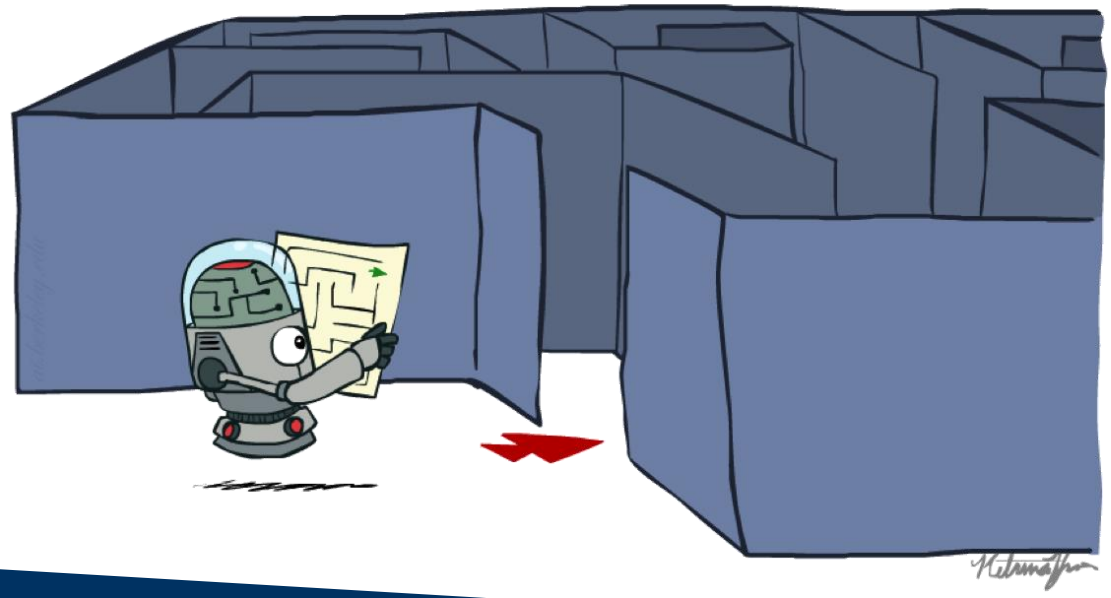
上一章节复习

- 人工智能基本概念：定义、评价、实现
- 人工智能发展简史：萌芽、诞生、发展
- 人工智能主要学派：符号、连接、行为
- 人工智能典型应用：AI+X、视觉、语言

第二章

搜索



计算机解决问题有哪些可行手段？

问题求解定义

- 考虑一种抽象的统一说法：
  - 一个问题就是从一个实例描述集合到解集合的映射：

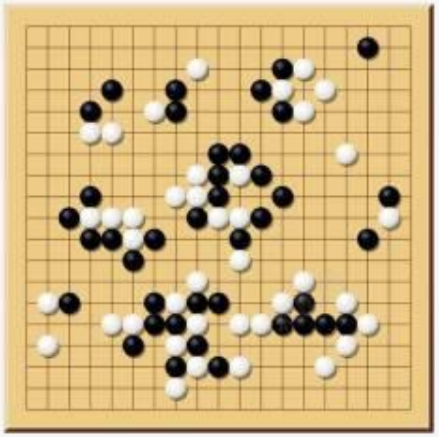
$$T: I \rightarrow O$$

  - 令$i \in I$ 或 $(i, o) \in I \times O$，则称i为问题T的实例
    - 如果I是有穷集，T就是有穷问题
    - $o = T(i)$为问题T对应i的解

- 计算机求解问题：
  - 计算机只会做一件事：自动执行程序。用计算机解决一个问题，就要写出一个解决这个问题的有穷程序（无穷长的程序写不完）
  - 如果针对问题 T 写了程序P，把 T 的任意实例 i 作为P的输入，P 的输出总是$o = T(i)$，我们就可以说P解决了问题T。

基于算法的系统（物理规则）
(Algorithm-Based Systems, ABS)

- 算法要求对问题（**规则**）的全面认识
- 任何有穷问题都有算法。但，以围棋为例：
  - 对任何棋局，确定下一个棋子的最佳位置。
    - 有穷问题。理论上存在最佳算法；
  - 可能局面太多（约为$3^{361} \approx 10^{172}$）
    - 分析每个局面需要的时间太长；
  - 程序太长，没有足够的存储器存放；



- 能用算法解决的问题，必须满足条件：
  - 可计算：建模后得到的抽象问题是可计算的
  - 有限长度：算法存在有穷长度的描述，而且描述不“过于长”
  - 有限时间：每个实例的求解都能在合理时间内完成

注：宇宙总原子个数为10的80次方

基于搜索的系统（暴力主义）
(Searching-Based Systems, SBS)

- 算法的局限
  - 算法需要领域专业知识，缺乏全面认识
  - 无明显规则约束，较难用显式算法建模(流体力学，人类未知等)
- 另一类方法：搜索，通过探查和回溯的方法寻找解。
  - 围棋，博弈树搜索（AlphaGo，AlphaZero）
    - 理论上可设计一个穷尽搜索引擎，找出最佳棋招
    - 实际上状态空间太大（$3^{361} \approx 10^{172}$）计算开销无法承受
    - 应用各种搜索策略，如限制最大搜索深度，加入随机性因素，设法在不能穷尽搜索的情况下合理地评估局面
- 搜索的优势：可能利用部分知识解决问题
- 搜索的缺陷：搜索状态爆炸←固有矛盾→规则集不完全

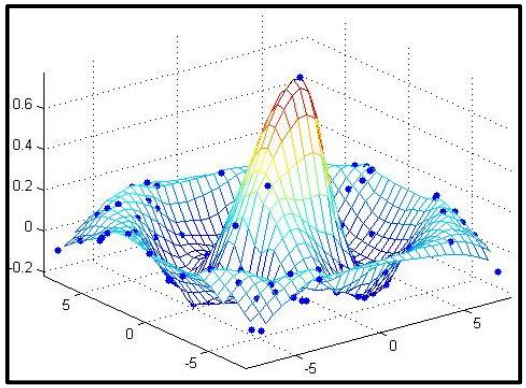
基于实例的系统（经验主义）

（Case-Based Systems, CBS）

- 对问题的了解非常少，或基本没有有关求解的知识，还能用计算机解决问题吗？
- 给定一组输入和结果$\{(i_1, o_1), (i_2, o_2), \ldots (i_n, on)\}$是一个问题的实例，就可能实现一个简单求解程序（死记硬背）

```
if x is i1 then o1

elif x is i2 then o2

... ...

elif x is in then on

else "I can't handle it"
```

拟合



- 推而广之：“归纳” 或 “学习” -> 机器学习
- 数据驱动的研究范式正在深刻变革人类研究的手段

算法 vs. 搜索

- 例子：给定三角形ABC的顶点和某点P的坐标，判断点P是否在ABC内。
  - 算法求解：内角和法，同侧法，重心法
- 例子：计算平方根

快速算法

```
1  float InvSqrt(float x)
2  {
3          float xhalf = 0.5f*x;
4          int i = *(int*)&x;
5          i = 0x5f3759df - (i >> 1);
6          x = *(float*)&i;
7          x = x*(1.5f - xhalf*x*x);
8          return x;
9  }
```

二分法

以二分法求 $\sqrt{2}$

为例，1的平方等于1，2的平方等于4，4 > 2，因为函数单调递增，所以 $\sqrt{2}$

的值肯定是在（1，2）之间，然后取中间数1.5，而1.5的平方等于2.25，2.25 > 2，进一步缩小范围，

的值肯定是在（1，1.5）之间，再取中间数1.25，1.25的平方等于1.5625，1.5625 < 2，则 $\sqrt{2}$

的值肯定是在（1.25，1.5）之间，依次类推就可以一步步的逼近真实值。

https://www.cnblogs.com/nsnow/archive/2010/08/09/1796111.html

搜索求解策略

- 在求解一个问题时：
  - 问题的表示，找到一个合适的表示方法
  - 选择一种相对合适的求解方法。由于绝大多数需要人工智能方法求解的问题缺乏直接求解的方法，因此，搜索不失为一种求解问题的一般方法。Search is useful for agents that plan ahead!

- 本章概要：
  - 搜索的基本概念
  - 状态空间知识表示和搜索策略
    - 盲目搜索
    - 启发式搜索

本章主要内容

- 搜索的基本概念
- 状态空间表示方法
- 盲目搜索策略
- 启发式搜索策略

搜索的基本问题与主要过程

- 搜索中需要解决的基本问题：
  - （1）是否一定能找到一个解 Completeness
  - （2）找到的解是否是最佳解 Optimality
  - （3）时间与空间复杂性如何 Time & space complexity
  - （4）是否终止运行或是否会陷入一个死循环 Goal test
- 搜索的主要过程：
  - （1）出发点：初始（正向）或目标（逆向）状态，当前状态；(从哪来)
  - （2）扫描操作算子集，选择适当的操作，生成新状态 (我是谁)
  - （3）检查所生成的新状态是否满足结束状态， (将去哪)
    - 满足：得到问题的一个解，包含解答路径
    - 不满足，将新状态作为当前状态，返回第(2)步再进行搜索。

