人工智能中的搜索

- 我们进入到具体的技术阶段
- 曾经很红的“专家系统”，后来发展为“知识工程”的核心是什么呢？

- 主要就是一些数据
- 加上一些规则（就是知识）[操作这些数据的]
- 加上通用的控制策略

- 这些方法现在依然在用
- 深度学习在没有大量数据可学的时候，无法发挥功效
- 还是要用规则的

- 跟图灵机的定义差不多
- 跟冯诺依曼体系结构的计算机也差不多

- 要了解几个典型的例子，体会出人工智能的味道

1.1 专家系统（产生式系统）的基本组成

- 组成三要素：
  - 一个综合数据库——存放信息
  - 一组产生式规则——知识
  - 一个控制系统——规则的解释或执行程序（控制策略）（推理引擎）

规则的一般形式

- IF <前提> THEN <结论>
- IF <条件> THEN <行动>
- 或者简写为：

  <前提> —> <结论>

  <条件> —> <行动>

1.2 产生式系统的基本过程

过程PRODUCTION

1，DATA←初始数据库

2，until DATA满足结束条件，do

3，{

4，　　在规则集中选择一条可应用于DATA　　的规则R

5，　　DATA ←R应用到DATA得到的结果

6，}

一个简单的例子

- 问题：设字符转换规则

  $A \wedge B \rightarrow C$

  $A \wedge C \rightarrow D$

  $B \wedge C \rightarrow G$

  $B \wedge E \rightarrow F$

  $D \rightarrow E$

  已知：A，B

  求：F

一个简单的例子（续1）

一、综合数据库

{x}，其中x为字符

二、规则集

1，IF A∧B THEN C
2，IF A∧C THEN D
3，IF B∧C THEN G
4，IF B∧E THEN F
5，IF D THEN E

一个简单的例子（续2）

三、控制策略

顺序排队

四、初始条件

{A，B}

五、结束条件

F∈{x}

求解过程

数据库	可触发规则	被触发规则
A，B	（1）	（1）
A，B，C	（2） （3）	（2）
A，B，C，D	（3） （5）	（3）
A，B，C，D，G	（5）	（5）
A，B，C，D，G，E	（4）	（4）
A，B，C，D，G，E，F		

1，IF A∧B THEN C　　2，IF A∧C THEN D

3，IF B∧C THEN G　　4，IF B∧E THEN F

5，IF　D　THEN E

1.3 问题表示举例

例1：传教士与野人问题（M-C问题）

问题：N个传教士，N个野人，一条船，可同时乘坐k个人，要求在任何时刻，在河的两岸，传教士人数不能少于野人的人数。

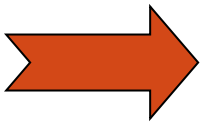
问：如何过河。

以N=3，k=2为例求解。

M-C问题（续1）

初始

	L	R
m	3	0
c	3	0
B	1	0



目标

	L	R
m	0	3
c	0	3
B	0	1

M-C问题（续2）

1，综合数据库

（m, c, b)，

其中：$0 \le m, c \le 3, b \in \{0, 1\}$

2，初始状态

（3，3，1）

3，目标状态（结束状态）

（0，0，0）

M-C问题（续3）

4，规则集

IF (m, c, 1) THEN (m-1, c, 0)

IF (m, c, 1) THEN (m, c-1, 0)

IF (m, c, 1) THEN (m-1, c-1, 0)

IF (m, c, 1) THEN (m-2, c, 0)

IF (m, c, 1) THEN (m, c-2, 0)

M-C问题（续4）

IF (m, c, 0) THEN (m+1, c, 1)

IF (m, c, 0) THEN (m, c+1, 1)

IF (m, c, 0) THEN (m+1, c+1, 1)

IF (m, c, 0) THEN (m+2, c, 1)

IF (m, c, 0) THEN (m, c+2, 1)

5，控制策略：（略）

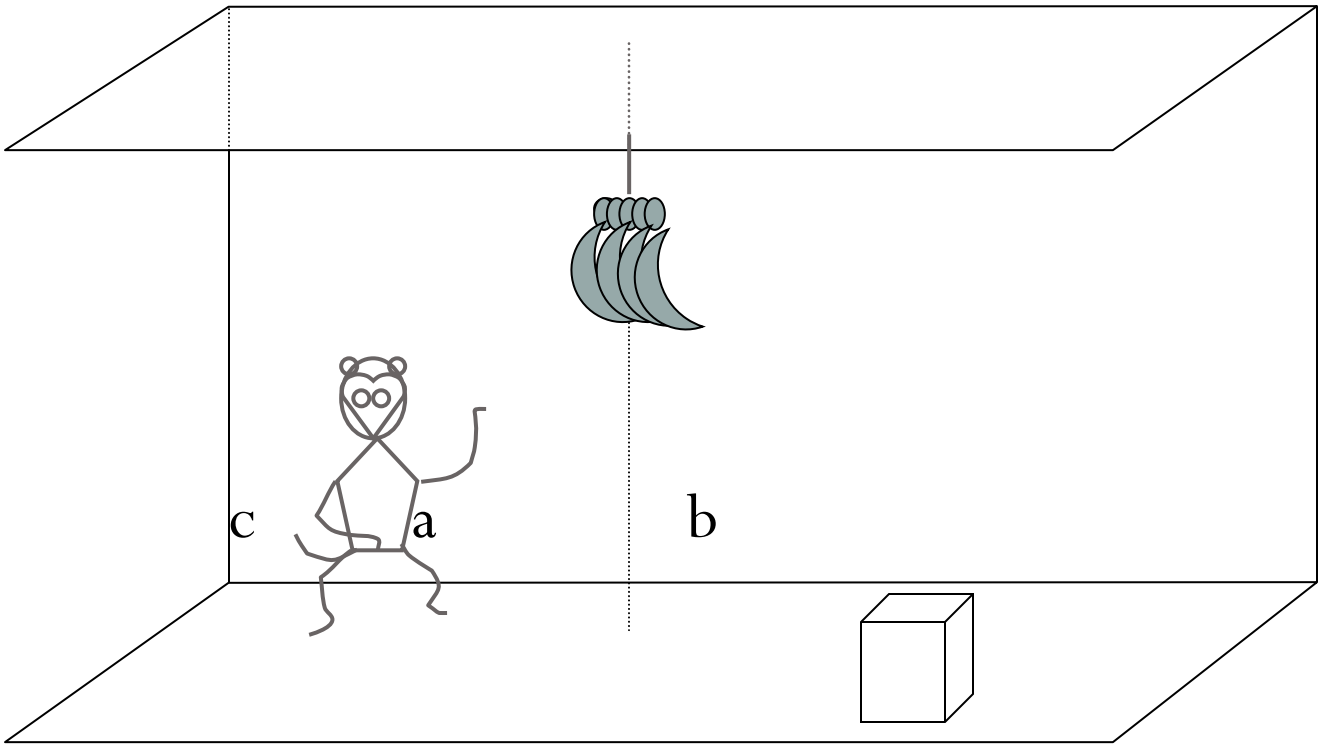
M-C问题（第二种方法）

4，规则集：

IF (m, c, 1) AND $1 \leq i+j \leq 2$ THEN (m-i, c-j, 0)

IF (m, c, 0) AND $1 \leq i+j \leq 2$ THEN (m+i, c+j, 1)

猴子摘香蕉问题



猴子摘香蕉问题（续1）

1，综合数据库

（M, B, Box, On, H）

M：猴子的位置

B：香蕉的位置

Box：箱子的位置

On=0：猴子在地板上

On=1：猴子在箱子上

H=0：猴子没有抓到香蕉

H=1：猴子抓到了香蕉

猴子摘香蕉问题（续2）

2，初始状态

（c, a, b, 0, 0）

3，结束状态

（x_1, x_2, x_3, x_4, 1）

其中x_1～x_4为变量。

猴子摘香蕉问题（续3）

4，规则集

r1: IF (x, y, z, 0, 0) THEN (w, y, z, 0, 0)【猴子可移动】

r2: IF (x, y, x, 0, 0) THEN (z, y, z, 0, 0)【猴子可以带同位置箱子动】

r3: IF (x, y, x, 0, 0) THEN (x, y, x, 1, 0)【猴子可以爬上箱子】

r4: IF (x, y, x, 1, 0) THEN (x, y, x, 0, 0)【猴子可以从箱子下来】

r5: IF (x, x, x, 1, 0) THEN (x, x, x, 1, 1)

【猴子、香蕉、箱子同一位置，且站在箱子上，可以摘香蕉】

其中x, y, z, w为变量

1.4 产生式系统的特点

- 数据驱动
- 知识的无序性
- 控制系统与问题无关
- 数据、知识和控制相互独立

1.5 产生式系统的类型

- 正向、逆向、双向产生式系统
- 可交换的产生式系统
- 可分解的产生式系统

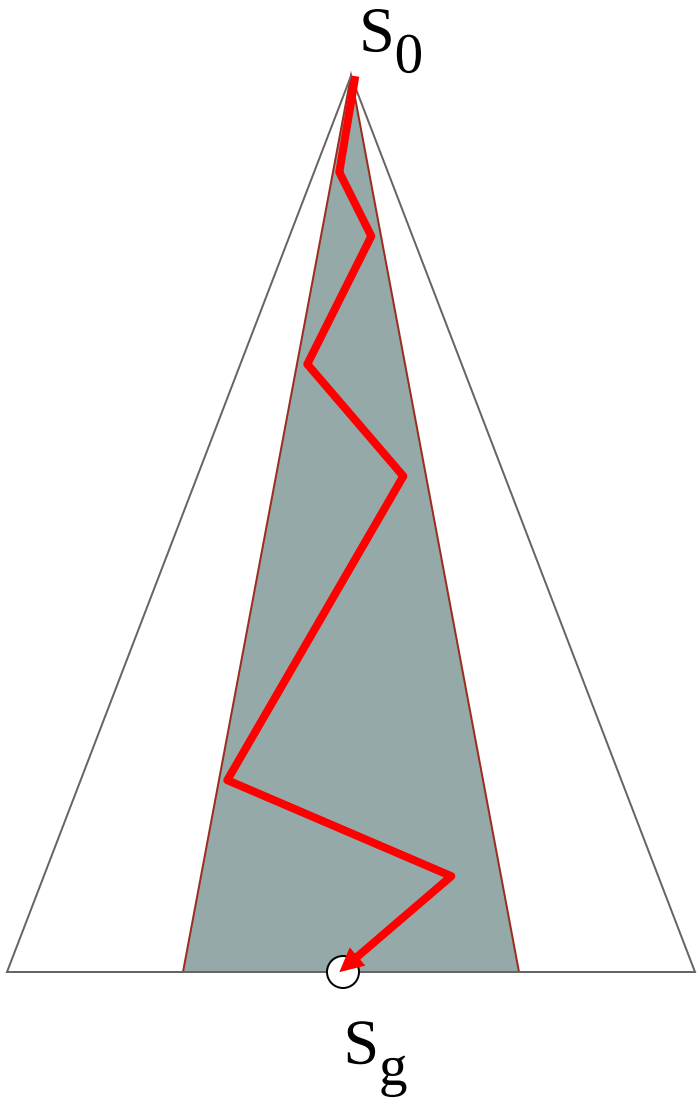
第二章 产生式系统的搜索策略

- 内容：

  状态空间的搜索问题。
- 搜索方式：
  - 盲目搜索
  - 启发式搜索
- 关键问题：

  如何利用知识，尽可能有效地找到问题的解（最佳解）。

产生式系统的搜索策略（续1）

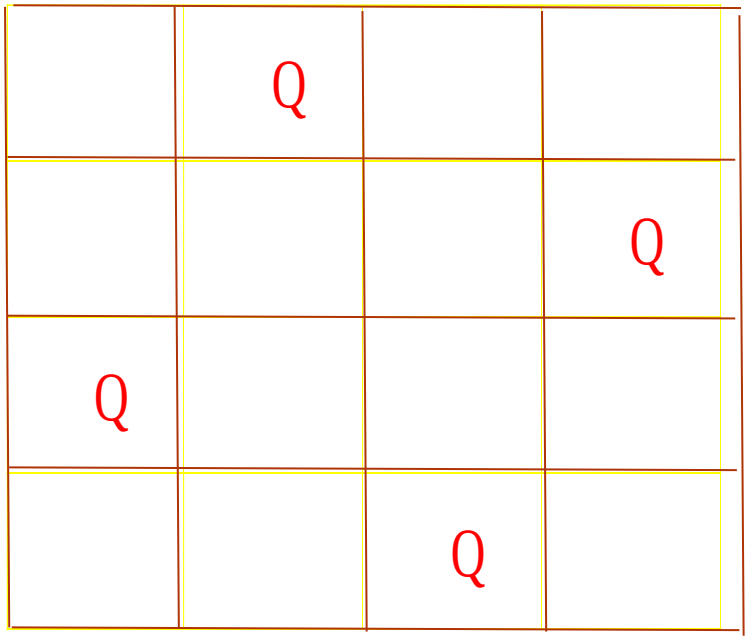


产生式系统的搜索策略（续2）

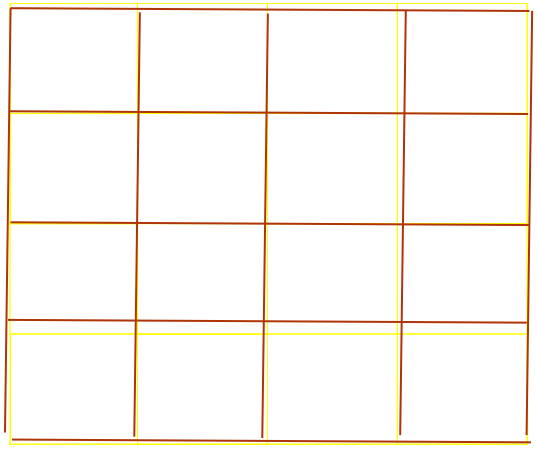
- 讨论的问题：
  - 有哪些常用的搜索算法。
  - 问题有解时能否找到解。
  - 找到的解是最佳的吗？
  - 什么情况下可以找到最佳解？
  - 求解的效率如何。

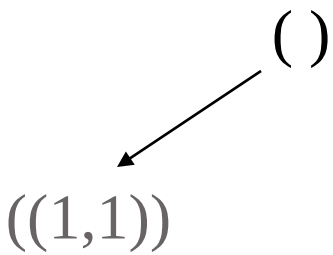
2.1 回溯策略

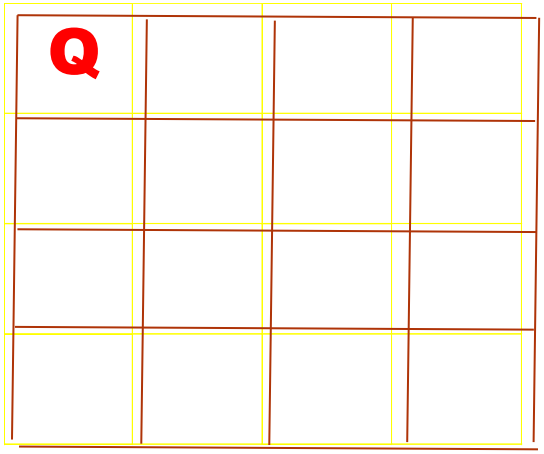
- 例：皇后问题

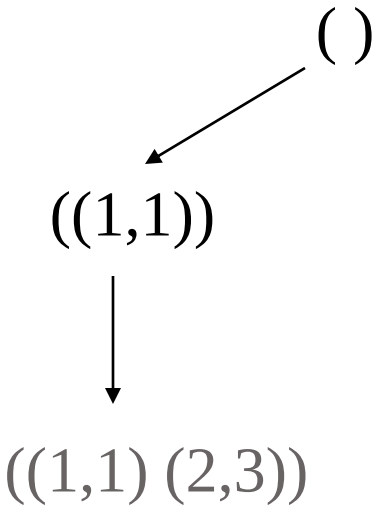


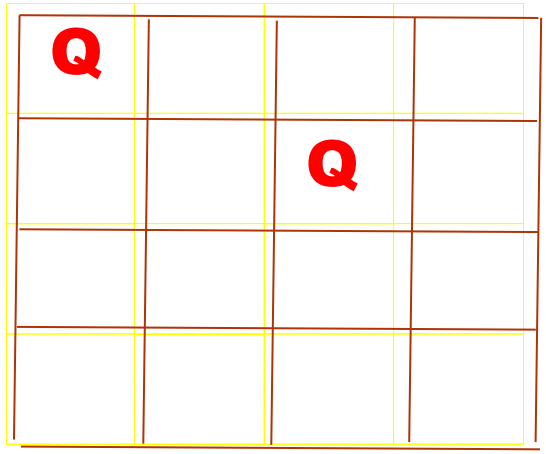
( )

