



南京大學  
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

<https://www.lamda.nju.edu.cn/guolz/IntroAI/fall2025/index.html>

Email: [guolz@nju.edu.cn](mailto:guolz@nju.edu.cn)

# 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

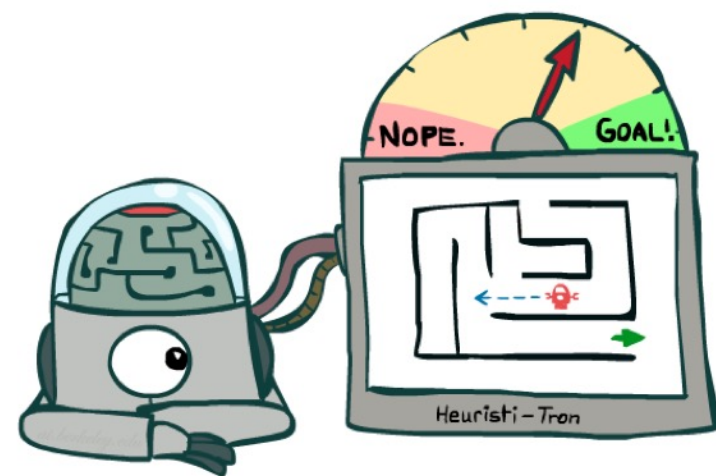
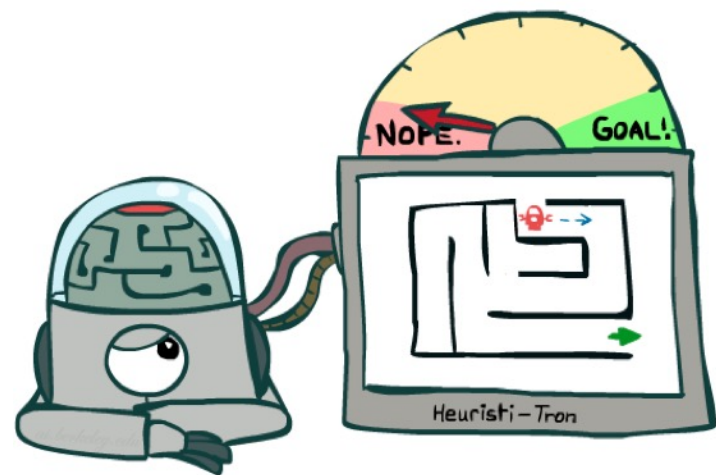
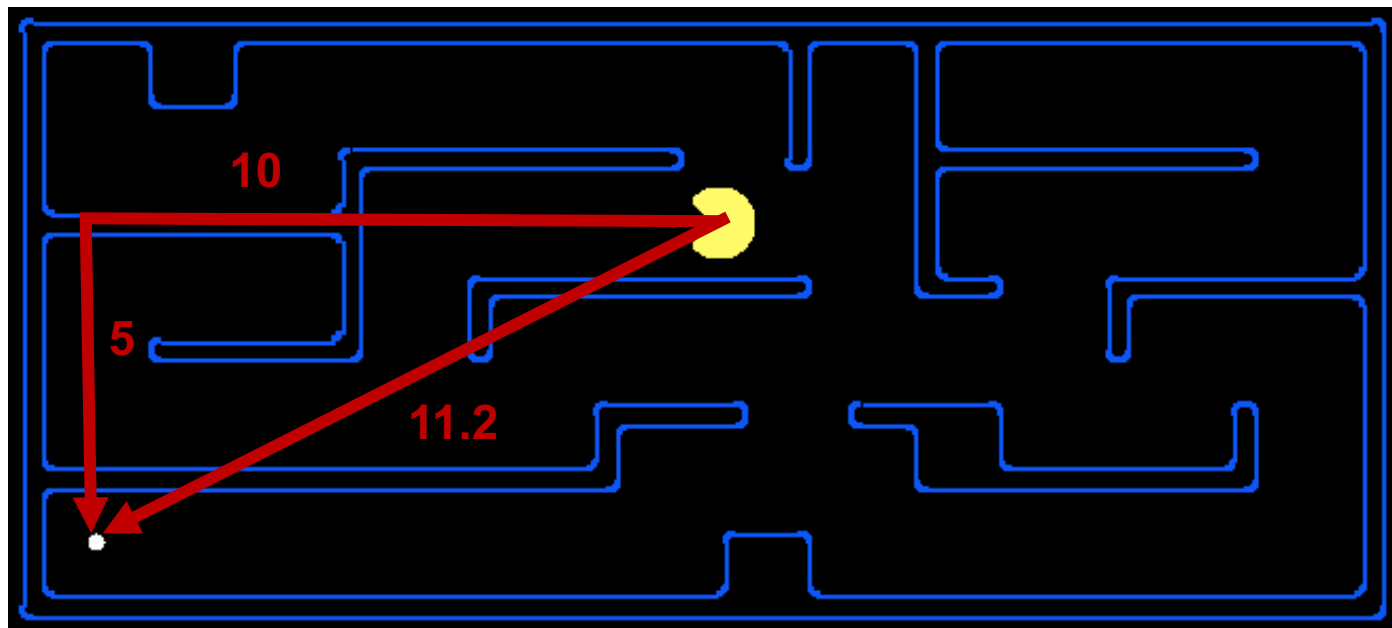
## □ 本章小结



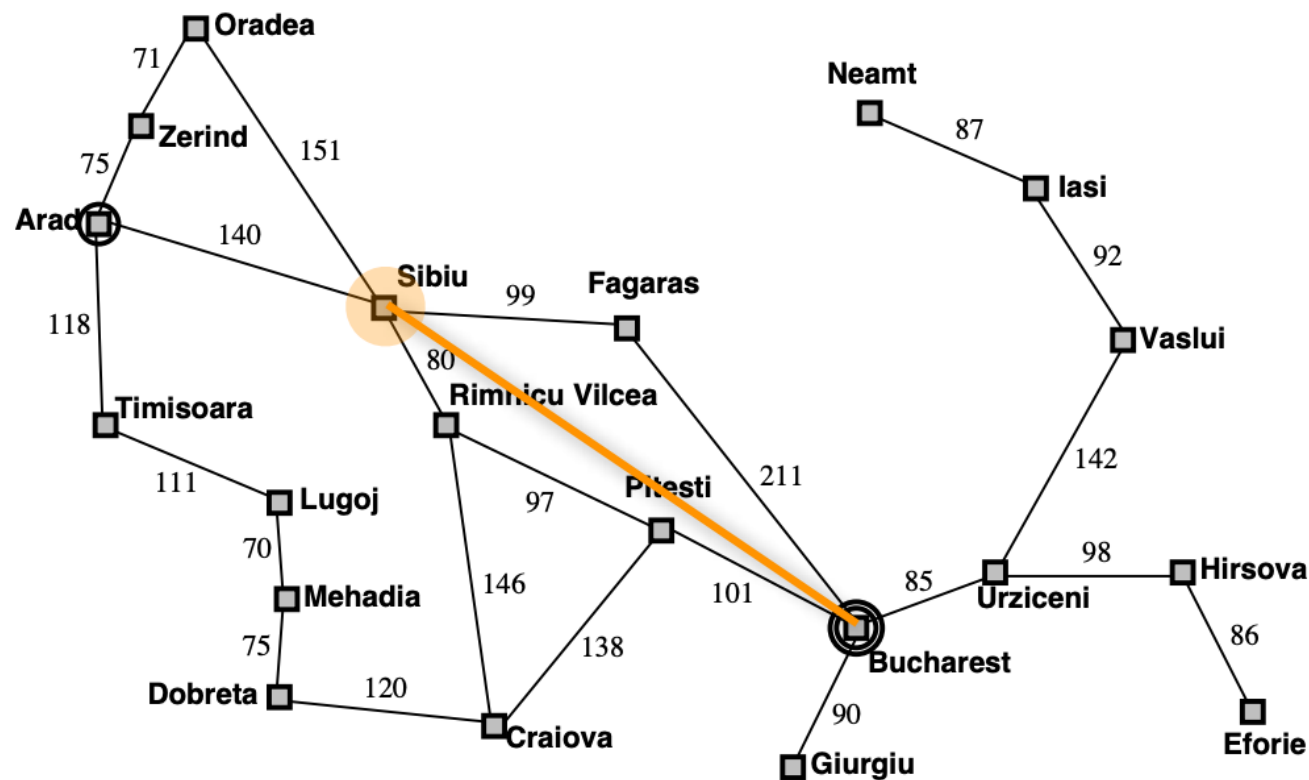
# 启发式信息 (Heuristic Information)

## □ 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**？

✓  $41 - 25 = 16$

✓  $16 - 5 * 3 = 1$

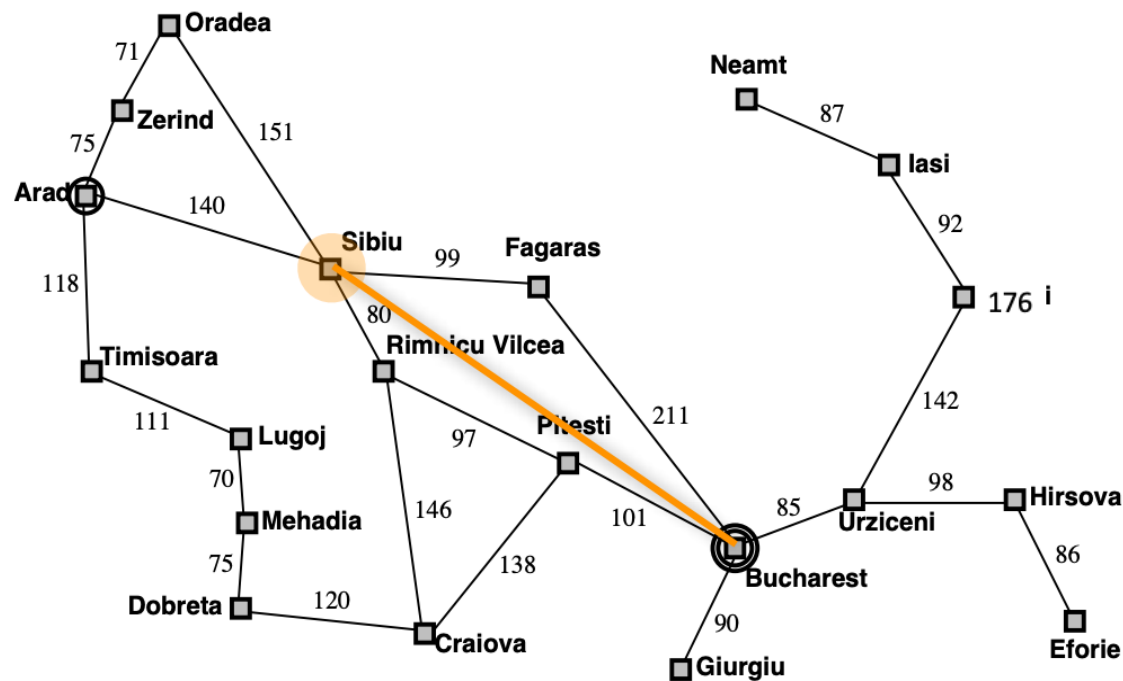
✓  $1 - 1 = 0$

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？



Straight-line distance  
to Bucharest

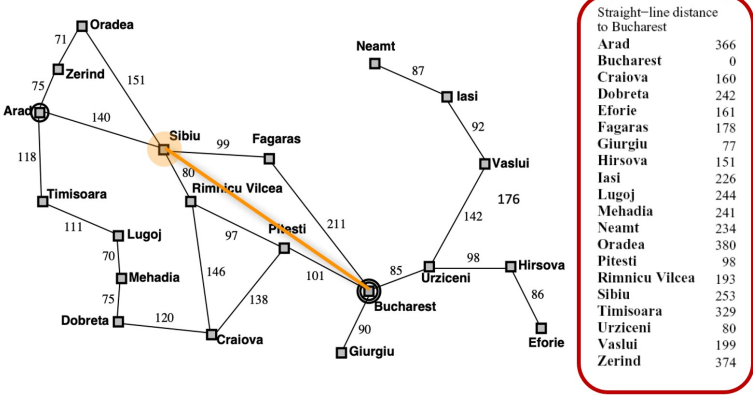
|                |     |
|----------------|-----|
| Arad           | 366 |
| Bucharest      | 0   |
| Craiova        | 160 |
| Dobreta        | 242 |
| Eforie         | 161 |
| Fagaras        | 178 |
| Giurgiu        | 77  |
| Hirsova        | 151 |
| Iasi           | 226 |
| Lugoj          | 244 |
| Mehadia        | 241 |
| Neamt          | 234 |
| Oradea         | 380 |
| Pitesti        | 98  |
| Rimnicu Vilcea | 193 |
| Sibiu          | 253 |
| Timisoara      | 329 |
| Urziceni       | 80  |
| Vaslui         | 199 |
| Zerind         | 374 |

$h(x)$

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

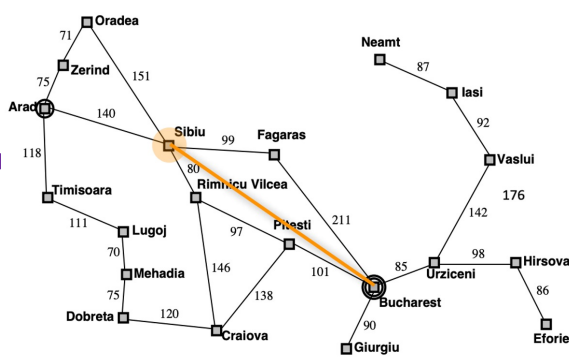
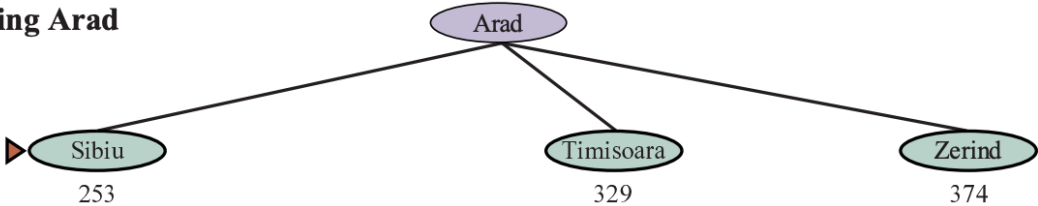


# 贪婪搜索 (greedy search)

(a) The initial state

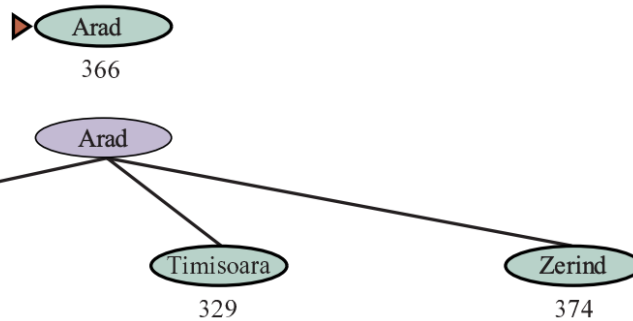


(b) After expanding Arad

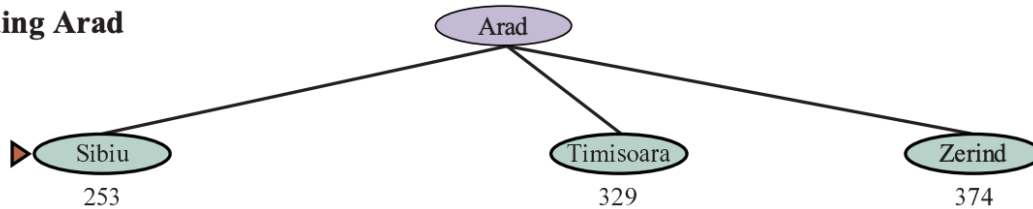


| Straight-line distance to Bucharest |     |
|-------------------------------------|-----|
| Arad                                | 366 |
| Bucharest                           | 0   |
| Craiova                             | 160 |
| Dobreta                             | 242 |
| Eforie                              | 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 |

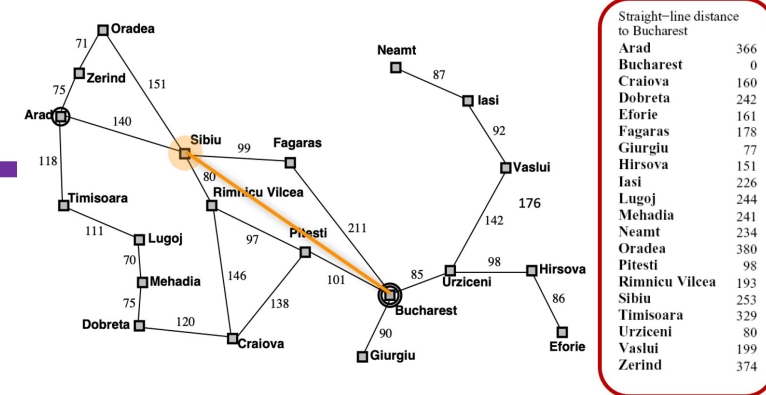
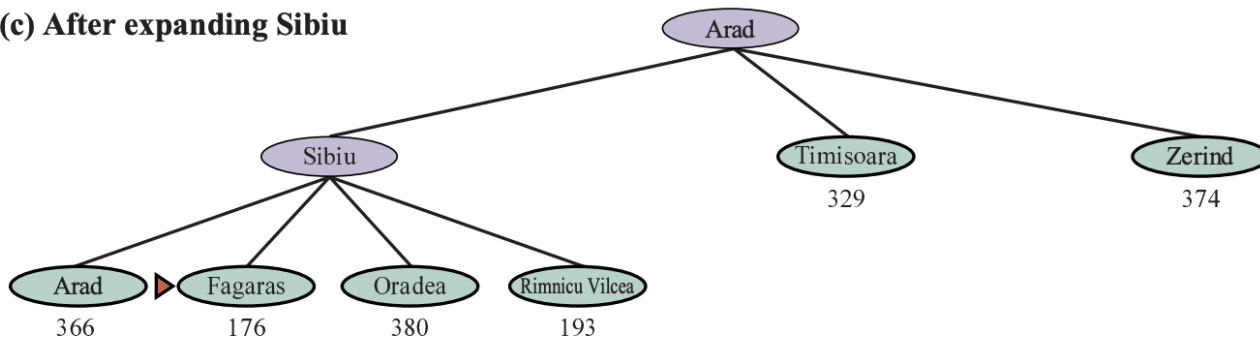
# 贪婪搜索 (greedy search)



