人工智能

Artificial Intelligence

主讲：相明

西安交通大学电信学院计算机系

E_mail：mxiang@mail.xjtu.edu.cn

第五章 搜索策略

- 5.1 基本概念
- 5.2 状态空间的搜索策略
- 5.3 与/或树的搜索策略

5.1 基本概念

5.1.1 什么是搜索

- 采用某种策略，在知识库中寻找可利用的知识，从而构造一条代价较小的推理路线，使问题得到解决的过程称为搜索。
- 搜索分为盲目搜索和启发式搜索。
  - 盲目搜索是按照预定的控制策略进行搜索，在搜索过程中获得的中间信息不用来改进控制策略。
  - 启发式搜索是在搜索中加入了与问题有关的启发性信息，用以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。

5.1.2 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：

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

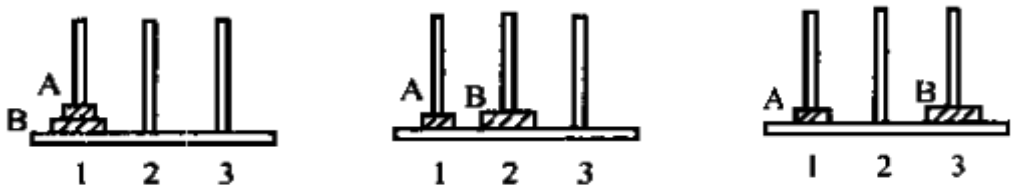
- 算符

  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。

例5.1　二阶梵塔问题。设有三根钢针，在1号钢针上穿有A、B两个金片，A小于B，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片，任何时刻都不能使B位于A的上面。



（1）设用$S_K=(S_{k0},S_{k1})$表示问题的状态（第k个状态），S_{K0}表示金片A所在的钢针号，S_{k1}表示金片B所在的钢针柱号，全部可能的状态有九种：

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

$S_3=(2,1)$, $S_4=(2,2)$, $S_5=(2,3)$

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

问题的初始状态集合为$S=\{S_0\}$,目标状态集合为$G=\{S_4,S_8\}$。

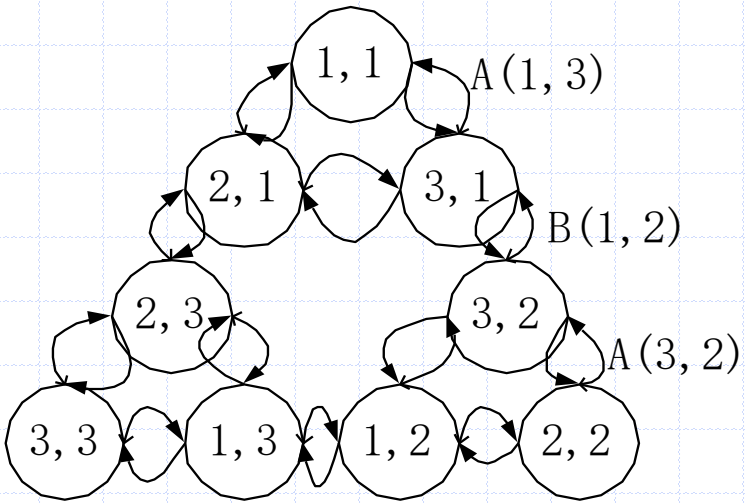
（2）算符：A(i,j)及B(i,j)。A(i,j)表示把A金片从第i号钢针移到第j号钢针。B(i,j)与之同理。算符共有12个。

（3）在状态空间图中，从初始节点(1,1)到目标节点(2,2)或(3,3)的任何一条通路都是问题的一个解。

其中最短的路径长度是3，

它由3个算符组成。

例如:A(1,3),B(1,2),A(3,2)



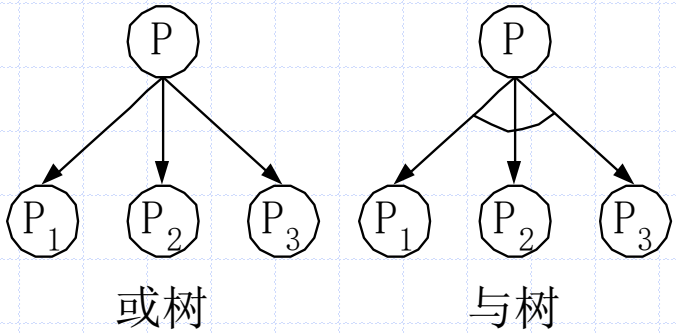
- 采用状态空间表示方法，首先要把问题的一切状态都表示出来，其次要定义一组算符。
- 问题的求解过程是一个不断把算符作用于状态的过程。如果在使用某个算符后得到的新状态是目标状态，就得到了问题的一个解。这个解就是从初始状态到目标状态所采用算符的序列。使用算符最少的解称为最优解。

