代价树的盲目搜索

4.2 状态空间的有信息（启发式）搜索

4.3 与/或树的启发式搜索

4.4 博弈树的启发式搜索

4.5 智能搜索算法（遗传算法、DE、SI）

1. 状态空间的盲目搜索

基本思想

状态空间搜索的基本思想

先把问题的初始状态作为当前扩展节点对其进行扩展，生成一组子节点，然后逐个检查问题的目标状态是否出现在这些子节点中。若出现，则搜索成功，找到了问题的解；若没出现，则继续扩展。

重复上述过程，直到目标状态出现或者没有可供操作的节点为止。所谓对一个节点进行“扩展”是指对该节点用某个可用操作进行作用，生成该节点的一组子节点。

算法的数据结构和符号约定

Open表：用于存放刚生成的节点

Closed表：用于存放已经扩展或将要扩展的节点

若子节点的状态是已发现的（已在**Open**或**Closed**表中），将被丢弃

S_0：表示问题的初始状态

1. 状态空间的盲目搜索

广度优先搜索（1/4）

基本思想

从初始节点S_0开始逐层向下扩展，在第n层节点还没有全部搜索完之前，不进入第n+1层节点的搜索。Open表中的节点总是按进入的先后排序，先进入的节点排在前面，后进入的节点排在后面。

搜索算法

(1)把初始节点S_0放入Open表中；

(2)如果Open表为空，则问题无解，失败退出；

(3)把Open表的第一个节点取出放入Closed表，并记该节点为n；

(4)考察n是否为目标节点。若是，则得到问题的解，成功退出；

(5)若节点n不可扩展，则转第(2)步；

(6)扩展节点n，将其子节点放入Open表的尾部，并为每一个子节点设置指向父节点的指针，然后转第(2)步。

1. 状态空间的盲目搜索

广度优先搜索（2/4）

例 八数码难题。在3×3的方格棋盘上，分别放置了表有数字1、2、3、4、5、6、7、8的八张牌，初始状态S_0，目标状态S_g，如下图所示。可以使用的操作有

空格左移，空格上移，空格右移，空格下移

即只允许把位于空格左、上、右、下方的牌移入空格。要求应用广度优先搜索策略寻找从初始状态到目标状态的解路径。

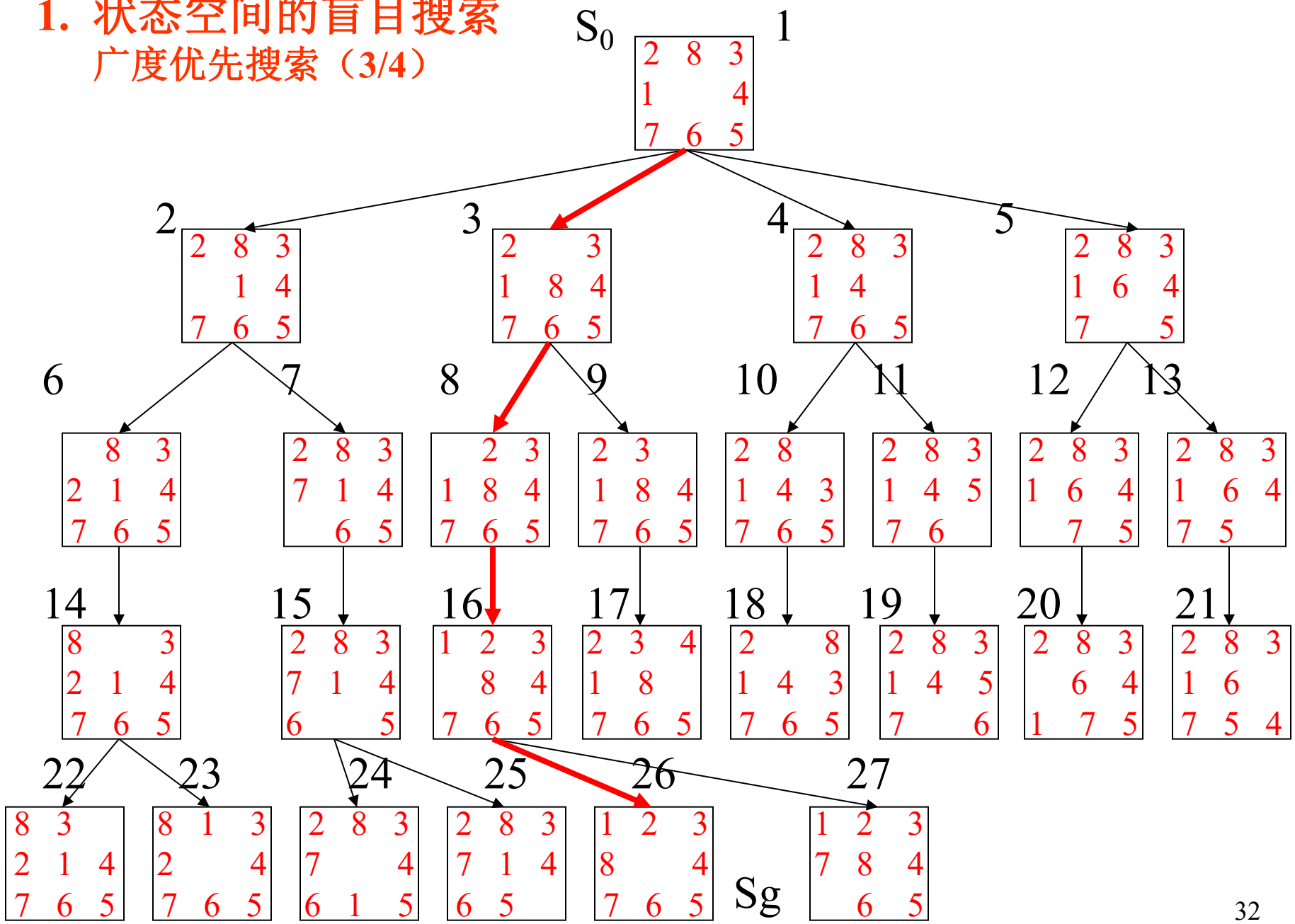
S_0

2	8	3
1		4
7	6	5

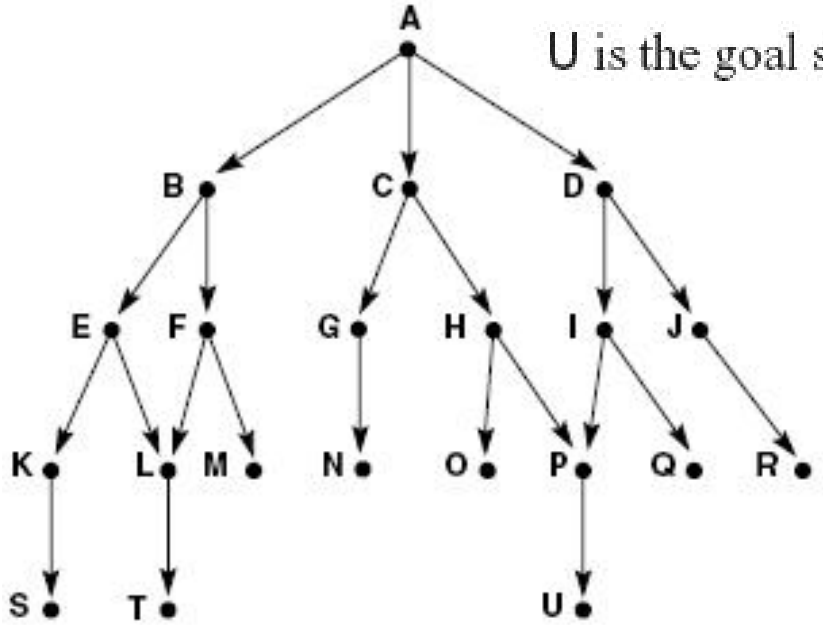
S_g

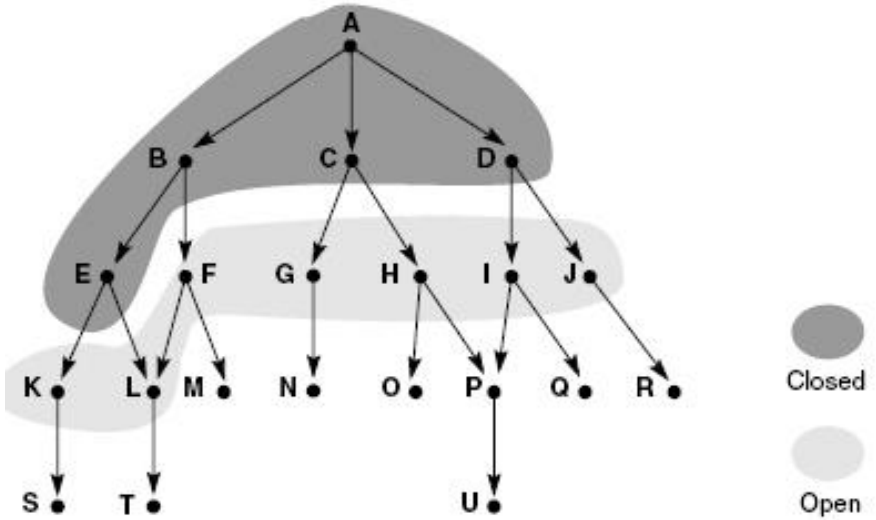
1	2	3
8		4
7	6	5

1. 状态空间的盲目搜索
广度优先搜索（3/4）



另一个广度优先搜索例子





Graph at iteration 6

1. open = [A]; closed = [ ]
2. open = [B,C,D]; closed = [A]
3. open = [C,D,E,F]; closed = [B,A]
4. open = [D,E,F,G,H]; closed = [C,B,A]
5. open = [E,F,G,H,I,J]; closed = [D,C,B,A]
6. open = [F,G,H,I,J,K,L]; closed = [E,D,C,B,A]
7. open = [G,H,I,J,K,L,M] (as L is already on open); closed = [F,E,D,C,B,A]
8. open = [H,I,J,K,L,M,N]; closed = [G,F,E,D,C,B,A]
9. and so on until either U is found or open = [ ].

