第5章

搜索技术

5.1 搜索问题的定义

搜索是为了达到某一目标而进行寻找的过程。搜索技术就是对寻找目标的过程进行引导和控制的技术，它是人工智能领域的基本技术之一。在生活中我们无时无刻不在做出选择，出门旅行该走哪一条路，运输货物时应该如何摆放，其实都是在多种方案中搜索最好的或者看起来最好的，都是人工智能搜索技术需要解决的问题。

例如常见的农夫过河问题：农夫、狼、羊、白菜在河边准备渡河，只有农夫能开船，船上除农夫外只能放一个动物或物品，如没有农夫看管，狼会吃羊，羊会吃白菜，试问农夫如何才能让所有动物和物品安全渡河？若要解决此问题，每次渡河前（当前状态）必须从多种可选方案中选出正确或看起来正确的方案（候选的“下一个状态”），其实质是从多种有前后关系的方案中不断进行选择，直到搜索出一个满足条件的渡河方案。同时，如何最快地搜索到满足条件的方案，满足条件的方案是否是代价最小的（这里可考虑渡河步骤最少的），也是需要考虑的问题。

搜索技术是人工智能技术的重要组成部分，也是早期人工智能的主要基础技术之一。在早期的研究中，深度优先、广度优先等盲目搜索技术和启发式搜索技术就得到了广泛应用，如求解八数码问题、五子棋问题等。1968 年，尼尔斯·约翰·尼尔森（Nils John Nilsson）发明了 A^* 搜索算法，为人工智能领域带来了重大影响。A^* 算法广泛应用于状态空间求解。博弈搜索也是搜索技术的另一大应用。从 20 世纪 60 年代起，人们对博弈搜索的研究也极大促进了人工智能的发展。20 世纪 90 年代，IBM 公司的“深蓝”计算机战胜了人类的国际象棋世界冠军，把博弈搜索推向了高峰。近年来，深度学习技术兴起，人们把深度学习技术和博弈搜索技术相结合，使得计算机战胜了人类的围棋世界冠军，宣告了人工智能发展新高潮的来临。

5.2 状态空间

问题求解过程中的每一步问题状况可称为一个状态，一个问题的全部状态以及这些状态之间的相互关系称为状态空间，通常可以用图来表示，称为状态空间图。很多搜索问题都可以转化为状态空间图的搜索，上述农夫过河问题也是如此。若我们用一个长度为 4 的数组表示过河状态，某一个数组元素值为 0 表示在河的左岸，值为 1 表示在河的右岸，数组第 1 个元素到第 4 个元素分别表示农夫、狼、羊、白菜的过河状态，上述问题则转化为：出发时的状态可表示为(0,0,0,0)，最后需要达到的状态为(1,1,1,1)，渡河过程中每一步的状态和可选择的方案都可以表示为一组数组值，如(1,0,1,0)表示农夫和羊过了河，狼和白菜留在河的左岸。又因为狼要吃羊、羊要吃白菜，所以有些状态是不能出现的，如(1,1,0,0)、(0,1,1,0)等。如果一个状态可以直接变化为另外一个状态，二者之间就有一个链接，如(0,0,1,0)可以直接变化为(1,1,1,0)或(1,0,1,1)。因此，所有可能的选择可以构成一棵“树”，如图 5.1 所示，则搜索的过程相当于找到从结点(0,0,0,0)出发到结点(1,1,1,1)的过程。

如图 5.1 所示的状态和连接构成的图便是农夫过河问题的状态空间图。基于状态空间图的搜索即是找到从初始状态到目标状态的路径的过程。

农夫过河问题比较简单，其搜索过程产生的状态是有限的，因此可以构建完整的状态空间图。然而实际生活中碰到的问题往往比较复杂，比如下围棋，其状态太多，不可能用状态空间图完整地表示出来。一般的处理方式是边搜索边生成后续的状态结点，直到找到目标状态结点，这样可以避免生成不会搜索到的状态结点。在搜索过程中，生成的无用状态结点越少，搜索效率也就越高，这也是评价一个搜索策略好坏的标准之一。如图 5.2 所示，如果最终搜索的路径是 1-2-5-6-3-7-10-11，则虚线结点不用生成。

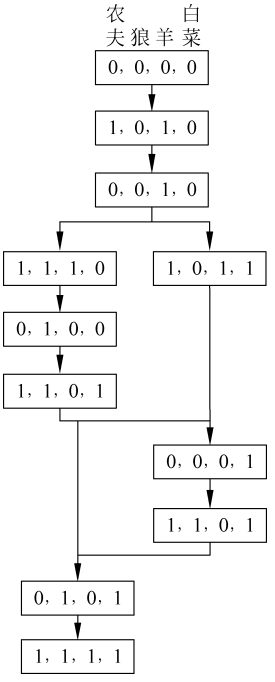


图 5.1 农夫过河问题的状态空间

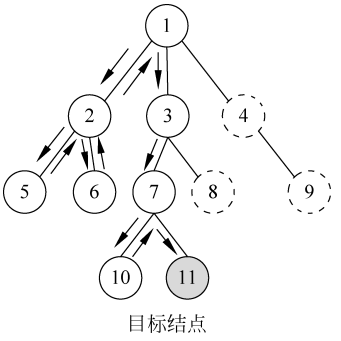


图 5.2 图的生成

搜索策略可分为盲目搜索和启发式搜索。在状态空间图的搜索过程中，如何从待选择结点集合中选择下一步的搜索目标呢？如果在选择时利用了相关知识和启发策略，则这种搜索称为启发式搜索，否则称为盲目搜索。

5.3　盲目搜索

盲目搜索又称为通用搜索，常见的盲目搜索策略包括深度优先和广度优先搜索策略。盲目搜索不针对特定问题，实际上相当于按照一定的顺序遍历状态空间图中的所有结点以找到目标结点，因此算法简单，适应性非常强，但是搜索效率低下。对于状态很多的问题，如下国际象棋，盲目搜索不可能在有限的时间内完成搜索工作，但是对于一些较为简单的问题，盲目搜索可以发挥其优势。

5.3.1　深度优先搜索

深度优先搜索首先考虑纵深方向的搜索，如果没有下级的结点可搜索，则“回溯”到上一级，更换其他路线继续往纵深方向搜索，直到搜索到目标结点。从多个下级结点进行选择时，通常选择从左边起第一个未被搜索到的结点开始向右搜索（选择从右边起也是一样）。如图 5.3(a)是需要搜索的状态空间图，假设目标结点是 11，初始出发结点是 1，先逐级往下级搜索直到搜索到结点 5（见图 5.3(b)），结点 5 没有下级，并且此时还没有搜索到目标结点，因此需要“回溯”到结点 2。在图 5.3(c)中，回到结点 2 后继续搜索结点 2 的下级结点中未被搜索

