XML数据流上的有序查询处理

应用背景及涉及问题 (1)

- 很多基于XML的发布/订阅系统以及XML数据流处理系统都会有很多的有序查询需求[EDBT06]
  - 在股票信息监控上，我们常常需要查找某交易之后的某一笔交易
  - RSS新闻信息在线处理上查找头几条关于某某的新闻
- 用XPath来表示上面的查询需求，E.g.
  - **//contract[/content="XXX"]/following::contract**
  - **/news-col/news[/title="XXX"][position()<10]**

x7

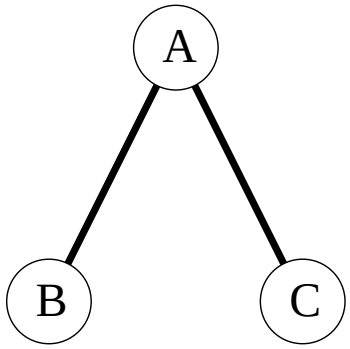
x7 这两个查询有一个共同的特点，就是我们需要在某一结点结束</XXX>之后的某个时间我们才能确定之前的这一结点是否是有用结点
xm, 2006-3-27

应用背景及涉及问题 (2)

- 这两个有序查询有一个共同的特点，就是我们需要在某一结点结束(**</content >,</news>**)**之后的某个时间**我们才能确定之前的这一结点是否是有用结点

- 如何快速高效地确定这些所有的之前缓存的待确定的结点**是否是有用结点**就成了影响这些流处理系统效率的一个重要的因素

XML数据流及上面的查询需求




```
<A>
   <B>
   </B>
   <C>
   </C>
</A>
```

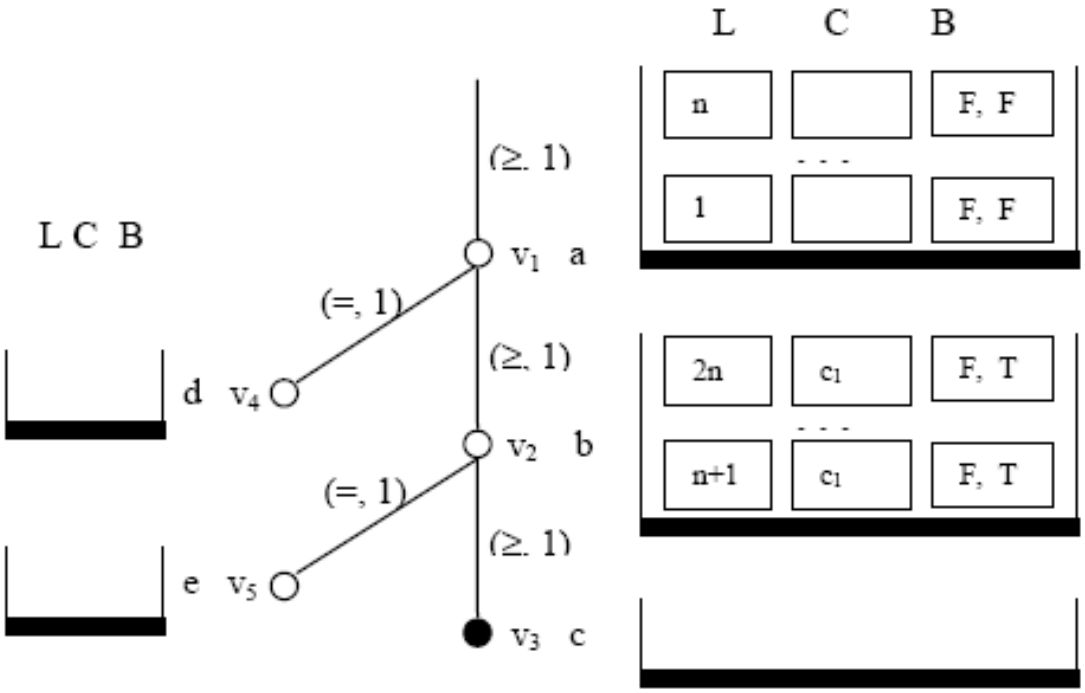
- **XPath有序查询**
  - **Following**
  - **Following-Sibling**
  - **Position Predicate**

- **问题：**
  - 在一次遍历深度优先XML文档的前提下，找到一个XPath查询的所有目标结点
- **已有的工作：**
  - 需要在遇到一个结点结束标签之时确定当前的结点是否构成结果
  - 处理XPath中的**有序查询**的时候，Following关系需要首先将查询从Following处分解，在得到两个部分结果后，再连接得到最后的结果，会缓存很多的中间无用结果
  - 没有对Position谓词的支持

