OUMOUSS EL MEHDI (M2 - D&K)
November 20, 2016

Programming Project: Streaming Algorithms for XPath Web Data Models

Implementation analysis

NB: For executing my implementation, please refer to the “read.me” document in the project directory.

Assignment 1
Implement a streaming algorithm for XPath queries of the form

$$//e_1/e_2/\cdots/e_n,$$

Where e_i are element names.

Implementation was done based on t**he Knuth–Morris–Pratt (KMP)** algorithm, which is a string searching algorithm that helps searching for occurrences of a "word" within a main “text string” bypassing re-examination of previously matched characters whenever a mismatch occurs.

KMP-Prefix (Pattern P)	KMP-Matcher (Text T, Pattern P)

begin
  m ← |P| // pattern length
  Π ← 0 // Prefix table
  i ← 0
  for j = 2 upto m step 1 do
    while i > 0 and P[i+1] ≠ P[j] do
      i ← Π[i]
    if P[i+1] = P[j] then
      i ← i+1
    Π← i
  return Π
end
	begin
	  n ← |T|
	  m ← |P| // pattern length
	  Π ← KMP-Prefix(P) // Prefix table
	  i ← 0
	  for j = 1 upto m step 1 do
	    while i > 0 and P[i+1] ≠ T[j] do
	      i ← Π[i]
	    if P[i+1] = T[j] then
	      i ← i+1
	    if i = m then
	      output (j-m)
	      i ← Π
	end

Table 1. KMP pseudocode

Assignment 2
Implement a streaming algorithm for queries of the form:

$$//p_1//p_2//\cdots//p_n ,$$

Where each p_i is a path of the form:

$$//e_{i1}/e_{i2}/\cdots/e_{im} ,$$

And e_{ij} are element names.

As for the second assignment, we tried to build over what we already have done in assignment 1. In other words, we are implementing the structure in assignment 1 but recursively over chunks of the text in the stream.

As a whole, the implementation can be summarize as follow:

```
Input: Text, Query
Output: Matches
while not at end of file
    read line
    if startElement then
        push to stack
        Context Initialisation
        Maching process
    else
        pop from stack
    end
end
```

Figure 1. Xpath Streaming Algorithm

As the elements comes, we have the matching process which determines if we have a match or not. Here we are going to tackle the general case, which is queries of the form: $//p_1//p_2//\cdots//p_n$

