第 5 章　搜索求解策略

教材：

王万良《人工智能及其应用》（第3版）

高等教育出版社，2017.5

第5章 搜索求解策略

- 在求解一个问题时，涉及到两个方面：一是该问题的表示，如果一个问题找不到一个合适的表示方法，就谈不上对它求解。另一方面则是选择一种相对合适的求解方法。由于绝大多数需要人工智能方法求解的问题缺乏直接求解的方法，因此，搜索为一种求解问题的一般方法。
- 下面首先讨论搜索的基本概念，然后着重介绍搜索问题的表示和搜索策略，主要有回溯策略、宽度优先搜索、深度优先搜索等盲目的搜索策略，以及A*算法、极大较小值等启发式搜索策略。

第5章 搜索求解策略

- 5.1 搜索的概念
- 5.2 状态空间表示法
- 5.3 状态空间盲目搜索
- 5.4 状态空间启发式搜索
- 5.5 与或树表示
- 5.6 与或树盲目搜索
- 5.7 与或树启发式搜索
- 5.8 博弈树搜索

第5章 搜索求解策略

- ✓ **5.1 搜索的概念**
- **5.2 状态空间表示法**
- **5.3 状态空间盲目搜索**
- **5.4 状态空间启发式搜索**
- **5.5 与或树表示**
- **5.6 与或树盲目搜索**
- **5.7 与或树启发式搜索**
- **5.8 博弈树搜索**

5.1 搜索的概念

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，从而构造一条代价最小的推理路线，使问题得以解决的过程称为**搜索**

5.1.1 搜索的基本问题与主要过程

- **搜索中需要解决的基本问题：**

（1）是否一定能找到一个解。

（2）找到的解是否是最佳解。

（3）时间与空间复杂性如何。

（4）是否终止运行或是否会陷入一个死循环。

5.1.2 搜索的类型

- 按问题的表示方式

  - 状态空间搜索(State Space)

    寻找从初始状态到目标状态的解。

  - 与或树搜索(And/Or Tree)

    寻找与或树代表的问题是否可解或不可解。

5.1.2 搜索的类型

- **按是否使用启发式信息**

  - **盲目搜索（Blind Search）：**在不具有对特定问题的任何有关信息的条件下，按固定的步骤（依次或随机调用操作算子）进行的搜索。
  - **启发式搜索(Heuristic Search)：**考虑特定问题领域可应用的知识，动态地确定调用操作算子的步骤，优先选择较适合的操作算子，尽量减少不必要的搜索，以求尽快地到达结束状态。

第5章 搜索求解策略

- 5.1 搜索的概念
- ✓ 5.2 状态空间表示法
- 5.3 状态空间盲目搜索
- 5.4 状态空间启发式搜索
- 5.5 与或树表示
- 5.6 与或树盲目搜索
- 5.7 与或树启发式搜索
- 5.8 博弈树搜索

5.2 状态空间表示法

状态空间表示法用“状态”和“算符”来表示问题

- 状态—描述问题求解过程不同时刻的状态
- 算符—表示对状态的操作

算符的每一次使用就使状态发生变化。当到达目标状态时，由**初始状态到目标状态所使用的算符序列就是问题的一个解**。

5.2 状态空间表示法

- ①**状态**：表示系统状态、事实等叙述型知识的一组变量或数组。

$$Q=[q_1,q_2,\Lambda\ ,q_n]$$

- 当给每一个分量以确定的值时，就得到了一个具体的状态

- ②**算符**：表示引起状态变化的过程型知识的一组关系或函数。使问题由一个状态变为另一个状态，也称为操作。在产生式系统中，每一条产生式规则就是一个算符。

$$F=\{f_1,f_2,\cdots,f_m\}$$

5.2 状态空间表示法

③**状态空间：**利用状态变量和操作符号，表示系统或问题的有关知识的符号体系，状态空间是一个四元组：

$$(S, O, S_0, G)$$

S：状态集合。

O：操作算子的集合。

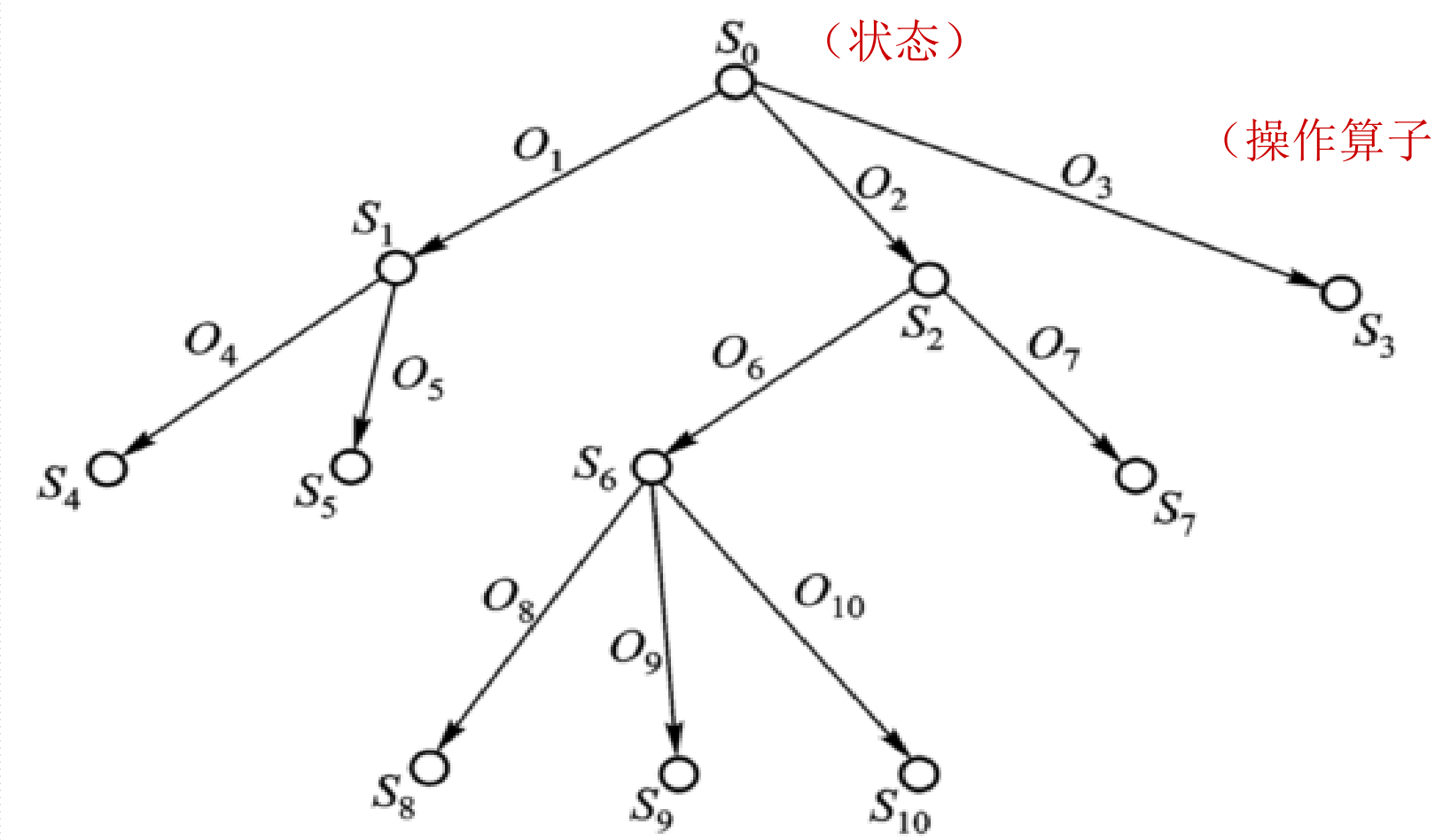
S_0：包含问题的初始状态是 S 的非空子集。

G：若干具体状态或满足某些性质的路径信息描述。

④**状态空间图**

状态空间的图示形式称为状态空间图。其中节点表示状态；有向边表示算符。

5.2 状态空间表示法



状态空间的有向图描述

5.2 状态空间表示法

- **求解路径：**从S_0结点到G结点的路径。

- **状态空间的一个解：**一个有限的操作算子序列。

$$S_0 \xrightarrow{O_1} S_1 \xrightarrow{O_2} S_2 \xrightarrow{O_3} \Lambda \xrightarrow{O_k} G$$

$O_1,\ldots,O_k$ ：状态空间的一个解。

5.2 状态空间表示法

- **例1 二阶“梵塔”问题状态空间表示**
- 有三个柱子(1，2，3)和两个不同尺寸的圆盘(A，B)。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,全部两个圆盘都堆在柱子1上(最大的在底部,最小的在顶部)。要求把所有圆盘都移到另一个柱子上，搬动规则为:

(1)一次只能搬一个圆盘

(2)不能将大圆盘放在小圆盘

(3)可以利用空柱子。

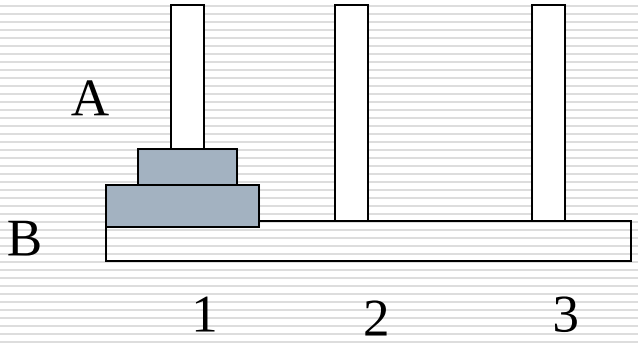
5.2 状态空间表示法

二阶“梵塔”问题状态空间表示

- 如何用状态空间方法来描述问题？
- 状态的表示
  - 柱子的编号用i,j来代表
    (i,j)表示问题的状态其中: i代表A所在的柱子
    j 代表B所在的柱子
  - 状态集合 (9种可能的状态)
    s0=(1,1), s1=(1,2), s2=(1,3)
    s3=(2,1), **s4=(2,2),** s5=(2,3)
    s6=(3,1), s7=(3,2), **s8=(3,3)**

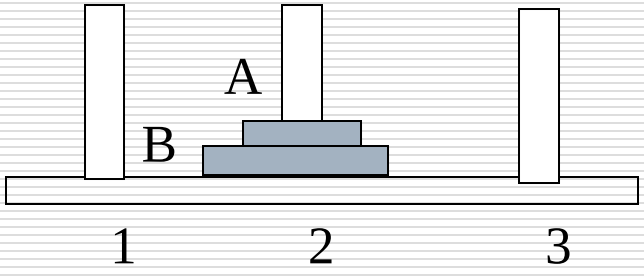
5.2 状态空间表示法

- 初 始 状 态
  S={s0}

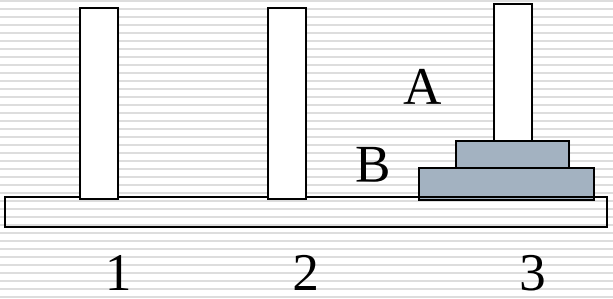


S_0=(1,1)

- 目标状态G={s4,s8}



S_4=(2,2)



S_8=(3,3)

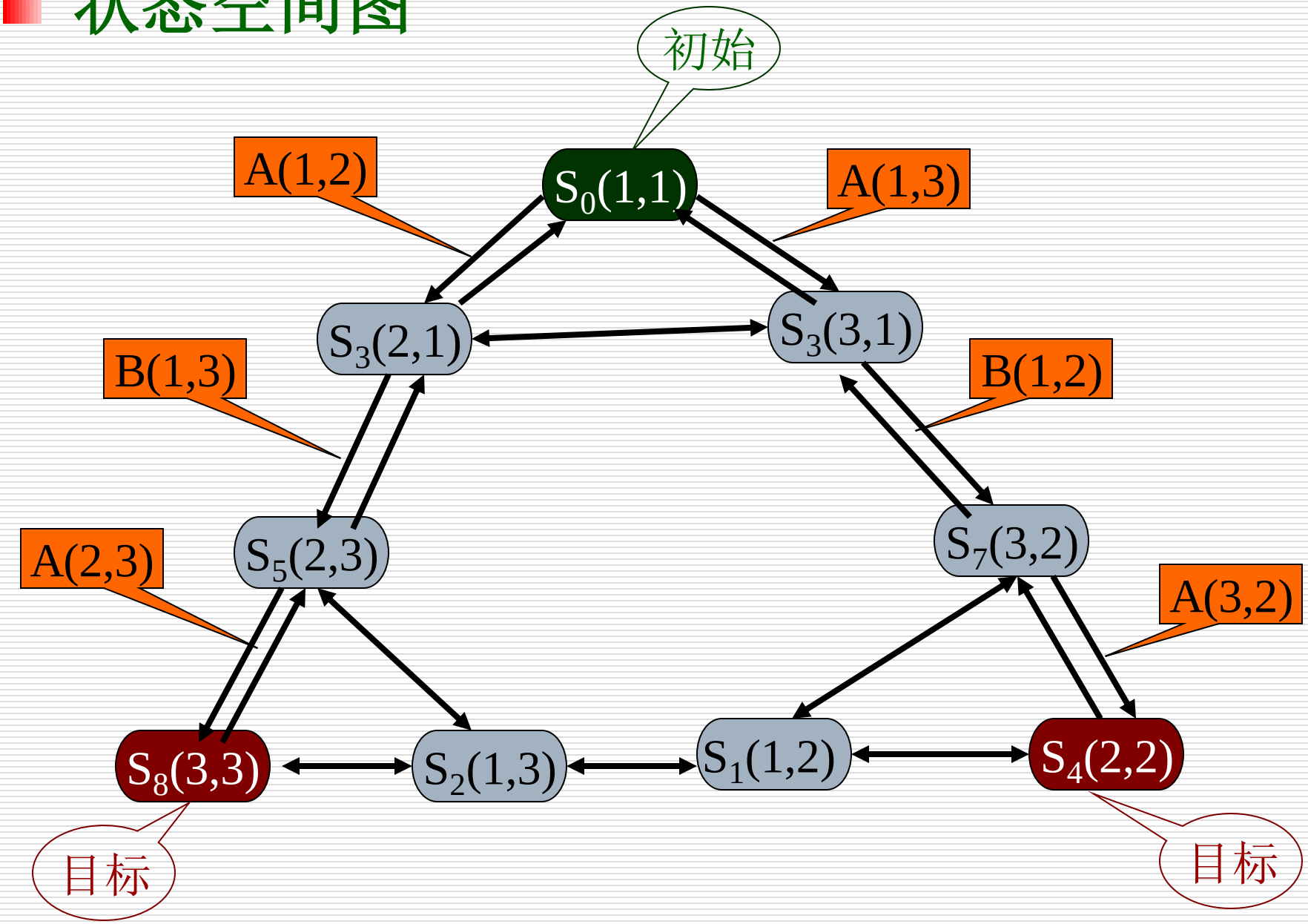
5.2 状态空间表示法

- **算符如何定义？**
  - 定义算符A(i,j)表示把A从i移到j；B(i,j)表示把B从i移到j。
  - 算符集合(共12个算符）：

    A(1,2),A(1,3),A(2,1),A(2,3),A(3,1),A(3,2)

    B(1,2),B(1,3),B(2,1),B(2,3),B(3,1),B(3,2)

状态空间图

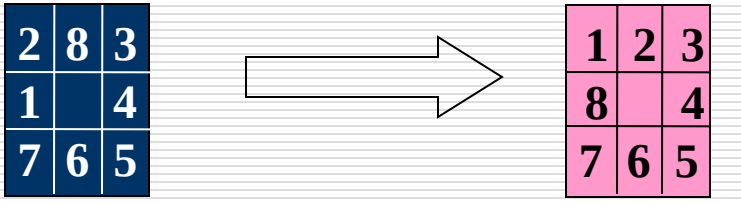

初始
A(1,2)
A(1,3)
$S_0(1,1)$
$S_3(2,1)$
$S_3(3,1)$
B(1,3)
B(1,2)
A(2,3)
$S_5(2,3)$
$S_7(3,2)$
A(3,2)
$S_8(3,3)$
$S_2(1,3)$
$S_1(1,2)$
$S_4(2,2)$
目标
目标