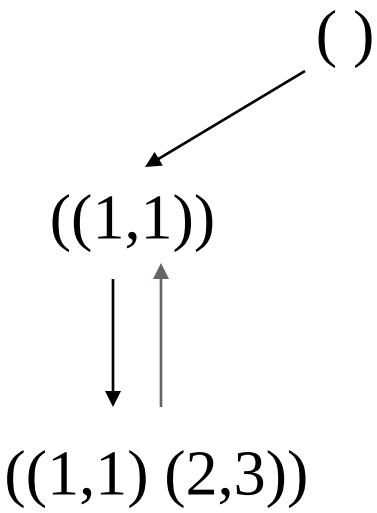


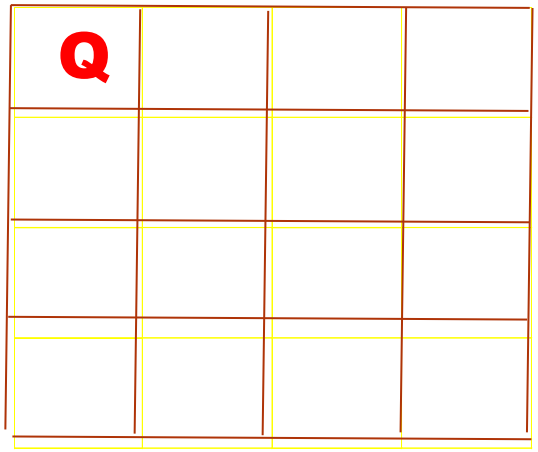

( )
((1,1))

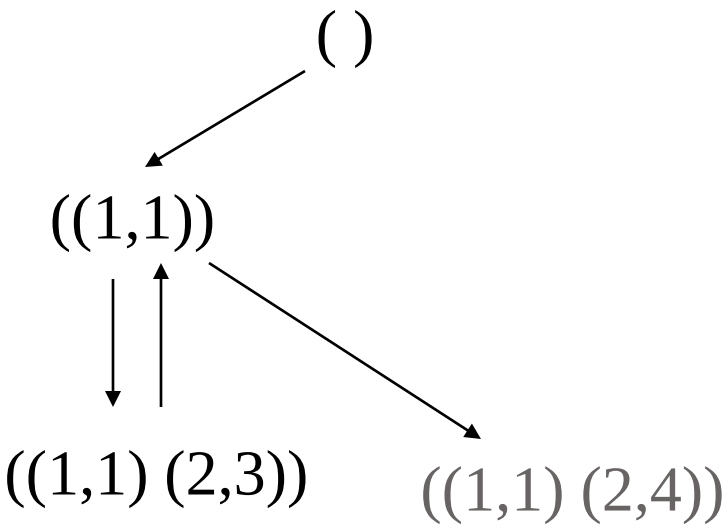

Q

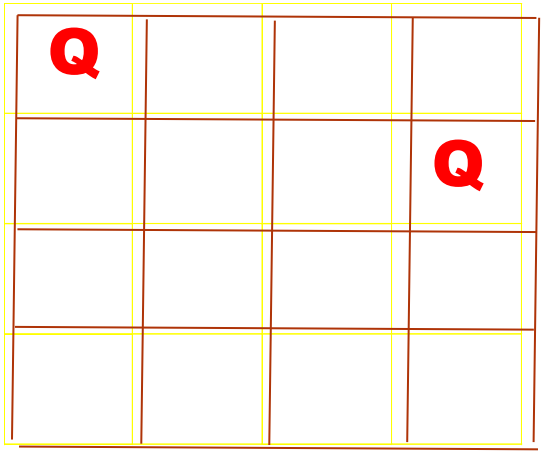

( )
((1,1))
((1,1) (2,3))

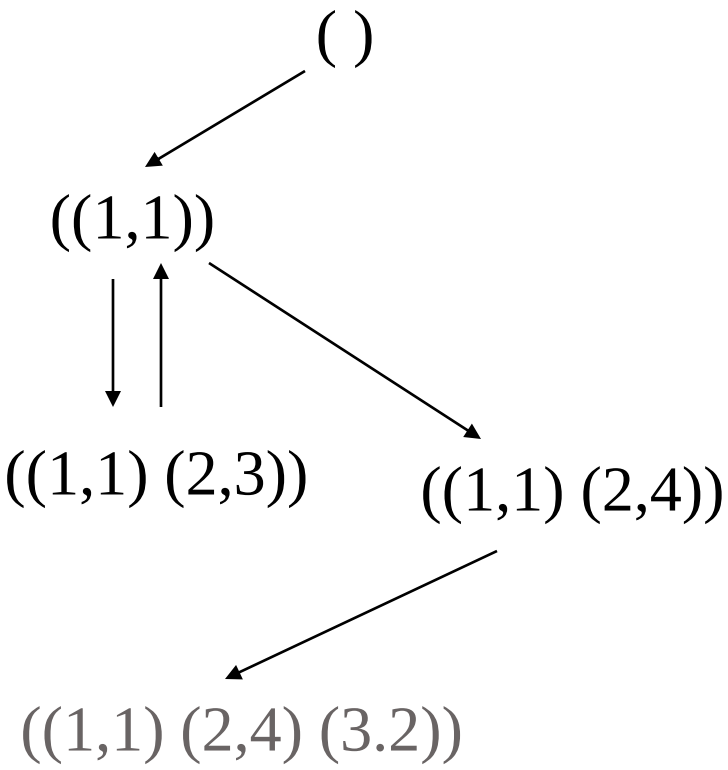

Q
Q

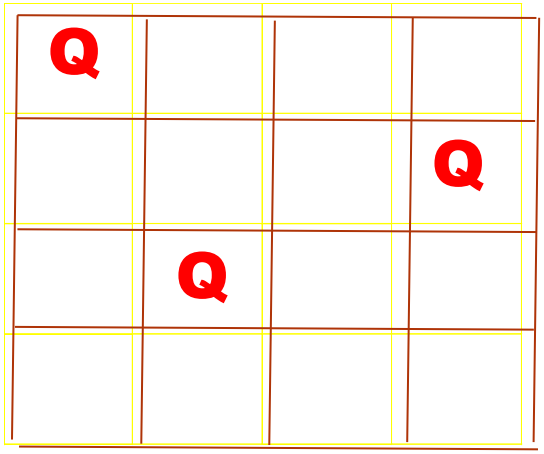

( )
((1,1))
((1,1) (2,3))

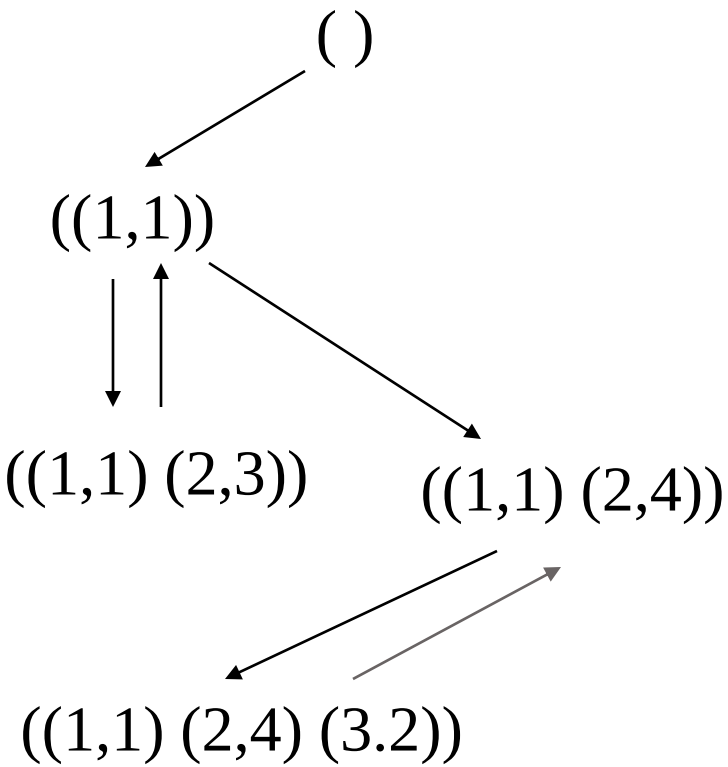

Q

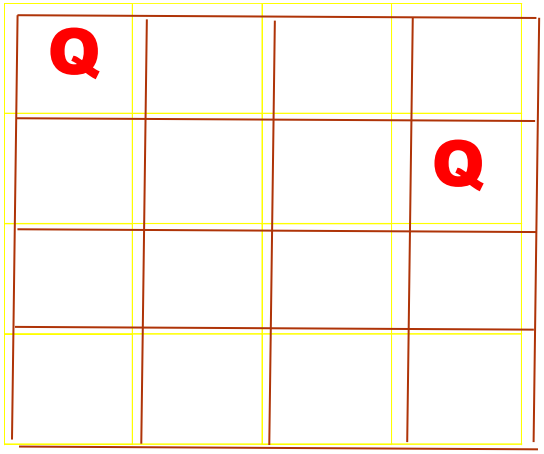

( )
((1,1))
((1,1) (2,3))
((1,1) (2,4))

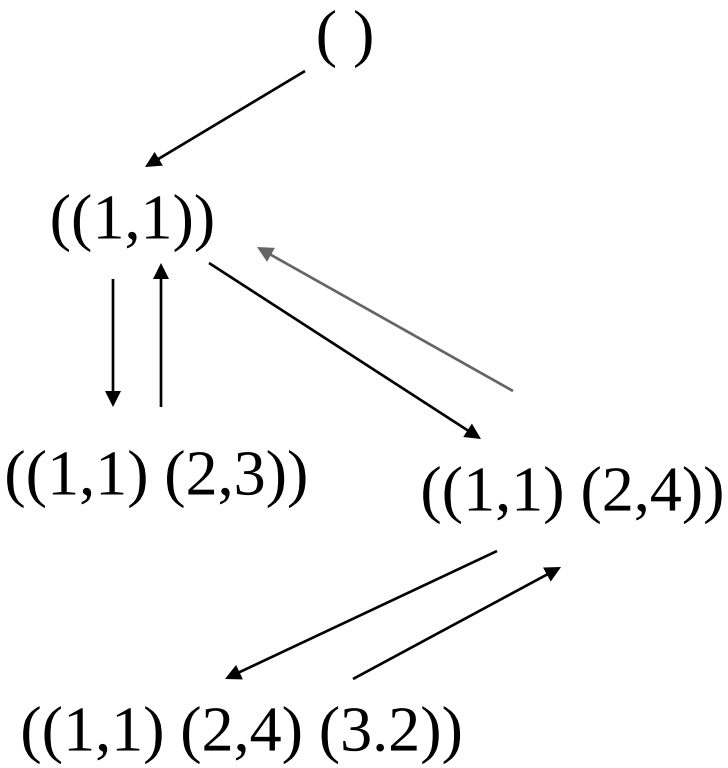

Q
Q

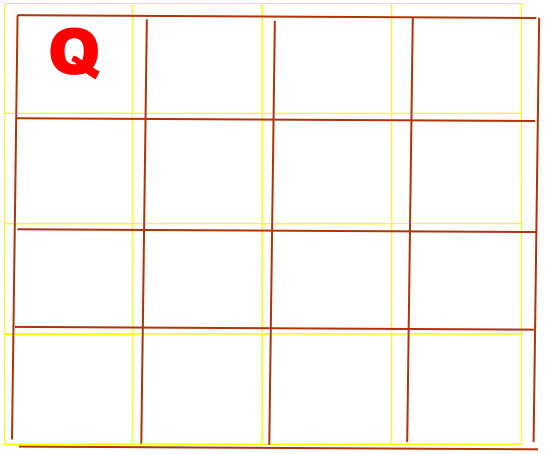

( )
((1,1))
((1,1) (2,3))
((1,1) (2,4))
((1,1) (2,4) (3.2))

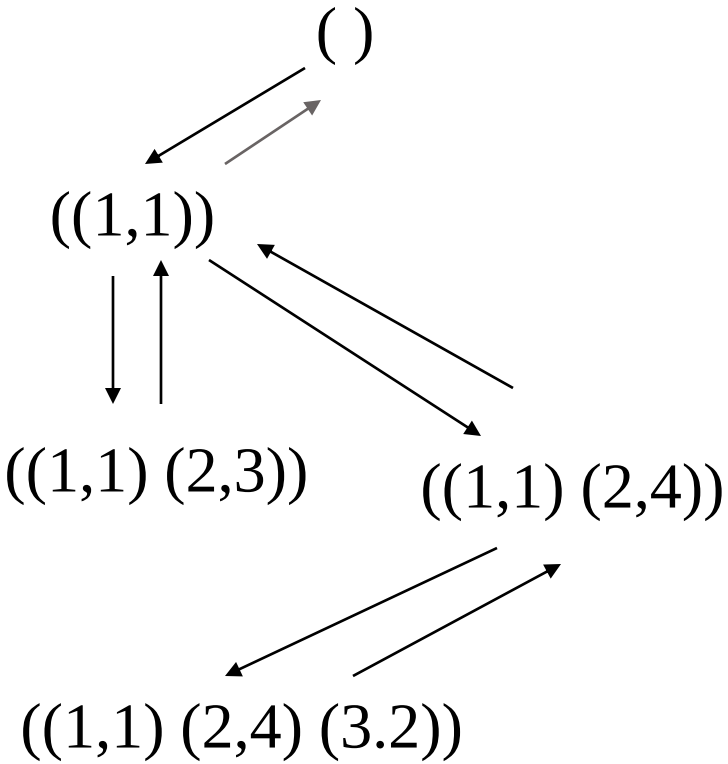

Q
Q
Q

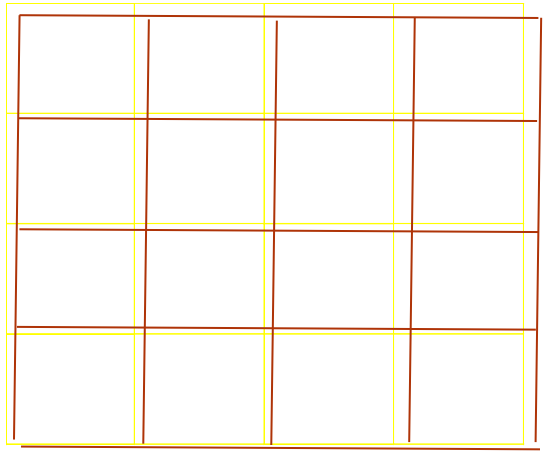

( )
((1,1))
((1,1) (2,3))
((1,1) (2,4))
((1,1) (2,4) (3.2))