**(b) After expanding Arad**



**(c) After expanding Sibiu**

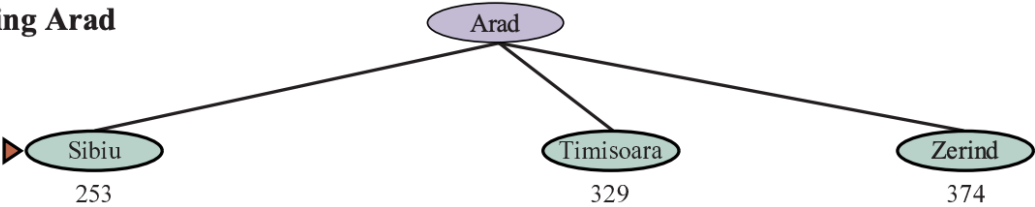


# 贪婪搜索 (greedy search)

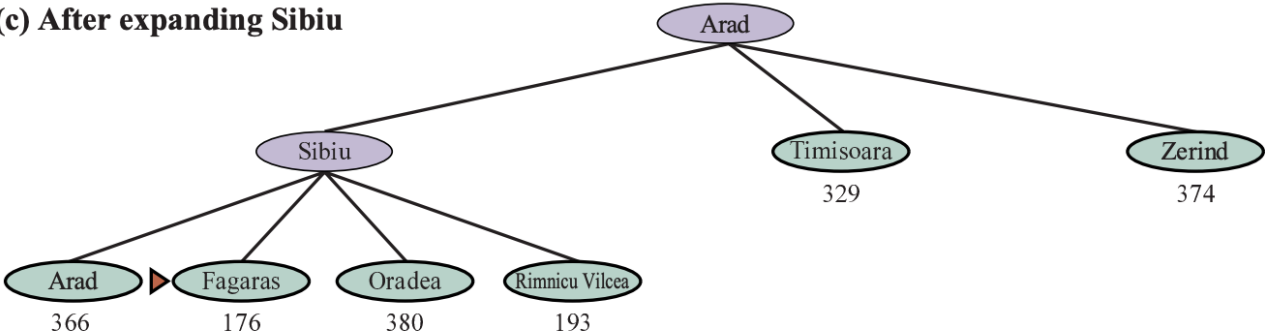
(a) The initial state



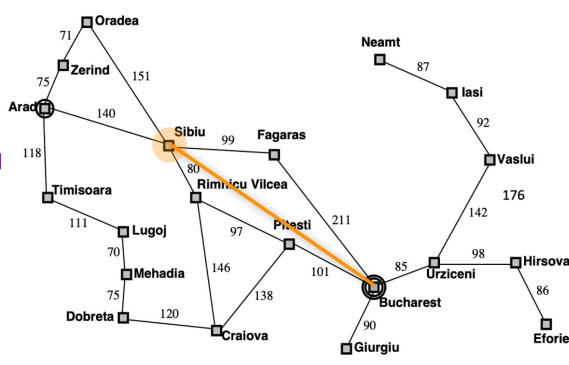
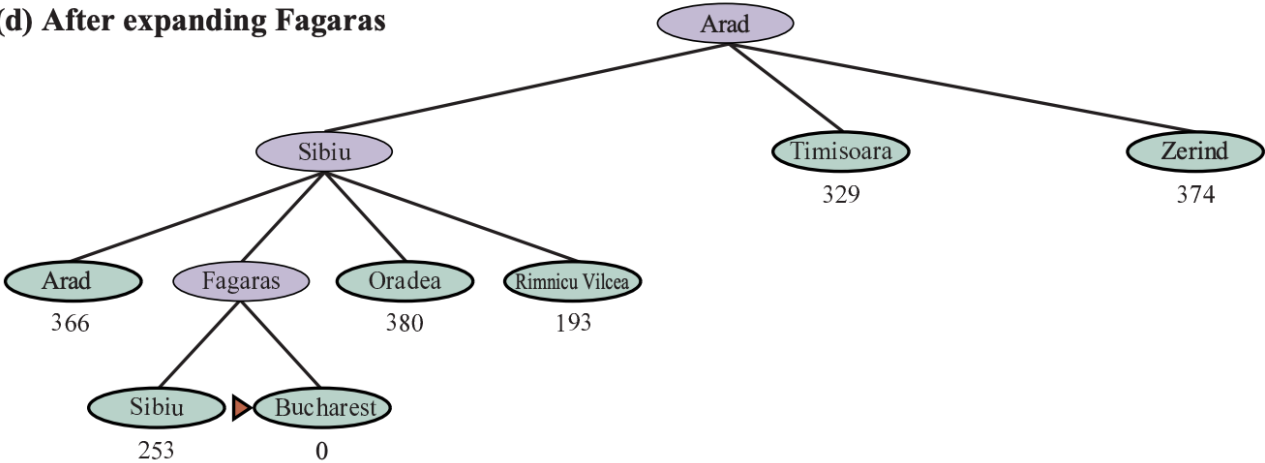
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



| 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)

□ 策略：每一步均扩展距离目标状态更近的节点

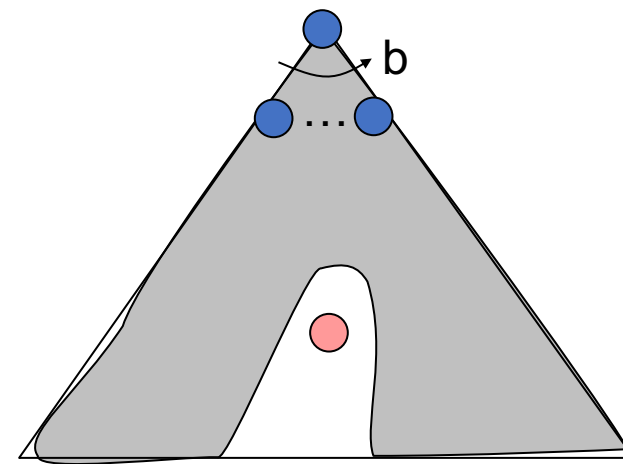
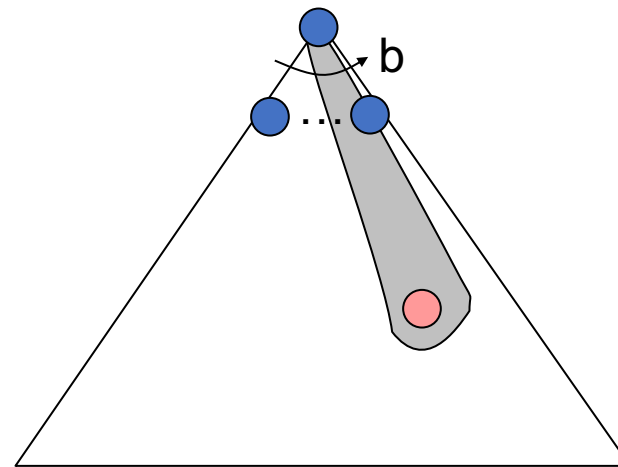
✓ 启发式信息：估计当前状态距离目标状态的距离

□ Common case:

✓ 距离目标越来越近

□ Bad case:

✓ 非完备、非最优、依赖启发式函数



# Recall

## □无信息搜索

- ✓ 宽度优先搜索：优先扩展浅层节点，逐层搜索，先入先出
- ✓ 深度优先：优先扩展深层节点，后入先出
- ✓ 迭代加深的深度优先搜索：限制搜索深度，防止陷入无穷深分支
- ✓ 一致性代价搜索：优先扩展从起点到当前节点，累计代价低的节点

## □有信息搜索：有关于目标状态的信息，例如距离估计等

- ✓ 贪心搜索：优先扩展距离目标状态近的节点

# 提纲

## □ 启发式搜索

➤ 贪婪搜索

➤ A\*搜索

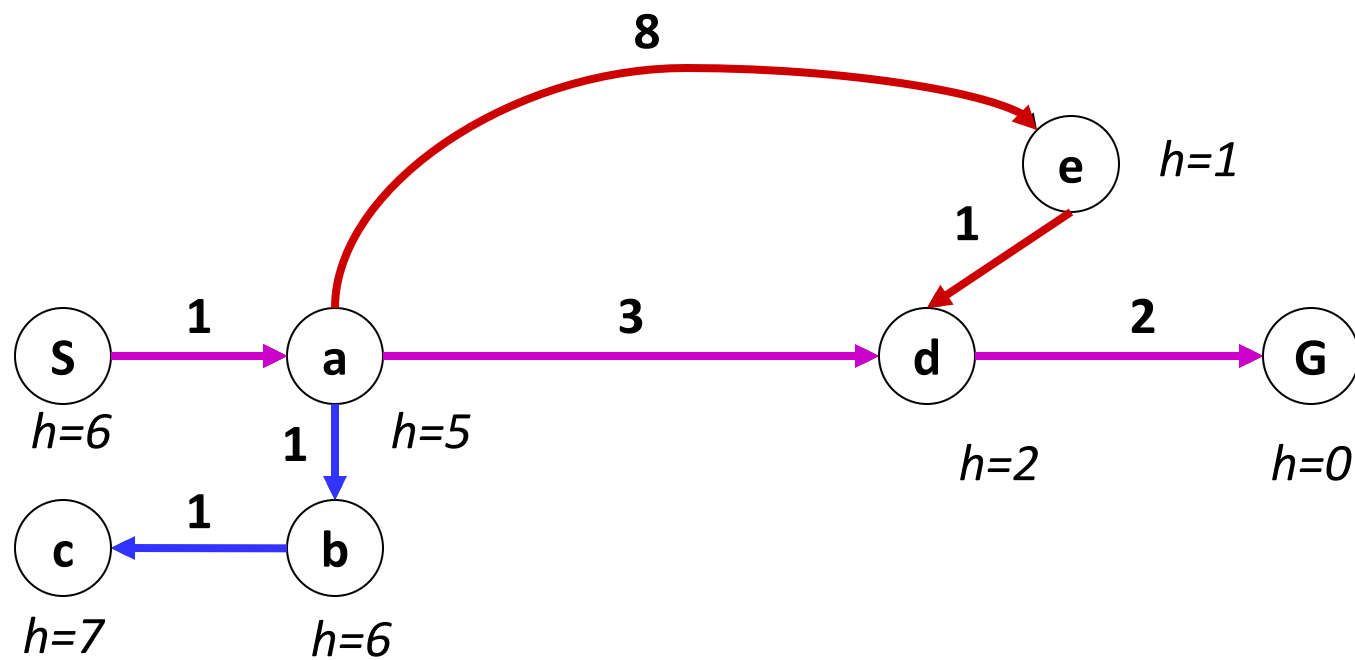
➤ 启发式函数

## □ 本章小结



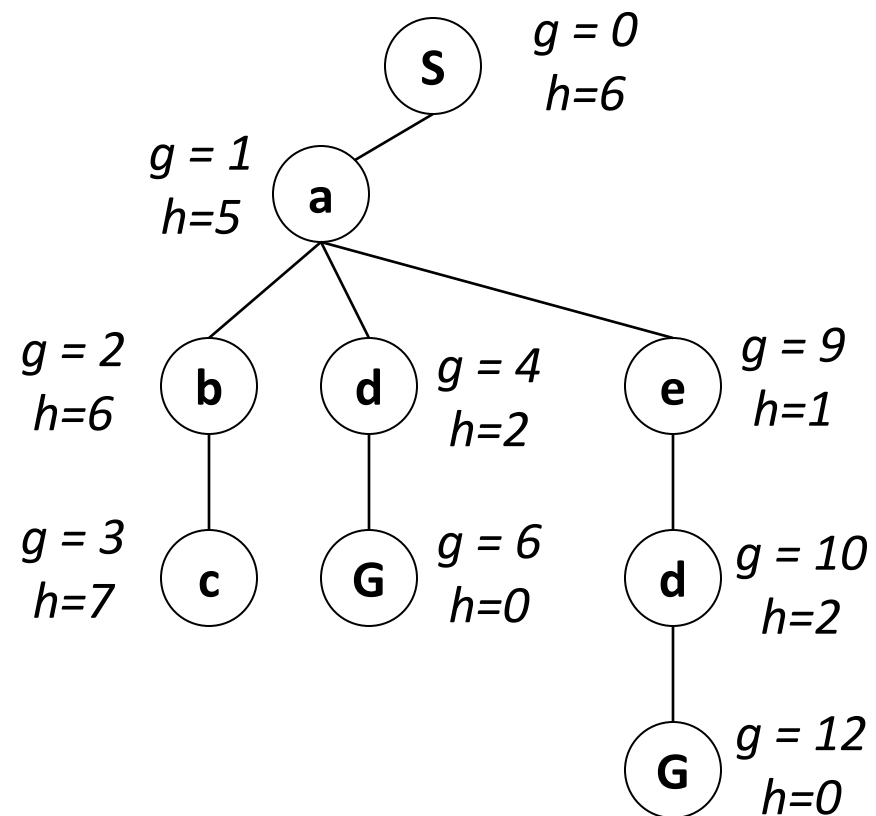
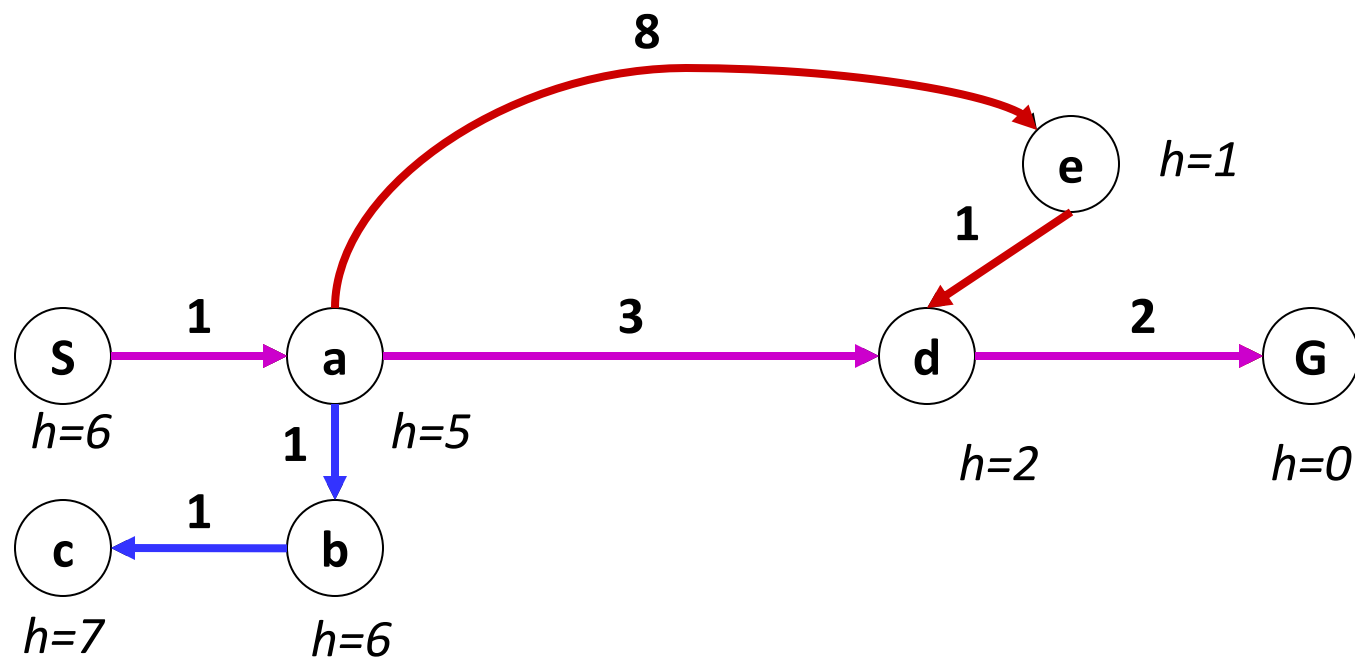
# A\*算法

- 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



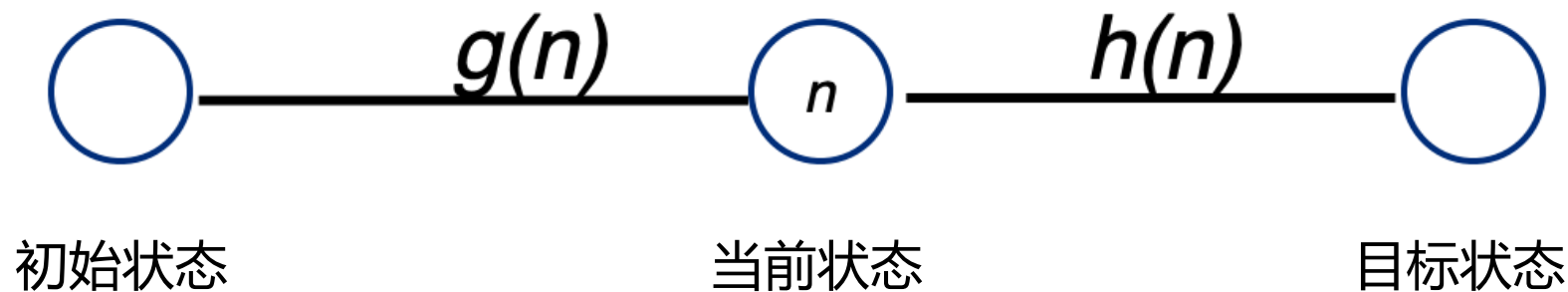
# A\*算法

- 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



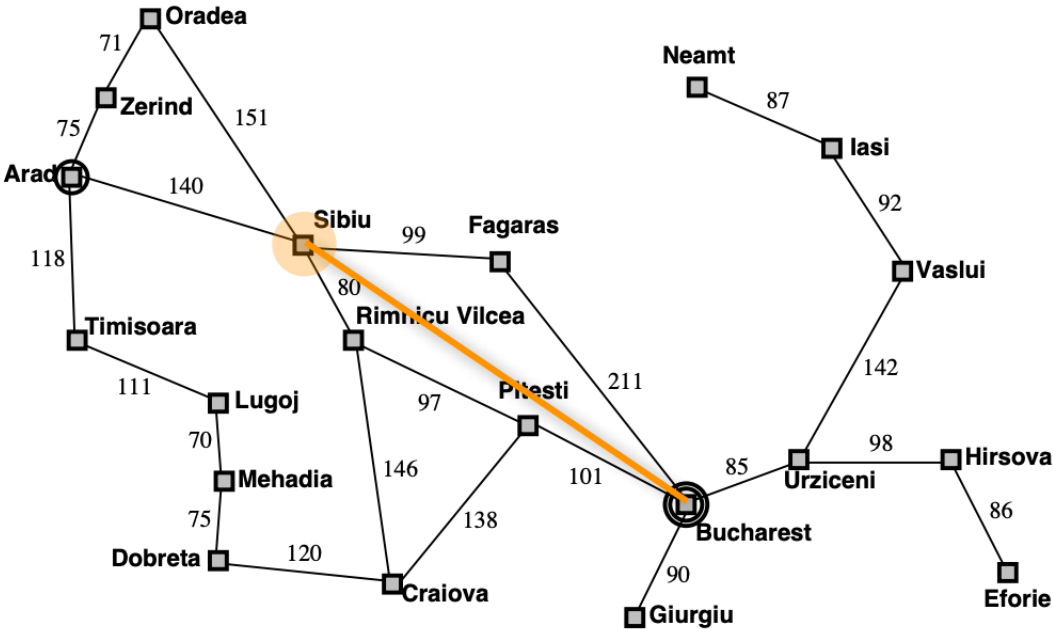
# A\*算法

**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



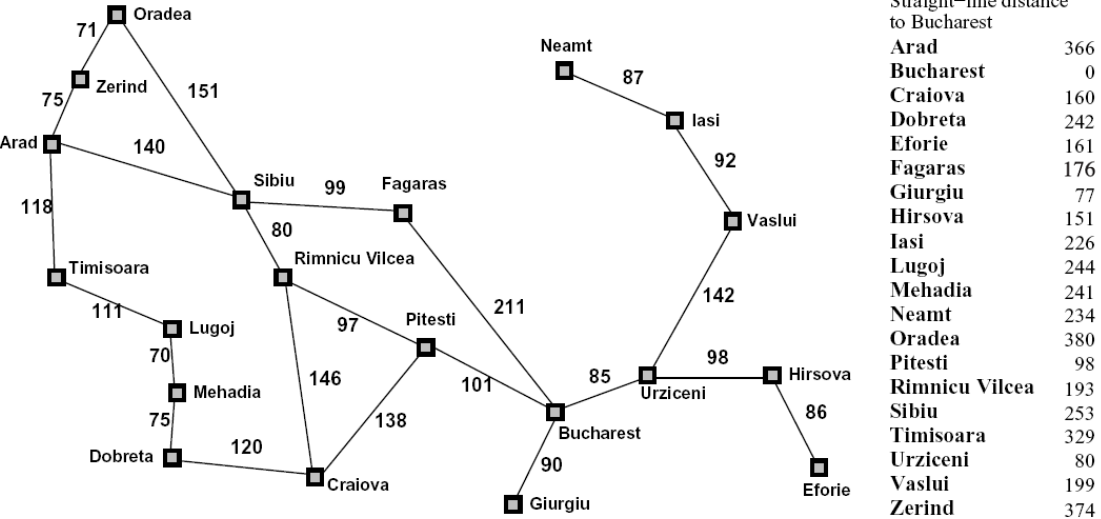
|           |     |                |     |
|-----------|-----|----------------|-----|
| Arad      | 366 | Mehadia        | 241 |
| Bucharest | 0   | Neamt          | 234 |
| Craiova   | 160 | Oradea         | 380 |
| Drobeta   | 242 | Pitesti        | 100 |
| Eforie    | 161 | Rimnicu Vilcea | 193 |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

