



# 第六章 问题求解与搜索策略

- 6.1 基本概念
- 6.2 状态空间的搜索策略
- 6.3 与/或树的搜索策略
- 6.4 搜索的完备性与效率

# 问题求解

- 问题求解是人工智能的核心问题之一
- 问题求解的目的
  - 机器自动找出某问题的正确解决策略
  - 更进一步，能够举一反三，具有解决同类问题的能力
- 是从人工智能初期的智力难题、棋类游戏等问题的研究中开始形成和发展起来的一大类技术，搜索技术是问题求解的主要手段之一
  - 问题表示
  - 解的搜索

## ■ 示例——八数码难题

在 $3 \times 3$ 的棋盘，摆有八个棋子，每个棋子上标有1至8的某一数字。棋盘上还有一个空格，与空格相邻的棋子可以移到空格中。

2	8	3
1		4
7	6	5

初始状态



1	2	3
8		4
7	6	5

目标状态

- 如何将棋盘从某一初始状态变成最后的目标状态？

# 问题示例



■ 怎样找到  
两点之间的  
最短路  
径呢？

## 6.1.2 状态空间表示法

(1) 很多问题的求解过程都可以看作是一个搜索过程。问题及其求解过程可以用状态空间表示法来表示。

(2) 状态空间用“状态”和“算符”来表示问题。

### □ 状态

状态用以描述问题在求解过程中不同时刻的状态，一般用一个向量表示：

$$S_K=(S_{k0},S_{k1},\dots)$$

### □ 算符

使问题从一个状态转变为另一个状态的操作称为算符。在产生式系统中，一条产生式规则就是一个算符。

### □ 状态空间

由所有可能出现的状态及一切可用算符所构成的集合称为问题的状态空间。

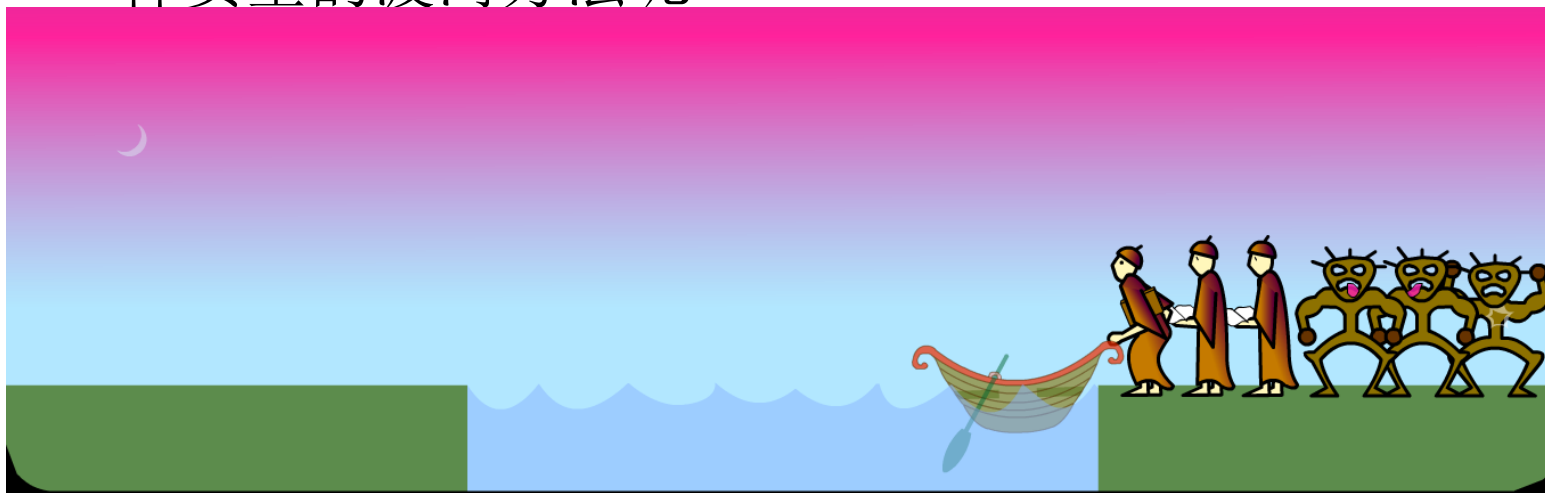
(3) 采用状态空间求解问题，可以用下面的一个三元组表示：

$$(S,F,G)$$

其中S是问题初始状态的集合；F是算符的集合；G是目标状态的集合。

## ■ 传教士野人问题 ( Missionaries& Cannibals, MC问题)

有三个传教士M和三个野人C过河，只有一条能装下两个人的船，在河的一方或者船上，如果野人的人数大于传教士的人数，那么传教士就会有危险，你能不能提出一种安全的渡河方法呢？



- 状态：问题在某一时刻所处的“位置”，“情况”等
- 根据问题所关心的因素，一般用向量形式表示，每一位表示一个因素



状态可有多种表示方法：

(左岸传教士数, 右岸传教士数, 左岸野人数, 右岸野人数, 船的位置)

或

(左岸传教士数, 左岸野人数, 船的位置)

初始状态：(0, 0, 0)

0：右岸

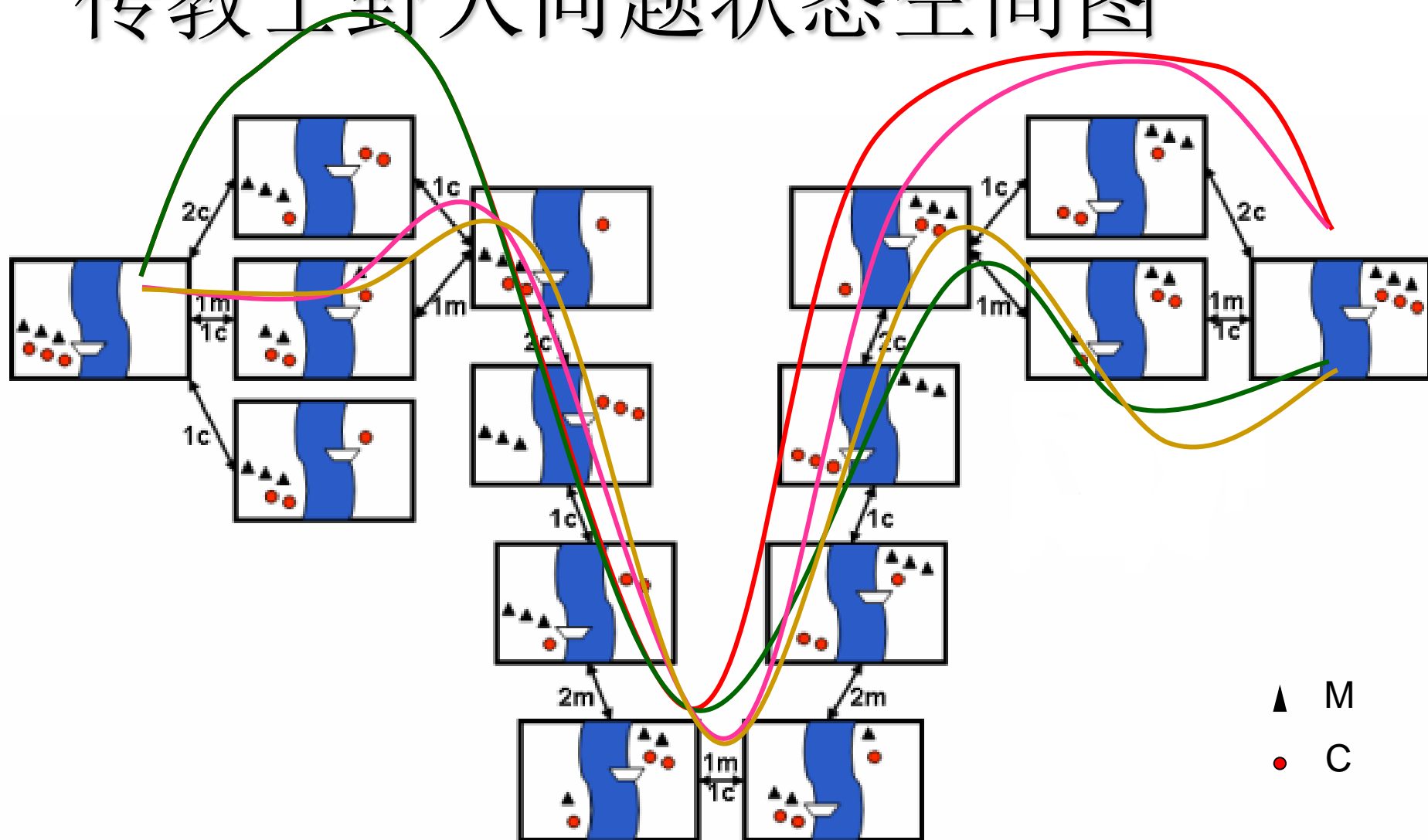
目标状态：(3, 3, 1)

