

通知

- 作业1发布
 - 关于智能体和搜索部分内容
 - 9月29日（1周以后）课堂上交
 - 在另一张纸上手写完成即可（打印也可以）
 - 写清姓名和学号

树搜索方法框架

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

 选择一个叶 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 (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

- 关键元素：
 - 搜索前沿
 - 扩展: 产生子节点
 - 探索(选择)策略
- 主要问题： 从哪些前沿中的节点开始探索？

实现说明

节点的域：

状态 (state) , 父节点(parent), 行动(action), 路径成本(path-cost)

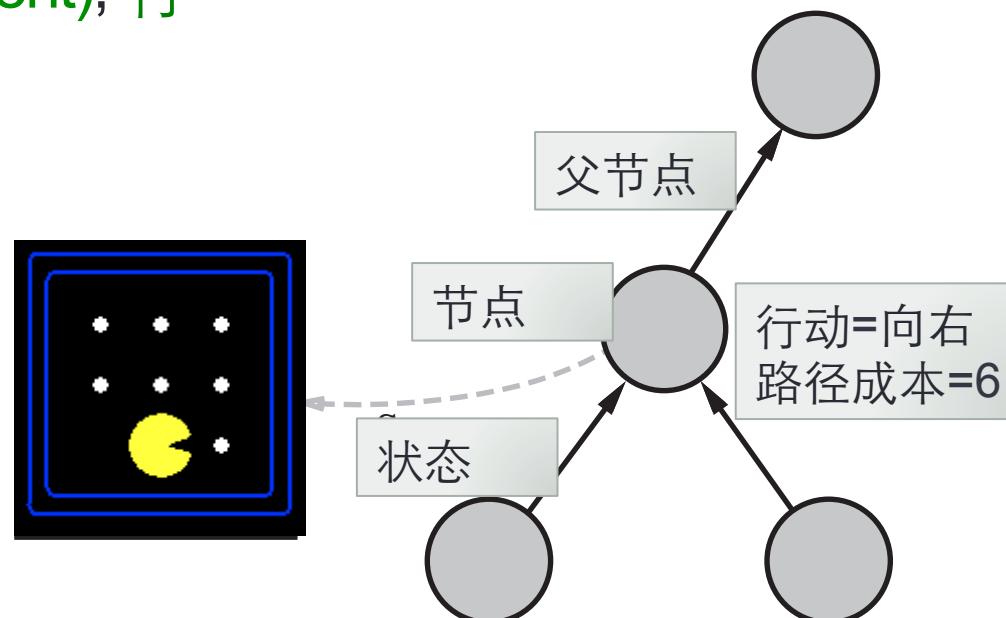
父节点 node 通过行动 a 达到一个子节点，这个子节点的域：

state = result(node.state, a)

parent = node

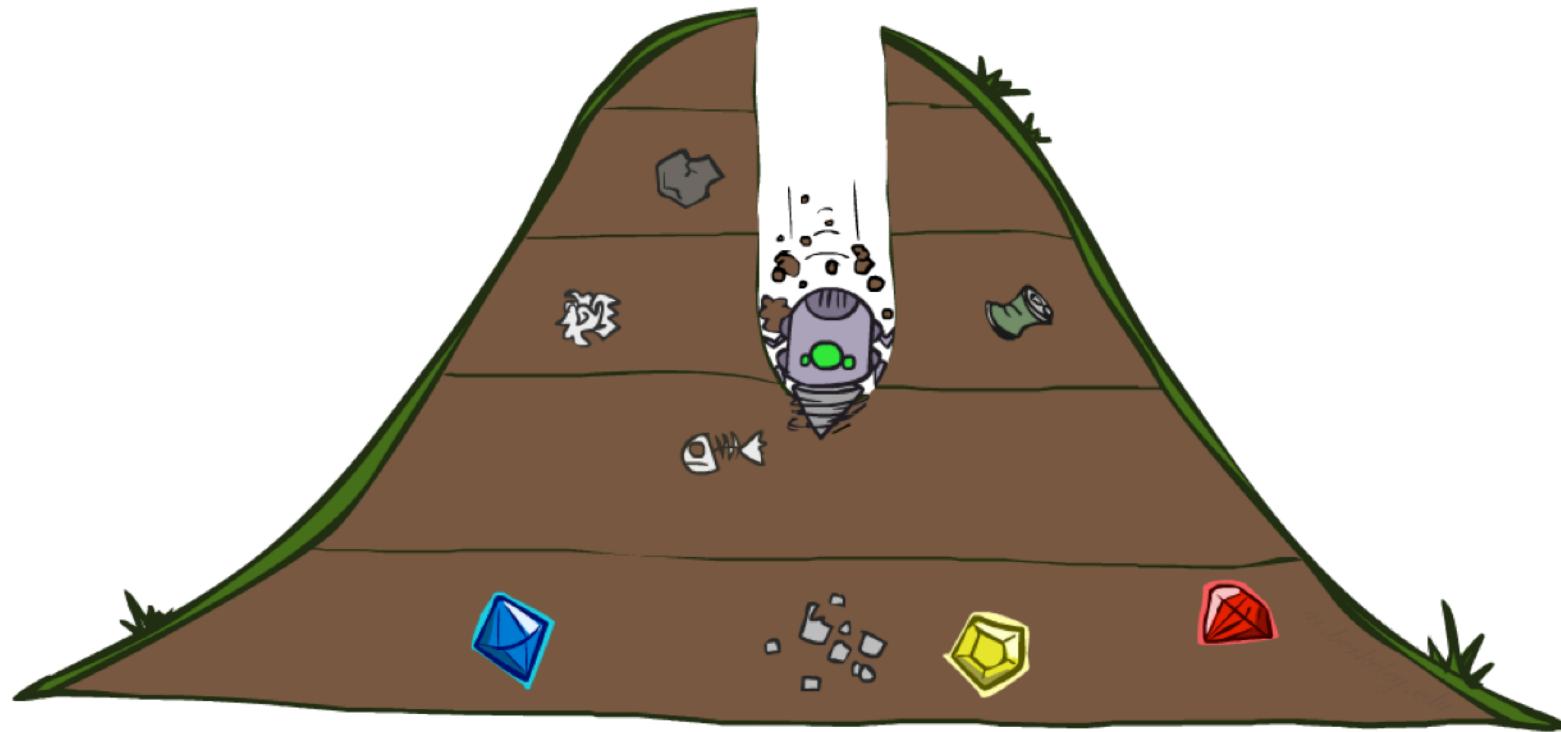
action = a

path-cost = node.path-cost +
step-cost(node.state, a , state)



获取解是通过追踪路径上的节点，收集相应的行动

深度优先搜索

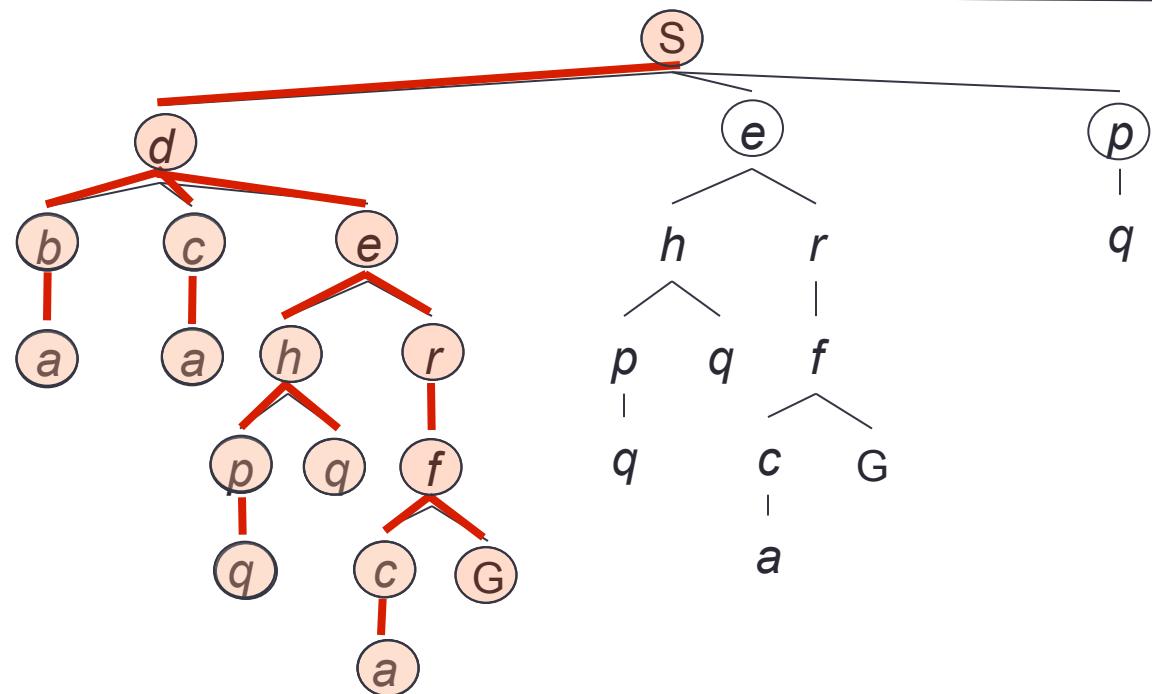
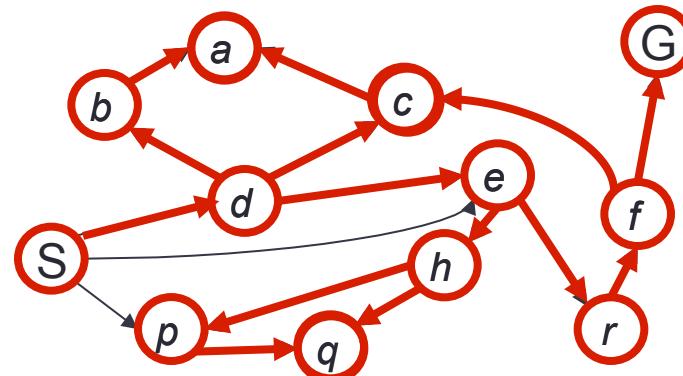


深度优先搜索

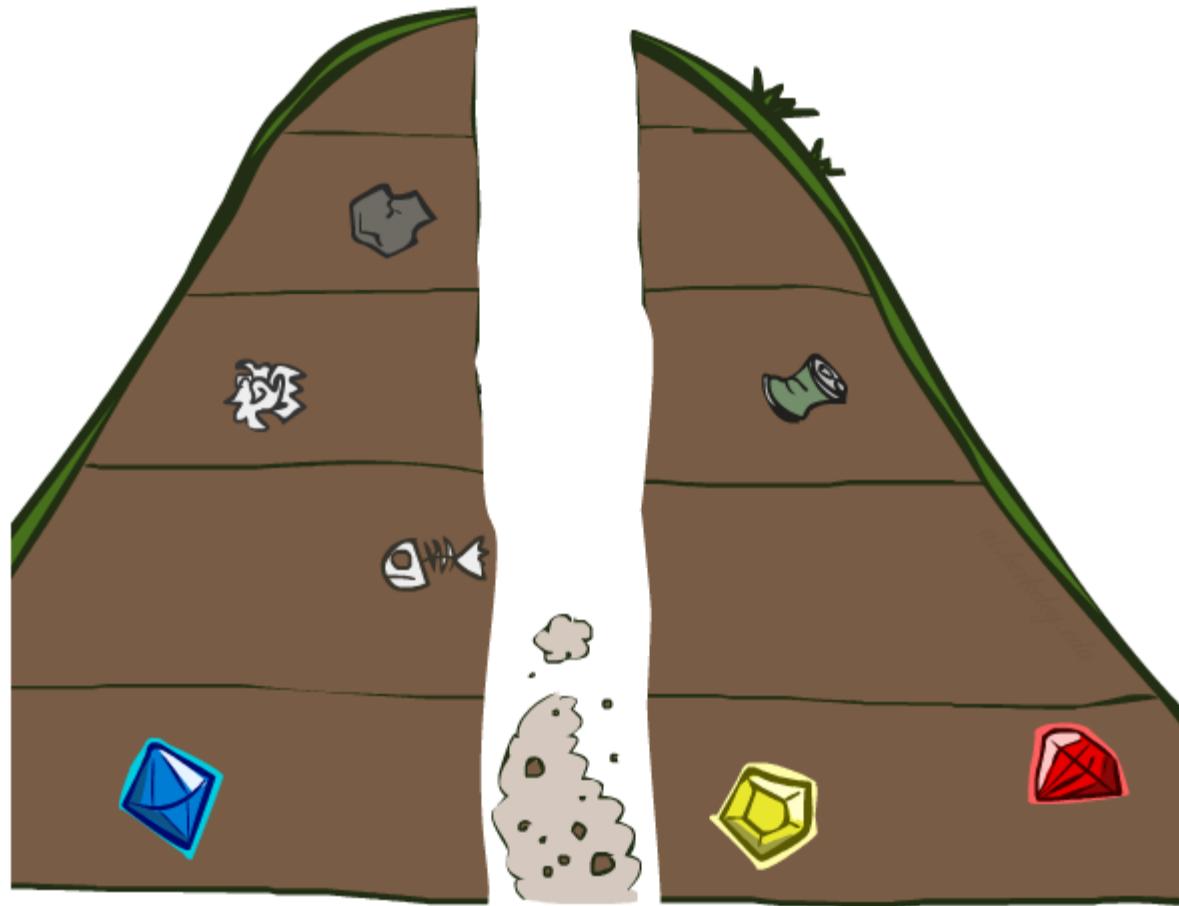
策略：总是最先扩展一个最深的节点

实现：

前沿是一个后进先出的栈

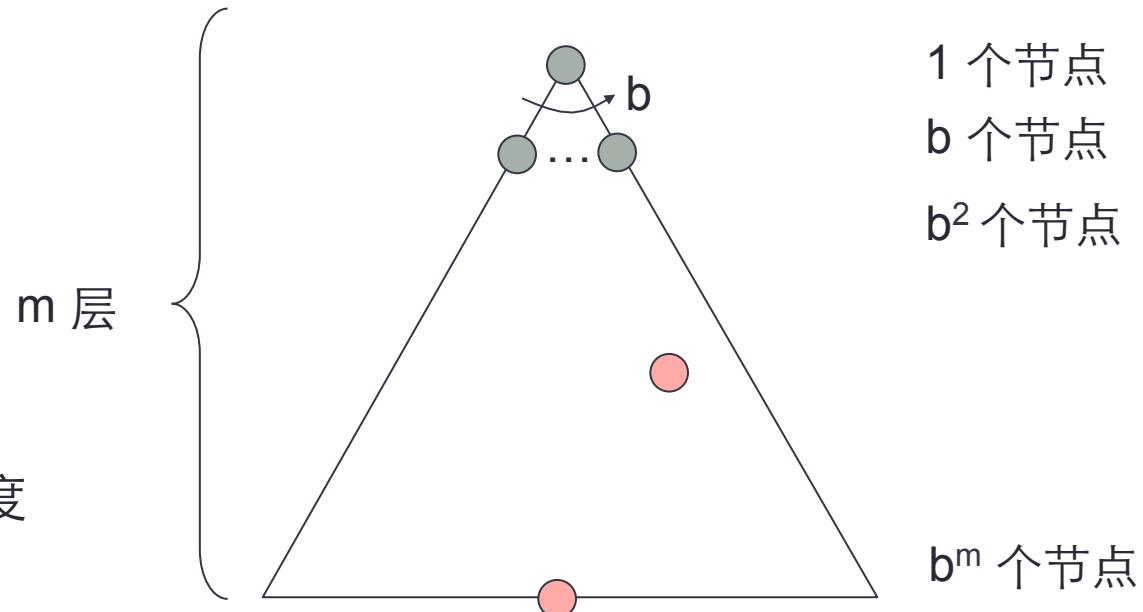


搜索算法的属性



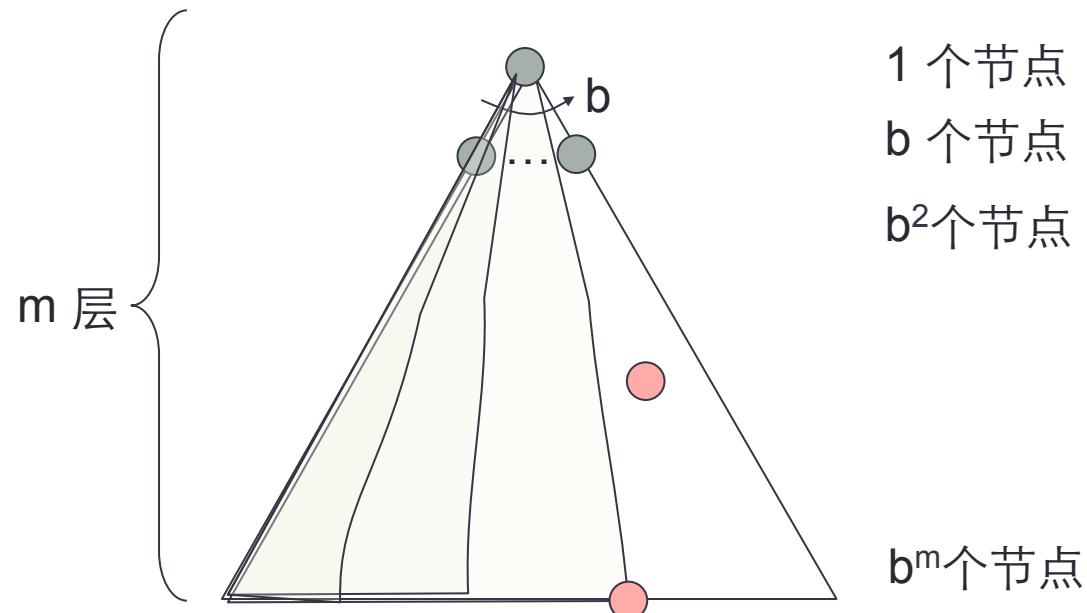
搜索算法的属性

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

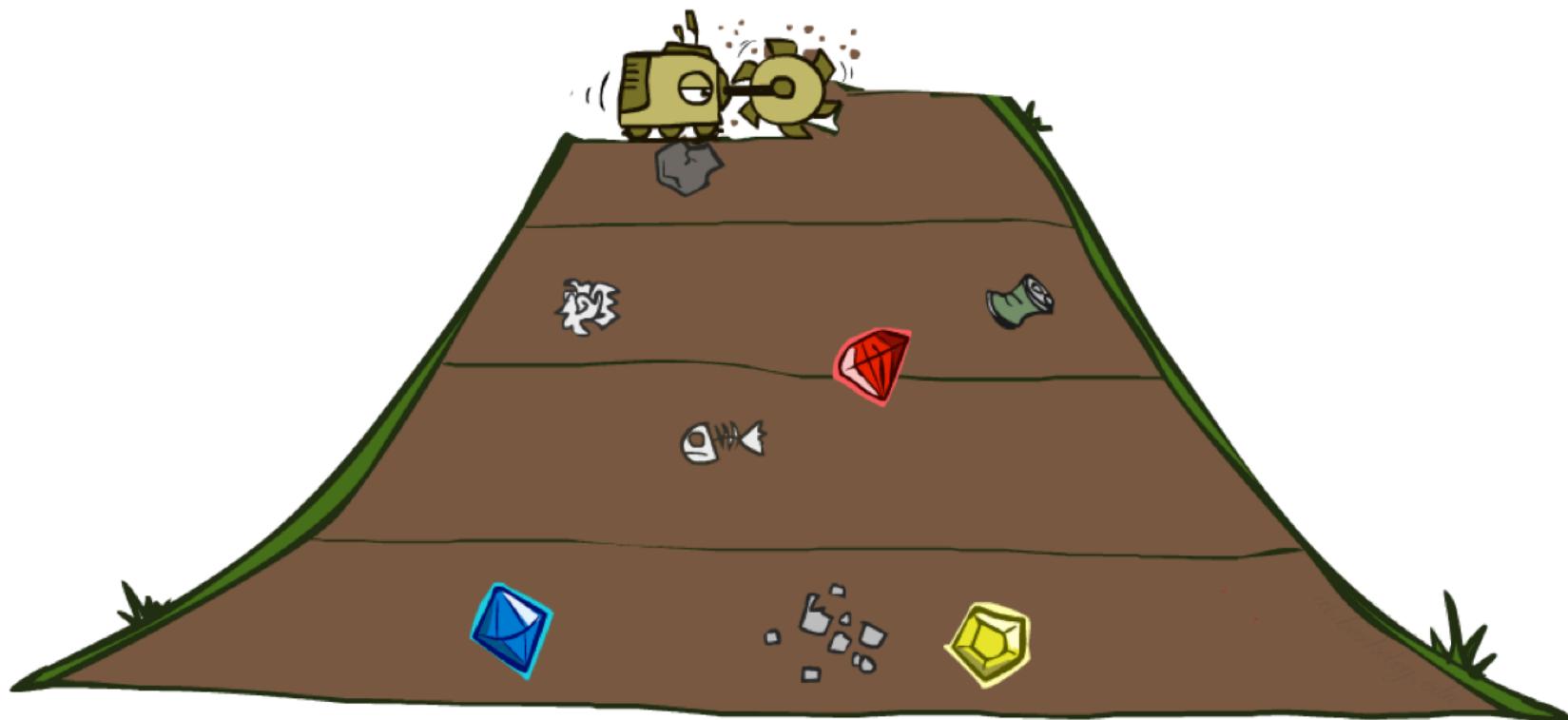


深度优先搜索 (DFS) 属性

- 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 $O(bm)$
- 完全性?
 - m 可能是无穷。除非可以避免循环搜索
- 优化性?
 - 不优化。发现的只是树最左边的一个解。