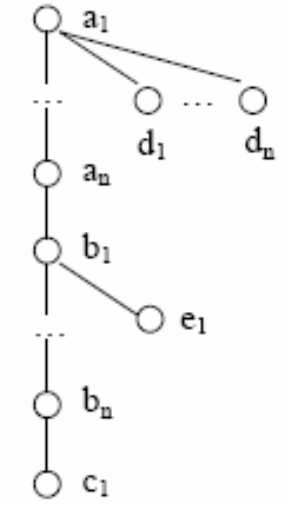
相关的工作

- **YFilter的方法**
  - 能够处理大量的单路径查询，但是如果是查询有分枝节点，需要将查询分解为单路径查询，然后做连接来得到最后的结果
- **TwigM的方法**
  - 对每一个中间的没有确定的中间结果，记录一个Boolean序列表明，各个子树是否已经找到一个匹配
  - 处理文档中嵌套的结点时，使用一个栈来缓存所有的嵌套结点
  - 但是在处理Following Step的时候，需要将查询从Following Step处分解成两个查询，然后对两个查询的结果做连接操作来得到结果
  - 没有对Position谓词进行支持

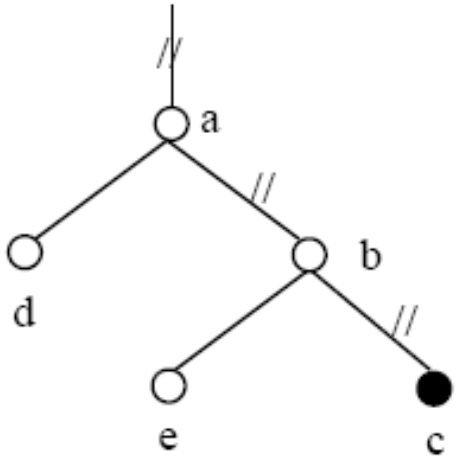
TwigM



L: level B: branch match C: candidates



(a) Data D_1



(b) Query Q_1

我们的想法

- 基于Chen的基于栈的对流查询的处理方法，针对XPath有序查询（Following，Following-Sibling和Position谓词）
  - 对于查询树上的结点，我们根据后继关系需要增加一个O-Spec（Order-Specification）
  - 对于含有Position谓词的XPath查询，我们会在查询树的适当结点上增加一个P-Spec（Position-Specification）
  - 改进流上的XPath处理算法，利用P-Spec和O-Spec，在处理XPath有序查询是，我们能够保证在XML数据流上扫描一次就找到所有的解，使用较少的缓存，避免了最后的连接操作，显著地提高流查询处理系统的性能

x1

注意这里需要加一些东西来说明：由于流数据本身的特点，我们在看到当前的流位置时可能不能确定之前的结点对于查询来说是有用还是没有用（举个例子），所以需要缓存所有的不确定结点，这里看是否需要说明流数据里面应该尽量减少缓存对应的结点，这样我们就需要能够尽早地知道一个缓存的结点是有用的节点（可以输出到结果中，并释放缓存）或者说我们知道一个缓存的结点是无用的结点，可以直接释放缓存

xm, 2006-3-22

Preliminary

- XML
- SAX
- XPath
- Encoding(SEP) [CIKM05]
- TwigM Algorithm [ICDE06]

有序查询的匹配

- 与Lu[DEXA 2005]文章类似，把含有有序谓词要求的XPath查询叫做**有序XPath查询**

- 这里我们将有序的XML查询分为两个部分
  - 一个部分是Following，Following-Sibling关系，要求一个查询树上的结点是某个结点的后继或者是后继兄弟 A/following::B
  - 一个部分是Position谓词，考虑某个特定位置出现的查询结点 A/B[5]，A/B[position()=last()]

