搜　索

2019年度南京大学“专创融合”特色示范课程培育项目

高 阳

http://cs.nju.edu.cn/rl, 2019.9.10

搜 索

状态空间搜索

状态空间搜索

搜索 — 一种问题求解方法

动机：理解AI中的静(不变, Statics)和动(变, Dynamics)

静：知识表示(Knowledge Representation)

动：推断或推理(Inference or Reasoning)

状态空间搜索 — GOFAI(Good Old-Fashioned Artificial Intelligence, Pre-statistical AI)的主要推断范型

由Newell和Simon在上世纪60年代，在通用问题求解器中发展起来的主要范型。

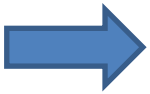
状态空间表示

状态空间图：节点 N 和连接(弧) A

初始状态：S 是 N 的非空子集

目标状态：G 是 N 的非空子集

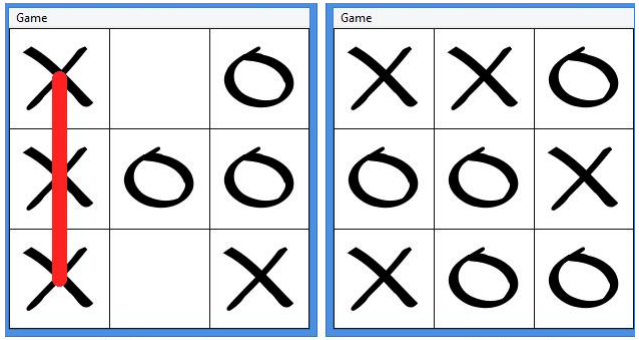
状态空间搜索：寻找从初始状态到目标状态的解路径(Solution Path)



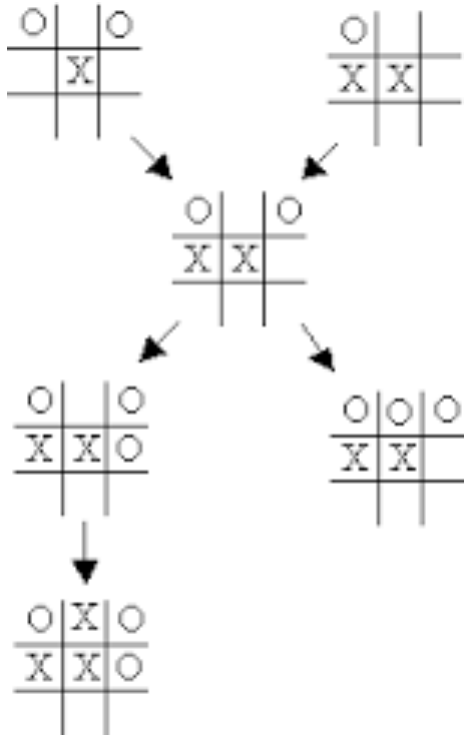
复杂问题求解的结构和策略

图和树的区别？

井字棋(Tic-Tac-Toe)

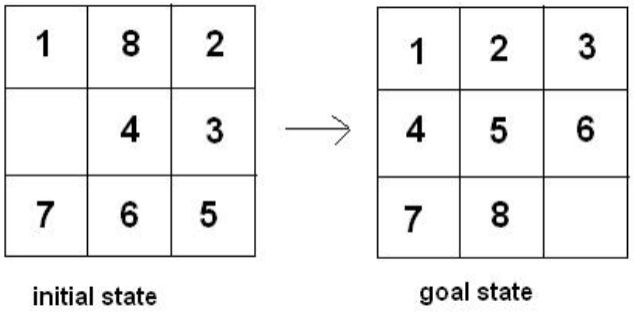


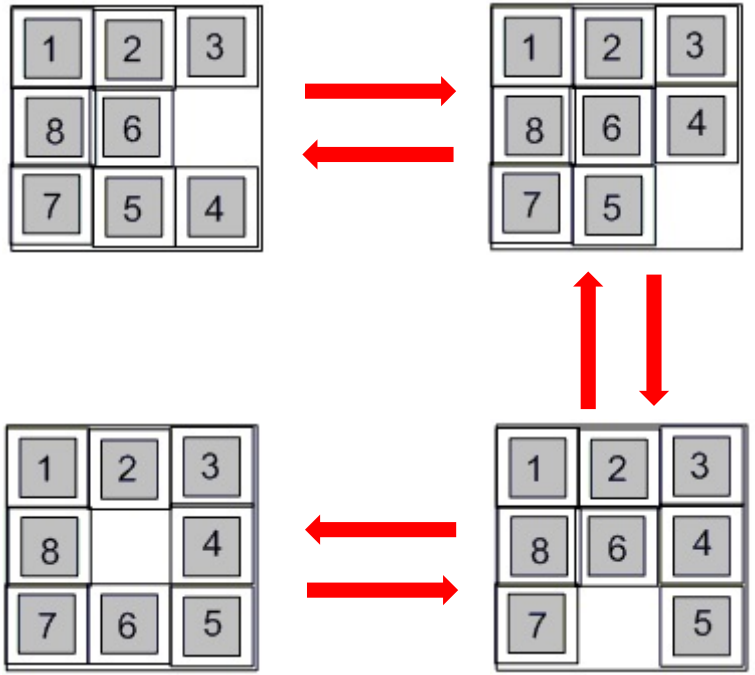
X player wins　　game ends in a draw



状态转移图：有向非循环图DAG

拼图游戏(8-Puzzle)

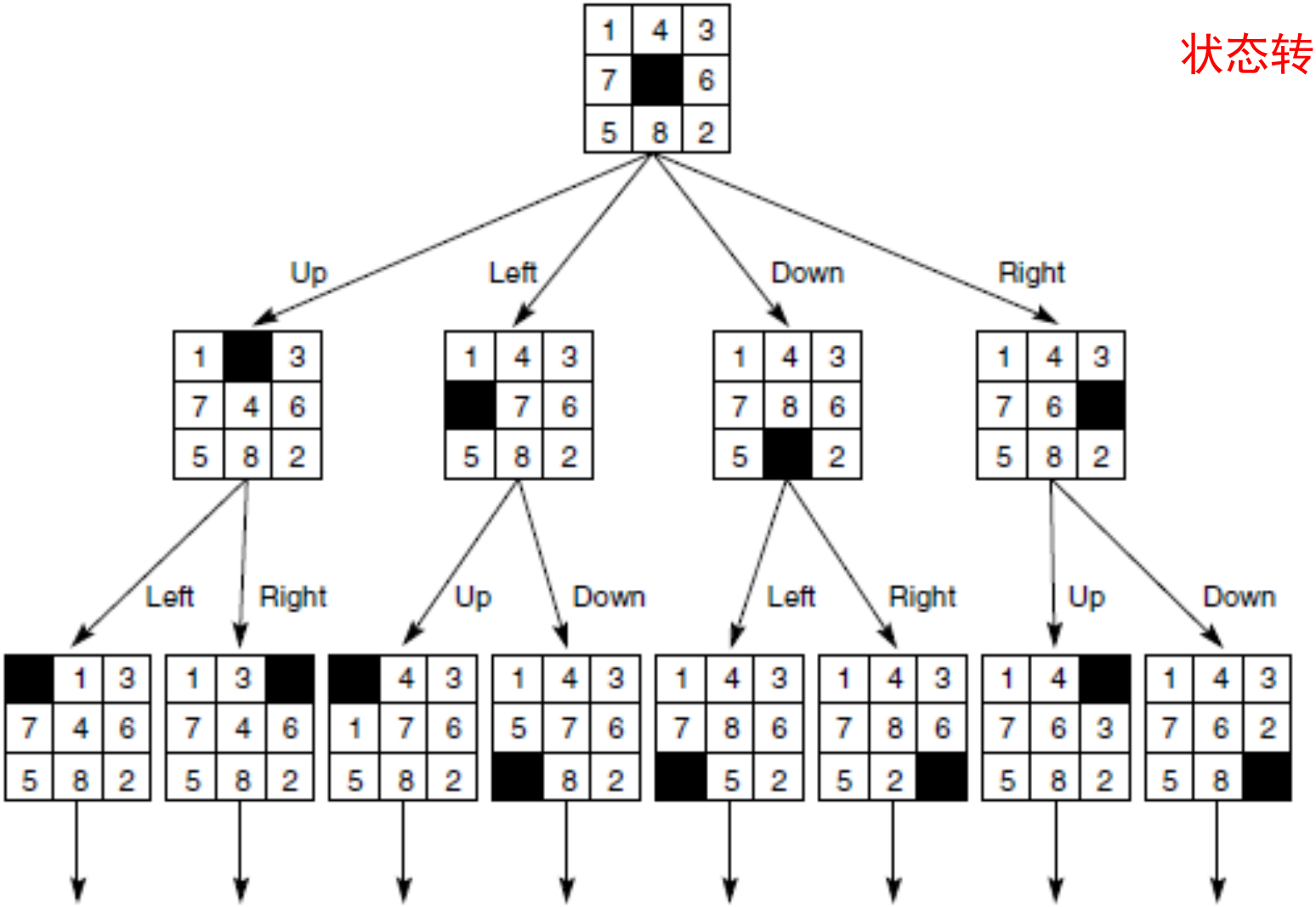




状态转移图：存在回路

拼图游戏(8-Puzzle)

状态转移树



状态空间表示

状态空间图：节点 N 和连接(弧) A

初始状态：S 是 N 的非空子集

目标状态：G 是 N 的非空子集

状态空间搜索：寻找从初始状态到目标状态的解路径(Solution Path)

