真前缀标记树——一种面向用户的子树选取策略表示方法

陈华骏[1,3],郑 智[1],倪德明[2]
(1. 中山大学 软件学院,广东 广州 510275;
2. 中山大学 计算机软件研究所,广东 广州 510275;3. 广东技术师范学院,广东 广州 510665)

摘要:针对周期性任务处理中用户对操作数据的规则定义问题,提出了一种带有标记的真前缀树的表示方法,定义了树形数据集合上的选择/排斥规则。根据这种规则,用户制定的周期任务能自动地处理原有的和变化的数据。并给出真前缀标记树的一种GUI表现方式。

关键词:真前缀树;标记;规则
中图分类号:TP311.11 **文献标识码**:A **文章编号**:1673-629X(2006)12-0009-04

Marked Proper-Prefixed Tree: a User-Oriented Approach to Sub Tree Selection Strategy

CHEN Hua-jun[1,3], ZHENG Zhi[1], NI De-ming[2]
(1. Software School, Sun Yat-sen University, Guangzhou 510275, China;
2. Software Research Institute, Sun Yat-sen University, Guangzhou 510275, China;
3. Guangdong Polytechnic Normal University, Guangzhou 510665, China)

Abstract: In order to evoke operations automatically and periodically, it's necessary to define a set of rules to handle these data. Proposes a new expression approach by using a marked proper-prefixed tree, which defines the included directory sub-tree and the excluded directory sub-tree separately. A proper-prefixed tree is presented after trimming a marked tree. A representation form for marked proper-prefixed tree in GUI is given.

Key words: proper-prefix tree; mark; rule

0 引 言

在含有周期性任务处理的应用中,用户常常需要定义一组规则来约束相应的数据集合,根据这组规则,制定的任务能周期性地处理原有的和变化的数据。以Windows文件系统的周期备份任务为例,用户通常会按照如下方式制定备份策略:

1) 指定需要备份的目录和文件,备份任务在周期执行时,可以根据备份策略自动地将指定备份的目录下新增加的目录或者文件备份,不需要用户进行干预;

2) 指定不需要备份的目录和文件,备份任务在周期执行时,自动排除不进行备份的目录下新增加的目录或者文件,不需要用户进行干预。

上述周期任务的操作数据是以目录形式组织的数据集,是一棵目录树的子集。用于表示树子集的方法有许多,正则表达式[1]通过命令模式匹配的方式来确定满足条件的数据集,XML[2]查询语言如XPath[3],XQuery[4]等是利用路径表达式来查询确定满足条件的数据集,文献[5~7]是研究如何提高XML文档的查询效率。以上各种方法都能获得一个树结构数据的子集,但是在关于规则的定义上存在以下几个问题:

(1)不适宜于表达用户对目录结构数据自上而下的操作模式;

(2)难以在图形用户界面(GUI)中表示;

(3)缺少对选择/非选择(排斥)集简便而直观的定义。

文中针对上述问题,提出了一种带有标记的真前缀树的表示模型,在该模型中,树的选择/排斥节点状态使用一种标记表示,并且通过一组函数定义节点状态的转换关系。最后对真前缀标记树进行了总结。

收稿日期:2006-03-13
基金项目:广东省人民政府专项基金项目(20054981)
作者简介:陈华骏(1977-),女,湖北武汉人,硕士研究生,研究方向为软件工程;倪德明,副教授,研究方向为海量数据存储。

1 真前缀标记树

1.1 基本定义

在提出真前缀树的概念之前,先给出一般树节点的相

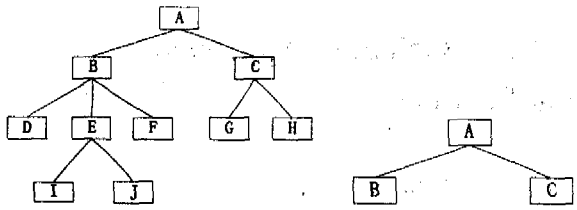
关定义。

定义1 树节点的基本定义。设 T 为一棵树，$N(T)$ 表示 T 的节点集合，根节点记作 R^T，T 中结点 n 的父节点记作 $F(n)$，节点 n 的子节点集合记作 $C(n)$。n 的所有子孙节点的集合记作 $D(n)$，n 的所有祖先节点集合记作 $A(n)$，n 的所有兄弟节点集合记作 $B(n)$。