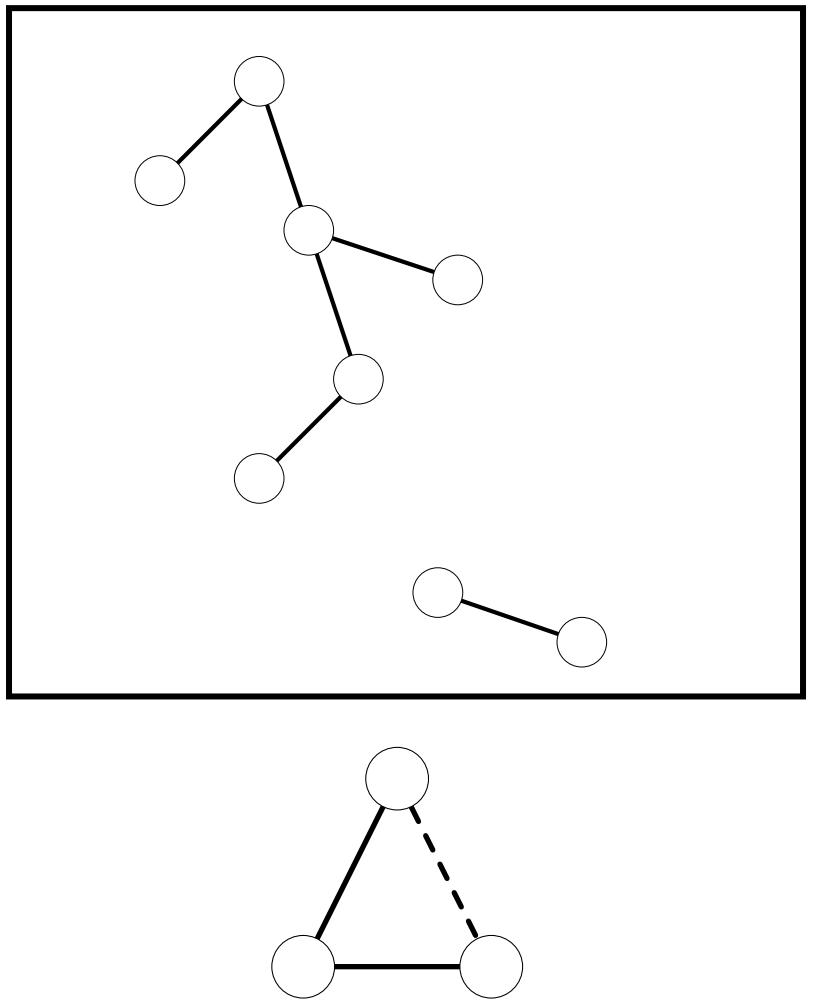
x2

- 有效地处理XPath有序查询可以为基于XML的查询/订阅系统提供更好的支持

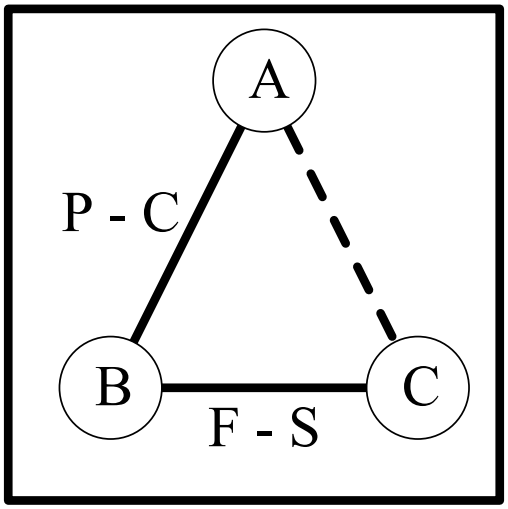
x2 （由于在流上不能向前看的特点，我们需要将Preceding和Preceding-Sibling转化[paper]）
xm, 2006-3-22

