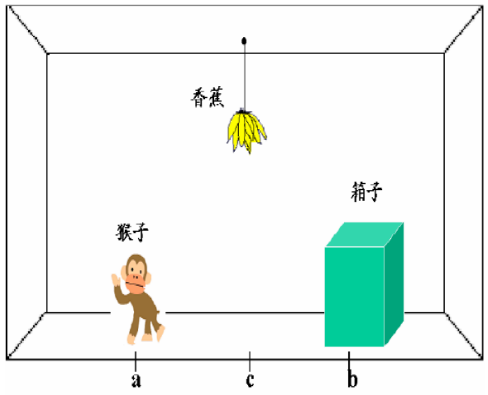
第二章 状态空间知识表示及其搜索技术

- ◆状态空间法
- ◆图搜索
- ◆盲目式搜索
- ◆启发式搜索

□ 猴子摘香蕉问题



猴子摘香蕉问题示意图

思考：

(1) 问题描述方式

(2) 问题求解步骤

1

前言

- 在学习本章内容之前，我们先了解一下有关知识及其表示的概念。

  人类的智能活动过程主要是一个获得并运用知识的过程，知识是智能的基础。为了使计算机具有智能，就必须使它具有知识。

  那什么是知识呢？

2

讨论话题 1：知识在世界中的存在形式有哪些？

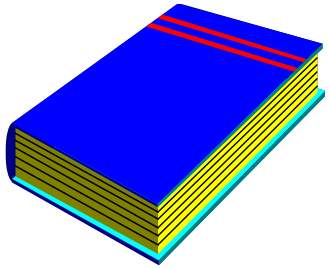
例如：纸质书籍刊物，电子的书籍刊物，.....

讨论话题 2：人工智能领域中知识存在于哪里？以什么形式存在？

知识一般概念

- 知识是人们在改造客观世界的实践中积累起来的认识和经验
- 认识：包括对事物现象、本质、属性、状态、关系、联系和运动等的认识
- 经验：包括解决问题的微观方法和宏观方法
- 微观方法：如步骤、操作、规则、过程、技巧等
- 宏观方法：如战略、战术、计谋、策略等
- 知识的有代表性的定义
- (1) Feigenbaum：知识是经过剪裁、塑造、解释、选择和转换了的信息
- (2) Bernstein：知识由特定领域的描述、关系和过程组成
- (3) Heyes-Roth：知识=事实+信念+启发式
- 知识、信息、数据及其关系
- 数据是信息的载体，本身无确切含义，其关联构成信息
- 信息是数据的关联，赋予数据特定的含义，仅可理解为描述性知识
- 知识可以是对信息的关联，也可以是对已有知识的再认识
- 常用的关联方式：if …… then ……

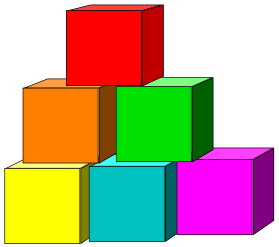
什么是知识?



- 一般来说，我们把有关信息关联在一起所形成的信息结构称为知识。
- 知识表示就是对知识的一种描述，一种计算机可以接受的用于描述知识的数据结构。

5

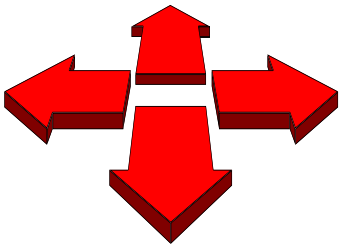
知识的要素



- 知识的要素是指构成知识的必需元素。在这里，我们关心的是一个人工智能系统所处理的知识的组成成分。一般而言，人工智能系统的知识包含**事实**、**规则**、**控制**和**元知识**。

6

知识的要素



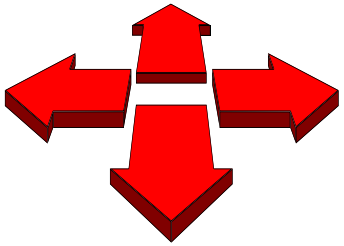
- **事实**：事物的分类、属性、事物间关系、科学事实、客观事实等. 是有关问题环境的一些事物的知识，常以“---是---”形式出现，也是最低层的知识。例如：雪是白色的，人有四肢。

- **规则**：事物的行动、动作和联系的因果关系知识。

这种知识是动态的，常以“如果---那么---”形式出现。例如启发式规则，如果下雨，则出门带伞。

7

知识的要素



- 控制：是有关问题的求解步骤、规划、求解策略等技巧性知识，告诉怎么做一件事。也包括当有多个动作同时被激活时，选择哪一个动作来执行的知识。
- 元知识：怎样使用规则、解释规则、校验规则、解释程序结构等知识。是有关知识的知识，是知识库中的高层知识。元知识与控制知识有时有重叠。

8

- 每种以**知识**和**符号操作**为基础的智能系统，其问题求解方法都需要某种**对解答的搜索**。
- 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

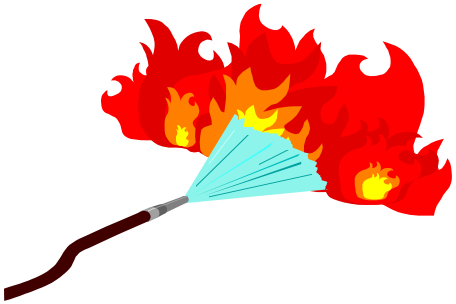
  求解的方法
- **知识表示**方式是学习人工智能的中心内容之一。

9

知识表示的概念

- 什么是知识表示
- 是对知识的描述，即用一组符号把知识编码成计算机可以接受的某种结构。其表示方法不唯一。
- 知识表示的要求
- 表示能力：能否正确、有效地表示问题。包括：
- 表示范围的广泛性
- 领域知识表示的高效性
- 对非确定性知识表示的支持程度
- 可利用性：可利用这些知识进行有效推理。包括：
- 对推理的适应性：推理是根据已知事实利用知识导出结果的过程
- 对高效算法的支持程度：知识表示要有较高的处理效率
- 可实现性：要便于计算机直接对其进行处理
- 可组织性：可以按某种方式把知识组织成某种知识结构
- 可维护性：便于对知识的增、删、改等操作
- 自然性：符合人们的日常习惯
- 可理解性：知识应易读、易懂、易获取等

知识表示的一般方法



- 状态空间法
- 问题归约法
- 谓词逻辑法
- 语义网络
- 另外还有框架表示以及剧本表示，过程表示.
- 在表示和求解比较复杂的问题时，采用单一的表示方法是不够的，往往采用多种方法的混合表示.目前这仍是人工智能专家感兴趣的研究方向.

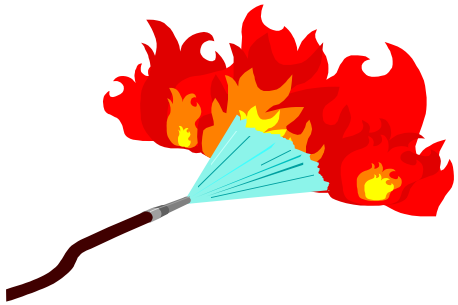
11

状态空间法

问题求解(problem solving)是个大课题，它涉及归约、推断、决策、规划、常识推理、定理证明和相关过程的核心概念。在分析了人工智能研究中运用的问题求解方法之后，就会发现许多问题求解方法是采用试探搜索方法的。也就是说，这些方法是通过在某个可能的解空间内寻找一个解来求解问题的。这种基于解答空间的问题表示和求解方法就是状态空间法，它是以状态和算符(operator)为基础来表示和求解问题的。

12

状态空间法



- 状态空间法

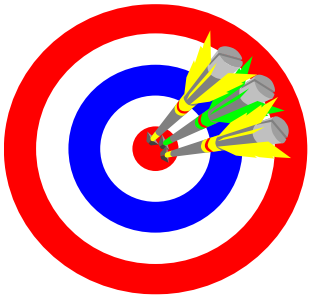
  状态 (State)

  算符 (Operator)

  状态空间方法 (Method on State Space)

13

状态

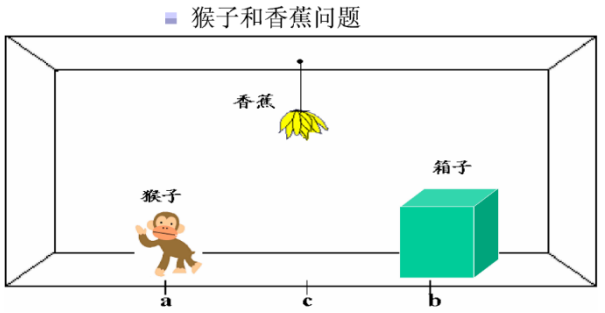


- **状态**（state）：为描述某类不同事物间的差别而引入的一组最少变量 q_0 q_1，…，q_n的有序集合.

  矢量形式： $Q=[q_0, q_1, \ldots q_n]^T$

  式中每个元素q_i为集合的分量，称为**状态变量**。

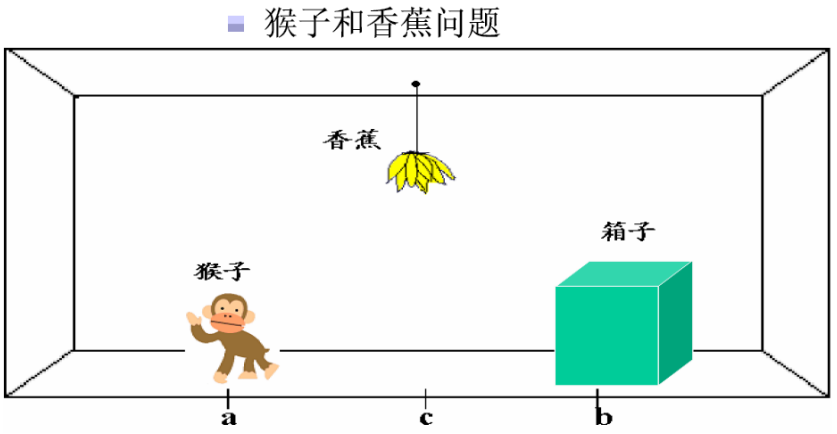
  给定每个分量的一组值就得到**一个具体的状态**，如 $Q_k=[q_{0k}, q_{1k}, \ldots, q_{nk}]$



14

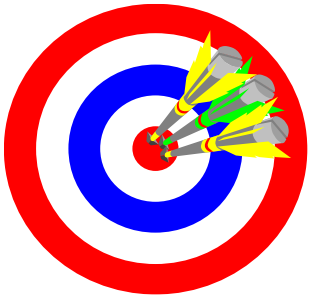
例：

- 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。

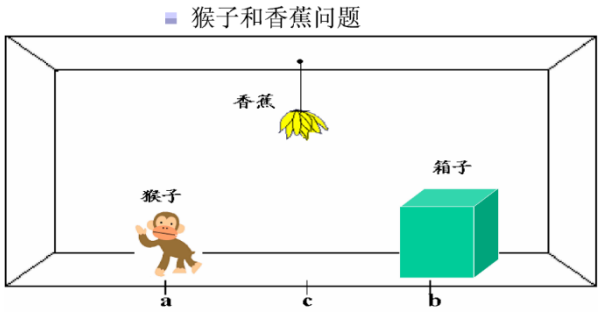


15

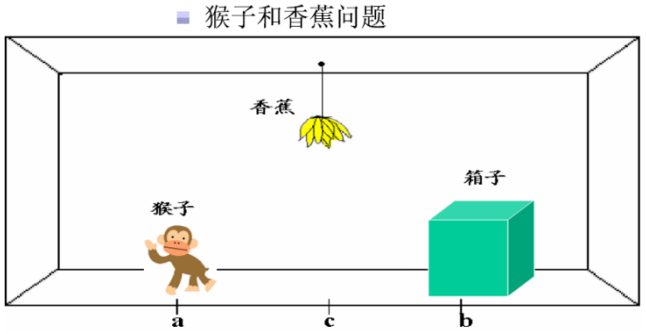
算符



- **算符**（operator）：**把问题从一种状态变换为另一种状态的手段.**
- **算符可为走步、过程、规则、数学算子、运算符号或逻辑符号等。**



16



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：

$$(W,\ 0,\ Y,\ Z) \xrightarrow{\text{goto (U)}} (U,\ 0,\ Y,\ Z)$$

2. pushbox (V)猴子把箱子推到水平位置V，即有：

$$(W,\ 0,\ W,\ Z) \xrightarrow{\text{pushbox(V)}} (V,\ 0,\ V,\ Z)$$

3. climbbox 猴子爬上箱顶，即有：

$$(W,\ 0,\ W,\ Z) \xrightarrow{\text{climbbox}} (W,\ 1,\ W,\ Z)$$

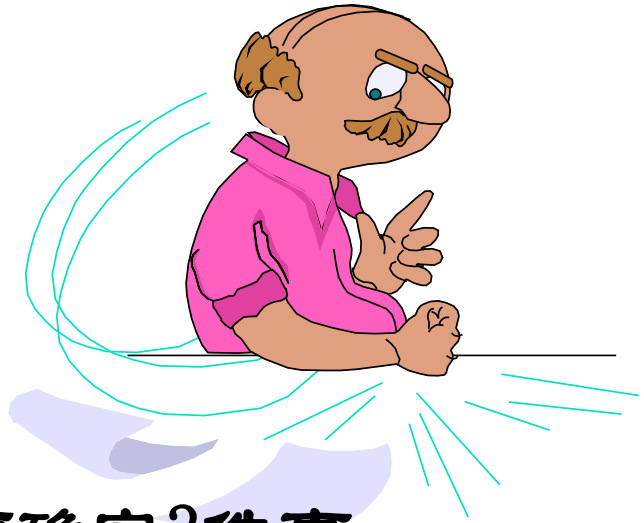
4. grasp 猴子摘到香蕉，即有：

$$(c,\ 1,\ c,\ 0) \xrightarrow{\text{grasp}} (c,\ 1,\ c,\ 1)$$

17

- 问题的**状态空间**：是一个表示该问题全部可能状态及其关系的图。
- 它包含三种说明的集合，即三元状态（S，F，G）
- S —— 初始状态集合；
- F —— 操作符集合；
- G ——目标状态集合。

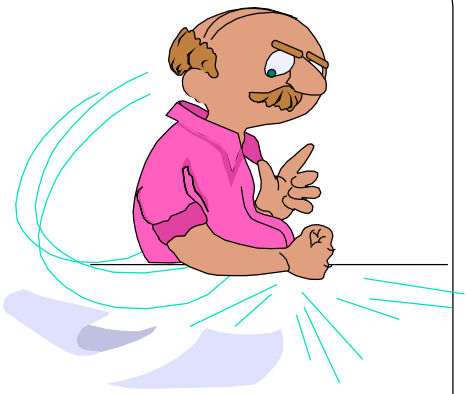
18



- 对一个问题的状态描述，必须确定3件事：
- (1) 该状态描述方式，特别是初始状态描述；
- (2) 操作符集合及其对状态描述的作用；
- (3) 目标状态描述的特性。

19

状态空间表示



- 典型的例子：
- 下棋、迷宫及各种游戏。

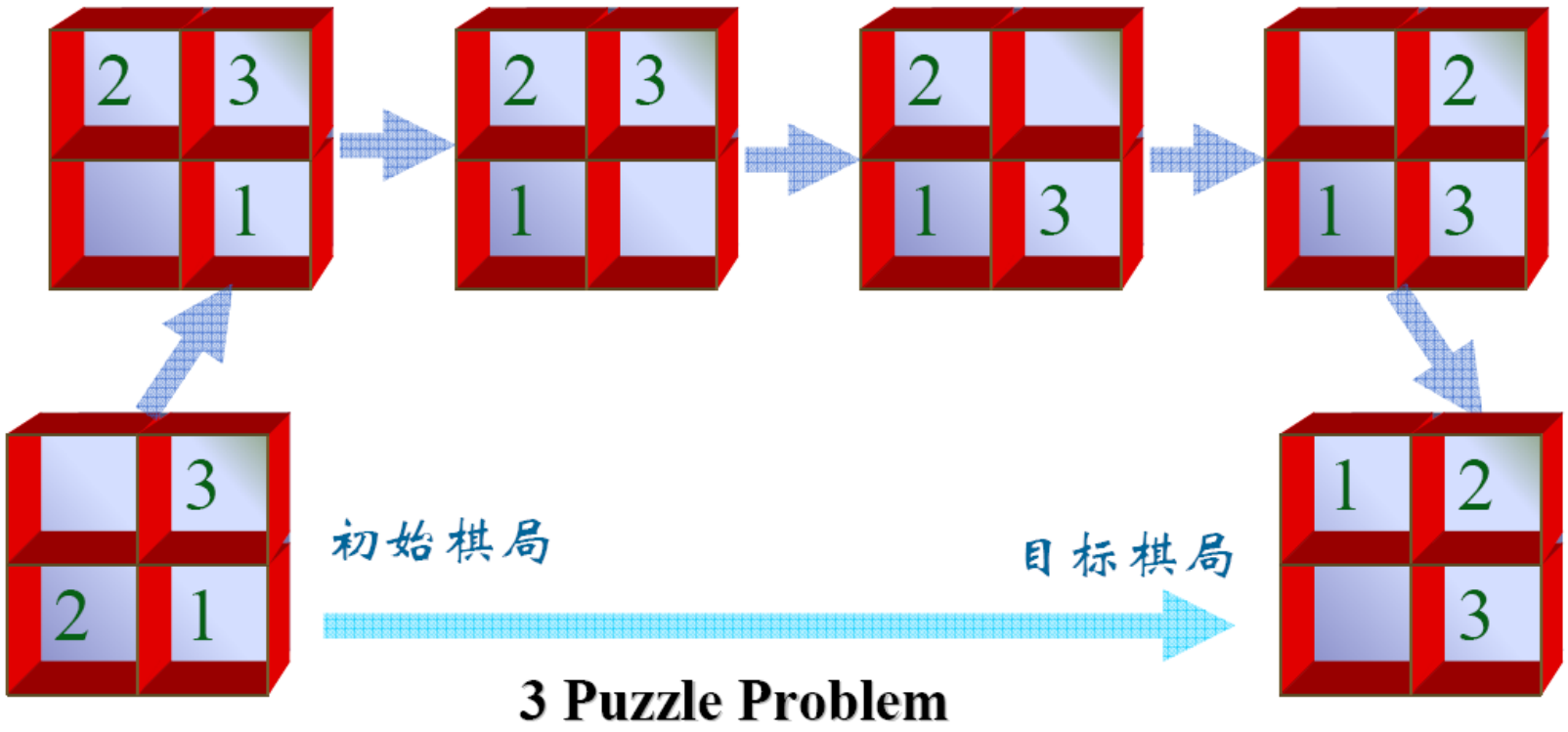
20

三数码难题

- 问题描述：
- 三数码难题由3个编有1-3并放在2×2方格棋盘上可走动的棋子组成。棋盘上总有一个空格，以便让空格周围的棋子走进来。直至从初始状态到达目标状态。

21

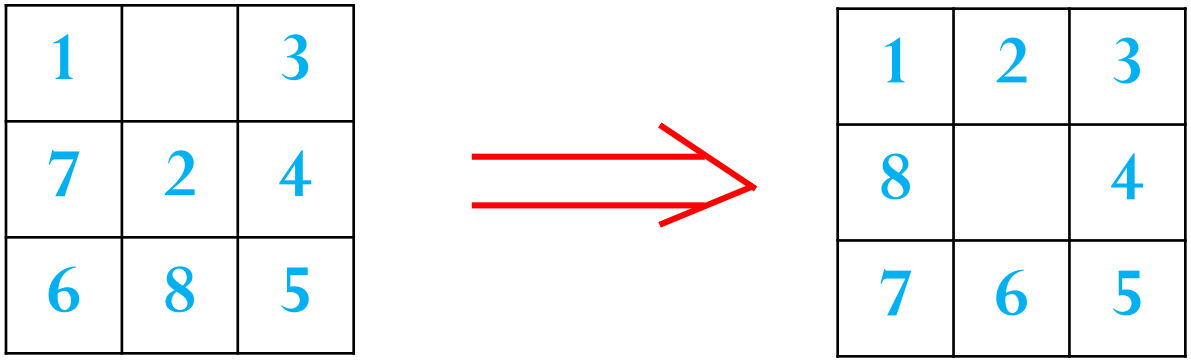
三数码难题



3 Puzzle Problem

22

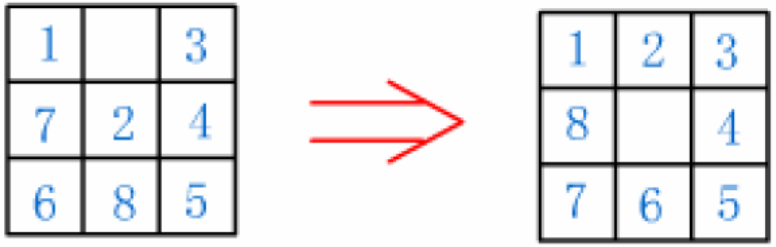
八数码难题



初始棋局　　目标棋局

23

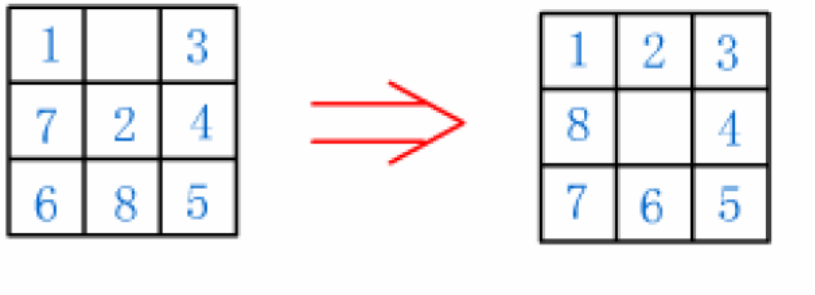
表示



•根据问题状态、操作算符和目标条件选择各种表示，是高效率求解必须的。在问题求解过程中，会不断取得经验，获得一些简化的表示。

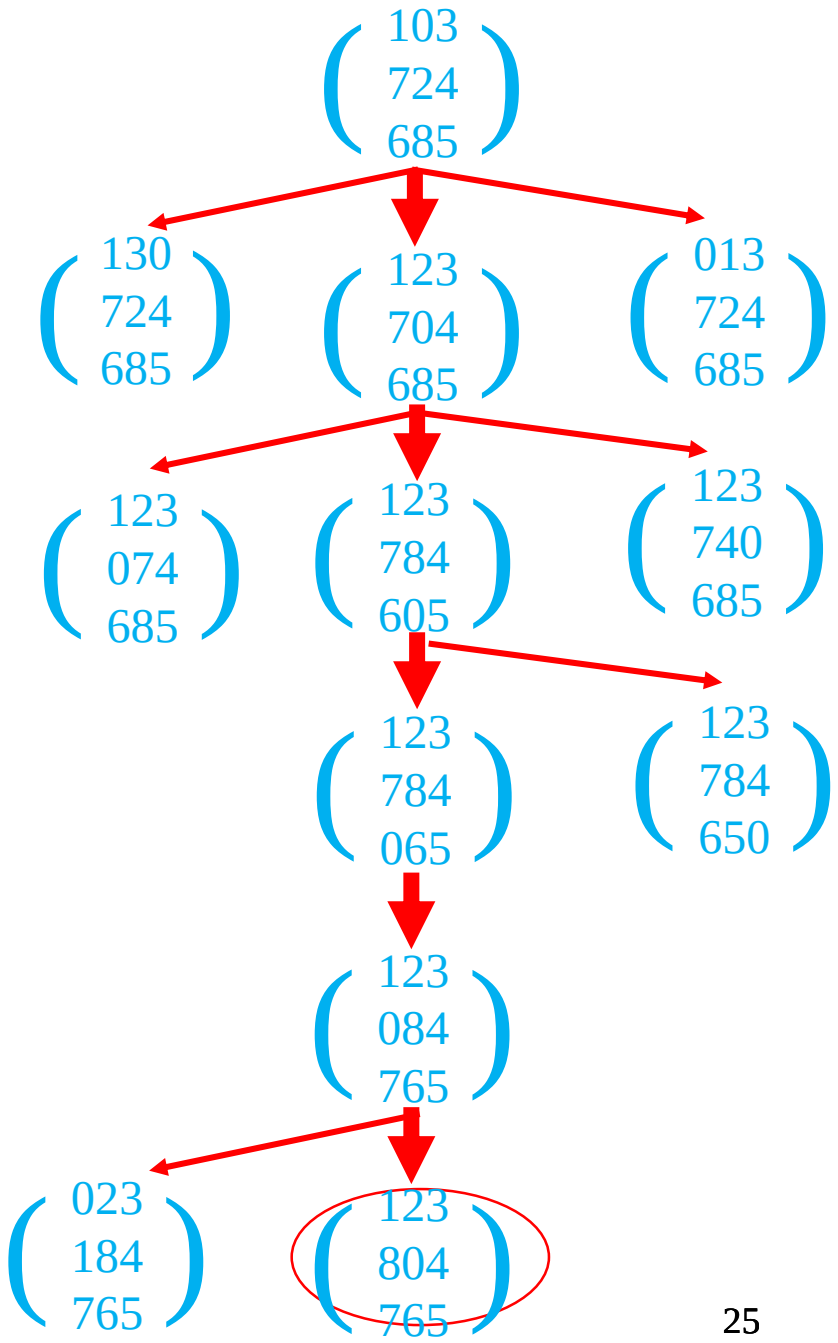
- 制定操作算符集：
- * 直观方法——为每个棋牌制定一套可能的走步：左、上、右、下四种移动。这样就需32个操作算子。
- * 简易方法——仅为空格制定这4种走步，因为只有紧靠空格的棋牌才能移动。
- * 空格移动的唯一约束是不能移出棋盘。