1. 状态空间的盲目搜索

广度优先搜索（4/4）

优点：

广度优先搜索是一种完备策略，即只要问题有解，它就一定可以找到解。并且，广度优先找到的解，还一定是路径最短的解。

缺点：

盲目性较大，尤其是当目标节点距初始节点较远时，将产生许多无用的节点，因此搜索效率较低。

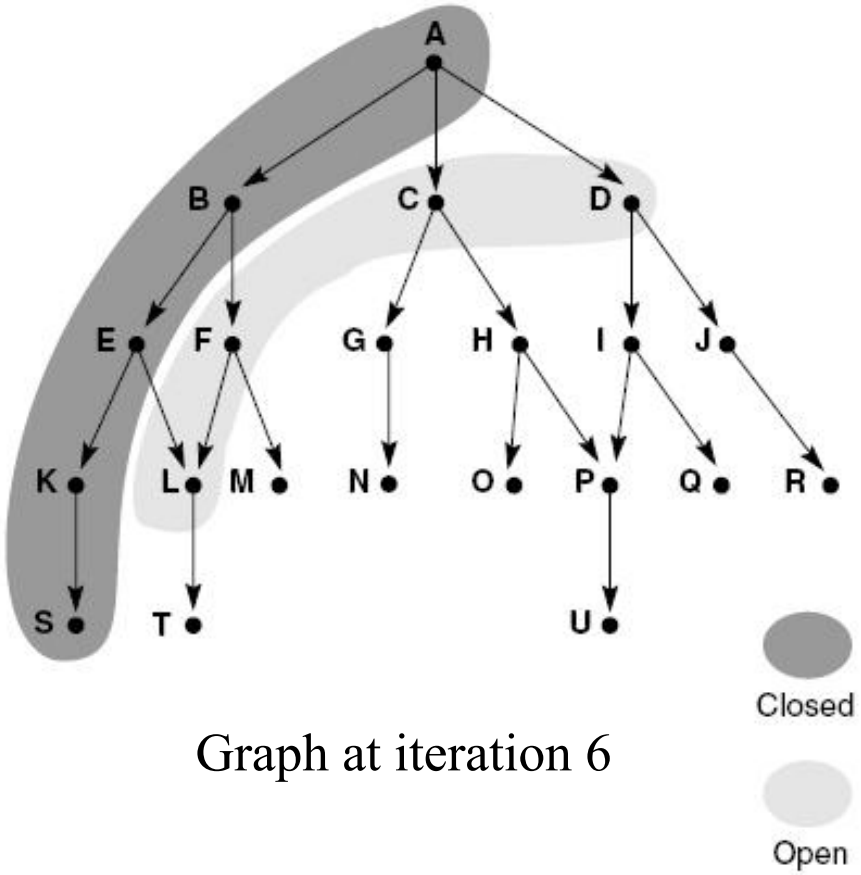
1. 状态空间的盲目搜索

深度优先搜索

深度优先搜索算法和广度优先搜索算法的步骤基本相同，它们之间的主要差别在于**Open**表中的节点排序不同。在深度优先搜索算法中，最后进入**Open**表的节点总是排在最前面，即后生成的节点先扩展。

深度优先搜索是一种非完备策略。对某些问题，它有可能找不到最优解，或者根本就找不到解。常用的解决方法是增加一个深度限制，当搜索达到规定深度但没找到解时向宽度搜索，称为有界深度优先搜索。

深度优先搜索例子



Graph at iteration 6

1. open = [A]; closed = [ ]
2. open = [B,C,D]; closed = [A]
3. open = [E,F,C,D]; closed = [B,A]
4. open = [K,L,F,C,D]; closed = [E,B,A]
5. open = [S,L,F,C,D]; closed = [K,E,B,A]
6. open = [L,F,C,D]; closed = [S,K,E,B,A]
7. open = [T,F,C,D]; closed = [L,S,K,E,B,A]
8. open = [F,C,D]; closed = [T,L,S,K,E,B,A]
9. open = [M,C,D], (as L is already on closed); closed = [F,T,L,S,K,E,B,A]
10. open = [C,D]; closed = [M,F,T,L,S,K,E,B,A]
11. open = [G,H,D]; closed = [C,M,F,T,L,S,K,E,B,A]

2. 代价树的盲目搜索

代价树的代价及广度优先搜索(1/2)

代价树的代价：用g(n)表示从初始节点S_0到节点n的代价，用$c(n, n_1)$表示从父节点n到其子节点n_1的代价。这样，对节点n_1的代价有：

$$g(n_1)=g(n)+c(n, n_1)。$$

代价树搜索的目的是为了找到最佳解，即找到一条代价最小的解路径。

代价树的广度优先搜索算法：

(1) 把初始节点S_0放入Open表中，置S_0的代价$g(S_0)=0$；

(2) 如果Open表为空，则问题无解 ，失败退出；

(3) 把Open表的第一个节点取出放入Closed表，并记该节点为n；

(4) 考察节点n是否为目标。若是，则找到了问题的解，成功退出；

(5) 若节点n不可扩展，则转第(2)步；

(6) 扩展节点n，生成其子节点$n_i(i=1, 2, \ldots)$，将这些子节点放入Open表中，并为每一个子节点设置指向父节点的指针。按如下公式：

$$g(n_i)=g(n)+c(n, n_i) \quad i=1,2,...$$

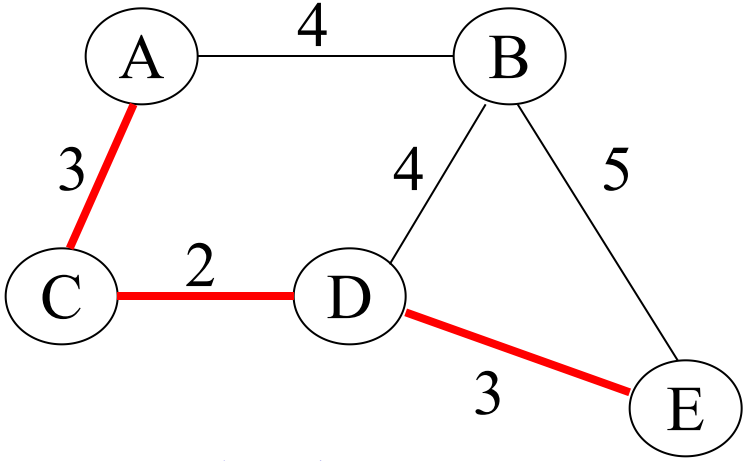
计算各子节点的代价，并根据各节点的代价对Open表中的全部节点按由小到大的顺序排序。然后转第(2)步。

2. 代价树的盲目搜索

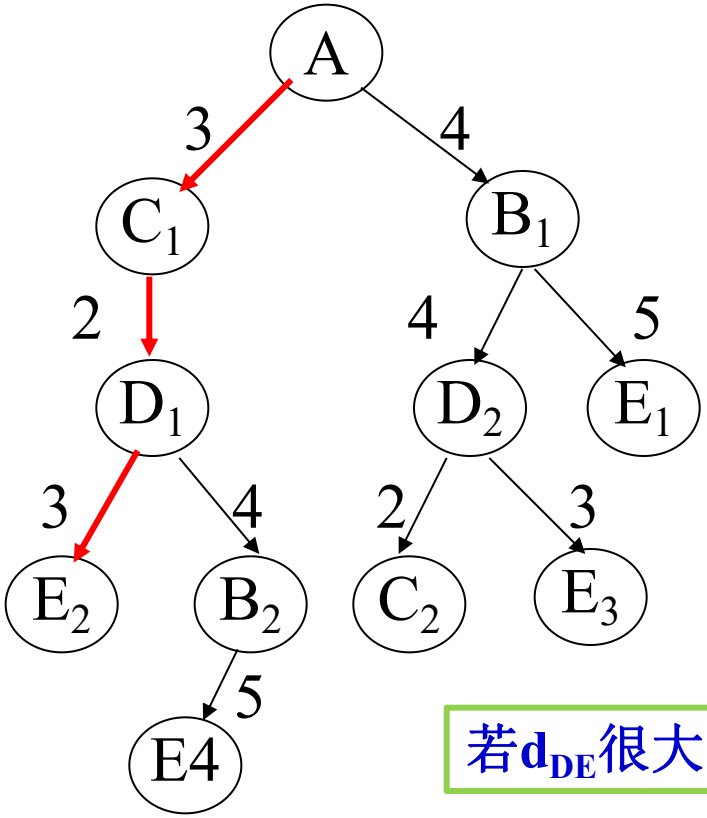
代价树的代价及广度优先搜索(2/2)

例 城市交通问题。设有5个城市，它们之间的交通线路如左图所示，图中的数字表示两个城市之间的交通费用，即代价。用代价树的广度优先搜索，求从A市出发到E市，费用最小的交通路线。

解：网络图转化为代价树，如右图所示。其中，红线为最优解，其代价为8



城市交通图



若d_{DE}很大呢？ 30

若问题有解，上述算法一定能找到，且找到的一定是最优解

2. 代价树的盲目搜索

代价树的深度优先搜索

代价树的深度优先搜索算法和代价树的广度优先搜索算法的步骤也基本相同，它们之间的主要差别在于**Open**表中的节点排序不同。在代价树的深度优先搜索算法中，每当扩展一个节点后，仅是把刚生成的子节点按照其边代价由小到大放入**Open**表的首部，而不需要对**Open**表中的全部节点再重新进行排序。

代价树的深度优先搜索是一种非完备策略。对某些问题，它有可能找不到最优解，或者根本找不到解。为此，也可增加一个深度限制，当搜索达到规定深度但仍没找到解时向宽度搜索，称为代价树的有界深度优先搜索。它们的具体算法描述和例子省略。

第4章 搜索策略

4.1 搜索概述

4.x 搜索的盲目策略

4.2 状态空间的启发式搜索

- 4.2.1启发性信息和估价函数
- 4.2.2 A算法
- 4.2.3 A*算法
- 4.2.4 A*算法应用举例

4.3 与/或树的启发式搜索

4.4 博弈树的启发式搜索

4.5 智能搜索算法（遗传算法、DE、SI）

4.2.1 启发性信息和估价函数

概念

启发性信息

启发性信息是指那种与具体问题求解过程有关的，并可指导搜索过程朝着最有希望方向前进的控制信息。

启发信息的启发能力越强，扩展的无用节点越少。包括以下3种：

① 有效地帮助确定扩展节点的信息；

② 有效地帮助决定哪些后继节点应被生成的信息；

③ 能决定在扩展一个节点时哪些节点应从搜索树上删除的信息。

估价函数

用来估计节点重要性，定义为从初始节点S_0出发，约束经过节点n到达目标节点S_g的所有路径中最小路径代价的估计值。一般形式：

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

其中，g(n)是从初始节点S_0到节点n的实际代价；h(n)是从节点n到目标节点S_g的最优路径的估计代价。（详见p126)

4.2.1 启发性信息和估价函数

例4.5 八数码难题。设问题的初始状态S_0和目标状态Sg如下图所示，且估价函数为

$$f(n)=d(n)+W(n)$$

其中：d(n)表示节点n在搜索树中的深度

W(n)表示节点n中“不在位”的数码个数。

请计算初始状态S_0的估价函数值$f(S_0)$

S_0

2	8	3
1		4
7	6	5

S_g

1	2	3
8		4
7	6	5

解：即g(n)=d(n)，h(n)=W(n)。 d(n)说明用从S_0到n的路径上的单位代价表示实际代价； W(n)说明用结点n中“不在位”的数码个数作为启发信息。可见，某节点中的“不在位”的数码个数越多，说明它离目标节点越远。

对初始节点S_0，由于$d(S_0)=0$，$W(S_0)=3$，因此有 $f(S_0)=0+3=3$

4.2.2 A算法

概念和算法描述

在状态空间搜索中，如果每一步都利用估价函数 **f(n)=g(n)+h(n)** 对**Open**表中的节点进行排序，则称**A**算法。它是一种启发式搜索算法。