下面给出真前缀树的定义。

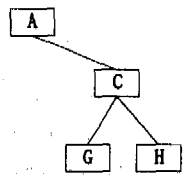
定义2 真前缀树。设一棵树为 T，子树 P 是 T 的真前缀树，如果 $\forall n \in N(P)$，有 $F(n) \subseteq N(P)$，记为 $P \subseteq T$。

真前缀树是对字符串前缀[8]概念的一个扩展，其含义与字符串前缀类似，它是所在树 T 的一棵子树，并且根节点与 T 的相同。图1是真前缀树的一些例子，根据定义可以知道，(b)、(c)是(a)中树 T 的真前缀树，(d)不是 T 的真前缀树，因为(d)的根节点不是(a)的根节点。

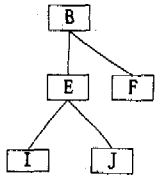


(a)树 T

(b)树 P1，P1⊆T



(c) 树 P2，P2⊆T



(d)树 P3，P3⊆T

图1 真前缀树举例

1.2 真前缀标记树

当用户在树上对节点进行筛选时，需要用一组标记来表示一个树节点是否被用户选中。这里用一个三元组 v 来表示节点的选择状态，v 中每个元素的变化规则及含义说明如下：

定义3 树节点状态标记。设一棵树为 T，n 是 T 中的任意节点，用一个三元组 $v = <C,E,H>$ 标记节点 n 的状态，令集合 FLAG = $\{-1,0,1\}$，$C,E,H \in$ FLAG；$n < a,b,c >$ 表示节点 n 的状态为 $< a,b,c >$。定义如下几个函数表示对 n 的标记 v 中各元素的约束条件：

(1) 单节点筛选函数 $M_c : N(T) \to$ FLAG，用于确定 v 中 C 元素的取值，该元素表示节点是被选中还是排斥。

$$M_c(n) = \begin{cases} -1, n \text{ 被排斥} \\ 0, n \text{ 被忽略} \\ 1, n \text{ 被选中} \end{cases}$$

(2) 继承扩展函数 $M_e : N(T) \to$ FLAG，用于确定 v 中 E 元素的取值，表示节点的选择状态是否是从父节点继承下来的。

$$M_e(x) = \begin{cases} -1, x \in D(n) \text{ 或 } x \in A(n), n \text{ 的状态是继承扩展得到的，不是筛选设置的} \\ 0, M_c(n) = 0, n \text{ 被忽略} \\ 1, n \text{ 的状态是筛选设置的，不是继承扩展得到的} \end{cases}$$

(3) 筛选回朔函数 $M_h : N(T) \to$ FLAG，用于确定 v 中 H 元素的取值，表示一个节点的子孙节点中是否存在与其选择状态不一致的节点。

$$M_h(n) = \begin{cases} -1, \exists x \in D(n) \text{ 且 } M_c(n) \neq M_c(x), \\ \exists y \in D(n) \text{ 使得 } M_c(y) \neq M_c(n) \\ 0, M_c(n) = 0, n \text{ 被忽略} \\ 1, \forall x \in D(n) \text{ 且 } M_c(n) = M_c(x), \\ \forall y \in D(n) \text{ 有 } M_c(y) = M_c(n) \end{cases}$$

(4) T 中节点的状态标记函数 $M : N(T) \to$ FLAG × FLAG × FLAG，表示节点当前的状态；$M(n) = \langle M_c(n_c), M_h(n_h), M_e(n_e) \rangle$。

(5) T 中所有节点的标记集合 $T^M = \{v \mid v \in M(N(T))\}$。

(6) 在 T 上对任意节点的操作函数 $S : T^M \times N(T) \to T^M$，表示对节点进行选择时，其标记的转换规则。函数 S 定义如下：

$S(t^m, n) = \{v \mid v = M(x)$，其中 $x \notin A(n)$ 且 $x \notin D(n)\} \cup$

$\{v \mid v = \langle 1,1,1 \rangle$，当 $M(n) \in F_1$ 或 $M(n) = \langle 1,-1,-1 \rangle$ 且 $\forall y \in A(n), M(y) \neq \langle 1,-1,1 \rangle\} \cup$

$\{v \mid v = \langle 0,0,0 \rangle$，当 $M(n) = \langle 1,-1,1 \rangle$ 或 $M(F(x)) = \langle 1,-1,-1 \rangle\} \cup$