24



从初始棋局开始，试探由每一合法走步得到的各种新棋局，然后计算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。

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**状态图**。图中每个节点标有它所代表的棋局。首先把适用的算符用于初始状态，以产生新的状态；然后，再把另一些适用算符用于这些新的状态；这样继续下去，直至产生目标状态为止。



八数码难题部分状态图

十五数码难题(思考)

11	9	4	15
1	3		12
7	5	8	6
13	2	10	14

初始状态

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

目标状态

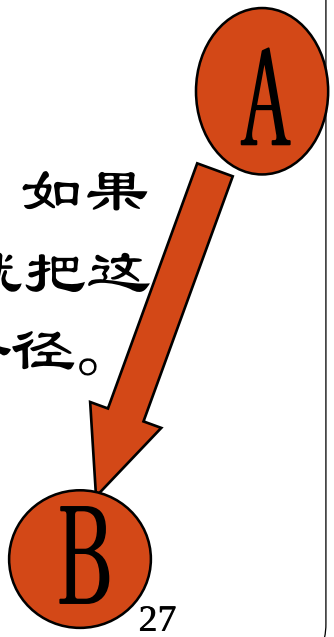
26

状态图示法

有向图(directed graph)

图，由节点（不一定是有限的节点）和边的集合构成。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。



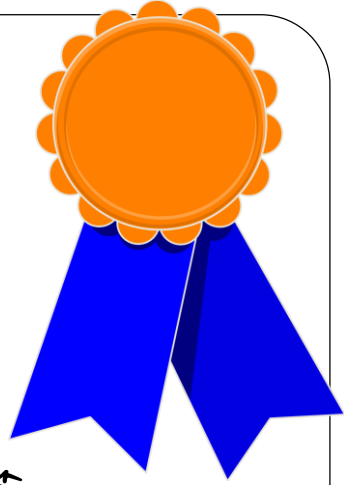
27

状态图示法

- 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.
- 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。
- 图的显式说明　　指各节点及其具有代价的弧线可以由一张表明确给出
- 图的隐式说明　　指各节点及其具有代价的弧线不可以由一张表明确给出（起始节点＋后继算符，把后继算符应用于各节点，以扩展节点）

28

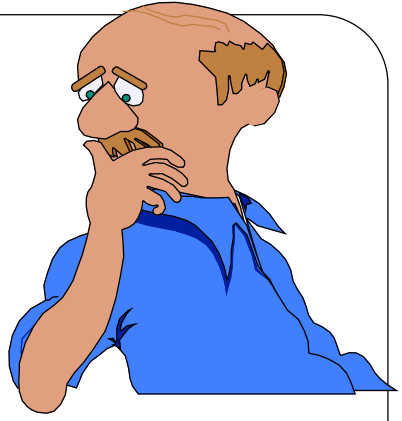
状态图示法



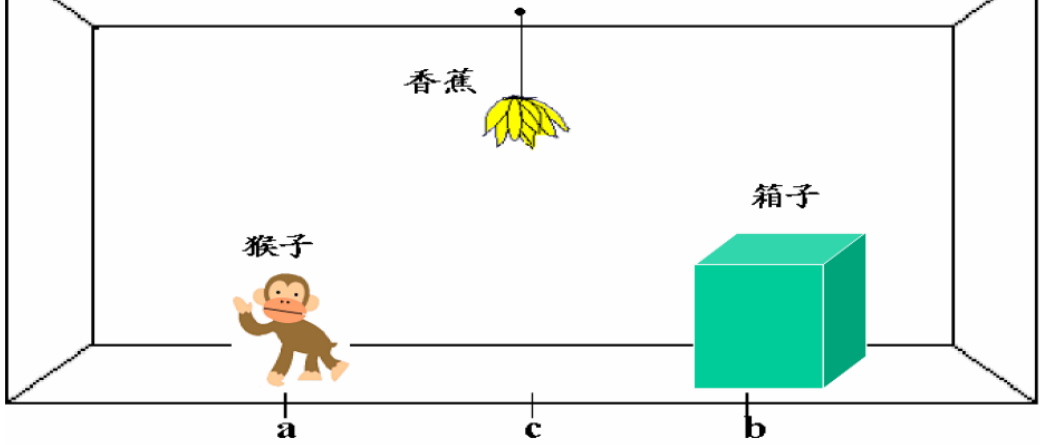
- 搜索某个状态空间以求得算符序列的一个解答的过程，就对应于使隐式图足够大的一部分变为显示图以便包含目标的过程。
- 这样的搜索图是状态空间问题求解的主要基础。

29

状态空间表示举例



- 猴子和香蕉问题

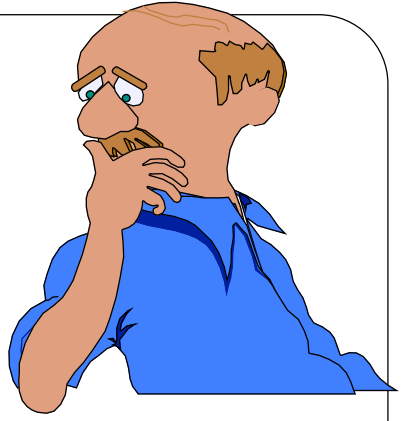


问题描述

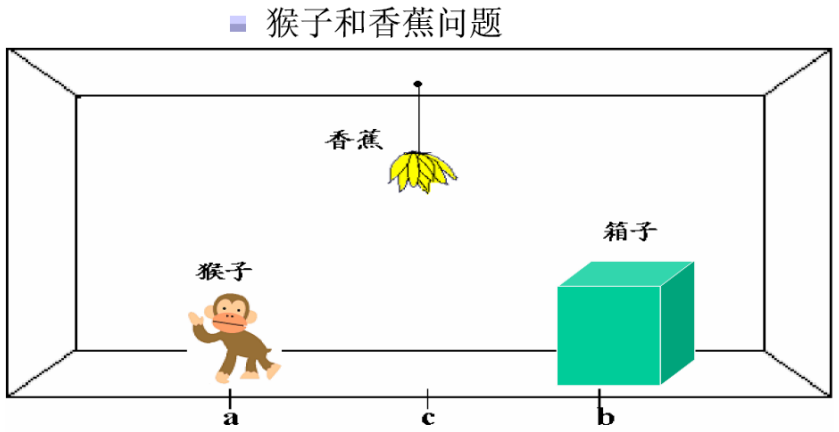
在一个房间内有一只猴子(可把这只猴子看做一个机器人)、一个箱子和一束香蕉。香蕉挂在天花板下方，但猴子的高度不足以碰到它。那么这只猴子怎样才能摘到香蕉呢?

30

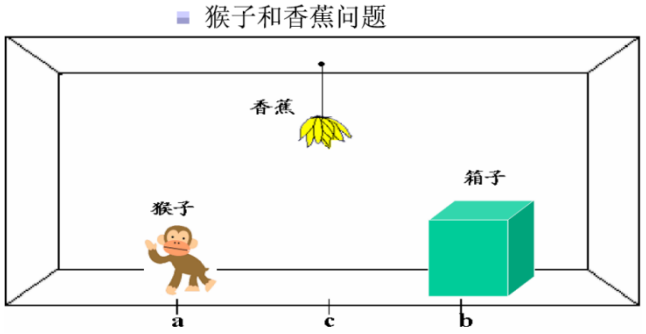
解题过程



- 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。



31



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：

$$(W,\ 0,\ Y,\ Z) \xrightarrow{\text{goto (U)}} (U,\ 0,\ Y,\ Z)$$

2. pushbox (V)猴子把箱子推到水平位置V，即有：

$$(W,\ 0,\ W,\ Z) \xrightarrow{\text{pushbox(V)}} (V,\ 0,\ V,\ Z)$$

3. climbbox 猴子爬上箱顶，即有：

$$(W,\ 0,\ W,\ Z) \xrightarrow{\text{climbbox}} (W,\ 1,\ W,\ Z)$$

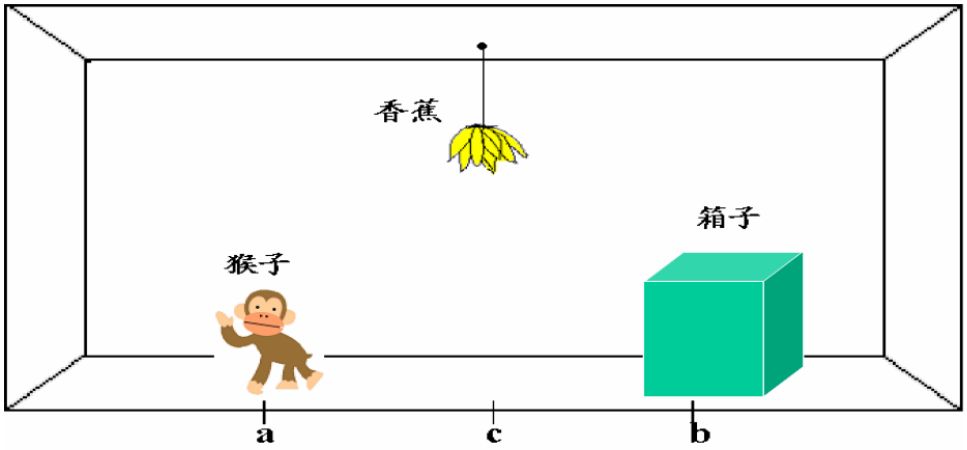
4. grasp 猴子摘到香蕉，即有：

$$(c,\ 1,\ c,\ 0) \xrightarrow{\text{grasp}} (c,\ 1,\ c,\ 1)$$

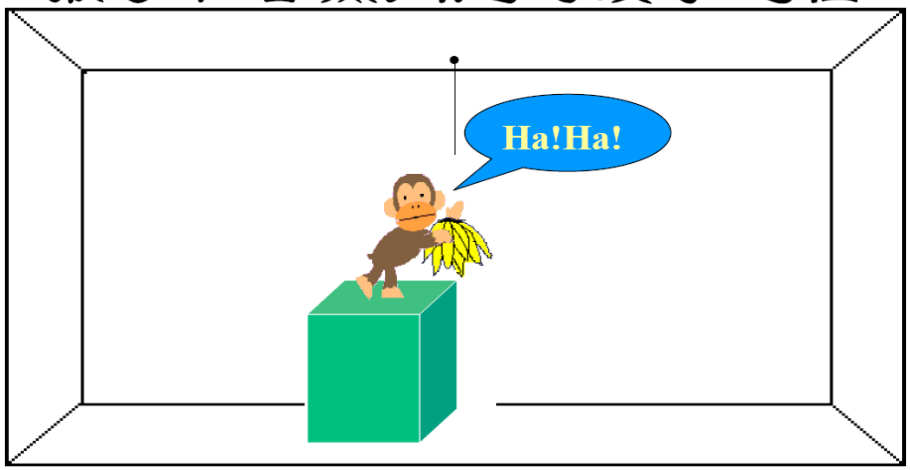
32

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

猴子和香蕉问题

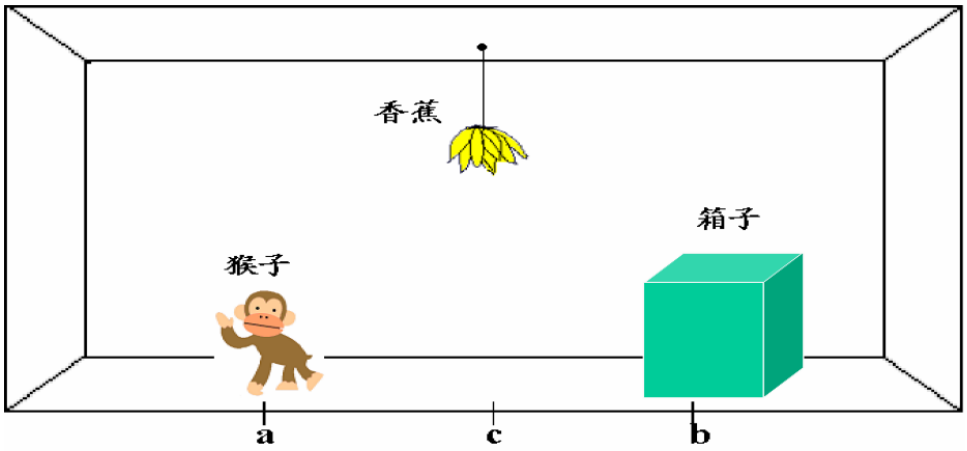


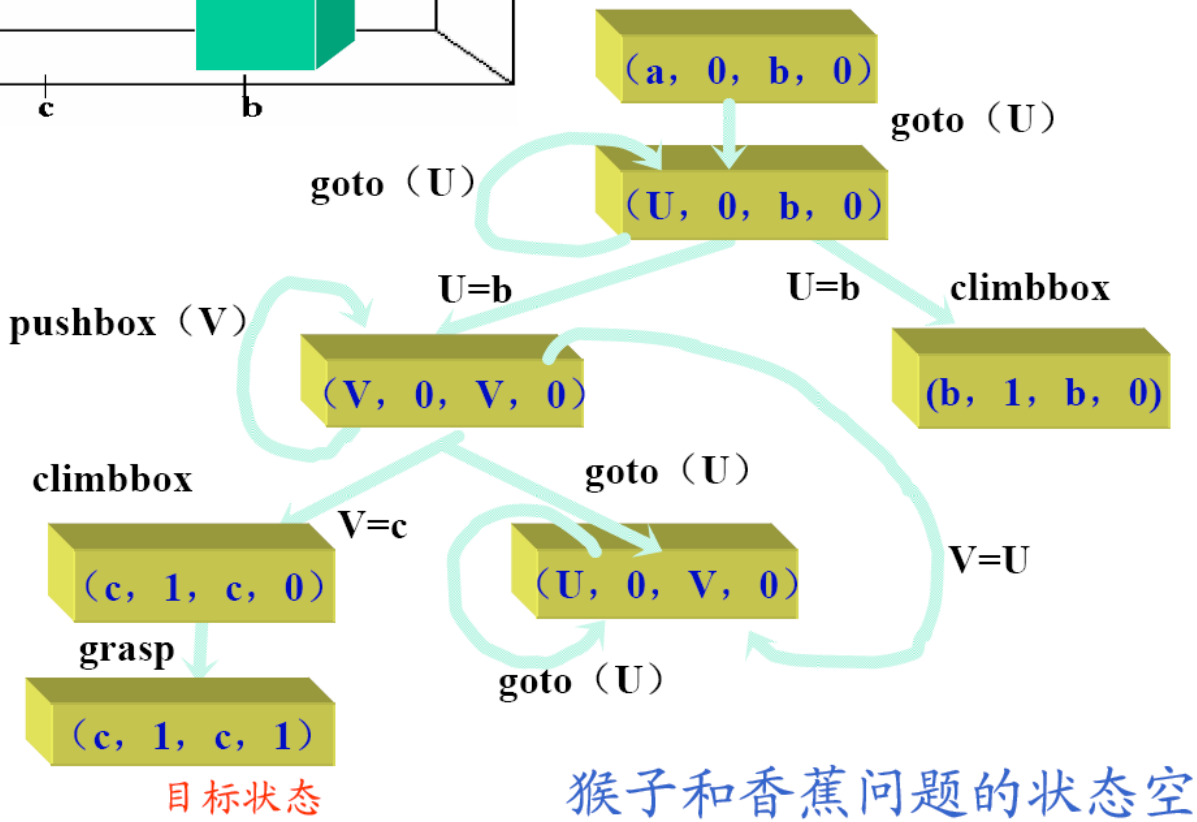
猴子和香蕉问题的演示过程



33

- 猴子和香蕉问题





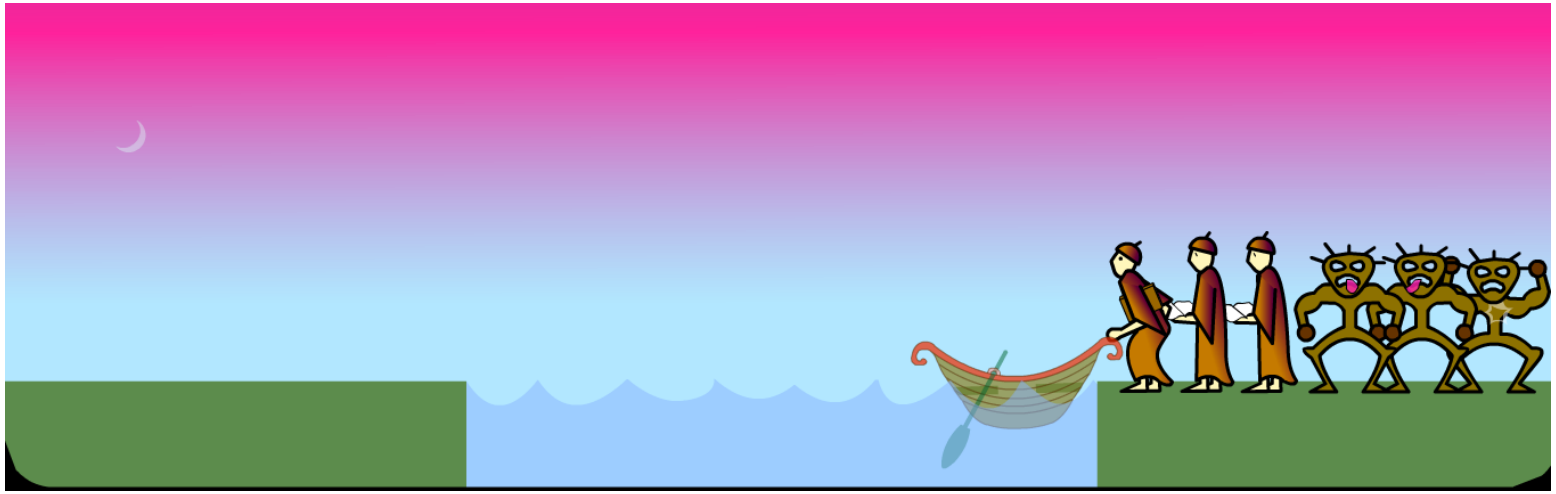
猴子和香蕉问题的状态空间图

34

状态空间表示举例2

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？



- 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

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

（左岸传教士数，左岸野人数，船的位置）

初始状态：（0，0，0）

目标状态：（3，3，1）

0：右岸

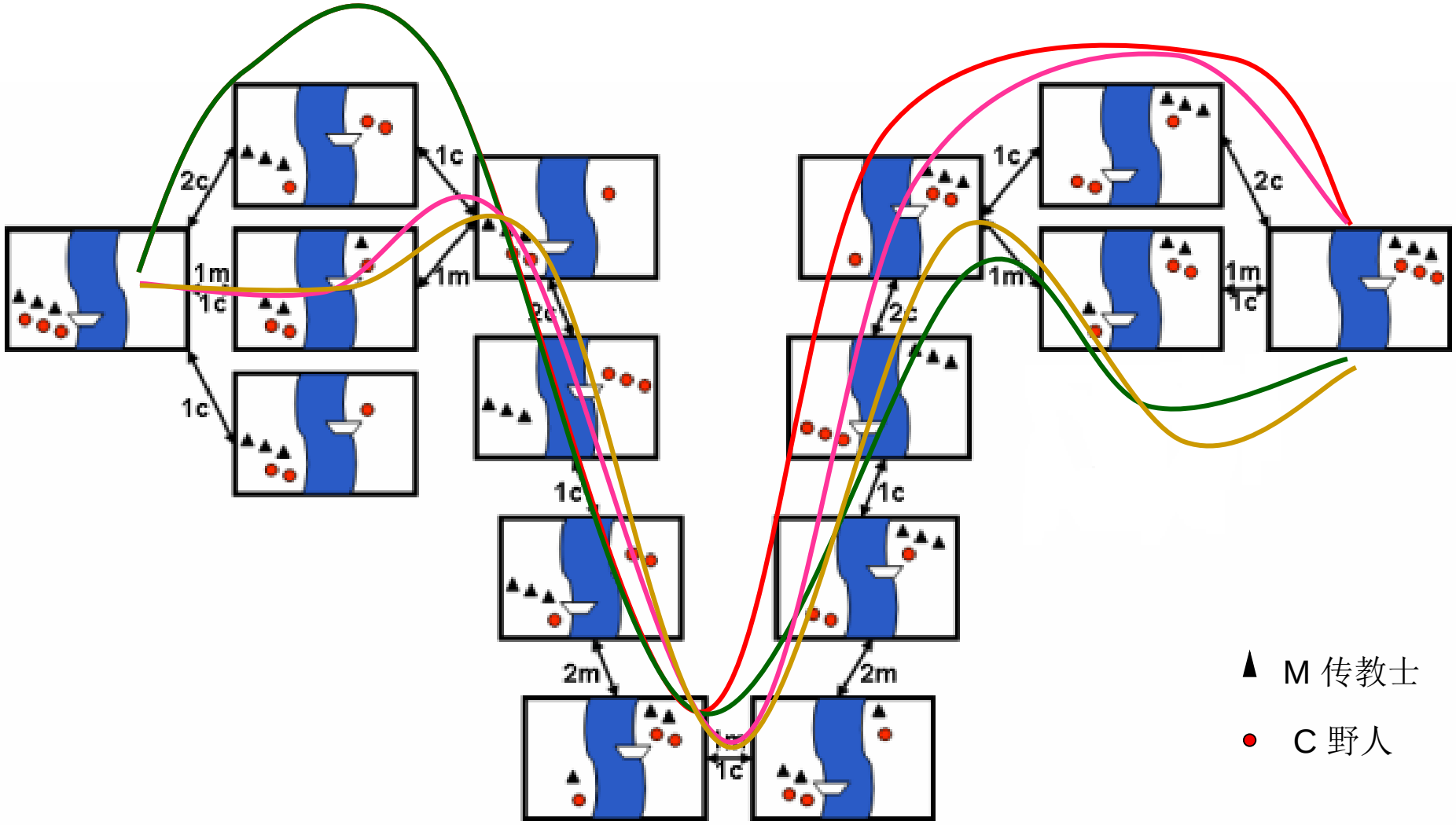
1：左岸

哪些操作能导致状态变化？

状态的转换

- 算子（算符，操作符）——使状态发生改变的操作
- MC问题中的算子
  - 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

