人工智能概论-确定性推理

赵亚伟

zhaoyw@ucas.ac.cn

中国科学院大学 大数据分析技术实验室

2018.5.26

问题：

- 知识表示有那些方法?
- 知识表示的目的是什么?
- 构建智能系统的关键是什么?

目录

- 图搜索策略
- 盲目搜索
- 启发式搜索
- 消解原理
- 规则演绎系统
- 产生式系统
- 非单调推理
- 小结

图搜索策略

- **图搜索控制策略: 一种在图中寻找路径的方法。**
  - 图中每个节点对应一个状态;
  - 每条连线对应一个操作符。

- **用产生式系统的数据库和规则来标记:**
  - 初始节点——初始数据库;
  - 目标节点——目标数据库;
  - 状态图的一条路径问题——求得把一个数据库变换为另一数据库的规则序列问题。

图搜索的一般过程

图搜索过程（GraphSearch）

- 1）建立一个只含有起始节点S的搜索图G，把S放到一个叫做OPEN 的**未扩展节点表**中。
- 2）建立一个叫做CLOSED的**已扩展节点表**，其初始为空表.
- 3）LOOP：若OPEN表是空表，则失败退出。
- 4）选择OPEN表上的第一个节点，把它从OPEN表移出并放进CLOSED表中。称此节点为节点n
- 5）若n为一*目标节点*，则有解并成功退出，此解是追踪图G中沿着指针**从n到S这条路径**而得到的(指针将在第7步中设置)。

图搜索的一般过程Cont.

- 6）扩展节点n，同时生成不是n的祖先的那些后继节点的集合M。把M的这些成员作为n的后继节点添入图G中。

没说如何将M加进OPEN表中，放末端还是前端

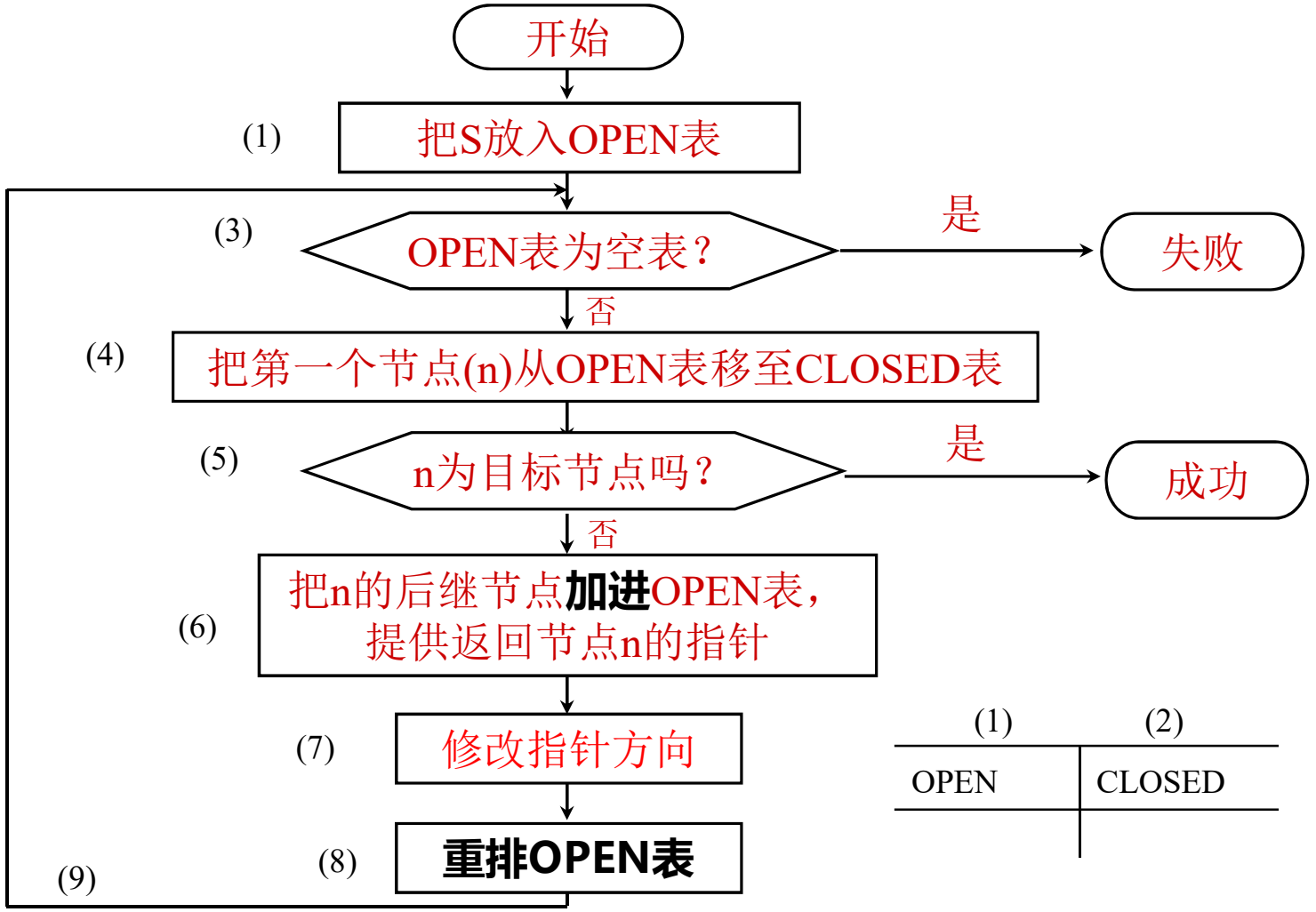
- 7）对那些未曾在G中出现过的M成员设置一个通向n的指针。把M的这些成员**加进**OPEN表。对已经在OPEN或CLOSED表上的每一个M成员，确定是否需更改通到n的指针方向。对已在CLOSED表上的每个M成员，确定是否需要更改图G中通向它的每个后裔节点的指针方向。

- 8）按某一任意方式或按某个探试值，**重排OPEN表**。

没说明如何重排OPEN表，按队列还是按栈重排?

- 9）GO LOOP。

图搜索过程框图（图搜索元算法）



图搜索结果

- **图搜索的生成结果：**
  - 搜索图（G）
  - 搜索树（T）　（本质上也是图 G）
- **修正算法：**
  - 一次只生成一个后继节点；
- **思考：**
  - （1）结果路径的形成中，为什么其节点顺序是明确的？
  - （2）OPEN表中的节点具有什么特点？
  - （3）CLOSED表中的节点具有什么特点？
  - （4）对OPEN表节点的排序有何意义？
- **提出：盲目搜索与启发式搜索。**

目录

- 图搜索策略
- 盲目搜索
- 启发式搜索
- 消解原理
- 规则演绎系统
- 产生式系统
- 非单调推理
- 小结

盲目搜索

- **盲目搜索又叫做无信息搜索，一般只适用于求解比较简单的问题。**
  - 特点：不需重排OPEN表；
  - 种类：宽度优先、深度优先、等代价搜索等。
- **宽度优先搜索（Breadth-first）**
  - 定义：以接近起始节点的程度逐层扩展节点的搜索方法。
- **特点：**
  - 搜索代价高，但若有解存在，则一定可以找到。

宽度优先搜索算法

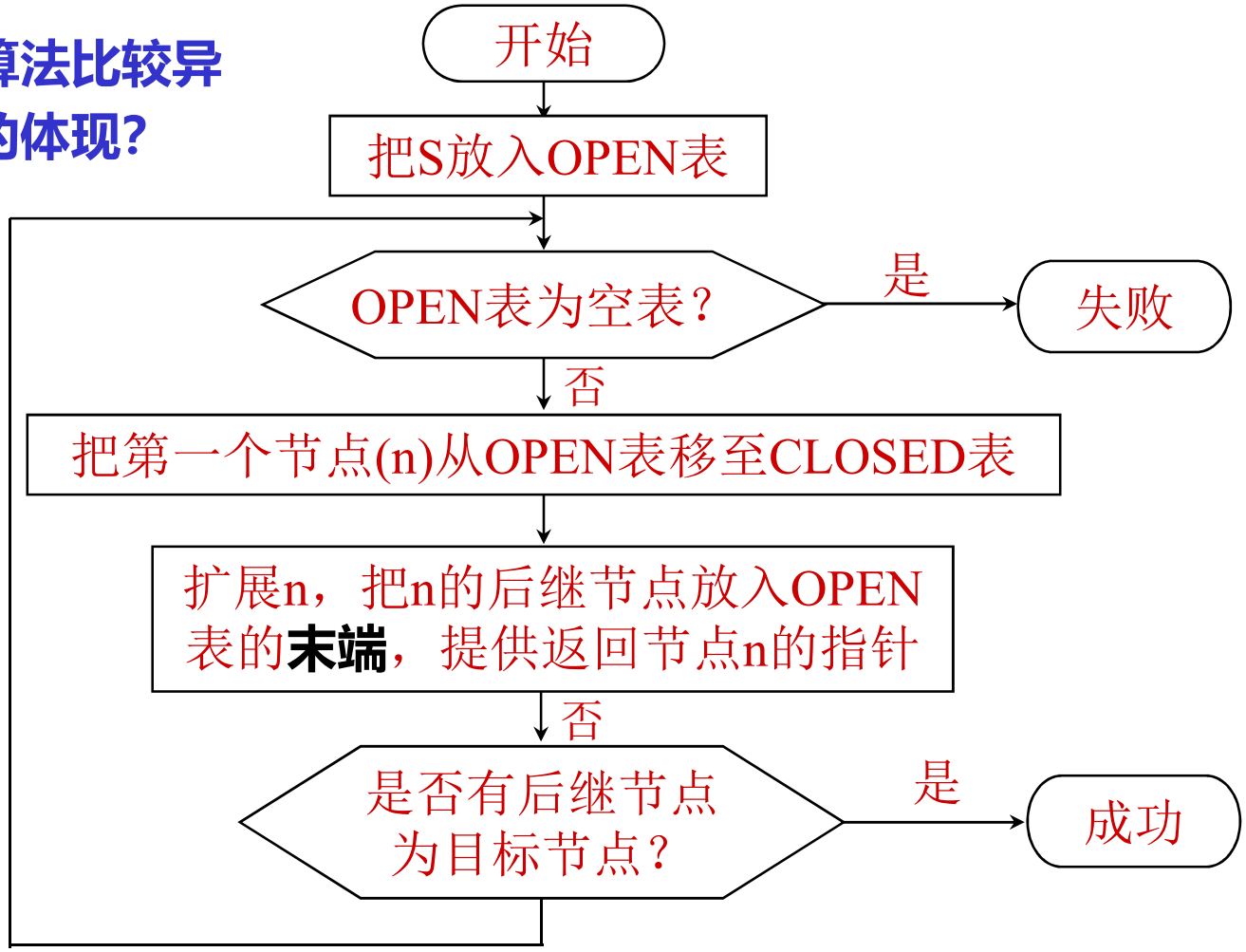
- 1）把起始节点放到OPEN表中(如果该起始节点为一目标节点，则求得一个解答)。
- 2）如果OPEN是个空表，则没有解，失败退出；否则继续。
- 3）把**第一个**节点(节点n)从OPEN表移出，并把它放入CLOSED的扩展节点表中。
- 4）扩展节点n。如果没有后继节点，则转向上述第(2)步。
- 5）把n的所有后继节点放到OPEN表的**末端**，并提供从这些后继节点回到n的指针。

放在末端，宽度优先搜索，形成队列，先进先出

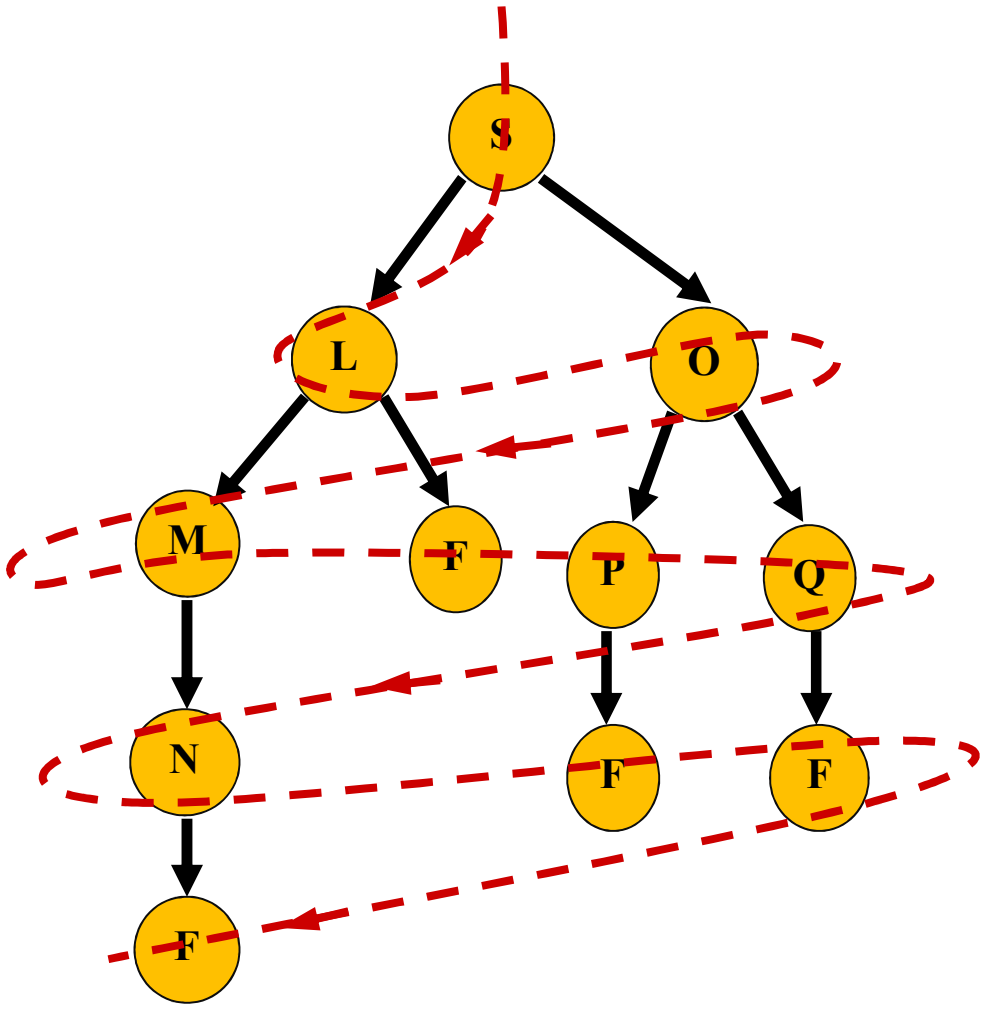
- 6）如果n的任一个后继节点是个目标节点，则找到一个解答，成功退出；否则转向第(2)步。

宽度优先搜索算法框图

思考：与原始算法比较异同，宽度优先的体现?



宽度优先搜索示意图



算法解析

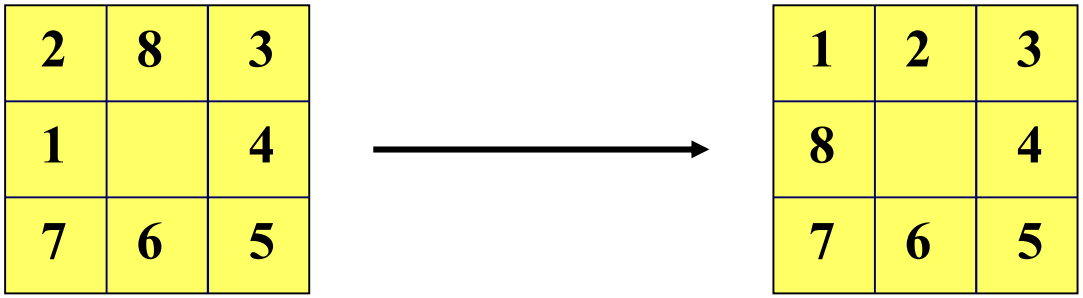
- OPEN表
  - S
  - L, O
  - O, M, F
  - M, F, P, Q
  - F, P, Q, N
  - P, Q, N
  - …
- CLOSED表
  - S，扩展得L, O
  - L，扩展得M, F
  - O，扩展得P, Q
  - M，扩展得N
  - F，无后继
  - P，扩展得F
  - …