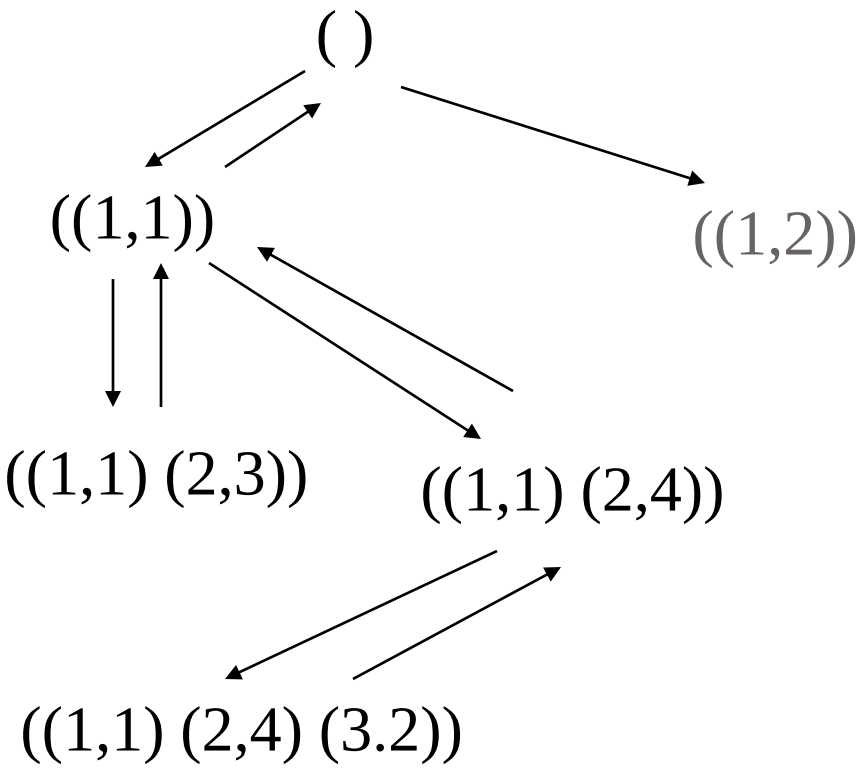

Q
Q

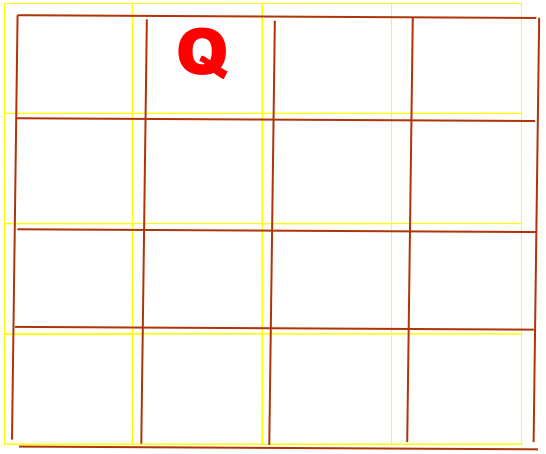

( )
((1,1))
((1,1) (2,3))
((1,1) (2,4))
((1,1) (2,4) (3.2))

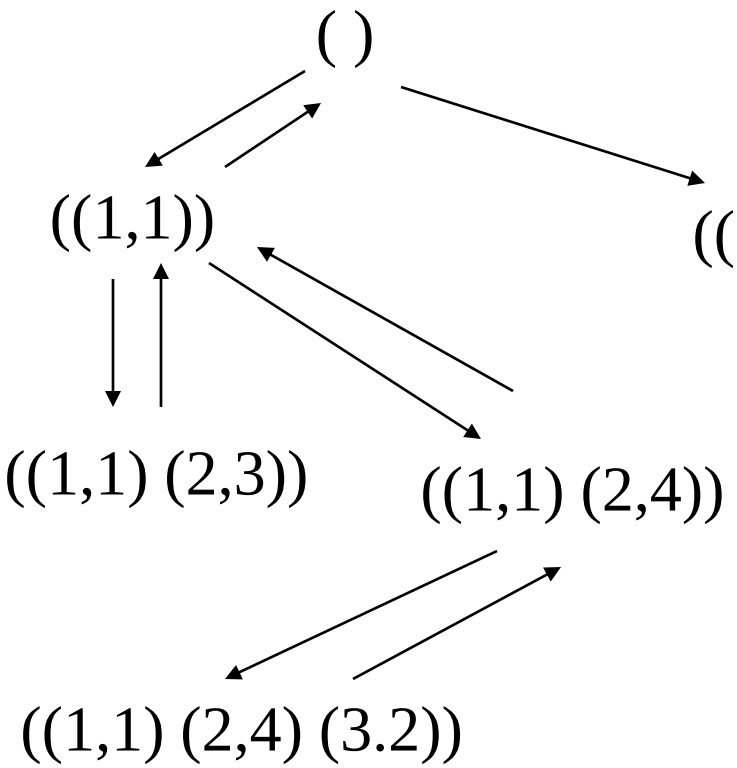

Q

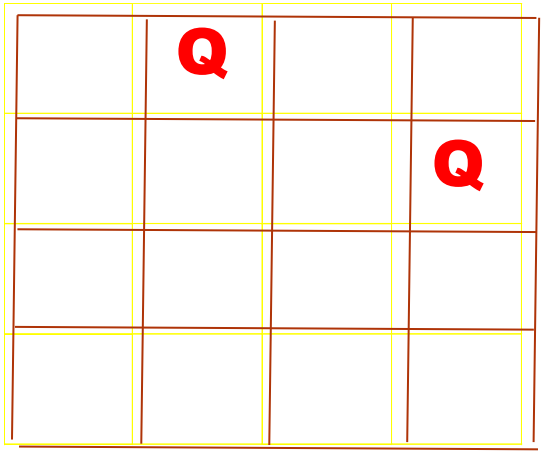

( )
((1,1))
((1,1) (2,3))
((1,1) (2,4))
((1,1) (2,4) (3.2))

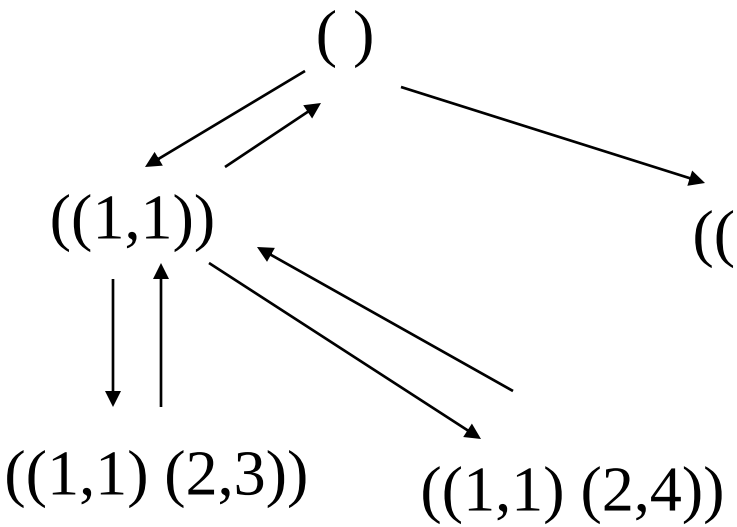


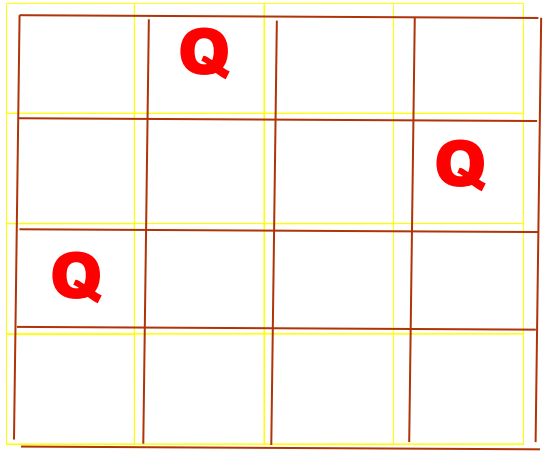

( )
((1,1))
((1,2))
((1,1) (2,3))
((1,1) (2,4))
((1,1) (2,4) (3.2))


Q

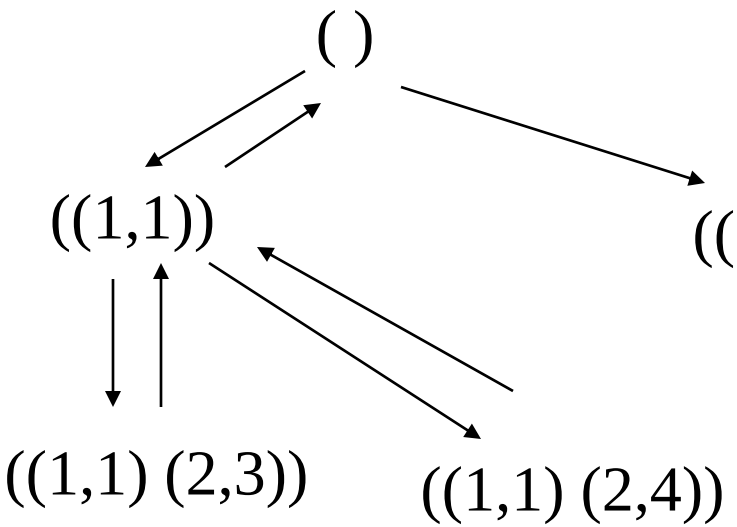

( )
((1,1))
((1,2))
((1,1) (2,3))
((1,1) (2,4))
((1,2) (2,4))
((1,1) (2,4) (3.2))

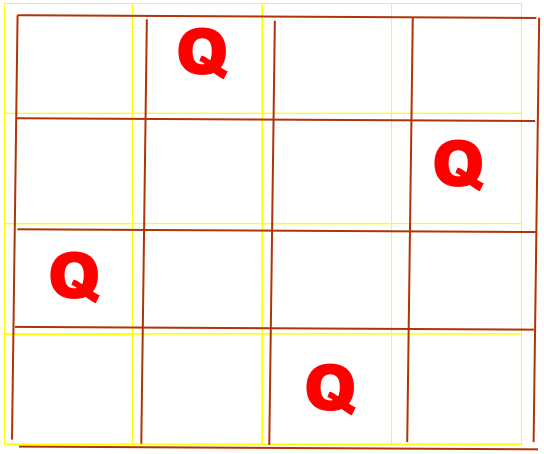

Q
Q

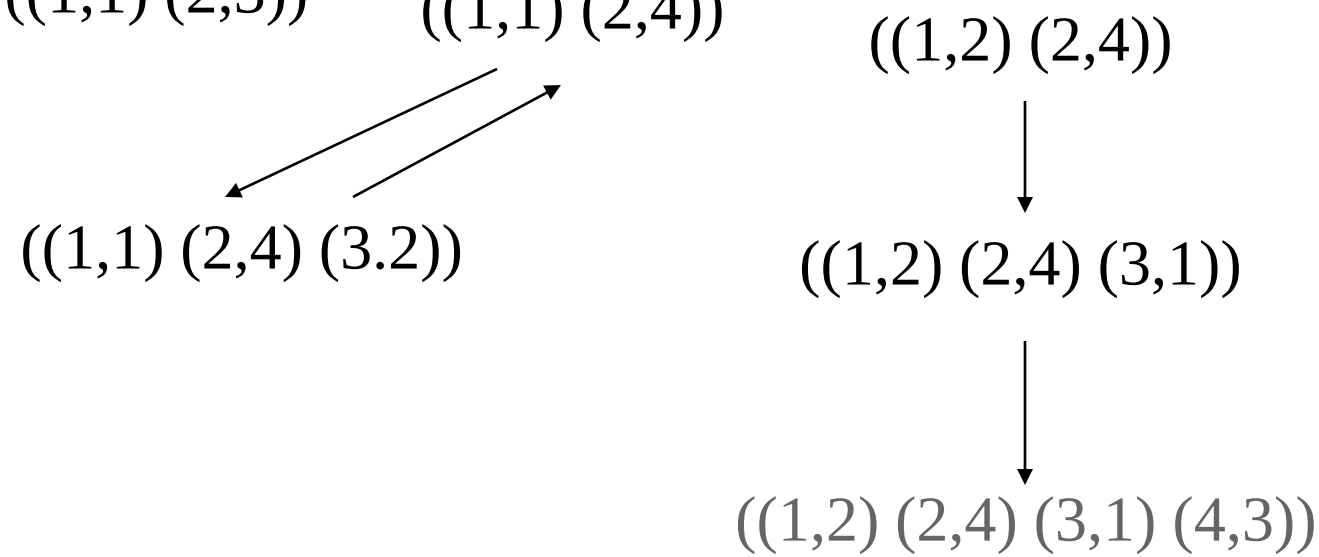



Q
Q
Q

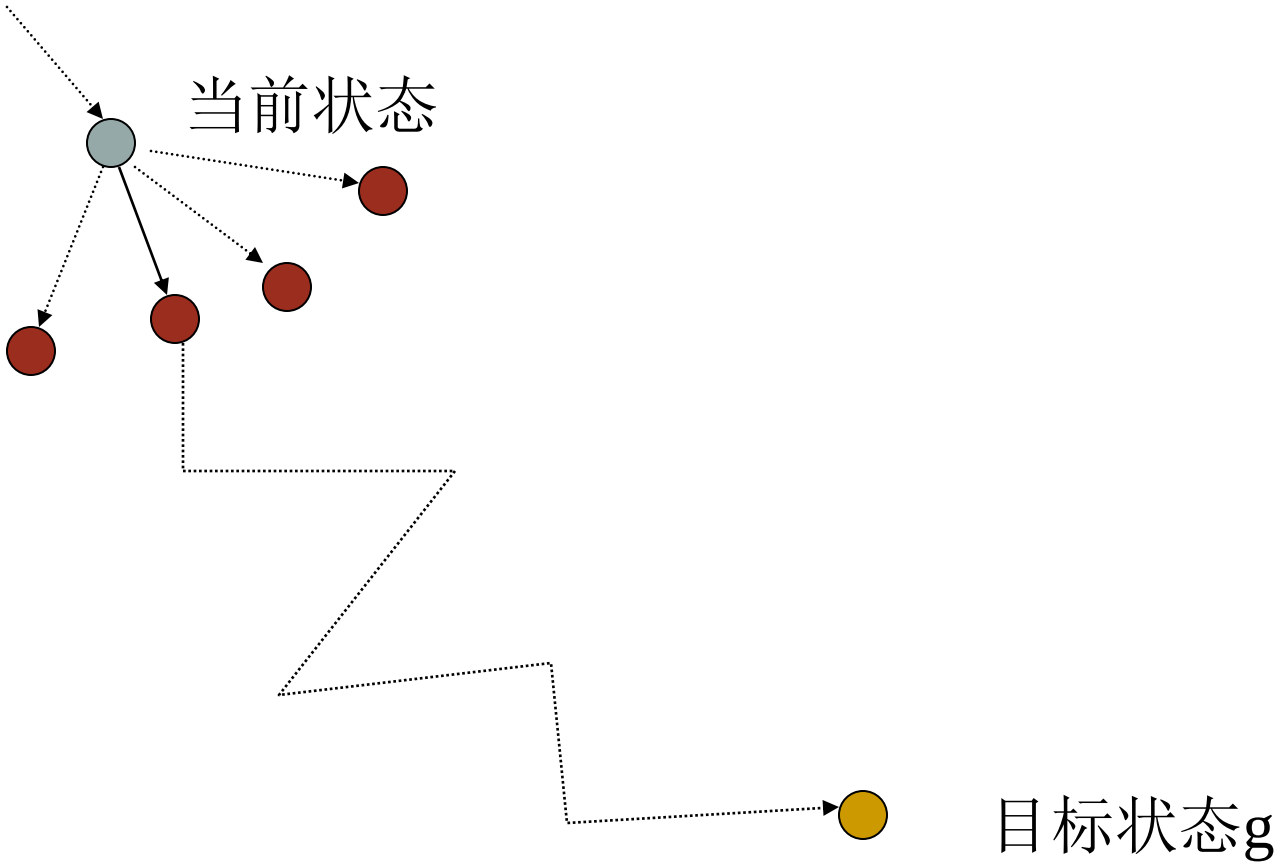
( )
((1,1))
((1,2))
((1,1) (2,3))
((1,1) (2,4))
((1,2) (2,4))
((1,1) (2,4) (3.2))
((1,2) (2,4) (3,1))


( )
((1,1))
((1,2))
((1,1) (2,3))
((1,1) (2,4))


