汇编程序覆盖测试中分支路径数的计算

唐 科，汪文勇，向 渝，罗光春

(电子科技大学计算机科学与工程学院 成都 610054)

【摘要】说明了覆盖测试是软件测试中的重要方法，是软件动态测试的基本手段。并提出通过基本块存储矩阵和邻接表来处理汇编语言程序流图，从而计算其分支路径数的算法并验证其正确性，为进一步的分支覆盖率计算奠定基础。实验证明，该方法能高效准确地计算出给定汇编程序的分支路径数。

关 键 词 覆盖测试; 基本块; 程序流图; 存储矩阵; 邻接表

中图分类号 TP311.56 文献标识码 A

An Algorithm for Calculating the Branch Routes of an Assembly Program in Software Coverage Testing

TANG Ke，WANG Wen-yong，XIANG Yu，LUO Guang-chun

(School of Computer Science and Engineering, UEST of China Chengdu 610054)

Abstract Coverage testing is a basic method for dynamic software testing. An algorithm is proposed for calculating the branch routes of an embedded assembly program in software coverage testing. The correctness of the algorithm is verified on a sample assembly program and it's time complexity is analyzed also.

Key words coverage testing; basic block; program flow graph; storage matrix; adjacency list

本质上，程序的执行表现为一系列逻辑路径的动态组合，测试时尽量覆盖所有可能的路径是软件测试的目标之一，有关的工作叫做覆盖测试。分支覆盖率是衡量覆盖测试质量的基本指标，其计算公式为：

分支覆盖率＝被测分支路径数/静态分支路径数＊100%

可见，计算分支覆盖率与计算程序的静态分支路径数(以下简称分支路径数)密切相关。

目前大多数测试工具是针对C/C++等高级语言的，针对汇编语言的测试工具相当少见。由于汇编语言非结构化的特点，其实现程序往往采用很多条件和无条件跳转指令，使得程序结构复杂化，程序的执行路径难于判断，其分支路径数计算相当困难。在研究汇编程序软件测试的过程中，本文提出了一种计算汇编源程序分支路径数的算法。

1 分支路径数的计算

分支路径数算法的基本思路是：

1) 用邻接矩阵表示程序流图；
2) 用该邻接矩阵构造程序流图的邻接表；
3) 遍历邻接表，计算程序的分支路径数。

收稿日期：2004 – 11 – 25

作者简介：唐 科(1969 –)，男，硕士生，工程师，主要从事网络计算及应用、软件工程方面的研究；汪文勇(1967 –)，男，副教授，主要从事网络计算技术、软件工程方面的研究.

用邻接矩阵表示程序流图和该邻接矩阵构造程序流图邻接表的目的是为了把程序流图表达成易于遍历的数据结构；遍历邻接表，计算程序的分支路径数，是分支路径算法的核心部分。在该算法的主要步骤都针对汇编语言的特点进行了特殊处理，这是本文所介绍的方法的重要特点。

1.1 基本定义

定义1　基本块是指程序中一个顺序执行的语句序列，其中只有一个出口语句和一个入口语句，执行时只能从入口语句进入，从出口语句退出。对于一个给定的程序，可以把它划分为一系列的基本块。

定义2　入口语句是指：1) 程序的第一个语句；2) 或者能够由条件转移语句或无条件转移语句转移到的语句；3) 或者紧跟在条件转移语句后面的语句。

定义3　出口语句是指：1) 下一条入口语句之前的那条语句；2) 或者程序的终止语句。

1.2 基本块数据结构

下面，定义基本块的结构struct　Basic_Block：

```
typedef  struct  Basic_Block
{
  int  execution_tag;          //基本块是否被执行的标记(执行赋值1，未执行赋值0)
  char*  stop_label;           //是否含有结束语句的标记
  int  block_id;               //基本块号，用邻接矩阵存储基本块的标号
  int  in_begin_number;        //基本块入口语句行号，块内语句的起始行号
  int  out_stop_number;        //基本块出口语句行号，块内语句的终止行号
  int  divergent_path;         //到终止节点的分支路径数
  int  outdegree;              //基本块节点的出度
  char*  defining_label;       //基本块内定义的标号，格式：LABEL：MOV R1，A
  char*  reference_label;      //基本块内跳转语句或调用子程序时引用的标号，
                               //引用标号的格式为LCALL  LABEL 或AJMP  LABEL
                               //或JB 0AH, LABEL
  struct Basic_Block  *next;   //指向下一个Basic_Block节点的指针
 } Basic_Block  *BLink_list;
```