之前工作对有序查询的处理

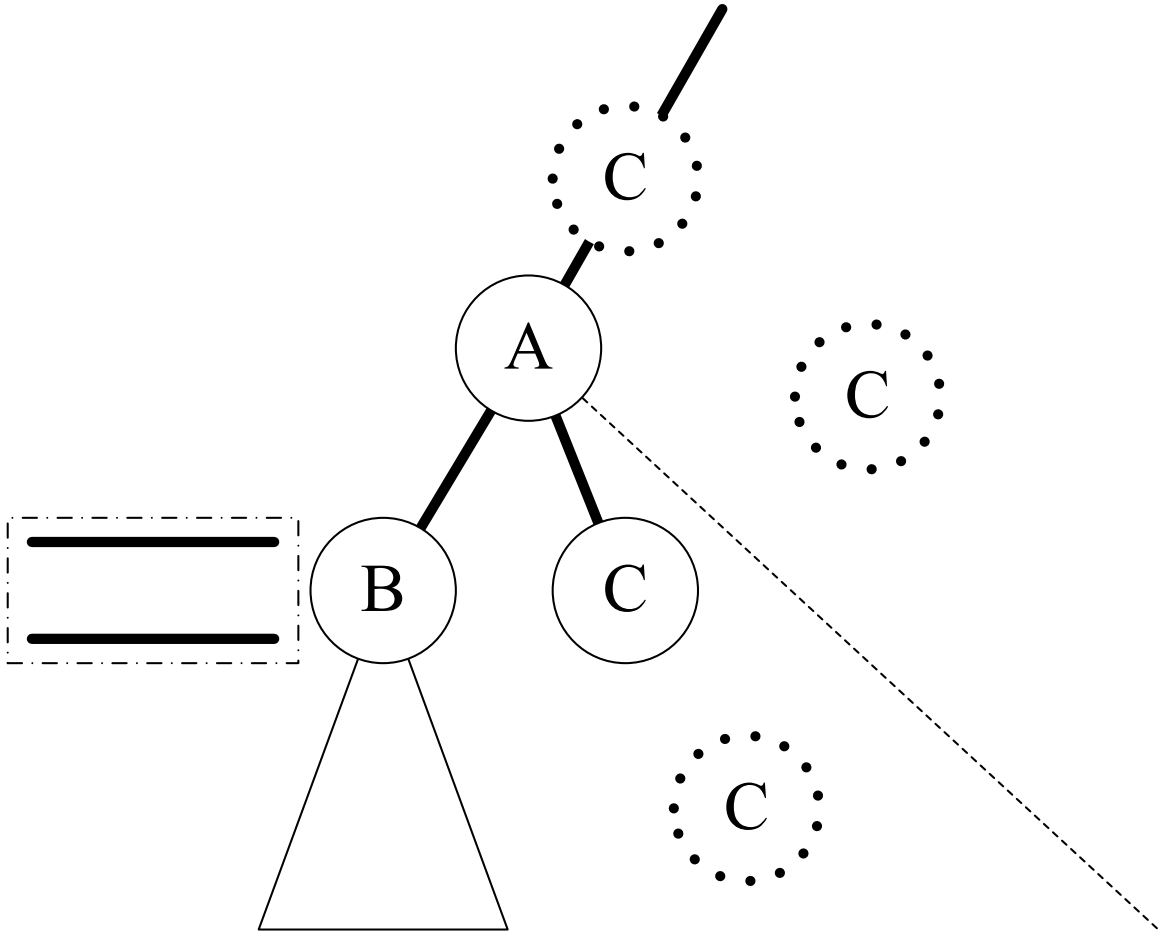
- 之前工作中的处理策略可能会造成缓存很多无用的查询结果
- 对于查询*A/B/Following-Sibling::C*，如果将查询分解成为两个分枝，将查询匹配到文档上去，会缓存*n*个无用的*C*节点，而其实在扫描的过程中就可以判断一些*C*是否是有用
- 查询*//A/Following::B*会被分解成为两个查询，*//*A，*//*B，在获得结果之后，再做连接得到最后的结果，但是这样会缓存相当数目的中间结点



Following-Sibling (1)