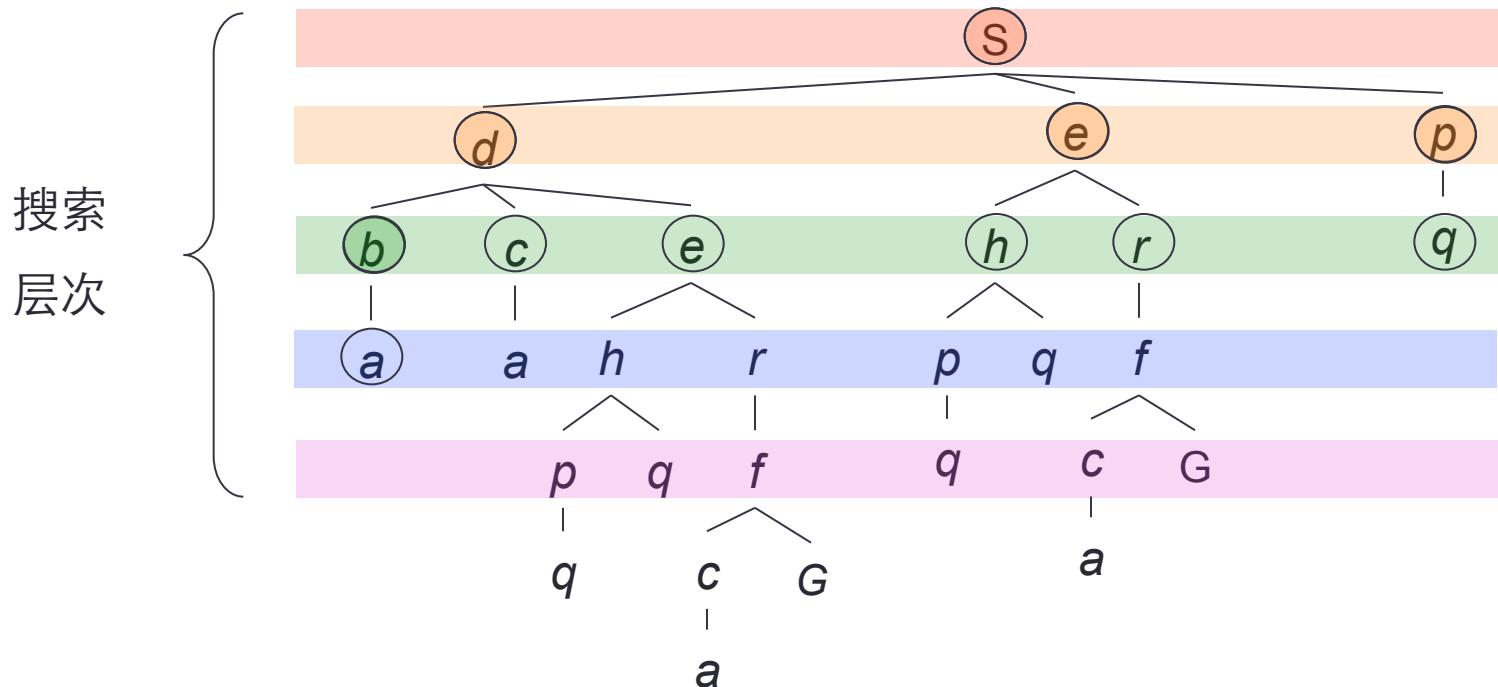
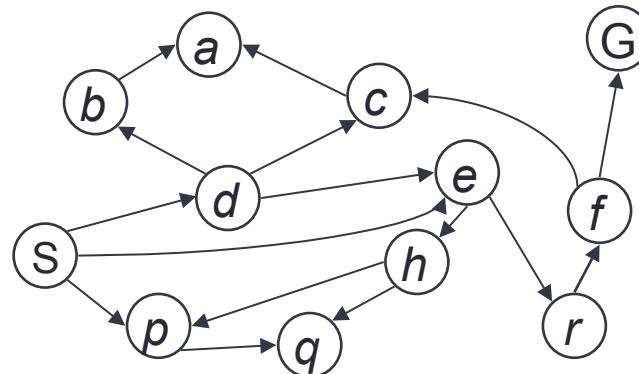
广度优先搜索 (BFS)



广度优先 (BFS) 搜索

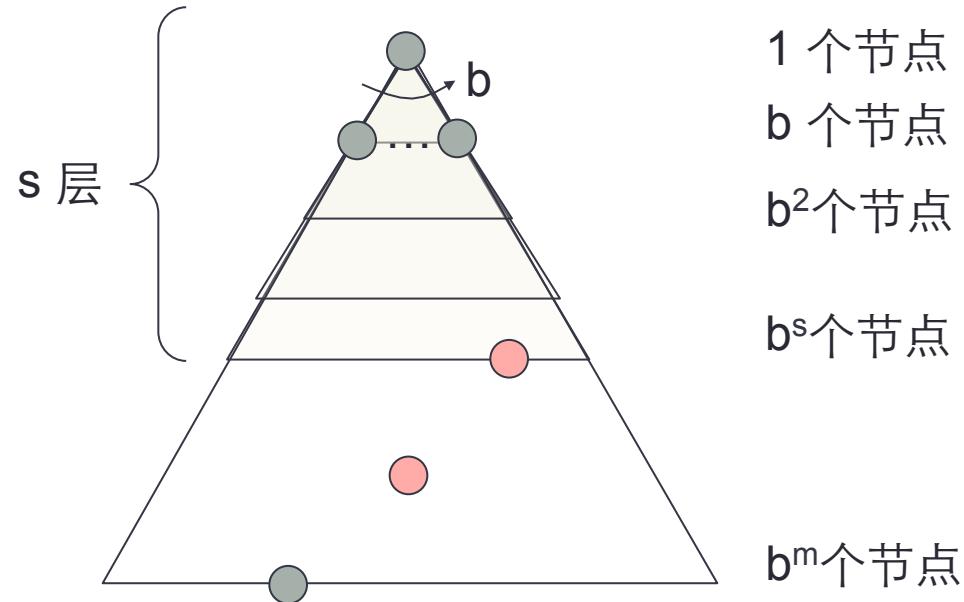
策略：优先扩展最浅的节点

实现：**先进先出的队列**

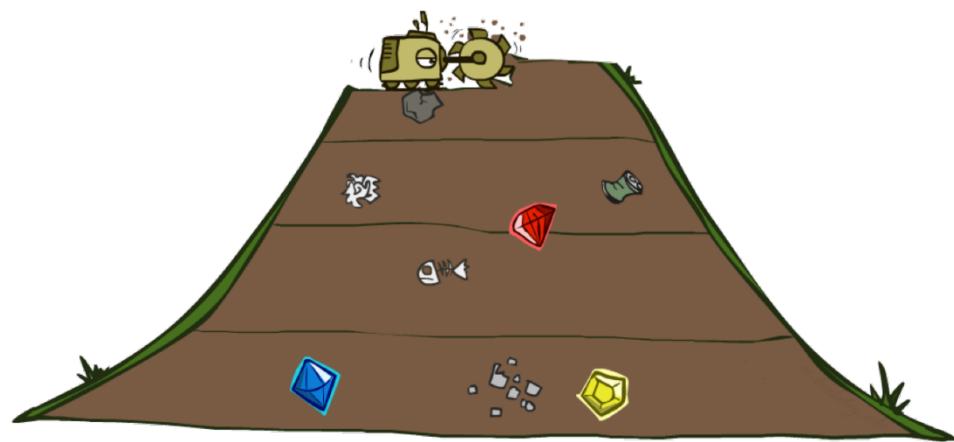
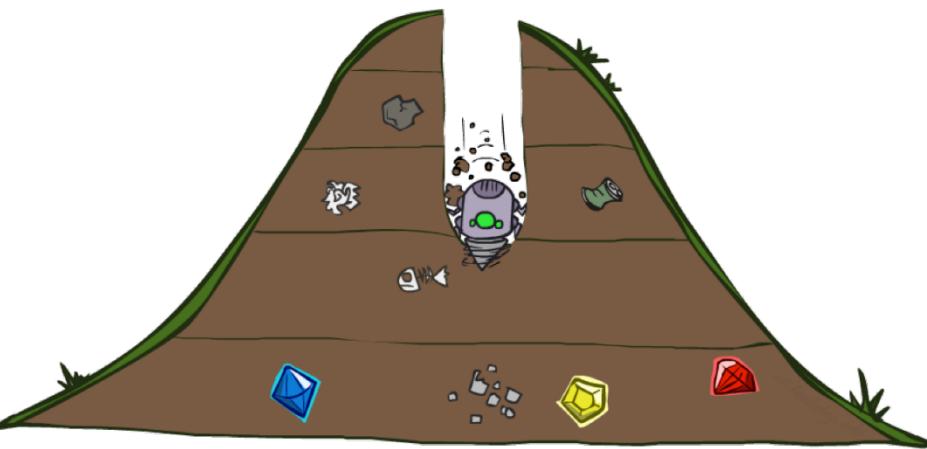


广度优先搜索属性

- 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
 - 大概是探索的上一层, 所以 $O(b^s)$
- 完全性?
 - 如果一个解存在, s 深度一定是有有限的, 所以是完全的!
- 优化性?
 - 是, 如果所有步骤成本都是1时 (常数)



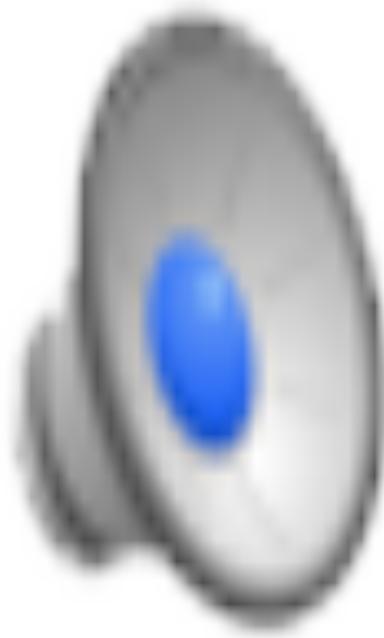
问题：DFS vs BFS



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

演示视屏：迷宫 DFS/BFS (1)

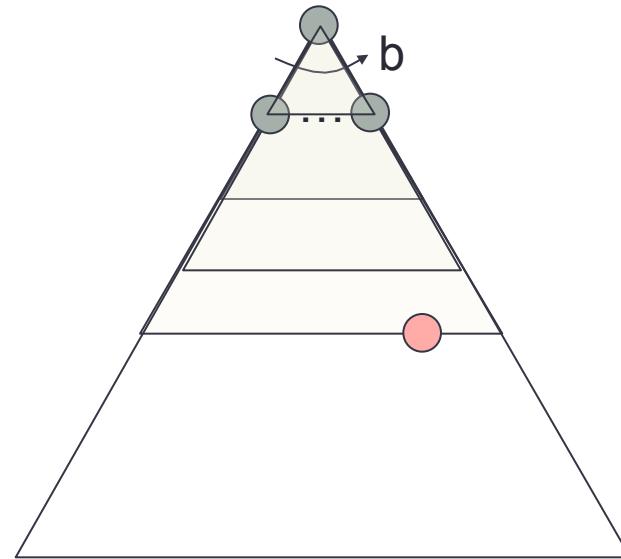


演示视屏：迷宫 DFS/BFS (2)

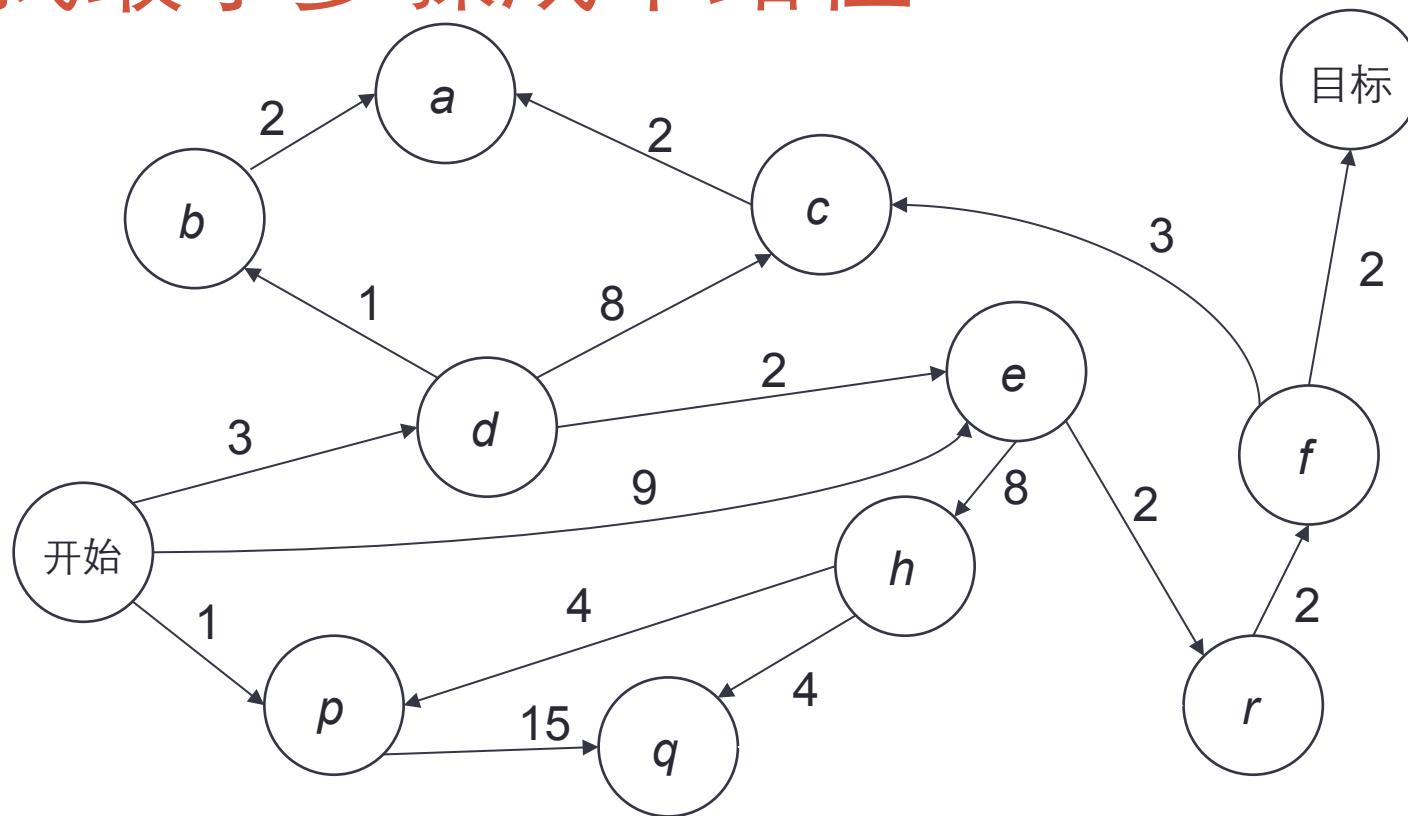


迭代加深

- 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。

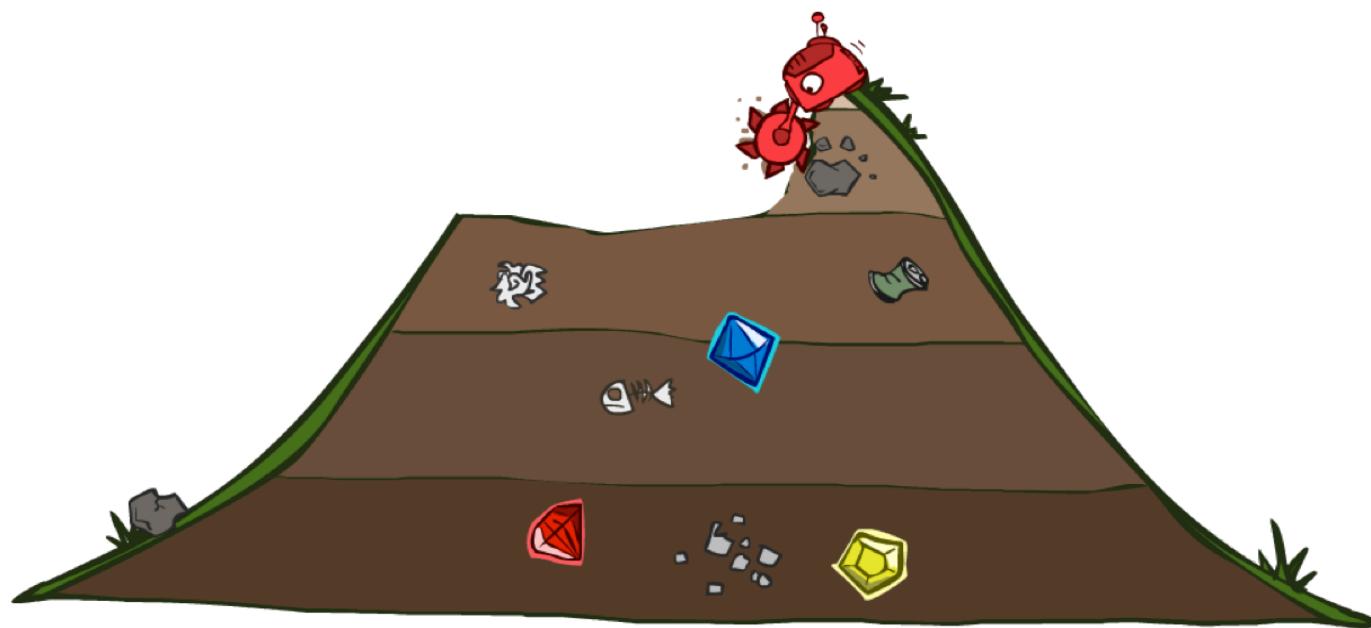


寻找最小步骤成本路径



- BFS找到的是最少行动数量的路径，而不是最小成本路径。
- 我们将用一个相似的算法找到最小成本路径。

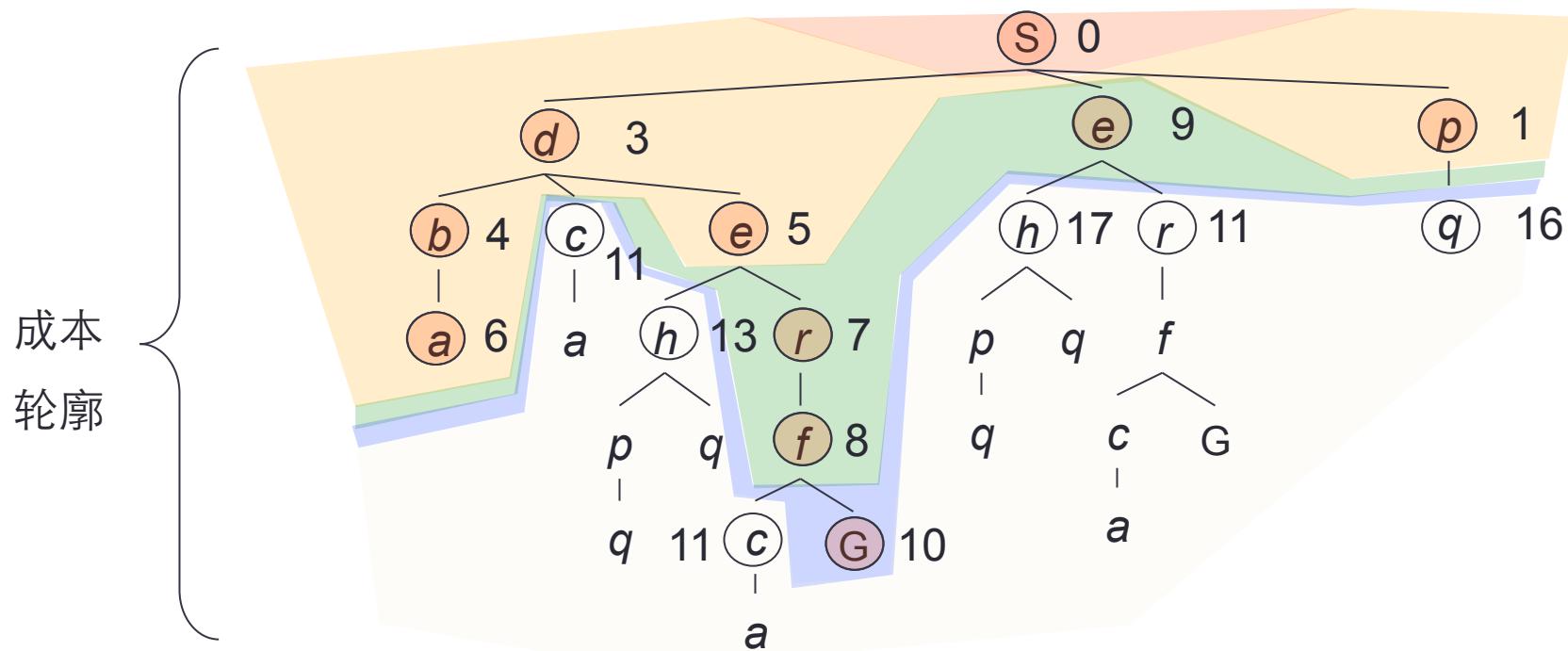
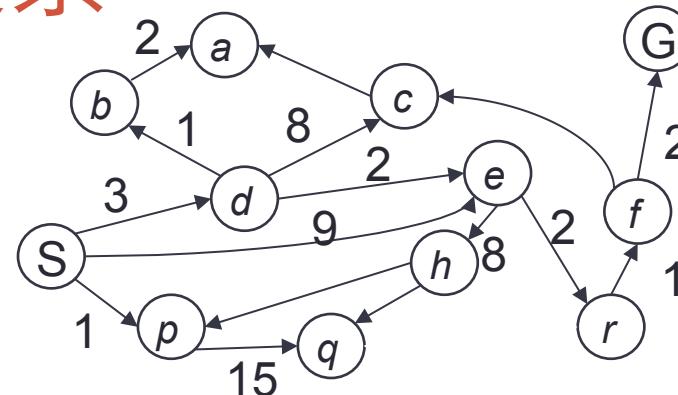
基于成本的统一搜索 (Uniform Cost Search)



