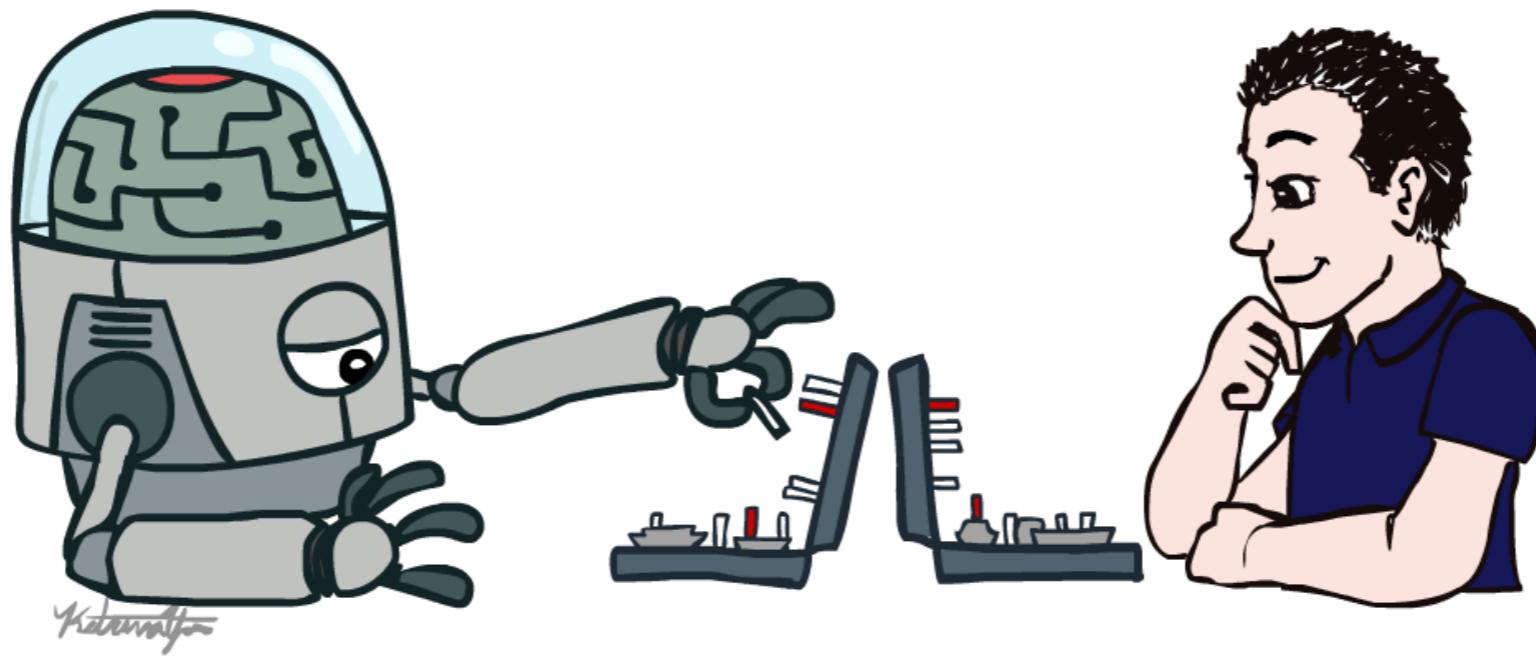
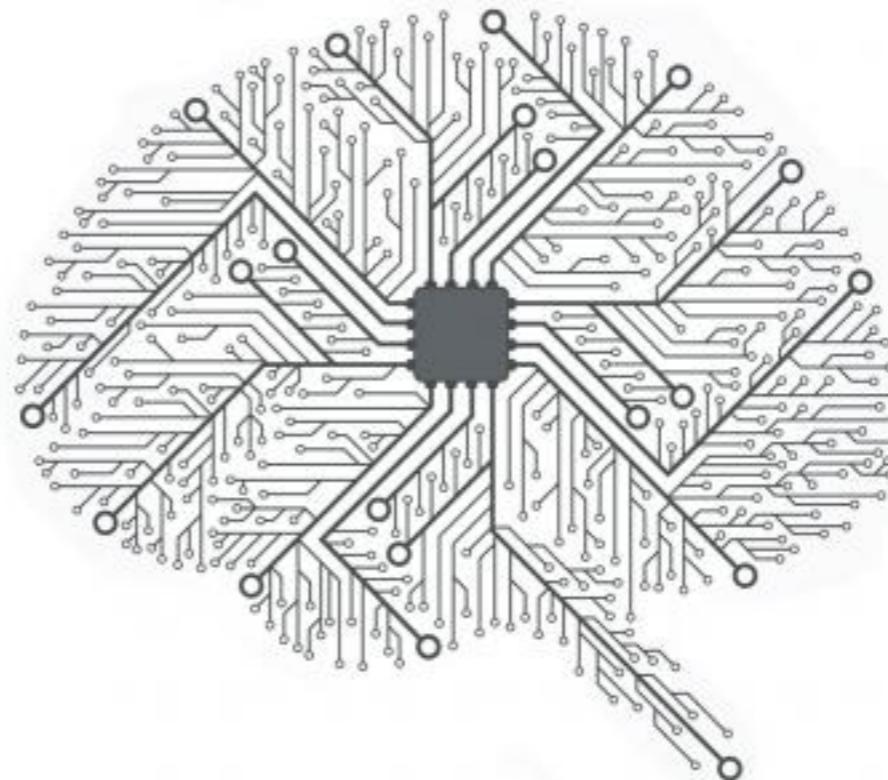


人工智能



第二章·通过搜索求解问题

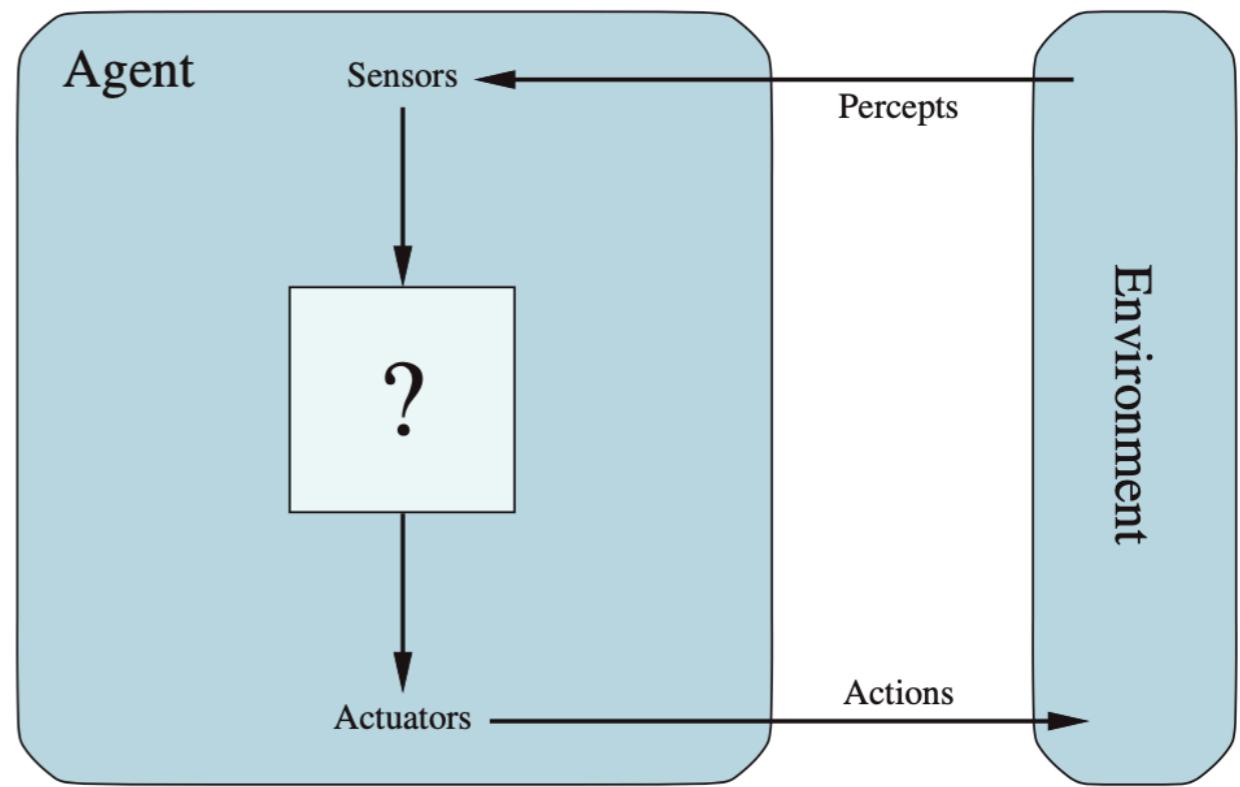
- Agent和环境
- 提前计划的Agent
- 搜索问题
- 搜索算法



Agent

- Agent ——代理、智能体

- 就是能够行动的某种东西
- 指能自主活动的软件或者硬件实体。
- 是指驻留在某一环境下，能持续自主地发挥作用，具备驻留性、反应性、社会性、主动性等特征的计算实体。



智能体和环境

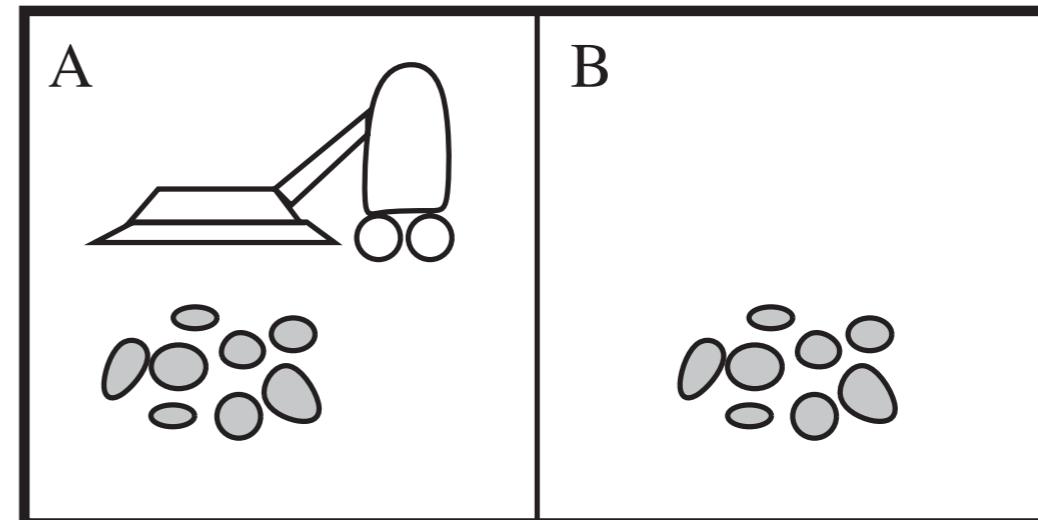
- 一个智能体通过自身的传感器感知环境，并通过自身的促动器去相应的行动。
- 人是智能体
 - 传感器 = 视觉，听觉，触摸，嗅觉，味觉，主体感觉
 - 促动器（激励器） = 肌肉，分泌物质，大脑状态的改变
- 便携式计算器是智能体
 - 传感器 = 按键状态传感器
 - 促动器 = 数字显示

智能体函数(function)和智能体程序(program)

- 智能体函数: 感知历史 (序列) 到行动的映射
 - $f : P^* \rightarrow A$
- 智能体程序 I , 运行在某个计算机 M, 以实现智能体函数
 - $f = \text{Agent}(I, M)$
 - 运行在智能体内
 - M, 有限的运算和存储空间
 - 程序运行可能比较慢; 也可能被新的感知所打断 (或忽略它们) 。
 - 不是每一个智能体函数都能被实现, M的限制

举例：吸尘器世界问题

- 感知对象：
 - [地点, 洁净状态]
([A, 有灰尘])
- 行动：
 - 左, 右, 吸尘, 无动作



吸尘器智能体

智能体函数

感知序列	行动
[A, 干净]	向右移动
[A, 灰尘]	吸尘
[B, 干净]	向左移动
[B, 灰尘]	吸尘
[A, 干净], [B, 干净]	向左移动
[A, 干净], [B, 灰尘]	吸尘
...	...

智能体程序

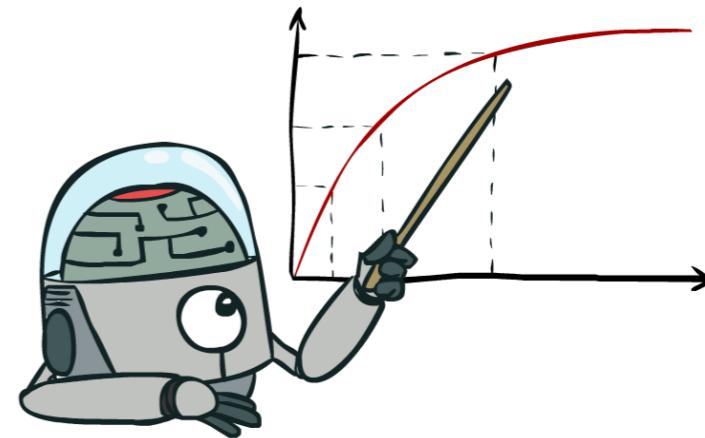
Function 简单吸尘器([位置,状态])
returns 一个行动
 if 状态 = 灰尘 then return 吸尘
 else if 位置 = A then return 向右移动
 else if 位置 = B then return 向左移动

合理性 (Rationality)

- 固定的性能指标 (performance measure) 衡量环境状况
 - 评估智能体的行为所导致环境变化后的状态，比如，每小时每一平米干净的地面加一分
 - 关注的是环境状态，而不是智能体的态度、状态
 - 同样的性能指标，可能被不同行为的智能体所达成
- 一个合理的智能体 (rational agent)
 - 给定最新的感知序列，和关于环境的预先知识(prior knowledge)，选择为了最大化性能指标期望值的行动。
 - 问题：之前定义的吸尘器智能体实现的是一个合理的智能体函数吗？

合理性 (Rationality)

- 合理性决定于4个因素
 - 性能指标
 - 智能体的预先知识
 - 智能体能够采取的行动集合
 - 智能体的感知序列
- 达成合理性意味着最大化期望值

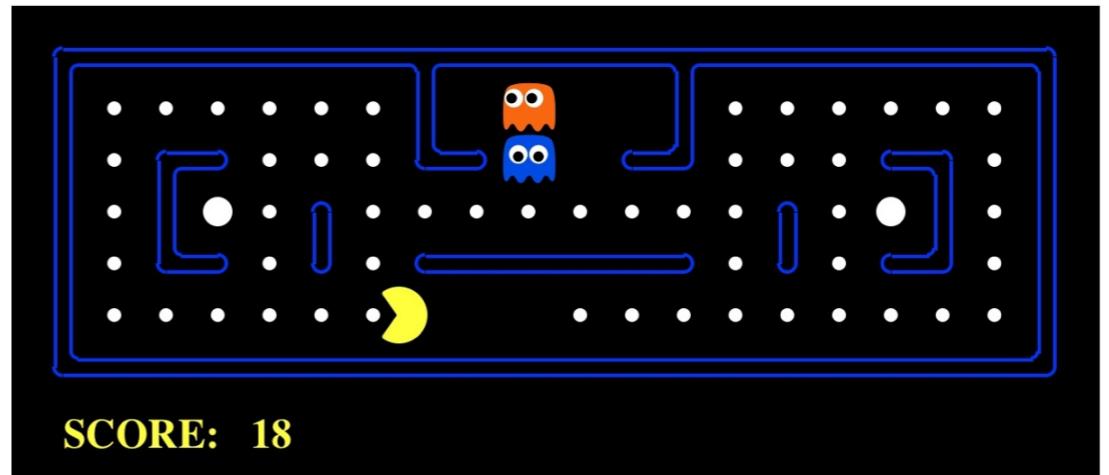


合理性 (Rationality)

- 合理的智能体对环境是无所不知的吗?
 - 不是。受限于传感器及其感知。
- 合理的智能体具有超常的洞察力吗?
 - 不是。可能缺少对环境动态变化的掌握。
- 那么，合理的智能体在环境中探索和学习吗?
 - 是的。只有具备这些，才能在未知的环境中行动。
- 合理的智能体不尽然都会表现的成功，但是，
 - 它们都是有自主性的（不是简单规则化的选择行为动作，超越初始的程序）

任务环境描述-- PEAS

- 性能指标(Performance measure)
 - -1 每走一步； +10 一个豆子； +500 吃完所有豆子； -500 被吃
- 环境 (Environment)
 - 整个游戏环境, pacman、豆子和妖怪的状态
- 促动器(Actuators)
 - 左、右、上、下
- 传感器 (Sensors)
 - 整个游戏环境状态可见



环境的类型

- Observable \ 可观察的
- Single or Multi Agents \ 是否有其他的智能体
- Deterministic \ 确定性的
- Episodic \ 片段式的
- Static \ 静态的
- Discrete \ 离散的

环境类型的示例

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

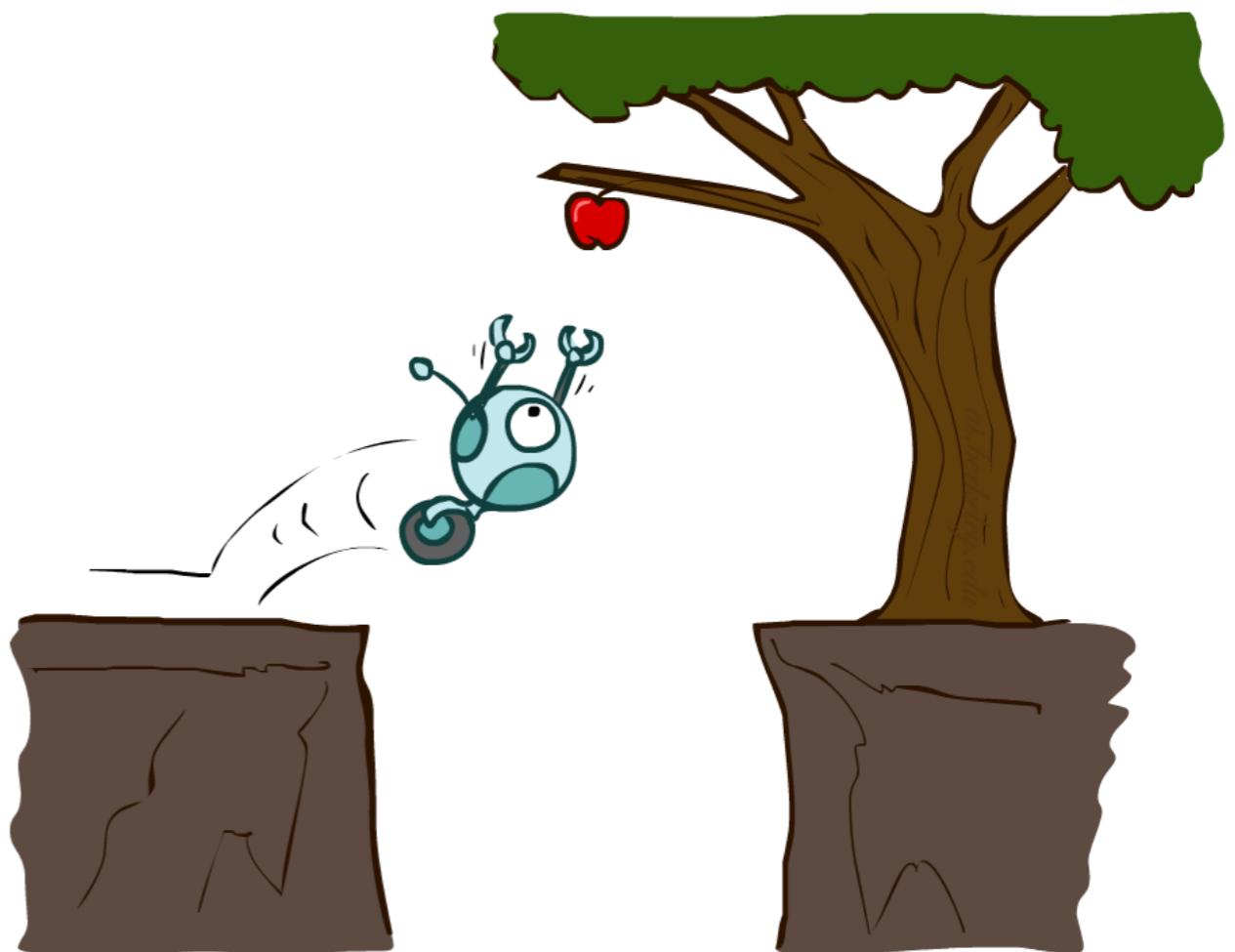
Pacman??

