# TOWARDS AN EFFECTIVE XML KEYWORD SEARCH

### 介绍

问题1：搜索目标节点

问题2：搜索条件节点

问题3：给查询结果排序

歧义1：关键词可以以XML标签名和节点值的形式出现。

歧义2：关键词可以以不同类型的XML节点的值的形式出现，表达不同的含义。

歧义3：关键词可以以不同上下文中的XML标签名的形式出现，表达不同的含义。

**主要贡献：**

1. 这是第一个解决**关键词歧义问题**的工作。我们还确定了一个有效的XML关键词搜索引擎应该满足的**三个关键问题**。
2. 我们定义了我们自己的**XML检索词频率（TF）**和**XML文档频率（DF）**，这是后来提出的所有公式的基础。
3. 我们提出了**三个重要的准则**，以识别用户期望的**搜索目标节点类型**，并在指导方针的基础上设计一个公式来计算一个特定的**搜索目标节点类型**的置信度。
4. 我们设计公式模拟人的直觉来计算每个候选节点类型是期望的**搜索条件节点类型**的置信度，我们考虑了查询中的关键词共现问题。

“共现”指文献的特征项描述的信息共同出现的现象，这里的特征项包括文献的外部和内部特征，如题名、作者、关键词、机构等。 而“共现分析”是对共现现象的定量研究， 以揭示信息的内容关联和特征项所隐含的知识。

1. 我们提出了一种新的面向相关性的排序方案，称为**XML TF\*IDF相似度**，它可以捕获XML的层次结构，并以启发式的方式解决**歧义1-3**。此外，我们设计查询结果的受欢迎程度来将结果与可比的相关性分数进行区分。
2. 我们实现的这个被提出的**关键词搜索引擎原型**技术被称为XReal。大量的实验表明了它们的有效性、效率和可扩展性。

第3节中介绍了数据模型。第4节推断用户搜索意图。第5节讨论了排序方案。第7节介绍了搜索算法。（）内的内容为讲述部分。

### 3.准备工作

3.1 TF\*IDF Cosine Similarity 余弦相似度

TF\*IDF的基本思想：

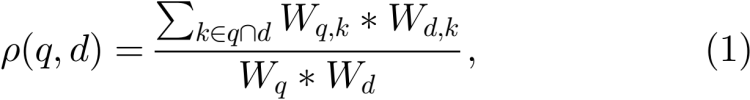
**规则1：**在许多文档中出现的关键词应该被视为比在少数文档中出现的关键词更不重要。

**规则2：**有更多查询关键词出现的文档应该被认为对那个关键词更重要，对比有更少关键词出现的文档。

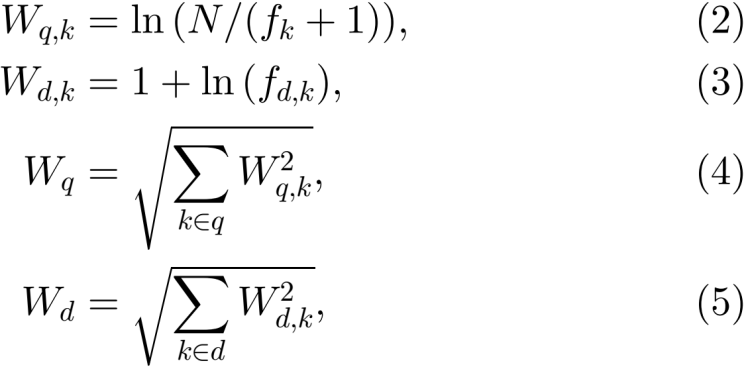
规则1和2是说，关键词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。

**规则3：**在长文档和短文档之间需要一个规范化因子来平衡，因为规则2对短文档不公平，因为短文档一般包含更少的关键词。

（为了结合上述三种规则，TF\*IDF相似性被设计为：）



（q表示一个查询，d表示一个平面文档，K是出现在q和d中的关键词，较大的ρ(q,d)值表示q和d彼此更相关。我们通常用下面的公式计算权重。）



（其中N是文件的总数，**文档频率Fk是包含关键词k的文档数量；检索词频率Fd,k是k在文档d中出现的次数**。）

（Wq,k是关于Fk单调递减的，它反映规则1。而Wd,k是关于Fd,k（检索词频率）单调递增的，它反映规则2。（2）和（3）的对数是为了规范原文档频率Fk和原检索词频率Fd,k。最后，Wq和Wd关于q和d的尺寸增加，发挥规范化因子的作用，以反映规则3。）

（然而，原始的TF \* IDF对XML是不够的，因为它无法完成识别搜索意图，或者解决XML层次结构导致的关键词歧义问题，如Example 3所示。）

**Example 3.** 假设发布一个关键词查询“艺术”，以搜索在图1 XML数据中对“艺术”感兴趣的客户。理想情况下，系统应该将那些兴趣节点里有“艺术”的客户排在那些没有的人之前。此外，它应该给只对艺术感兴趣的客户A更高的排名，相比另一个有很多兴趣而“艺术”只是其中之一的客户B（如图1中的C4）。

然而，如果直接对XML数据使用原始的TF \* IDF，则会产生两个问题。1）如果不考虑客户节点结构，如果客户A正好比客户B有更多的关键词在它的子树里面，那么A就相当于信息检索IR里面的长文档，那么客户A可能就比B排名更低。2）更糟的是，假设一个客户C对“艺术”不感兴趣而他的属性节点“address”地址在“艺术街”。如果在基础的XML数据里面C比A和B的关键词数量更少，那么C可能比A和B的排名更高。

3.2 Data Model数据模型

我们将XML文档建模为有根的、有标记的树。相比于一般的有向图模型，包含边和IDREF边在我们的模型中是区分开来的。另外，我们用一个节点的前缀路径，而不是它的标签名进行结果检索和排序。

在我们的数据模型，有向边分为包含边或IDRef边。包含边u → v 表示u是节点v的父节点。从节点u指向节点v的IDRef边被表示为u → v（虚线） ，IDRef类型的节点u的属性具有一个值等于节点v的ID类型的属性。例如，图1中的有向实线表示包含边；从purchase（这是customer的一个属性）到book的虚线是一个从customer C2 到book B1的IDRef边。

**定义3.1（节点类型）.** XML文档中节点n的类型是从根到n的前缀路径。如果两个节点共享相同的前缀路径，则它们的节点类型相同。

（在定义3.1中，两个节点需要共享同一前缀路径而不是它们的标签名的原因是，在一个文档中可能有两个或多个节点具有相同的标签名，但具有不同的语义（即在不同的上下文中）。例如，在图1中，出版社的名字和客户的名字是不同的节点类型，分别是storeDB/ books/book/publisher/name和storeDB/customers/ customer/name。此外，当XML数据库包含多个XML文档时，节点类型也应该包含文件名。）

