

产生式系统的搜索(1)

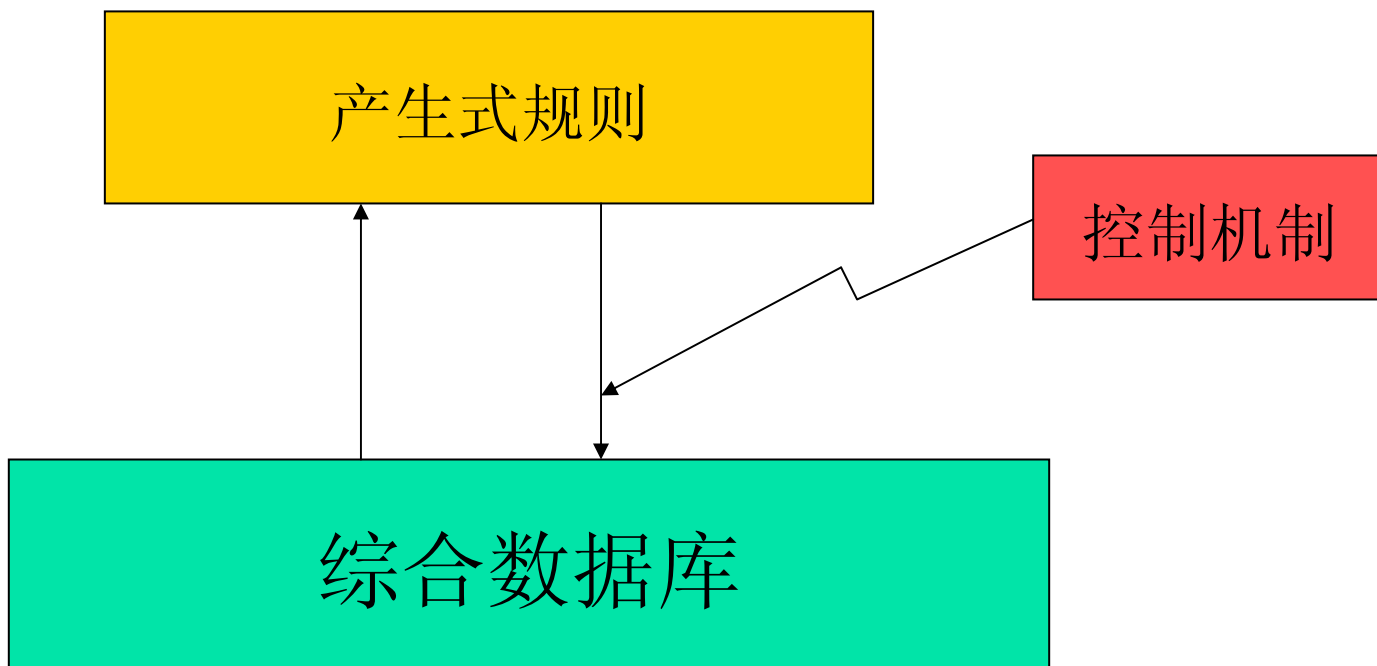
张文生 研究员/教授

中国科学院自动化研究所

内容

- 回溯策略
- 图搜索
- 无信息搜索
- 启发式搜索(**A***)
- **A***算法的可采纳性

回顾



- 控制机制

- 控制策略

- 激励---点燃

- 两类

- 不可撤回的控制策略:

- 试探性控制策略

- 回溯型

- 图搜索

- 具体手段

- 冲突删除策略

状态

- 任一时刻, 综合数据库的情况;

2	3	7
	5	1
4	8	6

$\{A, B, C, D\}$

$(c, a, b, 0, 0)$

状态空间

- 状态空间

- 所有可能的状态的全体。

2	3	7
	5	1
4	8	6

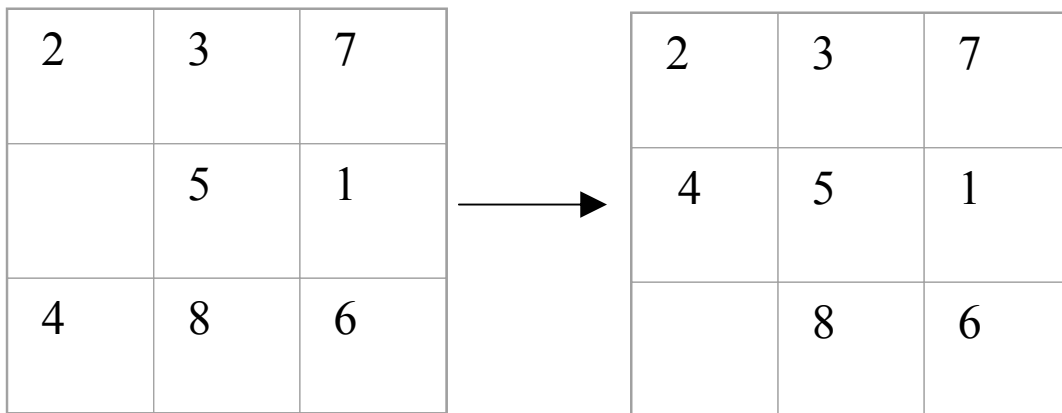
5	8	6
	1	2
7	4	3

.....

1	2	4
	6	5
7	8	3

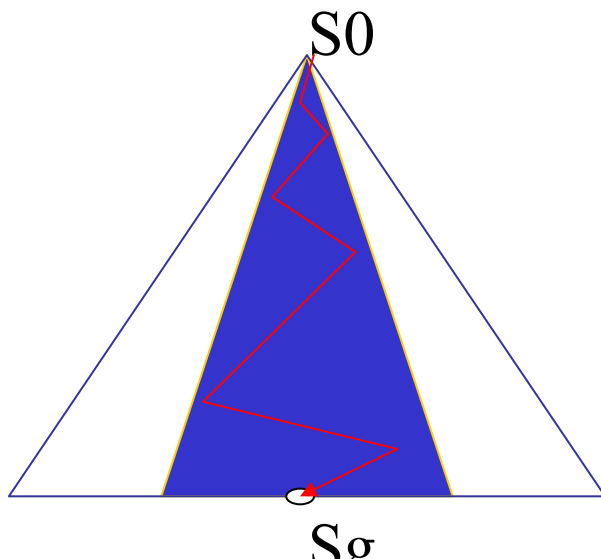
状态转移

- 初始状态
- 目标状态
- 状态转移
 - 规则



搜索(search)

- 路径
 - 状态序列
- 搜索
 - 寻找从初始状态到目标状态的路径;



搜索的必要性

- **AI为什么要研究search?**
 - 问题没有直接的解法;
 - 解方程组;
 - 定理证明;
 - 需要探索地求解;

搜索与检索的区别

- 状态是否动态生成;
 - 检索: 静态;
 - 在数据库中检索某人的纪录;
 - 搜索: 动态生成;
 - 下棋

几个问题

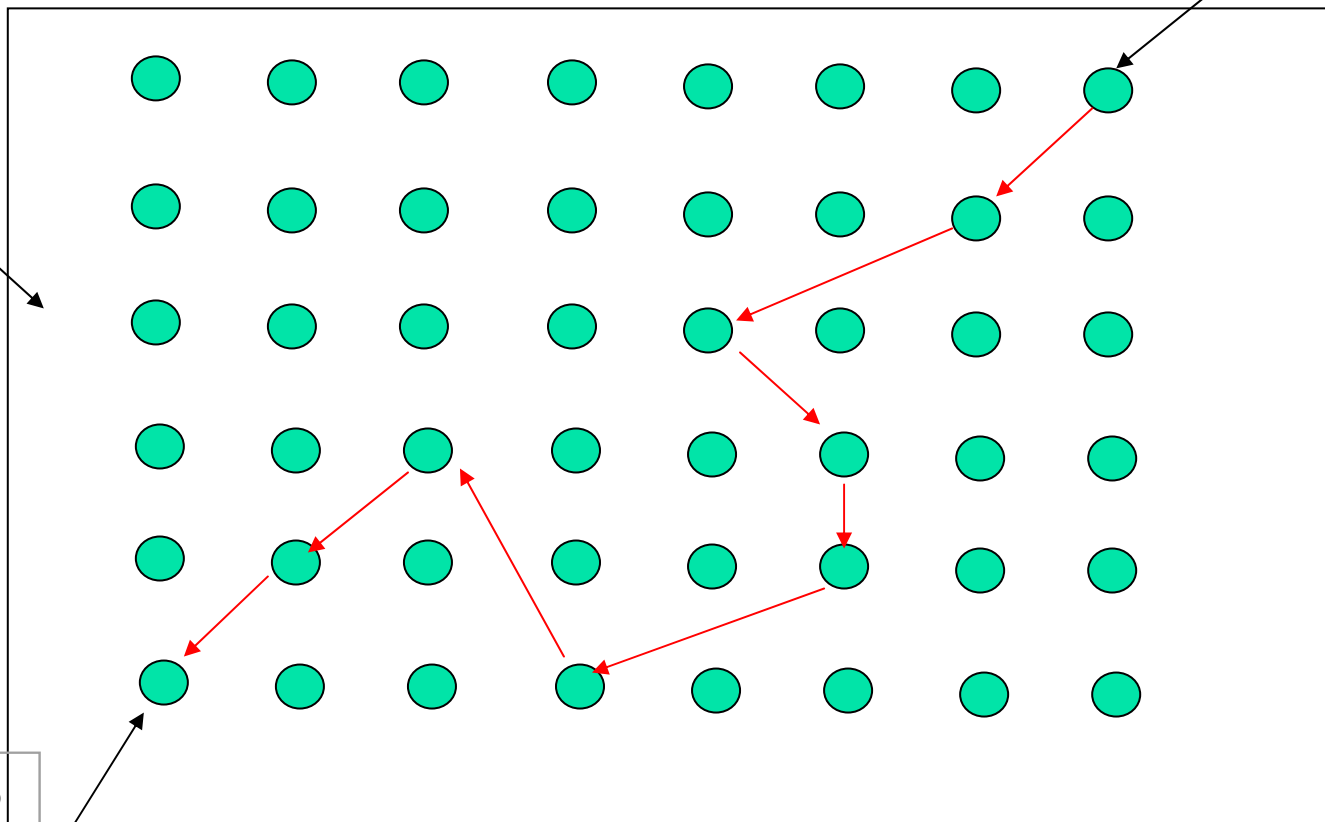
- 目标状态是否确定?
 - 确定: 定理证明, **eight-puzzle**
 - 不确定: 求积分, 下棋;
 - 确定目标的性质;
- 问题的解: 路径(解路径)/ 目标状态;
 - 需要路径: 下棋
 - 不需要路径: 电路设计
 - 需要/不需要: 诊病
- 约束条件
 - 目标状态不确定时, 用来约束目标状态的性质;
 - **$X+Y=4$** : 非整数解/ 整数解

- 多解性;
 - **$X+Y=4$** :整数解
- 最优解
 - 评价标准/判断准则;
 - **$\min(x*y)$**
 - 北京->上海: 时间最短/费用最少
- 最优解是否唯一?
 - 下棋

搜索问题

状态空间

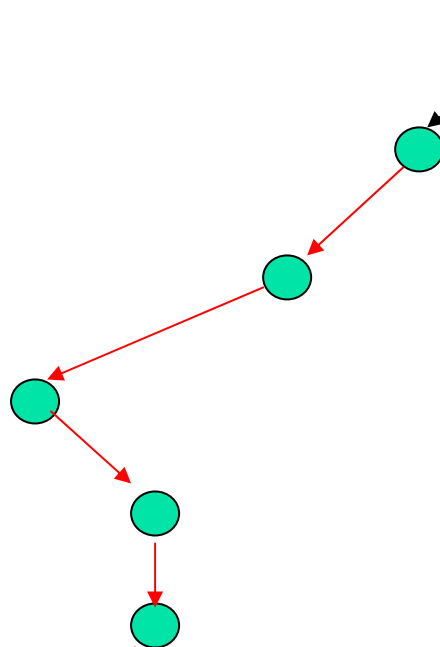
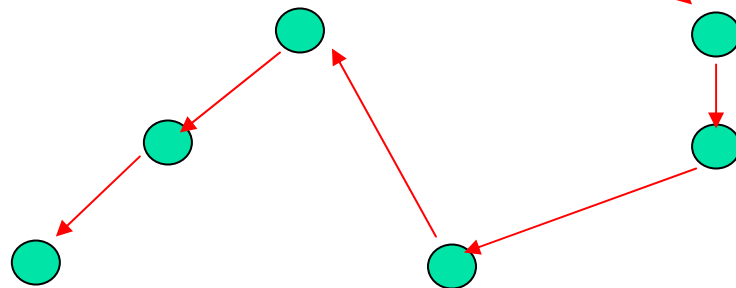
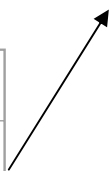
2	3	7
	5	1
4	8	6



1	2	3
8		4
7	6	5

搜索不是检索

1	2	3
8		4
7	6	5

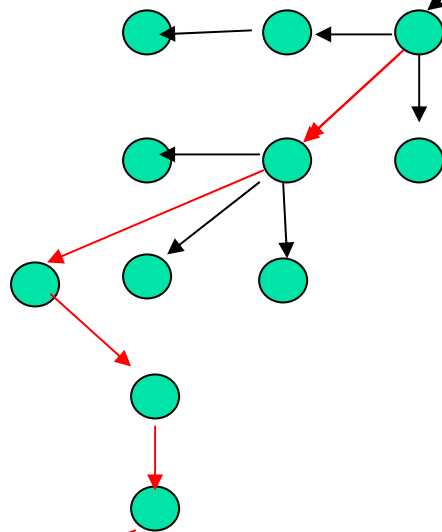
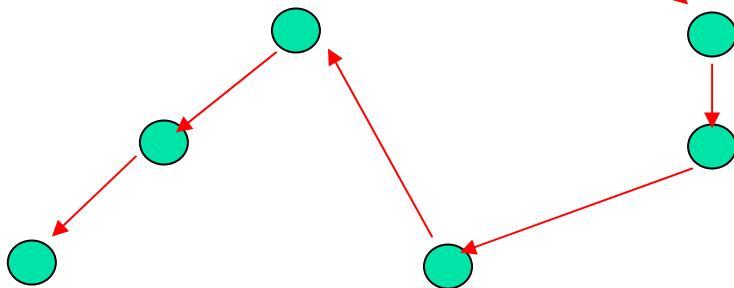


2	3	7
	5	1
4	8	6



难点

1	2	3
8		4
7	6	5

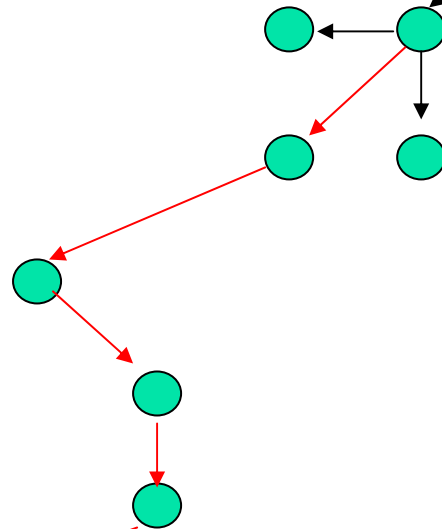
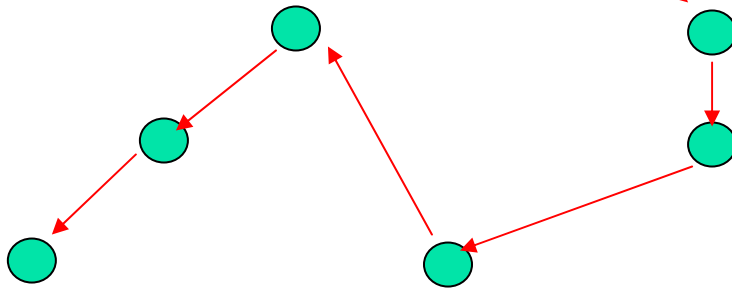