目标节点判断顺序：**SLOMFP…**

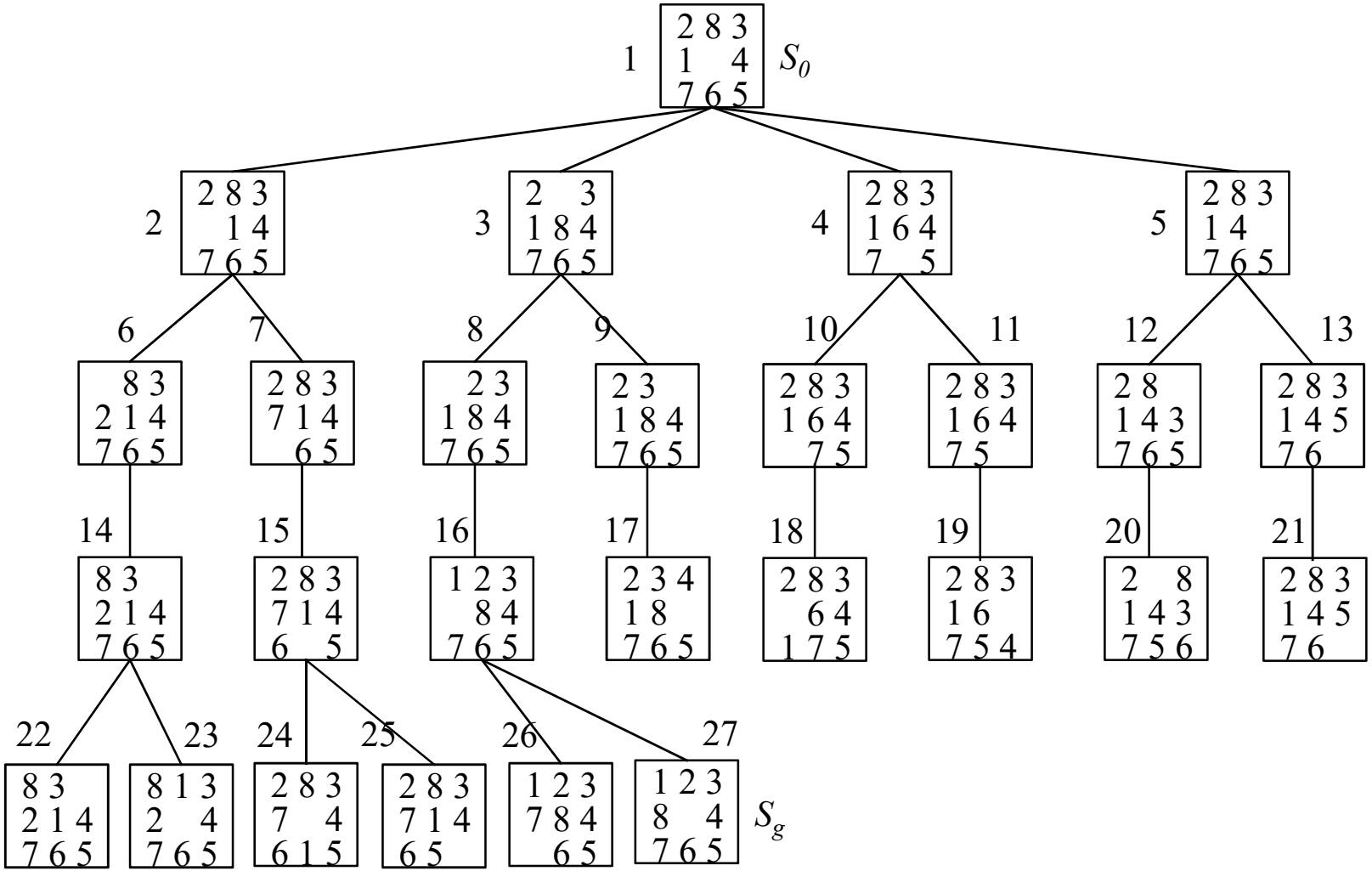
例子：八数码问题

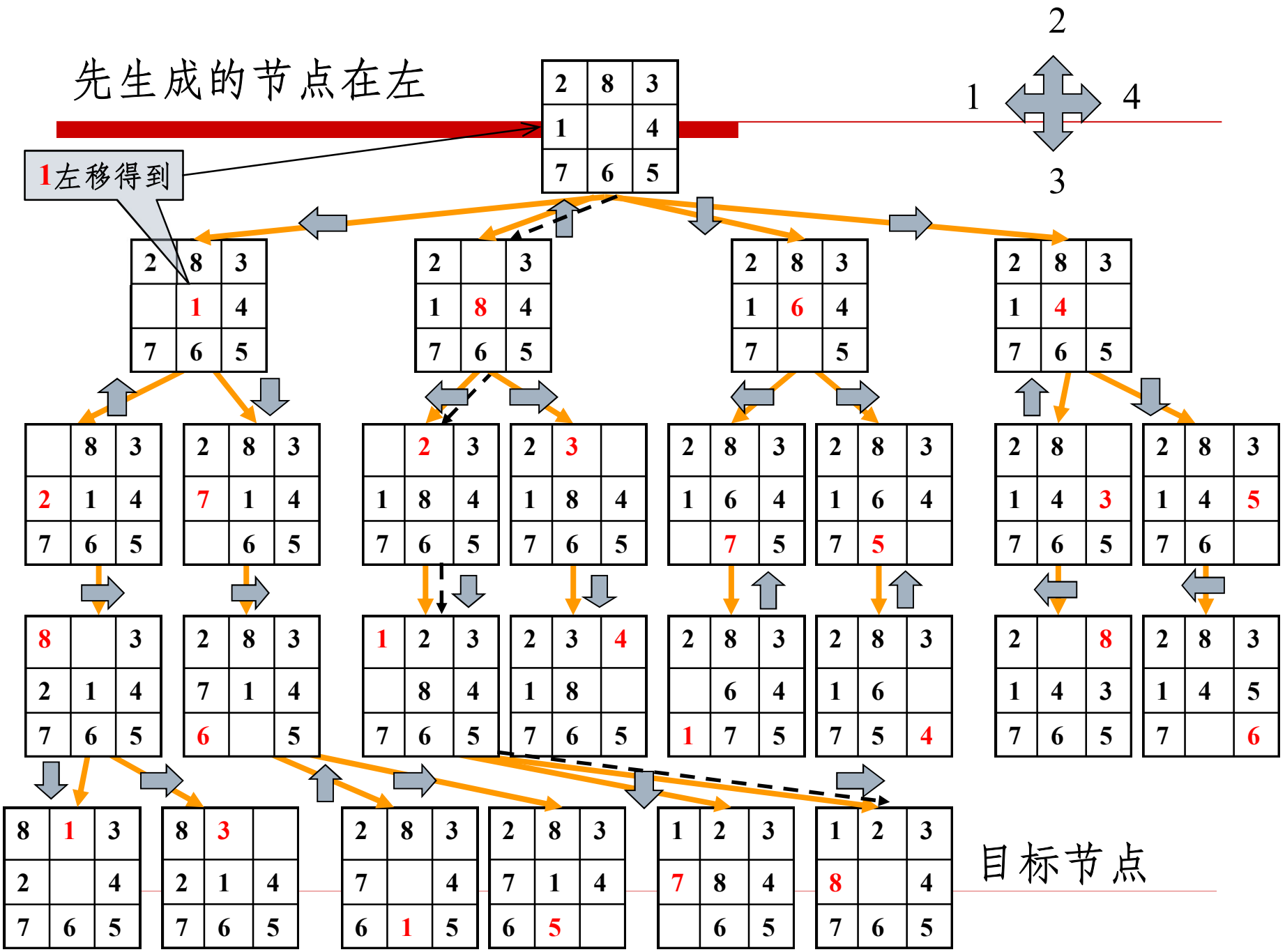
- 例子

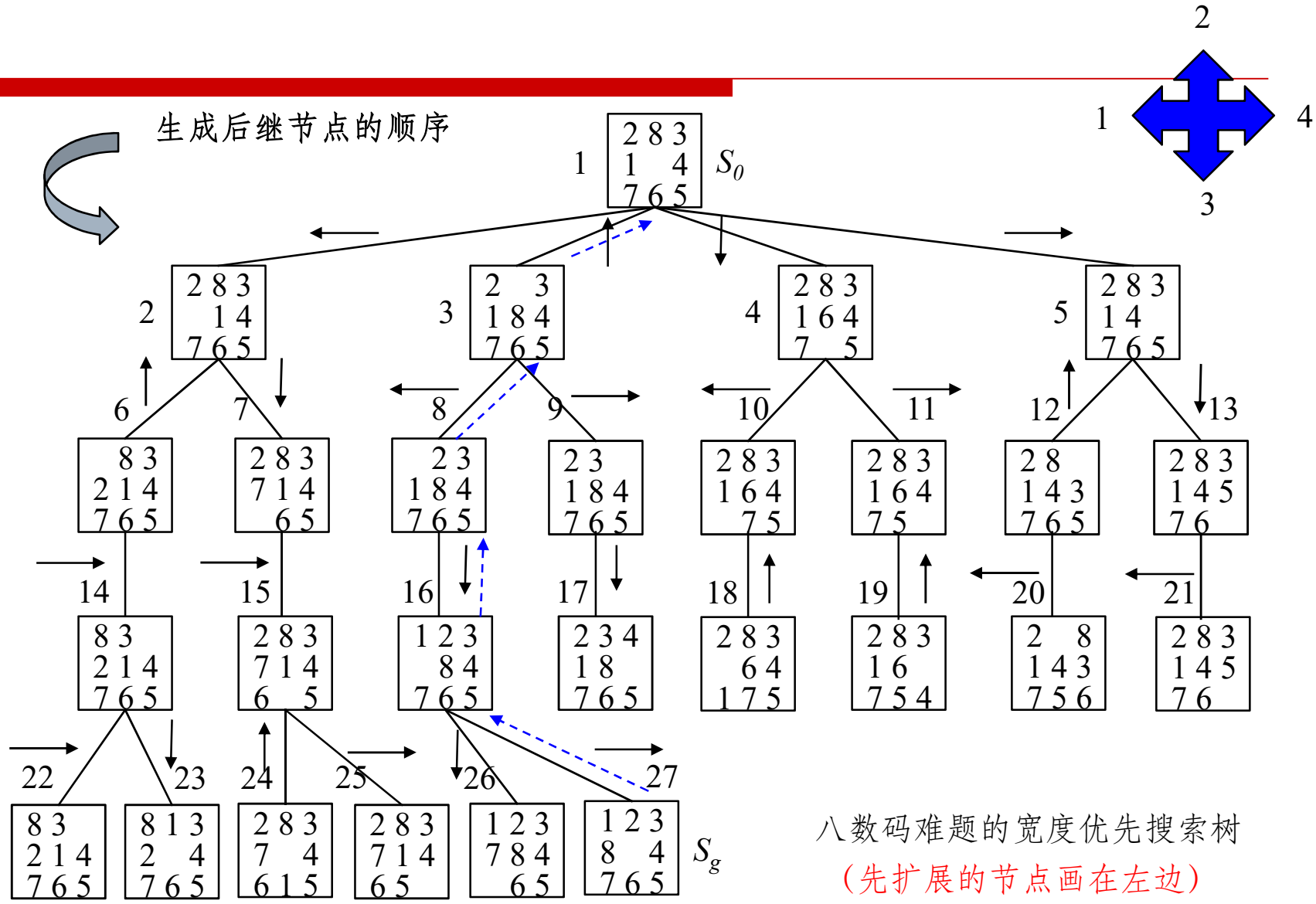
  八数码难题（8-puzzle problem）



规定：将牌移入空格的顺序为：从空格左边开始顺时针旋转。不许斜向移动，也不返回先辈节点。从图可见，要扩展26个节点，共生成46个节点之后才求得解（目标节点）


1
2 8 3
1 4
7 6 5
S_0
2
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
4
2 8 3
1 6 4
7 5
5
2 8 3
1 4
7 6 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 3
1 6 4
7 5
11
2 8 3
1 6 4
7 5
12
2 8
1 4 3
7 6 5
13
2 8 3
1 4 5
7 6
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 3
6 4
1 7 5
19
2 8 3
1 6
7 5 4
20
2 8
1 4 3
7 5 6
21
2 8 3
1 4 5
7 6
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
7 8 4
6 5
27
1 2 3
8 4
7 6 5
S_g


先生成的节点在左
1左移得到
2
1
4
3
目标节点

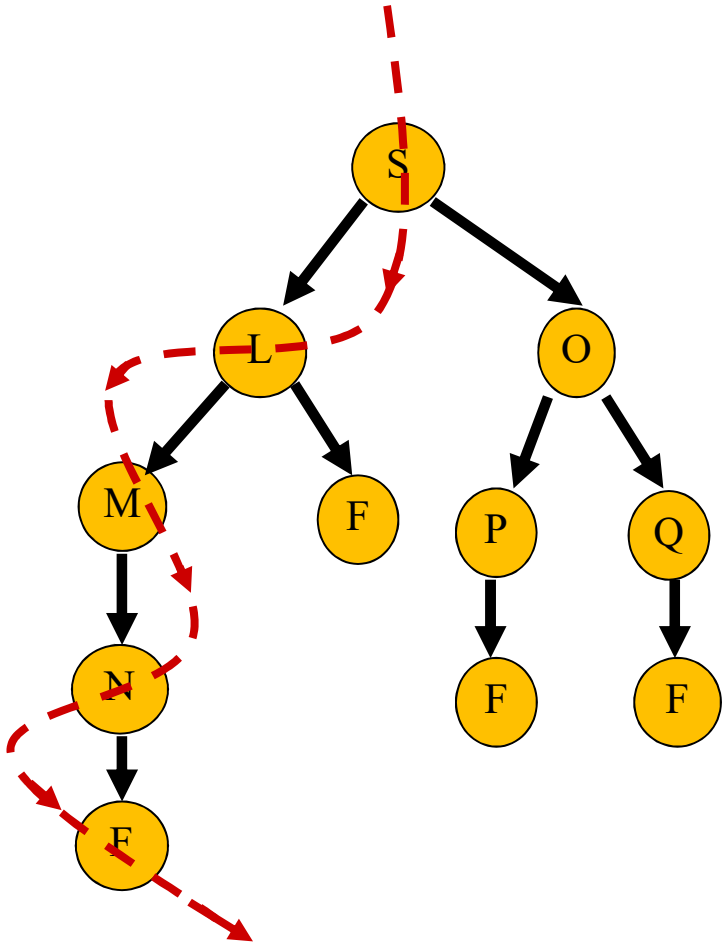

生成后继节点的顺序
2
1
4
3
S_0
S_g
八数码难题的宽度优先搜索树
（先扩展的节点画在左边）

深度优先搜索(Depphth-first)

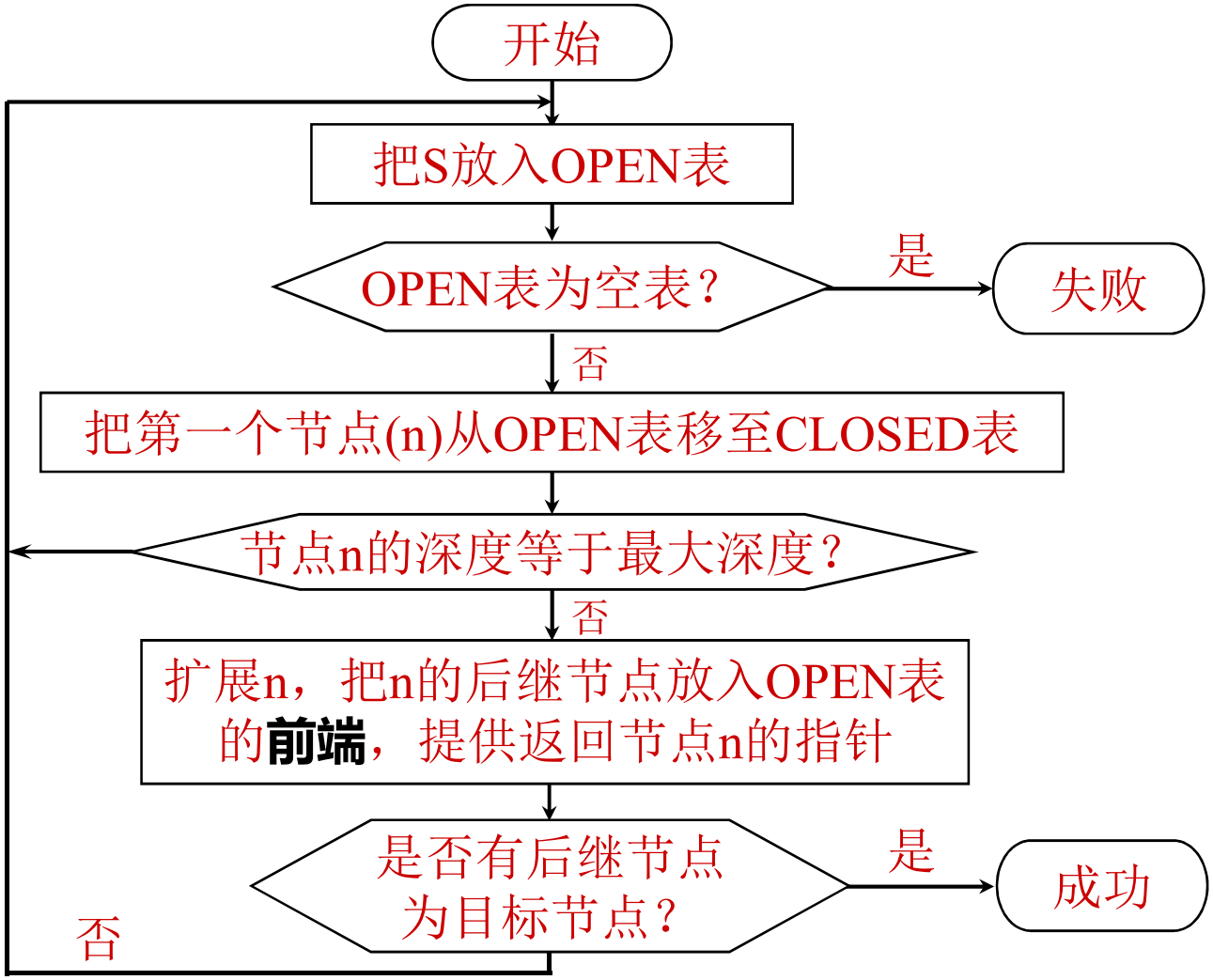
- 定义：
  - 首先扩展最新产生的(即最深的)节点。
- 特点：
  - 防止搜索过程沿着无益的路径扩展下去，往往给出一个节点扩展的最大深度——深度界限。
  - 与宽度优先搜索算法最根本的不同在于：将扩展的后继节点放在**OPEN**表的**前端**。