智能体类型

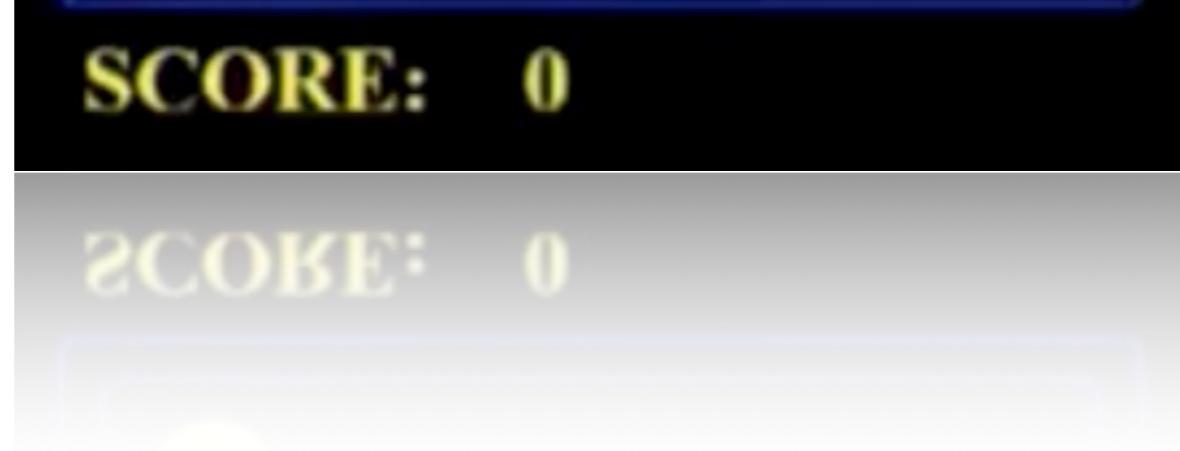
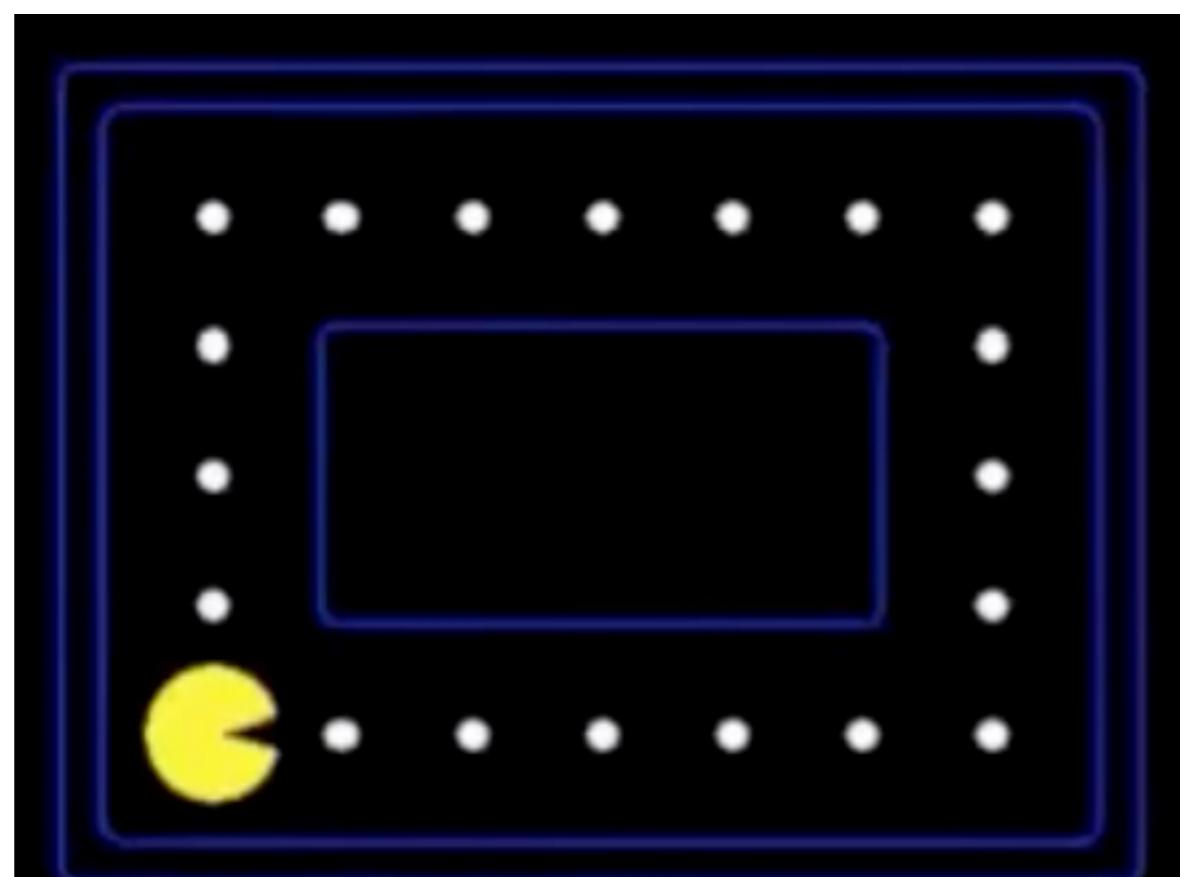
- 从简单的到复杂的
 - 简单反射型智能体(reflex)
 - 记录状态的反射智能体 (reflex with state)
 - 基于目标的智能体 (goal-based)
 - 基于功效的智能体 (utility-based)

反射智能体

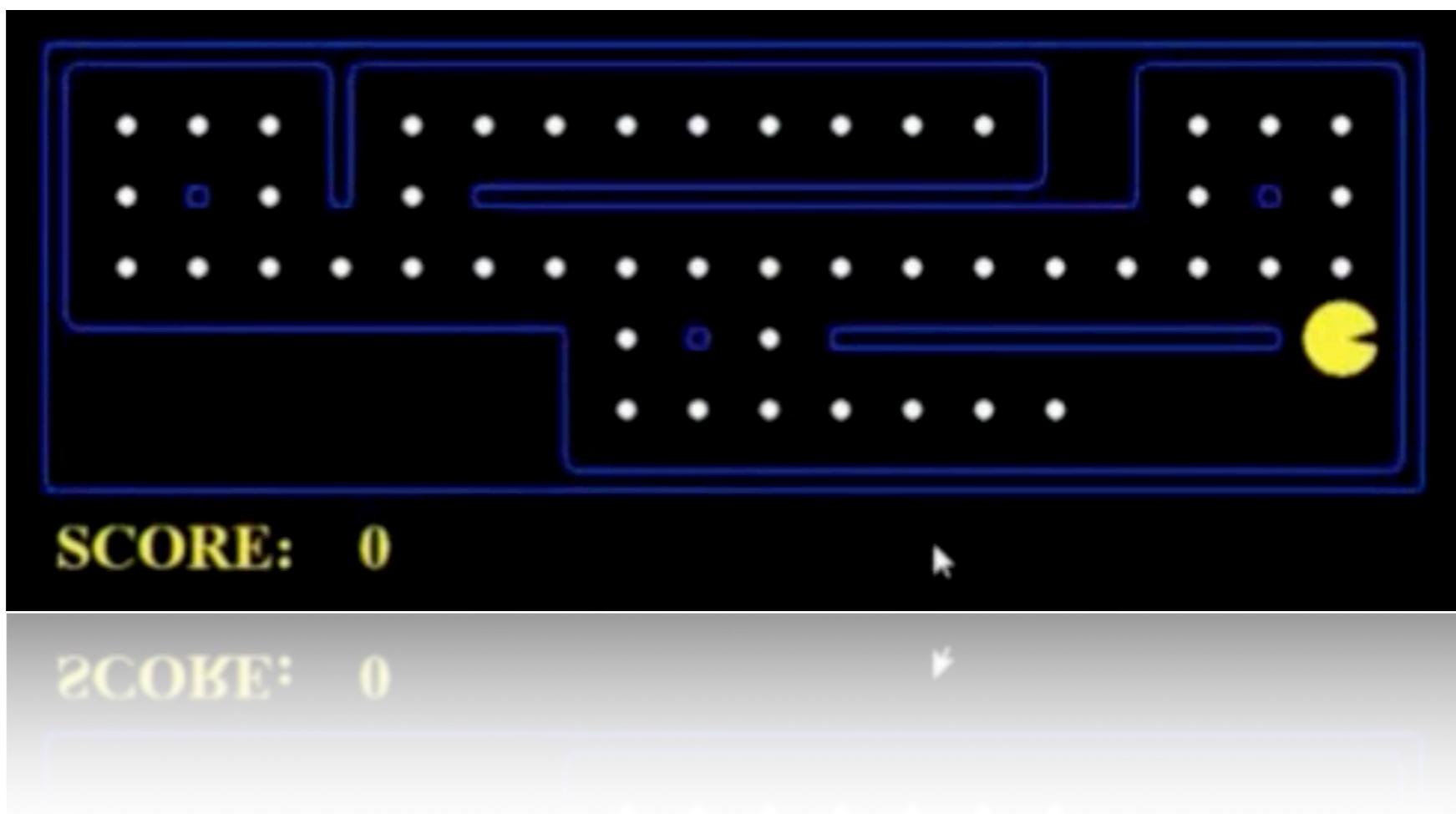
- 根据当前感知（可能还有记忆）选择动作
- 不要考虑他们行为的未来后果
- 只考虑世界当前是怎样的
- 可能有记忆



成功的反射智能体

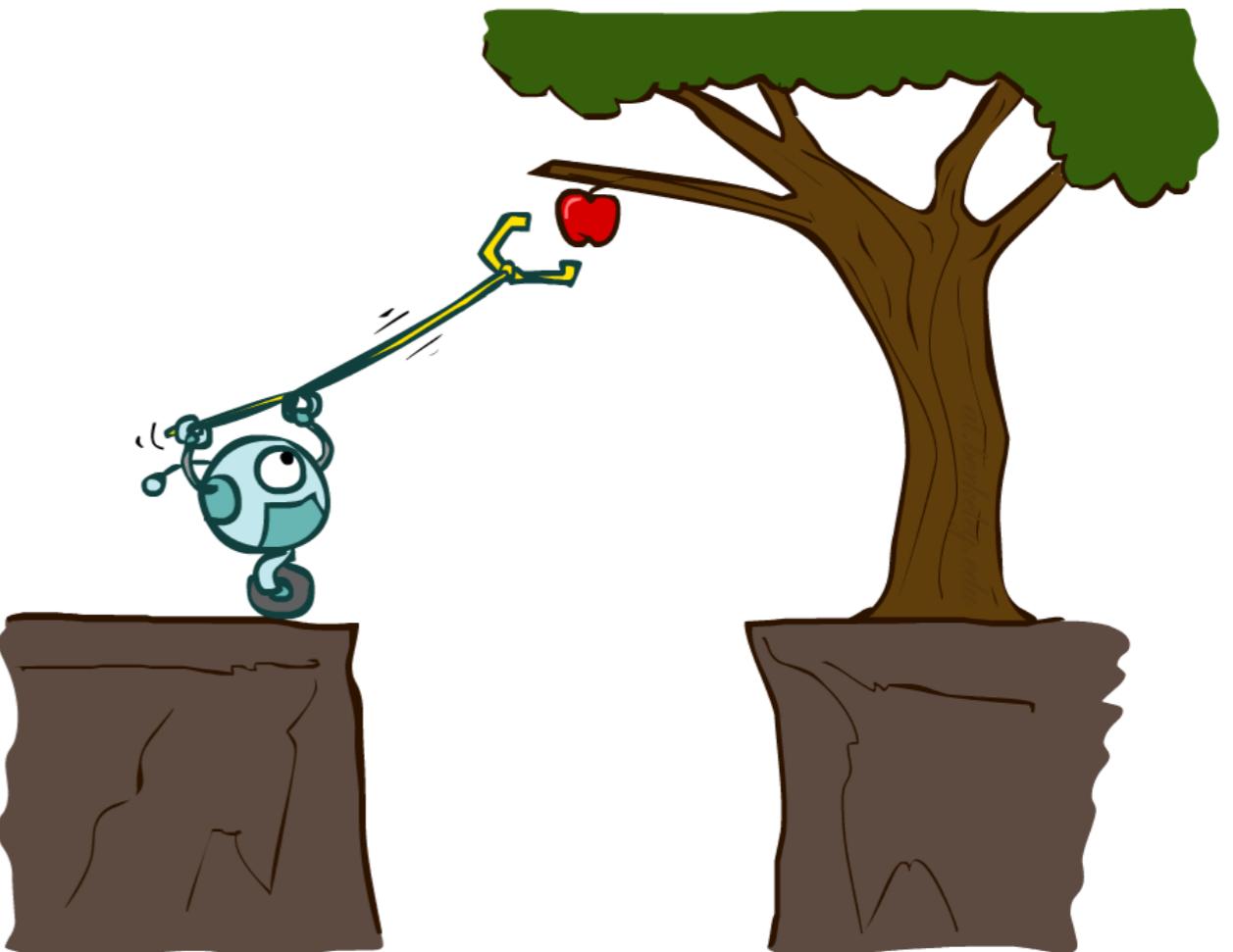


失败的反射智能体

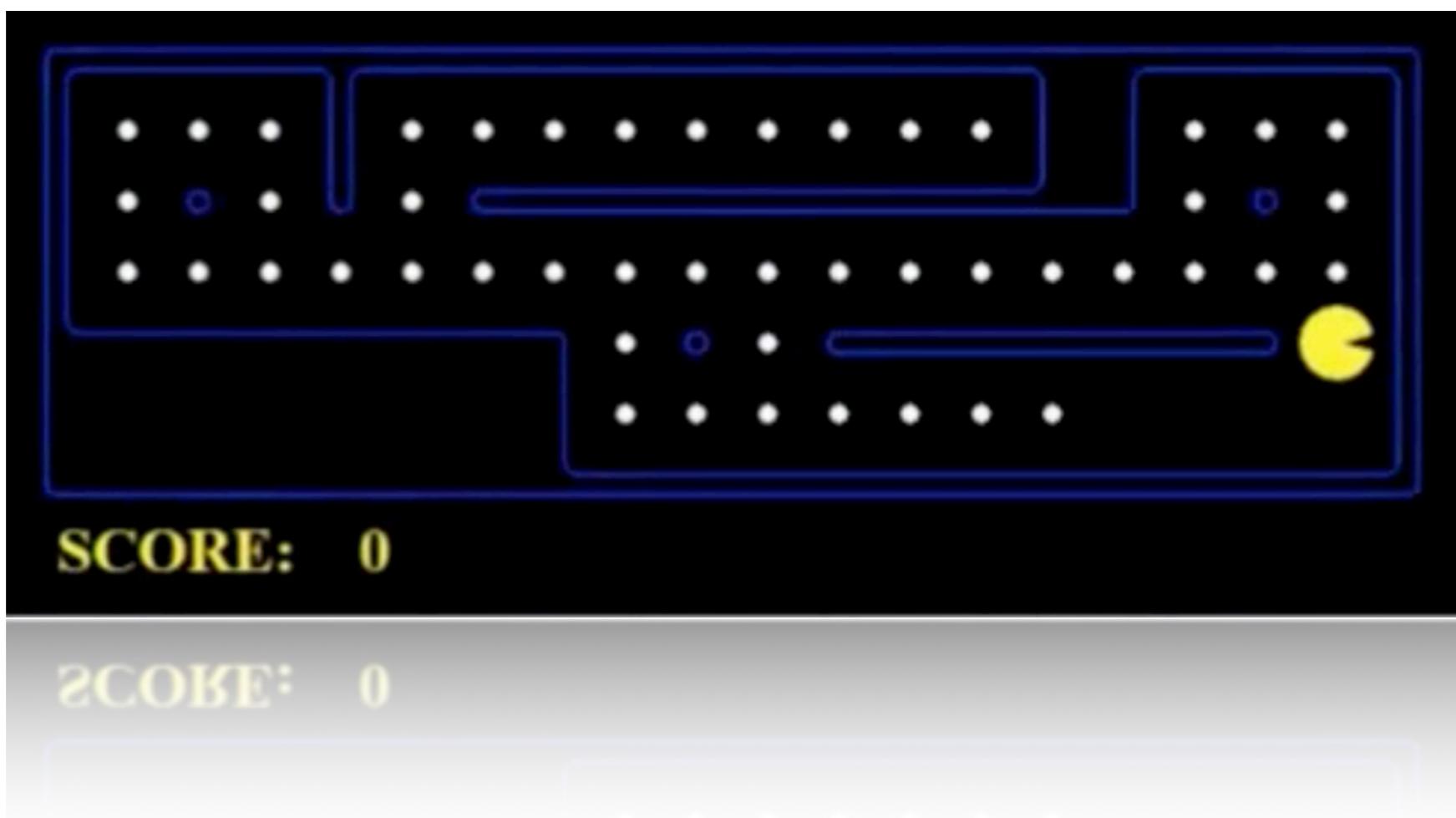


有计划的智能体

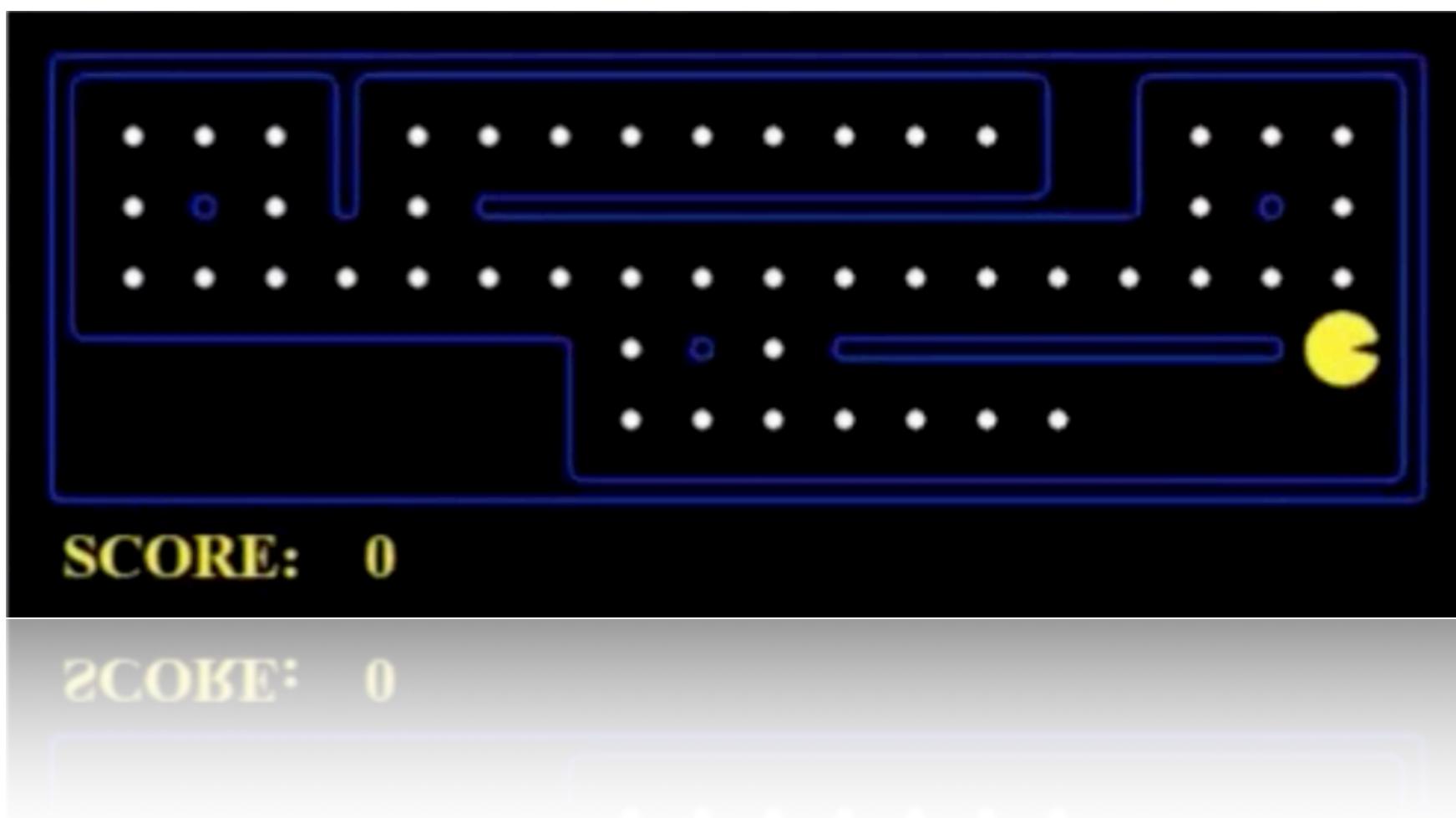
- 决策基于对行动的预测结果
- 一定有一个转换模型 (transition model) : 根据不同的行动, 环境是如何演变的
- 必须设定一个目标
- 可能的策略
 - 在开始之前制定一个完整、优化的计划 (offline plan), 然后再执行。
 - 先制定一个简单的计划, 开始执行; 当走错的时候再重新计划。