基于成本的统一搜索

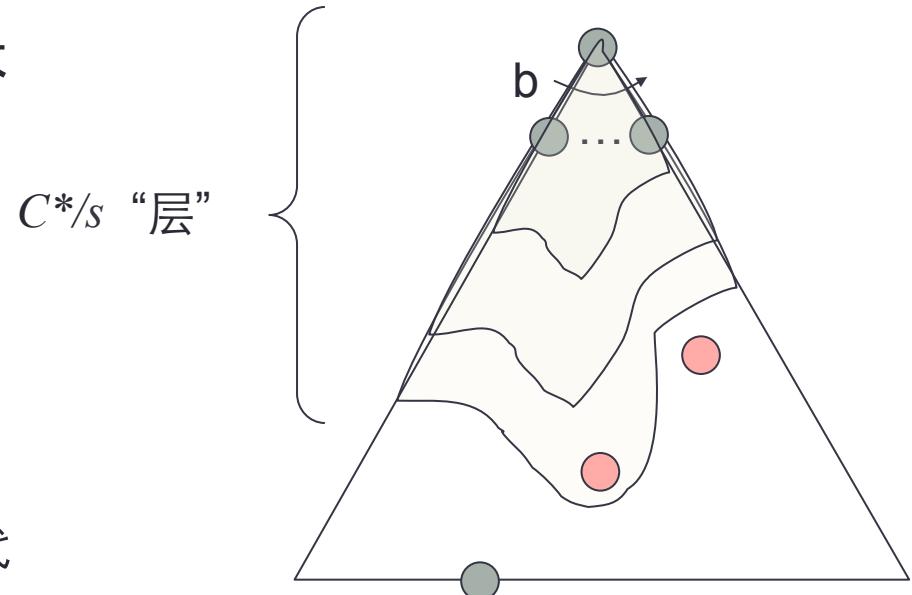
策略：首先扩展一个成本最低的节点

前沿用优先级队列存储
(优先级：累计成本)



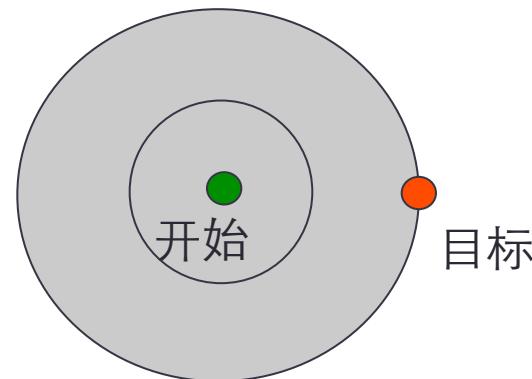
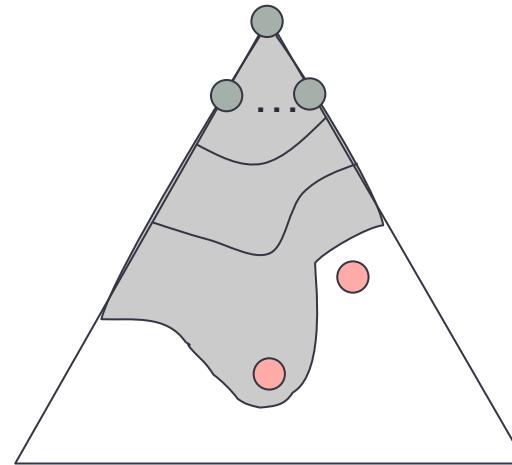
基于成本的统一搜索 (UCS) 属性

- 哪些节点被扩展?
 - 处理所有累计成本小于最低路径成本解的所有节点
 - 如果那个解的成本是 C^* ，并且步骤成本至少是 s ，那么其“有效深度”大概是 C^*/s
 - 时间花费 $O(b^{C^*/s})$ (有效深度为指数)
- 前沿节点占用多少空间?
 - 大概总是上一层，所以是 $O(b^{C^*/s})$
- 完全性?
 - 假设最优解的路径成本有限，步骤代价都为正数，则是！
- 优化性?
 - 是。

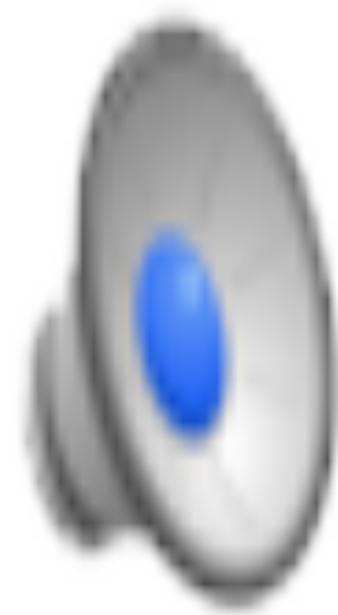


存在不足

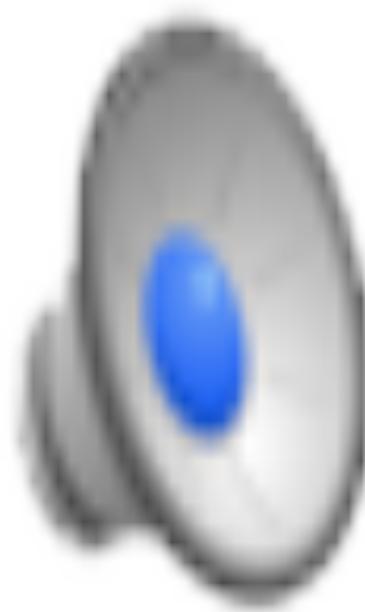
- 逐步搜索成本轮廓由小到大范围内的节点
- 优势: 完全性和优化性
- 不足:
 - 在每个“方向”上都探索
 - 没有关于目标位置的信息
- 效率很低



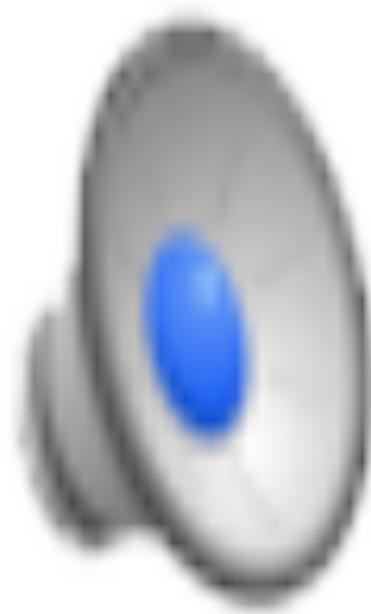
演示视频：空白UCS



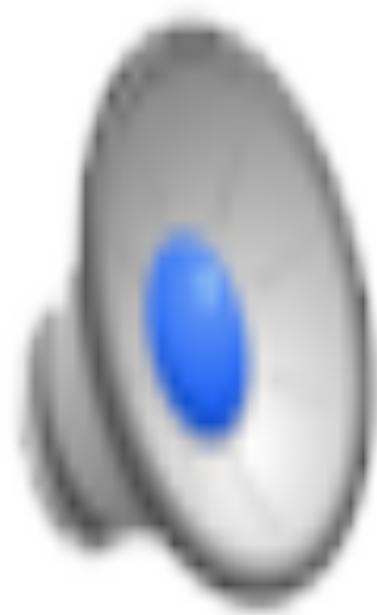
演示视频：有深/浅水的迷宫 --- DFS,
BFS, or UCS ? (1)



演示视频：有深/浅水的迷宫 --- DFS,
BFS, or UCS ? (2)



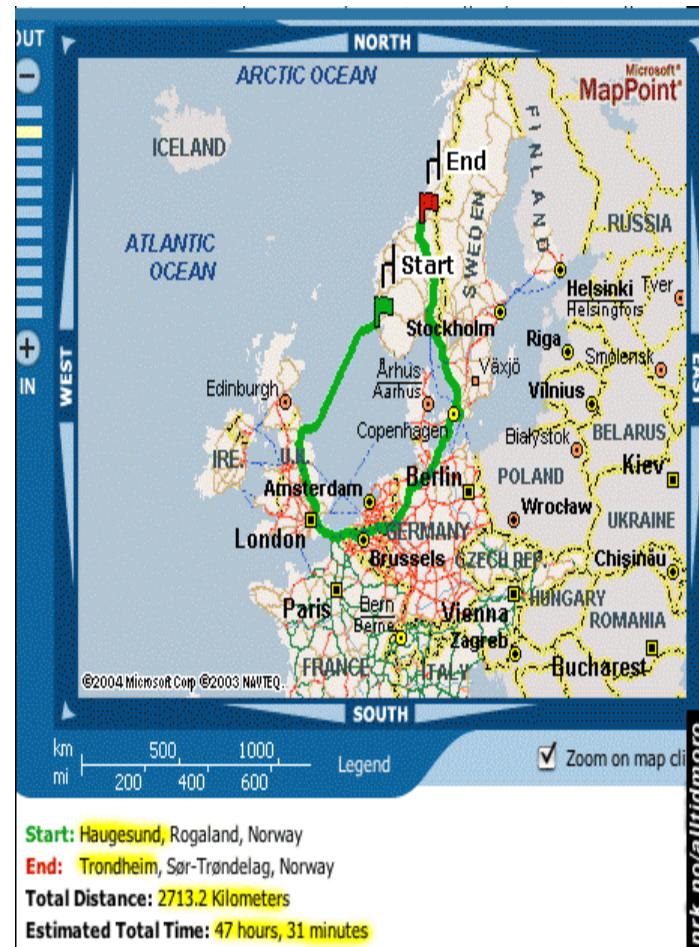
演示视频：有深/浅水的迷宫 --- DFS,
BFS, or UCS ? (3)



Search Gone Wrong?



© 2005 MapQuest.com, Inc.



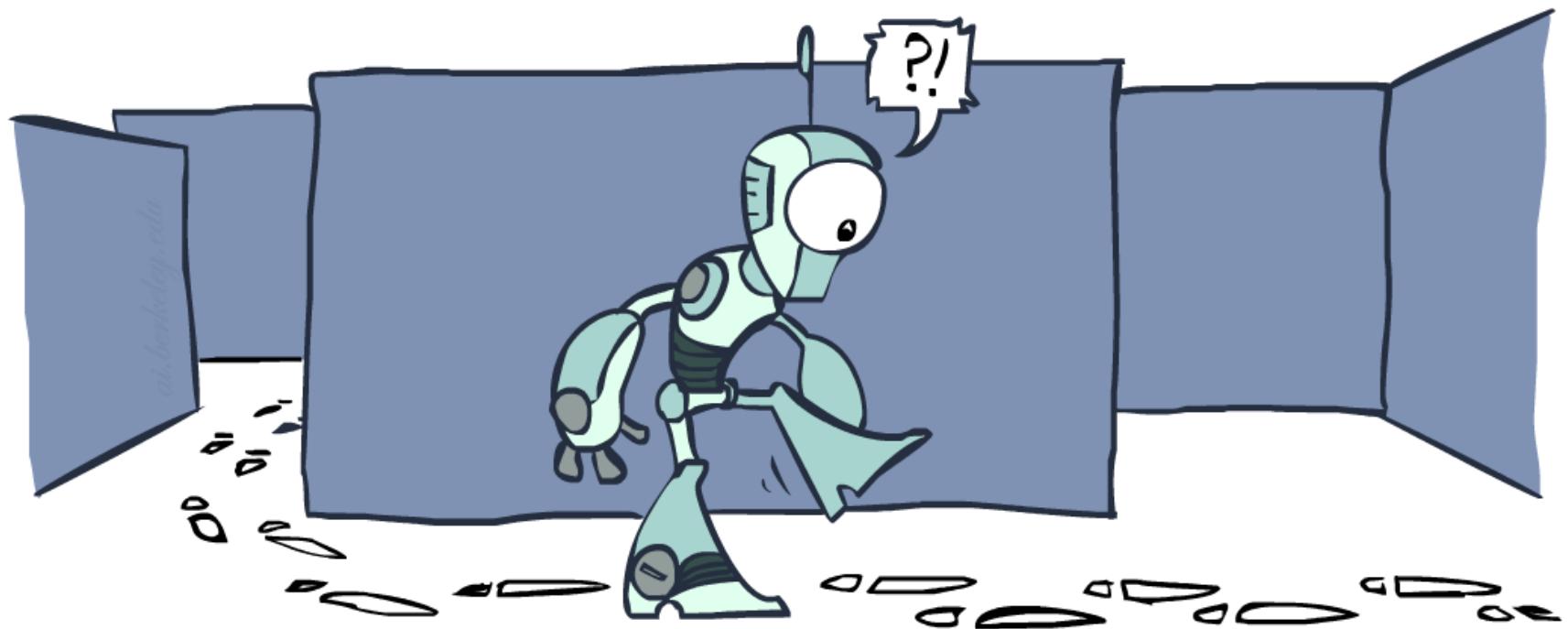
Start: Haugesund, Rogaland, Norway

End: Trondheim, Sør-Trøndelag, Norway

Total Distance: 2713.2 Kilometers

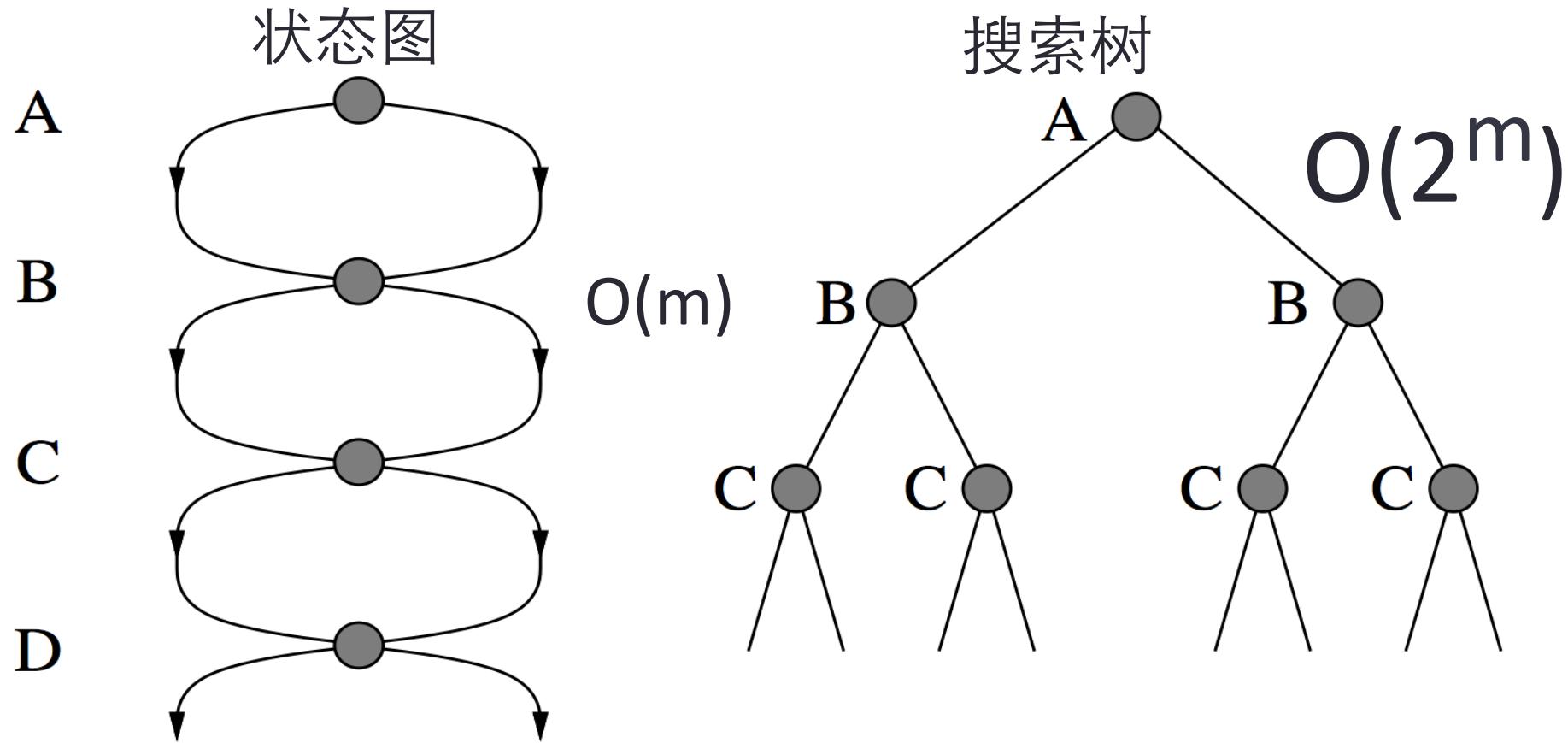
Estimated Total Time: 47 hours, 31 minutes

树搜索 VS 图搜索



树搜索的思考

- 没有检测重复出现的状态，导致产生节点成树深度的指数增长



通用树搜索算法

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

 选择一个叶 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 (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