搜索方向

（1）数据驱动：从初始状态出发的正向搜索

- 正向搜索——从问题给出的条件出发。

（2）目的驱动：从目的状态出发的逆向搜索

- 逆向搜索——从想达到的目的入手，看哪些操作算子能产生该目的以及应用这些操作算子产生目的时需要哪些条件。

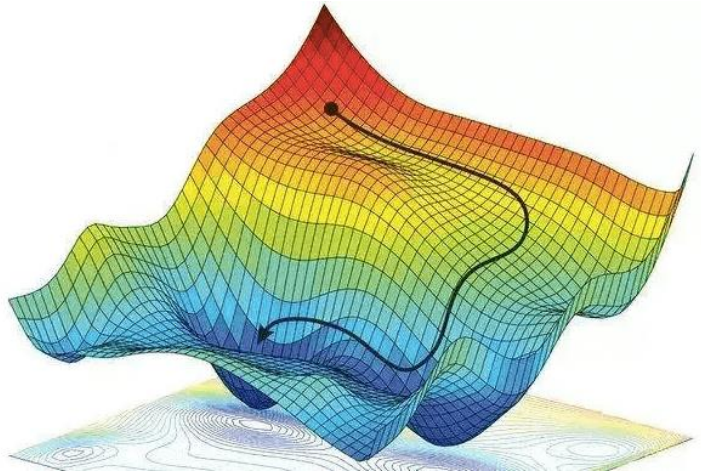
（3）双向搜索

- 双向搜索——从开始状态出发正向搜索，同时又从目的状态出发逆向搜索，直到两条路径在中间的某处汇合为止。

搜索策略

（1）盲目搜索：在不具有对特定问题的任何有关信息的条件下，按固定的步骤（依次或随机调用操作算子）进行的搜索。

（2）启发式搜索：考虑特定问题领域可应用的知识，动态地确定调用操作算子的步骤，优先选择较适合的操作算子，尽量减少不必要的搜索，以求尽快地到达结束状态。

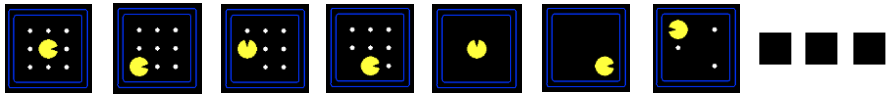


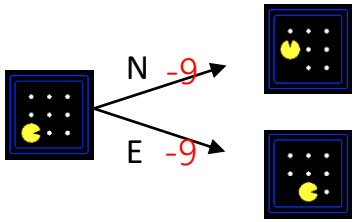
本章主要内容

- 搜索的基本概念
- 状态空间表示方法
- 盲目搜索策略
- 启发式搜索策略

搜索问题组成

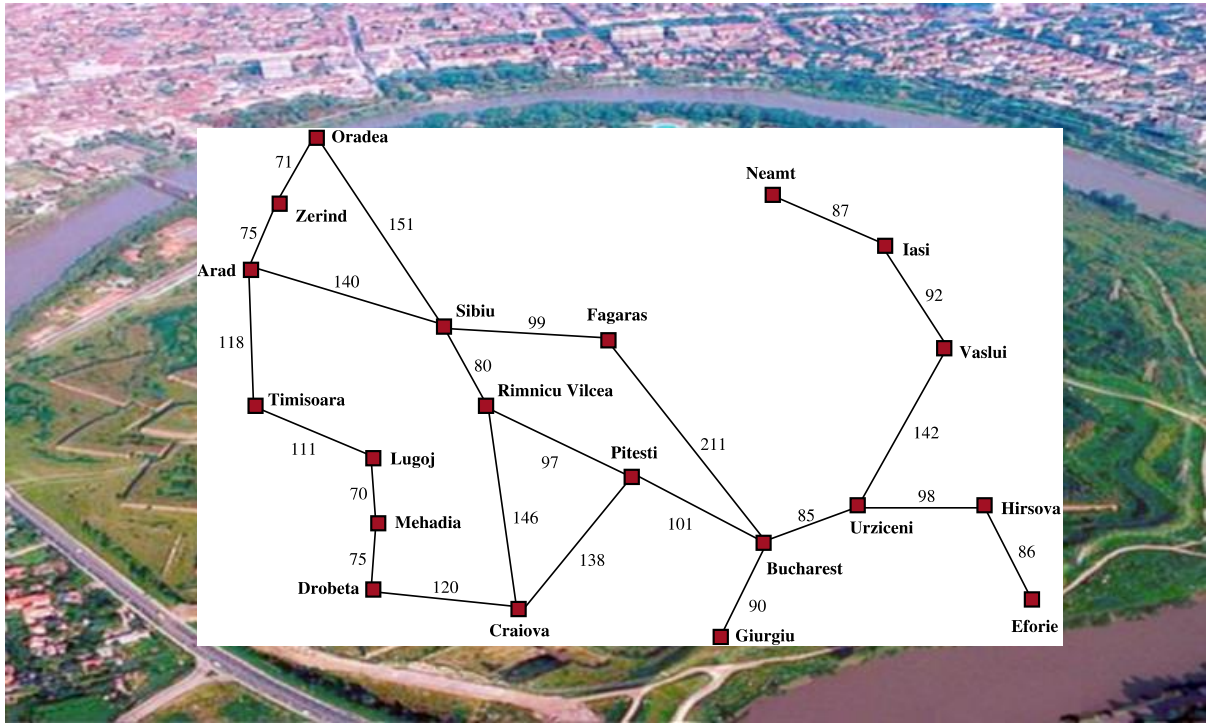
- 一个搜索问题包含:
  - 状态空间 S
  - 初始状态 s_0
  - 每个状态下的动作集合 $\mathcal{A}(s)$
  - 转移模型 $Result(s,a)$
  - 目标验证 $G(s)$
    - 没有豆子剩下的 S
  - 动作代价 $c(s,a,s')$
    - +1 per step; -10 food; -500 win; +500 die; -200 eat ghost





- 一个解是一个能达到目标状态的动作序列
- 最优解有所有解中最低的cost

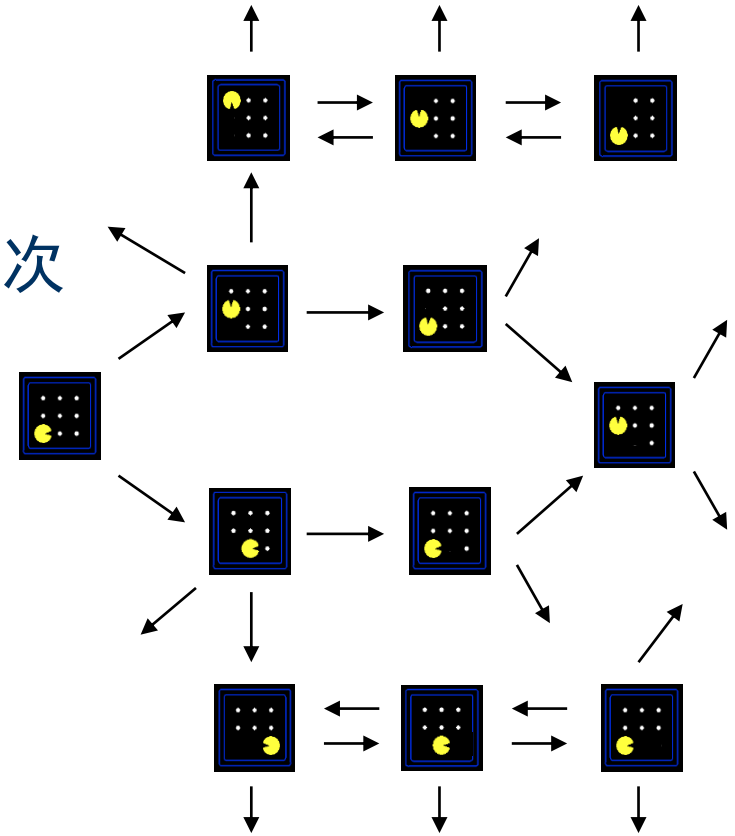
Example: Traveling in Romania



- 状态空间:
  - 访问的城市
- 初始状态
  - Arad
- 动作:
  - 旅行到相邻的城市
- 转移模型:
  - 到达相邻的城市
- 目标检验:
  - s = Bucharest?
- 动作代价:
  - 从 s to s' 的路径长度
- 解是什么?

状态空间图

- 状态空间图：搜索问题的数学表达
  - 节点 (抽象了) 环境的状态
  - 边表示状态转移 (标记出造成转移的动作)
  - 目标检验表示为目标节点 (可能唯一)
- 状态空间图中，每个状态只出现一次
- 实际上很难完整画出状态空间图
  - 占用过多存储空间
  - 节点和边的思想非常有用！



