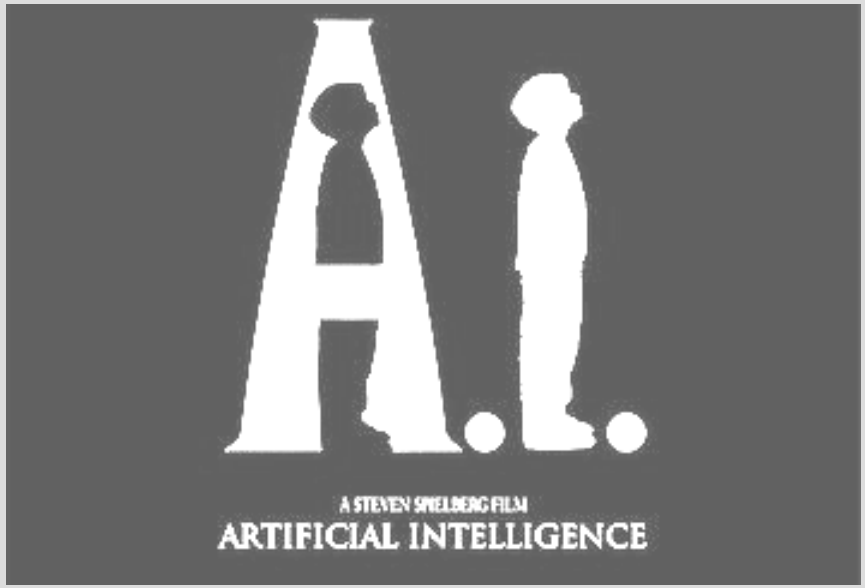
人工智能（Artificial Intelligence）



福州大学数学与计算机学院

陈昭炯

2010年3月10日星期三

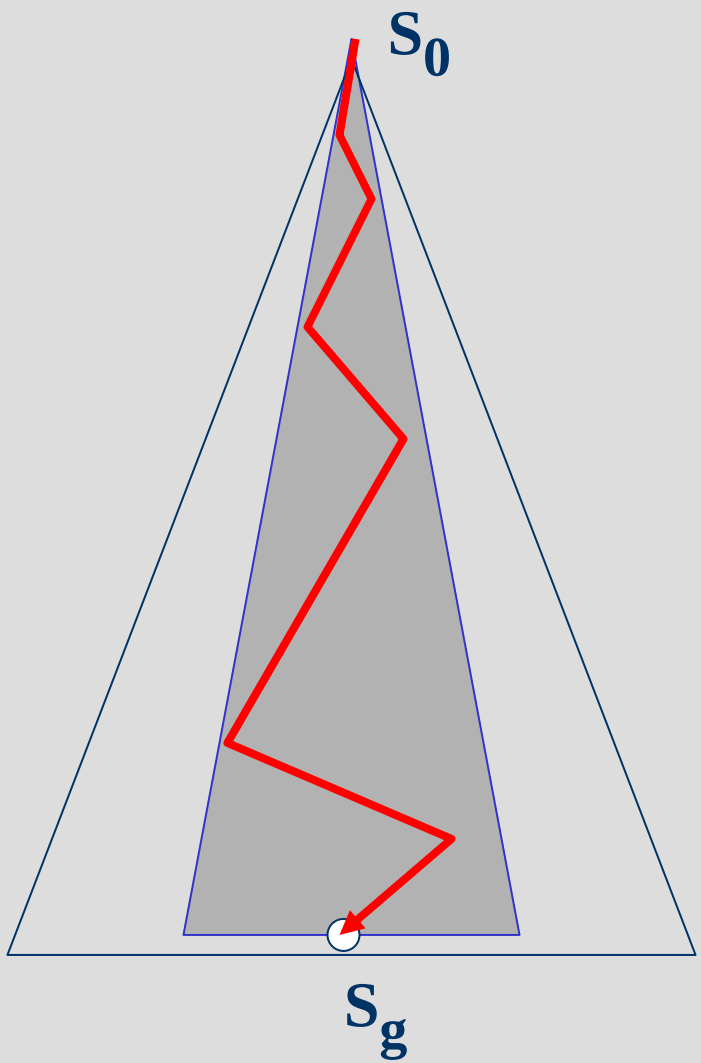
第一章 搜索问题

- 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）。

搜索问题（续1）



基本概念

搜索的含义

- 依问题的实际情况寻找可利用的知识，构造代价较少的推理路径从而解决问题的过程
- 离散的问题通常没有统一的求解方法
- 搜索策略的优劣涉及能否找到最好的解、计算时间、存储空间等
- 搜索分为盲目搜索和启发式搜索
- 盲目搜索：按预定的策略进行搜索，未用问题相关的或中间信息改进搜索。效率不高，难求解复杂问题，但不失可用性
- 启发式搜索：搜索中加入问题相关的信息加速问题求解，效率较高，但启发式函数不易构造

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?(完备性)
- 找到的解是最佳的吗？(最优性) — 什么情况下可以找到最佳解？
- 求解的效率如何?（时间、空间复杂度）

状态空间表示法

状态：描述问题求解中任一时刻的状况；变量的有序组合

- 算符：一个状态→另一状态的操作
- 状态空间：所有求解路径构成的图；{状态，算符} 表示

问题求解过程：

- 初始状态：描述问题求解中的初始状况
- 算符：一个状态→另一状态的操作
- 目标测试：确定给定的状态是否为目标状态
- 路径耗散函数：设定每一步算符操作的耗散值

问题的解：从初始状态到目标状态的路径

最优解：所有解中耗散值最小的解

例：二阶梵塔问题

状态描述：（S_A,S_B）

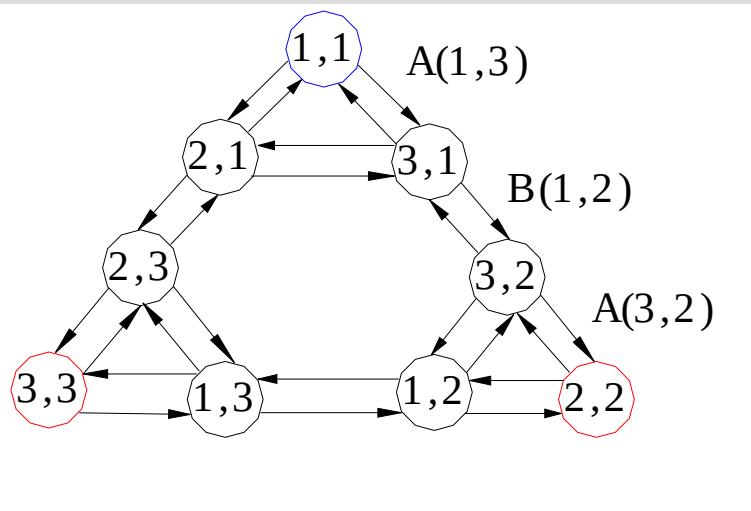
可能状态：S_0=(1,1),S=(1,2),S=(1,3),S=(2,1),S_g=(2,2),S=(2,3)

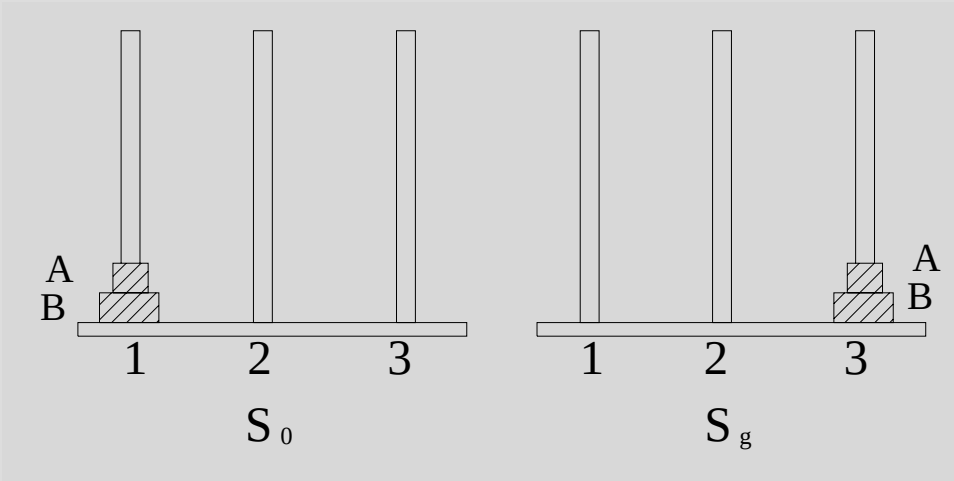
S=(3,1),S=(3,2),S_g=(3,3)

算符：*A(i,j)*—将*A*从*i*轴移至*j*轴；　*B(i,j)*—将*B*从*i*轴移至*j*轴

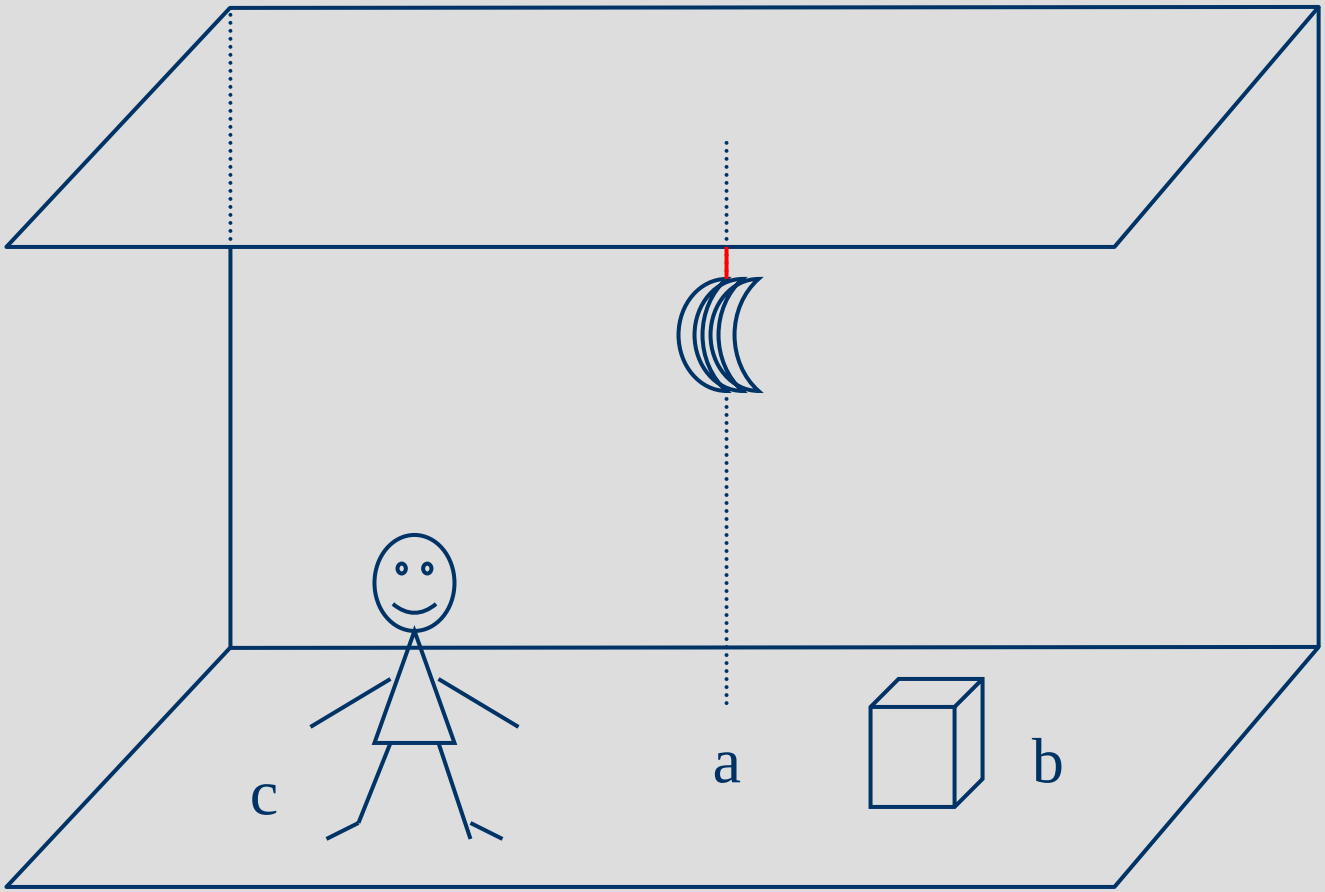
可能算符：A(1,2), A(1,3), A(2,1), A(2,3), A(3,1), A(3,2)

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





猴子摘香蕉问题



《人工智能》陈昭炯

状态：(w,t,x,y,z)		初始状态：(c,a,b,0,0)
w:猴子的水平位置	{a,b,c}	目标状态：(a,a,a,1,1)
t:天花板上香蕉对应的地面位置	{a,b,c}	可能状态：(b,a,b,0,0)
x:箱子的水平位置	{a,b,c}	(c,a,c,1,0)
y:猴子是否在箱子上	{0,1}	(c,a,c,1,1)
z:猴子是否拿到香蕉	{0,1}	……

操作符：

Goto(u):猴子走到u处　(w,t,x,y,0) → (u,t,x,y,0)

Push(v):猴子推箱到v处　(w,t,w,0,0) →(v,t,v,0,0)

Climb: 猴子爬上箱子　(w,t,w,0,0) →(w,t,w,1,0)

Grasp: 猴子拿到香蕉　(a,a,a,1,0) →(a,a,a,1,1)

《人工智能》陈昭炯

例：修道士与野人问题(1968)