$\{v \mid v = \langle -1,1,-1 \rangle, M(n) = \langle 1,1,1 \rangle$ 且 $M(F(x)) = \langle -1,1,1 \rangle\} \cup$

$$\left\{ v \left| \begin{array}{l} v = \langle -1,1,1 \rangle, \text{当 } M(n) = \langle 1,-1,1 \rangle \text{ 且 } M \\ (F(n)) = \langle -1,-1,1 \rangle \text{ 或 } M(n) = \langle 1,-1, \\ -1 \rangle \text{ 且 } \exists y \in A(n), M(y) = \langle 1,-1,1 \rangle \end{array} \right. \right\} \cup$$

$\{v \mid v = \langle 1,1,-1 \rangle, M(n) \in F_2\} \cup$

$\{v \mid v = \langle M_c(x), M_e(x), -1 \rangle, x \in A(n)\} \cup \{v \mid v = \langle M_c(x), -1,1 \rangle, x \in D(n)\}$

其中 $F_1 = \{\langle 0,0,0 \rangle, \langle -1,-1,-1 \rangle, \langle -1,1,-1 \rangle\}$，$F_2 = \{\langle 0,0,0 \rangle, \langle -1,-1,-1 \rangle, \langle -1,-1,1 \rangle < -1,1,1 >\}$

定义了上述筛选函数后，可以对用户筛选后的树进行处理，得到一棵选择子树和一棵排斥子树。详细说明见第二节。

通过节点操作函数 S，文中对节点的状态标记 v 的取值、命名、描述进行了总结，如表1所示，其中对不同的状态值给出了一种图形表现形式，供参考。

可以用节点状态转换图来说明函数 S 的运算过程，如图2所示。节点的初始状态为“忽略”状态，即，树 T 的所有节点都没有被选中。当选中节点 $n(n \in N(T))$ 时，n 的状态变为“选中” $< 1,1,1 >$ 状态，n 的祖先节点 $A(n)$ 变为“继承选中扩展排斥” $< 1,-1,-1 >$ 状态，n 的子孙节点 $D(n)$ 由忽略状态变为“继承选中” $< 1,1,-1 >$ 状态，

此时完成一次选择操作,称为初始操作。之后的操作依次在初始操作的基础上进行叠加。

表 1　节点的状态描述表

状态值	名称	图形表示	描述
⟨0,0,0⟩	忽略	☐	n 处于初始忽略状态，其所有子孙也被忽略
⟨1,1,1⟩	选中	☑	n 被筛选选中，其所有子孙也被选中
⟨-1,1,1⟩	排斥	☒	n 被筛选排斥，其所有子孙也被排斥
⟨1,1,-1⟩	继承选中	☑	n 的选中状态是由其祖先扩展得到的，其祖先至少有一个是被筛选选中的。n 的所有子孙也被选中
⟨-1,1,-1⟩	继承排斥	☒	n 的排斥状态是由其祖先扩展得到的，其祖先至少有一个是被筛选排斥的。n 的所有子孙也被选中
⟨1,-1,1⟩	选中扩展排斥	☑	n 被筛选选中，但其子孙节点至少有一个是被筛选排斥的。其它的子孙被选中
⟨-1,-1,1⟩	排斥扩展选中	☒	n 被筛选排斥，但其子孙节点至少有一个是被筛选选中的。其所有子孙也被排斥
⟨1,-1,-1⟩	继承选中扩展排斥	☑	n 的孩子中至少有一个节点状态是忽略；或者 n 同继承选中并且 n 的子孙节点至少有一个是被筛选排斥的
⟨-1,-1,-1⟩	继承排斥扩展选中	☒	同继承排斥，并且 n 的子孙节点至少有一个是被筛选选中的。其它的子孙被排斥

