Web Data Models Project Report

Streaming Algorithms for XPath

CHEN Hang

Implementation Analysis

This project is for implementing and analyzing efficient algorithms for fragments of XPath queries.

The format of the XML document is given like: 0(begin tag)|1(end tag) element. It constructs a complete XML. There are 2 implementation Algorithm: Streaming algorithm and LazyDFA algorithm, the first one deals with the simple query("//a/b") and the second one deals with the complex query("//a//a/b").

My implementation in Java has 3 principal part:

Main

It is the main function of this implementation. Firstly we will get 2 arguments from input with format: arg0 = file path, arg1 = query. If the input is not valid, the program will not continue and tell you the invalid input. If the input is correct, we will check that the query is a simple query or a complex query. If it's simple, creating an instance of the class ***StreamingAlgo*** to deal with the input, and if it's complex creating an instance of the class "LazyDFAAlgo".

Streaming algorithm

The class ***StreamingAlgo*** and ***lazyDFAAlgo*** both implement the interface ***AlgoXPath***.

The interface ***AlgoXPath*** has 3 functions: "*getQuery*" is for analyzing the input query; "*handleXML*" is for handling the input file according to the information of query, then find the goal node; "*getResult*" is for outputing the result.

StreamingAlgo

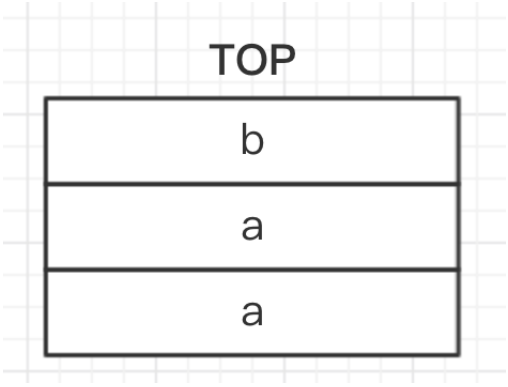
The idea of the algorithm is that:

- Using "*getQuery*" we can get a list of query elements.

  For example: for "//a/a/b", we have [a,a,b].

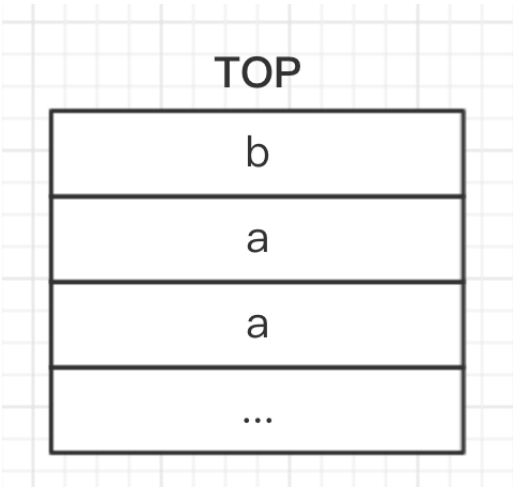
- Then handling the input file with "*handleXML*", we create a stack to record the elements. If we read "0 elememt", we push this into the stack. If it's "1 element", we pop from the stack. That makes sure that the elements in the stack form the path of the current element in the XML file.

For example: we read "0 a \n 0 a \n 0 b". The current element is "b" and it's XML path is "a-a-b". So the stack is like that:



- After each push in stack, we will check it with the query list, to see if the top elements can match the query list. If the top elements have the same elements with the same order, this top one element matches the query, we add it's number to the result list.

  For example: the match condition is: query: "//a/a/b" and the stack is:



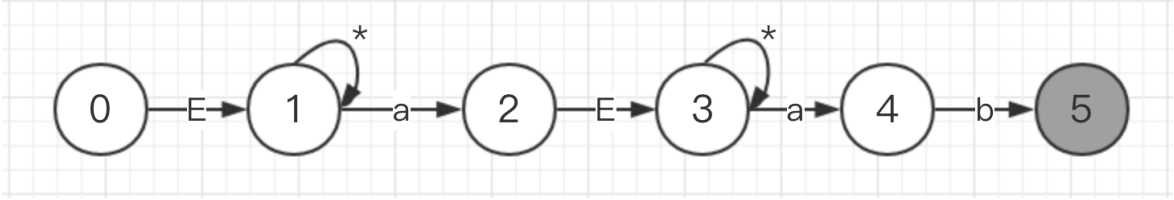
- When reading the file over, using "*getResult*" to print the corresponding elements' numbers.

LazyDFA algorithm

The main idea is that:

- Creating a NFA according to the query with the function "*getQuery*",

  For example, for the query "//a//a/b", we get NFA model:



  The condition "E" means the empty String, a state with "E" condition can jump to the next state directly. A state with "*" condition can jump to itself. The number 5 is the end node.