图搜索： 需要检测和消除解路径上的循环

树搜索： 免除检测的开销

搜索树

根节点：初始状态

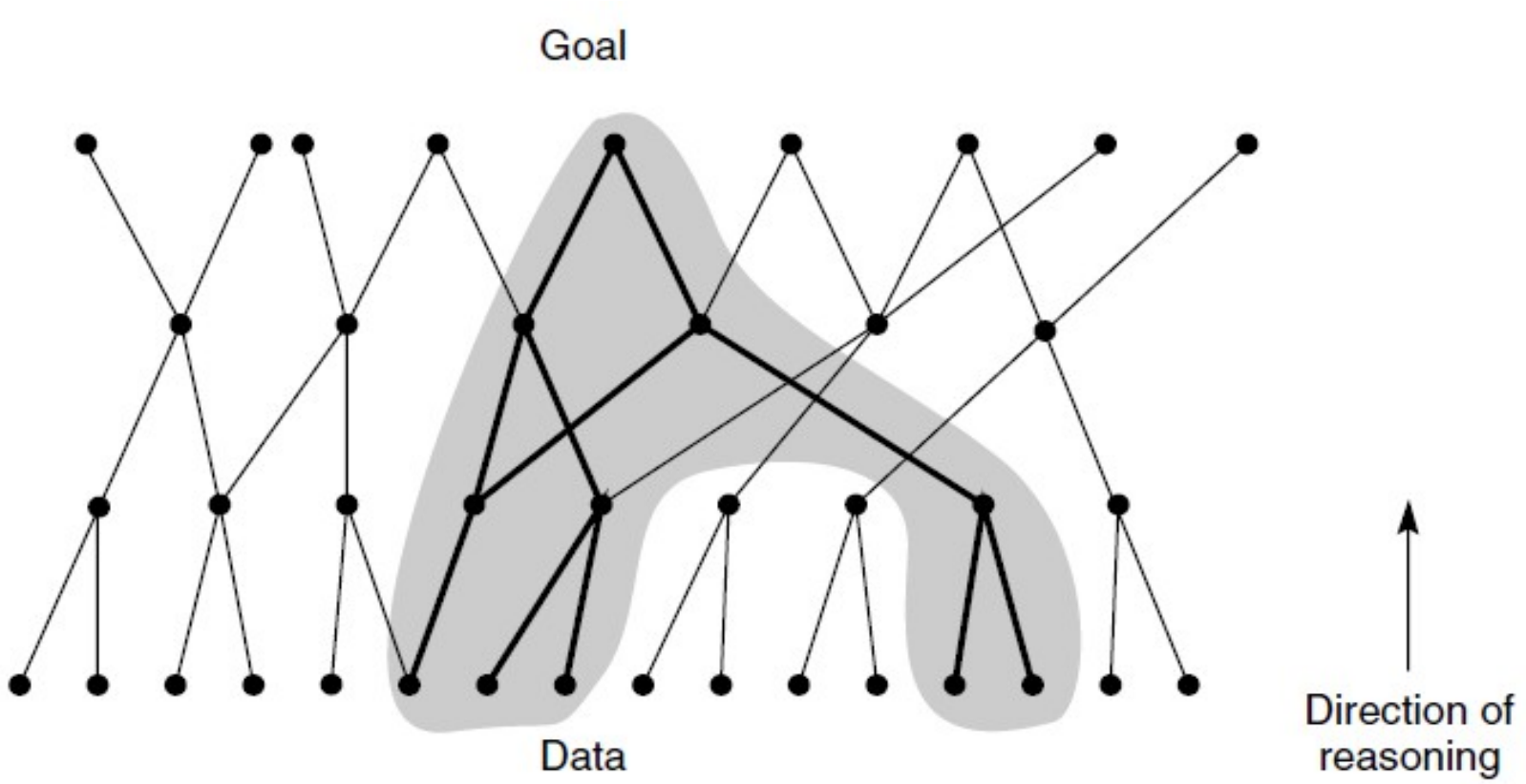
连　接：父节点上的合法动作

后　继：在父节点上采用合法动作所达到的子节点

搜索树：删除后继节点中已经出现的等价状态，所形成的树

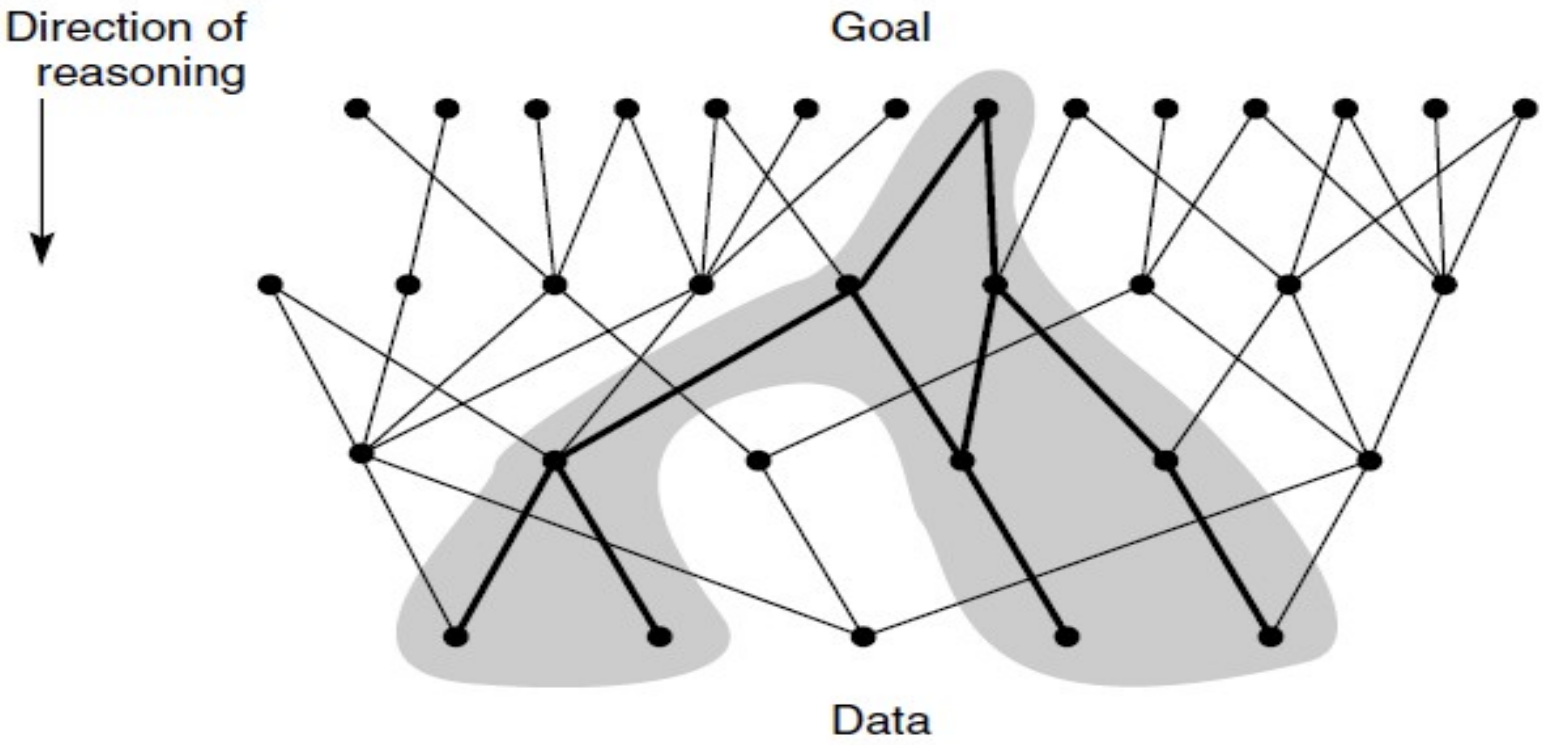
搜索方向，搜索次序，数据结构/算法

状态空间的搜索方向



数据驱动 – 前向搜索

状态空间的搜索方向



目标驱动 – 后向搜索

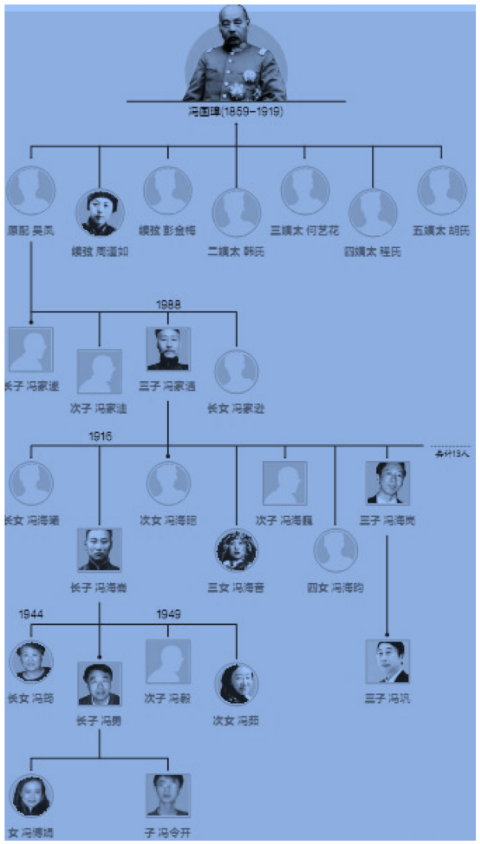
数据驱动 vs 目标驱动

问题1：谁是罪犯？

某村农民王某被害，有四个嫌疑犯A，B，C，D，公安局派出五个侦察员，他们带回的信息各不一样。

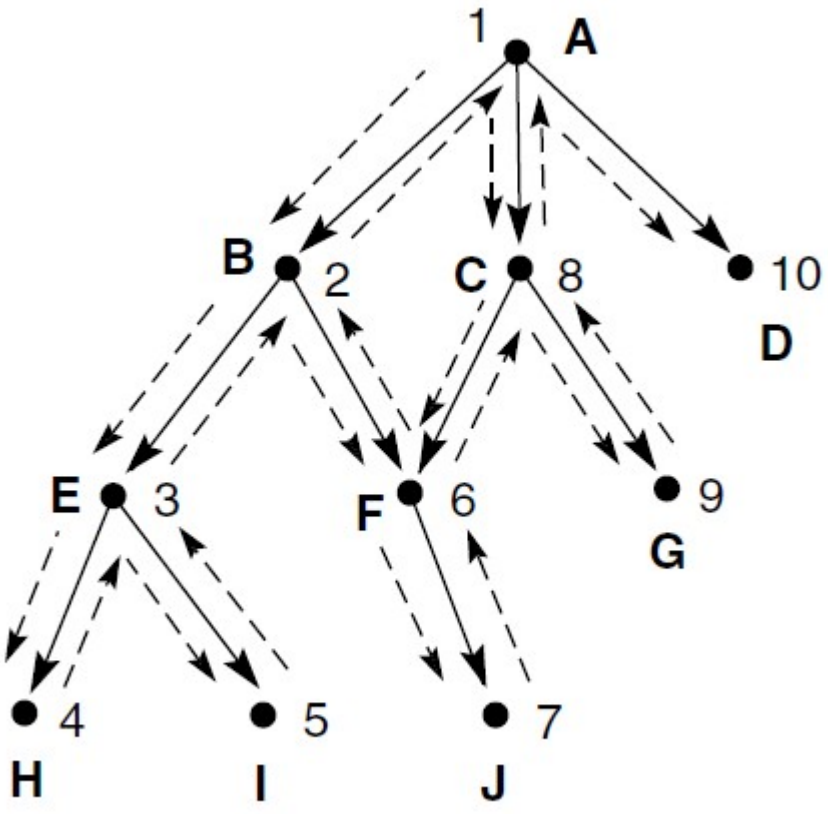
甲说：A，B中至少有一人作案；
乙说：B，C中至少有一人作案；
丙说：C，D中至少有一人作案；
丁说：A，C中至少有一人与此案无关；
戊说：B，D中至少有一人与此案无关。

问题2：冯巩是冯国璋的后人吗？



	目标形式表示	潜在目标数目	问题初始事实	匹配规则数目
数据驱动	难	多	多	
目标驱动	易		少	多

回朔技术



CS: Current State　　SL[]: State List
NSL[]:New State List　　DE[]: Dead End

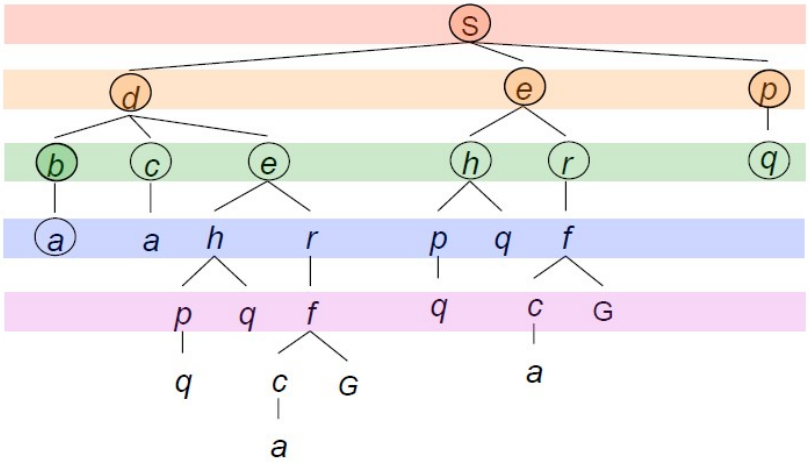
初值：SL=[A]；NSL=[A]；DE=[ ]；CS=A；

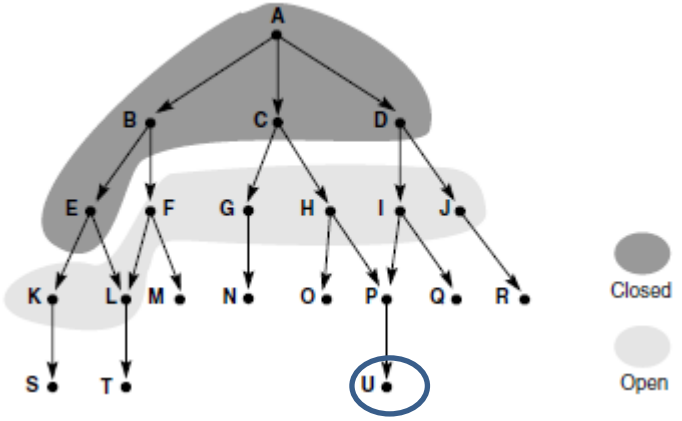
	CS	SL	NSL	DE
0	A	[A]	[A]	[]
1	B	[BA]	[BCDA]	[]
2	E	[EBA]	[EFBCDA]	[]
3	H	[HEBA]	[HIEFBCDA]	[]
4	I	[IEBA]	[IEFBCDA]	[H]
5	F	[FBA]	[FBCDA]	[EIH]
6	J	[JFBA]	[JFBCDA]	[EIH]
7	C	[CA]	[CDA]	[BFJEIH]
8	G	[GCA]	[GCDA]	[BFJEIH]

通过试错方式搜索状态空间所有解路径

宽度优先搜索

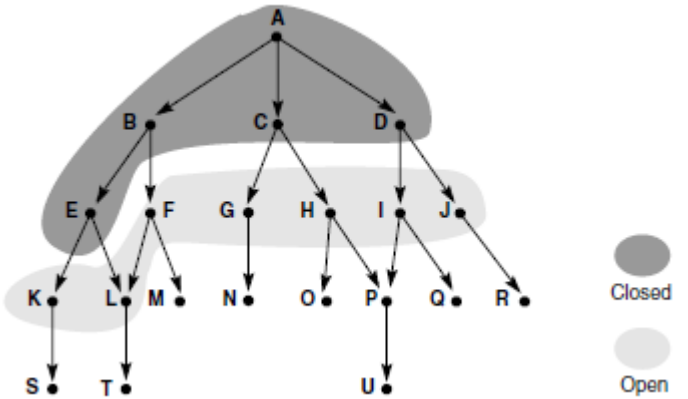
只有给定层不再存在需要探索的状态搜索才会转移到下一层





思考：采用回朔机制和数据结构，写出宽度优先搜索每步的状态。

宽度优先搜索



初值：OPEN =[A]；CLOSED=[ ]			
	CS	OPEN	CLOSED
0	A	[A]	[]
1	B	[BCD]	[A]
2	C	[CDEF]	[BA]
3	D	[DEFGH]	[CBA]
4	E	[EFGHIJ]	[DCBA]
5	F	[FGHIJKL]	[EDCBA]
6	G	[GHIJKLM]	[FEDCBA]
7	H	[HIJKLMN]	[GFEDBCA]
8	…	[…]	[…]

搜索次序：

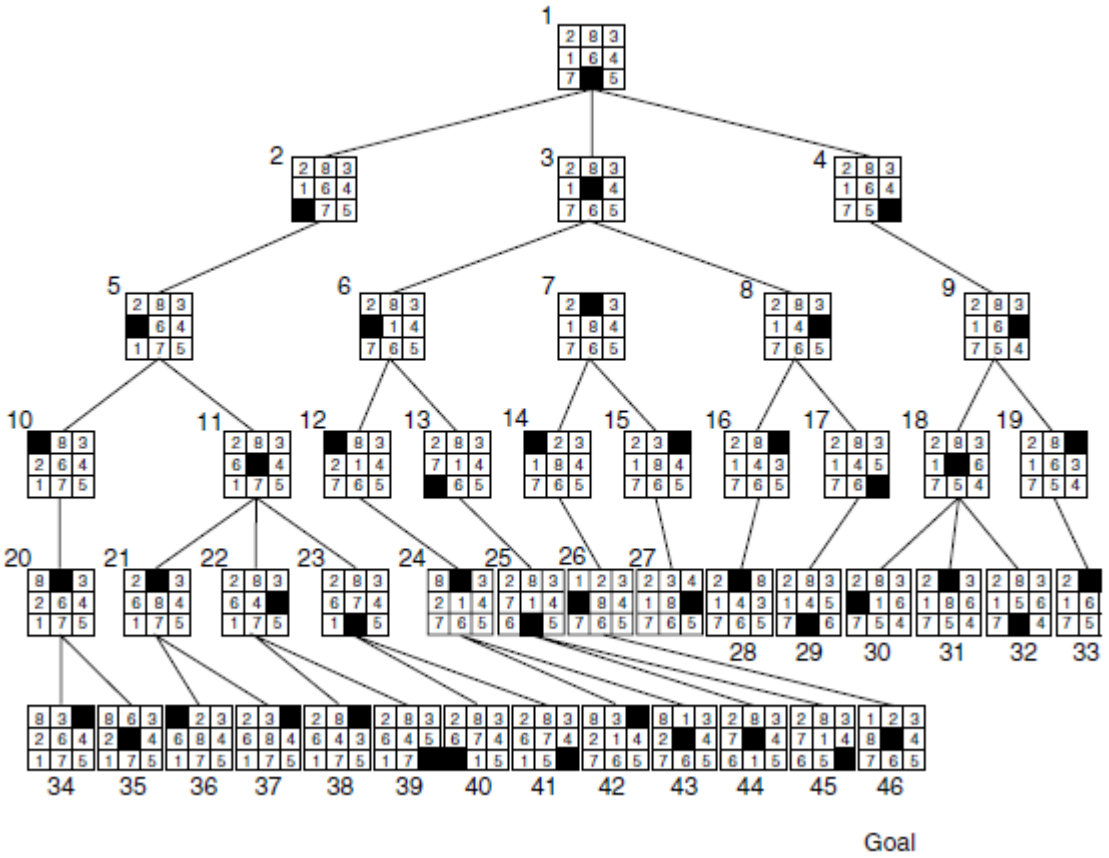
A, B, C, D, E, F, G, H, I, J, K, L , M, N, O,