1：左岸

- 算子（算符，操作符）——使状态发生改变的操作
- MC问题中的算子
  - 将传教士或野人运到河对岸
  - **Move-1m1c-lr**: 将一个传教士(m)一个野人(c)从左岸(l)运到右岸(r)
  - 所有可能操作
    - |                     |                     |                   |
|---------------------|---------------------|-------------------|
| <b>Move-1m1c-lr</b> | <b>Move-1m1c-rl</b> | <b>Move-2c-lr</b> |
| <b>Move-2c-rl</b>   | <b>Move-2m-lr</b>   | <b>Move-2m-rl</b> |
| <b>Move-1c-lr</b>   | <b>Move-1c-rl</b>   | <b>Move-1m-lr</b> |
| <b>Move-1m-rl</b>   |                     |                   |

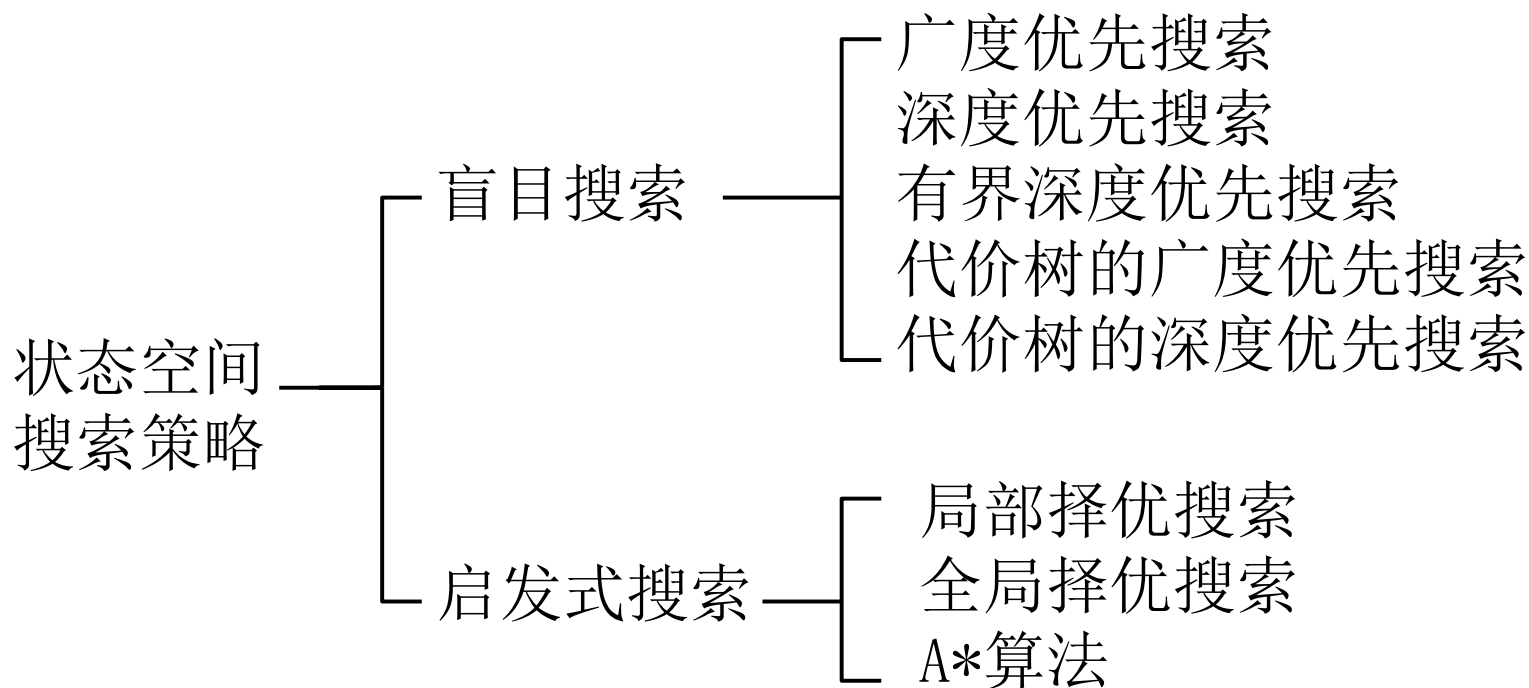


# 传教士野人问题状态空间图



## 6.2 状态空间的搜索策略

求解过程转化为在状态空间图中**搜索**一条从初始节点到目标节点的路径问题



## 6.2.1 状态空间的一般搜索过程



- 必须记住下一步还可以走哪些点  
OPEN表(记录还没有扩展的点)
- 必须记住哪些点走过了  
CLOSED表(记录已经扩展的点)
- 必须记住从目标返回的路径  
每个状态的节点结构中必须有指向父节点的指针

## ■ OPEN表和CLOSE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。
- CLOSE表用于存放将要扩展的节点。对一个节点的扩展是指：用所有可适用的算符对该节点进行操作，生成一组子节点