S_0：河左岸有*M*个*Missionaries*和*k*个*Cannibals*,1条*boat*

条件：1）M和C都会划船，船一次只能载*k*人

2）在任一岸上，M人数不得少于C的人数，否则被吃

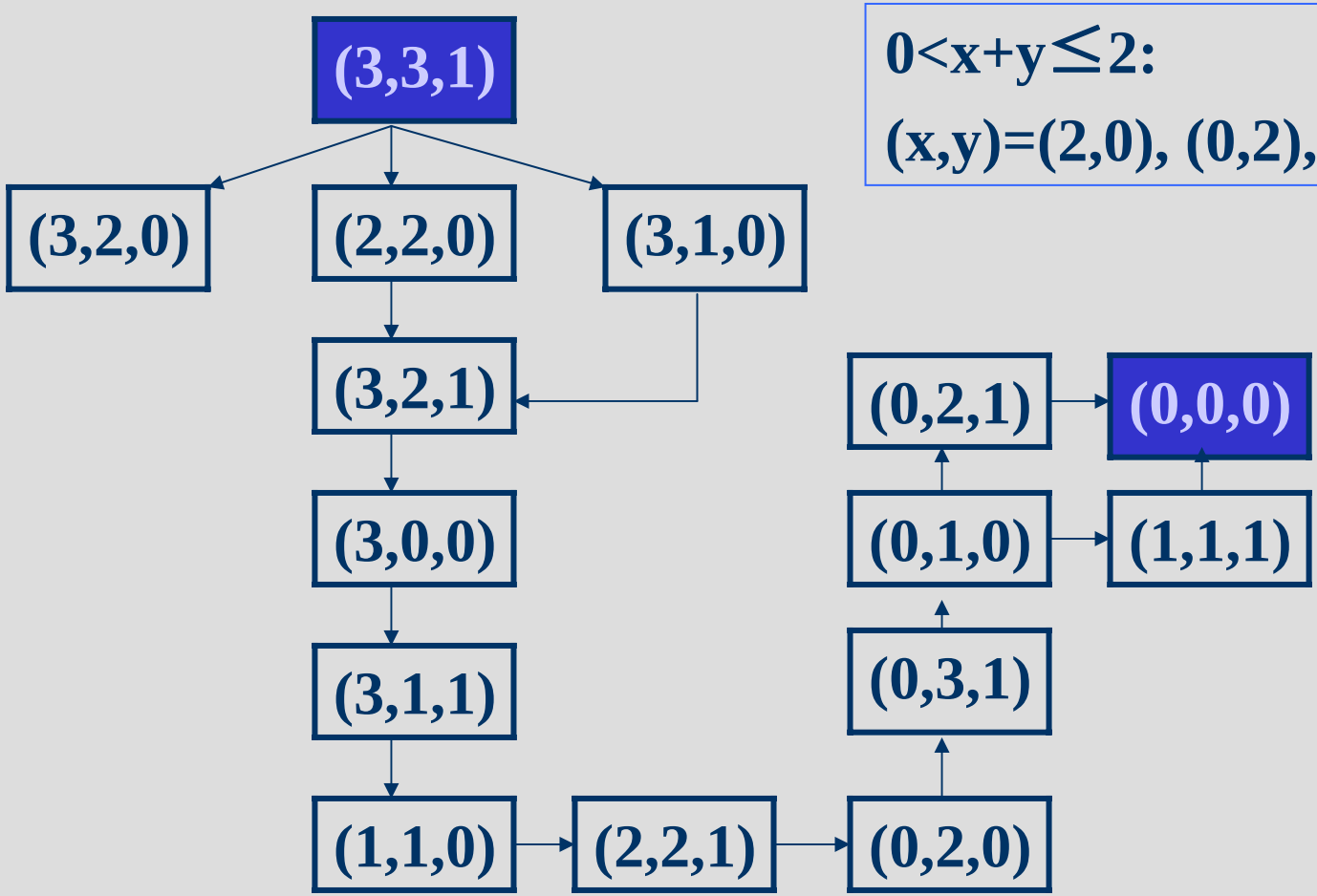
目标：安全抵达对岸的最佳方案

左岸:(m,c,b);$0\le m\le N, 0\le c\le k, b\in\{0,1\}$;$S_0$:(N,k,1);$S_g$(0,0,0)

状态总数：(N+1)×(k+1)×2;N=K=3时，32种，但合法状态16种

$$\begin{cases} m\ge c \wedge m\ge 1 \\ 3-m\ge 3-c \wedge 3-m\ge 1 \end{cases} \Rightarrow m=c=1,2 \Rightarrow \begin{cases} m=0 \\ m=3 \\ m=c=1,2 \end{cases}$$

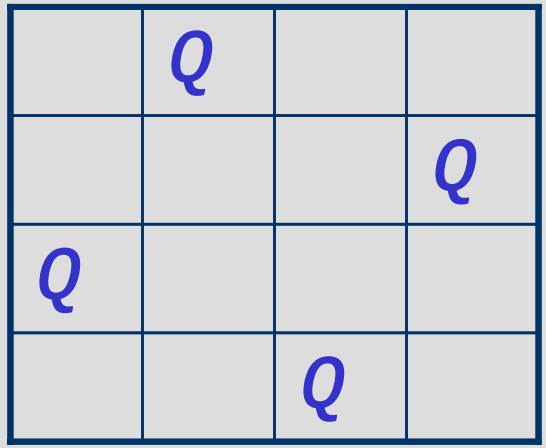
$$b=1:\begin{cases} m\Leftarrow m-x \\ c\Leftarrow c-y \\ b=0 \end{cases}, \quad \begin{cases} x\le m \\ y\le c \\ x+y\le 2 \end{cases} \qquad b=0:\begin{cases} m\Leftarrow m+x \\ c\Leftarrow c+y \\ b=1 \end{cases}, \quad \begin{cases} x\le 3-m \\ y\le 3-c \\ x+y\le 2 \end{cases}$$



$0<x+y\leq 2$:

(x,y)=(2,0), (0,2),(1,1),(1,0),(0,1)

例：皇后问题（1850）



Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal

初始状态：棋盘上无皇后

算符：将皇后添加到棋盘上的任一空格

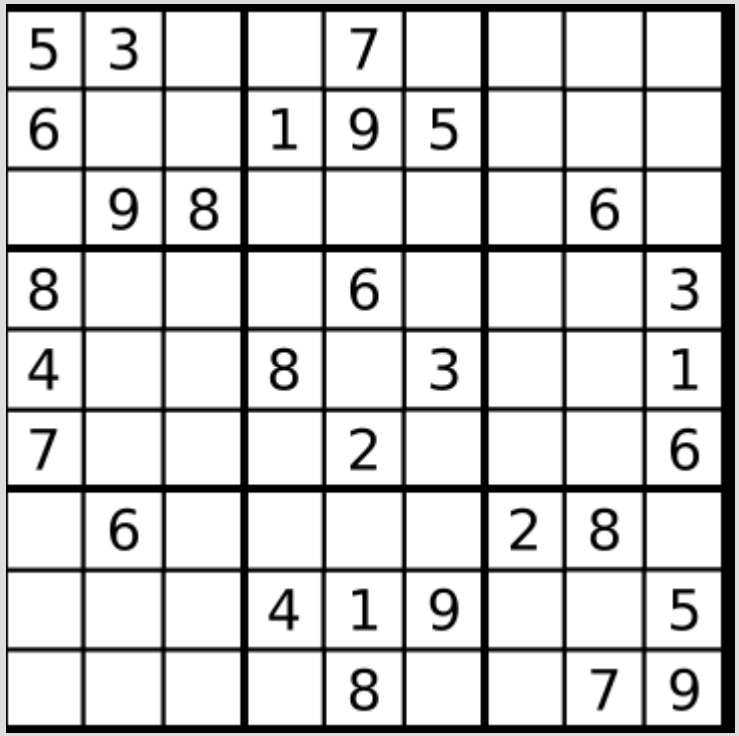
目标测试：8皇后都在棋盘上，且互相攻击不到

路径耗散函数：每一步耗散值1

返回

- 8皇后：92种解，本质解12个
- 找到一般n皇后问题复杂度为O(n)的算法(1989)

例：数独，Latin square



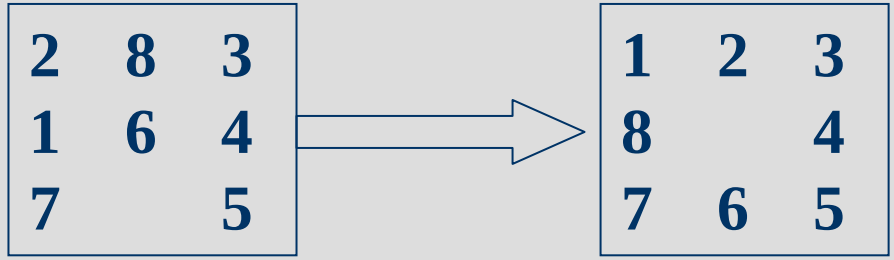
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid so that each column, each row, and each of the nine 3×3 boxes (also called blocks or regions) contains the digits from 1 to 9 only one time each. The puzzle setter provides a partially completed grid.

例：八数码游戏

（九宫重排问题）



8-Puzzle

sliding-block puzzle

初始状态：任一状态都可为初始状态

算符：将空位移向四个方向

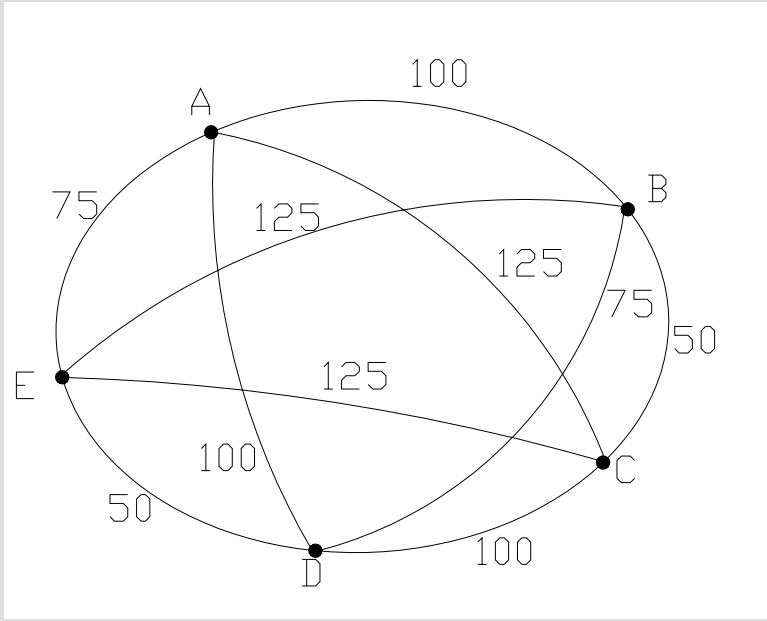
目标测试：确定当前状态是否为目标状态

路径耗散函数：每一步耗散值1

- 其状态可以划分为两个不相交的集合。(证明)
- 8数码， 9！/2＝181440 ；15数码，1.3万亿；24数码,10^{25}
- 一般n×n数码是NP完全问题（1986）

例：TSP问题

Traveling Salesman Problem



从某城市出发遍历所有n个城市一遍且仅一遍再回到出发地，求最短路径

初始状态：在某一城市

算符：移向下一个未访问过的城市

目标测试：是否处于出发地且访问过所有城市一次

路径耗散函数：路程长度，旅行费用等

- No general method of solution is known
- NP－hard(Karp,1972)
- 有效的启发式算法（1973）
- 完全多项式近似方案（1998）
- ……

若干应用实例

•寻径问题：计算机网络的路由，军事行动的规划，飞机航线旅行规划系统，机器人导航（寻径问题拓广）

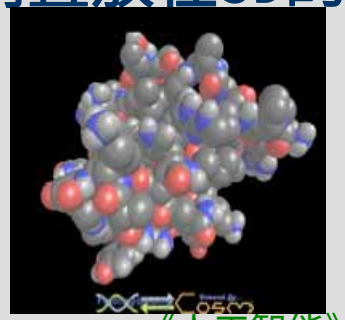
•旅行问题：电路板自动钻孔机的运动规划，仓库货物码放机的运动规划

•超大规模集成电路的布局：在一个芯片上放置上百万个元器件及连线，还要达到芯片面积最小、电路延迟最小、杂散电容最小和产量最大。单元布局和通道寻径（1991）

•自动装配排序：找到一个装配物体（电动机）各部件的次序

•蛋白质设计：寻找一个氨基酸序列，当该序列叠放在3D的蛋白质结构里，可治愈某种疾病

•因特网搜索：寻找问题的答案、相关信息等



1.2 图搜索策略

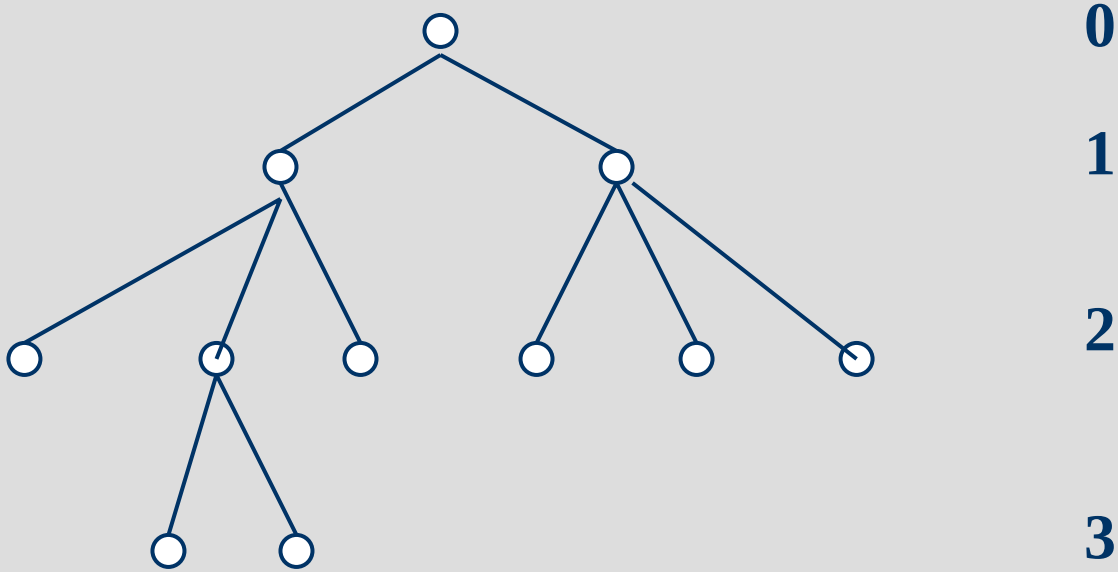
- 问题的引出
  - 回溯搜索：只保留从初始状态到当前状态的一条路径。
  - 图搜索：保留所有已经搜索过的路径。

