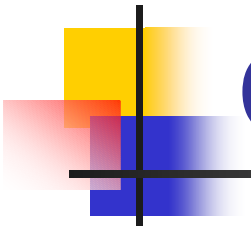
串流XML資料之路徑查詢處理

講者: 黃文彬

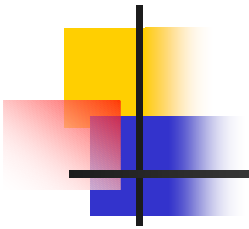
作者: 陳耀輝, 黃文彬, 何銘啓, 余美伶, 曾柏銜

嘉義大學資訊工程系



Outline

- Introduction
- Related work
- Our method
  - PATH algorithm (example)
  - TWIG algorithm (example)
- Experiments

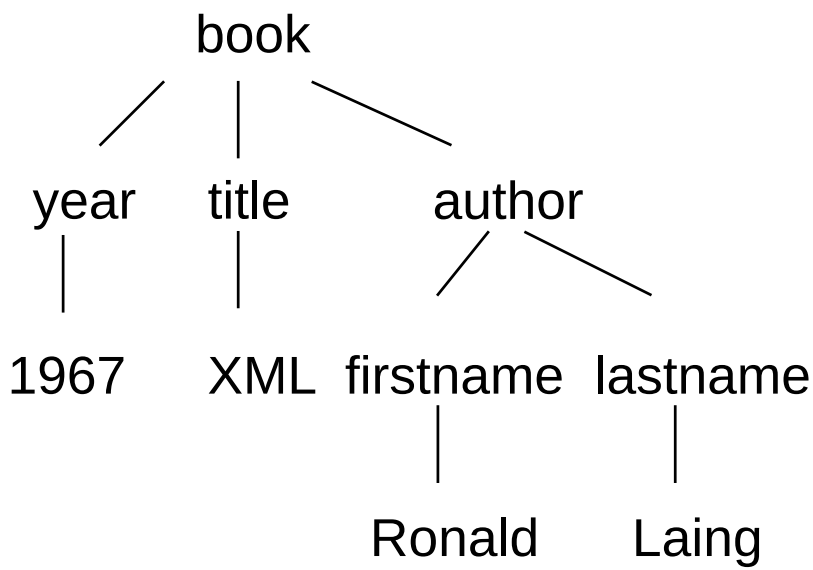


Introduction

- eXtensible Markup Language (XML)
  - The universal format for structured documents and data on the web
- XML documents
  - Syntax, no abstract model
  - Documents, elements and attributes
  - Ordered tree, nested, hierarchically organized structure

XML Architecture

```
<book year = “1967” >
   <title> XML </title>
   <author>
      <firstname> Ronald
      </firstname>
      <lastname> Laing
      </lastname>
   </author>
</book>
```

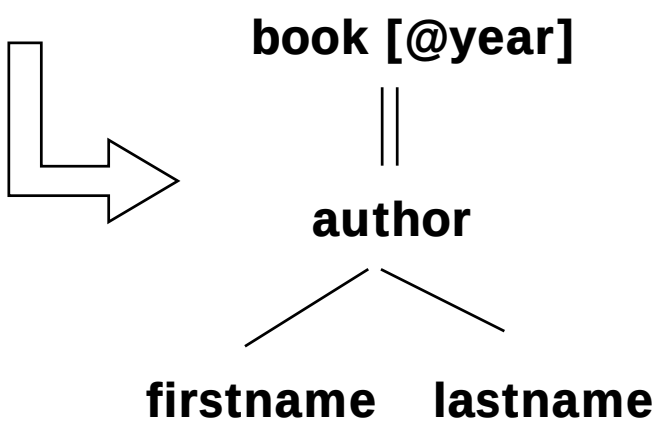


Stream XML data

<book><title></title><author> … </author></book>

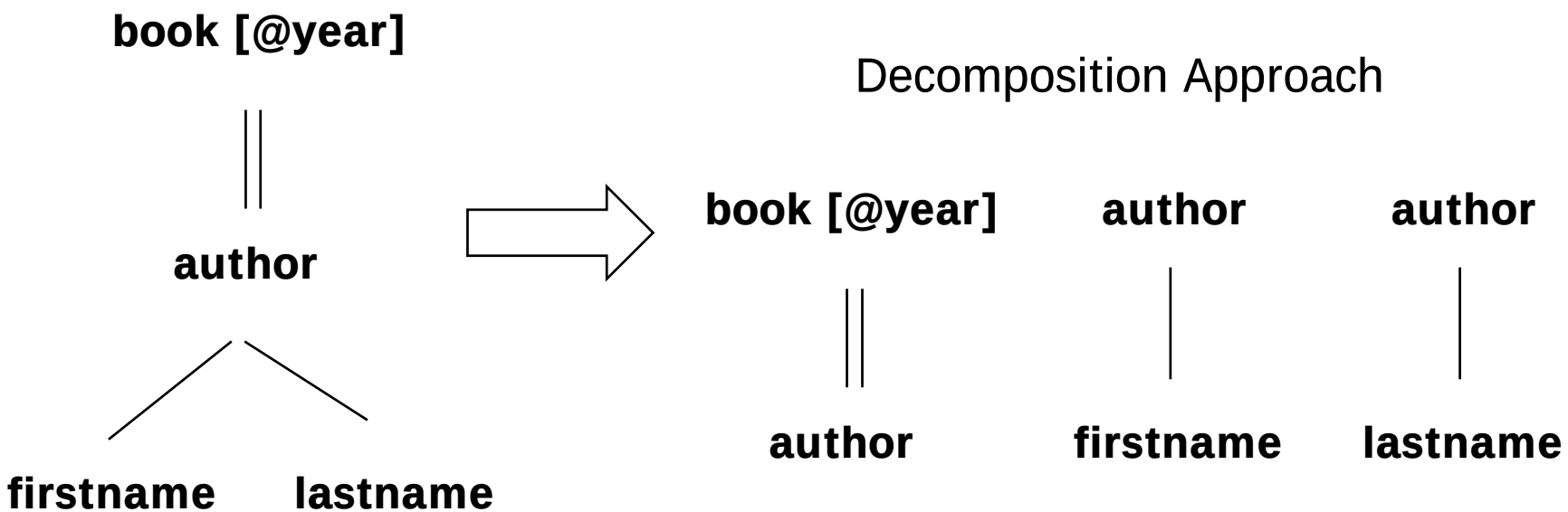
XML Query Processing

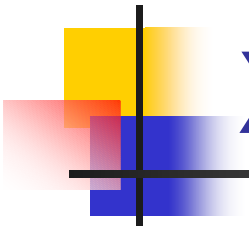
- XPath query language
  - Using a path expression
  - Addressing parts of an XML document
  - Example
    - book[@year]//author[firstname]/lastname



XML Query Processing (cont.)

- Locating all occurrences of a twig pattern is a core operation in XML query processing.

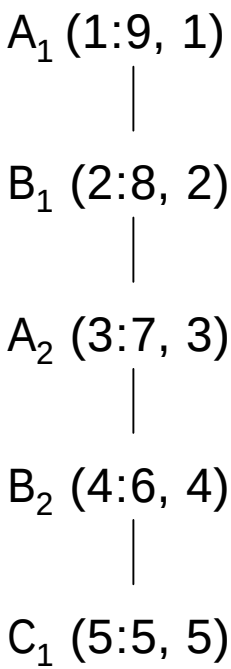




XML Query Processing (cont.)

- Disadvantage of the decomposition approach:
  - Intermediate result sizes can get very large
- Holistic twig join approach for matching XML query twig patterns [Bruno et al., SIGMOD'02]
  - Multiple stacks
  - Using pointer to save storage space

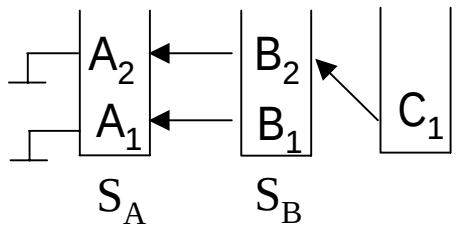
XML Query Processing (cont.)



