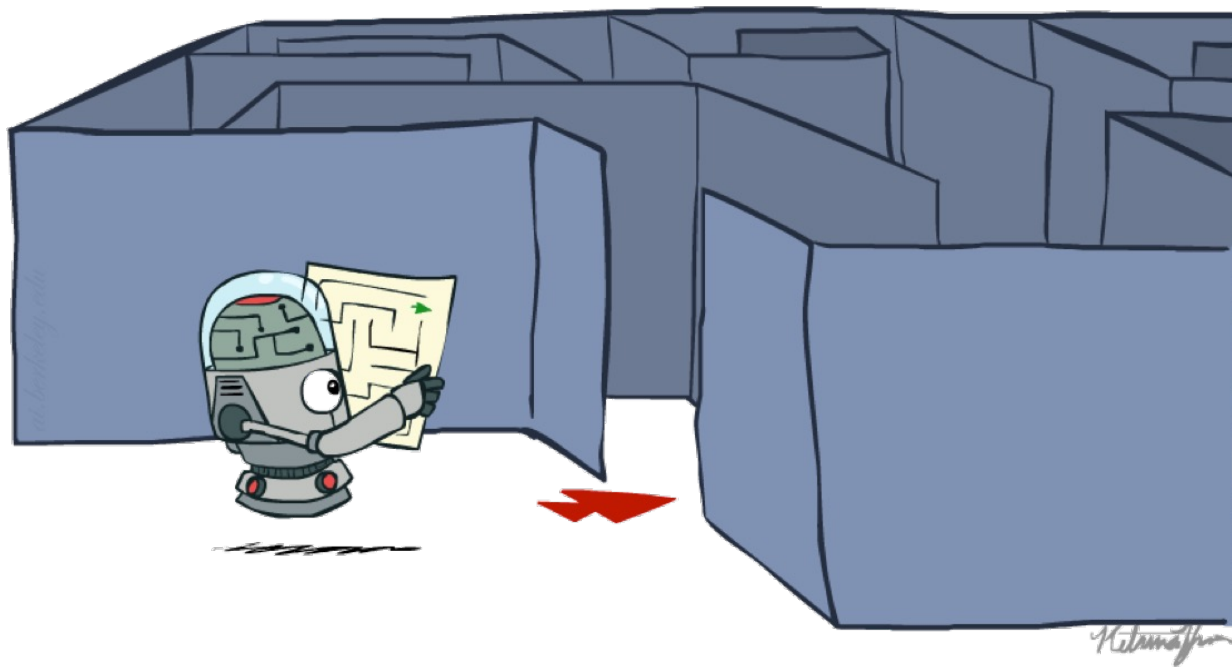


CS 188: 人工智能

搜索

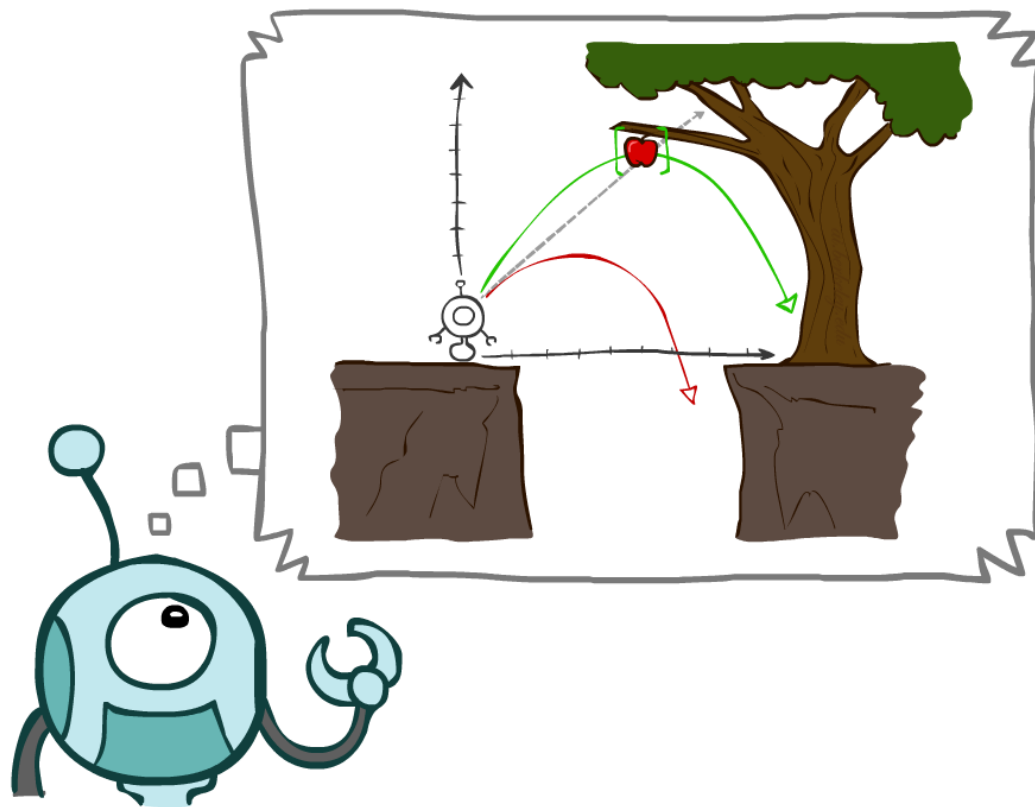


加州大学伯克利分校

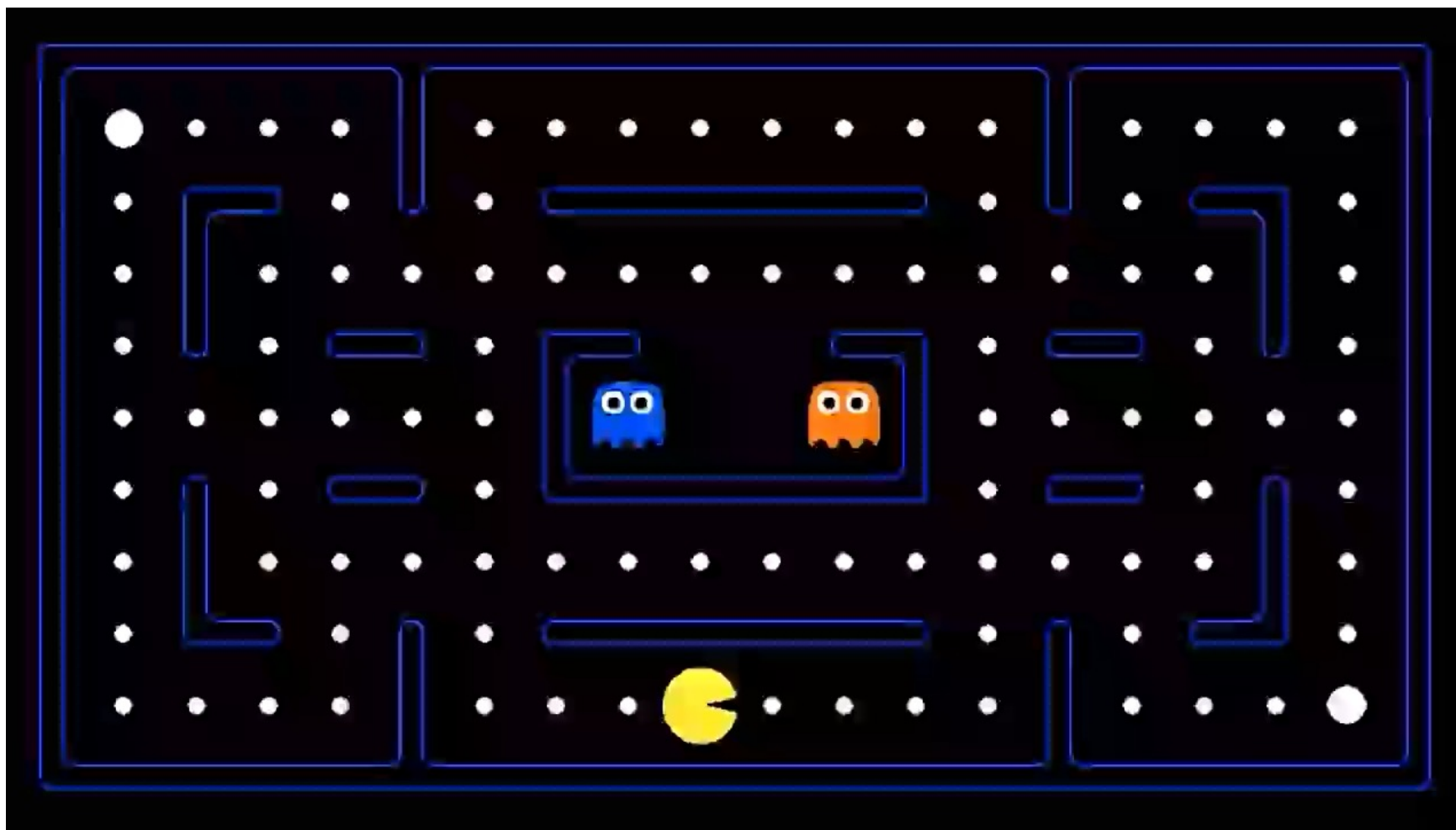
[本课件由 Dan Klein 与 Pieter Abbeel 为 UC Berkeley 《人工智能导论》(CS188) 制作 (ai.berkeley.edu)。]

本节课内容

- 理性智能体
- 搜索问题
- 无信息搜索方法
 - 深度优先搜索
 - 广度优先搜索
 - 一致代价搜索

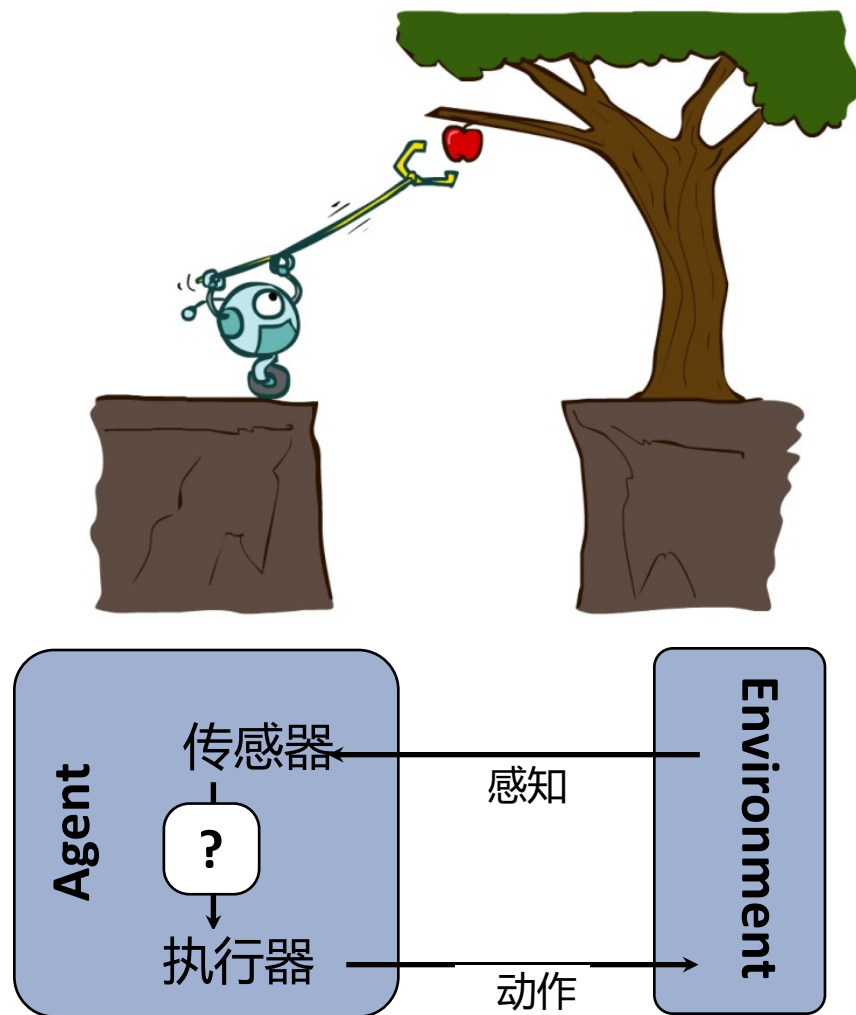


吃豆人智能体

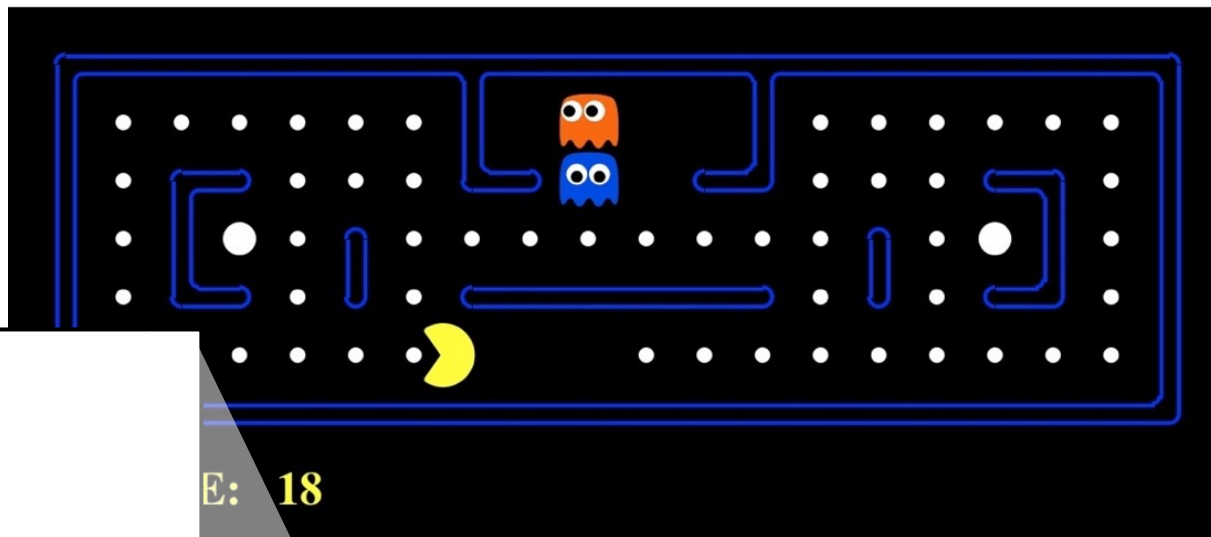


理性智能体

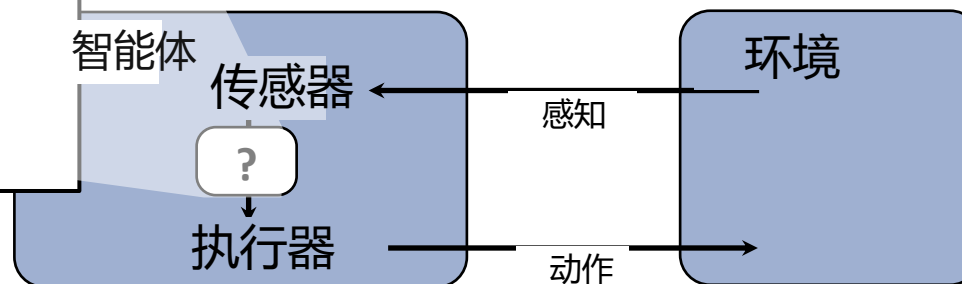
- 智能体 (agent) 是能够感知环境并采取行动的实体。
- 理性智能体会选择使自身 (期望) 效用最大化的行动。
- 感知、环境与动作空间的特性, 决定了选择理性动作所用的技术
- 本课程关注:
 - 针对各类问题的一般人工智能技术
 - 学会识别何时以及如何用已有技术求解新问题



作为智能体的吃豆人

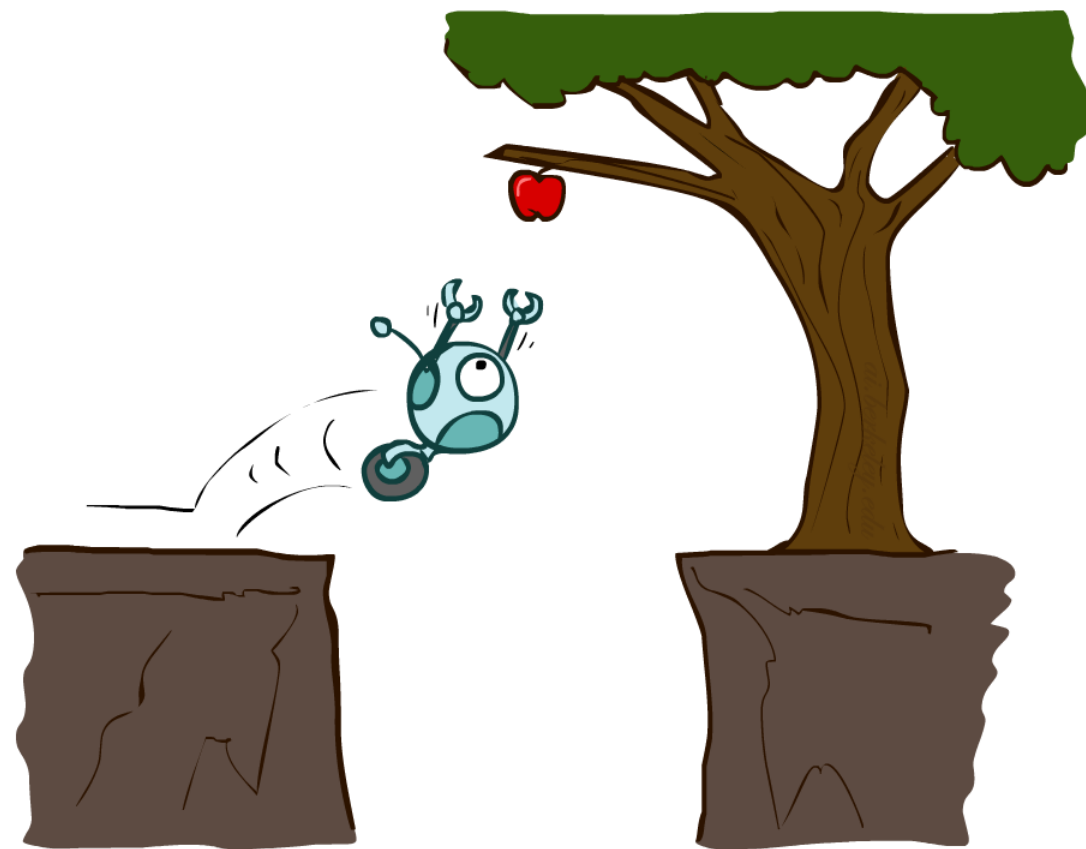


```
class GoWestAgent(Agent):  
  
    def getAction(self, percept):  
        if WEST in percept.getDirections():  
            return WEST  
        else:  
            return STOP
```



反射式智能体

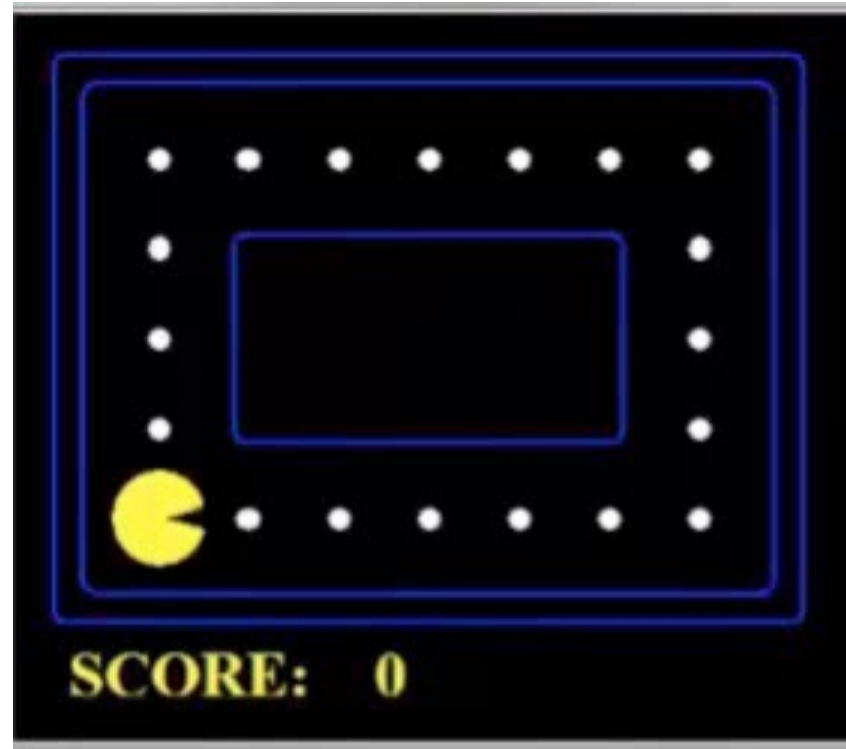
- 反射式智能体：
 - 根据当前感知（或记忆）选择动作
 - 可以拥有记忆或对世界当前状态的模型
 - 不考虑其行动对未来的影响
 - 只考虑世界 “现在是什么样”
- 反射式智能体可以是理性的吗？



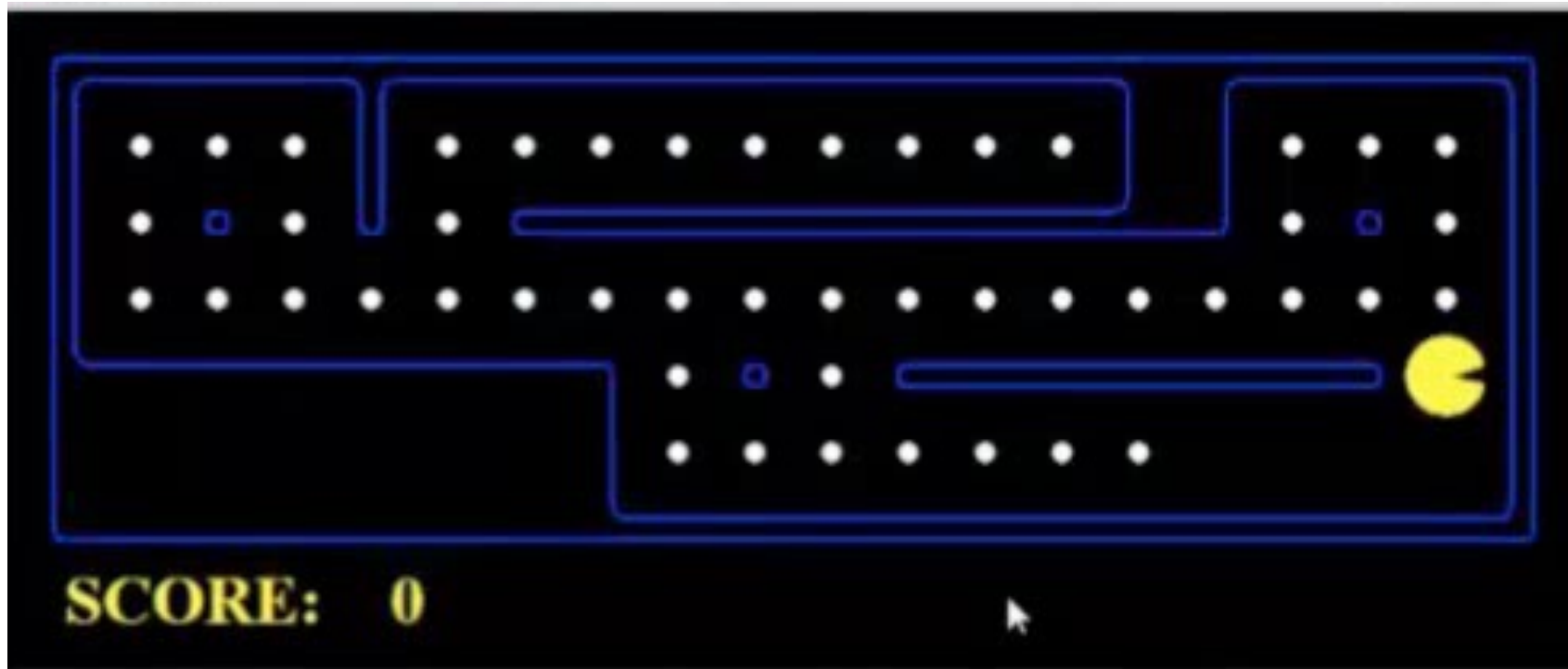
[演示：反射最优 (L2D1)]

[演示：反射最优 (L2D2)]

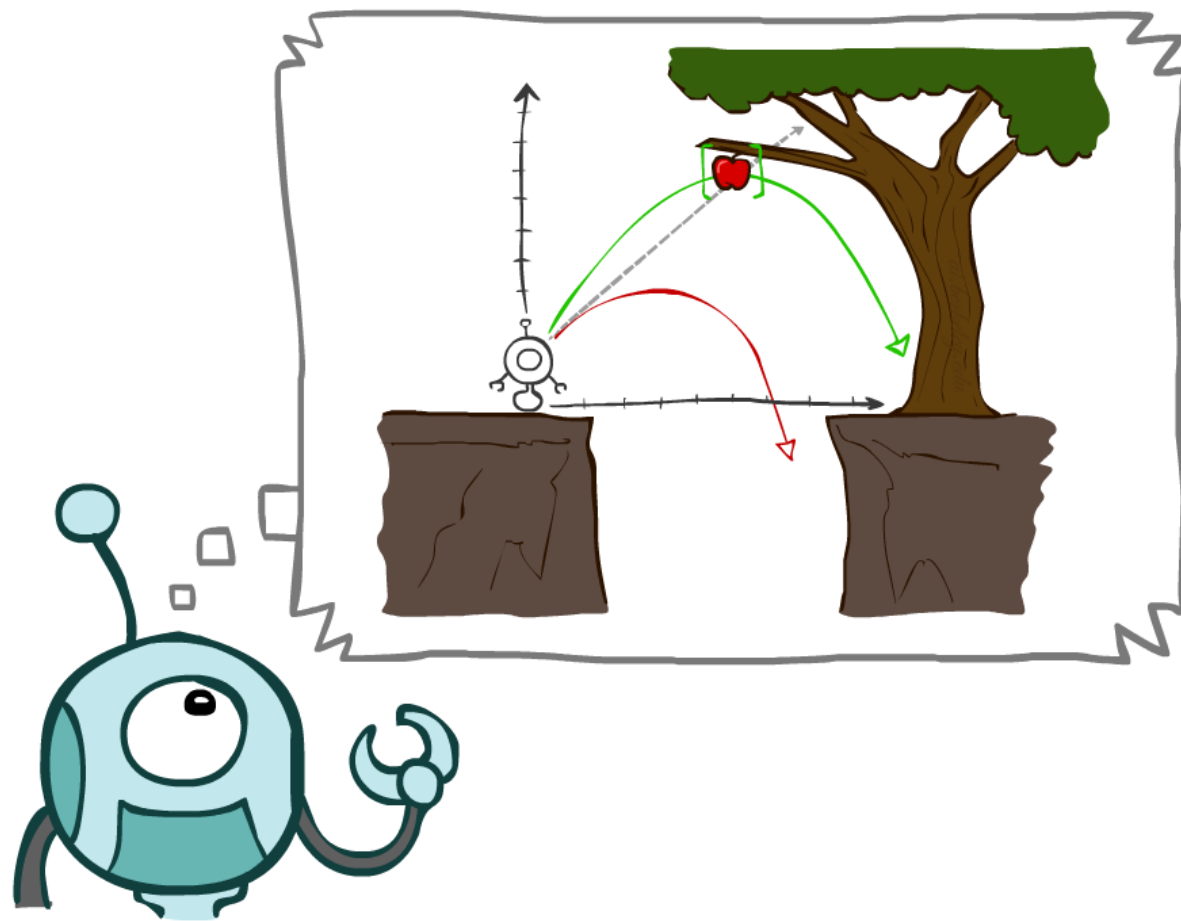
吃掉相邻的豆子（若有）



吃掉相邻的豆子（若有）

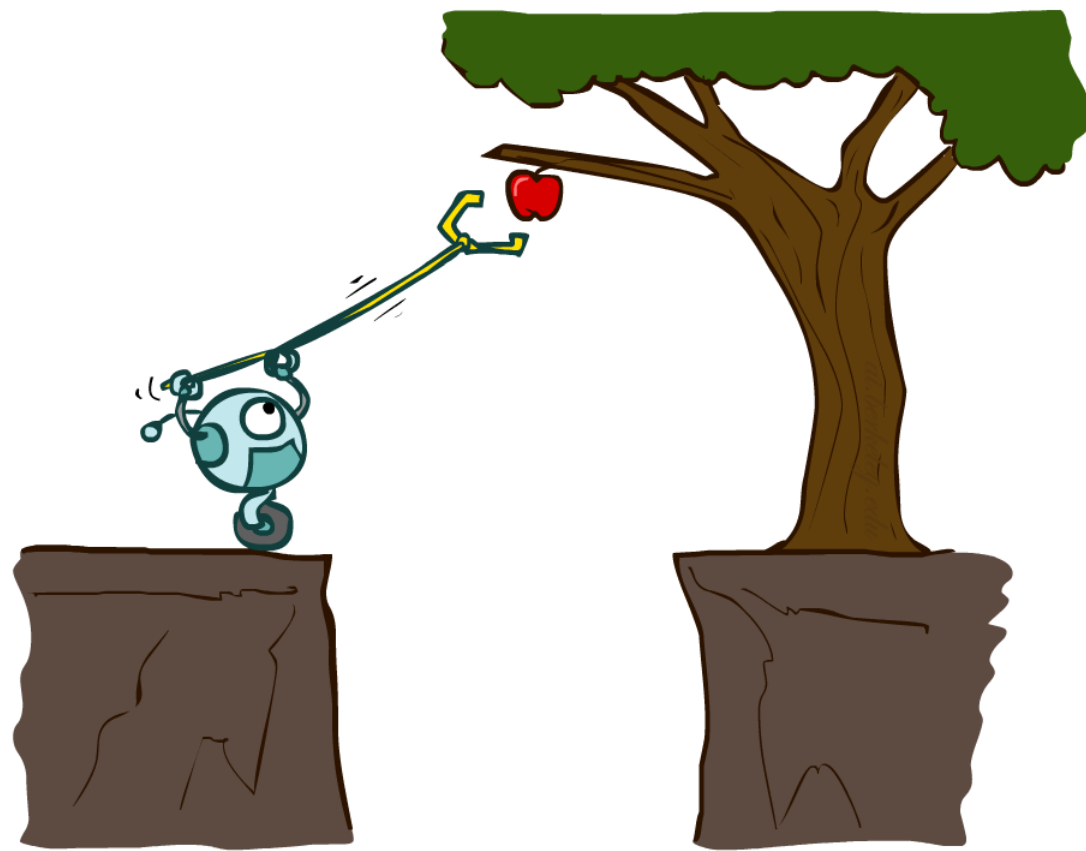


会规划的智能体



规划式智能体

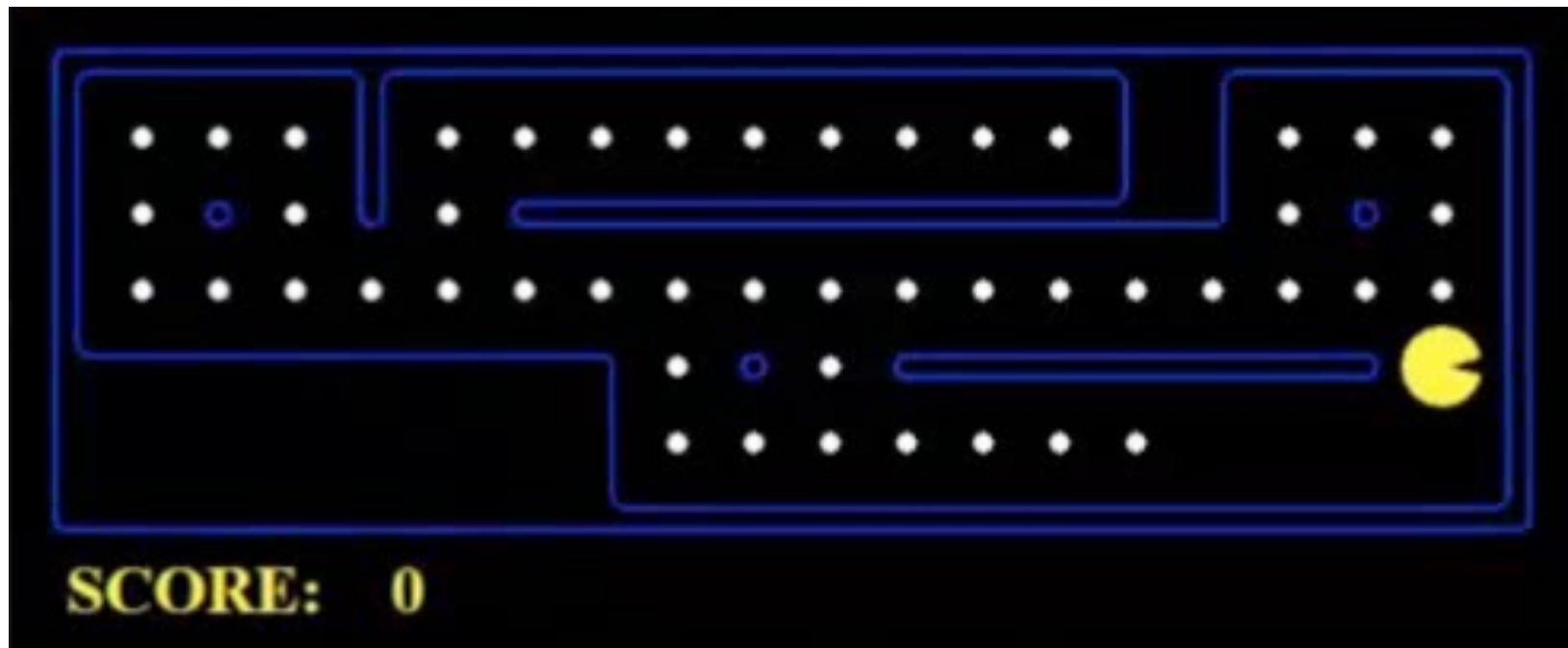
- 规划式智能体：
 - 会问 “如果.....会怎样”
 - 基于（假设的）行动后果做决策
 - 必须拥有世界如何随行动演化的模型
 - 必须明确目标（检验条件）
 - 考虑世界 “将会是什么样”
- 最优规划 vs. 完备规划
- 规划 vs. 重规划