前往最近豆子的Pacman



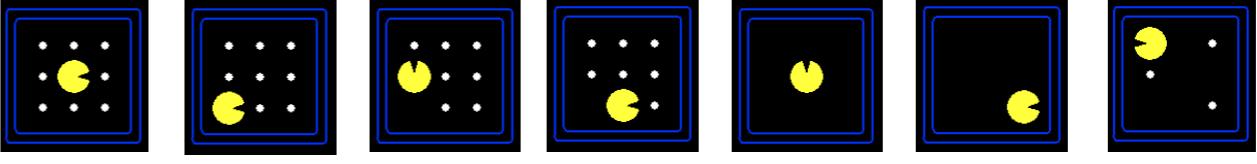
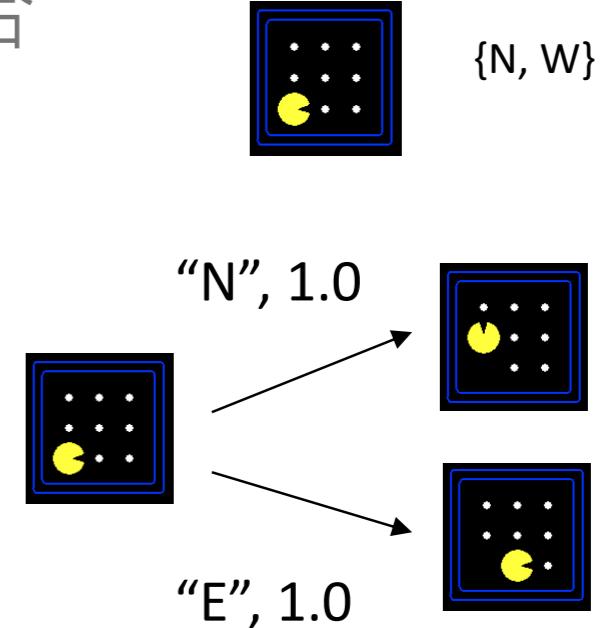
提前计划完整路径的Pacman



搜索问题



搜索问题

- 一个搜索问题包含：
 - 一个状态空间：
 - 在每一个状态里，一个可允许的动作集合
 - 一个转换模型 $\text{results}(s, a)$
 - 一个步骤成本函数 $\text{cost}(s, a, s')$
 - 一个开始状态，和一个目标到达测试
 - 一个解决方案是一系列动作（一个规划），从开始状态到一个目标状态
- 

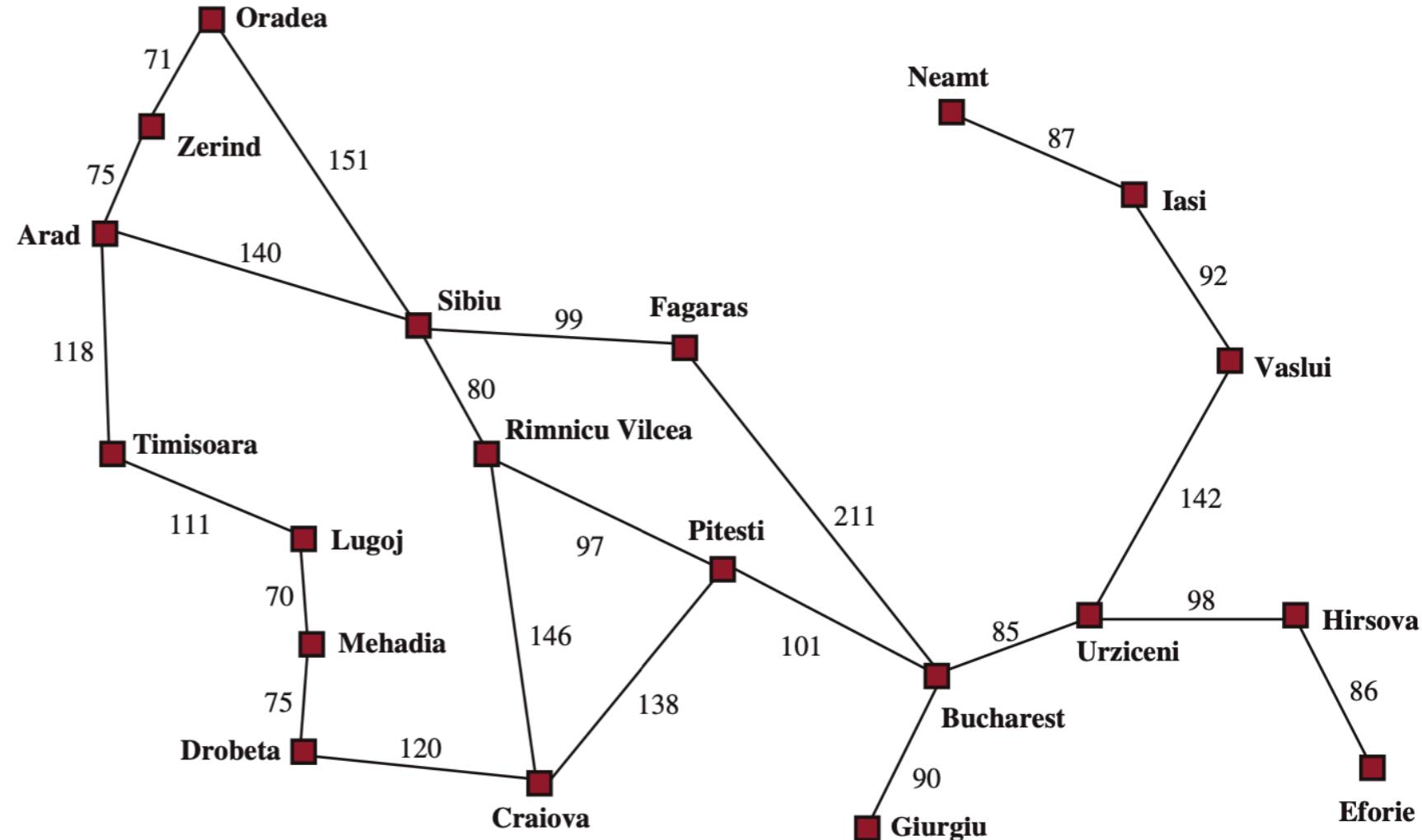
搜索问题

- 定义搜索问题是一个建模的过程
 - 抽象的数学描述



例子：罗马尼亚旅行问题

- 状态空间：城市
- 开始状态：Arad
- 动作：开车前往下一个城市
- 成本：距离（时间？过路费？）
- 目标：is state == Bucharest



状态空间

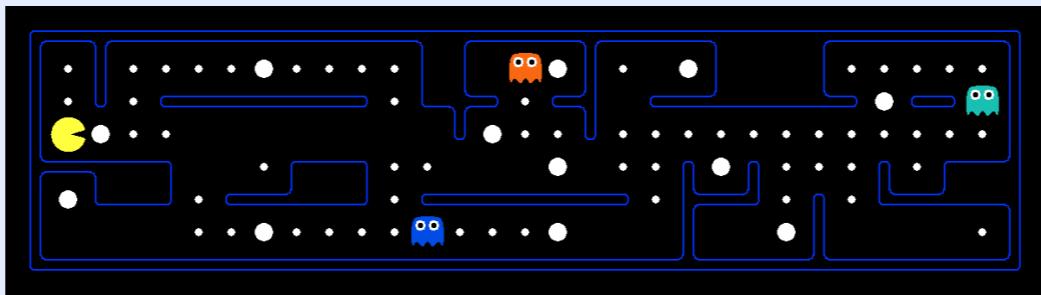
- 真实世界里的状态包括许多细节

World State Space \neq Search Problem State Space

- 搜索问题定义的状态是一种抽象，去掉不必要的细节

状态空间

真实世界里的状态包括许多细节



搜索问题定义的状态是一种抽象，去掉不必要的细节

问题: 寻路

状态: (x,y) 位置

动作: NSEW

转换函数: 改变位置

目标测试: $\text{is } (x,y)=\text{END}$

问题: 吃完所有豆子

状态: $\{(x,y)\text{位置, 豆子是否还在}\}$

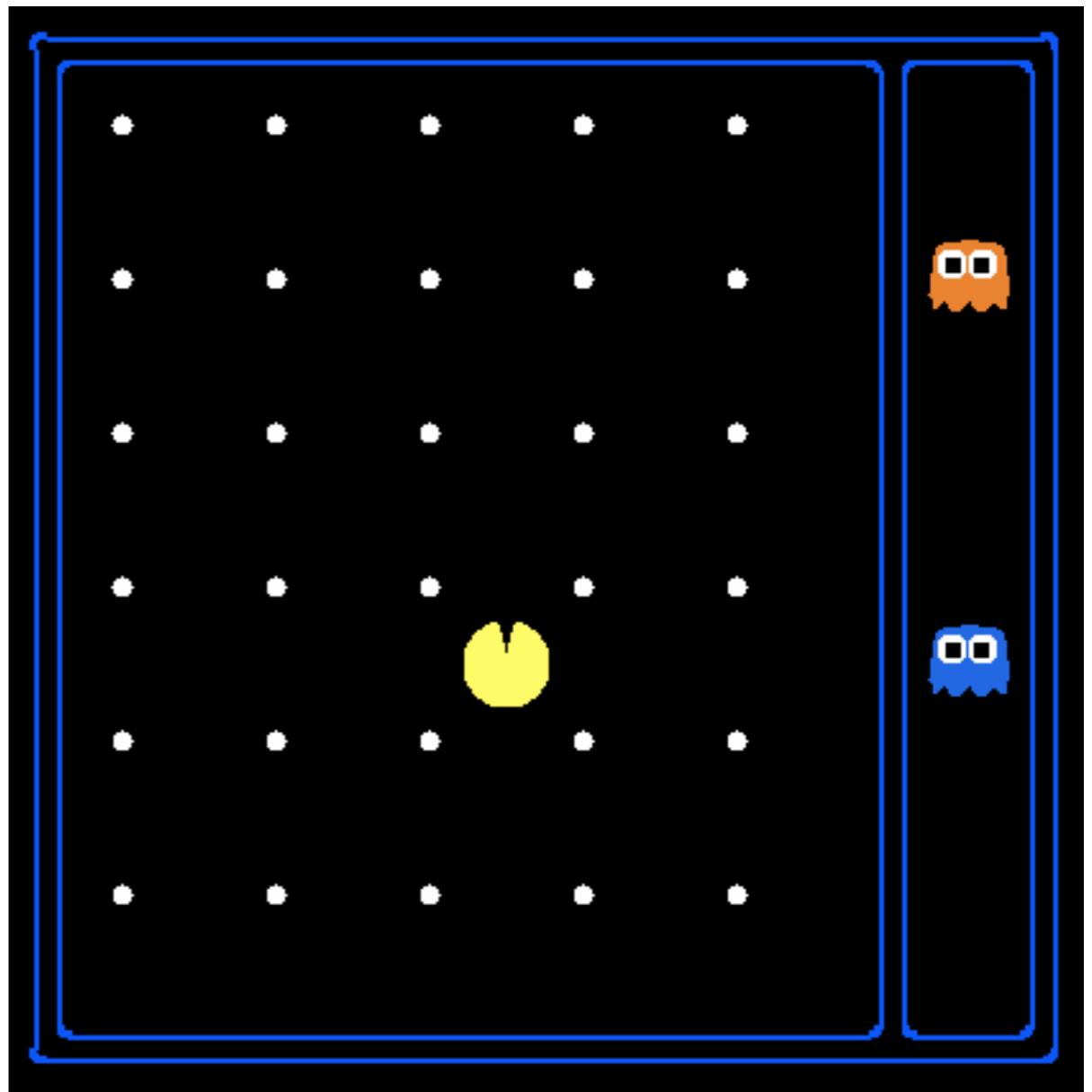
动作: NSEW

转换函数: 改变位置和豆子

目标测试: 豆子是否都还在

状态空间大小

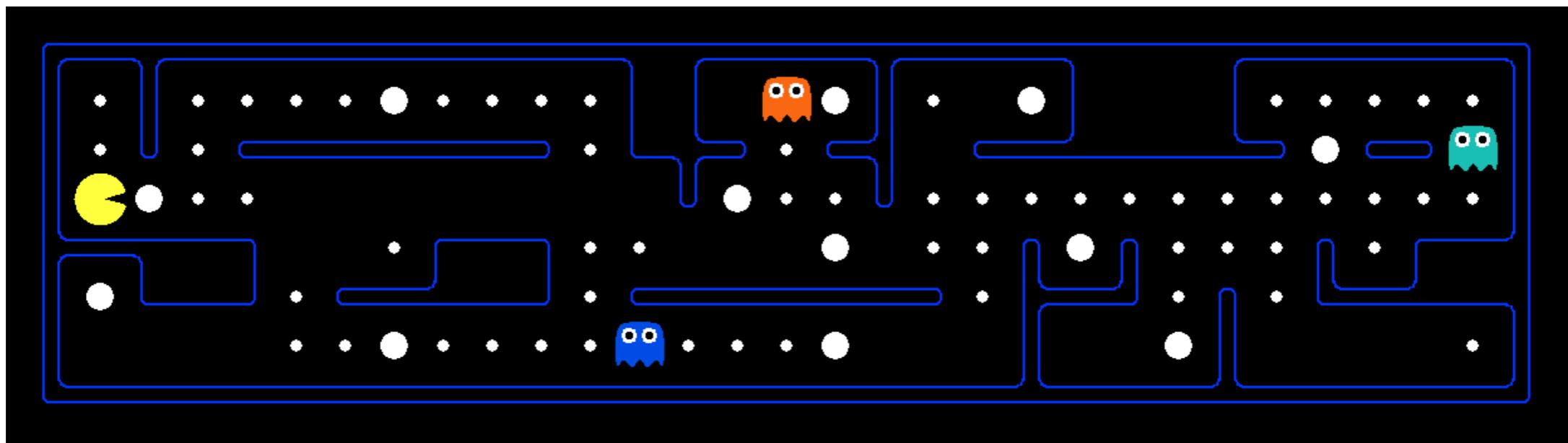
- 世界设置:
 - Pacman的位置: 120
 - 豆子数量: 30
 - 妖怪位置: 12
 - Pacman的方向: NSEW
- 有多大
 - 世界状态空间?
 $120 \times (2^{30}) \times (12^2) \times 4$
 - 寻路问题的状态空间?
120
 - 吃豆子问题的状态空间?
 $120 \times (2^{30})$