# A\*算法

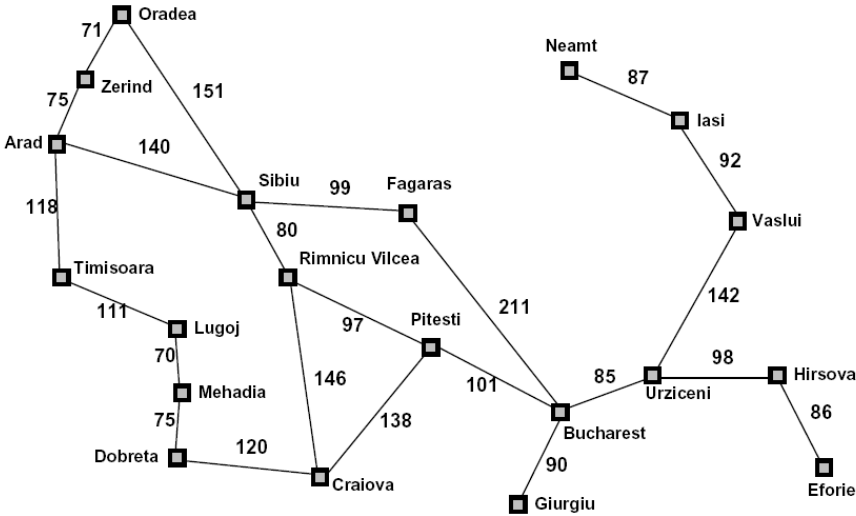


Arad

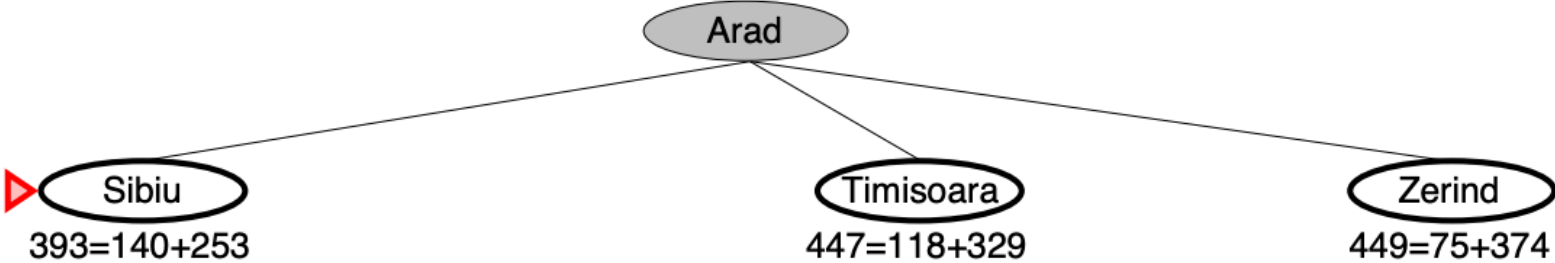
366=0+366



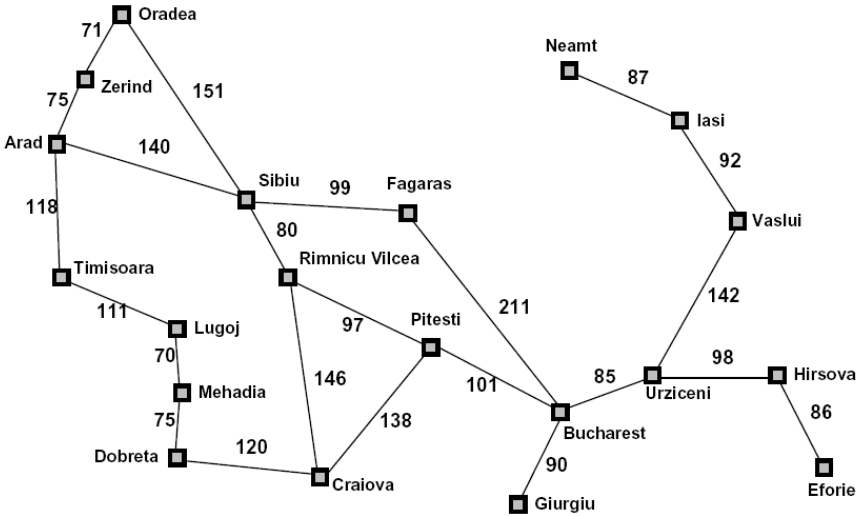
# A\*算法



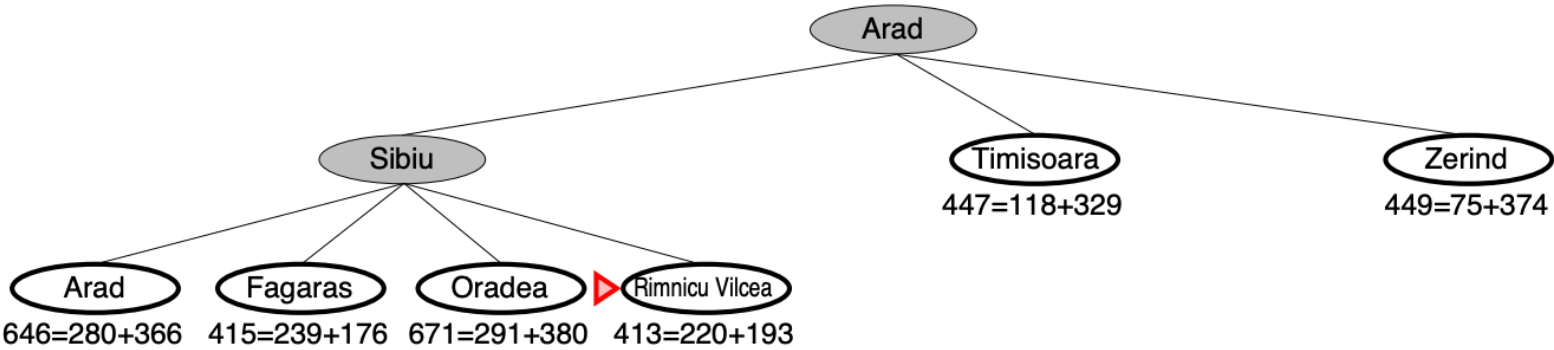
| Straight-line distance to Bucharest |     |
|-------------------------------------|-----|
| Arad                                | 366 |
| Bucharest                           | 0   |
| Craiova                             | 160 |
| Dobreta                             | 242 |
| Eforie                              | 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 |



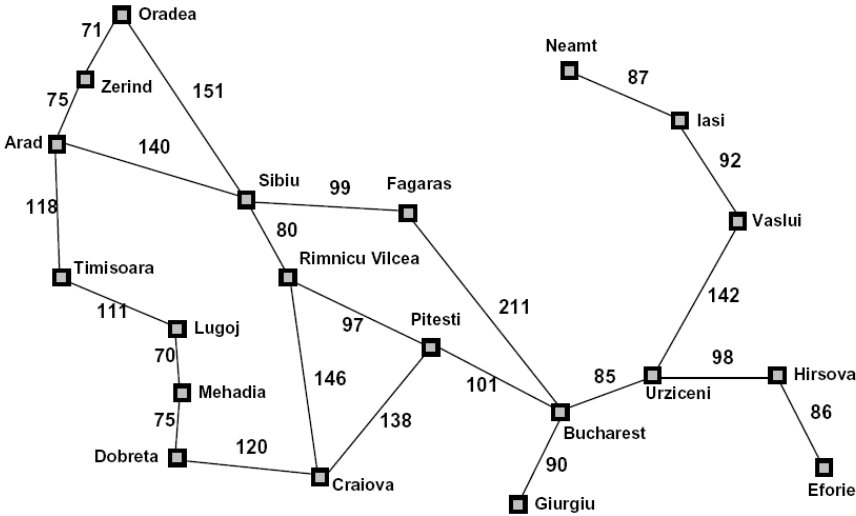
# A\*算法



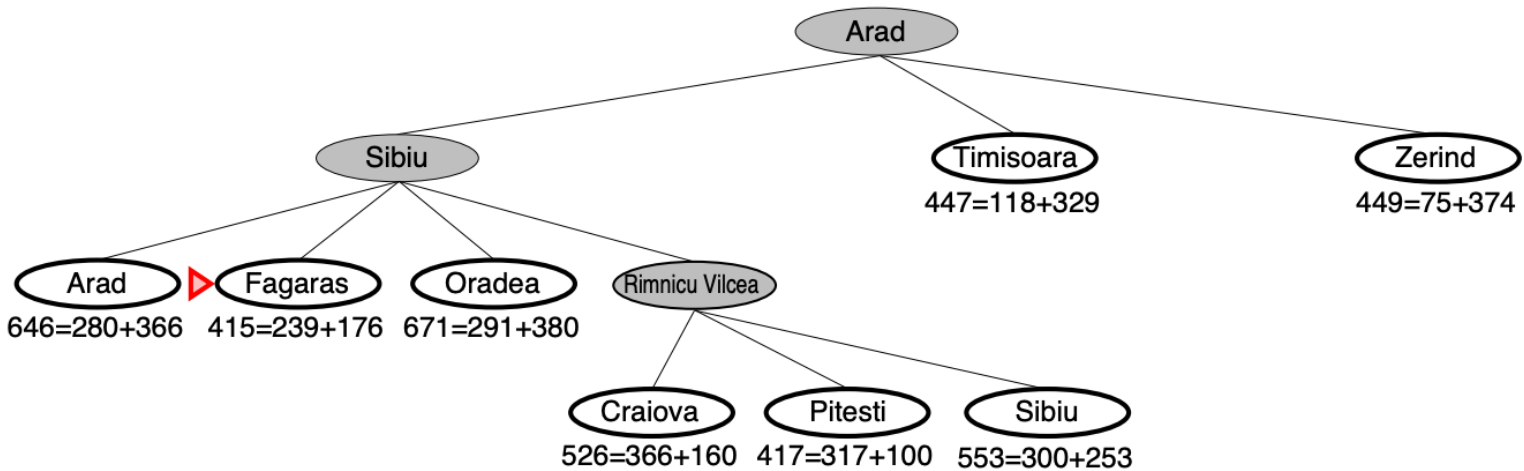
| Straight-line distance to Bucharest |     |
|-------------------------------------|-----|
| Arad                                | 366 |
| Bucharest                           | 0   |
| Craiova                             | 160 |
| Dobreta                             | 242 |
| Eforie                              | 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 |



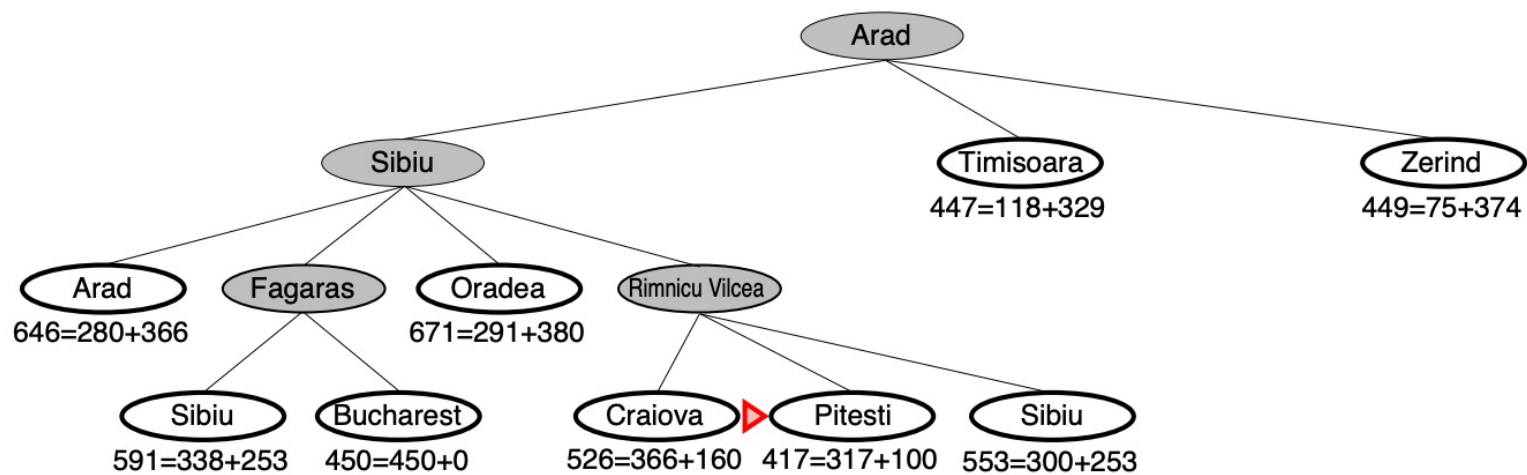
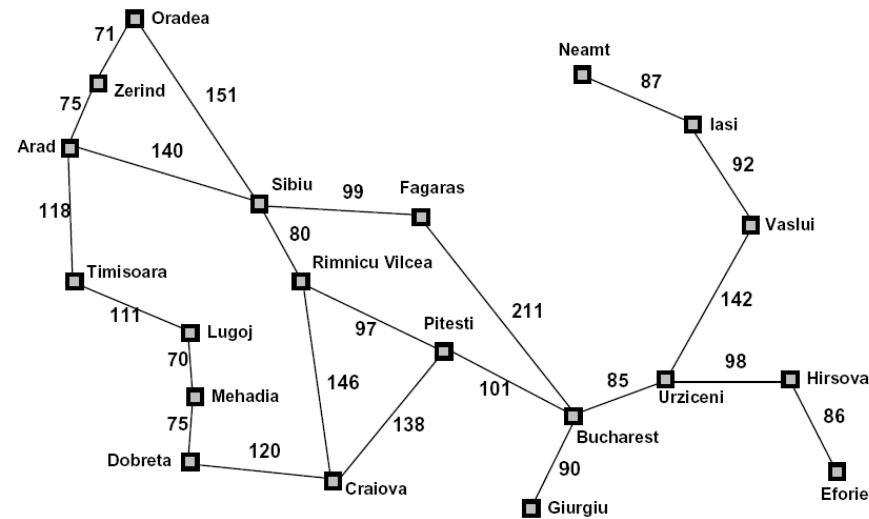
# A\*算法



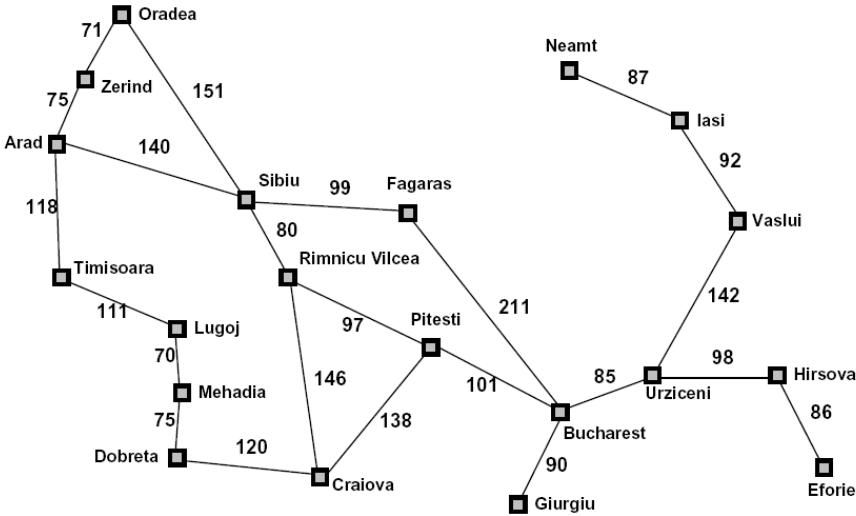
| Straight-line distance<br>to Bucharest |     |
|----------------------------------------|-----|
| Arad                                   | 366 |
| Bucharest                              | 0   |
| Craiova                                | 160 |
| Dobreta                                | 242 |
| Eforie                                 | 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 |