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



- 路径的代价（耗散值）

一条路径的代价（耗散值）等于连接这条路径各节点间所有代价（耗散值）的总和。用$C(x_i, x_j)$表示从父节点x_i到子节点x_j的边代价（耗散值）。

- 代价树：边上标有代价的树状结构图
- 若干记号：

S_0－初始态；S_g－目标态；

$g(x)$－从S_0到节点x的代价；$g(x_j)=g(x_i)+C(x_i, x_j)$

$h(x)$－x到S_g最优路径的估计代价

状态空间的搜索策略

一般图的搜索过程

*OPEN*表：存放刚生成的节点

状态节点	父节点

*CLOSED*表：存放当前将要扩展和前面已扩展的节点

编号	状态节点	父节点

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。

1）$S_0 \rightarrow OPEN$，$S_0 \rightarrow G_0$，

2）$OPEN=Nil$→无解；否则

3）$OPEN$的第一个节点→$CLOSED$，记为n

4）节点n =目标→得解；否则

- m_j:扩展的新节点
- m_k:已生成的节点
- m_1:已生成并被扩展的节点

5）扩展节点n, M={n扩展出的子节点－n 的先辈}；$G_{n-1} \cup M \rightarrow G_n$

6)处理M，$\forall x \in M$,考虑

① m_j :if $x \notin G_{n-1}$,$x \rightarrow OPEN$;

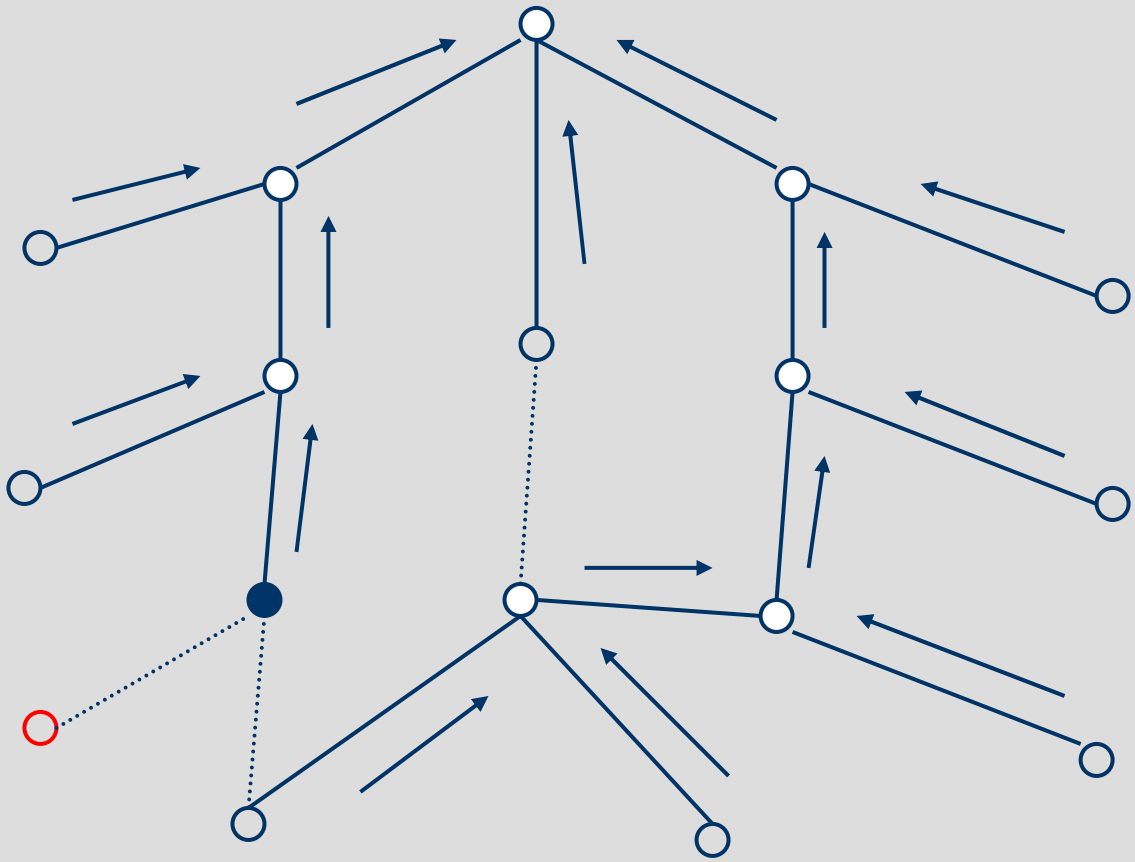
②m_k: if $x \in G_{n-1}$(已生成过),判断x的父节点是否需改变（依代价）

③$m1$: if $x \in G_{n-1}$ AND $x \in CLOSED$(已扩展过)，判断x的原后继节点的父指针是否需改变

7）按某种搜索策略对$OPEN$表排序

①队列方式($FIFO$)：广度优先； ②堆栈方式（$LIFO$）：深度优先; ③其它

8）转2）



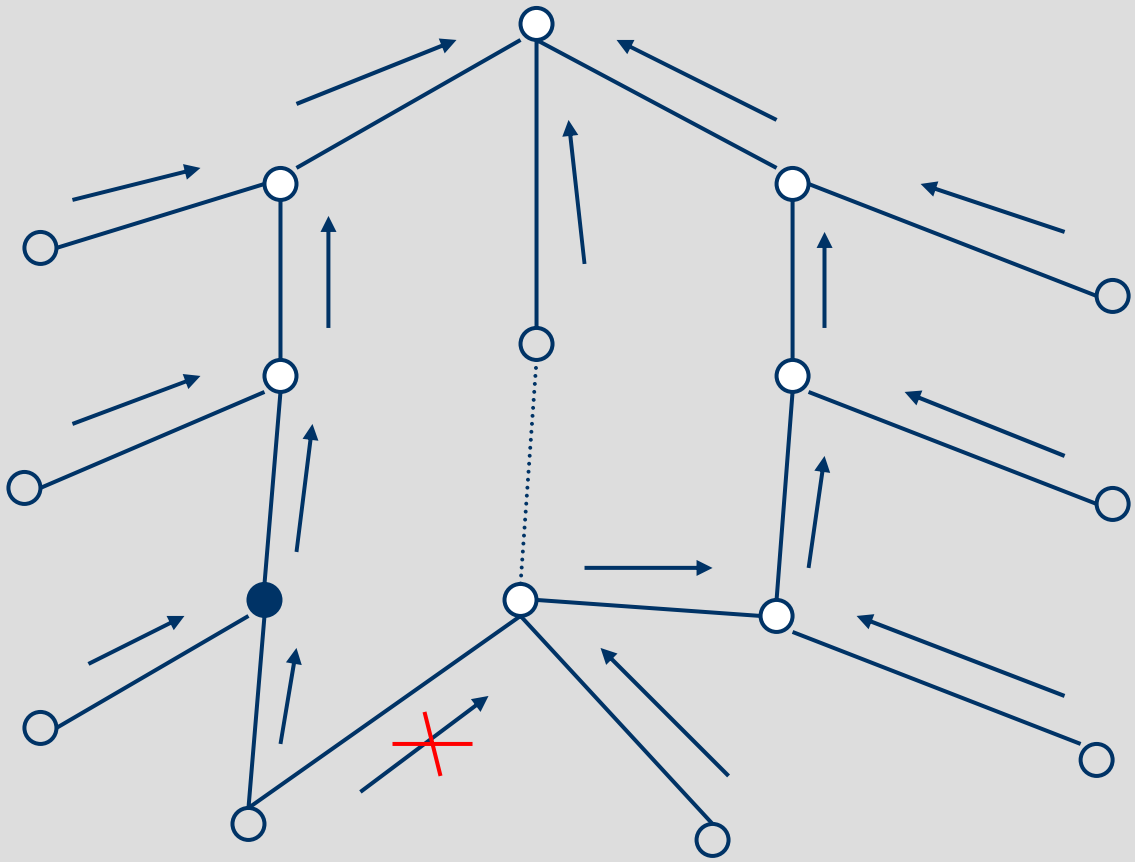
○ 已扩展 ($\in CLOSED$)

● 选中当前正扩展

○已生成，未扩展($\in OPEN$)

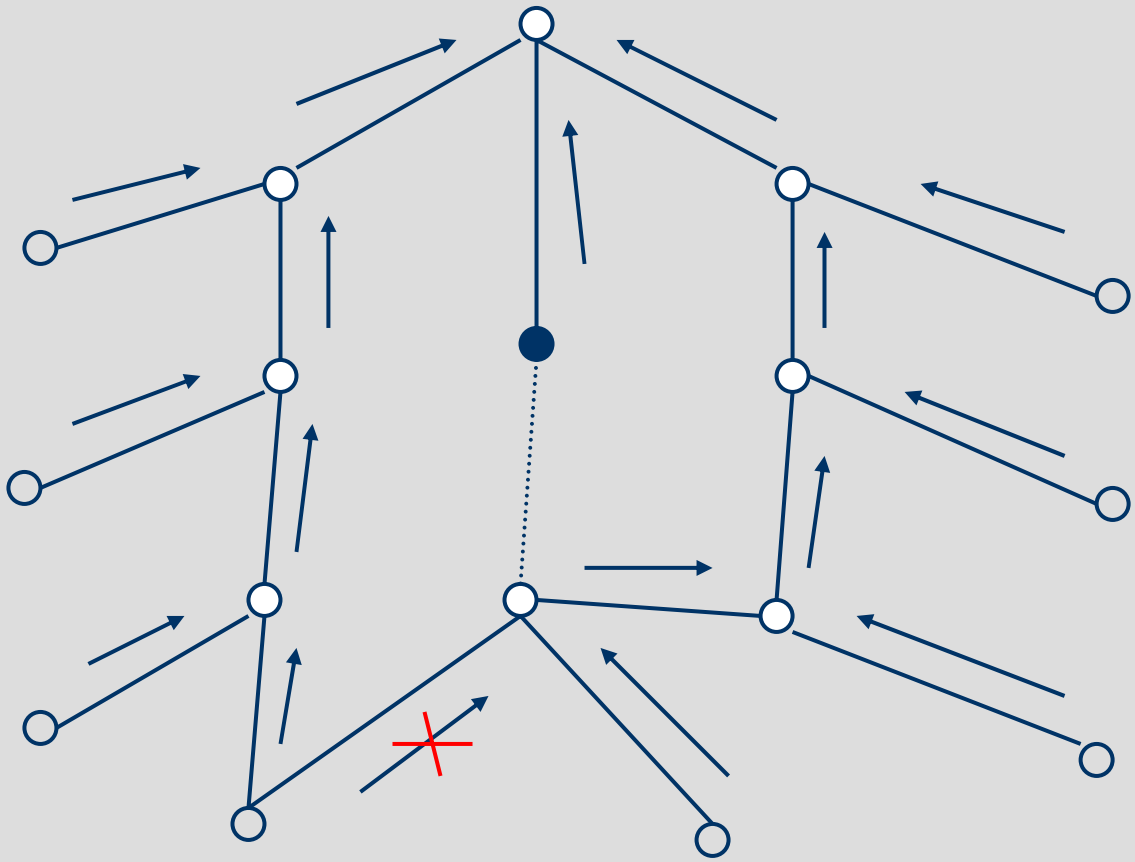
○ 未生成过

《人工智能》陈昭炯



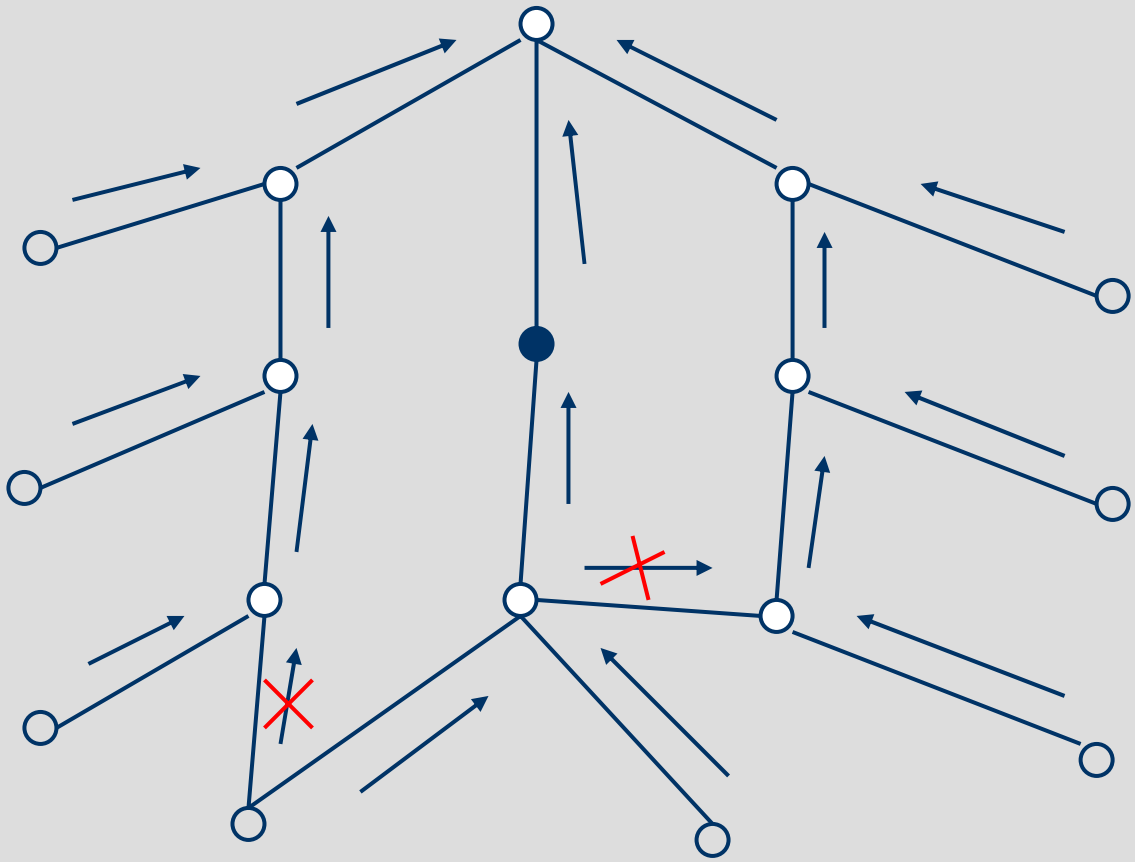
○ 已扩展 ($\in CLOSED$)　　○已生成，未扩展($\in OPEN$)

