



南京大學

NANJING UNIVERSITY

人工智能导论

有信息/启发式搜索

(Informed/Heuristic Search)

郭兰哲

南京大学 智能科学与技术学院

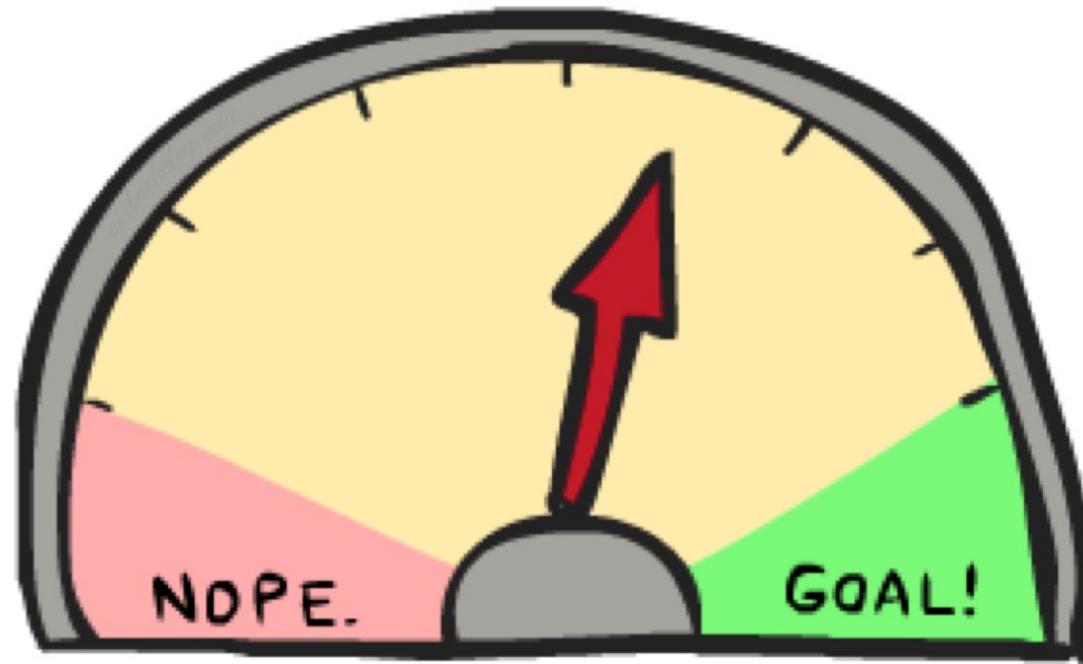
<https://www.lamda.nju.edu.cn/guolz>

Email: guolz@nju.edu.cn

Recall

- 一个搜索问题可以用5个组成部分形式化描述
 - 状态空间、初始状态、目标状态、动作、状态转移、代价
- 搜索问题的求解
 - 从初始状态出发，按照某种策略不断从边缘节点集合选择节点并扩展，直到达到目标状态或者没有状态可以扩展
 - 树搜索和图搜索：是否允许重复扩展同一节点
- 几种不同的无信息搜索策略：
 - 宽度优先搜索(BFS)、广度优先搜索(DFS)、迭代加深的深度优先搜索(IDDFS)、一致性代价搜索(UCS)

有信息搜索 (Informed Search)



提纲

□ 启发式搜索

➤ 贪婪搜索

➤ A*搜索

➤ 启发式函数

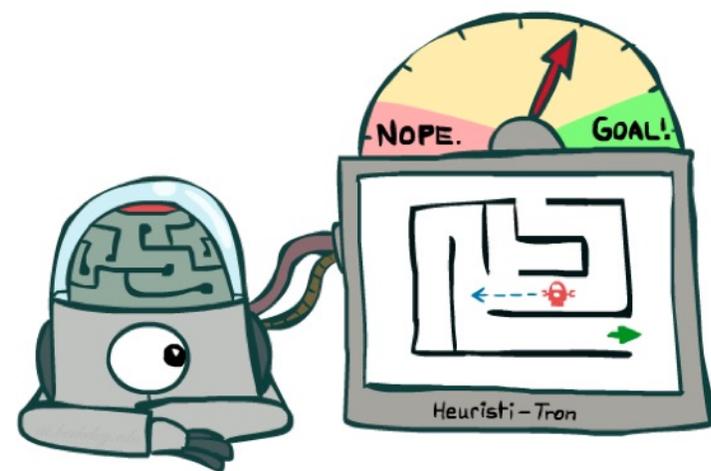
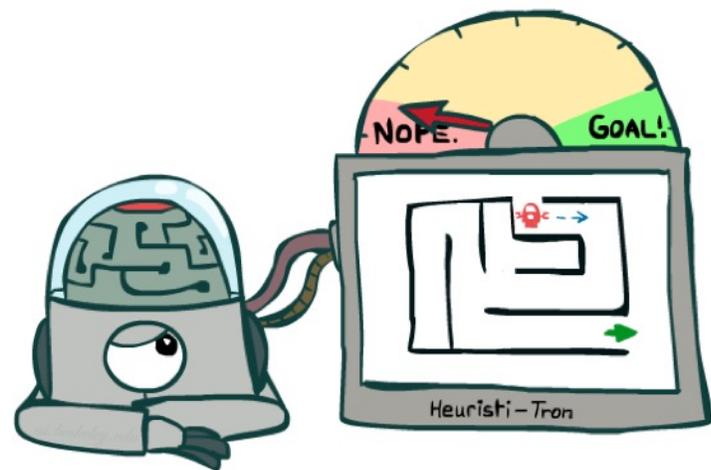
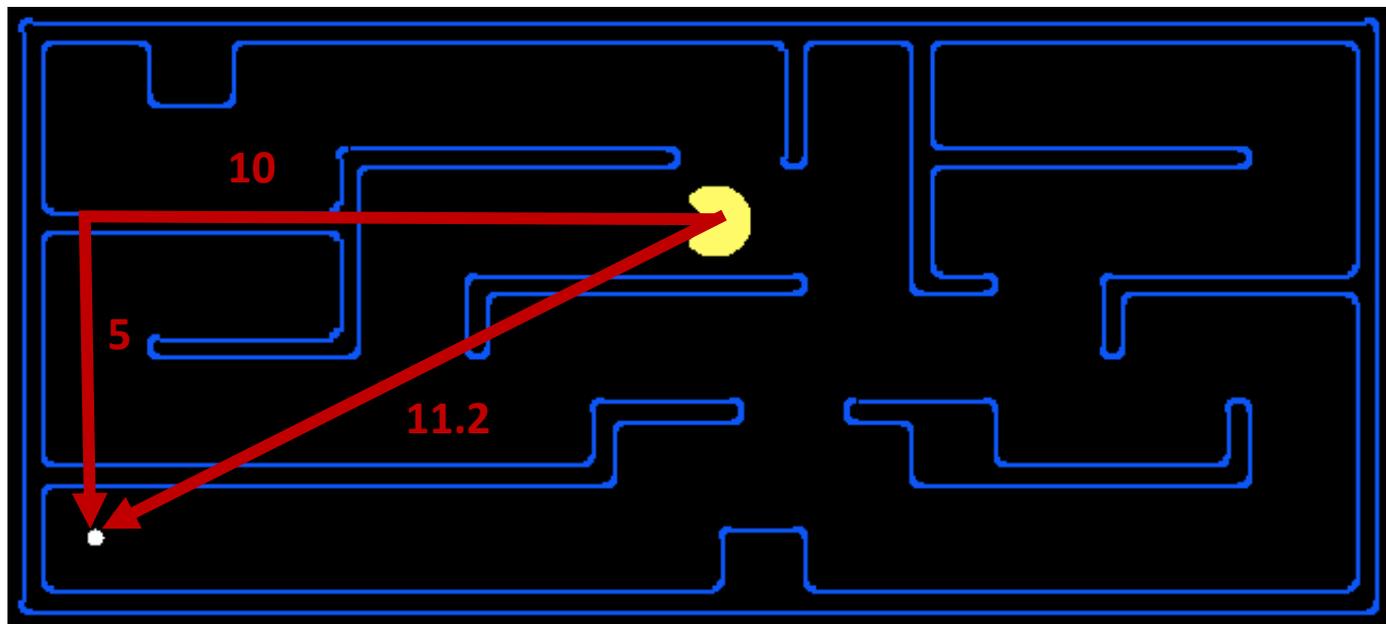
□ 本章小结



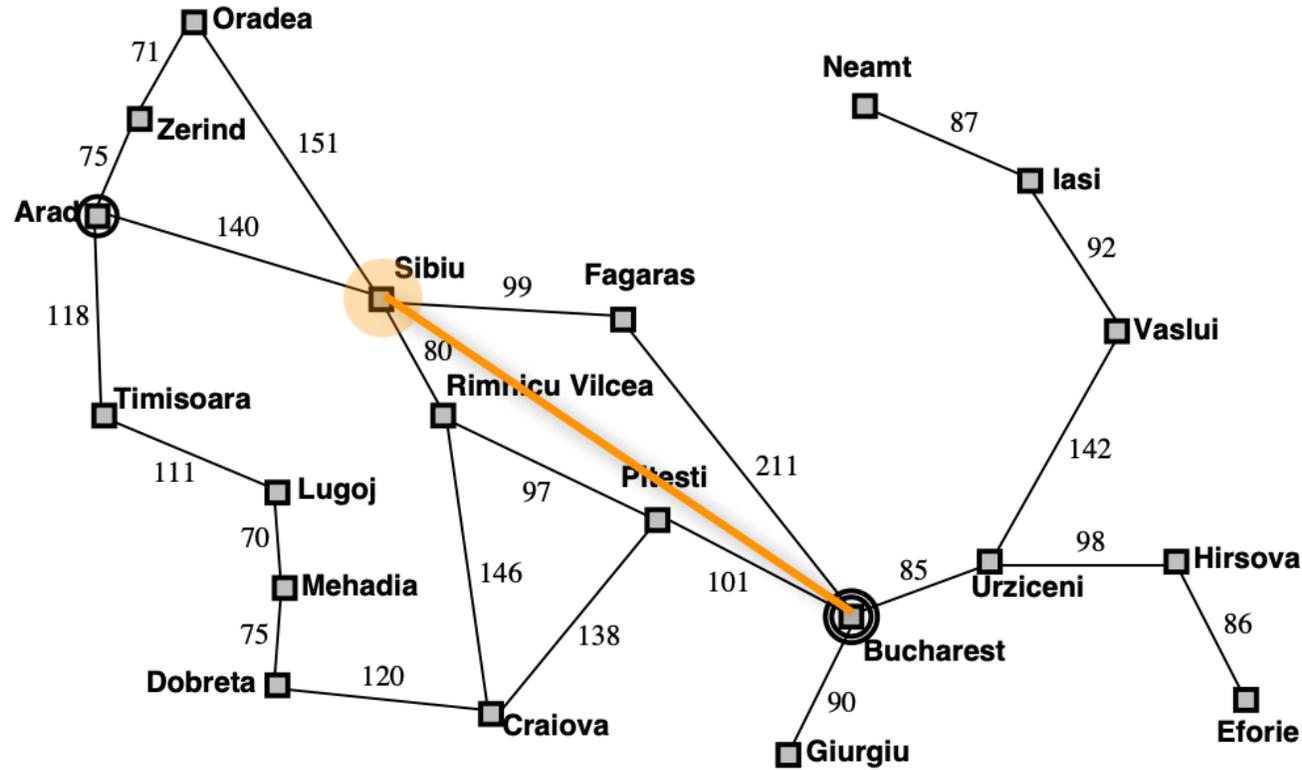
启发式信息 (Heuristic Information)

□ 启发式信息:

- ✓ 估计离目标状态有多远的信息
- ✓ 不同的问题需要设计不同的启发式信息
- ✓ 例如: 曼哈顿距离、欧式距离



启发式信息 (Heuristic Information)



Straight-line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
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

$h(x)$

提纲

□ 启发式搜索

➤ 贪婪搜索

➤ A*搜索

➤ 启发式函数

□ 本章小结



贪婪搜索 (greedy search)



Extra: 贪婪算法 (greedy algorithm)

贪婪算法，又称贪心算法，是一种在每一步选择中都采取在**当前状态下最好或最优（即最有利）的选择**，从而希望导致结果是最好或最优的算法

示例

假设你是一个售货员，钱柜里的货币只有 25 分、20分、5分和1分四种硬币，如果你要找给客户41分钱的硬币，如何安排才能找给客人的钱**既正确且硬币的个数又最少**？

$$✓ 41-25=16$$

$$✓ 16-5*3=1$$

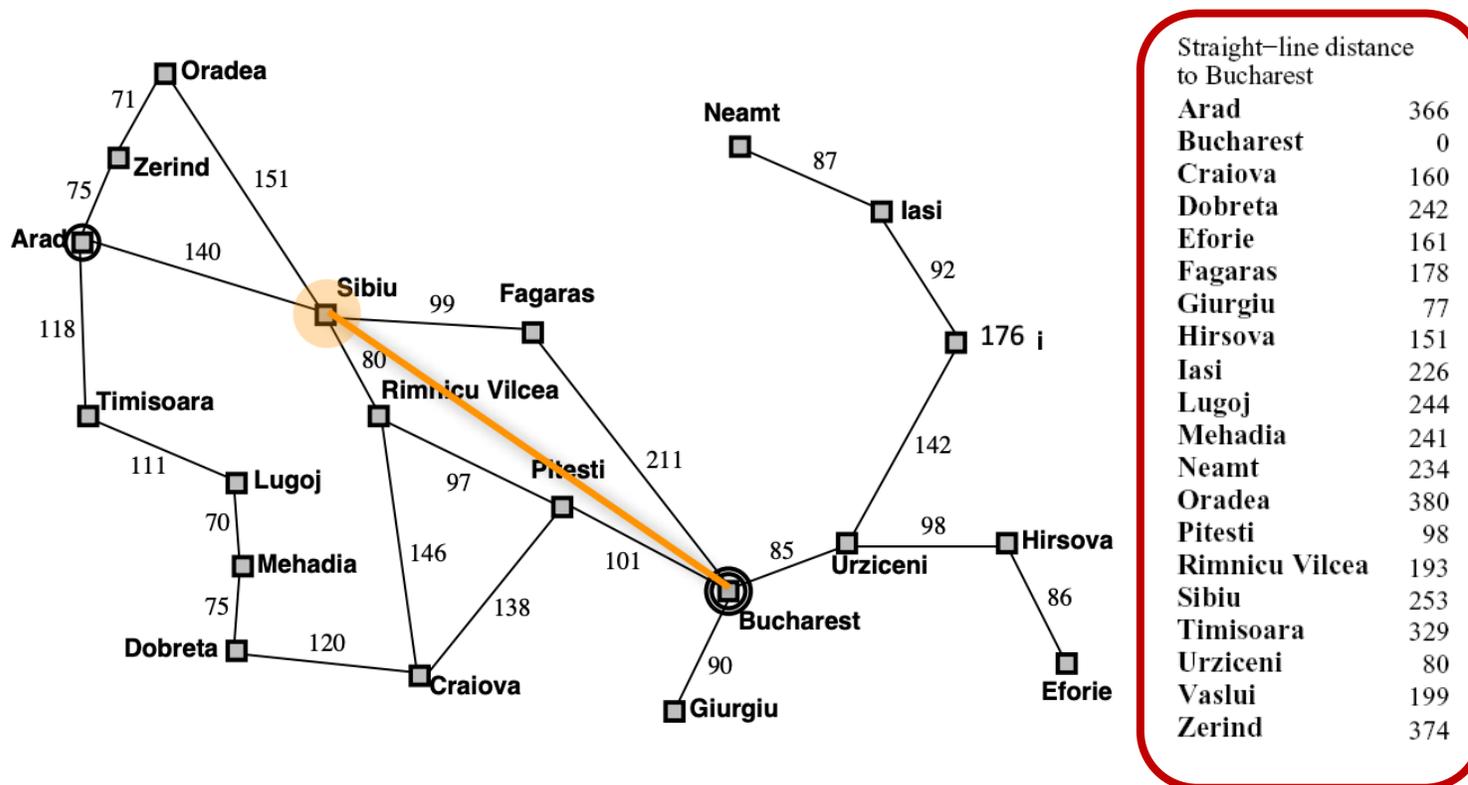
$$✓ 1-1=0$$

1个25分，3个5分，1个1分

没有从总体上考虑其它可能情况，每次选取局部最优解，一般不能得到最优解

贪婪搜索 (greedy search)

