人工智能基础与进阶

搜索算法

上海交通大学

目录 content

第一节 搜索与路径规划

第二节 深度优先搜索

第三节 广度优先搜索

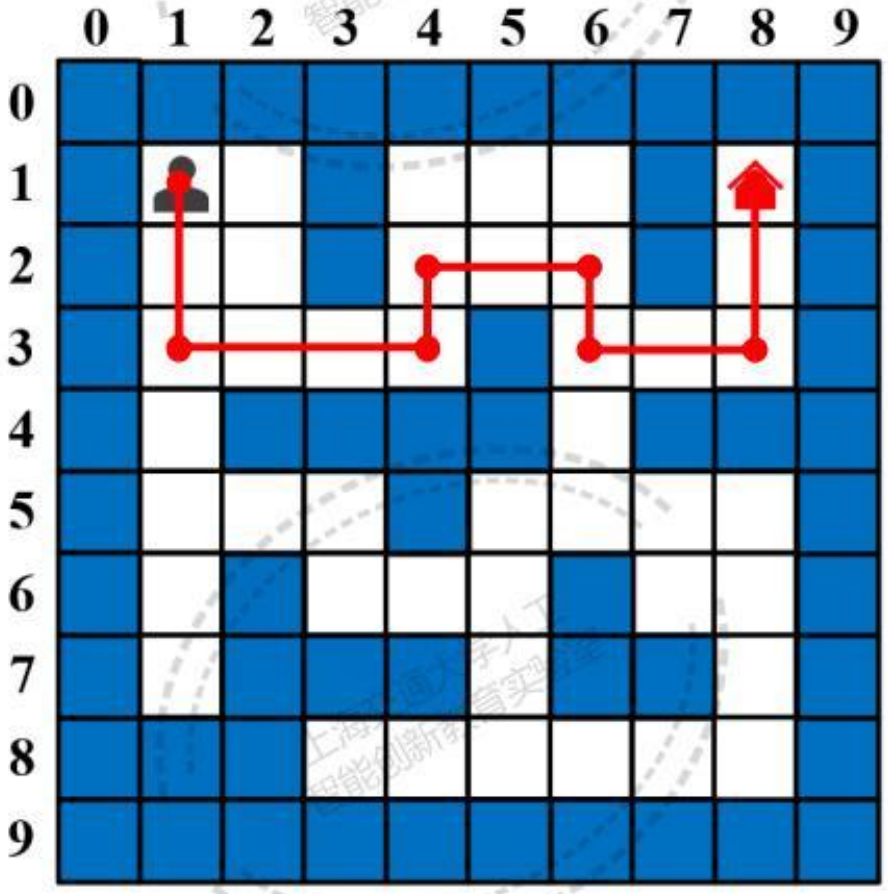
第四节 A*搜索

第五节 A*搜索解决迷宫问题

第一节 搜索与路径规划

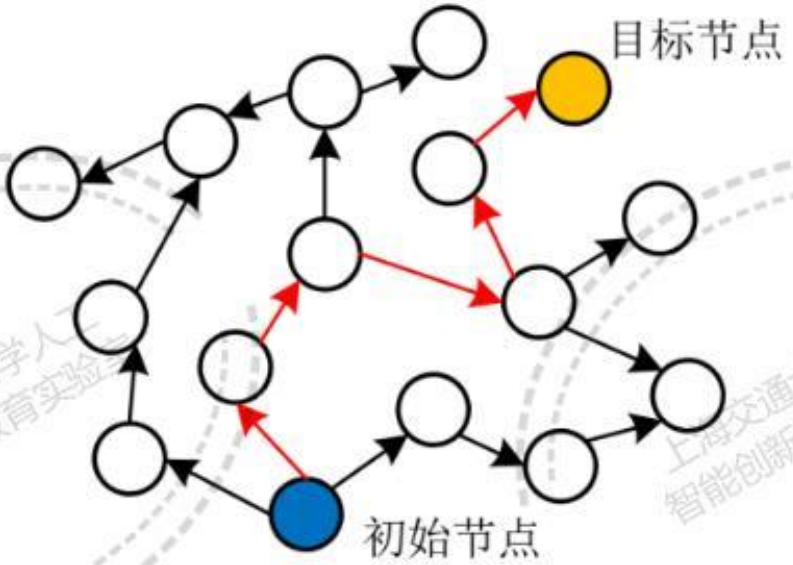
什么是路径规划？

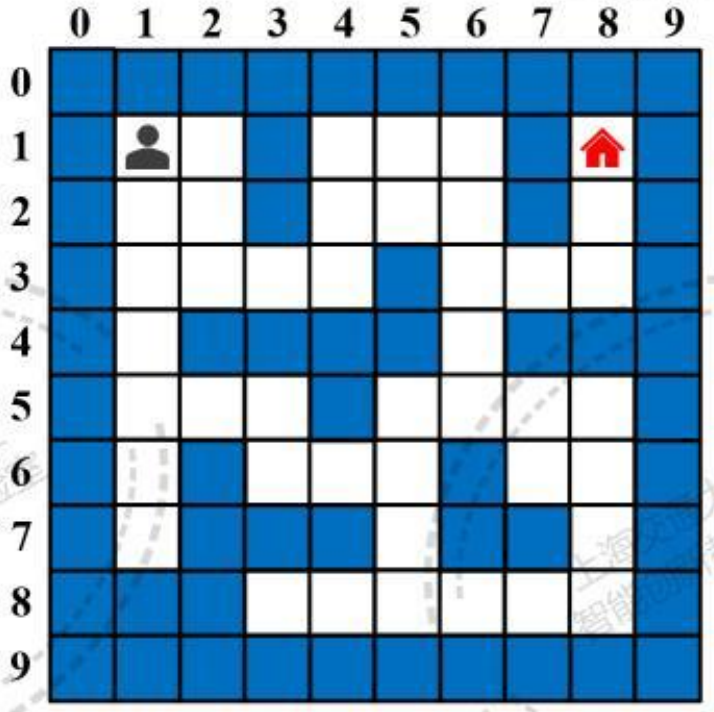
迷宫问题



什么是路径规划？

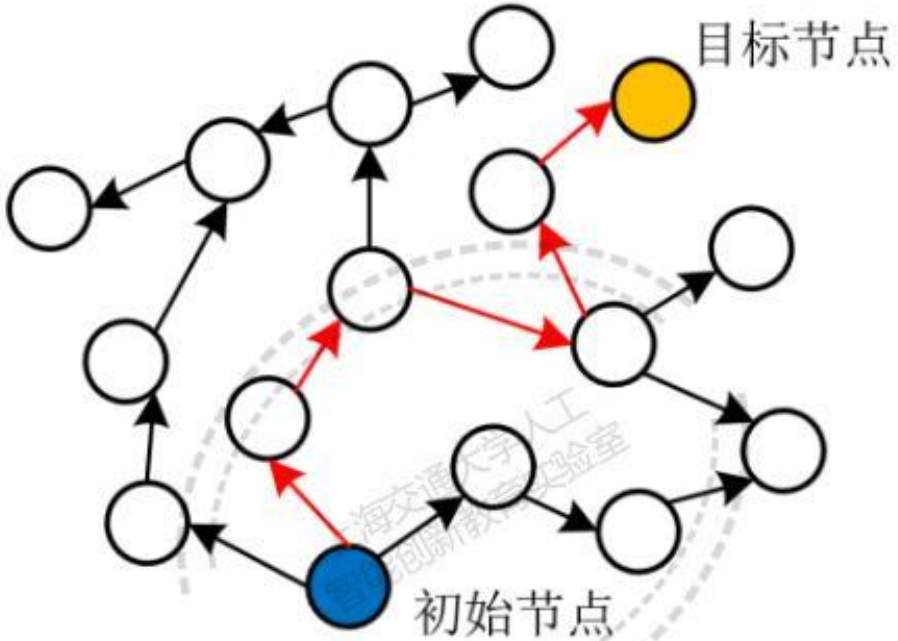
给定任意一个地图，图中的每一个节点表示一个地点，每一条边表示两个地点之间的通路，每个通路具有不同的耗费。路径规划问题就是在一个地图中找出从初始地点到目标地点的一条最佳路径的问题。





什么是搜索算法？

给定任意一个图，图中的每一个节点表示一个状态，每一条边表示状态的转移。搜索算法就是在一个图中找出从初始节点到目标节点的一条路径的算法。



什么是搜索算法？

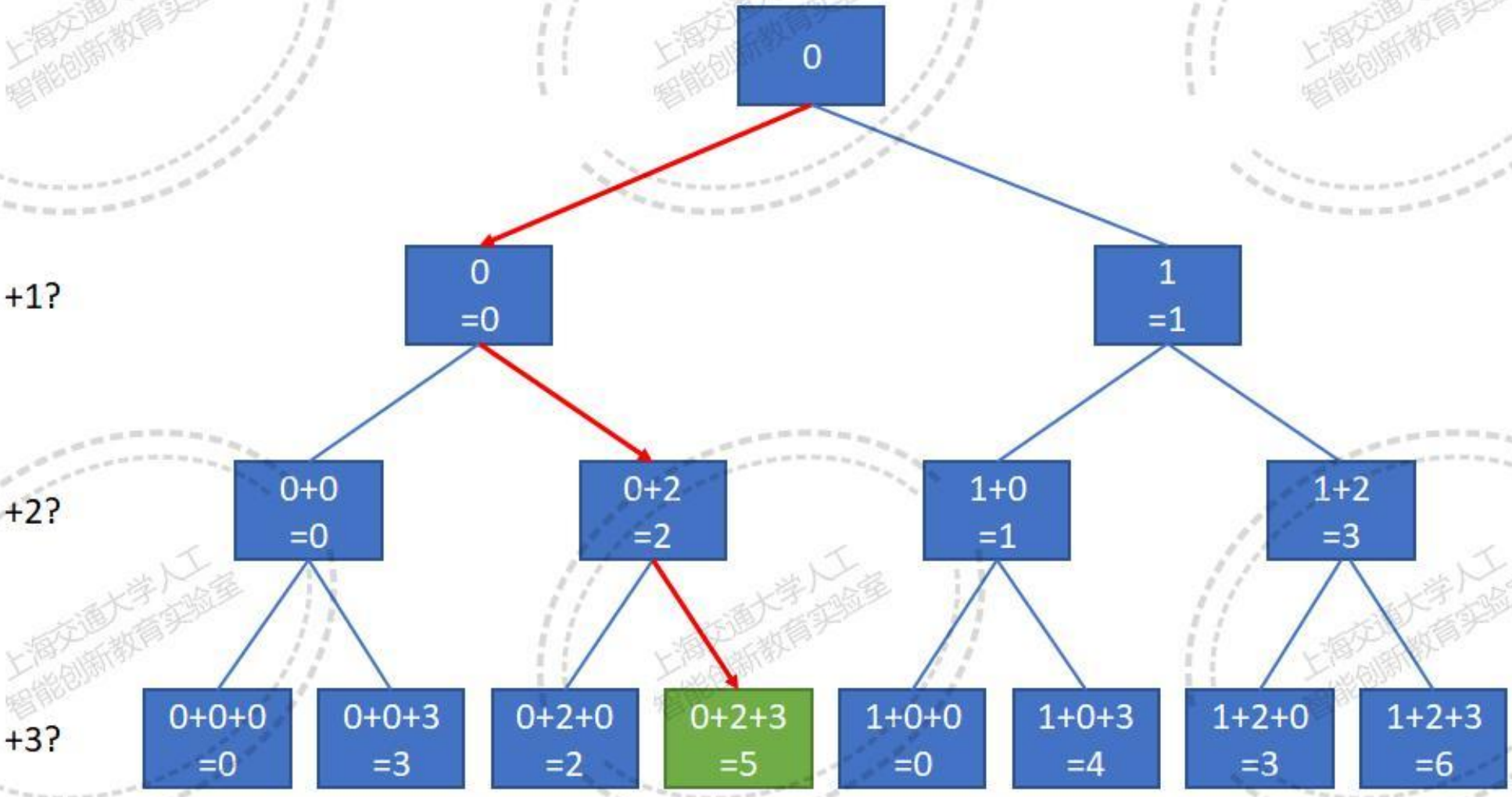
部分和问题

给定几个整数，判断是否可以从中选出若干数，使它们的和恰好为 k ，每个数字只能使用一次。

比如：给定整数1、2、3，判断能否挑出几个数，和正好是5？

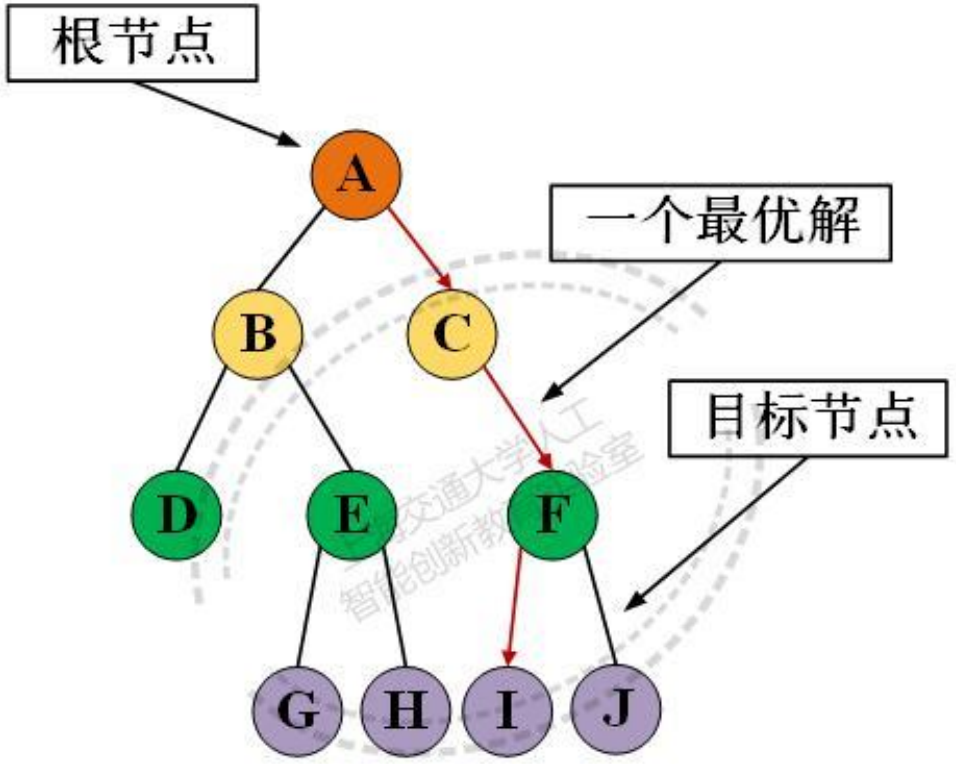
答案：能。2+3=5

什么是搜索算法？

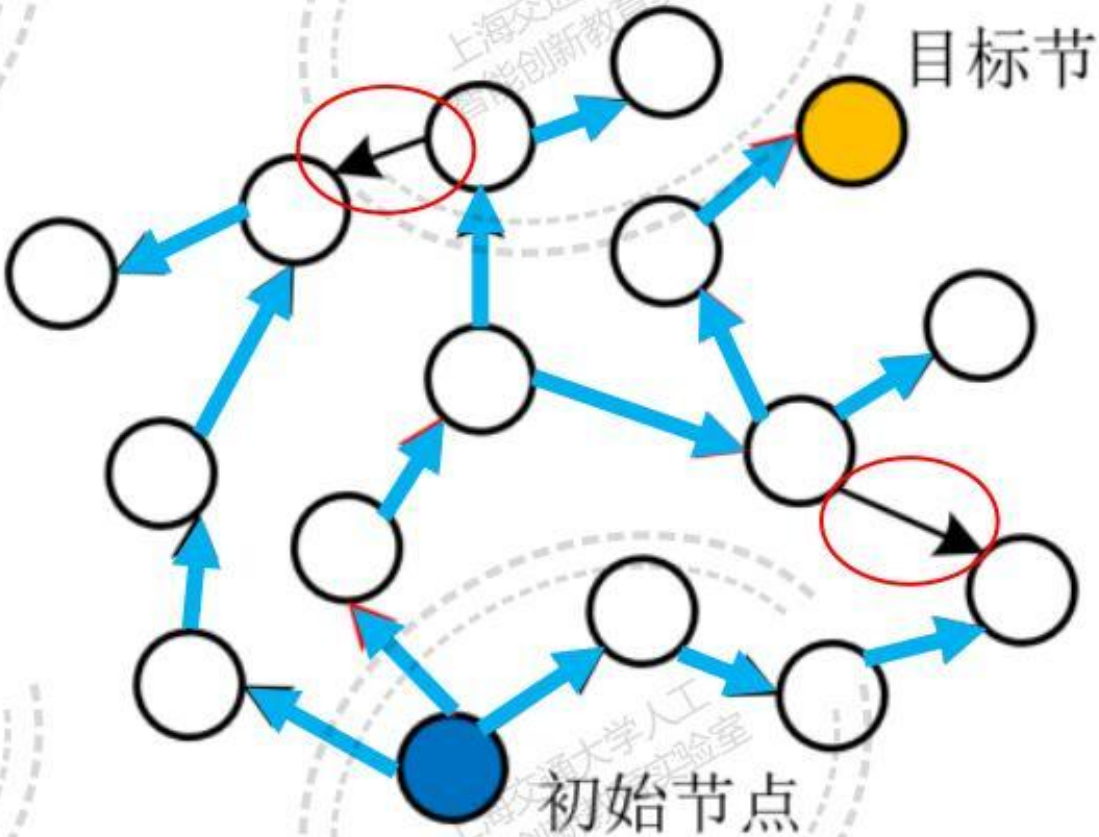

0
0
=0
1
=1
+1?
0+0
=0
0+2
=2
1+0
=1
1+2
=3
+2?
0+0+0
=0
0+0+3
=3
0+2+0
=2
0+2+3
=5
1+0+0
=0
1+0+3
=4
1+2+0
=3
1+2+3
=6
+3?

