人工智能

- 问题求解与搜索策略

马　超
武汉大学

chaoma@whu.edu.cn

提纲

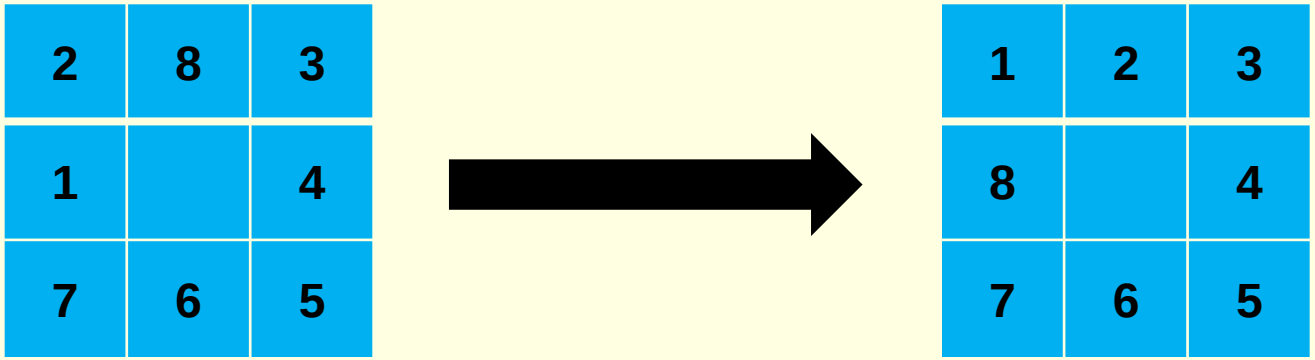
- 问题求解
  - 问题的形式化定义 / 解空间的搜索
- 搜索策略
  - 搜索树的构建
  - 无信息搜索
    - 广度优先 / 深度优先 / 迭代深入 / 代价一致
  - 有信息搜索
    - 贪婪算法 / A*算法
  - 局部搜索
    - 爬山法 / 禁忌算法 / 模拟退火算法
  - 评价指标
    - 完备性 / 最优性 / 时间复杂度 / 空间复杂度

问题求解：概述

- 问题求解是人工智能的核心问题之一
- 问题求解的目的
  - 机器自动找出某问题的正确解决策略
  - 能够举一反三，具有解决同类问题的能力
- 起源于人工智能初期的智力难题、棋类游戏、简单数学定理证明等问题的研究，从而开始形成和发展起来的一大类技术
- 求解的手段多种多样
- 其中搜索技术是问题求解的主要手段之一
  - 问题的形式化定义
  - 解空间的搜索

问题求解：示例

- **八数码问题**
  - 在3×3的棋盘，摆有八个棋子，每个棋子上标有1至8的某个数字。棋盘上还剩余一个空格，与空格相邻的棋子可以移到空格中。



初始状态　　目标状态

- **问题求解：**如何找到一个操作序列将棋盘从初始状态变成目标状态？

问题求解：示例

问题有了，可怎么让计算机理解这些问题呢?

- 如何从武汉大学信息学部选择最短的路径前往黄鹤楼呢？

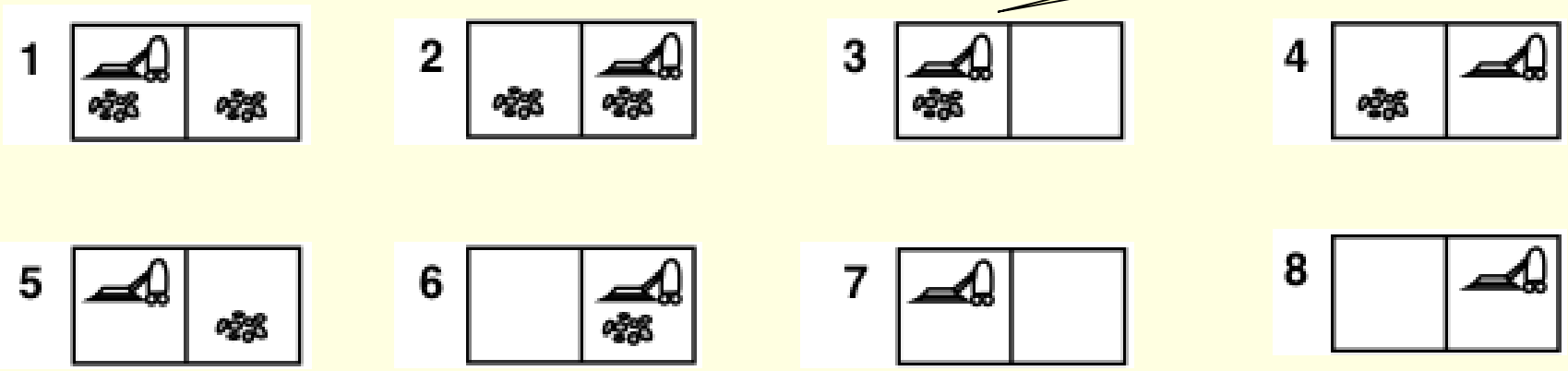
问题求解：问题的形式化定义

- 例：**一个真空吸尘器的世界**
  - 假设：一个吸尘器的世界只有两块地毯大小，地毯或者是脏的，或者是干净的
  - 吸尘器能做的动作只有三个

    {向左(***Left***)，向右(***Right***)，吸尘(***Suck***)}
  - 该场景中一共有多少种可能的状态？

动作

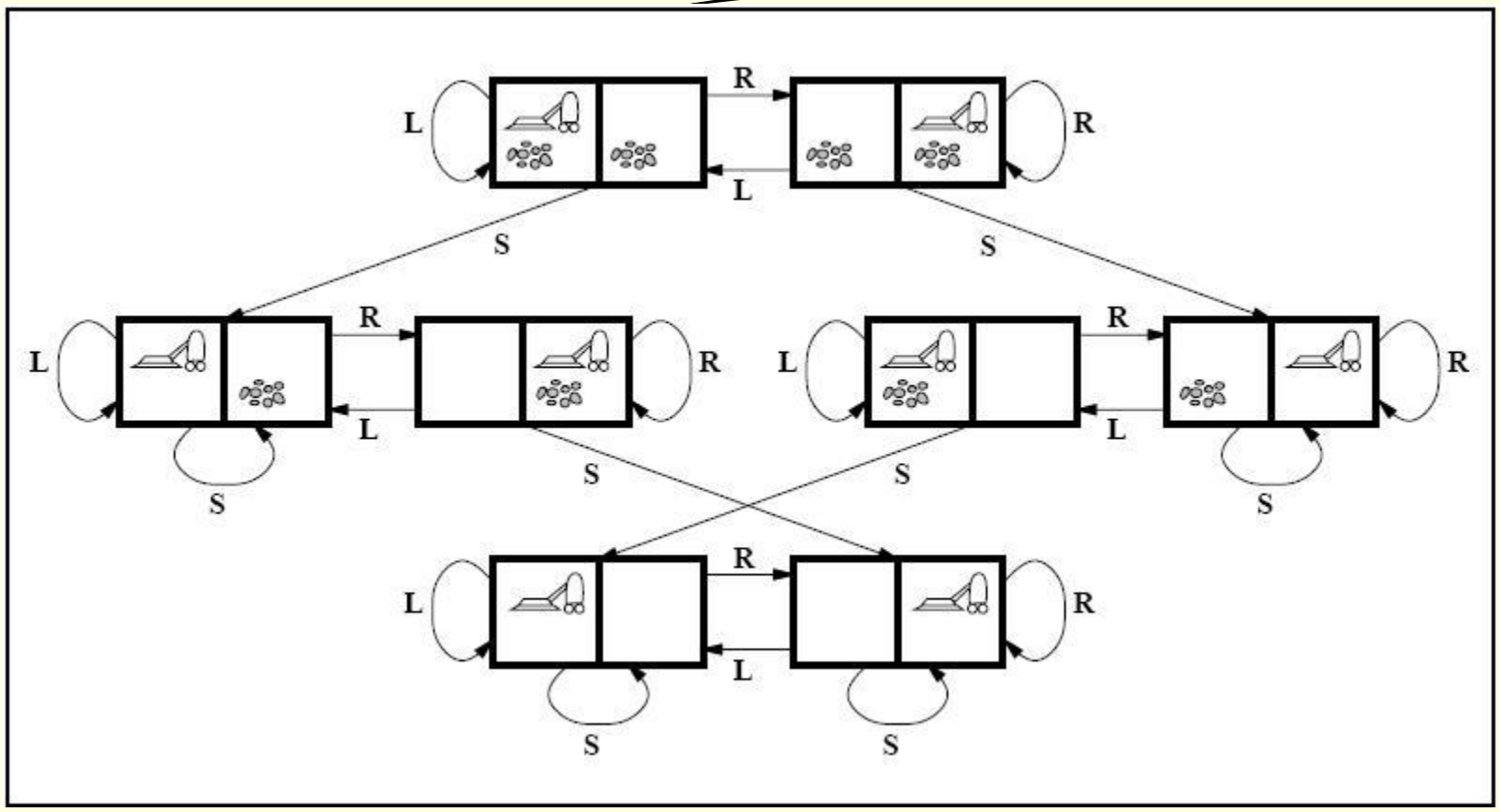
状态



问题求解：问题的形式化定义

- 状态之间通过执行特定动作实现相互转换

状态空间图

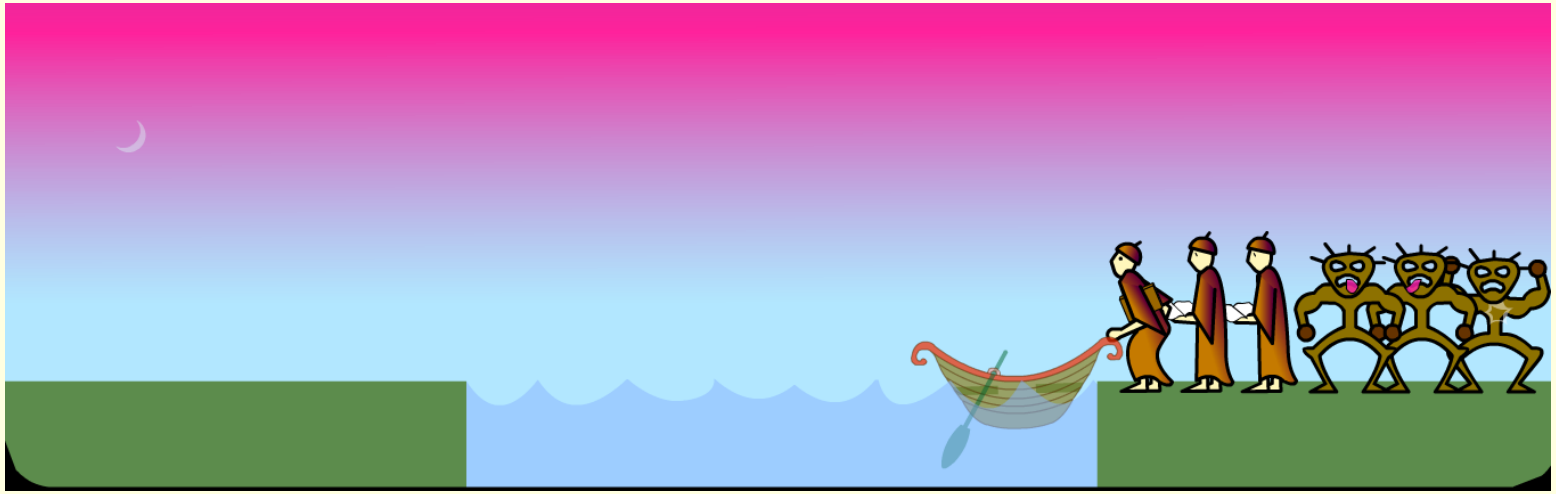


问题求解：问题的形式化定义

- 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（M或C）的船，在河的任意一边或者船上，如果野人的人数多于传教士的人数，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？



问题求解：问题的形式化定义

- 状态及其形式化表示
  - 状态：问题在某一时刻所处的“位置”，“情况”等
  - 状态的形式化表示：根据问题所关心的因素，一般用向量形式表示，向量中的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)

哪些动作能导致状态转换？

初始状态：(0, 0, 0)

目标状态：(3, 3, 1)

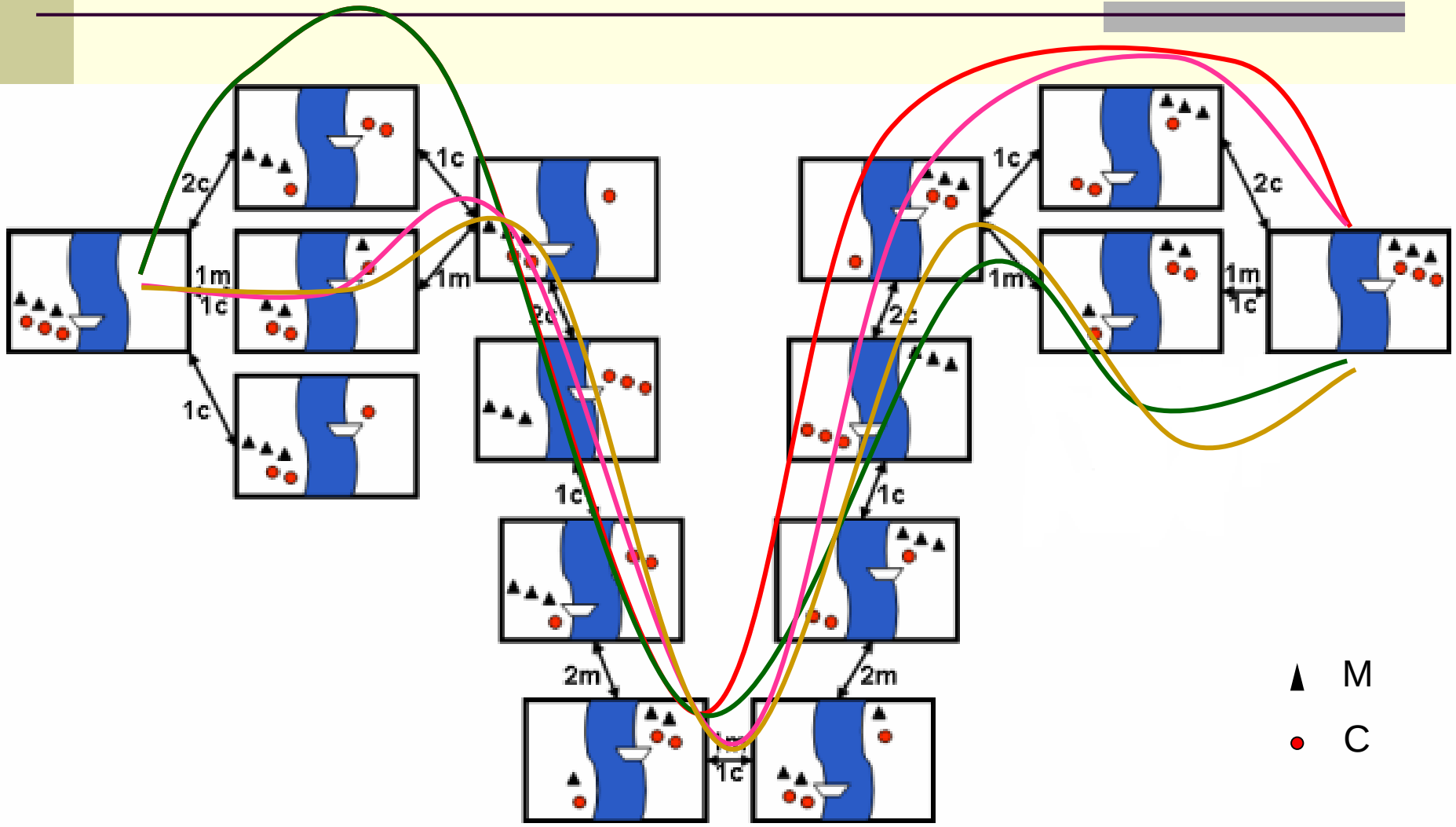
0：右岸

1：左岸

问题求解：问题的形式化定义

- 状态之间的转换
  - 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-r**
      Move-2m-lr　Move-2m-rl　Move-1c-lr
      Move-1c-rl　Move-1m-lr　Move-1m-rl

问题求解：解空间的搜索



问题求解：小结

- 问题的形式化定义
  - 给定：
    - 状态集合$S=\{s_i\}$，算子集合$O=\{o_j\}$，初始状态s_s，目标状态s_t
  - 假设：
    - 根据实际场景设定
  - 目标：
    - 搜索一个操作序列使得初始状态s_s转换为目标状态s_t
- 解空间的搜索
  - 问题求解过程转化为在状态空间图中**搜索**一条从初始节点（初始状态）到目标节点（目标状态）的路径问题。

提纲

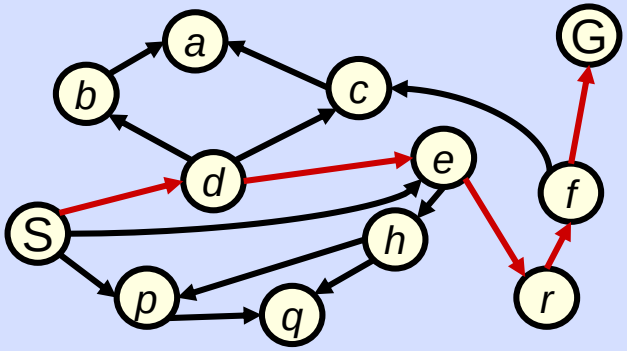
- 问题求解
  - 问题的形式化定义 / 解空间的搜索
- 搜索策略
  - **搜索树的构建**
  - 无信息搜索
    - 广度优先 / 深度优先 / 迭代深入 / 代价一致
  - 有信息搜索
    - 贪婪算法 / A*算法
  - 局部搜索
    - 爬山法 / 模拟退火算法 / 遗传算法
  - 评价指标
    - 完备性 / 最优性 / 时间复杂度 / 空间复杂度

搜索策略：搜索树的构建

- 状态空间图
  - 包含问题所有可能的状态（状态空间爆炸）
    - 时间复杂度：算法运行时间过长
    - 空间复杂度：内存空间不够
- 搜索树
  - 有效消减搜索空间，提高求解效率
  - 重点：如何构建状态树

搜索策略：搜索树的构建

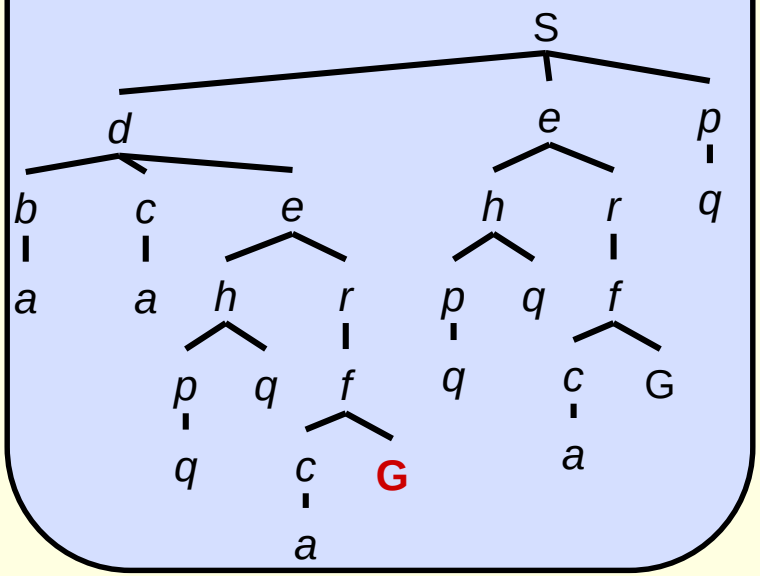
状态空间图



搜索树中的每个叶子节点代表状态空间图中的一条独一无二的完整路径。

按需构建搜索树并尽量使之规模最小。

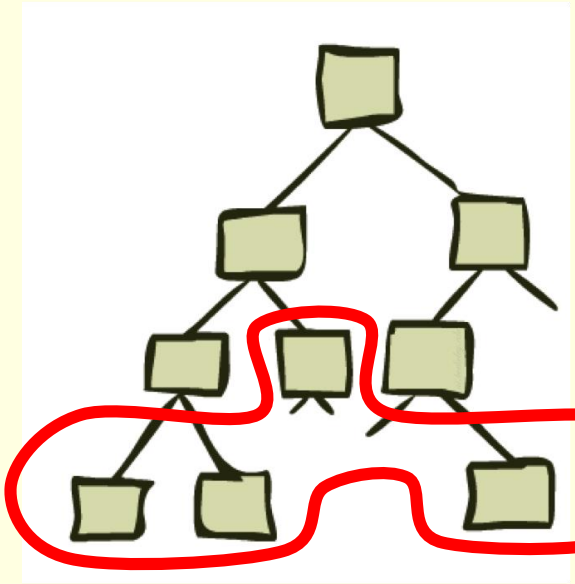
搜索树



基于搜索树的搜索

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- **一般的搜索原则：**
  - *扩展出潜在的节点*
  - *记录已搜索的边缘节点*
  - *试图扩展尽可能少的树节点*



提纲

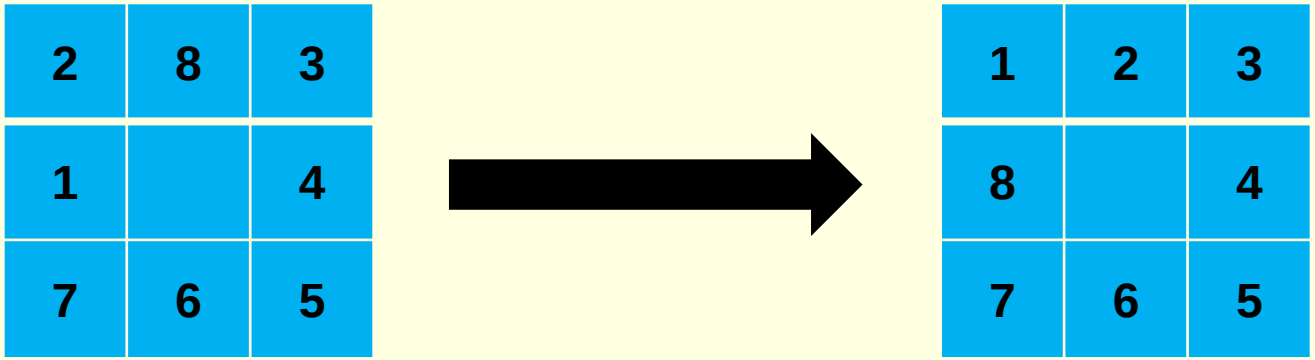
- 问题求解
  - 问题的形式化定义 / 解空间的搜索
- 搜索策略
  - 搜索树的构建
  - **无信息搜索**
    - 广度优先 / 深度优先 / 迭代深入 / 代价一致
  - 有信息搜索
    - 贪婪算法 / A*算法
  - 局部搜索
    - 爬山法 / 禁忌算法 / 模拟退火算法
  - 评价指标
    - 完备性 / 最优性 / 时间复杂度 / 空间复杂度

无信息搜索 (Uninformed Search)

- 无信息搜索（盲目搜索）
  - 广度优先搜索
  - 深度优先搜索
  - 迭代深入搜索
- 该类搜索策略特点
  - 不使用任何与待求解问题相关的先验知识
  - 保证完备性（若有解，则一定可以找到解）
  - 不保证最优性（若有多个解，不保证一定找到最优解）
  - 时间复杂度高
  - 空间复杂度高
  - 不适合大规模状态空间的问题求解

无信息搜索：示例

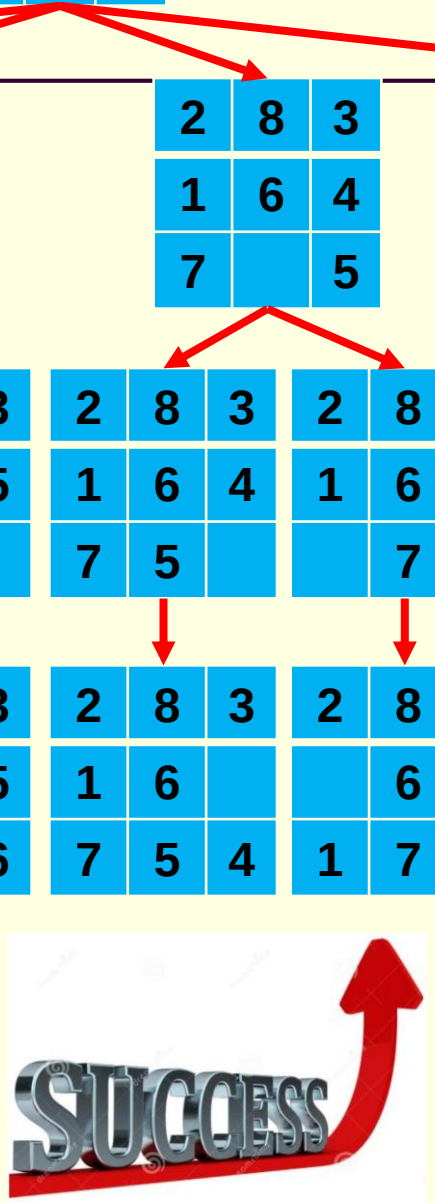
- **八数码问题**
  - 在3×3的棋盘，摆有八个棋子，每个棋子上标有1至8的某个数字。棋盘上还剩余一个空格，与空格相邻的棋子可以移到空格中。