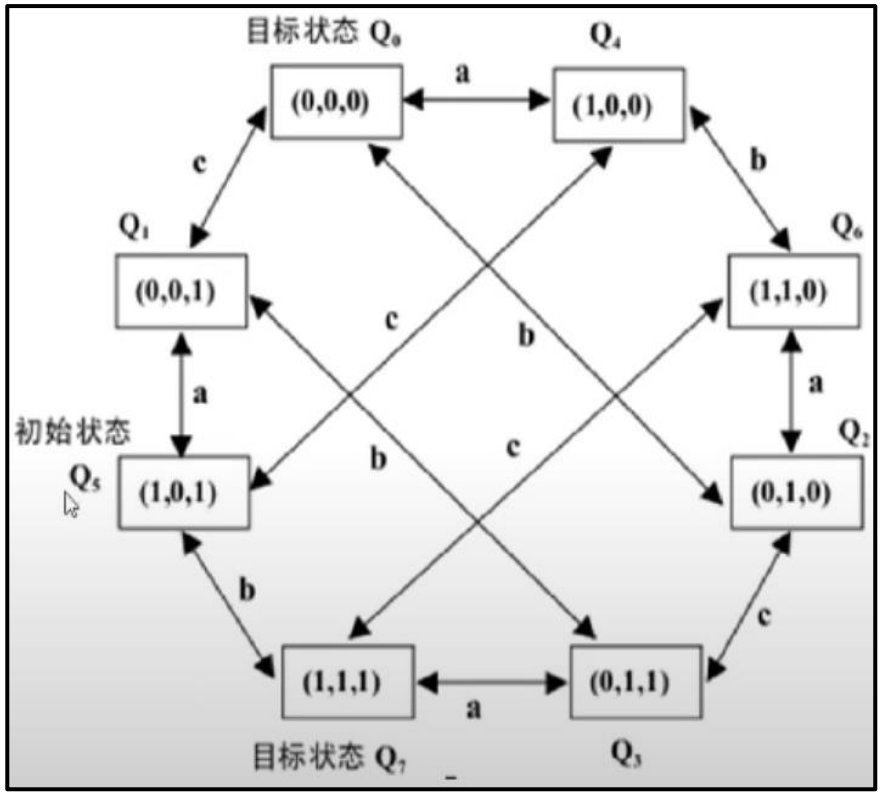
钱币翻转示例

初始状态

目标状态

目标状态

状态空间图



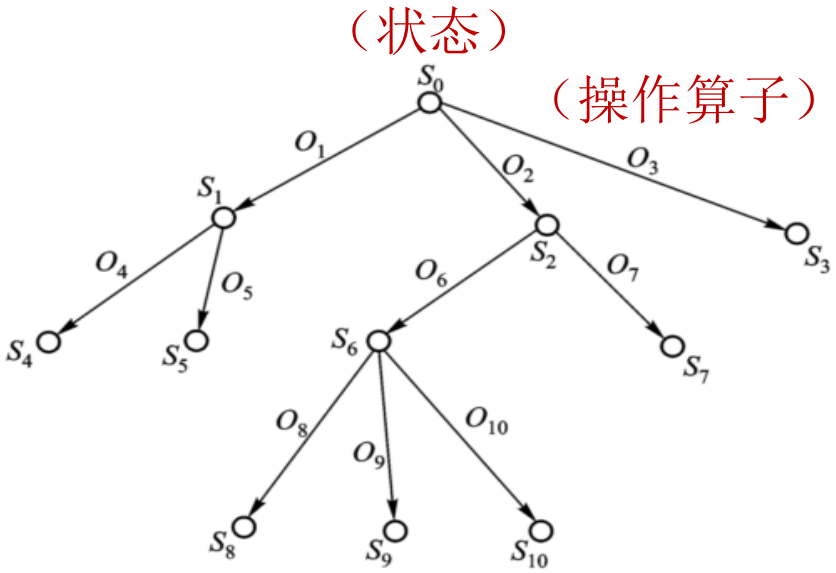
1代表正、0代表反

随着变量增多搜索空间几何增长

如互联网用户画像的推荐系统

搜索树

- 用有向图描述搜索过程
- 结点表示状态、边表示求解步骤
- 初始状态为根结点
- 初始状态转换为目的状态的操作算子序列←→图中寻找某一路径（搜索的一个解）



八数码问题的搜索树

- 例：八数码问题的状态空间

2	3	1
5		8
4	6	7

初始状态

1	2	3
8		4
7	6	5

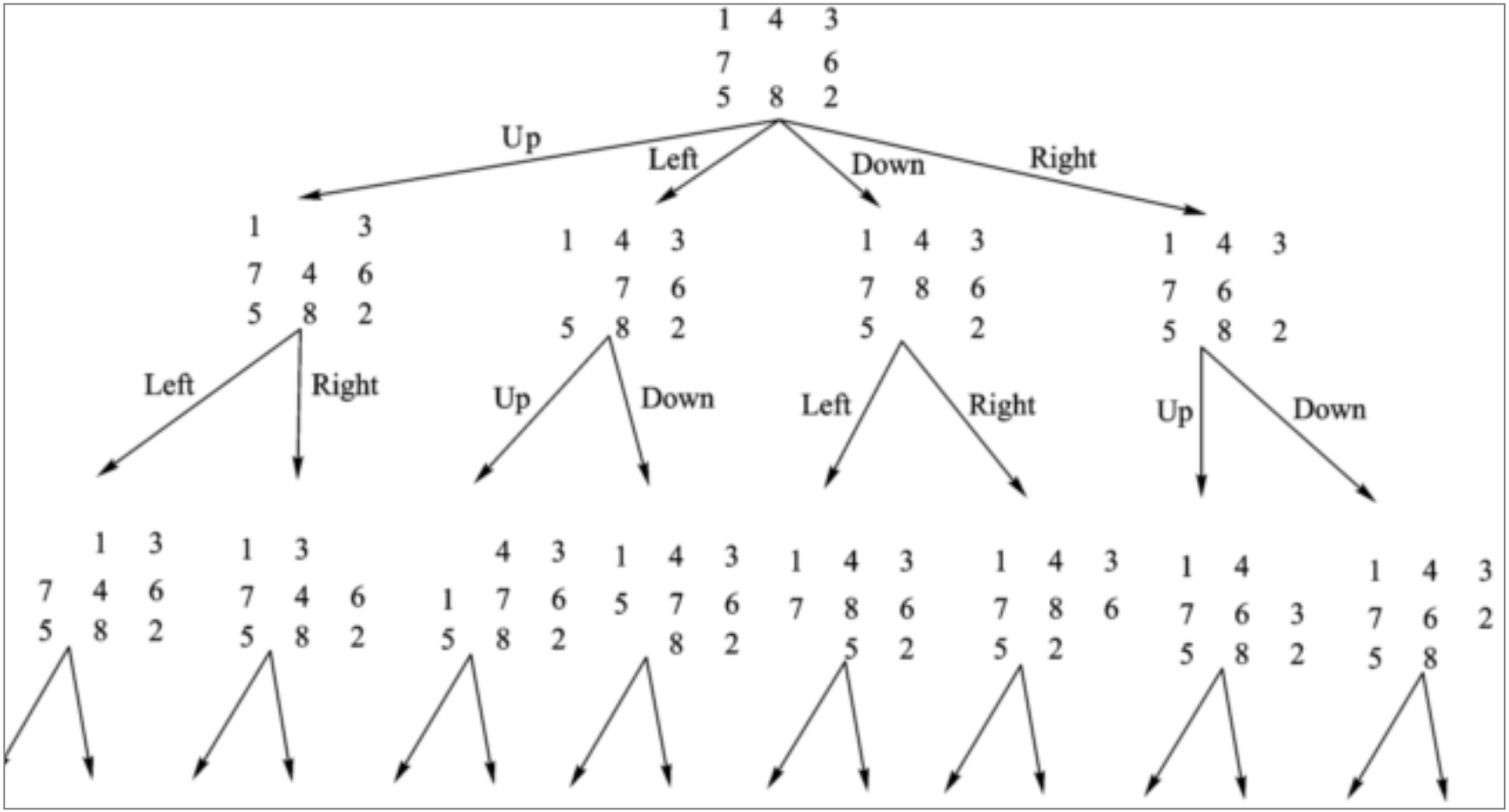
目标状态

状态集 S：所有摆法

操作算子：

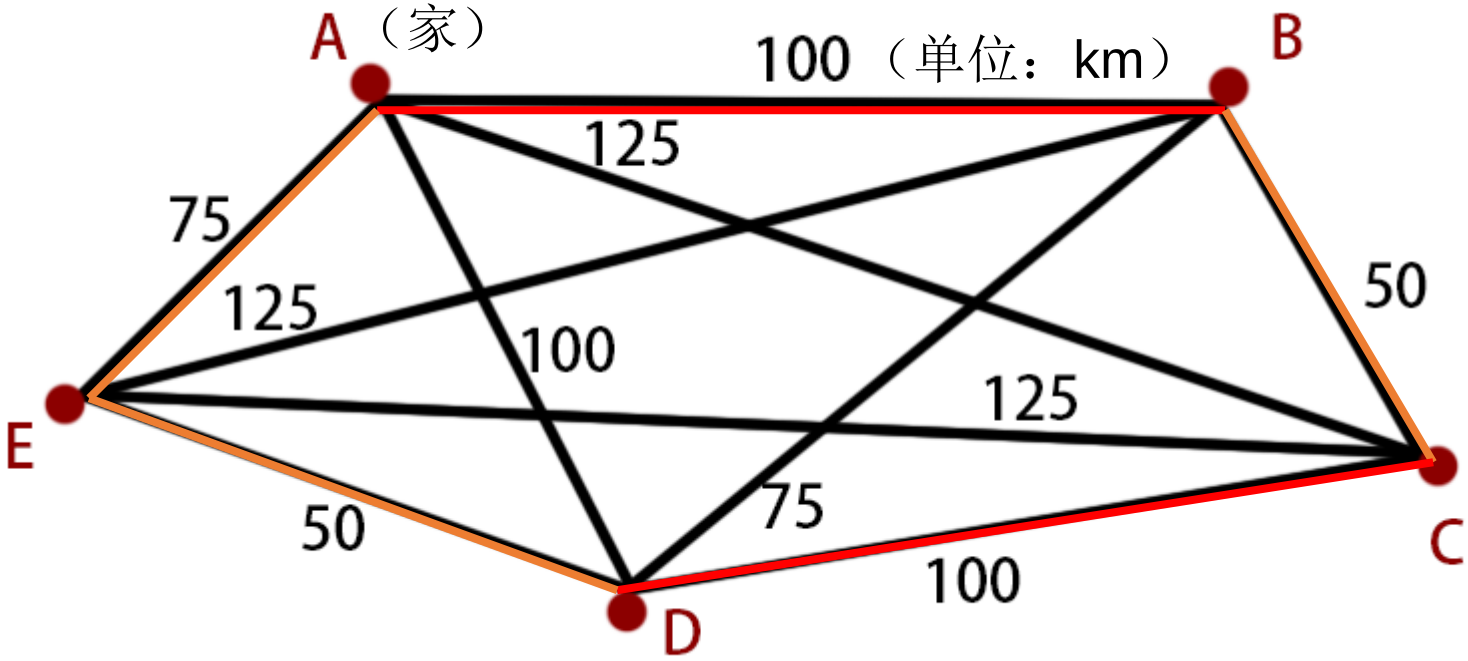
- 将空格向上移Up
- 将空格向左移Left
- 将空格向下移Down
- 将空格向右移Right

八数码问题的搜索树



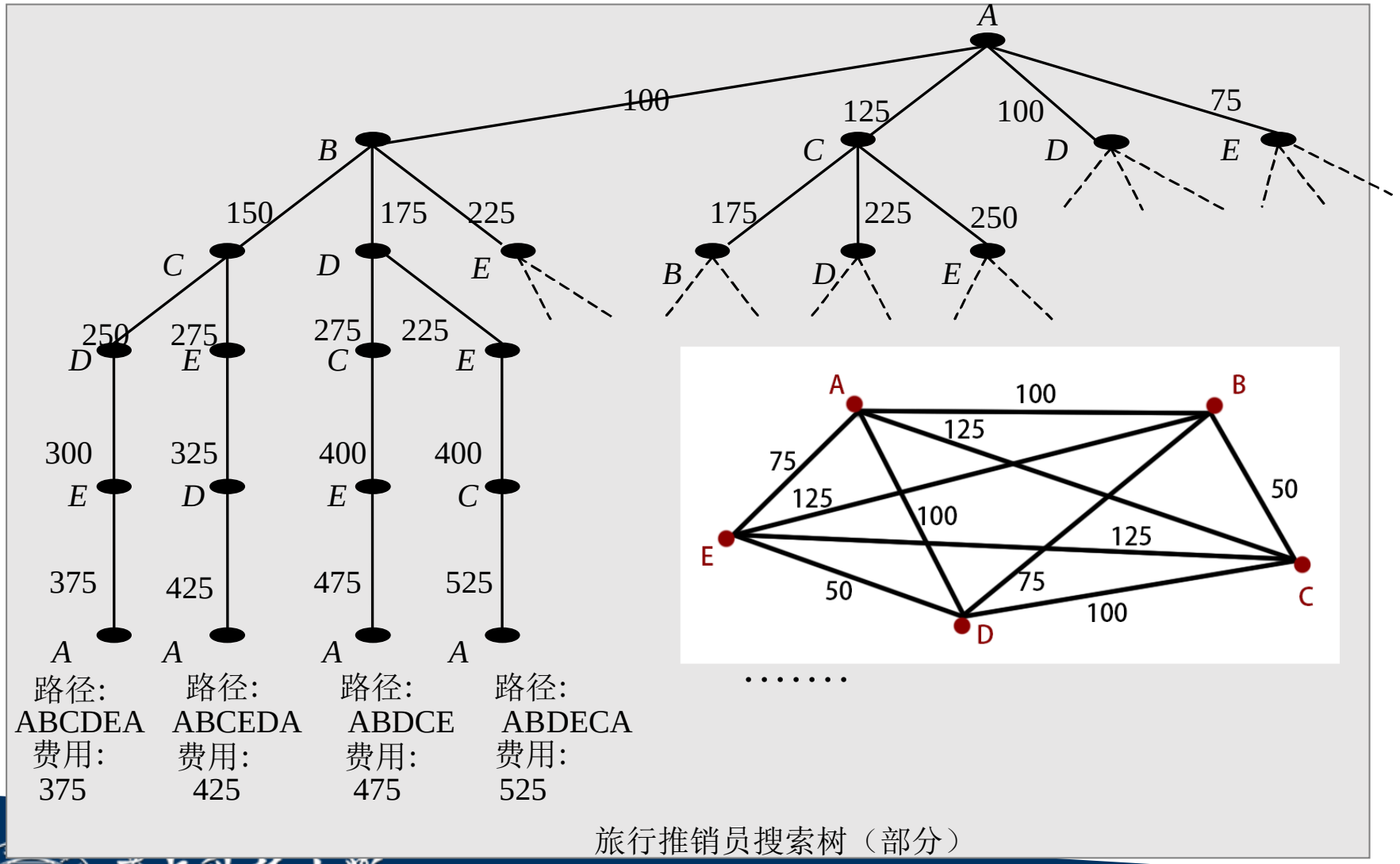
旅行商问题的搜索树

- 例： 旅行商问题（traveling salesman problem, TSP）



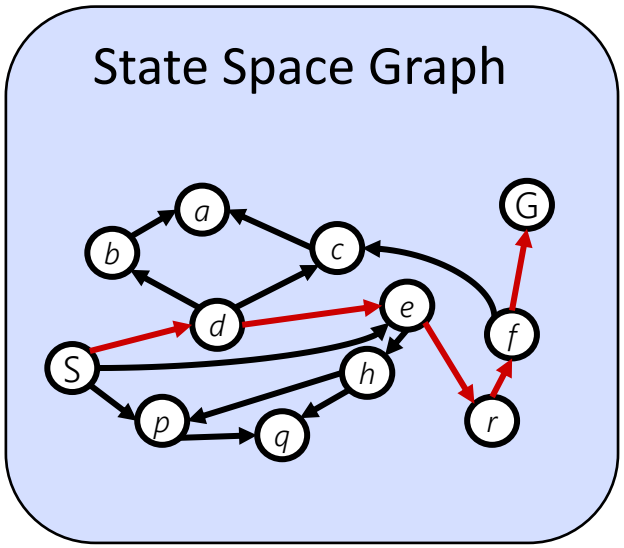
可能路径：费用为375的路径（A，B，C，D，E，A）

旅行商问题的搜索树



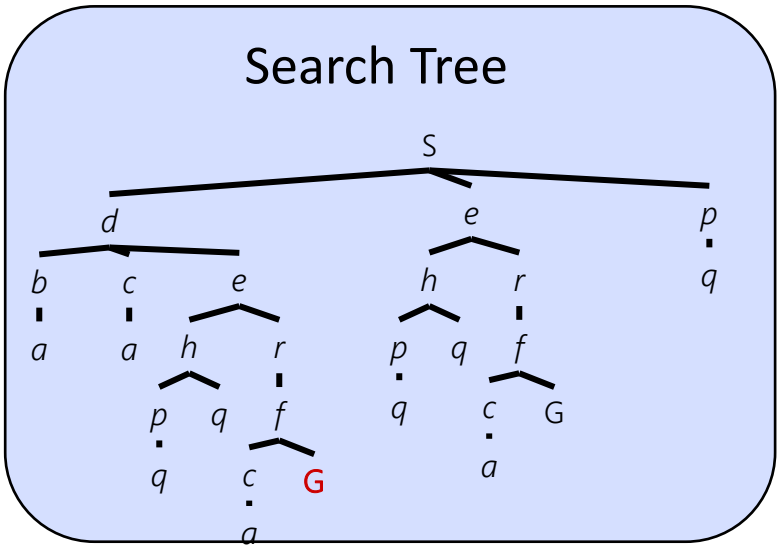
旅行推销员搜索树（部分）

状态空间图 vs. 搜索树



状态空间图中每个状态只出现一次，搜索树中可以出现多次

边搜索边生成有向图，生成的无用状态越少，搜索效率越高，对应的搜索策略就越好



状态空间图：

- 准确表示问题各状态间关系，直观地反映问题的实际结构
- 可表示问题环和共享子问题，有助于避免重复搜索和计算
- 空间复杂度高，存储和管理开销大，不利于解决大规模问题

搜索树：

- 搜索树以树形结构直观表示问题解决过程，直观地展示了从初始状态到目标状态的路径。
- 可按需逐步构建，无需一次性表示所有可能的状态和转换，易于实现，解决大规模问题
- 可能存在多条表示相同状态转换的路径，造成搜索和计算冗余

本章主要内容

- 搜索的基本概念
- 状态空间表示方法
- 盲目搜索策略
- 启发式搜索策略

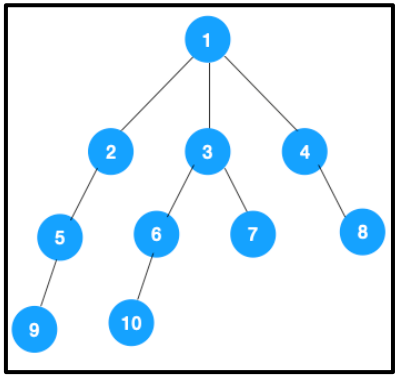
- 图搜索策略：如何从还没有被扩展的节点中优先选择一个进行扩展，以便尽快地找到一条符合条件的路径。不同的选择方法就构成了不同的图搜索策略。
- 盲目搜索：没有问题相关先验信息
- 启发式搜索：利用了与问题相关的知识或启发信息

深度优先搜索

- 深度优先搜索（Depth-first search, DFS）：首先扩展最新产生的节点，深度相等的节点按生成次序的盲目搜索。

盲目≠随机

节点分类



- 通常来讲，在search的过程中，会将所有的点分成几类
  - Explored：已经搜索过，属于从起始点到当前节点之间最优路径上的点；
  - Frontier：当前已经生成的结点，其后裔节点等待被搜索，还不确定它们在不在最优路径上；
  - Unexplored：还没有预见到的节点。
- 搜索过程是：把Unexplored的点放到Frontier集合里面，然后把Frontier集合里的点挪到Explored集合里面。当最终的节点挪到Explored里面之后，整个搜索过程就结束了。

