第3讲 搜索问题求解

尹慧琳，yinhuilin@tongji.edu.cn
同济大学 电子与信息工程学院

第 3 讲 搜索问题求解

第3讲 搜索问题求解

3.1 搜索问题

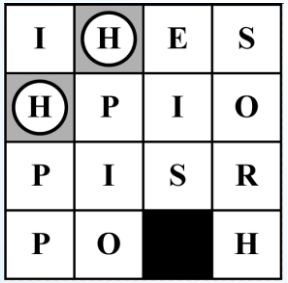
- 引例1：8数码问题

在3×3的棋盘上有8个数字棋子和1个空格。与空格相邻的棋子可以滑动到空格中，目的是通过移动，将给定起始状态变换为目标状态。

7	2	4
5		6
8	3	1

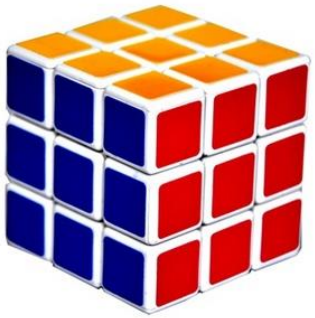
初始状态

漫画人物



英语拼词

华容道



魔方

	1	2
3	4	5
6	7	8

目标状态

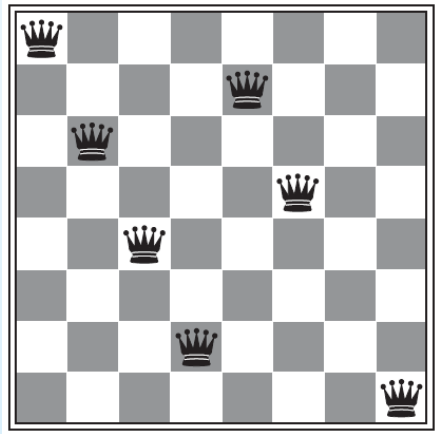
8数码问题属于滑块难题家族。

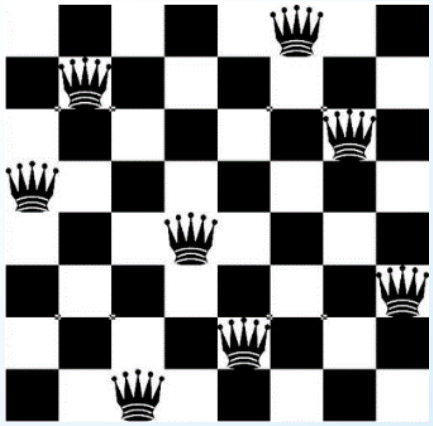
第3讲 搜索问题求解

3.1 搜索问题

- 引例2：八皇后问题

在国际象棋的8×8棋盘上摆放八个皇后，使其不能发生互相攻击（即，任意两个皇后都不能处于同一行、同一列或同一斜线上）。如何摆放？有多少种摆法？





第3讲 搜索问题求解

3.1 搜索问题

- 引例3：传教士和野人问题

三个传教士和三个野人在河的左岸想过河。岸边有一条小船，只能供两个人乘坐：可以是两个传教士、一个传教士和一个野人，或两个野人。但无论在船上或岸边，若野人超过传教士，就会把传教士吃掉。

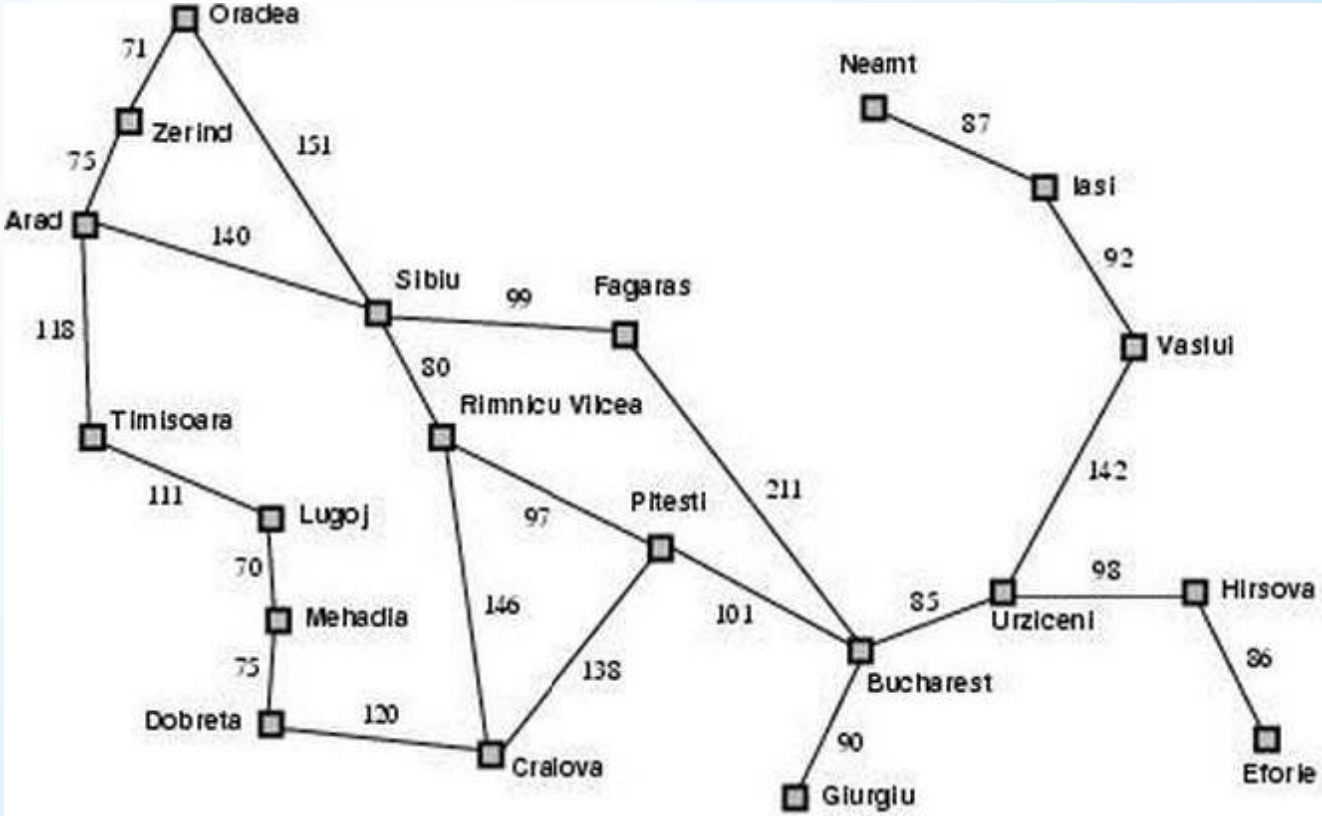
怎样用这条船将其摆渡过河？

第3讲 搜索问题求解

3.1 搜索问题

- 现实世界问题：最短路径（寻径）问题

在多点网络图上，任意二点间的边表示二者存在通路，边上面的数字表示路径的距离。需要寻找起始点到目的地的最短路径。



第 3 讲 搜索问题求解

3.1 搜索问题

3.2 搜索问题的要素

3.3 搜索问题求解

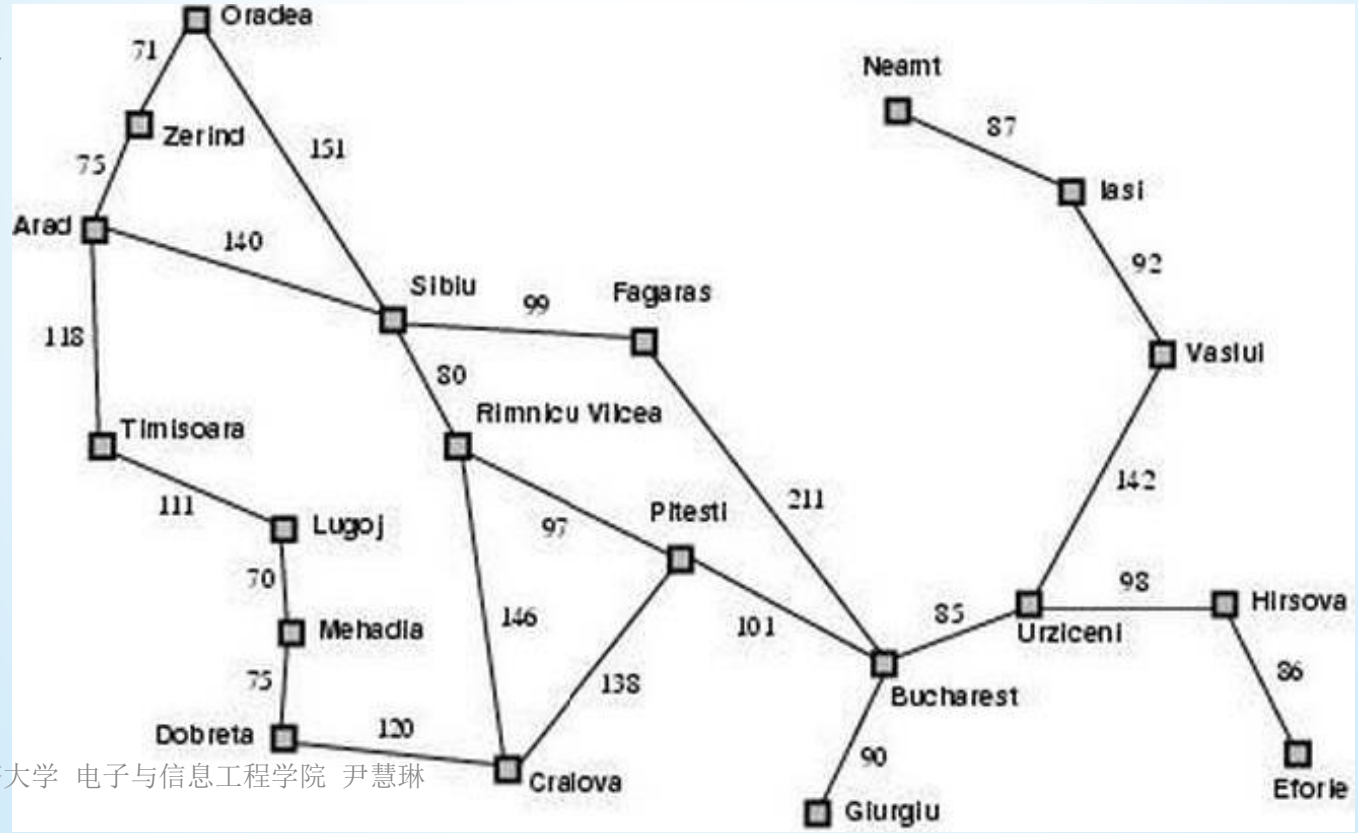
3.4 无信息搜索策略

3.5 有信息搜索策略

第3讲 搜索问题求解

3.2 搜索问题的要素

- 以寻径问题为例：一个正在罗马尼亚Arad城市旅游的Agent，需要寻找前往Bucharest的路径，罗马尼亚的相关城市交通如图示。



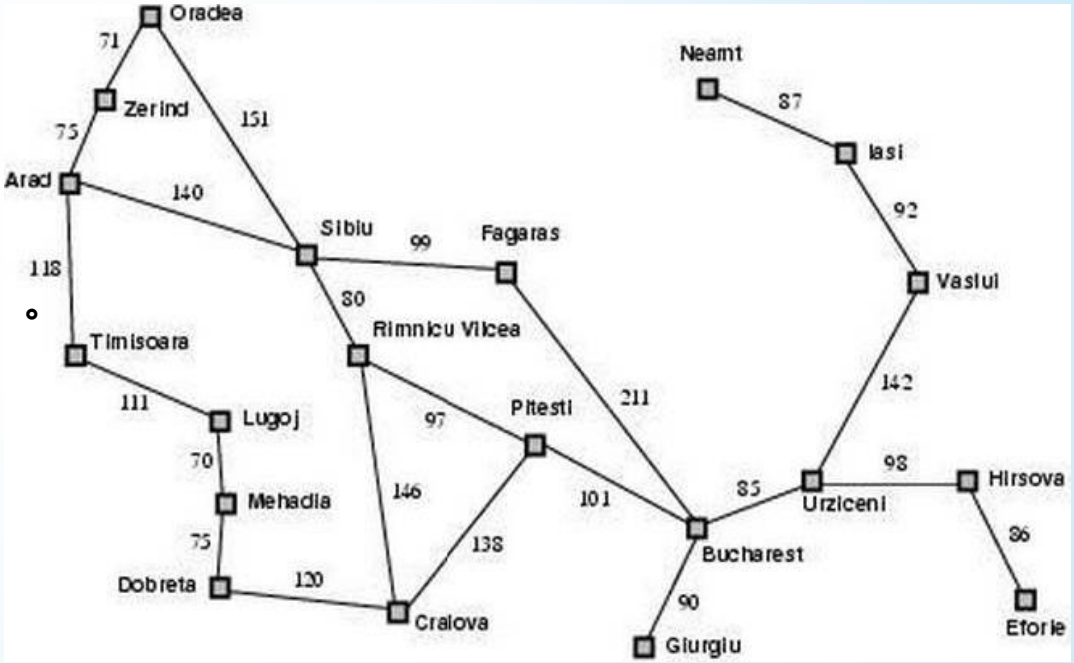
罗马尼亚
部分公路图

第3讲 搜索问题求解

3.2 搜索问题的要素

- 状态空间

  问题在不同时期或不同条件下所表现出的所有状态的有机组成。
  抽象为一个图或一个树（任意两节点连通且没有闭环）。



- 搜索问题构成：
  - 状态空间
  - 初始状态和目标状态
  - 路径代价 / 后继函数

- 搜索问题的解：从初始状态到目标状态的合理的动作序列

第3讲 搜索问题求解

3.2 搜索问题的要素

- 问题形式化的五个要素

1）状态集合S：

In(Bucharest),In(...)