如何使用贪心的思想求解？

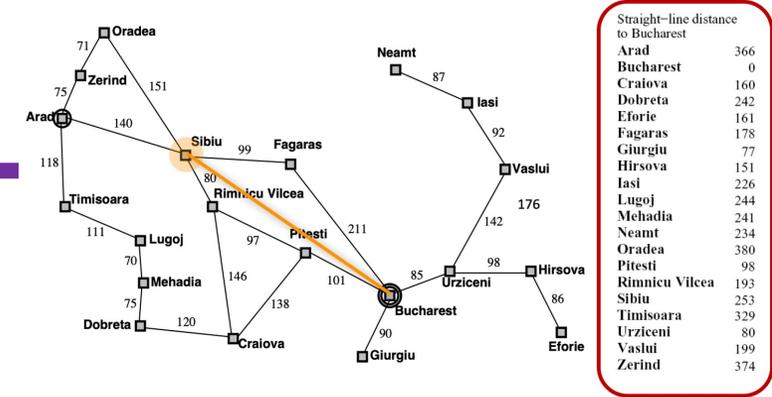


$h(x)$

每次均选择离目标状态最近的节点

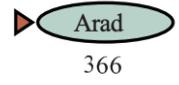
贪婪搜索 (greedy search)

(a) The initial state

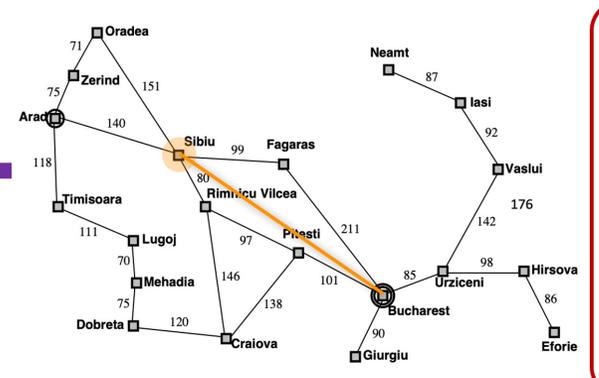
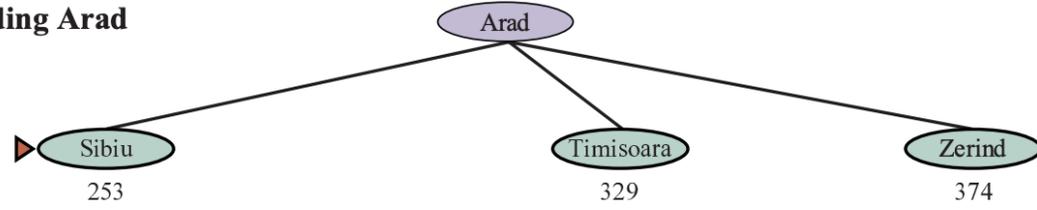


贪婪搜索 (greedy search)

(a) The initial state

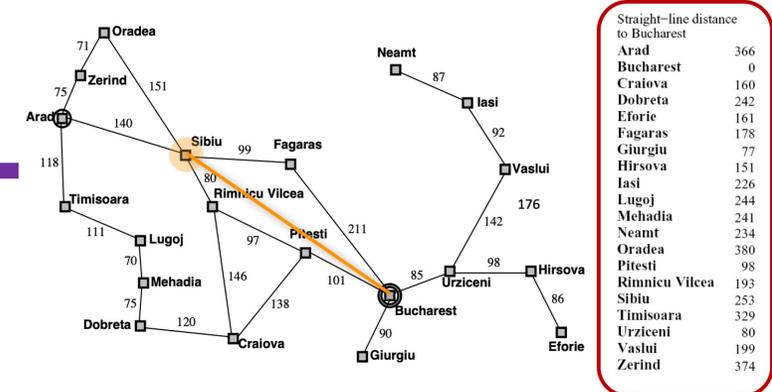


(b) After expanding Arad



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

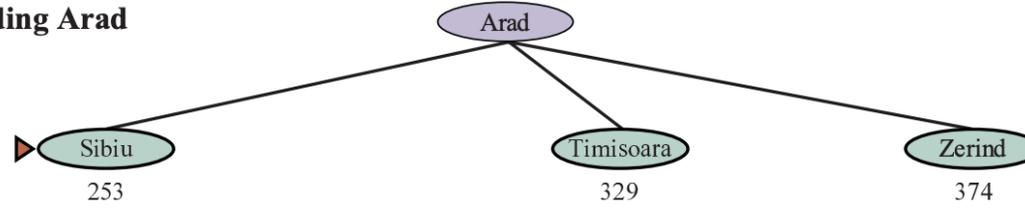
贪婪搜索 (greedy search)



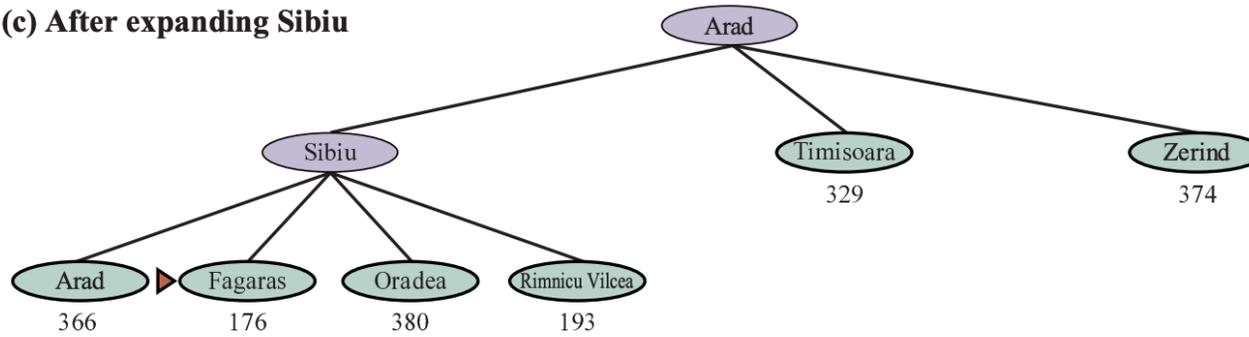
(a) The initial state



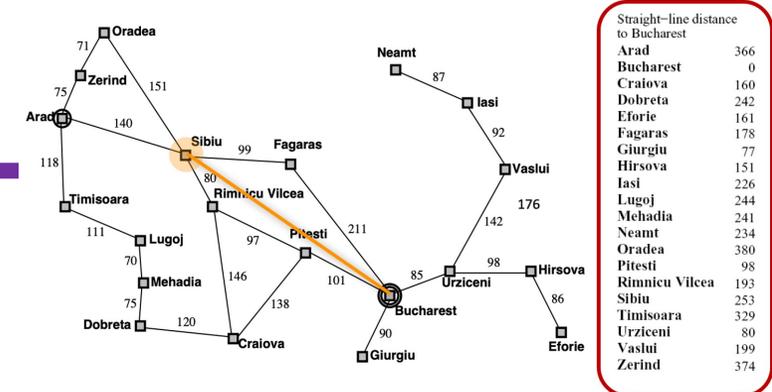
(b) After expanding Arad



(c) After expanding Sibiu



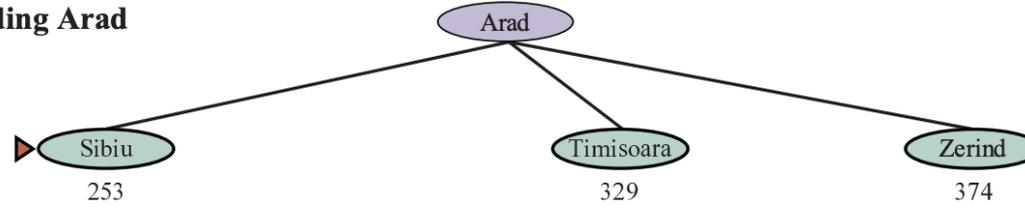
贪婪搜索 (greedy search)



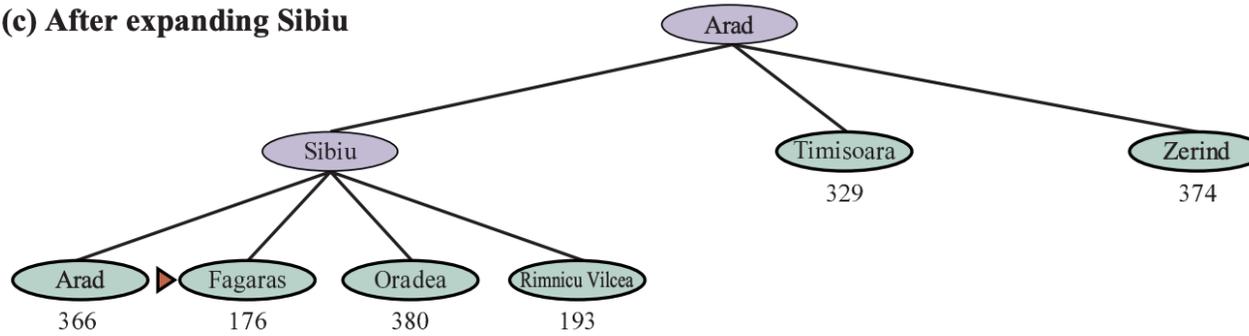
(a) The initial state



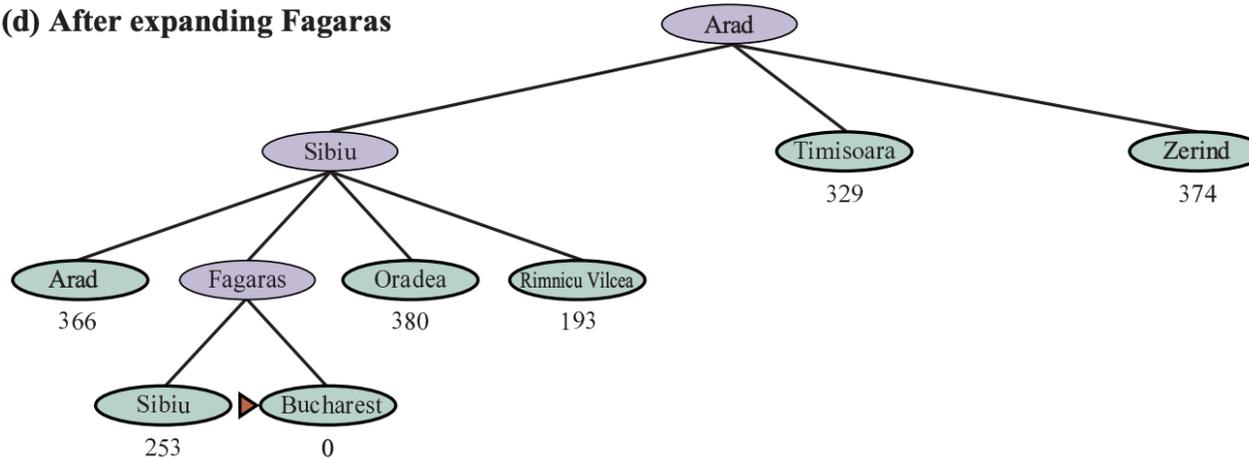
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



贪婪搜索 (greedy search)

□ 策略：每一步均扩展距离目标状态更近节点

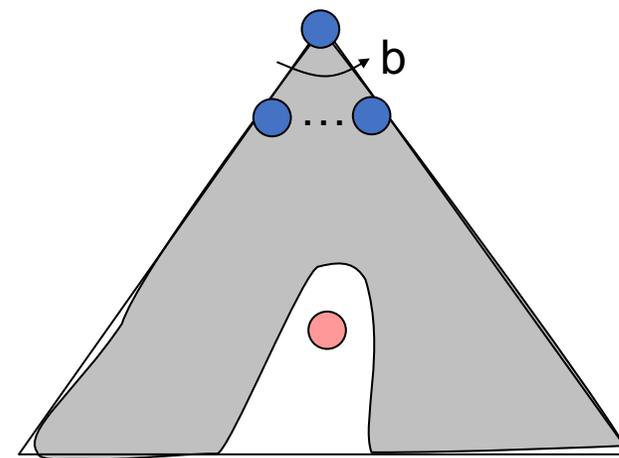
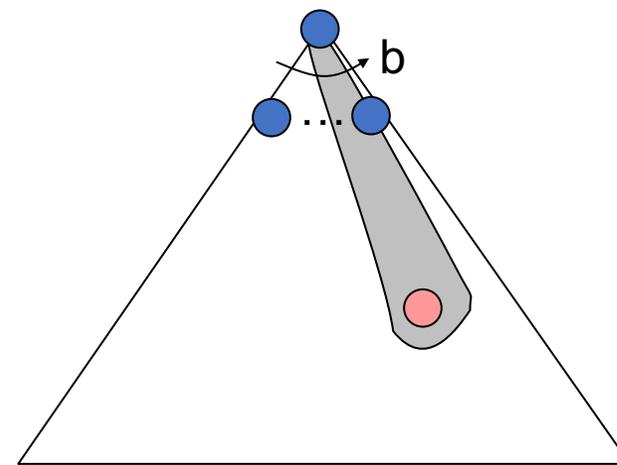
✓ 启发式信息：估计当前状态距离目标状态的距离

□ Common case:

✓ 距离目标越来越近

□ Bad case:

