第 5 章 搜索求解策略

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，然后着重介绍状态空间知识表示和搜索策略，主要有回溯策略、宽度优先搜索、深度优先搜索等盲目的图搜索策略，以及 **A** 及 **A*** 搜索算法等启发式图搜索策略。

第 5 章 搜索求解策略

- **5.1** 搜索的概念
- **5.2** 状态空间的搜索策略
- **5.3** 盲目的图搜索策略
- **5.4** 启发式图搜索策略

第 5 章 搜索求解策略

- ✓ **5.1** 搜索的概念
- **5.2** 状态空间的搜索策略
- **5.3** 盲目的图搜索策略
- **5.4** 启发式图搜索策略

5.1 搜索的概念

- 问题求解:
  - 问题的表示。
  - 求解方法。
- 问题求解的基本方法: 搜索法、归约法、归结法、推理法及产生式等。

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。

5.1.1 搜索的基本问题与主要过程

- 搜索的主要过程:

(1) 从初始或目的状态出发，并将它作为当前状态。

(2) 扫描操作算子集，将适用当前状态的一些操作算子作用于当前状态而得到新的状态，并建立指向其父结点的指针 。

(3) 检查所生成的新状态是否满足结束状态，如果满足，则得到问题的一个解，并可沿着有关指针从结束状态反向到达开始状态，给出一解答路径；否则，将新状态作为当前状态，返回第 (2) 步再进行搜索。

5.1.2 搜索策略

1. 搜索方向：

(1) 数据驱动：从初始状态出发的正向搜索。

正向搜索——从问题给出的条件出发。

(2) 目的驱动：从目的状态出发的逆向搜索。

逆向搜索：从想达到的目的入手，看哪些操作算子能产生该目的以及应用这些操作算子产生目的时需要哪些条件。

(3) 双向搜索

双向搜索：从开始状态出发作正向搜索，同时又从目的状态出发作逆向搜索，直到两条路径在中间的某处汇合为止。

5.1.2 搜索策略

- **2.** 盲目搜索与启发式搜索：

（1）盲目搜索：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。

（2）启发式搜索：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** 状态空间的搜索策略
- **5.3** 盲目的图搜索策略
- **5.4** 启发式图搜索策略