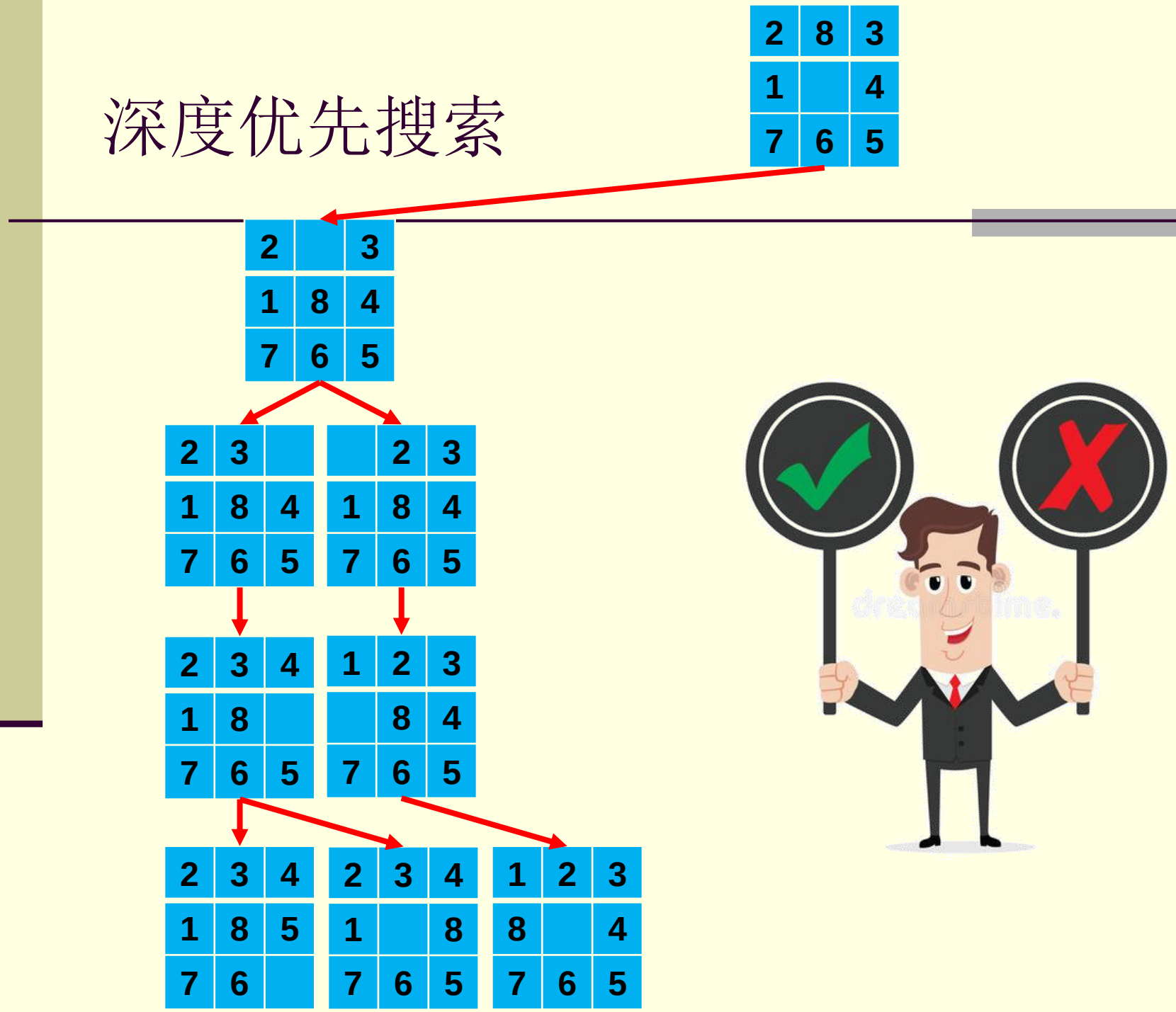
初始状态　　目标状态

- **问题求解：**如何找到一个操作序列将棋盘从初始状态变成目标状态？

广度优先搜索



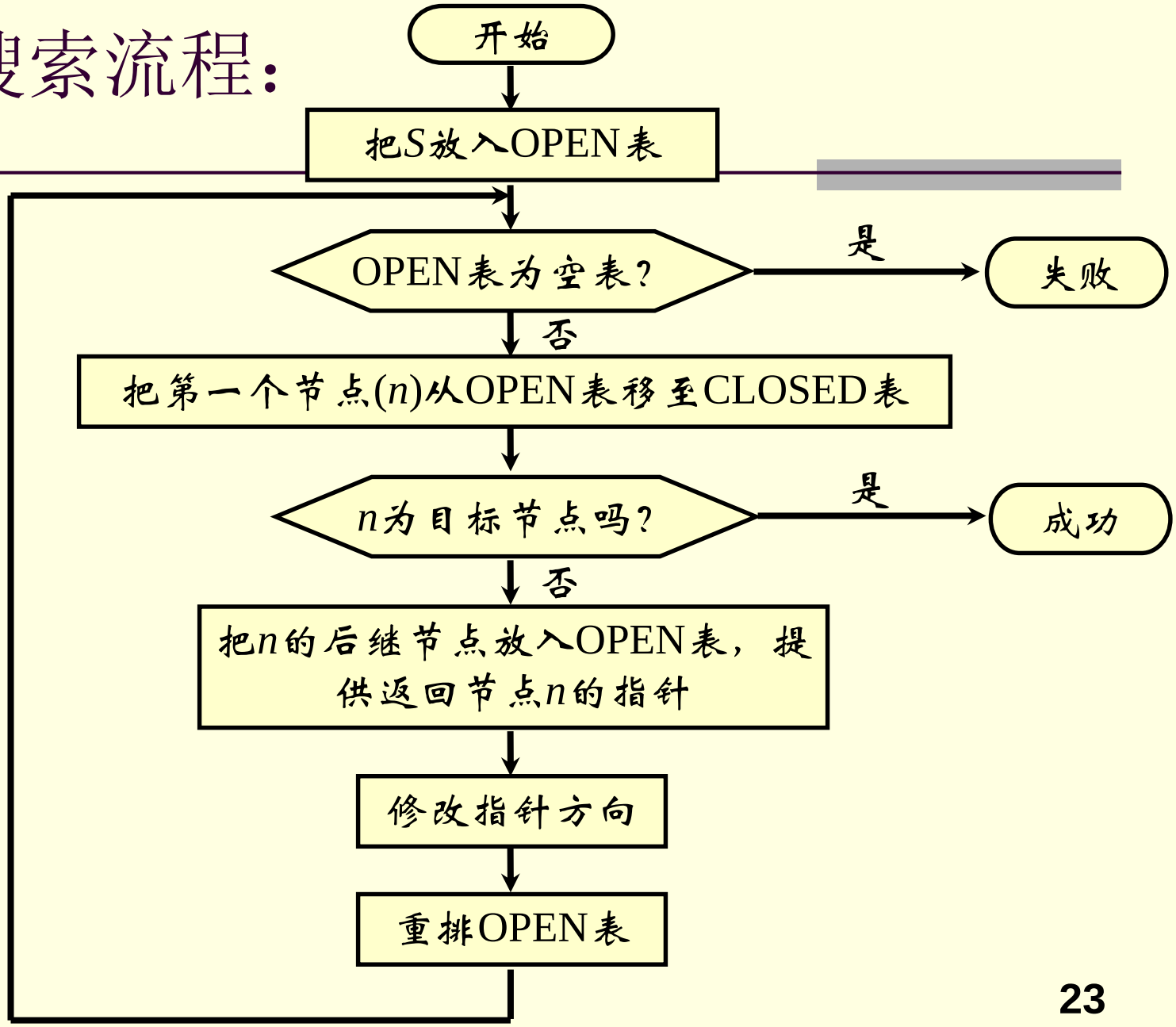
深度优先搜索


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

搜索树的关键数据结构：

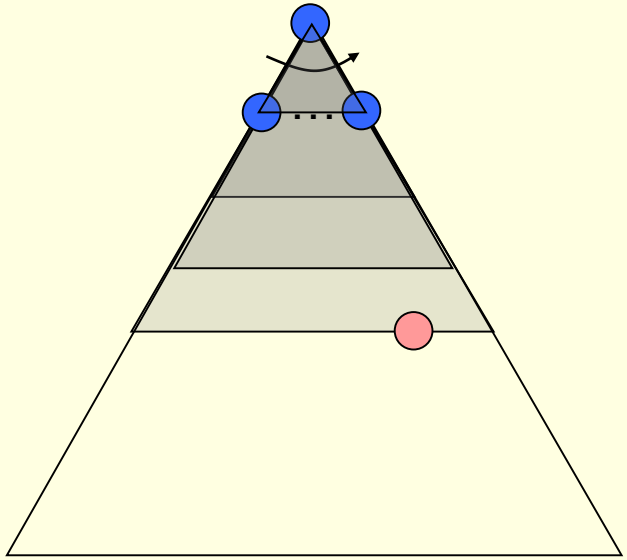
- 判断从当前状态还可以探索哪些状态
  - OPEN表
- 记录所有已经探索过的状态
  - CLOSED表
- 记录从目标状态返回初始状态的路径
  - 搜索树中的每个节点（根节点除外）必须包含指向父节点的指针

一般搜索流程：


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
n为目标节点吗？
是
成功
否
把n的后继节点放入OPEN表，提供返回节点n的指针
修改指针方向
重排OPEN表

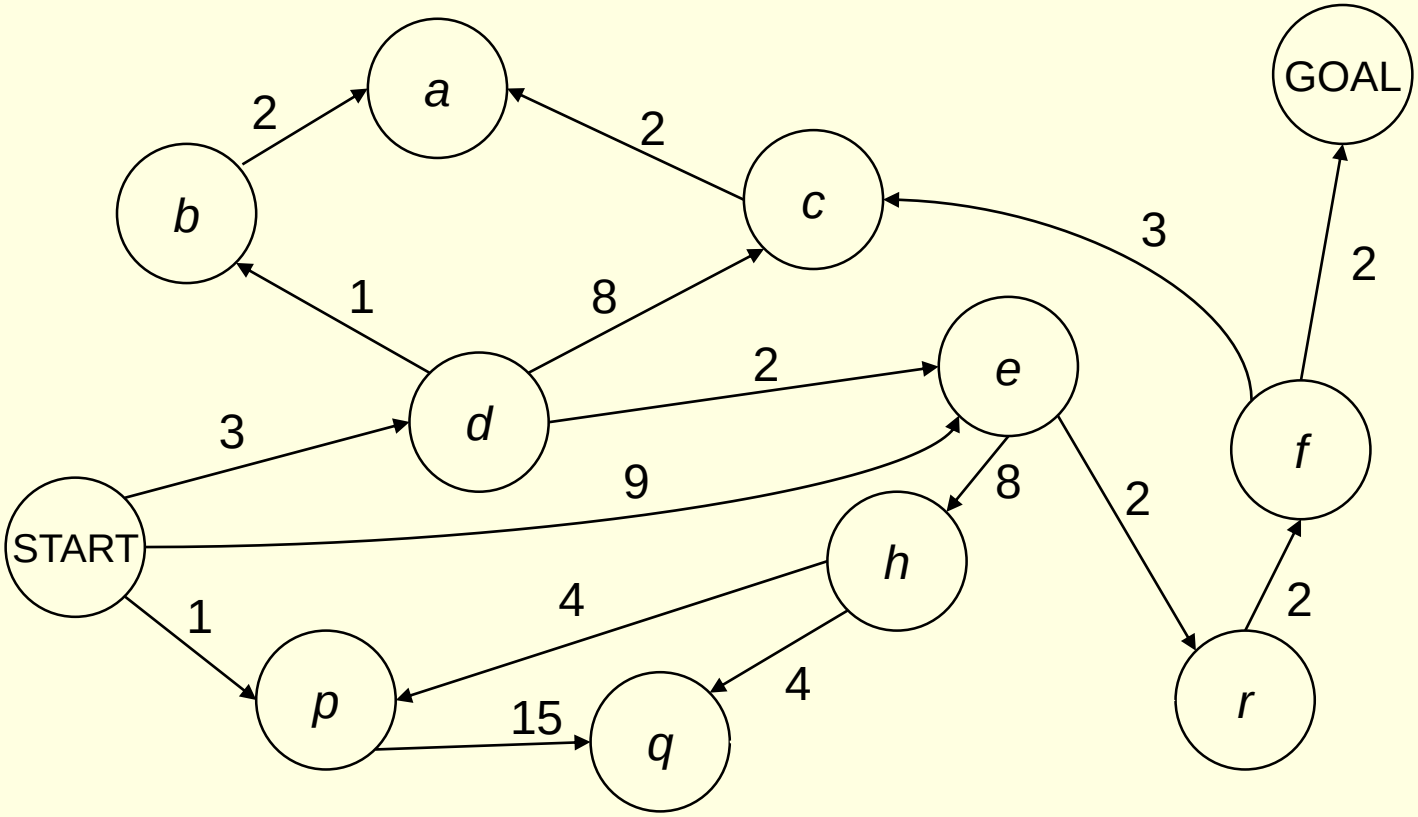
迭代深入搜索

- **思路：**结合深度优先与广度优先
  - 在深度为d的范围内进行深度优先搜索，若无解
  - 在深度为2d的范围内进行深度优先搜索，若无解
  - 在深度为3d的范围内进行深度优先搜索，若无解
  - ……



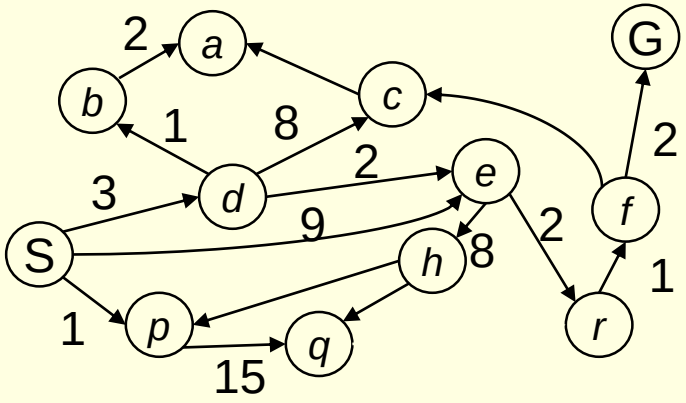
- 浪费冗余？
  - 通常大多数的节点都在底层，所以上层的节点生成多次影响不是很大。

代价敏感搜索(Cost-Sensitive Search)



- 广度优先搜索
  - 仅能保证找到一个包含动作数最少的解
  - 无法保证找到一个总代价最小的解

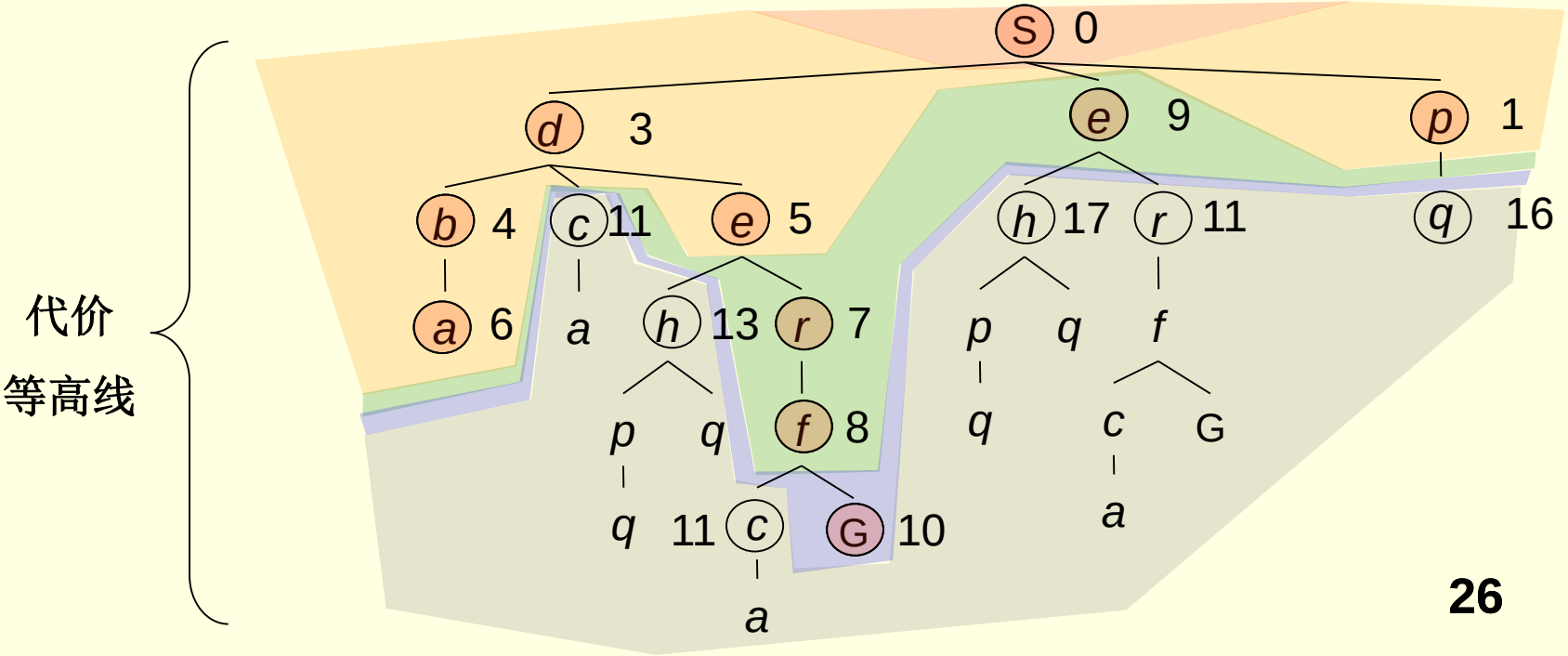
代价一致搜索(Uniform Cost Search)



策略：扩展代价最小的后继节点:

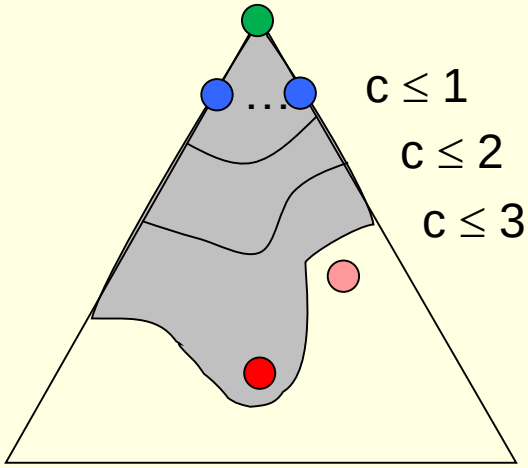
边缘节点：优先队列

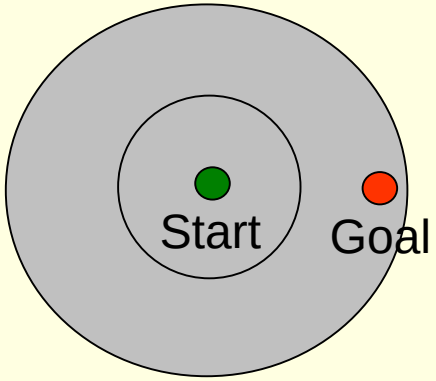
(优先级：累积代价)



代价一致搜索 (UCS) 特性

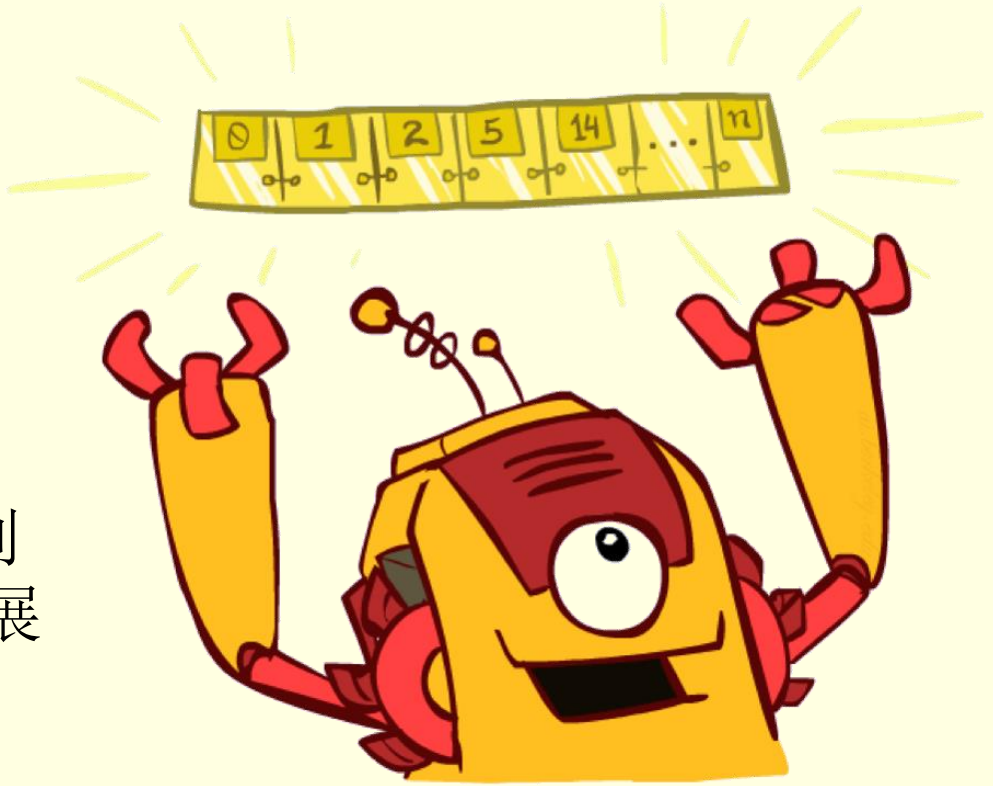
- **UCS探索了递增的代价等高线**
- **优点：**
  - 完备性
  - 最优性
- **缺点：**
  - 在每一个“方向”上进行探索
  - 没有关于目标信息





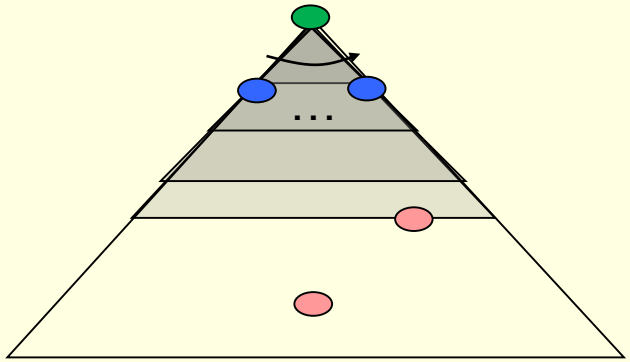
无信息搜索算法之间的异同

- **基本数据结构类似**
  - OPEN表
  - CLOSED表
  - 指向父节点的指针
- **边缘处理策略不同**
  - 所有的边缘是优先队列(即附加优先级的可扩展后继节点集合)

