Research on Data Acquisition Algorithm

Jiachao Rong, Yang Zhang
Department of economics and management, Tongren Polytechnic College, Gui Zhou, Tongren, 554300, China

Abstract. This paper studies the inclusion relational structure connection algorithms in three basic structure relationships, namely MPMGJN algorithm, Stack-Tree algorithm and Queue-Tree algorithm. The MPMGJN algorithm is the most basic structure connection algorithm and is less efficient due to unnecessary scanning and searching during connection. On this basis, this paper compares and analyzes two improved cache-based structure connection algorithms, Stack-Tree algorithm and Queue-Tree algorithm. At the end of the paper, the Queue-Tree algorithm implemented in C# is given, and the algorithm is tested with simulation data.

1. Introduction
The amount of data that needs to be processed in the financial sector is so large that the difficulty of querying critical information increases. With the development of XML-related standards, a large amount of XML data has appeared on the Internet. The research on how to effectively manage these documents has promoted the generation of XML database technology. The main idea of judging the structural relationship between nodes is to decompose a complex query into a set of binary basic structural relationships, and combine the query results of the binary basic structural relationships to form the final result. Therefore, the efficiency of structural connection execution has a direct impact on the efficiency of XML query processing, so it is important to explore the structure connection algorithm.

2. Direct merge structure connection algorithm——MPMGLN algorithm
The structure connection algorithm is mostly based on the idea of merging. The MPMGJN algorithm is a commonly used direct merging algorithm. Compared with the standard merging connection implemented in the commercial database system, the connection number is greatly reduced, and it has strong practical value.

The basic idea of the MPMGJN algorithm: to set up the two relational tables A and D of the join, the first tuple a1 in the outer table A firstly searches the inner tuple D for the first tuple that may be connected with the tuple (That is, the condition a1.begin<D.begin is satisfied, that is, the scan start point, and then the scan is sequentially performed downward from the scan start point, and all the tuples satisfying the connection condition (a1.begin<D.end) are connected, and then A The next tuple of the list repeats the above scanning process until the A or D list is scanned.
Figure 1. Scan process in inna-list of MPMGJN algorithm

It can be noticed that for the D list, except that the first scan is performed from the very beginning of the list, each time after the search starts from the beginning of the previous scan, this is because one element node may be more the descendants of an ancestor node, the so-called nested problem of the same name. Therefore, during the entire connection of the two tables, the potential ancestor list A list is only scanned once, and there are repeated scans during the D list scanning process. As shown in Figure 1.

3. Stack-Tree algorithm
The MPMGJN algorithm needs to repeatedly scan the D list. In the worst case, the performance is very poor. In order to avoid the connection efficiency caused by unnecessary repeated scans of the D list, the researchers have proposed a cache-based structural connection algorithm.

The Stack-Tree algorithm is a stack-based structure connection algorithm that only needs to perform a sequential scan of the A and D lists to complete all the connection operations of the scale. The algorithm needs to create a stack for the A list, and store the nodes in the A list that may be connected. At the same time, the nodes stored in the stack have a nested relationship, that is, n nodes are stored in the stack from bottom to top. \{s1, s2, ..., sn\}, for each node si, it is the descendant node of node si-1.

The basic idea of the Stack-Tree-Desc algorithm: When scanning a list to get a new node a, if the stack is empty at this time, the node a is directly pushed onto the stack. If it is not empty, it is necessary to judge the node a. Whether it is a descendant of the top node of the stack, and if so, it is pushed onto the stack. When scanning the D list, a new node d is obtained to determine whether it is a descendant of the top node of the stack. If it is, it can be determined that it is also a descendant of all the nodes in the stack, and the connection is performed in turn, because the stack maintains Nested ancestor nodes, and it can be determined that the ancestors of the d-nodes are not likely to be in other un-stacked node elements in the A list, because all nodes that may become d ancestors have been pushed onto the stack; If d is not a descendant of the top node of the stack, the top node of the stack is popped off, and it is determined whether it is a descendant of the new stack top node until the stack is empty.

Obviously, the result of the connection obtained by the algorithm through the stack is ordered according to the descendants, but it is customary to accept the output of the ancestors in an orderly manner. In fact, in order to make further and more complex queries, it is necessary to output the ancestors in order, so the Stack-Tree-Anc algorithm appears. Although the algorithm can output the result of ancestor order connection, it requires more overhead. In addition to maintaining a stack, it maintains two lists for each element in the stack. One is used to temporarily store the connection of the node. As a result, another is used to temporarily store the connection results inherited by all descendant nodes of the node. The specific algorithm is not specifically described here.

4. Queue-Tree algorithm
Another cache-based structural join algorithm is the Queue-Tree algorithm. The algorithm only needs to maintain a queue, and scans the A and D lists once to output the result of the ancestor order. This algorithm can avoid the overhead caused by the Stack-Tree-Anc algorithm to maintain a stack and two lists.

The A list node is divided into a number of disjoint sub-collections, each set including an ancestor node and all descendant nodes of the node, which should be some consecutive nodes in the A list. The
first node of each sub-set is the root node of the set, denoted as r, and the set of all descendant nodes in the set is denoted as Ar.

The algorithm implements the structure connection according to the ancestor order. To this end, a queue Dqueue needs to be established to store all the descendant nodes of the root node in the D list in the sub-collection. Obviously, each node in the Ar set is in D. The descendants in the list are also all included in the queue.

