南京大学
NANJING UNIVERSITY

人工智能导论

启发式搜索

郭兰哲

南京大学 智能科学与技术学院

Homepage: www.lamda.nju.edu.cn/guolz

Email: guolz@nju.edu.cn

提纲

- 启发式搜索
  - 贪婪搜索
  - A*搜索
  - 启发式函数
- 本章小结

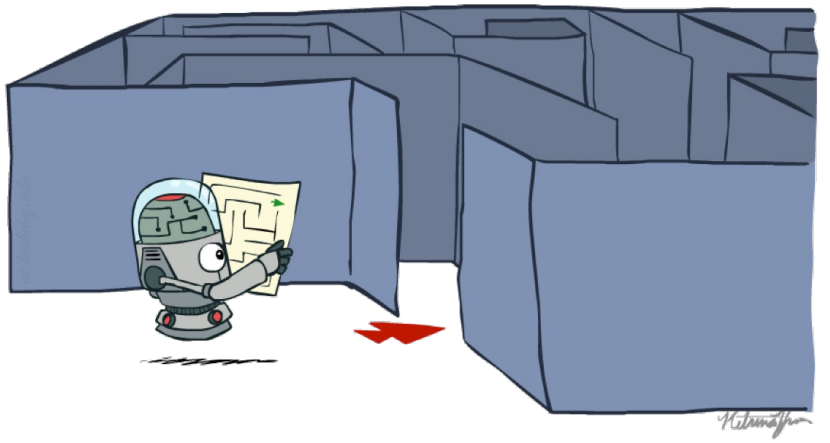
提纲

- 启发式搜索
  - 贪婪搜索
  - A*搜索
  - 启发式函数设计
- 本章小结

回顾：搜索

搜索问题

- ✓ 状态空间
- ✓ 起始和目标状态
- ✓ 状态转移
- ✓ 动作和代价



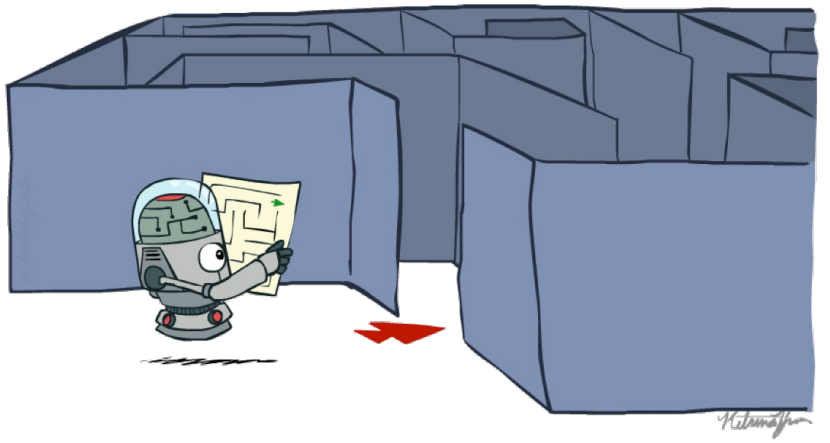
搜索算法

- ✓ 构建搜索树
- ✓ 选择不同的边缘节点进行扩展
- ✓ 最优策略：代价最小的到达目标节点的行动序列

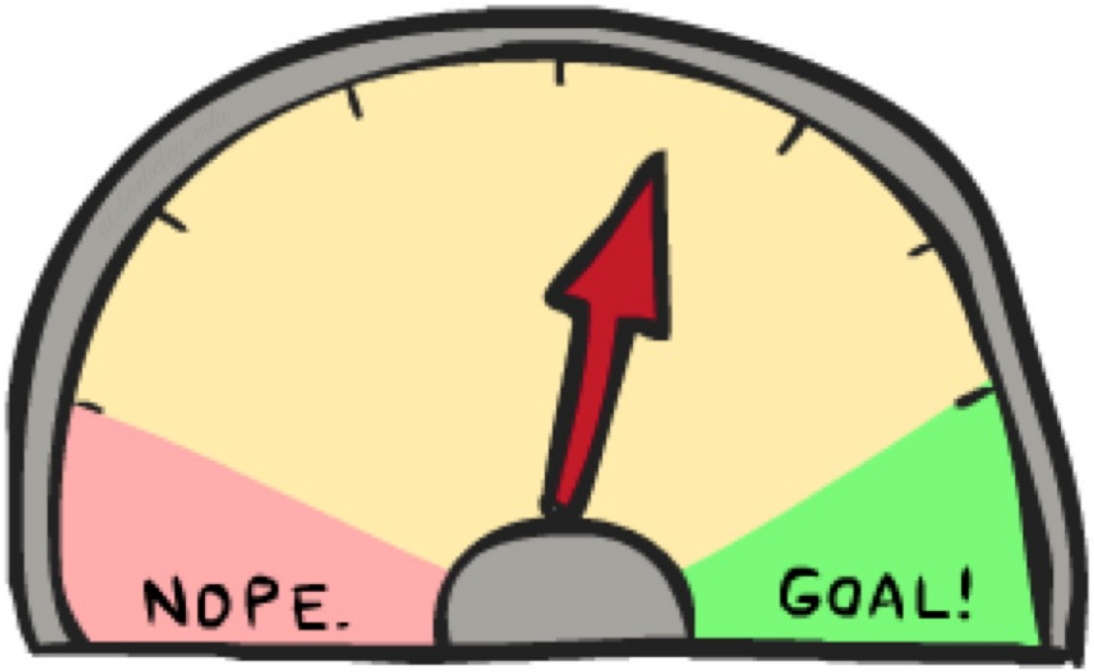
回顾：无信息搜索

无信息搜索算法

- ✓ 宽度优先搜索：逐层扩展节点
- ✓ 深度优先搜索：优先扩展最深的节点
- ✓ 迭代加深的深度优先搜索：限制深度的DFS
- ✓ 一致性代价搜索：优先扩展路径代价最小的节点



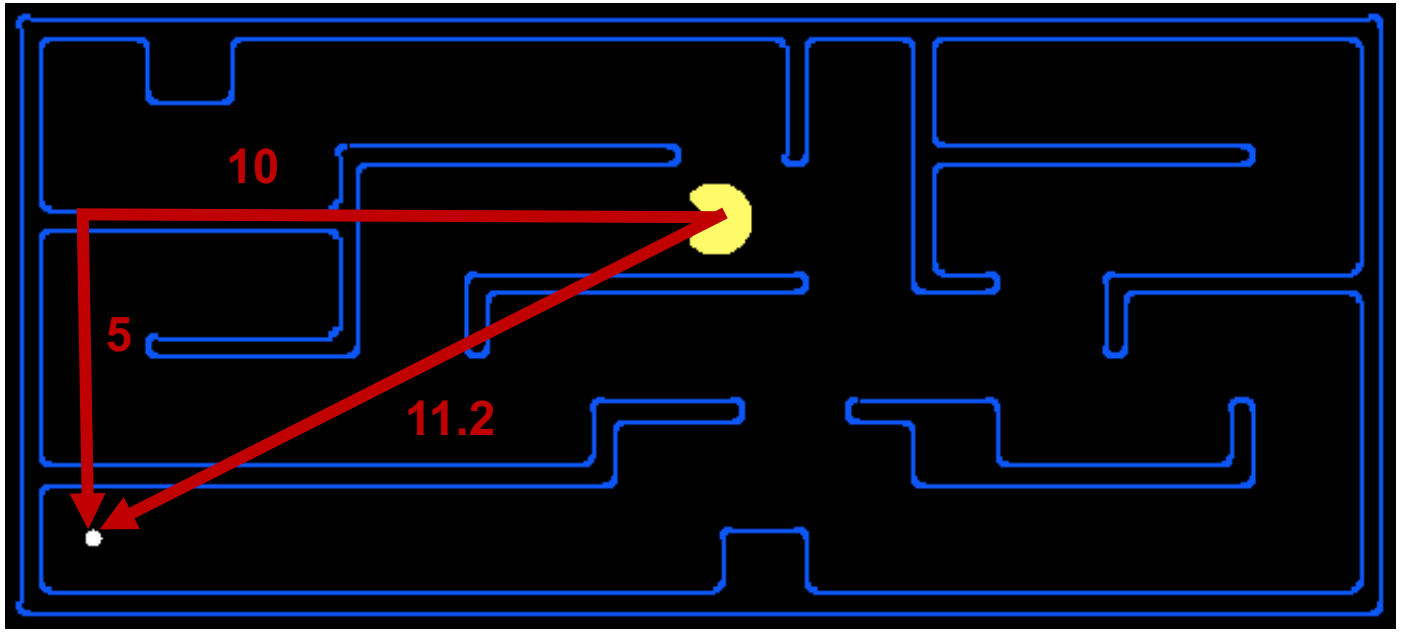
有信息搜索（Informed Search）

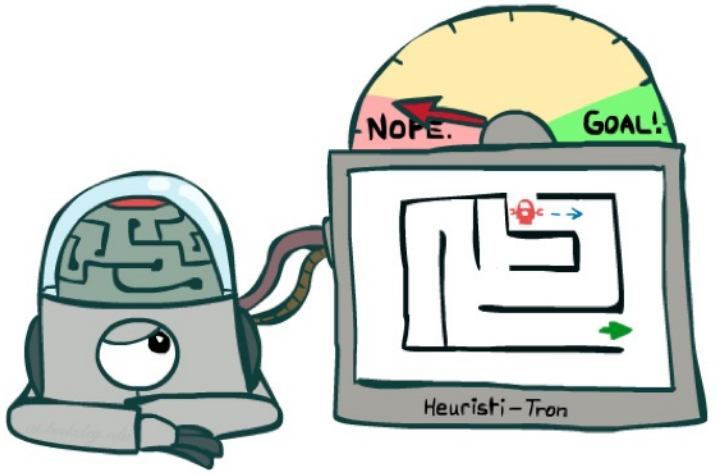


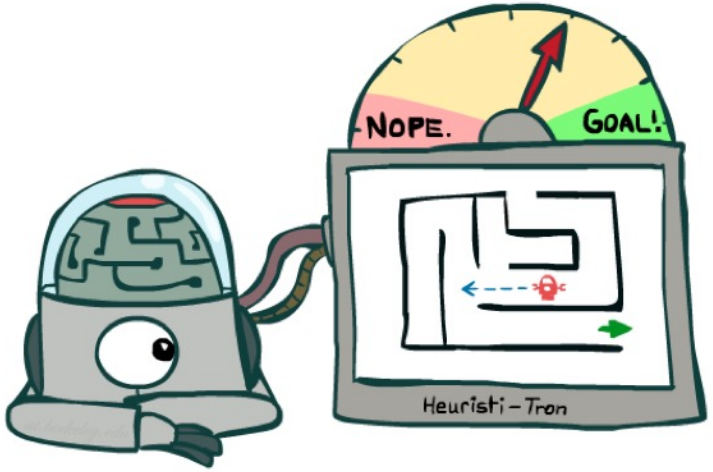
启发式信息（heuristic）

启发式信息：

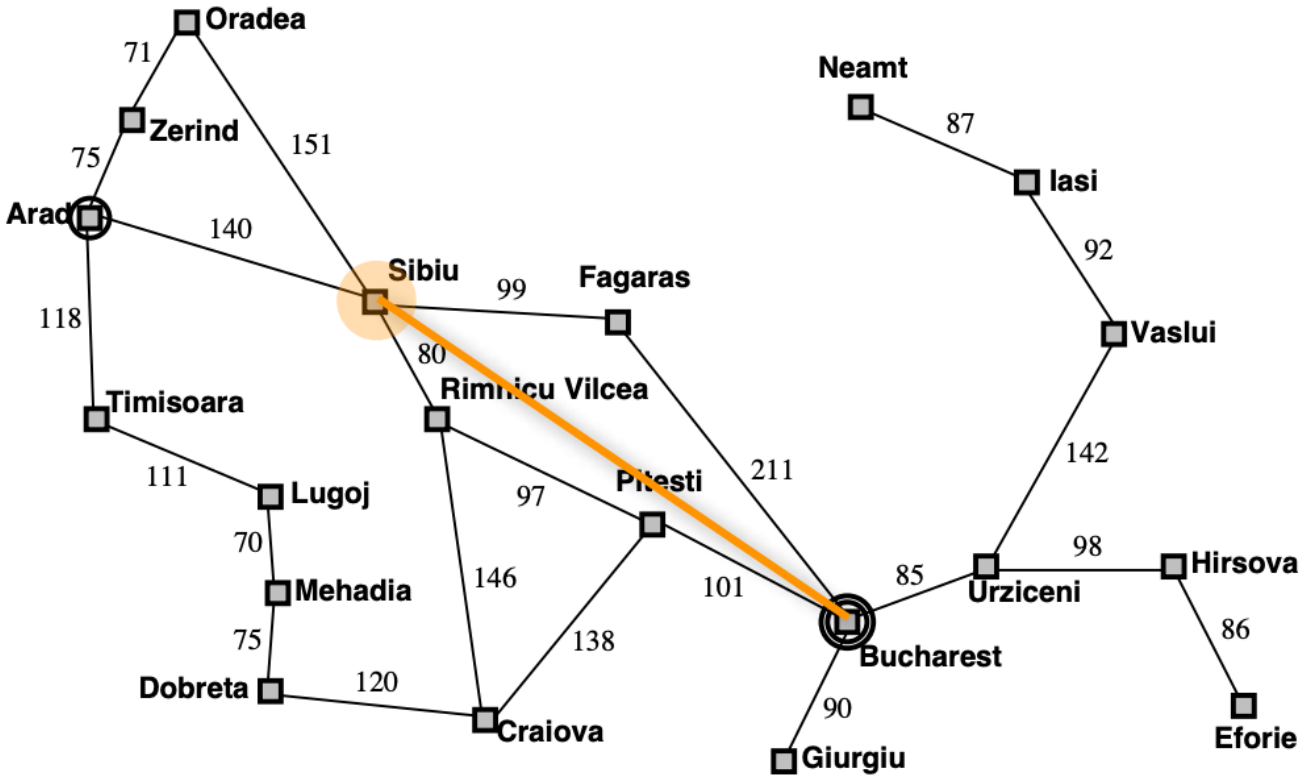
- ✓ 估计离目标状态有多远的信息
- ✓ 不同的问题需要设计不同的启发式信息
- ✓ 例如：曼哈顿距离、欧式距离