P, Q, R, S, T, U

特点：

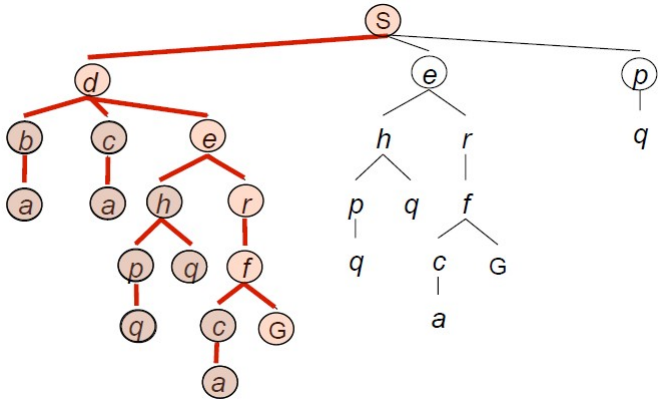
- 先入先出FIFO，从右侧加入列表，左侧移出
- 可保证发现目标状态的最短路径
- 组合爆炸

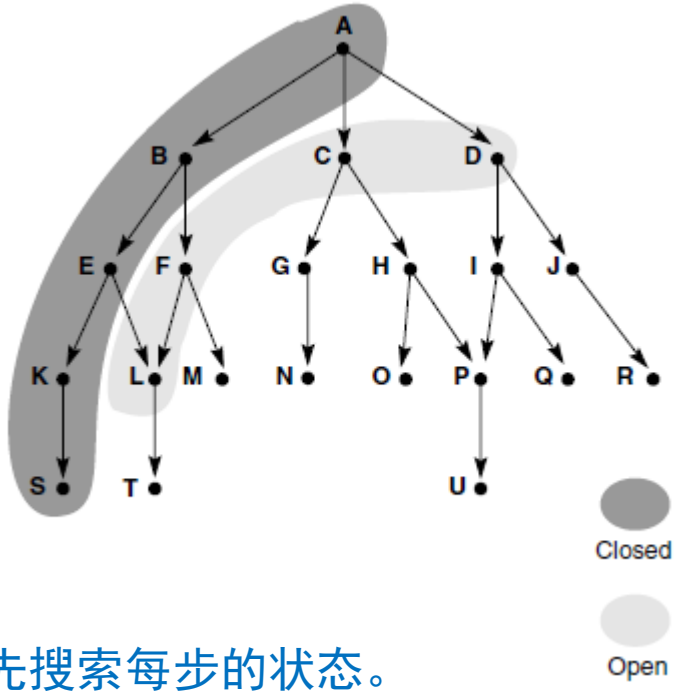
宽度优先搜索(8-Puzzle)



深度优先搜索

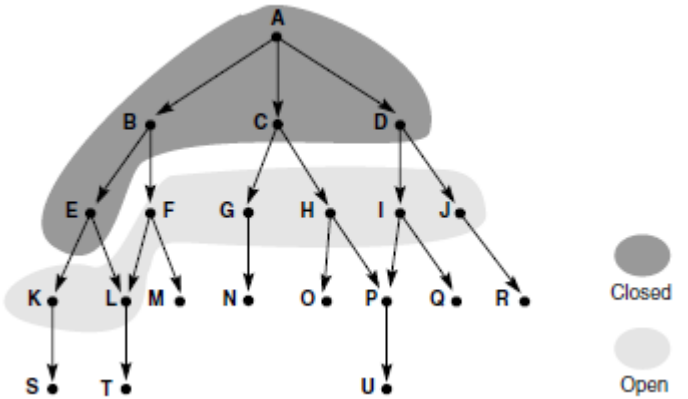
在分析一个节点的兄弟节点之前，
必须分析完所有的孩子节点和其后代





思考：采用回朔机制和数据结构，写出深度优先搜索每步的状态。

深度优先搜索



初值：OPEN =[A]；CLOSED=[ ]			
	CS	OPEN	CLOSED
0	A	[A]	[]
1	B	[BCD]	[A]
2	E	[EFCD]	[BA]
3	K	[KLFCD]	[EBA]
4	S	[SLFCD]	[KEBA]
5	L	[LFCD]	[SKEBA]
6	T	[TFCD]	[LSKEBA]
7	F	[FCD]	[TLSKEBA]
8	…	[…]	[…]

搜索次序：

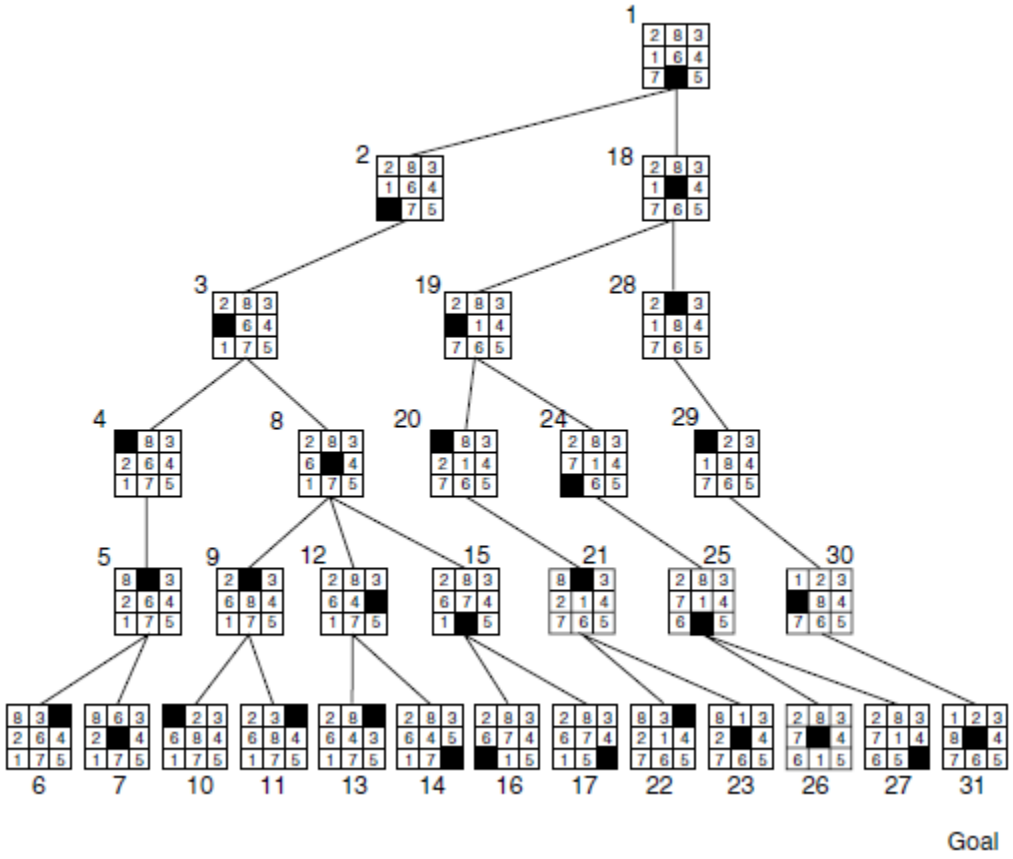
A, B, E, K, S, L, T, F, M, C, G, N, H, O, P,

U, D, I, Q, J, R

特点：

- 后入先出LIFO，从左侧加入列表，左侧移出
- 不保证发现目标状态的最短路径
- “可能迷失”在深度空间中

深度优先搜索(8-Puzzle)



宽度优先 vs 深度优先

问题1：宽度优先的优点和缺点？

问题2：深度优先的优点和缺点？

解决方案：迭代加深的深度优先搜索

讨论：有界深度优先 vs 迭代加深深度优先

迭代加深深度优先

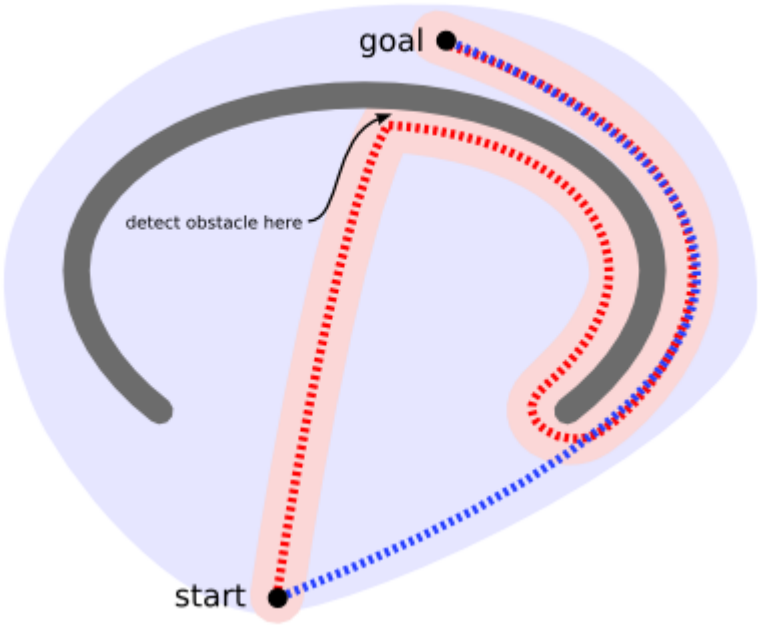
```
procedure IDDFS(root)
    for depth from 0 to ∞
        found ← DLS(root, depth)
        if found ≠ null
            return found

procedure DLS(node, depth)
    if depth = 0 or node is a goal
        return node
    else if depth > 0
        foreach child of node
            found ← DLS(child, depth−1)
            if found ≠ null
                return found
    return null
```

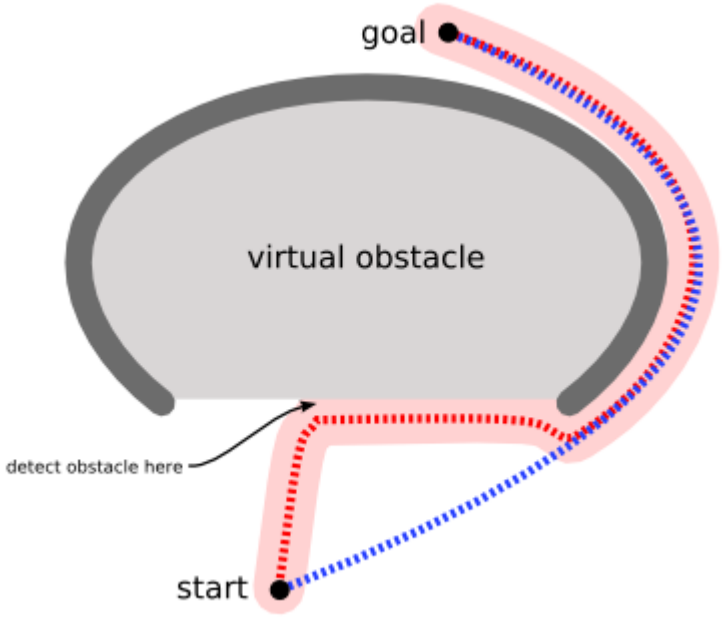
迭代加深深度优先与广度优先等价

搜 索

从穷举搜索到启发式搜索

人工智能中的启发

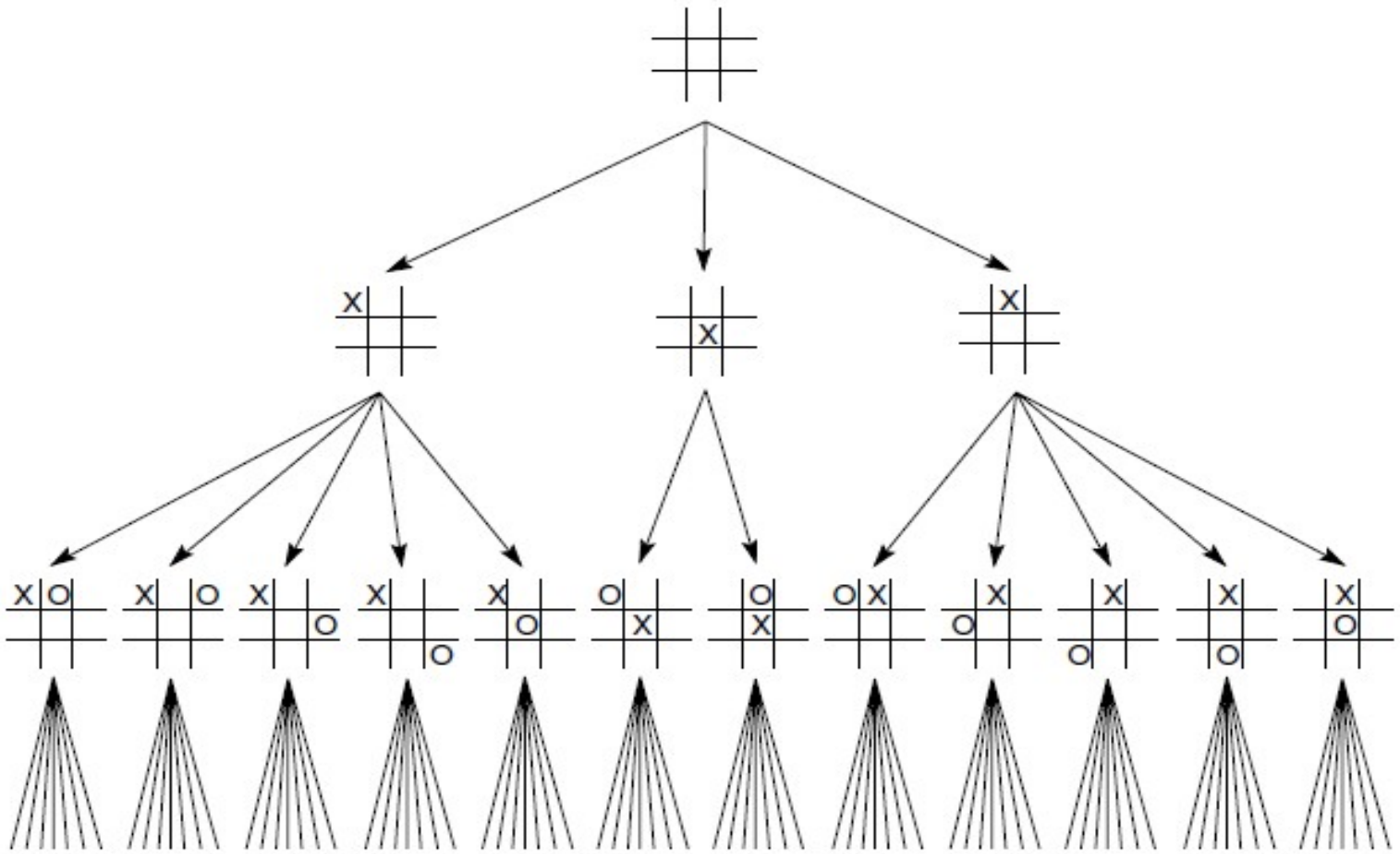




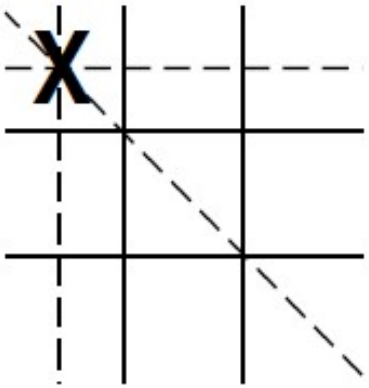
启发(Heurisitic)：有了启发，系统才展示出智能。 —— Newell and Simon