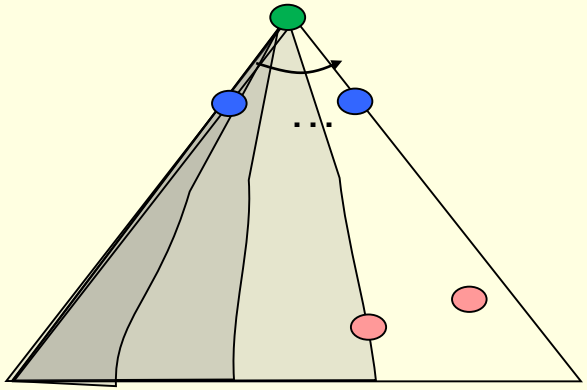


无信息搜索算法之间的异同（续上）

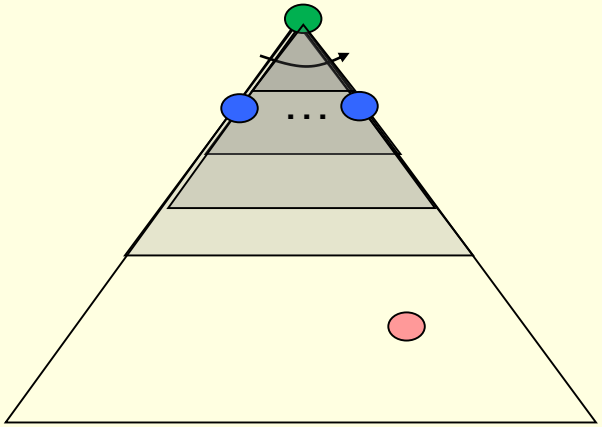
广度优先搜索



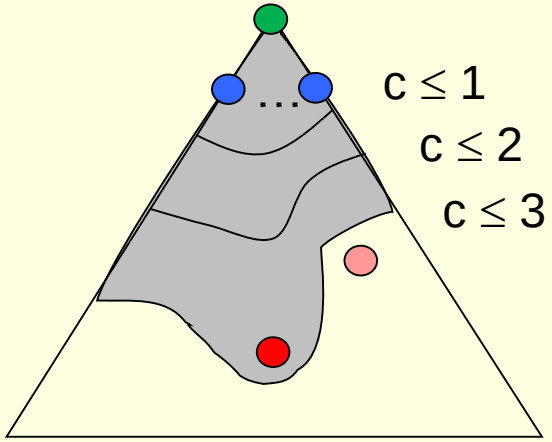
深度优先搜索



迭代深入搜索

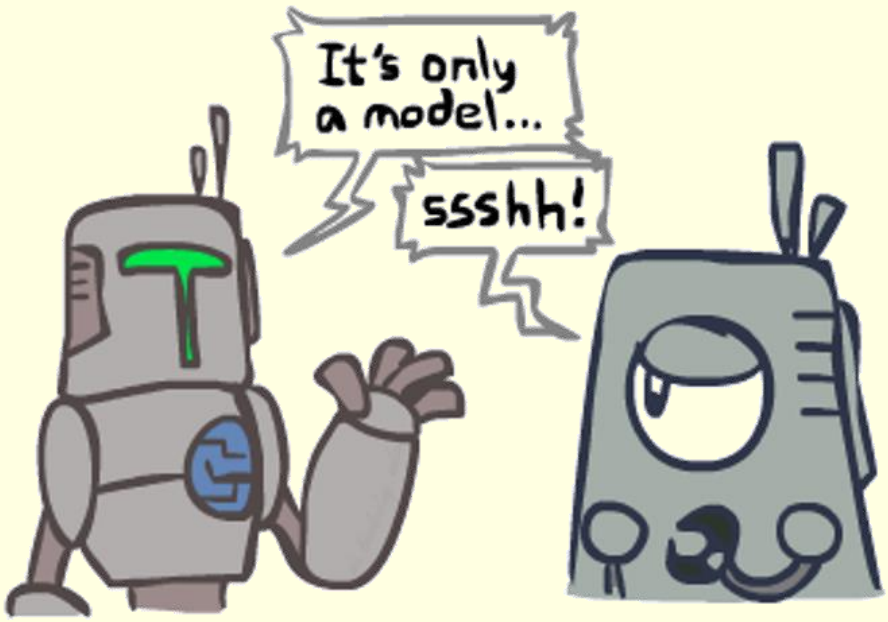


代价一致搜索



搜索与模型

- **搜索是在问题世界的模型执行**
  - 实际上并不在真实世界上试验所有的规划
  - 规划全部是在“模拟中”
  - 你的搜索只能和你的模型一样好…



提纲

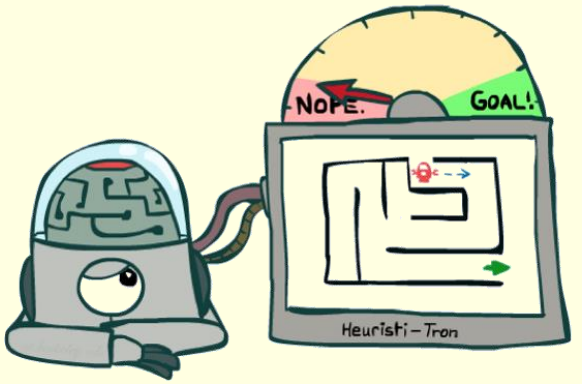
- 问题求解
  - 问题的形式化定义 / 解空间的搜索
- 搜索策略
  - 搜索树的构建
  - 无信息搜索
    - 广度优先 / 深度优先 / 迭代深入 / 代价一致
  - **有信息搜索**
    - 贪婪算法 / A*算法
  - 局部搜索
    - 爬山法 / 禁忌算法 / 模拟退火算法
  - 评价指标
    - 完备性 / 最优性 / 时间复杂度 / 空间复杂度

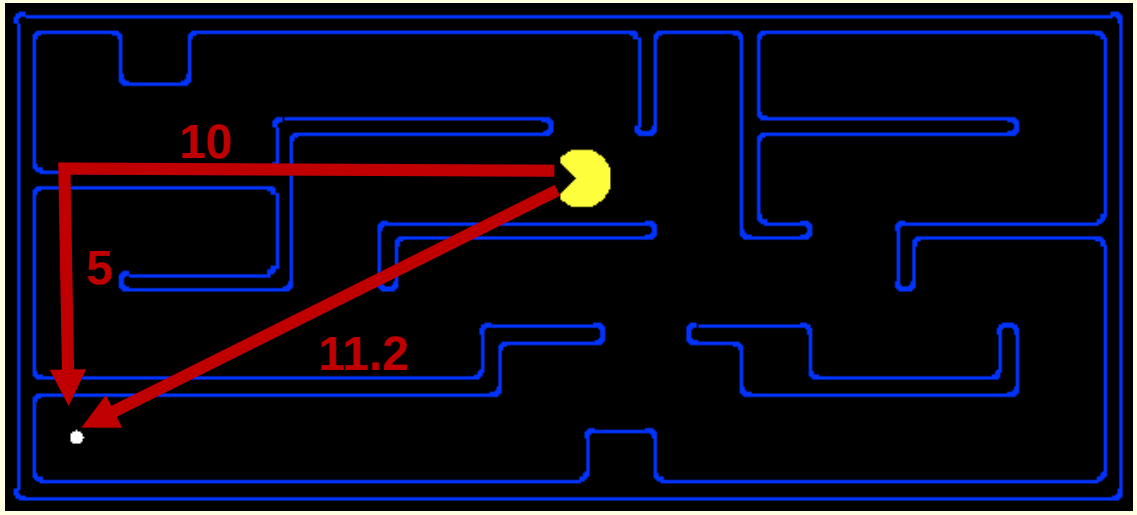
有信息搜索 (Informed Search)

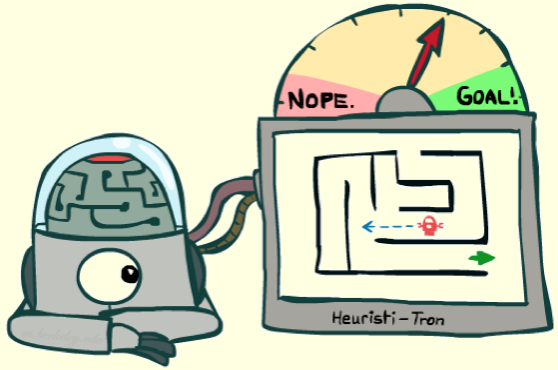
- 有信息搜索（启发式搜索）
  - 贪婪算法（贪心算法）
  - A*算法
- 该类搜索策略特点
  - 使用与待求解问题相关的先验知识
    - 启发式函数
  - 保证完备性（若有解，则一定可以找到解）
  - 保证最优性（若有多个解，保证一定找到最优解）
  - 时间复杂度可控（启发式函数引入有限的额外计算）
  - 空间复杂度较低（相较广度优先搜索而言）
  - 较为适合大规模状态空间的问题求解

有信息搜索：启发策略

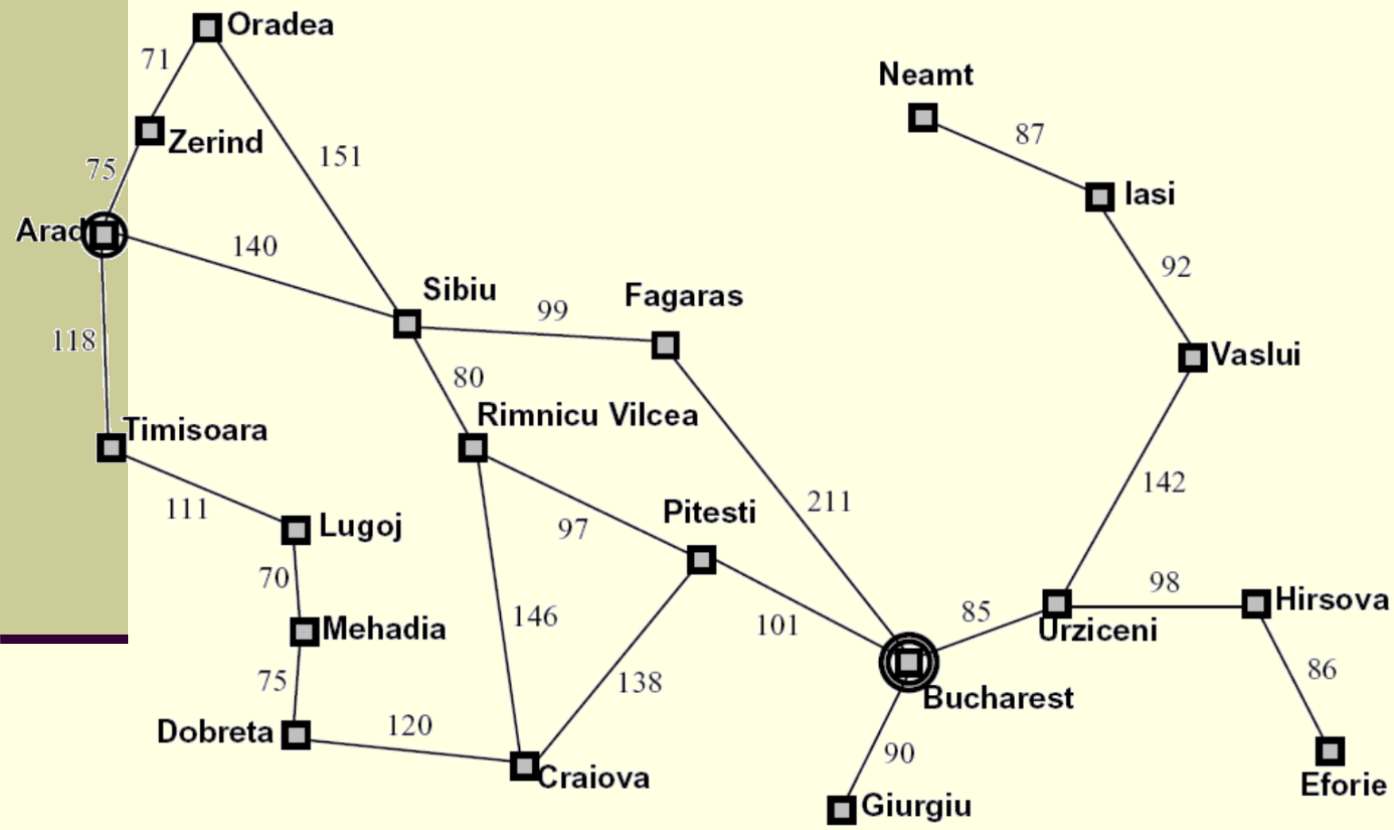
- **启发策略：**
  - *估计*一个状态到目标距离的函数
  - 问题给予算法的额外信息，为特定搜索问题而设计
  - 例：曼哈顿距离，欧几里得距离等





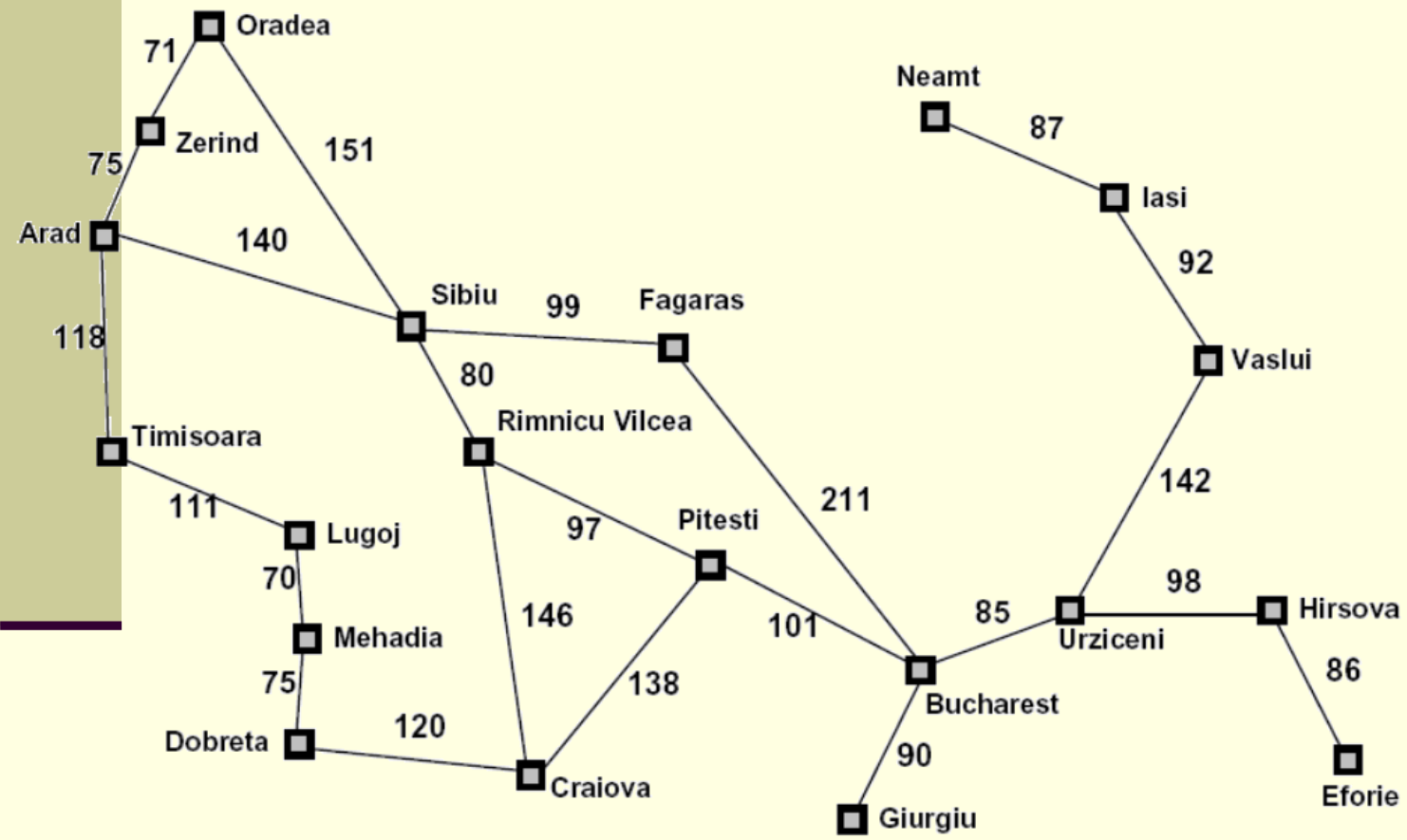


示例：罗马尼亚旅行



- 状态空间：
  - **Cities**
- 后继函数：
  - **Roads**
- 初始状态：
  - **Arad**
- 目标状态：
  - **Bucharest**

启发函数：示例

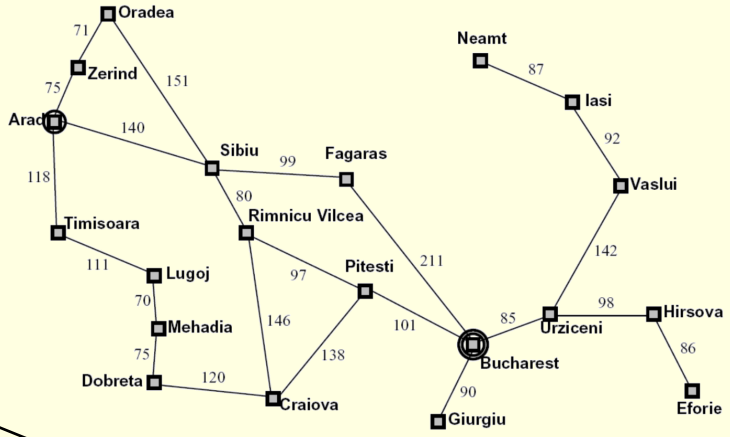


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

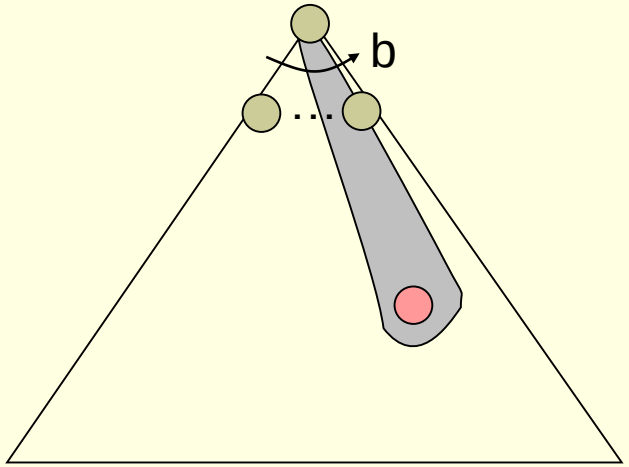
贪婪搜索(Greedy Search)：示例

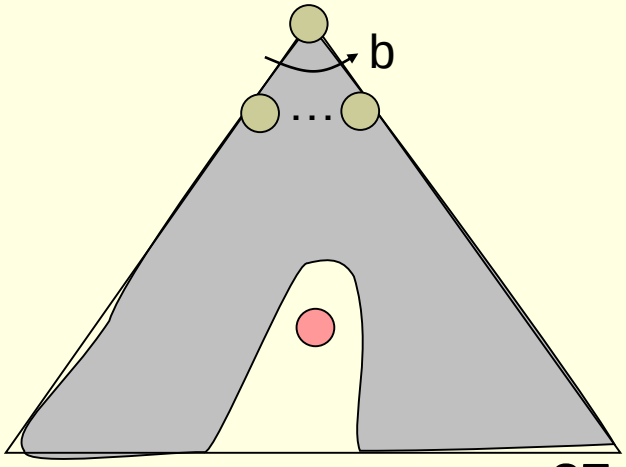
- 扩展距离目标状态最近的节点



贪婪搜索(Greedy Search)：分析

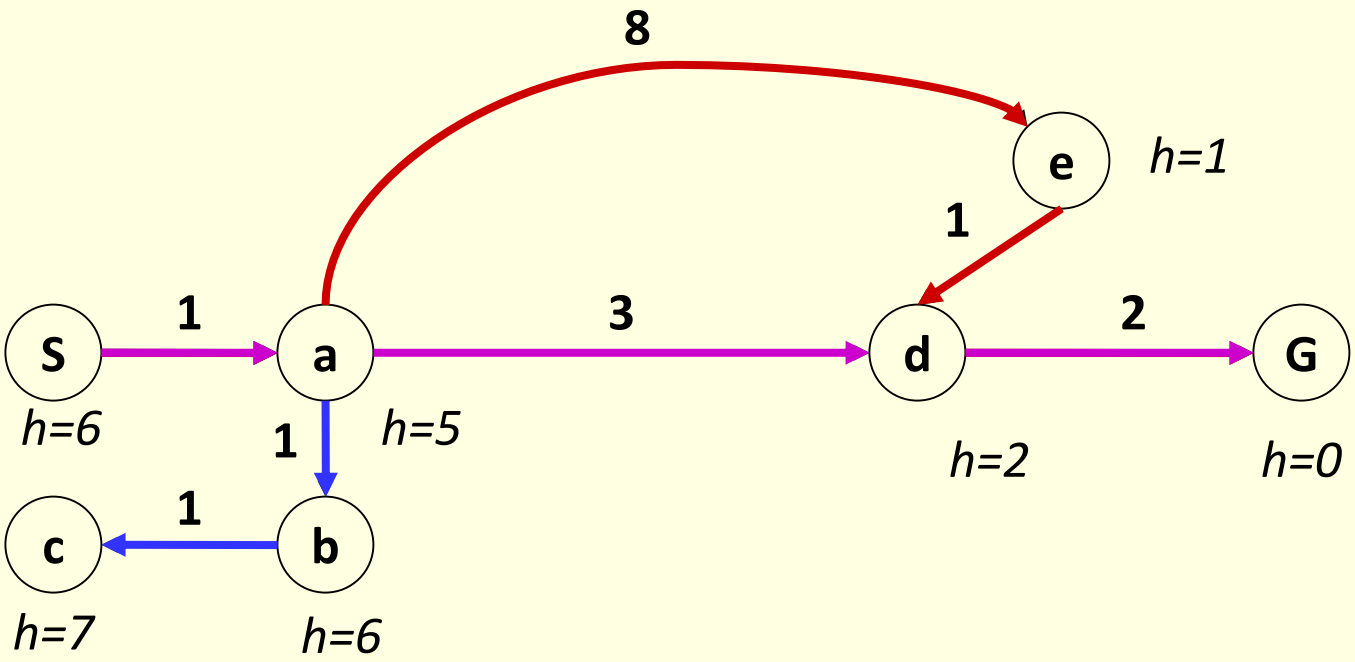
- 策略: 扩展你认为最接近目标状态的节点
  - 启发式: 对每个边缘节点估计其到目标节点的最近距离
  - 只使用启发函数 **f(n)=h(n)** 来评价节点
- 通常情况:
  - 最佳优先使你直接（或很快）到达目标
- 最坏情况:
  - 类似深度优先搜索



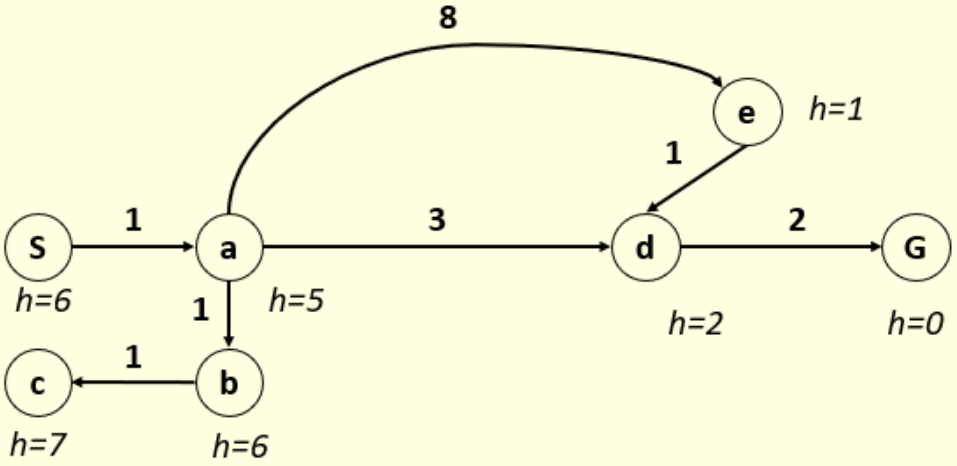


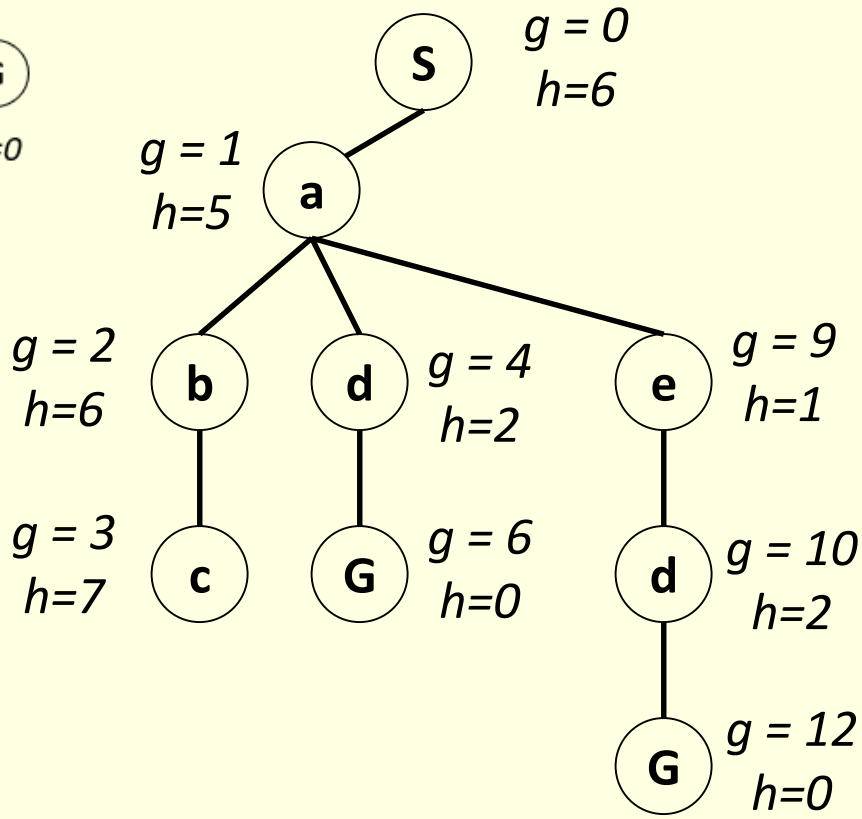
A*算法(A* Search)：UCS + Greedy

- UCS依据初始状态S到当前状态n的路径代价排序-后向代价g(n)
- Greedy依据当前状态n到目标状态G的估计路径代价排序-前向代价h(n)
- A*依据二者的和进行排序：f(n) = g(n) + h(n)