✓ 非完备、非最优、依赖启发式函数



提纲

□ 启发式搜索

➤ 贪婪搜索

➤ A*搜索

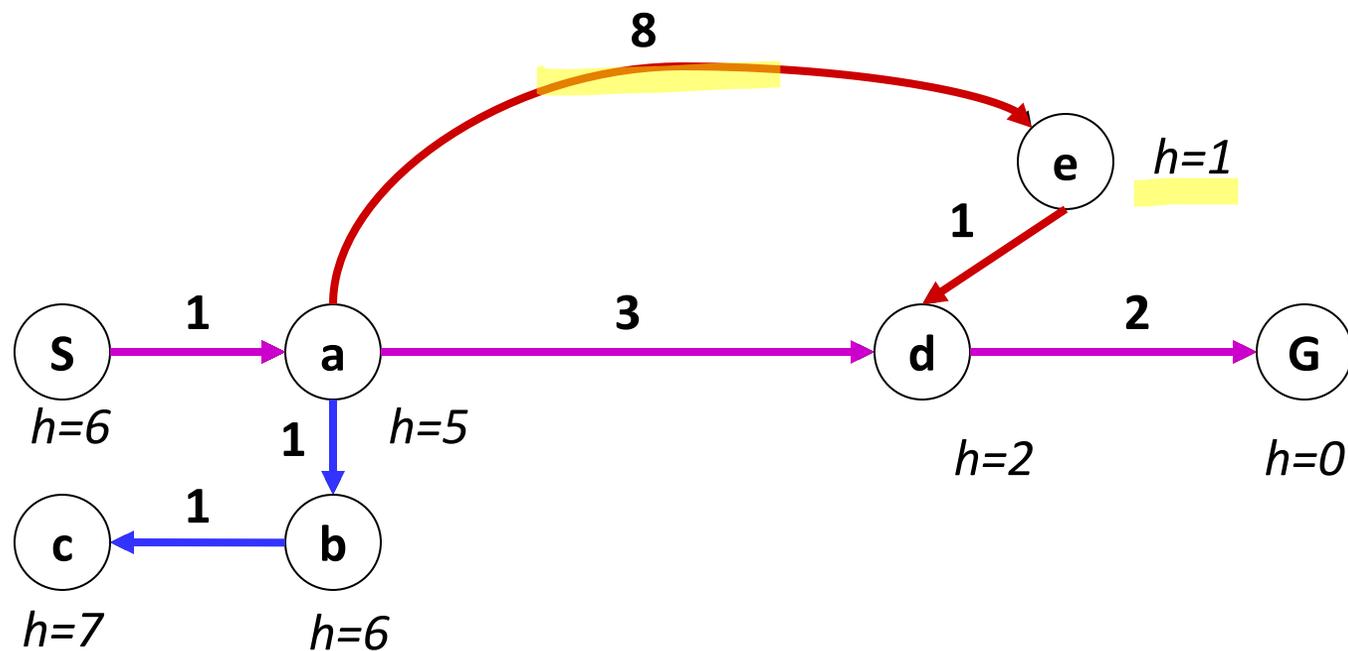
➤ 启发式函数

□ 本章小结



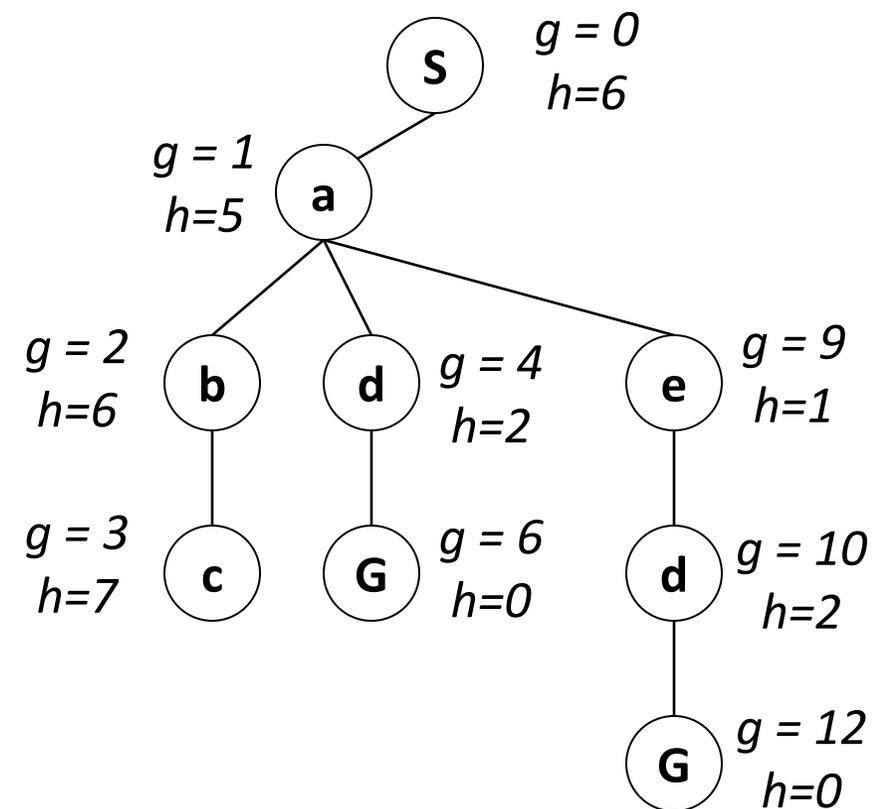
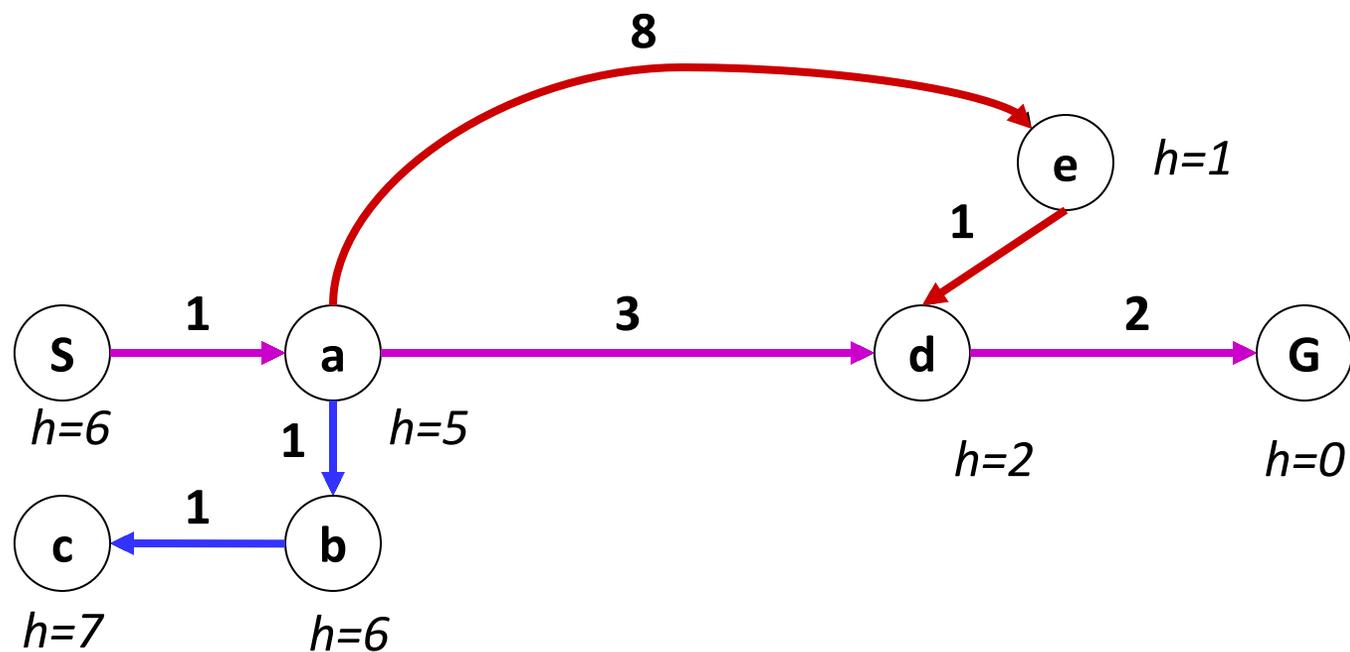
A*算法

- 一致性代价搜索：到当前节点路径的代价消耗
- 贪婪搜索：节点到目标状态的估计



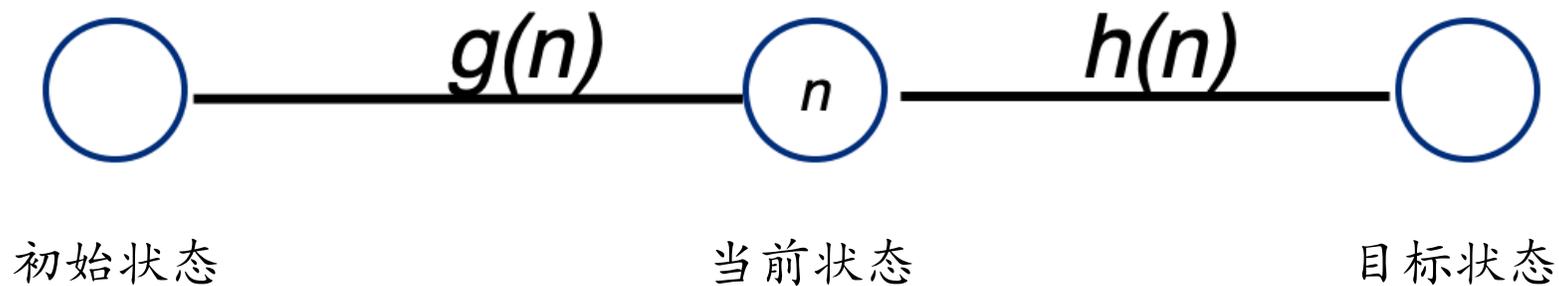
A*算法

- 一致性代价搜索：到当前节点路径的代价消耗
- 贪婪搜索：节点到目标状态的估计



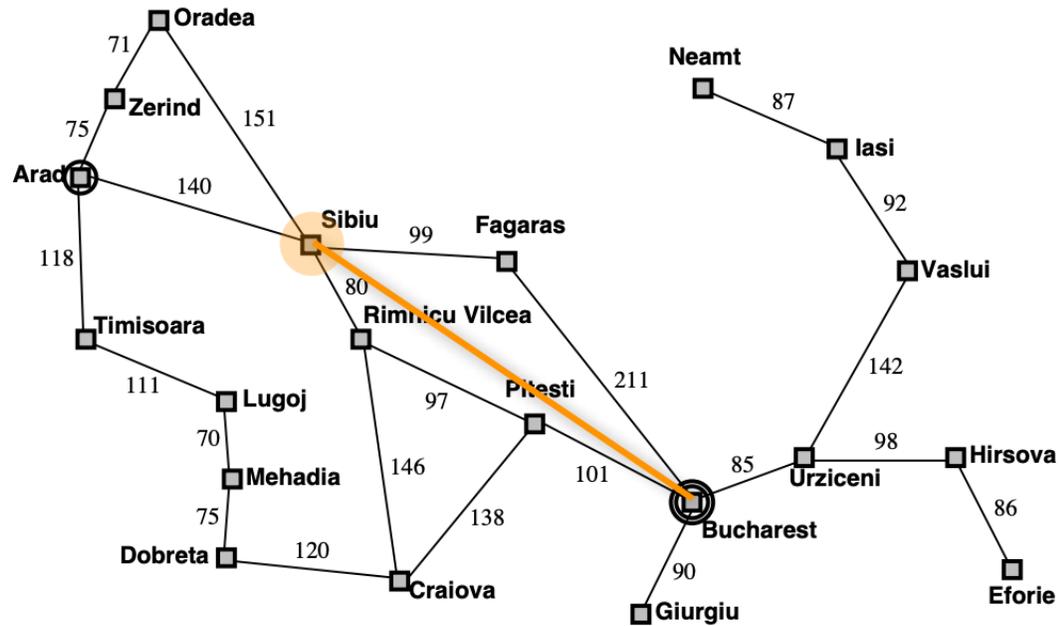
A*算法

$$\text{A*算法: } f(n) = g(n) + h(n)$$



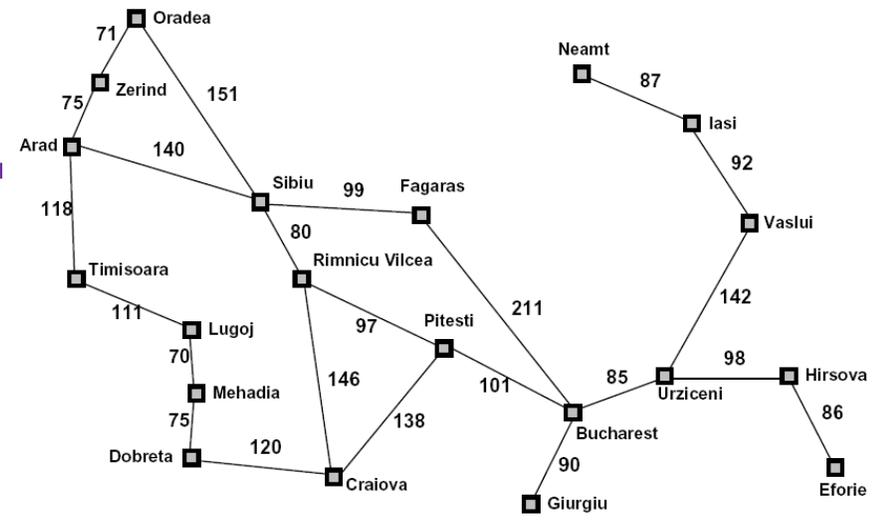
- $g(n)$ 表示从起始结点到结点 n 的开销代价值
- $h(n)$ 表示从结点 n 到目标结点路径中所估算的最小开销代价值

A*算法



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

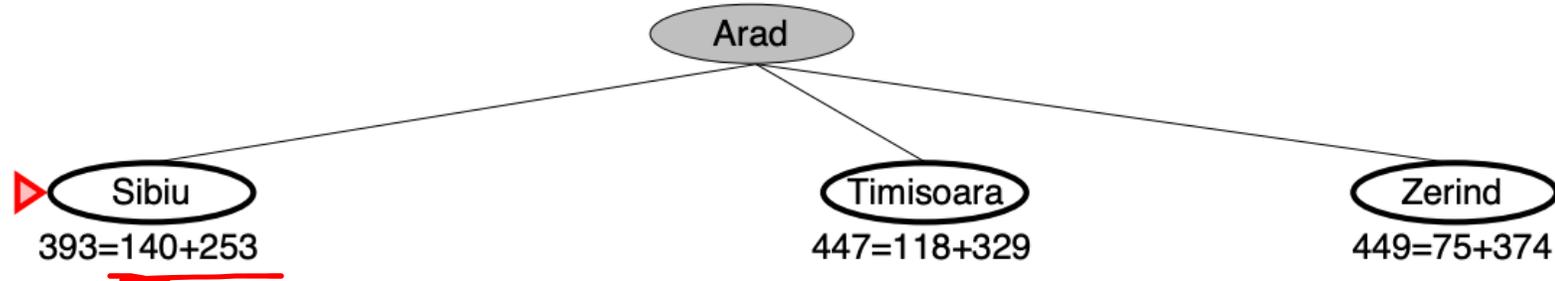
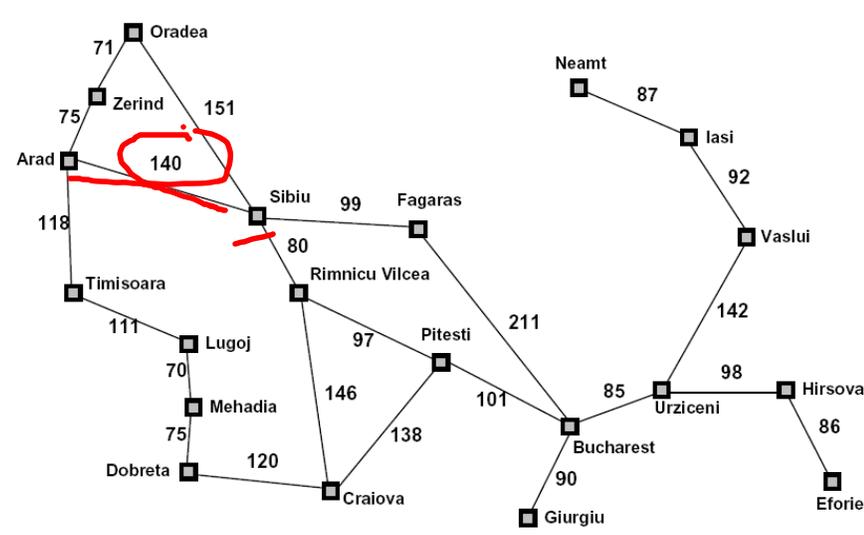
A*算法



	Straight-line distance to Bucharest
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
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

