IN4331 Web Data Management - Lab assignment 1

Team:
David Hoepelman
1521969
david.hoepelman@gmail.com

Gijs Kuijer
4094107
gijs.kuijer@gmail.com

Introduction

Our group choose chapter 6 from the Web Data Management book and completed all exercises and assignments in this chapter. There was only one big assignment in this chapter mainly consisting of parsing and evaluating C-TP trees. A brief discussion will follow.

Assignment code

The source code for our assignment is available in a public GitHub repository at https://github.com/dhoepelman/in4331-ex1

Assignment discussion

The assignment in chapter 6 started by creating an algorithm to evaluate C-TP tree patterns (instructions 1). The algorithm was already largely given in the chapter but had to be implemented. During implementation we encountered that the algorithm was not very complete and that we missed implementation of some classes, such as the PatternNode. Next to completing the missing pieces we also changed the algorithm slightly in how it worked as we thought it was clearer. We will give a high level overview of the algorithm.

StackEval is an algorithm that evaluates a C-TP pattern, but does this in a single linear traversal of the XML document (the SAX API is used). We represent the query as a tree of **PatternNode** objects. Each PatternNode object has 1 parent node (except the root) and any number children nodes. Each PatternNode has a **TPEStack**, which contains a stack of **Match**es. Each Match is a connection between a xml attribute/element node pre-order number and a PatternNode.

When StackEval encounters an opening tag we check the children of the current PatternNode (or the root node if there is no current node) to see if any match. If so we create a new match and place it on the TPEStack of the PatternNode. We then replace the current PatternNode with the one just encountered. This enforces the ancestor conditions for every PatternNode. Our algorithm works slightly differently from the StackEval in the book, as there recursively all descendants of PatternNode's are searched for a match.

When encountering a closing tag it first checks if this closes the current PatternNode (last PatternNode with successful match). If so it is checked if all the PatternNode children have a Match, i.e. if the descendant conditions for the PatternNode are fulfilled. If they are not the Match is deleted so that in the end only valid Matches remain.

To implement instruction 2, an algorithm to compute result tuples, we implemented a flag in the PatternNode that is set if we want to have its result. If this flag is set the algorithm, which is launched after evaluating the C-TP tree patterns, will take the PatternNode into account and will output it's preorder number into a table, thus creating a table who's rows are the tuples corresponding to the provided pattern.

Extension 1, providing support for wildcard patterns, is implemented by adding an extra criterion to the name comparison of the started element and the PatternNode's. If the PatternNode name was a '*' a match is added for the current element, whatever that elements name is. We also changed here that the name of the element is saved on the Match instead of the name of the PatternNode. By doing this we could output the actual name of the element that was matched in the results and did also not have to change the endElement method.

Extensions 2 and 3 had us extend the algorithm to make optional nodes available. We implemented this feature by adjusting the PatternNode class and including an optional flag. We also altered the endElement method to take this flag into account. If this flag was set the endElement method would not require the child to have been matched. In our result algorithm we simply inserted "null" in every spot where a value was not available. We did not have to check if the PatternNode is optional as the evaluation algorithm already took care of this.

The values of elements and attributes are taken care of by extension 4. We extended the PatternNode and Match with a valuePredicate. In the characters method of StackEval we saved the characters to the Match which was last opened if the preNumber corresponded. If the preNumber did not correspond it may not be filled with the value as the value might then be corresponding to an element inside the last matched element. It could also be that the element inside the last matched element is already closed. This would be valid XML but for XML databases this would not be valid. Therefore we do also not consider this case. The extension is further implemented by checking if the PatternNode valuePredicate and the Match valuePredicate are equal, and if not remove the match.

Extension 5 was implemented by letting algorithm 2 get the names of the matches, then after this iterate over the attributes and get their names and values and after this end an element tag, list the children or contents (such as text values) and place an end element tag. This function was implemented recursively to make it as efficient as possible.