类型：

全局择优： 从**Open**表的所有节点中选择一个估价函数值最小的进行扩展。

局部择优：仅从刚生成的子节点中选择一个估价函数值最小的进行扩展。

考虑全局择优搜索算法

若取**f(n)=g(n)**，退化为代价树的广度优先搜索

若取**f(n)=d(n)**，退化为广度优先搜索

4.2.2 A算法

概念和算法描述

全局择优搜索A算法描述：

(1)把初始节点S_0放入Open表中，$f(S_0)=g(S_0)+h(S_0)$；

(2)如果Open表为空，则问题无解，失败退出；

(3)把Open表的第一个节点取出放入Closed表，并记该节点为n；

(4)考察节点n是否为目标节点。若是，则找到了问题的解，成功退出；

(5)若节点n不可扩展，则转第(2)步；

(6)扩展节点n，生成其子节点$n_i(i=1, 2, \ldots)$，计算每一个子节点的估价值$f(n_i)(i=1, 2, \ldots)$，并为每一个子节点设置指向父节点的指针，然后将这些子节点放入Open表中；

(7)根据各节点的估价函数值，对Open表中的全部节点按从小到大的顺序重新进行排序；

(8)转第(2)步。

4.2.2 A算法

例4.6 八数码难题。设问题的初始状态S_0和目标状态S_g如图所示，估价函数与例4.5相同。请用全局择优搜索解决该问题。

解：该问题的全局择优搜索树如下图所示。在该图中，每个节点旁边的数字是该节点的估价函数值。

例如，对节点S_2，其估价函数值的计算为：$f(S_2)=d(S_2)+W(S_2)=2+2=4$

S_0

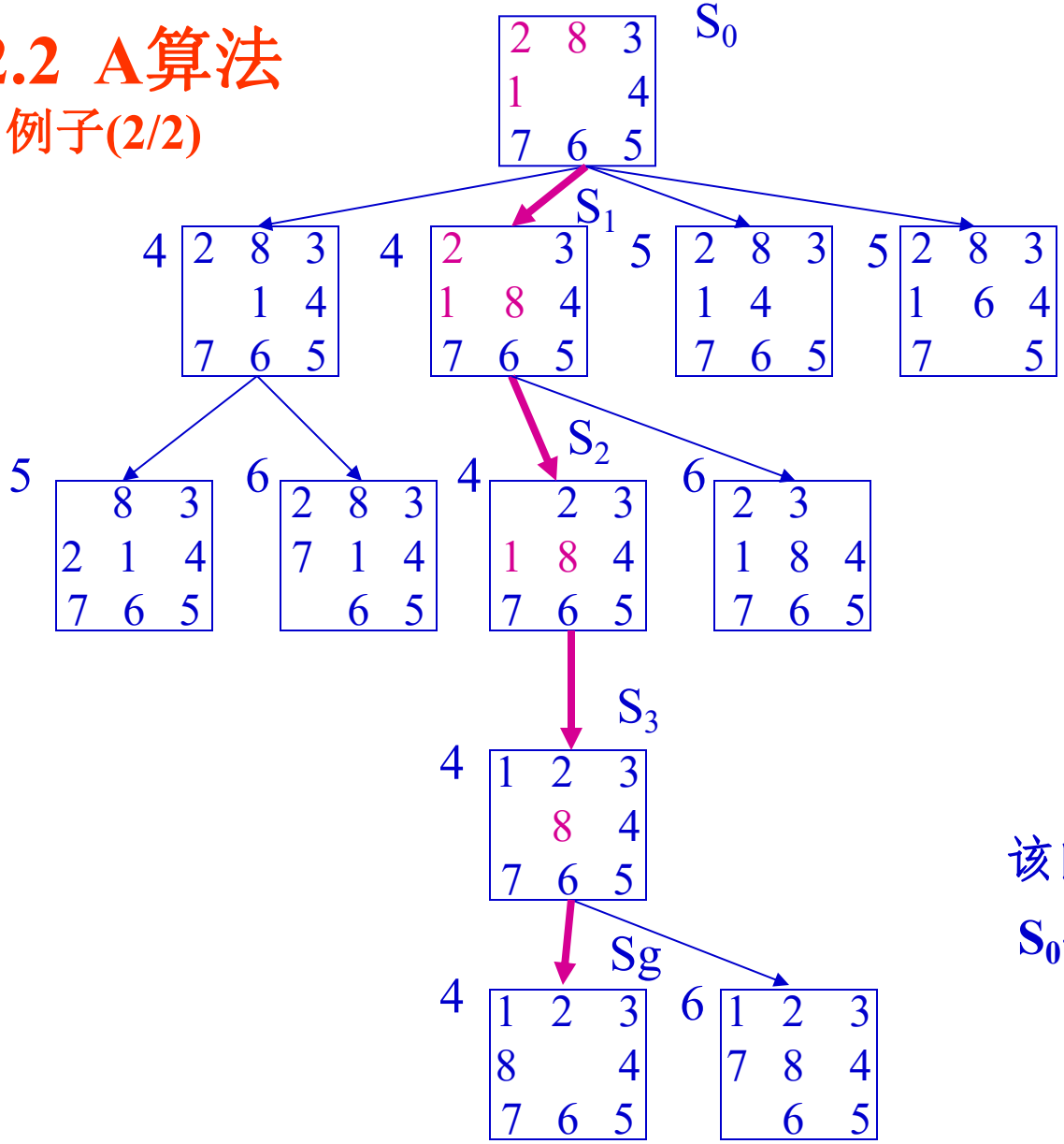
2	8	3
1		4
7	6	5

Sg

1	2	3
8		4
7	6	5

4.2.2 A算法

例子(2/2)



该问题的解为：

$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_g$

八数码难题的全局择优搜索树

4.2.3 A*算法

A*算法是对A算法的估价函数f(n)=g(n)+h(n)加上某些限制后得到的一种启发式搜索算法

假设f*(n)是从初始节点S_0出发，约束经过节点n到达目标节点S_g的最小代价，估价函数f(n)是对f*(n)的估计值。记

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

其中， g*(n)是从S_0出发到达n的最小代价,h*(n)是n 到S_g的最小代价

如果对A算法（全局择优）中的g(n)和h(n)分别提出如下限制：

第一，g(n)是对最小代价g*(n)的估计，且g(n)>0；

第二，h(n)是最小代价h*(n)的下界，即对任意节点n均有h(n)≤h*(n)。

则称满足上述两条限制的A算法为A*算法。

4.2.3 A*算法

1.A*算法的可采纳性

可采纳性的含义：

对任一状态空间图，当从初始节点到目标节点有路经存在时，如果搜索算法总能在有限步内找到一条从初始节点到目标节点的最佳路径，并在此路径上结束，则称该搜索算法是可采纳的(admissible)。

A*算法可采纳性的证明过程：

第一步，对有限图，A*算法一定能够成功结束。定理4.1

第二步，对无限图，A*算法也一定能够成功结束。引理4.1、4.2和推论4.1,定理4.2

第三步，A*算法一定能够结束在最佳路径上。定理4.3和推论4.2

4.2.3 A*算法

1.A*算法的可采纳性

定理4.1 对有限图，如果从初始节点S_0到目标节点S_g有路径存在，则算法A*一定成功结束。

证明：

首先证明算法必然会结束。

由于搜索图为有限图，如果算法能找到解，则成功结束；如果算法找不到解，则必然会由于Open表变空而结束。因此，A*算法必然会结束。

然后证明算法一定会成功结束。

由于至少存在一条由初始节点到目标节点的路径，设此路径为

$$S_0=n_0, n_1, \dots, n_k=S_g$$

算法开始时，节点n_0在Open表中，且路径中任一节点n_i离开Open表后，其后继节点n_{i+1}必然进入Open表，这样，在Open表变为空之前，目标节点必然出现在Open表中。因此，算法一定会成功结束。

4.2.3 A*算法

1.A*算法的可采纳性

引理4.1 对无限图，如果从初始节点S_0到目标节点S_g有路径存在，且A*算法不终止的话，则从Open表中选出的节点必将具有任意大的f值。

证明：设d*(n)是A*生成的从初始节点S_0到节点n的最短路经长度，由于搜索图中每条边的代价都是一个正数，令其中的最小的一个数是e，则有

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

因为g*(n)是最佳路径的代价，故有

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

又因为h(n)≥0，故有

$$f(n)=g(n)+h(n) \geq g(n) \geq d^*(n) \times e$$

如果A*算法不终止的话，从Open表中选出的节点必将具有任意大的d*值，因此，也将具有任意大的f值。

4.2.3 A*算法

1.A*算法的可采纳性

引理4.2 在A*算法终止前的任何时刻，Open表中总存在节点n’，它是从初始节点S_0到目标节点的最佳路径上的一个节点，且满足$f(n')\leq f^*(S_0)$。

证明：设从初始节点S_0到目标节点t的一条最佳路径序列为

$$S_0= n_0, n_1, \ldots , n_k=S_g$$

算法开始时，节点S_0在Open表中，当节点S_0离开Open表进入Closed表时，节点n_1进入Open表。以此类推，A*没有结束以前，在Open表中必存在最佳路径上的节点。设这些节点中排在最前面的节点为n'，则有

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

由于n'在最佳路径上，故有$g(n')=g^*(n')$，从而

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

又由于A*算法满足$h(n')\leq h^*(n')$，故有

$$f(n')\leq g^*(n')+h^*(n')=f^*(n')$$

因为在最佳路径上的所有节点的f*值都应相等，因此有

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

4.2.3 A*算法

1.A*算法的可采纳性

定理4.2 对无限图，若从初始节点S_0到目标节点S_g有路径存在，则A*算法必然会结束。

证明：（反证法）假设A*不结束，由引理4.1知Open表中的节点有任意大的f值，这与引理4.2的结论相矛盾，因此，A*算法只能成功结束。

推论4.1 Open表中任一具有$f(n)<f^*(S_0)$的节点n，最终都被A*算法选作为扩展的节点。

(证明略)

下面给出A*算法的可采纳性

4.2.3 A*算法

1. A*算法的可采纳性

定理4.3 A*算法是可采纳的，即若存在从初始节点S_0到目标节点S_g的路径，则A*算法必能结束在最佳路径上。

证明：证明过程分以下两步进行：

先证明A*算法一定能够终止在某个目标节点上。

由定理4.1和定理4.2可知，无论是对有限图还是无限图，A*算法都能够找到某个目标节点而结束。

再证明A*算法只能终止在最佳路径上。（反证法）

假设A*算法未能终止在最佳路径上，而是终止在某个目标节点t处，则有 $f(t)=g(t)>f^*(S_0)$

但由引理4.2可知，在A*算法结束前，必有最佳路径上的一个节点n'在Open表中，且有 $f(n')\leq f^*(S_0)<f(t)$

这时，A*算法一定会选择n'来扩展，而不可能选择t，从而也不会去测试目标节点t，这就与假设A*算法终止在目标节点t相矛盾。因此，A*算法只能终止在最佳路径上。

4.2.3 A*算法

1.A*算法的可采纳性

推论4.2 在A*算法中，对任何被扩展的节点n，都有$f(n)\leq f^*(S_0)$。

证明：令n是由A*选作扩展的任一节点，因此n不会是目标节点，且搜索没有结束。由引理4.2可知，在Open表中有满足

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

的节点n'。若n=n'，则有$f(n)\leq f^*(S_0)$；否则，选择n扩展，必有

$$f(n)\leq f(n')$$

所以有

$$f(n)\leq f^*(S_0)$$

4.2.3 A*算法

2.A*算法的最优性

A*算法的搜索效率很大程度上取决于启发函数h(n)。一般来说，在满足$h(n) \le h^*(n)$的前提下，h(n)的值越大越好。h(n)的值越大，说明它携带的启发性信息越多，A*算法搜索时扩展的节点就越少，搜索效率就越高。A*算法的这一特性被称为最优性。