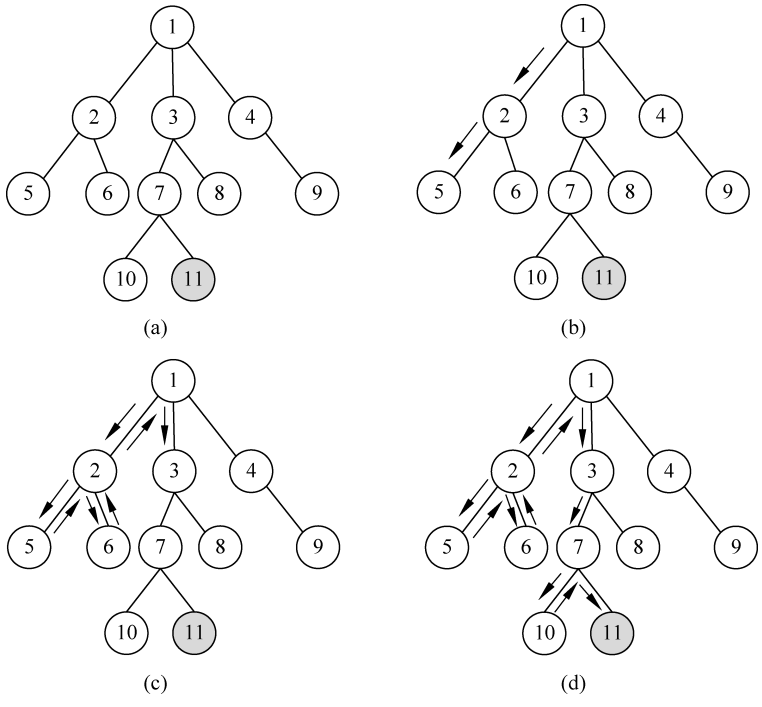


图 5.3　深度优先搜索过程

到的结点 6 并继续往下搜索，发现结点 6 也没有下级，则继续“回溯”到结点 2，此时结点 2 已经没有未被搜索的下级结点，因此继续“回溯”到结点 1，选择结点 1 的下级结点中未被搜索的结点 3，图 5.3(d)中，按照此规则继续从结点 3 往下搜索，直到搜索到目标结点 11。

深度优先搜索的特点是能很快地往纵深方向搜索。若目标结点在图的最左边，则能很快搜索到。若在右边，则搜索效率大大降低。同时要注意，深度优先搜索时已经被搜索过的结点不能重复搜索，否则会出现“死循环”问题，即一直循环搜索部分结点，永远不会结束。

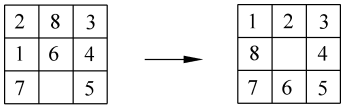


图 5.4 八数码问题示例

以下以八数码问题为例，看看如何用深度优先搜索解决此问题。八数码问题是 8 个数字放在一个 3×3 的表格中，表格中的数字可以和相邻的空格交换位置。八数码问题的操作过程是给定一个初始状态和一个目标状态，需要通过上述交换操作把初始状态变为目标状态，图 5.4 是一个八数码问题示例。

若我们将表格中的空格按照左、上、右、下的顺序依次尝试和其他数字交换位置，则其深度优先搜索过程如图 5.5 所示，图中每个结点的序号代表该结点被搜索的顺序。

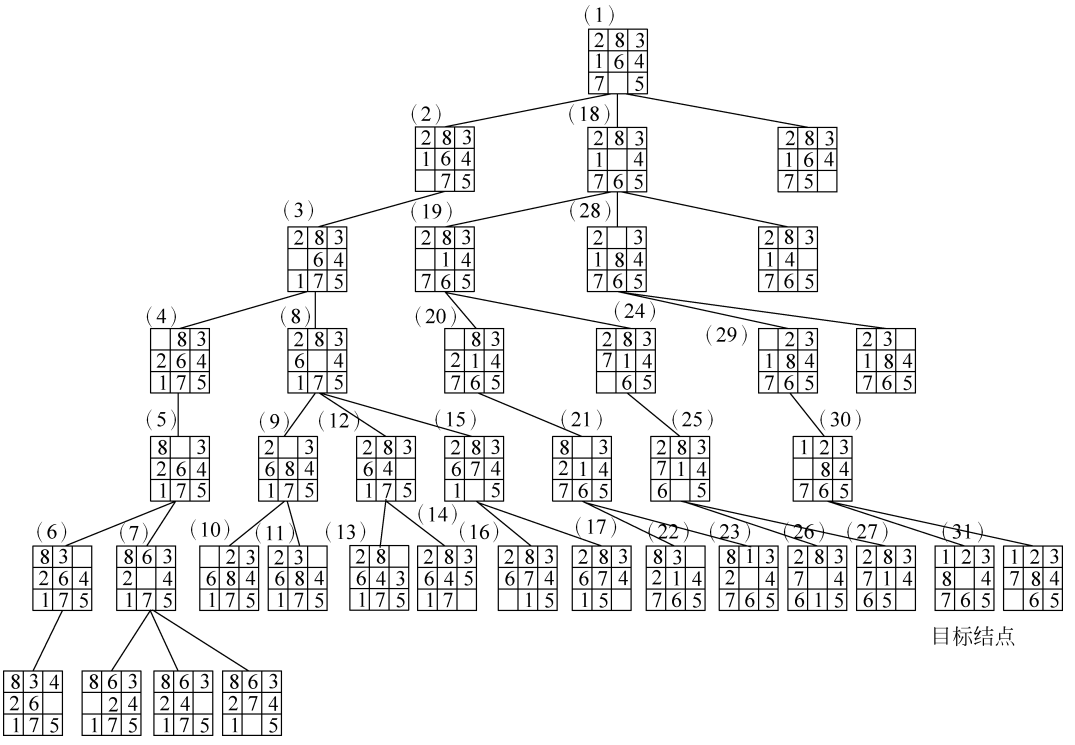


图 5.5 深度优先搜索求解八数码问题

在进行深度搜索时，应优先搜索下级结点，但是如果这个状态空间图的深度过深，导致结点过多，会使得算法的运行时间过长，为了把搜索限制在一定的时间范围内，通常可以限制搜索的深度(所谓“深度”即是从初始结点到某个结点所经过的“层”数)。图 5.5 中的搜索指定了搜索的深度为 6，当到达搜索深度限制后，无论是否还有下级结点，都回溯到上一级