5.1.3 与/或树表示法

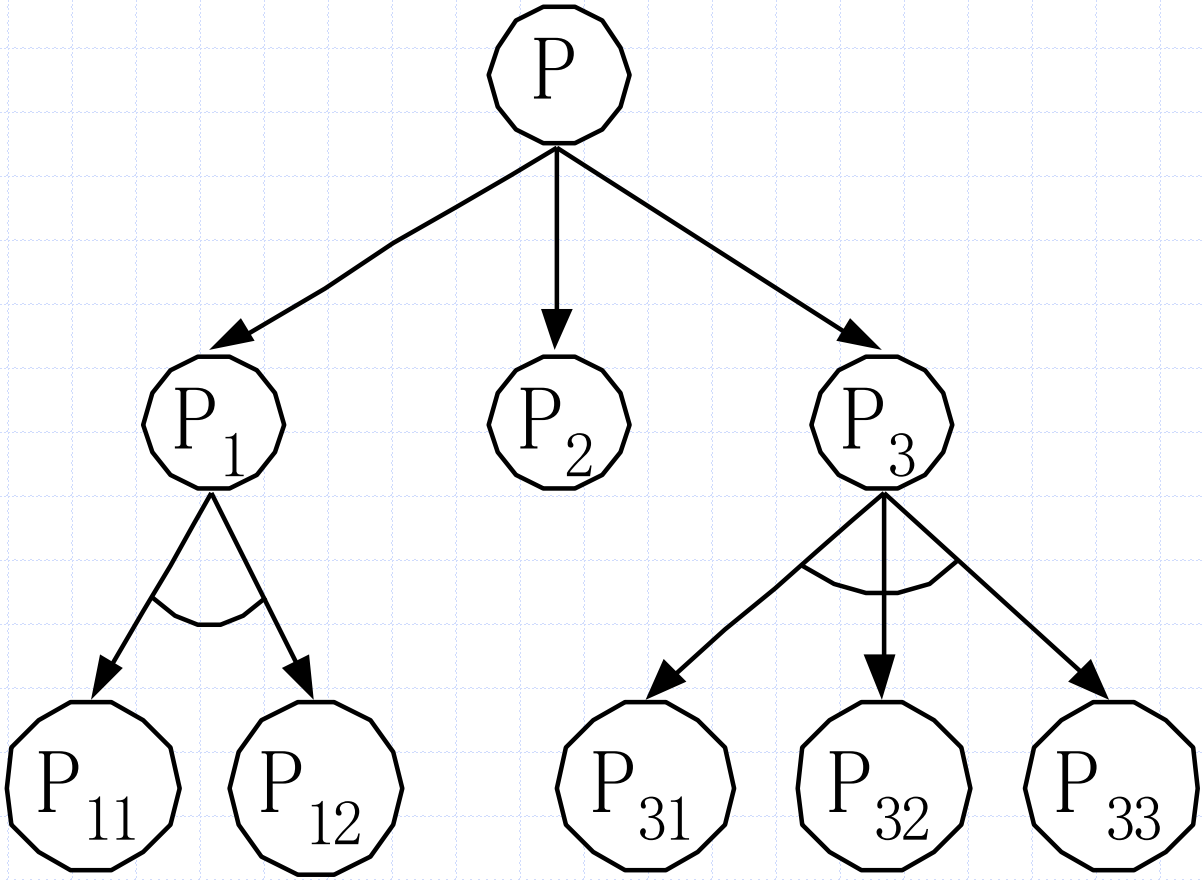
- 与/或树是用于表示问题及其求解过程的另一种方法，通常用于表示比较复杂问题的求解。
- 对于一个复杂问题，直接求解往往比较困难。此时可通过下述方法进行简化：
  - 分解