下面通过一个定理来描述这一特性。

定理4.4 设有两个A*算法A_1*和A_2*，它们有

A_1*：$f_1(n)=g_1(n)+h_1(n)$

A_2*：$f_2(n)=g_2(n)+h_2(n)$

如果A_2*比A_1*有更多的启发性信息，即对所有非目标节点均有

$h_2(n)>h_1(n)$

则在搜索过程中，被A_2*扩展的节点也必然被A_1*扩展，即A_1*扩展的节点不会比A_2*扩展的节点少，亦即A_2*扩展的节点集是A_1*扩展的节点集的子集。

4.2.3 A*算法

2.A*算法的最优性

证明：（用数学归纳法）

(1) 对深度d(n)=0的节点，即n为初始节点S_0，如n为目标节点，则A_1*和A_2*都不扩展n；如果n不是目标节点，则A_1*和A_2*都要扩展n。

(2) 假设对A_2*中d(n)=k的任意节点n结论成立，即A_1*也扩展了这些节点。

(3) 证明A_2*中d(n)=k+1的任意节点n，也要由A_1*扩展。（用反证法）

假设A_2搜索树上有一个满足d(n)=k+1的节点n，A_2*扩展了该节点，但A_1*没有扩展它。根据第(2)条的假设，知道A_1*扩展了节点n的父节点。因此，n必定在A_1*的Open表中。既然节点n没有被A_1*扩展，则有

$$f_1(n) \geq f^*(S_0)$$

即 $g_1(n)+h_1(n) \geq f^*(S_0)$。但由于d=k时，$A_2$*扩展的节点$A_1$*也一定扩展，故有

$$g_1(n) \leq g_2(n)$$

因此有$h_1(n) \geq f^*(S_0)-g_1(n) \geq f^*(S_0)-g_2(n)$

另一方面，由于A_2*扩展了n，因此有

$$f_2(n) \leq f^*(S_0)$$

即 $g_2(n)+h_2(n) \leq f^*(S_0)$，亦即 $h_2(n) \leq f^*(S_0)-g_2(n)$，所以有 $h_1(n) \geq h_2(n)$

这与我们最初假设的$h_1(n)<h_2(n)$矛盾，因此反证法的假设不成立。

4.2.3 A*算法

3.h(n)的单调限制

如果能够保证，每当扩展一个节点时就已经找到了通往这个节点的最佳路径，将有效节省搜索代价。

在A*算法中，为满足这一要求，需对启发函数h(n)增加单调性限制。

定义4.1 如果启发函数满足以下两个条件：

(1) $h(S_g)=0$;

(2) 对任意节点n_i及其任一子节点n_j，都有

$$0 \le h(n_i)-h(n_j) \le c(n_i, n_j)$$

其中$c(n_i, n_j)$是n_i到其子节点n_j的边代价，则称h(n)满足单调限制。

$$h(n_i)-c(n_i, n_j) \le h(n_j)$$

+ A*算法的要求： h(n)是最小代价h*(n)的下界，即对任意节点n均有$h(n) \le h^*(n)$。

& $h^*(n_i)-c(n_i, n_j) = h^*(n_j)$

==> $h^*(n_i)-h(n_i) \ge h^*(n_j)- h(n_j)$

4.2.3 A*算法

3. h(n)的单调限制

定理4.5 如果h满足单调条件，则当A*算法扩展节点n时，该节点就已经找到了通往它的最佳路径，即g(n)=g*(n)。

证明：设A*正要扩展节点n，而节点序列 $S_0=n_0, n_1, \ldots, n_k=n$ 是由初始节点S_0到节点n的最佳路径。假设A*算法未找到通往n的最佳路径。其中，n_i是这个序列中最后一个位于Closed表中的节点，则上述节点序列中的n_{i+1}节点必定在Open表中。由单调条件

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

由于n_i和n_{i+1}都在最佳路径上，有 $g^*(n_{i+1})=g^*(n_i)+c(n_i, n_{i+1})$

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

一直推导下去可得 $$g^*(n_{i+1})+h(n_{i+1})\le g^*(n_k)+h(n_k)$$

由于节点在最佳路径上，故有 $f(n_{i+1})\le g^*(n)+h(n)$

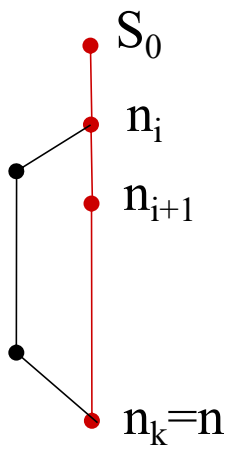
因为这时A*扩展节点n而不扩展节点n_{i+1}，则有

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

即 $g(n)\le g^*(n)$

但A*算法未找到通往n的最佳路径，应当有 $g(n)>g^*(n)$，矛盾。

所以A*算法找到了通往n的最佳路径，有 $g(n)=g^*(n)$



4.2.3 A*算法

3. h(n)的单调限制

定理4.6 如果h(n)满足单调限制，则A*算法扩展的节点序列的f 值是非递减的，即$f(n_i) \le f(n_{i+1})$。

证明：假设节点n_{i+1}在节点n_i之后立即扩展，由单调限制条件可知

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

即 $$f(n_i)-g(n_i)-f(n_{i+1})+g(n_{i+1}) \le c(n_i, n_{i+1})$$

亦即 $$f(n_i)-g(n_i)-f(n_{i+1})+g(n_i)+c(n_i, n_{i+1}) \le c(n_i, n_{i+1})$$

所以 $$f(n_i)-f(n_{i+1}) \le 0$$

即 $$f(n_i) \le f(n_{i+1})$$

以上两个定理都是在h(n)满足单调性限制的前提下才成立的。如果h(n)不满足单调性限制，则它们不一定成立。

在h(n)满足单调性限制下的A*算法常被称为改进的A*算法。

4.2.4 A*算法应用举例

例4.7 八数码难题。

f(n)=d(n)+P(n)

d(n) 深度

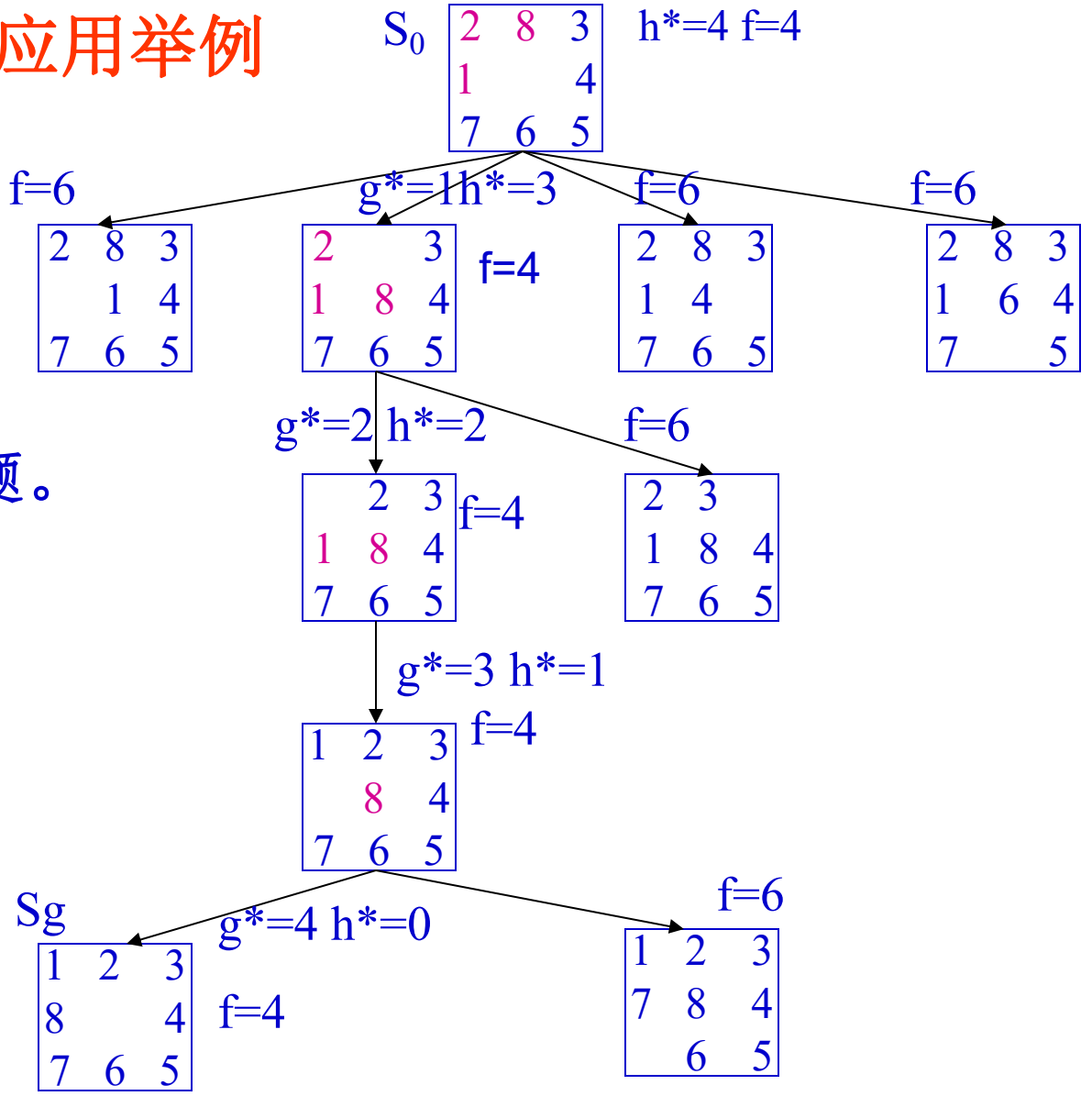
P(n)与目标距离

显然满足

P(n)≤ h*(n)

即满足A*条件

W(n)≤P(n)



