Journal of Southeast University (English Edition) Vol. 23, No. 3, pp. 431 – 434 Sept. 2007 ISSN 1003—7985

Matching algorithm for label-based integrable-ware query

Xiao Kun[1] Chen Shihong[1,2]

([1]School of Computer, Wuhan University, Wuhan 430072, China)

([2]National Multimedia Software Engineering Technology Research Center, Wuhan University, Wuhan 430072, China)

Abstract: Based on tree-inclusion matching, retrieval may be transformed into matching between the query tree and the integrable-ware label tree. Considering the retrieval specialities of integrable-ware, three theorems of matching are given. On this basis, the inverted-path string algorithm for the integrable-ware label tree query is proposed. This algorithm searches from leaf nodes rather than from root nodes, and considers about the path length and the total number of leaf nodes. It can terminate the failed matching as early as possible and avoid spending too much time on loop comparisons in character string matching. It utilizes the dictionary suffix order to skip much of the impossibility matching path. The experimental results show that this algorithm enhances the recall and the precision of integrable-ware query efficiency while maintaining the searching speed of the integrable-ware.

Key words: integrable-ware; retrieval; label; tree-inclusion matching

An integrable-ware is an educational resource component and teaching unit reflecting knowledge-points independently. It is easily used for different integrable-ware systems. The integrable-ware system, which is an intelligent tutor system, decomposes the course-ware into a lot of integrable-wares, and it can develop different teaching contents only using different integrable-ware systems.

In the integrable-ware system, an effective query allows users to search the requirable integrable-ware quickly. At present, the keyword matching and the path containment query are two major queries of integrable-ware. However, the keyword matching little shows the interrelation of the keywords. Thus, the recall of the keyword-based query is lower. The speed of the path containment query is not very quick[1], because it must match with every branch path in the integrable-ware label tree.

Based on the integrable-ware specialities, this paper provides an inverted-path string algorithm. Considering this, the unordered-label tree results are given for reference[2–3]. The recall and the precision of the inverted-path string query are superior in principle and experiment. This paper gives the realization for the inverted-path algorithm.

1 Tree-Inclusion Matching

If a tree matching is successful, the structural mapping between the query tree and the integrable-ware label tree must satisfy the matching relationships of the tree structures. However, in the integrable-ware query based on XML, users usually have no idea about the exact structures of the integrable-ware trees. In order to understand the tree matching, the tree must be first introduced.

Definition 1 An acyclic connected graph T is a tree, if there is $T=(V, E, \text{root}(T))$. V is a limited node set and $\text{root}(T)$ is the root node of tree. E is an edge set. It is the binary relationship, which satisfies the anti-reflexive, the skew symmetric and the transitivity. If there is $(u, v) \in E$, u is the father node of v. It is expressed as $u=\text{parent}(v)$ or $v=\text{child}(u)$. A tree T must satisfy three conditions[4–5]:

① No nodes are the father of the root node;

② Every node except the root node has only the father node in V;

③ Every node except the root node has $(\text{root}(V), v) \in E^*$, and E^* is the transition closure of E.

If there is $(v_1, v_2) \in E^*$ about v_1 and v_2, v_1 is the ancestor node of v_2. It is expressed as $v_1=\text{ancestor}(v_2)$ or $v_2=\text{descendant}(v_1)$. $A(u)$ is the node set of the ancestor nodes of u. $D(u)$ is the node set of the descendant nodes of u. If there is $u \in V$, $T[u]=(V', E', u)$ is the sub-tree of T and the root node of $T[u]$ is u. Additionally, there are $V'=\{u\} \cup D(u)$ and $E'=E \cap (V' \times V')$ in $T[u]$.

If every node has a label in T, T is called a label tree. If the order of the brother nodes is not important in T, T is called an unordered-label tree. The matching of the unordered-label tree T can transform into the mapping of the tree[4]. The tree mapping is defined as

Received 2007-05-18.

Foundation item: The National High Technology Research and Development Program of China (863 Program) (No. 2002AA111010).

Biographies: Xiao Kun (1976—), male, graduate; Chen Shihong (corresponding author), male, professor, chen_lei0605@sina.com.

follows[1]:

Definition 2 Tree mapping: $T_1 = (V_1, E_1, \text{root}(T_1))$ and $T_2 = (V_2, E_2, \text{root}(T_2))$ are two unorder ed-label trees. $h \subseteq T_1 \times T_2$ is a mapping from T_1 to T_2, if $(v_{1i}, v_{2i}) \in h$ satisfies the conditions:

① $v_{1i} = v_{1j} \Leftrightarrow v_{2i} = v_{2j}$. This means that the node is the bijection in two trees.