Q
Q
Q
Q

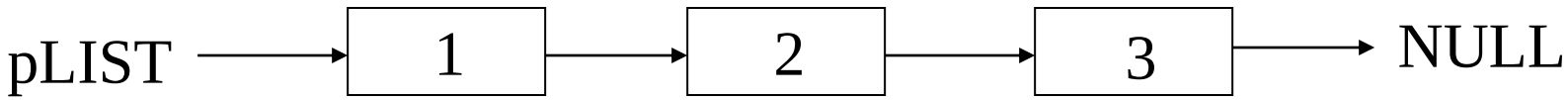

((1,2) (2,4))
((1,1) (2,4) (3.2))
((1,2) (2,4) (3,1))
((1,2) (2,4) (3,1) (4,3))

递归的思想



一个递归的例子

```
int ListLenght(LIST *pList)
{
   if (pList == NULL) return 0;
   else return ListLength(pList->next)+1;
}
```



回溯搜索算法

BACKTRACK（DATA）

DATA：当前状态。

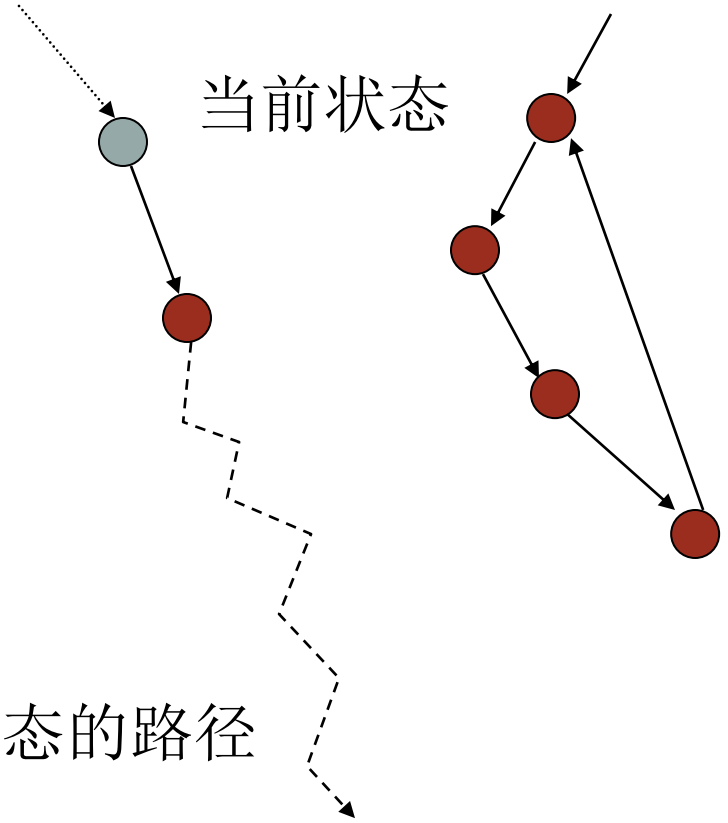
返回值：从当前状态到目标状态的路径
（以规则表的形式表示）
或FAIL。

回溯搜索算法

```
递归过程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,     RETURN CONS(R, PATH);
```

存在问题及解决办法

- 问题：
  - 深度问题
  - 死循环问题



- 解决办法：
  - 对搜索深度加以限制
  - 记录从初始状态到当前状态的路径

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。

回溯搜索算法1

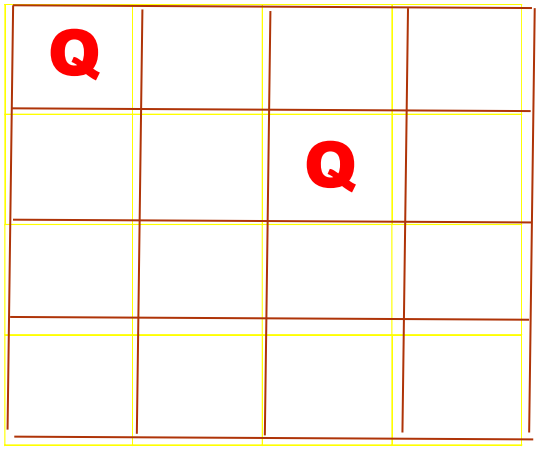
```
1,      DATA:=FIRST(DATALIST)
2,      IF MENBER(DATA,TAIL(DATALIST))
        RETURN FAIL;
3,      IFTERM(DATA) RETURN NIL;
4,      IF DEADEND(DATA) RETURN FAIL;
5,      IF LENGTH(DATALIST)>BOUND
        RETURN FAIL;
6,      RULES:=APPRULES(DATA);
7,  LOOP: IF NULL(RULES) RETURN FAIL;
8,      R:=FIRST(RULES);
```

回溯搜索算法1（续）

```
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);
```

一些深入的问题

- 失败原因分析、多步回溯

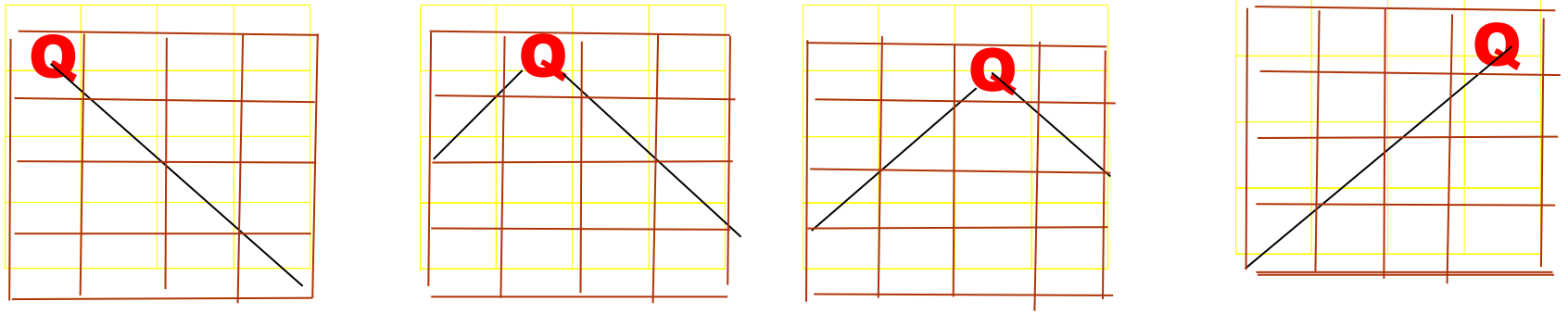


一些深入问题（续）

- 回溯搜索中知识的利用

  基本思想(以皇后问题为例)：

  尽可能选取划去对角线上位置数最少的。



3　　2　　2　　3

2.2 图搜索策略

- 问题的引出

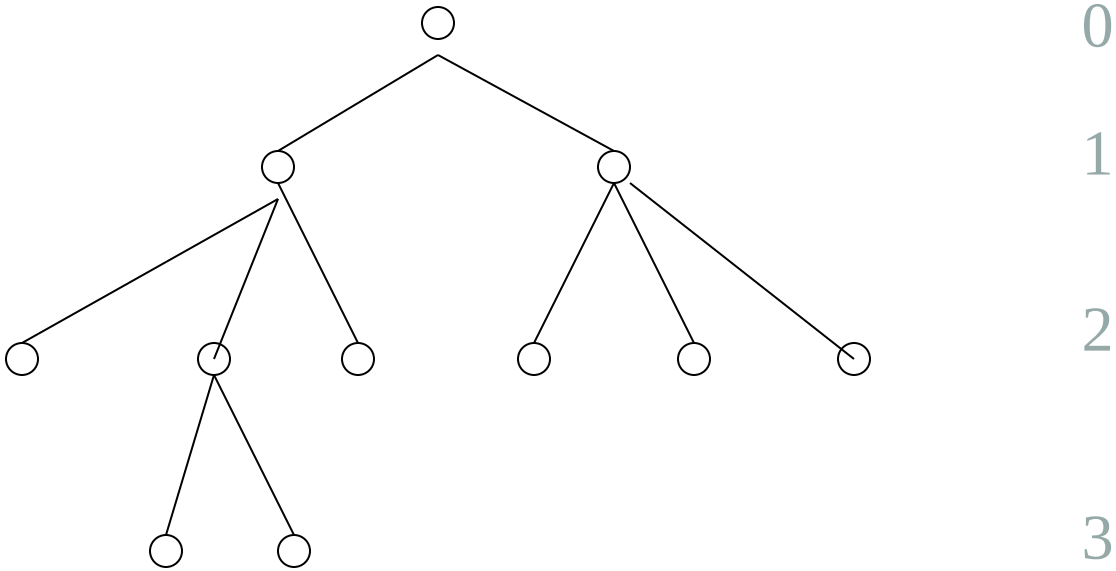
  - 回溯搜索：只保留从初始状态到当前状态的一条路径。
  - 图搜索：保留所有已经搜索过的路径。

一些基本概念

- 节点深度：

  根节点深度=0

  其它节点深度=父节点深度+1



一些基本概念（续1）

- 路径

  设一节点序列为$(n_0, n_1,\ldots,n_k)$，对于i=1,…,k，若节点n_{i-1}具有一个后继节点n_i，则该序列称为从n_0到n_k的路径。
- 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。

一些基本概念（续1）

- 扩展一个节点

  生成出该节点的所有后继节点，并给出它们之间的耗散值。这一过程称为“扩展一个节点”。

一般的图搜索算法

1, $G=G_0$ (G_0=s), OPEN:=(s);

2, CLOSED:=( );

3, LOOP: IF OPEN=( ) THEN EXIT(FAIL);

4, n:=FIRST(OPEN), REMOVE(n, OPEN),

ADD(n, CLOSED);

5, IF GOAL(n) THEN EXIT(SUCCESS);

6, EXPAND(n)→{m_i}, G:=ADD(m_i, G);

