拓扑排序和强连通算法在源代码分析中的应用①

Application of Topological Sort and Strongly Connected Components in Source Analysis

李义军　任子真　(沈阳化工学院 计算机学院 辽宁 沈阳 110142)

摘　要： 缺少文档的开源项目和文档不完整的开源项目，分析其源代码是了解其运行机理的主要方法，本文提出一种基于函数分析顺序的分析方法，并且提出构建函数分析顺序的过程。过程为首先使用强连通算法消除函数递归调用，得到一些组件，再用拓扑排序对组件和组件内部的各函数节点排序，得到一个分层链表，该链表包含各函数的调用顺序和分析顺序，再用一种展开算法将分层链表展开，最终得到源代码中的函数分析顺序。

关键词： 深度遍历 拓扑排序 强连通算法 开源 函数分析顺序

开源项目的开发人员分散，文档不完整，想弄清运行机理，必须分析源代码。即使有文档，想要深入理解软件项目的实现细节，也得看源代码。但是项目的函数众多，没有正确的分析顺序，思路容易混乱，并且分析所有源代码不现实，有的项目源代码太多，比如 Linux 有几百万行源代码，必须分工合作，认清应该分析哪一部分源代码，以及其依赖哪一部分源代码。

1　源码分析的基本思想

源代码中所实现的软件功能的承载体是函数，分析源代码应从分析函数入手，每个函数完成一定的功能，多个函数合力完成一个总的功能。分析函数，首先建立函数之间的关系，由于函数的调用关系可通过关系图描述，因此可通过函数的调用关系构建函数关系图，再用图论算法解决分析顺序问题，之后按顺序分析各个函数的功能，进而将各个函数所实现的局部功能组合成一个总的功能，最终明白源代码所实现的功能和软件的实现过程，从而可在此基础上继续开发，以及软件出问题时，明白问题原因，从而解决问题。简单示例，见图 1，a1 调用 a2，a2 调用 a3，分析顺序相反，只有明白 a3，才能明白 a2，只有明白 a2，才能明白 a1。

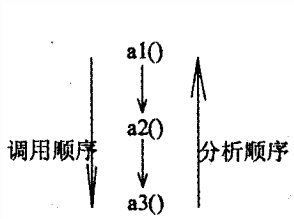

图 1　源代码分析顺序示意图

2　关系图的类型

由于函数的调用是有方向，所以调用关系图是有向图。有向图又分为无环图和有环图。从函数调用关系看，函数关系环是由于递归造成，不论直接递归还是间接递归，函数调用自身，必然造成环，假如函数没有递归，那么关系图为无环图。由于拓扑排序针对有向无环图，必须采取相应的措施消除递归造成的调用环。通过强连通算法将关系图消除递归，得到一些组件，组件之间无递归，从而可进行拓扑排序。

3　拓扑排序和强连通算法的具体应用步骤

在本文中，关系表示为调用和被调用。具体到本文，邻接矩阵是表示函数之间调用关系的矩阵，以及组件之间调用关系的矩阵，当行对应的函数和列对应的函数有调用关系，则相应位置 1，没有调用关系，

① 收稿时间:2008-07-16

则相应位为 0。

具体应用分为 4 个步骤：步骤 1，通过扫描源代码文件，得到函数个数和函数之间的关系，得到函数关系图，用邻接矩阵表示；步骤 2，使用强连通算法消除关系图中的调用环，进一步构建组件的有向无环图，用邻接矩阵表示；步骤 3，实施排序，得到组件调用顺序；步骤 4，将排好序的链表展开，得到源代码中的函数的分析顺序。

就本文所涉及问题，需用到拓扑排序算法和强连通算法，这些算法都基于深度遍历算法，深度遍历算法要得到每个函数节点的访问开始时间和访问结束时间，特别是访问结束时间，拓扑排序算法和强连通算法都要用到，将深度遍历算法描述[1]如下:

Depth-First Forest 的简写为 DFS。

DFS(G)

1)将图中的每一个函数节点的颜色值设置为白色，每一个函数节点的父函数节点设置为空。

2)建立全局变量 time，初始为 0。

3)扫描图中的每一个函数节点 u，假如该函数节点的颜色值为白色，则调用 DFS_VISIT(u)访问该函数节点及其邻接节点。

DFS_VISIT(u)