为什么需要启发：1. 没有精确解；2. 约简搜索的状态空间。

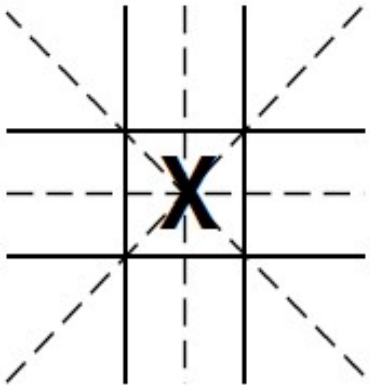
传统的状态空间



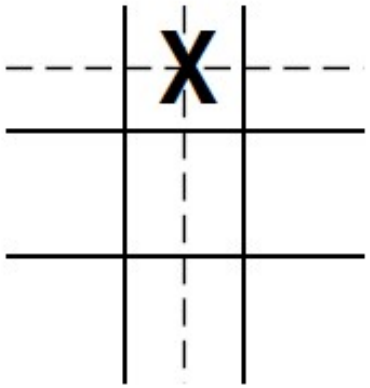
启发



Three wins through
a corner square



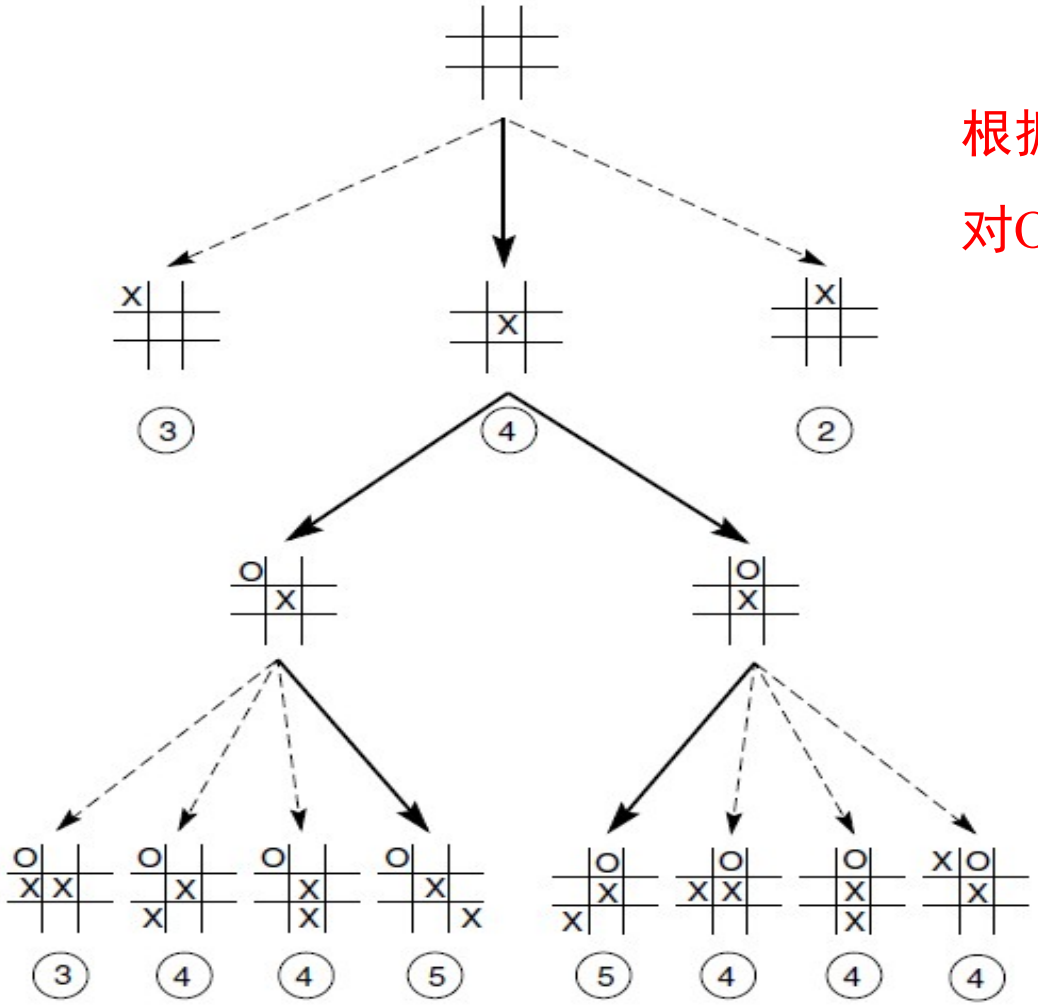
Four wins through
the center square



Two wins through
a side square

启发：取获胜线路最多的着点

启发诱导的状态空间

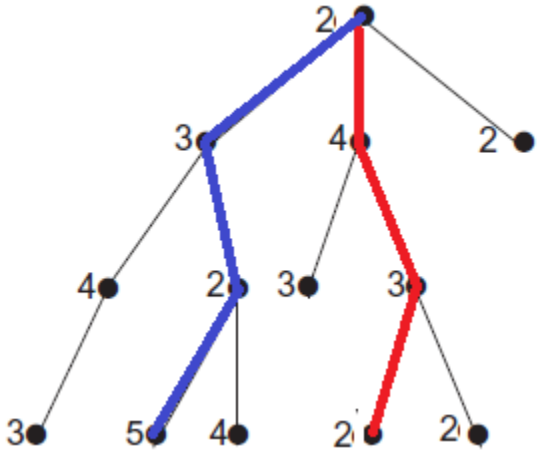


根据启发值，

对OPEN列表中的状态进行排序

爬山搜索

- ✓ 第一步：扩展当前节点以及其子节点，并进行评估；
- ✓ 第二步：选择“最优”的子节点作为下一节点；
- ✓ 第三步：如果所有子节点的评估，都比当前节点“劣”，则算法终止。

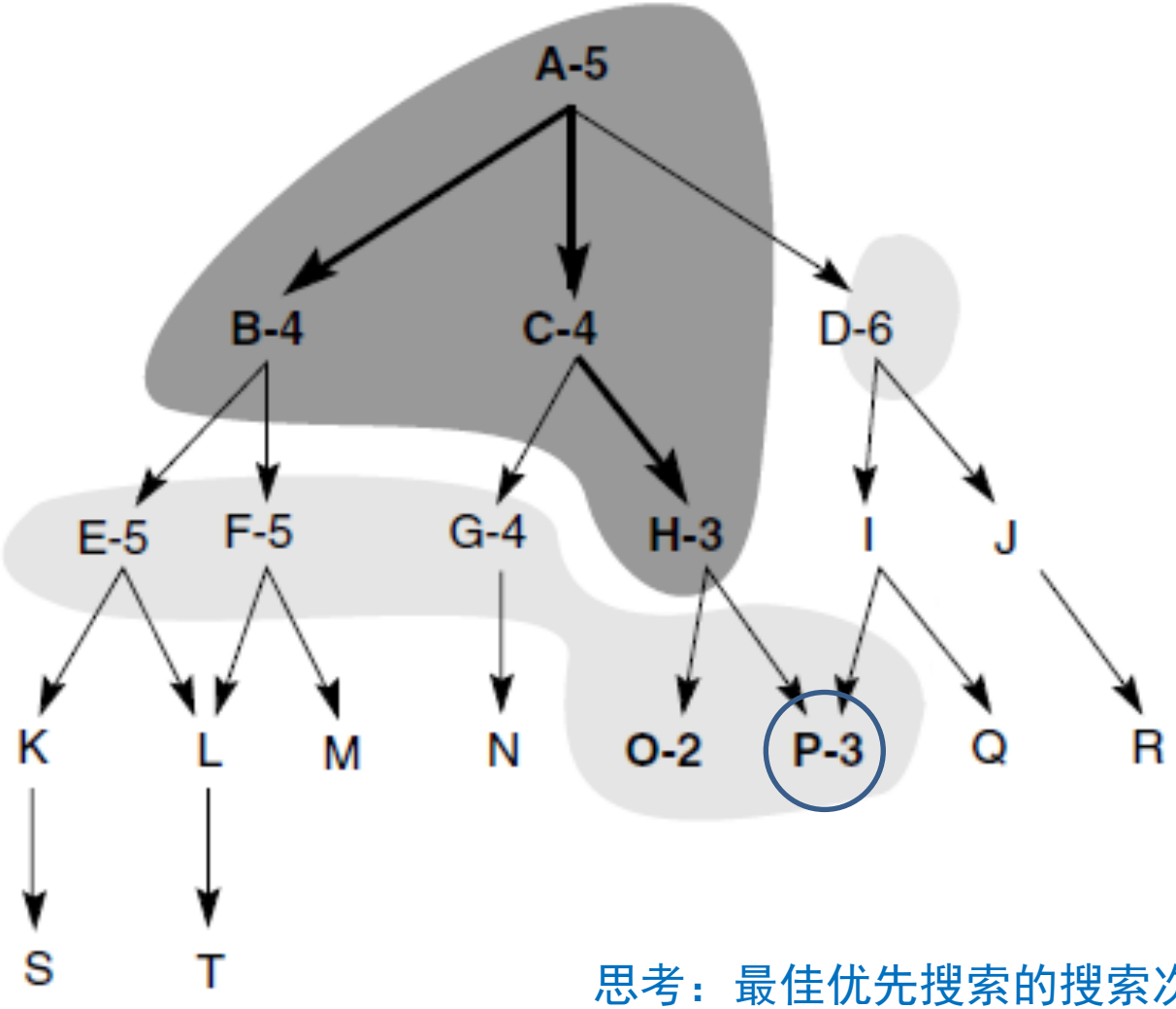


Local Optima

解决方案：

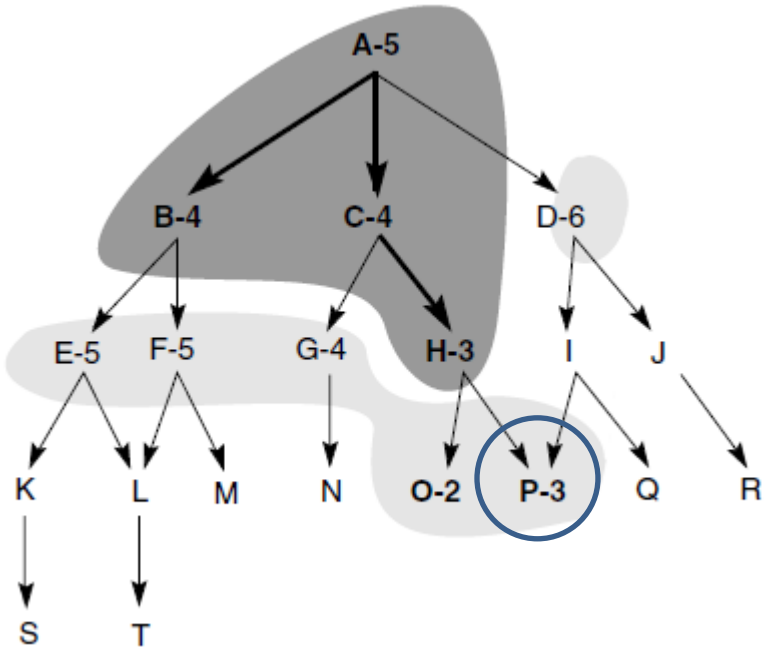
在爬山搜索中增加回朔机制

最佳优先搜索



思考：最佳优先搜索的搜索次序

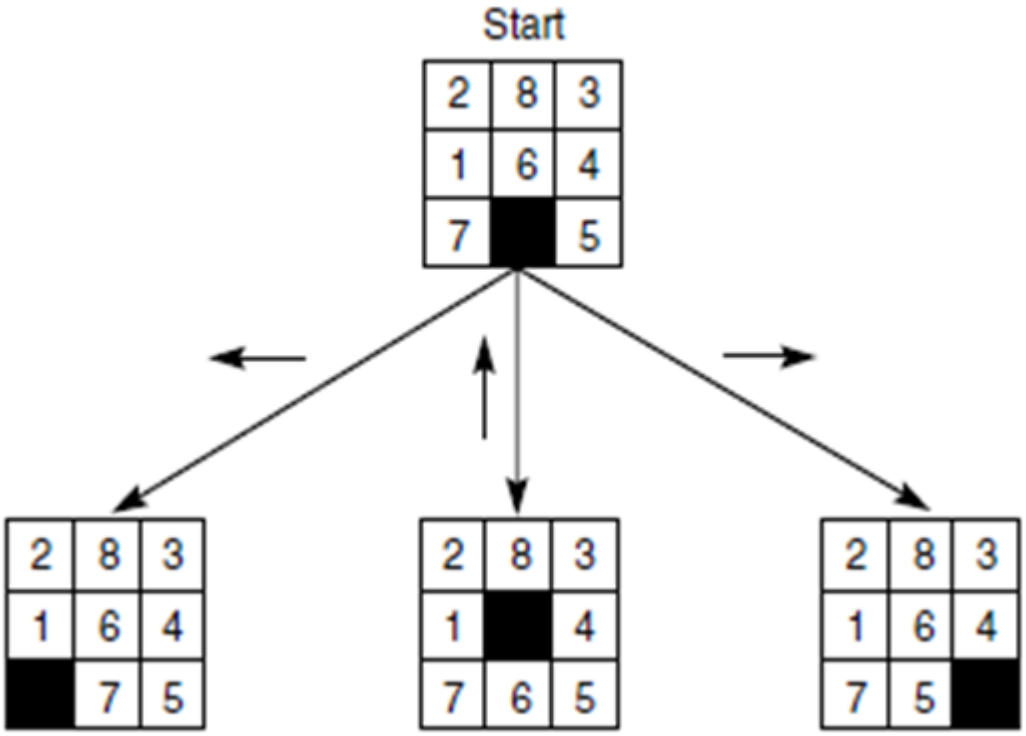
最佳优先搜索



初值：OPEN =[A]；CLOSED=[ ]			
	CS	OPEN	CLOSED
0	A	[A]	[]
1	B	[BCD]	[A]
2	C	[CEFD]	[BA]
3	H	[HGEFD]	[CBA]
4	O	[OPGEFD]	[HCBA]
5	P	[PGEFD]	[OHCBA]

如何定义启发式估值函数h(n)?