(a) Data

A//B//C

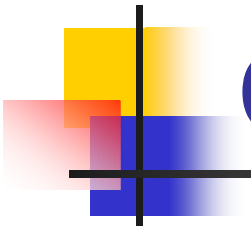
(b) Query



(c) Multiple stacks

$A_1\ B_1\ C_1$
$A_1\ B_2\ C_1$
$A_2\ B_2\ C_1$

(d) Query results



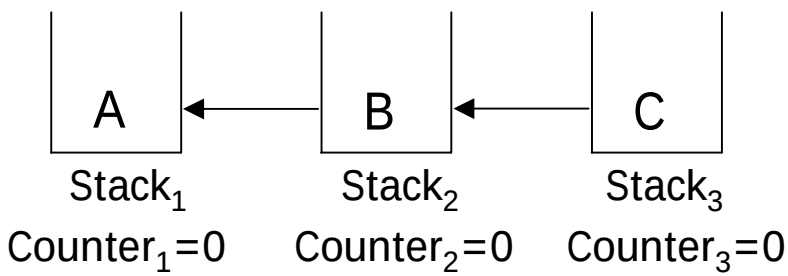
Outline of Our method

- Our method can process streaming XML data
- Less intermediate results are stored
- Like Holistic Twig Join algorithm, but:
  - We do not index source data
  - We directly process twig queries
- We also can handle duplicated data nodes

PATH algorithm

- We use depth first search to get the element order of query, then each element in the query is assigned a counter and a stack
- Counter:
  - the node is satisfied which query element
- Stack:
  - store intermediate result, save storage space
- Pointer:
  - record represent the relationship between stacks
  - reconstruct result

Query: A // B / C



Example (PATH algorithm)

Streaming data:

$<A_1><B_1><A_2><A_3></A_3></A_2></B_1></A_1>$

Query:

A // B / A

Tree representation: (data)

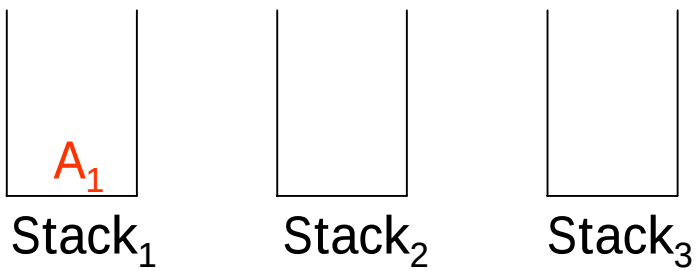
(query)

Example (PATH algorithm)

Data: <A_1><B_1><A_2><A_3></A_3></A_2></B_1></A_1>

Query: A // B / A

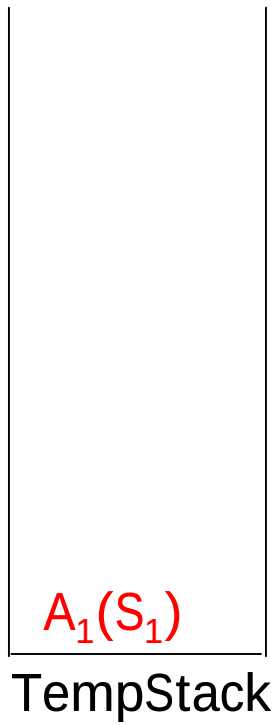
Read <A_1>



$Counter_1 = 1$

$Counter_2 = 0$

$Counter_3 = 0$

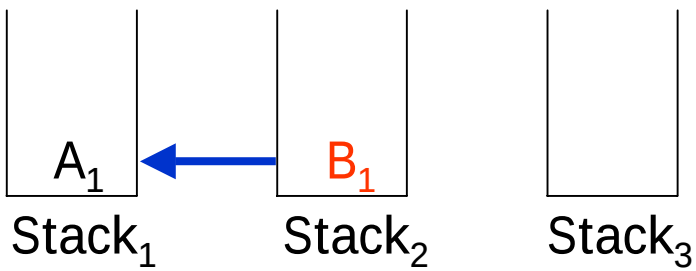


Example (PATH algorithm)

Data: <A_1><B_1><A_2><A_3></A_3></A_2></B_1></A_1>

Query: A // B / A

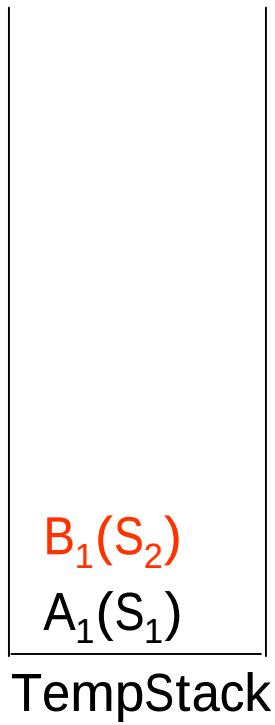
Read <B_1>



$Counter_1 = 1$

$Counter_2 = 1$

$Counter_3 = 0$

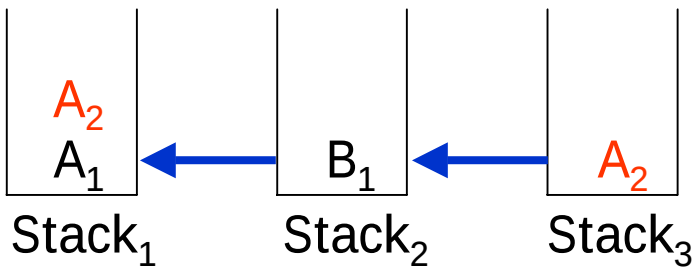


Example (PATH algorithm)

Data: <A_1><B_1><A_2><A_3></A_3></A_2></B_1></A_1>

Query: A // B / A

Read <A_2>

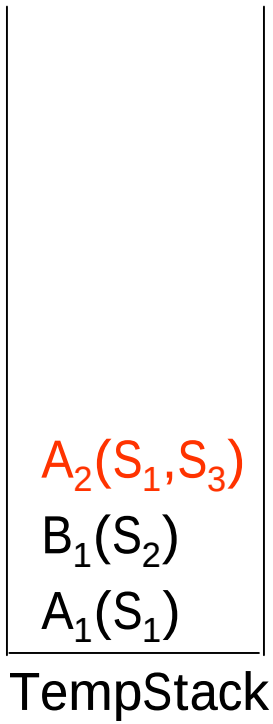


Counter$_1$ = 2

Counter$_2$ = 1

Counter$_3$ = 1

Answer: A_1 B_1 A_2

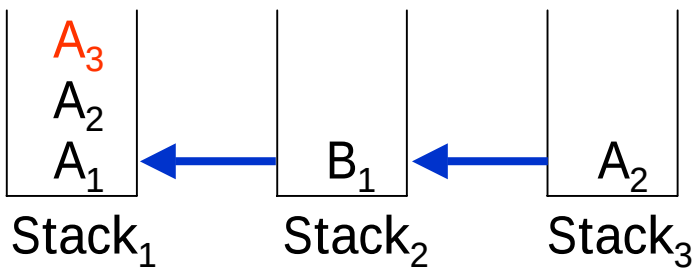


Example (PATH algorithm)

Data: <A_1><B_1><A_2><A_3></A_3></A_2></B_1></A_1>

Query: A // B / A

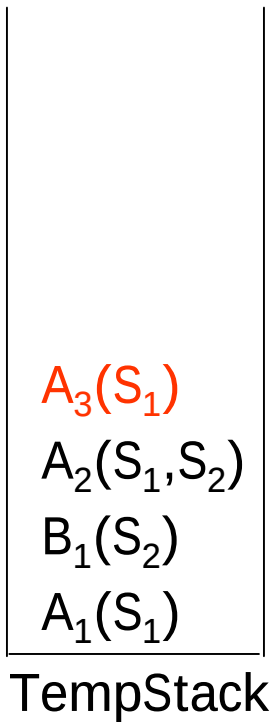
Read <A_3>



$Counter_1 = 3$

$Counter_2 = 1$

$Counter_3 = 1$

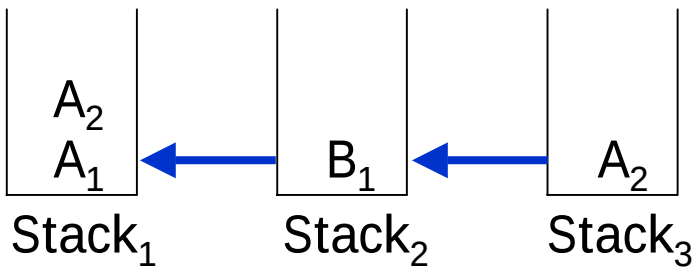


Example (PATH algorithm)