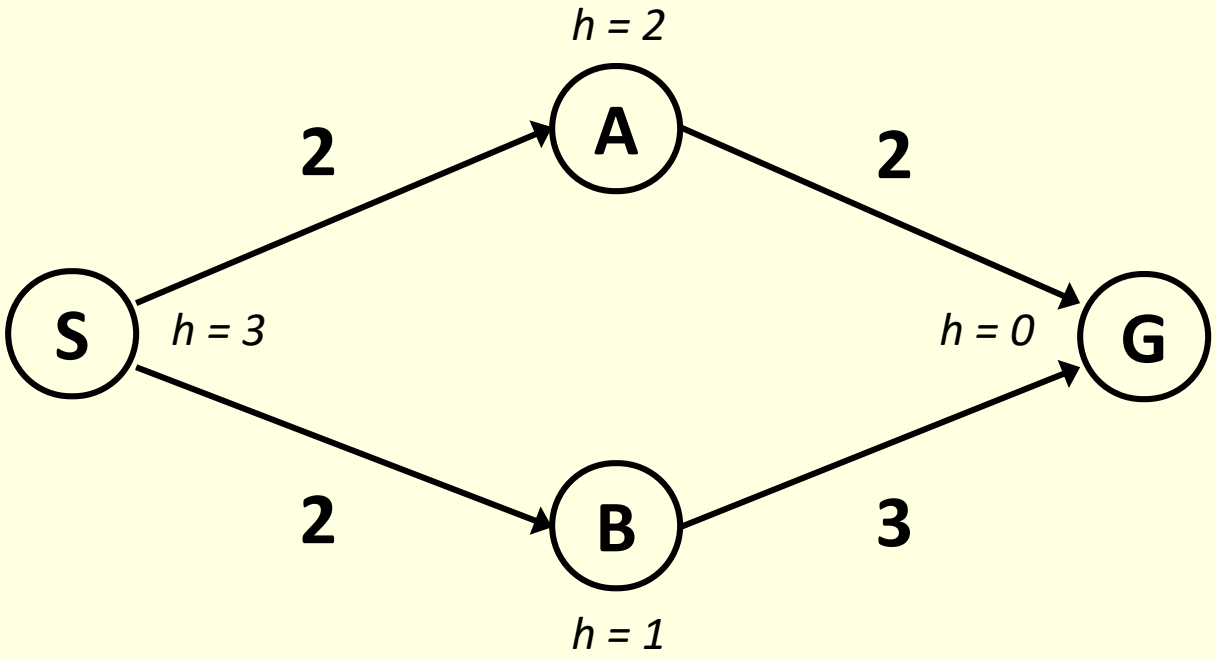
A*算法(A* Search)：UCS + Greedy





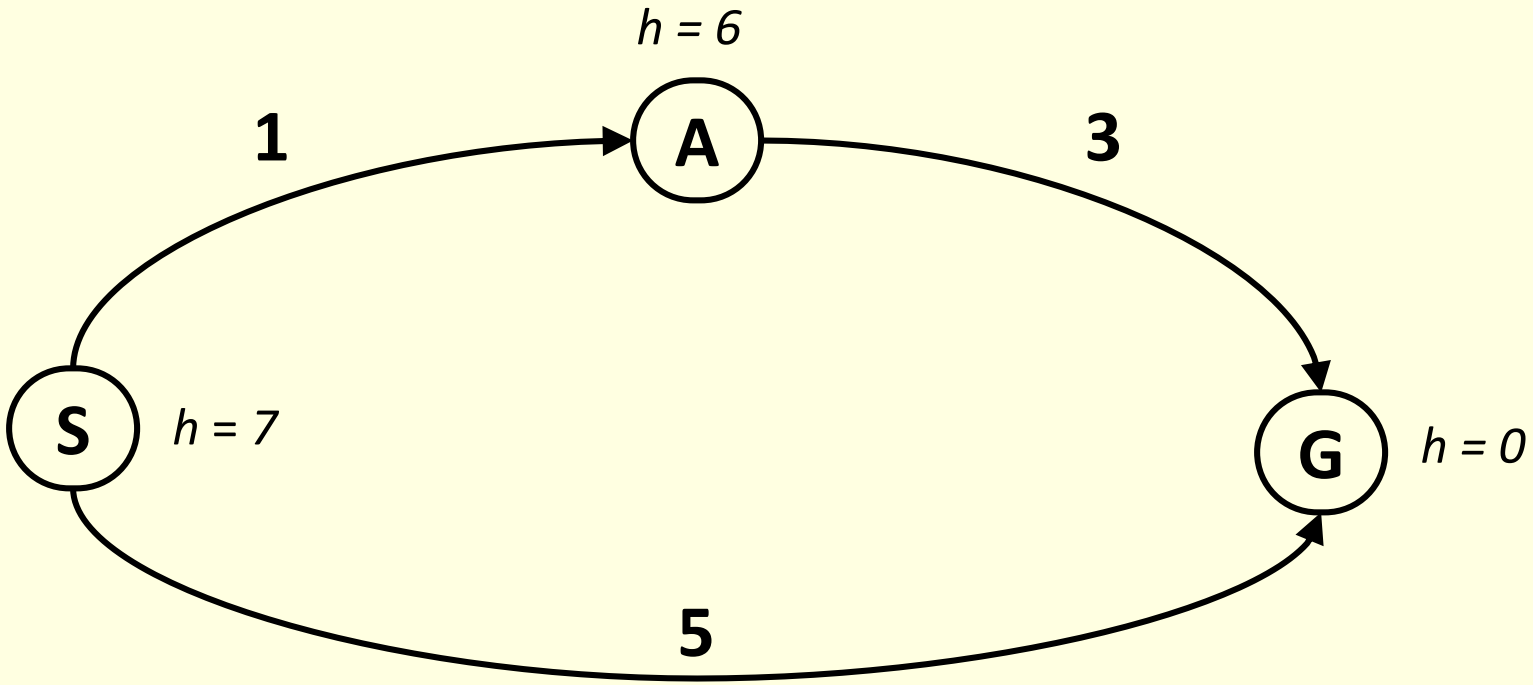
A*算法(A* Search)：停止条件

- 当目标状态入列时，应该停止吗？



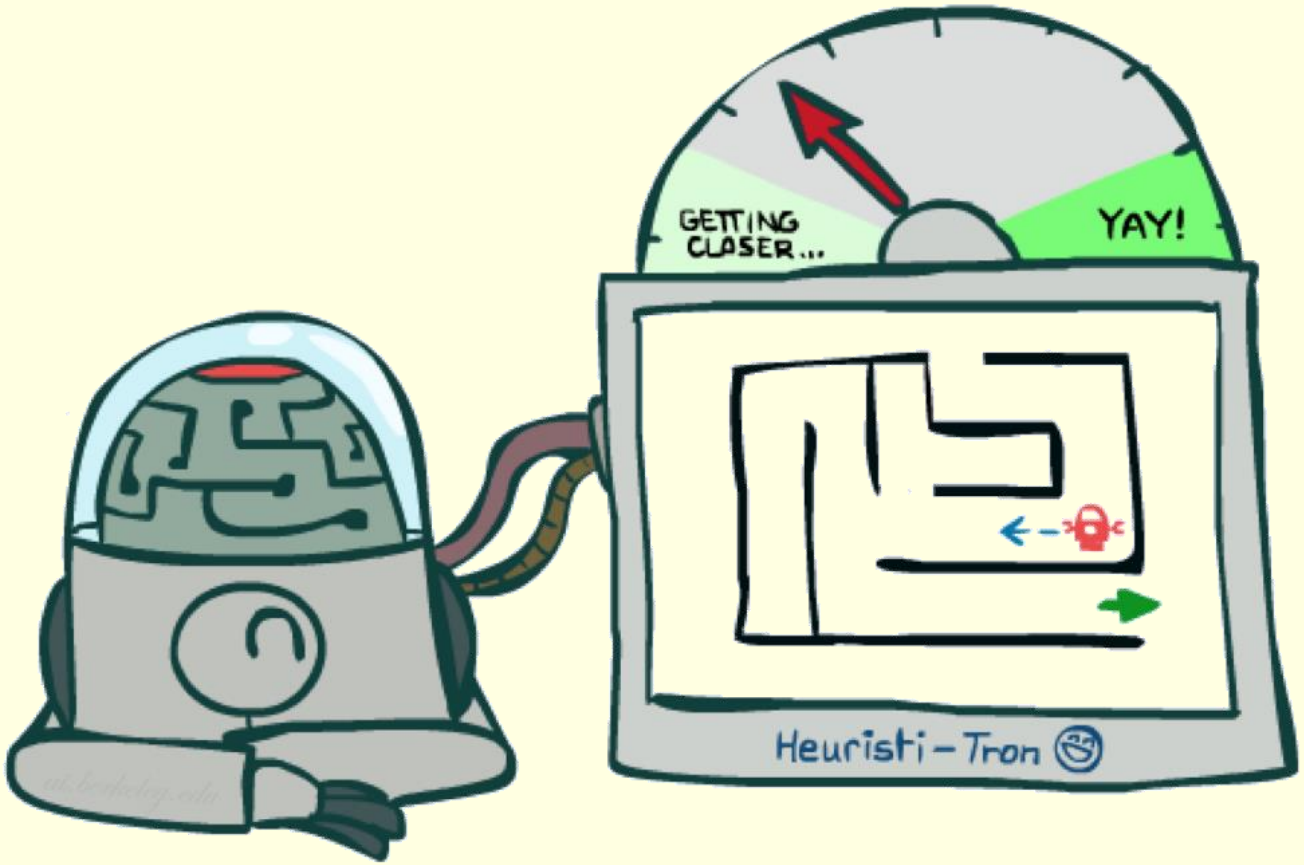
- **NO：只有目标状态出列时才停止！**

A*算法(A* Search)：启发函数



- 哪错了?
- 估计目标耗散 > 实际目标耗散
- 需要估计耗散要小于等于实际耗散!

启发函数：可采纳



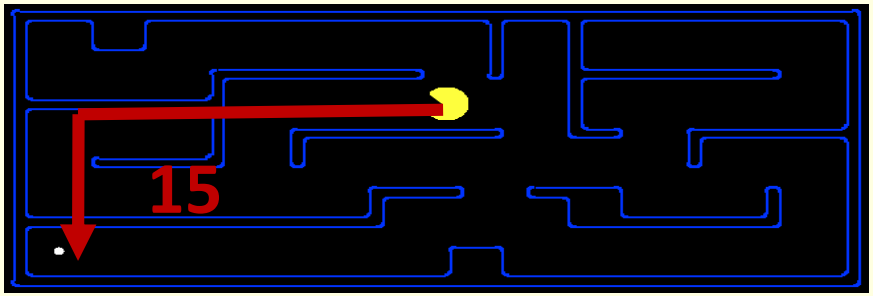
启发函数：可采纳

- 启发函数 **h** 是 **可采纳的**，那么：

$$0 \leq h(n) \leq h^*(n)$$

  - 其中**h*(n)**是当前节点**n**到目标节点的**真实耗散**

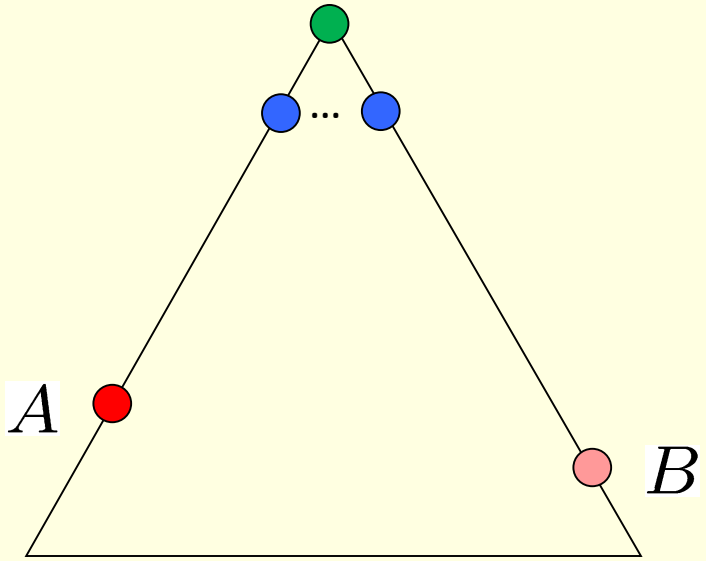
- 例如：



- 如何设计可采纳的启发函数是**A***算法实际应用中的重点

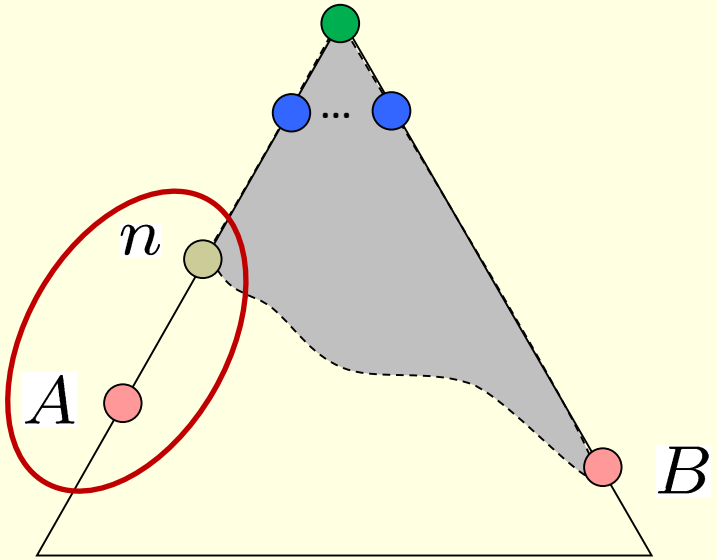
A*树搜索算法：最优性

- 假定：
  - A为最优目标节点
  - B为次优目标节点
  - h(n)为可采纳的启发函数
- 结论：
  - A在B之前离开边缘集合



A*树搜索算法：最优性

- 证明：
  - 假设 B 在边缘集合上
  - A的某个祖先节点n也在边缘集合上 (maybe A!)
  - 那么 n 将在B之前被扩展
    - f(n) <= f(A)



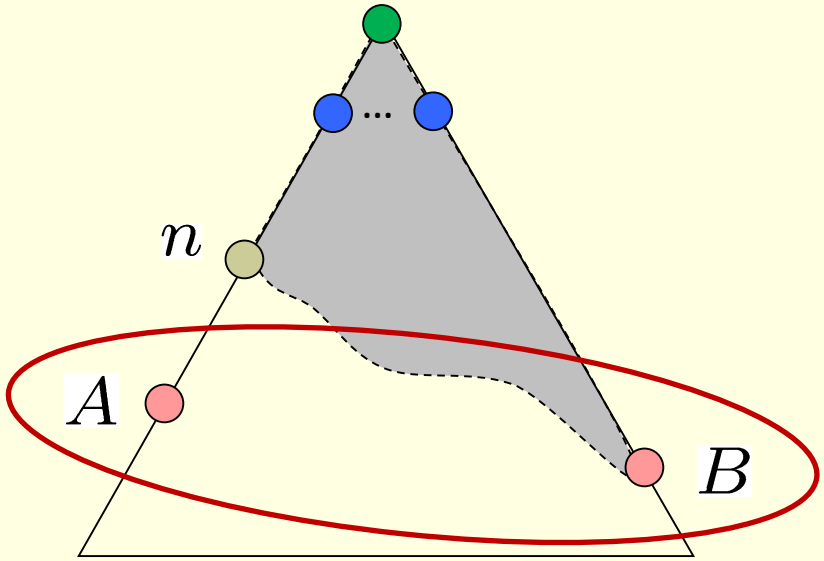
$f(n) = g(n) + h(n)$ 代价函数f的定义

$f(n) \leq g(A)$ A是最优解

$g(A) = f(A)$ h(A) = 0

A*树搜索算法：最优性

- 证明：
  - 假设 B 在边缘集合上
  - A的某个祖先节点n也在边缘集合上 (maybe A!)
  - 那么 n 将在B之前被扩展
    - f(n) <= f(A)
    - f(A) < f(B)

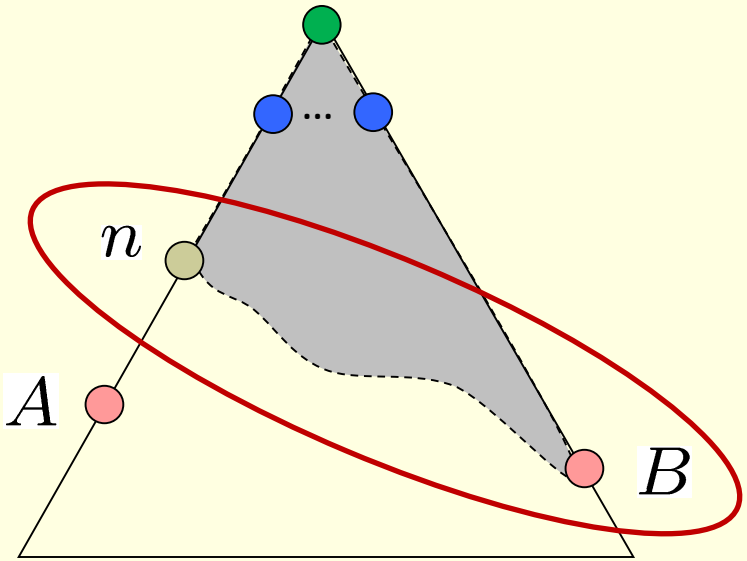


$g(A) < g(B)$ B 是次优解

$f(A) < f(B)$ h(A) = 0

A*树搜索算法：最优性

- 证明：
  - 假设 B 在边缘集合上
  - A的某个祖先节点n也在边缘集合上 (maybe A!)
  - 那么 n 将在B之前被扩展
    - f(n) <= f(A)
    - f(A) < f(B)
    - n 一定在 B 之前被扩展
  - A的所有祖先在B之前扩展
  - A在B之前扩展
  - A*是最优的



$$f(n) \leq f(A) < f(B)$$

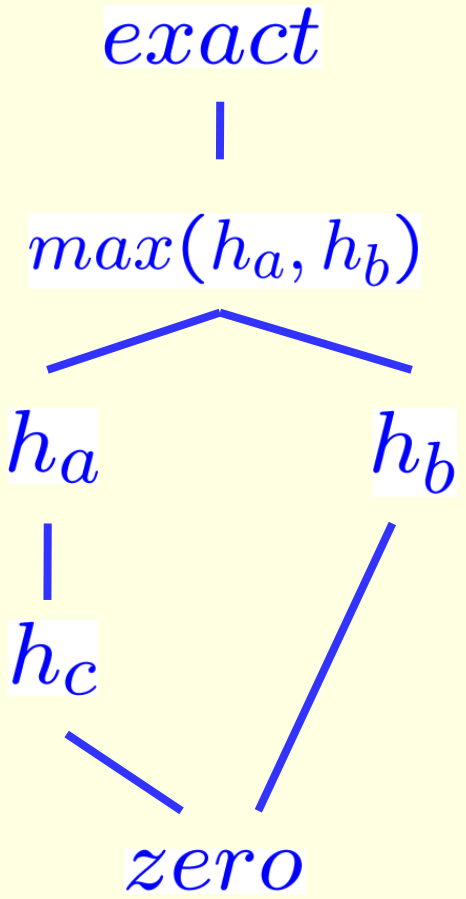
启发函数：特性

- 占优势: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

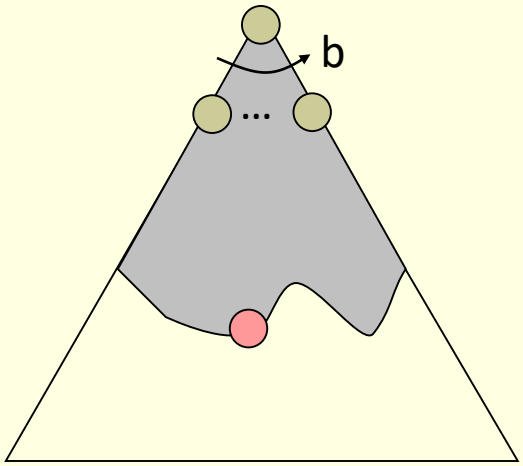
- 可采纳的启发函数集合:

$$h(n) = max(h_a(n), h_b(n))$$

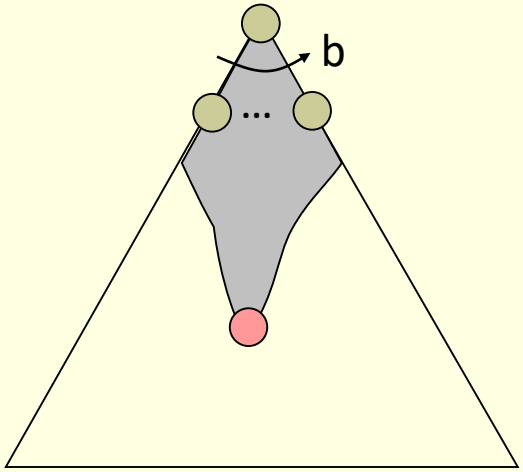


A*树搜索算法特性

代价一致搜索(UCS)

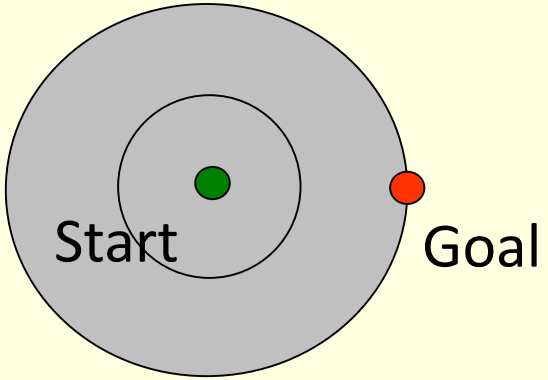


A*树搜索算法

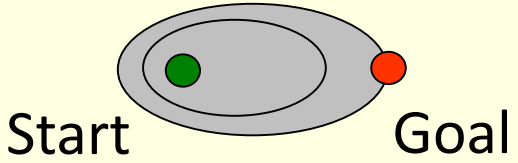


UCS vs. A*

- 代价一致搜索（UCS）在所有可能的搜索“方向”上等概率地扩展；



- A*搜索受启发函数的引导主要朝目标节点扩展，而且能够保证最优性。



实例对比

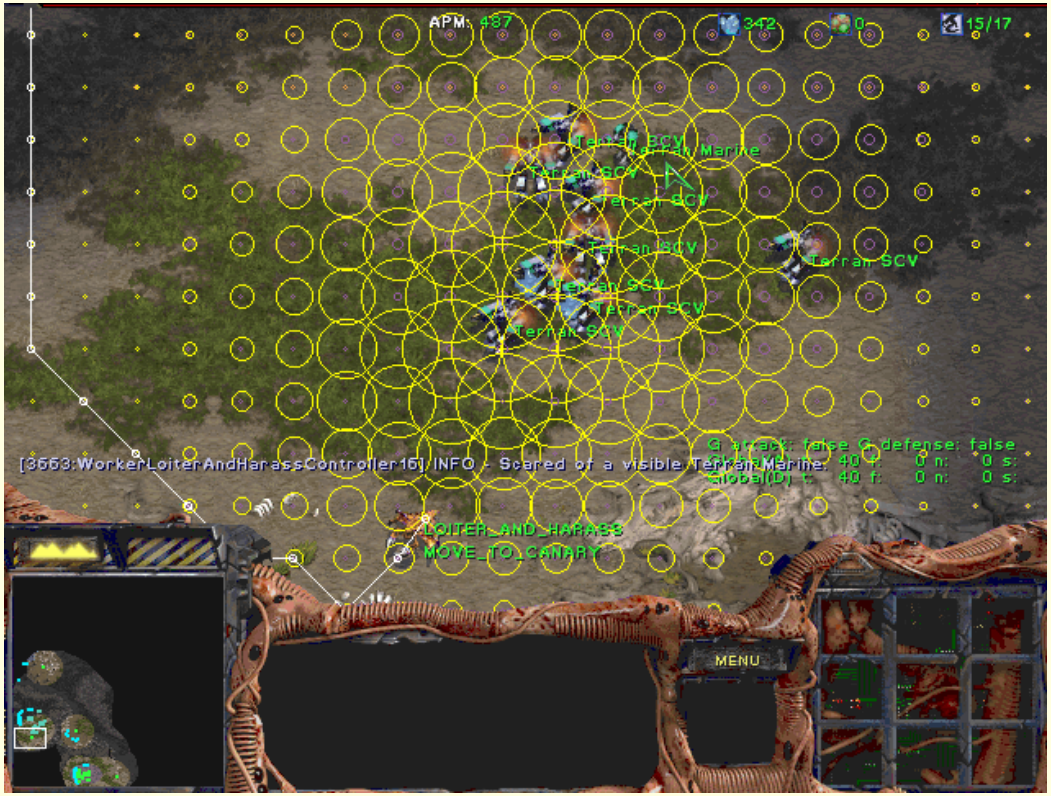
贪婪算法

代价一致搜索

A*算法

A*算法的应用

- 视频游戏
- 人脸识别
- 路径规划
- 资源调度/优化
- 机器人动作规划
- 机器翻译
- 自然语言分析
- 语音识别
- …



提纲

- 问题求解
  - 问题的形式化定义 / 解空间的搜索
- 搜索策略
  - 搜索树的构建
  - 无信息搜索
    - 广度优先 / 深度优先 / 迭代深入 / 代价一致
  - 有信息搜索
    - 贪婪算法 / A*算法
  - **局部搜索**
    - 爬山法 / 禁忌算法 / 模拟退火算法
  - 评价指标
    - 完备性 / 最优性 / 时间复杂度 / 空间复杂度