放在前端，后进先出，形成栈

深度优先搜索示意图

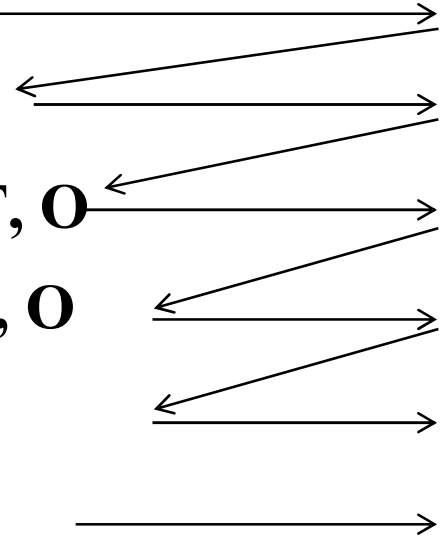


深度优先算法框图


开始
把S放入OPEN表
OPEN表为空表？
是
失败
否
把第一个节点(n)从OPEN表移至CLOSED表
节点n的深度等于最大深度？
否
扩展n，把n的后继节点放入OPEN表
的前端，提供返回节点n的指针
是否有后继节点
为目标节点？
是
成功
否

算法解析

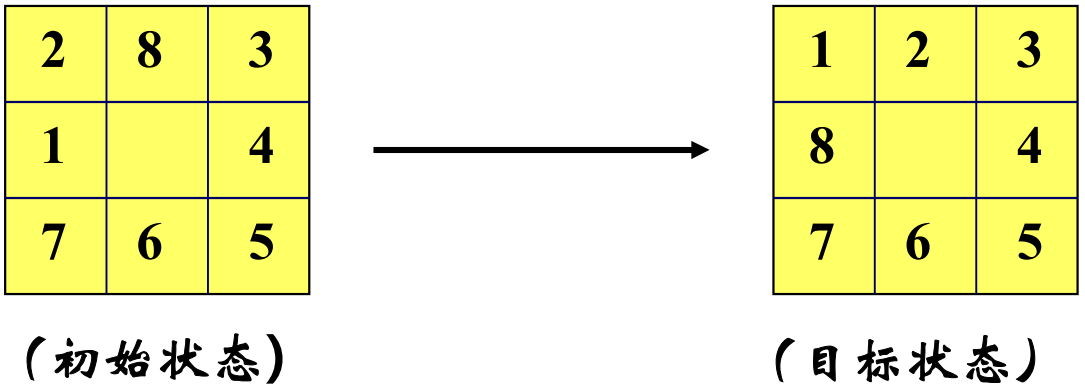
- OPEN表
  - S
  - L, O
  - M, F, O
  - N, F, O
  - F, O
  - O
  - …
- CLOSED表
  - S，扩展得L, O
  - L，扩展得M, F
  - M，扩展得N
  - N，扩展得F，尾指针，返回
  - F，无后继
  - O，扩展得P, Q
  - …



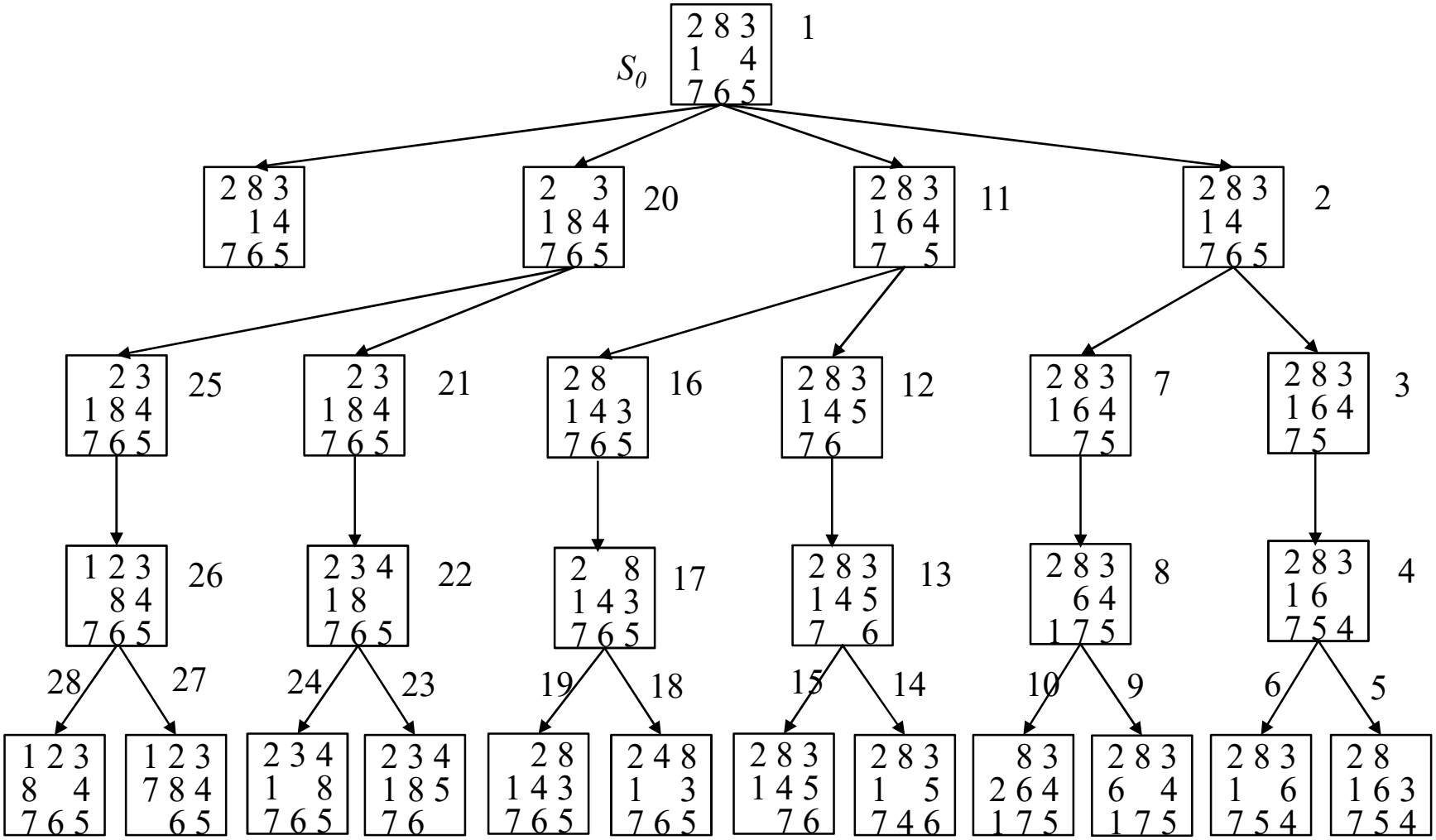
目标节点判断顺序：**SLMNFO…**

例子：八数码问题

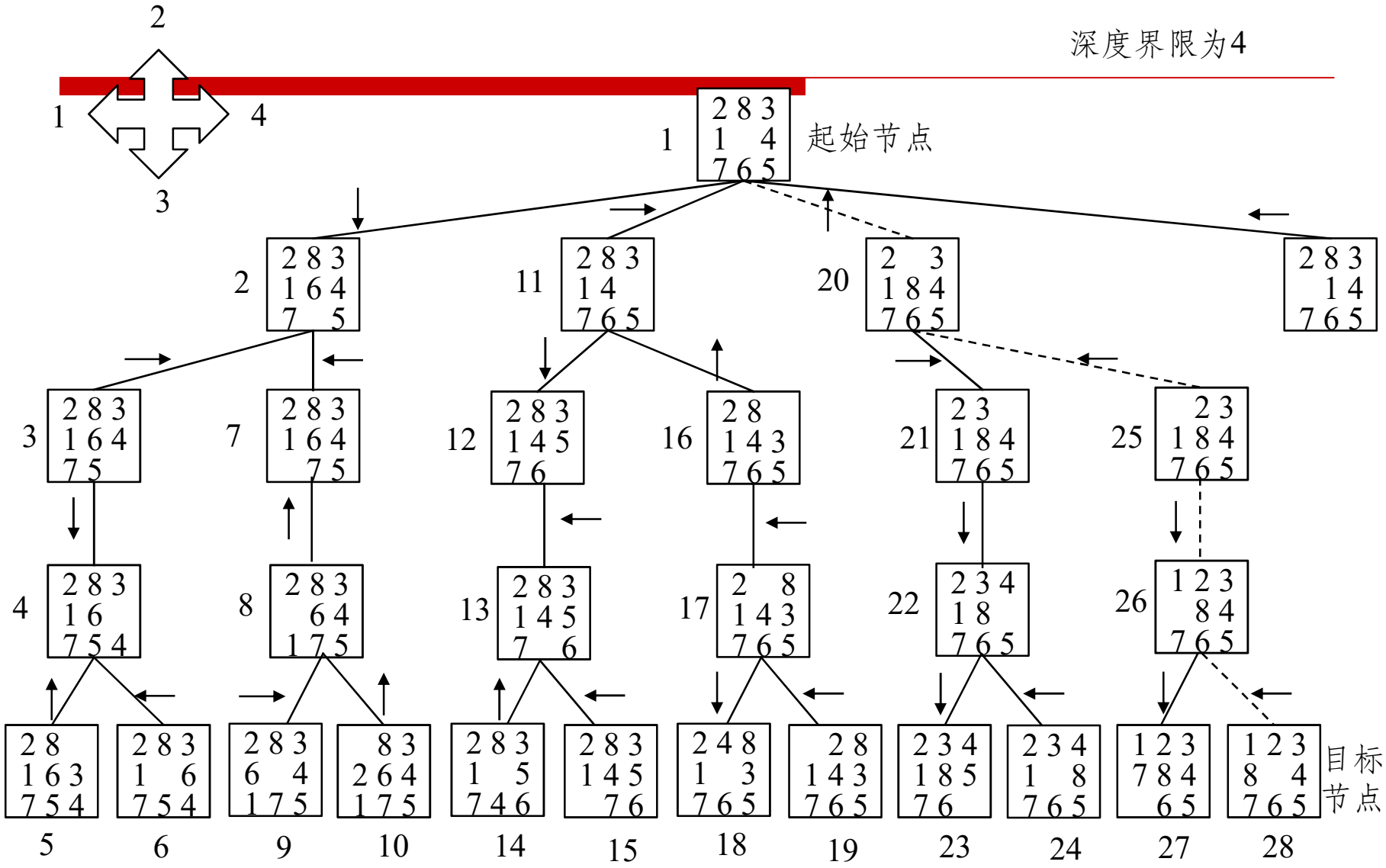
- 例子：
  - 有界深度(4)优先的八数码问题深度优先搜索树？



（初始状态）　　（目标状态）

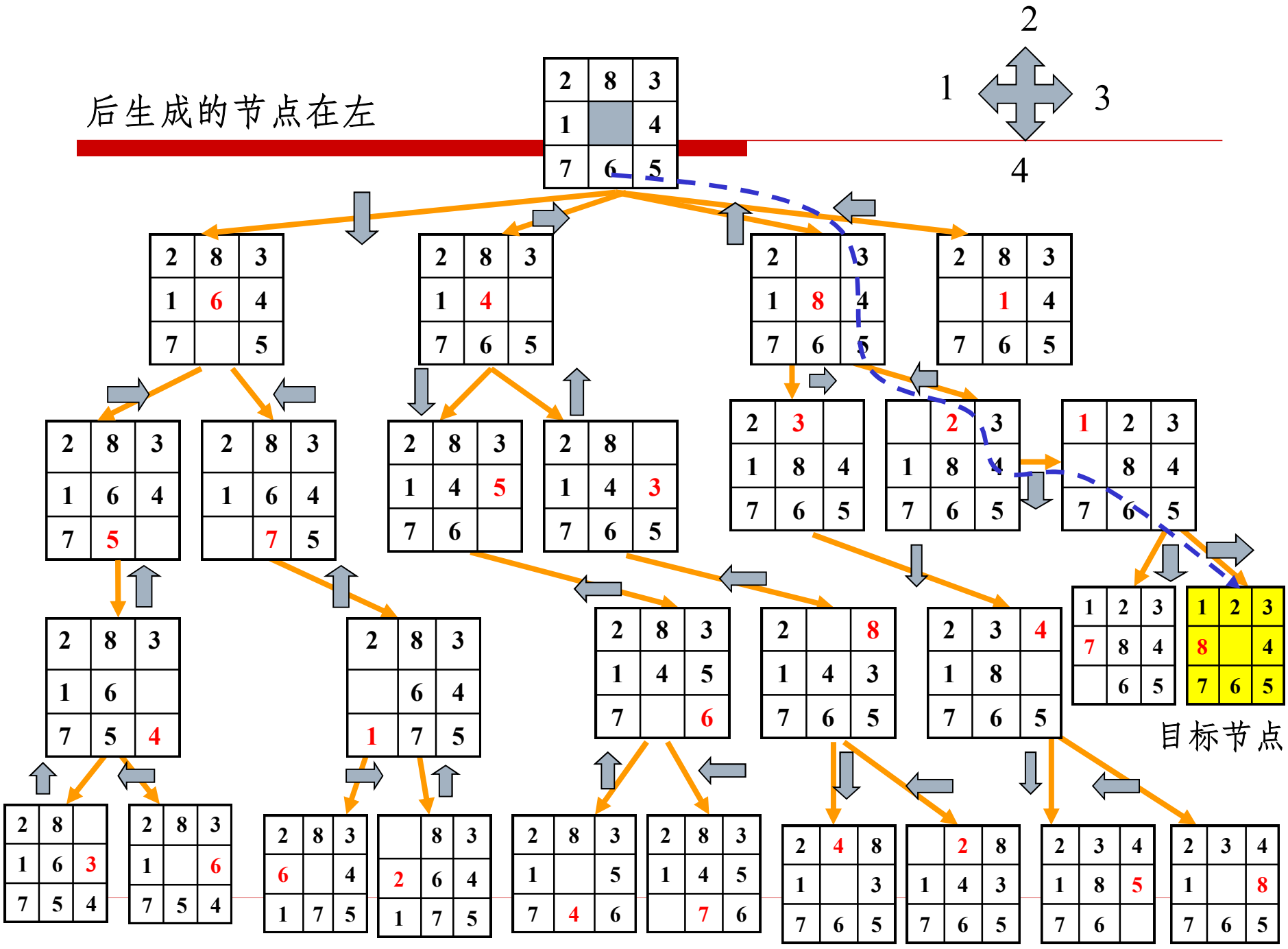

S_0
2 8 3
1 4
7 6 5
1
2 8 3
1 4
7 6 5
2
2 8 3
1 6 4
7 5
11
2 3
1 8 4
7 6 5
20
2 8 3
1 4
7 6 5
2 3
1 8 4
7 6 5
25
2 3
1 8 4
7 6 5
21
2 8
1 4 3
7 6 5
16
2 8 3
1 4 5
7 6
12
2 8 3
1 6 4
7 5
7
2 8 3
1 6 4
7 5
3
1 2 3
8 4
7 6 5
26
2 3 4
1 8
7 6 5
22
2 8
1 4 3
7 6 5
17
2 8 3
1 4 5
7 6
13
2 8 3
6 4
1 7 5
8
2 8 3
1 6
7 5 4
4
28
27
24
23
19
18
15
14
10
9
6
5
1 2 3
8 4
7 6 5
1 2 3
7 8 4
6 5
2 3 4
1 8
7 6 5
2 3 4
1 8 5
7 6
2 8
1 4 3
7 6 5
2 4 8
1 3
7 6 5
2 8 3
1 4 5
7 6
2 8 3
1 5
7 4 6
8 3
2 6 4
1 7 5
2 8 3
6 4
1 7 5
2 8 3
1 6
7 5 4
2 8
1 6 3
7 5 4

深度界限为4



八数码难题的深度优先搜索树

后生成的节点画在左边


后生成的节点在左
2
1
3
4
目标节点

等代价搜索

定义

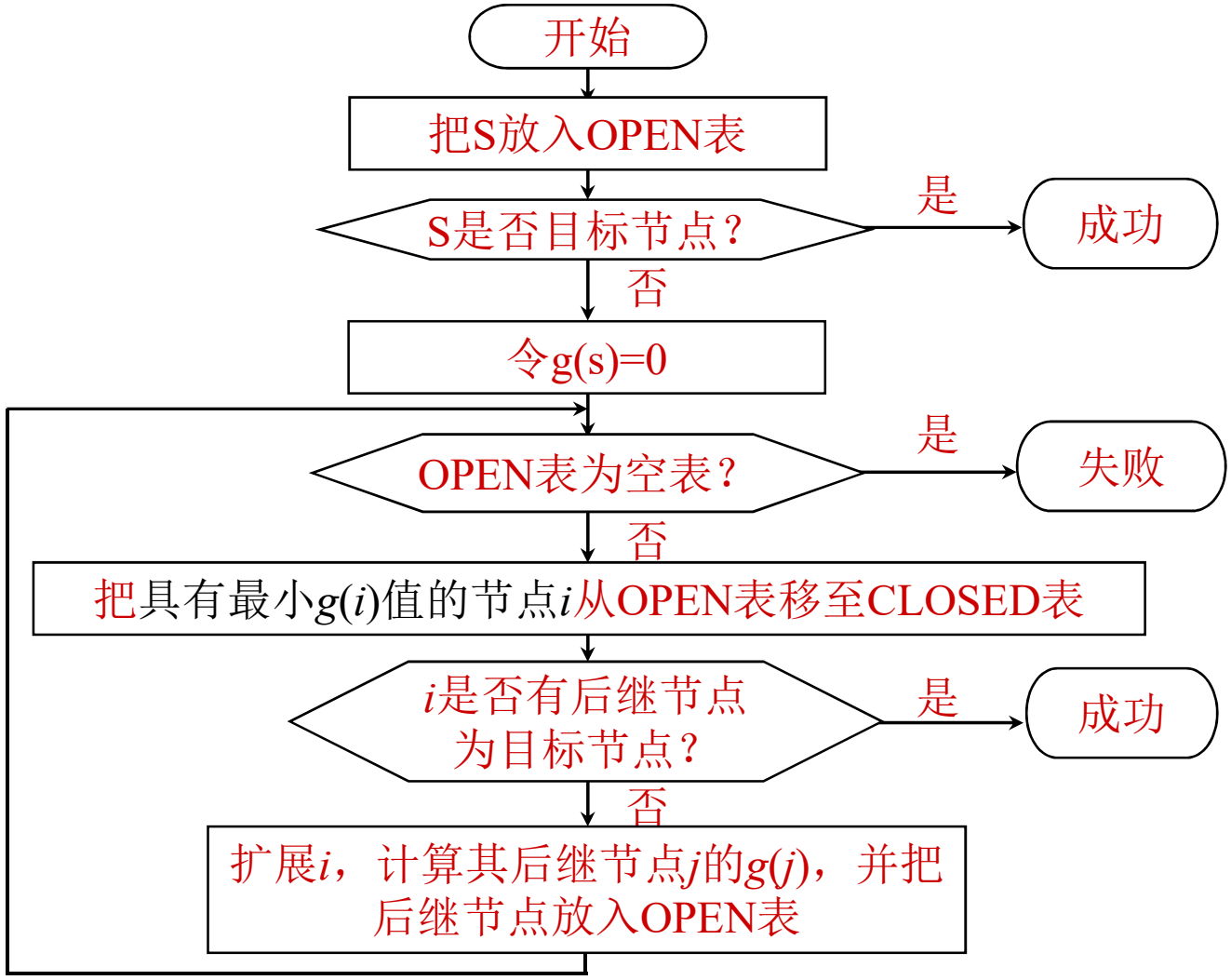
- 是宽度优先搜索的一种推广，不是沿着等长度路径断层进行扩展，而是沿着等代价路径断层进行扩展。又称“一致搜索”
- 搜索树中每条连接弧线上的有关代价，表示时间、距离等花费。