5.2 状态空间表示法

例2 八数码问题状态空间表示

八数码问题。

在3×3的方格棋盘上放置八张牌，初始状态和目标状态如下图。



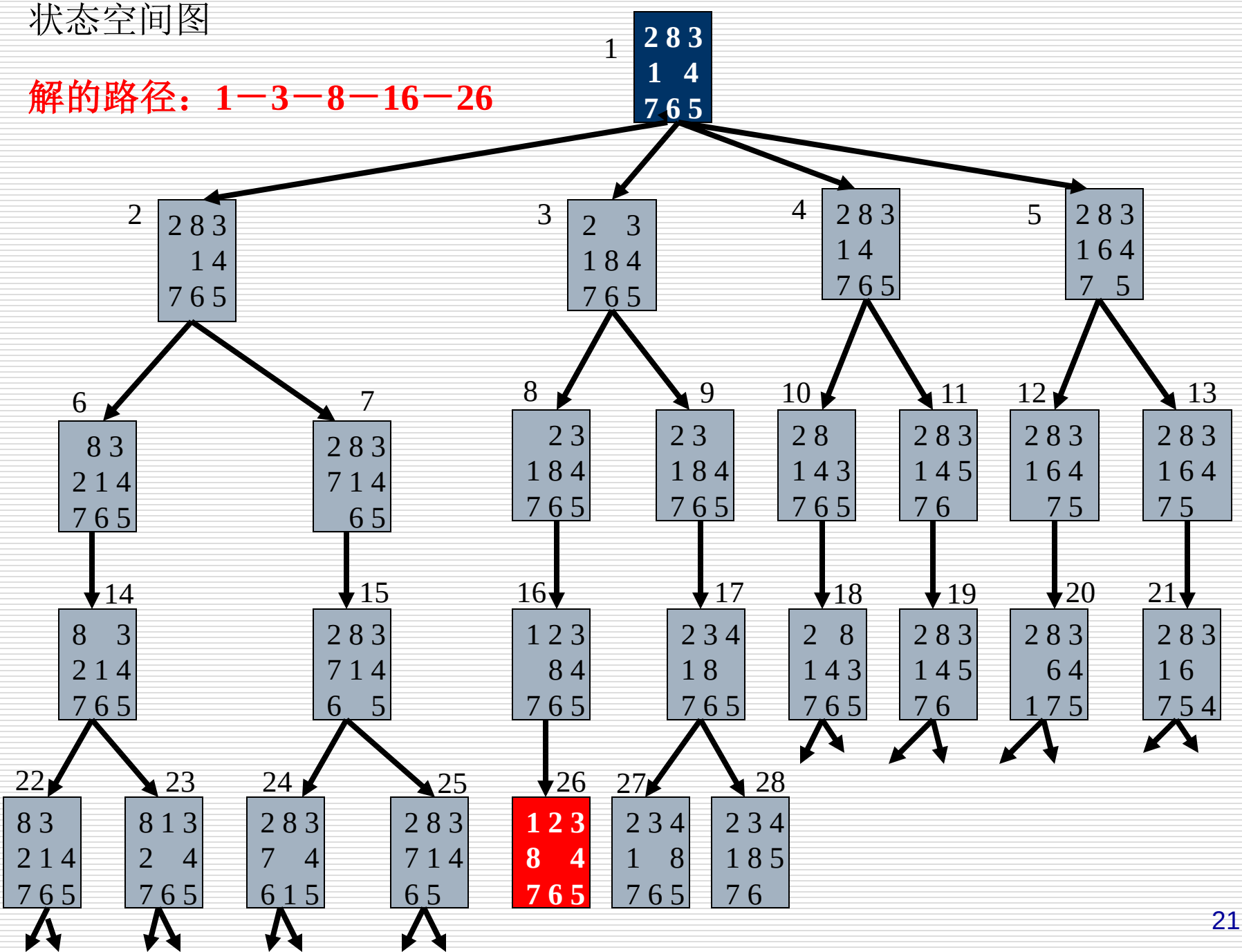
初始状态　　目标状态

状态集 S ：所有摆法　　操作算符？

将空格向上移Up
将空格向左移Left
将空格向下移Down
将空格向右移Right

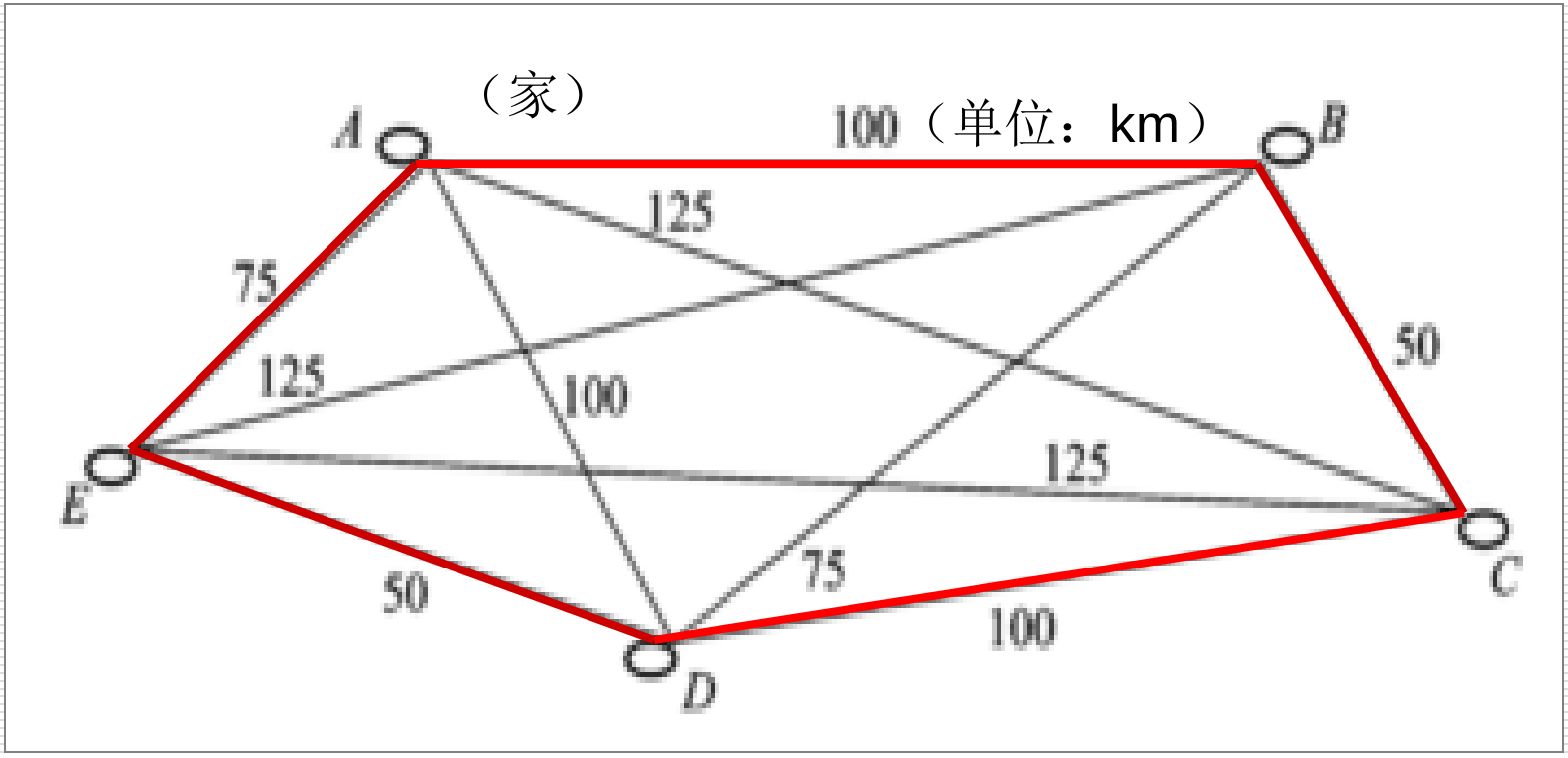
状态空间图

解的路径：1—3—8—16—26



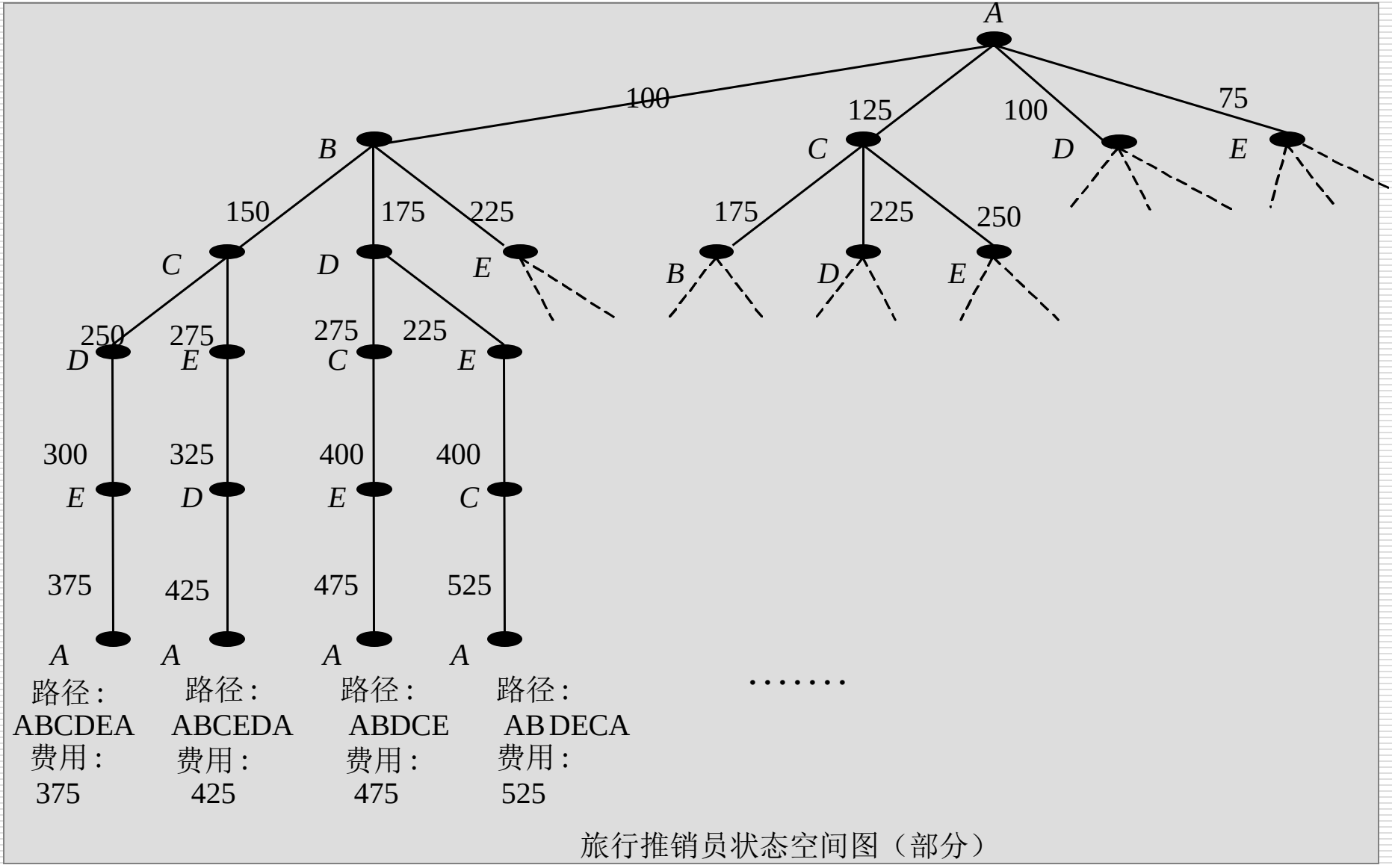
5.2 状态空间表示法

例3　旅行商问题（traveling salesman problem, TSP）或邮递员路径问题。



可能路径：费用为375的路径（*A*，*B*，*C*，*D*，*E*，*A*）

5.2 状态空间表示法



旅行推销员状态空间图（部分）

5.2 状态空间表示法

状态空间表示法用“状态”和“算符”来表示问题

- 状态—描述问题求解过程不同时刻的状态
- 算符—表示对状态的操作

算符的每一次使用就使状态发生变化。当到达目标状态时，由**初始状态到目标状态所使用的算符序列就是问题的一个解**。

第5章 搜索求解策略

- 5.1 搜索的概念
- 5.2 状态空间表示法
- ✓ 5.3 状态空间盲目搜索
- 5.4 状态空间启发式搜索
- 5.5 与或树表示
- 5.6 与或树盲目搜索
- 5.7 与或树启发式搜索
- 5.8 博弈树搜索

5.3 状态空间盲目搜索

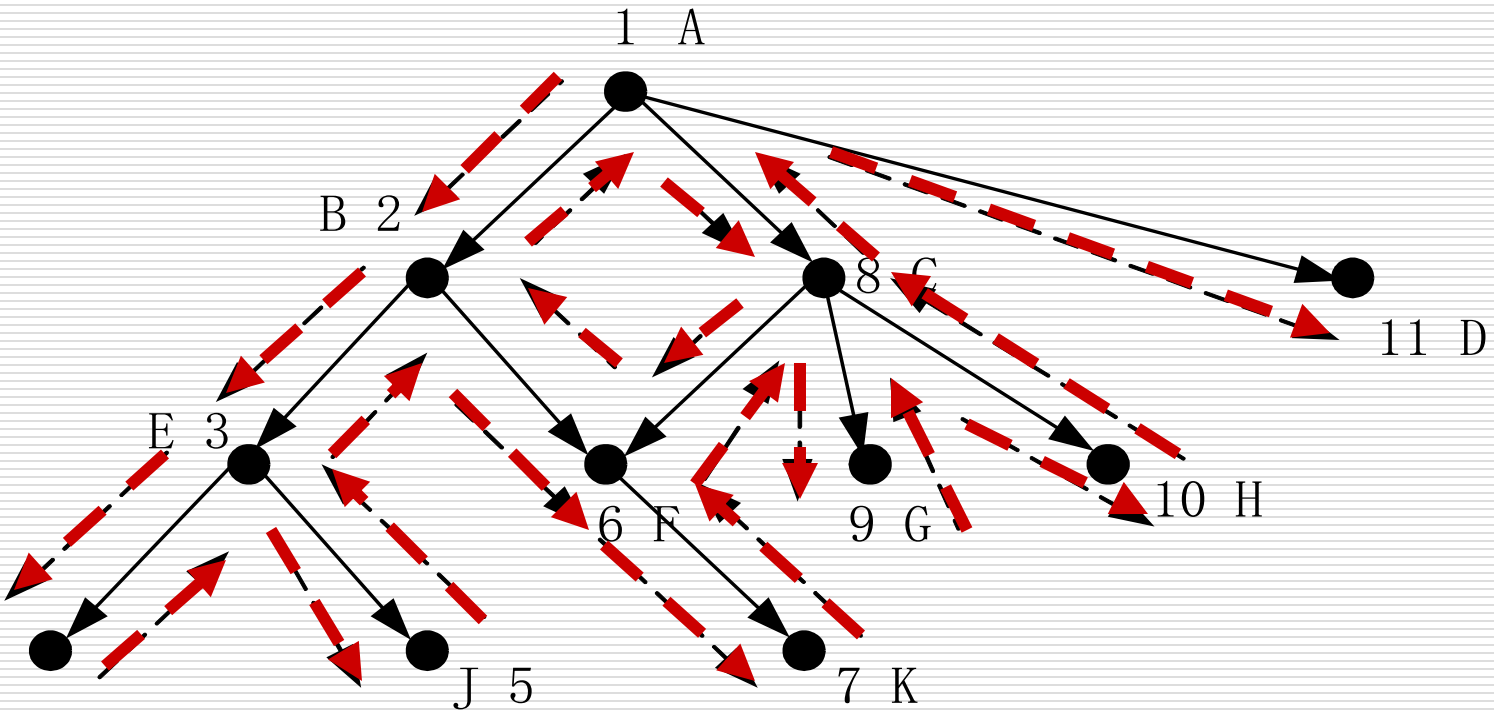
- **5.3.1 回溯策略**
- **5.3.2 宽度优先搜索策略**
- **5.3.3 深度优先搜索策略**

5.3.1 回溯策略

- **带回溯策略的搜索：**

从初始状态出发，不停地、试探性地寻找路径，直到它到达目的或“不可解结点”，即“死胡同”为止。若它遇到不可解结点就回溯到路径中最近的父结点上，查看该结点是否还有其他的子结点未被扩展。若有，则沿这些子结点继续搜索；如果找到目标，就成功退出搜索，返回解题路径。

5.3.1 回溯策略



回溯搜索示意图

5.3.1 回溯策略

- **回溯搜索的算法**

(1) PS（path states）表：保存当前搜索路径上的状态。如果找到了目的，PS就是解路径上的状态有序集。

(2) NPS（new path states）表：新的路径状态表。它包含了等待搜索的状态，其后裔状态还未被搜索到，即未被生成扩展 。

(3) NSS（no solvable states）表：不可解状态集，列出了找不到解题路径的状态。如果在搜索中扩展出的状态是它的元素，则可立即将之排除，不必沿该状态继续搜索。

5.3.1 回溯策略

- 搜索算法的回溯思想：

（1）用未处理状态表（NPS）使算法能返回（回溯）到其中任一状态。

（2）用一张“死胡同”状态表（NSS）来避免算法重新搜索无解的路径。

（3）在PS 表中记录当前搜索路径的状态，当满足目的时可以将它作为结果返回。

（4）为避免陷入死循环必须对新生成的子状态进行检查，看它是否在该三张表中 。

5.3 状态空间盲目搜索

- **5.3.1 回溯策略**
- **5.3.2 宽度优先搜索策略**
- **5.3.3 深度优先搜索策略**

5.3 状态空间盲目搜索

1、特点：

1)搜索按规定的路线进行，不使用与问题有关的启发性信息

2)适用于其状态空间图是树状结构的问题求解。

2、搜索过程：

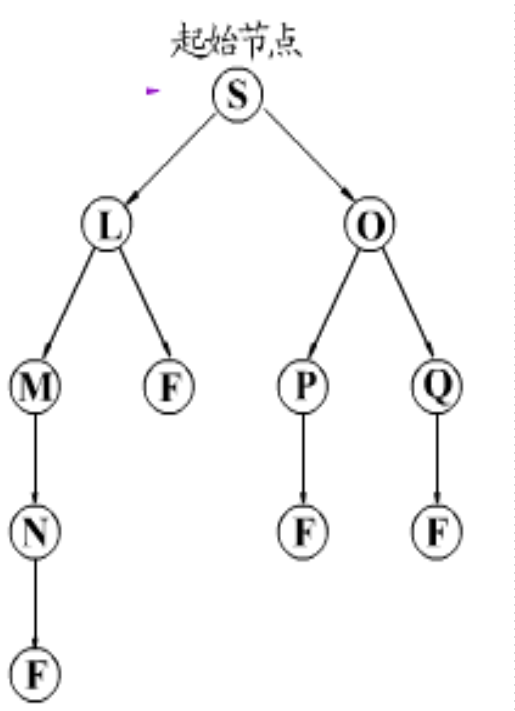
一般都要用两个表，这就是OPEN表和CLOSED 表。

OPEN表用于待考察的节点。

CLOSED表用于存放已考察的节点。

3、当目标状态出现的时候，结束搜索。

5.3.2 宽度优先搜索



宽度优先搜索示意图

宽度优先搜索过程：

1） 把起始节点放到OPEN表中。

2） 如果OPEN是个空表，则没有解，失败退出；否则继续。

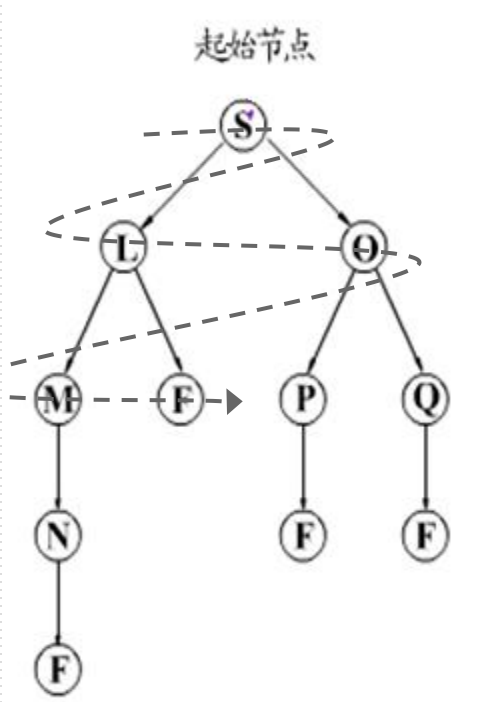
3） 把第一个节点（节点 n）从OPEN表移出，并把它放入CLOSED扩展节点表中。

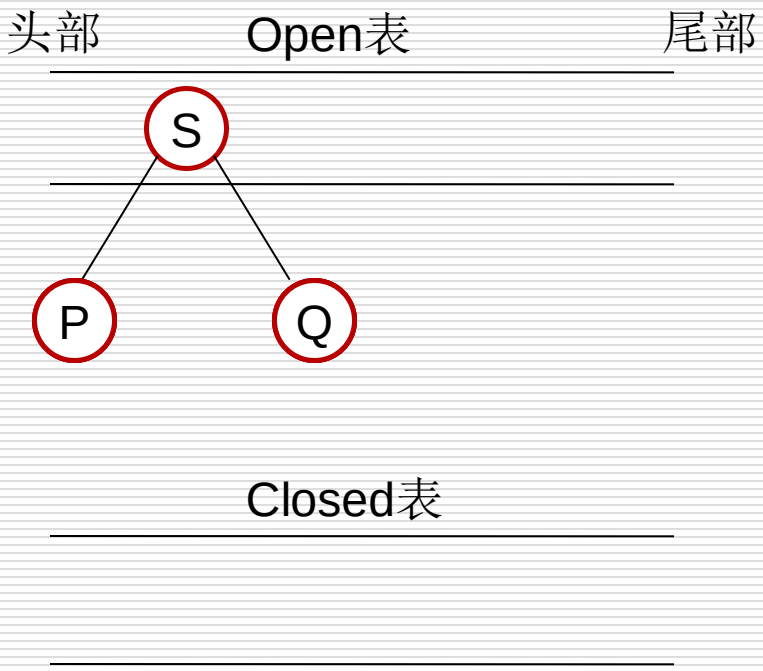
4） 考察节点n是否为目标节点。如果是，则求得了问题的解，退出。

5） 如果节点n不可扩展，则转第2）步。

6） 把n 的所有子节点放到**OPEN表的尾部**，并为其配置指向父节点的指针，然后转第2）步

5.3.2 宽度优先搜索

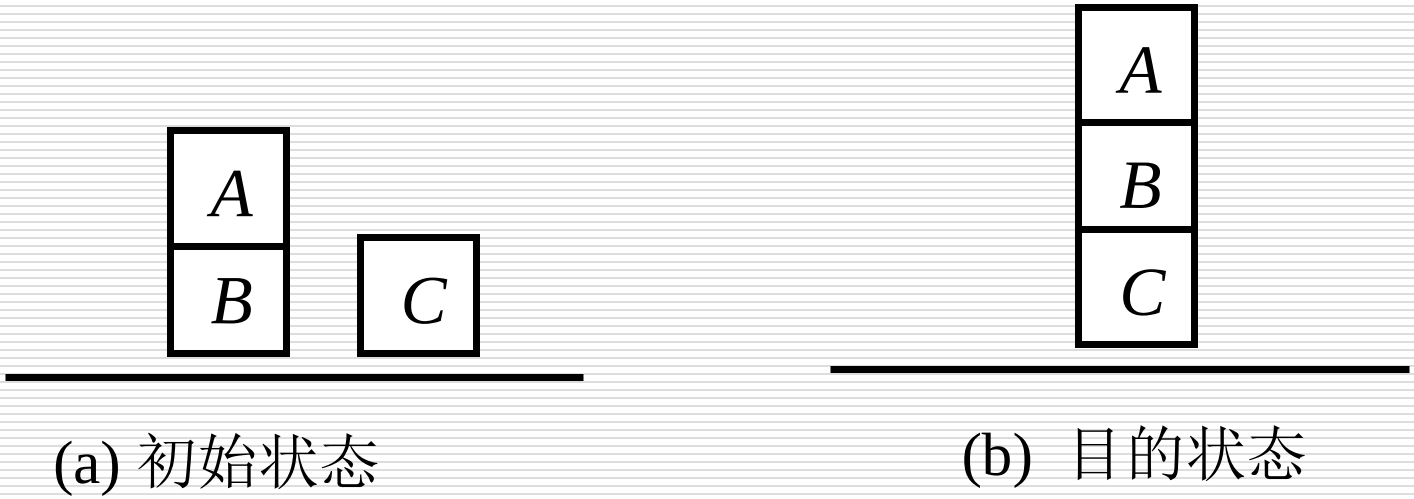




宽度优先搜索示意图

5.3.2 宽度优先搜索

- **例5.4** 通过搬动积木块，希望从初始状态达到一个目的状态，即三块积木堆叠在一起。



(a) 初始状态　　(b) 目的状态

积木问题

5.3.2 宽度优先搜索策略

- 操作算子为*MOVE*（*X*，*Y*）：把积木*X*搬到*Y*（积木或桌面）上面。

MOVE（*A*，*Table*）：“搬动积木*A*到桌面上”。

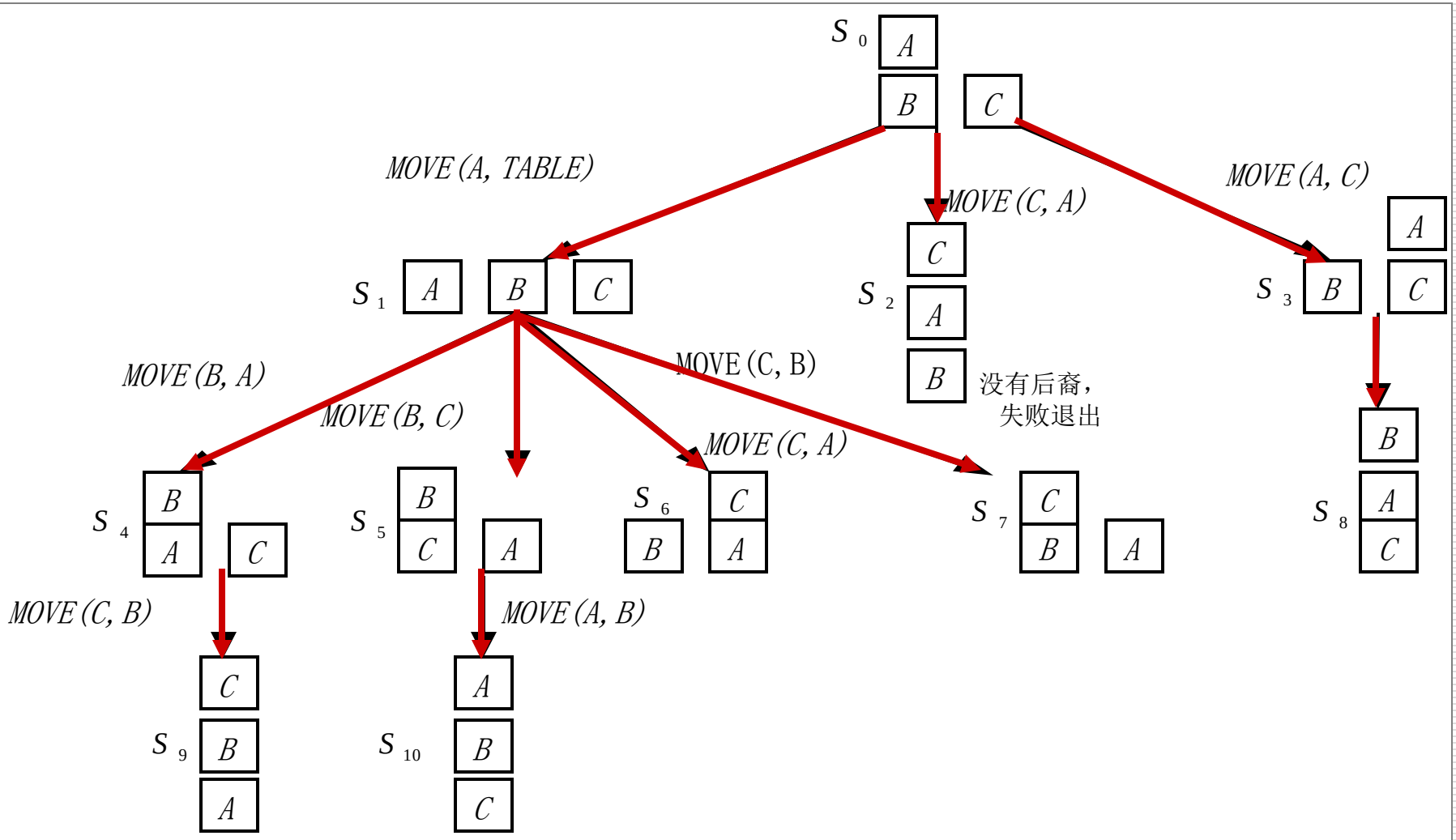
- 操作算子可运用的先决条件：

（1）被搬动积木的顶部必须为空。

（2）如果 *Y* 是积木，则积木 *Y* 的顶部也必须为空。

（3）同一状态下，运用操作算子的次数不得多于一次。

5.3.2 宽度优先搜索

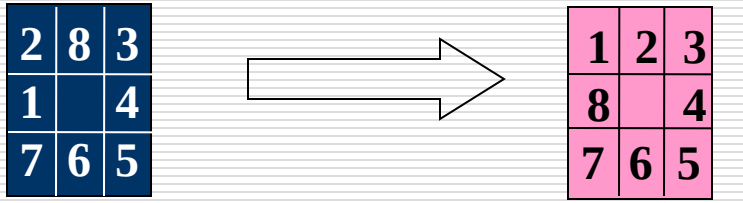


积木问题的宽度优先搜索树

5.3.2 宽度优先搜索

八数码问题。

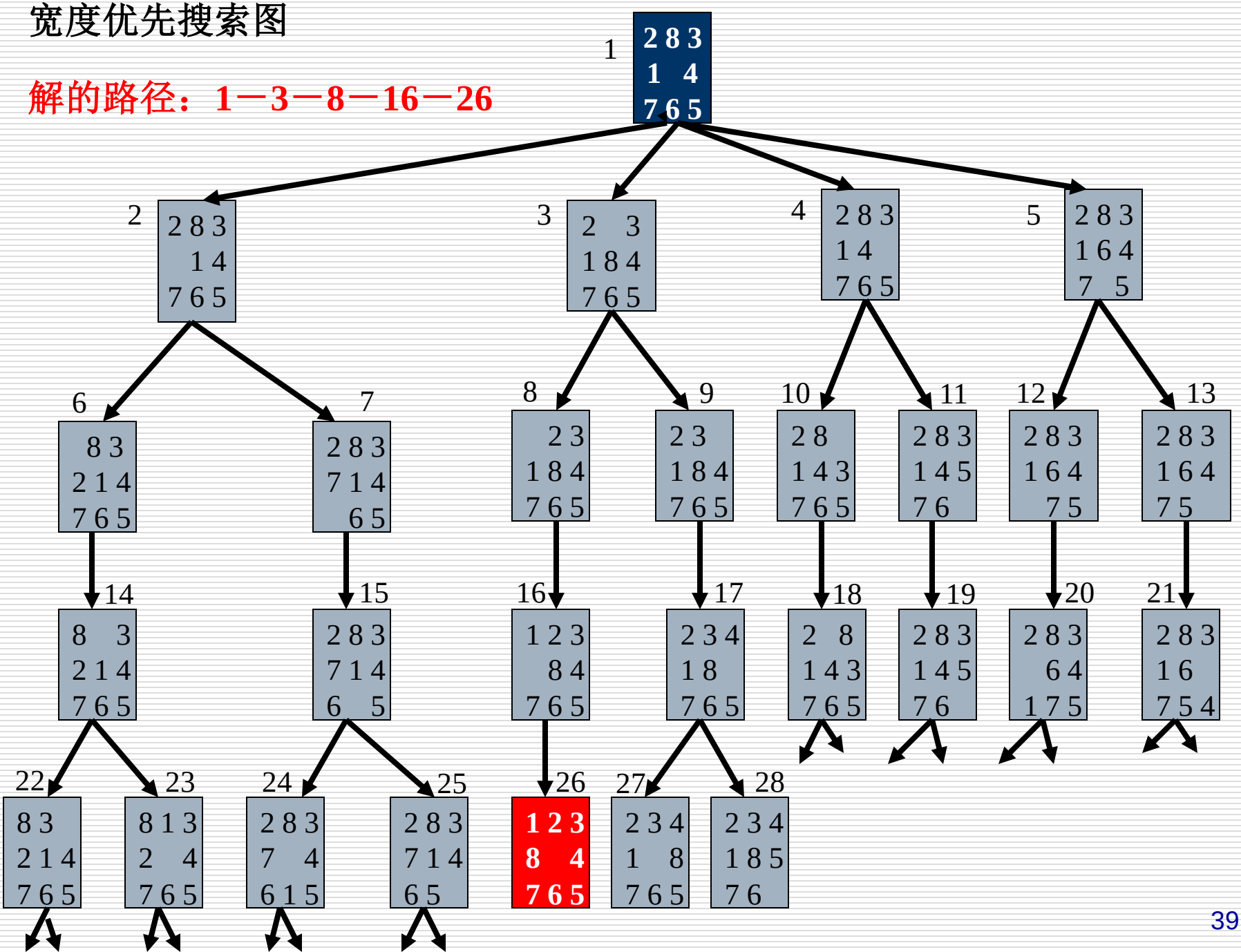
在3×3的方格棋盘上放置八张牌，初始状态和目标状态如下图。



初始状态　　目标状态

宽度优先搜索图

解的路径：1—3—8—16—26



- Open表的变化(宽度优先搜索）　初始 (1)