搜索问题的分类

- **普通搜索问题：**
  - 求出一条从初始状态到目标状态之间的行动序列
- **全局搜索问题：**
  - 求出所有从初始状态到目标状态之间的行动序列
- **最优化搜索问题：**
  - 求出从初始状态到目标状态之间耗散最少的行动序列

搜索策略的分类

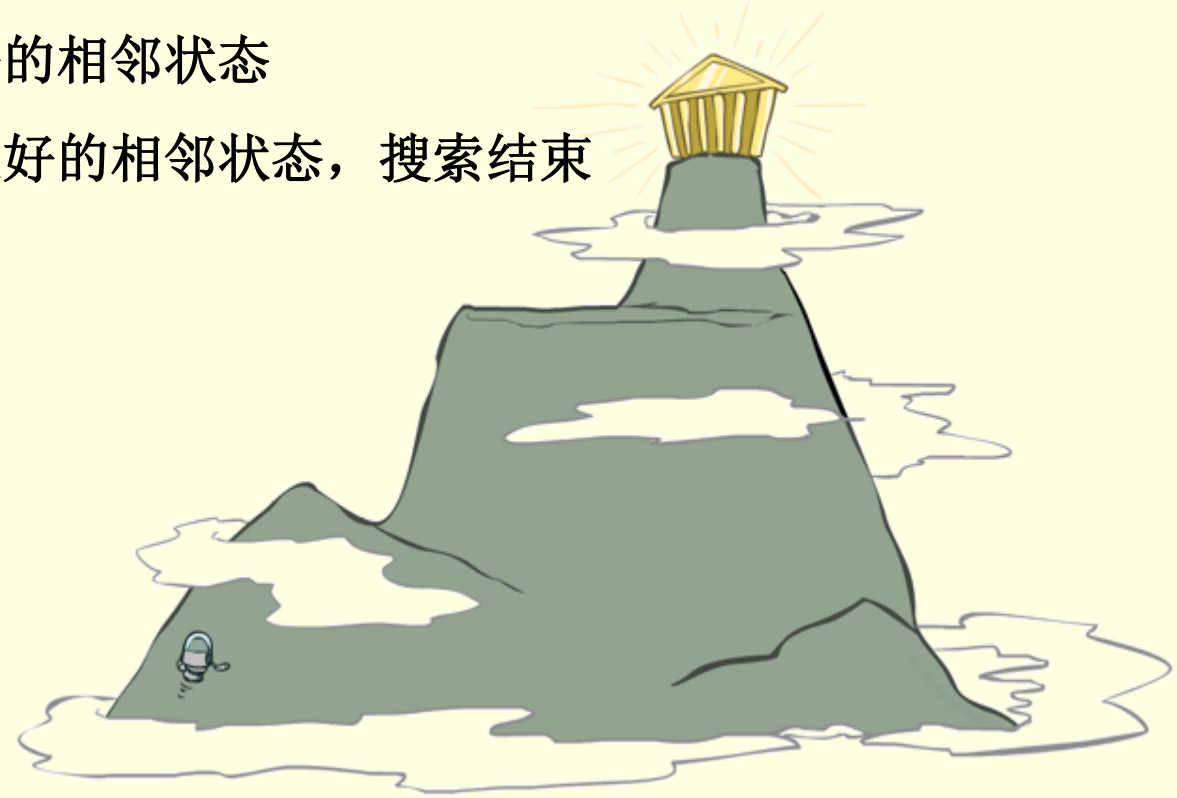
- **无信息搜索：**
  - 无先验知识，从当前节点扩展所有邻居节点
- **有信息搜索：**
  - 有先验知识，从当前状态扩展启发函数值最小邻居节点
- **局部搜索：**
  - 有先验知识，从当前状态仅扩展目标函数值更大/更小的邻居节点

局部搜索 (Local Search)

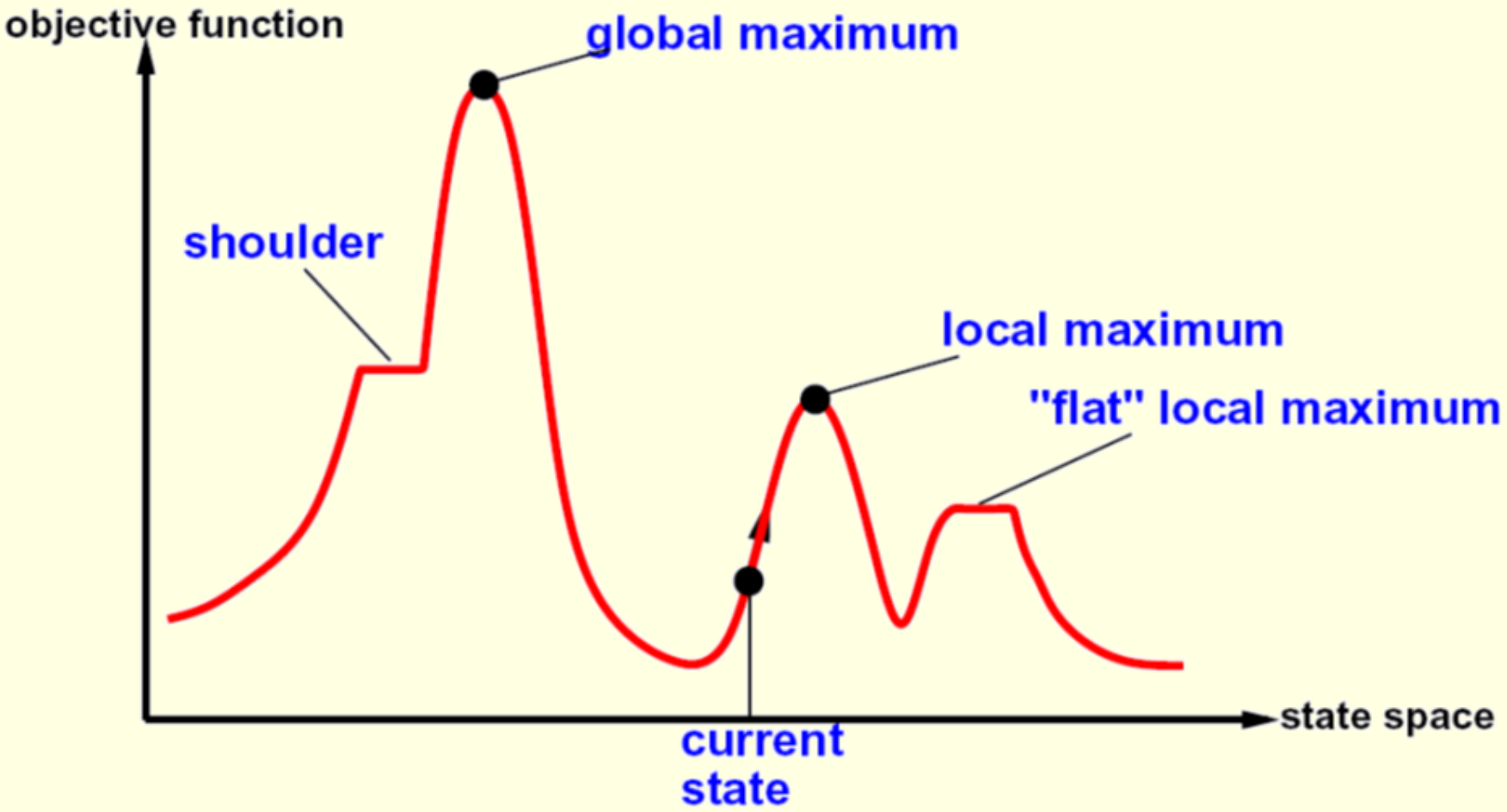
- 局部搜索
  - 爬山法
  - 禁忌搜索
  - 模拟退火算法
- 该类搜索策略特点
  - 使用与待求解问题相关的先验知识
    - 目标函数
  - 不保证完备性（若有解，不保证可以找到解）
  - 不保证最优性（若有多个解，不保证一定找到最优解）
  - 时间复杂度较低
  - 空间复杂度较低
  - 适合大规模状态空间的问题求解

爬山法

- 主要思路：
  - 可在任意位置开始
  - 循环：移动到最好的相邻状态
  - 如果没有比当前更好的相邻状态，搜索结束
- 优点：
  - 控制逻辑简单
  - 搜索效率高
  - 通用性强
- 缺点：
  - 完备性无法保证
  - 最优性无法保证

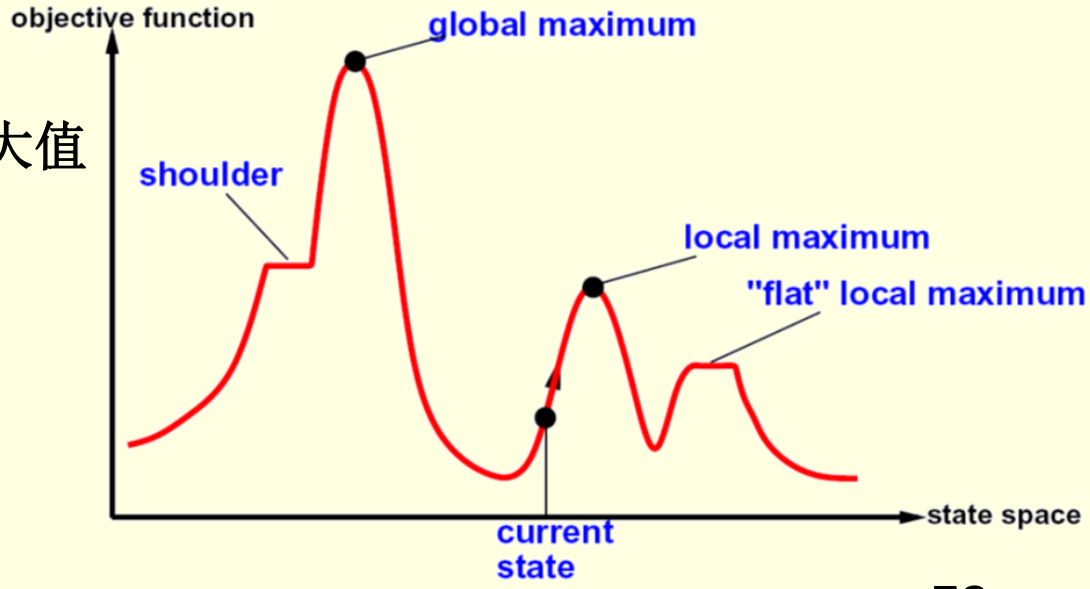


爬山法：图解


objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

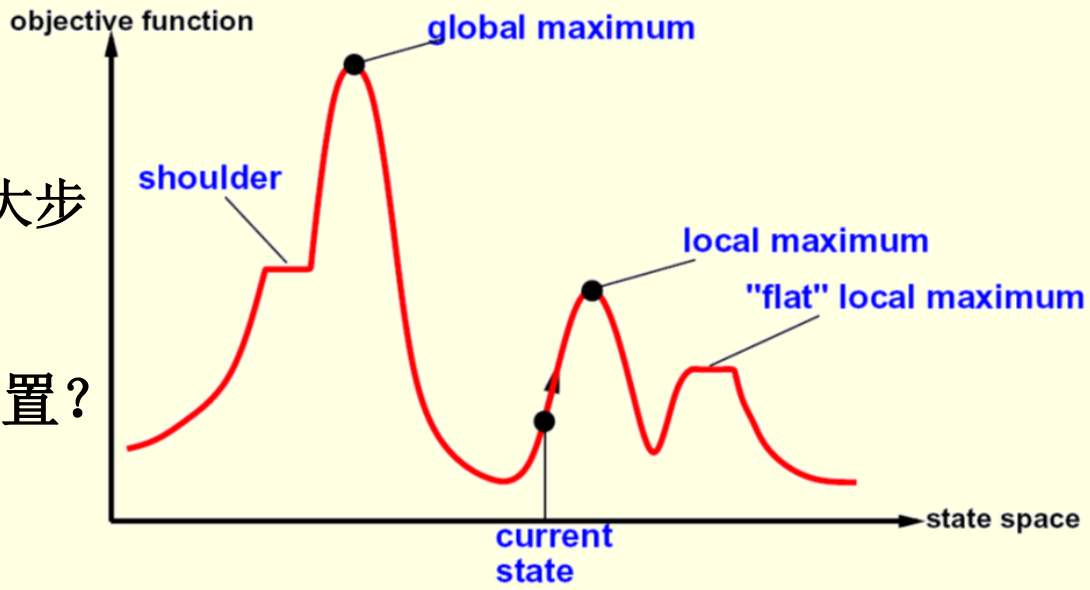
爬山法：主要缺陷

- 局部最大值：
  - 由于探索能力受限，可能被困在局部最大值
- 高原：
  - 原因与局部最大值类似
- 山脊：
  - 一系列相邻的局部最大值
- 结果不确定：
  - 起点依赖
  - 目标函数选取



爬山法：改进方案

- 多起点：
  - 选择多个不同的起始点，同时或先后进行搜索
- 回溯：
  - 允许退回一大步
- 前跨：
  - 向未探索方向前跨一大步



- 多少个起始点？如何放置？
- 回溯/前跨的步幅多大？
- 方向如何选定？

局部搜索：基本概念

- 领域：
  - 函数优化问题中：
    - “领域”通常被定义为以当前状态为中心的球体
  - 组合优化问题中：

$N: x \in D \to N(x) \in 2^D,$

且$x \in N(x)$，称为一个邻域映射，其中2^D表示D的所有子集组成的集合。

$N(x)$称为x的邻域，$y \in N(x)$称为x的一个邻居。

局部搜索：基本概念

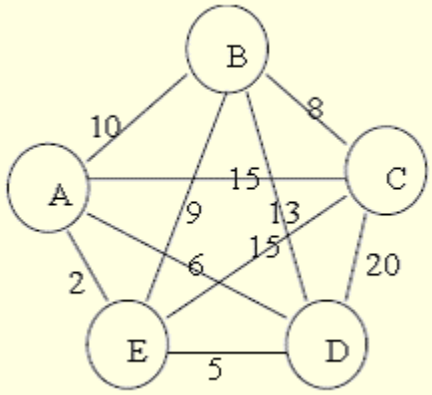
- 领域：示例
  - TSP问题解的一种表示方法为D={x=(i_1,i_2,…,i_n)| i_1,i_2,…,i_n是1,2,…,n的排列}，定义它的邻域映射为2－opt，即x中的两个元素进行对换，N(x)中共包含x的C_n^2=n(n-1)/2个邻居和x本身。
  - 例如：x=(1,2,3,4)，则C_4^2=6，N(x)={(1,2,3,4), (2,1,3,4), (3,2,1,4), (4,2,3,1), (1,3,2,4), (1,4,3,2), (1,2,4,3)}

局部搜索：基本流程

- 步骤一：
  - 选定一个初始可行解x^0，记录当前最优解$x^{best}:=x^0$, $T=N(x^{best})$；
- 步骤二：
  - 当$T \setminus \{x^{best}\}=\Phi$时，或满足其他停止运算准则时，输出计算结果，停止运算；否则，从T中选一集合S，得到S中的最好解x^{now}；若$f(x^{now}) < f(x^{best})$，则$x^{best} := x^{now}$，$T=N(x^{best})$；否则$T:=T \setminus S$；重复步骤二。

局部搜索：示例

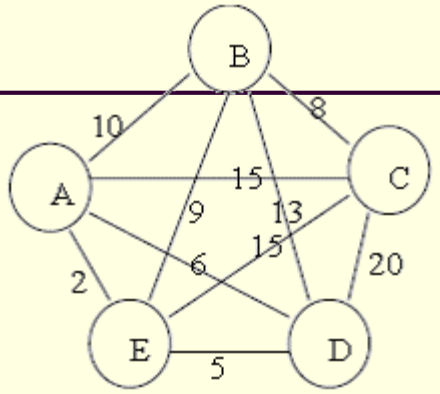
- 五城市对称TSP(Traveling Salesman Problem)问题：



$$D=(d_{ij})=\begin{bmatrix}0&10&15&6&2\\10&0&8&13&9\\15&8&0&20&15\\6&13&20&0&5\\2&9&15&5&0\end{bmatrix}$$

- **初始解为$x^{best}=(ABCDE)$，$f(x^{best})=45$，定义邻域映射为对换两个城市位置的2-opt，选定A城市为起点。**

局部搜索：示例



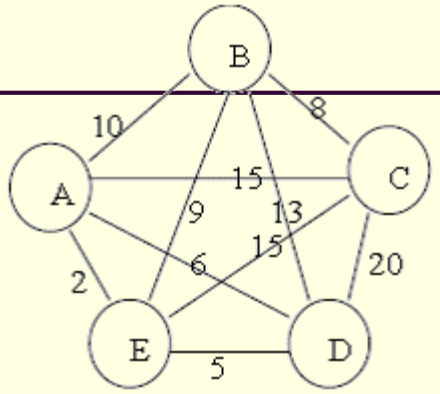
- 五城市对称TSP问题：
- **策略一：全邻域搜索**
  - **步骤一：**

$N(x^{best})=\{(ABCDE)，(ACBDE)，(ADCBE)，(AECDB)，(ABDCE)，(ABEDC)，(ABCED)\}$；

对应目标函数为$f(x)=\{45, 43, 45, 60, 60, 59, 44\}$

$x^{best}:=x^{now}=(ACBDE)$

局部搜索：示例



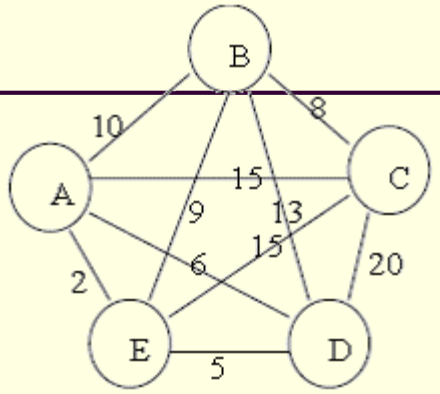
- 五城市对称TSP问题：
- **策略一：全领域搜索**
  - **步骤二：**

$N(x^{best})=\{(ACBDE)，(ABCDE)，(ADBCE)，(AEBDC)，(ACDBE)，(ACEDB)，(ACBED)\}$，

对应目标函数为$f(x)=\{43, 45, 44, 59, 59, 58, 43\}$

$x^{best}:=x^{now}=(ACBDE)$

局部搜索：示例



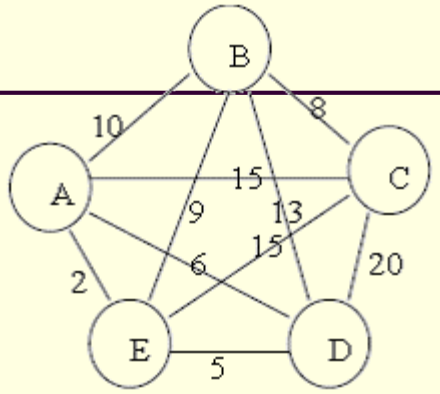
- 五城市对称TSP问题：
- **策略二：一步随机搜索**
  - **步骤一：**

从$N(x^{best})$中随机选一点，如$x^{now}=(ACBDE)$，

对应目标函数为$f(x^{now})=43<45$

$x^{best}:=x^{now}=(ACBDE)$

局部搜索：示例



- 五城市对称TSP问题：
- **策略二：一步随机搜索**
  - **步骤二：**

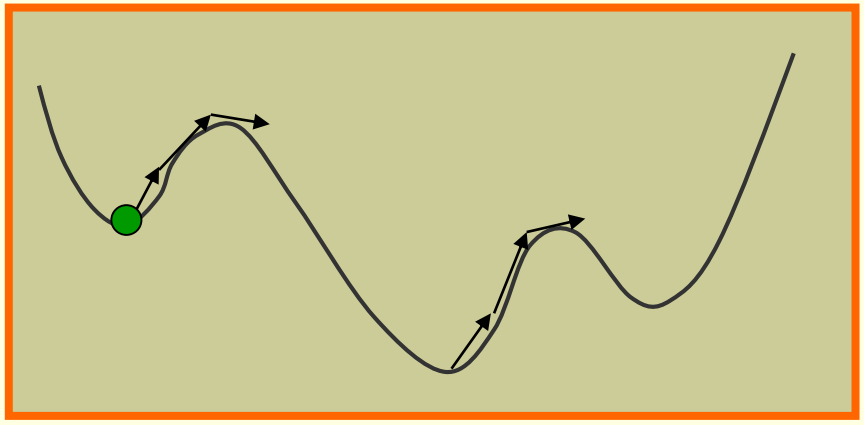
从$N(x^{best})$中再次随机选一点，如$x^{now}=(ADBCE)$，

对应目标函数为$f(x^{now})=44 > 43$

$x^{best}:=x^{now}=(ACBDE)$

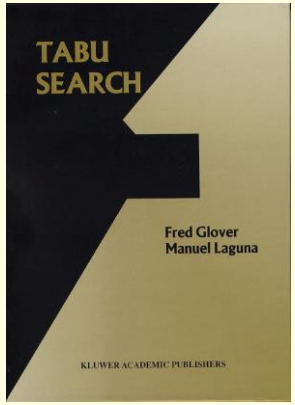
局部搜索：示例

- 五城市对称TSP问题：
  - 简单易行，但无法保证全局最优性；
  - 局部搜索主要依赖起点的选取和邻域的结构；
  - 为了得到好的解，可以比较不同的邻域结构和不同的初始点；
  - 如果初始点的选择足够多，总可以计算出全局最优解。



禁忌搜索：起源与特点

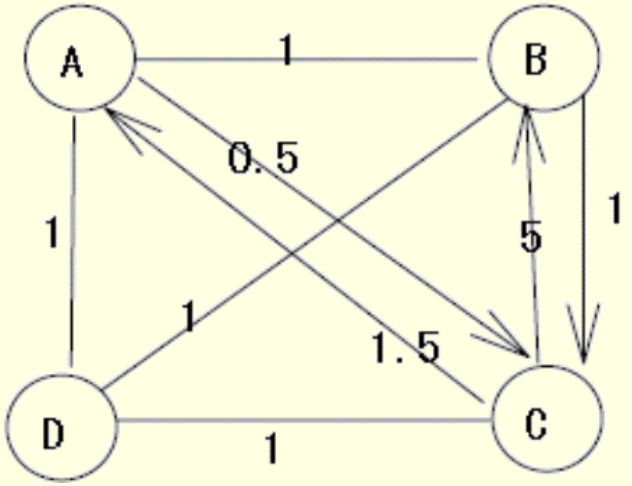
- 起源：
  - **禁忌搜索（Tabu search）是局部邻域搜索算法的推广，Fred Glover在1986年提出这个概念，进而形成一套完整算法。**
- 特点：
  - 禁忌：禁止重复前面的工作；
  - 跳出局部最优点。



http://leeds-faculty.colorado.edu/glover/

禁忌搜索：示例

- 四城市非对称TSP问题：

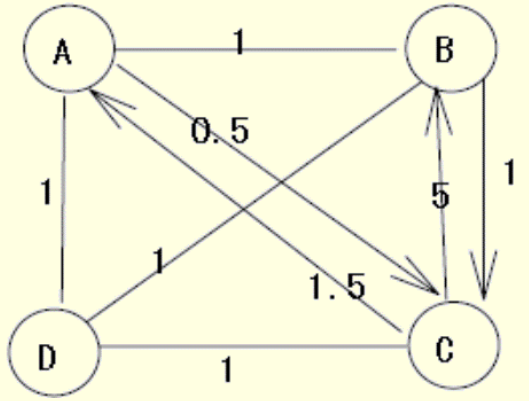


$$D=(d_{ij})=\begin{bmatrix} 0 & 1 & 0.5 & 1 \\ 1 & 0 & 1 & 1 \\ 1.5 & 5 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