传教士野人问题状态空间图



- 例2.1 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。

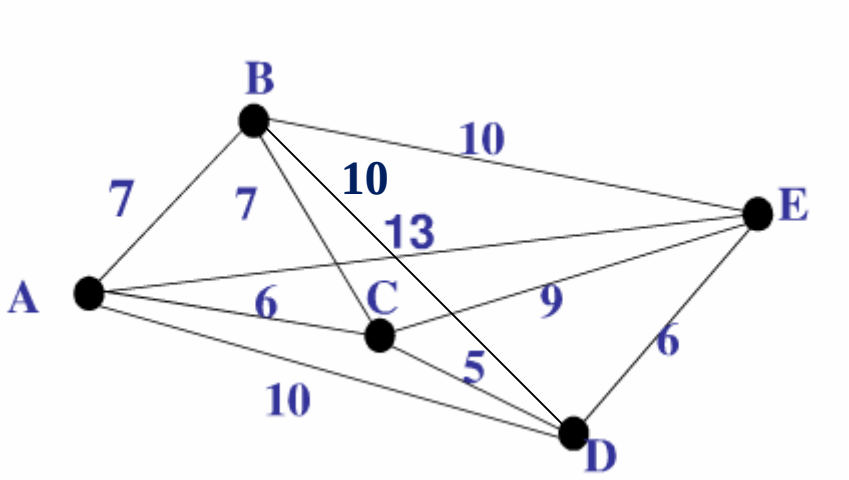


图2.3 推销员旅行问题

- 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)

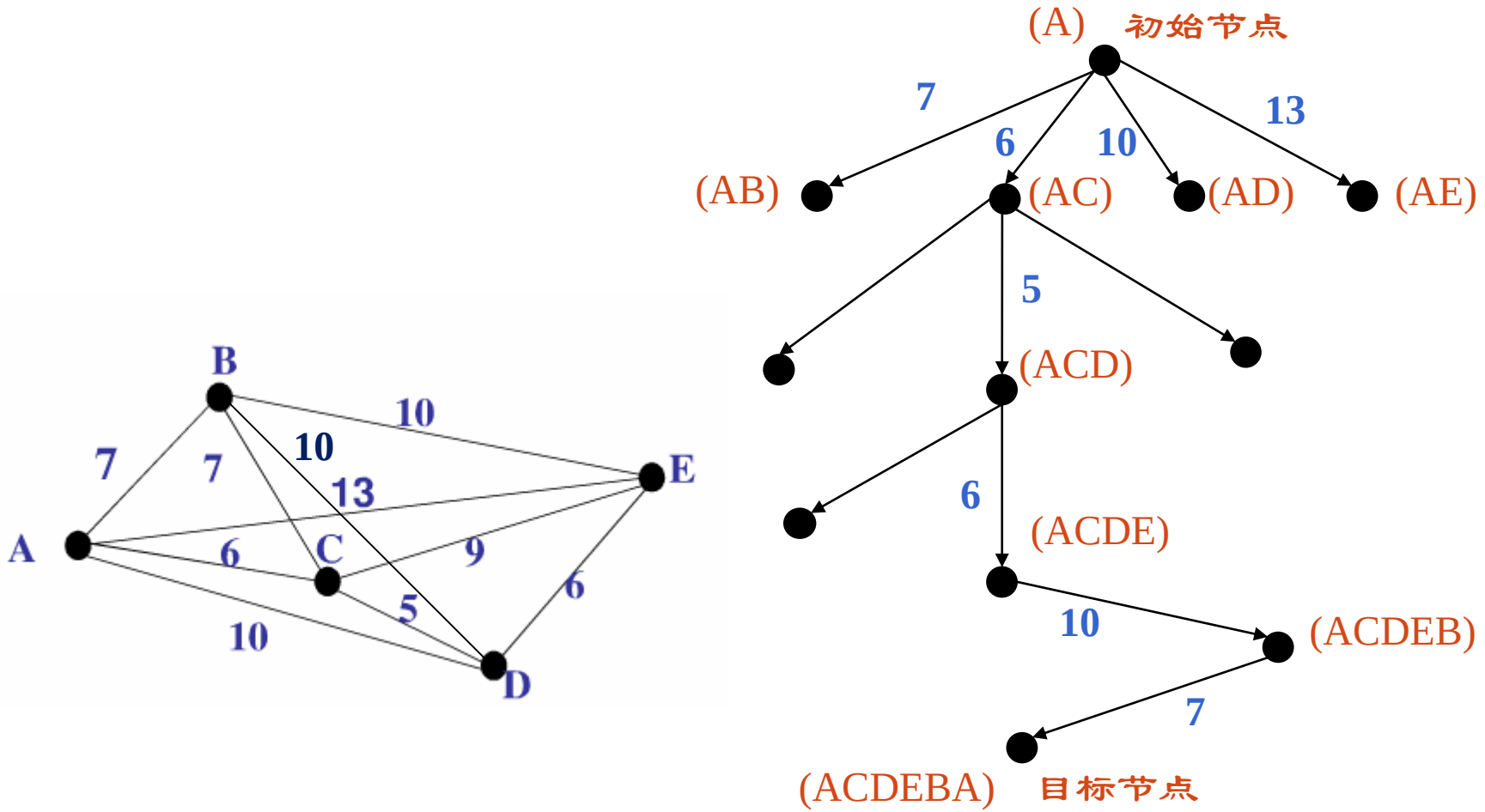


图2.4 推销员旅行问题状态空间图

- 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

搜索推理技术

- 知识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**搜索推理过程**.

- 本节在上一节知识表示的基础上研究**问题求解的方法**，是人工智能研究的又一核心问题。内容包括早期搜索推理技术，如图搜索策略和消解原理；以及高级搜索推理技术，如规则演绎系统、产生式系统.

搜索技术

- 搜索技术是人工智能的基本技术之一，在人工智能各应用领域中被广泛地使用。
- 早期的人工智能程序与搜索技术联系就更为紧密，几乎所有的早期的人工智能程序（智力难题、棋类游戏、简单数学定理证明）都是以搜索为基础的。
- 现在，搜索技术渗透在各种人工智能系统中，可以说没有哪一种人工智能的应用不用搜索方法，在专家系统、自然语言理解、自动程序设计、模式识别、机器人学、信息检索和博弈都广泛使用搜索技术。

搜索技术

- 什么是搜索
  - 根据问题的实际情况不断寻找可利用的知识，构造出一条代价较少的推理路线，使问题得到圆满解决的过程称为搜索
  - 包括两个方面：

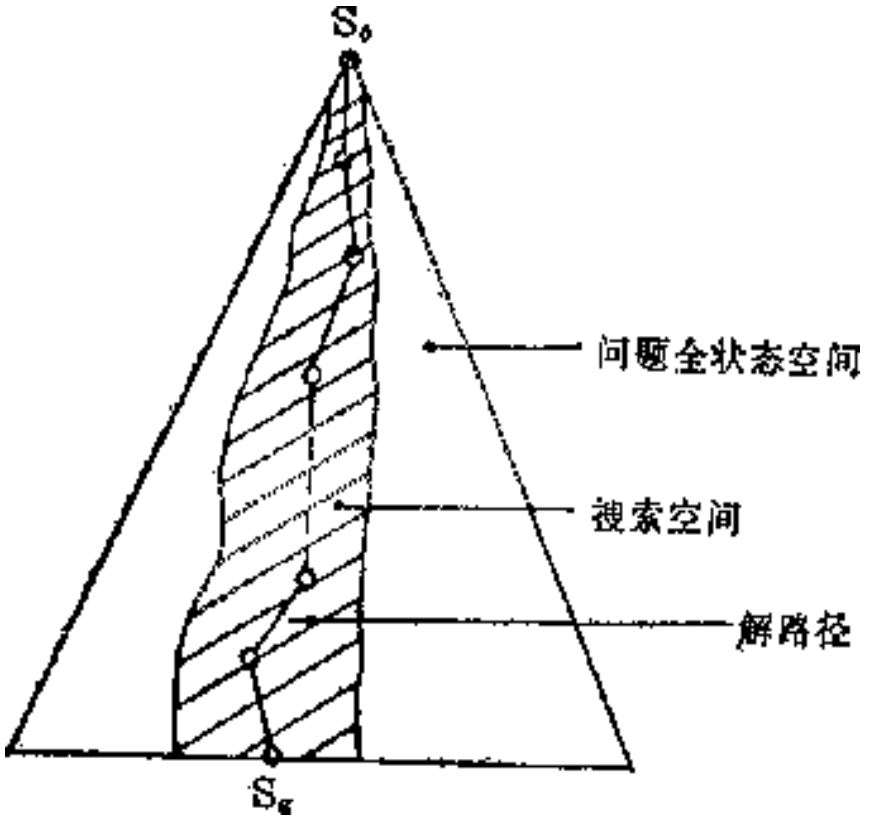
    ——— 找到从初始事实到问题最终答案的一条推理路径

    ——— 找到的这条路径在时间和空间上复杂度最小

搜索推理技术

搜索方法好的标准，一般认为有两个：

- (1) 搜索空间小
- (2) 解最佳



搜索推理技术

- 图搜索策略
- 盲目搜索
- 启发式搜索

} 状态空间搜索

- 图搜索策略
- 一种在图中寻找路径的方法.
- 图中每个节点对应一个状态，每条连线对应一个操作符.

图的搜索过程

必须记住从目标返回的路径
些点

深度优先搜索

状态：(城市名)

算子：常德→益阳

益阳→常德

益阳↔汨罗

益阳↔宁乡

益阳↔娄底

…

图的搜索过程

必须记住下一步还可以走哪些点

OPEN表(记录还没有扩展的点

用于存放刚生成的节点)

必须记住哪些点走过了

CLOSED表(记录已经扩展的点

用于存放已经扩展或将要扩展的节点)

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

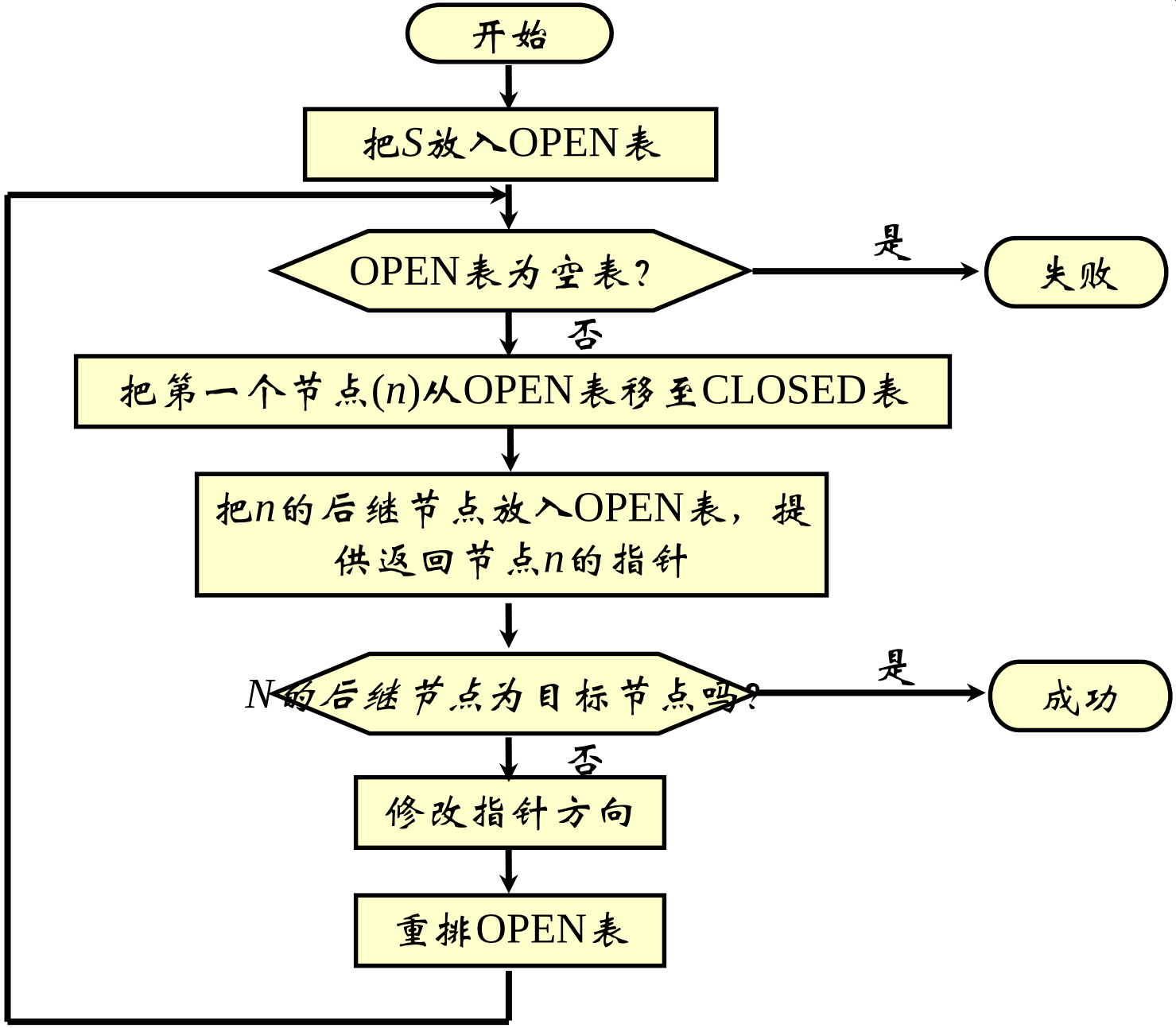
图的搜索过程

- 状态空间搜索的基本思想
- 先把问题的初始状态作为当前扩展节点对其进行扩展，生成一组子节点，然后检查问题的目标状态是否出现在这些子节点中。若出现，则搜索成功，找到了问题的解；若没出现，则再按照某种搜索策略从已生成的子节点中选择一个节点作为当前扩展节点。重复上述过程，直到目标状态出现在子节点中或者没有可供操作的节点为止。所谓对一个节点进行“扩展”是指对该节点用某个可用操作进行作用，生成该节点的一组子节点。

图搜索过程要点

- 搜索过程的要点如下：
  - 起始节点：对应于初始状态描述
  - 后继节点：把适用于某个节点状态描述的一些算符用来推算该节点而得到的新节点，称为该节点的后继节点
  - 指针：从每个后继节点返回指向其父辈节点
  - 考察各后继节点看是否为目标节点.
- 搜索过程扩展后继节点的次序
  - 如果搜索是以接近起始节点的程度（由节点之间连结弧线的数目来衡量）依次扩展节点，称为广（宽）度优先搜索
  - 如果搜索时首先扩展最新产生的节点，称为深度优先搜索

图的一般搜索策略


开始
把S放入OPEN表
OPEN表为空表？
是
失败
否
把第一个节点(n)从OPEN表移至CLOSED表
把n的后继节点放入OPEN表，提供返回节点n的指针
N的后继节点为目标节点吗？
是
成功
否
修改指针方向
重排OPEN表

图搜索过程

(1)把初始节点S0放入OPEN表,并建立目前只包含S0的图,记为G

(2)检查OPEN表是否为空,若为空则问题无解,退出

(3)把OPEN表的第一个节点取出放入CLOSED表,记该节点为n

(4)考察n是否为目标节点.如是,则问题得解,退出

(5)扩展节点n,生成一组子节点.把其中不是节点n先辈的那些节点记作集合M,并把这些子节点作为节点n的子节点加入G中

(6)针对M中子节点的不同情况,分别进行处理

1. 对于那些未曾在G中出现过的M成员设置一个指向父节点(即n)的指针,并把它们放入OPEN表
2. 对于那些先前已在G中出现过的M成员,确定是否需要修改它指向父节点的指针
3. 对于那些先前已在G中出现并且已经扩展了的M成员,确定是否需要修改其后继节点指向父节点的指针

(7)按某种搜索策略对OPEN表中的节点进行排序

(8)转第(2)步

图搜索过程

(1)把初始节点S0放入OPEN表,并建立目前只包含S0的图,记为G

(2)检查OPEN表是否为空,若为空则问题无解,退出

(3)把OPEN表的第一个节点取出放入CLOSED表,记该节点为n

(4)考察n是否为目标节点.如是,则问题得解,退出

(5)扩展节点n,生成一组子节点.把其中不是节点n先辈的那些节点记作集合M,并把这些子节点作为节点n的子节点加入G中

(6)针对M中子节点的不同情况,分别进行处理

1. 对于那些未曾在G中出现过的M成员设置一个指向父节点(即n)的指针,并把它们放入OPEN表
2. 对于那些先前已在G中出现过的M成员,确定是否需要修改它指向父节点的指针
3. 对于那些先前已在G中出现并且已经扩展了的M成员,确定是否需要修改其后继节点指向父节点的指针

(7)按某种搜索策略对OPEN表中的节点进行排序

(8)转第(2)步

图搜索过程

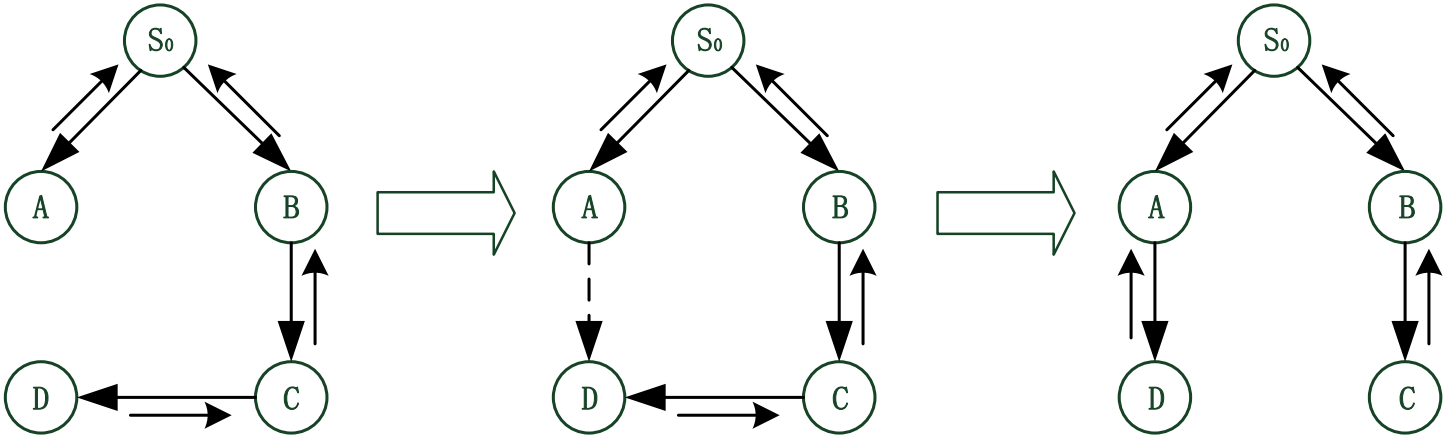
- 注:这是一个通用的搜索过程,后面讨论的状态空间各种搜索策略都是其特例.各种搜索策略的主要区别就是对OPEN表中节点排序准则不同
- 算法结束后，将生成一个图G，称为搜索图。同时由于每个节点都有一个指针指向父节点，这些指针指向的节点构成G的一个支撑树，称为搜索树。
- 从目标节点开始，将指针指向的状态回串起来，即找到一条解路径.