结点继续搜索，直到找到目标结点。当搜索到结点 6 时，由于到达了深度限制，所以略过结点 6 的下级，回溯到结点 5，按照此策略继续搜索，直到搜索到结点 31。

限制搜索深度可以减少搜索时间，但有可能造成搜索不到目标结点，例如如果图 5.5 的搜索深度限制设置为 5，则搜索不到正确解。因此在设置搜索深度限制时，需要根据经验设置合理值，或者当搜索不到目标结点时，动态地逐步增大搜索深度。

5.3.2　广度优先搜索

广度优先搜索首先考虑水平方向的搜索，如果水平方向上所有结点都已搜索完毕，并且未搜索到目标结点，则继续搜索下一级，直到搜索到目标结点。通常选择从左边起开始搜索同级的结点(选择从右边起也是一样)。如图 5.6 所示，从结点 1 开始搜索，按从左至右的顺序逐层搜索每一个结点，直到目标结点 11。

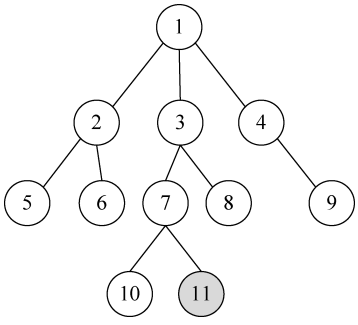


图 5.6　广度优先搜索过程

图 5.7 是用广度优先搜索求解八数码问题的过程，序号代表搜索顺序，注意图中省略了一些结点(如结点 4)的下级，在真实的搜索过程中不能省略。

广度优先搜索和深度优先搜索都属于盲目搜索，两者的搜索效率本质上没有差别，当状态空间图的宽度(指同一级的结点个数)过大时，也可以采取限制搜索宽度的方式控制算法的执行时间。

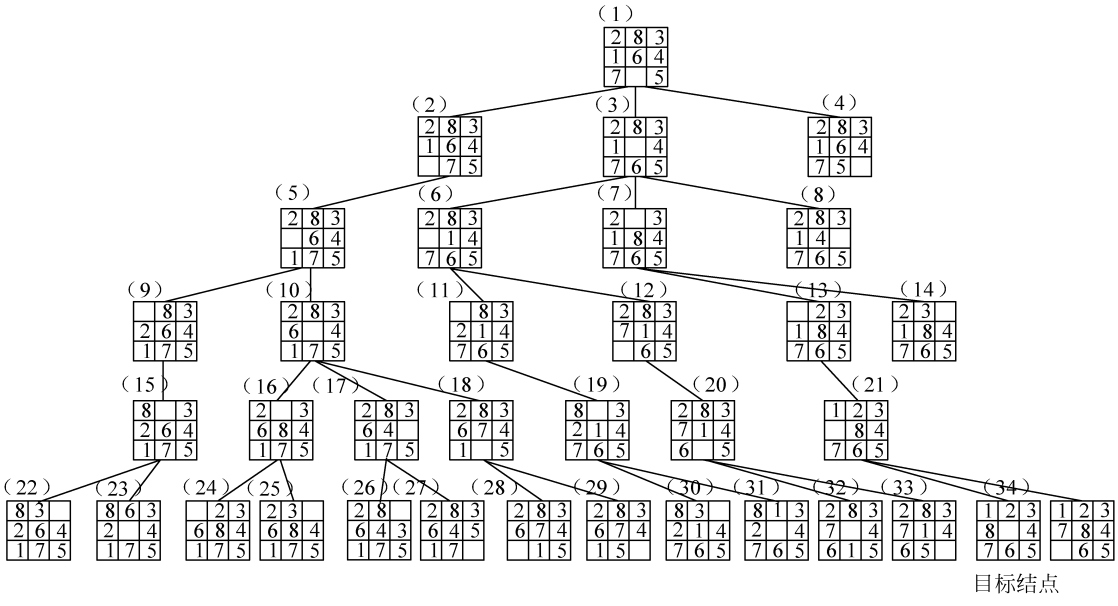


图 5.7　广度优先搜索求解八数码问题

5.4 启发式搜索

人类在进行选择时，很多时候会利用一定的线索或者经验知识来帮助确定选择方向。比如，在爬山时，如果面前有两条路（一条为上坡，另一条为下坡），我们很可能会选择上坡的路，虽然选择上坡路不一定能到达山顶，但是根据人的经验，上坡的路到达山顶的概率更大，这体现了一定的智能。

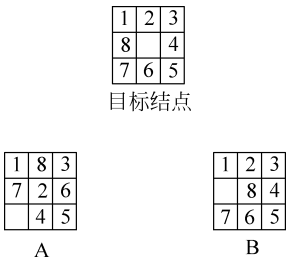


图 5.8 八数码问题结点选择

人在解决八数码问题时，会根据经验选择看起来更好的结点作为下级结点。图 5.8 中，当人在面临 A 和 B 两个结点时，往往会选择 B 作为下一步的搜索方向，因为 B 和目标结点看起来更相似（看起来更好）。

启发式搜索便是利用一个启发函数来模拟人在进行选择时的这种“启发性”，用这个启发函数来评估当前备选状态和目标状态的相似性，一般用 $h(x)$ 来表示，x 代表一个备选状态。确定启发函数是启发式搜索算法的关键。

通常，启发函数可以定义为对备选状态到目标状态的距离或者差异的度量，也可以定义为当前结点在最佳路径上的概率或者某种规则，具体如何定义需要考虑待解决的问题的实际情况。以图 5.9 所示的八数码问题为例，当前状态为 A，B、C、D 为 3 个备选子结点。如何对备选结点进行评估呢？