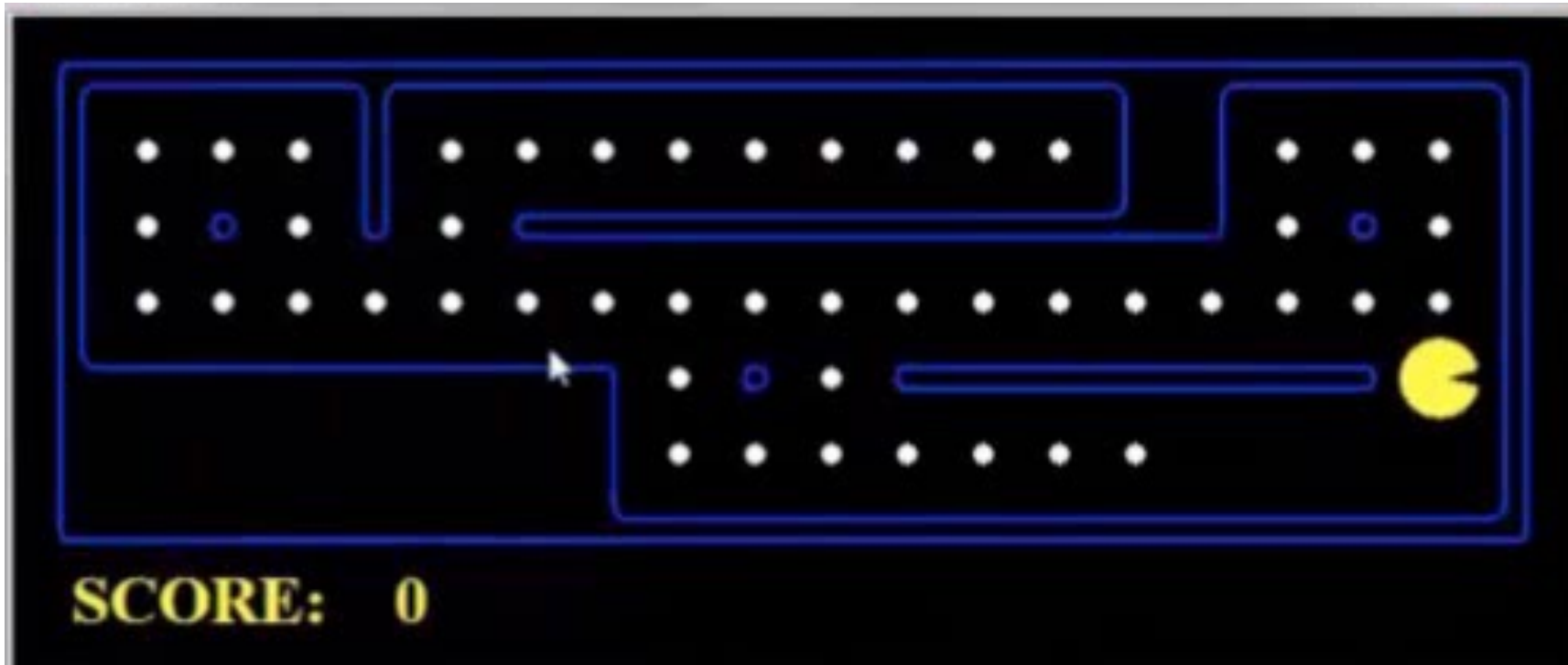
[演示：重规划 (L2D3)]

[演示：Mastermind 猜谜 (L2D4)]

演示视频：重规划



演示视频：Mastermind



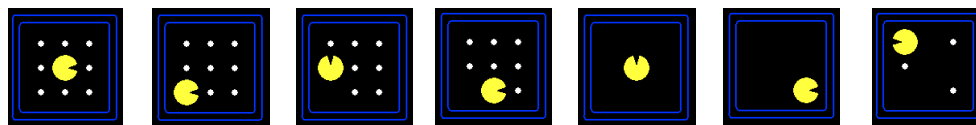
搜索问题



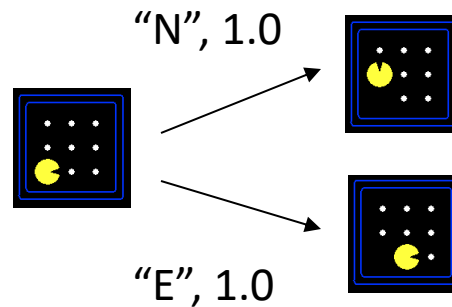
搜索问题

- 一个搜索问题由以下部分组成：

- 状态空间



- 后继函数
(含动作与代价)



- 初始状态与目标检验
- 解是一个动作序列（规划），它把初始状态转化为目标状态

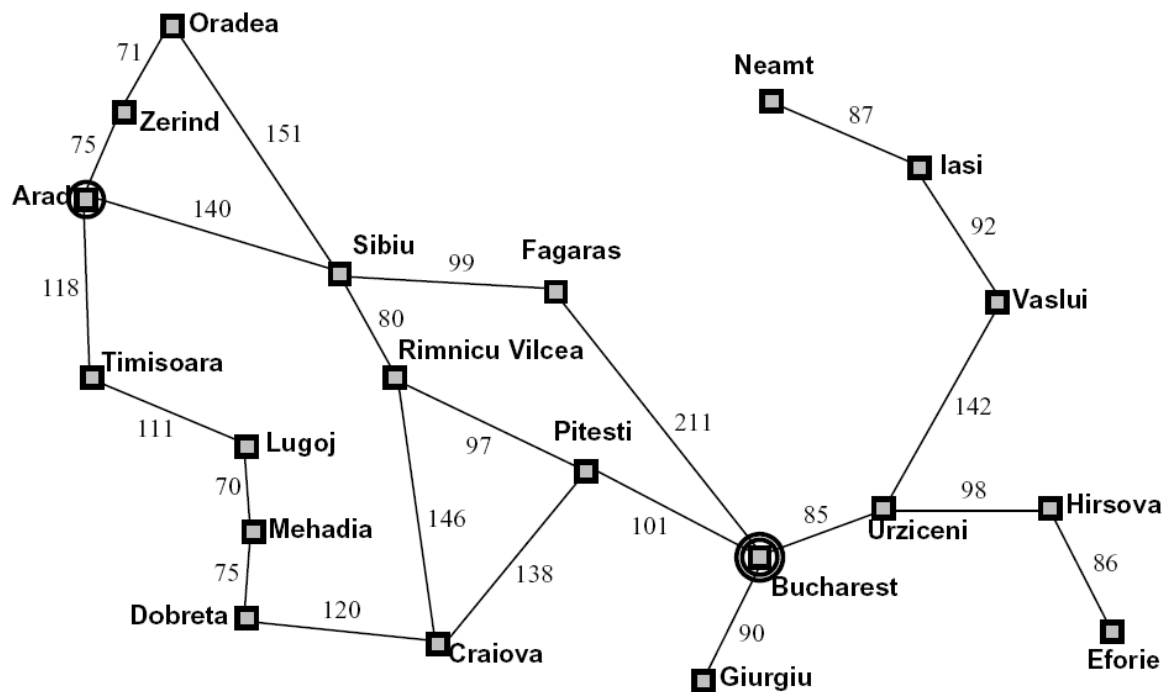
我们为什么关心搜索？

- 我们将讨论的算法应用极其广泛
 - 路径规划 / 路由问题（人或机器人）
 - 芯片设计（布线）
 - 资源规划问题
 - 机器人运动规划
 - 蛋白质设计
 - 语言分析
 - 机器翻译
 - 电子游戏
 - 语音识别

搜索问题即模型



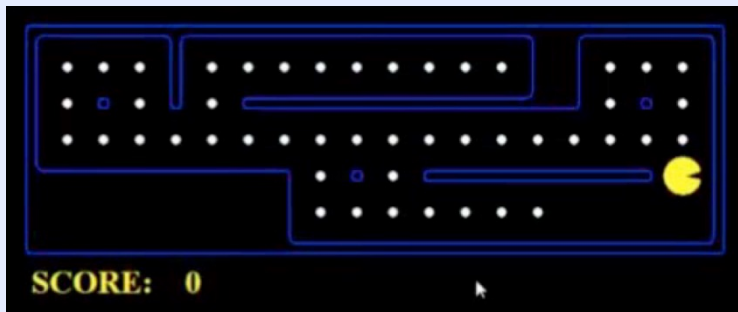
例子：罗马尼亚旅行



- 状态空间：
 - 城市
- 后继函数：
 - 道路：前往相邻城市，代价 = 距离
- 初始状态：
 - Arad (阿拉德)
- 目标检验：
 - 当前状态是否 == Bucharest (布加勒斯特)？
- 解是什么？

状态空间里有什么？

世界状态包含环境的每一个细节



搜索状态只保留规划所需的细节（抽象）

- 问题：寻路

- 状态：(x,y) 位置
- 动作：东南西北
- 后继：只更新位置
- 目标检验：(x,y) == 终点？

- 问题：吃光所有豆子

- 状态：{(x,y), 豆子布尔值}
- 动作：东南西北
- 后继：更新位置，并可能更新豆子布尔值
- 目标检验：所有豆子布尔值为假

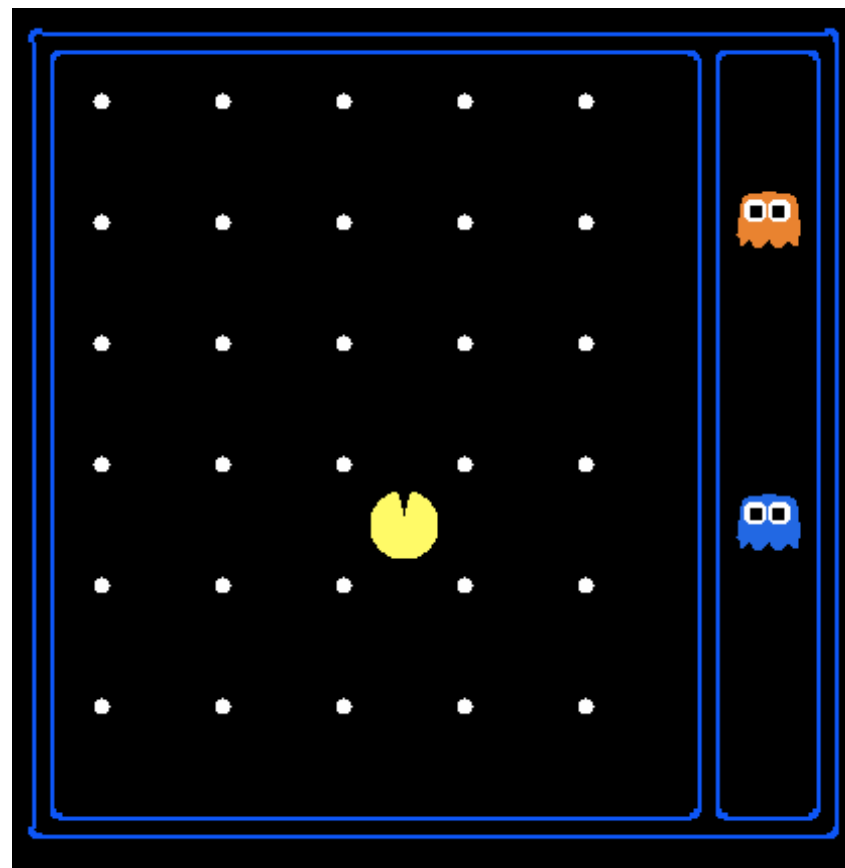
状态空间的规模?

- 世界状态:

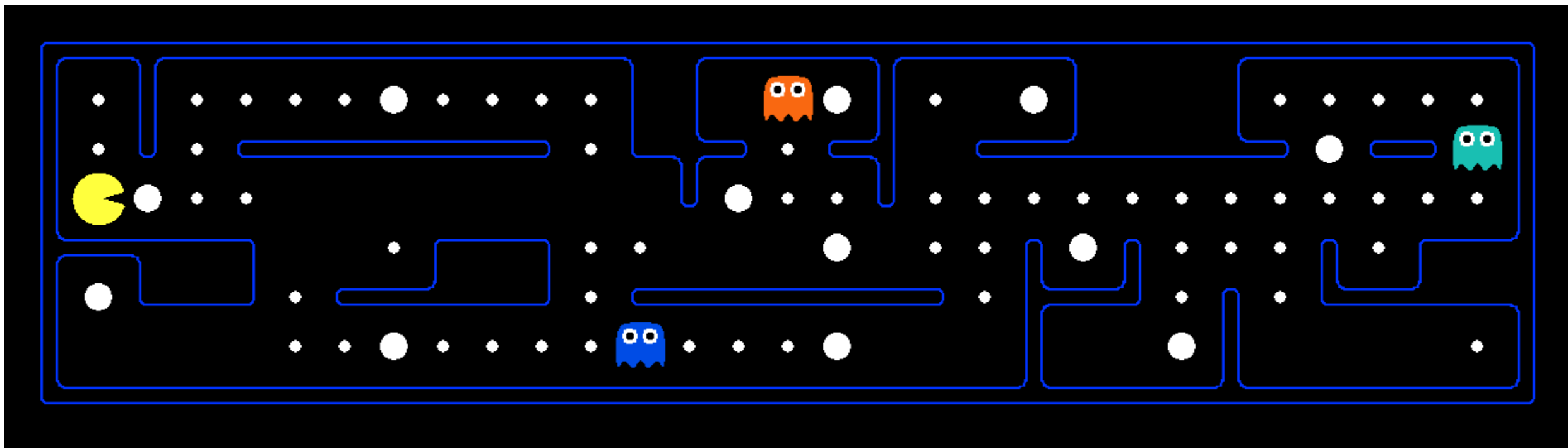
- 智能体在 12×10 网格上的位置
- 30 颗豆子的状态
- 幽灵在 12×1 网格上的位置
- 智能体朝向也有影响: 东南西北

- 数量有多少?

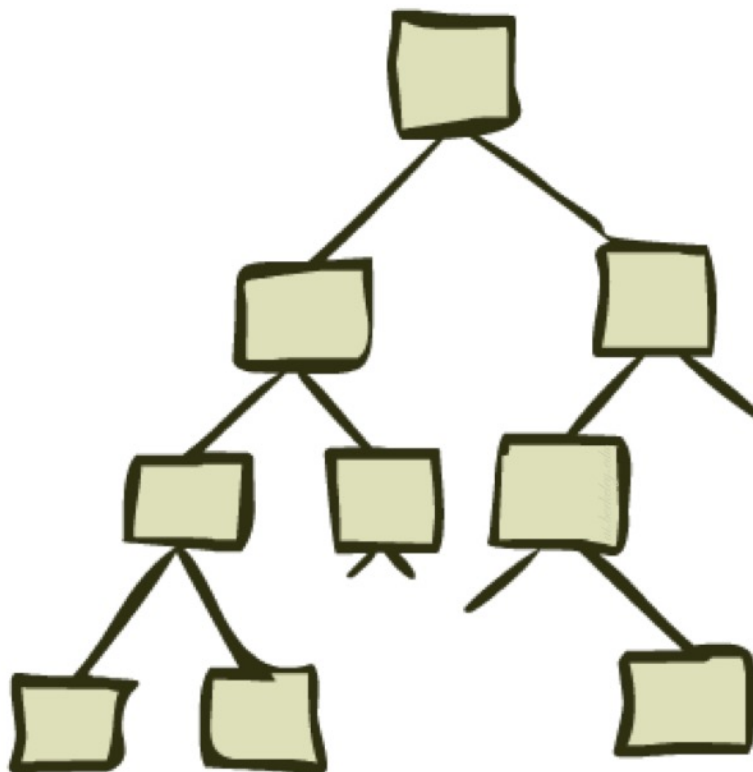
- 世界状态数?
 $120 \times (2^{30}) \times (12^2) \times 4$
- 寻路问题的状态数?
120
- 吃光豆子问题的状态数?
 $120 \times (2^{30})$



练习：安全通行

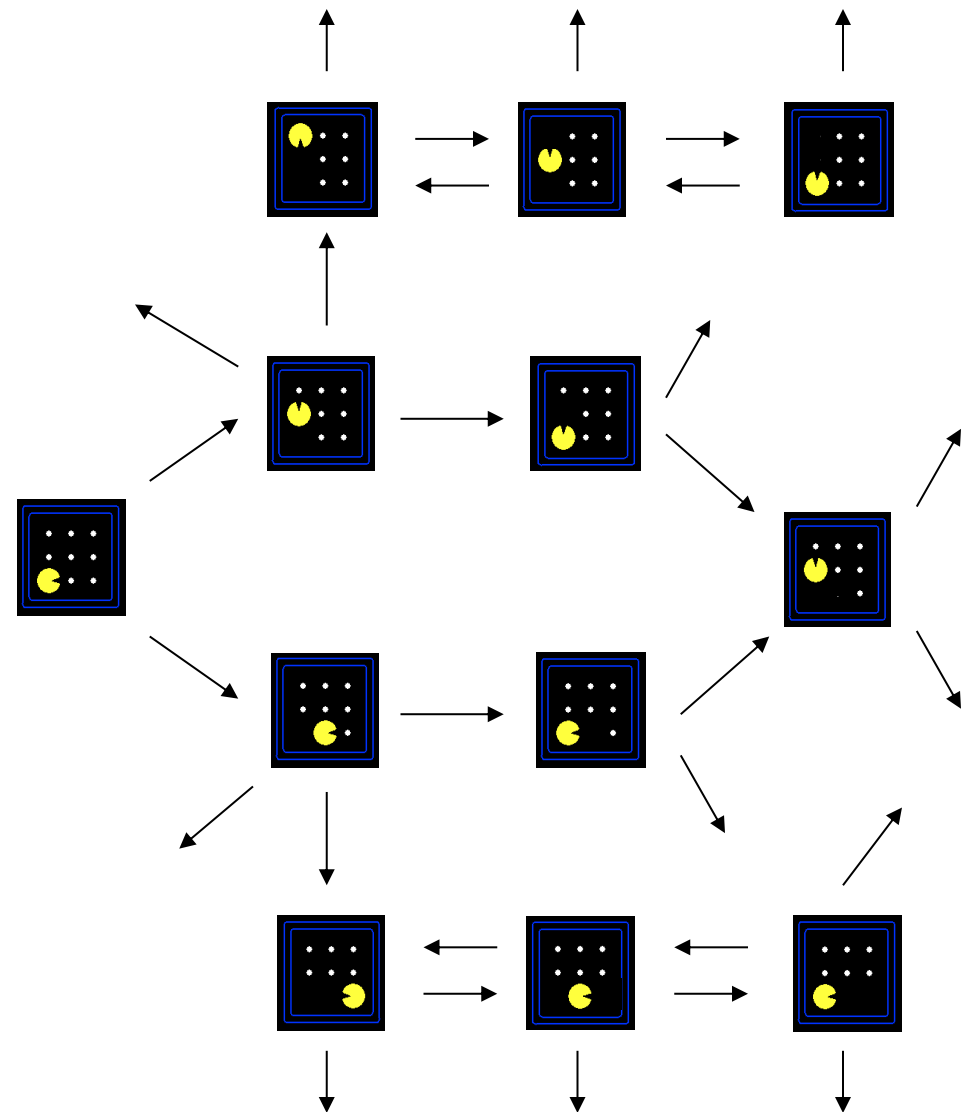


- 问题：在让幽灵始终保持害怕的状态下吃光所有豆子
- 状态空间需要描述哪些内容？
 - （智能体位置、豆子布尔值、能量豆布尔值、剩余害怕时间）



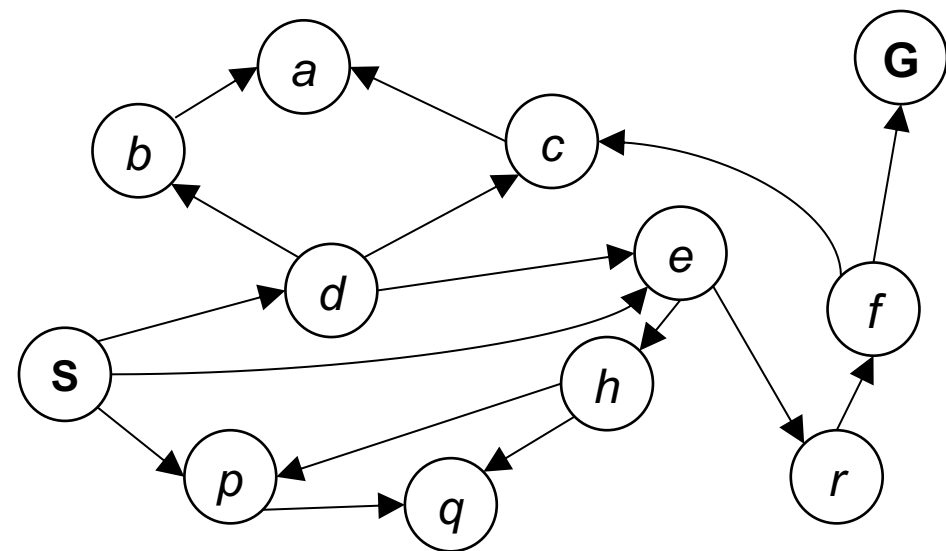
状态空间图

- 状态空间图：搜索问题的一种数学表示
 - 节点是（抽象化的）世界状态配置
 - 弧表示后继关系（动作的结果）
 - 目标检验对应一组目标节点（可能只有一个）
- 在状态空间图中，每个状态只出现一次！
- 我们很少能把整张图放进内存（它太大了），但它是有用的概念



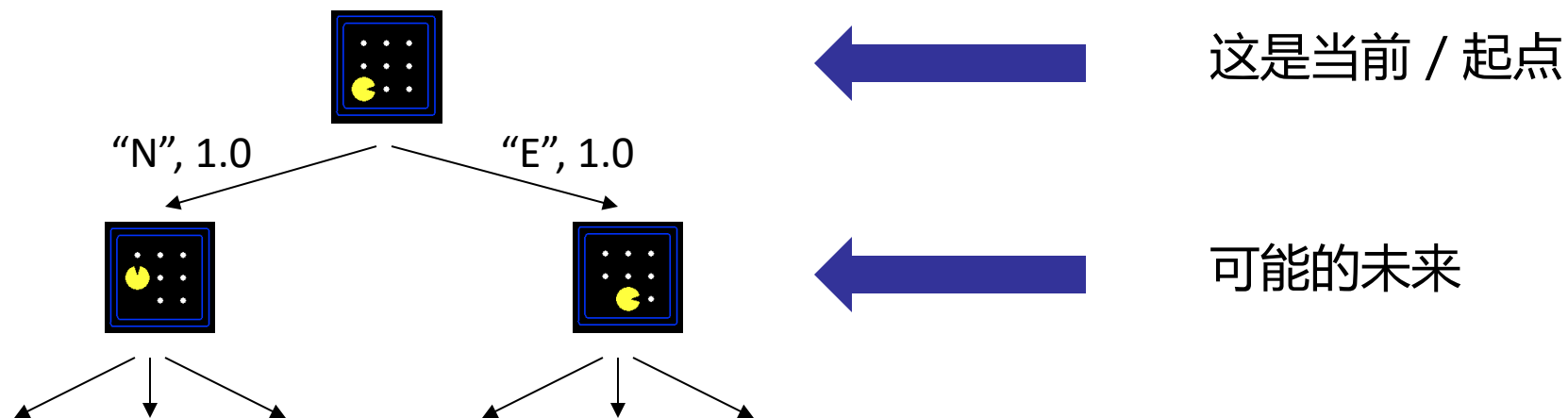
状态空间图

- 状态空间图：搜索问题的一种数学表示
 - 节点是（抽象化的）世界状态配置
 - 弧表示后继关系（动作的结果）
 - 目标检验对应一组目标节点（可能只有一个）
- 在状态空间图中，每个状态只出现一次！
- 我们很少能把整张图放进内存（它太大了），但它是有用的概念



微型搜索问题的状态空间图

搜索树

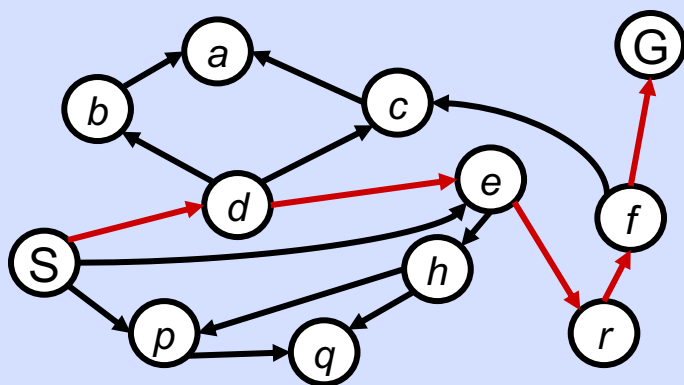


- 搜索树:

- 一棵关于“如果.....会怎样”的规划及其结果的树
- 初始状态是根节点
- 子节点对应后继
- 节点显示状态，但对应的是达到这些状态的“规划”
- 对大多数问题，我们永远无法真正构建出整棵树