    把一个复杂问题分解为若干个较为简单的子问题，每个子问题又可继续分解。重复此过程，直到不需要再分解或者不能再分解为止。如此形成“与”树。
  - 等价变换

    利用同构或同态的等价变换，把原问题变换为若干个较为容易求解的新问题。如此形成“或”树。



与/或树



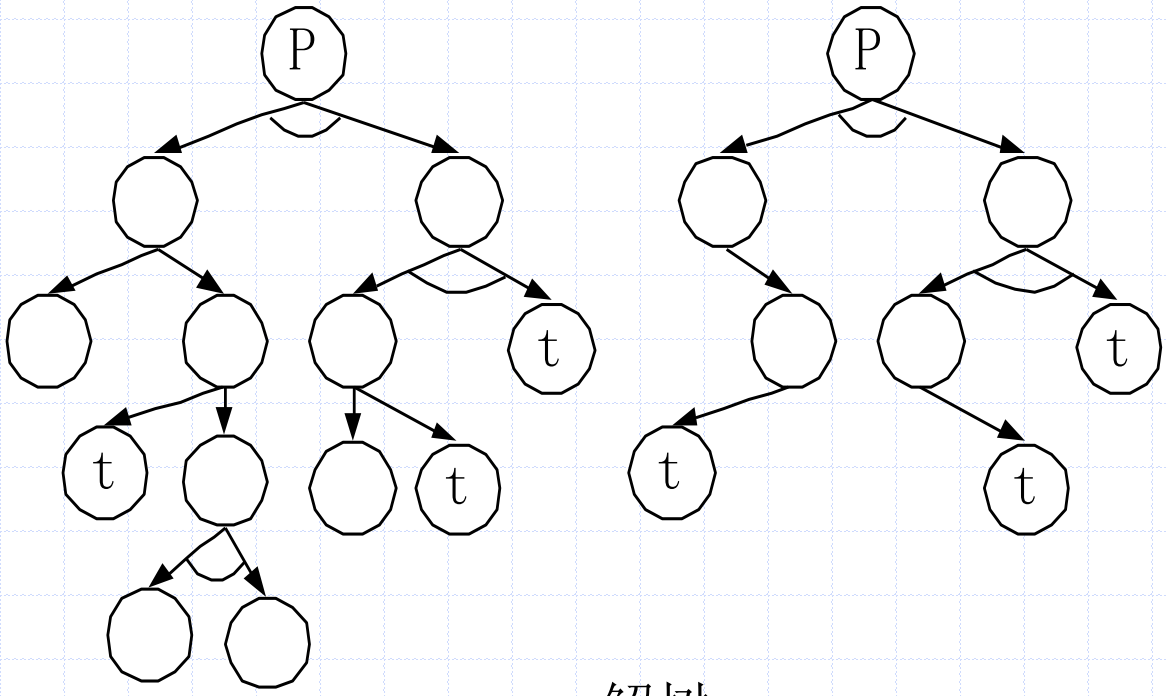
与/或树

一些基本概念

- 本原问题
  - 不能再分解或变换，而且直接可解的子问题。
- 端节点与终止节点
  - 在与/或树中，没有子节点的节点统称为端节点；本原问题所对应的节点称为终止节点。
- 可解节点
  - 在与/或树中，满足下列条件之一者，称为可解节点：
    - 它是一个终止节点；
    - 它是一个“或”节点，且其子节点中至少有一个是可解节点；
    - 它是一个“与”节点，且其子节点全部是可解节点。
- 不可解节点
  - 关于可解节点的三个条件全部不满足的节点