算法

- 在等价搜索算法中，把从节点i到其后续节点 j 的连接弧线记为 $c(i, j)$，把从起始节点S到任一节点 i 的路径代价记为 $g(i)$。在搜索树上，假设$g(i)$也是从起始节点S到节点的最少代价路径上的代价

思考：如何动态计算 $g(i)$？

等代价搜索算法框图


开始
把S放入OPEN表
S是否目标节点？
是
成功
否
令g(s)=0
OPEN表为空表？
是
失败
否
把具有最小g(i)值的节点i从OPEN表移至CLOSED表
i是否有后继节点
为目标节点？
是
成功
否
扩展i，计算其后继节点j的g(j)，并把
后继节点放入OPEN表

目录

- 图搜索策略
- 盲目搜索
- 启发式搜索
- 消解原理
- 规则演绎系统
- 产生式系统
- 非单调推理
- 小结

启发式搜索

- **思考：**
  - （1）图搜索方法的基本步骤？
  - （2）宽度优先、深度优先、等代价方法的特点？
  - （3）盲目搜索的缺点？

✓ 基本步骤：初始化，判断OPEN表是否为空，选择节点n，判断n是否目标节点，扩展节点n，重排OPEN表、调整指针，循环。
✓ 各自特点：重排OPEN表的依据不同。
✓ 盲目搜索可能带来组合爆炸。

- **启发式信息：**
  - 用来加速搜索过程的问题领域信息，一般与有关问题具体领域背景有关，不一定具有通用性。
- **启发式搜索：利用启发式信息的搜索方法**
  - 特点：重排**OPEN**表，选择最有希望的节点加以扩展
  - 种类：有序搜索、**A***算法等

启发式搜索策略和估价函数

- **有序搜索（Ordered Search）**
  - 总是选择“最有希望”的节点作为下一被扩展节点
- **估价函数（Evaluation Function）**
  - 为获得某些节点“希望”的启发信息，提供一个评定侯选扩展节点的方法，以便确定哪个节点最有可能在通向目标的最佳路径上。
  - $f(n)$——表示节点n的估价函数值
- **应用节点“希望”程度（估价函数值）重排OPEN表；有序搜索也称为*最佳优先搜索*;**
- **估价函数举例：**
  - （1）棋局的得分；
  - （2）距离目标状态的距离量度；
  - （3）TSP问题中的路径；

思考：f 函数的计算，重排序的方法?

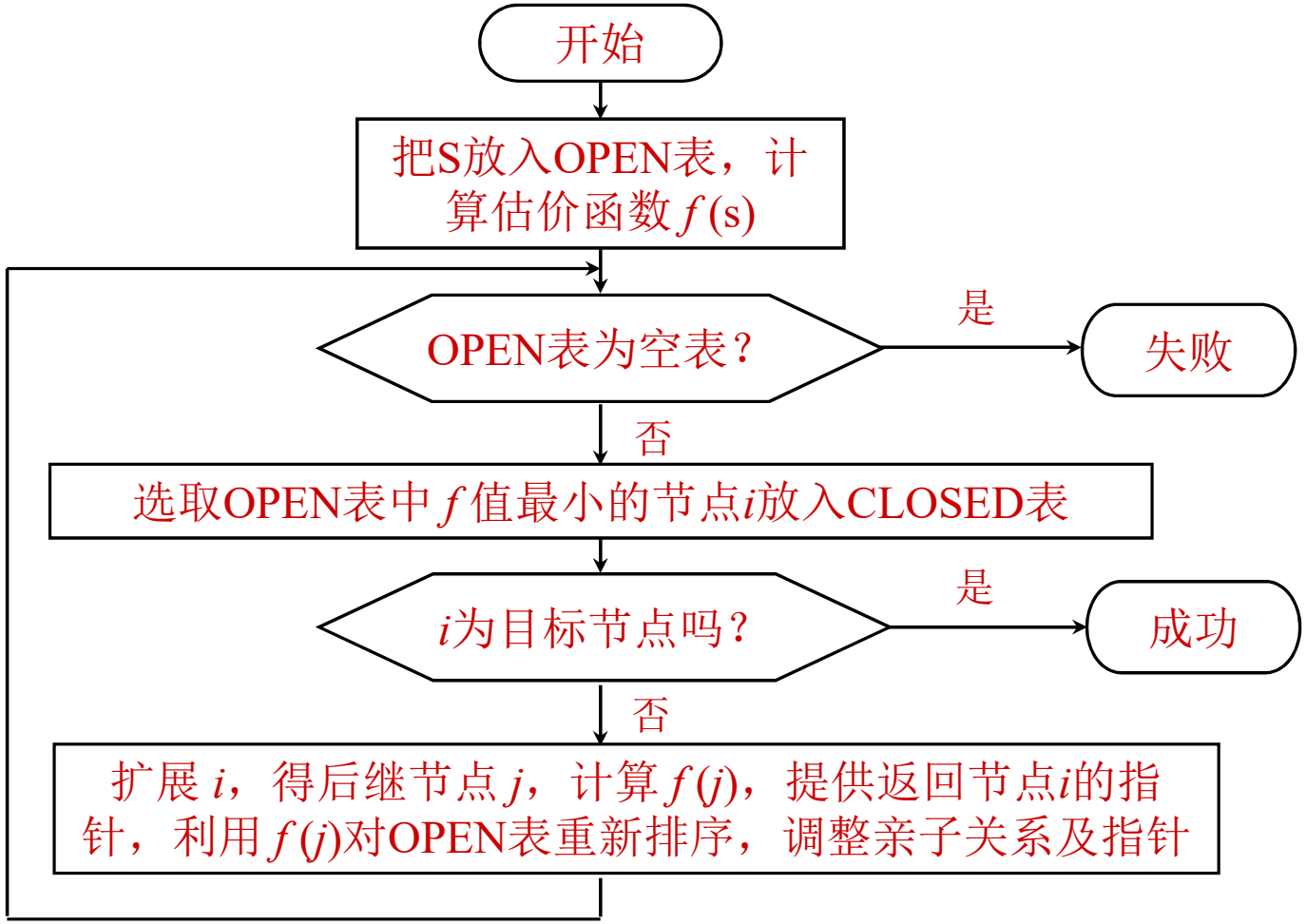
有序搜索(Ordered Search; Best-first Search)

- **实质**：选择OPEN表上具有最小 f 值的节点作为下一个要扩展的节点。

- **Nilsson（尼尔逊）方法**：一个节点的“希望”越大，则其 f 值越小。被选择的节点是估价函数最小的节点。

思考：如果把宽度优先、深度优先、等代价搜索方法作为有序搜索的特例，那么它们的 f 函数如何计算？

有序搜索算法框图


开始
把S放入OPEN表，计算估价函数 f (s)
OPEN表为空表？
是
失败
否
选取OPEN表中 f 值最小的节点i放入CLOSED表
i为目标节点吗？
是
成功
否
扩展 i，得后继节点 j，计算 f (j)，提供返回节点i的指针，利用 f (j)对OPEN表重新排序，调整亲子关系及指针

八数码问题

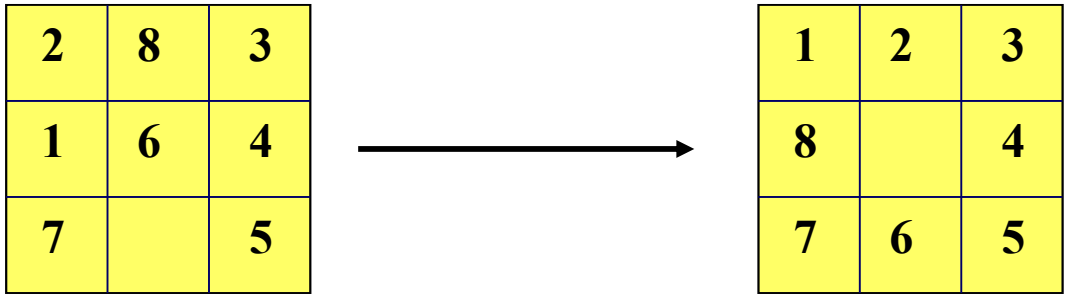
- （1）估价函数设置：

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

  $d(n)$: 节点n的深度

  $W(n)$: 错放的棋子数

- （2）如下的八数码难题（**8-puzzle problem**）

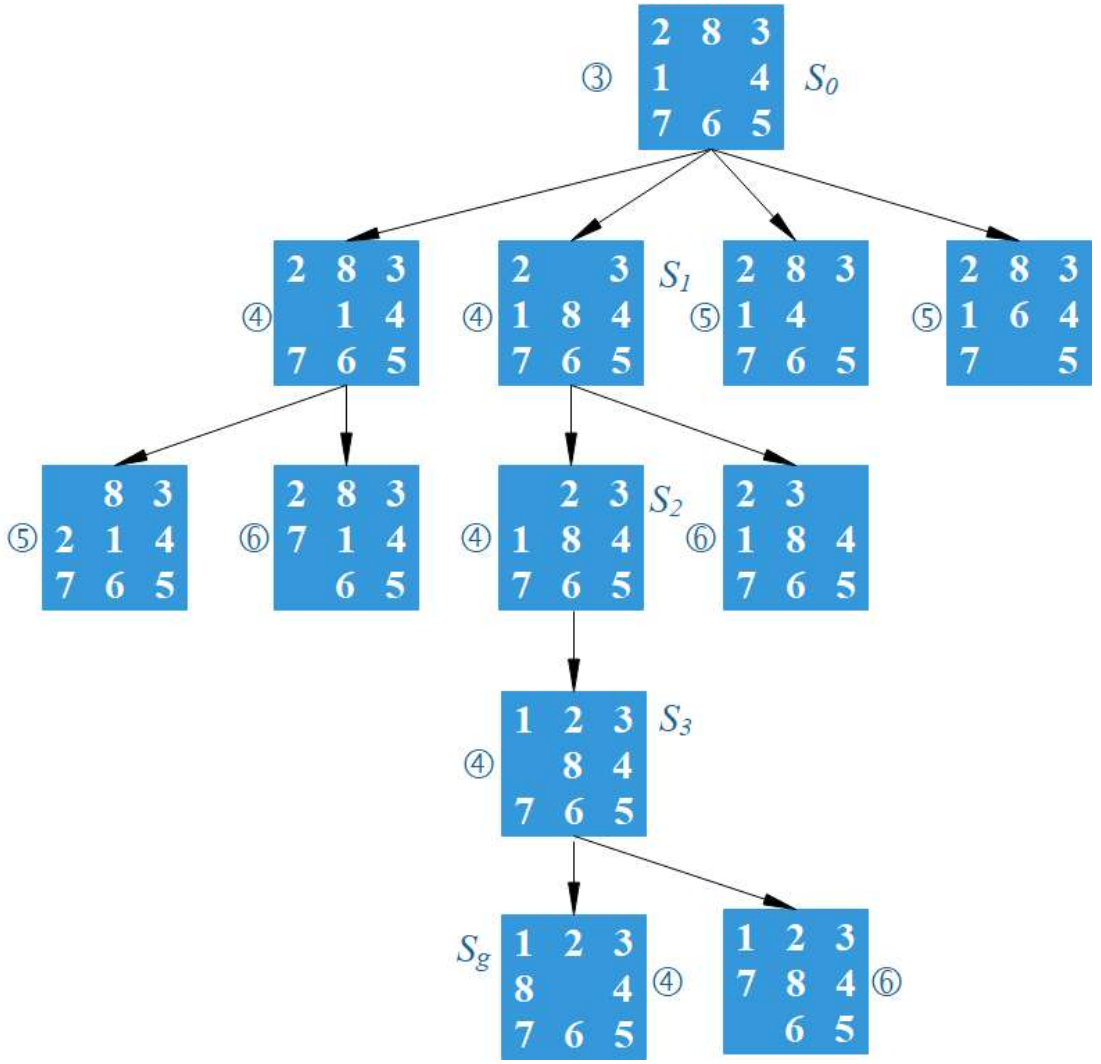


（初始状态）　　（目标状态）

- （3）八数码难题的有序搜索树见下图：

八数码难题有序搜索树

- 其中③表示$W(n)=3$，如根节点的2、8、3与目标节点不一致，所以$W(1)=3$，其中$W(1)$中的1表示第一个节点



- **f函数的重要性**
  - 有序搜索的有效性直接取决于f，f是提高搜索效率的关键；
  - 如果f不准确，可能失去最佳解，也可能失去全部解；
- **f一般选择策略**
  - 搜索时间与空间的折衷；
  - 保证有解或有最佳解；
- **f选择的三种典型情况：**
  - 最优解答：状态空间中有多条解答路径，求解最优解答，如A*算法；
  - 搜索代价与解答质量的综合：问题类似于（1），但搜索过程可能超出时间与空间的界限。在适当的搜索实验中找到满意解答，并限制满意解答与最优解答的差异程度；如：TSP问题；
  - 最小搜索代价：不考虑解答的最优化（只有一个解答或多个解答间无差异），尽量使搜索代价最小；如：定理证明。

思考：
(1) f不能识别某些节点的真实“希望”值会怎么样？
(2) f过多估计了全部节点又会怎么样？

A*算法

- **思考**：经过节点n的最佳路径，怎么表示？怎么求解最优解答路径。

- **估价函数** f^*：对节点n定义 $f^*(n)=g^*(n)+h^*(n)$,表示从S开始通过节点n的一条最佳路径的代价。其中 $g^*(n)$ 表示从起始节点S到n的最佳路径，$h^*(n)$表示从n到某目标节点的最佳路径。

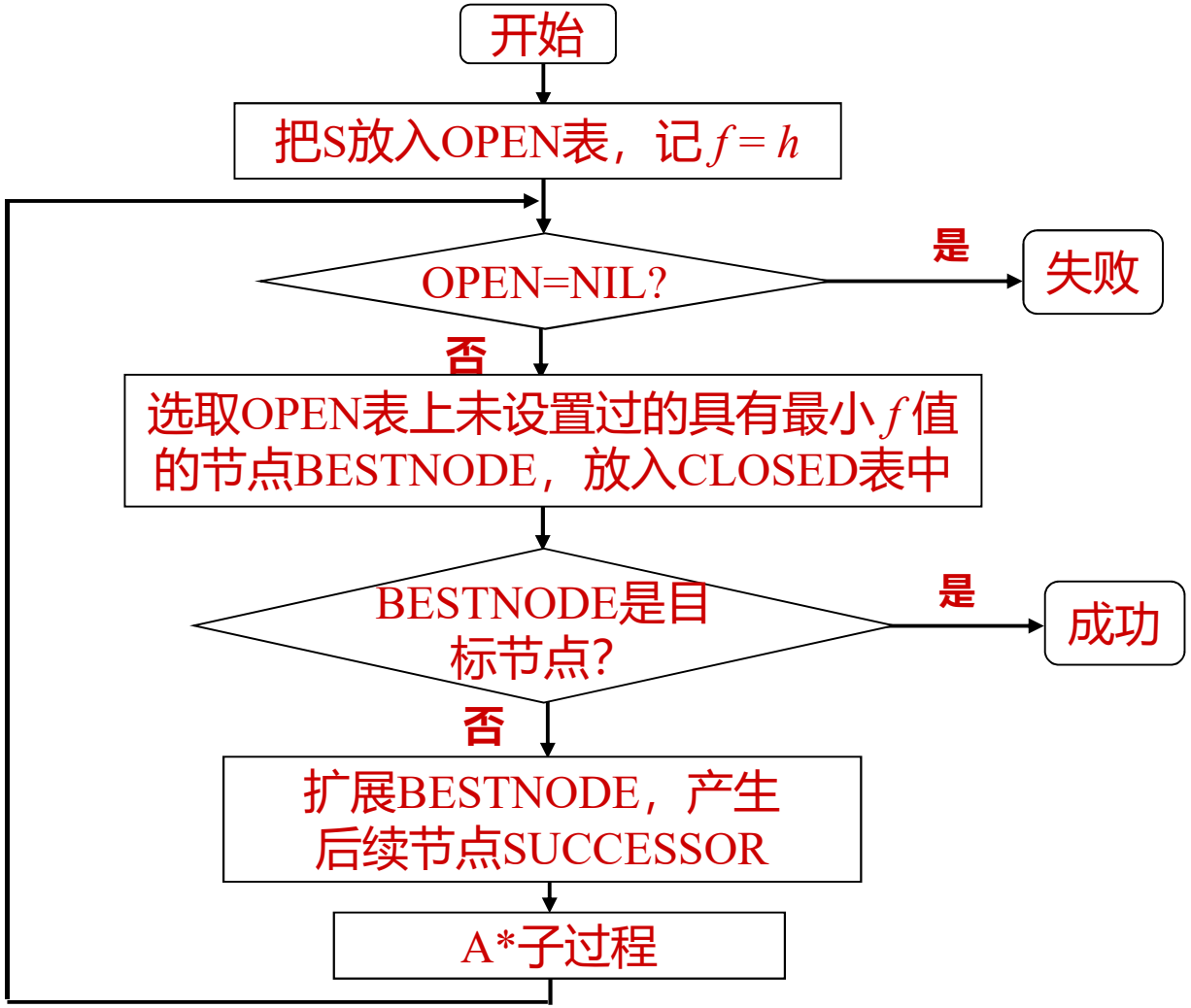
- **估价函数** f：$f(n)=g(n)+h(n)$； 其中g是g^*的估计，h是h^*的估计；
  - g 的一个选择就是搜索树中从 S 到 n 的这段路径的代价；显然有$g(n)\geq g^*(n)$;
  - h 的依赖于领域的启发信息，比如八数码问题中的 $W(n)$, h 称为**启发式函数**;