八数码难题h(n)=P(n)的搜索树

4.2.4 A*算法应用举例

例4.8 修道士和野人问题

解：用m表示左岸的修道士人数，c表示左岸的野人数，b表示左岸的船数，用三元组(m, c, b)表示问题的状态。

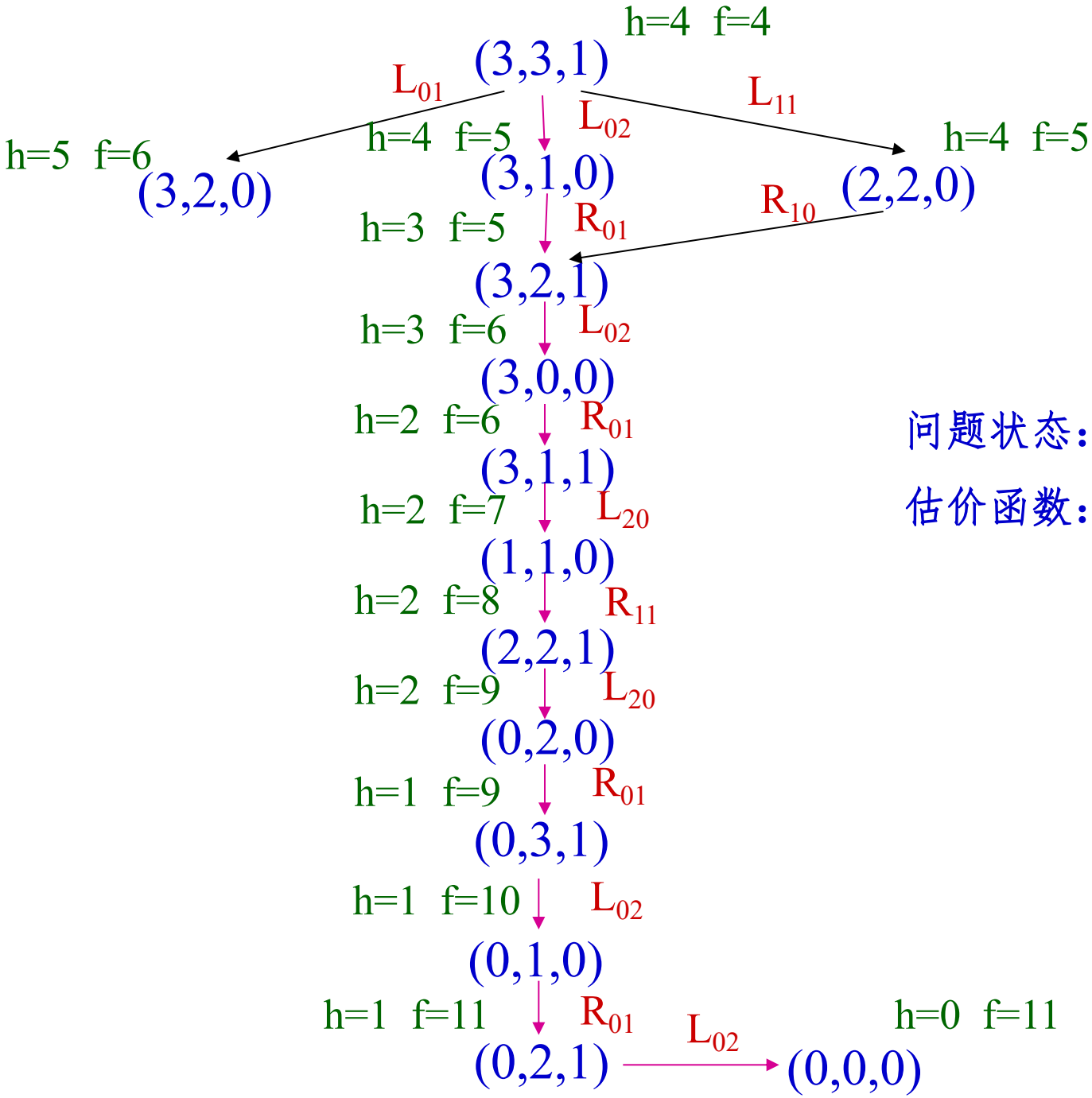
对A*算法，首先需要确定估价函数。设g(n)=d(n)，h(n)=m+c-2b，则有

$$f(n)=g(n)+h(n)=d(n)+m+c-2b$$

其中，d(n)为节点的深度。通过分析可知$h(n)\leq h^*(n)$，满足A*算法的限制条件。

M-C问题的搜索过程如下页图所示。

此例中估价函数值或可理解为 船跑的次数



问题状态：**(m,c,b)**

估价函数：**h(n)=m+c-2b**

Q1: 有哪些主要改进/新型搜索算法（非博弈、非仿生，如**hybrid A***、**D***等）及搜索算法的实际应用（如机器人路径规划、自动泊车等）

第4章 搜索策略

4.1 搜索概述

4.x 搜索的盲目策略

4.2 状态空间的启发式搜索

4.3 与/或树的启发式搜索

4.3.1解树的代价和希望树

4.3.2 与/或树的启发式搜索算法

4.4 博弈树的启发式搜索

4.5 智能搜索算法（遗传算法、DE、SI）

4.3.1 解树的代价与希望树

1. 解树的代价（1/ 2）

解树的代价可按如下方法计算：

(1)若n为终止节点，则其代价h(n)=0；

(2)若n为或节点，且子节点为$n_1, n_2, \ldots, n_k$，则n的代价为：

$$h(n) = \min_{1 \le i \le k} \{c(n, n_i) + h(n_i)\}$$

其中，$c(n, n_i)$是节点n到其子节点n_i的边代价。

(3)若n为与节点，且子节点为$n_1, n_2, \ldots, n_k$，则n的代价可用和代价法或最大代价法。

若用和代价法，则其计算公式为：$h(n) = \sum_{i=1}^{k} \{c(n, n_i) + h(n_i)\}$

若用最大代价法，则其计算公式为：$h(n) = \max_{1 \le i \le k} \{c(n, n_i) + h(n_i)\}$

(4)若n是端节点，但又不是终止节点，且n不可扩展，其代价定义为h(n)=∞。

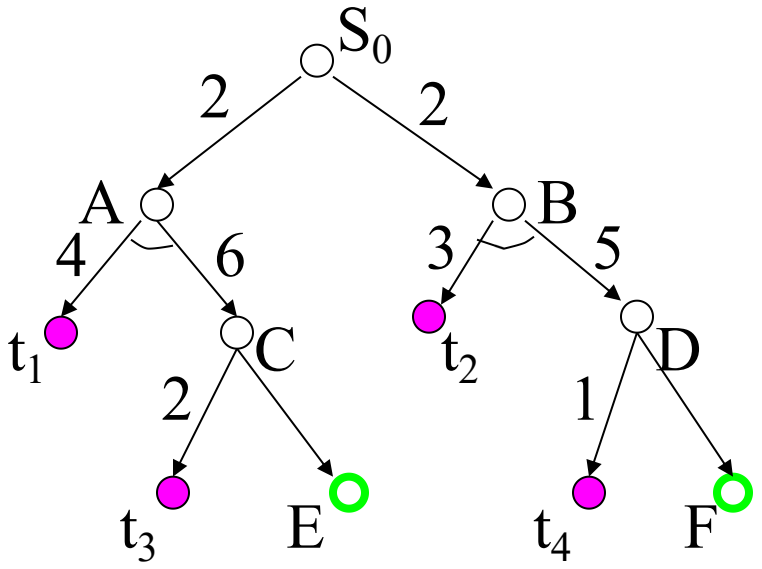
(5)根节点的代价即为解树的代价。

4.3.1 解树的代价与希望树

1. 解树的代价（2/2）

例4.9 设下图是一棵与/或树，在该树中，t_1、t_2、t_3、t_4为终止节点；E、F是端节点；边上的数字是该边的代价。请计算解树的代价。

解：该树包括两棵解树，左边的解树由S_0、A、t_1、C及t_2组成；右边的解树由S_0、B、t_2、D及t_4组成。



与/或树的代价

先计算左边的解树

按和代价：$h(S_0)=2+6+4+2=14$

按最大代价：$h(S_0)=(2+6)+2=10$

再计算右边的解树

按和代价：$h(S_0)=1+5+3+2=11$

按最大代价：$h(S_0)=(1+5)+2=8$

4.3.1 解树的代价与希望树

2. 希望树

希望树是指搜索过程中最有可能成为最优解树的那棵树。

与/或树的启发式搜索过程就是不断地选择、修正希望树的过程，在该过程中，希望树是不断变化的。

定义4.2 希望解树

(1) 初始节点S_0在希望树T

(2) 如果n是具有子节点$n_1, n_2, \ldots, n_k$的或节点，则n的某个子节点n_i在希望树T中的充分必要条件是

$$\min_{1\le i\le k}\{c(n,n_i)+h(n_i)\}$$

(3) 如果n是与节点，则n的全部子节点都在希望树T中。

4.3.2 与/或树的启发式搜索过程

与/或树的启发式搜索过程如下：

(1) 把初始节点S_0放入Open表中，计算$h(S_0)$；

(2) 计算希望树T；

(3) 依次在Open表中取出T的端节点放入Closed表，并记该节点为n；

(4)如果节点n为终止节点，则做下列工作：

① 标记节点n为可解节点；

② 在T上应用可解标记过程，对n的先辈节点中的所有可解解节点进行标记；

③ 如果初始解节点S_0能够被标记为可解节点，则T就是最优解树，成功退出；

④ 否则，从Open表中删去具有可解先辈的所有节点。

⑤ 转第(2)步。

4.3.2 与/或树的启发式搜索过程

(5) 如果节点n不是终止节点，但可扩展，则做下列工作：

① 扩展节点n，生成n的所有子节点；

② 把这些子节点都放入Open表中，并为每一个子节点设置指向父节点n的指针

③ 计算这些子节点及其先辈节点的h值；

④ 转第(2)步。

(6) 如果节点n不是终止节点，且不可扩展，则做下列工作：

① 标记节点n为不可解节点；

② 在T上应用不可解标记过程，对n的先辈节点中的所有不可解解节点进行标记；

③ 如果初始节点S_0能够被标记为不可解，则问题无解，失败退出；

④ 否则，从Open表中删去具有不可解先辈的所有节点。

⑤ 转第(2)步。

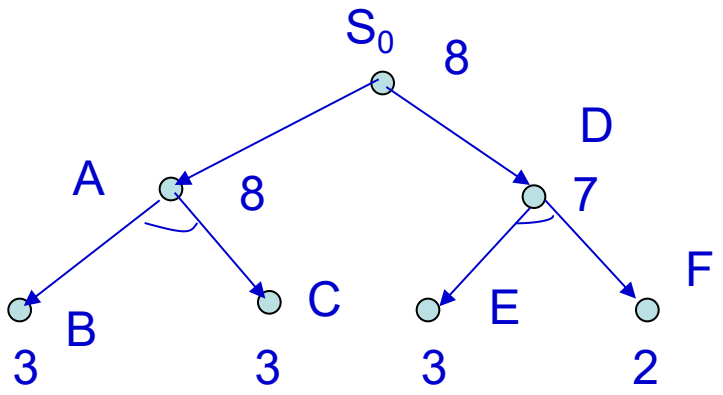
4.3.2 与/或树的启发式搜索过程

例子

要求搜索过程每次扩展节点时都同时扩展两层，且按一层或节点、一层与节点的间隔方式进行扩展。它实际上就是下一节将要讨论的博弈树的结构。

设初始节点为S_0，对S_0扩展后得到的与/或树如右图所示。其中，端节点B、C、E、F，下面的数字是用启发函数估算出的h值，节点S_0、A、D旁边的数字是按和代价法计算出来的节点代价。