(5) 扩展节点n, 生成一组子节点. 把其中不是节点n先辈的那些节点记作集合M, 并把这些子节点作为节点n的子节点加入G中

(6) 针对M中子节点的不同情况, 分别进行处理

1. 对于那些未曾在G中出现过的M成员设置一个指向父节点(即n)的指针, 并把它们放入OPEN表
2. 对于那些先前已在G中出现过的M成员, 确定是否需要修改它指向父节点的指针
3. 对于那些先前已在G中出现并且已经扩展了的M成员, 确定是否需要修改其后继节点指向父节点的指针

- **说明:**
- (1) n的先辈节点。
- (2) 对已存在于OPEN表的节点(如果有的话)删除之; 但删除之前要比较其返回初始节点的新路径与原路径, 如果新路径“短”, 则修改这些节点在OPEN表中的原返回指针, 使其沿新路返回。
- (3) 对已存在于CLOSED表的节点(如果有的话), 做与(2)同样的处理, 并且再将其移出CLOSED表, 放入OPEN表重新扩展(为了重新计算代价)。
- (4) 对其余子节点配上指向n的返回指针后放入OPEN表中某处。



修改返回指针示例

进一步说明：

(1) 步6中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。

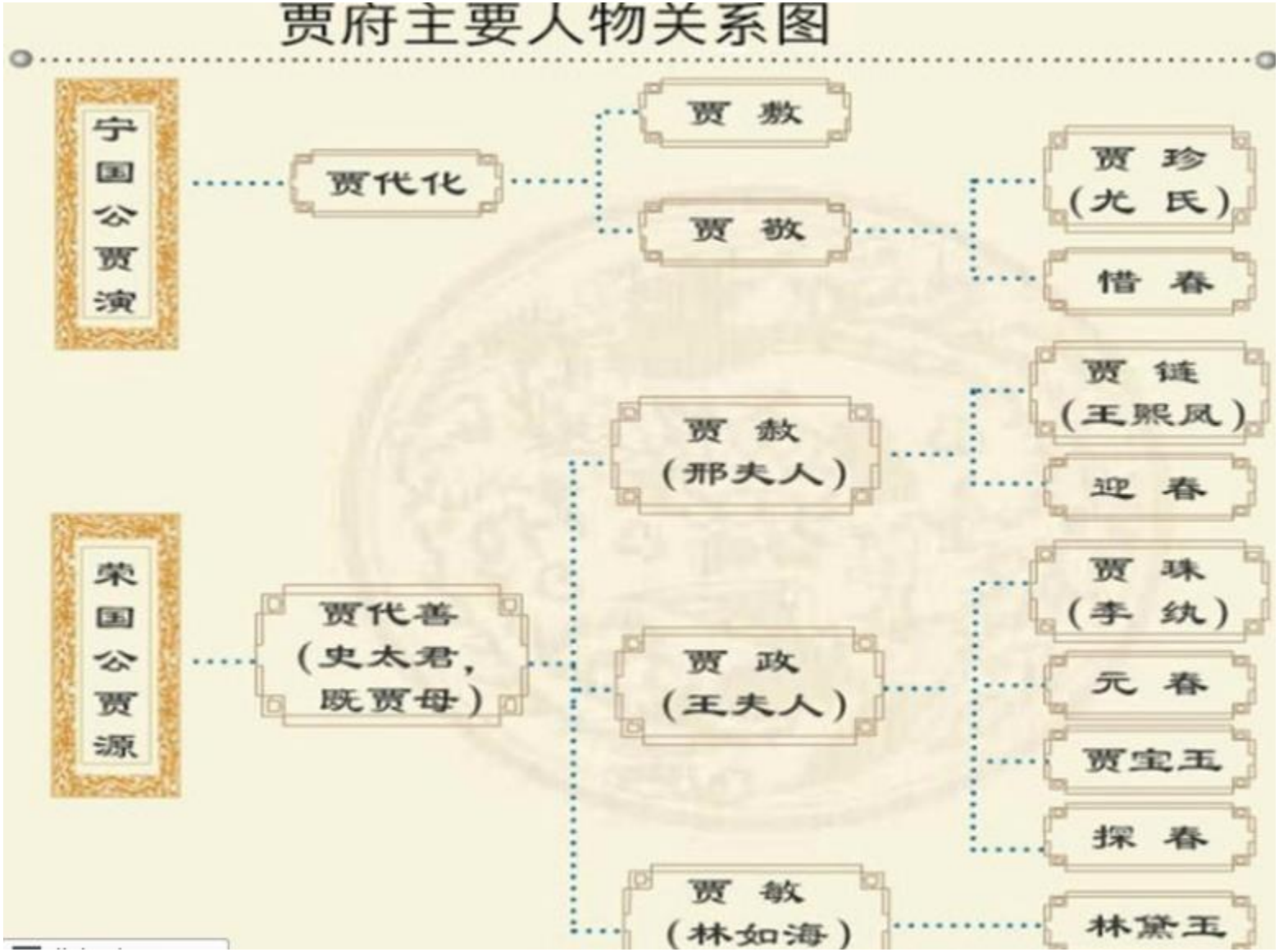
图搜索方法分析

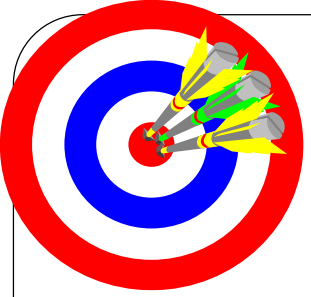
- 每当被选作扩展的节点为目标节点时，这一过程就宣告成功结束。这时，能够重现从起始节点到目标节点的这条成功路径，其办法是从目标节点按指针向S返回追溯。
- 当搜索树不再剩有未被扩展的端节点时，过程就以失败告终(某些节点最终可能没有后继节点，所以OPEN表可能最后变成空表)。在失败终止的情况下，从起始节点出发，一定达不到目标节点。
- 图搜索过程的第7步对OPEN表上的节点进行排序，以便能够从中选出一个“最好”的节点作为第3步扩展用。
- 这种排序可以是任意的即盲目的(属于盲目搜索)，也可以用以后要讨论的各种启发思想或其它准则为依据(属于启发式搜索)。

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

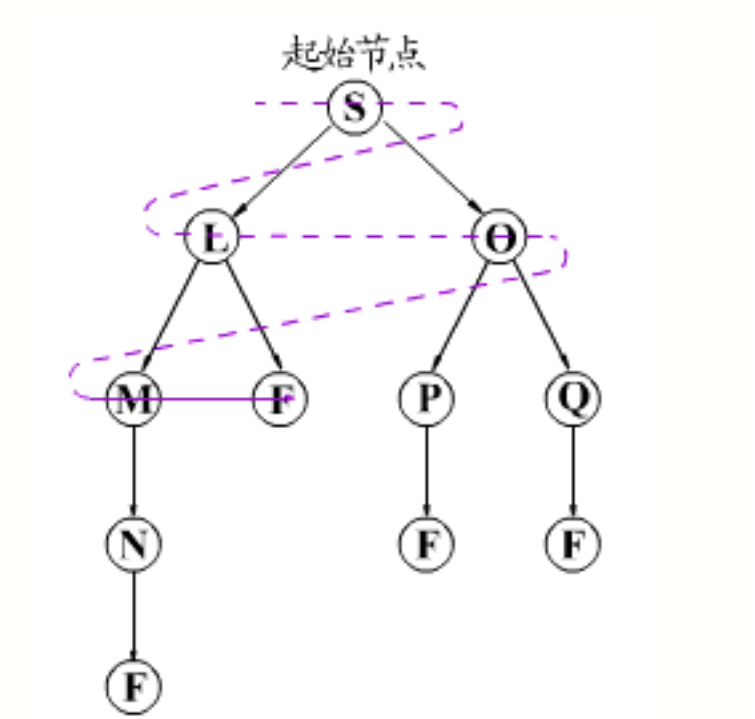
盲目搜索


贾府主要人物关系图
宁国公贾演
贾代化
贾敷
贾敬
贾珍（尤氏）
惜春
荣国公贾源
贾代善（史太君，既贾母）
贾赦（邢夫人）
贾政（王夫人）
贾敏（林如海）
贾琏（王熙凤）
迎春
贾珠（李纨）
元春
贾宝玉
探春
林黛玉

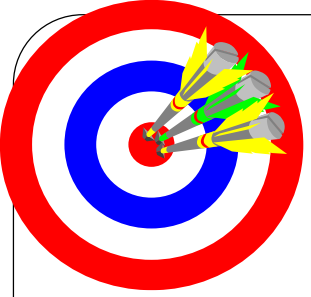


宽度优先搜索策略

- 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



宽度优先搜索策略算法

（1）把初始节点S。放入OPEN表；

（2）如果OPEN表为空，则问题无解，退出；

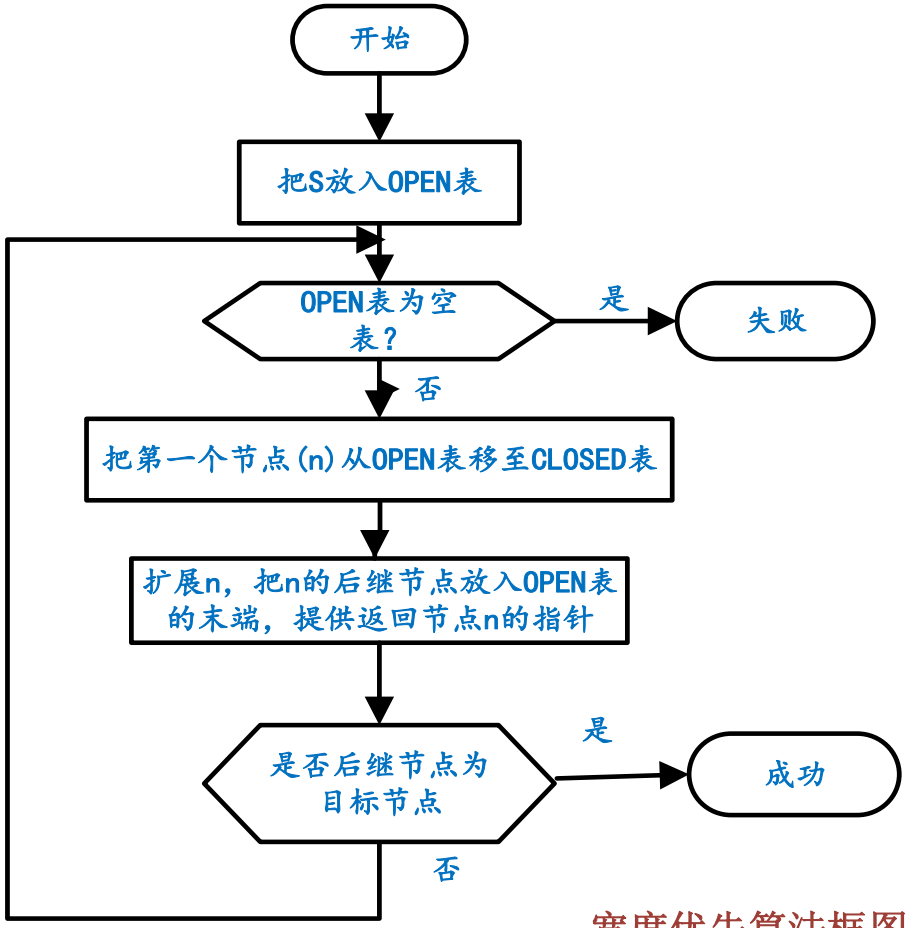
（3）把OPEN表的第一个节点（记为n）取出放入CLOSED表；

（4）若节点n不可扩展，则转第（2）步；

（5）扩展节点n，将其子节点放入OPEN表的末端，并为每一个子节点都配置指向父节点的指针；

（6）后继节点是否为目标节点。若是，则求得了问题的解，退出；否则转第（2）步.

宽度优先搜索策略框图



宽度优先算法框图

八数码难题

__8--puzzle problem

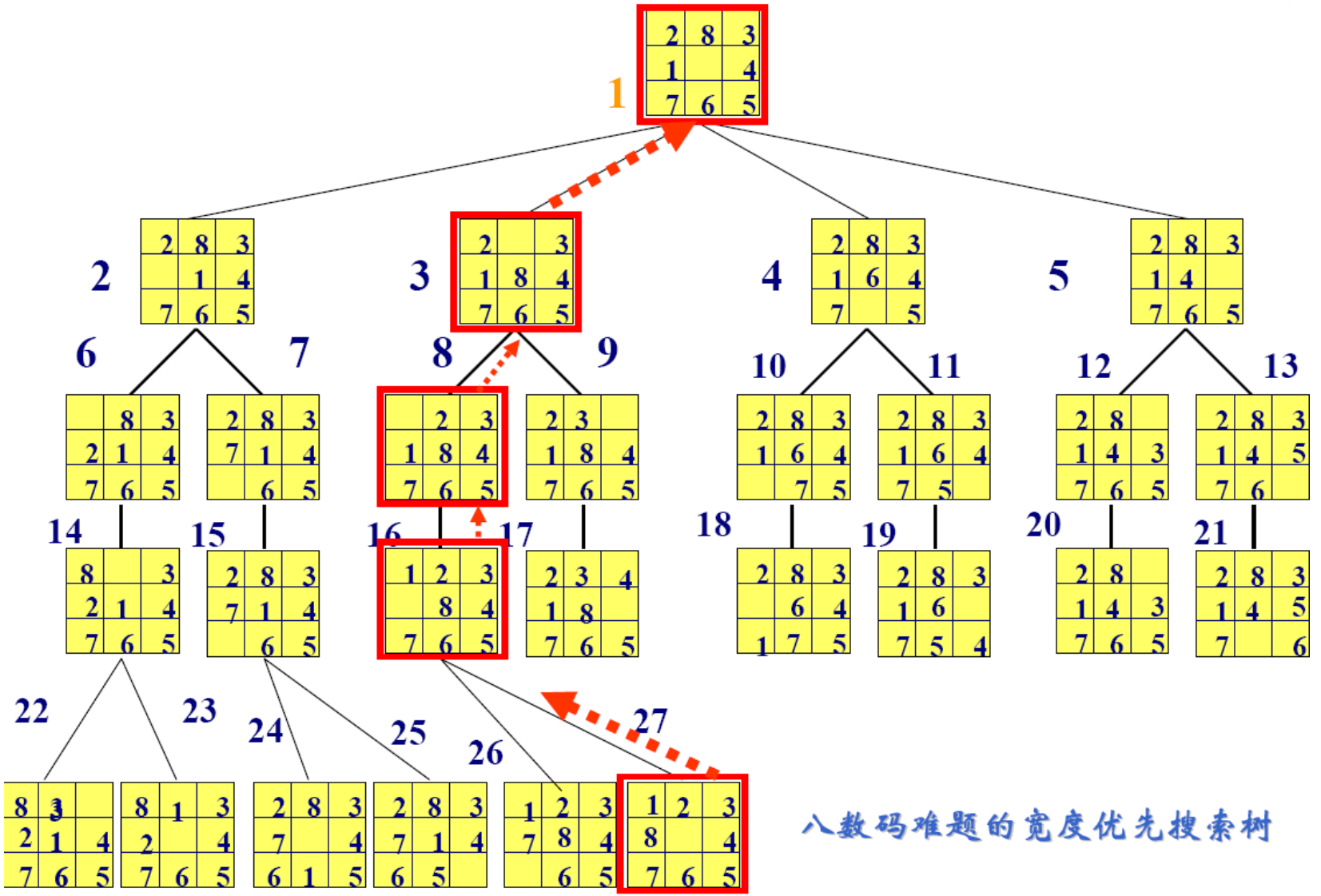
（初始状态）　　（目标状态）

要求寻找从初始状态到目标状态的路径。

解：　应用宽度优先搜索，可得到如下图所示的搜索树，并可得到解的路径是：

S。 → 3 → 8 → 16 → 27

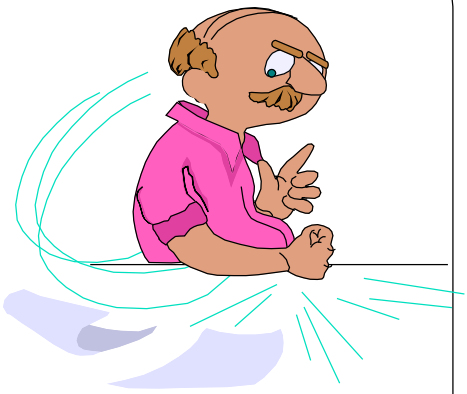
S_0 → 3 → 8 → 16 → 27



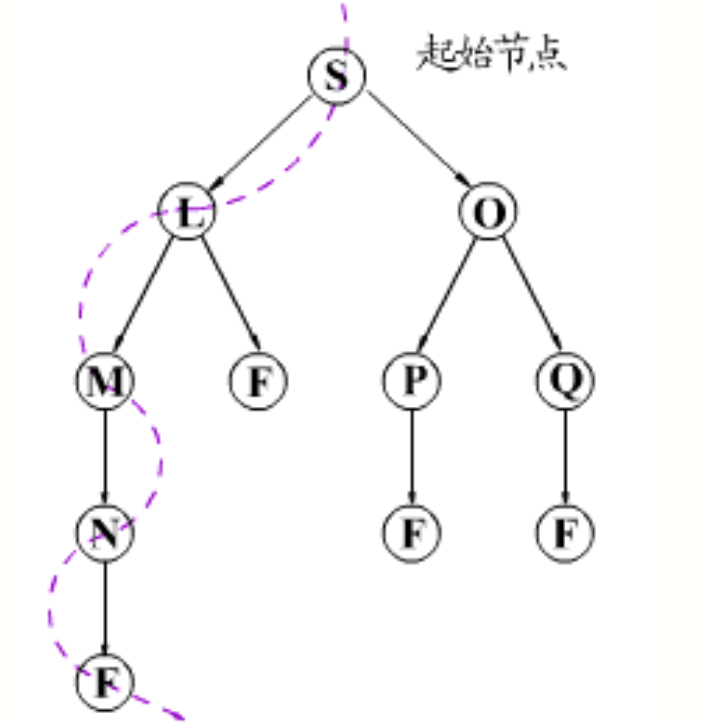
八数码难题的宽度优先搜索树

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。

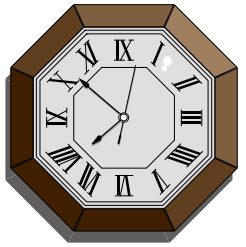
定义-节点的深度：

(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。

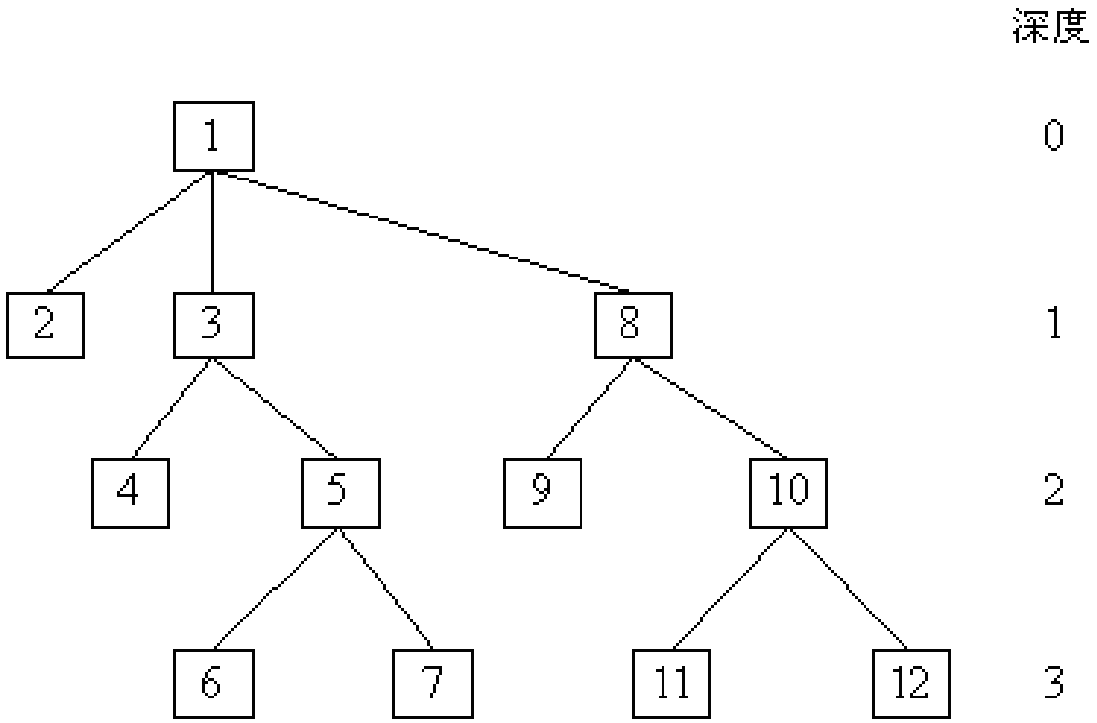
◇与宽度优先搜索算法最根本的不同：将扩展的后继节点放在OPEN表的前端。

算法的具体过程



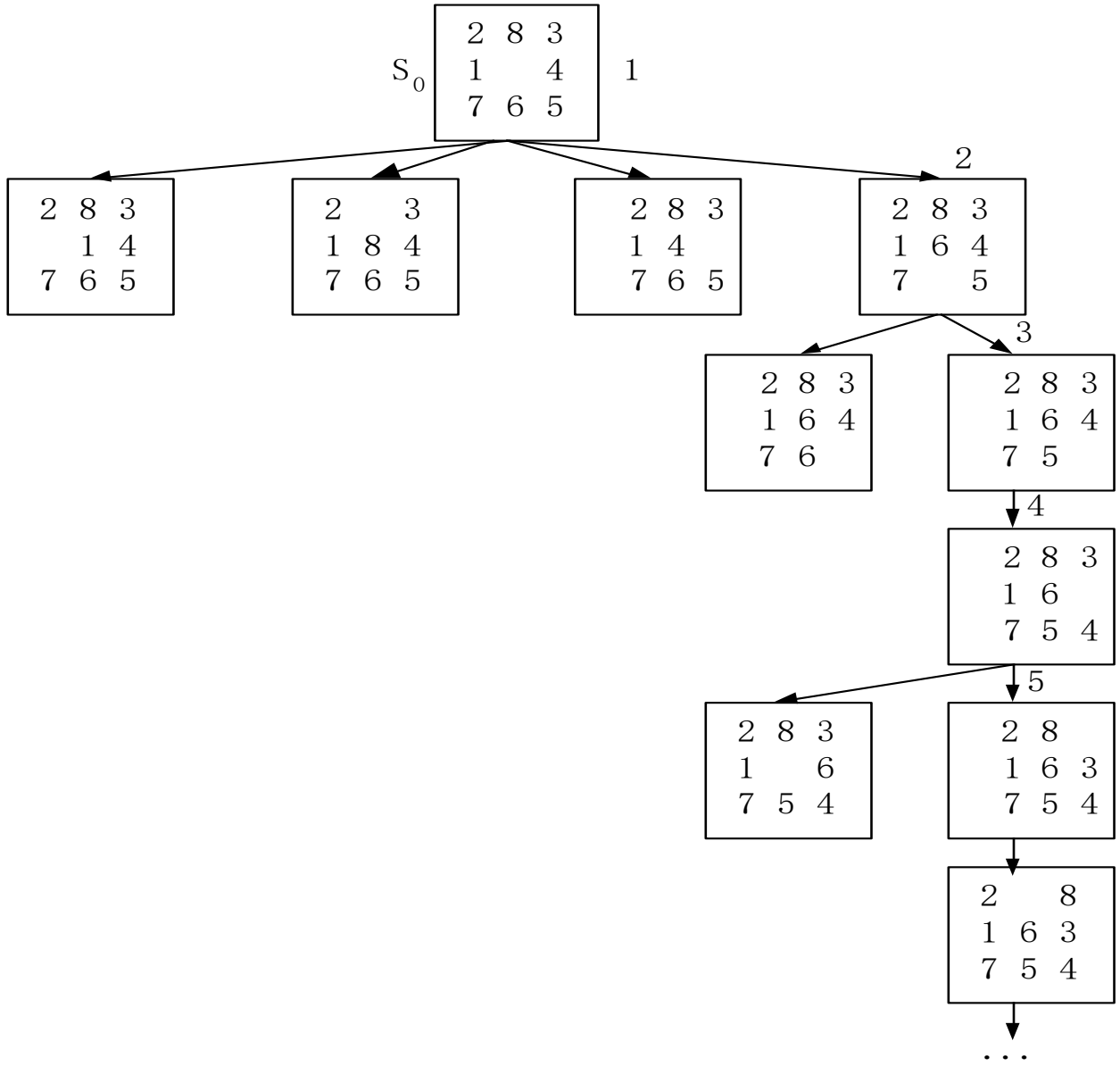
1. 把起始节点 S放到未扩展节点 OPEN表中。如果此节点为一目标节点，则得到一个解。
2. 如果 OPEN为一空表，则失败退出。
3. 把第一个节点（节点n）从 OPEN表移到CLOSED表。
4. 如果节点 n的深度等于最大深度，则转向(2)
5. 扩展节点n，产生其全部后裔，并把它们放入OPEN表的前端，如果没有后裔，则转向(2)
6. 后继节点是否为目标节点。若是，则求得了问题的解，退出；否则转第（2）步.

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


S_0
2 8 3
1 4
7 6 5
1
2 8 3
1 4
7 6 5
2 8 3
1 8 4
7 6 5
2 8 3
1 4
7 6 5
2
2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 6
3
2 8 3
1 6 4
7 5
4
2 8 3
1 6
7 5 4
5
2 8 3
1 6
7 5 4
2 8
1 6 3
7 5 4
2 8
1 6 3
7 5 4
...