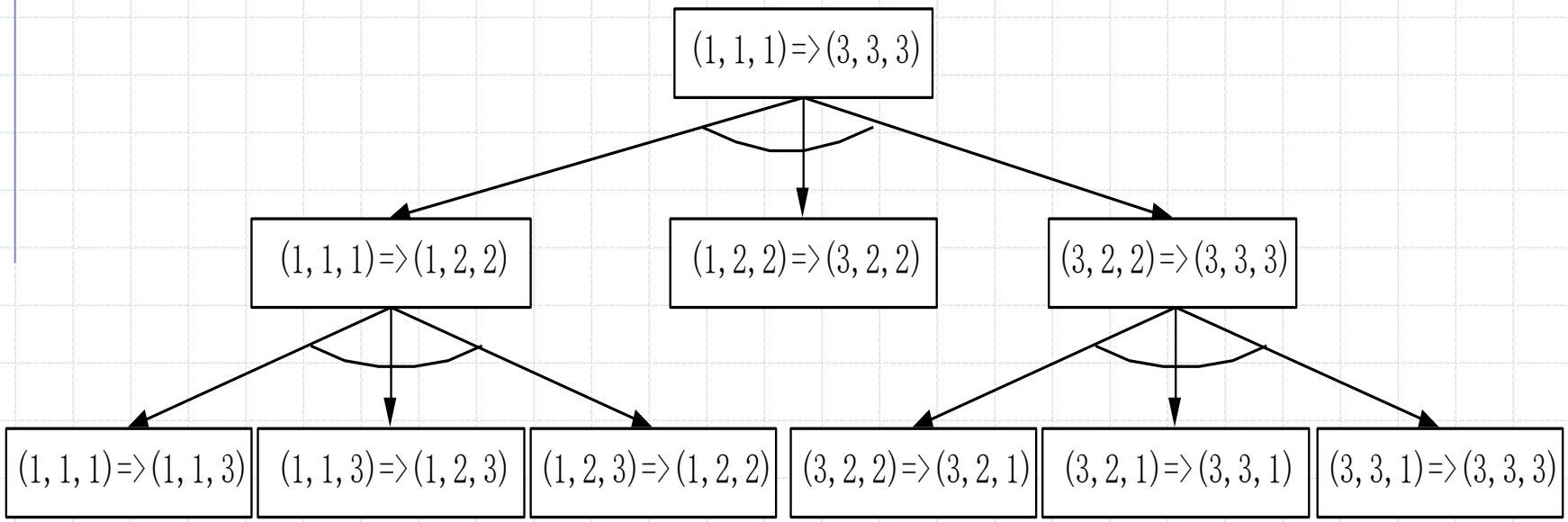
解树

由可解节点所构成，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(t表示终止节点)。

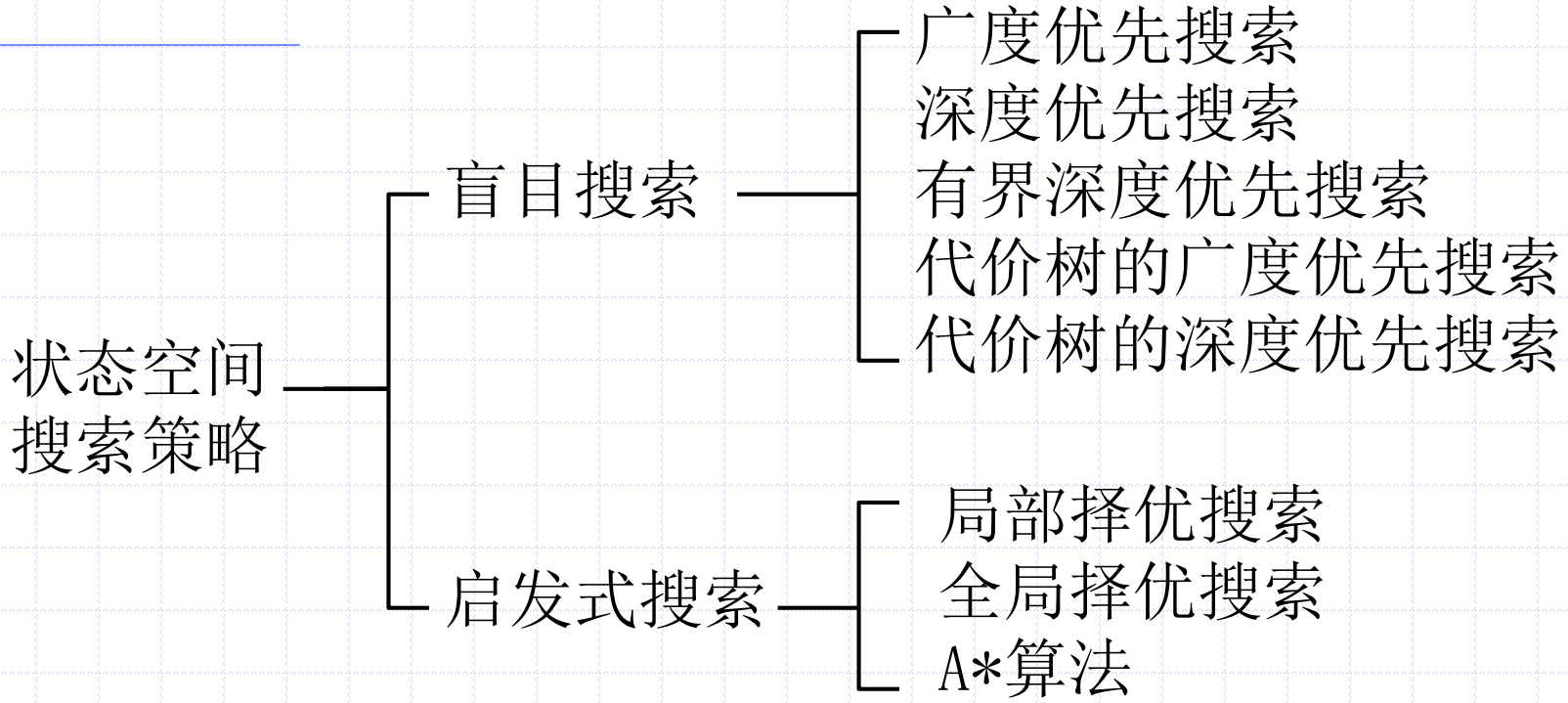


解树

三阶梵塔问题的与/或树


(1, 1, 1)=>(3, 3, 3)
(1, 1, 1)=>(1, 2, 2)
(1, 2, 2)=>(3, 2, 2)
(3, 2, 2)=>(3, 3, 3)
(1, 1, 1)=>(1, 1, 3)
(1, 1, 3)=>(1, 2, 3)
(1, 2, 3)=>(1, 2, 2)
(3, 2, 2)=>(3, 2, 1)
(3, 2, 1)=>(3, 3, 1)
(3, 3, 1)=>(3, 3, 3)

5.2 状态空间的搜索策略



- 盲目搜索的特点：
  - 搜索按规定的路线进行，不使用与问题有关的启发性信息；
  - 适用于其状态空间图是树状结构的一类问题。(同样适用于一般的图)
- 启发式搜索要使用与问题有关的启发性信息，并以这些启发性信息指导搜索过程，可以高效地求解结构复杂的问题。

5.2.1 状态空间的一般搜索过程

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（对节点扩展前要先将它放入close表）。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，并记该节点为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. 针对M中子节点的不同情况，分别进行如下处理：（动态更新open表及图G）
   1. 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；（子节点不在OPEN表）
   2. 对于那些先前已经在G中出现过的M成员，确定是否需要修改它指向父节点的指针；（子节点在OPEN表中，或在CLOSE表中）
   3. 对于那些先前已在G中出现并且已经扩展了的M成员，确定是否需要修改其后继节点指向父节点的指针；（子节点在CLOSE表中）
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 初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。
- 通过搜索所得到的图称为搜索图。由搜索图中所有的节点及反向指针（指向父节点的指针）所构成的集合是一棵树，称为搜索树。
- 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的路径上的算符构成。
- 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。