Data: <A_1><B_1><A_2><A_3></A_3></A_2></B_1></A_1>

Query: A // B / A

Read </A_3>

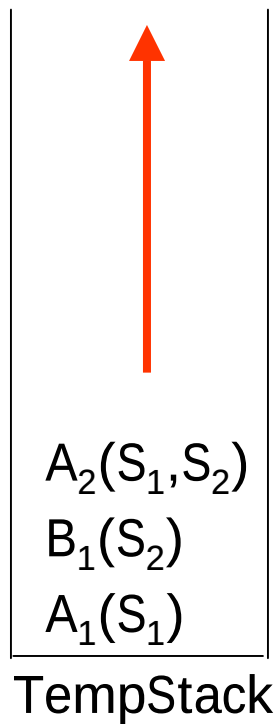


$Counter_1 = 2$

$Counter_2 = 1$

$Counter_3 = 1$

Pop $A_3(S_1)$

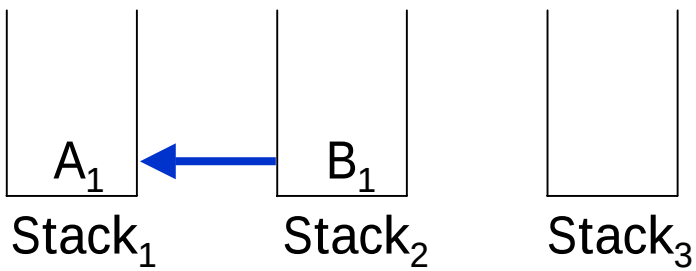


Example (PATH algorithm)

Data: <A_1><B_1><A_2><A_3></A_3></A_2></B_1></A_1>

Query: A // B / A

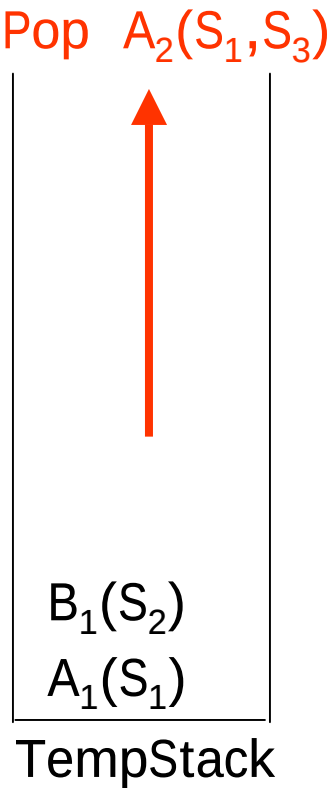
Read </A_2>



$Counter_1 = 1$

$Counter_2 = 1$

$Counter_3 = 0$

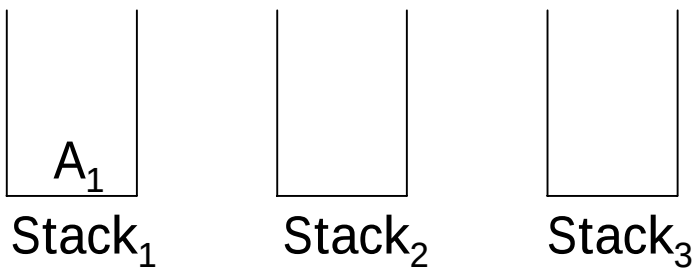


Example (PATH algorithm)

Data: <A_1><B_1><A_2><A_3></A_3></A_2></B_1></A_1>

Query: A // B / A

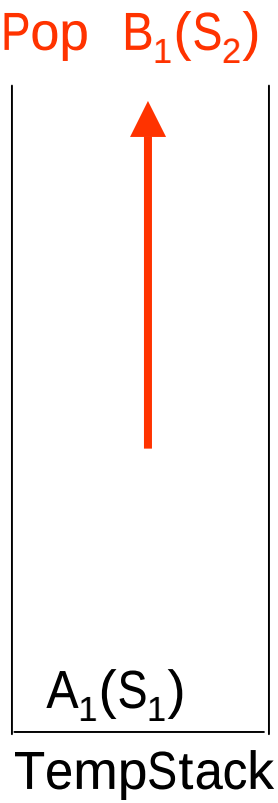
Read </B_1>



$Counter_1 = 1$

$Counter_2 = 0$

$Counter_3 = 0$

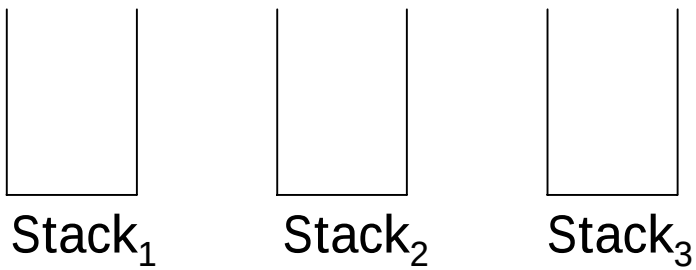


Example (PATH algorithm)

Data: <A_1><B_1><A_2><A_3></A_3></A_2></B_1></A_1>

Query: A // B / A

Read </A_1>

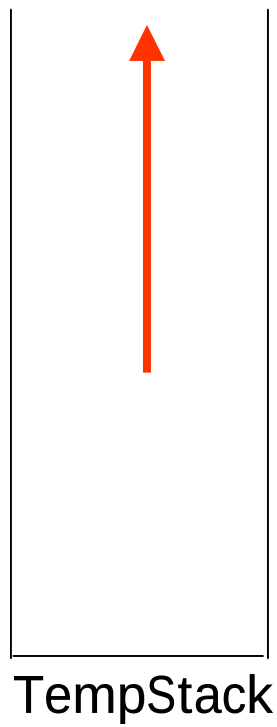


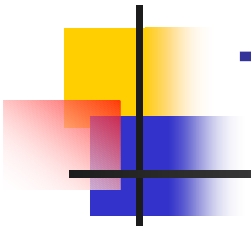
$Counter_1 = 0$

$Counter_2 = 0$

$Counter_3 = 0$

Pop $A_1(S_1)$

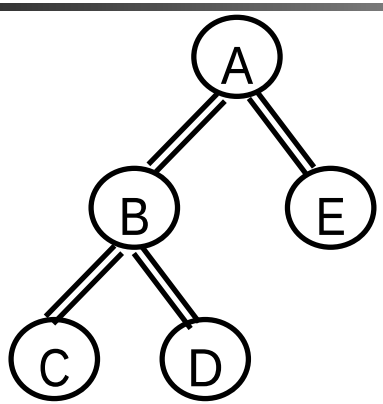




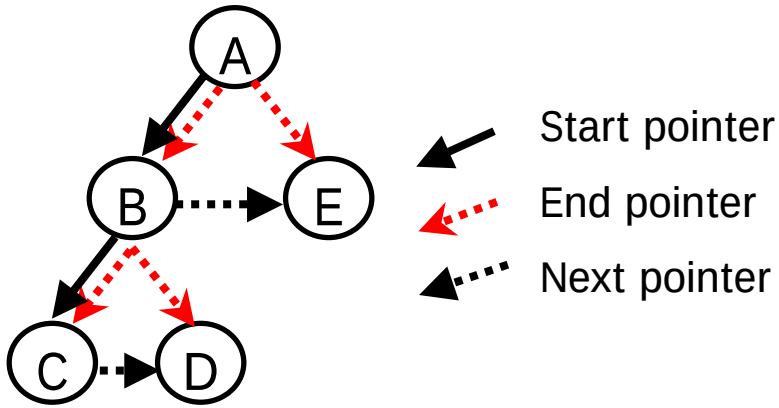
TWIG algorithm

- Counters and stacks are the same as PATH algorithm
- Pointer:
  - record represent the relationship between stacks
  - follow different pointers to reconstruct result