搜索树

搜索的状态空间呈现为树结构，搜索过程形成的一棵树叫搜索树。初始状态对应着根节点，目标状态对应着某个叶结点。



搜索树



路径规划的搜索算法的分类

搜索算法的本质是在**状态空间中从问题的初始状态搜索通向目标状态的路径**，可以大致分为两类：盲搜索启发式搜索。

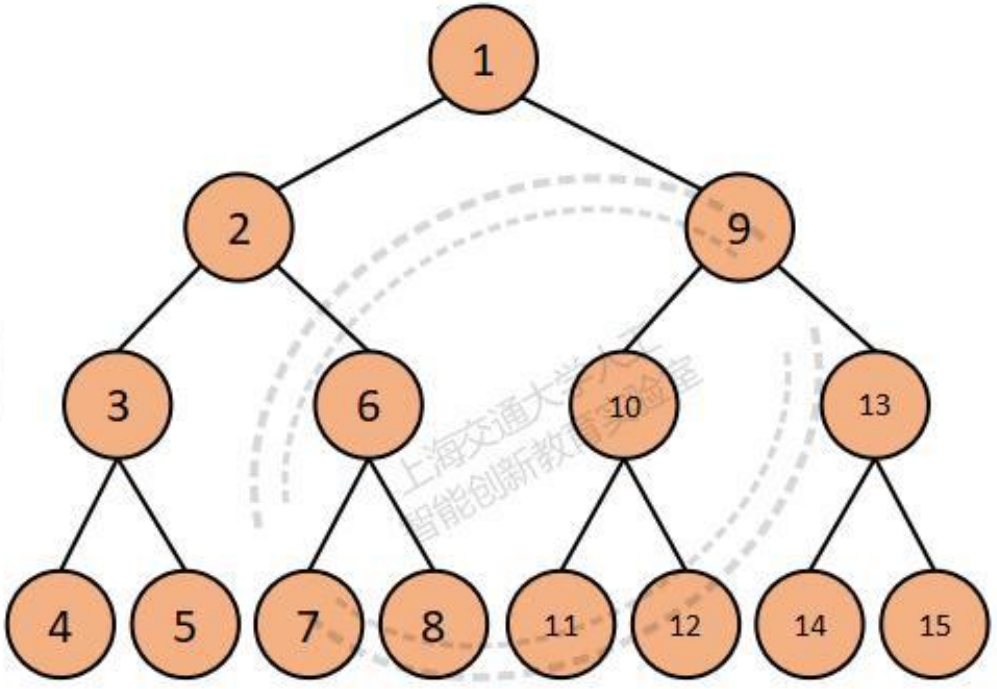
盲目搜索：只能按照**预先规定的搜索策略**进行搜索，没有任何中间信息来改变这些策略，不考虑问题本身的特性，因此具有盲目性，不便于复杂问题的求解。

启发式搜索：是在搜索的过程中**加入与问题有关的信息**，用于指导搜索朝着相对更好的方向前进，加速问题的求解。

第二节　深度优先搜索

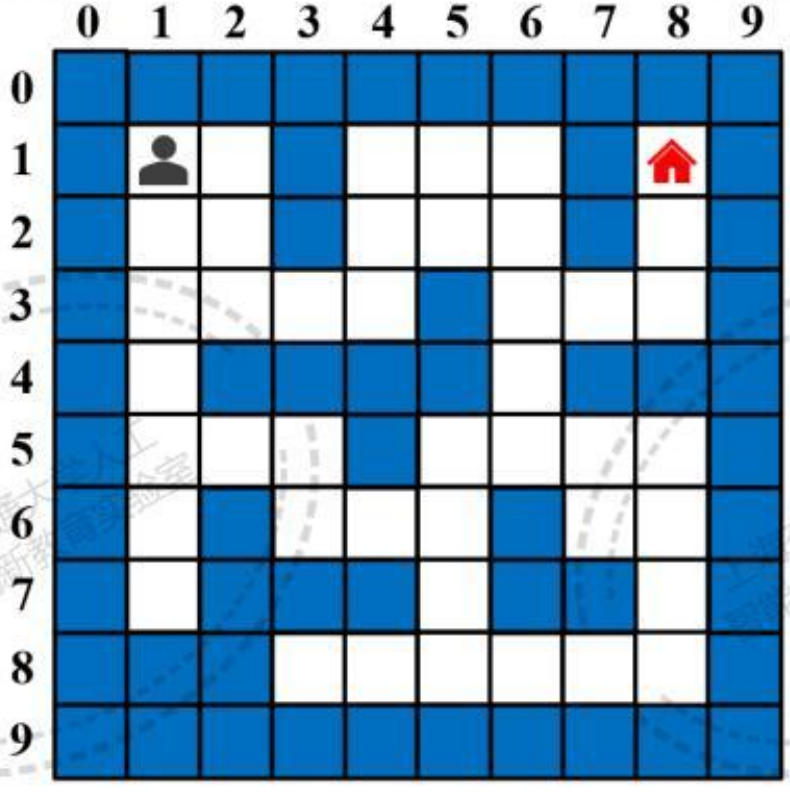
深度优先搜索

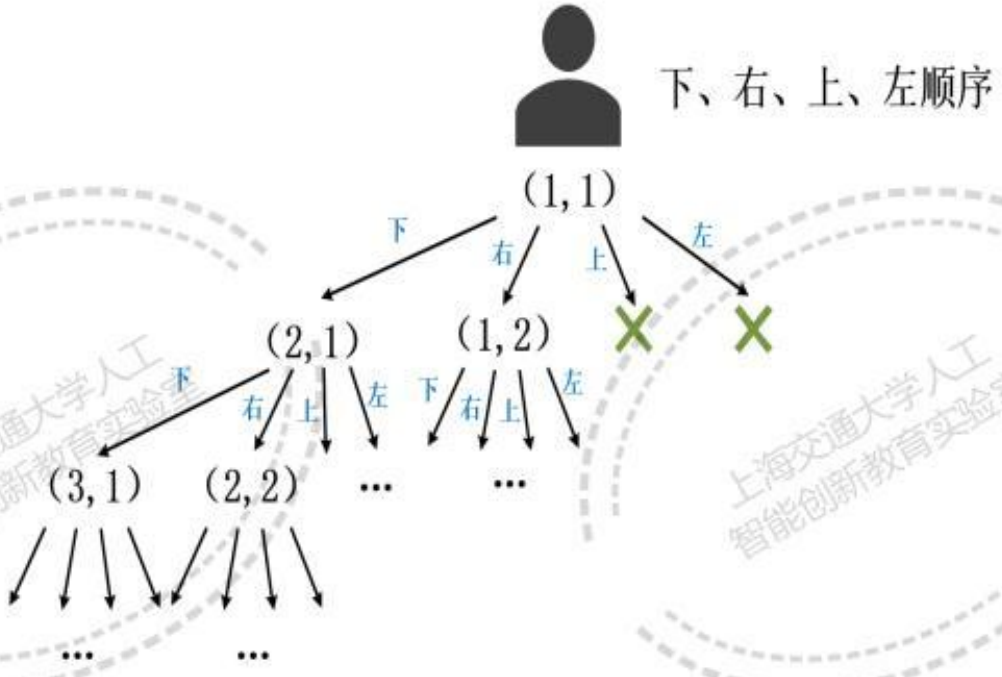
- 深度优先搜索的搜索规则：对每个搜索到的状态节点，如果有未搜索的相邻节点，就一直进行状态转移，直到无法转移为止，然后返回上一个结点。俗话讲：不撞南墙不死心。



深度优先搜索解决迷宫问题

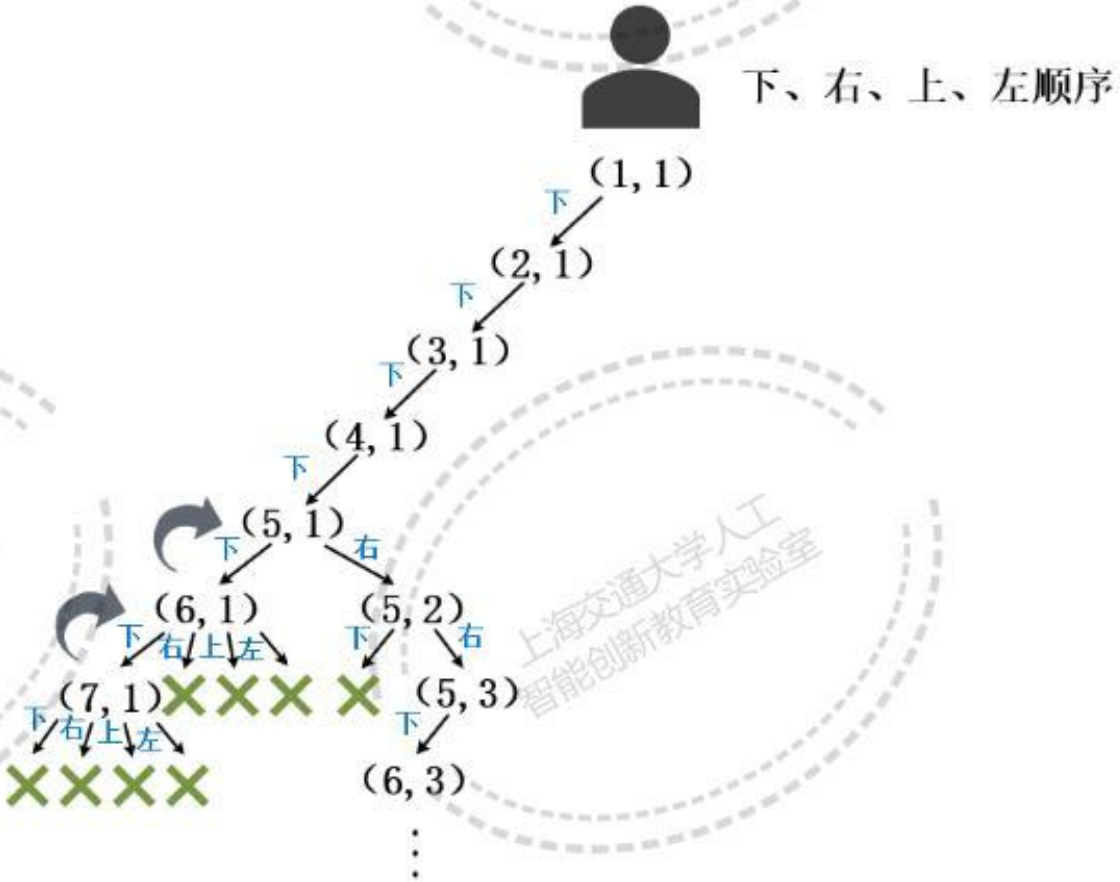
在迷宫问题中，每个节点面临的选择有四种：向下、向右、向上、向左走。也就是说，搜索树的每个节点最多有4个分支。





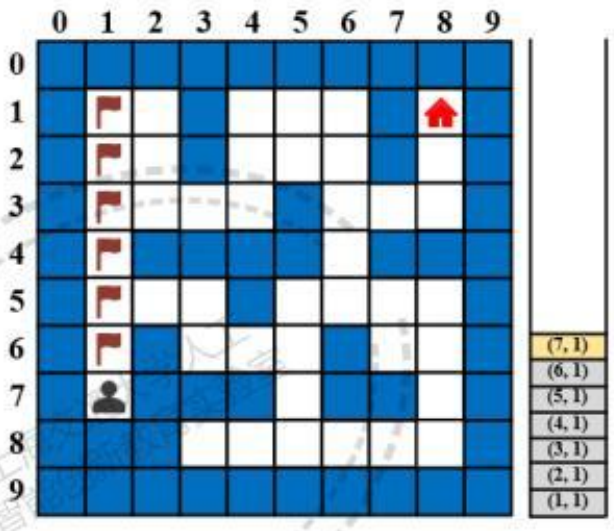
深度优先搜索解决迷宫问题

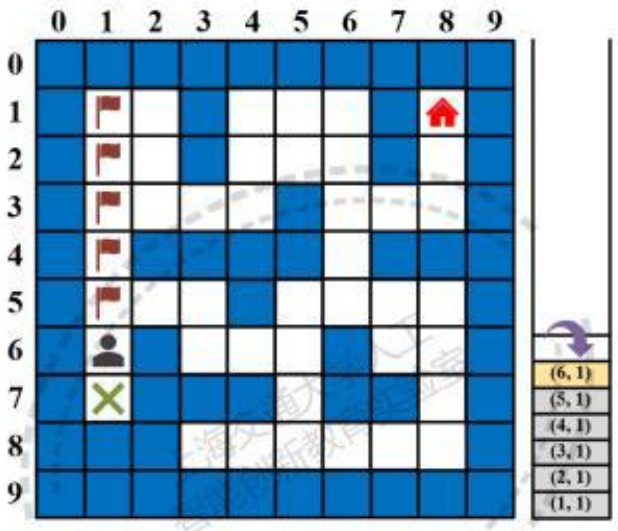
观察DFS在搜索树上的搜索过程，可以从观察出DFS就是不断往深层搜索，直到没有可以搜索的分支为止，再向上返回。

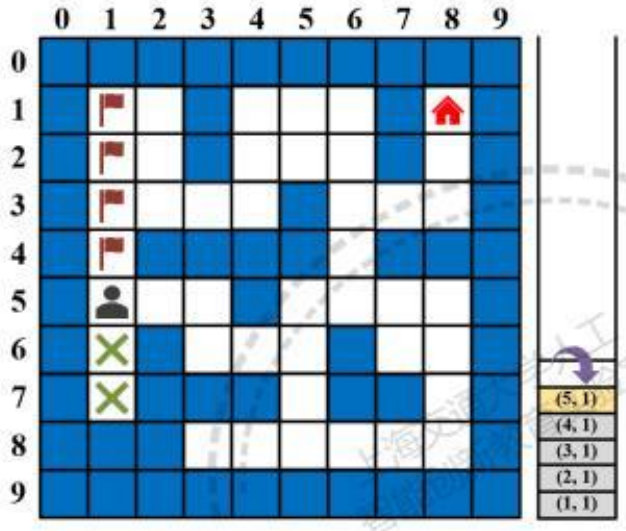


深度优先搜索使用的数据结构

DFS进行搜索时，会使用“栈”结构存放路径的节点，栈就像一个仓库用来存放数据，遵循**先进后出**的规则，放入数据叫**入栈**，拿出数据叫**出栈**。







深度优先搜索使用的数据结构

Python如何实现栈呢?

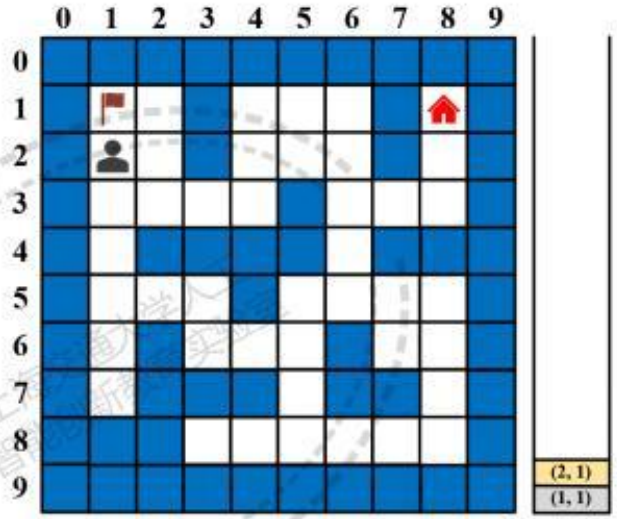
对一个列表list = $[a_1,a_2,a_3,\ldots,a_n]$

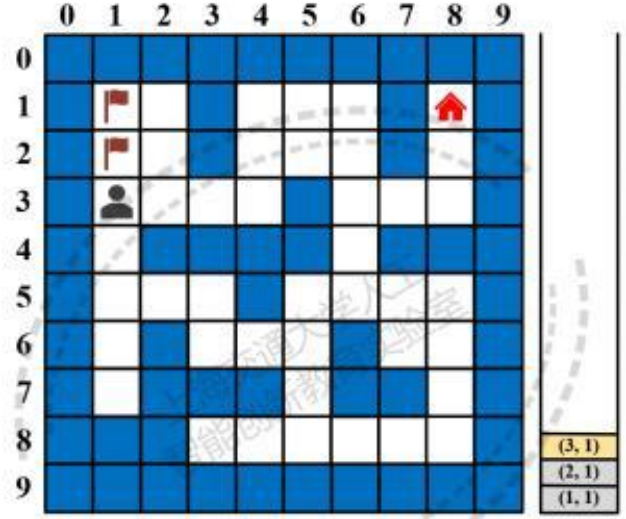
list.append(x)：往列表尾部添加元素x

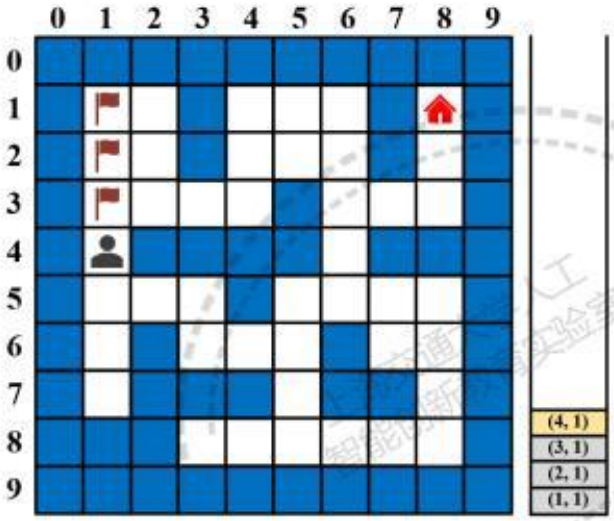
x = list.pop()：删除并返回列表尾部元素x

深度优先搜索解决迷宫问题

在迷宫问题中，DFS的搜索过程就像一个人在迷宫里四处乱走，寻找出口。DFS会在每个节点分别尝试向四个方向移动，只要发现一个方向能走，就立即走，直到四个方向都不能走为止，再返回之前的节点，再看剩余的方向是否能走。

