教学安排（星期三，9:50~12:15，五教5102）

周次	日期	内容	作业	编程	项目	内容
1	9月14日	绪论				绪论。课程内容。Python
2	9月21日	无信息搜索		1		状态空间表示。宽度优先，一致代价，深度优先
3	**9月28日**	**有信息搜索**	**1**		**1**	**算法复杂度分析。贪婪最佳优先搜索，A* 算法**
4	10月 5日	约束满足	2			启发函数的设计，回溯算法，局部搜索
5	10月12日	对抗搜索		2		极小极大搜索，截断搜索与剪枝，蒙特卡洛树搜索
6	10月19日	命题逻辑	3			命题逻辑，演绎推理
7	10月26日	谓词逻辑	4			谓词逻辑，归结原理
8	11月 2日	回归问题	5			简单线性回归，多元回归，正则化
9	11月 9日	分类问题		3	2	Logistic 回归，Softmax 回归
10	11月16日	前馈神经网络	6			前馈神经网络。PyTorch
11	11月23日	卷积神经网络		4		卷积神经网络。TensorFlow
12	11月30日	马尔可夫决策过程	7			强化学习的数学基础
13	12月 7日	策略迭代与价值迭代		5		状态转移概率已知情况下的预测与控制
14	12月14日	蒙特卡洛与时序差分	8			状态转移概率未知情况下的预测与控制
15	12月21日	深度强化学习				价值函数的近似，深度强化学习

问题求解 逻辑推理 机器学习

人工智能基础

Fundamentals of Artificial Intelligence

江瑞

自动化系

无信息搜索算法

宽度优先

- 首先扩展初始状态
- 然后扩展初始状态的子节点
- 依此类推直到达到终止状态

一致代价

- 首先扩展初始状态
- 然后扩展路径代价最小的子节点
- 依此类推直到达到终止状态

深度优先

- 首先扩展初始状态
- 然后扩展开节点表中最后加入的节点
- 依此类推直到达到终止状态

深度受限

- 设一个深度上限

迭代加深

- 不断增加深度上限

关注的算法性能

完备性

- 是否能保证找到解?

最优性

- 是否能保证找到最优解?

算法分析

时间效率

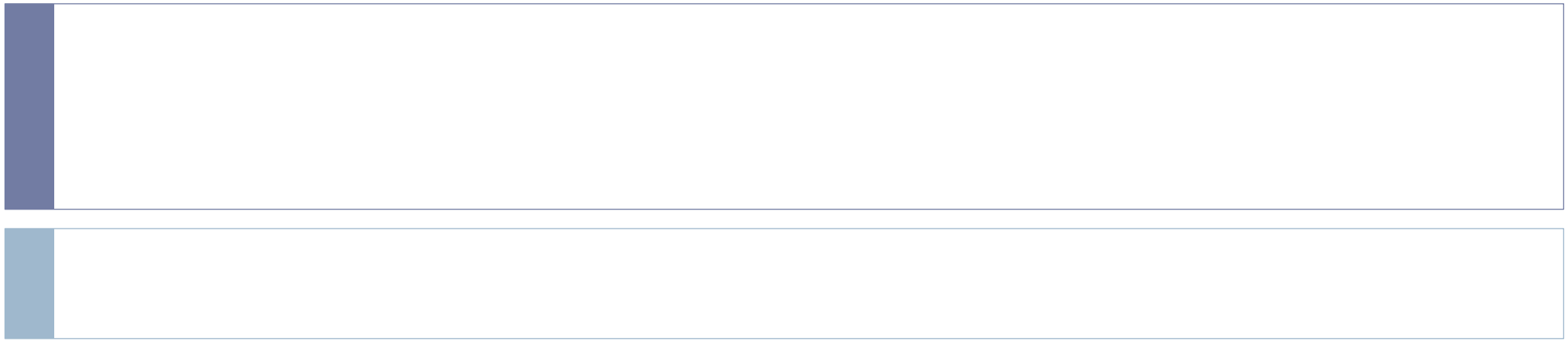
- 需要多少时间?

空间效率

- 消耗多少内存?

算法复杂度分析

Complexity of an Algorithm



基本概念

计算模型

- 随机访问计算机
  - CPU：单核
  - 数据：支持整型浮点等基本类型
  - 运算：支持加减乘除等基本运算
  - 指令：不支持并行计算

输入大小

- 算法执行效率由输入大小决定
  - 元素的个数：输入包含多少个基本类型数据
  - 数据的维度：输入向量的维度、矩阵的维度
  - 数据的字长：输入数据由多少位二进制表示

执行效率 $=f$(输入大小)

复杂度

度量单位

- 时间复杂度（时间效率）
  - 经验度量：普通时间单位
  - 理论度量：基本操作次数
- 空间复杂度（空间效率）
  - 理论度量：使用内存大小

基本操作

- 算法中消耗时间最多的操作
  - 最内层循环中最消耗时间的操作

Big-O

如果函数 $f(n)$ 的**上界由 $g(n)$ 的某个常数倍界定**，则称

$$f(n) \in O(g(n))$$

即存在常数 c 和 n_0，使得

$$f(n) \leq cg(n), \text{ for all } n \geq n_0$$

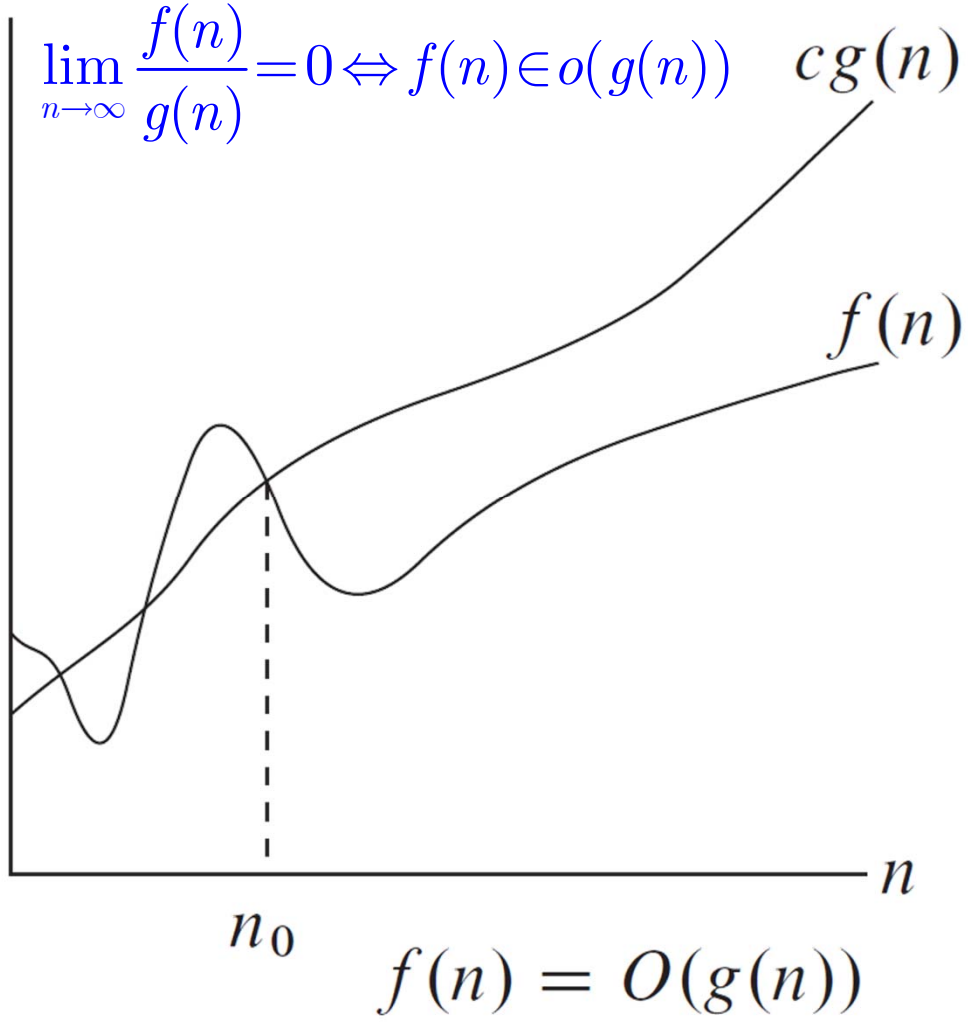
如果对于任意常数 c 均存在 n_0，使得

$$f(n) < cg(n), \text{ for all } n \geq n_0$$

则称

$$f(n) \in o(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Leftrightarrow f(n) \in o(g(n))$$



Big-**Omega**

如果函数 $f(n)$ 的**下界由 $g(n)$ 的某个常数倍界定**，则称

$$f(n) \in \Omega(g(n))$$

即存在常数 c 和 n_0，使得

$$f(n) \geq cg(n), \text{ for all } n \geq n_0$$

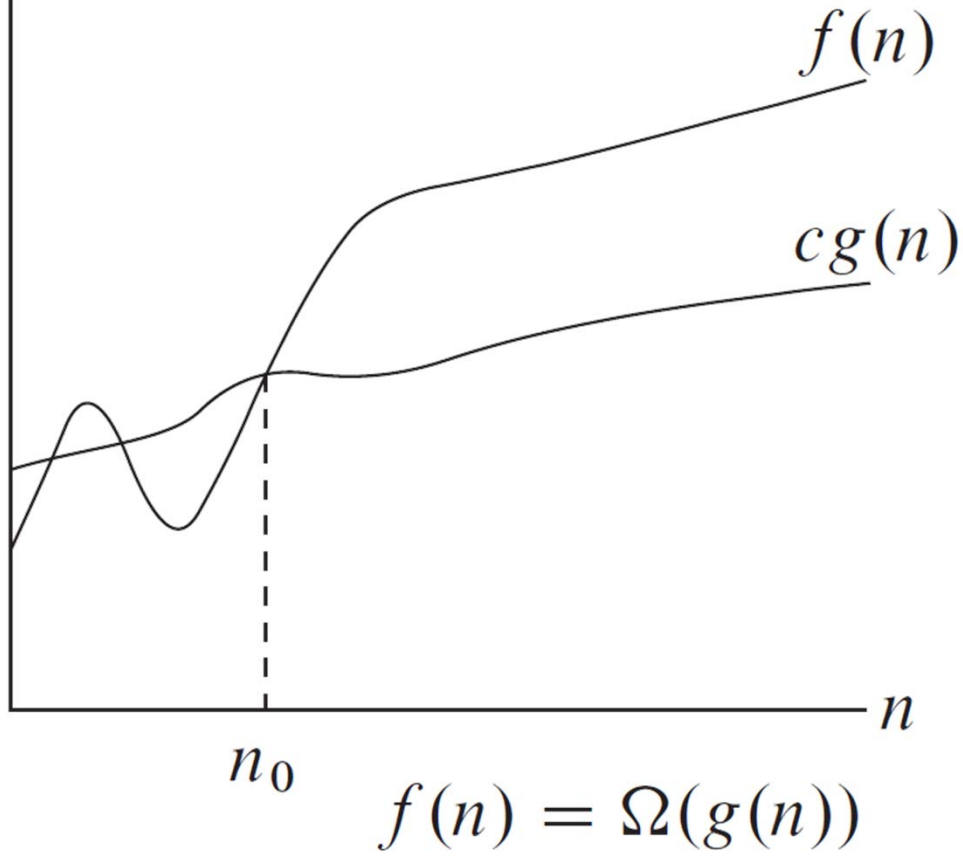
如果对于任意常数 c 均存在 n_0，使得

$$f(n) > cg(n), \text{ for all } n \geq n_0$$

则称

$$f(n) \in \omega(g(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Leftrightarrow f(n) \in \omega(g(n))$$



Big-**Theta**

如果函数 $f(n)$ 的**上界和下界均由 $g(n)$ 的常数倍界定**，则称

$$f(n) \in \Theta(g(n))$$

即存在常数 c_1, c_1和 n_0，使得

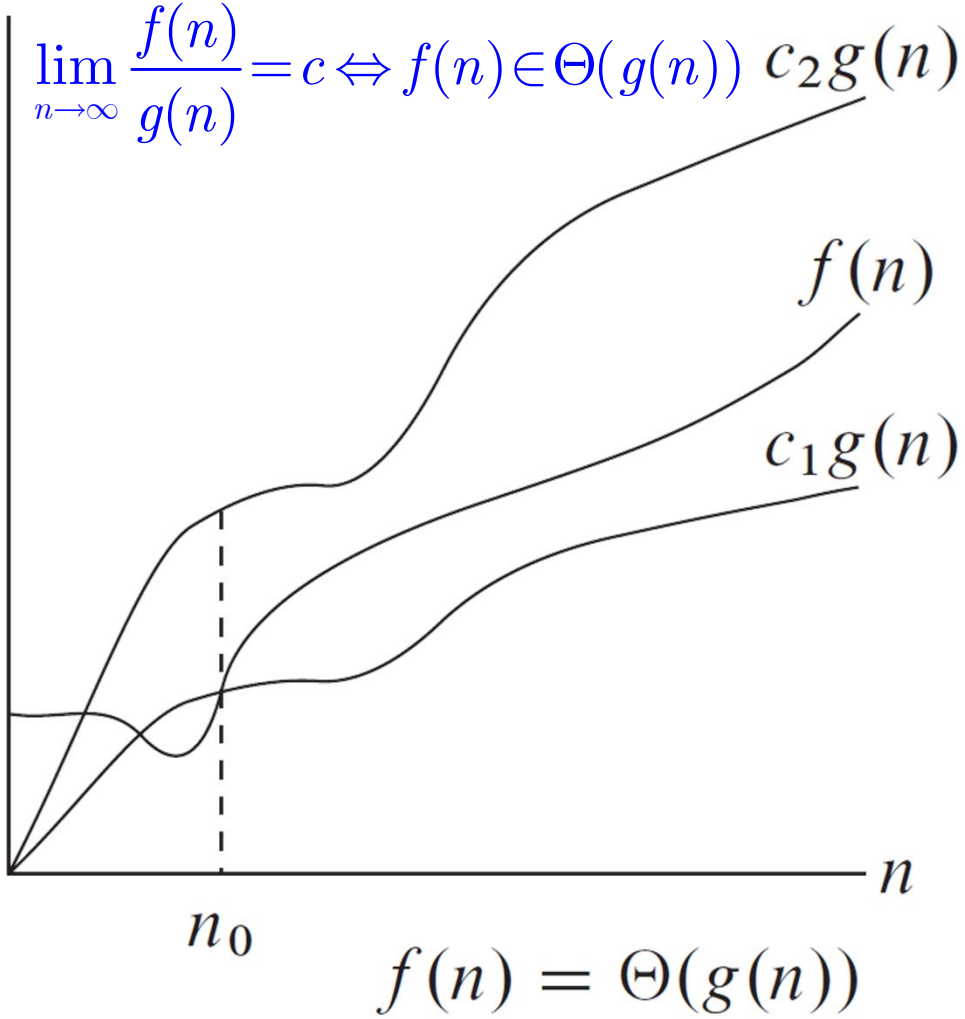
$$c_2 g(n) \le f(n) \le c_1 g(n), \text{ for all } n \ge n_0$$

$f(n) \in O(g(n))$ is like $f \le g$

$f(n) \in \Omega(g(n))$ is like $f \ge g$

$f(n) \in \Theta(g(n))$ is like $f = g$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \Leftrightarrow f(n) \in \Theta(g(n))$



举例

ALGORITHM BUBBLESORT(A)
INPUT: An array
OUTUT: The ordered array

```
i ← 1
REPEAT
    stop ← TRUE
    FOR j ← 1 TO n−i DO
        IF A[j] > A[j+1]
            swap A[j] and A[j+1]
            stop ← FALSE
    i ← i + 1
UNTIL stop
```

- 最好情况

$$\sum_{j=1}^{n-1} 1 = n-1 \in \Theta(n)$$

- 最坏情况

$$\sum_{i=1}^{n-1}\sum_{j=1}^{n-i} 1 = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

- 平均情况

$$\frac{1}{n-1}\sum_{k=1}^{n-1}\sum_{i=1}^{k}\sum_{j=1}^{n-i} 1 \in \Theta(n^2)$$