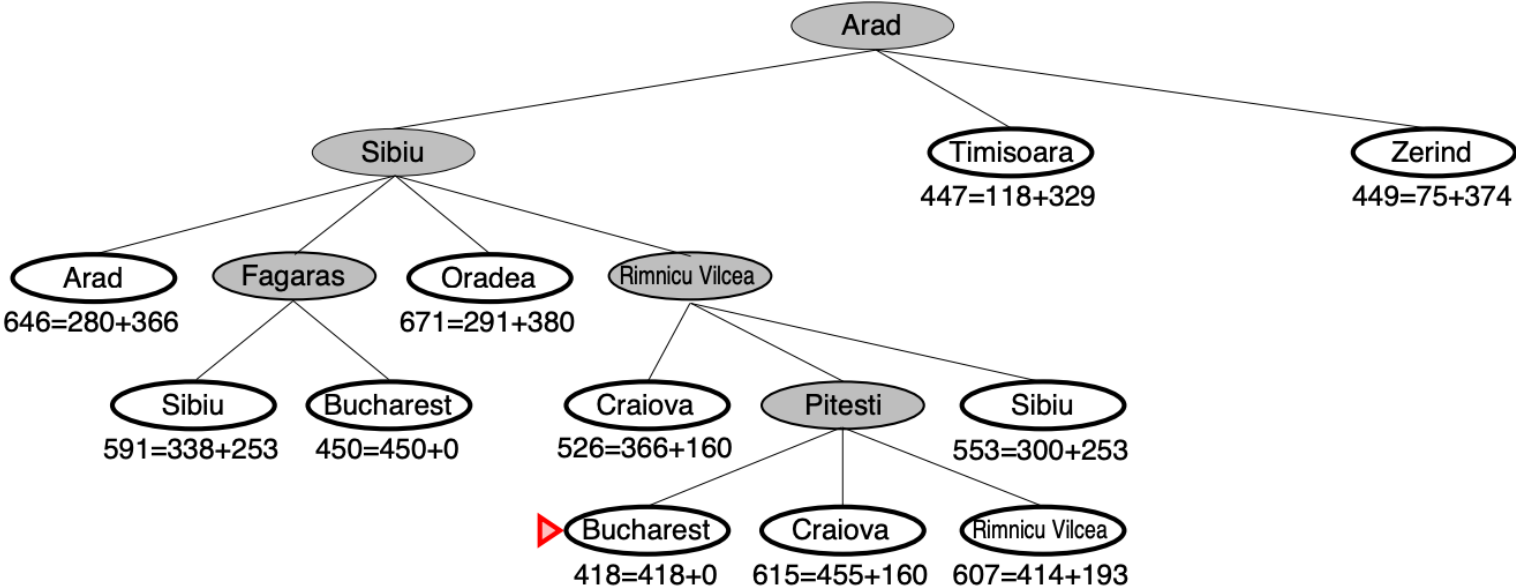
# A\*算法



# 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

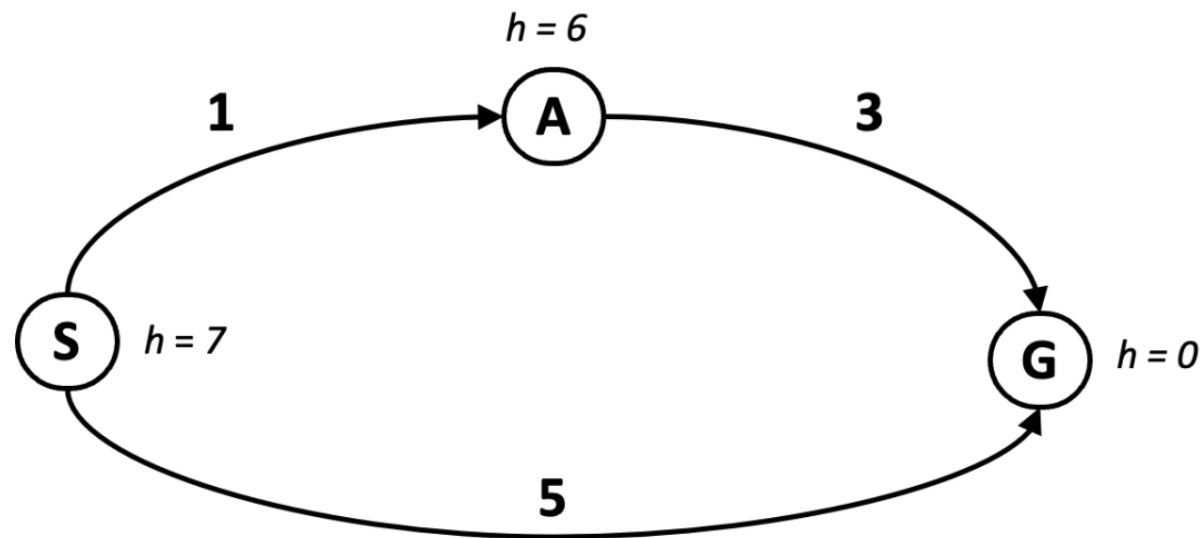
在一些常见的搜索问题中，状态数量是有限的，此时图搜索A\*算法和排除环路的树搜索A\*算法均是完备的，即一定能够找到一个解。

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

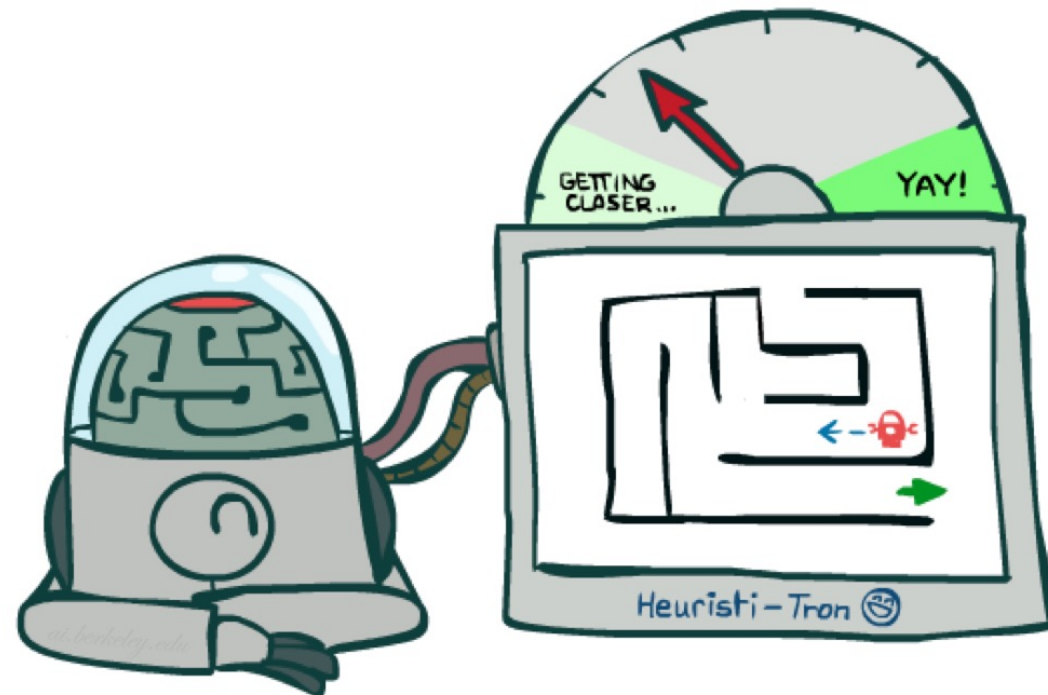
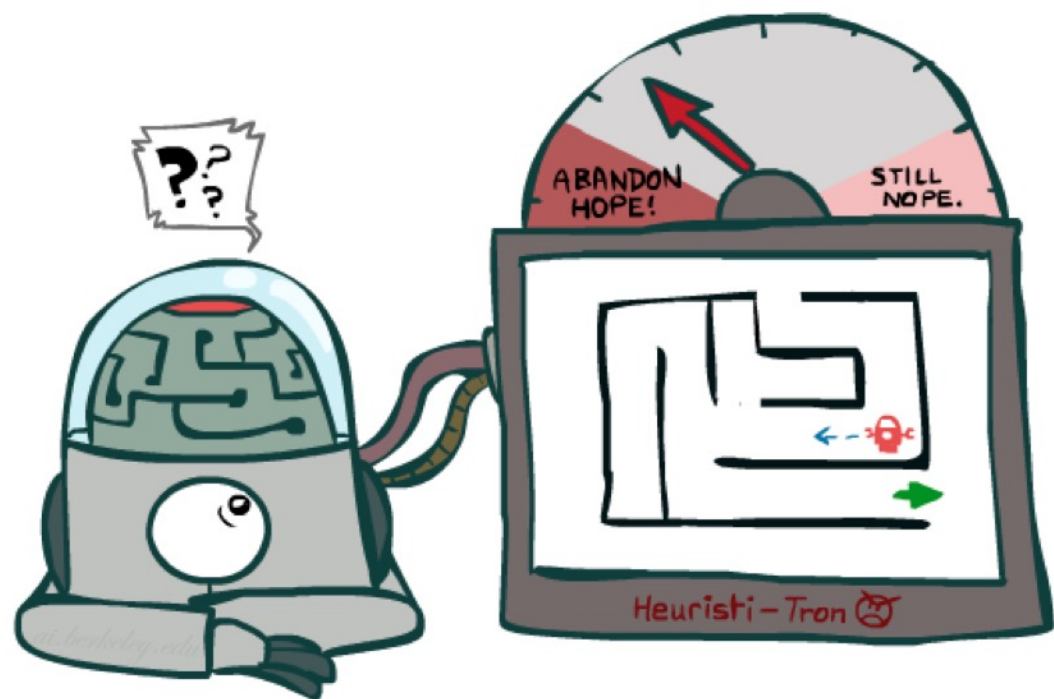
# A\*算法的最优性

A\*算法能否找到最佳路线?

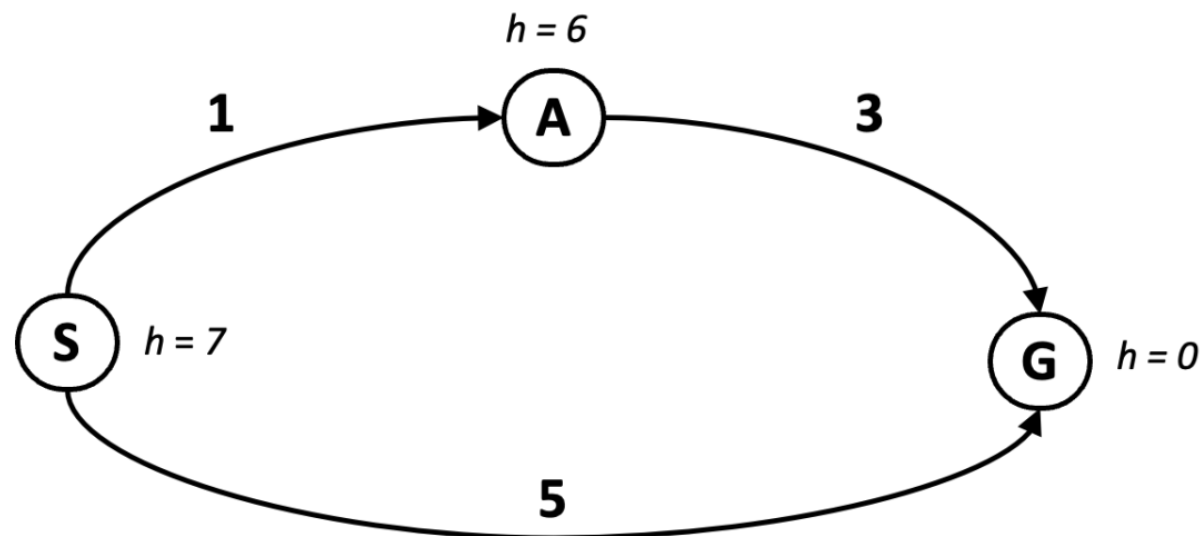


Quiz: 如何修改启发式信息可以使算法找到最优解?

# A\*算法的最优性



# A\*算法的最优性

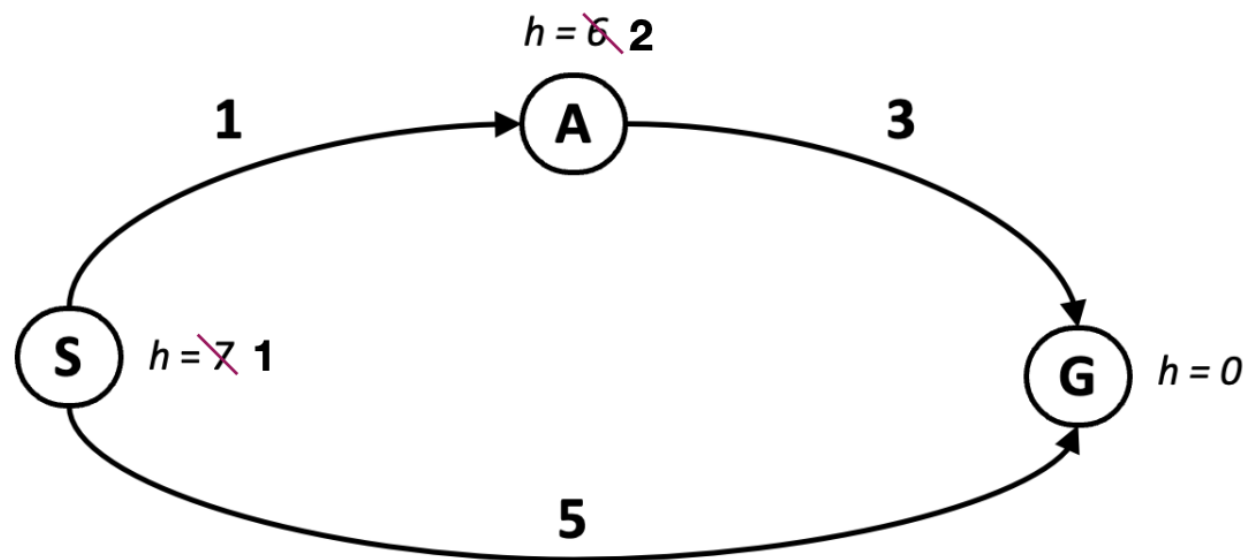


显然，为了保障A\*算法的最优性，需要额外假设  
不应该过高估计 $h(n)$ 从而导致好的节点被忽略

# 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

---

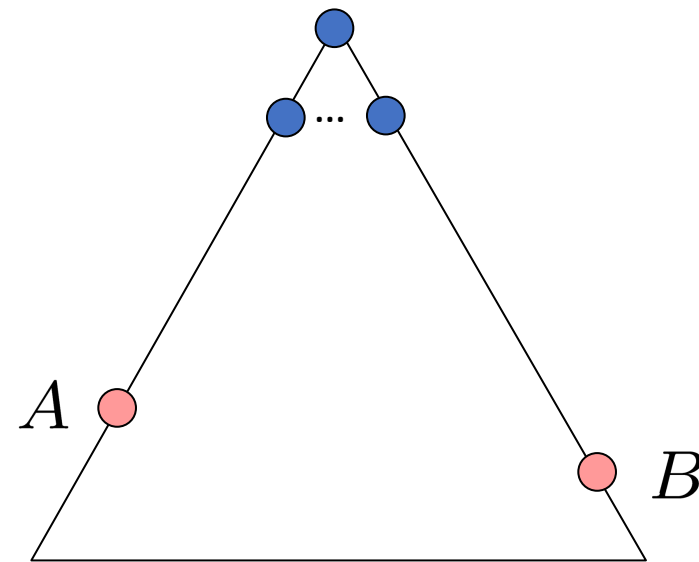
**定理：**

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：

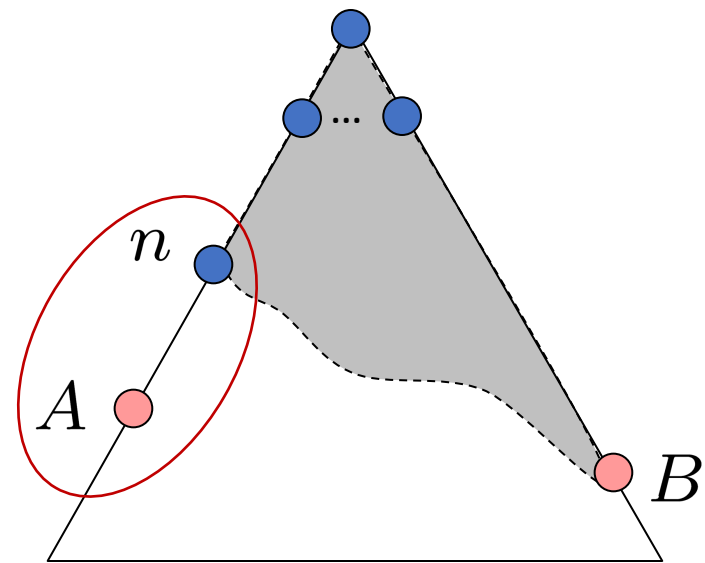
- 假设  $B$  位于边缘集
- $A$  的某个父节点  $n$  也在边缘集
- 那么,  $n$  比  $B$  更早被扩展

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

$$f(n) = g(n) + h(n)$$

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

$$g(A) = f(A)$$



# 保障最优性的条件：可采纳性

证明：

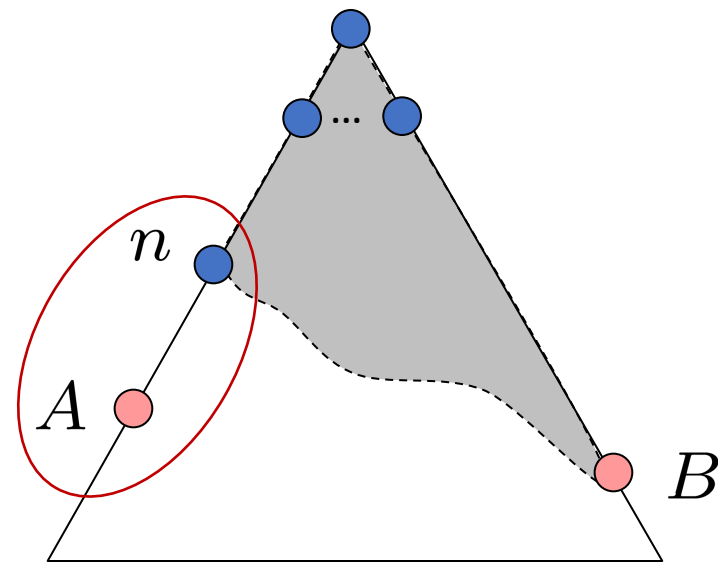
- 假设  $B$  位于边缘集
- $A$  的某个父节点  $n$  也在边缘集
- 那么,  $n$  比  $B$  更早被扩展