□ A*算法：

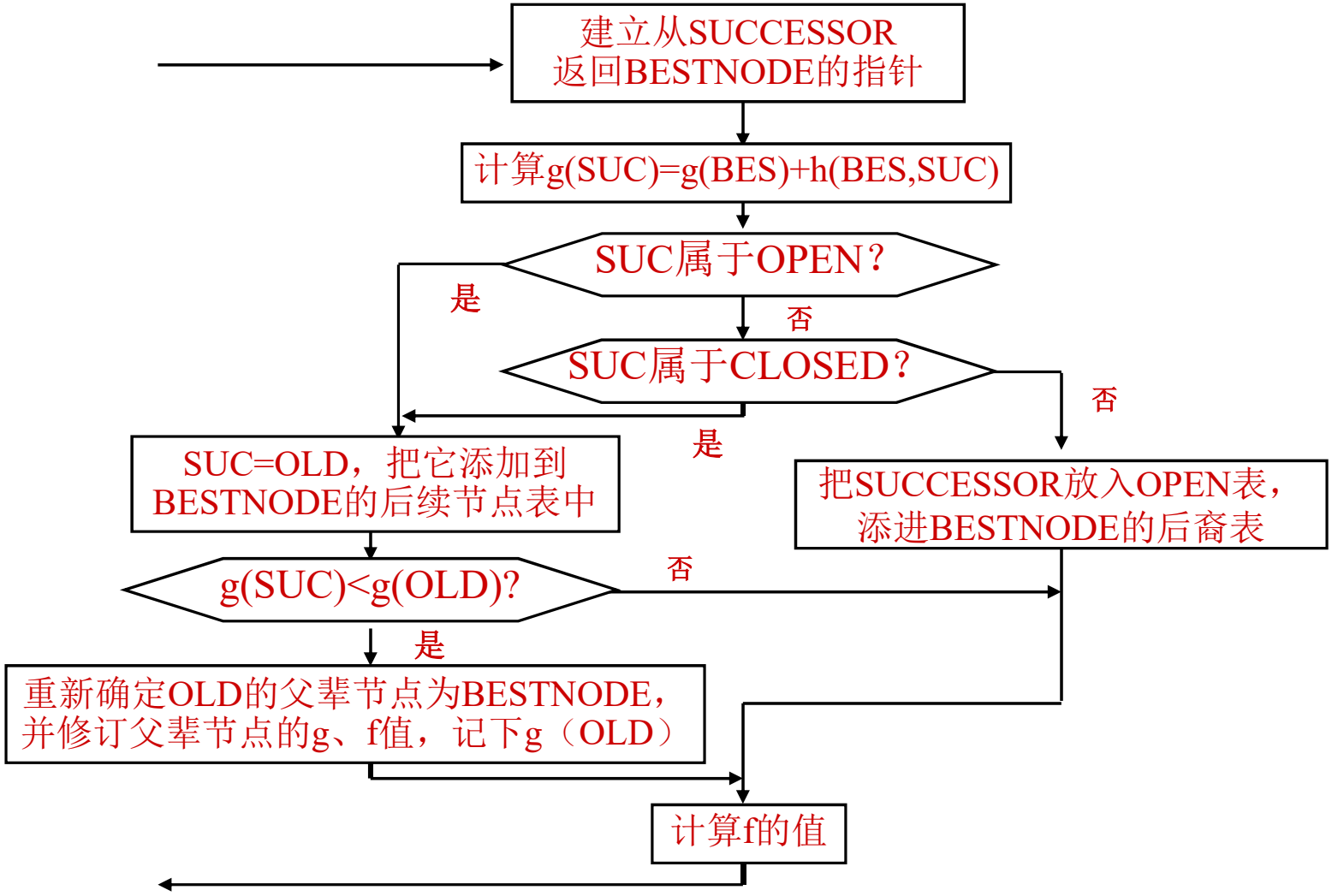
- 定义1　在Graph Search过程中，如果第8步的重排OPEN表是依据$f(x)=g(x)+h(x)$进行的，则称该过程为**A算法**。
- 定义2　在A算法中，如果对所有的x存在$h(x)\leq h^*(x)$,则称$h(x)$为$h^*(x)$的**下界**，它表示某种偏于保守的估计。
- 定义3　采用$h^*(x)$的下界$h(x)$为启发函数的A算法，称为**A*算法**。当$h=0$时，A*算法就变为**等代价搜索算法**。

启发式搜索算法（如A*算法）和等代价搜索算法实现类似，不同的是启发式搜索是根据 f 值而不是根据 g 值对优先级队列排队

A*算法总框图


开始
把S放入OPEN表，记 $f=h$
OPEN=NIL?
是
失败
否
选取OPEN表上未设置过的具有最小 f 值的节点BESTNODE，放入CLOSED表中
BESTNODE是目标节点?
是
成功
否
扩展BESTNODE，产生后续节点SUCCESSOR
A*子过程

A*算法子过程


建立从SUCCESSOR
返回BESTNODE的指针
计算g(SUC)=g(BES)+h(BES,SUC)
SUC属于OPEN?
是
否
SUC属于CLOSED?
否
是
SUC=OLD，把它添加到
BESTNODE的后续节点表中
把SUCCESSOR放入OPEN表，
添进BESTNODE的后裔表
g(SUC)<g(OLD)?
否
是
重新确定OLD的父辈节点为BESTNODE，
并修订父辈节点的g、f值，记下g（OLD）
计算f的值

- **思考：**
  - （1）宽度优先搜索、深度优先搜索与等代价搜索的关系？
  - （2）有序与A*算法等搜索方法的关系？
  - （3）等代价搜索与A*算法的关系？
- **基于状态空间搜索算法的智能系统应用领域：**
  - 问题求解（智能问题）：迷宫
  - 下棋软件：国际象棋、围棋、中国象棋等
  - 策略游戏：RPG（Role Playing Games）

目录

- 图搜索策略
- 盲目搜索
- 启发式搜索
- 消解原理
- 规则演绎系统
- 产生式系统
- 非单调推理
- 小结

- **如何证明如下储蓄问题**

  前提：每个储蓄钱的人都获得利息。

  结论：如果没有利息，那么就没有人去储蓄钱。

消解原理

- **回顾：**
  - （1）谓词公式、原子公式、复合公式、合式公式；
  - （2）推理规则、置换、合一、最通用合一者；
  - （3）合式公式的等价关系；
- **消解原理（Resolution Principle）：** **对谓词公式进行分解和化简，多用于证明问题或定理（机器证明）**
  - （1）一种用于子句公式集的重要推理规则；
  - （2）子句是由文字的析取组成的公式；
  - （3）一个原子公式和原子公式的否定叫作**文字**；
- **消解的过程：** 消解规则应用于母体子句对，以便产生导出子句；

举例：{ E1∨E2 ， ～ E2∨E3 } 消解导出 E1∨E3

问题：如何求取子句集？消解过程怎么进行？

子句集的求取

- 步骤：共9步。下面结合例子进行介绍
- 实例问题： 将下列谓词演算公式化为一个子句集

$$(\forall x)\ \{P(x) \Rightarrow \{(\forall y)\ [P(y) \Rightarrow P(f(x,y))] \wedge \sim(\forall y)[Q(x,y) \Rightarrow P(y)]\ \}\ \}$$

- (1) 消去蕴涵符号
  - 应用$\vee$和$\sim$符号，以$\sim A \vee B$替换$A \Rightarrow B$。

$$(\forall x)\ \{\sim P(x) \vee \{(\forall y)\ [\sim P(y) \vee P(f(x,y))] \wedge \sim (\forall y)\ [\sim Q(x,y) \vee P(y)]\}\}$$

(2) 减少否定符号的辖域

每个否定符号～最多只用到一个谓词符号上，并反复应用**狄•摩根定律**。

$$(\forall x)\{\sim P(x)\vee\{(\forall y)[\sim P(y)\vee P(f(x,y))]\wedge(\exists y)[Q(x,y)\wedge\sim P(y)]\}\}$$

(3) 对变量标准化

对哑元（虚构变量）改名，以保证每个**量词**有其自己**唯一的哑元**。

$$(\forall x)\{\sim P(x)\vee\{(\forall y)[\sim P(y)\vee P(f(x,y))]\wedge(\exists w)[Q(x,w)\wedge\sim P(w)]\}\}$$

*y*被替换为*w*（哑元）

(4) ***消去存在量词∃***

(a) 对于全称量词辖域内的存在量词，以Skolem函数代替存在量词内的约束变量;

(b) 对于自由存在量词，以一个新常量替代

$(\forall x)\{\sim P(x)\vee\{(\forall y)[\sim P(y)\vee P(f(x,y))]\wedge Q(x,g(x))\wedge\sim P(g(x))]\}\}$

式中，w=g(x)为一Skolem函数。

(5) *化为前束形*

把所有全称量词移到公式的左边，并使每个量词的辖域包括这个量词后面公式的整个部分。

前束形={**前缀**}(全称量词串){**母式**}(无量词公式)

$(\forall x)(\forall y)\ \{\sim P(x)\vee\ \{\ [\sim P(y)\vee P(f(x,y))]\ \wedge\ [Q(x,g(x))\wedge\sim P(g(x))]\ \}\ \}$

(6) 把母式化为合取范式

任何母式都可写成由一些谓词公式和(或)谓词公式的否定的析取的有限集组成的合取。(分配律)

$$(\forall x)(\forall y)\{[\sim P(x)\vee\sim P(y)\vee P(f(x,y))]\wedge[\sim P(x)\vee Q(x,g(x))]\wedge[\sim P(x)\vee\sim P(g(x))]\}$$

(7) 消去全称量词

所有余下的量词均被全称量词量化了。消去前缀，即消去明显出现的全称量词。

$$\{[\sim P(x)\vee\sim P(y)\vee P(f(x,y))]\wedge[\sim P(x)\vee Q(x,g(x))]\wedge[\sim P(x)\vee\sim P(g(x))]\}$$

(8) 消去连词符号∧ （注意：开始出现子句）

用{A,B}代替(A∧B)，消去符号∧。最后得到一个有限集，其中**每个公式是文字的析取**。

~P(x)∨~P(y)∨P(f(x,y))
~P(x)∨Q(x,g(x))
~P(x)∨~P(g(x))

每个公式是文字的析取

(9) 更换变量名称

可以更换变量符号的名称，使一个变量符号不出现在一个以上的子句中。

~P(x1)∨~P(y)∨P［f(x1,y)］
~P(x2)∨Q［x2,g(x2)］
~P(x3)∨~P［g(x3)］

总结：9步法求取子句集

- (1)消去蕴涵符号
- (2)减少否定符号的辖域（狄·摩根定律）
- (3)变量标准化（哑元唯一）
- (4)消去存在量词(∃)
- (5)化为前束形
- (6)化为合取范式(∧)
- (7)消去全称量词(∀)
- (8)消去连词符号(∧)
- (9)更换变量名称(同一变量不出现在一个以上子句)

练习题：求如下合适公式

(1) $(\forall x)\{[(\exists y)(S(x,y)\wedge M(y))] \Rightarrow [(\exists y)(I(y)\wedge E(x,y))]\}$

(2) $\sim\{[\sim(\exists x)I(x)] \Rightarrow [\sim(\exists x)(\exists y)(S(x,y)\wedge M(y))]\}$

的子句集。

消解推理规则

- 思考：求得子句集后，如何进行消解推理？
- *消解式*的*定义*：

> 令L_1, L_2为两任意原子公式；L_1和L_2具有相同的谓词符号，但一般具有不同的变量。已知两子句$L_1 \vee \alpha$和$\sim L_2 \vee \beta$,如果L_1和L_2具有**最一般合一** σ，那么通过消解可以从这两个父辈子句推导出一个新子句$(\alpha \vee \beta)\sigma$。这个新子句叫做**消解式**。

- *消解式*的*求法*
  - 取两个子句的**析取**，然后消去互补对，得到消解式

例题：求子句集$\{P \vee Q, \sim P \vee L\}$的消解式？

消解式求解的典型例子

- **(a) 假言推理**

  父辈子句： P ，~P ∨Q(即P=>Q)
  消解式：Q

- **(b) 合并**

  父辈子句： P ∨Q , ~P ∨Q
  消解式：Q ∨Q = Q

- **(c) 重言式**

  父辈子句： P ∨Q , ~P ∨ ~ Q
  消解式：Q ∨ ~ Q 或 P ∨ ~ P

- **(d) 矛盾(空子句)**

  父辈子句： ~P ，P
  消解式：NIL

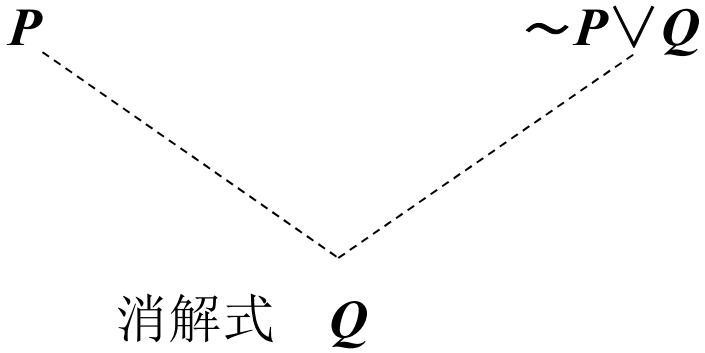
- **(e)三段论(链式)**

  父辈子句： ~P ∨Q (即P=>Q), ~ Q ∨R (即Q=>R)
  消解式： ~P ∨ R (即P=>R)

含有变量的消解式

提问：含有变量的消解式怎么求解?

- **例1：**



P	~P	Q	~P∨Q	P∧(~P∨Q)
1	0	0	0	0
0	1	0	1	0
1	0	1	1	1
0	1	1	1	0

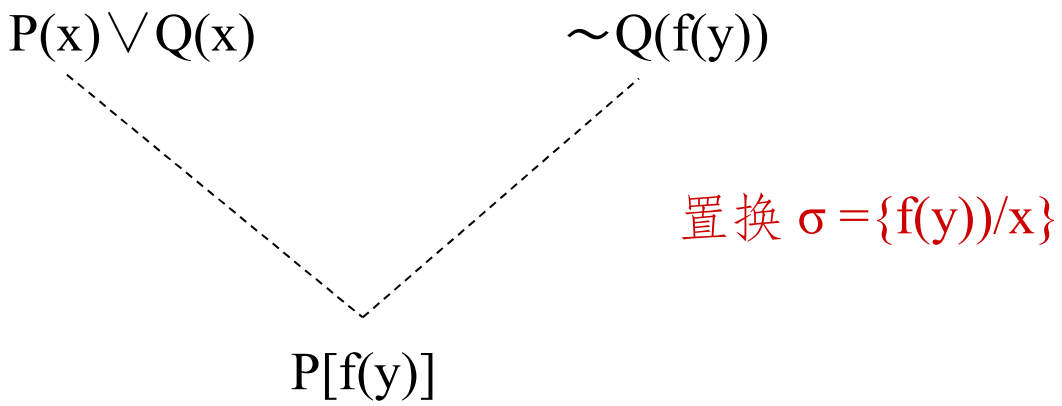
$\therefore P \wedge (\sim P \vee Q) \rightarrow Q$

- **含有变量的子句集之消解式**

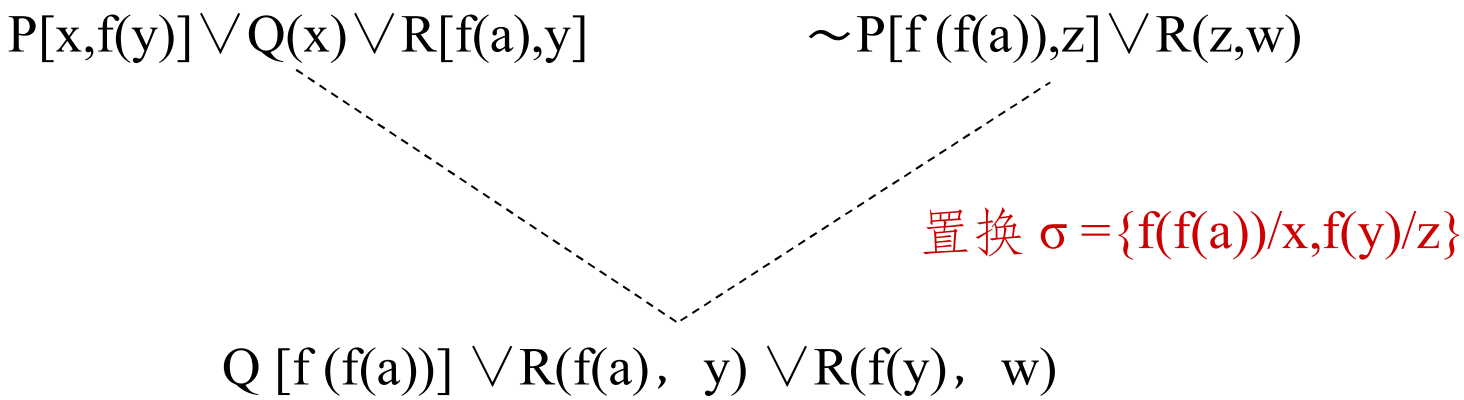
要把消解推理规则推广到含有变量的子句，必须找到一个作用于父辈子句的**置换**，使父辈子句含有互补文字。

含变量的消解式

- 例2.