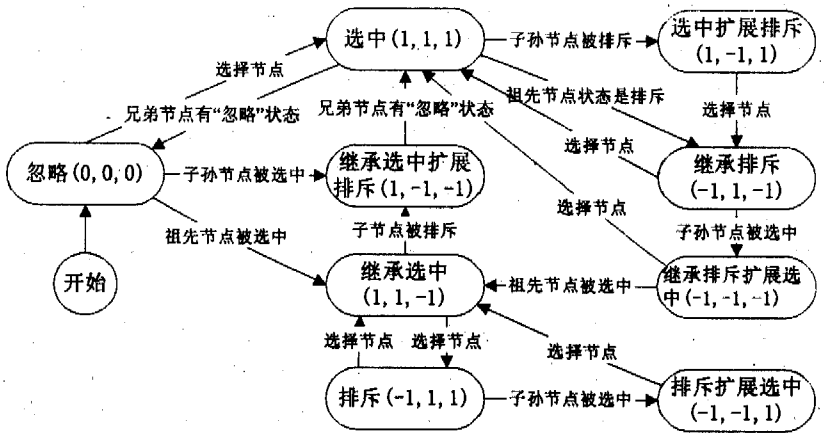


图 2　节点状态转换图

2　分析与比较

文中通过一个例子将真前缀标记树与正则表达式进行比较,说明它在表示用户对目录结构数据周期型(策略型)操作中的优越性。

例 1:用户需要对一个文件系统目录树制定以下周期备份策略:

(1) 目录 D 需要周期备份,将来新加入的目录和文件也要备份。

(2) 目录 RECYCLER, Share, temp, test 不需要备份,将来新加入的这些目录的目录和文件也不要备份。文件 BackupApp.rar 不要备份。

(3) test 目录虽然不要备份,但是它的 javaxml2 目录的 HelloClient.java, HelloHandler.java, HelloServer.java 文件需要周期备份。

2.1　真前缀标记树表示

标记树 T^M 的表示过程如下:

(1) $D<1,1,1>$,节点 D 的所有子孙节点标记变为 $<1,1,-1>$

(2) D/RECYCLER $<-1,1,1>$, D/Share $<-1,1,1>$, D/temp $<-1,1,1>$, D/test $<-1,1,1>$, D/BackupApp.rar $<-1,1,1>$。节点 RECYCLER, Share, temp, test 的子孙节点标记为 $<-1,1,-1>$,节点 D 标记变为 $<1,-1,1>$

(3) D/test/javaxml2/HelloClient.java $<1,1,1>$, D/test/javaxml2/HelloHandler.java $<1,1,1>$, D/test/javaxml2/HelloServer.java $<1,1,1>$。节点 javaxml2 标记变为 $<-1,-1,-1>$,节点 test 标记变为 $<-1,-1,1>$

根据 T^M 得到的真前缀标记树 T^P 为:

* $D<1,1,1>$,
* D/RECYCLER $<-1,1,1>$, D/Share $<-1,1,1>$, D/temp $<-1,1,1>$, D/test $<-1,1,1>$, D/BackupApp.rar $<-1,1,1>$,
* D/test/javaxml2/HelloClient.java $<1,1,1>$, D/test/javaxml2/HelloHandler.java $<1,1,1>$, D/test/javaxml2/HelloServer.java $<1,1,1>$

图 3 为真前缀标记树 T^P 的一种表示方式。在每一次周期备份中,可以通过树 T^P 的各节点标记信息扩展得到满足备份策略的文件系统资源。

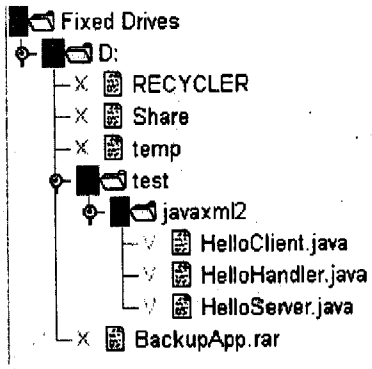


图 3　真前缀标记树 T^M 的图形表示

2.2　正则表达式表示

由于正则表达式只用于匹配而不包含规则标记,用户需要用另外的手段表示与数据相对应的规则。上例中,需要表示目录或者文件是否进行周期备份的规则。此外,正则表达式一般用来匹配线性结构的数据,如字符串,对于具有层次结构的树来说,需要分层地表达各节点的信息,将例子中的周期备份规则可以用正则表达式表示为表 2 的形式。

由表 2 可以看出,正则表达式在表示例 1 的备份策略

时存在以下几点局限性：

(1) 不是面向树型结构的；

(2) 选择/排斥集需要使用另外的方式表达；

(3) GUI 表现困难；

(4) 需要用户了解正则表达式的匹配命令。

表 2 正则表达式表示

步骤	规则	资源指定
1	备份	D:/*
2	不备	D:/RECYCLE/*
3	不备	D:/Share/*
4	不备	D:/temp/*
5	不备	D:/test/*
6	不备	D:/BackupApp.rar
7	备份	D:/test/javaxml2/javaxml2/HelloClient.java
8	备份	D:/test/javaxml2/javaxml2/HelloHandler.java
9	备份	D:/test/javaxml2/javaxml2/HelloServer.java

3 结束语

文中针对在周期性任务处理中树型结构数据的定义提出了真前缀标记树的表示方法，该方法使用一个三元组表示用户对节点数据的选择/排斥策略，通过一组变化规则进行树节点的状态转换，并给出一种 GUI 的表现形式，说明真前缀标记树方法不仅易于在 GUI 中表现，也使用户对操作规则的理解更简单、更直观。

参考文献：

[1] Tansley D. Linux and Unix Shell Programming[M]. [s. l.]: Addison Wesley/Pearson, 2000.