状态空间图 vs. 搜索树

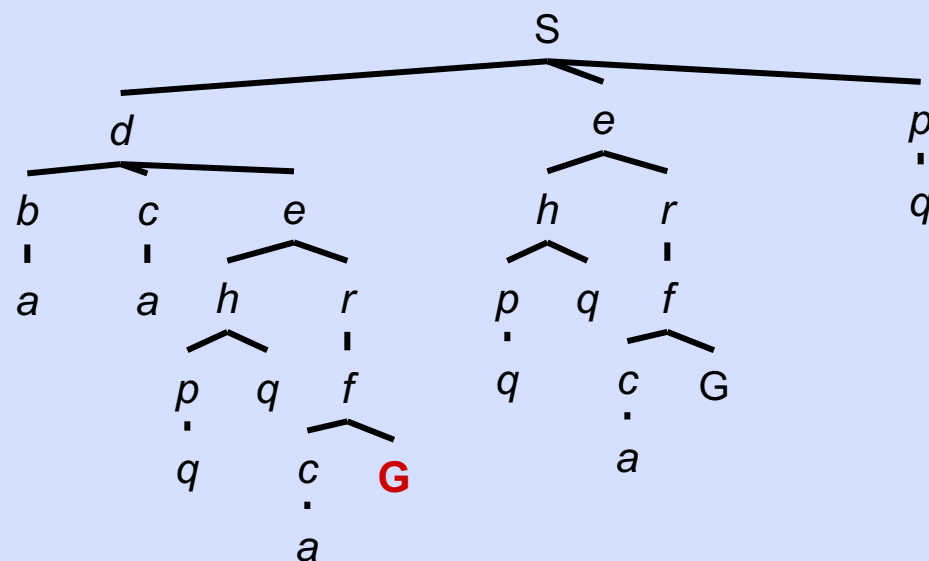
状态空间图



搜索树中的每个节点，对应状态空间图中的一条完整路径。

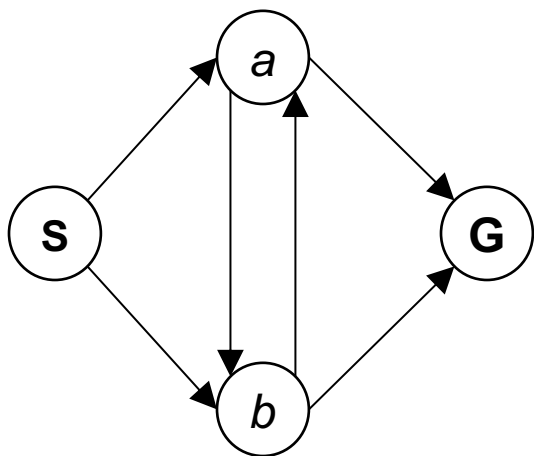
两者都按需构建并且尽可能少地构建。

搜索树



练习：状态空间图 vs. 搜索树

考虑这个 4 状态的图：

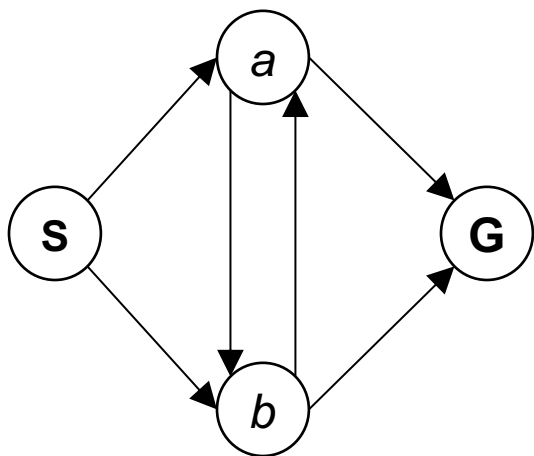


它（从 S 出发）的搜索树有多大？

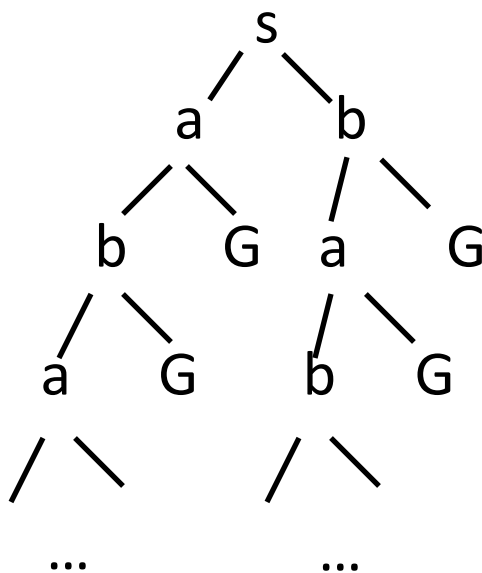


练习：状态空间图 vs. 搜索树

考虑这个 4 状态的图：

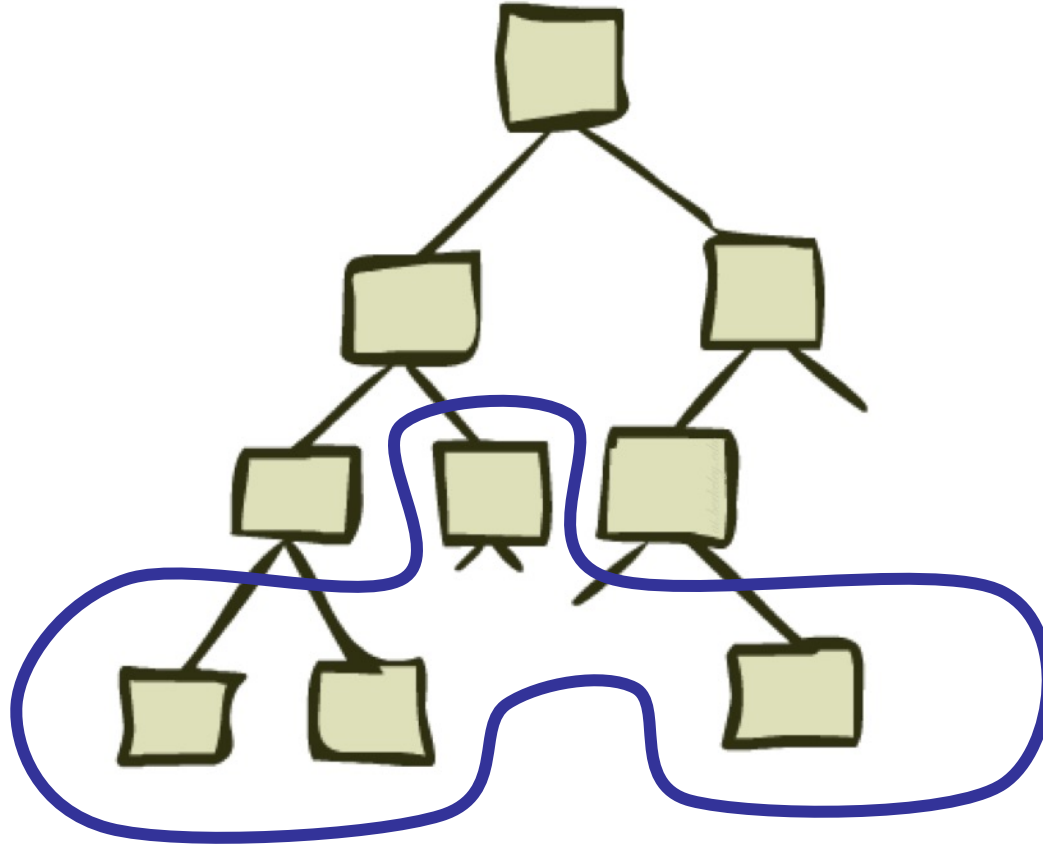


它（从 S 出发）的搜索树有多大？

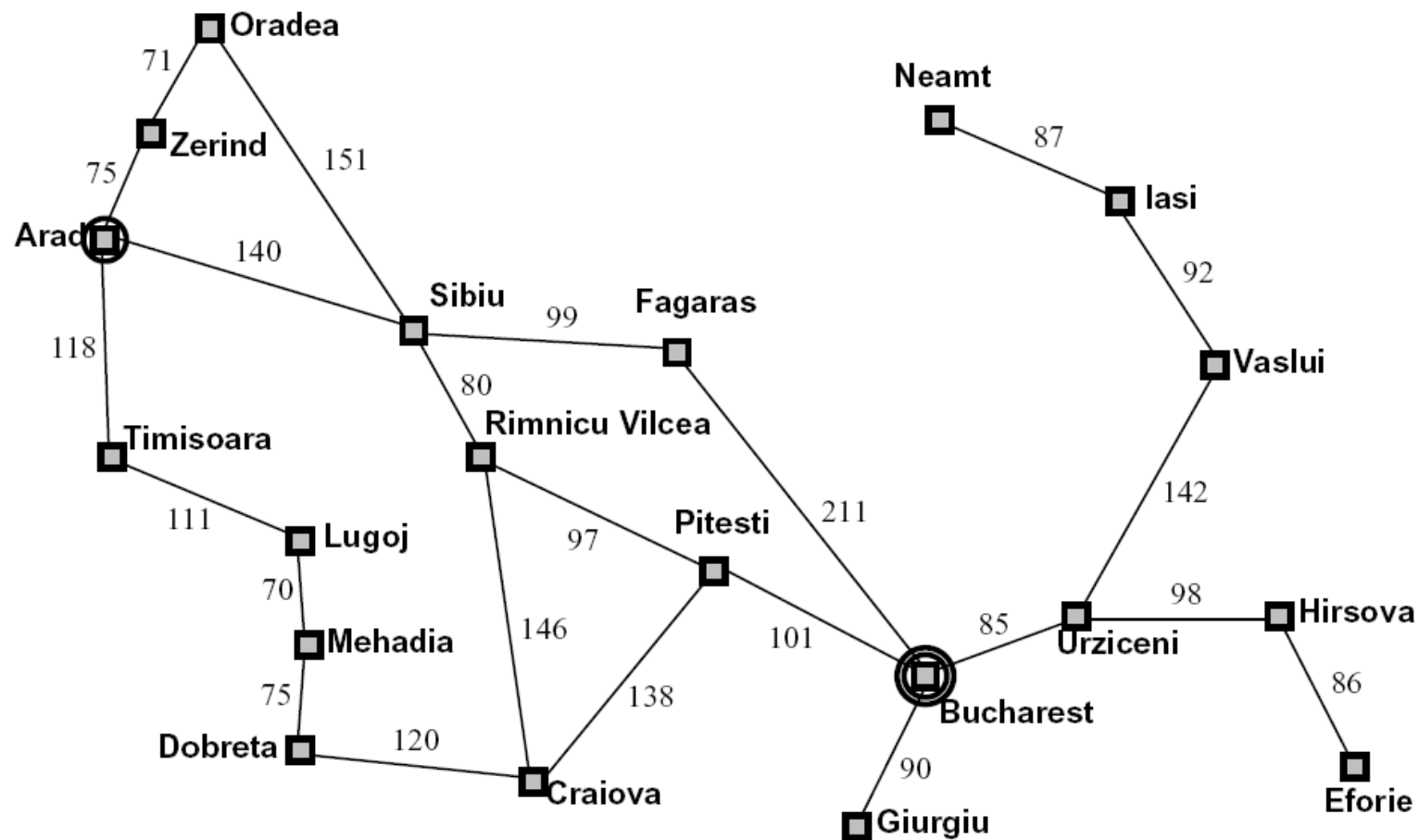


注意：搜索树中存在大量重复结构！

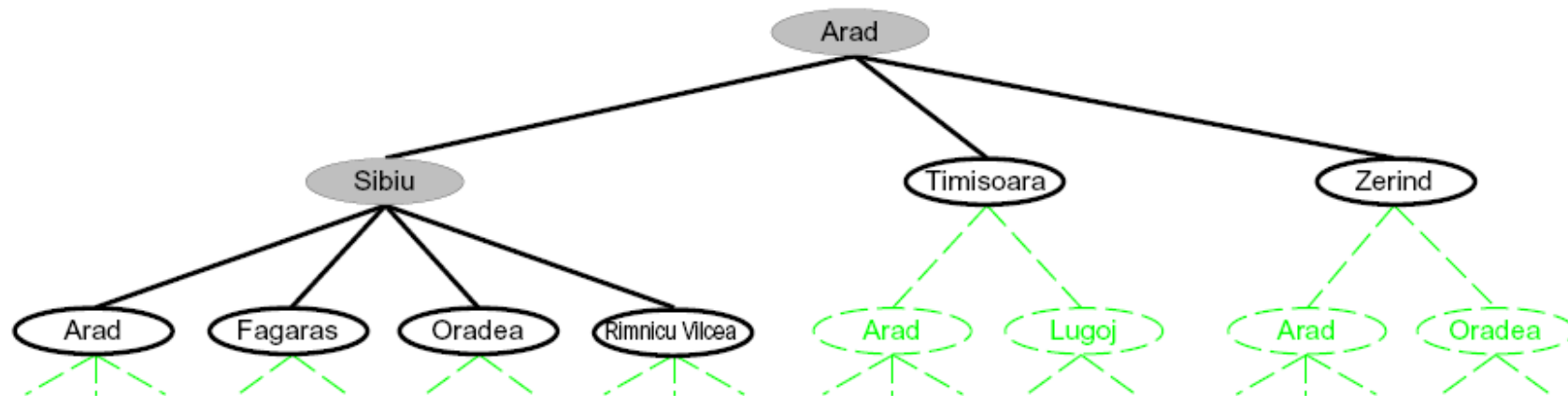
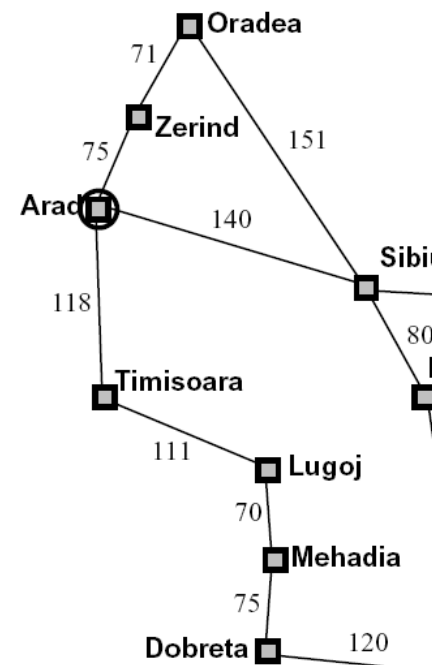
树搜索



例子：罗马尼亚搜索



在搜索树上进行搜索



- 搜索：

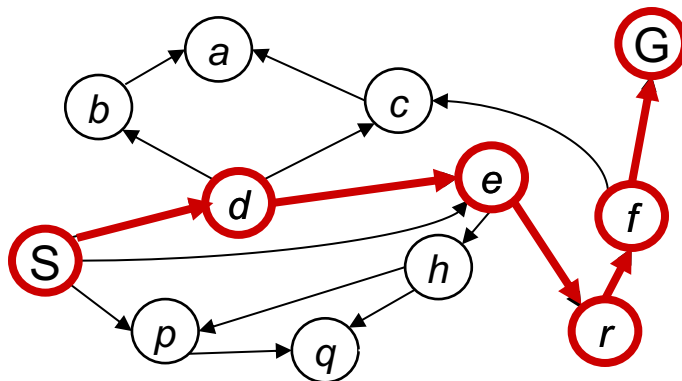
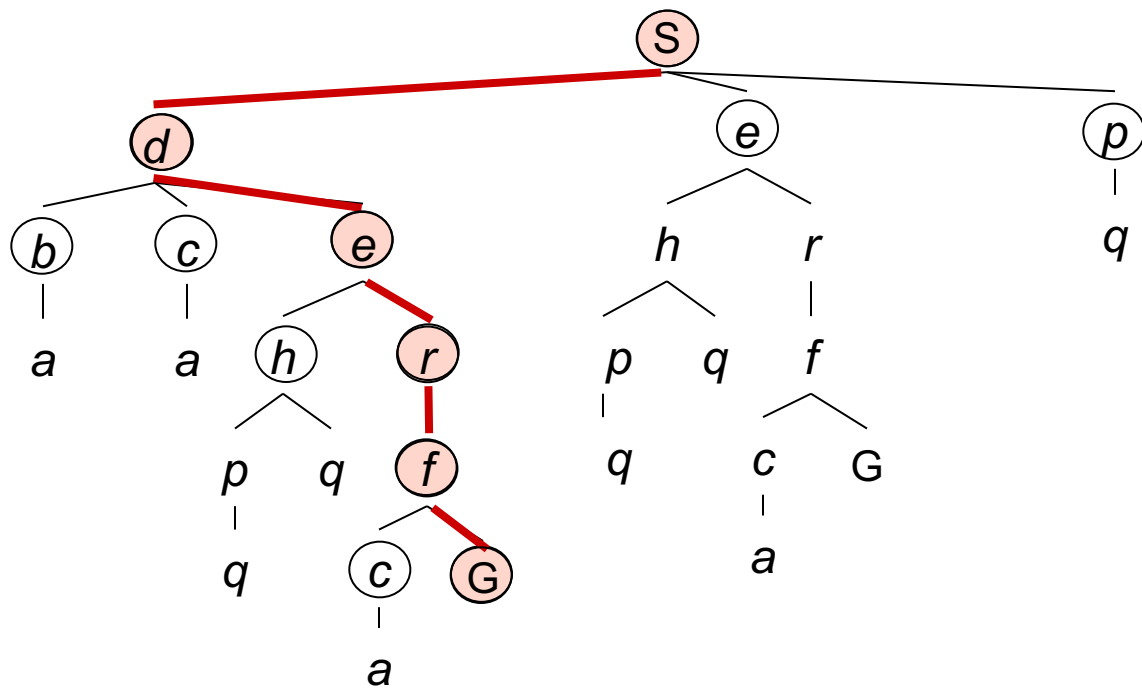
- 展开可能的规划（树节点）
- 维护一个由待考察的部分规划组成的边缘（fringe）
- 尽量少展开树节点

通用树搜索

```
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
```

- 重要概念：
 - 边缘 (fringe)
 - 扩展
 - 探索策略
- 核心问题：应该探索哪些边缘节点？

例子：树搜索



~~S~~
~~S → d~~
S → e
S → p
S → d → b
S → d → c
~~S → d → e~~
S → d → e → h
~~S → d → e → r~~
~~S → d → e → r → f~~
S → d → e → r → f → c
~~S → d → e → r → f → G~~

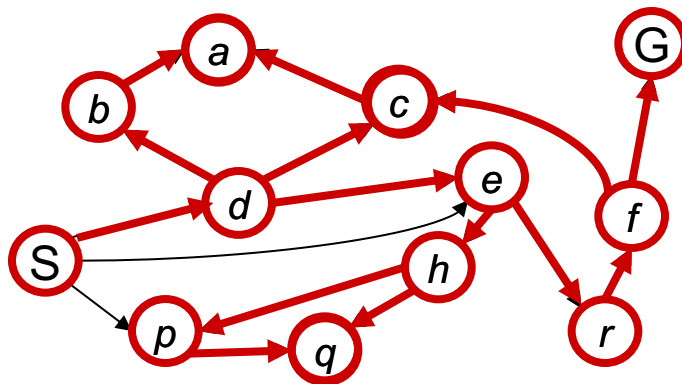
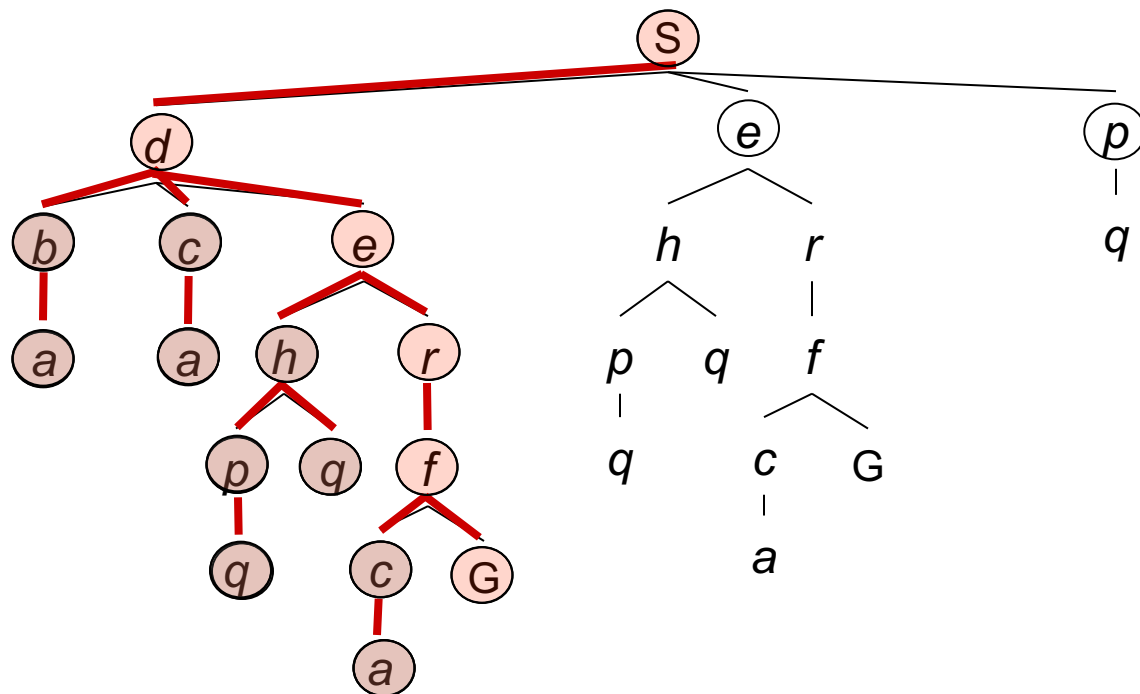
深度优先搜索



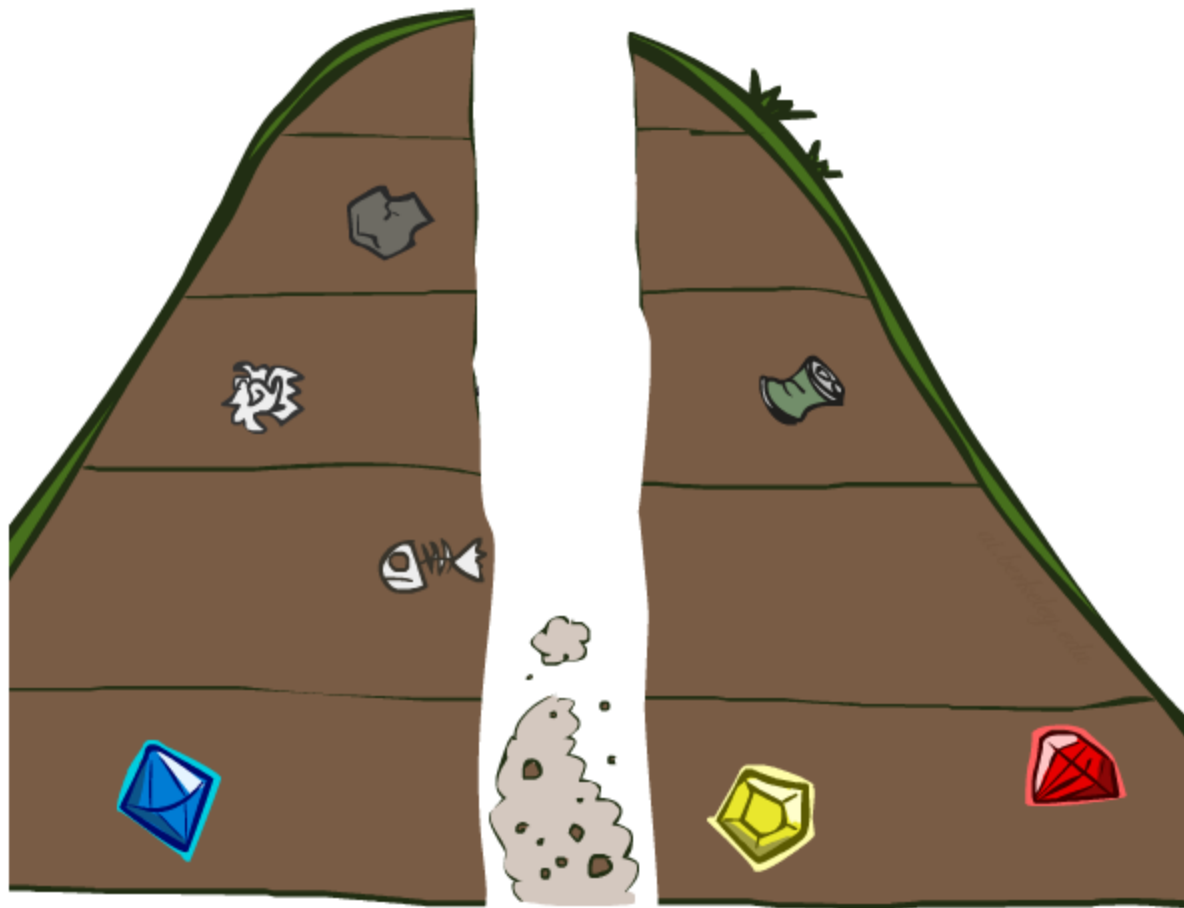
深度优先搜索

策略：优先展开
最深的节点

实现：
边缘是 LIFO 栈



搜索算法的性质



搜索算法的性质

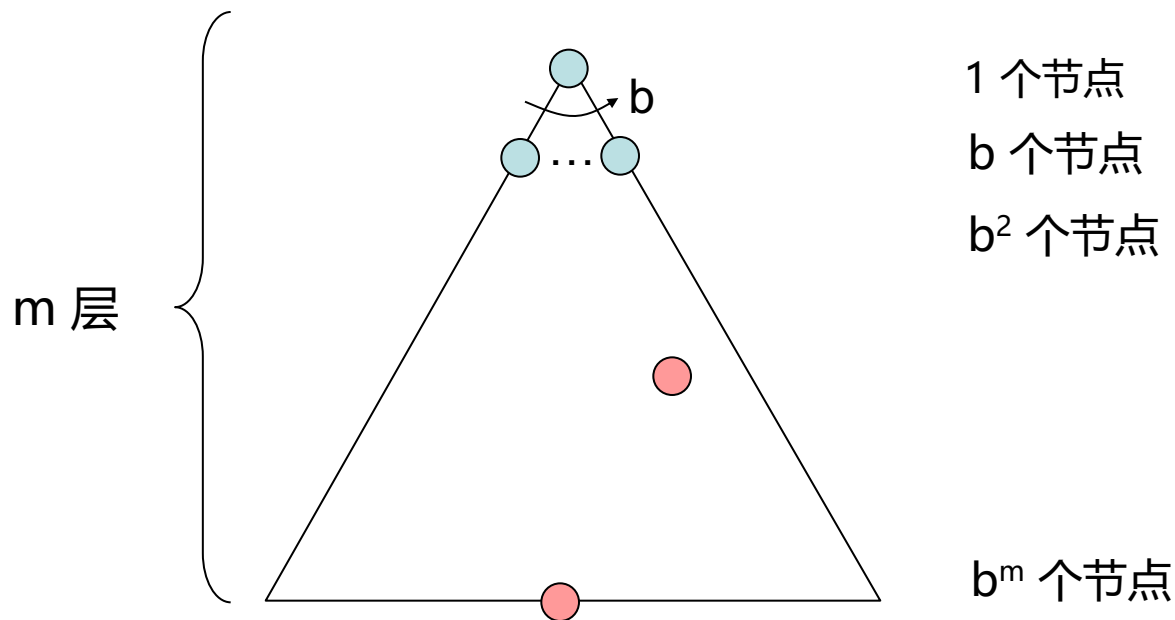
- 完备性：若解存在，能保证找到解吗？
- 最优性：能保证找到代价最小的路径吗？
- 时间复杂度？
- 空间复杂度？

- 搜索树示意：

- b 是分支因子
- m 是最大深度
- 解可能出现在不同深度

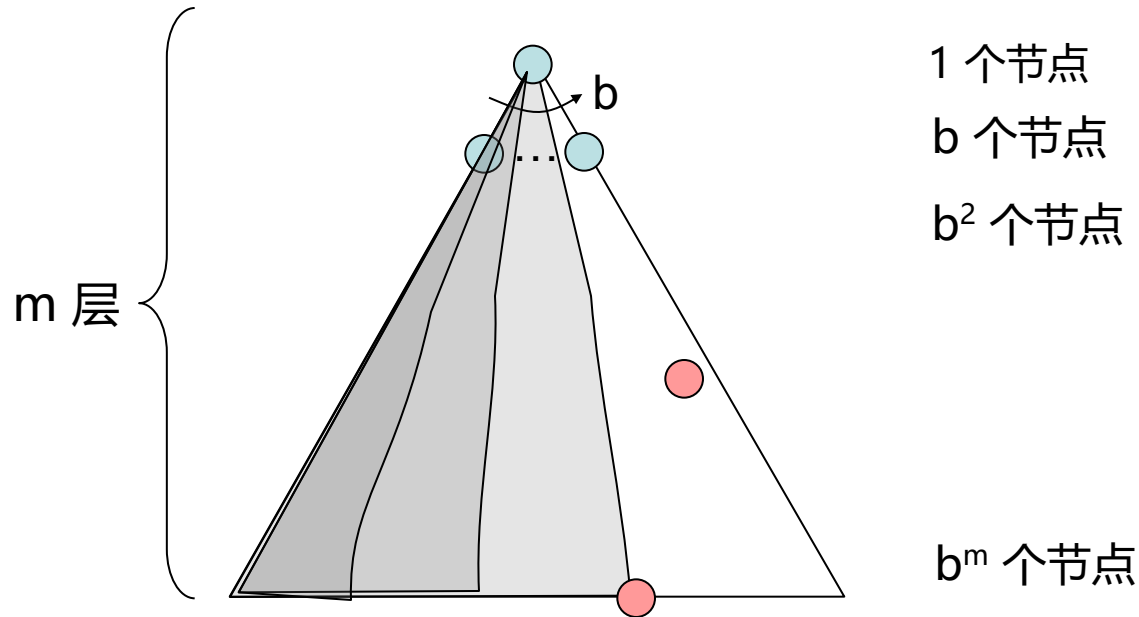
- 整棵树的节点数？

- $1 + b + b^2 + \dots + b^m = O(b^{m+1})$



深度优先搜索 (DFS) 的性质

- DFS 会展开哪些节点？
 - 树的某条最左分支。
 - 可能处理整棵树！
 - 若 m 有限，时间为 $O(b^m)$
- 边缘占用多少空间？
 - 只保存到根路径上的兄弟节点，所以 $O(b^m)$
- 是否完备？
 - m 可能无穷大，所以只有防止循环时才完备（后述）
- 是否最优？
 - 否，它找到的是“最左边”的解，不管深度或代价



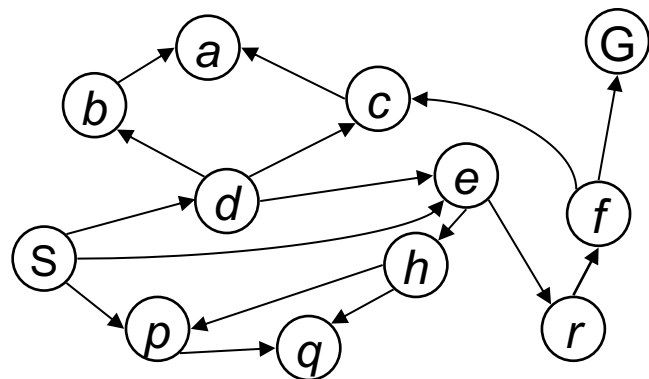
广度优先搜索



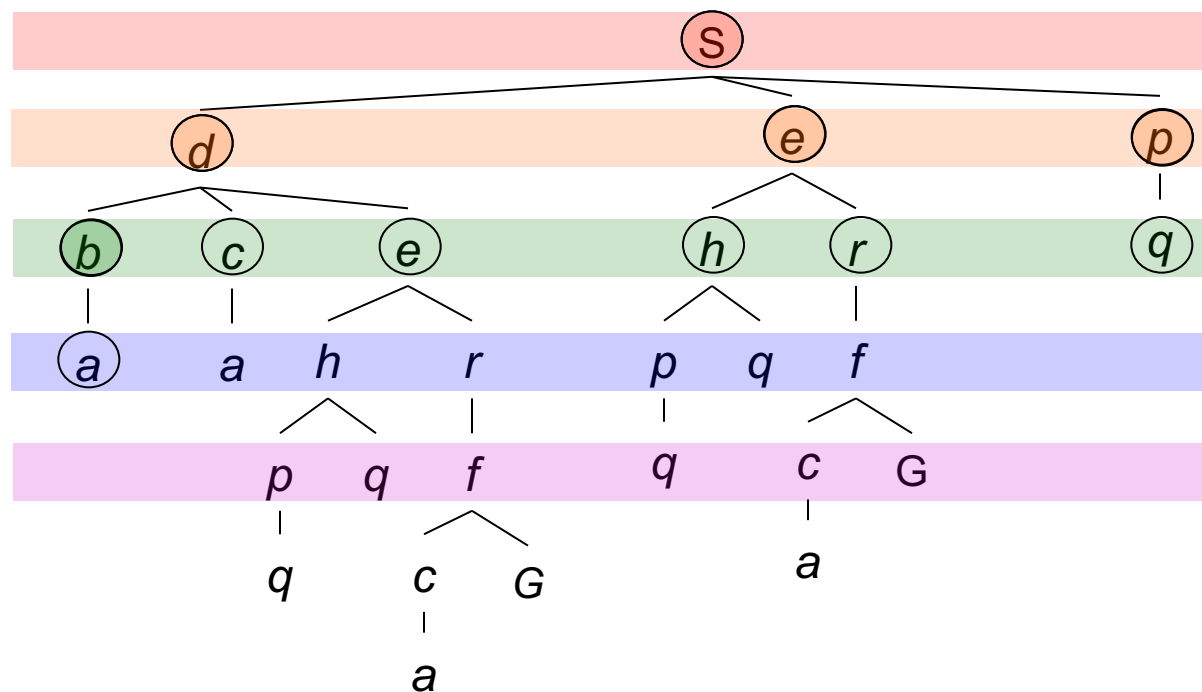
广度优先搜索

策略：优先展开
最浅的节点

实现：边缘
是 FIFO 队列



搜索
层



广度优先搜索 (BFS) 的性质

- BFS 会展开哪些节点?

- 处理最浅解之上的所有节点
- 设最浅解的深度为 s
- 搜索耗时 $O(b^s)$

- 边缘占用多少空间?

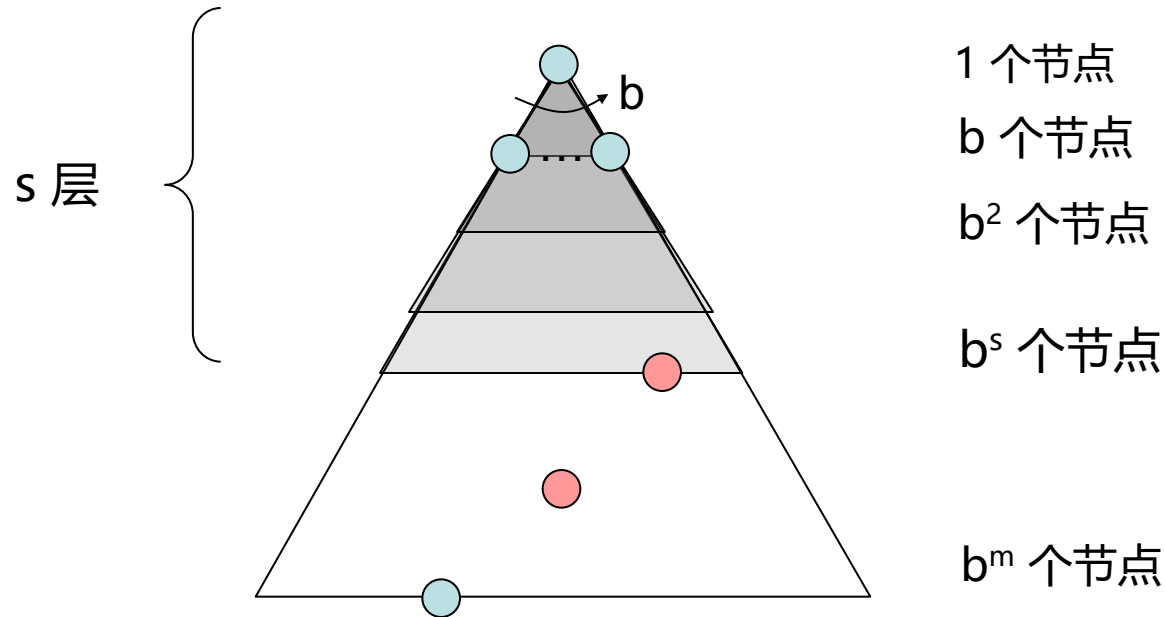
- 最后大致是最后一层, 所以 $O(b^s)$

- 是否完备?

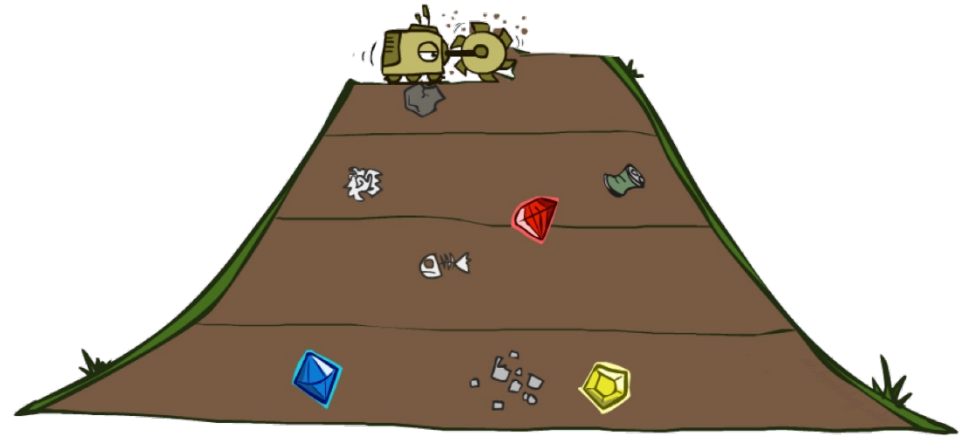
- 若解存在则 s 必有限, 所以完备!

- 是否最优?