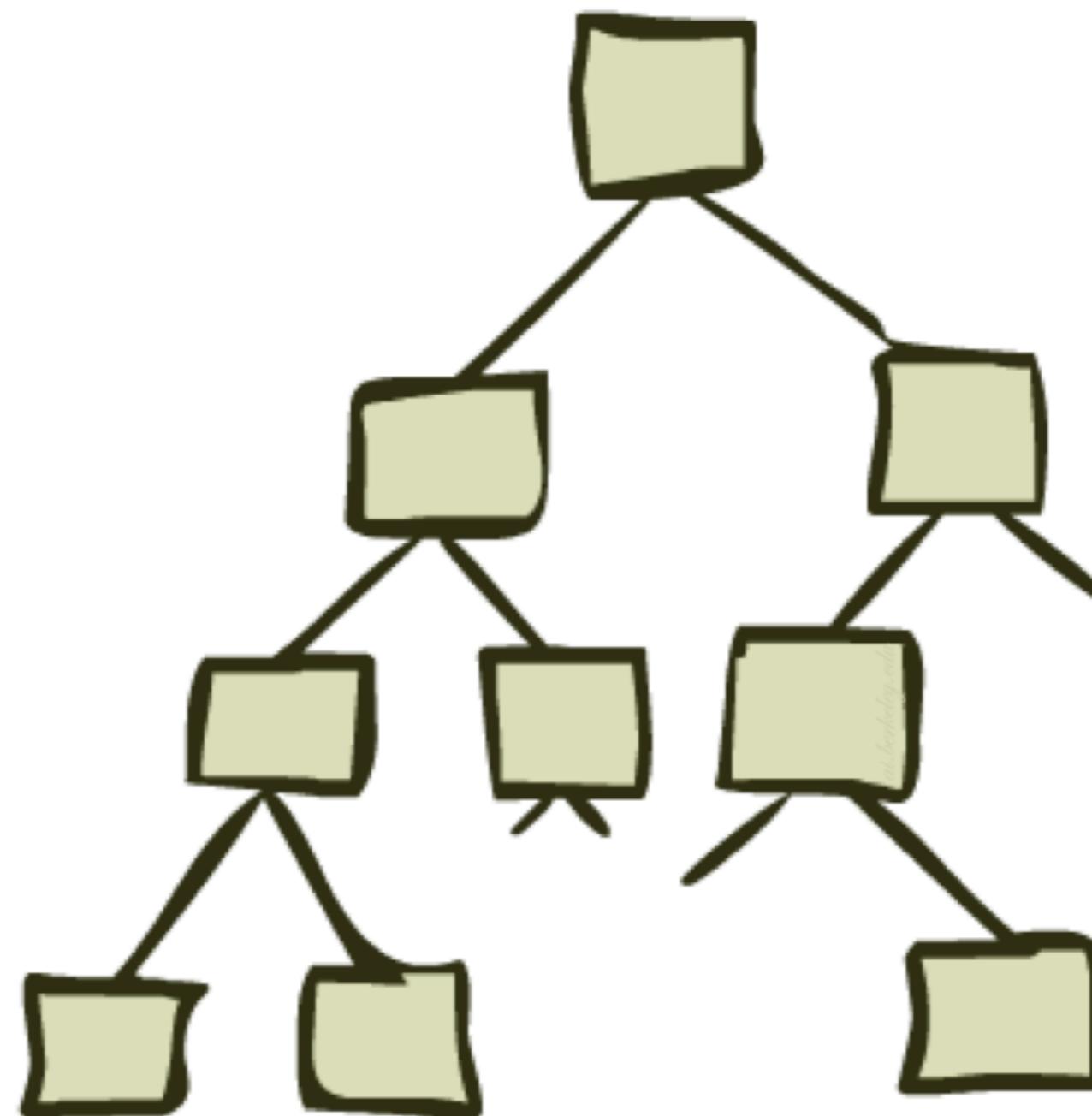
更复杂的状态空间

- 问题：吃掉所有豆子，同时保持所有怪物都在被威慑状态（吃到大的豆子妖怪会失去力量一段时间，此时pacman可以吃掉妖怪）
- 状态空间如何定义？

Pacman位置，豆子布尔值，能量丸（大豆子）布尔值，剩余威慑时间

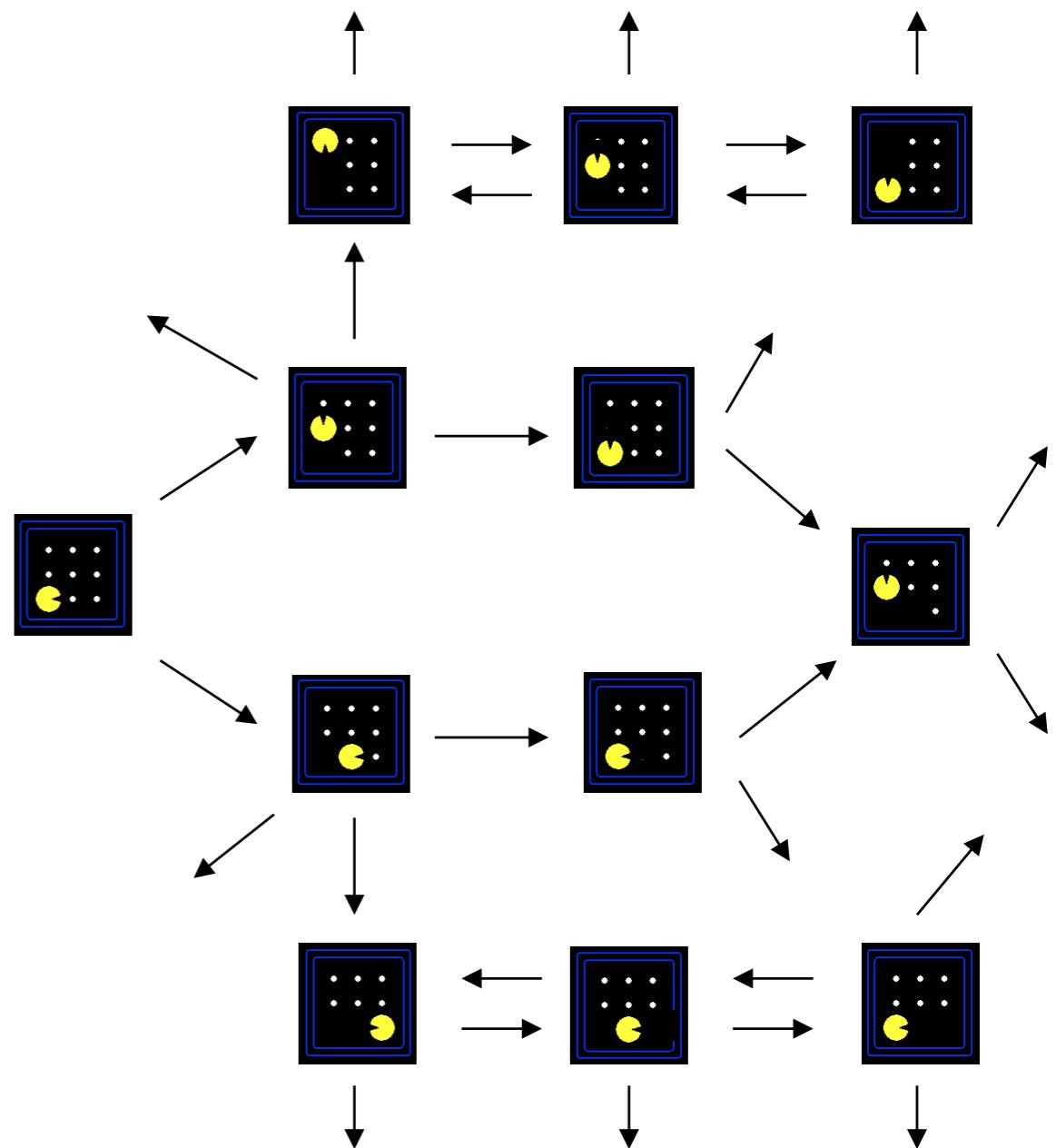


状态空间图和搜索树

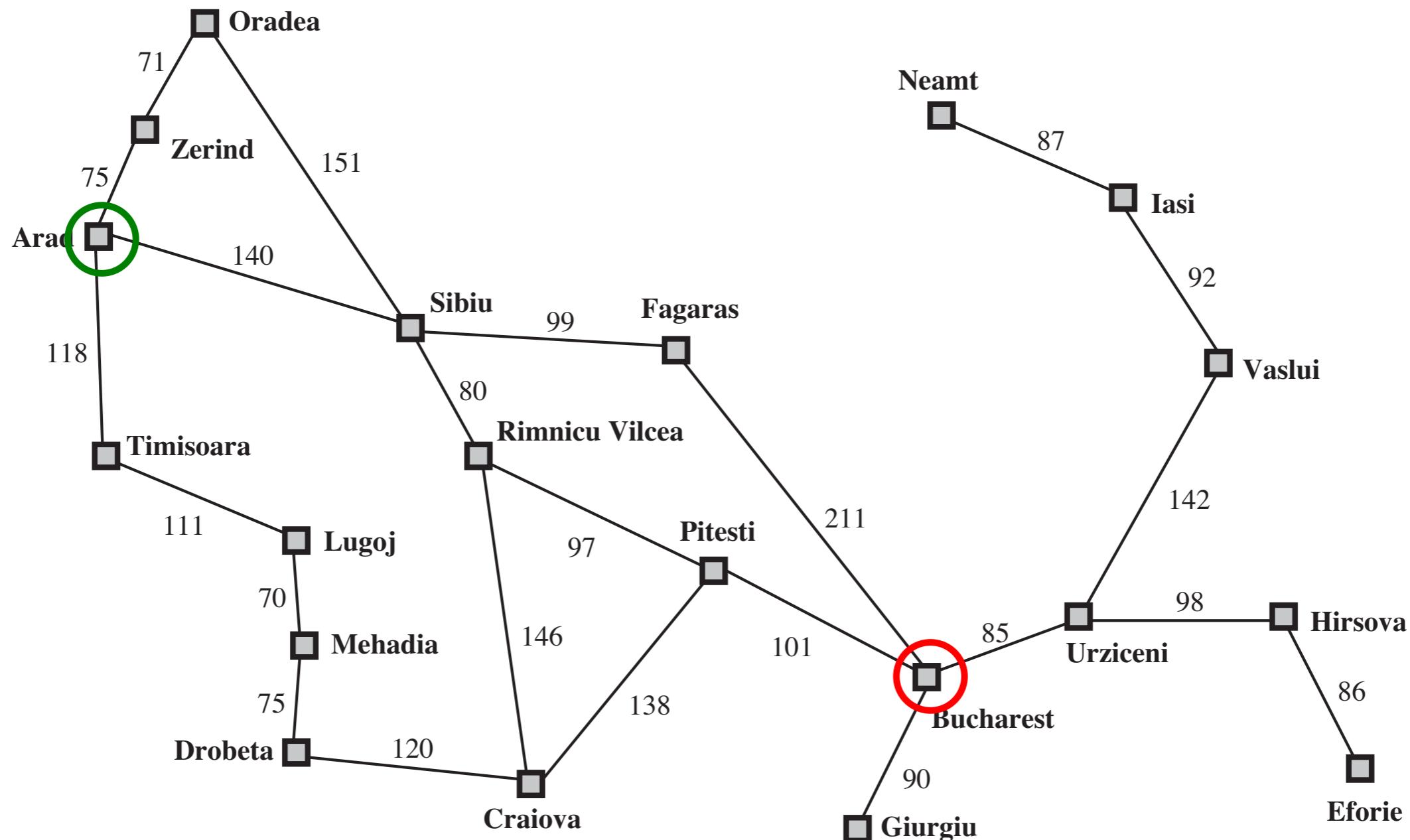


状态空间图

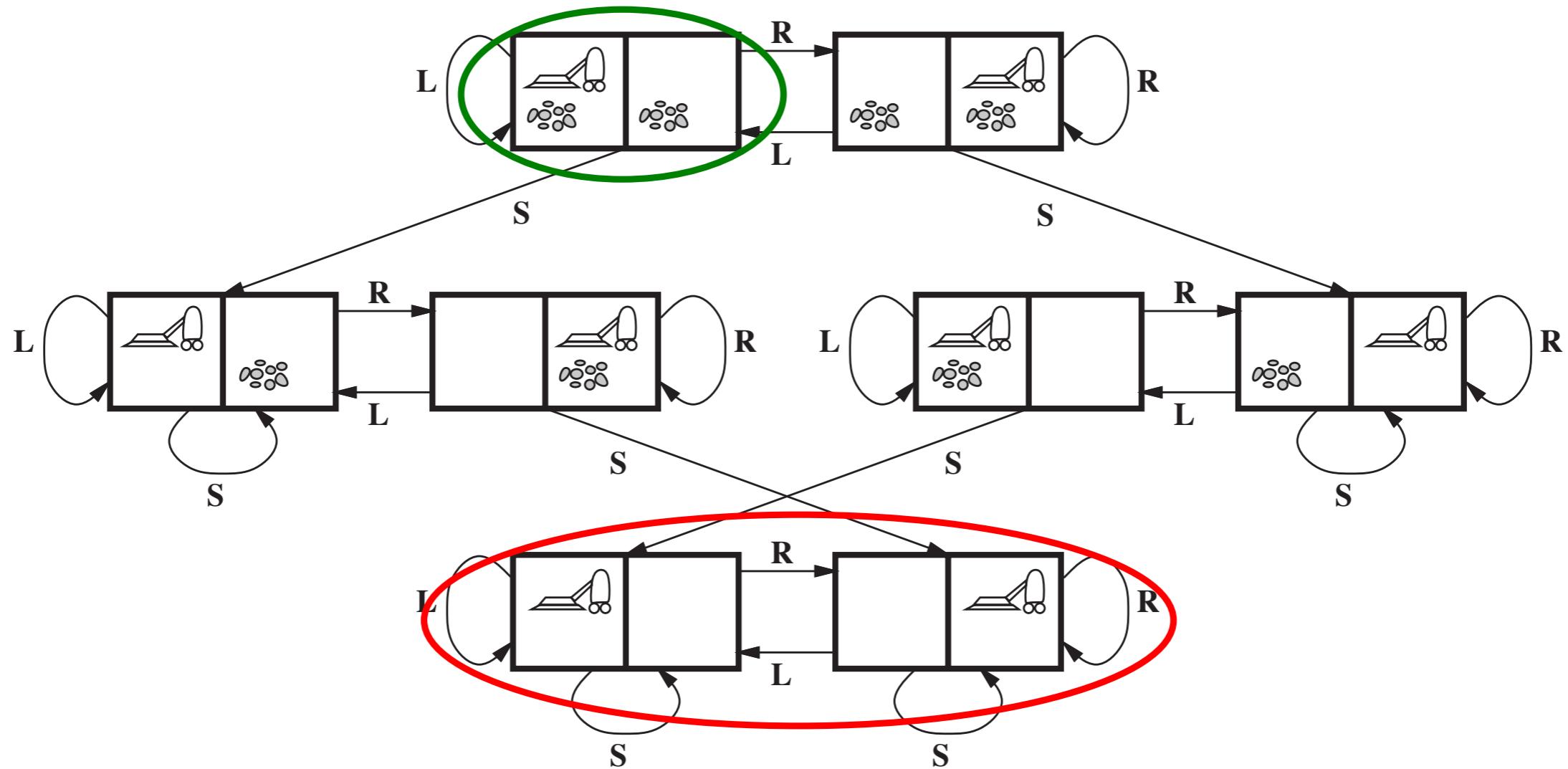
- 对搜索问题的数学表达
 - 节点是搜索问题中定义的状态
 - 边代表动作所导致的转换
 - 目标测试是当前节点是否是目标节点
- 每个状态只出现一次!
- 状态图有时很大难以完全构建，但它是一个有用的概念。



罗马尼亚旅行问题的状态图

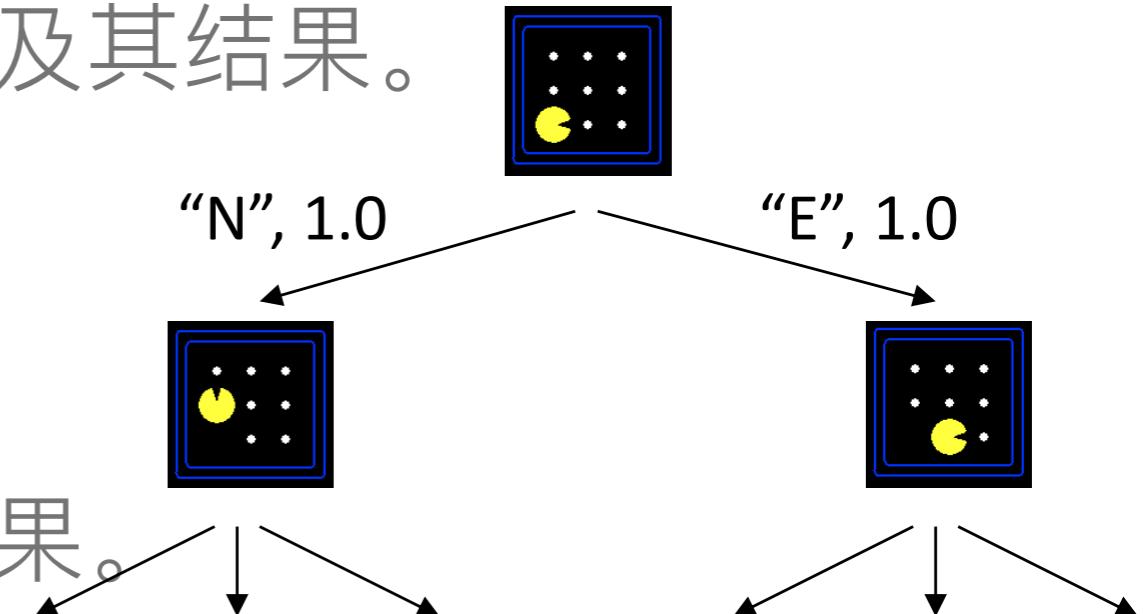


吸尘器世界问题的状态图

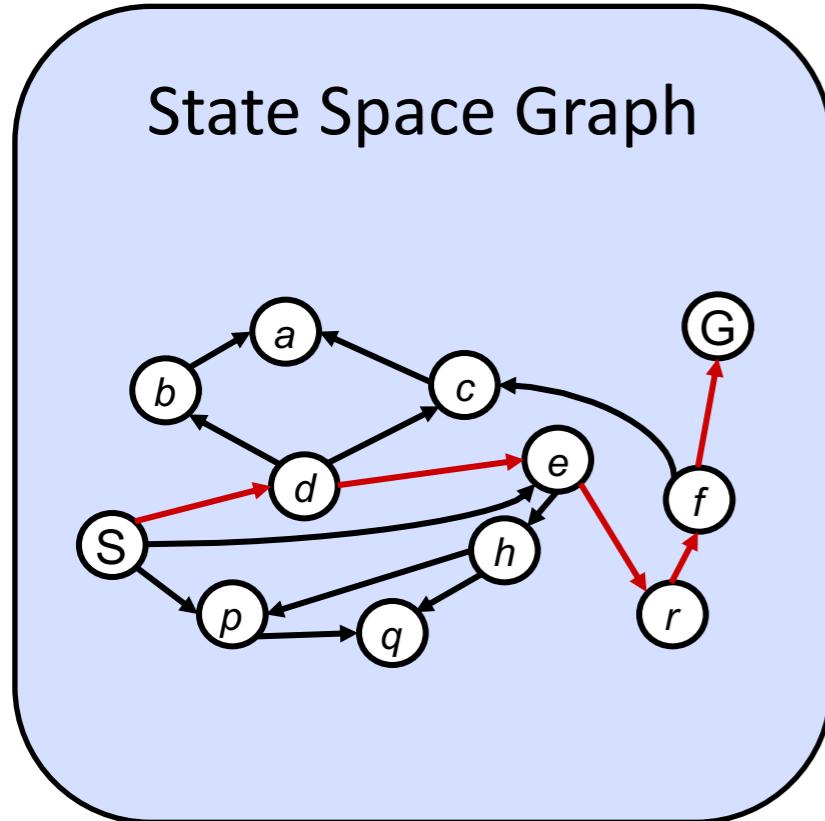


搜索树

- 树上反映的是可能的行动规划及其结果。
- 开始状态是根节点。
- 子节点反映的是可能的动作结果。
- 节点代表状态，但一个状态可能出现多次，反映的是不同行动规划的结果。
- 对于大多数问题，我们实际上很难建立整个搜索树。