5.2 状态空间的搜索策略

- **5.2.1** 状态空间表示法
- **5.2.2** 状态空间的图描述

5.2.1 状态空间表示法

- 状态：表示系统状态、事实等叙述型知识的一组变量或数组：

$$Q = [q_1, q_2, \cdots, q_n]^T$$

- 操作：表示引起状态变化的过程型知识的一组关系或函数：

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

5.2.1 状态空间表示法

- 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。

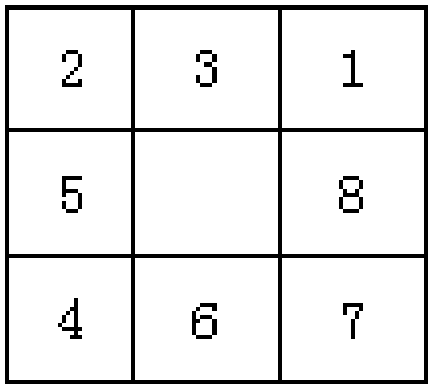
S_0 ：包含问题的初始状态是 S 的非空子集。

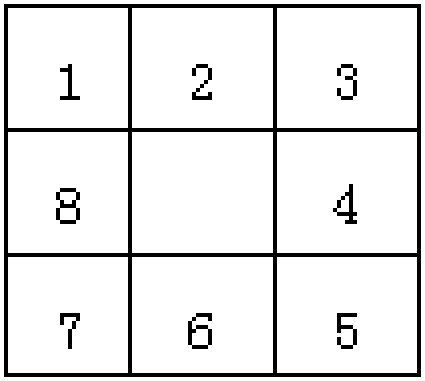
G ：若干具体状态或满足某些性质的路径信息描述。

Introduction of Artificial Intelligence

5.2.1 状态空间表示法

- 例 **5.1** 八数码问题的状态空间。

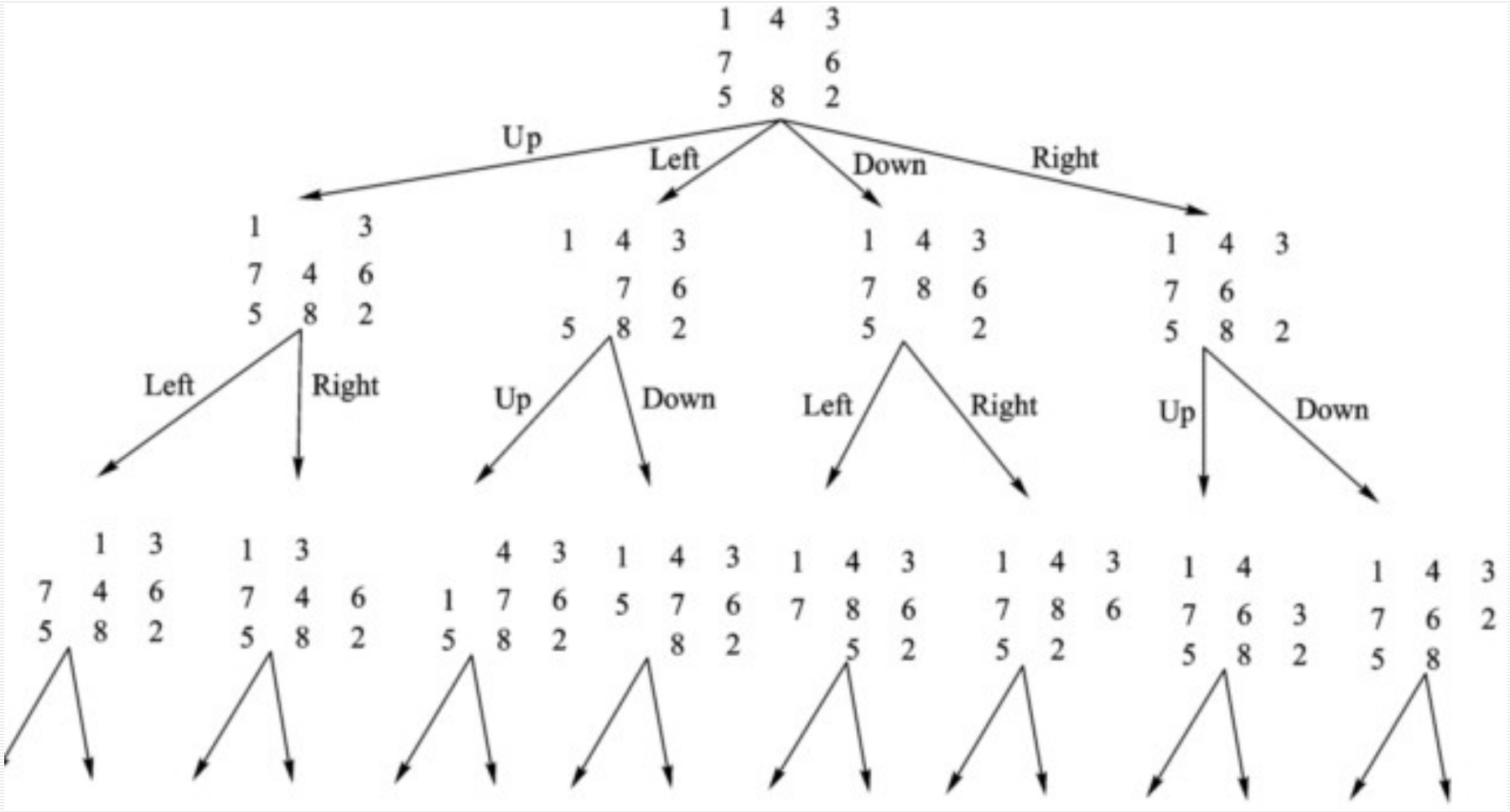




状态集 S ： 所有摆法　　操作算子：

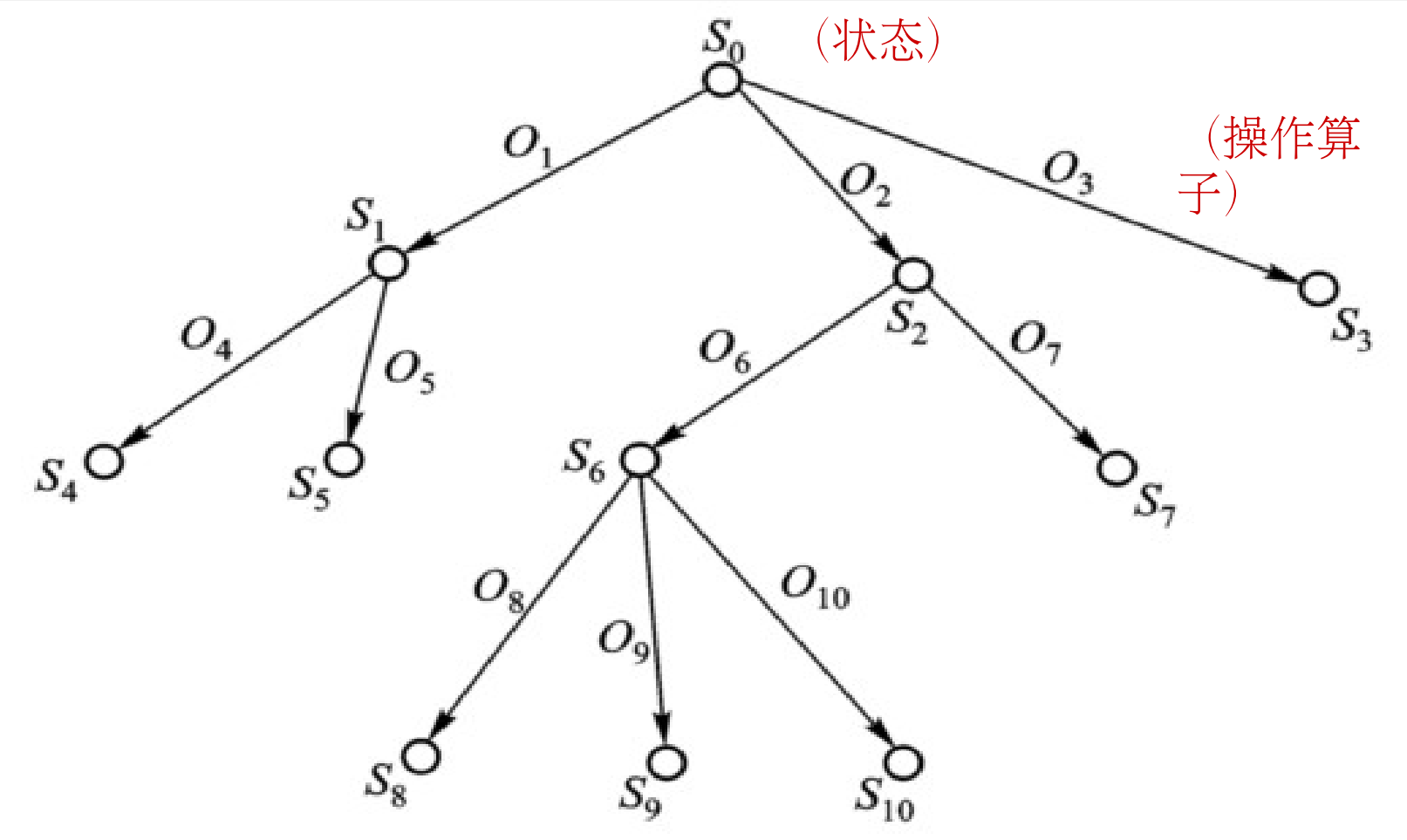
将空格向上移 Up
将空格向左移 Left
将空格向下移 Down
将空格向右移 Right

