中国科学技术大学

University of Science and Technology of China

人工智能讲义

问题求解：搜索

March 8, 2018

Outline

Outline

图/网络的最短路径问题

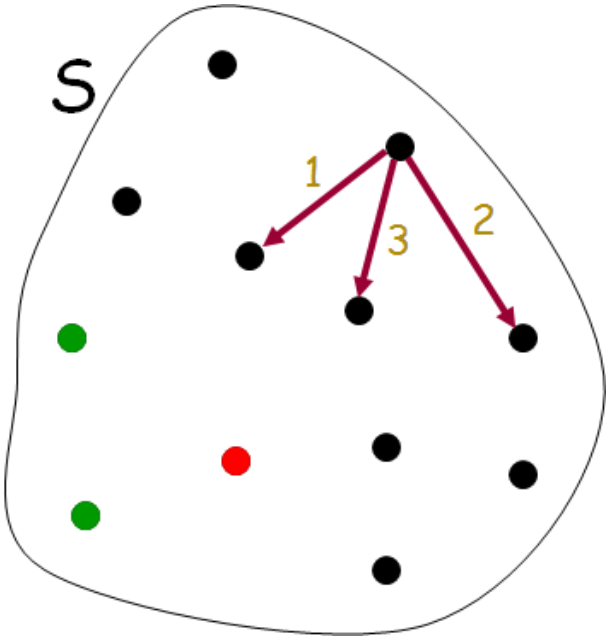
给定一个图 $\mathbb{G} = (V, E)$

- 指定起点/源点为 I，终点/目标为 G 时：
- 若边无权或边权都相等时，求 I 到 G 的最短路径，即边数最少的路径。
- 若边上有不完全相同的权值，求 I 到 G 的最短路径，即路径上边权值和最小的路径。
- 来自数据结构课程的结论：
  - Dijkstra 算法：计算节点 I 到其他所有节点的最短路径。主要特点是以起点 I 为中心向外层层扩展，直到扩展到终点为止。时间复杂度 $O(n^2)$
  - Bellman-Ford 算法：单源负权边，时间复杂度 $O(ne)$
  - Floyd 算法：多源，时间复杂度 $O(n^3)$，空间复杂度 $O(n^2)$

添加新的约束条件，获得新问题

- 算法时间复杂度为 $O(p(\log n))$，其中 $p(\cdot)$ 表示多项式函数。
- WHY? 现实中大量问题，图的节点数目非常多，是问题规模的指数函数。
- 算法? AI 中的搜索算法! 不能/无法遍历所有的节点。
- 所以我们说 AI 中的搜索就是有时间约束的最短路径问题。

搜索的定义



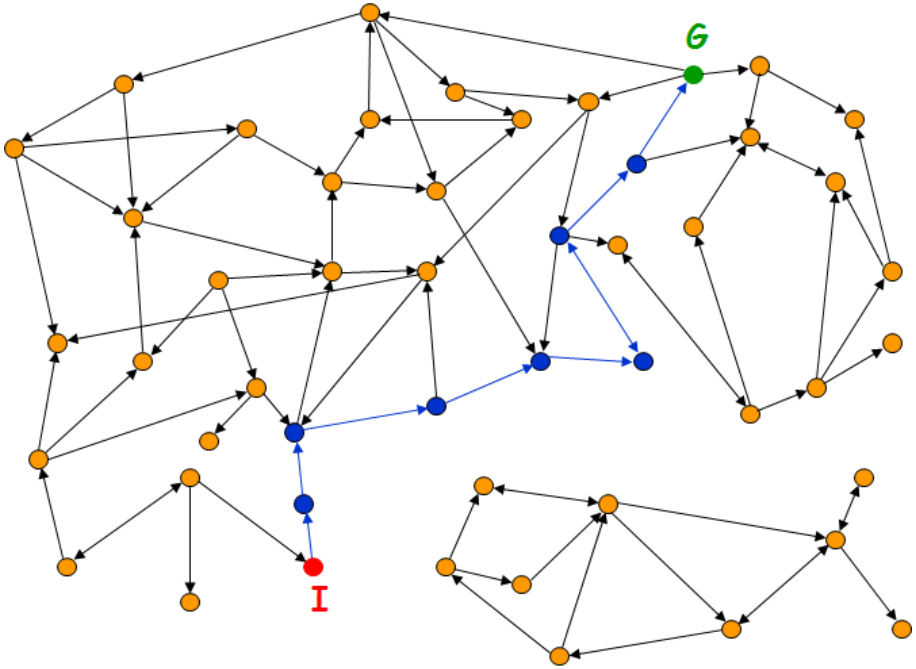
搜索：基于“状态”的定义

- 状态空间：$\mathbb{S}$
- 后继函数：$successors : \mathbb{S} \to 2^{\mathbb{S}}$
- 初始状态/初态：s_0
- 目标测试：
  $GOAL : \mathbb{S} \to \{T, F\}$
- 路径耗散/arc cost：

搜索问题的五要素

- 上述五个组成要素，描述了一个搜索问题。

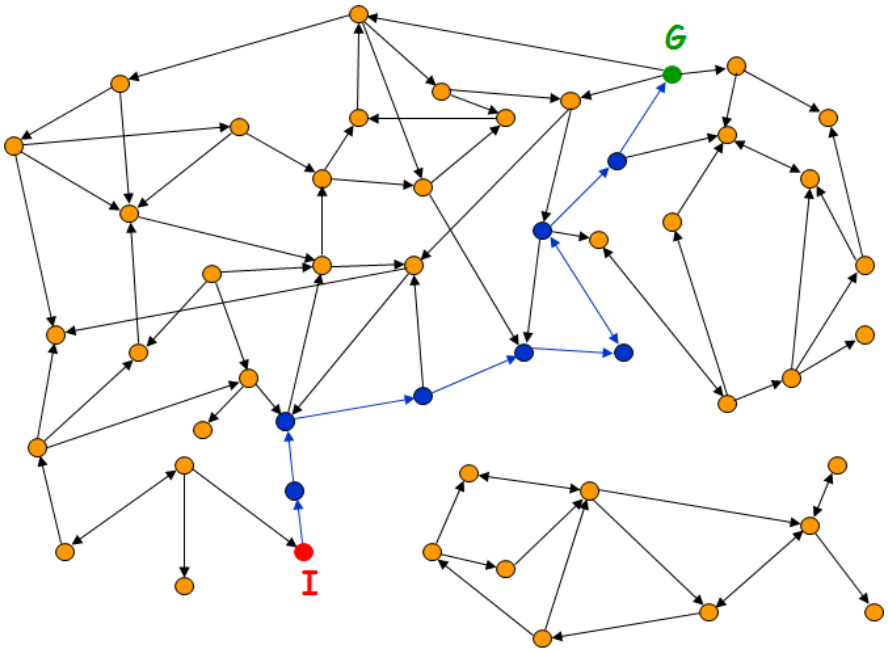
搜索问题与状态图



用状态图来描述搜索问题

- 每个节点表示一个状态
- 弧及其两个端点表示后继函数
- 可能包括多个不连通的分量
- 标注上初态 I 和终态 G，如图所示

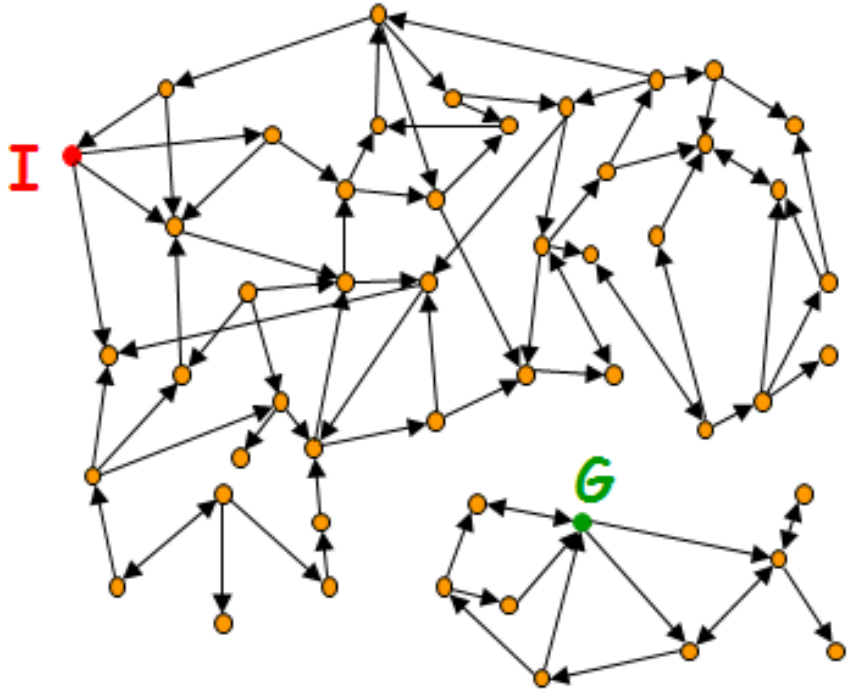
搜索问题的“解”



路径与最短路径

- 如图所示
- 解：从初始节点 I 到任何目标节点 G 的一条路径，沿箭头方向；
- 最优解：解是路径，可能存在多条从 I 到 G 的路径，最优解就是路径耗散 (路径上 cost 之和) 最小的路径

搜索问题的“解”



无解的情况

- 如图所示
- 表示初态和终态的节点在不连通的两个分量中。

三国华容道 $\Longrightarrow$ 搜索问题

- 棋盘的任意一种布局就是一个状态，是状态图中的一个节点，所有可能的的布局构成状态空间/状态图的顶点集；
- 从棋盘的一种布局“合法地”移动一个旗子，得到另一种布局，即所谓的“后继函数”；
- 任何一种初始布局可为初态，曹操跑出来为终态；
- 每移动一个棋子，路径耗散为 1；
- 问题的解：从初态到终态的一个移动序列；最优解：路径耗散/移动次数最少的移动序列。

问题：状态空间有多大？即多少个不同状态？

三国华容道初始状态

横刀立马(81)

层拦叠障(62)

层层设防(102)

水泄不通(79)

过五关(34)

峰回路转(138)

一路进军(58)

井中之蛙(68)

8	2	
3	4	7
5	1	6

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

?

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

=$1000

数码游戏

- Introduced in 1878 by Sam Loyd
- dubbed himself "America's greatest puzzle-expert"
- 将 $n \times n$ 的方格棋盘中的 $1, 2, \ldots, n^2 - 1$ 数字从某个给定的布局调整到另一个给定的布局；棋盘中只有一个空格，调整时，只能把空格"边相邻"的数字移动到空格中。
- 右上图是一个悬赏，过去 100 多年了，至今无人领取。

8	2	
3	4	7
5	1	6

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

?

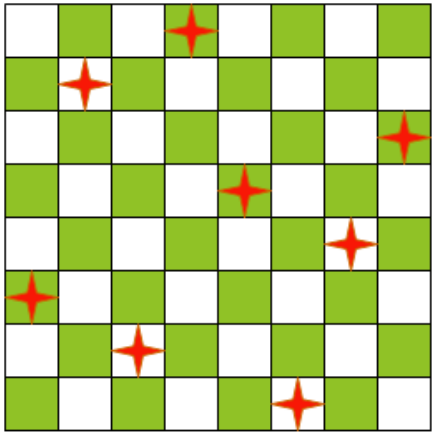
1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

=$1000

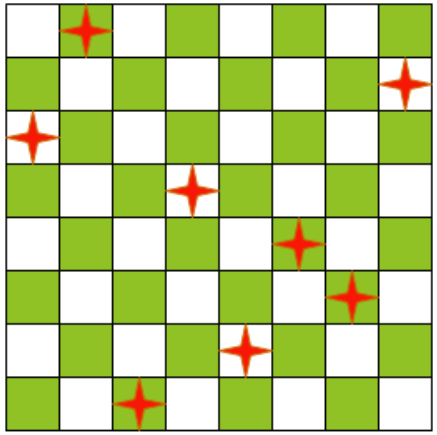
数码游戏 $\Longrightarrow$ 搜索问题

- 每个棋盘布局是状态/节点；
- 移动某个数字到空格是后继函数；
- 初态和终态给定；
- 路径耗散，单步为 1。
- 搜索问题的解：一个从初态到终态的移动序列。