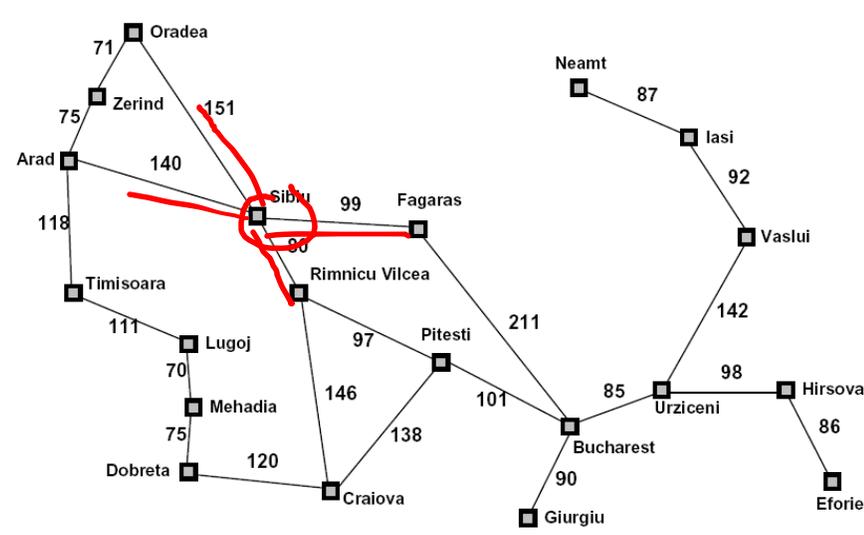
▶ Arad
 $366 = 0 + 366$

A*算法

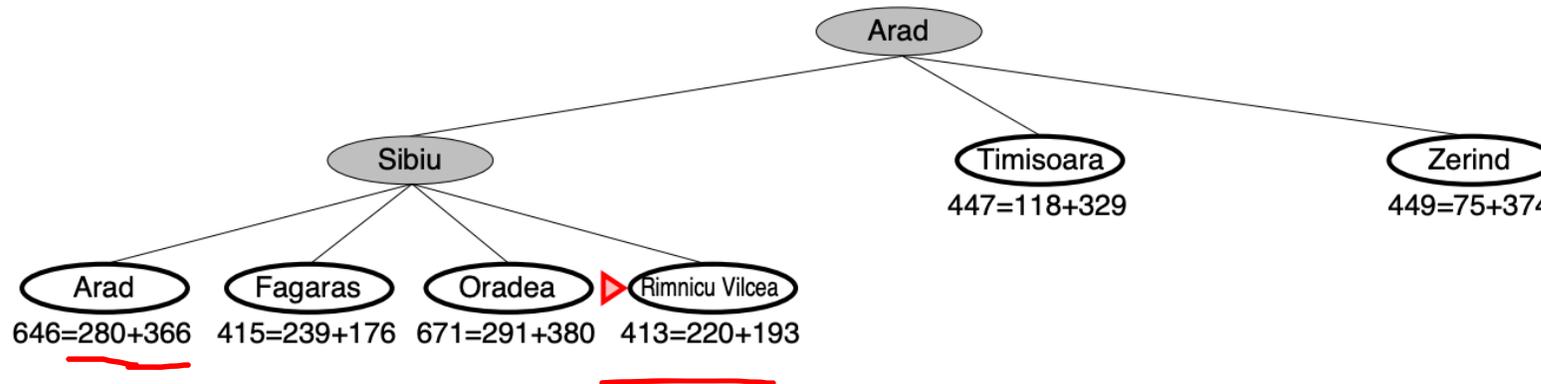


A*算法

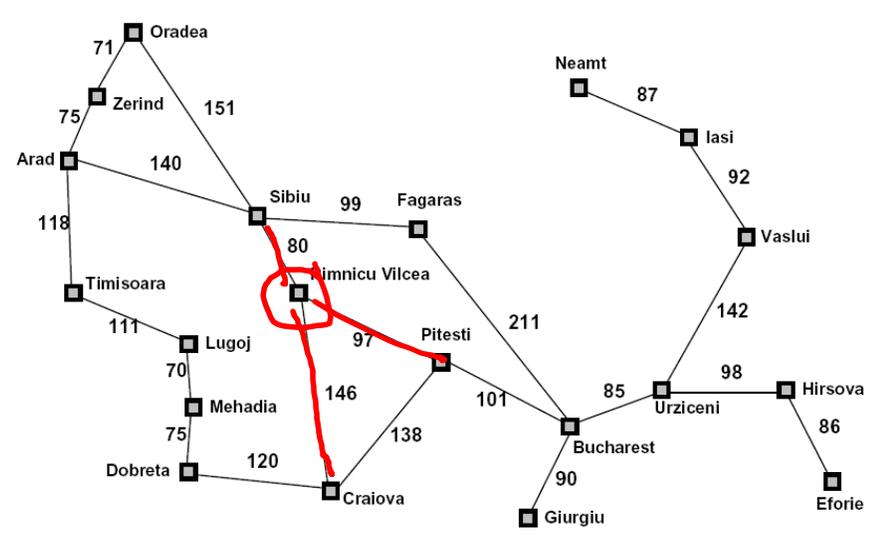
$$f = g + h$$



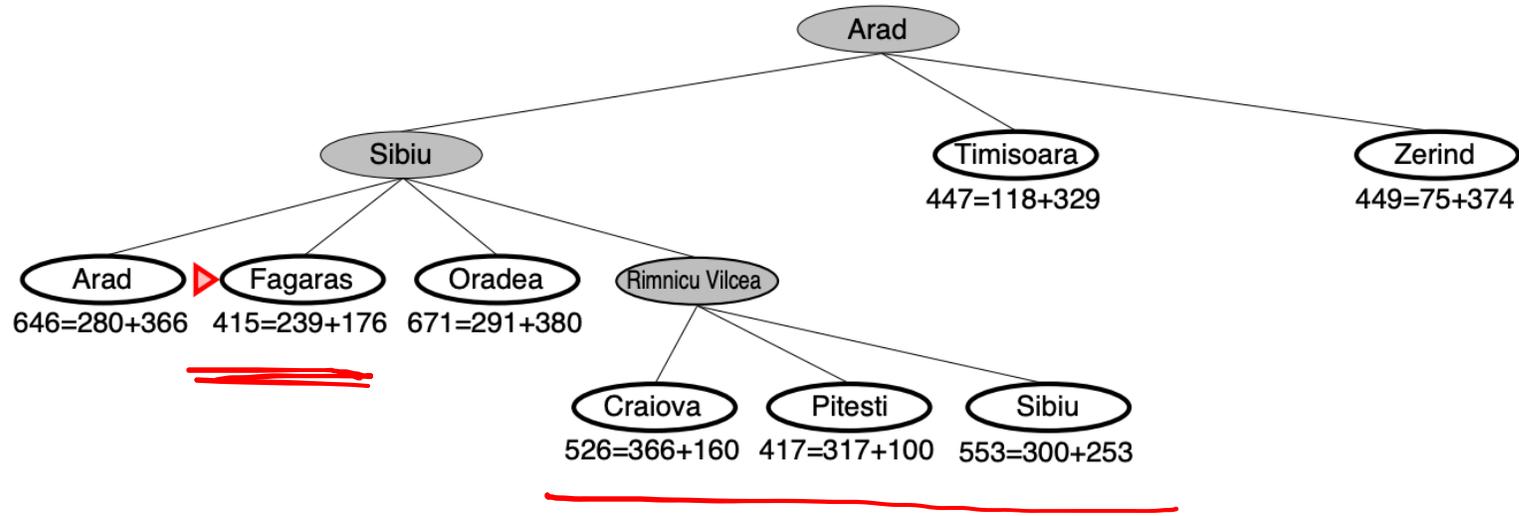
City	Straight-line distance to Bucharest
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



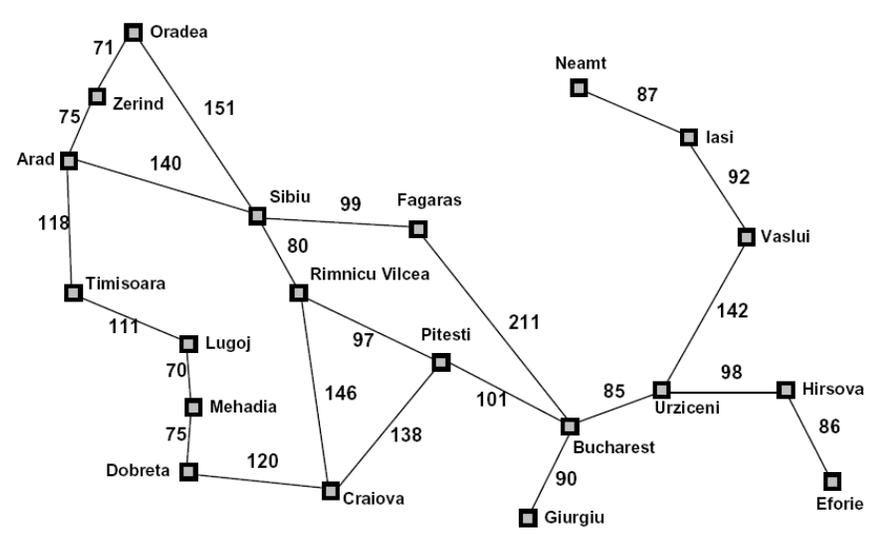
A*算法



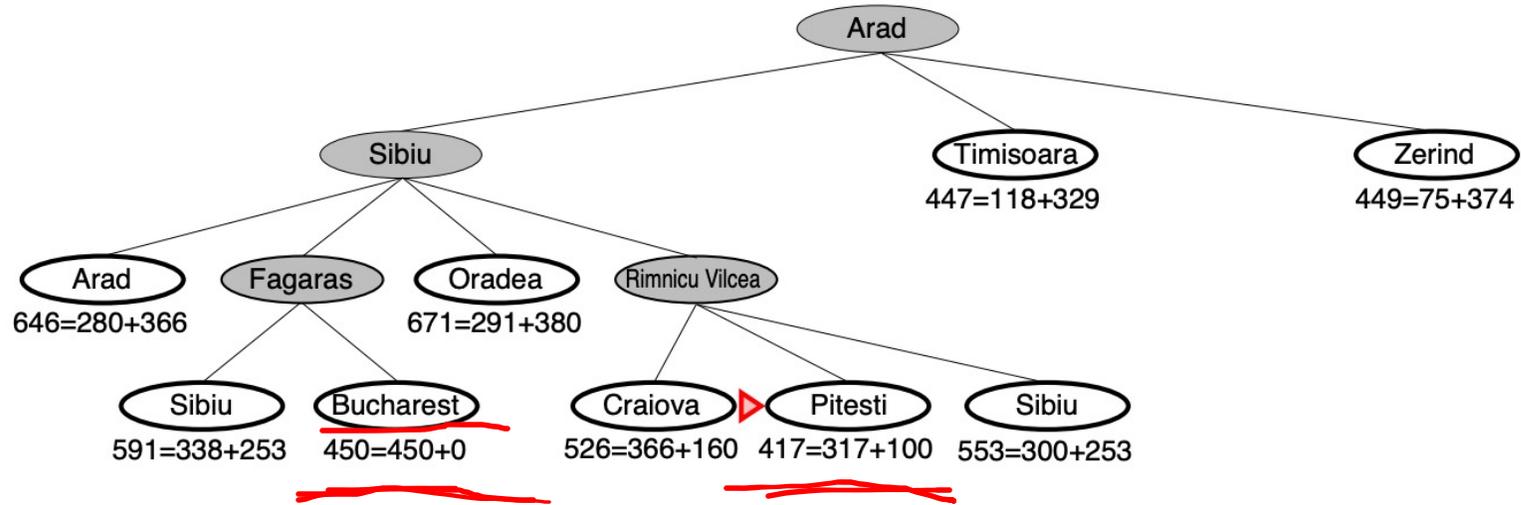
City	Straight-line distance to Bucharest
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



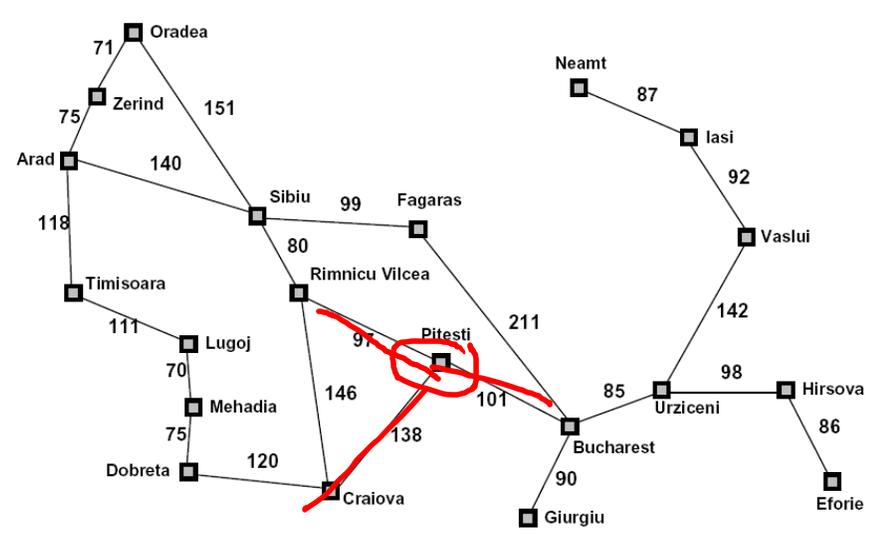
A*算法



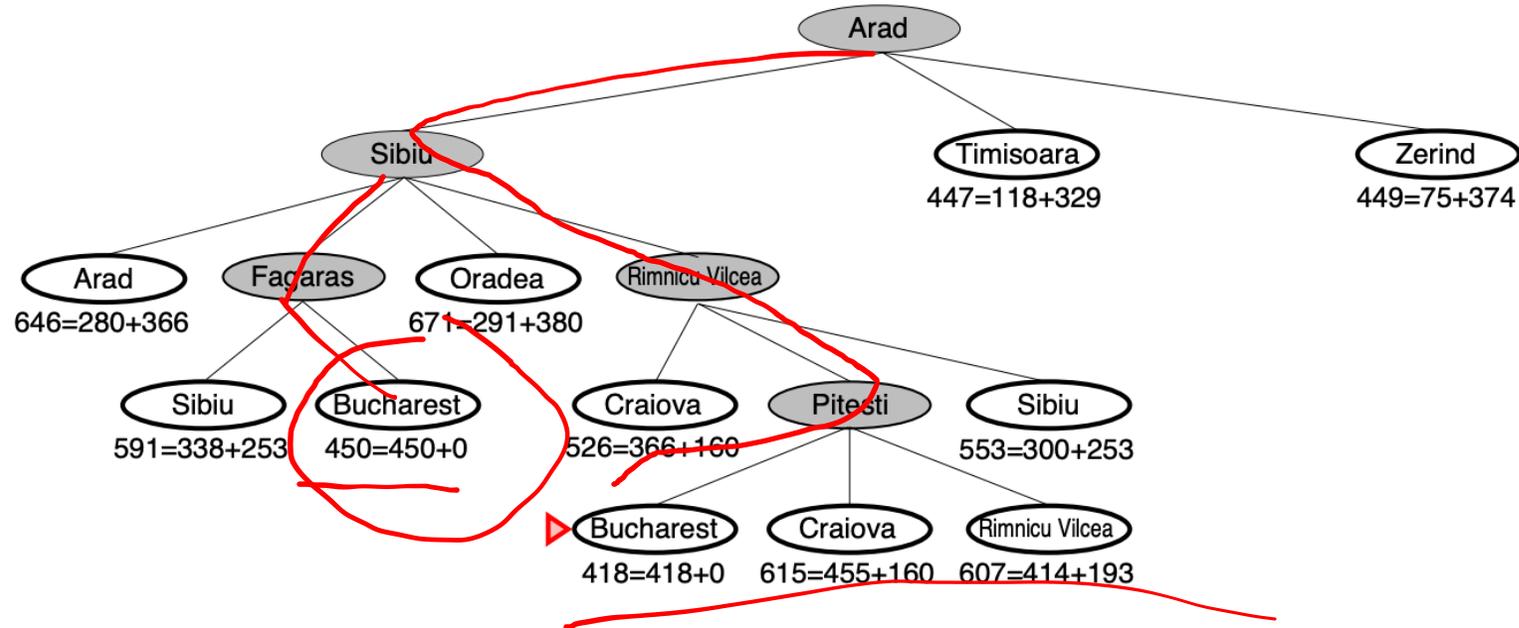
City	Straight-line distance to Bucharest
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