启发式信息（heuristic）



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
  - 启发式函数设计
- 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？

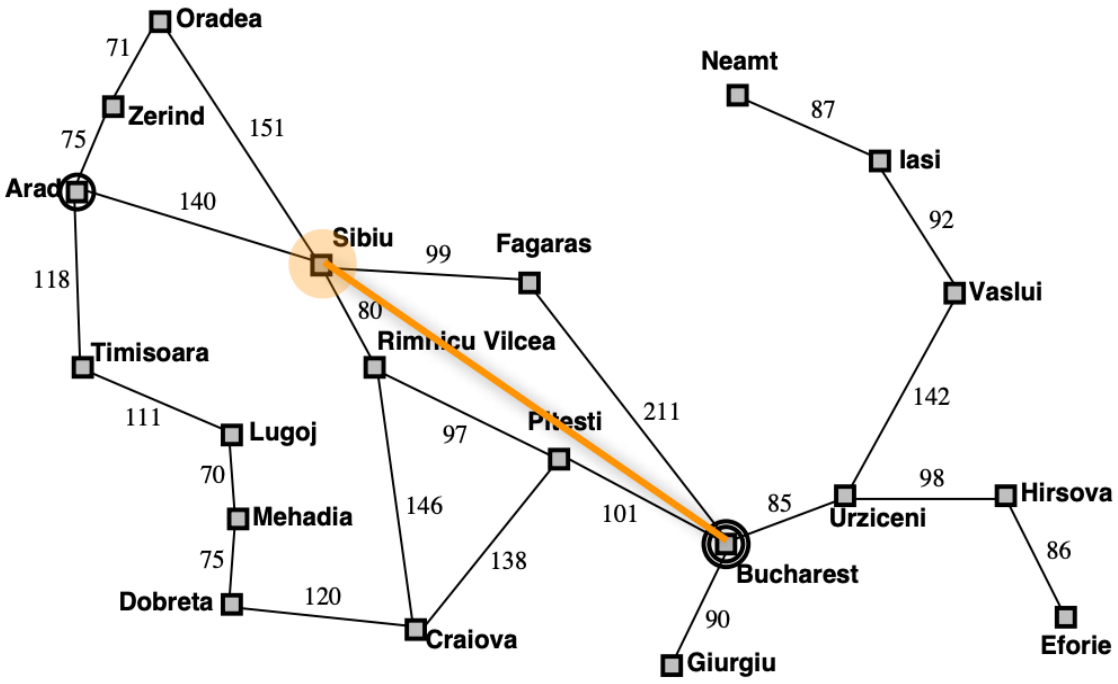
- ✓ 41-25=16
- ✓ 16-5*3=6
- ✓ 1-1=0

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)

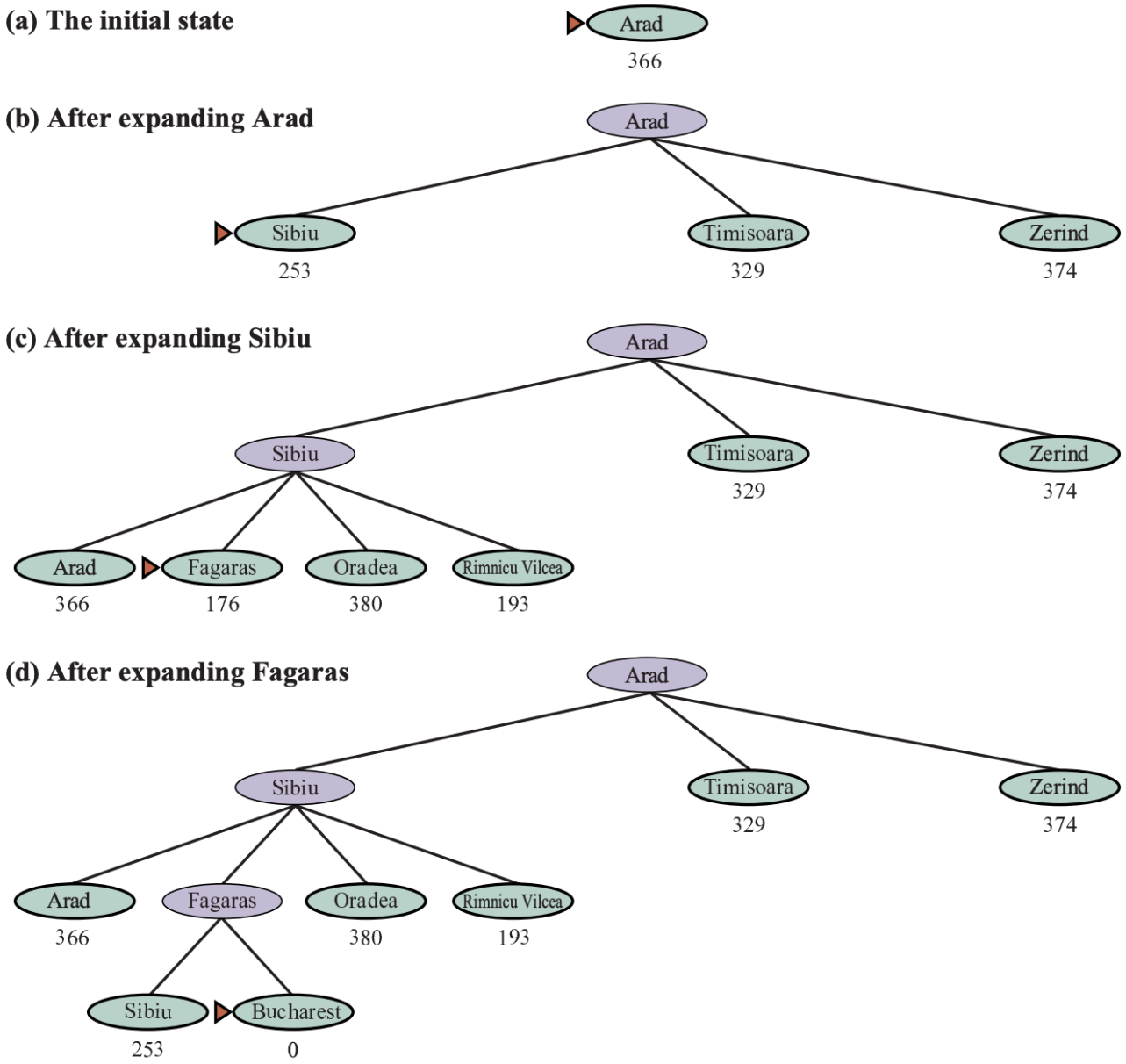
每次均选择离目标状态最近的节点



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

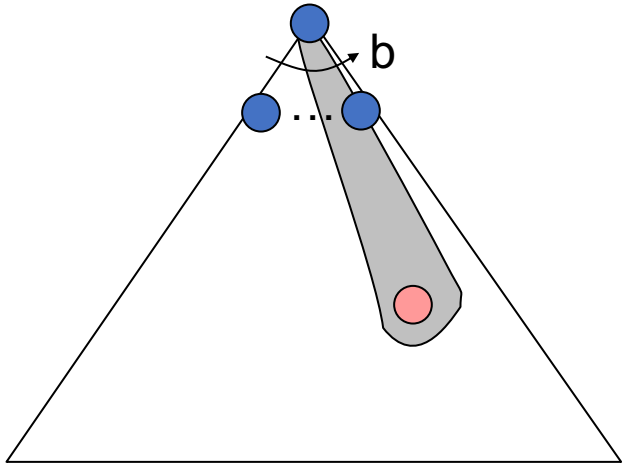
h(x)

贪婪搜索（greedy search)

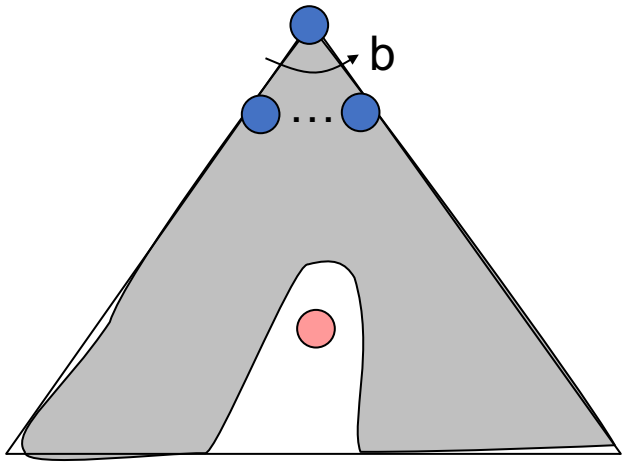


贪婪搜索（greedy search）

- 策略：每一步均扩展距离目标状态更近的节点
  - ✓ 启发式信息：估计当前状态距离目标状态的距离



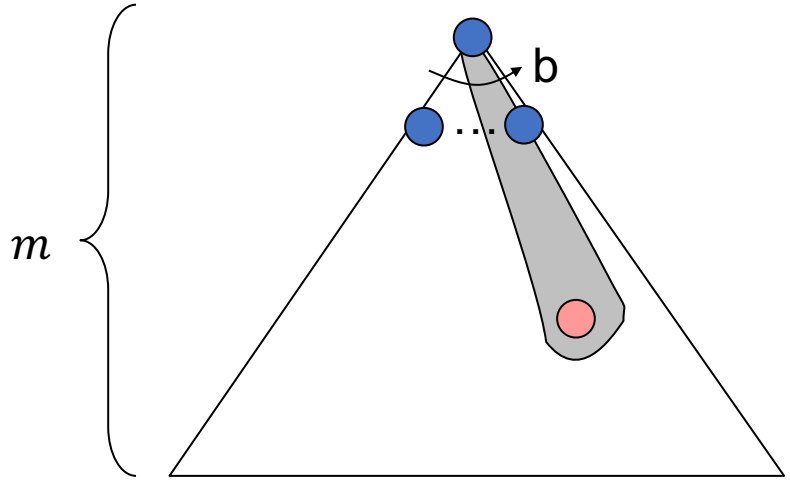
- Common case：
  - ✓ 距离目标越来越近



- Bad case：
  - ✓ Badly-guided DFS

性能分析

- 时间复杂度?
  - $O(b^m)$
  - 如果有好的启发式函数，复杂度可以有效降低
- 空间复杂度?
  - $O(b^m)$
- 完备性?
  - No
- 最优性?
  - No