2	3	7
	5	1
4	8	6



启发式方法

1	2	3
8		4
7	6	5

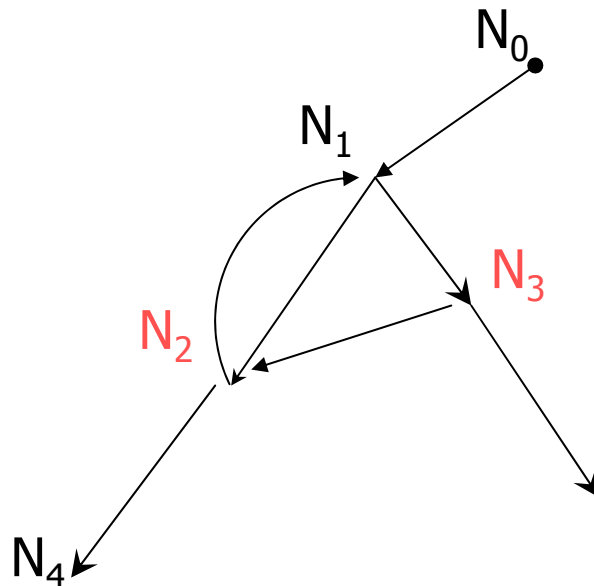
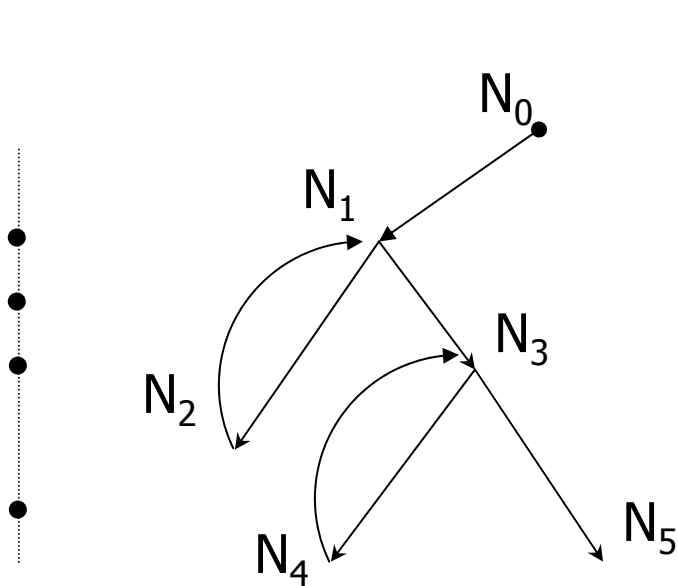


2	3	7
	5	1
4	8	6



控制策略

- 不可撤回的控制策略;
- 试探性控制策略
 - 回溯型
 - 图搜索



不可撤回的控制策略

- 例子: **eight-puzzle**

- 评价函数:**f**

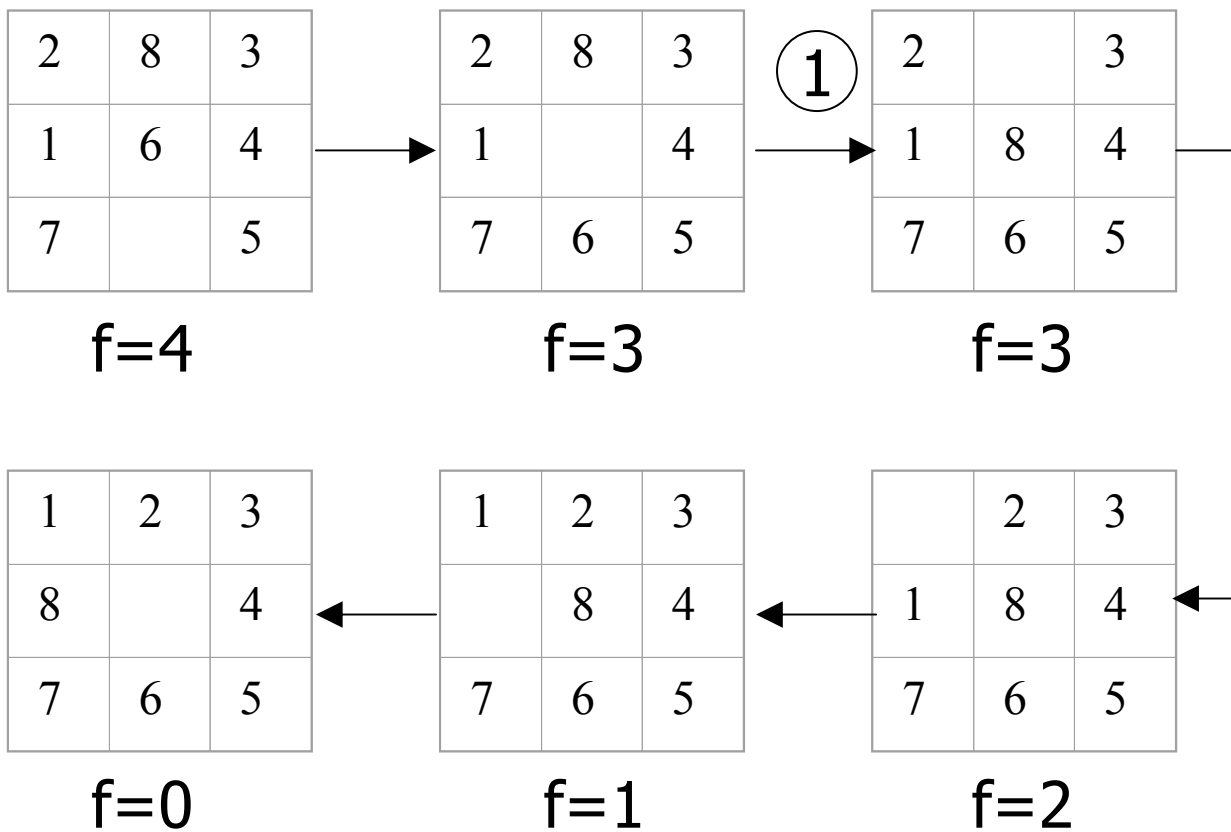
2	8	3
1	6	4
7		5

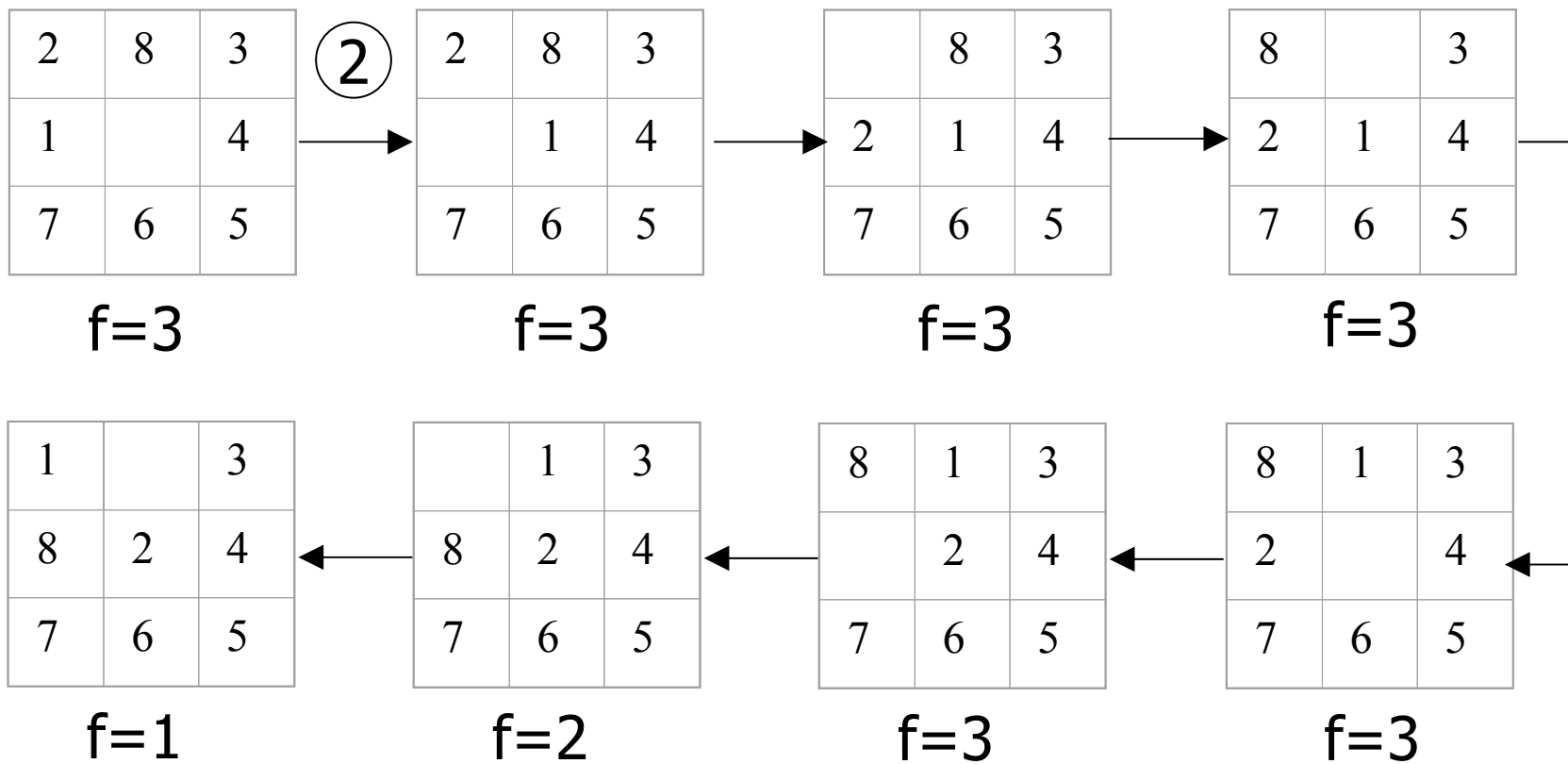
与

1	2	3
8		4
7	6	5

的差异为**4**

规定: 评价函数非增;





可能无解

1	2	5
	8	4
7	6	3

$f=2$

1	2	3
	8	4
7	6	5

目标

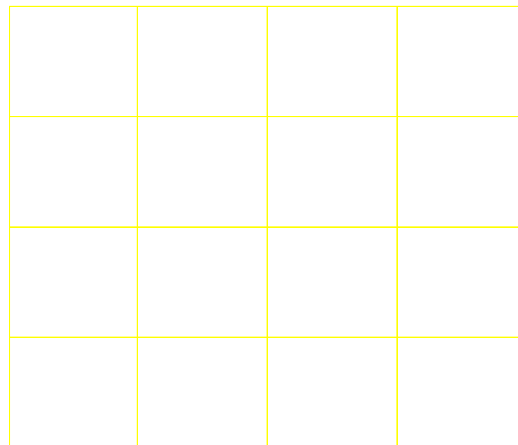
- 回溯策略
- 图搜索
- 无信息搜索
- 启发式搜索
- A*算法的可采纳性

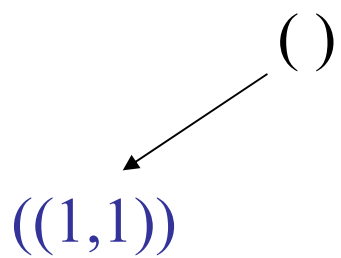
回溯策略

- 例：四皇后问题

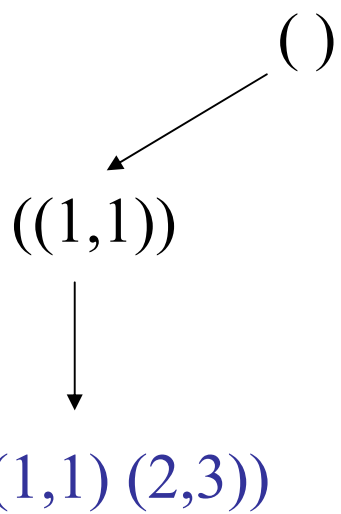
	Q		
			Q
Q			
		Q	

()