1　(2,3,4,5)

2　(3,4,5,6,7)

3　(4,5,6,7,8,9)

4　(5,6,7,8,9,10,11)

5　(6,7,8,9,10,11,12,13)

6　(7,8,9,10,11,12,13,14)

7　(8,9,10,11,12,13,14,15,)

8　(9,10,11,12,13,14,15,16)

9　(10,11,12,13,14,15,16,17)

10　(11,12,13,14,15,16,17,18)

11　(12,13,14,15,16,17,18,19)

12　(13,14,15,16,17,18,19,20)

13　(14,15,16,17,18,19,20,21)

14　(15,16,17,18,19,20,21,22,23)

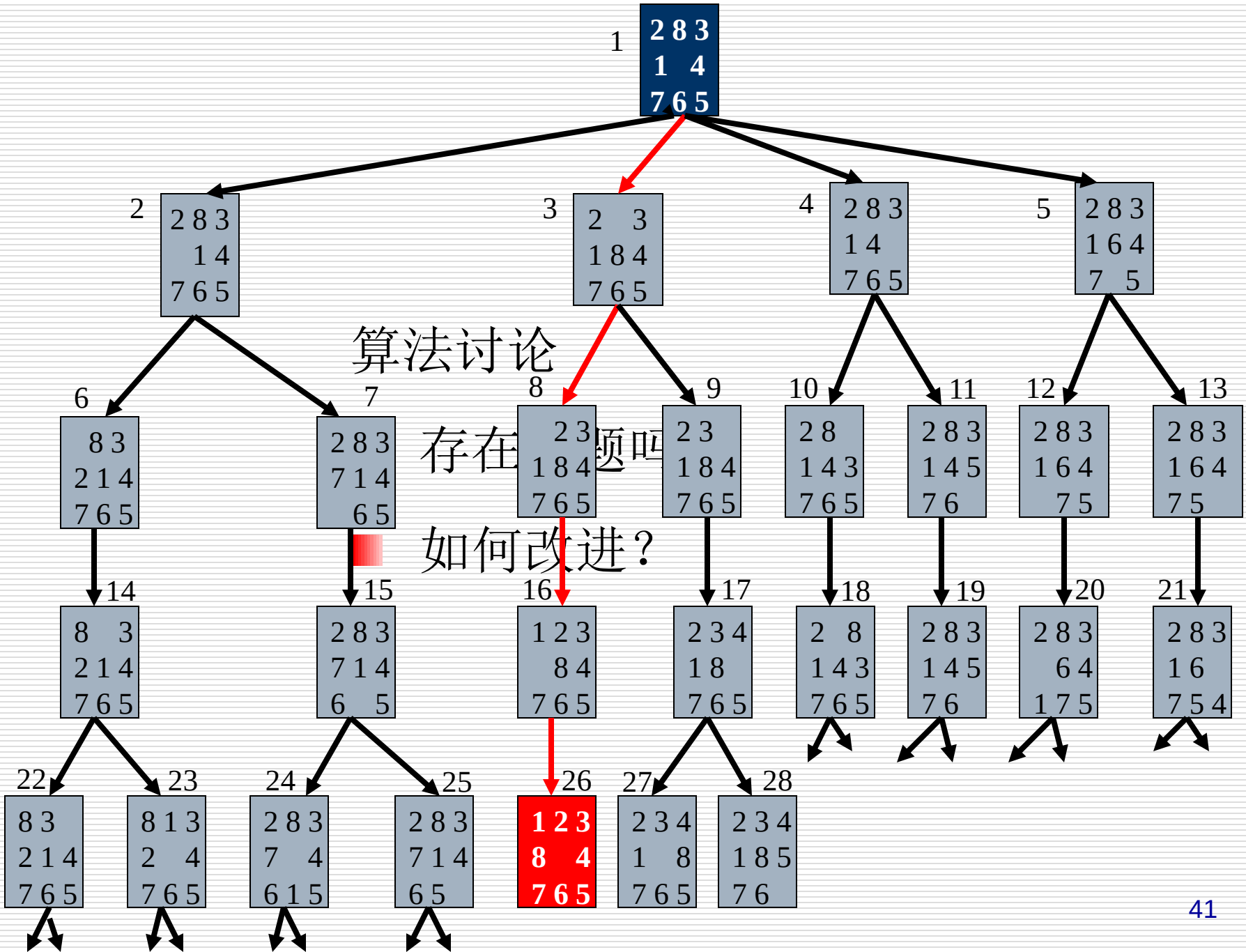
15　(16,17,18,19,20,21,22,23,24,25)

16　(17,18,19,20,21,22,23,24,25,26)

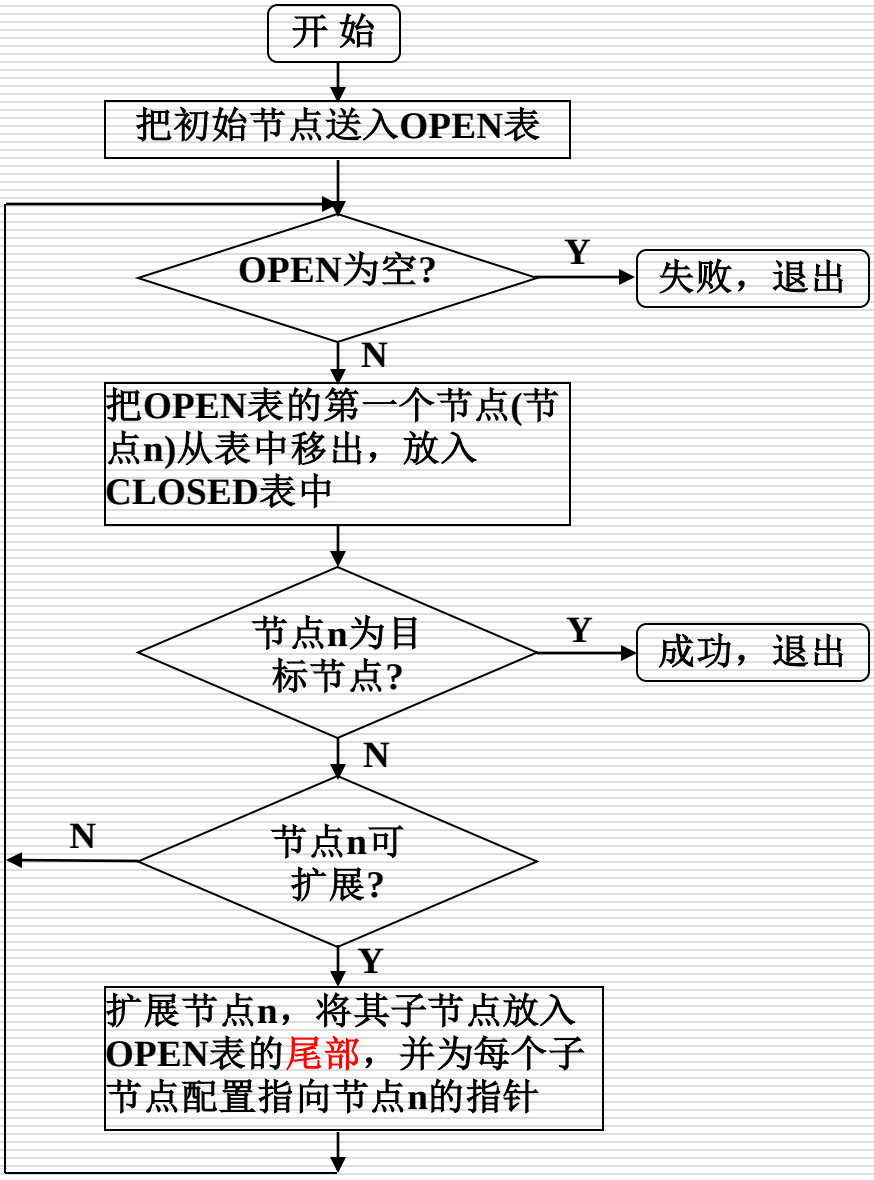
..　…　….　… ……

25　(26,27,28,……)

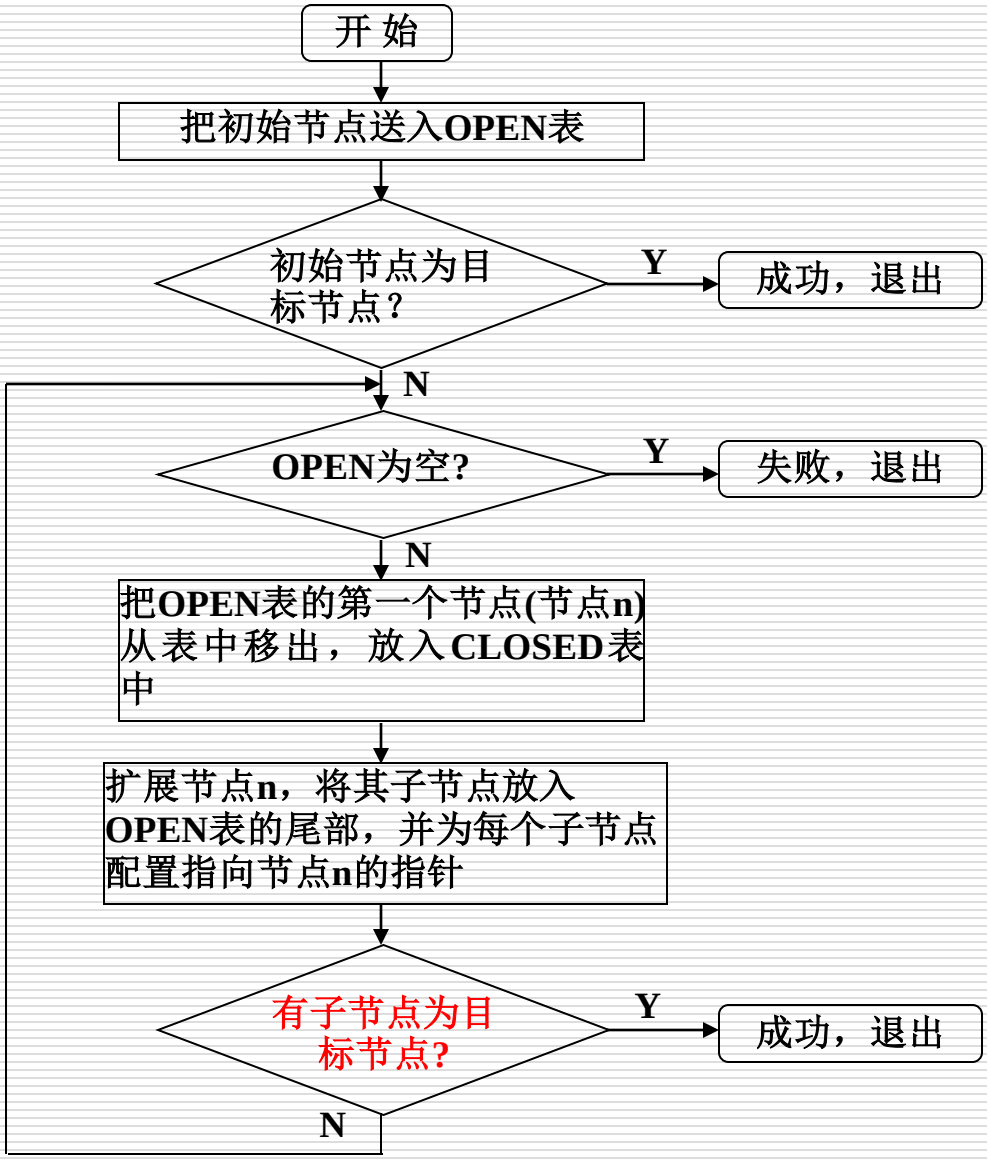
26　找到目标节点。


1
2 8 3
1 4
7 6 5
2
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
4
2 8 3
1 4
7 6 5
5
2 8 3
1 6 4
7 5
算法讨论
存在问题吗
如何改进？
6
8 3
2 1 4
7 6 5
7
2 8 3
7 1 4
6 5
8
2 3
1 8 4
7 6 5
9
2 3
1 8 4
7 6 5
10
2 8
1 4 3
7 6 5
11
2 8 3
1 4 5
7 6
12
2 8 3
1 6 4
7 5
13
2 8 3
1 6 4
7 5
14
8 3
2 1 4
7 6 5
15
2 8 3
7 1 4
6 5
16
1 2 3
8 4
7 6 5
17
2 3 4
1 8
7 6 5
18
2 8
1 4 3
7 6 5
19
2 8 3
1 4 5
7 6
20
2 8 3
6 4
1 7 5
21
2 8 3
1 6
7 5 4
22
8 3
2 1 4
7 6 5
23
8 1 3
2 4
7 6 5
24
2 8 3
7 4
6 1 5
25
2 8 3
7 1 4
6 5
26
1 2 3
8 4
7 6 5
27
2 3 4
1 8
7 6 5
28
2 3 4
1 8 5
7 6

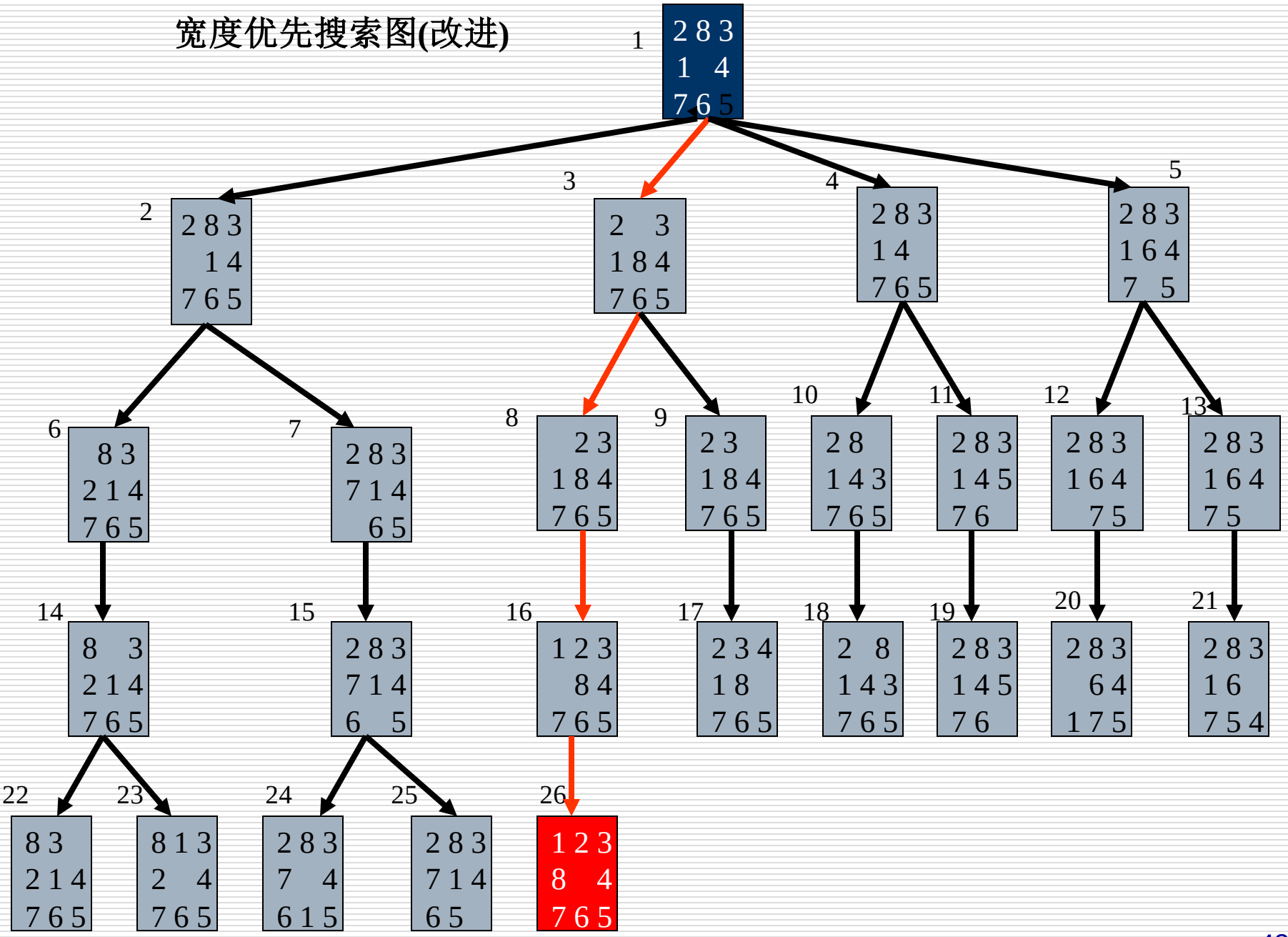
宽度优先搜索



宽度优先搜索(改进)



宽度优先搜索图(改进)


1
2 8 3
1 4
7 6 5
2
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
4
2 8 3
1 4
7 6 5
5
2 8 3
1 6 4
7 5
6
8 3
2 1 4
7 6 5
7
2 8 3
7 1 4
6 5
8
2 3
1 8 4
7 6 5
9
2 3
1 8 4
7 6 5
10
2 8
1 4 3
7 6 5
11
2 8 3
1 4 5
7 6
12
2 8 3
1 6 4
7 5
13
2 8 3
1 6 4
7 5
14
8 3
2 1 4
7 6 5
15
2 8 3
7 1 4
6 5
16
1 2 3
8 4
7 6 5
17
2 3 4
1 8
7 6 5
18
2 8
1 4 3
7 6 5
19
2 8 3
1 4 5
7 6
20
2 8 3
6 4
1 7 5
21
2 8 3
1 6
7 5 4
22
8 3
2 1 4
7 6 5
23
8 1 3
2 4
7 6 5
24
2 8 3
7 4
6 1 5
25
2 8 3
7 1 4
6 5
26
1 2 3
8 4
7 6 5

- Open表的变化(改进的宽度优先搜索）

初始 (1)

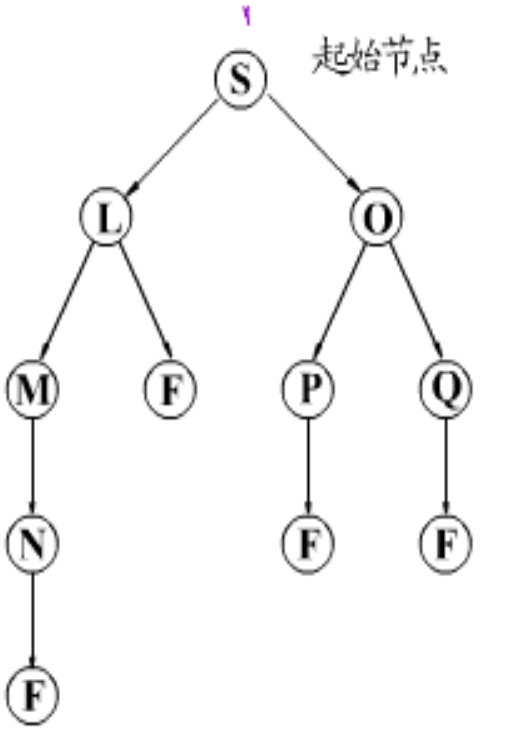
1	(2,3,4,5)
2	(3,4,5,6,7)
3	(4,5,6,7,8,9)
4	(5,6,7,8,9,10,11)
5	(6,7,8,9,10,11,12,13)
6	(7,8,9,10,11,12,13,14)
7	(8,9,10,11,12,13,14,15,)
8	(9,10,11,12,13,14,15,16)
9	(10,11,12,13,14,15,16,17)
10	(11,12,13,14,15,16,17,18)
11	(12,13,14,15,16,17,18,19)
12	(13,14,15,16,17,18,19,20)
13	(14,15,16,17,18,19,20,21)
14	(15,16,17,18,19,20,21,22,23)
15	(16,17,18,19,20,21,22,23,24,25)
16	(17,18,19,20,21,22,23,24,25,)