5.2.2 状态空间的图描述



八数码状态空间图

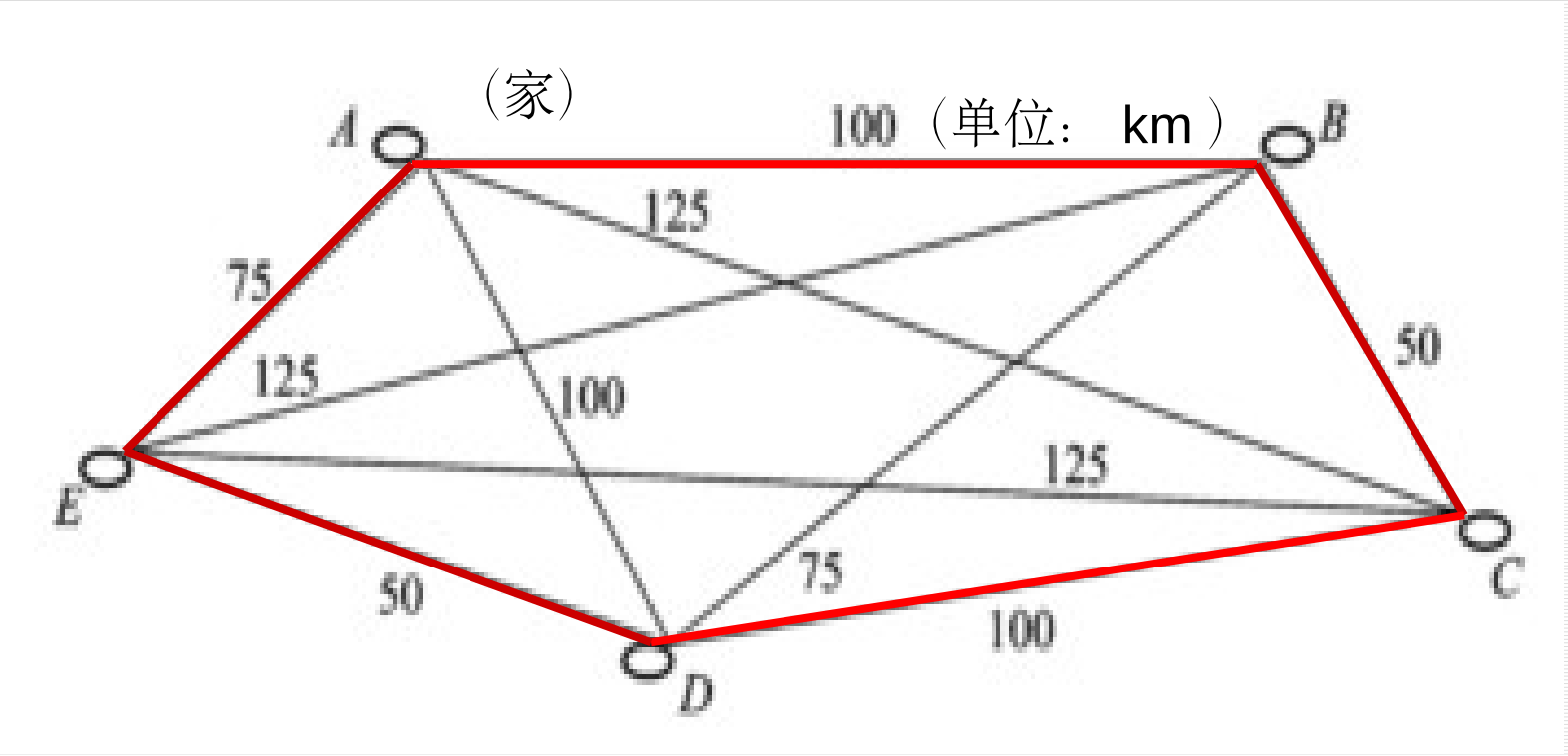
5.2.2 状态空间的图描述



状态空间的有向图描述

5.2.2 状态空间的图描述

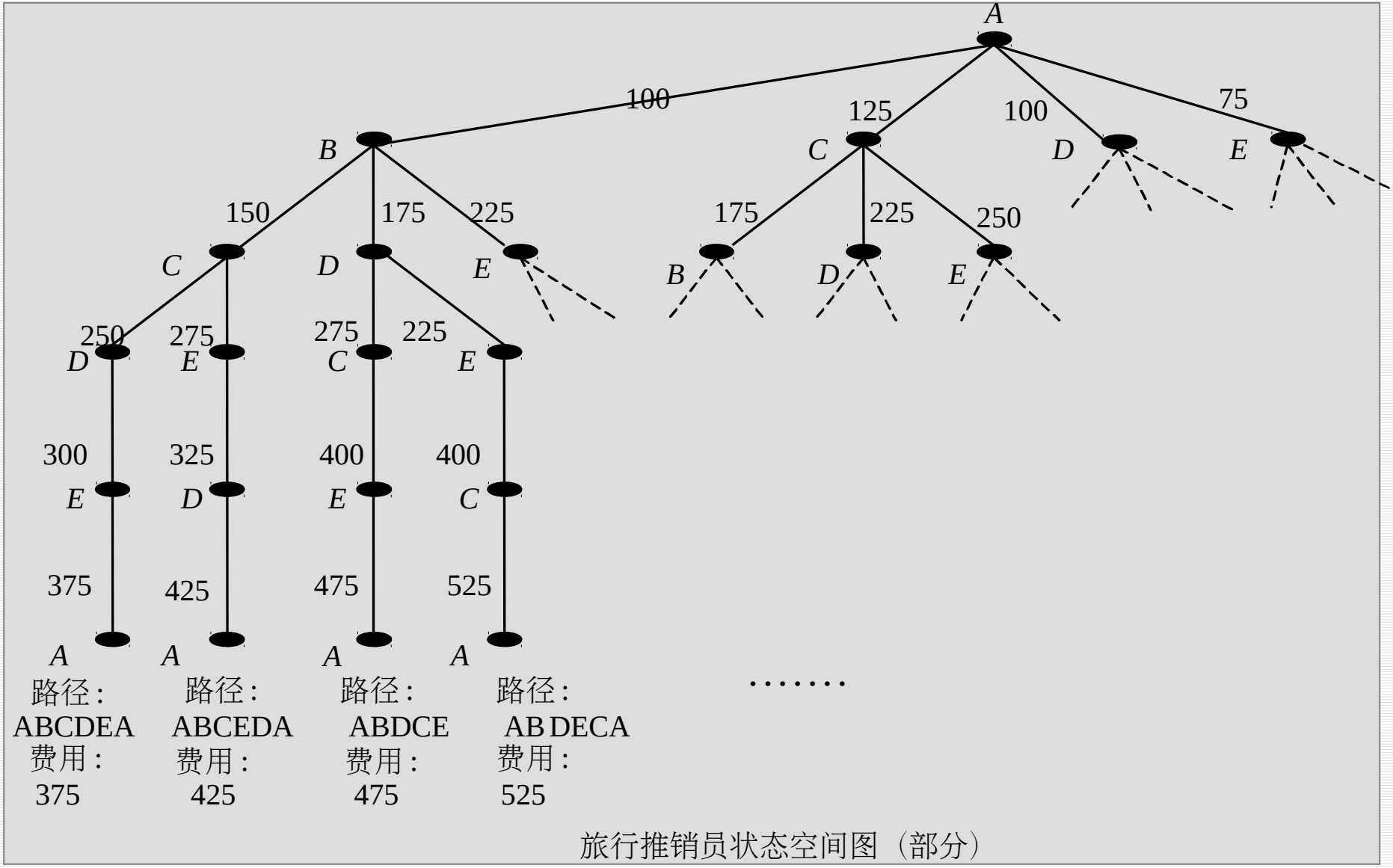
例 **5.3** 旅行商问题（**traveling salesman problem, TSP**）或邮递员路径问题。



可能路径：费用为 375 的路径

（A，B，C，D，E，A）

5.2.2 状态空间的图描述



旅行推销员状态空间图（部分）

第 5 章 搜索求解策略

- **5.1** 搜索的概念
- **5.2** 状态空间知识表示方法
- ✓ **5.3** 盲目的图搜索策略
- **5.4** 启发式图搜索策略

5.3 盲目的图搜索策略

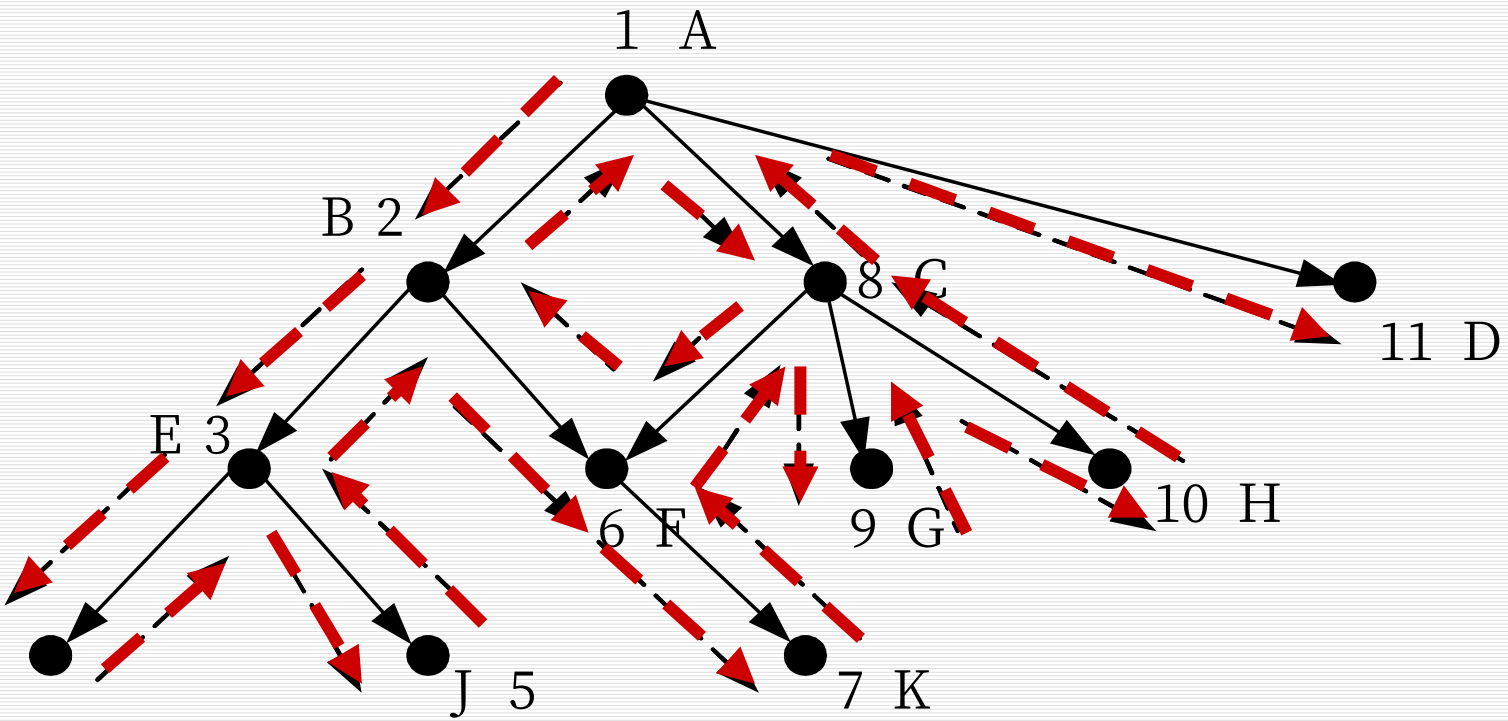
- **5.3.1** 回溯策略
- **5.3.2** 宽度优先搜索策略
- **5.3.3** 深度优先搜索策略

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

- 图搜索算法（深度优先、宽度优先、最好优先搜索等）的回溯思想:

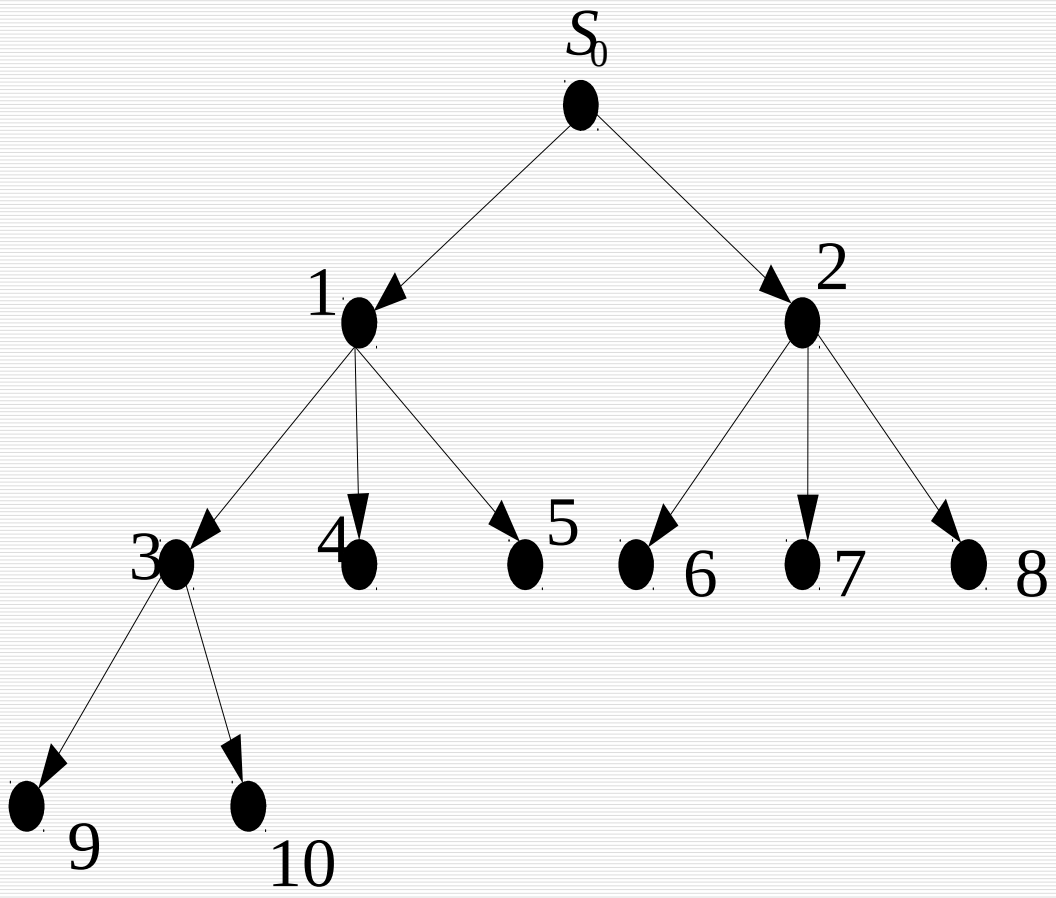
（1）用未处理状态表（NPS）使算法能返回（回溯）到其中任一状态。

（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，

5.3.2 宽度优先搜索策略

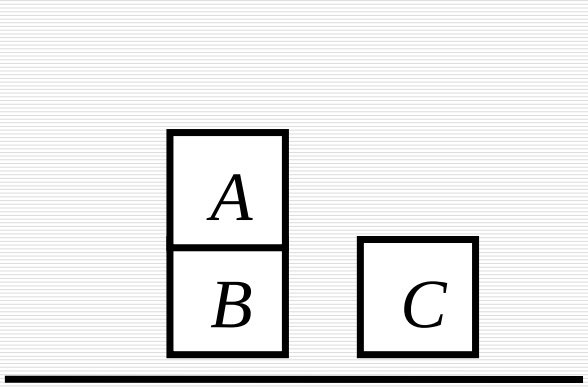


宽度优先搜索法中状态的搜索次序

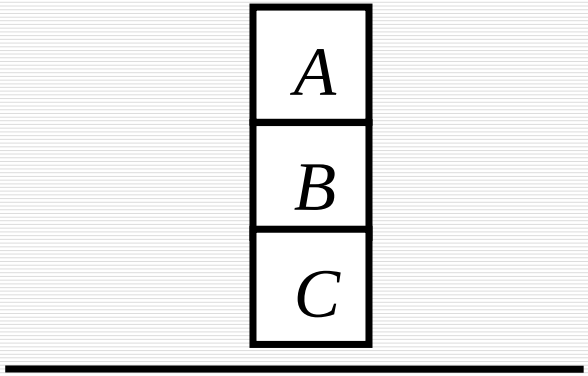
- open 表（NPS 表）：已经生成出来但其子状态未被搜索的状态。
- closed 表（ PS 表和 NSS 表的合并）：记录了已被生成扩展过的状态。