8-Puzzle

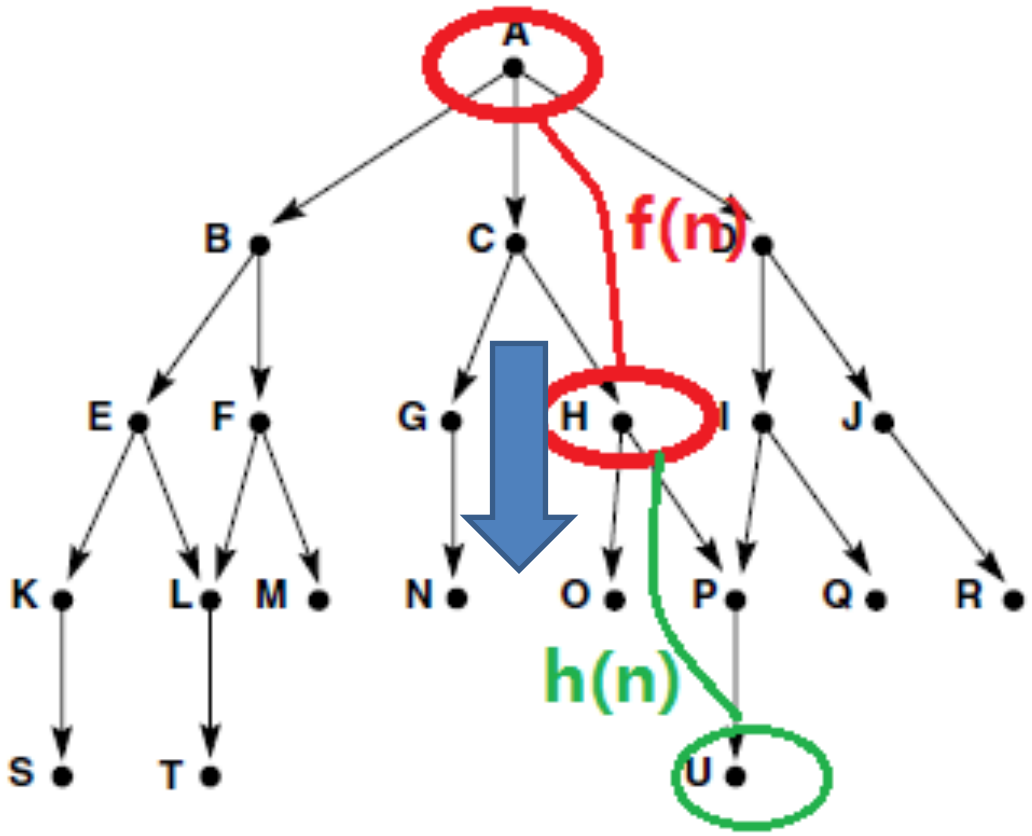


如何定义启发式估值函数h(n)?

三种启发

	错位牌数	错位牌距离和	距离+2*颠倒牌数
	5	6	6
	3	4	4
	5	6	6

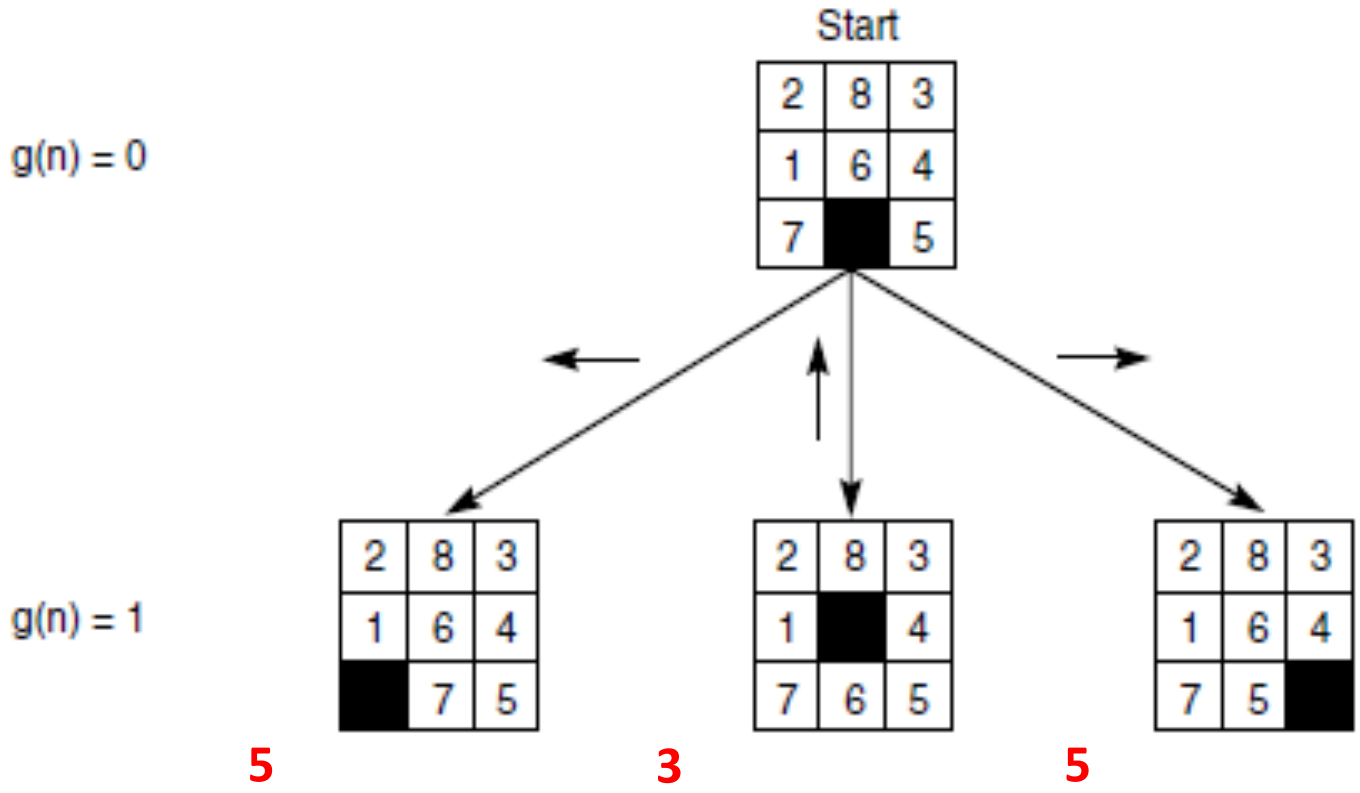
启发式评估函数



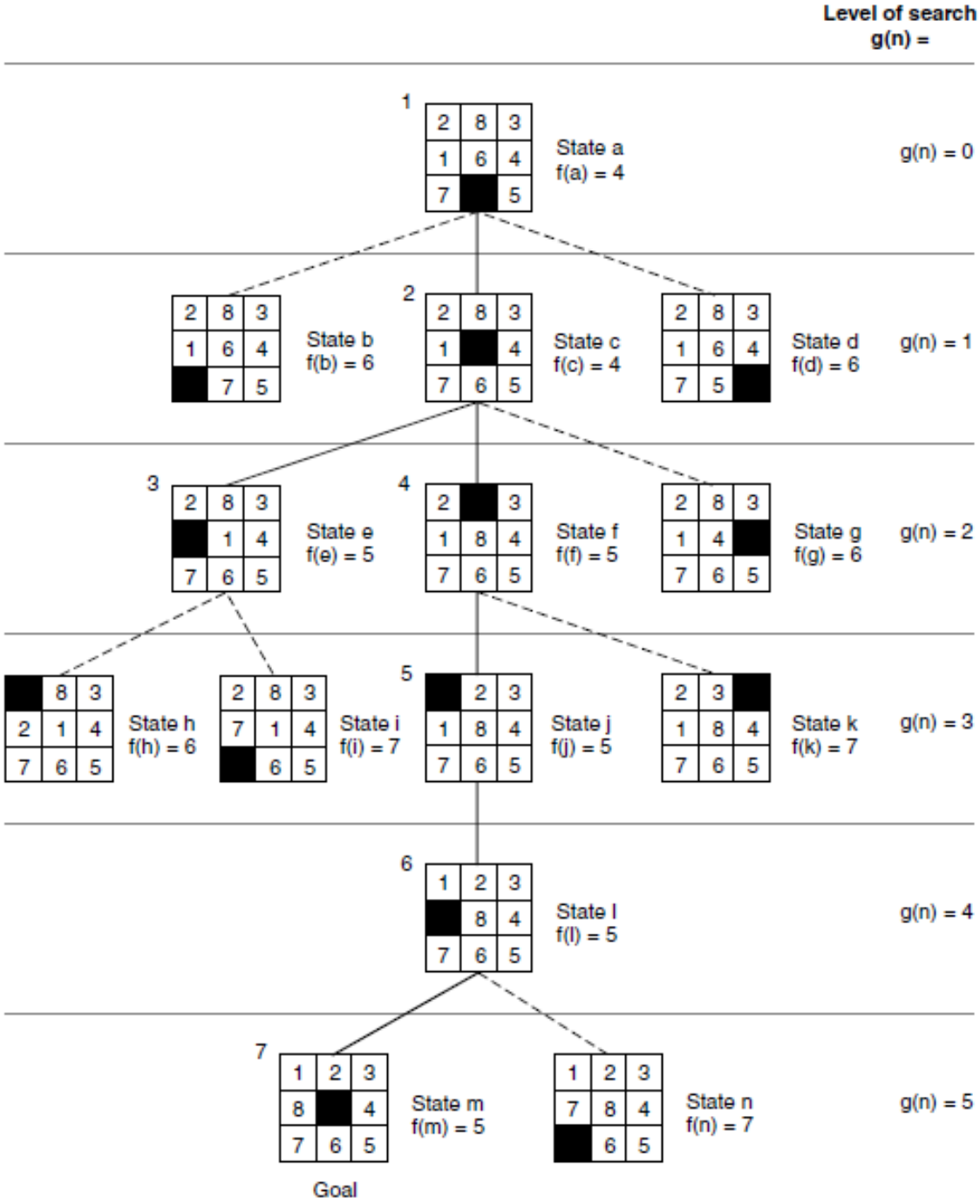
当存在相同启发式估值时如何选择

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

启发诱导的搜索(8-Puzzle)



h(n)定义为与目标相比错位牌的数目


Level of search
g(n) =
1
State a
f(a) = 4
g(n) = 0
2
State b
f(b) = 6
State c
f(c) = 4
State d
f(d) = 6
g(n) = 1
3
State e
f(e) = 5
4
State f
f(f) = 5
State g
f(g) = 6
g(n) = 2
State h
f(h) = 6
State i
f(i) = 7
5
State j
f(j) = 5
State k
f(k) = 7
g(n) = 3
6
State l
f(l) = 5
g(n) = 4
7
State m
f(m) = 5
State n
f(n) = 7
g(n) = 5
Goal

A*算法

- A's 评估函数

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

- 最优评估函数

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

- $g(n) \geq g^*(n)$

- 如果 $h(n) \leq h^*(n)$, 则A算法为 A* 算法

A*算法可采纳性和单调性

思考：深度/宽度优先搜索属于A*算法吗？

A* 可采纳性

- 可采纳性(admissible)
  - 最优评估函数一个算法是可采纳的，当存在一个解路径时，算法总是终止在此最优解路径上。

- A* 算法是可采纳的
  - A*可以终止；
  - 解路径上的节点总是会被A*展开(访问到)；
  - 当存在解路径时， A*算法终止在此解路径上。

A* 最优性

- A_1*

$$f_1(n) = g_1(n) + h_1(n)$$

- A_2*

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

  - 如果$h_1(n)<h_2(n)$，则A_2*展开的节点数小于A_1*。

A* 单调性

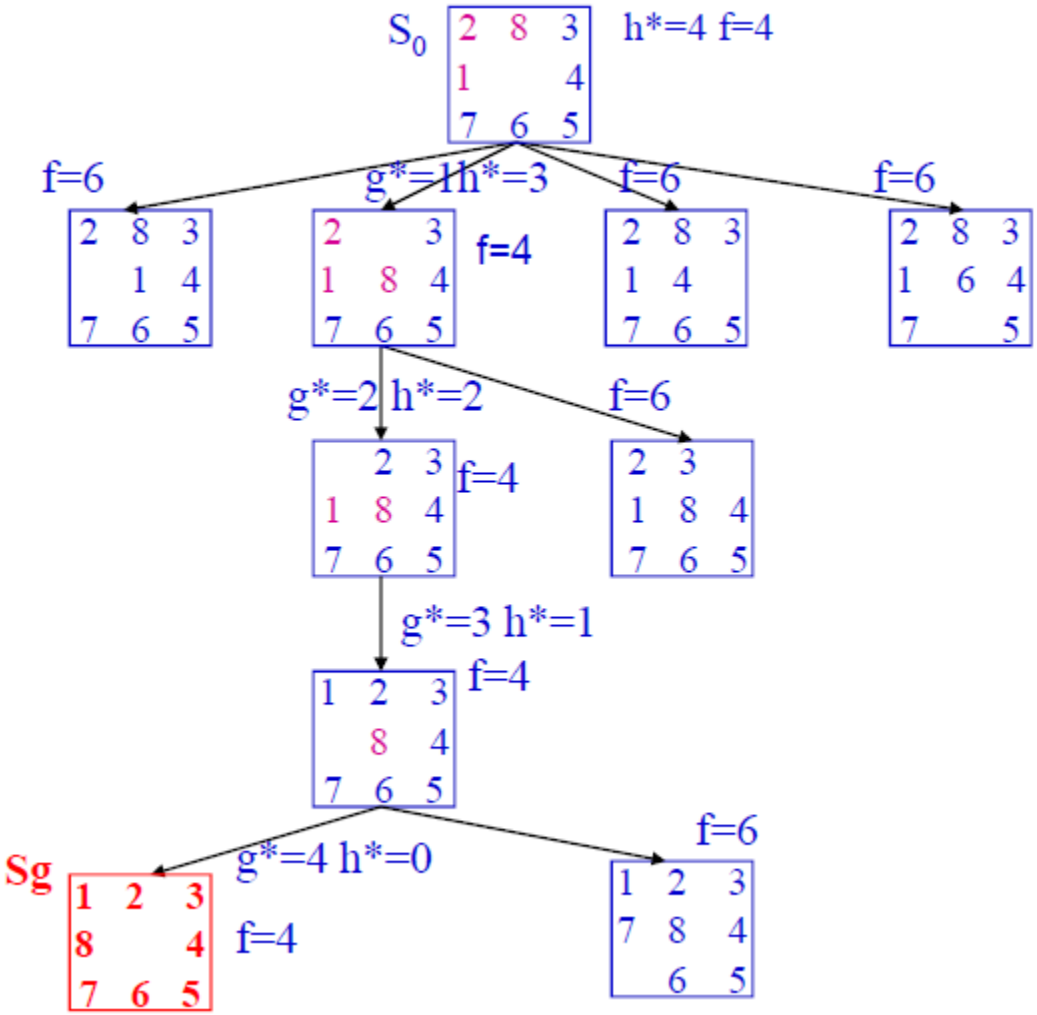
- 启发函数h是单调的
  - n_j是n_i的后继节点，如果

$$h(n_i) - h(n_j) <= cost(n_i, n_j)$$

  - $h(goal) = 0$

- 单调性：在首次访问的目标状态的路径，保证一定是最短路径。

启发诱导的搜索(8-Puzzle)