5.2.2 广度优先搜索

- 基本思想：

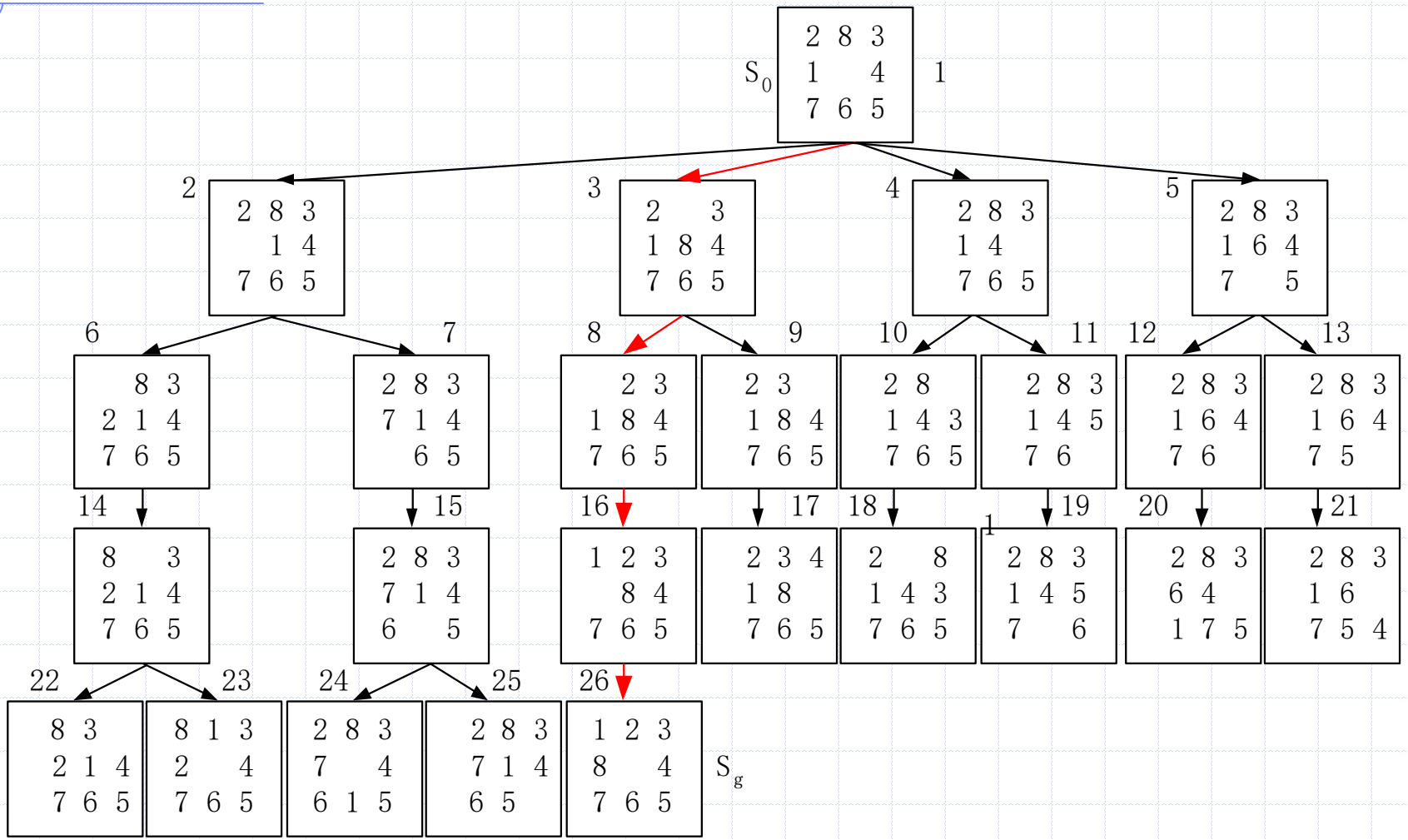
  从初始节点S_0开始，逐层地对节点进行扩展，并考察它是否为目标节点。在第n层的节点没有全部扩展并考察之前，不对第n＋1层的节点进行扩展。
- OPEN表中节点总是按进入的先后顺序排列，先进入的节点排在前面，后进入的排在后面。

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表的第一个节点取出放入CLOSE表（记为节点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. 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。(只考虑有效子节点，见一般搜索过程)