- 仅当所有代价为 1 时最优 (代价问题后述)



练习：DFS vs. BFS

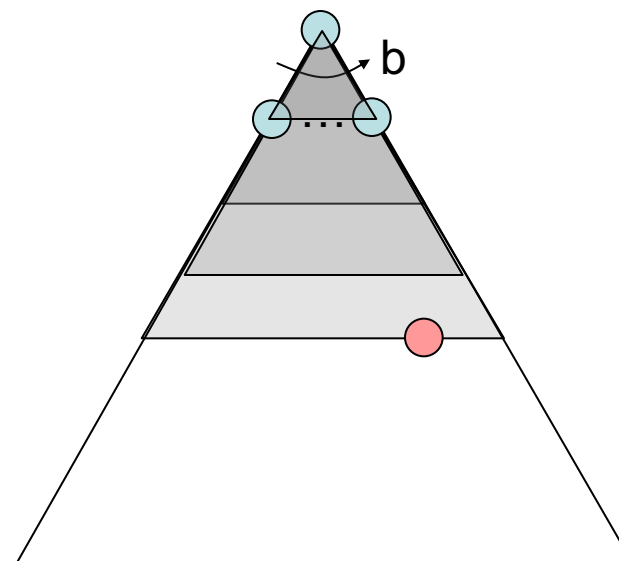


练习：DFS vs. BFS

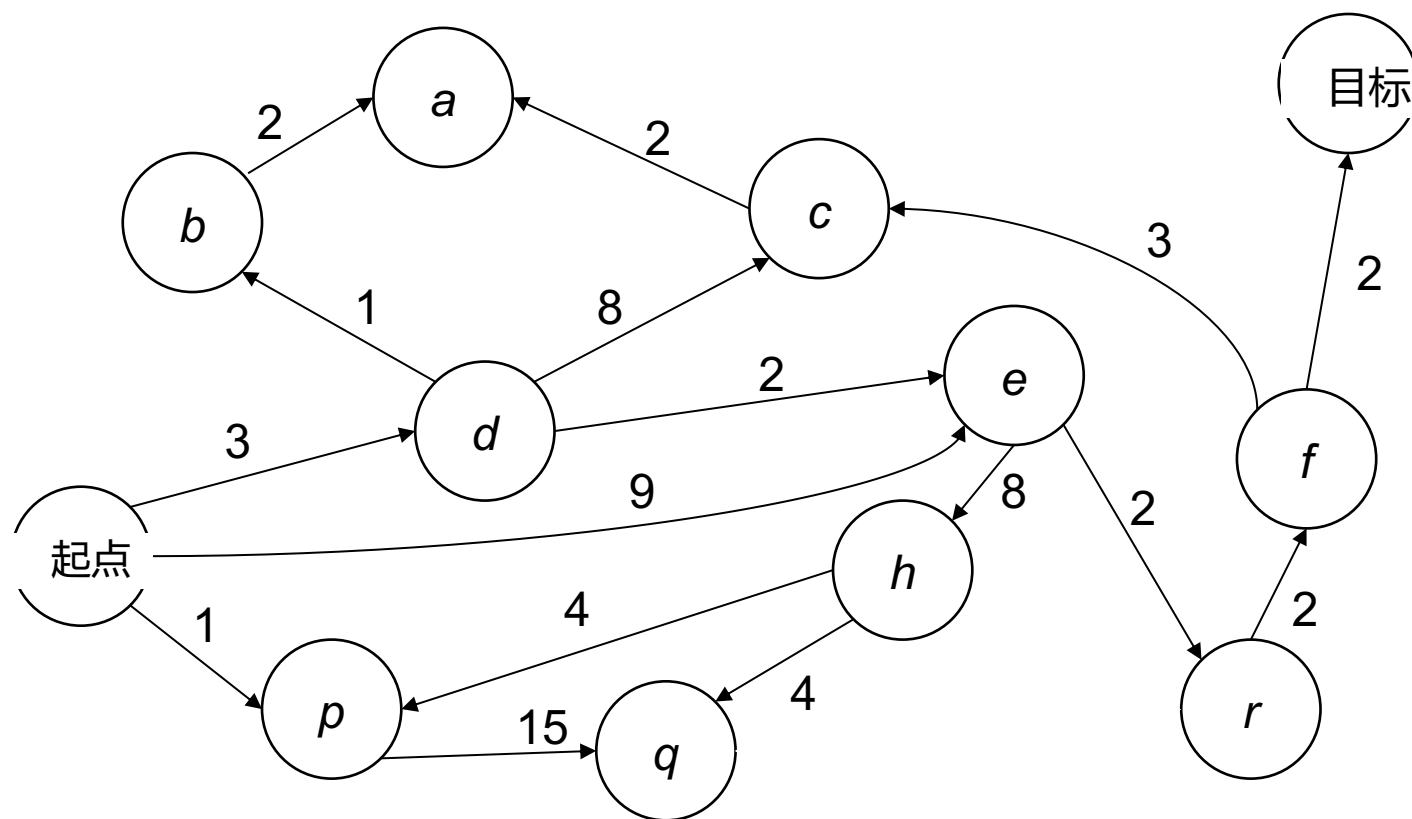
- BFS 何时优于 DFS?
- DFS 何时优于 BFS?

迭代加深

- 思路：用 DFS 的空间优势，配合 BFS 的时间 / 最浅解优势
 - 运行深度限制为 1 的 DFS。若无解.....
 - 运行深度限制为 2 的 DFS。若无解.....
 - 运行深度限制为 3 的 DFS。
- 这样重复不就浪费了吗？
 - 通常大部分工作发生在搜索的最深层，所以其实不亏！

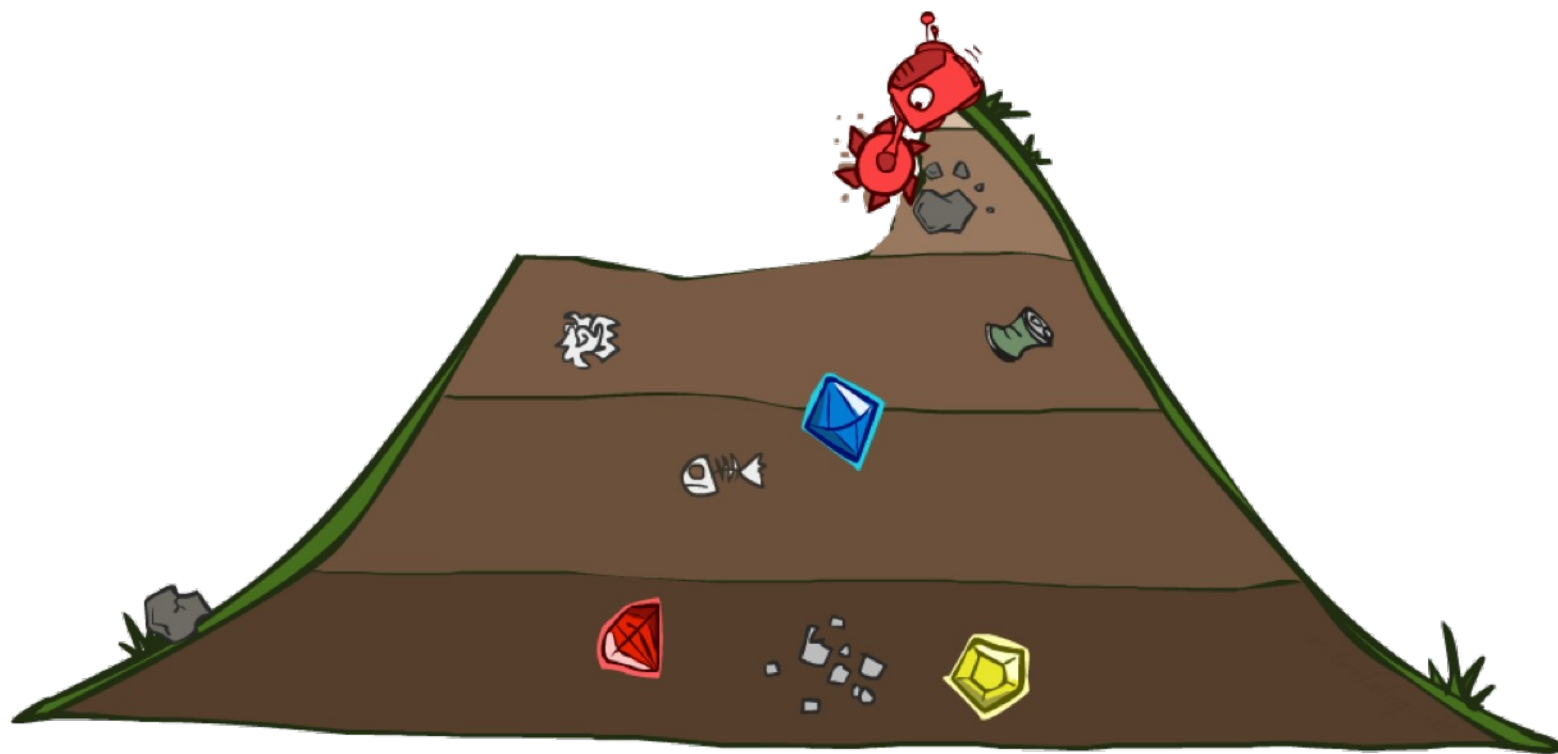


代价敏感的搜索



BFS 找到的是按动作数计的最短路径，而不是代价最小的路径。
下面我们将介绍一个能找到最小代价路径的类似算法。

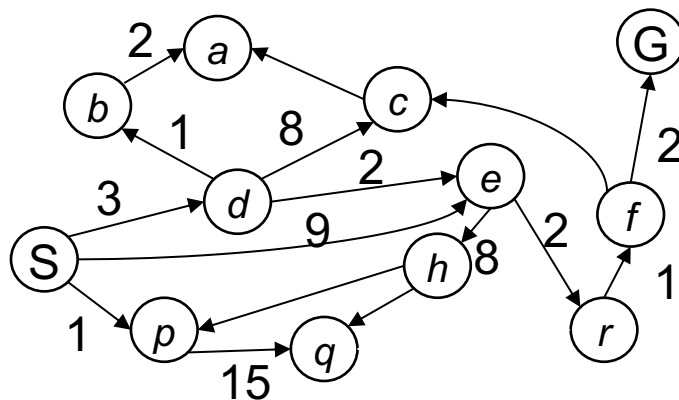
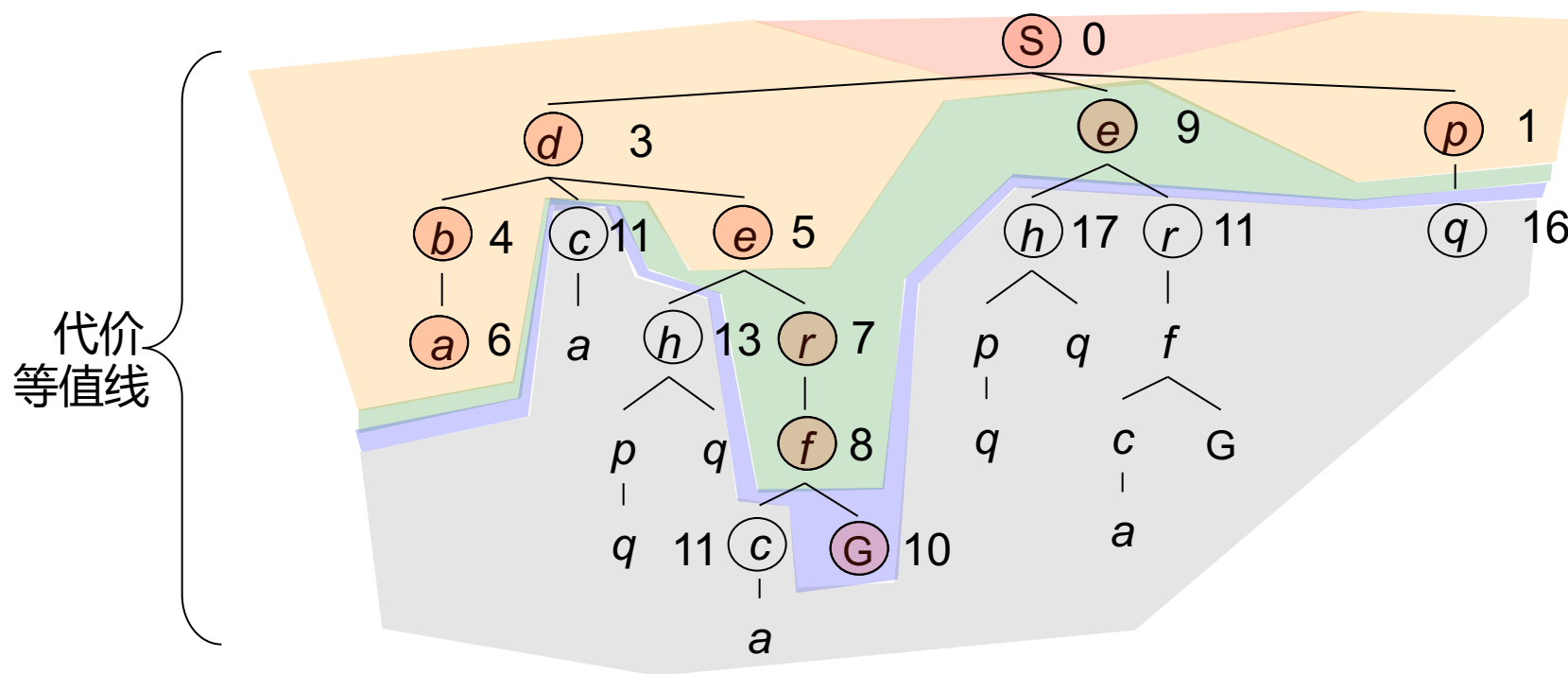
一致代价搜索



一致代价搜索

策略：优先展开
代价最低的节点：

边缘是优先队列
(优先级：累积代价；
树中的数字表示
累积代价)



一致代价搜索 (UCS) 的性质

- UCS 会展开哪些节点?

- 处理所有代价小于最便宜解的节点!
- 若该解代价为 C^* , 且每条弧的代价至少为 ϵ , 则 “有效深度” 约为 C^*/ϵ
- 耗时 $O(b^{C^*/\epsilon})$ (关于有效深度呈指数级)

- 边缘占用多少空间?

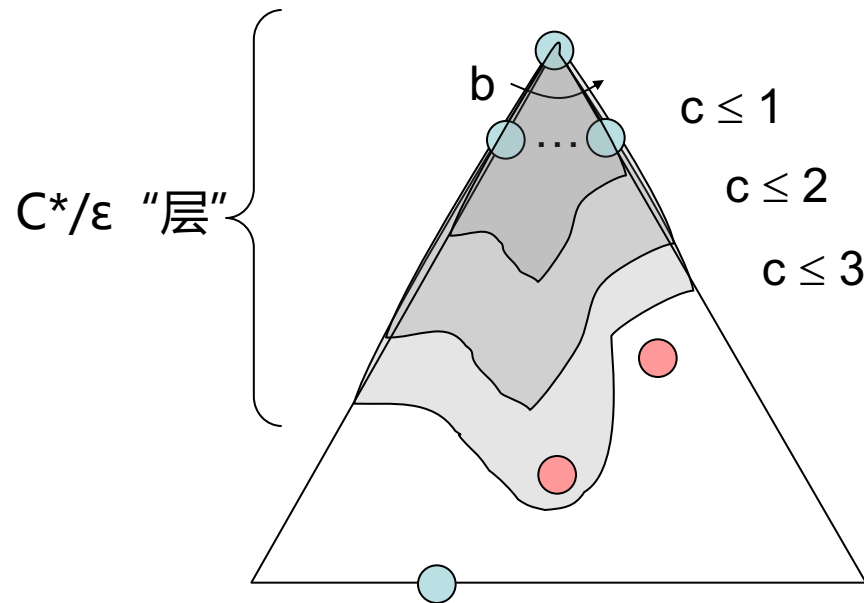
- 最后大致是最后一层, 所以 $O(b^{C^*/\epsilon})$

- 是否完备?

- 假设最优解代价有限且最小弧代价为正, 则完备!

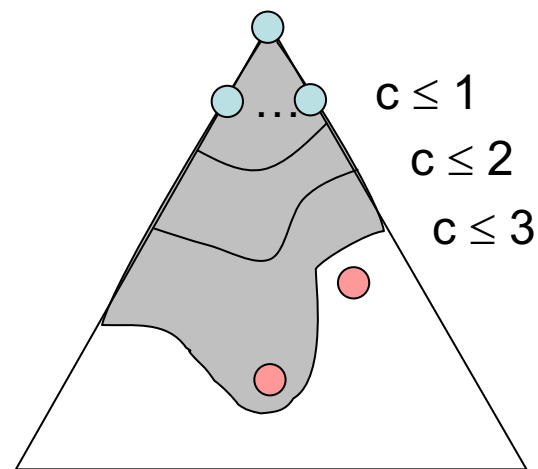
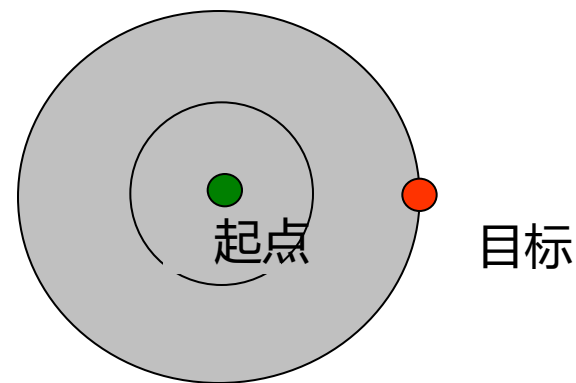
- 是否最优?

- 是的! (证明在下节课用 A^* 给出)



一致代价搜索的问题

- 记住：UCS 按代价递增的等值线扩展
- 好的一面：UCS 完备且最优！
- 坏的一面：
 - 向所有“方向”探索选项
 - 没有关于目标位置的信息
- 我们很快会修复这一点！

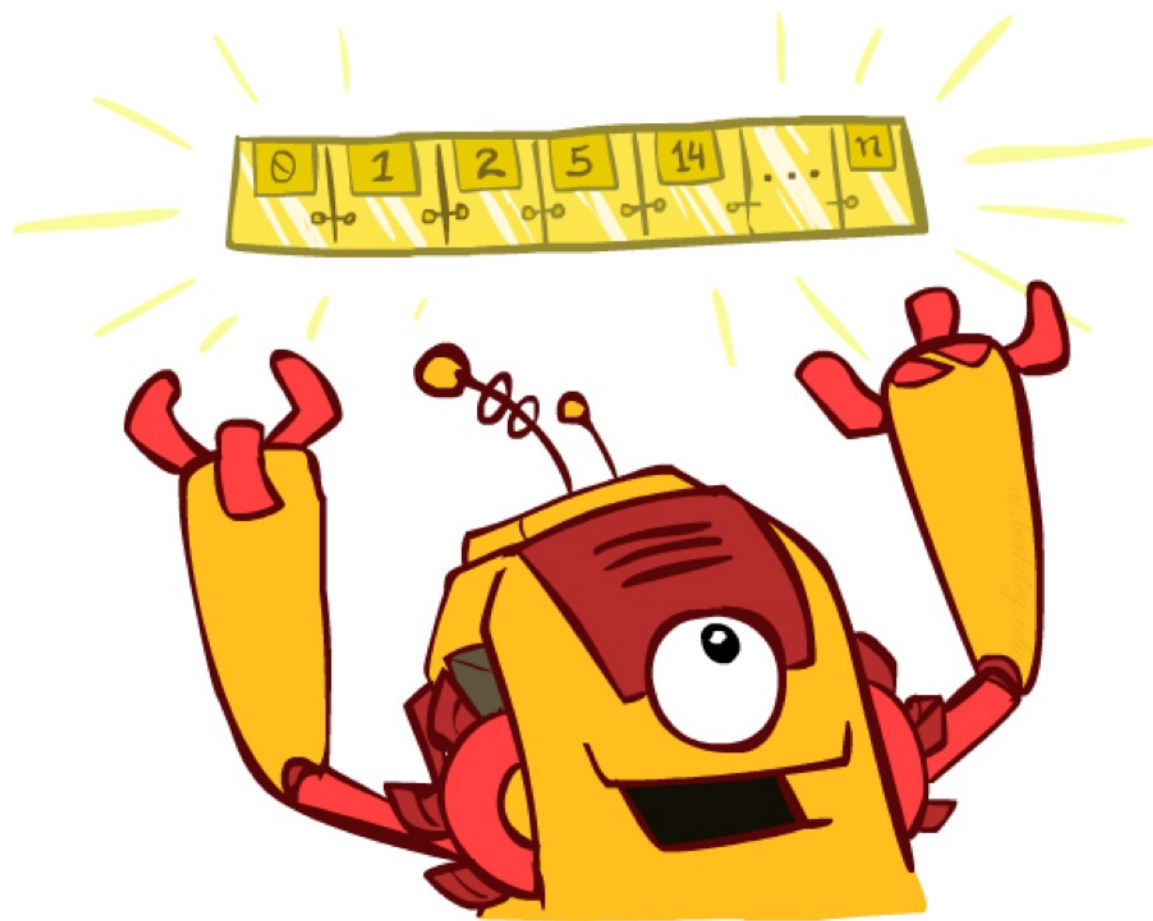


[演示：空网格 UCS (L2D5)]

[演示：深水/浅水迷宫 DFS/BFS/UCS (L2D7)]

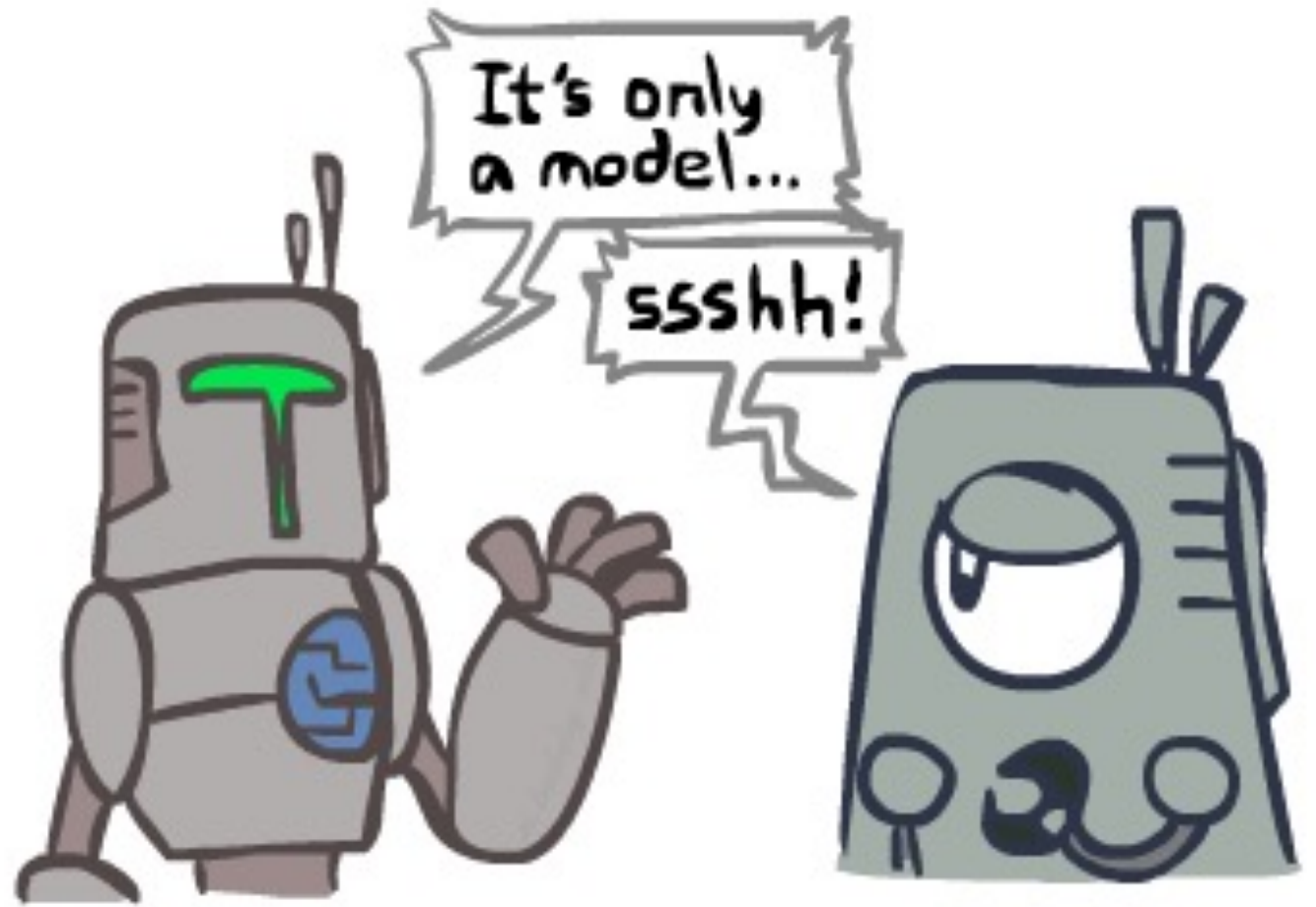
一个队列走天下

- 这些搜索算法的区别只在于边缘策略
 - 概念上，所有边缘都是优先队列（即带优先级的节点集合）
 - 实践中，对 DFS 和 BFS，可以用栈和普通队列来避免真实优先队列的 $\log(n)$ 开销
 - 甚至可以只写一份实现，接受可变的队列对象



搜索与模型

- 搜索在世界的模型上进行
 - 智能体并不会在真实世界中真的尝试所有规划!
 - 规划完全“在模拟中”进行
 - 搜索的效果上限取决于你的模型.....

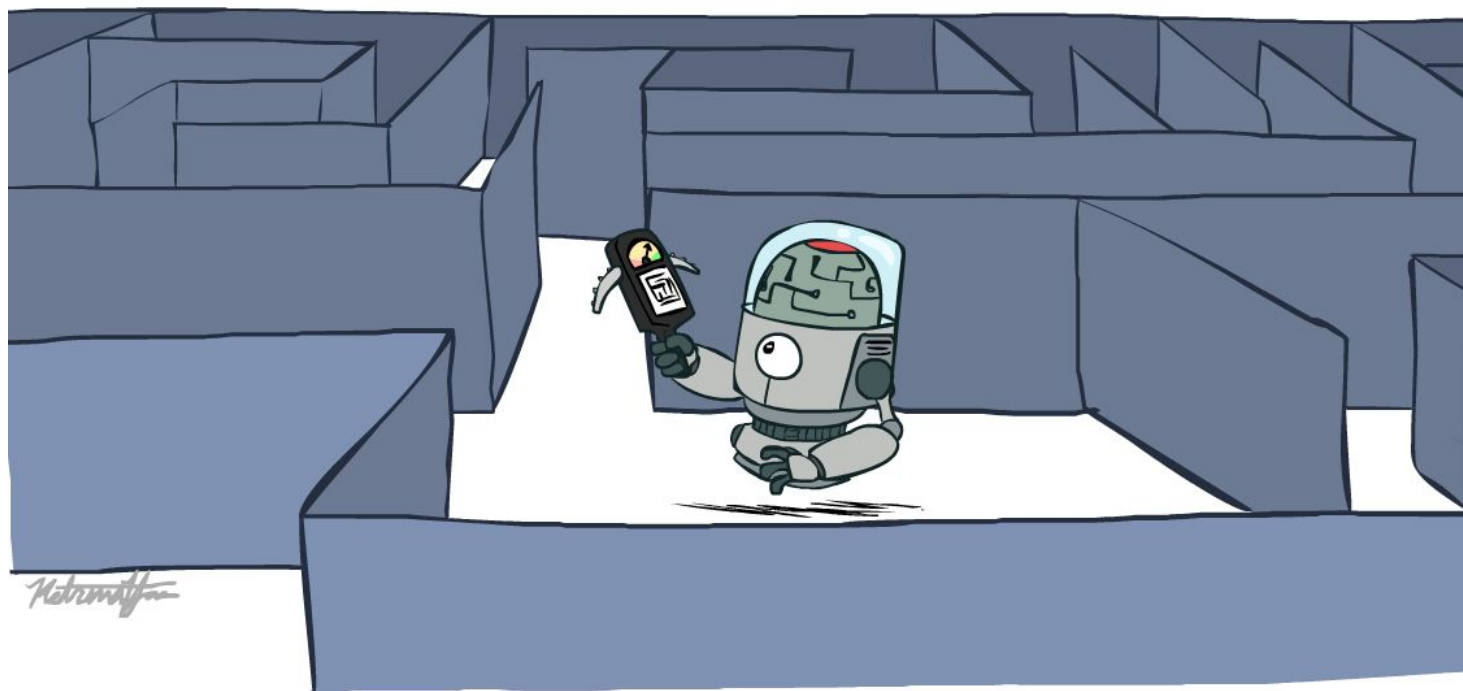


搜索出问题了?