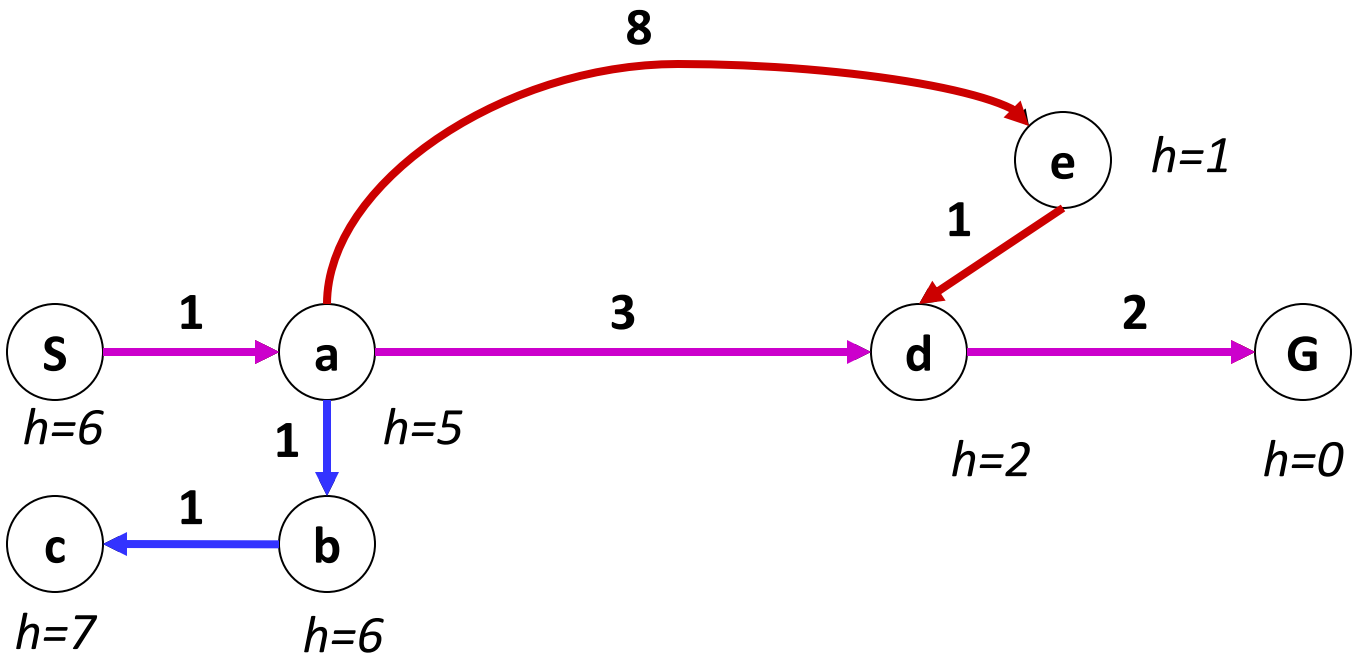


提纲

- 启发式搜索
  - 贪婪搜索
  - A*搜索
  - 启发式函数设计
- 本章小结

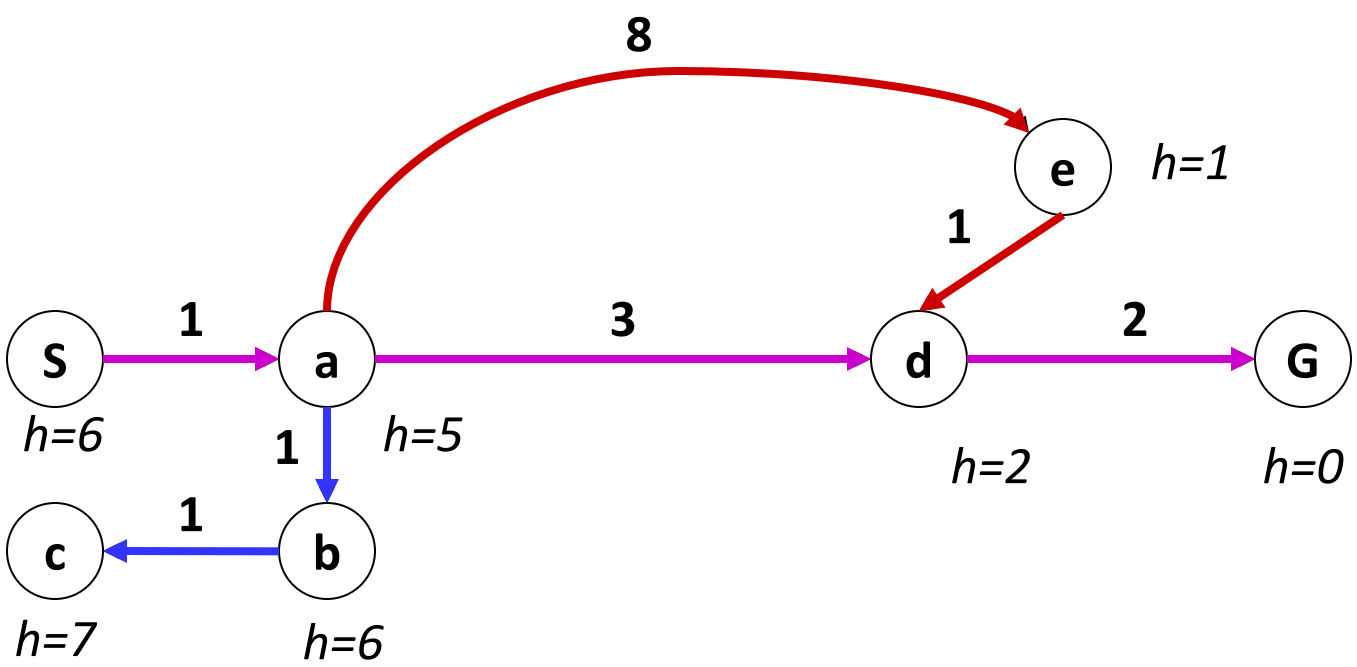
A*算法

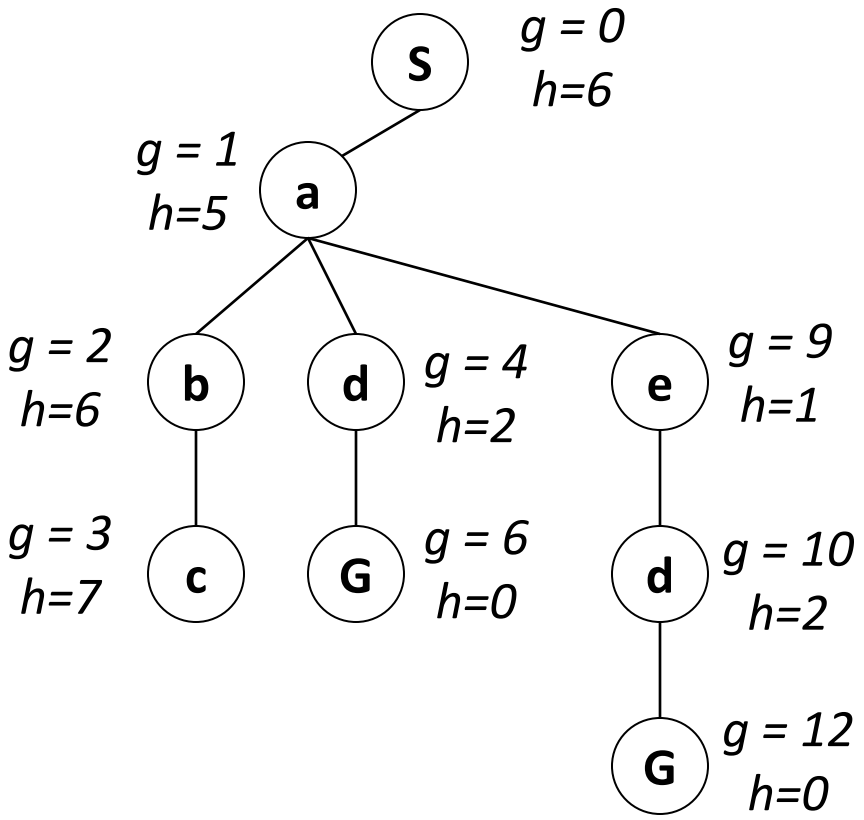
- 一致性代价搜索：路径的代价消耗
- 贪婪搜索：节点到目标状态的估计



A*算法

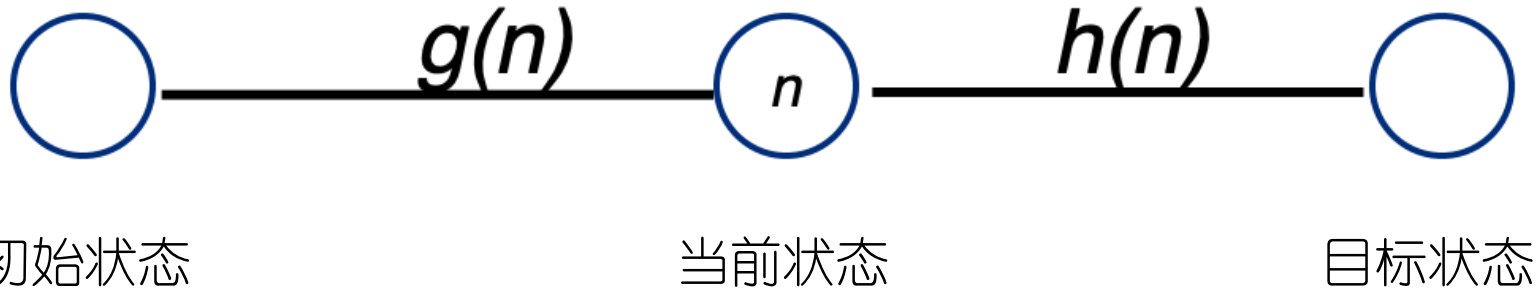
- 一致性代价搜索：路径的代价消耗
- 贪婪搜索：节点到目标状态的估计





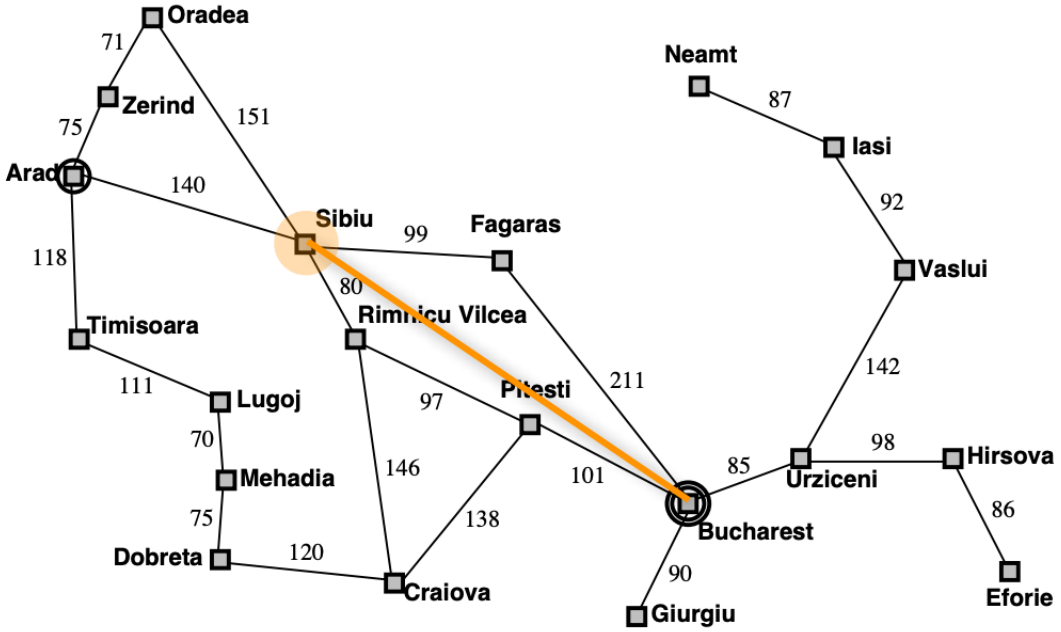
A*算法

A*算法： $f(n) = g(n) + h(n)$



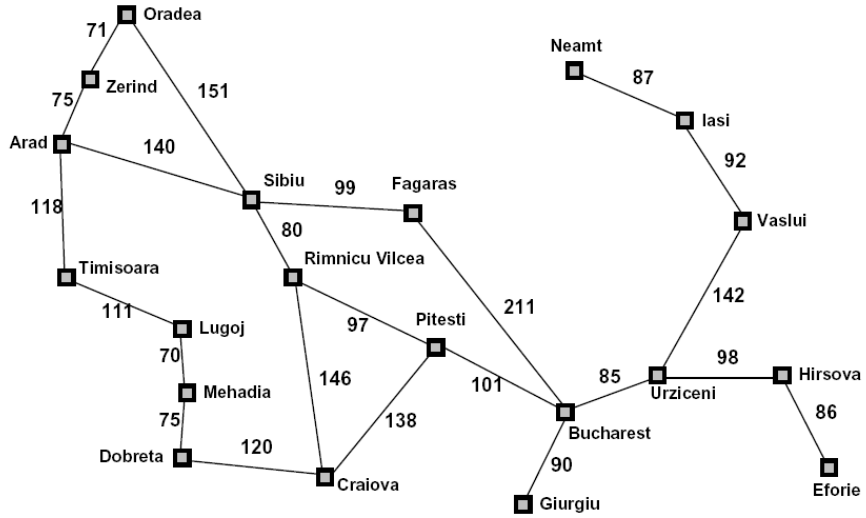
- $g(n)$表示从起始结点到结点n的开销代价值，$h(n)$表示从结点n到目标结点路径中所估算的最小开销代价值。
- 评价函数$f(n)$可视为经过结点n、具有最小开销代价值的路径。