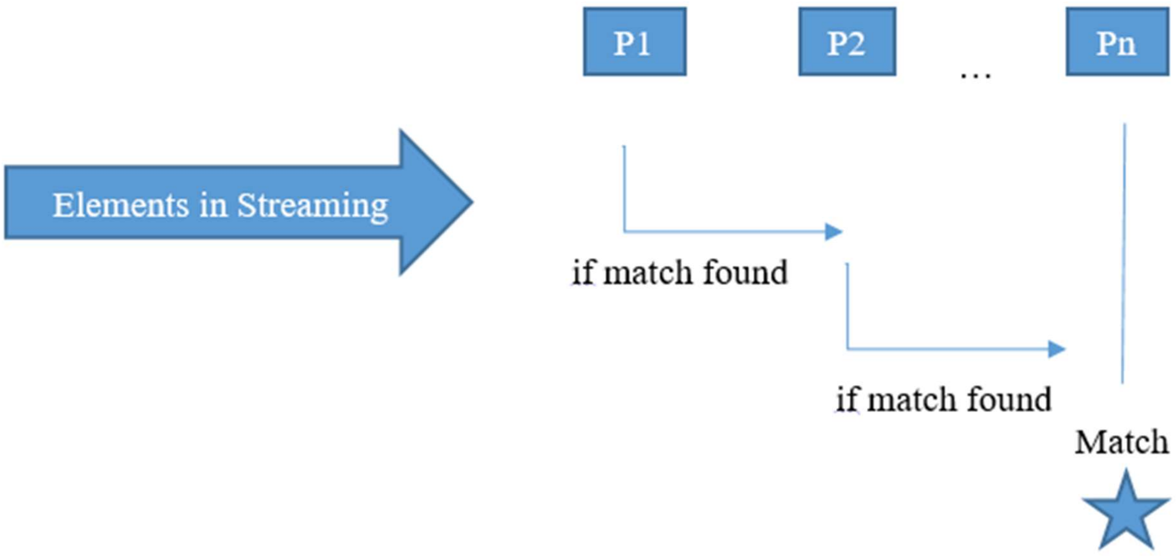


Figure 2. The matching cascade

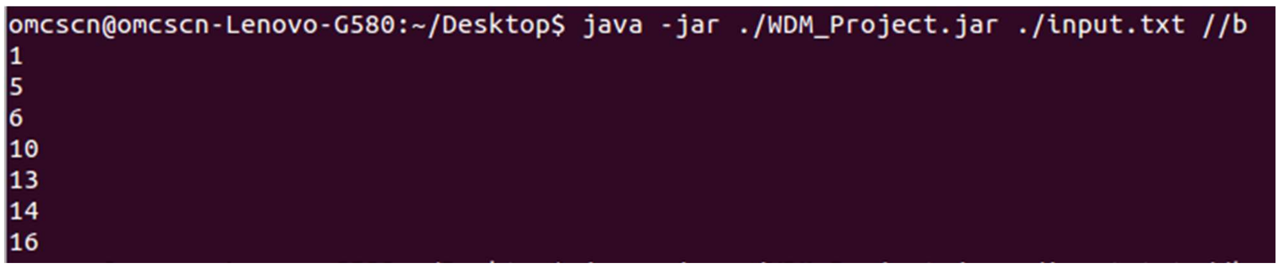
The matching process is implemented here as a recursive method, that calls every time the KMP algorithm. Concerning the context initialization (or states context), usually a two dimensional array is used, as a matter of preference, here we use an array of LinkedList. In the structure, we store the different states of matches we have, and this is passed to the following pattern in order to know from where the match process should be carried.

Consistency:
The implementation here presented answers correctly to all the next queries (with regard to the input.txt file that comes with project):

Query	Results
//a	0, 2, 3,4,8,9,11,12, 15
//b	1, 5, 6, 10, 13, 14, 16
//c	7
//a/b	1, 5, 6, 10, 13, 14, 16
//a/b//a	2, 3,4, 15
//a/b//b	5, 6, 16
//a/b//a/b	5, 6, 16
//a//a//a	3, 4, 9, 12, 15
//a//a//b	5, 6, 10, 13, 14, 16

Table 2. Queries and results

Example of execution:



```
omcscn@omcscn-Lenovo-G580:~/Desktop$ java -jar ./WDM_Project.jar ./input.txt //b
1
5
6
10
13
14
16
```

Image 1. Command line screenshot

Experimental section

Experiment 1:
For the same input file, relation between simple paths lengths and execution time

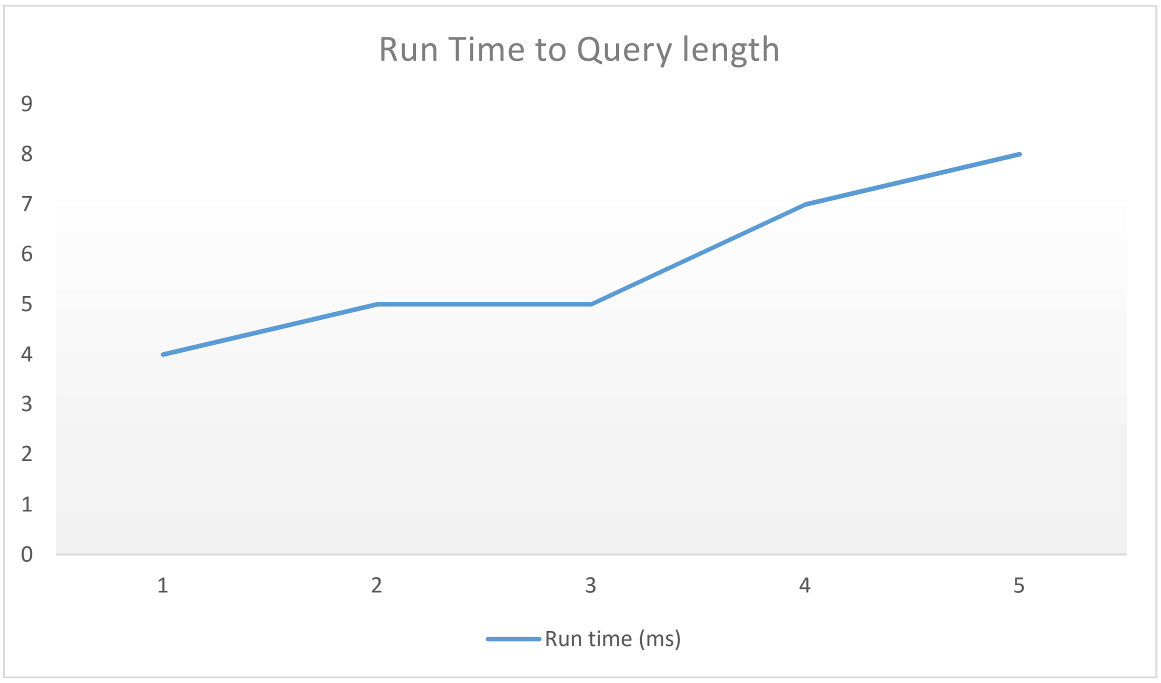


Figure 3. Run Time to Query length

Experiment 2: How execution time with regard to query complexity?