重排九宫的广度优先搜索

操作符：空格左移、上移、右移、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）。

3、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（完备性）。

- 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。

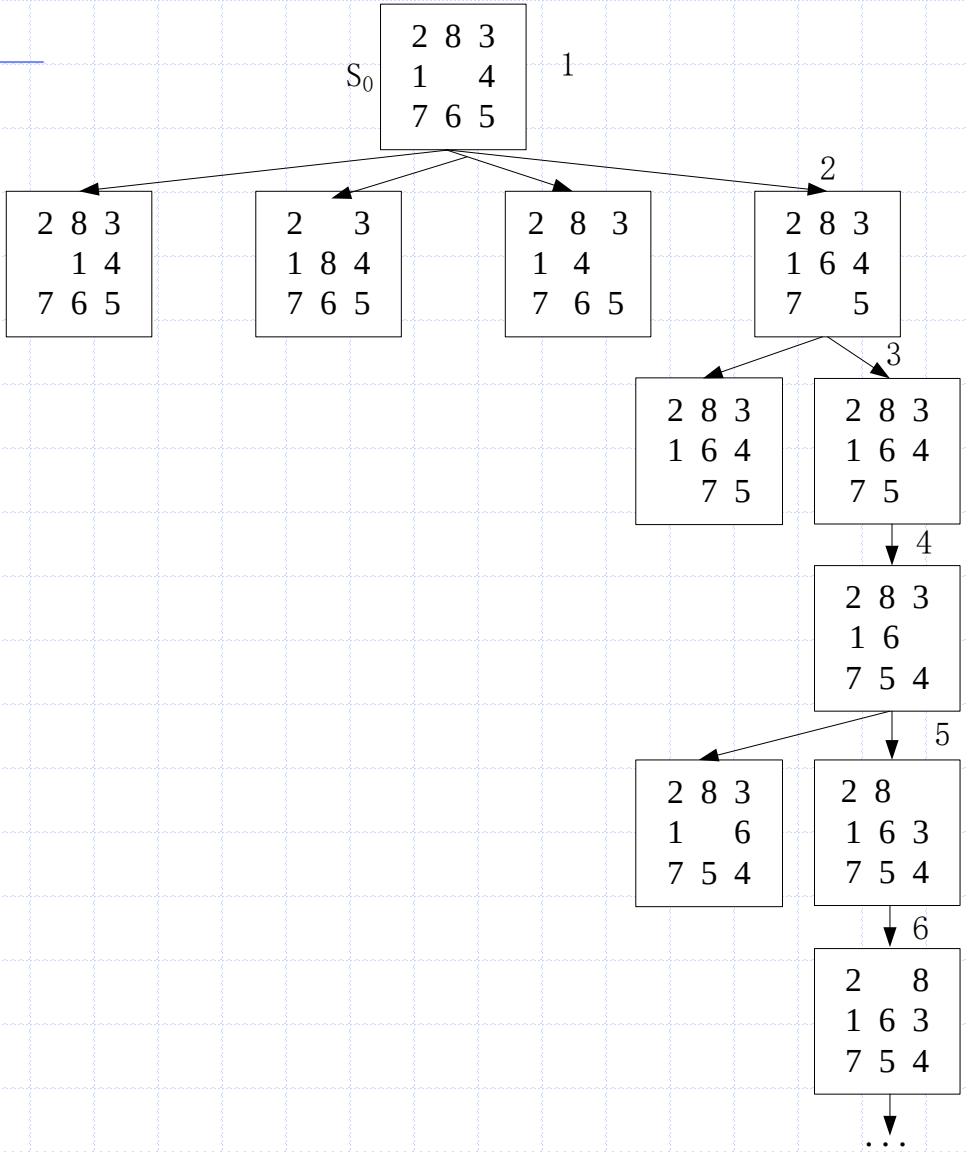
5.2.3 深度优先搜索

- 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，而且即使问题有解，它也不一定能求得解。
- 用深度优先求得的解，不一定是路径最短的解。
- 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。

5.2.4 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表的第一个节点取出放入CLOSE表（记为节点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. 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。