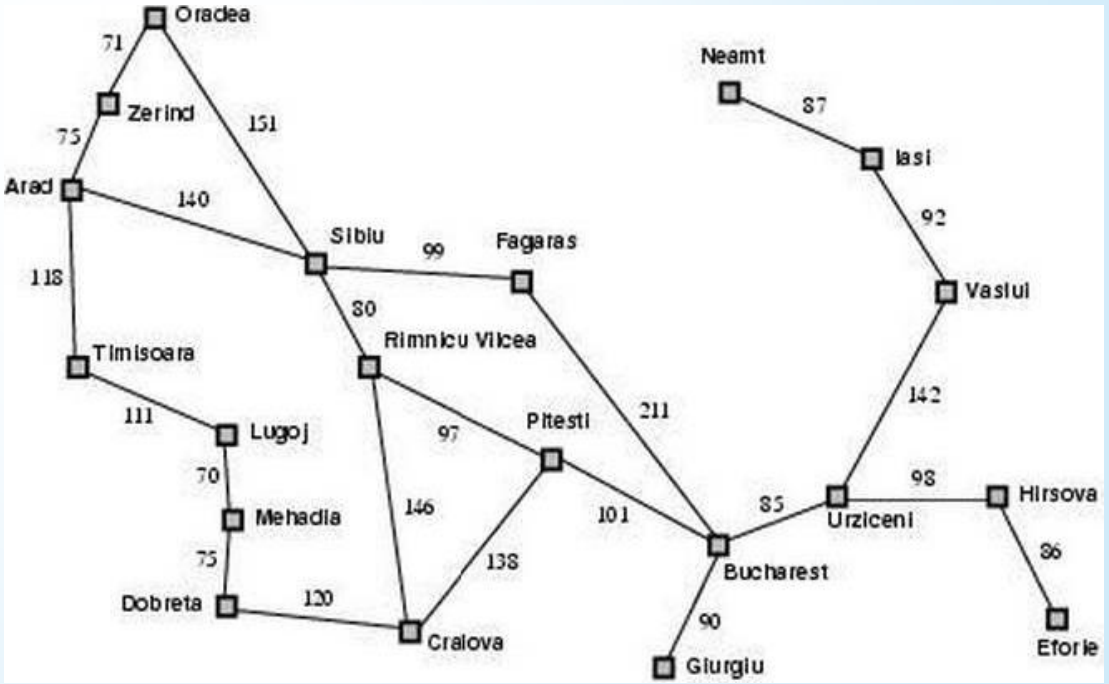
2）初始状态：

In(Arad)

3）动作集合A：

表示特定状态*s*下的可执行的动作集合。

迁移函数或后继函数



第3讲 搜索问题求解

3.2 搜索问题的要素

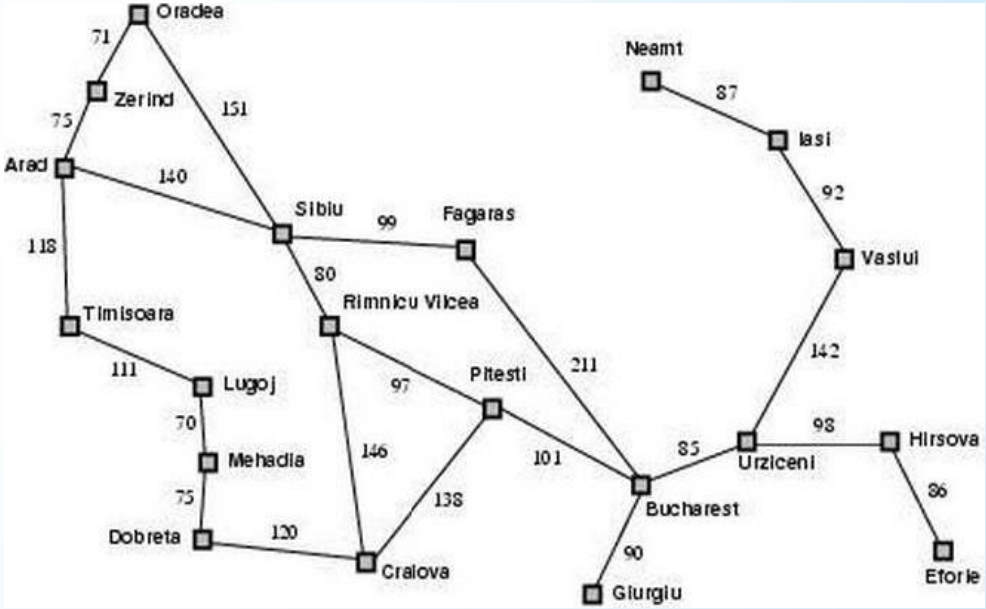
- 问题形式化的五个要素

4）目标测试

确定给定的状态是否是目标状态。

5）路径代价

为每条路径分配一个数值代价，即边加权。



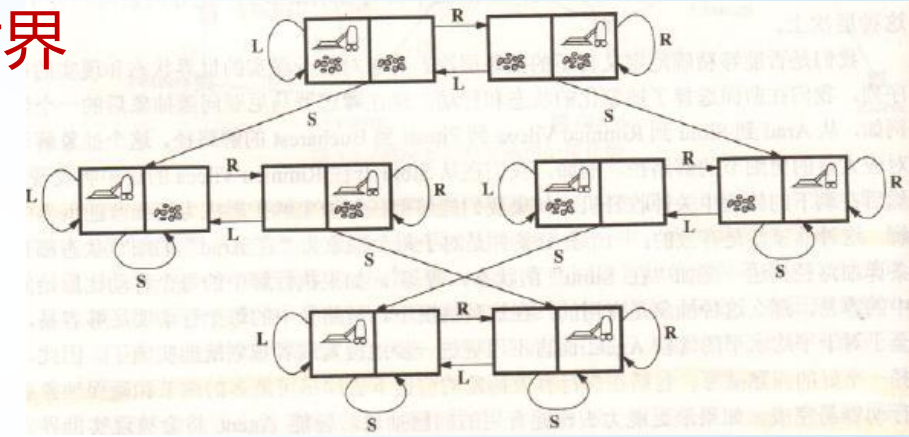
- 单步代价：状态 s 下执行动作 a，到达后继状态 s' 的单步代价用 $c(s,a,s')$ 表示。
- 路径代价：该路径上的每个动作（每条边）的单步代价的总和。

P75：5元组

第3讲 搜索问题求解

3.2 搜索问题的要素

问题实例1：**机器人吸尘器世界**



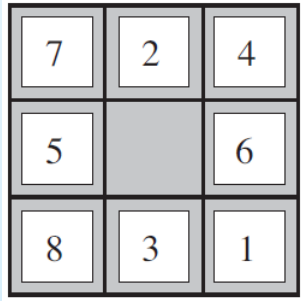
- 状态：由Agent和灰尘位置确定
  - 可能的状态：2*2*2=8个
  - N个位置问题的可能状态？
- 问题的形式化：
  - 状态集合
  - 初始状态
  - 动作集合：左移（L）、右移（R）、吸尘（S）
  - 目标测试：测试是否所有区域都已干净
  - 路径代价：等于路径的步数（每一步的代价为1）

第3讲 搜索问题求解

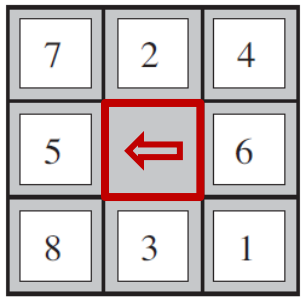
3.2 搜索问题的要素

问题实例2：8数码问题

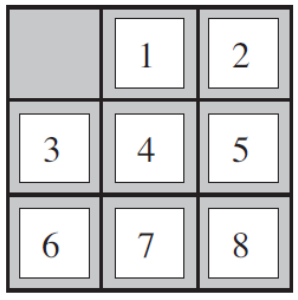
- 问题的形式化：
  - 状态集合
  - 初始状态
  - 动作集合：定义为空格的移动：左移、右移、上移、下移
  - 目标测试：检测状态是否与目标布局相符
  - 路径代价：等于路径的步数（每一步的代价为1）



初始状态



状态转移



目标状态

第3讲 搜索问题求解

3.2 搜索问题的要素

问题实例3：八皇后问题

- 问题的（增量，全态）形式化：
  - 状态集合：棋盘上逐次摆放一至八个皇后
  - 初始状态：没皇后
  - 动作集合：每次添加一个皇后，其不发生攻击
  - 目标测试：八个，且没攻击
  - 路径代价：添加皇后的步数。

第3讲 搜索问题求解

3.2 搜索问题的要素

- 现实世界的问题：其他问题
- 旅行商问题（TSP）：一个城市交通图上，要求每个城市都能且仅能被访问一次，目标是找最短路。——NP难问题
- VLSI布线问题：要求在一个芯片上放置几百万个元器件和连线，较小的芯片面积 、较少的电路延迟、较小的杂散电容和较大的产量。分为单元布局和通道布线二类问题。
- 机器人导航问题：将离散状态的寻径问题一般化为连续问题，机器人导航可以在连续空间上运用，可能的动作和状态是无限集合。
- 自动装配序列问题：目标是找到装配对象各个部件的次序。关键在于生成合法动作。任何实用算法都应避免搜索全部状态空间。

第 3 讲 搜索问题求解

3.1 搜索问题

3.2 搜索问题的要素

3.3 搜索问题求解

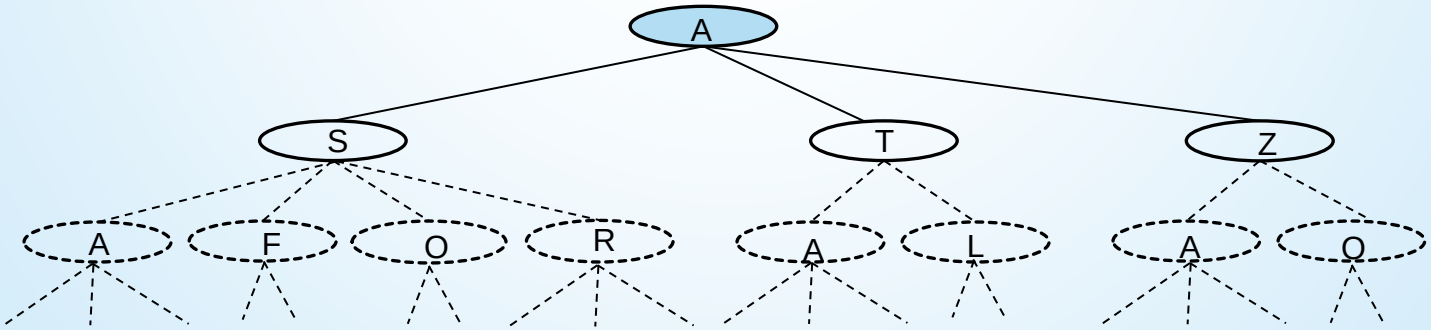
3.4 无信息搜索策略

3.5 有信息搜索策略

第3讲 搜索问题求解

3.3 搜索问题求解

- 搜索求解的概念
  - 问题：给定一个目标，决定要考虑的动作与状态
  - 解：一个达到目标的动作序列
  - 搜索：为达到目标，寻找这样的动作序列的过程
  - 搜索操作(重复)：目标检测：判断当前节点是否是目标状态
    扩展：对当前节点应用各种合法动作，生成后继状态加入边缘集
  - 搜索策略：在选择哪一个节点执行扩展操作时需要依据的原则