OPEN表

状态节点	父节点

CLOSE表

编号	状态节点	父节点

# 搜索的一般过程

1. 把初始节点 $S_0$ 放入OPEN表，并建立目前只包含 $S_0$ 的图，记为G；
2. 检查OPEN表是否为空，若为空则问题无解，退出；
3. 把OPEN表的第一个节点取出放入CLOSE表，并计该节点为n；
4. 考察节点n是否为目标节点。若是，则求得了问题的解，退出；
5. 扩展节点n，生成一组子节点。把其中不是节点n先辈的那些子节点记做集合M，并把这些子节点作为节点n的子节点加入G中；
6. 对于那些未曾在G中出现过的M成员设置一个指向父节点（即节点n）的指针，并把它们放入OPEN表；（不在OPEN表）
7. 按某种搜索策略对OPEN表中的节点进行排序；
8. 转第2步。

# 一些说明

- 一个节点经一个算符操作后一般只生成一个子节点。但适用于一个节点的算符可能有多个，此时就会生成一组子节点。这些子节点中可能有些是当前扩展节点的父节点、祖父节点等，此时不能把这些先辈节点作为当前扩展节点的子节点。
- 一个新生成的节点，它可能是第一次被生成的节点，也可能是先前已作为其它节点的子节点被生成过，当前又作为另一个节点的子节点被再次生成。此时，它究竟应选择哪个节点作为父节点？一般由原始节点到该节点的代价来决定，处于代价小的路途上的那个节点就作为该节点的父节点。
- 在搜索过程中，一旦某个被考察的节点是目标节点就得到了一个解。该解是由从初始节点到该目标节点路径上的算符构成。
- 如果在搜索中一直找不到目标节点，而且**OPEN**表中不再有可供扩展的节点，则搜索失败。
- 通过搜索得到的图称为搜索图，搜索图是状态空间图的一个子集。由搜索图中的所有节点及反向指针所构成的集合是一棵树，称为搜索树。根据搜索树可给出问题的解。

# 盲目搜索

- 不同的搜索策略其搜索的效率是不同的
- 盲目搜索又称无信息搜索
  - 宽度优先搜索
  - 深度优先搜索
  - 特点
    - 搜索过程中不使用与问题有关的经验信息
    - 采用固定策略对OPEN表排序
    - 搜索效率低
    - 不适合大空间的实际问题求解

## 6.2.2 广度优先搜索

- 基本思想：

从初始节点 $S_0$ 开始，逐层地对节点进行扩展并考察它是否为目标节点。在第 $n$ 层的节点没有全部扩展并考察之前，不对第 $n+1$ 层的节点进行扩展。

- OPEN表中节点总是按进入的先后顺序排列，先进入的节点排在前面，后进入的排在后面。

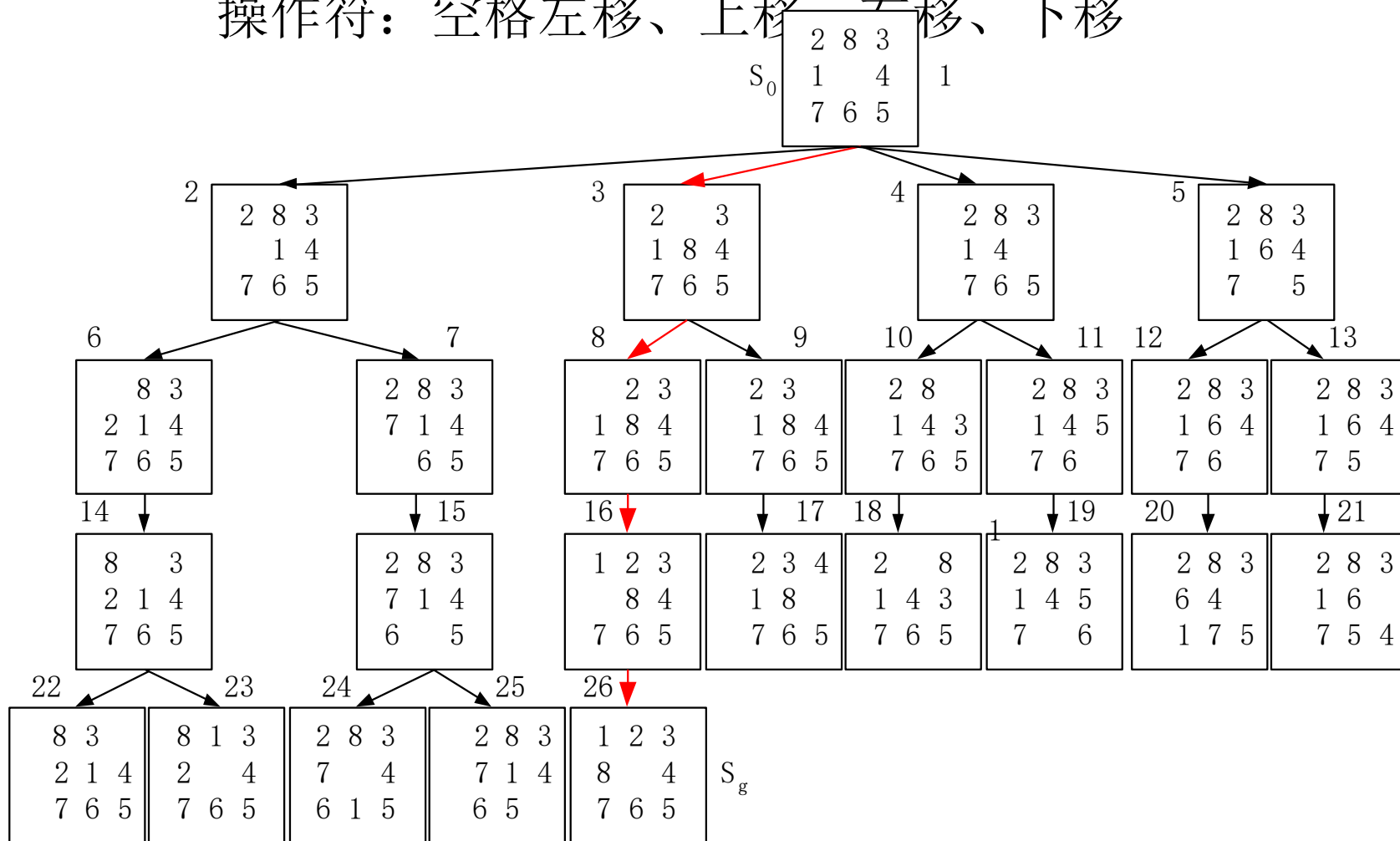


# 广度优先搜索过程

1. 把初始节点 $S_0$ 放入OPEN表。
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CLOSE表。
4. 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点n不可扩展，则转第2步。
6. 扩展节点n，将其子节点放入OPEN表的尾部，并为每一个子节点都配置指向父节点的指针，然后转第2步。

# 重排九宫的广度优先搜索

操作符：空格左移、上移、下移





在上述广度优先算法中需要注意两个问题：

- 1、对于任意一个可扩展的节点，总是按照固定的操作符的顺序对其进行扩展（空格左移、上移、右移、下移）。
- 2、在对任一节点进行扩展的时候，如果所得的某个子节点（状态）前面已经出现过，则立即将其放弃，不再重复画出（不送入**OPEN**表）。