问题：状态空间有多大？即多少个不同状态？



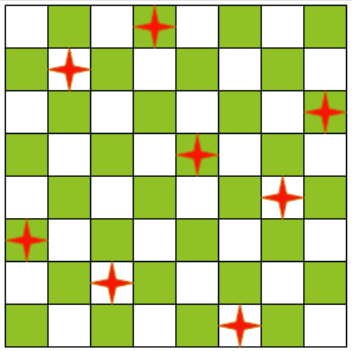
正确的解



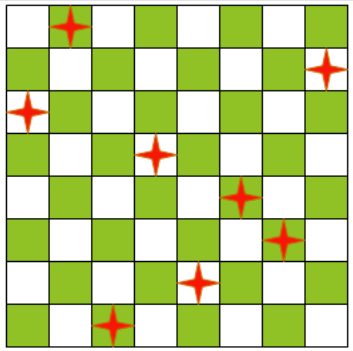
错误的解

8 皇后问题

- 在 8×8 的国际象棋棋盘上，寻找放置 8 个相互不能攻击到的皇后。(注：皇后的攻击路线是直线和 45 度斜线)；
- 上图展示了一个正确的放置 8 皇后方法和一个错误的放置方法；
- 问题扩展：在 $n \times n$ 的棋盘上放置 n 相互不能攻击到的皇后。



正确的解



错误的解

n 皇后问题 $\Longrightarrow$ 搜索问题

- 任意放置 n 个皇后的棋盘布局就是一个节点/状态；
- 任意一个皇后移动到相邻的方格，定义为后继函数；
- 初态为任意给出的一个棋盘布局；终态为满足相互攻击不到的棋盘布局（不是某个特殊的棋盘布局，而是满足某些条件的棋盘布局为终态)；
- 路径耗散：一次移动皇后到相邻格子，代价为 1.
- 搜索问题的解：从初态到终态的移动序列。(其实 n 皇后问题不需要对解要求这么多，仅给出终态即可)

问题：状态空间有多大？即多少个不同状态？

	C1	C2	C3	C4	C5	C6	C7	C8	C9
R1			8				2		
R2		3		8		2		6	
R3	7				9				5
R4		5						1	
R5			4				6		
R6		2						7	
R7	4				8				6
R8		7		1		3		9	
R9			1				8		

数独游戏

- 数独起源于 18 世纪初瑞士数学家欧拉等人研究的拉丁方阵（Latin Square）
- 20 世纪 70 年代，人们在美国纽约的一本益智杂志《Math Puzzles and Logic Problems》上发现了这个游戏
- 数独盘面是个九宫，每一宫又分为九个小格。在这八十一格中给出一定的已知数字，在其他的空格上填入 1-9 的数字。使 1-9 每个数字在每一行、每一列和每一宫中都只出现一次。
- 世界数独锦标赛：首届于 2006 年在意大利卢卡举办，第八届于 2013 年在北京举办

	C1	C2	C3	C4	C5	C6	C7	C8	C9
R1			8				2		
R2		3		8		2		6	
R3	7				9				5
R4		5						1	
R5			4				6		
R6		2						7	
R7	4				8				6
R8		7		1		3		9	
R9			1				8		

数独游戏 $\Longrightarrow$ 搜索问题

- 状态为某个棋盘布局；要求棋盘不会产生“冲突”；
- 后继函数就是在棋盘上的某个空格填上一个数字；使得棋盘是合法的；
- 初态为初始时刻给出的棋盘布局；终态为填满 81 个数字，且符合要求的布局；
- 路径耗散：每填写一次数据，代价为 1（忽略删除某个方格内数字的代价）；
- 数独问题的解：从初态到终态的一个填写数字序列。(数独问题的解可以忽略过程，仅包含符合要求的终态即可)。

搜索问题：状态的解释

状态

- 事物可能的抽象表示，关键属性上相同，不重要细节的影响可以忽略不计
- 比如棋盘的布局，棋子偏移 1 毫米，对布局/状态没有影响
- 状态空间是离散的，有限或者无限

问题	状态定义	状态数目
8-数码	每个格子放置一个数	9！=362,880
15-数码	每个格子放置一个数	$16! \sim 2.09 \times 10^{13}$
24-数码	每个格子放置一个数	$25! \sim 10^{25}$
八皇后问题	每行放一个皇后	8^8
N皇后问题	每行放一个皇后	N^N
N皇后问题	任何一个位置都可以放一个皇后	$(N^2)!/(N^2-N+1)!$

构建“好的”状态空间

- 状态数目通常随问题规模指数增加；
- n 皇后的例子表明，不同的状态定义方法，影响状态空间的大小。
- 存在大量“不可达/违背约束”的状态！能否尽可能减少它们？

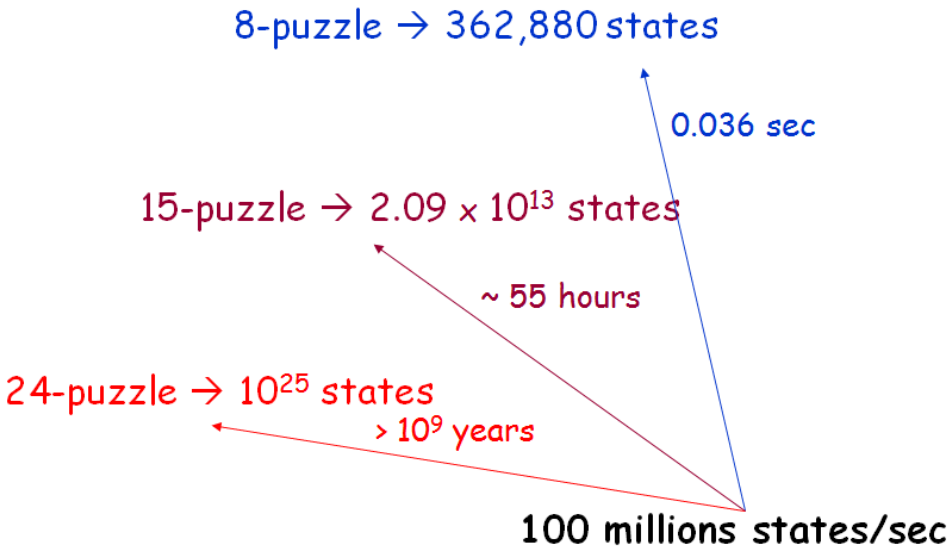
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

?

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

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

$n_2 = 0\ n_3 = 0\ n_4 = 0$
$n_5 = 0\ n_6 = 0\ n_7 = 1$
$n_8 = 1\ n_9 = 1\ n_{10} = 4$
$n_{11} = 0\ n_{12} = 0 n_{13} = 0$
$n_{14} = 0 n_{15} = 0 \rightarrow$ 逆序值 = 7



Sam Loyd 不用担心钱被领走!

- 逆序值: 每个数被逆序的次数之和
- 逆序值的奇偶性不会被后继函数改变
- 整个状态图分为两个不联通分量，分别对应逆序值的奇与偶

ustc

搜索问题：何谓“好的”状态

“好的”状态/状态空间

- 所有状态？尽可能让状态都是可达的/可行的/合法的；
- 状态数目越少越好，通常是问题规模的指数函数；
- 能方便设计后继函数和搜索算法。

进一步解释

- 不可达的状态，在搜索问题无解时非常重要；
- 状态太多时，计算机存储空间无法存储所有的状态（空间受限）；也无法遍历一遍（时间受限）；
- 从数据结构的角度看：搜索算法的简洁性和高效性，约束状态空间和后继函数的设计。

搜索问题：后继函数的解释

8	2	7	1569	1456	45	13456	159	34569
69	69	1	3	7	245	4568	2589	24569
4	5	36	1269	16	8	1367	1279	23679
16	67	2	4	156	57	9	3	8
5	78	4	178	9	37	2	6	7
69	3	68	5678	2	57	57	4	1
3	4	35	257	8	6	13457	12579	234579
2	468	9	57	45	1	345678	578	34567
7	1	568	25	3	9	4568	258	2456

后继函数：状态图的边

- 如左图所示数独求解器的例子。
- 每个状态可行的所有“行动”的集合，方格内左上角小写的数字
- 后继函数的返回值，就是行动的结果，称之为后继状态
- 左图表示状态，其后继函数会导致 138 个不同的后继状态
- 后继函数是问题建模（现实问题转换为搜索问题）的“关键”！最复杂的部分。

路径耗散

- 通常标注在状态图的边上，表示执行一个行动/动作的花费/代价，可以认为是后继函数的执行代价；
- 也可以认为是一种“奖励”，执行完动作后获得的 reward，但是此时最优解通常对应最大化行动序列获得的奖励之和；
- 一般总是正的；
- 而且我们总是假设路径耗散 c_i 有一个有限的下界，即大于某个正常数 $c_i \geq \epsilon > 0$

路径耗散的例子

- 数码问题，每移动一次，代价为 1；
- n 皇后问题，每放置/撤回一个皇后，代价为 1，从一个状态转移到下一个状态，花费的代价 $\leq 2n$;
- 数独问题，填写/擦除一个数字，代价各为 1，从一个状态转移到下一个状态，花费的代价 $\leq 2n$

搜索问题：初态和终态的解释

初态：单源或多源

- 给出某个状态，并精确地描述出来；如三国华容道，数独问题等；
- 给出某些状态，它们满足某些共同的性质或条件；
- 仅仅给出某些条件或性质，称满足条件的都是合法的初态，如初始随机放置 8 个皇后的 8 皇后问题，条件是“有 8 个皇后存在”。

终态

- 可以是一个被精确定义的状态
- 可以是满足某些条件的状态
- 可以是满足某些条件的状态集合

搜索问题：解的解释

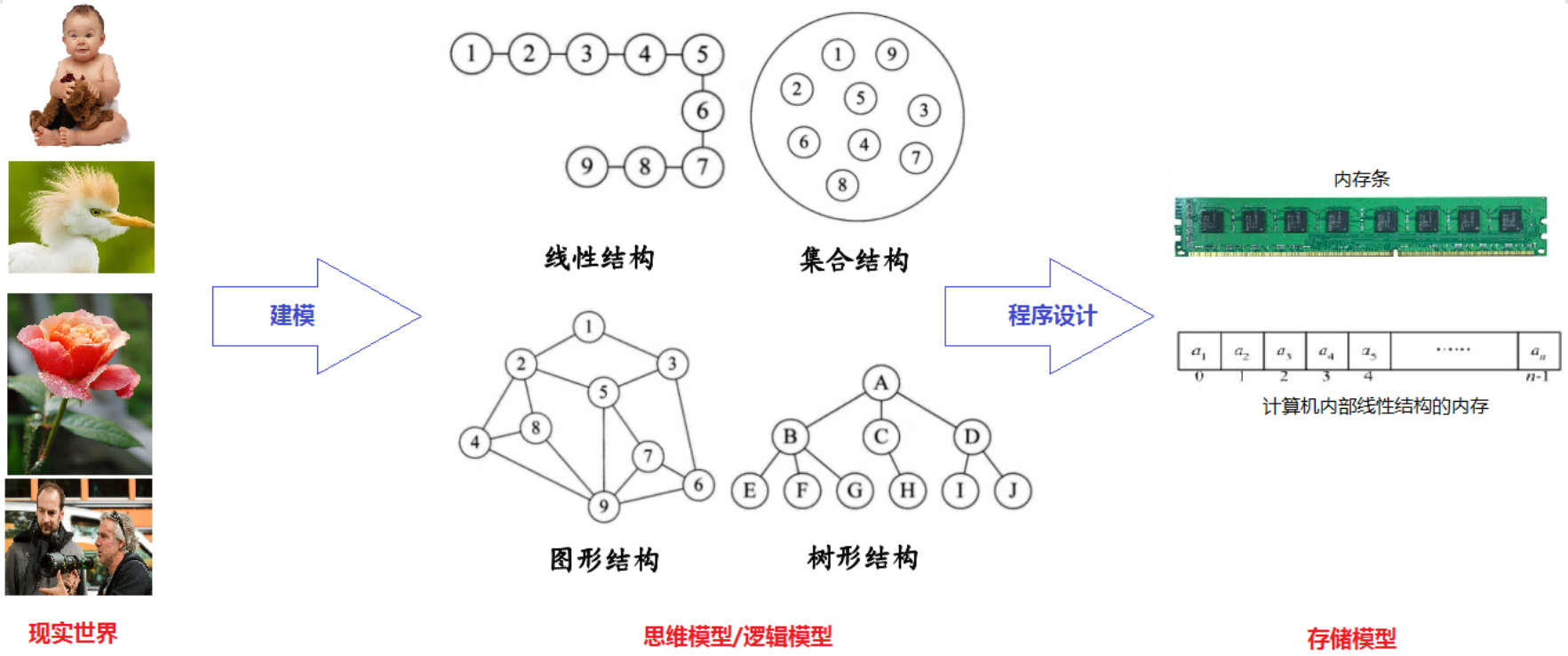
解：从初态到终态的路径

- 可能有唯一解
- 可能无解
- 可能有多个解，路径耗散最小的称之为“最优解”

解的约束太强了，有些问题可以放松约束

- 很多问题，我们需要解表示的行动序列，比如下棋；
- 然而，如 8 皇后问题，我们并不关心如何从初态调整到终态的过程，仅仅需要最终的棋盘布局，即终态；因此，可以只要解序列的最后一个状态即可，降低了对解的要求；
- 数独问题，对解的要求类似 8 皇后问题，这类问题即通常所谓的“约束满足问题”。