② $v_{1i} = \text{ancestor}(v_{1j}) \Leftrightarrow v_{2i} = \text{ancestor}(v_{2j})$. This means that the relationships among the nodes are kept in the mapping.

The domain of h is $\{v_1 \in T_1 \mid v_2 \in T_2, (v_1, v_2) \in h\} \subseteq T_1$, and the range of h is $\{v_2 \in T_2 \mid v_1 \in T_1, (v_1, v_2) \in h\} \subseteq T_2$.

Definition 3 Tree-inclusion matching[3,6]: $T_1 = (V_1, E_1, \text{root}(T_1))$ and $T_2 = (V_2, E_2, \text{root}(T_2))$ are two unordered-label trees. The mapping f: $V_1 \rightarrow V_2$ is the tree-inclusion matching from T_1 to T_2, if f satisfies the following four conditions:

① If there are $\forall v_1 \in V_1$ and $\exists v_2 \in V_2$, there is $v_2 = f(v_1)$;

② There are $\forall u, v \in V_1$ and $u = v$, if and only if there are $f(u) = f(v)$ and $f(u), f(v) \in V_2$;

③ There is $\text{label}(v) = \text{label}(f(v))$;

④ If there is $u = \text{ancestor}(v)$, if and only if there is $f(u) = \text{ancestor}(f(v))$.

T_1 is the query tree and T_2 is the integrable-ware label tree (see Fig. 1).

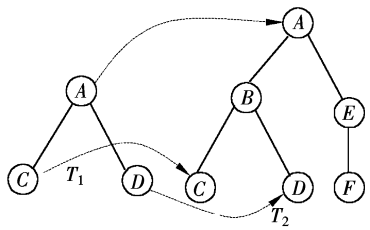

Fig. 1 Tree-inclusion matching

The tree-inclusion matching only requires maintaining the ancestor-descendout relationships among the nodes. It can effectively protect the discrepancies among the label items, and enhance the precision and the recall of the integrable-ware.

2 Speciality of Integrable-Ware Label Tree

2.1 Label of integrable-ware

The labelled items are divided into "function" and "depiction" in the integrable-ware. The labelled items are also called label items or sub-label items. The contents of an item is called the label value. The "function" describes what aspects the integrable-ware is used for. The "depiction" describes some expanding integrable-ware information. The "triangle proof" integrable-ware is described as follows:

```
〈interware〉
〈function〉
  〈course〉maths〈/course〉
    〈knowledge〉
      〈name〉triangle〈/name〉
      〈way〉proof〈/way〉
    〈/knowledge〉
  〈/function〉
  〈depiction〉
    〈author〉zhang〈/author〉
    〈author〉li〈/author〉
    〈medium〉picture〈/medium〉
  〈/depiction〉
〈/interware〉
```

For an integrable-ware, the labelled items map into the father nodes and child nodes in the tree, and the label values map into the leaf nodes. Relying on this method, an integrable-ware label tree is built. Fig. 2 is the integrable-ware label tree of the above description. Some label values of the nodes are abbreviations. For example, F means "function", and C means "course", etc. The retrieval conditions of the integrable-ware can be transformed into the query tree similarly.

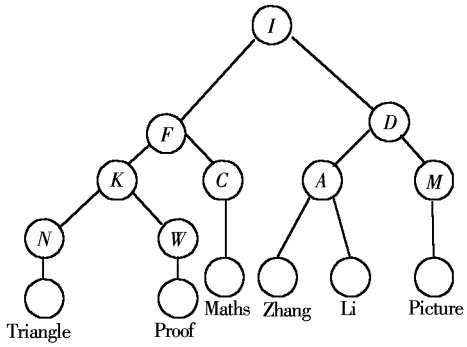

Fig. 2 Integrable-ware label tree

2.2 Theorem of tree-inclusion matching

Theorem 1 The nodes matching with the leaf nodes of the query tree must be the leaf nodes of the integrable-ware tree.

According to the building process of the tree, the label values only occur on the leaf nodes.

