人工智能概论

慕彩红
人工智能学院
mucaihongxd@foxmail.com

第二章 状态空间表示与问题求解

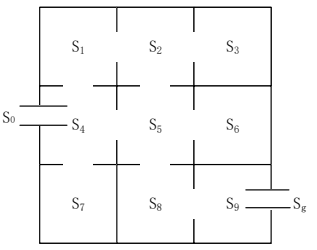
第二章 状态空间表示与问题求解

- 内容提要：
- 2.1 问题求解与状态空间表示法
- 2.2 图搜索策略
- 2.3 盲目式搜索
- 2.4 启发式搜索

引言

- 人工智能的多个研究领域从求解现实问题的过程来看，都可抽象为一个“问题求解”过程，问题求解过程实际上就是一个搜索过程。
- 在搜索过程开始之前，必须先用某种方法或某几种方法的混和来表示问题。
- 问题求解技术主要涉及两个方面：
  - **问题的表示**
  - **求解的方法**
- **状态空间表示法就是用来表示问题及其搜索过程的一种方法。**

- 例：迷宫问题



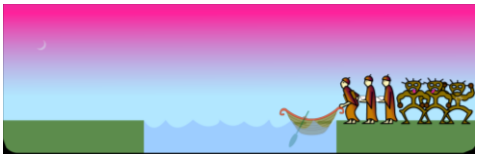
- 约束：每次只能走一步

- 例：八数码问题

初始棋局　　目标棋局

- 例：传教士野人问题
（Missionaries& Cannibals，MC问题）

有三个传教士M和三个野人C过河，只有一条能装下两个人的船，在河的一方或者船上，如果野人的人数大于传教士的人数，那么传教士就会有危险，你能不能提出一种安全的渡河方法呢？



- 调度
- 导航
- 分配
- 路径规划
- 游戏

问题明确了，可如何如何进行问题表示，才能让计算机知道这些问题呢？

问题的状态空间表示

- 状态空间方法：基于解答空间的问题表示和求解方法，它是以状态和算符为基础来表示和求解问题的。
- 主要包括：
  - 状态
  - 算符
  - 状态空间
- 典型的例子：下棋、迷宫、各种游戏等。

问题的状态空间表示

- **状态（state）：**为描述某类不同事物间的差别而引入的一组最少变量 q_0，q_1，…，q_n的有序集合.

矢量形式：$Q=[q_0, q_1, \cdots q_n]$

式中每个元素q_i为集合的分量，称为状态变量。

给定每个分量的一组值就得到一个具体的状态，

如 $Q_k=[q_{0k}, q_{1k}, \cdots, q_{nk}]$

问题的状态空间表示

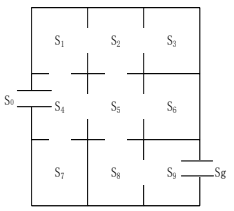
- **迷宫问题的状态：**

- 以每个格子作为一个状态,并用其标识符作为其表示。

该迷宫的状态可表示为：$S_k=\{S_i\}$

S_i可以取值为$\{S_0, S_1, S_2, S_3, \ldots, S_g\}$

- 初始状态：S_0
- 目标状态：S_g



问题的状态空间表示

- 八数码问题的状态：

- 将棋局

1		3
7	2	4
6	8	5

⇒

1	2	3
8		4
7	6	5

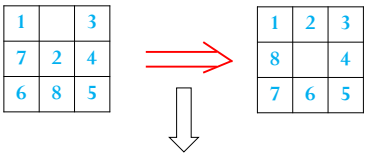
X_1	X_2	X_3
X_8	X_0	X_4
X_7	X_6	X_5

用向量$S=(X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8)$表示状态

X_i为变量，X_i的值就是方格X_i内的数字，取值为$\{0,1,2\ldots8\}$，其中0表示空格。于是，向量S就是该问题的状态表达式。

问题的状态空间表示

- 八数码问题的状态:



初始状态:S_0=(210345867)

目标状态:S_g=(012345678)