算法的时间复杂度需要分情况考虑

小结

- 对数函数
  底不相等，复杂度相同
- 多项式函数
  幂次相等，复杂度相同
- 指数函数
  底不相等，复杂度不同

$$\lim_{n\to\infty}\frac{\log_a n}{\log_b n}=\lim_{n\to\infty}\frac{\ln n/\ln a}{\ln n/\ln b}=\lim_{n\to\infty}\frac{\ln b}{\ln a}=c$$

$$\lim_{n\to\infty}\frac{\sum_{0\le k\le p}a_k n^k}{\sum_{0\le k\le q}a_k n^k}=\lim_{n\to\infty}\frac{n^p}{n^q}=\text{取决于}p\text{和}q\text{的比较}$$

$$\lim_{n\to\infty}\frac{a^n}{b^n}=\lim_{n\to\infty}\left(\frac{a}{b}\right)^n=\text{取决于}a\text{和}b\text{的比较}$$

- 增长阶次
  对数 $<$ 幂函数 $<$ 指数 $<$ 阶乘
  $\boldsymbol{\log n \ < n^{\alpha}\ (\alpha>0) < a^n \ < n! < n^n}$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}0 & f(n)\in o(g(n))\\ c>0 & f(n)\in\Theta(g(n))\\ \infty & f(n)\in\omega(g(n))\end{cases}$$

基本复杂度类别

类别	名称	说明	
1	**常数** Constant	最优的复杂度，运行效率与输入无关，非常罕见。一般来说当输入趋于无穷大时，算法执行时间总会趋于无穷大，需要的内存也会趋于无穷大	哈希表
$\log n$	**对数** Logarithmic	不能考虑所有的输入元素，否则会成为线性复杂度	二分查找
n	**线性** Linear	例如一重循环	线性查找
$n\log n$	***n*-log-*n***	例如很多根据分治思想设计的算法	归并排序 堆排序
n^2	**二次** Quadratic	例如两重循环嵌套	两重循环
n^3	**三次** Cubic	例如三重循环嵌套	三重循环
2^n	**指数** Exponential	例如穷举集合的所有子集 “指数复杂度”泛指增长阶次为指数或更高的算法	穷举子集
$n!$	**阶乘** Factorial	例如求全排列。	排列组合

增长阶次对比

n	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n	$n!$
10^1	3.3	10^1	3.3×10^1	10^2	10^3	10^3	3.6×10^6
10^2	6.6	10^2	6.6×10^2	10^4	10^6	1.3×10^{30}	9.3×10^{157}
10^3	10	10^3	1.0×10^4	10^6	10^9		
10^4	13	10^4	1.3×10^5	10^8	10^{12}		
10^5	17	10^5	1.7×10^6	10^{10}	10^{15}		
10^6	20	10^6	2.0×10^7	10^{12}	10^{18}		

n	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n	$n!$
10^1	3.3ns	10ns	33ns	0.1μs	1μs	1μs	3.6ms
10^2	6.6ns	0.1μs	0.66μs	10μs	1ms	4×10^{13}y	3×10^{141}y
10^3	10ns	1μs	10μs	1ms	1s		
10^4	13ns	10μs	0.13ms	0.1s	16.7m		
10^5	17ns	0.1ms	1.7ms	10s	11.6d		
10^6	20ns	1ms	20ms	16.7m	31.7y		

- **指数复杂度算法仅能解决很小的问题**
- 在每秒执行10亿条指令的计算机上运行

- 算法的执行效率在输入很小时并没有太大区别
- 算法复杂度分析更关注输入很大时算法的效率
- **算法即便增长阶次相同，其效率也可能有差别（相差常数倍）**

问题的复杂性

易处理的问题

如果一个算法求解一个问题时，在最坏情况下的时间复杂度属于$O(p(n))$，其中 $p(n)$ 是问题输入大小 n 的多项式函数，则称这个算法能够在多项式时间内解决该问题。因此这个问题是容易处理的。

能不能举出一些例子?

- 算法是用来求解问题的
  - 算法时间复杂度低 $\Rightarrow$ 问题简单
  - 算法时间复杂度高 $\Rightarrow$ 问题困难
  - **可以用算法的复杂度来定义问题的难易程度**
- 多项式复杂度的算法是“**快**” 的
- 一个问题可以由多项式复杂度算法解决则该问题是“**容易**”的
- 一个问题不能由多项式复杂度算法解决则该问题是“**困难**”的

优化问题

- **需要求出最优解的问题**
- 旅行商优化问题
  求访问全国每个省会城市一次且仅一次后回到起点的最短旅行路线
- 背包优化问题
  如何选择物品，使得背包中装入物品的总价格最高
- 0-1整数规划
  给定一个整数矩阵 C 和一个整数向量 d，求一个0-1向量 x，使得 Cx=d

判定问题

- **要求回答“是”或“否”的问题**
- 旅行商判定问题
  是否存在一条访问全国每个省会城市一次且仅一次后回到起点的旅行路线，其长度不超过10万公里?
- 背包判定问题
  背包中装入物品的总价格是否能够高于一百万元?
- 0-1整数规划判定
  给定一个整数矩阵 C，一个整数向量 d，以及一个0-1向量 x，Cx=d 是否成立?

非确定性算法 (Non-deterministic algorithm)

非确定性算法

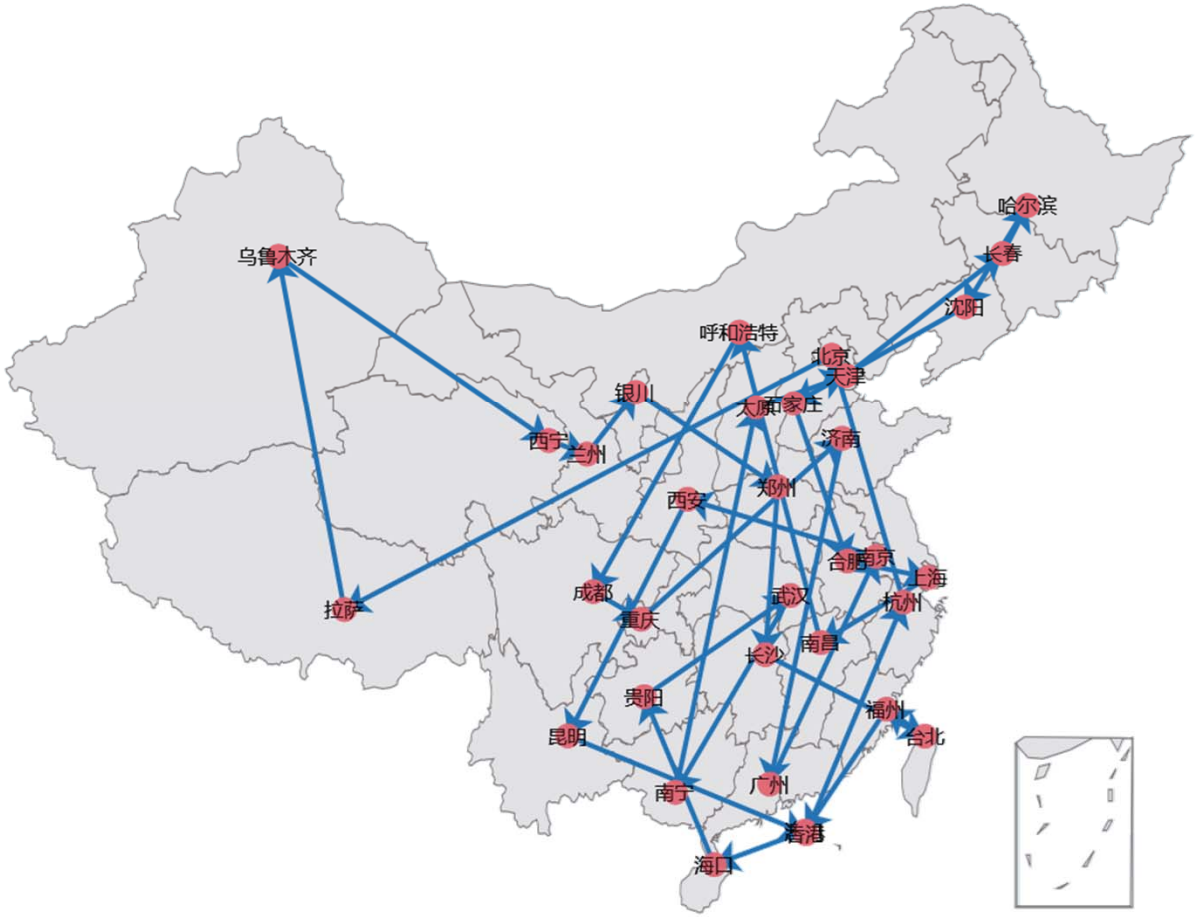
一个非确定性算法是一个两阶段的判定问题求解过程，包括：

① 非确定性猜想阶段：产生一个候选解

② 确定性验证阶段：　使用一个确定性算法来判断该候选解是否成立

如果对一个判定问题的任意实例，一个非确定性算法均能返回正确的判定结果，则称该算法解决了该问题。如果一个非确定性算法的验证阶段具有多项式时间复杂度，则称该算法是**非确定性多项式算法 (NP)**

- 猜想 + 验证的两阶段问题求解过程
- **NP 是指非确定性多项式**
  不是指非多项式



P 类问题

- 确定性算法在多项式时间可解的判定问题

NP 类问题

- 非确定性多项式算法可解的判定问题
- 对问题的一个猜想在多项式时间可验证

Class P

P 类问题是指可以通过确定性算法在多项式时间求解的判定问题。这一类问题称为**多项式问题**。

显然：$P \subseteq NP$

Class NP

NP 类问题是指可以通过非确定性多项式算法求解的判定问题。这一类问题称为**非确定性多项式问题**。

尚不清楚：$NP \subseteq P$

P=NP ?

多项式时间可归约

多项式时间可归约

两个判定问题 A 和 B，如果 A 的每一个实例都能通过一个函数转化为 B 的一个实例，则称 A 可归约为 B

如果该函数能够在多项式时间完成归约，则称 A 多项式时间可归约为 B

直观意义：A 不比 B 难

既然问题A能用问题B来解决，倘若B的时间复杂度比A的时间复杂度还低了，那求解A的算法就可以改进为求解B的算法，两者的时间复杂度还是相同

- 一个判定问题可以在多项式时间内转化为另一个判定问题

- A：求解一元一次方程
- B：求解一元二次方程
  - 每个一元一次方程都可以转化为一个一元二次方程
  - 因此一元一次方程可归约为一元二次方程
  - 如果能求解一元二次方程，就能求解一元一次方程
  - 而且求解一元一次方程不会比求解一元二次方程难

NP 完全问题 NP-complete

NP 完全问题

如果任何一个 NP 类问题都能在多项式时间归约为一个特定的 NP 类问题，则称该问题是一个 NP 完全问题

- 该问题自己属于 NP 类
- NP 类中任何一个问题都能在多项式时间归约为该问题
- NP 类中任何一个问题都不比该问题难
- 该问题是 NP 类中最难的

- 所有 *NP* 完全问题之间可以进行多项式时间归约
- 这些问题构成一个集合，它们之间等价
- 这个集合包含 *NP* 类中最难的问题

NP 难题 NP-hard

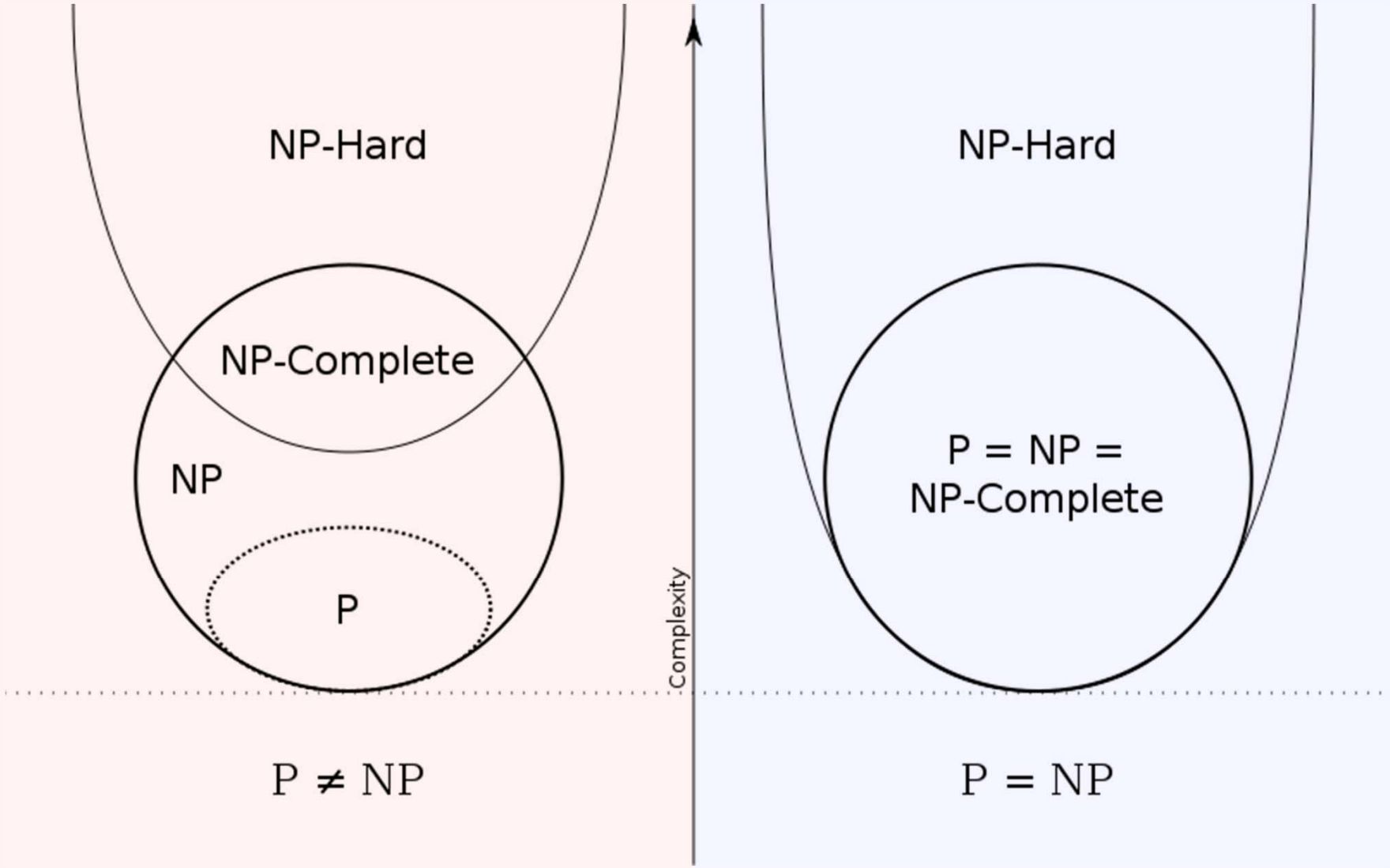
NP 难题

如果任何一个 NP 类问题都能在多项式时间归约为一个特定的问题，则称该问题是一个 NP 难题

- 该问题**自己不一定属于 NP 类**
- NP 类中任何一个问题都不比该问题难
- 该问题至少和 NP 类中最难的问题一样难

- NP 难题本身不一定属于 NP
- NP 难题至少和 NP 中最难的问题一样难
- 所有 NP 完全问题均可在多项式时间归约为 NP 难题
- NP 完全问题一定是 NP 难题