（为了便于稍后的讨论，我们使用标签名称代替节点的前缀路径来表示本文中所有示例中的节点类型。此外，为了将内容部分与叶节点分离，我们将XML节点区分为**数据节点**或**结构节点**。）

**定义3.2（数据节点）.** XML数据的叶节点中包含的没有标签名的文本值被定义为**数据节点**。

**定义3.3（结构节点）.** 用标签名标记的XML节点被称为**结构节点**。一个包含其他结构节点作为其子节点的结构节点被称为**内部节点**；否则，它被称为**叶节点**。

（在本文中，我们不考虑内部节点n同时包含数据节点和结构节点的情况，因为我们可以在节点索引中添加一个带有标签名称“value”的虚拟结构节点在n和数据节点之间，而不改变XML数据，这样就可以很容易地避免上述情况。）

（根据以上两种定义，XML数据的值部分和结构部分被分离。例如，在图1客户C1的子树中，地址是一个内部节点，街道是叶结点，“Art Street”是一个数据节点。）

**定义3.4（单值类型）.** 如果t类型的节点在其父节点的子节点中最多出现一次，则结构节点t是单值类型。

**定义3.5（多值类型）.** 如果t类型的节点在其父节点的子节点中出现多次，则结构节点t是多值类型。

**定义3.6（分组类型）.** 如果t类型的节点的子节点都是同一个多值类型的节点，则内部节点t被定义为分组类型。

（解析数据时，可以很容易地识别单值类型和多值类型的XML节点。单值（或多值或分组）类型的节点称为单值（或多值或分组）节点。例如，在图1中，address是单值节点，而interest是多值节点，interests是interest的分组节点。）

（在本文中，为了便于后面的介绍，我们假设每一个多值节点都有一个分组节点作为它的父节点，因为我们可以在不改变数据的情况下轻松地在索引中引入一个虚拟的分组。注意，分组节点也是一个单值节点。因此，内部节点的子节点要么是同一个多值类型，要么是不同的单值类型。）

**定义3.7（引用连接）.** 如果存在一条从u到v的有向路径P：u →... → v ，P中的每条边都是IDRef边，则节点v有一个来自节点u的引用连接（RC），表示为RC(u,v)。某一从u指向v的引用连接路径P的距离被定义为参与P的IDRef边的数量。

（例如，在图1中，从 customer C2到book B2的距离是2，因为有两条IDRef边参与。）

3.3 XML TF & DF

（受数据统计在IR排序中的重要作用的启发，我们试图利用它来解决XML关键词搜索的歧义，因为它通常提供了一种直观和令人信服的方式来建模和捕捉人类的直觉。）

**Example 4.** 当我们在如图1所示的数据库的领域里谈论“art”，我们首先把它考虑为 customer节点的interest属性或者book节点的category (或title)属性里的一个值。然而，我们很少首先考虑它为其他节点类型的值（例如，值为“Art Street”的street）。

（这种直觉的原因是，通常在许多interest类型和category类型的节点的文本值中包含“art”，而在street节点中“art”很少出现。这种直觉（基于领域知识）总是可以通过底层数据的统计来捕获。同样，当我们直观地谈论“interest”，我们首先考虑它为一个节点类型而不是一个book节点的title属性的值。此外，把“interest”匹配为XML标签interest的原因，也可以用统计观点来解释，即，所有的interest节点在其子树中包含关键词“interest”。）

统计法在XML关键词搜索中的重要性形式化如下：

**Intuition 1.** 越多T型的XML节点（及其子树）在其文本值或标签名中包含一个查询关键词k，就越直观地发现，T型节点和关于关键词k的查询的关系越密切。

在这里，我们维护和使用两个重要的基本统计术语：fa,k和。

**Definition 3.8 ((XML TF) fa,k).** XML数据中节点a中关键词k的出现次数。

**Definition 3.9 ((XML DF)** **).** XML数据中在其子树中包含关键词k的T型节点的数量。

（在这里，我们分别用与原始TF\*IDF中的词频fd,k和文档频率fk相似的方法来定义fa,k和；不同的是我们使用来区分不同节点类型的统计数据，因为**用来测量XML相似度的粒度是子树而不是文档**。因此，fa,k和可以直接用于测量数据节点（父节点为T类型）和基于原始TF \*IDF直觉的查询之间的相似度。此外，也有助于解决歧义，如**Intuition 1**所示。接下来我们将讨论如何将这两组统计数据用于有歧义的XML关键词搜索的面向相关性的排序。）

### 推断关键词搜索意图

（在这一部分中，我们将讨论如何根据XML数据中的统计信息和查询中的关键词共现模式来解释关键词查询的搜索意图。）

4.1 推断搜索目标节点类型

为了得到准确的答案，一个搜索引擎首先需要关注的问题是期望的搜索目标节点类型，因为关键词查询中的搜索目标不能像结构化查询语言中那样明确指定。给定一个关键词查询q，当且仅当以下三条准则成立时，节点类型T被视为期望的搜索目标节点类型：

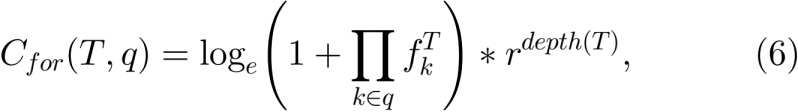
**Guideline 1:** T与q中的每个查询关键词直接相关，即，对于每个关键词k，都应该有一些T型节点在其子树中含有k。

**Guideline 2:** T类型的XML节点应该是信息丰富的，足以包含足够的相关信息。

**Guideline 3:** T类型的XML节点不应包含太多不相关的信息。

（准则2希望一个更高等级的内部节点类型T作为返回节点，而准则3希望那个T型节点的等级不太靠近根节点。例如，让我们参考图1：根据准则2，interest,street等类型的叶节点通常不是理想的返回节点的候选者，因为它们没有提供很多信息。根据准则3，customers和books类型的节点也不是好的候选者，因为它们作为一个单独的关键词搜索结果过于巨大。）

结合上述准则，我们定义了节点类型T是关于给定关键词查询q的期望搜索目标节点类型的**置信度Cfor(T,q)**，如下：



其中K代表查询q中的关键词；是在子树中包含关键词k作为值或者标签名的T型节点的数量（如第3.3节所解释的，以反映Intuition 1）；r是一个(0,1]范围的缩减因子，通常被选为0.8，depth(T)代表文档中T型节点的深度。