● 选中当前正扩展



○ 已扩展 ($\in CLOSED$)　　○已生成，未扩展($\in OPEN$)

● 选中当前正扩展



○ 已扩展 ($\in CLOSED$)　　○已生成，未扩展($\in OPEN$)

● 选中当前正扩展

1）不同的搜索策略只是*OPEN*表的排序不同，过程类似

2）*G*称为搜索图，由节点及其父指针→搜索树

3）目标找到后，其路径由逐级上行的父指针构成

4）盲目搜索仅适用于树结构，不出现图搜索中6）②③

广度优先搜索

1, $G:=G_0(G_0=S_0)$, OPEN:=(S_0), 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, $G_{n-1} \cup M \rightarrow G_n$;

7, IF GOAL(at M) THEN EXIT(SUCCESS);

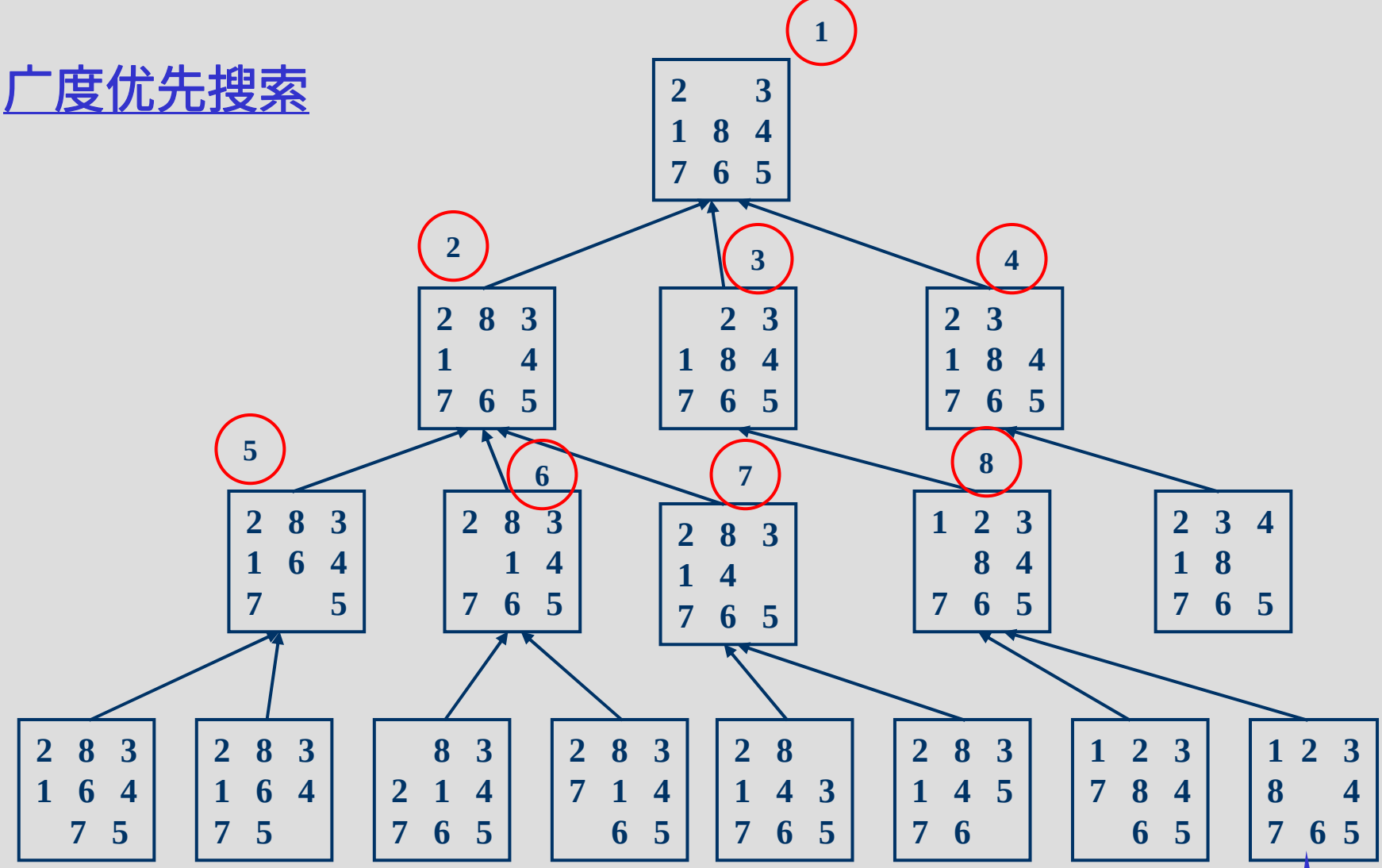
8, ADD(M,OPEN), 并标记M 中的节点到n 的指针;

9, GO LOOP;

ADD(x,y):添加x至y 的尾部

记录搜索最优解过程中的所有节点

广度优先搜索



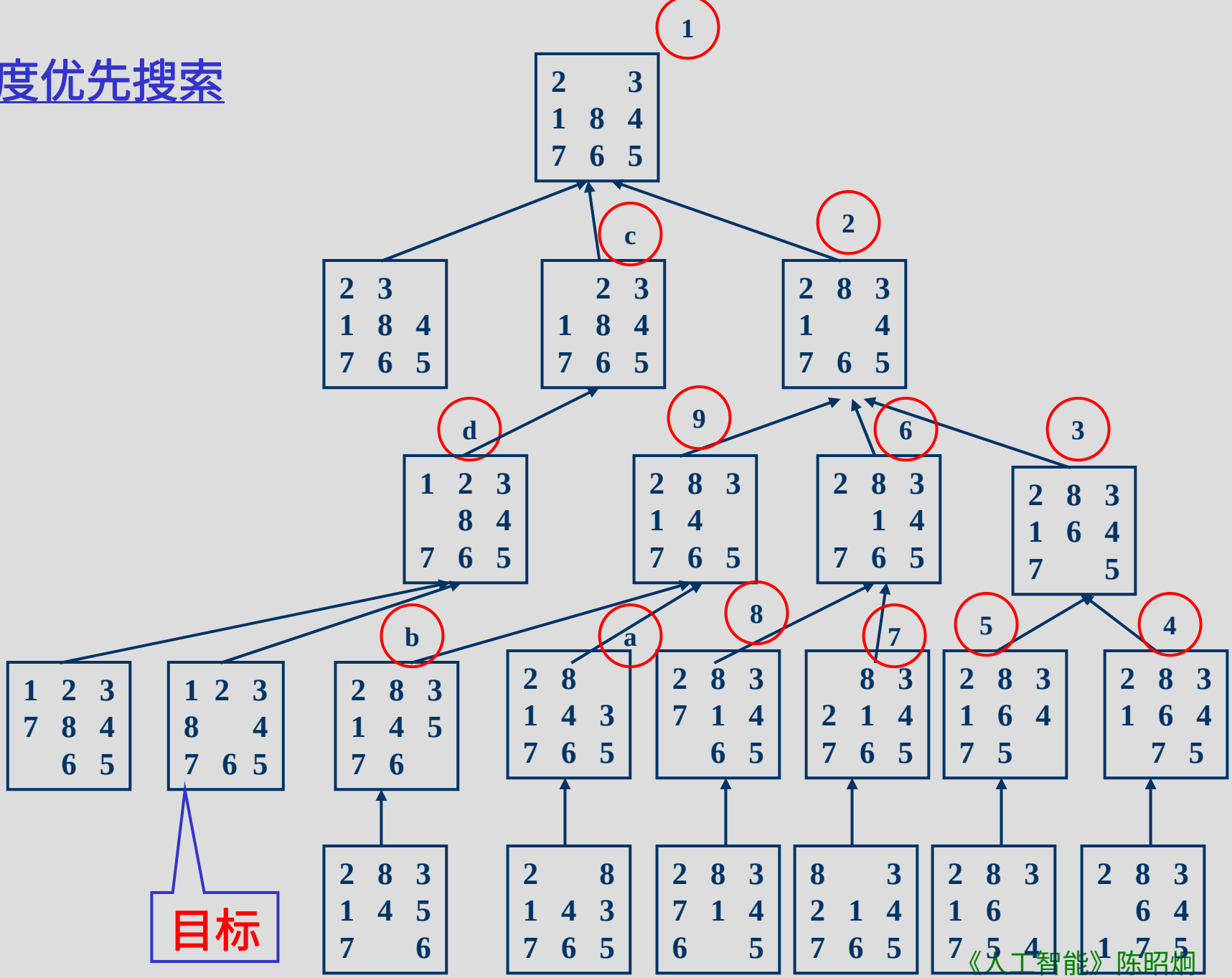
生成的新节点按队列方式压入OPEN表底部（先进先出）

深度优先搜索

1, G:=G_0(G_0=S_0), OPEN:=(S_0), 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, G_{n-1}∪M →G_n;
7, IF 目标在M中 THEN EXIT(SUCCESS);
8, ADD(OPEN,M), 并标记M中的节点到n的指针;
9, GO LOOP;

ADD(x,y):添加x至y的尾部
—节点的所有后代被完全搜索过则该节点从内存中删除

深度优先搜索


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

广度优先搜索效率分析

最坏情况已生成的节点数＝$b+b^2+b^3+\cdots+b^d+(b^{d+1}-b)=o(b^{d+1})$

•每个节点有b个后继节点；解的深度为 d；搜索树的最大深度m;

深度	节点数	时间	内存
2	1100	0.11s	1 M
4	111，100	11s	106 M
6	10^7	19m	10 G(KM)
8	10^9	31h	1 T(KG)
10	10^{11}	129d	101 T(KG)
12	10^{13}	35y	10 P(KT)
14	10^{15}	3523y	1 E(KP)

b=10

10^4 个节点 */s*

10^3 bytes/node

内存需求比时间消耗更大

广度优先难以解决大的搜索问题（时间）

深度优先搜索效率分析

存储的节点数＝$b\cdot d_m+1$；*b=10,d=12,* 存储空间＝*118K*；*O(b·d);*百亿倍

广度优先搜索的性质

- 当问题有解时，一定能找到解且为最优解(耗散值是节点深度的非递减函数)
- 是一个通用的与问题无关的方法
- 最坏情况时，搜索空间等同于穷举
- 时间、空间效率较低 $o(b^{d+1})$

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（无穷分支）
- 最坏情况时，搜索空间等同于穷举
- 时间效率较低 $o(b^{m})$
- 是一个通用的与问题无关的方法

回溯搜索

- 每次只生成一个后继节点而不是所有的后继，内存O(d)

有界深度优先搜索

- 避免搜索陷入某一无穷分支死循环,设定深度界限L;
- L的选择问题；考虑$L<d$ 和 $L>d$的情况
- $L>d$,问题有解一定可以找到解，但不一定最优
- $L>d$,时间复杂度 $o(b^l)$; 空间复杂度 $o(bl)$

迭代深入深度优先搜索

- 用于寻找最合适的深度限制的通用策略
- **For depth ←0 to m do 有界深度优先搜索**
- 问题有解一定可以找到解且最优(条件限制)
- 时间复杂度$o(b^d)$; 空间复杂度 $o(bd)$
  生成节点总次数= $db+(d-1)b^2+\cdots+b^d$
- 当搜索空间很大且解的深度未知时，首选的盲目搜索法

双向搜索

- 同时运行2个搜索,一个从初态开始一个从目标态开始,中间相遇时终止,学习并分析.

盲目搜索策略的算法评价

	广度	深度	有界深度	迭代深入	双向
完备性	是[1]	否	否	是[1]	是[1,4]
时间	$o(b^{d+1})$	$o(b^{m})$	$o(b^{l})$	$o(b^{d})$	$o(b^{d/2})$
空间	$o(b^{d+1})$	$o(bm)$	$o(bl)$	$o(bd)$	$o(b^{d/2})$
最优性	是[3]	否	否	是[3]	是[3,4]