P, NP, NP 完全, NP 难题之间的关系



小节

算法复杂度

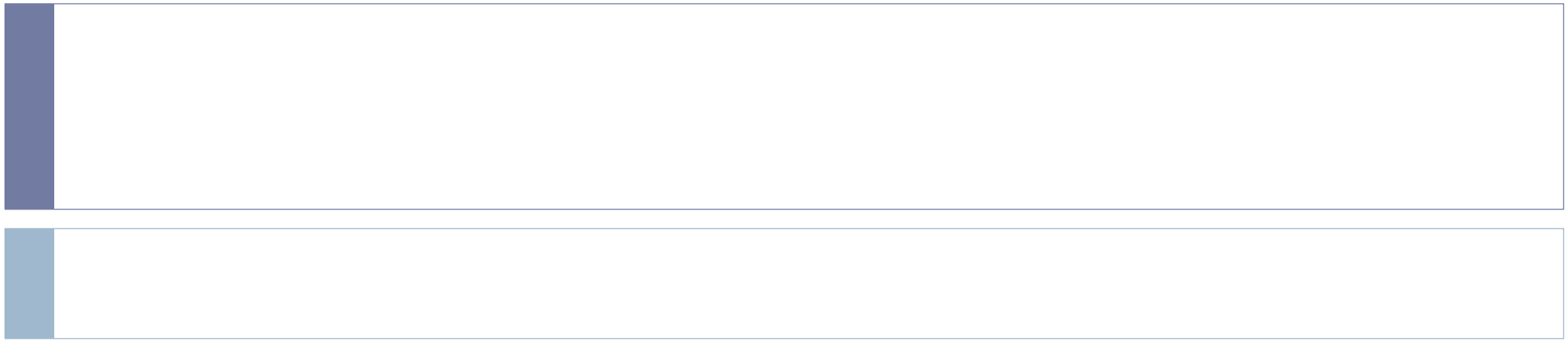
类别	名称	说明
1	**常数** Constant	最优的复杂度，运行效率与输入无关，非常罕见。一般来说当输入趋于无穷大时，算法执行时间总会趋于无穷大，需要的内存也会趋于无穷大
$\log n$	**对数** Logarithmic	不能考虑所有的输入元素，否则会成为线性复杂度
n	**线性** Linear	例如一重循环
$n\log n$	***n*-log-*n***	例如很多根据分治思想设计的算法
n^2	**二次** Quadratic	例如两重循环嵌套
n^3	**三次** Cubic	例如三重循环嵌套
2^n	**指数** Exponential	例如穷举集合的所有子集。“指数复杂度”泛指增长阶次为指数或更高的算法
$n!$	**阶乘** Factorial	例如求全排列。

问题复杂性

- P　是指确定性算法在多项式时间可解
- NP 是指非确定性多项式时间算法可解
- 为什么要研究 NP 完全性?
  - 能够快速验证也意味着有可能可以快速求解
  - 尚不清楚是否每一个NP类的问题都可以快速求解，但如果一个NP完全问题可以快速求解，则所有NP问题都可以快速求解

无信息搜索算法的复杂度

Complexity of Non-informative Searching Algorithms



宽度优先搜索的复杂度

宽度优先搜索仅能解决很小规模问题

ALGORITHM BREADTH-FIRST-SEARCH(problem)
INPUT A problem
OUTUT A solution or failure

```
node ← problem.INITIAL(PATH-COST = 0)
IF problem.IS-GOAL(node.STATE) THEN RETURN node
open   ← an FIFO queue with node inserted
closed ← an empty set
WHILE NOT open.EMPTY() DO
    node ← open.POP()
    closed.ADD(node.STATE)
    FOR EACH child IN problem.EXPAND(node) DO
        IF child.STATE is not in open or closed THEN
            IF problem.IS-GOAL(child.STATE) THEN
                RETURN child
            open.PUSH(child)
RETURN failure
```

- 设每个节点扩展出 b 个节点（分支因子）
- 目标位于第 d 层的最后一个节点
- 检查到目标结点时一共扩展出的节点为

$$b + b^2 + \cdots + b^d = O(b^d)$$

- **时间效率为指数复杂度**
  对每个节点执行各种操作的时间为常数
- **空间效率为指数复杂度**
  扩展的节点不在 open 表就在 closed 表

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

一致代价搜索的复杂度

一致代价搜索仅能解决很小规模问题

ALGORITHM UNIFORM-COST-SEARCH(problem)
INPUT A problem
OUTUT A solution or failure

```
node   ← problem.INITIAL(PATH-COST = 0)
open   ← a priority queue ordered by PATH-COST with node
closed ← an empty set
WHILE NOT open.EMPTY() DO
    node ← open.POP()
    IF problem.IS-GOAL (child.STATE) THEN
        RETURN child
    closed.ADD(node.STATE)
    FOR EACH child IN problem.EXPAND(node) DO
        IF  child.STATE is not in open or closed THEN
            open.PUSH(child)
        ELSE IF child.STATE in open with higher COST THEN
                replace that node with child
RETURN failure
```

- 设最优路径代价为 C^*
- 每一步行动代价至少为 ε
- 搜索到目标结点时一共扩展出的节点为

$$b + b^2 + \cdots + b^d = O(b^{1+\lfloor C^*/\varepsilon \rfloor})$$

- **时间效率为指数复杂度**
  设对每个节点执行各种操作的时间为常数
- **空间效率为指数复杂度**
  扩展的节点不在 open 表就在 closed 表
- 一直代价搜索扩展出的节点可能远多于宽度优先搜索

深度优先搜索的复杂度

深度优先搜索能够解决稍大规模问题

ALGORITHM DEPTH-FIRST-SEARCH(problem)
INPUT A problem
OUTUT A solution or failure

```
node ← problem.INITIAL(PATH-COST = 0)
open ← a stack with node inserted
WHILE NOT open.EMPTY() DO
    node ← open.POP()
    IF problem.IS-GOAL(node.STATE) THEN
        RETURN node
    IF NOT node.IS-CYCLE() THEN
        FOR EACH child IN problem.EXPAND(node) DO
            open.PUSH(child)
RETURN failure
```

- 设每个节点扩展出 b 个节点
- 搜索到第m层的最后一个节点时发现目标
- **时间效率为指数复杂度**

  检查到目标结点时一共扩展出的节点为

  $$b + b^2 + \cdots + b^m = O(b^m)$$

- **空间效率为线性复杂度**

  树搜索仅保存从根节点到叶节点的路径，以及该路径上未被扩展的兄弟节点（即Open表内容）

  $$b + b + \cdots + b = O(bm)$$

- **在空间效率上的巨大优势使得深度优先搜索成为很多领域的主力军**

牺牲最优性，用极少空间，使问题可解

深度受限搜索的复杂度

深度受限搜索能够解决稍大规模问题

ALGORITHM DEPTH-FIRST-SEARCH(problem, limit)
INPUT A problem
OUTUT A solution or failure

```
node ← problem.INITIAL(PATH-COST = 0)
open ← a stack with node inserted
WHILE NOT open.EMPTY() DO
    node ← open.POP()
    IF problem.IS-GOAL(node.STATE) THEN
        RETURN node
    IF node.DEPTH() > limit THEN
        result ← cutoff
    IF NOT node.IS-CYCLE() THEN
        FOR EACH child IN problem.EXPAND(node) DO
            open.PUSH(child)
RETURN result
```

- 设每个节点扩展出 b 个节点
- 目标所处的层次位于深度限制 l 之内
- **时间效率为指数复杂度**

  检查到目标结点时一共扩展出的节点为

  $$b + b^2 + \cdots + b^l = O(b^l)$$

- **空间效率为线性复杂度**

  树搜索仅保存从根节点到叶节点的路径，以及该路径上未被扩展的兄弟节点（即Open表内容）

  $$b + b + \cdots + b = O(bl)$$

- 在空间效率上有巨大优势

与深度优先搜索类似

如果选取得当，l 可能小于 m，实际上效率更高一些

迭代加深搜索的复杂度

迭代加深搜索兼具正确性复杂度优势

ALGORITHM ITERATIVEDEEPINGSEARCH(problem, limit)
INPUT A problem
OUTUT A solution or failure

```
FOR depth FROM 0 TO ∞ DO
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    IF result ≠ cutoff THEN
        RETURN result
RETURN failure
```

- 设每个节点扩展出 b 个节点
- 目标所处的最浅层次为 m
- **时间效率为指数复杂度**

  检查到目标结点时一共扩展出的节点为

  $$mb + (m-1)b^2 + \cdots + b^m = O(b^m)$$

- **空间效率为线性复杂度**

  树搜索仅保存从根节点到叶节点的路径，以及该路径上未被扩展的兄弟节点（即Open表内容）

  $$b + b + \cdots + b = O(bm)$$

- 兼具宽度优先搜索和深度优先搜索的优点

时间复杂度与宽度优先搜索类似
空间复杂度与深度优先搜索类似

无信息搜索算法对比

- 仅使用问题定义提供的信息
- 宽度优先搜索：　优先扩展最浅节点
- 一致代价搜索：　优先扩展最低代价节点
- 深度优先搜索：　优先扩展最深节点
  - 深度受限搜索：不扩展超过最大深度的节点
  - 迭代加深搜索：逐渐增加深度的深度受限搜索

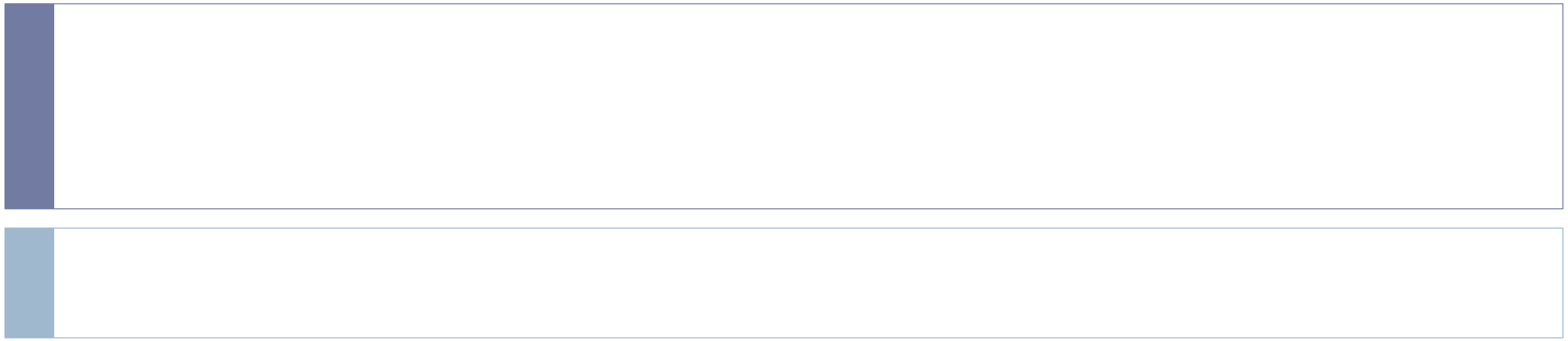
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

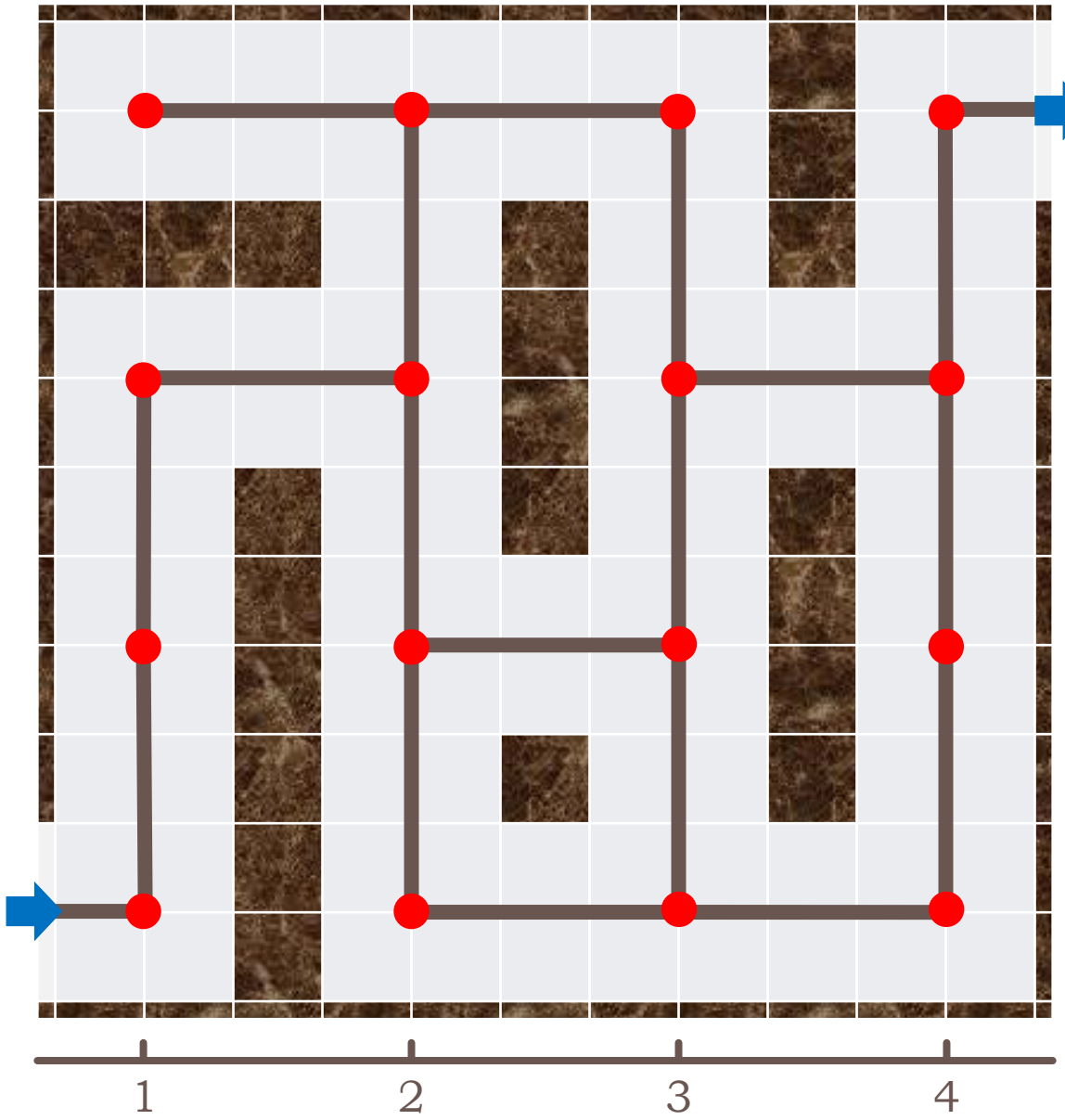
- **简单**
  容易理解
  容易实现
  不易出错
- **通用**
  与问题关系小
  方便进行重用
- **低效**
  比拼算力而不是智能
  难以求解大规模问题
- **起点**
  虽然本身不一定是好方法
  却是研究高效方法的起点