（公式（6）中，第一个乘数实际上将Intuition 1做成模型解决了Guideline 1。同时，它有效地处理了Guideline 3，因为候选的臃肿节点（即靠近根节点的节点）将被分配一个很小的值，导致置信度很小。第二个乘数简单地减小了那些在XML数据库中嵌套的很深的节点类型的置信度，以处理Guideline 2。）

（此外，我们在第一个乘数中使用的乘积而不是和来为每个节点类型T结合所有查询关键词的统计数据。原因是，每个查询的搜索意图通常有一个独特的期望搜索目标节点类型，所以使用乘积来确保节点类型被直观地联系到所有的查询关键词，以获得较高的作为期望类型的置信度。因此，如果节点类型T不能包含查询的所有关键词，则它的置信度被设为0。）

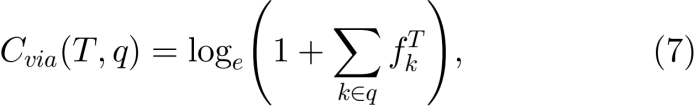
**Example 5.** 给定一个查询“customer interest art” ，节点类型customer作为期望的搜索目标节点类型通常具有较高的置信度，因为这三个统计数据，和的值（即，在其嵌套的文本值或标签名中分别包含“customer”，“interest”和“art”的以customer节点为根节点的子树的数量）通常大于1。相反，节点类型customers没有较高的置信度，因为fcustomers”customer”=fcustomers”interest”=fcustomers”art”=1。同样的，节点类型interest没有较高的置信度,因为的值通常很小。例如，图1中的XML数据，=0。

最后，当已知每个节点类型示所需类型的置信度时，当最高置信度显著高于第二时，选择具有最高置信度的节点作为所需的搜索目标节点。然而，当几个节点类型具有可比较的置信度值时，可以为用户提供一个选择来决定所需的节点，或者系统将搜索每个可信的候选节点。对于判断可比较的阈值，我们采纳从我们的实验研究得到的结果：当这些节点类型的得分差异率在10%以内，它们被视为“可比”。虽然不是全自动的，我们的推理方法仍然为没有语法的歧义关键词查询系统用户交互提供了指导。例如，如果关键词查询不明确，搜索引擎可以提供指导，让用户浏览和选择所需的节点类型，然后采用排序策略对单个匹配进行排序。

4.2 推断搜索条件节点类型

类似于推断所需的搜索目标节点，Intuition 1也有助于推断搜索条件节点类型。然而，与搜索目标要求节点类型与所有关键词相关联的情况不同的是，如果节点类型与某些（不一定是所有）关键词密切相关，则节点类型作为期望的搜索条件节点就具有足够高的置信度，因为一个查询可能需要多个节点类型作为搜索条件。例如，我们可以通过查询“name smith interest fashion”搜索名为“Smith”并对“fashion”感兴趣的客户，在这种情况下，系统应该能够高置信度地推断name和interest是搜索条件节点类型，即使关键词“interest”可能与name节点不相关。

因此，我们定义了节点类型T是期望的搜索条件节点类型的**置信度Cvia(T,q)**，如下：



（和（6）相比，（7）使用的是的和而不是乘积，因为节点类型只要与某些关键词相关，就具有足够高的置信度作为搜索条件节点。然后，每个可能的搜索条件节点类型的置信度将被纳入到XML TF\*IDF相似度里面（将在第5.2节讨论）。）

4.3 体现关键词共现

在这一部分中，我们讨论**数据节点**的搜索条件置信度。尽管统计学提供了一种计算**结构节点**类型作为搜索条件的置信度的宏观方法，仅靠它不足以推断出单个数据节点是查询中给定关键词的搜索条件的可能性。

**Example 6.** 考虑一个查询 “customer name Rock interest Art”，搜索姓名包含“Rock”和兴趣包含“Art”的顾客。基于统计数据，我们可以推断出name类型和interest类型的节点有较高的置信度是搜索条件，因为关键词“name”和“interest”分别在节点类型name 和interest高频率地出现。然而，统计数据不足以帮助系统推断出用户希望“Rock”是name的一个值和“Art”是interest的一个值，而在关键词共现捕获的帮助下这就是直观的了。因此，如果纯粹基于统计信息，搜索引擎很难将图1中的客户C4（(with name “Art” and interest “Rock”）与C3（(with name “Rock” and interest “Art”）区别开来。

（根据上面的例子，查询中的关键词共现模式提供了一种微观方法来测量单个**数据节点**作为搜索条件的可能性。因此，为了分别在查询和XML数据中体现关键词kt匹配v的祖宗节点的类型和关键词k匹配v中的一个值（如果它们的确存在于查询中）的共现，我们定义了以下距离。）

**Definition 4.1 ( In-Query Distance(IQD)).** 在查询q中关键词k和关键词/节点类型kt之间的**查询中距离IQD** *Dist q(q,kt,k)*，被定义为q中kt和k之间的**位置距离**的绝对值；否则， Dist q(q,kt,k) =∞ 。

注意，上述定义假定在查询q中没有重复的KT和K，查询q中两个关键词K1和K2的**位置距离**就是查询中K1位置和K2位置的差异。

（设计IQD是出于这种现象：当用户希望在关键词查询中同时指定谓词kt和它的值k时，尽管不同用户有不同的搜索习惯，他们总是将KT和K放在一起，不管一个特定用户将k指定在Kt之前或之后。）

**Definition 4.2 (Structural Distance (SD)).** kt和k关于数据节点v之间的**结构距离SD** *Dist s(q,v,kt,k)*，被定义为XML数据中v和最近的v的kt型祖宗节点之间的**深度距离**。

（我们设计IQD和SD分别用来表示在用户输入的查询和底层XML数据中节点类型kt和关键词k的亲密度。用直觉来思考，当在查询中和XML数据中这样的kt和k两者之间都是紧密相连的话，数据节点v就会受到青睐，就像Intuition 2中所陈述的和Definition 4.3中所体现的一样。）

**Intuition 2.** 对于数据节点v，如果在用户查询和XML数据中匹配祖宗节点类型的关键词kt和匹配v的关键词k都是紧密相连地出现，就可以直观地发现，v作为关于关键词kt和k的搜索条件具有很高的置信度。

**Definition 4.3 (Value-Type Distance (VTD)).** kt和k之间关于数据节点v的值型距离VTD *Dist(q,v,kt,k)*被定义为：

max(Dist q(q,kt,k),Dist s(q,v,kt,k)).

