Computer Science and Application 计算机科学与应用, 2019, 9(7), 1398-1405
Published Online July 2019 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2019.97157

Hans汉斯

Application of Forest in the Analysis of Recursive Algorithm

Yuekun Liu, Sunying Zhuan, Beining Xu, Bihe Shen

School of Information and Control Engineering, Qingdao University of Technology, Qingdao Shandong
Email: m15692307278@163.com

Received: Jul. 4th, 2019; accepted: Jul. 18th, 2019; published: Jul. 25th, 2019

Abstract

This paper presents a method of recursive algorithm analysis by forest. This method maps recursive logic to forest. Tree is used as processing unit. The recursive structure of program is analyzed and tree nodes are defined. Tree is established by searching and calling state table. The tree is not limited to binary tree. According to practical problems, it can be composed of single branch tree, binary tree, multi-branch tree or even multi-tree. According to the implementation of the program, many trees need to jump between trees. According to the comparison between the tree and the invocation state table, the analysis is comprehensive, and the analysis structure is clear, the results are accurate, and the application is extensive, which effectively solves the difficulties in the analysis of the results of the recursive algorithm.

Keywords

Forest, Logistic Map, State Table, Recursive Algorithm, Search, Backtrack

森林在递归算法分析中的应用

刘月锟，颛孙盈，徐贝宁，沈碧荷

青岛理工大学信息与控制工程学院，山东 青岛
Email: m15692307278@163.com

收稿日期：2019年7月4日；录用日期：2019年7月18日；发布日期：2019年7月25日

摘　要

本文提出用森林进行递归算法分析的方法，该方法将递归逻辑映射到森林，具体处理上以树为处理单元，对程序递归结构分析，定义树节点，通过搜索建立树和调用状态表，所建立的树不局限于二叉树，根据

文章引用: 刘月锟, 颛孙盈, 徐贝宁, 沈碧荷. 森林在递归算法分析中的应用[J]. 计算机科学与应用, 2019, 9(7): 1398-1405. DOI: 10.12677/csa.2019.97157

实际问题可以是单分支树、二叉树、多叉树，甚至多棵树组成的森林，到达叶子节点进行回溯，根据程序执行情况，多棵树需要树间跳转；依据树和调用状态表的对照，既分析全面，又使分析结构清晰、结果准确、适用广泛，有效解决了递归算法结果分析中的难题。

关键词

森林，逻辑映射，状态表，递归算法，搜索，回溯

1. 引言

递归算法是一种直接或者间接地调用自身算法的过程，递归算法把大问题转化为规模缩小的同类问题的子问题，将一个复杂问题逐步简化并最终转化为一个最简单的问题，最简单问题的解决就意味着整个问题解决。递归算法对解决一大类问题是十分有效的，它往往使算法的描述简洁而且易于理解，为一些问题提供了最简单的解决方案[1]。递归算法简洁、规整，虽然在整体结构上易于理解，但在分析递归运行情况时，总给人不可捉摸、难以理解的感觉，主要原因是递归调用实现机制太抽象，递归函数层层跳转、回溯，使人们对递归算法的分析难以把握。

多年来，很多学者对递归算法的分析进行了深入研究，但还存在一些不足，文献[2] [3] [4]在讨论递归算法结果时，一般根据函数调用的过程直接分析，对于简单的问题容易实现，但对复杂递归函数分析就困难重重，分析过程过于抽象，极易出错；文献[5] [6]使用递推函数对递归算法分析研究，这种方法不具有通用性，而且一些递归算法难以转换为递推式；文献[7] [8] [9]提出了用递归树研究递归的方法，将递归与树进行映射，便于操作，但分析时局限于单棵树，注重于函数调用关系的研究，而递归调用中有时是多棵树组成的森林，递归树在处理这些问题时显得不足，且在树上标注，处理结构不清晰。

本文将森林引入递归算法的分析处理过程，解决递归算法分析中存在的问题，通过建立递归与森林的逻辑映射关系，根据对程序递归结构分析，判断树结构，由入口函数，建立搜索树，所建立的树不仅限于二叉树，根据实际问题可以是单分支树，可以是二叉树、多叉树，甚至多棵树，到达叶子节点进行回溯，将程序结构映射为森林，并根据执行情况建立状态调用表，既分析全面，又使分析结构清晰、结果准确，有效解决了递归算法结果分析中的难题。

2. 树在递归分析中的应用

递归算法整个过程分为两部分：递推和回归，递归调用由多个子递归构成，递归的结构和处理与树的特点很相似，树由多个子树构成，对树的遍历就是搜索和回溯的过程，所以在递归算法分析上可以将递归结构映射到树型结构上，可以用树结构进行递归算法的分析。实际问题中，存在一些分支语句中含有递归的情形，这时需要分成多棵树，多棵树组成逻辑上的森林。