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

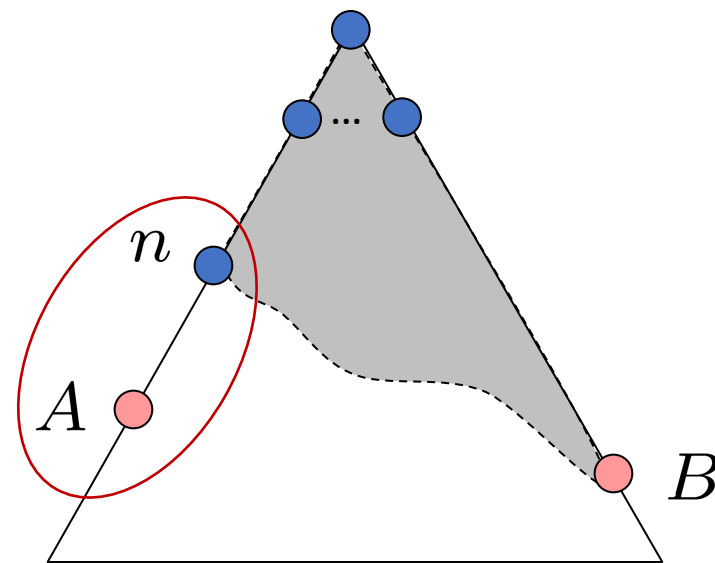
证明：

- 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：

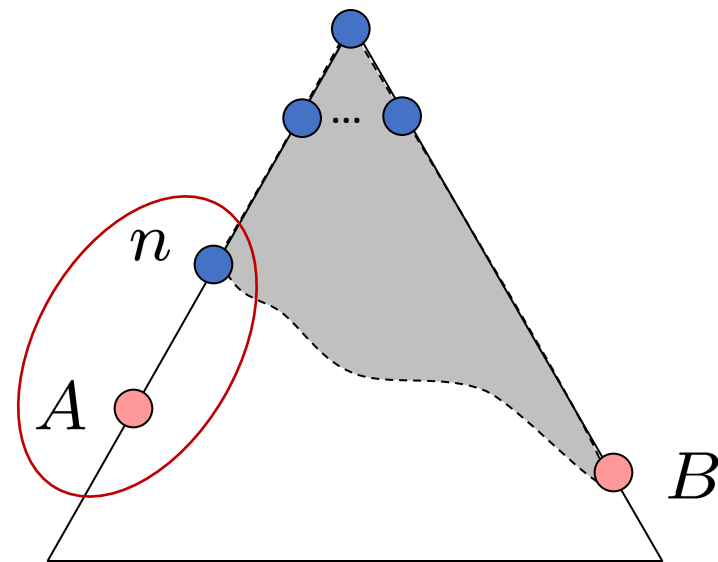
- 假设  $B$  位于边缘集
- $A$  的某个父节点  $n$  也在边缘集
- 那么,  $n$  比  $B$  更早被扩展

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

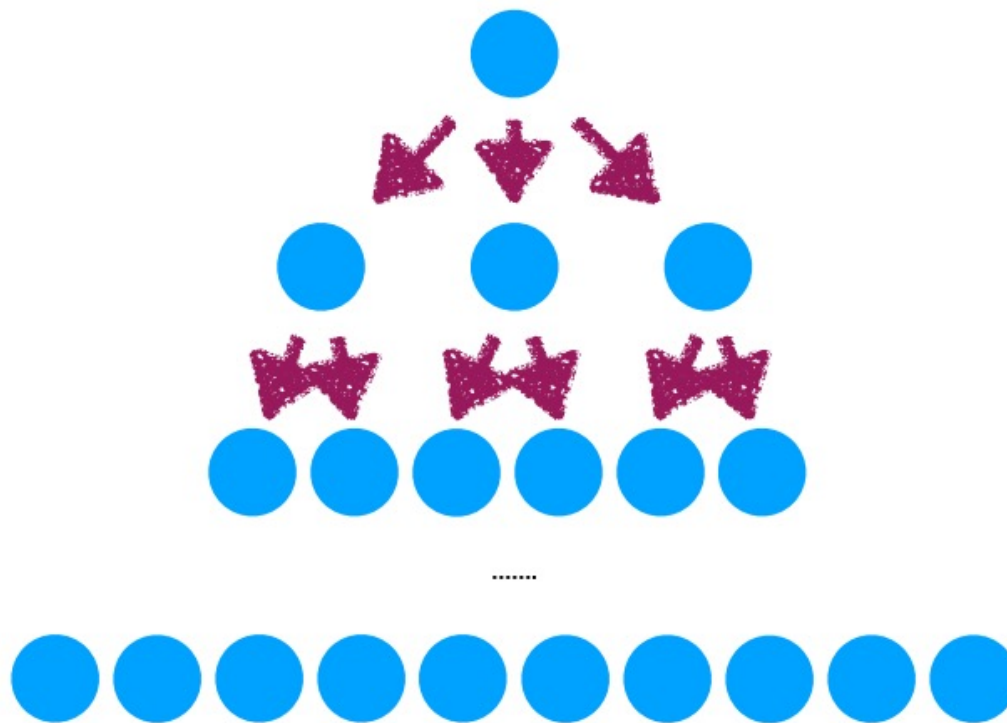
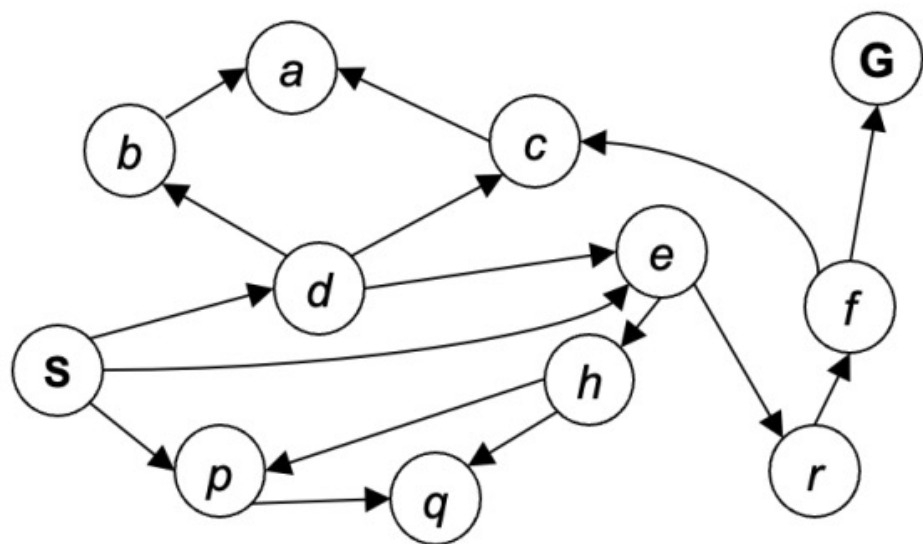
$$2. f(A) \leq f(B)$$

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

- 所有节点  $n$  都比  $B$  更早被扩展，所以  $A$  比  $B$  更早被扩展

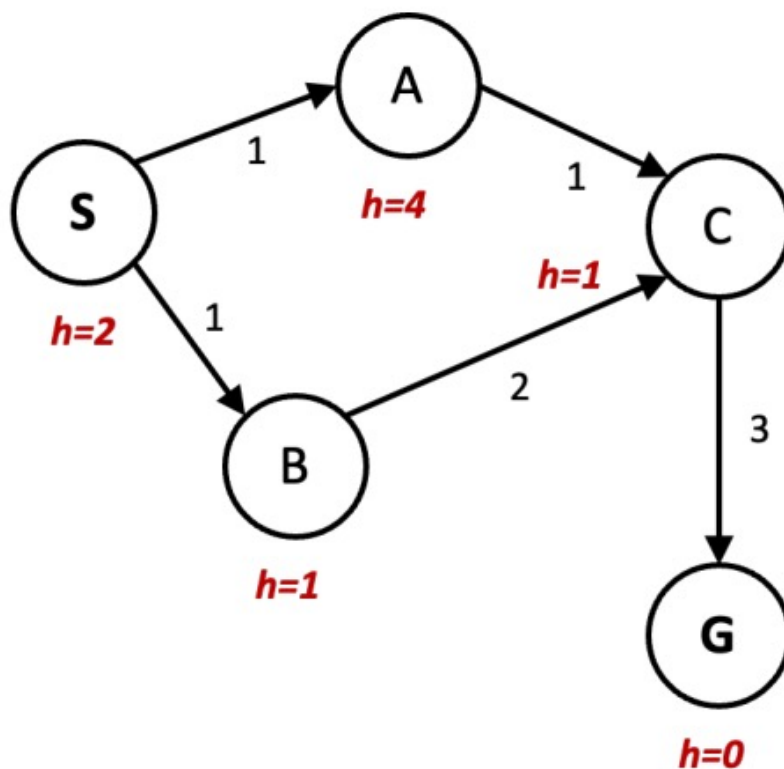


# 树搜索 vs 图搜索



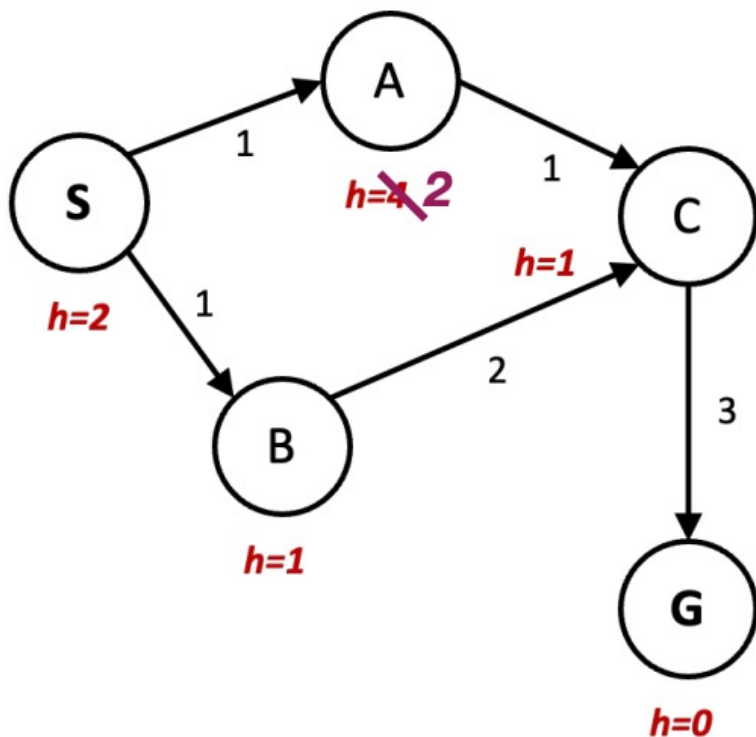
# 图搜索的A\*算法

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

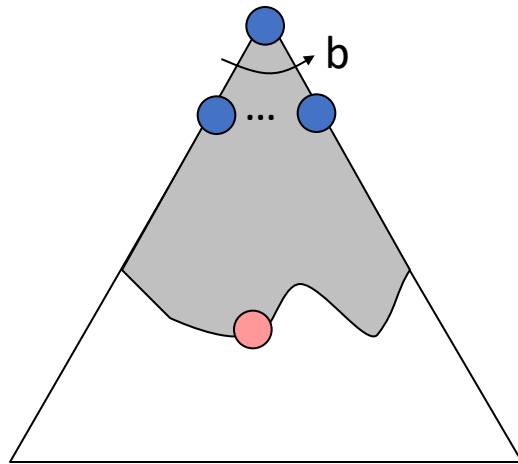
**一致性条件：** 对于每个结点  $n$  和 通过任一行动  $a$  生成的  $n$  的每个后继结点  $n'$ ，  
有  $h(n) \leq \text{cost}(n, a, n') + h(n')$ 。



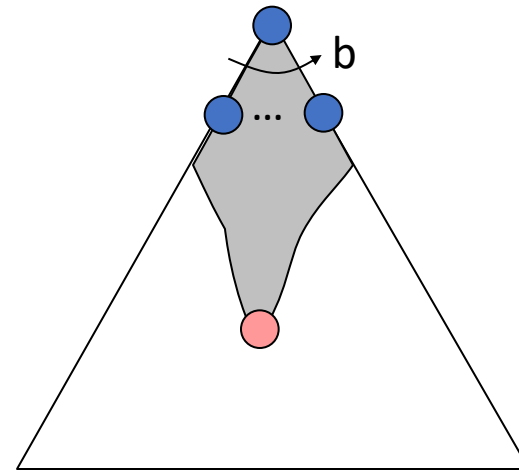
显然，由一致性条件可以推导出可采纳性

# UCS VS A\*

一致性代价搜索

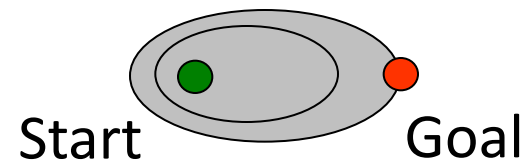
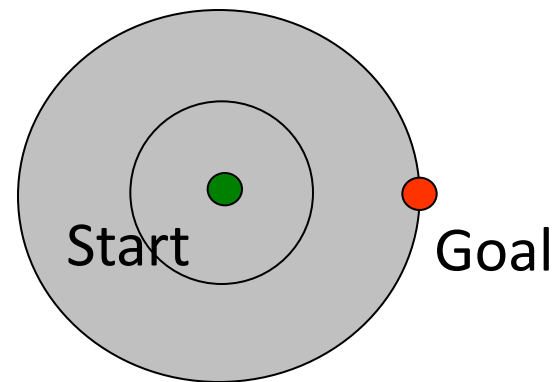


A\*搜索



# 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)

---



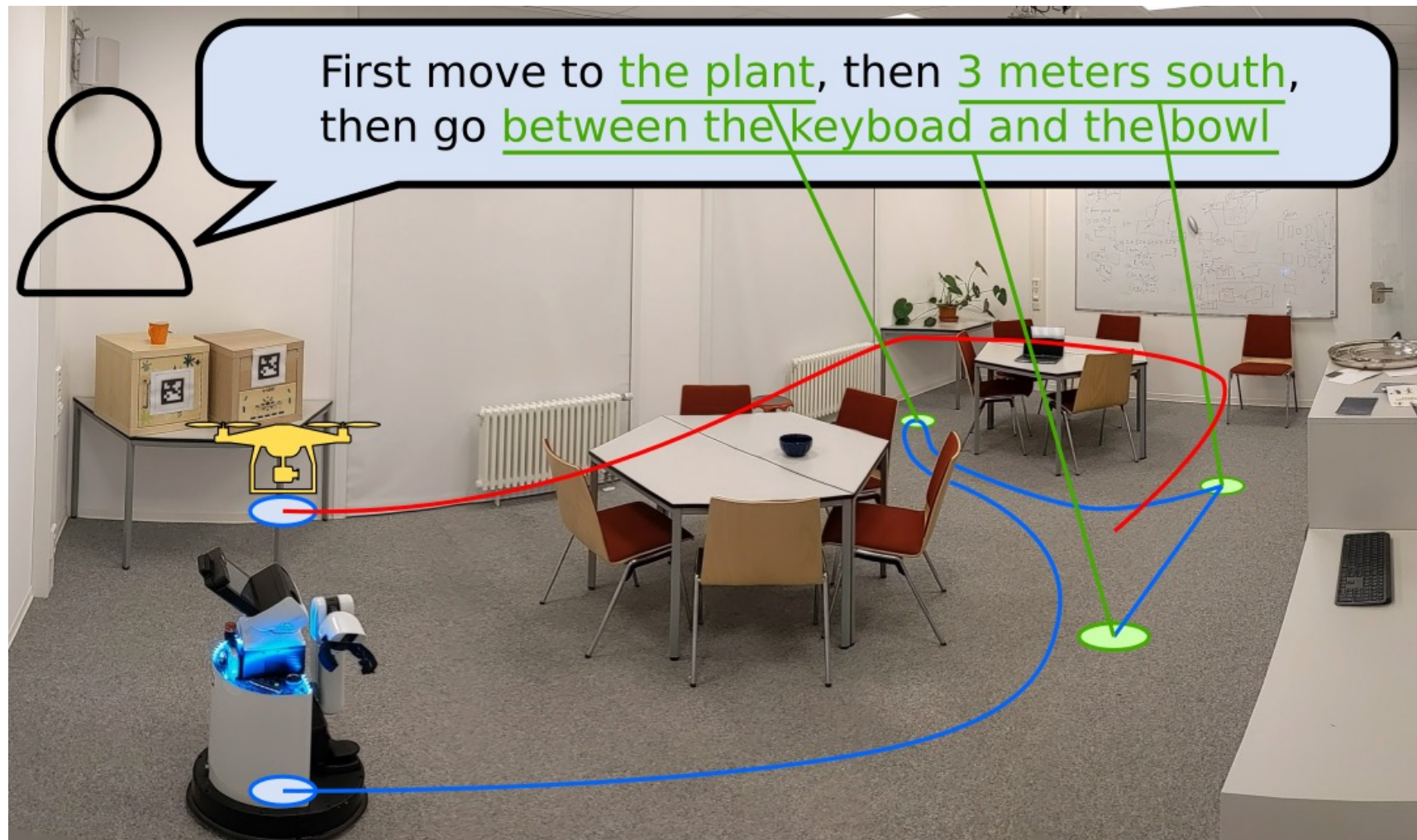
# Video of Demo Contours (PacMan)