一般来说，Dist(q,v,kt,k)的值越小，查询q就越可能想要节点v作为搜索条件。因此，我们定义数据节点v是关于同时出现在查询q和节点v中的关键词k的搜索条件节点的置信度如下：



**Example 7.** 再考虑示例6中的查询q：“customer name Rock interest Art”。令n3和i3分别为客户C3姓名和兴趣下面的数据节点“Art Smith”和“rock music”。同样，令n4和i4分别为客户C4姓名和兴趣下面的数据节点“Rock Davis”和“art”。现在，name和Art之间的查询中距离Dist q(q,name,Art)=3；结构距离Dist s(q,n3,name,Art)=1；结果为Dist(q,n3,name,Art)=3和Cvia(q,n3,Art)=4/3。同样的，Cvia(q,i3,Rock)=1;Cvia(q,n4,Rock)=2;Cvia(q,i4,Art)=2。我们发现，客户C4的两个谓词比客户C3的两个谓词有更大的置信度成为搜索条件。直观地说，C4比C3更可能是q的结果。接下来我们将讨论如何将这些值纳入在我们的XML TF \* IDF相似度。

### 面向相关性的排序

在本节中，我们首先总结了XML中关键词搜索的一些独特特性，并解决了传统XML TF\*IDF相似度的局限性。然后，我们提出了一个新的XML TF \* IDF相似度，其中包括前面定义的置信度公式，以解决相关性排序中的关键词歧义问题。

5.1 XML关键词搜索原则

与平面文档相比，XML中的关键词搜索有其自身的特点。为了使IR风格的排序方法能顺利地应用于它，我们提出了搜索引擎应该遵守的三条原则。

**Principle 1:** 我们令查询的条件节点类型为**单值节点类型**V，期望的目标节点类型为D，理想情况下，只有被嵌入到V型节点里面的值和结构节点可以影响D型节点的相关度，而其他被嵌入D型节点里面的节点的存在不应该对D型节点的相关度产生影响。换句话说，以D型节点d为根的子树的大小（除非这棵子树以搜索条件节点为根）不应该影响d和查询的相关度。

**Example 8.** 当使用以street节点为搜索条件，customer节点为搜索目标的查询“Art Street”时，不管其它嵌入到C1和C3里面的节点的大小、值和结构，匹配关键词“street”的顾客节点（如图1中的C1）应该比没有匹配关键词“street”的顾客节点（如图1中的C3）排名更高。

**Principle 2:** 我们令查询的条件节点类型为**多值节点类型**V’，期望的目标节点类型为D，如果有D型节点d包含很多V’型节点，那么出现一个和查询相关的V’型节点通常就足够表明d是和查询相关的。换句话说，包含一个和查询相关的V’型节点的D型节点的相关度不应该太受其它和查询不相关的V’型节点影响。

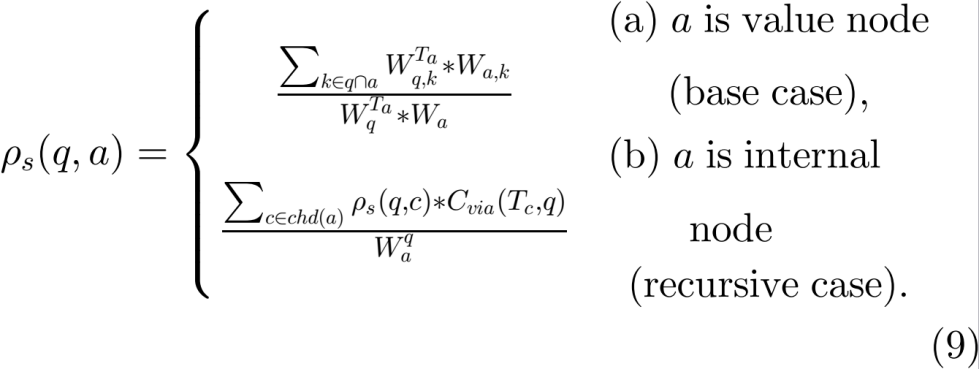
**Example 9.** 考虑使用查询“art”搜索对艺术感兴趣的顾客时，一个有“art”和其他很多兴趣的顾客（如图1中的C4）应该被视为比另一个没有“art”兴趣但是地址为“art street” 的顾客（如图1中的C1）和查询更相关。

（上述两个原则主要是说：查询结果的内部结构应该作为反映查询结果相关性的一个关键因素。）

**Principle 3:** 查询中关键词之间的距离对表达搜索意图很重要。

（如果我们知道准确的搜索条件节点，前两个原则看起来很简单。然而，当系统没有搜索条件节点类型的准确信息（就如大多数情况下，用户发起纯关键词查询）时，这些原则在设计XML TF\*IDF相似度公式中很重要。接下来，我们将利用它们设计XML TF\*IDF相似度公式。）

5.2 XML TF\*IDF相似度



我们提出了一个递归公式（9），它表明了XML的层次结构，用来计算一个期望的XML搜索目标类型的节点和一个关键词查询之间的XML TF\*IDF相似度。在公式（9）中，q表示一个关键词查询；a表示一个XML节点；ρs(q,a)表示q和a之间的相似度值。它首先（基础情况）计算XML数据的叶节点l和查询之间的相似度，然后（递归情况）基于内部节点n的每个孩子c和查询的相似度值和孩子c作为搜索条件节点类型的置信度，递归地计算内部节点n和查询之间的相似度，直到我们得到搜索目标节点的相似度。

（我们简单的讨论一下对公式（9）的直观感受。

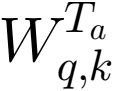
1.在基本情况下，我们用和原始TF\*IDF类似的方法计算XML叶节点和给定查询之间的相似度值，因为叶节点只包含没有进一步结构的关键词。

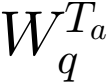
2.在递归的情况下：一方面，如果内部节点a有更多的与查询相关的子节点，而另一个内部节点a’则更少，那么很可能a比a’与查询更相关。（9b）中的分子反映了这种情况。另一方面，我们应该考虑内部节点的大小作为归一化因子，因为大的节点有更大可能包含更多与查询相关的子节点。（9b）中的分母反映了这种情况。）

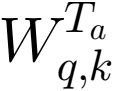
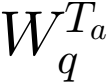
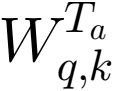
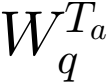
接下来，我们将说明公式（9）中的每个因子是怎么对 XML结构的相似度做出贡献的。