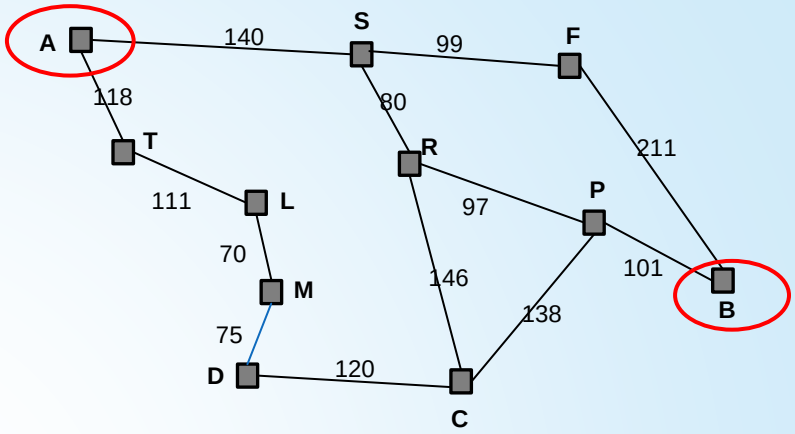


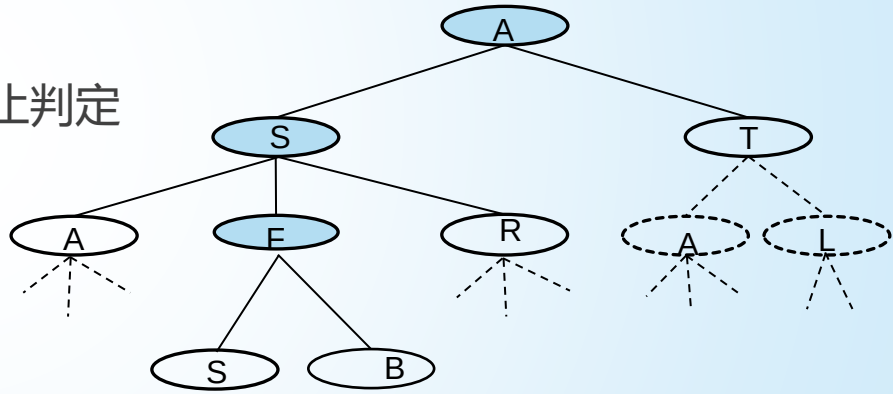
第3讲 搜索问题求解

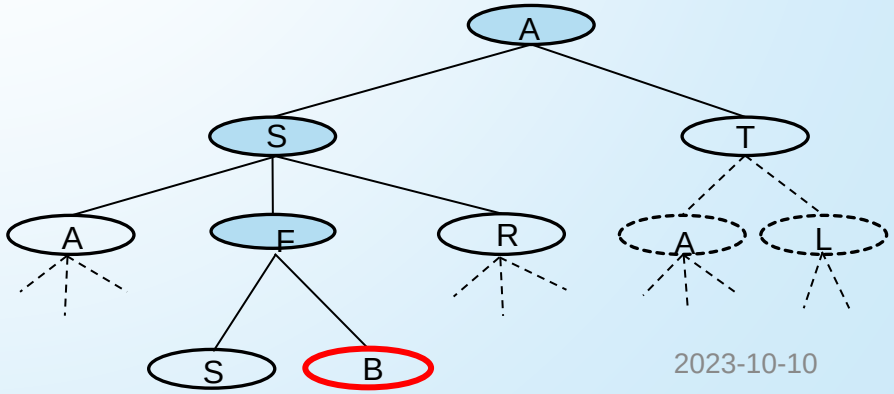
3.3 搜索问题求解

- 搜索求解的过程
  - 确定搜索的起点
  - 决定如何往下走 – 搜索策略
  - 判断是否要继续 – 目标测试、终止判定
- 搜索过程举例
  - 初始状态 A，扩展生成 S 和 T
  - 扩展 S，生成 A, F, R
  - 扩展 F，生成 S, B
  - 检测 B

成功！

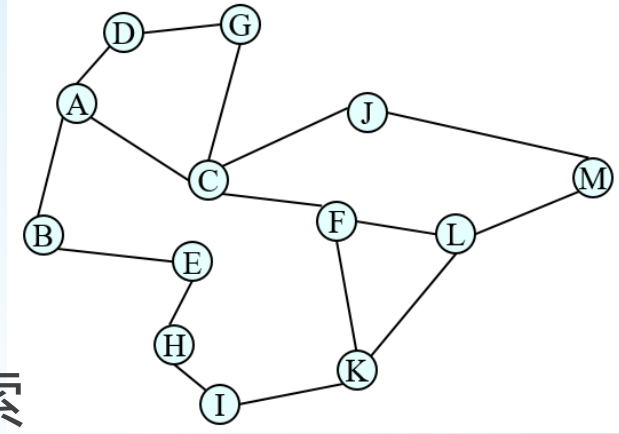






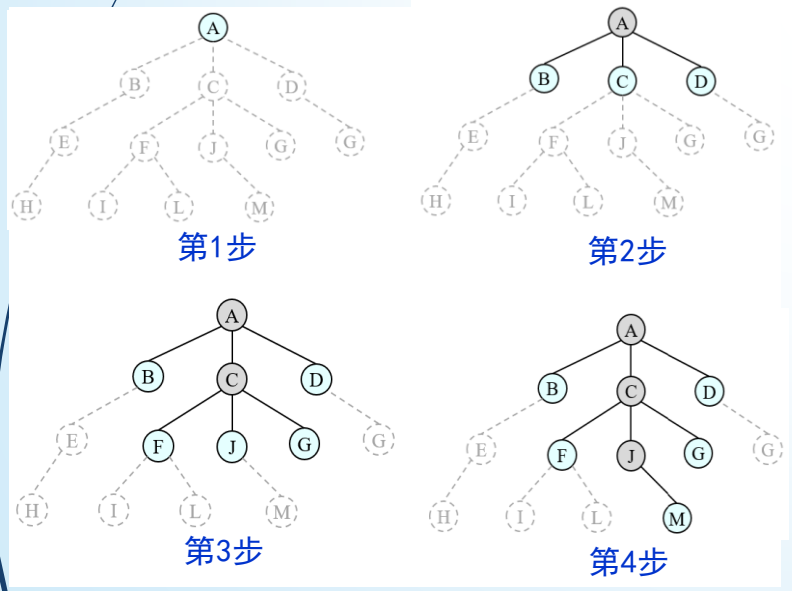
第3讲 搜索问题求解

3.3 搜索问题求解



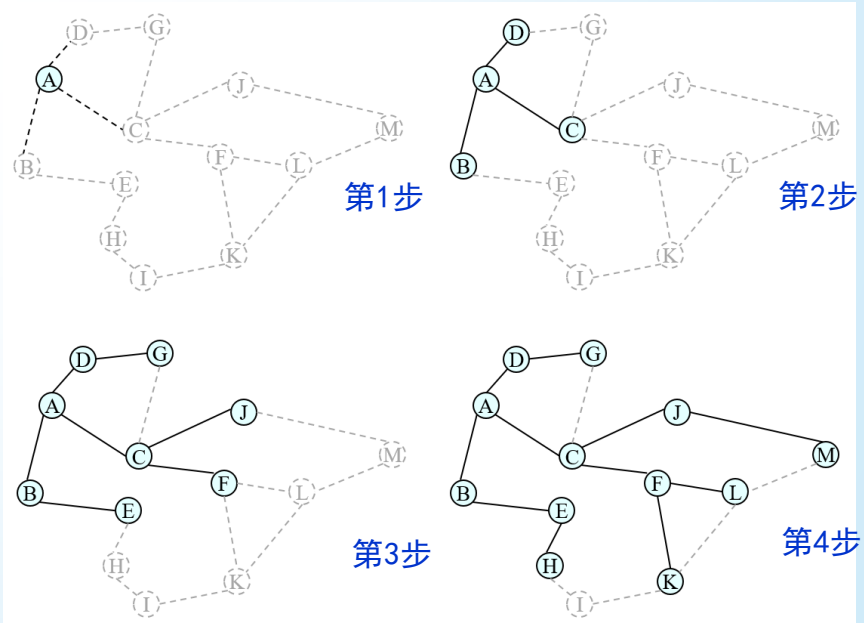
- 树搜索

  用搜索树来寻找一个解



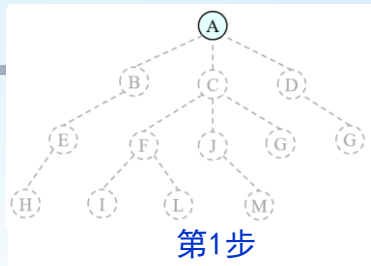
- 图搜索

  用搜索图来寻找解

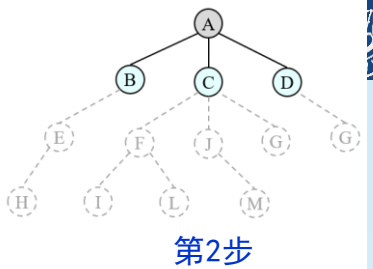


第3讲 搜索问题求解

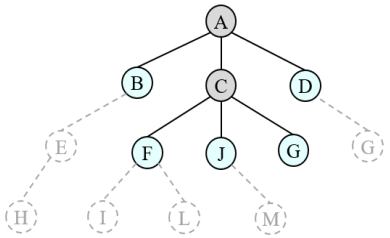
3.3 搜索问题求解



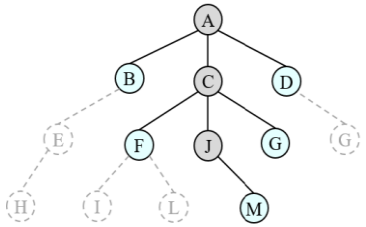
第1步



第2步



第3步



第4步

算法 3.2 一个通用的树搜索算法

agent TREE-SEARCH

input *problem*

output a solution, or failure

local *frontier*, to store the set of all leaf nodes

initialize the *frontier* using the initial state of *problem*

loop do

if the *frontier* is empty **then return** failure

choose a leaf node and remove it from the *frontier*

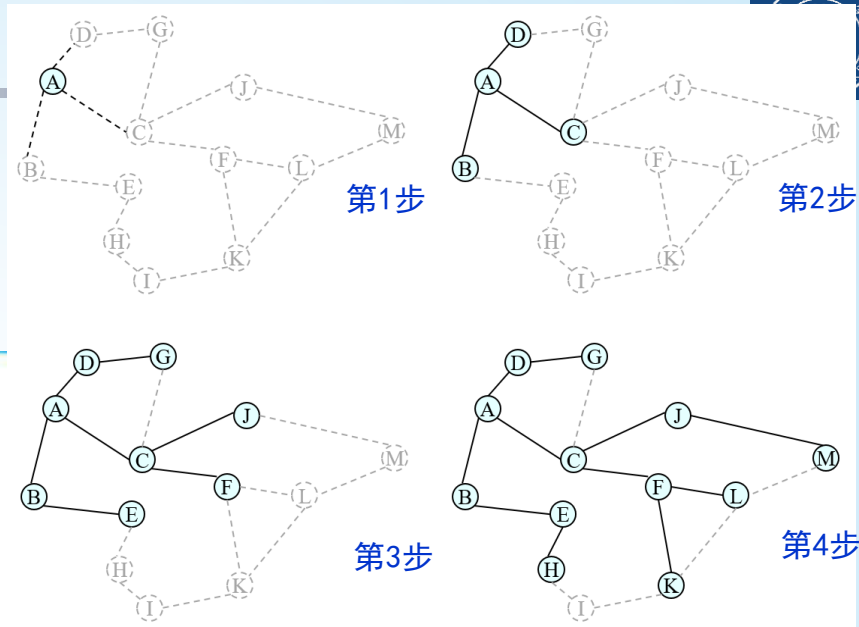
if the node contairs a goal state **then return** the corresponding solution

expand the chosen node, adding the resulting nodes to the *frontier*

loop end

第3讲 搜索问题求解

3.3 搜索问题求解



算法 3.3 一个通用的图搜索算法

agent GRAPH-SEARCH

input *problem*

output a solution, or failure

local *frontier*, to store the set of all leaf nodes

explored, to remember every expanded nodes

initialize the *frontier* using the initial state of *problem*

initialize the *explored* to be empty

loop do

if the *frontier* is empty **then return** failure

choose a leaf node and remove it from the *frontier*

if the node contains a goal state **then return** the corresponding solution

add the node to the *explored*

expand the chosen node, adding the resulting nodes to the *frontier*

only if not in the *frontier* or the *explored*

loop end

第3讲 搜索问题求解

3.3 搜索问题求解：搜索策略

- 搜索方向：
  - 正向搜索：从初始状态出发
  - 逆向搜索：从目的状态出发
  - 双向搜索：从初始状态出发作正向搜索，同时又从目的状态出发作逆向搜索，直到两条路径在中间的某处汇合为止
- 搜索方法分类
  - 无信息搜索（盲目搜索）

