全国大学生计算机博弈大赛培训教程

王静文　吴晓艺　编著

清华大学出版社

全国大学生计算机博弈大赛培训教程

王静文　吴晓艺　编著

清华大学出版社
北京

内 容 简 介

本书主要介绍计算机博弈的基本原理、常用的搜索算法、全国大学生计算机博弈大赛常规比赛项目(包括亚马逊棋、点格棋、六子棋、苏拉卡尔塔棋和西洋跳棋)的设计与实现及当前先进的搜索算法在计算机博弈中的应用等。为有兴趣参与计算机博弈程序设计的读者提供参考。

图书在版编目(CIP)数据

全国大学生计算机博弈大赛培训教程/王静文,吴晓艺编著.--北京:清华大学出版社,2013
ISBN 978-7-302-32531-4

Ⅰ.①全… Ⅱ.①王… ②吴… Ⅲ.①电子计算机－高等学校－教材 Ⅳ.①TP3

中国版本图书馆 CIP 数据核字(2013)第 108045 号

责任编辑: 刘　颖　赵从棉
封面设计: 傅瑞学
责任校对: 刘玉霞
责任印制: 刘海龙

出版发行: 清华大学出版社
网　　址: http://www.tup.com.cn, http://www.wqbook.com
地　　址: 北京清华大学学研大厦 A 座　　**邮　　编:** 100084
社 总 机: 010-62770175　　**邮　　购:** 010-62786544
投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn
印 装 者: 北京市清华园胶印厂
经　　销: 全国新华书店
开　　本: 185mm×230mm　　**印　　张:** 10.75　　**字　　数:** 226 千字
版　　次: 2013 年 7 月第 1 版　　**印　　次:** 2013 年 7 月第1 次印刷
印　　数: 1～3000
定　　价: 24.00 元

产品编号: 046083-01

前 言

1. 目的/目标

本书讨论计算机博弈程序(软件)的分析、设计、实现方法和过程。对计算机博弈的一些相关项目进行分析、实现,并能引导学生独立完成相关软件,为有兴趣参与计算机博弈程序设计的读者提供参考。

2. 预备知识要求

本书主要讲述计算机博弈及其实现的过程,读者需有基本的计算机语言知识,并能够编写简单的应用程序,对所学的语言并无特定的要求,例如 C、C++、Java 等语言均可作为具体实现的语言,本书中关于搜索和估值方面的内容均有相关的伪码,读者很容易将相关内容转换为自己所熟悉的语言,同时提供的示例从简单开始,逐步加深,便于学习。

3. 学习方法

对于学生来说重要的是如何学会自己动手完成相关程序或软件,而不是从书上或网上复制程序,本书在撰写过程中以分析设计为主,以代码实现为辅,通过对软件的分析,从算法的原理出发,将结构、流程与伪码相结合,引导学生独立完成相关软件。同时注重程序算法的效率,对效率从理论到实践进行研究。

通过软件工程的方法分析设计相关软件,使读者能从全局观念出发来设计完成软件,从实例中体会到从全局出发、以工程方法设计软件的重要性。

4. 内容提要

第 1 章介绍计算机博弈的一些基本情况,第 2 章介绍计算机博弈软件设计的基本原理和基本方法,第 3～7 章介绍全国大学生计算机博弈锦标赛中的一些项目的分析、设计和实现,包括亚马逊棋、点格棋、六子棋、苏拉卡尔塔棋和西洋跳棋等,并以软件结构结合伪码为主,兼顾不同计算机语言学习者的实现,部分示例采用目前使用量较大的 C++ 或 Java 语言来描述,并在表达中尽量使不同读者易于转换,第 8 章介绍目前全国大学生计算机博弈大赛部分项目的规则,附录为学生提供计算机博弈相关的网络资源,参考文献为写作过程中用到的参考资料,可以作为读者学习的极其有用的参考。

5. 错误

无论笔者有多少发现错误的技巧，总有一些错误漏网，而读者往往最能发现错误，如果读者发现任何认为是错误的地方，请提出纠正建议，然后发送电子邮件至 wang_jingwen@yeah.net，在此感谢读者的帮助。