CS 188: 人工智能

有信息搜索



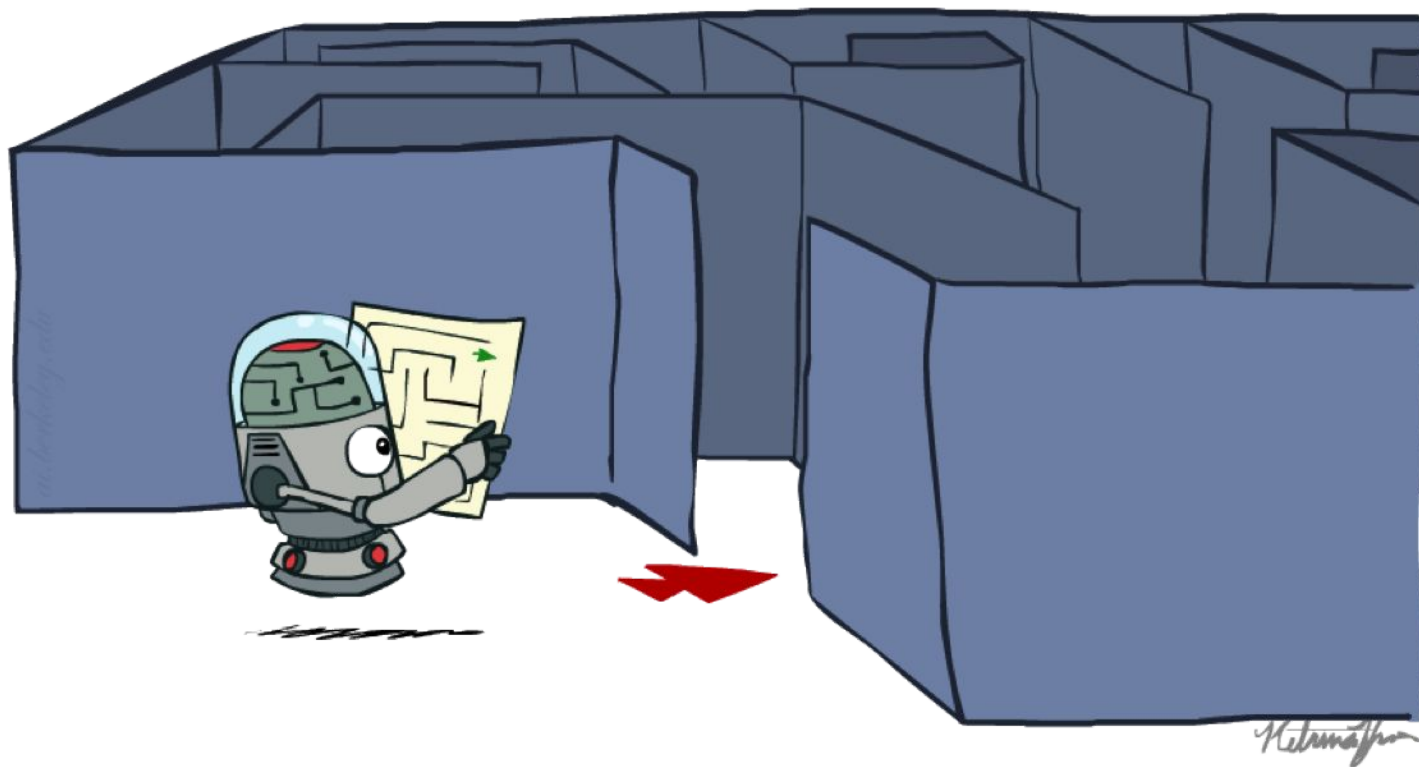
加州大学伯克利分校

本节课内 容

- 有信息搜索
 - 启发式
 - 贪婪搜索
 - A^* 搜索
- 图搜索



回顾：搜索



回顾：搜索

- 搜索问题：

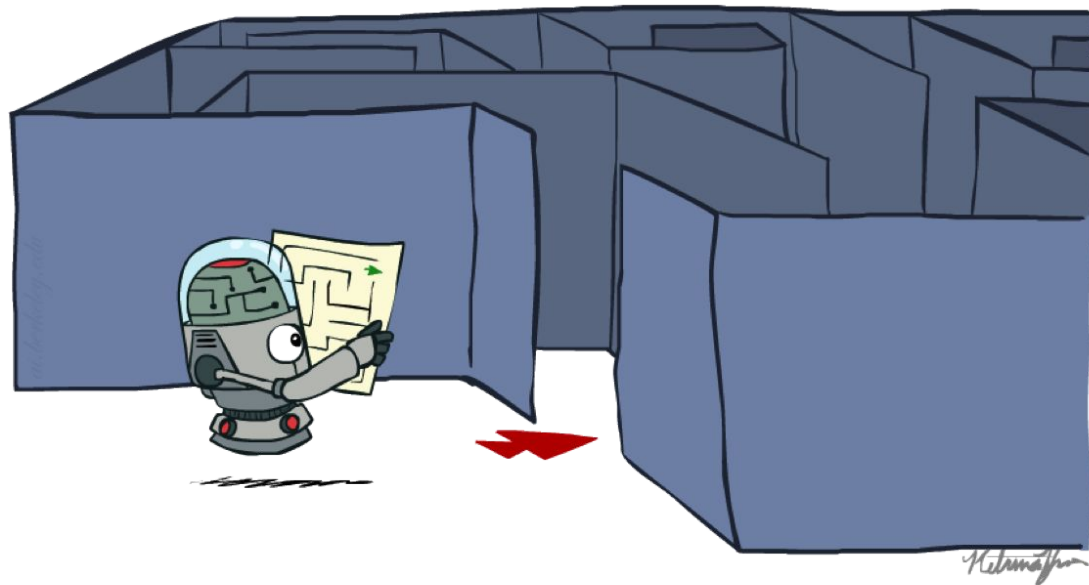
- 状态（世界的配置）
- 动作与代价
- 后继函数（转移动态）
- 初始状态与目标检验

- 搜索树：

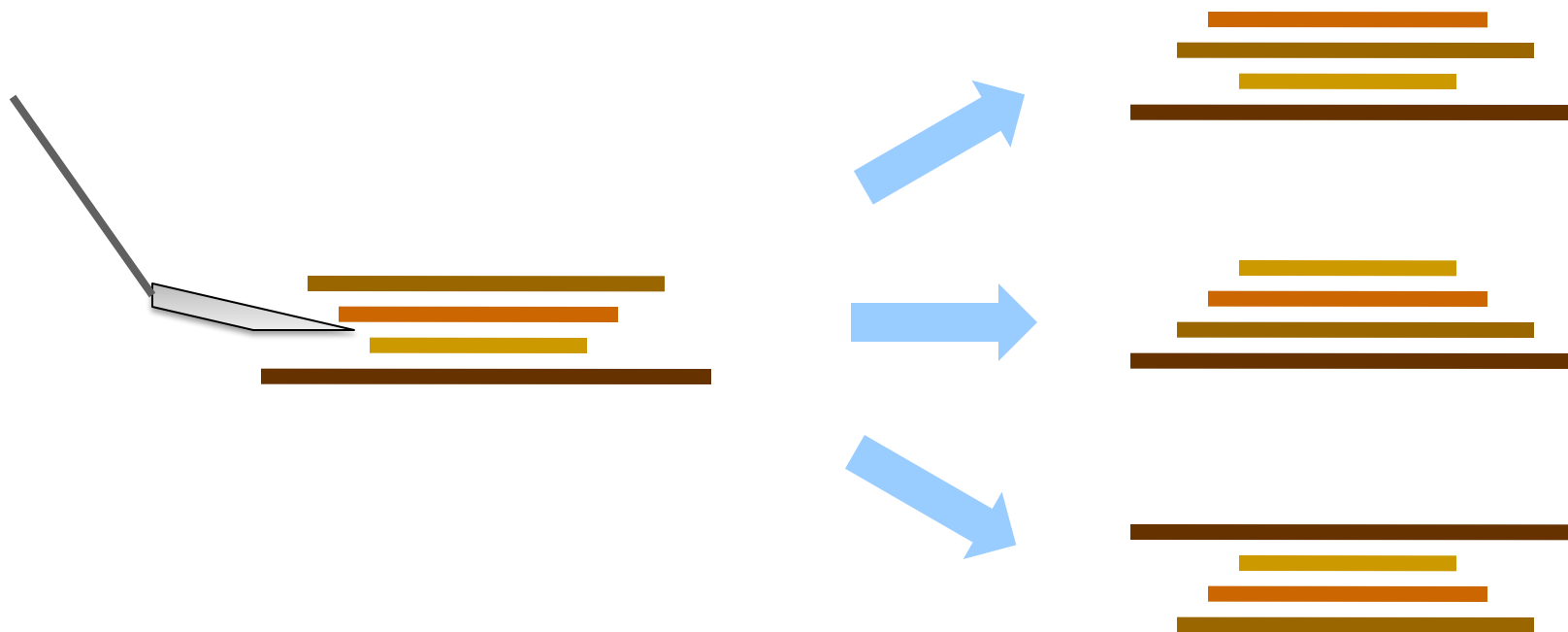
- 节点：表示到达状态的规划
- 规划有代价（动作代价之和）

- 搜索算法：

- 系统地构建搜索树
- 选择边缘（未探索节点）的排序方式
- 最优：找到代价最小的规划



例子：煎饼问题



代价：翻转煎饼的次数

例子：煎饼问题

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES

Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†

Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

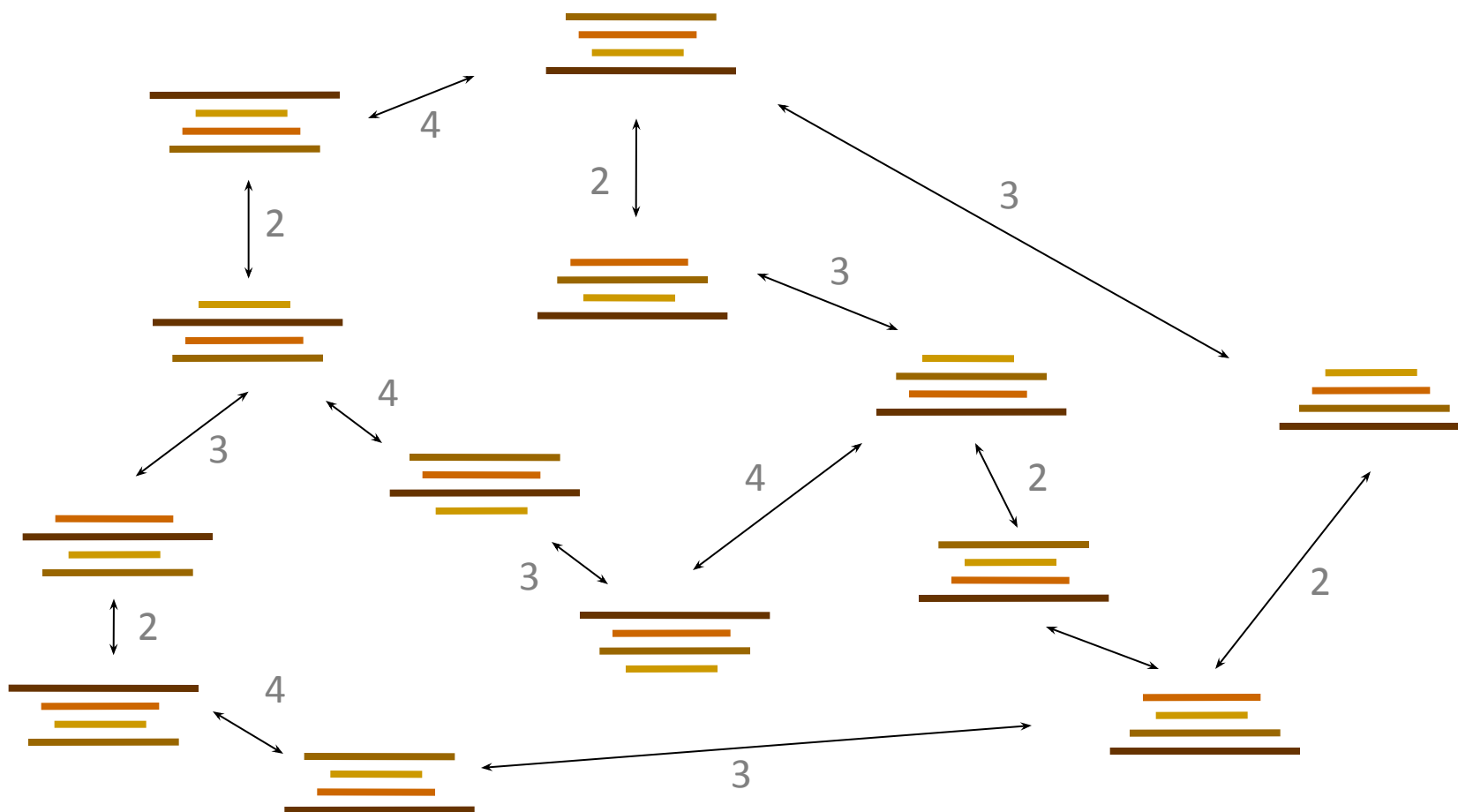
Received 18 January 1978

Revised 28 August 1978

For a permutation σ of the integers from 1 to n , let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n . We show that $f(n) \leq (5n+5)/3$, and that $f(n) \geq 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n + 3$.

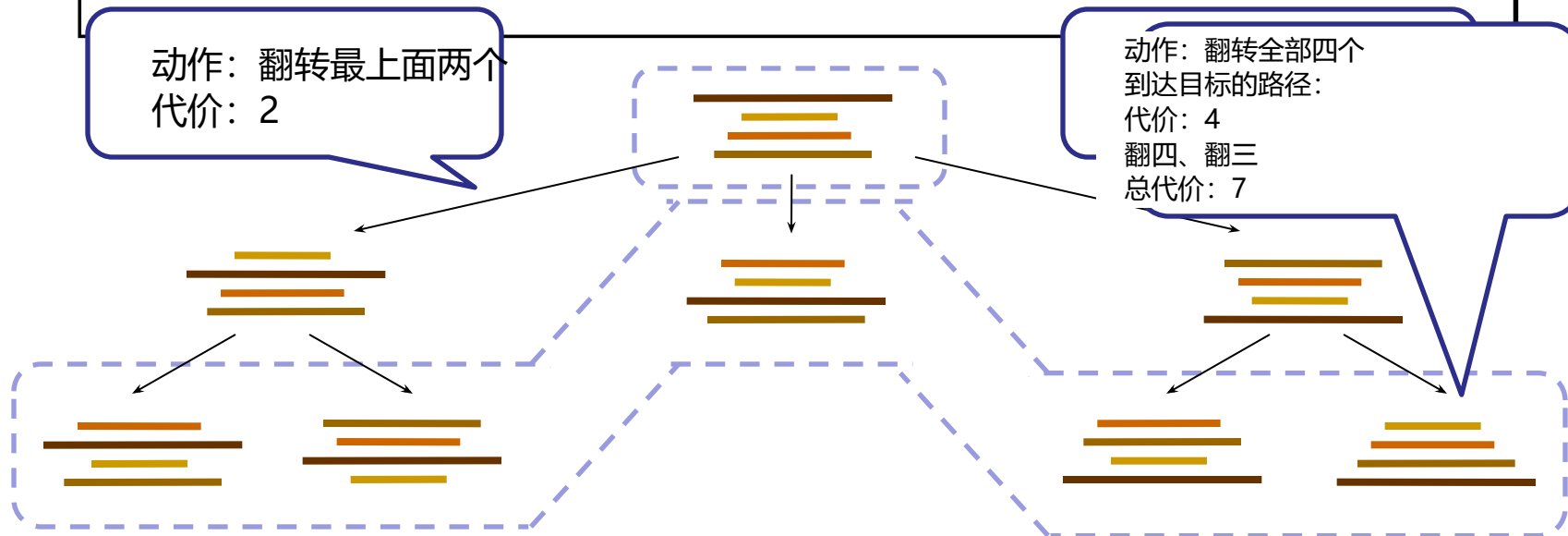
例子：煎饼问题

以代价为边权的状态空间图



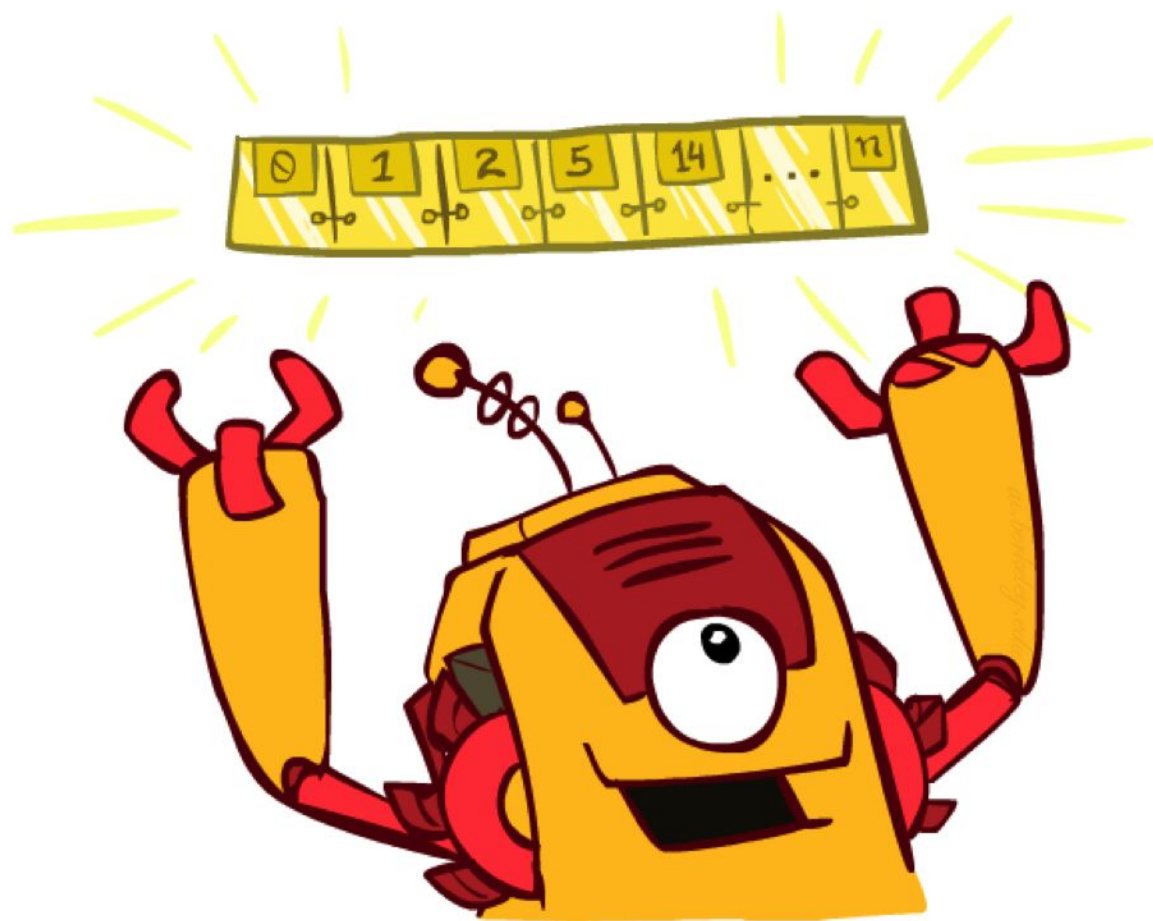
通用树搜索

```
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
```



一个队列走天下

- 这些搜索算法的区别只在于扩展策略
 - 概念上，所有边缘都是优先队列（即带优先级的节点集合）
 - 实践中，对 DFS 和 BFS，可以用栈和队列来避免真实优先队列的 $\log(n)$ 开销
 - 甚至可以只写一份实现，接受可变的队列对象



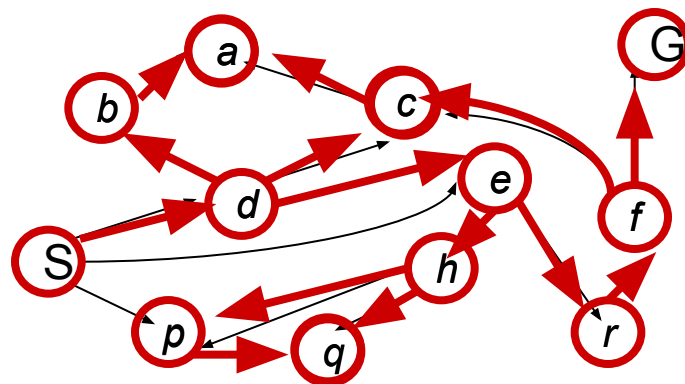
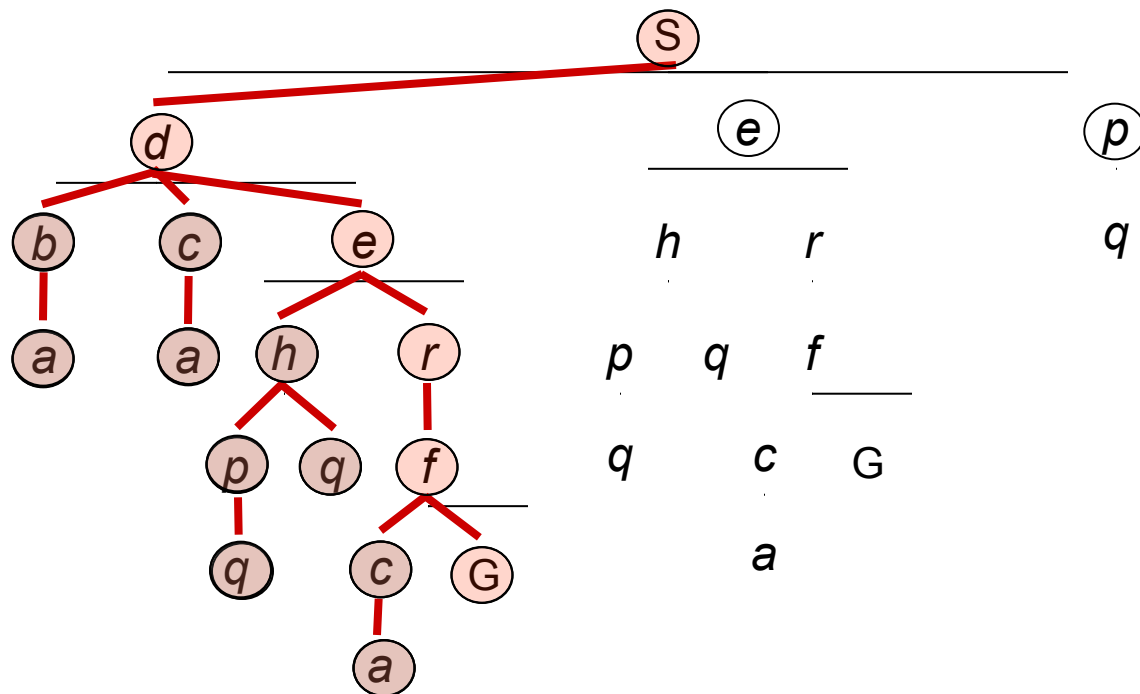
无信息搜索



深度优先搜索

策略：优先展开
最深的节点

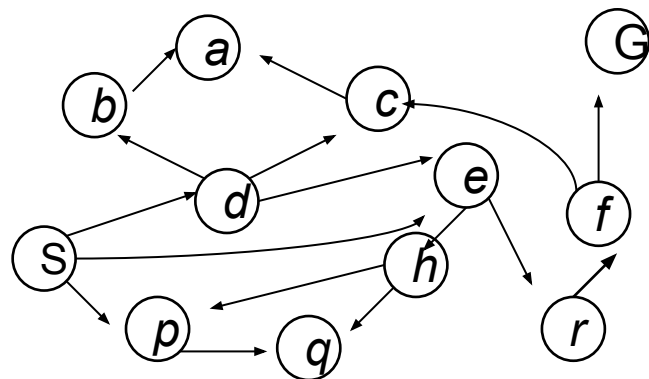
实现：
边缘是 LIFO 栈



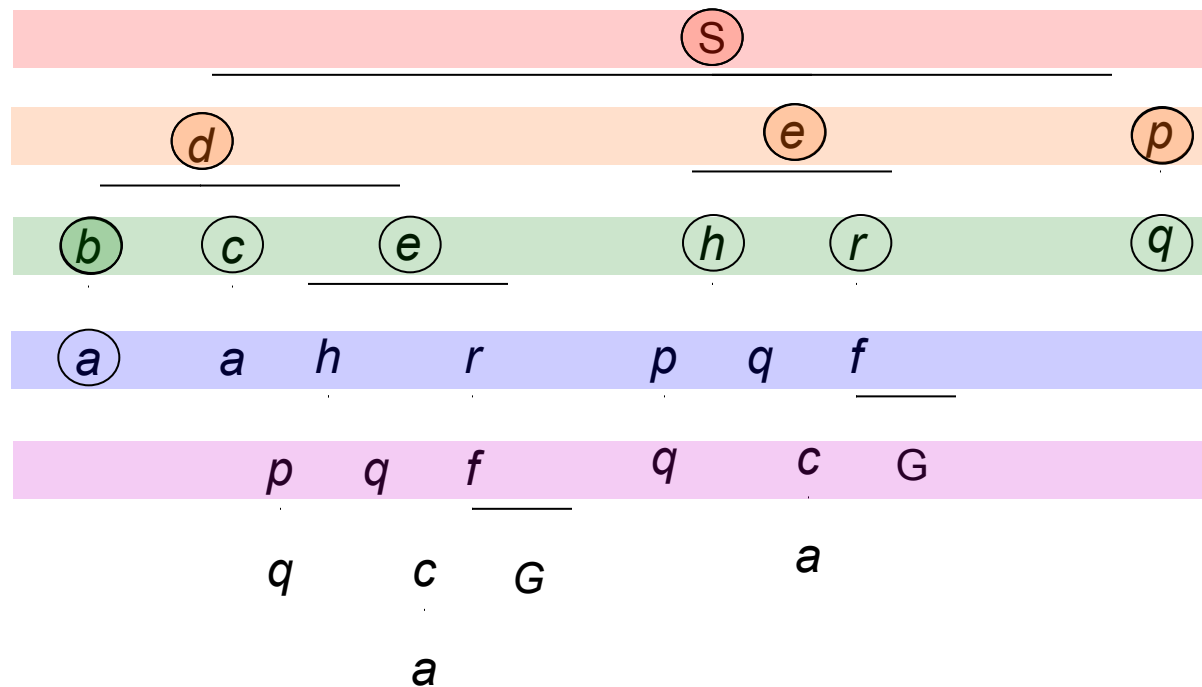
广度优先搜索

策略：优先展开
最浅的节点

实现：边缘
是 FIFO 队列

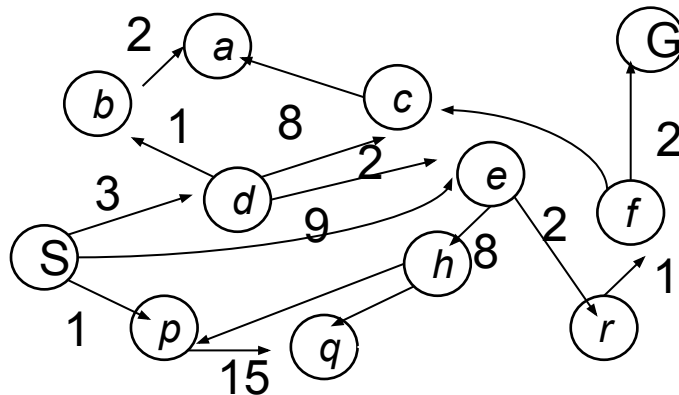
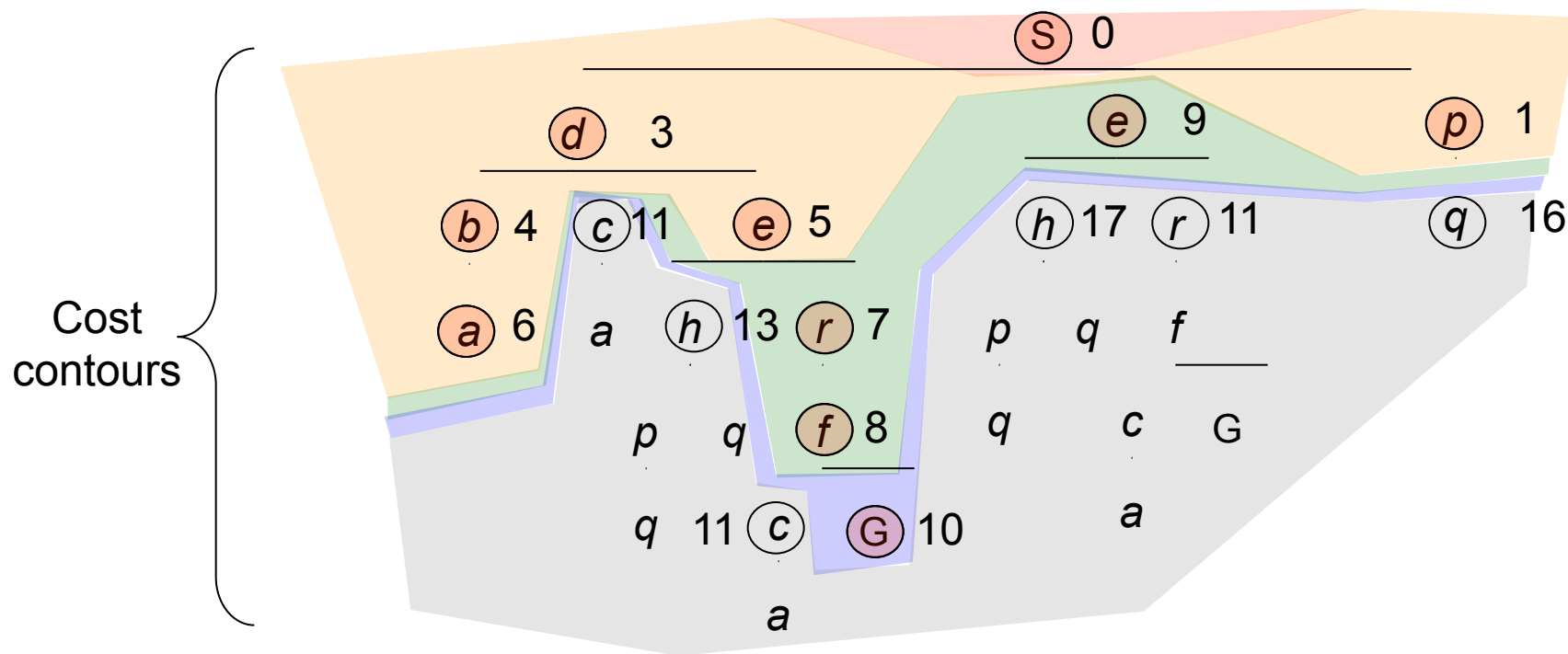


搜索
层



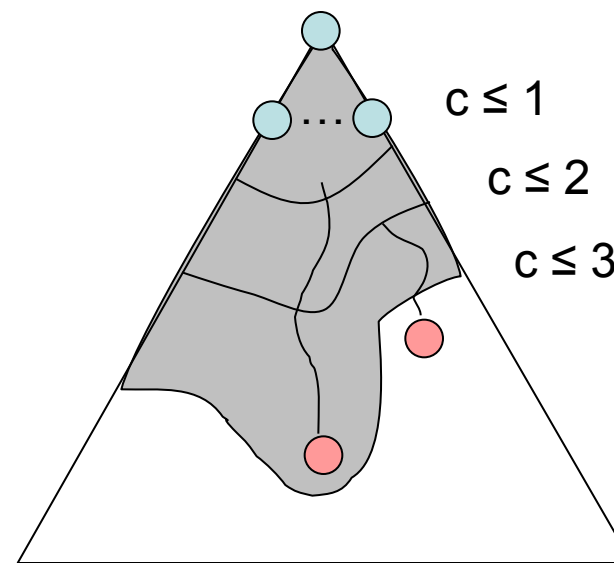
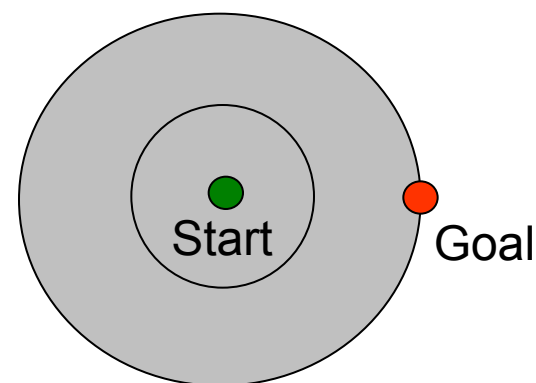
一致代价搜索

策略：优先展开
代价最低的节点：
边缘是优先队列
(优先级：累积代价；
树中的数字表示
累积代价)



一致代价搜索

- 策略：按路径代价最低扩展
- 好的一面：UCS 完备且最优！
- 坏的一面：
 - 向所有“方向”探索选项
 - 没有关于目标位置的信息



[演示：等值线 UCS 空网格 (L3D1)]

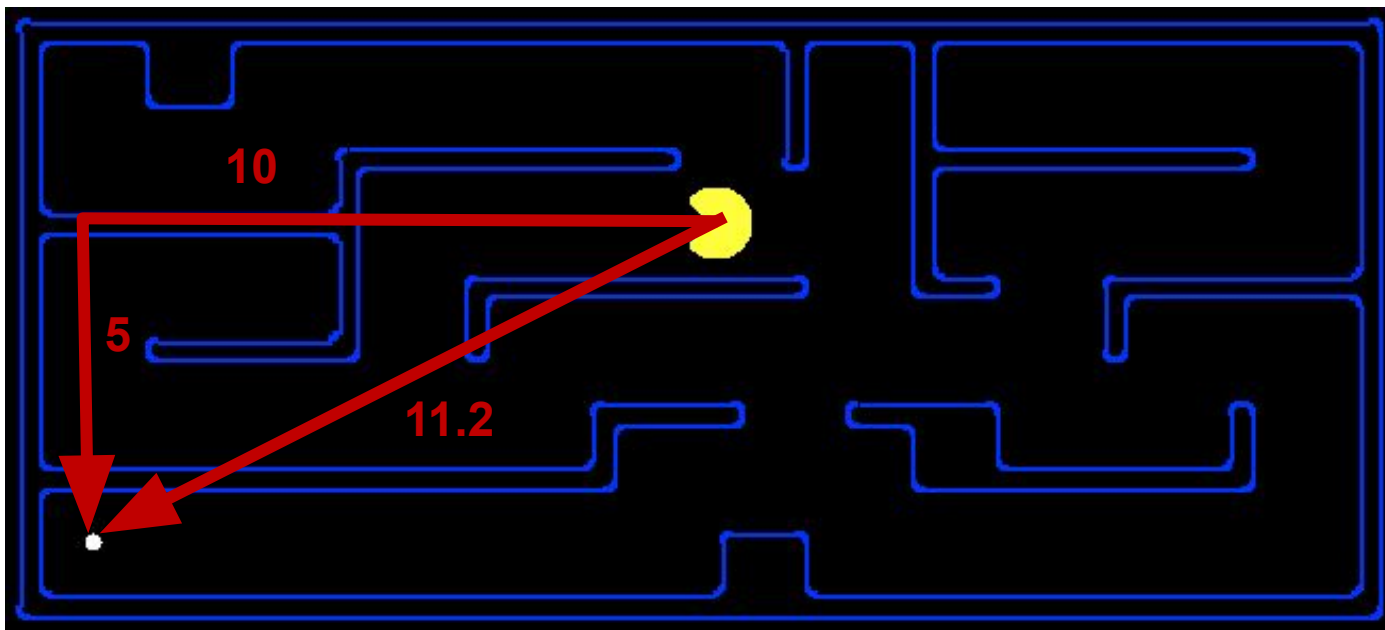
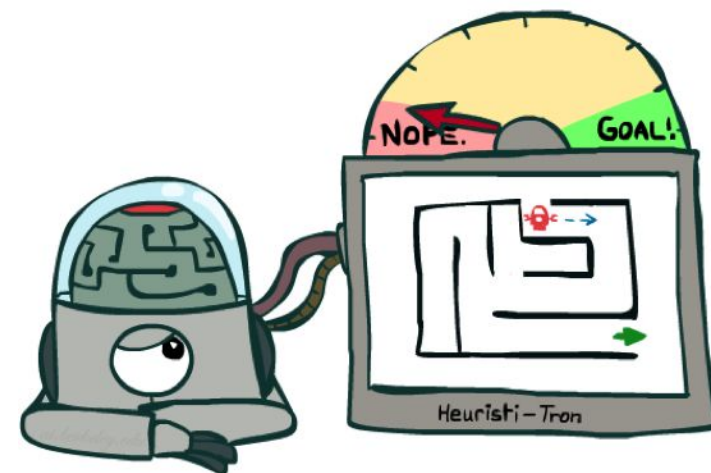
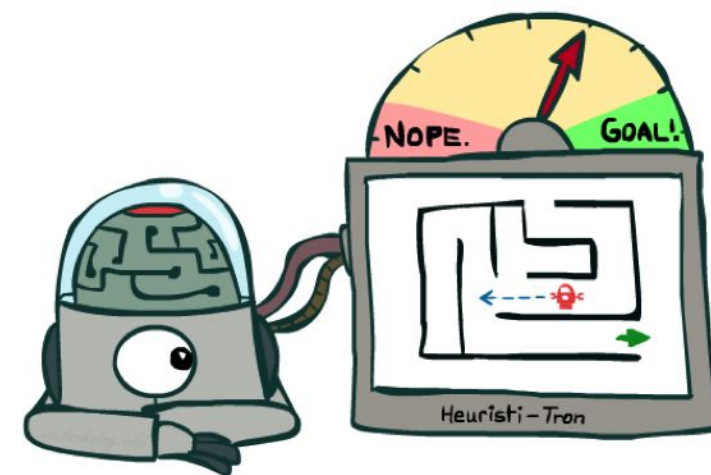
[演示：等值线 UCS 吃豆人小迷宫 (L3D3)]