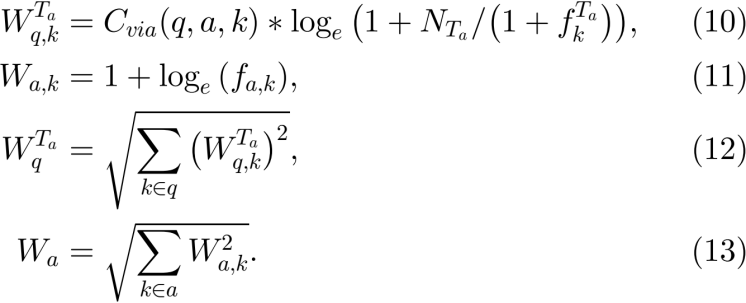
5.2.1 XML TF\*IDF的基本情况

由于XML叶节点只包含没有下层结构的关键词，所以我们采用原始TF\*IDF的方法通过使用统计术语和fa,k来计算叶节点和关键词查询之间的相似度。

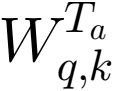
（然而，不同于原始TF\*IDF的Rule 1 给查询中关键词关于所有文档都是相同的权重（即公式（2）中的Wq,k），我们模型化并区分了查询中关键词关于不同XML叶节点类型的权重（即公式（10）中的）。也就是说，在原始TF\*IDF中，对于不同的文档，关键词k在查询q中的权重都是一样的，而在XML TF\*IDF中，对于不同的节点类型，关键词k在查询q中的权重是不一样的。如例10所示。）

**Example 10.** 关键词street可能非常频繁地出现在图1的address节点中，而很少出现在其它节点中。因此，区分在address节点中street的查询中权重（低）和其它节点中street的查询中权重（高）是很有必要的。同样的，我们区分一个查询关于不同XML节点类型的权重（即），而不是给查询对于所有文档都以一个固定的权重。

现在让我们仔细看看公式（9）。基础情况下，对于XML中的叶节点，每个K表示一个同时出现在查询q和节点a中的关键词；Ta是a的父节点类型；表示关于节点类型Ta的关键词k在q中的权重。Wa,k表示节点a中k的权重；表示q关于节点类型Ta的权重；Wa表示a的权重。遵守原始TF\*IDF的协议，我们分别为，Wa,k，和Wa提出了这些公式：



在公式（10）中，是Ta类型节点的总数量，而是包含关键词k的Ta类型节点的数量；Cvia(q,a,k)是节点a是关于关键词k的搜索条件节点的置信度。

在公式（11）中，fa,k是数据节点a中k出现的次数。和原始TF\*IDF中的Rule 1和Rule 2相似的是，是关于单调递减的，而Wa,k是关于fa,k单调递增的。Wa是关于a的大小一般地增加的，因此把它作为分母的一部分来发挥标准化因子的作用，使包含多个关键词的叶子节点和包含少数关键词的叶子节点之间达到平衡。

5.2.2 XML TF\*IDF的递归情况

公式（9）的递归情况，基于以下两种直觉以一种从底向上的方式递归地计算内部节点a和关键词查询q之间的相似度值。

**Intuition 3.** 如果内部节点a有一个孩子c，这个c的类型有很高的置信度是关于q的搜索条件节点类型（即Cvia(Tc,q)很大），而且c是和q高度相关的（即ρs(q,c)很大），则a和q是相关联的。

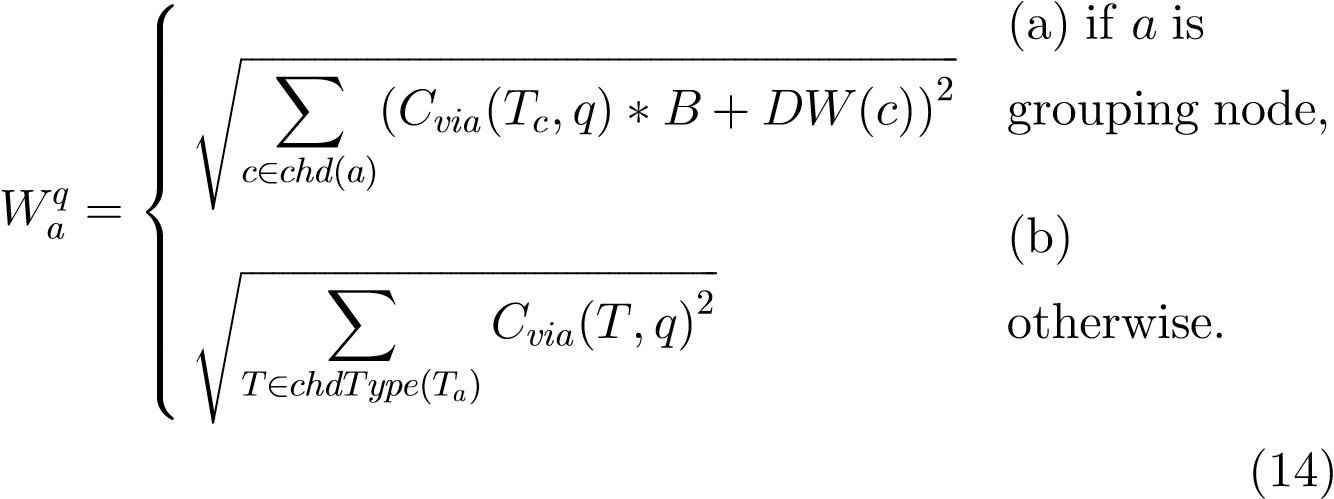
**Intuition 4.** 如果内部节点a有更多与查询相关的孩子节点，则a和q是更加相关的。

在（9）的递归情况下，c表示a的一个孩子节点；Tc是c的节点类型；Cvia(Tc,q)是Tc是搜索条件节点类型的置信度；ρs(q,c)表示节点c和查询q之间被递归计算的相似度；是a关于给定查询q的整体权重。

（接下来，我们解释（9）中内部节点a的相似度的设计：我们首先获取a的所有孩子的相似度值的加权和，其中每个孩子c的权重是c的搜索条件节点置信度。这个加权和正是（9）的分子，这也符合上面的Intuition 3和Intuition 4。另外，因为直觉4通常偏爱有更多孩子的内部节点，所以我们需要将a和q的相关度标准化，所以被用作分母。）

5.2.3 标准化因子设计

方程（14）提出了的设计，它被用作XML TF\*IDF相似度公式递归情况中的标准化因子，我们基于5.1中指出的Principle 1和Principle 2设计。



**分组节点情况：**公式（14a）提出了内部节点a是分组节点的情况；对于a的每个孩子节点c（即c∈chd(a)）,B被视为一个布尔标志：如果ρs(q,c)>0，则B=1，否则B=0；DW(c)

是一个较小的c的默认权重值，如果B=0,DW(c)=1/loge(e-1+|chd(a)|),如果B=1,DW(c)=0，其中|chd(a)|是a的孩子数，所以关于节点a的随着与查询不相关的孩子节点数增加而增加，但是为了反映Principle 2它增加地非常慢。然而，和Cvia(Tc,q)相比DW(c)通常是不重要的。