6. 致谢

本书在撰写过程中得到很多人的支持和帮助，陈建、史孝兵、尹本立、冯建、张金苗、熊鹏、梁媛、吴梦妮等对各章的编写给出了不少很好的建议，并实现了具有相当水准的博弈软件，使本教材所涉及的内容都得以具体的实现，使得本书更加完善，同时感谢参与试读的同学，在学习期间抽出宝贵的时间来阅读本书，对本书的易读性、易用性提出了很多宝贵的意见。

编　者

2013 年 4 月

目 录 <<

第1章 <<

概　述

1.1　计算机博弈概述

计算机博弈(computer game)也称为机器博弈，最早来源于博弈论思想，博弈论最初主要研究象棋、桥牌以及各种与赌博、胜负相关的问题，是两个人在公平的对局中利用对方的策略变换自己的对抗策略，达到取胜的目的。目前，博弈论已经广泛应用于生物学、军事策略、计算机科学等领域，博弈论的思想与计算机相结合产生了计算机博弈，由于计算机游戏的题目简单、条件明确、周期短、见效快，因此，通常使用它来研究计算机的思维，让计算机学会像人的思维方式一样来“思考”问题，具备人一样的博弈能力，目前的计算机博弈主要研究针对人机对战的棋盘类游戏。

人工智能(artificial intelligence，AI)是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学，而人工智能最关心的两个问题是知识表示与搜索，也是计算机博弈中所要解决的问题，因而，人工智能中的相关技术在当前计算机博弈游戏中被广泛地应用，计算机博弈也逐渐成为人工智能中的一个重要的分支。

美国麻省理工学院著名教授 Claude Shannon 早在 1948 年就撰写了 Programming a Computer Playing Chess 一文，创立了计算机博弈的第一个里程碑。

1966 年麻省理工学院的 Mac Hack 6 开发小组就开发了第一个能与人进行对弈的计算机棋类游戏，由此开始了计算博弈成长之路。

1997 年 5 月 11 日，国际象棋世界冠军卡斯帕罗夫与 IBM 公司的国际象棋计算机“深蓝Ⅱ”(更深的蓝)的六局对抗赛落下帷幕。“深蓝Ⅱ”以 3.5∶2.5 的比分战胜了世界冠军卡斯帕罗夫，在第 6 局，仅下到 19 步卡斯帕罗夫就向“深蓝”拱手称臣。这一天也成为计算机人工智能的一个重要的里程碑，计算机博弈也得到越来越多的学者的重视。

1.2 国际计算机奥林匹克大赛

由国际机器博弈协会主办的国际计算机奥林匹克(computer olympic)大赛是目前世界上规模最大的计算机博弈大赛,自1989年在英国伦敦举办第1届计算机奥林匹克以来,目前已经进行了15届计算机奥林匹克,吸引了全世界众多计算机博弈爱好者的参加。我国于2008年在北京成功举办了第13届计算机奥林匹克,各国参赛队伍总数近100支,并吸引了"深蓝"之父许峰雄博士到场。

计算机奥林匹克大赛的比赛项目包括在我国较为普及的项目,如围棋(含九路围棋)、国际跳棋、五子棋、六子棋、中国象棋、日本将棋、西洋双陆棋、幻影围棋等。除此之外,同时还有一些国家的民间棋类,如:海克斯(Hex)、亚马逊(Amazons)、克拉巴(Clobber)、苏拉卡尔塔(Surakarta)、点格棋(Dots and Boxes)等近40种比赛项目。

1.3 全国大学生计算机博弈大赛

全国大学生计算机博弈大赛起源于中国计算机博弈锦标赛,自2006年以来,至今已经成功举办了6届中国计算机博弈锦标赛,比赛的项目由最早的中国象棋、围棋、九路围棋和六子棋4个项目,发展到目前已经达到中国象棋、围棋、九路围棋、幻影围棋、亚马逊棋、苏拉卡尔塔棋、六子棋、点格棋等10多个项目,参赛的代表队也由初始阶段的10个左右的代表队发展到目前的由30多个院校参加、100多个代表队近400名选手参与的规模。