森林为 M 棵互不相交的树的集合，表示为 $F = \{T_1, T_2, \cdots, T_m\}$ [10]，$T_1, T_2, \cdots, T_m$ 为组成森林的树，可以将递归算法逻辑映射为森林，具体处理上以树为处理单元，建立标准的处理流程。

递归算法映射到树时，在分析中对树的遍历采用先根遍历，边界条件即为树的叶子节点，下面以一简单递归类型的程序结构为例，说明使用树对递归算法分析的过程，在第 3 部分采用具体实例进行详细说明：

```
(1)   void Recursive (int m, int n)
{
(2)      if (m==0)
(3)            return ;
(4)      if (n==0)
   {
(5)       a= Recursive (m-1,1);
(6)      return a;
           }
(7)   else
   {
(8)      b= Recursive (m-1,n);
(9)      printf("%d %d",m,n);
(10)     c= Recursive (m,n-1);
   }
}
```

2.1. 分析程序结构

首先分析程序的整体结构，函数 Recursive 为递归函数，作为树的基础节点，通过深度搜索对树节点进行扩展，每次递归函数执行过程的节点作为树的一个节点类型；其次要分析递归函数在程序结构中所处位置，语句(5)的递归函数与语句(8)、(10)的递归函数处于不同的语句结构体中，执行到语句(5)时程序跳转到另外一棵树，所以要分成两个不同的树进行处理，这两棵树构成森林；第三，语句(8)、(10)在一棵树中进行处理，处理过程扩展为二叉树，b 和 c 处于二叉树的不同分支；第四，两个分支中的其他语句，如：程序中的处理语句(9)在 b 执行到叶子节点回溯到 c 的过程中执行。

程序中语句(2)、(4)为边界，搜索到叶子节点时执行，执行语句(4)时跳到另一棵树执行搜索；语句(1)为程序入口，可看作根节点。

2.2. 建立树型图

根据对程序结构的分析，明确树的总体结构，本示例为两棵树构成的森林，每棵树为二叉树，入口函数作为树的根节点，从根节点进行深度搜索，由搜索情况建立树节点，到叶子节点进行回溯。多棵树组成的森林，在搜索树节点时，根据实际情况进行树间跳转，根据搜索情况绘制树型图，标明节点序号。

2.3. 建立状态调用表

树只建立节点，根据节点扩展情况，对应树的节点建立对应状态调用表，调用表反映状态变化及其他语句块执行，一方面反映程序整体全貌，另一方面使分析更加清晰、直观；状态调用表主要内容有：节点编号、节点名、节点动作、节点值等，根据树的搜索情况建立该表，回溯节点前加 H 代表回溯动作，回溯到根节点，递归算法分析结束，结果为递归算法处理结果。

3. 实例分析

3.1. 汉诺塔问题

有 A，B，C 三根柱子，A 柱上按大小顺序从下往上摞着 n 片圆盘，现在要将这些圆盘从 A 柱移至 C

柱，并保持上小下大的顺序。移动规则如下：每次只能移动一个盘，大盘不能放在小盘上。

程序主体结构：

```
void hanoi (int n,char a,char b,char c)
{
    if(n==1)
      {
          printf(“%c-->%c”, a,c);    //1
      }
    else
      {
           hanoi(n-1,a,c,b );
           printf(“%c-->%c”, a,c);    //2
           hanoi(n-1,b,a,c);
          }
}
```

递归分析：

将递归函数作为树的节点，程序中有两个递归函数分别为树的左右分支，左分支：hanoi(n-1,a,c,b)，右分支：hanoi(n-1,b,a,c)，其他处理语句单独列明：如：1 语句在 n 等于 1 时执行，为出口语句，2 语句在右分支前执行，在下述实例中只在 4、5、7 节点前执行。

假如初始调用为 hanoi(3,'A','B','C')，则调用树结构如图 1 所示，执行过程调用状态如表 1 所示。

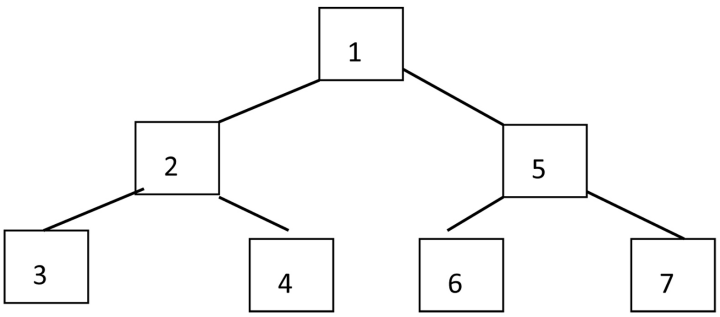

Figure 1. The tree structure of the Tower of Hanoi
图 1. 汉诺塔执行的树结构

Table 1. The Tower of Hanoi call status table on execution procedure
表 1. 汉诺塔执行过程调用状态表