A*算法

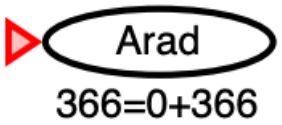


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

A*算法

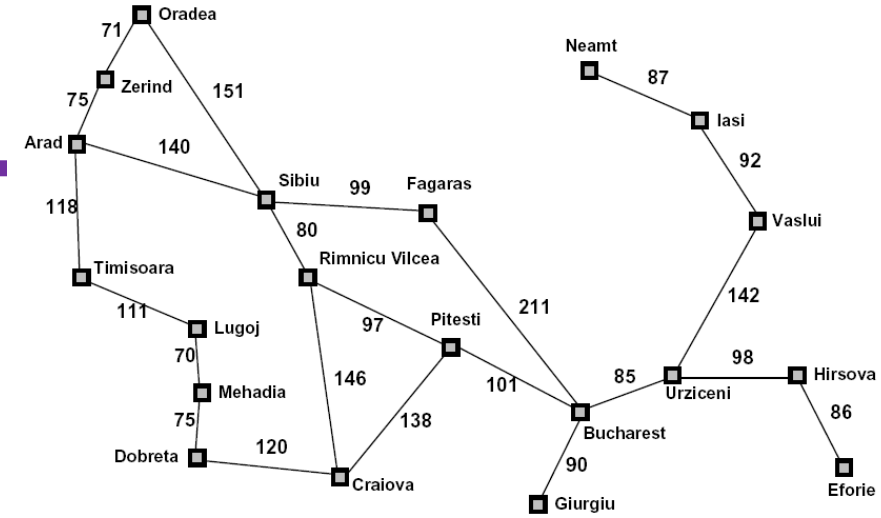


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

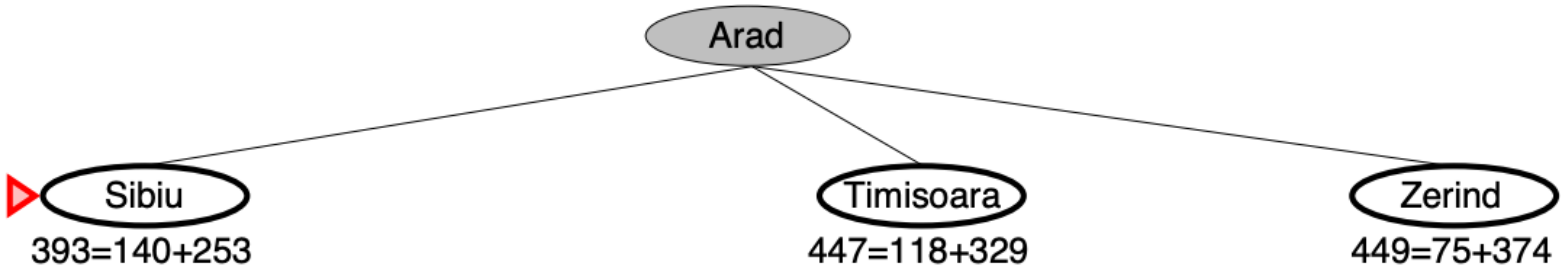


366=0+366

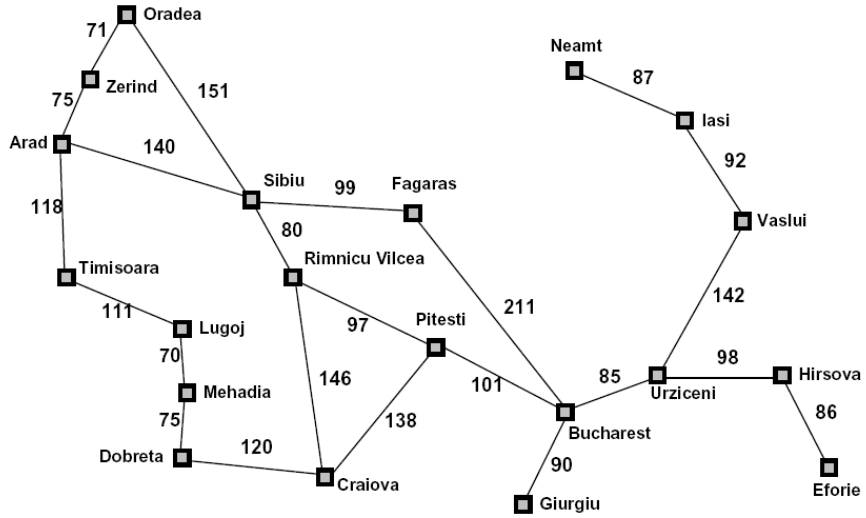
A*算法



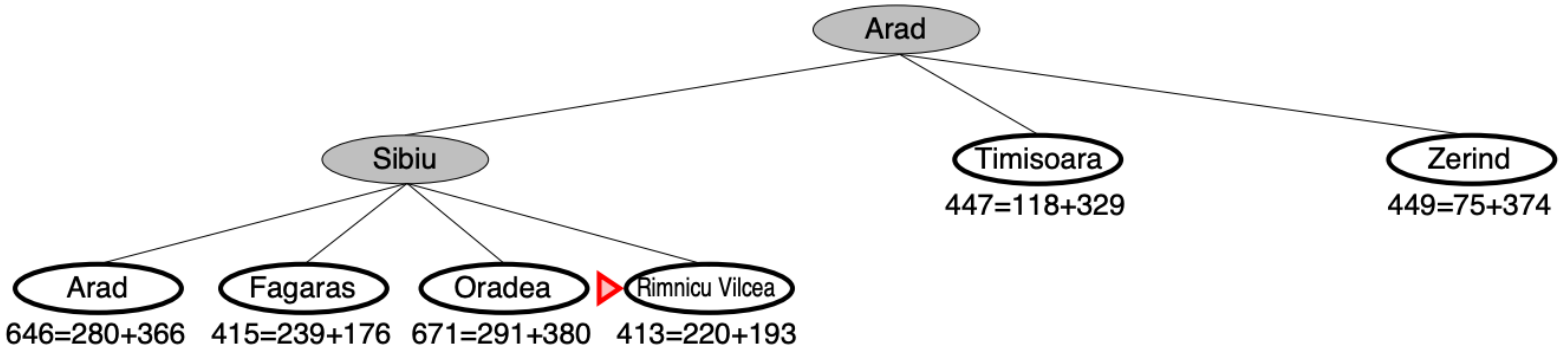
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



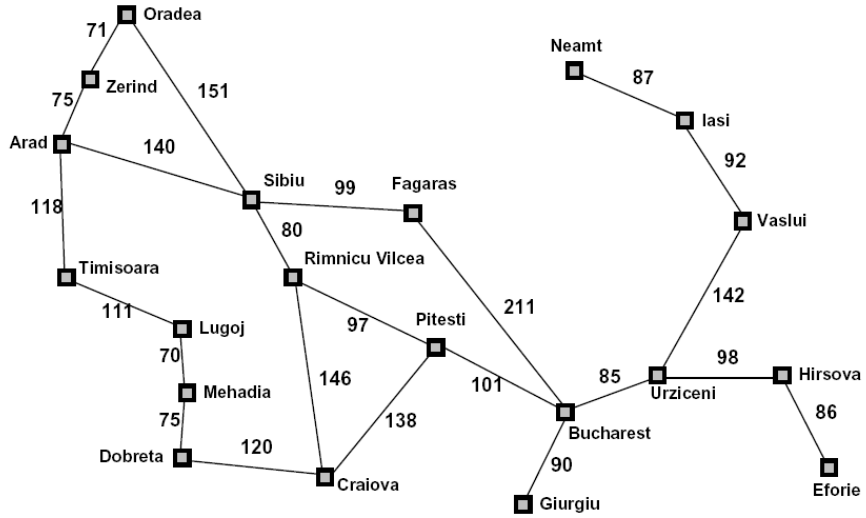
A*算法



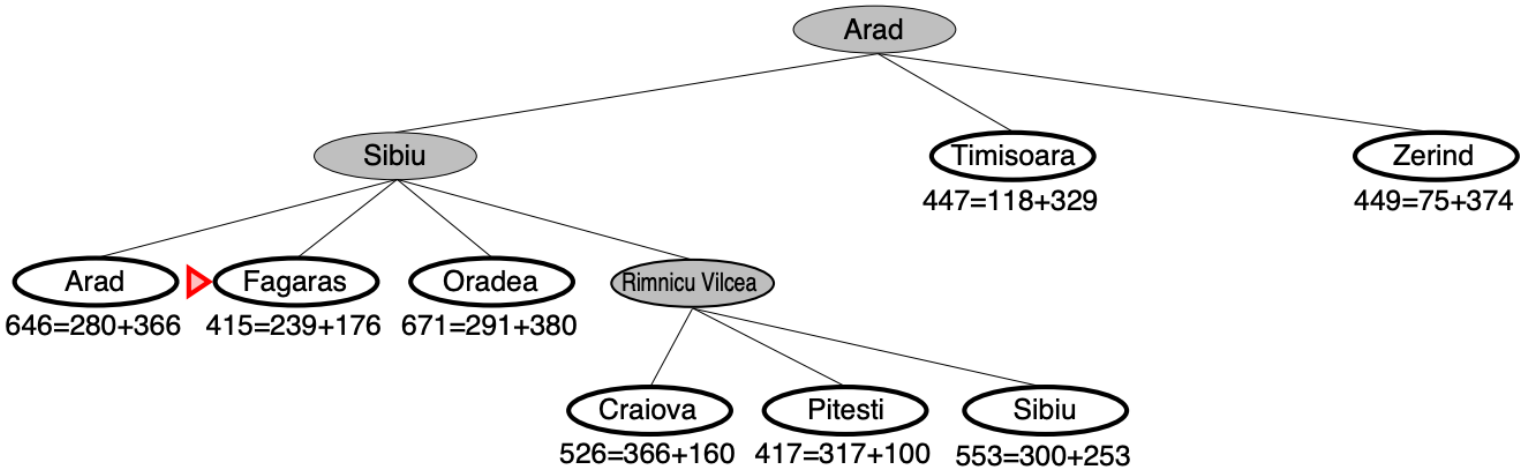
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



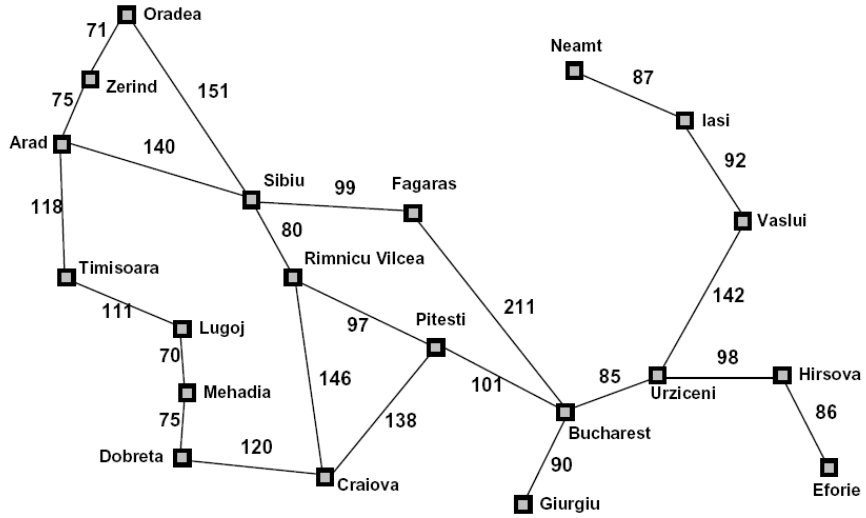
A*算法



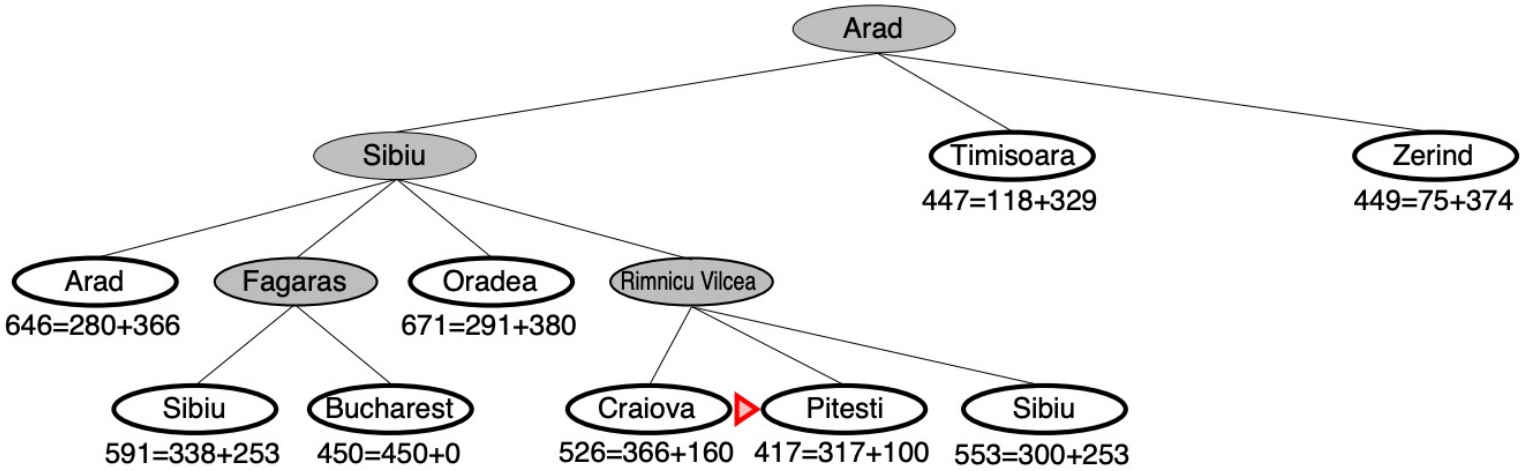
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