从2011年开始,由中国计算机博弈锦标赛发展成全国大学生计算机博弈大赛暨计算机博弈锦标赛,并由教育部主办,人工智能协会承办,在北京科技大学成功举办了第1届全国大学生计算博弈大赛,2012年在东北大学举办的第2届全国大学生计算博弈锦标赛中,参赛的队伍已经达到170多个代表队,同时增加国际跳棋、爱恩斯坦棋和军棋等项目的比赛,为新参与该项活动的学校提供了一个更好的交流学习的机会,亚马逊棋、苏拉卡尔塔棋、六子棋、点格棋等参赛队伍均超过了20个,计算机博弈吸引了越来越多的大学生博弈爱好者参与。

计算机博弈软件的开发过程使计算机程序设计语言和算法等课程与实际项目有效接轨,使学生能更好地理解计算机语言、算法等课程的用途,将"枯燥"学习过程转换为"快乐"学习过程。

第 2 章 <<

计算机博弈基础

2.1 计算机博弈的基本原理

2.1.1 基本原理

计算机博弈的基本思想并不复杂，将计算机博弈的过程用“树”的方式表达出来称为博弈树，而计算机博弈实际上就是对博弈树上的节点进行估值和对博弈树进行搜索的过程。在博弈过程中站在其中的一方的立场上来构建一个博弈树，博弈树的根节点就是当前的棋局，而博弈树的子节点就是假设再行棋一步以后产生的棋局，再构建更下一步的棋局，直至某一深度或结束为止，由此构建出的博弈树通常是非常巨大的，往往无法直接通过构造博弈树而分出胜负，博弈程序的任务则是根据博弈树搜索出一种当前棋局的最佳走法。

通常计算机博弈需要满足以下条件：

(1) 双方对弈，对弈的双方轮流走步。

(2) 信息完备，对弈的双方所得到的信息是一样的，不存在一方能看到而另一方看不到的情况，每一方不仅知道对方走过的每一步棋，而且还能估计出对方未来可能走的棋。

(3) 零和，即对一方有利的局面对另一方肯定不利，不存在对双方均有利或均不利的情况，对弈的结果是一方赢而另一方输，或者是和棋。

下面从一个较为简单的游戏来介绍计算机是如何进行下棋的，井字游戏(Tic-Tac-Toe)是计算机博弈游戏中最为简单的例子之一，在如图 2-1(a)的棋盘中，双方轮流下棋，若其中一方在水平、垂直或斜线方向形成三个棋子连线，则获胜，下棋过程大致如图 2-1 所示。

图 2-1(h)中，白棋在右侧形成三个棋子连线，故白棋获胜。

针对井字游戏，在设计游戏智能过程中可按照以下基本原则进行：

(1) 如果下在该位置可以赢棋，那就下在该位置。

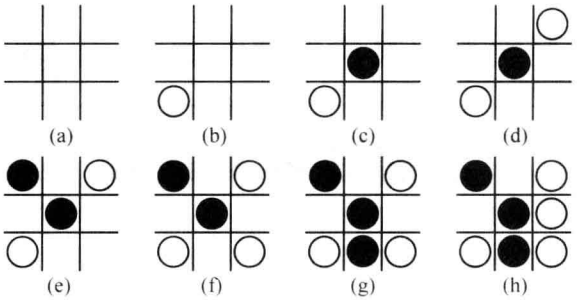

图 2-1 井字游戏

(2) 如果对手下在该位置可以赢棋，那就下在该位置。

(3) 如果中心位置空闲，那么下在中心位置要优于边上和角上的位置。

(4) 如果角上位置空闲，那么下在角上位置要优于边上位置。

(5) 如果只有边上位置有空闲，那只有下在边上位置。

图 2-1 中所示的过程只是下棋走法中的一种，如果计算机能够搜索出所有可能的下棋过程是最好的了，只要从中选择出能赢棋的走法在走棋过程中使用就可以了，那么，这种方法是否可行呢？至少有两个理由说明这种方法是不可行的。