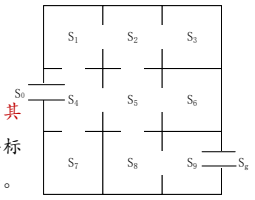
问题的状态空间表示

➢ 算符（Operator）：当对一个问题状态使用某个可用操作时，它将引起该状态中某些分量值的变化，从而使问题从一个具体状态变为另一个具体状态。

- 算符可理解为状态集合上的一个函数，它描述了状态之间的关系。
- 算符可以是某种操作、规则、 行为、变换、函数、算子、过程等。
- 算符也称为操作,问题的状态也只能经定义在其上的这种操作而改变。

问题的状态空间表示

- 迷宫问题的算符



➢ 以每个格子作为一个状态,并用其标识符作为其表示。那么,两个标识符组成的序对就是一个算符。

于是:

- S: S_0
- 该迷宫的算符 F: F={(S_0, S_4), (S_4, S_0), (S_4, S_1), (S_1, S_4), (S_1,S_2), (S_2, S_1), (S_2, S_3), (S_3, S_2), (S_4, S_7), (S_7, S_4), (S_4, S_5), (S_5, S_4), (S_5,S_2), (S_2, S_5), (S_5, S_6), (S_6, S_5), (S_5, S_8), (S_8, S_5), (S_8, S_9), (S_9, S_8), (S_9, S_g)}
- G: S_g

- 八数码难题的算符

- 制定操作算符集:
- * 直观方法——为每个棋牌制定一套可能的走步：左、上、右、下四种移动。这样就需32个操作算子。
- * 简易方法——仅为空格制定这4种走步，因为只有紧靠空格的棋牌才能移动。
- * 空格移动的唯一约束是不能移出棋盘。

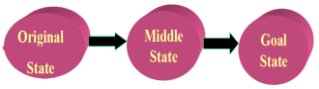
16

问题的状态空间表示

➢ 问题的状态空间（State space）：利用状态变量和算符表示系统或问题的有关知识的符号体系。

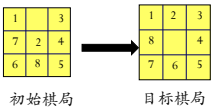
➢ 状态空间常用一个四元组表示：（S，O，S_0，G）

- S：为状态集合；
- O：为状态转换规则集合（操作的集合）；
- S_0：为问题的所有初始状态的集合；
- G：为目标状态的集合。



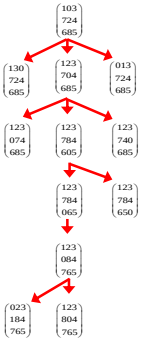
问题的状态空间表示

- 状态空间图



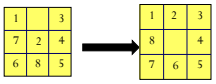
初始棋局　　目标棋局

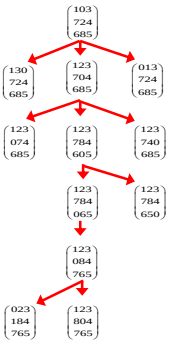
➢ **从初始棋局开始，试探由每一合法走步得到的各种新棋局，然后再走一步而得到的下一组棋局。这样继续下去，直至达到目标棋局为止。**



问题的状态空间表示

- 状态空间图



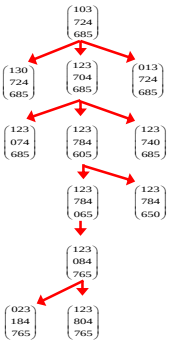


➢ 把初始状态可达到的各状态所组成的空间设想为一幅由各种状态对应的节点组成的图。这种图称为状态空间图。图中每个节点是它所代表的棋局。首先把适用的算符用于初始状态，以产生新的状态；算符可以看成状态空间图的边。

问题的状态空间表示

➢ 状态图示法

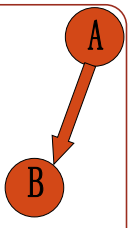
状态空间的图示形式称为状态空间图。其中图的节点表示状态，图的边表示算符。



状态图示法

有向图(directed graph)

图，由节点（不一定是有限的节点）的集合构成。有向图，是指图中的一对节点用弧线连接起来，从一个节点指向另一个节点。



- 路径
- 某个节点序列（$n_{i1}, n_{i2}, \ldots, n_{ik}$）当j=2, 3, …k时，如果对于每一个$n_{ij-1}$都有一个后继节点$n_{ij}$存在，那么就把这个节点序列叫做从节点$n_{i1}$至节点$n_{ik}$的长度为k的路径。

代价 加在各弧线的指定数值，以表示加在相应算符上的代价。

寻找从一种状态变换为另一种状态的某个算符序列问题就等价于寻求图的某一路径的问题.