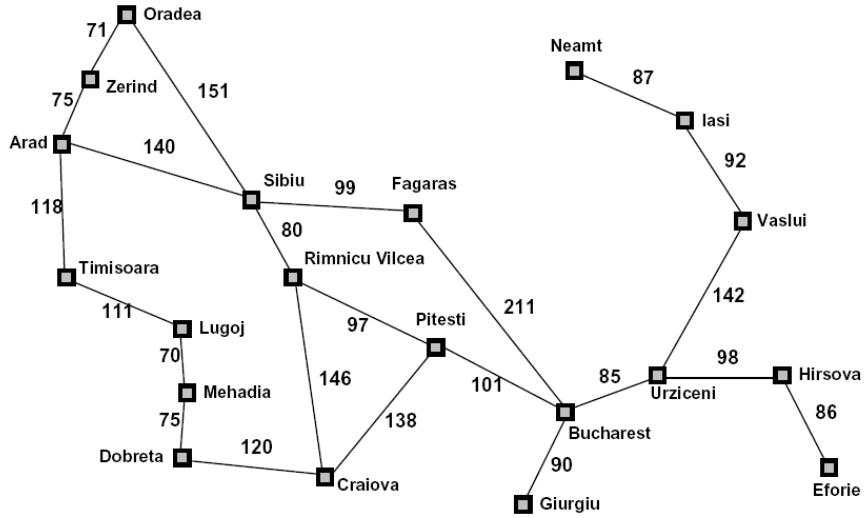
A*算法



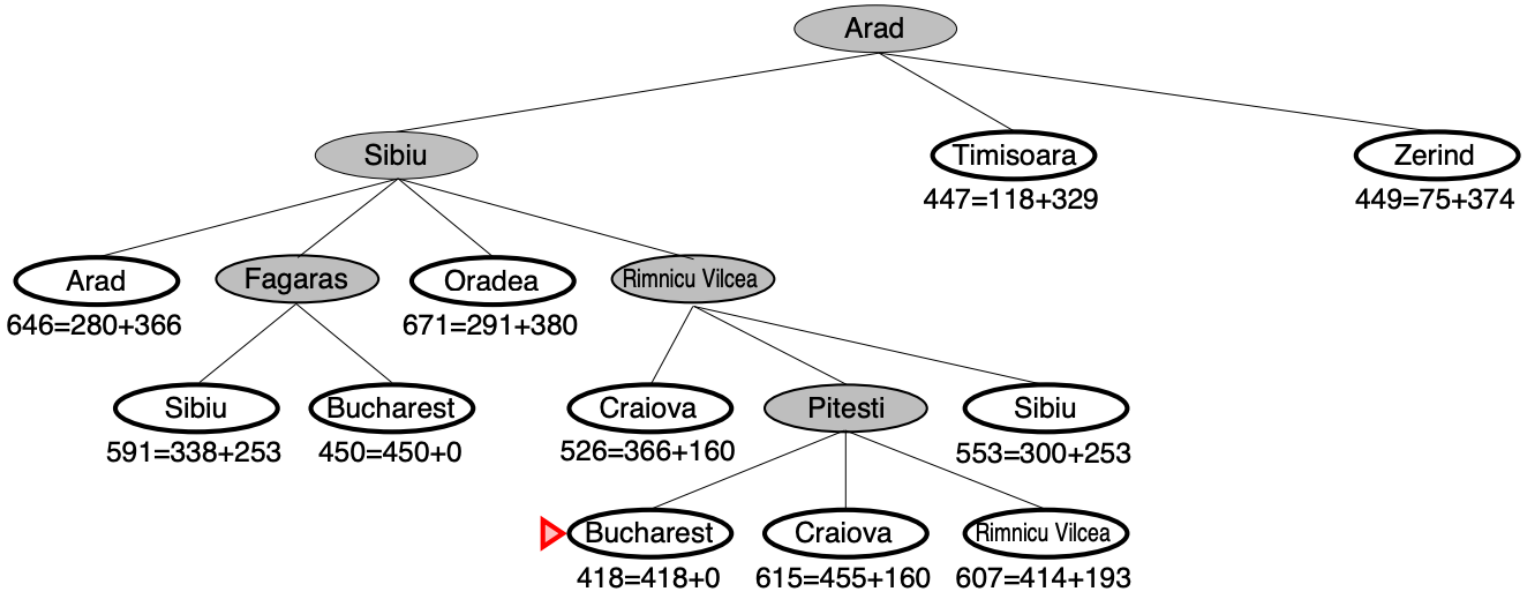
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



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



A*算法的完备性

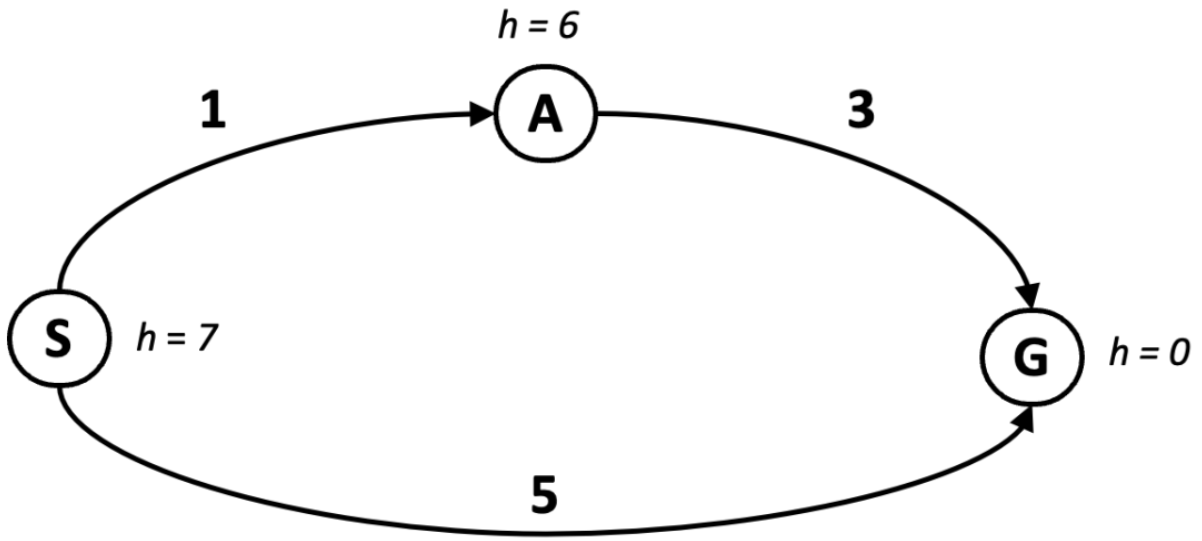
在一些常见的搜索问题中，状态数量是有限的，此时图搜索A*算法和排除环路的树搜索A*算法均是完备的，即一定能够找到一个解。