因此，广度优先搜索的本质是，以初始节点为根节点，在状态空间图中按照广度优先的原则，生成一棵搜索树。

# 广度优先搜索的特点

- 优点：

只要问题有解，用广度优先搜索总可以得到解，而且得到的是路径最短的解。

- 缺点：

广度优先搜索盲目性较大，当目标节点距初始节点较远时将会产生许多无用节点，搜索效率低。

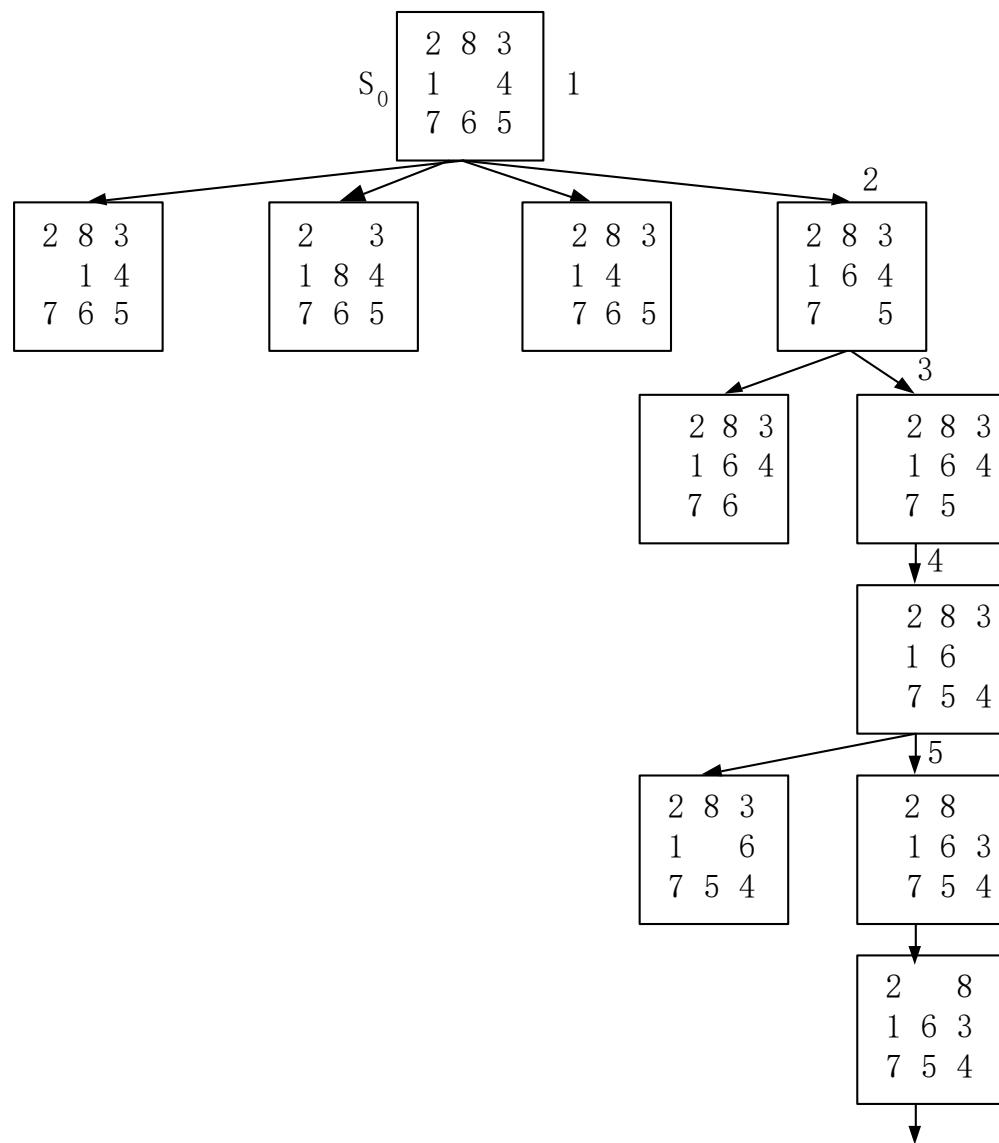
## 6.2.3 深度优先搜索

- 深度优先搜索与广度优先搜索的唯一区别是：广度优先搜索是将节点 $n$ 的子节点放入到**OPEN**表的尾部，而深度优先搜索是把节点 $n$ 的子节点放入到**OPEN**表的首部。

# 深度优先搜索过程

1. 把初始节点 $S_0$ 放入OPEN表。
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CLOSE表。
4. 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点n不可扩展，则转第2步。
6. 扩展节点n，将其子节点放入OPEN表的首部，并为每一个子节点都配置指向父节点的指针，然后转第2步。

# 重排九宫的深度优先搜索



# 深度优先搜索的特点

- 在深度优先搜索中，搜索一旦进入某个分支，就将沿着该分支一直向下搜索。如果目标节点恰好在此分支上，则可较快地得到解。但是，如果目标节点不在此分支上，而该分支又是一个无穷分支，则就不可能得到解。所以深度优先搜索是不完备的，即使问题有解，它也不一定求得解。
- 本质：以初始节点为根节点，在状态空间图中按照深度优先的原则，生成一棵搜索树。



## 6.2.4 有界深度优先搜索

### ■ 基本思想：

- 对深度优先搜索引入搜索深度的界限（设为 $d_m$ ），当搜索深度达到了深度界限，而仍未出现目标节点时，就换一个分支进行搜索。

### ■ 搜索过程：

1. 把初始节点 $S_0$ 放入OPEN表中，置 $S_0$ 的深度 $d(S_0)=0$ 。
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CLOSE表。
4. 考察节点 $n$ 是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点 $n$ 的深度 $d(n)=d_m$ ，则转第2步（此时节点 $n$ 位于CLOSE表，但并未进行扩展）。
6. 若节点 $n$ 不可扩展，则转第2步。
7. 扩展节点 $n$ ，将其子节点放入OPEN表的首部，为每一个子节点都配置指向父节点的指针，将每一个子节点的深度设置为 $d(n)+1$ ，然后转第2步。