传教士野人问题（Missionaries& Cannibals问题）

- 状态：问题在某一时刻所处的“位置”，“情况”等
- 根据问题所关心的因素，一般用向量形式表示，每一位表示一个因素

（考虑从左岸向右岸渡河）

状态可有多种表示方法：

(左岸传教士数,右岸传教士数,左岸野人数,右岸野人数,船的位置)

或

(右岸传教士数,右岸野人数,船的位置)

初始状态：(0, 0, 0)

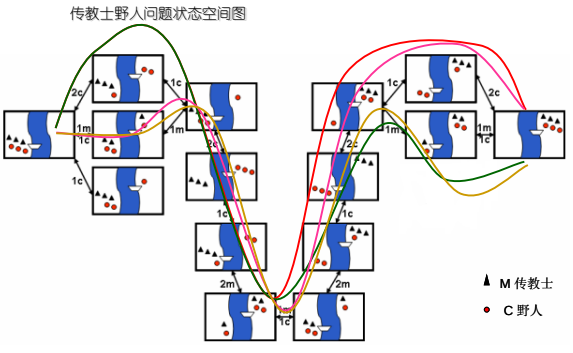
目标状态：(3, 3, 1)

0：左岸
1：右岸

哪些操作能导致状态变化？

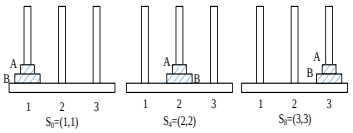
状态的转换

- **算符（操作）**——使状态发生改变的操作
- MC问题中的算符：将传教士或野人运到河对岸
  - Move-1m1c-lr：将一个传教士(m)一个野人(c)从左岸(l)运到右岸(r)
  - 所有可能操作
    - Move-1m1c-lr Move-1m1c-rl
    - Move-2c-lr Move-2c-rl
    - Move-2m-lr Move-2m-rl
    - Move-1c-lr Move-1c-rl
    - Move-1m-lr Move-1m-rl



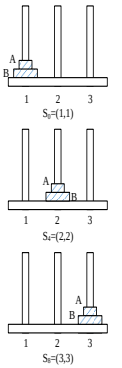
二阶梵塔难题

- 问题描述：设有三根柱子，它们的编号分别是1号、2号和3号。在初始情况下，1号柱子上穿有A、B两个盘子，A比B小，A位于B的上面。
  - 要求把这两个盘子全部移到另一根柱子上，而且规定每次只能移动一个盘子，任何时刻都不能使大的位于小的上面。
  - 如何将A、B移到2号或3号柱子上面?



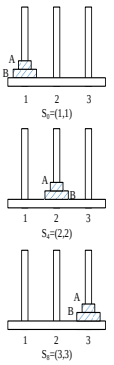
二阶梵塔难题--状态

- 状态：设$S_K=(S_{K0},S_{K1})$表示问题的状态，
  - S_{K0} 表示盘片A所在的柱号，取值1, 2, 3
  - S_{K1} 表示盘片B所在的柱号，取值1, 2, 3
- 全部可能的状态，共有以下9种:
  - S_0=(1,1), S_1=(1,2), S_2=(1,3),
  - S_3=(2,1), S_4=(2,2), S_5=(2,3),
  - S_6=(3,1), S_7=(3,2), S_8=(3,3).
- 问题的初始状态集合S={S_0}，目标集合为G={S_4,S_8}