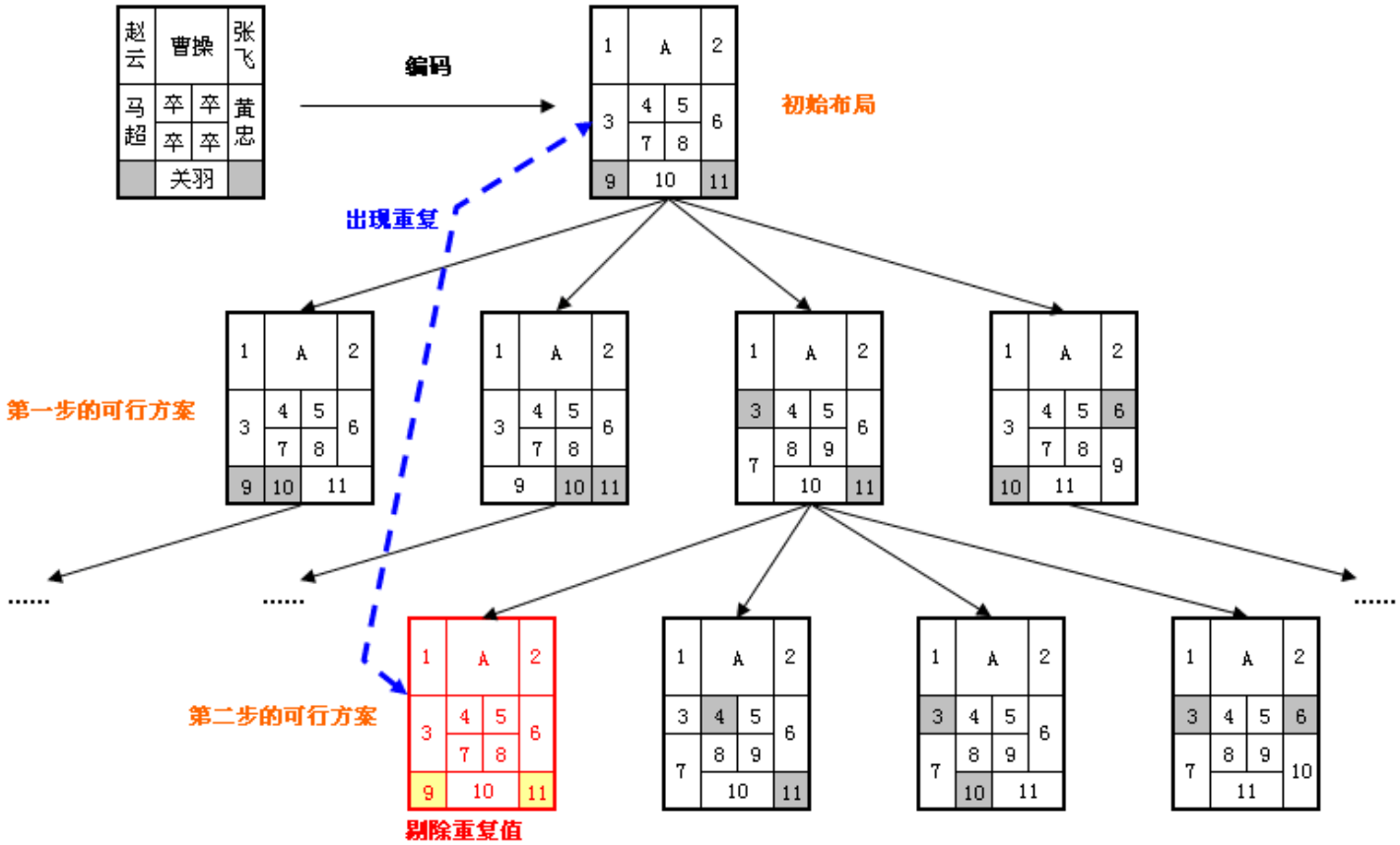
问题求解：应用计算机技术



AI 和数据结构

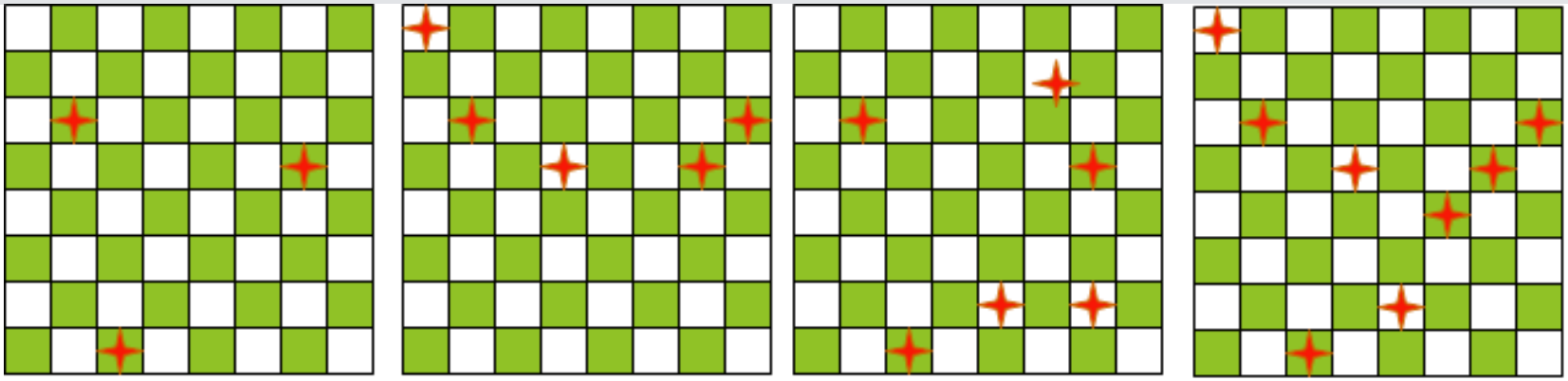
- 数据结构：更侧重在通过程序设计实现从逻辑模型到存储模型 (物理模型) 的转换；数据结构的设计应以程序设计实现的简洁和高效为指导。
- AI：更多地讨论如何将现实世界建模为逻辑模型，将更复杂的现实问题转化为各种受限的、带约束的逻辑模型。

三国华容道的计算机求解



建模和编程

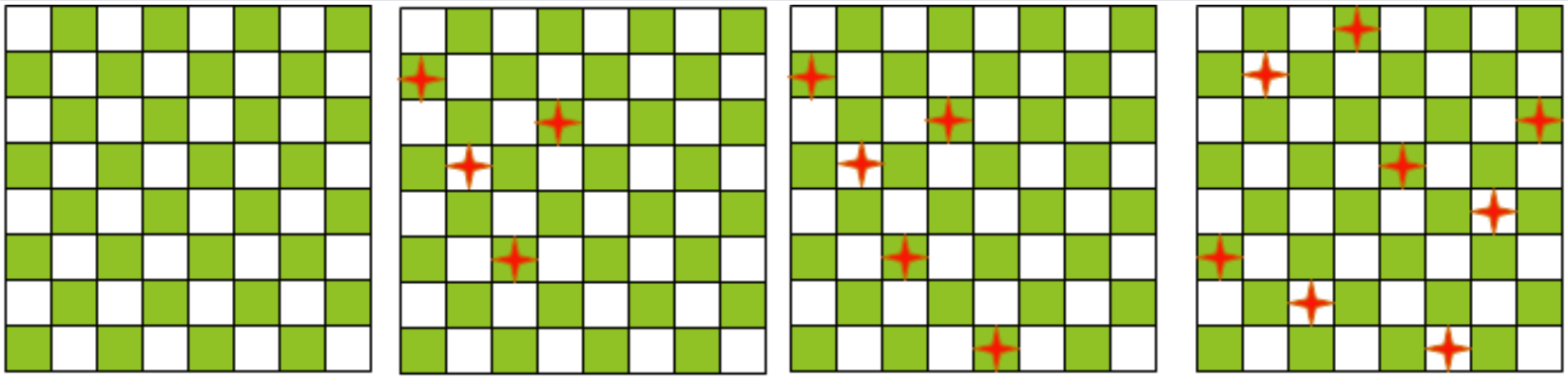
- 建模：编码状态/棋盘布局，编码可行的移动，利用基本数据结构数组和树等；
- 编程实现：在线性内存空间上实现



8 皇后问题的形式化 1：现实问题建模为搜索问题

- 状态，任何 0，1，2，.....,8 个皇后在棋盘上时的布局代表一个状态
- 初始状态：0 个皇后在棋盘上
- 目标测试/目标状态：8 个皇后在棋盘上，彼此间不互吃
- 路径耗散：放置一个皇后，代价为 1
- 后继函数：在一个空的格子上放置一个皇后，得到一个新状态
- 状态空间/状态图：高度为 9 的 64-叉树

约 $64 \times 63 \times \ldots \times 57 \backsim 3 \times 0^{14}$ states



8 皇后问题的形式化 2：现实问题建模为搜索问题

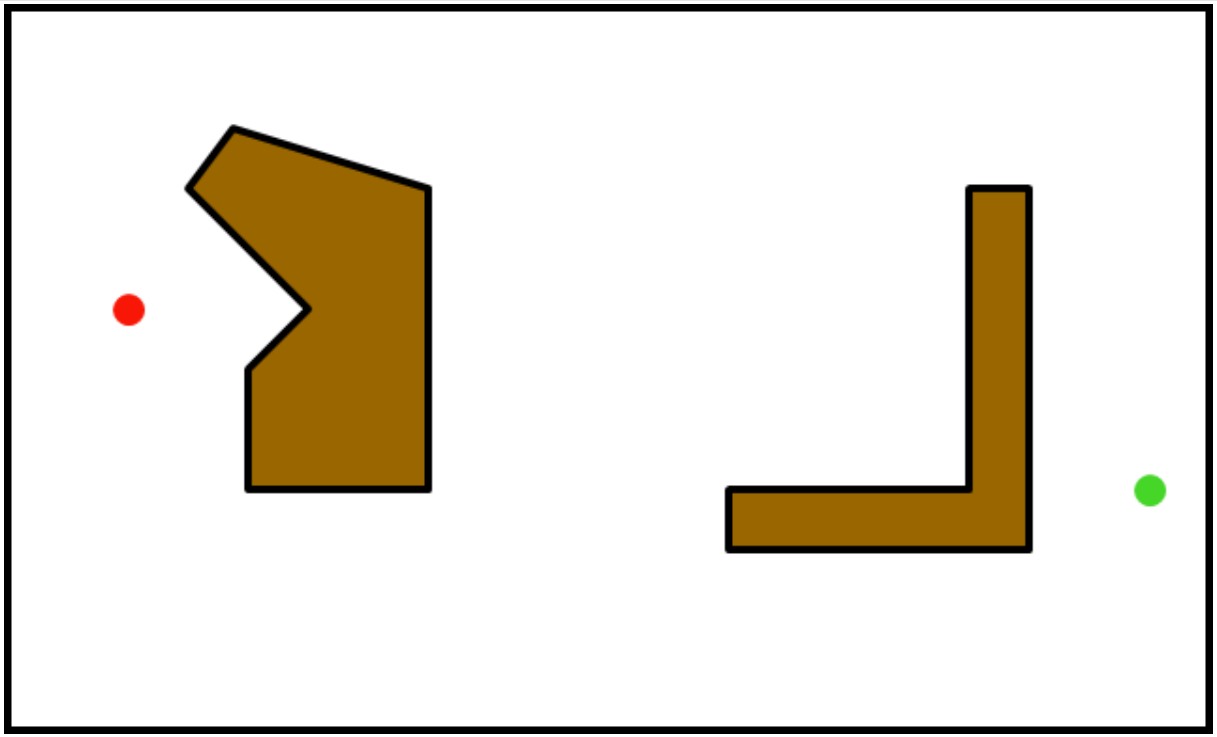
- 状态，任何 0，1，2，.....,8 个皇后在棋盘左侧，且互不攻击时代表一个状态；所谓棋盘左侧互不攻击，就是前 k 列每列一个皇后，互不攻击；
- 初始状态：0 个皇后在棋盘上；
- 目标测试/目标状态：8 个皇后在棋盘上；
- 路径耗散：放置一个皇后，代价为 1；
- 后继函数：在 k+1 列放置第 k+1 个皇后到一个不受攻击的位置；
- 状态数目急剧减少！

约 2057 states

当 n 增大时

- “解”是一个状态，而非从初始态到目标状态的序列；这类搜索问题，一般称为“设计问题”，学术上称为“约束满足问题”；
- 上述形式化方法 2 是否能“简单”求解 N 皇后问题?
  - 8 皇后 $\rightarrow$ 2057 个状态
  - 100 皇后 $\rightarrow 10^{52}$ 个状态
- 能否有算法能快速求解 N 皇后问题?
  - 问题特点：很多个解，解的分布较好（较均匀）
  - 爬山法