1)将函数节点 u 的颜色值设为灰色。

2)全局时间变量增 1，将全局时间变量的值赋给函数节点的开始时间。

3)扫描函数节点 u 调用的所有函数节点 v，假如函数节点 v 的颜色为白色，那么将 u 设置为 v 的父函数节点，递归调用 DFS_VISIT(v)。

4)将函数节点的颜色值设为黑色，表示已经全部访问 u 调用的所有函数节点以及间接调用的所有函数节点，间接的意思为 a 调用 b，b 调用 c，a 和 c 为间接调用关系。

5)全局时间变量 time 增 1，并将 time 的值赋给函数节点 u 的完成时间。

现用范例来说明排序过程，共 8 个函数 a、b、c、d、e、f、g、h。函数关系如下：h 调用 d、g，d 调用 c 且 c 又调用 d，c 调用 b，b 调用 a，a 调用 e，e 调用 b，g 调用 f，f 调用 g，g 调用 c，f 调用 b、e，图形描述见图 2，用 G 表示。

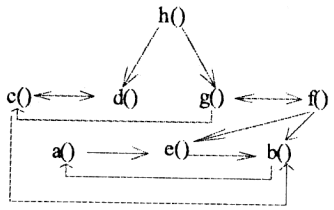

图 2　函数调用图

3.1 定义数据结构

用邻接矩阵表示关系图,在内存中 1 位表示一个关系，按本文范例分析 8 个函数，用 8*8=64 位，需用 8 个字节就可以完全表示。图形 G 的邻接矩阵见式 1。

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \quad (1)$$

由于范例共有 8 个函数,所以邻接矩阵 8 行 8 列。行坐标从上往下分别为 a、b、c、d、e、f、g、h。列坐标从左往右也分别为 a、b、c、d、e、f、g、h。

3.2 消除调用环

使用强连通算法消除调用环，就本文的应用而言，强连通算法描述[1]如下：

1) 调用 DFS(G)计算每个函数节点的访问完成时间。
2) 将图形 G 转置，得到 GT。
3) 调用 DFS(GT)，在本次 DFS 过程中，按 DFS(G)得到的各个函数节点的访问完成时间，对函数节点降序排列，即从 DFS(G)中最后完成的函数节点按序访问，排序算法可使用插入式排序。
4) 强连通组件是图中强连通部分。该步骤的作用是输出各个强连通组件所包含的函数节点。具体实现是从每个强连通组件所包含的函数节点中选择一个函数节点作为每个强连通组件的父节点，其余作为子节点，各个强连通组件的父节点组成一个链表，称为主干链表。子节点也组成一个链表，作为主干链表中的父节点的子链表。

G^T的邻接矩阵见式 2。

组件与组件之间也有调用依赖关系，通过关系构建一个组件关系图，也用邻接矩阵表示，同样在内存中 1 位表示一个关系，该位为 1 表示有调用关系。假如一个组件包含超过 2 个函数节点，那么组件数将小于函数个数，组件调用关系的邻接矩阵所占空间要小于函数关系的邻接矩阵。图 3 为范例的组件关系图。

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (2)$$

得到 4 个组件，组件 1 包含函数 h，组件 2 包含函数 c、d，组件 3 包含 g、f，组件 4 包含 a、e、b。组件 1 调用组件 2、组件 3，组件 2 调用组件 4，组件 3 调用组件 2、组件 4。

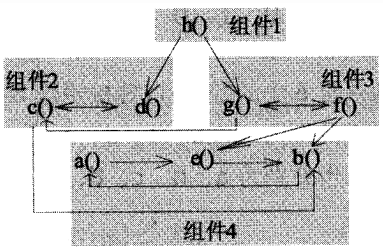

图 3　强连通组件图

3.3 拓扑排序

3.3.1 构建组件关系的邻接矩阵

要拓扑排序，须构建组件的关系矩阵，见式 3。

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad (3)$$

行坐标从上往下分别为组件 1、组件 2、组件 3、组件 4。列坐标从左往右也分别为组件 1、组件 2、组件 3、组件 4。

3.3.2 算法描述

就本文而言，拓扑排序的算法描述[1]为：

1)调用 DFS(G)计算每个组件节点的访问完成时间。

2)按照组件节点和函数节点的完成时间的顺序，形成一个有序链表。

3)返回链表，链表中节点顺序即为函数调用顺序和函数分析顺序。

就本文范例而言，注重函数分析顺序，并且组件数量较少，组件间的关系体现得还并不明显。由于在强连通算法中已经执行 DFS(G)，各函数节点的初始访问完成时间和转置后完成时间都已确定，拓扑排序可以融入进强连通算法中。