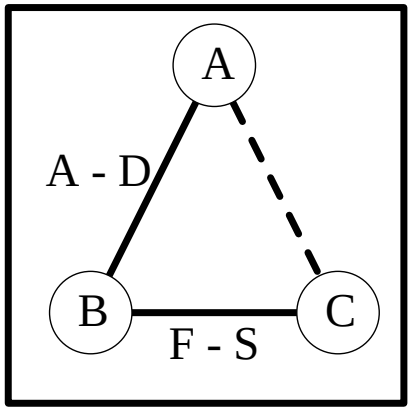
x8

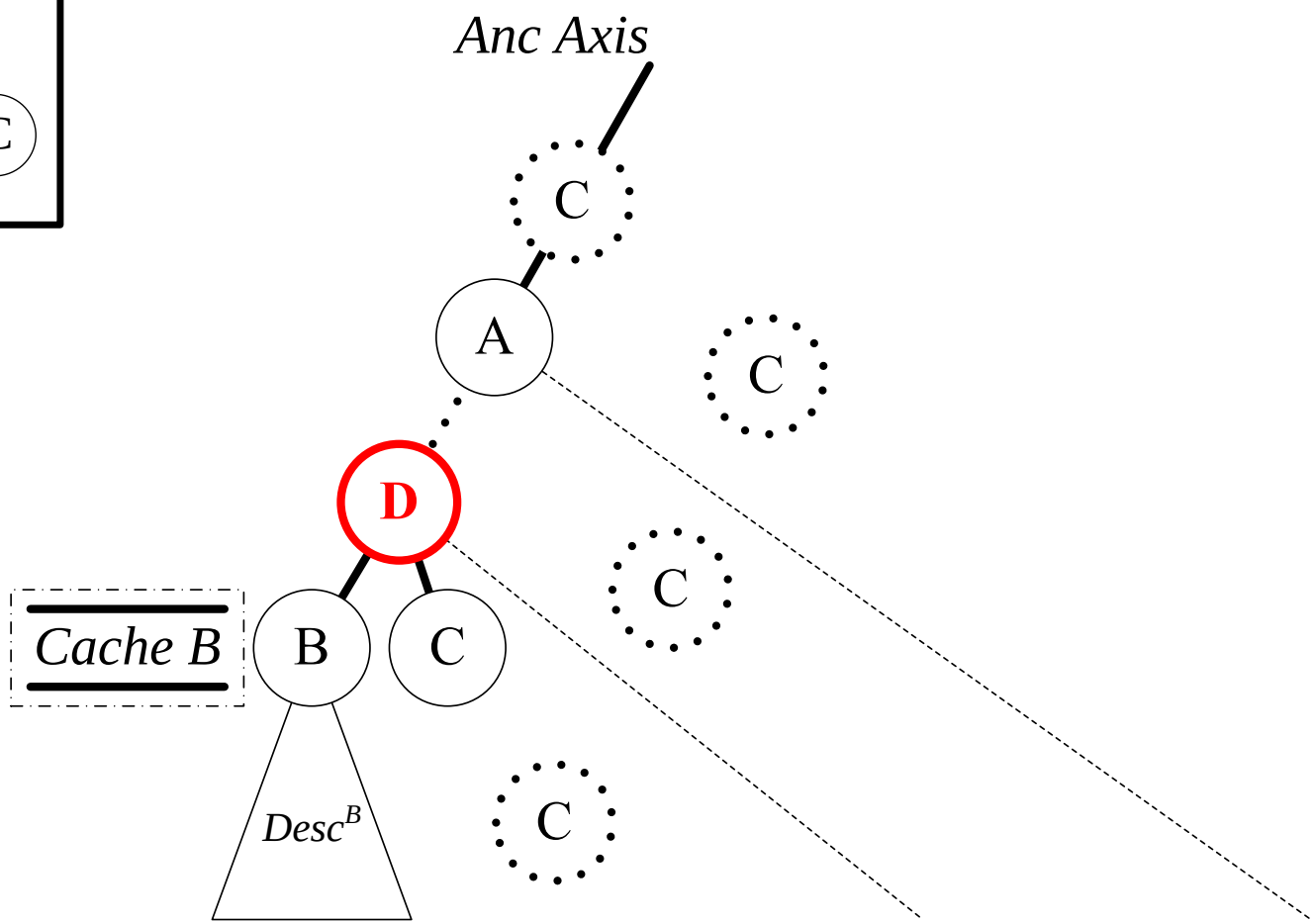


x8 在A结点结束</A>时，我们要清空B的Cache

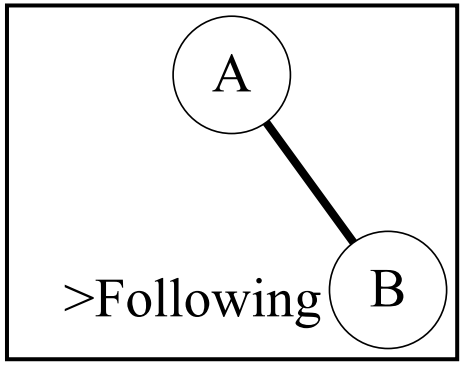
xm, 2006-3-28

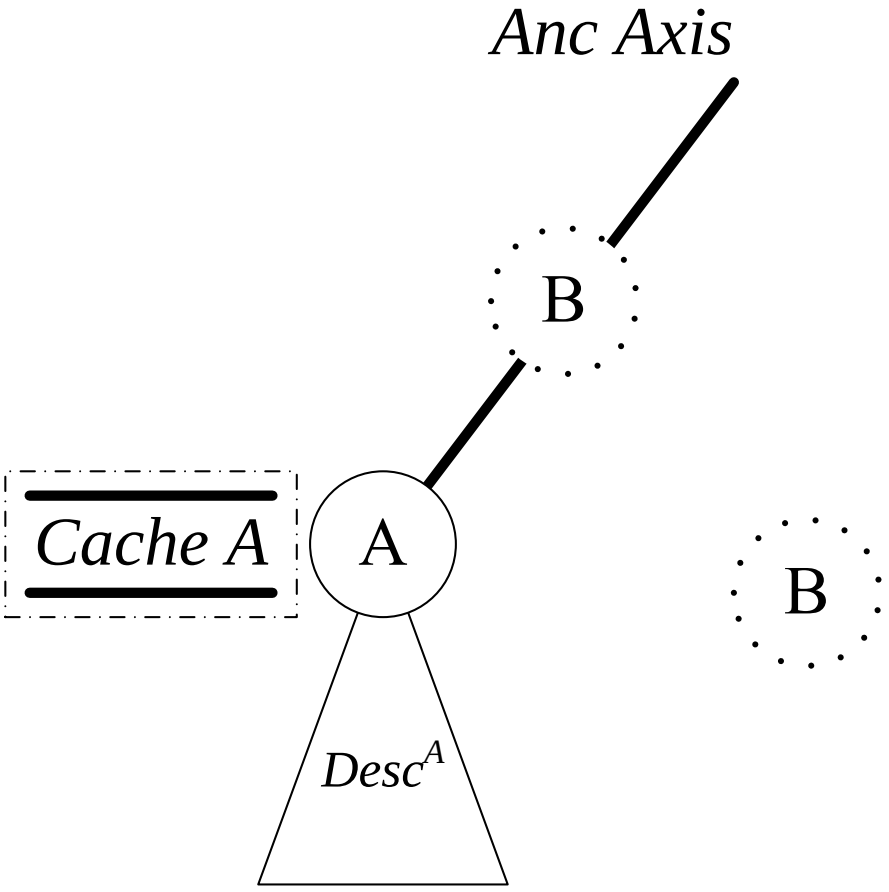
Following-Sibling (2)