（接下来我们解释设计（14a）的原因。）

（对分组节点a的公式的直觉来自于Principle 2，所以除非c在其子树中包含一些与查询相关的关键词，否则我们不在归一化中计算Cvia(Tc,q)（q和c相关时，B才等于1）。这样的话，即使a有很多相同类型的与查询不相关的孩子节点，也不会对a和q的相似度产生很大的影响。同时，在已知默认权重DW(c)的情况下，如果两个分组节点包含相同的与查询相关的孩子集合，我们也提供了一种方法来区分并挑选总孩子数较少的分组节点。换句话说，在这种情况下，我们考虑了结果的特殊性。）

**非分组节点情况：** 当内部节点a是非分组节点时，我们基于a的类型而不是每个内部节点来计算。在（14b）中，chdType(Ta)表示a的孩子们的节点类型，即使每个a型内部节点可能有不同的孩子节点集合，它为所有的a型节点计算出相同的值。

这种设计有两个优点。第一个优点，它建模Principle 1来实现一个标准，即单个节点a的子树大小不会影响a与查询的相似度。

**Example 11.** 给定一个查询q“customer Art Street”,因为address有很高的置信度是搜索条件，根据（14b）中标准的，C1（地址在“Art Street”）将比C2（对“street art”感兴趣）排名更前。然而，如果我们基于每个独立节点的大小来计算标准化因子，那么address节点的高置信度将不会对C2（甚至没有address和street节点）的标准化因子做出贡献。因此，由于C2较小导致了较小的标准化因子，C2将比C1排名更前。

第二个优点，（14b）的设计在计算成本方面具有优势。在非分组节点的基于节点类型而不是节点来计算的情况下，(每次查询我们只需要对所有a型节点计算一次，

而不用重复地为每个a型节点都计算一次)。

（需要注意的是，（14b）中的归一化因子可能有利于有更多的嵌套节点类型的节点。然而，包含查询关键词但搜索置信度低的节点通常不足以超过具有高置信度的与查询相关的搜索条件节点。此外，我们不会为同一类型的所有节点选择相同的归一化因子，因为我们要让（9a）中单调递增免受内部节点相似度的影响，为了避免对那些嵌套在搜索目标节点附近的节点不利。）

在基本情况下，如果有很多的T型节点包含关键词k，那么k在T型节点中就不那么重要了。然而，我们认为T型节点对关键词k是重要的（即大的Cvia(T,k)）。这两个看起来有些矛盾，但其实都是准确的相关度排名的关键。

**Example 12.** 我们考虑用查询“customer art road”来搜索顾客的情况，在统计数据里面address通常比其他类型有更高的权重，因为关键词“road”高频率地出现在address里面。但是如果没有顾客住在“art road”，而有一些顾客住在“art street”，那么这些顾客就会比地址里面包含“road”而不包含“art”的顾客排名更高，因为“road”出现频率太高导致关键词“road”比关键词“art” 权重更低。

5.2.4 XML TF\*IDF的优点

**Compatibility（兼容性）.** XML的TF \* IDF相似度在半结构化和非结构化数据上都可以工作，因为非结构化数据是一种更简单的没有结构的半结构化数据，而且忽视节点类型的话可以轻松地将关于数据节点的XML TF\*IDF相似度（9a）简化为原始TF\*IDF（1）。

**Robustness（鲁棒性）.** 不像现有的需要查询结果覆盖所有关键词的方法，我们采用一种基于启发式的方法，不强迫查询结果中出现所有关键词；相反，我们根据与查询的相关性对结果排序。这样，就可以找到更多相关的结果，因为一个用户的查询通常不能将他的真实信息需求完全描述出来。用户绝不希望返回一个空的结果，即使没有结果可以包含所有的关键词，我们的方法仍然能够将最相关的结果返回给用户。

### **POPULARITY SCORE BY IDREF 通过IDREF计算流行度分数**

到目前为止，我们的XML的TF \* IDF相似度只反映了XML数据中一个搜索目标节点实例Nfor的相关度。然而，当两个或多个搜索目标节点实例具有可比的相关度分数时，是否有什么方法可以区分它们呢？答案是IDRef，它在一定程度上反映了一个查询结果的流行度。从用户的角度来看，当有很多结果具有可比的相关度分数时，希望首先返回最受欢迎的结果，从而节省用户的时间，让他们不用查看所有的结果来找到自己想要的结果。查询结果的相关度和流行度分数不应该被简单地结合起来，因为它们本质上被认为是正交的。例如，一个高度相关的查询结果可能会有一个非常低的人气分数，这可能导致一个较低整体排名分数，这是我们所不希望看到的。

对于一个T型的搜索目标节点实例NT（即一个以T型搜索目标节点为根的子树）的流行度，我们提出了以下准则：

**Guideline 4:** 以有指向NT（或其后代）的引用连接的节点为根的子树和查询的相关度越高，NT应该越受欢迎。

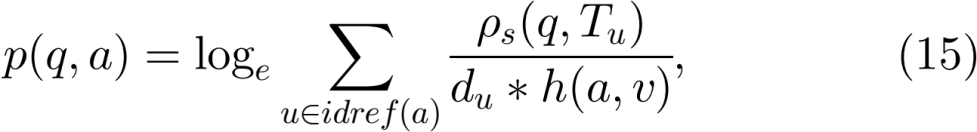
**Guideline 5:** 指向NT的引用连接数量越多，NT应该越受欢迎。

**Guideline 6:** 指向NT的引用连接越靠近，NT应该越受欢迎。

（直观地说，Guideline 4更支持那些被另一个相关度高的子树引用的查询结果。它还解释了为什么定义3.7限制只让（直接或间接）指向v的引用作为它的引用连接，而从它指向其它节点的引用被省略。准则5更支持那些被尽可能多的节点引用的查询结果；准则6更支持直接引用连接。）

**Example 13.** 考虑图1中被发布在StoreDB上的查询Q“XML keyword search”。假设n本书B1,...,Bn有可比的相关度分数。如果一本书Bi被尽可能多的其它书引用(Guideline 5)，而且引用Bi的书和查询Q也有很高相关度(Guideline 4)，那么Bi是受欢迎的；最后，直接引用也是被期望的(Guideline 6)，因为这反映了引用和被引用的书关系更密切。

