文章编号:1000 - 582X(2005)07 - 0082 - 04

关系型数据库 BOM 表的遍历算法的改进及实现*

石为人,张　星,马振红,林荫华
(重庆大学 自动化学院,重庆　400030)

摘　要:定义了单层 BOM 和多层 BOM 树的结构,并总结了实际运用中的多级型 BOM 遍历的两种基本算法:递归查找法、分层查找法. 在介绍这两种算法的原理和对其速度、资源占用、实用性进行评价的基础上,提出了一种综合以上两种算法的优点,且适合普遍使用的关系型数据库存储的改进的多级型 BOM 遍历算法,使该改进算法包含速度快、资源占用低、实用性高的优点. 同时详细介绍了此算法的软件实现,并且在实际的运用中取得了明显的效果.

关键词:BOM;递归查找法;分层查找法
中图分类号:TP393　　**文献标识码**:A

在 MRPII 系统中,物料清单 BOM(Bill of Material)是系统中最基本的资料. 它是一种描述装配件结构的零件表,其中包括所有子装配件、零件、原材料的清单,以及制造一个装配件所需要的所有物料的数量与层次关系. BOM 是制造业信息系统的一个核心部件,企业的原材料和产成品都将通过 BOM 建立逻辑上的相关关系. BOM 数据在数据库端的保存方式一般都是以单层 BOM 方式进行保存,但是在实际运用过程中都希望能一眼就看出其层次关系及使用数量. 通过 BOM 多级遍历,可以使企业在实际生产经营管理中查询某个产品都由哪些原材料组成;同时,BOM 查询结果也是企业编制生产计划、进行产品配套、生产领料和加工过程跟踪的依据;也是企业计算产品成本进行市场报价的参考;通过 BOM 遍历,还可使企业对产品的设计系列化、标准化、通用化. 因此,设计一种快速有效的 BOM 遍历算法就显得尤其重要[1].

笔者在总结基于单层 BOM 遍历的两种算法(递归查找法、分层查找法)的实现原理基础上,提出了一种改进的 BOM 遍历算法以及用此算法编制的软件系统及其应用实例.

1　BOM 定义

1.1　单层 BOM 结构定义

PMonolayer = (P,R);

P = {p_1,p_2,…,p_n} p_i为产品;

R = { <$p_i(m)$,$p_j(k)$> | $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$ },R 为单层 BOM 的关系集,p_i为 p_j的父件,p_j为 p_i的子件;m、k 为构成 m 个父件所需 k 个子件,如图 1 所示.

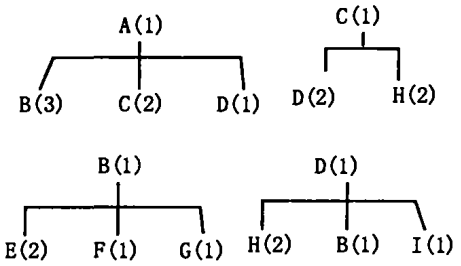

图 1　单层 BOM

1.2　多层 BOM 树结构定义

Pmultilayer = (P,R);

P = { p_1,p_2,…,p_n} p_i为产品;

R = {Pmonolayer} 即多层 BOM 树是由许多单层 BOM 构造而成,如图 2 所示.

2　遍历策略

2.1　递归算法

BOM 遍历的递归算法采用先根遍历的方法,以树中物料是否有下一层物料为递归条件[2].

其算法描述为:

1) 访问根结点;

2) 如果有子件,访问其子件,跳到第 1)步;

3) 如果有下一个兄弟件,跳到第 1)步;

* 收稿日期:2005 - 03 - 01
基金项目:重庆市制造业信息化重大专项子项目"重庆青山工业有限责任公司 CAD/CIMS 一期工程"(2001 - 03)
作者简介:石为人(1948 -),男,重庆人,重庆大学教授,主要从事群决策与分布式智能系统的研究.

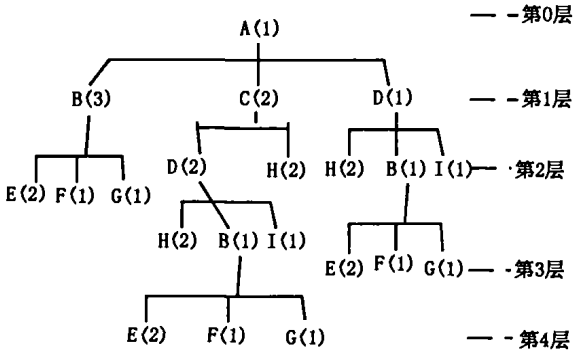

图 2　多层 BOM 树

4) 如果其父件有下一个兄弟件,访问其父件的下一个兄弟件,跳到第 1)步;

5) 判断当前物件是否顶层物件,是则退出;

6) 用其父件代替自身,跳到第 4)步.