    除了问题定义中提供的状态信息外，没有任何附加信息。
  - 有信息搜索（启发式搜索）

    除了基本问题定义，还会进一步利用一些与问题有关的启发式信息来引导更有效的搜索。

第 3 讲 搜索问题求解

3.1 搜索问题

3.2 搜索问题的要素

3.3 搜索问题求解

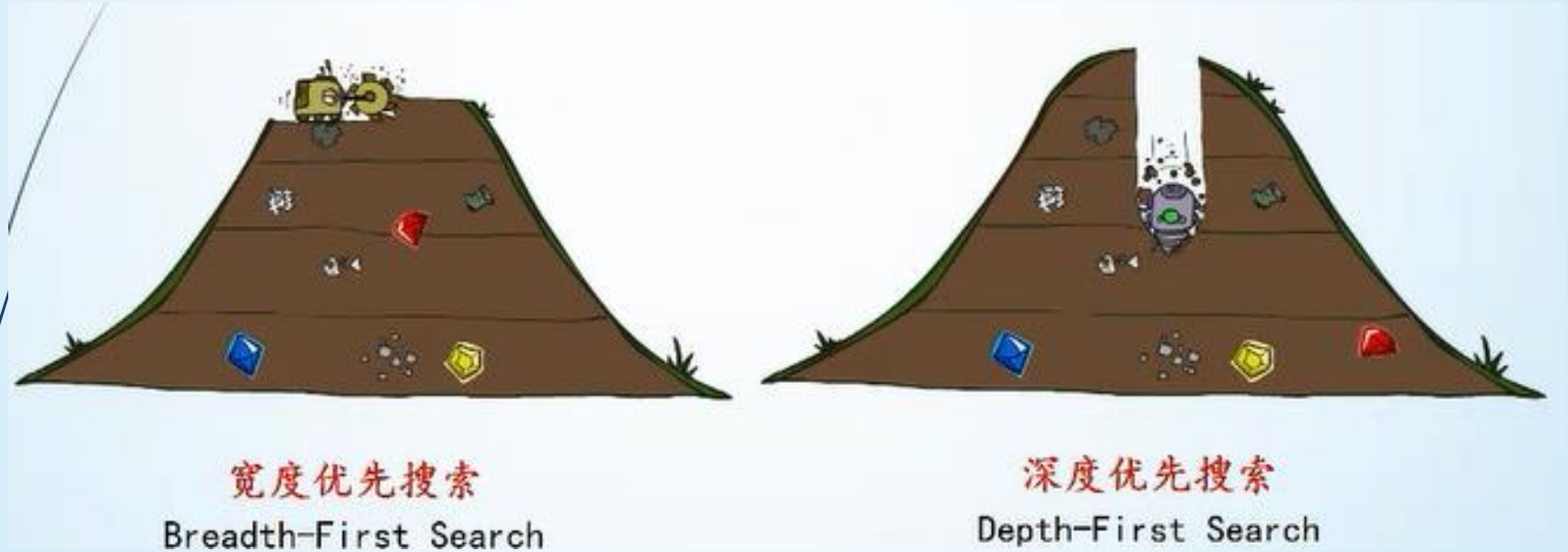
3.4 无信息搜索策略

3.5 有信息搜索策略

第3讲 搜索问题求解

3.4 无信息搜索策略

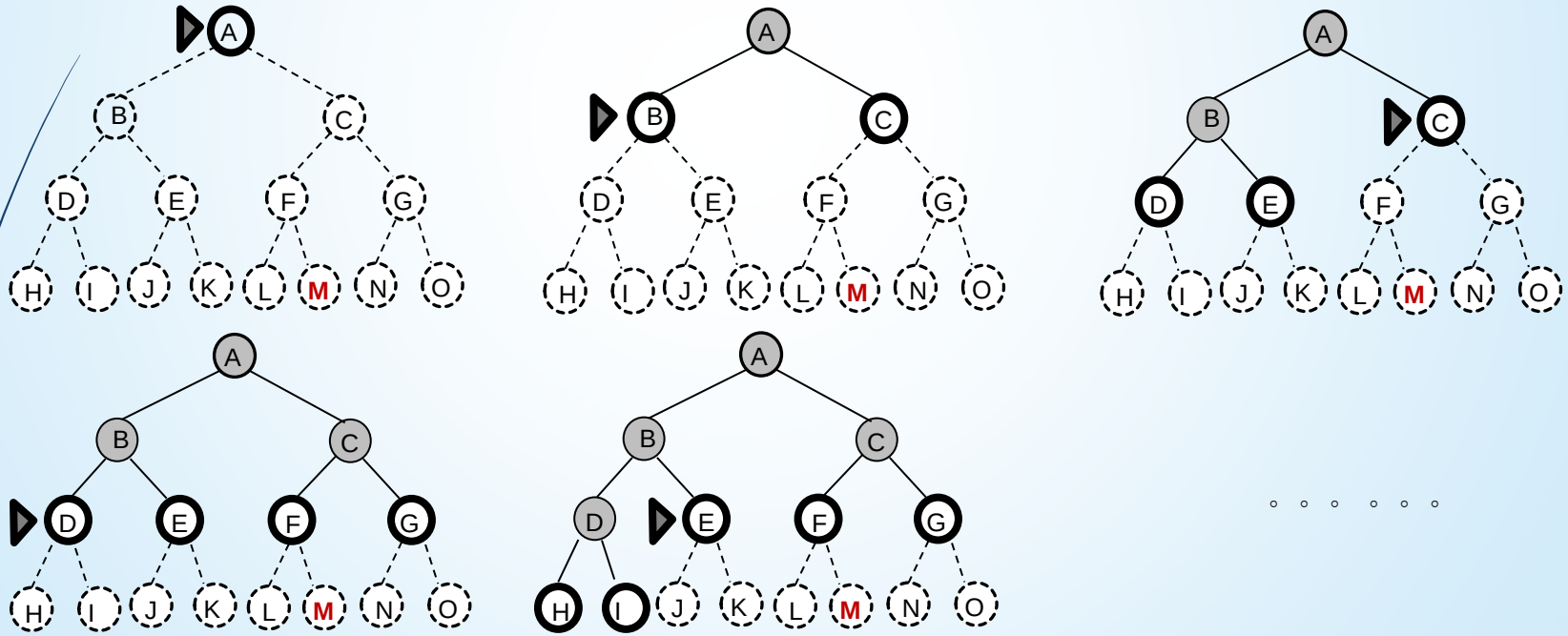
- 策略：根据节点扩展的顺序进一步区分为两大类



第3讲 搜索问题求解

3.4 无信息搜索策略：宽度优先搜索BFS

- 原则：每次总是扩展深度最浅的未扩展节点
- 实现方法：将边缘集合组织成FIFO队列，依次处理



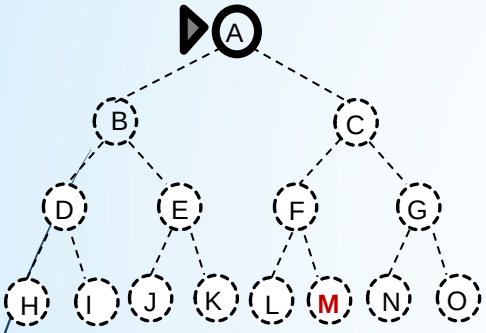
。。。。。。

3层二叉树的宽度优先搜索过程，M 是唯一的目标节点

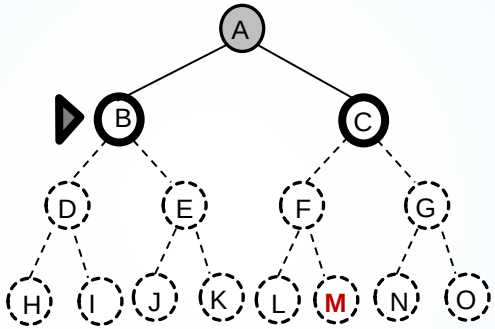
第3讲 搜索问题求解

3.4 无信息搜索策略：宽度优先搜索BFS

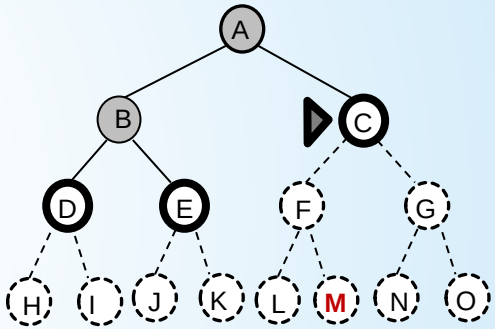
closed=( ), open=(A)



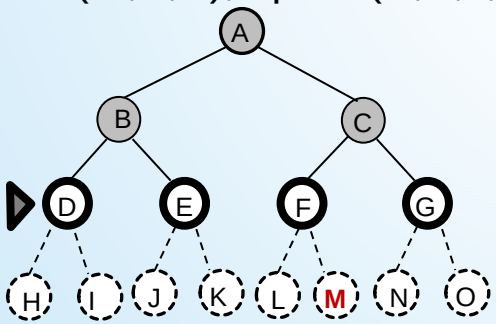
closed=(A), open=(B,C)



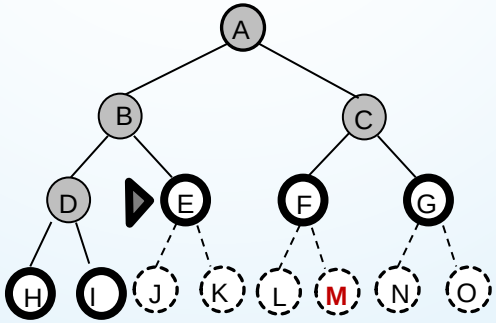
closed=(A,B), open=(C,D,E)



closed=(A,B,C), open=(D,E,F,G)



closed=(A,B,C,D), open=(E,F,G,H,I)