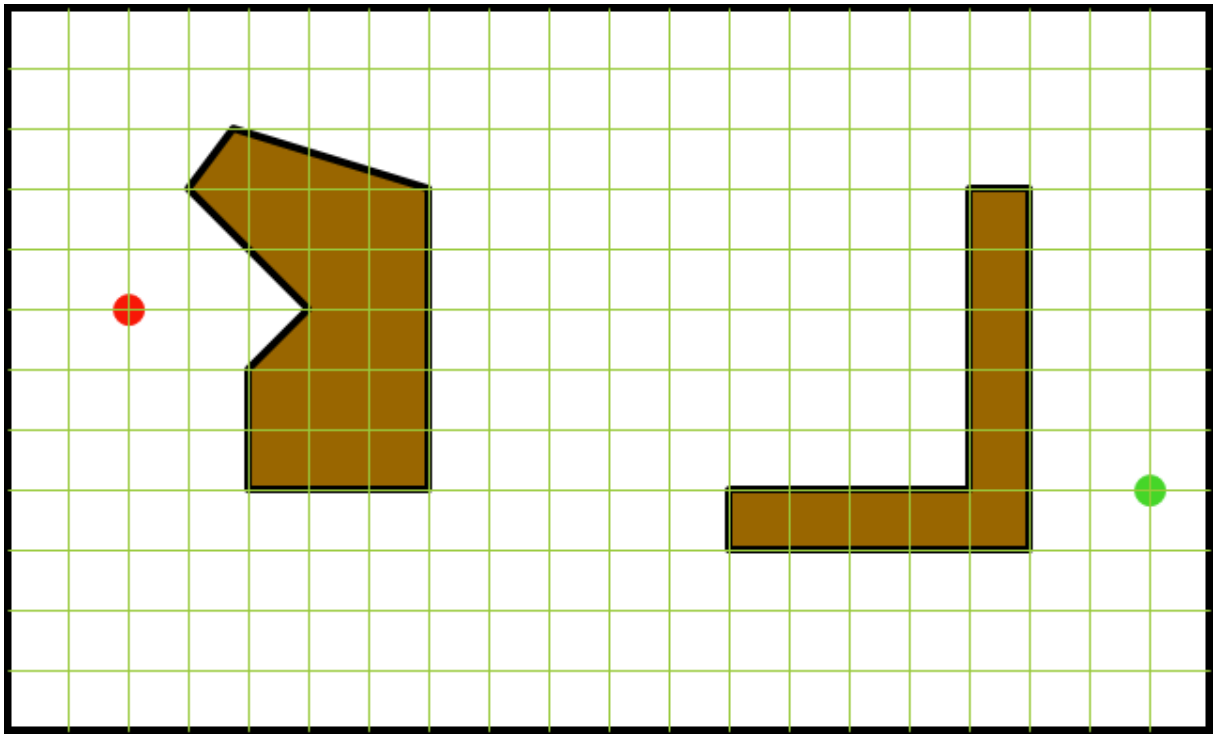
路径规划问题形式化



路径规划问题形式化：把现实问题转化为搜索问题

- 如上图所示场景，有不可触碰的障碍物 (咖啡色所示)，可能是游戏地图，可能是汽车自动驾驶地图等；
- 红点是出发点，绿点是目标终点；
- 寻找一条从红点到绿点的路径，或最短路径。

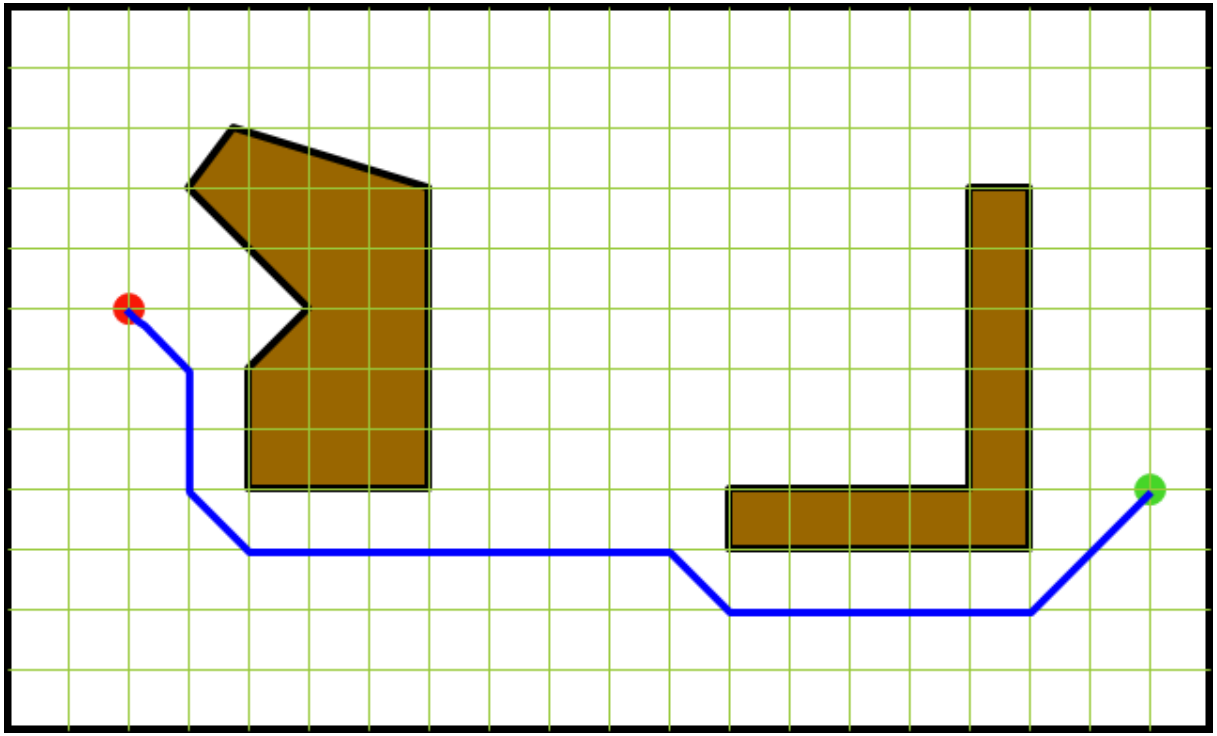
路径规划问题形式化 1



路径规划问题形式化 1：网格化

- 如上图所示，把地图网格化；
- 令网格边长为 1，则对角线距离为 $\sqrt{2}$；
- 状态就用当前位置的坐标 (x, y) 来表示，所有的整数点坐标值构成状态空间；
- 小方格的边为后继函数，后继状态是当前状态的东南西北四个正方向上，距离为 1；
- 也可以定义后继函数包括四条 45 度对角线，即后继状态在当前状态周围最近的 8 个整数坐标位置上。

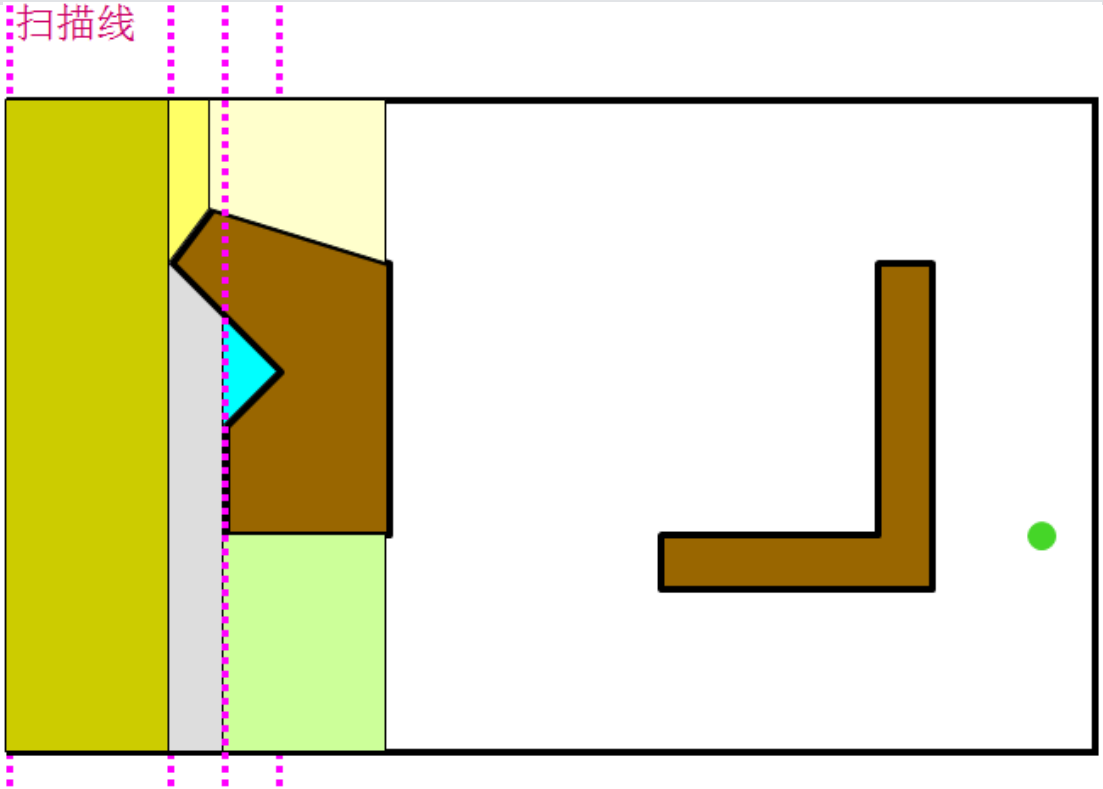
路径规划问题形式化 1



路径规划问题形式化 1：网格化

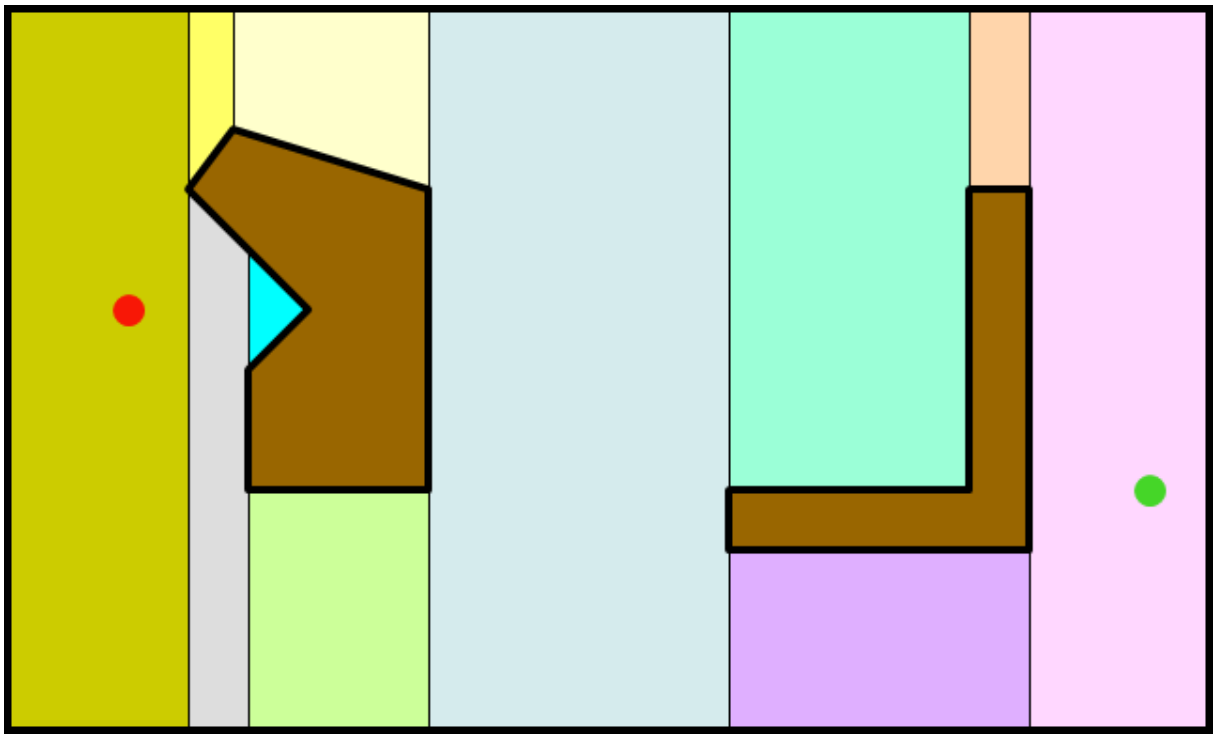
- 解是一个序列，相邻整数点/状态之间距离 $\leq \sqrt{2}$，如上图所示
- 最优解，仅仅是网格化的离散空间中的最优解