此时，S_0的右子树是当前的希望树。



扩展S_0后得到的与/或树

按和代价法：

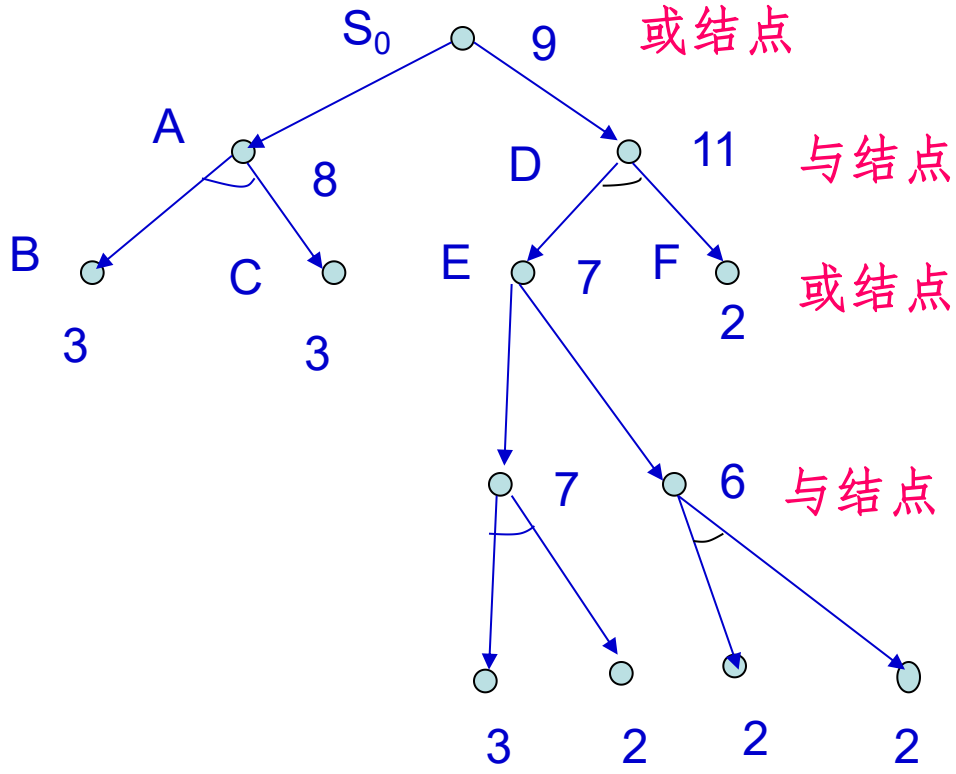
例，节点A的值=3+1+3+1=8

加条件 $c(n,n_i)=1$

4.3.2 与/或树的启发式搜索过程

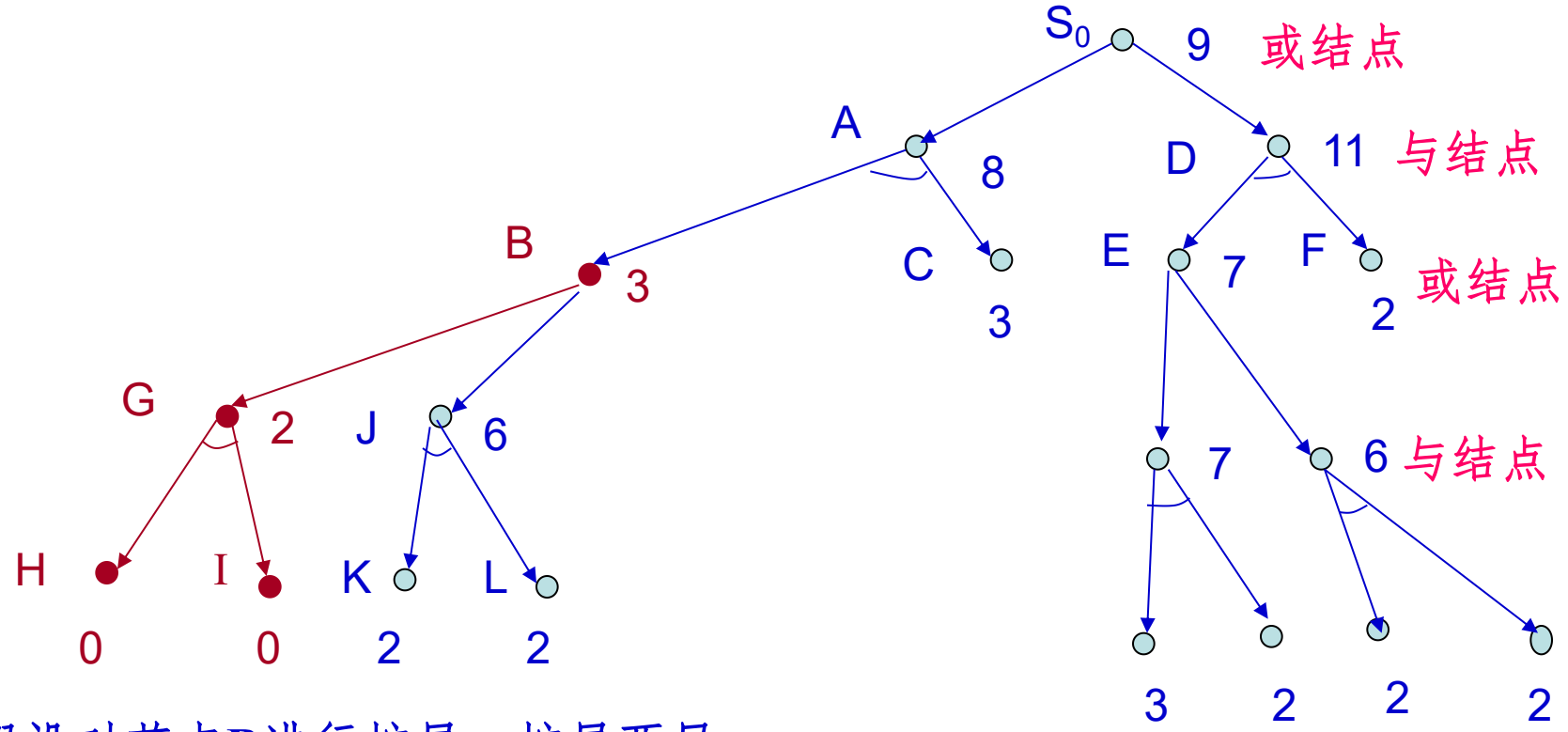
假设扩展节点E，得到如右图所示的与/或树。

此时，由右子树求出的$h(S_0)=12$。但是，由左子树求出的$h(S_0)=9$。显然，左子树的代价小。因此，当前的希望树应改为左子树。



扩展节点E后得到的与/或树

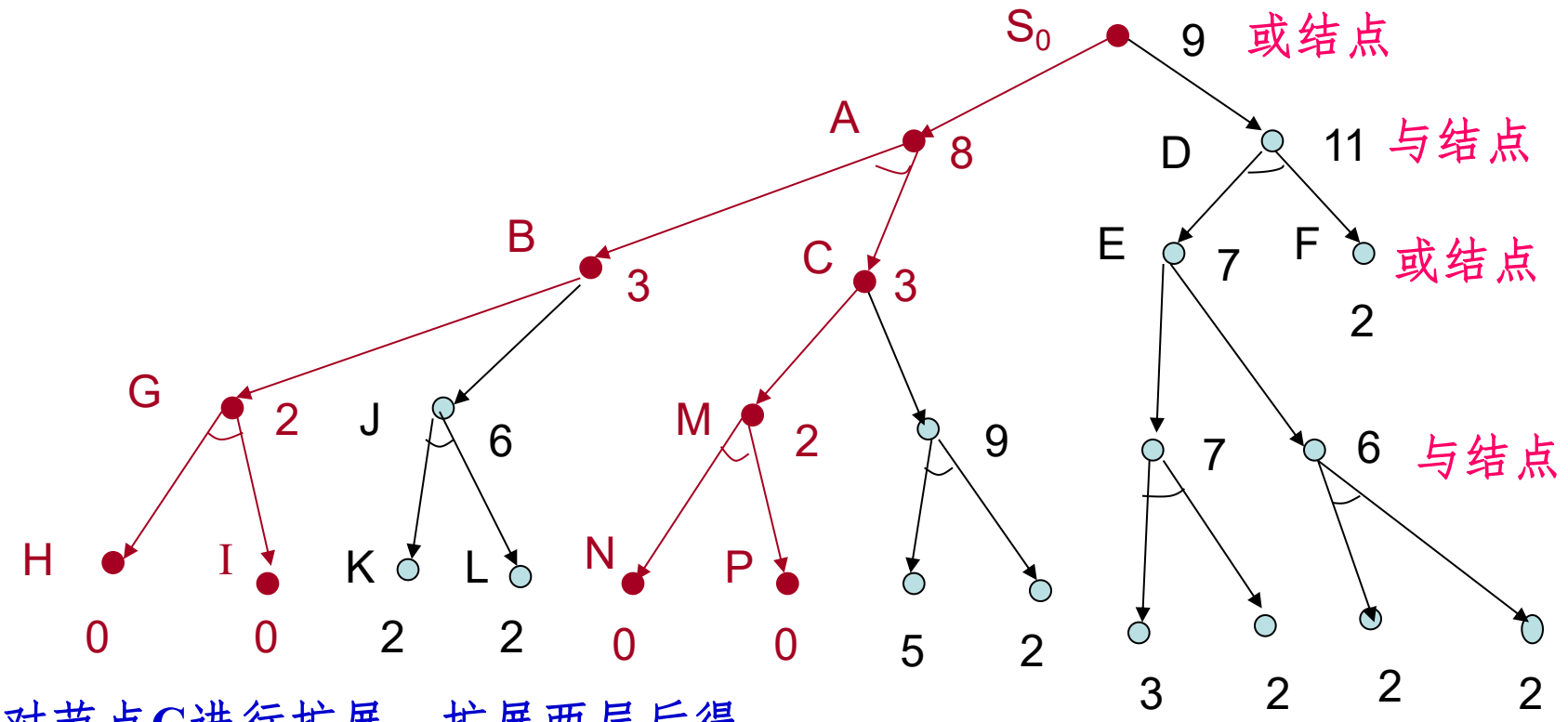
4.3.2 与/或树的启发式搜索过程



扩展节点B后得到的与/或树

假设对节点B进行扩展，扩展两层后得到的与/或树如下图所示。由于节点H和I是可解节点，故调用可解标记过程，得节点G、B也为可解节点，但不能标记S_0为可解节点，须继续扩展。当前的希望树仍然是左子树。

4.3.2 与/或树的启发式搜索过程



扩展节点C后得到的与/或树

对节点C进行扩展，扩展两层后得到的与/或树如右图所示。由于节点N和P是可解节点，故调用可解标记过程，得节点M、C、A也为可解节点，进而可标记S_0为可解节点，这就的到了代价最小的解树。