1: 若b是有限的，则是完备的

3：若单步耗散是相同的，则是最优的

4：若每个方向都采用广度优先搜索

b: 节点的平均分支数

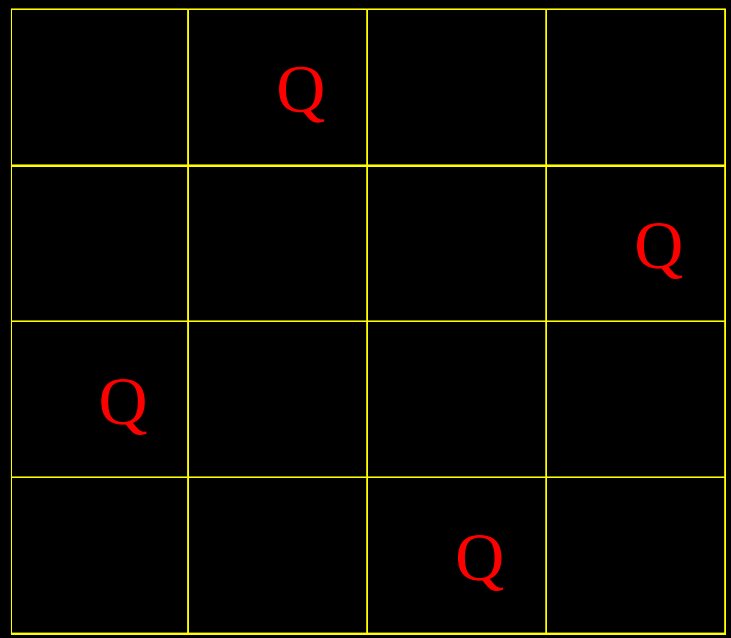
d：最浅的解的深度

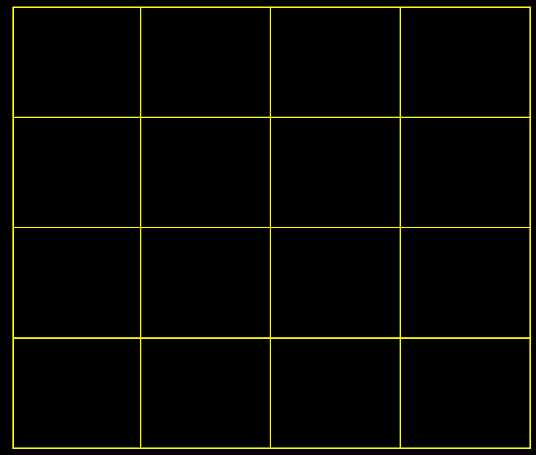
m：搜索树的最大深度

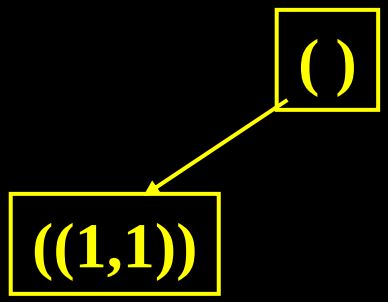
l：深度限制

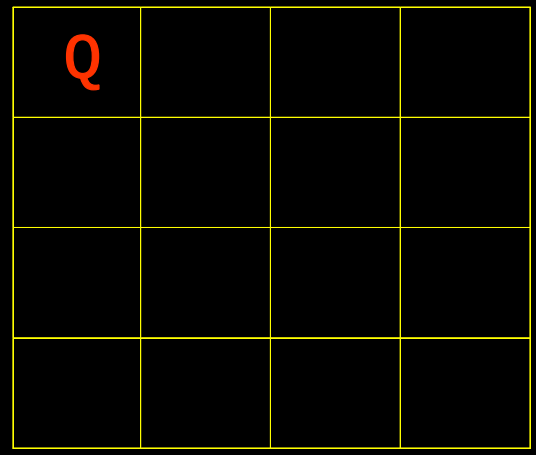
1.1 回溯策略

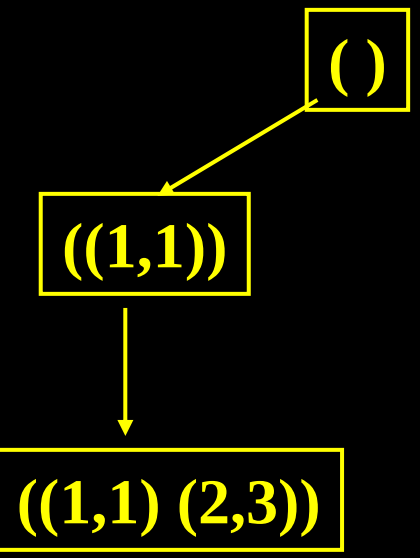
- 例：皇后问题



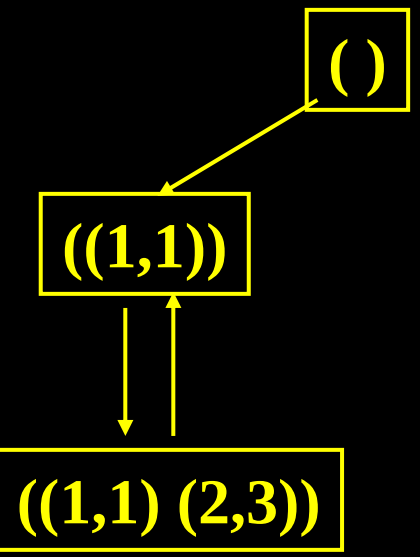



( )
((1,1))