路径规划问题形式化 2



路径规划问题形式化 2：扫描线方法

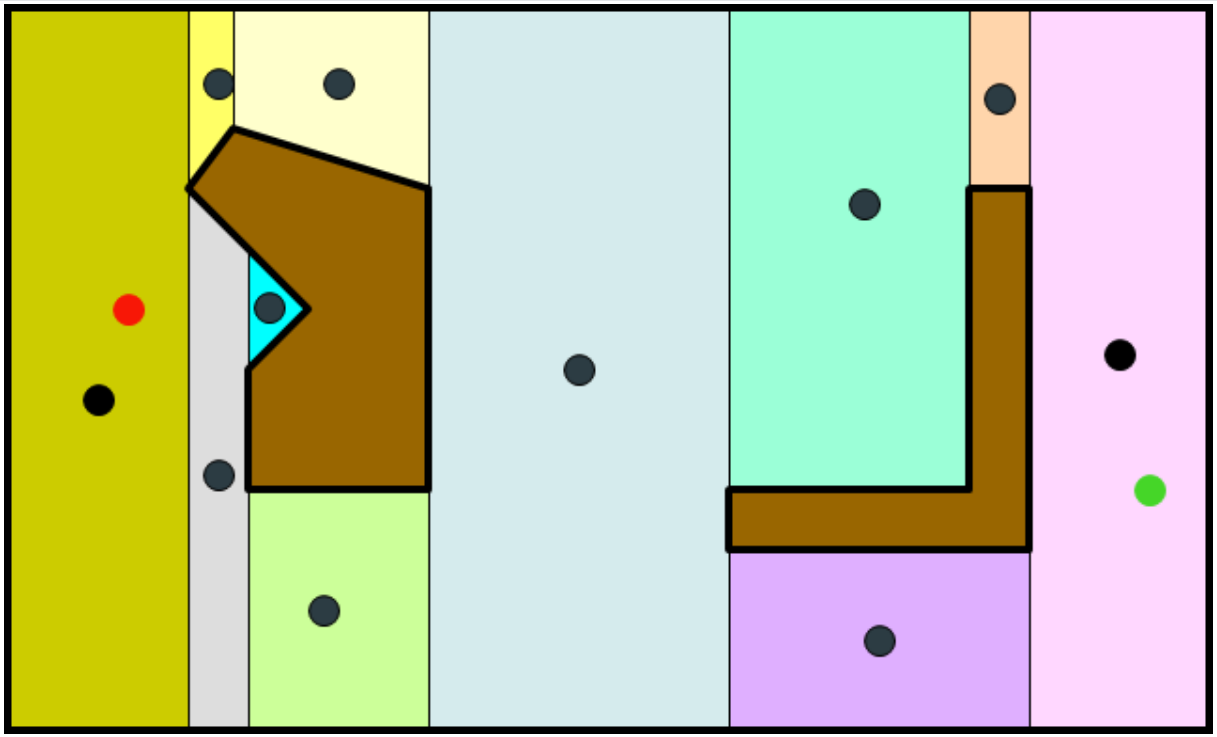
- 假设垂直扫描线从左到右进行扫描；
- 遇到障碍物的顶点（不妨设障碍物为多边形）则暂停，标记；
- 标记方法：对两次暂停之间的被扫描区域染上不同的颜色；
- 被障碍物割裂的扫描线获得不同颜色的区域。



路径规划问题形式化 2：扫描线方法

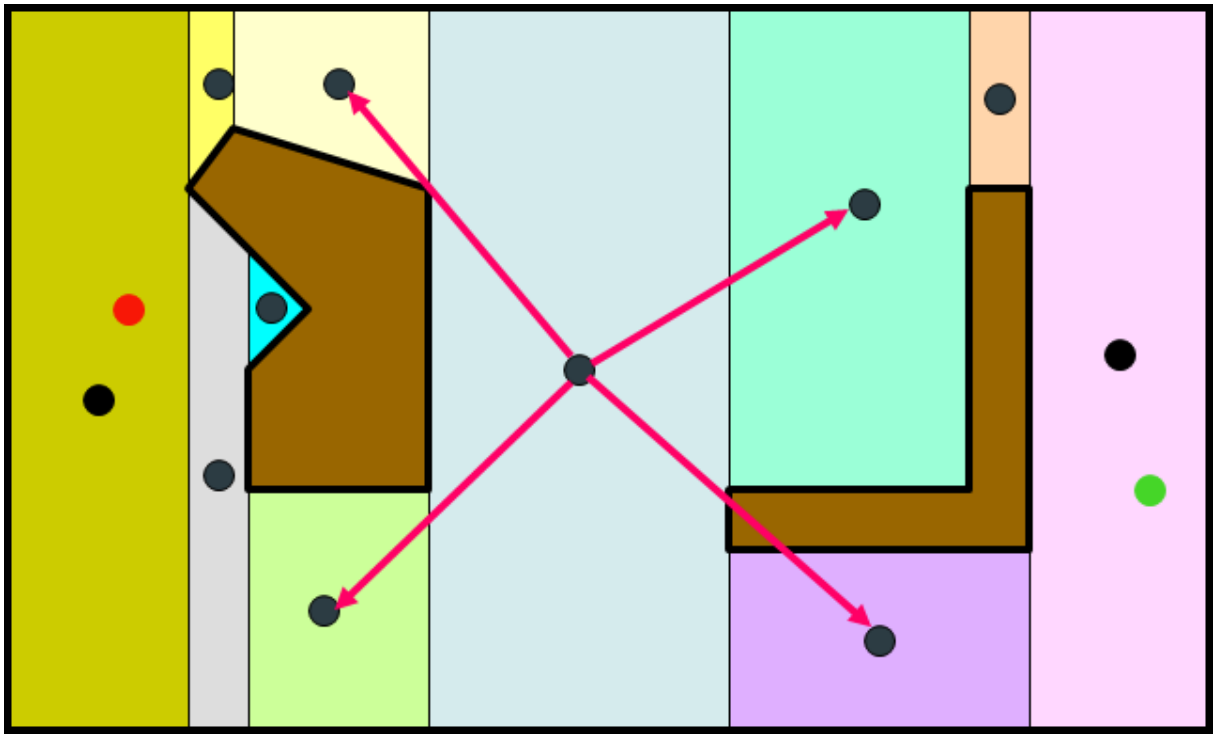
- 如上图所示，扫描整个区域后，得到彩色的着色图。

路径规划问题形式化 2



路径规划问题形式化 2：扫描线方法

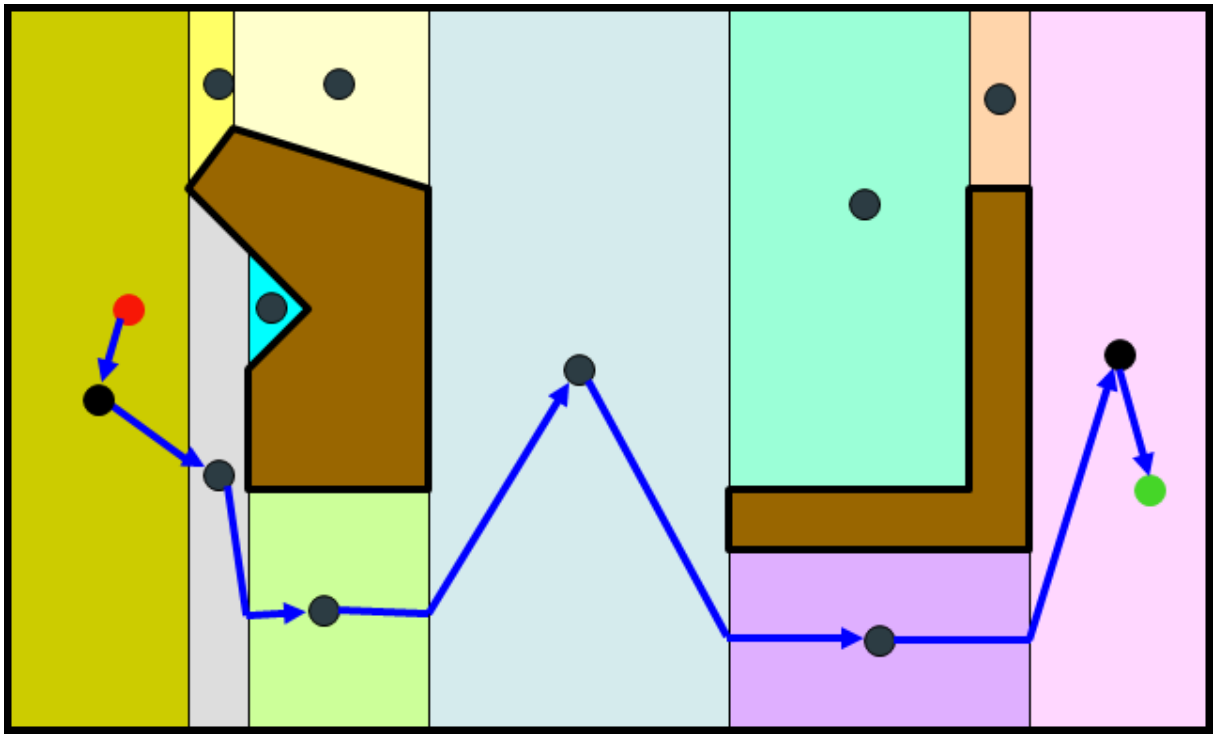
- 每个染色区域用其重心点作为代表
- 每个重心点代表一个状态，获得状态空间
- 如上图，共有 11 个不同色彩区域，即 11 个不同状态，初始位置和目标位置视为额外的两个状态，共 13 个状态。



路径规划问题形式化 2：扫描线方法

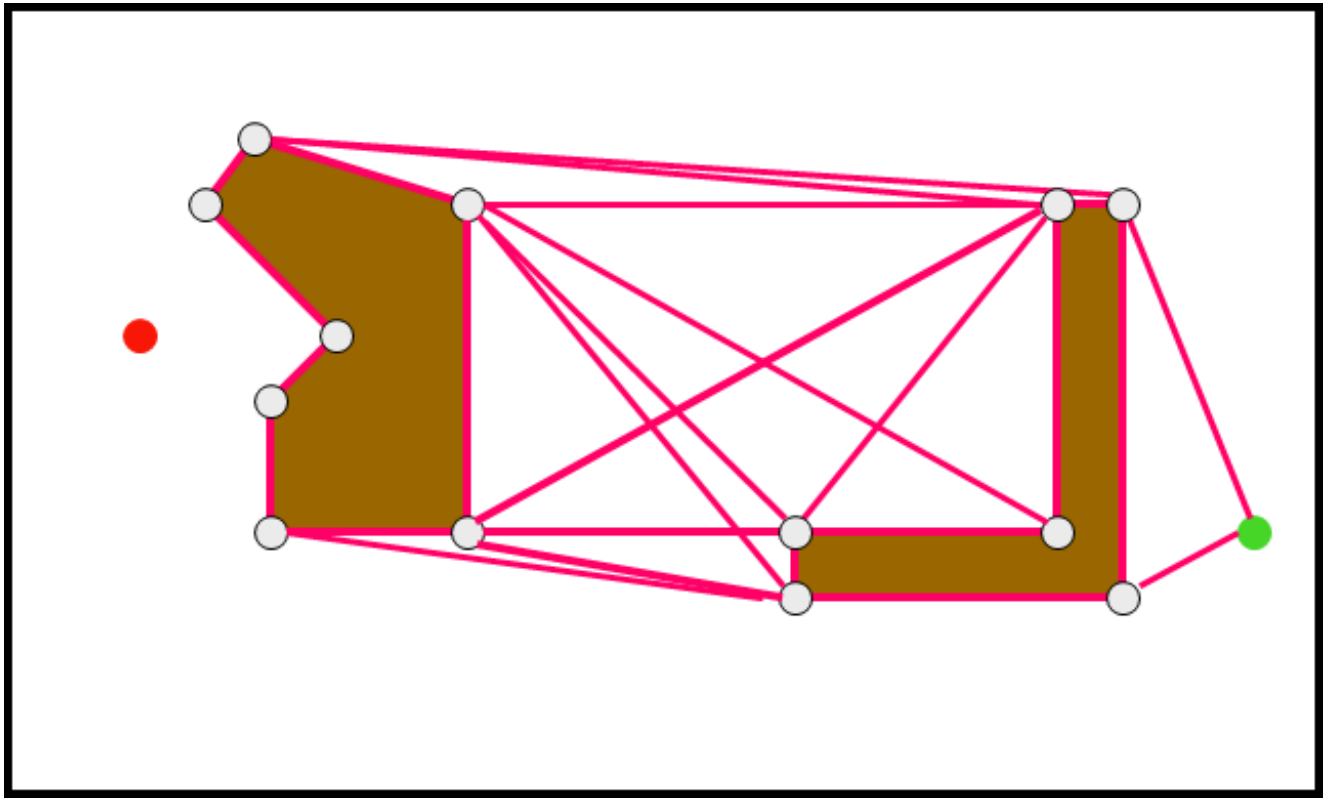
- 后继函数：具有相邻边界线的色块之间互为后继
- 获得状态图，如上图所示
- 边/弧的路径耗散/代价为状态点之间的距离，如欧拉距离