有界深度优先搜索

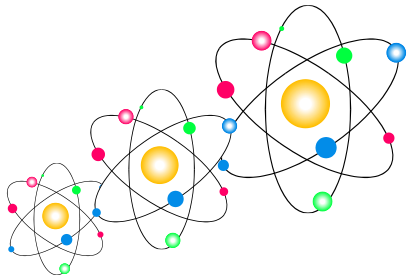
- 基本思想：

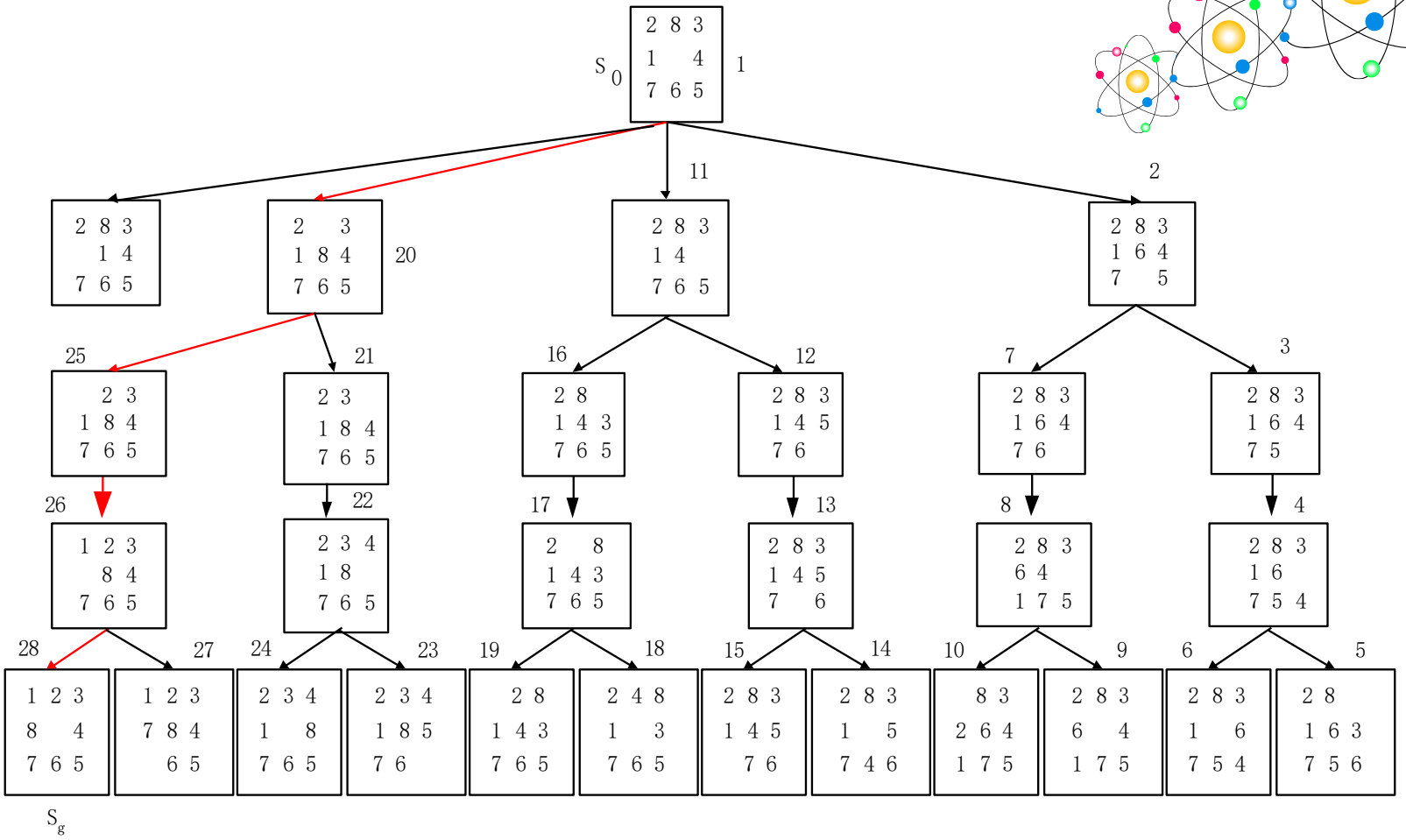
  对深度优先搜索引入搜索深度的界限（设为d_m），当搜索深度达到了深度界限，而仍未出现目标节点时，就换一个分支进行搜索。

有界深度优先搜索

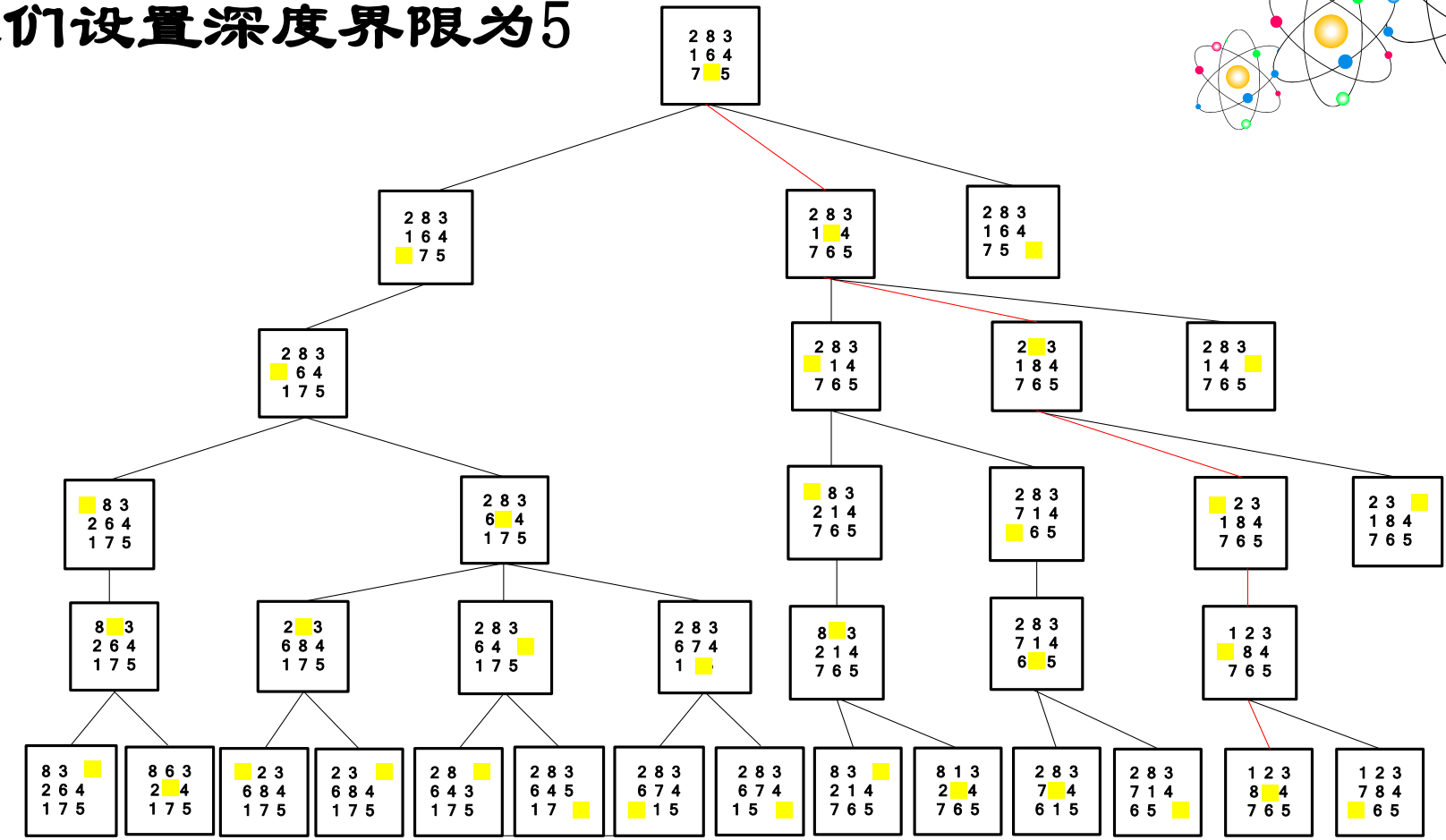
- 如果问题有解，且其路径长度$\leqslant d_m$，则上述搜索过程一定能求得解。但是，若解的路径长度$>d_m$，则上述搜索过程就得不到解。这说明在有界深度优先搜索中，深度界限的选择是很重要的。
- 要恰当地给出d_m的值是比较困难的。即使能求出解，它也不一定是最优解。

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





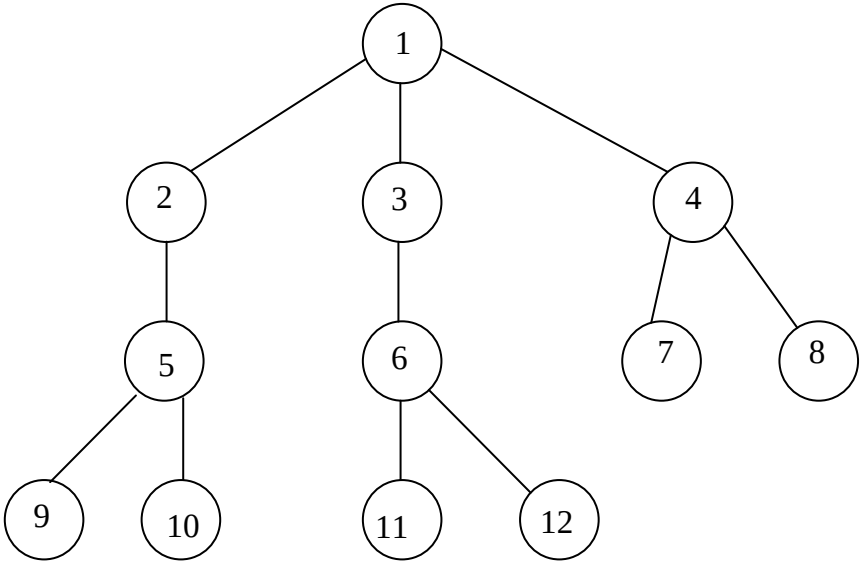
例：按深度优先搜索生成的八数码难题搜索树，我们设置深度界限为5



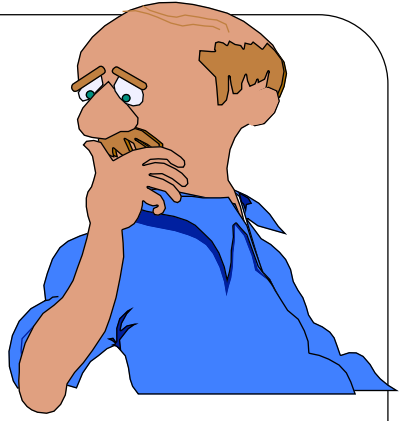
练习

(1)请根据深度优先搜索策略列出图中树的节点的访问序列(在目标节点为节点8，所有情况中都选择最左分支优先访问)。

(2)请写出根据深度优先搜索策略搜索时从初始状态迭代至搜索结束的过程中OPEN表和CLOSED表所存储的内容。



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。

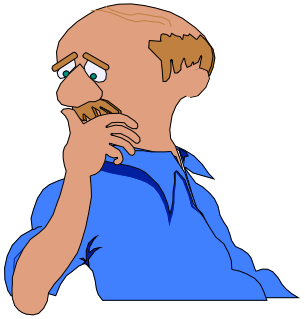
- 等代价搜索中的几个记号

  ◇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)

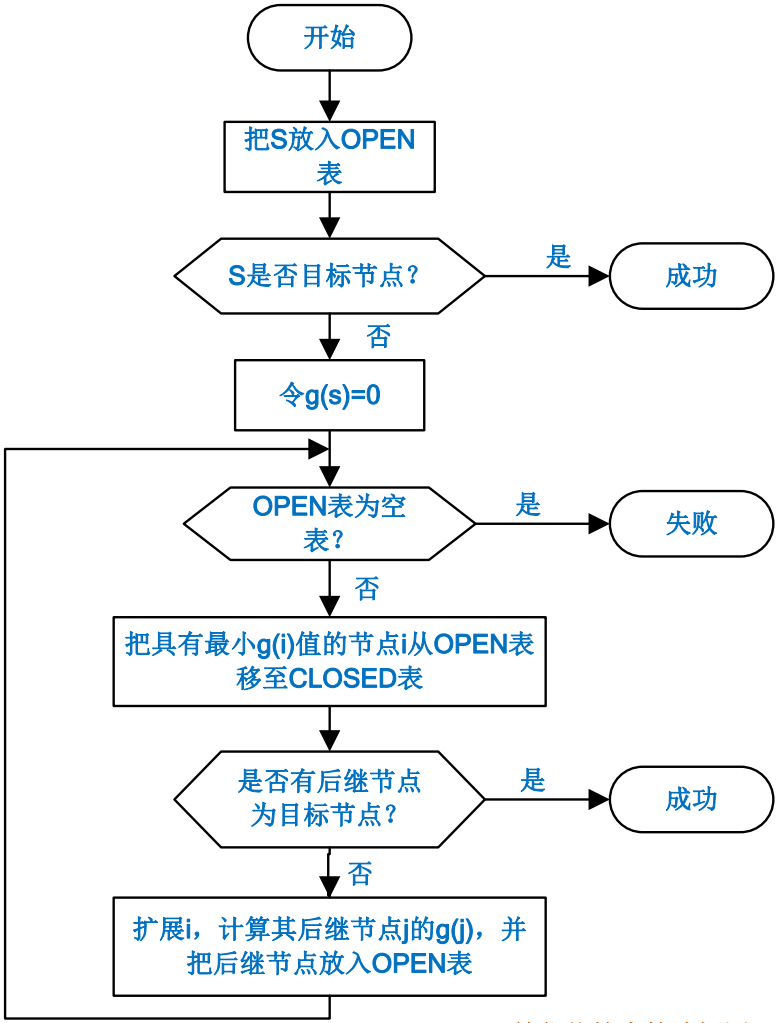
  ◇从起始节点S到任一节点i的路径代价记为g(i) 。

等代价搜索算法



1. 把起始节点S放到未扩展节点表OPEN中。如果此起始节点为一目标节点,则求得一个解；否则令g(S)=0;
2. 如果OPEN是个空表，则没有解而失败退出;
3. 从OPEN表中选择一个节点i，使其g(i)为最小.如果有几个节点都合格，那么就要选择一个目标节点作为节点i(要是有目标节点的话)；否则,就从中选一个作为节点i。把节点i从OPEN表移至扩展节点表CLOSED中;
4. 如果节点i为目标节点，则求得一个解;
5. 扩展节点i。如果没有后继节点，则转向第(2)步;
6. 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. 提供回到节点i的指针;
7. 转向第(2)步。

等代价搜索算法框图



等代价搜索算法框图

代价树的广度优先搜索

- 基本思想：

  每次从OPEN表中选择节点往CLOSED表传送时，总是选择其中代价最小的节点。也就是说，OPEN表中的节点在任一时刻都是按其代价从小到大排序的。代价小的节点排在前面，代价大的节点排在后面。

- 如果问题有解，代价树的广度优先搜索一定可以求得解，并且求出的是最优解。

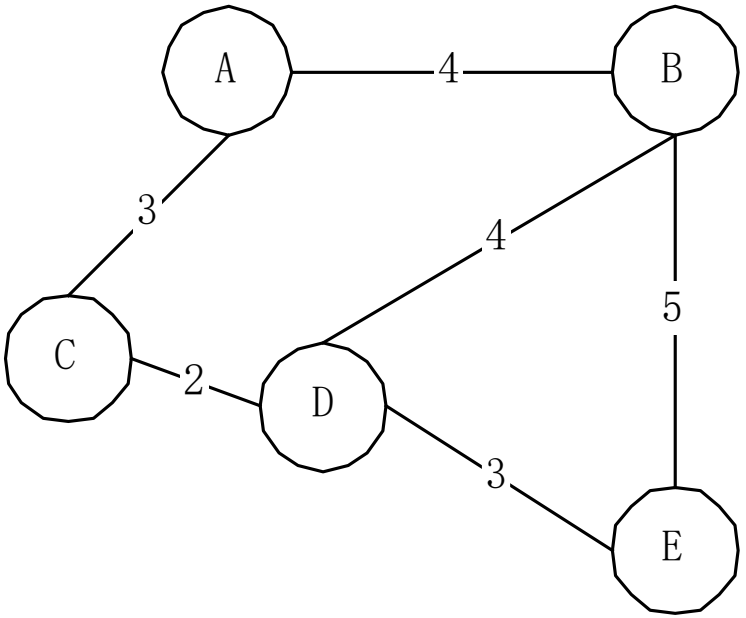
代价树的广度优先搜索

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，为每一个子节点都配置指向父节点的指针，计算各子节点的代价，并将各子节点放入OPEN表中。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。

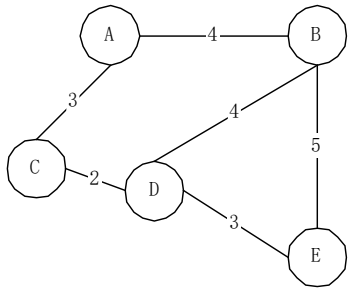
代价树的深度优先搜索

- 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。

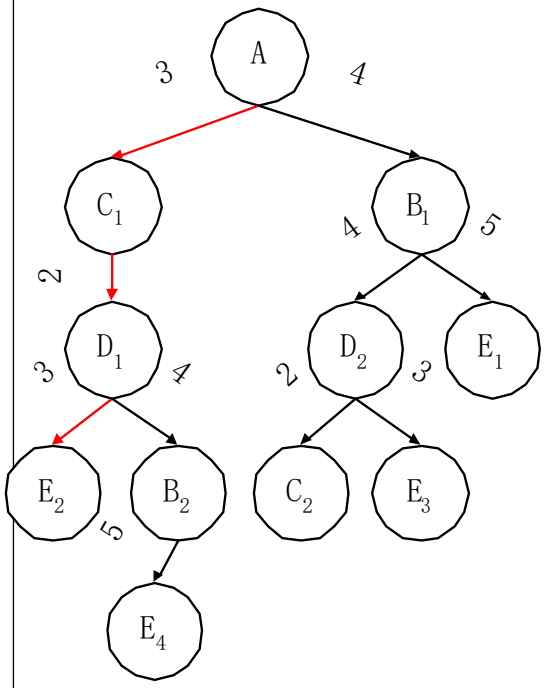
代价树搜索举例



交通图



交通图



交通图的代价树

在图所示的代价树中，首先对A进行扩展，得到B1和C1，由于C1的代价小于的B1代价，所以首先把C1送入CLOSED表进行考察。此时代价树的宽度优先搜索与代价树的深度优先搜索是一致的。

但往下继续进行时，两者就不一样了。对进行C1扩展得到D1，D1的代价为5。此时OPEN表中有D1和B1，B1的代价为4。若按代价树的宽度优先搜索方法进行搜索，应选B1送入CLOSED表，但按代价树的深度优先搜索方法，则应选D1送入CLOSED表。扩展后D1，再选E2，到达了目标节点。

所以，按代价树的深度优先搜索方法，得到的解路径是A→C1→D1→E2代价为8，**这与宽度优先搜索得到的结果相同，但这只是巧合，一般情况下这两种方法中得到的结果不一定相同**

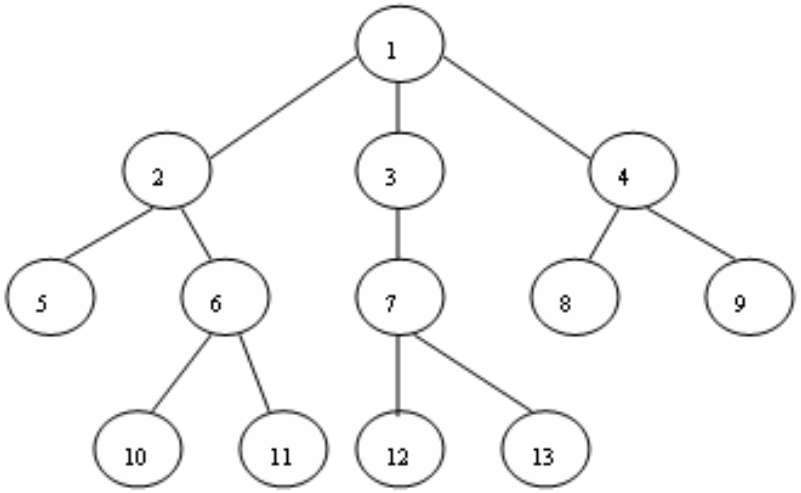
在**代价树的宽度优先搜索**中，每次都是从OPEN表的**全体节点中选择一个代价最小的节点**送入CLOSED表进行观察，而**代价树的深度优先搜索**是**从刚扩展出的子节点中选一个代价最小的节点**送入CLOSED表进行观察。

盲目搜索

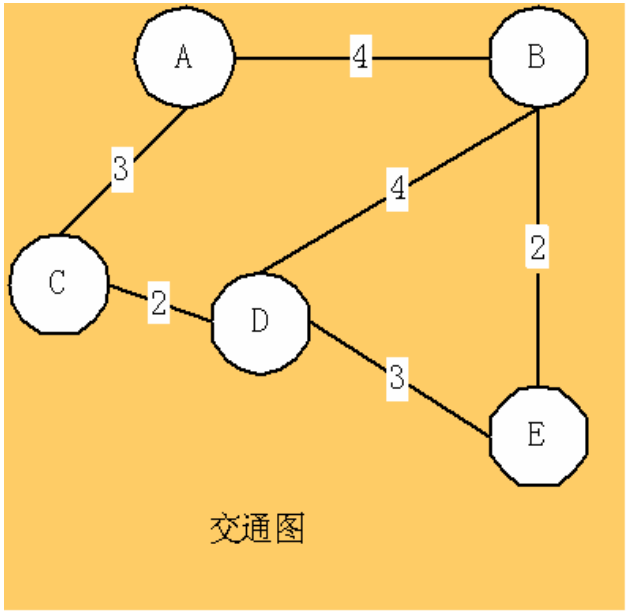
- 按照事先规定的路线进行搜索
  - 广度优先搜索是按“层”进行搜索的，先进入OPEN 表的节点先被考察
  - 深度优先搜索是沿着纵深方向进行搜索的，后进入OPEN表的节点先被考察
- 按已经付出的代价决定下一步要搜索的节点
  - 代价树的广度优先
  - 代价树的深度优先

作业题

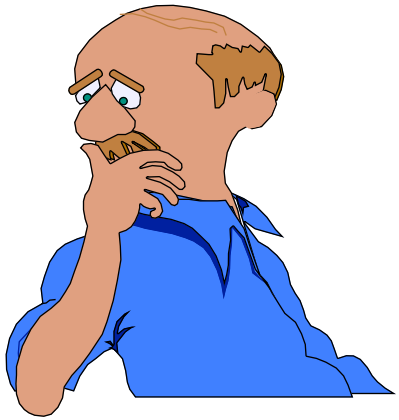
- 1 补充题：列出图中树的节点访问序列以满足下面的两个搜索策略，并写出其搜索过程中的open和closed表（在所有情况中都选择最左分枝优先访问，设节点12为目标节点）：
- （1）深度优先搜索；
- （2）宽度优先搜索。



- 2 补充题：根据右边的交通图，分别利用代价树的广度优先搜索和代价树的深度优先搜索求出从A点出发到达E点的路径。



启发式搜索



- 为什么需要启发式搜索
- ◇盲目搜索的不足
- 效率低，耗费过多的计算空间与时间
- 可能带来组合爆炸
- 分析前面介绍的宽度优先、深度优先搜索，或等代价搜索算法，其主要的差别是OPEN表中待扩展节点的顺序问题。人们就试图找到一种方法用于排列待扩展节点的顺序，即选择最有希望的节点加以扩展，那么，搜索效率将会大为提高.
- ◇什么可以做为启发信息？
- 进行搜索技术一般需要某些有关具体问题的领域的特性信息，把此种信息叫做启发信息。