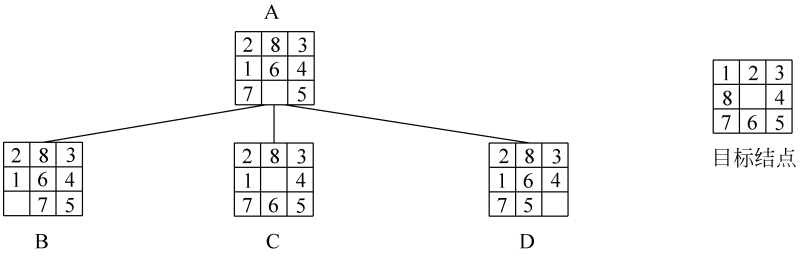


图 5.9 八数码问题结点评估

我们可以通过一些策略来定义启发函数对结点进行评估，最直观的策略是考虑不在正确位置的数字的个数，则我们定义启发函数为：

$$h(x)=\text{不在正确位置的数字的个数}$$

根据上述启发函数的定义，我们可以计算出每个结点的 $h(x)$ 值，显然，$h(x)$ 值越小，则结点看起来越好。可计算出 $h(B)=5$，$h(C)=3$，$h(D)=6$，则 C 是最好的选择。根据这个启发函数，重新搜索前面的八数码问题的状态空间图。图 5.10 中深色背景的结点是被选中过的结点，浅色背景的结点是未被选中过的结点，每个结点上方的括号中标明了该结点的 $h(x)$ 值。图 5.10 的步骤①选择了结点 A，然后展开其下级结点 B、C、D 作为备选结点。步骤②从备选结点中选择了 C 并继续展开结点 C 的下级，此时备选结点有 5 个（B、D、E、F、G），其中结点 E 和 F 的 $h(x)$ 值最小且值相同，在步骤③中通常选择左边的结点 E 并按此规则搜索到结点 P。在步骤④中，由于结点 P 没有下级，所以在备选结点中 F 的值最小，因此选择结点 F 并直到搜索到目标结点。

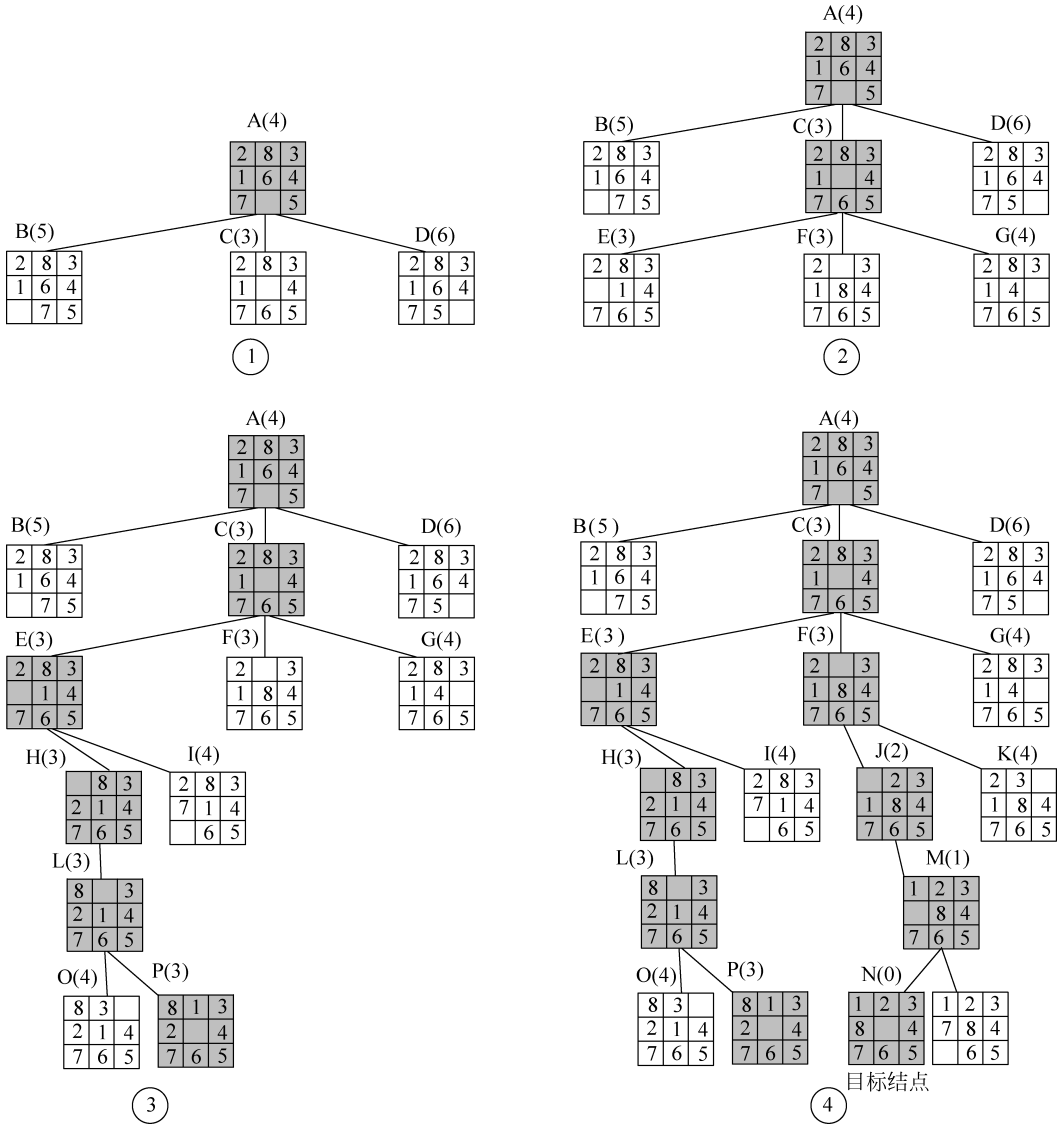


图 5.10　不考虑距离的八数码问题启发式搜索

然而，上述搜索过程中为了简化问题省略了部分结点，如 P 结点并未继续展开下级。如果 P 结点继续展开且有 $h(x)$值不大于 3 的下级结点，则会继续选择其下级结点，这样可能搜索不到或者要花更多的时间才能搜索到目标结点。如何才能避免这种情况？一种方案是在计算 $h(x)$值时不仅考虑备选结点与目标结点的相似性，同时也考虑从初始结点搜索到备选结点的代价，这样如果当前的下级备选结点的代价过大，即使其下级与目标结点相似度很高，也可能选择其他结点，这便是完整的启发式搜索，其典型的代表是 A 算法。

根据上述思路完善算法，用 $f(x)$代表搜索 x 所花费的总代价，增加一个函数 $g(x)$，代表从初始结点搜索到当前结点 x 所花的代价，$h(x)$为启发函数，代表从当前结点到目标结点的可能代价，则代价函数为：

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

用代价函数代替启发函数对所有结点进行评估。还是以上述八数码问题为例，$g(x)$代表从初始结点到 x 结点的最小步数，则 $f(\mathrm{A})=4+1=5$，$f(\mathrm{H})=3+4=7$，根据代价函数重新进行搜索，其过程变为图 5.11。由图 5.11 可见，比起前面不考虑从初始结点到备选结点代价的搜索，A 算法在此例中有着更好的搜索效率。