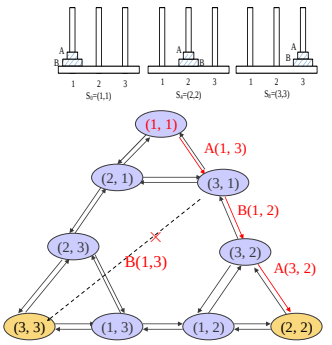
二阶梵塔难题--算符

- 算符： A(i,j), B(i,j)
  - A(i,j): A盘从柱i移到柱j;
  - B(i,j): B盘从柱i移到柱j
- 全部可能的算符，共有12种:
  - 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)



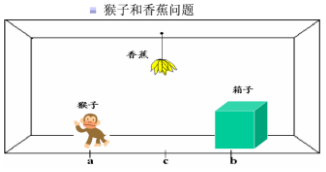
二阶梵塔难题--状态空间图

- 状态空间图
  - 从初始节点(1, 1)到目标节点(2, 2)及(3, 3)的任何一条路径都是问题的一个解。
  - 最短的路径长度是3，它由3个操作组成。
  - 如从(1, 1)开始，通过使用操作A(1, 3)、B(1, 2)及A(3, 2)，可到达(2, 2)。



二阶梵塔难题的状态空间图

猴子和香蕉问题

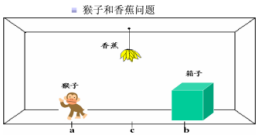


问题描述
在一个房间内有一只猴子，一个箱子和一束香蕉。香蕉挂在天花板下方，但猴子的高度不足以碰到它。那么这只猴子怎样才能摘到香蕉呢?

29

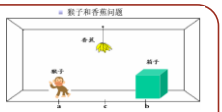
解题过程

- 用一个四元表列（W，X，Y，Z）来表示问题状态.
- 其中：W-猴子的水平位置；X－当猴子在箱子顶上时取X=1；否则取X=0；Y－箱子的水平位置；Z-当猴子摘到香蕉时取Z=1；否则取Z=0。



30

操作（算符）：



1. goto（U）表示猴子走到水平位置U或者用产生式规则表示为：

   (W，0，Y，Z) —goto (U)→ (U，0，Y，Z)

2. pushbox（V）猴子把箱子推到水平位置V，即有：

   (W，0，W，Z) —pushbox(V)→ (V，0，V，Z)

3. climbbox 猴子爬上箱顶，即有：

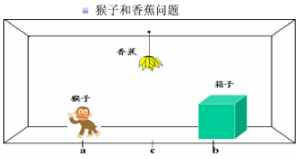
   (W，0，W，Z) —climbbox→ (W，1，W，Z)

4. grasp 猴子摘到香蕉，即有：

   (c，1，c，0) —grasp→ (c，1，c，1)

31

- 该初始状态变换为目标状态的操作序列为：
- {goto(b)，pushbox(c)，climbbox，grasp}



初始状态：

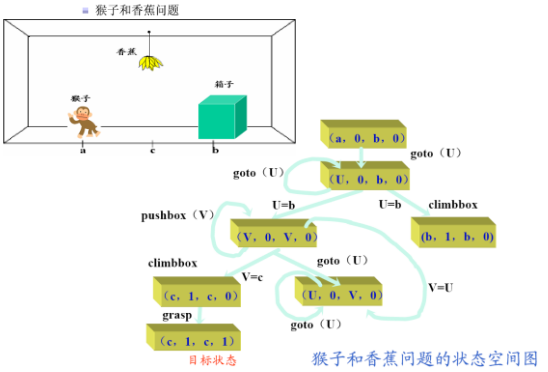
(a，0，b，0)

目标状态：

(c，1，c，1)

猴子和香蕉问题的演示过程

32



猴子和香蕉问题的状态空间图

33

- **推销员旅行问题（旅行商问题）**
-
- 一个推销员计划出访推销产品。他从一个城市（如A）出发，访问每个城市一次，且最多一次，然后返回城市A。要求寻找最短路线。