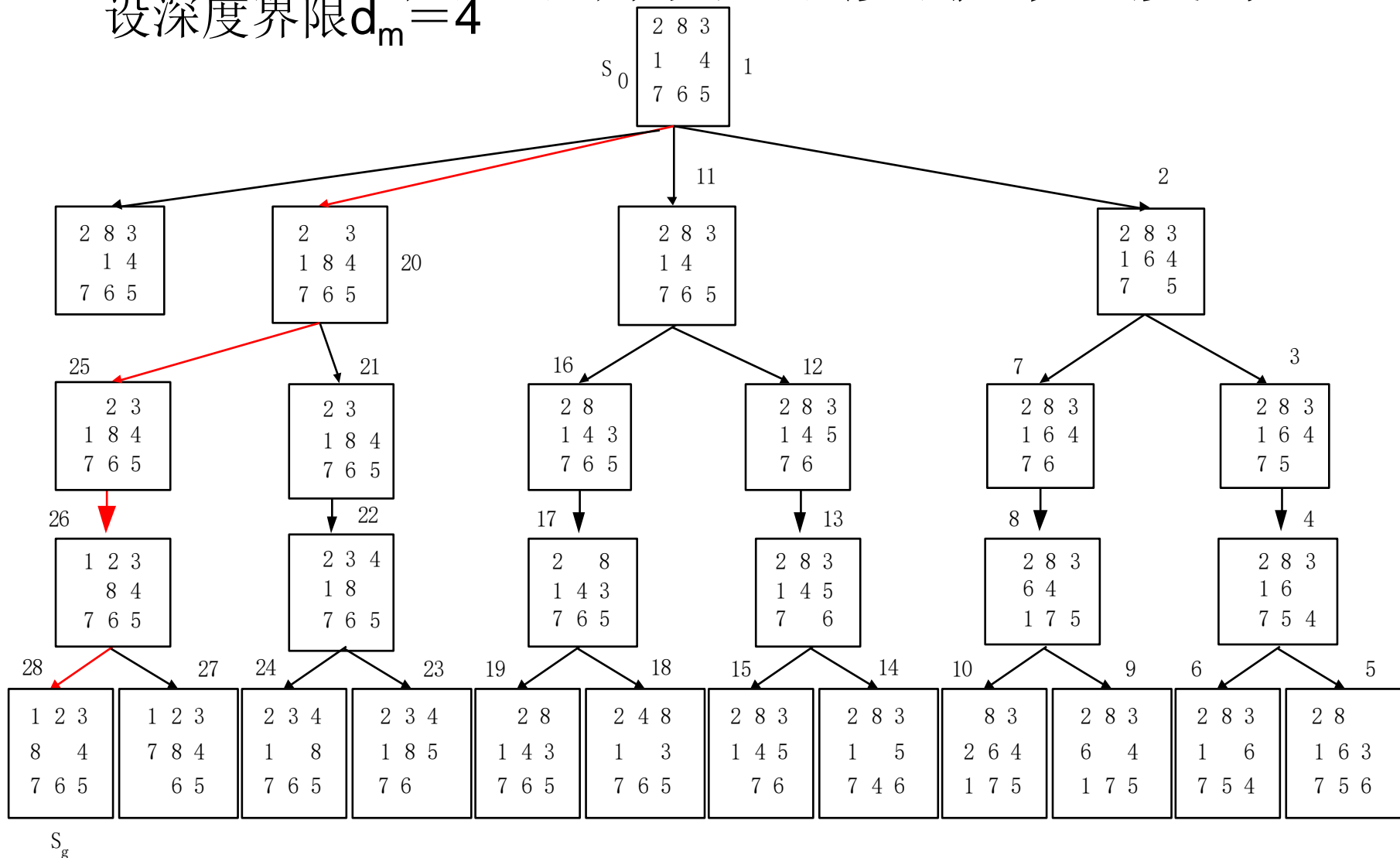
- 如果问题有解，且其路径长度 $\leq d_m$ ，则上述搜索过程一定能求得解。但是，若解的路径长度 $> d_m$ ，则上述搜索过程就得不到解。这说明在有界深度优先搜索中，深度界限的选择是很重要的。
- 要恰当地给出 $d_m$ 的值是比较困难的。即使能求出解，它也不一定是最优解。

# 有界深度优先搜索的一些改进方法

1. 先任意设定一个较小的数作为 $d_m$ ，然后进行上述的有界深度优先搜索，当搜索达到了指定的深度界限 $d_m$ 仍未发现目标节点，并且CLOSE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 $d_m$ ，继续向下搜索。如此不断地增大 $d_m$ ，只要问题有解，就一定可以找到它。但此时找到的解不一定是最优解。
2. 为了找到最优解，可增设一个表R，每找到目标节点 $S_g$ 后，就把它放入到R的前面，并令 $d_m$ 等于该目标节点所对应的路径长度，然后继续搜索。由于后求得的解的路径长度不会超过先求得的解的路径长度，所以后求得的解一定是最优解。

# 重排九宫的有界深度优先搜索

设深度界限 $d_m=4$



## 6.2.5 代价树的广度优先搜索

- 边上标有代价(或费用)的树称为代价树。
- 用 $g(x)$ 表示从初始节点 $S_0$ 到节点 $x$ 的代价，用 $c(x_1, x_2)$ 表示从父节点 $x_1$ 到子节点 $x_2$ 的代价，则有：

$$g(x_2) = g(x_1) + c(x_1, x_2)$$

- 基本思想：

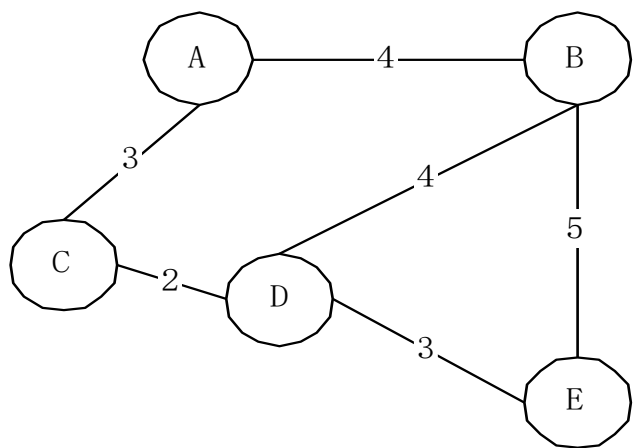
每次从**OPEN**表中选择节点往**CLOSE**表传送时，总是选择其中代价最小的节点。也就是说，**OPEN**表中的节点在任一时刻都是按其代价从小到大排序的。代价小的节点排在前面，代价大的节点排在后面。

- 如果问题有解，代价树的广度优先搜索一定可以求得解，并且求出的是最优解。
- 该算法应用的条件:该算法是针对代价树的算法。为了采用该算法对图进行搜索，必须先将图转换为代价树。转换方法见例6.7。