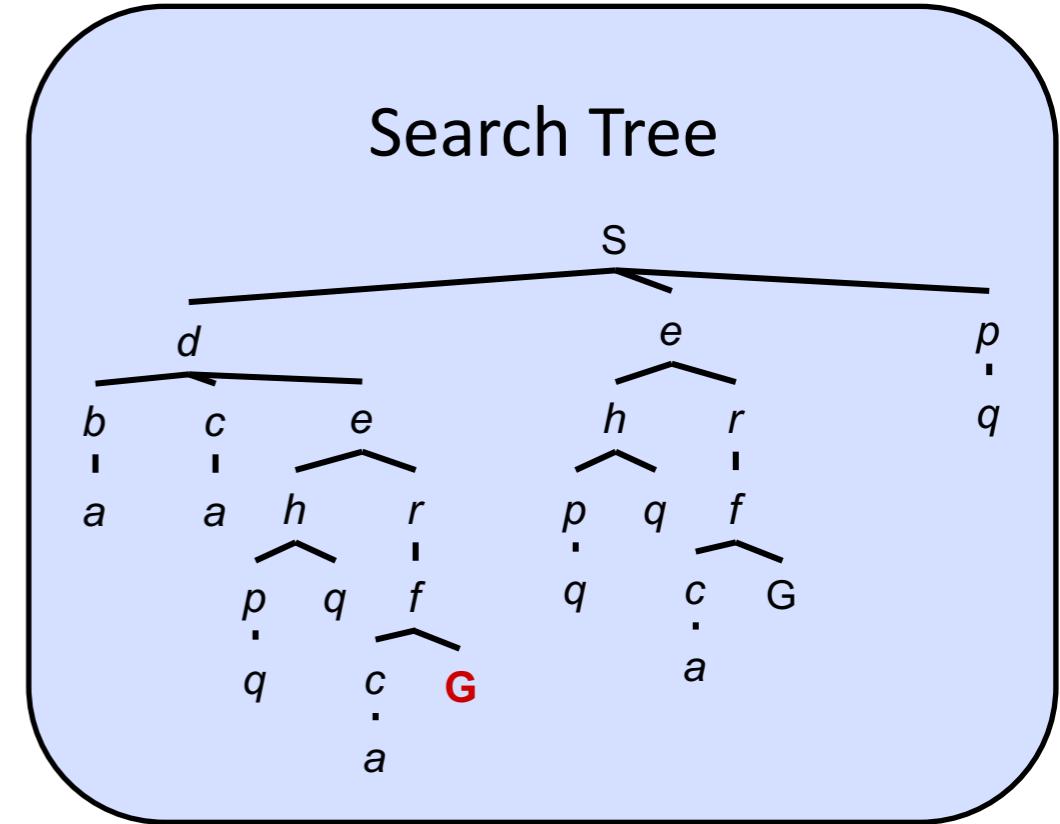


状态图 vs 搜索树



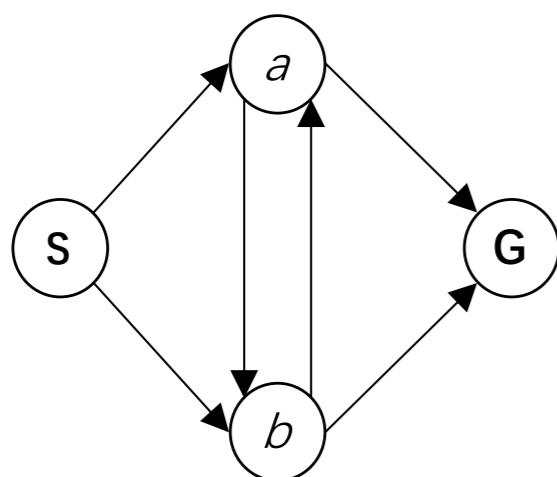
搜索树中的每个节点都对应状态空间图中的一条完整路径。

按需构建状态图或者搜索树，并且尽可能少地构建。

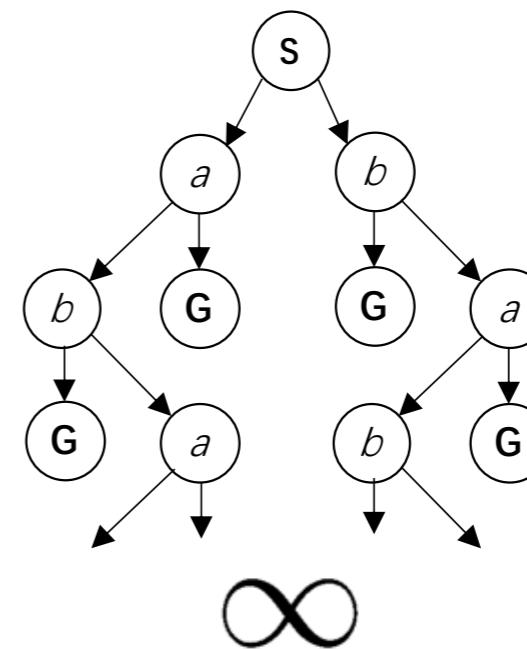


状态图 vs 搜索树

一个4节点状态图：

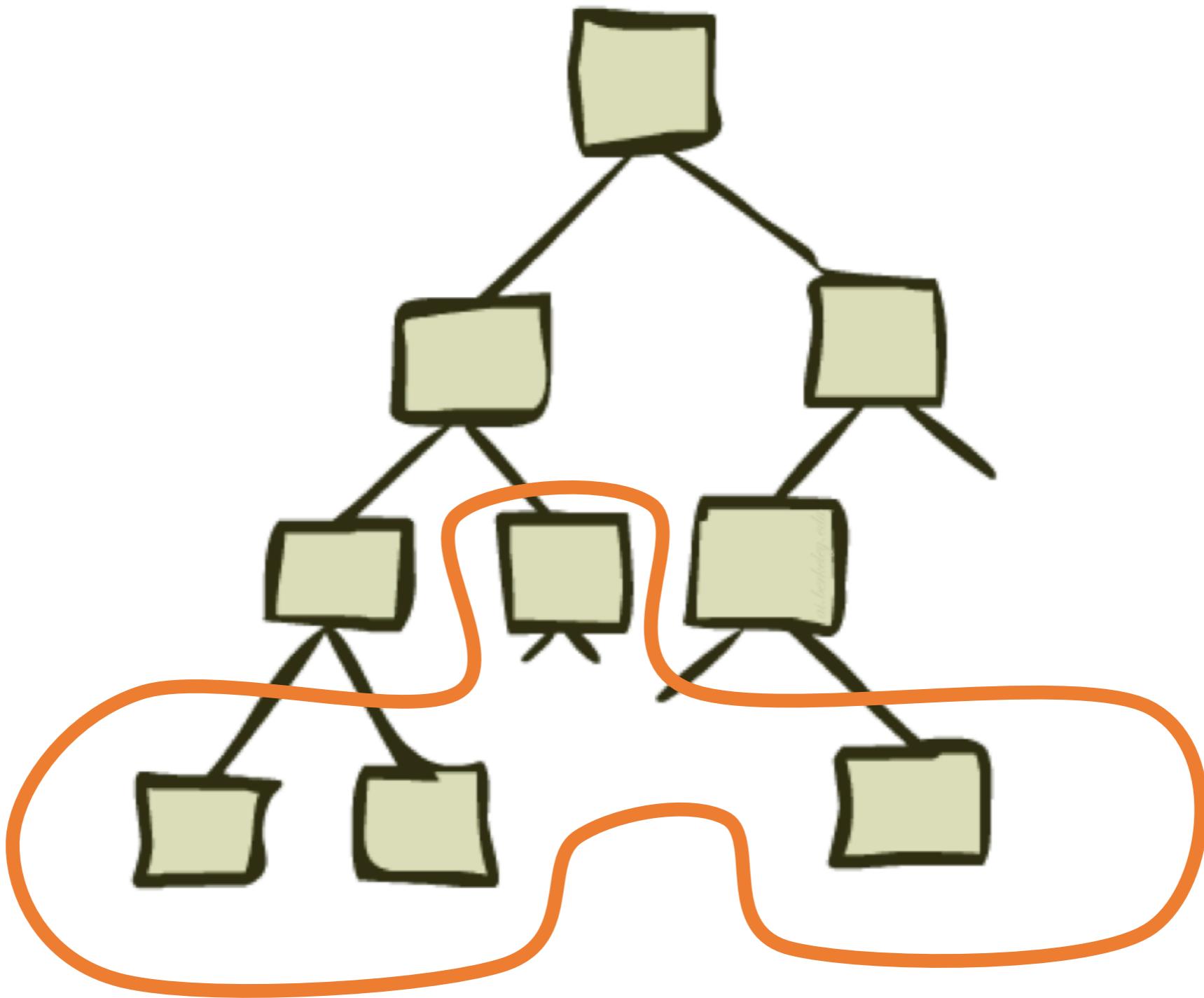


它的搜索树有多大（从S状态开始）？

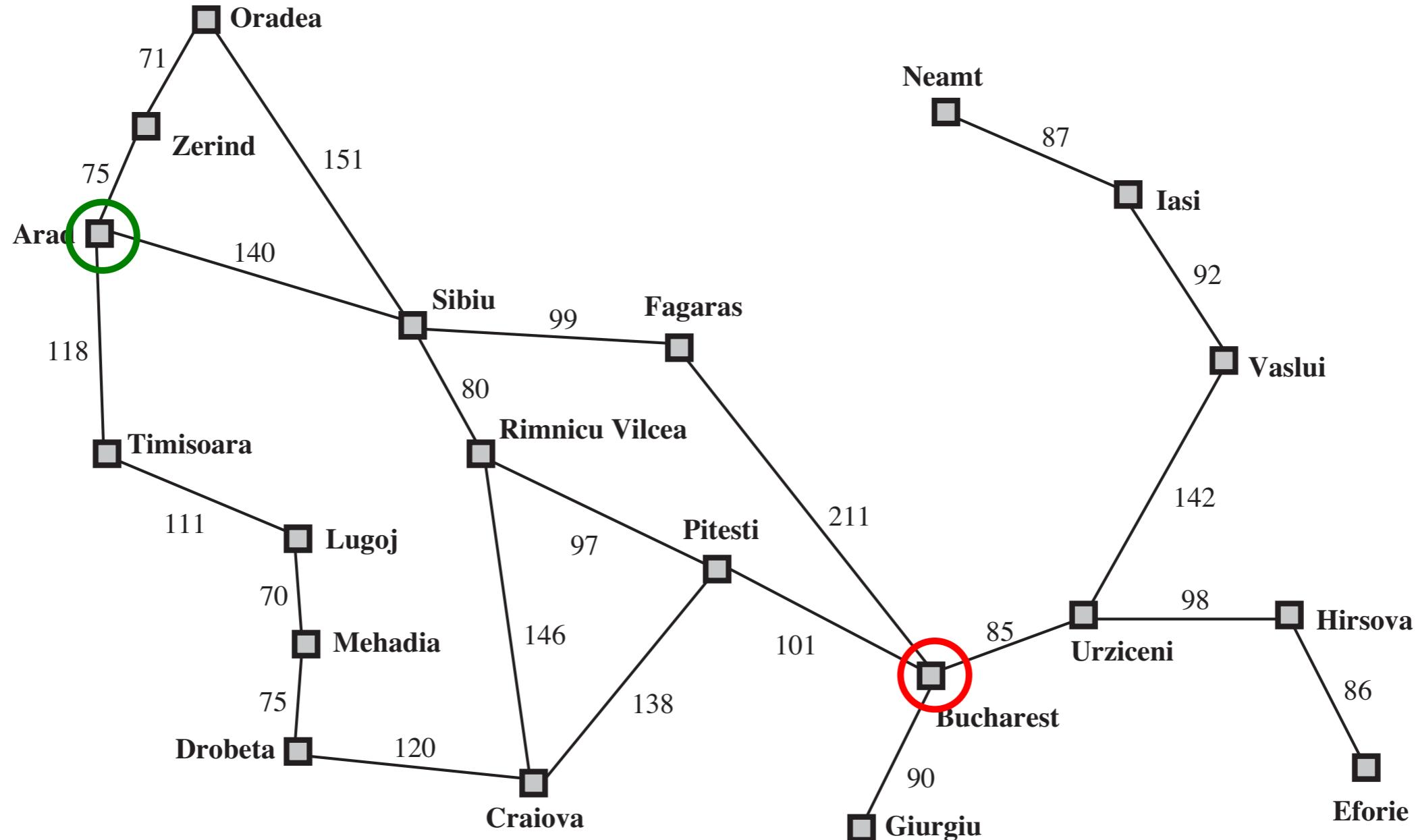


搜索树中有很多重复的结构!

树搜索

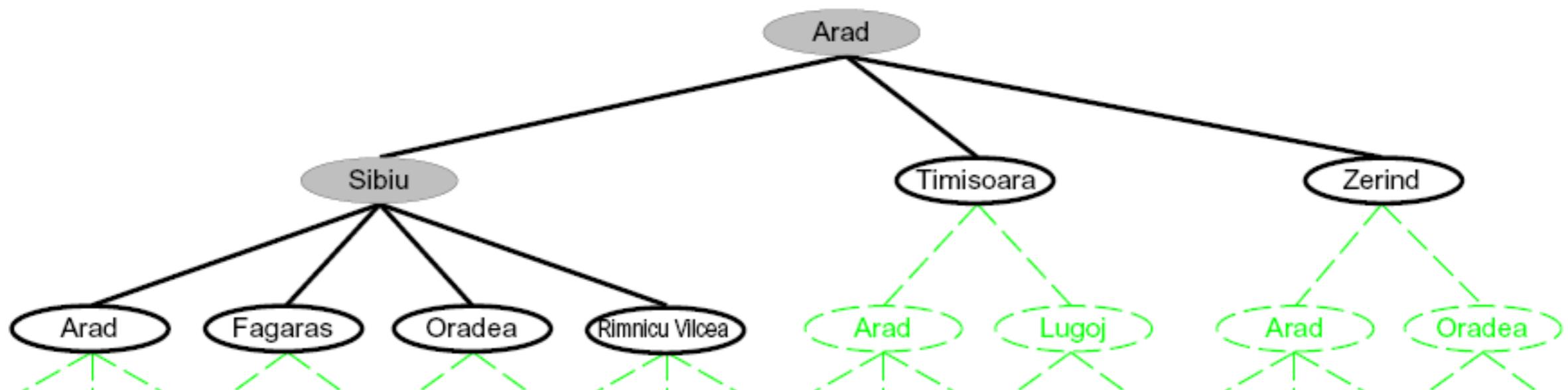


搜索问题：罗马尼亚旅行



搜索树搜索

- 扩展树节点（寻找潜在的行动规划）
- 从搜索前沿（当前的所有叶节点）中考虑扩展；前沿代表了当前所有的规划部分
- 合理的搜索策略使得树节点扩展的越少越好



通用的树搜索方法框架

- 主要问题：从哪些前沿叶节点开始探索扩展？

function 树搜索(问题) **returns** 一个解, 或失败

把问题的初始状态放入搜索前沿

loop do

if 搜索前沿 为空 **then return** 失败

 选择一个 节点 并把它从 搜索前沿 中移除

if 这个 节点 包含一个目标状态 **then return** 相应的解
(从根节点到此节点的路径)

 扩展这个选择的 节点, 把扩展结果的 节点 加入 搜索前沿

一个节点的实现

节点的属性：

state, parent, action, path-cost

A child of node by action a has

state = result(node.state,a)

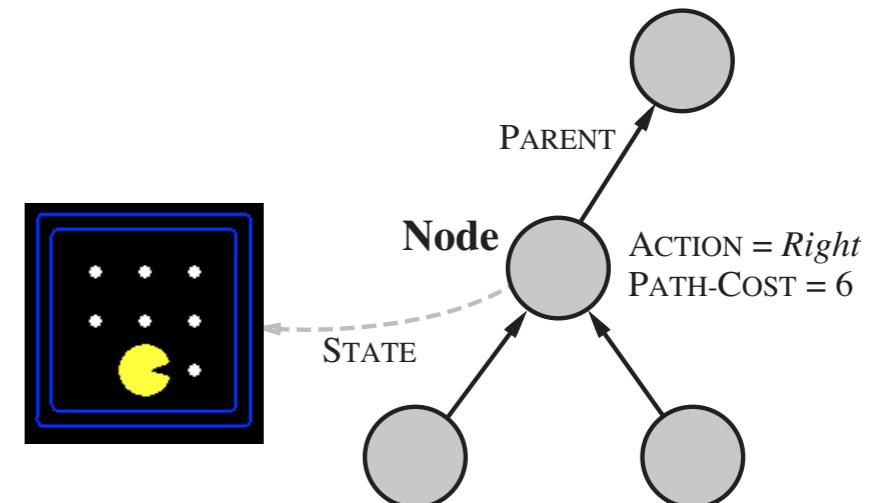
parent = node

action = a

path-cost= node.path-cost +

step-cost(node.state, a, self.state)

解的获取通过回溯父节点指针采集行动，从而获得一个行动序列，即行动规划

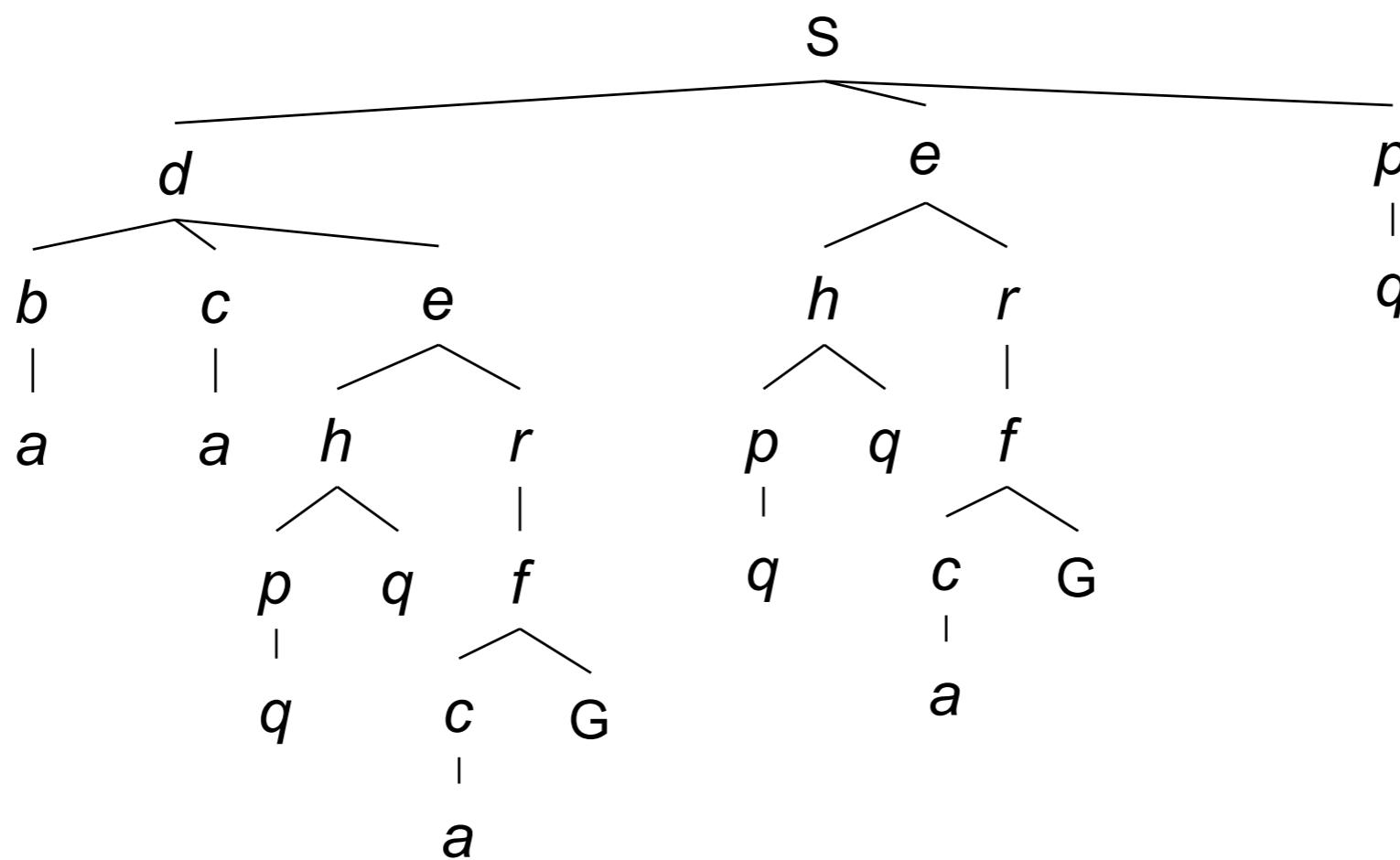
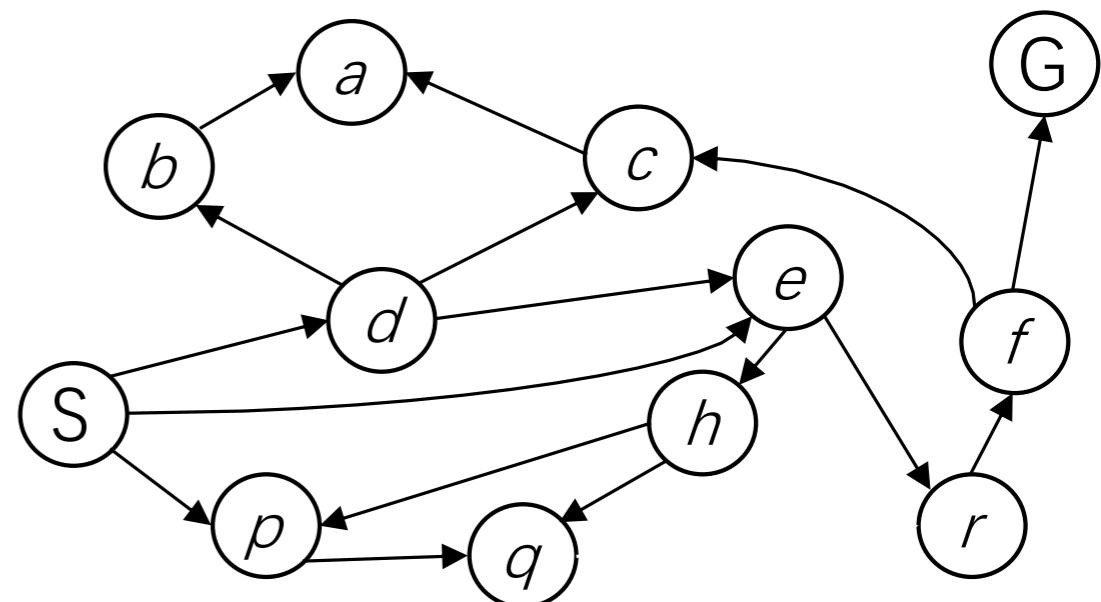


深度优先搜索



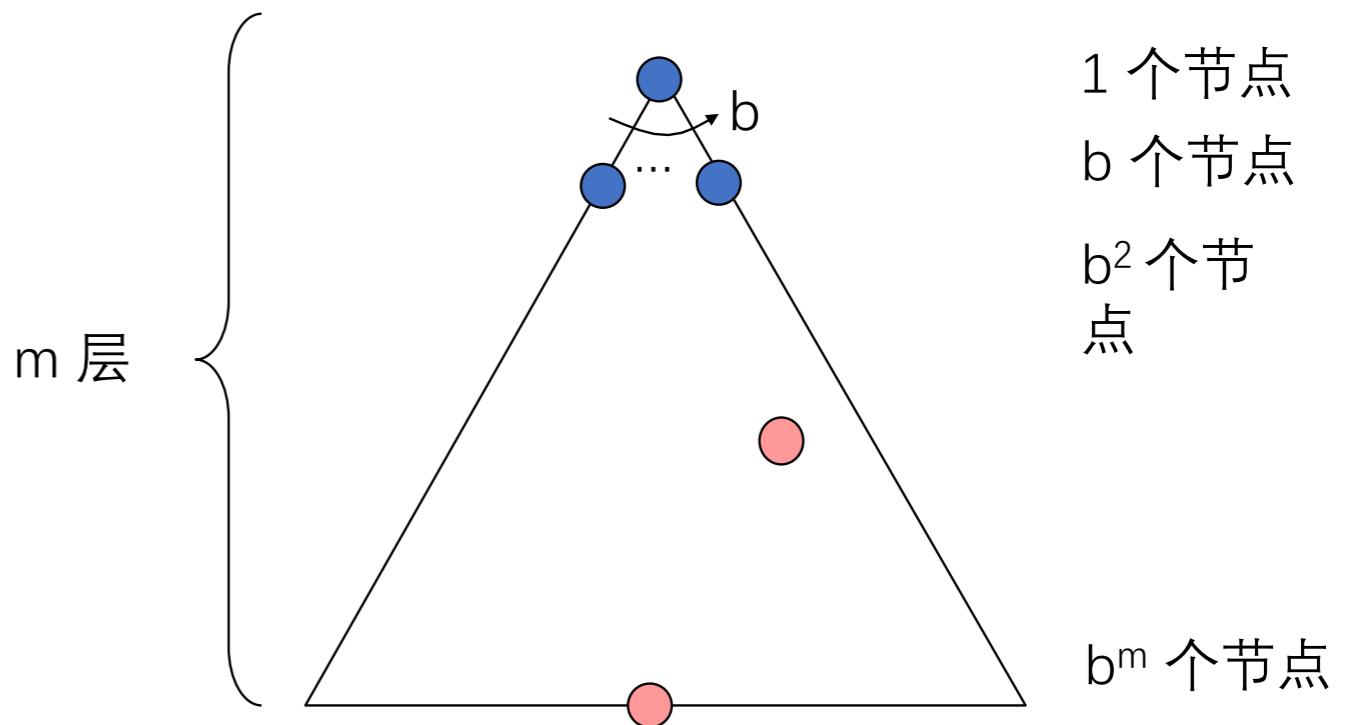
深度优先搜索

- 策略：总是最先扩展一个最深的节点
- 实现：前沿是一个后进先出的队列



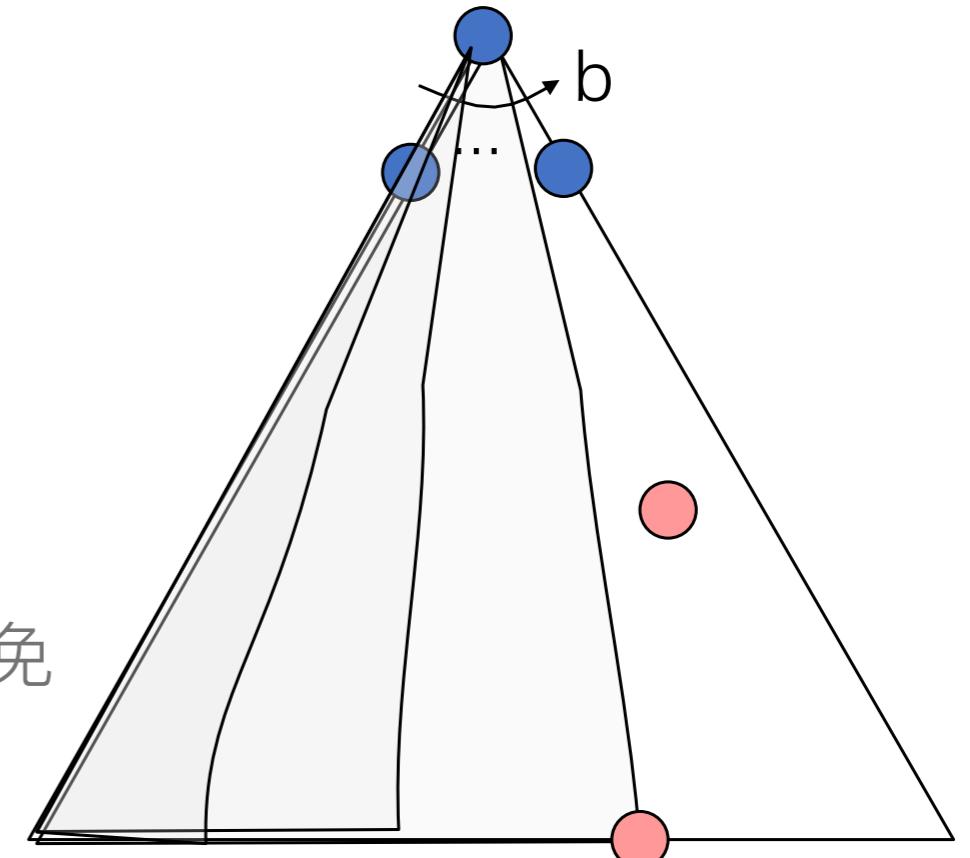
搜索算法的属性

- 完全性：保证能找到一个存在的解？
- 最优性：保证能找到最小路径成本的解？
- 时间复杂性？
- 空间复杂性？
- 搜索树：
 - b : 最大分支数
 - m : 最大深度
 - 解可能存在于不同深度
- 整个树的节点总数？
 - $1 + b + b^2 + \dots + b^m = O(b^m)$