通用图搜索

Function 图搜索(问题) **returns** 一个 解, 或是失败
把问题的初始状态放入 搜索前沿

初始化已探索节点集为空

loop do

if 探索前沿 为空 **then return** 失败

选择一个叶 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** 相关路径

把这个节点加入已探索节点集

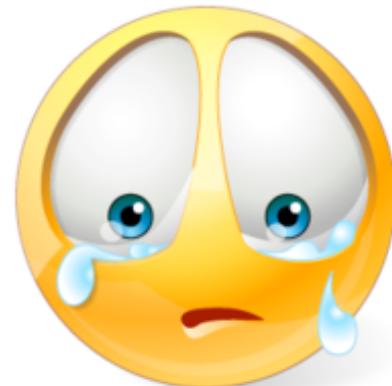
扩展这个选择的 节点, 把扩展结果的 节点 加入 搜索前沿

但是只有当该节点不在搜索前沿或已探索节点集里

树搜索 VS 图搜索

- 图搜索

- 避免无限循环：在一个有限的状态空间里， m 是有限的
- 消除了成指数增长的重复路径
- 要求内存空间不断增长！



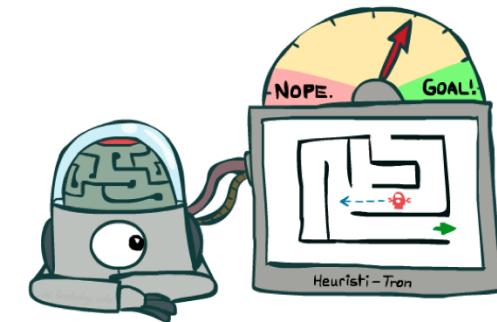
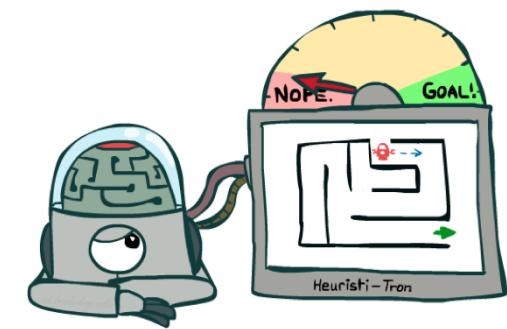
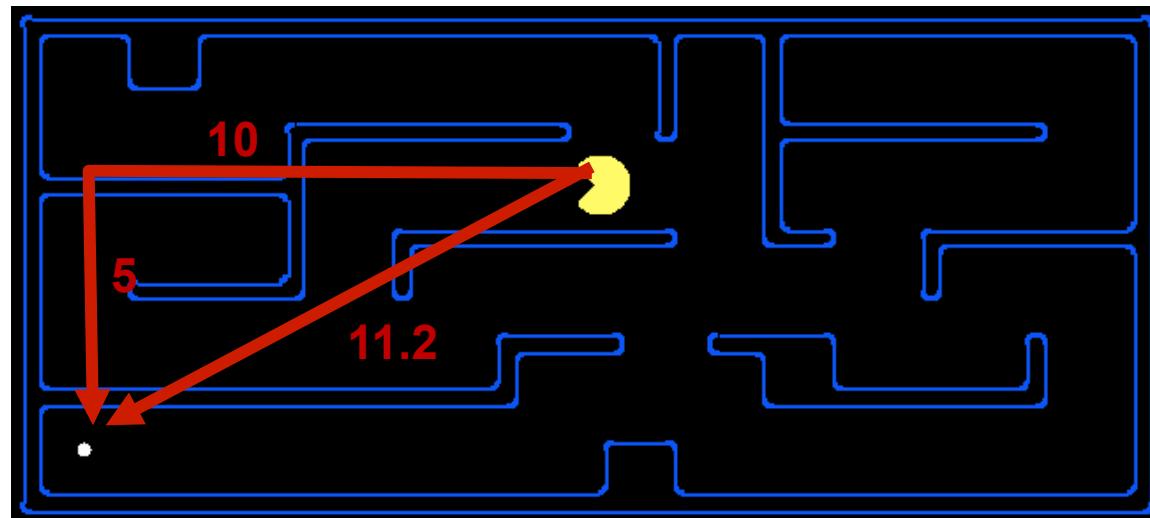
人工智能导论： 启发式搜索

齐琦
海南大学

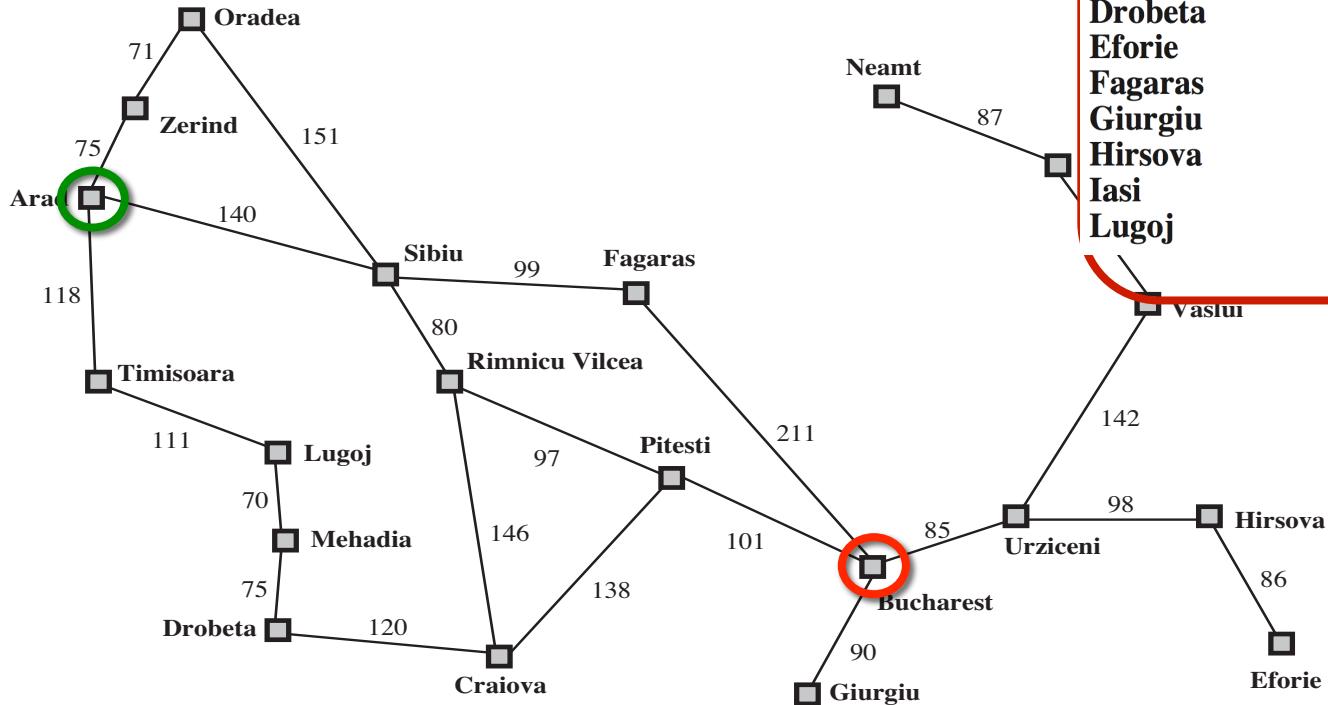
搜索启发法 (Heuristics)

■ Heuristic :