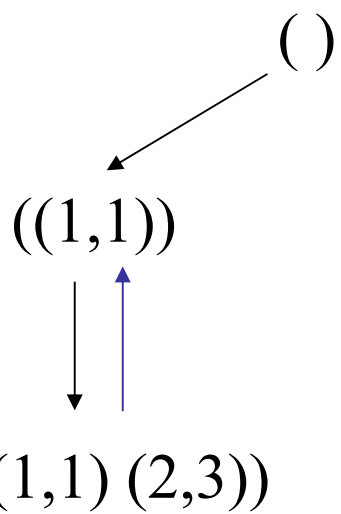




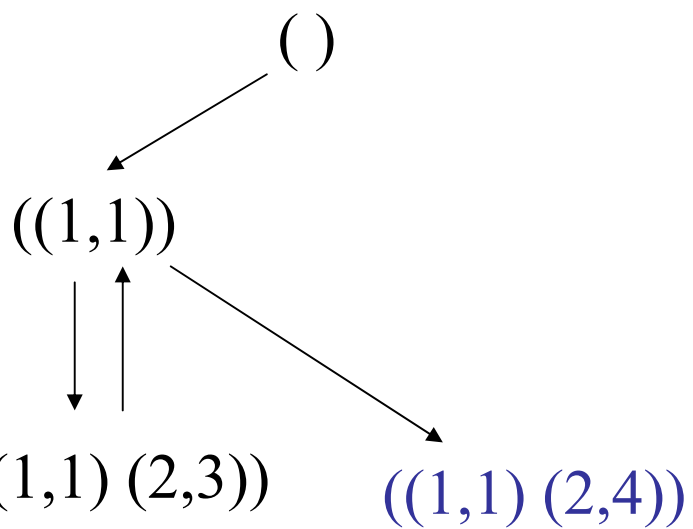
Q			



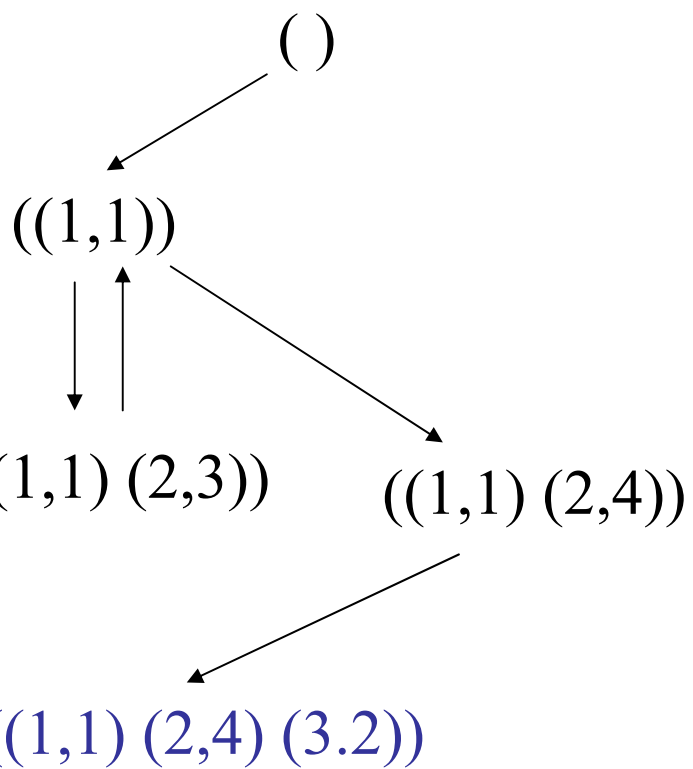
Q			
		Q	



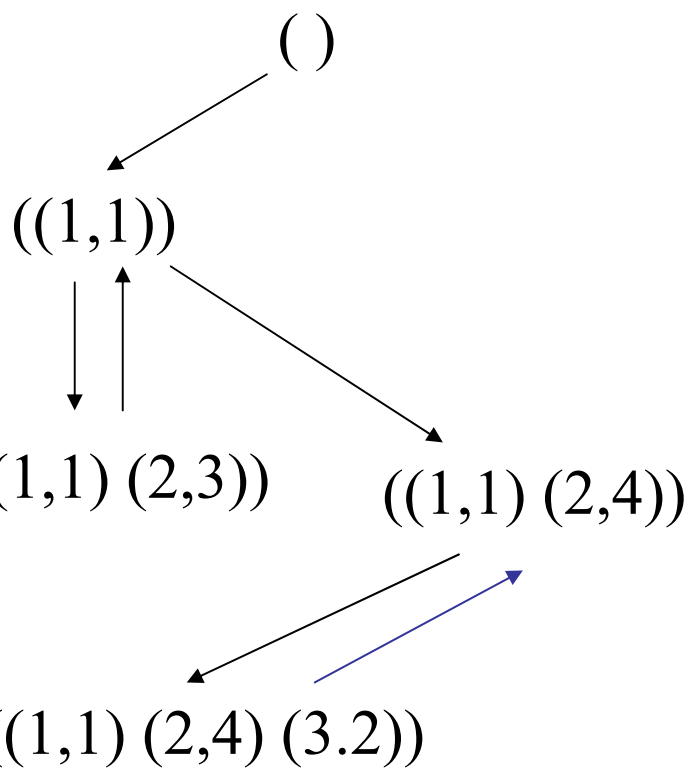
Q			



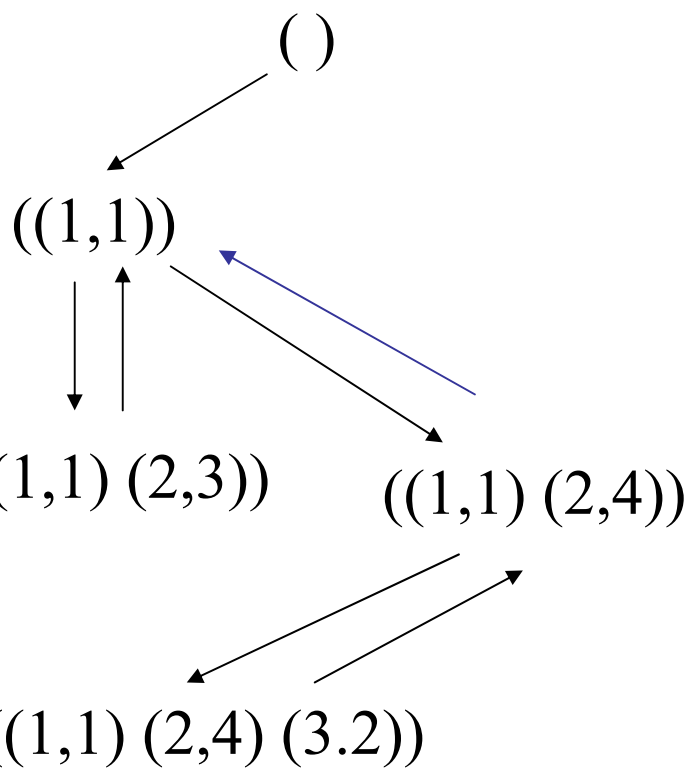
Q			
			Q



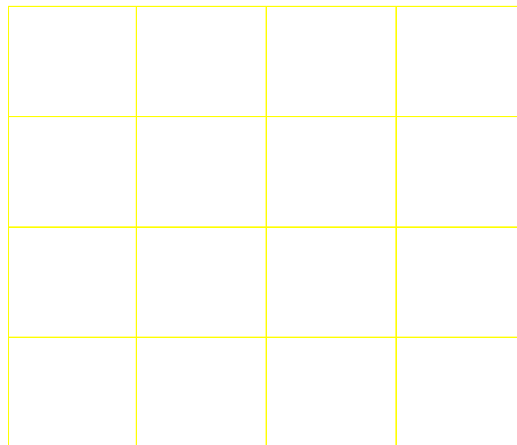
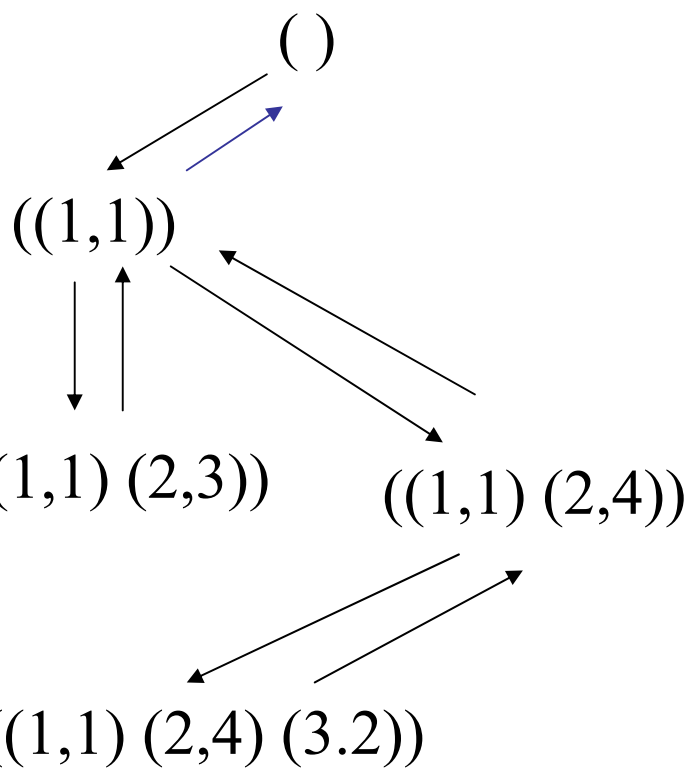
Q			
			Q
	Q		