- **初始解$x^0=(ABCD)$，$f(x^0)=4$，邻域映射为两个城市顺序对换的2 - opt，始、终点都是A城市，禁忌长度为3。**

禁忌搜索：示例

- 四城市非对称TSP问题：



- **第一步：**

解的形式

A	B	C	D

$f(x^0)=4$

禁忌对象及长度

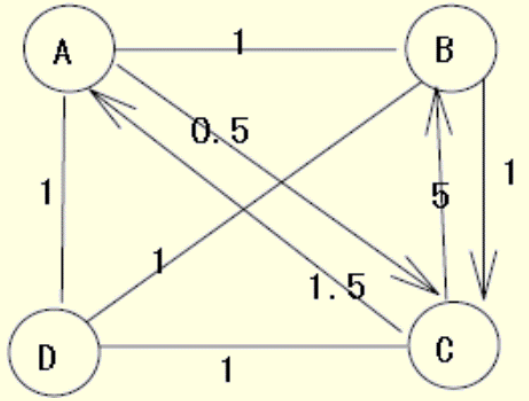
	B	C	D
A			
B			
C			

候选解

对换	评价值
CD	4.5 ☻
BC	7.5
BD	8

禁忌搜索：示例

- 四城市非对称TSP问题：



- **第二步：**

解的形式

A	B	D	C

$f(x^1)=4.5$

禁忌对象及长度

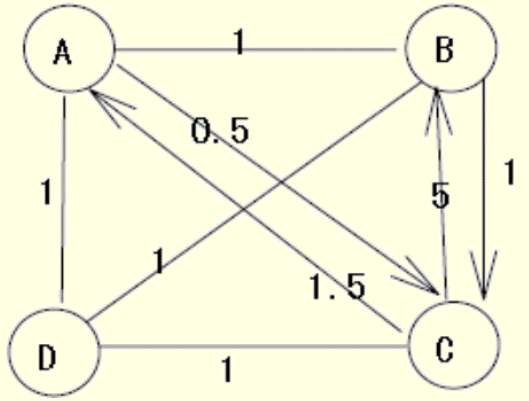
	B	C	D
A			
B			
C		3	

候选解

对换	评价值
CD	4.5 T
BC	3.5 ☻
BD	4.5

禁忌搜索：示例

- 四城市非对称TSP问题：



- **第三步：**

解的形式

A	C	D	B

$f(x^2)=3.5$

禁忌对象及长度

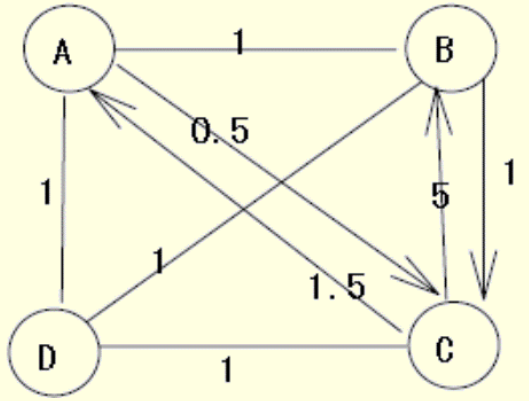
	B	C	D
A			
B	3		
C		2	

候选解

对换	评价值	
CD	8	T
BC	4.5	T
BD	7.5	☻

禁忌搜索：示例

- 四城市非对称TSP问题：



- **第四步：**

解的形式

A	C	B	D

$f(x^3)=7.5$

禁忌对象及长度

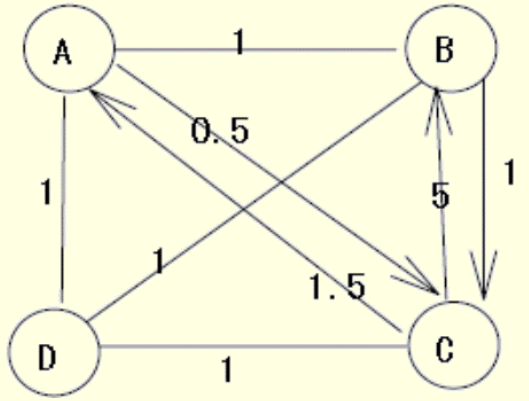
	B	C	D
A			
B	2	3	
C	1		

候选解

对换	评价值
CD	4.5 T
BC	4.5 T
BD	3.5 T

禁忌搜索：示例

- 四城市非对称TSP问题：



- **第四步：禁忌长度3 --> 2**

解的形式

A	C	B	D

$f(x^3)=7.5$

禁忌对象及长度

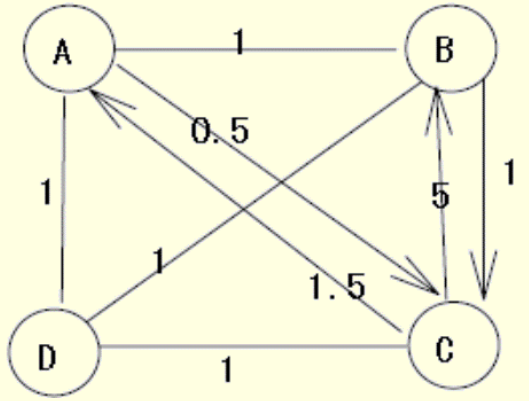
	B	C	D
A			
B	1	2	
C	0		

候选解

对换	评价值
CD	4.5 ☻
BC	4.5 T
BD	3.5 T

禁忌搜索：示例

- 四城市非对称TSP问题：



- **第五步：**

解的形式

A	D	B	C

$f(x^4)=4.5$

禁忌对象及长度

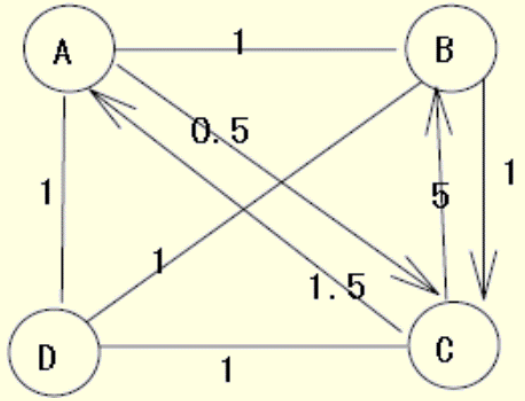
	B	C	D
A			
B	0	1	
C		2	

候选解

对换	评价值	
CD	7.5	T
BC	8	☻
BD	4.5	T

禁忌搜索：示例

- 四城市非对称TSP问题：



- **第六步：**

解的形式

A	D	C	B

$f(x^5)=8$

禁忌对象及长度

	B	C	D
A			
B	2	0	
C	1		

候选解

对换	评价值	
CD	3.5	T
BC	4.5	T
BD	4	☻

禁忌搜索：变化因素

- 禁忌表：
  - **禁忌对象：禁忌表中被禁的那些变化元素**
  - **禁忌长度：禁忌的步数**
- 状态变化：
  - **解的简单变化**
  - **解向量分量的变化**
  - **目标值变化**

禁忌搜索：变化因素

- 解的简单变化

假设$x, y \in D$，邻域映射为N，其中D为优化问题的定义域，则简单解变化

$$x \to y \in N(x)$$

是从一个解变化到另一个解。

禁忌搜索：变化因素

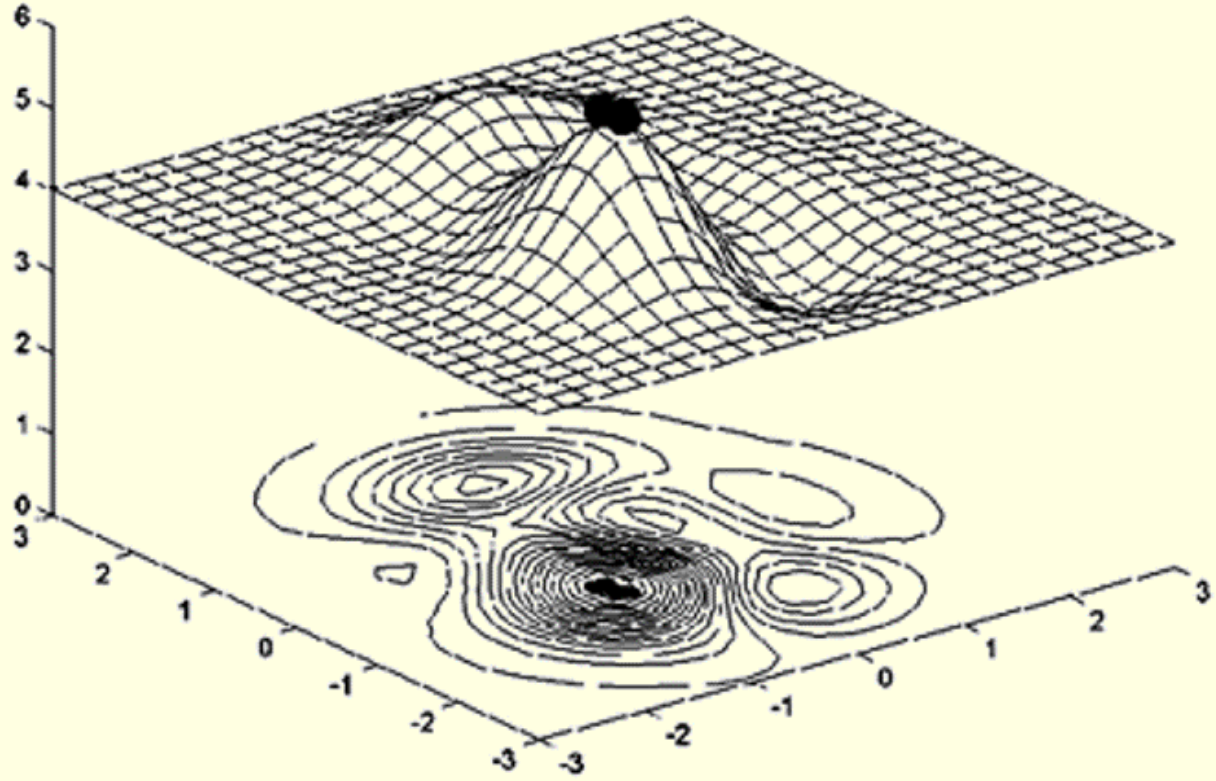
- 向量分量的变化
  - **设原有的解向量为$(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$，**
  - **向量分量的最基本变化为**

$(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) \rightarrow (x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_n)$

即只有第i个分量发生变化，也包含多个分量变化的情形。

禁忌搜索：变化因素

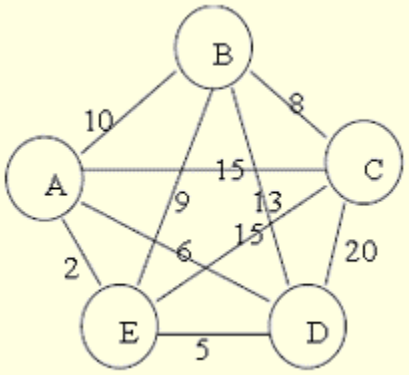
- 目标值的变化
  - **目标值的变化隐含着解集合的变化。**



禁忌搜索：禁忌表

- 禁忌对象的选择

情况1：禁忌对象为简单的解变化



$$D=(d_{ij})=\begin{bmatrix}0 & 10 & 15 & 6 & 2\\ 10 & 0 & 8 & 13 & 9\\ 15 & 8 & 0 & 20 & 15\\ 6 & 13 & 20 & 0 & 5\\ 2 & 9 & 15 & 5 & 0\end{bmatrix}$$

禁忌长度为4，从2 - opt邻域中选出最佳的5个解组成候选集Can_$N(x^{now})$，初始解$x^{now}=x^0=(ABCDE)$，$f(x^0)=45$，$H=\{(ABCDE;45)\}$。

禁忌搜索：禁忌表

- 禁忌对象的选择

情况1：禁忌对象为简单的解变化

第1步——

$x^{now}=(ABCDE)$，$f(x^{now})=45$，$H=\{(ABCDE;45)\}$

$Can\_N(x^{now})=\{(ACBDE;43)$，$(ABCDE;45)$，$(ADCBE;45)$，$(ABEDC;59)$，$(ABCED;44)\}$。

$x^{next}=(ACBDE)$

禁忌搜索：禁忌表

- 禁忌对象的选择

情况1：禁忌对象为简单的解变化

第2步——

$x^{now}=(ACBDE)$，$f(x^{now})=43$，$H=\{(ABCDE;45)$，$(ACBDE;43)\}$

$Can\_N(x^{now})=\{(ACBDE;43)$，$(ACBED;43)$，$(ADBCE;44)$，$(ABCDE;45)$，$(ACEDB;58)\}$。

$x^{next}=(ACBED)$

禁忌搜索：禁忌表

- 禁忌对象的选择

情况1：禁忌对象为简单的解变化

第3步——

$x^{now}=(ACBED)$，$f(x^{now})=43$，$H=\{(ABCDE;45)$，$(ACBDE;43)$，$(ACBED;43)\}$

$Can\_N(x^{now})=\{(ACBED;43)$，$(ACBDE;43)$，$(ABCED;44)$，$(AEBCD;45)$，$(ADBEC;58)\}$。

$x^{next}=(ABCED)$

禁忌搜索：禁忌表

- 禁忌对象的选择

情况1：禁忌对象为简单的解变化

第4步——

$x^{now}=(ABCED)$，$f(x^{now})=44$，$H=\{(ABCDE;45)$，$(ACBDE;43)$，$(ACBED;43)$，$(ABCED;44)\}$

$Can\_N(x^{now})=\{(ACBED;43)$，$(AECBD;44)$，$(ABCDE;45)$，$(ABCED;44)$，$(ABDEC;58)\}$。

$x^{next}=(AECBD)$

禁忌搜索：禁忌表

- 禁忌对象的选择

情况1：禁忌对象为简单的解变化

第5步——

$x^{now}=(AECBD)$，$f(x^{now})=44$，$H=\{(ACBDE;43)$，$(ACBED;43)$，$(ABCED;44)$，$(AECBD;44)\}$

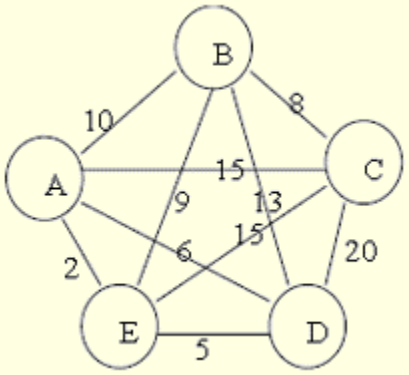
$Can\_N(x^{now})=\{(AEDBC;43)$，$(ABCED;44)$，$(AECBD;44)$，$(AECDB;44)$，$(AEBCD;45)\}$。

$x^{next}=(AEDBC)$

禁忌搜索：禁忌表

- 禁忌对象的选择

情况2：禁忌对象为分量变化



$$D=(d_{ij})=\begin{bmatrix}0 & 10 & 15 & 6 & 2\\ 10 & 0 & 8 & 13 & 9\\ 15 & 8 & 0 & 20 & 15\\ 6 & 13 & 20 & 0 & 5\\ 2 & 9 & 15 & 5 & 0\end{bmatrix}$$

禁忌长度为3，从2 - opt邻域中选出最佳的5个解组成候选集Can_$N(x^{now})$，初始解$x^{now}=x^0=(ABCDE)$，$f(x^0)=45$。

禁忌搜索：禁忌表

- 禁忌对象的选择

情况2：禁忌对象为分量变化

第1步——

$x^{now}=(ABCDE)$，$f(x^{now})=45$，$H=\Phi$

$Can\_N(x^{now})=\{(ACBDE;43)$，$(ADCBE;45)$，$(AECDB;60)$，$(ABEDC;59)$，$(ABCED;44)\}$。

$x^{next}=(ACBDE)$

禁忌搜索：禁忌表

- 禁忌对象的选择

情况2：禁忌对象为分量变化

第2步——

$x^{now}=(ACBDE)$，$f(x^{now})=43$，$H=\{(B,C)\}$

Can_$N(x^{now})=\{(ACBED;43)$，$(ADBCE;44)$，$(ABCDE;45)$，$(ACEDB;58)$，$(AEBDC;59)\}$。

$x^{next}=(ACBED)$

禁忌搜索：禁忌表

- 禁忌对象的选择

情况2：禁忌对象为分量变化

第3步——

$x^{now}=(ACBED)$，$f(x^{now})=43$，$H=\{(B,C)，(D,E)\}$

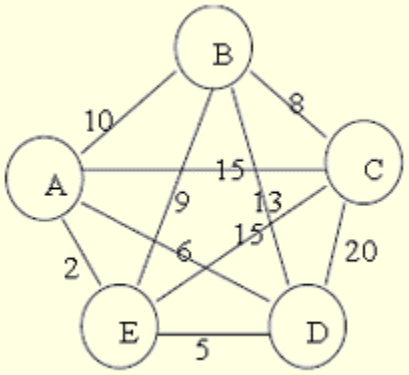
$Can\_N(x^{now})=\{(ACBDE;43)，(ABCED;44)，(AEBCD;45)$，$(ADBEC;58)$，$(ACEBD;58)\}$。

$x^{next}=(AEBCD)$

禁忌搜索：禁忌表

- 禁忌对象的选择

情况3：禁忌对象为目标值变化



$$D=(d_{ij})=\begin{bmatrix}0 & 10 & 15 & 6 & 2\\ 10 & 0 & 8 & 13 & 9\\ 15 & 8 & 0 & 20 & 15\\ 6 & 13 & 20 & 0 & 5\\ 2 & 9 & 15 & 5 & 0\end{bmatrix}$$

禁忌长度为3，从2 - opt邻域中选出最佳的5个解组成候选集Can_$N(x^{now})$，初始解$x^{now}=x^0=(ABCDE)$，$f(x^0)=45$。

禁忌搜索：禁忌表

- 禁忌对象的选择

情况3：禁忌对象为目标值变化

第1步——

$x^{now}=(ABCDE)$，$f(x^{now})=45$，$H=\{45\}$

$\text{Can\_}N(x^{now})=\{(ABCDE;45)$，$(ACBDE;43)$，$(ADCBE;45)$，$(ABEDC;59)$，$(ABCED;44)\}$。

$x^{next}=(ACBDE)$

禁忌搜索：禁忌表

- 禁忌对象的选择

情况3：禁忌对象为目标值变化

第2步——

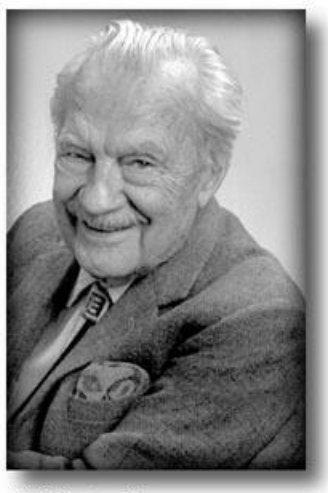
$x^{now}=(ACBDE)$，$f(x^{now})=43$，$H=\{45，43\}$

$Can\_N(x^{now})=\{(ACBDE;43)，(ACBED;43)，(ADBCE;44)，(ABCDE;45)，(ACEDB;58)\}$。

$x^{next}=(ADBCE)$

模拟退火算法：起源与特点

- 起源：
  - **最早的思想由Metropolis等（1953）提出；**
  - **1983年Kirkpatrick等将其应用于组合优化。**
- 特点：
  - 克服陷入局部最小值；
  - 克服初值依赖性。



Nick Metropolis

模拟退火算法：物理退火过程

- 什么是退火：
  - 退火是指将固体加热到足够高的温度，使分子呈随机排列状态，然后逐步降温使之冷却，最后分子以低能状态排列，固体达到某种稳定状态。

模拟退火算法：物理退火过程

- 加温过程：
  - 增强粒子的热运动，消除系统原先可能存在的非均匀态；
- 等温过程：
  - 对于与环境换热而温度不变的封闭系统，系统状态的自发变化总是朝自由能减少的方向进行，当自由能达到最小时，系统达到平衡态；
- 冷却过程：
  - 使粒子热运动减弱并渐趋有序，系统能量逐渐下降，从而得到低能的晶体结构。

模拟退火算法：借鉴物理退火过程

- 目标相似：
  - 模仿自然界退火現象而得，利用了物理中固体物质的退火过程与一般优化问题的相似性；
- 过程相似：
  - 从某一初始温度开始，伴随温度的不断下降，结合概率突跳特性在解空间中随机寻找全局最优解。

模拟退火算法：形式化表达

- 数学表述：
  - **在温度T，分子停留在状态r满足Boltzmann概率分布**

$$P\{\overline{E} = E(r)\} = \frac{1}{Z(T)} \exp\left(-\frac{E(r)}{k_B T}\right)$$

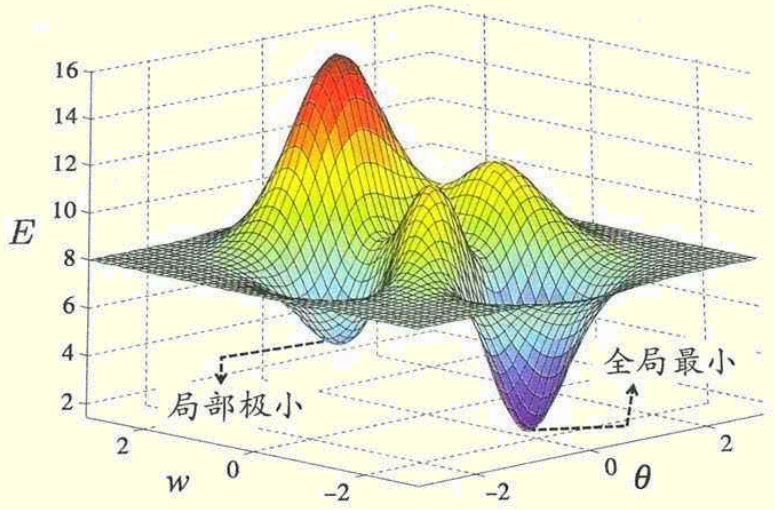
$\overline{E}$表示分子能量的一个随机变量，$E(r)$表示状态r的能量，$k_B > 0$为Boltzmann常数。$Z(T)$为概率分布的标准化因子：

$$Z(T) = \sum_{s \in D} \exp\left(-\frac{E(s)}{k_B T}\right)$$

模拟退火算法：形式化表达

- 数学表述：
  - 在**同一个温度T**，选定两个能量$E_1<E_2$，有：

$$P\{\overline{E}=E_1\}-P\{\overline{E}=E_2\}=\frac{1}{Z(T)}\exp\left(-\frac{E_1}{k_BT}\right)\left[1-\exp\left(-\frac{E_2-E_1}{k_BT}\right)\right]$$



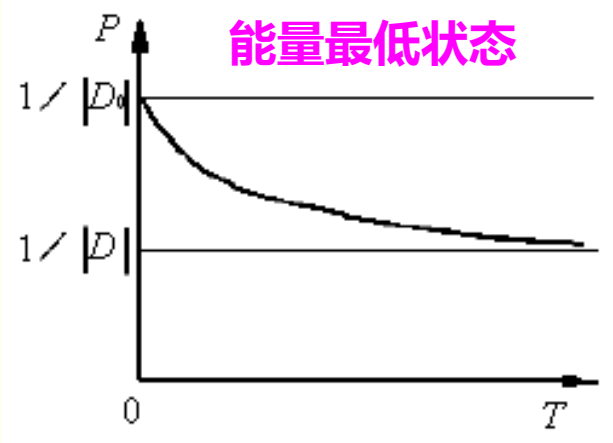
模拟退火算法基本思想：在一定温度下，搜索从一个状态随机地变化到另一个状态；随着温度的不断下降直到最低温度，搜索过程以概率1停留在最优解

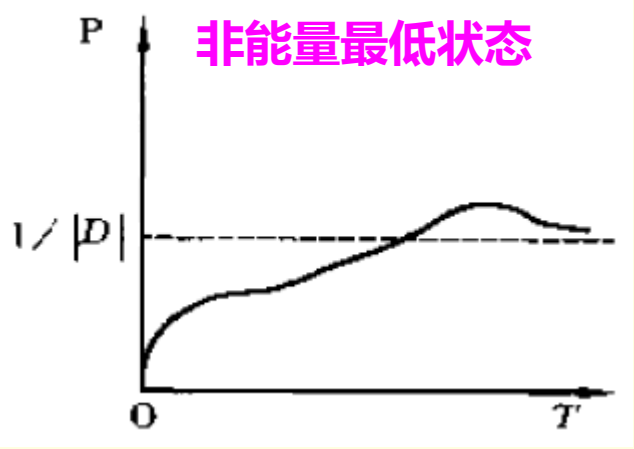
模拟退火算法：通俗解释

- **Boltzman概率分布：**
  - 在同一个温度，分子停留在能量小状态的概率大于停留在能量大状态的概率；
  - 温度越高，不同能量状态对应的概率相差越小；温度足够高时，各状态对应概率基本相同；
  - 随着温度的下降，能量最低状态对应概率越来越大；温度趋于0时，其状态趋于1。

模拟退火算法：通俗解释

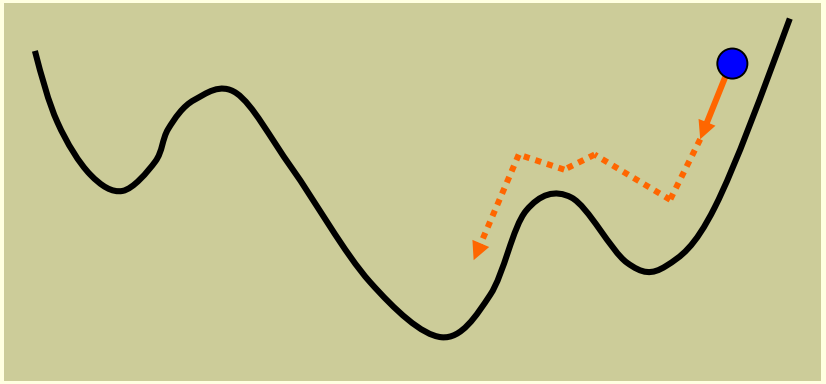
- **若|D|为状态空间D中状态的个数，D_0是具有最低能量的状态集合：**
  - 当温度很高时，每个状态概率基本相同，接近平均值1/|D|；
  - 状态空间存在超过两个不同能量时，具有最低能量状态的概率超出平均值1/|D| ；
  - 当温度趋于0时，分子停留在最低能量状态的概率趋于1。