（略）

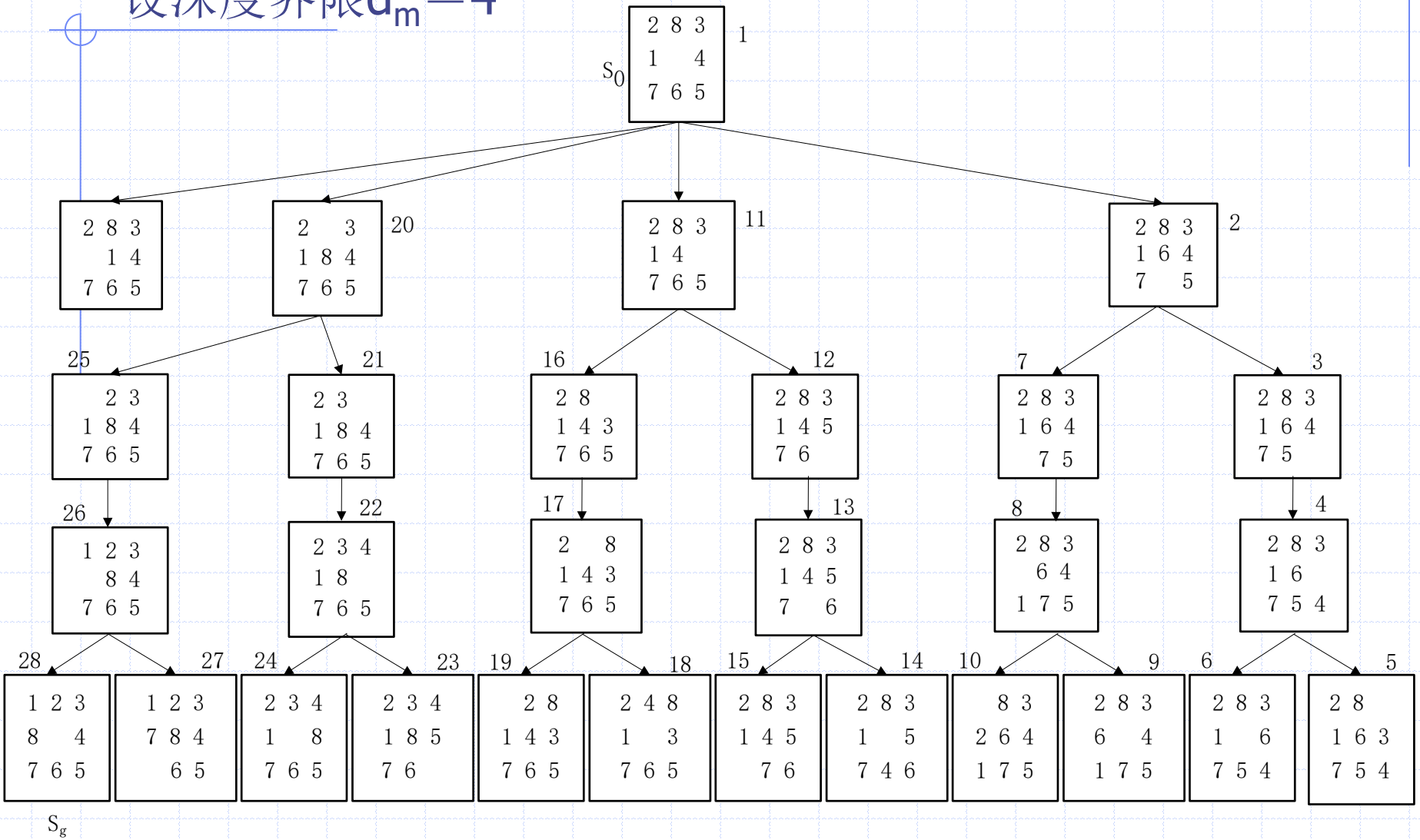
- 如果问题有解，且其路径长度≤d_m，则上述搜索过程一定能求得解。但是，若解的路径长度>d_m，则上述搜索过程就得不到解。
- 要恰当地给出d_m的值是比较困难的。
- 即使能求出解，它也不一定是最优解。

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R，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。由于后面求得的解的路径长度不会超过先前求得的解的路径长度，所以后面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$d_m=4$



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(即该算法是完备的)。

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。

5.2.5 启发式搜索

- 盲目搜索具有较大的盲目性，产生的无用节点较多，效率不高。
- 启发式搜索采用问题自身的特性信息，以指导搜索朝着最有希望的方向前进。这种搜索针对性较强，因而效率较高。

1、启发性信息与估价函数

- 可用于指导搜索过程，且与具体问题有关的信息称为启发性信息。
- 用于评估节点重要性的函数称为估价函数。其一般形式为：

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

- 其中g(x)表示从初始节点S_0到节点x的代价；h(x)是从节点x到目标节点S_g的最优路径的代价的估计，它体现了问题的启发性信息。h(x)称为启发函数。
- 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×(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。

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

2、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价值，并按估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。