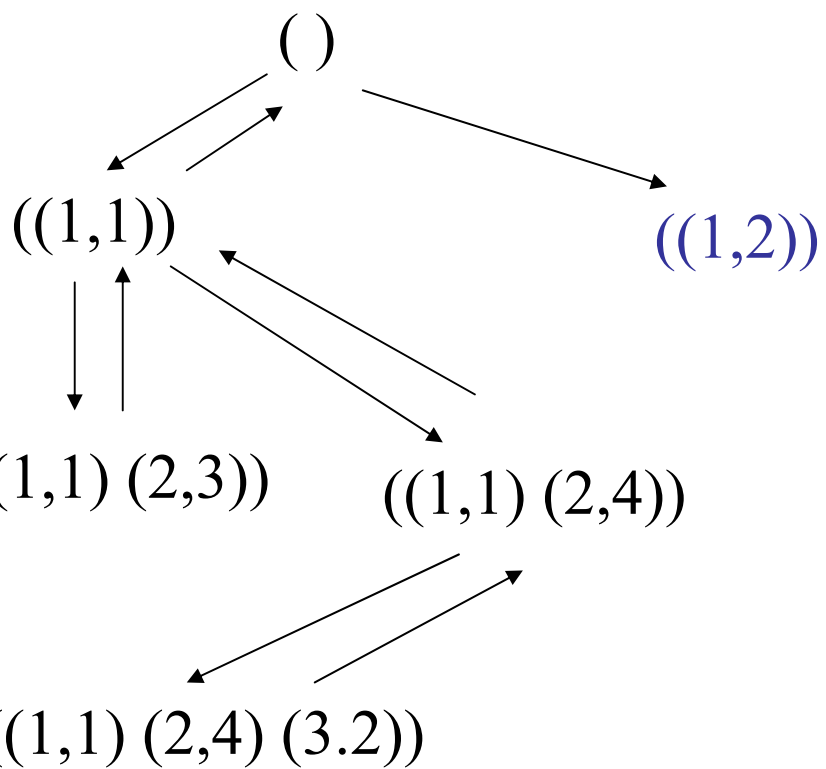


Q			
			Q

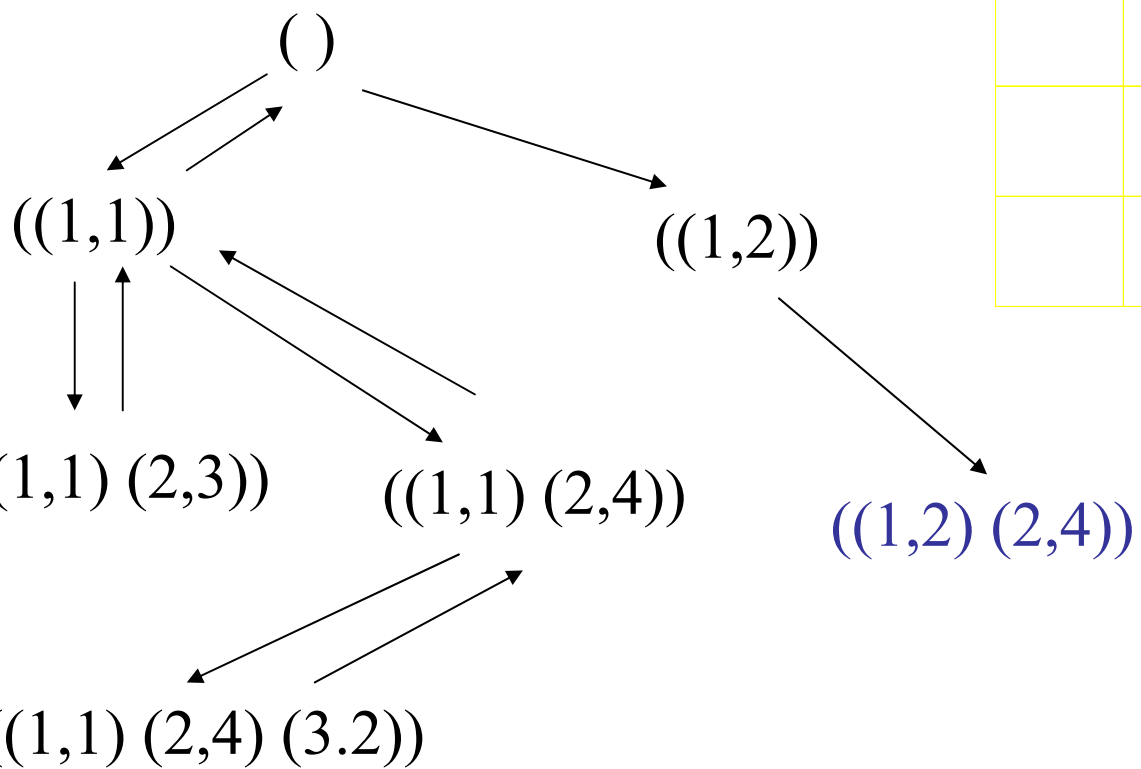


Q			

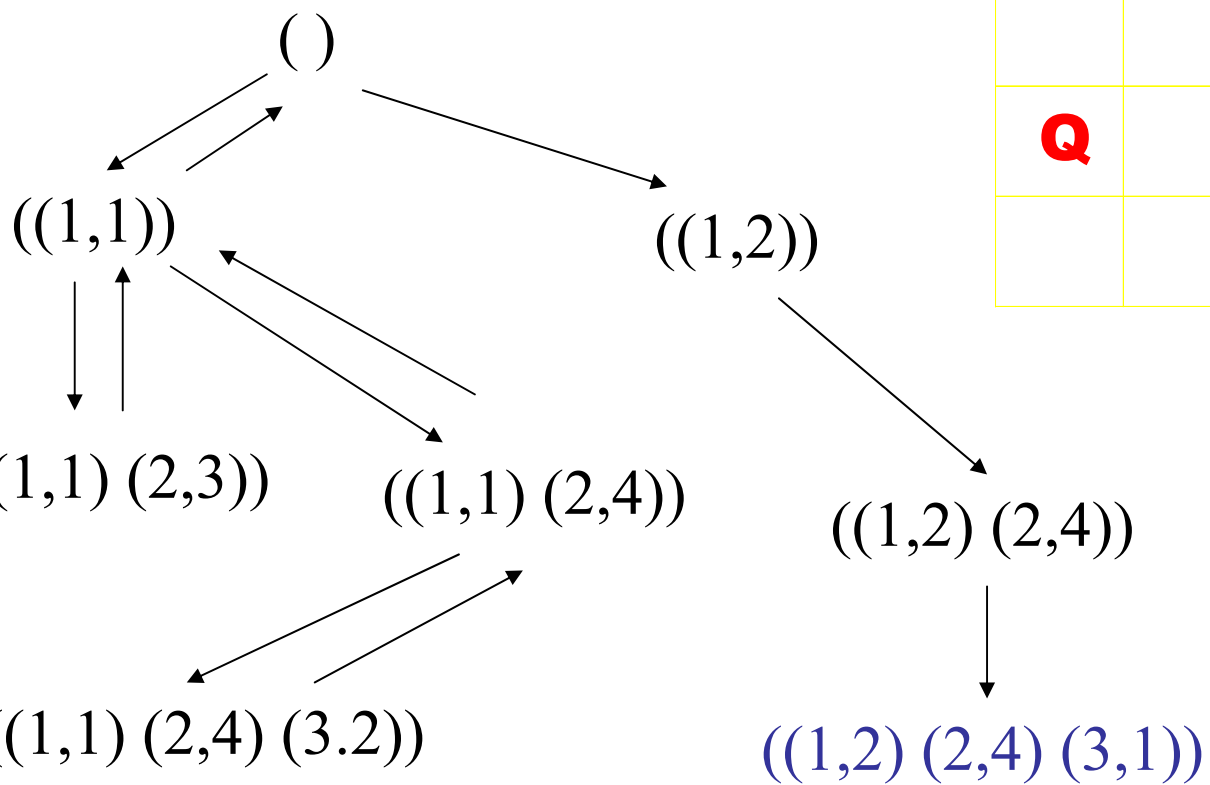




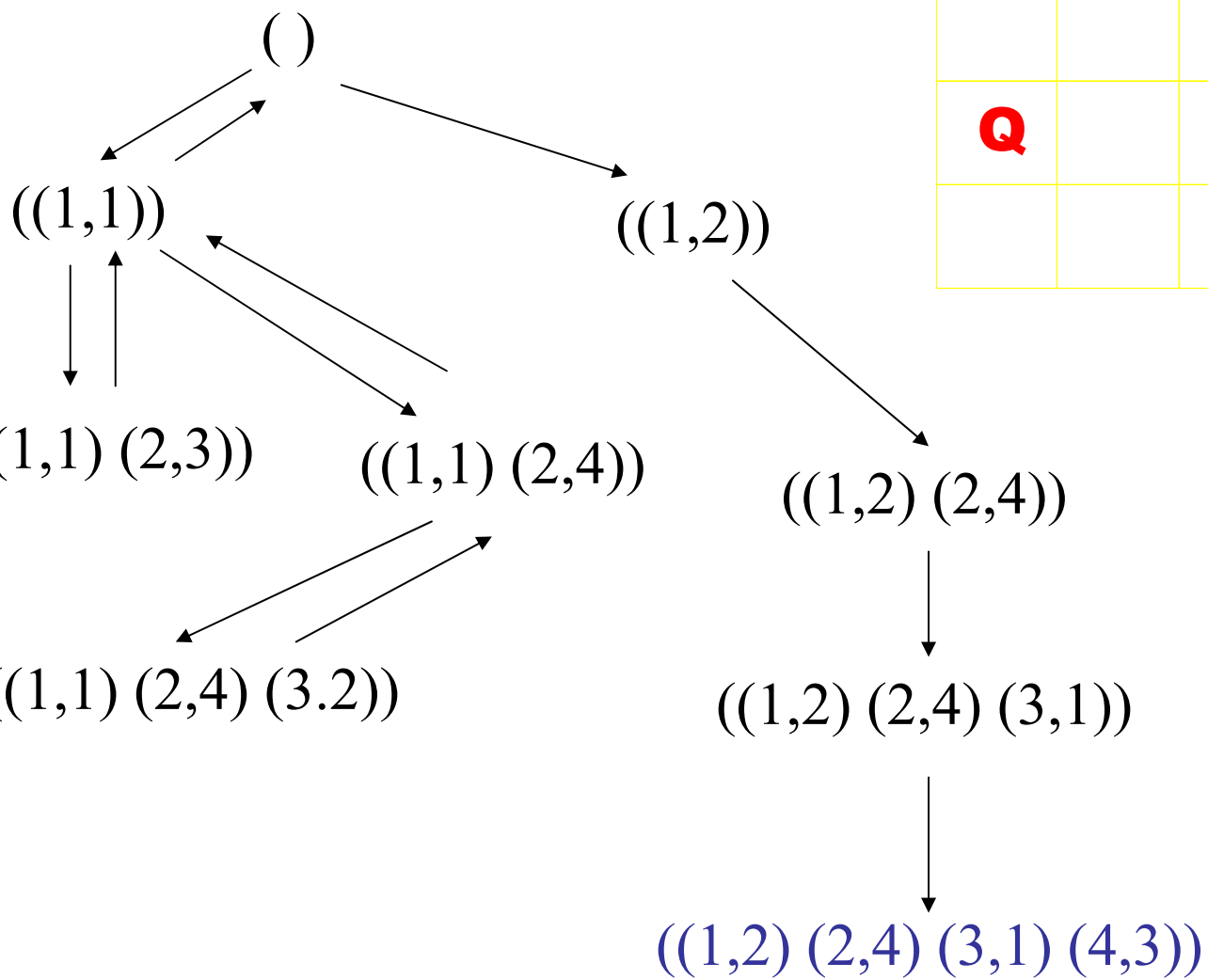
	Q		



	Q		
			Q



	Q		
			Q
Q			



	Q		
			Q
Q			
		Q	

回溯搜索算法

BACKTRACK (DATA)

DATA: 当前状态。

返回值：从当前状态到目标状态的路径
(以规则表的形式表示)
或**FAIL**。

回溯搜索算法

BACKTRACK(DATA)

```
1      IF Term(DATA)      RETURN NIL;
2      IF Deadend(DATA) RETURN FAIL;
3      Rules:=Apprules(DATA);
4  LOOP: IF Null(Rules) RETURN FAIL;
5      R:=First(Rules);
6      Rules:=Tail(Rules);
7      Rdata:=Gen(R, DATA);
8      Path:=BACKTRACK(Rdata);
9      IF Path =FAIL GO LOOP;
10     Else RETURN Cons(R, Path);
```

分析节点的情况

- **失败节点**：返回**FAIL**
 - 步骤**2**：领域相关条件判断
 - 步骤**4**：无规则可用时
- **成功节点**
 - 步骤**1**：叶节点, 返回**NIL**
 - 步骤**10**：中间节点, 返回包含**R**的路径
- 如果成功，返回一条包含**R**的路径, $(R_{i1}, R_{i2}, \dots, R_{in})$

存在问题及解决办法

- 问题和解决方法:
 - 深度问题
 - 对搜索深度加以限制
 - 死循环问题
 - 状态重复: **$A \rightarrow B, B \rightarrow C, C \rightarrow A$**
 - 记录从初始状态到当前状态的路径

修正的回溯搜索算法**1**

BACKTRACK1**** (**DATALIST**)

DATALIST: 从初始到当前的状态表（逆向）

返回值：从当前状态到目标状态的路径
（以规则表的形式表示）
或**FAIL**。

修正的回溯搜索算法1

```
1  DATA:=FIRST(DATALIST)
2  IF MEMBER(DATA, TAIL(DATALIST))
   RETURN FAIL;
3  IF Term(DATA) RETURN NIL;
4  IF Deadend(DATA) RETURN FAIL;
5  IF Length(DATALIST)>BOUND
   RETURN FAIL;
6  RULES:=Apprules(DATA);
7  LOOP: IF NULL(RULES) RETURN FAIL;
8  R:=FIRST(RULES);
```

```
9    RULES:=Tail(RULES);
10   RDATA:=Gen(R, DATA);
11   RDATAList:=Cons(RDATA, DATAList);
12   PATH:=BACKTRCK1(RDATAList)
13   IF PATH=FAIL GO LOOP;
14   RETURN Cons(R, PATH);
```

一些问题

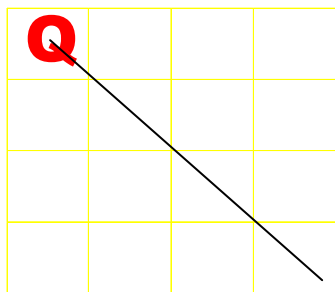
- 失败原因分析、多步回溯

Q			
		Q	

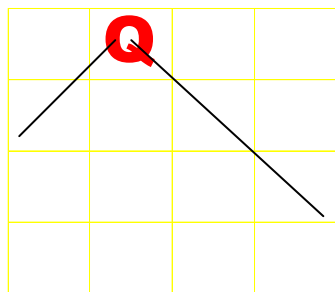
■ 回溯搜索中知识的利用

基本思想：

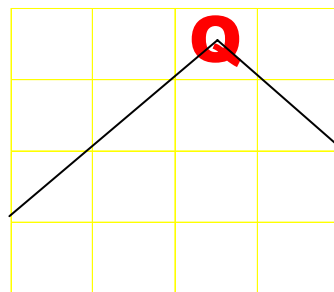
尽可能选取划去对角线上位置数最少的。



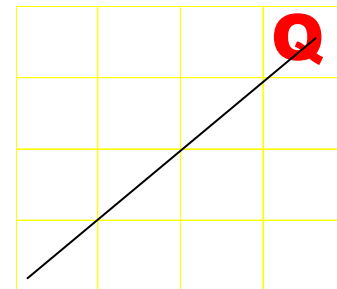
4



3



3



4

- 回溯策略
- 图搜索
- 无信息搜索
- 启发式搜索
- A*算法的可采纳性

图搜索策略

■ 问题的引出

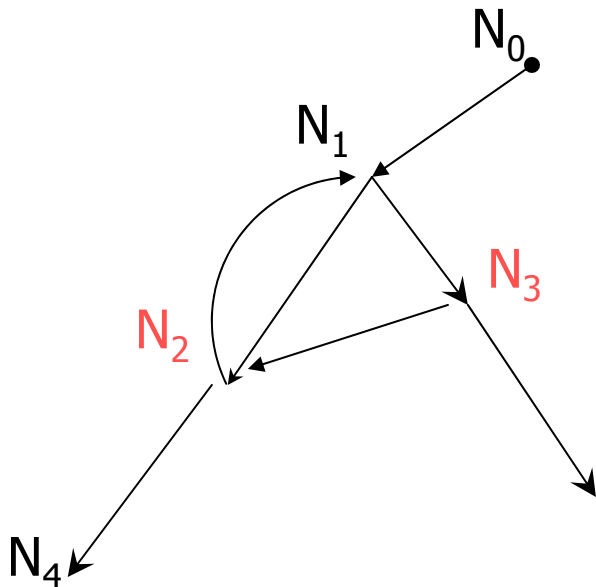
- 八皇后问题 / 找最短路径

■ 回溯与图搜索的区别

- 回溯：放弃的状态永远放弃；
- 图搜索：放弃的状态以后还可能再用；

■ 算法：

- 回溯搜索：只保留从初始状态到当前状态的一条路径。
- 图搜索：保留所有已经搜索过的路径。

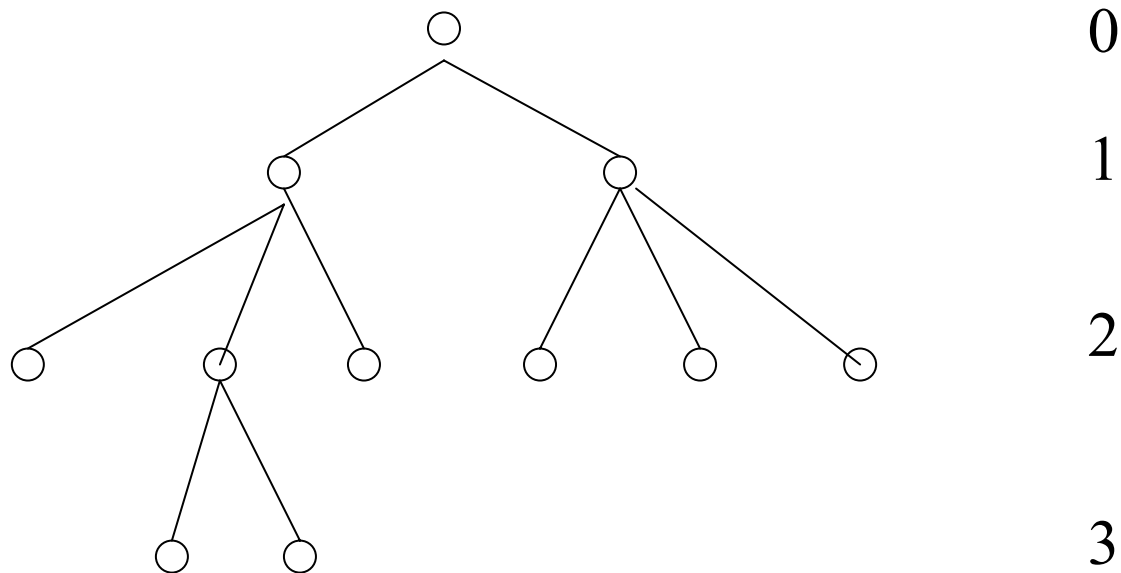


一些基本概念

- 图：一个节点的集合，节点由弧连接起来。
- 有向图：弧是一个节点指向另一个节点的图，称为有向图。
-
- 后继/父亲：如果有一条弧从 n_i 指向 m_j ，则 m_j 称为 n_i 的后继， n_i 称为 m_j 的父亲。

- 路径：如果存在一个节点序列（ $n_{i0}, n_{i1}, \dots, n_{ik}$ ）， n_{ij} 是 n_{ij-1} 的后继， $j=1, \dots, k$ ，则称这个序列是从节点 n_{i0} 到节点 n_{ik} 的一条路径，长度为 k 。
- 祖先/后裔：如果存在一条从 n_i 到 m_j 的路径，则称 m_j 是 n_i 的后裔， n_i 称为 m_j 的祖先。
- 树：每个节点最多只有一个父辈。没有父辈的节点称为根节点，没有后继的节点称为叶节点。

- 节点深度：
根节点深度=**0**
其它节点深度=父节点深度+**1**



- 扩展一个节点

- 生成出该节点的所有后继节点。

- 弧的费用

- 有一条弧连接 n_i 和 n_j 两个节点, 用 $C(n_i, n_j)$ 表示使用规则从 n_i 到 n_j 的费用(或耗散值)。
 - 玉泉路 \rightarrow 天安门

- 路径的耗散值

- 一条路径的耗散值等于连接这条路径各节点间所有耗散值的总和。用 $C(n_i, n_j)$ 表示从 n_i 到 n_j 的路径的耗散值。

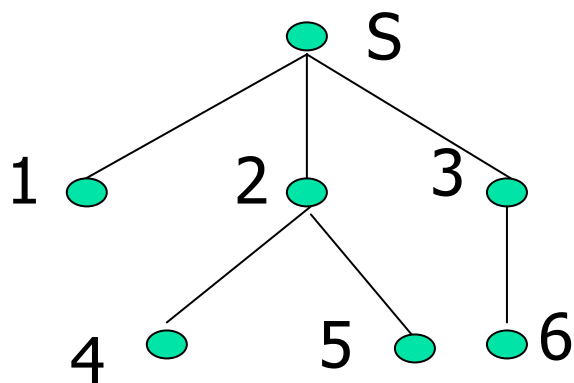
GRAPHSEARCH的思路

- **OPEN**表
 - 已经生成但未扩展节点
- **CLOSED**表
 - 已扩展节点
- 扩展节点*i*生成节点*j*
- 指针
- 调整指针

GRAPHSEARCH(simple version)

1. 建立一个只有起始节点**S**组成的图**G**, 把**S**放到**OPEN**表中;
2. 建立一个**CLOSED**表, 置为空;
3. **While(!NULL (OPEN))**
 - a) 从**OPEN**表中取出 (并删除) 第一个节点**n**放入**CLOSED**表。
 - b) 如果**n**是目标节点, 成功结束;
 - c) 扩展节点**n**, 把**n**的后继加入**G**中;
 - d) 把**n**的后继加入**OPEN**表中, 并建立它们到**n**的指针;
 - e) 对**OPEN**表中的节点排序;
4. 返回**FAIL**;

例子



OPEN

{S}

{ }

{1,2,3}

{**2**,1,3}

{1,3}

{1,3,**4**,**5**}

{**3**,1,4,5}

{1,4,5}

{1,4,5,**6**}

CLOSE

{ }

{S}

{S}

{S}

{S,**2**}

{S,2}

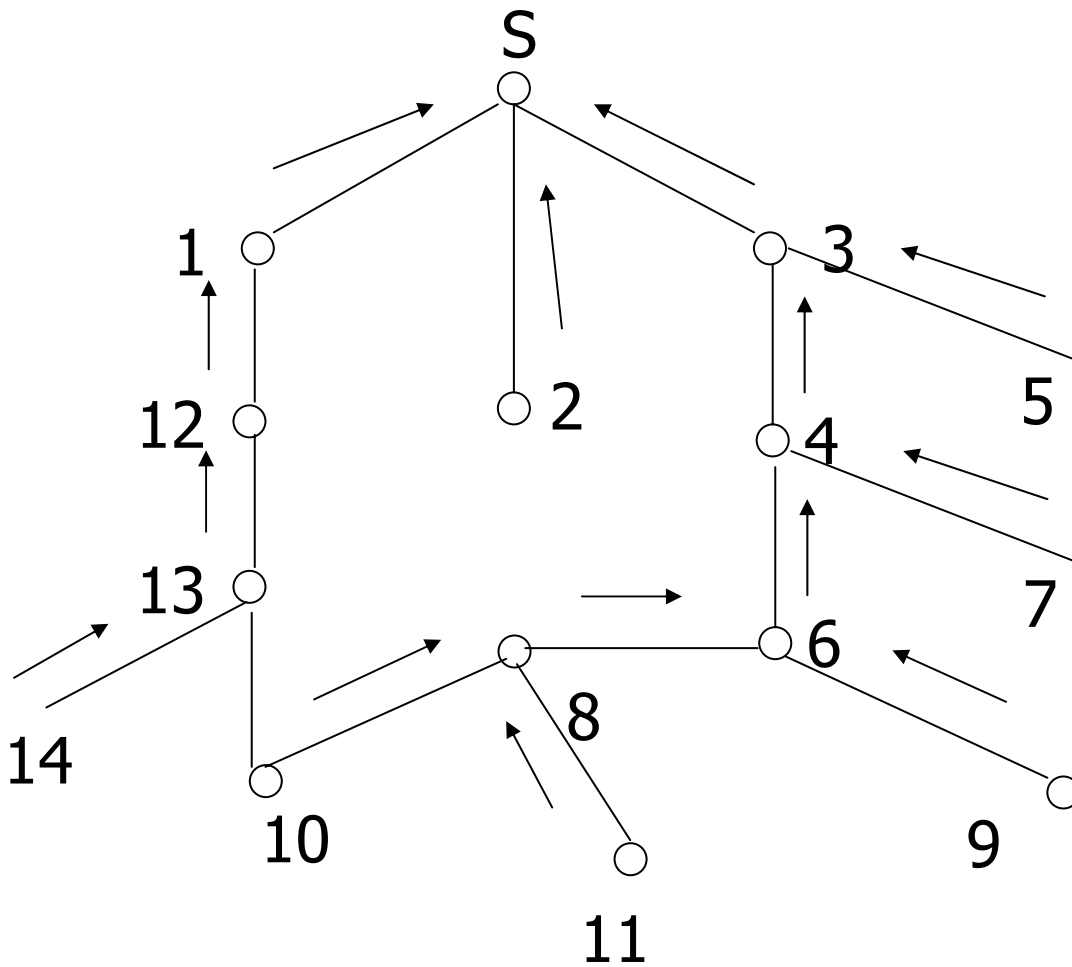
{S,2}

{S,2,**3**}

{S,2,3}

GRAPHSEARCH

1. 建立一个只有起始节点**S**组成的图**G**, 把**S**放到**OPEN**表中;
2. 建立一个**CLOSED**表, 置为空;
3. **While(!NULL (OPEN))**
 - a) 从**OPEN**表中取出（并删除）第一个节点**n**放入**CLOSED**表;
 - b) 如果**n**是目标节点, 成功结束;
 - c) 扩展节点**n**, 产生节点**n**的不是**n**的祖先的后继节点集合 **M**, 把**M**中的这些成员作为**n**的后继加入**G**中;
 - d) 对**M**的那些既不在**OPEN**表中又不在**CLOSED**表中的成员, 加入到**OPEN**表中, 并建立它们到**n**的指针; 对那些在**OPEN**表中的成员, 决定是否调整它们的指针; 对那些在**CLOSED**表中的成员, 决定是否调整它们的指针, 并决定是否调整它们在**G**中的后裔的指针;
 - e) 对**OPEN**表中的节点排序;
4. 返回**FAIL**;