一般的图搜索算法（续）

7, 标记和修改指针：

ADD(m_j, OPEN), 并标记m_j到n的指针；

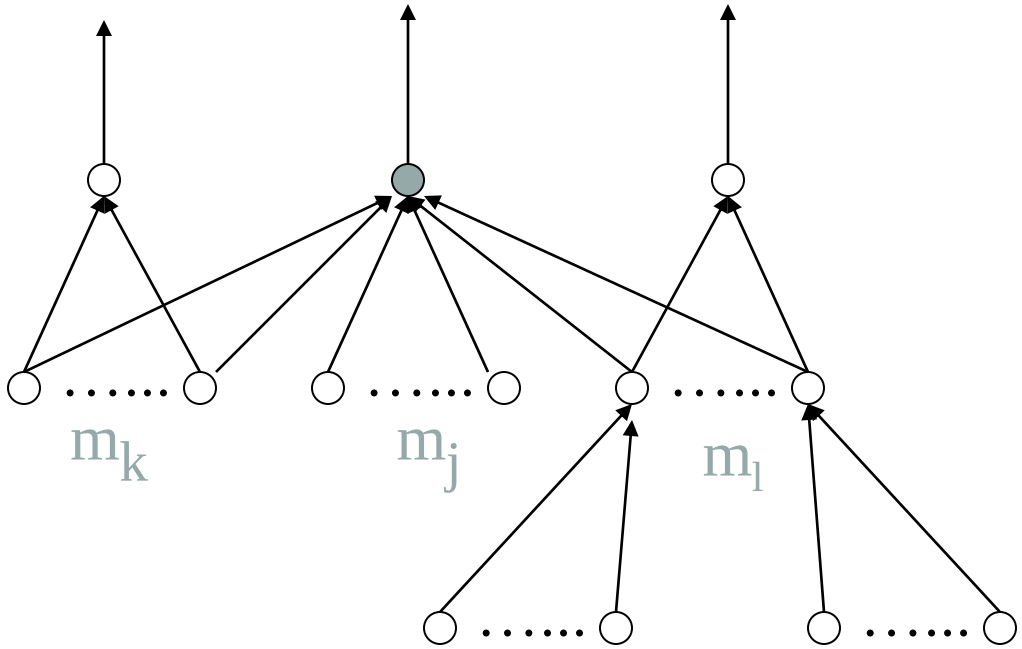
计算是否要修改m_k、m_l到n的指针；

计算是否要修改m_l到其后继节点的指针；

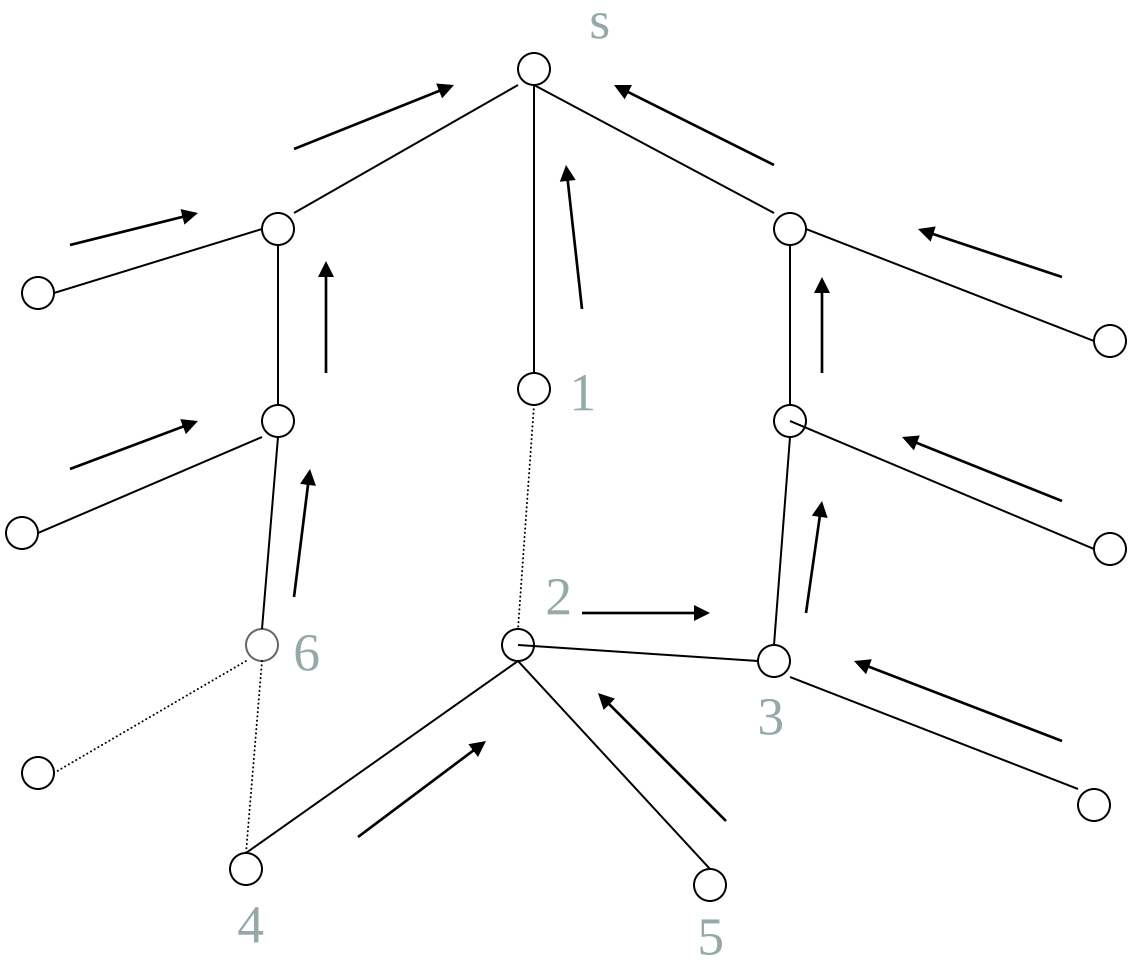
8, 对OPEN中的节点按某种原则重新排序；

9, GO LOOP；

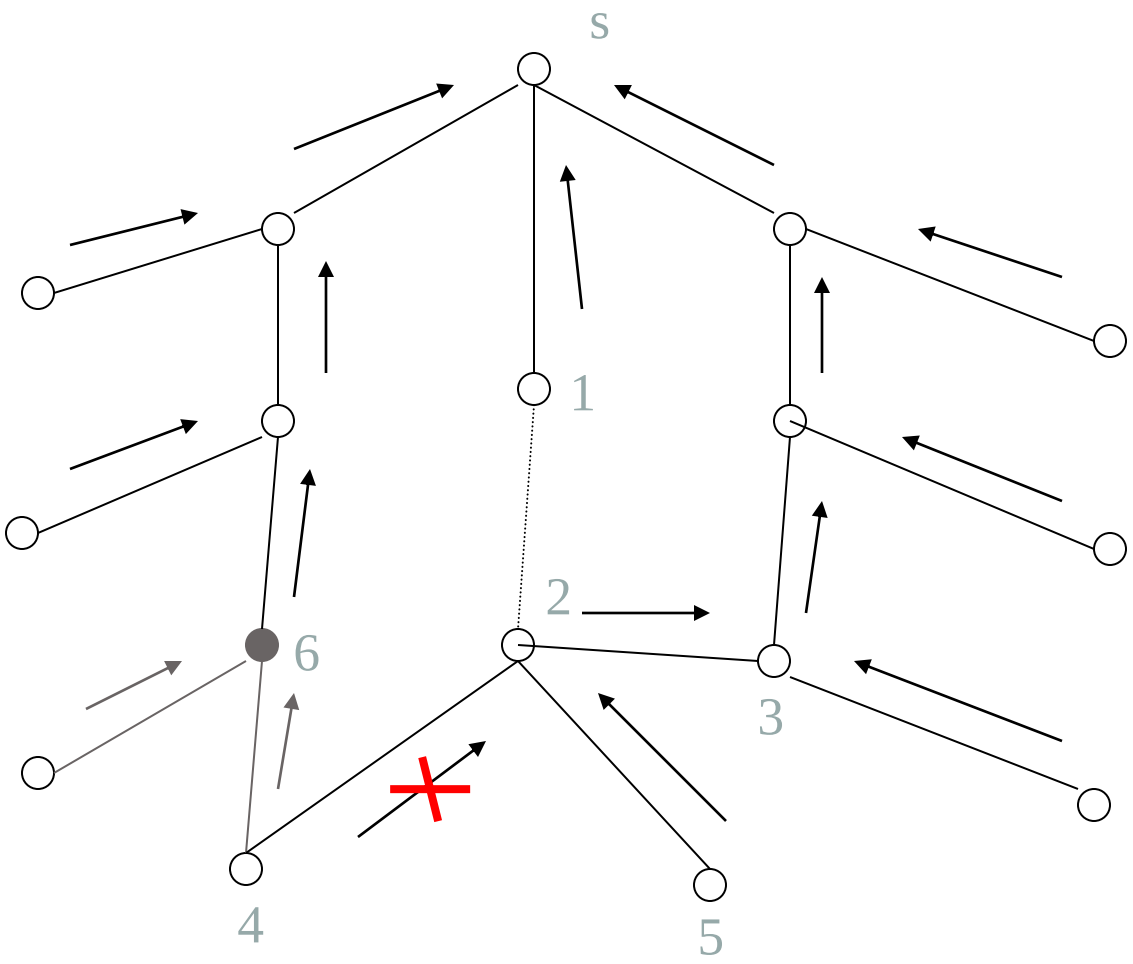
节点类型说明



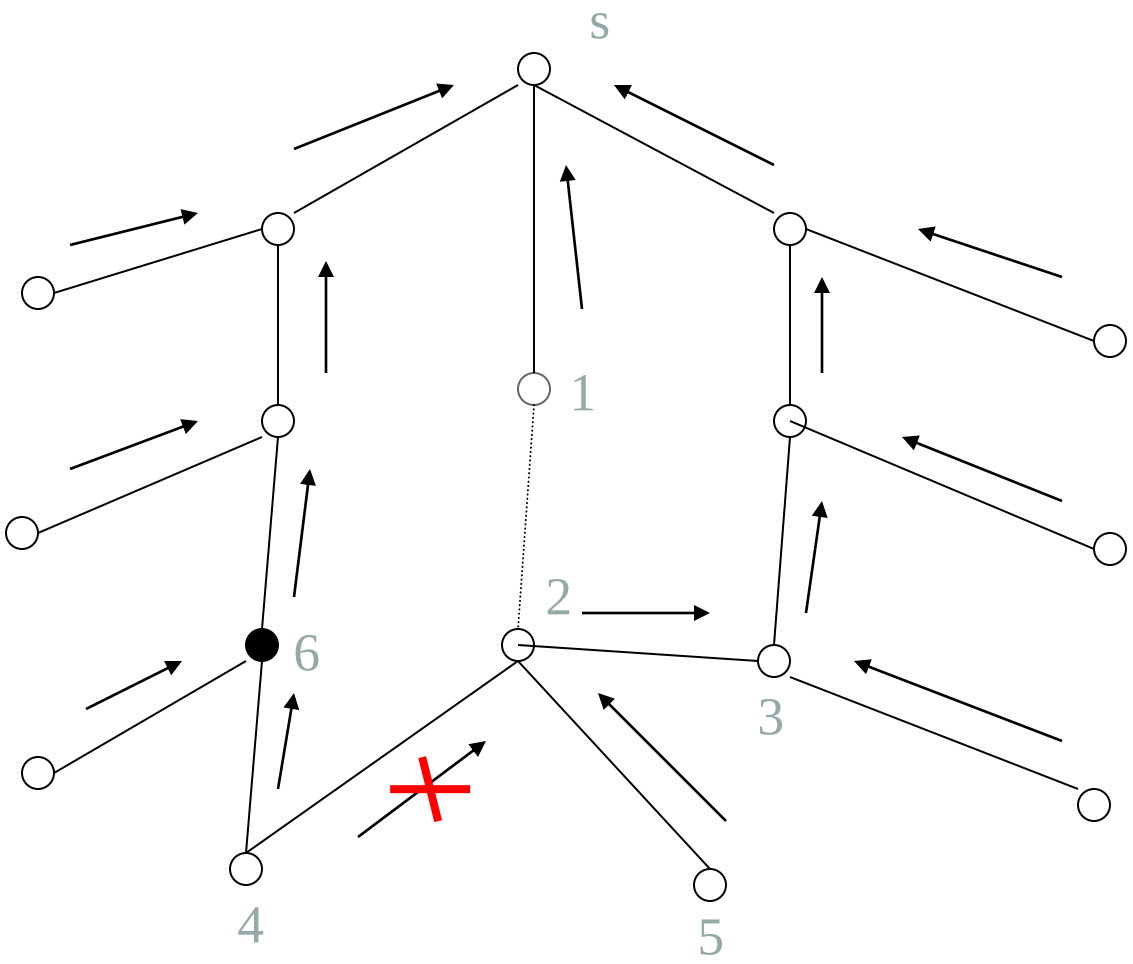
修改指针举例



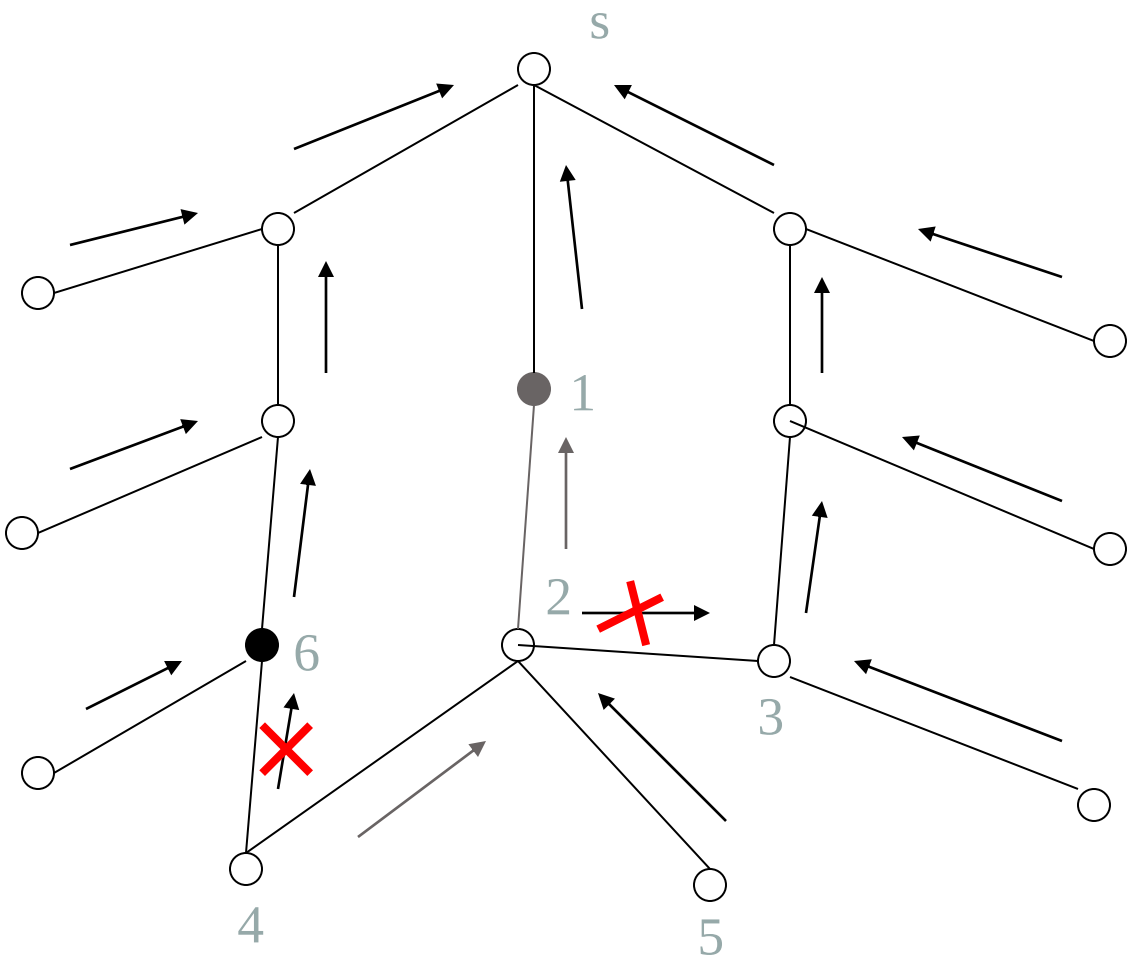
修改指针举例（续1）



修改指针举例（续2）



修改指针举例（续3）



2.3 无信息图搜索过程

- 深度优先搜索
- 宽度优先搜索

深度优先搜索

1, G:=G_0(G_0=s), OPEN:=(s), CLOSED:=( );

2, LOOP: IF OPEN=( ) THEN EXIT (FAIL);

3, n:=FIRST(OPEN);

4, IF GOAL(n) THEN EXIT (SUCCESS);

5, REMOVE(n, OPEN), ADD(n, CLOSED);

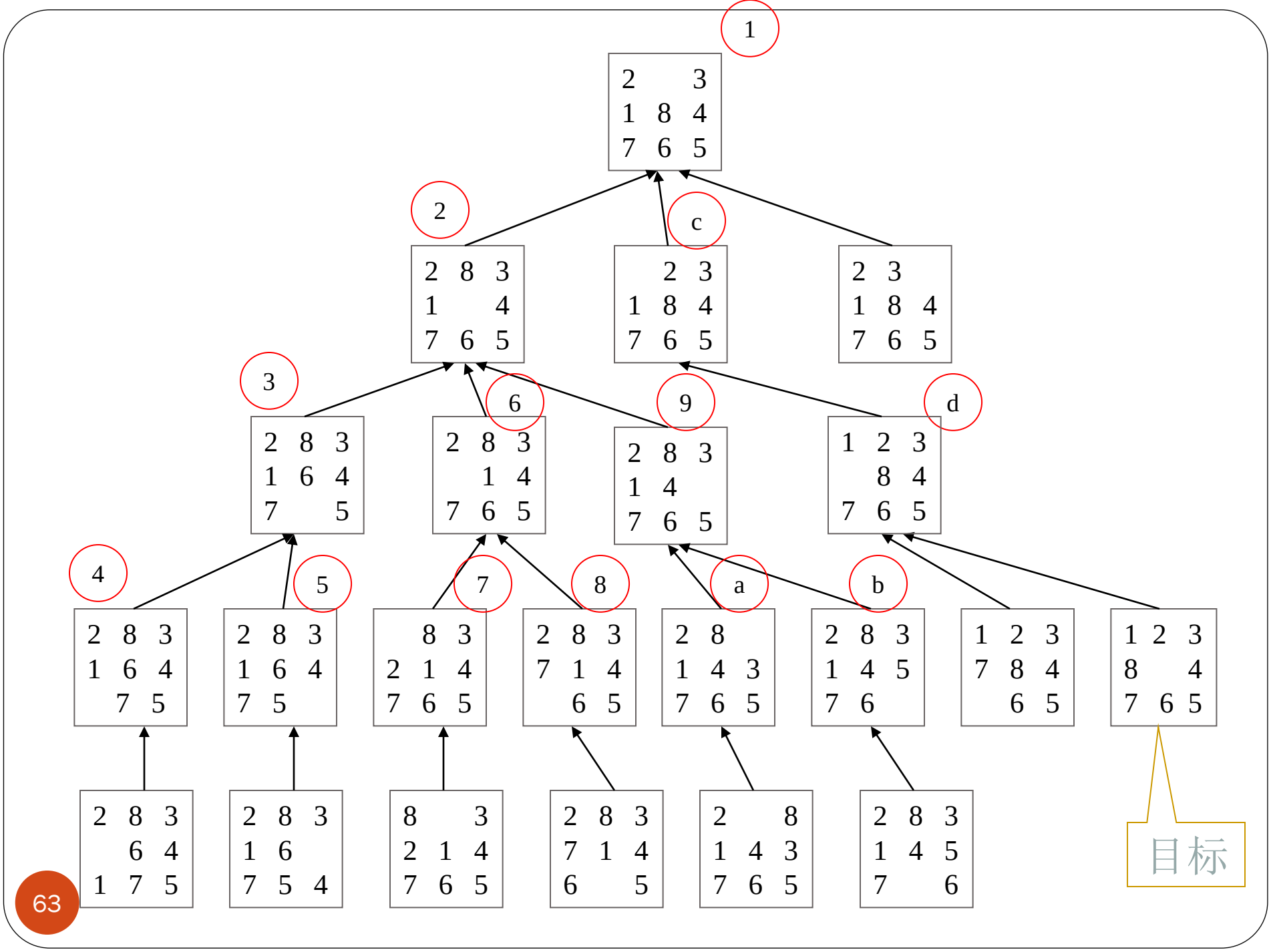
6, IF DEPTH(n)≥Dm GO LOOP;

7, EXPAND(n) → {m_i}, G:=ADD(m_i, G);

8, IF 目标在{m_i}中 THEN EXIT(SUCCESS);

9, ADD(m_j, OPEN), 并标记m_j到n的指针;

10, GO LOOP;