(1) 如果一种棋，在棋盘上有 x 个可以下棋的位置，共有 y 步才能下完，那么下棋过程中最多可能的步子共有 x^y 种，对于上述井字游戏，如果不考虑对称的因素，那么在第一步下棋时，如果模拟到整个棋局结束，需要模拟的总的步数为 9!，共 362 880 种可能步数，考虑排除相应的对称因素外，如，在第一步下棋中只需要考虑中间点、四个角点中的一个和中间水平与中间垂直中的一个点即可，即：第一步棋只需要考虑三个可下棋的位置，以此类推来考虑第二步、第三步……下棋的位置，这样可以得到 255 168 多种下法，而对于国际象棋来说，可能的下法有 36^{40} 种，显然，以目前计算机的计算速度来计算，这种搜索方法是不可行的。

(2) “这就叫人工智能？这就是机器博弈？”，如果上述方法可行的话，这可能是你最可能说出的那句话，对于硬件崇拜者，这也许是个方法，而大多数设计者用一个优秀的搜索算法来解决这个问题会带来更大的快乐。

在实际应用中并不搜索所有的节点，仅仅搜索有价值的节点，并对它们进行估值，如何对节点进行合理的估值并设计相应的估值函数是计算机博弈中的一个重要环节，在后续章节中会专门对估值函数的设计方法进行讨论。

为方便表达游戏的状态，通常使用树或图来表达，图 2-2 表示了一种博弈树，树的根部为起始状态，move A、move B 等表示可以选择的下棋位置，player1、player2 表示在树的该层走棋的一方，W 代表赢棋，L 表示输棋，进行初始化之后，player1 可选择 A、B、C 和 D 位置下棋，然后轮到 player2 下棋，以此类推，向下进行的深度称为层次，在计算机博弈中通常

使用 ply 来表示。图中,L 表示 Lost,W 表示 Win,不同游戏者所处的层次分别用圆圈和矩形表示。

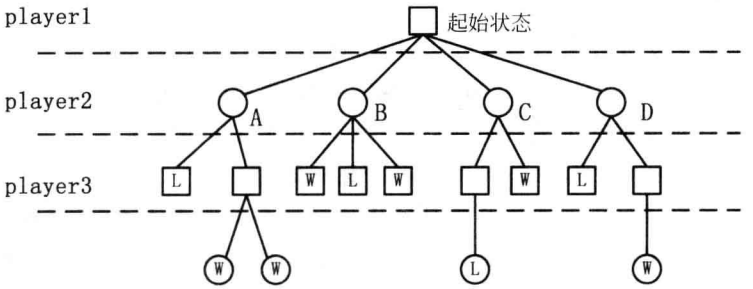

图 2-2 博弈树

那么,我们现在所要做的工作就是要找到获胜的节点,为更好理解搜索的过程,先介绍一下博弈树中的一些常用的概念:

- 分支因子(branching factor,b):从游戏起始出发游戏者可以移动到的位置,例如,井字游戏的分支因子为 9(当游戏开始时,下棋方共有 9 个位置可以选择)。
- 层次(ply):博弈树的层次,游戏者通过下棋而进入博弈树的下一层次。
- 深度(depth,d):在博弈树中向下搜索的层次称为深度(或搜索深度),井字游戏的搜索深度一般为 6～7,而国际象棋通常要达到 40。

在实际搜索过程中通过穷举法搜索得到制胜的下法在大多数游戏中是不现实的,通过选择合适的算法实现最优搜索则是计算机博弈游戏中的一个重要的环节。

计算机博弈或人机对弈一般来说需要满足以下条件:

(1) 某种棋局可以在博弈程序中以一定的方式表示出来,并能使程序获得当前棋局的状态。

以上述的井字棋为例,我们可以使用一个二维的数组来表示棋盘,比如采用整型的二维数组 board[3][3]来表示棋盘,也可以使用一维数组来表示棋盘,如采用整型的一维数组 board[9],9 个变量分别表示棋盘中的 9 个位置。当其值为 0 时可以表示当前位置没有棋子,当其值为 1 时可以表示当前位置为“X”方,而当其值为－1 时可以表示当前位置为“O”方,这样,我们就可以将棋盘的当前状态完整地表示出来,在不同的棋种中棋局的表示方法略有不同,主要是根据相关棋局的特征来确定棋局的具体表示方法。