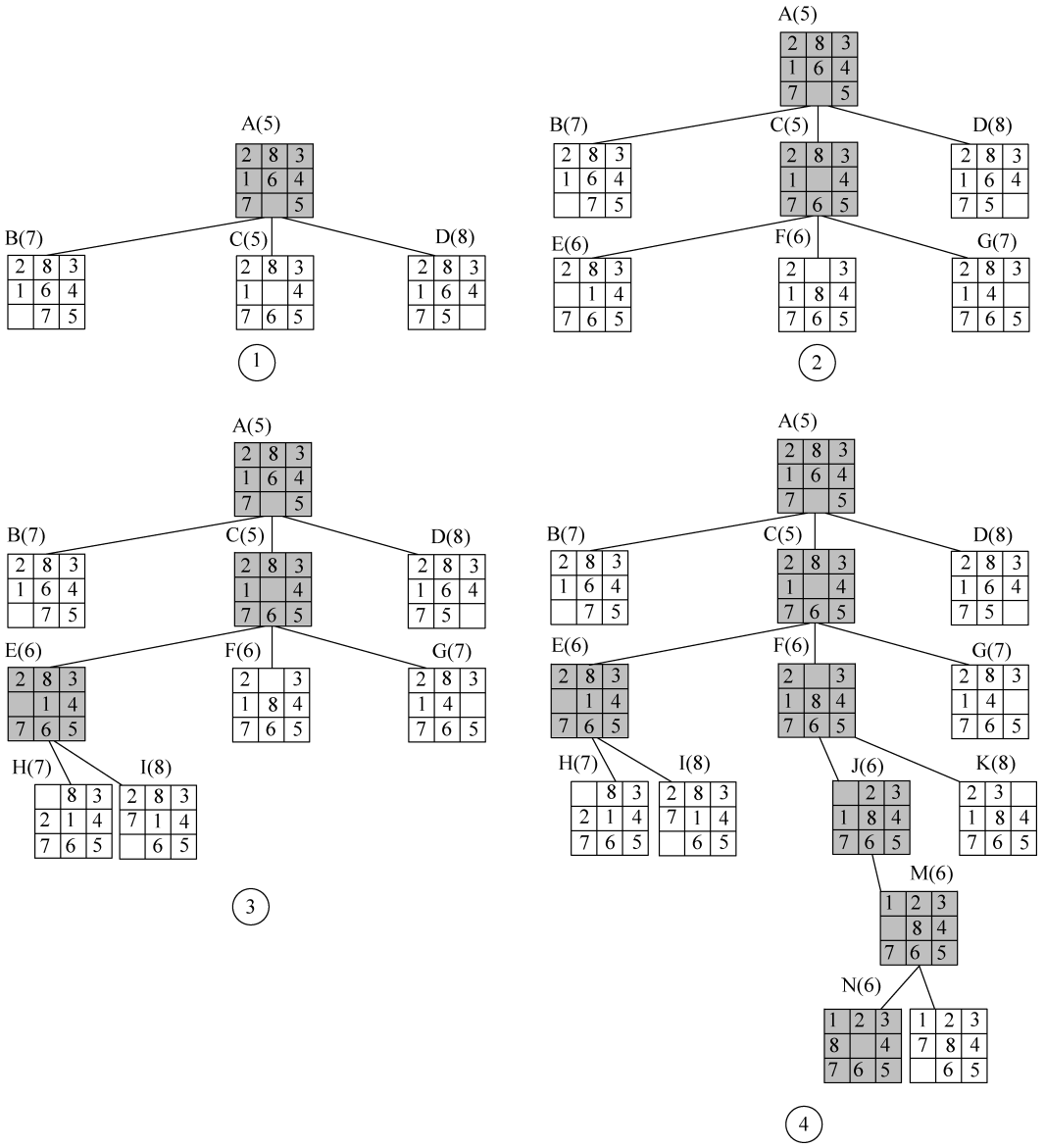


图 5.11　A 算法求解八数码问题

A 算法可以用以下思路来实现：用一个 OPEN 表保存搜索图上已经生成但还没扩展下级的结点，用一个 CLOSE 表保存已经生成且已经扩展了下级的结点。OPEN 表中的结点按照代价函数值由小到大排列。算法先把初始结点放入 OPEN 表，从 OPEN 表中取出第一个结点(代价函数值最小的)，若该结点是目标结点则结束，否则把结点放入 CLOSE 表

中并扩展该结点的下级；检查扩展出的所有下级结点，如果结点不在 OPEN 表也不在 CLOSE 表中，则把下级结点放到 OPEN 表中并重新排序（图 5.12 步骤①、②中对 A，B 的扩展）；若该下级结点在 OPEN 表中，则说明找到了另一条到该下级结点的路径，根据最短的路径，重新计算该下级结点的代价函数值并重排 OPEN 表（观察图 5.12 步骤③、④中 F 结点代价函数值的变化）；若该下级结点在 CLOSE 表中，则到该下级结点又有新的路径，根据最短路径重新计算该下级结点的代价函数值并更新，把该下级结点从 CLOSE 表中移除，放到 OPEN 表中，并重排 OPEN 表，如此循环，直到找到目标结点或者 OPEN 表为空。