Theorem 2 For one successful path matching, the path length from root node to leaf node in the query tree is shorter than that in the integrable-ware tree.

The path length from root node to leaf node v_1 in T_1 is m, and the path length from root node to leaf node v_2 in T_2 is n. And, there is $v_2 = f(v_1)$. We assume $m > n$.

Relying on definition 2, the ancestor nodes of v_1 are greater than those of v_2, and there is $A(v_1) = m > A(v_2) = n$.

According to the conditions ① and ④ of the definition 3, there is $A(v_2) = \{f(v_{11}), f(v_{12}), \ldots, f(v_{1m}), \ldots\} \geqslant m$, in other words, $n \geqslant m$.

It conflicts with the assumption $m > n$.

Theorem 3 For one successful tree matching, the total of leaf nodes in the query tree is less than that in the integrable-ware tree.

From theorem 1, the nodes matching with the leaf nodes of the query tree are only the leaf nodes of the integrable-ware tree. If the number of leaf nodes in the query tree are greater than those in the integrable-ware tree, the matching must fail because some leaf nodes in the query tree will not match with those in the integrable-ware tree.

The multi-path query of the semi-structure is equal to the tree-inclusion matching[5], and the tree may be expressed as the encoded character string[7]. Thus, the multi-path query of XML is equal to the matching of the character string[8]. The character string matching avoids the decomposition of the query expression and enhances the efficiency of the XML query.

3 Inverted-Path Query

Considering theorem 1, theorem 2 and theorem 3, we present the inverted-path string algorithm. It includes two steps. First, we build an inverted-path string sequence for all integrable-wares in the base. Secondly, a matching between the inverted-path string of the integrable-ware base and those of the query are executed. If it satisfies the tree-inclusion matching, it will be successful.

3.1 Inverted-path algorithm

"#", "-" and ";" are three special characters which are not used in the integrable-ware label trees and the query tree. "#" is used to split the integrable-ware in the integrable-ware base; ";" is used to split the paths in the integrable-ware label tree (query tree), and "-" is used to split the labels in the path.

Algorithm 1 Build the inverted-path string of the tree

Input: integrable-ware label tree (query tree) T.

Output: the character string S.

① $S = \varnothing$;

② Building the character strings paths from root node to leaf nodes for T, recording the total number of the paths in T, the path lengths, and the total number of paths in any given path length;

③ Reversing the character strings of every path;

④ Sorting the character strings based on the path lengths;

⑤ For the same path length, sorting the character strings based on the dictionary suffix order;

⑥ Joining the character strings into a long character string S with ";".

S is an inverted-path string for T. The character strings splitted ";" are called the sub-strings of S. The character strings splitted "-" are called the tags of the sub-strings.

Fig. 3 is the inverted-path string of an integrable-ware tree, where the inverted-path is C-B-A; D-B-A; E-A; F-A. The string "4; 2; 3; C-B-A; D-B-A; 2; 2; E-A; F-A;" is the inverted-path string of the integrable-ware.

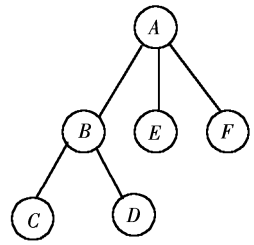

Fig. 3 Inverted-path of integrable-ware tree

Algorithm 2 Build the inverted-path string for the integrable-ware base

Input: integrable-ware base D.

Output: inverted-path string SD.

① SD = $\varnothing$;

② For the integrable-ware tree T in D;

③ Building the character string S for T based on algorithm 1;

④ Appending "#" into S, and inserting S into SD relying on the total number of the paths of T and the dictionary suffix order;

⑤ End for.

SD is the inverted-path string for the integrable-ware base

Algorithm 3 Match the inverted-path string between the query tree and the integrable-ware tree

Input: inverted-path string Q for query tree; inverted-path string S for integrable-ware label tree.

Output: If the matching succeeds, returned value is true. Otherwise, returned value is false.

Q_i is the i-th sub-string in Q and S_j is the j-th sub-string in S. Q_k is the total number of sub-strings waiting for matching in Q. S_k is the total number of sub-strings, which are to be matched in S.

① If Q_k is much more than S_k, the algorithm finishes and the returned value is false.

② If the path length of Q_i is longer than that of S_j, the algorithm finishes and the returned value is false.