有信息搜索

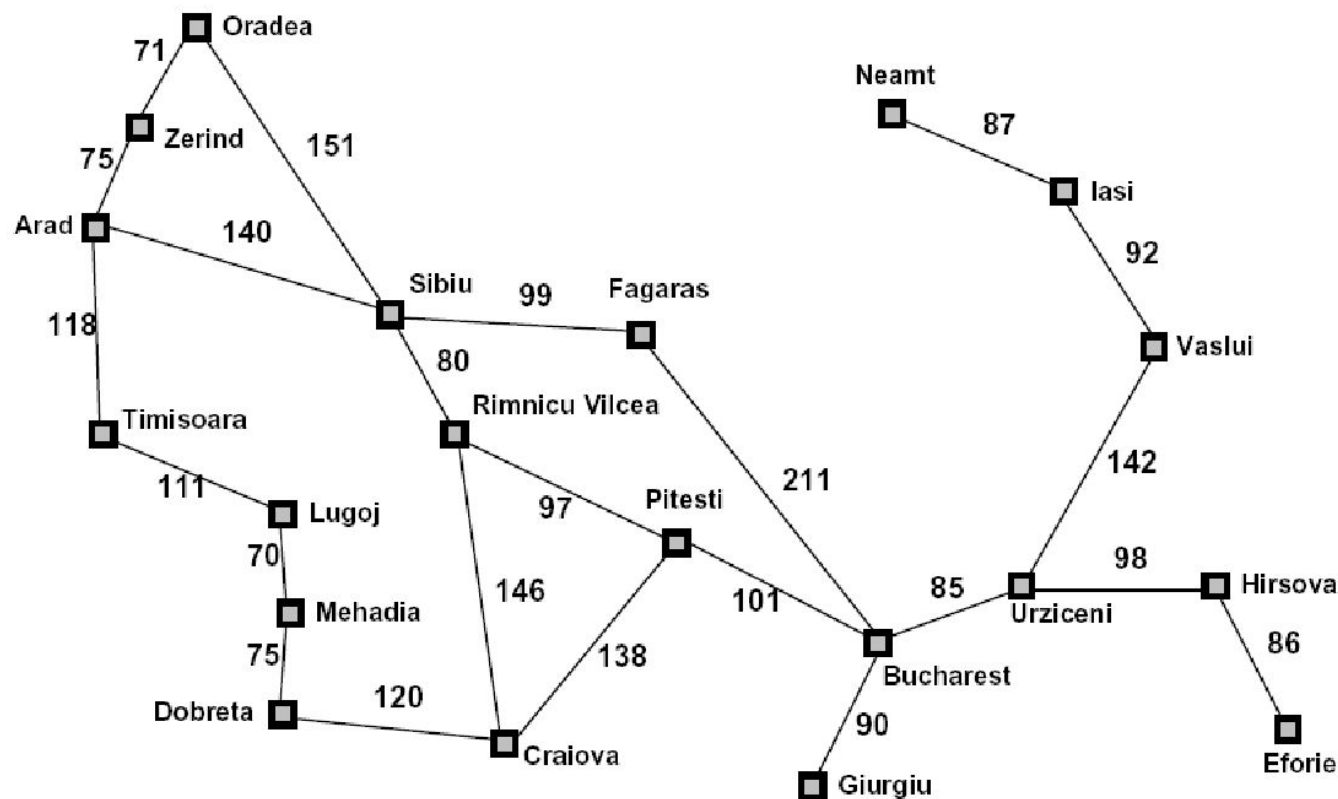


搜索启发式

- 启发式 (heuristic) 是：
 - 一个估计某状态离目标有多近的函数
 - 针对特定的搜索问题而设计
 - 例子：寻路中的曼哈顿距离、欧几里得距离



例子：启发式函数



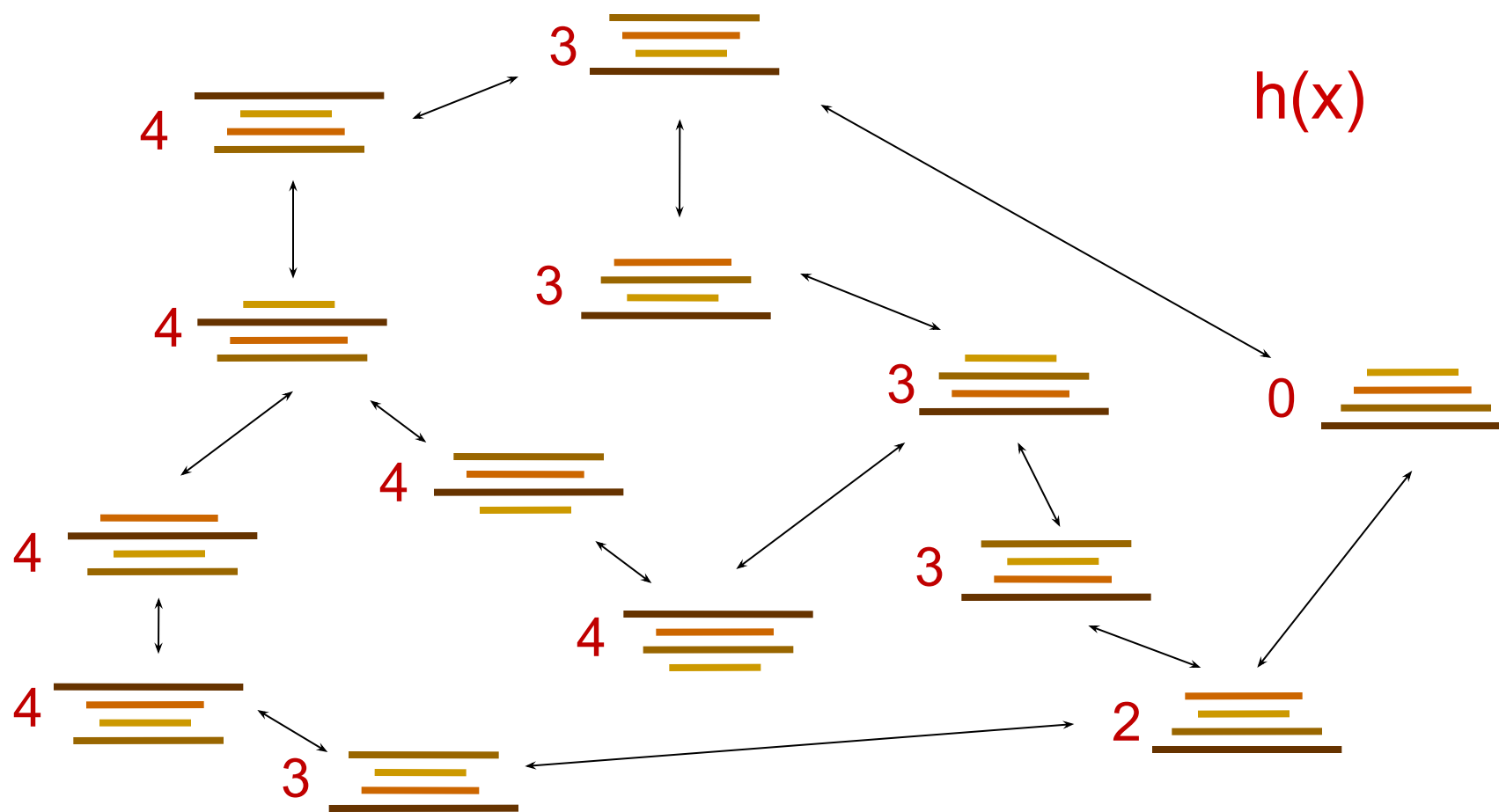
Straight-line distance
to Bucharest

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

$h(x)$

例子：启发式函数

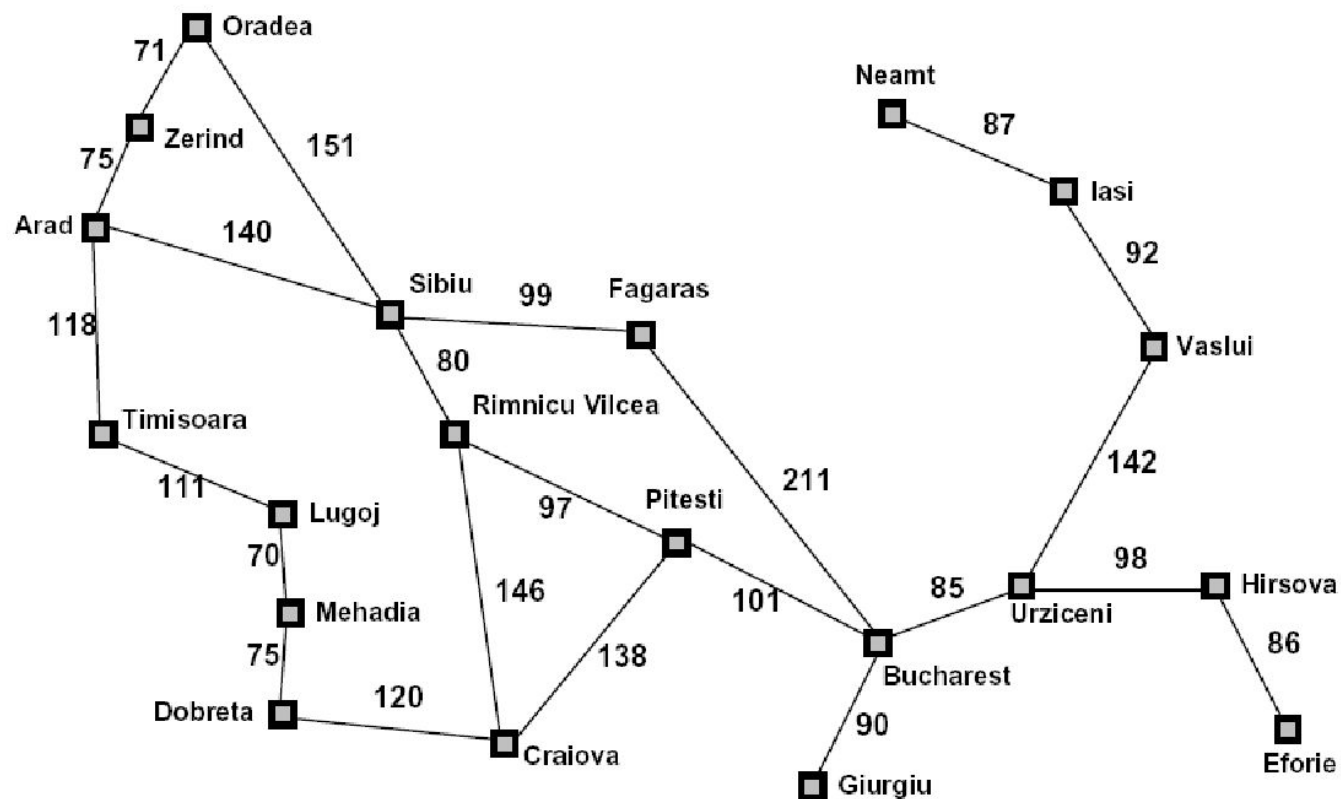
启发式：仍未归位的最大煎饼的编号



贪婪搜索



例子：启发式函数



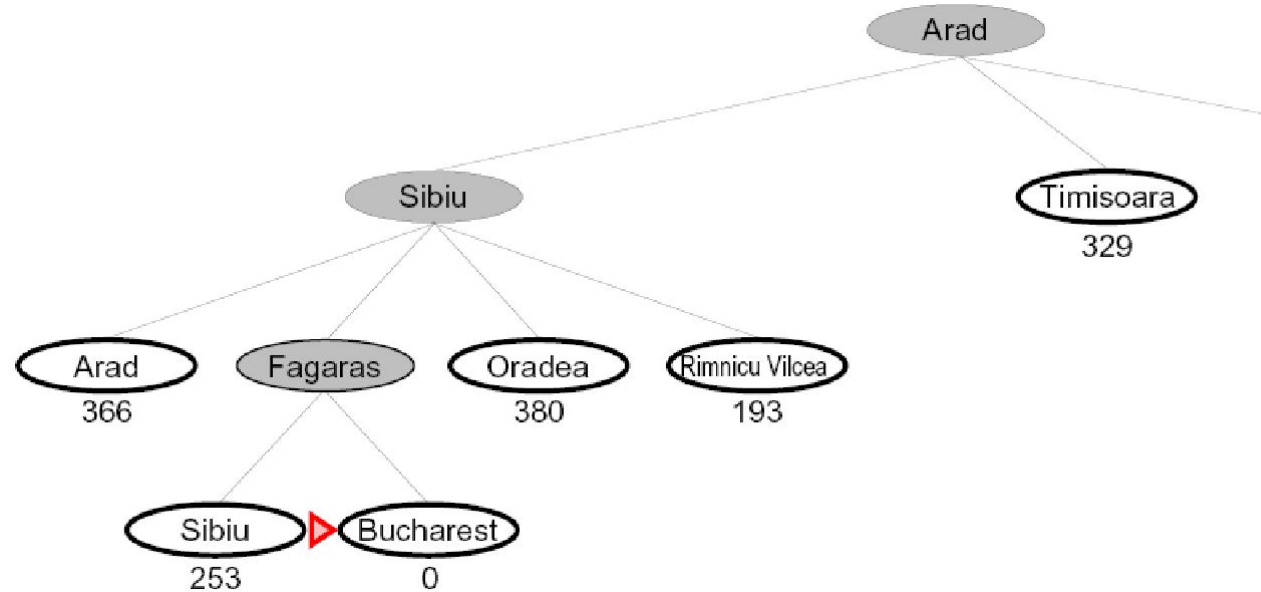
Straight-line distance
to Bucharest

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

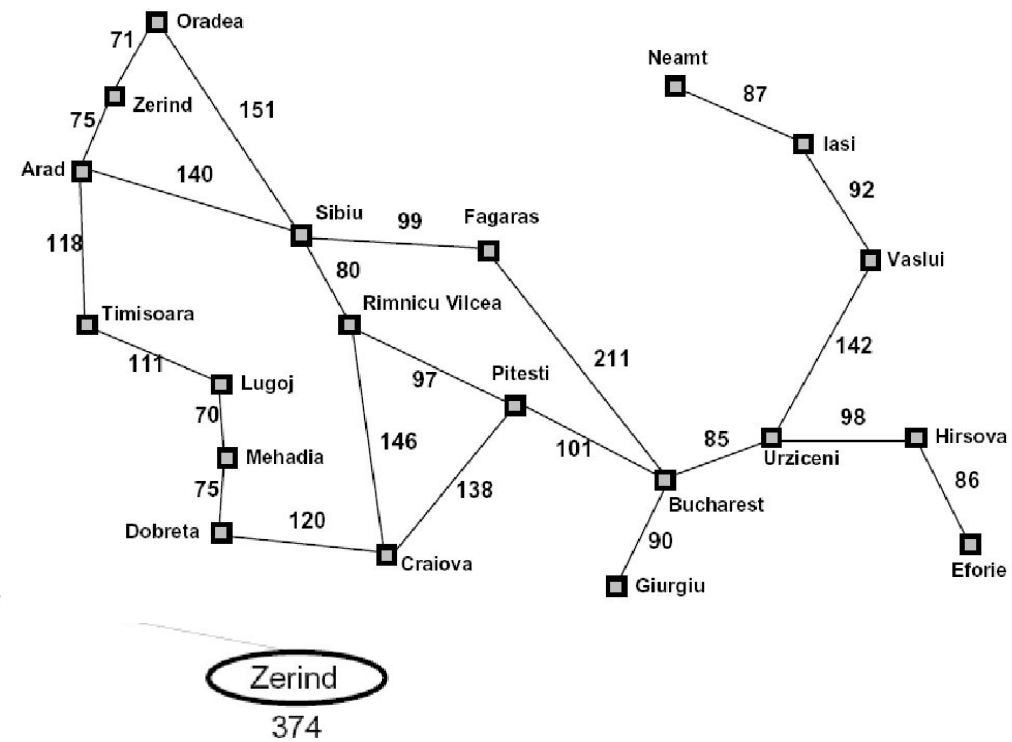
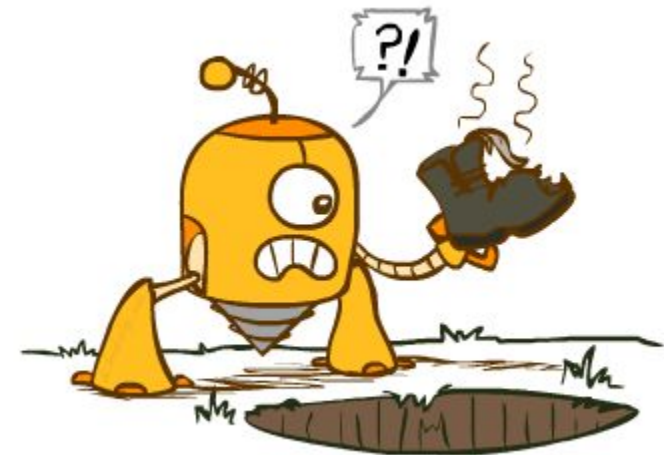
$h(x)$

贪婪搜索

- 展开看起来离目标最近的节点.....

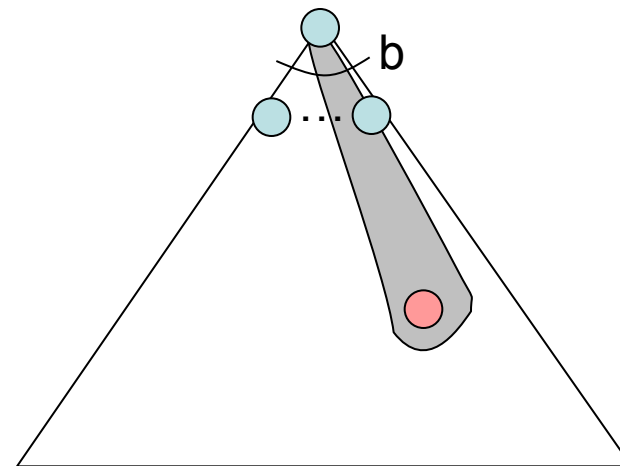
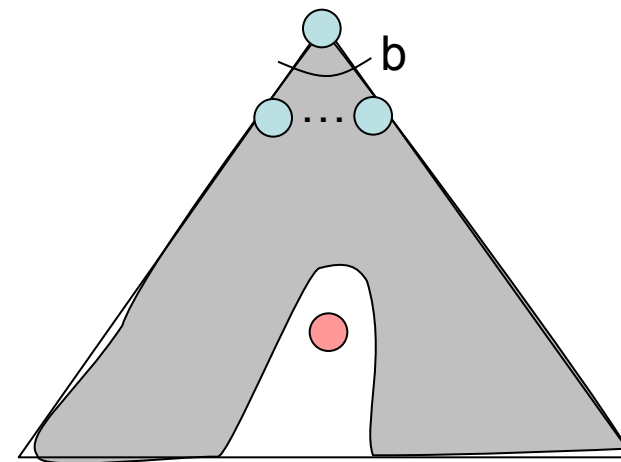


- 可能出什么问题？



贪婪搜索

- 策略：展开你认为离目标状态最近的节点
 - 启发式：对每个状态到最近目标距离的估计
- 常见情形：
 - 最佳优先会把你直接带到（次优的）目标
- 最坏情形：就像引导很差的 DFS



[演示：等值线 贪婪 空网格 (L3D1)]
[演示：等值线 贪婪 吃豆人小迷宫 (L3D4)]

A* 搜索



A* 搜索



UCS



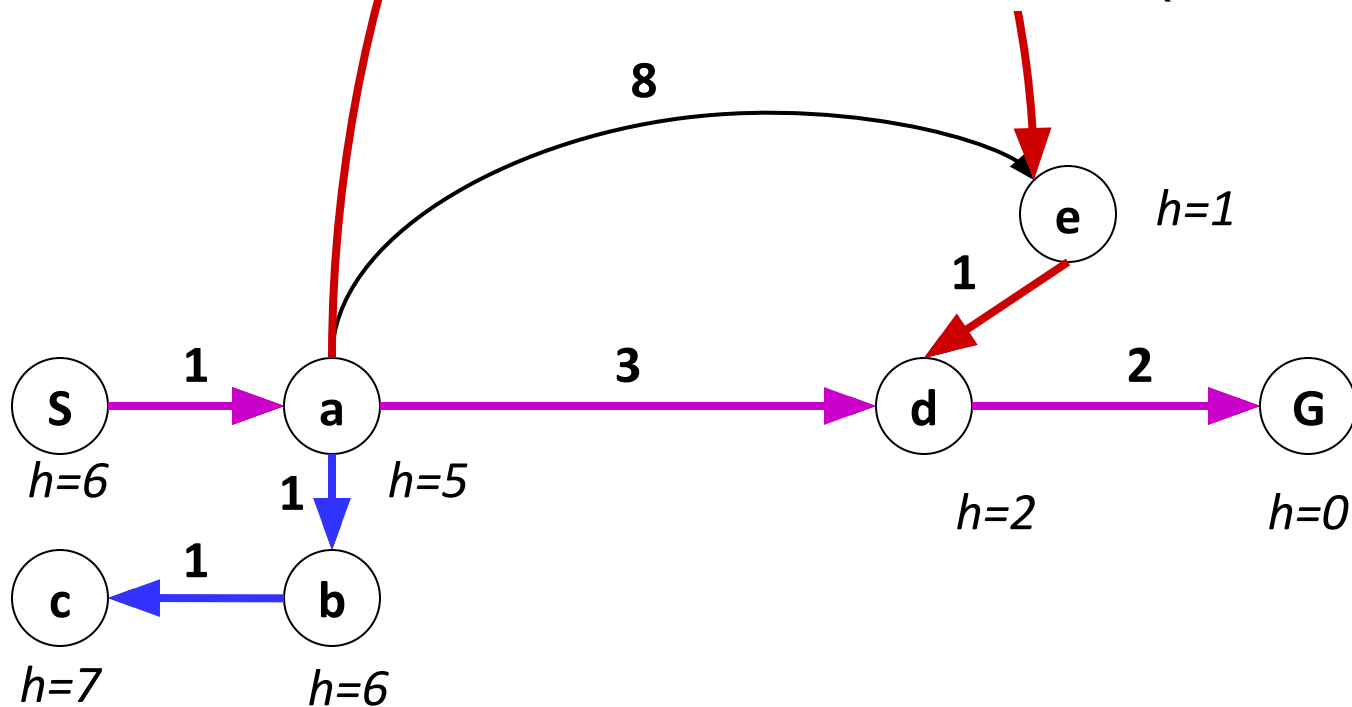
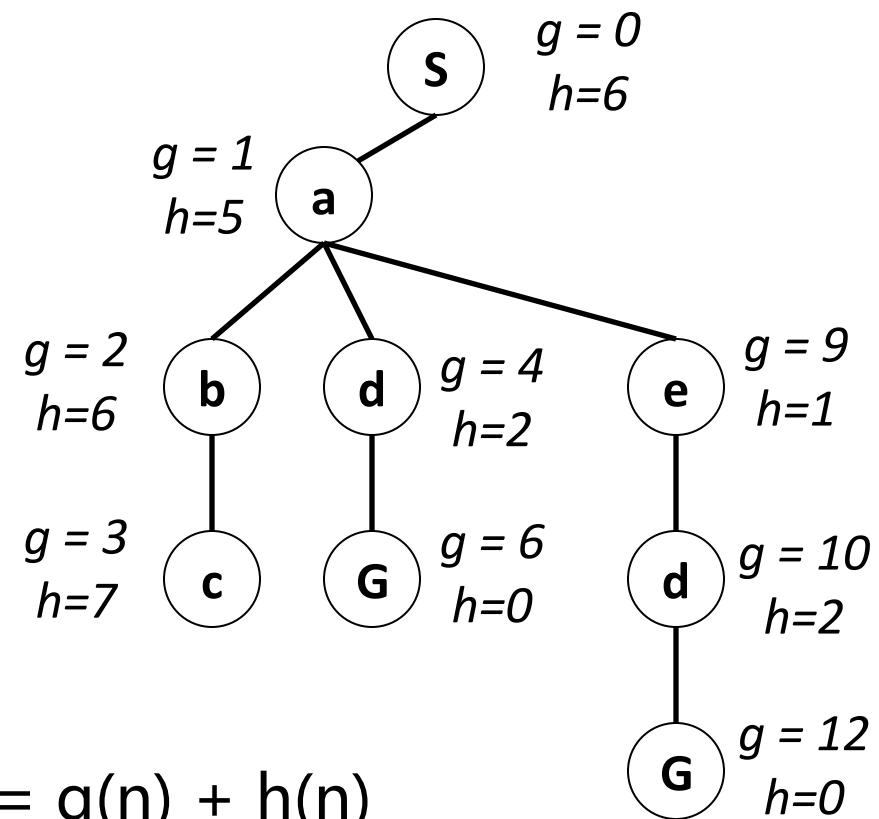
Greedy



A*

结合 UCS 与贪婪搜索

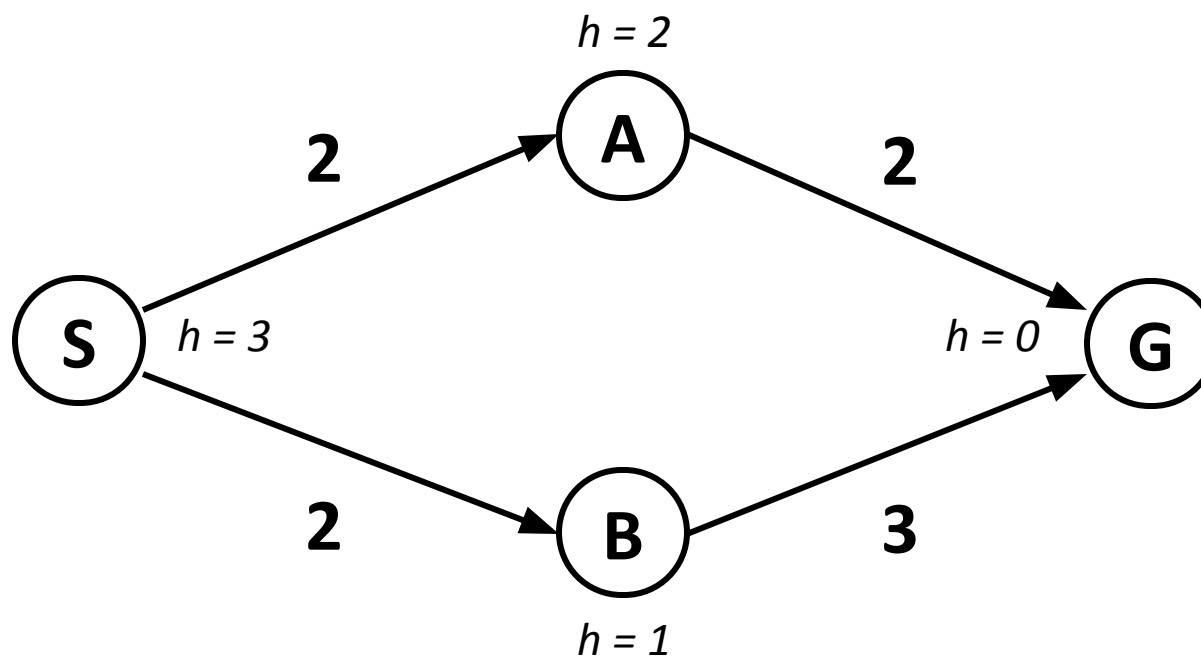
- 一致代价按路径代价（后向代价 $g(n)$ ）排序
- 贪婪按到目标的接近程度（前向代价 $h(n)$ ）排序



- A* 搜索按两者之和（总代价）排序: $f(n) = g(n) + h(n)$

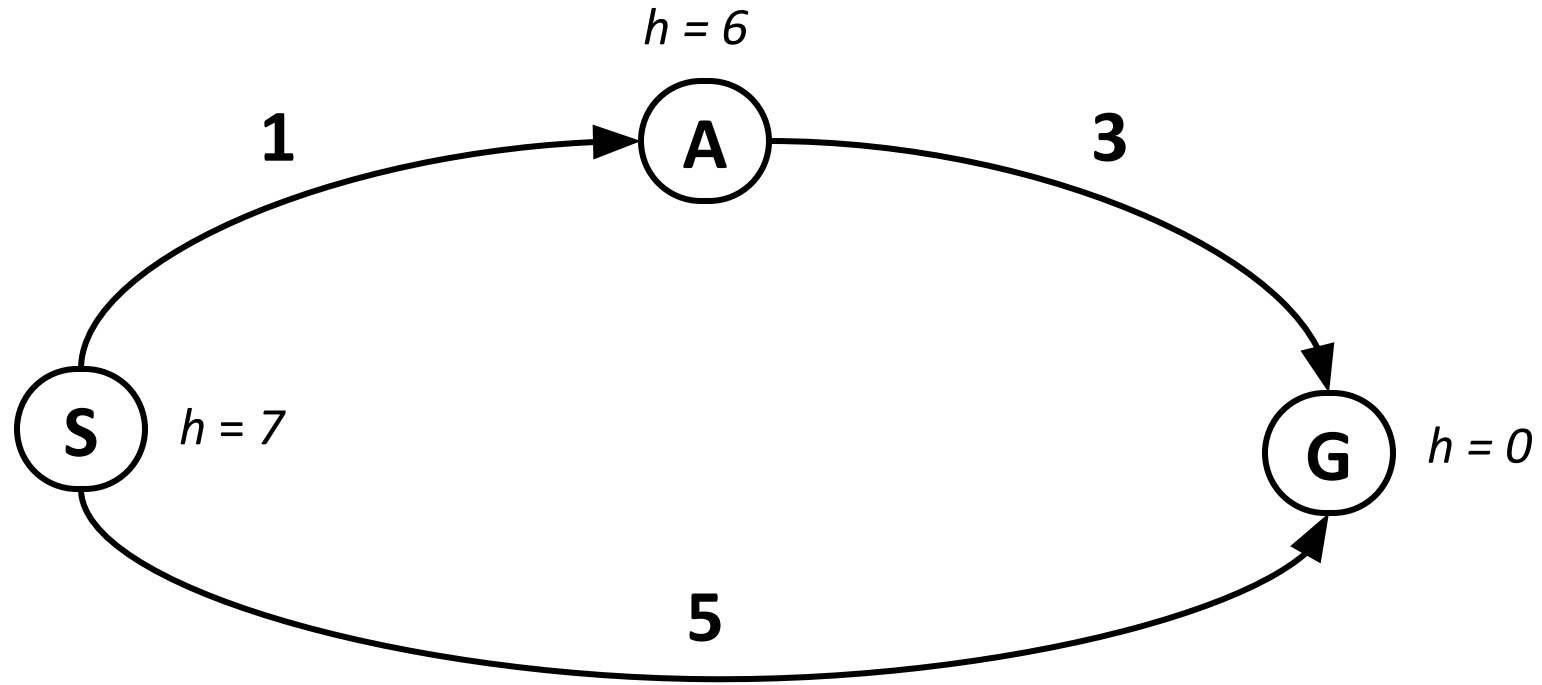
A* 何时终止?

- 目标入队时就应该停止吗?