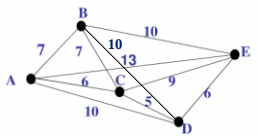
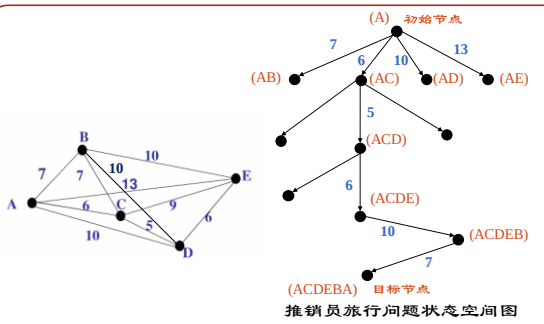


图2.3 推销员旅行问题

- 状态描述：目前为止访问过的城市列表（A...）
- 初始状态：（A）
- 目标状态：（A.....A）



推销员旅行问题状态空间图

- 算符：下一步走向的城市(a) (b) (c) (d) (e)
- 约束：每个城市只能走过一次，A除外

小结

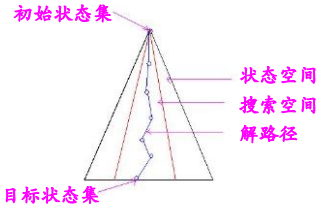
状态空间表示
- 初始状态
- 算符
- 目标状态

第二章 状态空间表示与问题求解

- 2.1 问题求解与状态空间表示法
- 2.2 图搜索策略
- 2.3 盲目式搜索
- 2.4 启发式搜索

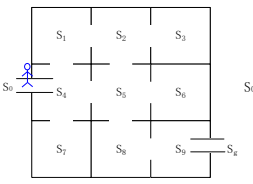
引言

- 问题表示是问题求解所必须的，从问题表示到问题的解决，有一个求解过程,也就是搜索过程。
- 状态空间法的求解过程等价于在状态空间图中搜索一条从初始节点到目标节点的路径问题。

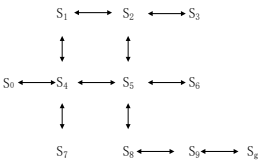


状态图搜索

- 走迷宫其实就是从该有向图的初始节点(入口)出发，寻找通向目标节点(出口)的路径的问题。



9格子迷宫



迷宫的状态空间图

状态图搜索

由于搜索的目的是为了寻找初始节点到目标节点的路径，所以在搜索过程中就得随时记录搜索轨迹。

- 必须记住下一步还可以走哪些点

  OPEN表(记录待扩展的点)

- 必须记住哪些点走过了

  CLOSED表(记录已扩展的点)

- 必须记住从目标返回的路径

  每个表示状态的节点结构中必须有指向父节点的指针

所谓对一个节点进行“扩展”是指对该节点用某个可用算符进行作用，生成该节点的一组子节点（后继节点）

状态图搜索

记录当前待考察的节点

记录考察过的节点

(1) OPEN表与CLOSED表

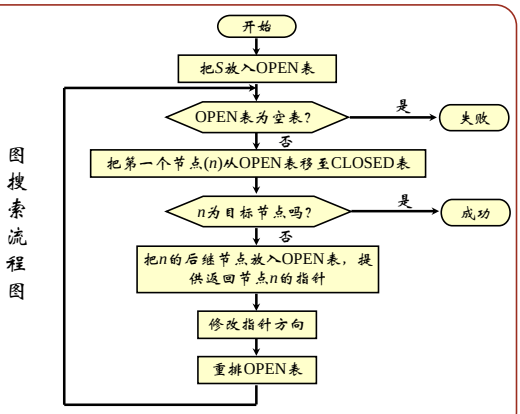
(2) 搜索（状态）图与搜索树

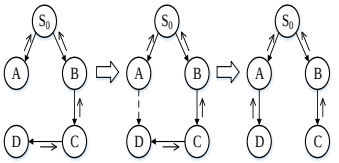
OPEN

节点	父节点编号

CLOSED

编号	节点	父节点编号





修改返回指针示例

进一步说明：

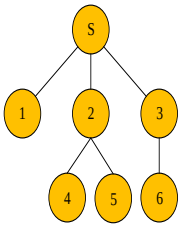
(1) 修改返回指针的原因是，因为这些节点又被第二次生成，所以它们返回初始节点的路径已有两条，但这两条路径的“长度”可能不同。 那么，当新路短时自然要走新路。