针对图 2 所示产品 A 的遍历顺序为 A(1)、B(3)、E(2)、F(1)、G(1)、C(2)、D(2)、H(2)、B(1)、E(2)、F(1)、G(1)、I(1)、H(2)、D(1)、H(2)、B(1)、E(2)、F(1)、G(1)、I(1). 递归算法采用先根遍历[3],因此较好地展现了物料间的层次关系和 BOM 树的结构;利用堆栈操作,其优点是程序简单;根据图论的知识可以分析得知该堆栈的大小为大于该多层 BOM 树的最大层次,小于该多层 BOM 树的最大单子树的节点数量,所以在进行大规模物料遍历时,系统资源消耗大,且无法为程序的堆栈设定大小,一般高级语言程序设计中必须为堆栈设定大小,当遍历过程使用的堆栈大于设定值时会导致系统无法运行或系统崩溃.

2.2　层次遍历算法

BOM 的分层遍历算法就是从 BOM 树的根节点出发,按层次一层一层的往下遍历,类似于数据结构中树的层次遍历过程[2]. 其算法描述如下:

1) 取得当前产品的子件入队列 A;

2) 判断队列空否,空则结束,否则进入步骤 3)执行;

3) 队首元素 a1 出队列送入结果集 B,计算记录 a1 的层次码和使用量;

4) 获取记录 a1 的子件并计算其各子件的用量、计算其各子件的新层次码后入队列;

5) 跳到步骤 2)执行.

针对图 2 所示产品 A 的遍历顺序为 A(1)、B(3)、C(2)、D(1)、E(2)、F(1)、G(1)、D(2)、H(2)、H(2)、B(1)、I(1)、H(2)、B(1)、I(1)、E(2)、F(1)、G(1)、E(2)、F(1)、G(1). 分层遍历能在一次搜索过程中将节点的子件全部搜索出来,大大地提高了搜索的速度,能较好地按层次遍历一个大规模的 BOM 树结构;但由于其按层次搜索的特点,其物料的存放顺序未能展现物料间的父子关系,而在进行物料结构的取代、删除时,需要知道物料间的父子关系,因而 BOM 遍历的分层算法将无法进行以上操作. 并且通过分析可以计算出该算法使用队列的大小为大于等于该多级 BOM 树的单层最大节点数、小于单层最大节点数和下层子节点数之和. 因此在实现该算法的时候也一样难以确定所用队列的大小,所以该算法也无法被工程所采用.

3　遍历策略的改进

3.1　改进算法的目标

该改进算法的目标为:既能像递归算法那样能较好地展现物料间的父子关系、数量关系以及 BOM 树的整体结构[4],以利于进行物料结构嵌套错误检查等操作;又能像分层遍历能在一次搜索过程中将节点的子件全部搜索出来,大大地提高搜索的速度;并且能克服两种算法都有的存储空间大小无法确定的矛盾.

3.2　改进算法的存储策略

因为绝大多数的 BOM 表都是基于关系型数据库,因此可以充分利用其集合操作[5]的优势和表的海量存储[6],利用临时表来实现堆栈的功能. 其入栈操作可以转变为:使用 INSERT INTO 语句插入即可;其出栈操作可以转变为:使用 SELECT 语句,选取最大行号的记录,将取出的数据存入变量,然后删除这条记录. 因此可以建立一张临时表#strack_tab 用于实现堆栈功能. 其中 ROW_ID 用于作为堆栈的大小的标识,max(ROW_ID)作为栈顶数据项 min(ROW _ID)作为栈底元素. FLOOR_CODE 作为层次表示码,S_CODE 子项物料代码,NUMBER 作为针对顶层物料的所需数量.

其语法如下:

```
CREATE TABLE [#strack_tab]
([ROW_ID] [int] IDENTITY (1, 1) NOT NULL
    [FLOOR_CODE] [varchar] (500)NOT NULL ,
    [S_CODE] [char] (20) NOT NULL ,
    [NUMBER] [real] NULL )
    ON [PRIMARY]
    GO
    ALTER TABLE   [#strack_tab] WITH NOCHECK
ADD
    CONSTRAINT      [PK_#strack_tab] PRIMARY
KEY   CLUSTERED
      ([ROW_ID],
    [S_CODE]   )
    ON [PRIMARY]
    GO
```

3.3　改进算法的描述为

由图 2 所示产品可以得出关系表,如表 1 所示.