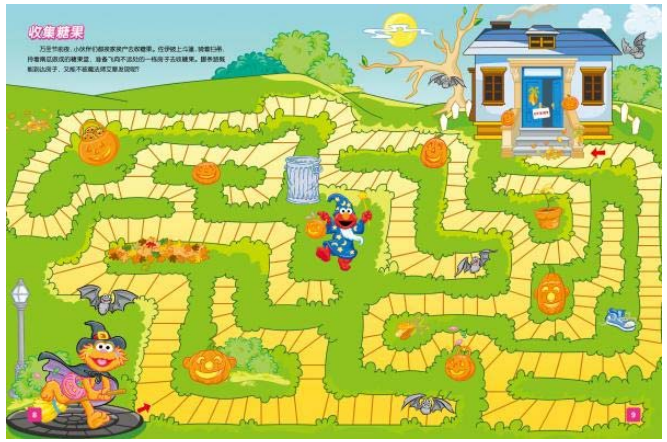
有信息搜索

Informed Search

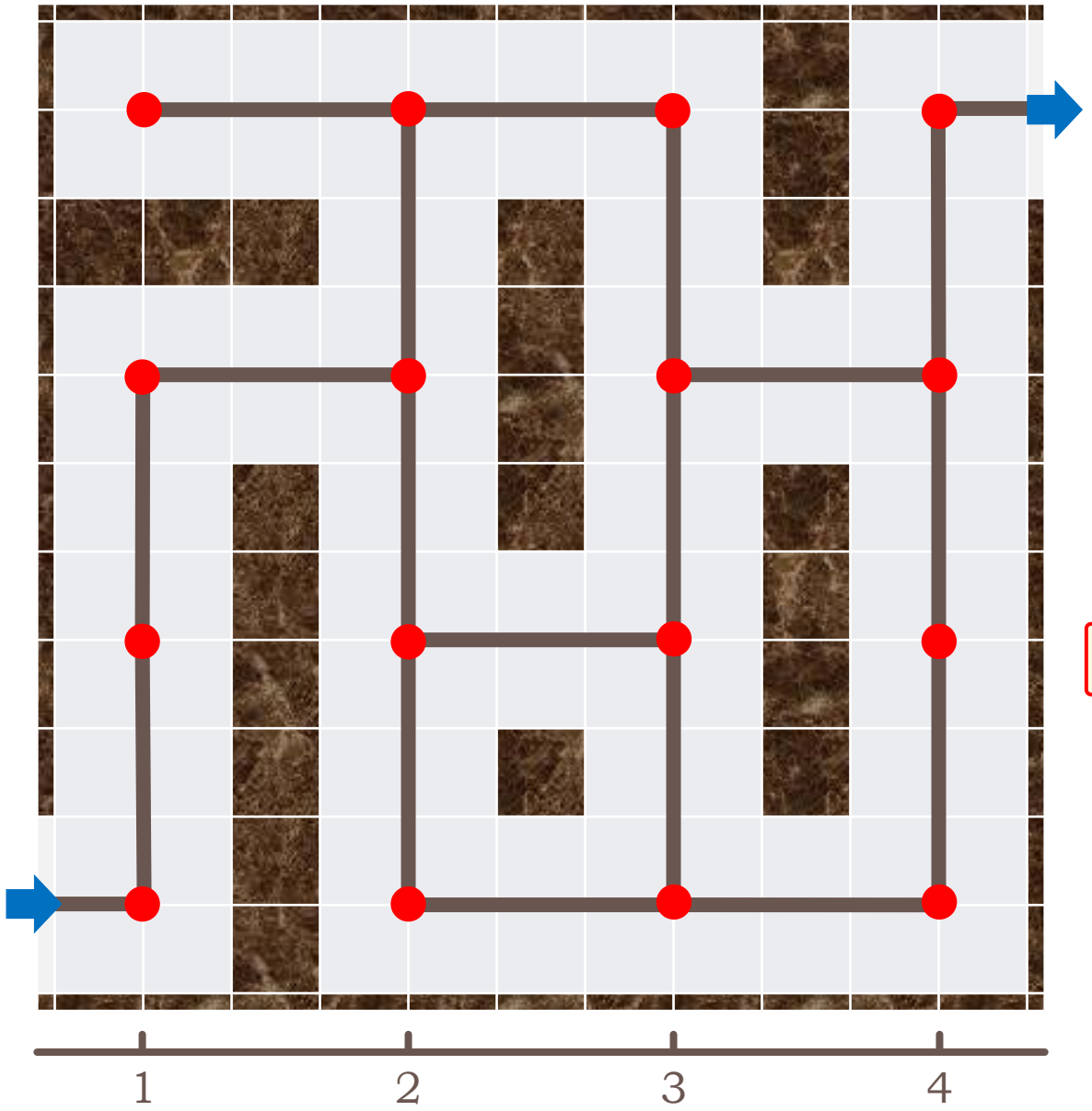


迷宫问题

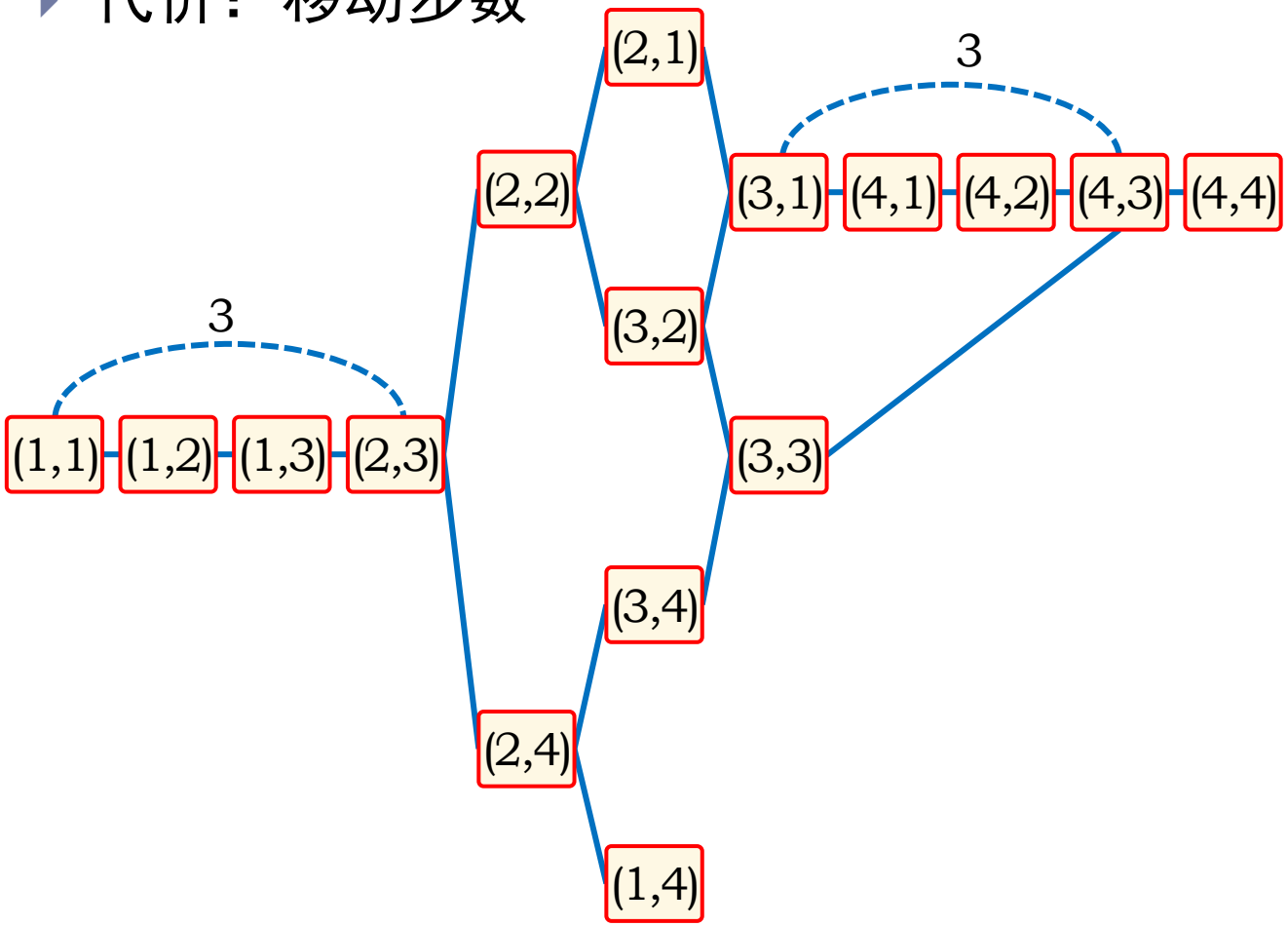




状态空间



- 状态：交叉点的坐标
- 行动：相邻交叉点之间的移动
- 代价：移动步数



一致代价搜索算法

ALGORITHM UNIFORM-COST-SEARCH(problem)
INPUT A problem
OUTUT A solution or failure

```
node   ← problem.INITIAL(PATH-COST = 0)
open   ← a priority queue with node ordered by PATH-COST g(n)
closed ← an empty set
WHILE NOT open.EMPTY() DO
    node ← open.POP()
    IF problem.IS-GOAL (child.STATE) THEN RETURN child
    closed.ADD(node.STATE)
    FOR EACH child IN problem.EXPAND(node) DO
        IF  child.STATE is not in open or closed THEN
            open.PUSH(child)
        ELSE IF child.STATE is in open with higher PATH-COST THEN
            replace that node with child
RETURN failure
```

注意：代价不能为负

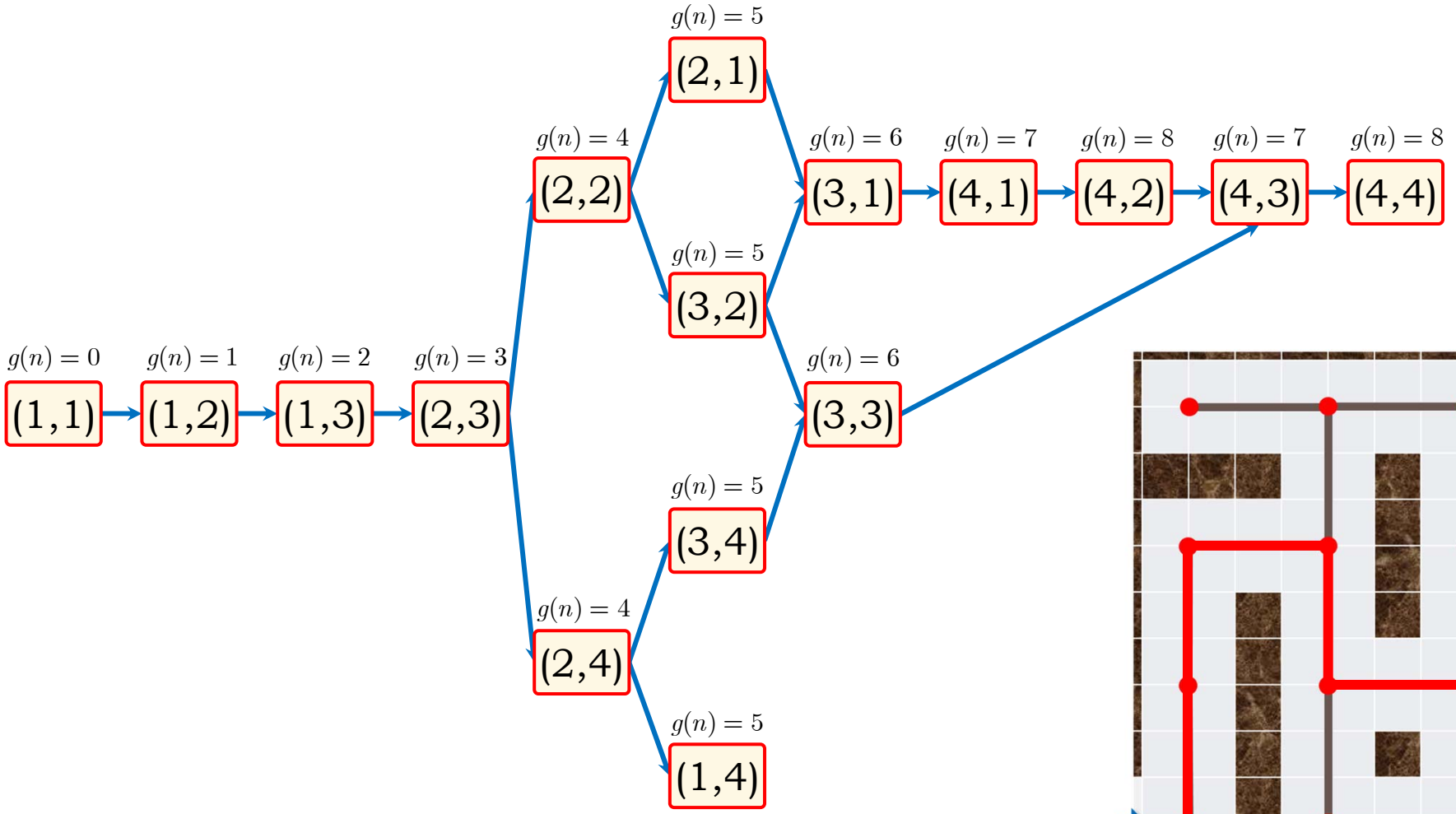
开节点表：优先队列
闭节点表：集合

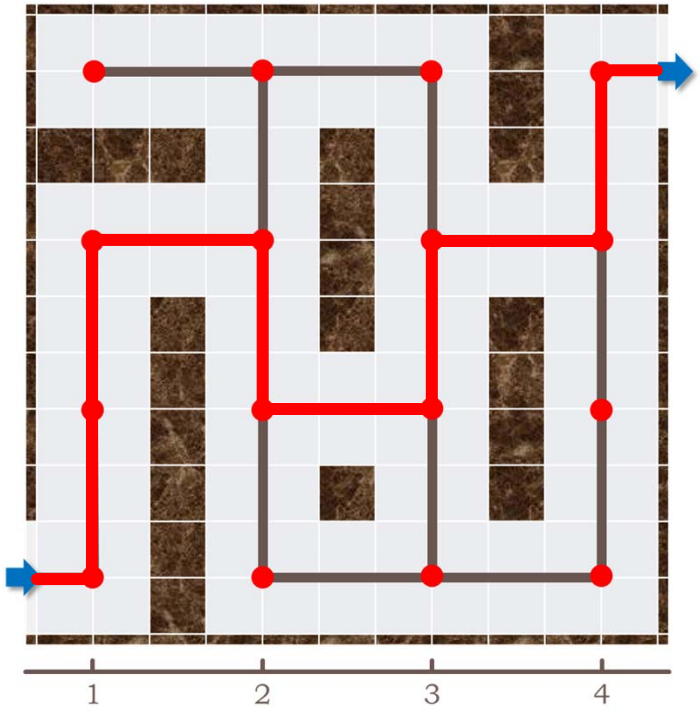
最优先出

顺序扩展

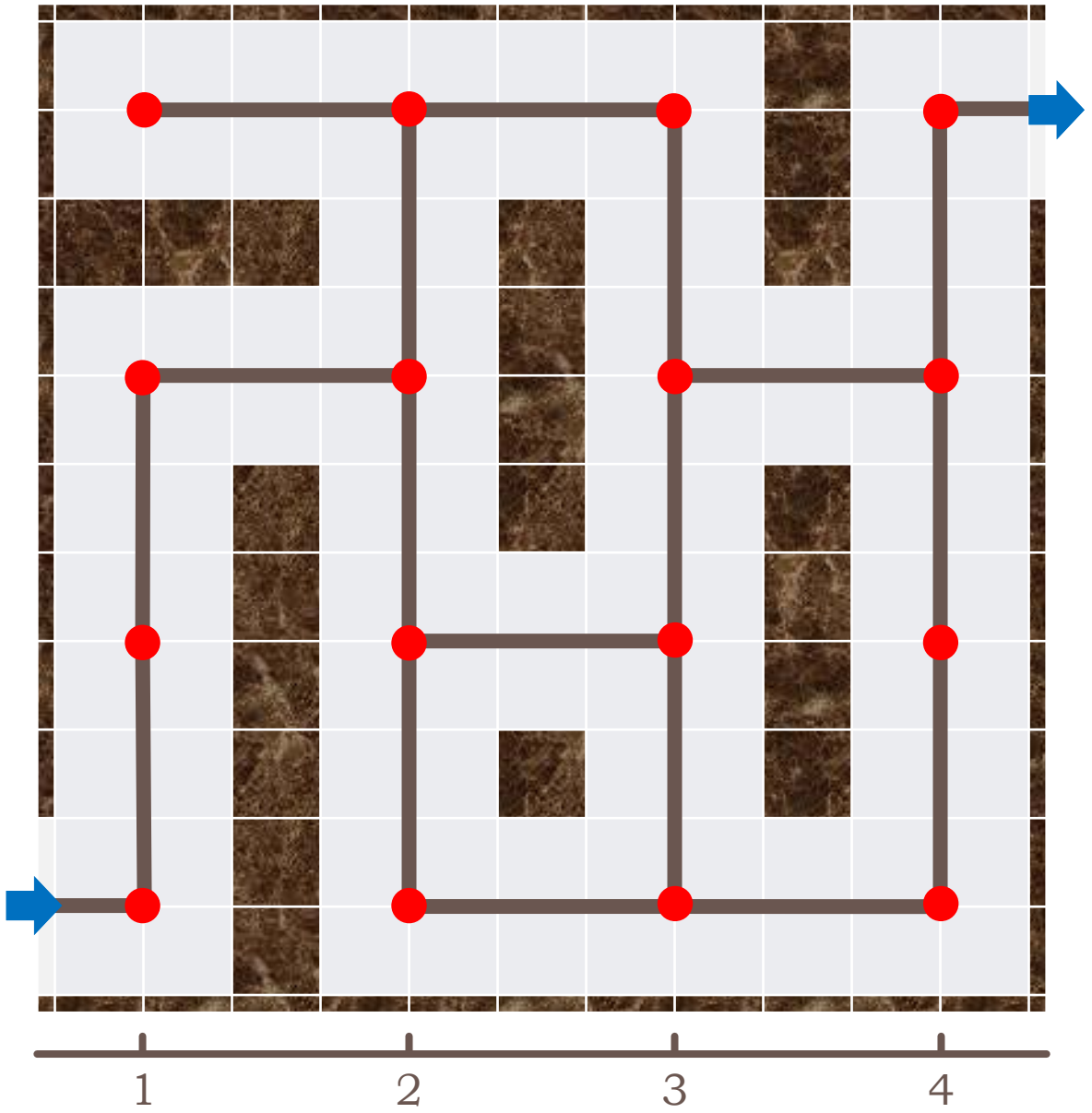
开节点表更新

解决迷宫问题


g(n) = 0
(1,1)
g(n) = 1
(1,2)
g(n) = 2
(1,3)
g(n) = 3
(2,3)
g(n) = 4
(2,2)
g(n) = 5
(2,1)
g(n) = 5
(3,2)
g(n) = 4
(2,4)
g(n) = 5
(3,4)
g(n) = 5
(1,4)
g(n) = 6
(3,1)
g(n) = 6
(3,3)
g(n) = 7
(4,1)
g(n) = 8
(4,2)
g(n) = 7
(4,3)
g(n) = 8
(4,4)