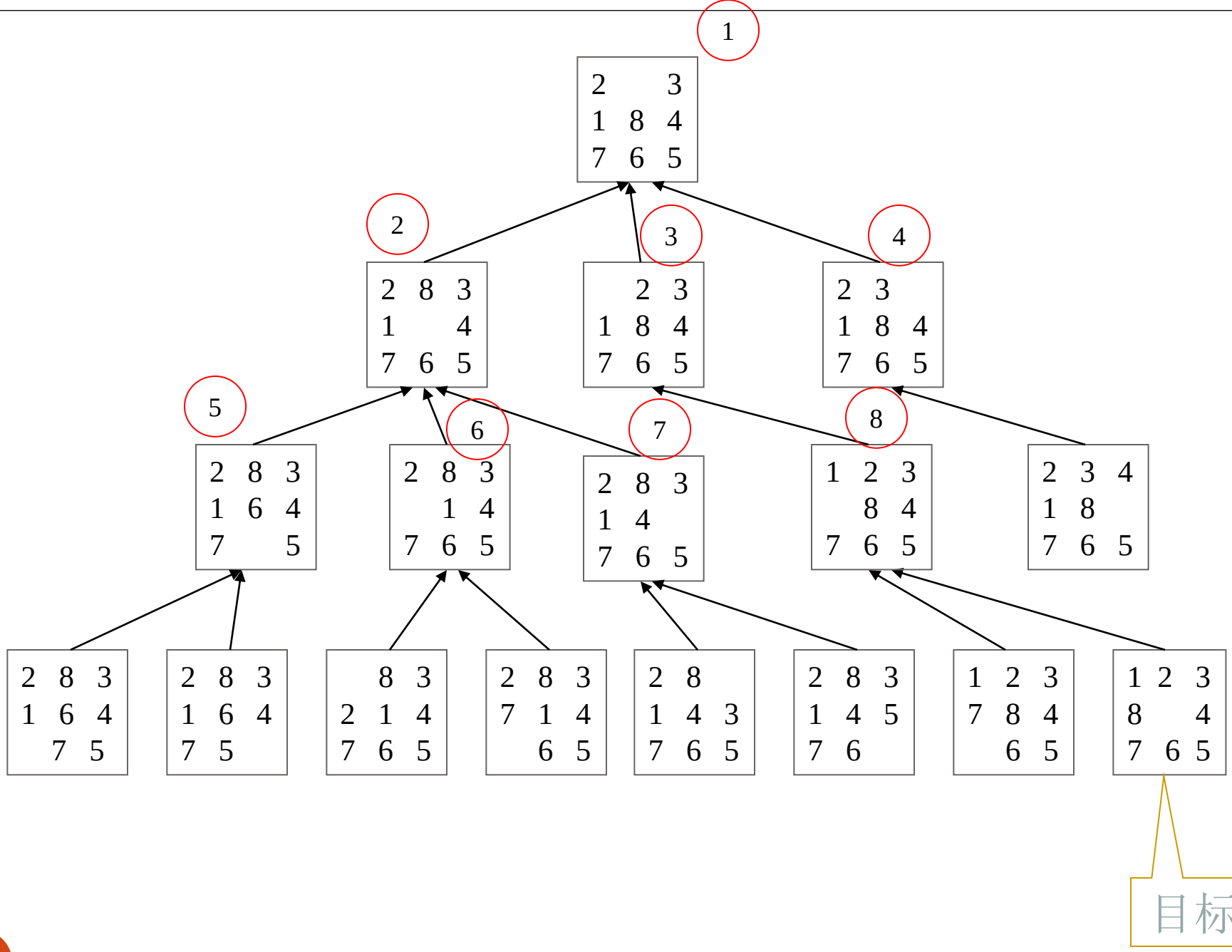

1
2 3
1 8 4
7 6 5
2
2 8 3
1 4
7 6 5
c
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
3
2 8 3
1 6 4
7 5
6
2 8 3
1 4
7 6 5
9
2 8 3
1 4
7 6 5
d
1 2 3
8 4
7 6 5
4
2 8 3
1 6 4
7 5
5
2 8 3
1 6 4
7 5
7
8 3
2 1 4
7 6 5
8
2 8 3
7 1 4
6 5
a
2 8
1 4 3
7 6 5
b
2 8 3
1 4 5
7 6
1 2 3
7 8 4
6 5
1 2 3
8 4
7 6 5
2 8 3
6 4
1 7 5
2 8 3
1 6
7 5 4
8 3
2 1 4
7 6 5
2 8 3
7 1 4
6 5
2 8
1 4 3
7 6 5
2 8 3
1 4 5
7 6
目标

深度优先搜索的性质

- 一般不能保证找到最优解
- 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，搜索空间等同于穷举
- 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

1, G:=G0(G0=s), OPEN:=(s), CLOSED:=( );
2, LOOP: IF OPEN=( ) THEN EXIT (FAIL);
3, n:=FIRST(OPEN);
4, IF GOAL(n) THEN EXIT (SUCCESS);
5, REMOVE(n, OPEN), ADD(n, CLOSED);
6, EXPAND(n) →{m_i}, G:=ADD(m_i, G);
7, IF 目标在{m_i}中 THEN EXIT(SUCCESS);
8, **ADD(OPEN, m_j),** 并标记m_j到n的指针;
9, GO LOOP;


1
2 3
1 8 4
7 6 5
2
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
4
2 3
1 8 4
7 6 5
5
2 8 3
1 6 4
7 5
6
2 8 3
1 4
7 6 5
7
2 8 3
1 4
7 6 5
8
1 2 3
8 4
7 6 5
2 3 4
1 8
7 6 5
2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
8 3
2 1 4
7 6 5
2 8 3
7 1 4
6 5
2 8
1 4 3
7 6 5
2 8 3
1 4 5
7 6
1 2 3
7 8 4
6 5
1 2 3
8 4
7 6 5
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
- 方法与问题无关，具有通用性
- 效率较低
- 属于图搜索方法

渐进式深度优先搜索方法

- 目的
  - 解决宽度优先方法的空间问题和回溯方法不能找到最优解的问题。
- 思想

  首先给回溯法一个比较小的深度限制，然后逐渐增加深度限制，直到找到解或找遍所以分支为止。

2.4 启发式图搜索

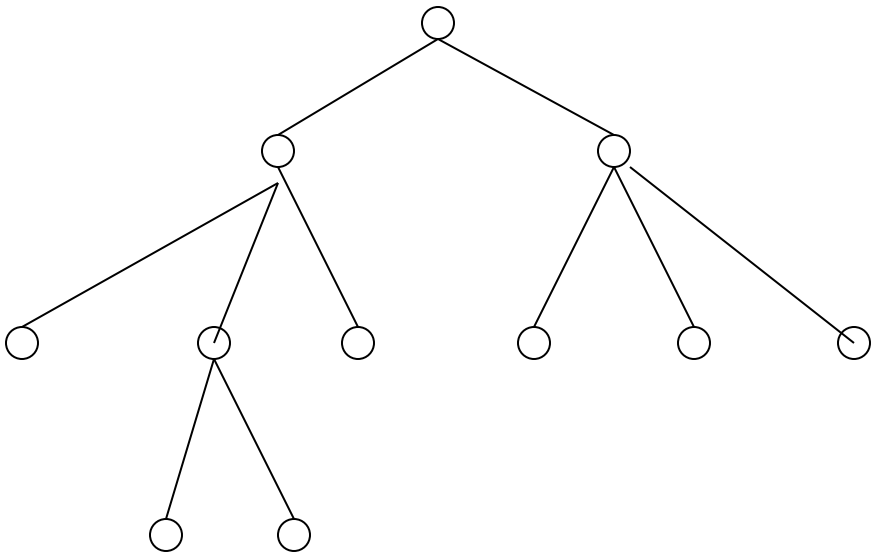
- 利用知识来引导搜索，达到减少搜索范围，降低问题复杂度的目的。
- 启发信息的强度
  - 强：降低搜索工作量，但可能导致找不到最优解
  - 弱：一般导致工作量加大，极限情况下变为盲目搜索，但可能可以找到最优解

希望：

- 引入启发知识，在保证找到最佳解的情况下，尽可能减少搜索范围，提高搜索效率。

基本思想

- 定义一个评价函数f，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。



1，启发式搜索算法A（A算法）

- 评价函数的格式：

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

f(n)：评价函数

h(n)：启发函数

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

A算法

1, OPEN:=(s), f(s):=g(s)+h(s);

2, LOOP: IF OPEN=( ) THEN EXIT(FAIL);

3, n:=FIRST(OPEN);

4, IF GOAL(n) THEN EXIT(SUCCESS);

5, REMOVE(n, OPEN), ADD(n, CLOSED);

6, EXPAND(n) → $\{m_i\}$,

计算$f(n, m_i):=g(n, m_i)+h(m_i)$;

A算法（续）

ADD(m_j, OPEN), 标记m_j到n的指针；

IF $f(n, m_k)<f(m_k)$ THEN $f(m_k):=f(n, m_k)$,

标记m_k到n的指针；

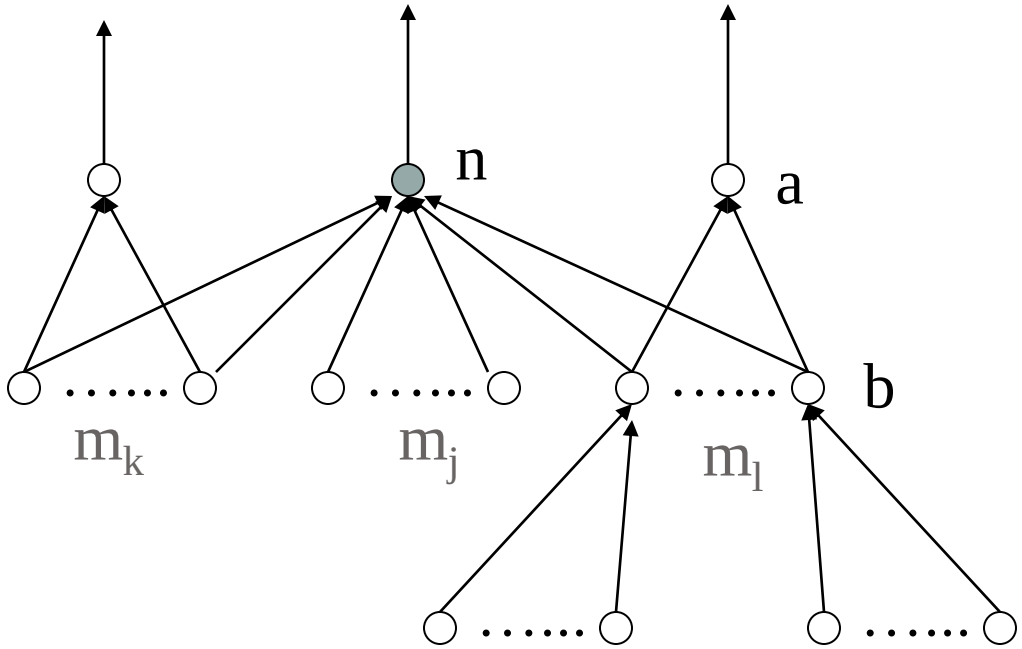
IF $f(n, m_l)<f(m_l,)$ THEN $f(m_l):=f(n, m_l)$,

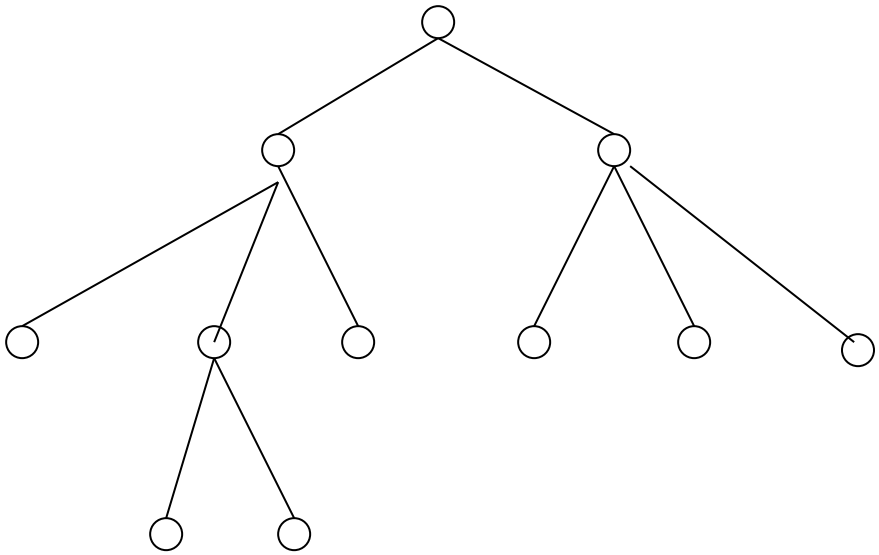
标记m_l到n的指针,

ADD(m_l, OPEN);