路径规划问题形式化 2



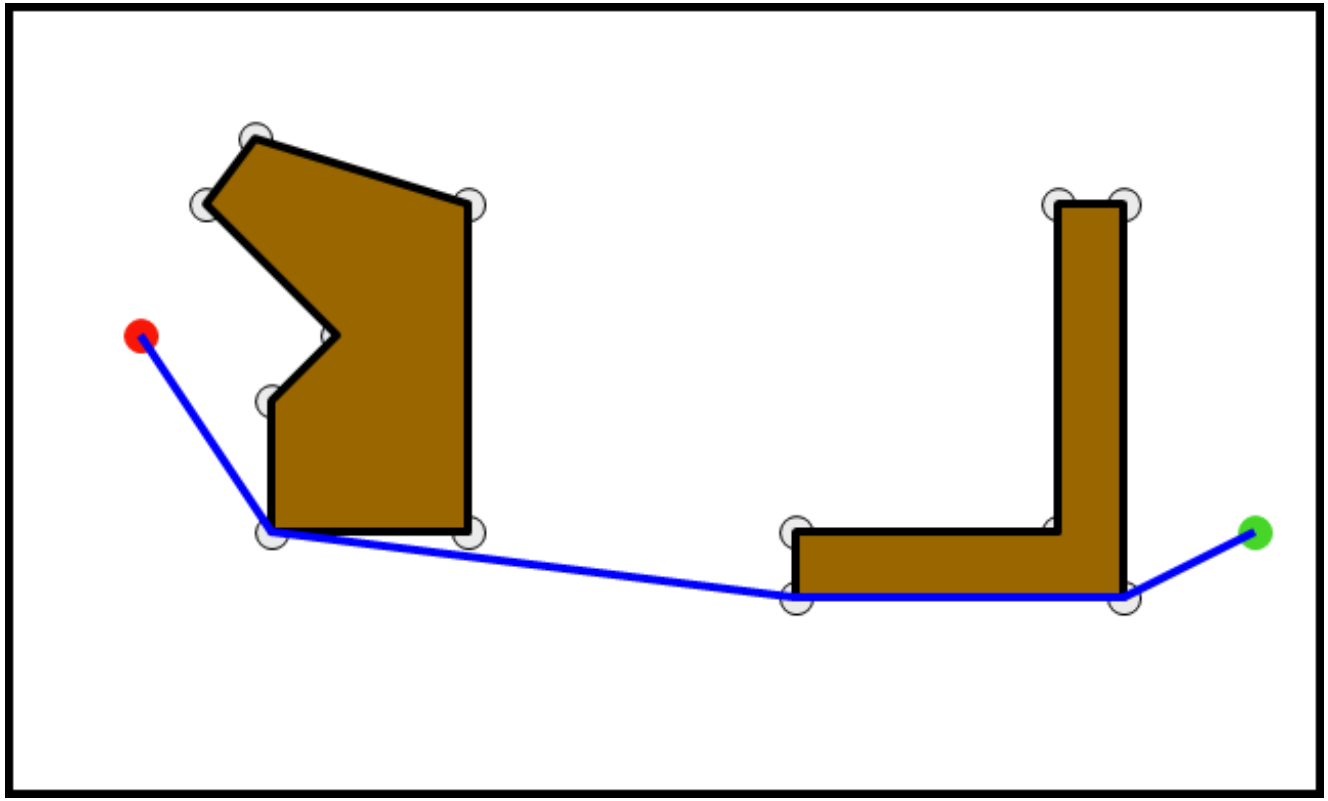
路径规划问题形式化 2：扫描线方法

- 路径规划问题的解为一条路径，如上图所示
- 路径需要通过某些特定的边界点
- 最优路径需要进一步优化和平滑



路径规划问题形式化 3：障碍物边界法

- 取障碍物（不妨设为多边形）的顶点以及初始地点和目标地点构成图的顶点；
- 对任何顶点，连接其可视顶点（没有被障碍物阻挡），获得后继函数;
- 路径耗散/代价：边/弧的长度，顶点间的距离。



路径规划问题形式化 3：障碍物边界法

- 路径规划问题的解，如上图所示
- 最优解是连续的

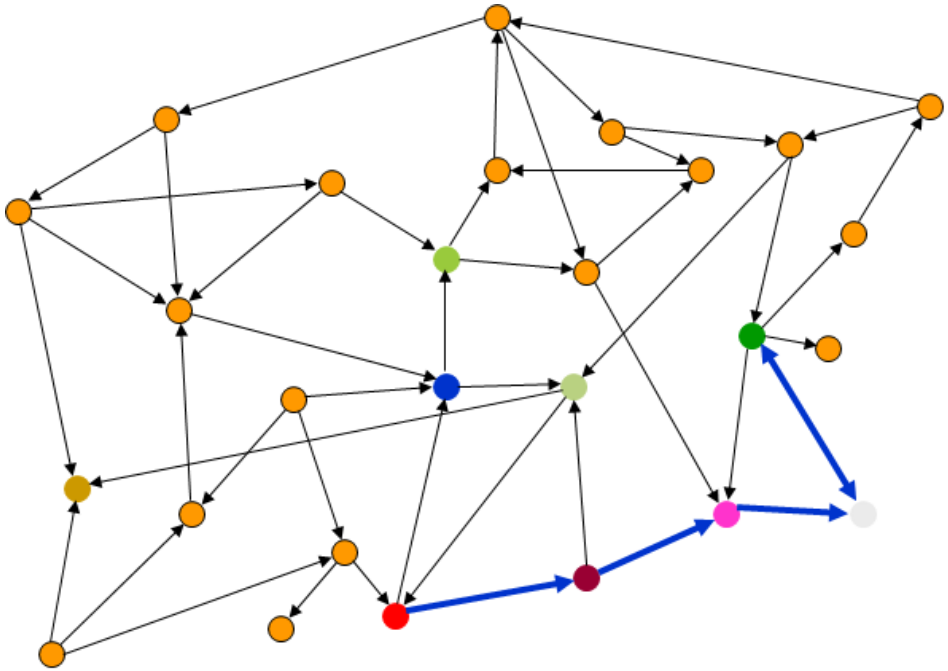
赫伯特 · 西蒙：设计实验证明人类解决问题的过程是一个搜索过程

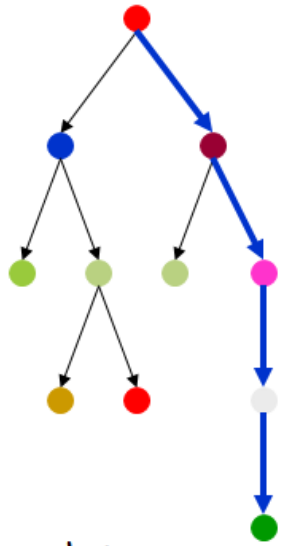
- 路径搜索与导航
- 打包/邮件分发
- 超大规模集成电路布局设计
- 蛋白质比对和折叠
- 制药
- 电脑游戏
- ……

思考：现实问题建模成通用的“搜索问题”（描述其 5 个要素）。

模型 $\Longrightarrow$ 编程求解/算法设计运行

编程求解搜索问题



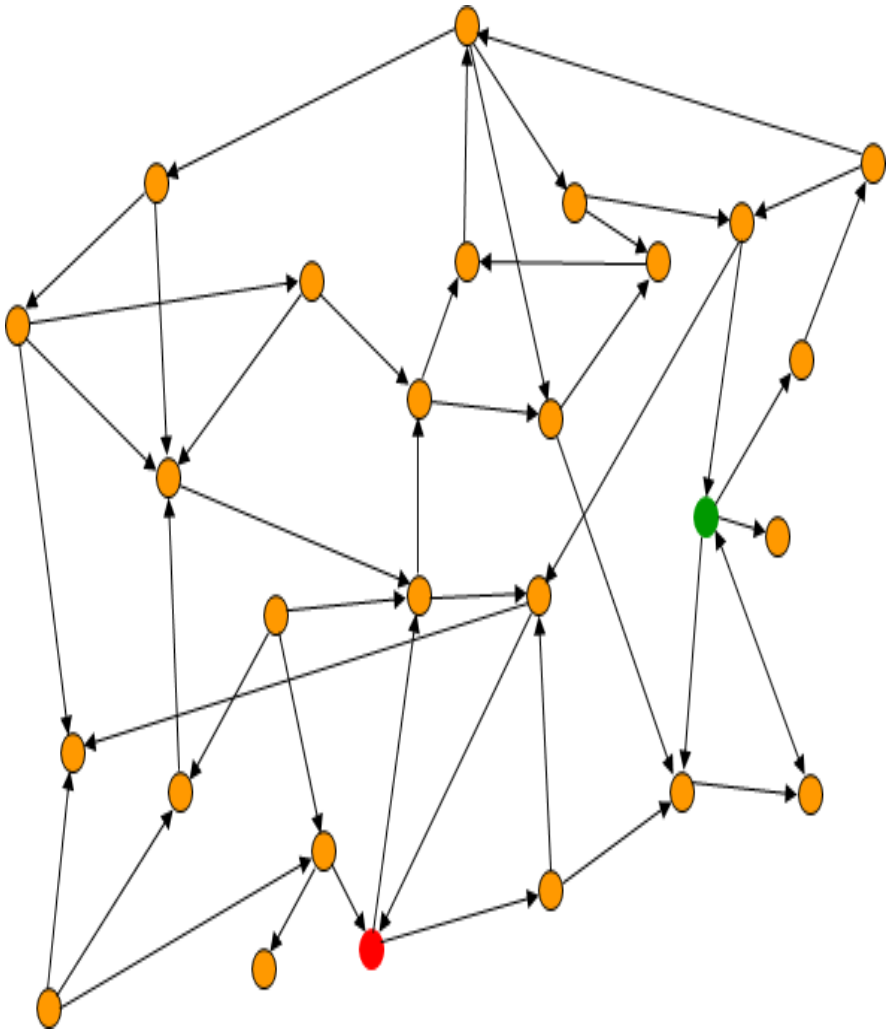


Search tree

算法思想：利用图的宽度优先遍历算法

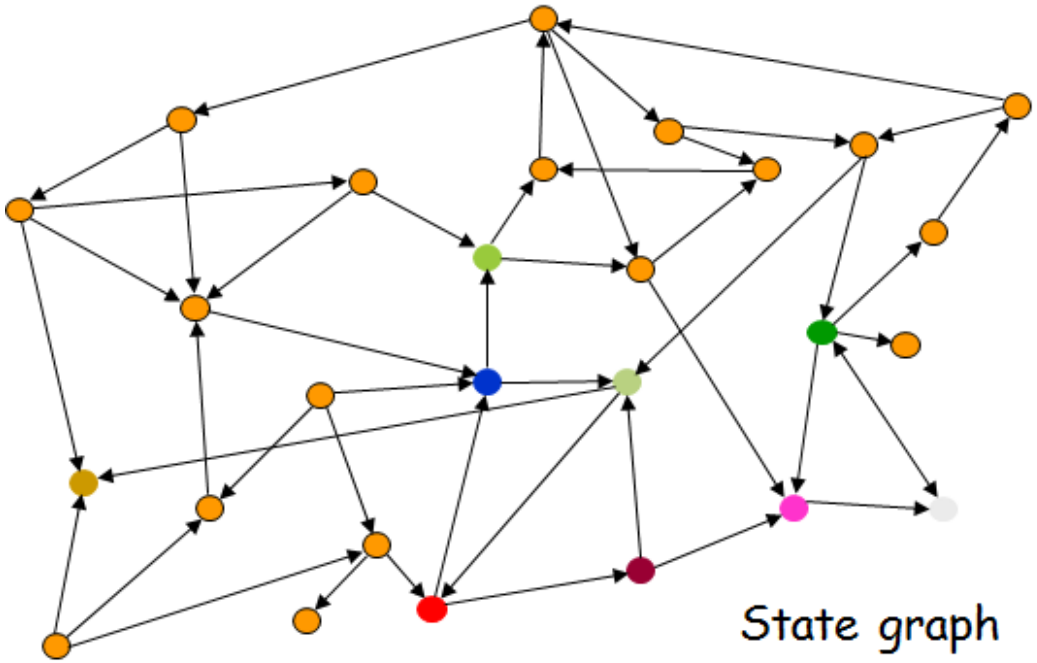
- 如上图所示状态图如左，表示一个搜索问题，红点为初态，进行宽度优先遍历，得到右图的搜索树/生成树
- 我们称宽度优先遍历为求解搜索问题的基准算法，简称为宽度优先搜索算法