1
2
3
4

前瞻性



- 经验：离出口近的路口应该有较高优先级
  - 欧氏距离 $h(n)=\sqrt{(x_G-x_n)^2+(y_G-y_n)^2}$
  - 曼哈顿距离 $h(n)=|x_G-x_n|+|y_G-y_n|$
- 以该距离函数作为评价指标引导走向出口

open ← a priority queue ordered by PATH-COST $g(n)$

open ← a priority queue ordered by HEURISTIC-SCORE $h(n)$

- 该距离函数称为**启发函数**
- 启发函数体现了对**问题定义之外信息**的利用使搜索更具前瞻性
- 使用启发函数的搜索方法称为**最佳优先搜索**

启发函数

- 启发：引入与问题相关的领域经验知识
- 函数：**对当前状态最小代价的一个估计**
  - 当前节点到目标节点最小路径代价的度量
  - 当前节点到目标节点距离的度量
  - 当前节点可能的好坏程度的度量
- $h(n) \geq 0$
- $h(n)$ 越小表示 n 越接近目标
- $h(n) = 0$，目标节点

量化与问题相关的领域知识
将知识引入到搜索的过程中

- 经验：离出口近的路口应该有较高优先级
  - 欧氏距离 $h(n) = \sqrt{(x_G - x_n)^2 + (y_G - y_n)^2}$
  - 曼哈顿距离 $h(n) = | x_G - x_n | + | y_G - y_n |$
- 以该距离函数作为评价指标引导走向出口

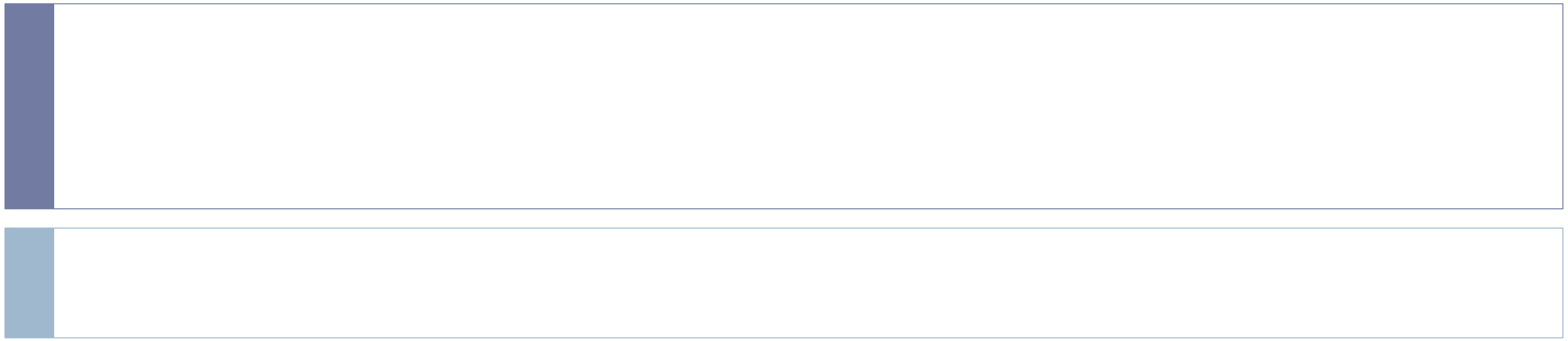
open ← a priority queue ordered by PATH-COST $g(n)$

open ← a priority queue ordered by HEURISTIC-SCORE $h(n)$

- 该距离函数称为**启发函数**
- 启发函数体现了对**问题定义之外信息**的利用使搜索更具前瞻性
- 使用启发函数的搜索方法称为**最佳优先搜索**

贪婪最佳优先搜索

Greedy Best-first Search



贪婪最佳优先搜索

把一致代价搜索算法的
代价函数换为启发函数

ALGORITHM GREEDY-BEST-FIRST-SEARCH(problem)
INPUT A problem
OUTUT A solution or failure

```
node ← problem.INITIAL(PATH-COST = 0)
open ← a priority queue with node ordered by HEURISTIC-SCORE h(n)
closed ← an empty set
WHILE NOT open.EMPTY() DO
    node ← open.POP()
    IF problem.IS-GOAL(child.STATE) THEN RETURN child
    closed.ADD(node.STATE)
    FOR EACH child IN problem.EXPAND(node) DO
        IF  child.STATE is not in open or closed THEN
            open.PUSH(child)
        ELSE IF child.STATE is in open with higher score THEN
            replace that node with child
RETURN failure
```

启发函数不能为负

开节点表：优先队列
闭节点表：集合

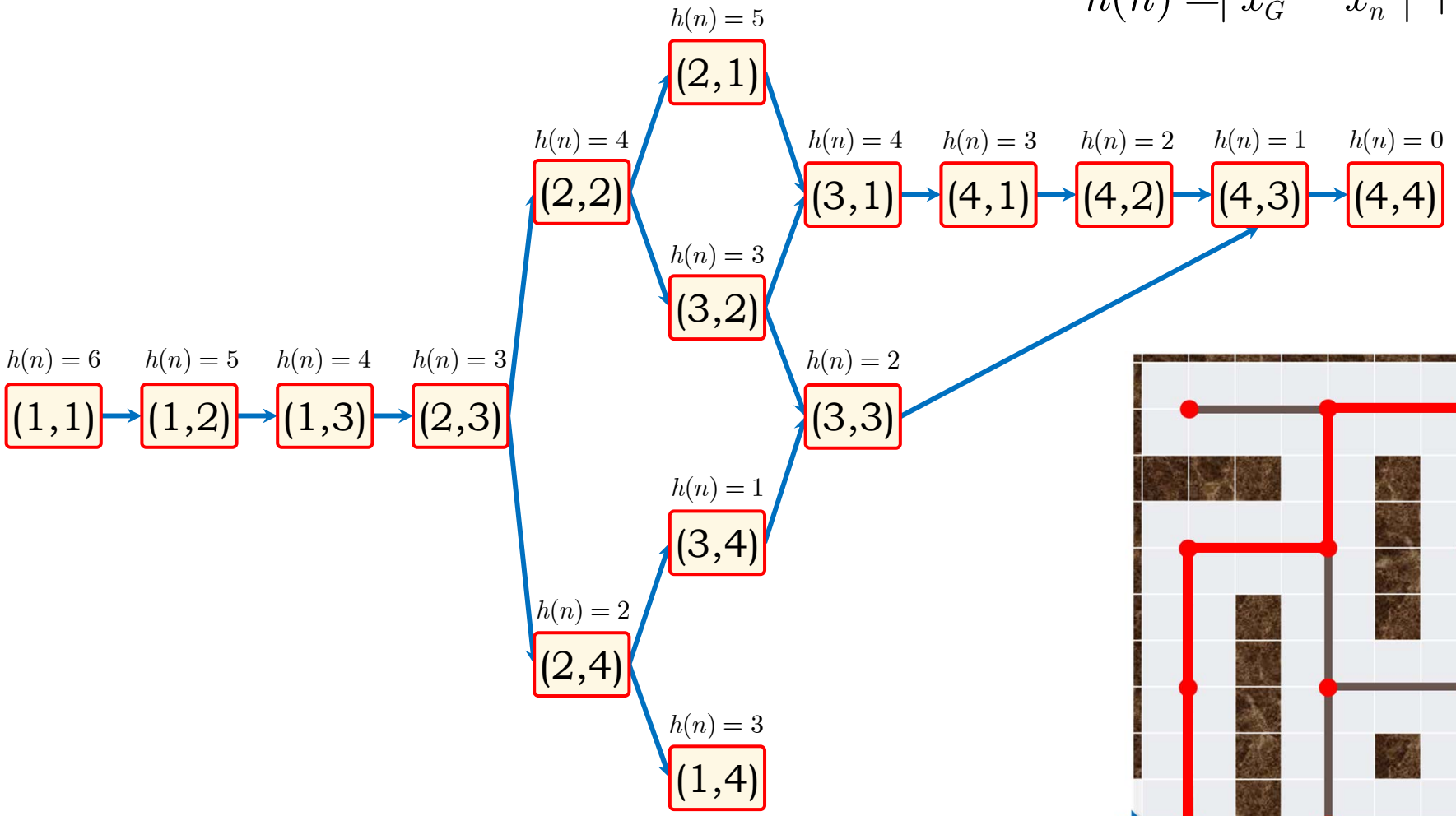
最优先出

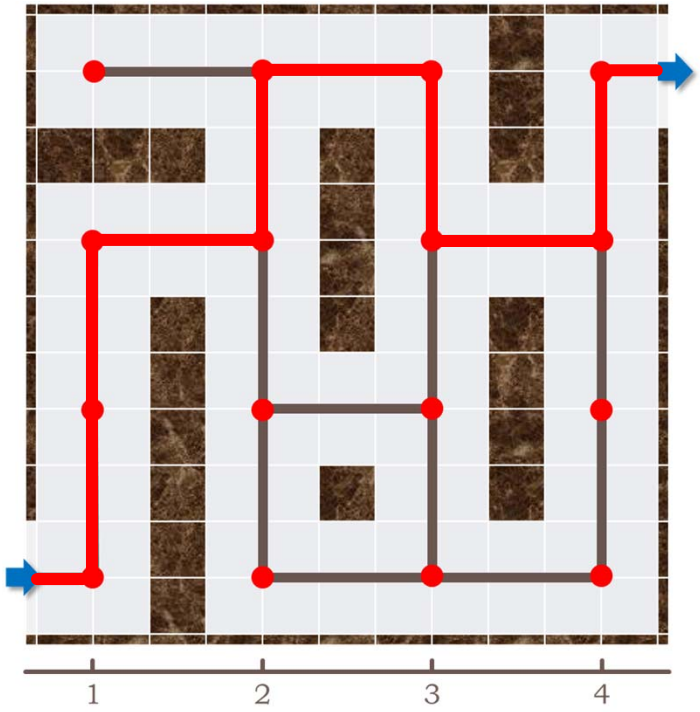
顺序扩展

开节点表更新

解决迷宫问题

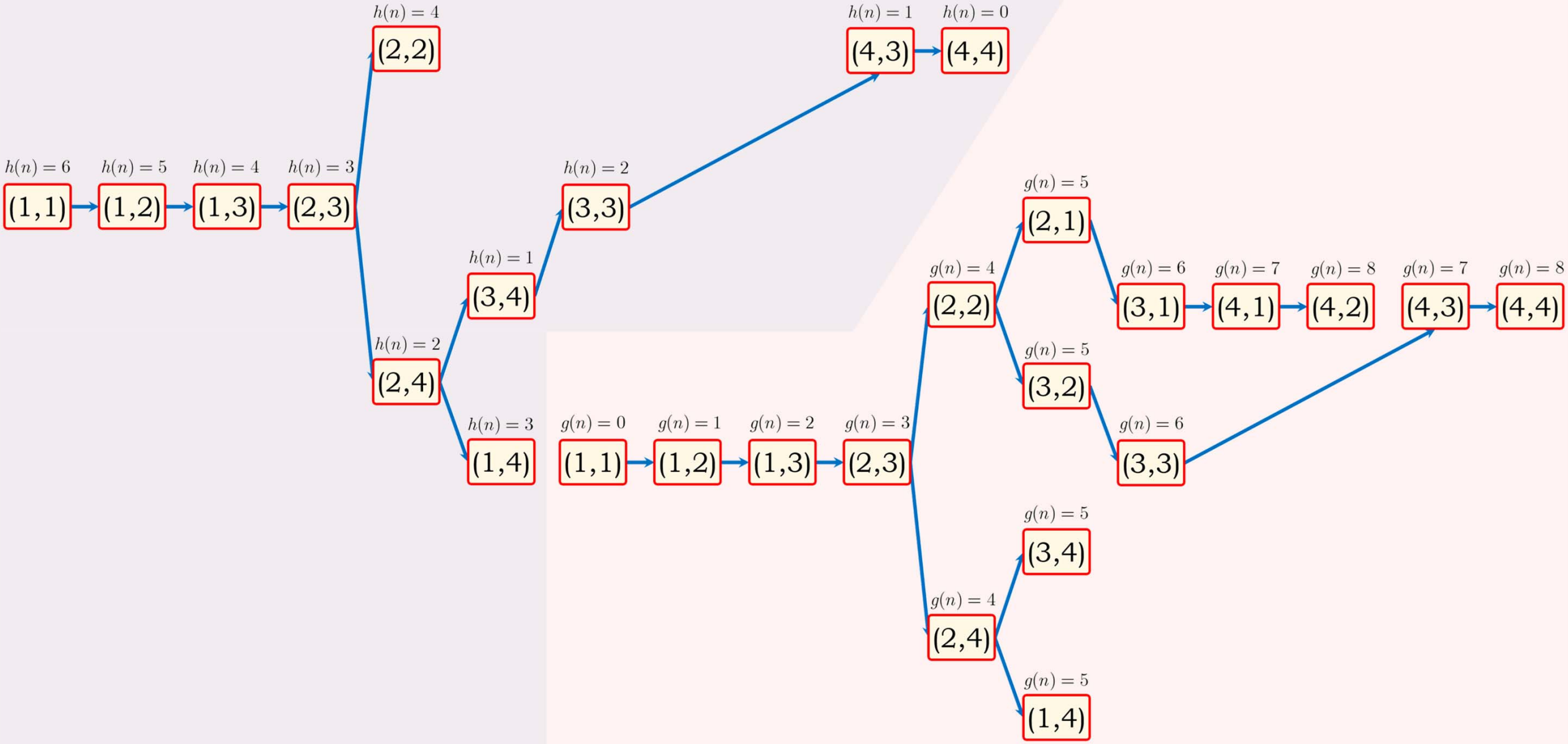
$$h(n) = | x_G - x_n | + | y_G - y_n |$$





贪婪最佳优先搜索

一致代价搜索



贪婪最佳优先搜索

- **仅使用启发函数** $h(n)$
- 完备性
  - 具有完备性（可以遍历所有节点）
- **最优性**
  - **不能保证找到问题的最优解**
- 时间效率
  - 指数复杂度 $O((b^*)^m)$
- 空间效率
  - 指数复杂度 $O((b^*)^m)$
- 分支因子通常较小，从而复杂度较低
- 经常表现很好，能很快找到解(不一定最优)

一致代价搜索

- **仅使用路径代价** $g(n)$
- 完备性
  - 具有完备性（可以遍历所有节点）
- **最优性**
  - **能够找到问题的最优解**（单步代价非负）
- 时间效率
  - 指数复杂度 $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
- 空间效率
  - 指数复杂度 $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$
- 分支因子通常较大，从而复杂度较高