表 1 对应层次关系表

行号	层次编码	子件代码	使用数量
1	.1	B	3
2	..2	E	6
3	..2	F	3
4	..2	G	3
5	.1	C	2
6	..2	D	4
7	...3	H	8
8	...3	B	4
9	4	E	8
10	4	F	4
11	4	G	4
12	...3	I	4
13	..2	H	4
14	.1	D	1
15	..2	H	2
16	..2	B	1
17	...3	E	2
18	...3	F	2
19	...3	G	2
20	..2	I	1

针对图 2 所示产品 A,可以看出在其遍历过程中有 3 次对产品 B 进行遍历,结果见表 1 的 2、3、4 行和 9、10、11 行以及 17、18、19 行. 有 2 次对产品 D 进行遍历,结果见表 1 的 7 - 12 行,15 - 20 行. 在现实的生产过程中像这种复用型的通用件是很常见的,在一个大的系统中要对这种通用件进行多次遍历将会浪费大量的时间,给系统带来很大的负担. 因此对该算法进行再次的改进以提高算法的速度和系统的效率会有很大的帮助.

改进的算法描述如下:

1) 取得当前产品的子件入临时表 A;

2) 判断表 A 空否,空则结束,否则进入步骤 3)执行;

3) 从表 A 弹出记录 a1 送入结果集 B,获取记录 a1 的用量作为 a1 的子件的用量倍数,获取记录 a1 的层次码;

4) 判断记录 a1 是否出现在结果集中,未出现转到步骤 5)执行;出现转到步骤 7)执行;

5) 获取记录 a1 的子件并计算各子件的用量、计算各子件的新层次码后存入 A 表;

6) 跳到步骤 2)执行;

7) 取得结果集中同记录 a1 相同的物料的第 1 条记录 b1 的行号;取得改行的层次代码放入变量 c,并查找该行之后的第 1 个层次代码小于或等于该层次代码的行号;在这两行记录之间的记录即为 a1 展开的记录,依次拷贝记录结果集,根据当前记录的层次码和结果集中记录相对 b1 的层次码计算各记录的层次码;利用记录 a1 和 b1 的数量相对倍数计算各子记录的使用量;

8) 跳到步骤 2)执行.

观察表 1 和图 2 可以发现表 1 的 2 ~ 4 行的层次代码在图 2 中是第 2 层的代码,9 ~ 11 行的层次代码是图 2 的第 4 层的代码,所以可以知道在表中的层次码是由其上层代码加 1 得到的. 而物料 B 的子件的使用量可由图 1 可以看出一个 B 由两个 E、一个 F、一个 G 组成,也可以由表 1 看出记录 2、3、4 除以记录 1 的使用量正好也是 2、1、1. 所以当遍历过程中第 2 次出现通用件的时候可以不用查看图 1 和图 2. 而直接从表 1 中直接改变层次码和使用量而直接得出结果.

该算法不仅继承了先根遍历算法较好地展现了物料间的层次关系和 BOM 树的结构、利用临时表模仿堆栈操作使得程序简单;而且也继承了层次遍历算法的在一次分层遍历能在一次搜索过程中将节点的子件全部搜索出来,大大地提高了搜索的速度,能较好的按层次遍历一个大规模的 BOM 树结构;并且能解决前面两种算法都不可避免的存储空间大小无法定义的缺点.