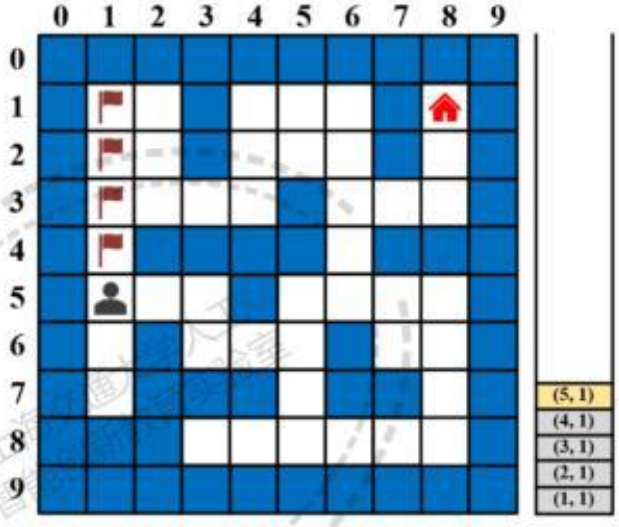


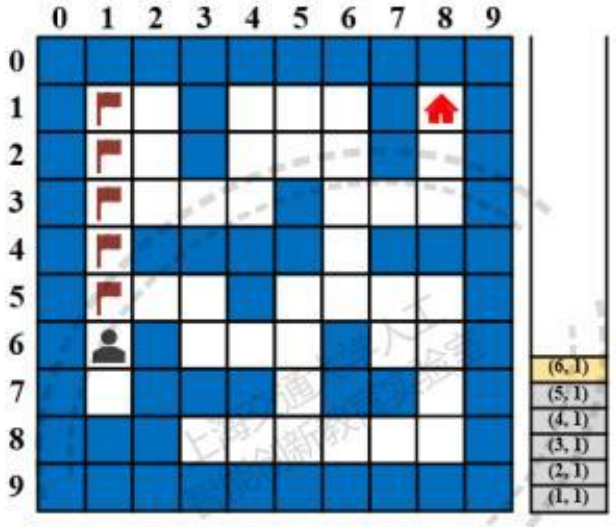


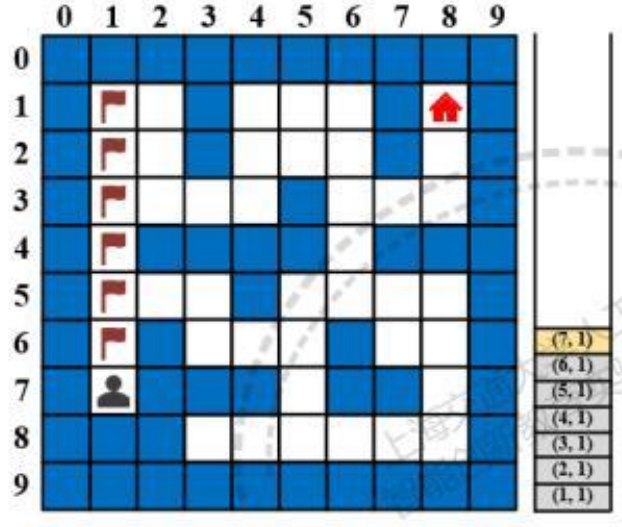


深度优先搜索解决迷宫问题

一开始先一直往下走，棕色旗子代表已经走过的节点，将节点依序入栈。

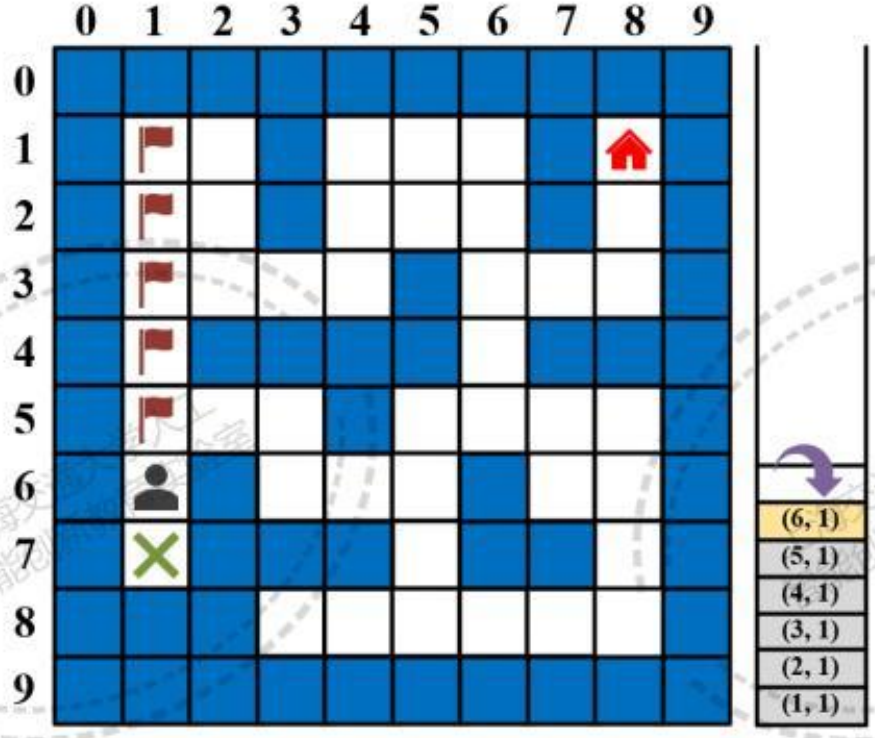


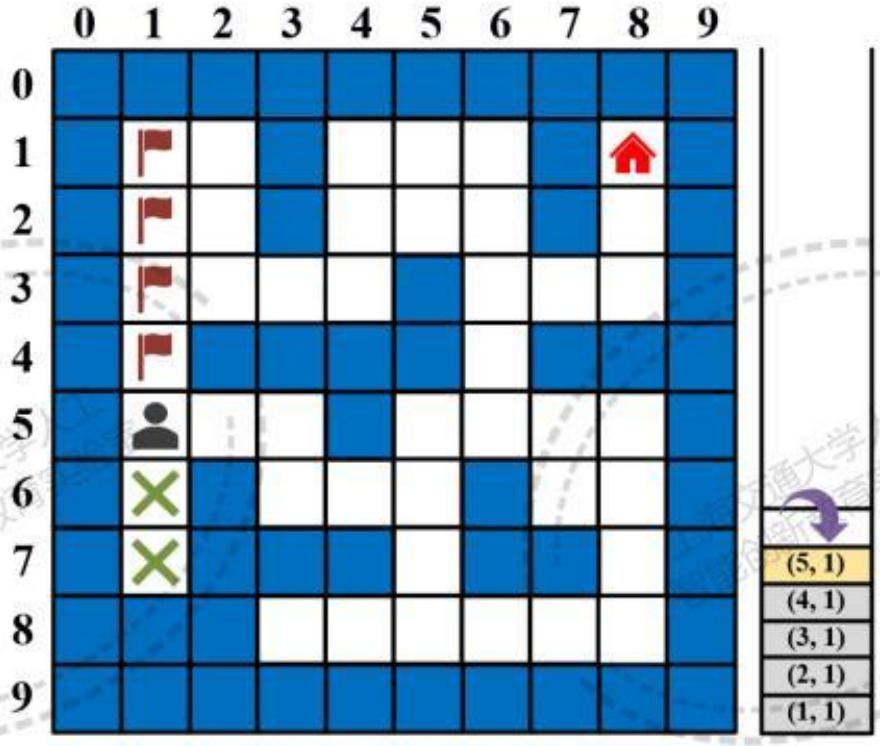




深度优先搜索解决迷宫问题

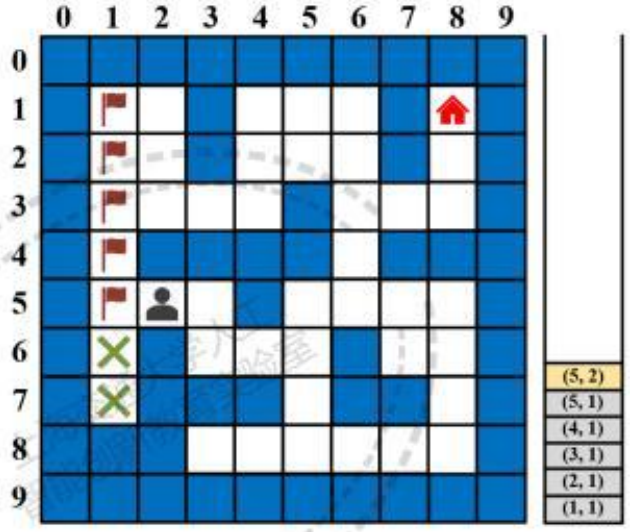
当到达(7, 1)点时，不能继续往下走了，而往右是墙壁，往上是走过的节点，最后往左也是墙壁，因此四个方向无法前进，于是退回上一个节点往其他方向尝试。

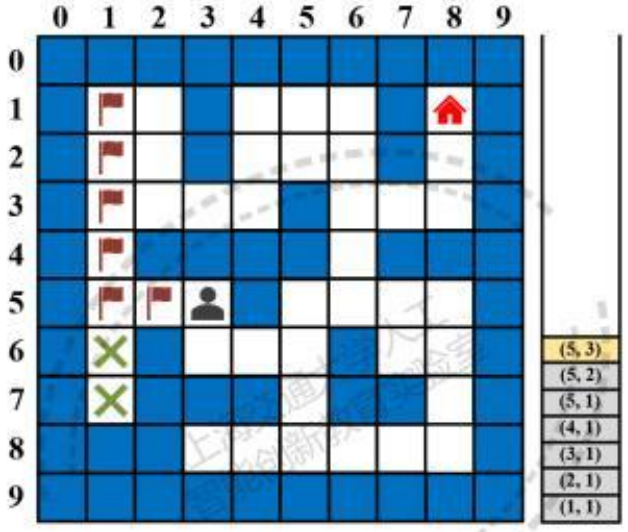


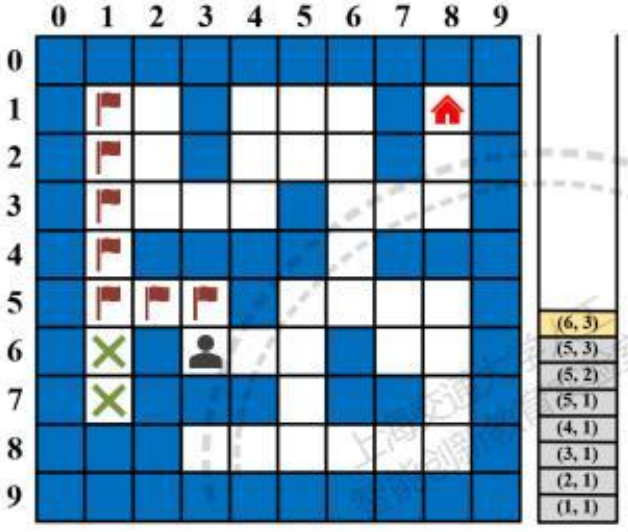


深度优先搜索解决迷宫问题

(5, 1) 点往下已经走过，改从右边前进到(5, 2) 点，每到一个新节点都要重新遍历方向顺序，此时往下是墙壁无法前进则继续往右到(5, 3) 点。接着能往下到(6, 3) 点，同样的步骤反复进行直到终点入栈为止。

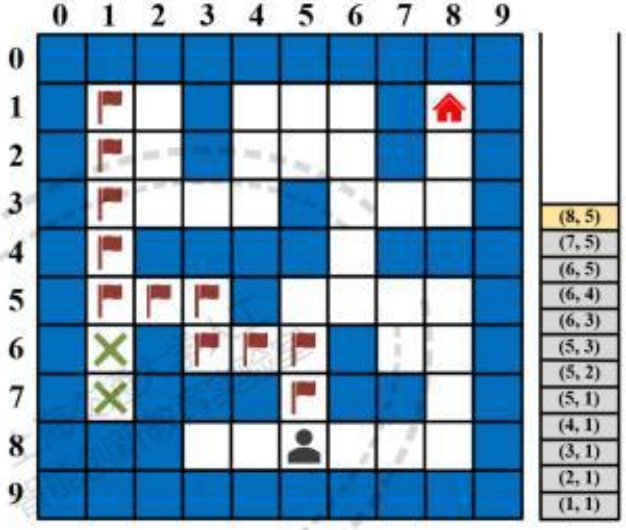


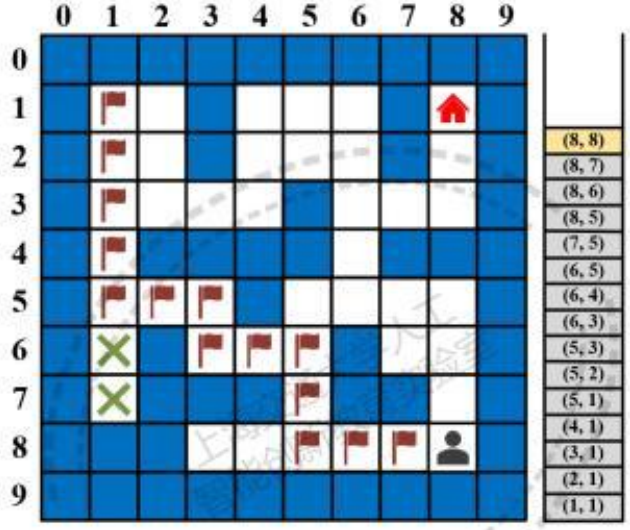


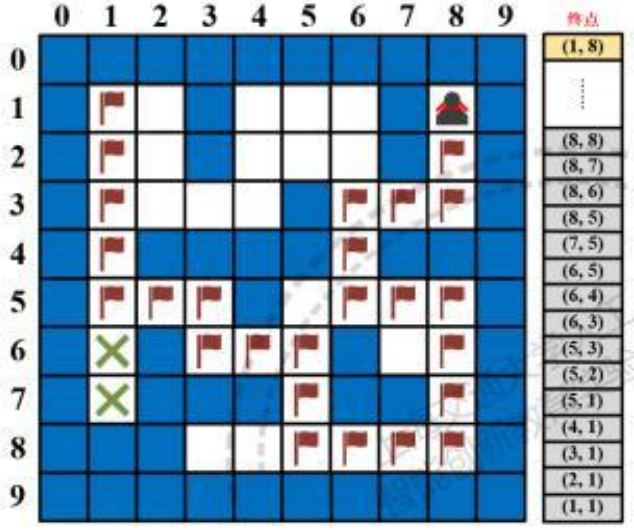


深度优先搜索解决迷宫问题

最后，终点入栈则表示算法结束并且找到路径，栈存放的数据就是搜索的路径从起始点开始跟着旗子走到终点。

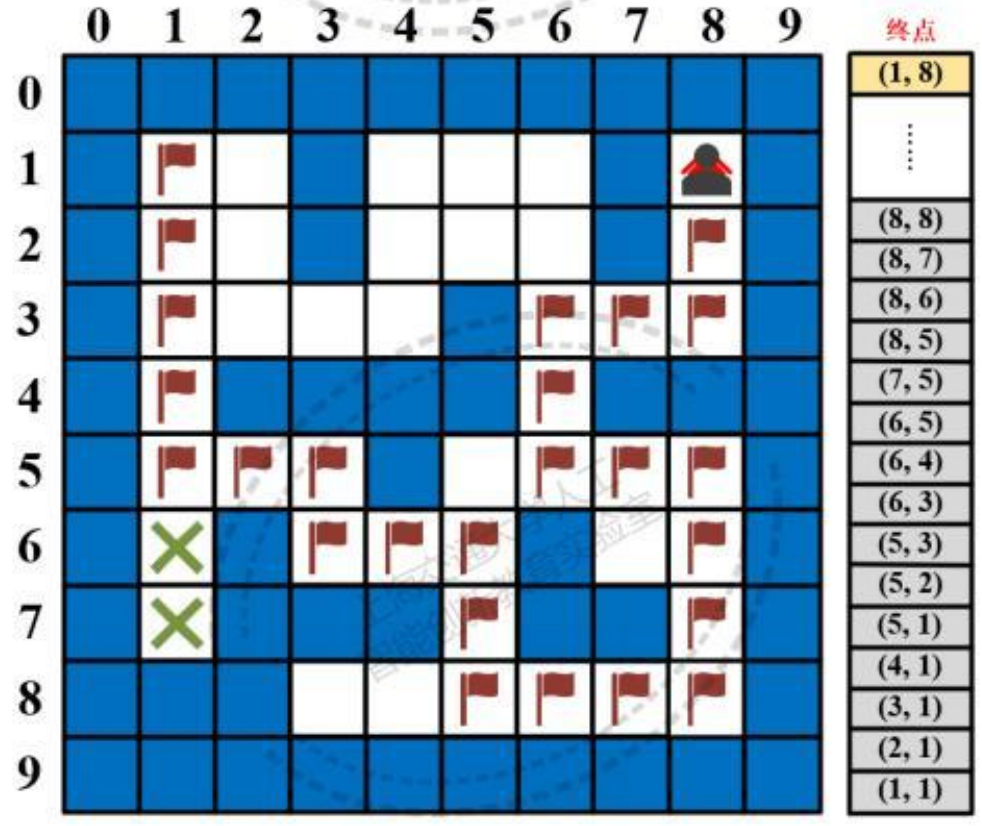






深度优先搜索的缺陷

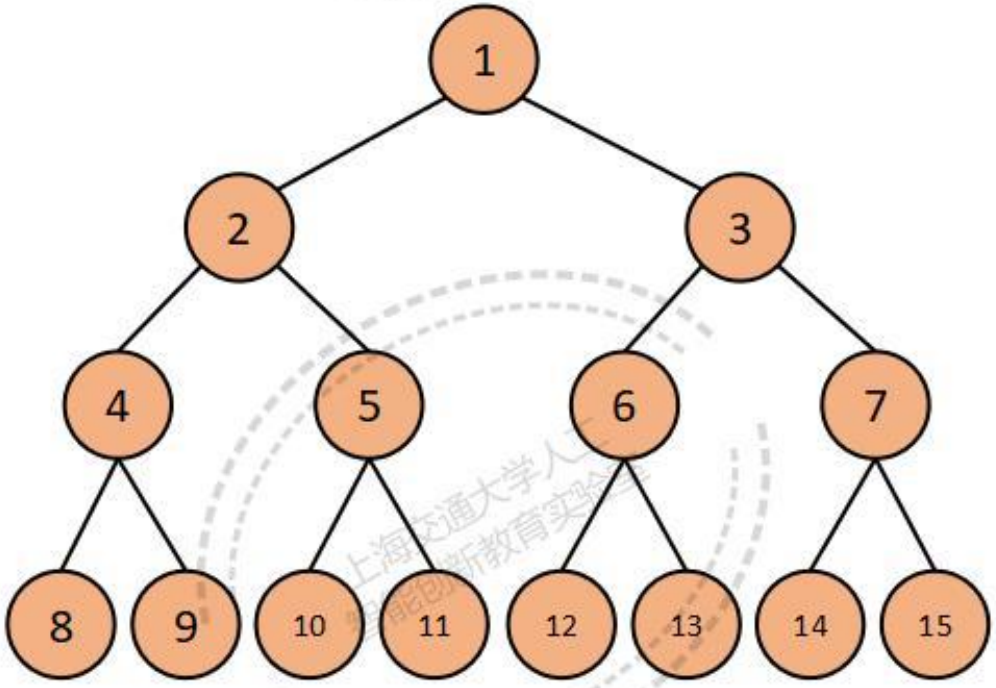
实际上，深度优先搜索并不适合求解路径规划问题，因为它有一个致命的缺陷：无法找到最短路径。