OPEN

{S}

{1,2,3}

{**3**,1,2}

{**4**,5,1,2}

{**6**,7,5,1,2}

{**8**,9,7,5,1,2}

{10,11,9,7,5,1,2}

CLOSE

{}

{S}

{S}

{S,3}

{S,3,4}

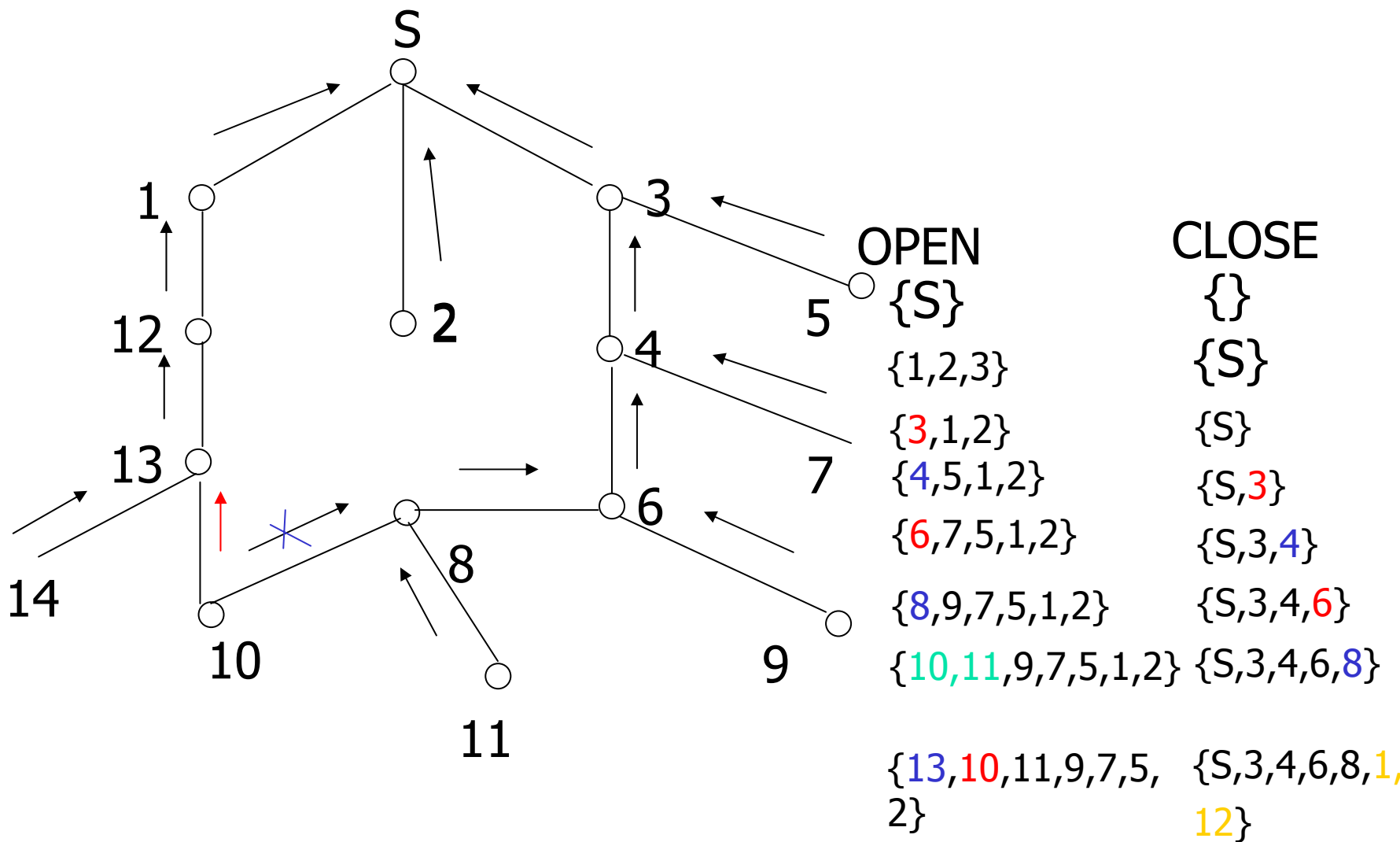
{S,3,4,6}

{S,3,4,6,8}

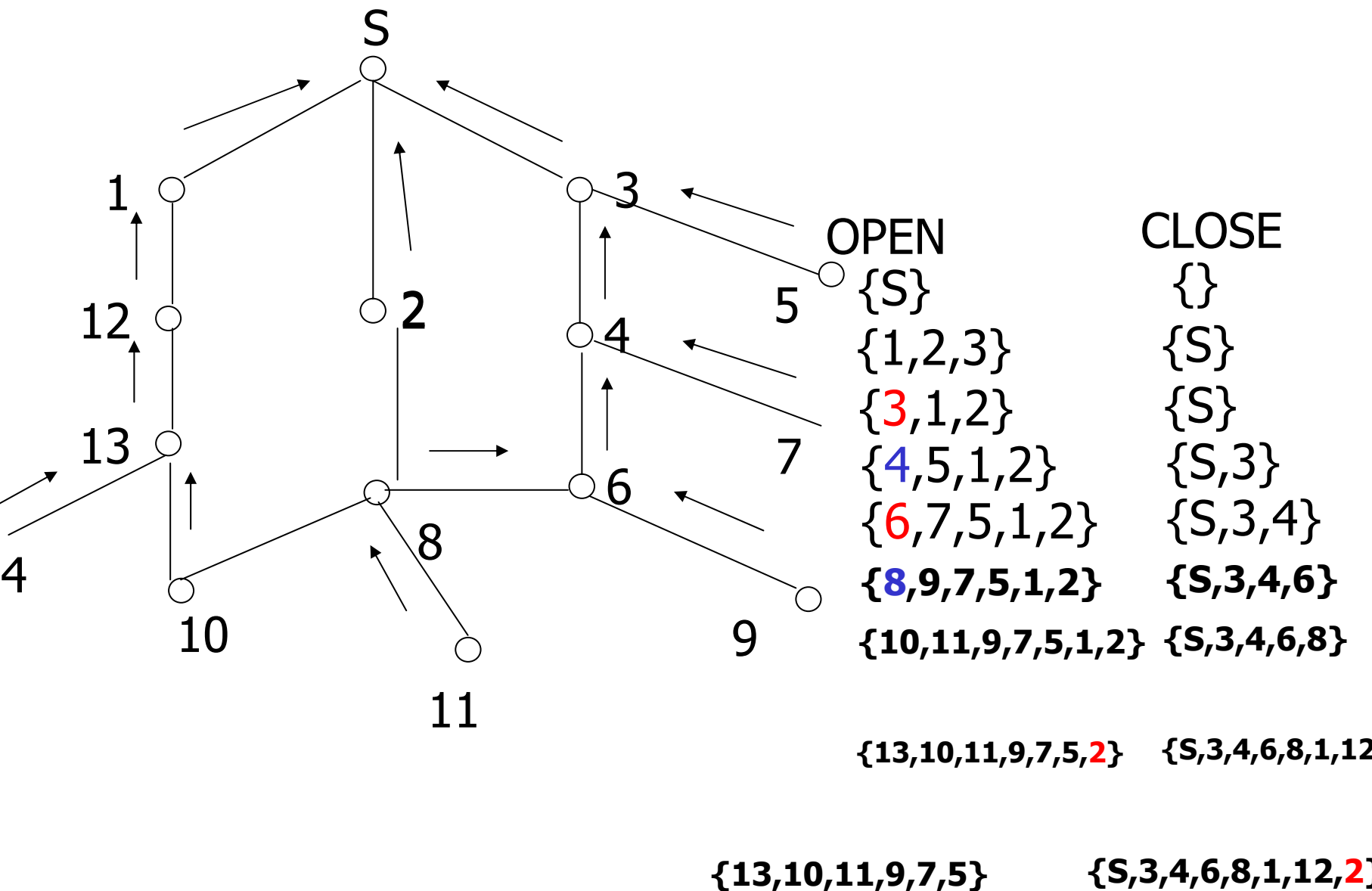
{**13**,10,11,9,7,5,2} {S,3,4,6,8,**1**,12}

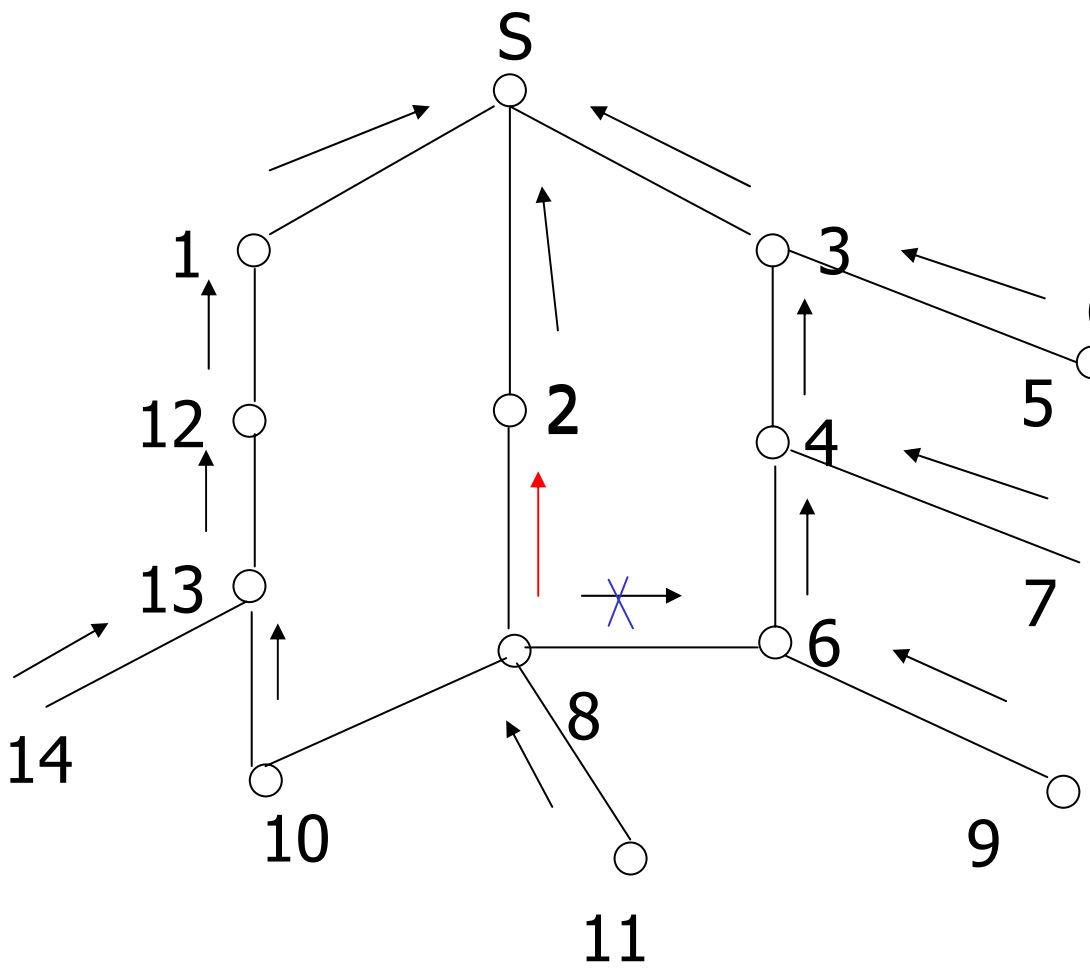
{**14**,10,11,9,7,5,2}

{S,3,4,6,8,1,12,**13**}



OPEN表中的节点修改指针

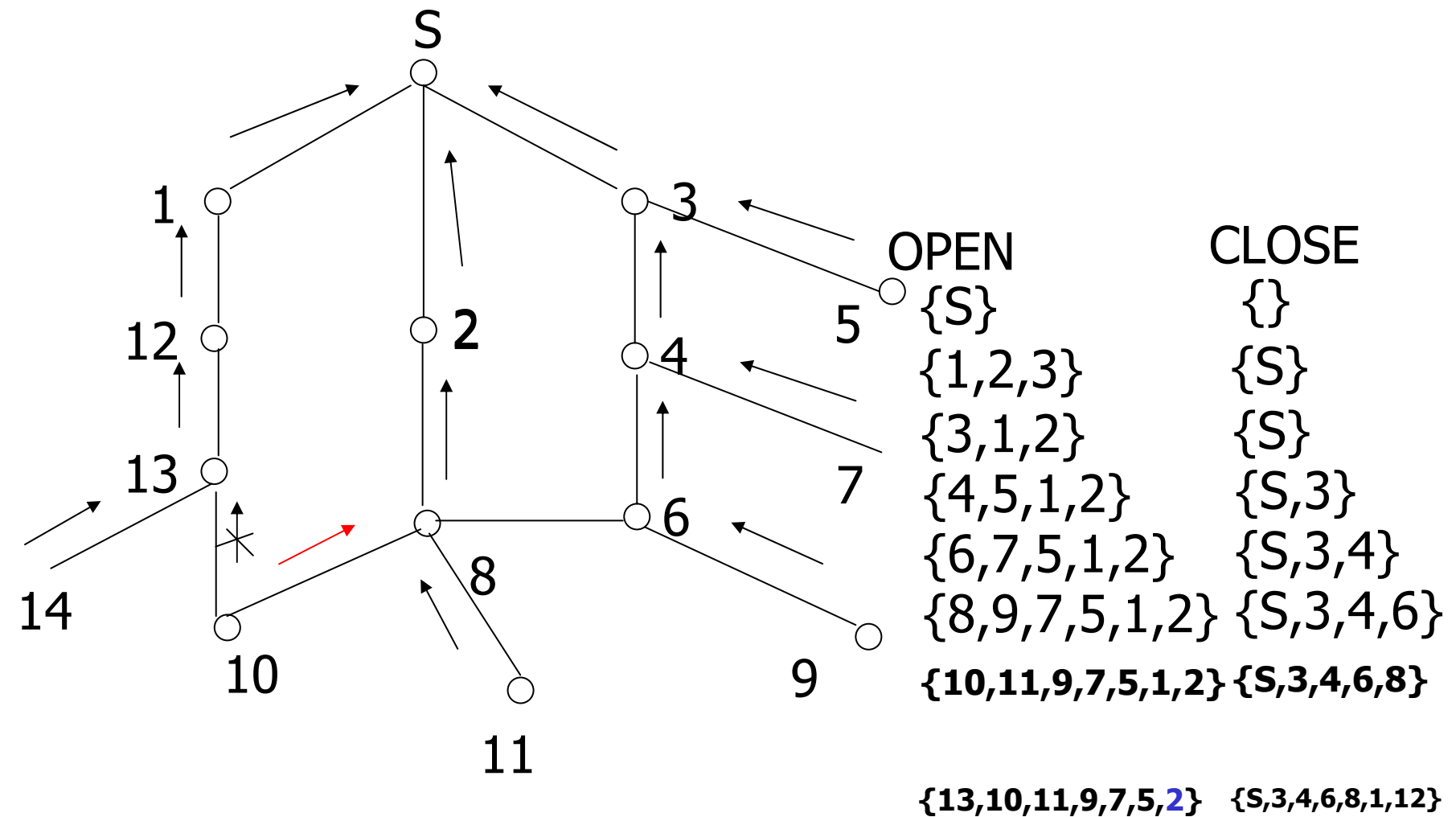




OPEN	CLOSE
{S}	{}
{1,2,3}	{S}
{3,1,2}	{S}
{4,5,1,2}	{S,3}
{6,7,5,1,2}	{S,3,4}
{8,9,7,5,1,2}	{S,3,4,6}
{10,11,9,7,5,1,2}	{S,3,4,6,8,1,2}
{13,10,11,9,7,5,2}	{S,3,4,6,8,1,12,2}