5.3.2 宽度优先搜索策略

- 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”。

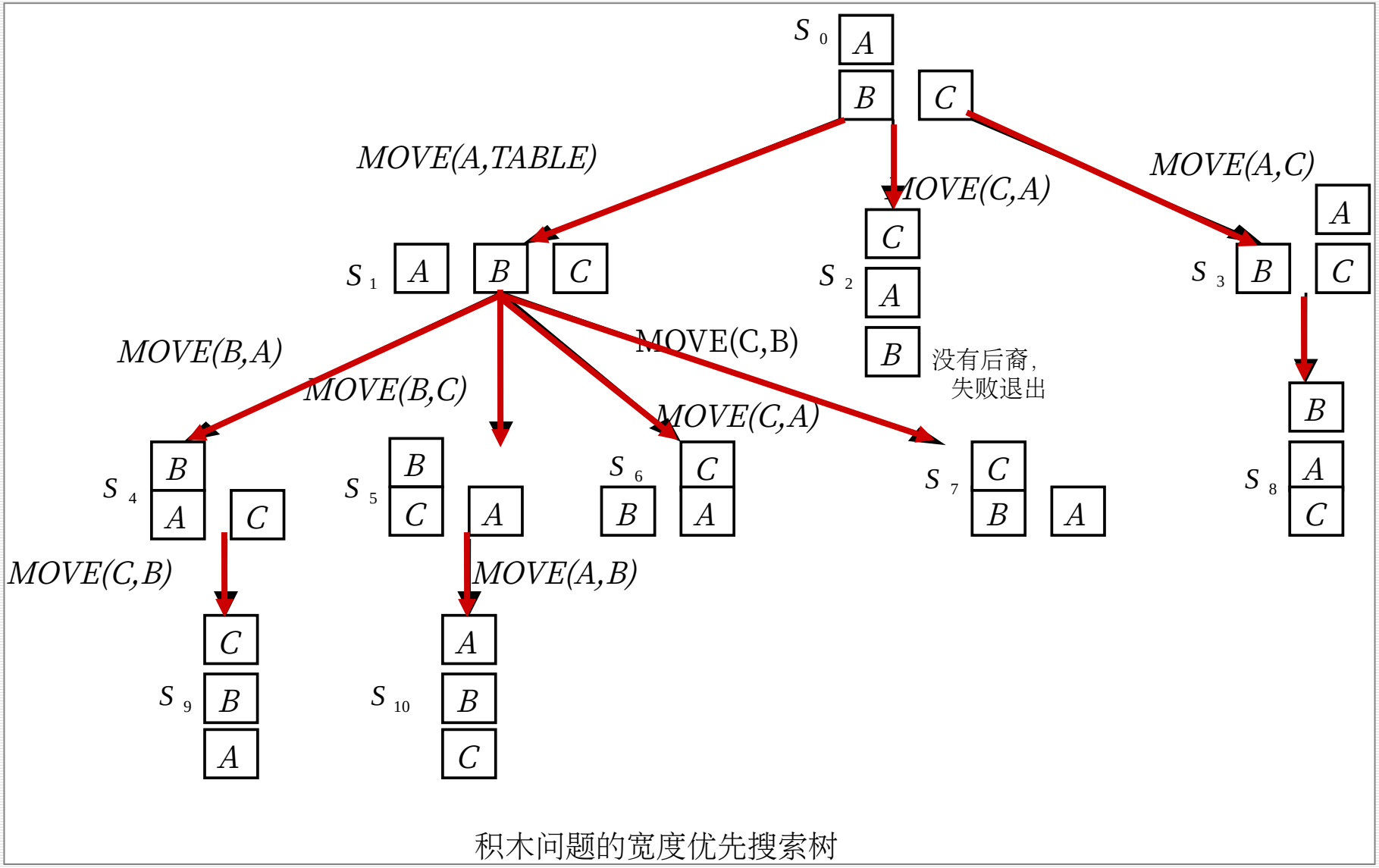
- 操作算子可运用的先决条件：

（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。

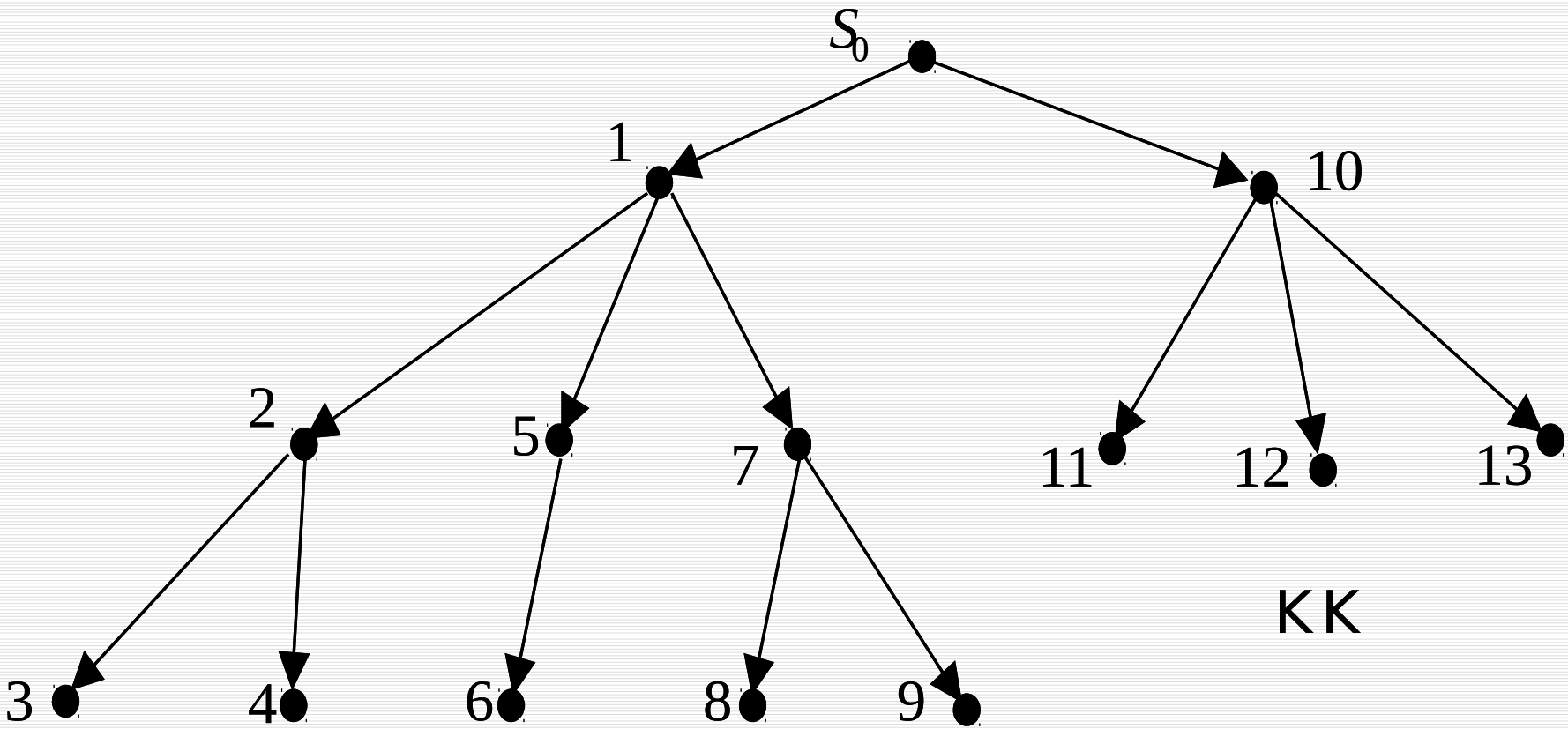
（3）同一状态下，运用操作算子的次数不得多于一次。

5.3.2 宽度优先搜索策略



积木问题的宽度优先搜索树

5.3.3 深度优先搜索策略



深度优先搜索法中状态的搜索次序

5.3.3 深度优先搜索策略

- 在深度优先搜索中，当搜索到某一个状态时，它所有的子状态以及子状态的后裔状态都必须先于该状态的兄弟状态被搜索。
- 为了保证找到解，应选择合适的深度限制值，或采取不断加大深度限制值的办法，反复搜索，直到找到解。

5.3.3 深度优先搜索策略

- 深度优先搜索并不能保证第一次搜索到的某个状态时的路径是到这个状态的最短路径。
- 对任何状态而言，以后的搜索有可能找到另一条通向它的路径。如果路径的长度对解题很关键的话，当算法多次搜索到同一个状态时，它应该保留最短路径。