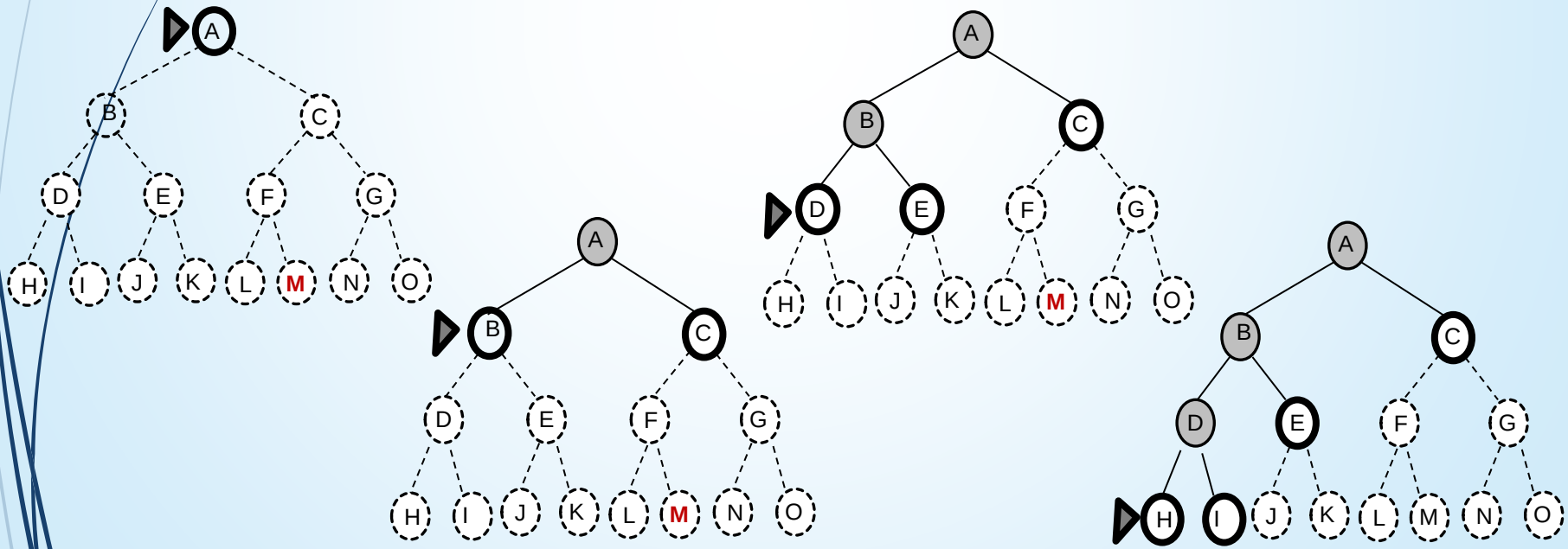
。 。 。 。 。 。

3层二叉树的宽度优先搜索过程，M 是唯一的目标节点

第3讲 搜索问题求解

3.4 无信息搜索策略：深度优先搜索DFS

- 原则：扩展最深的未扩展节点
- 搜索过程：

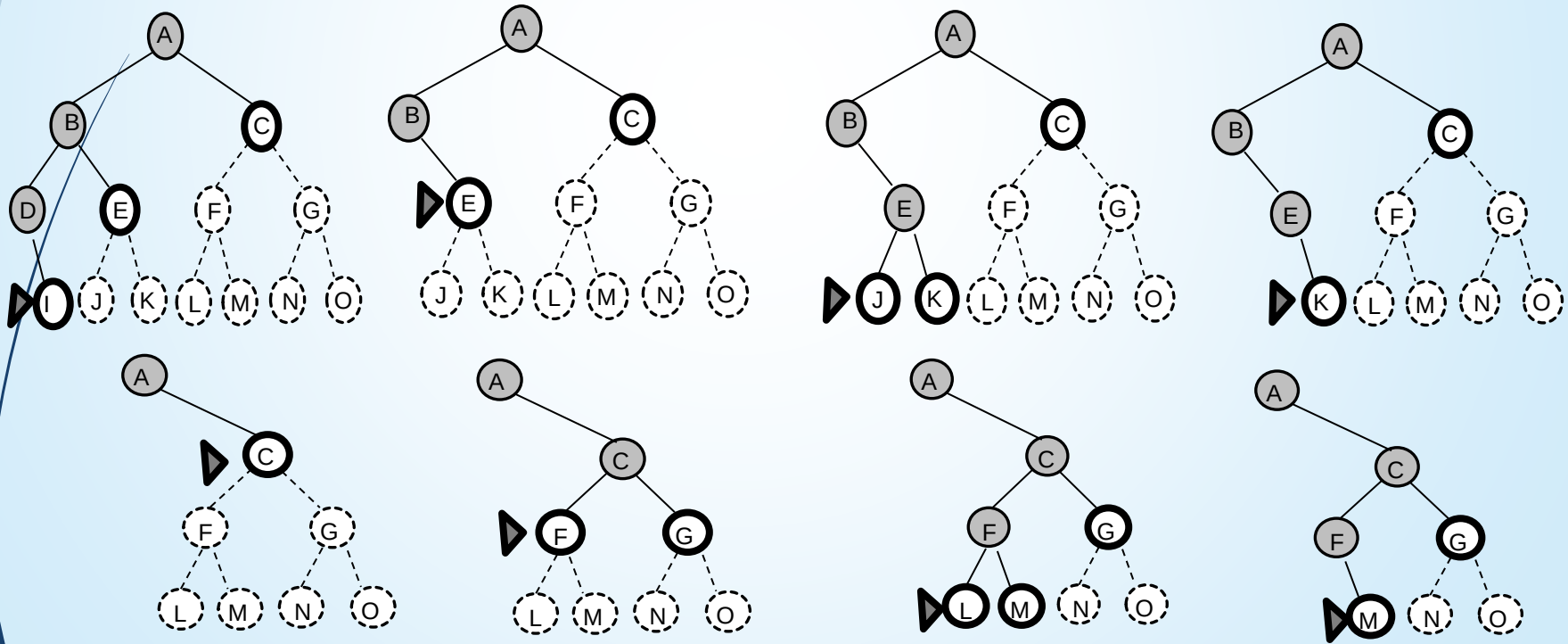


第3讲 搜索问题求解

3.4 无信息搜索策略：深度优先搜索DFS

- 实现方法：将边缘集合组织成LIFO队列，依次处理

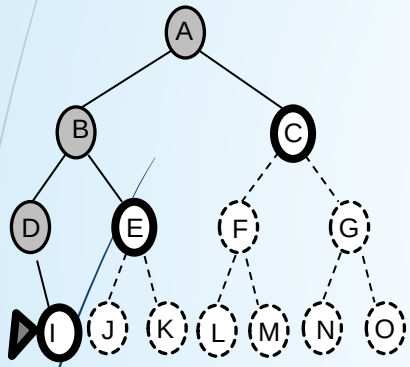
(最新生成的节点 最早被选择扩展)



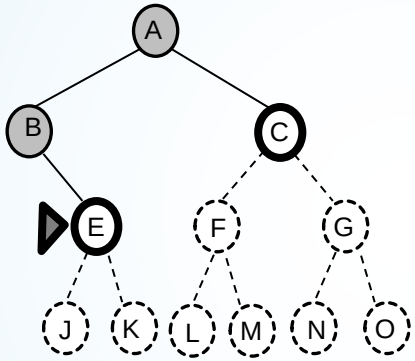
第3讲 搜索问题求解

3.4 无信息搜索策略：深度优先搜索DFS

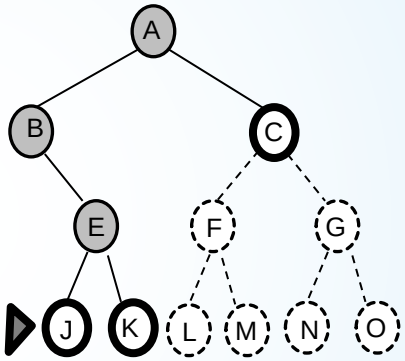
closed=(A,B,D)
open=(C,E,I)



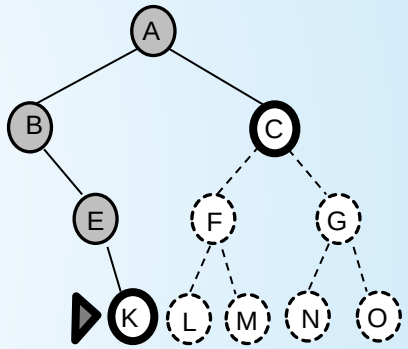
closed=(A,B,D,I),
open=(C,E)



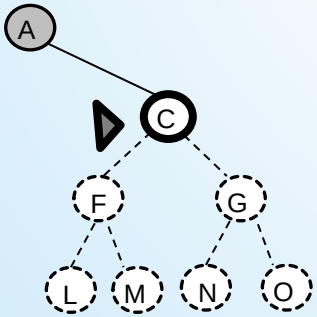
closed=(A,B,D,I,E),
open=(C,K,J)



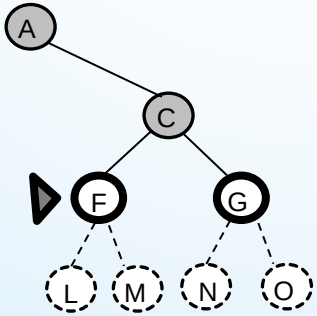
closed=(A,B,D,I,E,J),
open=(C,K)



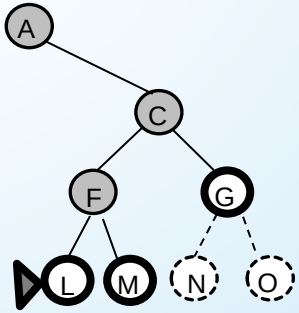
closed=(A,B,D,I,E,J,K)
open=(C)



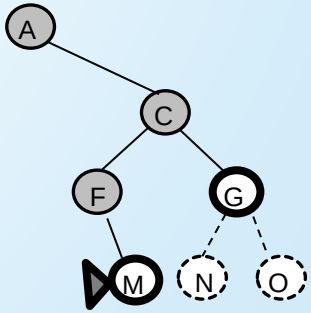
closed=(A,B,D,I,E,J,K,C)
open=(G,F)



closed=(A,B,D,I,E,J,K,C,F),
open=(G,M,L)



closed=(A,B,D,I,E,J,K,C,F,L),
open=(G,M)



第3讲 搜索问题求解

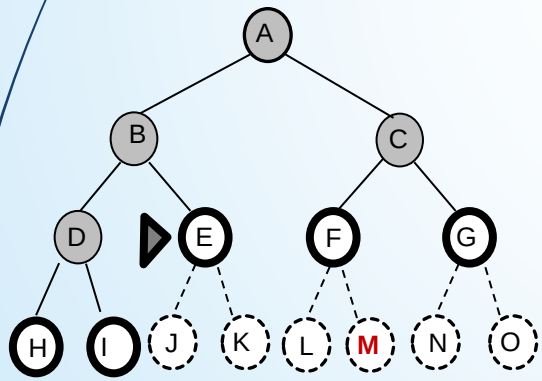
3.4 无信息搜索策略

- 算法性能比较分析：
  - 完备性：均是完备的，极端情况就是把整个状态空间遍历一遍
  - 最优性：都只是把首先达到的目标节点找到，便停止搜索，不一定最优
  - 时间复杂度：均为指数级别，搜索耗时较长
  - 空间复杂度：宽度优先搜索是指数级别；
    深度优先搜索是线性级别
- 挑战性
  - 搜索时长：指数复杂性 P83 汉诺塔问题
  - 内存占用

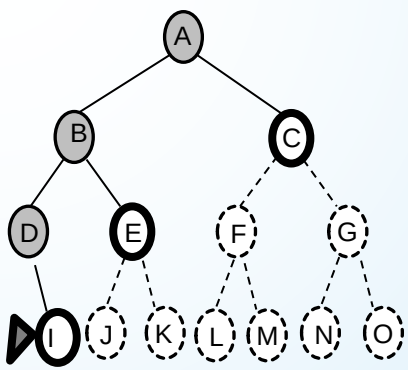
第3讲 搜索问题求解

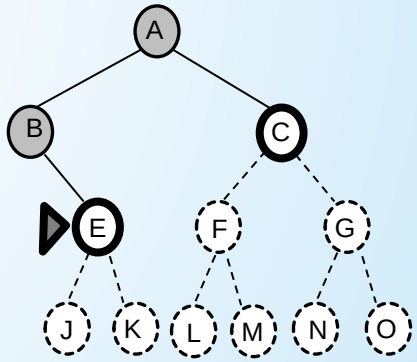
3.4 无信息搜索策略

- 复杂度：
  - 时间复杂度：均为指数级别，搜索耗时较长 $b + b^2 + b^3 + \cdots = O(b^d)$
  - 空间复杂度：宽度优先搜索是指数级别；深度优先搜索是线性级别$O(bd)$



宽度优先





深度优先

第3讲 搜索问题求解

3.4 无信息搜索策略

宽度优先	深度优先
每次扩展深度最（）的节点	每次扩展深度最（）的节点
边缘集合组织成（）队列	边缘集合组织成（）队列
保留全部节点，占用空间大。空间复杂度是（）级别。	不全部保留节点，占用空间少。空间复杂度是（）级别。
无入栈、出栈操作，运行速度快。	有入栈、出栈操作，运行速度慢。

第3讲 搜索问题求解

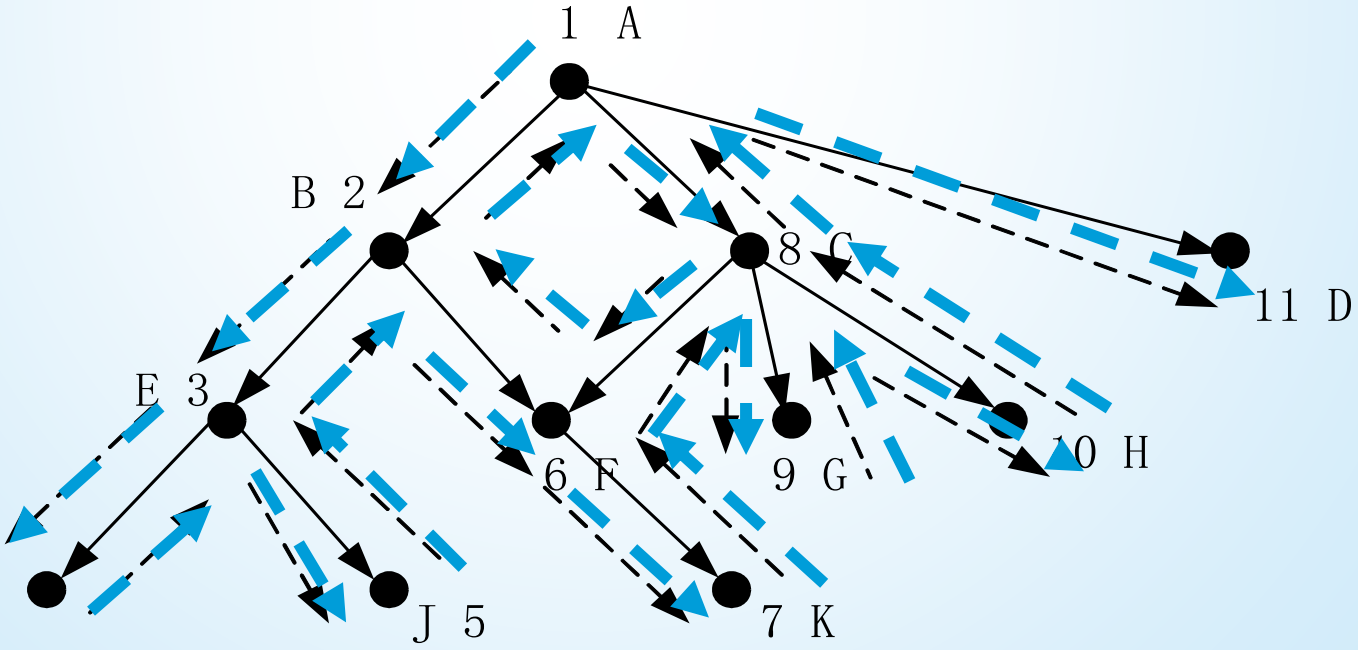
3.4 无信息搜索策略

宽度优先	深度优先
每次扩展深度最浅的节点	每次扩展深度最深的节点
边缘集合组织成FIFO队列，用队列（queue）实现	边缘集合组织成LIFO队列，用堆栈（stack）实现
保留全部节点，占用空间大。空间复杂度是指数级别。	不全部保留节点，占用空间少。空间复杂度是线性级别。
无入栈、出栈操作，运行速度快。	有入栈、出栈操作，运行速度慢。

第3讲 搜索问题求解

3.4 无信息搜索策略

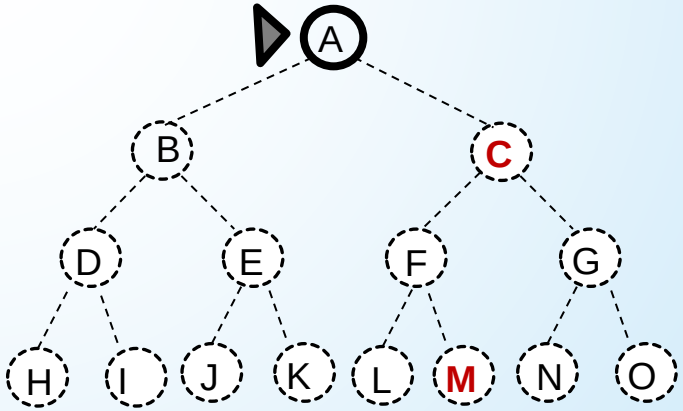
- 回溯(backtracking)搜索：
  - 改进：每次扩展只产生一个后继节点
  - 优点：进一步降低空间复杂度为常数级