3、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。

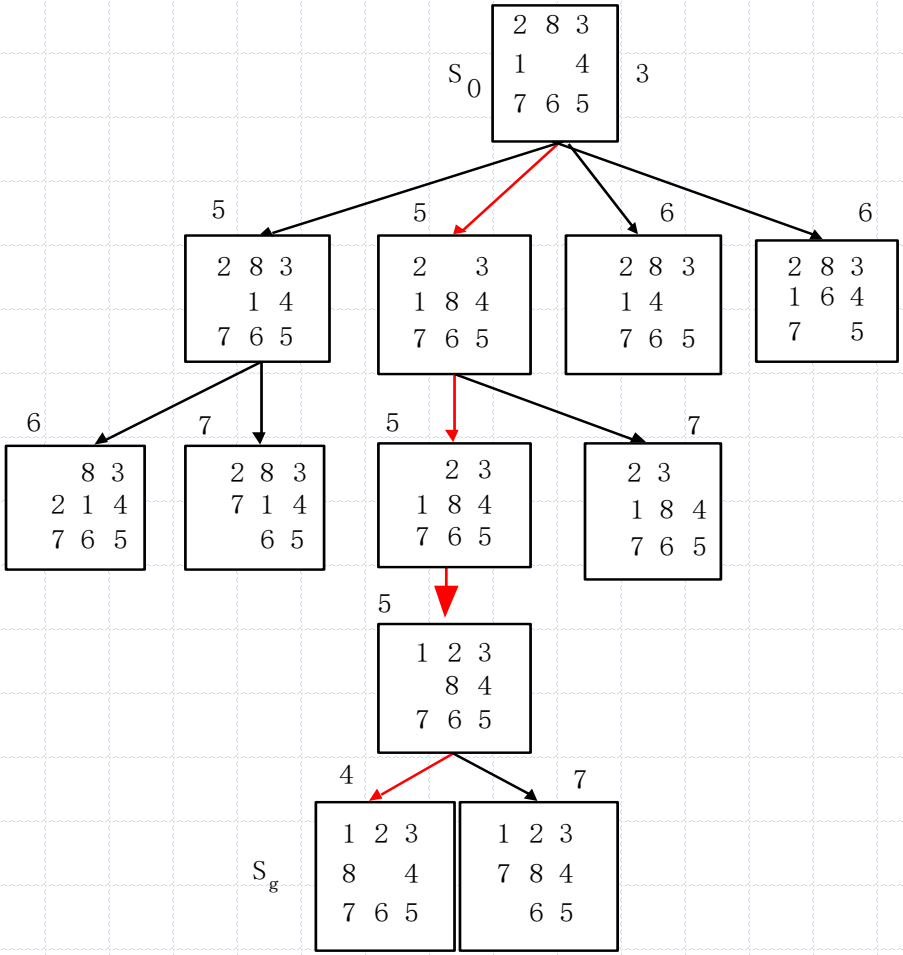
- 搜索过程：

1. 把初始节点S_0放入OPEN表，计算$f(S_0)$。　　$f(x) = g(x)+h(x)$
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，根据估价函数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。

重排九宫问题的全局择优搜索树

设估价函数为：f(x)=g(x)+h(x)=d(x)+h(x)，其中d(x)表示节点x的深度，起始节点S0的深度为0，h(x)表示节点x的格局与目标节点格局不相同的牌数。



5.2.6 A*算法

如果5.2.1中给出的一般搜索过程满足如下限制，则它就称为A*算法：

1、把OPEN表中的节点按估价函数

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

的值从小至大进行排序（一般搜索过程的第7步）。

2、 g(x)是从初始节点S_0到节点x的路径的代价， g(x)是对g*(x)的估计。

3、h(x)是h*(x)的下界，即对所有的x均有：

$$h(x)\leq h^*(x)$$

其中，g*(x)是从初始节点S_0到节点x的最小代价；h*(x)是从节点x到目标节点的最小代价。

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. 针对M中子节点的不同情况，分别进行如下处理：
    1. 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）
    2. 对于那些先前已经在G中出现过的M成员，确定是否需要修改它指向父节点的指针；（在OPEN表或CLOSE表中,对g(x)进行更新）
    3. 对于那些先前已在G中出现并且已经扩展了的M成员，确定是否需要修改其后继节点指向父节点的指针；（在CLOSE表中, 对节点n子节点的子节点更新g(x) ）
7. 对OPEN表中的节点按估价函数f(x)=g(x)+h(x)进行排序；
8. 转第2步。

- 在A*算法中，g(x) 是从初始节点S_0到节点x的路径的代价，恒有$g(x) \geq g^*(x)$。而且在算法执行过程中随着更多搜索信息的获得，g(x)的值呈下降的趋势。

- h(x)的设计依赖于具体问题领域的启发性信息。$h(x) \leq h^*(x)$限制了启发式信息在估价函数中的比重，有利于算法的稳定性，保证了A*算法总能找到最优解。

1、A*算法的可纳性

⑩ A*算法的可纳性

对于可解状态空间图(即从初始节点到目标节点有路径存在)，如果一个搜索算法能在有限步内终止，并且能找到最优解，则称该搜索算法是可纳的。A*算法是可纳的。

证明步骤：

（1）对于有限图，A*算法一定会在有限步内终止。

（2）对于无限图，只要从初始节点到目标节点有路径存在，则A*算法也必然会终止。

（3）A*算法一定终止在最优路径上。

2、A*算法的最优性

A*算法的搜索效率在很大程度上取决于h(x)，在满足$h(x) \leq h^*(x)$的条件下，h(x)的值越大越好。h(x)的值越大，表明它携带的启发性信息越多，搜索时扩展的节点数越少，搜索的效率越高。

3、h(x)的单调性

在A*算法中，每当扩展一个节点(节点n)时都要检查其子节点是否在OPEN表或CLOSE表中，如果在，则要调整子节点指向父节点的指针，这就增加了搜索的代价。如果对启发函数h(x)加上单调性限制，就可免除大部分上述检查及调整工作，从而减少搜索代价。

- 所谓单调性限制是指h(x)满足如下两个条件：

1、$h(S_g)=0$;

2、设x_j是节点x_i的子节点，则有

$$h(x_i)\leq h(x_j)+c(x_i,x_j)$$

其中，S_g是目标节点；$c(x_i,x_j)$是节点x_i到其子节点x_j的代价。

- 可以证明，当A*算法的启发函数h(x)满足单调性限制时，可得到如下两个结论：

1、若A*算法选择节点x_n进行扩展，则

$$g(x_n)=g^*(x_n)$$ (由于该性质故可不作有些子节点指针的更新)

2、由A*算法所扩展的节点序列其f值是非递减的（递增的）。

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

完

谢谢