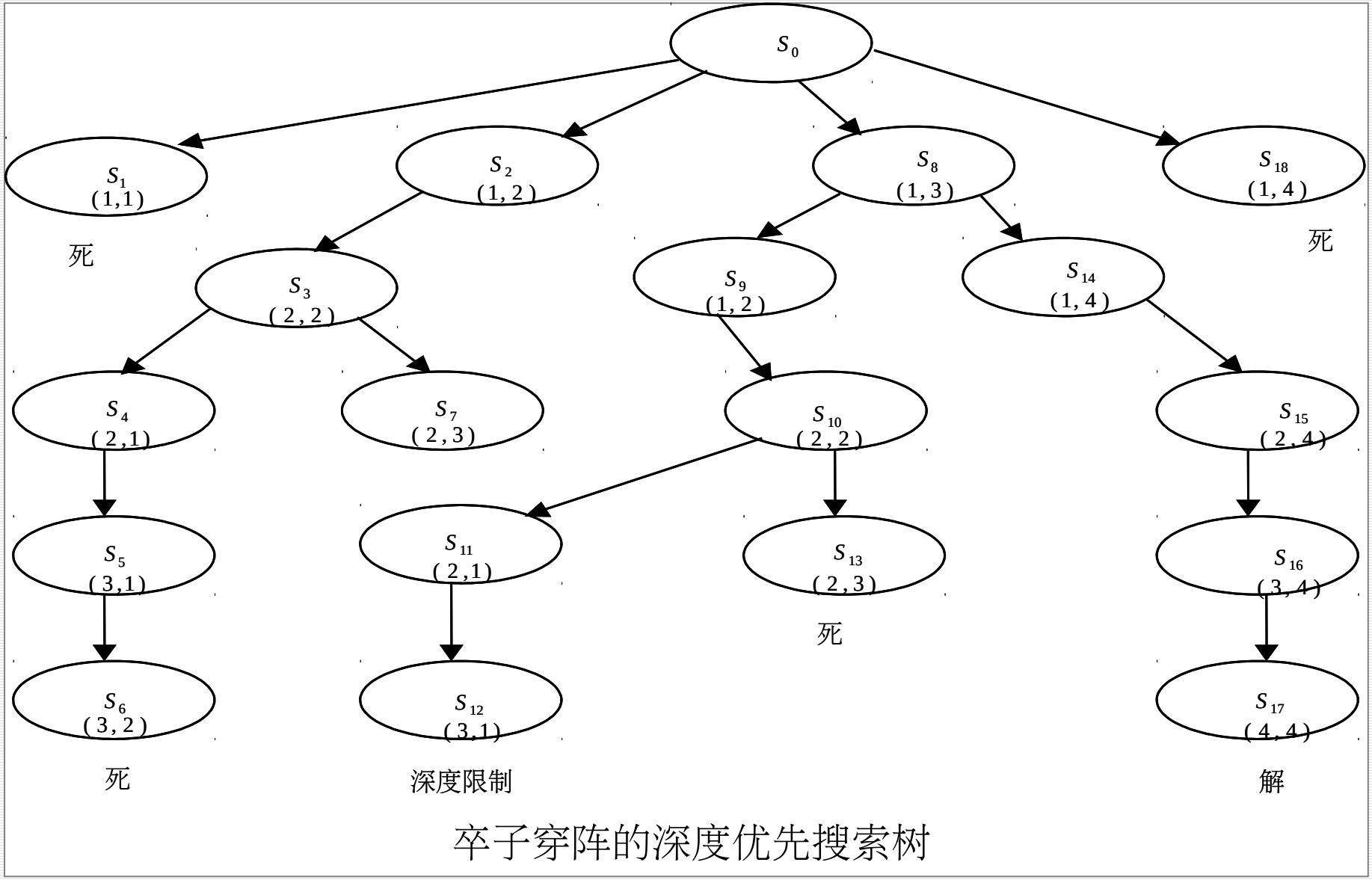
5.3.3 深度优先搜索策略

- 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。假定深度限制值为 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 深度优先搜索策略



卒子穿阵的深度优先搜索树

第 5 章 搜索求解策略

- **5.1** 搜索的概念
- **5.2** 状态空间知识表示方法
- **5.3** 盲目的图搜索策略
- ✓ **5.4** 启发式图搜索策略
- **5.5** **与 / 或**图搜索策略

5.4 启发式图搜索策略

- **5.4.1** 启发式策略
- **5.4.2** 启发信息和估价函数
- **5.4.3** ***A*** 搜索算法
- **5.4.4** ***A**** 搜索算法及其特性分析

5.4.1 启发式策略

- “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。

5.4.1 启发式策略

- 运用启发式策略的两种基本情况：

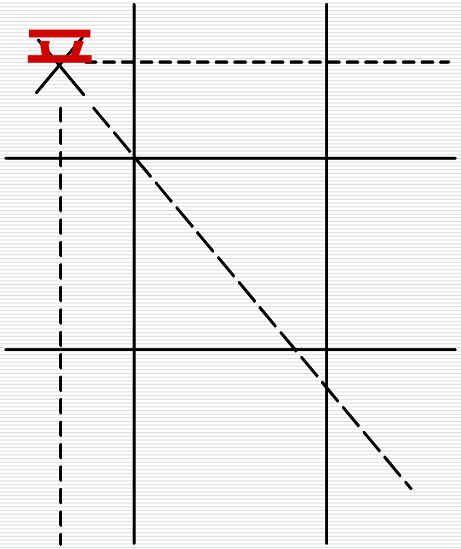
（1）一个问题由于在问题陈述和数据获取方面固有的模糊性，可能会使它没有一个确定的解。

（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。

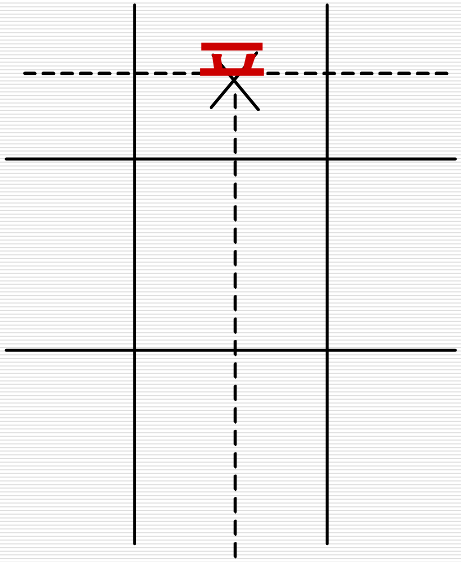
- 例 **5.6** 一字棋。在九宫棋盘上，从空棋盘开始，双方轮流在棋盘上摆各自的棋子 ㅍ 或 ㄴㅈ （每次一枚），谁先取得三子一线（一行、一列或一条对角线）的结果就取胜。

- ㅍ 和 ㄴㅈ 能够在棋盘中摆成的各种不同的棋局就是问题空间中的不同状态。
- 9 个位置上摆放 { 空， ㅍ ， ㄴㅈ } 有 3^9 种棋局。
- 可能的走法： $9\times8\times7\times\cdots\times1$ 。