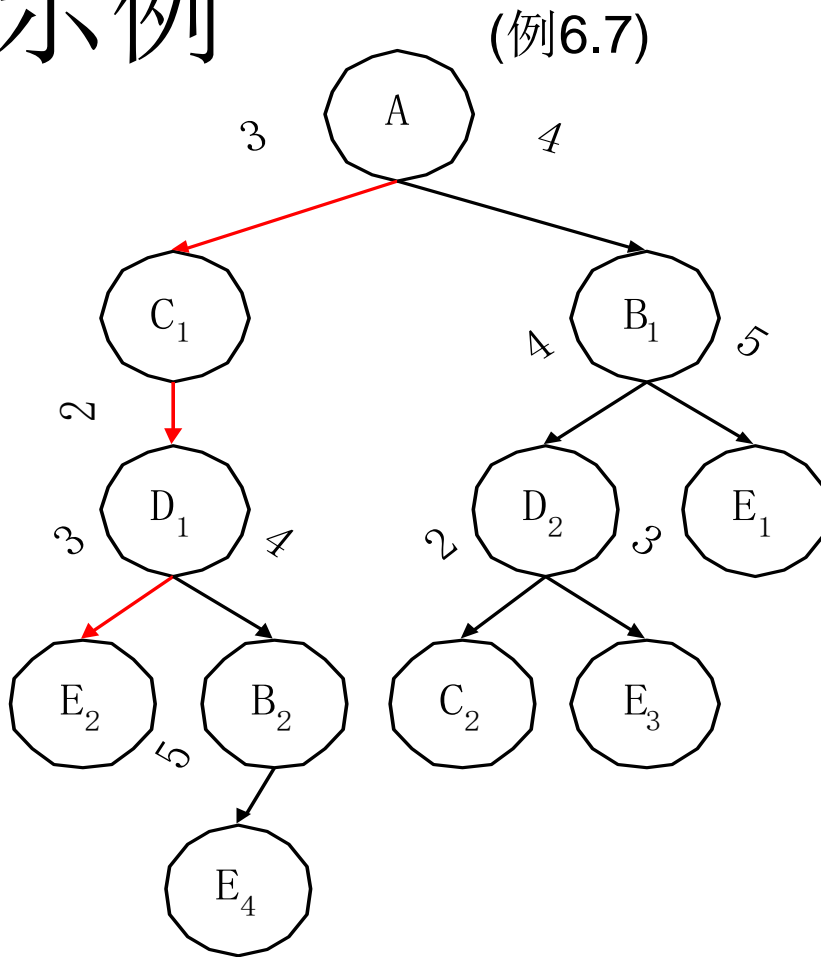
# 代价树广度优先搜索过程

1. 把初始节点 $S_0$ 放入OPEN表，令 $g(S_0)=0$ 。
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CLOSE表。
4. 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点n不可扩展，则转第2步。
6. 扩展节点n，为每一个子节点都配置指向父节点的指针，计算各子节点的代价，并将各子节点放入OPEN表中。按各节点的代价对OPEN表中的全部节点进行排序(按从小到大的顺序)，然后转第2步。

# 代价树示例



交通图



交通图的代价树

## 6.2.6 代价树的深度优先搜索

### ■ 搜索过程：

1. 把初始节点 $S_0$ 放入OPEN表，令 $g(S_0)=0$ 。
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CLOSE表。
4. 考察节点 $n$ 是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点 $n$ 不可扩展，则转第2步。
6. 扩展节点 $n$ ，将其子节点按代价从小到大的顺序放到OPEN表中的首部，并为每一个子节点都配置指向父节点的指针，然后转第2步。

### ■ 代价树的深度有限搜索是不完备的。



# 总 结

- 1、上述各种搜索方法的本质是，以初始节点为根节点，按照既定的策略对状态空间图进行遍历，并希望能够尽早发现目标节点。
- 2、由于对状态空间图遍历的策略是既定的，因此这些方法统称为盲目搜索方法。

## 6.2.7 启发式搜索

### ■ 有信息搜索

- 搜索过程中利用与问题有关的经验信息（**启发式信息**）
- 引入估价函数来估计节点位于解路径上的“希望”，函数值越小“希望”越大
- 搜索过程中按照估价函数的大小对**OPEN**表排序
- 每次选择估价函数值最小的节点作为下一步考察的节点

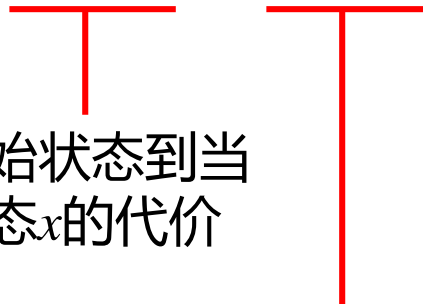
# A算法

- 在图搜索算法中，如果能在搜索的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，则该搜索算法为A算法。由于估价函数中带有问题自身的启发性信息，因此，A算法又称为启发式搜索算法。
- 对启发式搜索算法，又可根据搜索过程中选择扩展节点的范围，将其分为全局择优搜索算法和局部择优搜索算法。

# A算法

## □ 估价函数

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



从起始状态到当前状态 $x$ 的代价

从当前状态 $x$ 到目标状态的估计代价 (启发函数)

$g(x)$  有利于搜索的完备性，但影响搜索的效率。  
 $h(x)$  有利于提高搜索的效率，但影响搜索的完备性。

# 启发函数示例

设有如下结构的移动将牌游戏：

B	B	B	W	W	W	E
---	---	---	---	---	---	---

该游戏规则：

1. 当一个牌移入相邻的空位置时，费用为一个单位。
2. 一个牌至多可跳过两个牌进入空位置，其费用等于跳过的牌数加1。

要求：把所有的B都移至W的右边，请设计启发函数 $h(x)$ 。

解：根据要求可知，W左边的B越少越接近目标，因此可用W左边B的个数作为 $h(x)$ ，即

$$h(x) = 3 \times (\text{每个W左边B的个数的总和})$$

这里乘以系数3是为了扩大 $h(x)$ 在 $f(x)$ 中的比重。

# 局部择优搜索

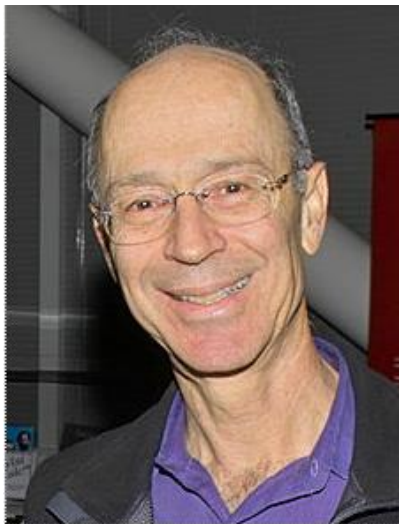
- 基本思想：

当一个节点被扩展以后，按 $f(x)$ 对每一个子节点计算估计价值，并选择最小者作为下一个要考察的节点。
- 搜索过程：
  1. 把初始节点 $S_0$ 放入OPEN表，计算 $f(S_0)$ 。
  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CLOSE表。
  4. 考察节点 $n$ 是否为目标节点。若是，则求得了问题的解，退出。
  5. 若节点 $n$ 不可扩展，则转第2步。
  6. 扩展节点 $n$ ，用估价函数 $f(x)$ 计算每个子节点的估计价值，并按估计价值从小到大的顺序放到OPEN表中的首部，并为每一个子节点都配置指向父节点的指针，然后转第2步。
- 深度优先搜索、代价树的深度优先搜索均为局部择优搜索的特例。

# 全局择优搜索

- 基本思想：  
每当要选择下一个节点进行考察时，全局择优搜索每次总是从OPEN表的全体节点中选择一个估价值最小的节点。
- 搜索过程：
  1. 把初始节点 $S_0$ 放入OPEN表，计算 $f(S_0)$ 。
  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CLOSE表。
  4. 考察节点 $n$ 是否为目标节点。若是，则求得了问题的解，退出。
  5. 若节点 $n$ 不可扩展，则转第2步。
  6. 扩展节点 $n$ ，用估价函数 $f(x)$ 计算每个子节点的估价值，并为每一个子节点都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至大的顺序进行排序，然后转第2步。
- ◆ 广度优先搜索、代价树的广度优先搜索是全局择优搜索的特例。

## 6.2.8 A\*算法



彼得.哈特

- 1968年，彼得.哈特对A算法进行了很小的修改，并证明了当估价函数满足一定的限制条件时，算法一定可以找到最优解
- 估价函数满足一定限制条件的算法称为A\*算法