A*算法



City	Straight-line distance to Bucharest
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
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



A*算法的完备性

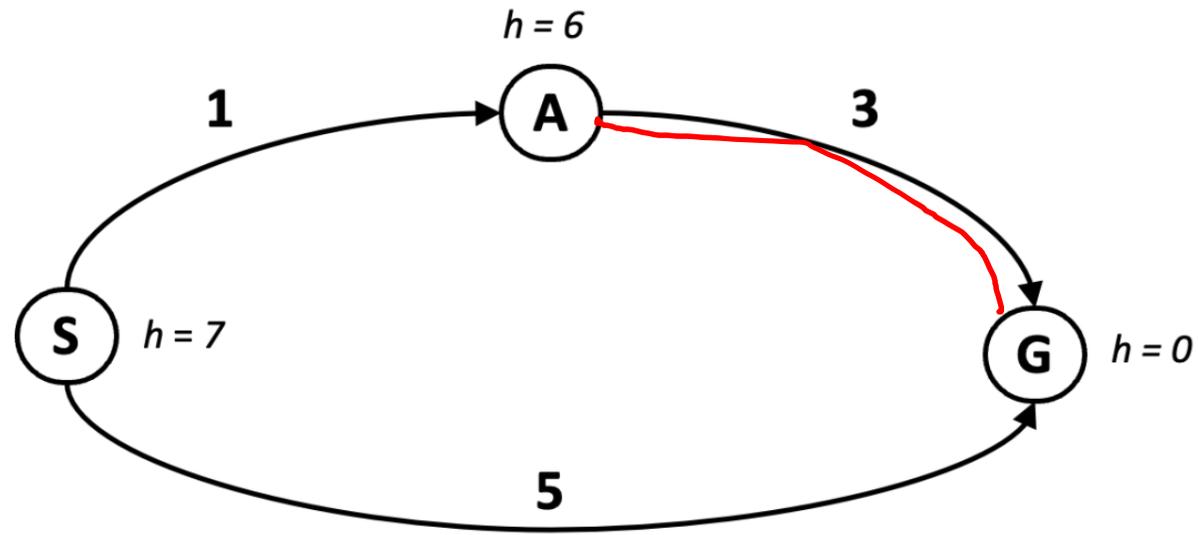
在一些常见的搜索问题中，状态数量是有限的，此时图搜索A*算法和排除环路的树搜索A*算法均是完备的，即一定能够找到一个解。

在更普遍的情况下，如果所求解问题和启发函数满足以下条件，则A*算法是完备的：

- ✓ 搜索树中分支数量是有限的，即每个节点的后继结点数量是有限的
- ✓ 单步代价的下界是一个正数
- ✓ 启发函数有下界

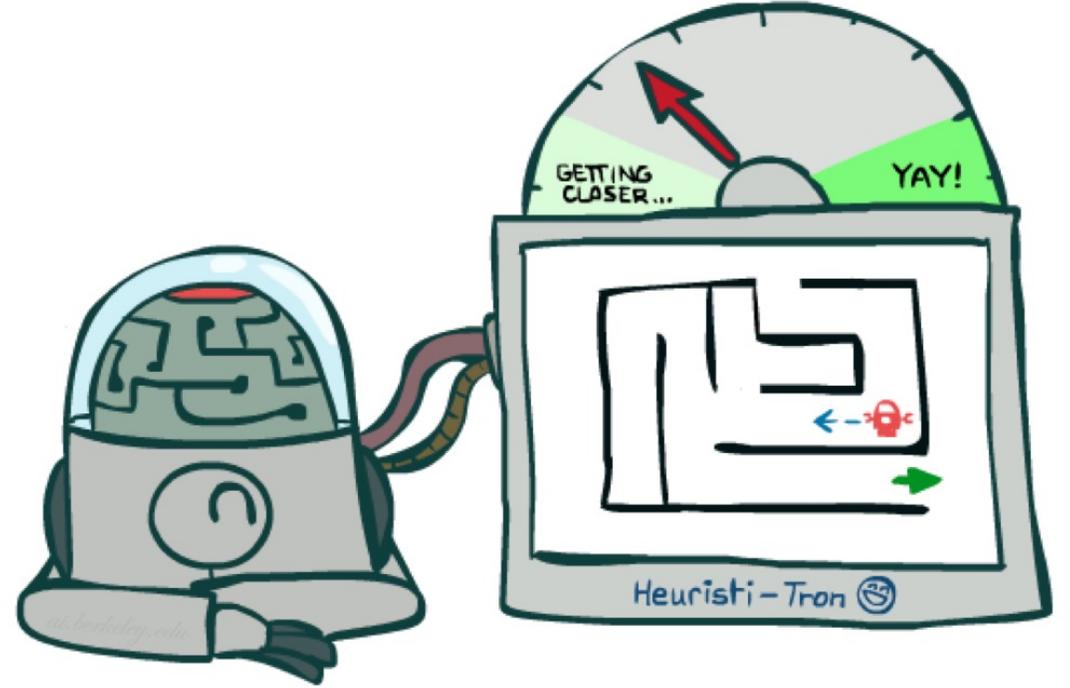
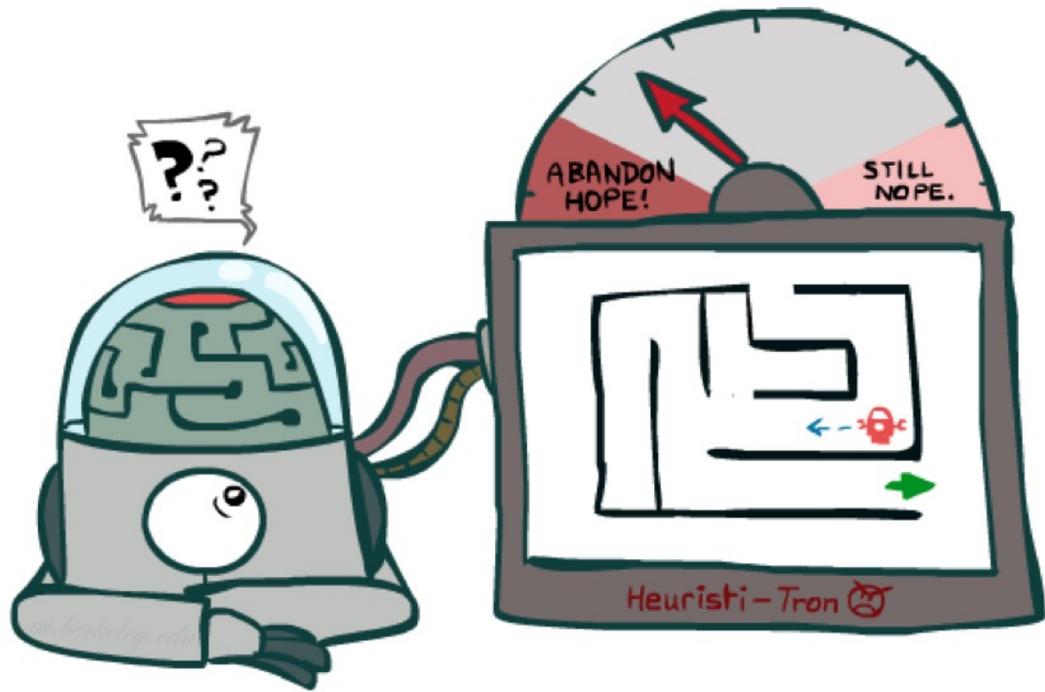
A*算法的最优性

A*算法能否找到最佳路线？

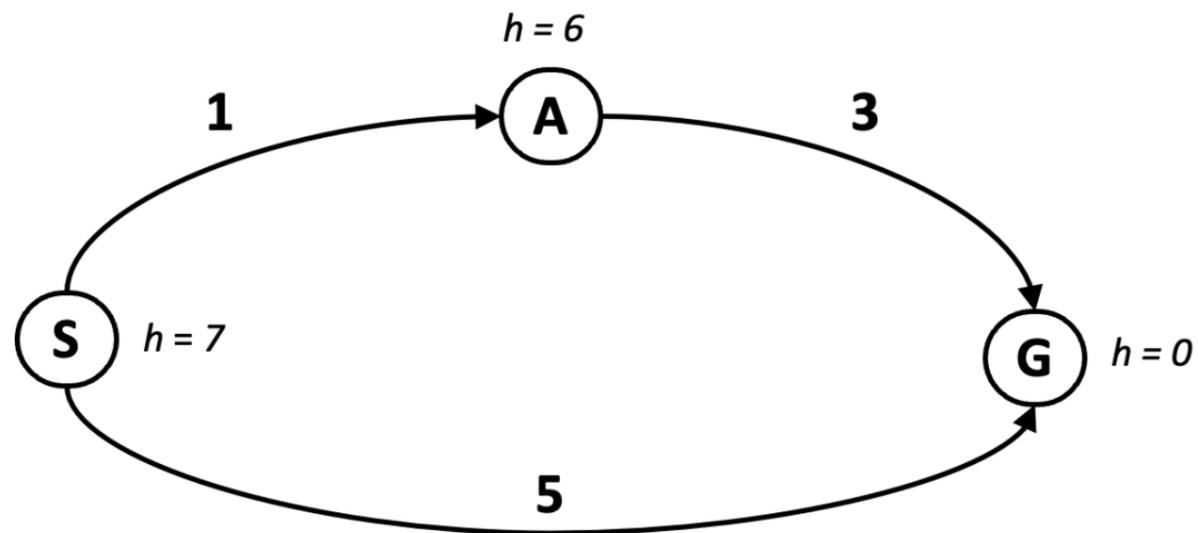


Quiz: 如何修改启发式信息可以使算法找到最优解？

A*算法的最优性



A*算法的最优性

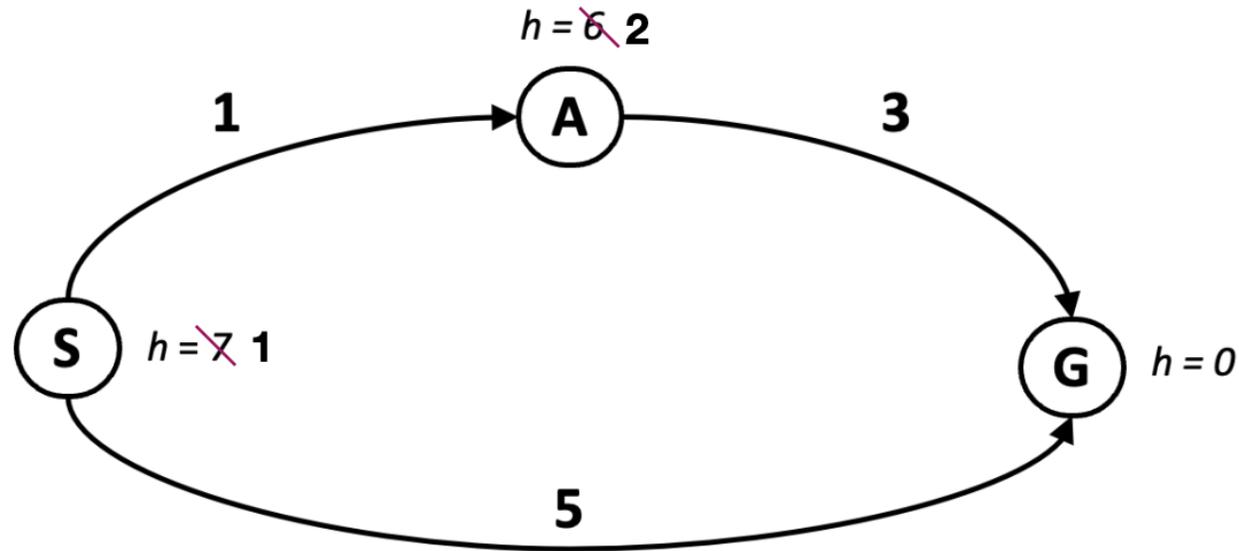


显然，为了保障A*算法的最优性，需要额外假设
不应该过高估计 $h(n)$ 从而导致好的节点被忽略

保障最优性的条件：可采纳性

可采纳性 (admissible)：对于任意结点 n ，有 $h(n) \leq h^*(n)$

启发函数不会过高估计 (over-estimate) 从结点 n 到终止结点所应该付出的代价 (即估计代价小于等于实际代价)



保障最优性的条件：可采纳性

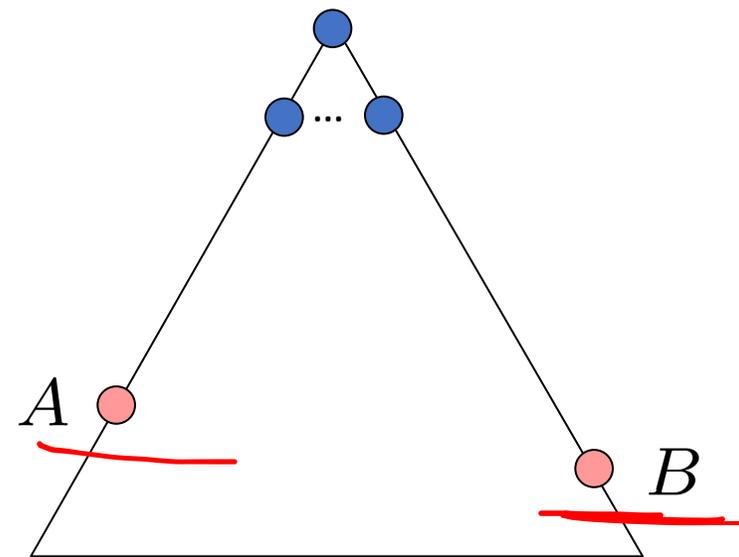
定理：

如果启发函数是可采纳的，那么树搜索的A*算法满足最优性。

保障最优性的条件：可采纳性

假设：

- A是最优的目标节点
- B是次优的目标节点
- 启发函数是可采纳的



保障最优性的条件：可采纳性

证明：

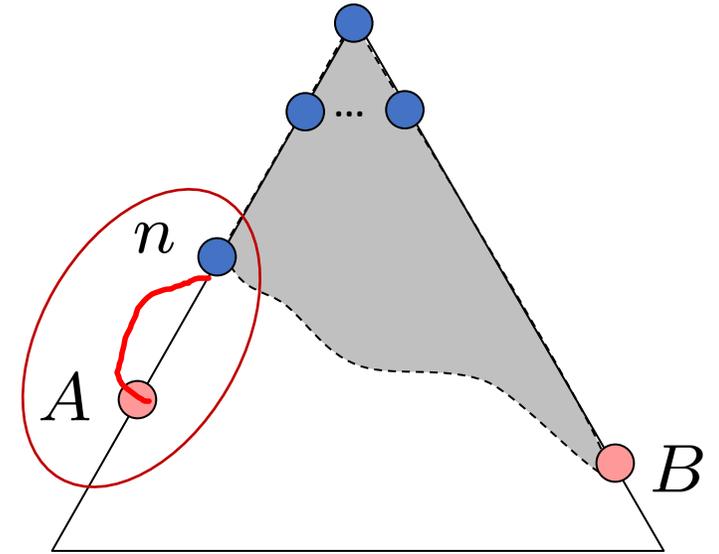
- 假设 B 位于边缘集
- A 的某个父节点 n 也在边缘集
- 那么, n 比 B 更早被扩展