第3讲 搜索问题求解

3.4 无信息搜索策略

- 深度受限搜索
  - 针对：深度优先搜索的复杂度取决于m，若m无限则搜索失败
  - 改进：设置深度界限 l，深度 l 以外的节点被视为无后继节点
  - 优点：降低了复杂度，解决了无穷路径的问题
  - 缺点：
    - l的选择



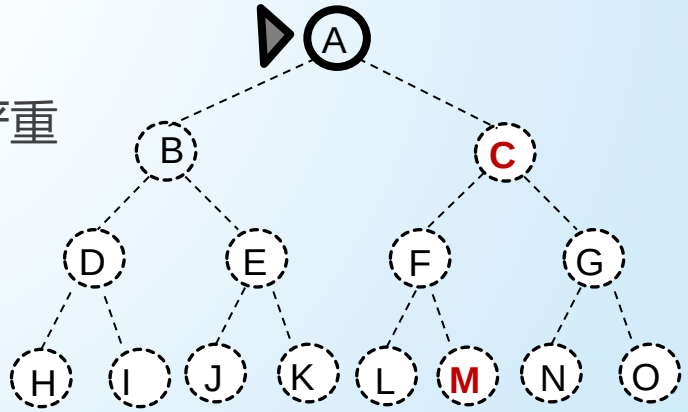
第3讲 搜索问题求解

3.4 无信息搜索策略

- 迭代加深搜索
  - 针对：深度优先搜索的局限性，且深度受限搜索中深度界限难以设定的困难
  - 改进：以深度优先搜索相同的顺序访问搜索树节点，逐步增加深度限制，反复运行直到找到目标
  - 优点：将深度优先和宽度优先相结合
  - 缺点：存在状态的重复生成，越上层越严重

一般认为：

当搜索空间较大且不知道解所在深度时，迭代加深搜索是首选的无信息搜索方法。

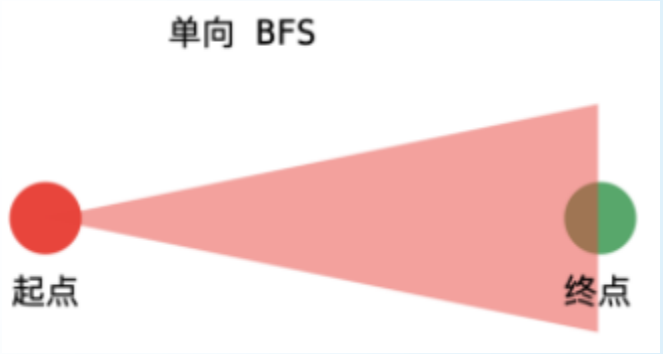


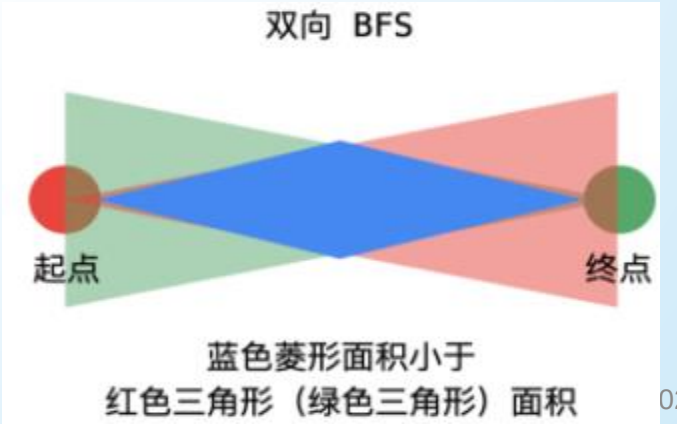
迭代加深搜索过程

第3讲 搜索问题求解

3.4 无信息搜索策略：双向搜索

- 原理：同时进行两个搜索，一个是从初始状态向前搜索，而另一个则从目标状态向后搜索。当两者在中间相遇时停止。
- 优点：降低搜索的复杂度
- 性能：
  - 完备性：可满足
  - 最优性：不一定满足
  - 复杂度：$O(b^{d/2})$





第 3 讲 搜索问题求解

3.1 搜索问题

3.2 搜索问题的要素

3.3 搜索问题求解

3.4 无信息搜索策略

3.5 有信息搜索策略

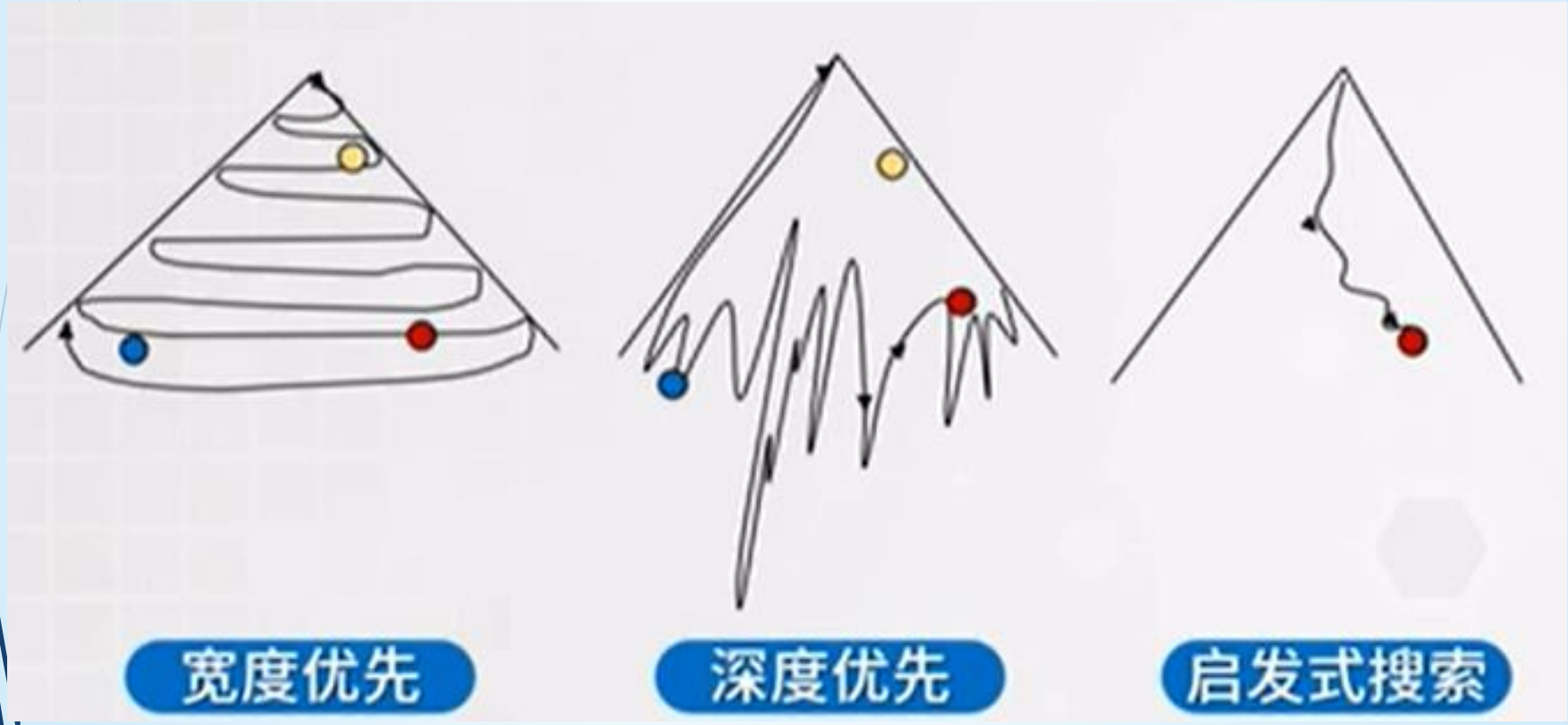
第3讲 搜索问题求解

3.5 有信息搜索策略

- 依据：
  - 问题本身定义
  - 启发式信息
- 特点：基于评价函数$f(n)$，选择“最有希望”的节点进行扩展
- 基本的有信息搜索策略：
  - 一致代价搜索（Uniform-cost Search）
  - 贪婪搜索 （Greedy Search）
  - A*搜索 （A* Search）

第3讲 搜索问题求解

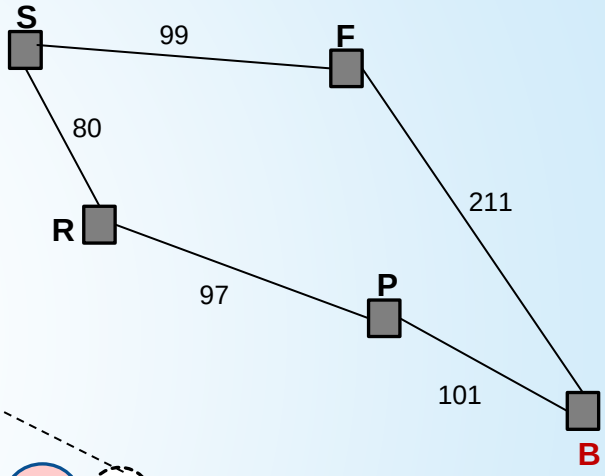
3.5 有信息搜索策略

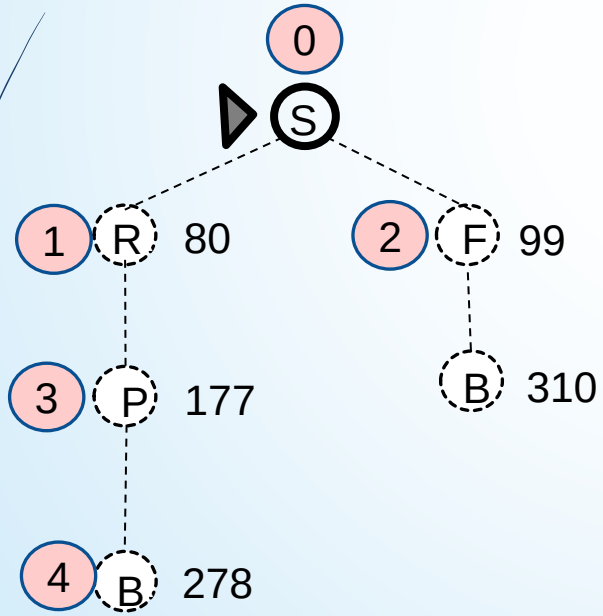


第3讲 搜索问题求解

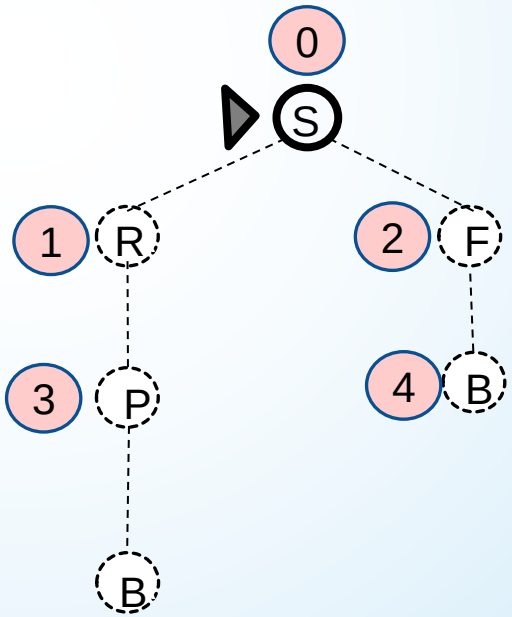
3.5 有信息搜索策略：一致代价搜索

- 搜索策略：扩展最低代价的未扩展节点
- 评价函数：定义为路径代价 f(n)=g(n)