(2) 博弈过程在计算机可判别的规则中进行。

在使用二维数组的状态表示中,如果当前位置的值为 0,表示当前位置为空,则下棋方可以在该位置下棋。如果水平、垂直或斜线方向有三个同值的数据,则表示该方已经赢棋,这些规则都是计算机可以识别的规则。

(3) 博弈程序具有从所有可行的走法中选择最佳走法的技术。

例如在图 2-1 的井字棋的游戏过程中,图 2-1(f)中的白棋有两个位置可以直接赢棋,那

么，这两个位置就可以作为最佳位置进行选择，计算机可以通过判断这两个位置下棋后能赢棋来确定在该位置可以直接下棋，在没有直接可以获胜的情况下需要评估所下位置的价值来确定具体下在哪个位置。

(4) 博弈程序应具有适当的估值方法，用于评估当前局面的优劣。

例如在上述的井字棋中，我们可以通过我方可能胜利的线路总数减去对方可能胜利的线路总数来确定当前位置的价值，通过具体价值来评估当前的局面。

(5) 具有适合的运行界面以准确地表达当前局面。

在目前的大多数计算机博弈比赛中都要求比赛软件以可视化形式来表示当前局面，可以根据不同的语言采用不同的界面制作工具来完成。例如使用 C++ 作为编程语言时，通常可以使用 Visual C++ 的 MFC(微软基础类)来制作界面，如果使用 Java 语言则可以使用 AWT(abstract windowing toolkit，抽象窗口工具包)或 SWING 来制作界面，运行界面必须符合比赛的基本要求。

2.1.2 计算机博弈的搜索方法

在计算机博弈中，博弈过程可能产生惊人庞大的搜索空间，通常这个搜索空间是无法使用穷举搜索(遍历整个空间，找到获胜的走法就可以了)来完成，要搜索这些庞大而复杂的空间需要使用相应的技术来判断备选状态，探索问题空间，人类解决问题是以一些判断性规则为基础的，这些规则使我们的搜索仅寻找在状态空间中最“有效”的一部分，这些技术和规则被称为启发(heuristic)，启发式算法依赖于评估逻辑表达式，从而降低了搜索空间的复杂度。

启发就是有所选择地搜索问题空间的策略，它引导搜索沿高成功概率的路线前进，避免做多余的或明显愚蠢的行为，启发也是人工智能研究的中心问题之一，是计算机博弈的基础。

启发式搜索就是在状态空间中的搜索过程中对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到目标。这样可以省略大量无谓的搜索路径，提高了效率。在启发式搜索中，对位置的估价是十分重要的。采用了不同的估价可以有不同的效果。

在计算机博弈程序设计过程中，通常需要从博弈树中搜索出最佳位置，搜索的方法通常有两种，一种是深度优先的搜索方法，一种是宽度优先的搜索方法。

在深度优先搜索中，搜索过程是尽可能地向搜索空间的更深层次进行，只有在找不到状态(或最佳位置)时，才会搜索它的兄弟节点，对于图 2-3 来说，深度优先的搜索顺序是 A、B、E、K、S、L、T、F、M、C、G、N、H、O、P、U、D、I、Q、J、R。

宽度优先的搜索方法与深度优先的搜索方法正好相反，宽度优先的搜索方法是一层一层地搜索空间，只有在给定的层上不再存在要搜索的状态时，算法才转移到下一个更深的层次，对图 2-3 而言宽度优先的搜索顺序是 A、B、C、D、E、F、G、H、I、J、K、L、M、N、O、P、Q、R、S、T、U。