CLOSE表中的节点修改指针

{13,10,11,9,7,5} {S,3,4,6,**8**,1,12,2}



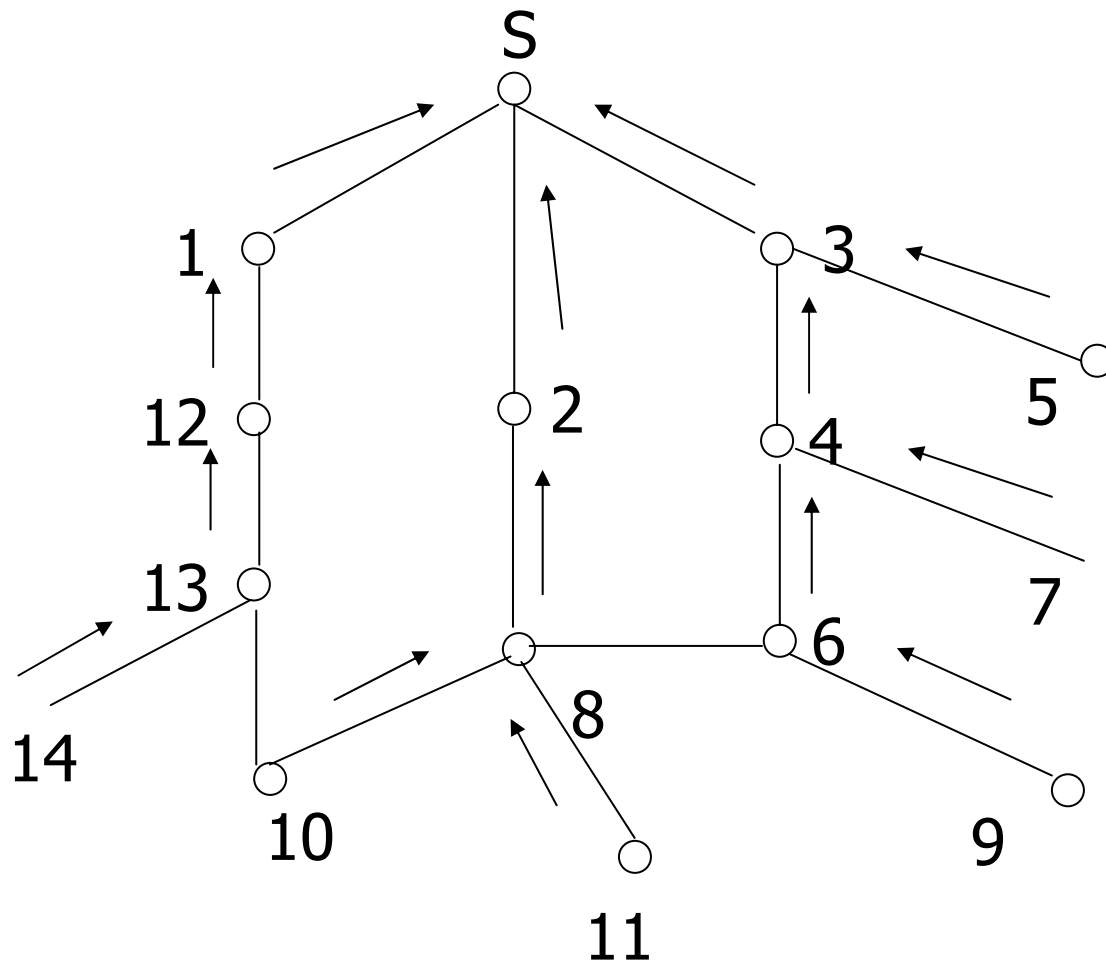
CLOSE表中的节点(8)的后裔(10)修改指针

{13,10,11,9,7,5} {S,3,4,6,8,1,12,2}

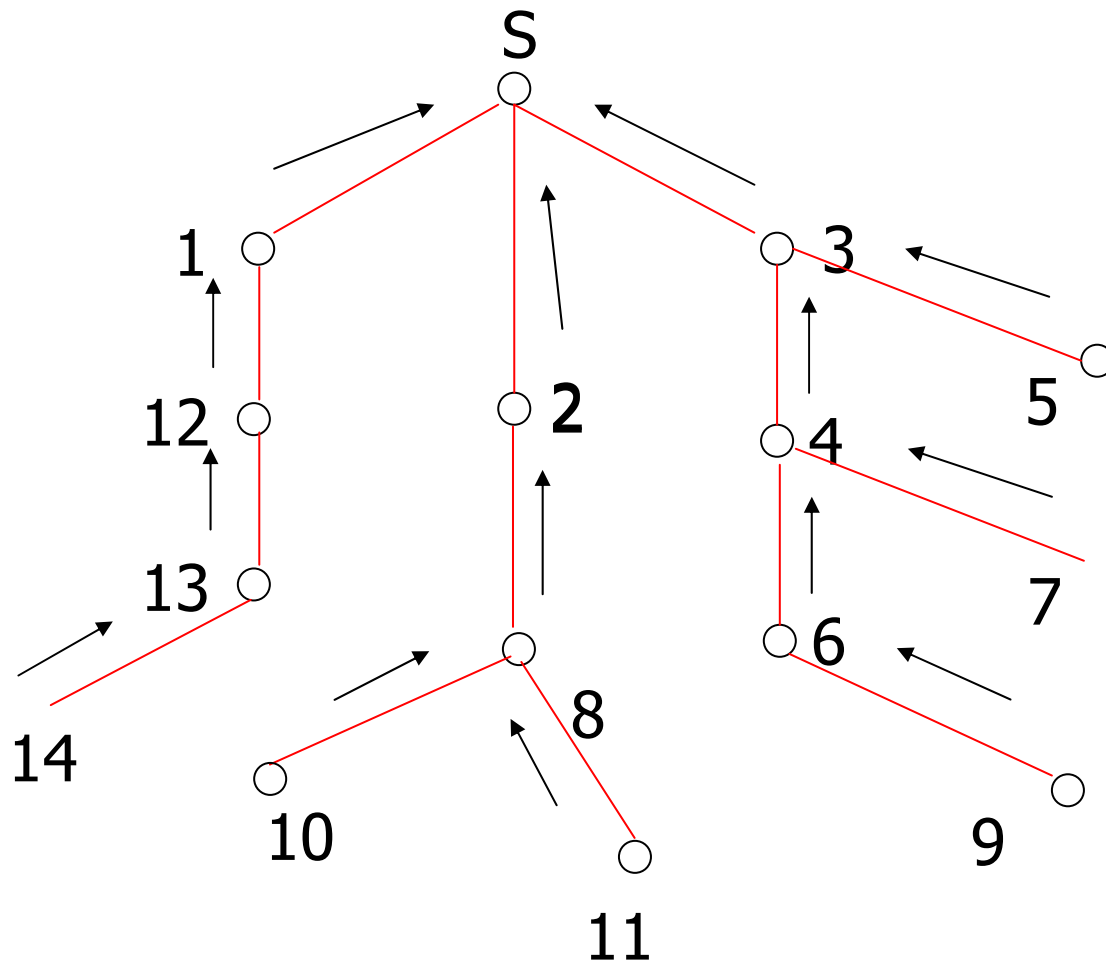
GRAPHSEARCH说明

- **OPEN**表 —— **CLOSED**表;
- 目标节点即将被扩展时, 算法成功结束;
- 检查新产生的节点以前是否产生过, 计算量较大。
- 算法结束后, 将生成一个图**G**, 称为**搜索图**。同时由于每个节点都有一个指针指向父节点, 这些指针指向的节点构成**G**的一个**支撑树**, 称为**搜索树**。
- 从目标节点开始, 将指针指向的状态回串起来, 即找到一条**解路径**。
- 树/不修改指针, 图/修改指针。
- 修改指针: 找最优解

搜索图



搜索树



图搜索与回溯算法的区别

- 扩展节点:

- 回溯算法: 生成一个儿子节点.
- 图搜索: 扩展节点, 生成所有儿子节点.

- 候选节点:

- 回溯算法: 一个.
- 图搜索: 多个.

- 回溯:

- 回溯算法: 返回父亲节点.
- 图搜索: 不一定返回父亲节点.

- 回溯策略
- 图搜索
- 无信息搜索
- 启发式搜索
- **A***算法的可采纳性

无信息图搜索过程

- 无信息图搜索

- 如果在**GRAPHSEARCH**中，对节点的排序不使用与问题相关的信息，则称为无信息图搜索。

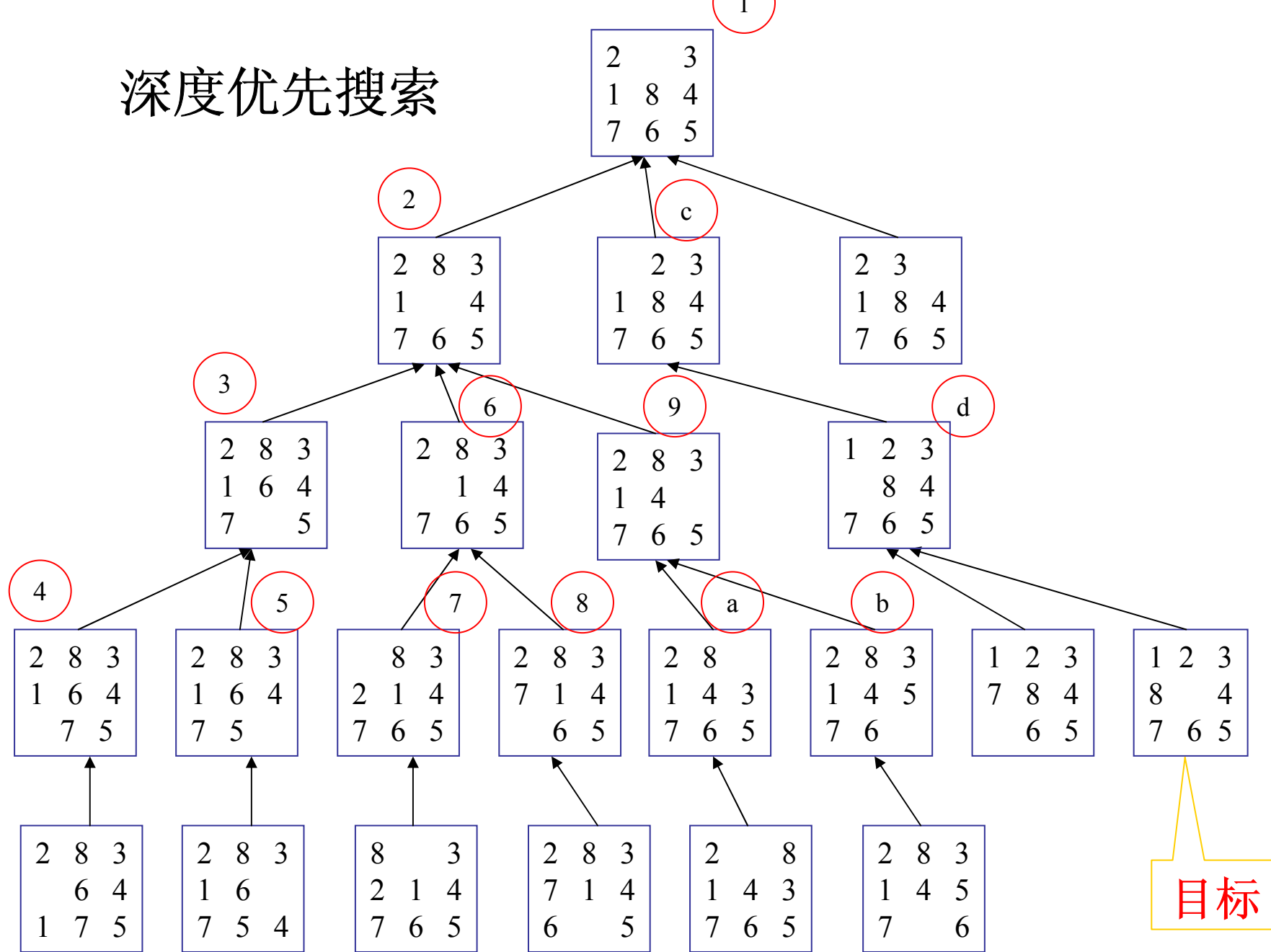
- 深度优先搜索

- 不同的节点, 深度越大, 在**OPEN**表中越靠前;
- 深度相同, 任意排序;

- 宽度优先搜索

- 不同的节点, 深度越小, 在**OPEN**表中越靠前;