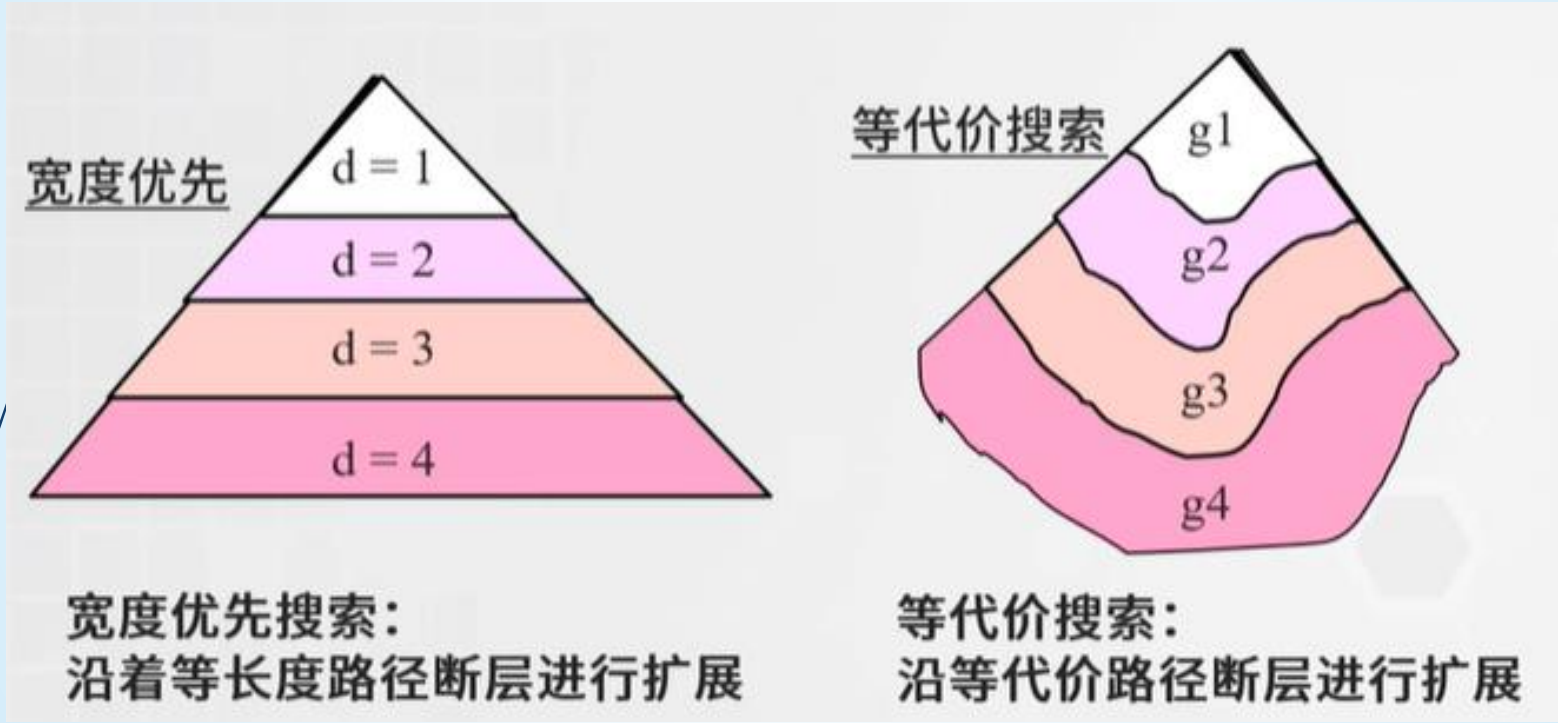
用一致代价搜索求解



用宽度优先搜索求解

第3讲 搜索问题求解

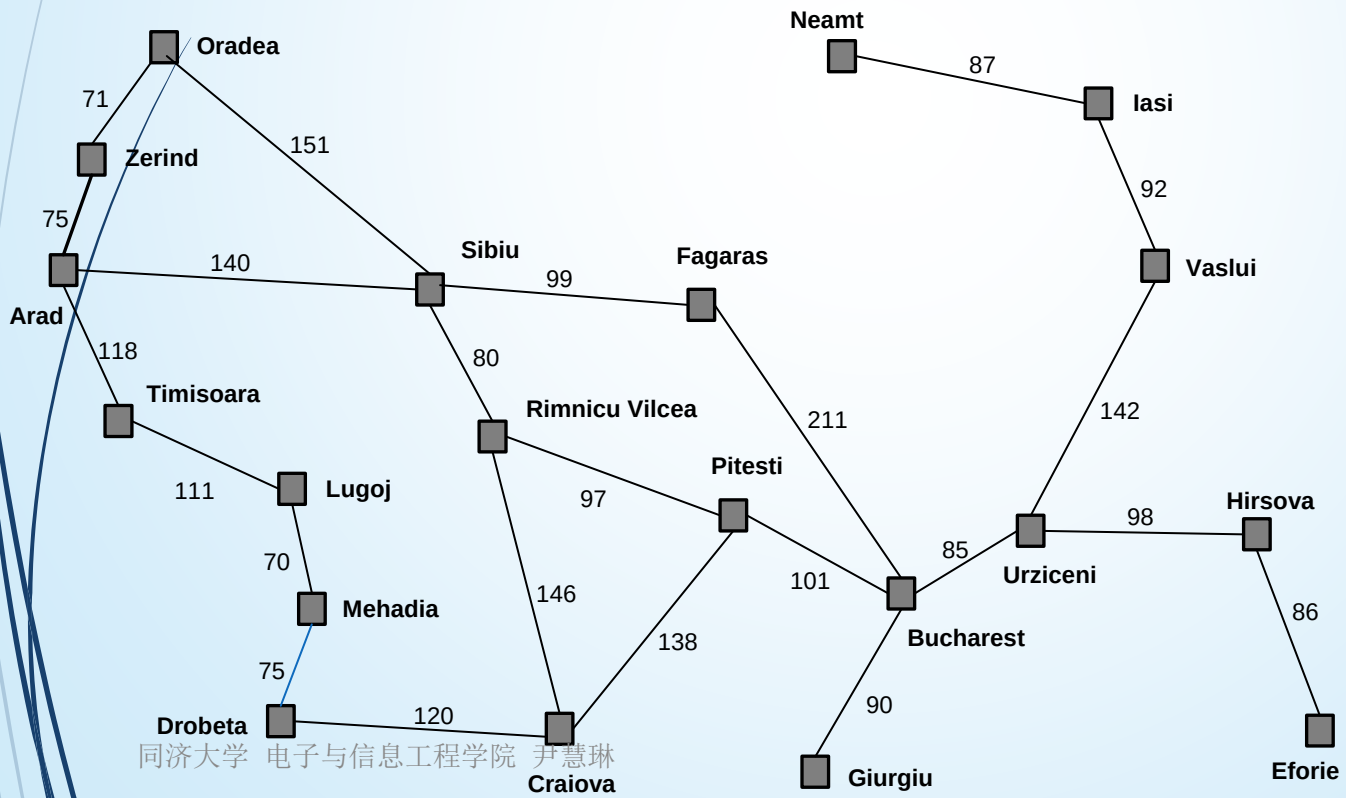
3.5 有信息搜索策略



第3讲 搜索问题求解

3.5 有信息搜索策略：贪婪搜索

- 搜索策略：试图扩展最接近目标的节点
- 评价函数：只使用启发式信息 $f(n) = h(n)$

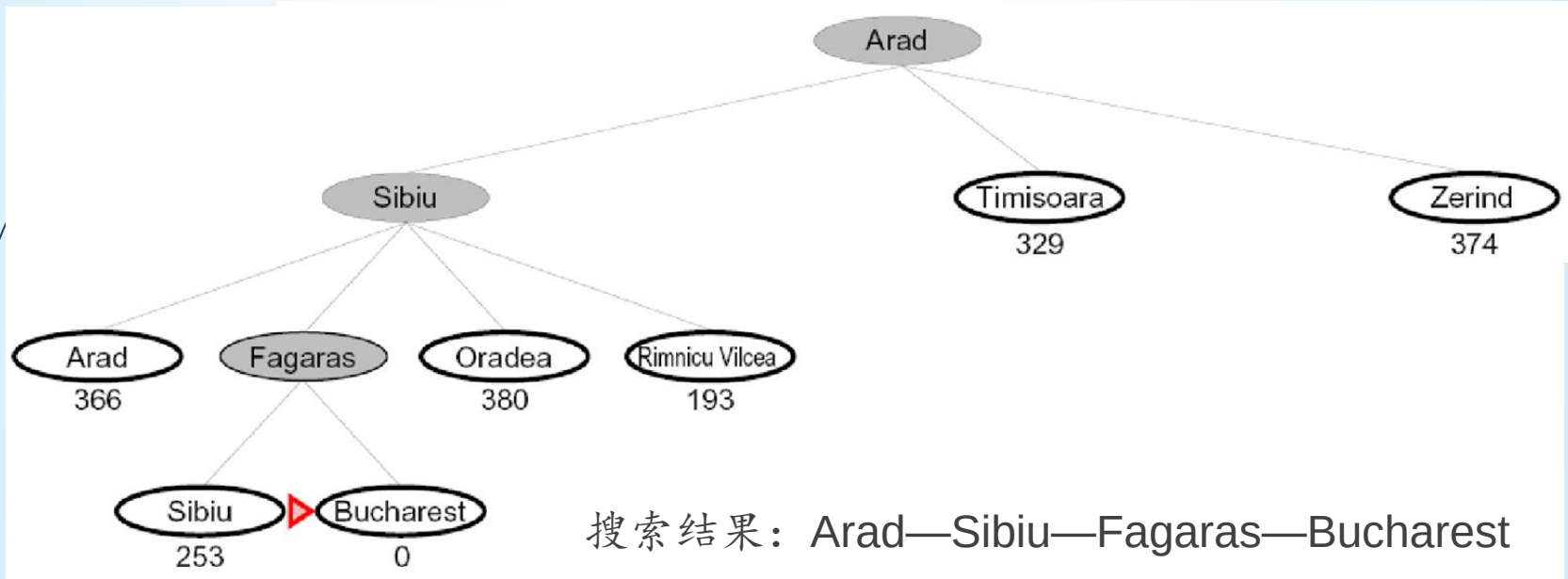


节点（城市）	H值（直线距离）
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

第3讲 搜索问题求解

3.5 有信息搜索策略：贪婪搜索

- 搜索过程：仅使用启发式信息作为评价函数 $f(n) = h(n)$

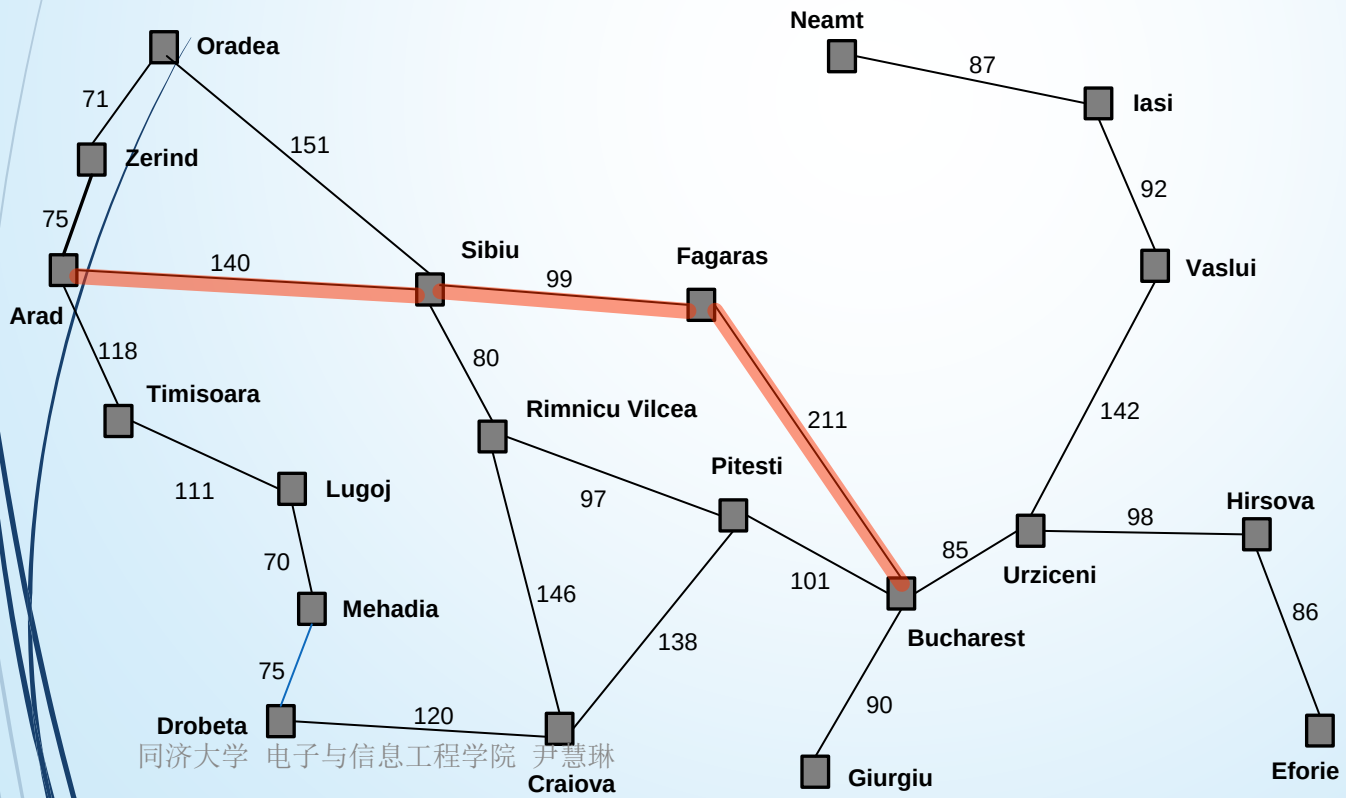


- 是否最优?