- 例3.



消解反演求解过程

- **回顾：**
  - （1）基于谓词逻辑的知识表达方式；
  - （2）基于消解原理的基本推理原理；
  - （3）子句集求解的9步法；
  - （4）消解式的求取规则；
  - （5）带变量的消解式求取方法；

- **问题：如何采用上述消解方法进行逻辑推理?**
- 思考：
  - (1)什么是反证法（反演）？怎样用于数学定理证明？
  - (2)如何将反证法的思想运用于消解原理？

消解反演推理

- **消解反演推理方法：**
  - (1) 给出{S}，L
  - (2) 否定L，得～L；
  - (3) 把～L添加到S中去，把新产生的集合{～L，S}化成子句集；
  - (4) 应用消解原理，力图推导出一个表示矛盾的空子句

- **问题：利用反证法原理说明消解反演推理方法的原理?**

- **例题** —— 试采用消解反演方法证明如下储蓄问题

  前提：每个储蓄钱的人都获得利息。

  结论：如果没有利息，那么就没有人去储蓄钱。

证明

- **(1) 规定原子公式：**
  - S(x,y) 表示 “x储蓄y”
  - M(x) 表示 “x是钱”
  - I(x) 表示 “x是利息”
  - E(x,y) 表示 “x获得y”
- **(2)用谓词公式表示前提和结论：**
  - 前提： $(\forall x)[(\exists y)(S(x,y) \wedge M(y))] \Rightarrow [(\exists y)(I(y) \wedge E(x,y))]$
  - 结论： $[\sim(\exists x) I(x)] \Rightarrow [\sim(\exists x)(\exists y)(S(x,y) \wedge M(y))]$
  - 结论的否定式：
    - $\sim\{[\sim(\exists x) I(x)] \Rightarrow [\sim(\exists x)(\exists y)(S(x,y) \wedge M(y))]\}$

(3) 化为子句形（9步法）

把前提化为子句形：

1) ～S(x,y)∨～M(y)∨I(f(x))

2) ～S(x,y)∨～M(y)∨E(x,f(x))

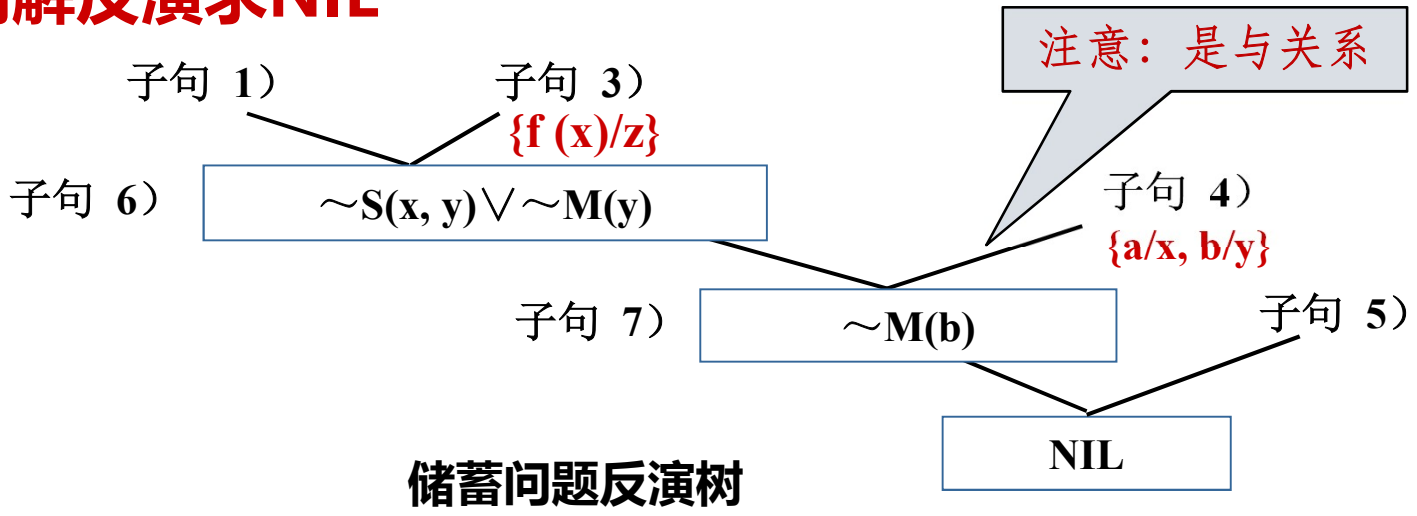
把结论的否定式化为子句形：

3) ～I(z)

4) S(a,b)

5) M(b)

(4) 消解反演求NIL



储蓄问题反演树

(5) So，结论成立！

反演求解过程

- **反演求解过程**
  - 从反演树求取答案的步骤：
    - (1) 把目标公式的否定式产生的每个子句添加到目标公式否定之否定的子句中去。
    - (2) 按照反演树，执行和以前相同的消解，直至在根部得到某个子句止。
    - (3) 用根部的子句作为一个解答语句。

- **提示：把一棵根部有NIL的反演树变换为根部带有回答语句的一棵证明树。**

小结

- **消解原理内容**
  - （1）子句集的求解（9步法）
  - （2）消解推理（置换、合一）
  - （3）反演证明（反演树）
  - （4）反演求解（重言式反演求解）

- **消解原理的应用**
  - （1）定理证明（四色定理）
  - （2）机器证明（吴文俊院士的吴氏方法）
  - （3）机器推理

- **消解方法多用于机器定理证明，只是AI发展过程中的一个阶段，解决简单问题可以，解决复杂问题比较难**

目录

- 图搜索策略
- 盲目搜索
- 启发式搜索
- 消解原理
- **规则演绎系统**
- 产生式系统
- 非单调推理
- 小结

规则演绎系统

引言

对于许多比较复杂的系统与问题，利用搜索技术与消解原理很难甚至无法求解(搜索与消解效率低，子句求取可能丢失重要信息)，需要一些更先进的推理技术与系统求解方法，如规则演绎系统，产生式系统等。

只要有规则和事实，就可以演绎（推理），演绎出新规则（断言）

定义

基于规则的问题求解系统运用If → Then规则来建立，每个if可能与某断言(assertion)集中的一个或多个断言匹配。有时把该断言集称为**工作内存**，then部分用于规定放入工作内存的**新断言**。这种基于规则的系统叫做**规则演绎系统**（rule based deduction system）。在这种系统中，通常称每个if部分为前项，每个then部分为后项。

分类

正向推理(Forward reasoning)、逆向推理(Backward reasoning)

正向演绎

- **事实：春天来了**
- **规则：**
  - （1）春天来了，燕子就飞回来了
  - （2）燕子飞回来后就筑巢
- **目标：燕子筑巢了吗?**

- **正向推理（又称正向演绎、前向推理）**
  - **1.**春天来了（事实）
  - **2.**春天来了，燕子就飞回来了

  } 燕子飞回来了（中间结果）

  - 燕子飞回来了（中间结果）
  - **3.**燕子飞回来后就筑巢

  } **燕子筑巢了（结论）**

逆向演绎

- **事实：春天来了**
- **规则：**
  - （1）春天来了，燕子就飞回来了
  - （2）燕子飞回来后就筑巢
- **目标：燕子筑巢了吗?**

- **逆向推理（又称逆向演绎、反向推理）**
  - **1.**燕子筑巢了
  - **2.**燕子飞回来后就筑巢

  燕子飞回来了（子目标）
  - 燕子就飞回来了（子目标）
  - **3.**春天来了，燕子就飞回来了

  春天来了（事实，与给定事实一致
  所以，**燕子筑巢了（结论）**）

规则正向演绎系统(forward chaining)

□ 定义

正向规则演绎系统是从事实到目标进行操作的，即从状况条件到动作进行推理的，也就是从if到then的方向进行推理的。

□ 求解过程

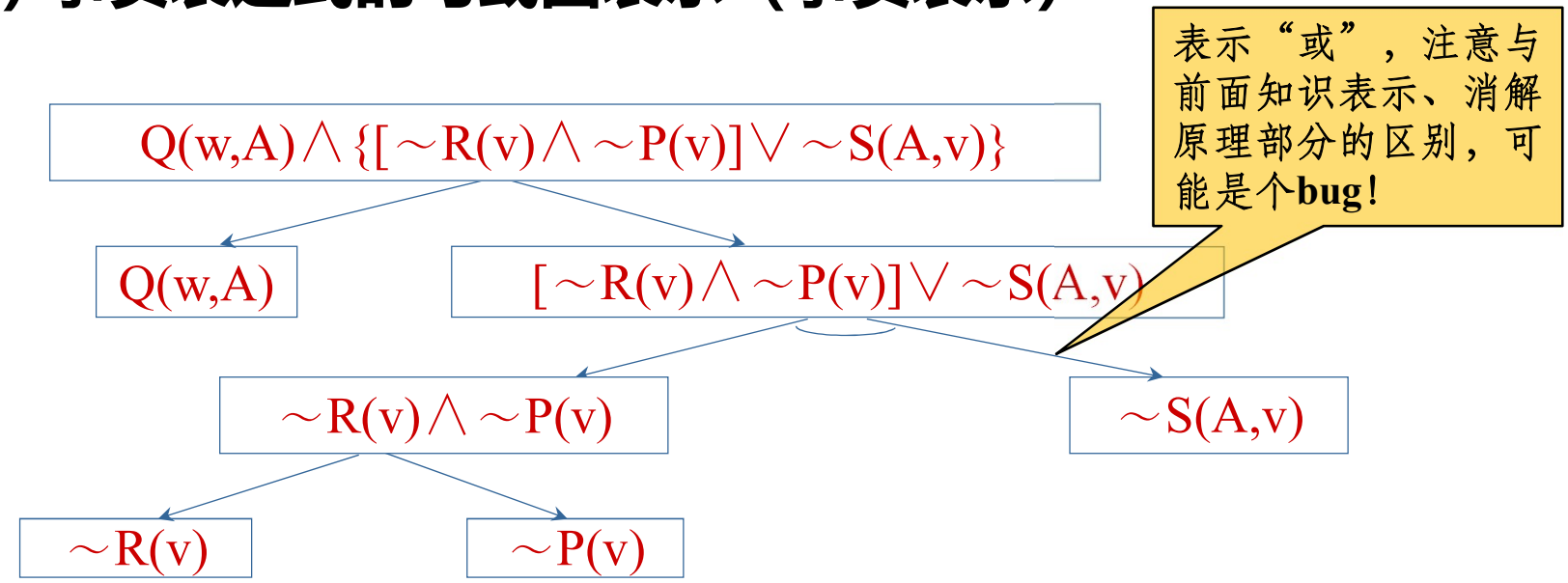
(1) 事实表达式的与或形变换（事实表示）

在基于规则的正向演绎系统中，我们把事实表示为非蕴涵形式的与或形，作为系统的总数据库。

与或图

- **事实表达式的与或形表示**
  - 子句集求取法的9步法中不必6、8两步
  - 例如：$(\exists u)(\forall v)\{Q(v,u) \wedge \sim[(R(v) \vee P(v)) \wedge S(u,v)]\}$
  - 可化为：$Q(w,A) \wedge \{[\sim R(v) \wedge \sim P(v)] \vee \sim S(A,v)\}$

(2) 事实表达式的与或图表示（事实表示）



(3) 与或图的F规则变换(知识推理)

- 推理规则（知识）的表示

  这些规则是建立在某个问题辖域中普通陈述性知识的蕴涵公式基础上的。我们把允许用作规则的公式类型限制为下列形式：

  $$L \Rightarrow W$$

  式中，L是**单文字**；W为**与或形的唯一公式**。

- F规则变换推理方法

  将上述规则应用于事实与或图中的**叶节点**，得到新的与或结构图，不断扩展该与或图。扩展的与或图既能表示**原始事实**，又能表示从原始事实与**推理规则**推出的**新的事实**表达式。

  F规则要求前项为单文字事实，后项为与或形

例子：F规则变换

- 普通蕴涵公式转换为单文字前项蕴涵式
- 举例：**公式** $(\forall x)\{[(\exists y)(\forall z)P(x,y,z)]=>(\forall u)Q(x,u)\}$**可以通过下列步骤加以变换：**
  - (1)暂时消去蕴涵符号

    $$(x)\{\sim[(y)(z)P(x,y,z)]\vee(u)Q(x,u)\}$$
  - (2)把否定符号移进第一个析取式内，调换变量的量词

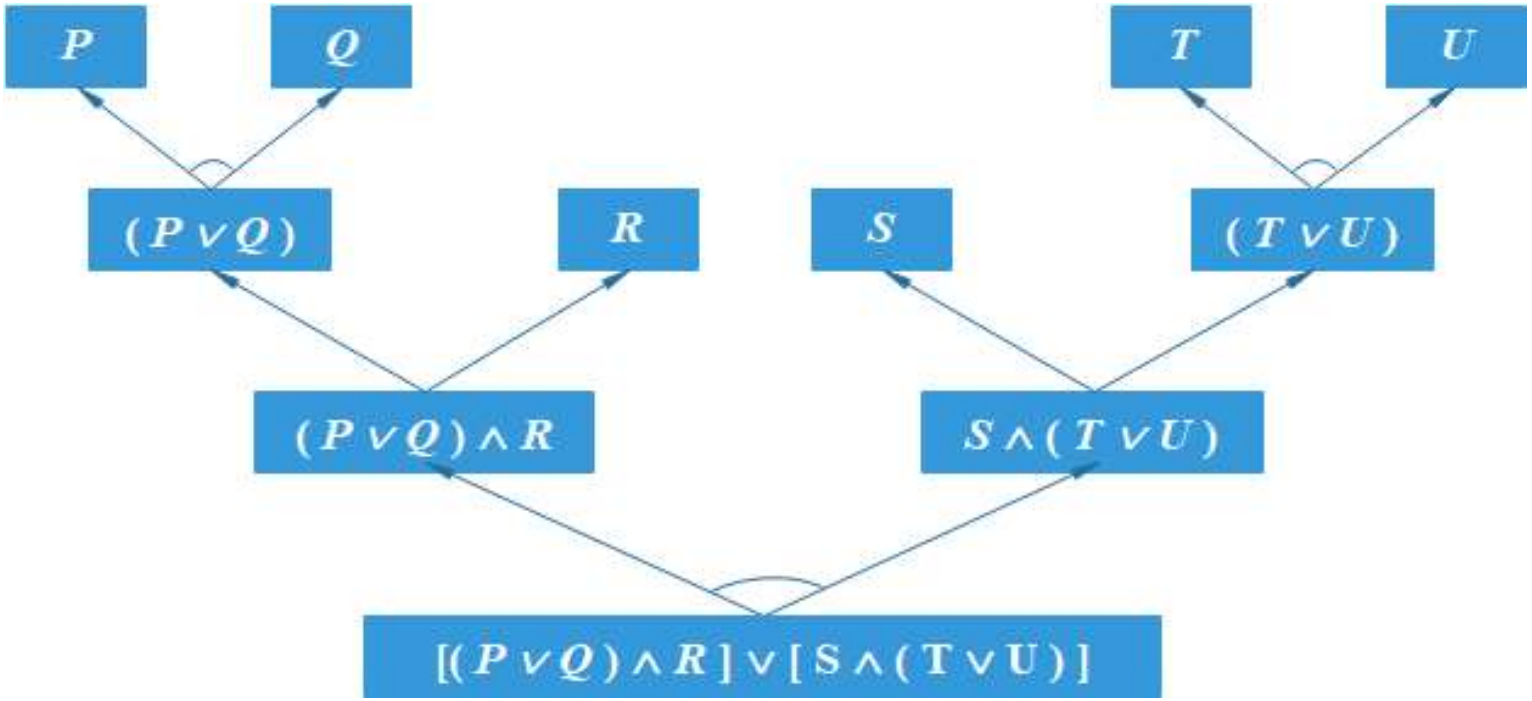
    $$(x)\{(y)(z)[\sim P(x,y,z)]\vee(u)Q(x,u)\}$$
  - (3) 进行 SKolem 化