第三节　广度优先搜索

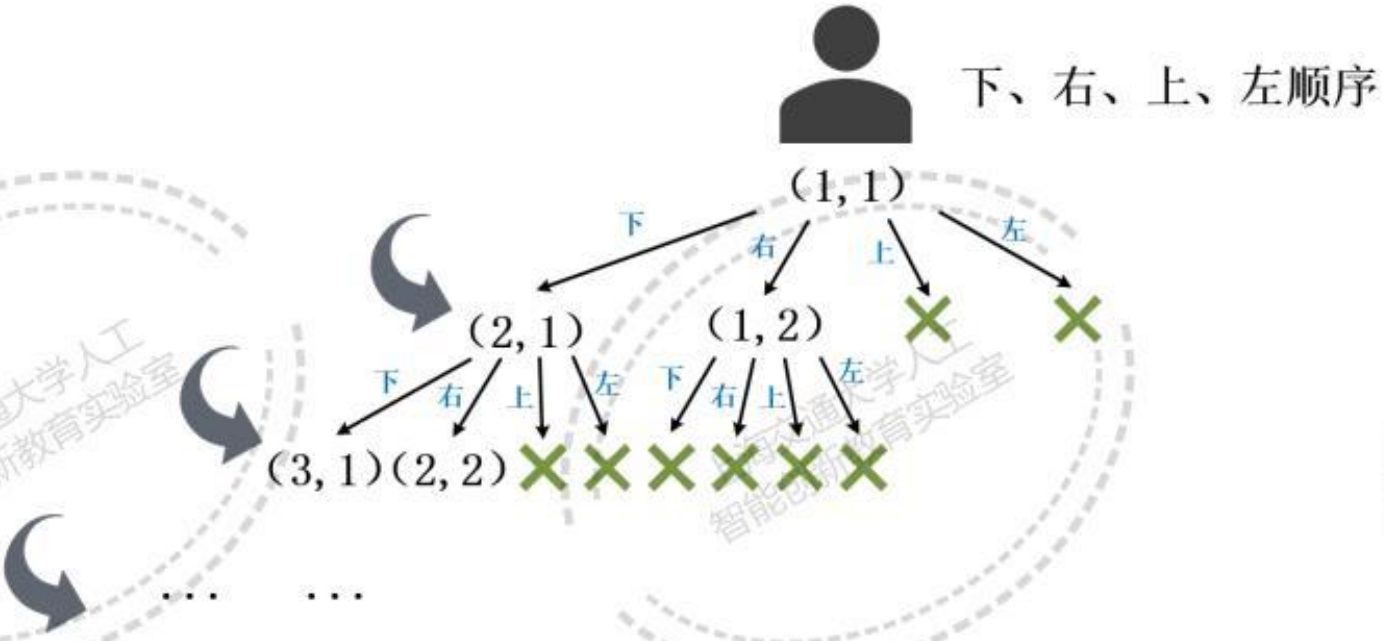
广度优先搜索

广度优先搜索的搜索规则：按照到起始节点的状态转移次数（步数），依次搜索所有的节点，也就是一层一层地遍历搜索树。



广度优先搜索解决迷宫问题

同样，对迷宫问题来说，观察BFS在搜索树上的搜索顺序，就会发现BFS是一层一层搜索的。



广度优先搜索使用的数据结构

BFS使用队列来存储搜索的节点。队列和栈相反，遵循**先进先出**的原则，从队尾加入数据称为“入队”，从队头删除数据称为“出队”，先加入的数据会优先被删除，因此称为先进先出。

广度优先搜索使用的数据结构

Python如何实现队列呢?

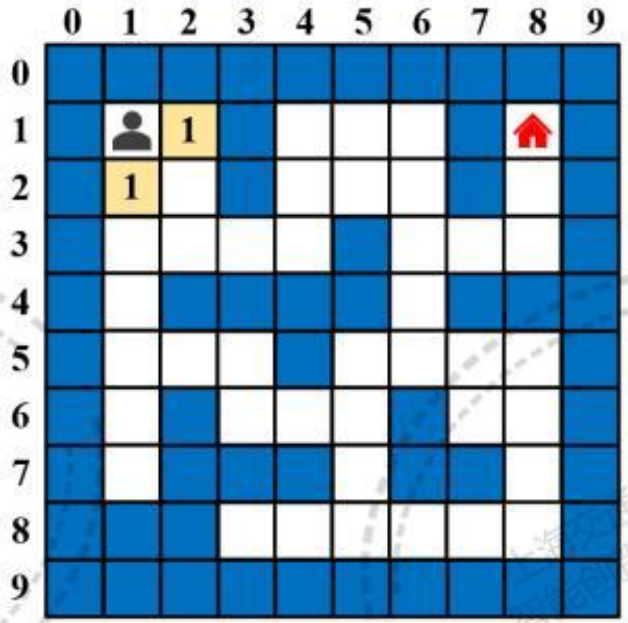
对一个列表list = $[a_1,a_2,a_3,\ldots,a_n]$

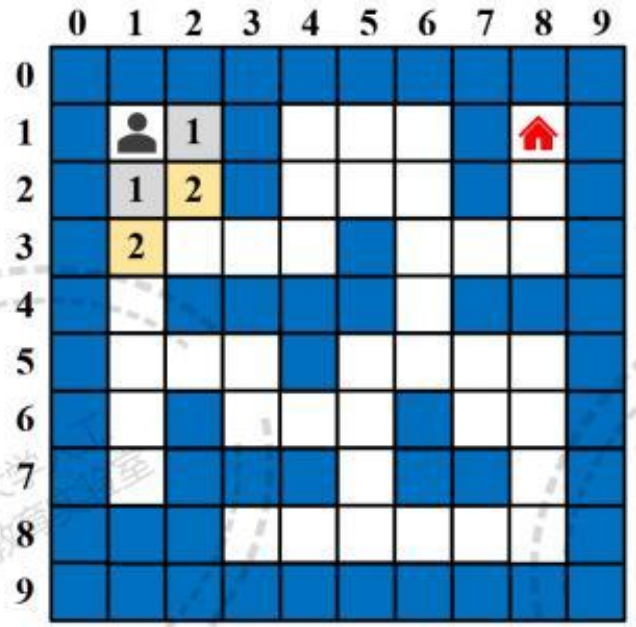
list.insert(0, x)：往列表头部插入元素x

x = list.pop()：删除并返回列表尾部元素x

广度优先搜索解决迷宫问题

开始时，把起始点放入队列。然后每次从队列中取出一个节点，检查它相邻的所有未搜索过的点，如果不是目标点，就加入队列。以此类推。



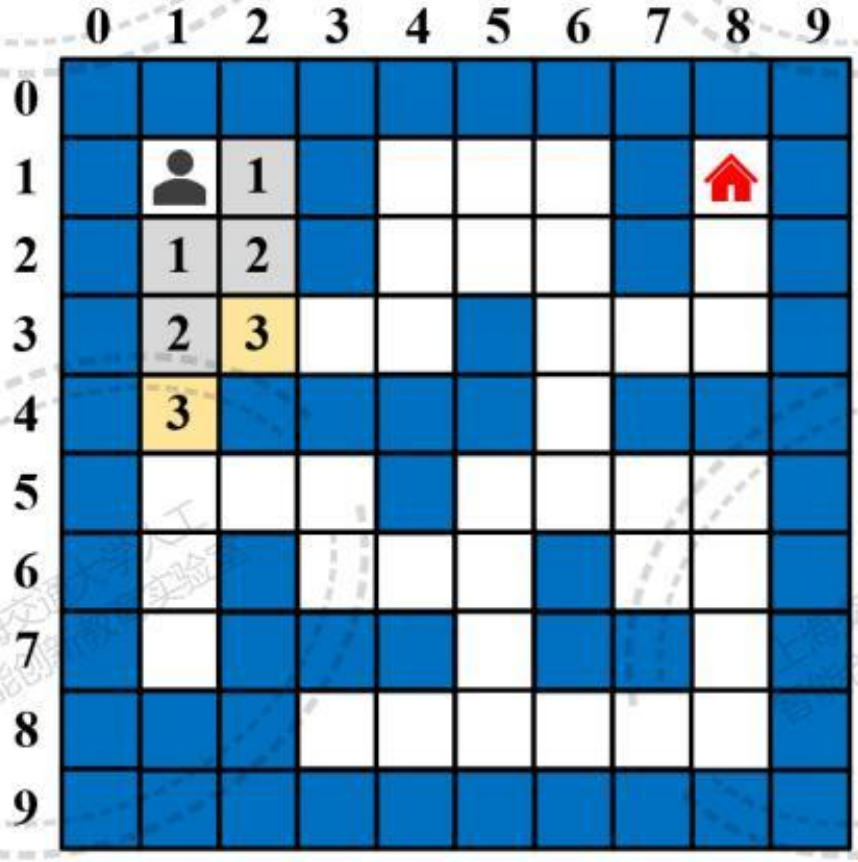


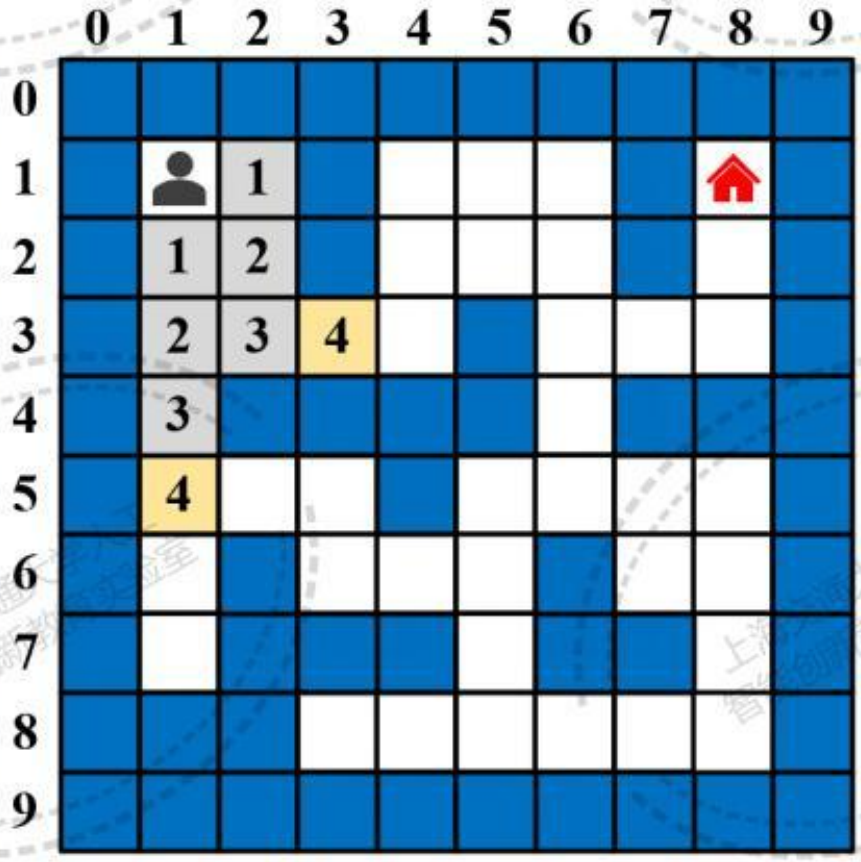
(1,1) ← | (2,1) | (1,2) | ←

(2,1) ← | (1,2) | (3,1) | (2,2) | ←

广度优先搜索解决迷宫问题

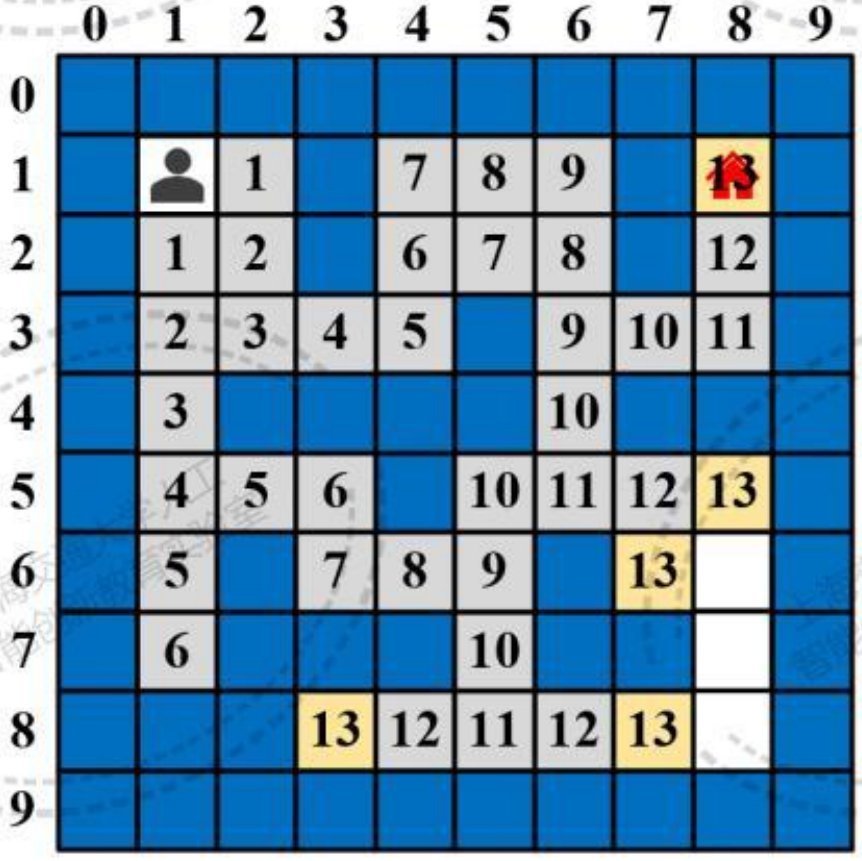
反复相同步骤直到终点被访问为止。广度优先搜索的过程，就像流动扩散的一滩水，逐渐蔓延到整个地图，直到找到终点。