按和代价法，该最优解的代价为9。

第4章 搜索策略

4.1 搜索概述

4.x 搜索的盲目策略

4.2 状态空间的启发式搜索

4.3 与/或树的启发式搜索

4.4 博弈树的启发式搜索

- 4.4.1 概述
- 4.4.2 极/大极小过程
- 4.4.3 α-β剪枝

4.5 智能搜索算法（遗传算法、DE、SI）

4.4.1 概述

博弈的概念

博弈是一类富有智能行为的竞争活动，如下棋、打牌等。

博弈的类型

双人完备信息博弈：两位选手（例如MAX和MIN）对垒，轮流走步，每一方不仅知道对方已经走过的棋步，而且还能估计出对方未来的走步。

机遇性博弈：存在不可预测性的博弈，例如掷币等。

“与”与“或”，MAX and MIN （p136)

博弈树

若把双人完备信息博弈过程用图表示出来，就得到一棵与/或树，这种与/或树被称为博弈树。在博弈树中，那些下一步该MAX走步的节点称为MAX节点，下一步该MIN走步的节点称为MIN 节点。

4.4.1 概述

博弈树的特点

(1) 博弈的初始状态是初始节点；

(2) 博弈树中的“或”节点和“与”节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，例如MAX方。所有能使自己一方获胜的终局都是本原问题，相应的节点是可解节点；所有使对方获胜的终局都是不可解节点。

对简单的博弈问题，可生成整个博弈树，找到必胜的策略。

对于复杂的博弈问题，不可能生成整个搜索树，如国际象棋，大约有10^{120}个节点。一种可行的方法是用当前正在考察的节点生成一棵部分博弈树，并利用估价函数f(n)对叶节点进行估值。

4.4.2 极大极小过程

对叶节点的估值方法是：那些对MAX有利的节点，其估价函数取正值；那些对MIN有利的节点，其估价函数取负值；那些使双方均等的节点，其估价函数取接近于0的值。

为非叶节点的值，必须从叶节点开始向上倒退。其倒退方法是：

对于MAX节点，由于MAX 方总是选择估值最大的走步，因此，MAX节点的倒退值应该取其后继节点估值的最大值。

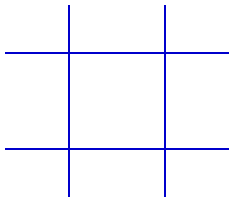
对于MIN节点，由于MIN方总是选择使估值最小的走步，因此MIN节点的倒退值应取其后继节点估值的最小值。

这样一步一步地计算倒退值，直至求出初始节点的倒推值为止。这一过程称为极大极小过程。

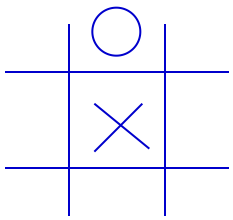
4.4.2 极大极小过程

例4.10一字棋游戏

设有一个三行三列的棋盘，如下图所示，两个棋手轮流走步，每个棋手走步时往空格上摆一个自己的棋子，谁先使自己的棋子成三子一线为赢。设MAX方的棋子用×标记，MIN方的棋子用〇标记，并规定MAX方先走步。



一字棋棋盘



棋局1

$e(P)=e(+P)-e(-P)$

$=6-4=2$

解：估价函数e(+P)：棋盘P上有可能使×成三子为一线的数目；

e(-P)： P上有可能使〇成三子为一线的数目；

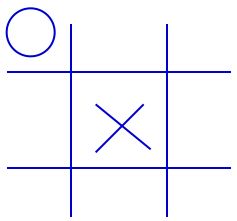
当MAX 必胜 e(P)为正无穷大；

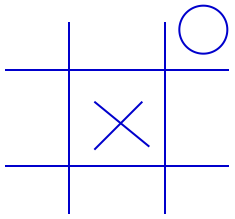
MIN 必胜 e(P)为负无穷大。

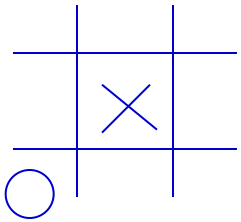
胜负未定则e(P)=e(+P)-e(-P)

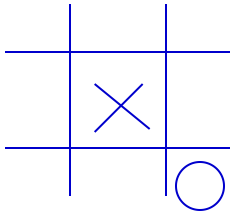
4.4.2 极大极小过程

具有对称性的棋盘可认为是同一棋盘。如下图所示：







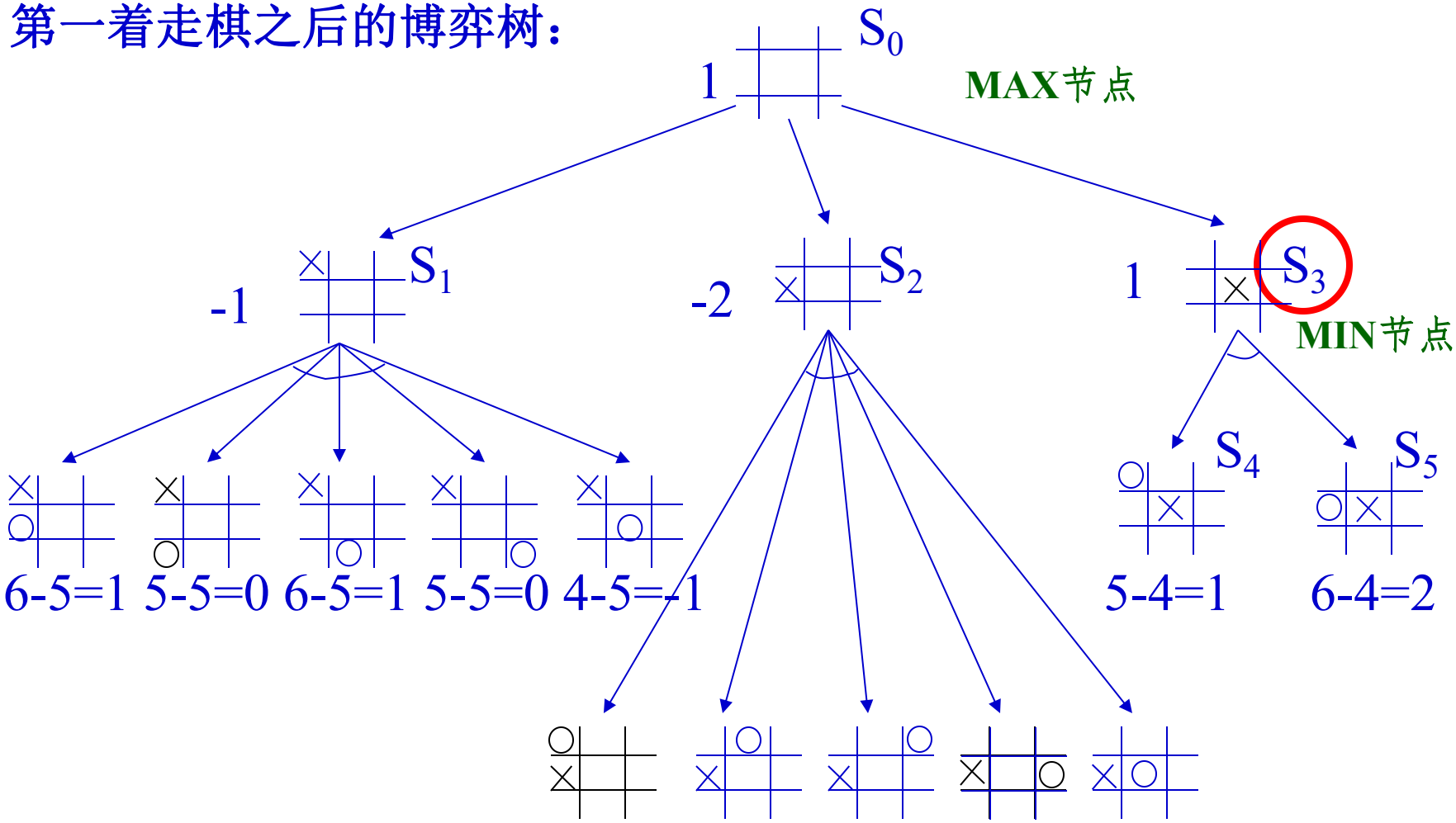


×为MAX方的棋子

○为MIN方的棋子

其e(P)=e(+P)-e(-P)=5-4=1

第一着走棋之后的博弈树：



一字棋的极大极小搜索

4.4.3 α-β剪枝

剪枝的概念

极大极小过程是先生成与/或树，然后再计算各节点的估值，这种生成节点和计算估值相分离的方式，需生成和考查规定深度内的所有节点，搜索效率较低。

如果能边生成节点边对节点估值，并剪去一些没用的分枝，这种技术被称为α-β剪枝。

剪枝方法

(1) MAX节点（或节点）的α值为当前子节点的最大倒推值；

(2) MIN节点（与节点）的β值为当前子节点的最小倒推值；

(3) α-β剪枝的规则如下：

β剪枝

任何MAX节点n的α值大于或等于它先辈节点的β值，则n 以下的分枝可停止搜索，并令节点n的倒推值为α。这种剪枝称为β剪枝。

α剪枝

任何MIN节点n的β值小于或等于它先辈节点的α值，则n 以下的分枝可停止搜索，并令节点n的倒推值为β。这种剪枝称为α剪枝。

4.4.3 α-β剪枝

4.4.3 The Alpha-Beta Procedure

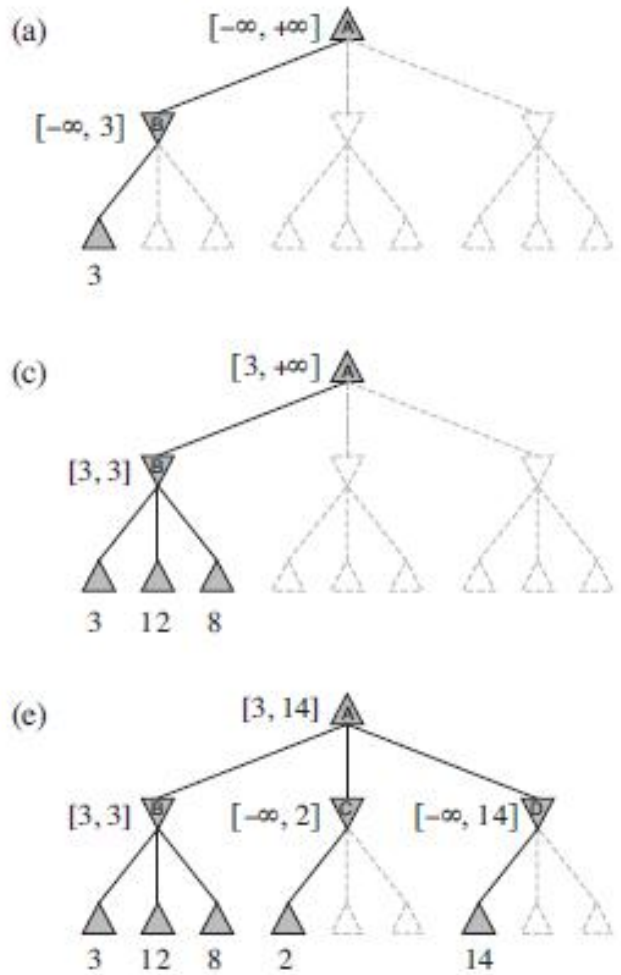
Straight minimax requires a two-pass analysis of the search space, the first to descend to the ply depth and there apply the heuristic and the second to propagate values back up the tree. Minimax pursues all branches in the space, including many that could be ignored or pruned by a more intelligent algorithm. Researchers in game playing developed a class of search techniques called *alpha-beta* pruning, first proposed in the late 1950s (Newell and Simon 1976), to improve search efficiency in two-person games (Pearl 1984).