h(n)定义为错位牌到正确位置需要移动的格子数目和

搜 索

博弈树搜索

极小极大过程

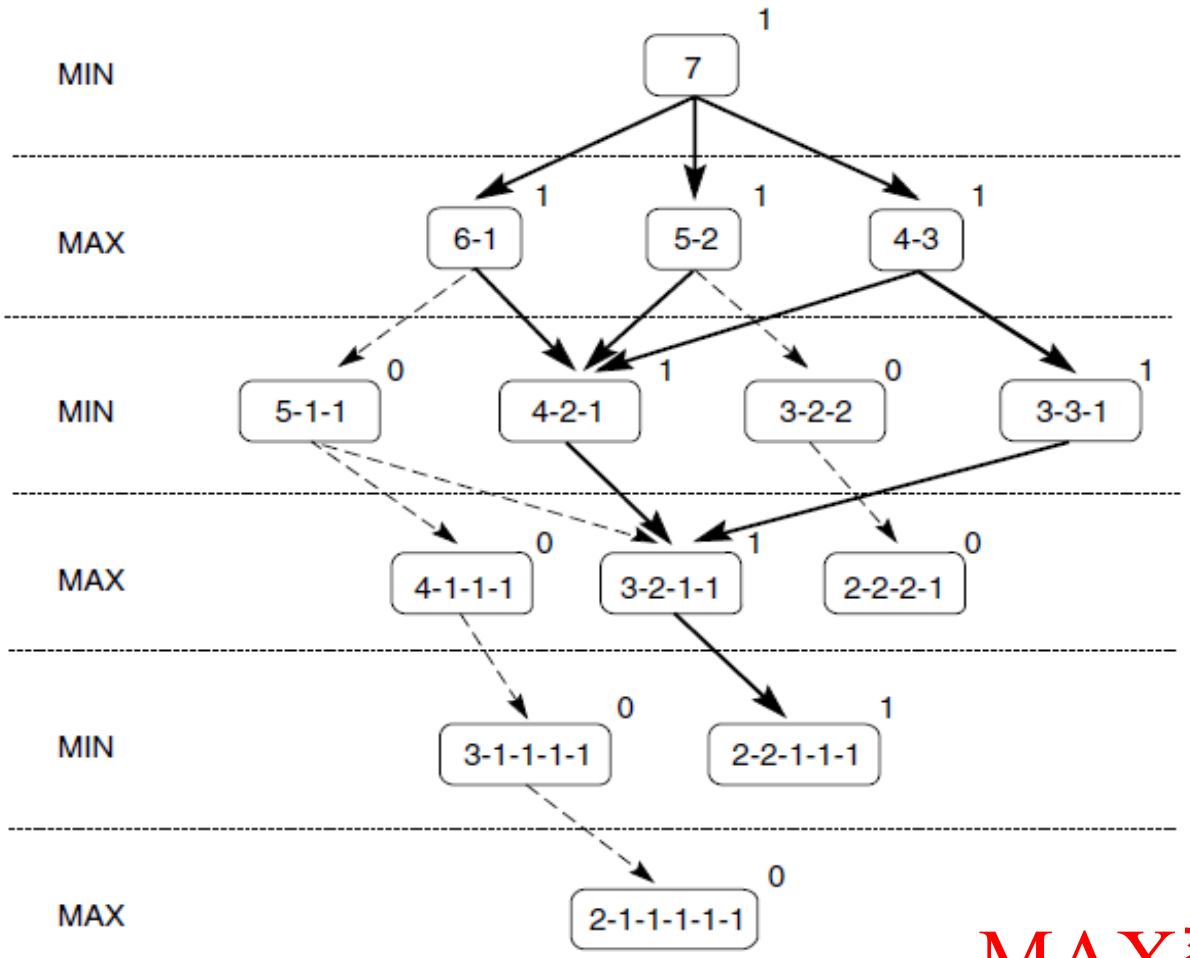
博弈：对对手行为/策略的建模

MAX：代表我方玩家，最大化其收益(赢得博弈)

MIN：代表对手，最小化MAX的收益

MIN总是移动到使MAX收益最坏的状态

余一棋



MIN走第一步

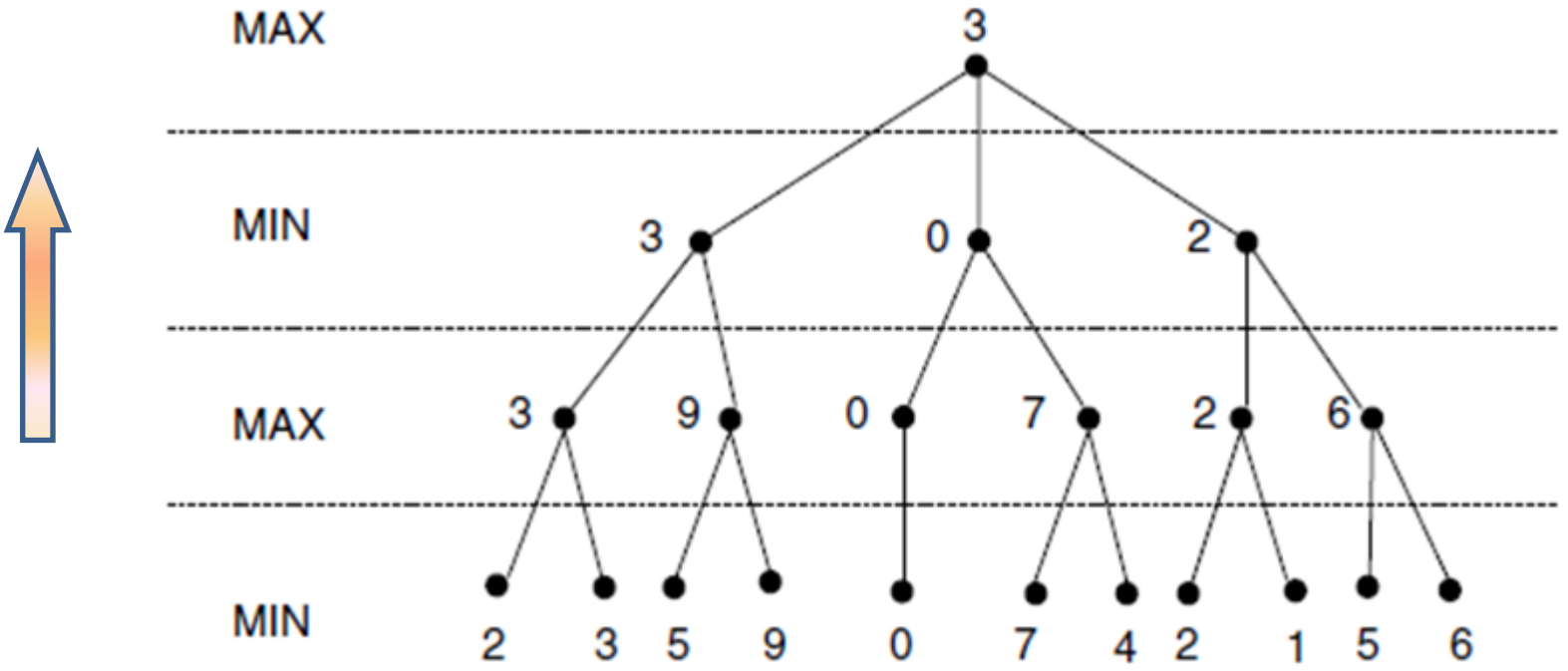
MAX走第二步

MAX获胜：值函数为1

MIN获胜：值函数为0

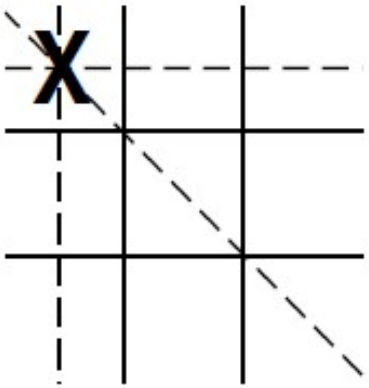
规则：将其分为两个不相等的两堆

极小极大搜索

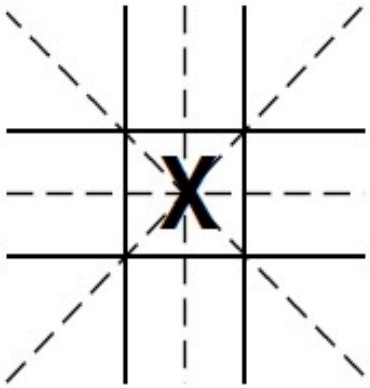


- 将启发值自底向上传播
  - ✓ 如果父状态是MAX节点，将孩子节点中最大值传给它
  - ✓ 如果父状态是MIN节点，将孩子节点中最小值传给它

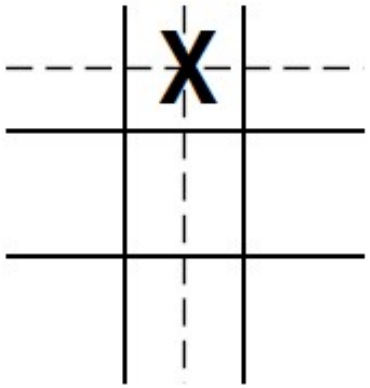
固定层深的极小极大过程



Three wins through
a corner square

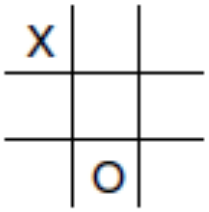


Four wins through
the center square

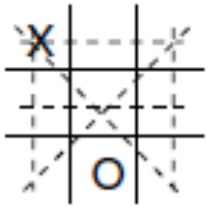


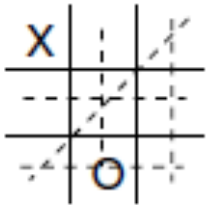
Two wins through
a side square

定义启发式函数



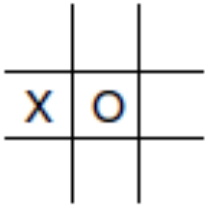
X has 6 possible win paths:





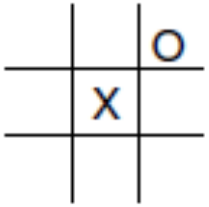
O has 5 possible wins:

E(n) = 6 – 5 = 1



X has 4 possible win paths;
O has 6 possible wins

E(n) = 4 – 6 = –2

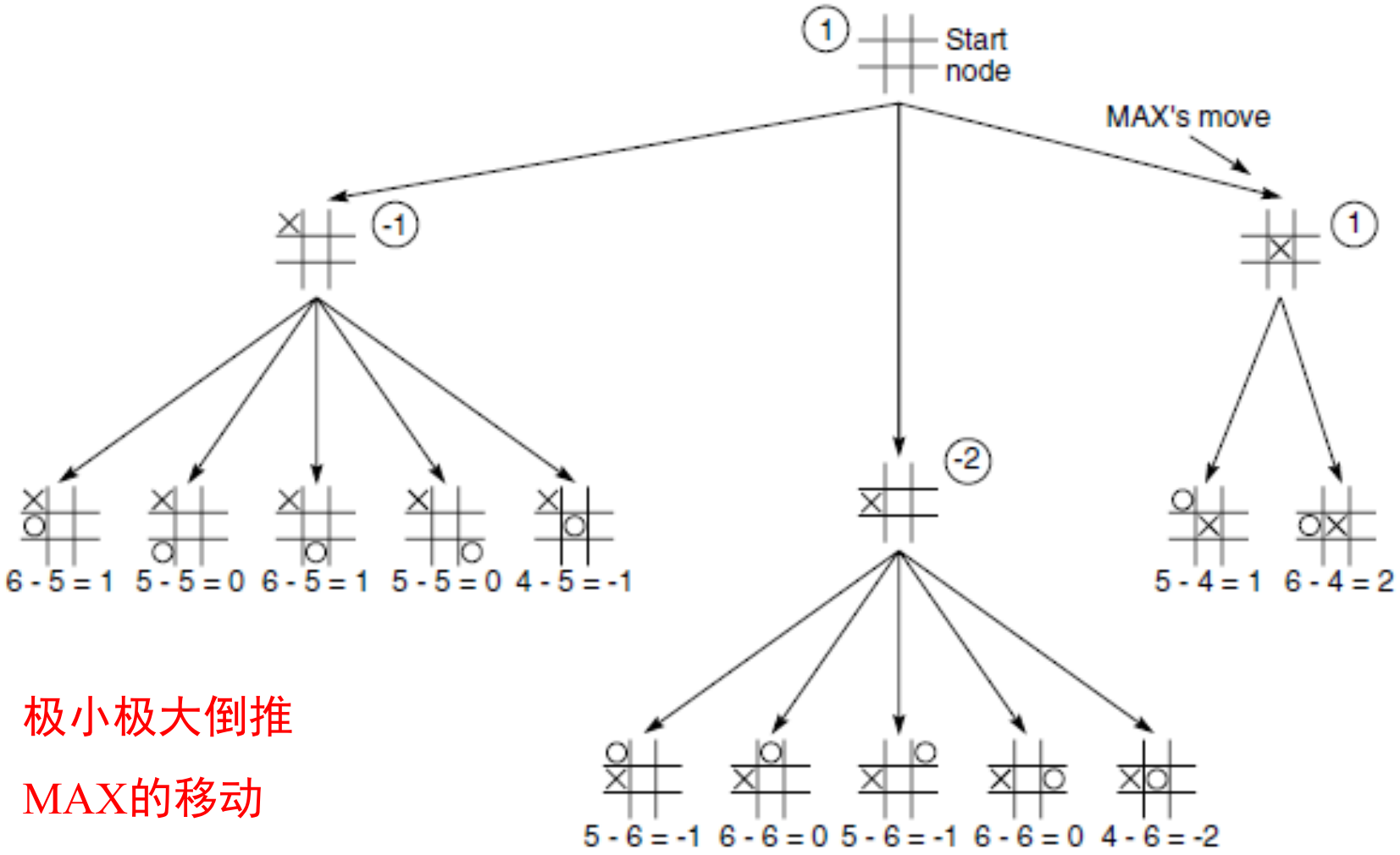


X has 5 possible win paths;
O has 4 possible wins

E(n) = 5 – 4 = 1

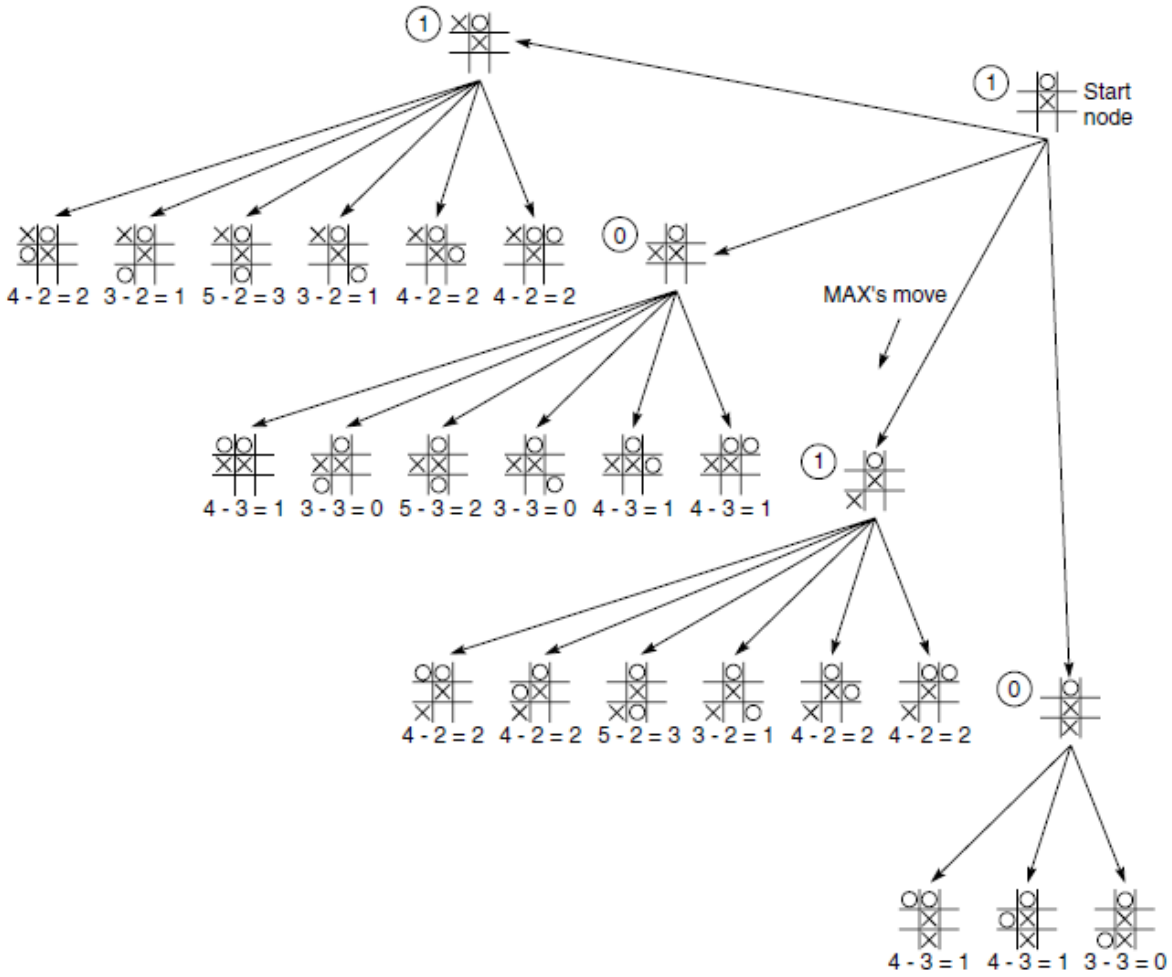
- 启发式评估：E(n)=M(n)-O(n)
  - ✓ M(n)是当前玩家可能获胜的行数
  - ✓ O(n)是对手可能获胜的行数

例：MAX开局移动

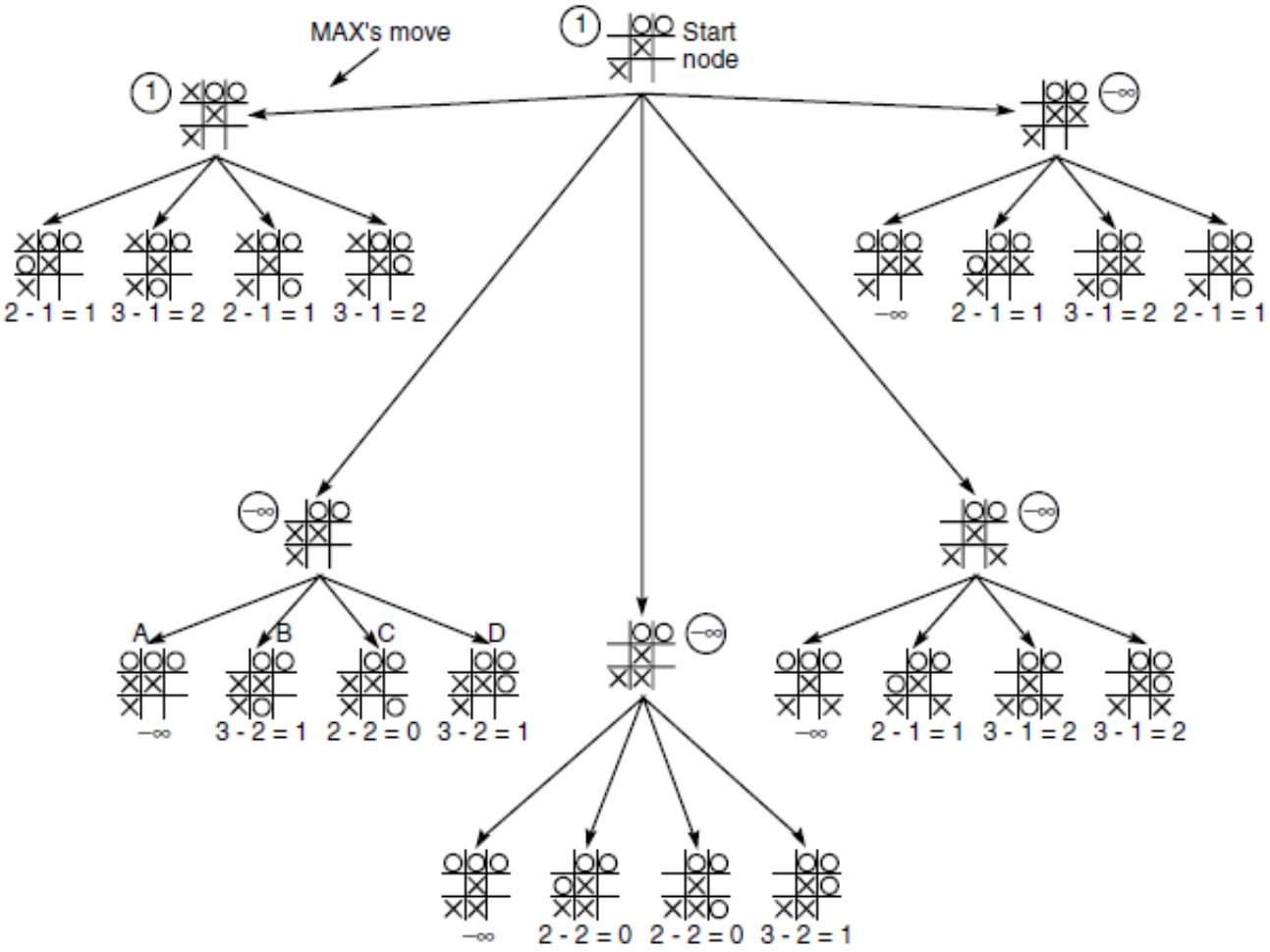


- 极小极大倒推
- MAX的移动

例：MAX第二步移动


1
Start node
4 - 2 = 2
3 - 2 = 1
5 - 2 = 3
3 - 2 = 1
4 - 2 = 2
4 - 2 = 2
0
MAX's move
4 - 3 = 1
3 - 3 = 0
5 - 3 = 2
3 - 3 = 0
4 - 3 = 1
4 - 3 = 1
1
4 - 2 = 2
4 - 2 = 2
5 - 2 = 3
3 - 2 = 1
4 - 2 = 2
4 - 2 = 2
0
4 - 3 = 1
4 - 3 = 1
3 - 3 = 0

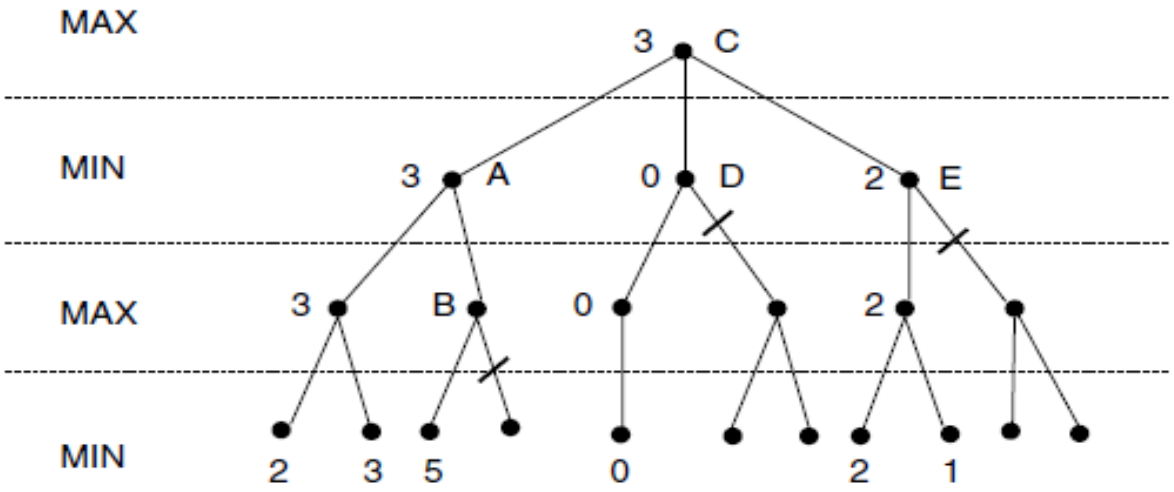
例：MAX第三步移动


MAX's move
1
Start node
1
−∞
2 - 1 = 1
3 - 1 = 2
2 - 1 = 1
3 - 1 = 2
−∞
2 - 1 = 1
3 - 1 = 2
2 - 1 = 1
−∞
−∞
A
B
C
D
−∞
3 - 2 = 1
2 - 2 = 0
3 - 2 = 1
−∞
−∞
2 - 1 = 1
3 - 1 = 2
3 - 1 = 2
−∞
2 - 2 = 0
2 - 2 = 0
3 - 2 = 1

α-β过程

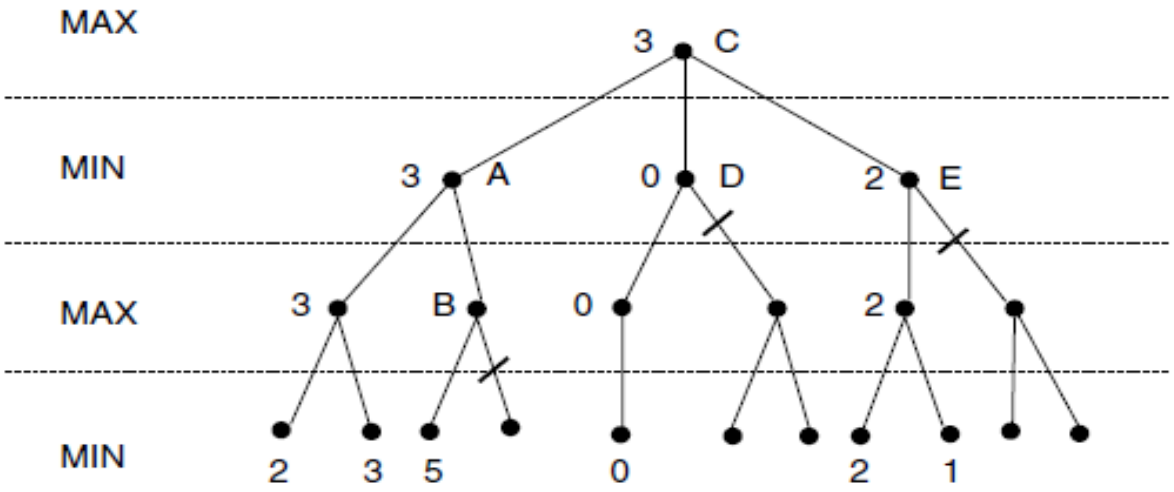
- 极小极大过程
  - ✓ 在预判层应用启发式评估
  - ✓ 展开所有的后继分支
  - ✓ 沿树向上传播评估值
- α-β剪枝
  - ✓ 当确定是一个dead end时，停止展开其后继节点
  - ✓ 对博弈树的深度优先搜索，且维护
    - ✓ Alpha: 与MAX节点关联，从不减小
    - ✓ Beta: 与MIN节点关联，从不增大