[2] Bray T, Paoli J, Sperberg-McQueen C M, et al. Extensible markup language (XML) 1.0 (2nd Edition)[EB/OL]. W3C Recommendation, 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[3] Clark J, DeRose S. XML Path Language (XPath), Version 1.0[EB/OL]. W3C Recommendation, 1999-11. http://www.w3.org/TR/xpath.

[4] Boag S, Chamberlin D, Robie J, et al. XQuery 1.0: An XML Query Language[EB/OL]. W3C Candidate Recommendation, 2005-11-03. http://www.w3.org/TR/xquery/.

[5] 王 强，武港山. 对 XPath 模式定位能力的扩充[J]. 计算机研究与发展，2001(6)：674-678.

[6] 吴恒山，吴亚辉，班鹏新. XML 查询优化的面向路径可扩展模型[J]. 华中科技大学学报：自然科学版，2004(3)：62-64.

[7] 王 静，孟小峰，王 宇，等. 以目标节点为导向的 XML 路径查询处理[J]. 软件学报，2005，16(5)：827-837.

[8] Yazdani N, Min P S. Prefix Trees: New Efficient Data Structures for Matching Strings of Different Lengths[C]//2001 International Database Engineering & Applications Symposium (IDEAS '01). Grenoble, France: IEEE Computer Society, 2001：76-85.

(上接第 8 页)

230ms，第二部分的车牌定位算法平均时间为 12ms，相比同类算法亦有提高，且算法简单，边缘清晰、定位准确。只是由于某些原图(如汉字模糊)，未能清晰构画出汉字的边界。

文中算法改进在于将模糊边缘检测算法运用到车牌定位中，并利用其优异的性能，节省了计算时间，得到了清晰的边缘图像，为后面的分割和识别节省了计算时间，提供了可靠的边缘信息。并且根据边缘信息，设计了相应的车牌定位算法，定位准确、时间短，适用于车牌识别这种适时性要求高的系统，具有良好的应用前景。

图 2 原始图像

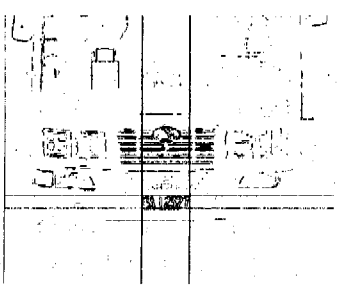


图 3 定位后的图像

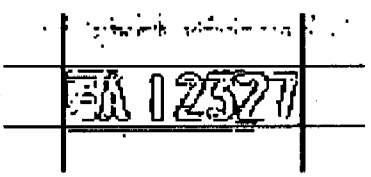


图 4 文中算法检测的边缘

图 5 v_soble 算法检测的边缘

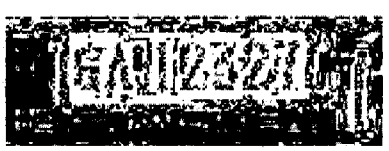


图 6 gauss 算法检测的边缘

参考文献：

[1] LIANG L R, LOONEY G L. Competitive fuzzy edge detection[J]. Applied Soft Computing, 2003(3)：123-137.

[2] 段 震，姚芳兵，张 铃. 基于构造性学习方法的车牌定位[J]. 微机发展，2004，14(8)：41-43.

[3] 鲁继文，张二虎. 基于分类器的图像模糊边缘检测快速算法[J]. 计算机应用，2005(10)：2374-2375.

[4] Zheng Danian, Zhao Yannan, Wang Jiaxin. An efficient method of license plate location[J]. Pattern Recognition Letters, 2005(26)：2431-2438.

[5] Li Xiaobo, Liu Zhi-Qiang, Lenug Ka-Ming. Detection of vehicles from traffic scenes using fuzzy integrals[J]. Pattern Recognition, 2002(35)：967-980.