1. $f(n) \leq f(A)$

$$f(n) = g(n) + \underline{h(n)}$$

$$\underline{f(n)} \leq \underline{g(A)}$$

$$\underline{g(A)} = \underline{f(A)}$$



保障最优性的条件：可采纳性

证明：

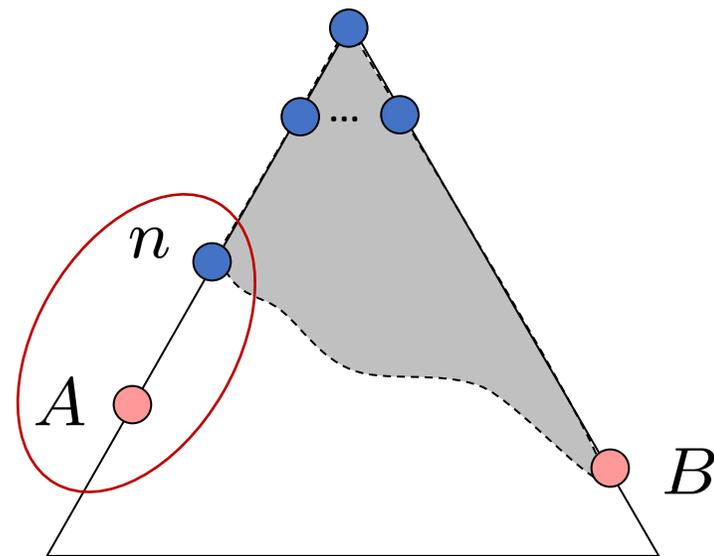
- 假设 B 位于边缘集
- A 的某个父节点 n 也在边缘集
- 那么, n 比 B 更早被扩展

1. $f(n) \leq f(A)$

2. $f(A) \leq f(B)$

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

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



保障最优性的条件：可采纳性

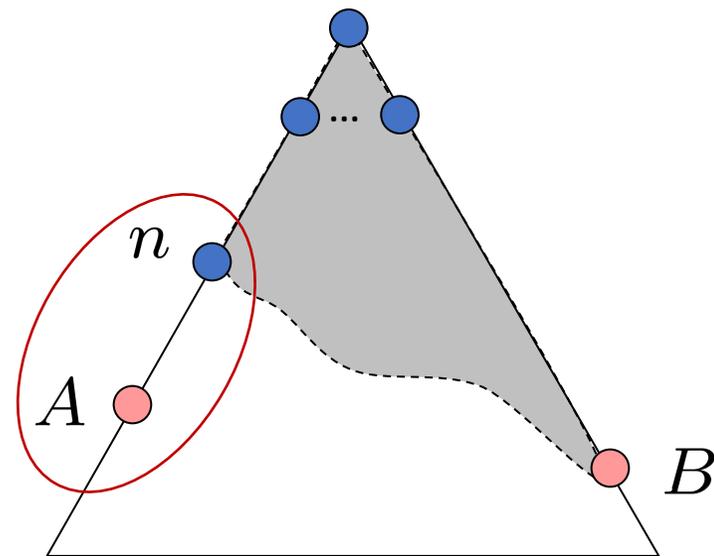
证明：

- 假设 B 位于边缘集
- A 的某个父节点 n 也在边缘集
- 那么, n 比 B 更早被扩展

1. $f(n) \leq f(A)$

2. $f(A) \leq f(B)$

3. $f(n) \leq f(A) \leq f(B)$



保障最优性的条件：可采纳性

证明：

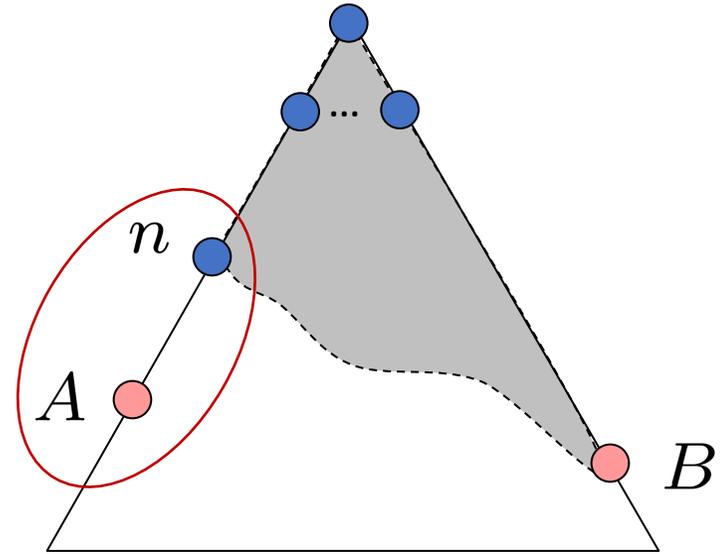
- 假设 B 位于边缘集
- A 的某个父节点 n 也在边缘集
- 那么, n 比 B 更早被扩展

1. $f(n) \leq f(A)$

2. $f(A) \leq f(B)$

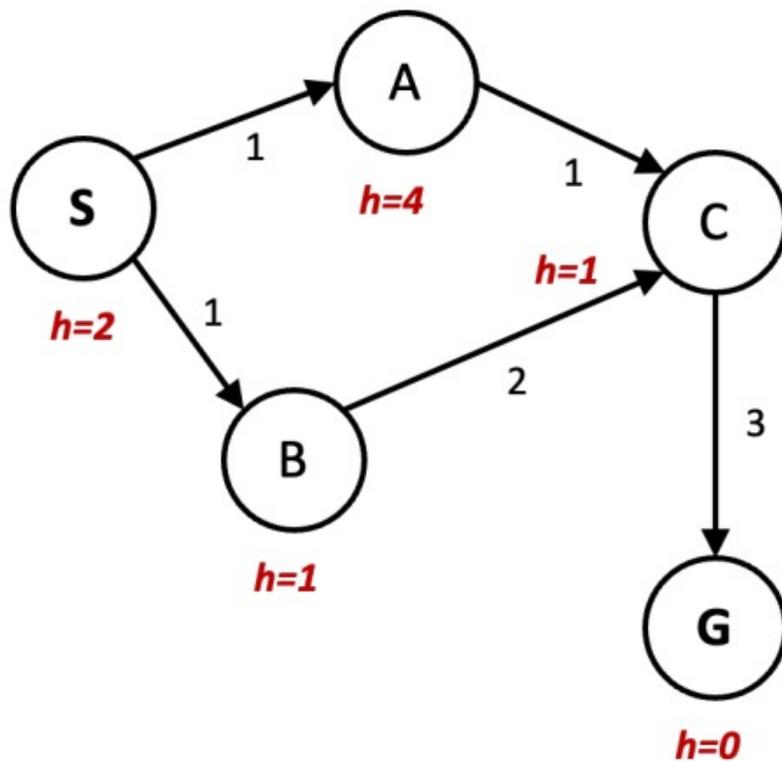
3. $f(n) \leq f(A) \leq f(B)$

- 所有节点 n 都比 B 更早被扩展，所以 A 比 B 更早被扩展



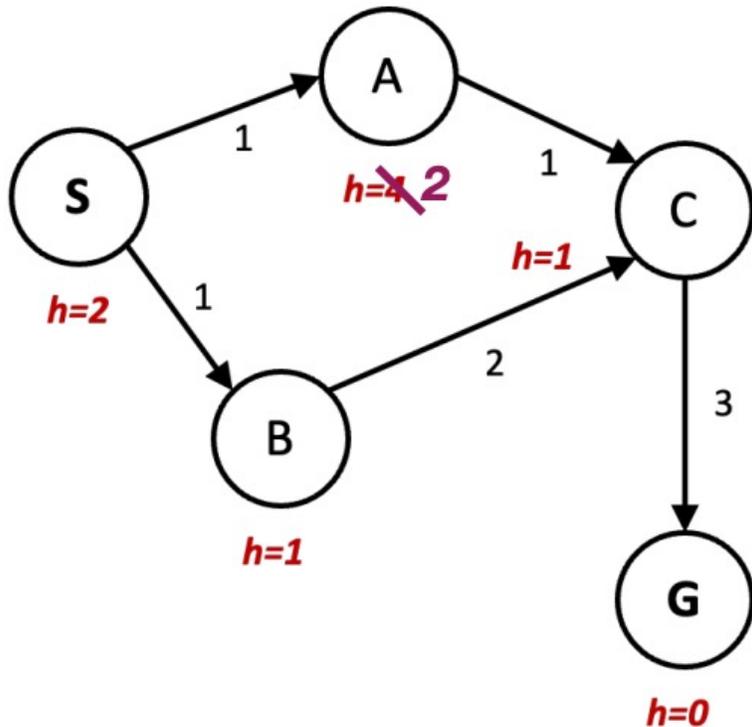
图搜索的A*算法

图搜索的A*算法能否找到最优解？



保障最优性的条件：一致性

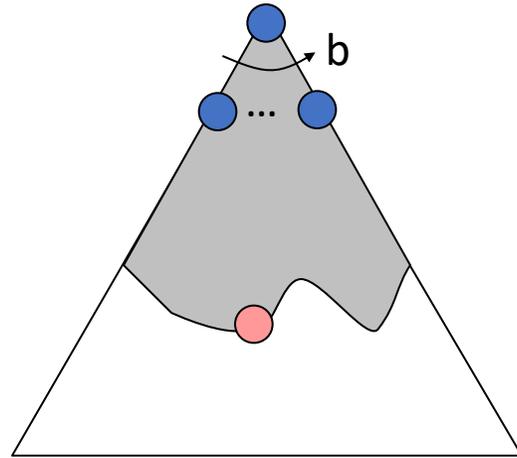
一致性条件： 对于每个结点 n 和通过任一行动 a 生成的 n 的每个后继节点 n' ，
有 $h(n) \leq \text{cost}(n, a, n') + h(n')$ 。



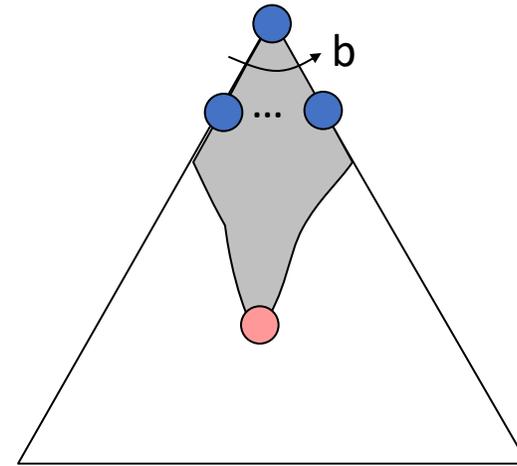
显然，由一致性条件可以推导出可采纳性

UCS VS A*

一致性代价搜索

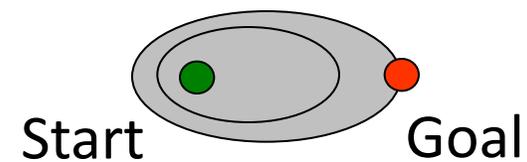
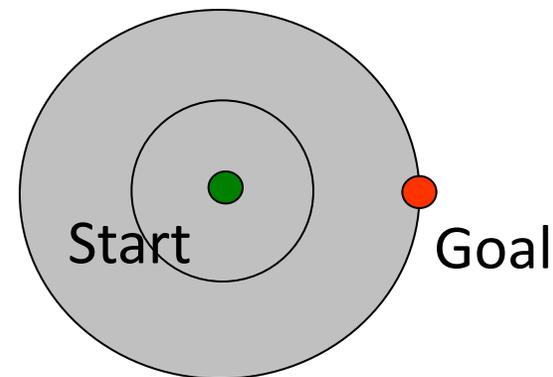


A*搜索



UCS VS A*

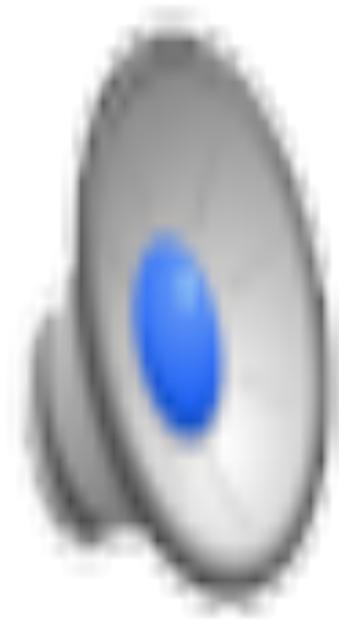
- UCS 在所有的方向上探索
- A*利用关于目标的信息，但是依赖信息的准确性



Video of Demo Contours -- UCS



Video of Demo Contours -- Greedy



Video of Demo Contours – A^*



Video of Demo Contours (PacMan)



Video of Demo Contours (PacMan)

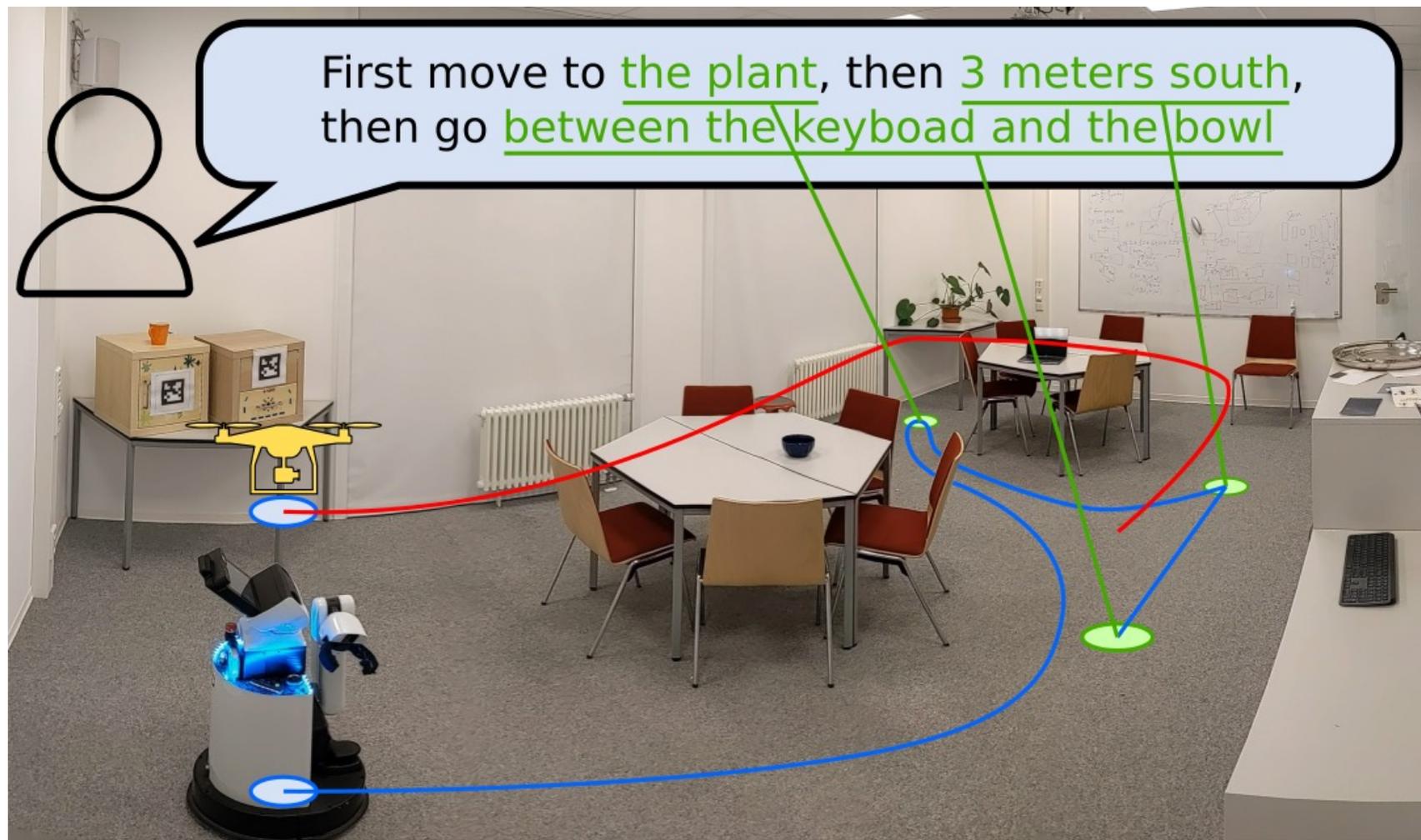


Video of Demo Contours (PacMan)