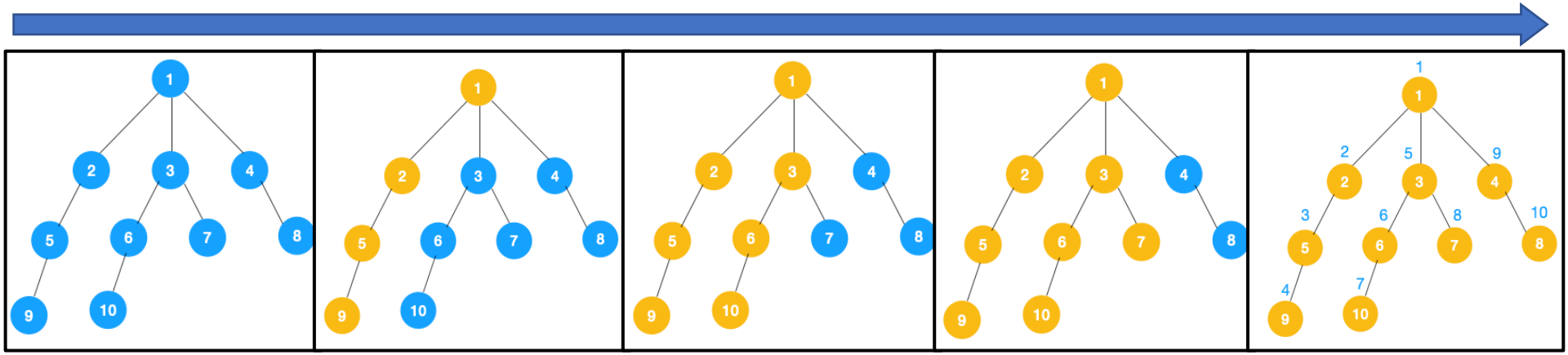
深度优先搜索示例

- 深度优先搜索（Depth-first search, DFS）：首先扩展最新产生的节点，深度相等的节点按生成次序的盲目搜索。

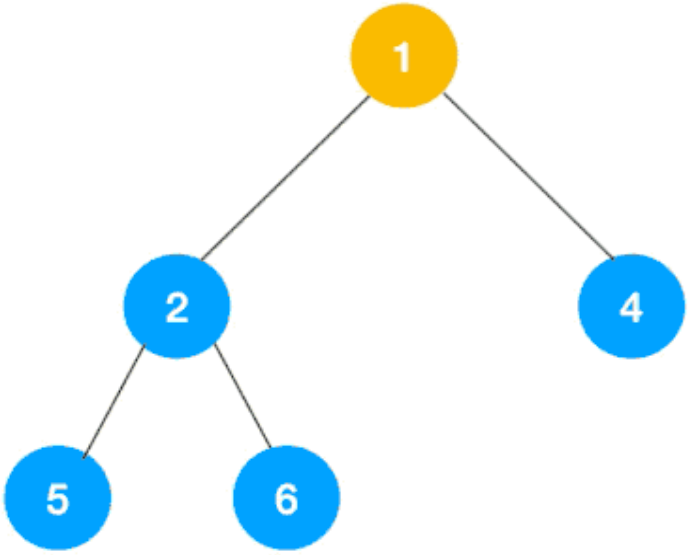
特点：

（1）扩展最深的节点使搜索沿状态空间某条单一路径从起始节点向下进行；

（2）仅当搜索到达一个没有后裔的状态时，才考虑另一条替代的路径。



深度优先搜索算法实现



根节点压栈

遍历节点

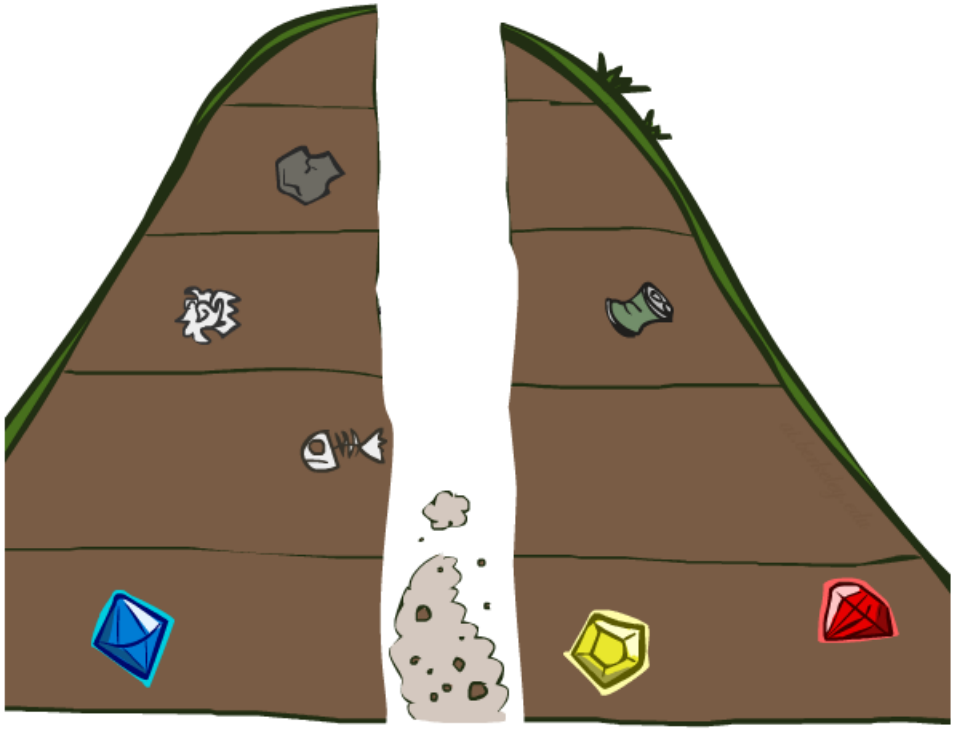
```
1.  /**
2.   * 使用栈来实现 dfs
3.   * @param root
4.   */
5.  public static void dfsWithStack(Node root) {
6.      if (root == null) {
7.          return;
8.      }
9.
10.     Stack<Node> stack = new Stack<>();
11.     // 先把根节点压栈
12.     stack.push(root);
13.     while (!stack.isEmpty()) {
14.         Node treeNode = stack.pop();
15.         // 遍历节点
16.         process(treeNode)
17.
18.         // 先压右节点
19.         if (treeNode.right != null) {
20.             stack.push(treeNode.right);
21.         }
22.
23.         // 再压左节点
24.         if (treeNode.left != null) {
25.             stack.push(treeNode.left);
26.         }
27.     }
28. }
```

深度优先搜索

- 在深度优先搜索中，当搜索到某一个状态时，它所有的子状态以及子状态的后裔状态都必须先于该状态的兄弟状态被搜索。
- 为了保证找到解，应选择合适的深度限制值，或采取不断加大深度限制值的办法，反复搜索，直到找到解。
- 对任何状态而言，以后的搜索有可能找到另一条通向它的路径。如果路径的长度对解题很关键的话，当算法多次搜索到同一个状态时，它应该保留最短路径。