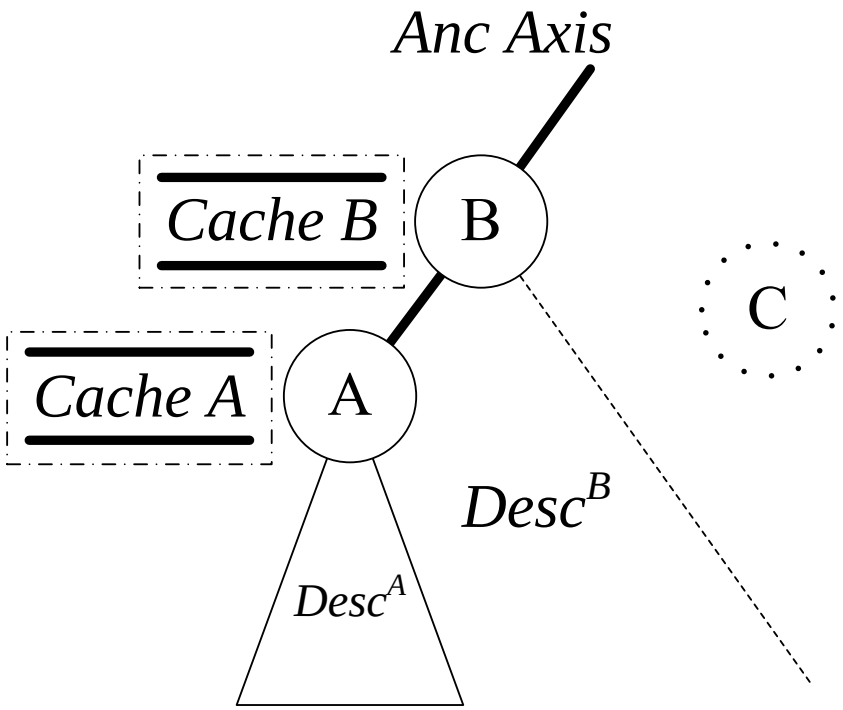


Following (1)





Following (2)



- 对于Following来说，有可能所有的祖先查询结点都需要做级联缓存
- E.g.
  - **B/A/Following::C**

x6

有序XPath查询

- 针对两个部分的XPath查询上的有序要求，我们提出两种查询节点上的要求
  - O-Spec (Order Specification)
  - P-Spec (Position Specification)