    $$(x)\{(y)[\sim P(x,y,f(x,y))]\vee(u)Q(x,u)\}$$
  - (4)把所有全称量词移至前面然后消去

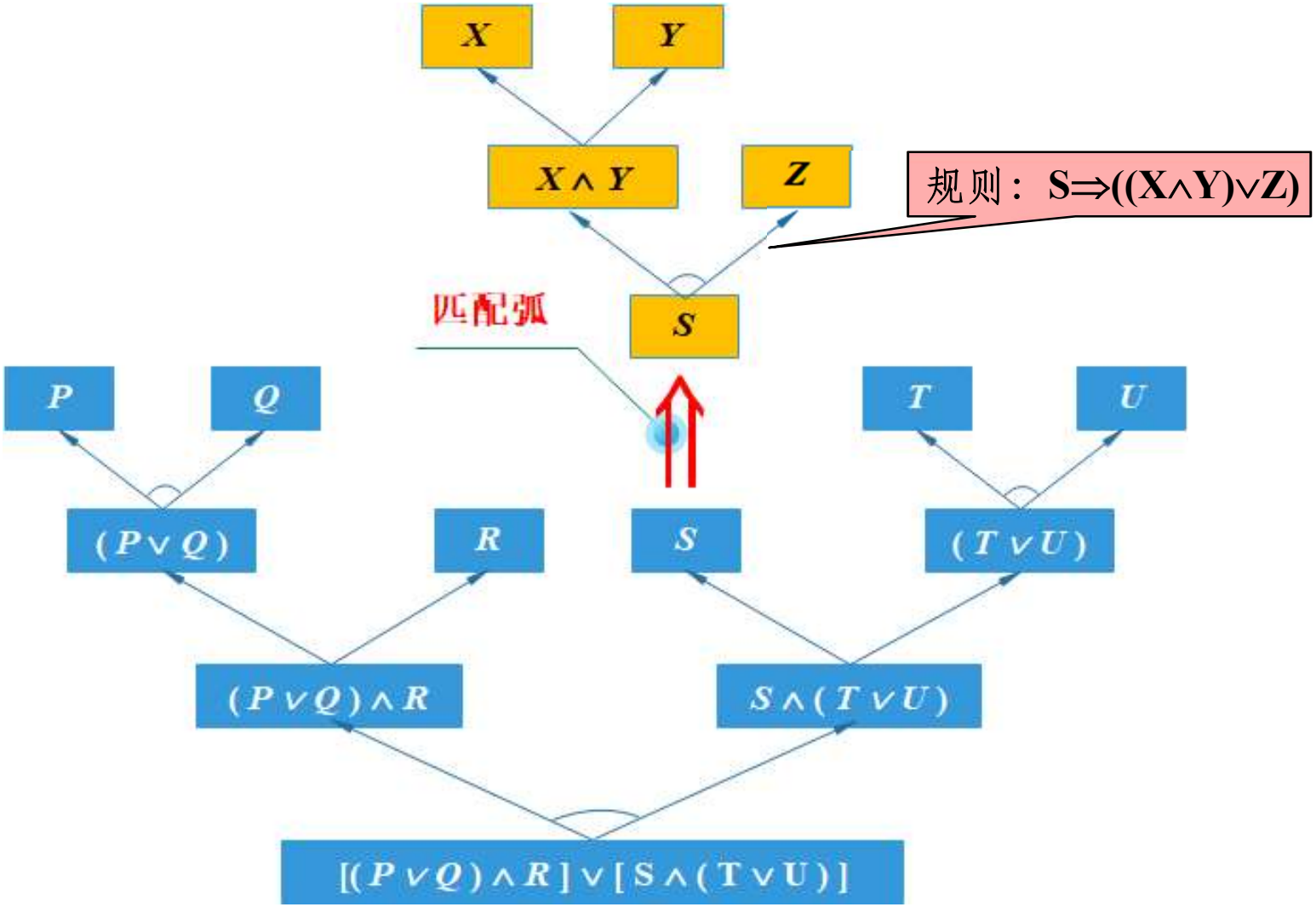
    $$\sim P(x,y,f(x,y)\vee\ Q(x,u)$$
  - (5)恢复蕴涵式

    $$P(x,y,f(x,y)=>Q(x,u)$$

正向演绎推理-无变量与或图



引用一条规则L⇒W得到的与或图



(4)作为终止条件的目标公式

事实：$A \vee B$

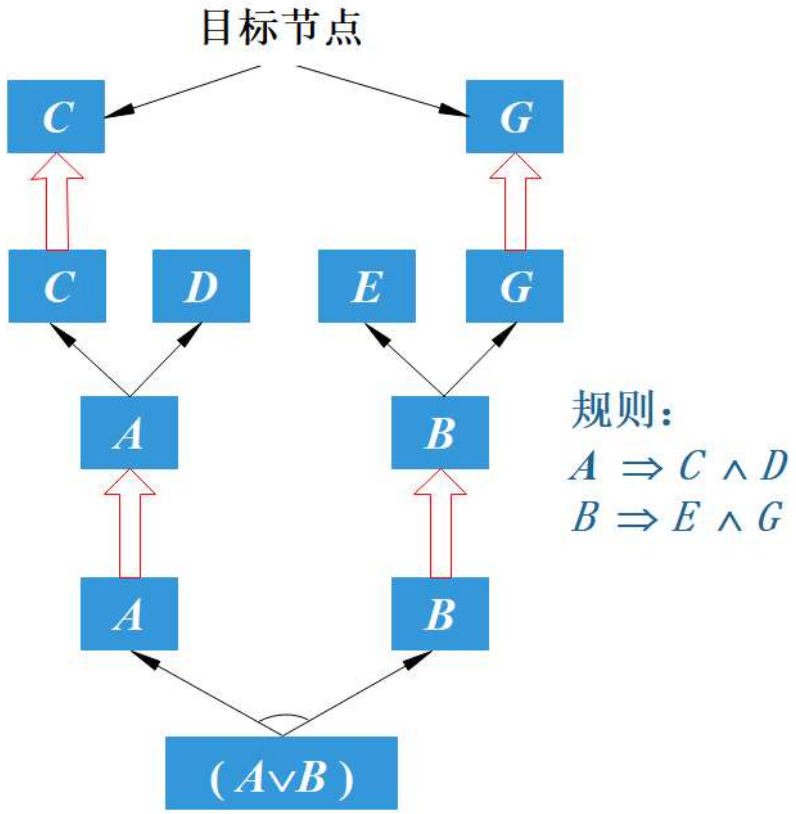
规则：$A \Rightarrow C \wedge D$

$B \Rightarrow E \wedge G$

目标：$C \vee G$

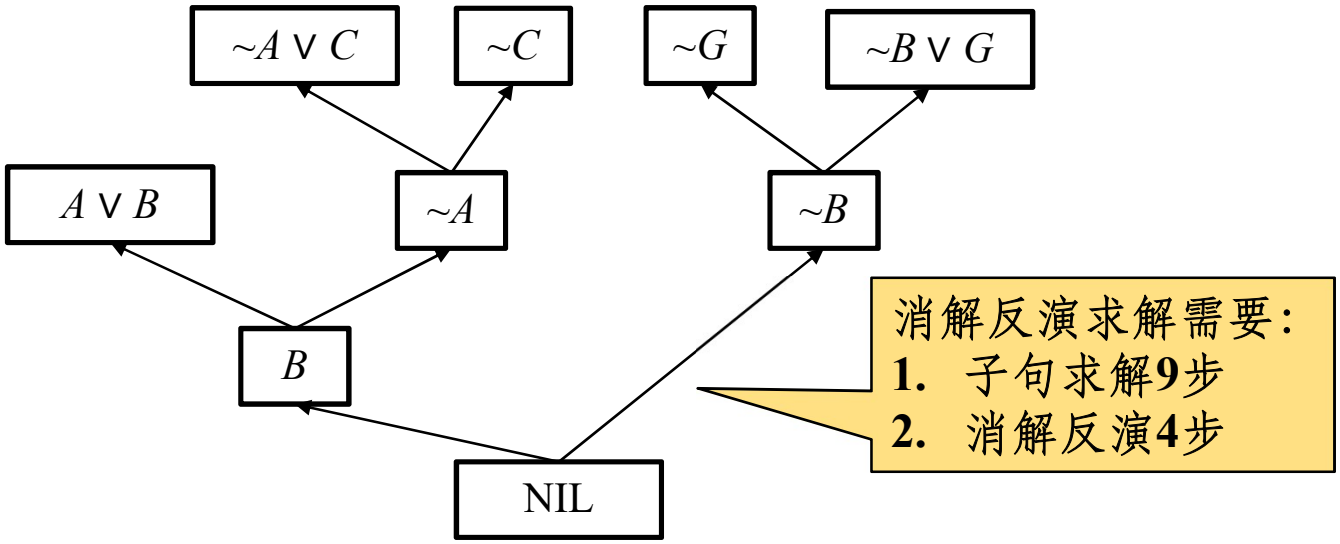
应用正向演绎推理的与或图的目的就是证明某个目标成立

正向演绎系统产生一个含有以目标节点作为终止的一个解图时，证明目标成立



满足终止条件的与或图

思考：为什么正向演绎系统的求解效率比消解原理高?



用消解反演求证目标公式的解图

说明：两次与或图的扩展，需要通过四次消解才能达到同等目标。

规则逆向演绎系统 backward chaining

- **定义**

  逆向规则演绎系统是从**then**向**if**进行推理的，即从目标或动作向事实或状况条件进行推理的。

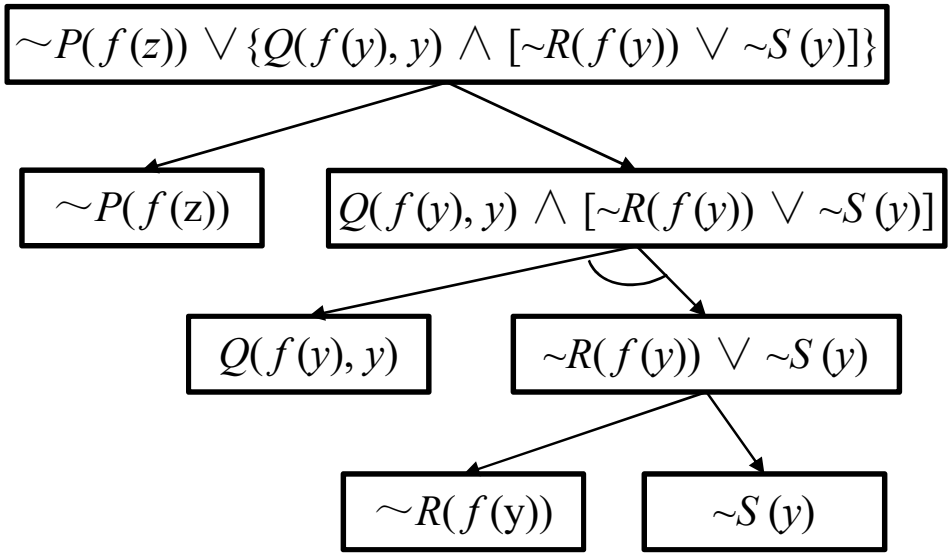
- **求解过程**
  - 目标表达式的与或形式
  - 与或图的**B**规则变换
  - 作为终止条件的事实节点的一致解图

a) 目标表达式的与或形式

目标公式化为与或形：

目标公式： **(∃y)(∀x){P(x)=>[Q(x,y) ∧~[P(x) ∧S(y)]]}**

与或标准形： **~P(f(z)) ∨{Q(f(y),y) ∧[~R(f(y)) ∨ ~s(y)]}**



一个目标公式的与或图表示

b) 与或图的B规则变换

- 我们把允许用作规则的公式类型限制为下列形式：

$$W \Rightarrow L$$

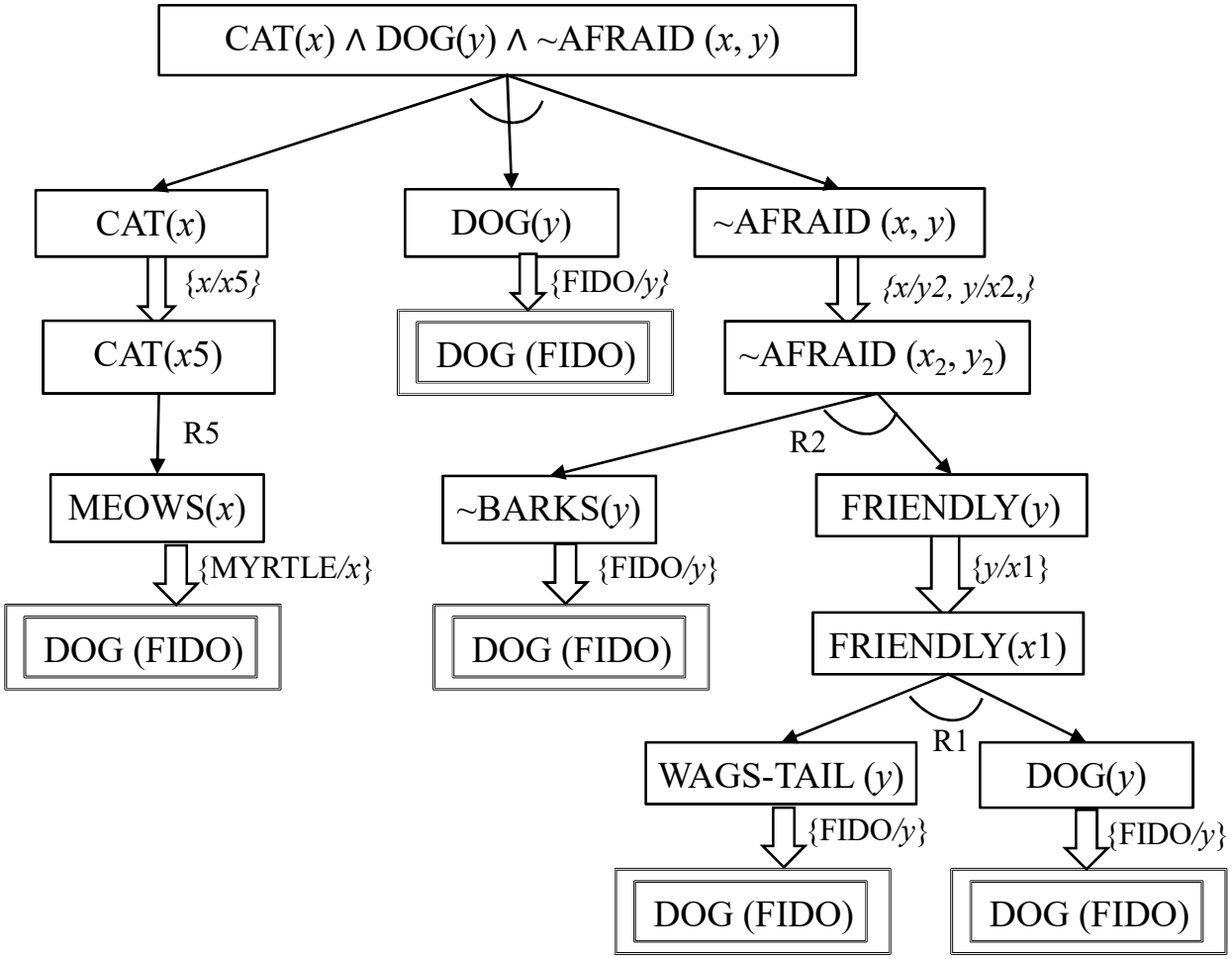
式中，**L**是**单文字**；**W**为**与或形的唯一公式**

- 推理方法就是利用规则扩展与或图的叶节点，不断归约到已知的前提事实。

c) 终止条件

- 逆向演绎系统的成功终止条件是与或图包含有某个终止在事实节点上的一致解图。

例子



逆向系统的一个一致解图

规则双向演绎系统

a)正向演绎系统与逆向演绎系统的局限性

b)规则双向呀演绎系统

正向和逆向组合系统是建立在两个系统相结合的基础上的。此组合系统的总数据库由表示目标和表示事实的两个与或图结构组成。这些与或图结构分别用正向系统的F规则和逆向系统的B规则来修正。

c)规则双向呀演绎系统的复杂性

终止条件：终止涉及两个与或图之间的适当交接处。

- **定义:** 如果 (n ,m) 中有一个为事实节点 , 另一个为目标节点 , 而且如果 n 和 m 都由可 合一的文字所标记 , 或者 n 有一个外向 k 线连接符接至一个后继节点集 (S_i), 使得对此集 的每个元 CANCEL(n ,m) 都成立 , 那么就称这两节点 n 和 m 互相 CANCEL(即互相 抵消) 。

- 一个简单的终止条件是判定在事实与或图叶节点和目标与或图叶节点之间都存在CANCEL关系。

- 当事实图的根节点和目标图的根节点互相 CANCEL 时 , 就得到一个候补解。在事 实和目标图内证明该目标根节点和事实根节点互相 CANCEL 的图结构叫做候补 CANCEL 图。如果候补 CANCEL 图中所有匹配的 mgu 都是一致的 , 那么这个候补解就是一个实际解。

小结

- 对于许多复杂的系统和问题，采用搜索推理方法很难甚至无法使问题得到解决
- 规则演绎系统、产生式系统、不确定推理等是更先进的推理技术和系统，适合求解更复杂的问题
- 对于发展更快，适合解决更为复杂的问题的技术则包括计算智能、机器学习、自动规划系统等