α-β过程



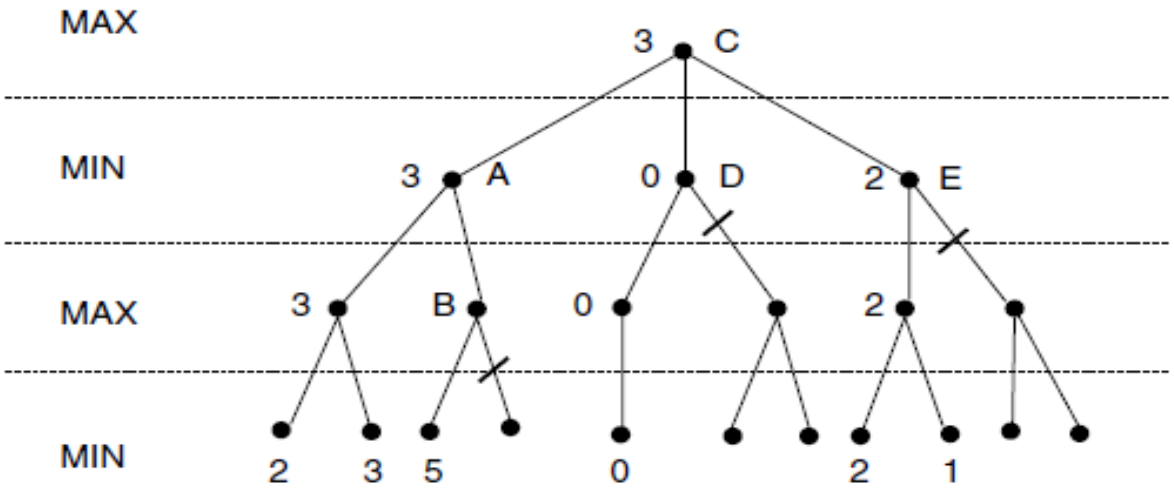
A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since 5 > 3
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

α-β过程



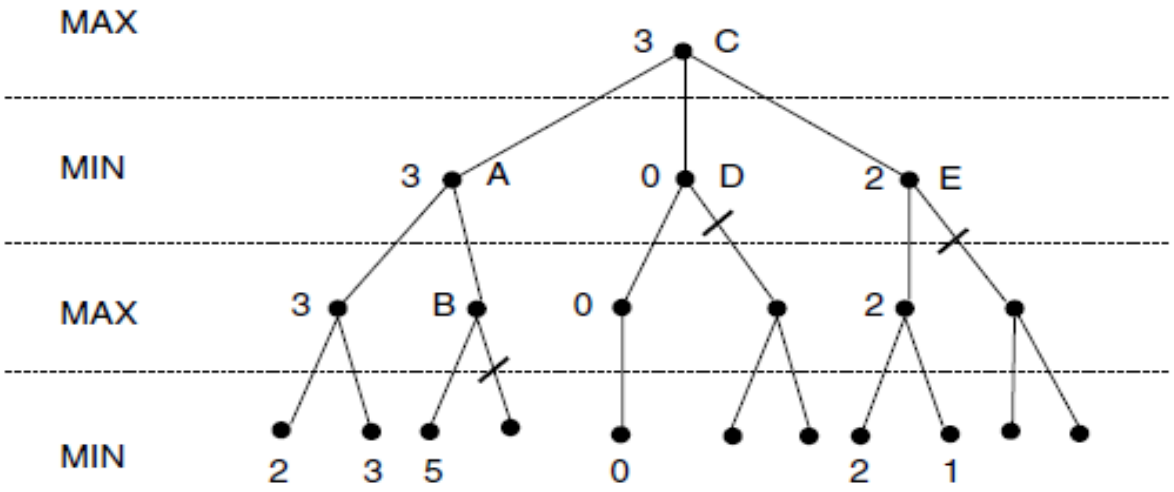
A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since 5 > 3
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

α-β过程



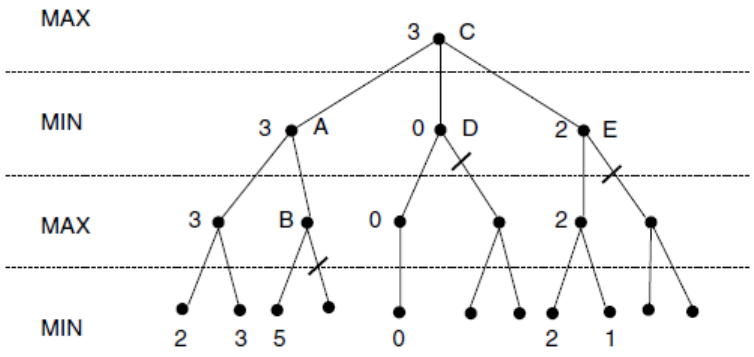
A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since 5 > 3
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

α-β过程



A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since 5 > 3
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

剪枝规则

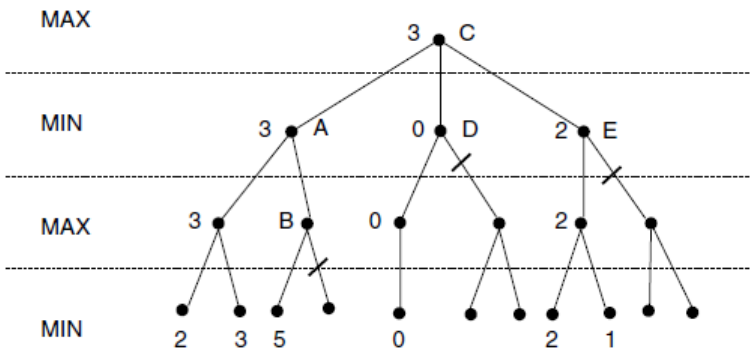


A has β = 3 (A will be no larger than 3)
B is β pruned, since 5 > 3
C has α = 3 (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

剪枝规则

- ✓ Alpha剪枝: 任一MIN节点，如果其Beta值小于等于其祖先MAX节点的Alpha值，则停止搜索
- ✓ Beta剪枝: 任一MAX节点，如果其Alpha值大于等于其祖先MIN节点的Beta值，则停止搜索

博弈树搜索



A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since 5 > 3
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

- α-β剪枝
  - ✓ 对子节点排序非常敏感
  - ✓ 静态启发：吃子启发
  - ✓ 动态启发：历史启发、杀手启发、置换表启发

查阅文献，了解中国象棋的博弈树搜索策略

搜 索

Monte Caro 树搜索

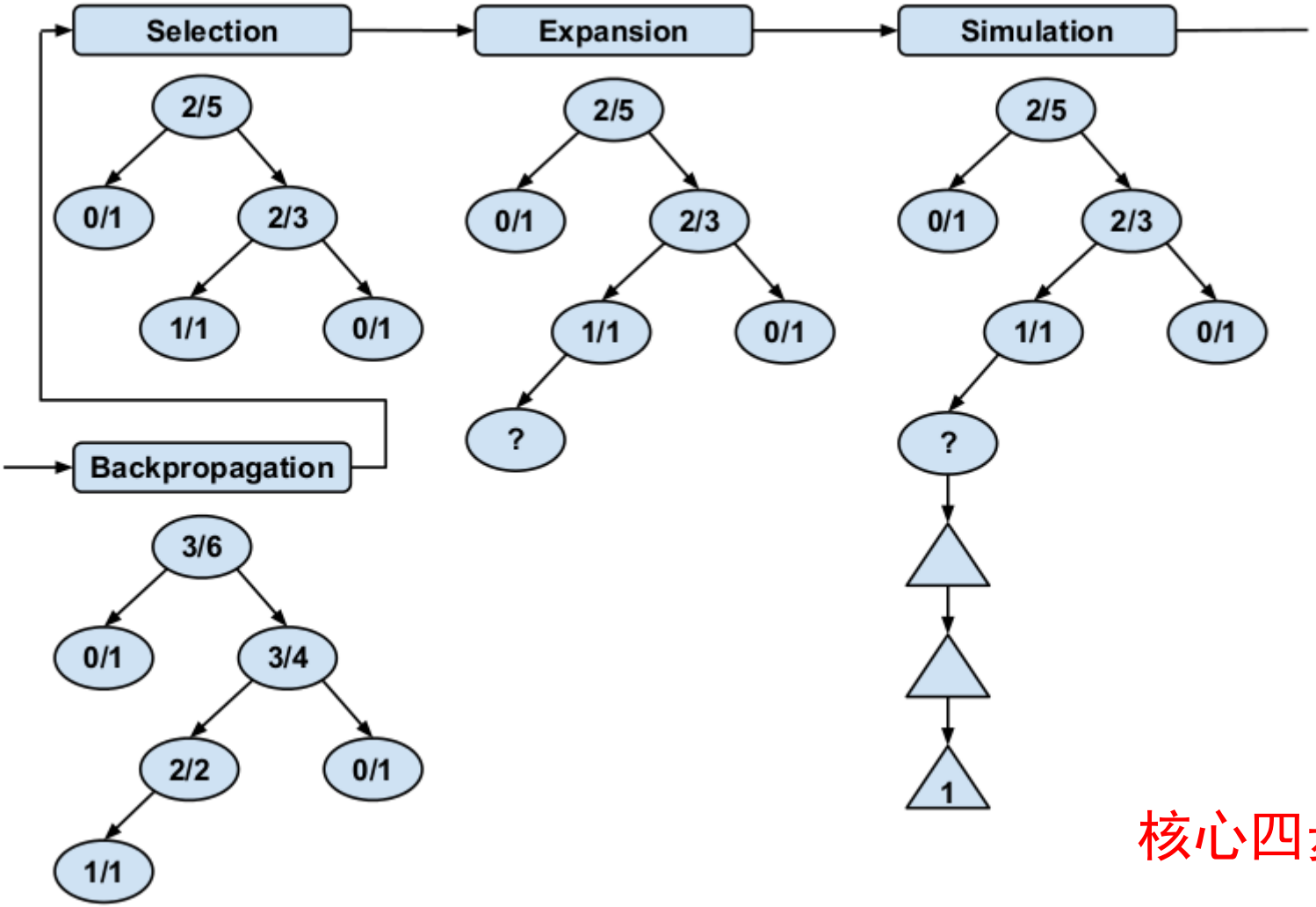
蒙特卡洛树搜索

基本原理：随机抽样+假设检验+树搜索

解决：迭代产生游戏树，解决树空间太大的问题

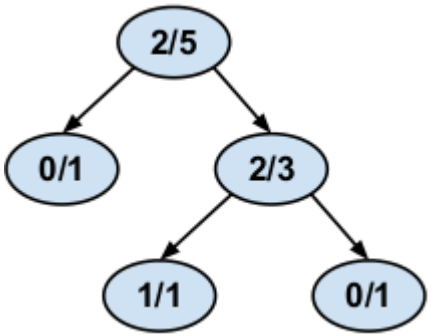
性质：用概率的方式，产生合理的推理，但并不保证一定最优

蒙特卡洛树搜索



核心四步

蒙特卡洛树的表示



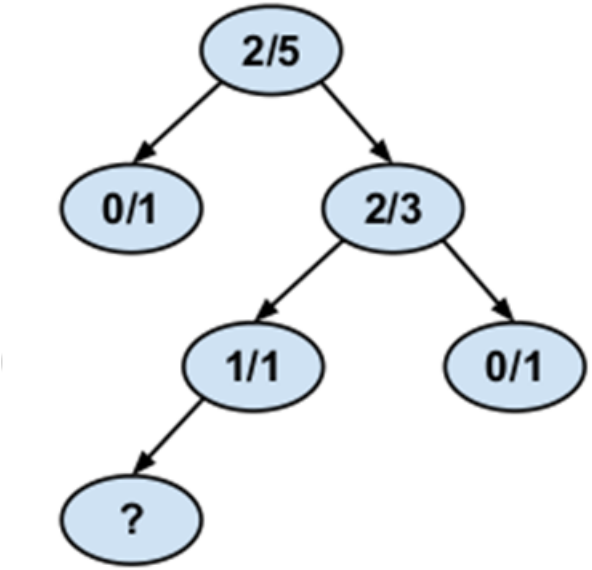
预测值(estimated value)：

第1个数字：代表在这个子树上赢的次数

第2个数字：在这个子树上模拟的次数

SELECTION步骤：

利用树策略选择节点



蒙特卡洛树的选择策略

UCT策略：

平衡Exploration和Exploitation。

Exploration approach 促使去探索尚未发现的树的其他领域。这会将倾向于探索树的广度，而不是深度。

Exploitation approach 倾向于选择拥有最大预测值的路径。这种是属于贪心算法，趋于探索树的深度。

$$UCT(node) = \frac{W(node)}{N(node)} + \sqrt[c]{\frac{ln(N(parentNode))}{N(node)}}$$

蒙特卡洛树的其他三步

Expansion：

添加一个 “？”的叶子节点。这是每次迭代中唯一添加的节点。

Simulation：

又称 playout 或者 rollout。执行操作，直到达到结束状态，或者满足设定的阈值，就停止该操作。然后基于模拟的结果，建立新添加节点的值。

Backpropagation：

利用新添加节点的值，对之前的树进行更新。从新的节点开始，算法反向遍历回到根节点。

思考和讨论

1. 状态空间表示与其他问题表示的区别？
2. 状态空间图都能转为搜索树吗？
3. DATA-DRIVEN与GOAL-DRIVEN的区别？
4. 爬山算法与最佳优先搜索算法的区别？
5. 最佳优先搜索的本质？
6. 什么情况下启发式函数f(n)=g(n)+h(n), 需要加上？
7. 了解AlphaGo里面的蒙特卡洛树搜索。

实验

1. 九宫图。将1~9九个数字分别填到3*3中，使其每一横\竖\斜都等于15。分别用深度优先搜索、宽度优先搜索和启发式搜索实现。分析并讨论三种搜索方法的效率，以及启发式函数的设计。

要求：可以用任何语言，需要有日志说明搜索路径，实验报告分析效率和启发式函数设计。10月10日前发给助教，检查。

作业提交到公共邮箱: nju_course_ai_2019@163.com

邮件主题格式：姓名_学号_第X次实验

谢 谢 ！