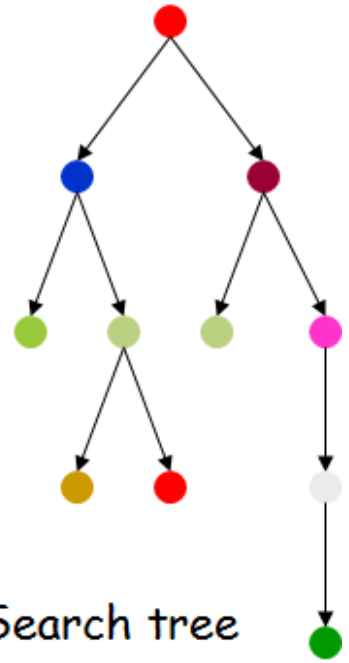
我们关心的搜索算法



与基准算法对比

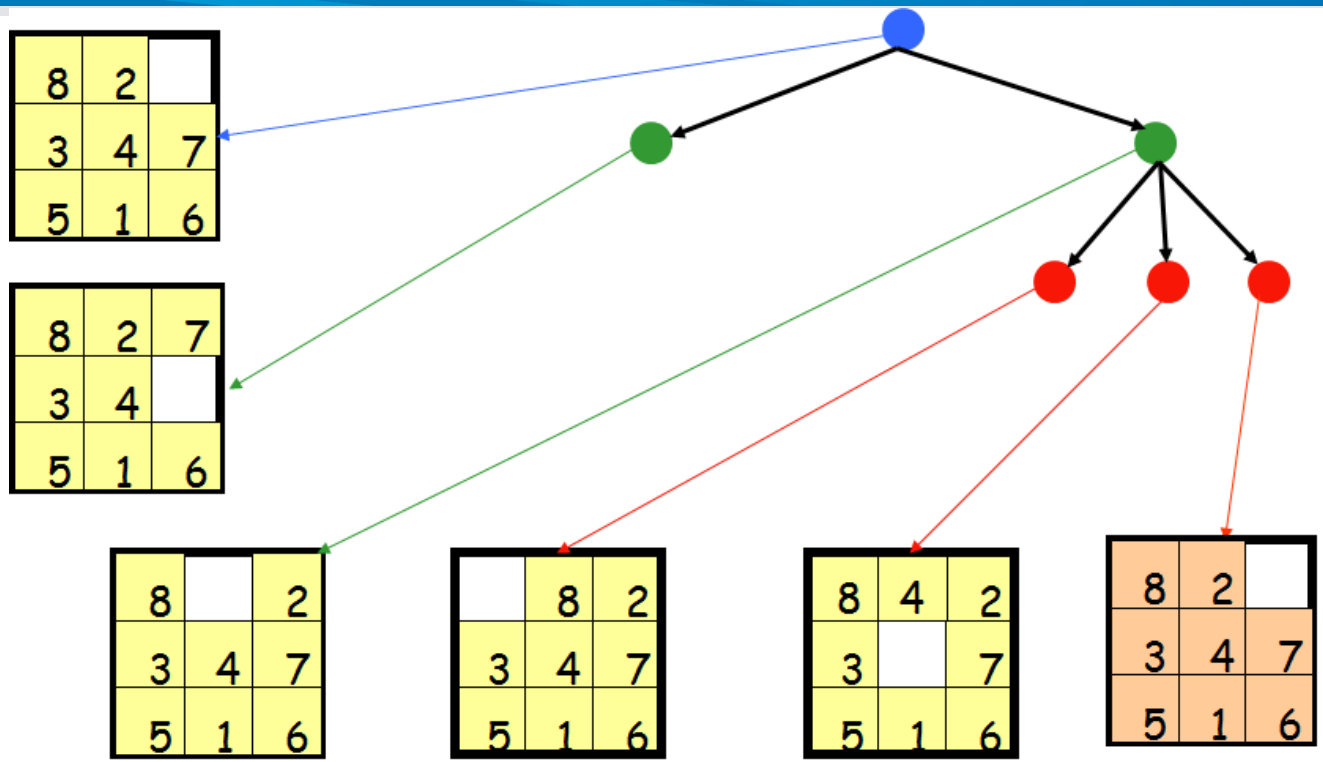
- 仅仅搜索整个状态图的一小部分（状态数的对数多项式量级）就能获得解或最优解
- 其它算法与基准算法比，相同点：
  - 算法大框架基本一致
  - 所有的状态（可达 + 不可达）构成的状态图
- 其它算法与基准算法比，不同点：
  - 从状态图中选择/处理的状态子集不一样
  - 状态子集构成的状态子图（弧）不一样，后继函数定义不一样





搜索算法：从状态图到搜索树

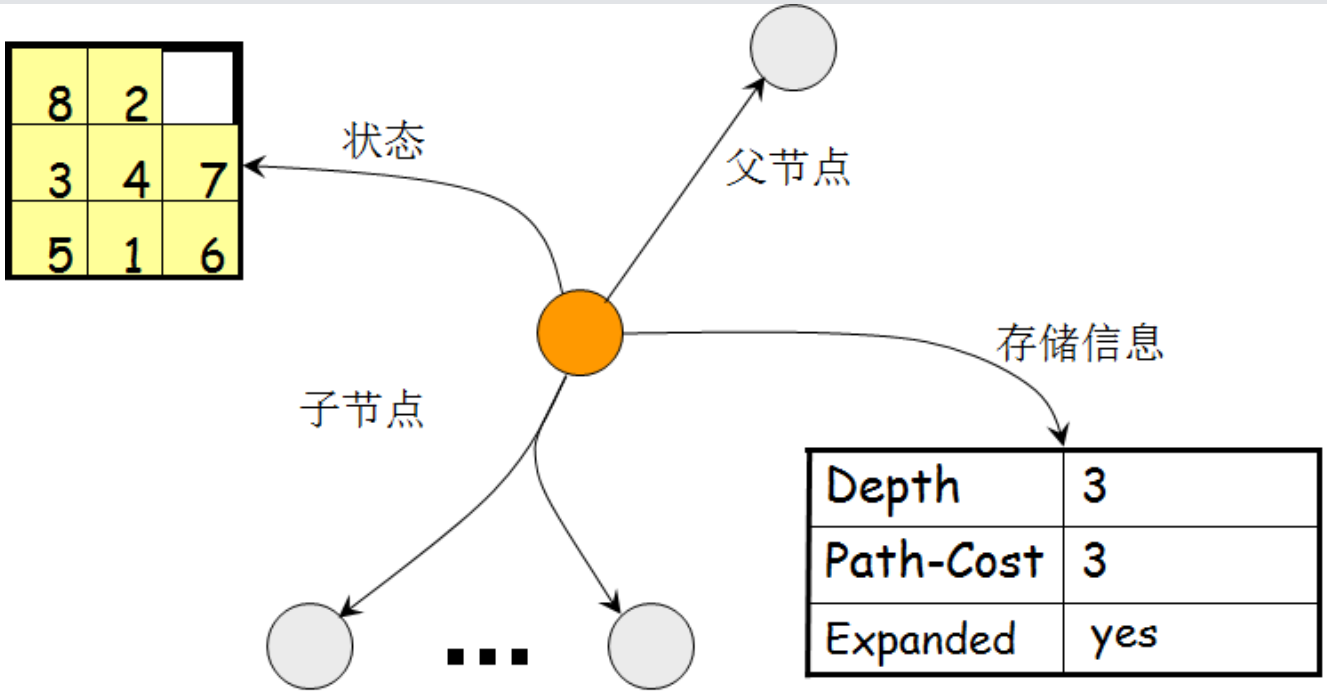
- 左边的状态图描述了一个搜索问题
- 右边是在状态图上进行搜索，求解搜索问题（获得路径或最短路径）
- 注意：状态图中的节点/状态可能被重复访问，体现在同一节点/状态会变成搜索树的多个节点



数码游戏的搜索树

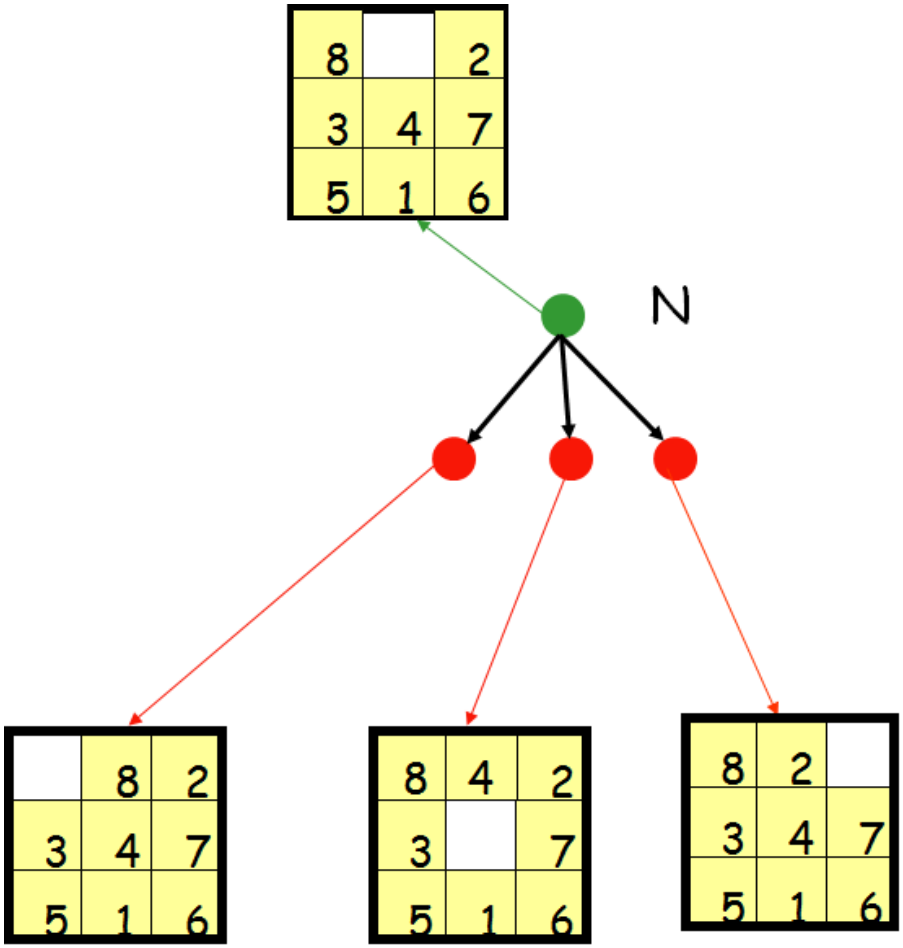
- 如上图所示，8 数码问题的搜索树的一部分
- 搜索树的根节点（蓝色）和最右下角的红色节点，棋盘布局一样，是同一个状态 (状态图的同一个节点)
- 同样的状态，在树中可能有多个节点表示；我们在后面的讲解中区分二者
- 若允许状态被重复访问，即搜索树中同一状态可用多个节点描述，则搜索树可能是无限深的

搜索算法：数据结构设计



求解问题，有了模型（逻辑结构，状态图）之后，设计数据结构

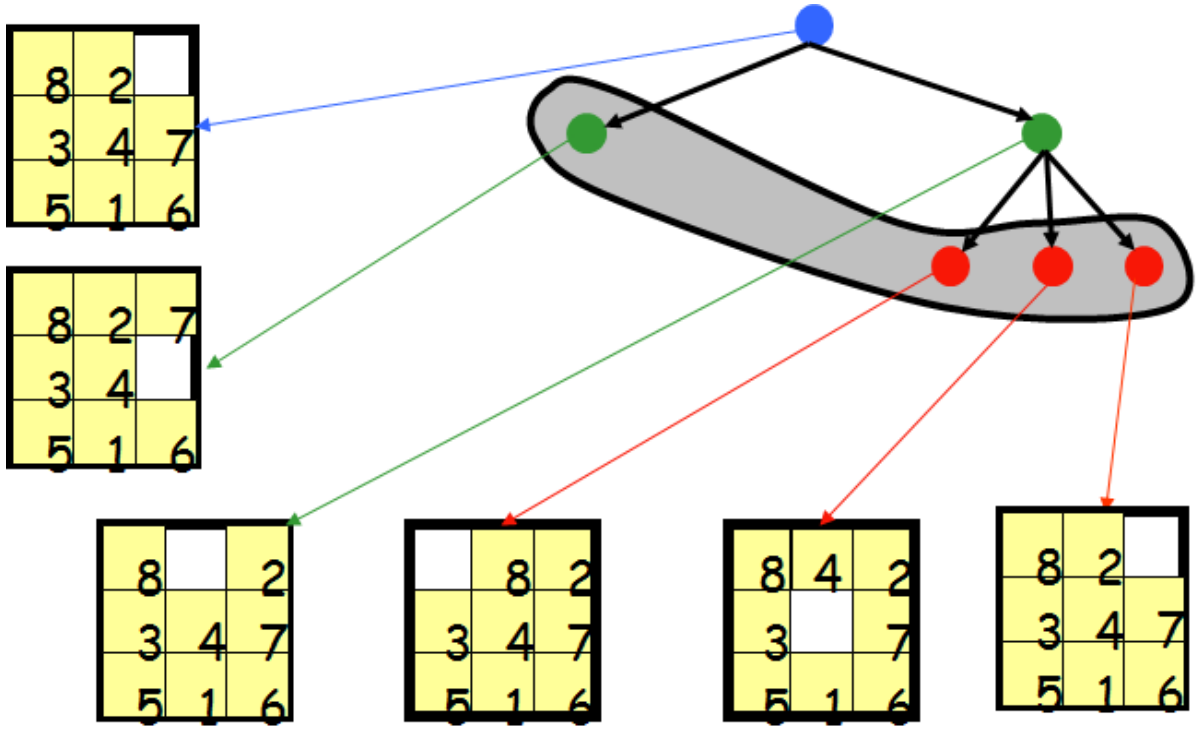
- 节点 N 信息包括（深度，路径代价，是否已扩展），其中深度/Depth 就是节点 N 的（从根开始）路径长度，路径代价就是从根开始的路径耗散，节点扩展是对节点的一个操作，若完成扩展操作，则标注为 yes
- 通常初始状态为搜索树的根，其深度为 0