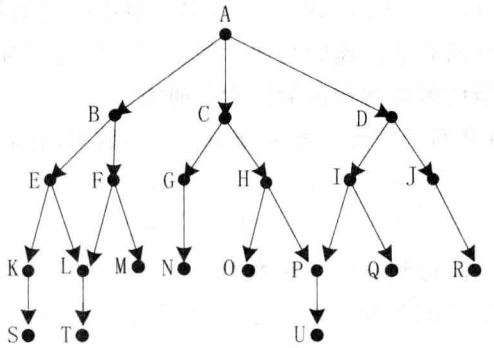

图 2-3 深度优先搜索和宽度优先搜索示例图

宽度优先搜索在搜索分析第 $n+1$ 层之前必然先分析第 n 层的所有节点，因此，宽度优先搜索找到的目标节点的路径总是最短的，对于一个有简单解的问题，宽度优先的搜索方案能够保证发现这个解，但是，如果存在一个不利的分支因子，也就是各个状态都有相对很多个后代，那么组合爆炸可能使算法无法在现有可用内存的条件下找到解。

深度优先的搜索方法可以迅速地深入搜索的空间，如果已知解路径很长，那么深度优先搜索不会浪费时间来搜索大量“浅层”状态。但深度优先搜索可能在深入空间时，失去了达到目标的更短路径，或者陷入不能达到目标的无限长路径。通常对于具有很多分支的空间，显然深度优先搜索方法具有更高的效率。

平衡深度优先搜索和宽度优先搜索的一个很好的折中方法是对深度优先搜索使用一个界限，一旦搜索在某个层次以下，深度界限就强制停止对这条路径的搜索，形成了一种对某个搜索空间的“横扫”，这种思想产生的算法弥补了深度优先搜寻和宽度优先搜索算法的不足，在 1987 年，Korf 提出了迭代加深的深度优先搜索算法，即对空间进行深度界限为 1 的深度优先搜索，如果找不到目标，再进行深度界限为 2 的深度优先搜索，这样继续下去，每次将深度界限加 1，在每一次迭代中，算法执行一次当前深度范围内的完全深度优先搜索，在两次迭代之间不保存任何状态空间信息，这种算法虽然弥补了深度优先搜索和宽度优先搜索算法的各自缺点，但由上面的过程可以看出，其搜索过程所需要搜索的量也会在一定程度上增加。

2.1.3 递归

递归(recursive)是数学和计算机科学中的一个基本概念，广义递归的定义是一种直接或间接引用自身的定义方法，在编程语言中，递归可以简单定义为自我调用的程序，递归程序不能总是自我调用，否则程序永远不会终止，因此，一个合法的递归应包含两部分：基础情况和递归部分，基础情况是指递归对象的表现形式，而递归部分是指递归的条件和方法，在递归的条件中必须有一个终止条件以避免递归进入无限循环。

递归算法就是通过解决同一个问题的一个或多个更小的实例最终解决一个大问题的算法，即直接或间接调用自身的算法。递归与博弈树的以递归方式定义的结构的研究是互相重合的，研究递归有助于帮助解决博弈树的搜索的研究。

以递归定义的一个典型例子是斐波那契(Fibonacci)数列，它的定义可递归表示为

$$\begin{cases} F_0 = 0, & F_1 = 1 \\ F_n = F_{n-1} + F_{n-2}, & n > 1 \end{cases} \tag{2-1}$$

根据这一定义，我们可以得到一个无穷数列 0,1,1,2,3,5,8,13,21,34,55,…，这个数列就称为斐波那契数列，斐波那契数列产生于 12 世纪，一直到 18 世纪才由 A. De. Moivre 提出了它的非递归形式，从 12 世纪到 18 世纪之间，人们只能以递归的形式来计算斐波那契数列。斐波那契数列的非递归形式可以用如下公式来计算：

$$F_n = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$$

其中：$\phi = \frac{1}{2}(1+\sqrt{5})$，$\hat{\phi} = 1-\phi$。

根据斐波那契数列的递归定义，可以很自然地写出计算 F_n 的递归算法，将该过程设计成一个函数可以用伪码描述如下：

```
1  function long Fib(long n)
2  {
3      if(n<=1)
4          return n
5      else
6          return Fib(n-2)+Fib(n-1)
7  }
```