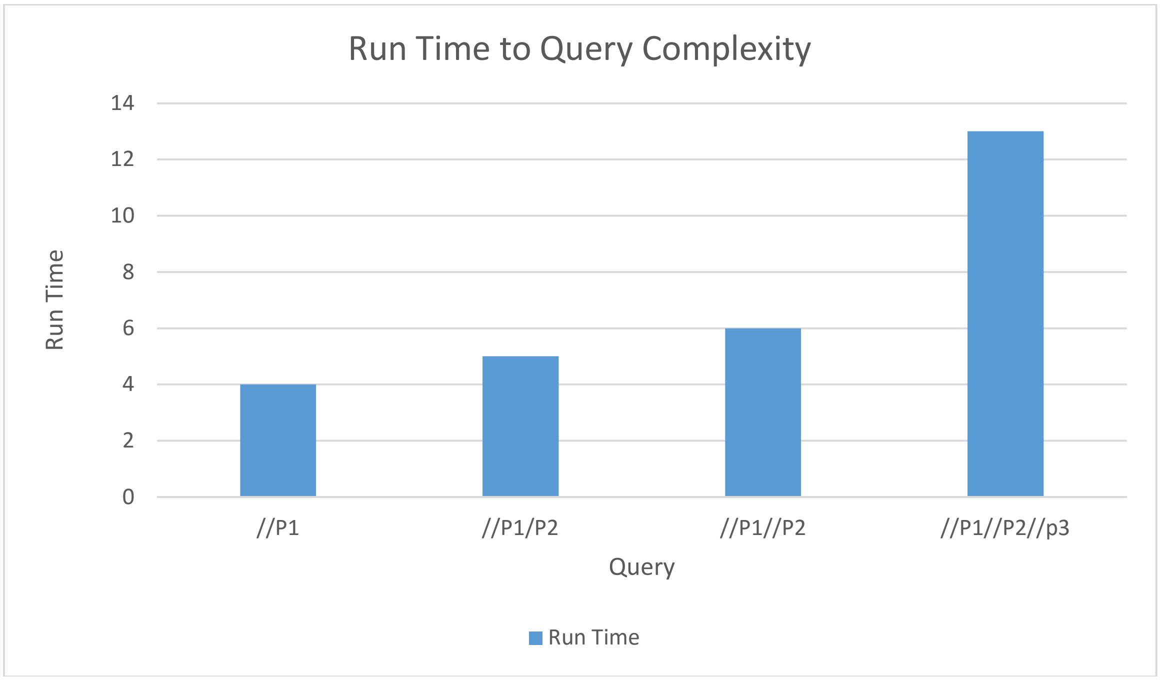


Figure 4. Run Time to Query Complexity

NB: It is worth to mention that even though the results shown above follow up with the intuition we may have. However, it is not always consistent. For example, we may have a short query, but sense it doesn't have a match at all, therefore we are making a parse over the whole document.

Experiment 3: For the same Xpath query, relation between document size and run time

	Doc1	Doc2	Doc3
Size (bytes)	80	168	224
Run time (ms)	3	5	6

Table 3. Document size to run time.

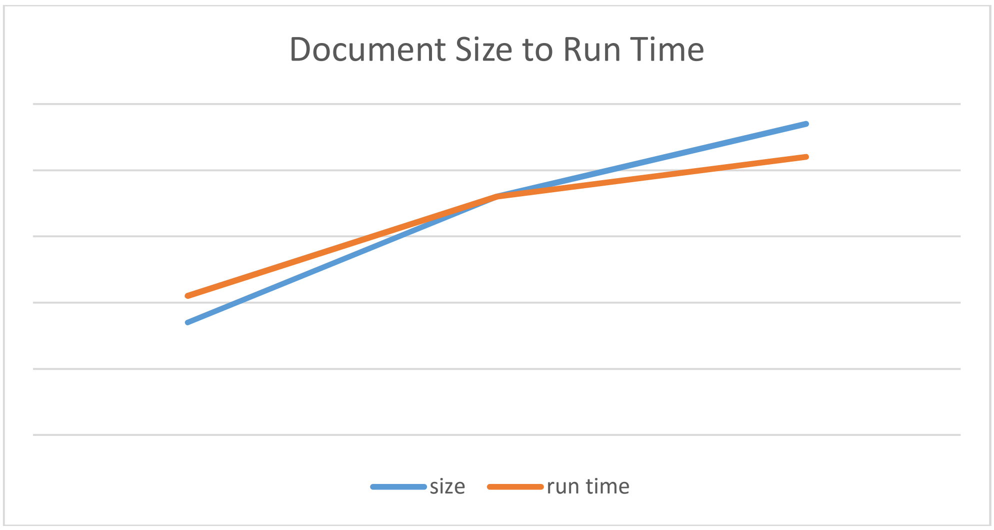


Figure 5. Document Size to Run Time evaluation

References

- [GGM + 04] Todd J Green, Ashish Gupta, Gerome Miklau, Makoto Onizuka, and Dan Suciu. Processing XML streams with deterministic automata and stream indexes. ACM TODS, 29(4):752–788, 2004.

- Knuth–Morris–Pratt(KMP) Pattern Matching(Substring search)
  https://www.youtube.com/watch?v=GTJr8OvyEVQ

- Knuth–Morris–Pratt algorithm
  https://www.youtube.com/watch?v=5i7oKodCRJo