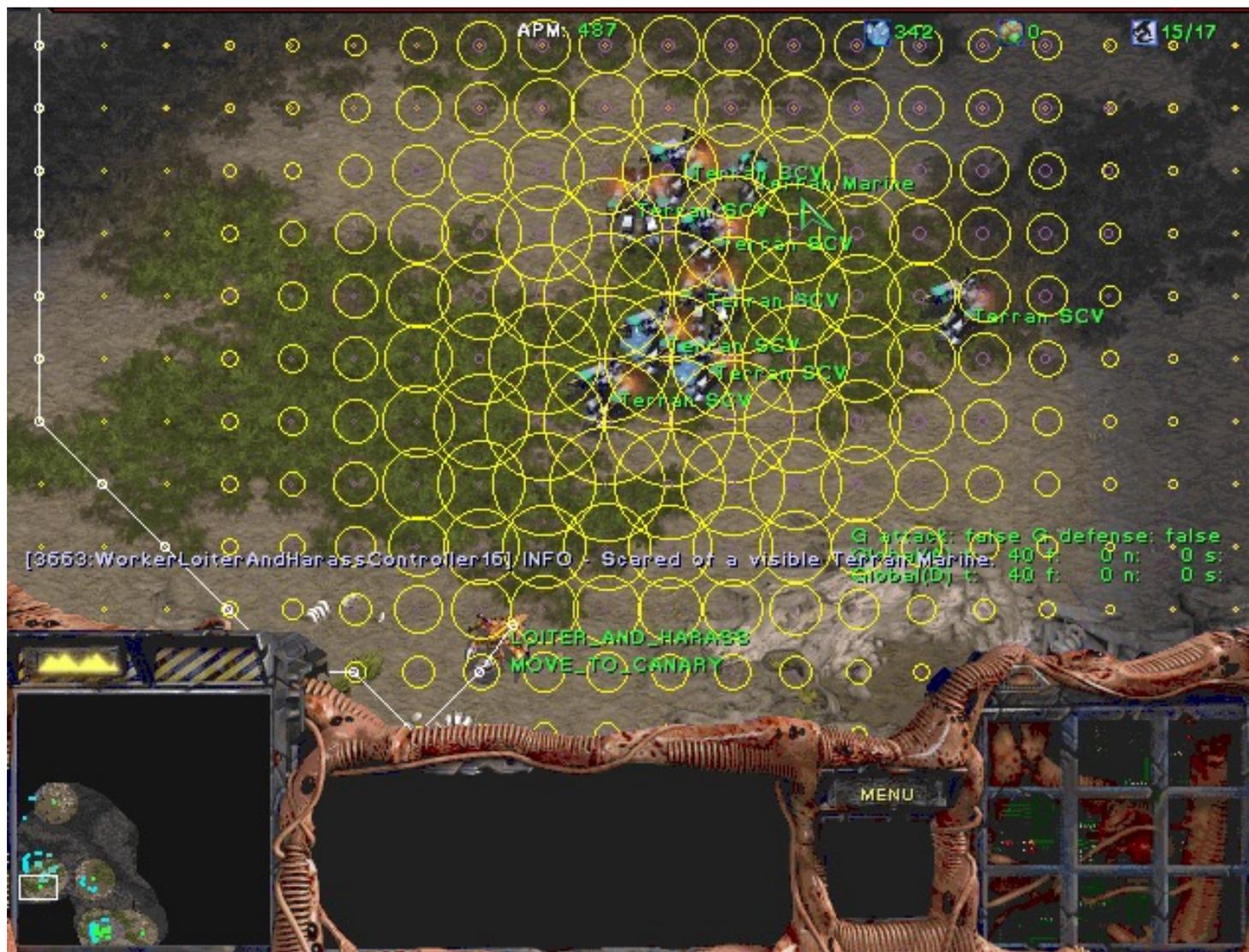
A*算法的应用

机器人导航



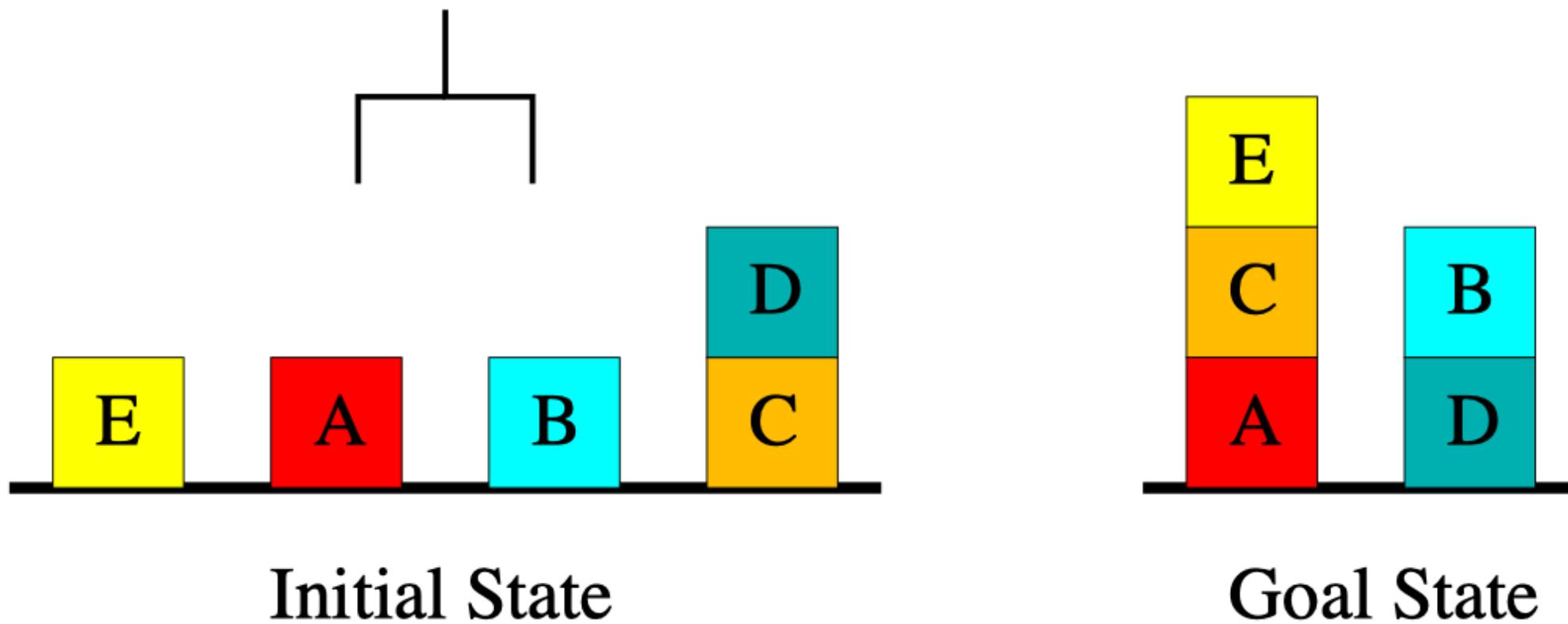
A*算法的应用

游戏AI



A*算法的应用

经典规划



提纲

□ 启发式搜索

➤ 贪婪搜索

➤ A*搜索

➤ 启发式函数设计

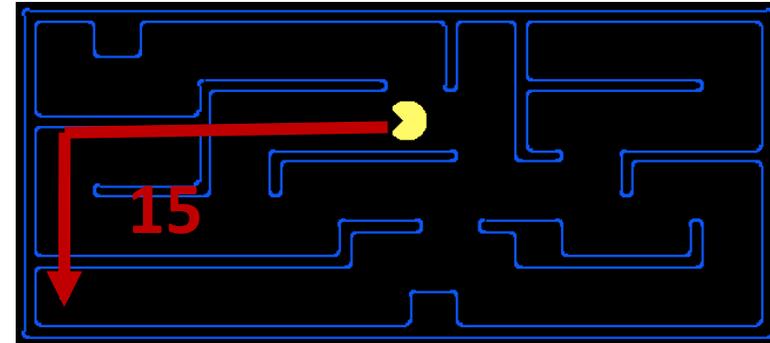
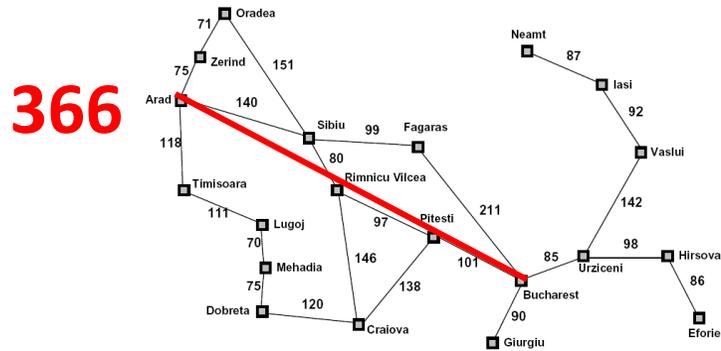
□ 本章小结



启发式函数 (heuristic function)

启发式搜索的关键在于如何设计启发式函数？

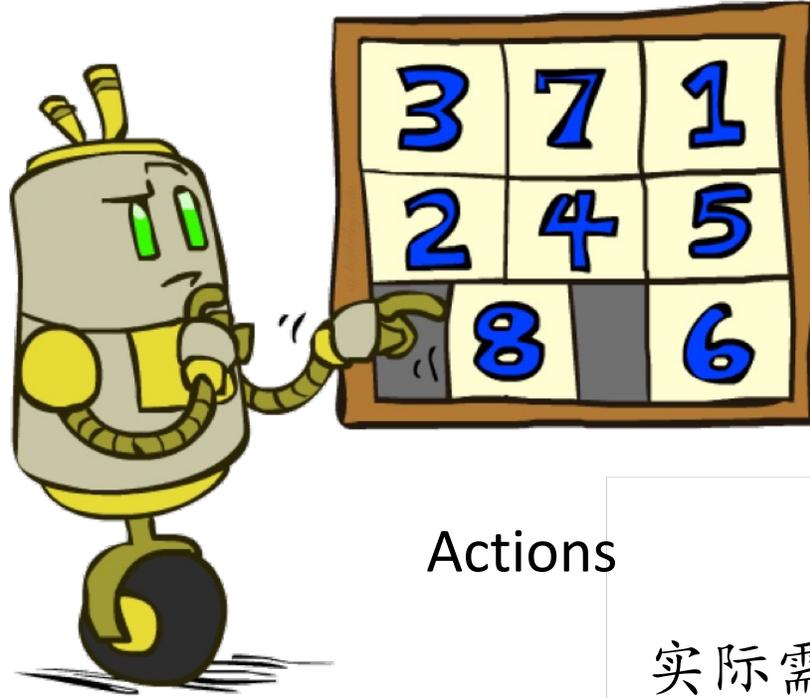
可采纳的启发式函数 (admissible heuristics) 可以看做松弛问题 (relaxed problems) 的解



例子：八数码问题

7	2	4
5		6
8	3	1

Start State



Actions

	1	2
3	4	5
6	7	8

Goal State

实际需要的步数为26步

Any idea for
admissible heuristic?

例子：八数码问题

启发式函数

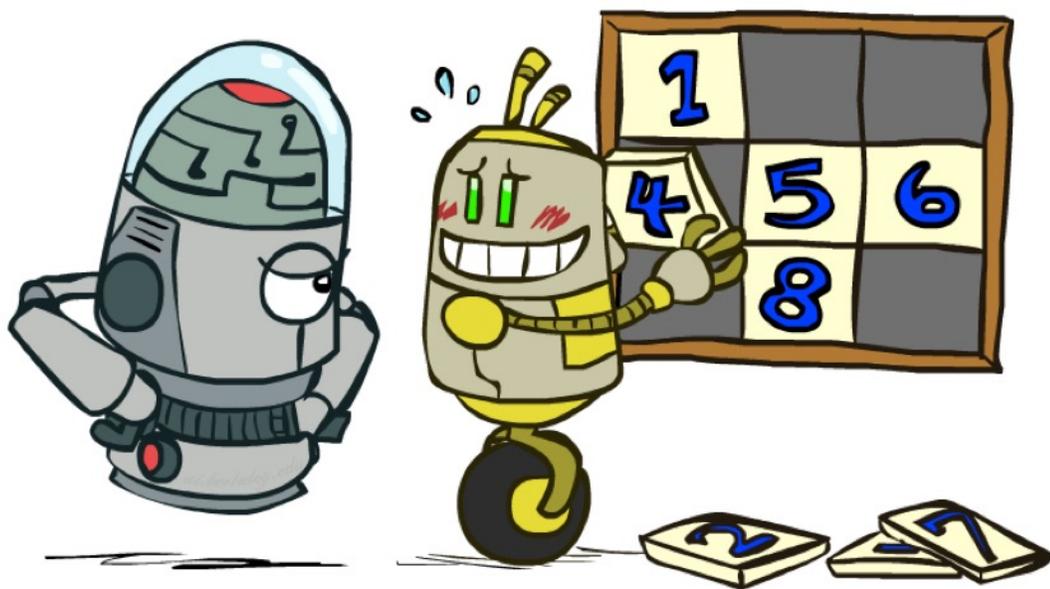
✓ 不在正确位置的棋子数

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State



$$h(s) = ?$$

$$h(s) = 8$$

例子：八数码问题

启发式函数

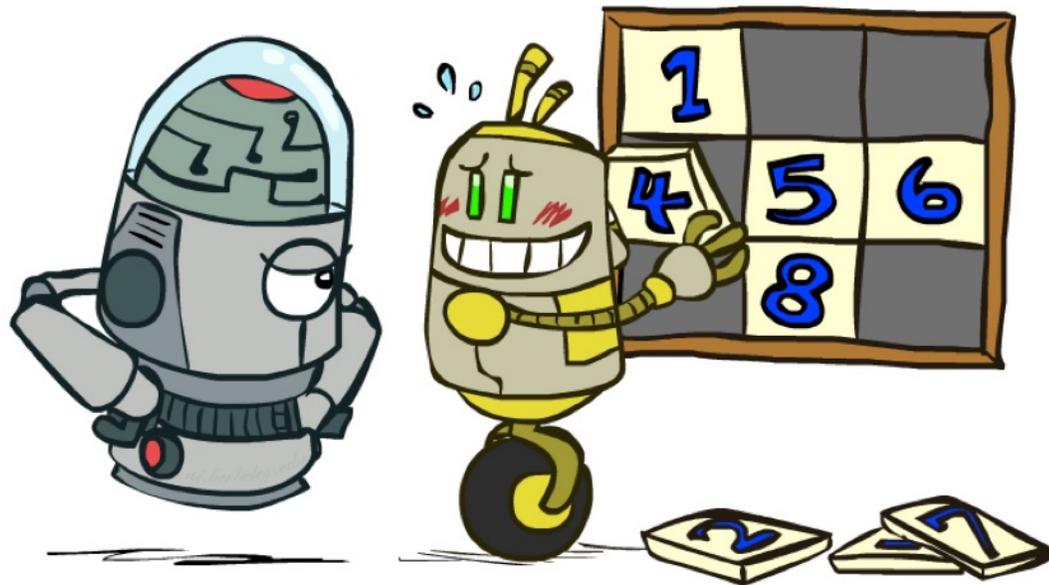
✓ 所有棋子到目标位置的距离和
(曼哈顿距离)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State



$$h(s) = ?$$

$$h(s) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

例子：八数码问题

100个八数码问题，不同搜索策略扩展的平均节点数和有效分支因子

d	Search Cost (nodes generated)			Effective Branching Factor		
	BFS	$A^*(h_1)$	$A^*(h_2)$	BFS	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