基本块一一对应于程序流图中的节点，假定程序流图有n个节点，则基本块也是n个。将基本块分为5类，分别对应于线性程序块、条件跳转指令、无条件跳转指令、子程序调用指令、子程序中断和程序终止指令。程序中所有的相同类型指令通过上述结构构成的一个单链表结构体集中在一起，而程序流图则通过5个单链表基本块之间的逻辑指向关系表达。

为了便于算法实现，增加了两个空节点，其中一个0号节点作为整个程序流图的起始节点，n+1号节点作为整个程序流图的终止节点，则总的基本块数为n+2。将0号节点指向1号节点，0号节点的出度变成1。再在n个基本块节点中，选取出度为0的节点，假设为m，将m号节点指向n+1号节点，则m号节点的出度变为1，n+1号节点的出度不变，仍为0。

1.3 构造基本块邻接矩阵

基本块邻接矩阵是一个(n+2)＊(n+2)的矩阵，定义为一个整型二维数组 int storage_block[n+2][n+2]。当基本块i可以直接到达基本块j时，storage_block[i][j]被赋值1，否则赋值0。

判断基本块之间是否有直接到达关系的方法是，对于基本块i，j，k，当i中的转移语句(条件转移、无条件转移、子程序调用)的标号等于j中的入口语句处的标号时，则i可以直接到达j；当i中出口语句的行号加1等于k中的入口语句的行号时，则i可以直接到达k。

1.4 构造基本块邻接表

根据基本块存储矩阵，可构造基本块的邻接表，即程序流图的邻接表。

构造基本块邻接表的思路如下：对基本块图中的每个顶点建立一个单链表，第i个单链表中的节点表示以顶点i为出发点直接到达的节点。单链表中的每个节点由三个域组成，邻接点域(Adjvex)表示由顶点i直接到达的节点；链域(Nextarc)表示指向下一个i的直接到达节点；visit_tag表示遍历单链表时，本节点是否已被访问。在表头节点中，也有三个域，其中链域(Firstarc)指向第一个表节点，outdegree表示本表头节点的出度，blockid表示节点的id号，即基本块的id号。

1.5 计算分支路径数算法

由基本块邻接表可以计算基本块节点之间的分支路径数。从0～(n+1)的路径总数是被扫描处理的整个程序的分支路径数。为计算分支路径数，需另外定义一个分支路径邻接表。

算法的基本思想是，访问此前构造的基本块邻接表，依次取出每一条弧所连接的两个节点，并判断弧尾节点的访问标志是否为0，若是则生成单链表，挂在分支路径邻接表中，再将有向图邻接表中的弧尾节点的访问标志visit_tag置为1；否则继续寻找下一条弧连接的两个节点，继续上述步骤，直到遇见基本块邻接表中的出度为0的节点。这时，1～n的一条路径已经生成，它由分支路径邻接表里的单链表构成。

遍历有向图邻接表，每个节点的访问标志visit_tag是否都为1，若是则算法结束。当基本块邻接表中的所有节点和弧均被遍历，路径数的计算才算完毕，此时构造出从1～n的所有节点的分支路径单链表。

2 算法验证

下面以一个简单的汇编源程序来验证算法。序号1到31分别为第1到第31条程序语句的行号。

```
1                ORG    0000H
…                 …
16     START:    MOV    DPTR, #2000H
17               MOVX    @DPTR, A
18               ANL     P1, #0FH
19               MOV     P2, A
20               JNB     P1.0,  M1
21               LJMP    AR0
22         M1:   JNB     P1.1,  M2
23               LCALL   AR1
24         M2:   SETB    P1.2
25               MOV     SP, #61H
26               END
27         AR0:  CLR     P1.0
28               LJMP    M2
29         AR1:  MOV     DPTR,  #3000H
30               MOV     28H,   #00H
31               RET
```

	0	1	2	3	4	5	6	7	8
0	0	1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0
3	0	0	0	0	1	1	0	0	0
4	0	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	0	1
6	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	0