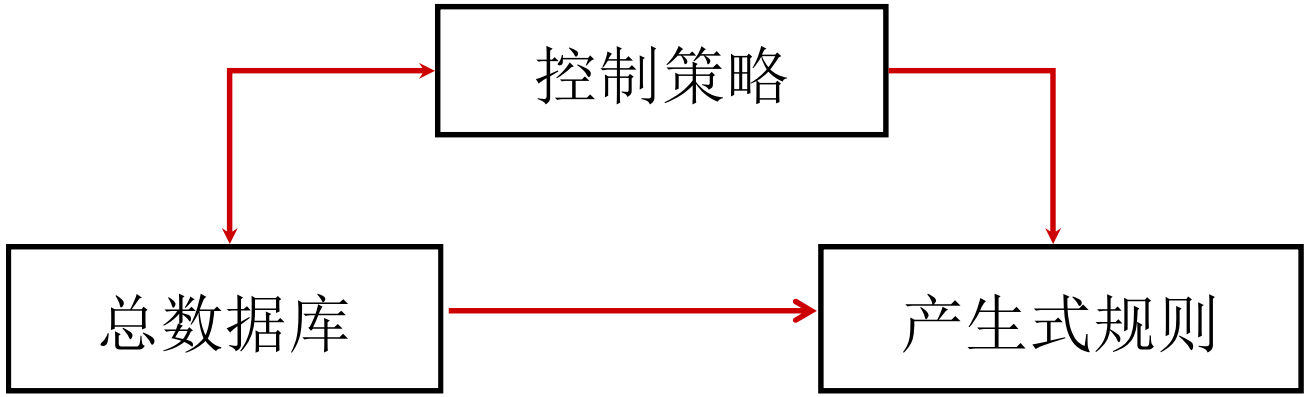
目录

- 图搜索策略
- 盲目搜索
- 启发式搜索
- 消解原理
- 规则演绎系统
- 产生式系统
- 非单调推理
- 小结

产生式系统Production System

- **定义：**
  - 用来描述若干个不同的以一个基本概念为基础的系统。这个基本概念就是产生式规则或产生式条件和操作对的概念。
- **实质：**
  - 在产生式系统中，论域的知识分为两部分：用**事实**表示静态知识，如事物、事件和它们之间的关系；用**产生式规则**表示推理过程和行为。由于这类系统的知识库主要用于存储规则，因此又把此类系统称为**基于规则的系统**。

产生式系统的组成



□ 控制策略：选择规则到执行操作的步骤

□ 1 匹配

把当前数据库与规则的条件部分相匹配。若完全匹配，则启用该规则。

□ 2 冲突

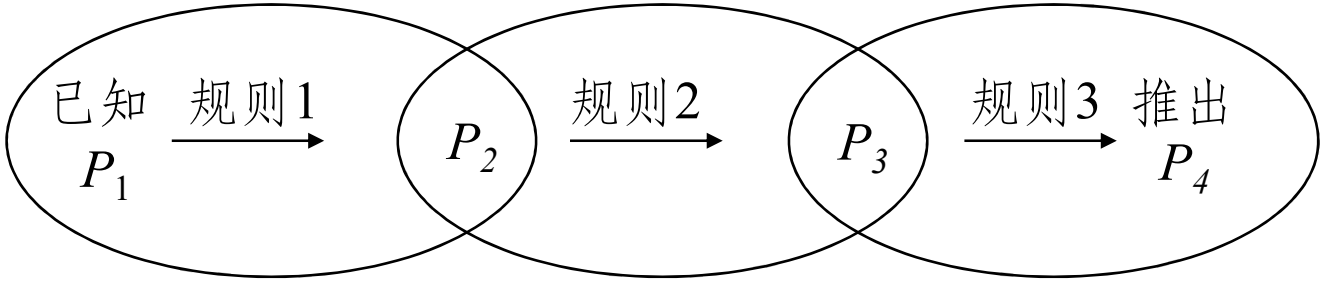
当有一条以上规则的条件部分和当前数据库相匹配时，就需要决定首先使用哪一条规则，这称为冲突解决。如专一性排序、规则排序、数据排序、就近排序等。

□ 3 操作

操作就是执行规则的操作部分，经过操作后，总数据库被修改，并记住所应用的规则序列，以便给出解答路径。若满足结束条件，推理停止。

产生式系统的推理

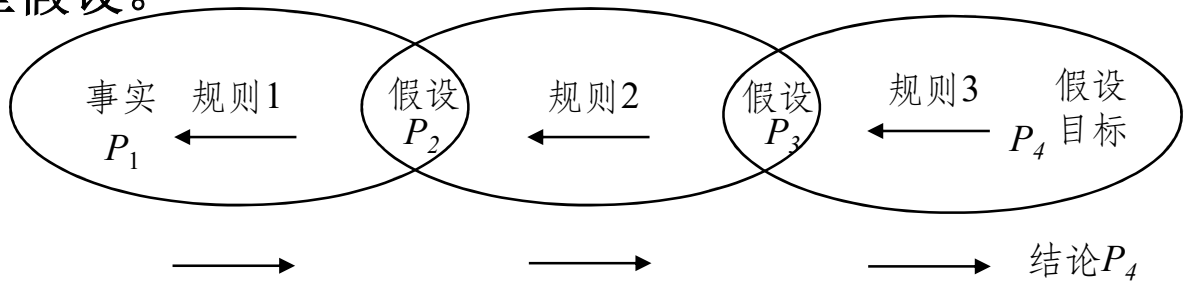
- **正向推理：从一组表示事实的谓词或命题出发，使用一组产生式规则，用以证明该谓词公式或命题是否成立。**
  - 例如：规则集合R1～R3
    - R1：P1－>P2
    - R2：P2－>P3
    - R3：P3－>P4



正向推理过程

□ 逆向推理：

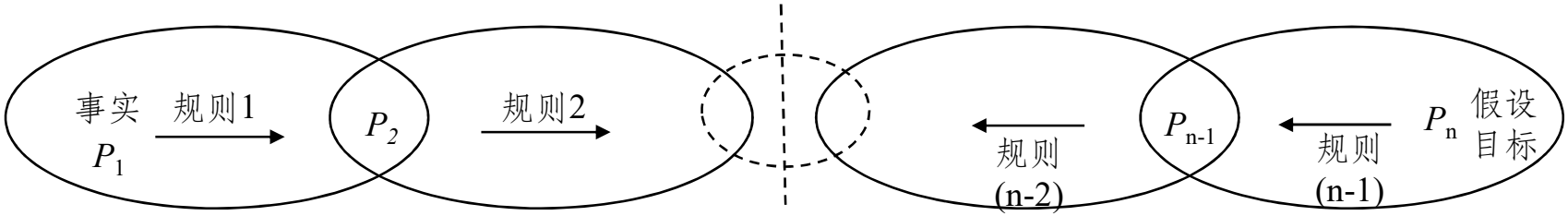
- 从表示目标的谓词或命题出发，使用一组产生式规则证明事实谓词或命题成立，即首先提出一批假设目标，然后逐一验证这些假设。



正、逆向推理过程

项目	正向推理	逆向推理
驱动方式	数据驱动	目标驱动
推理方法	从一组数据出发向前推到结论	从可能的解答出发，向后推理验证解答
启动方法	从一个事件启动	由询问关于目标状态的一个问题而启动
透明程度	不做解释其推理过程	可解释其推理过程
推理方向	由底向上推理	由顶向下推理
典型系统	CLPS,OPS	PROLOG

- **双向推理：**双向推理的推理策略是同时从目标向事实推理和从事实向目标推理，并在推理过程中的某个步骤，实现事实与目标的匹配。
- **实例：美国斯坦福研究所的专家系统工具KAS，就是采用正向、逆向混合推理的产生式系统。**



双向综合推理过程

- **举例：参考教材，产生式系统实例**

目录

- 图搜索策略
- 盲目搜索
- 启发式搜索
- 消解原理
- 规则演绎系统
- 产生式系统
- 非单调推理
- 小结

非单调推理（自学）

□ 定义

非单调推理(例如：谓词逻辑)用来处理那些不适合用谓词逻辑表示的知识。它能够较好地处理不完全信息、不断变化的情况以及求解复杂问题过程中生成的假设，具有较为有效的求解效率。

缺省推理

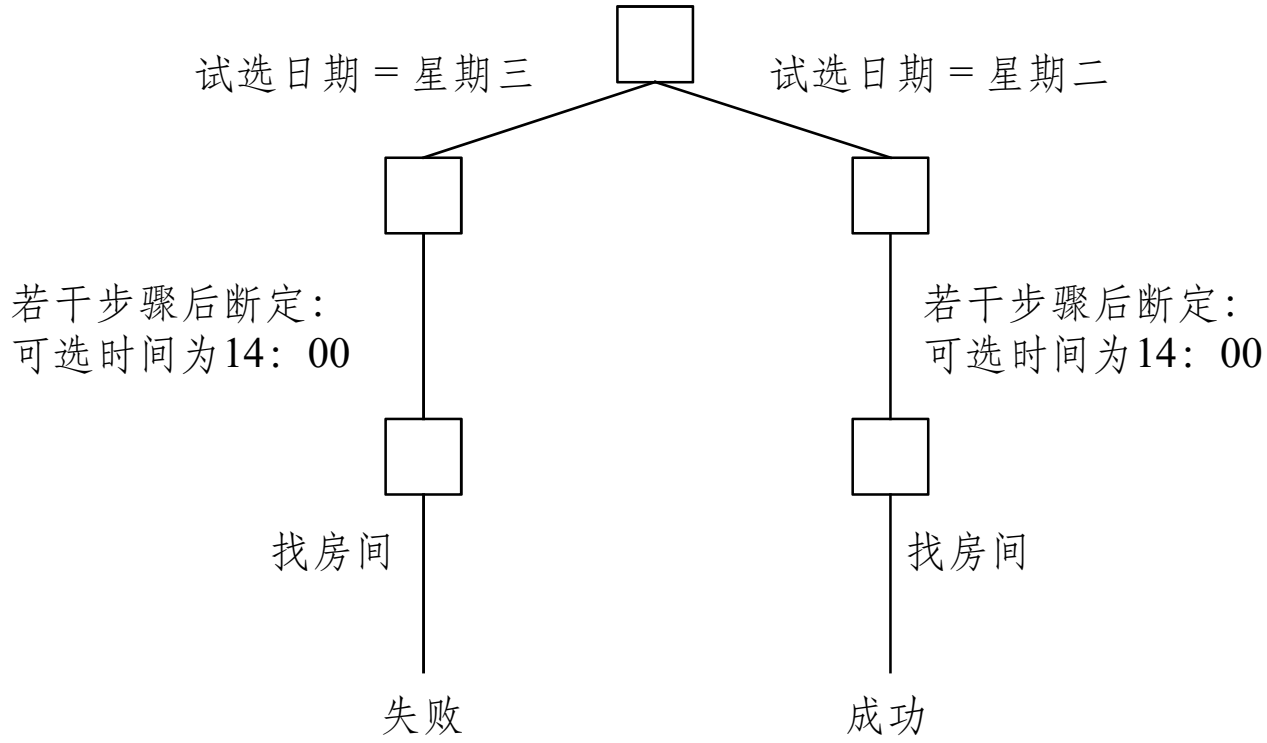
- **定义1：** 如果X不知道，那么得结论Y。
- **定义2：** 如果X不能被证明，那么得结论Y。
- **定义3：** 如果X不能在某个给定的时间内被证明，那么得结论Y。
- **基于下述一些原因, 非单调推理系统是必要的：**

(1) 不完全知识的出现要求缺省推理。

(2) 一个不断变化的世界必须用适应不断变化的数据库来描述。

(3) 产生一个问题的完全解可能要求关于部分解的暂时的假设。

会议安排的非单调推理实例：



非面向从属关系的回溯

非单调推理系统

- **正确性维持系统（TMS，又称真值维持系统）**
  - 用以保持其它程序所产生的命题之间的相容性。一旦发现某个不相容，它就调出自己的推理机制，面向从属关系的回溯，并通过修改最小的信念集来消除不相容。
- **在TMS中，每一个命题或规则均称为节点，且对任一节点，以下两种状态必居其一：**

IN	相信为真
OUT	不相信为真，或无理由相信为真，或当前没有相信的理由

- **(1) 支持表**
  - 支持表：（SL（IN-节点）（OUT-节点））
  - 例如：下列节点

（1）现在是冬天（SL（）（））

（2）天气是寒冷的（SL（1）（3））

（3）天气是温暖的（SL（）（））

- **(2) 条件证明**
  - 条件证明：（CP（结论）（IN-节点）（OUT-节点））

- **例如：**
  - （1）日期（会议）＝ 星期三　（SL（）（2））
  - （2）日期（会议）≠ 星期三　（SL（5）（））
  - （3）时间（会议）＝14：00　（SL（57，103，45）（））
  - （4）矛盾　（SL（1，3）（））
  - （5）不相容N-1　（CP4（1，3）（））

目录

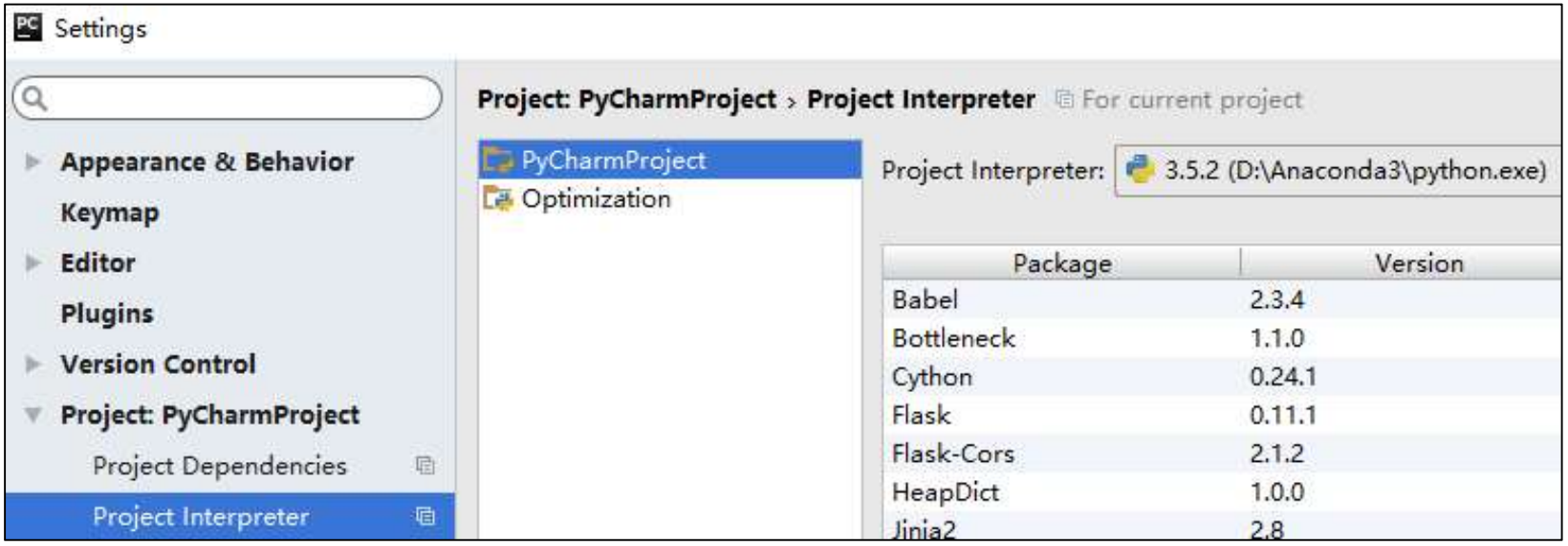
- 图搜索策略
- 盲目搜索
- 启发式搜索
- 消解原理
- 规则演绎系统
- 产生式系统
- 非单调推理
- 小结

小结

- **经典搜索推理技术**
  - 图搜索技术
    - 盲目搜索
    - 启发式搜索
  - 消解反演
- **高级搜索推理技术**
  - 规则演绎系统
  - 产生式系统
  - 非单调推理

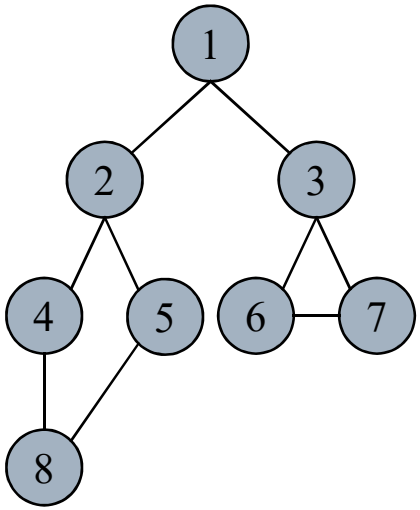
实验1：Python

- 建议使用Python 3.5.x
- IDE可选用Spider、PyCharm
  - 可安装Anaconda3（其中包含了Python 3.5和编辑器Spider，以及常用的包）
  - 使用PyCharm可将Interpreter（解释器）设置为Anaconda的Python3.5

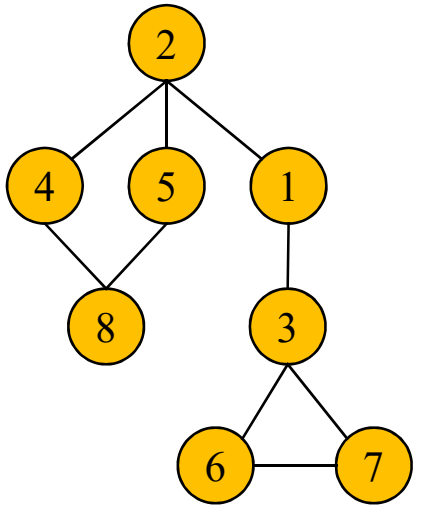


实验2：BFS & DFS

```
def main():
    #定义节点并存储于数组nodes中
    nodes = [i+1 for i in
range(8)]
    #定义边并存储于数组sides中
    sides=[(1, 2),   #边
          (1, 3),
          (2, 4),
          (2, 5),
          (4, 8),
          (5, 8),
          (3, 6),
          (3, 7),
          (6, 7)]
    G = Gragh(nodes,sides)
    print("深度优先：")
    print(G.DFS(1))
    print("宽度优先：")
    print(G.BFS(1))
```



从节点1开始
深度优先
[1, 3, 7, 6, 2, 5, 8, 4]
宽度优先
[1, 2, 3, 4, 5, 6, 7, 8]



从节点2开始
深度优先:
[2, 5, 8, 4, 1, 3, 7, 6]
宽度优先:
[2, 1, 4, 5, 3, 8, 6, 7]

代码

- 参见“2018-春季-人工智能-No03-Topic 04-确定性推理-BFS-DFS实验Python代码.docx”