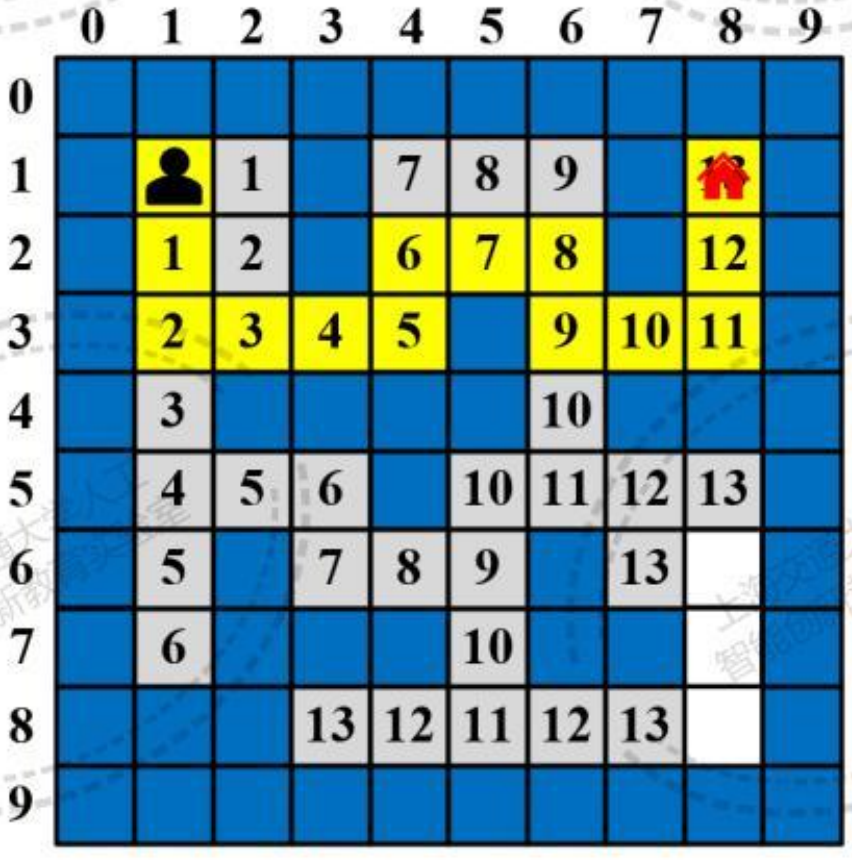




广度优先搜索解决迷宫问题

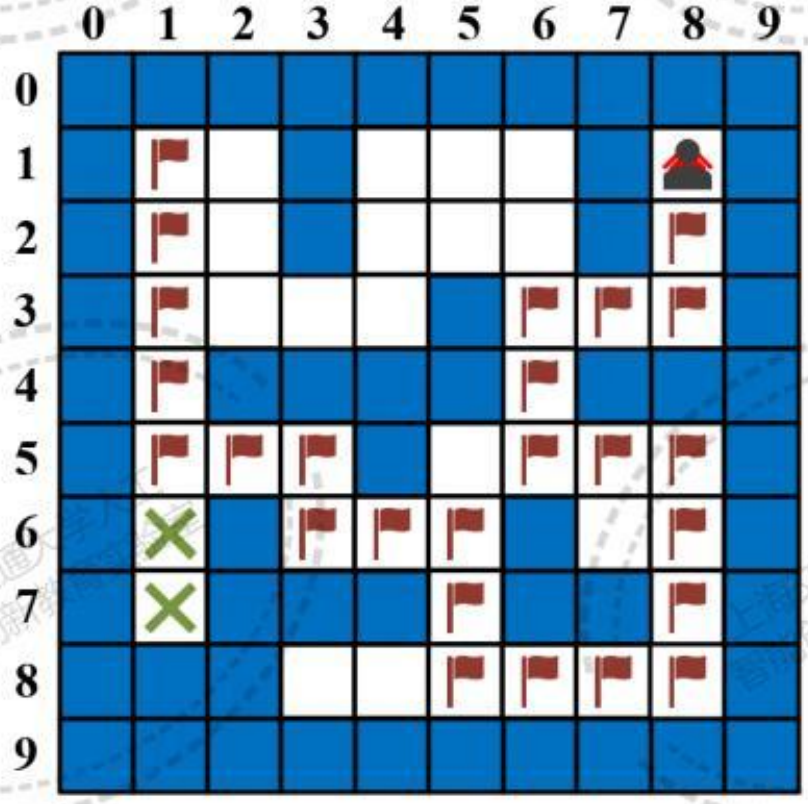
第十三步时，终点加入队列则算法停止，从终点依序返回父节点就能找到路径。

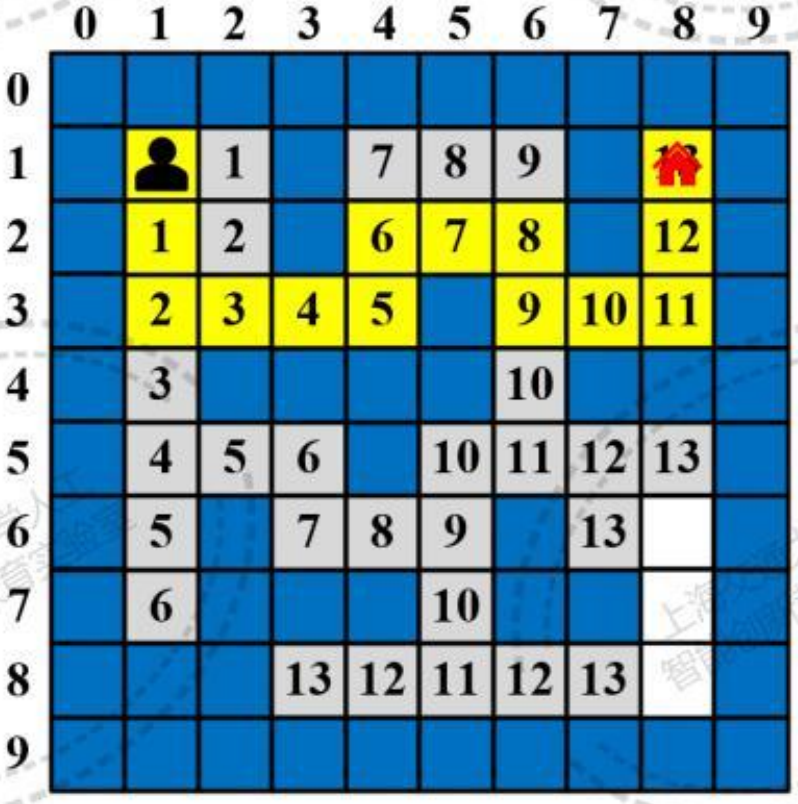




广度优先搜索解决迷宫问题

很显然与DFS找到的路径不相同，BFS找到的是最优路径，而DFS则是绕了一大圈的路径。



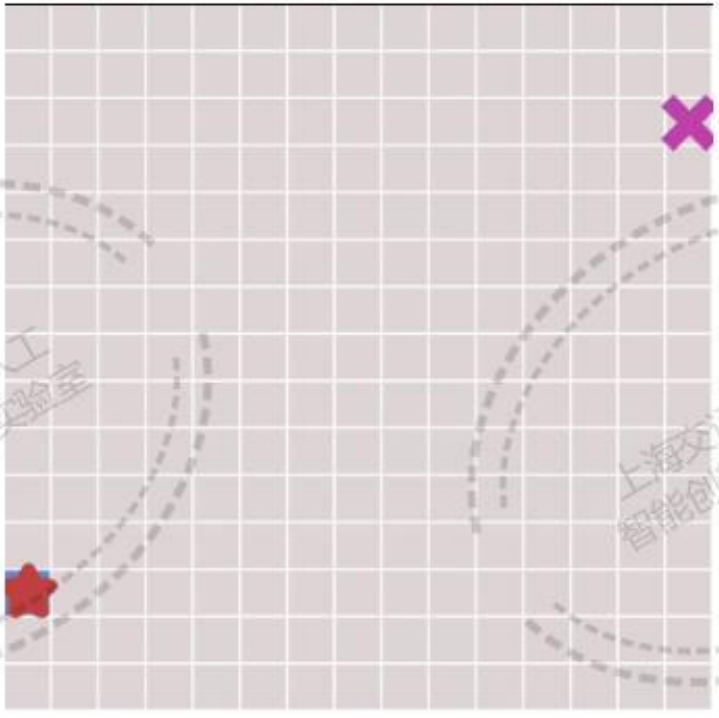


第四节 A* 搜索

盲目搜索的问题

对于路径规划问题，盲目搜索的直接问题是：不管目标点在哪儿，只顾埋头找路。搜索没有方向性，时间耗费太大。以广度优先搜索为例，搜索边界会向四面八方延申：

广度优先搜索

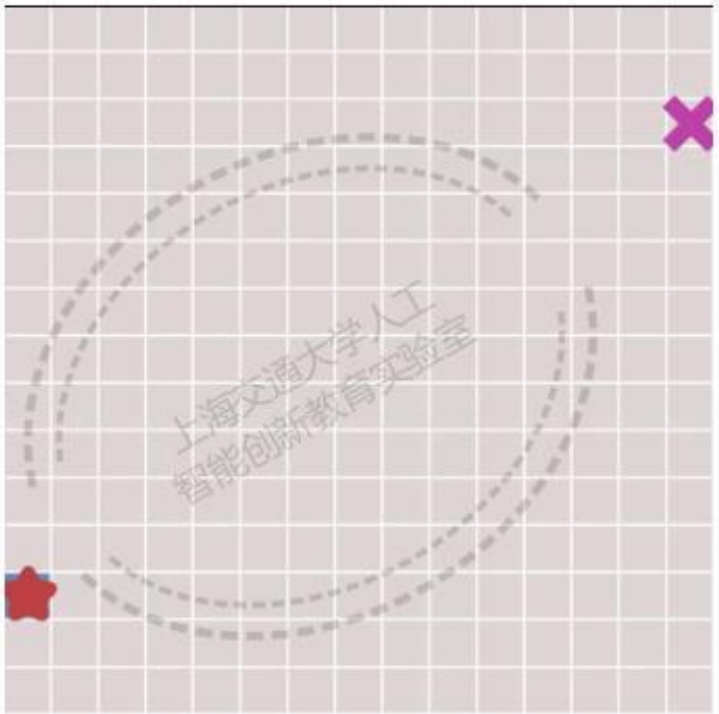


从另一个角度来看，广度优先搜索的机制是：在已搜索的区域边界上，选取距离出发点最近的点作为下一个搜索点。

启发式搜索——贪心搜索

启发式搜索中，引入启发函数，估计当前点距离目标点的距离。那么一个最直接的想法是：对于搜索边界上的所有相邻点，每次都选取当前距离目标点估计距离最近的点，作为下一个搜索点。这个搜索方法叫做贪心搜索。

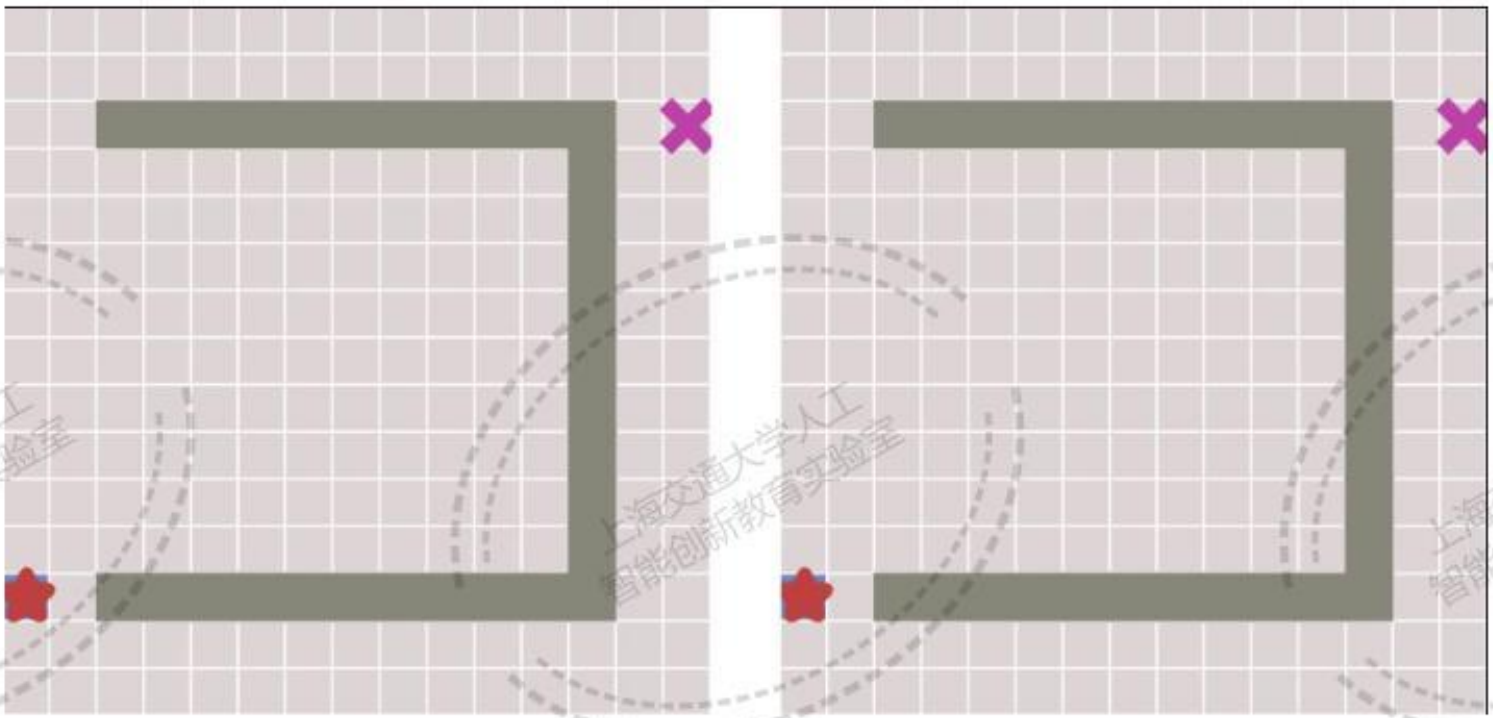
贪心搜索



启发式搜索——贪心搜索

贪心搜索的速度比广度优先搜索快得多，但是，它也丢失了一个优良的性质：它无法找到最短路径。对比广度优先搜索和贪心搜索：

广度优先搜索　　贪心搜索



A* 搜索

- 贪心算法无法找到最优路径的直接原因就是，它并不考虑当前搜索点到起始点的距离。所以，当我们同时考虑当前搜索点到起始点的距离和到目标点的估计距离时，就产生了A*搜索算法。
- A*搜索中，对每个点计算一个代价，

  公式表示为：$f(n) = g(n) + h(n)$
- 其中，$f(n)$是从初始状态经由状态n到目标状态的总代价；

  $g(n)$是在状态空间中从初始状态到状态n的转移代价；

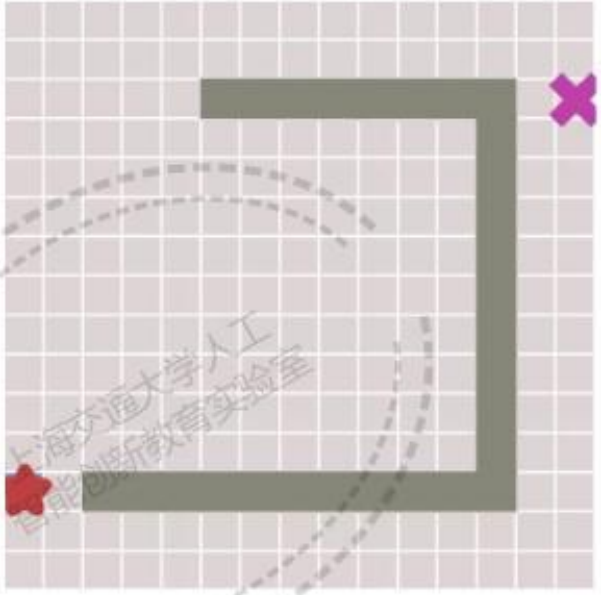
  $h(n)$是从状态n到目标状态的估计代价。

（对于路径搜索问题，状态就是图中的节点，代价就是距离）
- 对于搜索边界上的所有相邻点，每次都选取总代价最小的点作为下一个搜索点。

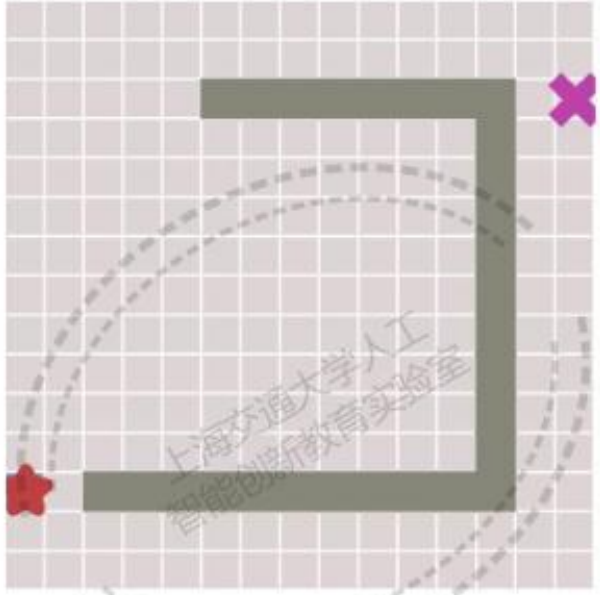
A* 搜索

A*算法既考虑了距离目标点的距离，防止了对错误方向的搜索，加快了搜索的过程；又考虑了到起始点的路径长度，以找到更短的路径。

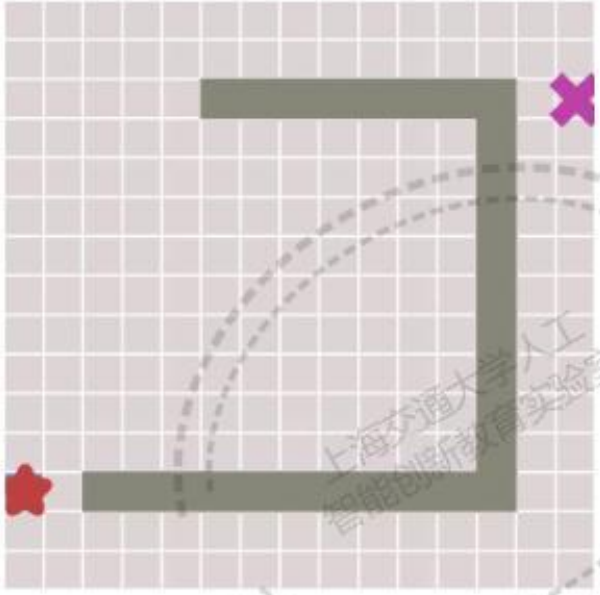
广度优先搜索



贪心搜索



A*搜索



第五节 A*搜索解决迷宫问题

A*搜索解决迷宫问题

用A*算法解决迷宫问题的思路就是**从起始点开始，每一步都选择代价最小的方块前进，直到抵达终点时停止搜索。**

算法核心公式就是每一步都选择最小的F值：F = G + H，其中：

F 为方格的总代价；

G 为起始点到当前方格的实际代价；

H 为当前方格到终点的估计代价。

基于搜索算法的迷宫案例求解-A*算法

G值是怎么计算的？

假设现在车子在某一方格，邻近有8个方格可走，作如下假设：

✓ 当往上、下、左、右这4个方向走一格时，实际代价为10；

✓ 当往左上、左下、右上、右下这4个方向走一格时，实际代价为14，即走对角线的实际代价为走直线的1.4倍。

基本公式：G = 当前格G值 + 移动代价

A*搜索解决迷宫问题

H值是如何预估出来的？

很显然，在只知道当前点和终点，不知道这两者的路径情况下，我们无法精确地确定距离大小，所以只能进行估计。有多种方式可以估计H值，如**曼哈顿距离**、欧式距离等，最常用的方法是使用曼哈顿距离进行估计：

H = 当前方块到终点的水平距离 + 当前方块到终点的垂直距离

A*搜索解决迷宫问题

A*算法的算法步骤：

要定义两个列表：开放列表存放当前可以前进的方格，封闭列表存放所有搜索过的方格。

1. 把选中的格加进封闭列表内，把这个格称为当前格（开始时以起点作为当前格）。

2. 寻找当前格可通过的相邻方格（排除在封闭列表内和障碍物的方格）。如果没有在开放列表内，则计算总代价F并设当前方格为父方格，再加进开放列表内，然后检查是否为终点，如果是，算法结束；如果已经在开放列表内，则计算通过当前格到此格的新G值是否比原来的G值较低，如果是，则更改父方格并重新计算F值。

A*搜索解决迷宫问题

A*算法的算法步骤：

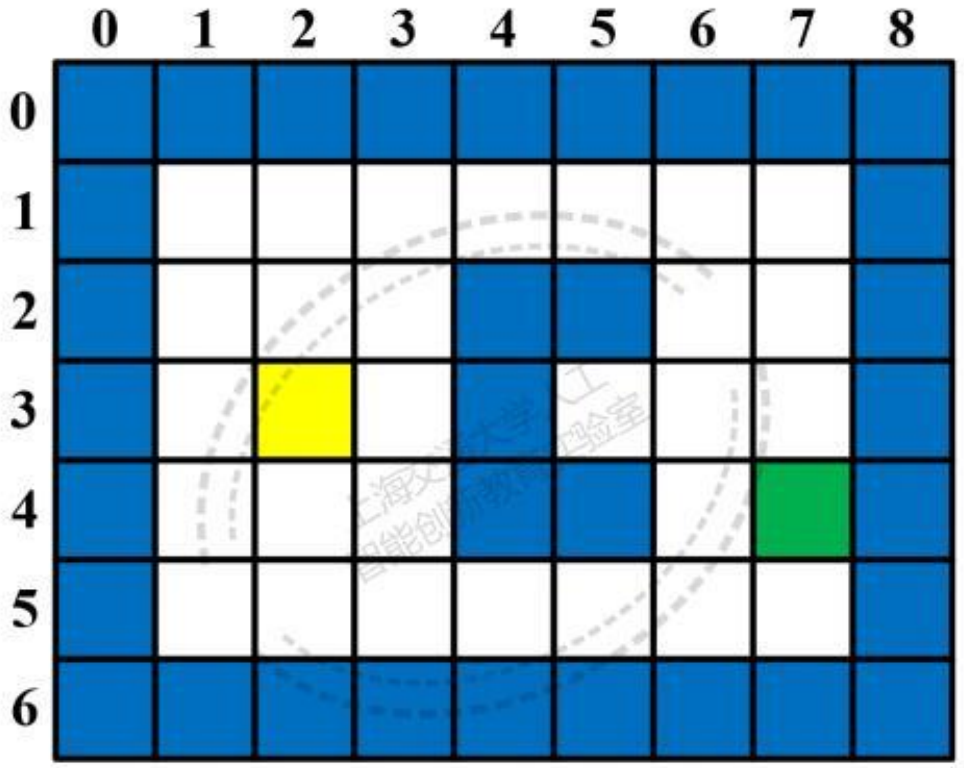
4. 判断开放列表是否为空，如果是则找不到路径，算法结束。否则继续步骤5。

5. 遍历所有在开放列表内的方格，选择总代价F最小的方格为下一个选中的方格。

6. 将下一步设为当前格，并从开放列表内删除。回到步骤1。

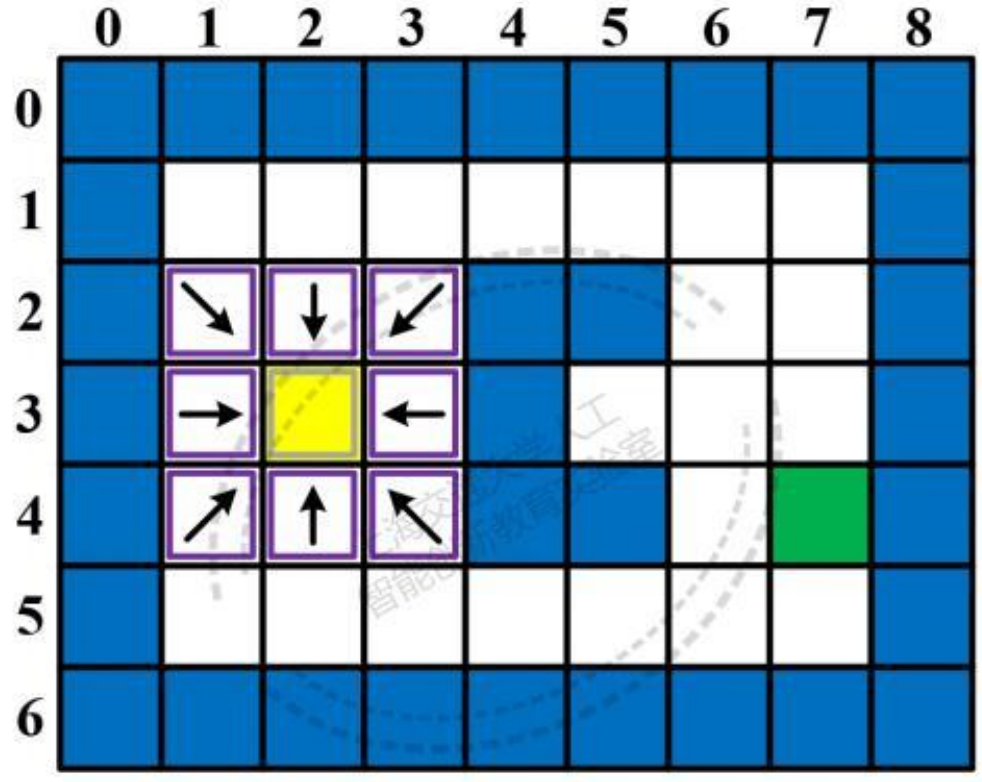
A*搜索解决迷宫问题

下图是一张迷宫地图，蓝色方格代表墙壁（障碍物），黄色方格(3, 2)点代表出发点（起始格），而绿色方格(4, 7)点代表终点（终点格），A*算法如何从出发点走到终点呢？