节点	H(n,a,b,c)	动作	节点值	备注
1	H(3,’A’,’B’,’C’)			
2	H(2,’A’,’C’,’B’)			
3	H(1,’A’,’B’,’C’)	‘A’→’C’		
H2	H(2,’A’,’C’,’B’)			回溯到 2 节点，搜索 4 节点
		‘A’→’B’		执行语句 2，父节点为 2
4	H(1,’C’,’A’,’B’)	‘C’→’B’		

Continued

H2	H(2,’A’,’C’,’B’)		回溯节点 2，2 节点搜索完毕
H1	H(3,’A’,’B’,’C’)		回溯到 1 节点，搜索右分支
		‘A’→’C’	在 5 节点前执行
5	H(2,’B’,’A’,’C’)		
6	H(1,’B’,’C’,’A’)	‘B’→’A’	
H5	H(2,’B’,’A’,’C’)		回溯节点 5
		‘B’→’C’	执行语句 2，父节点为 5
7	H(1,’A’,’B’,’C’)	‘A’→’C’	父节点为 5
H5	H(2,’B’,’A’,’C’)		回溯节点 5，节点 5 搜索完毕
H1	H(3,’A’,’B’,’C’)		回溯到根节点，处理完毕

3.2. 2018 noip 普及组试题三(3)

```
int findans(int n,int m)
{
    if(n==0) return m;
    if(m==0) return n%3;
    return findans(n-1,m)-findans(n,m-1)+findans(n-1,m-1);
}
```

将 return findans(n-1,m)-findans(n,m-1)+findans(n-1,m-1)，修改为：

```
f1= findans(n-1,m)
    f2= findans(n,m-1)
    f3= findans(n-1,m-1)
    return f1-f2+f3
```

调用实例 findans(1,2)

树为三叉树，第一分支为 findans(n-1,m)，第二分支为 findans(n,m-1)，第三分支为 findans(n-1,m-1)，函数简称为 f(n,m)。

语句 f1-f2+f3 在第三分支后执行，return m，return n%3 为出口语句。

调用树结构如图 2 所示，执行过程调用状态如表 2 所示。

结果为：3。

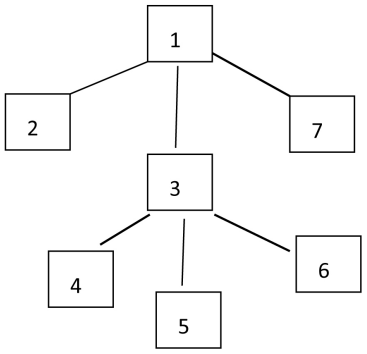

Figure 2. The tree structure of 2018 noip popularization group
图 2. 2018 noip 普及组树结构

Table 2. The call status table of 2018 noip popularization group
表 2. 2018 noip 普及组调用状态表

节点	f(n,m)	动作	节点值	备注
1	f(1,2)			
2	f(0,2)	f1 = 2	f1 = f(0,2) = 2	
H1	f(1,2)			回溯，搜索第二分支
3	f(1,1)			
4	f(0,1)	f1 = 1	f1 = f(0,1) = 1	
H3	f(1,1)			回溯，搜索第二分支
5	f(1,0)	f2 = 1	f2 = f(1,0) = 1	
H3	f(1,1)			回溯，搜索第三分支
6	f(0,0)	f3 = 0	f3 = f(0,0) = 0	
		f2 = f1 − f2 + f3 = 0		
H3	f(1,1)		f2 = 0	回溯节点 3，搜索完毕
H1	f(1,2)			回溯，搜索第三分支
7	f(0,1)	f3 = 1	f3 = f(0,1) = 1	
		f1 − f2 + f3 = 2 – 0 + 1 = 3		
H1	f(1,2)		3	回溯到根节点，结束

3.3. 阿克曼函数

阿克曼函数定义：

n+1　m=0,n>0

A(m,n)=A(m-1,1) n=0,m>0

A(m-1,A(m,n-1)) n>0,m>0

程序主体结构：

```
int Ackerman(int m, int n)
{
    if (m==0)
        return n + 1;
    if (n==0)
        return Ackerman(m - 1, 1);
    else
        return Ackerman(m - 1,Ackerman(m, n - 1));
}
```