组件分析顺序为：组件 4->组件 2->组件 3->组件 1。得到组件分析顺序，对分工合作有很大的帮助，假如开发人员只想了解组件 2，那么他只需分析组件 4，若想进一步了解组件 2 的用处，可以分析组件 3，从而使开发人员只需分析部分源代码，减少了盲目性。最终得到多层链表，返回后的链表见图 4。

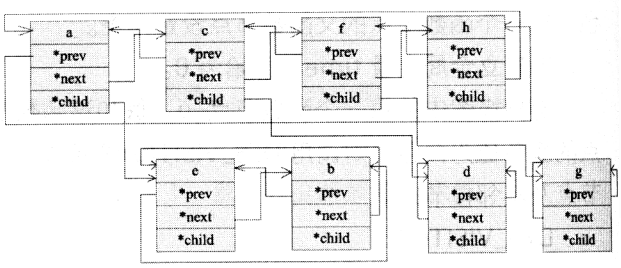

图 4　函数调用的分析顺序层次图

3.4 组件展开方法及分析顺序

经过拓扑排序后得到组件顺序，将组件的最先访问的一个函数即父节点，作为为入口点展开组件，得到函数调用顺序，因为分析顺序和调用顺序相反，在强连通算法中已经对函数关系图转置，所以可直接得到函数分析顺序。

链表的形式见图 4，使用双向链表，这样可快速输出函数调用顺序和函数分析顺序，因为这两个顺序正好相反，切换方向即可得到另外一种顺序。主干链表上的节点为组件中的一个入口点，每个节点的 child 指针，指向该组件所包含的函数节点，图 4 中组件 4 的分析入口点选择 a，a 的 child 指针指向子函数节点为 e、b，组件 2 的分析入口点为 c，c 的 child 指针指向子函数节点 d，组件 3 的分析入口点为 f，f 指向子函数节点 g，组件 1 的入口点为 h，没有子函数节点。

(下转第 95 页)

(上接第 98 页)

父函数节点从左往右输出，并且先输出子函数节点链表，从右往左输出各个子函数节点，之后输出对应父函数节点，本文范例得到顺序为：b->e->a->d->c->g->f->h。

4 用例测试

输入基本格式：1 函数节点数量 2 各个函数名称 3 函数之间的关系。第一个字段为函数节点数量，表明有多少个函数，第二个字段为函数名称，使用符号/隔开。第三个字段为个函数之间的关系，用符号>表示其左边函数调用其右边函数，三个用例见下面：

输入 1：8 a/b/c/d/e/f/g/h h>dg d>c c>d c>b b>a a>e e>b g>fc f>gbe

输出 1：b->e->a->d->c->g->f->h

输入 2：8 a/b/c/d/e/f/g/h a>b b>c

输出 2：c->b->a->d->e->f->g->h

输入 3：6 a/b/c/d/e/f a>b b>c c>d d>e e>f

5 结论

经过测试，拓扑排序和强连通算法可以很好完成源代码中函数调用关系的分析，解决面对大量源代码无从下手的难题，进一步扫描关系图的邻接矩阵，可确定一个函数被多少个其他函数调用，以及该函数调用了多少个函数，例如在式 2 中可看到，第 4 行为 00100001，可确定 d 被 c、h 调用，使开发人员对项目的全局把握性更好，不会失去方向感，目标明确。

参考文献

1 Cormen T H, Leiserson C E, Rivest RL, Stein C.算法导论第二版影印版,The MIT Press,2002.

2 严蔚敏,吴伟民.数据结构(C 语言版)第二版.北京:清华大学出版社,1997.