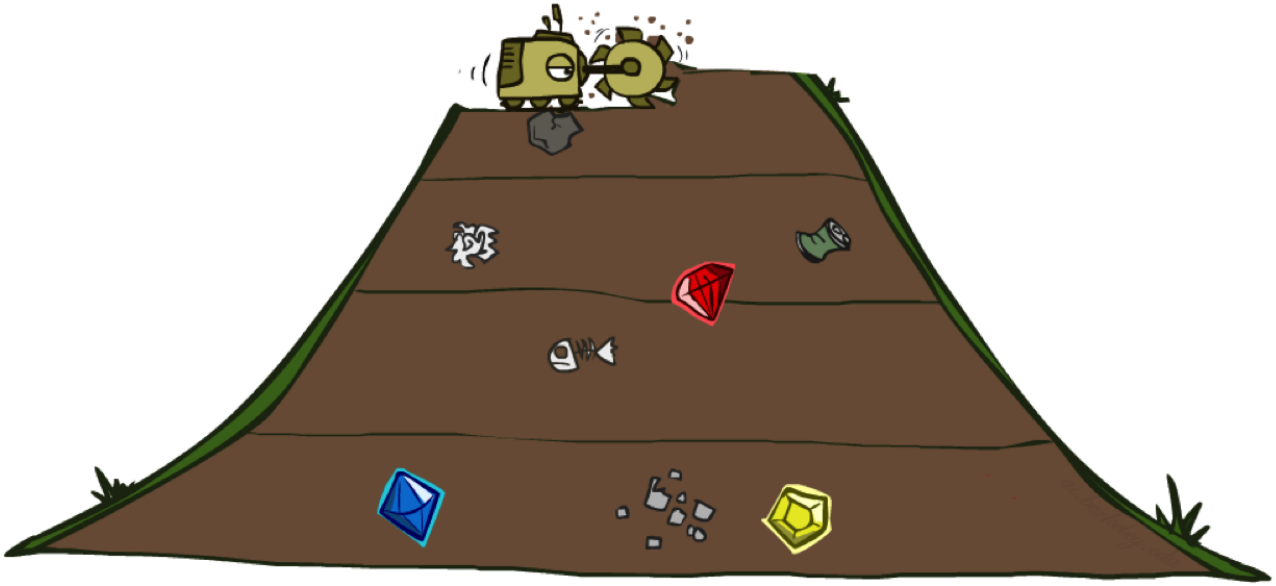
深度优先搜索

- 深度优先搜索并不能保证第一次搜索到的某个状态时的路径是到这个状态的最短路径。



宽度优先搜索

- 宽度优先搜索（Breadth-first search，BFS）：以接近起始节点的程度（深度）为依据，进行逐层扩展的节点搜索方法。



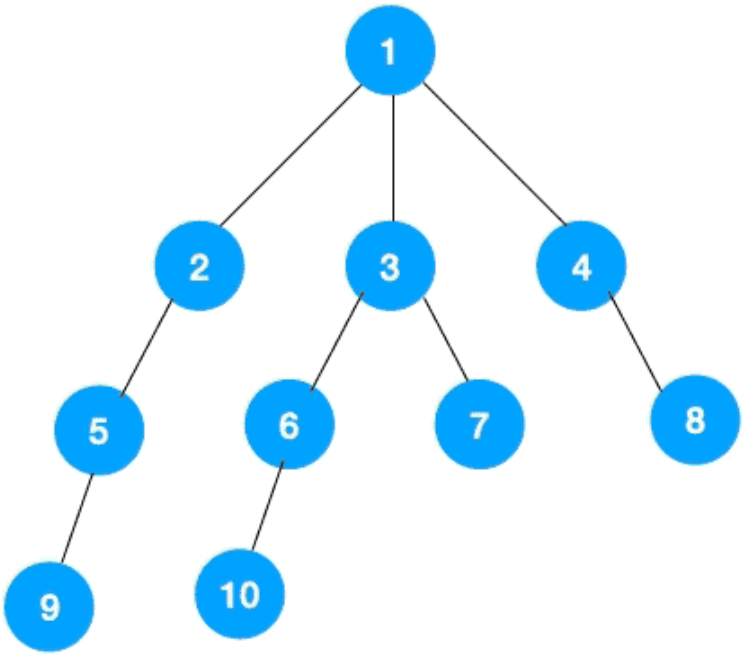
宽度优先搜索

- 宽度优先搜索（Breadth-first search，广度优先搜索）：以接近起始节点的程度（深度）为依据，进行逐层扩展的节点搜索方法。

特点：

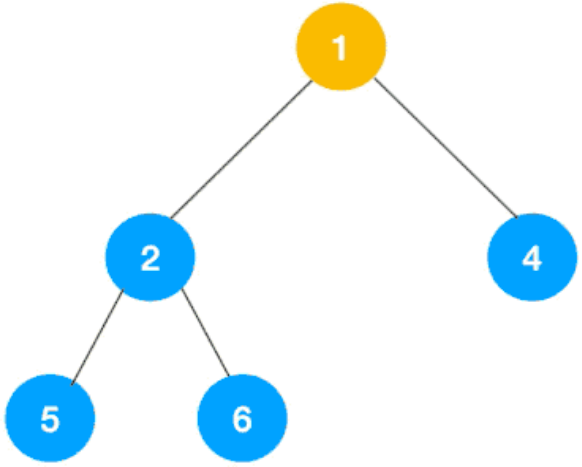
（1）每次选择深度最浅的节点首先扩展，搜索是逐层进行的；

（2）一种高价搜索，但若有解存在，则必能找到它。



宽度优先算法搜索过程示例

深度优先与宽度优先搜索实现对比

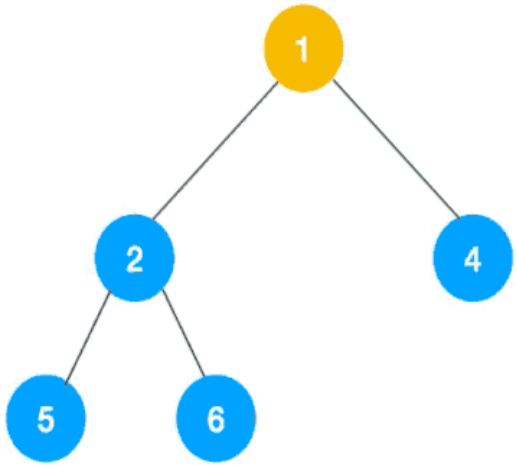


根节点压栈

栈

1

遍历节点



根节点压栈

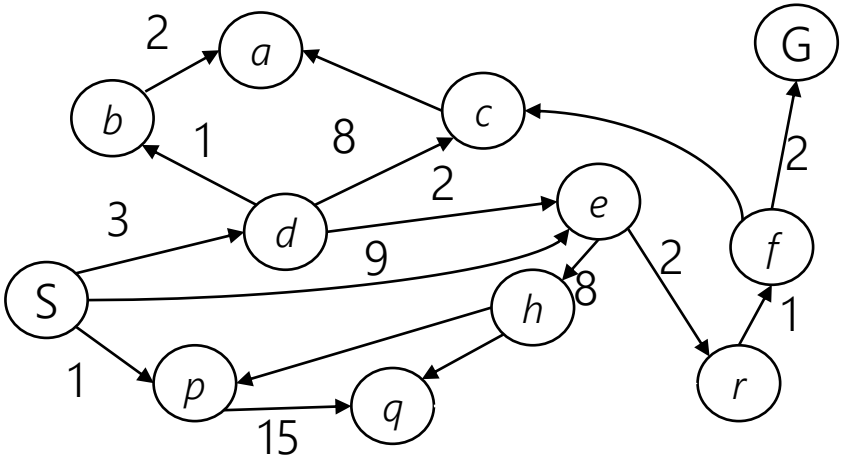
队列

1

遍历节点

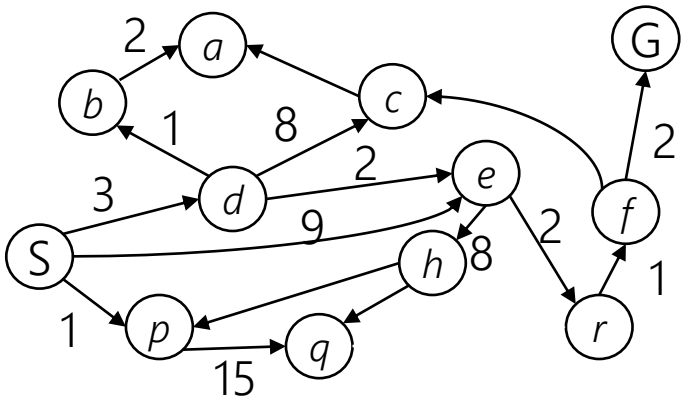
统一代价搜索

- 宽度优先搜索适于处理无权重图搜索问题
- 统一代价搜索（Uniform cost search， UCS）是对宽度优先的改进，可以解决有权重图搜索问题
- 节点的扩展选择以g(n)=从根节点到节点n的代价为依据，选择g(n)最小的节点扩展

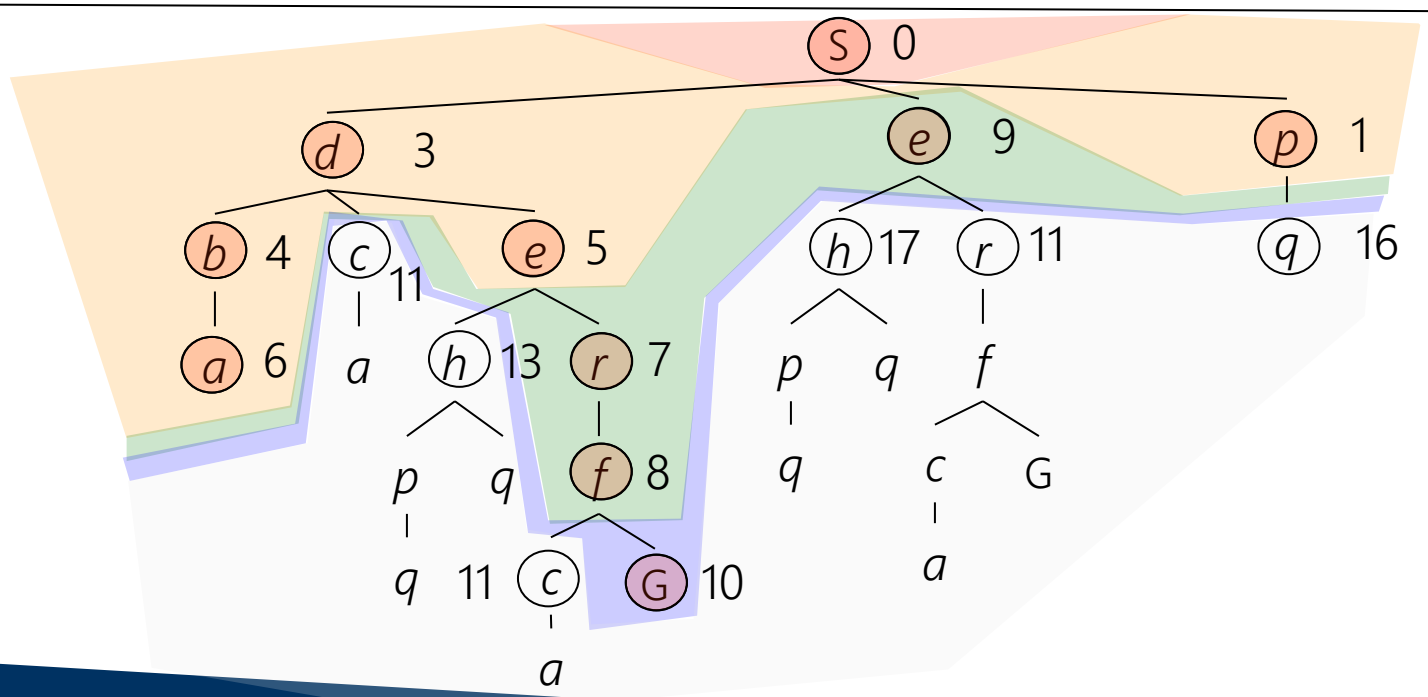


统一代价搜索

g(n)=从根节点到节点n的代价

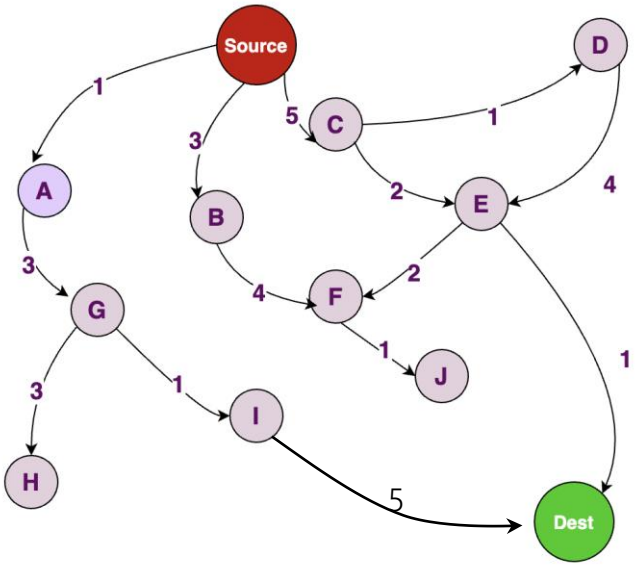


Cost contours



统一代价搜索-练习

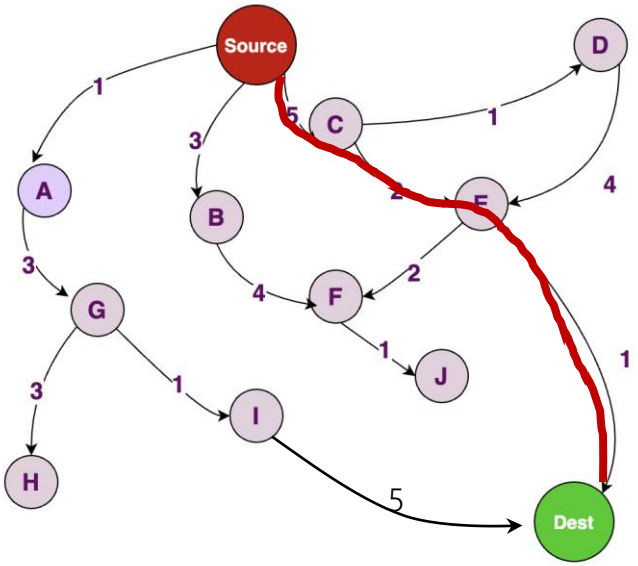
A B G I C D E H F J Dest



The minimum distance between the source and destination nodes is 8.

1 3 4 5 5 6 7 7 7 8 8

统一代价搜索-答案



The minimum distance between the source and destination nodes is 8.

本章主要内容

- 搜索的基本概念
- 状态空间表示方法
- 盲目搜索策略
- 启发式搜索策略

启发式策略

- 启发式信息：用来简化搜索过程有关具体问题领域的特性信息叫做启发信息
- 启发式图搜索策略（利用启发信息的搜索方法）的特点：重排OPEN表，选择最有希望的节点加以扩展
- 适合用启发式策略的两种基本情况：

（1）由于存在问题陈述和数据获取的模糊性，可能会使待求解问题没有一个确定的解；

（2）虽然一个问题可能有确定解，但是其状态空间特别大，搜索中生成扩展的状态数会随着搜索的深度呈指数级增长。

X+Y =2 ➡ X=？　　s.t. X,Y为整数 X=? ➡ s.t. X,Y＞0　X=1

启发信息和估价函数

- 在具体求解中，能够利用与该问题有关的信息来简化搜索过程，称此类信息为启发信息。
- 启发式搜索：利用启发信息的搜索过程。
- 按运用的方法分类：
  （1）陈述性启发信息：用于更准确、更精炼地描述状态
  （2）过程性启发信息：用于构造操作算子
  （3）控制性启发信息：表示控制策略的知识
- 按作用分类：
  （1）用于扩展节点的选择，即用于决定应先扩展哪一个节点，以免盲目扩展
  （2）用于生成节点的选择，即用于决定要生成哪些后继节点，以免盲目生成过多无用的节点
  （3）用于删除节点的选择，即用于决定删除哪些无用节点，以免造成进一步的时空浪费

启发信息和估价函数

- 估价函数（evaluation function）：估算节点“希望”程度的量度。
- 估价函数值 $f(n)$：从初始节点经过n节点到达目标节点的路径的最小代价估计值，其一般形式是

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

- $g(n)$：从初始节点 S_0 到节点 n 的实际代价；
- $h(n)$：从节点 n 到目标节点 S_g 的最优路径的估计代价，称为启发函数。

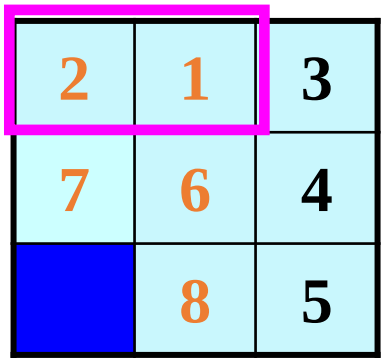
$h(n)$比重大：降低搜索工作量，但可能导致找不到最优解；

$h(n)$比重小：一般导致工作量加大，极限情况下变为盲目搜索，但可能可以找到最优解。

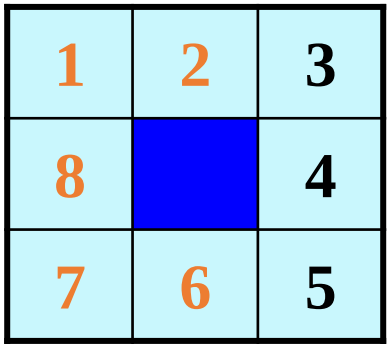
启发信息和估价函数

例 八数码问题的启发函数：

- 启发函数1：取一棋局与目标棋局相比，其位置不符的数码数目，例如$h(S_0) = 5$；
- 启发函数2：各数码移到目标位置所需移动的距离的总和，例如$h(S_0) = 6$；
- 启发函数3：对每一对逆转数码乘以一个倍数，例如3倍，则$h(S_0) = 3$；
- 启发函数4：将位置不符数码数目的总和与3倍数码逆转数目相加，例如$h(S_0) = 8$。