启发式函数 (heuristic function)

- 如果启发式函数 $h(n) = 0$ 会怎么样?
 - 如果采用真实的代价作为启发式函数会怎么样?
-
- ✓ 第一种等价于一致性代价搜索
 - ✓ 第二种是最佳情况, 但是不可能实现
 - ✓ 合理的启发式函数介于两者之间

提纲

□ 启发式搜索

➤ 贪婪搜索

➤ A*搜索

➤ 启发式函数设计

□ 本章小结

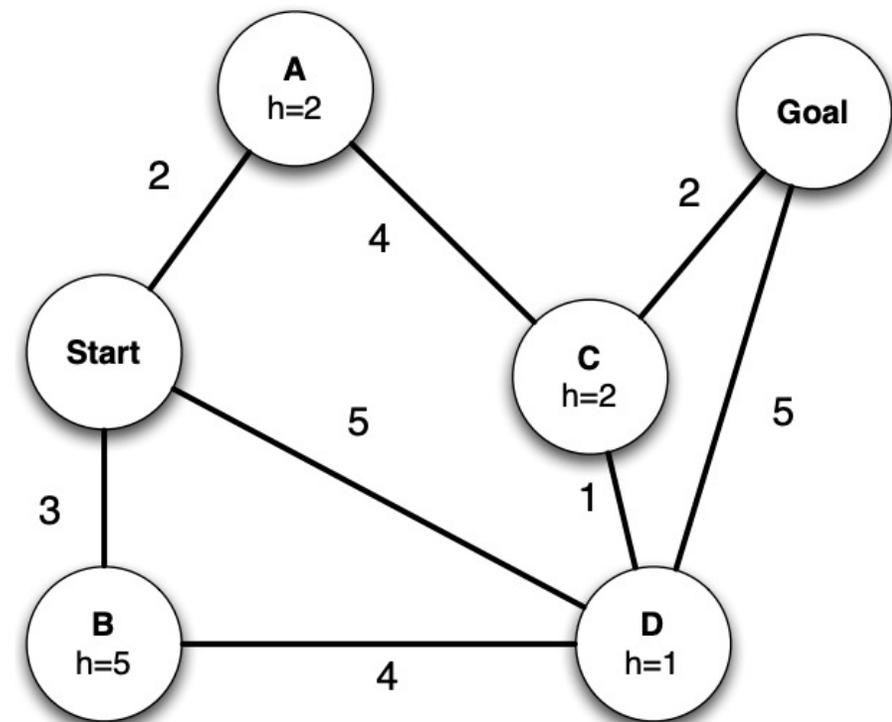


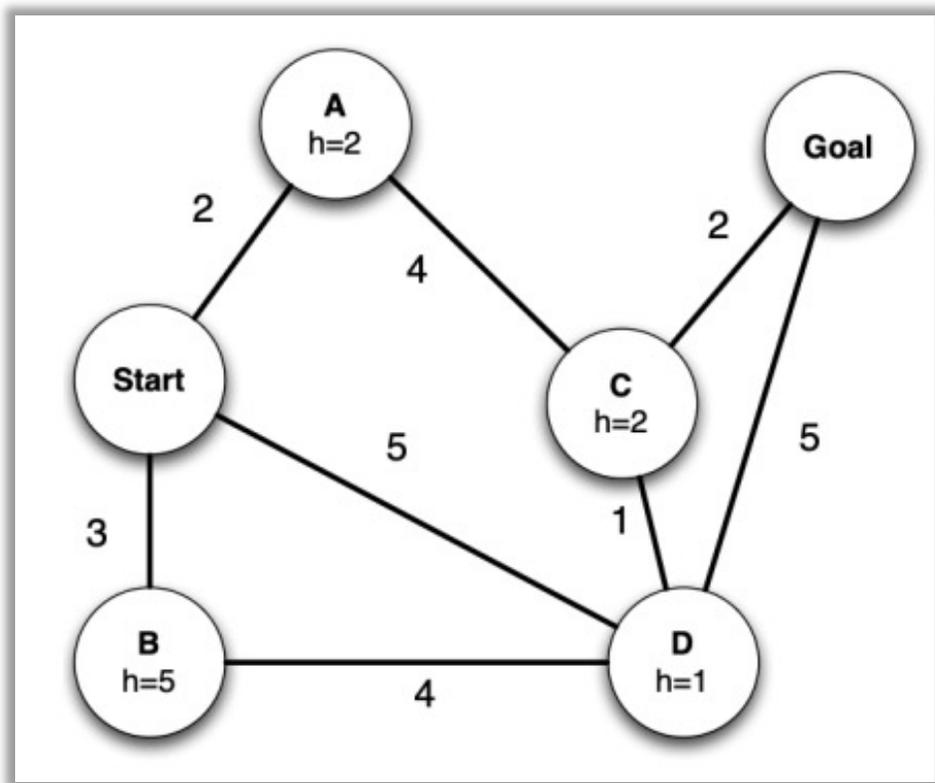
小结

- 启发式函数：当前状态距离目标状态路径消耗的估计
- A*算法：同时考虑初始状态到当前节点的路径消耗，也考虑当前节点到目标节点的路径消耗估计
- A*算法树搜索和图搜索的最优性条件：可采纳性和一致性
- 如何设计启发式函数是启发式搜索的关键，通常可以采用一个松弛问题

课堂练习

- 考虑下图的问题，请分别使用5种搜索算法的图搜索版本进行求解，给出搜索过程中状态结点的扩展顺序，以及算法最终返回的路径（若多个结点优先度相同，则优先扩展字典序较小的结点），起始状态和目标状态分别记为S和G
 - 深度优先搜索（DFS）
 - 广度优先搜索（BFS）
 - 一致性代价搜索（UCS）
 - 贪婪搜索（Greedy Search）
 - A* 搜索（A* Search）



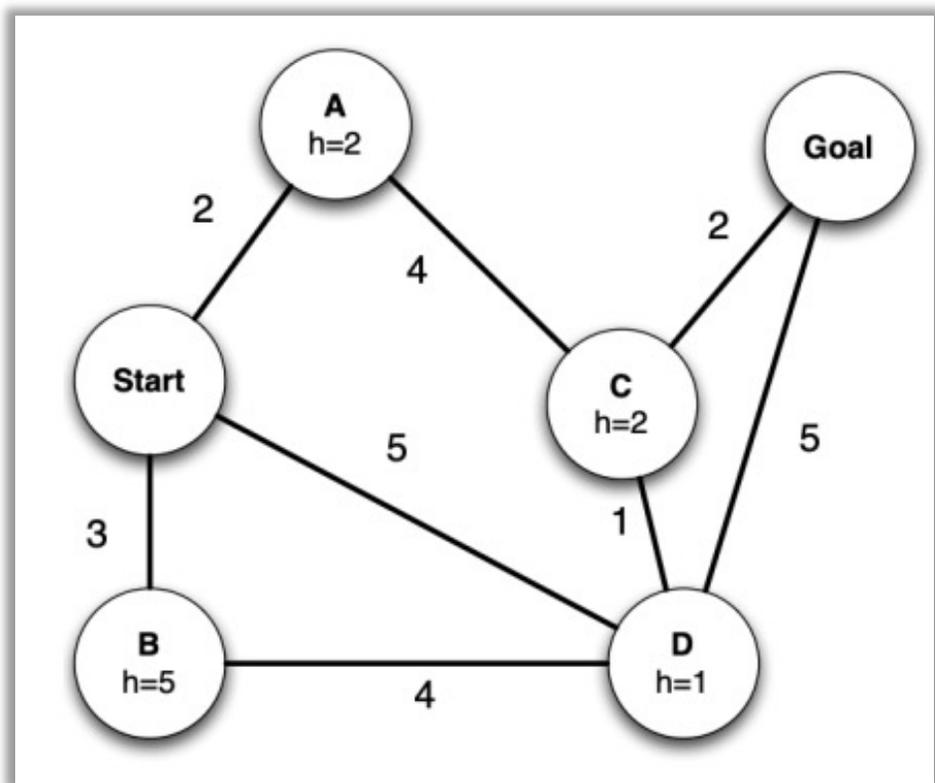


答案:

✓ 节点扩展: Start, A, C, D, B, Goal

✓ 路径: Start-A-C-D-Goal

a) 深度优先搜索

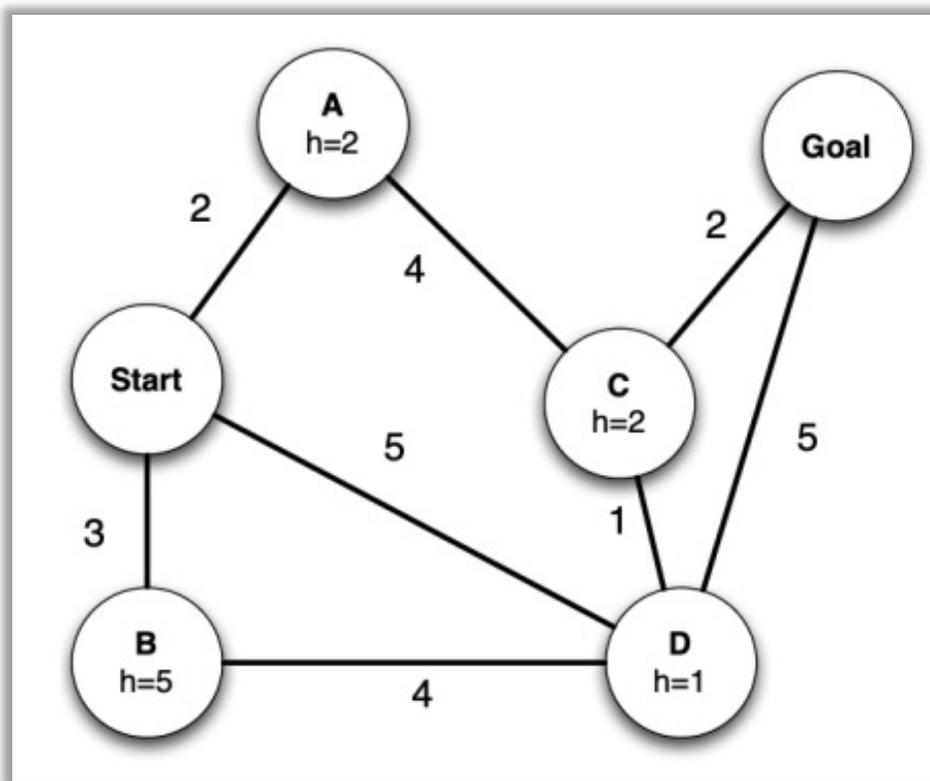


b) 广度优先搜索

答案:

✓ 节点扩展: Start, A, B, D, C, Goal

✓ 路径: Start-D-Goal

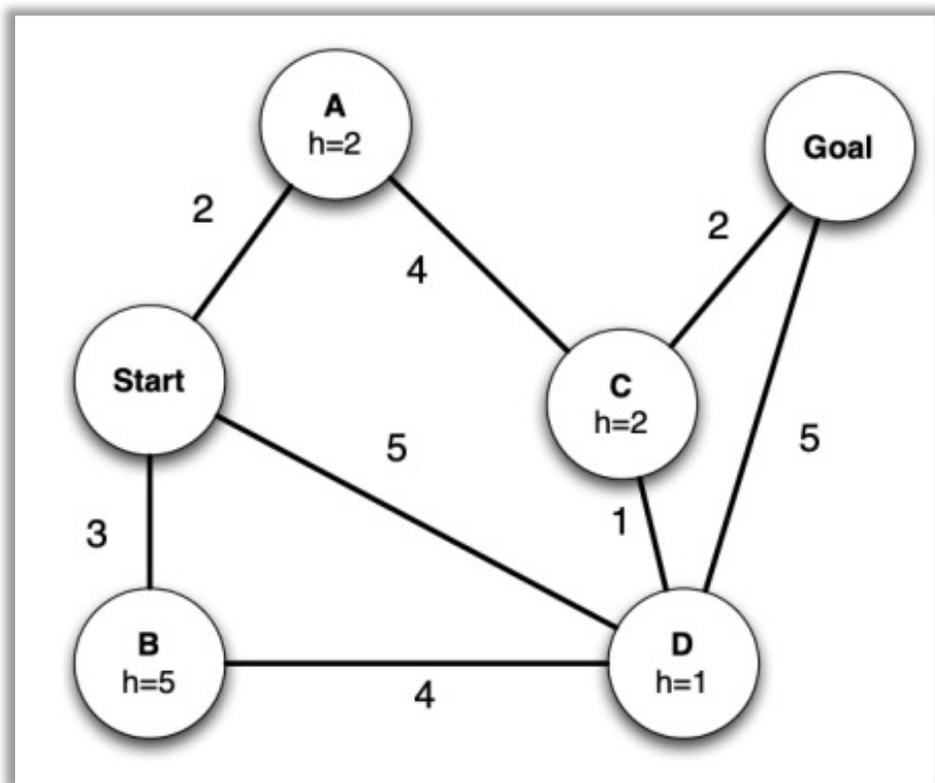


答案:

✓ 节点扩展: Start, A, B, D, C, Goal

✓ 路径: Start-A-C-Goal

c) 一致性代价搜索

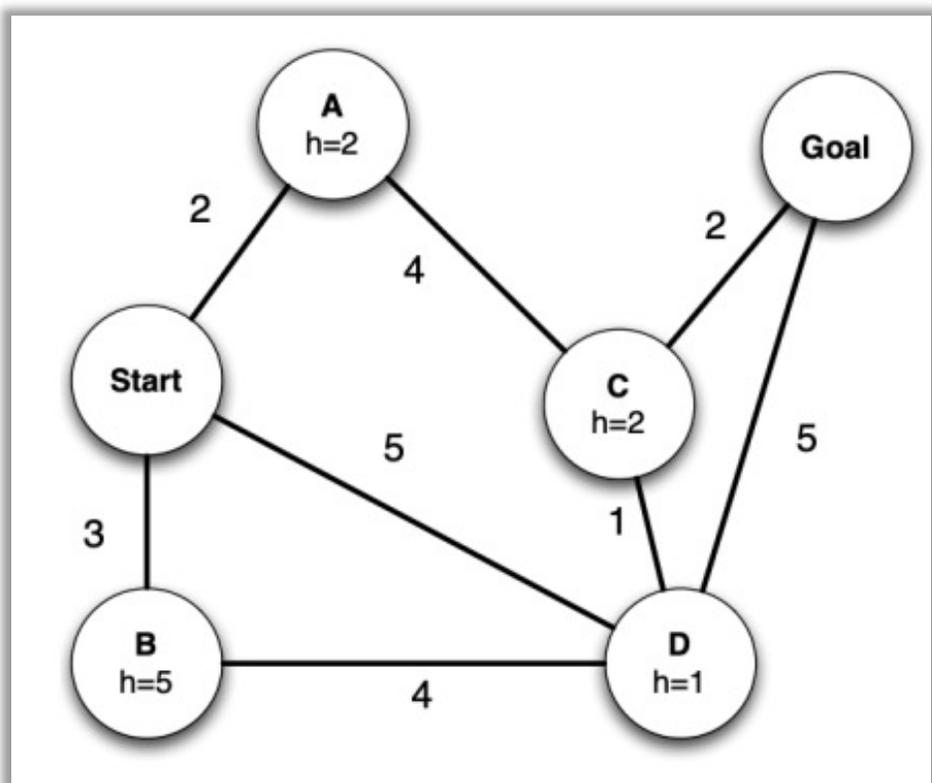


答案:

✓ 节点扩展: Start, D, Goal

✓ 路径: Start-D-Goal

d) 贪婪搜索



答案:

✓ 节点扩展: Start, A, D, B, C, Goal

✓ 路径: Start-A-C-Goal

e) A*搜索