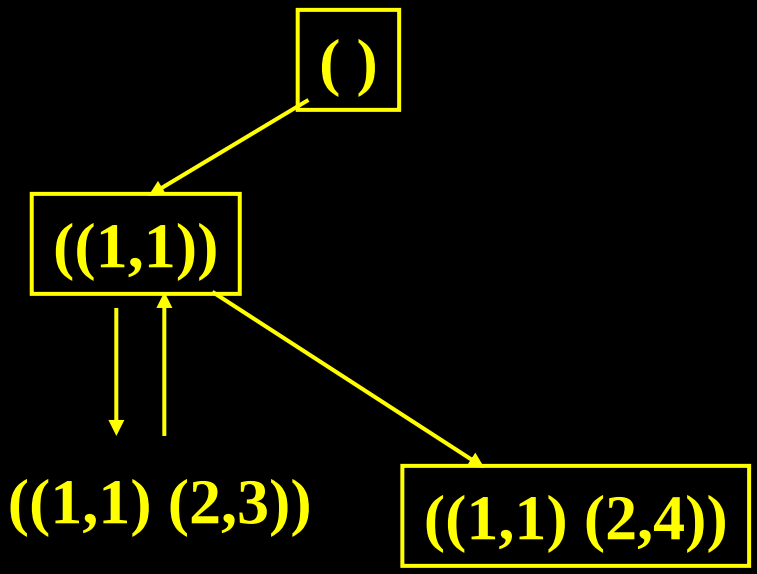

Q



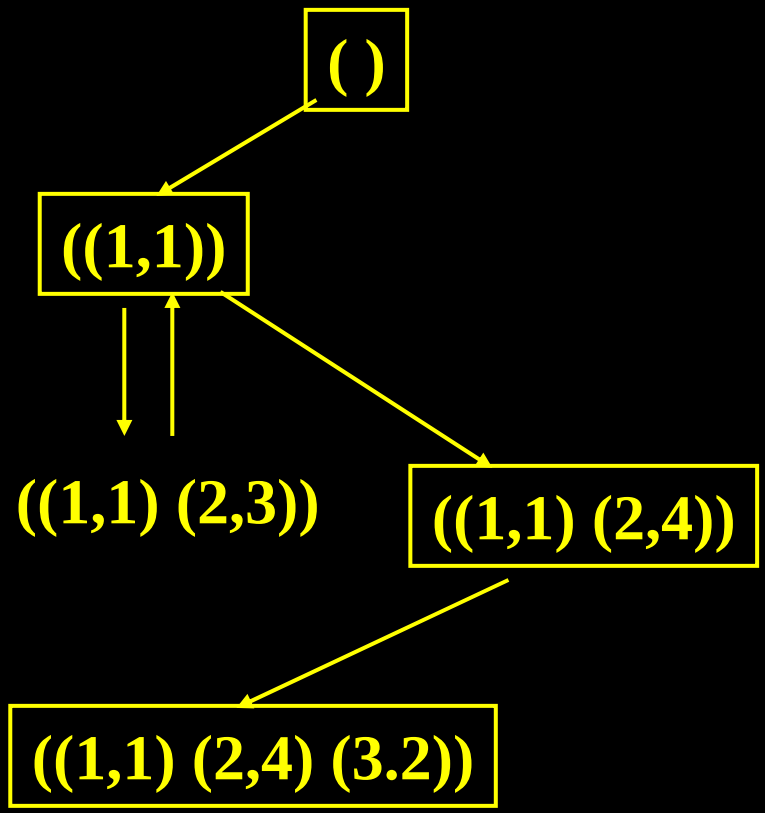
Q			
		Q	



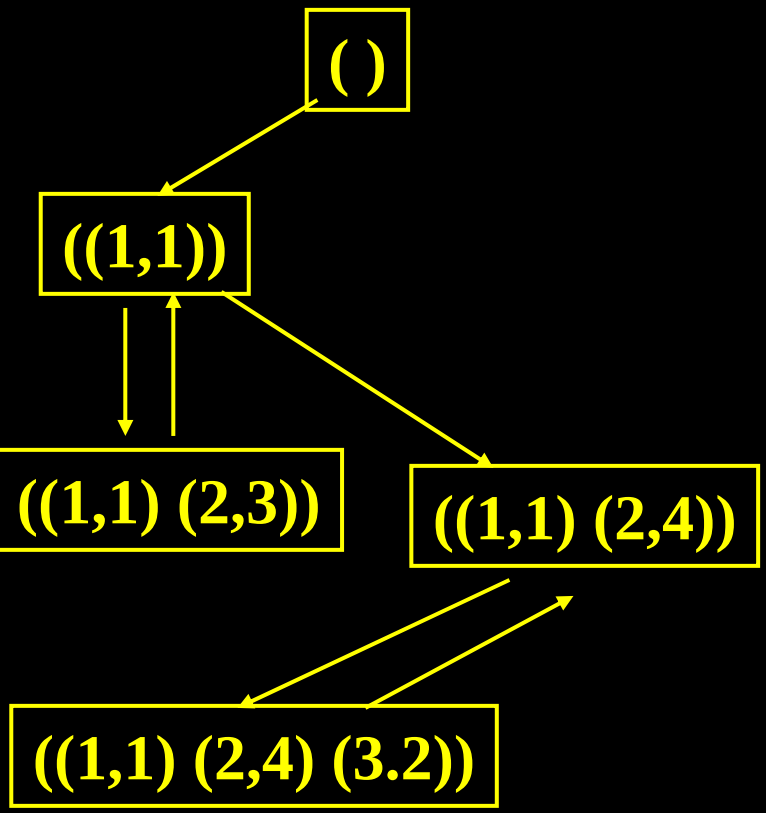
Q			



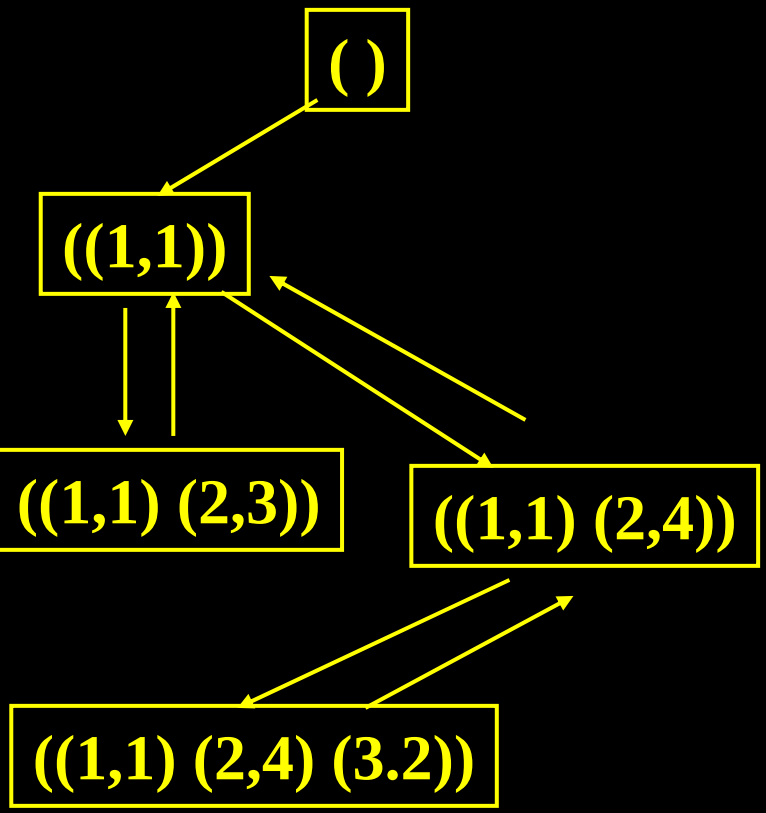
Q			
			Q



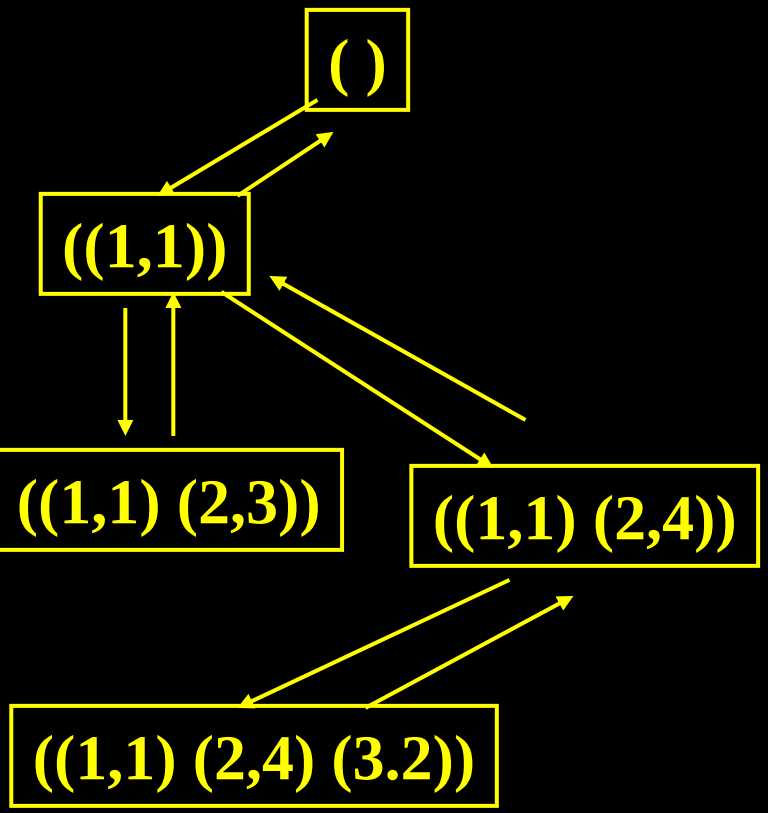
Q			
			Q
	Q		

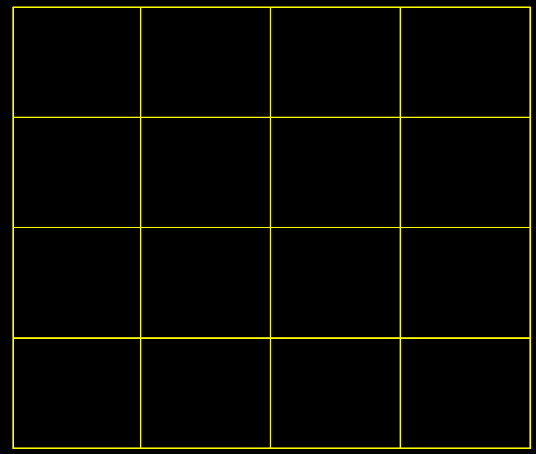


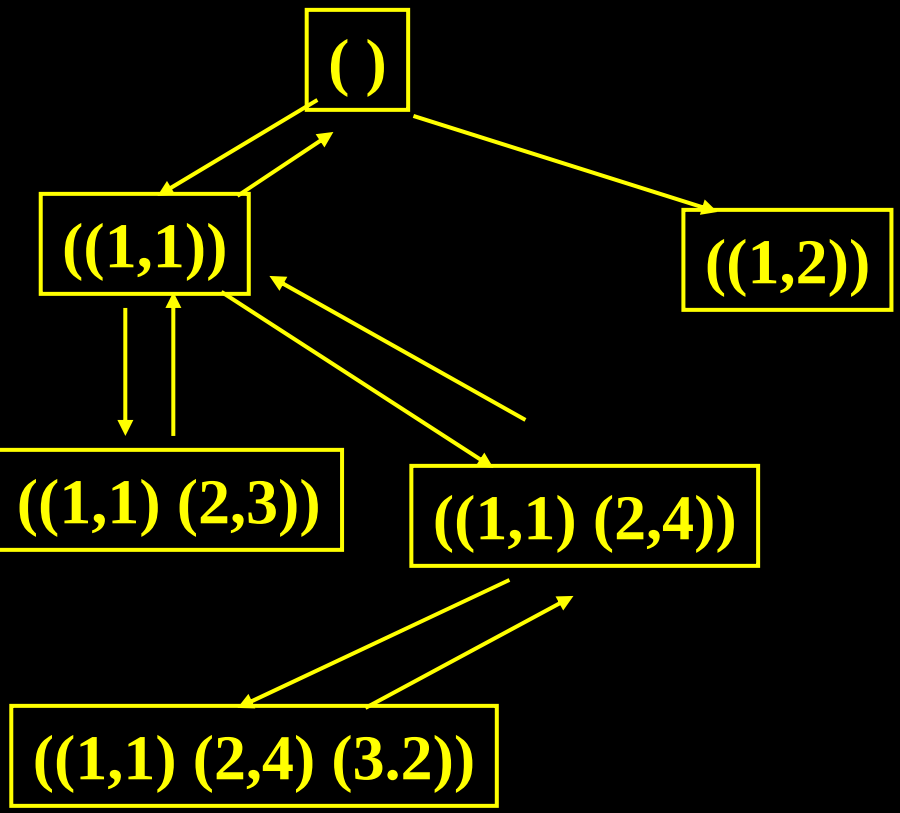
Q			
			Q

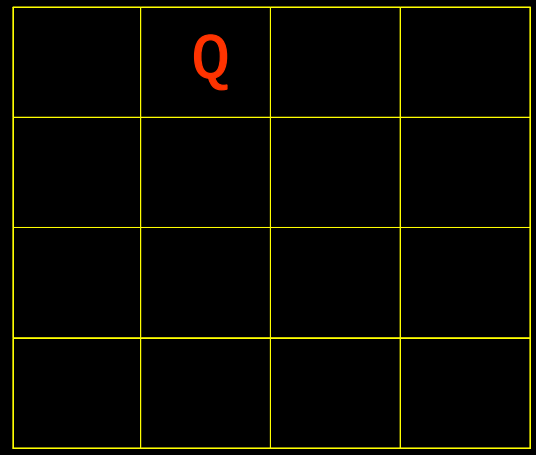


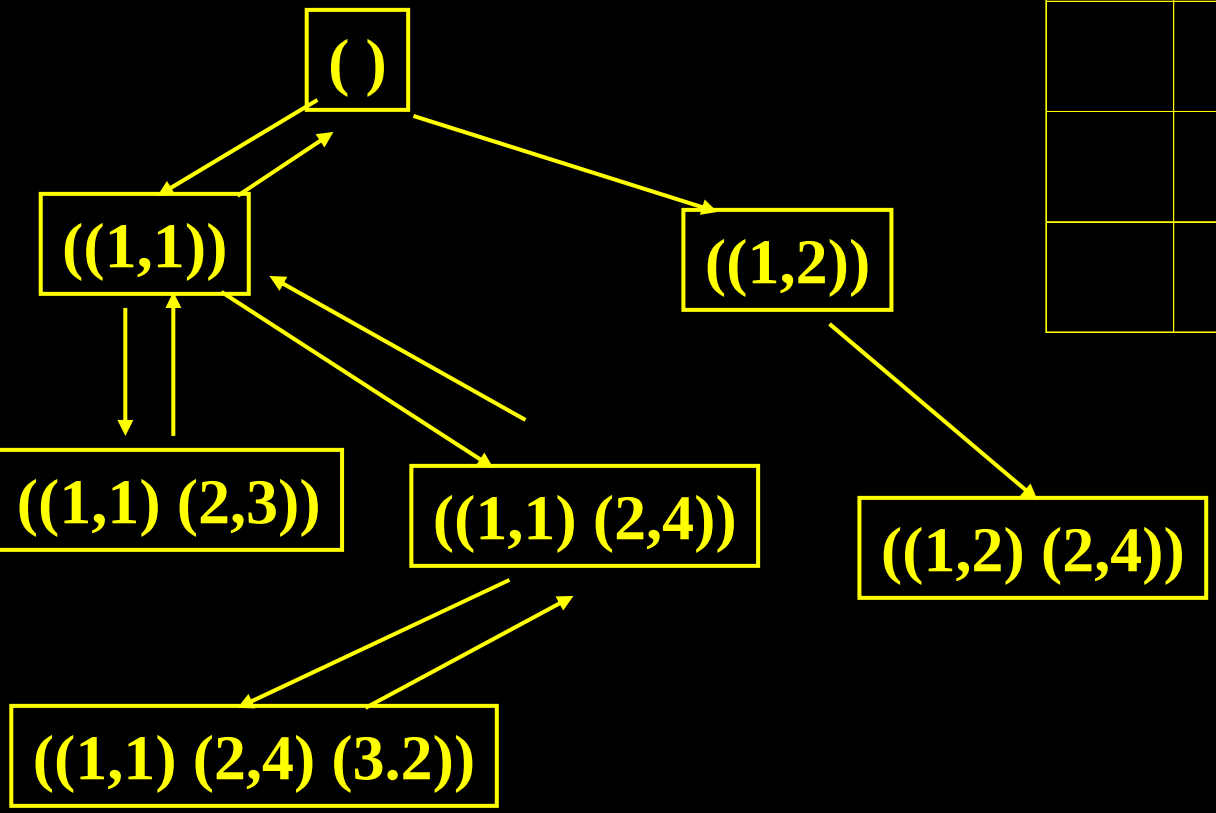
Q			

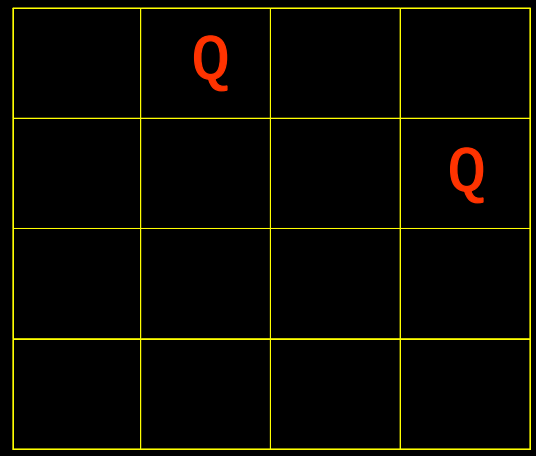

( )
((1,1))
((1,1) (2,3))
((1,1) (2,4))
((1,1) (2,4) (3.2))

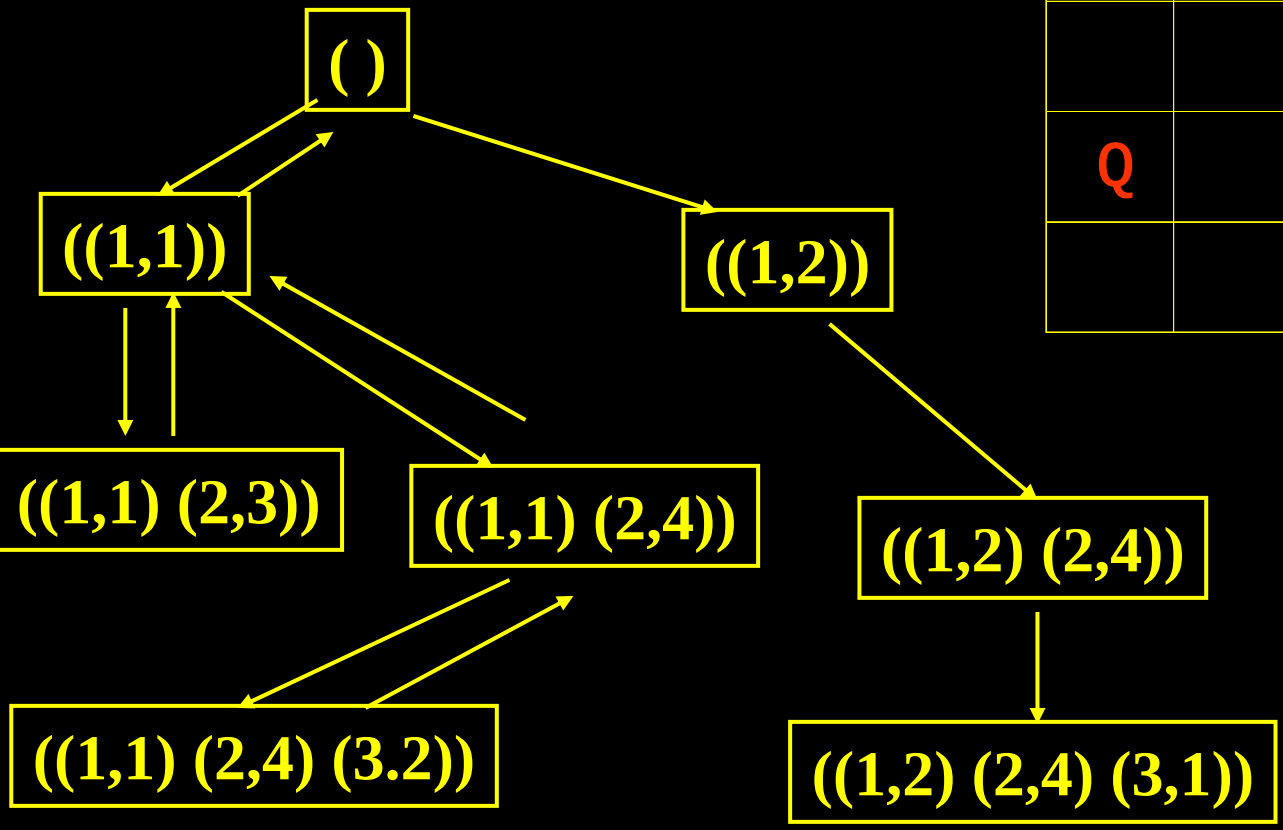


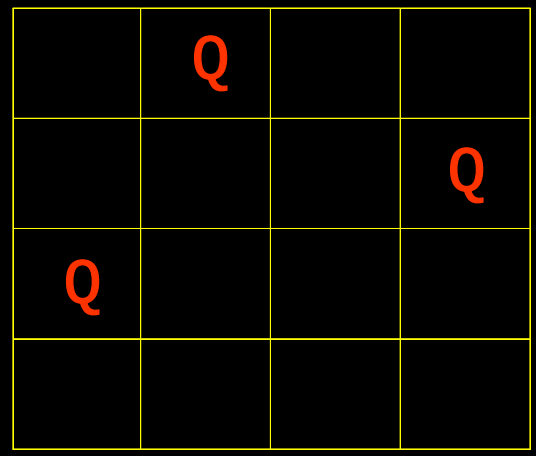

( )
((1,1))
((1,2))
((1,1) (2,3))
((1,1) (2,4))
((1,1) (2,4) (3.2))