贪婪最佳优先搜索

一致代价搜索

- 仅使用启发函数 $h(n)$
- 完备性
  - 具有完备性
- 最优性
  - 不能保证找到
- 时间效率
  - 指数复杂度
- 空间效率
  - 指数复杂度
- 分支因子通常
- 经常表现很好

- 仅使用路径代价 $g(n)$

节点）

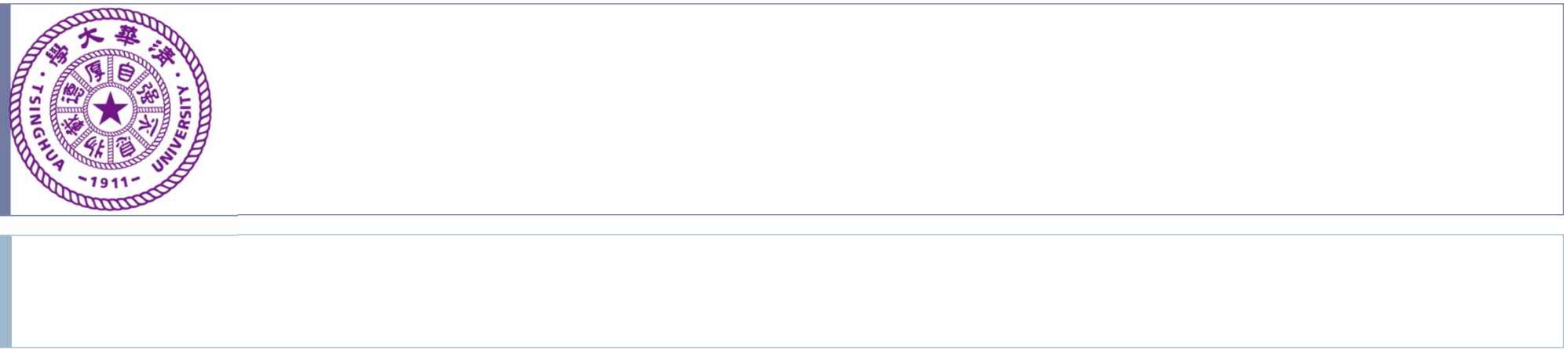
步代价非负）

复杂度较高

是否能够设计出一个搜索算法
兼具一致代价搜索的最优性和
贪婪最佳优先搜索较高的效率

A* 算法

A* Algorithm



A* 算法

把一致代价搜索算法的代价函数
换为路径代价与启发函数的加和

ALGORITHM A-STAR-ALGORITHM(problem)
INPUT A problem
OUTUT A solution or failure

```
node ← problem.INITIAL(PATH-COST = 0)
open ← a priority queue with node ordered by f(n) = g(n) + h(n)
closed ← an empty set
WHILE NOT open.EMPTY() DO
    node ← open.POP()
    IF problem.IS-GOAL (child.STATE) THEN RETURN child
    closed.ADD(node.STATE)
    FOR EACH child IN problem.EXPAND(node) DO
        IF  child.STATE is not in open or closed THEN
            open.PUSH(child)
        ELSE IF child.STATE is in open with higher score THEN
            replace that node with child
RETURN failure
```

启发函数不能为负
单步代价不能为负

开节点表：优先队列
闭节点表：集合

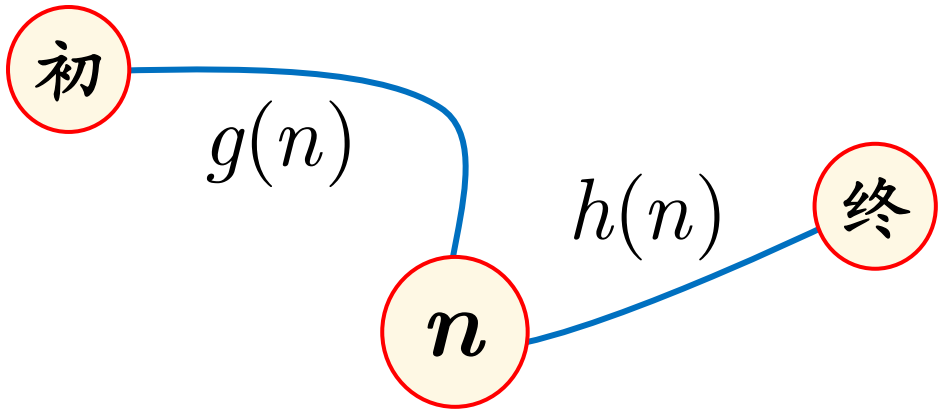
最优先出

顺序扩展

开节点表更新

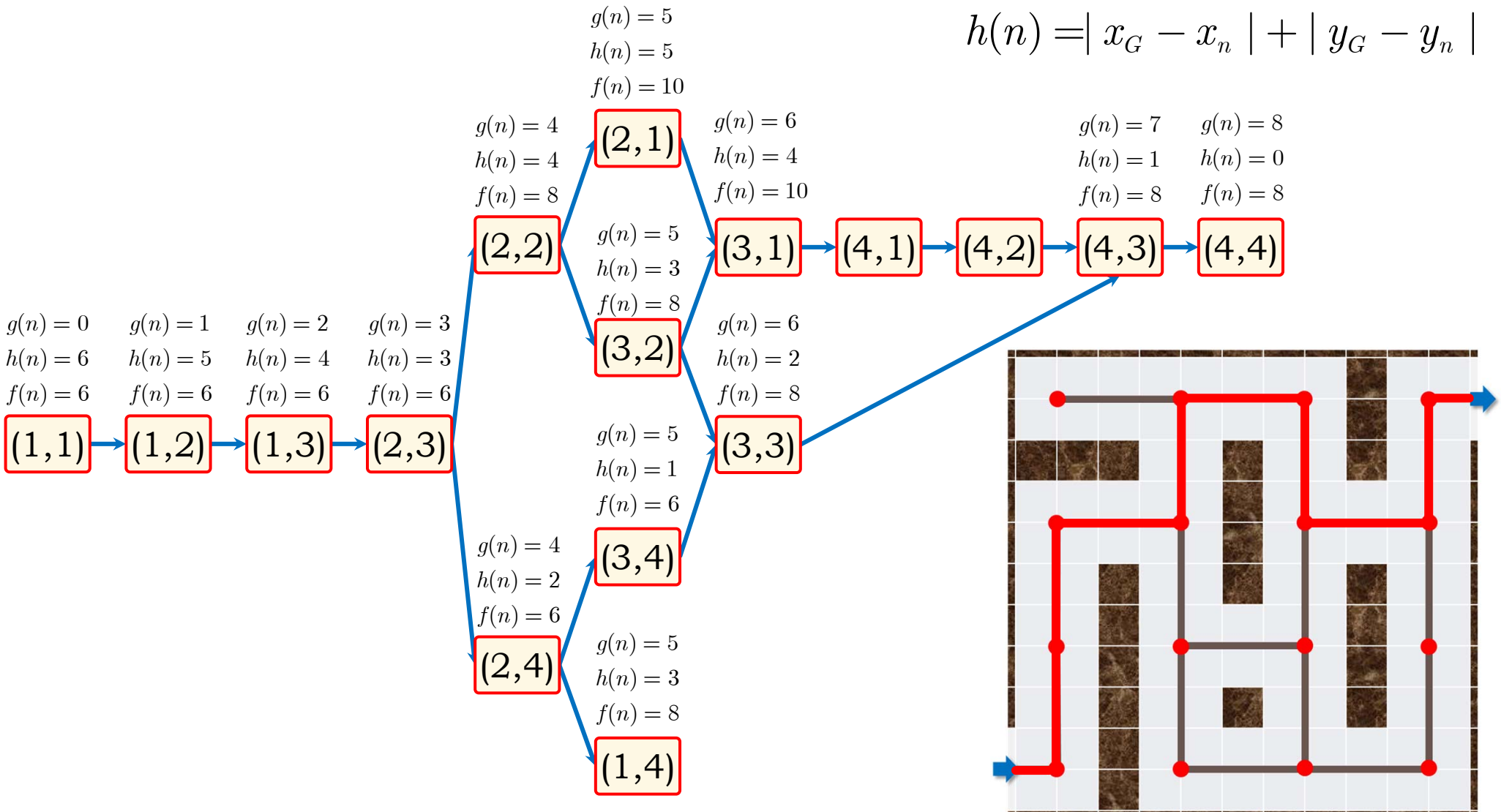
评价函数

- **路径代价：** $g(n)$，非负，g(初始节点)$= 0$
- 从初始节点到当前节点真实路径代价的计算值，来源于问题定义
- **启发函数：** $h(n)$，非负，h(目标节点)$= 0$
- 从当前节点到目标节点最小路径代价的估计值，来源于领域知识
- **评价函数：** $f(n) = g(n) + h(n)$，非负
- 从初始节点到目标节点最小路径代价的估计值，两个部分的加和



解决迷宫问题

$$h(n) = | x_G - x_n | + | y_G - y_n |$$



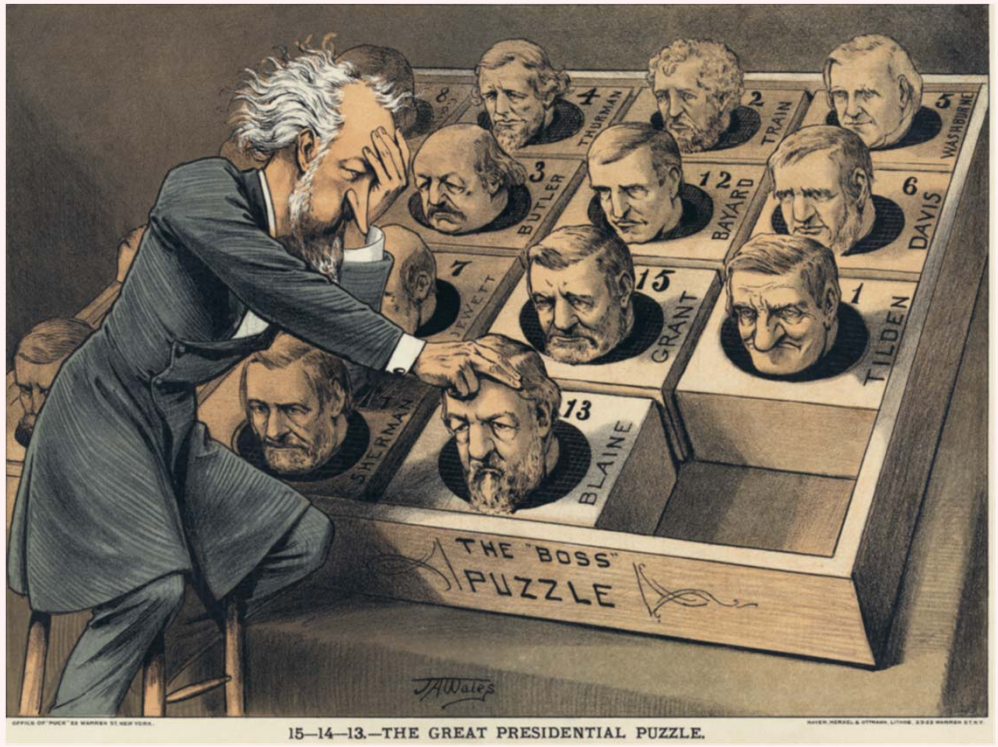
十五数码

十九世纪末美国风靡一时的社会现象

十五数码问题是解决问题的经典范例

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	



15—14—13.—THE GREAT PRESIDENTIAL PUZZLE.

Finding a Republican presidential candidate in 1880 is as hard as solving a 15 puzzle