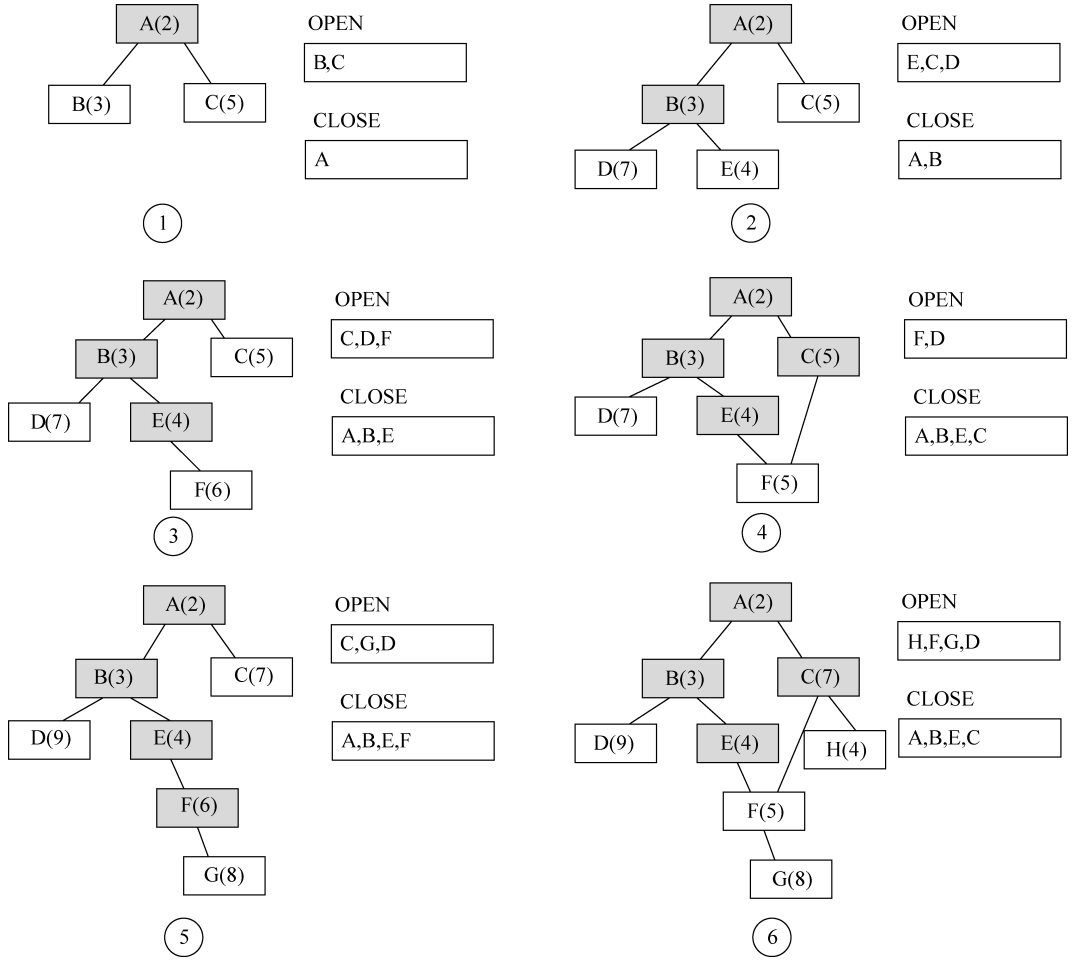


图 5.12　A 算法的实现

根据 A 算法的定义，无法保证一定能找到最优解，因此提出了 A* 算法，其定义是：在 A 算法中，如果代价函数 $f(x)=h(x)+g(x)$，$h^*(x)$是经过 x 结点到目标结点实际的最小代价，若 $h(x)\leqslant h^*(x)$，则称该 A 算法为 A* 算法且该算法能保证找到目标结点。换言之，A* 算法是 A 算法中满足 $h(x)\leqslant h^*(x)$的一类。

然而并不知道 $h^*(x)$是多少，只能根据具体问题来定义。以上述八数码问题为例，当规定 $h(x)$是位置错误的数字的数量，而 $h^*(x)$是实际把这些位置错误的数字全部移动到正确位置所需要的最少步数，显然 $h(x) \leqslant h^*(x)$，因此该算法为 A^*。又如图 5.13 所示的带障碍的最短路径求解问题，图中需要找到从出发点到目标点的最短路径(沿方格移动)，粗黑线是障碍物，由图可知，从出发点到目标点的实际最小代价是 $h^*(x)$，如果我们定义 $h(x)$的值是出发点和目标点之间的直线距离，由两点间直线最短可知 $h(x) \leqslant h^*(x)$，因此算法是 A^*。

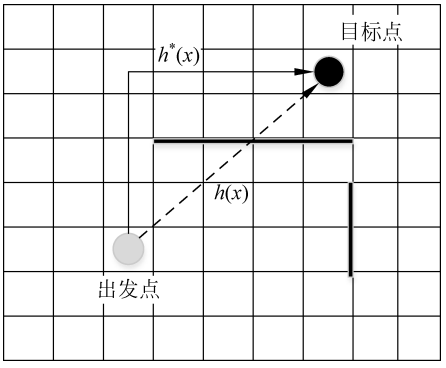


图 5.13　带障碍的最短路径求解

5.5　博弈搜索

博弈搜索与其他搜索最大的不同是需要有多个参与者，如国际象棋、围棋、游戏等，每个参与者都会按照有利于自己的方式进行选择，导致状态发生变化，这就是博弈。当搜索的状态空间中存在博弈时，通常的搜索算法往往不能适用，因为通常的搜索算法只考虑一种情况，如代价最小、距离最短等。在博弈搜索中不仅要考虑当前状态的选择，还要考虑对手可能会如何应对，因此构成博弈状态空间。

5.5.1　极大极小博弈

极大极小博弈假定对弈的只有两方，对弈双方使用同样的规则，必有一方有输赢，双方都以最有可能赢为目标行棋。对于我方来说，目的是要使得分最大而对方的目的是使他自己得分最大，等价于使我方得分最小，于是将我方称为 MAX 方，将对方称为 MIN 方。由于双方交替行棋，如图 5.14 所示。若有一个博弈的状态空间，假设状态都已经打分，结果值为 0 的状态代表 MIN 方胜利，反之代表 MAX 方胜利。MAX 方在选择结点时，会选择值大的，MIN 方

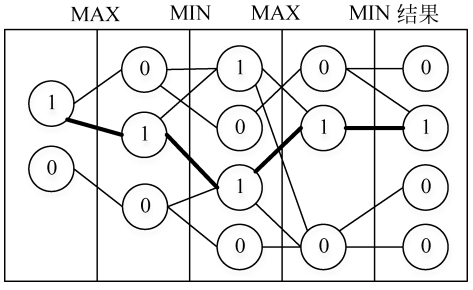


图 5.14　博弈状态空间

则会选择值小的。按照这一原则，当 MAX 方先选择时，可找出一条路径（图中加粗部分）确保 MAX 方胜利。

问题是如何对状态进行打分。这里采用从结果反推的思路：先对结果状态打分，再计算前一层的分数，若某一层是 MIN 方行棋后的结果，则此层中每一个结点的得分为其下级结点中最小的值，若此层是 MAX 方行棋后的结果，则此层中每一个结点的得分为其下级结点中最大的值，这样可以得到所有状态结点的分值，其过程如图 5.15 所示。图中用箭头表示值的传递方向，在图 5.15(a)中，由于上一步是 MIN 方行棋，所以每个结点取下级结点中最小的值，同理可推导出各级结点的得分。