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

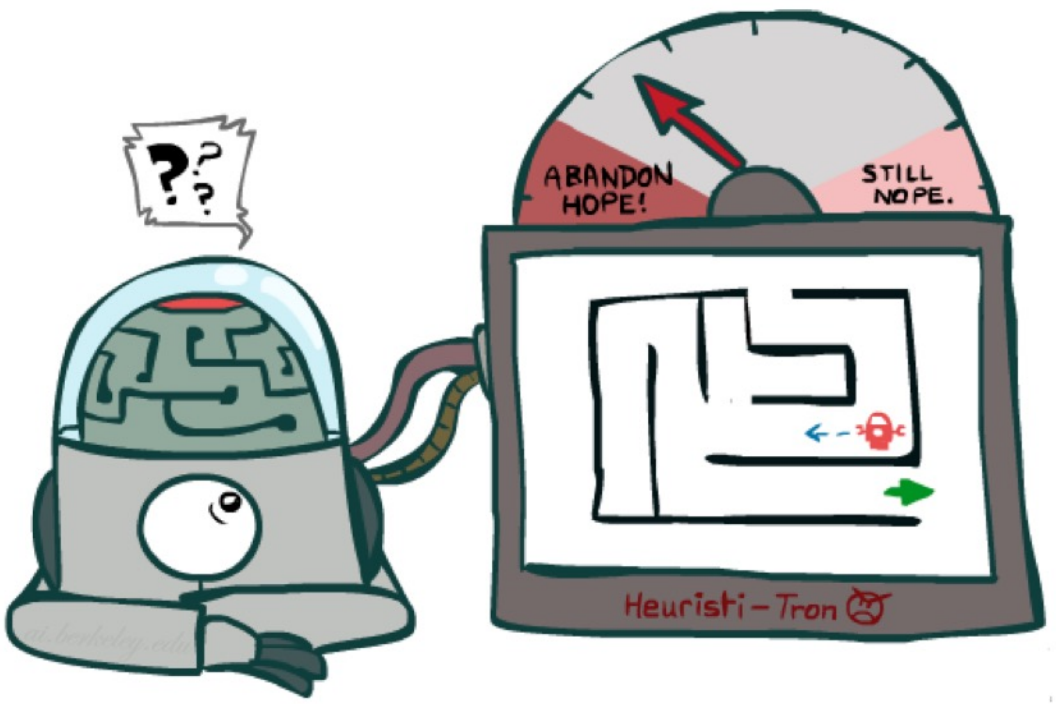
A*算法的最优性

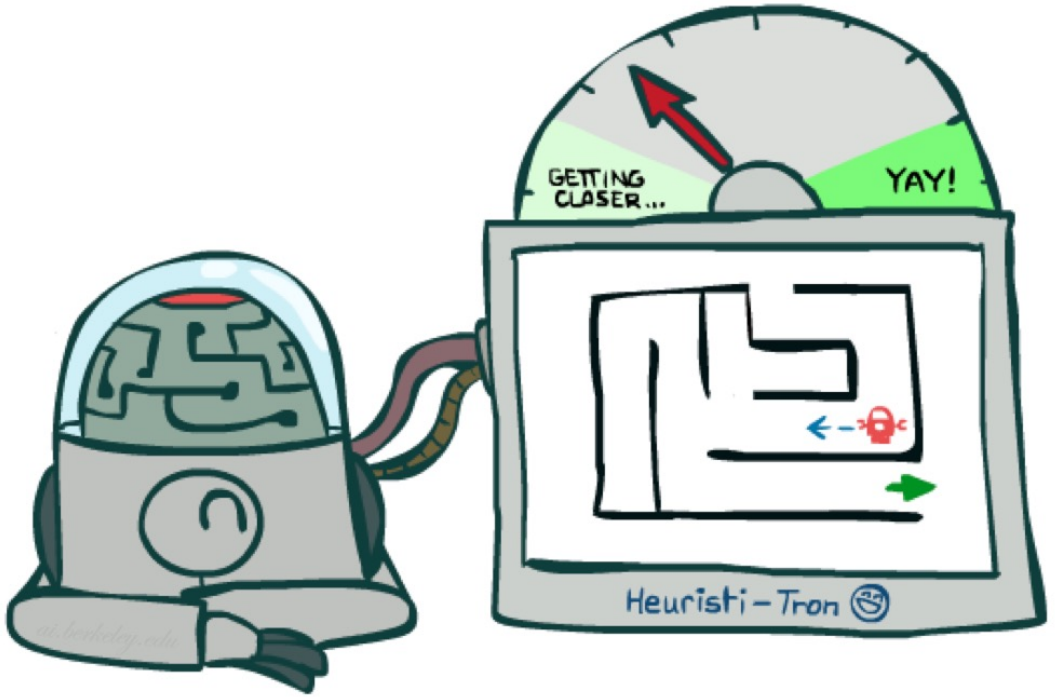
A*算法能否找到最佳路线？



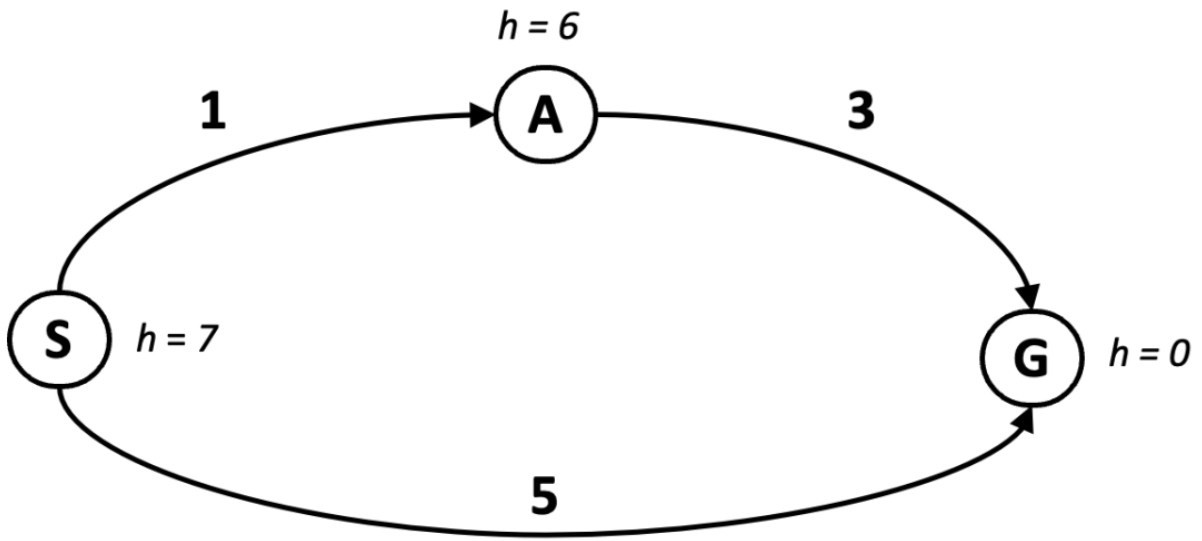
Quiz: 如何修改可以使算法找到最优解？

A*算法的最优性


ABANDON HOPE!
STILL NOPE.
Heuristi-Tron


GETTING CLOSER...
YAY!
Heuristi-Tron

A*算法的最优性



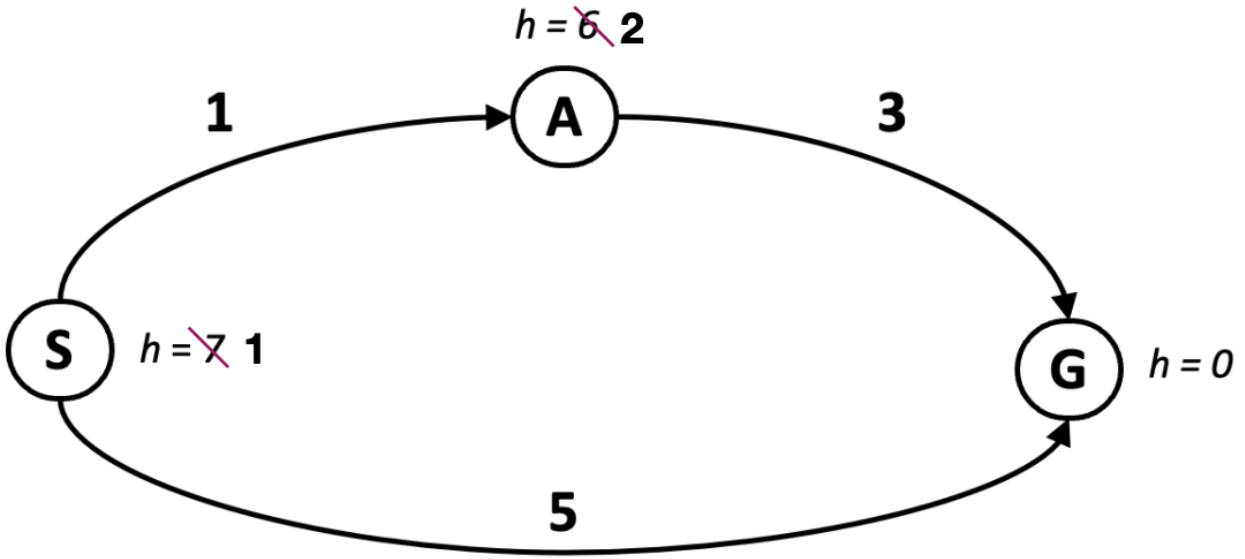
显然，为了保障A*算法的最优性，需要额外假设

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

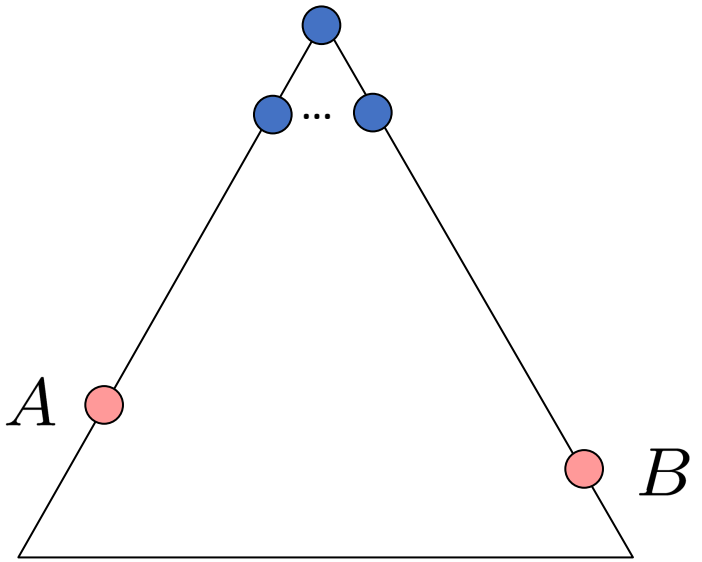
定理：

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：

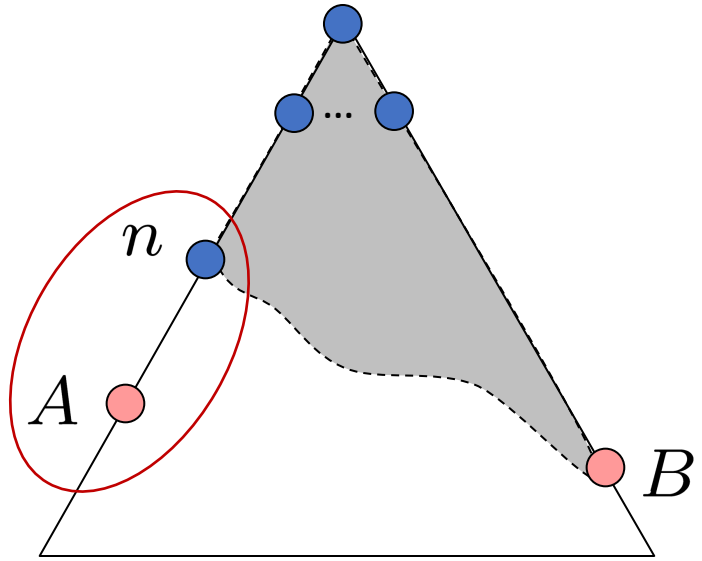
- A是最优的目标节点
- B是次优的目标节点
- 启发函数是可采纳的



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

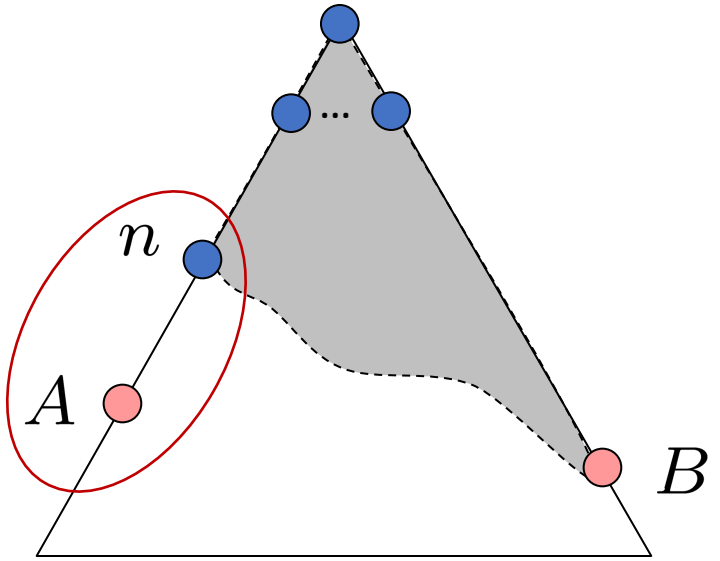


$$f(n) = g(n) + h(n)$$
$$f(n) \leq g(A)$$
$$g(A) = f(A)$$

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

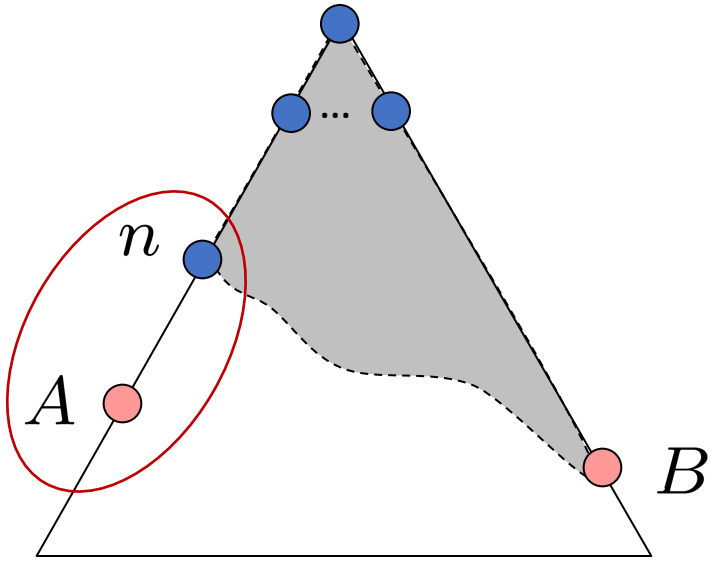


$$g(A) < g(B)$$
$$f(A) < f(B)$$

保障最优性的条件：可采纳性

证明：

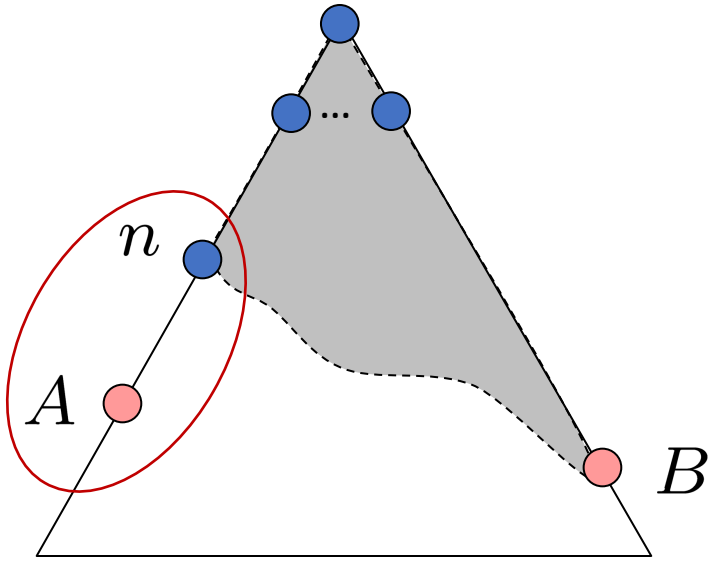
- 假设 B 位于边缘集
- A的某个父节点 n 也在边缘集
- 那么, n 比 B 更早被扩展
  1. $f(n) \leq f(A)$
  2. $f(A) \leq f(B)$
  3. $f(n) \leq f(A) \leq f(B)$