模拟退火算法：通俗解释

- **Metropolis准则（1953）：**
  - 固体在恒定温度下达到热平衡的过程可以用Monte Carlo方法（计算机随机模拟方法）加以模拟，虽然该方法简单，但必须大量采样才能得到比较精确的结果，计算量很大；
  - 以概率接受新状态
    - 在**高温**下，**可接受**与当前状态**能量差较大的新状态**；
    - 在**低温**下，**只接受**与当前状态**能量差较小的新状态**。



模拟退火算法：基本步骤

给定初温$t=t_0$，随机产生初始状态$s=s_0$，令$k=0$；

Repeat

　Repeat

　　产生新状态s_j=Generate(s)；

　　if min{1,exp[-($C(s_j)$-$C(s)$)/t_k]}>=random[0,1]　$s=s_j$;

　Until 抽样稳定准则满足；

　退温t_{k+1}=update(t_k)并令$k=k+1$；

Until 算法终止准则满足；

输出算法搜索结果。

三函数

两准则

初始温度

模拟退火算法：关键因素

- **状态产生函数：**
  - 原则
    - 产生的候选解应遍布全部解空间
  - 方法
    - 在当前状态的邻域结构内以一定概率方式产生
      - 均匀分布
      - 正态分布
      - 指数分布
      - ……

模拟退火算法：关键因素

- **状态接受函数：**
  - 原则
    - 在固定温度下，接受使目标函数下降的候选解的概率要大于使目标函数上升的候选解概率；
    - 随温度的下降，接受使目标函数上升的解的概率要逐渐减小；
    - 当温度趋于零时，只能接受目标函数下降的解。
  - 方法
    - 一般采用**min[1,exp(-ΔC/t)]**

模拟退火算法：关键因素

- **初始温度：**
  - 收敛性
    - 通过理论分析可以得到初温的解析式，但解决实际问题时难以得到精确的参数；初温应充分大。
  - 实践经验
    - 初温越大，获得高质量解的机率越大，但花费较多的计算时间
  - 方法
    - 均匀抽样一组状态，以各状态目标值得方差为初温；
    - 随机产生一组状态，确定两两状态间的最大目标值差，根据差值，利用一定的函数确定初温；
    - 利用经验公式。

模拟退火算法：关键因素

- **温度更新函数：**
  - $t_{k+1} = \alpha t_k,\ k \geq 0,\ 0 < \alpha < 1$ ，α越接近1温度下降越慢，且其大小可以不断变化；
  - $t_k = \frac{K-k}{K} t_0$ ，其中t_0为起始温度，K为算法温度下降的总次数。

模拟退火算法：关键因素

- **内循环终止准则：**
  - 常用的Metropolis抽样稳定准则
    - **检验目标函数的均值是否稳定；**
    - **连续若干步的目标值变化较小；**
    - **按一定的步数抽样。**
- **外循环终止准则：**
  - **设置终止温度的阈值；**
  - **设置外循环迭代次数；**
  - **算法搜索到的最优值连续若干步保持不变；**
  - **概率分析方法。**

模拟退火算法：优缺点

- **优点：**
  - 大概率找到全局最优解；
  - 初值鲁棒性强；
  - 简单、通用、易实现。
- **缺点：**
  - 优化过程较长
    - 较高的初始温度
    - 较慢的降温速率
    - 较低的终止温度
    - 同一温度下的大量采样

问题规约：起源与特点

- 起源：
  - **最早的思想由？？？提出；**
- 特点：
  - 将初始问题依照规则变换成若干子问题；
  - 可直接求解的问题称为本原问题。

问题规约：关键要素（符号积分问题）

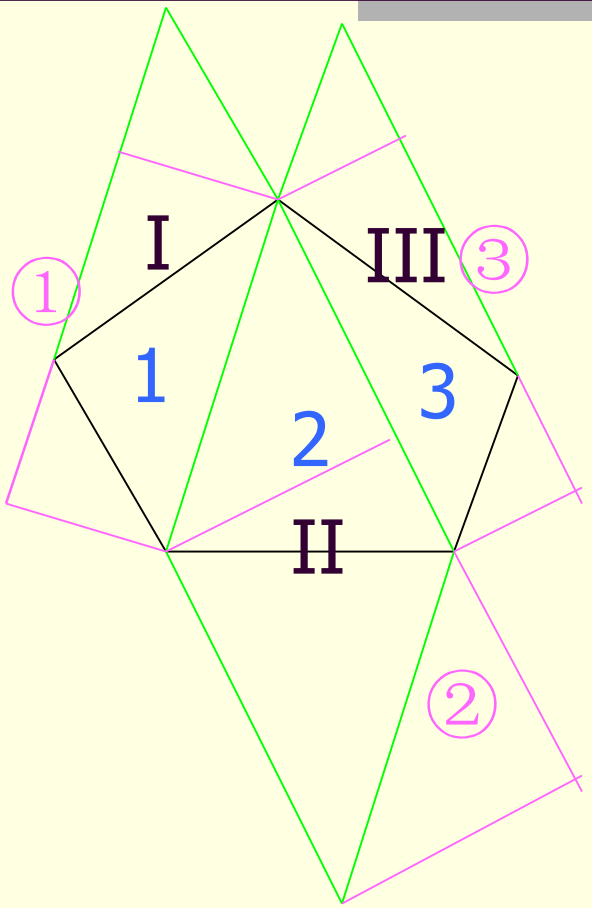
- 初始问题描述：
  - $\int f(x)dx$
- 变换规则：
  - 积分规则
- 本原问题描述
  - 可直接求原函数的积分：
    - $\int sin(x)dx$
    - $\int e^x dx$

问题规约：问题分解

- 三种分解可能：
  - And
  - Or
  - And 和 Or
- 问题分解过程：
  - 从初始问题出发，依据变换规则以迭代的方式逐层构建子问题，直至把初始问题规约（化简）为一个本原问题的集合。

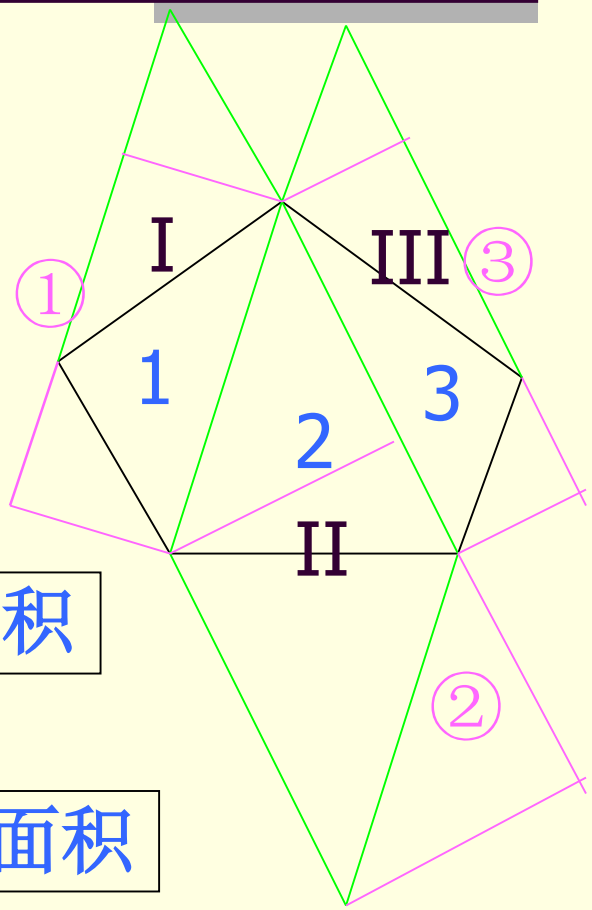
问题规约：示例1

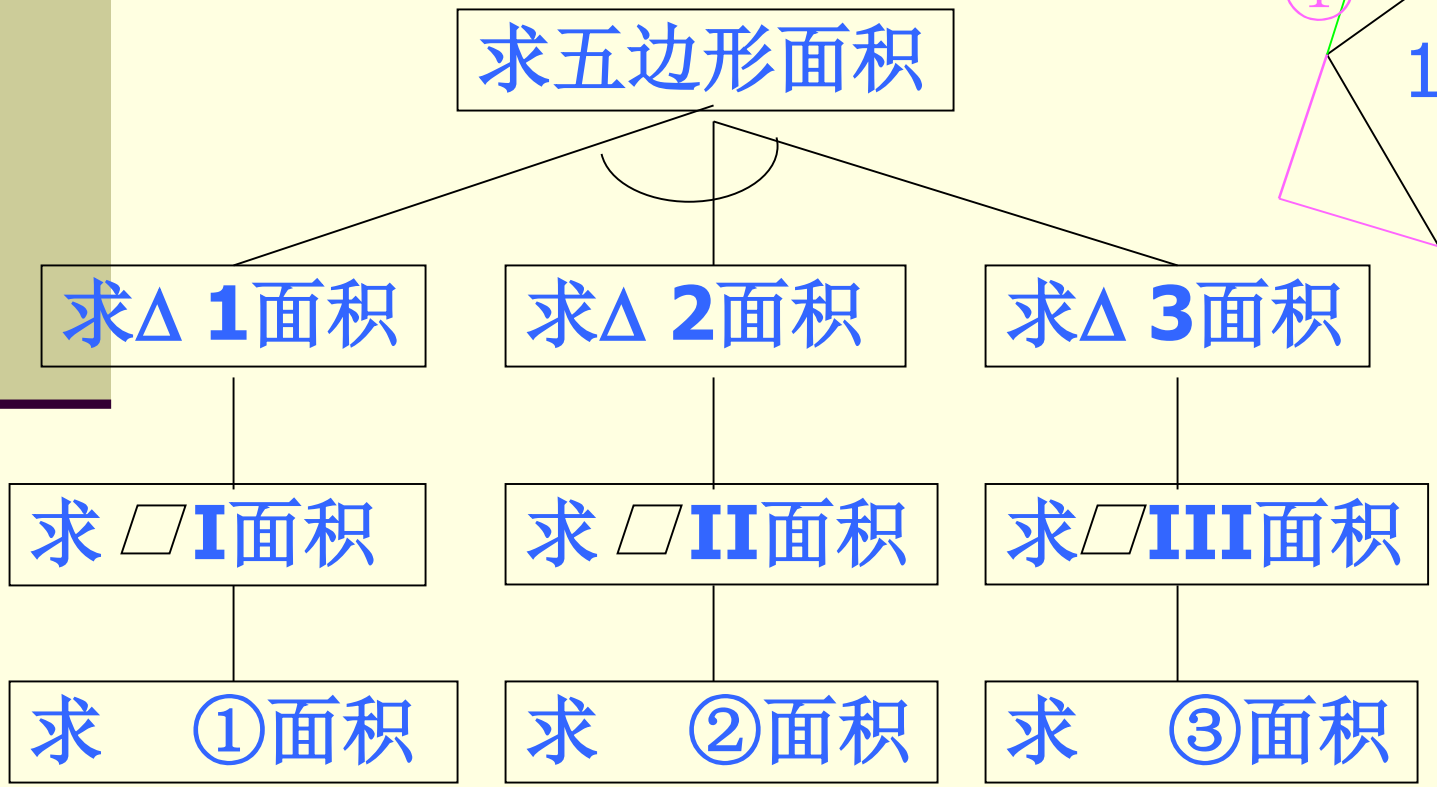
- 给定：
  - 如右图所示的图形
- 假定：
  - 我们已经会求矩形的面积
- 目标：
  - 计算右图所示图形的面积



问题规约：示例1（续上）

- 求解步骤：

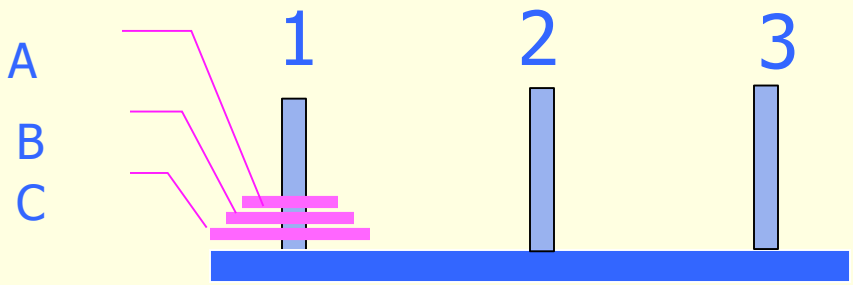




问题规约：示例2

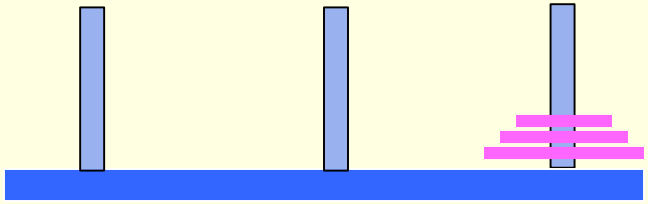
- 汉诺塔问题的两种解法
  - 状态空间法
    - 初始状态：

      (1 1 1) 表示C, B, A三个盘都在柱1上



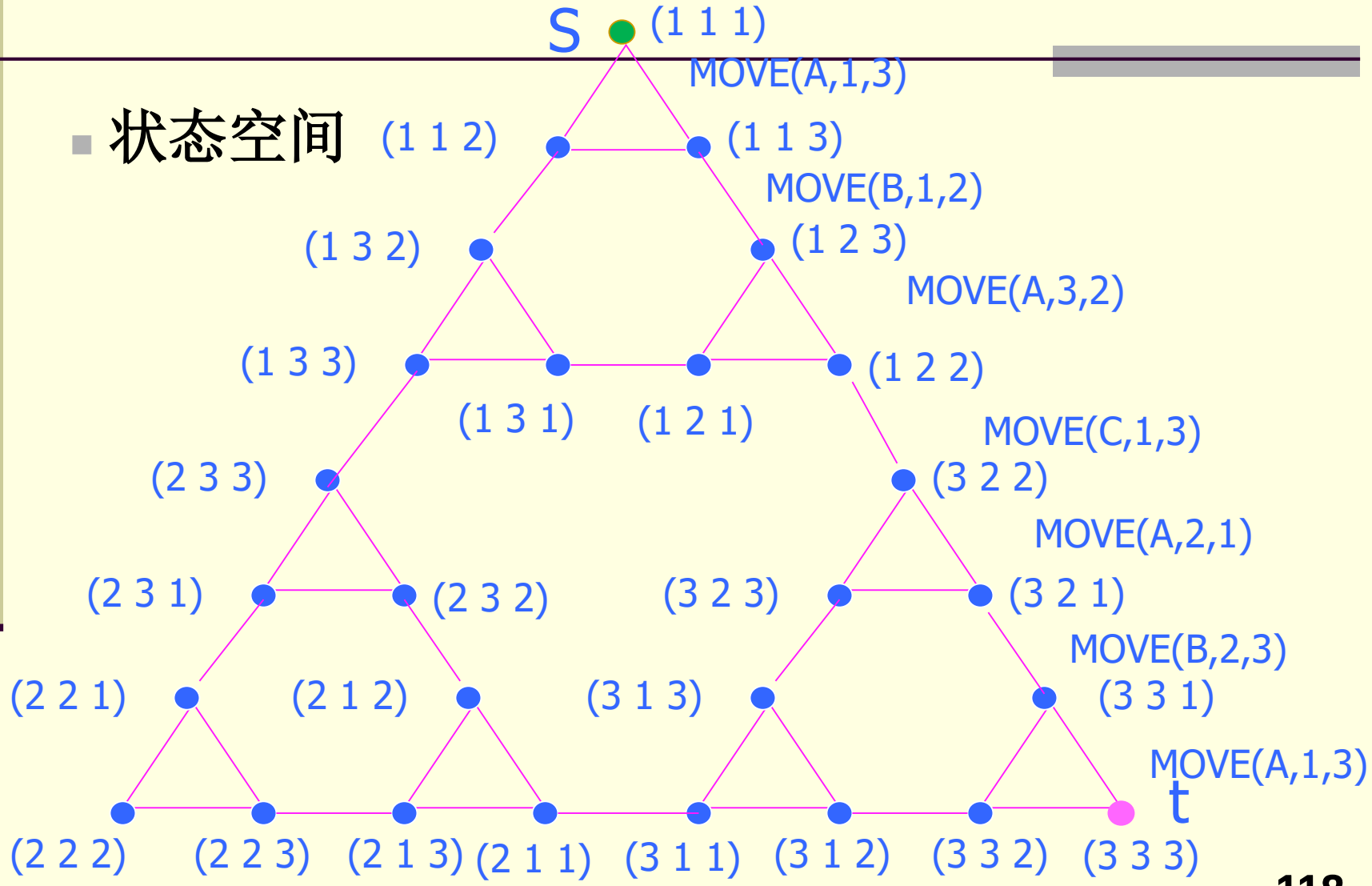
    - 目标状态

      (3 3 3) 表示C, B,A三个盘都在柱3上



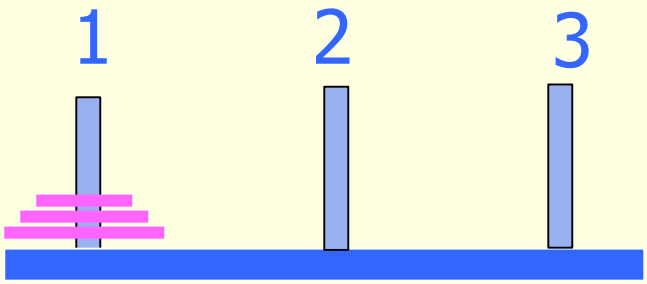
问题规约：示例2（续上）

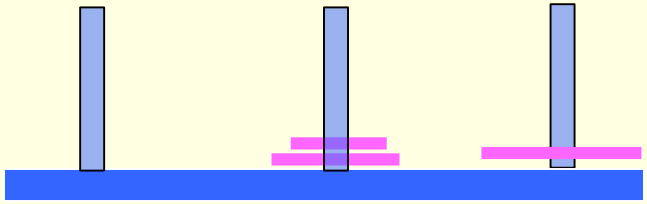
- 状态空间



问题规约：示例2（续上）

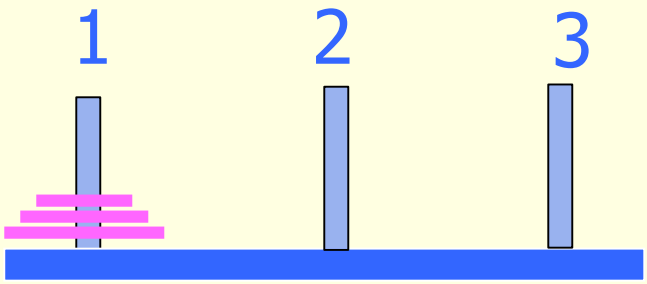
- 采用规约法
  - 要把所有圆盘移至柱3，必须先把C盘移至柱3，而在移动C盘至柱3前，柱3必须为空。
  - 只有把A、B移至柱2后，才能将C移到柱3。
  - 在C移至柱3后，再解决将A、B移至柱3。

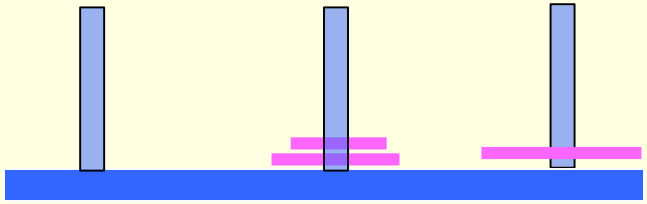




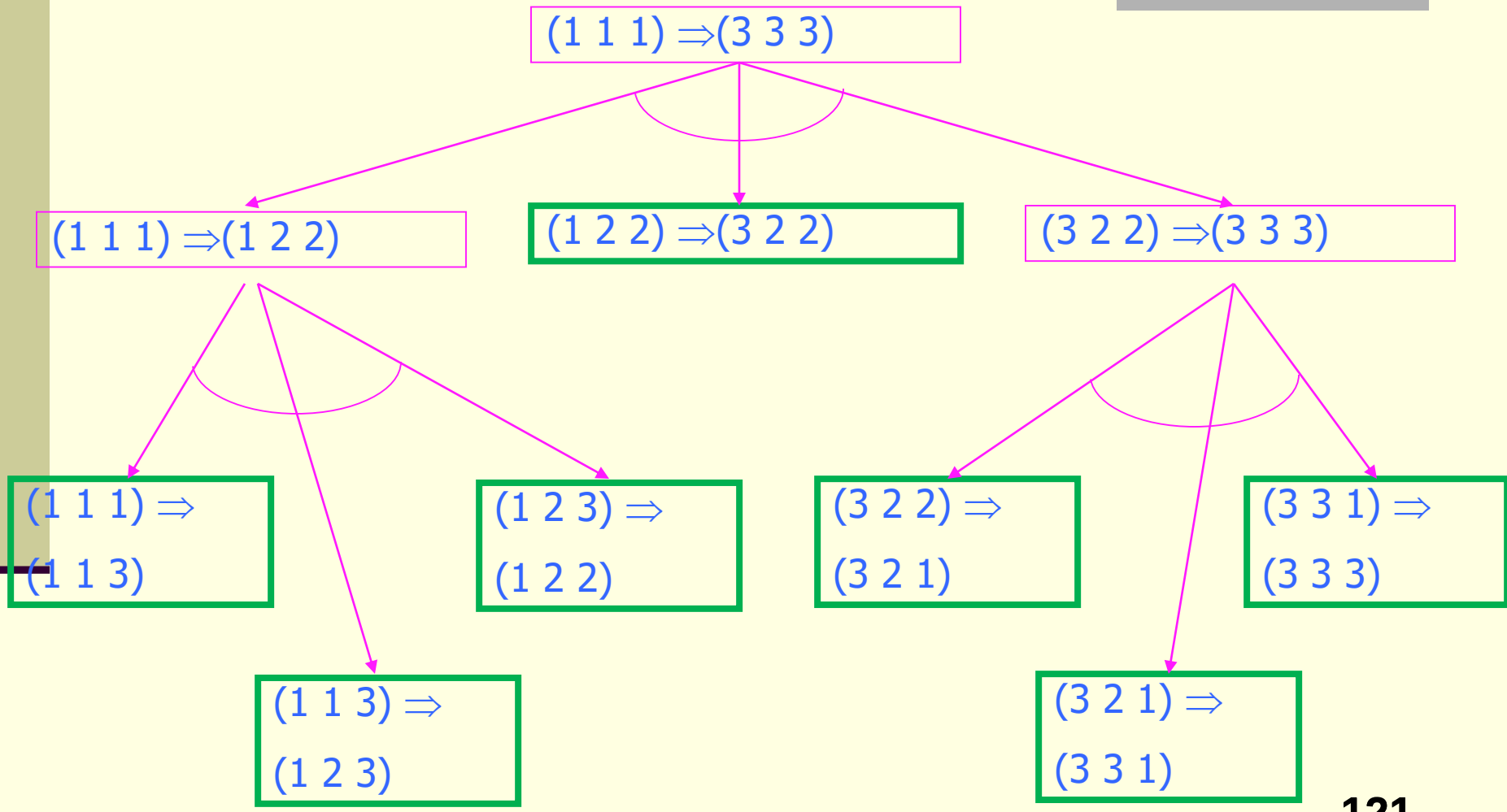
问题规约：示例2（续上）

- 初始问题分解：
  - 移动A、B至柱2的双圆盘问题；
  - 移动C至柱3的单元盘问题；（本原问题）
  - 移动A、B至柱3的双圆盘问题。





问题规约：示例2（续上）

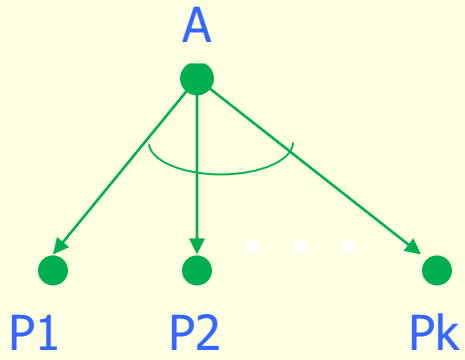

(1 1 1) ⇒(3 3 3)
(1 1 1) ⇒(1 2 2)
(1 2 2) ⇒(3 2 2)
(3 2 2) ⇒(3 3 3)
(1 1 1) ⇒
(1 1 3)
(1 2 3) ⇒
(1 2 2)
(3 2 2) ⇒
(3 2 1)
(3 3 1) ⇒
(3 3 3)
(1 1 3) ⇒
(1 2 3)
(3 2 1) ⇒
(3 3 1)

问题规约：小结

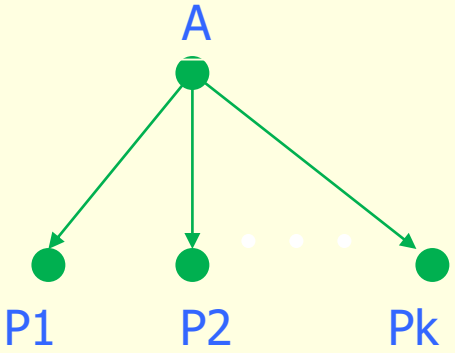
- 状态空间法与问题归约法的比较：
  - 状态空间——问题空间
  - 操作——归约
  - 求解路径——本原问题
- 归约就是化简，即把复杂问题分解为若干子问题，且使得：
    - 每个子问题比原问题好解；
    - 这些子问题解决了，原问题就解决了。
- 归约法的分类
  - 有序归约(分段归约)　汉诺塔问题
  - 无序归约(分解归约)　求五边形的面积