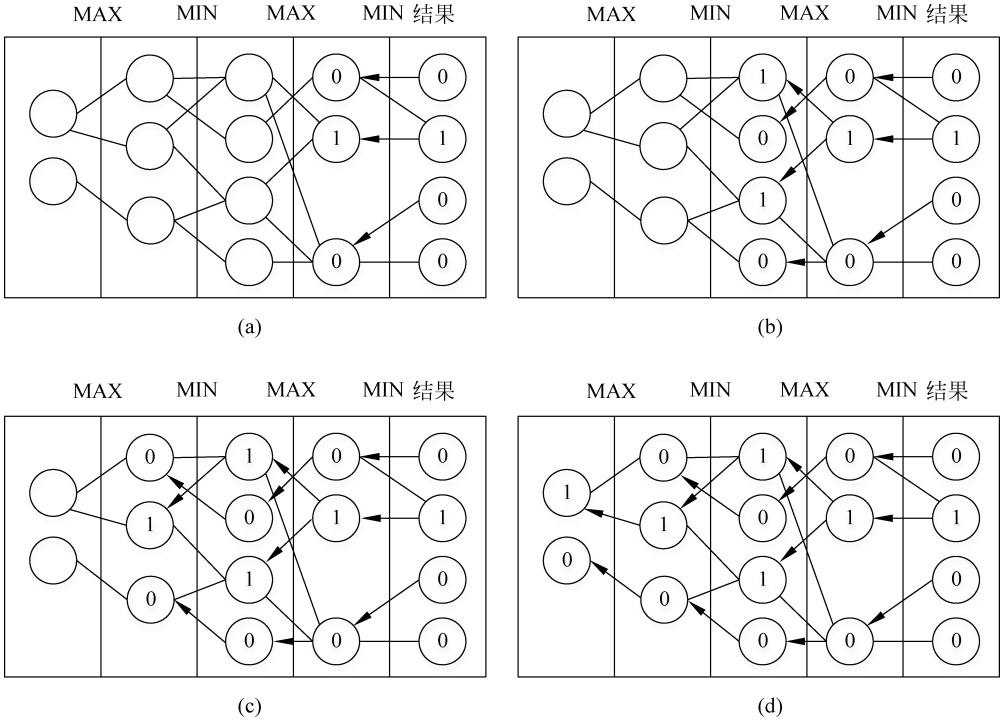


图 5.15　极大极小博弈算法状态评分生成过程

下面用极大极小算法来解决余一棋博弈问题。余一棋问题是有一副牌，开始的时候放成一摞，其中一方把这一摞分为两摞，分成的两摞牌数量不能相同，另一方再从所有分好的牌中选择一摞并继续拆分，直到某一方无法再进行拆分，则判这一方失败。例如，现有两摞牌，数量分别为 1 和 3（记为 1-3），则行棋的一方只能选择 3，拆分为 1 和 2，则变为 3 摞牌，数量为 1、1 和 2（记为 1-1-2），此时，另一方已经无法拆分，则先前那一方获胜。根据此规则，一个 7 张牌的余一棋问题，若让对手先行棋，则状态空间及评分见图 5.16。按此状态空间图，我方（MAX 方）只要每次选择最大的评分，则可以找到最有可能获胜的行棋步骤。

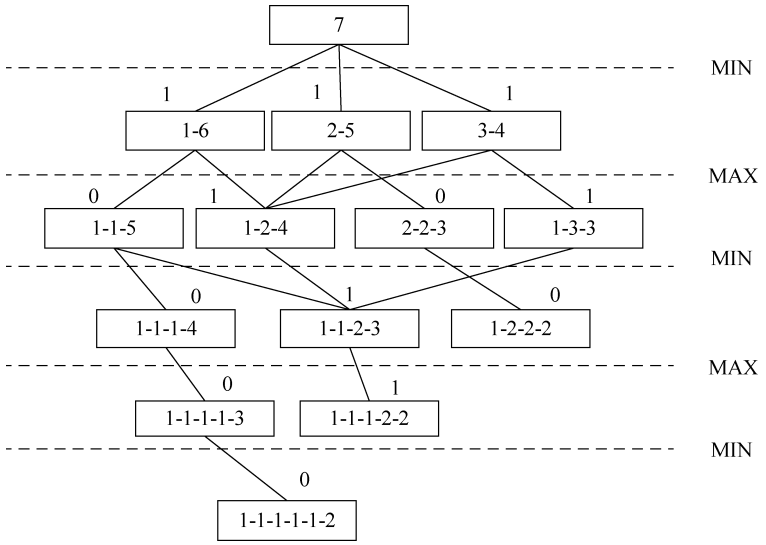


图 5.16 极大极小博弈算法解决余一棋问题

5.5.2 固定深度博弈搜索

虽然极大极小算法可以最大概率地找到赢棋路径，但在很多棋类博弈中，事实上是无法构建完整的博弈状态空间的。因为该算法要求穷举所有可能的状态，而在现有条件下穷举如此多的状态并进行搜索是不可能的，如象棋、围棋。因此，不可能用极大极小算法来处理此类问题。此时，可以采取一些其他的策略，如固定深度博弈搜索策略。固定深度博弈搜索策略只考虑当前棋局往前走固定步数的可能情况，这样根据最后步骤的状态评分，再利用极大极小算法倒推前面结点的评分。然而，对于叶子结点，由于不是最终结点，如何精确地评分呢？此时可以引入启发式搜索的思路，例如评估叶结点和最终获胜棋局的相似度，从而得出结点的评估得分。

5.5.3 α-β 剪枝算法

在搜索效率上，固定深度博弈搜索算法比极大极小搜索算法有一定的提升。但是，固定深度博弈搜索算法未限定搜索宽度，对于某些博弈来说每一步的选择非常多，总体计算量非常大。α-β 剪枝算法可以对一些无用选择进行有效删除，即删除部分看起来不好的结点，从而实现对状态空间的“剪枝”，缩小搜索空间，提高搜索效率。

图 5.17 是 α-β 剪枝算法示例，图 5.17(a)中根据已有结点，可以确定 4 是 MAX 方行棋时值最大的结点，未标注值的结点无论是何值也不影响 MAX 方的选择，所以这些分枝可以删除。同理，图 5.17(b)中根据已有结点，确定 7 是 MIN 方行棋时值最小的结点，所以未标注值的结点都可以删除，这就是 α-β 剪枝算法的原理。