The basic idea of the Queue-Tree algorithm: first determine the first node r in the sub-set in the A list, start searching from the current node of the D list, and find the first node that can be connected, that is, satisfy r.begin <D.begin, from this point, let all nodes that can connect with r enter the queue, store the node pairs that can be connected to the output; then, in turn, look for each node in Ar in the queue. The connected node is stored in the output, and then the above process is repeated for the next node of the A list.

In this algorithm, three aspects of optimization are performed:
(a). If the node r has no nested elements in the A list, then there is no need to maintain the queue and the result can be output directly.
(b). The nodes entering the queue are not all descendant nodes of r in the D list. If r is recorded as a in the next node in the A list, then the immediate descendants of r (ie children) and a in the D list The nodes before the offspring did not enter the queue.
(c). In order to reduce the inclusion relationship between Ar and the queue, in order to reduce the unnecessary matching, the pre-detection technique is adopted, that is, when a node in Ar is processed, the next node of the node is predicted to be in the queue. The location that can be matched.

5. Algorithm simulation implementation
An abstract XML tree was introduced as test data in the experiment. As shown in Figure 2, the XML document is named text.xml, and all the nodes in the document tree have been assigned a list to be attributed. Among them, the triangle node represents the unrelated node, the box node represents the node in the A list, and the circle node represents the node in the D list.

![Figure 2. XML document tree](image)

For the XML document tree set in the experiment, all sets of the A list can be divided into two self and {a1, a2, a3, a4} and {a5}. For the first subset of the A list {a1, a2, a3, a4}, because there are nests, their descendants {d3, d4, d6, d7} in the D list need to enter the queue, and with the descendants of a1 A2a3a4 performs matching connection; for the second sub-set {a5} of the A list, since there is no nesting, its connection result with the D list is directly output, and does not need to enter the queue.
6. Key code

```java
// LinkedListNode
LinkedListNode<Node> a = new LinkedListNode<Node>(new Node());
LinkedListNode<Node> d = new LinkedListNode<Node>(new Node());
LinkedListNode<Node> r = new LinkedListNode<Node>(new Node());
a = aList.First;
d = dList.First;
while ((a != null && d != null) || dQueue.Count != 0)
{
    r = a;
a = a.Next;
    while (d.Value.begin <= r.Value.begin) // Find the starting point that matches the current node r in the D list.
    {
        d = d.Next;
    }
    // There is no nested relationship between a and r, that is, not the same sub-collection, no need to use the queue
    if (a == null || a.Value.begin > r.Value.end)
    {
        for (; d != null && d.Value.begin < r.Value.end; d = d.Next)
        {
            append(r, d); // Find the node that can satisfy the connection condition and connect with r in D
        }
    }
    else // a is a descendant of r, you need to use the queue to store the descendant non-child nodes of r
    {
        for (; d.Value.begin < r.Value.end; d = d.Next)
        {
            int h = d.Value.begin;
            //int s = r.Value.end;
            append(r, d); // Connected to the node in D where r meets the connection condition
            if (d.Value.begin > a.Value.begin && d.Value.level > r.Value.level + 1) // Send the descendant of r to the child node into the queue
            {
                dQueue.Enqueue(d);
            }
        }
        int length = dQueue.Count;
        if (length == 0) // The queue is empty, the subcollection has only one node
        {
            while (a.Value.begin < r.Value.end)
            {
                a = a.Next;
            }
        }
        else // For each node a in A, match its descendants in the queue
        {
            int i, m;
        }
    }
}
```
 LinkedListNode<Node> p = new LinkedListNode<Node>(new Node());
for (i = 0, p = a, a = a.Next; p != null; p = a, a = a.Next)
{
    // Locate the starting point of the first possible match of the current p-node
    while(dQueue.ElementAt<LinkedListNode<Node>>(i).Value.begin<= p.Value.begin)
    {
        i++;
        if (i + 1 > length) // The sub-sets rooted at r are all matched
        {
            while (a.Value.begin < r.Value.end)
                a = a.Next;
            break;
        }
    // Connect the nodes that satisfy the condition from the start of the match in the queue
    for (m = i; m + 1 <= dQueue.Count && dQueue.ElementAt<LinkedListNode<Node>>(m).Value.begin < p.Value.end; m++)
    {
        append(p, dQueue.ElementAt<LinkedListNode<Node>>(m));
        if (a != null && dQueue.ElementAt<LinkedListNode<Node>>(m).Value.begin <= a.Value.begin)
        {
            i = m; // Record where the next match starts
        }
        if (a != null || (m + 1 > length && a.Value.begin > r.Value.end))
        {
            break;
        }
    }
    }
    dQueue.Clear(); // The nodes in the current sub-collection are all connected, and the queue is cleared.
    }
    showResult();
}

7. Conclusion
This paper mainly studies the XML document coding, query technology and query algorithm. The performance of XML structure join algorithm is the key to XML query. The current structure join algorithm can not effectively deal with various situations of structure connection, so it has important practical and theoretical significance for its research.

References
[1] World Wide Web Consortium. Extensible Markup Language (XML)1.0 (Fourth Edition). W3C Recommendation.16 August 2016.
[2] World Wide Web Consortium. XML Path Language (XPath) Version1.0. W3C Recommendation. 16 November 2009.
[3] World Wide Web Consortium. XML Path Language (XPath) Version 2.0. W3C Recommendation. 23 July 2014.
[4] Wan CX and Liu YS. Efficient supporting XML Query and Keyword Search in Relational Database Systems. In: proceeding of the 3rd International Conference on Web-Age Information Management. 11 August 2008.