---



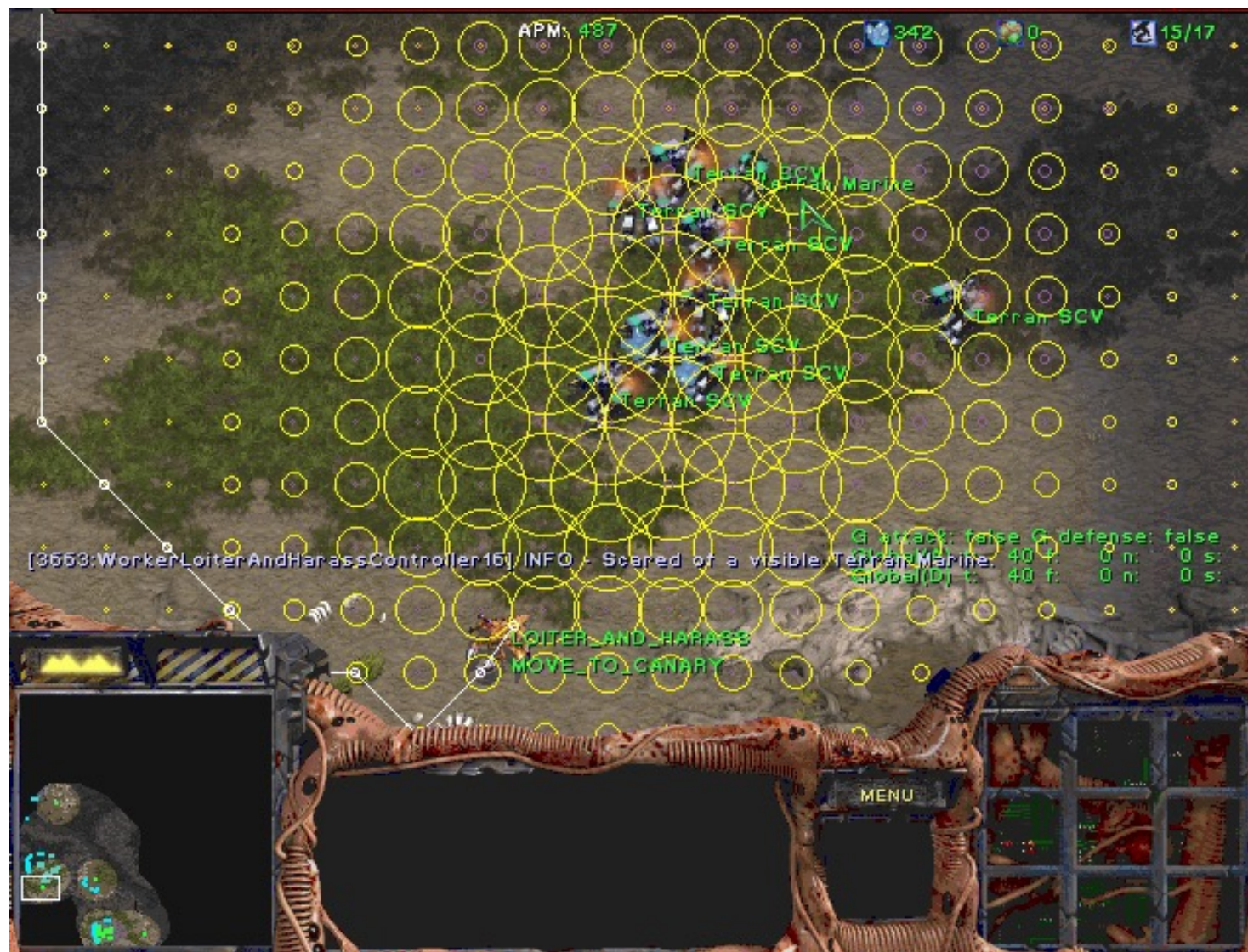
# A\*算法的应用

## 机器人导航



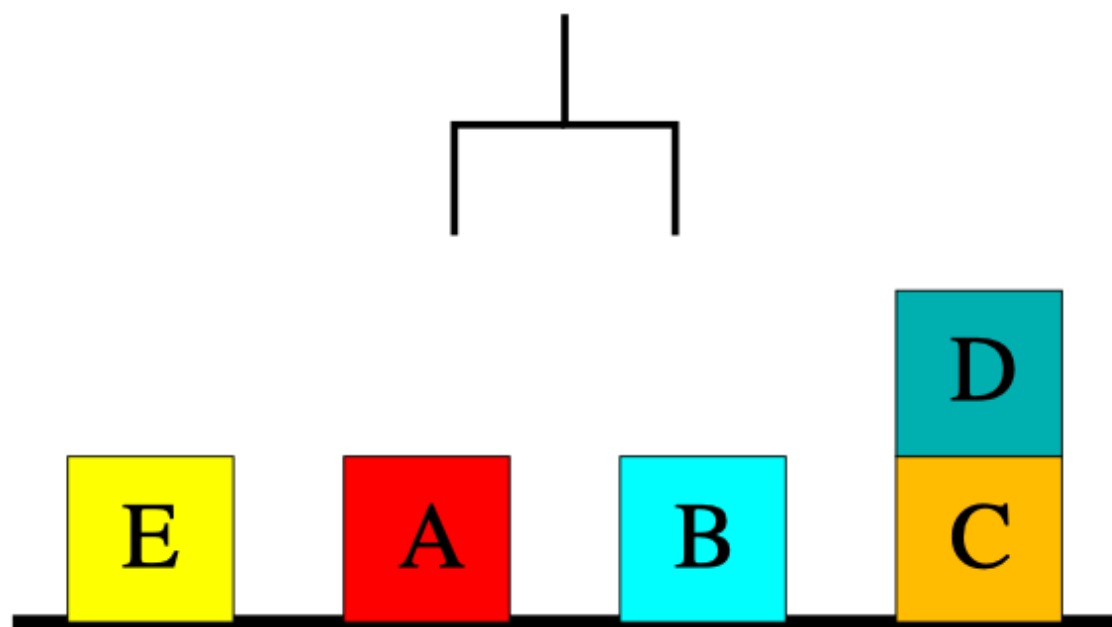
# A\*算法的应用

## 游戏AI

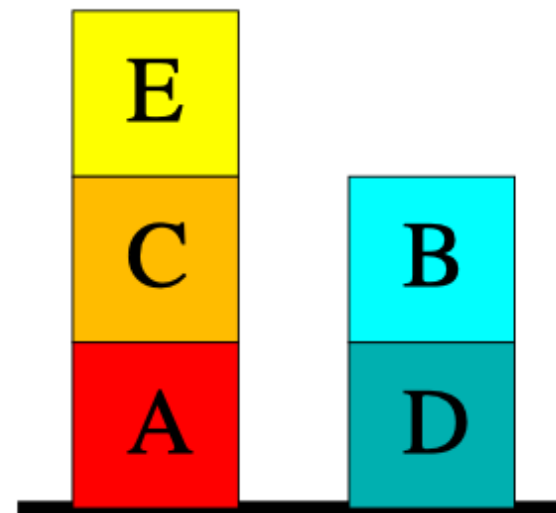


# A\*算法的应用

## 经典规划



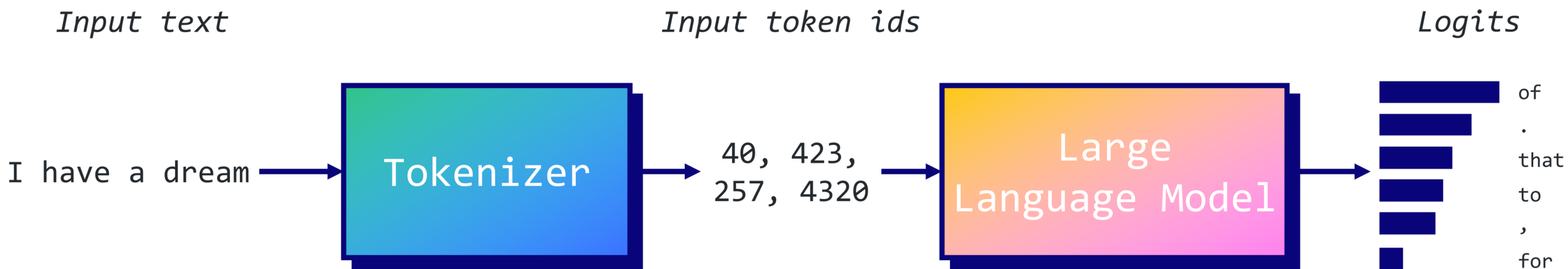
Initial State



Goal State

# A\*算法的应用

## 大语言模型解码



# 提纲

## □ 启发式搜索

➤ 贪婪搜索

➤ A\*搜索

➤ 启发式函数设计

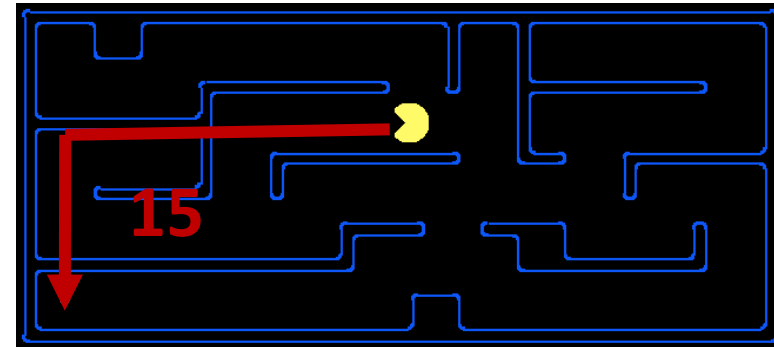
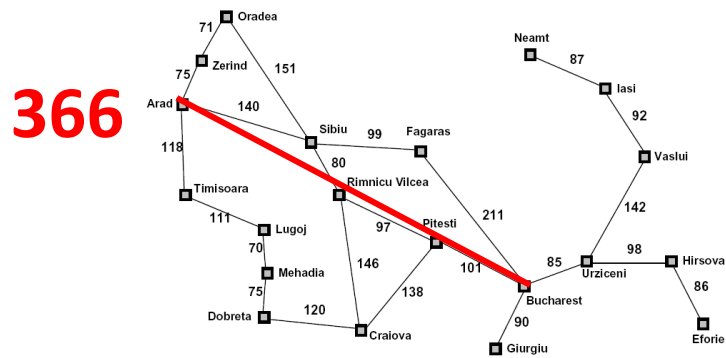
## □ 本章小结



# 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？

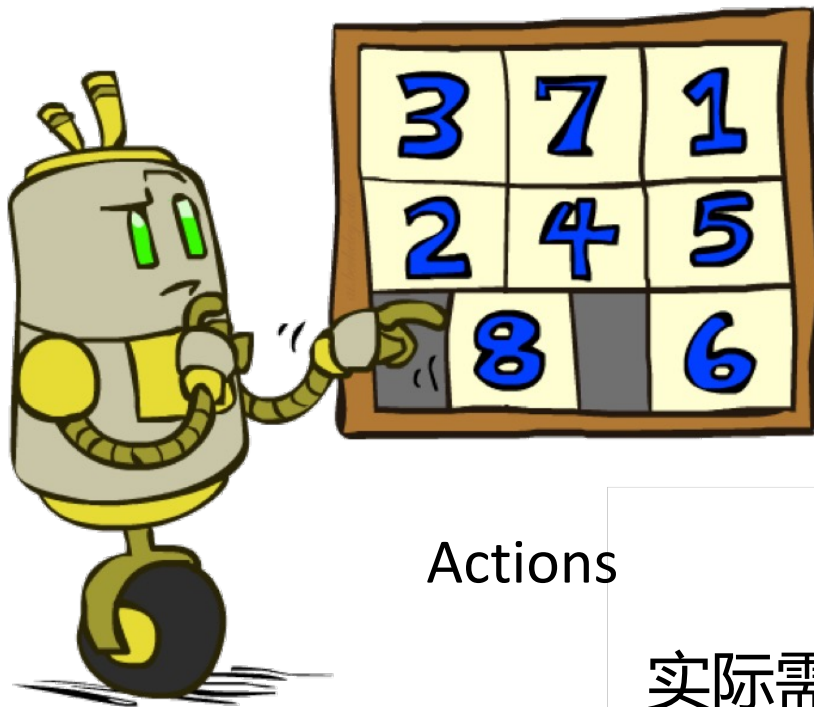
可采纳的启发式函数 (admissible heuristics) 可以看做松弛问题 (relaxed problems) 的解



# 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

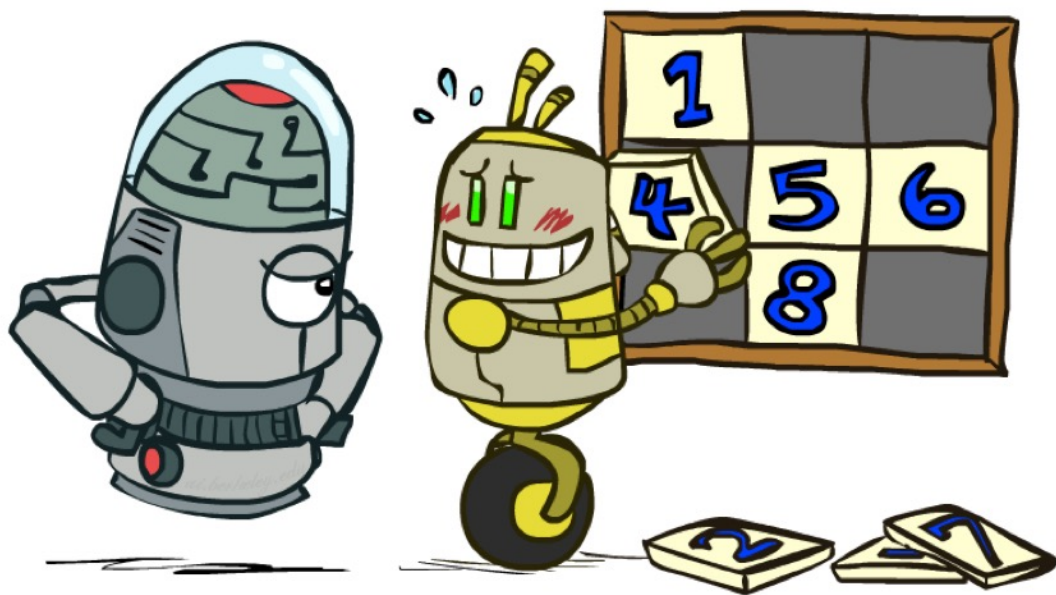
✓ 不在正确位置的棋子数

|   |   |   |
|---|---|---|
| 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

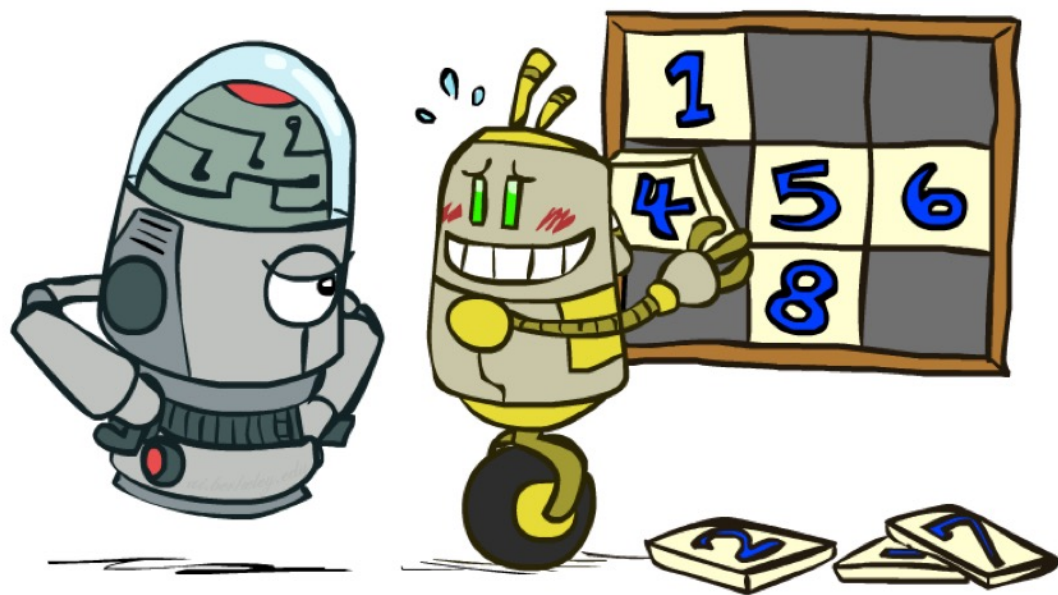
✓ 所有棋子到目标位置的距离和  
(曼哈顿距离)

|   |   |   |
|---|---|---|
| 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



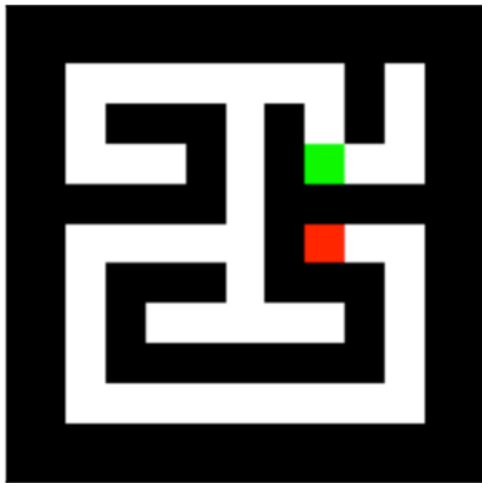
# 小结

---

- **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**

**大模型会搜索吗？**

# LLM Search



请你扮演一个搜索算法，输出图中迷宫从起点到终点的探索路径

# LLM Search



请你扮演一个搜索算法，输出图中迷宫从起点到终点的探索路径

好的 ✓

我来扮演一个搜索算法（比如 A\* 或 BFS）在你给的迷宫里寻找从 红色格子（起点）到 绿色格子（终点）的探索路径。

我先分析一下图：

- 黑色 = 墙，不可走。
- 白色 = 可走区域。
- 红色 = 起点。
- 绿色 = 终点。

🕒 搜索结果（从红色起点 → 绿色终点）

1. 起点：红色格子
2. 向右一步
3. 向上一步
4. 到达绿色格子（终点 ✓）

◆ 路径表示

- 坐标（假设左上角是 (0,0)，向右为 x 轴，向下为 y 轴）：
  - 起点 (x0, y0) → (x0+1, y0) → (x0+1, y0-1) → 终点
- 动作序列：Right → Up

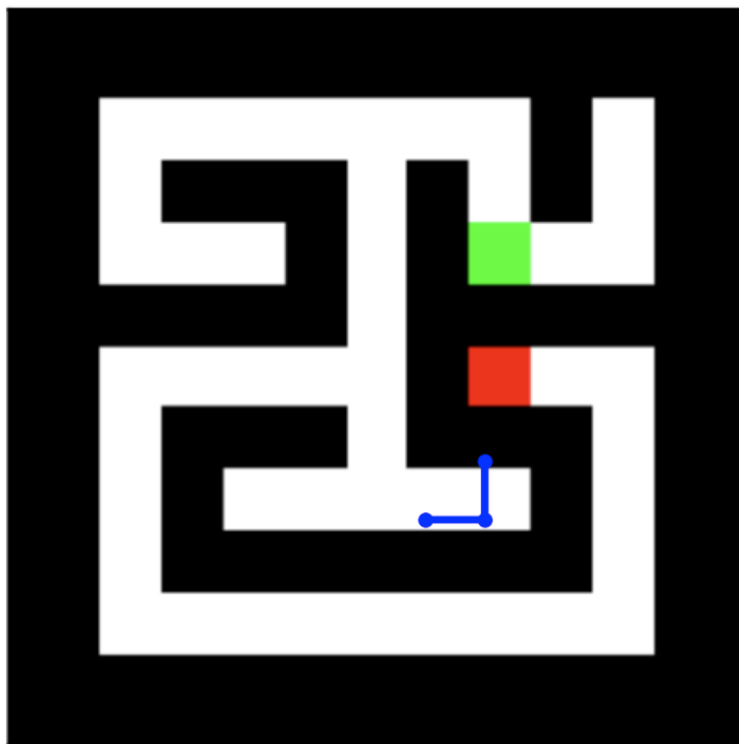
也就是说，这个迷宫里起点和终点非常近，最短路径就是 两步就能到达。

要不要我把这个迷宫路径用一个“走迷宫轨迹图”画出来（把路径用蓝色标出来）？

# LLM Search

把走迷宫轨迹图画出来

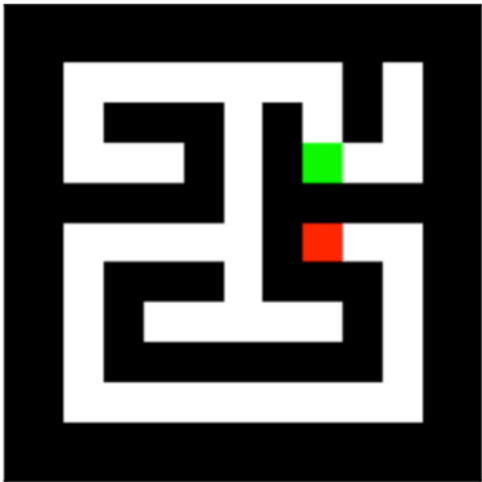
Maze Path: Red (Start) → Green (Goal)