(2)这里对路径的长短是按路径上的节点数来衡量的，后面我们将会看到路径的长短也可以其“代价”(如距离、费用、 时间等)衡量。

图搜索过程

- 注：这是一个通用的搜索过程，后面讨论的状态空间各种搜索策略都是其特例。
- 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。

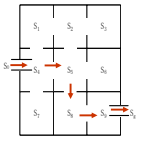
图搜索--例一



Open 表	Closed 表
{S}	{}

图搜索--例二

- 9格子迷宫



Open 表	Closed 表
$\{S_0\}$	{}
$\{S_4\}$	$\{S_0\}$
$\{S_1,S_5,S_7\}$	$\{S_0,S_4\}$
$\{S_5,S_1,S_7\}$	$\{S_0,S_4\}$
$\{S_1,S_7\}$	$\{S_0,S_4,S_5\}$
$\{S_1,S_7,S_8,S_6,S_2\}$	$\{S_0,S_4,S_5\}$
$\{S_8,S_6,S_2,S_1,S_7\}$	$\{S_0,S_4,S_5\}$
$\{S_6,S_2,S_1,S_7,S_9\}$	$\{S_0,S_4,S_5,S_8\}$
$\{S_9,S_6,S_2,S_1,S_7\}$	$\{S_0,S_4,S_5,S_8\}$
$\{S_6,S_2,S_1,S_7,S_g\}$	$\{S_0,S_4,S_5,S_8,S_9\}$
$\{S_g,S_6,S_2,S_1,S_7\}$	$\{S_0,S_4,S_5,S_8,S_9\}$
$\{S_6,S_2,S_1,S_7\}$	$\{S_0,S_4,S_5,S_8,S_9,S_g\}$

$S_0 \rightarrow S_4 \rightarrow S_5 \rightarrow S_8 \rightarrow S_9 \rightarrow S_g$

图搜索分类

- 对OPEN表中节点排序方式产生了不同的搜索策略，不同的搜索搜索策略效率不同。
- 这种排序可以是任意的即盲目的(属于盲目搜索)，也可以用各种启发思想或其它准则为依据(属于启发式搜索)。

图搜索策略

- 无信息搜索（盲目式搜索）
  - 宽度优先搜索
  - 深度优先搜索
  - 等代价搜索
- 有信息搜索（启发式搜索）
  - A算法
  - A*算法

第二章 状态空间表示与问题求解

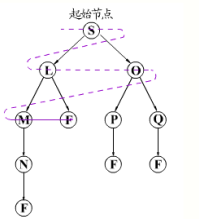
- 2.1 问题求解与状态空间表示法
- 2.2 图搜索策略
- 2.3 盲目式搜索
- 2.4 启发式搜索

盲目搜索

- 概念
  - ◇没有启发信息的一种搜索形式(**按预定的控制策略进行搜索，在搜索过程中获得的中间信息不用来改进控制策略**)
  - ◇一般只适用于求解比较简单的问题
- 特点
  - ◇不需重排OPEN表
- 种类
  - ◇宽度优先
  - ◇深度优先
  - ◇等代价搜索

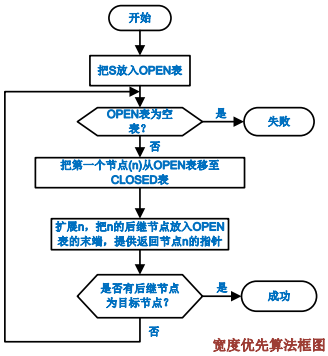
宽度（广度）优先搜索策略

- 宽度优先搜索(breadth-first search)的定义：如果搜索是以接近起始节点的程度依次扩展节点的，那么这种搜索就叫做宽度优先搜索(breadth-first search).
- 这种搜索是逐层进行的；在对下一层的任一节点进行搜索之前，必须搜索完本层的所有节点