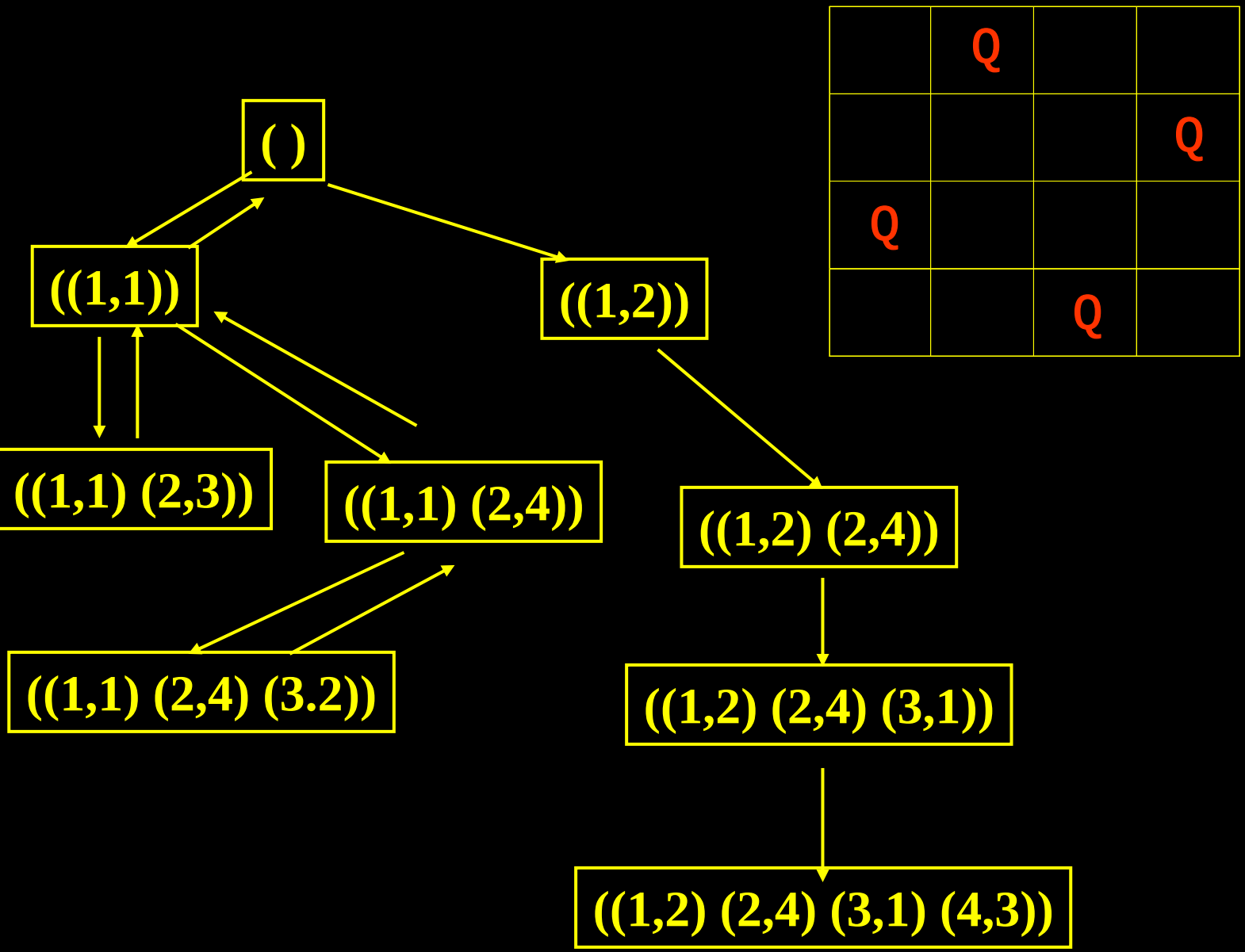

Q


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

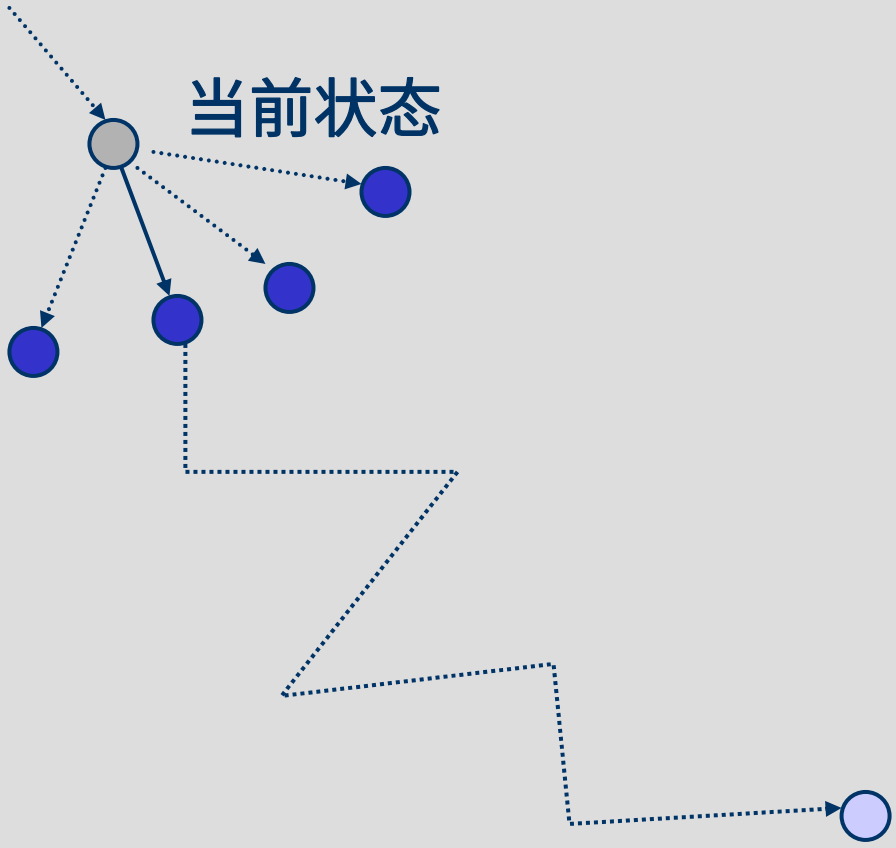

Q
Q


( )
((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))