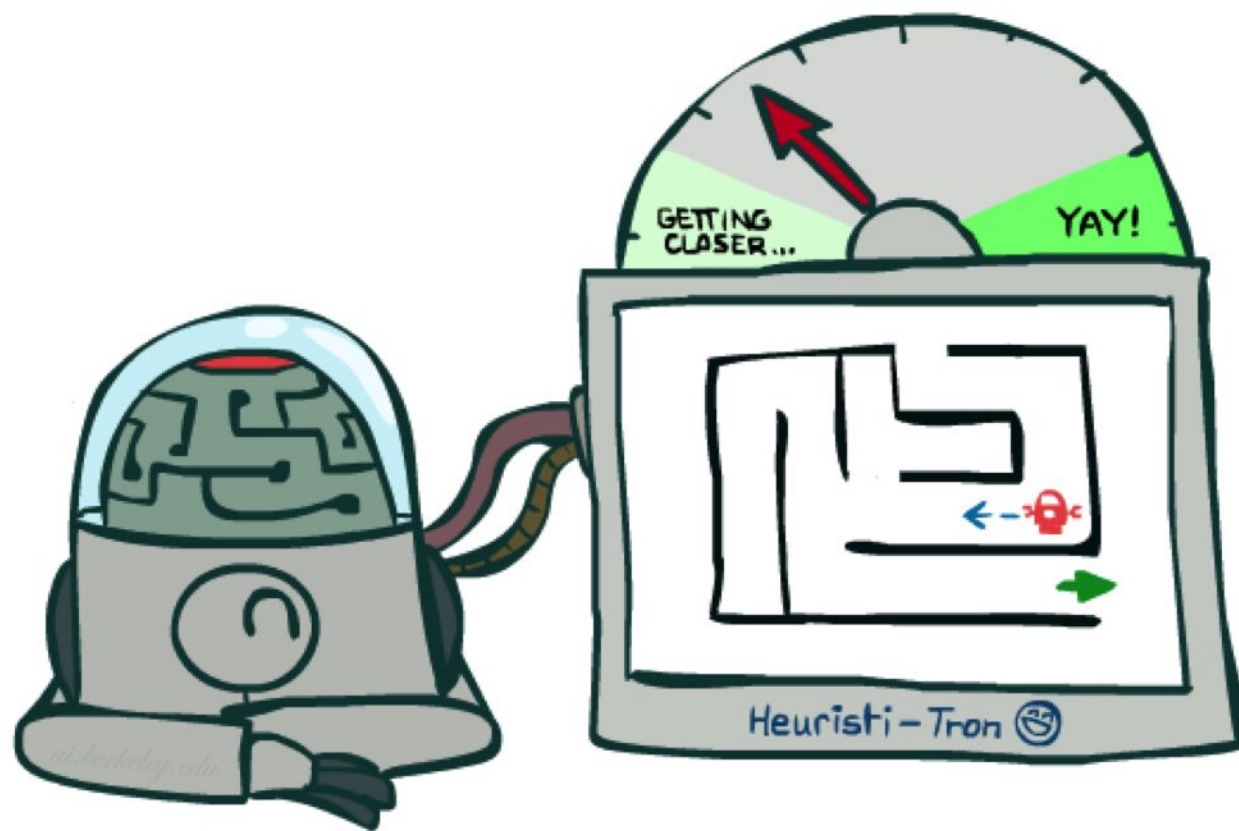
- 不: 只有当目标出队时才停止

A* 最优吗?

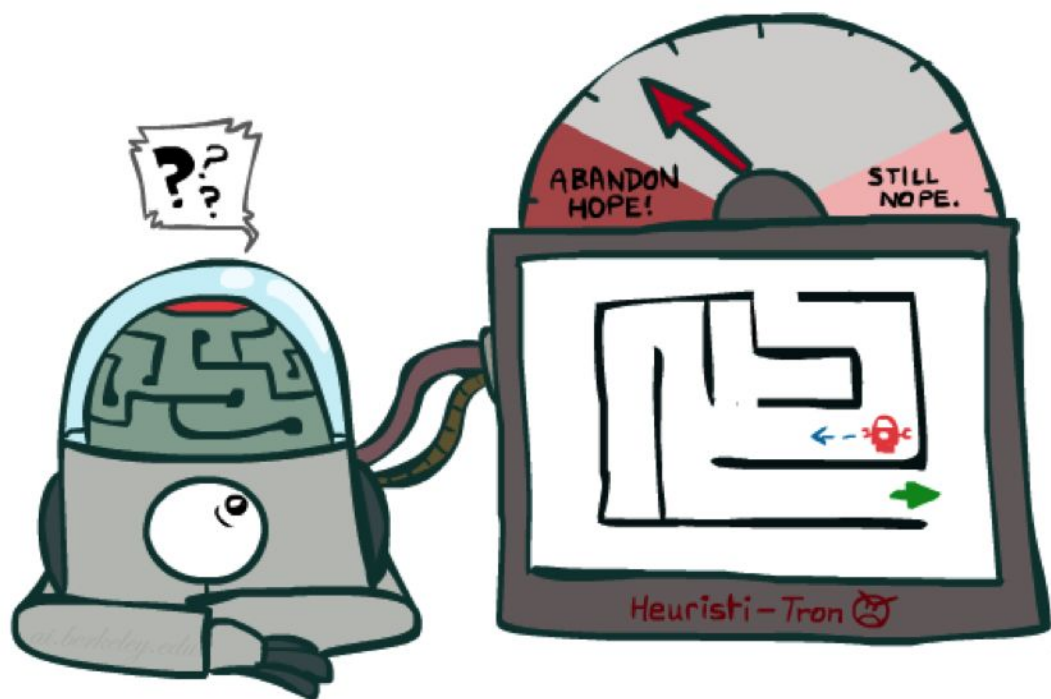


- 哪里出了问题?
- 实际的坏目标代价 < 估计的好目标代价
- 我们需要估计值小于实际代价!

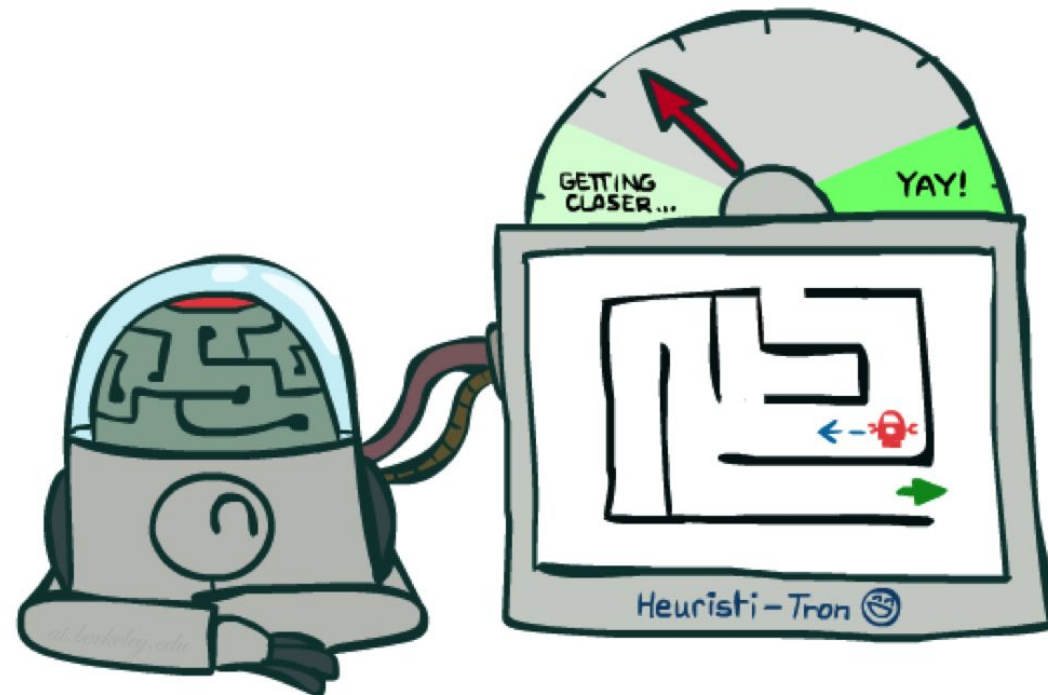
可采纳启发式



思想：可采纳性



不可采纳（悲观）的启发式会把好规划困在边缘，从而破坏最优性



可采纳（乐观）的启发式会拖慢坏规划，但绝不会高估真实代价

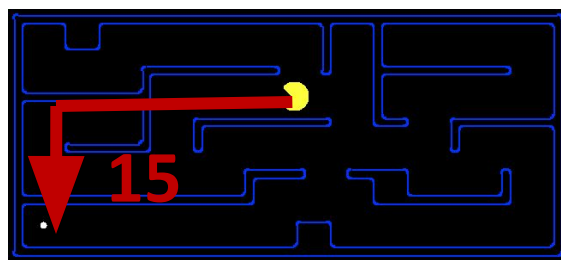
可采纳启发式

- 启发式 h 是可采纳的（乐观的），若：

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

其中 $h^*(n)$ 是到最近目标的真实代价

- 例子：

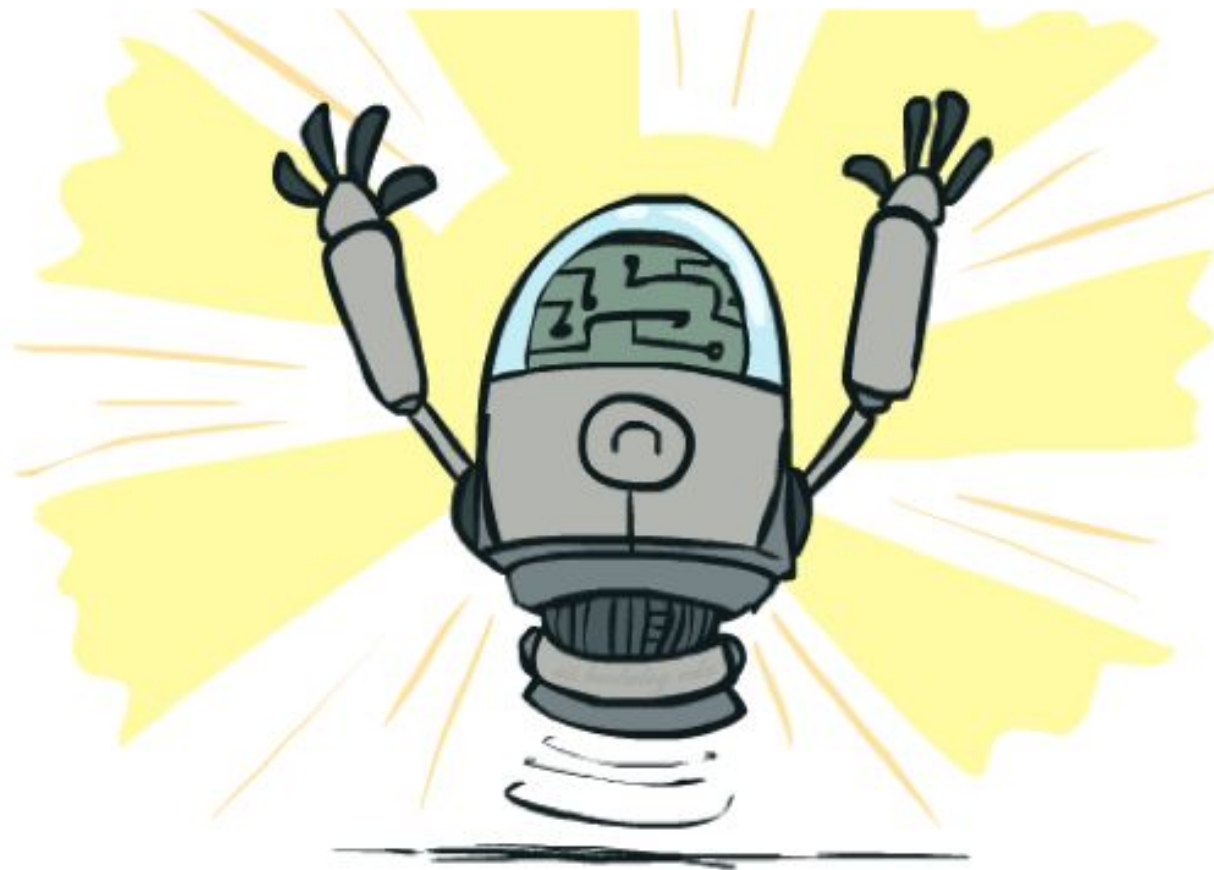


4



- 找到既好又廉价的可采纳启发式是成功的关键

A* 树搜索的最优性



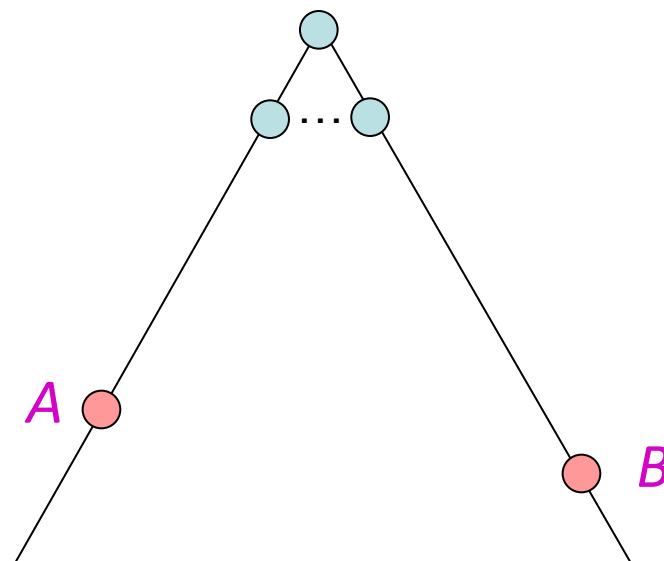
A* 树搜索的最优性

假设：

- A 是最优目标节点
- B 是次优目标节点
- h 是可采纳的

断言：

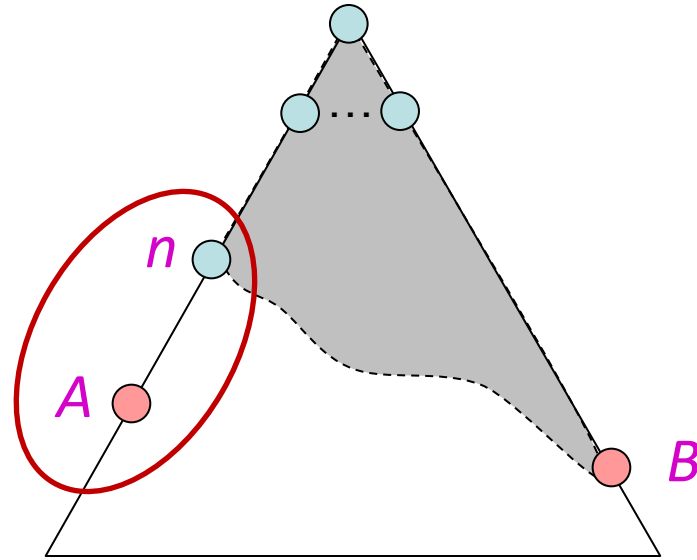
- A 会先于 B 被选中扩展



A* 树搜索的最优性：阻塞

证明：

- 假设 B 在边界 (frontier) 上
- A 的某个祖先 n 也在边界上 (也许就是 A 本身!)
- 断言：n 会先于 B 被扩展
 1. $f(n) \leq f(A)$



$f(n) = g(n) + h(n)$ —— f 代价的定义

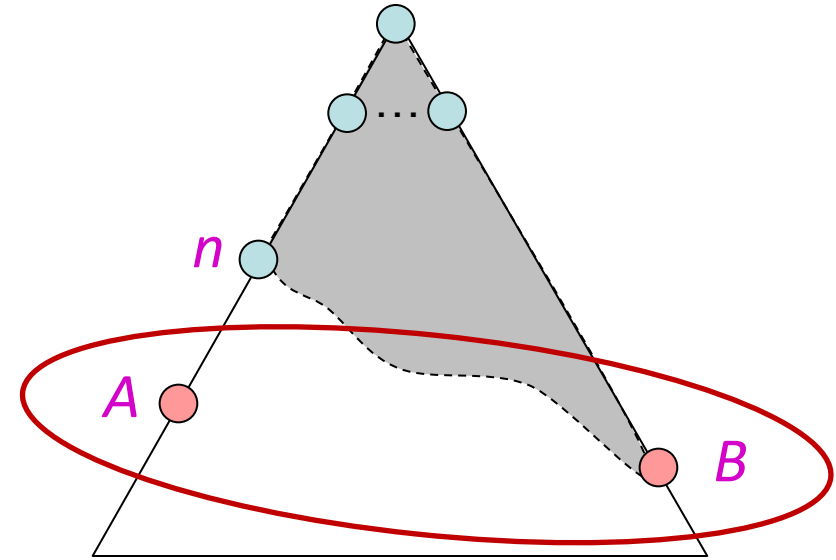
$f(n) \leq g(A)$ —— h 的可采纳性

$g(A) = f(A)$ —— 目标处 $h = 0$

A* 树搜索的最优性：阻塞

证明：

- 假设 B 在边界 (frontier) 上
- A 的某个祖先 n 也在边界上 (也许就是 A 本身!)
- 断言：n 会先于 B 被扩展
 1. $f(n) \leq f(A)$
 2. $f(A) < f(B)$

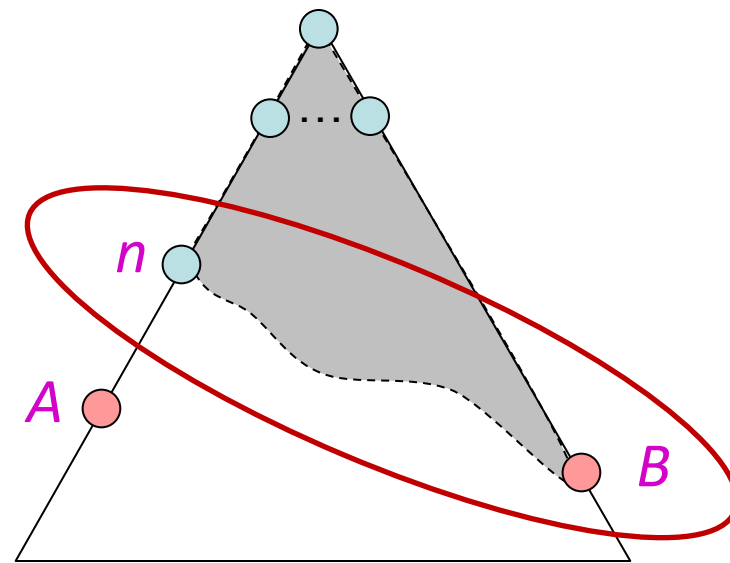


$g(A) < g(B)$ —— B 是次优的
 $f(A) < f(B)$ —— 目标处 $h = 0$

A* 树搜索的最优性：阻塞

证明：

- 假设 B 在边界 (frontier) 上
- A 的某个祖先 n 也在边界上 (也许就是 A 本身!)
- 断言：n 会先于 B 被扩展
 1. $f(n) \leq f(A)$
 2. $f(A) < f(B)$
 3. n 先于 B 被扩展
- A 的所有祖先都先于 B 被扩展
- A 先于 B 被扩展
- A* 树搜索是最优的

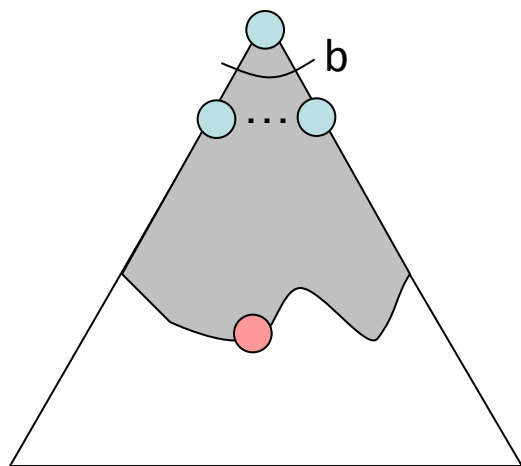


$$f(n) \leq f(A) < f(B)$$

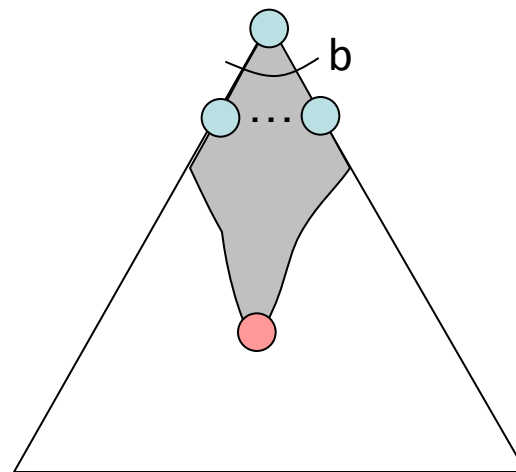
A^* 的性质

A* 的性质

一致代价

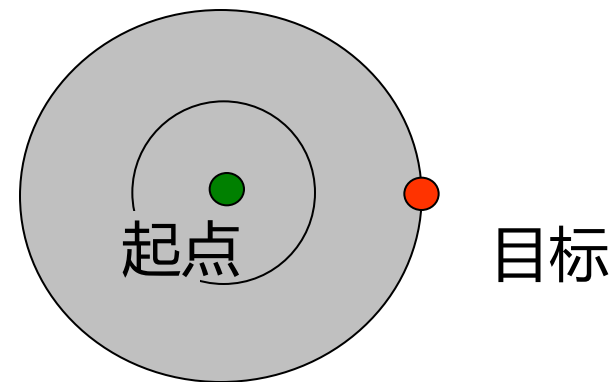


A*

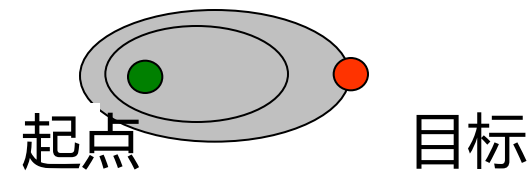


UCS 与 A* 的等值线

- 一致代价在所有“方向”上均匀扩展



- A* 主要朝目标扩展，但也会适度对冲以保证最优性



[演示：等值线 UCS / 贪婪 / A* 空网格 (L3D1)]

[演示：等值线 A* 吃豆人小迷宫 (L3D5)]

比较



贪婪



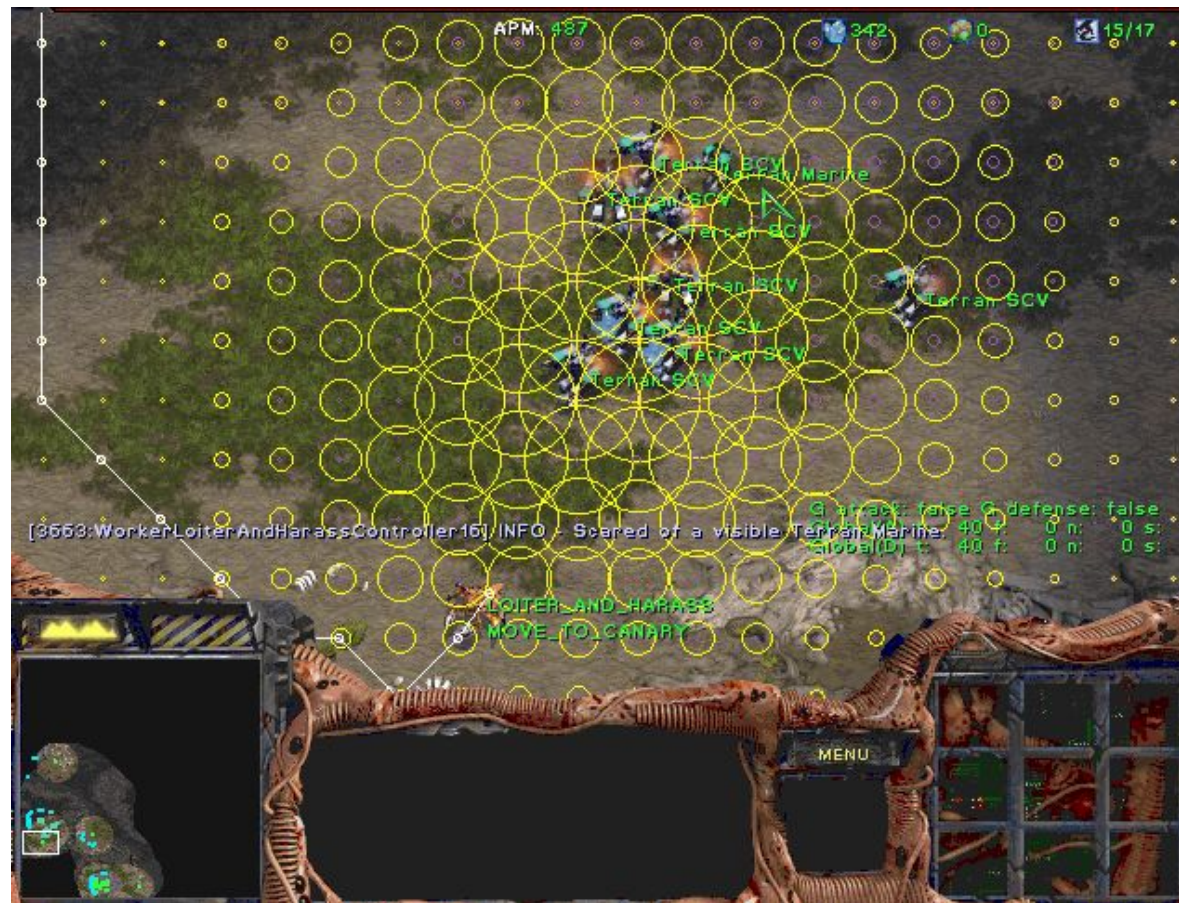
一致代价



A^*

A* 的应用

- 电子游戏
- 路径规划 / 路由问题
- 资源规划问题
- 机器人运动规划
- 语言分析
- 机器翻译
- 语音识别
- ...



[演示: UCS / A* 吃豆人微型迷宫 (L3D6, L3D7)]

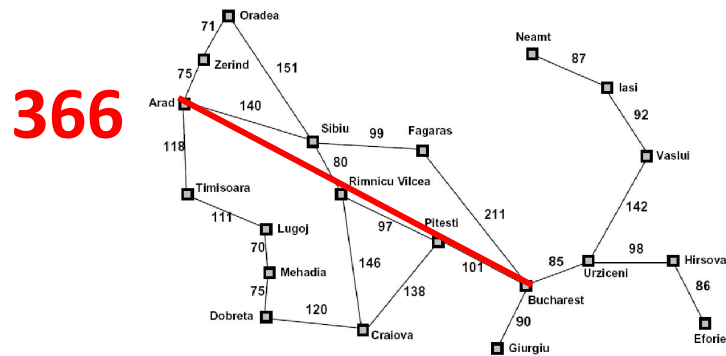
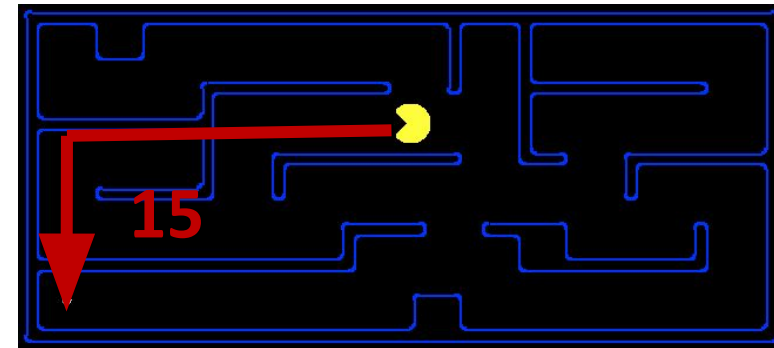
[演示: 猜算法 空旷/浅水/深水 (L3D8)]

构造启发式



构造可采纳启发式

- 求解困难搜索问题并保证最优，大部分工作量在于提出可采纳启发式
- 可采纳启发式通常是“松弛问题”的解——即在原问题基础上允许新的动作

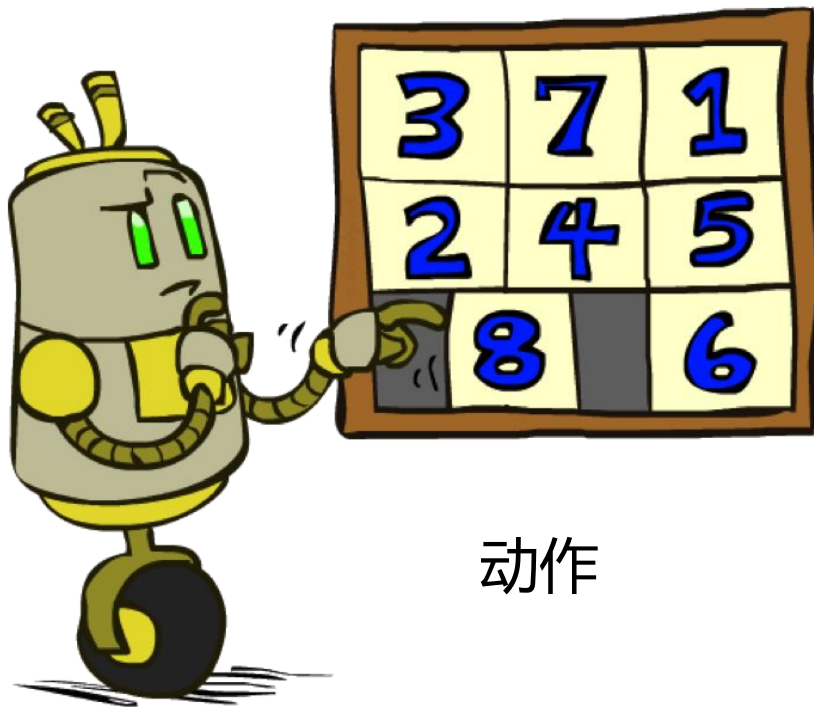


- (不可采纳的启发式往往也有用)

例子：8 数码

7	2	4
5		6
8	3	1

初始状态



动作

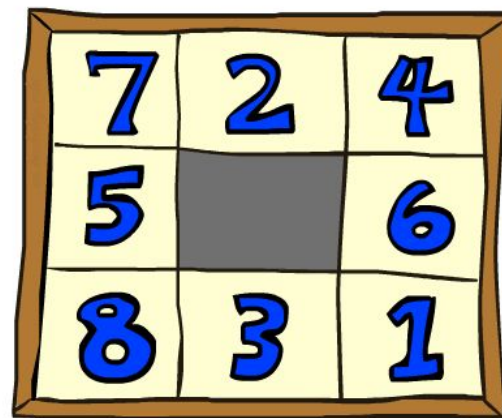
	1	2
3	4	5
6	7	8

目标状态

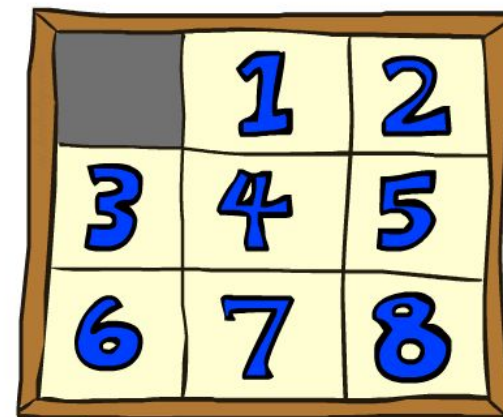
- 状态是什么?
- 有多少个状态?
- 动作是什么?
- 从初始状态出发有多少个后继?
- 代价应该是什么?
- 有什么启发式?

8 数码 I

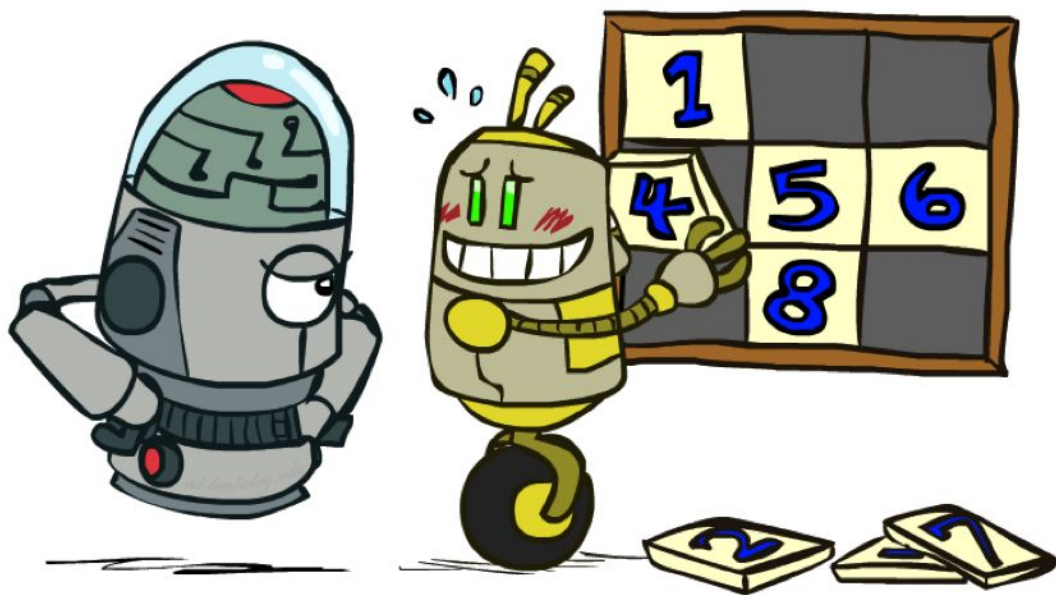
- 启发式：放错位置的数字牌个数
- 为什么它是可采纳的？
- $h(\text{start}) = 8$
- 这是松弛问题的启发式



初始状态



目标状态



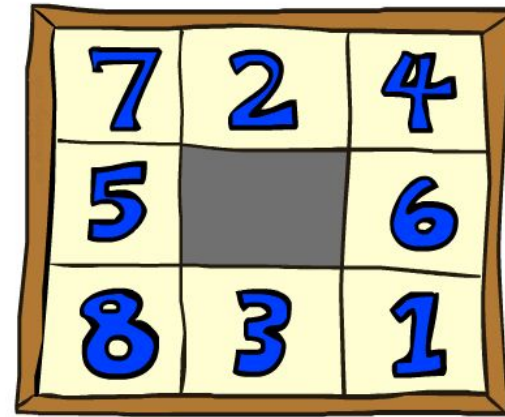
A* 扩展的搜索节点数
当最优路径步数为.....