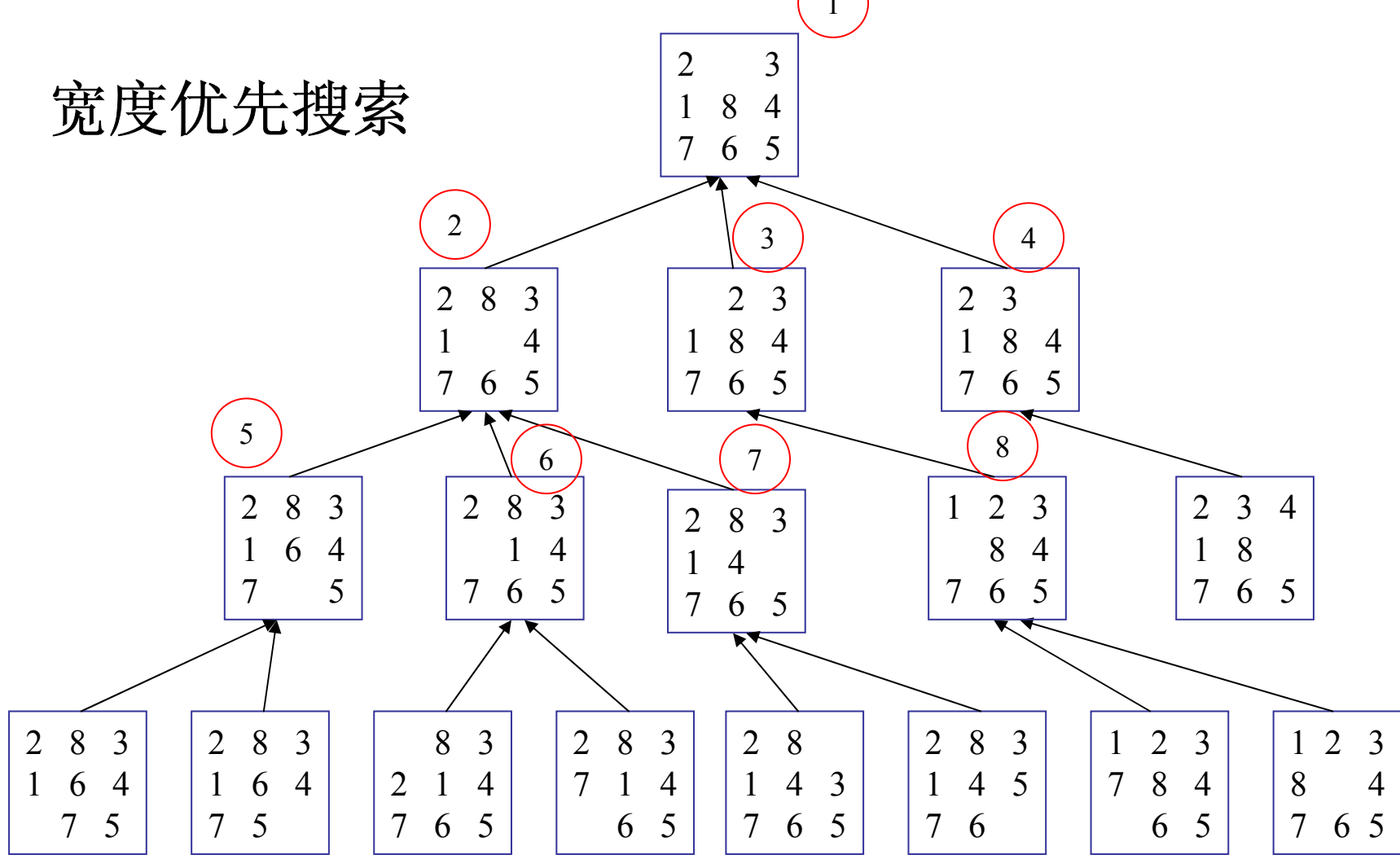
深度优先搜索



深度优先搜索的性质

- 一般设置深度限制；当深度限制不合理时，可能找不到解；可以将算法改为可变深度限制。
- 最坏情况时，搜索空间趋于穷举
- 与回溯法的差别：图搜索
- 是一个通用的与问题无关的方法

宽度优先搜索



目标

宽度优先搜索的性质

- 当问题有解时，一定能找到解。
- 当问题为单位耗散值，且问题有解时，一定能找到最优解。

- 回溯策略
- 图搜索
- 无信息搜索
- 启发式搜索
- **A***算法的可采纳性

启发式搜索(heuristic search)

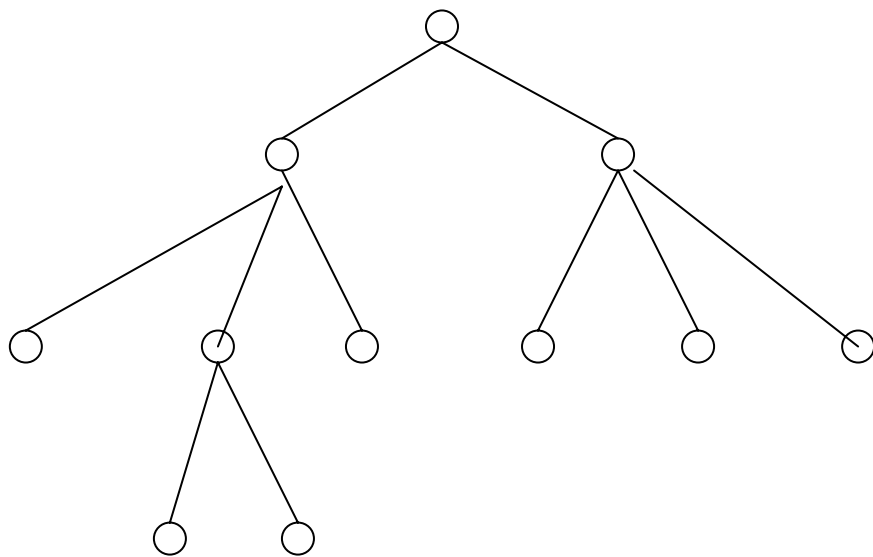
- 无信息搜索一般需要产生大量的节点，因而效率较低。
- 启发式信息
 - 为提高效率，可以使用一些问题相关的信息，以减小搜索量，这些信息就称为启发式信息。
- 启发式搜索
 - 使用启发式信息指导的搜索过程称为启发式搜索。
 - 启发式图搜索：对节点排序。
- 例子：
 - 王皇路 → 天安门

■ 启发信息的强度

- 强：降低搜索工作量，但可能导致找不到最优解。
- 弱：一般导致工作量加大，极限情况下变为盲目搜索，
但可能可以找到最优解。

基本思想

- 定义一个评价函数 $f(n)$ ，对当前的搜索状态进行评估，找出一个最有希望的节点来扩展。



例子: eight-puzzle

- 评价函数

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

- $d(n)$: 节点 n 的深度;

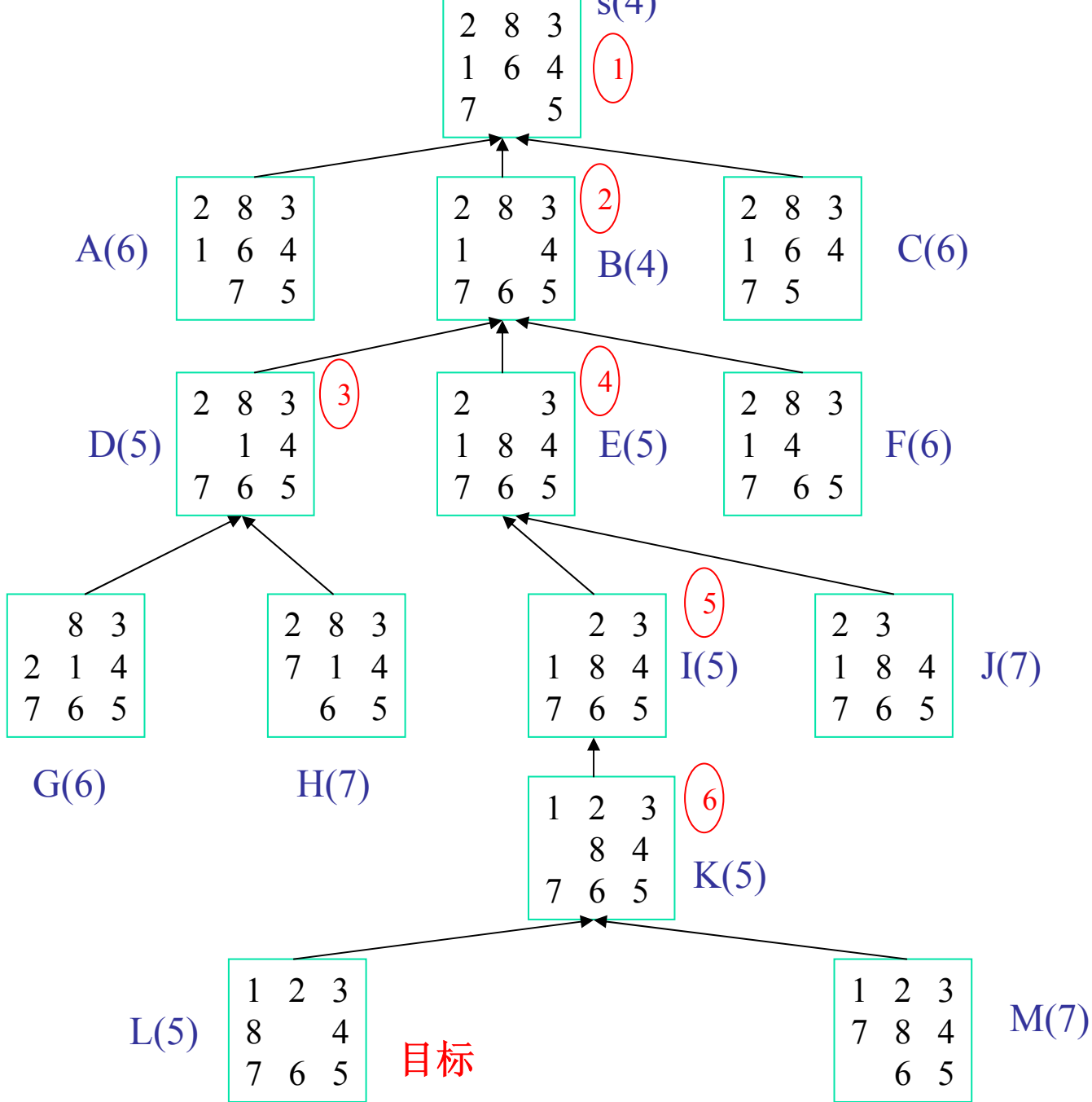
- $W(n)$: 与目标相比, 错位的数字数目;

2	8	3
1	6	4
7		5

初始状态

1	2	3
8		4
7	6	5

目标状态

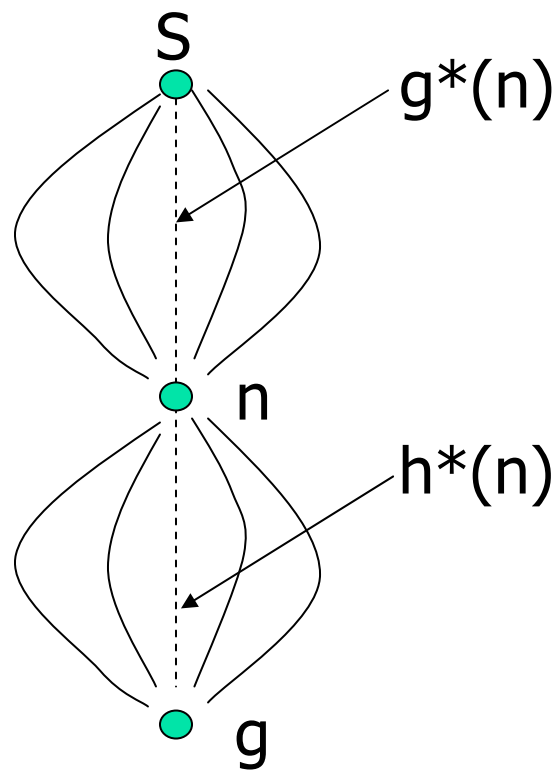


- 启发式就是要猜测：

- 从节点**n**开始，找到最优解的可能性有多大？
- 从起始节点开始，经过节点**n**，到达目标节点的最佳路径的费用是多少？

- **g**

- **h**



符号的意义

- $g^*(n)$: 从 s 到 n 的最短路径的耗散值
- $h^*(n)$: 从 n 到 g 的最短路径的耗散值
- $f^*(n)=g^*(n)+h^*(n)$: 从 s 经过 n 到 g 的最短路径的耗散值
- $g(n)$ 、 $h(n)$ 、 $f(n)$ 分别是 $g^*(n)$ 、 $h^*(n)$ 、 $f^*(n)$ 的估计值
- $h(n)$: 启发函数

几个性质

- **$f^*(S)$**

- **$f^*(S) = g^*(S) + h^*(S) = h^*(S)$**
- 从 **S** 无约束地到达目标的最佳路经上的耗散值

- **$g(n)$**

- 一般取实际走过的路径的费用和
- **$g(n) \geq g^*(n)$**
- 最佳路经上的节点 **n** ， 满足 **$g(n) = g^*(n)$** 。
- 随着算法的执行， 由于指针的变动， **$g(n)$** 会下降。

- **$h \equiv 0$**

- 没有启发式信息

A算法

- $f(n) = g(n) + h(n)$
- $g(n)$ 取实际走过的路径的费用和
- 每一条弧上的费用大于一个小正数 ε
- 节点排序是按照 **$f(n)$** 从小到大排

算法**A***

- 在**A**算法中，如果满足条件：

$$0 \leq h(n) \leq h^*(n)$$

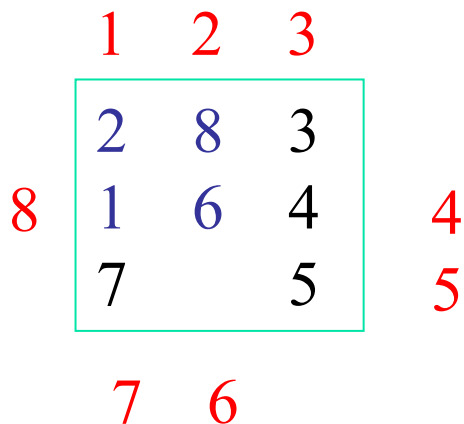
则**A**算法称为**A***算法。

A*算法的说明

■ 8数码问题

■ $h(n)$ = “不在位” 的将牌个数

■ $h(n)$ = 将牌 “不在位” 的距离和



将牌1: 1
将牌2: 1
将牌6: 1
将牌8: 2

- 对于宽度优先算法，当问题为单位耗散值，且问题有解时，一定能找到最优解。

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

- $g(n)$

- $h(n) = 0 \leq h^*(n)$

- 回溯策略
- 图搜索
- 无信息搜索
- 启发式搜索
- **A***算法的可采纳性

可采纳性

- 可采纳性:

- 对任一个图，存在从**S**到目标的路径，如果一个搜索算法总是结束在一条从**S**到目标的最佳路径上，则称此算法是可采纳的。

- 一条: 多条

A*算法的性质

- 性质一：

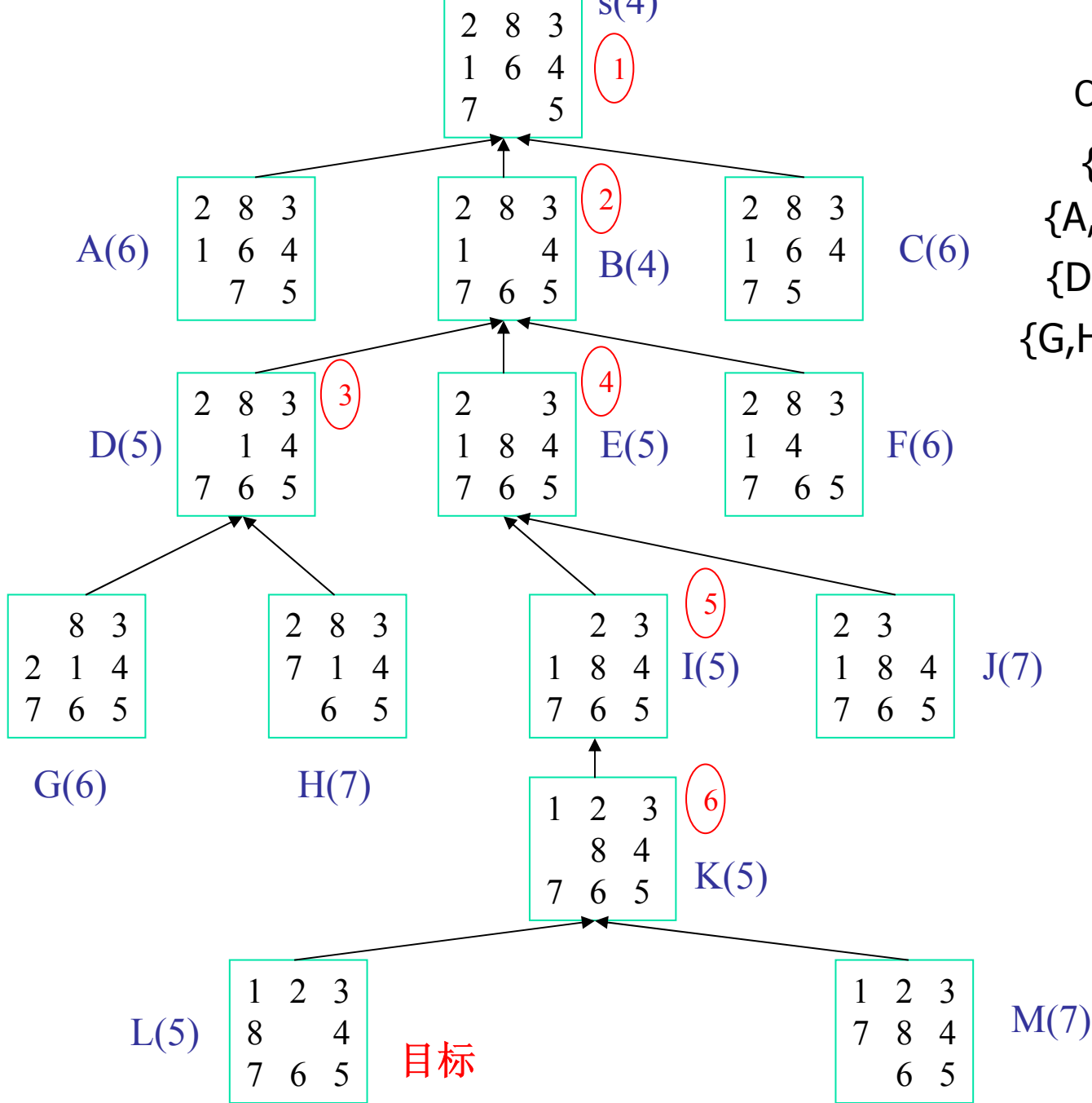
- **GRAPHSEARCH**算法对有限图终止。

- 算法当**OPEN**表为空时结束。

- 图中的任何一个节点都只进入**OPEN**表一次，并且图中节点是有限的，所以**OPEN**表一定会被取空，导致算法结束。

■ 性质二:

- 在**A***算法结束前的任意时刻, **OPEN**表中都至少有一个节点**n**, 节点**n**在从初始节点**S**到目标的最佳路径上, 并且**A***算法已经发现了这条路径, 并且节点**n**满足 $f(n) \leq f^*(S)$ 。



OPEN	CLOSED
{S}	{}
{A,B,C}	{S}
{D,E,F,A,C}	{S,B}
{G,H,E,F,A,C}	{S,B,D}

- $$\begin{aligned}
 \mathbf{f}(\mathbf{n}) &= \mathbf{g}(\mathbf{n}) + \mathbf{h}(\mathbf{n}) \\
 &= \mathbf{g}^*(\mathbf{n}) + \mathbf{h}(\mathbf{n}) \\
 &\leq \mathbf{g}^*(\mathbf{n}) + \mathbf{h}^*(\mathbf{n}) = \mathbf{f}^*(\mathbf{S})
 \end{aligned}$$

■ 性质三:

- 如果存在从**S**到目标节点的路径，则**A***扩展的节点**n**，一定满足:

$$f(n) \leq f^*(S)$$

- 根据性质二，在**A***算法结束前的任意时刻，**OPEN**表中都至少有一个最佳路径上的节点**n**，且满足**f(n) ≤ f*(S)**，
- 如果扩展的节点是节点**n**，则性质成立;
- 如果扩展的节点不是节点**n**，则因为**A***排序是由小到大排，所以选择扩展的节点**n₁**一定满足

$$f(n_1) \leq f(n) \leq f^*(S)$$

性质成立。

■ 性质四：

- 如果存在从**S**到目标节点的路径，则**A***对无限图结束。

- 如果**A***不结束，设**n**为当前扩展节点，则

$$\mathbf{f}(\mathbf{n}) = \mathbf{g}(\mathbf{n}) + \mathbf{h}(\mathbf{n}) \geq \mathbf{g}^*(\mathbf{n}) + \mathbf{h}(\mathbf{n}) \geq \mathbf{g}^*(\mathbf{n})$$

$\mathbf{g}^*(\mathbf{n})$ 为从起始节点**S**到当前节点**n**的最佳路径的费用。

- 因为每一条弧上的费用大于一个小正数 ε ；所以

$$\mathbf{g}^*(\mathbf{n}) \geq \mathbf{d}^*(\mathbf{n}) * \varepsilon ;$$

其中， $\mathbf{d}^*(\mathbf{n})$ 为**S**到**n**的最佳路径的弧的数目。

- \mathbf{A}^* 不结束 $\Rightarrow \mathbf{d}^*(\mathbf{n}) \rightarrow \infty \Rightarrow \mathbf{g}^*(\mathbf{n}) \rightarrow \infty \Rightarrow \mathbf{f}(\mathbf{n}) \rightarrow \infty$
- 根据性质三，**A***扩展的节点**n**，一定满足 $\mathbf{f}(\mathbf{n}) \leq \mathbf{f}^*(\mathbf{S})$ ，所以矛盾。
- 所以**A***对无限图结束。

■ 性质五:

- **A***是可采纳的。

- 如果一个图，存在从**S**到目标的路径，**A***一定终止在解路径上。
- **A***一定终止在最佳路径上。

- 如果一个图，存在从**S**到目标的路径，**A***一定终止在解路径上。
 - 不明显.
 - **A***对有限图或无限图都终止
 - 但并不说明一定终止在解路径上。即算法可能因为**OPEN**表为空而终止，而并不是因为找到解路径而终止。
 - **OPEN**表为空而终止？

- 设算法因为**OPEN**表为空而终止，在**A***算法结束前的任意时刻，**OPEN**表中都至少有一个最佳路径上的节点（性质二）
- 则这个节点一定被扩展过，而它的在最佳路径上的后继会被放到**OPEN**表中，并且同样被扩展过，.....，
- 由此可以推出最佳路径上的最后一个节点一定被扩展过，
- 因为它是目标节点，一定在算法循环内结束，与**OPEN**表为空而结束矛盾，所以一定终止在解路径上。

■ **A***一定终止在最佳路径上。

- 不明显.
- 次优路径?
- 设没有终止在最佳路径上，而最后扩展的目标节点为 G_1 ,

$$f(G_1) = g(G_1) + h(G_1) \geq g(G_1)$$

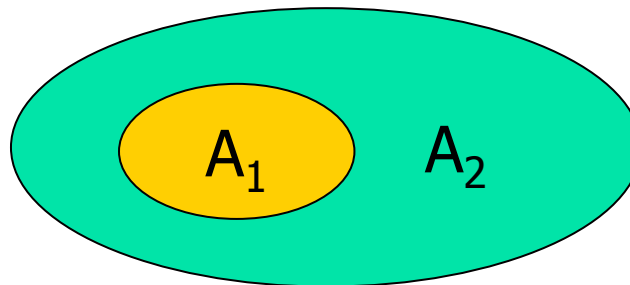
因为并非最佳路径，所以有

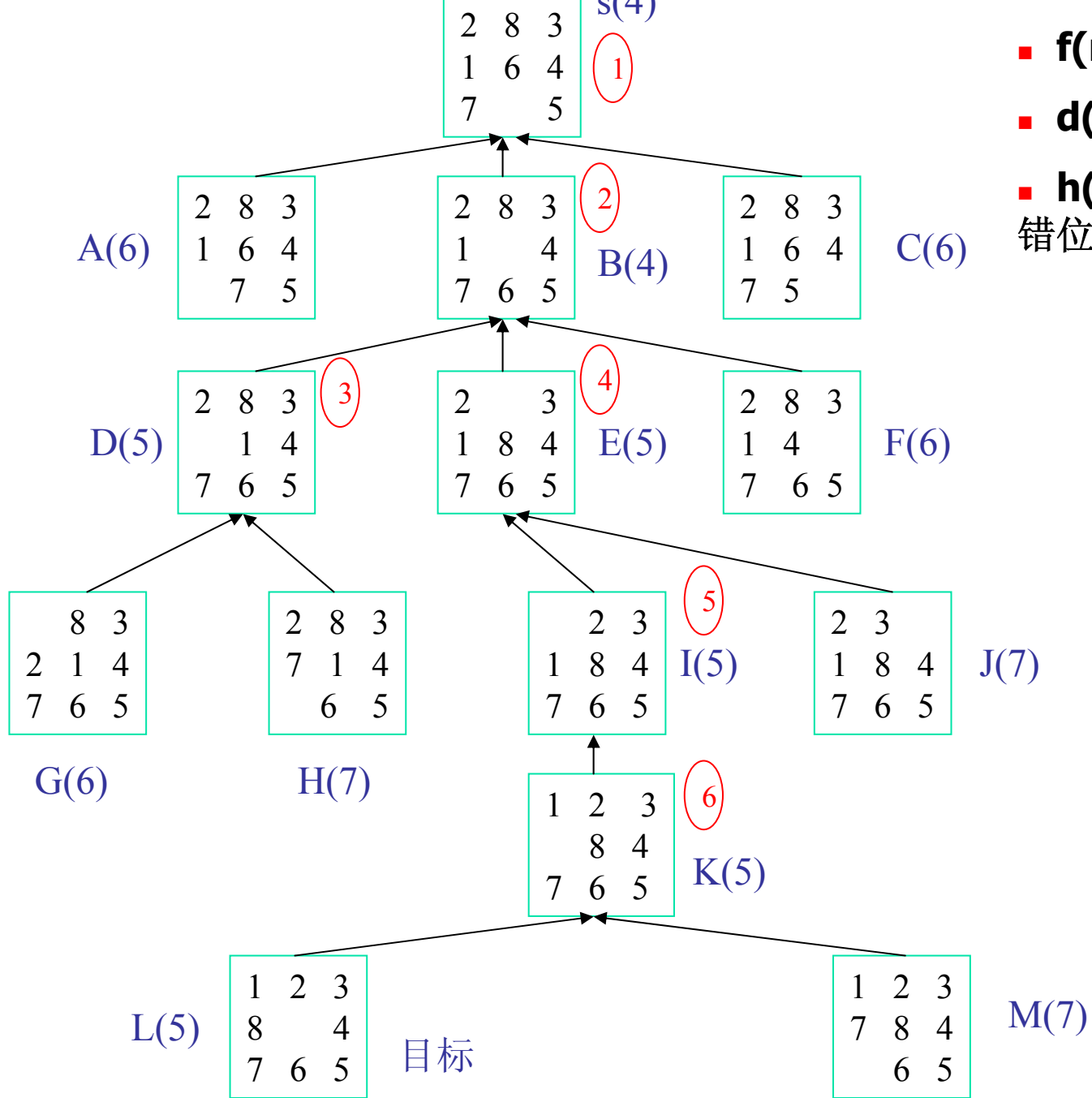
$$f(G_1) \geq g(G_1) > f^*(S)$$

- 而根据性质四， $f(G_1) \leq f^*(S)$ ，矛盾。

■ 性质六:

- A_1 与 A_2 是具有不同启发函数的 A^* 算法, 假设 $h_1 > h_2$, 则 A_1 具有更多的启发式信息, 由 A_1 扩展的节点必然被 A_2 扩展。





■ $f(n) = d(n) + h(n)$

■ $d(n)$: 节点 n 的深度;

■ $h(n)$: 与目标相比, 错位的数字数目;

- 对**A₁**终止时搜索树的深度用归纳法证明。
 - 对深度为**0**的节点，即起始节点，**A₁**和**A₂**都扩展，所以成立；
 - 设对深度小于等于**k**的所有节点，**A₁**扩展的都被**A₂**扩展。
 - 对深度为**k+1**的节点**n**，根据归纳法，**n**的祖先如果被**A₁**扩展，则一定被**A₂**扩展。
 - 因此对**A₂**来说，从**n**开始用指针回溯的路径的费用一定小于或等于**A₁**中从**n**开始用指针回溯的路径的费用。即，

$$g_2(n) \leq g_1(n)$$
 - 又, $h_1(n) > h_2(n)$
 - 所以, $f_1(n) = g_1(n) + h_1(n)$

$$> g_2(n) + h_2(n) = f_2(n)$$

- (反证法) n 在 A_1 和 A_2 的**OPEN**表中, 设 A_1 扩展 n , 而 A_2 没有扩展 n ,
- 只要推出 $f_1(n) \leq f_2(n)$ 即矛盾;
- $f_1(n) \leq f^*(s) \leq f_2(n)$
- A_1 扩展 n , $f_1(n) \leq f^*(s)$ (性质三)
- A_2 是A*算法, 所以 A_2 一定终止在目标节点 G 上并找到最优解, 即, 最后一个扩展的节点是目标节点 G 。
- $f_2(G) = g_2(G) + h_2(G) = g_2^*(G) + h_2(G)$
- 因为 $0 \leq h \leq h^*$, 而 $h_2^*(G) = 0$, 所以 $h_2(G) = 0$
- $f_2(G) = g_2^*(G) = f^*(s)$
- n 和 G 都在**OPEN**表中, 而 A_2 扩展了 G 没有扩展 n , 所以一定有: $f_2(G) \leq f_2(n)$
- $f^*(s) \leq f_2(n)$

关于**A***和启发式信息

- $h(n) \leq h^*(n)$ ，一定可以找到最优解。
- 满足以上条件， $h(n)$ 越大，搜索的节点数越少。当 $h(n) = h^*(n)$ 时，搜索的节点数最少。
 - 效率是另一个问题；

- 不满足 $h(n) \leq h^*(n)$, 则可能找不到最优解; 但搜索效率可能提高;

- 例子: **eight-puzzle**

- $h(n) = p(n) + 3S(n)$,

其中: $p(n)$ 是每一个数字离目标位置的距离和, $S(n)$ 是一个序列分, 如下得到: 对非中心的外圈上的数字按顺时针方向走一圈, 如果它的后继不是目标状态下的后继, 则记为**2**; 否则, 记为**0**, 对中心位置, 有数字, 记为**1**; 否则记为**0**。把这些分都加起来, 就得到序列分 $S(n)$ 。

2		6
7	1	8
5	4	3

$$p(n) = 2+1+2+2+2+3+1+2=15$$

$$S(n) = 2+2+2+2+0+0+2+1=11$$

$$h(n) = p(n) + 3*S(n) = 48$$

2	1	6
	4	8
7	5	3

$$p(n_1) = 1+1+2+1+1+3+0+2=11$$

$$S(n_1) = 2+2+2+2+2+2+2+1=15$$

$$h(n_1) = p(n_1) + 3*S(n_1) = 56$$

■ $g(n) \equiv 0$

- 如果目的仅仅是找到一个解，对解的好坏没有要求，则可设 $g(n) \equiv 0$.
- 这时从起始节点到当前节点的路径的费用对搜索没有影响，而只关心从当前节点到目标节点所需要的搜索量，即只与 h 有关.
- 对 h 要求很大.

■ $f = g + w * h$

- w 是一个正数，
- 利用 w 控制 g 和 h 在搜索中起的作用。
- 经验表明，让 w 与搜索深度成相反方向变化时，效果较好。
- 即，深度越小时， w 越大，强调启发式知识的作用；深度越大时， w 越小，越来越接近于宽度优先，以便保证找到解。