- 一个函数估计一个状态有多接近一个目标
- 为一个特定搜索问题设计的
- 例如：曼哈顿距离，欧几里德距离



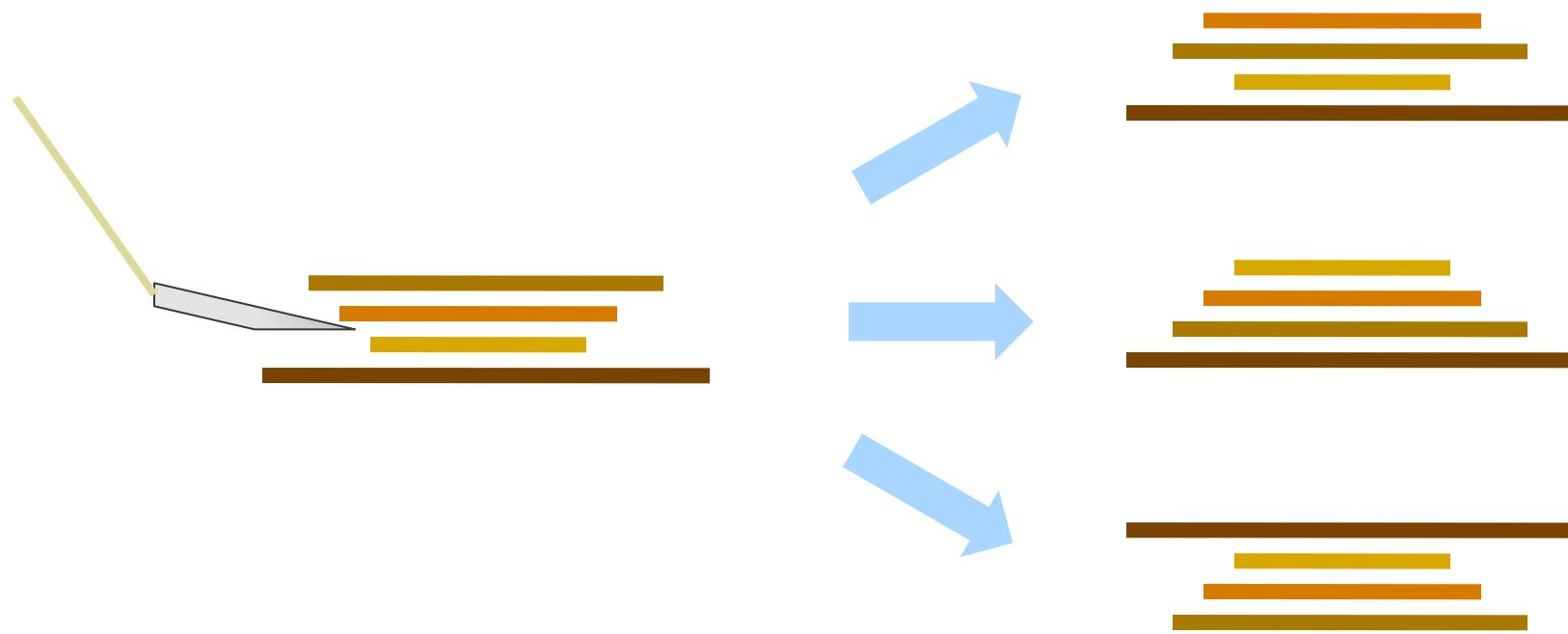
举例：到布加勒斯特的距离（罗马尼亚旅行问题）



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

$h(x)$

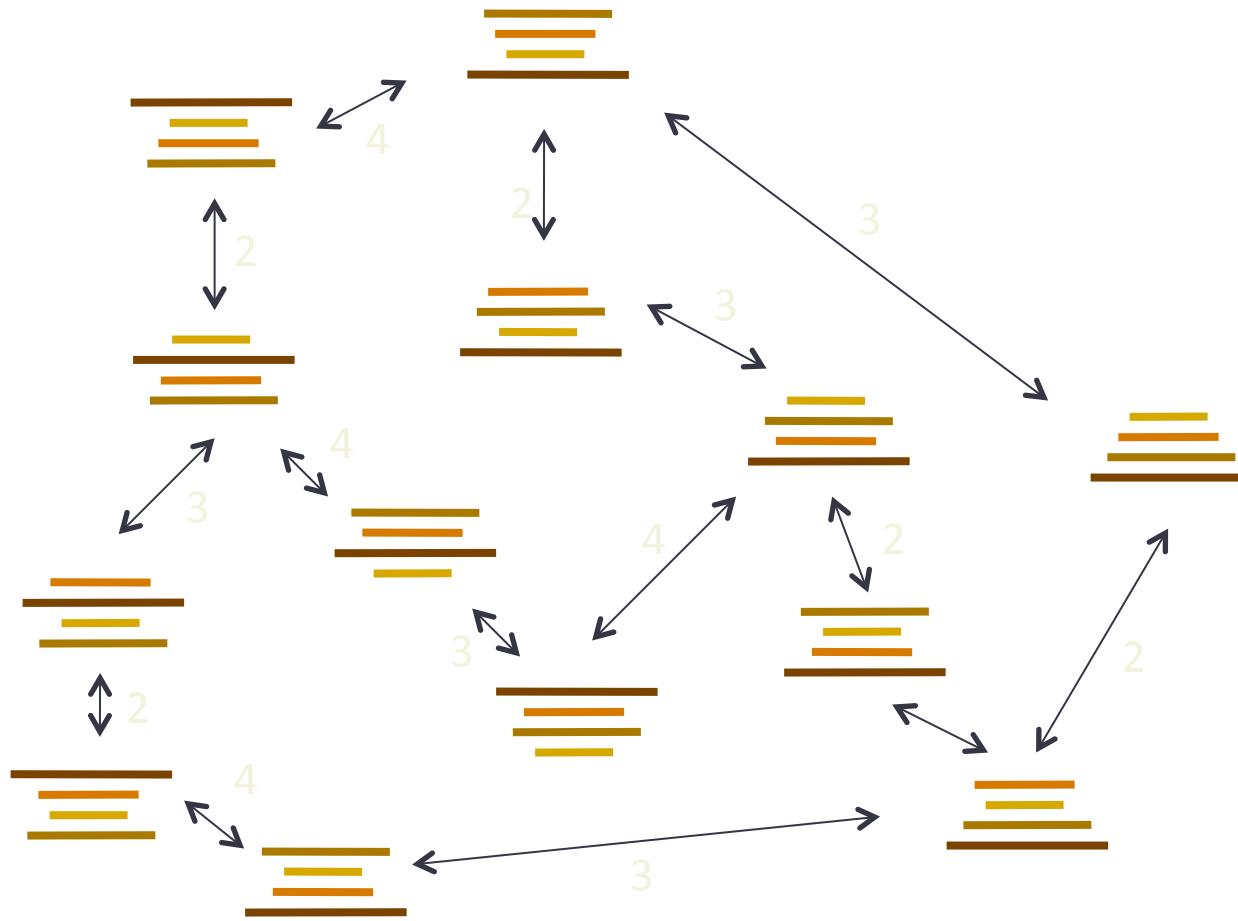
举例：薄饼问题



步骤成本：被翻转薄饼的数量

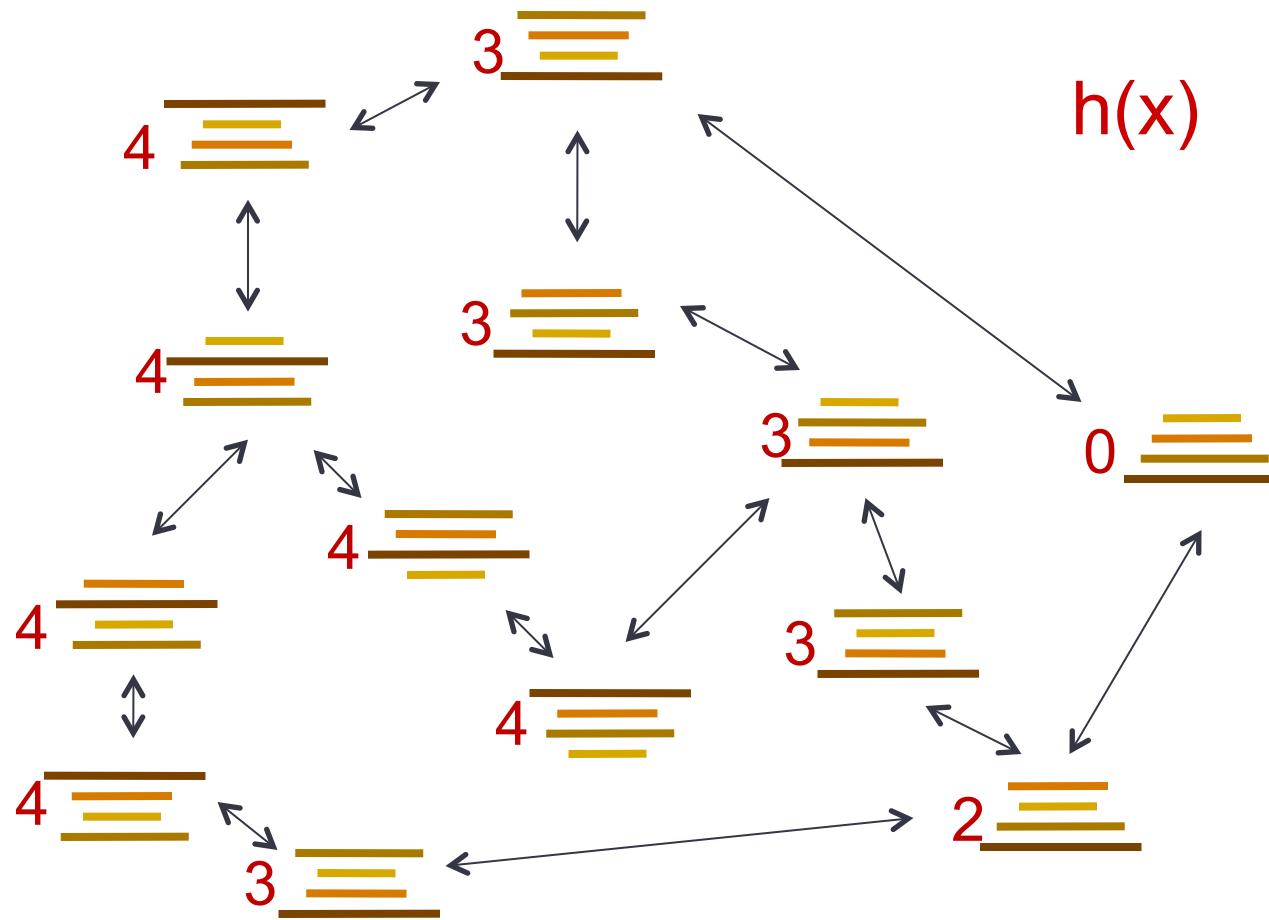
举例：薄饼问题

状态图及其步骤成本



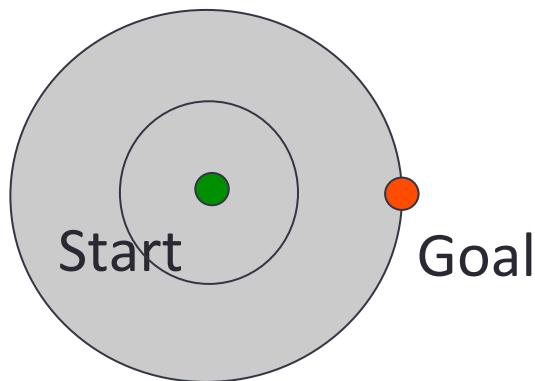
举例：薄饼问题

Heuristic: 还有多少个薄饼没有摆对位置

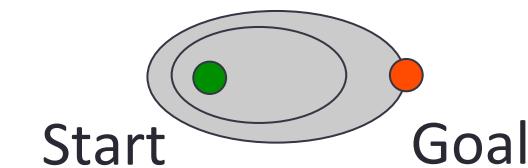


启发式的影响

指引搜索过程朝向目标，而不是朝向各个方向里的所有空间



基础搜索（无信息启发）



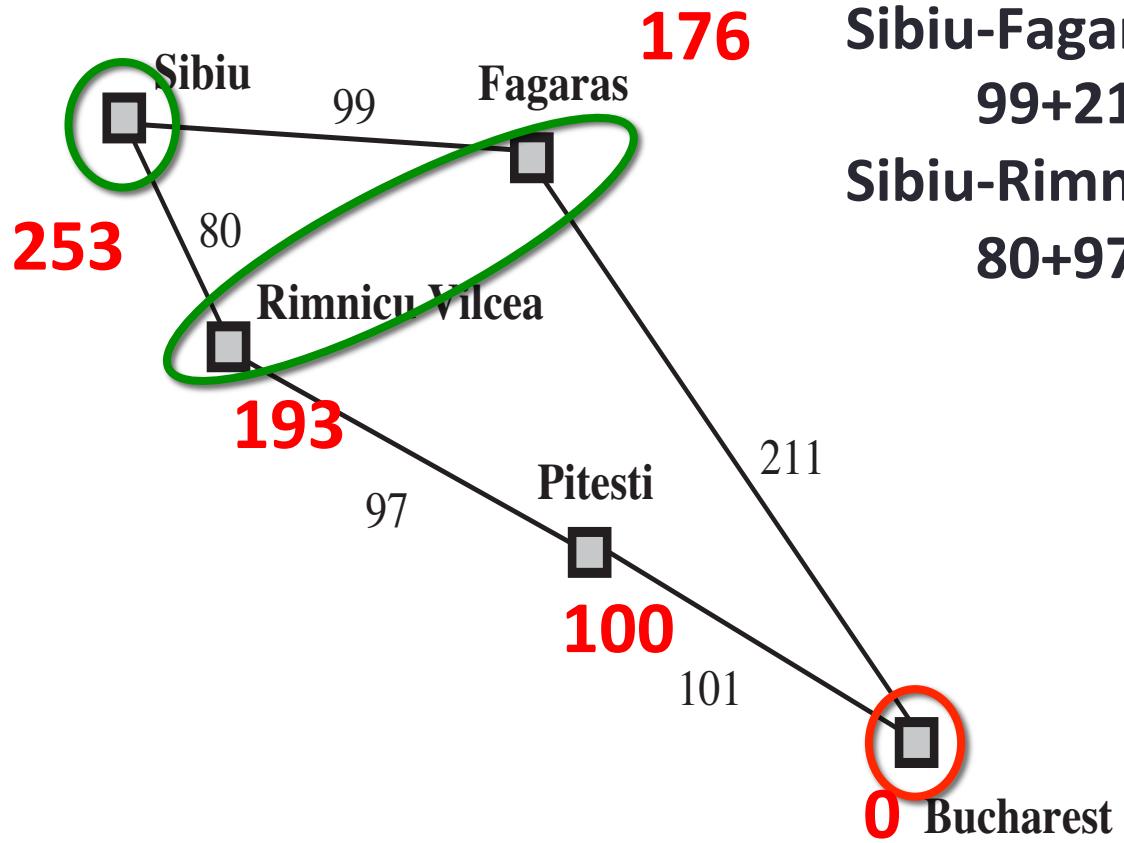
信息启发的

贪婪搜索



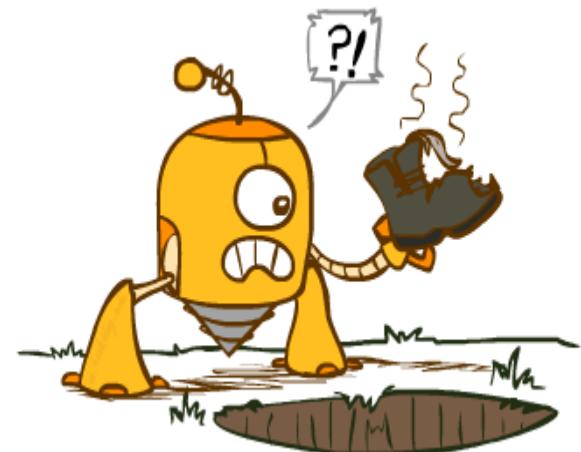
贪婪搜索

- 扩展那个离目标似乎最近的节点（搜索前沿节点按h值排序）
- 会出现什么问题？



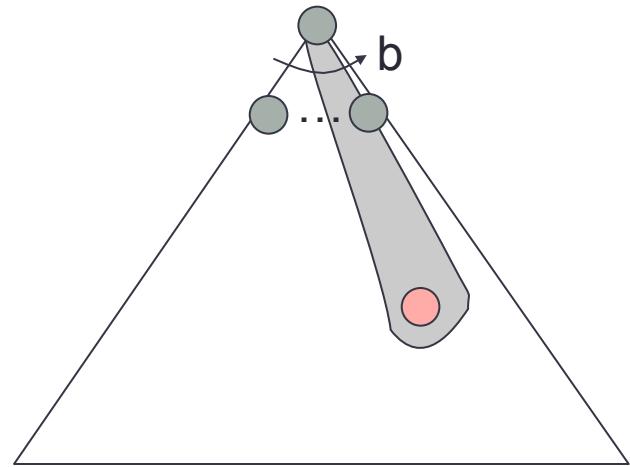
$$\text{Sibiu-Fagaras-Bucharest} = \\ 99 + 211 = 310$$

$$\text{Sibiu-Rimnicu Vilcea-Pitesti-Bucharest} = \\ 80 + 97 + 101 = 278$$



贪婪搜索

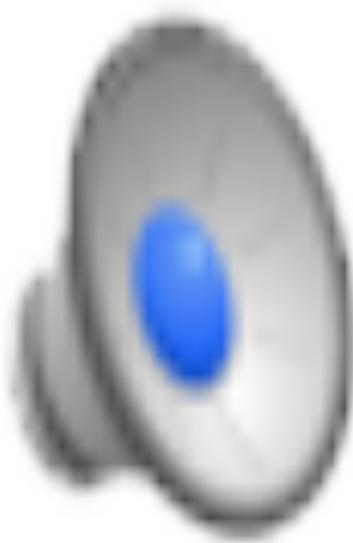
- 策略：扩展一个似乎离目标节点最近的节点，根据 h 值
- 问题1：没有考虑行动（步骤）成本，导致选择的路径可能长而蜿蜒
- 问题2：依赖于 h 值，可是它也有可能完全是错的



视频展示：贪婪算法（空环境里）



视频展示：贪婪算法（Pacman 小迷宫）



A* 搜索



A* 搜索



基于成本的统一搜索法 (UCS)



www.shutterstock.com · 172490930

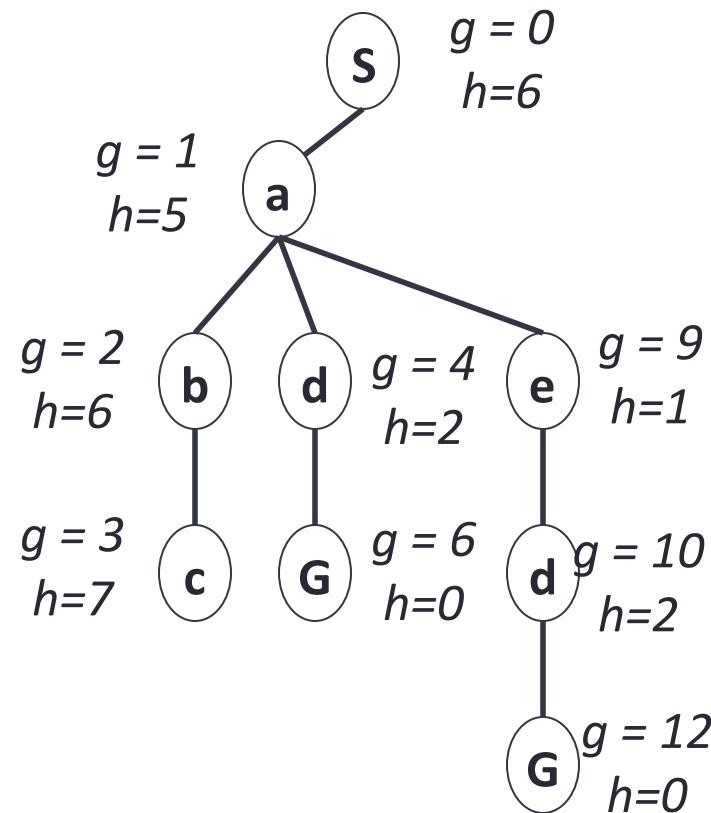
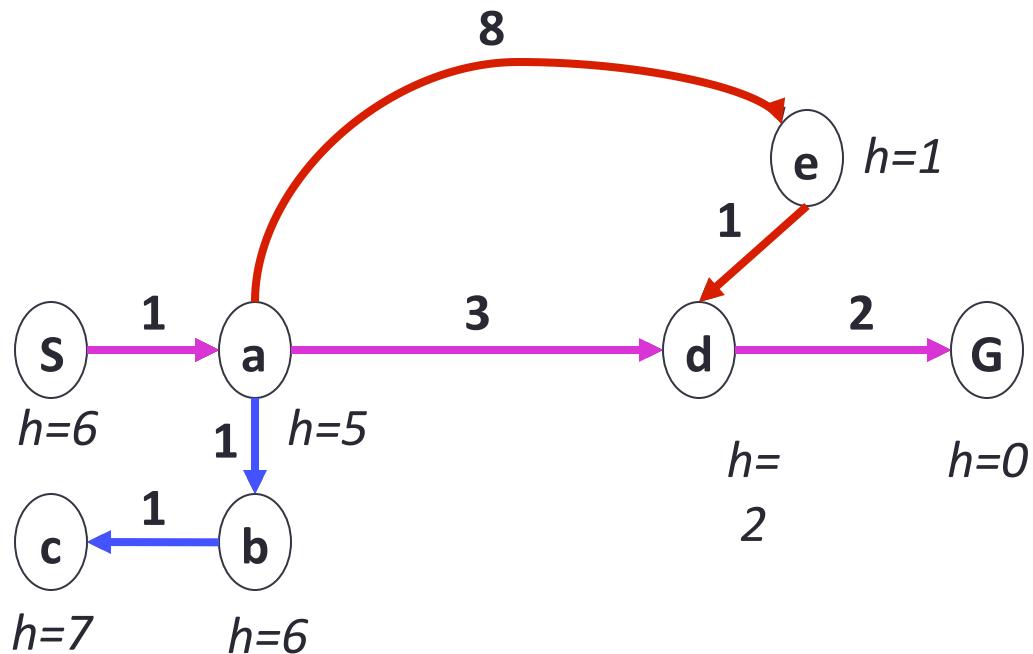


贪婪法 (Greedy)

A*

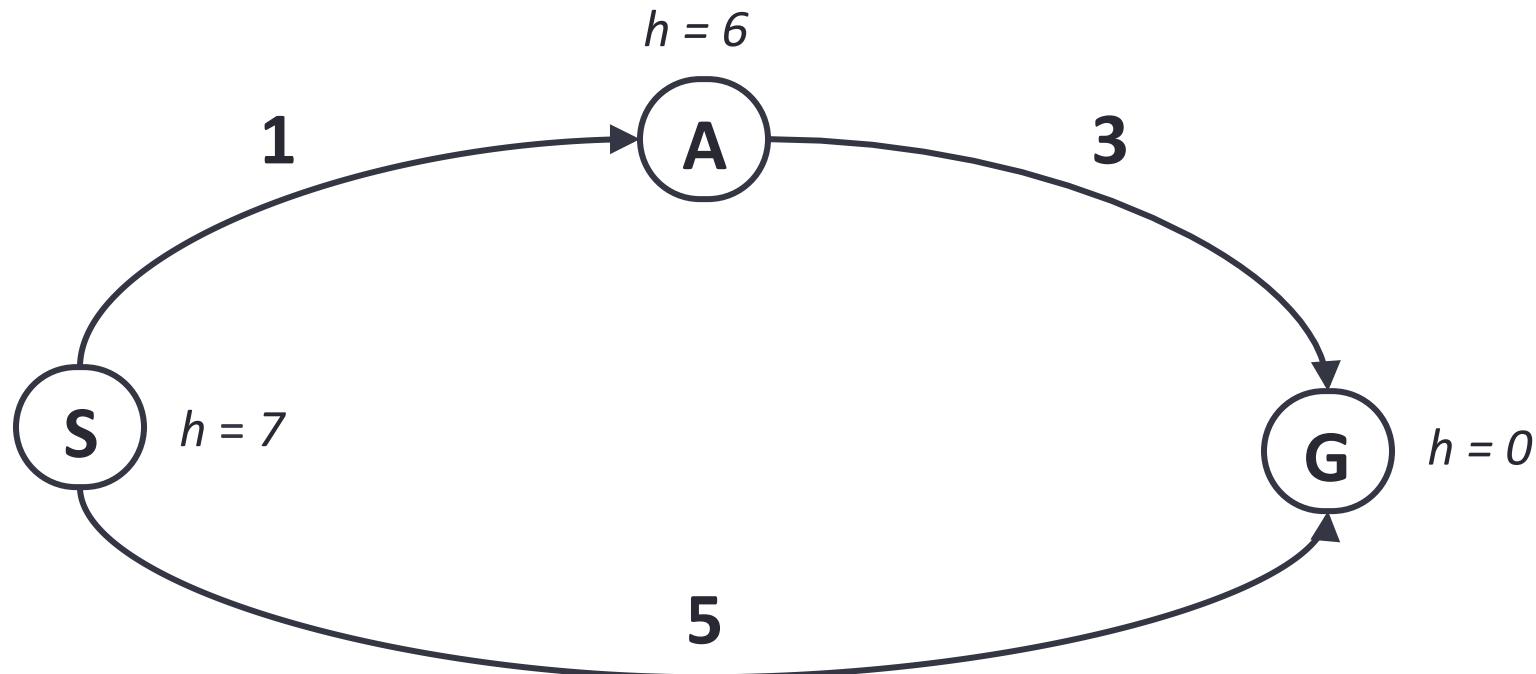
结合统一搜索和贪婪搜索

- 统一：把路径成本排序，即来程的成本 $g(n)$
- 贪婪：排序按照与目标的临近性，即前程成本 $h(n)$



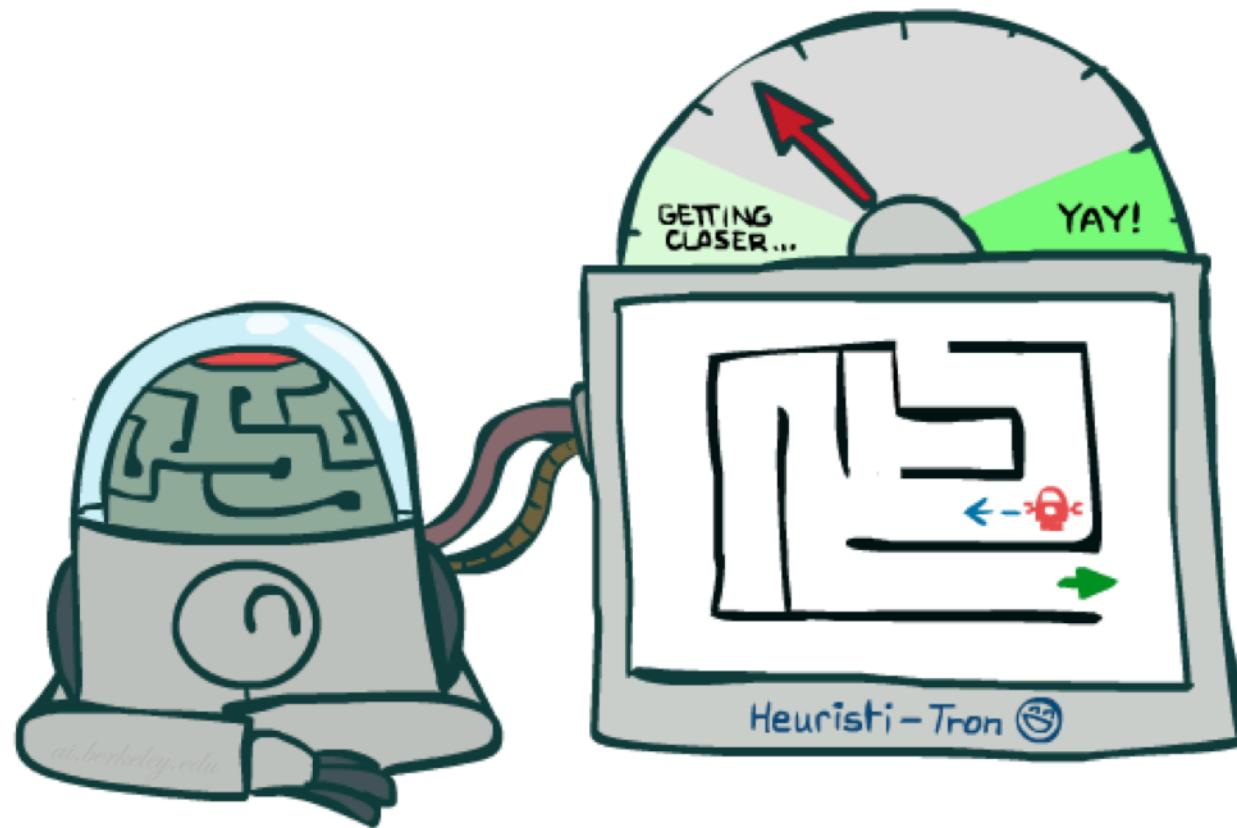
- A* 搜索 : $f(n) = g(n) + h(n)$

A* 是最优的吗?



- 哪个地方错了?
- 实际到目标成本 < 估计的到目标成本
- 我们需要估计值小于实际成本

可接纳的启发式函数



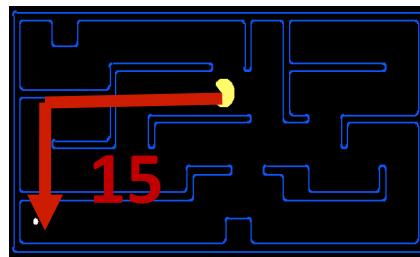
可接纳的启发式函数 (Admissible Heuristics)

- h 是 可接纳的 (乐观的, 想的比实际好) :

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

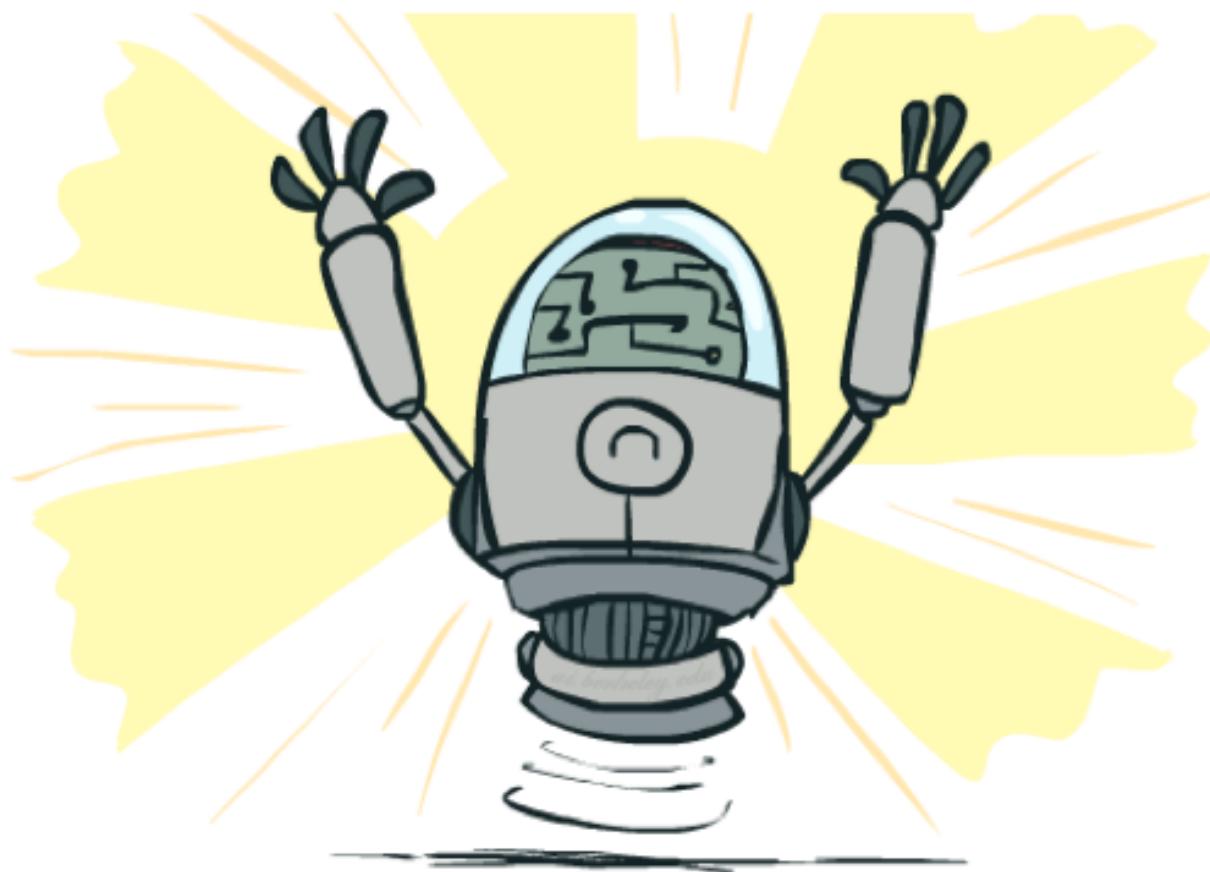
$h^*(n)$ 是到一个最近的目标节点的真成本值

- 举例:



- 在实际应用A*算法时, 一个主要任务就是设计可接纳的启发式函数。

A* 树搜索的最优性



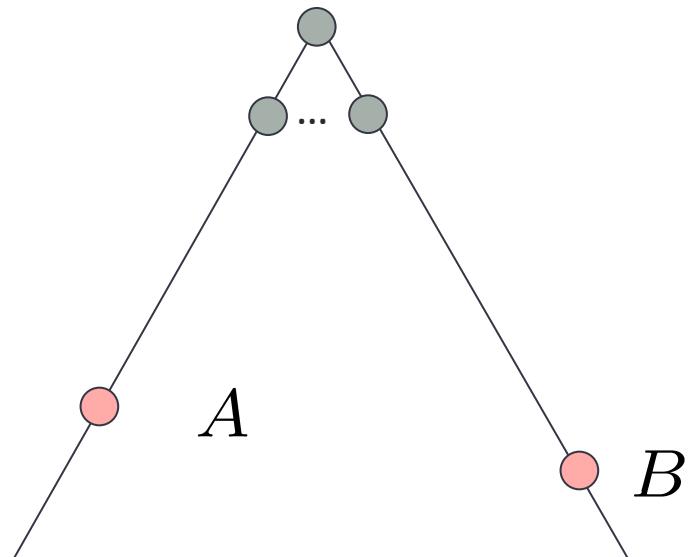
A* 树搜索的最优化

假定：

- A：一个最优目标节点
- B：一个次优目标节点
- h 是可接纳的

如果最优化成立，那么：

- A 将会在B之前被选择先扩展

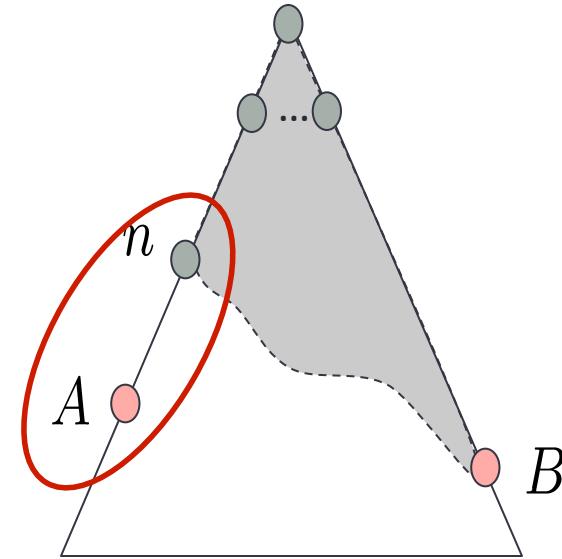


A* 树搜索的最优化

证明：

- 假设B在搜索前沿里
- A的某个祖先节点 n 也在搜索前沿里
- 宣称: n 会先于 B 被选择扩展

1. $f(n)$ 小于或等于 $f(A)$



$$f(n) = g(n) + h(n) \quad f \text{ 的定义}$$

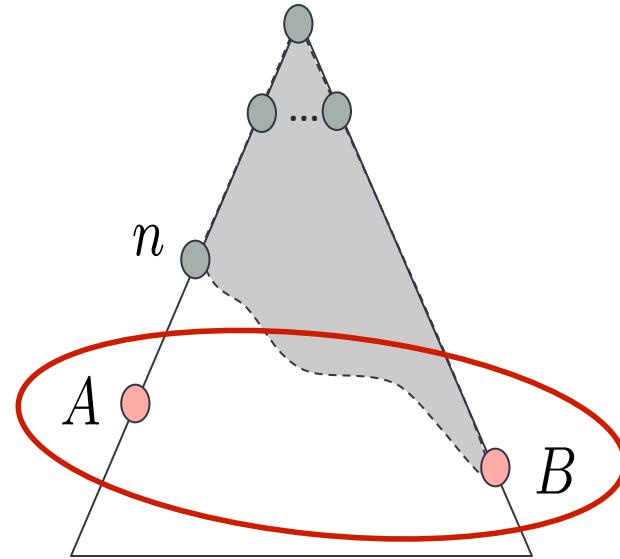
$$f(n) \leq g(A) \quad h \text{ 的可接纳性}$$

$$g(A) = f(A) \quad h = 0 \text{ 当在目标节点}$$

A* 树搜索的最优化

证明：

- 假设B在搜索前沿里
- A的某个祖先节点 n 也在搜索前沿里
- 宣称: n 会先于 B 被选择扩展
 1. $f(n)$ 小于或等于 $f(A)$
 2. $f(A)$ 小于 $f(B)$



$$g(A) < g(B)$$

$$f(A) < f(B)$$

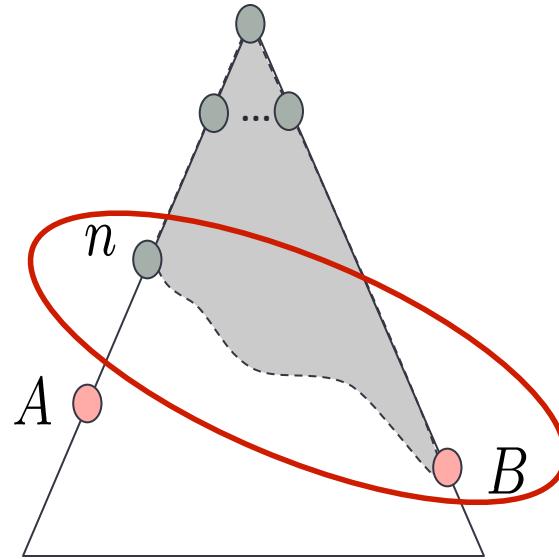
B 次优解

$h = 0$ 在目标节点

A* 树搜索的最优性

证明：

- 假设 B 在搜索前沿里
- A 的某个祖先节点 n 也在搜索前沿里
- 宣称: n 会先于 B 被选择扩展
 1. $f(n)$ 小于或等于 $f(A)$
 2. $f(A)$ 小于 $f(B)$
 3. n 先于 B 被扩展
- 所有 A 的祖先节点都会在 B 之前被扩展
- A 先于 B 被扩展
- A* 搜索是最优的

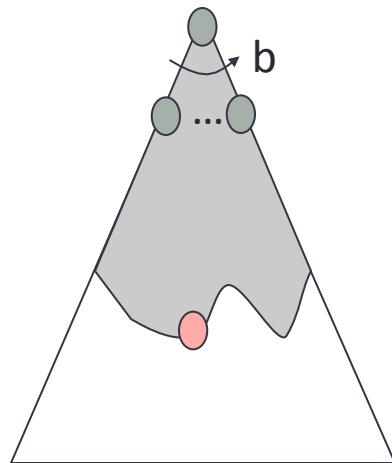


$$f(n) \leq f(A) < f(B)$$

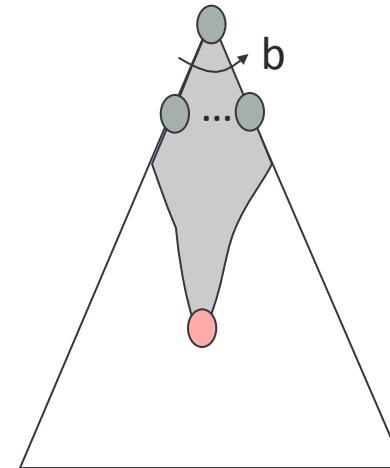
A* 搜索的属性

A*搜索的属性

成本统一搜索

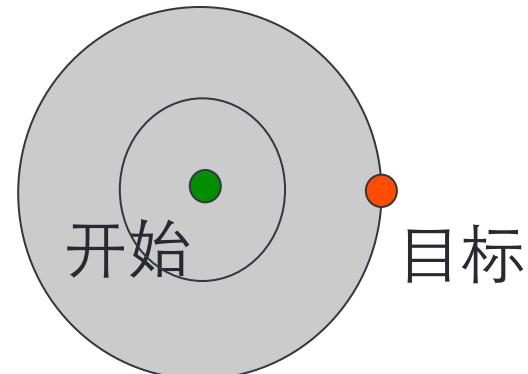


A*搜索

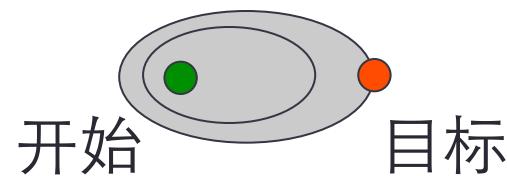


成本统一搜索 vs A* 搜索轮廓

- 基于成本的统一搜索在各个方向上均匀探索



- A* 在朝向目标的方向上进行探索，同时保证解的最优性



视频展示：搜索轮廓（空迷宫）– 基于成本的统一搜索



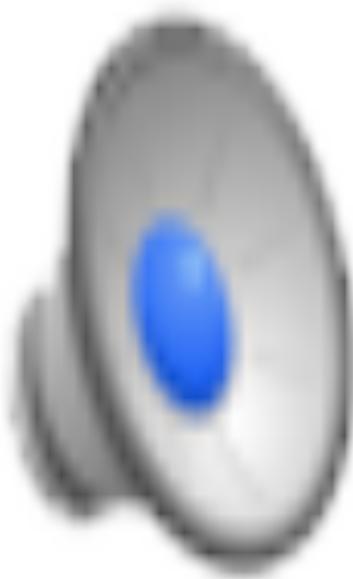
视频展示：搜索轮廓（空迷宫）- 贪婪搜索



视频展示：搜索轮廓（空迷宫）- A*

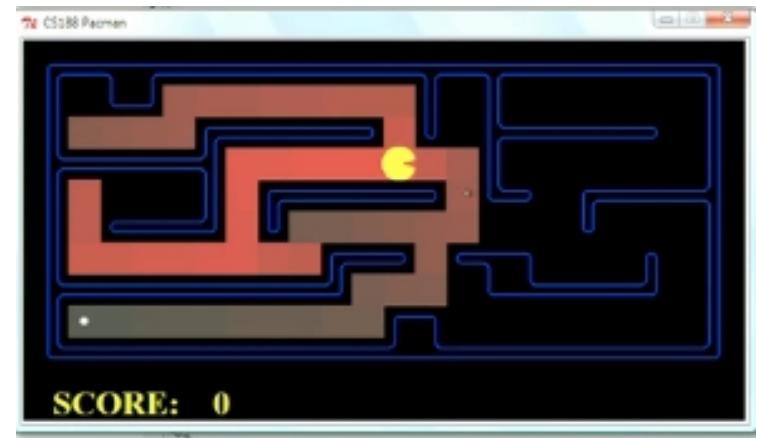
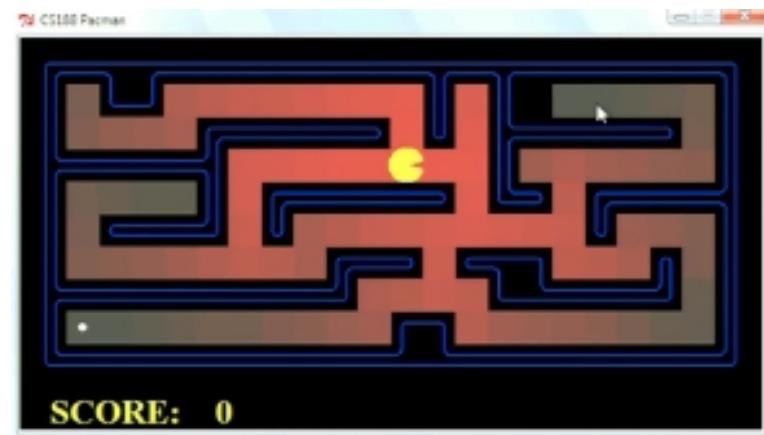


视频展示：搜索轮廓 (Pacman 迷宫) – A*



比较

贪婪搜索

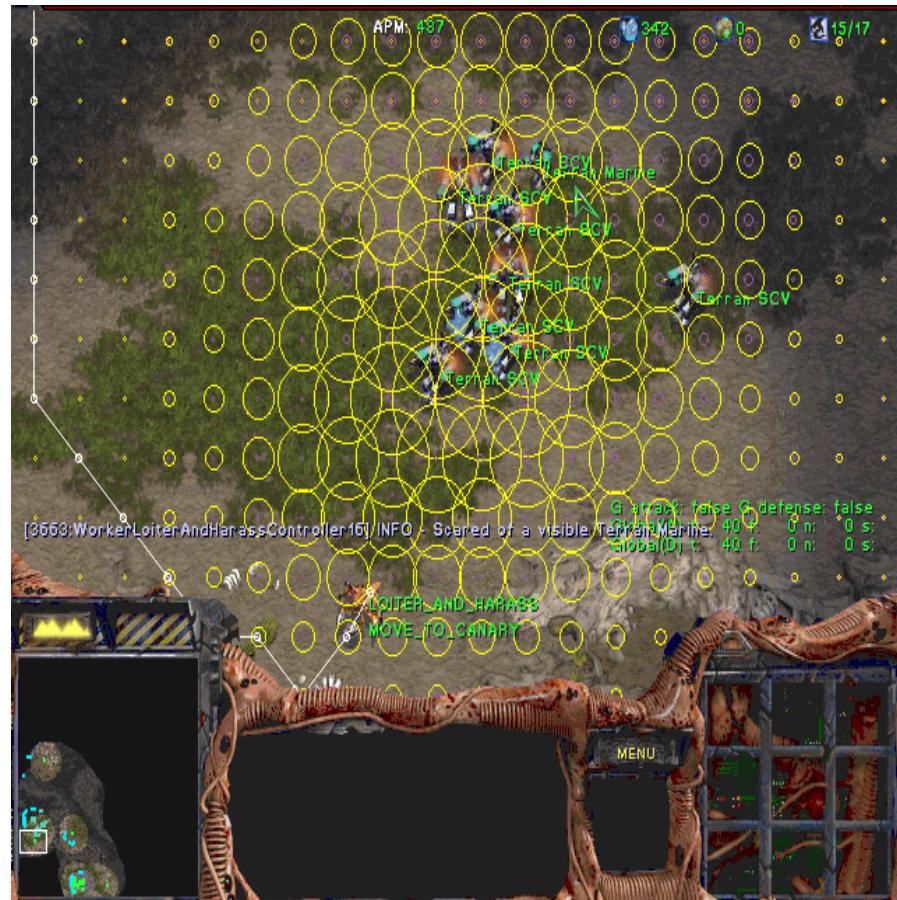


A*

基于成本的统一搜索

A* 应用

- 视频游戏
- 路径搜索问题
- 资源规划问题
- 机器人移动规划
- 语言分析
- 机器翻译
- 语音识别
- ...

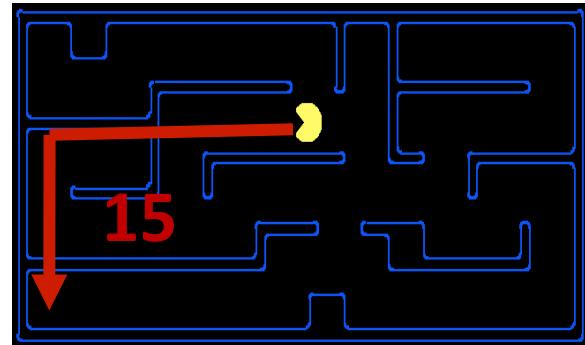
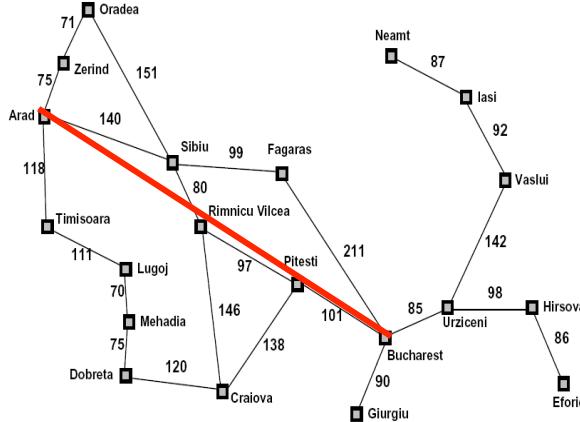


创建启发式函数

创建可接纳的启发式函数

- 在求解很难的搜索问题时，大部分的工作是找到可接纳的启发式函数。
- 可接纳的启发式函数信息，通常是对应的松弛问题的解，解除对行动的限制。

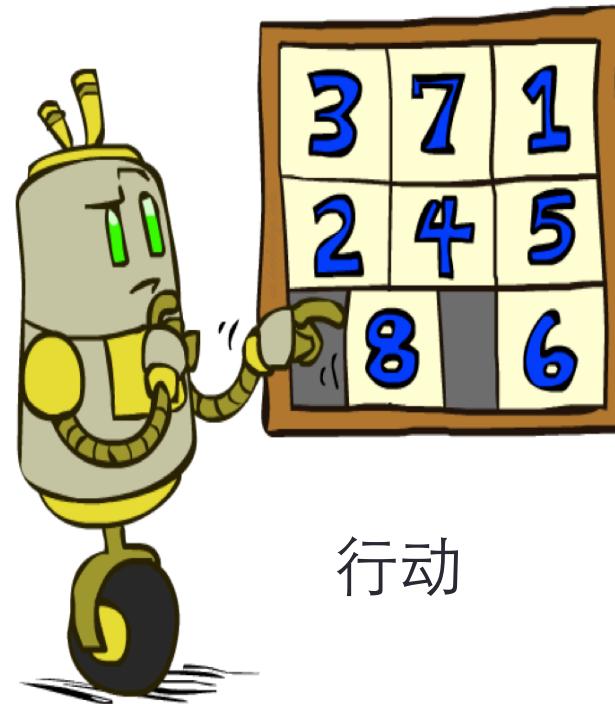
366



举例: 8 数字谜题

7	2	4
5		6
8	3	1

开始状态



行动

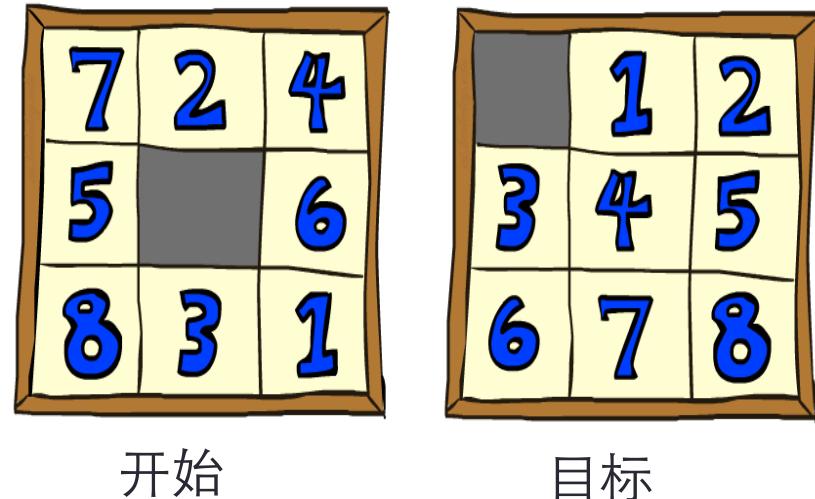
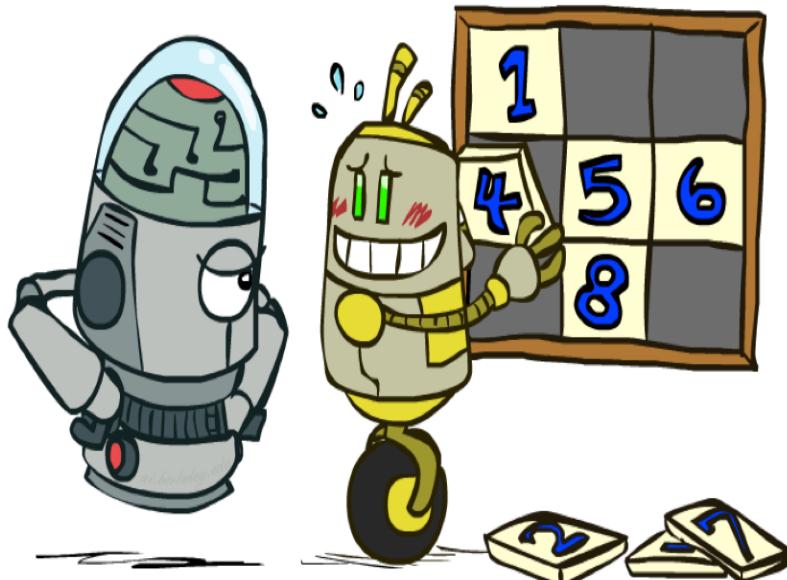
	1	2
3	4	5
6	7	8

目标状态

- 状态?
- 行动?
- 行动成本?