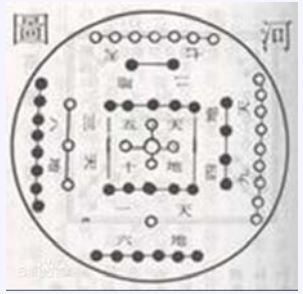
重排九宫

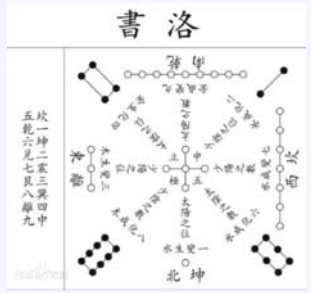
1	2	3
4	5	6
7	8	

1	2	3
4	5	6
7	8	

3×3的九宫格上有八个数字，每次可以将一个数字移到相邻的空格中。**如何移动数字才能将九宫格从初始状态变成最后的目标状态?**

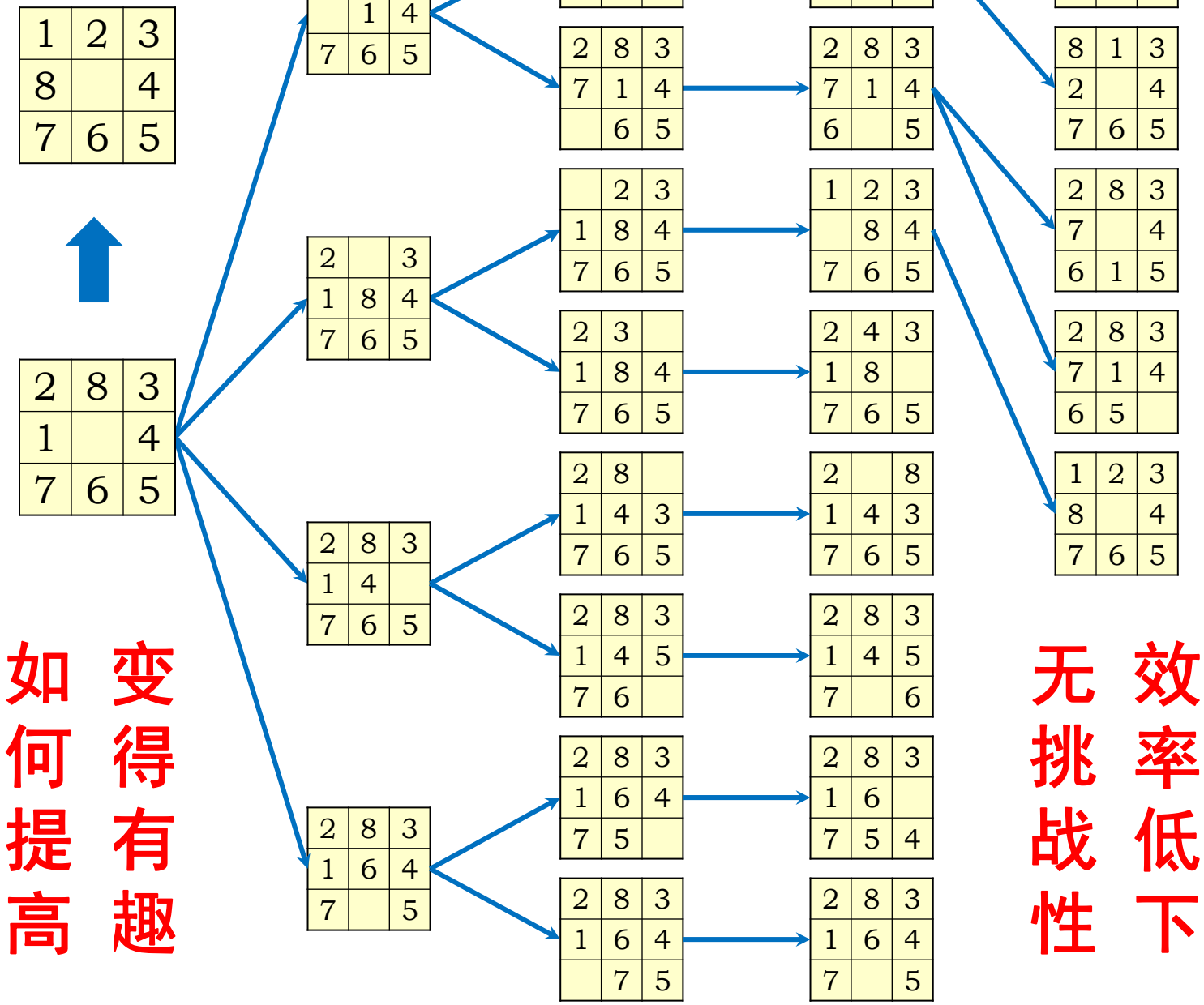
在我国可以追溯到远古神话历史时代的**河图**和**洛书**





2014年入选非物质文化遗产名录

宽度优先搜索



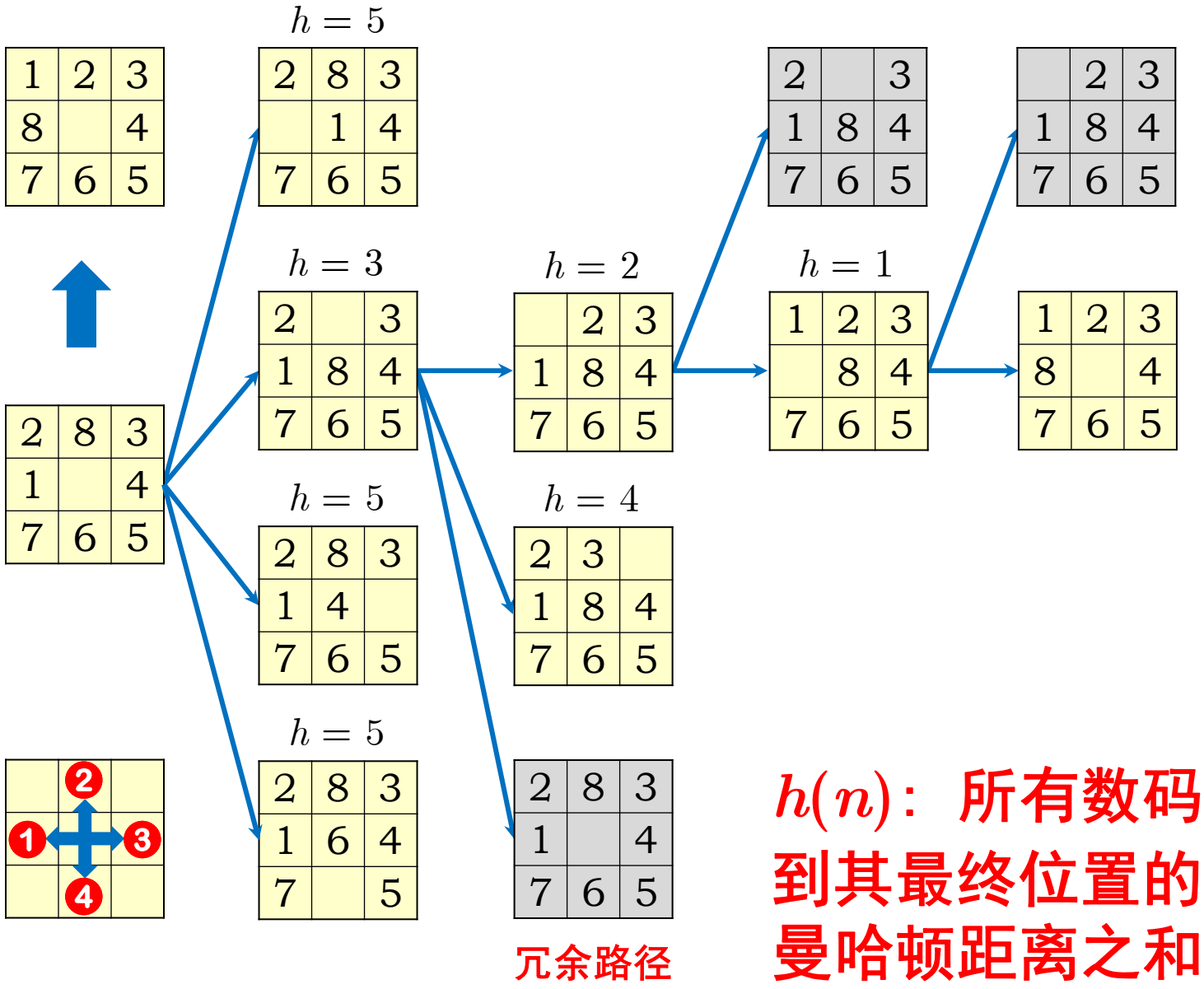
如何提高

变得有趣

无挑战性

效率低下

A* 算法



A* 算法的特点

一致代价搜索

- 仅使用路径代价 $g(n)$
- 完备性
  - 具有完备性
- 最优性
  - 能够找到问题的最优解
- 时间效率
  - 指数复杂度
- 空间效率
  - 指数复杂度
- 分支因子通常较大，从而复杂度较高

贪婪最佳优先搜索

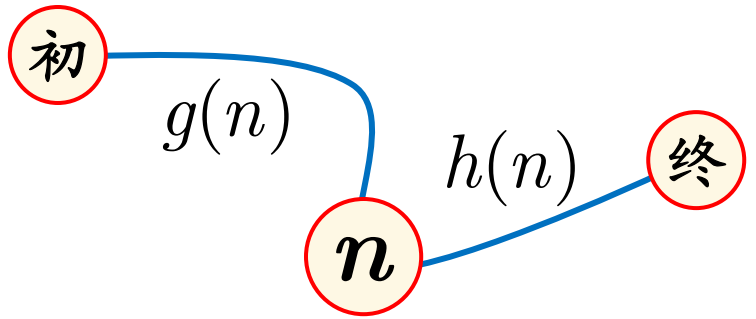
- 仅使用启发函数 $h(n)$
- 完备性
  - 具有完备性
- 最优性
  - 不能保证找到问题的最优解
- 时间效率
  - 指数复杂度
- 空间效率
  - 指数复杂度
- 分支因子通常较小，从而复杂度较低

A* 算法

- 使用路径代价与启发函数的和 $f(n) = g(n) + h(n)$
- 完备性
  - 具有完备性
- 最优性
  - 能够找到问题的最优解
- 时间效率
  - 指数复杂度
- 空间效率
  - 指数复杂度
- 分支因子通常较小，从而复杂度较低

A* 算法的最优性

- **最优性：如果 $h(n)$ 是可采纳的，则树搜索 A* 算法是最优的**
- **最优性：如果 $h(n)$ 是一致的，　则图搜索 A* 算法是最优的**



- 路径代价：$g(n)$，非负，g(初始节点)$= 0$
- 　　从初始节点到当前节点真实路径代价的计算值，来源于问题定义
- 启发函数：$h(n)$，非负，h(目标节点)$= 0$
- 　　从当前节点到目标节点最小路径代价的估计值，来源于领域知识
- 评价函数：$f(n) = g(n) + h(n)$，非负
- 　　从初始节点到目标节点最小路径代价的估计值，两个部分的加和

启发函数的可采纳性 Admissibility

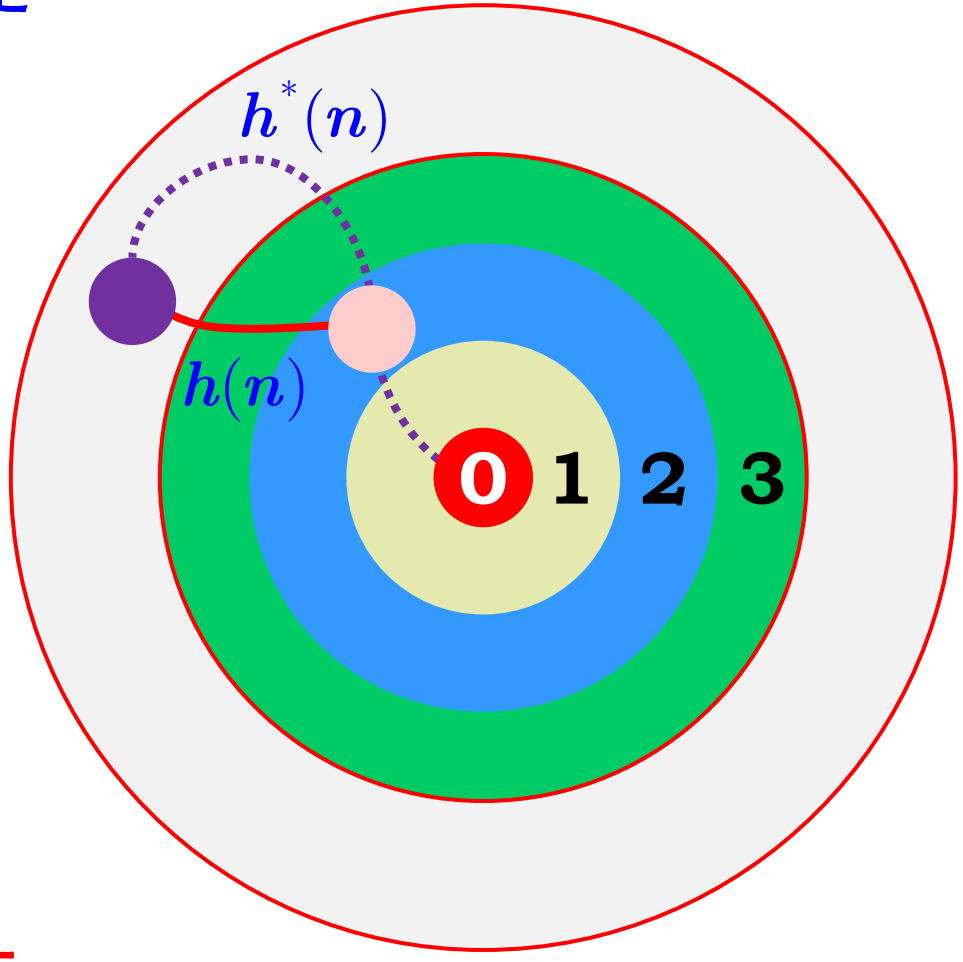
- 令 $h^*(n)$ 为节点 n 到目标的真实路径代价，则满足

$$h(n) \le h^*(n)$$

的启发函数 $h(n)$ 称为可采纳的。

- 可采纳的启发函数不会高估到达目标的代价
- $f(n)$ 不会超过经过节点 n 的解的真实代价 $C^*(n)$

$$f(n) = g(n) + h(n) \le g(n) + h^*(n) = C^*(n)$$



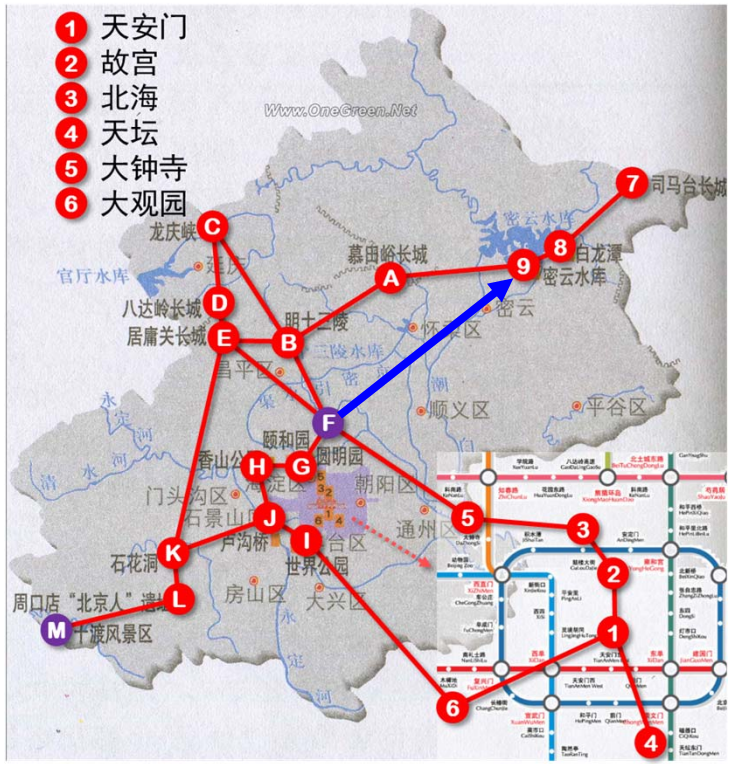
可采纳的启发函数是对真实代价的乐观估计

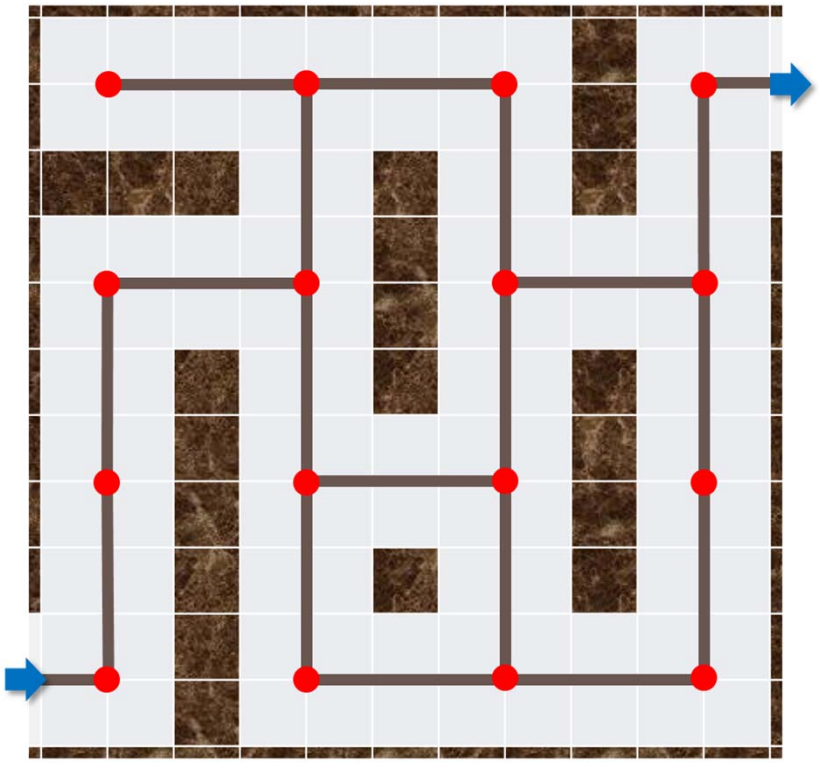
可采纳的启发函数

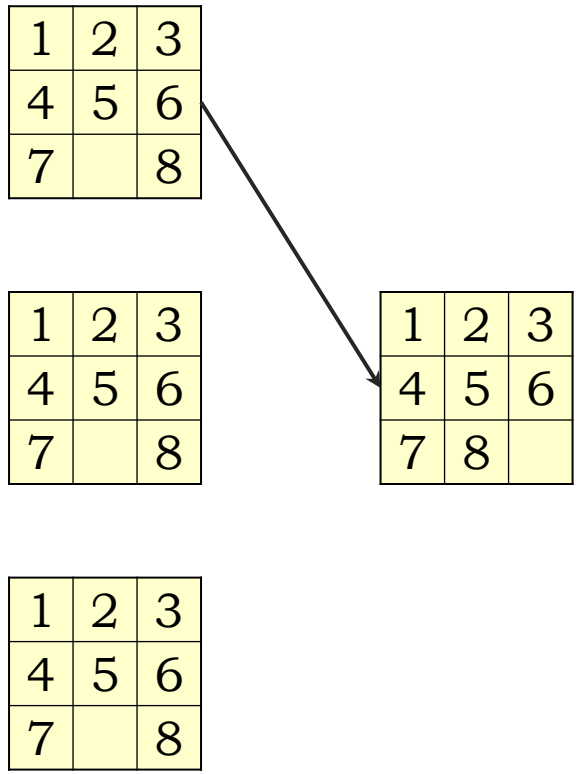
$$h(n) \leq h^*(n)$$

- 旅途规划： 景点到目标的欧式距离
- 迷宫探索： 所在点到出口的曼哈顿距离
- 重排九宫： 数码到其最终位置的曼哈顿距离之和
- …

为什么?