26为16的子节点，是目标节点，得到解

5.3 状态空间盲目搜索

- **5.3.1 回溯策略**
- **5.3.2 宽度优先搜索策略**
- **5.3.3 深度优先搜索策略**

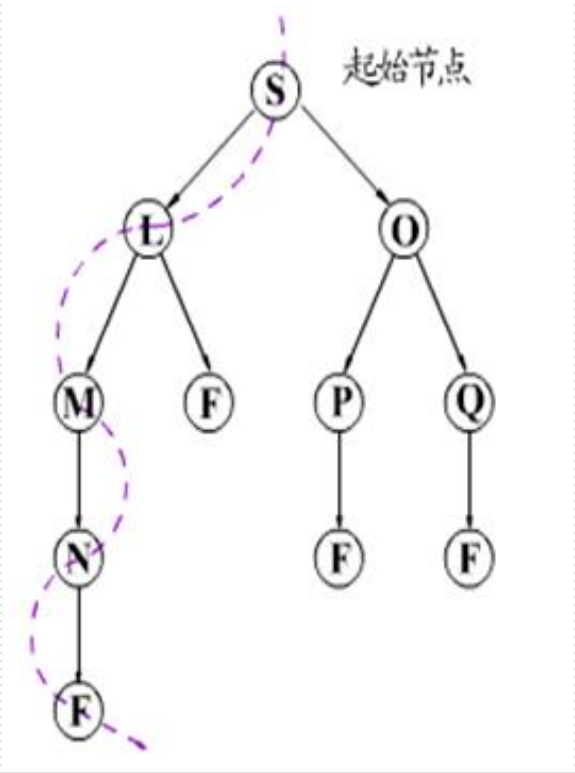
5.3.3 深度优先搜索

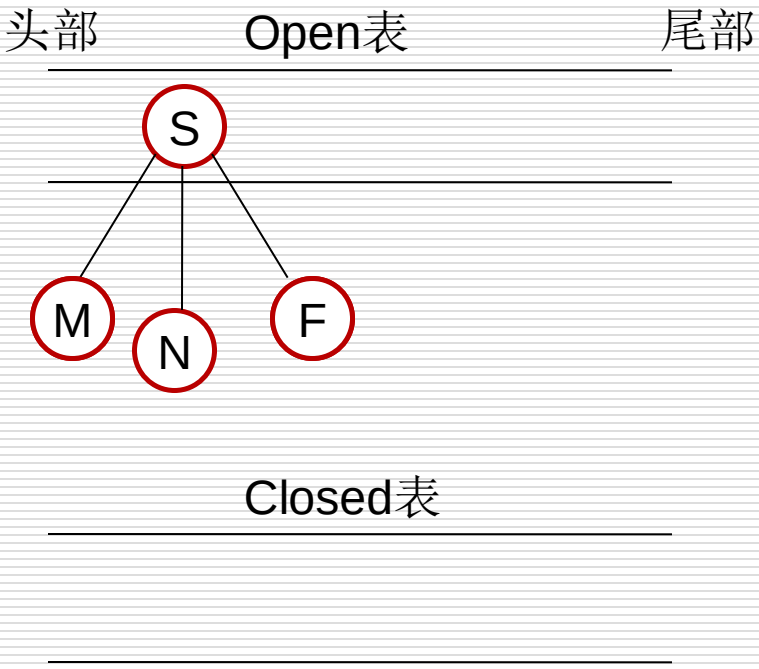


深度优先搜索示意图

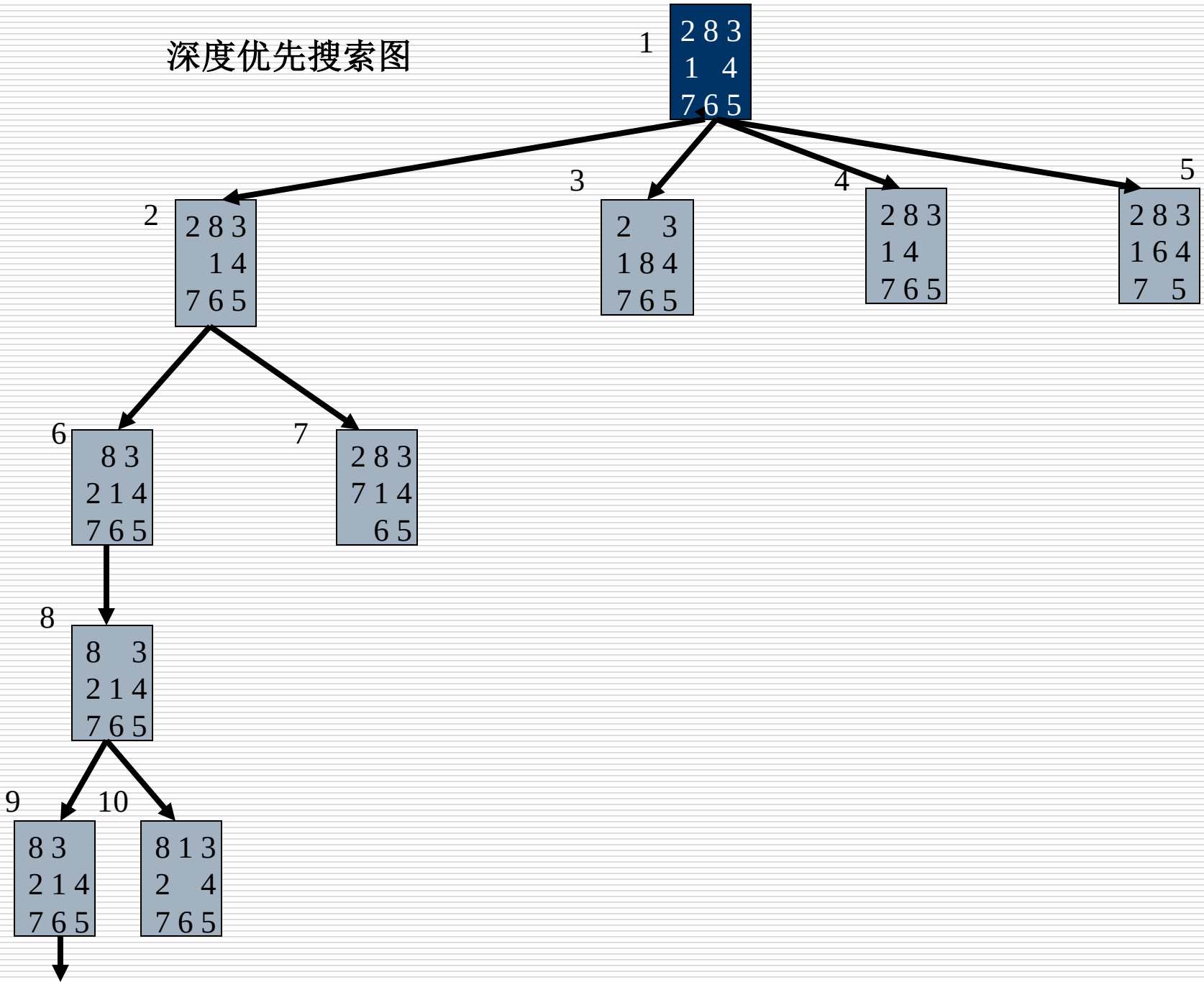
- **基本思想：**
- 从初始节点S0开始，在其子节点中选择一个节点进行扩展并考察它是否为目标节点，若不是目标节点，则在该子节点的子节点中选择一个节点进行考察，一直如此向下搜索。当到达某个子节点，且该子节点即不是目标节点又不能继续扩展时，才选择其兄弟节点进行扩展。
- 节点按后进入open表的顺序排列，即**后进入的节点排在表的最前面**

5.3.3 深度优先搜索

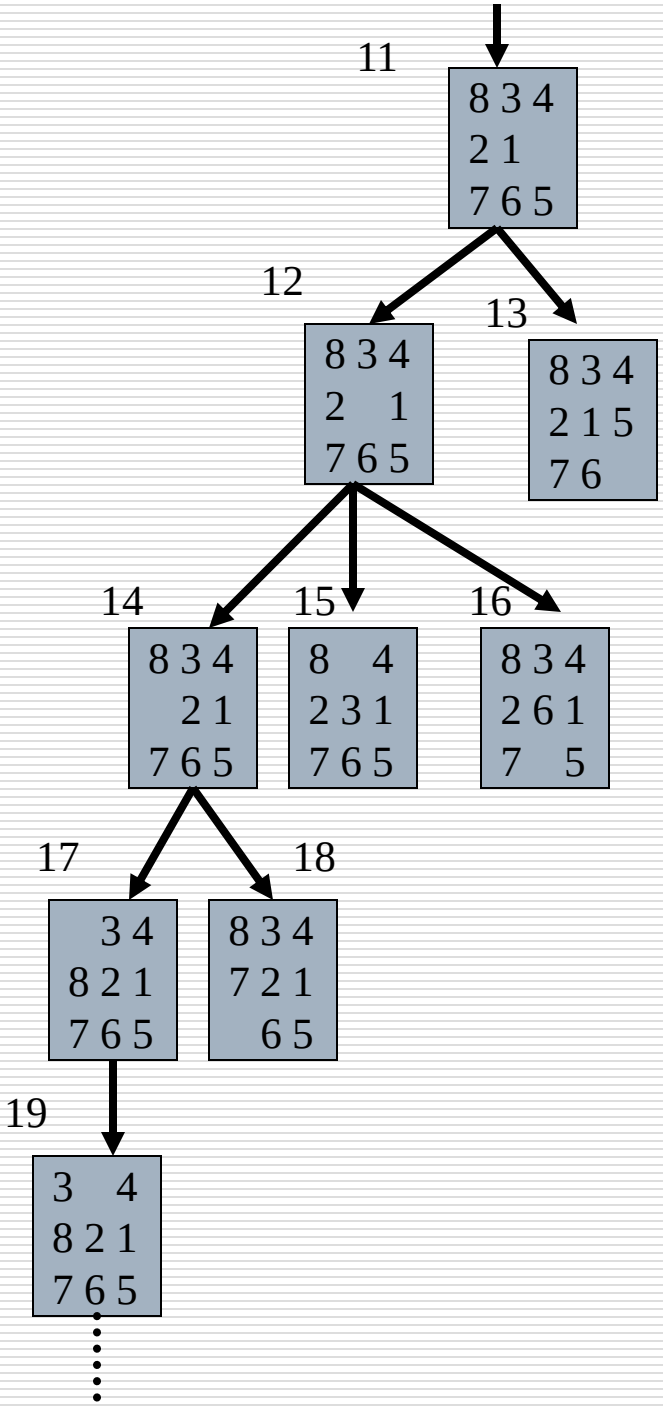




深度优先搜索图


1
2 8 3
1 4
7 6 5
2
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
4
2 8 3
1 4
7 6 5
5
2 8 3
1 6 4
7 5
6
8 3
2 1 4
7 6 5
7
2 8 3
7 1 4
6 5
8
8 3
2 1 4
7 6 5
9
8 3
2 1 4
7 6 5
10
8 1 3
2 4
7 6 5

深度优先搜索图（续）


11
8 3 4
2 1
7 6 5
12
8 3 4
2 1
7 6 5
13
8 3 4
2 1 5
7 6
14
8 3 4
2 1
7 6 5
15
8 4
2 3 1
7 6 5
16
8 3 4
2 6 1
7 5
17
3 4
8 2 1
7 6 5
18
8 3 4
7 2 1
6 5
19
3 4
8 2 1
7 6 5

5.3.3 深度优先搜索

- **Open表(深度优先搜索)**

初始（1）

1 （2,3,4,5）

2 (6,7,3,4,5)

3 (8,7,3,4,5)

4 (9,10,7,3,4,5)

5 (11,10,7,3,4,5)

6 (12,13,10,7,3,4,5)

7 (14,15,16,13,10,7,3,4,5)

8 (17,18,15,16,13,10,7,3,4,5)

9 (19,18,15,16,13,10,7,3,4,5)

……

…….

算法讨论

- 存在问题：无限搜索、无法找到最优解等。
- 改进方法：？

5.3.3 有界深度优先搜索

- 对深度优先搜索引入搜索深度的限制(设为dm)，当搜索深度达到深度界限时，尚未出现目标节点，就选择其兄弟节点进行扩展。

- 节点按后进入open表的顺序排列，即后进入的节点排在前面

- 深度的确定：**固定深度**

  可变深度

5.3.3 有界深度优先搜索

- 有界深度优先搜索算法如下：

(1) 把起始节点 S 放到OPEN表中。如果此节点为一目标节点，则得到一个解。
(2) 如果OPEN为一空表，则失败退出。
(3) 把第一个节点（节点n）从OPEN表移到CLOSED表。
(4) 如果节点n的深度等于最大深度dm，则转向(2) 。
(5) 扩展节点n ，产生其全部后裔，并把它们放入OPEN表的首部。如果没有后裔，则转向(2) 。
(6) 如果后继节点中有任一个为目标节点，则求得一个解，成功退出；否则，转向(2) 。

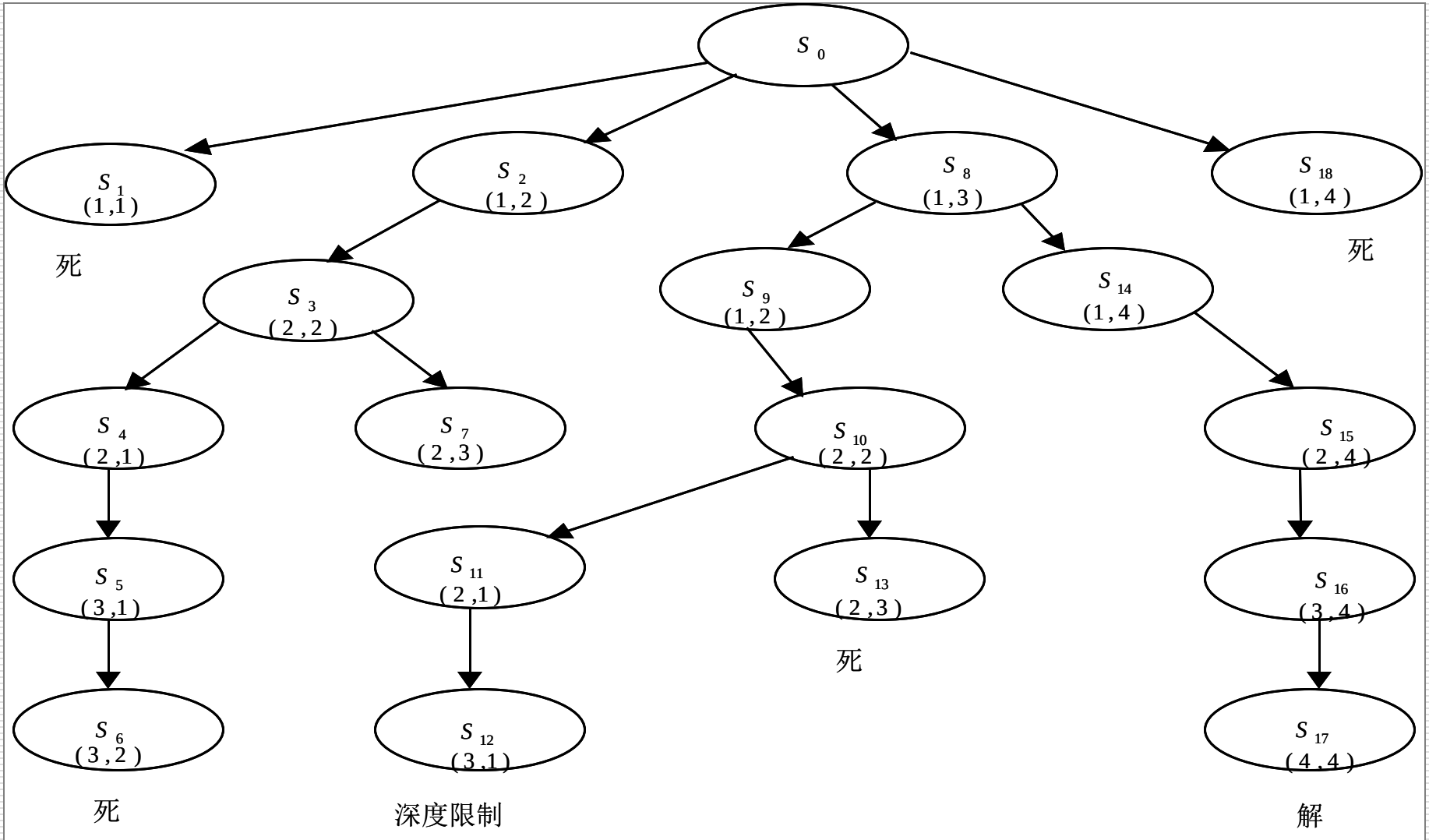
5.3.3 有界深度优先搜索

- **例5.5　卒子穿阵问题**，要求一卒子从顶部通过下图所示的阵列到达底部。卒子行进中不可进入到代表敌兵驻守的区域（标注1），并不准后退。假定深度界限值dm=5。

行	1	2	3	4	列
1	1	0	0	0	
2	0	0	1	0	
3	0	1	0	0	
4	1	0	0	0	

阵列图

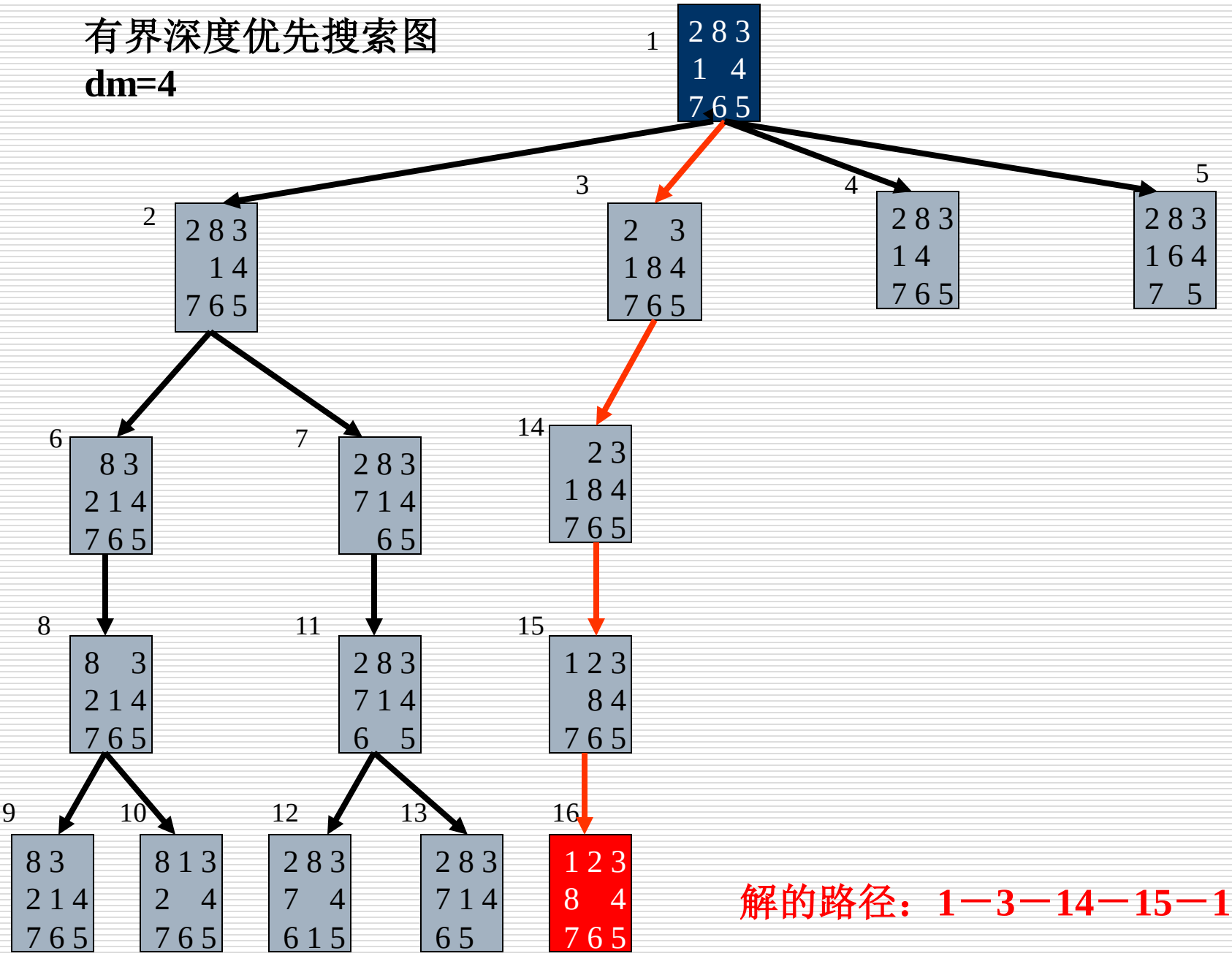
5.3.3 有界深度优先搜索



卒子穿阵的深度优先搜索树

有界深度优先搜索图

dm=4



解的路径：1—3—14—15—16

5.3.3 有界深度优先搜索

- **Open表(有界深度优先搜索)**

初始（1）

1 （2,3,4,5）

2 (6,7,3,4,5)

3 (8,7,3,4,5)

4 (9,10,7,3,4,5) dm=4

5 (10,7,3,4,5) dm=4

6 (7,3,4,5)

7 (11,3,4,5)

8 (12,13,3,4,5) dm=4

9 (13,3,4,5) dm=4

10 (3,4,5)

11 (14,4,5)

12 (15,4,5)

13 (16,4,5)

14 16 为目标节点

5.3.3 有界深度优先搜索

算法讨论

- 存在问题：

  1、dm的值很难给出。

  2、不能保证找到最优解。

- 改进方法：

  可变深度

5.3.3 有界深度优先搜索

- **有界深度优先搜索改进的基本思想：**
- 先任意给定一个较小的数作为dm，然后进行上述的有界深度优先搜索，当搜索到达了指定的深度界限dm仍未发现目标节点，并且CLOSED表中仍有待扩展节点时，就将这些节点送回OPEN表，同时增大深度界限dm继续向下搜索。如此不断地增大dm，只要问题有解，就一定可以找到它。

5.3.3 深度优先搜索

- 深度优先搜索
- 有界深度优先搜索
- 有界深度优先搜索改进

5.3 状态空间盲目搜索

- **状态空间盲目搜索策略**

  宽度/ 深度优先搜索

  有界深度优先搜索及其改进

搜索按规定的路线进行，不使用与问题有关的启发性信息，在决定被扩展的节点时，都没有考察该节点在解的路径上可能性有多大以及它是否有利于问题求解和求出的解是否为最优解等，因此这些方法都具有**较大的盲目性，产生的无用节点较多，搜索空间较大，效率不高。**

第5章 搜索求解策略

- 5.1 搜索的概念
- 5.2 状态空间表示法
- 5.3 状态空间盲目搜索
- ✓ 5.4 状态空间启发式搜索
- 5.5 与或树表示
- 5.6 与或树盲目搜索
- 5.7 与或树启发式搜索
- 5.8 博弈树搜索

5.4 状态空间启发式搜索

- “**启发**”（heuristic）：关于发现和发明操作算子及搜索方法的研究。
- 在状态空间搜索中，**启发式**被定义成一系列操作算子，并能从状态空间中选择最有希望到达问题解的路径。
- **启发式策略**：利用与问题有关的启发信息进行搜索。