### A\*算法的限制条件

$$f(x) = \underbrace{g(x)} + \underbrace{h(x)}$$

大于0

不大于 $x$ 到目标的实际代价



# 利用A\*算法求解八数码问题

## ■ 估价函数的定义

□  $f(x) = g(x) + h(x)$

□  $g(x)$ : 从初始状态到 $x$ 需要进行的移动操作的次数

□  $h(x)$ :  $x$ 状态下错放的棋子数  
满足限制条件

2	8	3
7	1	4
	6	5

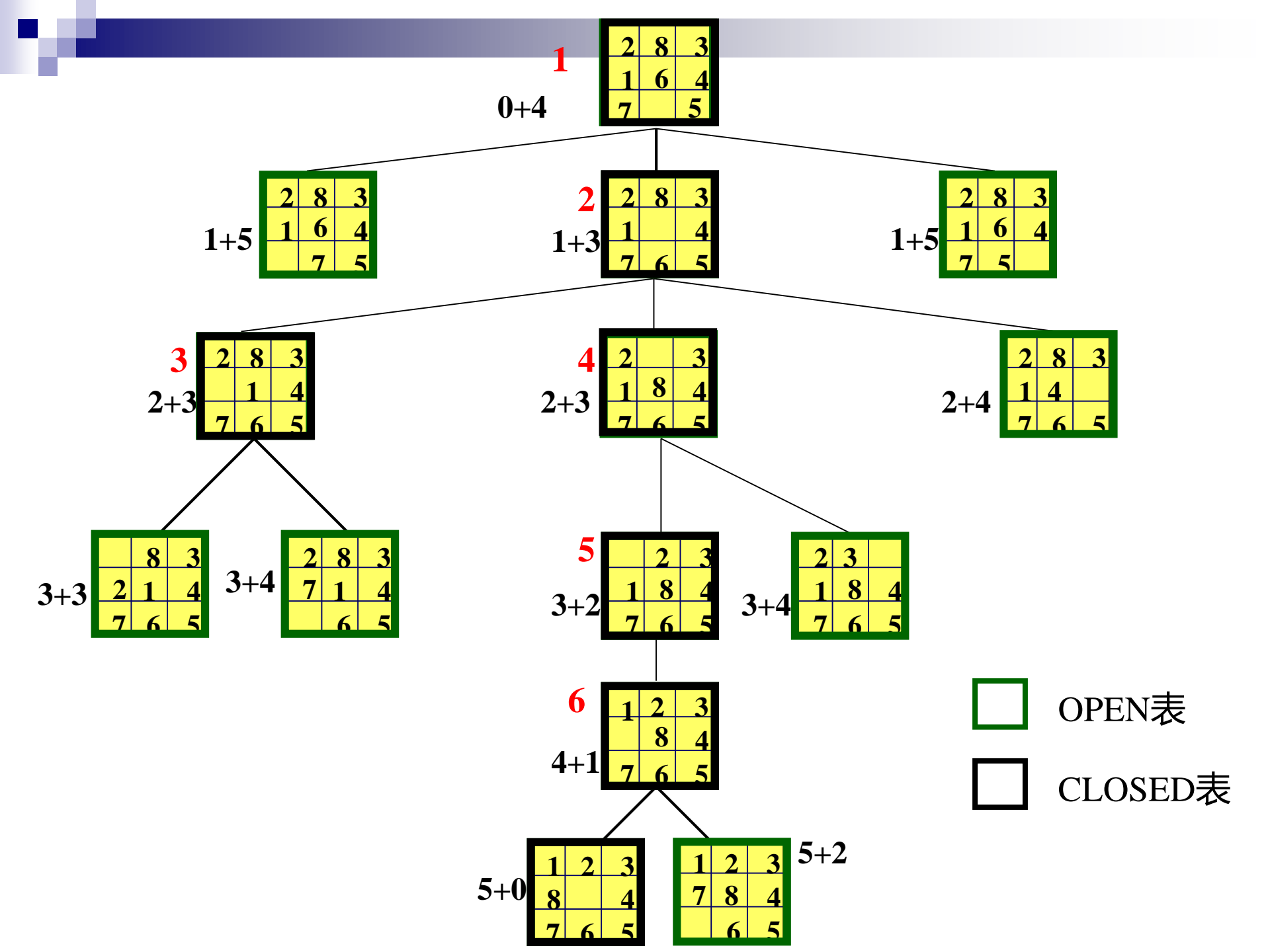
$h(x)=4$

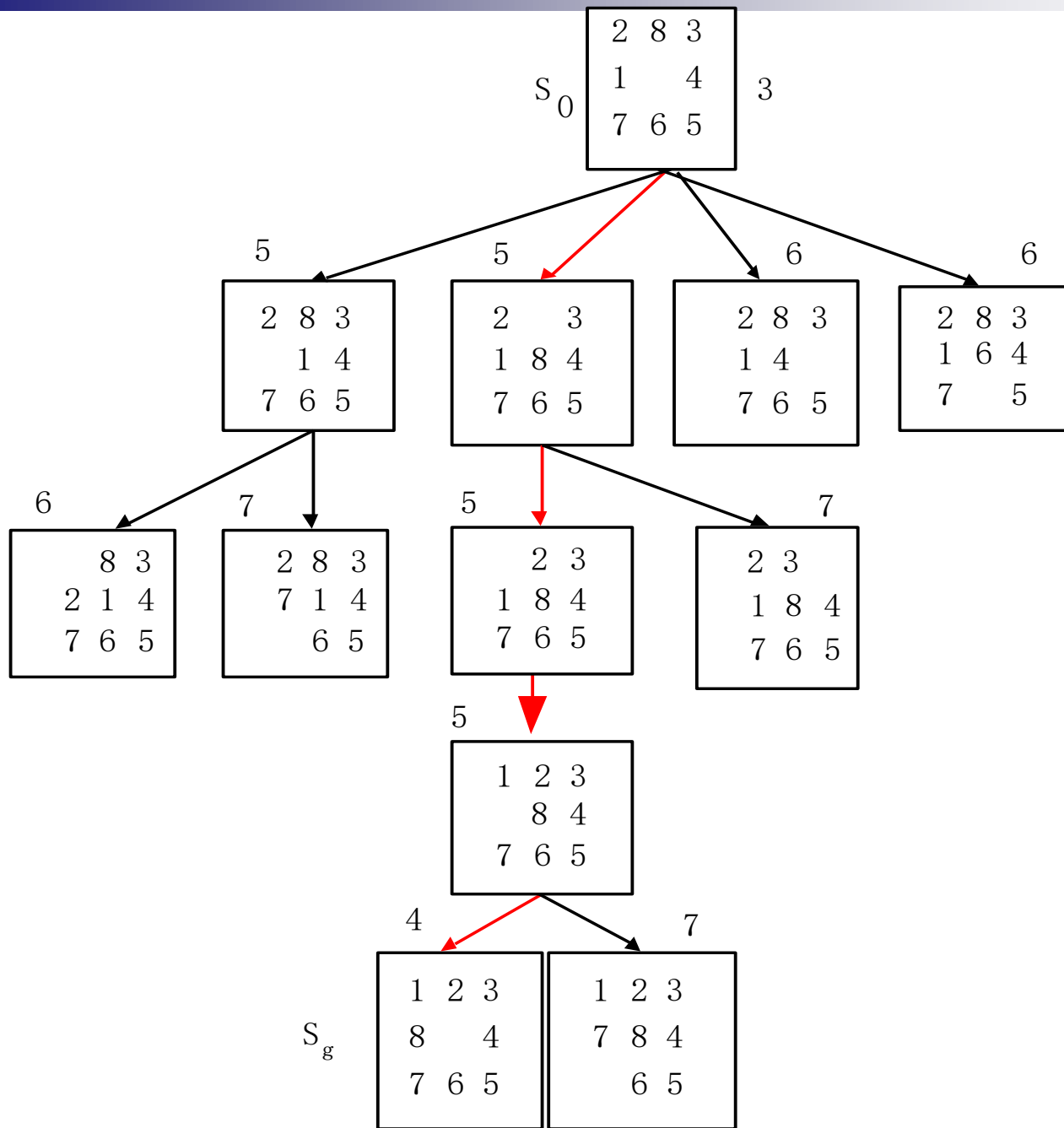
1	2	3
7	8	4
	6	5

$h(x)=2$

1	2	3
	8	4
7	6	5

$h(x)=1$





# 估价函数对算法的影响

- 不同的估价函数对算法的效率可能产生极大的影响
- 不同的估价函数甚至产生不同的算法

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

- $h(x)=0$ : Dijkstra算法, 非启发式算法
- $g(x)=0$ : 贪婪搜索, 无法保证找到解
- 即使采用同样的形式  $f(x) = g(x) + h(x)$ , 不同的定义和不同的值也会影响搜索过程

# 估价函数对算法的影响示例

## ■ 例：八数码问题

□  $f(x) = g(x) + h(x)$

□  $g(x)$ : 从初始状态到 $x$ 需要进行的移动操作的次数

□  $h(x)$ : 所有棋子与目标位置的曼哈顿距离之和

■ 曼哈顿距离：两点之间水平距离和垂直距离之和

■ 仍满足估价函数的限制条件

2	8	3
7	1	4
	6	5

$$h(x) = 2 + 1 + 1 + 2 = 6$$

**1**  
**0+5**

2	8	3
1	6	4
7		5

**1+6**

2	8	3
1	6	4
	7	5

**2**  
**1+4**

2	8	3
1		4
7	6	5

**1+6**

2	8	3
1	6	4
7	5	

**2+5**

2	8	3
	1	4
7	6	5

**3**  
**2+3**

2		3
1	8	4
7	6	5

**2+5**

2	8	3
1	4	
7	6	5

**4**  
**3+2**

	2	3
1	8	4
7	6	5

**3+4**

2	3	
1	8	4
7	6	5

**5**  
**4+1**

1	2	3
	8	4
7	6	5