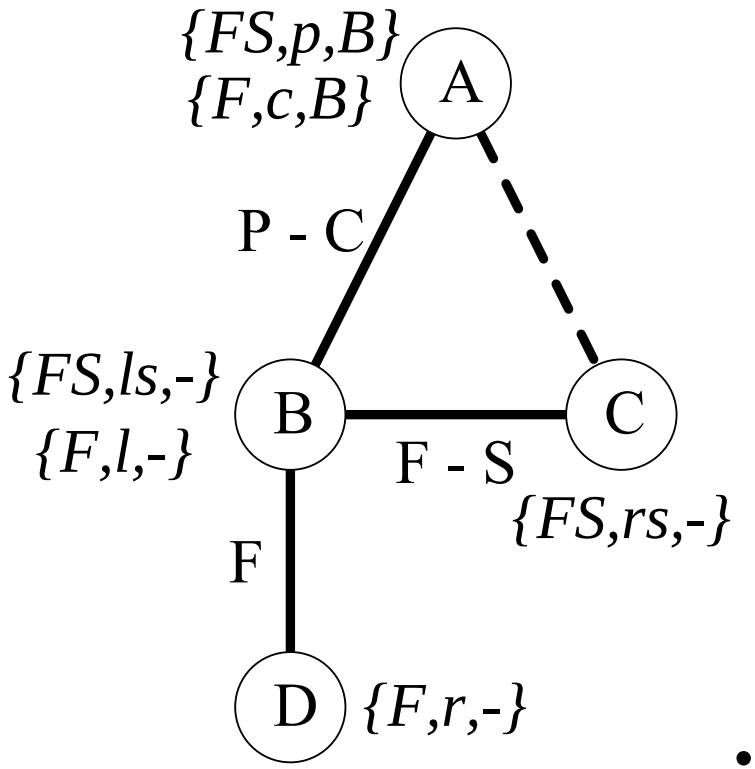
x6 在处理查询时，当查询是有序的XPath查询时，只有所有的含有O-Spec和P-Spec的结点满足两种有序要求时，我们才说找到一组对应的解

xm, 2006-3-22

x5

O-Spec (Order Specification)

x4



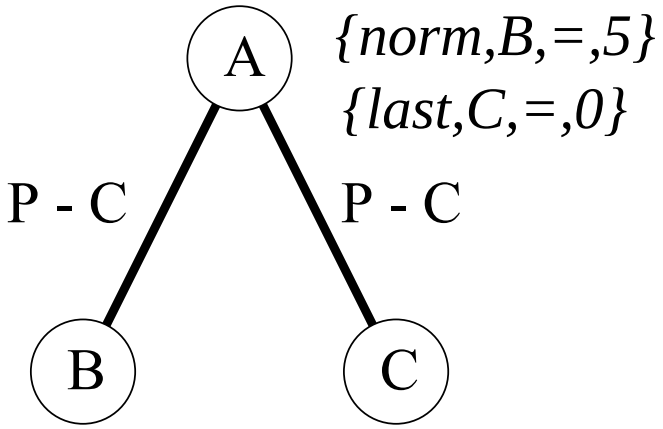
- 对于一个含有Following或者是Following-Sibling的查询，在相关的节点上标记一个三元组 **O-Spec *{type,relation,target}***
  - 如果*A/B/following-sibling::C*，则在*B*结束之后，只有在遇到一个*C*兄弟的时候，才能确定*B*是否是解，所以需要先缓存部分解*B*
    - 在A上标记 ***O-Spec{FS, p, B}***
    - 在B上标记 ***O-Spec{FS, ls, -}***
    - 在C上标记 ***O-Spec{FS, rs, -}***
  - 如果*B/following::D*，则在*B*结束之后，只有在遇到一个之后出现的*D*时，才能确定*B*是否是解，所以需要先缓存部分解*D*
    - 在*B*上标记 ***O-Spec{F, l, -}***
    - 在*D*上标记 ***O-Spec{F, r, -}***
- E.g.
  - A/B[following-sibling::C]/following::D

x4 对于Following Sibling而言，有两种可能：一种是左兄弟节点在返回路径上，C不在路径上，这样B结束的时候需要缓存，然后等每个C结束的时候，去Check每个B缓存的东东，将其置到父亲A中，这是B只要一个标志表示要缓存就好，然后C要标记每个C结束之时去checkB的缓存；
另一种情况是B不在返回路径上，然后C在结束时需要去检查是否之前有B，如果有的话就表示OK，
xm, 2006-3-22

x5 发现一个严重问题，级联兄弟依赖关系，这种东东想想怎么办法来处理
xm, 2006-3-22

x3

P-Spec (Position Specification)



- 对于一个有Position谓词的查询，在查询树上的相关结点上标记一个三元组
  P-Spec {*type, target, op, num*}
  - 如果查询中涉及的是*A/B[position() op num]*, 则在遇到满足条件的*B*时才能算构成解的*B*
    - 查询*A*结点上标记***P-Spec{comp,B,op,num}***
  - 但如果涉及的是*A/C[position() op last()]*或*A/C[position() op last()-num]*时，只有在当前活动的*A*节点结束之时，我们才能决定最后一个*B*，在这种情况下
    - 查询节点A上标记***P-Spec{last,C,op,num}***