5.4 状态空间启发式搜索

- **运用启发式策略的两种基本情况：**

（1）一个问题由于在问题陈述和数据获取方面固有的模糊性，可能会使它没有一个确定的解。

（2）虽然一个问题可能有确定解，但是其状态空间特别大，搜索中生成扩展的状态数会随着搜索的深度呈指数级增长。

一字棋：9！，西洋跳棋：10^{78}，国际象棋：10^{120}，围棋：10^{761}。

假设每步可以搜索一个棋局，用极限并行速度（10^{-104}年/步）来处理，搜索一遍国际象棋的全部棋局也得10^{16}年即1亿亿年才可以算完！

5.4 状态空间启发式搜索

- 在搜索过程中，关键的一步是如何确定下一个要考察的节点，确定的方法不同就形成了不同的搜索策略。如果在确定节点时能充分利用与问题求解有关的控制信息，**估计出节点的重要性，**就能在搜索时选择重要性较高的节点，以利于求得最优解。

5.4 状态空间启发式搜索

- 用于估计节点的重要性的函数称为估计函数。其一般形式为：

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

- 其中g(x)为从初始节点S0到节点x已经实际付出的代价；h(x)是从节点x到目标节点Sg的最优路径的估计代价，h(x)称为启发函数，它体现了问题的启发性信息。
- 定义启发函数要根据具体问题具体分析，可以参考的思路有：

① 一个结点到目标结点的某种距离或差异的度量；

② 一个结点处在最佳路径上的概率；

③ 根据经验主观打分

5.4 状态空间启发式搜索

- **5.4.1 全局择优搜索**
- **5.4.2 图搜索**
- **5.4.3 A*算法**

5.4.1 全局择优搜索

- **全局择优搜索考察节点的特点：**

- 每次总是从**OPEN表的全体节点**中选择一个估价值最小的节点。

5.4.1 全局择优搜索

- 全局择优搜索过程如下：

(1) 把初始节点S0放入OPEN表，f(S0)。

(2) 如果OPEN表为空，则问题无解，退出。

(3) 把OPEN表的第一个节点(记为节点n)取出放入CLOSED表。

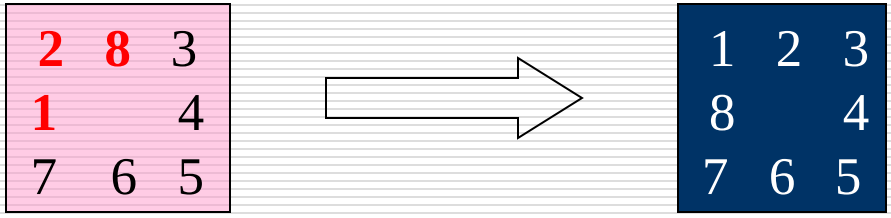
(4) 考察节点n是否为目标节点。若是，则求得问题的解，退出。

(5) 若节点n不可扩展，则转第(2)步。

(6) 扩展节点n，用估计函数f(x)计算每个子节点的估价值，并为每个子节点配置指向父节点的指针，把这些子节点都送入OPEN表中，然后对**OPEN表中的全部节点按估价值从小到大的顺序进行排序**，然后转第(2)步。

5.4.1 全局择优搜索

- **例子：用全局择优搜索求解八数码问题**

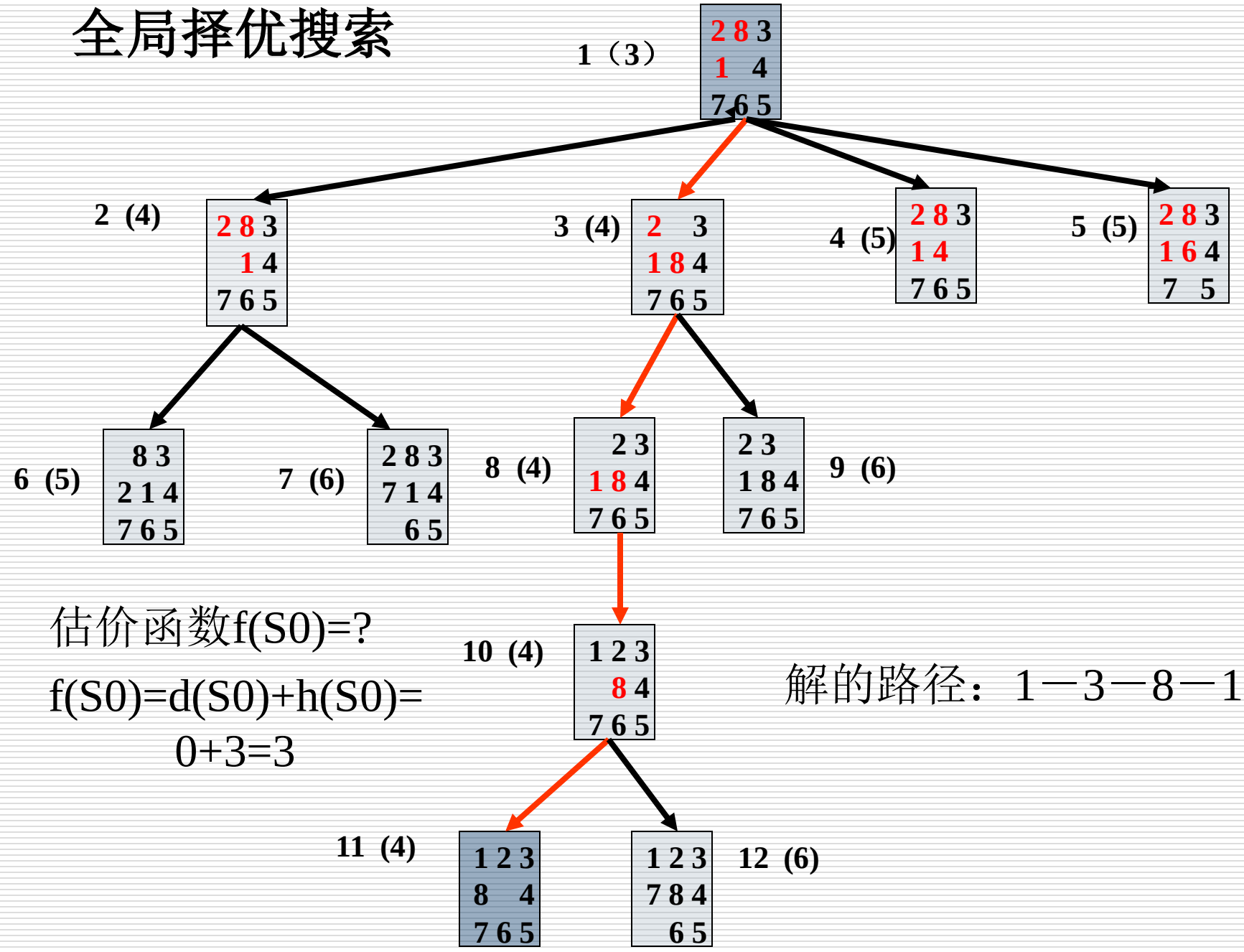


初始状态S0　　目标状态Sg

如何定义估计函数f(x)?

设估计函数：f(x) = d(x) + h(x)
其中：d(x)表示节点x的深度，h(x)表示节点x的棋局与目标节点棋局位置不相同的棋子数目。

全局择优搜索



估价函数f(S0)=?

f(S0)=d(S0)+h(S0)=

0+3=3

解的路径：1—3—8—10—11

5.4.1 全局择优搜索

- **八数码问题的估价函数**设计方法有多种，并且不同的估价函数对求解八数码问题有不同的影响。
  - 最简单的估价函数：取一格局与目的格局相比，其位置不符的数字个数。
  - 较好的估价函数：各数字移到目的位置所需移动的距离的总和。
  - 第三种估价函数：对每一对逆转数字乘以一个倍数。
  - 第四种估价函数：克服了仅计算数字逆转数目策略的局限，将位置不符数字个数的总和与3倍数字逆转数目相加。

5.4 状态空间启发式搜索

- **5.4.1 全局择优搜索**
- **5.4.2 有序搜索**
- **5.4.3 A*算法**

5.4.2 有序搜索

- 前面讨论的择优搜索仅适合于搜索的状态空间是树状结构，树状结构的状态空间中每个节点都只有唯一的父节点(根节点除外)。因此搜索算法放入到OPEN表中的节点的状态是各不相同的，不会有重复的节点。
- 如果状态空间是一个有向图的话，那么状态空间中的一个节点可能有多个父节点，因此，在OPEN表中会出现重复的节点，节点的重复将导致大量的冗余搜索，为此须对全局择优搜索进行修正。

5.4.2 有序搜索

- 修正的基本思想：

  当搜索过程生成一个节点i时，需要把节点i的状态与已生成的所有节点的状态进行比较，若节点i是一个已生成的节点，则表示找到一条通过节点i的新路径。若新路径使节点i的估价值更小，则修改节点i指向父节点的指针，使之指向新的父节点；否则，不修改节点i原有的父节点指针，即保留节点i原有的路径。[Nilsson,1971]

5.4.2 有序搜索

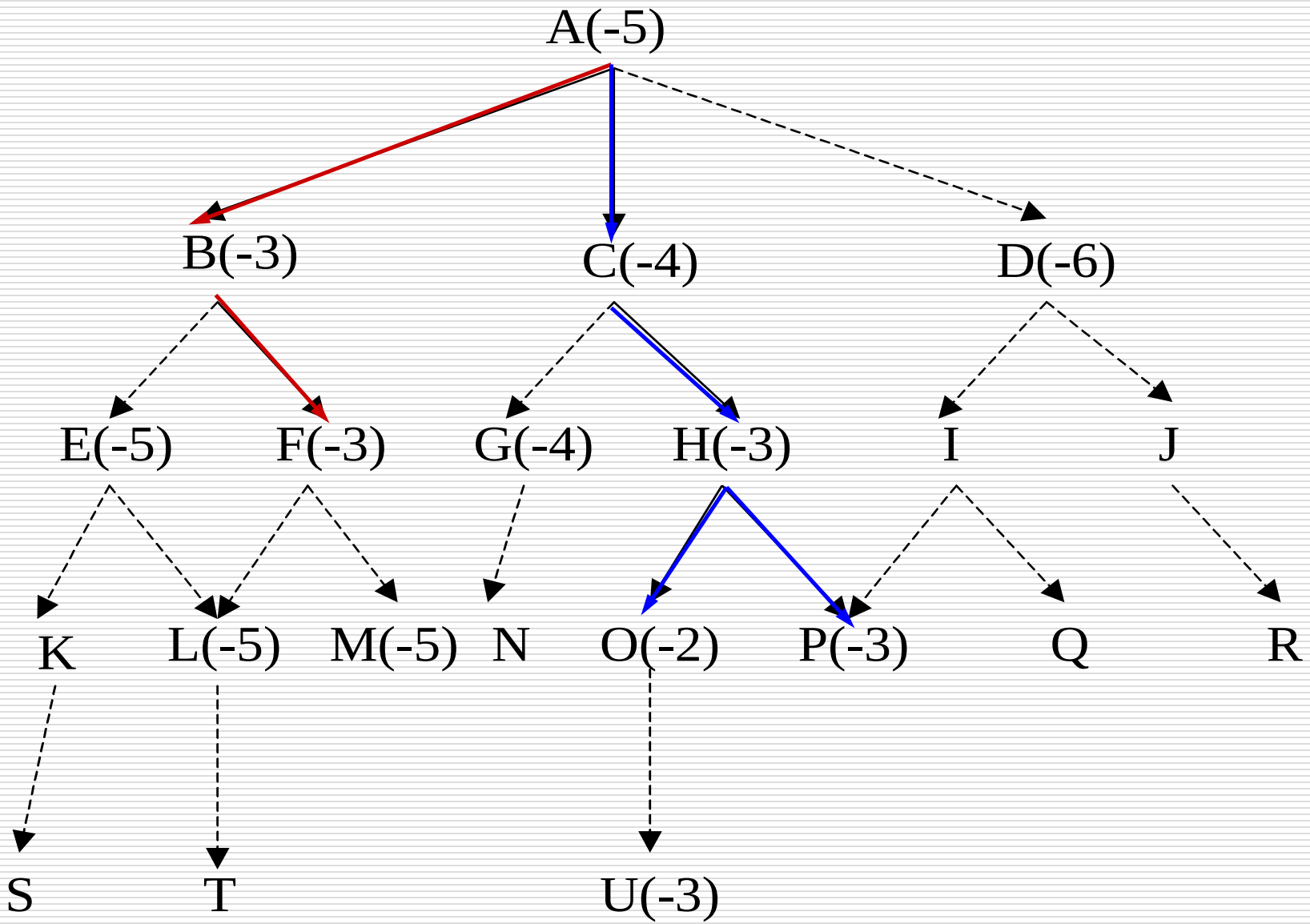
- **有序搜索的基本特点**：如何寻找并设计一个与问题有关的$h(n)$及构出$f(n)=g(n)+h(n)$，　然后以 $f(n)$的大小来排列待扩展状态的次序，每次选择$f(n)$ 值最小者进行扩展。

- open表：保留所有已生成而未扩展的状态。
- closed表：记录已扩展过的状态。
- 进入open表的状态是根据其估值的大小插入到表中合适的位置，每次从表中优先取出启发估价函数值最小的状态加以扩展。

5.4.2 有序搜索

- 由有序搜索算法的基本思想可见，如果节点i有多个父节点，则为节点i选择一个父节点，这个父节点使节点i具有最小的估计值f(i)。因此，尽管搜索的状态空间是一般的有向图，但是，由OPEN表和CLOSED表共同记载的搜索结果仍是一棵树。
- 与盲目搜索方法比较，有序搜索的目的在于减少被扩展的节点数，其有效性取决于启发式函数的选择。

5.4.2 有序搜索



5.4 状态空间启发式搜索

- **5.4.1 全局择优搜索**
- **5.4.2 有序搜索**
- **5.4.3 A*算法**

5.4.3 A*算法

- 在正式讨论A*算法之前，我们先介绍几个函数。

函数h*(n)： 从n到目标节点路径的最小代价；

函数g*(n)： 所有从S开始可达到n的路径的最小代价；

函数f*(n)： 从节点S到节点n的一条最佳路径的实际代价加上从节点n到某目标节点的一条最佳路径的代价之和，即 f*(n)=g*(n)+ h*(n)

- 因而f*(n)值就是从S开始约束通过节点n的一条最佳路径的代价，而f*(S)=h*(S)是一条从S到某个目标节点中间无约束的一条最佳路径的代价。

5.4.3 A*算法

- 我们希望估价函数f(n)是f*(n)的一个估计，此估计可由下式给出：

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

其中：g是g*的估计；h是h*的估计。对于g(n)来说。很显然$g(n)\geq g^*(n)$。对于h*(n)估计h(n)，它依赖于有关问题的领域的启发信息。

5.4.3 A*算法

- **定义1：** 在有序搜索过程中，如果重排OPEN表是依据f(x)=g(x)+h(x)进行的，则称该过程为A算法。

- **定义2：** 在A算法中，如果对所有的x，h(x)≤h*(x)成立,则称h(x)为h*(x)的下界，它表示某种偏于保守的估计。

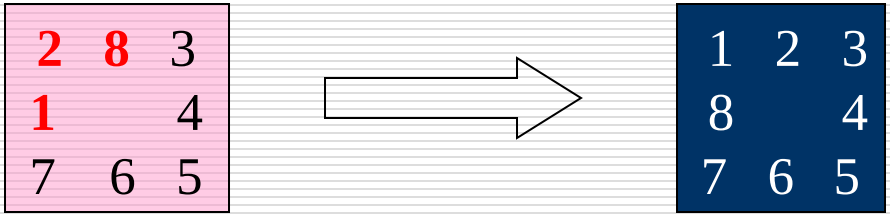
- **定义3：** 采用h*(x)的下界h(x)为启发函数的A算法，称为A*算法。

5.4.3 A*算法

- 如果某一问题有解，那么利用A*搜索算法对该问题进行搜索则一定能搜索到解，并且一定能搜索到最优的解而结束。
- **例子：**利用启发式搜索求解八数码问题的搜索树，其估价函数定义为

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

- $d(n)$ ：状态的深度，每步为单位代价。
- $w(n)$ ：“不在目标状态位置”的数字个数



初始状态S0　　目标状态Sg

5.4.3 A*算法

1. 可采纳性

对于可解状态空间图，如果一个搜索算法在优先步内终止，并能得到最优解，就称该搜索算法是可采纳的。

2. 单调性

搜索算法的单调性：在整个搜索空间都是局部可采纳的。一个状态和任一个子状态之间的差由该状态与其子状态之间的实际代价所限定。

$$h(n_i) - h(n_j) \leq cost(n_i, n_j); h(Goal) = 0$$

5.4.3 A*算法

3. 信息性

在两个A*启发策略的 h_1和h_2 中，如果对搜索空间中的任一状态 n 都有 $h_1(n) \le h_2(n)$ ，就称策略 h_2比h_1 具有更多的信息性。

第5章 搜索求解策略

- 5.1 搜索的概念
- 5.2 状态空间表示法
- 5.3 状态空间盲目搜索
- 5.4 状态空间启发式搜索
- ✓ 5.5 与或树表示
- 5.6 与或树盲目搜索
- 5.7 与或树启发式搜索
- 5.8 博弈树搜索

5.5 与或树表示

- **与/或树**是用于表示问题及其求解过程的又一种形式化方法，也称为**问题归约方法**。

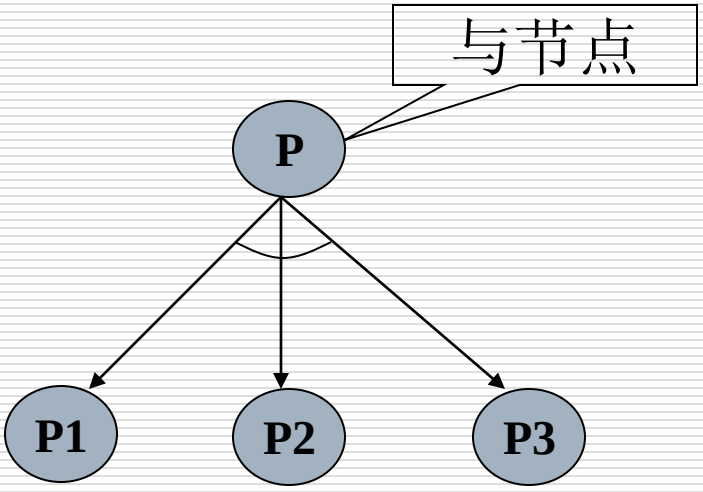
- **问题归约:**
  - 含义: 把复杂问题转换为若干需要同时处理的较为简单的子问题后再加以分别求解的策略,可以递归地进行,直到把问题转换为本原问题的集合.
  - 方法: **分解,等价变换**

5.5 与或树表示

1. 分解

- 如果一个问题P可以归约为一组子问题P1,P2,…,Pn，并且只有当所有子问题Pi都有解时原问题P才有解，任何一个子问题Pi无解都会导致原问题P无解，则称此种归约为问题的分解。
- **分解所得到的子问题的“与”同原问题P等价。**

5.5 与或树表示

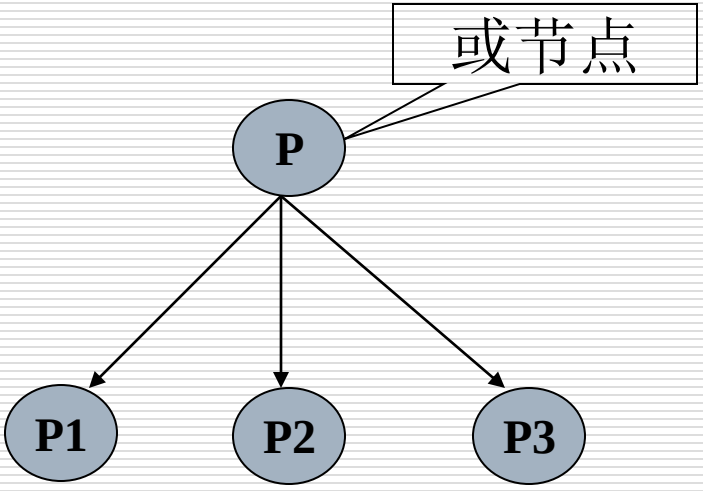


“与“树

2. 等价变换

- 如果一个问题P可以归约为一组子问题P1,P2,…,Pn，并且子问题Pi中只要有一个有解则原问题P就有解，只有当所有子问题Pi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。
- **变换所得到的子问题的“或”与原问题P等价。**