The idea for alpha-beta search is simple: rather than searching the entire space to the ply depth, alpha-beta search proceeds in a depth-first fashion. Two values, called *alpha* and *beta*, are created during the search. The alpha value, associated with MAX nodes, can never decrease, and the beta value, associated with MIN nodes, can never increase. Suppose a MAX node's alpha value is 6. Then MAX need not consider any backed-up value

From: G.F. Luger, *Artificial Intelligence- structure and strategies for complex problem solving,* 6th Ed.

4.4.3 α-β剪枝

alpha-beta搜索不断更新alpha和beta的




```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

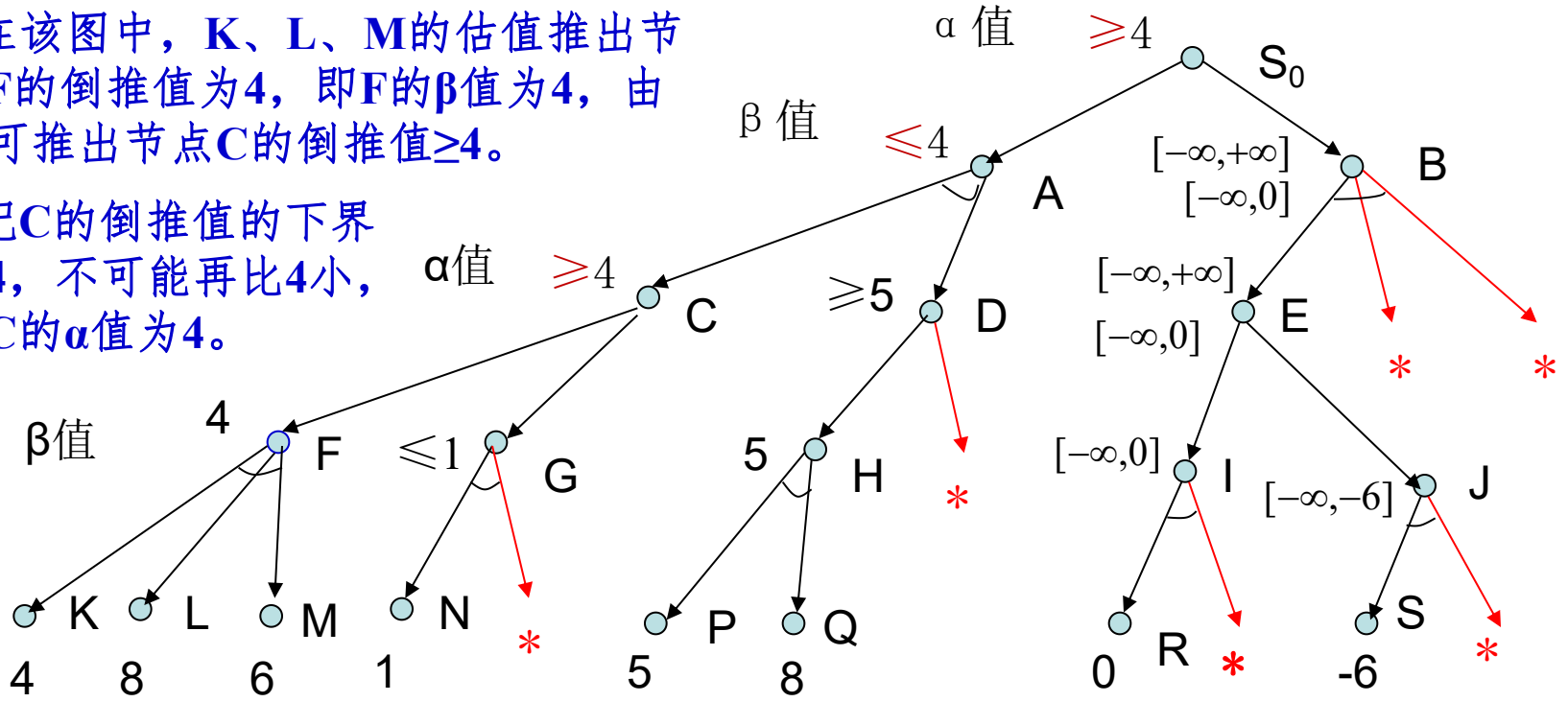
Figure 5.7 The alpha–beta search algorithm. Notice

From: *Artificial Intelligence: A Modern Approach*（3rd edition）

一个α-β剪枝的具体例子，如下图所示。其中最下面一层端节点旁边的数字是假设的估值。

在该图中，K、L、M的估值推出节点F的倒推值为4，即F的β值为4，由此可推出节点C的倒推值≥4。

记C的倒推值的下界为4，不可能再比4小，故C的α值为4。



由节点N的估值推知节点G的倒推值小于≤1,无论G的其它子节点的估值是多少，G的倒推值都不可能比1大。因此，1是G的倒推值的上界，所以G的值≤1 。另已知C的倒推值为4，G的其它子节点又不可能使C的倒推值增大。因此对G的其它分支不必再搜索，相当于把这些分枝剪去。由F、G的倒推值可推出节点C的倒推值为4 ，再由C可推出节点A的倒推值≤4，即A的β值为4。另外，由节点P、Q推出的节点I的倒推值为5，因此D的倒推值 ≥5，即D的α值为5。此时，D的其它子节点的倒推值无论是多少都不能使D及A的倒推值减少或增大，所以D的其他分枝被减去，并可确定A的倒推值为4 。以此类推，最终推出S_0的倒推值为4。

Q2: 进一步的对抗搜索算法及应用

作业 (题号可能不同)

4.5 用A*算法做

4.5　有一农夫带一条狼、一只羊和一筐菜，欲从河的左岸乘船到右岸，但受下列条件限制：

（1）船太小，农夫每次只能带一样东西过河；

（2）如果没有农夫看管，则狼要吃羊，羊要吃菜。

请设计一个过河方案，使得农夫、狼、羊都能不受损失地过河，画出相应的状态空间图。

提示：（1）用四元组(农夫, 狼, 羊, 菜)表示状态，其中每个元素都为 0 或 1，用 0 表示在左岸，用 1 表示在右岸。

（2）把每次过河的一种安排作为一种操作，每次过河都必须有农夫，因为只有他可以划船。

4.10　设有如图 4.27 所示的与/或树，请分别用和代价法、最大代价法求解树的代价。

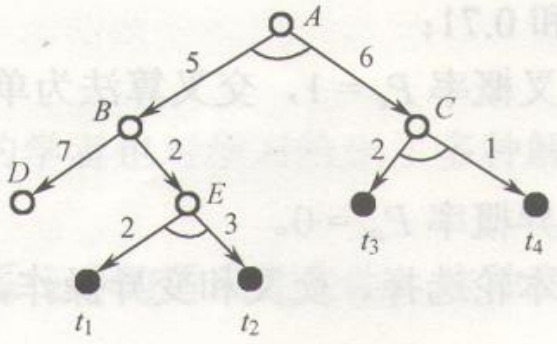


图 4.27　习题 4.10 的与/或树

4.11　设有如图 4.28 所示的博弈树，其中最下面的数字是假设的估值，请对该博弈树做如下工作：

（1）计算各节点的倒退值；

（2）利用 α-β 剪枝技术剪去不必要的分支。

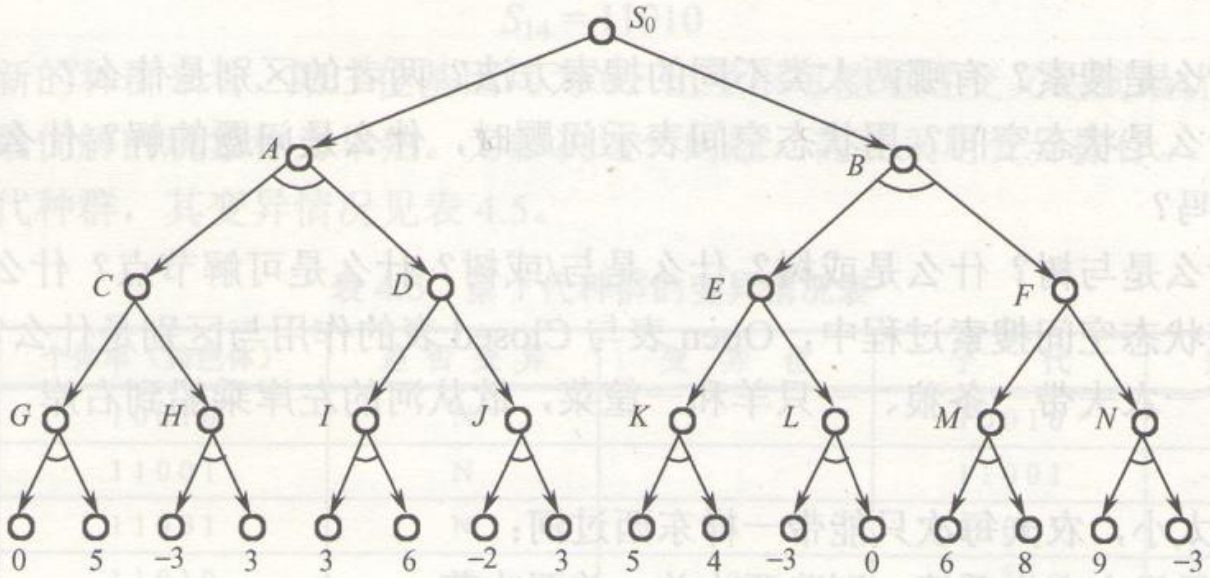


图 4.28　习题 4.11 的博弈树

4.20　设种群规模为 4，采用二进制编码，适应度函数 $f(x)=x^2$，初始种群如表 4.8 所示。

表 4.8　初始种群

编　号	个 体 串	x	适 应 值	百 分 比	累计百分比	选中次数
S_{01}	1010	10				
S_{02}	0100	4				
S_{03}	1100	12				
S_{04}	0111	7				

若遗传操作规定如下：

（1）选择概率 $P_r=1$，选择操作用轮盘赌算法，且依次生成的 4 个随机数分别为 0.42，0.16，0.89 和 0.71；

（2）交叉概率 $P_c=1$，交叉算法为单点交叉，交叉点为 3，交叉顺序按个体在种群中的顺序；

（3）变异概率 $P_m=0$。

请完成本轮选择、交叉和变异操作，并给出所得到的下一代种群。