启发式搜索

- ◇特点
- 重排OPEN表，选择最有希望的节点加以扩展
- ◇种类
- 有序搜索、A*算法等

启发式搜索的估价函数

- 定义

  ◇ 估价函数（evaluation function），估算节点希望程度的量度

  ◇ 表示方法

- f(n)——表示节点n的估价函数值

建立估价函数的一般方法

- ◇提出任意节点与目标集之间的距离量度或差别量度
- ◇在棋盘式的博弈和难题中根据棋局的某些特点来决定棋局的得分数。 这些特点被认为与向目标节点前进一步的希望程度有关

有序搜索

- 概念
- ◇有序搜索（ordered search），即最好优先搜索（best-first search）。
- ◇选择OPEN表上具有最小f值的节点作为下一个要扩展的节点。

有序搜索算法

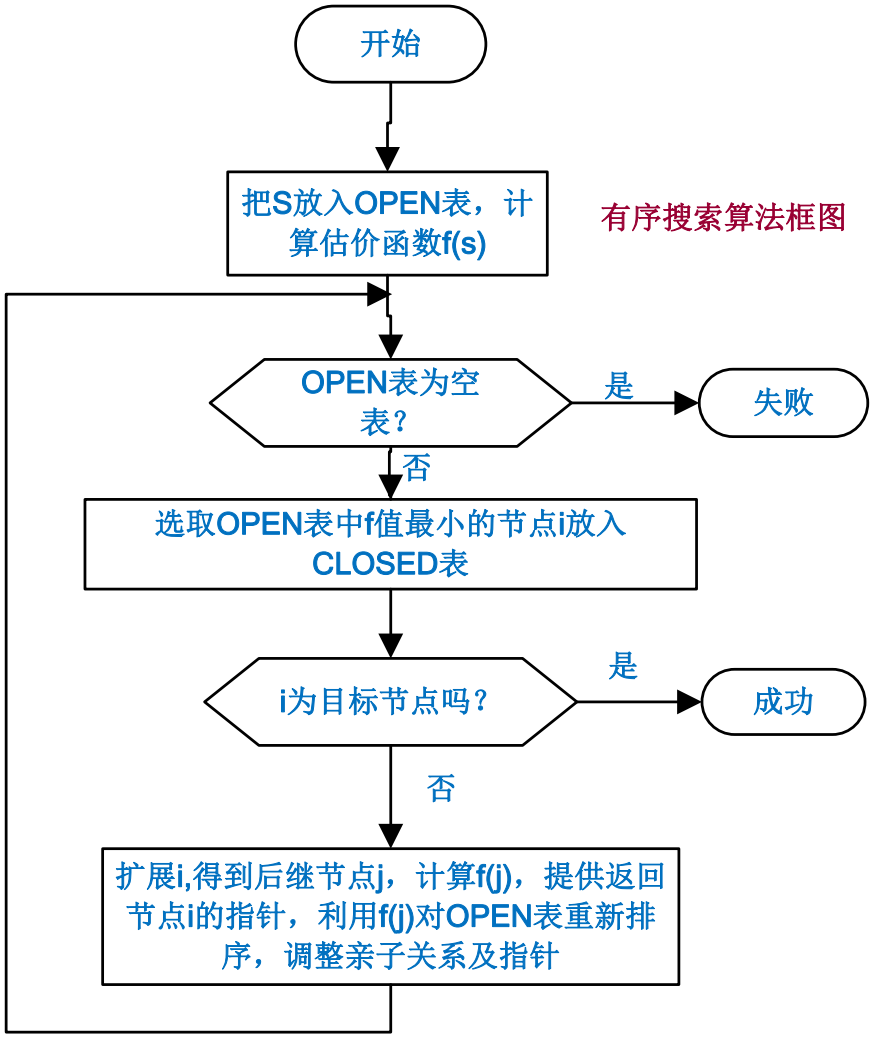
■ 算法

1. 把起始节点S放在OPEN表中，计算f(s)并把其值与节点S联系起来。
2. 如果OPEN是个空表，则失败退出，无解。
3. 从OPEN表中选择一个f值最小的节点i。结果有几个节点合格，当其中有一个为目标节点时，则选择此目标节点，否则就选择其中任一个节点作为节点i。
4. 把节点i从OPEN表中移出，并把它放入CLOSED的扩展节点表中。
5. 如果i是个目标节点，则成功退出，求得一个解。
6. 扩展节点i，生成其全部后继节点，对于i的每一个后继节点j:
    1. 计算f(j)。
    2. 如果j既不在OPEN表中，又不在CLOSED表中，则用估价函数f把它添入OPEN表，从j加一指向其父辈节点i的指针，以便一旦找到目标节点时记住一个解答路径。
    3. 如果j已在OPEN表上或CLOSED表上，则比较刚刚对j计算过的f值和前面计算过的该节点在表中的f值。

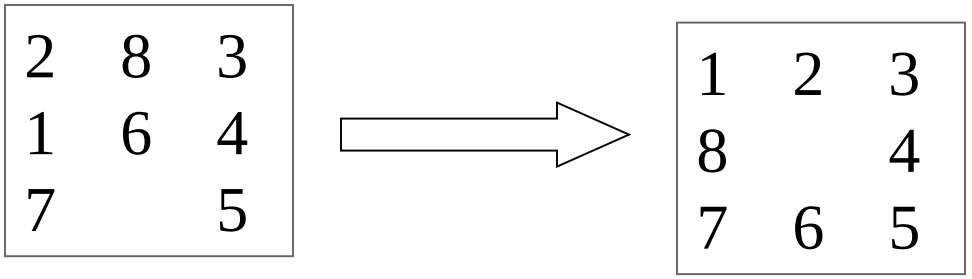
    如果新的f值较小，则

    1. 以此新值取代旧值。
    2. 从j指向i，而不是指向它的父辈节点。
    3. 如果节点j在CLOSED表中，则把它移回OPEN表

有序搜索算法框图


开始
把S放入OPEN表，计算估价函数f(s)
有序搜索算法框图
OPEN表为空表？
是
失败
否
选取OPEN表中f值最小的节点i放入CLOSED表
i为目标节点吗？
是
成功
否
扩展i,得到后继节点j，计算f(j)，提供返回节点i的指针，利用f(j)对OPEN表重新排序，调整亲子关系及指针

一个算法的例子



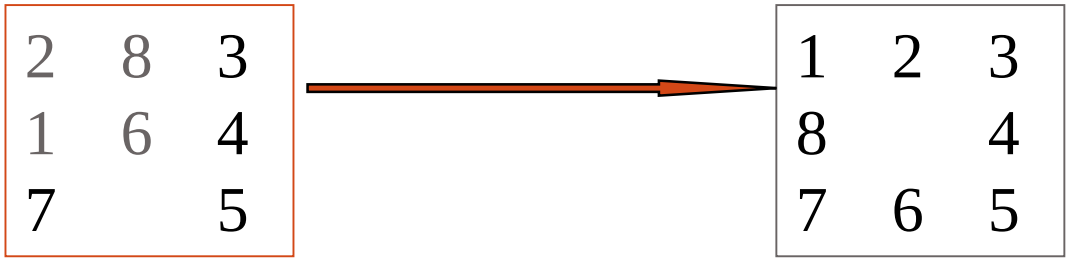
定义评价函数：

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

$g(n)$为从初始节点到当前节点的路径长度（深度）

$h(n)$为当前节点“不在位”的将牌数

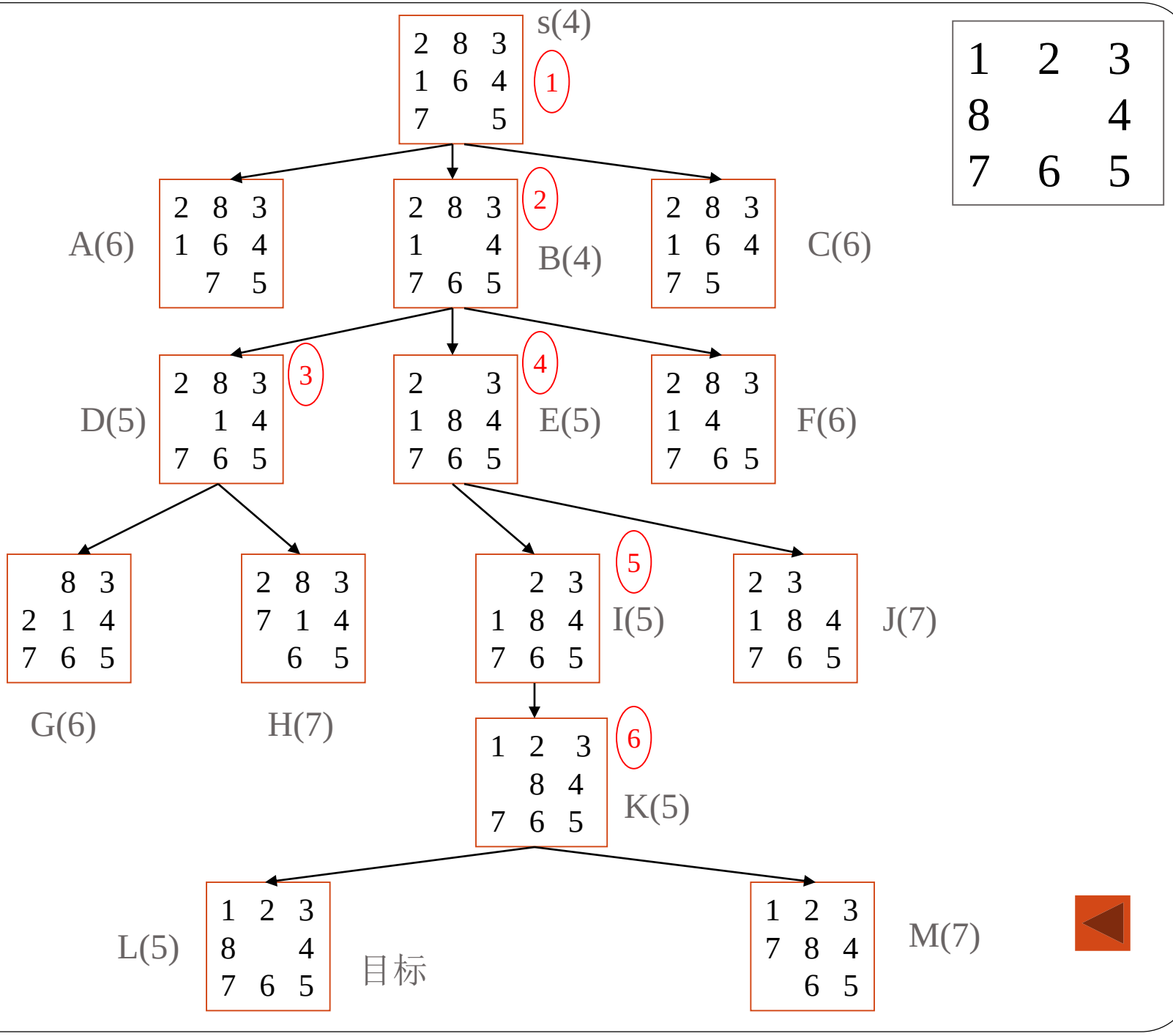
h计算举例



h(n) =4

g(n)=0

f(n)=4+0=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)
1 2 3
8 4
7 6 5

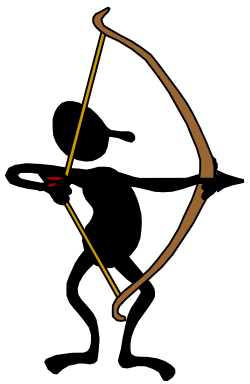
A*算法

- 算法特点
- ◇隶属于有序搜索算法的一种；
- ◇其特点在于对估价函数的定义上.
- 算法符号
- $k(n_i, n_j)$
- ◇表示任意两个节点ni和和nj之间最小代价路径的实际代价；
- ◇对于两节点间没有通路的节点，函数k没有定义；
- ◇从节点n到某个具体的目标节点t_i，某一条最小代价路径的代价可由$k(n, t_i)$给出。

◇ $h^*(n)$ 表示整个目标节点集合 $\{t_i\}$ 上所有 $k(n, t_i)$ 中最小的一个；

◇对于任何不能到达目标节点的节点n，函数h*没有定义。

◇ $h^*(n)$ 就是从n到目标节点最小代价路径的代价，而且从n到目标节点能够获得到 $h^*(n)$ 的任一路径就是一条从n到某个目标节点的最佳路径；

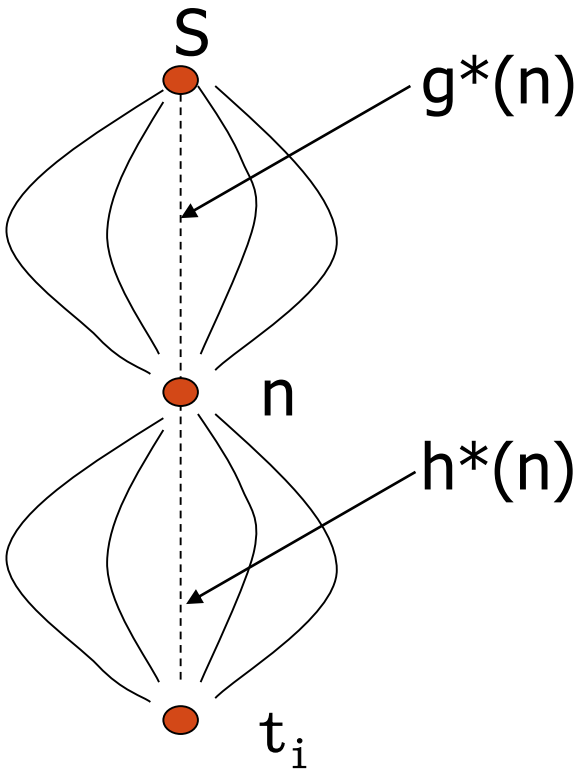


估价函数的设计

h*(n)的估计h(n)依赖于有关问题的领域的启发信息。h叫做启发函数

- 定义g*： g*(n)=k(S, n)；
- 定义f*： f*(n)=g*(n)+h*(n)；
- 希望估价函数f是f*的一个估计：
  f(n)=g(n)+h(n)
- （g是g*的估计, h是h*的估计）.

对于g(n)来说，一个明显的选择就是搜索树中从S到n这段路径的代价，这一代价可以由从n到S寻找指针时，把所遇到的各段弧线的代价加起来给出(这条路径就是到目前为止用搜索算法找到的从S到n的最小代价路径))。这个定义包含了g(n)≥g*(n)


S
g*(n)
n
h*(n)
t_i

A*算法

定义1

- 在GRAPHSEARCH过程中，如果第8步的重排OPEN表是依据f(n)=g(n)+h(n)进行的，则称该过程为A算法；

定义2

- 在A算法中，如果对所有的n存在h(n)≤h*(n)，则称h(n)为h*(n)的下界，它表示某种偏于保守的估计；

定义3

采用h*(n)的下界h(n)为启发函数的A算法，称为A*算法。

A*算法过程

(1) 把S放入OPEN表，记f=h，令CLOSED为空表。

(2) 重复下列过程，直至找到目标节点止。若OPEN为空表，则宣告失败。

(3) 选取OPEN表中未设置过的具有最小f值的节点为最佳节点BESTNODE，并把它放入CLOSED表。

(4) 若BESTNODE为一目标节点，则成功求得一解。

(5) 若BESTNODE不是目标节点，则扩展之，产生后继节点SUCCSSOR。

(6) 对每个SUCCSSOR进行下列过程：

(a) 建立从SUCCSSOR返回BESTNODE的指针。

(b) 计算g(SUC)=g(BES)+k(BES, SUC)。

(c) 如果SUCCSSOR∈OPEN，则称此节点为OLD，并把它添至BESTNODE的后继节点表中。

(d) 比较新旧路径代价。如果g(SUC)＜g(OLD)，则重新确定OLD的父辈节点为BESTNODE，记下较小代价g(OLD)，并修正f(OLD)值。

(e) 若至OLD节点的代价较低或一样，则停止扩展节点。

(f) 若SUCCSSOR不在OPEN表中，则看其是否在CLOSED表中。

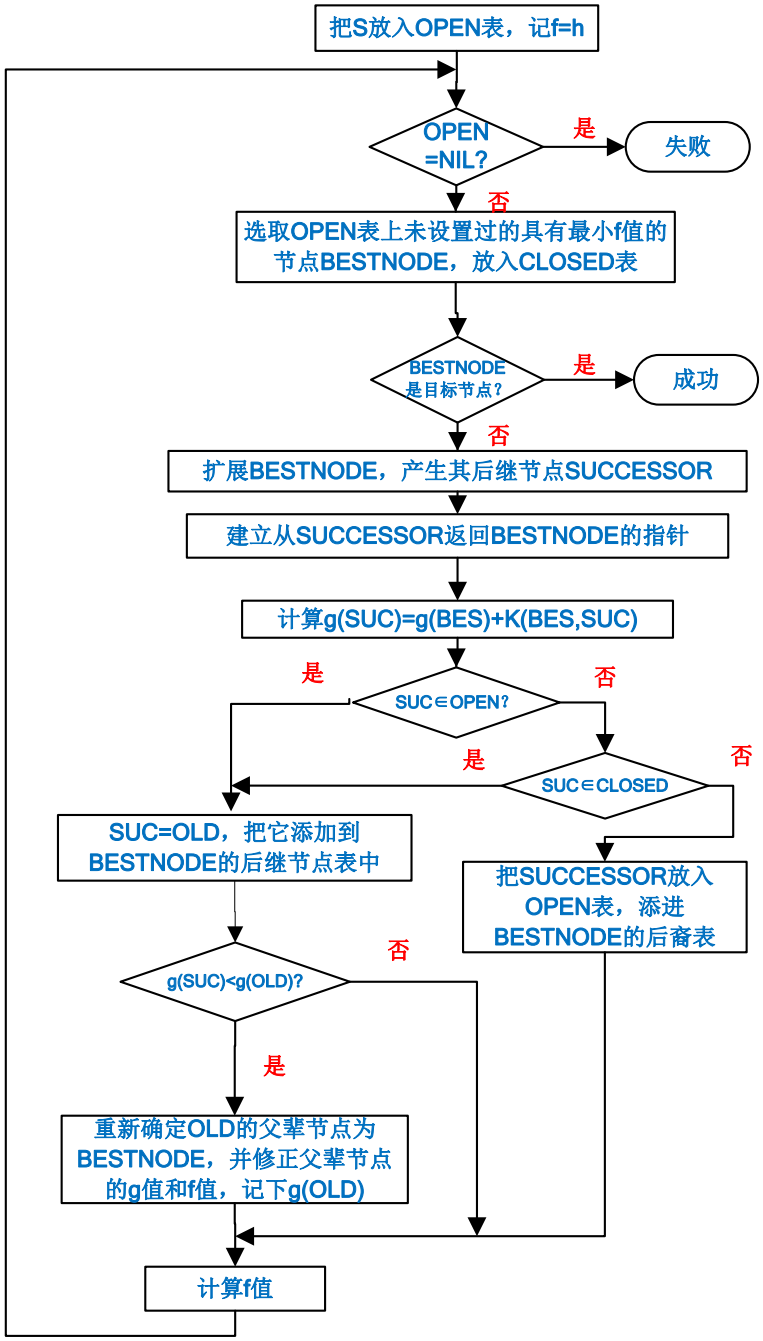
(g) 若SUCCSSOR在CLOSE表中，则转向c。

(h) 若SUCCSSOR既不在OPEN表中，又不在CLOSED表中，则把它放入OPEN表中，并添入BESTNODE后裔表，然后转向(7)

(7) 计算f值。

(8) GO LOOP

A*算法


把S放入OPEN表，记f=h
OPEN =NIL?
是
失败
否
选取OPEN表上未设置过的具有最小f值的节点BESTNODE，放入CLOSED表
BESTNODE是目标节点?
是
成功
否
扩展BESTNODE，产生其后继节点SUCCESSOR
建立从SUCCESSOR返回BESTNODE的指针
计算g(SUC)=g(BES)+K(BES,SUC)
是
SUC∈OPEN?
否
是
SUC∈CLOSED
否
SUC=OLD，把它添加到BESTNODE的后继节点表中
把SUCCESSOR放入OPEN表，添进BESTNODE的后裔表
否
g(SUC)<g(OLD)?
是
重新确定OLD的父辈节点为BESTNODE，并修正父辈节点的g值和f值，记下g(OLD)
计算f值

启发式搜索小结

- 启发式策略就是利用与问题有关的启发信息进行搜索的策略。
- 利用估价函数来估计待搜索节点的希望程度，并以此排序。估价函数一般由两部分组成：
- $$f(n)=g(n)+h(n)$$
- $g(n)$是从初始节点到节点n已付出的实际代价，$h(n)$是从节点n到目的节点的最佳路径的估计代价。$h(n)$体现了搜索的启发信息。
- 在$f(n)$中，$g(n)$的比重越大，越倾向于宽度优先搜索，而$h(n)$的比重越大，表示启发性越强。

启发式搜索小结

- $f(n)=g(n)+h(n)$
- 在$f(n)$中，$g(n)$的比重越大，越倾向于宽度优先搜索，而$h(n)$的比重越大，表示启发性越强。
- $g(n)$的作用一般是不可忽略的，保持$g(n)$项就保持了搜索的宽度优先成分，这有利于搜索的完备性，但会影响搜索的效率。

启发式搜索小结

- A算法
- 在GRAPHSEARCH过程中，如果第8步的重排OPEN表是依据f(n)=g(n)+h(n) 进行的,则称该过程为A算法;
- 显然，A算法对h(n) 没有明确的限制。
- A*算法
- 采用 h*(n)的下界h(n)为启发函数的A算法，称为A*算法。
- A*算法要求h(n)≤h*(n)。它表示某种偏于保守的估计。
- 如果算法有解，A*算法一定能够找到最优的解答。
- 一般说来，在满足h(n)≤h*(n)的前提下，h(n)的比重越大越好，h(n)的比重越大表示启发性越强。