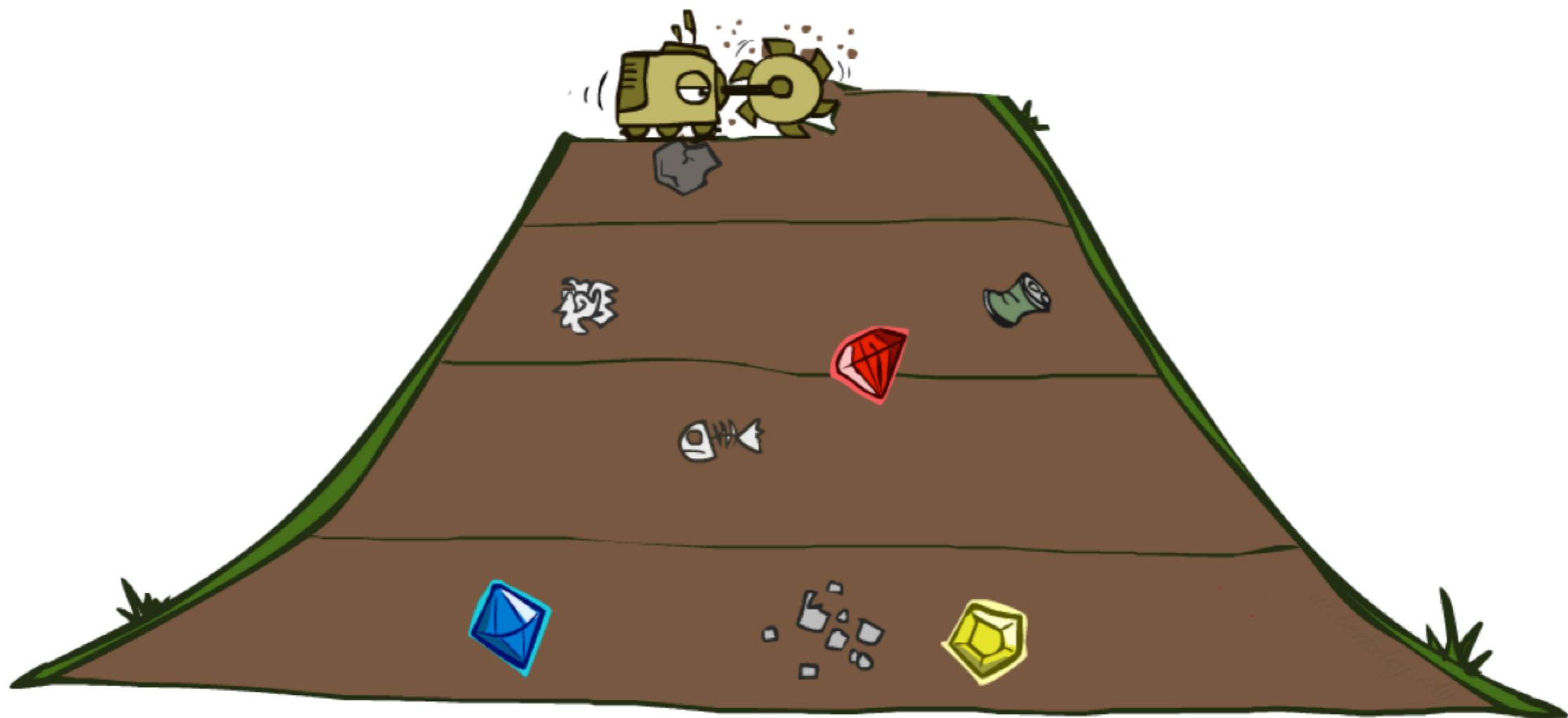


深度优先搜索 (DFS) 属性

- 哪些节点被扩展?
 - 某些左边的节点
- 可以产生整个树
 - 如果 m 是有限的, 时间复杂度 $O(b^m)$
- 存储前沿需多少空间?
 - 只存储一条路径从根到一个叶节点及沿路相关兄弟节点, 所以 $O(bm)$
- 完备性?
 - 不一定, 因为 m 可能是无穷。除非可以避免循环搜索
- 最优性?
 - 不最优。发现的可能是树最左边的一个次优解

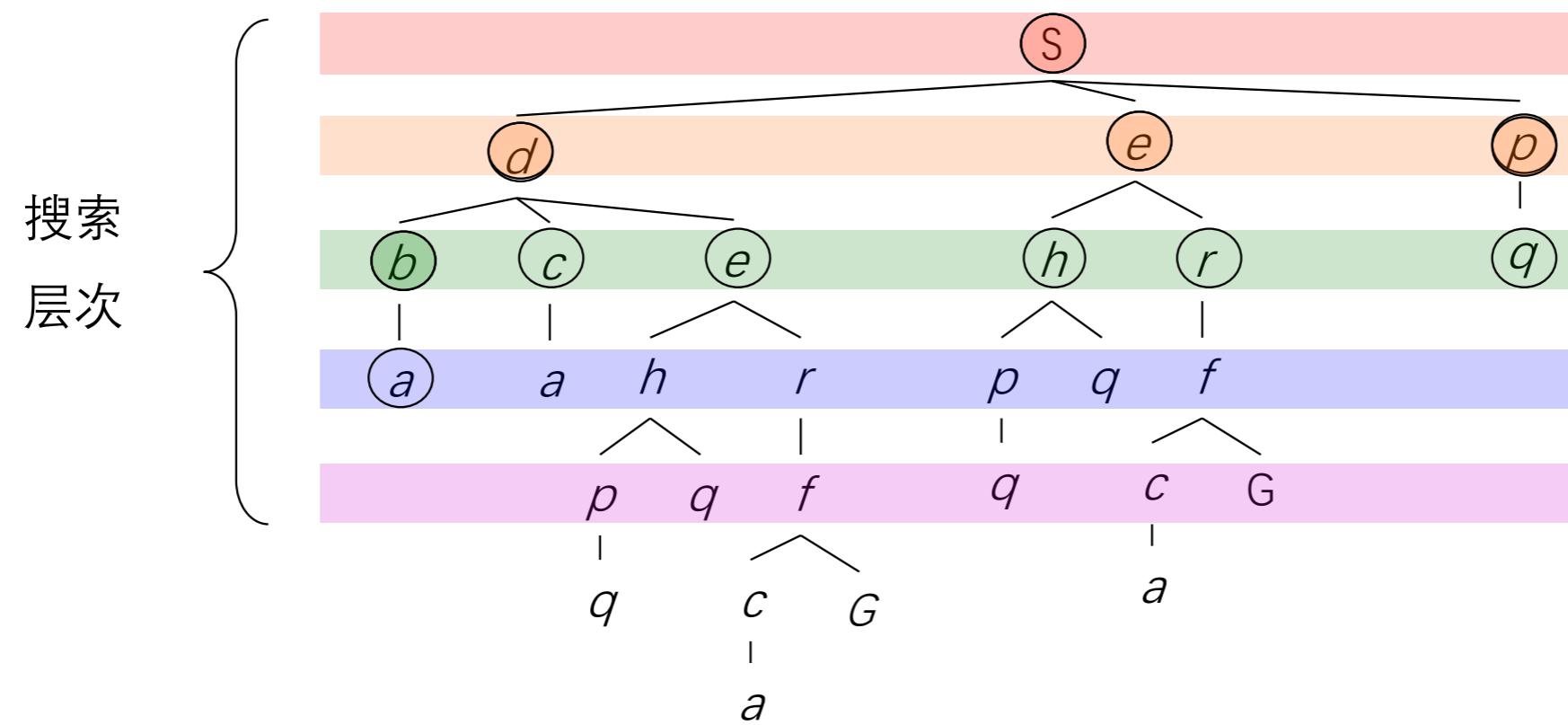
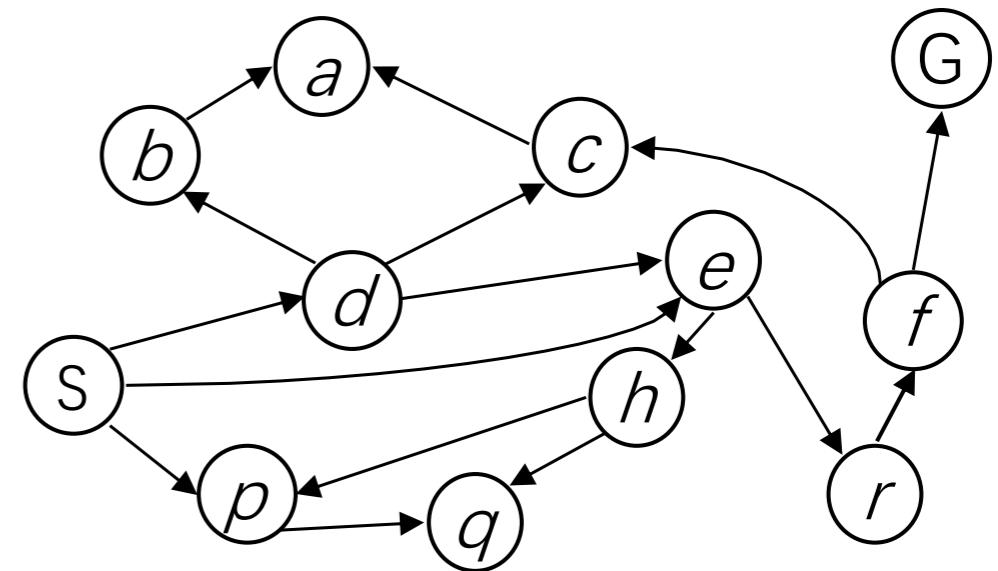


广度优先搜索



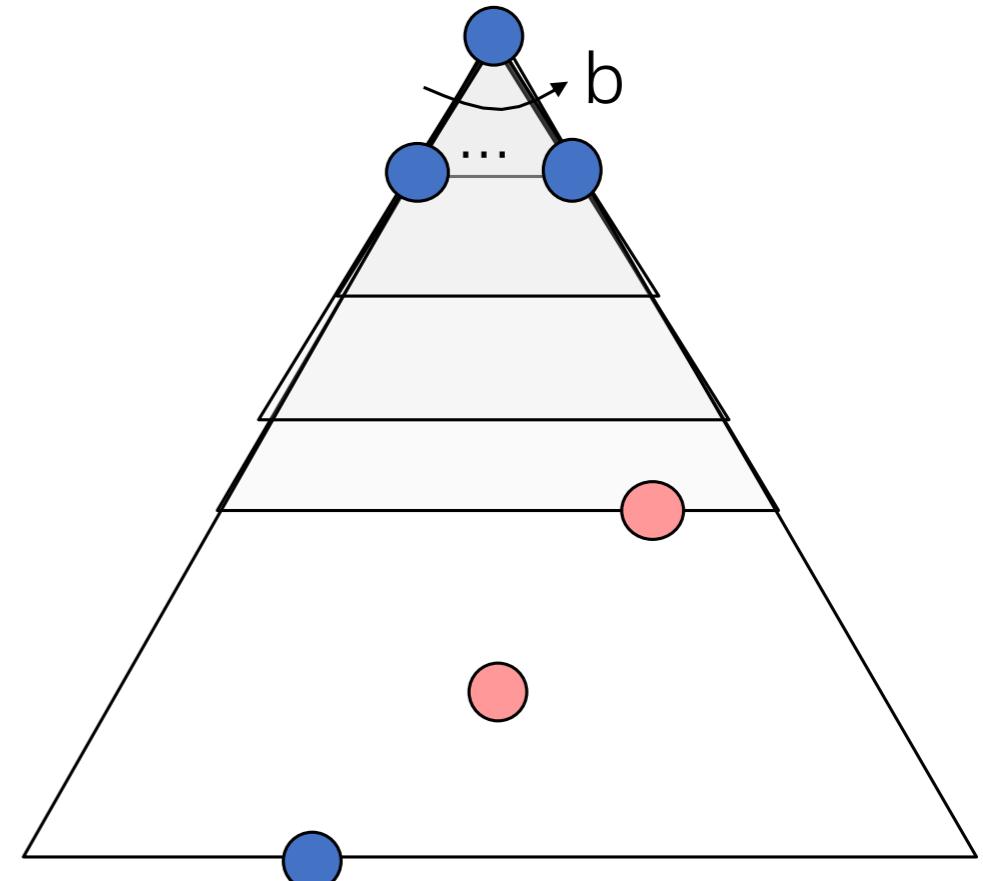
广度优先搜索

- 策略：优先扩展最浅的节点
- 实现：先进先出的队列



广度优先搜索属性

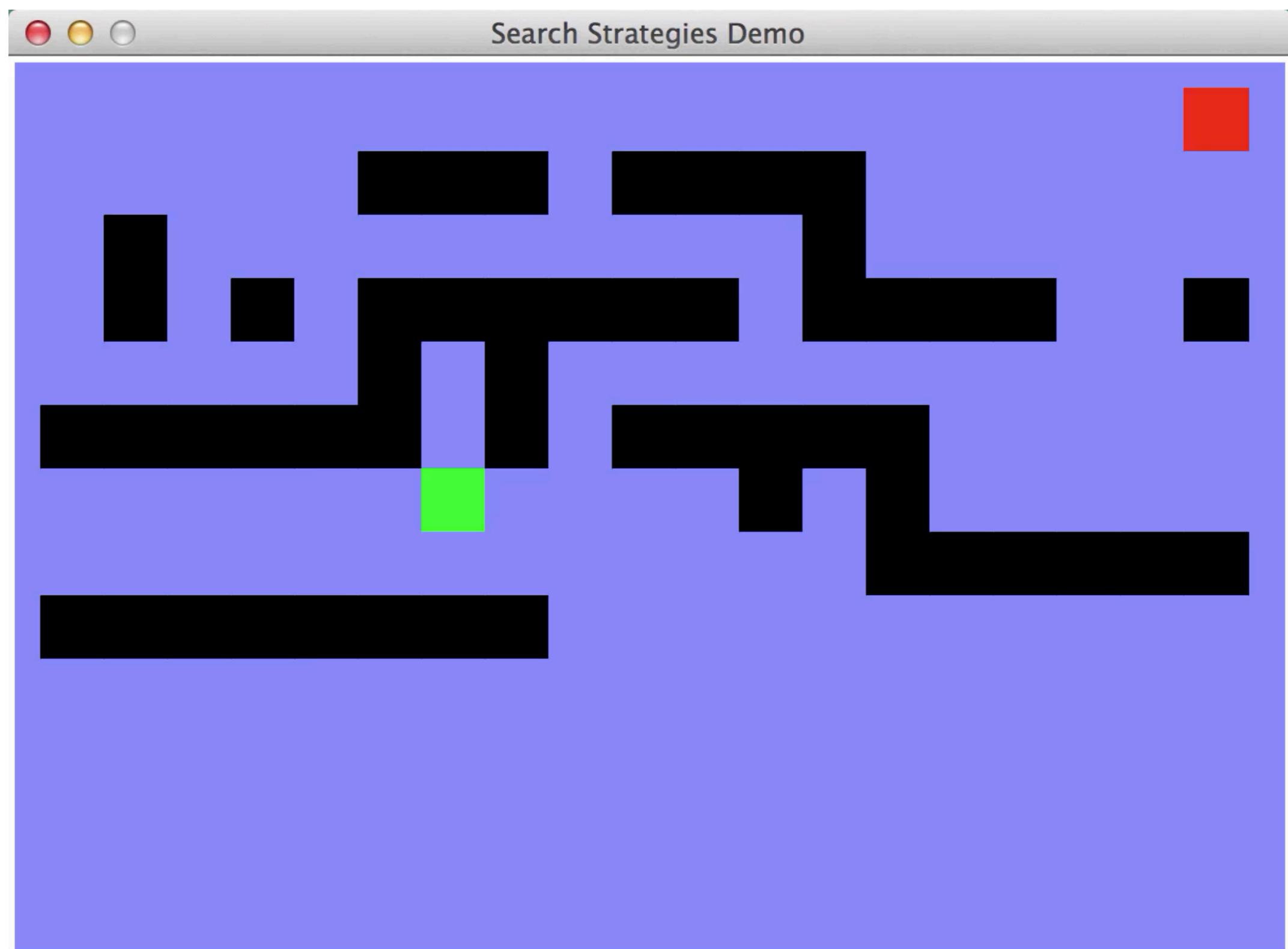
- 哪些节点被扩展?
 - 处理所有最浅层解以上的所有节点
 - 如果最浅解的深度为 s
 - 搜索时间 $O(b^s)$
- 前沿探索需要的存储空间
 - 大概是最后一层, 所以 $O(b^s)$
- 完备性?
 - 如果一个解存在, s 一定是有有限的, 所以是完备的!
- 最优性?
 - 是, 如果所有步骤成本都是1