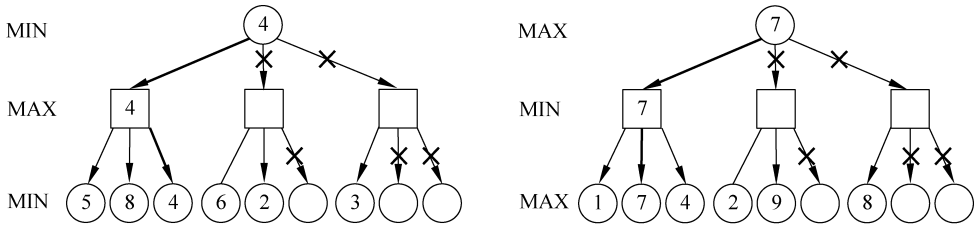


图 5.17　α-β 剪枝算法示例

5.5.4　博弈搜索的发展

在很长一段时间内 α-β 剪枝搜索算法是棋类算法的主流代表。1962 年，IBM 公司使用 α-β 剪枝算法获得了跳棋比赛的州冠军。1988 年，IBM 公司开始研究用于国际象棋的“深思”计算机，1997 年 IBM 公司的“深蓝”计算机战胜了人类国际象棋世界冠军卡斯帕罗夫（此时仍然主要使用 α-β 剪枝算法）。就在人们认为计算机不久后将在所有棋类领域超越人类时，在围棋领域计算机的水平却一直停滞不前，其主要原因有两点：一是围棋棋盘落点更多，更复杂，二是围棋的局面判断非常复杂，其他棋类（如国际象棋），进入残局后局面判断往往越来越简单（如只通过剩余棋子多少就可以判断），而围棋却不然，残局判断反而更加复杂。

2006 年，法国的一个计算机围棋研究团队将蒙特卡洛树搜索和信心上限决策方法相结合，使围棋程序的智能水平有了较大飞跃，但也只相当于业余围棋五段、六段的水平。随着深度学习方法的兴起，谷歌公司的 AlphaGo 将深度学习方法引入到蒙特卡洛树搜索中，从而在 2016 年战胜了人类围棋世界冠军并掀起了深度学习研究的热潮。

参考文献

[1]　李德毅. 人工智能导论[M]. 合肥：中国科学技术出版社，2018.
[2]　王万良. 人工智能导论[M]. 4 版. 北京：高等教育出版社，2017.
[3]　尼克. 人工智能简史[M]. 北京：人民邮电出版社，2017.
[4]　李开复，王咏刚. 人工智能[M]. 北京：文化发展出版社，2017.

扩展阅读

[1]　MAPLES. 蒙特卡洛树搜索（新手教程）[EB/OL]. (2018-11-01)[2020-05-31]. https://blog.csdn.net/qq_16137569/article/details/83543641.
[2]　遇到好事了. 人工智能导论 A* 算法推导证明[EB/OL]. (2020-01-14)[2020-05-31]. https://blog.csdn.net/qq_42549774/article/details/103979874.
[3]　王建雄. 博弈树启发搜索算法在五子棋游戏中的应用研究[J]. 科技情报开发与经济，2011，10.
[4]　宋宇，张正龙. A 算法在游戏寻径中的应用[J]. 科学咨询（科技・管理），2015，8.

习题 5

一、单项选择题

1. 下列关于搜索技术的描述错误的是(　　)。

A. 搜索技术是人工智能技术的重要组成部分,也是早期人工智能的主要基础技术之一

B. 搜索策略可分为盲目搜索策略和启发式搜索策略

C. 启发式搜索算法的关键是确定合适的启发函数

D. 博弈搜索和其他搜索最大的不同是其搜索的时间消耗更大

2. 下列有关状态空间的描述错误的是(　　)。

A. 在执行搜索时必须先生成完整的状态空间

B. 状态空间代表了搜索过程中可能遇到的各种状态

C. 通常可以用图表示状态空间

D. 状态空间中可能存在多个目标结点

3. 下列关于启发式搜索的描述正确的是(　　)。

A. 启发式搜索算法中,下级结点与目标结点的相似度越高则越应被优先搜索

B. 在八数码问题中,若定义启发函数的值为所有错牌与其正确位置的直线距离之和,则算法改为 A^* 算法

C. 深度优先搜索是一种启发式搜索算法

D. 启发式搜索算法不必考虑从初始结点搜索到备选结点的代价

4. 下列关于博弈搜索的描述正确的是(　　)。

A. 通常启发式搜索算法可以直接应用于博弈搜索

B. 极大极小博弈搜索算法可直接用于国际象棋博弈

C. AlphaGo 采用 α-β 剪枝算法战胜了人类围棋冠军

D. “深蓝”计算机主要采用 α-β 剪枝算法

二、多项选择题

1. 如果问题存在最优解,则下列算法中肯定能搜索到最优解的是(　　)。

A. 深度优先搜索　　B. 广度优先搜索

C. 有界深度优先搜索　　D. 启发式搜索

2. 用 A 算法求解带障碍最短路径问题(见图 5.13),下列启发函数中可保证算法是 A^* 的是(　　)。

A. 出发点到目标点的直线距离

B. 忽略所有障碍,先从垂直方向出发到目标点经过的方格数

C. 忽略所有障碍,先从水平方向出发到目标点经过的方格数

D. 以出发点和目标点为顶点确定的矩形包含的方格数

3. 下列关于极大极小博弈算法的描述正确的是(　　)。

A. MAX 方和 MIN 方都按照对各自最有利的方式行棋

B. 在对结点打分时，通常从下级结点开始用反推的方式计算上级结点的分值

C. 使用极大极小算法进行国际象棋博弈时，必须从最终的棋局结点向前反推

D. 极大极小算法通常和启发式搜索结合，解决较复杂的棋类博弈问题

三、判断题

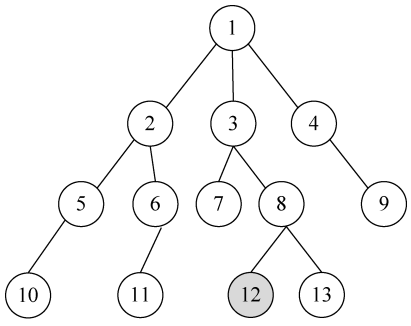
1. 把固定深度和广度优先搜索算法结合起来，可以解决同层结点过多的问题。（　　）

2. A 算法不一定能找到目标结点。（　　）

3. α-β 剪枝搜索算法是目前博弈搜索算法研究的热点。（　　）

四、简答题

1. 分别使用深度优先和广度优先搜索算法实现对如下树的搜索，并写出搜索顺序。



目标结点

2. 写出用 A^* 算法求解八数码问题的步骤。