7, OPEN中的节点按f值从小到大排序；

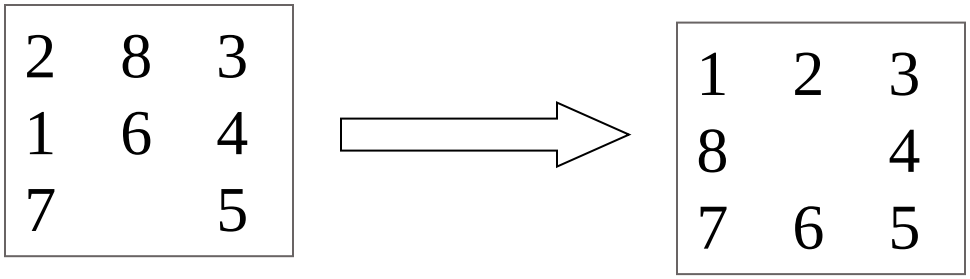
8, GO LOOP；


n
a
b
m_k
m_j
m_l



- Closed表
- Open表

一个A算法的例子



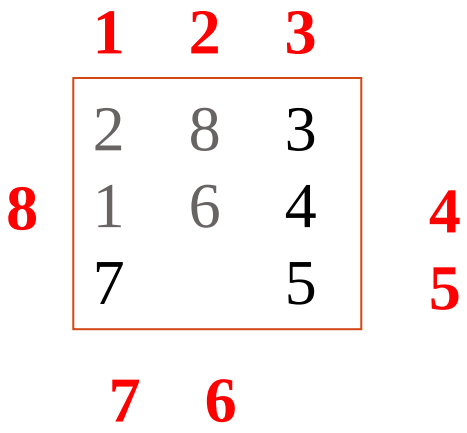
定义评价函数：

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

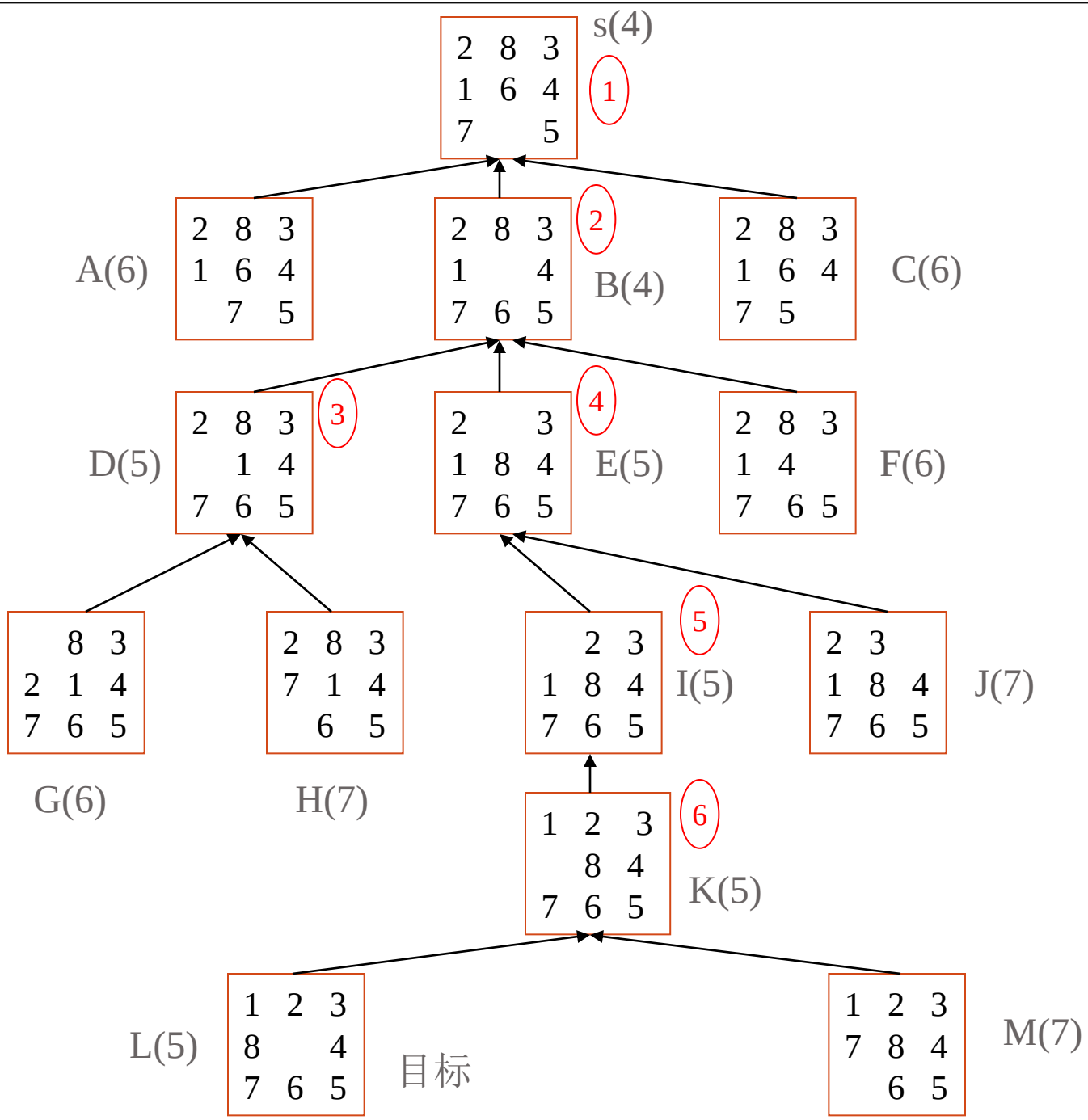
$g(n)$为从初始节点到当前节点的耗散值

$h(n)$为当前节点“不在位”的将牌数

h计算举例



$h(n) = 4$


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

2，最佳图搜索算法A*（A*算法）

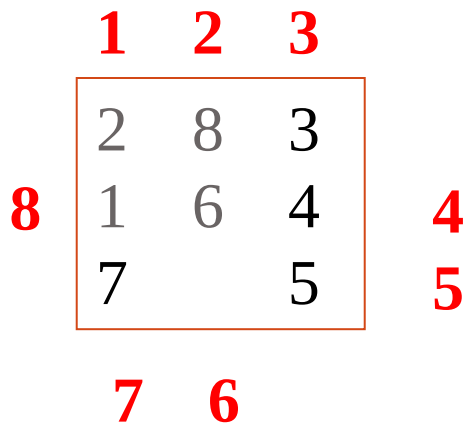
- 在A算法中，如果满足条件：

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

  则A算法称为A*算法。

A*条件举例

- 8数码问题
  - h1(n) = “不在位”的将牌数
  - h2(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

A*算法的性质

- A*算法的假设

  设n_i、n_j是任意两个节点，有：

  $C(n_i, n_j) > \varepsilon$

  其中ε为大于0的常数

- 几个等式

  $f^*(s) = f^*(t) = h^*(s) = g^*(t) = f^*(n)$

  其中s是初始节点，t是目标节点，n是s到t的最佳路径上的节点。

A*算法的性质（续1）

定理1：

对有限图，如果从初始节点s到目标节点t有路径存在，则算法A一定成功结束。

A*算法的性质（续2）

引理2.1：

对无限图，若有从初始节点s到目标节点t的路径，则A*不结束时，在OPEN表中即使最小的一个f值也将增到任意大，或有$f(n)>f^*(s)$。

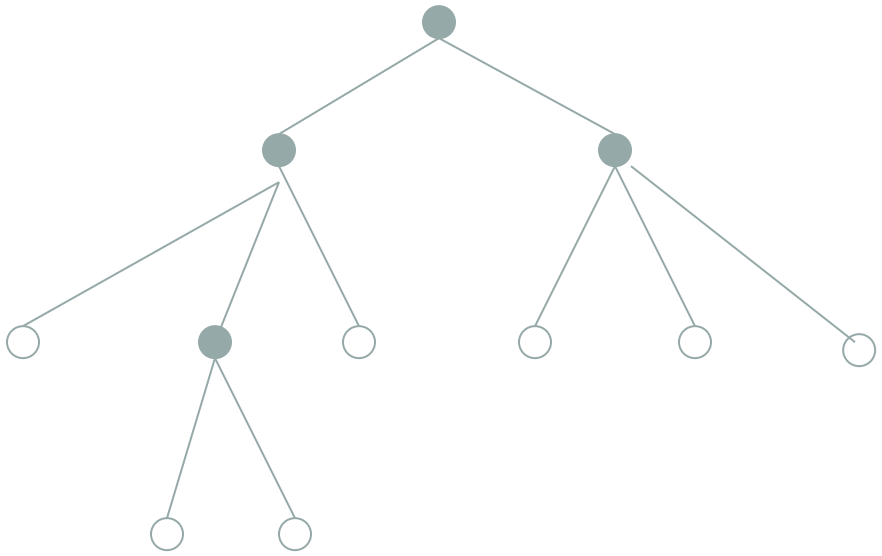
A*算法的性质（续3）

引理2.2：

A*结束前，OPEN表中必存在$f(n) \leq f^*(s)$。

存在一个节点n，n在最佳路径上。

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



A*算法的性质（续3）

定理2：

对无限图，若从初始节点s到目标节点t有路径存在，则A*一定成功结束。

引理2.1：A*如果不结束，则OPEN中所有的n有 $f(n) > f^*(s)$

引理2.2：在A*结束前，必存在节点n，使得 $f(n) \leq f^*(s)$

所以，如果A*不结束，将导致矛盾。

A*算法的性质（续4）

推论2.1：

OPEN表上任一具有$f(n)<f^*(s)$的节点n，最终都将被A*选作扩展的节点。

由定理2，知A*一定结束，由A*的结束条件，OPEN表中f(t)最小时才结束。而

$$f(t) \geq f^*(t) = f^*(s)$$

所以$f(n)<f^*(s)$的n，均被扩展。得证。

A*算法的性质（续5）

定理3 (可采纳性定理)：

若存在从初始节点s到目标节点t有路径，则A*必能找到最佳解结束。

可采纳性的证明

- 由定理1、2知A*一定找到一条路径结束
- 设找到的路径s→ t不是最佳的（t为目标）

  则：$f(t) = g(t) > f^*(s)$
- 由引理2.2知结束前OPEN中存在$f(n) \le f^*(s)$的节点n，所以

  $$f(n) \le f^*(s) < f(t)$$
- 因此A*应选择n扩展，而不是t。与假设A*选择t结束矛盾。得证。
- 注意：A*的结束条件

A*算法的性质（续6）

推论3.1：

A*选作扩展的任一节点n，有$f(n) \leq f^*(s)$。

- 由引理2.2知在A*结束前，OPEN中存在节点n’，$f(n') \leq f^*(s)$
- 设此时A*选择n扩展。
- 如果n＝n’，则$f(n) \leq f^*(s)$，得证。
- 如果n≠ n’，由于A*选择n扩展，而不是n’，所以有$f(n) \leq f(n') \leq f^*(s)$。得证。

A*算法的性质（续7）

定理4：设对同一个问题定义了两个A*算法A_1和A_2，若A_2比A_1有较多的启发信息，即对所有非目标节点有$h_2(n) > h_1(n)$，则在具有一条从s到t的路径的隐含图上，搜索结束时，由A_2所扩展的每一个节点，也必定由A_1所扩展，即A_1扩展的节点数至少和A_2一样多。

简写：如果$h_2(n) > h_1(n)$ (目标节点除外)，则A_1扩展的节点数≥A_2扩展的节点数

A*算法的性质（续7）

- 注意：

  在定理4中，评价指标是“扩展的节点数”，也就是说，同一个节点无论被扩展多少次，都只计算一次。

定理4的证明

- 使用数学归纳法，对节点的深度进行归纳
- （1）当d(n)＝0时，即只有一个节点，显然定理成立。
- （2）设d(n)≤k时定理成立。（归纳假设）
- （3）当d(n)=k+1时，用反证法。
- 设存在一个深度为k＋1的节点n，被A_2扩展，但没有被A_1扩展。而由假设，A_1扩展了n的父节点，即n已经被生成了。因此当A_1结束时，n将被保留在OPEN中。

定理4的证明（续1）

- 所以有：$f_1(n) \geq f^*(s)$
- 即：$g_1(n)+h_1(n) \geq f^*(s)$
- 所以：$h_1(n) \geq f^*(s) - g_1(n)$
- 另一方面，由于A_2扩展了n，有$f2(n) \leq f^*(s)$
- 即：$h_2(n) \leq f^*(s) - g_2(n)$ (A)
- 由于$d(n)=k$时，A_2扩展的节点A_1一定扩展，有
  $g_1(n) \leq g_2(n)$ （因为A_2的路A_1均走到了）
- 所以：$h_1(n) \geq f^*(s) - g_1(n) \geq f^*(s) - g_2(n)$ (B)
- 比较A、B两式，有 $h_1(n) \geq h_2(n)$，与定理条件矛盾。故定理得证。

对h的评价方法

- 平均分叉树
  设共扩展了d层节点，共搜索了N个节点，则:

  其中，b*称为平均分叉树。

  $$N = \left(1 - b^{*(d+1)}\right) / \left(1 - b^{*}\right)$$

- b*越小，说明h效果越好。
- 实验表明，b*是一个比较稳定的常数，同一问题基本不随问题规模变化。

对h的评价举例

例：8数码问题，随机产生若干初始状态。

- 使用h_1：

  d=14，N=539，　　b*=1.44；

  d=20，N=7276，　　b*=1.47；

- 使用h_2：

  d=14，N=113，　　b*=1.23；

  d=20，N=676，　　b*=1.27

A*的复杂性

- 一般来说，A*的算法复杂性是指数型的，可以证明，当且仅当以下条件成立时：

$$abs(h(n)-h^*(n)) \leq O(\log(h^*(n)))$$

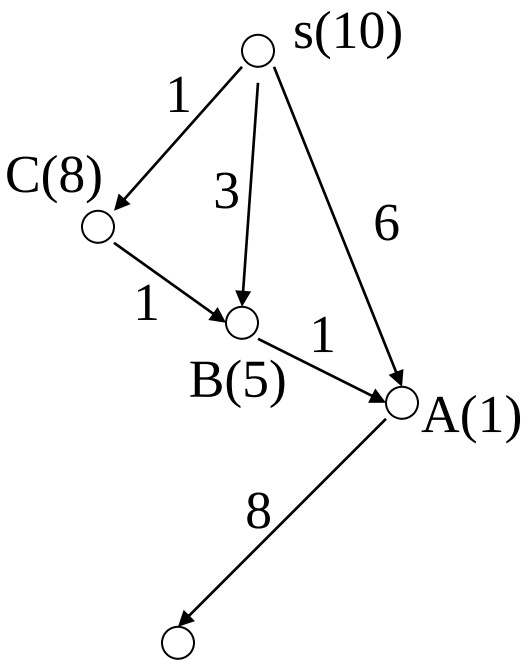
A*的算法复杂性才是非指数型的，但是通常情况下，h与h*的差别至少是和离目标的距离成正比的。

3，A*算法的改进

- 问题的提出：

  因A算法第6步对m_l类节点可能要重新放回到OPEN表中，因此可能会导致多次重复扩展同一个节点，导致搜索效率下降。

一个例子：

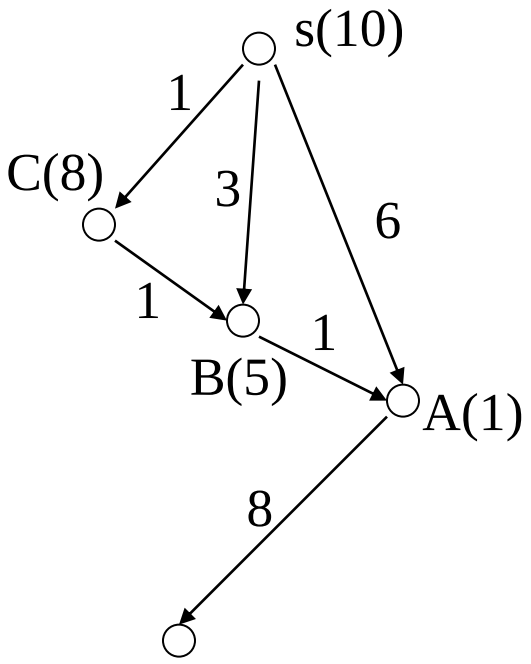


G 目标

OPEN表	CLOSED表
s(10)	s(10)
A(7) B(8) C(9)	A(7) s(10)
B(8) C(9) G(14)	B(8) s(10)
A(5) C(9) G(14)	A(5) B(8) s(10)
C(9) G(12)	C(9) A(5) s(10)
B(7) G(12)	B(7) C(9) s(10)
A(4) G(12)	A(4) B(7) C(9) s(10)
G(11)	

出现多次扩展节点的原因

- 在前面的扩展中，并没有找到从初始节点到当前节点的最短路径，如节点A。



解决的途径

- 对h加以限制
  - 能否对h增加适当的限制，使得第一次扩展一个节点时，就找到了从s到该节点的最短路径。
- 对算法加以改进
  - 能否对算法加以改进，避免或减少节点的多次扩展。

改进的条件

- 可采纳性不变
- 不多扩展节点
- 不增加算法的复杂性

对h加以限制

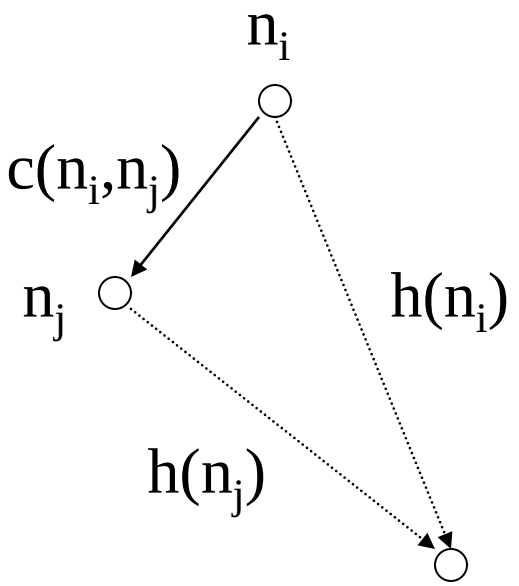
- 定义：一个启发函数h，如果对所有节点n_i和n_j，其中n_j是n_i的子节点，满足

$$\begin{cases} h(n_i) - h(n_j) \le c(n_i, n_j) \\ h(t) = 0 \end{cases}$$

或

$$\begin{cases} h(n_i) \le c(n_i, n_j) + h(n_j) \\ h(t) = 0 \end{cases}$$

则称h是单调的。



h单调的性质

- 定理5：

  若h(n)是单调的，则A*扩展了节点n之后，就已经找到了到达节点n的最佳路径。

  即：当A*选n扩展时，有g(n)=g*(n)。

定理5的证明

- 设n是A*扩展的任一节点。当n＝s时，定理显然成立。下面考察n ≠s的情况。
- 设P＝(n_0=s, n_1, n_2, …, n_k=n)是s到n的最佳路径
- P中一定有节点在CLOSED中，设P中最后一个出现在CLOSED中的节点为n_j，则n_{j+1}在OPEN中。