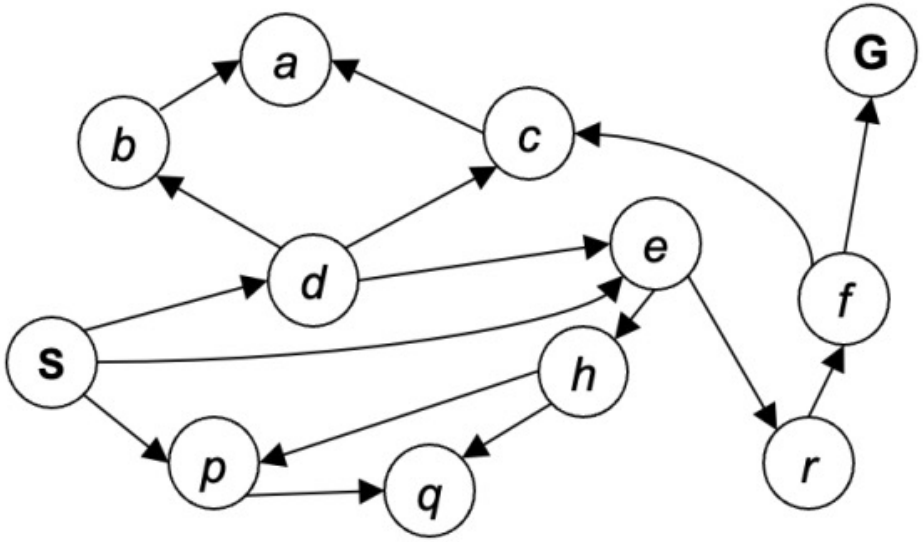
保障最优性的条件：可采纳性

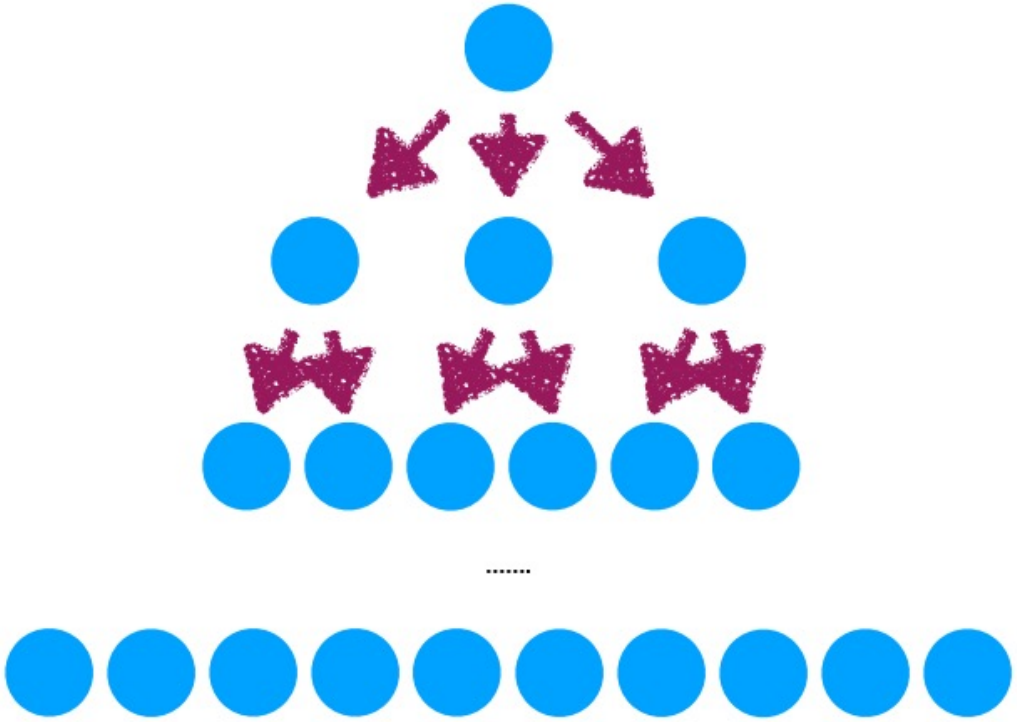
证明：

- 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



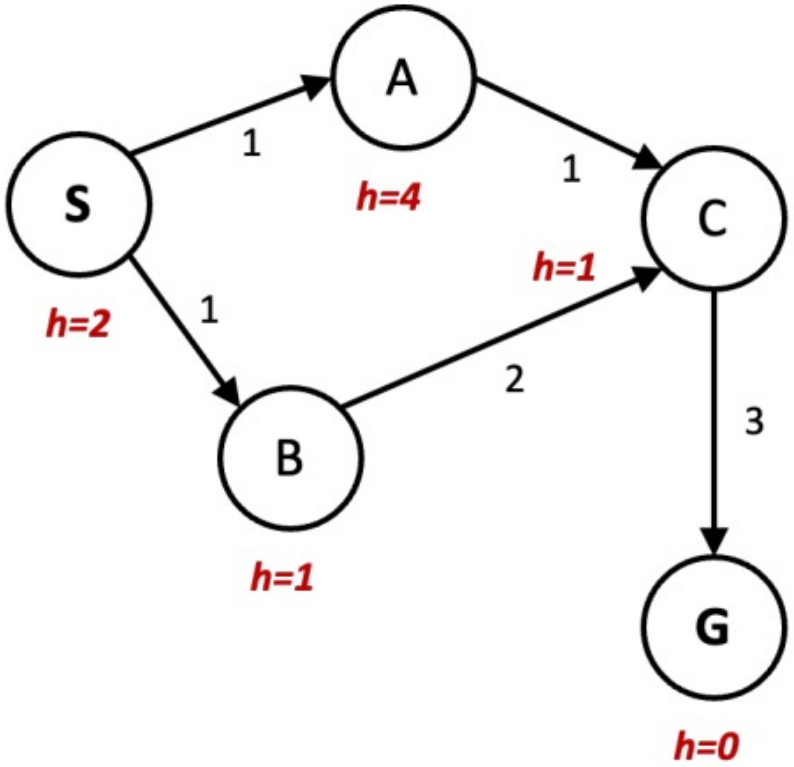
树搜索 vs 图搜索





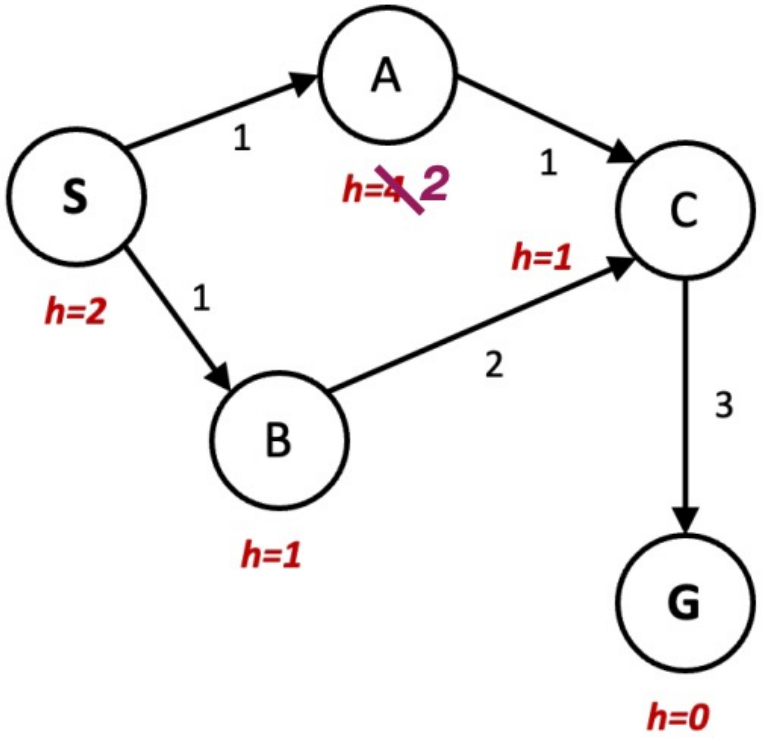
图搜索的A*算法

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

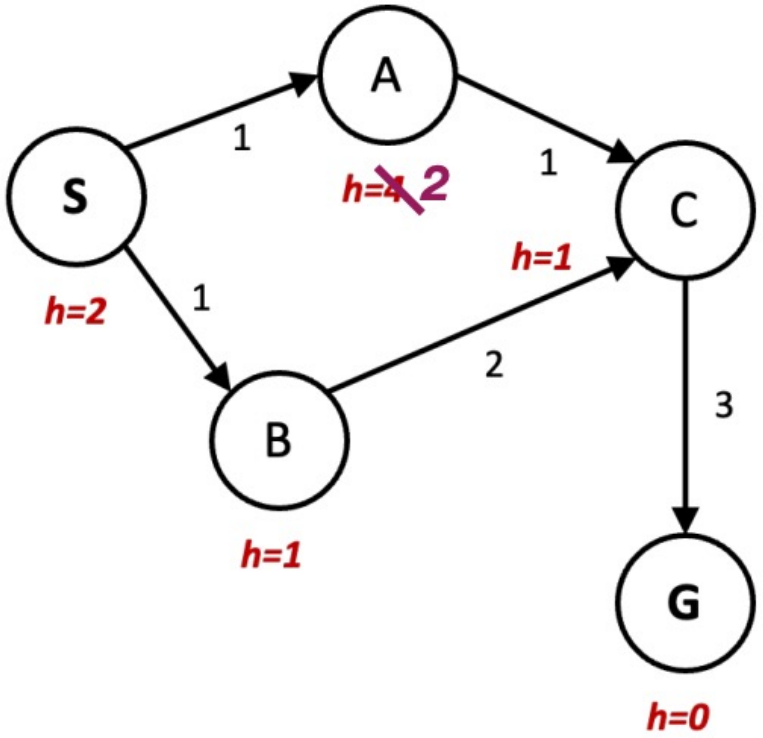
一致性条件：对于每个结点 n 和 通过任一行动a生成的n的每个后继节点n'，有$h(n) \leq cost(n, a, n') + h(n')$。



显然，由一致性条件可以推导出可采纳性

保障最优性的条件：一致性

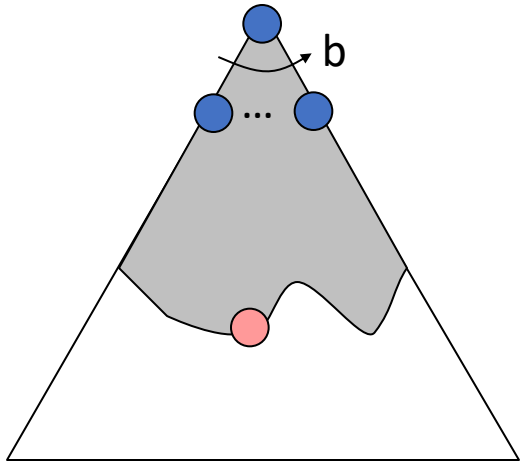
一致性条件：对于每个结点 n 和 通过任一行动a生成的n的每个后继节点n'，有$h(n) \leq cost(n, a, n') + h(n')$。



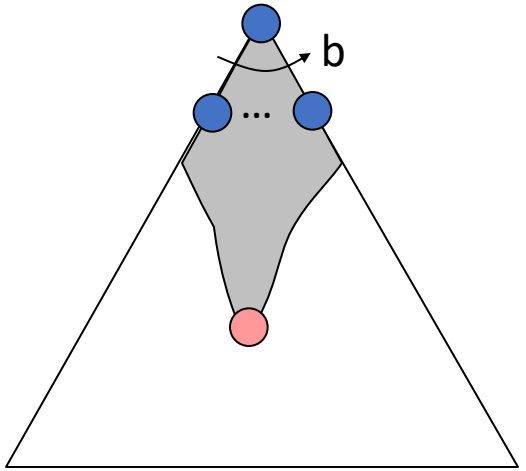
显然，由一致性条件可以推导出可采纳性

UCS VS A*

一致性代价搜索

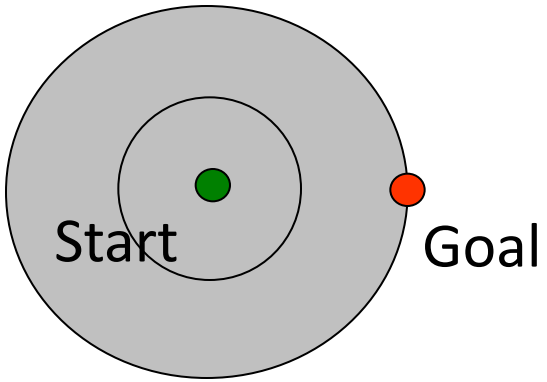


A*搜索

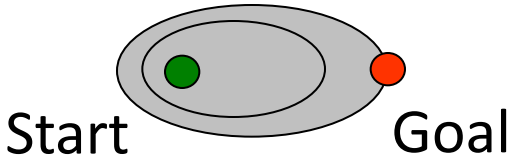


UCS VS A*

- 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

- ✓ 游戏AI
- ✓ 寻路算法
- ✓ 规划问题
- ✓ 机器人动作
- ✓ 机器翻译
- ✓ 语音识别
- ✓ ...