8 数字谜题

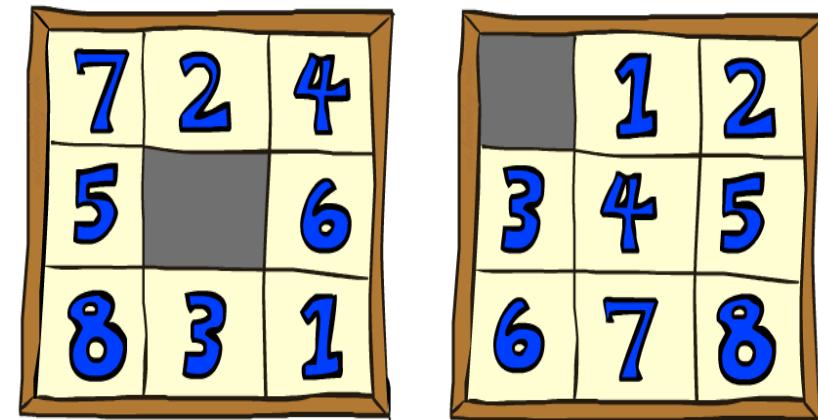
- 启发式: 错位方块的数量
- 为什么是可接纳性的?
- $h(\text{开始}) = 8$
- 松弛问题的启发信息



平均节点扩展数，当最优解的深度是：			
	...4 步	...8	...12
UCS	112	6,300	3.6×10^6
TILES	13	39	227

8 数字谜题

- 如果把条件再松弛一下，任何方块可以在任何时候朝任何方向滑动，无论那里有没有其他方块
- 曼哈顿距离
- 可接纳性的？
- $H(\text{开始}) =$
 $3 + 1 + 2 + \dots = 18$



开始

目标

Average nodes expanded when the optimal path has...			
	...4 steps	...8 steps	...12 steps
TILES	13	39	227
MANHATTAN	12	25	73

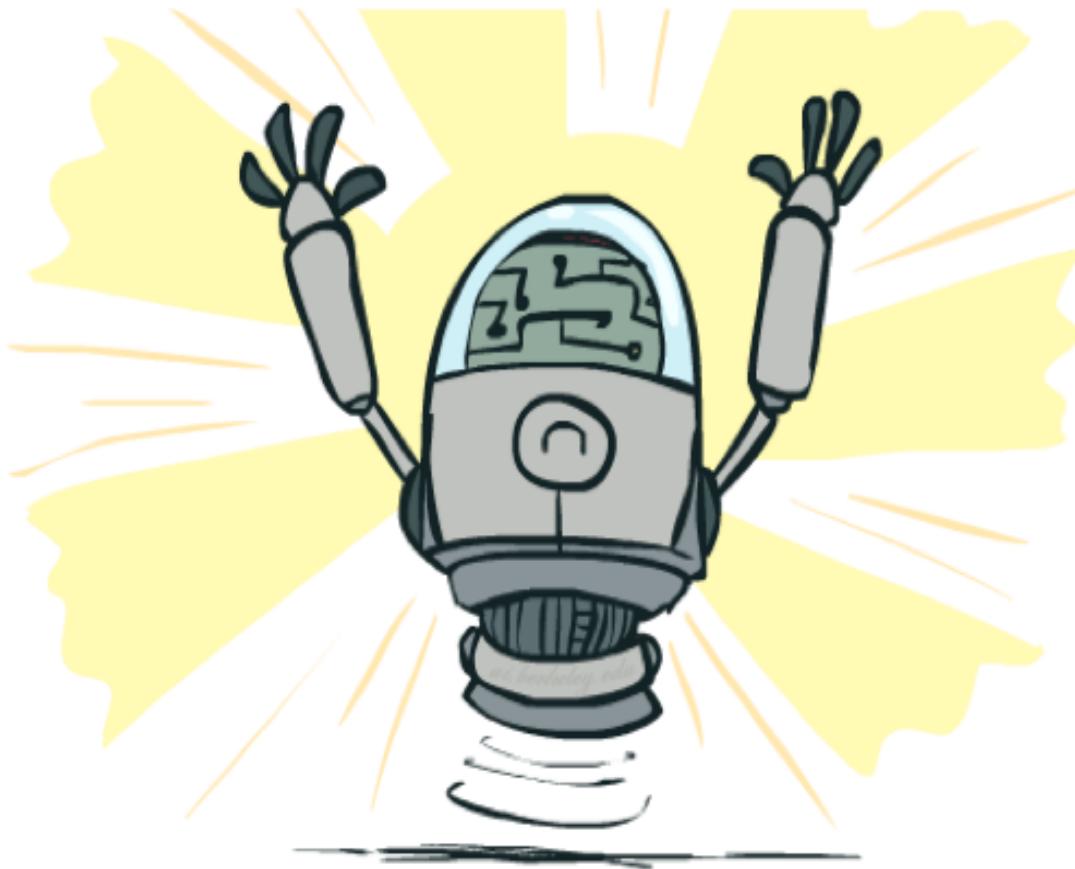
组合启发式信息

- 支配优势: $h_a \geq h_c$ 如果:

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

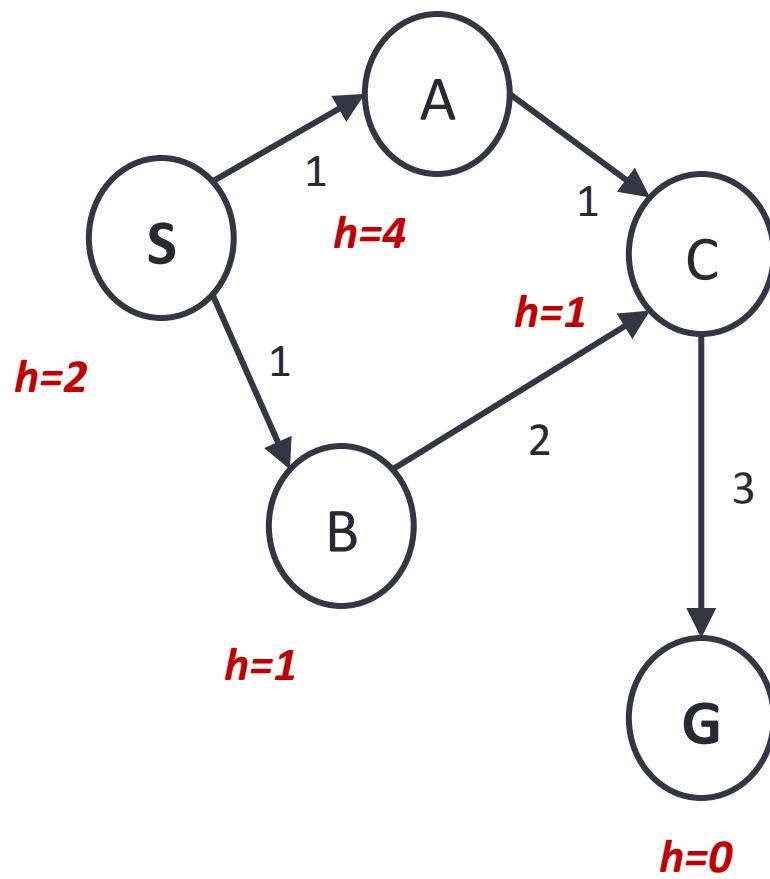
- 一般讲, 越大越好, 只要保持可接纳性。
- H 是 0 的话, 比较糟, (A^* 变成什么, 如果 $h=0?$)。
- 和真实目标成本相同最好, 但很难达到!
- 如果两个启发式函数, 都不支配对方?
 - 形成一个新的, 通过最大式组合:
$$h(n) = \max(h_a(n), h_b(n))$$
 - 这个既是可接纳的, 也是对之前任一个都有支配优势的, 启发式函数。

A* 图搜索的 优化性

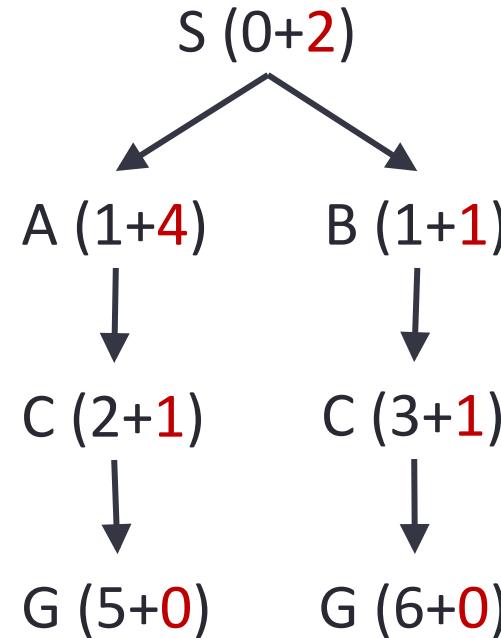


A*图搜索走错了?

状态空间图

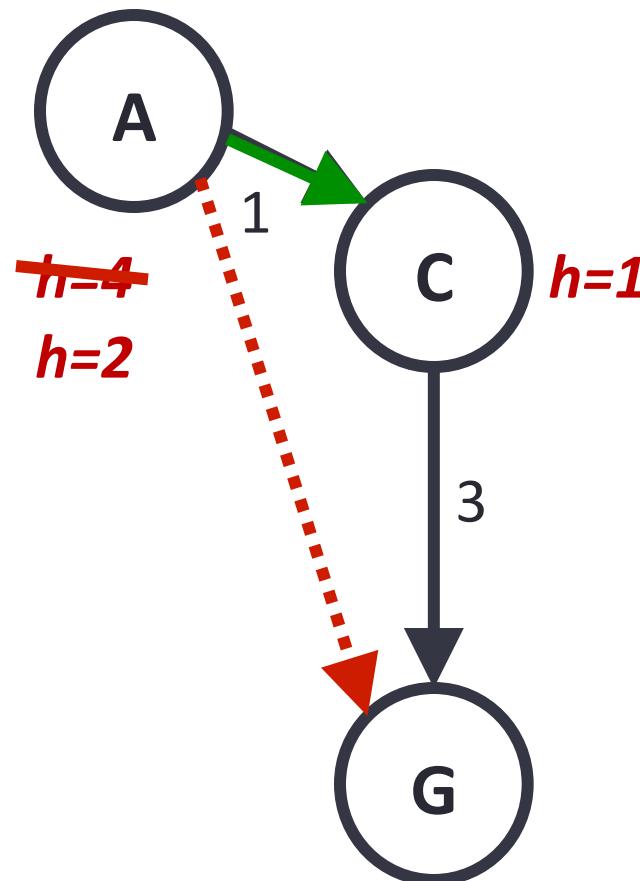


搜索树



C - G 不会被扩展，因为 C 已被访问过

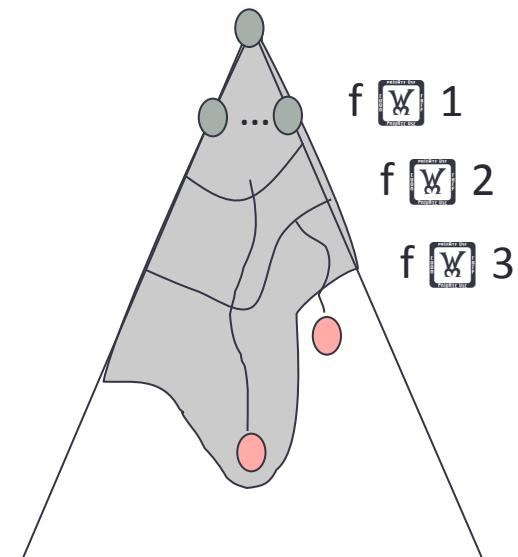
启发性函数的一致性



- 主体思想: 估计成本 \leq 实际成本
 - 可接纳性: 启发函数估计成本 \leq 实际路径成本
$$h(A) \leq \text{从 } A \text{ 到 } G \text{ 的实际路径成本}$$
 - 一致性: 启发函数估计的步骤成本 \leq 实际步骤成本
$$h(A) - h(C) \leq \text{cost}(A \text{ 到 } C)$$
- 一致性的结果:
 - f 值 在同一路径搜索中不会减少
$$h(A) \leq \text{cost}(A \text{ 到 } C) + h(C)$$
 - A* 图搜索是最优的 (找到的解是最优的)

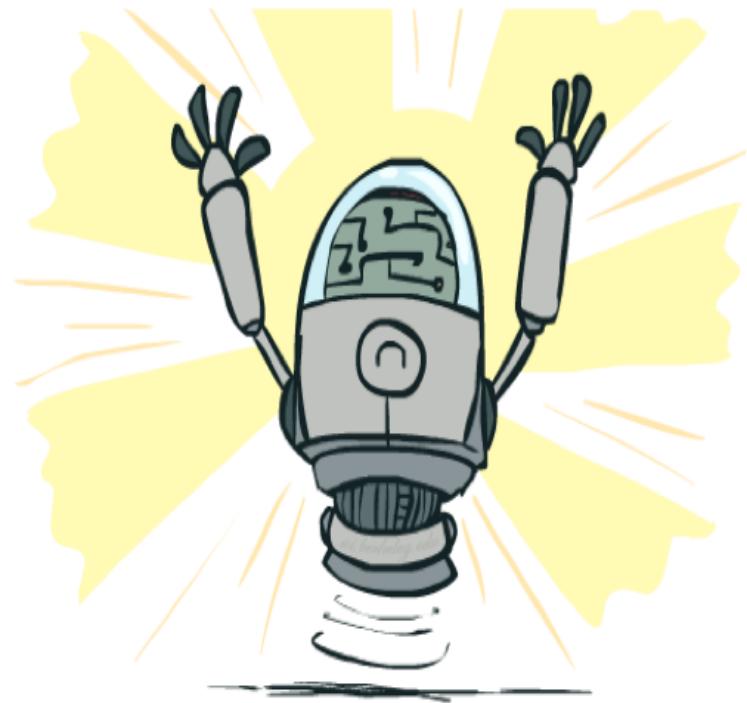
A* 图搜索的最优性 (Optimality)

- 如果 A* 和一个一致性的启发式函数，那么：
 - 事实 1: A* 扩展节点过程中， f 值递增 (f -轮廓)
 - 事实 2: 对于每个状态 s , 到达 s 的最优路径上的节点先于次优路径节点被扩展
 - 结果: A* 图搜索是最优的



最优化

- 树搜索：
 - A* 最优，如果启发式函数是可接纳性的
 - 基于成本的统一搜索(UCS) 是一个特例 ($h = 0$)
- 图搜索：
 - A* 最优，如果启发式函数是一致性的
 - UCS 最优 ($h = 0$ 也是一致性的)
- 一致性导致可接纳性
- 通常，从条件松弛问题中得到的启发式信息既具有可接纳性，又趋向于具有一致性



A*: 总结



A*: 总结

- A* 既使用了来程路径成本，又使用了前程路径成本的估计
- A* 是最优的，如果伴随使用可接纳性的和一致性的启发式函数
- 启发式函数的设计是关键：通常运用条件松弛问题的解

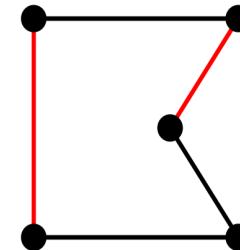
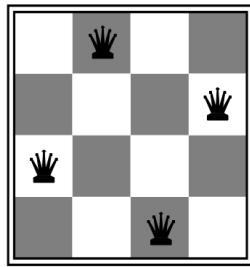


人工智能导论：局部搜索 和智能行为体

齐琦
海南大学

局部搜索

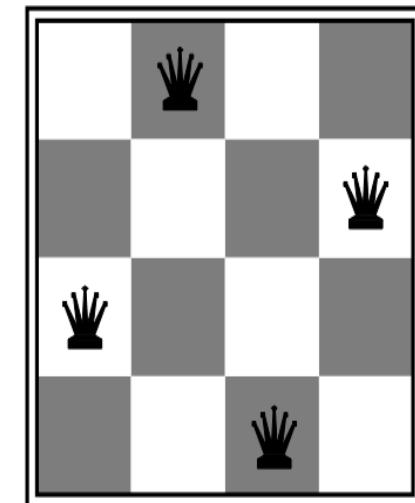
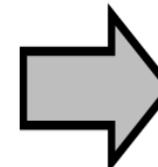
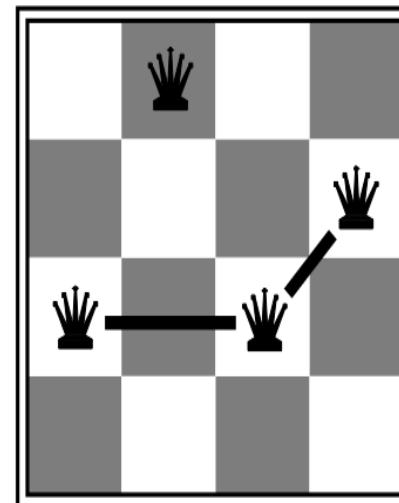
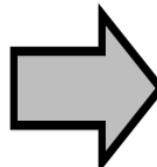
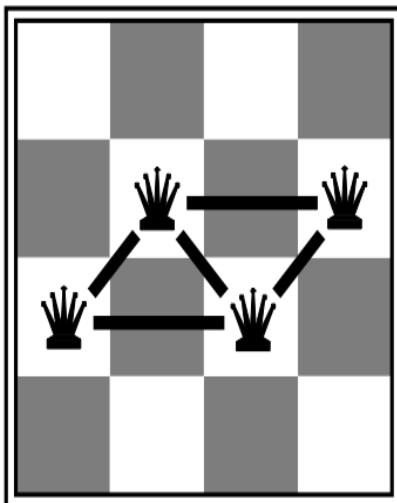
- 许多优化问题, **路径解** 不需要; 目标状态才是 问题的解
- 状态空间 = 完整结构布局的集合
- 找到 **满足约束的结构布局解**, 例如n个皇后问题; 或是找到**最优布局解**, 例如旅行推销商问题



- 在这些情况, 可以使用迭代改进算法; 保持一个单一当前状态, 并尝试改进它。

N 皇后问题的启发式信息

- 目标布局: n 个皇后在棋盘上不互相冲突
- 状态: n 个皇后在棋盘上布局, 一列放一个
- 启发式函数: 相互冲突的皇后对数



$h = 5$

$h = 2$

$h = 0$

爬山算法(Hill-climbing)

function HILL-CLIMBING(问题) **returns** 一个状态

当前节点 \leftarrow make-node(问题.初始状态)

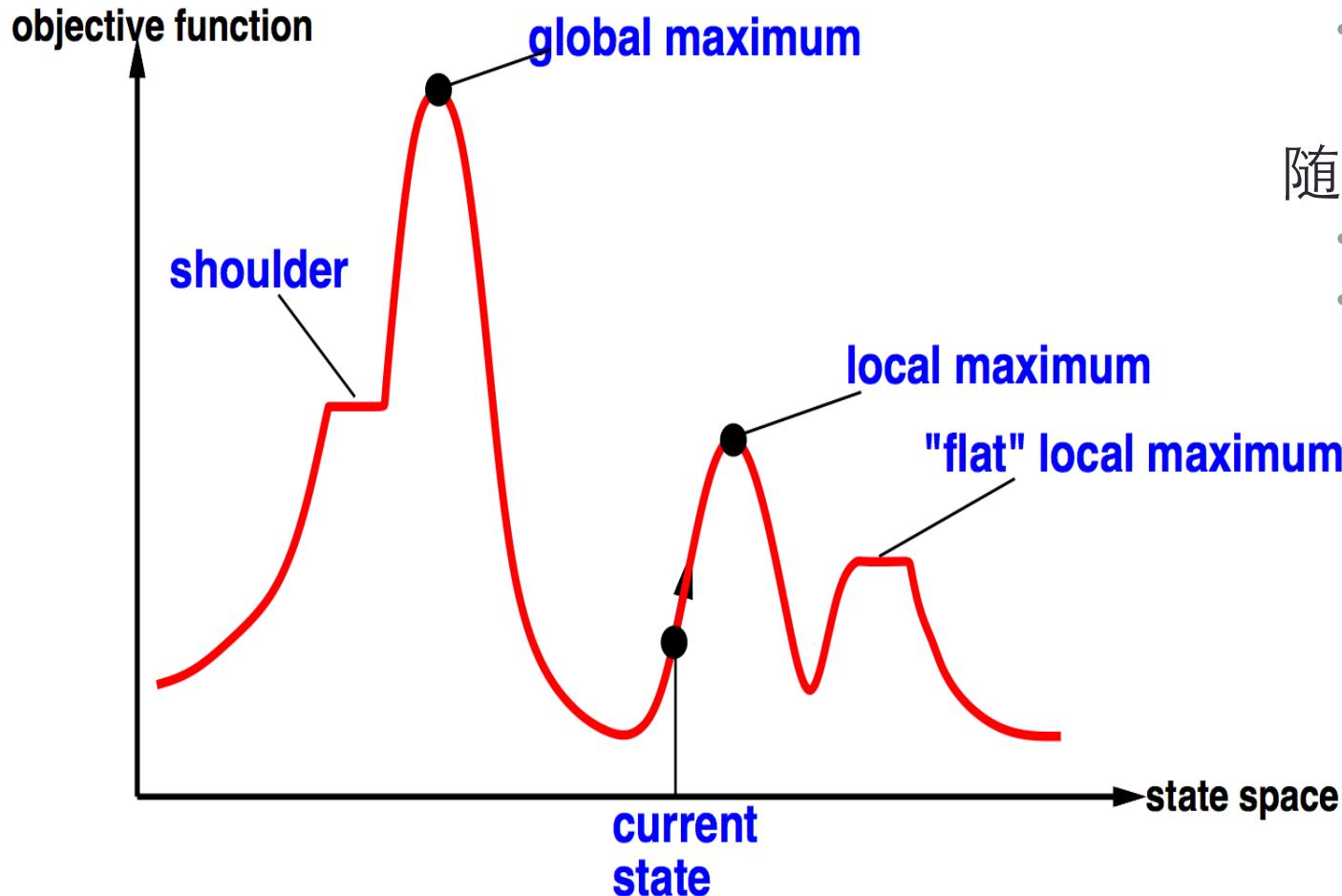
loop do

邻居节点 \leftarrow 选一个 当前节点 的后继节点中评估值最大的节点

if 邻居节点.value \leq 当前节点.value **then return** 当前节点.state

当前节点 \leftarrow 邻居节点

全局和局部最优解



随机开始点

- 找到全局最优

随机水平移动

- 跳出“shoulder”
- 无限循环在“flat local maxima”