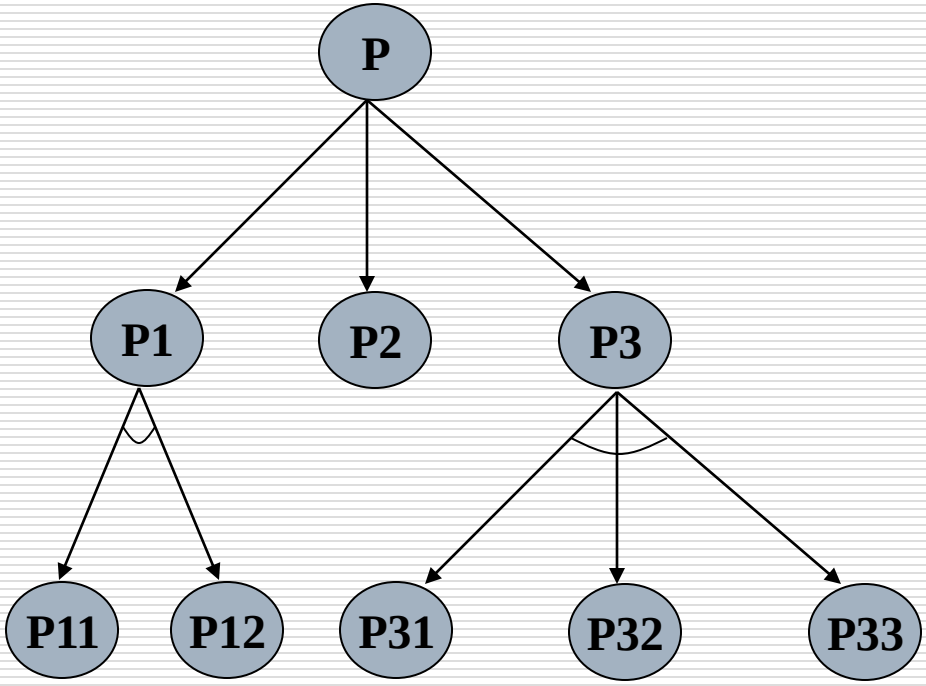
5.5 与或树表示



“或“树

5.5 与或树表示

- 上述两种方法也可结合起来使用，此时的图称为”与/或树”（AND/OR Tree）。
- **与或树中有几类节点？**



“与/或“树

3. 本原问题

- 不能再分解或变换，而且可以直接可解的子问题称为本原问题。

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，而端节点不一定是终止接点。

5. 可解节点

在与/或树中，满足下列条件之一者，称为可解节点：

1) 它是一个终止节点。

2) 它是一个“或”节点，且其子节点至少有一个是可解节点。

3) 它是一个“与”节点，且其子节点全部是可解节点。

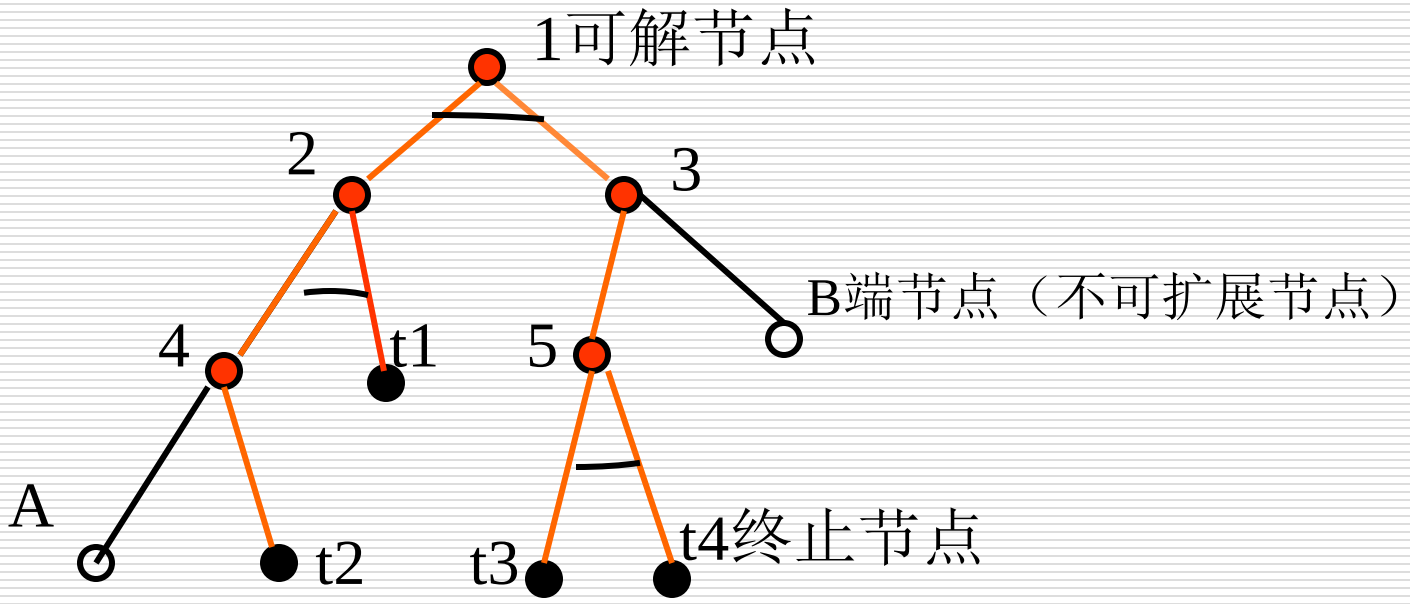
6. 不可解节点

关于可解节点的三个条件全部不满足的节点称为不可解节点。

7. 解树

由可解节点所构成的，并且由这些可解节点可推出初始节点为可解节点的子树称为解树。

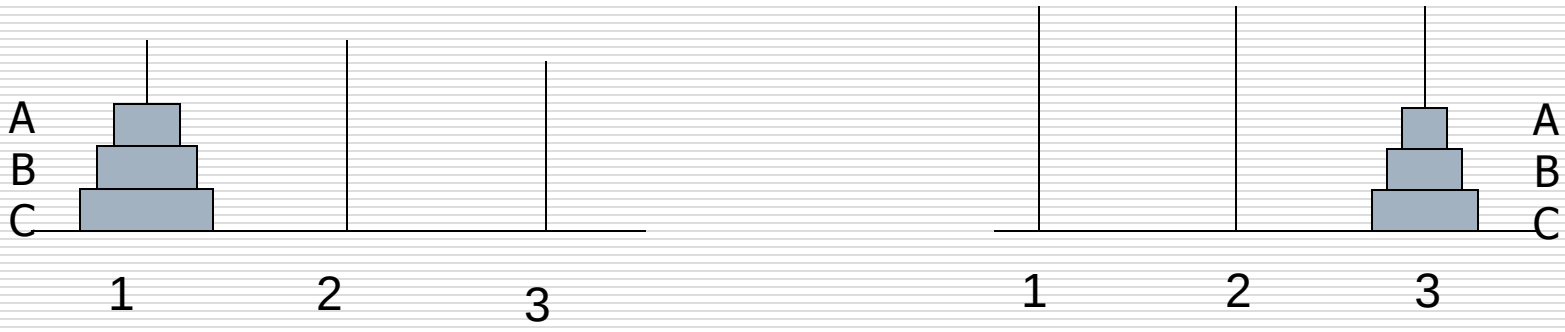
5.5 与或树表示



解 树

5.5 与或树表示

例：三阶梵塔问题。要求把1号柱子上的A,B,C三个盘片全部移到3号柱上。搬动规则为：(1)一次只能搬一个圆盘　(2)不能将大圆盘放在小圆盘上　(3)可以利用空柱子。



解：这个问题也可用状态空间法来解，本例主要用它来说明如何用与/或树法来解决问题。

5.5 与或树表示

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：

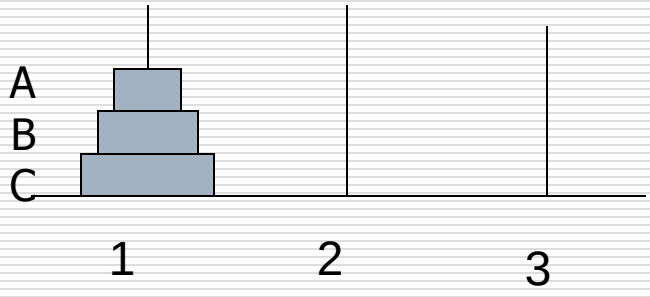
i 代表片**C**所在的柱号

j 代表片**B**所在的柱号

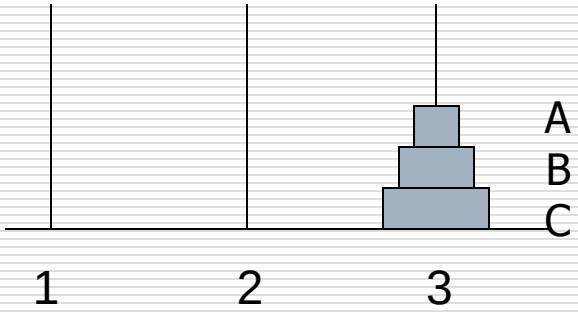
k 代表片**A**所在的柱号

5.5 与或树表示

初始状态(1,1,1)



目标状态 (3,3,3)



利用问题归约方法，原问题可分解为以下三个子问题：

(1) 把A及B移到2号柱子上的双圆盘移动问题。即(1, 1, 1)⟶(1, 2, 2)

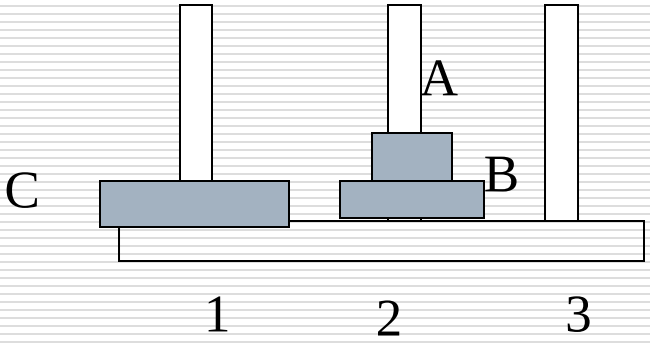
(2) 把C移到3号柱子上的单圆盘移动问题。即 (1, 2, 2)⟶(3, 2, 2)

(3) 把A及B移到3号柱子的双圆盘移动问题。即(3, 2, 2)⟶(3, 3, 3)

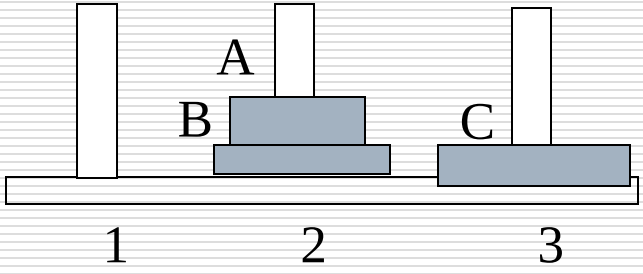
其中，子问题(1)和(3)都是一个二阶梵塔问题，它们都还可以再继续进行分解；子问题(2)是本原问题，它已不需要再分解。

5.5 与或树表示

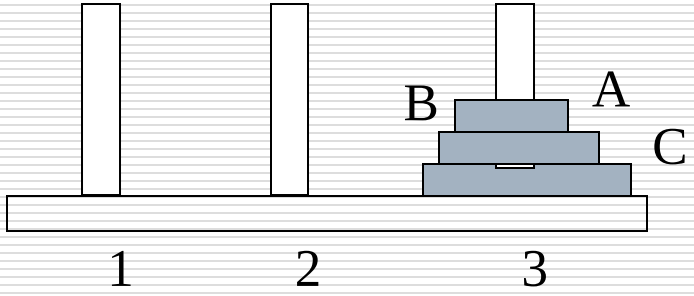
- 分解的关键状态



(1,2,2)



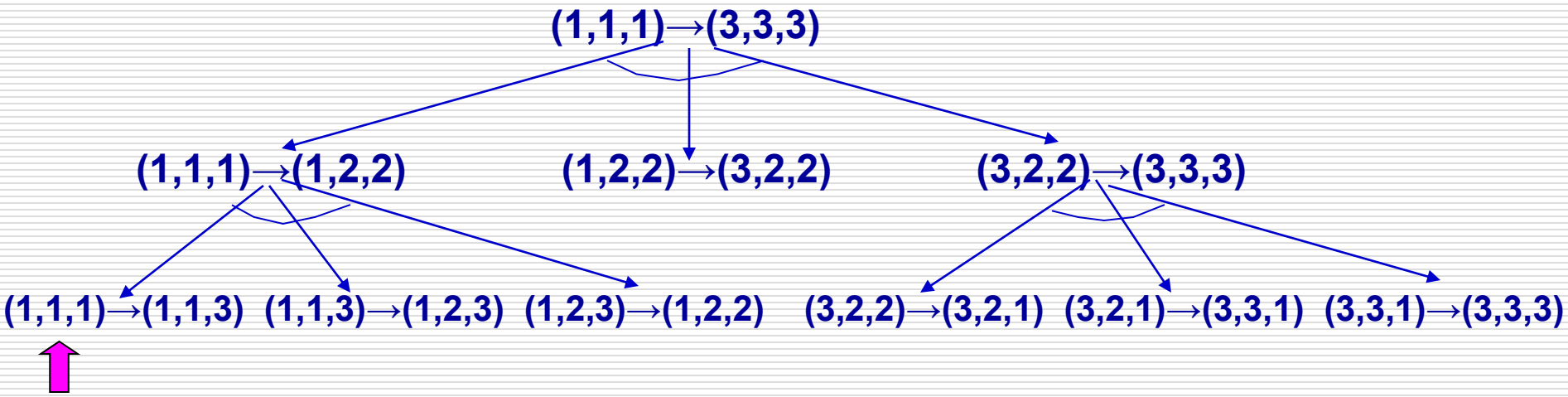
(3,2,2)



(3,3,3)

5.5 与或树表示

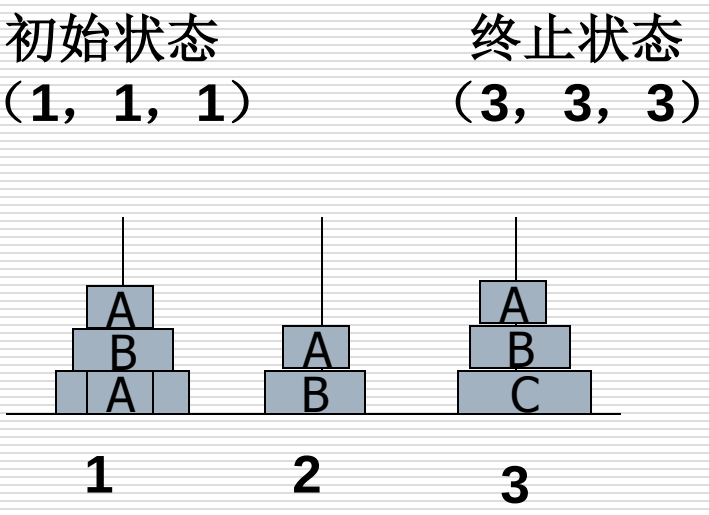
三阶梵塔问题的分解过程可用如下图与/或树来表示



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

5.5 与或树表示

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



第5章 搜索求解策略

- 5.1 搜索的概念
- 5.2 状态空间表示法
- 5.3 状态空间盲目搜索
- 5.4 状态空间启发式搜索
- 5.5 与或树表示
- ✓ 5.6 与或树盲目搜索
- 5.7 与或树启发式搜索
- 5.8 博弈树搜索

5.6 与或树盲目搜索

- **与或树搜索基本思想：**
  - **扩展（自上而下）**
  - **标示（自下而上）**
  - **结束条件：初始节点为可解或不可解**
- **搜索过程：**

（1）把原始问题作为初始节点S0, 并将其作为当前节点。

（2）应用分解或等价变换算符对当前节点进行扩展。

（3）为每个子节点设置指向父节点的指针

（4）选择合适的子节点作为当前节点,反复应用（2）（3）步，在此期间要多次应用**可解标示过程**和**不可解标示过程**，直到初始节点被标示为可解节点或不可解节点。

上述搜索过程将形成一颗与/或树，这种由搜索过程所形成的与/或树称为搜索树。

5.5 与或树盲目搜索

- **可解与不可解标示过程**
- **可解标示过程：**由可解子节点来确定父节点，祖父节点等为可解节点的过程
  - **‘与’节点只有当其子节点全为可解节点时，才为可解节点；**
  - **‘或’节点只要有一个子节点为可解节点，它就是可解节点。**
- **不可解标示过程：**由不可解子节点来确定父节点，祖父节点等为不可解节点的过程
  - **‘与’节点只要其子节点有一个为不可解节点，它就是不可解节点；**
  - **‘或’节点只有当其子节点都为不可解节点，它才是不可解节点。**

5.6 与或树盲目搜索

- 与或树宽度优先搜索
- 与或树深度优先搜索

5.6 与或树盲目搜索-宽度优先

按照“先产生的节点先扩展”的原则进行搜索。

与/或树的宽度优先搜索与状态空间的宽度优先搜索的主要差别是，在搜索过程中需要多次调用可解标示过程或不可解标示过程.

搜索过程要用两个表

OPEN表：用于存放刚生成的节点，对于不同的搜索策略，节点在OPEN表中的排列顺序是不同的。

CLOSED表：用于存放将要扩展或已扩展的节点。

5.6 与或树盲目搜索-宽度优先

- 与/或树的宽度优先搜索与状态空间的宽度优先搜索类似，是按“**先产生的节点先扩展**”的原则进行搜索，只是在搜索过程中要多次调用可解标示过程和不可解标示过程。搜索过程如下：

(1) 把初始节点S0放入OPEN表。

(2) 把OPEN表中的第一个节点(记为节点n)取出放入CLOSED表。

(3) 如果节点n可扩展，则：

① 扩展节点n，将其子节点放入OPEN表的**尾部**，并为每个子节点配置指向父节点的指针。

② 考察这些子节点中是否有终止节点。若有，则标示这些终止节点为可解节点，并应用可解标示过程对其先辈节点中的可解节点进行标示。如果初始节点S0也被标示为可解节点，就得到了解树，搜索成功，退出搜索过程；如果不能确定S0为可解节点，则从OPEN表中删去具有可解先辈的节点。

③ 转第(2)步。

5.6 与或树盲目搜索-宽度优先

(4) 如果节点n不可扩展，则：

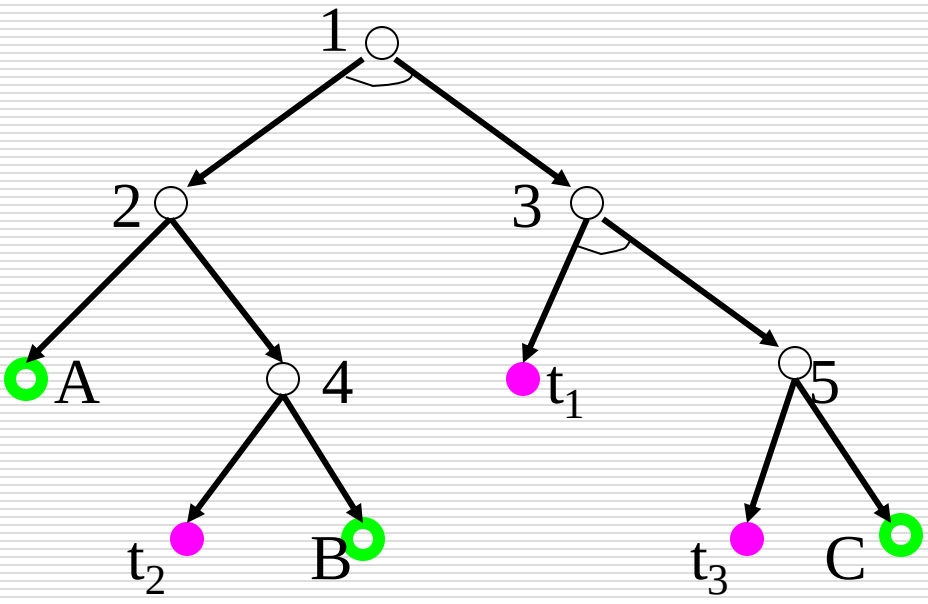
① 标示节点n为不可解节点。

② 应用不可解标示过程对节点n的先辈节点中不可解的节点进行标示。如果初始节点S0也被标示为不可解节点，则搜索失败，表明原始问题无解，退出搜索过程；如果不能确定S0为不可解节点，则从OPEN表中删去具有不可解先辈的节点

③ 转第(2)步。

5.6 与或树盲目搜索-宽度优先

例： 设有下图所示的与/或树，节点按标注顺序进行扩展，其中标记为t_1、t_2、t_3的节点是终止节点，节点A、B、C为不可解的端节点。



与/或树的宽度优先搜索

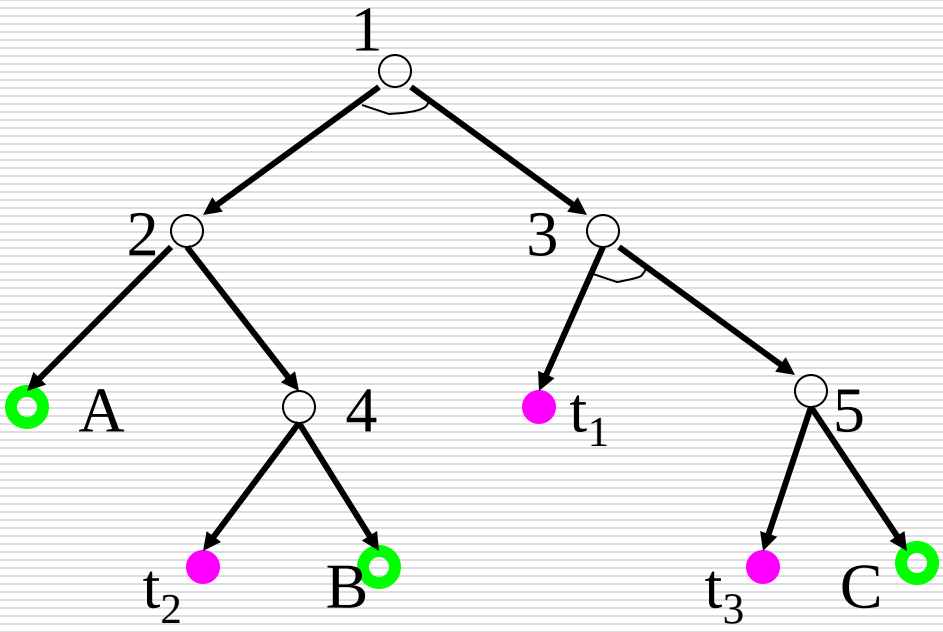
5.6 与或树盲目搜索-深度优先

与/或树的深度优先搜索和与/或树的宽度优先搜索过程基本相同，其主要区别在于OPEN表中节点的排列顺序不同。在扩展节点时，与/或树的深度优先搜索过程总是把刚生成的节点放在OPEN表的首部。

与/或树的深度优先搜索也可以带有深度限制dm。

5.6 与或树盲目搜索-深度优先

对上例，若按有界深度优先，且设dm=3，则其节点扩展顺序为：1，2，4，3，5。



与/或树的有界深度优先搜索

小结

- 与或树搜索基本思想：
  - 扩展（自上而下）
  - 标示（自下而上）
  - 结束条件：初始节点为可解或不可解
- 与或树盲目搜索
  - 宽度优先
  - 深度优先

第5章 搜索求解策略

- 5.1 搜索的概念
- 5.2 状态空间表示法
- 5.3 状态空间盲目搜索
- 5.4 状态空间启发式搜索
- 5.5 与或树表示
- 5.6 与或树盲目搜索
- ✓ 5.7 与或树启发式搜索
- 5.8 博弈树搜索

5.7 与或树启发式搜索

- 与/或树的启发式搜索可用来求取代价最小的解树，也被称之为与或树有序搜索。
- **1、解树的代价**

为了进行有序搜索，需要计算解树的代价。解树的代价可以通过计算解树中节点的代价得到。

设c(x,y)表示节点x到其子节点y的代价，计算节点x代价的方法如下：

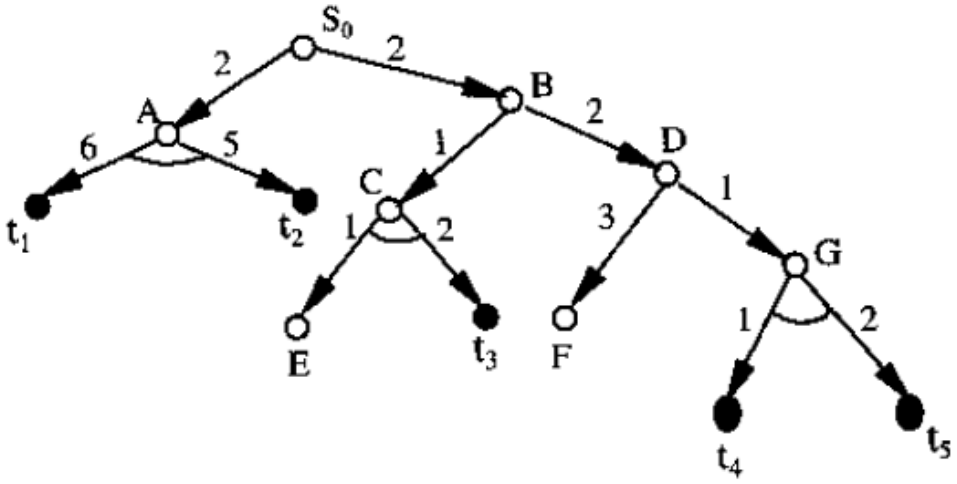
5.7 与或树启发式搜索

- 1) 如果x是终止节点，则定义节点x的代价h(x) = 0；
- 2) 如果x是”或”节点，y1,y2…yn是它的子节点，则节点x的代价为：h(x) = min{c(x,yi) + h(yi)}
- 3)如果x是”与”节点，则节点x的代价有两种计算方法：和代价法与最大代价法。
- 若按和代价法计算，则有：h(x) = ∑(c(x,yi) + h(yi)) 若按最大代价法计算，则有：h(x) = max{c(x,yi) + h(yi)}
- 4) 如果x不可扩展，且又不是终止节点，则定义h(x) = ∝

由上述计算节点的代价可以看出，如果问题是可解的，则由子节点的代价就可推算出父节点代价，这样逐层上推，最终就可求出初始节点S0的代价，S0的代价就是解树的代价。

5.7 与或树启发式搜索

例1 下图是一棵与/或树，其中包括两棵解树，一棵解树由S0,A t1和t2组成；另一棵解树由S0,B,D,G, t4和t5组成。在此与或树中， t1 、t2 、t3 、t4 、t5为终止节点；E、F是端节点，其代价为 ∞；边上的数字是该边的代价。