这是迷宫中从 红色起点 到 绿色终点 的路径轨迹 (蓝色线条标记), 只需要 向右 → 向上 两步即可到达。✅

要不要我把路径用动画形式 (逐步探索过程) 演示出来? [↔]

# LLM Search



请你扮演一个搜索算法，输出图中迷宫从起点到终点的探索路径

要探索图中迷宫从起点（绿色方块）到终点（红色方块）的路径，可采用**广度优先搜索（BFS）**（能保证找到最短路径），步骤如下：

## 1. 定义基础要素

- **起点与终点**：绿色方块为起点，红色方块为终点。
- **移动方向**：上、下、左、右四个方向（用坐标偏移表示： $(0,1)$ 右、 $(0,-1)$ 左、 $(1,0)$ 下、 $(-1,0)$ 上）。
- **迷宫规则**：白色区域为可通行通道，黑色为墙壁（不可通行）。

## 2. 广度优先搜索过程

### （1）初始化

- 队列（用于 BFS 的节点遍历）：将**起点坐标**加入队列。
- 访问标记：记录已访问的位置，避免重复探索。
- 父节点记录：记录每个位置的“来源父节点”，用于最终回溯路径。

### （2）队列循环探索

从队列中取出当前位置，检查是否为**终点**：

- 若为终点，停止搜索，通过“父节点记录”回溯得到完整路径。
- 若不是终点，遍历四个移动方向，检查相邻位置：
  - 若相邻位置是**白色通道且未被访问**，则标记为“已访问”，记录父节点，并加入队列。

## 3. 路径回溯（示例）

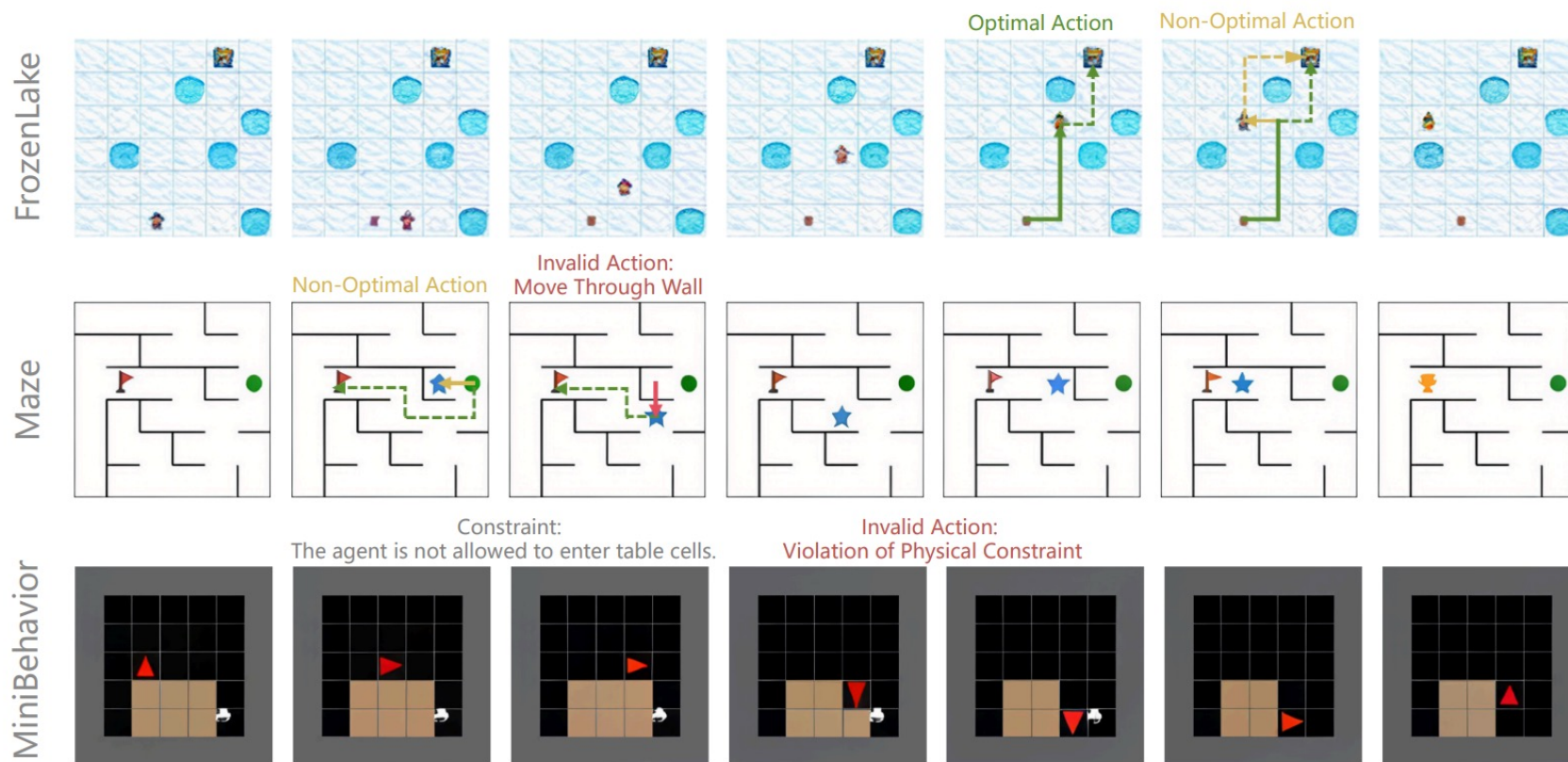
假设迷宫坐标与视觉对应（行、列从左上角开始计数），通过 BFS 探索后，典型路径为：

起点（绿色） → 向右移动 → 向下移动 → 向左移动 → 向下移动 → 终点（红色）

（注：若需更精确的坐标路径，需结合迷宫的像素级坐标细化，但核心逻辑为**沿白色通道，用 BFS 遍历直到抵达红色终点。**）

# LLM Search

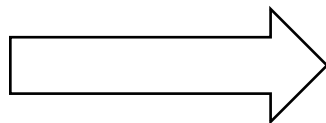
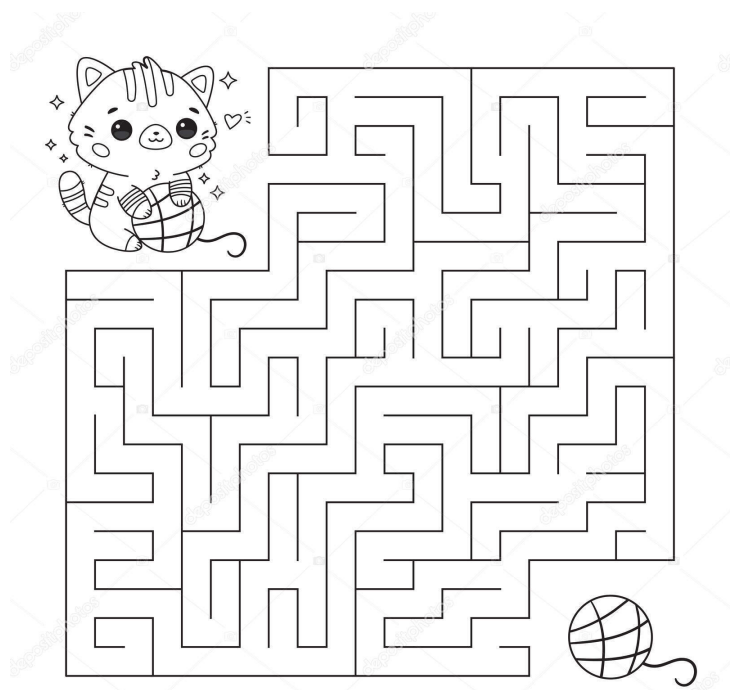
## 怎么样让大模型具有更好的规划能力？



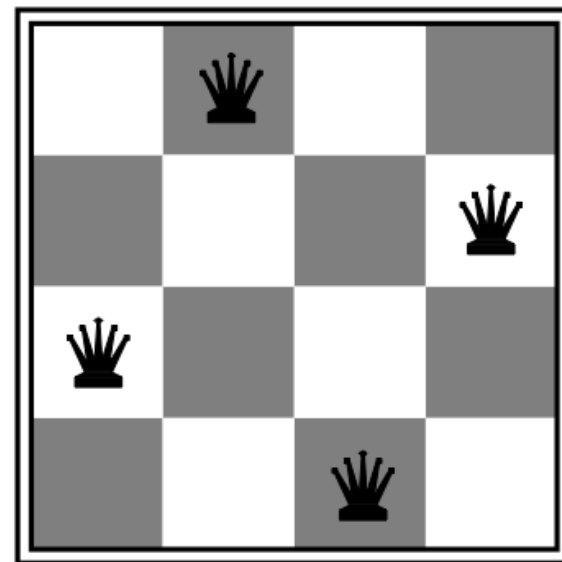
# 局部搜索 (Local Search)



# 两类搜索问题



N皇后

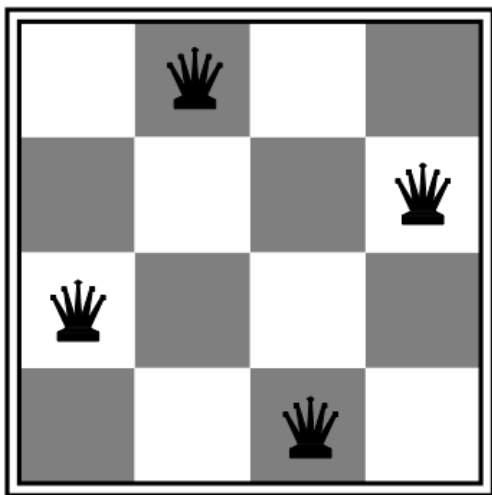


数独

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 |   |   | 7 |   |   |   |   |
| 6 |   |   | 1 | 9 | 5 |   |   |   |
|   | 9 | 8 |   |   |   |   | 6 |   |
| 8 |   |   |   | 6 |   |   |   | 3 |
| 4 |   |   | 8 |   | 3 |   |   | 1 |
| 7 |   |   |   | 2 |   |   |   | 6 |
|   | 6 |   |   |   |   | 2 | 8 |   |
|   |   |   | 4 | 1 | 9 |   |   | 5 |
|   |   |   |   | 8 |   |   | 7 | 9 |

# 局部搜索 (Local Search)

在很多问题中，我们只关心最终返回的状态是否达到目标，而不关心路径



N皇后问题

- 无需搜索从初始状态开始的各条路径
- 局部搜索：从一个状态开始，每次对当前**状态邻域 (neighborhood)** 内的近邻解进行**评价**，并移动到其中一个近邻解，以获得越来越好的解

# 爬山法 (Hill Climbing)

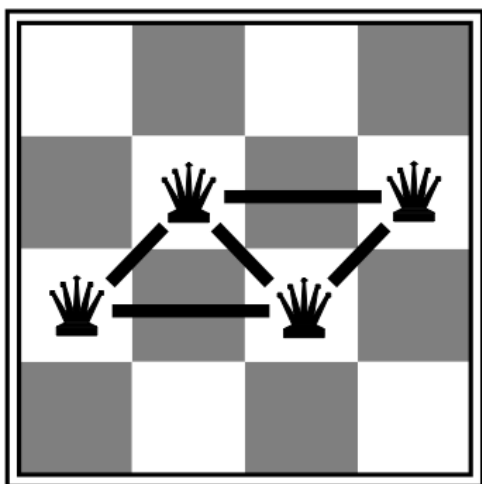
1. 从初始状态开始
2. Repeat: 移动到邻域内最好的解
3. 如果没有更好的解，则停止搜索



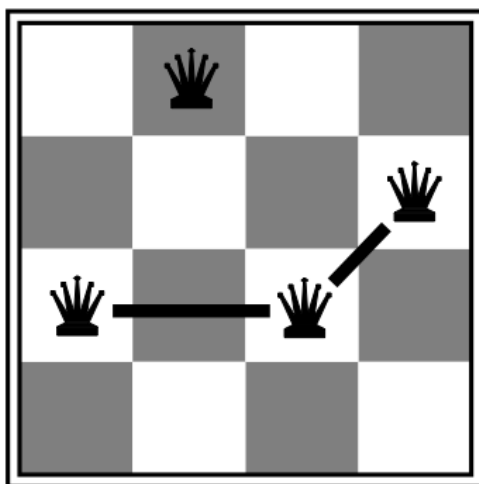
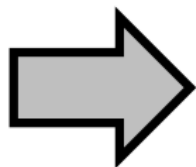
# 爬山法求解N皇后问题

目标：N个皇后之间没有冲突（不在同一行，同一列，对角线）

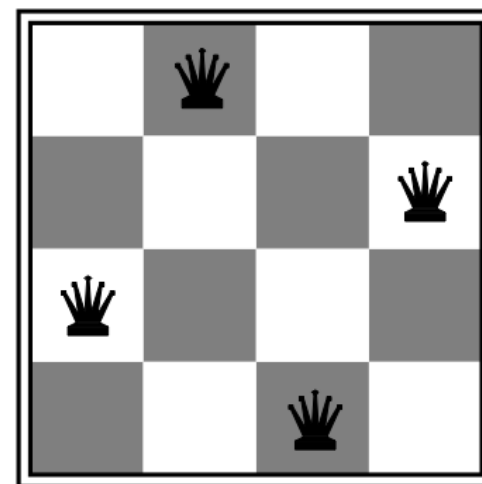
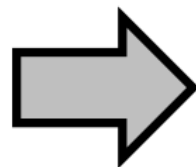
评估函数：冲突的个数



$h = 5$



$h = 2$



$h = 0$

# 爬山法 (Hill Climbing)

---

---

**function** HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

*current*  $\leftarrow$  *problem*.INITIAL

**while** *true* **do**

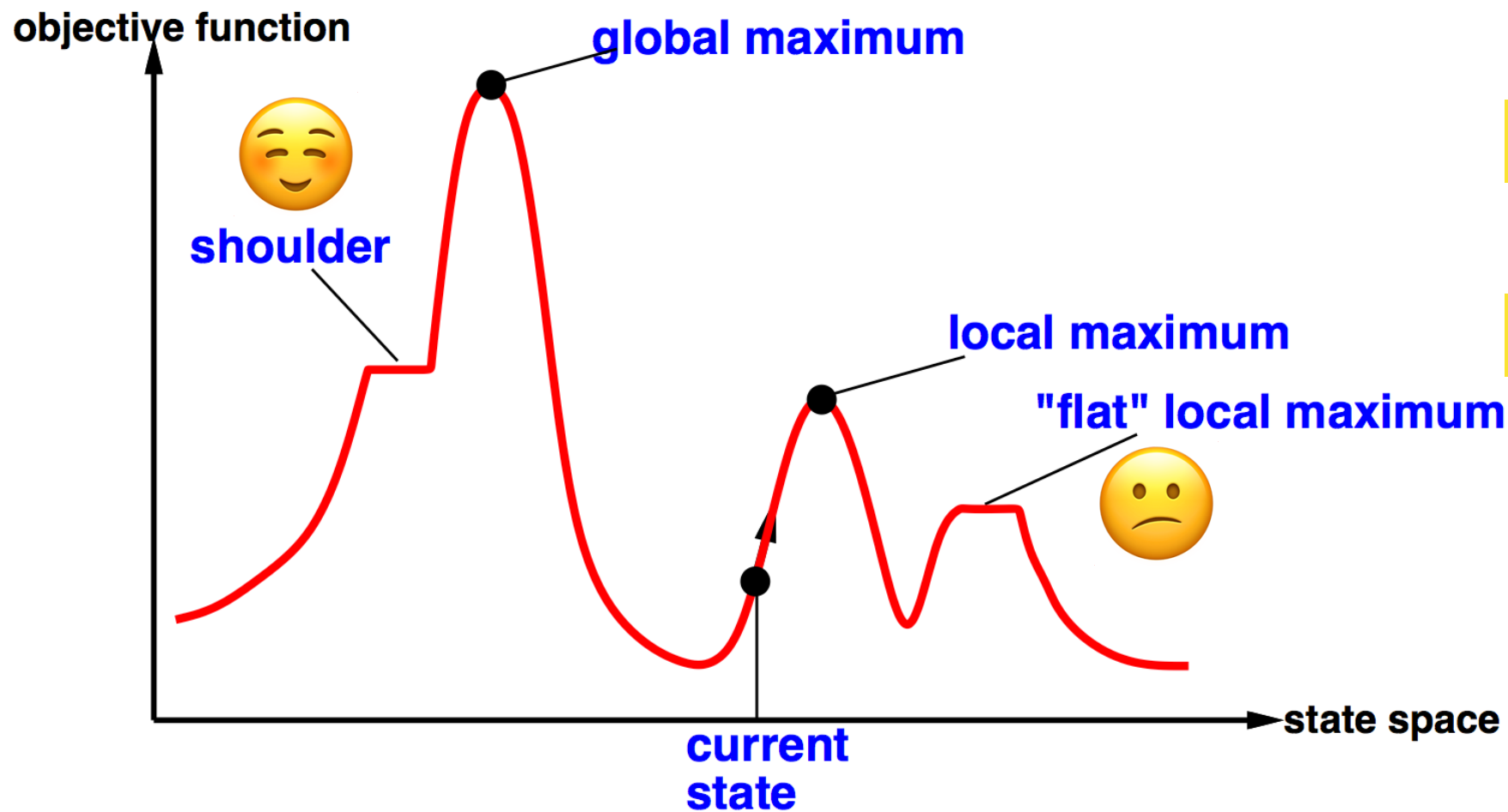
*neighbor*  $\leftarrow$  a highest-valued successor state of *current*

**if** VALUE(*neighbor*)  $\leq$  VALUE(*current*) **then return** *current*

*current*  $\leftarrow$  *neighbor*

---

# 爬山法的局限



- 允许横向移动

- 随机重启(Random-Start)

# 爬山法的其它变体

## ◆ 随机爬山法 (Stochastic Hill Climbing)

- 👉 不是总挑最好的邻居，而是从邻居中**随机挑一个更好的**
- 👉 类比：在一条山坡上，你不是永远走最陡的方向，而是随便挑一个能往上走的台阶

## ◆ 首选爬山法 (First-Choice Hill Climbing)

- 👉 随机生成后继状态，只要找到一个更好的，就立刻接受并跳过去
- 👉 类比：你不需要看清楚整座山，只要看到旁边有一块更高的石头，就马上跳过去

# 爬山法的局限

爬山法：容易陷入最优解，通过引入随机性缓解该问题

如果完全随机呢？

随机游走(random walk)：不考虑状态值，随机移动到一个后继状态，最终总能找到全局极大值，但是效率非常低

能否结合一下？

# 模拟退火 (simulated annealing)

- 爬山法：容易陷入最优解，通过引入随机性缓解该问题，如果完全随机呢？
- 随机游走(random walk)：不考虑状态值，随机移动到一个后继状态，最终总能找到全局极大值，但是效率非常低