  We define a class ***NFANode*** to stand for each node of NFA. For every instance of NFANode,

there are state number(int), is the state the end state(boolean), does the state have the transition to itself(boolean), it's edge(class ***NEdge*** which contains transition condition and target state).

We also define a class ***NFAFactory*** which is a factory class for NFA model, it will return a HashMap. All of the NFAnodes are stored in this HashMap.

- Reading the input file, creating a lazyDFA on the base of NFA with "*handleXML*".

  We define a class ***DFANode*** to stand each node of DFA. For every instance of DFANode, there are state number(int), is the state the end state(boolean), the list of corresponding NFA states(integer list). It will take all same type of states of NFA.

  For example, the first node of DFA state list: [0,1] because the first state of NFA model is "0" and it's transition condition to state "1" is "E". So we store them into a ***DFANode***. Then the input file comes a element, we will traverse all states in the list, if we find one has this element as it's transition condition, we store the transition target into next ***DFANode***. Just like that when it comes "a", for the first ***DFANode***, its NFAlist is [0,1], so according to the NFA model, we have the second ***DFANode*** with the NFAlist [1,2].

  We have a function "*match*" to check the NFAlist for each step of reading element. If there is a end state in the NFAlist for next ***DFANode***, we will store this element into the result list.

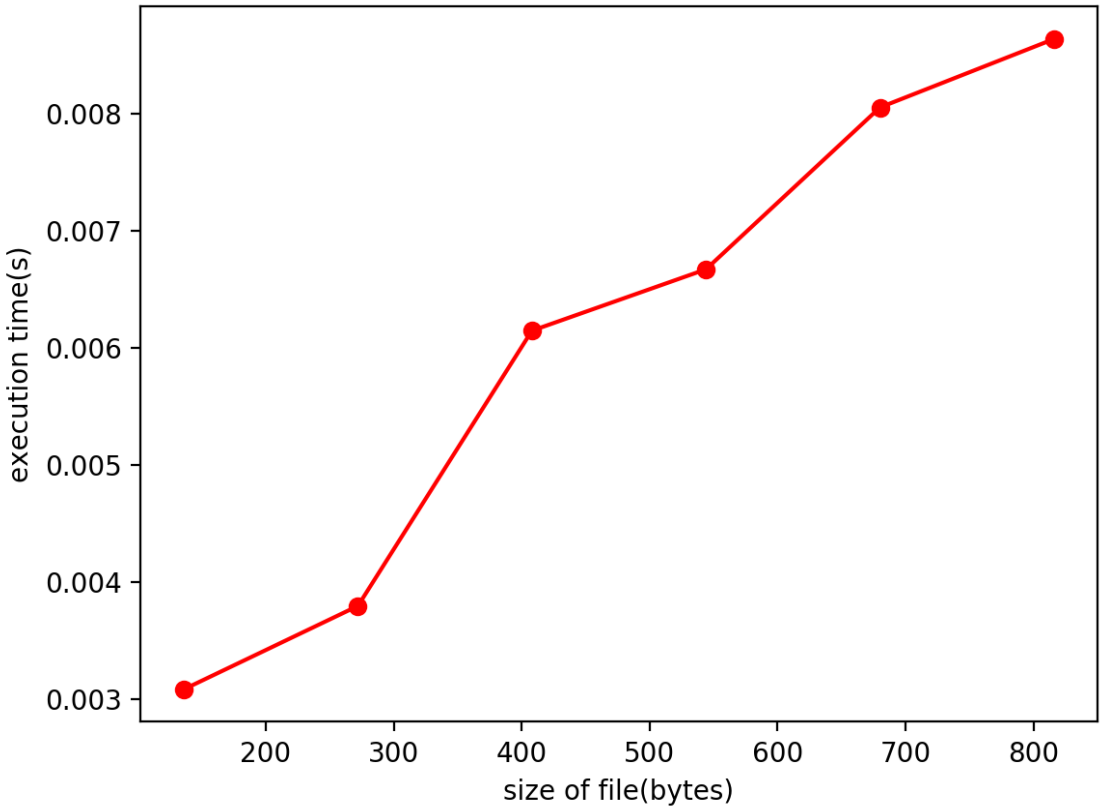
- When reading the file over, using "*getResult*" to print the corresponding elements' numbers.