树搜索 A* 算法具有最优性

ALGORITHM A-STAR-ALGORITHM(problem)
INPUT A problem
OUTUT A solution or failure

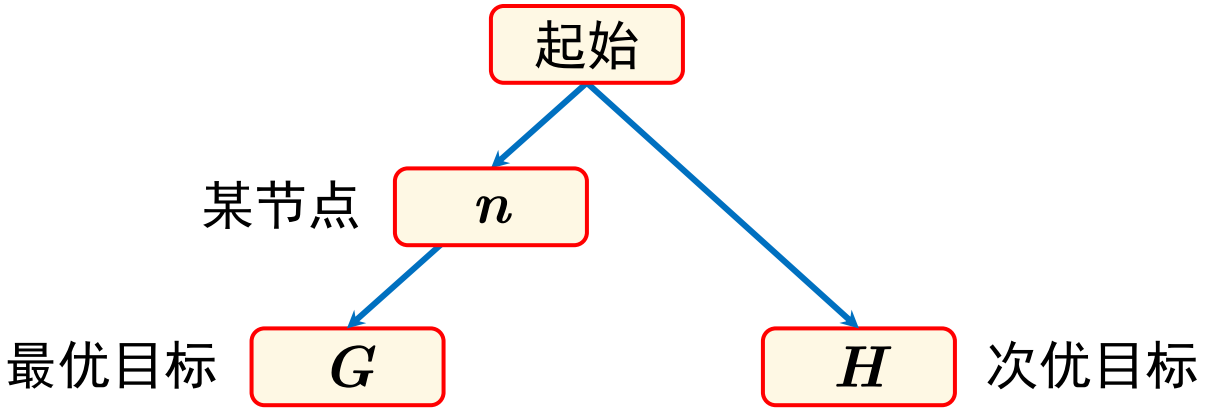
```
node ← problem.INITIAL(PATH-COST = 0)
open ← a priority queue ordered by f(n) = g(n) + h(n)
closed ← an empty set
open.insert(node)
WHILE NOT open.EMPTY() DO
    node ← open.POP()
    IF problem.IS-GOAL (child.STATE) THEN RETURN child
    closed.ADD(node.STATE)
    FOR EACH child IN problem.EXPAND(node) DO
        IF  child.STATE is not in open or closed THEN
            open.PUSH(child)
        ELSE IF child.STATE is in open with higher score THEN
            replace that node with child
RETURN failure
```

- **定理（A* 算法最优性）**
  如果 $h(n)$ 是可采纳的，则
  树搜索 A* 算法具有最优性

什么情况会使用树搜索的A*算法?

树搜索 A* 算法具有最优性

- **定理：如果 $h(n)$ 可采纳，则树搜索 A* 算法具有最优性**
- 说明：假设某次优目标 H 早先被生成并置于开节点表中
  节点 n 是任意处于通往最优目标 G 的路径上的点
  已经被插入到开节点表中但尚未被选中进行扩展



$$
\begin{aligned}
&h(G)=h(H)=0 && \Rightarrow f(H)>f(G)\\
&h(n)\le h^*(n) && \Rightarrow f(G)=g(n)+h^*(n)\ge g(n)+h(n)=f(n)\\
& && \Rightarrow f(H)>f(n)
\end{aligned}
$$

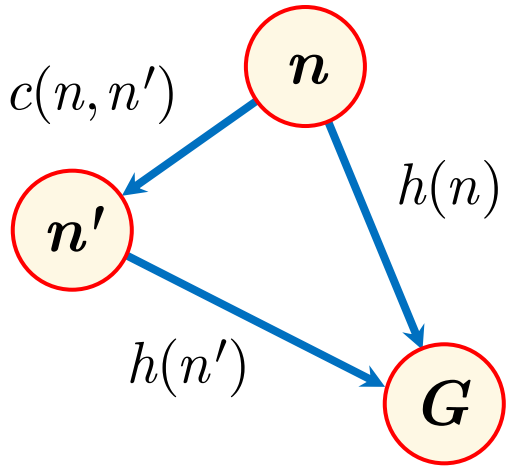
A* 永远不会扩展次优目标节点

启发函数的一致性 Consistency

- 对每个节点 n 及其任意后继节点 n'，满足

$$h(n) \le c(n, n') + h(n')$$

的启发函数 $h(n)$ 称为一致的（单调的）。



- 旅途规划：景点到目标的欧式距离
- 迷宫探索：所在点到出口的曼哈顿距离
- 重排九宫：数码到其最终位置的曼哈顿距离之和
- …

为什么？

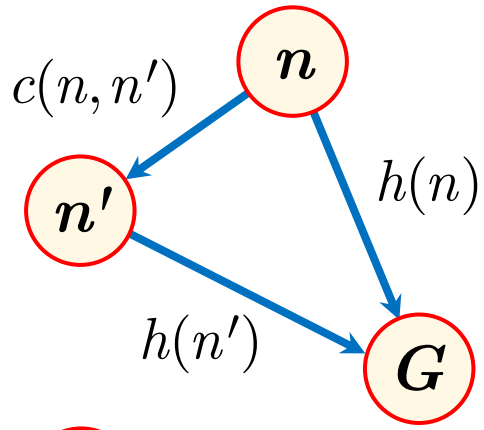
一致的启发函数满足三角不等式

一致的启发函数是可采纳的

- 对每个节点 n 及其任意后继节点 n'，满足

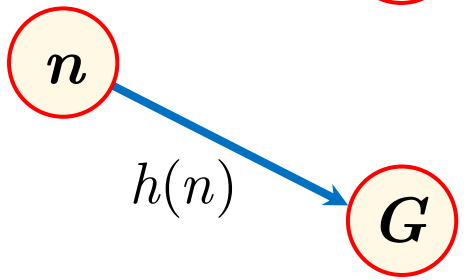
$$h(n) \le c(n,n') + h(n')$$

的启发函数 $h(n)$ 称为一致的（单调的）。



- 一步就能到目标的节点是可采纳的

$$h(n) \le c(n,G) + h(G) = h^*(n)$$

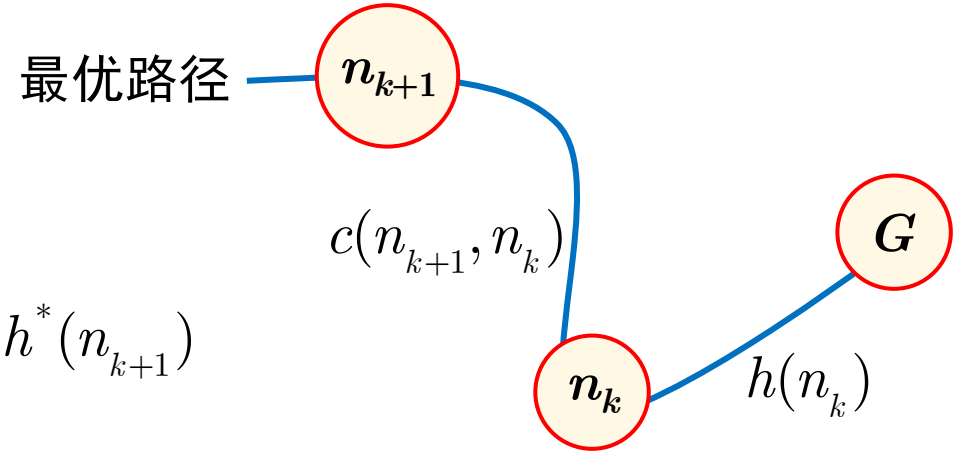


- 设 k 步能到目标的节点是可采纳的

$$h(n_k) \le h^*(n_k)$$

- 则 $k+1$ 步能到目标的节点也是可采纳的

$$h(n_{k+1}) \le c(n_{k+1},n_k) + h(n_k) \le c(n_{k+1},n_k) + h^*(n_k) = h^*(n_{k+1})$$

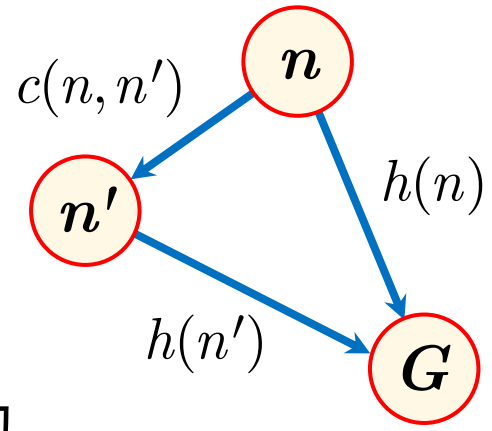


一致的启发函数产生非递减的代价评估

- 对每个节点 n 及其任意后继节点 n'，满足

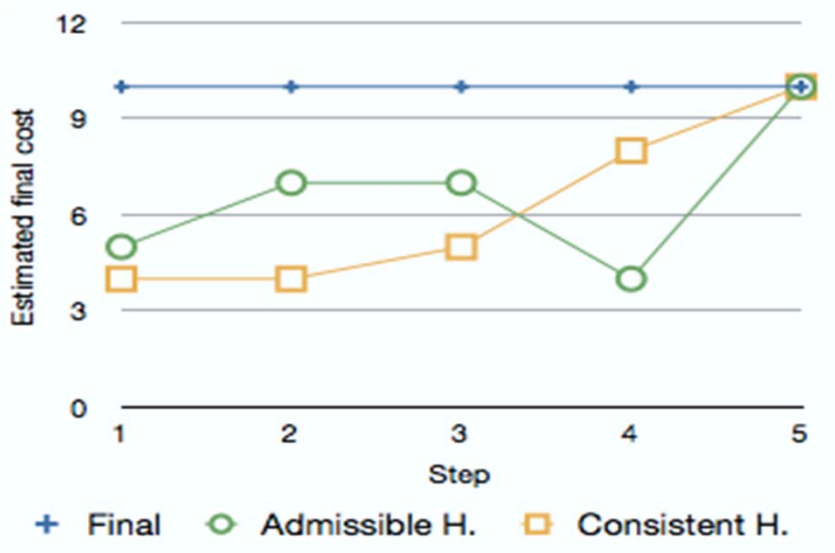
$$h(n) \leq c(n, n') + h(n')$$

的启发函数 $h(n)$ 称为一致的（单调的）。



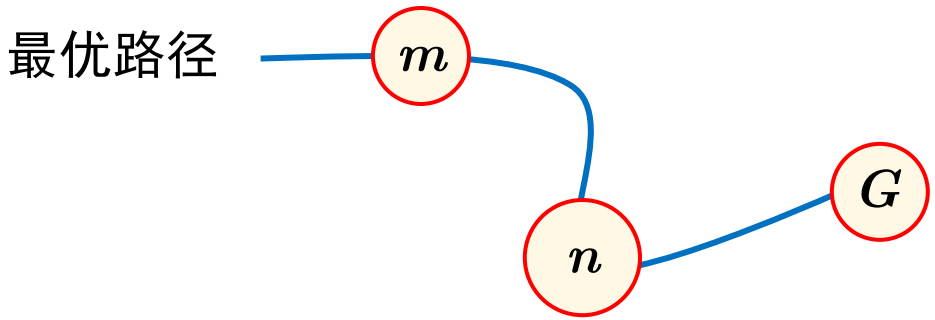
- 采用一致的启发函数，沿着任何路径的 $f(n)$ 值是非递减的

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$



图搜索 A* 算法的最优性

- **定理：如果 $h(n)$ 是一致的，则图搜索 A* 算法是最优的**
- 说明：A* 扩展节点 n 时，已发现到达该节点的最优路径
  - 考查从初始状态到该节点的最优路径
  - 由于在一致启发函数下代价评估的单调性
  - 该路径上每个节点的代价评估均小于 $f(n)$
  - 因此，该路径上的节点已经全部被扩展
  - 换言之，该最优路径已被发现



如果启发函数不满足一致性，则需要一些额外的工作保证 A* 算法的最优性。简单地说，就是需要监控闭节点表，将获得更小代价评估的节点重新放回到开节点表中。 **代价很大，尽量避免**

图搜索 A* 算法的高效性

- A* 算法不扩展 $f(n) > C^*$ 的节点，实际作用是**剪枝**
- A* 算法在所有从初始状态开始搜索的算法中效率最优
- 绝对误差： $\Delta = h^* - h$ h^*：从初始状态到目标状态的实际代价
- 相对误差： $\varepsilon = (h^* - h) / h^*$
- 时间效率：指数复杂度 $O(b^{\Delta})$ $O(b^{\varepsilon d})$ $O((b^{\varepsilon})^{d})$
- 空间效率：指数复杂度
- 与宽度优先搜索一样，求解大规模问题时会耗尽内存

虽然有信息搜索可以节省大量时间空间

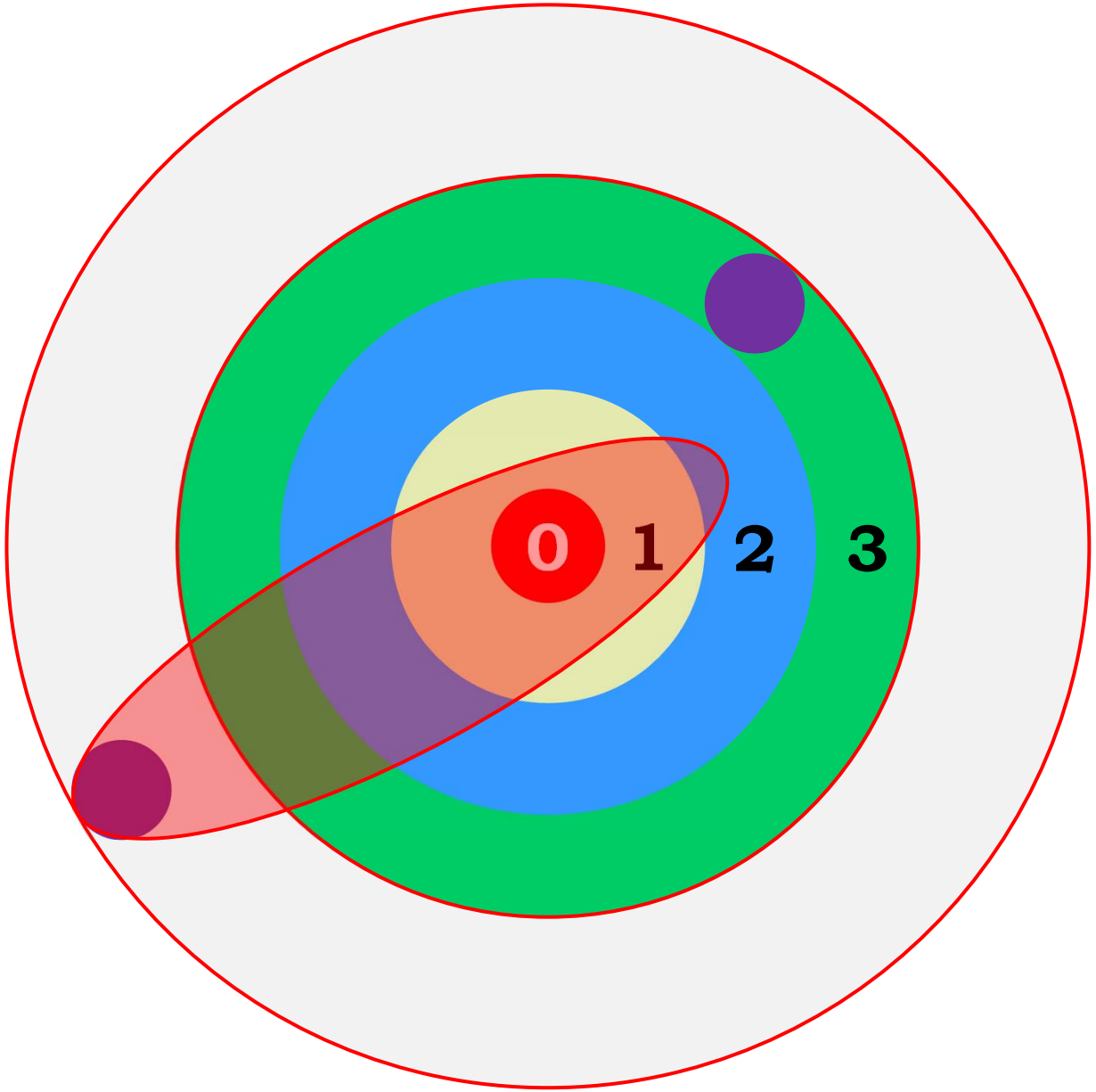
对大规模问题坚持找最优解仍然不可行

聪明的搜索

- 宽度优先搜索
- 迭代加深搜索

- 深度优先搜索

- A* 算法



Thank you very much!