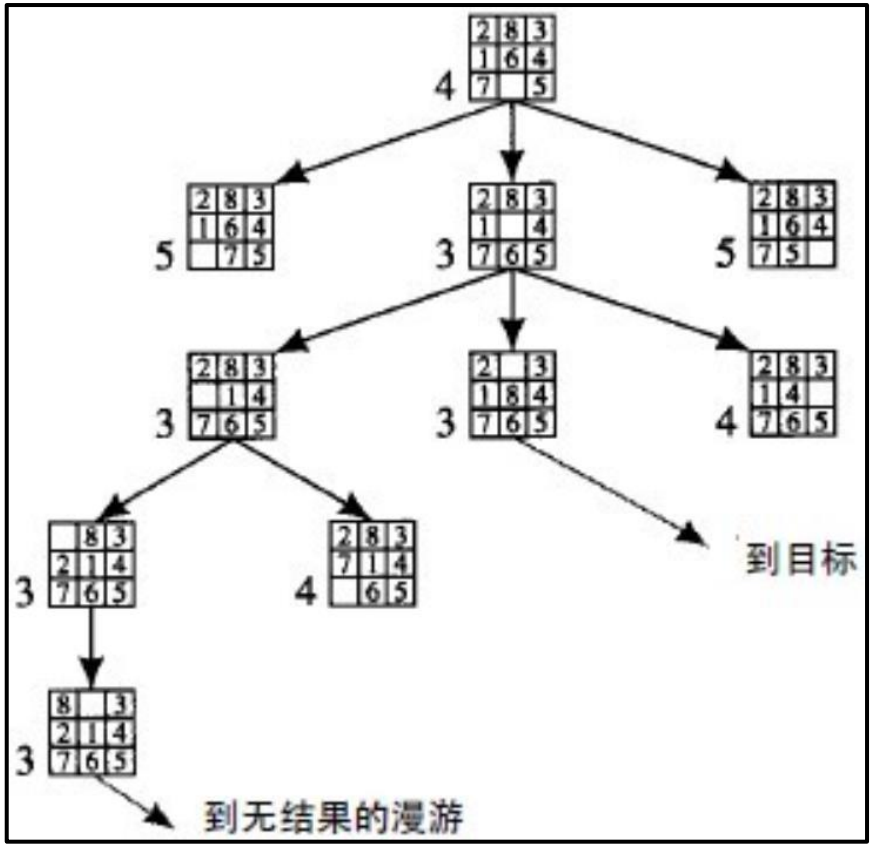


初始棋局

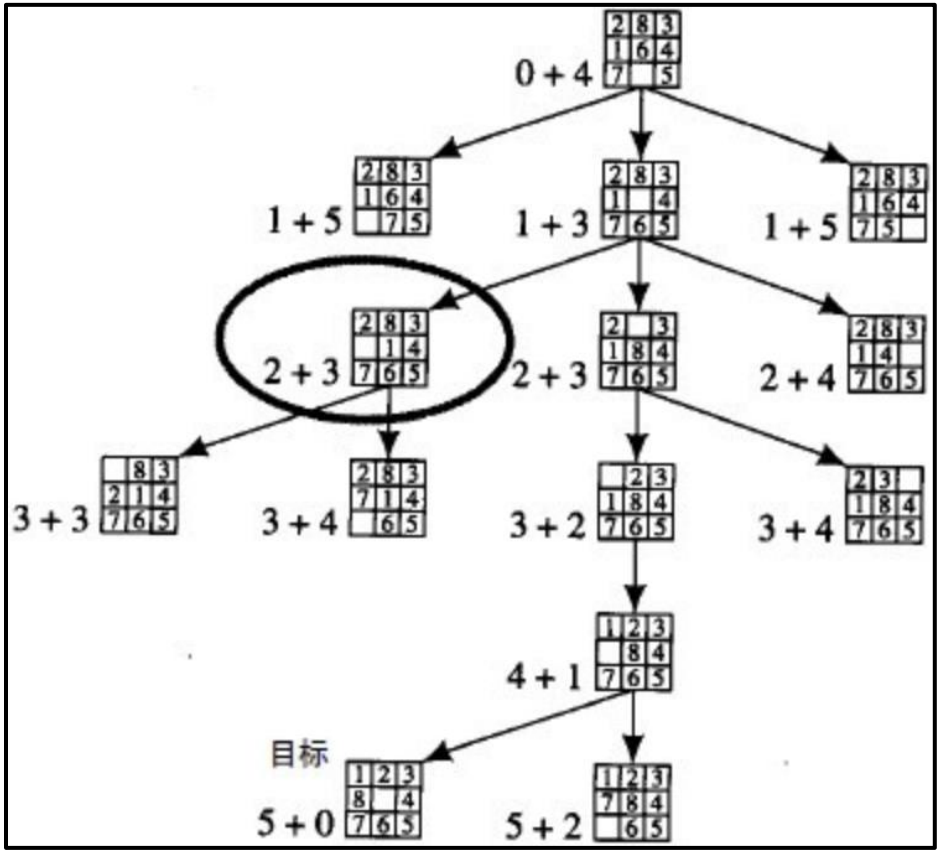


目标棋局

启发搜索示例



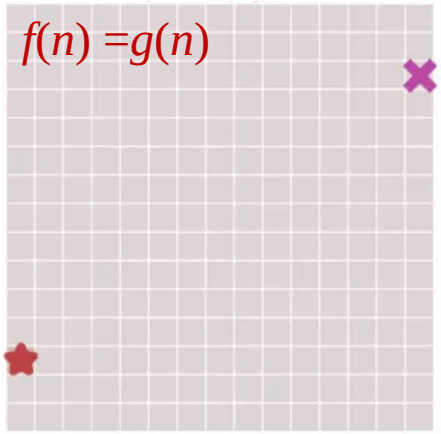
启发式搜索可能的结果



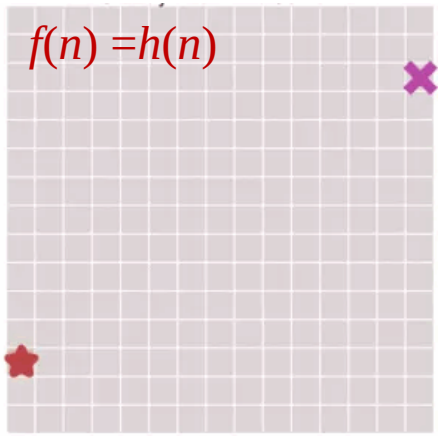
g(n)深度+h(n)不对位置个数

最佳优先搜索

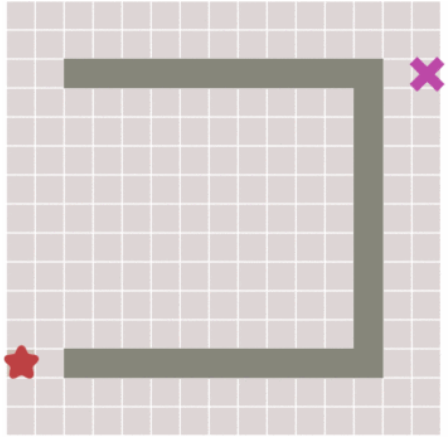
- 最佳优先搜索算法（Best-First-Search）是一种启发式搜索算法，基于宽度优先搜索，用启发估价函数对将要被遍历到的点进行估价，然后选择代价小的进行遍历（估价函数 $f(n)=h(n)$），直到找到目标节点或者遍历完所有点，算法结束。
- 最佳优先算法不能保证找到的路径是一条最短路径，但是其计算过程相对于统一代价搜索UCS算法会更快。



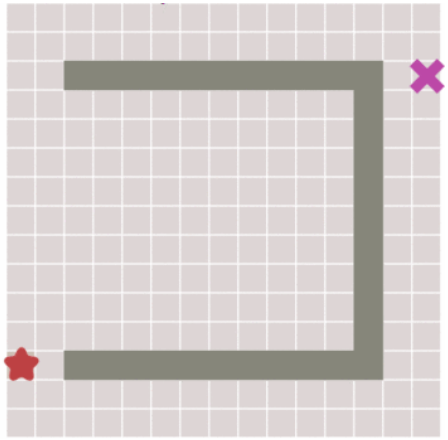
统一代价搜索



最佳优先搜索



统一代价搜索



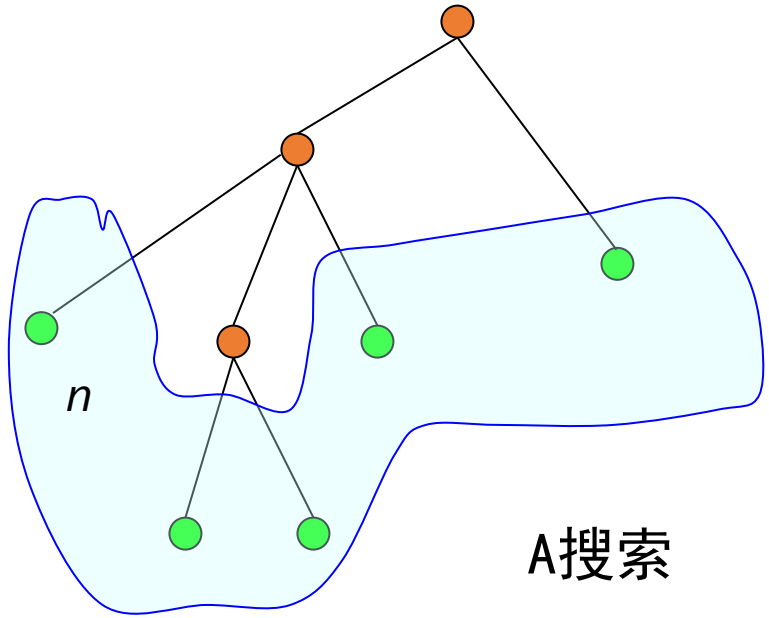
最佳优先搜索

A搜索

- A搜索算法：使用了估价函数 f 的最佳优先搜索。
  - 估价函数 $f(n) = g(n) + h(n)$
  - 如何寻找并设计启发函数$h(n)$，然后以$f(n)$的大小来排列OPEN表中待扩展状态的次序，每次选择 $f(n)$ 值最小者进行扩展。

$g(n)$：状态n的实际代价，例如搜索的深度；

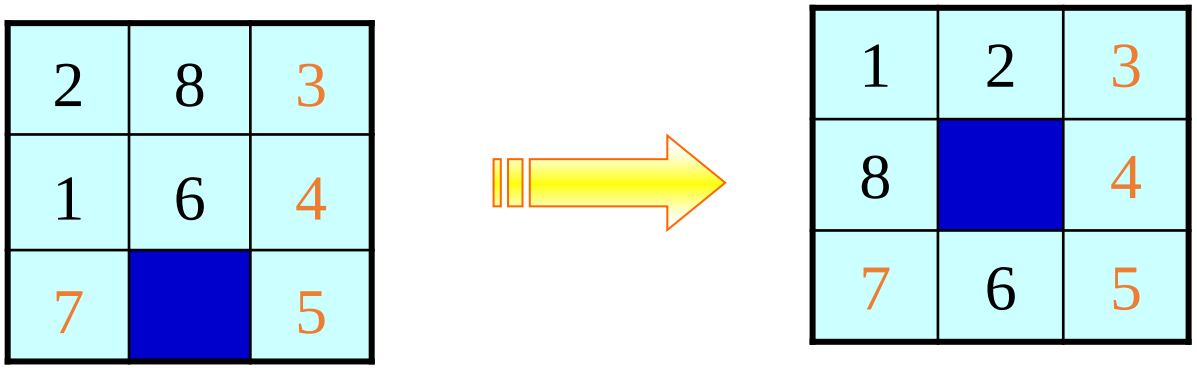
$h(n)$：对状态n与目标“接近程度”的某种启发式估计。



A搜索

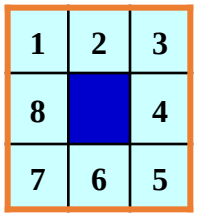
例：利用A搜索算法求解八数码问题，问最少移动多少次就可达到目标状态？

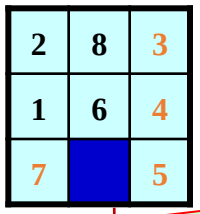
- 估价函数定义为 $f(n) = g(n) + h(n)$
- $g(n)$：节点 n 的深度，如 $g(S_0)=0$。
- $h(n)$：节点 n 与目标棋局不相同的位数（包括空格），简称“不在位数”，如 $h(S_0)=5$。