搜索算法核心操作：节点扩展

- 生成搜索树的关键操作；
- 节点扩展直观理解就是把一个节点用它的后继节点（集合）来替换；
- 节点 N 的扩展，包括的处理：
  - 评估状态/状态图节点 N 的后继函数，如到该状态的路径耗散是多少，估计从该状态到目标状态至少还要花费多少路径耗散等等；
  - 对后继函数返回的所有后继状态，在搜索树上分别产生一个子节点与之对应。
- 节点扩展 $\neq$ 节点产生

搜索算法：搜索树的边界



搜索树的边界：FRINGE

- 如上图中“云”中的那些状态（或搜索树中的节点）
- 未扩展状态的集合，可理解为：在排队等待扩展的那些状态！
- 与叶节点不同，叶节点可能已经扩展过了，但是没有子节点；
- 未扩展状态在搜索树上的节点，因为未扩展，所以还没有子节点，看上去“像”叶节点。

ustc

搜索策略：FRINGE 中节点的优先次序

- FRINGE 中的节点是已访问过/处理过状态的后继节点中未扩展的；
- 通常用“队列”来存储 FRINGE 中的对象；(先进先出是队列的特点)
- 但是我们修改“队列”的存储方式，重新定义一个 FRINGE 中节点的优先数组/有序数组，得到一个“搜索策略”。
- 影响/设计实现搜索策略的核心操作：(类似队列操作)
  - INSERT(node,FRINGE)：向优先数组 FRINGE 中插入优先级最低的节点 node;
  - REMOVE(FRINGE): 从优先级数组中删除优先级最高的节点。

搜索算法框架: 基准算法

Input: $\mathbb{G}$: 状态图; s_0: 初态;
Output: $path$: 代表解的路径
$path \longleftarrow (s_0), FRINGE \longleftarrow \phi$ /* 初始化 */;
if $(GOAL(s_0) = T)$ **then**
  return $path = (s_0)$;
end
INSERT(s_0, $FRINGE$);
while T **do**
  if $empty(FRINGE) == T$ **then**
    return $failure$ /* 返回 $failure$, 表示无解 */;
  end
  $N \longleftarrow$REMOVE($FRINGE$) /* 将未扩展节点中队头节点从队列移除到 N*/;
  $s \longleftarrow$STATE(N) /* 从节点 N 恢复为状态 s*/;
  update $path$;
  foreach s' *in* $successors(s)$ **do**
    为 s' 创建 N 的新子节点 N';
    if $GOAL(s') = T$ **then**
      return $(path, s')$/* 找到目标节点，返回解 */;
    end
    INSERT(s',$FRINGE$);
  end
end

节点扩展

搜索算法：评估算法性能

完备性

- 问题有解时，算法能否保证返回一个解?
- 问题无解时，算法能否保证返回 failure?

最优性

- 能否找到最优解?
- 返回代价/路径耗散最小的路径?

复杂性：算法需求的时间和内存

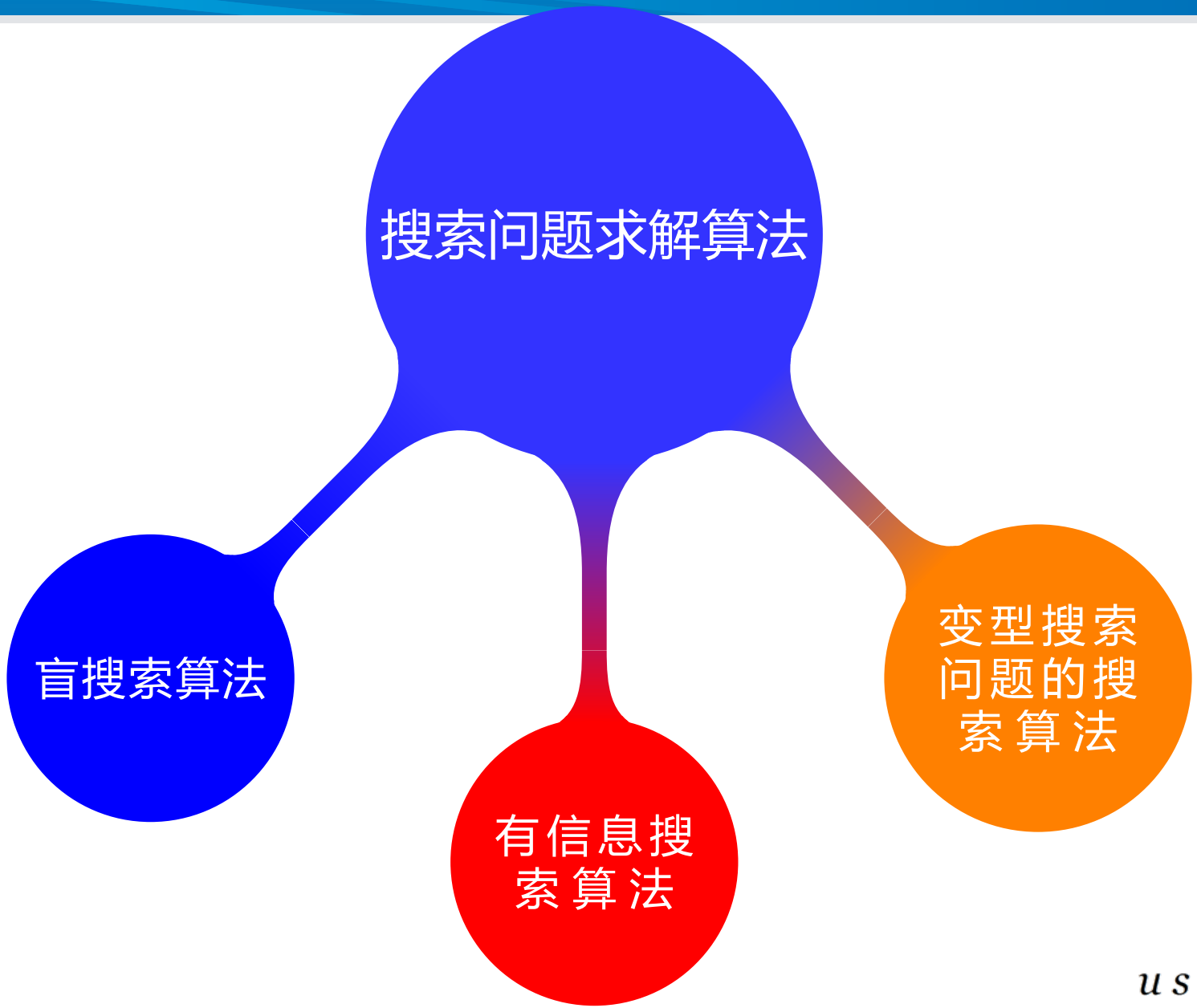
- 搜索树的大小：初态、后继函数和搜索策略决定，影响因素：分支因子（后继的最大个数），最浅目标状态的深度，路径的最大长度
- 时间复杂度：访问过的节点数目
- 空间复杂度：同时保存在内存中节点数目的最大值

搜索算法设计的目标

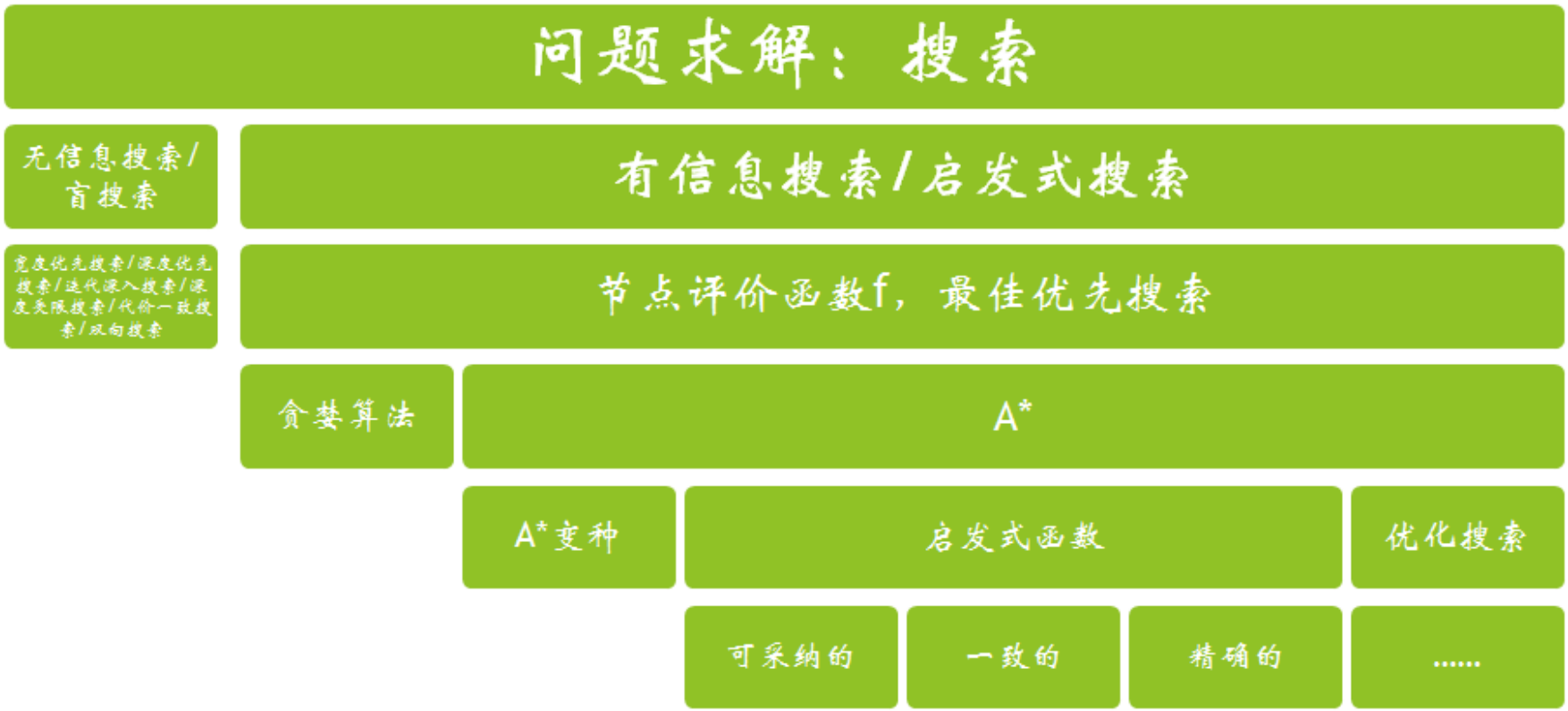
算法设计的根本目标：解决问题

- 很多搜索问题是 NP-hard，比如 TSP，数码问题，求解时间是问题规模的指数函数；
- 因此，别期望能在多项式时间内（时间复杂度是问题规模的多项式函数）求解出问题的所有实例（instances），目前没有通用算法！
- 没有免费的午餐定理表明：任意一个算法在所有问题实例上的平均性能是相同的。
- 算法设计的意义：对每个 instance/每类 instances 找到其最有效(尽可能高效）的求解算法。

搜索问题的求解算法分类



搜索学习路线图：部分


问题求解：搜索
无信息搜索/盲搜索
有信息搜索/启发式搜索
宽度优先搜索/深度优先搜索/迭代深入搜索/深度受限搜索/代价一致搜索/双向搜索
节点评价函数f，最佳优先搜索
贪婪算法
A*
A*变种
启发式函数
优化搜索
可采纳的
一致的
精确的
.....