具体化上述三个准则，我们如下定义了一个关于关键词查询q的T型搜索目标节点实例a的流行度p(q,a)：



其中idref(a)返回一个节点集u，u的每个节点都有一个指向v的引用连接，其中v是a或者它的后代；du表示Definition 3.7中定义的引用连接的距离（路径中IDREF边的数量）；h(a,v)表示节点a和v之间的高度距离。分子ρs(q,Tu)表示节点Tu或它的子树关于查询q的相似度，反映了准则4。Tu是u的父节点。分母du通过一个间接的IDREF关系减少节点Tu对节点a流行度的贡献，h(a,v)从a的后代推断对a的流行度的贡献，反映了准则6。集成函数的单调性特征用来反映准则5，而对数函数用来标准化每个这样的参与者u的原始相关度分数的影响。

### 算法

7.1 数据处理与索引构建

我们分析输入的XML文档，在这期间我们为每个访问过的节点n收集以下信息：1）给n分配一个杜威标签DeweyID；2）将n的前缀路径*prefixPath*作为它的节点类型存储在全局哈希表中，这样任何两个共享同一prefixpath的节点具有相同的节点类型；3）如果n是一个叶节点，我们创建一个数据节点a 作为它的孩子，并在同一时间算出两种基本统计数据fa,k和Wa。此外，我们还建立了两个索引，以加快关键词查询处理的速度。

构建的第一个索引被称为关键词倒排列表，它按文档顺序检索一个数据节点列表，列表中的数据节点的值都包含输入的关键词；此外，在每个倒排列表的顶部建立索引（例如，B +- 树）用来检索。特别是，我们为倒排列表设计和评估了三种候选方案：1）Dup，最基础的索引，它只存储DeweyID和XML TF fa,k；2）DupType，它比Dup额外存储了节点类型（即，它的前缀路径）；3）DupTypeNorm，它比DupType额外存储了一个与这个数据节点相关的标准化因子Wa。DupTypeNorm提供了XML TF \* IDF最有效的计算方法，因为它花费最少的索引查找时间；相比之下Dup和DupType需要额外的索引查找来获取Wa,k值，以计算Wa。

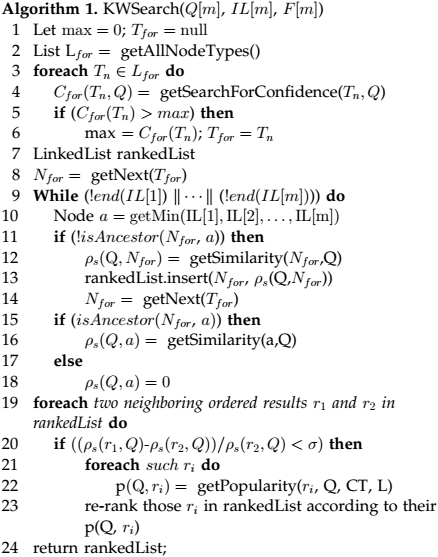
给定一个关键词k，倒排列表按文档顺序返回一组节点a，其中每个节点都包含输入的关键词并且是以数组的形式<DeweyID,prefixPath,fa,k,Wa>出现。为了便于解释算法，我们将这种数组命名为“Node”，它支持以下操作：

1. getDeweyID(a,k)返回数据节点a的DeweyID。
2. getPrefix(a,k)返回在XML数据中a的前缀路径。
3. getFrequency(a,k)返回数据节点a的XML TF fa,k。

构建的第二个索引被称为频率表，它为XML文档中关键词k和节点类型T的每个组合存储频率。最坏情况下空间复杂度为O(K\*N)，其中K是不同关键词的个数，N是XML数据库中节点类型的个数。因为在一个设计良好的XML数据库中节点类型的数量通常很小，所以频率表的大小相当于倒排表。频率表支持getFrequency(T,k)，返回值，可以用于第5节中的公式。

最后，在建立一个连接表CT来记录XML数据中节点之间的直接引用连接。CT中的每个记录都是<Dewey(v), cList(v)>的形式，其中cList(v)存储了一个按文档顺序的节点的DeweyID的列表，而RC(n,v)的距离d恒等于1。CT支持操作getCList(v)，用来返回cList(v)。我们在CT的顶部建立B+-树索引，用来快速检索cList(v)。

7.2 关键词搜索和排序

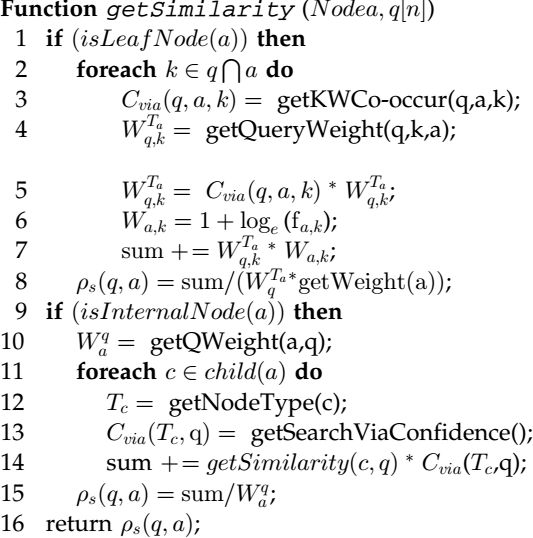


算法1给出了关键词搜索和结果排序的流程图。输入参数Q[m]是一个含有m个关键词的关键词查询。基于XML文档预处理后建立的倒排表，我们为查询中的每个关键词提取对应的倒排表IL[1],...,IL[m]。每个倒排表IL包含一组<DeweyID,prefixPath,,Wa>形式的数字元素。F是第7.1节中提到的频率表。特别的是，算法1分三步执行。

第一步，它确定用户的搜索意图，即确定最期望的搜索目标节点类型（第1-6行）。特别的是，它首先聚集XML文档中所有不同的节点类型（第2行）。然后，对于每个节点类型，我们通过公式（6）来计算它为搜索目标节点的置信度，并选择置信度最大的节点作为最期望的搜索目标节点类型Tfor（第3-6行）。

第二步，对于每个搜索目标节点候选者Nfor，计算n与给定关键词查询之间的XML TF \* IDF相似度（第7-18行）。我们维持rankedList来容纳每个搜索目标节点候选者的相似度（第7行）。Nfor最初被设置为在文档顺序中Tfor型的第一个节点（第8行）。我们以一个自下而上的方式递归地计算XML节点和给定查询之间的XML TF \* IDF相似度（第9-18行）：对于每个Nfor，它首先获取在文档顺序中第一个出现的节点a（第10行），然后调用函数getSimilarity()计算所有叶节点a的相似度，然后更进一步计算最低的内部节点的相似度（第15-18行），直到它达到Nfor节点，也就是前面计算的所有节点的根。如果N1是N2的祖宗，则函数isAncestor(N1,N2)返回true。然后，它计算现在的Nfor节点和查询之间的相似度（第12行），插入一对(Nfor,ρ)到rankeLlist里面（第13行），然后调用函数getNext()将指针移动到下一个Nfor并且以同样的方式计算下一个Nfor的相似度（第14行）。