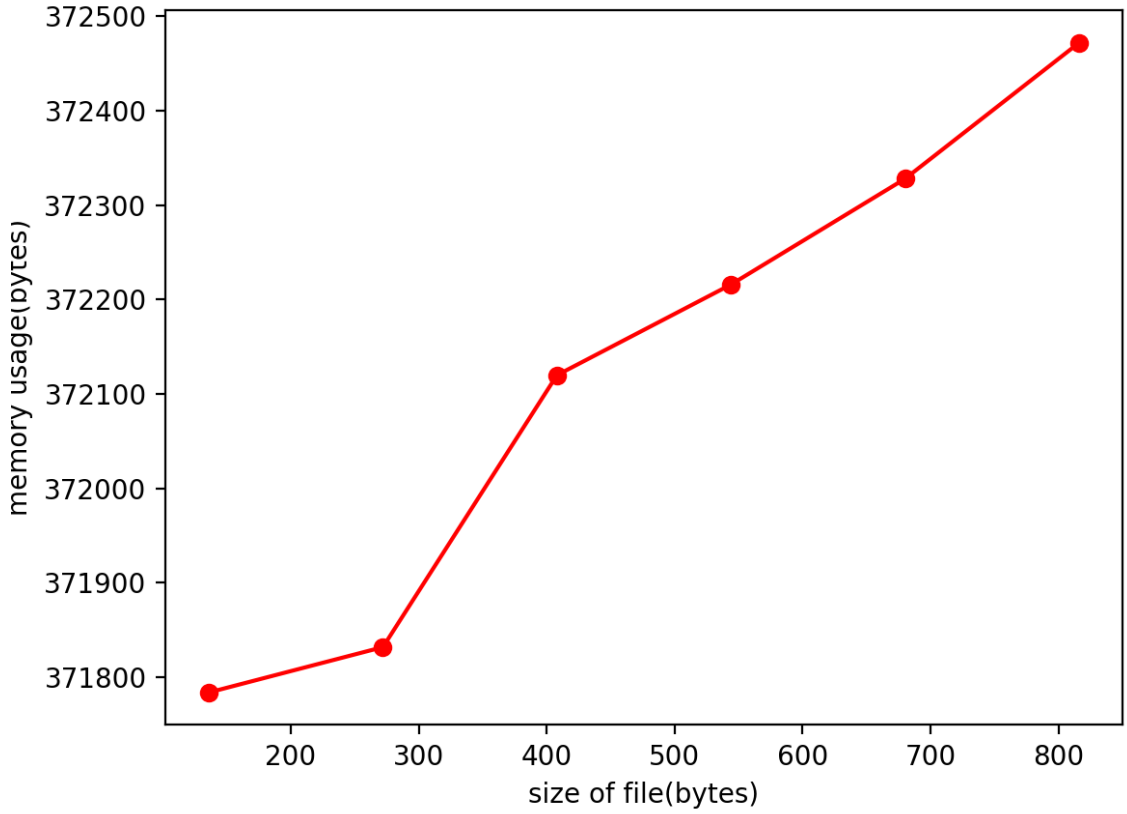
The usage of stack will facilite the query, which avoids loading data in memory and taking much, and could be compared with query information directly.

Experimental Section

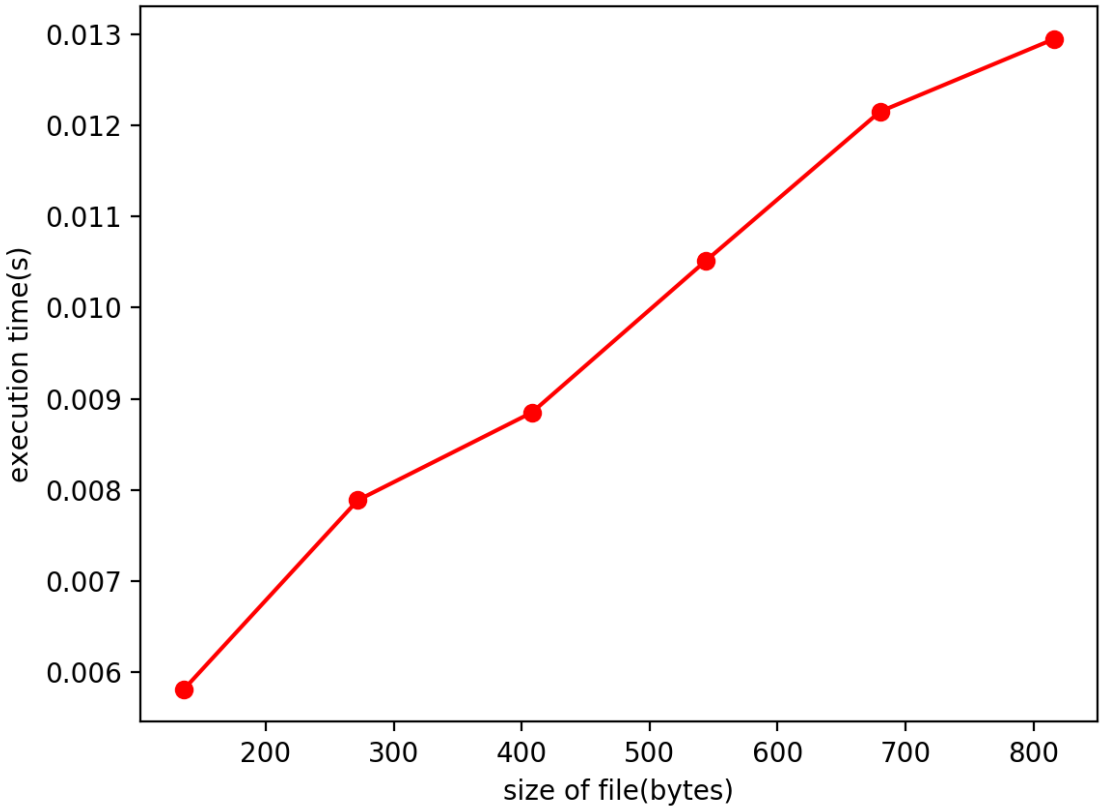
For the experimental evaluation analysis, we have created 6 input file with the size(136bytes, 272bytes, 408 bytes, 544 bytes, 680 bytes, 816 bytes).