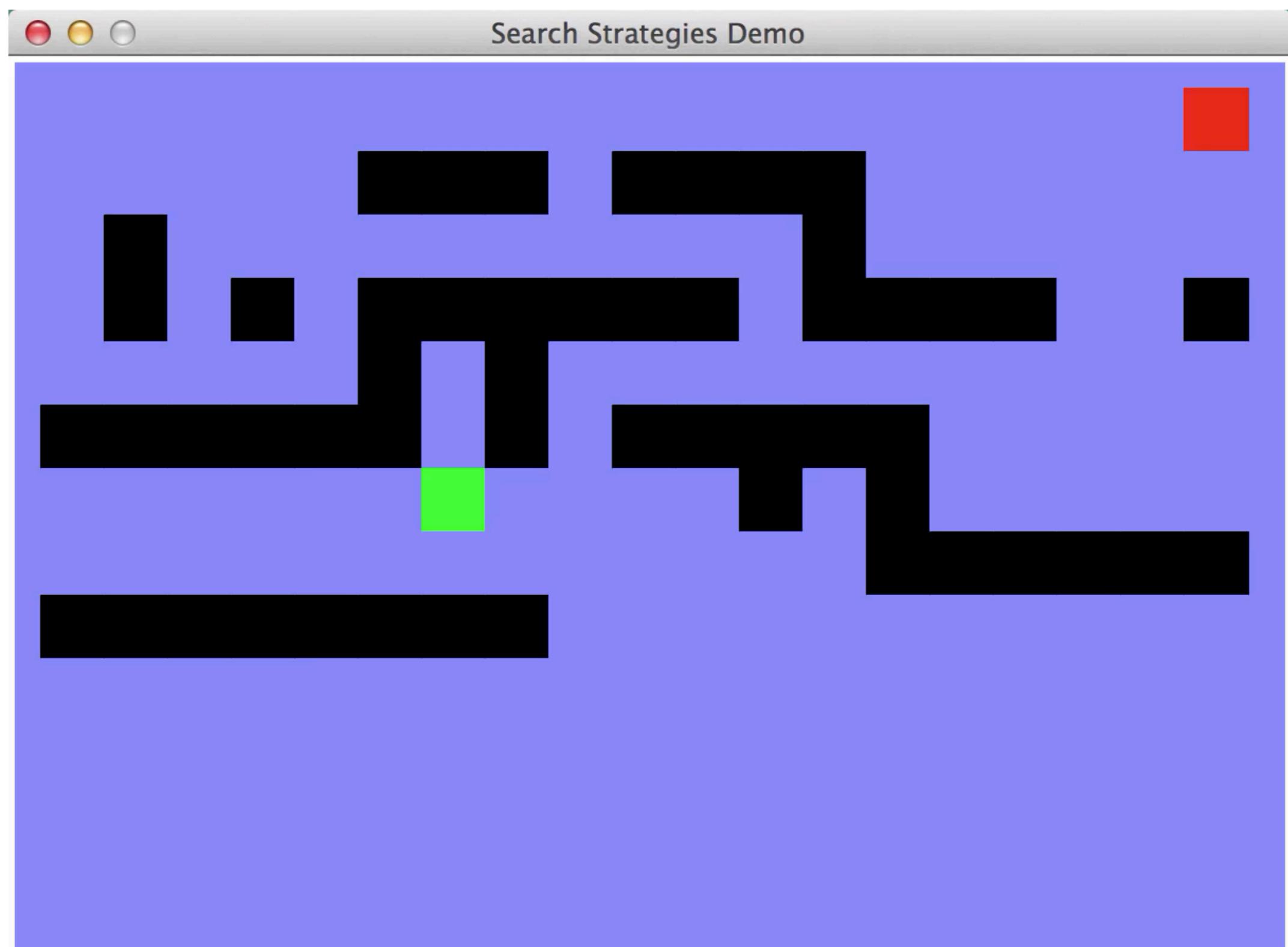
问题：深度优先搜索 vs 广度优先搜索

- 什么时候？广度优先搜索 **优于** 深度优先搜索
- 什么时候？深度优先搜索 **优于** 广度优先搜索

广度优先搜索示例

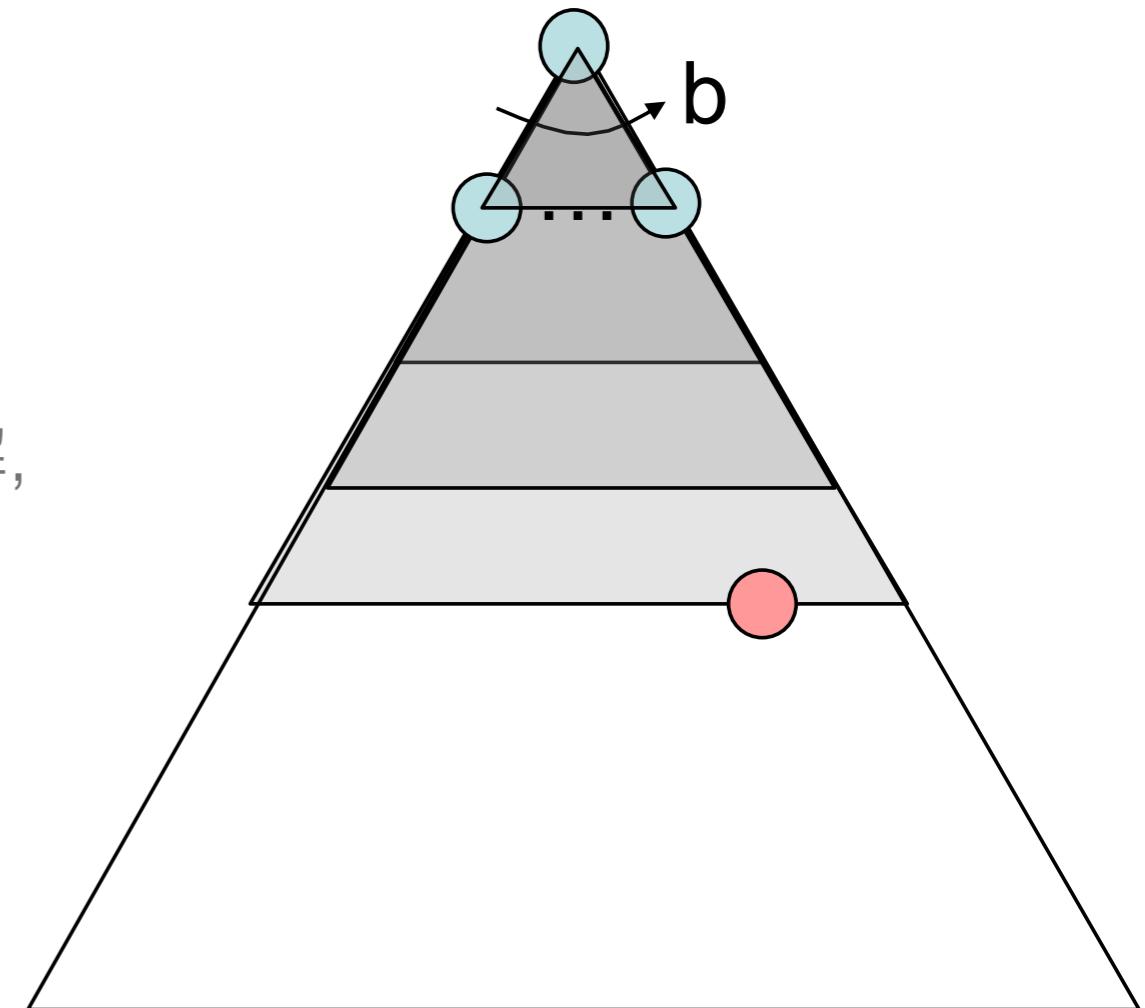


深度优先搜索示例



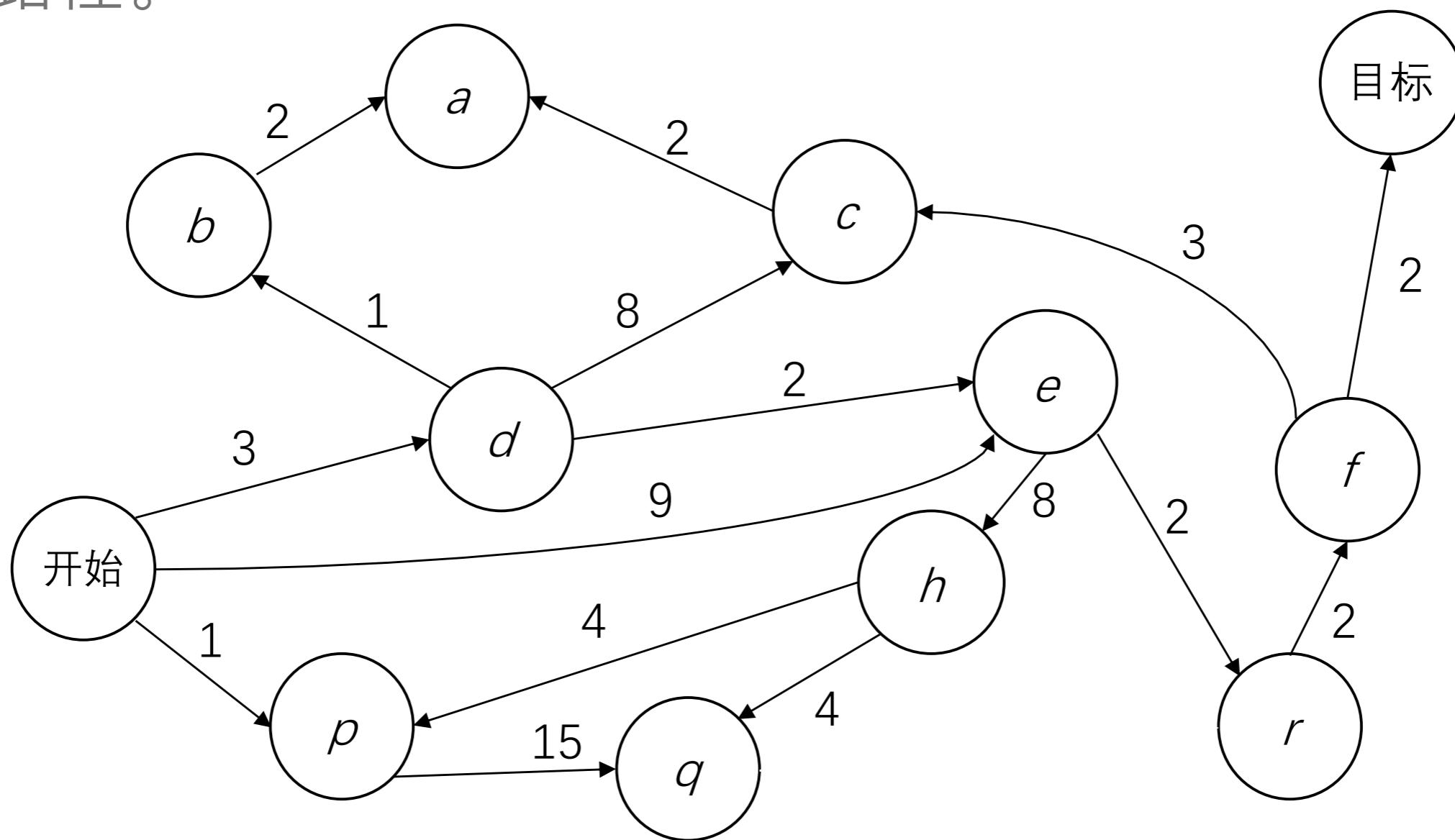
迭代加深

- 想法: 结合DFS的空间优势, 和BFS的时间/搜寻浅解的优势
 - 运行一次深度限制为1的DFS,如果没有解, ...
 - 运行一次深度限制为2的DFS,如果没有解,继续
 - 运行一次深度限制为3的DFS,如果没有解, ...
- 难道重复搜索不浪费吗?
 - 多数重复搜索集中在浅层, 节点数相对少, 所以还可以。

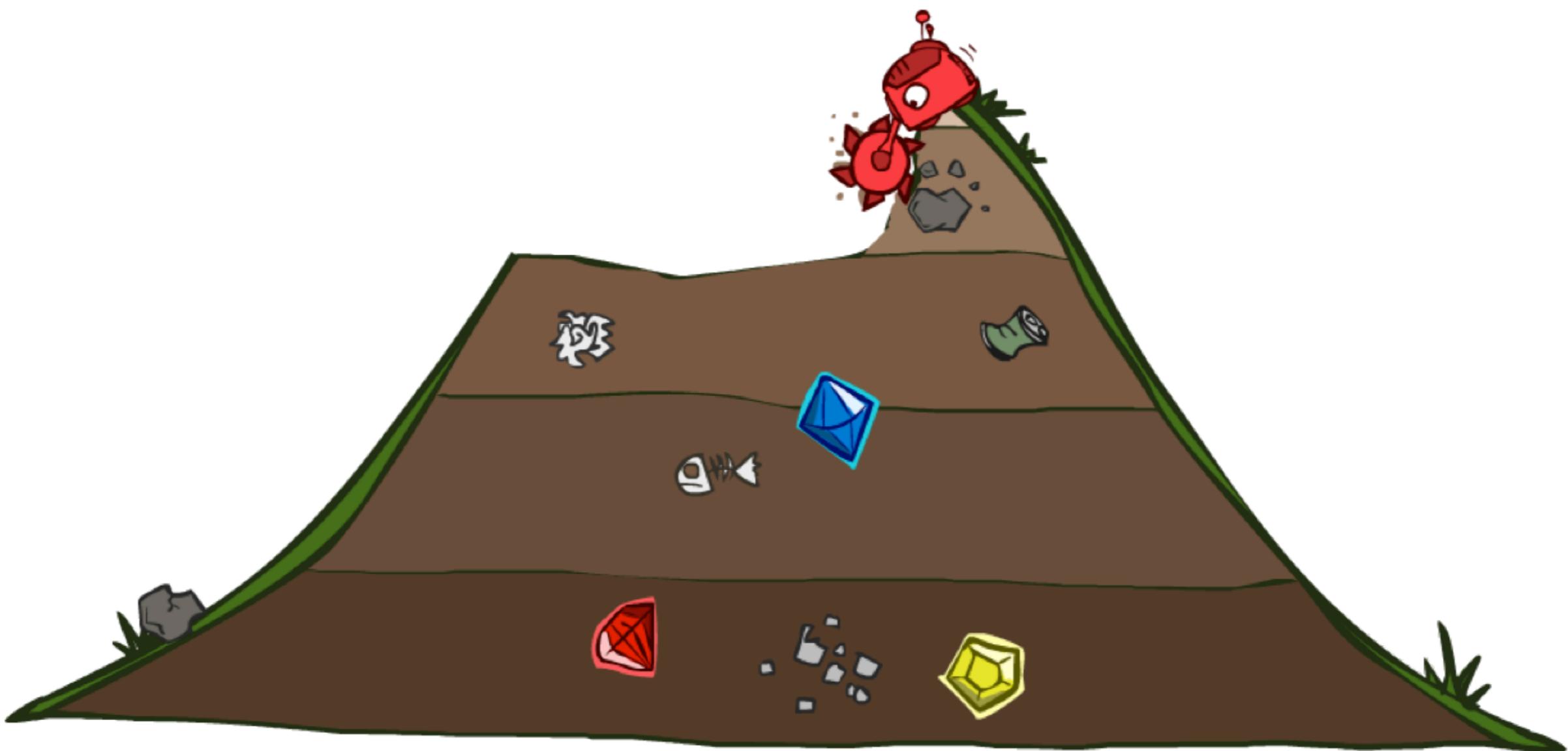


寻找最小步骤成本路径

- BFS找到的是最少行动数量的路径，而不是最小步骤成本路径。

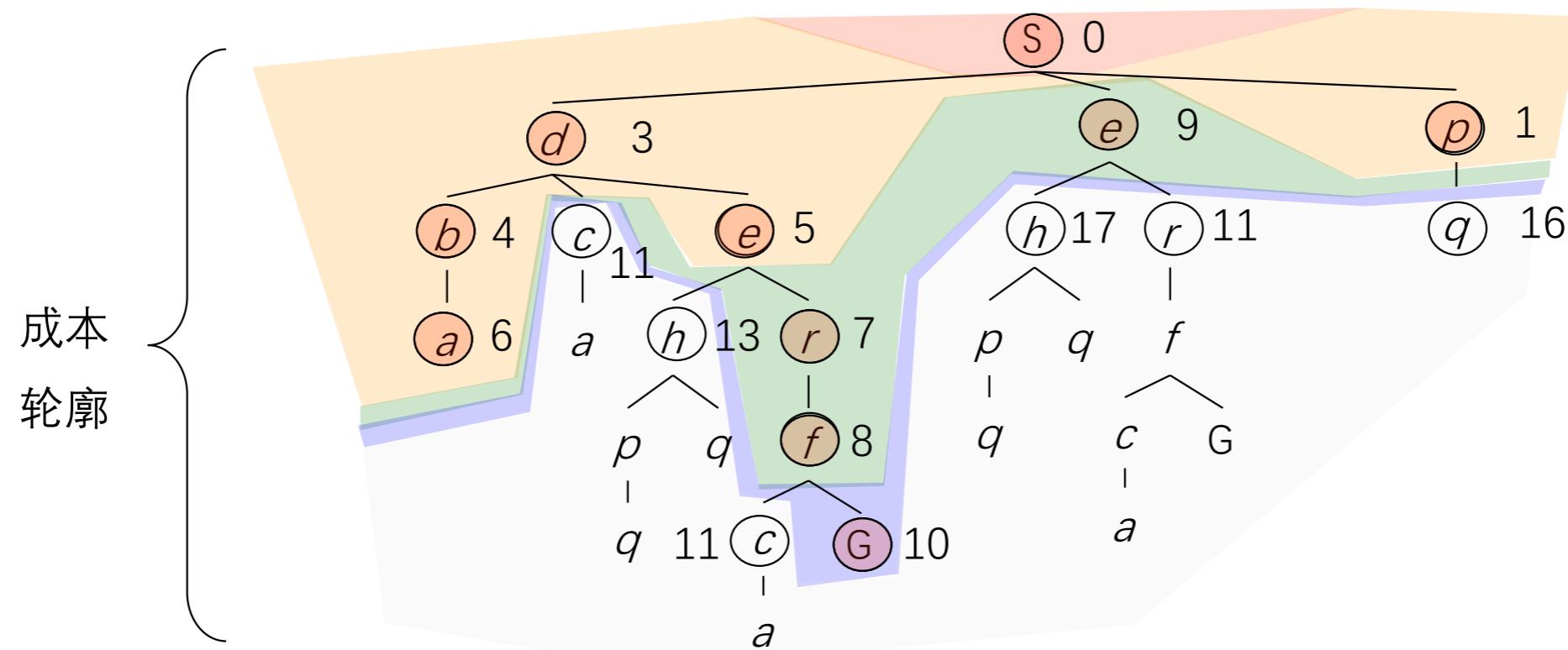
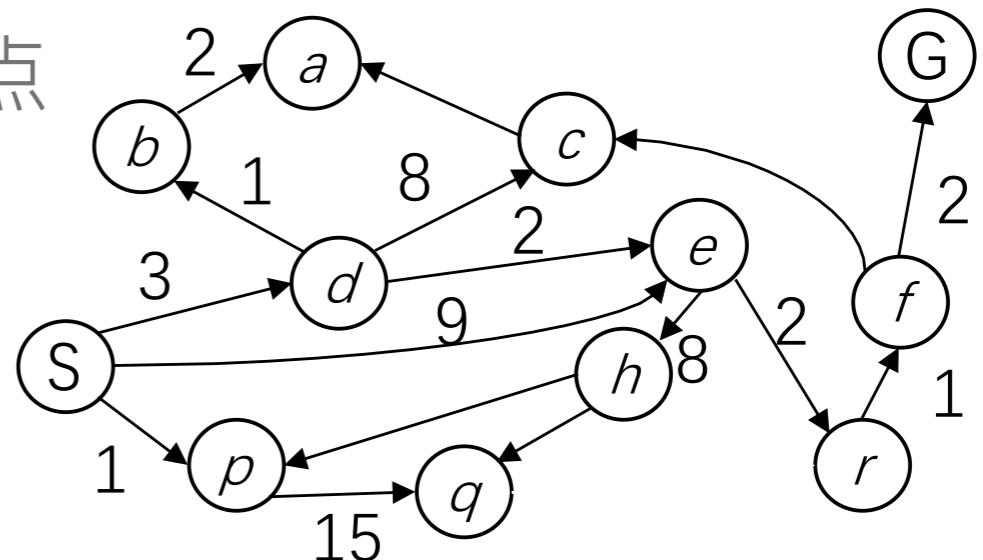


基于成本的统一搜索 (Uniform Cost Search)



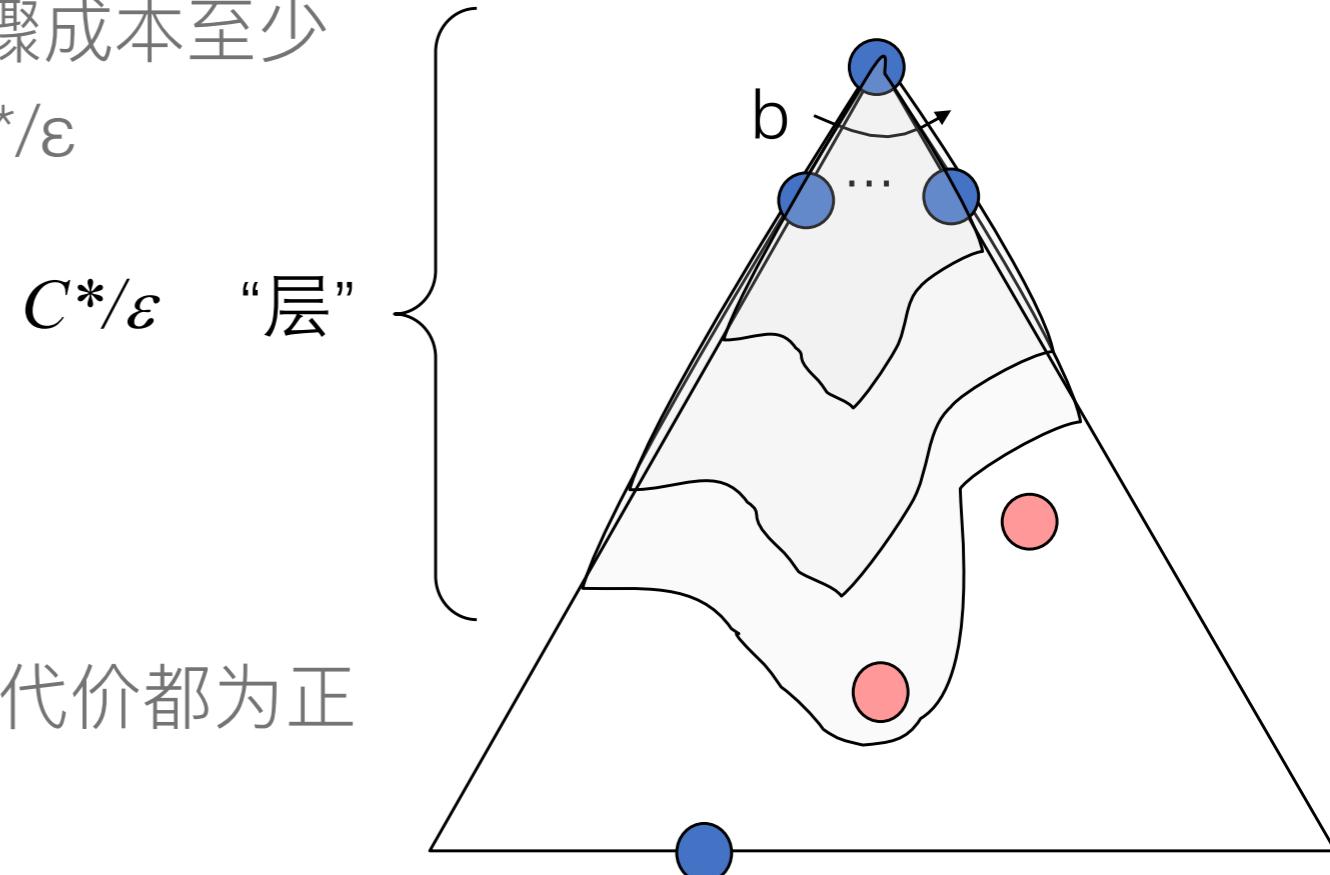
基于成本的统一搜索

- 策略：首先扩展一个成本最低的节点
- 实现：前沿用优先级队列存储
- 优先级：累计成本



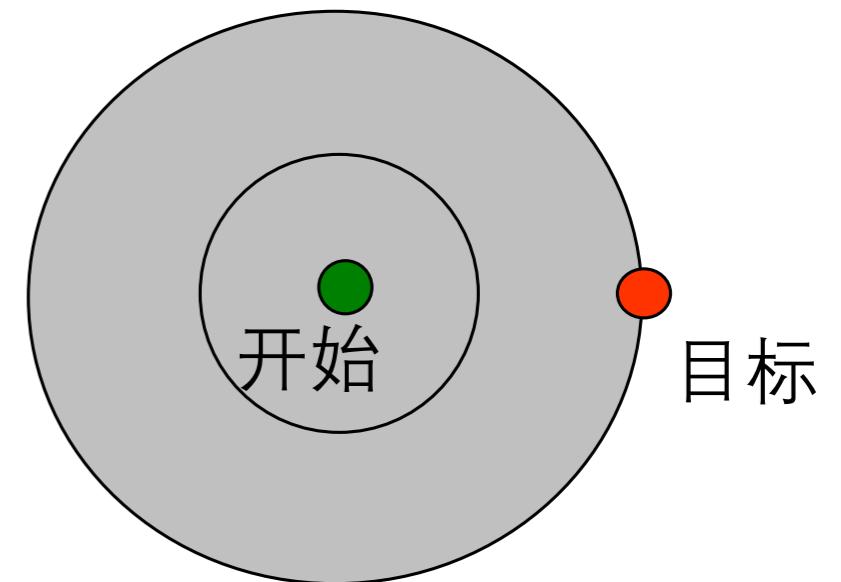
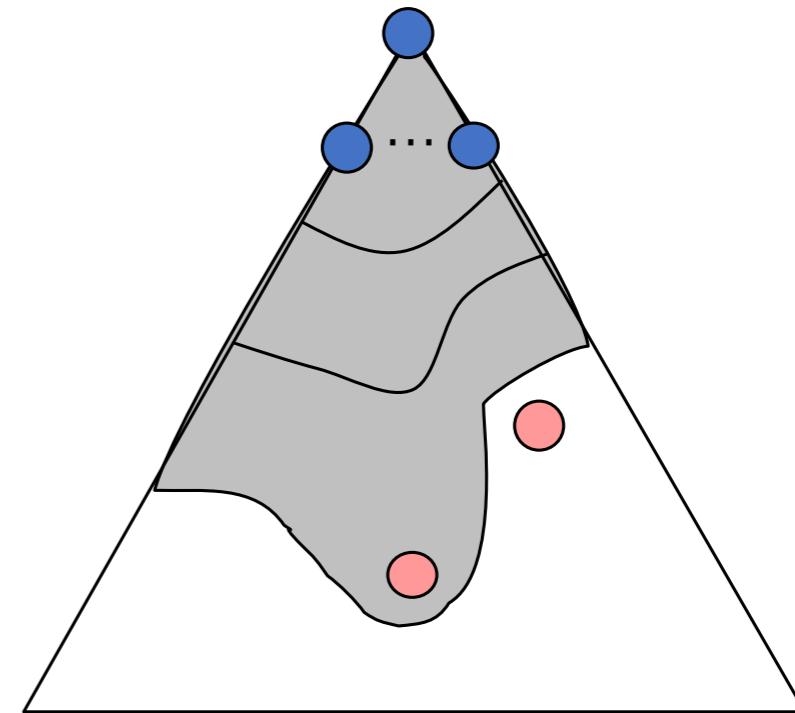
基于成本的统一搜索属性