4 改进算法的伪码实现

```
func BomExpand(string Pcode) /* Pcode 为要进行多级遍历的物料代码 */
begin
定义变量 F_str = "1";/* F_str 为层次标识符,其格式为 "11111",1 的个数为其物料的层次 */
定义变量 Num_real = 1;/* Num_real 为遍历过程中父件的使用个数 */
定义变量 Line_int1,Line_int2;/* Line_int1,Line_int2 为结果集中与 a1 有相同的子件代码的子记录的起始行号和终止行号 */
构造临时表#strack_tab;
将 P_code 的子件插入临时表#strack_tab;/* 使用的层次码为 F_str,使用数量为各子件的使用数,行号自动生成,子件代码为各子记录的子件代码 */
while(临时表的记录数量 > 0)
begin
从临时表取出一条记录 a1(条件为 ROW_ID = max(ROW_ID));
将 a1 记录存入结果集表 BOM_EXPAND_RETURN_TAB;/* 该表在数据库里已经建立,其结构同#strack_tab,使用的层次码为 a1 记录中的层次码,使用数量为记录 a1 中的使用数,行号自动生成,子件代码为 a1 的子件代码 */
F_str = a1 记录的层次码;
Num_real = a1 记录的使用数量;
在临时表中的 a1 记录删除;
if(记录 a1 在结果集中有记录) then
begin
Line_int1 = 与 a1 相同记录的起始行号;/* 结果集中子件代码与 a1 记录的层次代码相同的第一条记录的行号 */
Line_int2 = 与 a1 相同记录的终止行号;/* 结果集中层次代码的长度小于或等于起始行号的记录的层子代码长度的第 1 个记录的行号,如果没有找到就为结果集的最后一行的行数加 1 */
将结果集中大于 Line_int1 行 Line_int2 的各记录顺次插入结果集的末尾;/* 行号自动生成,使用的层次码为 F_str + 各记录的层次码 - Line_int1 行的层次
```

码,使用数量为各记录的使用数 ÷ Line_intl 行记录的使用数 × Num_rea,子件代码为各子记录的子件代码 * /

end

else if(记录 al 有子件) then

begin

将其子件记录存入临时表;/ * 行号自动生成,层次码为 F_str + "1",子件代码为查询出来的子件记录代码,使用数量为 Num_real × 子件记录的使用量 * /

end

end

将结果集中的层次代码用"."取代"1"末尾在加上各层次码的长度;/ * 这样就可以显示形如表 1 的层次代码 * /

end

5 实例

该算法运用于重庆青山公司一期 CIMS 工程的 MRPII,运行环境为 WIN2000 操作系统,数据库采用 Ms SQL Server2000,算法应用模块为 BOM 顺查,反查以及防止嵌套错误的计算和低层码计算,

改进算法与常用算法在运行时间和处理数据量上的比较如表 2,表 3 所示.

表 2　所有顶层物料进行遍历生成层次表时新旧算法处理速度的比较

BOM 记录/条	常用算法	改进算法
1×10^4	1	0.5
2×10^4	3	1.1
3×10^4	6	2
4×10^4	13	3
20×10^4	无法处理	20
增长速度	接近几何增长	接近线性增长

表 3　所有顶层物料进行遍历生成层次表时新旧算法处理能力比较

BOM 记录/条	常用算法	改进算法
13×10^4	N	N
15×10^4	N	N
17×10^4	N	N
18×10^4	Y	N
19×10^4	Y	N
25×10^4	Y	N

注:N 代表不失败,Y 代表失败.

所讨论的改进算法在实际的使用中取得了明显的效果,极大地提高了系统的效率,减少了软件对大规模的 BOM 的遍历时间和数据库系统资源的占用,大大地提高了数据库的性能和响应能力.

参考文献:

[1] 罗鸿, 王忠明. ERP 原理 设计 实施(第二版)[M]. 北京: 电子工业出版社,2003.

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

[3] 温咏堂. 制造资源计划系统[R]. 北京:机电部北京机械工业总的呼研究所,1990.

[4] 张列平. 制造资源计划 - MRPII 原理于实践[M]. 上海: 上海交通大学出版社,1992.

[5] 冯玉才,刘玉葆,王元珍. 关联规则开采的集合算法[J]. 小型微型计算机系统, 2003,24(3):30 - 34.

[6] 张小剑,陈伟,窦剑平. 关于 BOM 的数据库设计及算法的研究[J]. 计算机应用与软件, 2004,21(5):33 - 34.

Modeling and Algorithms Implement of Traversing of BOM Production Based on Relational Database

SHI Wei-ren, ZHANG Xing, MA Zhen-hong, LIN Yin-hua

(College of Automation, Chongqing University, Chongqing 400030, China)

Abstract: The mono-layer BOM and multi-layer BOM trees are defined, and two of the basic algorithms of traversal of multilevel BOM are summarized as well, which are Recursive Search and Level Search. Based on the analysis of algorithms principles, and evaluating the calculating rate, percentage of using CPU and practicability of them, a kind of improved Traversal of multi-level BOM algorithms is proposed. It is applied in relational database with high calculating rate, lower percentage of using CPU and high applicability. The software implement of these algori-thms is also introduced in detail, and the results of its application show that these algorithms are feasible and effective.

Key words: BOM; recursive search; level search

(编辑　吕赛英)