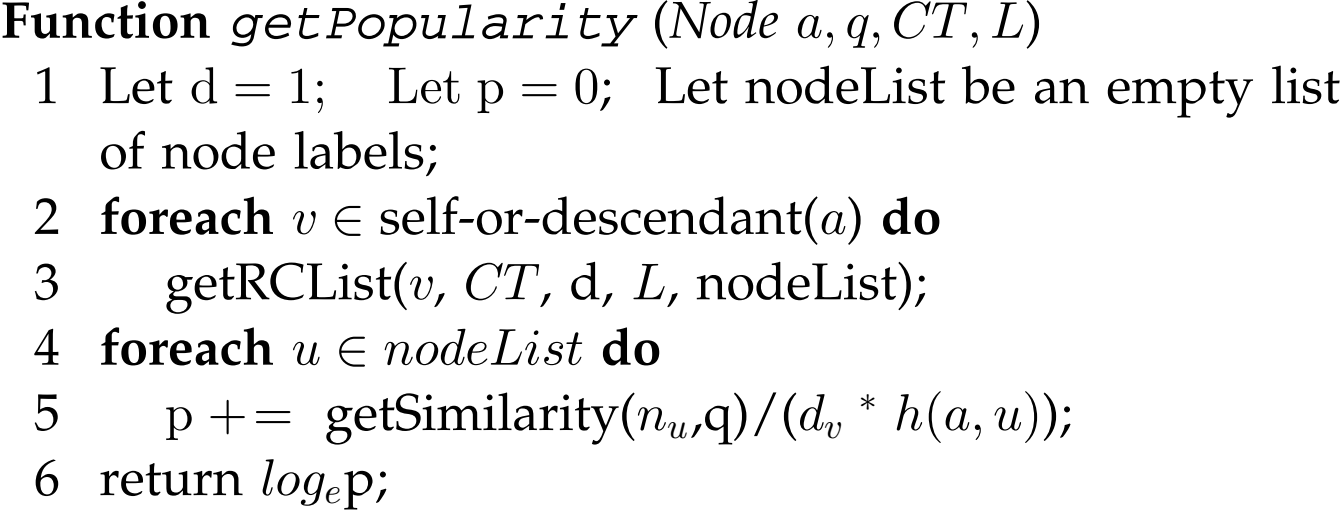
第三步，它收集rankedList里面相关度差小于指定阈值的结果（第19和20行），然后调用函数3计算他们的流行度，然后调整它们在rankedList里的排名位置（第21-23行）。

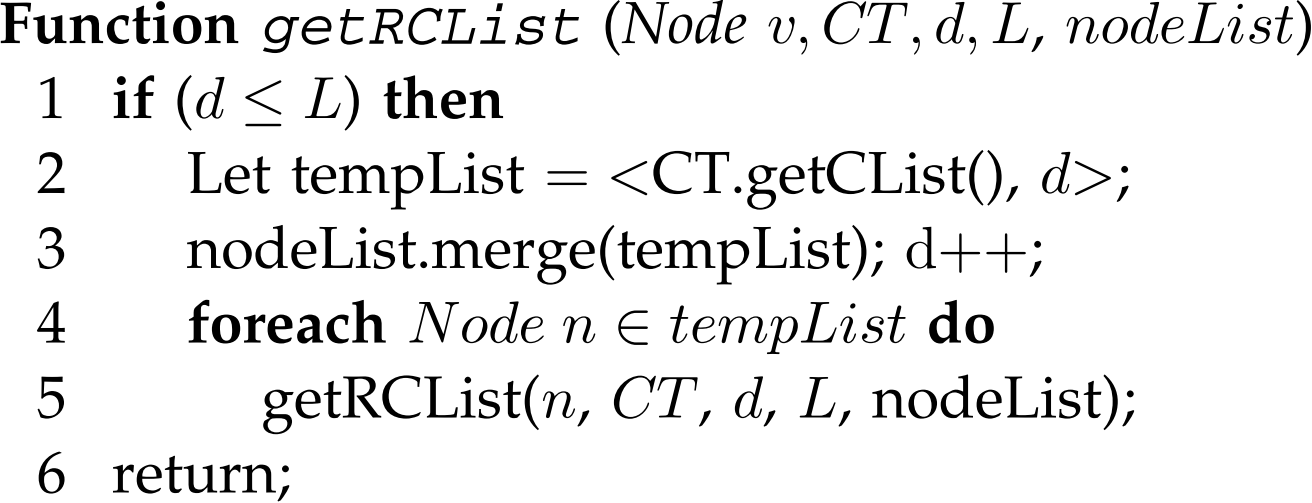


函数getSimilarity()介绍了计算节点a和给定大小n的查询q之间的XML TF\*IDF相似度的过程。我们考虑以下两种情况：

**Case 1:** a是一个叶节点（第1-8行）。对于同时出现在a和q中的每个关键词k，我们首先看k是否与匹配某些节点类型的关键词Kt同时出现。第3-8行介绍了公式（9a）中的ρs(q,a)的计算细节。第3,第5和第6行中的统计数据分别在公式（8）、（10）和（11）中进行了说明。

**Case 2:** a是一个内部节点（第9-15行）。我们按公式（9b）计算a关于查询q的相似度ρs(q,a)。我们首先计算a的每个孩子节点c的相似度和c成为搜索条件节点的置信度的乘积的和（第11-14行）。然后，ρs(q,a)被标准化（第15行），是a关于q的权重。最终，我们返回相似度值（第16行）。





函数getPopularity()分两步计算节点a关于查询q的流行度。第一步，调用函数getRCList，获取一列与a或者a的后代有引用连接RC的节点u（第1-3行），保存到nodeList里面。getRCList用递归的方式通过一个在CT上的深度有限的搜索找到这些节点u。在这里面，L是RC距离的上限，而且上面两个函数的所有变量的含义都和公式（15）中的一样。第二步，用公式（15）计算a的流行度（第4-6行）。

上面的搜索方法可以很好的适用于处理非结构化数据，它提供了一种简单的方法来将我们的排序技术结合到一个标准文本索引系统来处理非结构化和半结构化数据。