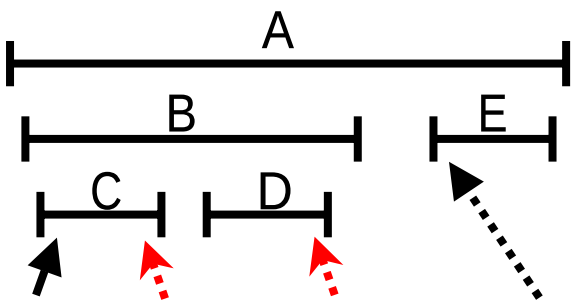
TWIG algorithm



(a) Twig query

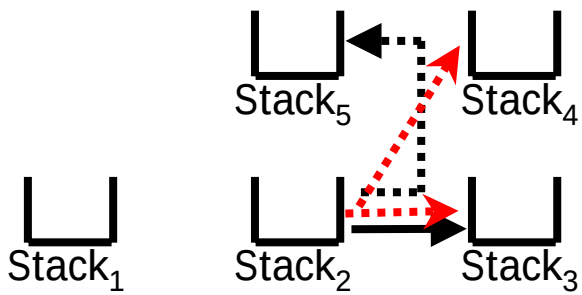


(b) Pointer structures of the twig query



(start_pt, end_1_pt, end_2_pt, next_pt, level)

(c) Pointer structures for the B element



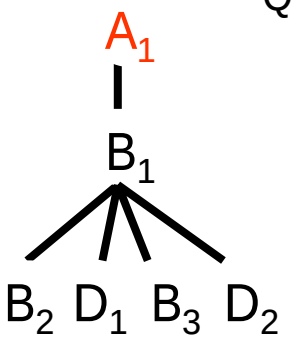
(d) Stack relationship for the B element

Example (TWIG algorithm)

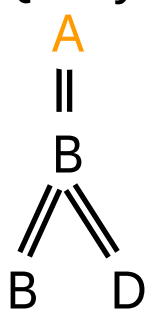
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

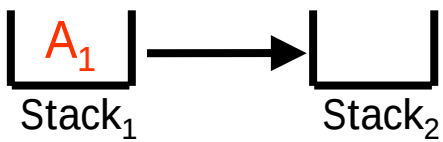


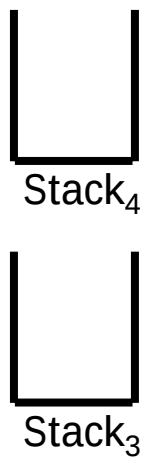
Query:

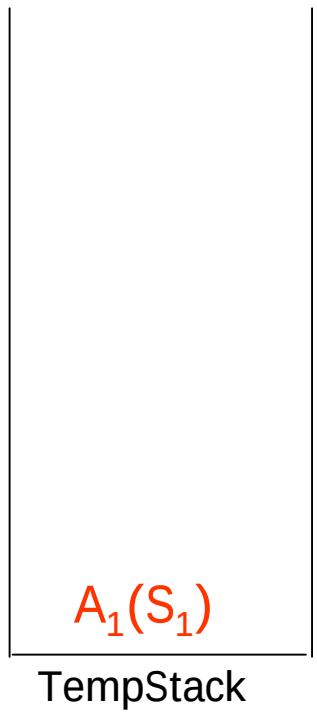


Read <A_1>

$Counter_1=1$
$Counter_2=0$
$Counter_3=0$
$Counter_4=0$





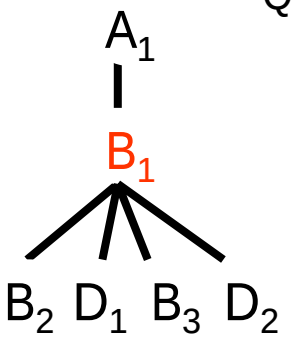


Example (TWIG algorithm)

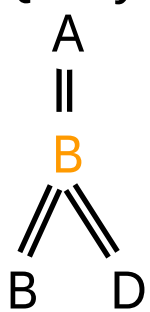
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

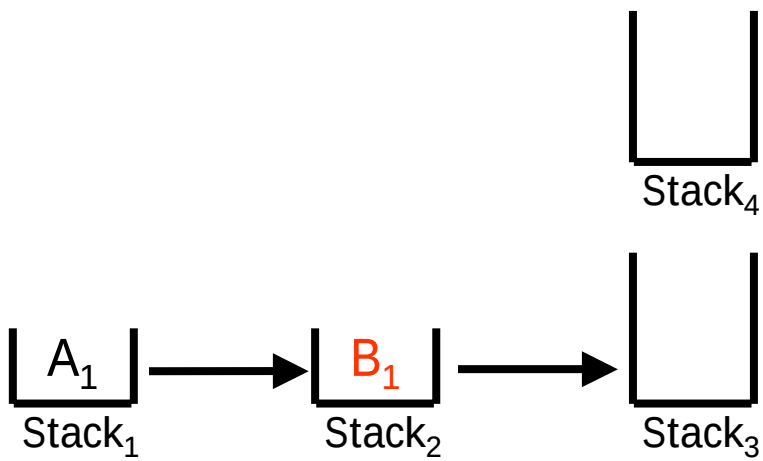


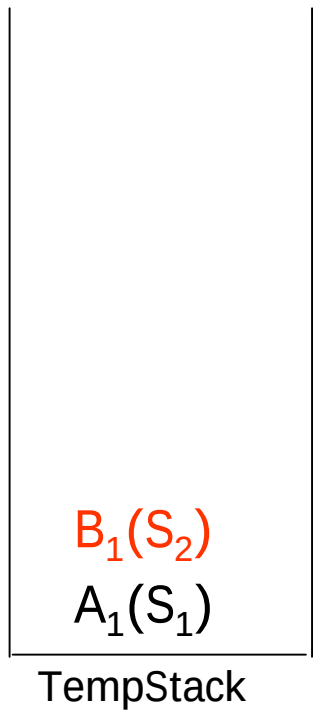
Query:



Read <B_1>

$Counter_1$=1
$Counter_2$=1
$Counter_3$=0
$Counter_4$=0



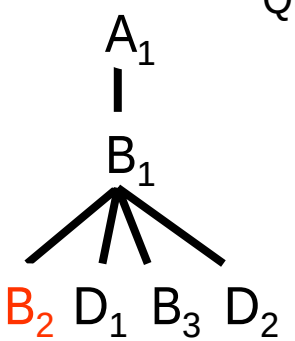


Example (TWIG algorithm)

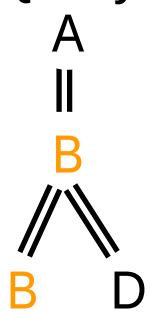
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

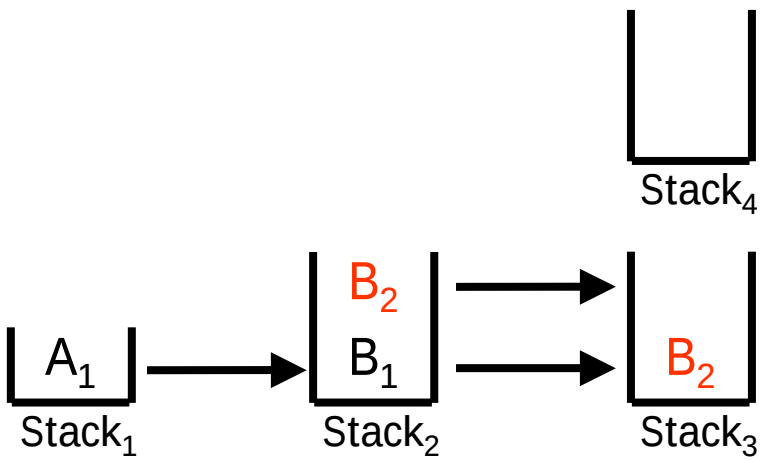


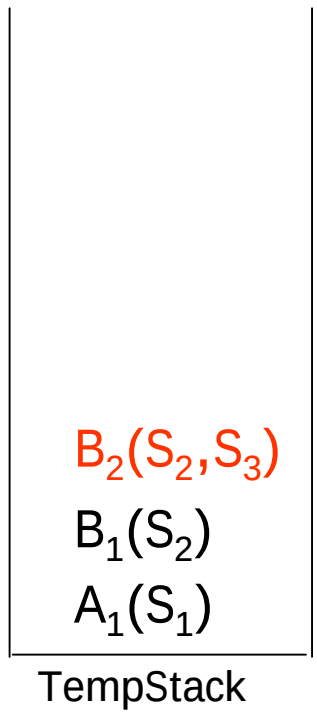
Query:



Read <B_2>

Counter$_1$=1
Counter$_2$=2
Counter$_3$=1
Counter$_4$=0



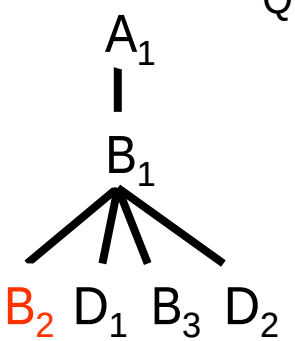


Example (TWIG algorithm)

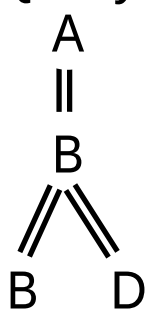
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:



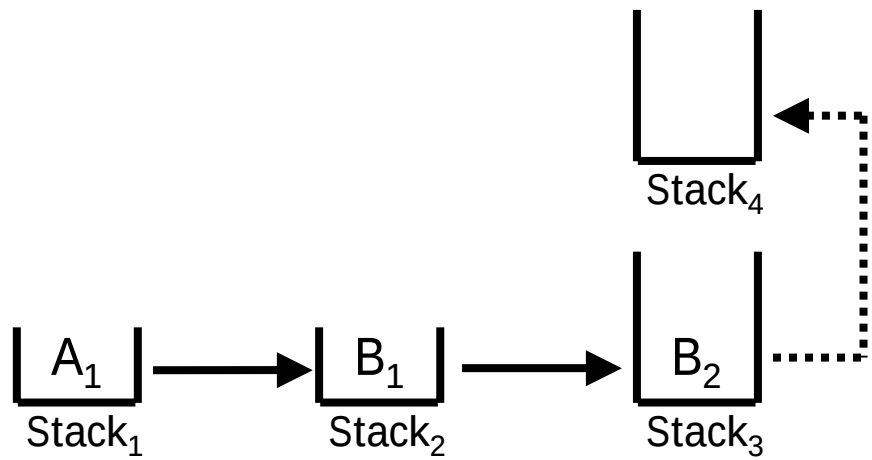
Query:

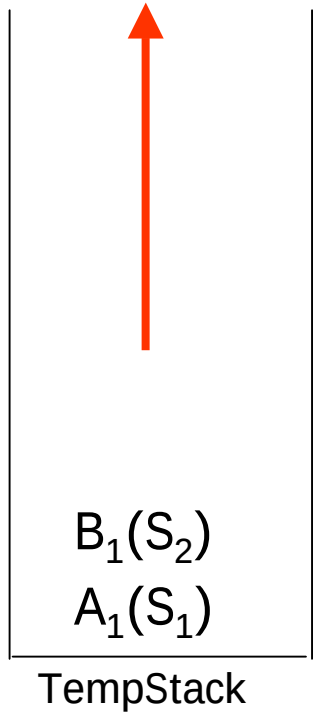


Read </B_2>

$Counter_1$=1
$Counter_2$=1
$Counter_3$=0
$Counter_4$=0

Pop $B_2(S_3)$



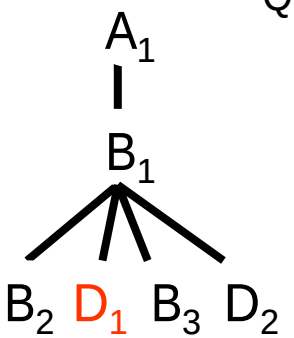


Example (TWIG algorithm)

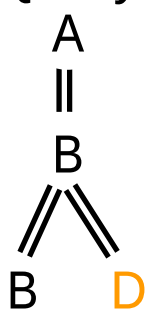
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

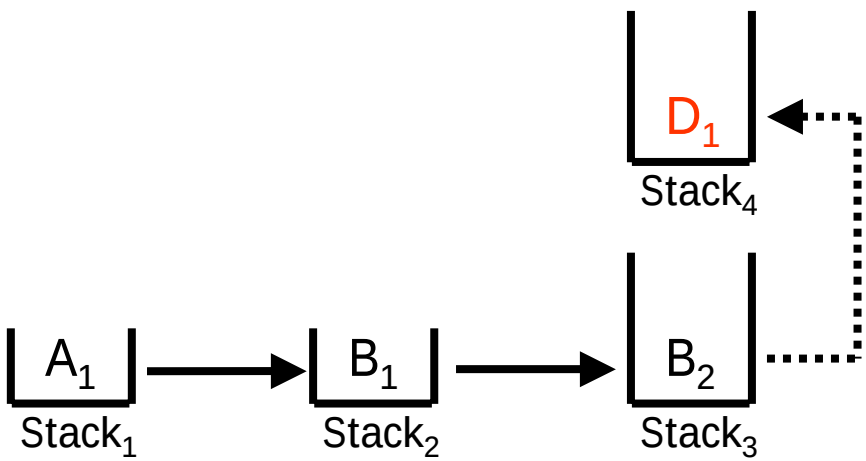


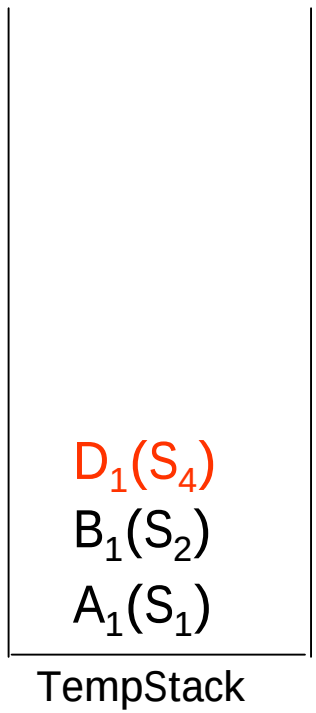
Query:



Read <D_1>

$Counter_1$=1
$Counter_2$=1
$Counter_3$=0
$Counter_4$=1



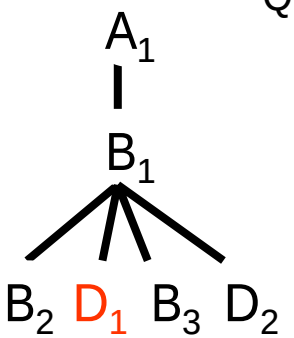


Example (TWIG algorithm)

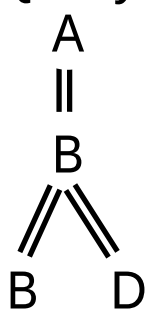
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

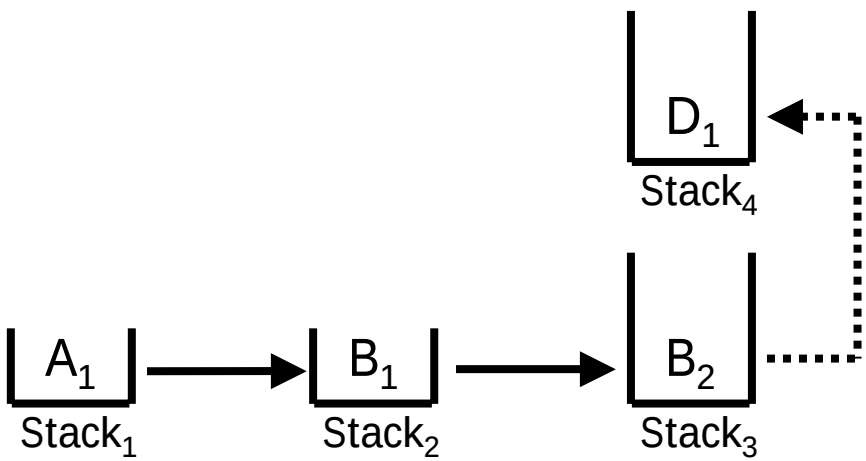


Query:

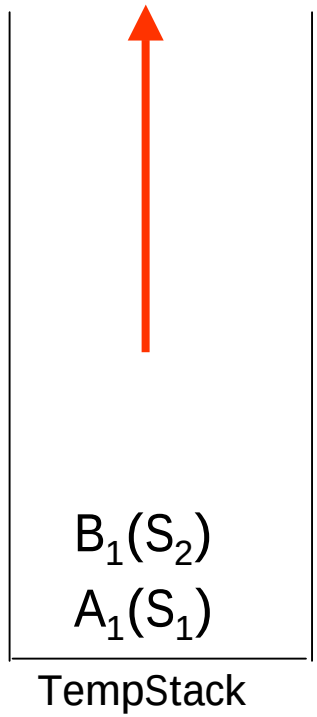


Read </D_1>

$Counter_1$=1
$Counter_2$=1
$Counter_3$=0
$Counter_4$=0



Pop $D_1(S_4)$

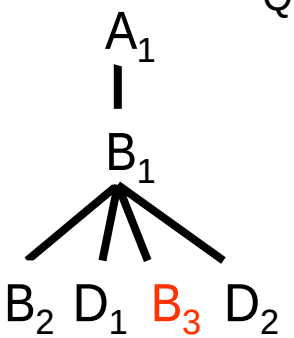


Example (TWIG algorithm)

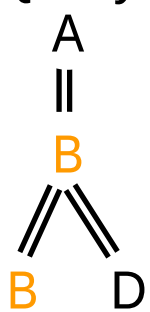
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

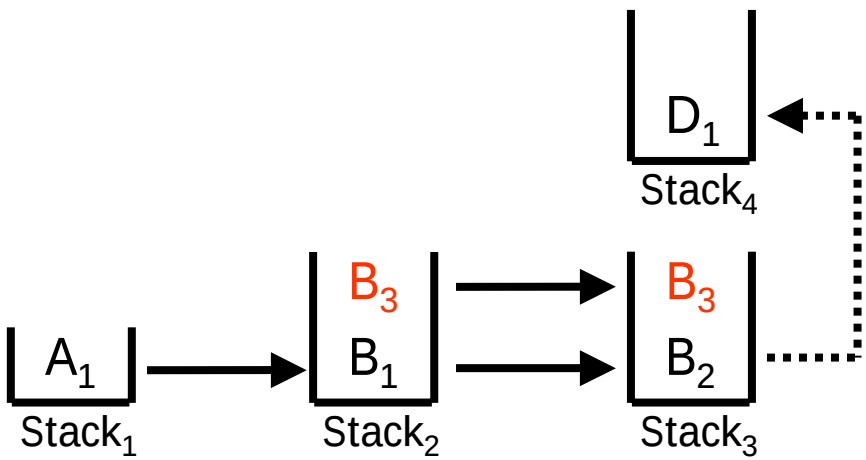


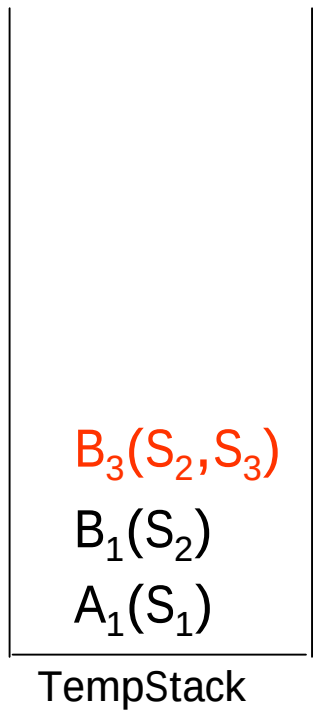
Query:



Read <B_3>

Counter$_1$=1
Counter$_2$=2
Counter$_3$=1
Counter$_4$=0



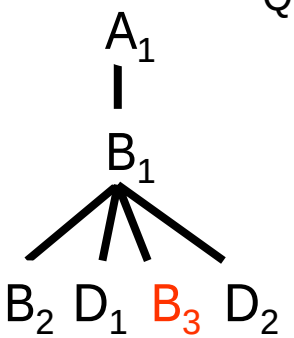


Example (TWIG algorithm)

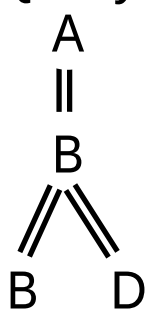
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:



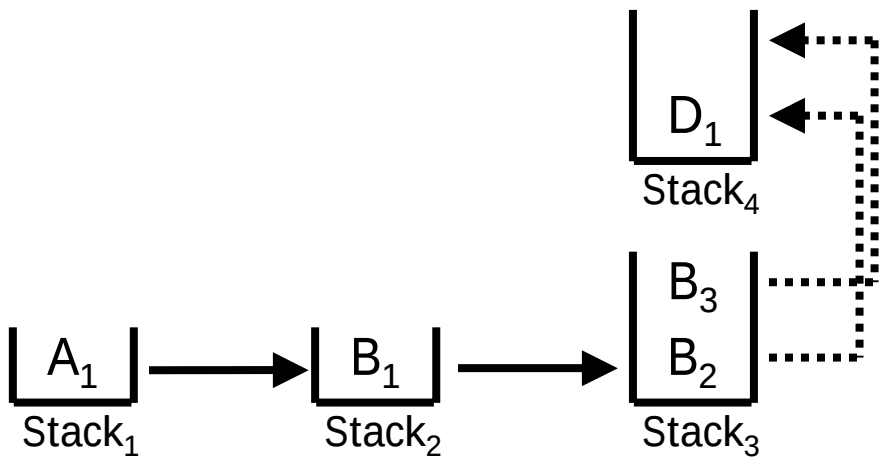
Query:

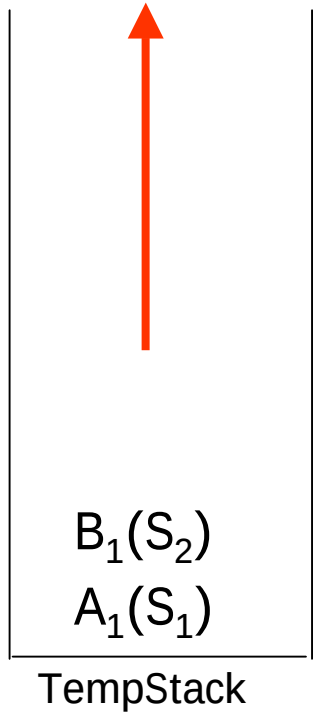


Read </B_3>

$Counter_1$=1
$Counter_2$=1
$Counter_3$=0
$Counter_4$=0

Pop $B_3(S_3)$



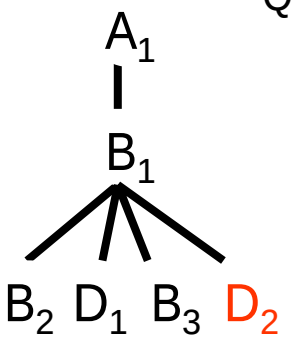


Example (TWIG algorithm)

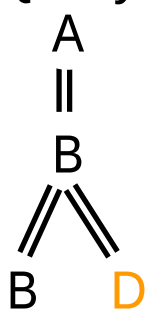
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

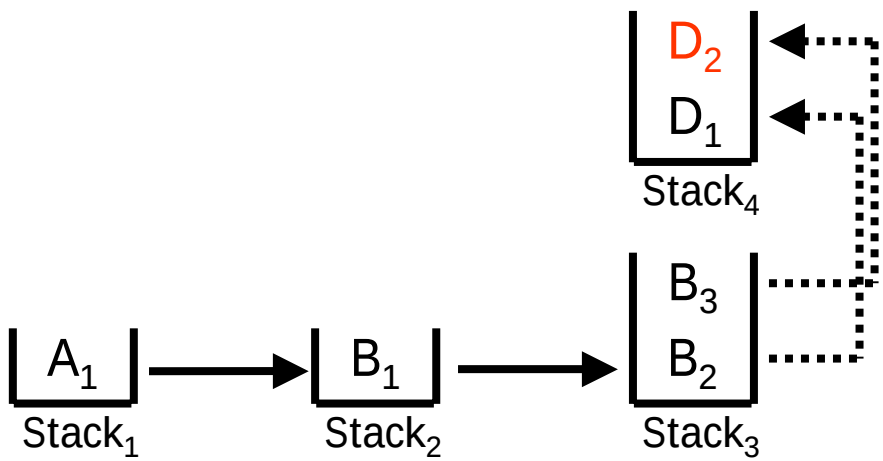


Query:



Read <D_2>

$Counter_1$=1
$Counter_2$=1
$Counter_3$=0
$Counter_4$=1

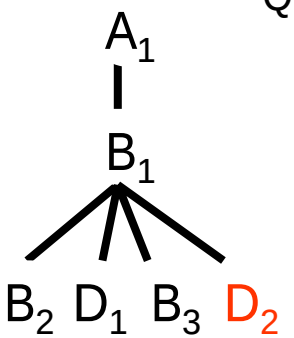


Example (TWIG algorithm)

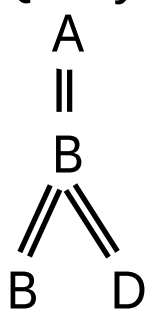
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

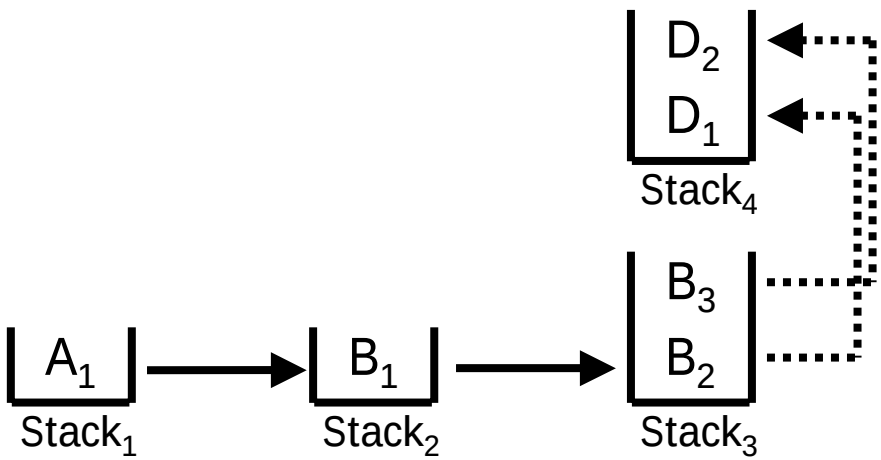


Query:

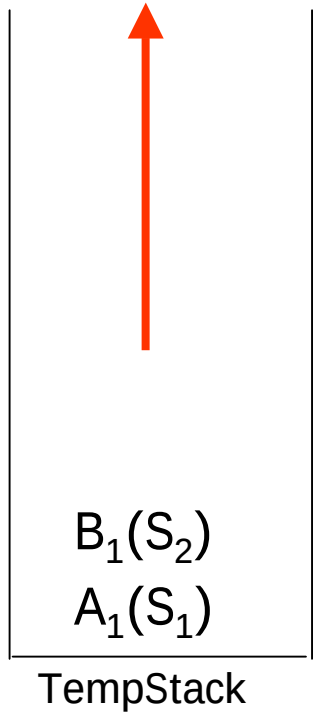


Read </D_2>

Counter$_1$=1
Counter$_2$=1
Counter$_3$=0
Counter$_4$=0



Pop $D_2(S_4)$

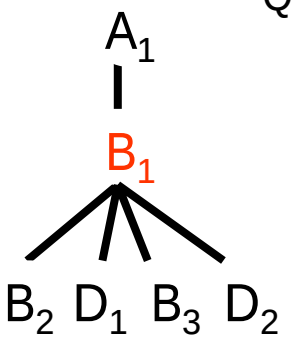


Example (TWIG algorithm)

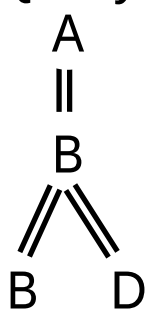
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

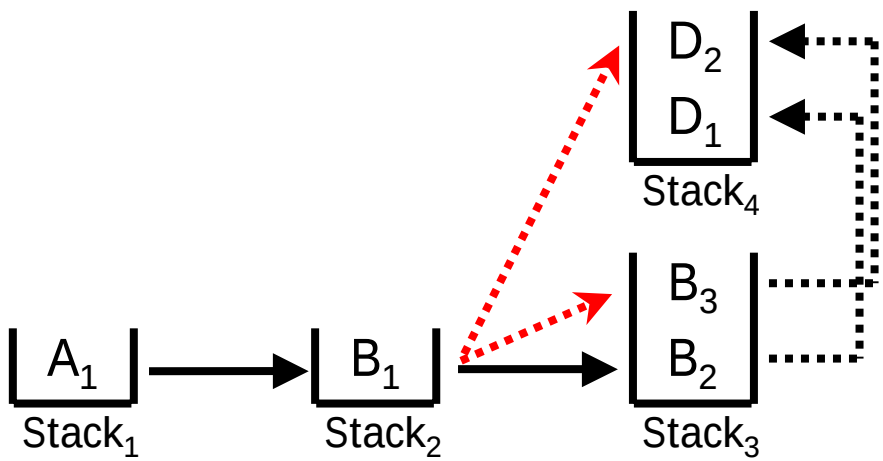


Query:

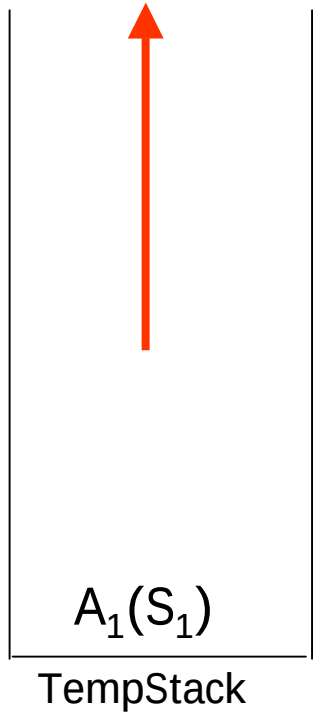


Read </B_1>

$Counter_1$=1
$Counter_2$=0
$Counter_3$=0
$Counter_4$=0



Pop $B_1(S_2)$

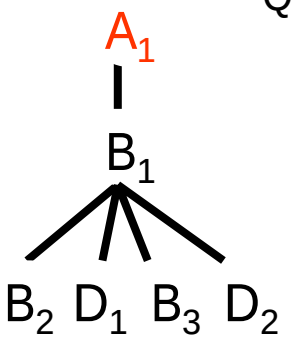


Example (TWIG algorithm)

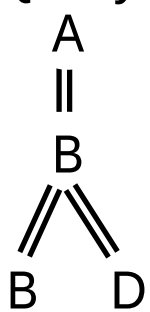
Data: <A_1><B_1><B_2></B_2><D_1></D_1><B_3></B_3><D_2></D_2></B_1></A_1>

Query: A // B[//B] // D

Data:

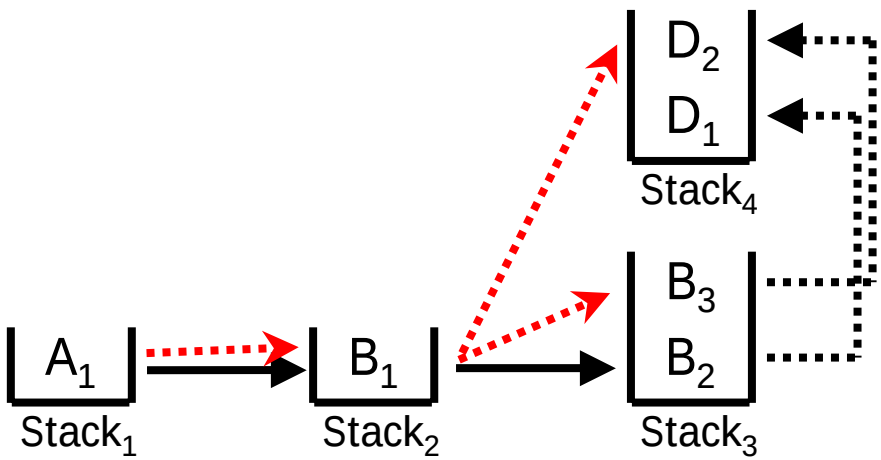


Query:

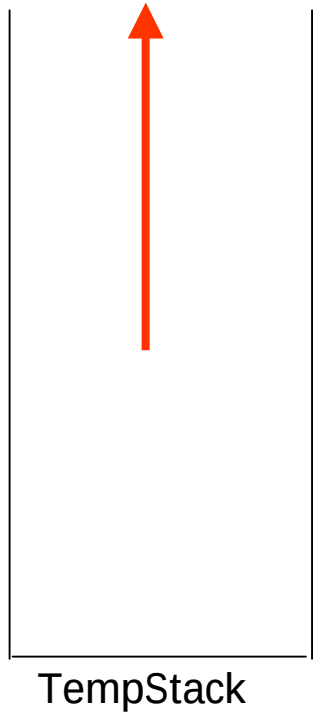


Read </A_1>

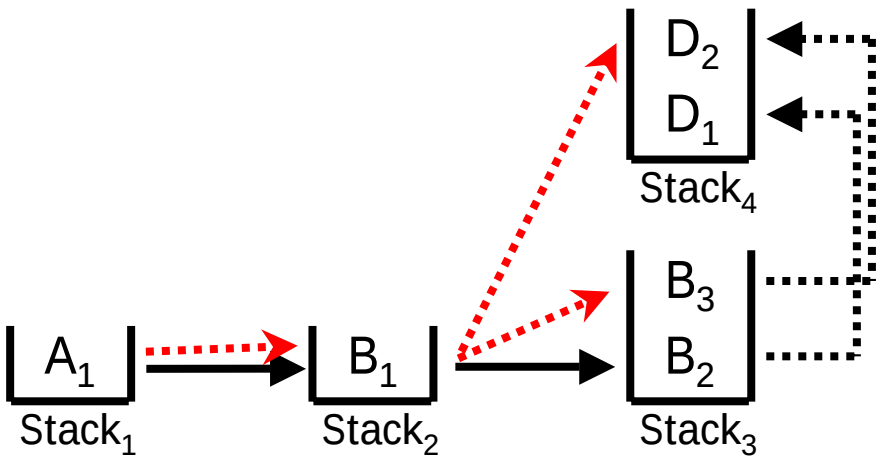
Counter$_1$=0
Counter$_2$=0
Counter$_3$=0
Counter$_4$=0



Pop A_1(S_1)



Example (TWIG algorithm)



Answer: $[A_1B_1B_2D_1]$, $[A_1B_1B_2D_2]$, $[A_1B_1B_3D_2]$

Experiments

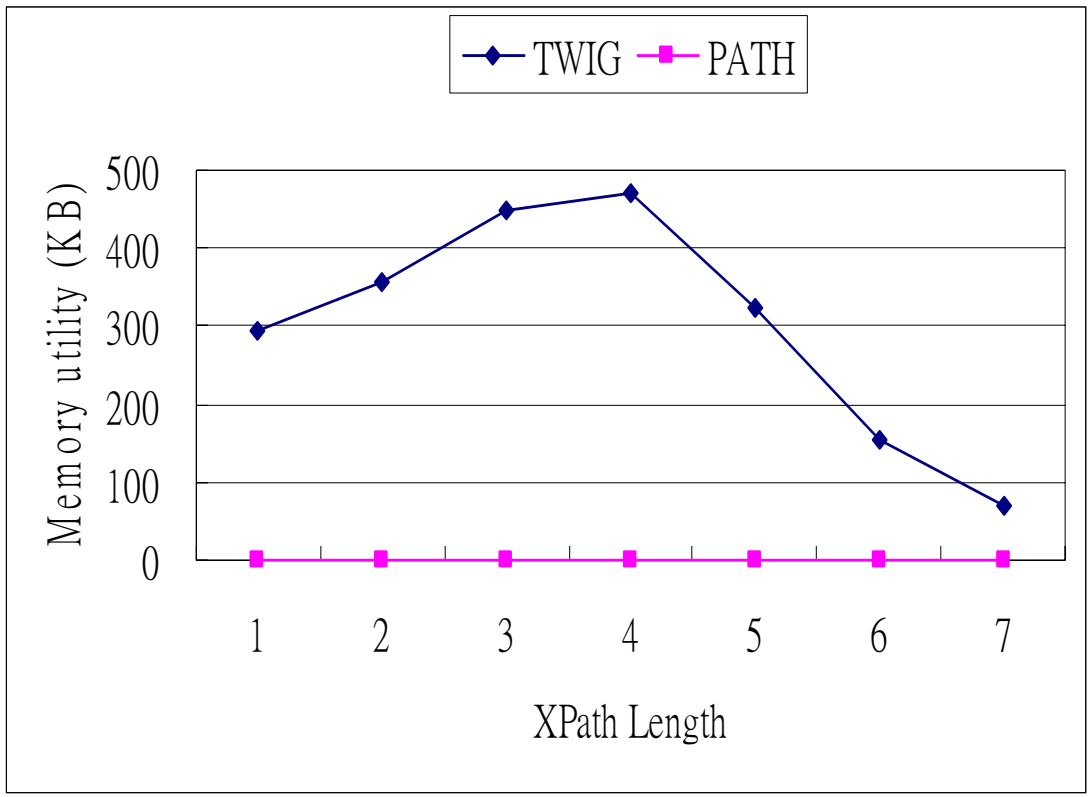
- Experiment platform
  - Java 2
  - Windows 2003 server
  - Intel Pentium 4 2.4 GHz
  - Memory 1GB
- Compared methods
  - PATH: proposed method using Start and End tags
  - TWIG: the same Path algorithm, but use pointer structures
  - XSQ: using Pushdown Transducer (PDT) [Peng et al., SIGMOD'03]
  - XMLTK: using Deterministic Finite Automaton (DFA) [Green et al., ICDT'03]

Experiments

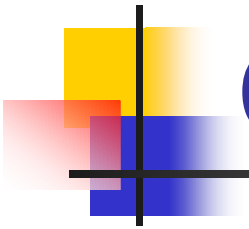
Execution time (seconds)

Dataset	XPath Queries	PATH	TWIG	XSQ	XMLTK
Auction (111MB)	site/people/person/name	11.69	11.71	27.56	21.84
	site/people/person[@id]/name	11.66	11.79	27.24	24.04
	open_auctions/open_auction/bidder/increase	10.85	11.06	30.80	25.85
	person[address]/profile[interest]/business	N/A	10.81	29.55	N/A
	person[@id][address]/profile[@income][interest]/business	N/A	10.72	N/A	N/A
NASA (23MB)	dataset//reference//holding	2.73	2.78	11.74	6.59
	dataset//reference[@type]//holding[@role]	2.68	2.74	11.69	N/A
	dataset[title]/descriptions[abstract]/details	N/A	3.41	11.13	N/A
	dataset[descriptions/abstract]/history/ingest/creator/lastName	N/A	4.12	N/A	N/A
	dataset[title]/descriptions[@xml:lang][abstract]/details	N/A	3.34	N/A	N/A
SigmodRecord (704KB)	issue//articles//author	0.17	0.18	0.49	0.43
	issuesTuple//articleTuple/title	0.15	0.17	0.44	0.39
	Issues//articles//author[@AuthorPosition]	0.17	0.18	0.51	N/A
	issuesTuple[volume]//articlesTuple[title]//author	N/A	0.25	1.12	N/A
	issues//articles[//title]//authors/author	N/A	0.34	N/A	N/A
Synthetic (116MB)	nation//country//city//company//manager//department//place //employee//email	16.28	16.68	105.98	22.21
	countries//city[@cityID]/place//countries/country	11.19	11.55	43.93	24.46
	company//countries[//city[@cityID]//department//name]//place	N/A	33.39	N/A	N/A
	countries[place]//country//place	N/A	12.41	61.48	N/A

Experiments



Memory utility for synthetic data (116MB)



Conclusion

- We propose new techniques for processing XPath queries on streaming XML data
- Process XPath queries without using indices
- Store less intermediate results to save storage space
- Process twig queries directly
- Handle duplicated data nodes
- Performance is better
- Memory usage is limited

Thank you