例题

- 对于如图所示的八数码问题，给出满足A*算法的启发函数，并给出相应的搜索图。

$S_0 =$

2	8	3
1	6	4
7		5

$S_g =$

1	2	3
8		4
7	6	5

S_0 =

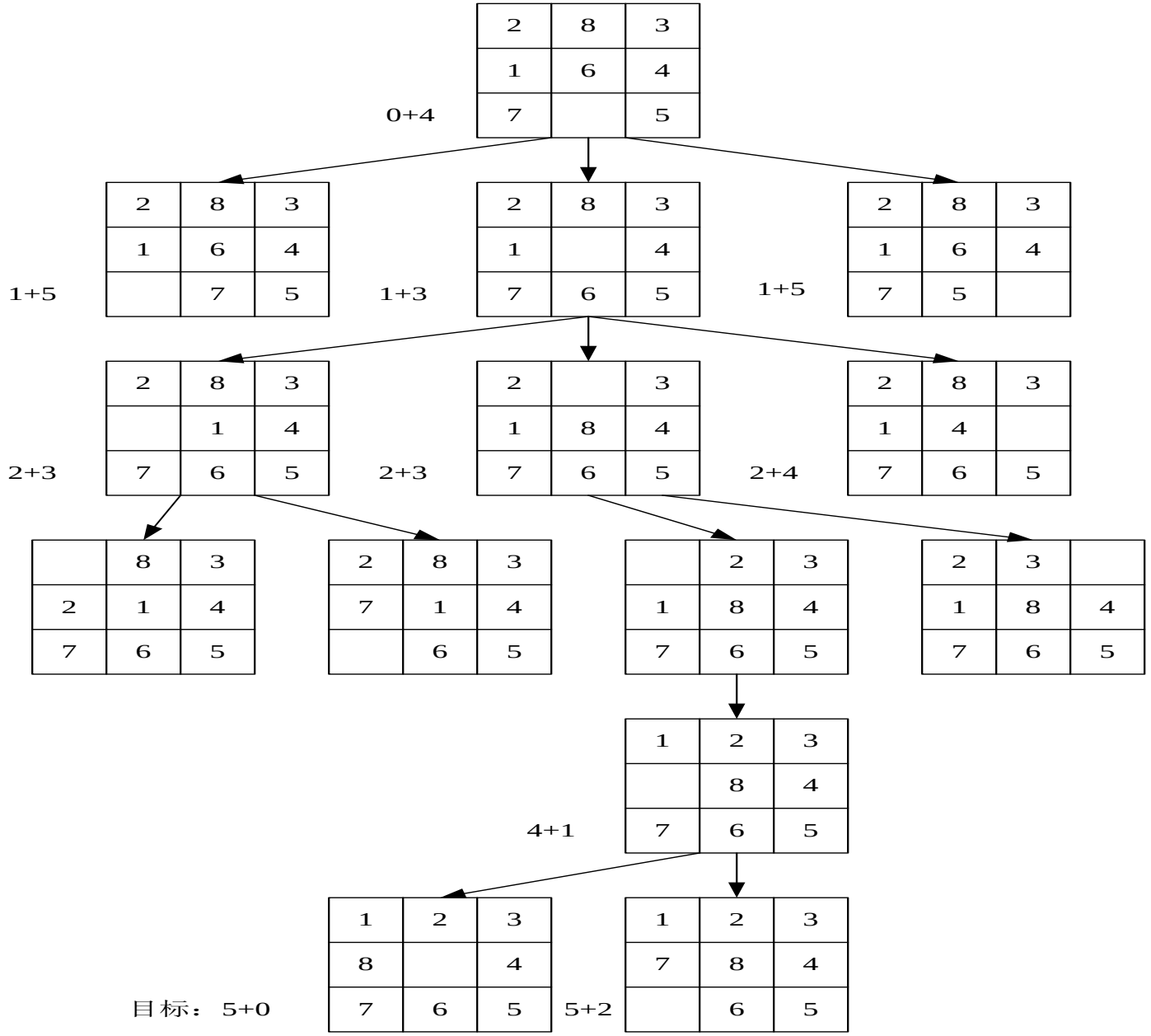
2	8	3
1	6	4
7		5

S_g =

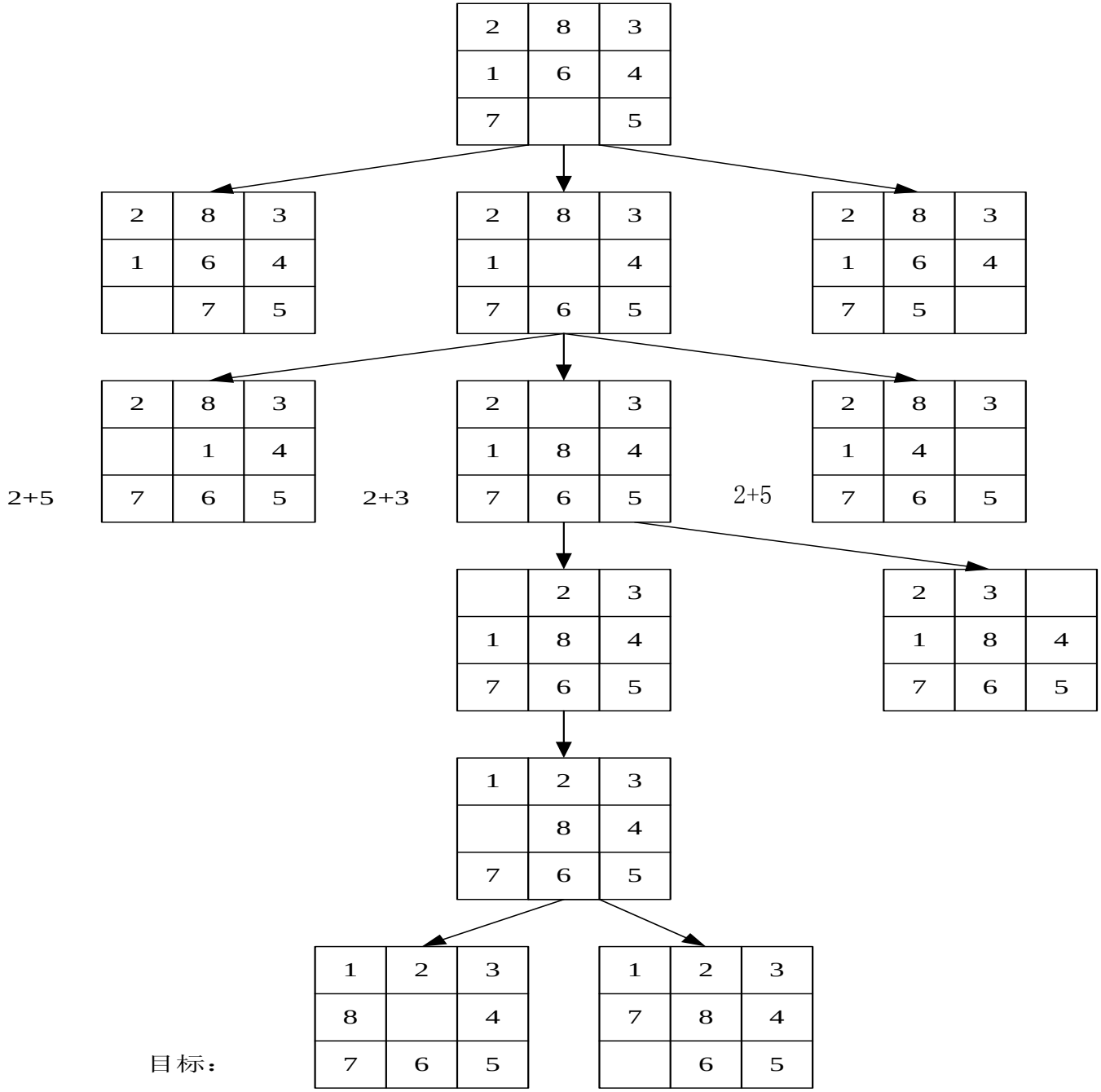
1	2	3
8		4
7	6	5

- 解：
- 启发函数的选取如下：$g(n)$表示节点n在搜索树中的深度，$h(n)=\omega(n)$表示节点n中不在目标状态中相应位置的数码个数，则根据

  $f(n)=\omega(n)+g(n)$，可以得到如图所示搜索过程。
- 在上面确定$h(n)$时，尽管并不知道$h^*(n)$具体为多少，但当采用单位代价时，通过对“不在目标状态中相应位置的数码个数”的估计，可以得出至少需要移动$h(n)$步才能够到达目标，显然$h(n)\leqslant h^*(n)$。因此它满足A*算法的要求。


2 8 3
1 6 4
7 5
0+4
2 8 3
1 6 4
7 5
1+5
2 8 3
1 4
7 6 5
1+3
2 8 3
1 6 4
7 5
1+5
2 8 3
1 4
7 6 5
2+3
2 8 3
1 8 4
7 6 5
2+3
2 8 3
1 4
7 6 5
2+4
8 3
2 1 4
7 6 5
2 8 3
7 1 4
6 5
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
1 2 3
8 4
7 6 5
4+1
1 2 3
8 4
7 6 5
目标：5+0
1 2 3
7 8 4
6 5
5+2

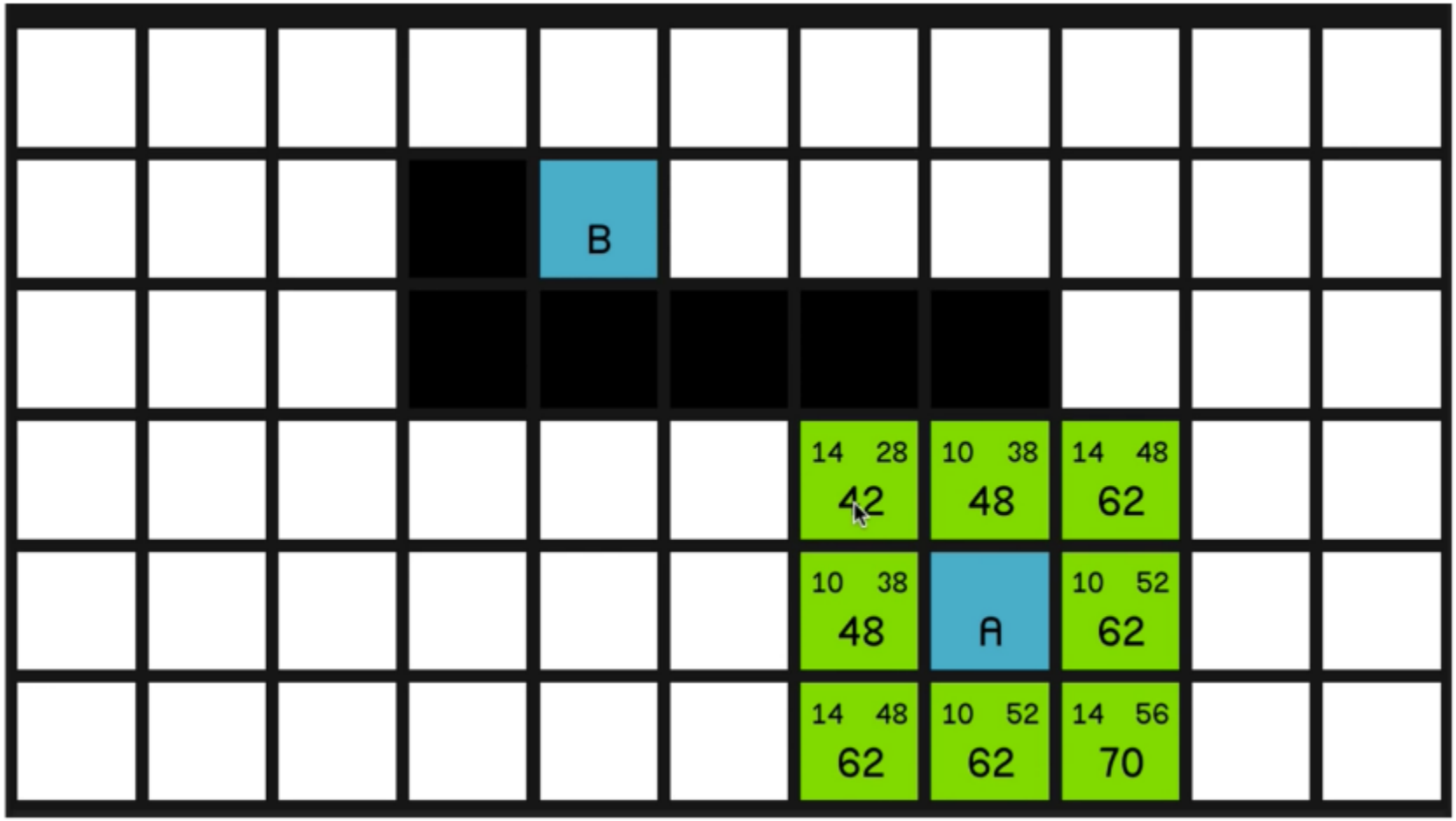
- 另解：
- 定义启发函数h(n)=p(n)为节点n的每一数码与其目标位置之间的距离总和。
- 显然有$\omega(n) \leqslant p(n) \leqslant h^*(n)$，相应的搜索过程也是A*算法。
- 然而，p(n)比$\omega(n)$有更强的启发性信息，由h(n)=p(n)构造的启发式搜索树，比$h(n)=\omega(n)$构造的启发式搜索树节点数要少。


2+5
2+3
2+5
目标：

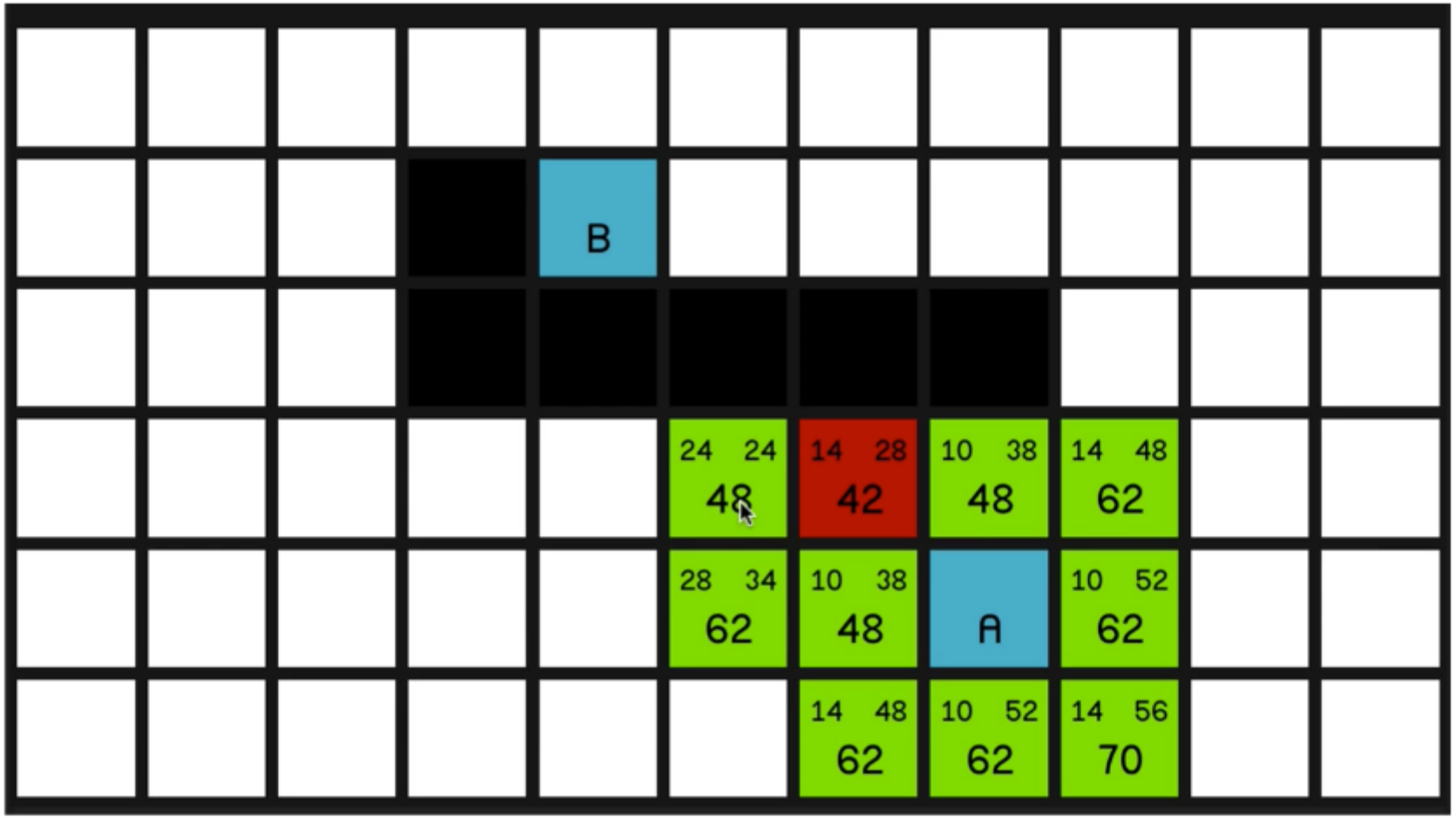
下面给出本题的启发函数的比较结果。

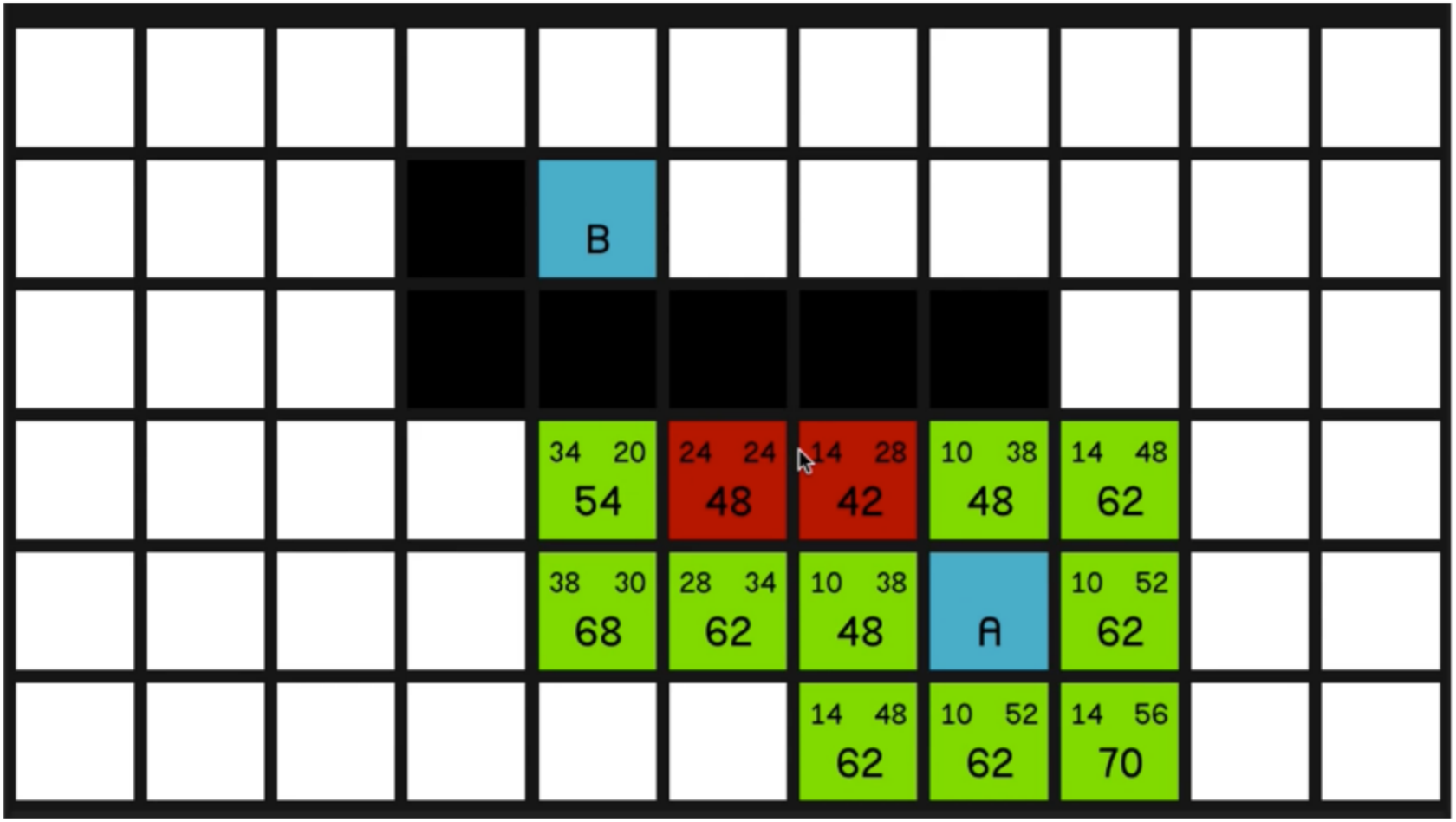
启发函数	h(n)=0	h(n)=ω(n)	h(n)=p(n)
扩展节点	26	6	5
生成节点	46	13	11