初始状态 S_0　　目标状态

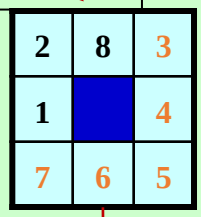
操作算子集： ↑, ↓, →, ←



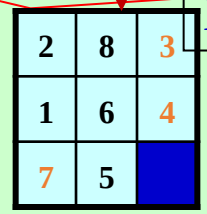


S（0+5=5）

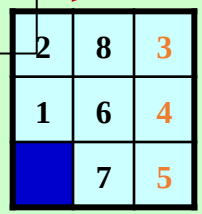
A（1+3=4）



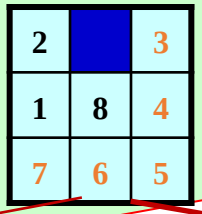
B（1+6=7）

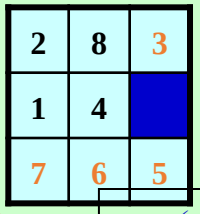


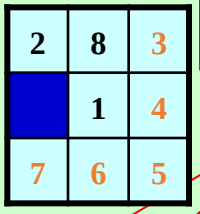
C（1+6=7）



D（2+4=6）

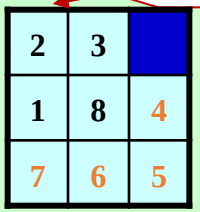


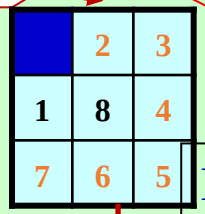




F（2+4=6）

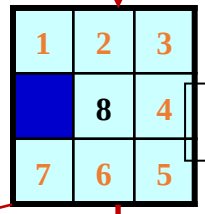
E（2+5=7）





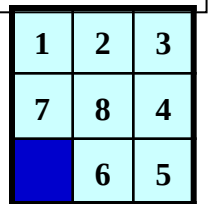
H（3+3=6）

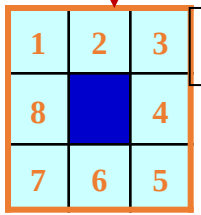
G（3+5=8）



I（4+2=6）

J（5+3=8）





K（5+0=5）

- 问题：A搜索算法能不能保证找到最优解（路径最短的解）？

$g(n)$：从初始节点 S 到节点 n 的实际代价

$h(n)$：对状态n与目标节点接近程度的某种启发式估计

为目的状态，停止搜索 退出

A*搜索

- A*搜索算法（最佳图搜索算法）：

定义$h^*(n)$为状态n到目的状态的最优路径代价，当A搜索算法的启发函数$h(n)$小于等于$h^*(n)$，即满足

$$h(n) \le h^*(n), \quad \text{对所有结点n}$$

时，被称为A*搜索算法。

- 如果某一问题有解，那么利用A*搜索算法对该问题进行搜索则一定能搜索到解，并且一定能搜索到最优的解。

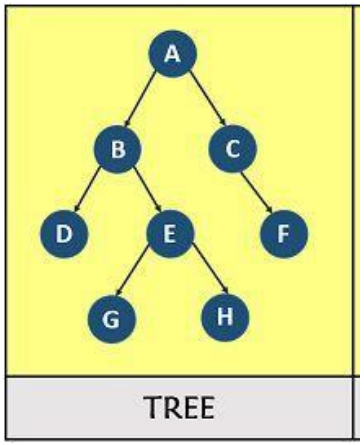
- 八数码问题中定义$h(n)$ 表示节点 n 与目标棋局不相同的位数（不包括空格，满足上述条件，因此其A搜索树也是A*搜索树，所得解路径为最优解路径。

A*搜索的最优性条件

- 启发函数的可采纳性（Admissibility)：

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

当一个搜索算法在最短路径存在时能保证找到它



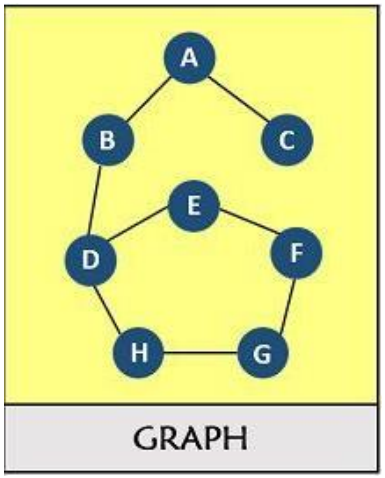
对应的树搜索是最优的

上例中的八数码问题中定义的$h(n)$表示了“不在位”的数码数，满足上述条件，因此所得解路径为最优解路径。

A*搜索的最优性条件

- 启发函数的一致性/单调性（Consistency / Monotonicity）：

对于每个节点 n 和其每个后继节点 n'



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

$c(n, n')$是从节点 n 到节点 n' 的实际代价。

图搜索中A*最优性的条件

一般来说，大多数可采纳的启发式函数都是单调的。

补充-启发函数的信息性

- 信息性：
  - 在两个A*启发策略的h_1和h_2中，如果对搜索空间中的任一状态n都有$h_1(n) \le h_2(n)$，就称策略h_2比h_1具有更多的信息性。
  - 如果某一搜索策略的$h(n)$越大，则A*算法搜索的信息性越多，所搜索的状态越少。
  - 但更多的信息性可能需要更多的计算时间，也有可能抵消减少搜索空间所带来的益处。

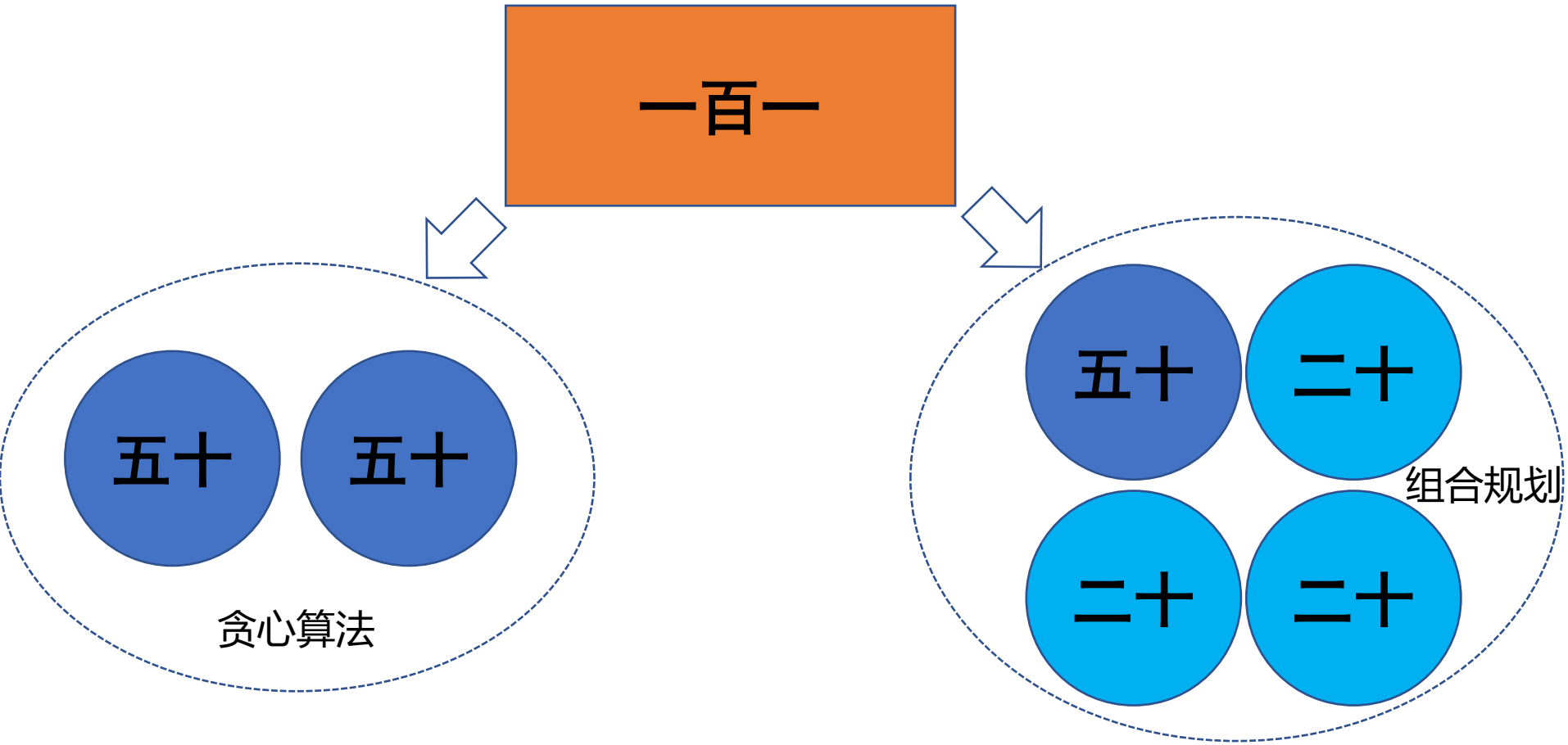
本章主要内容

- 搜索的基本概念
- 状态空间表示方法
- 盲目搜索策略
- 启发式搜索策略
- 扩展内容（Optional）

贪心算法

- 贪心（贪婪）算法：在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，它所做出的仅仅是在某种意义上的局部最优解。
  - 贪心算法不是对所有问题都能得到整体最优解，关键是贪心策略的选择，选择的贪心策略必须具备无后效性，即某个状态以前的过程不会影响以后的状态，只与当前状态有关。
  - 贪心的本质是通过每一步的局部最优，期望实现全局最优的一种算法思想。关键在于局部最优是否真的能实现全局最优。如果能实现，那么贪心算法基本上就是问题的最优解。

贪心算法

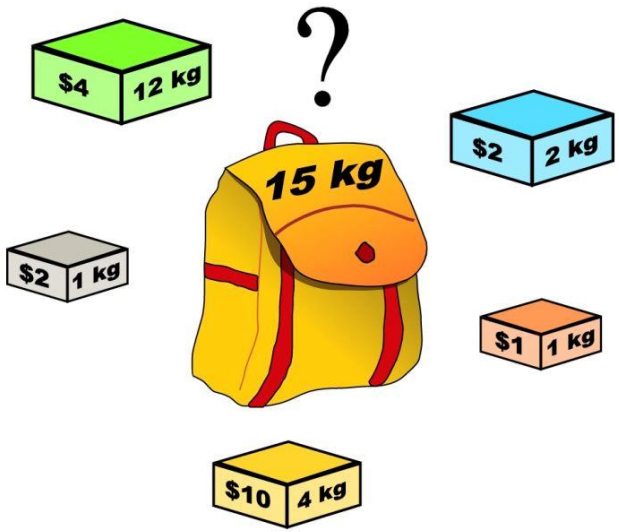


生活中处处可见：棋牌21点游戏更高复杂度的贪心游戏

贪心算法

- 例：背包问题

问题描述：有N件物品和一个最多能背重量为W 的背包。第i件物品的重量是weight[i]，得到的价值是value[i] 。**每件物品只能用一次**，求解将哪些物品装入背包里物品价值总和最大。



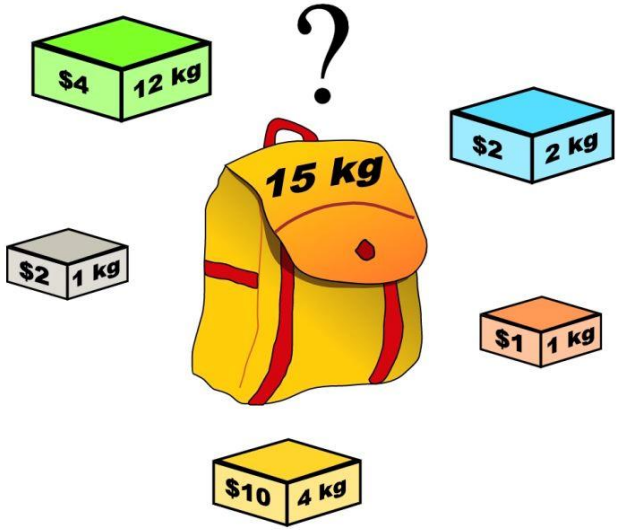
	物品1	物品2	物品3	物品4	物品5
价值$	4	2	10	1	2
重量kg	12	1	4	1	2