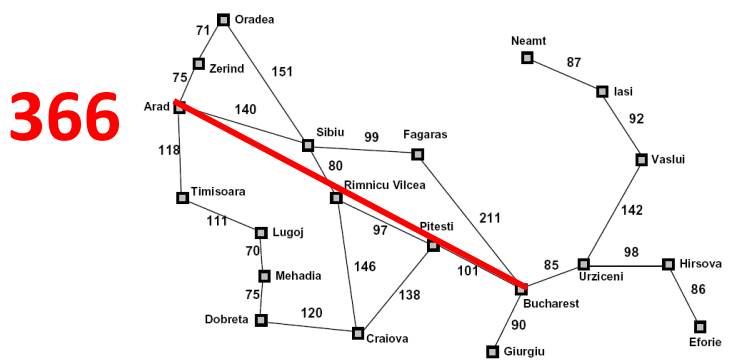
提纲

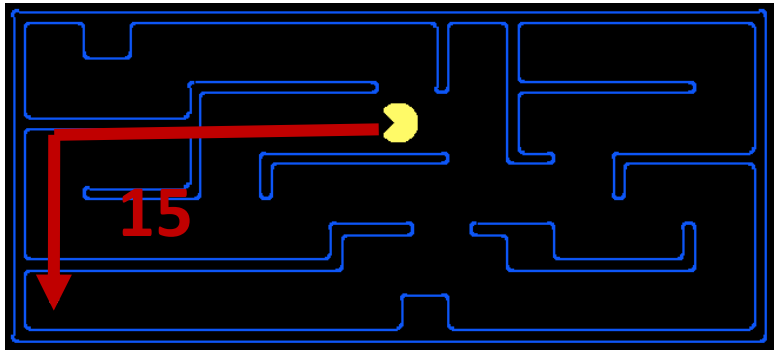
- 启发式搜索
  - 贪婪搜索
  - A*搜索
  - 启发式函数设计
- 本章小结

启发式函数（heuristic function）

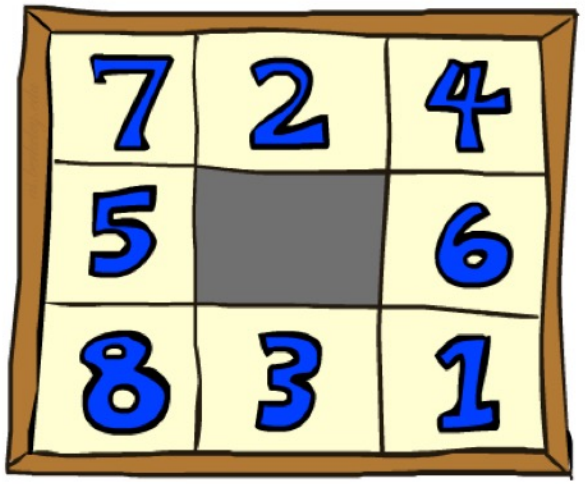
启发式搜索的关键在于如何设计启发式函数？

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

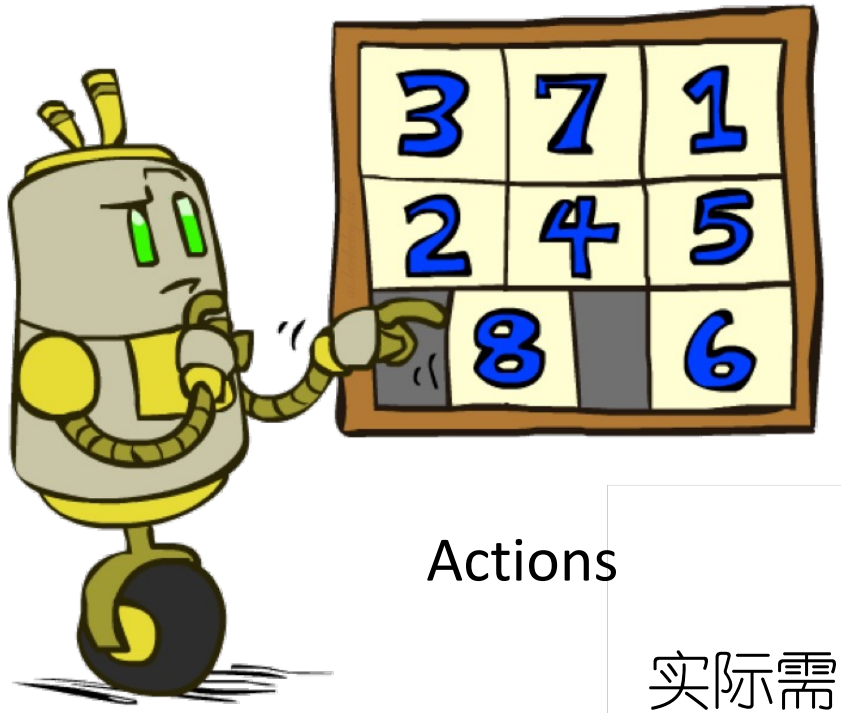




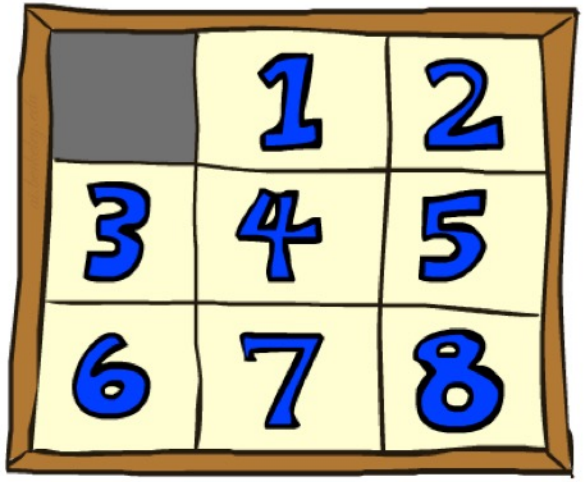
例子：八数码问题



Start State



Actions



Goal State

实际需要的步数为26步

八数码问题：

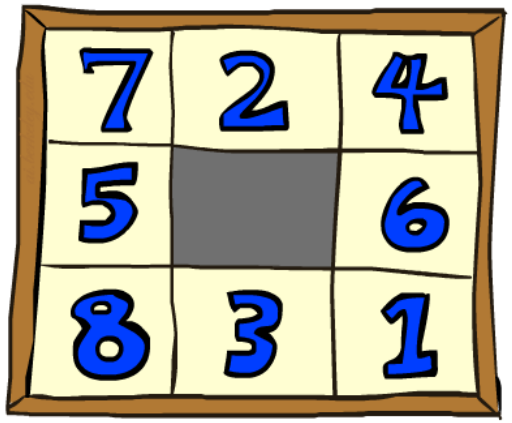
- ✓ 状态空间是什么？
- ✓ 多少种状态？
- ✓ 动作和代价？

Any idea for admissible heuristic?

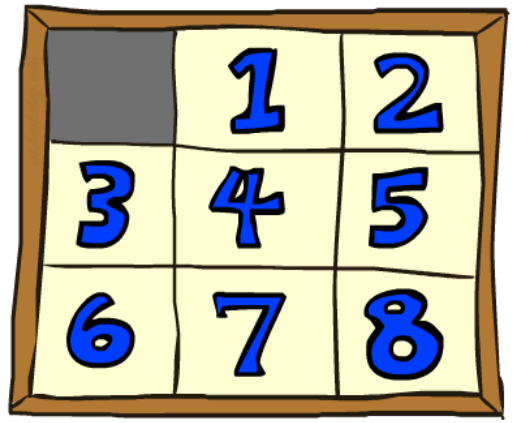
例子：八数码问题

启发式函数

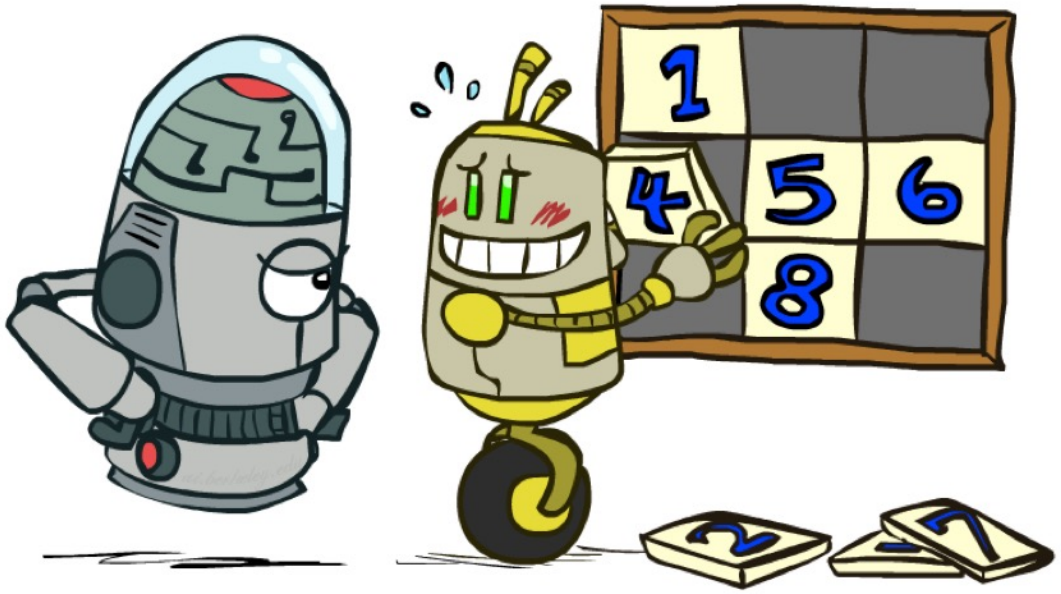
✓不在正确位置的棋子数



Start State



Goal State



$$h(s) = ?$$

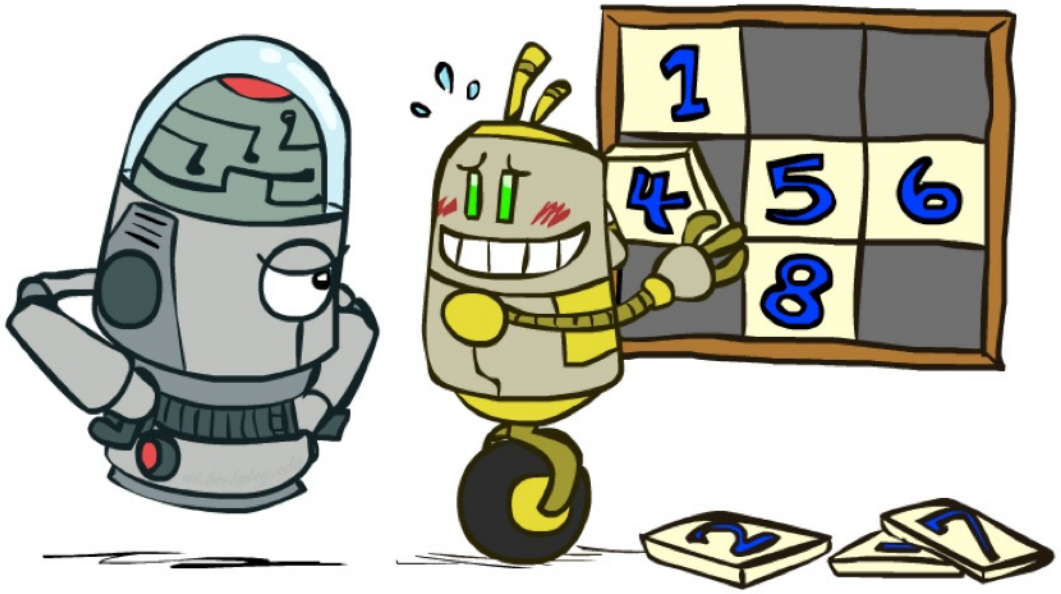
$$h(s) = 8$$

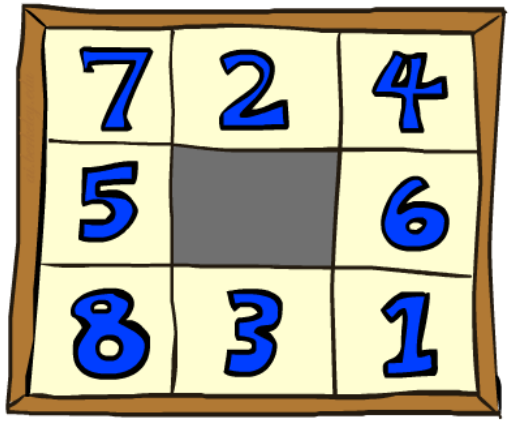
例子：八数码问题

启发式函数

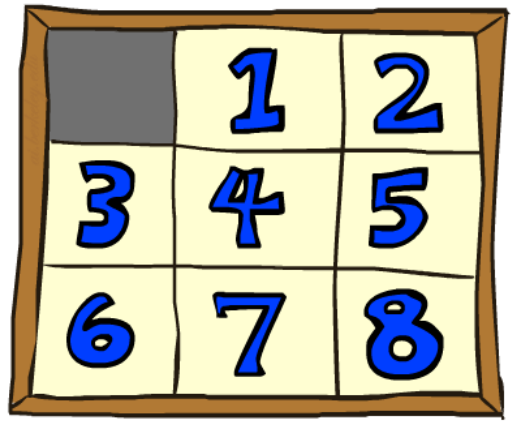
✓所有棋子到目标位置的距离和

（曼哈顿距离）





Start State



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

	Search Cost (nodes generated)			Effective Branching Factor		
d	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

本章小结

- 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