- 哪些节点被扩展?
 - 处理所有累计成本小于最低成本解的所有节点
 - 如果那个解的成本是 C^* ，并且步骤成本至少是 ε ，那么其“有效深度”大概是 C^*/ε
 - 时间花费 $O(b^{C^*/\varepsilon})$ (有效深度指数)
- 前沿节点占用多少空间?
 - 大概是最后一层, 所以是 $O(b^{C^*/\varepsilon})$
- 完备性?
 - 是, 如果最优解的成本有限, 步骤代价都为正数。
- 最优性?
 - 是。

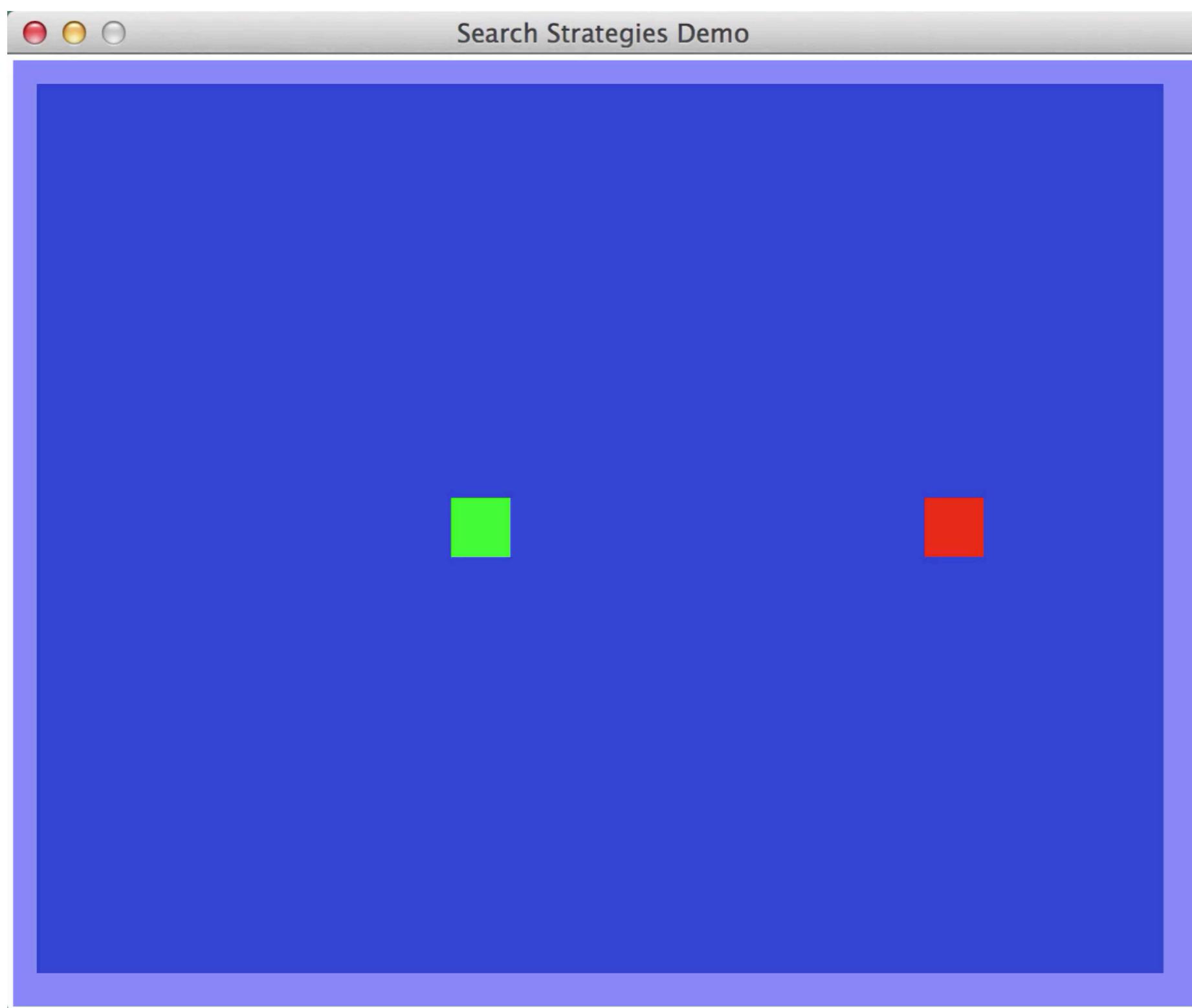


基于成本的统一搜索的不足

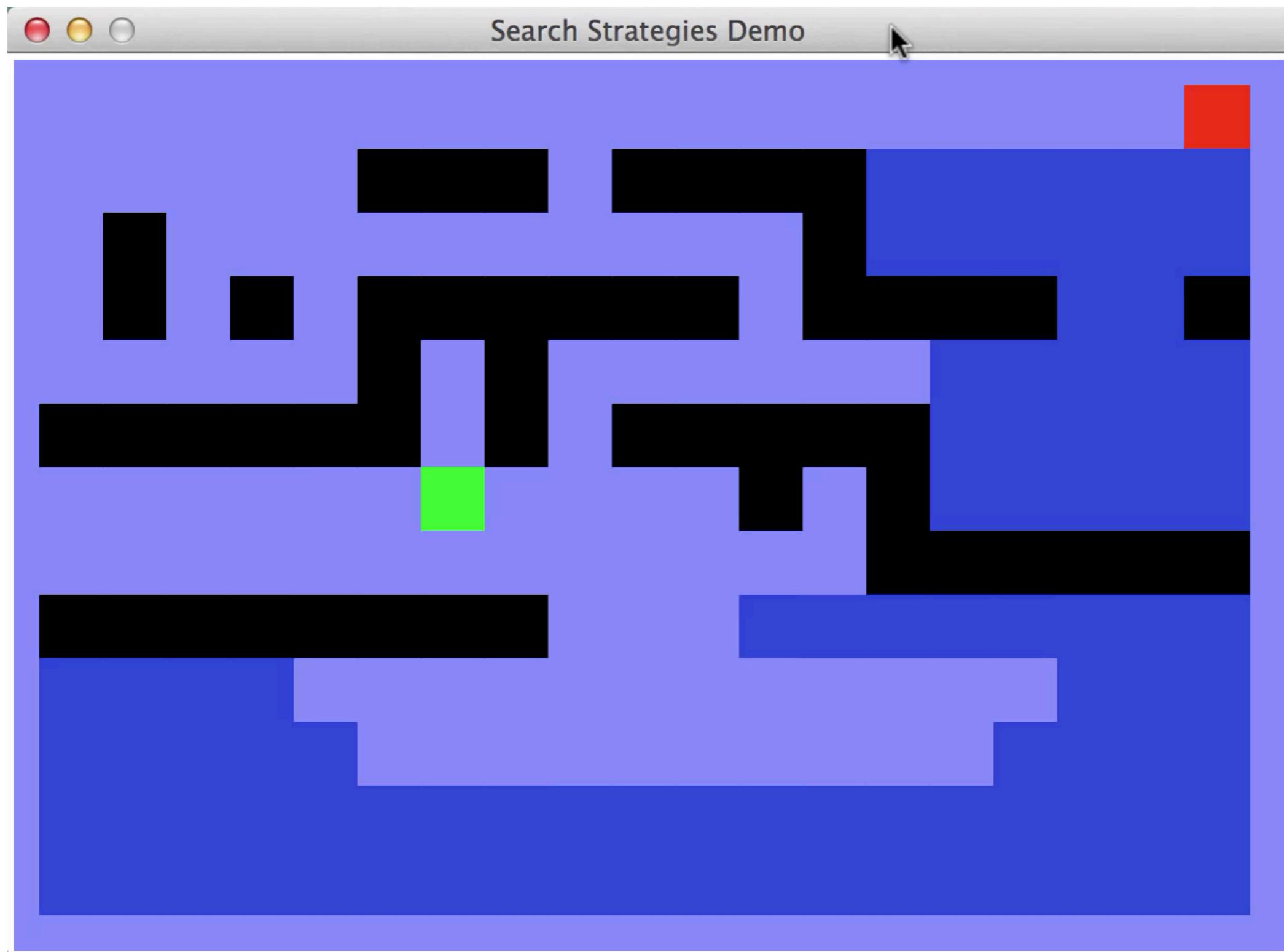
- 探索逐步提高的成本轮廓范围内的节点
- 优势: 完备性和最优化
- 不足:
 - 在每个“方向”上都探索
 - 没有关于目标位置的信息



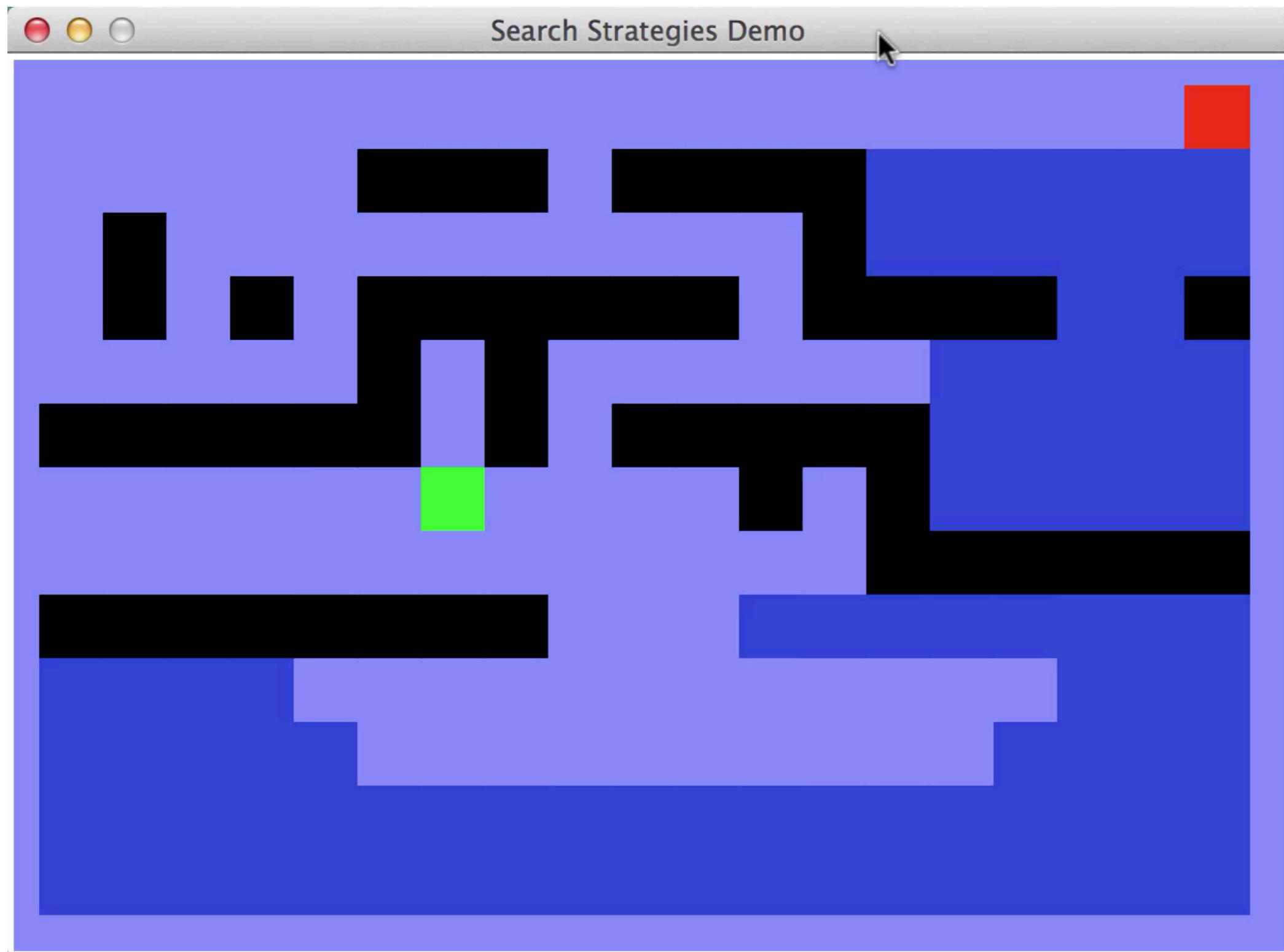
空迷宫中的基于成本的统一搜索



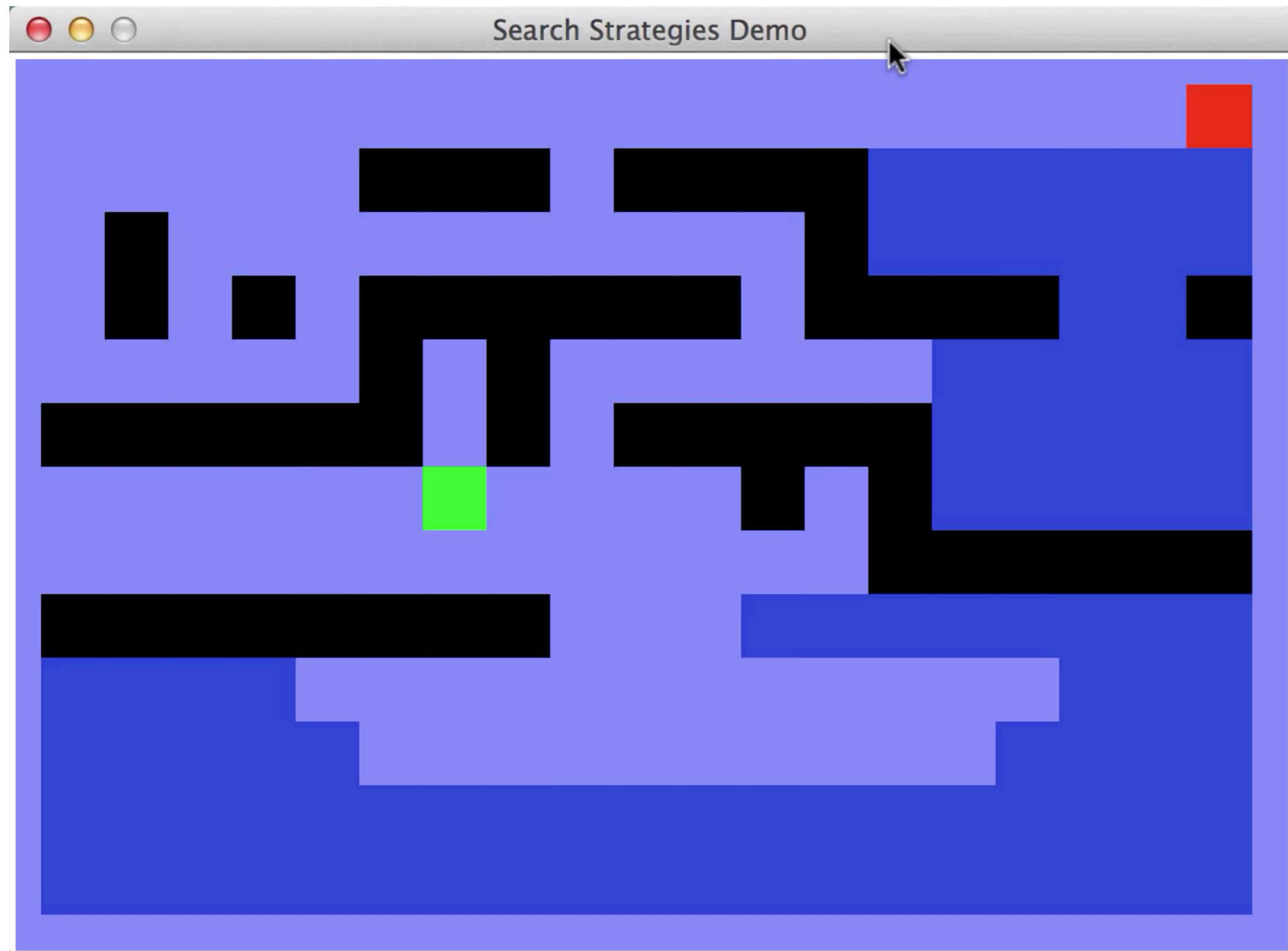
DFS, BFS, or UCS?



DFS, BFS, or UCS?

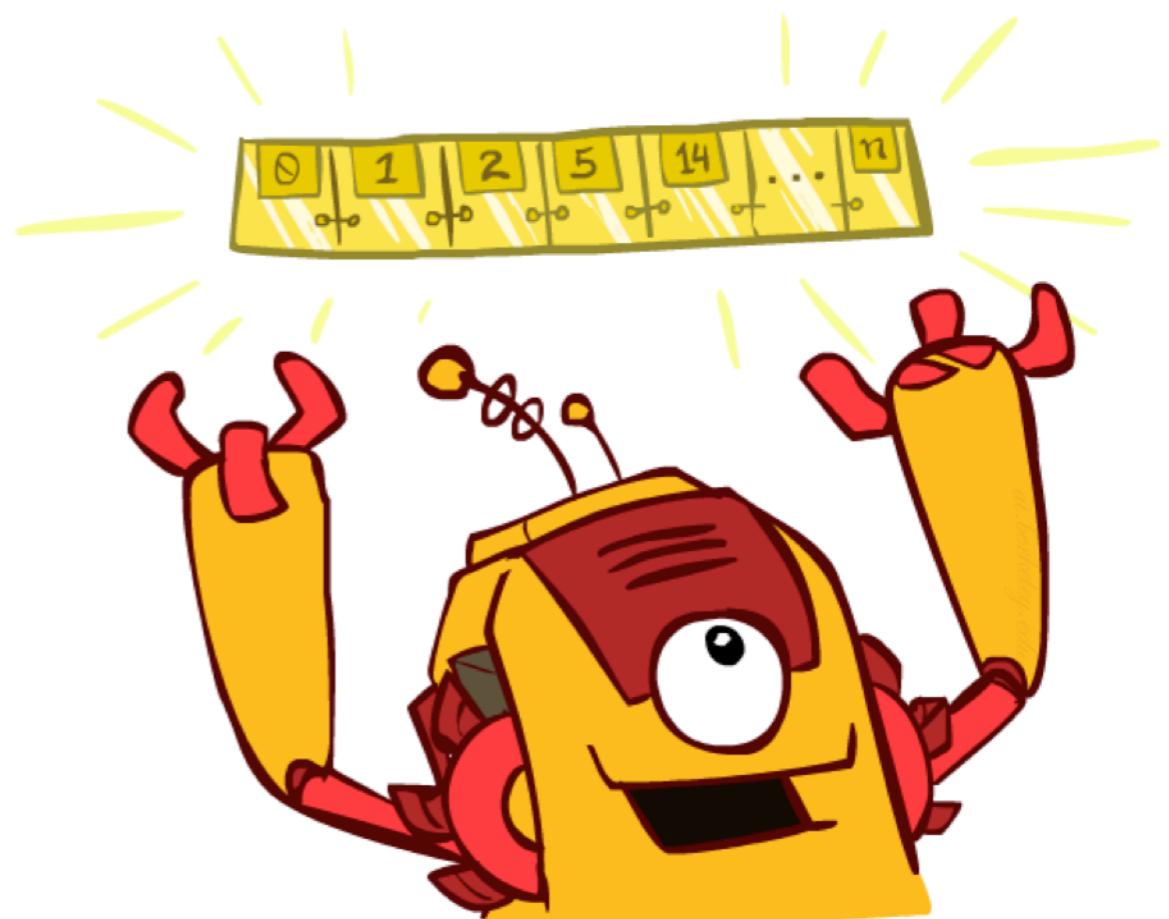


DFS, BFS, or UCS?



都是一个队列

- 除了边缘策略，所有这些搜索算法是相同的
 - 从概念上讲，所有的边缘都是优先队列(即集合与附加的节点优先级)
 - 实际上,DFS和BFS可以用栈和队列，避免 $\log(n)$ 的排序开销
 - 甚至可以同一套代码来实现，只需要一个参数来确定出队列的方式



搜索算法和模型

- 搜索算法操作的是对世界的抽象模型
 - Agent实际上不会尝试现实世界中的所有可能路径!
 - 计划都是“模拟”的
 - 你的建模方式决定了搜索算法的上限...