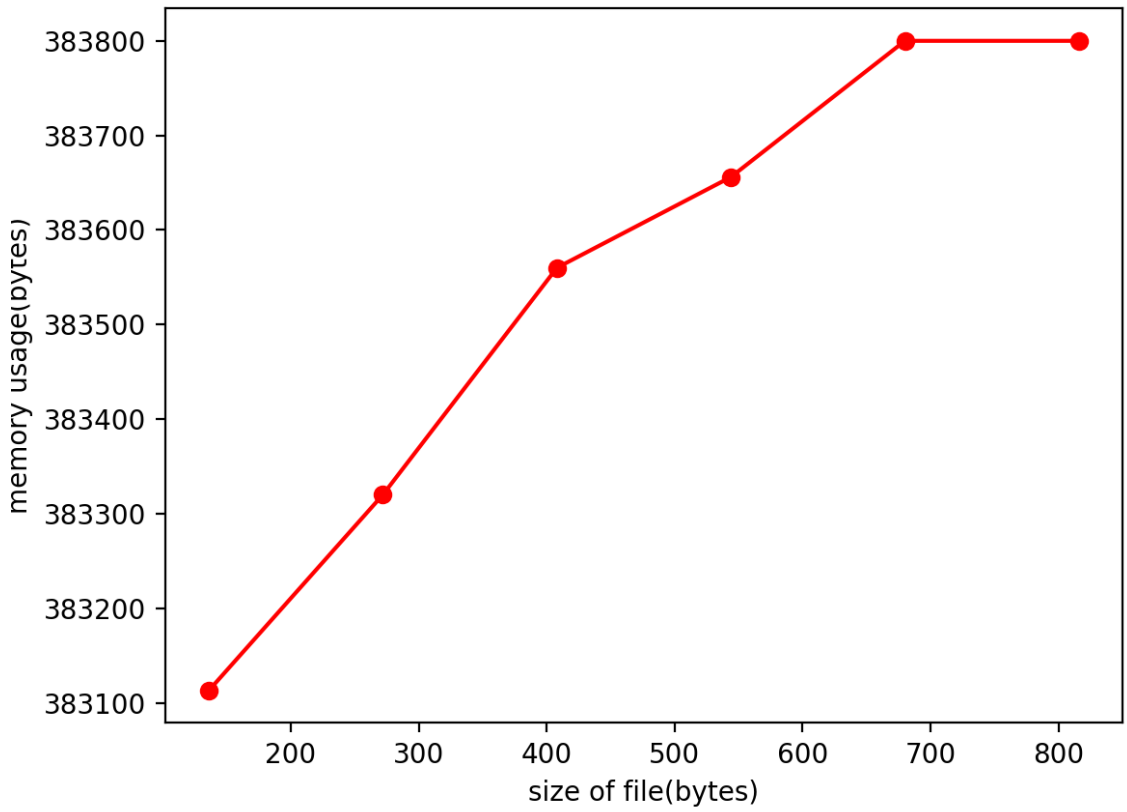
Firstly, we test with simple query, we take "//a/a/b", the results are:





Then, we test with complex query("//a//a/b"), the result are:


0.013
0.012
0.011
0.010
0.009
0.008
0.007
0.006
execution time(s)
200
300
400
500
600
700
800
size of file(bytes)


383800
383700
383600
383500
383400
383300
383200
383100
memory usage(bytes)
200
300
400
500
600
700
800
size of file(bytes)