- E.g. A[B[5]]/C[position()=last()]

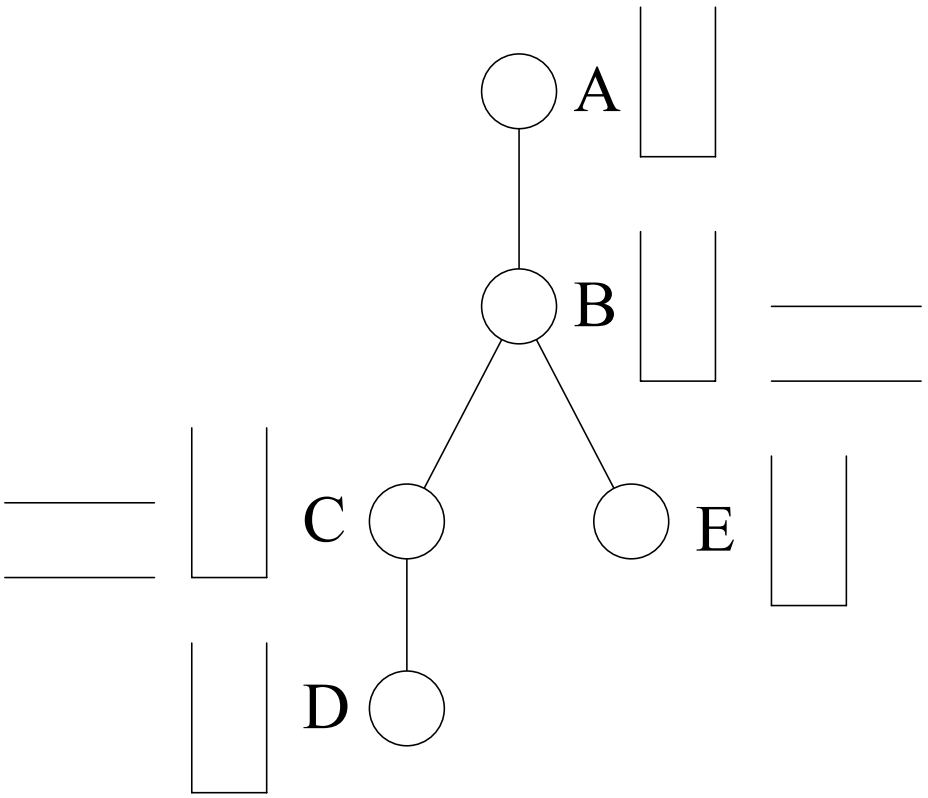
x3 可以给出一个很Formal的匹配方法，表明任何一个含有Sibp的节点只有在所有的涉及子节点都满足Sibc要求后，一般的结构谓词也满足才叫找到一个解

xm, 2006-3-22

Position Predicate

- 对于A/B[Position() op num]这类的Position谓词
  - 我们会在每个父查询结点(这里是A)上维护一个Counter
  - 每当一个满足要求的子查询结点(这里是B)结束，并将结果返回到父查询结点时，我们就利用这个Counter来判断这个返回的子查询结点是否满足查询的Positon谓词的要求
  - 在父查询结点结束之时，如果有子查询结点满足要求，则表示有解
- 含有last()的Position谓词 A/B[Position() op last()-num]
  - 我们需要在父查询结点(这里是A)维护一个缓冲队列
  - 每当一个满足要求的子查询结点(这里是B)结束，并将结果返回给父查询结点时，这里分几种情况：
    - 如果缓冲队列还不满，直接将子查询结点插入到缓冲队列中
    - 如果缓冲队列已满了，从队列首pop出一个子查询结点，并把当前结点插入到队列尾
  - 父查询结点结束时，如果队列满，表示有条件满足的子查询结点

Order-TwigM算法（1）



- 我们给每个XPath查询Q构造一个查询树，每个XPath的Step对应查询树上的一个结点，结点之间的关系根据XPath的Step确定
- 每个结点对应一个栈，用于处理嵌套结点关系
- 每个结点根据查询的Order要求和Position要求增加O-Spec和P-Spec
- 对于含有O-Spec而且二元组为(F,l,-)(FS,ls,-)的结点增加一个缓冲队列(递归的情况给每个祖先结点也增加一个缓冲队列)
- 对于含有P-Spec而且type为last的结点也增加一个缓从队列

S_A

$\{1,-1\}$

P-Spec

$\{last,E,-,0$

A//B[C/following-Sibling::D]/E[position()=last()]

Q_{Spec}

$\{FS,p,C\}$

S_B

Q_E