定理5的证明（续1）

- 由单调限制条件，对P中任意节点n_i有：

$$h(n_i) \le C(n_i, n_{i+1}) + h(n_{i+1})$$

$$g^*(n_i) + h(n_i) \le g^*(n_i) + C(n_i, n_{i+1}) + h(n_{i+1})$$

- 由于n_i、n_{i+1}在最佳路径上，所以：

$$g^*(n_{i+1}) = g^*(n_i) + C(n_i, n_{i+1})$$

- 带入上式有：

$$g^*(n_i) + h(n_i) \le g^*(n_{i+1}) + h(n_{i+1})$$

- 从i=j到i=k-1应用上不等式，有：

$$g^*(n_{j+1}) + h(n_{j+1}) \le g^*(n_k) + h(n_k)$$

- 即：$f(n_{j+1}) \le g^*(n) + h(n)$

注意：（n_j在CLOSED中，n_{j+1}在OPEN中）

定理5的证明（续2）

- 重写上式：$f(n_{j+1}) \leq g^*(n)+h(n)$
- 另一方面，A*选n扩展，必有：

  $f(n) = g(n)+h(n) \leq f(n_{j+1})$
- 比较两式，有：

  $g(n) \leq g^*(n)$
- 但已知$g^*(n)$是最佳路径的耗散值，所以只有：$g(n) = g^*(n)$。得证。

h单调的性质（续）

- 定理6：

  若h(n)是单调的，则由A*所扩展的节点序列其f值是非递减的。即$f(n_i) \leq f(n_j)$。

定理6的证明

- 由单调限制条件，有：

$$h(n_i) - h(n_j) \le C(n_i, n_j)$$

$= f(n_i)-g(n_i)$　　$= f(n_j)-g(n_j)$

$$f(n_i)-g(n_i) - f(n_j)+g(n_j) \le C(n_i, n_j)$$

$= g(n_i)+C(n_i, n_j)$

$$f(n_i)-g(n_i) - f(n_j)+ g(n_i)+C(n_i, n_j) \le C(n_i, n_j)$$

$f(n_i) - f(n_j) \le 0$，得证。

h单调的例子

- 8数码问题：
  - h为“不在位”的将牌数

$$h(n_i) - h(n_j) = \begin{cases} 1 \\ 0 \\ -1 \end{cases} \quad (n_j\text{为}n_i\text{的后继节点})$$

$h(t) = 0$

$c(n_i, n_j) = 1$

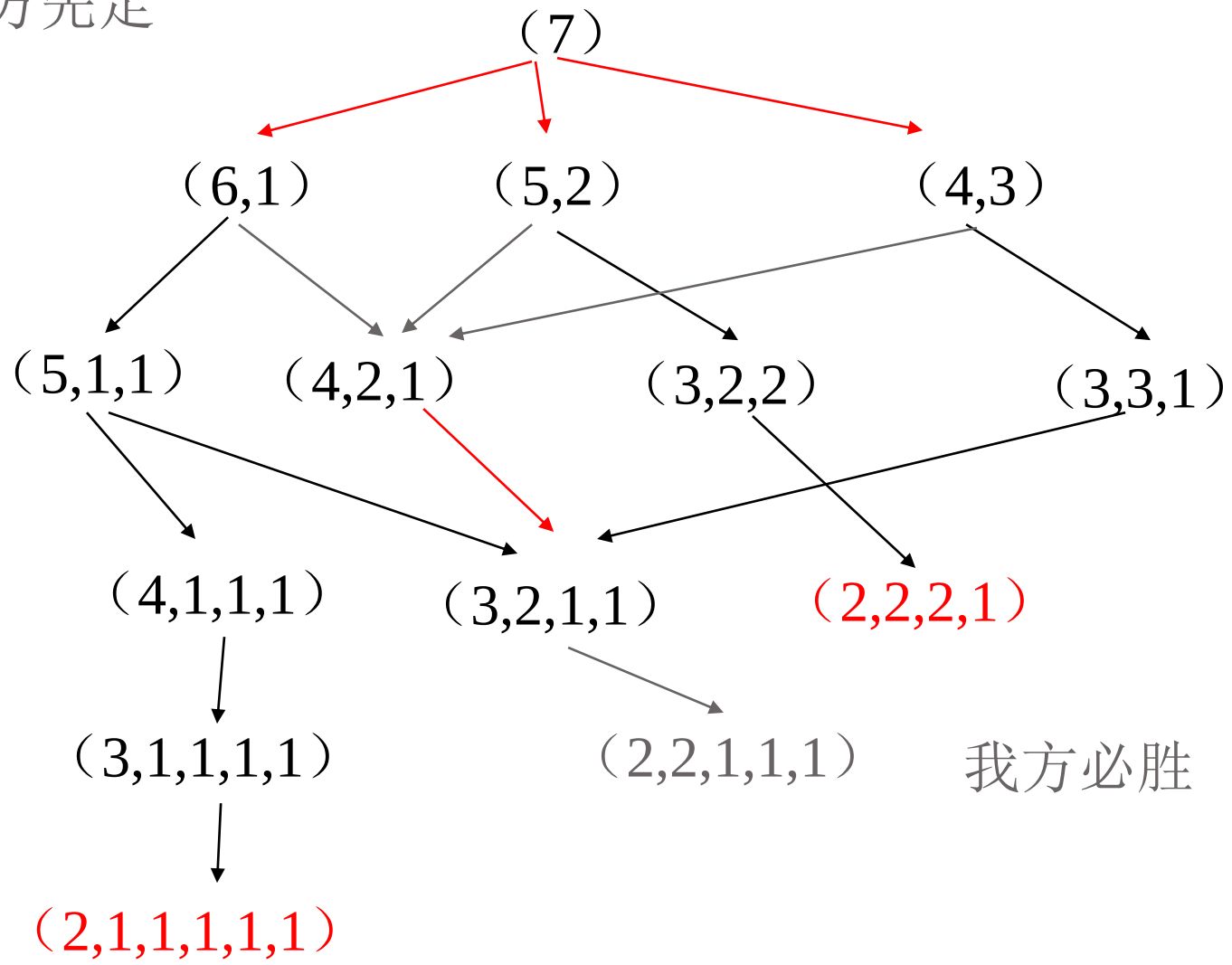
满足单调的条件。

3.3 博弈树搜索

- 博弈问题
  - 双人
  - 一人一步
  - 双方信息完备
  - 零和

分钱币问题

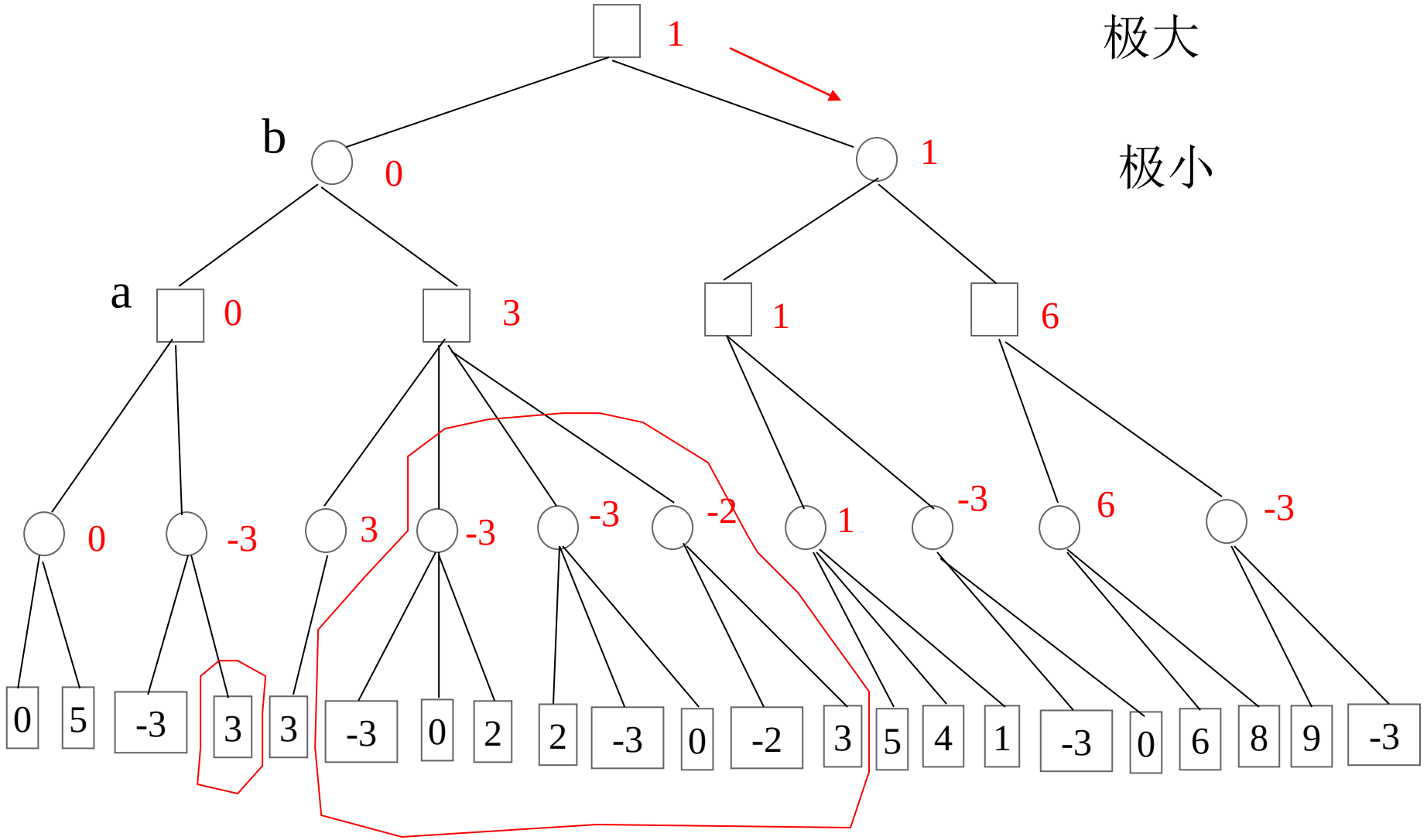
对方先走


（7）
（6,1）
（5,2）
（4,3）
（5,1,1）
（4,2,1）
（3,2,2）
（3,3,1）
（4,1,1,1）
（3,2,1,1）
（2,2,2,1）
（3,1,1,1,1）
（2,2,1,1,1）
我方必胜
（2,1,1,1,1,1）

中国象棋

- 一盘棋平均走50步，总状态数约为10的161次方。
- 假设1毫微秒走一步，约需10的145次方年。
- 结论：不可能穷举。

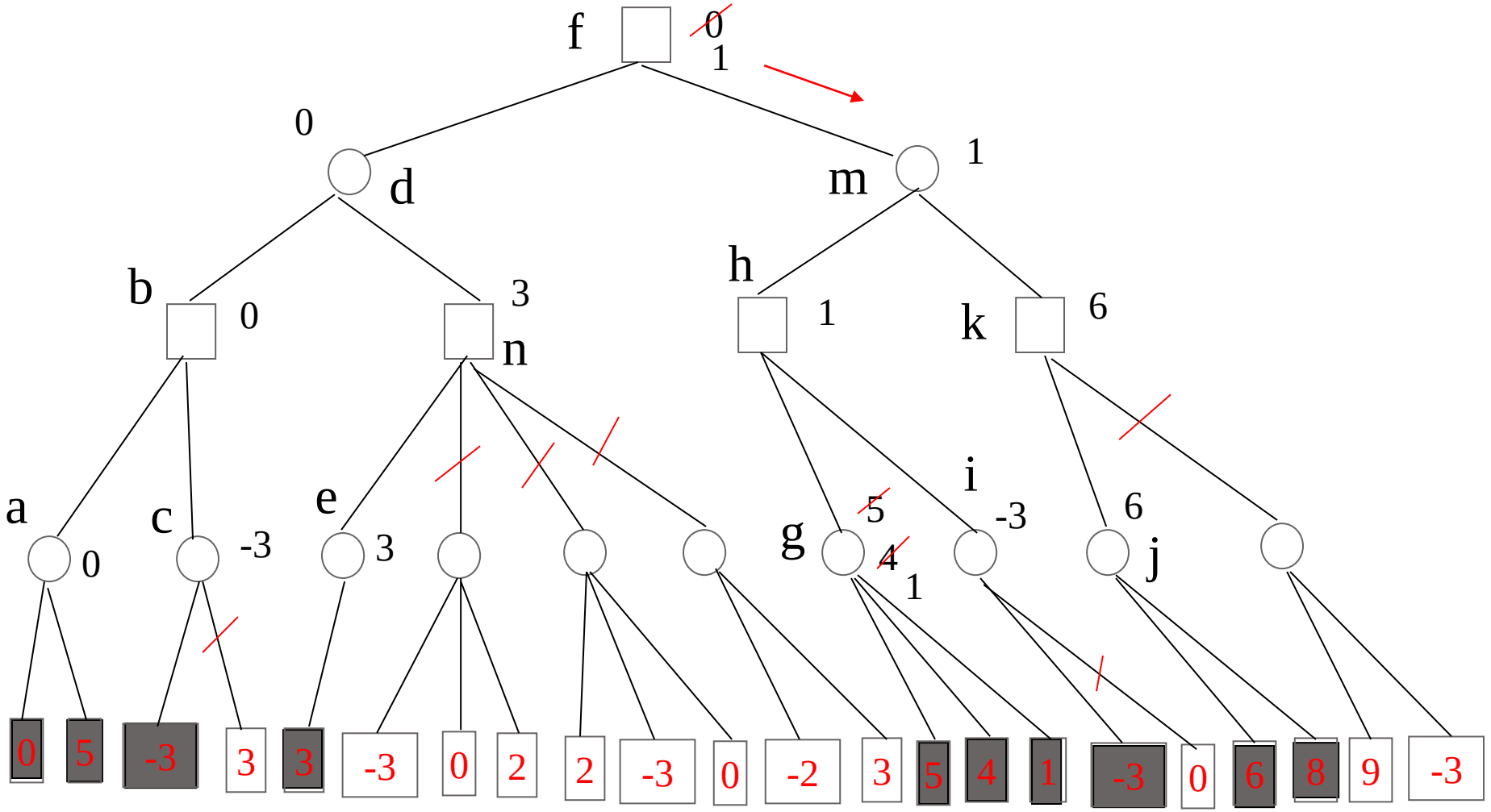
1，极小极大过程



α-β剪枝

- 极大节点的下界为α。
- 极小节点的上界为β。
- 剪枝的条件：
  - 后辈节点的β值≤祖先节点的α值时， α剪枝
  - 后辈节点的α 值≥祖先节点的β值时， β剪枝
- 简记为：
  - 极小≤极大，剪枝
  - 极大≥极小，剪枝

α-β剪枝（续）


f
0
1
0
d
1
m
b
0
3
n
h
1
k
6
a
0
c
-3
e
3
g
5
4
1
i
-3
6
j
0
5
-3
3
3
-3
0
2
2
-3
0
-2
3
5
4
1
-3
0
6
8
9
-3

- 博弈系统，必然要用到α-β搜索
- 包括Alpha Go系统也要用到

- 剪枝是必须要用到的，因为不影响正确性
- 看上去很啰嗦，其实没有啥

- 一个搜索10层左右的下棋程序就有很强的能力了

- 搜索很难不遇到指数复杂性问题
- 剪枝、搜索策略很难改变算法的复杂性

- 怎么样才能减小搜索的量？
- 宽度、深度两个方面

- 宽度：从根下面的一层，就去掉大部分的分支，找最有希望的。时间不允许的就不去计算了
- 深度：达到一定的深度就不去计算了，而是给个估算。Alpha Go给了一个蒙托卡罗估算

- 搜索是人工智能必须掌握的内容，基本功
- 搜索法让人感觉出了点人工智能的味道

- 搜索方法与人处理问题的方法不太一致
  - 用麻烦（牺牲计算时间）换取一般解法
  - 人是用发现一些规律的方法
    - 围棋的口诀（如：立二拆三）、细菌培养、西医治病
  - 目的是一致的
- 是现代传统科学思维方式的代表

其他的启示

- 人的一生，就是一个启发式搜索的过程
- 每个人的目标是不一致的
- 要达到你的目标，需要制定一个启发式函数
- 你的启发式函数越精确，越容易成功
- 有的人偏宽度优先，喜欢各种不同方向的尝试
- 有的人偏深度优先，找到一条路就走下去

- 都对，都不完善

- 做一件事、一个公司的发展也都是一个搜索的过程

- 比如，改造一个程序
- 比如，公司的多元化经营、还是专业化发展
- 比如，决策的民主化，还是集思广益

- 我们的人工智能导论课，哪怕你只学会了搜索，也是很有用的
- 所以，认真完成搜索的实验
- 真正的体会一下，不同的搜索策略，会产生什么影响