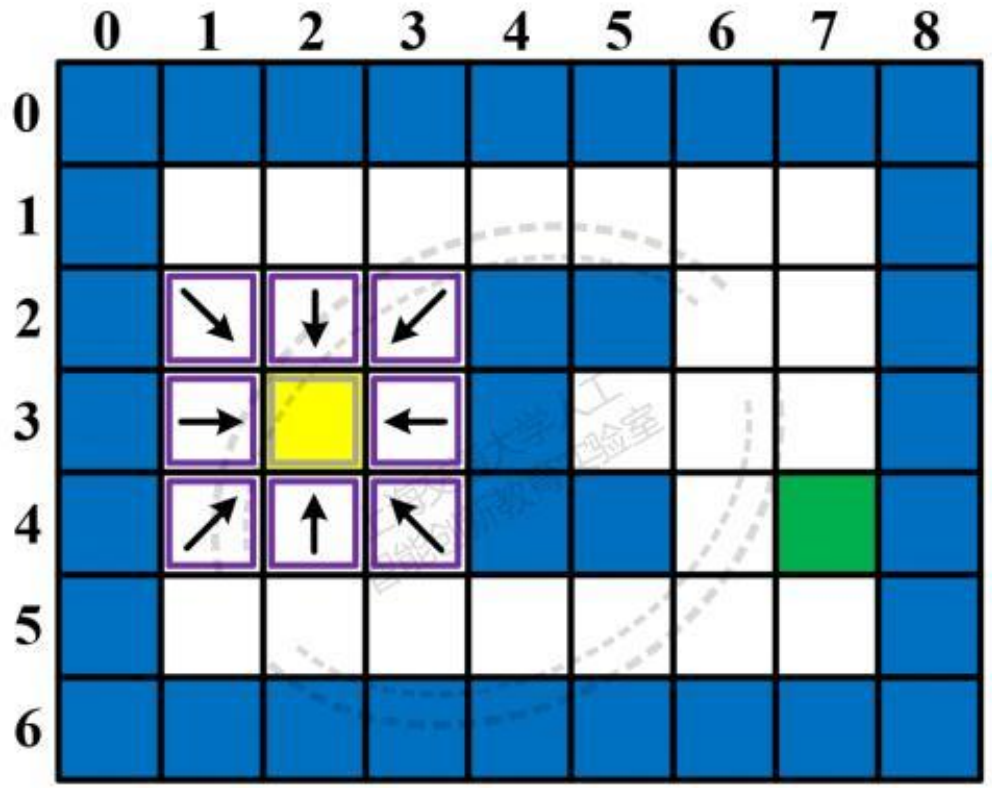
A*搜索解决迷宫问题

A*算法开始时，将起始格设为当前格放入封闭列表内（灰色框表示），作为当前格。



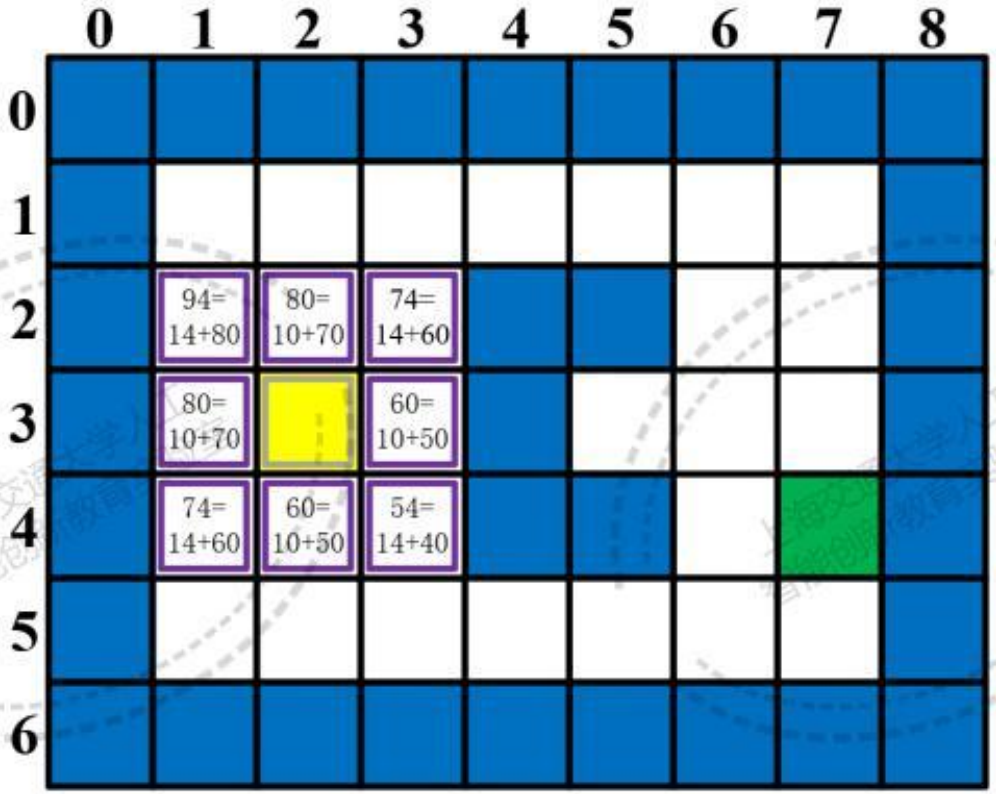
A*搜索解决迷宫问题

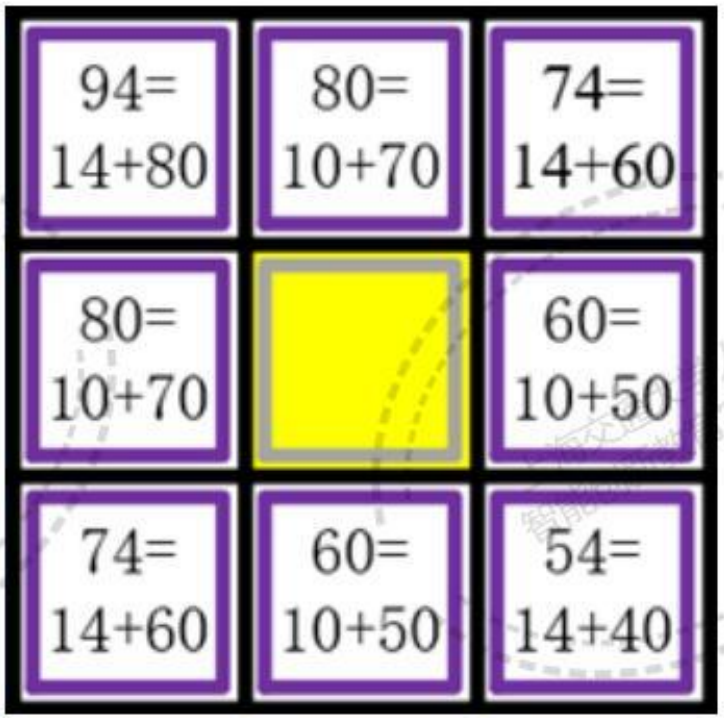
把当前格（起始格）周围可前进的方格放入开放列表，并将它们的父节点设定为当前格。



A*搜索解决迷宫问题

接着依据上述提到的A*算法公式F=G+H，求出每个在开放列表内方格的F值，决定F值最小（即为总代价最小）的方格为下一步的当前格。

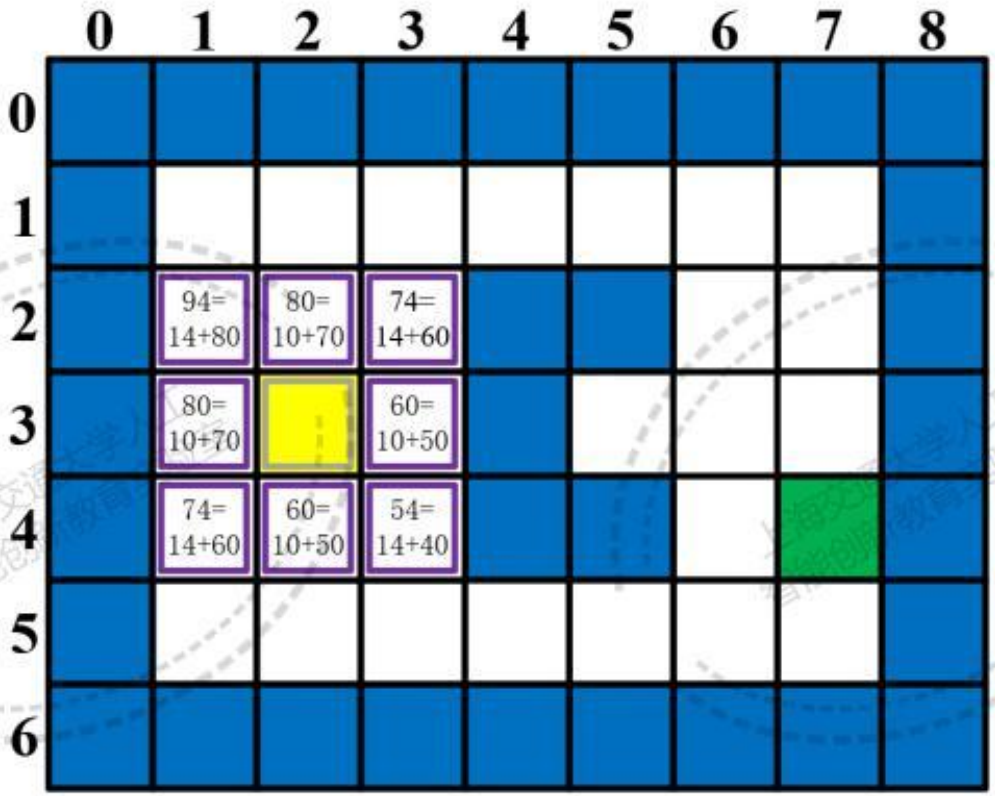


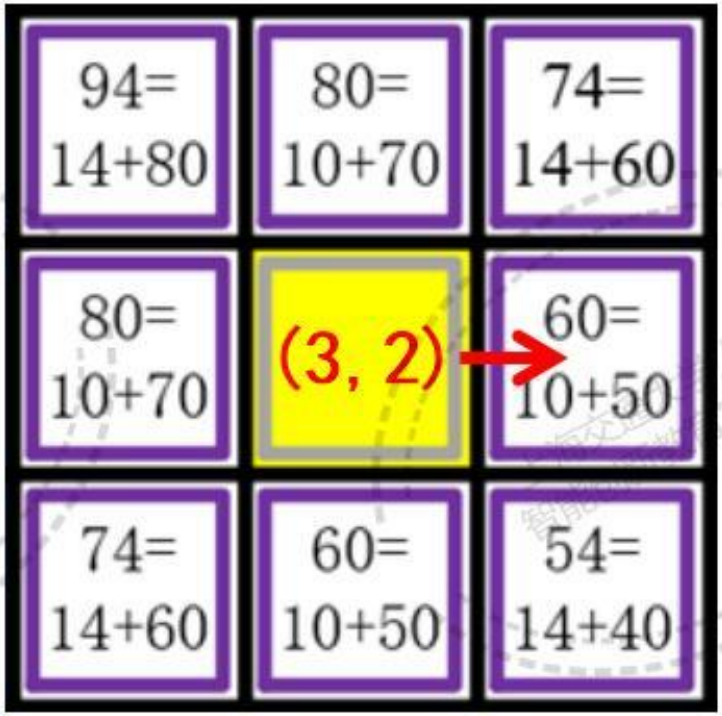


A*搜索解决迷宫问题

由(3, 3)点计算F值来说明，G值为起始格(3, 2)到当前格(3, 3)向右前进一格，

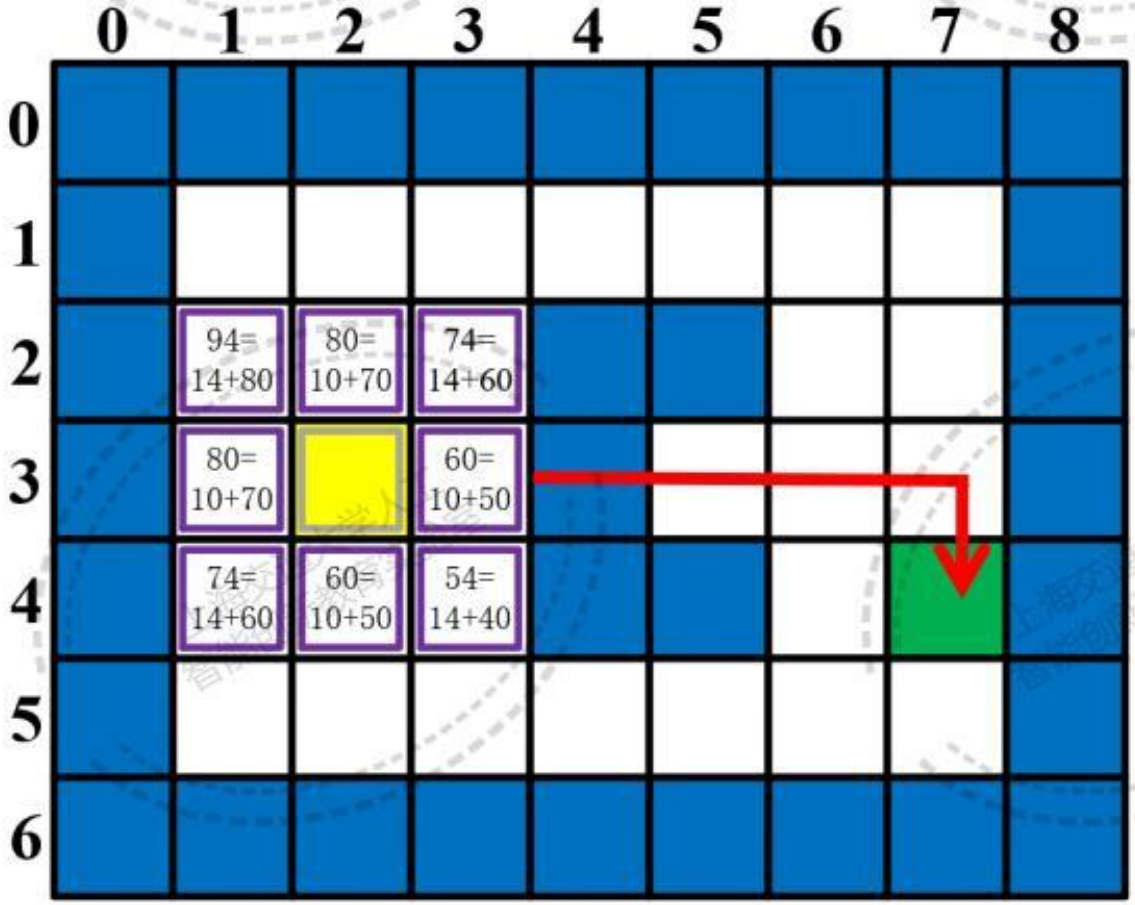
因此**G值为10**。





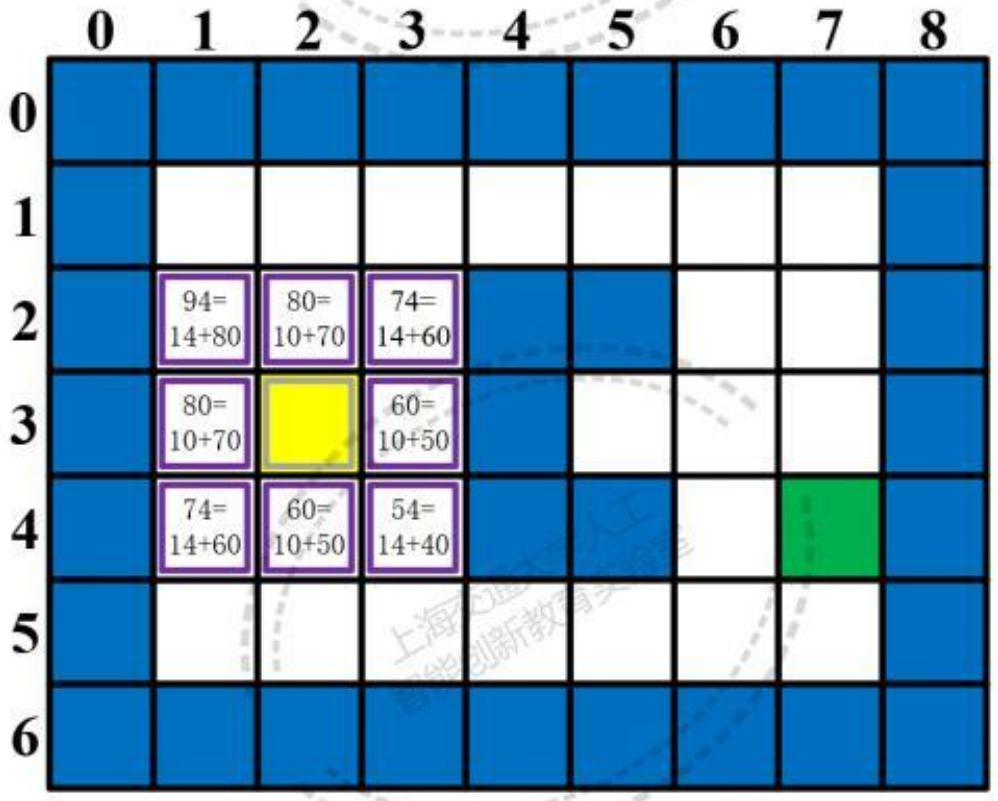
A*搜索解决迷宫问题

而H值为当前格(3, 3)到终点格(4, 7)的曼哈顿距离，即为水平距离7-3=4加上垂直距离4-3=1，4+1=5（格）一格代价为10，因此**H值为50**。