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

递归的思想



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,    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);
```

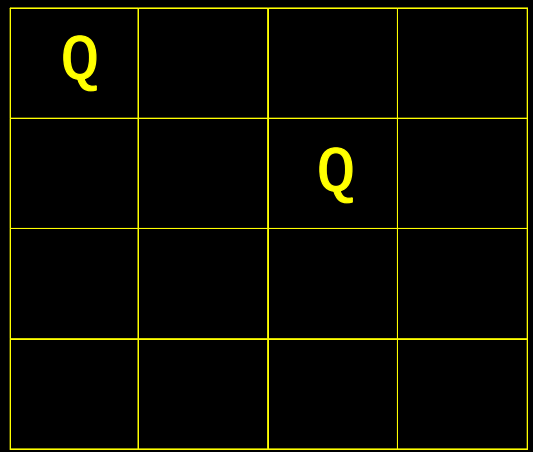
《人工智能》陈昭炯

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

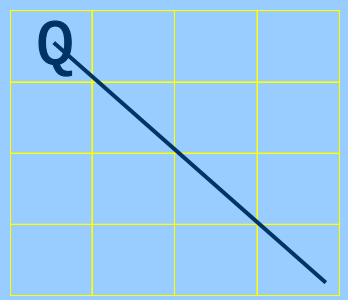


一些深入问题（续）

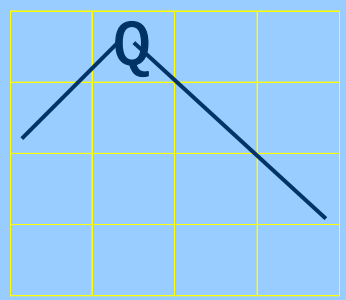
- 回溯搜索中知识的利用

  基本思想(以皇后问题为例)：

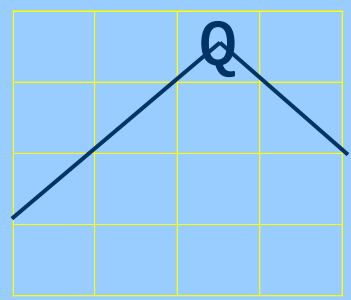
  尽可能选取划去对角线上位置数最少的。



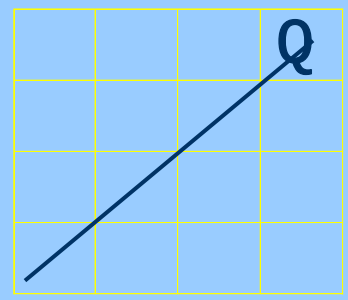
3



2

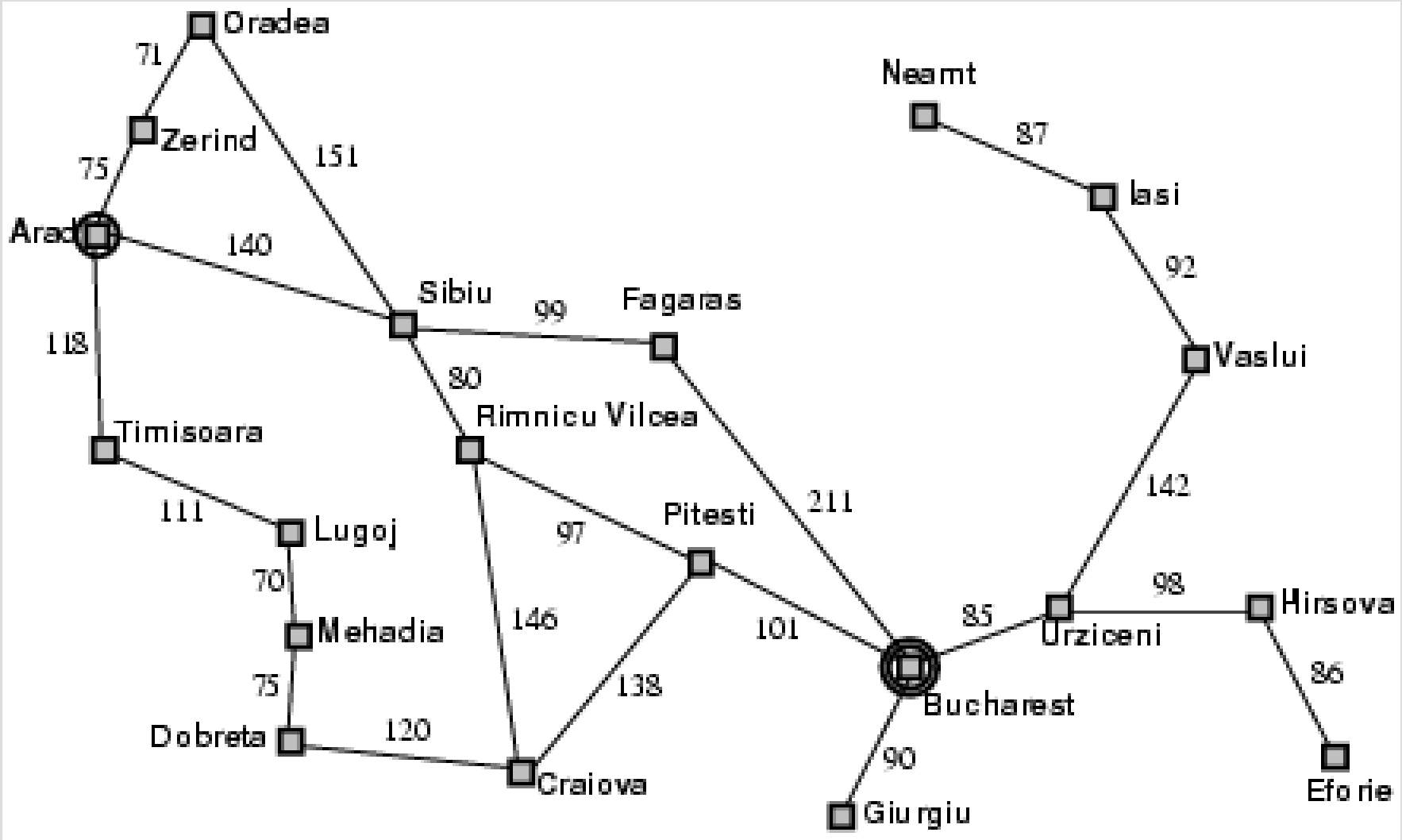


2



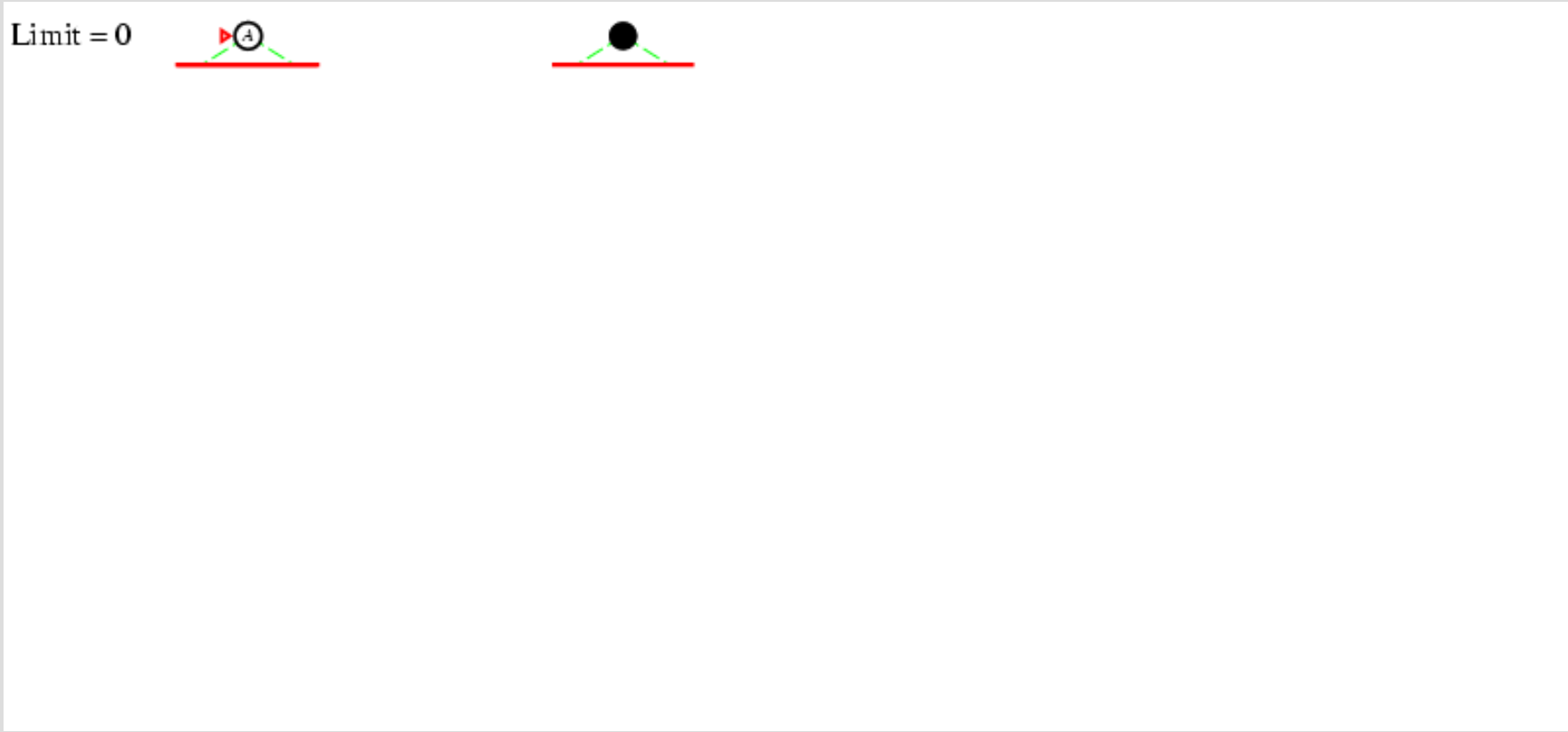
3

Example: Romania

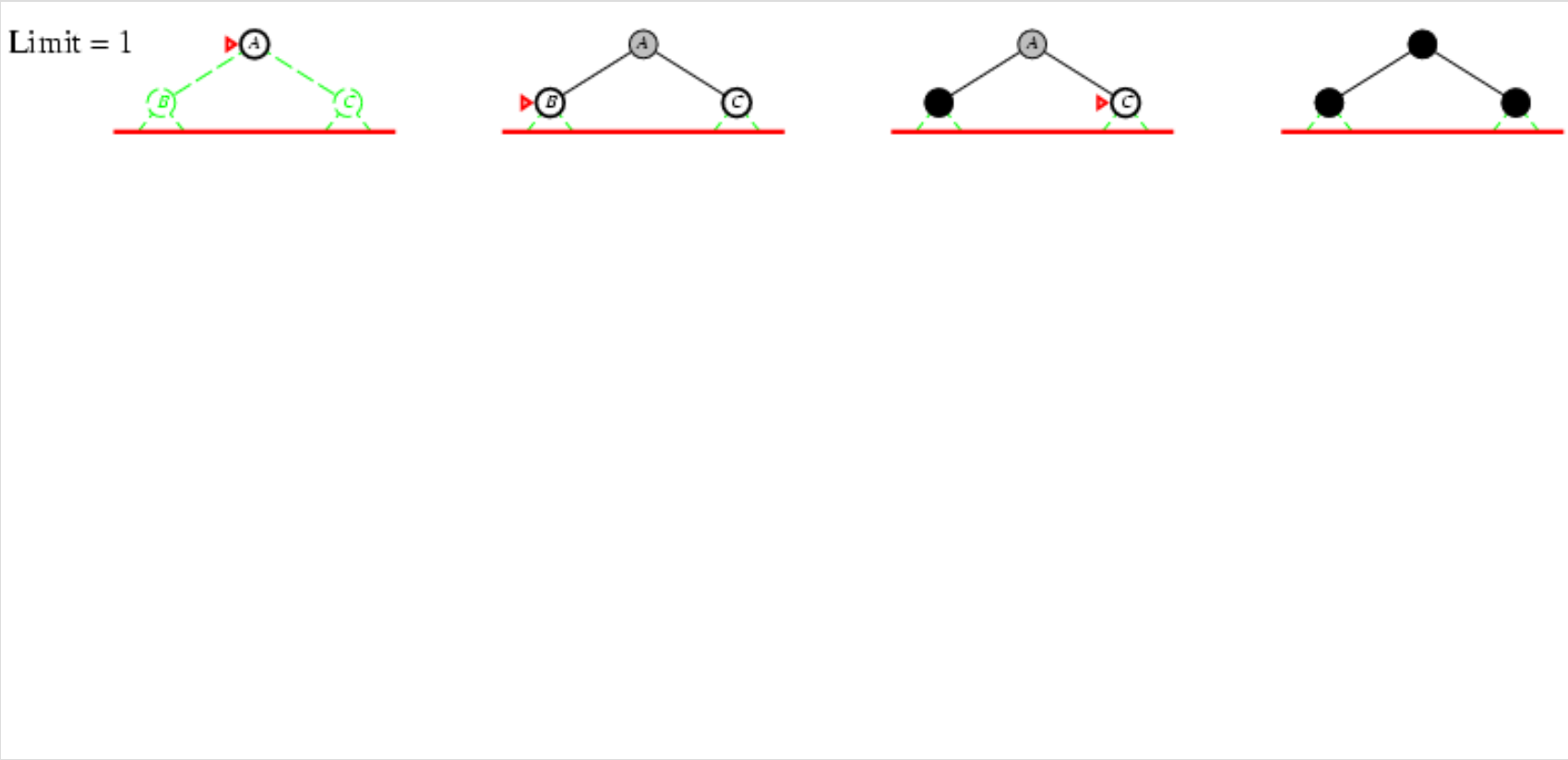


返回

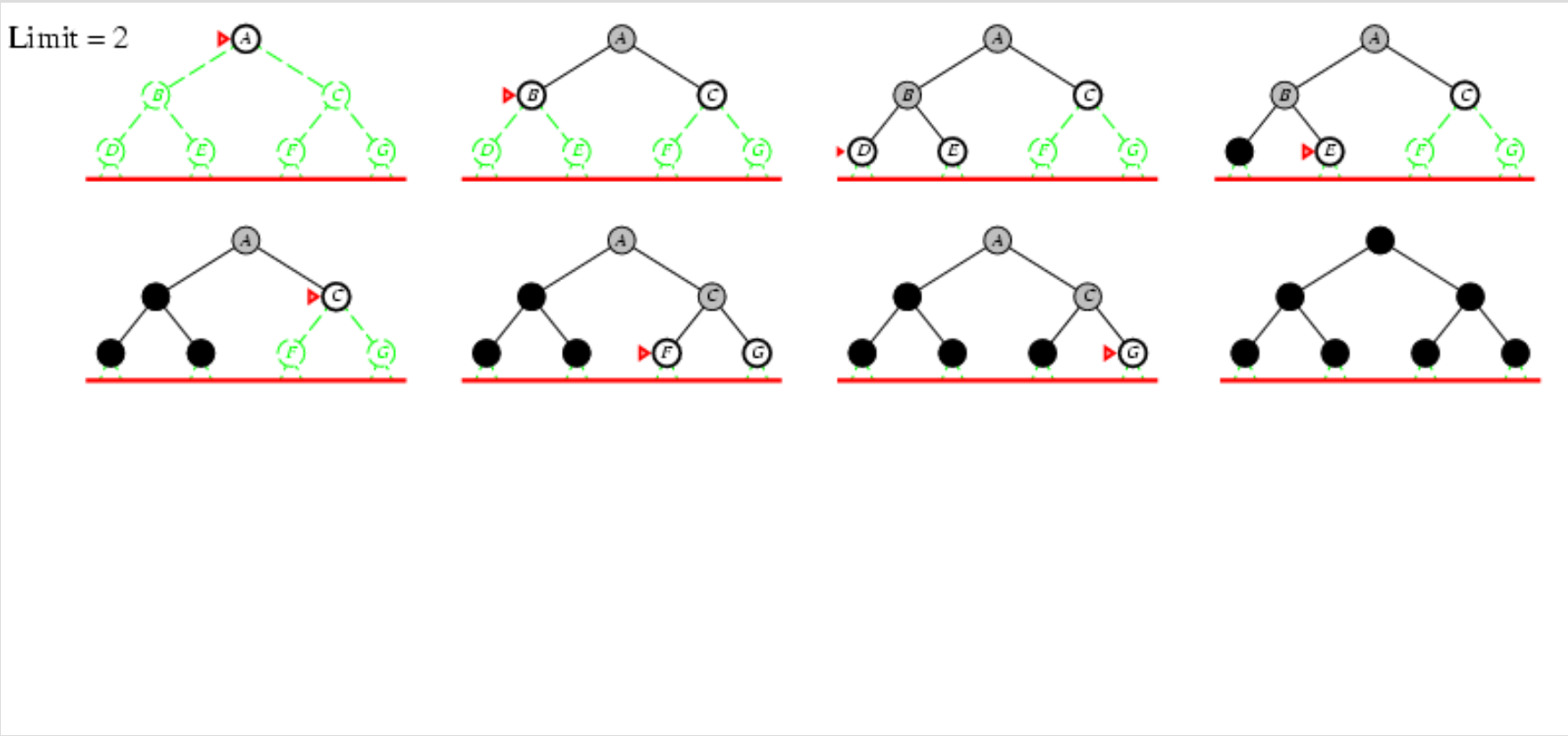
Iterative deepening search l =0



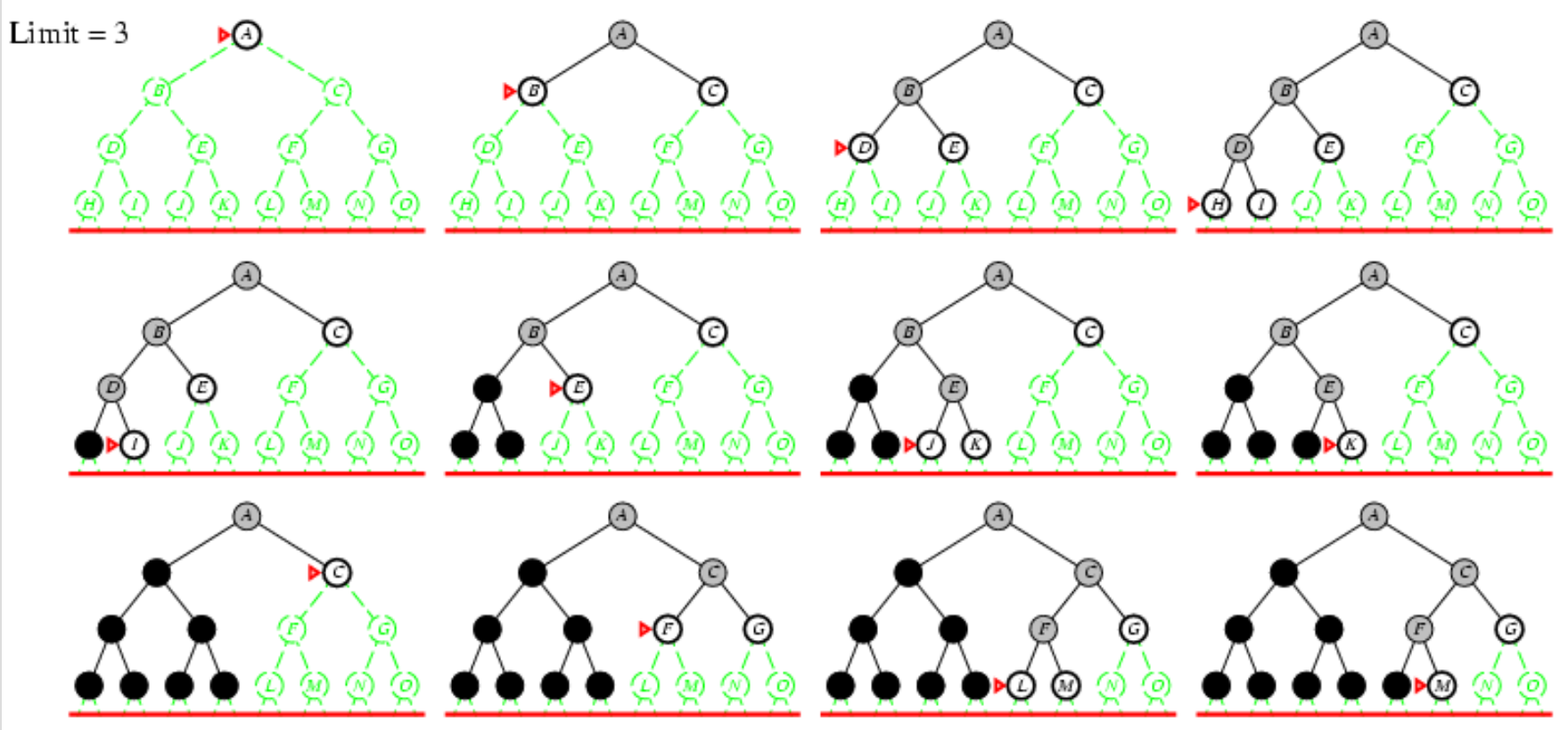
Iterative deepening search l =1



Iterative deepening search l =2



Iterative deepening search l =3



一些问题

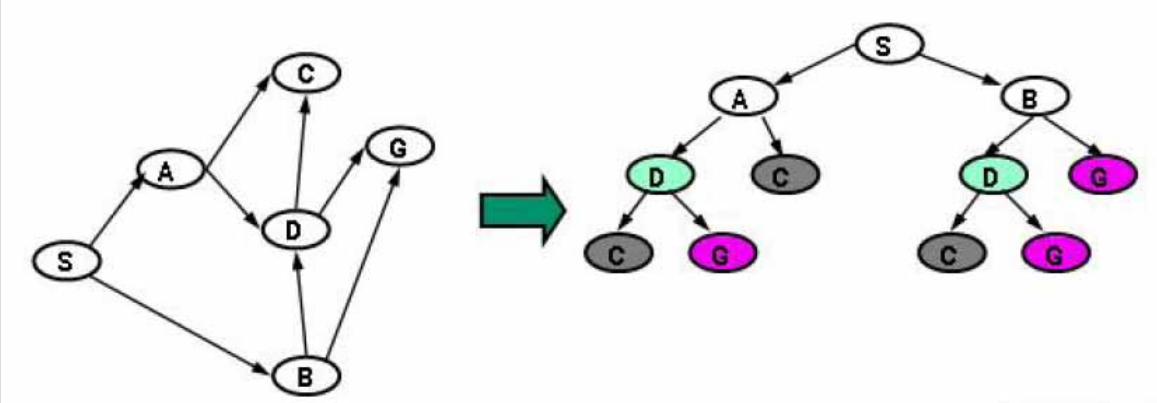
- 有界深度（迭代深入）每次变更深度时，是否重新从根结点开始搜索？如果不从根结点开始，能否从中间某个最接近于目标状态的节点继续下去？
- 如果每次从根结点开始,试与广度优先搜索比较生成节点的次数.
- 树的搜索如何处理重复？建模过程：8皇后（新的放在最左空列）不重复；滑块：重复，图搜索
- 记录所有节点，空间上有些不可行，如何处理
- 何种情况深度优先出现无穷分支？

一些问题（续1）

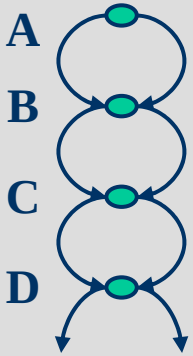
•广度搜索时，当前新生成的重复节点被删除不影响最优性；应用于深度则仍然不能保证最优性

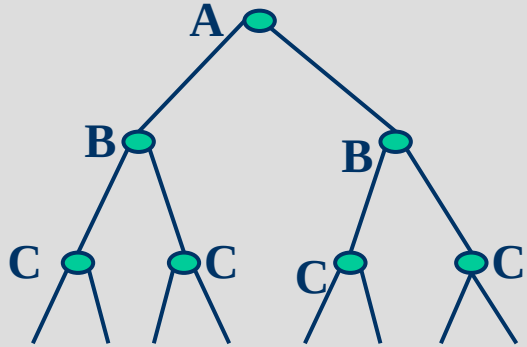
•**Searching the tree requires 180 times more work than searching the graph.**

•**Recognizing a repeated state takes time that varies with the size of the graph thus far seen. Solution？**



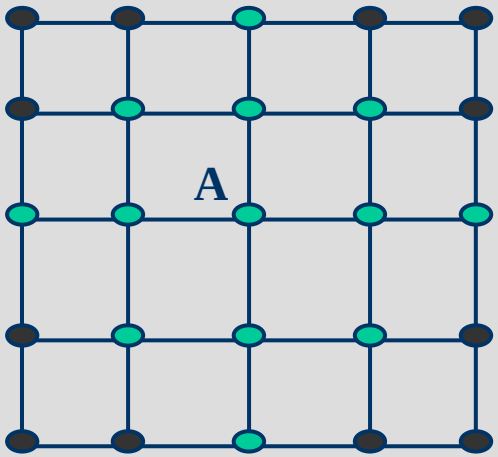
一些问题（续2）





状态空间：d+1个状态

搜索树：2^d个分支



任一给定状态：有四个后继，d^2级别状态个数

搜索树：4^d个分支

Summary

- **Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored**
- **Variety of uninformed search strategies**
- **Iterative deepening search uses only linear space and not much more time than other uninformed algorithms**