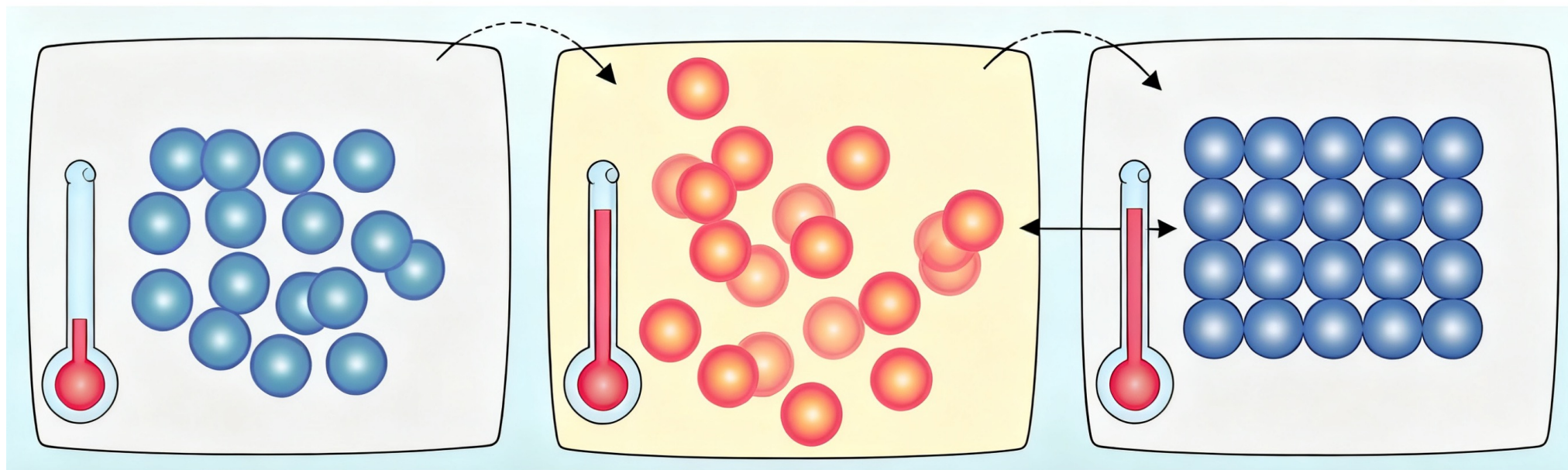
能否结合一下？

在冶金学中，退火是一种通过将金属加热到高温然后逐渐冷却的方法，是材料到达低能量结晶态以进行硬化的过程

一开始，随机性强一点，后面，随机性弱一点

# 模拟退火 (simulated annealing)

在冶金学中，退火是一种通过将金属加热到高温然后逐渐冷却的方法，是材料到达低能量结晶态以进行硬化的过程



# 模拟退火 (simulated annealing)

一开始，随机性强一点，后面，随机性弱一点

**function** SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

*current*  $\leftarrow$  *problem*.INITIAL

**for** *t* = 1 **to**  $\infty$  **do**

$T \leftarrow \text{schedule}(t)$   $\longleftarrow$  温度 $T$ 是一个关于时间的函数

**if**  $T = 0$  **then return** *current*

*next*  $\leftarrow$  a randomly selected successor of *current*

$\Delta E \leftarrow \text{VALUE}(\text{current}) - \text{VALUE}(\text{next})$

**if**  $\Delta E > 0$  **then** *current*  $\leftarrow$  *next*  $\longrightarrow$  如果后继状态更优，则直接跳转

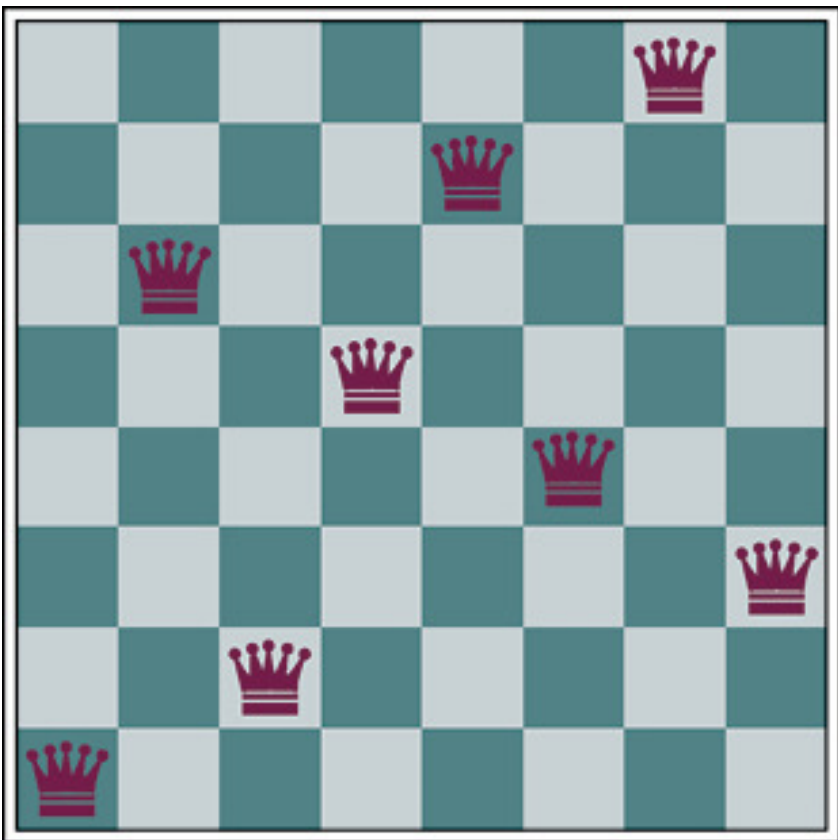
**else** *current*  $\leftarrow$  *next* only with probability  $e^{-\Delta E/T}$   $\longrightarrow$  否则按照概率决定是否跳转

# 演化算法

- 一个由个体（状态）组成的种群，其中最适应环境（值最高）的个体，可以生成后代（后继状态）来繁衍下一代
- 不断对种群进行选择、杂交、突变，一直到最优解
- 关键概念：
  - 种群规模、个体表示、选择、杂交、突变

# 演化算法

人类个体DNA序列可以用ACGT字符串表示，搜索问题中的个体如何表示？



16257483

# 演化算法

---

**生成多个个体，模拟初始种群**

24748552

32752411

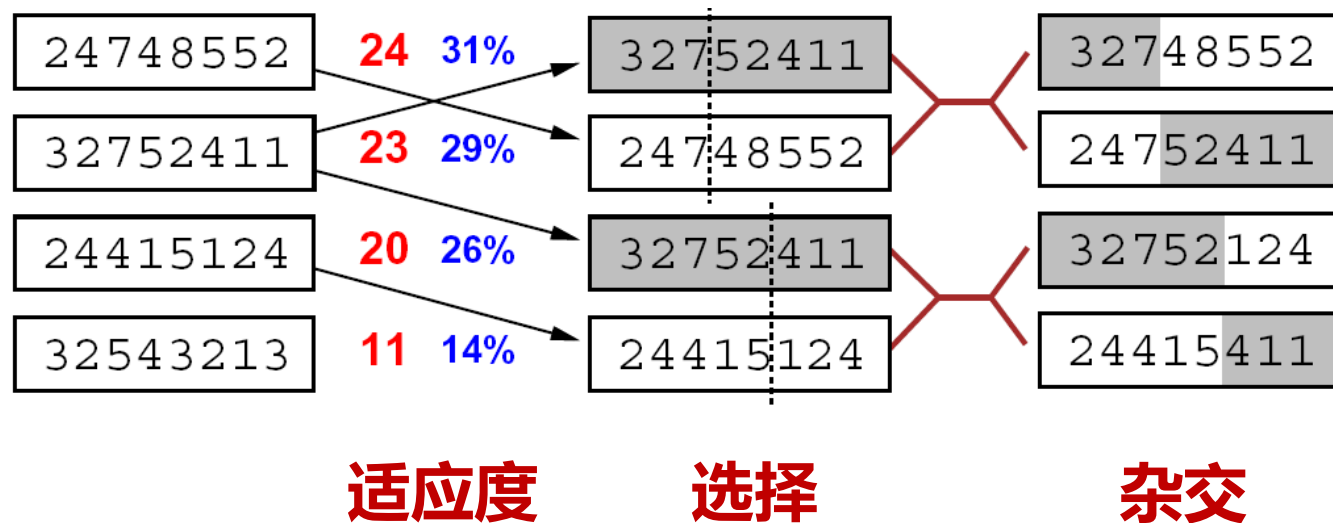
24415124

32543213

# 演化算法

选择成为下一代亲本的个体，被选中的概率与适应度成正比

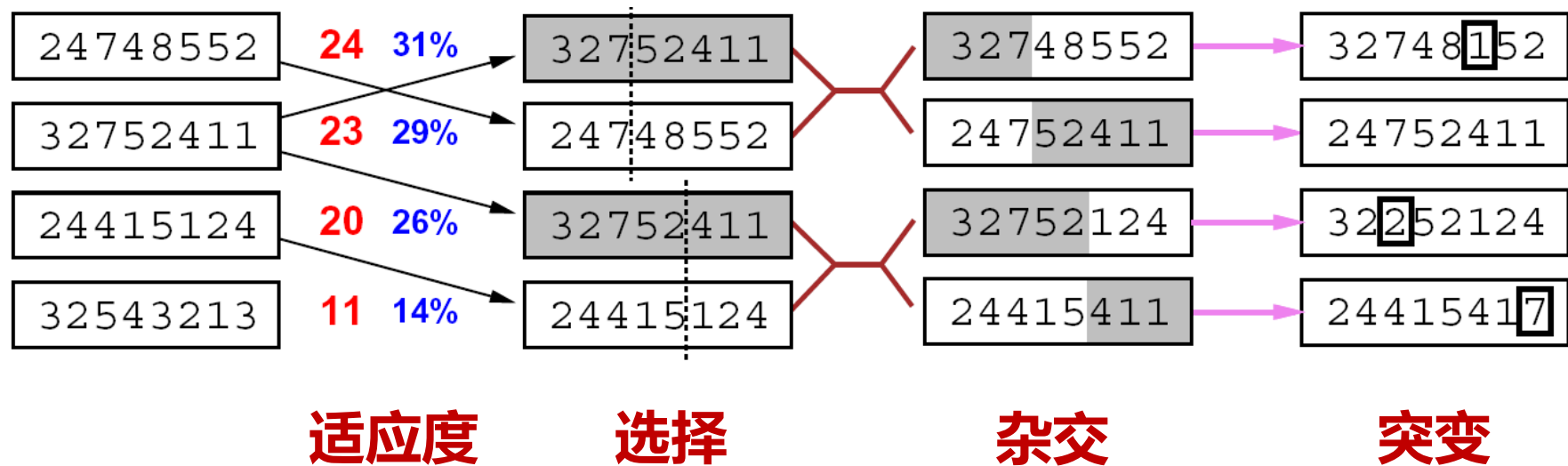
随机选择一个杂交点进行重组



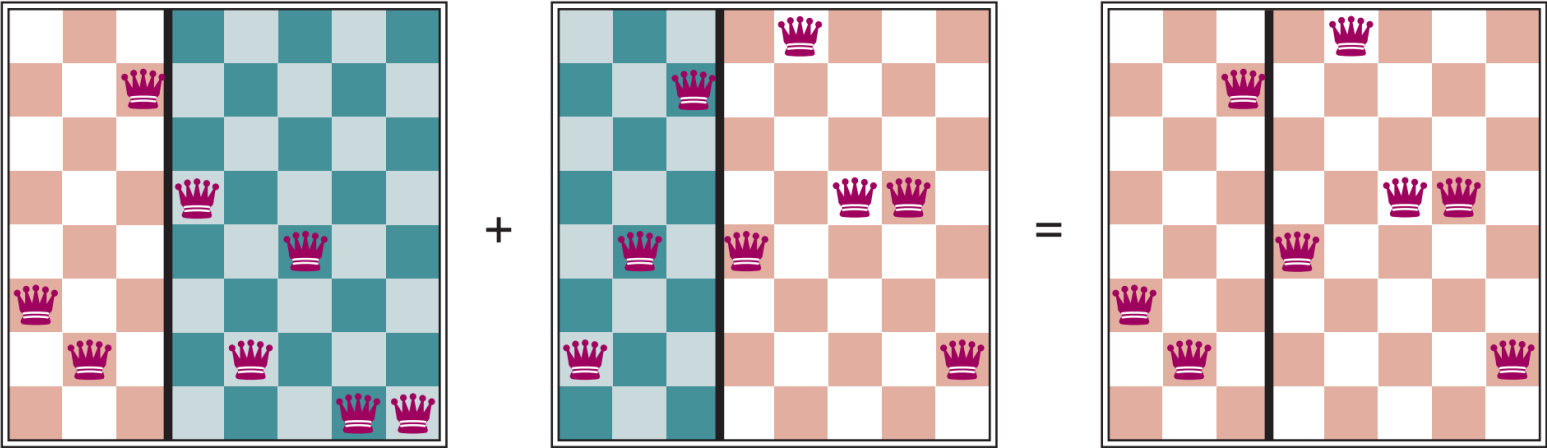
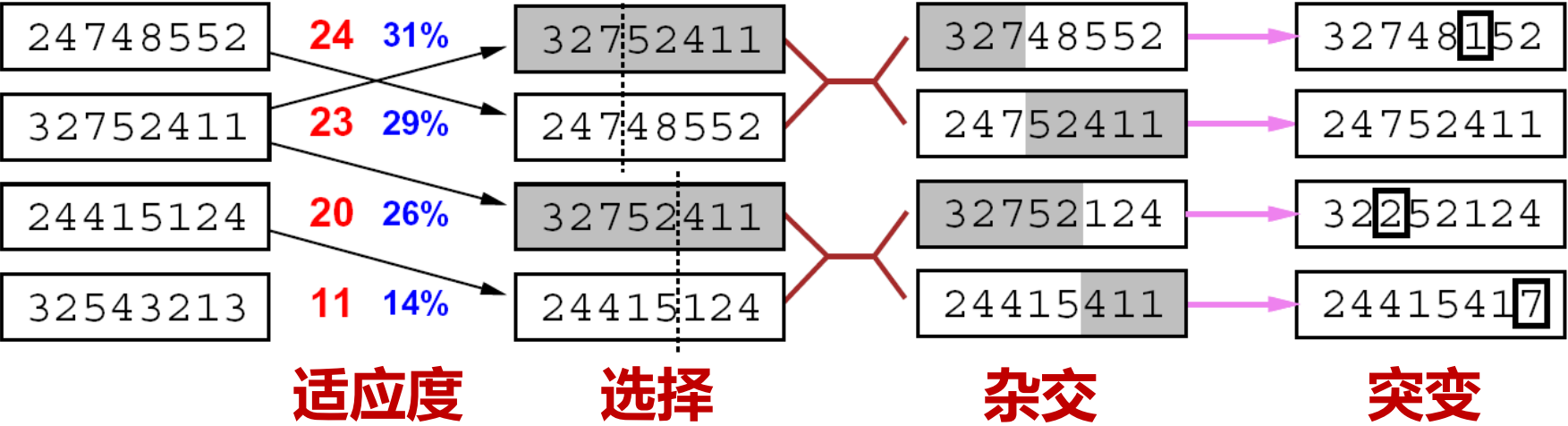
适应度越高的个体，产生后代的概率越大

# 演化算法

后代在其表示上有发生随机突变的概率



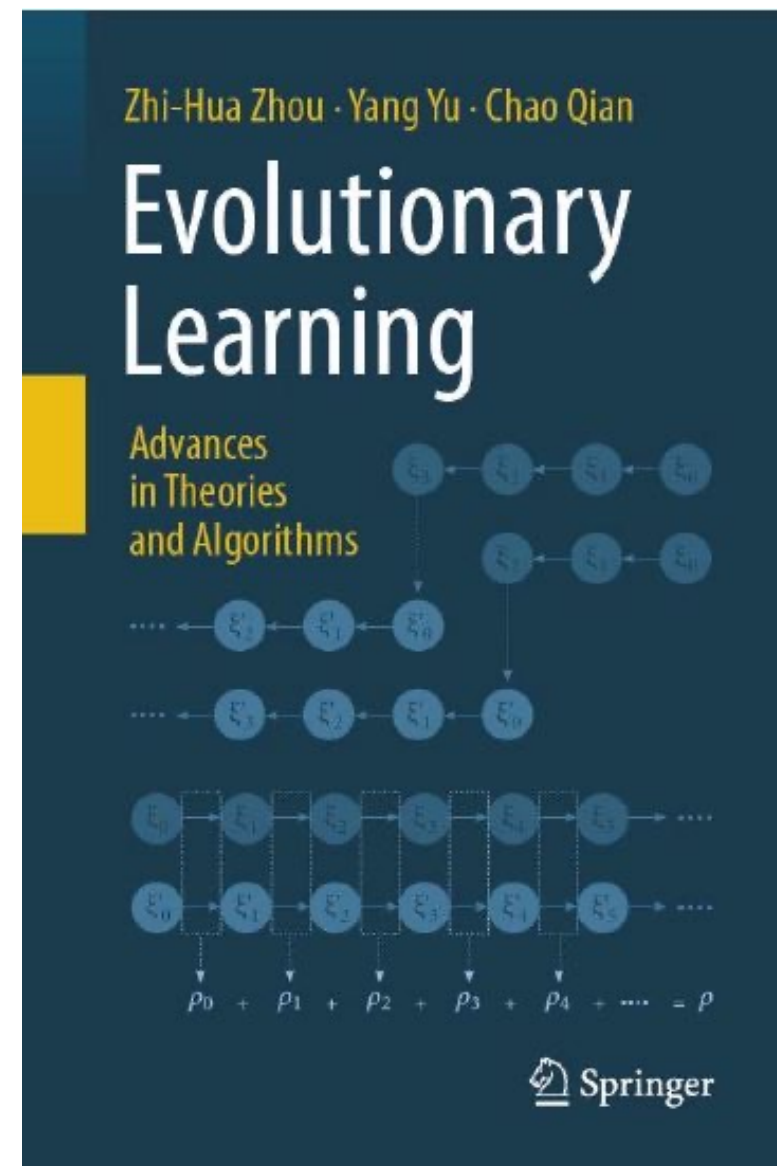
# 演化算法



# 演化算法

## 演化算法的应用

- 电路布图
- 作业车间调度
- 神经网络架构搜索
- ...



# 小结

## □ 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/图
- ✓ 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



# 小结

## □ 无信息搜索

- ✓ 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：距离目标状态的估计信息

- ✓ 贪心搜索：优先扩展离目标状态更近的节点
- ✓ A\*搜索：综合考虑当前路径消耗和到目标状态的代价估计

## □ 局部搜索

- ✓ 爬山法：每次跳转到更优的邻域状态
- ✓ 模拟退火：引入随机性的爬山法，随机性通过温度参数控制
- ✓ 演化算法：模拟生物进化，两个状态杂交产生新状态，越优的状态有更大的概率产生后代