与或图：表示法

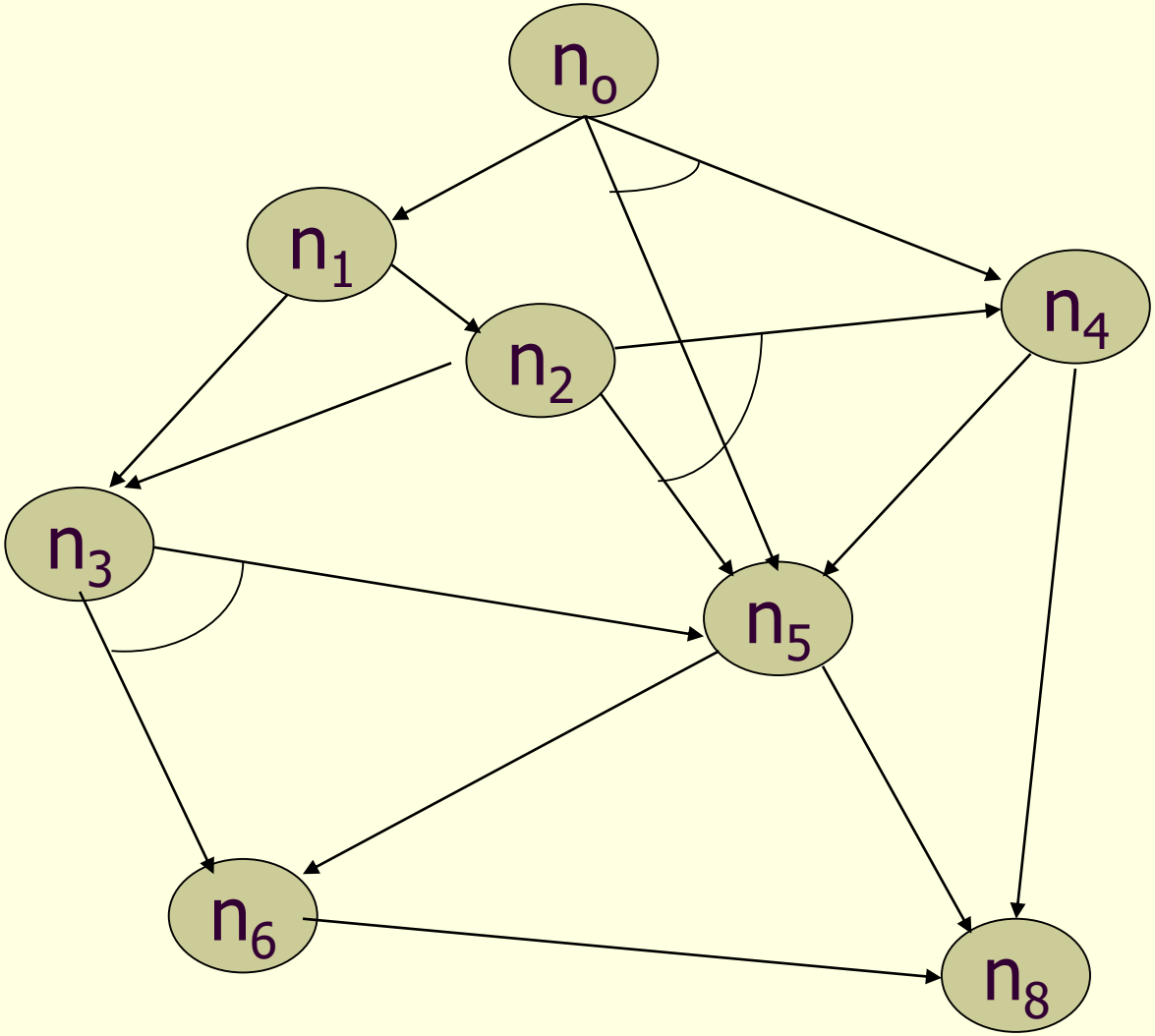
- 与扩展：
  - 将一个问题分解为若干子问题， 所有的子问题有解，原问题才有解
  - K-联接符



- 或扩展
  - 将一个问题转化为若干子问题，只要一个子问题有解，原问题就有解
  - 单线联接符



与或图：表示法（示例）



与或图：搜索

- 主要思路：
  - 从代表原始问题的根节点开始，按一定的规则(归约操作)进行与或扩展，直到代表本原问题的终节点；
  - 要选择适当的或分枝进行扩展，以求找到一个最佳分解方案；
  - 在与或图中搜索最佳分解方案，即在问题空间搜索一个最佳解图。
- 重要概念：
  - 根节点 / 终结点 / 叶节点
  - 可解节点 / 不可解节点
  - 解图 / 耗散值 / 最佳解图

与或图：重要概念

- 根节点 / 终节点 / 叶节点：
  - 根节点：无父节点的节点，初始问题
  - 终节点：可联合表示目标状态的节点，本原问题
  - 叶节点：无子节点的节点
- 可解节点：
  - 终节点是可解节点
  - 若n为一非终叶节点，且含有“或”后继节点，则只有当后继节点中至少有一个是可解节点时，n才可解
  - 若*n*为非终叶节点，且含“与”后继节点，则只有当后继节点全部可解时，*n*才可解

与或图：重要概念

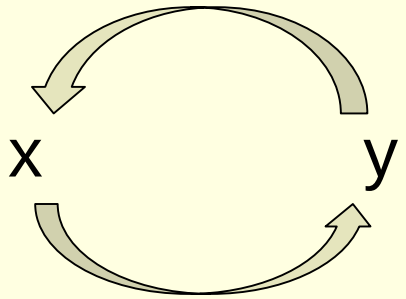
- 不可解节点：
  - 没有后继节点的非终叶节点为不可解
  - 若*n*为一非叶节点含有“或”后继节点，则仅当全部后继节点为不可解时，*n*不可解
  - 若*n*为一非叶节点含有“与”后继节点，则只要有一个后继节点为不可解时，*n*为不可解

与或图：重要概念

- 解图：
  - 自根节点开始选一外向连接，并从该连接指向的每个子节点出发，再选一外向连接，如此反复进行，直到所有外向连接都指向终节点为止
    - 解图纯粹是一种“与”图
    - 由于与或图中存在“或”关系；可产生或搜索到多个解图
    - 解图应无环，即任何节点的外向连接均不得指向自己或自己的先辈，否则会使搜索陷入死循环
    - 实际情况中，我们往往也假设与或图是无环的

与或图：重要概念

- 与/或图搜索仅对不含回路的图进行操作
  - 例如：



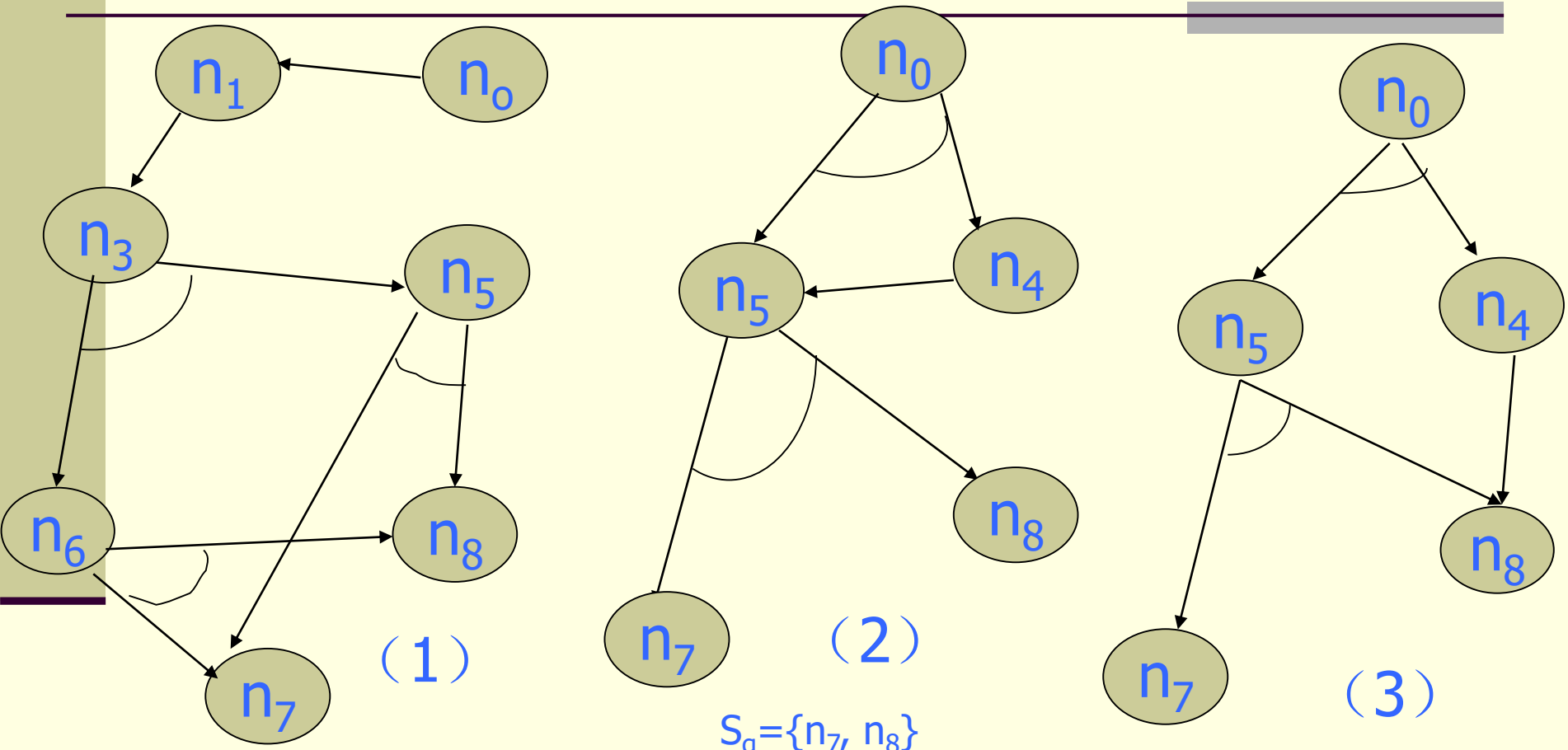
表示求了x就可以求y.，求了y就可以求x，两者都不可能求解。

与或图：耗散值计算规则

- 设从当前节点n到终节点集S_g费用估计为h’(n)，可按照以下步骤以递归的方式进行计算：
  - 若$n \in S_g$, 则h’(n)=0;
  - 若n有一组由“与”弧连接的后继节点{n_i} 则:

    h’(n)=c_1+c_2…+c_i+…+h’(n_1)+h’(n_2)+…+h’(n_i)
  - 若n既有“与”又有“或”弧，则“与”弧算作一个“或”后继，再取各or弧后继中费用最小者为n的费用。

与或图：重要概念



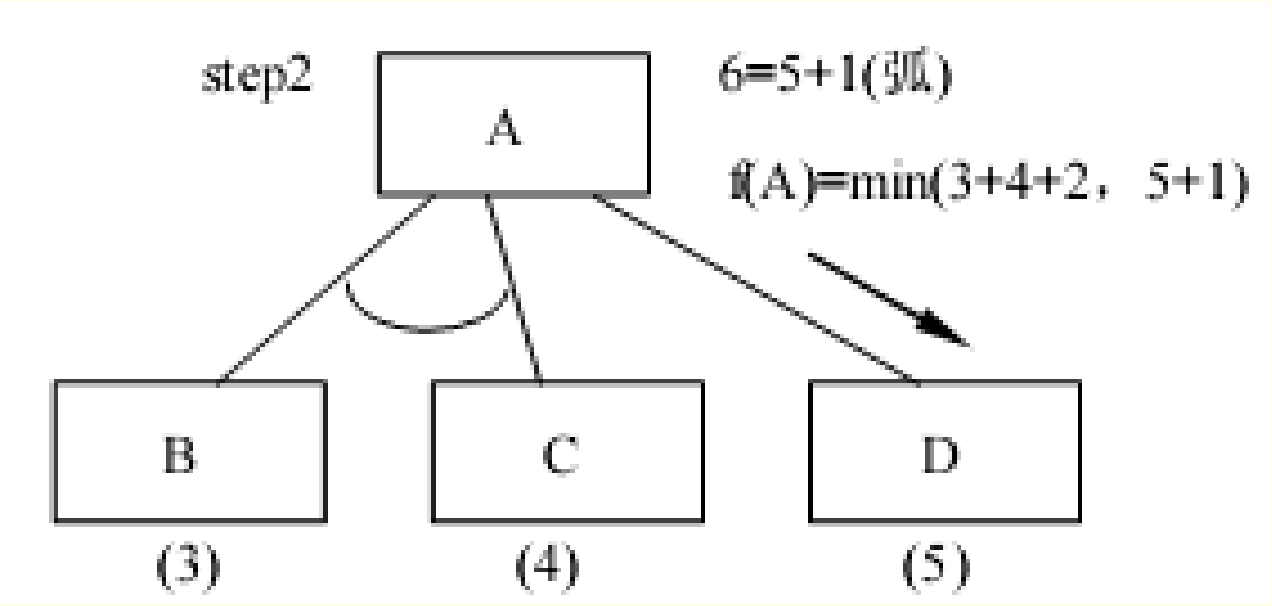
具有最小耗散值的解图称为最佳解图

与或图：搜索

- 第一步：A是根节点（初始问题）

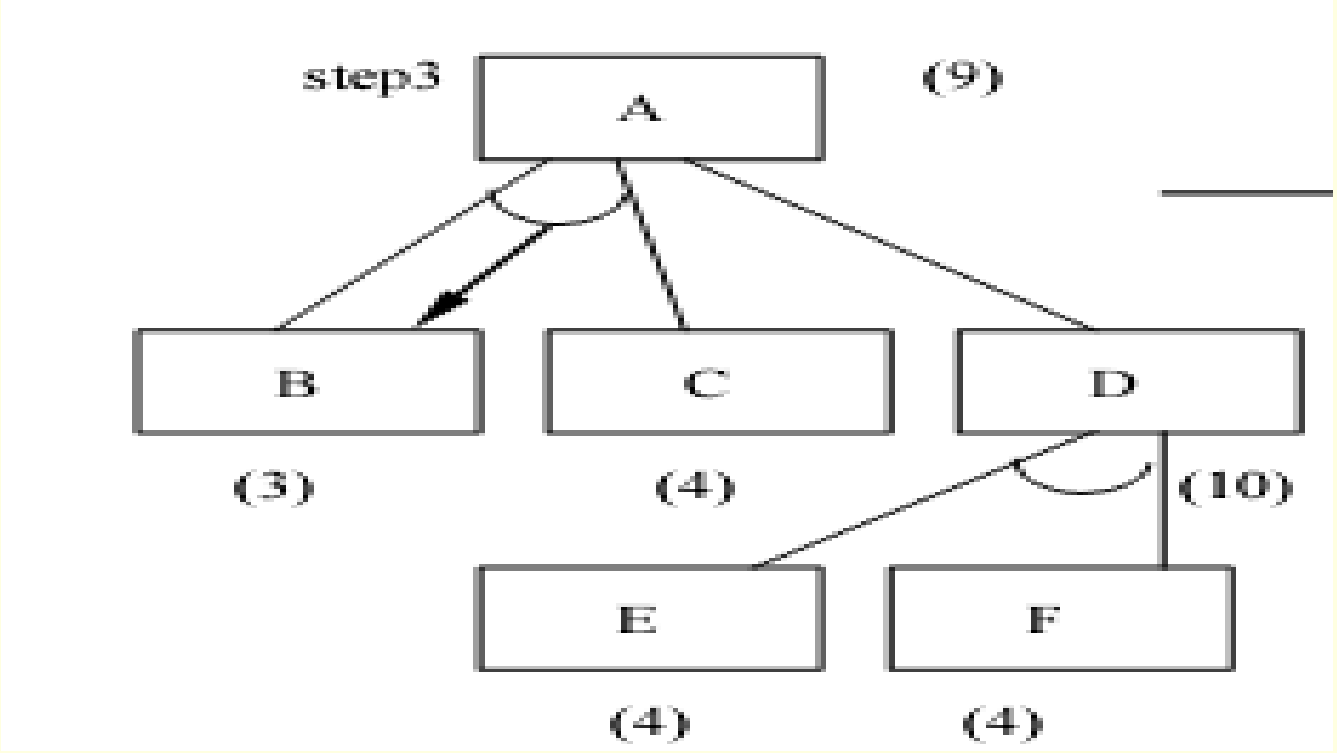
与或图：搜索

- 第二步，扩展A后，得到节点B,C和D, 因为B，C的耗费为9，D的耗费为6，所以把列D的弧标记为出自A最有希望的弧；



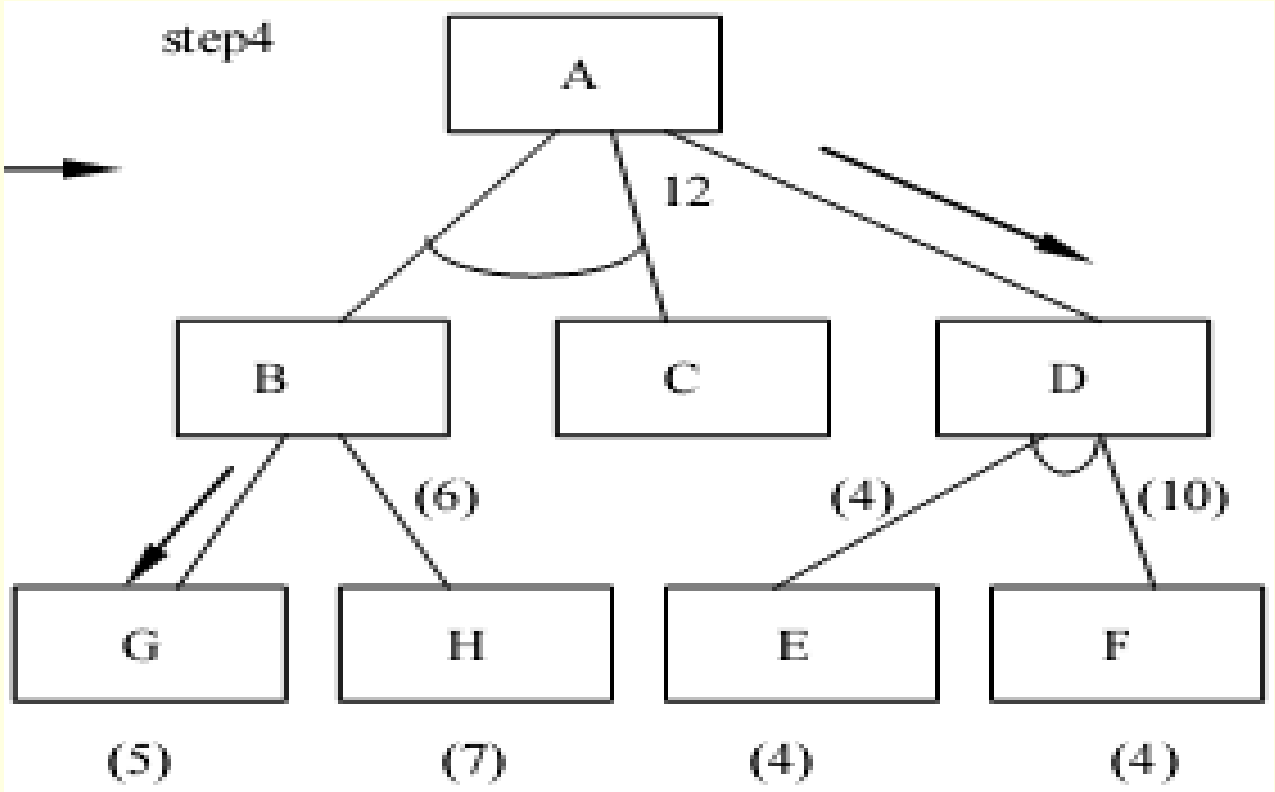
与或图：搜索

- 第三步，选择对D的扩展，得到E和F的与弧，其耗费估计值为10。此时回退一步后，发现与弧BC比D更好，所以将弧BC标志为目前最佳路径；



与或图：搜索

- 第四步，在扩展B后,再回传值发现弧BC的耗费为12（6+4+2），所以D再次成为当前最佳路径。



与或图：搜索

- 最后求得的耗费为:
  - h’(A)=min(12,4+4+2+1)=11
- 以上搜索过程由两个主要步骤组成：
  - 自顶向下，沿当前最优路产生后继节点
  - 自底向上，作估计耗费值修正，再重新选择最优路径。

与或图：AO*算法

- AO*算法基本步骤:

  1. 令G=Init,计算h’(Init) ；

  2. 在Init标记为solved之前或h’(Init)变成Futility之前，执行以下步骤：

     2.1 沿始于Init的已带标志的弧，选出当前沿标志路上未扩展的节点之一扩展（即求后继节点），此节点称为node。

与或图：AO*算法

- AO*算法基本步骤:

2.2生成node的后继节点。

若无后继节点，则令h’(node)=Futility,说明该节点不可解；

若有后继节点，称为successor,对每个不是node祖先的后继节点（避免回路），执行下述步骤

2.2.1 将successor加入G。

2.2.2 若successor$\in S_g$,则标志successor为solved，且令h’(successor)=0。

2.2.3 若successor$\in S_g$,则求h’(successor)

与或图：AO*算法

- AO*算法基本步骤:

2.3 自底向上作评价值修正，重新挑选最优路径。

// 令S为一节点集。

// S＝｛已标记为solved的点，或h’值已改变,

//　　需回传至其先辈节点的节点｝

令S初值＝｛node｝,重复下述过程，直到S为空时停止。

2.3.1　从S中挑选一节点,该节点的后辈点均不在S中(保证每一正在处理的点都在其先辈节点之前作处理),此节点称为current,并从S中删除;

与或图：AO*算法

- AO*算法基本步骤:

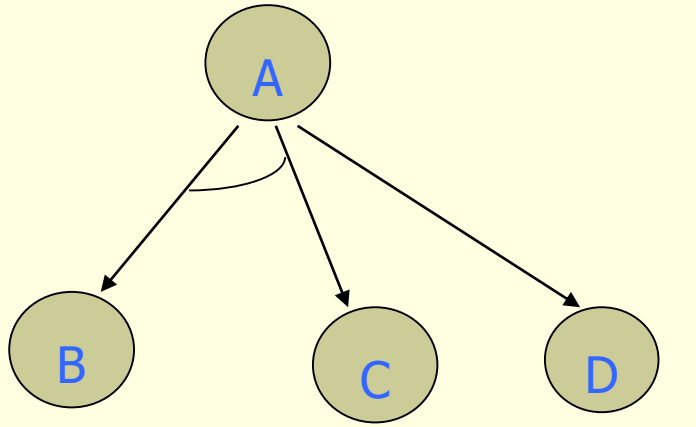
  2.3.2计算始于current的每条弧的费用,即每条弧本身的费用加上弧末端节点h’ 的值(注意区分与,或弧的计算方法),并从中选出极小费用的弧作为h’（current）的新值。

  2.3.3将费用最小弧标志为出自current的最优路径。

  2.3.4若current与新的带标志的弧所连接的节点均被标记为solved,则current被标记为solved

  2.3.5若current已标记为solved或current的耗散值已改变，则需要往回传，因此要把current的所有先辈节点加入S中。

与或图：AO*算法（执行示例）



（3）　（4）　（5）

step2.3.1 S={A}

step2.3.2 current：A

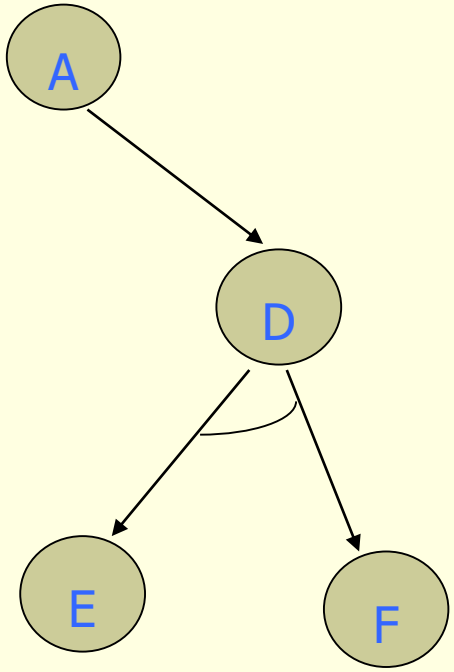
由于有A→B and C的弧，current的费用

=1+1+h’(B)+h’(C)=9

由于有A→D的弧，current的费用=1+5=6

A的费用=min(9,6)=6;

与或图：AO*算法（执行示例）



h’(E) = 4　　h’(F) = 4

node=D，扩展D得

successor={E,F}，

D的耗散值估计已经改变，向上回传，导致A的耗费为min（9，11）=9，所以，最优路径为A→BC弧。

Q & A