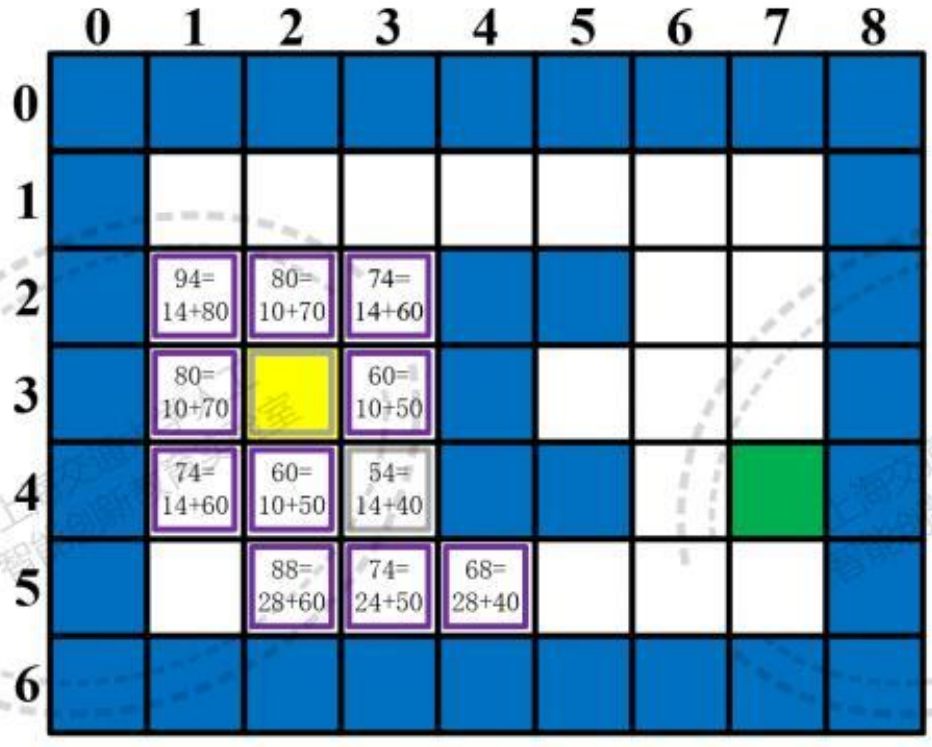
A*搜索解决迷宫问题

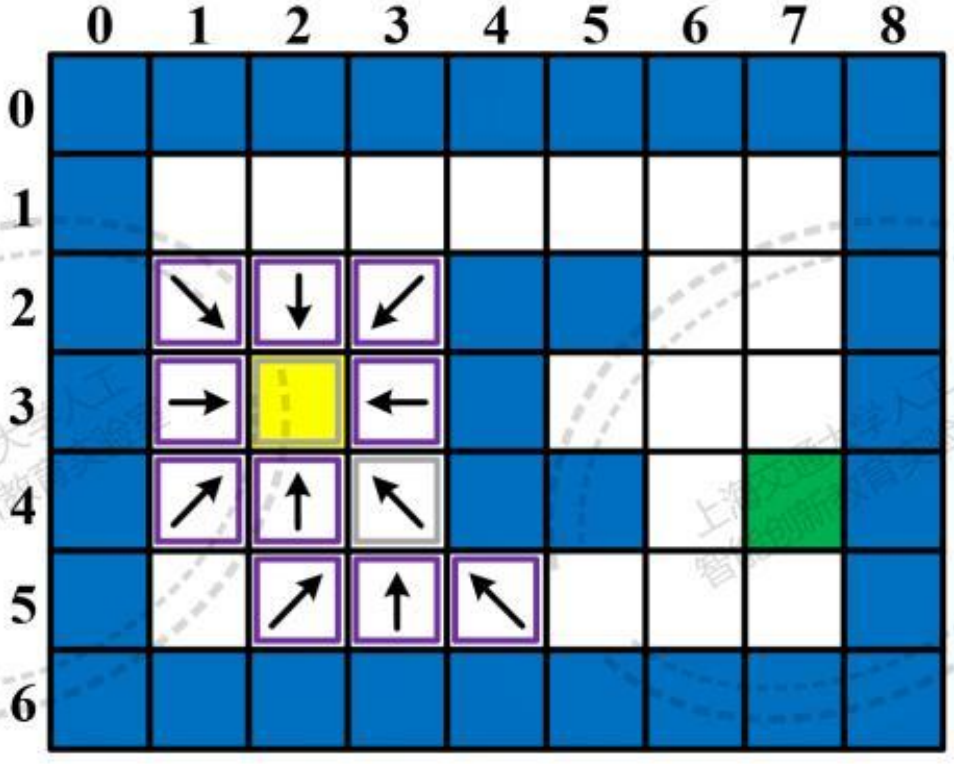
最后(3, 3)点的F值就为10+50=60。



A*搜索解决迷宫问题

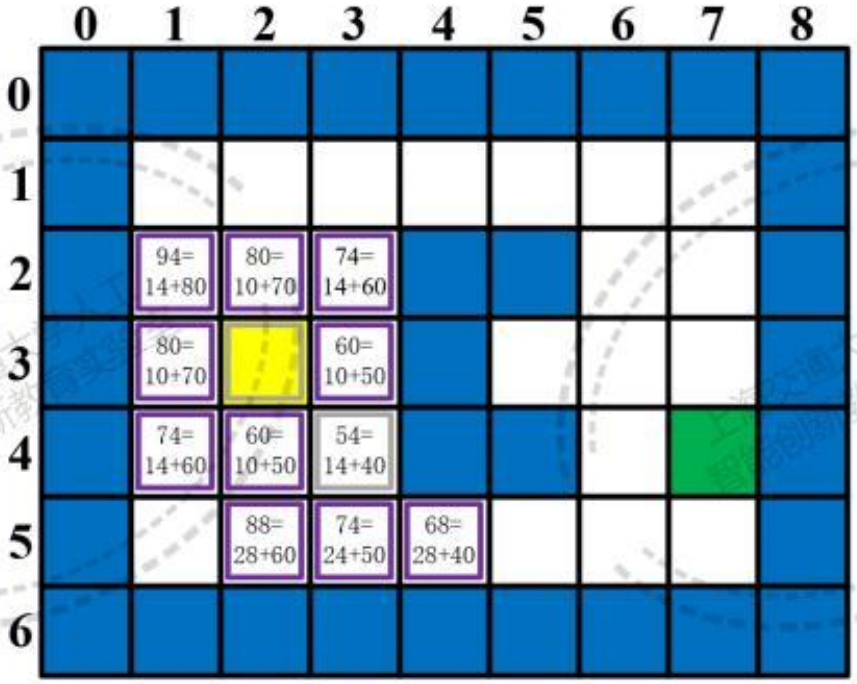
最终计算的结果是，右下角的方格总代价最小，而且它不是终点格，因此设为下一步的当前格，并从开放列表内删除。

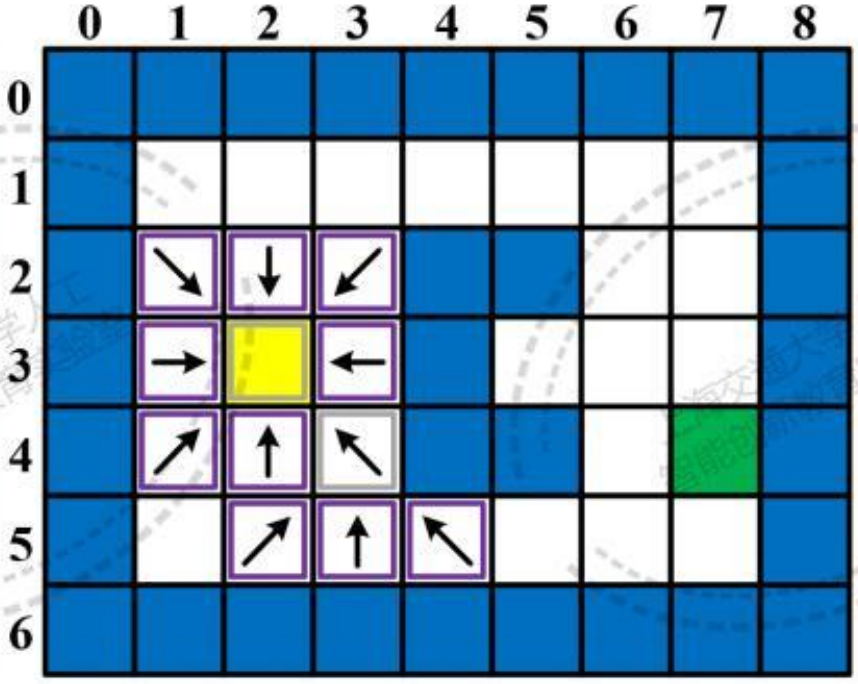




A*搜索解决迷宫问题

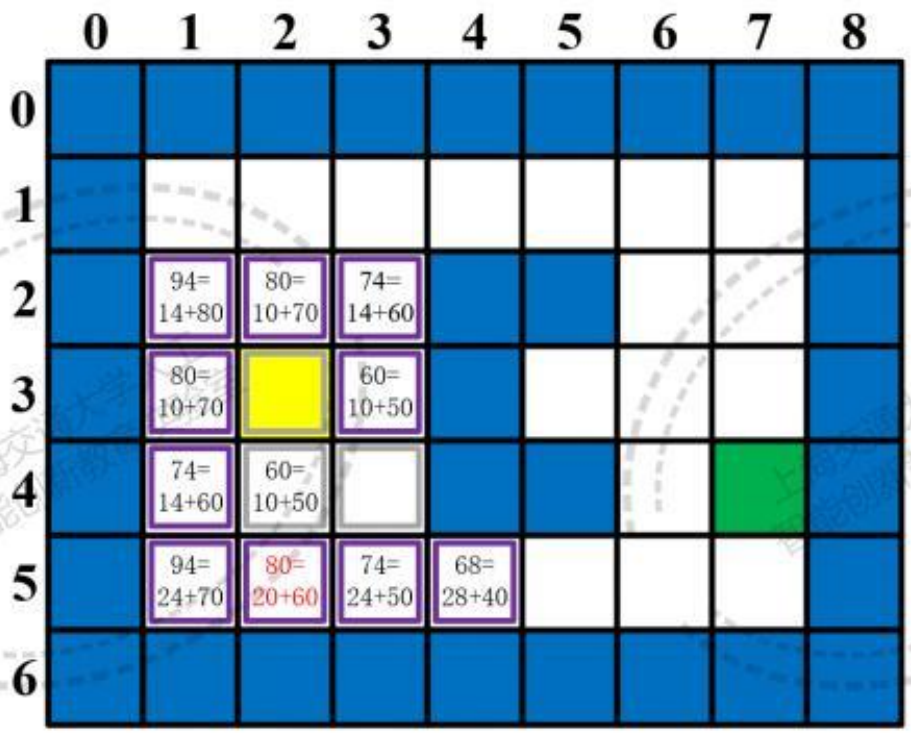
进入下一步。将当前格放入封闭列表内，将当前格不在开放列表内的相邻方格加进开放列表内，并设定为这些方格的父方格。对已经在开放列表中的相邻方格，则计算新的G值，看是否比当前G值小。

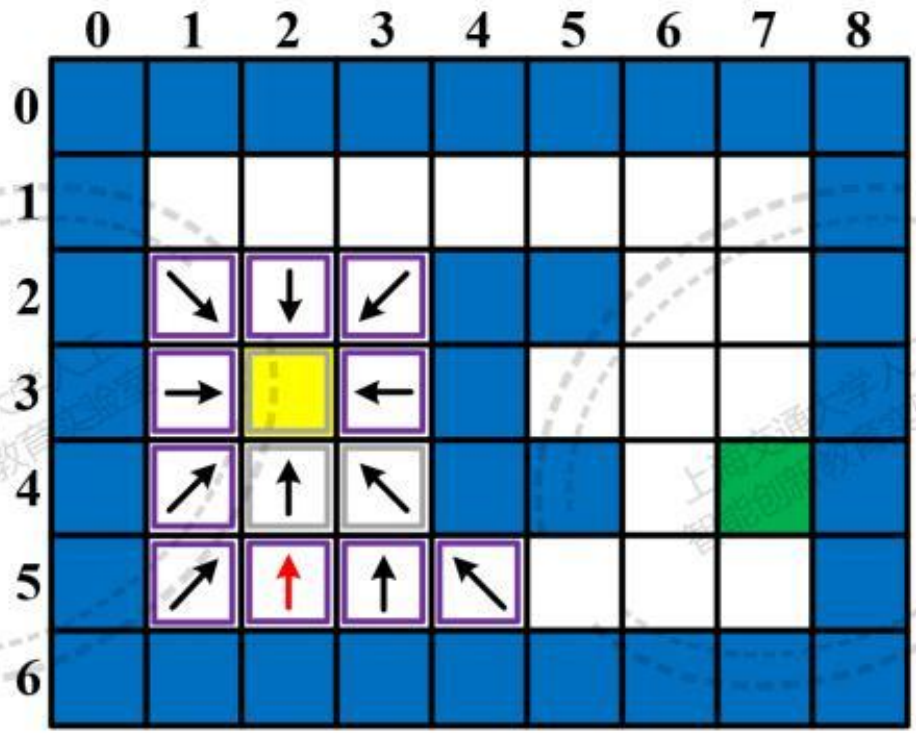




A*搜索解决迷宫问题

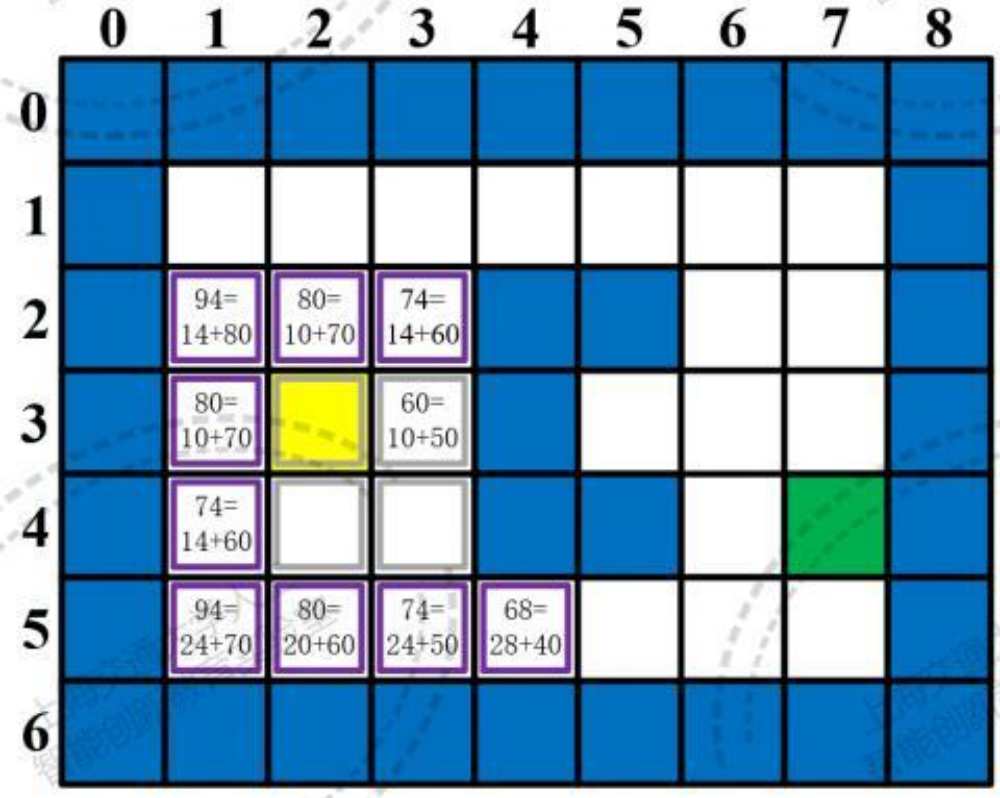
如果新的G值更小，就需要更新G值和父方格。在第三步中就出现了新G值更小的情况。第三步中，方格(5, 1)的新G值为20，小于之前的G值28，所以更新G值，并将父方格设定为当前格(4, 2)。