③ If the first tag of Q_i is less than that of S_j according to the dictionary suffix order, we skip the rest of the sub-strings with the same path length in S and return to ①.

④ If the matching between Q_i and S_j is successful and there are sub-strings still waiting for matching in Q, we return to ①.

⑤ The algorithm finishes and the returned value is true.

Algorithm 4 Search for integrable-ware in the base

Input: inverted-path string Q for query tree; inverted-path string S for integrable-ware label tree; inverted-path string SD for integrable-ware base.

Output: integrable-ware set R satisfying query and $R = \varnothing$.

① While SD $\neq \varnothing$;

② If the total number of the sub-strings in Q is much more than those of S, the loop terminates and the algorithm finishes;

③ If algorithm 3 returns true, there is $R = R \cup \{S\}$; otherwise, we return to ①;

④ End while.

3.2 Algorithm analysis

We often spend much time on the loop comparison in character string matching. The inverted-path

string algorithm makes the specialties of the integrable-ware label tree terminate the failed matching as early as possible. It searches from leaf nodes rather than from root nodes, and considers the path lengths. It utilizes dictionary suffix order to skip much of the impossibility matching path. It only requests the consistency of the precedence relation between the query string and the data strings, and enhances the recall.

3.3 Experimental results

In order to verify the effectiveness of the inverted-path string algorithm, we experiment in the integrable-ware base. The query results include the entire and approach matching integrable-ware. The recall of the inverted-path string algorithm is 0.97 and the precision is 0.94.

4 Conclusion

Based on the integrable-ware label tree specialties analysis, this paper presents the inverted-path string algorithm. The inverted-path string algorithm can terminate the failed matching as early as possible. At present, it is being used in practical intelligent tutor systems. Of course, it may be used in associated realm.

References

[1] Jia Xiaohui, Chen Dehua, Yan Mei, et al. Research on matching model and algorithm for faceted-based software component query[J]. *Journal of Computer Research and Development*, 2004, **41**(10): 1634 - 1638. (in Chinese)

[2] Dennis Shasha, Jason T L. ATreeGrep: approximate searching in unordered trees[C]//*Proc of the* 14*th International Conference on Scientific and Statistical Database Management* (*SSDBM'*02). Edinburgh, Scotland, 2002: 89 - 98.

[3] Xu Ruzhi, Qian Leqiu, Chen Jianping, et al. Research on matching algorithm for XML-based software component query[J]. *Journal of Software*, 2003, **14**(7): 1195 - 1202. (in Chinese)

[4] Wang Yuanfeng, Xue Yunjiao, Zhang Yong, et al. A matching model for software component classified in faceted scheme[J]. *Journal of Software*, 2003, **14**(3): 401 - 408. (in Chinese)

[5] Cooper B, Sample N, Franklin M, et al. A fast index for semi-structured data[C]//*Proc of the* 27*th International Conference on VLDB*. Roma, Italy, 2001: 341 - 350.

[6] Kilpelainen Pekka. Tree matching problems with applications to structured text databases[D]. Helsinki, Finland: Department of Computer Science of University of Helsinki, 1992.

[7] Chen M S, Yu P S, Wu K L. Optimization of parallel execution for multi-join queries[J]. *Knowledge and Data Engineering*, 1996, **8**(3): 416 - 428.

[8] Shasha Dennis, Wang Jason T L, Giugno Rosalba. Algorithmics and applications of tree and graph searching[C]//*Proc of the* 21*st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. Madison, WI, USA, 2002: 39 - 52.

基于标注的一种积件查询匹配算法

肖 锟[1] 陈世鸿[1,2]

([1] 武汉大学计算机学院,武汉 430072)

([2] 武汉大学国家多媒体软件工程技术研究中心,武汉 430072)

摘要:基于树的包容匹配思想,把积件的查询转化为查询树与积件标注树之间的匹配.通过研究积件查询的特点,提出积件标注树匹配的3个定理.在此基础上,提出积件查询的逆路径字符串匹配算法.该算法从叶节点开始进行匹配查找,同时考虑从叶节点到根节点的路径长度关系,能尽早终止不能匹配成功的路径,避免了字符串的循环反复查找,同时利用同一路径长度下字符串按字典排序,跳过大量不可能匹配的路径.实验结果表明,此方法在保持积件查找速度的前提下,能有效提高积件的查全率和查准率.

关键词:积件;查询;标注;树包容匹配

中图分类号:TP311