宽度优先搜索示意图

宽度优先搜索策略框图



宽度优先算法框图

八数码难题

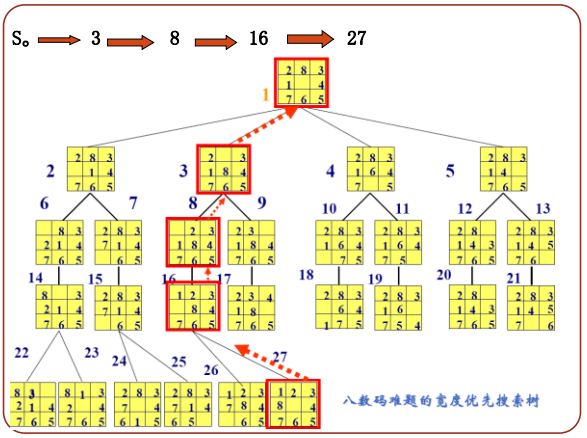
2	8	3
1		4
7	6	5

（初始状态）

1	2	3
8		4
7	6	5

（目标状态）

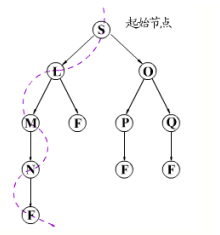
要求寻找从初始状态到目标状态的路径。



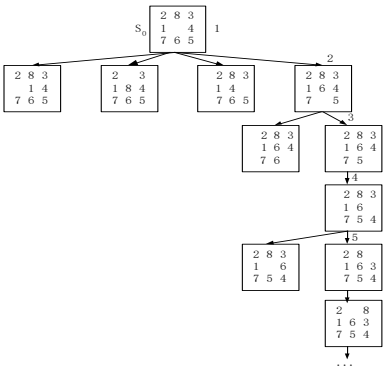
深度优先搜索

_ (depth-first search)

- 定义
  - ◇首先扩展最新产生的（即最深的）节点。
  - ◇深度相等的节点可以任意排列。
- 特点
  - ◇首先，扩展最深的节点的结果使得搜索沿着状态空间某条单一的 首先路径从起始节点向下进行下去；
  - ◇仅当搜索到达一个没有后裔的状态时，才考虑另一条替代的路径。



深度优先搜索示意图



算法

◇防止搜索过程沿着无益的路径扩展下去，往往给出一个节点扩展最大深度——深度界限。任何节点如果达到了深度界限，那么都将把它们作为没有后继节点来处理。

定义-节点的深度:

(1)起始节点的深度为0。

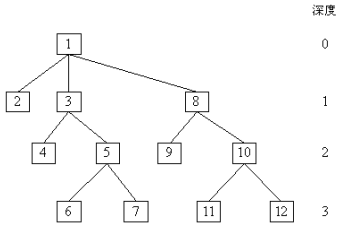
(2)任何其他节点的深度等于其父辈节点的深度加1。

◇与宽度优先搜索算法最根本的不同：将扩展的后继节点放在OPEN表的前端。

算法的具体过程

1. 把起始节点S放到未扩展节点OPEN表中。如果此节点为一目标节点，则得到一个解。
2. 如果OPEN为一空表，则失败退出。
3. 把第一个节点（节点n）从OPEN表移到CLOSED表。
4. 如果节点n的深度等于最大深度，则转向(2)
5. 扩展节点n，产生其全部后裔，并把它们放入OPEN表的前端,如果没有后裔，则转向(2)
6. 如果后继节点中有任一个为目标节点，则求得一个解，成功退出，否则，转向(2)