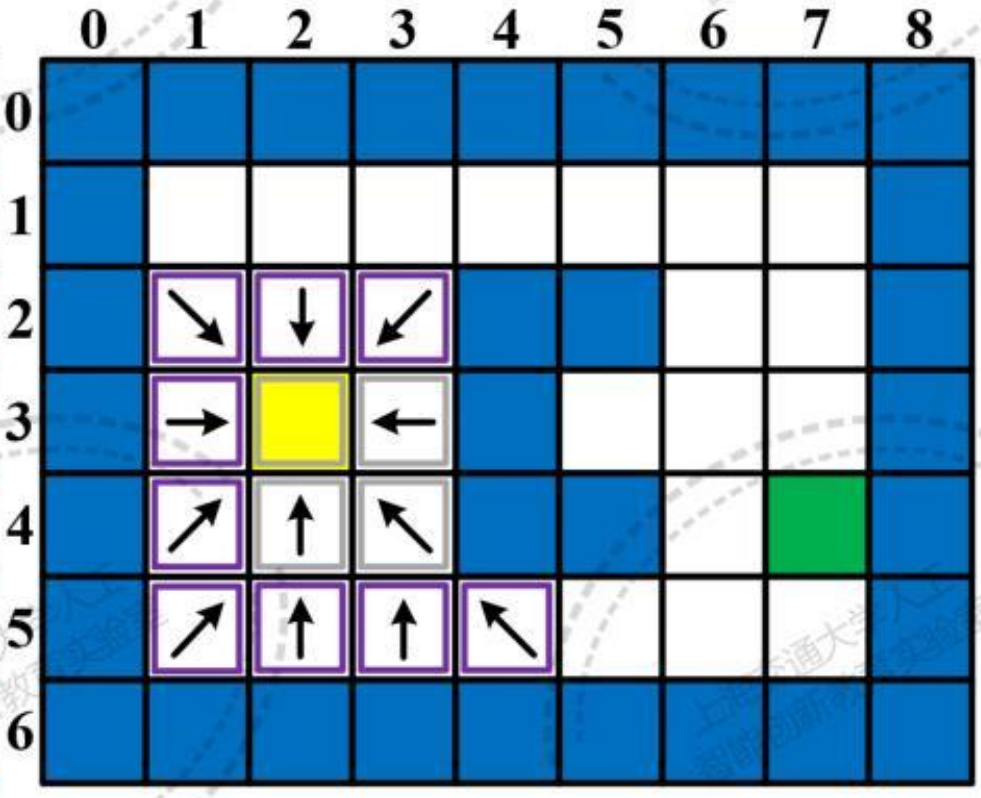




A*搜索解决迷宫问题

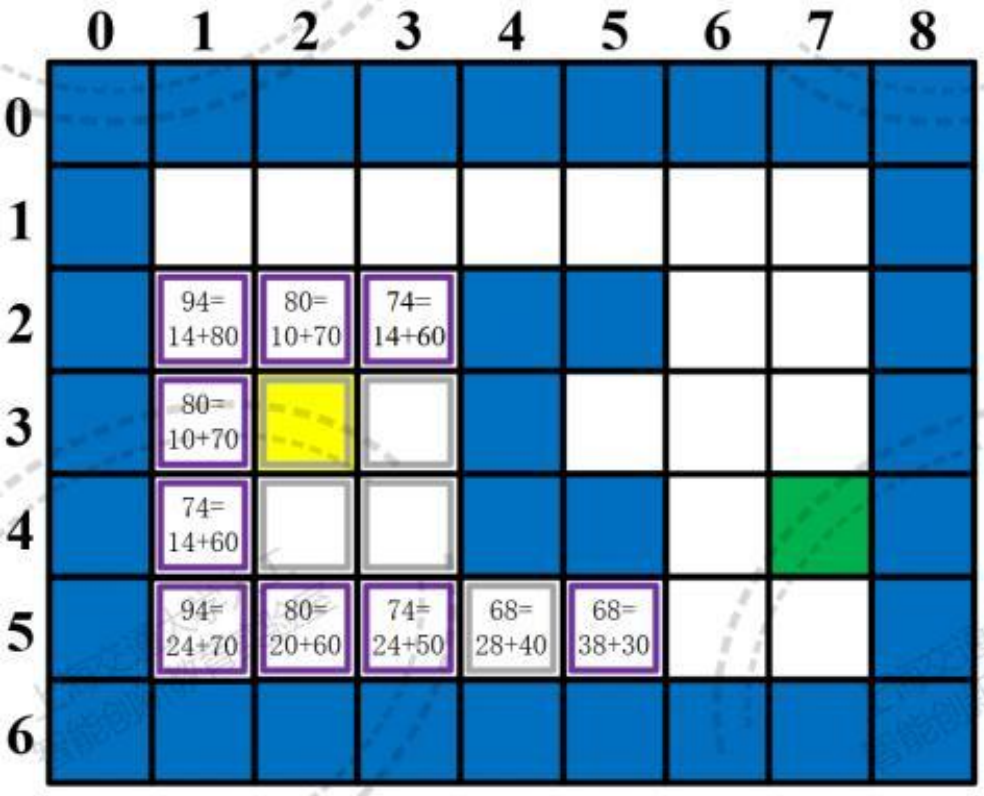
第四步

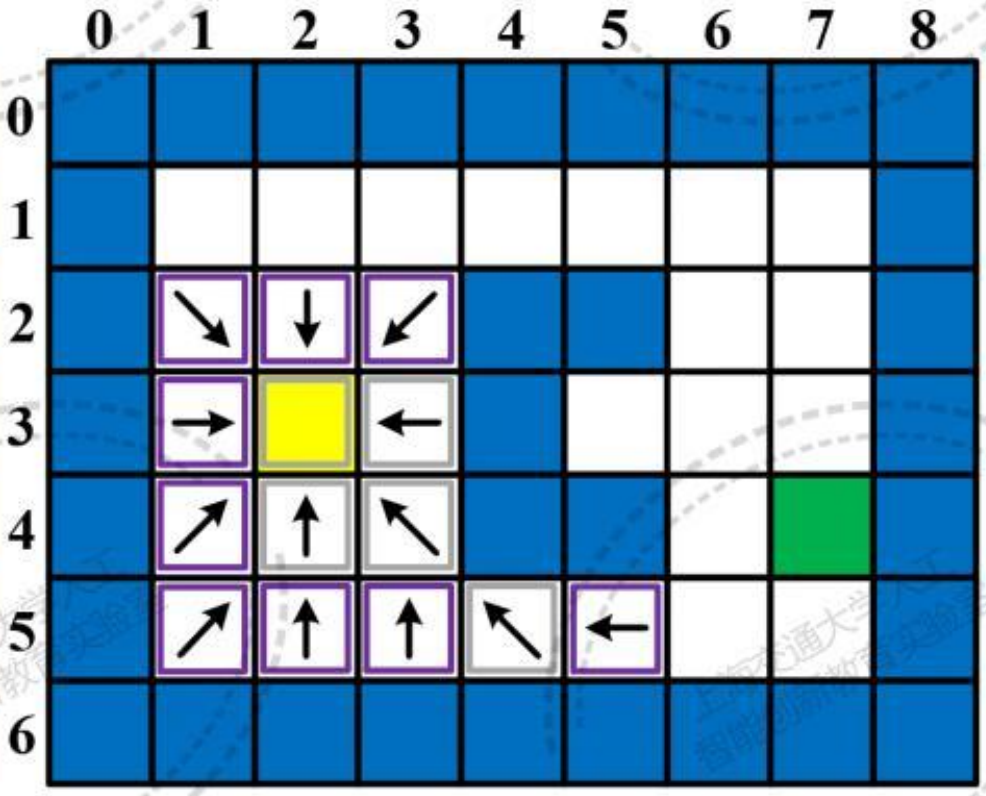




A*搜索解决迷宫问题

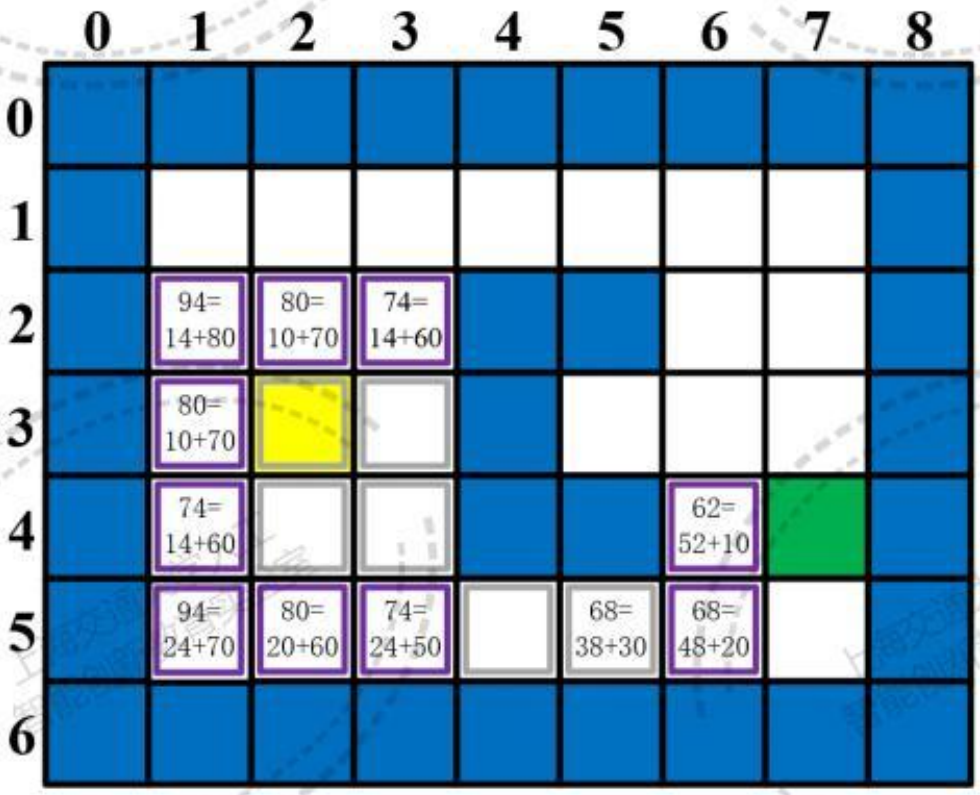
第五步

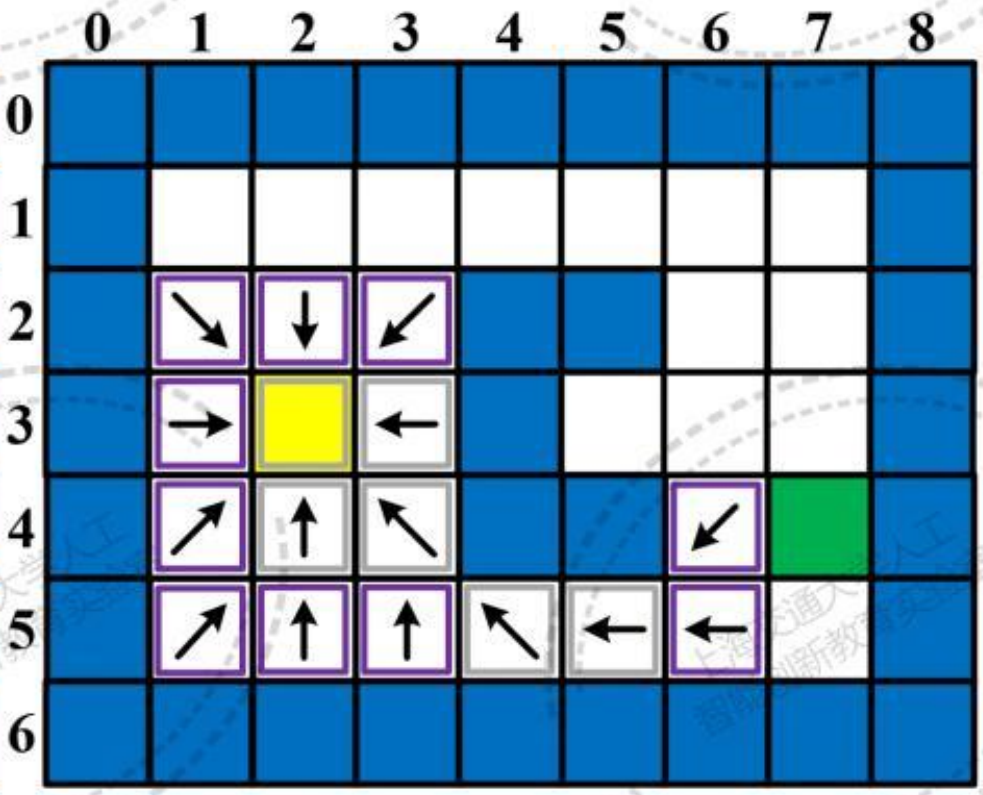




A*搜索解决迷宫问题

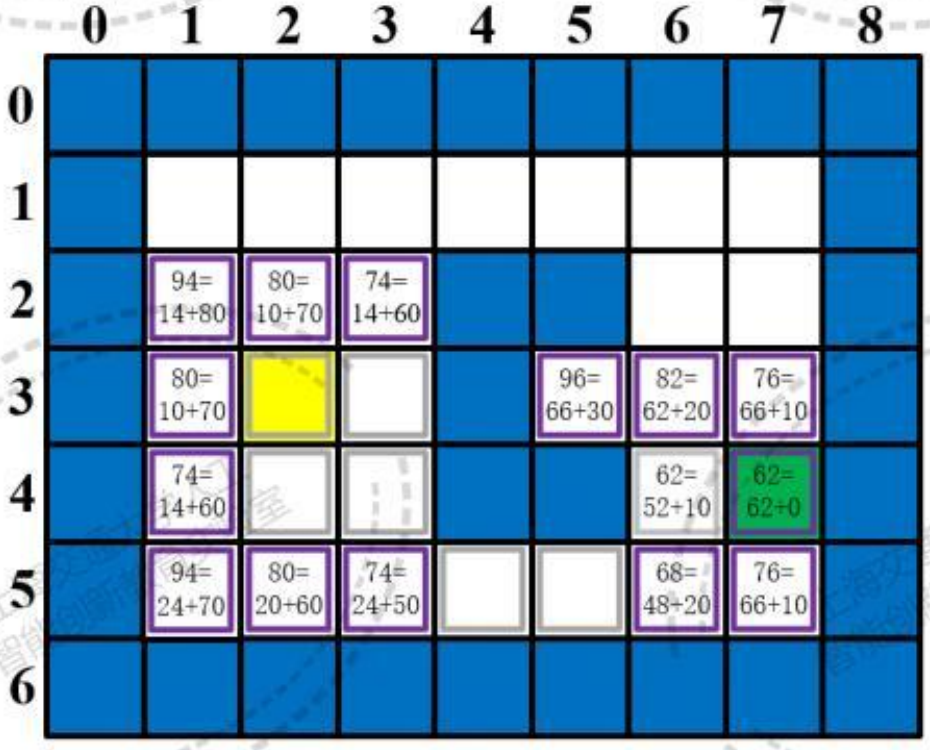
第六步

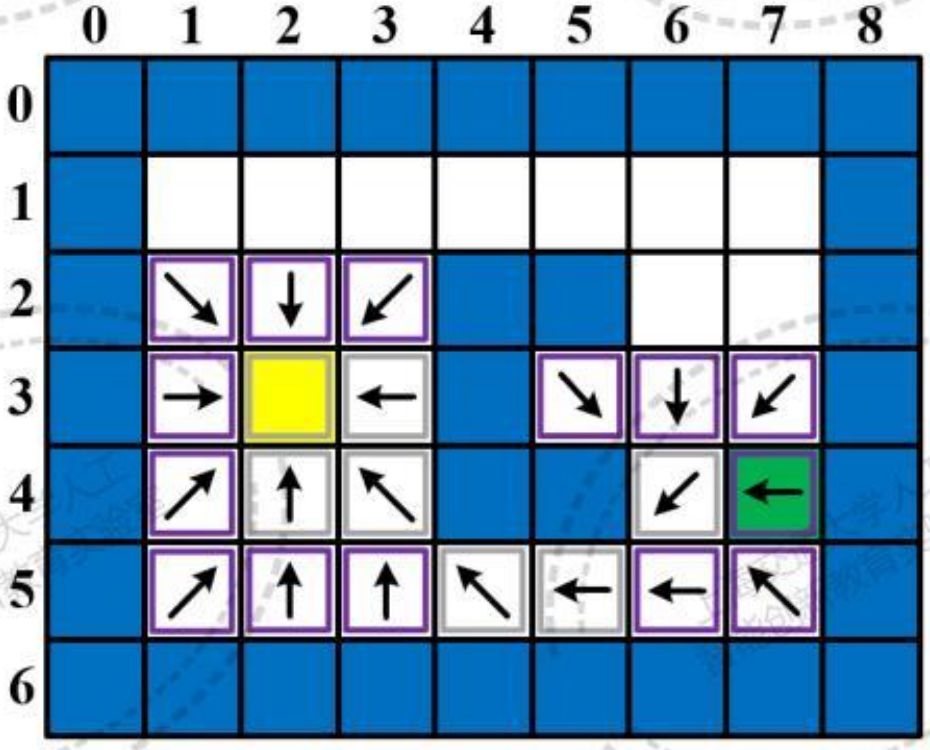




A*搜索解决迷宫问题

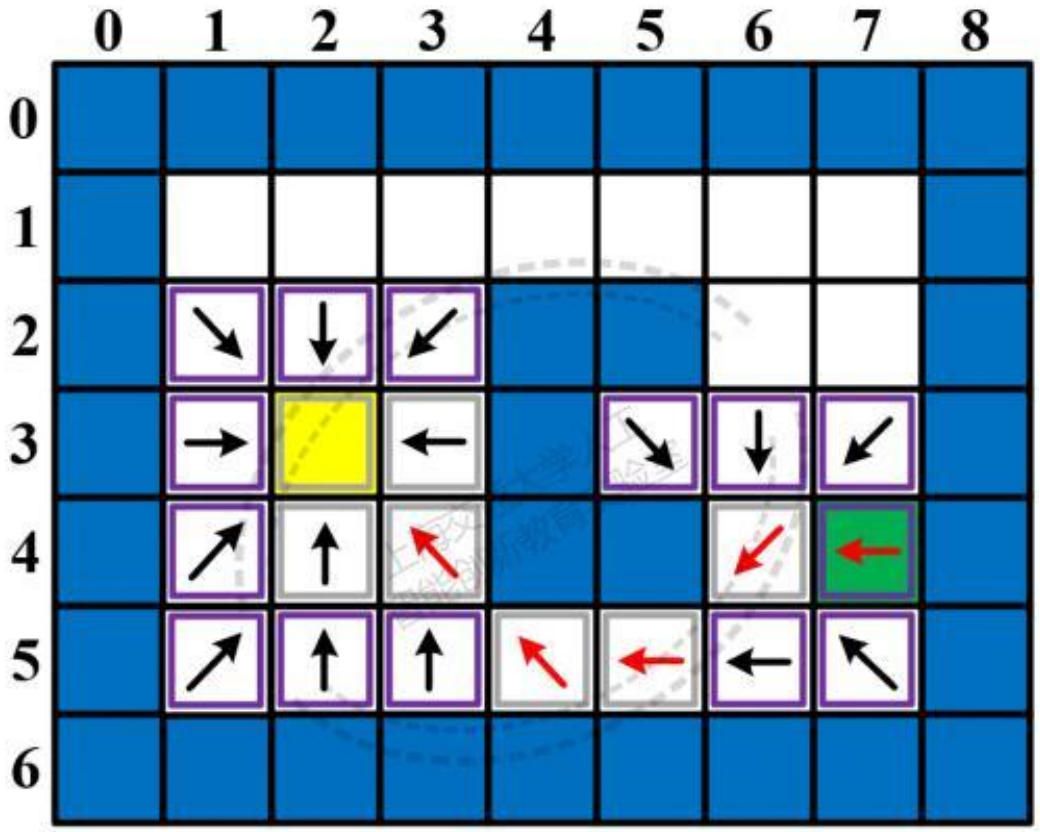
第七步中，终点格被放进开放列表内，算法结束。





A*搜索解决迷宫问题

最终，根据指向父方格的关系向上追溯（红色表示），从终点格追溯到起始格就可以得到最终路径。这就是记录父方格的作用。




感谢倾听
THANKS FOR YOUR ATTENTION

扩展资料

可以访问以下网址来查看更复杂的迷宫问题的寻路算法演示：

https://cs.stanford.edu/people/abisee/tutorial/