第3讲 搜索问题求解

3.5 有信息搜索策略：贪婪搜索

- 搜索策略： 试图扩展最接近目标的节点
- 评价函数： 只使用启发式信息 $f(n) = h(n)$

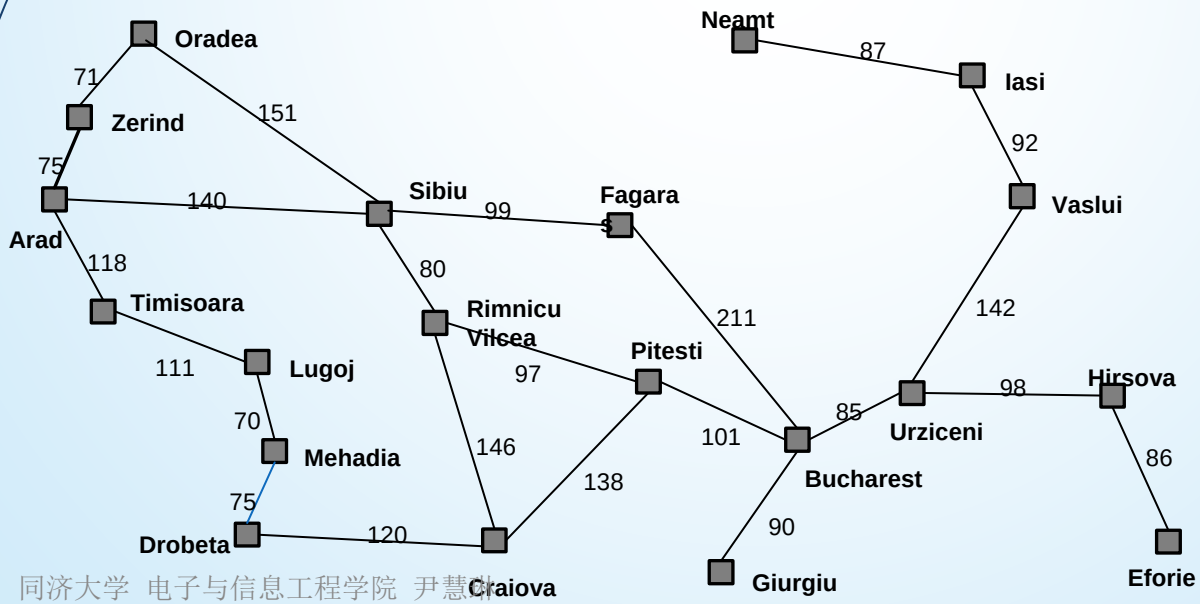


节点（城市）	H值（直线距离）
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

第3讲 搜索问题求解

3.5 有信息搜索策略：A*搜索

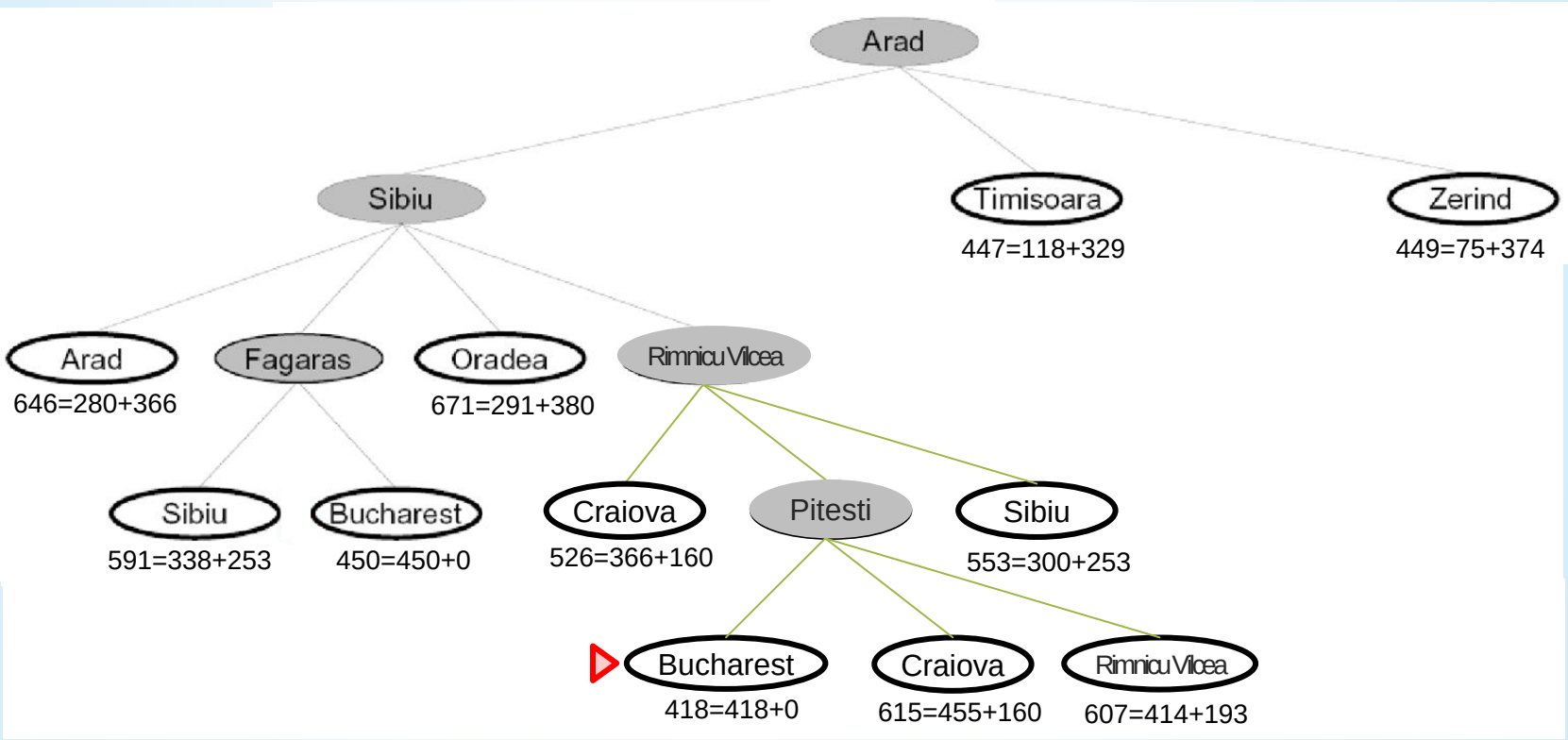
- 搜索策略：避免扩展代价高的路径，最小化总的评估代价
- 评价函数：$f(n) = g(n) + h(n)$
  - $g(n)$：到达该节点已经花费的代价
  - $h(n)$：从该节点到目标的估计代价



节点	H值
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

第3讲 搜索问题求解

3.5 有信息搜索策略：A*搜索



贪婪搜索的结果： Arad—Sibiu—Frgaras—Bucharest，路径代价： 450

A* 搜索的结果： Arad—Sibiu—Rimnicu Vicea—Pitestie—Bucharest，路径代价： 418

第3讲 搜索问题求解

3.5 有信息搜索策略：A*搜索

- A*算法分析
  - 优势：在从根节点开始扩展搜索解路径的问题求解方法中，A*算法是完备的、最优的、也是高效率的
  - 不足：指数级的时间复杂度，高空间复杂度引起的高内存占用

第3讲 搜索问题求解

【补充】启发式函数

- 启发式搜索策略的两个组成部分：
  - 启发式函数
  - 使用该启发式函数搜索状态空间的算法
- 启发式函数的作用
  - 由于在问题陈述和数据获取方面存在模糊性，可能会使一个问题没有一个确定的解，则需要运用启发式函数做出最有可能的估计
  - 即使一个问题有确定解，但因其状态空间过大，往往难以在给定的时空内得到最终解，可借助启发式函数引导搜索向最有希望的方向进行

启发式函数及算法设计一直是人工智能的核心问题。

第3讲 搜索问题求解

【补充】启发式函数

- 举例：8数码问题的启发式求解
  - 启发式算法：A*算法
  - 启发式函数： h_1=错位棋子的个数

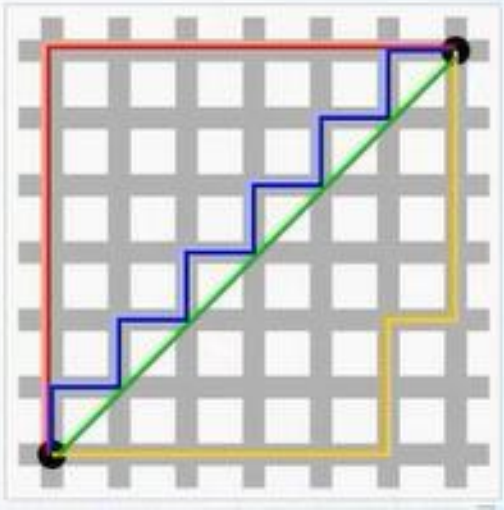
    h_2=所有棋子距离目标位置的曼哈顿距离之和

2	8	3
1	6	4
7		5

初始结点

1	2	3
8		4
7	6	5

目标结点



  - 初始结点的 h 值：

$$h_1 = 4$$

$$h_2 = 1+2+0+1+1+0+0+0 = 5$$

第3讲 搜索问题求解

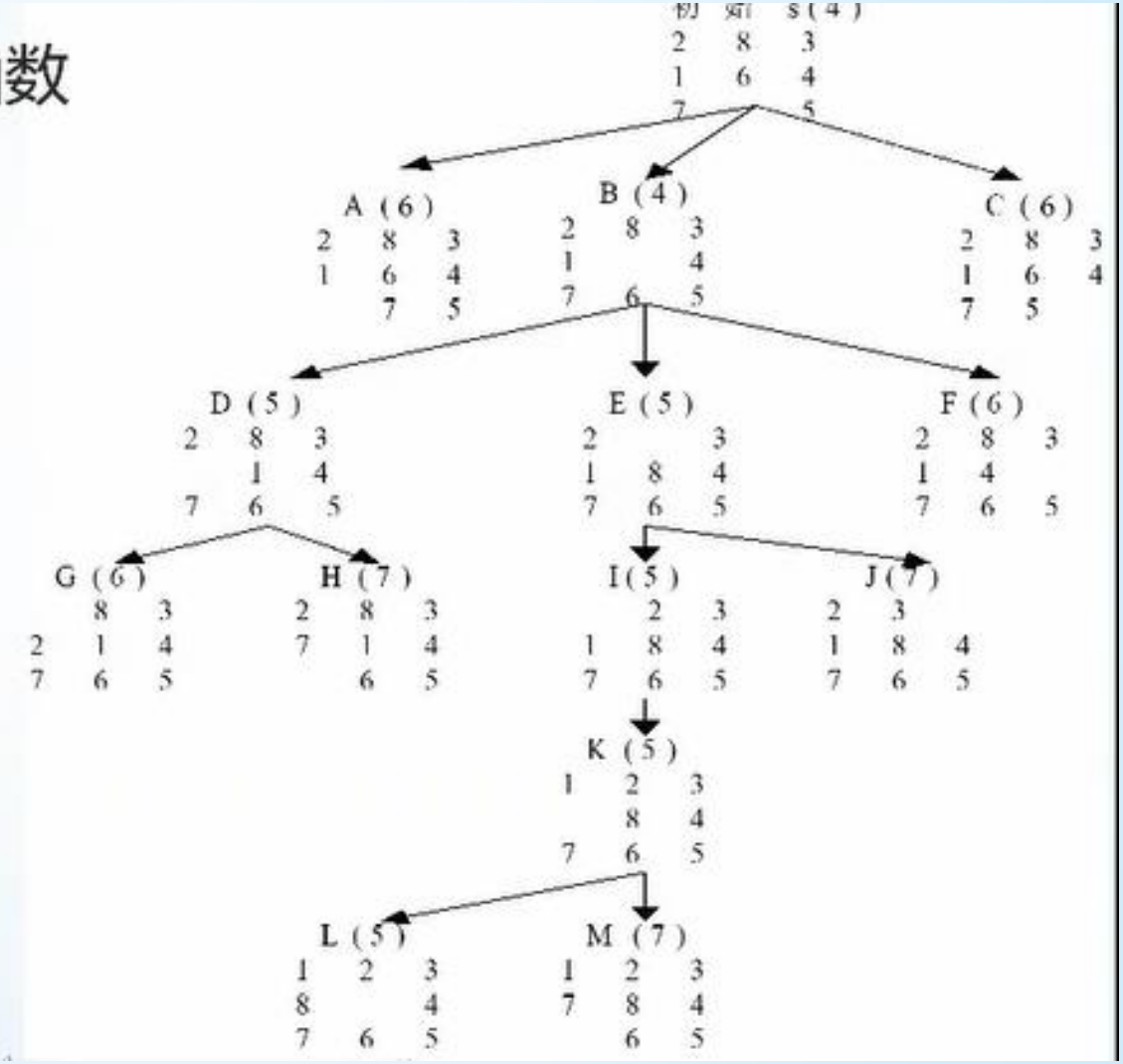
【补充】启发式函数

- A*搜索过程

以h_1为启发式函数

2	8	3
1	6	4
7		5

1	2	3
8		4
7	6	5



作业

- 水桶分水问题：有7L和5L两种规格的空水桶各一只，桶上均没有度量标记，有一个水龙头可以往桶中加水。如何能使大桶里恰好装4L水？
- 罗马尼亚寻径问题，用A*算法，应用直线距离启发式求解从Lugoj到Bucharest问题。给出节点扩展的顺序和每个节点的f,g,h值。