**5+0**

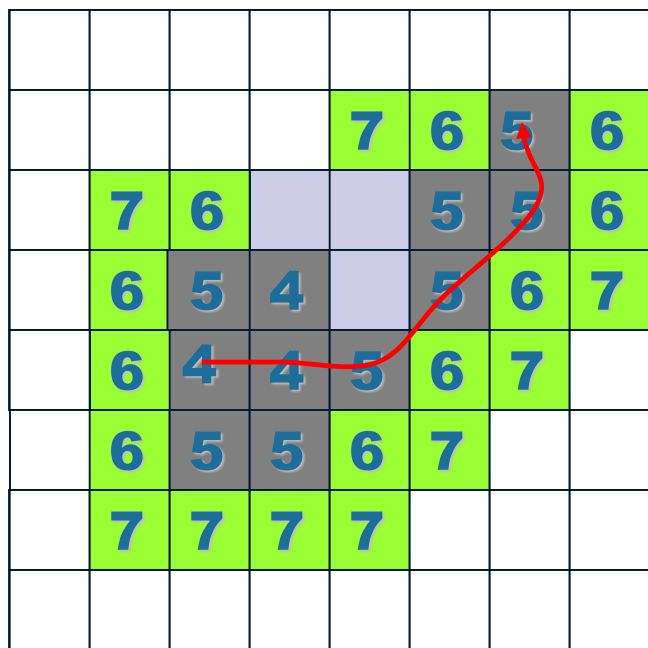
1	2	3
8		4
7	6	5

**5+2**

1	2	3
7	8	4
	6	5

# 例：路径规划

- A\*算法被广泛应用于静态路网中的最短路径规划



用距离表示代价

每一步可以往相邻的8个无障碍格子移动，移动距离为1

$g(x)$ : 从出发点S到当前点 $x$ 的距离

$h(x)$ : 从当前点 $x$ 到目标点G的距离

## 6.3 与/或树的搜索策略

6.3.1 与/或树的一般搜索过程

6.3.2 与/或树的广度优先搜索

6.3.3 与/或树的深度优先搜索

6.3.4 与/或树的有序搜索

6.3.5 博弈树的启发式搜索

6.3.6  $\alpha$ - $\beta$ 剪枝技术



## 6.4 搜索的完备性与效率

### 6.4.1 完备性

- 对于一类可解的问题和一个搜索过程，如果运用该搜索过程一定能求得该类问题的解，则称该搜索过程为完备的，否则为不完备的。
- 完备的搜索过程称为“搜索算法”。不完备的搜索过程不是算法，称为“过程”。
- 广度优先搜索、代价树的广度优先搜索、改进后的有界深度优先搜索以及A\*算法都是完备的搜索过程，其它搜索过程都是不完备的。

## 6.4.2 搜索效率

### ■ 外显率

外显率定义为

$$P=L/T$$

其中， $L$ 为从初始节点到目标节点的路径长度； $T$ 为整个搜索过程中所生成的节点总数。

- 外显率反映了搜索过程中从初始节点向目标节点前进时搜索区域的宽度。当 $T=L$ 时， $P=1$ ，表示搜索过程中每次只生成一个节点，它恰好是解路径上的节点，搜索效率最高。 $P$ 越小表示搜索时产生的无用节点愈多，搜索效率愈低。

# 有效分枝因数

- 有效分枝因数**B**定义为

$$B+B^2+\dots+B^L=T$$

其中，**B**是有效分枝因数，它表示在整个搜索过程中每个节点平均生成的子节点数目；**L**为路径长度；**T**为节点总数。

- 当**B=1**时，**L=T**，此时所生成的节点数最少，搜索效率最高。
- 不难证明，有效分枝因数与外显率之间由如下关系

$$P=L/T$$

$$T=B \times (B^L-1)/(B-1)$$

$$P=(L \times (B-1))/(B \times (B^L-1))$$

# 衡量搜索策略性能的准则

- 1、完备性
  - 2、尽量避免无用搜索。增强搜索的目的性，尽量避免产生及考察那些无用的节点。
  - 3、控制开销小。要求搜索策略实现简单。
- 显然以上准则很难同时满足。所以，在这些准则之间可以采取折衷的方法，从而使其综合效果比较好。



完

谢谢