	...4 步	...8 步	...12 步
UCS	112	6,300	3.6×10^6
错牌数	13	39	227

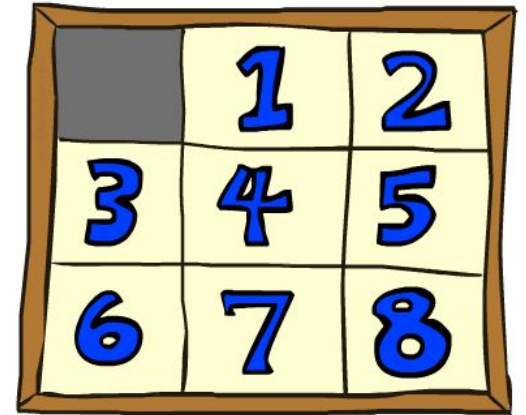
数据来源：Andrew Moor

8 数码 II

- 如果我们有一个更简单的 8 数码：任何数字牌任何时候都可以朝任意方向滑动，无视其他牌，会怎样？
- 总曼哈顿距离
- 为什么它是可采纳的？
- $h(\text{start}) = 3 + 1 + 2 + \dots = 18$



初始状态



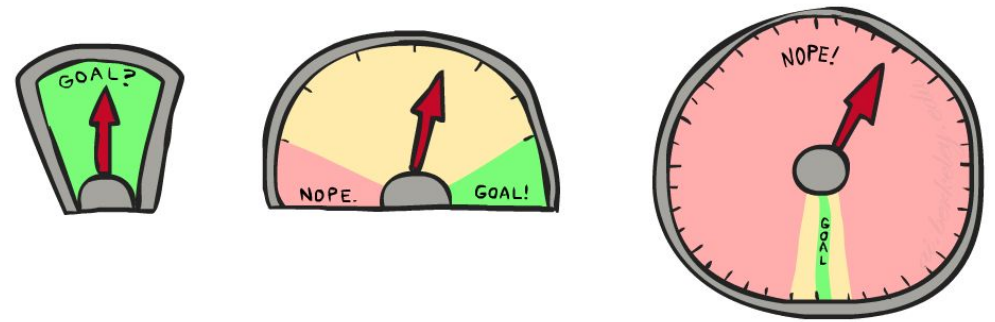
目标状态

当最优路径步数为.....时 平均扩展的节点数			
	...4 步	...8 步	...12 步
错牌数	13	39	227
曼哈顿	12	25	73

8 数码 III

- 如果直接用真实代价作为启发式呢？

- 它可采纳吗？
- 能减少扩展的节点数吗？
- 它有什么问题？



- 对 A*：这是估计质量与每节点计算量之间的权衡

- 启发式越接近真实代价，扩展的节点越少，但通常每个节点要花更多工作量来计算启发式本身

更多启发式性质

平凡启发式与支配关系

- 支配关系：若 $h_a \geq h_c$

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

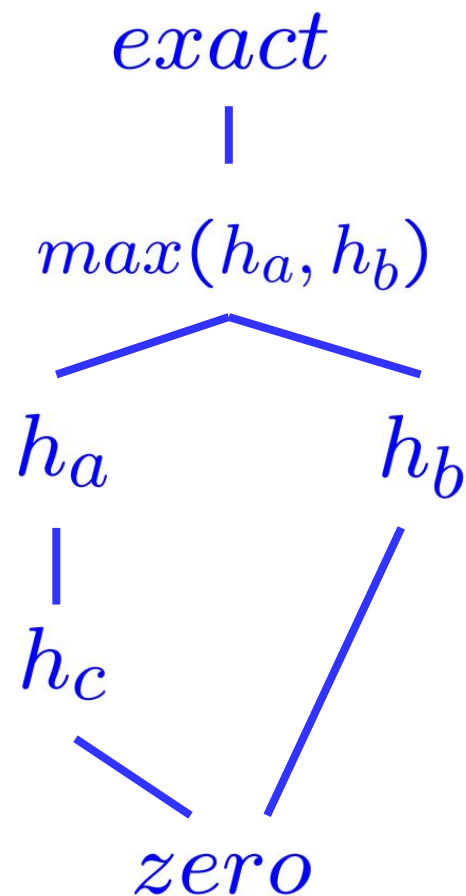
- 启发式构成一个半格：

- 可采纳启发式的最大值仍是可采纳的

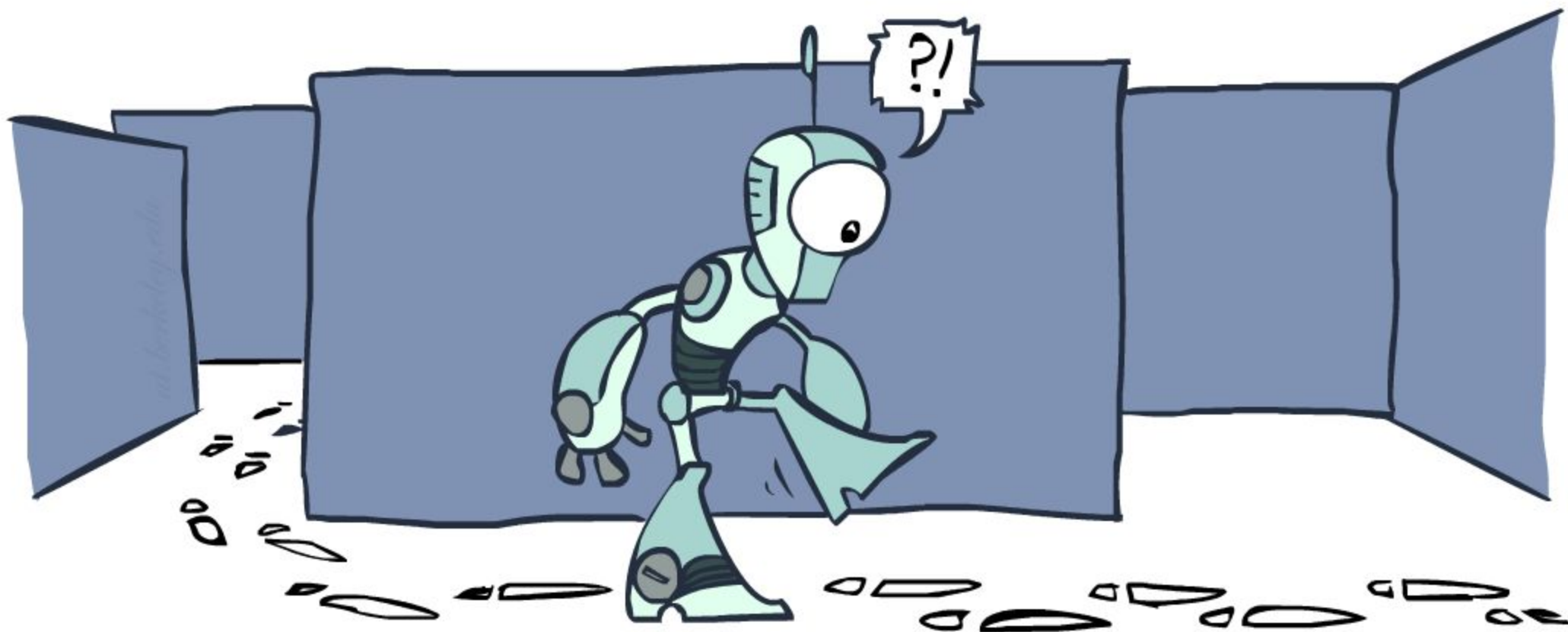
$$h(n) = \max(h_a(n), h_b(n))$$

- 平凡启发式

- 格的底部是零启发式（这会给我们什么？）
- 格的顶部是精确启发式



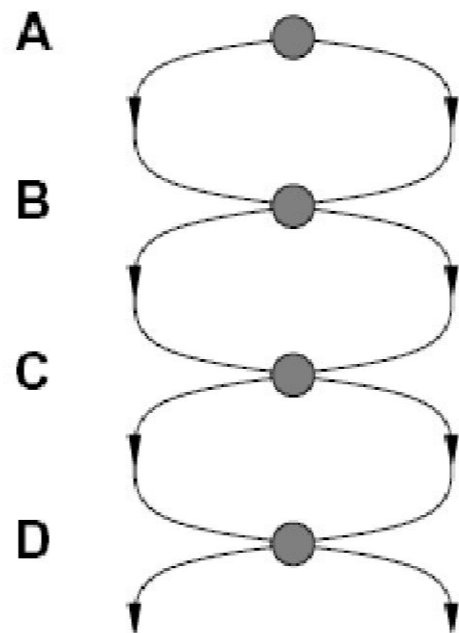
图搜索



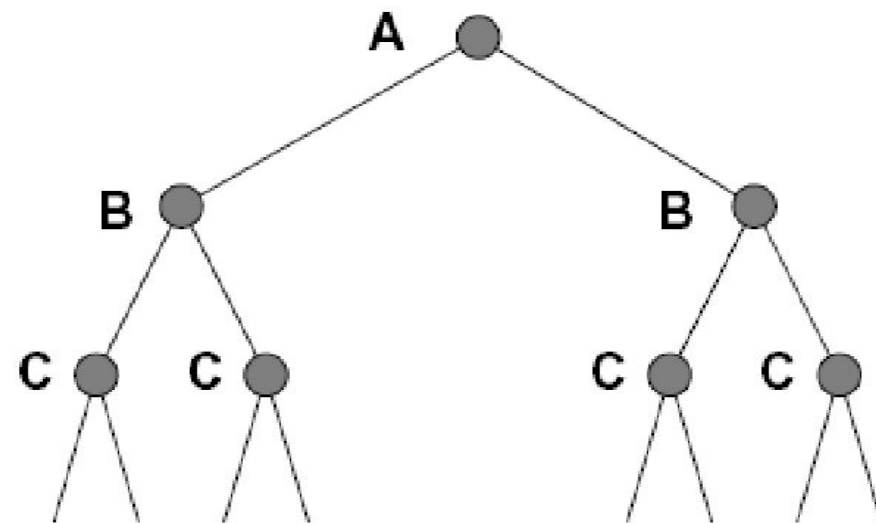
树搜索：额外的工作！

- 未能检测重复状态可能导致指数级增长的工作量。

状态图

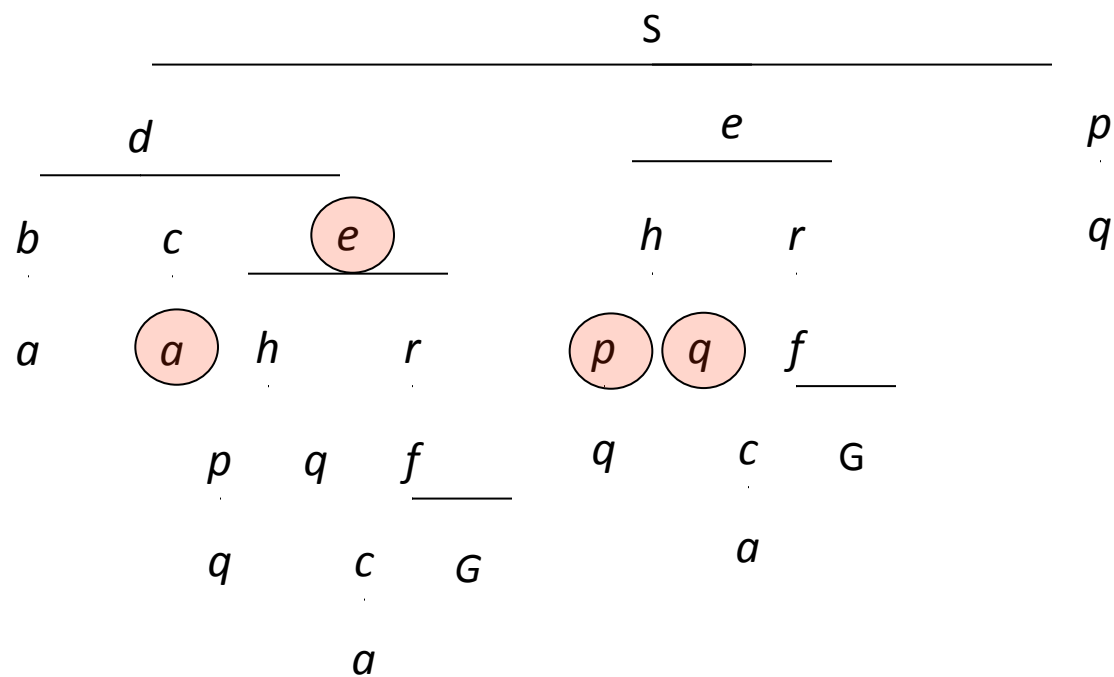


搜索树



图搜索

- 例如在 BFS 中，我们不必费心展开被圈出的节点（为什么？）



图搜索

- 思想：绝不重复展开同一个状态
- 如何实现：
 - 树搜索 + 已展开状态的集合 (closed set, 关闭列表)
 - 逐节点扩展搜索树，但是.....
 - 在展开一个节点前，先检查其状态是否从未被展开过
 - 若不新，跳过；若新，加入关闭列表
- 重要：关闭列表要用集合存储，而不是列表
- 图搜索会破坏完备性吗？为什么？
- 最优性又如何？

树搜索伪代码

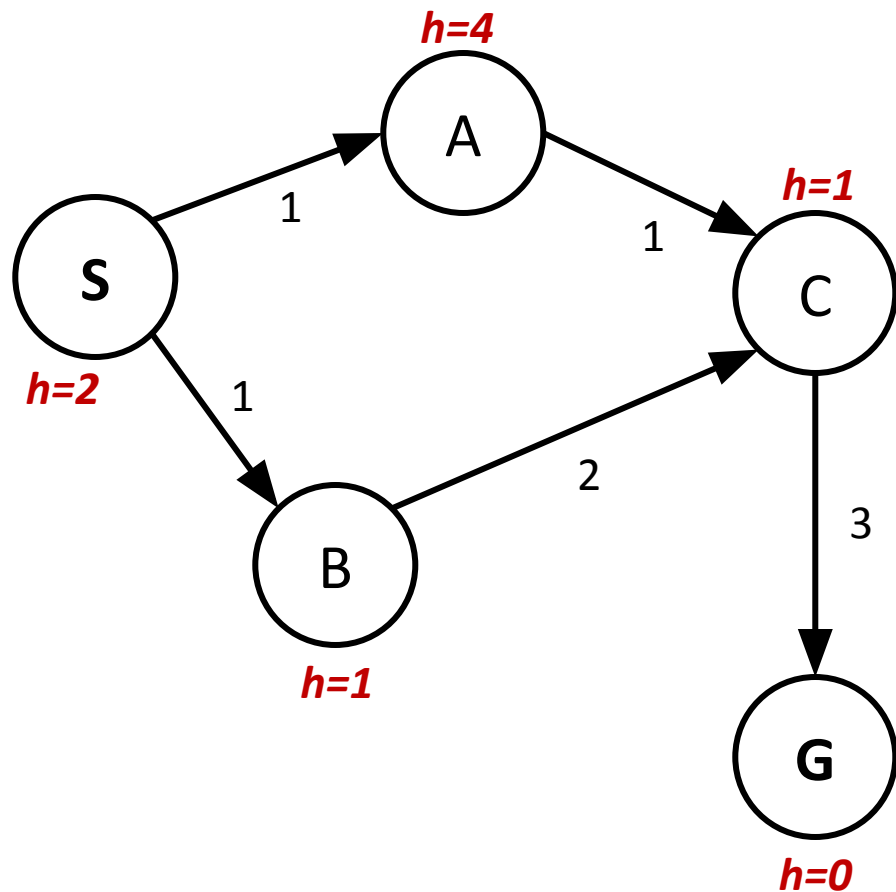
```
function TREE-SEARCH(problem, fringe) return a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    for child-node in EXPAND(STATE[node], problem) do
      fringe ← INSERT(child-node, fringe)
    end
  end
end
```

图搜索伪代码

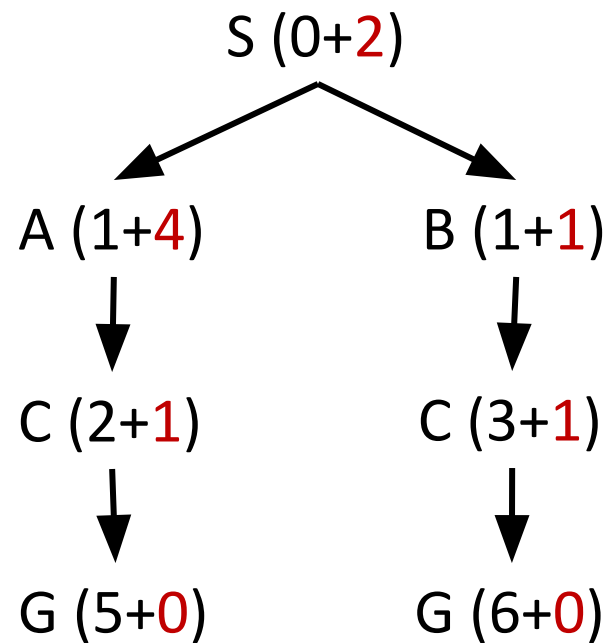
```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed  $\leftarrow$  an empty set
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe  $\leftarrow$  INSERT(child-node, fringe)
      end
    end
  end
```

A* 图搜索出问题了?

状态空间图



搜索树



用已展开集合做简单检查会挡住 C
复杂检查允许新的 C (若比旧的更便宜)
但需要重新计算 C 的所有后代

启发式的一致性

- 核心思想：启发式估计代价 $\leq$ 实际代价

- 可采纳性：启发式代价 $\leq$ 到目标的实际代价

$$h(A) \leq \text{从 A 到 G 的实际代价}$$

- 一致性：每条弧上，启发式“弧”代价 $\leq$ 实际代价

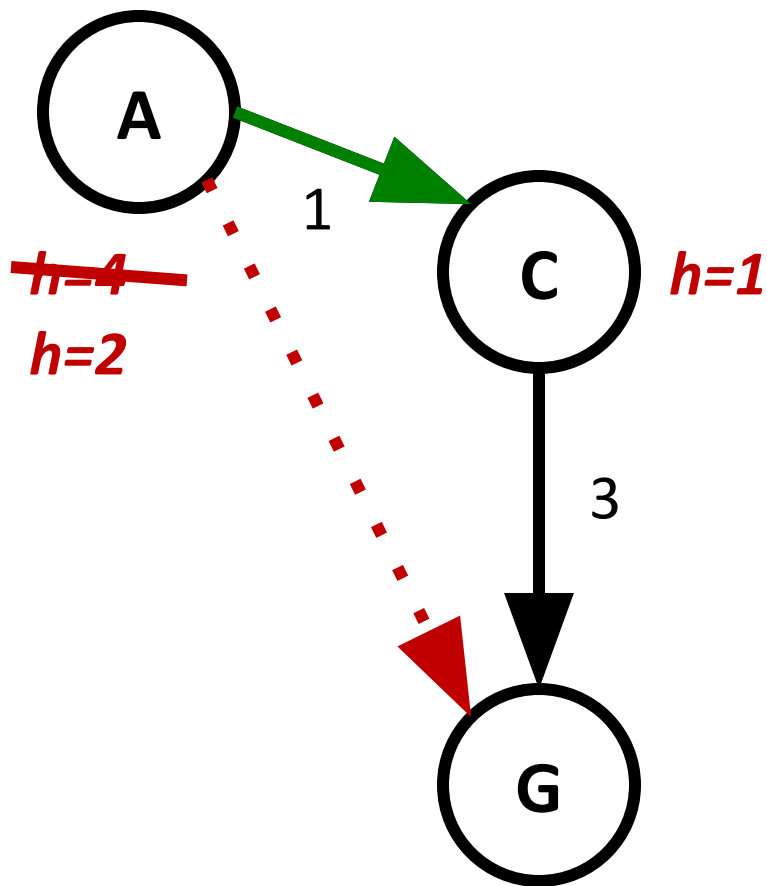
$$h(A) - h(C) \leq \text{A 到 C 的代价}$$

- 一致性的推论：

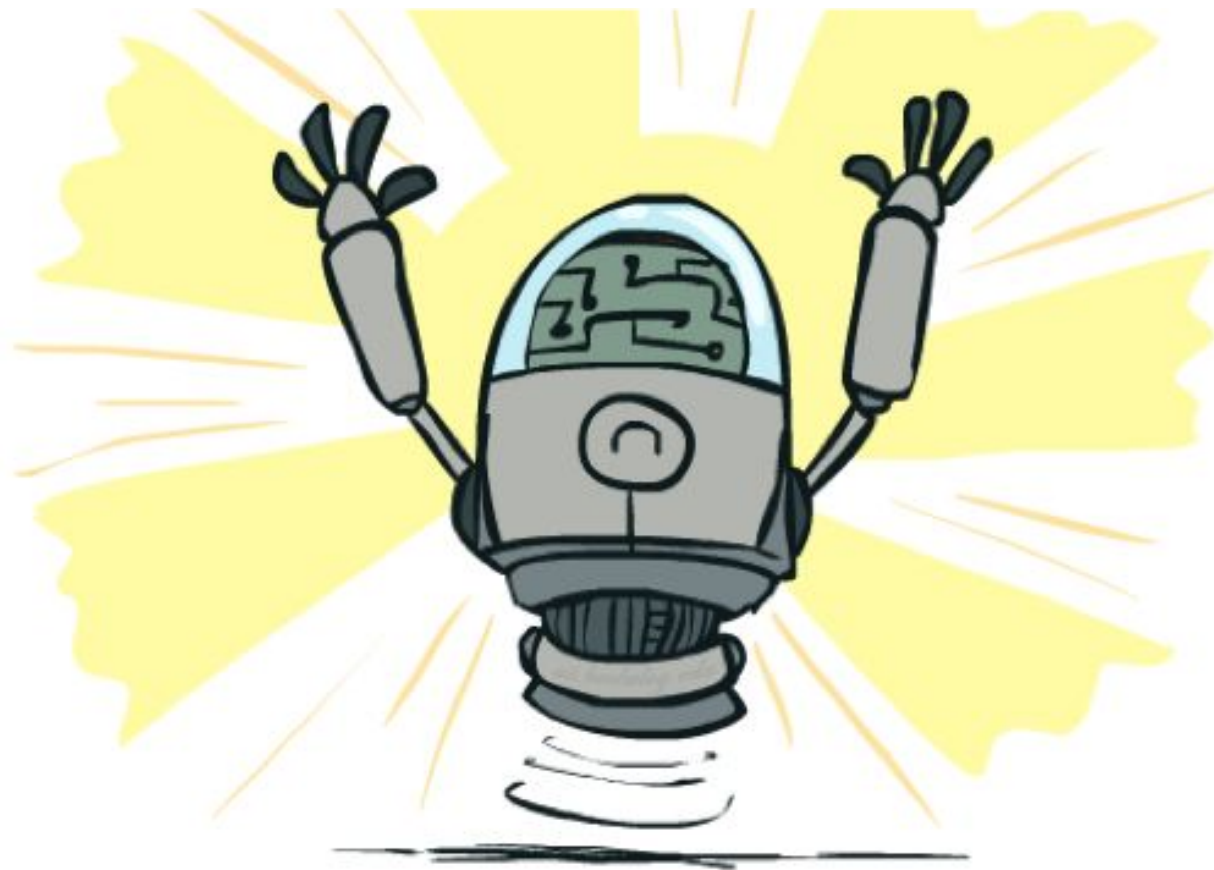
- 沿一条路径，f 值永不下降

$$h(A) \leq \text{A 到 C 的代价} + h(C)$$

- A* 图搜索是最优的

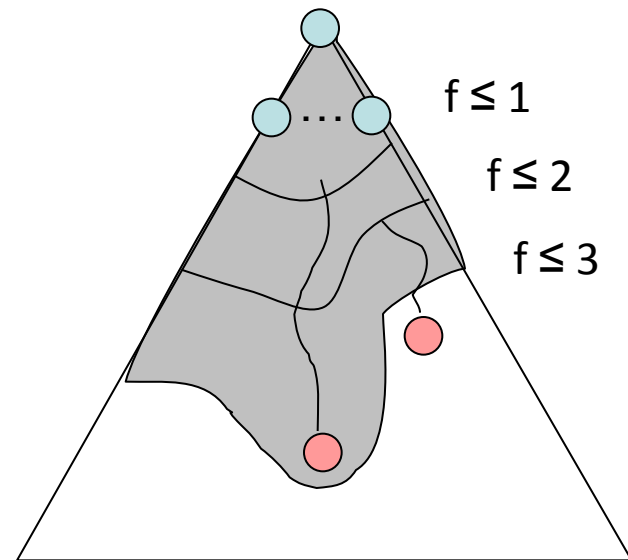


A* 图搜索的最优性



A* 图搜索的最优性

- 概要：考虑 A* 使用一致性启发式时的行为：
 - 事实 1：在树搜索中，A* 按总 f 值递增的顺序扩展节点 (f 等值线)
 - 事实 2：对每个状态 s ，最优到达 s 的节点会先于次优到达 s 的节点被扩展
- 结论：A* 图搜索是最优的



最优性

- 树搜索：
 - 若启发式可采纳， A^* 最优
 - UCS 是特例 ($h = 0$)
- 图搜索：
 - 若启发式一致， A^* 最优
 - UCS 最优 ($h = 0$ 是一致的)
- 一致性蕴涵可采纳性
- 一般而言，大多数自然的可采纳启发式往往是一致的，尤其是来自松弛问题的启发式

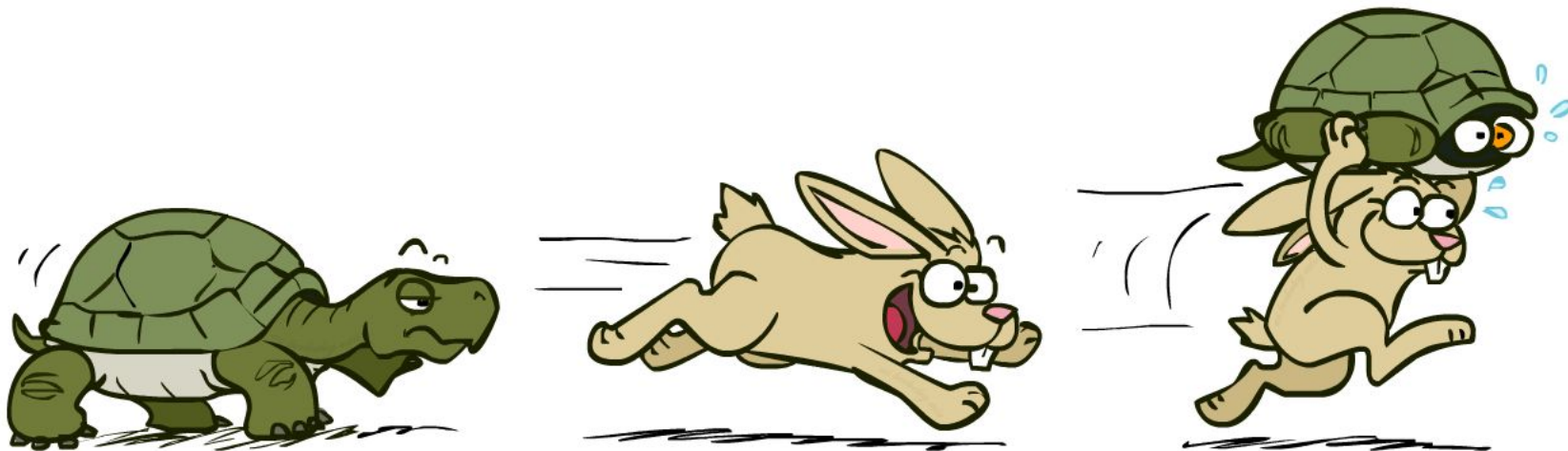


A*: 小结



A*: 小结

- A* 同时使用后向代价与（估计的）前向代价
- 在启发式可采纳 / 一致的树或图上，A* 是最优的
- 启发式设计是关键：常常利用松弛问题



树搜索伪代码

```
function TREE-SEARCH(problem, fringe) return a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    for child-node in EXPAND(STATE[node], problem) do
      fringe ← INSERT(child-node, fringe)
    end
  end
end
```

图搜索伪代码

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed  $\leftarrow$  an empty set
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe  $\leftarrow$  INSERT(child-node, fringe)
      end
    end
  end
```

A* 图搜索的最优性

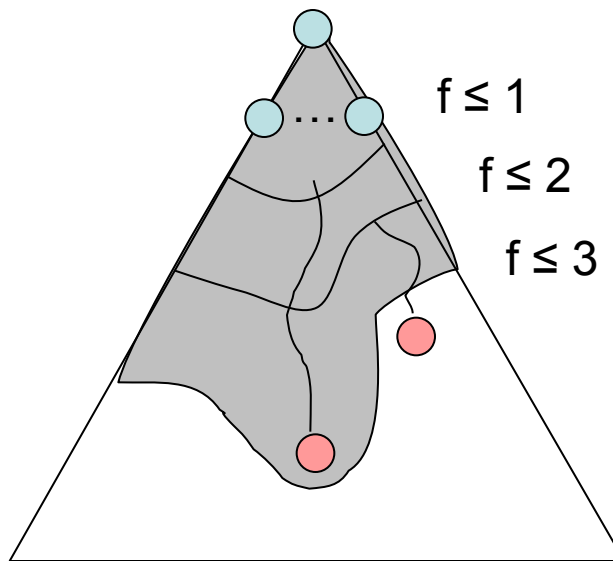
- 考虑 A* 的行为:

- 按总 f 值递增的顺序扩展节点 (f 等值线)

提醒: $f(n) = g(n) + h(n) = \text{到 } n \text{ 的代价} + \text{启发式}$

- 证明思路: 最优目标的 f 值最低, 所以必然最先被扩展

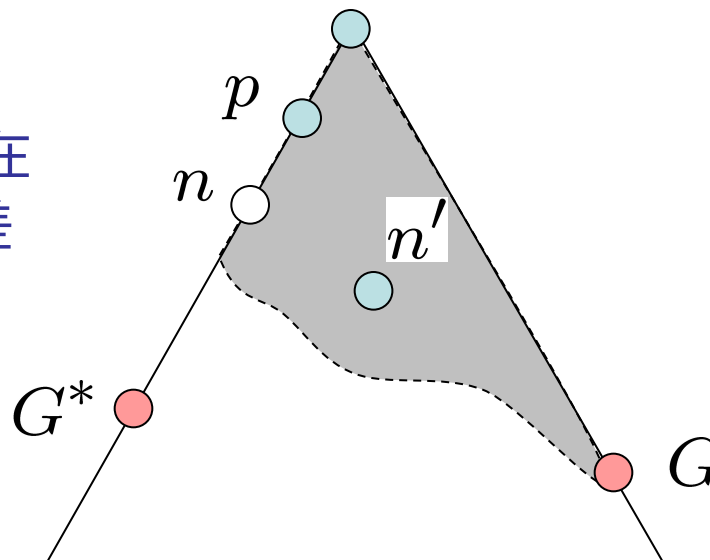
这个论证有一个问题。我们默认了什么成立的?



A* 图搜索的最优性

证明:

- 新的可能问题: 通往 G^* 路径上的某个 n , 在需要时不在队列中, 因为同一状态有个更差的 n' 先被出队并扩展 (灾难!)
- 取树中最高的这样的 n
- 设 p 为 n' 被弹出时队列中 n 的祖先
- 由一致性, $f(p) < f(n)$
- 由 n' 次优, $f(n) < f(n')$
- p 会先于 n' 被扩展
- 矛盾!



但是.....

- A* 会把整个已探索区域都保存在内存中
- => 在你等答案等到无聊之前，内存就先耗尽了
- 有一些使用更少内存的变体（见 3.5.5 节）：
 - IDA* 像迭代加深一样工作，只是用 f 限界代替深度限界
 - 每次迭代记住超过当前限界的最小 f 值，作为新的限界
 - 当 f 为实数且每个节点取值唯一时效率很低
 - RBFS 是递归的深度优先搜索，其 f 限界 = 当前节点任一祖先处最佳替代路径的 f 值
 - 超过限界时递归回退，但记住该分支上可达的最佳 f 值
 - SMA* 把所有可用内存都用于队列，尽量减少抖动
 - 队列满时，丢弃队列中最差的节点，但在其父节点中记住它的值