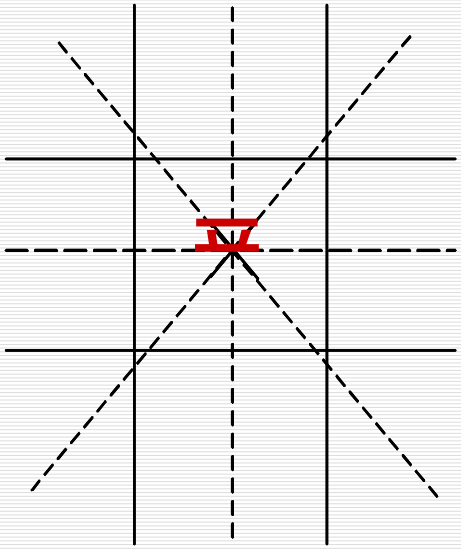
5.4.1 启发式策略



赢的几率 ③



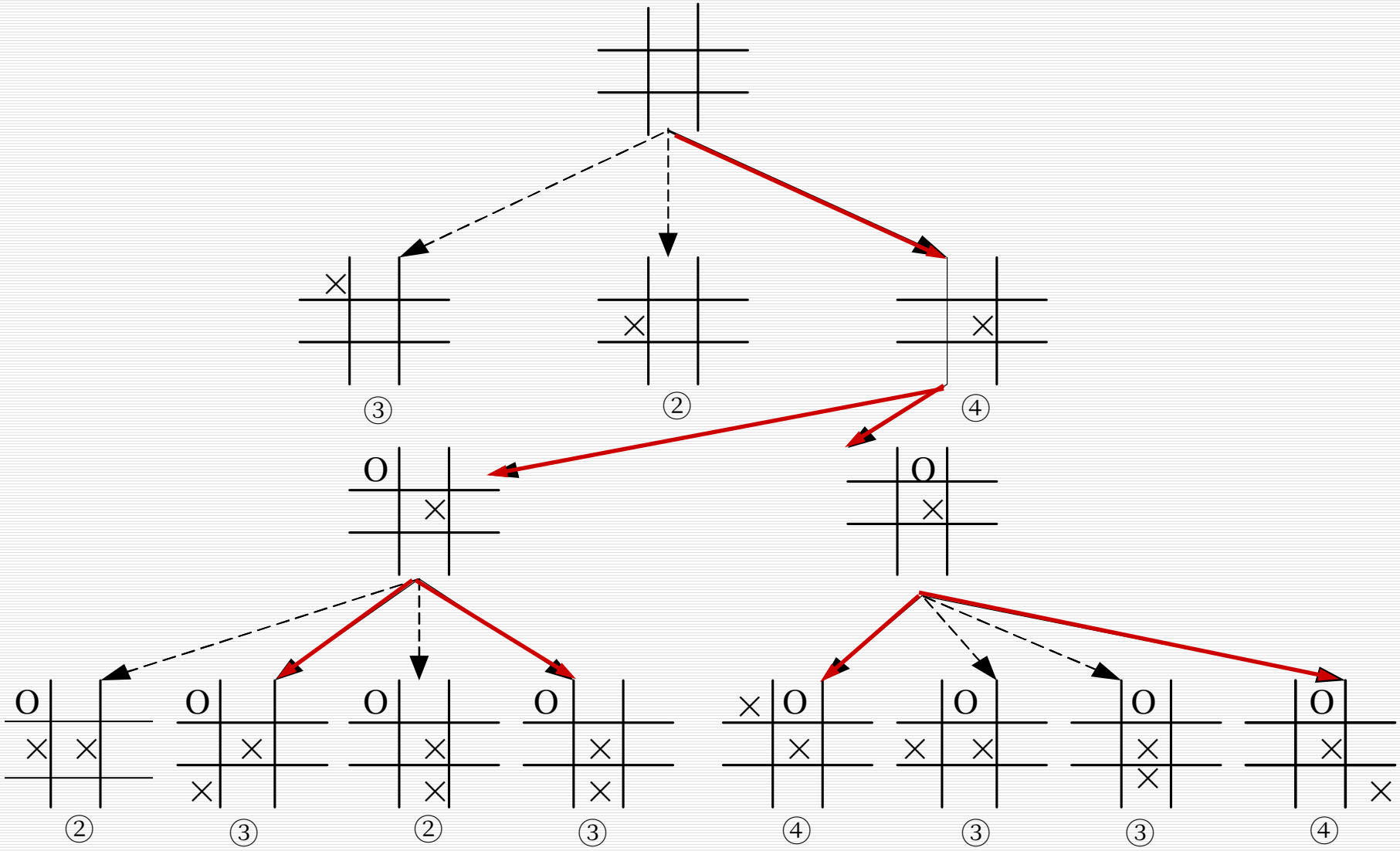
赢的几率 ②



赢的几率 ④

启发式策略的运用

5.4.1 启发式策略



启发式搜索下缩减的状态空间

5.4.2 启发信息和估价函数

- 在具体求解中，能够利用与该问题有关的信息来简化搜索过程，称此类信息为启发信息。
- 启发式搜索：利用启发信息的搜索过程。

5.4.2 启发信息和估价函数

- 求解问题中能利用的大多是非完备的启发信息：

（1）求解问题系统不可能知道与实际问题有关的全部信息，因而无法知道该问题的全部状态空间，也不可能用一套算法来求解所有的问题。

（2）有些问题在理论上虽然存在着求解算法，但是在工程实践中，这些算法不是效率太低，就是根本无法实现。

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！

5.4.2 启发信息和估价函数

- 启发信息的分类:

  （1）陈述性启发信息

  （2）过程性启发信息

  （3）控制性启发信息

- 利用控制性的启发信息的情况:

（1）没有任何控制性知识作为搜索的依据，因而搜索的每一步完全是随意的。

（2）有充分的控制知识作为依据，因而搜索的每一步选择都是正确的，但这是不现实的。

5.4.2 启发信息和估价函数

- 估价函数的任务就是估计待搜索结点的“有希望”程度，并依次给它们排定次序（在 open 表中）。

- 估价函数 $f(n)$：从初始结点经过 n 结点到达目的 结点的路径的最小代价估计值，其一般形式是

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

- 一般地，在 $f(n)$ 中，$g(n)$ 的比重越大，越倾向于宽度优先搜索方式，而 $h(n)$ 的比重越大，表示启发性能越强。

5.4.2 启发信息和估价函数

例 **5.7** 八数码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，其位置不符的将牌数目 。
- 较好的估价函数：各将牌移到目的位置所需移动的距离的总和。
- 第三种估价函数：对每一对逆转将牌乘以一个倍数。
- 第四种估价函数：克服了仅计算将牌逆转数目策略的局限，将位置不符将牌数目的总和与 3 倍将牌逆转数目相加。

5.4.3 *A* 搜索算法

- 启发式图搜索法的基本特点：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 $f(n)$ 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。

5.4.3 *A* 搜索算法

- 一般启发式图搜索算法（简记为 ***A*** ）

```
procedure heuristic_search
open：  =[start]；  closed：  =[ ]；  f(s)：  =g(s)+h(s)；  * 初始化
while open≠[ ] do
begin
从 open 表中删除第一个状态，称之为 n；
if n= 目的状态 then return(success)；
生成 n 的所有子状态;
if n 没有任何子状态 then continue；
for n 的每个子状态 do
case 子状态 is not already on open 表 or closed 表;
begin
计算该子状态的估价函数值;
将该子状态加到 open 表中;
        end；
```

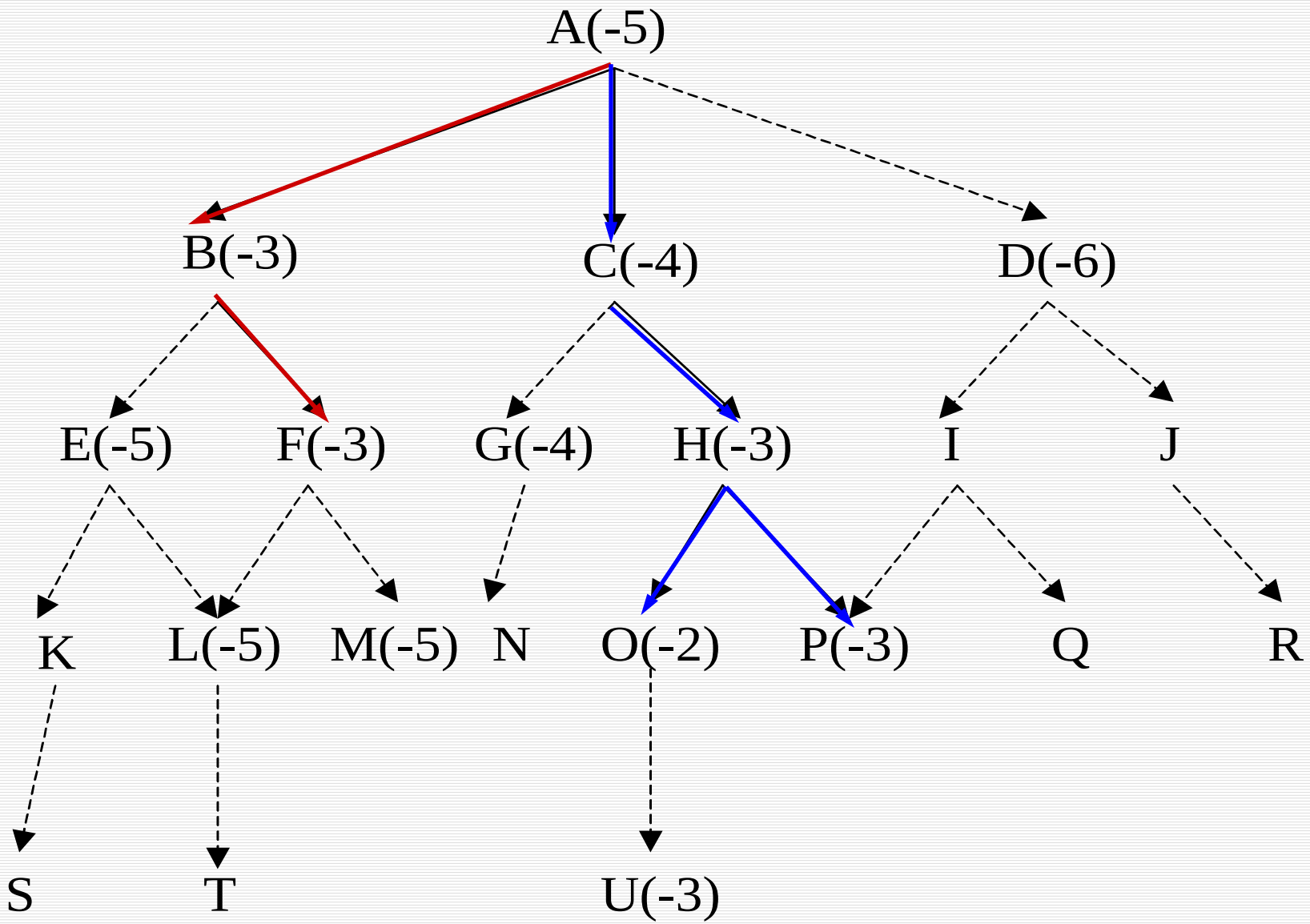
5.4.3 *A* 搜索算法

case 子状态 is already on open 表:
if 该子状态是沿着一条比在 open 表已有的更短路径而到达
then 记录更短路径走向及其估价函数值;
case 子状态 is already on closed 表:
if 该子状态是沿着一条比在 closed 表已有的更短路径而到达 then
begin
将该子状态从 closed 表移到 open 表中;
记录更短路径走向及其估价函数值;
end ;
case end ;
将 n 放入 closed 表中;
根据估价函数值，从小到大重新排列 open 表;
end ; *open 表中结点已耗尽
return(failure) ;
end.