模拟退火(Simulated annealing)

- 退火过程用来缓慢冷却金属使之达到一个稳定状态
- 基本想法：
 - 允许偶尔的随机移动，依赖于“温度”
 - 高温 => 更多的随机移动，系统可能走出局部最优格局
 - 逐渐降低温度，根据一个冷却的时间调度
- 理论上：存在一个冷却时间调度，使得找到全局最优的可能概率为1。

模拟退火(Simulated annealing)算法

function SIMULATED-ANNEALING(**问题**,**冷却调度**) **returns** 一个状态

当前节点 \leftarrow make-node(**问题**.initial-state)

for $t = 1$ **to** ∞ **do**

$T \leftarrow$ 调度(t)

if $T = 0$ **then return** 当前节点

下一节点 \leftarrow 一个随机选择的 当前节点 的后继节点

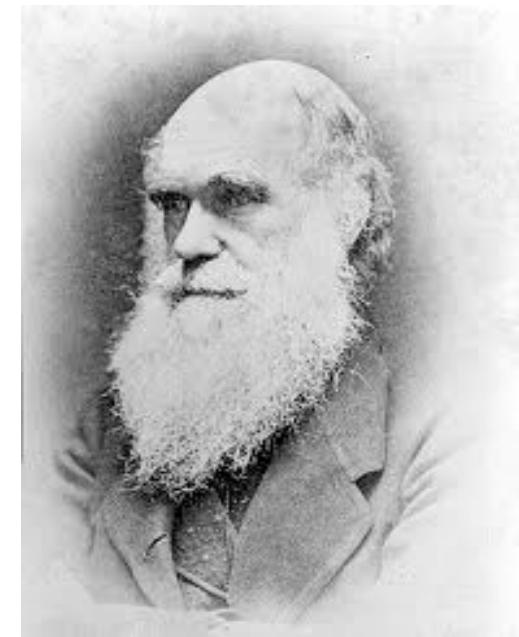
$\Delta E \leftarrow$ 下一节点.value - 当前节点.value

if $\Delta E > 0$ **then** 当前节点 \leftarrow 下一节点

else 当前节点 \leftarrow 下一节点, 如果随机概率大于等于 $e^{\Delta E/T}$

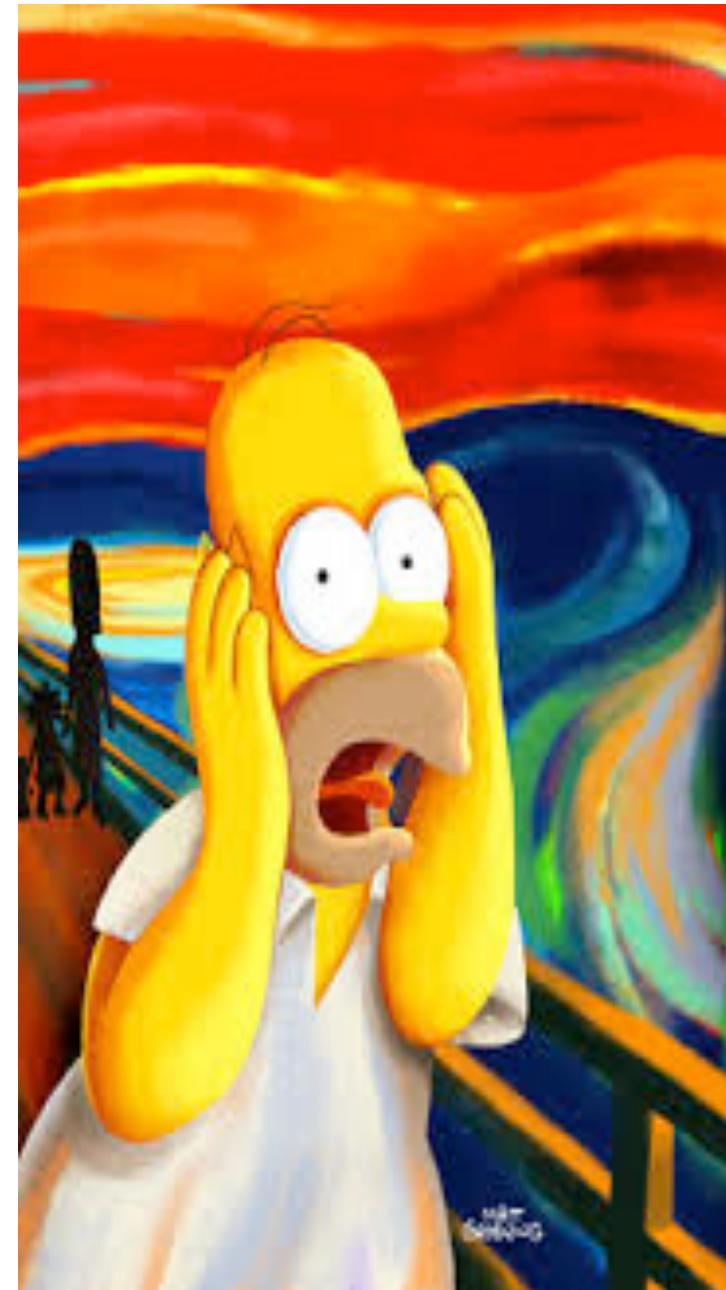
局部光束搜索(Local beam search)

- 基本思想:
 - K 拷贝某种局部搜索算法, 随机初始化
 - 在每一轮
 - 从 k 个当前状态产生所有的后继状态
 - 选出 k 个最好的, 赋给当前搜索状态
- 为什么和开始 K 个并行局部搜索不一样?
 - 搜索间相互 **交流!**
- 还有什么其他的著名算法采用相同的思想?
 - 进化算法!



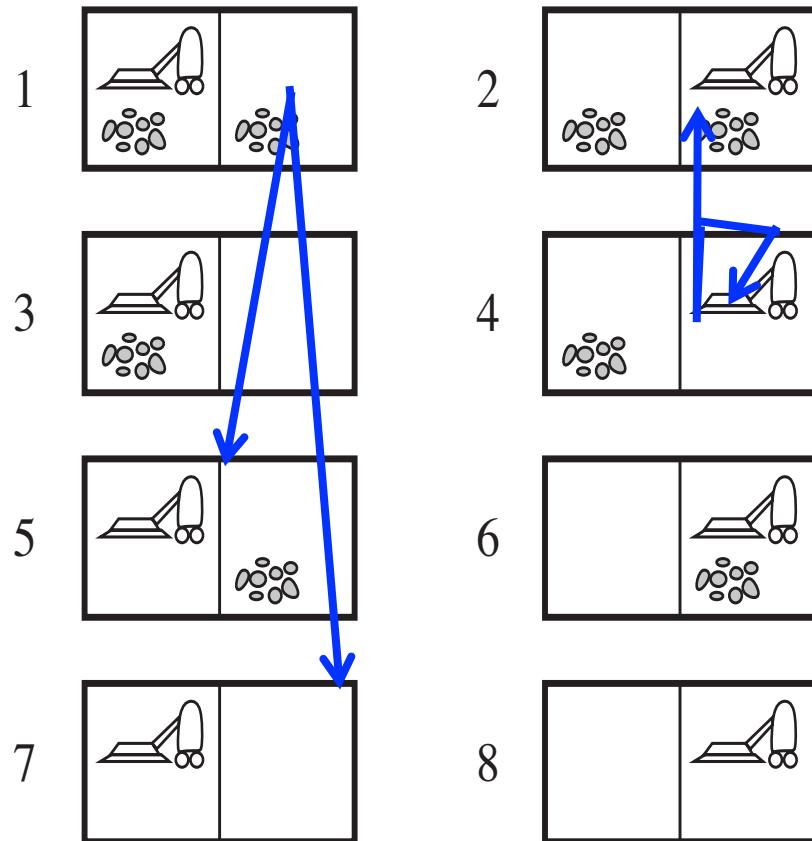
在真实世界里搜索

- 非确定性和部分可观察
- 非确定性: 行动有不可预测的影响
 - 问题构建需允许多个结果状态
 - 解则是 **条件化的规划**
 - 新算法: 与或搜索 (AND-OR)
 - 解规划中也可能有循环步骤!
- 部分可观察: 感知的不是整个状态



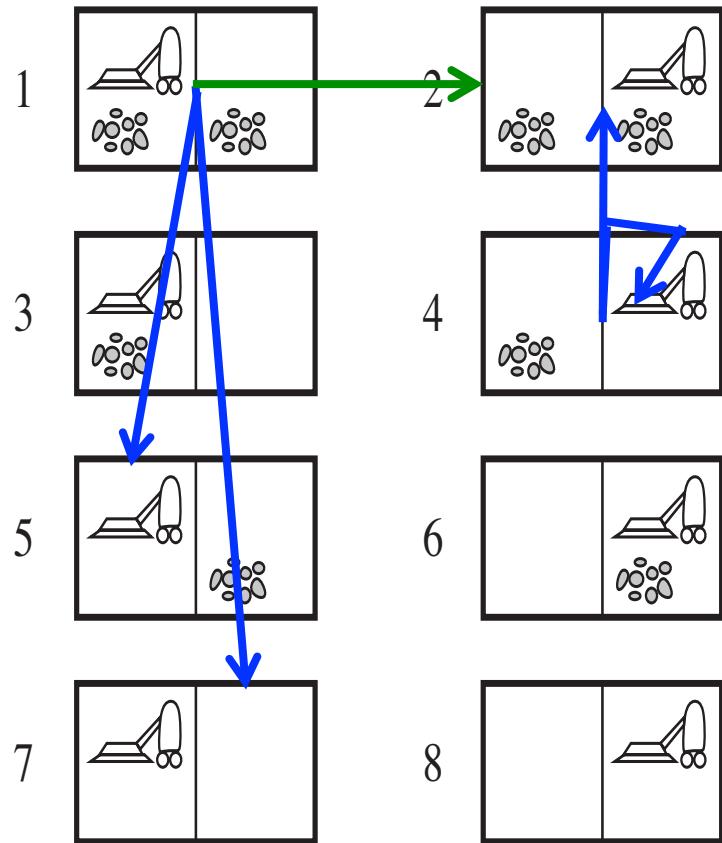
会出故障的吸尘器

- 如果格子有灰尘, 吸尘 可能也吸掉邻近格子里的灰尘
 - 例如, 状态 1 可能会到 5 或 7
- 如果格子是干净的, 吸尘 也可能发生故障, 放些灰尘在
 - 例如, 状态 4 可能会到 4 或 2



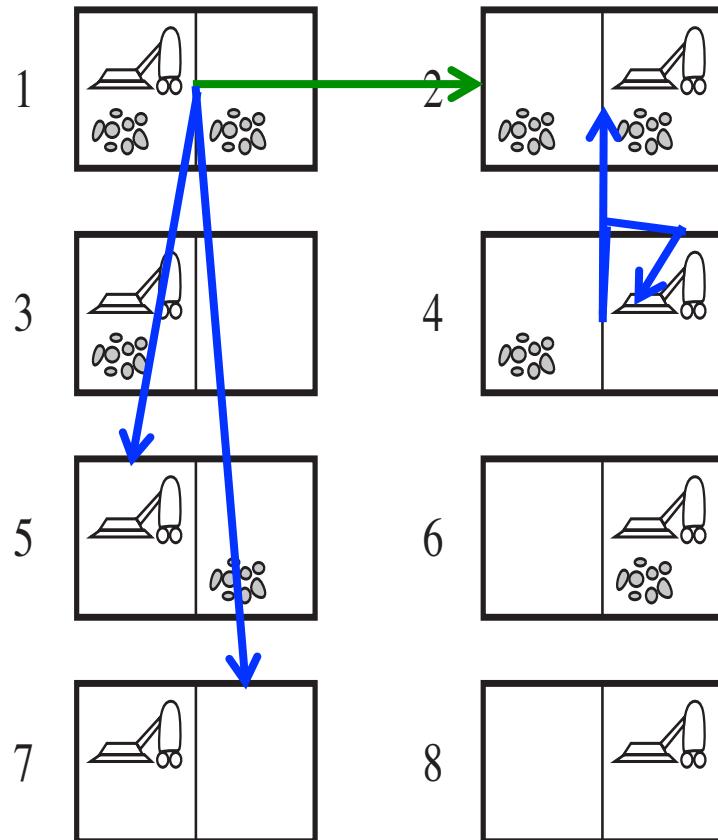
问题建立

- **Results(s,a)** 返回一个状态 集合
 - Results(1,吸尘) = {5,7}
 - Results(4,吸尘) = {2,4}
 - Results(1,向右) = {2}
- 其他的都和以前一样



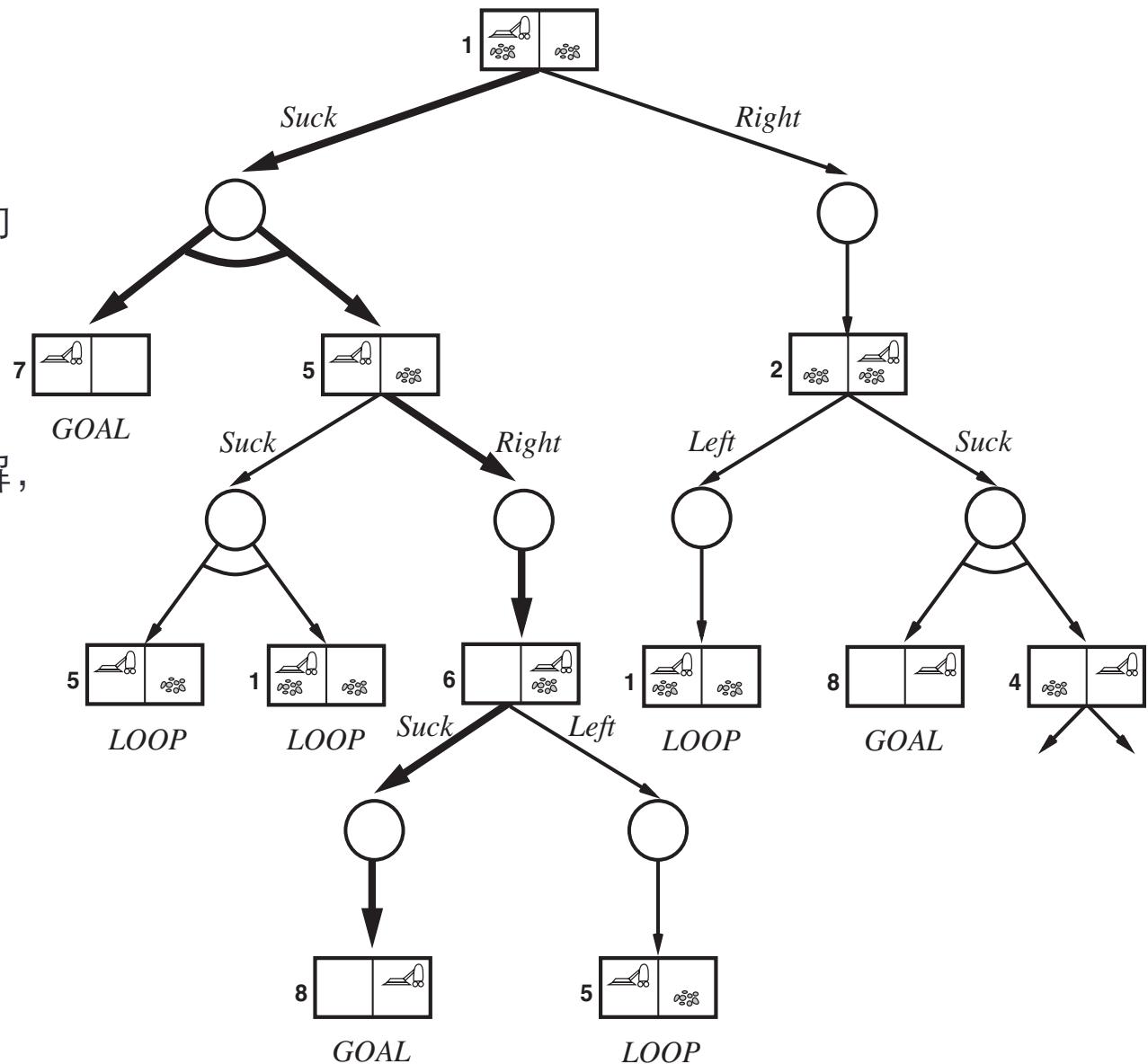
条件化的解

- 从状态 1, 行动 [吸尘] 能解决问题吗?
 - 不必然!
- 那么 [吸尘,右移,吸尘]?
 - 不必然!
- [吸尘; **if** 状态=5 **then** [右移,吸尘] **else** []]
- 这是一个 **依情况而定的解**
(条件性规划)
- 如何找到这样的解?



与或搜索树

- 或-节点:
 - 智能体选择行动;
 - **至少一个分支** 能得到解即成功
- 与-节点:
 - 自然选择行动的影响;
 - **所有分支** 必须都能被求解, 否则失败
- 两种节点交替排列



与或搜索算法 (递归，深度优先)

Function 与或-图-搜索(问题) **returns** 一个条件性规划, 或失败
或-搜索(问题.初始状态,问题,[])

Function 或-搜索(状态,问题,路径) **returns** 一个条件性规划, 或失败

if 问题.goal-test(状态) **then return** 空规划

if 状态 曾出现在 路径 **then return** 失败

for each 行动 **in** 问题.actions(状态) **do**

规划 \leftarrow 与-搜索(results(状态,行动),问题,[状态 | 路径])

if 规划 \neq 失败 **then return** [行动 | 规划]

return 失败

与或搜索，继续

Function 与-搜索(状态集,问题,路径) **returns** 一个条件规划, 或失败

for each s_i **in** 状态集 **do**

$plan_i \leftarrow OR\text{-SEARCH}(s_i, \text{问题}, \text{路径})$

if $plan_i = \text{失败}$ **then return** 失败

return [**if** s_1 **then** $plan_1$ **else if** s_2 **then** $plan_2$ **else** . . . **if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]

部分可观察环境

- 信念状态
 - 智能体可能会处在的状态集合
- 搜索问题需在信念空间里搜索；并添加观察模型
- 遵循通常的智能行为体设计过程

总结

- 非确定性（行动导致的） 要求解是条件化的规划
 - 通过与或搜索寻找解
- 无感知问题 的解是通常的行动规划
 - 通过在信念状态空间中去寻找解
- 普通的部分可观察环境 导致 感知上的非确定性
 - 在信念状态空间里进行与或搜索