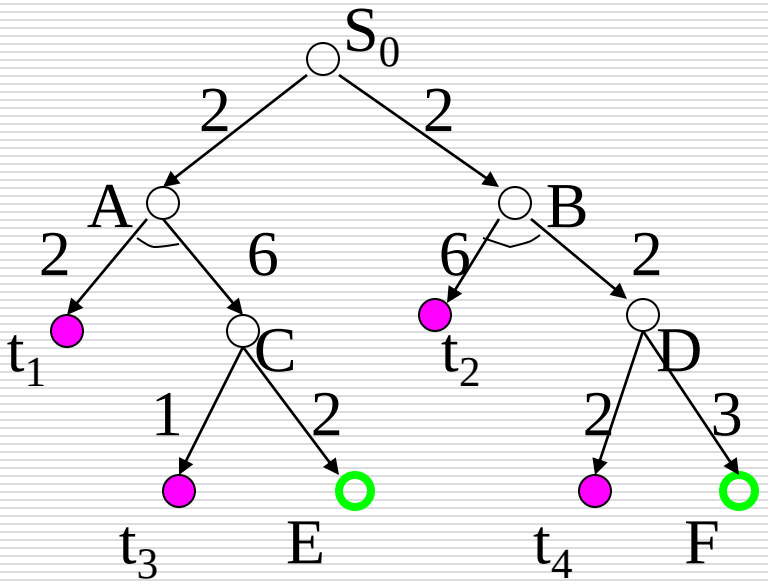
由左边的解树可得：　　由右边的解树可得：

按和代价：h(A)=11,h(S0)=13　　h(G)=3,h(D)=4,h(B)=6,h(S0)=8

按最大代价：h(A)=6,h(S0)=8　　h(G)=2,h(D)=3,h(B)=5,h(S0)=7

显然，若按和代价计算，右解树是最优解树，若按最大代价计算，右解树仍然是最优解树。

例2 设下图是一棵与/或树，它包括两棵解树，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。在此与或树中，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。



与/或树的代价

解：先计算左边的解树

按和代价：$h(S_0)=(2+6+1)+2=11$

按最大代价：$h(S_0)=(1+6)+2=9$

再计算右边的解树

按和代价：$h(S_0)=(6+2+2)+2=12$

按最大代价：$h(S_0)=6+2=8$

显然，若按和代价计算，左解树是最优解树，
若按最大代价计算，右解树是最优解树。

5.7 与或树启发式搜索

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：
- (1) 初始节点S0在希望树T
- (2) 如果n是具有子节点n1, n2, … , nk的或节点，则n的某个子节点ni在希望树T中的充分必要条件是

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

(3) 如果n是与节点，则n的全部子节点都在希望树T中。

5.7 与或树启发式搜索

3、与/或树的启发式搜索过程

(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，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。

(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 如果初始解节点S_0能够被标记为不可解节点，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。

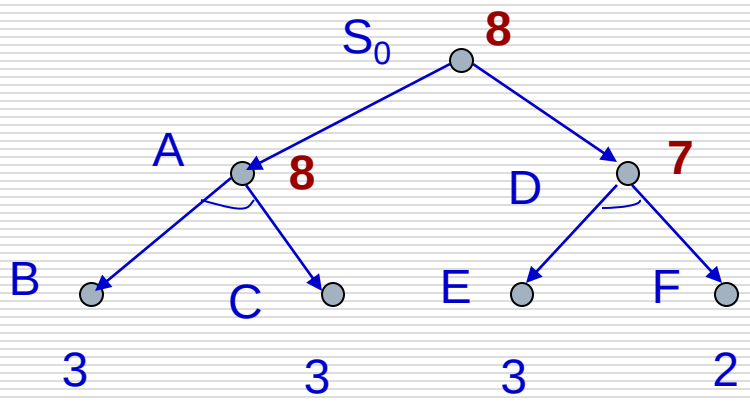
⑤ 转第(2)步。

例子3

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。

按和代价法计算节点S_0、A、D的节点代价？



扩展S_0后得到的与/或树

此时，S0的右子树是当前的希望树：

h(A)=3+1+3+1=8

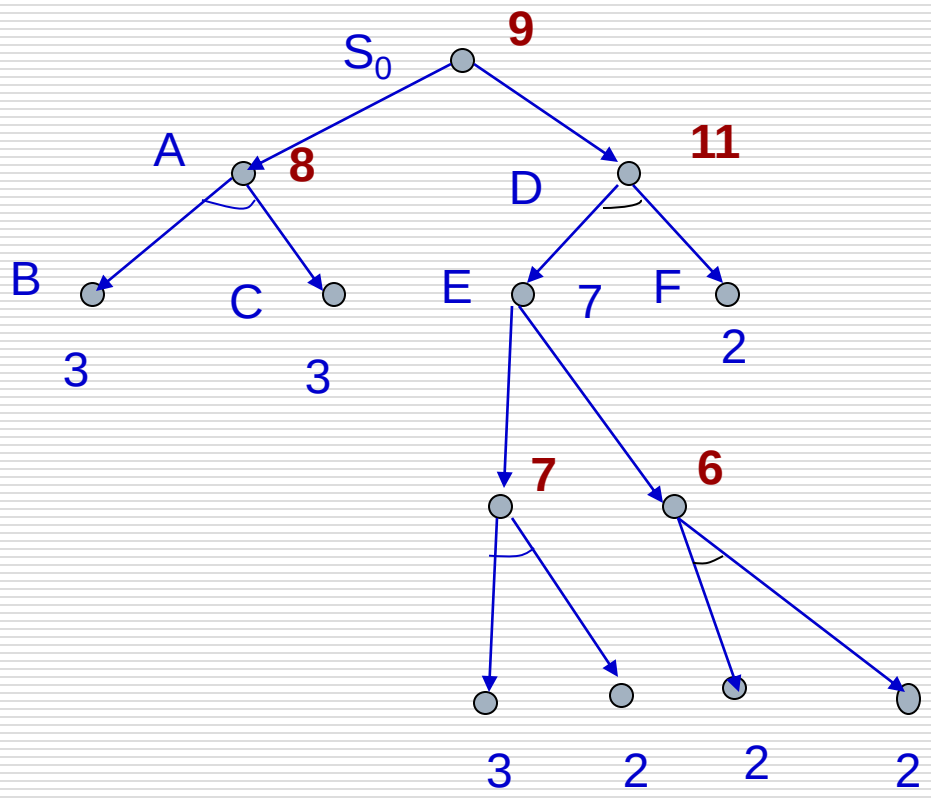
h(D)=3+1+2+1=7

h(S0)=7+1=8

5.7 与或树启发式搜索

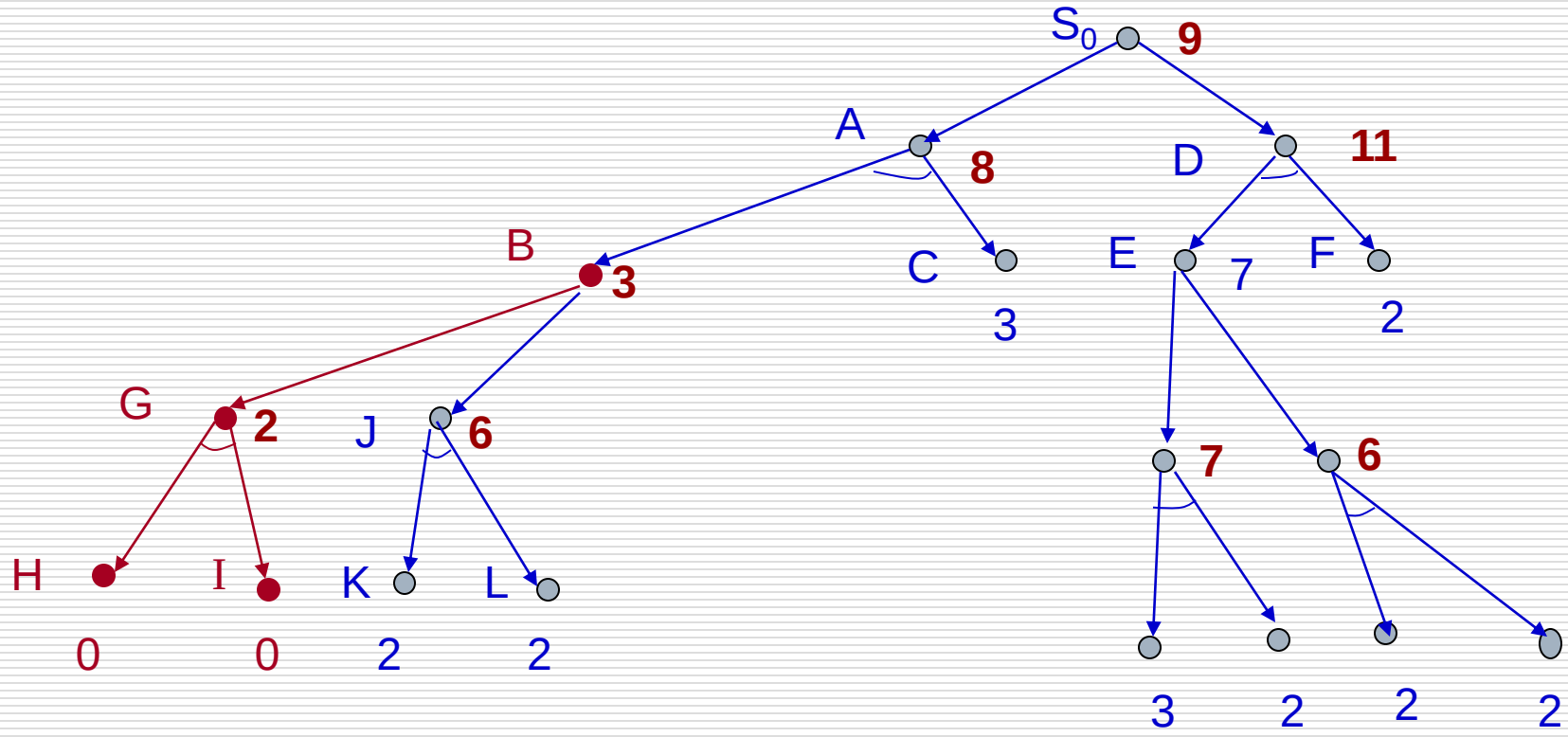
扩展节点**E**，得到如右图所示的与/或树。

此时，由右子树求出的$h(S_0)=12$。
由左子树求出的$h(S_0)=9$。
左子树的代价小为希望树。



5.7 与或树启发式搜索

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。



第5章 搜索求解策略

- 5.1 搜索的概念
- 5.2 状态空间表示法
- 5.3 状态空间盲目搜索
- 5.4 状态空间启发式搜索
- 5.5 与或树表示
- 5.6 与或树盲目搜索
- 5.7 与或树启发式搜索
- ✓ 5.8 博弈树搜索

5.8 博弈树搜索

- **5.8.1 博弈树概念**
- **5.8.2 极大极小分析法**
- **5.8.3 α－β剪枝技术**

5.8.1 博弈树搜索

博弈问题特点

- 全信息:对垒的双方A，B轮流采取行动，任何一方都了解当前的格局及过去的历史。
- 二人零和:博弈的结果只有三种情况：A方胜，B方败；B方胜，A方败；双方战成平局。
- 非偶然:博弈的过程是寻找置对手于必败状态的过程。

5.8.1 博弈树搜索

博弈的过程

在博弈过程中，任何一方都希望自己取得胜利。因此，当某一方当前有多个行动方案可供选择时，他总是挑选对自己最为有利而对对方最为不利的那个行动方案。这样，如果站在某一方（如正方，即在MAX要取胜的意义下），把上述博弈过程用图表示出来，则得到的是一棵“与或树”。

5.8.1 博弈树搜索

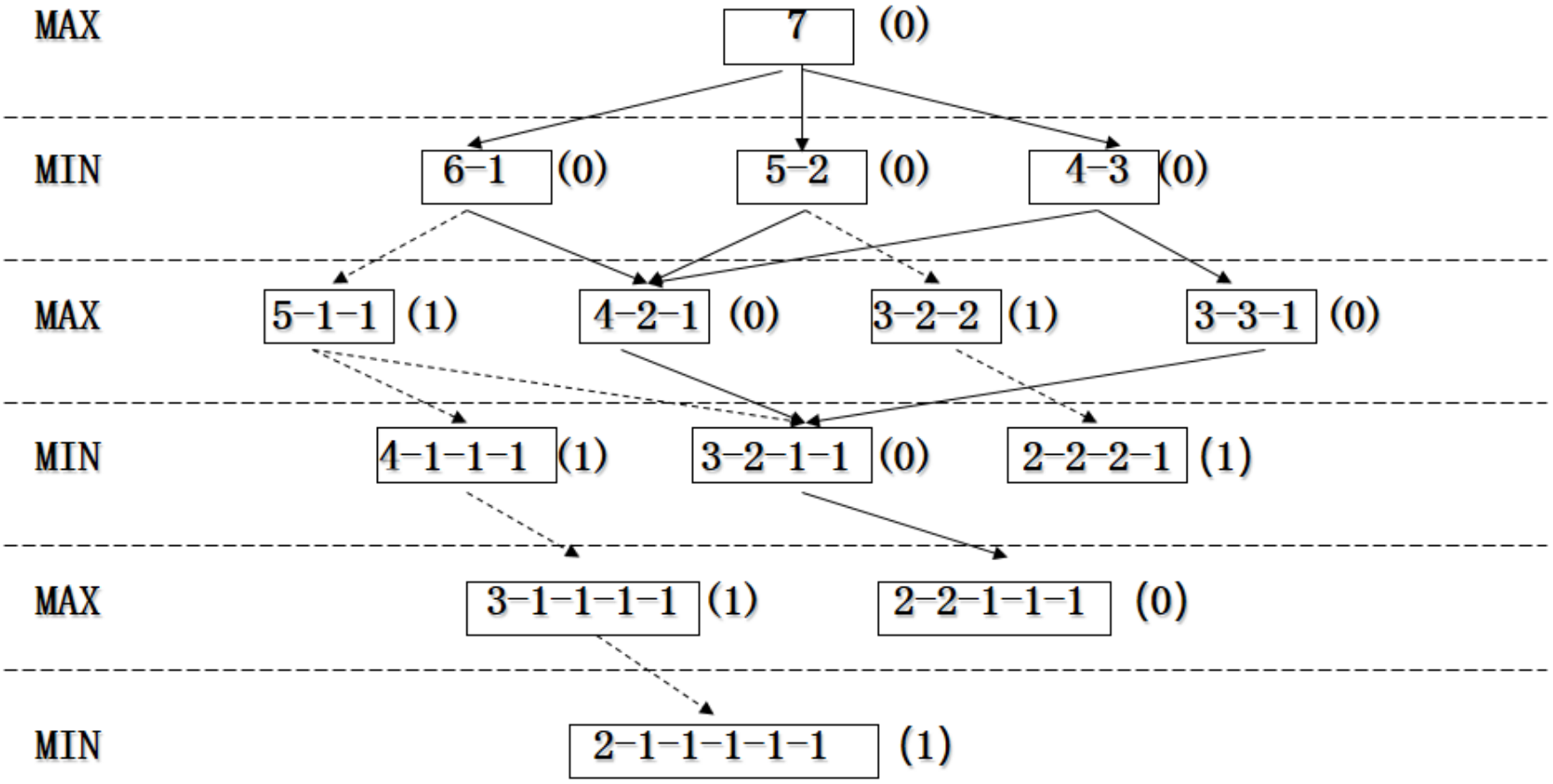
Grundy博弈

Grundy博弈是一个分钱币的游戏。有一堆数目为N的钱币，由两位选手轮流进行分堆，要求每个选手每次只把其中某一堆分成数目不等的两小堆。例如， 选手甲把N分成两堆后，轮到选手乙就 可以挑其中一堆来分，如此进行下去， 直到有一位选手先无法把钱币再分成不相等的两堆时就得认输。

命名博弈的双方，一方为“正方”，这类节点称为“MAX”节点；另一方为“反方”，这类节点称为“MIN”节点，正方和反方是交替走步的，因此MAX节点和MIN节点会交替出现。

5.8.1 博弈树搜索

- Grundy问题博弈树　　N=7



5.8.1 博弈树搜索

■ 博弈树的特点

■博弈的初始格局是初始节点。

■在博弈树中，“MAX”节点和“MIN”节点是逐层交替出现的。自己一方扩展的节点之间是“或”关系，对方扩展的节点之间是“与”关系。双方轮流地扩展节点。

■所有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。

5.8 博弈树搜索

- **5.8.1 博弈树概念**
- **5.8.2 极大极小分析法**
- **5.8.3 α－β剪枝技术**

5.8.2 极大极小分析法

- 极大极小分析法的基本思想
  - 设博弈的双方中一方为A，另一方为B。然后为其中的一方(例如A)寻找一个最优行动方案。
  - 为了找到当前的最优行动方案，需要对各个可能的方案所产生的后果（得分）进行比较。
  - 为计算得分，需要根据问题的特性信息定义一个估价函数，用来估算当前博弈树端节点的得分。

5.8.2 极大极小分析法

- 当端节点的估值计算出来后，再推算出父节点的得分。

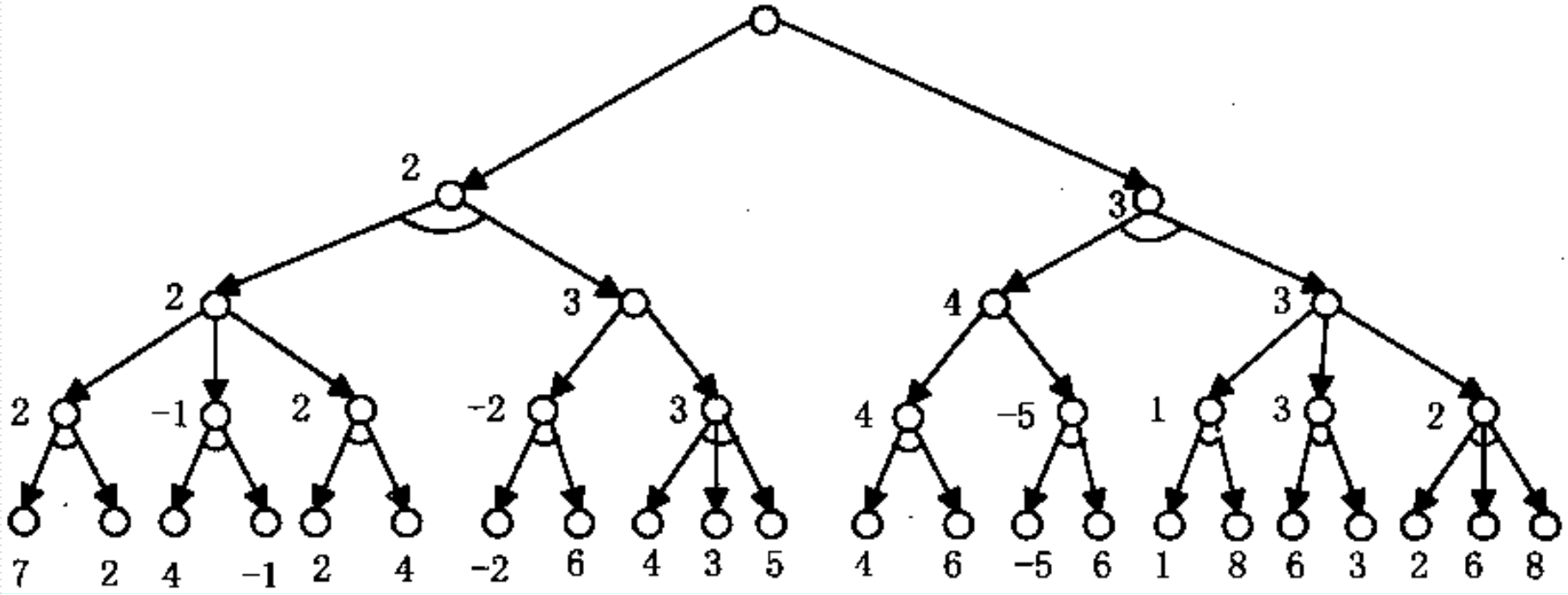
推算的方法是：

对“或”节点，选其子节点中一个最大的得分作为父节点的得分，这是为了使自己在可供选择的方案中选一个对自己最有利的方案；

对“与”节点，选其子节点中一个最小的得分作为父节点的得分，这是为了立足于最坏的情况。

- 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。

5.8.2 极大极小分析法



倒推值的计算

5.8.2 极大极小分析法

- 例1 一字棋游戏。设有如图4–19(a)所示的九个空格，由A，B二人对弈，轮到谁走棋谁就往空格上放一只自己的棋子，谁先使自己的棋子构成“三子成一线”谁就取得了胜利。

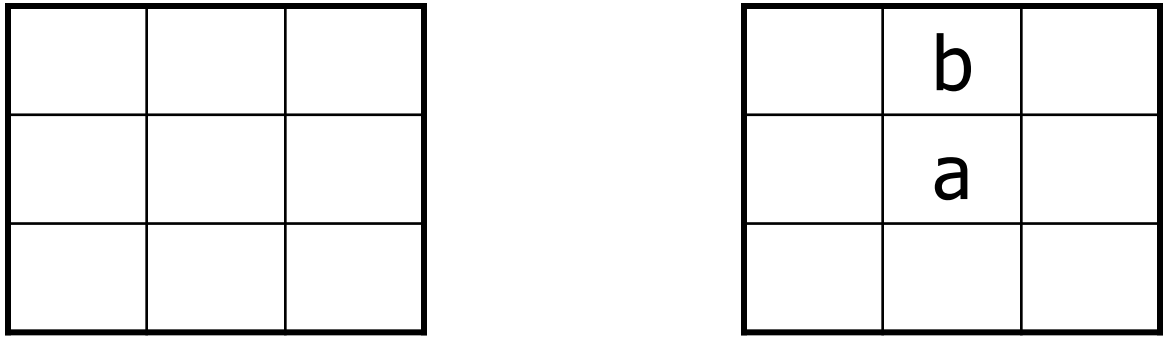


图1 一字棋

5.8.2 极大极小分析法

设A的棋子用“a”表示，B的棋子用“b”表示。为了不致于生成太大的博弈树，假设每次仅扩展两层。估价函数定义如下：

设棋局为P，估价函数为e(P)。

(1)若P是A必胜的棋局，则e(P)=+∞。

(2)若P是B必胜的棋局，则e(P)=-∞。

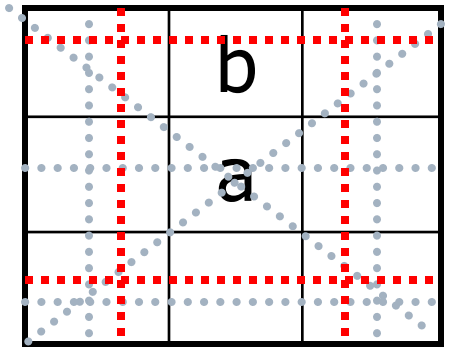
(3)若P是胜负未定的棋局，则e(P)=e(+P)-e(-P)