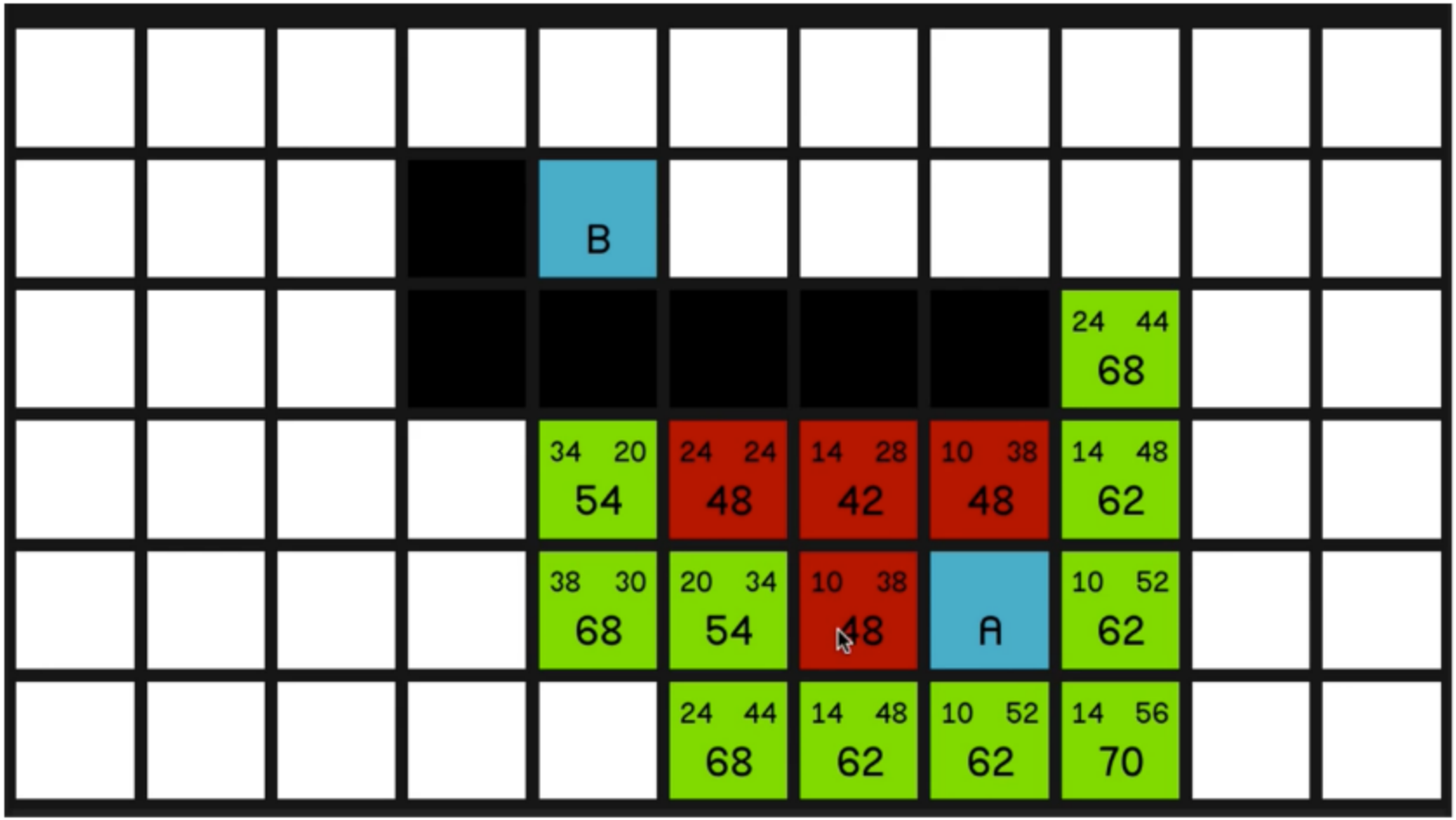
例2

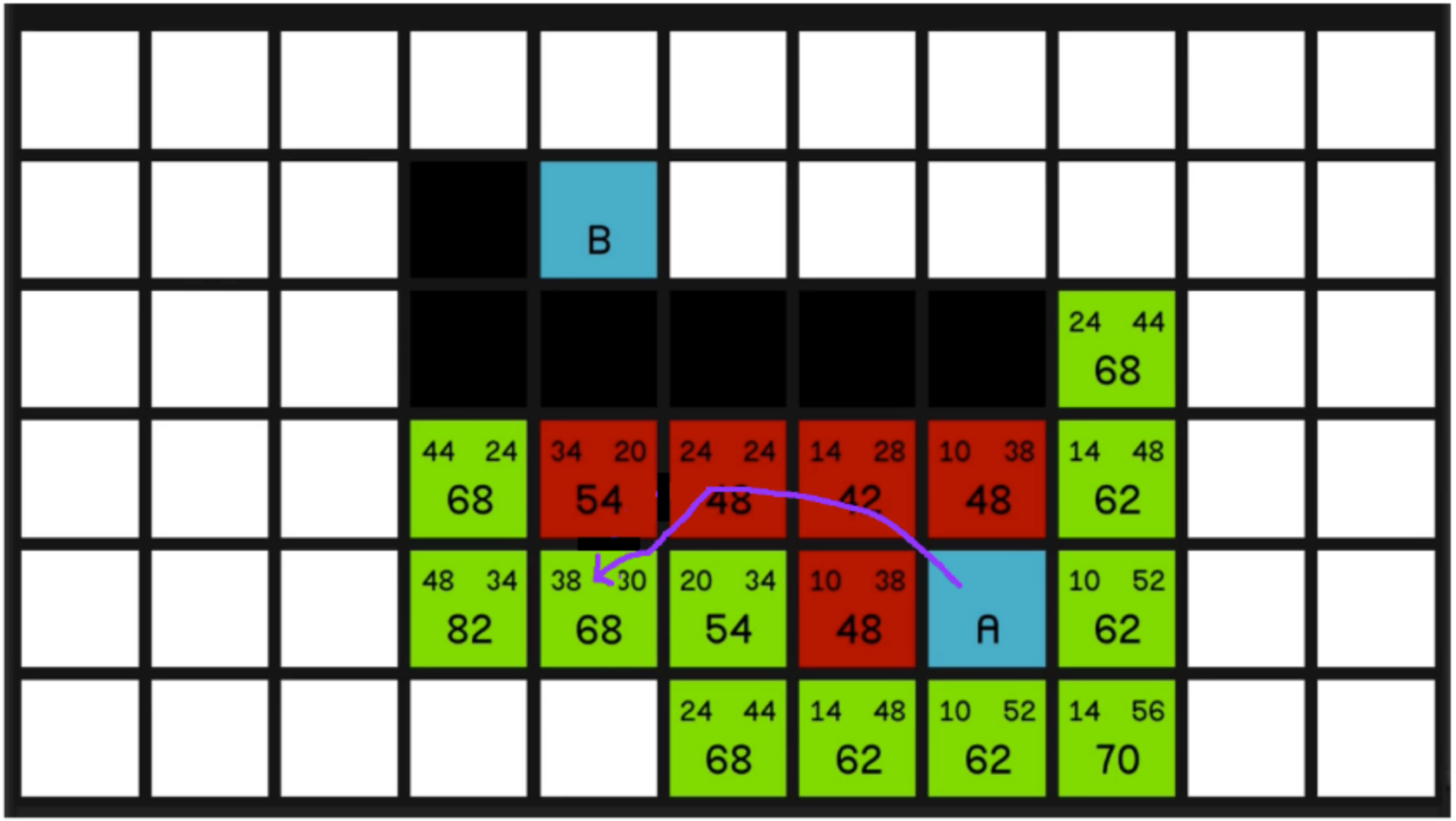


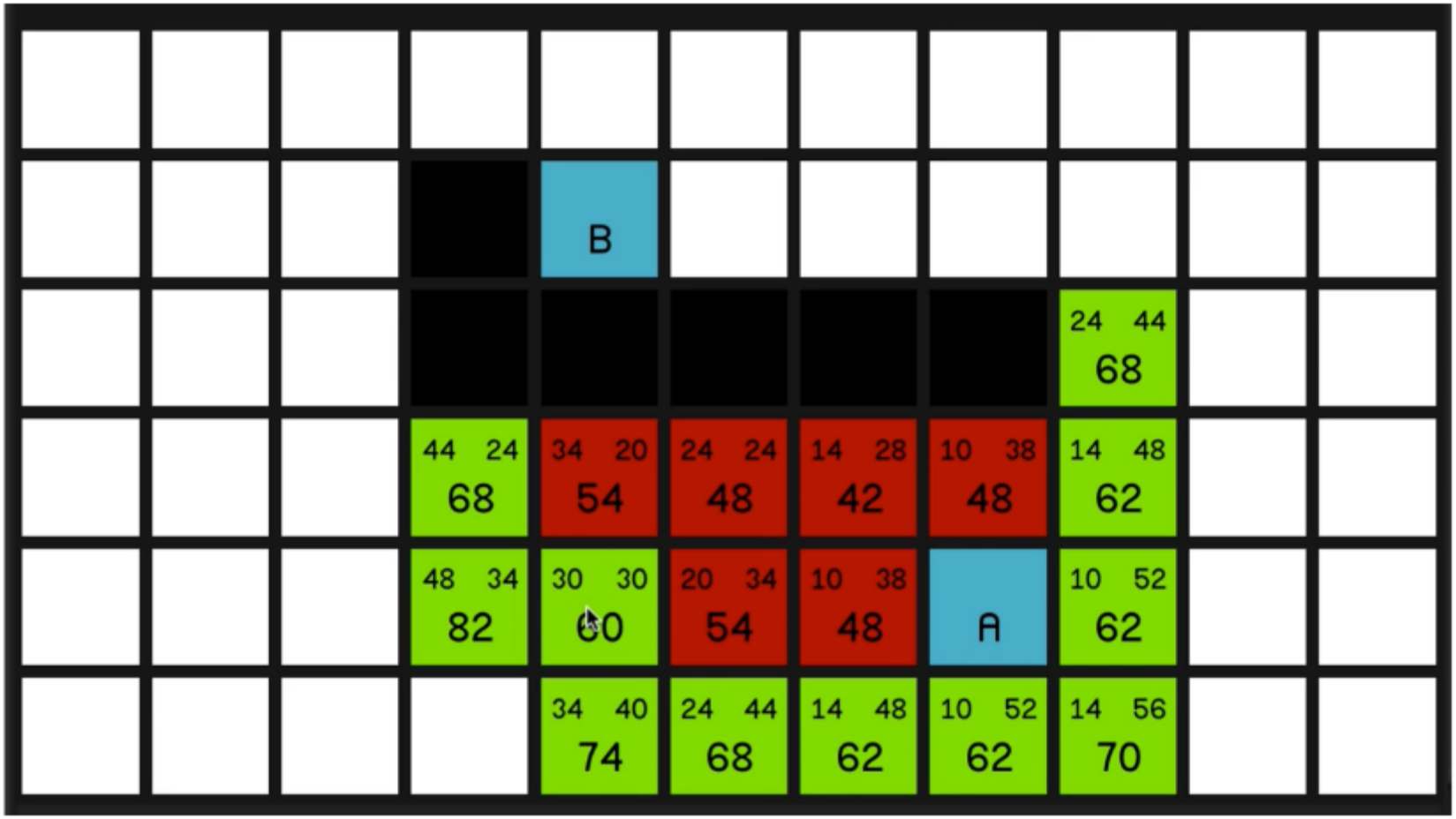
- 每个方块左上角的值记为G，表示该点到A的距离，右上角值为H，H取其到B的距离。F=H+G。

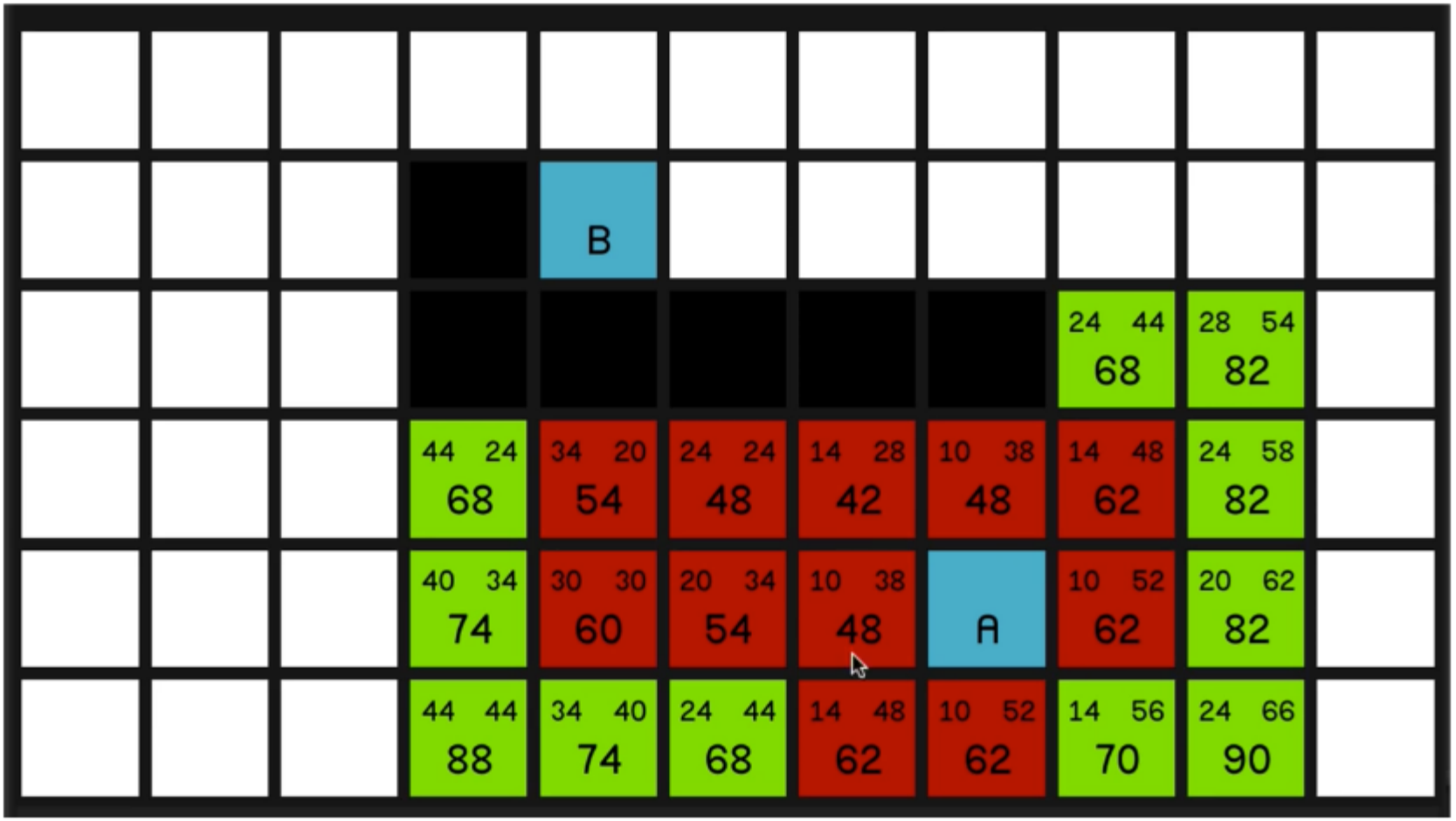

B
24 24
48
14 28
42
10 38
48
14 48
62
28 34
62
10 38
48
A
10 52
62
14 48
62
10 52
62
14 56
70

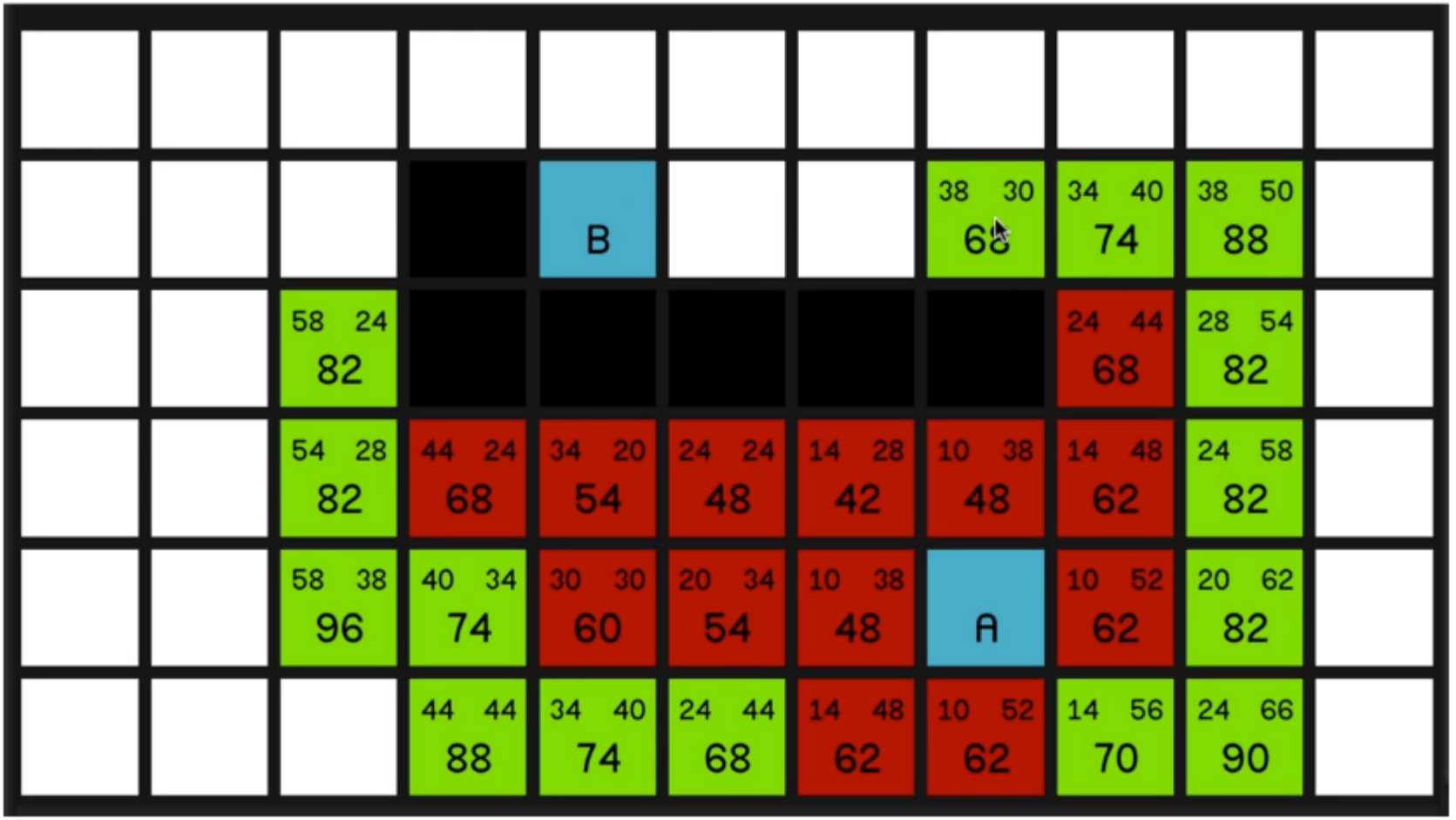

B
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
38 30
68
28 34
62
10 38
48
A
10 52
62
14 48
62
10 52
62
14 56
70

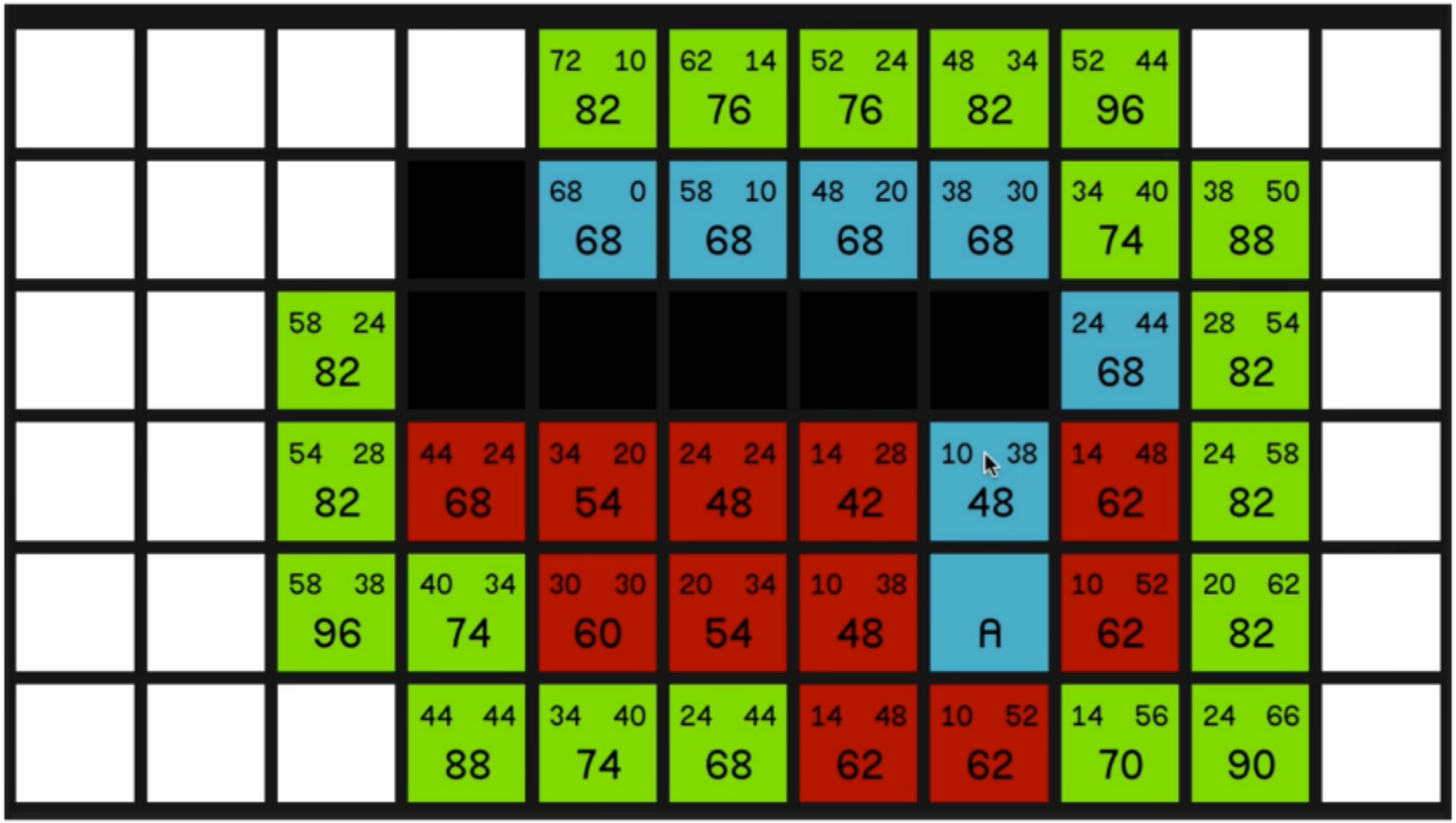

B
24 44
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
38 30
68
20 34
54
10 38
48
A
10 52
62
24 44
68
14 48
62
10 52
62
14 56
70


B
24 44
68
44 24
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
48 34
82
38 30
68
20 34
54
10 38
48
A
10 52
62
24 44
68
14 48
62
10 52
62
14 56
70


B
24 44
68
44 24
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
48 34
82
30 30
60
20 34
54
10 38
48
A
10 52
62
34 40
74
24 44
68
14 48
62
10 52
62
14 56
70

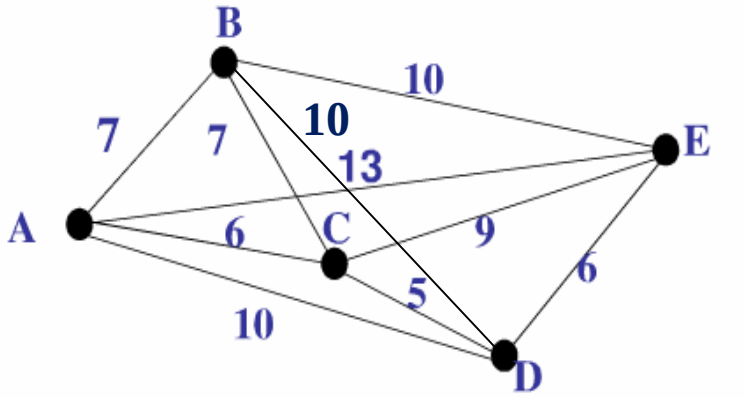

B
24 44
68
28 54
82
44 24
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
24 58
82
40 34
74
30 30
60
20 34
54
10 38
48
A
10 52
62
20 62
82
44 44
88
34 40
74
24 44
68
14 48
62
10 52
62
14 56
70
24 66
90


B
38 30
68
34 40
74
38 50
88
58 24
82
24 44
68
28 54
82
54 28
82
44 24
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
24 58
82
58 38
96
40 34
74
30 30
60
20 34
54
10 38
48
A
10 52
62
20 62
82
44 44
88
34 40
74
24 44
68
14 48
62
10 52
62
14 56
70
24 66
90


72 10 82
62 14 76
52 24 76
48 34 82
52 44 96
68 0 68
58 10 68
48 20 68
38 30 68
34 40 74
38 50 88
58 24 82
24 44 68
28 54 82
54 28 82
44 24 68
34 20 54
24 24 48
14 28 42
10 38 48
14 48 62
24 58 82
58 38 96
40 34 74
30 30 60
20 34 54
10 38 48
A
10 52 62
20 62 82
44 44 88
34 40 74
24 44 68
14 48 62
10 52 62
14 56 70
24 66 90

作业—1

- 下图表示了五个城市间的交通关系，用状态空间法规划一个最短的旅行路程：此旅程从城市A开始，访问其他城市不多于一次，并返回A。选择一个状态表示，表示出所求得的状态空间的节点及弧线，标出适当的代价，并指明图中从起始节点到目标节点的最佳路径。



小结

- 状态空间法是一种基于解答空间的问题表示和求解方法，它是以状态和操作符为基础的。在利用状态空间图表示时，我们从某个初始状态开始，每次加一个操作符，递增地建立起操作符的试验序列，直到达到目标状态为止。由于状态空间法需要扩展过多的节点，容易出现"组合爆炸"，因而只适用于表示比较简单的问题。

125

思考题

- 2-1. 什么是人工智能？试从学科和能力两方面加以说明。
- 2-2. 现在人工智能有哪些学派？它们的认知观是什么？
- 2-3. 人工智能的主要研究和应用领域是什么？其中，哪些是新的研究热点？

126