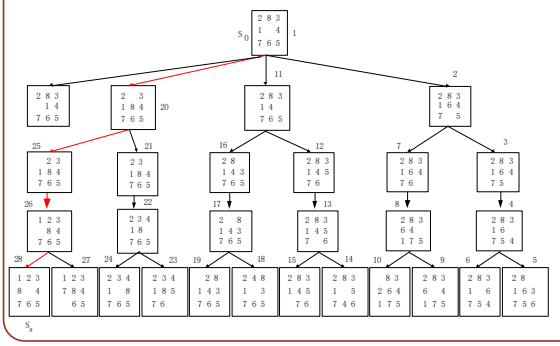
深度优先搜索基本思想



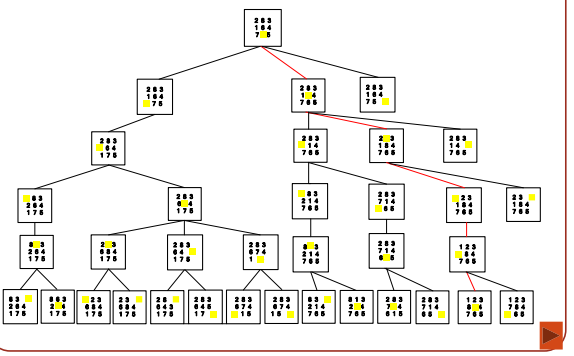
深度优先搜索的性质

- 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

例：按深度优先搜索生成的八数码难题搜索树，我们设置深度界限为4

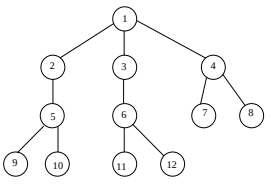


例：按深度优先搜索生成的八数码难题搜索树，我们设置深度界限为5



习题

- 1：列出图中树的节点访问序列以满足下面的两个搜索策略，并写出其搜索过程中的open和closed表（在所有情况中都选择最左分枝优先访问,设节点12为目标节点）：
- （1）深度优先搜索；
- （2）宽度优先搜索。



65

等代价搜索

- 概念
  - ◇宽度优先搜索的一种推广。
  - ◇不是沿着等长度路径断层进行扩展，而是沿着等代价路径断层进行扩展。
  - ◇搜索树中每条连接弧线上的有关代价，表示时间、距离等花费。
- 等代价搜索中的几个记号
  - ◇起始节点记为 S；
  - ◇从节点i到它的后继节点j的连接弧线代价记为 $c(i,j)$
  - ◇从起始节点S到任一节点i的路径代价记为$g(i)$。
  - ◇则对于节点i的每个后继节点j，

    $g(j)=g(i)+c(i,j)$

等代价搜索算法

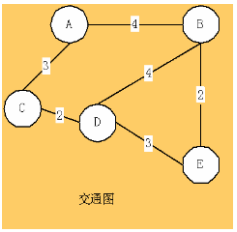
(代价树的广度优先搜索)

1. 把初始节点S_0放入OPEN表，令$g(S_0)=0$。
2. 如果OPEN表为空，则问题无解，退出。
3. 把OPEN表的第一个节点（记为节点i）取出放入CLOSED表。
4. 考察节点i是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点i不可扩展，则转第2步。
6. 扩展节点i，为每一个子节点都配置指向父节点的指针，对于节点i的每个后继节点j,计算 $g(j)=g(i)+c(i,j)$，并把所有后继节点j放进OPEN表. 按各节点的代价对OPEN表中的全部节点进行排序(按从小到大的顺序)，然后转第2步。

代价树的深度优先搜索

- 搜索过程：
  1. 把初始节点S_0放入OPEN表，令$g(S_0)=0$。
  2. 如果OPEN表为空，则问题无解，退出。
  3. 把OPEN表的第一个节点（记为节点n）取出放入CLOSED表。
  4. 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
  5. 若节点n不可扩展，则转第2步。
  6. 扩展节点n，将其子节点按代价从小到大的顺序放到OPEN表中的前端，并为每一个子节点都配置指向父节点的指针，然后转第2步。

代价树搜索举例



在代价树的宽度优先搜索中，每次都是从OPEN表的全体节点中选择一个代价最小的节点送入CLOSED表进行观察；

而代价树的深度优先搜索是从刚扩展出的子节点中选一个代价最小的节点送入CLOSED表进行观察。

盲目搜索

- 按照事先规定的路线进行搜索
  - 广度优先搜索是按“层”进行搜索的，先进入OPEN 表的节点先被考察
  - 深度优先搜索是沿着纵深方向进行搜索的，后进入OPEN表的节点先被考察
- 按已经付出的代价决定下一步要搜索的节点
  - 代价树的广度优先
  - 代价树的深度优先