因为递归函数是参数，语句 return Ackerman(m - 1,Ackerman(m, n - 1))；修改为：

```
a= Ackerman(m, n - 1);
b= Ackerman(m - 1,a);
return b;
```

因为 if 语句含有递归函数，所以分为两棵树，树 1 称为 t1，树 2 称为 t2。t1 有两个分支，左分支为

Ackerman(m, n - 1)，右分支为 Ackerman(m - 1,a)，函数简称为 A(m,n)。t2 有 1 个分支为 Ackerman(m - 1, 1)。

入口调用函数为：A(1,2)

t1 调用树结构如图 3 所示，执行过程调用状态如表 3 所示，跳转后的分支树 t2 的调用树结构如图 4 所示，执行过程调用状态如表 4 所示。

结果为：4。

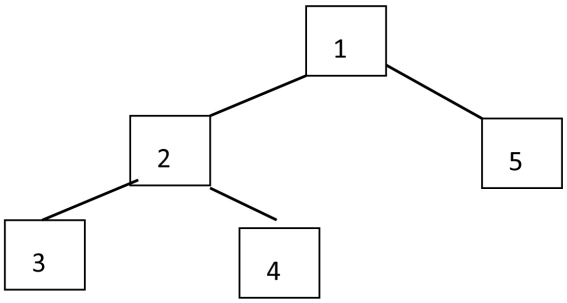

Figure 3. The tree structure of t1
图 3. t1 树结构

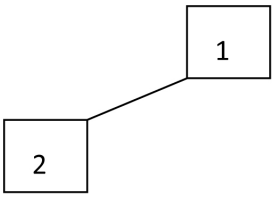

Figure 4. The tree structure of t2
图 4. t2 树结构

Table 3. The call status table of T1
表 3. T1 树调用状态表

节点	A(m,n)	动作	节点值	备注
1	A(1,2)			
2	A(1,1)			
3	A(1,0)	执行 t2 树	2	接收 t2 树返回值
H2	A(1,1)		a = 2	回溯节点 2
4	A(0,2)	n + 1 = 3	b = 3	
H2	A(1,1)		b = 3	回溯节点 2
H1	A(1,2)		3	回溯节点 1，搜索右分支
5	A(0,3)	n + 1 = 4	b = 4	
H1	A(1,1)		4	回溯根节点 1，搜索结束

Table 4. The call status table of T2
表 4. T2 树调用状态表

节点	A(m,n)	动作	节点值	备注
1	A(1,0)			
2	A(0,1)	n + 1	2	
H1	A(1,0)		2	回溯节点 1，t2 树执行完毕

4. 结论

当前对递归算法的分析方法主要有：根据函数调用的过程直接分析[2] [3] [4]，使用递推函数分析[5] [6]，这两种方法分析递归算法的局限性是显而易见的，在分析稍复杂的递归算法时是无能为力的；用递归树分析递归过程[7] [8] [9]，分析时局限于单棵树、状态调用过程不清晰；本文通过对递归不同类型典型实例的分析，将递归算法逻辑映射到森林，用树结构处理递归函数结果。给出了用森林分析递归函数的基本流程，通过分析程序结构、搜索建立分析树和状态变化对照表，到达叶子节点进行回溯，既分析全面，又使分析结构清晰、结果准确，有效解决了递归算法结果分析中的难题。

参考文献

[1] Prata, S. C Primer Plus [M]. 第 6 版. 姜佑, 译. 北京: 人民邮电出版社, 2016: 256-282.
[2] 晏素芹. 递归算法的教学方法探讨——以 C 程序设计为例[J]. 福建电脑, 2018, 34(8): 170-171.
[3] 方娇莉, 潘晟旻, 刘明. 程序设计微课教学设计方法研究——以递归为例[J]. 计算机教育, 2016(5): 116-120.
[4] 吴晓晨. 递归程序设计教学方法的研究[J]. 天津科技, 2017, 44(1): 69-71.
[5] 陈瑞环, 杨庆红, 姚兴. 使用递推解决递归问题的研究与应用[J]. 计算机应用与软件, 2011, 28(3): 186-187.
[6] 周法国, 韩智, 高天. 递归算法设计思想与策略分析[J]. 软件导刊, 2017, 16(10): 35-38.
[7] 张建波. 一种将递归过程转换为非递归过程的方法研究[J]. 计算机教育, 2017(8): 139-142.
[8] 黎远松. 基于树的递归算法分析技术[J]. 四川理工学院学报(自然科学版), 2012, 25(4): 50-51.
[9] 张俊. 基于递归树的递归调用分析[J]. 实验室研究与探索, 2010, 29(3): 83-87.
[10] 严蔚敏, 吴伟民. 数据结构(C 语言版) [M]. 北京: 清华大学出版社, 2009: 118-155.