函数 Fib(n)中又调用了 Fib(n－2)和 Fib(n－1)，这种在函数体内调用自己的做法称为递归调用，包含递归调用的函数又称为递归函数，编译器是利用系统栈来实现函数的递归调用，系统栈是实现递归调用的基础。

我们可以用递归树的方法来描述上述函数 Fib 执行过程中的调用关系，假设我们调用 Fib(5)，Fib(5)的执行过程可以用图 2-4 所示的递归树来表示。由图 2-4 可见，Fib(5)计算的执行过程为：Fib(5)需要分别调用 Fib(4)和 Fib(3)，Fib(4)需要分别调用 Fib(3)和 Fib(2)，Fib(3)又需要分别调用 Fib(2)和 Fib(1)……，其中 Fib(0)被调用了 3 次，Fib(1)被调用了5 次，可见使用递归在许多工作上是重复的，当然这也是费时的。

在上面的递归中，我们也可以看到上述的递归过程符合递归的两个基本条件：每一次递归调用必须包含更小的参数值，同时当 $n \leqslant 1$ 时终止了递归调用，即满足了递归调用必须有一个终止条件。

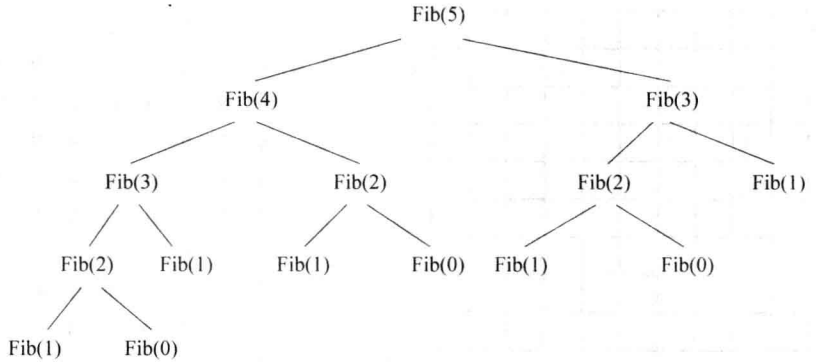

图 2-4 计算 Fib(5)的递归树

2.1.4 回溯

回溯(backtracking)是一种系统搜索问题解的方法，在搜索过程中先列出所有候选解，然后依次检查每一个候选解，在检查完所有或部分候选解之后，即可找到所有或部分候选解，回溯法是常用的搜索候选解的方法。

为了实现回溯，首先要定义一个解的空间，在这个空间中至少包含问题的一个解，一旦定义了解的空间，在这个空间就可以按照深度优先的方法从开始节点进行搜索。回溯算法的实现步骤如下：

(1) 定义一个解空间，它包含问题的解。

(2) 用适合搜索的方法组织该空间。

(3) 用深度优先的方法搜索该空间，利用界定函数避免移动不可能产生解的子空间。

回溯算法的特点是在搜索过程中就可能产生解空间，在搜索期间的任何时刻只保留从开始节点到当前节点的路径，回溯算法求解过程的本质就是遍历一棵“状态树”的过程，这棵状态树隐含在遍历过程中，回溯法的基本思想是从一条路往前走(沿着一个方向前进)，能进则进，不能进则退回来，换一条路再试，直到找到符合条件的位置就可以了，下面用迷宫问题来说明回溯算法。

迷宫是一个矩形区域，它有一个入口和一个出口，迷宫的入口位置为左上角，迷宫的出口位置为右下角，迷宫内部不能穿越障碍物或墙，障碍物沿着行和列放置，它们与迷宫的边界平行，图 2-5(a)所示为 10×10 的迷宫，图 2-5(b)是迷宫的矩阵表示。

为了能够更加简单描述回溯过程，用图 2-6 的 3×3 的迷宫来说明搜索的过程，其中位置数据用 maze(n,m)表示，图 2-6(a)是要搜索的迷宫，搜索过程从节点 maze(1,1)开始，它是一个活节点或 E-节点(expansion node)，为了避免再次走到该节点，将 maze(1,1)置为 1，从这个位置可以移动到 maze(1,2)或 maze(2,1)两个位置，因为此时两个位置的值都为 0，