5.8.2 极大极小分析法

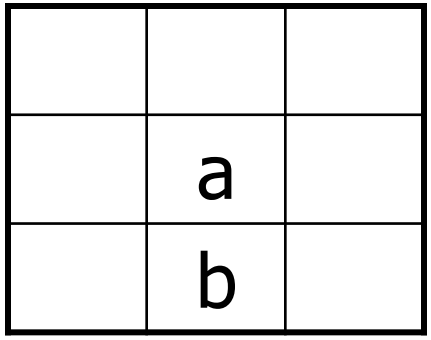
e(+p),e(-P)，其定义如下：

e(+P)：棋局P上有可能使a成为三子成一线的数目；

e(-P)：棋局P上有可能使b成为三子成一线的数目。



（a）



（b）

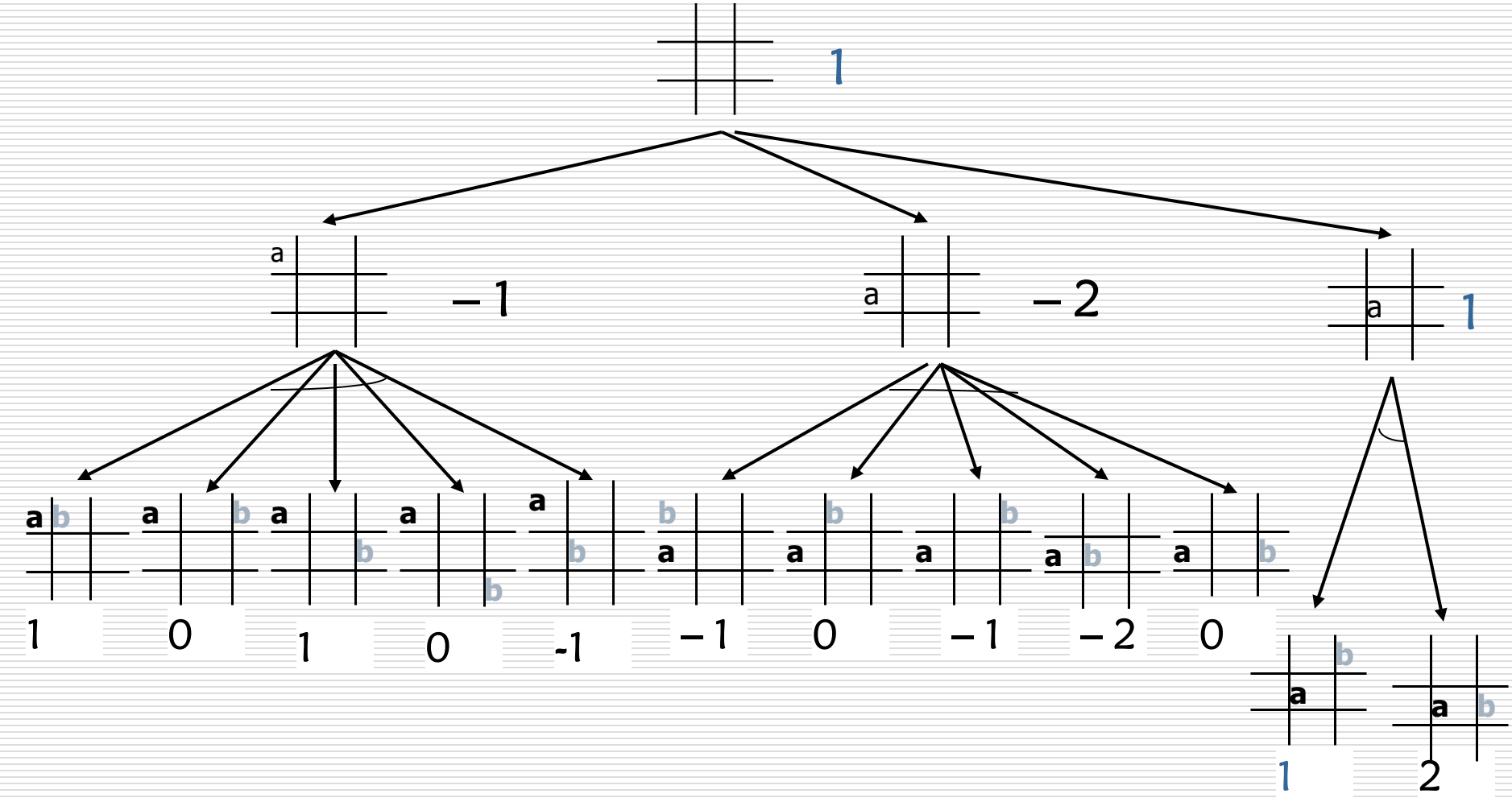
上图（a）所示棋局，**e(P)=6-4=2**

上图（b）所示棋局，**e(P)=6-4=2**

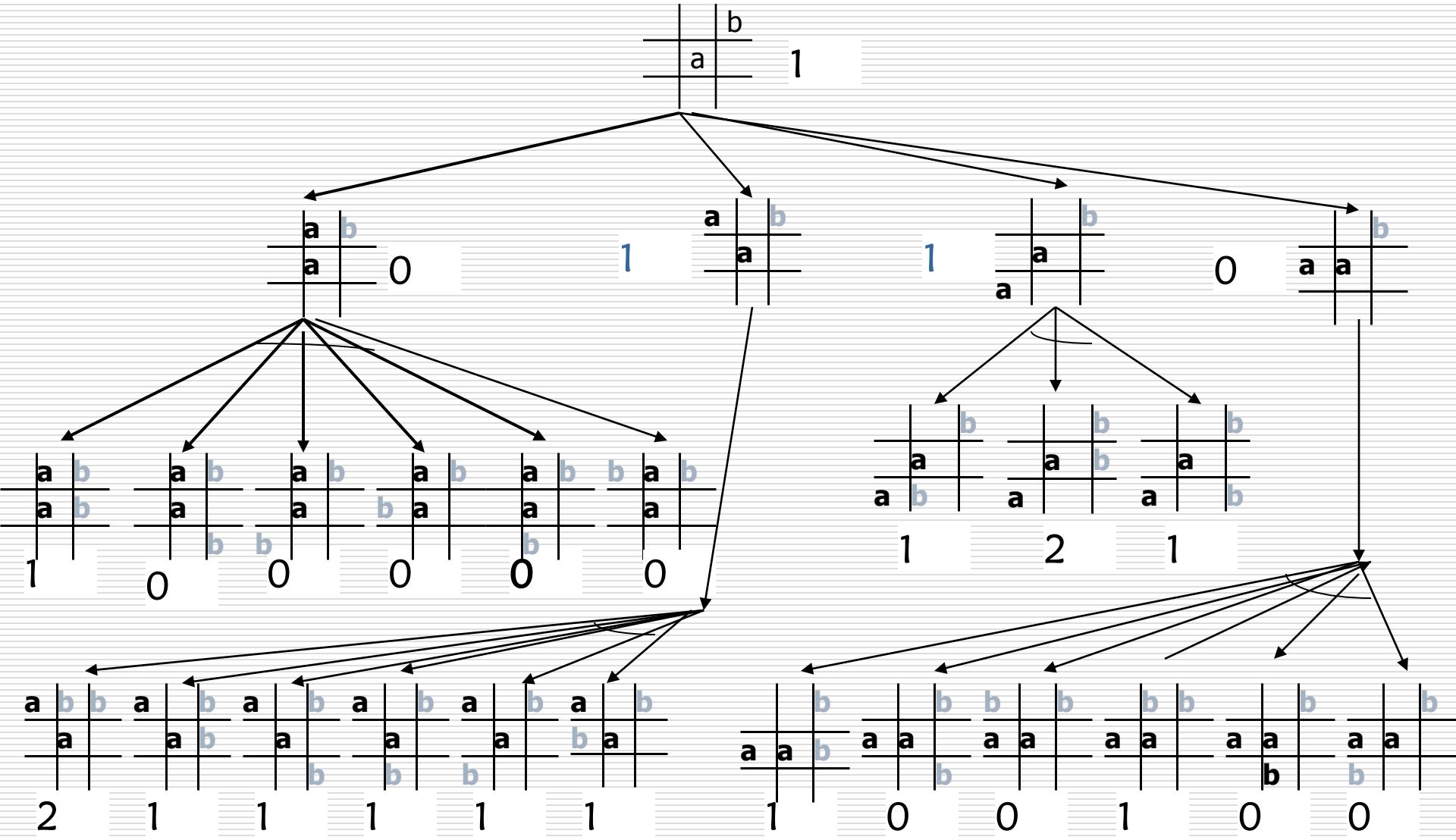
（a）和（b）具有对称性，可以作为一个棋局。

5.8.2 极大极小分析法

用这样的估计函数可得第一步的极大极小分析图，
其中扩展深度为2



5.8.2 极大极小分析法



5.8 博弈树搜索

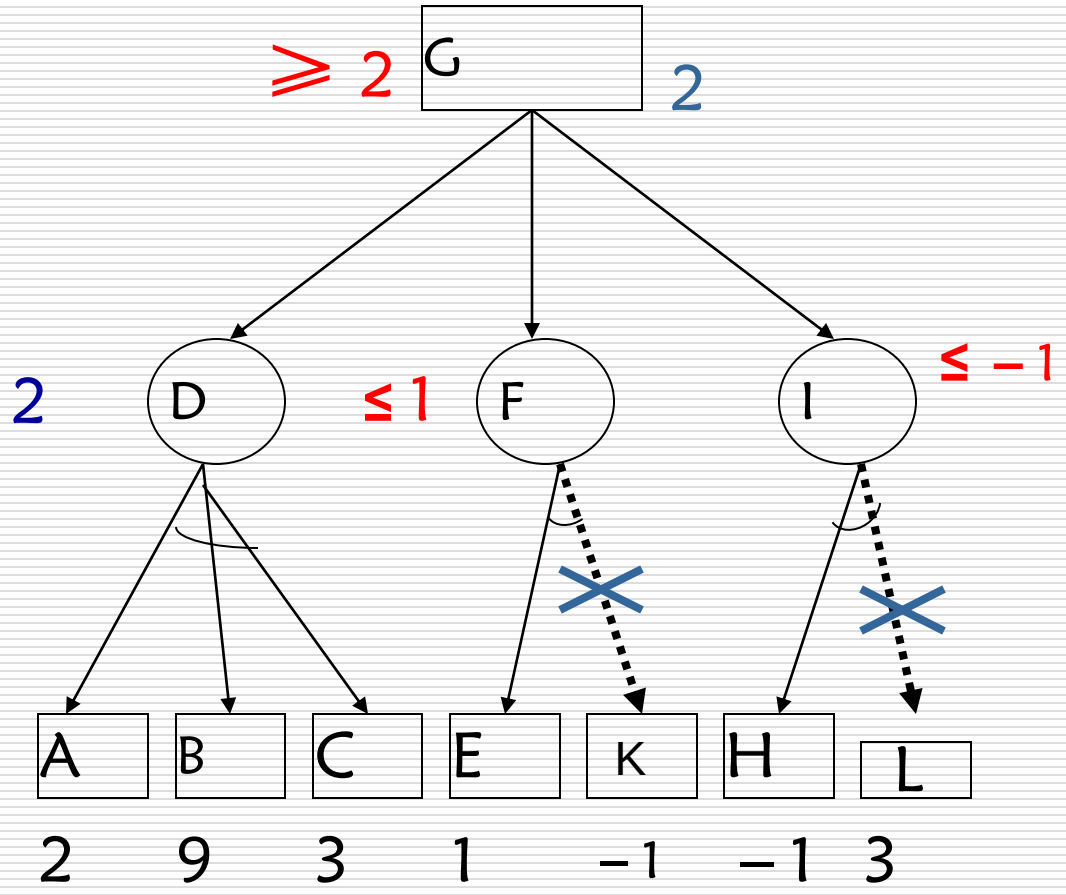
- **5.8.1 博弈树概念**
- **5.8.2 极大极小分析法**
- **5.8.3 α－β剪枝技术**

5.8.3 α－ß剪枝技术

- 极大极小分析法的缺点是效率低。
- α －ß剪枝技术的基本思想为：
  - 对于一个与节点MIN，若能估计出其倒推值的上确界β，并且这个β值不大于MIN的父节点（一定是或节点）的估计倒推值的下确界α，即α≥β，则就不必再扩展该MIN节点的其余子节点了（因为这些节点的估值对MIN父节点的倒推值已无任何影响了）。这一过程称为α剪枝。

5.8.3 α— β剪枝技术

α剪枝：

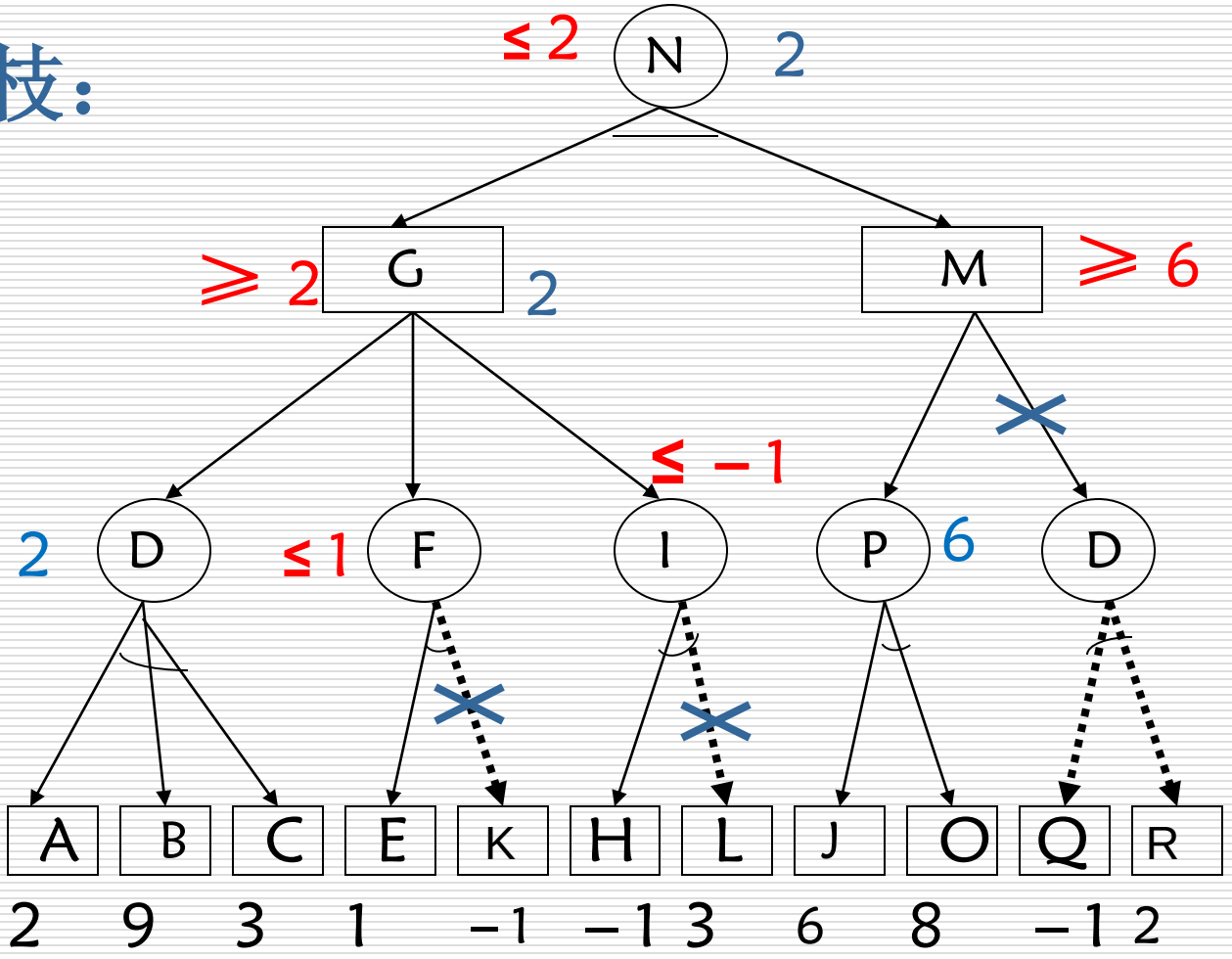


5.8.3 α－β剪枝技术

■对于一个或节点MAX，若能估计出其倒推值的下确界α，并且这个α值不小于MAX的父节点（一定是与节点）的估计倒推值的上确界β，即α≥β，则就不必再扩展该MAX节点的其余子节点了（因为这些节点的估值对MAX父节点的倒推值已无任何影响了）。这一过程称为β剪枝。

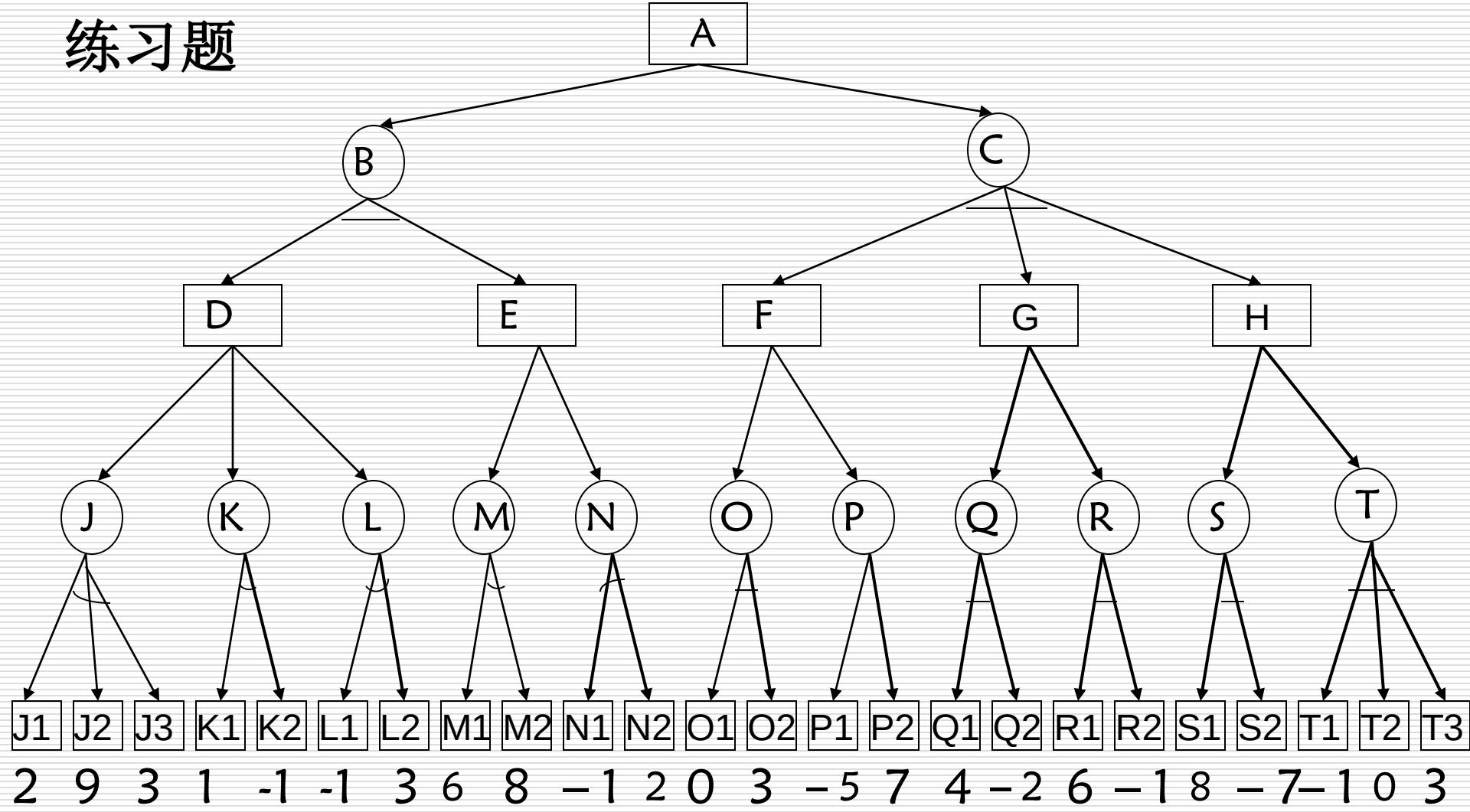
5.8.3 α－β剪枝技术

β剪枝：



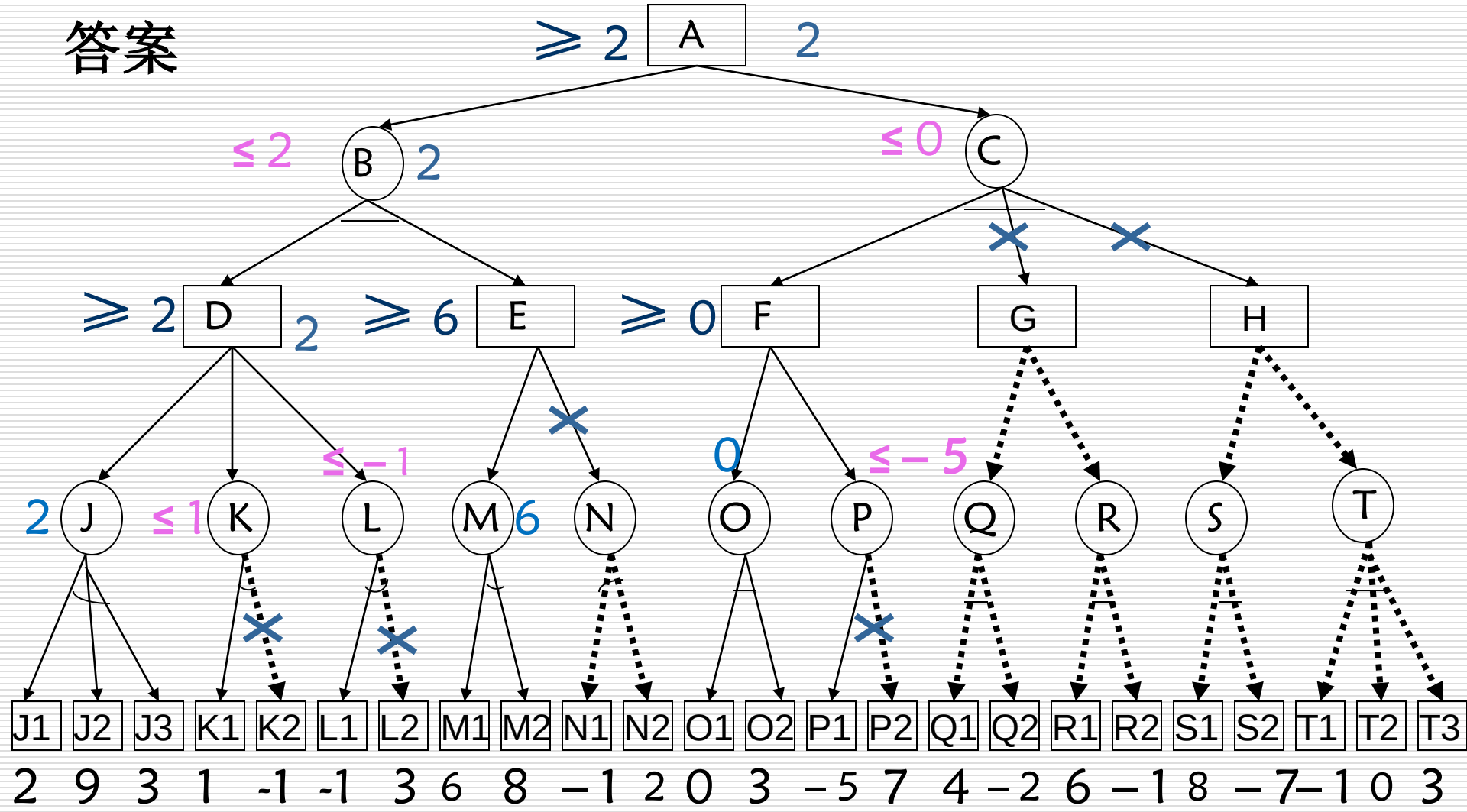
5.8.3 α— β剪枝技术

练习题



5.8.3 α－β剪枝技术

答案



THE END