每一件物品其实只有两个状态，取或者不取，所以可以使用回溯法搜索出所有的情况，那么时间复杂度就是O(2^n)，这里的n表示物品数量

贪心算法

- 例：背包问题

	物品1	物品2	物品3	物品4	物品5
价值$	4	2	10	1	2
重量kg	12	1	4	1	2



贪心的三种策略：

1、从物品中选取价值最大的放入。 3 ~~1~~ 2 5 4

2、从物品中选区重量最小的放入。 2 4 5 3

3、选取物品的价值与重量的比值大的放入。 3 2 4 5

第一第二种不能保证得到最优解，第三种也不能保证得到最优解，但是能得到最优的近似解。

如果16Kg，价值最大该怎么选择？

Dijkstra算法

- Dijkstra算法是基于贪心思想实现的：

求解单源最短路径问题：在赋权有向图 G=(V, E, W) 中，假设每条边 E[i] 的长度为 w[i]，**找到由顶点 V1到其余各点的最短路径。**

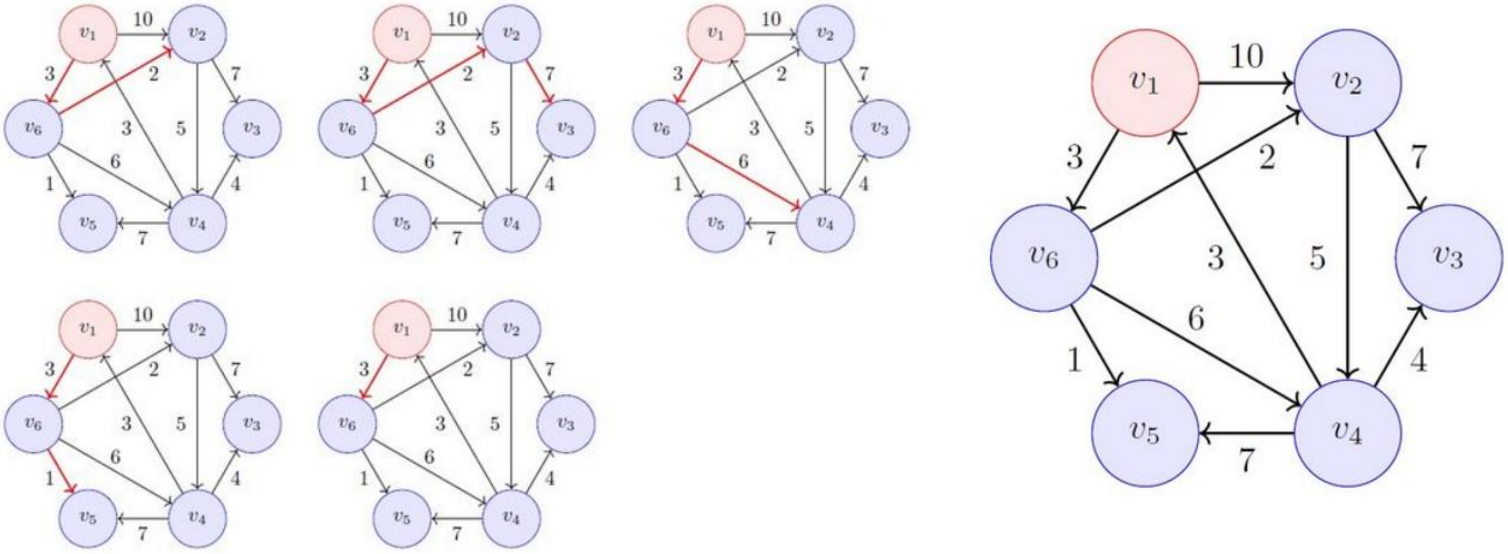


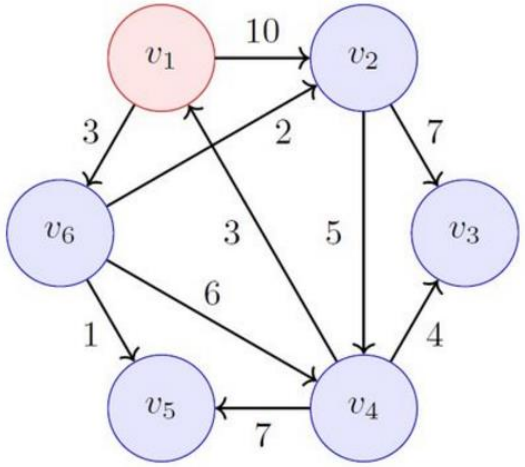
图　V中的源点s(v_1)到V中其余点的最短路径。

Dijkstra算法

- Dijkstra算法是基于贪心思想实现的：

把图中顶点集合V分成两组：

- 第一组为**已求出最短路径的顶点集合**（S）
- 第二组为其余**未确定最短路径的顶点集合**（V）



1. 初始 $S=\{v_1\}$ ；
2. 对于 $v_i \in V-S$ ，计算 $\mathrm{dist}[s,v_i]$ ；
3. 选择 $\min_{v_i \in V-S} \mathrm{dist}[s,v_i]$ ，并将这个 v_i 放进集合 S 中，更新 $V-S$ 中的顶点的 dist 值；
4. 重复 1 ，直到 $S=V$ 。

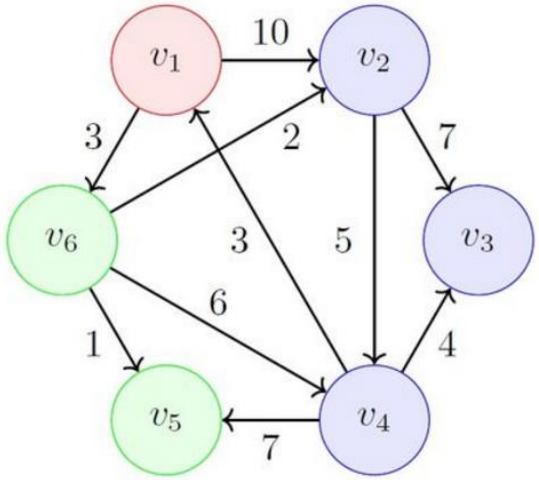
贪心

从源点到顶点的相对于集合S的最短路径

即从源点到顶点的路径中间只能经过已经包含在集合S中的顶点，不能直接到达的dist=∞

Dijkstra算法

- Dijkstra算法是基于贪心思想实现的：



$$S = \{v_1\}$$

$$\text{dist}[v_1, v_2] = w_{1,2} = 10 \quad \text{dist}[v_1, v_3] = \infty$$
$$\text{dist}[v_1, v_4] = \infty \quad \text{dist}[v_1, v_5] = \infty$$
$$\text{dist}[v_1, v_6] = w_{1,6} = 3 \surd \quad \text{dist}[v_1, v_1] = 0$$

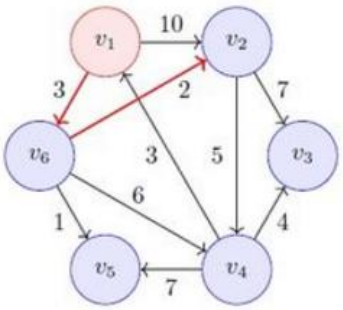
$$S = \{v_1, v_6\}$$

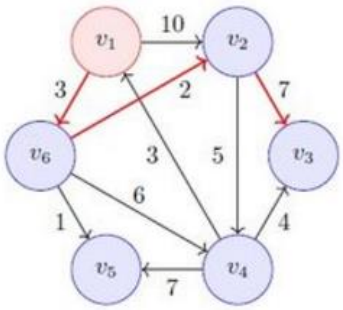
$$\text{dist}[v_1, v_2] = \text{dist}[v_1, v_6] + w_{6,2} = 5 \quad \text{dist}[v_1, v_3] = \infty$$
$$\text{dist}[v_1, v_4] = \text{dist}[v_1, v_6] + w_{6,4} = 9 \quad \text{dist}[v_1, v_5] = \text{dist}[v_1, v_6] + w_{6,5} = 4 \surd$$

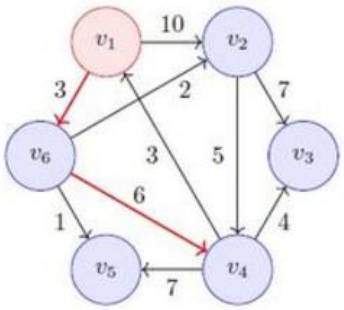
$$S = \{v_1, v_6, v_5\}$$

Dijkstra算法

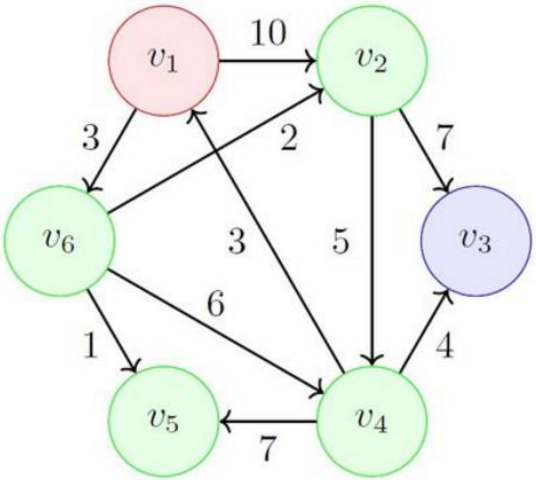
- Dijkstra算法是贪心算法的一个典型案例：

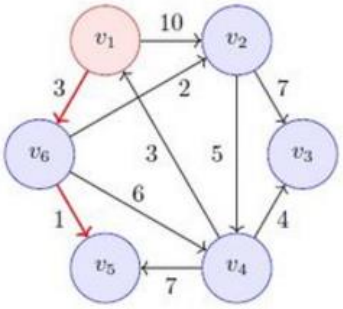


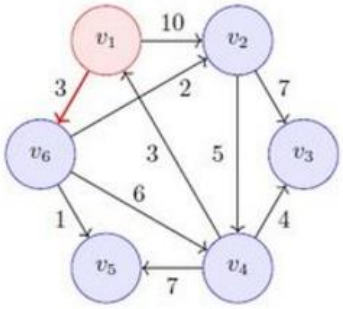




$]=\infty$







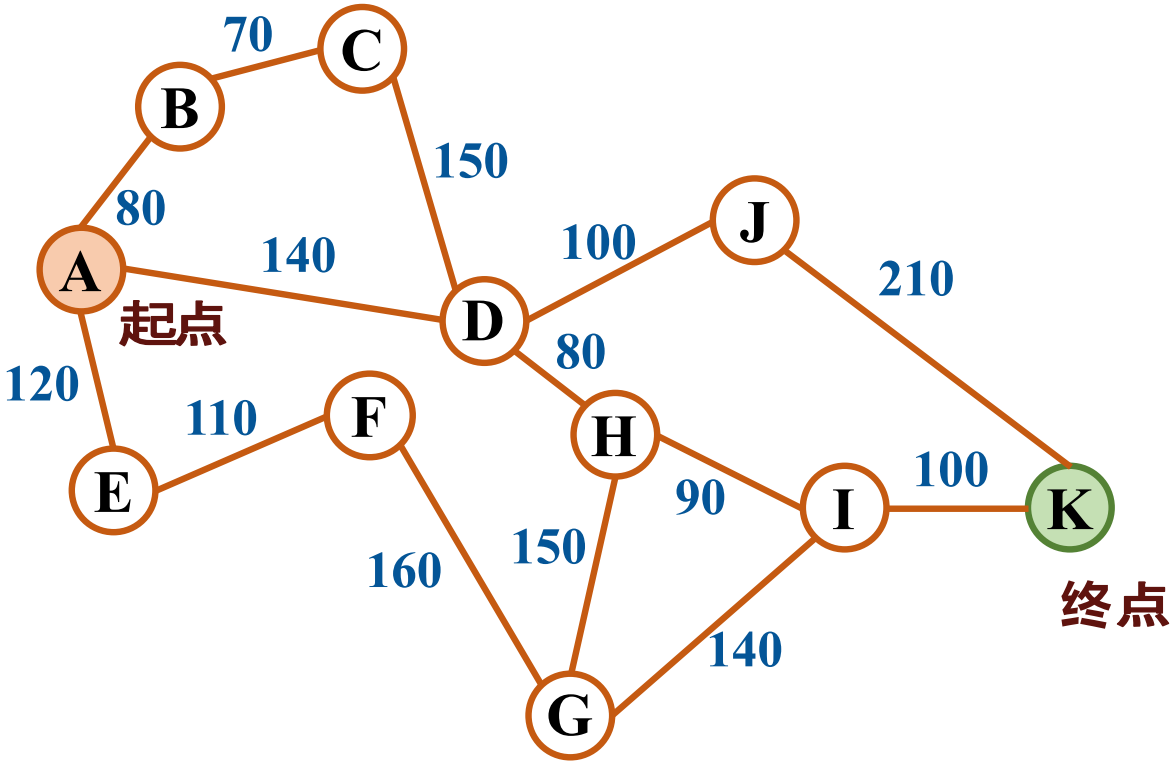
$$S=\{v_1,v_6,v_5,v_2,v_4,v_3\}$$

图　V中的源点s(v_1)到V中其余点的最短路径。

课后作业

2. 导航问题

- 某无人车计划从A点开往K点，已知各节点间的实际距离（见下一页的图示），选取启发函数为各节点到终点的直线距离（见下一页的表格）。请利用A*搜索算法找到解路径（给出完整的搜索过程），请问是否为距离最短的路径，从启发函数的特性上解释原因。



各地点之间的道路距离（可行车的实际距离）

地点	离终点的直线距离
A	370
B	380
C	350
D	250
E	330
F	250
G	240
H	180
I	90
J	170
K	0