图1 示例程序的基本块存储矩阵

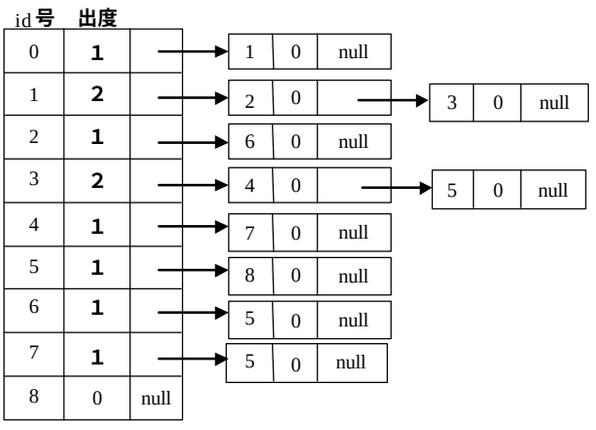

图2 示例程序的基本块邻接表

由程序流图可得到图1的基本块存储矩阵，由基本块存储矩阵可得到图2的基本块邻接表，通过分支路径数算法，由图2得到图3的分支路径邻接表。结果显示该程序共有三条路径。

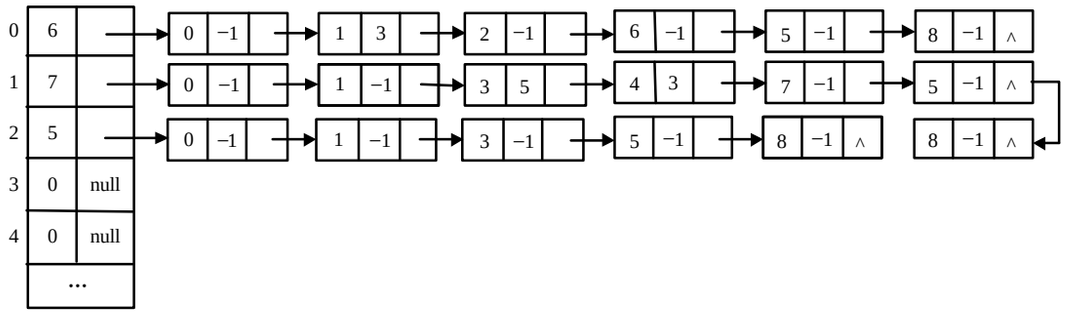

图3　示例程序的分支路径邻接表

3　算法复杂度分析

上述算法由控制结构(顺序、分支和循环三种)和原操作(固有数据类型的操作)构成，则算法时间取决于两者的综合效果。

在构造基本块存储矩阵算法中，初始化的时间复杂度为$O(n)$，对矩阵内的各元素赋值的时间复杂度为$O((n+1)^2)$，处理第i个节点(1～n中某个出度为0的节点)，引一条弧到第$n+1$个节点的时间复杂度为$O(n)$，所以该算法总的时间复杂度为$O(n^2+4n+1)$。同理，分析可得，构造基本块邻接表算法的时间复杂度为$O(n^2+5n+6)$。在计算分支路径数中，假设按最坏情况，1～n的节点中有$n-1$个出度均为2，一个出度为1，则一共有$2n-1$条弧，那么总的算法时间复杂度为O(head分支路径邻接表中的number之和)。

4　总　　结

本文讨论了一种计算汇编语言程序分支路径数的算法，采用邻接矩阵和邻接表存储、处理程序流图，并通过一个示例程序说明了算法的实现并验证了其正确性。本文的工作为覆盖测试的展开奠定了基础，下一步需要做的是研究高效的覆盖测试算法，以提高测试的覆盖率。

参　考　文　献

[1] 陈火旺, 刘春林. 程序设计语言编译原理[M]. 北京: 国防工业出版社, 2003. 272-306

[2] 郑人杰. 计算机软件测试技术[M]. 北京: 清华大学出版社, 1992. 93-129

[3] 王　璞, 张臻鉴, 王玉玺. 基于覆盖的软件测试技术在实时嵌入式软件中的应用研究[J]. 计算机工程与设计, 1998, 19(16): 45-54

[4] 魏光新, 苏　丽. 逻辑覆盖测试工具的设计与实现[J]. 计算机工程与应用, 2000, (5): 106-109

[5] 陈丽蓉, 熊光泽, 罗　蕾, 等. 嵌入式软件的覆盖测试[J]. 单片机与嵌入式系统应用, 2002, (11): 8-11

[6] 孙昌爱, 金茂忠. 基于程序插装的动态测试技术的实现[J]. 小型微型计算机系统, 2001, 22(12): 1 475-1 479

[7] 孙昌爱, 靳若明, 刘　超, 等. 实时嵌入式软件的测试技术[J]. 小型微型计算机系统, 2000, 21(9): 920-924

[8] Huang J C. Program instrumentation and software testing [J]. Computer, 1978, 11(4): 3-8

[9] Hawkins J, Howard R, Haung N. Automated real-time testing for embedded control system [DB/OL]. http:// arxiv. org/abs/cs/0111005, 2002-01-21/2004-12-05

[10] Coulter A. Gray box software testing methodology embedded software testing technique [C].IEEE CNF Digital Avionics Systems Conference, 1999.Proceedings.18th , St Louis, MO ,USA , 2 (10.A.5): 1-8

编　辑　熊思亮