5.4.3 *A* 搜索算法

- 每次重复时，***A*** 搜索算法从 **open** 表中取出第一个状态，如果该状态满足目的条件，则算法返回到该状态的搜索路径。
- 如果 **open** 表的第一个状态不是目的状态，则算法利用与之相匹配的一系列操作算子进行相应的操作来产生它的子状态。如果某个子状态已在 **open** 表（或 **closed** 表）中出现过，即该状态再一次被发现时，则通过刷新它的祖先状态的历史记录，使算法极有可能找到到达目的状态的更短的路径
- 接着，***A*** 搜索算法 **open** 表中每个状态的估价函数值，按照值的大小重新排序，将值最小的状态放在表头，使其第一个被扩展。

5.4.3 *A* 搜索算法



5.4.3 A 搜索算法

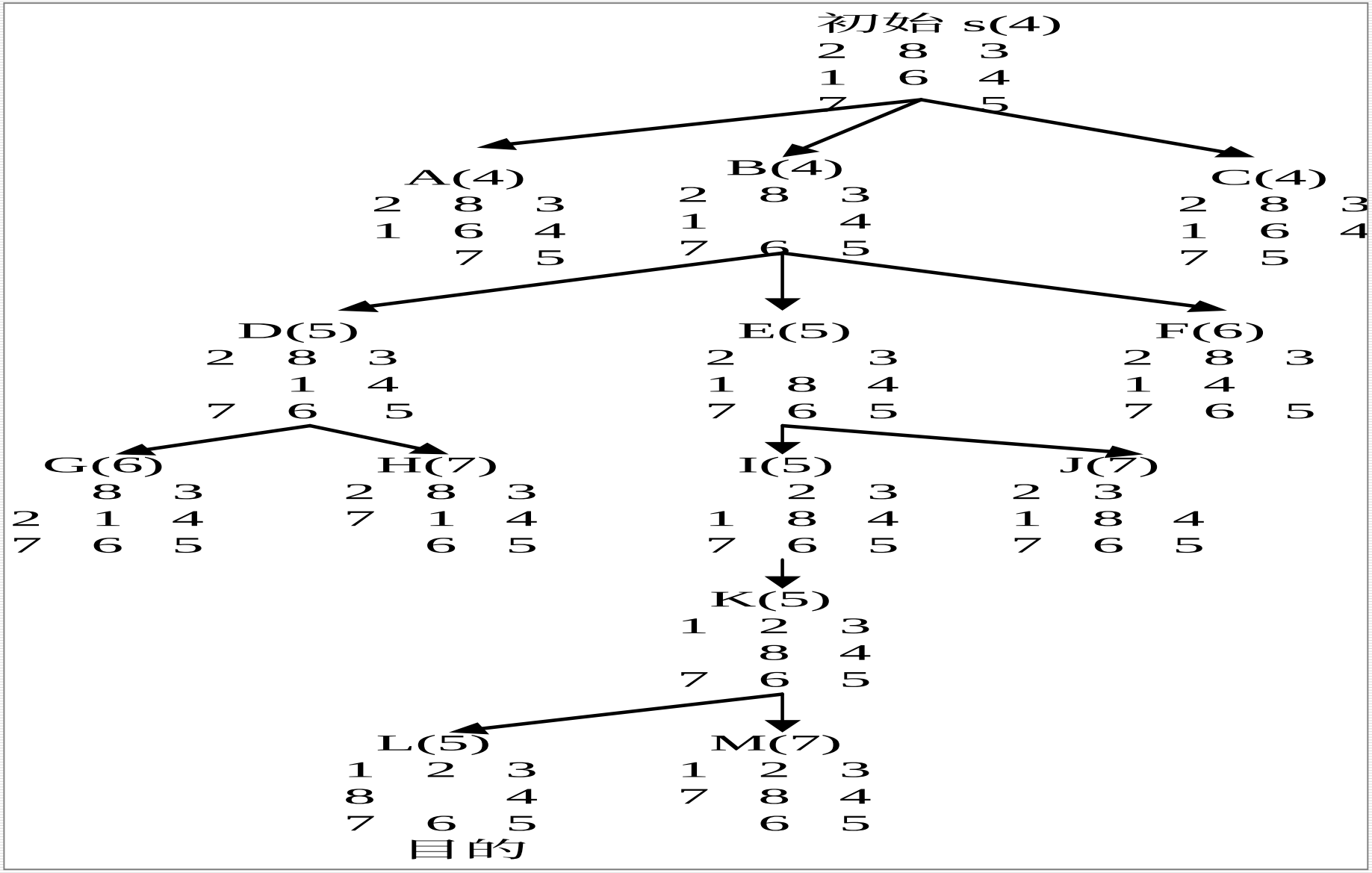
例 **5.8**　利用 A 搜索算法求解八数码问题的搜索树，其估价函数定义为

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

- $d(n)$：状态的深度，每步为单位代价。
- $w(n)$ 以“不在位”的将牌数作为启发信息的度量。

- $h^*(n)$ 为状态 n 到目的状态的最优路径的代价。
- $w(n)=h(n)\le h^*(n)$ A 搜索算法 → A^* 搜索算法。

5.4.3 *A* 搜索算法


初始 s(4)
2 8 3
1 6 4
7 5
A(4)
2 8 3
1 6 4
7 5
B(4)
2 8 3
1 4
7 6 5
C(4)
2 8 3
1 6 4
7 5
D(5)
2 8 3
1 4
7 6 5
E(5)
2 3
1 8 4
7 6 5
F(6)
2 8 3
1 4
7 6 5
G(6)
8 3
2 1 4
7 6 5
H(7)
2 8 3
7 1 4
6 5
I(5)
2 3
1 8 4
7 6 5
J(7)
2 3
1 8 4
7 6 5
K(5)
1 2 3
8 4
7 6 5
L(5)
1 2 3
8 4
7 6 5
目的
M(7)
1 2 3
7 8 4
6 5

5.4.3 *A* 搜索算法

- open 表和 closed 表内状态排列的变化情况

Open 表	Closed 表
初始化：(s(4))	(...)
一次循环后：	
(B(4) A(6) C(6) F(6))	(s(4))
二次循环后：	
(D(5) E(5) A(6) C(6) F(6))	(s(4) B(4))
三次循环后：	
(E(5) A(6) C(6) F(6) G(6) H(7))	(s(4) B(4) D(5))
四次循环后：	
(I(5) A(6) C(6) F(6) G(6) H(7) J(7))	(s(4) B(4) D(5) E(5))
五次循环后：	
(K(5) A(6) C(6) F(6) G(6) H(7) J(7))	(s(4) B(4) D(5) E(5) I(5))
六次循环后：	
(L(5) A(6) C(6) F(6) G(6) H(7) J(7) M(7))	(s(4) B(4) D(5) E(5) I(5) K(5))
七次循环后：	
L 为目的状态，则成功推出，结束搜索	(s(4) B(4) D(5) E(5) I(5) K(5) L(5))

5.4.4 A^* 搜索算法及其特性分析

- 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。
- 上例中的八数码 A 搜索树也是 A^* 搜索树，所得的解路（s，B，E，I，K，L）为最优解路，其步数为状态 L （5）上所标注的 5 。

5.4.4 *A** 搜索算法及其特性分析

1. 可采纳性

当一个搜索算法在最短路径存在时能保证找到它，就称它是可采纳的。

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。

5.4.4 A^* 搜索算法及其特性分析

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 $h_1(n) \leq h_2(n)$，就称策略 h_2 比 h_1 具有更好的信息性。

THE END