Gestão e Tratamento de Informação

Laboratório 10 – Exercícios

Nesta aula de laboratório iremos resolver exercícios relacionados com questões de qualidade de dados, nomeadamente abordando o problema da detecção de duplicados.

Tenha em atenção que o software usado nas aulas de laboratório oferece, como complemento às funções da biblioteca XPath 2.0, algumas **funções de extensão relacionadas com a medição de similaridade** entre cadeias de caracteres. Estas funções de extensão foram desenvolvidas com recurso ao SimPack, uma biblioteca de funções Java relacionadas com a medição de similaridade. Para mais informações sobre o SimPack deverá ser consultado o site http://www.ifi.uzh.ch/ddis/simpack.html.

Entre as várias funções de extensão disponíveis encontra-se a função `java:gti.similarity()`, a qual aceita três cadeias de caracteres como parâmetros de entrada. A primeira cadeia de caracteres corresponde ao nome da função de similaridade a ser utilizada, podendo tomar os seguintes valores

- **unnormalized-editdistance** – Função de distância entre cadeias de caracteres correspondente ao número mínimo de operações de edição necessárias para transformar uma das cadeias de caracteres na outra.

- **editdistance** – Função de similaridade entre cadeias de caracteres correspondente a uma versão normalizada (i.e., retornando valores entre 0 e 1) da distancia de edição. A função toma o valor máximo de 1 quando as cadeias de caracteres são exactamente iguais, tendendo para 0 à medida que as diferenças se acentuam.

- **jaro** – Função de similaridade entre cadeias de caracteres correspondente à medida originalmente proposta por Matthew Jaro (ver http://en.wikipedia.org/wiki/Jaro-Winkler). A função toma o valor máximo de 1 quando as cadeias de caracteres são exactamente iguais, tendendo para 0 à medida que as diferenças se acentuam.

- **jaro-winkler** – Função de similaridade entre cadeias de caracteres correspondente à extensão da medida de Jaro proposta por William Winkler (ver http://en.wikipedia.org/wiki/Jaro-Winkler). A função toma o valor máximo de 1 quando as cadeias de caracteres são exactamente iguais, tendendo para 0 à medida que as diferenças se acentuam.

As duas cadeias de caracteres seguintes correspondem aos dados que se querem comparar. Por exemplo, a expressão XPath `gti:similarity("editdistance", "Teste", "Testa")` ilustra a utilização da função de extensão descrita acima, retornando um valor de ≅ 0.8.

Exercício 1

Identificar os **duplicados**, sejam estes exactos ou aproximados, é um passo importante de um processo de **consolidação de dados**. Por **duplicados aproximados**, entendem-se os registos de informação que não são exactamente idênticos mas que ainda assim são duplicados (e.g., a mesma pessoa pode ser descrita pelas cadeias de caracteres "Maria A." ou "Maria Amélia").

A utilização de **funções de similaridade** é uma forma comum de abordar este problema. Dois registos com similaridade máxima correspondem a duplicados exactos, e dois registos com **elevada similaridade** têm uma **elevada probabilidade de serem duplicados**.

No URL https://dspace.ist.utl.pt/bitstream/2295/237039/1/cddb_data.xml encontra-se um documento XML contendo informação sobre discos compactos (CDs) e as músicas e artistas que se lhes encontram associados, contendo a descrição de 480 CDs escolhidos aleatoriamente da base de dados FreeDB (http://www.freedb.org/).

No URL https://dspace.ist.utl.pt/bitstream/2295/236816/1/cddb_dups.xml encontra-se um documento XML contendo informação sobre quais as descrições duplicadas de CDs que se encontram no primeiro ficheiro (i.e., os duplicados reais), sejam estes duplicados exactos ou aproximados.

1.1 - Escreva uma XQuery que, através da função `distinct-values()` da biblioteca XPath 2.0, permita retornar quais os títulos de CDs (i.e., os elementos `<dtitle>`) que se encontram no documento `cddb_data.xml`, eliminando os títulos que sejam duplicados exactos (i.e., mantendo apenas um título dos vários duplicados que forem encontrados).

1.2 - Escreva uma XQuery que, utilizando o resultado da alínea anterior, contabilize a diferença entre o número total de títulos de CD que se encontram no documento `cddb_data.xml`, e o número de títulos de CD distintos que se encontram descritos no mesmo documento.

1.3 - Escreva uma XQuery que, através da função `gti:similarity()`, permita retornar os pares de títulos de CDs no documento `cddb_data.xml`, que correspondam a duplicados exactos ou aproximados. Por duplicados aproximados, entende-se que devem ser considerados os pares de títulos com uma distância de edição inferior a 3.

1.4 – Uma solução simples para a alínea anterior envolve comparar cada um dos títulos de CDs descritos no ficheiro XML com todos os restantes títulos de CDs. Atendendo ao facto que se a diferença entre os tamanhos de duas cadeias de caracteres for superior a *x*, então a distância de edição entre as duas cadeias de caracteres é igual ou superior a *x*, optimize a XQuery da alínea anterior (i.e., escreva uma XQuery que apresente uma melhor

performance em termos de requisitos computacionais) com base numa técnica de filtragem que evite as comparações desnecessárias.

1.5 - Escreva uma XQuery que, através da função `gti:similarity()`, permita retornar quais os títulos de CDs que se encontram descritos no ficheiro `cddb_data.xml`, eliminando os títulos que sejam exactamente ou aproximadamente duplicados (i.e., manter apenas um título dos vários duplicados que forem encontrados). Por aproximadamente duplicados, entende-se que devem ser considerados os grupos de títulos com uma distância de edição inferior a 3.

1.6 – Atendendo às propriedades das funções *EditDistance*, *Jaro* e *JaroWinkler*, e assumindo as seguintes características para o problema da detecção de duplicados:

a) A performance computacional é um factor muito importante;

b) Os títulos de CDs duplicados devem-se muitas vezes a terminações diferentes nas cadeias de caracteres (e.g., "Best of Bowie 1" e "Best of Bowie – Vol 1" são duplicados);

Qual das funções de similaridade lhe parece ser mais adequada para a tarefa de encontrar os títulos de CDs aproximadamente duplicados? Justifique a sua resposta.

Exercício 2

A função `local:distinct-nodes()`, cujo código se apresenta abaixo, aceita como entrada uma sequência de nós XML. Esta função retorna uma sequência com os nós presentes na sequência de entrada, **eliminando os duplicados exactos** com base numa comparação entre os valores e os nós descendentes de cada nó na sequência de entrada (i.e., usando a função `deep-equal()` da biblioteca XPath 2.0 para comparar os nós e encontrar os duplicados exactos).

```
declare function local:distinct-nodes ($arg as node()*) as node()* {
   for $a at $apos in $arg
   let $before_a := fn:subsequence($arg, 1, $apos - 1)
   where every $ba in $before_a satisfies not(deep-equal($ba,$a))
   return $a
};
```

Figura 1 – Uma função XQuery que permite eliminar os elementos duplicados numa sequência.

2.1 – Utilizando a função `local:distinct-nodes()`, escreva uma XQuery que permita eliminar os elementos `<disc>` duplicados exactos que se encontrem no ficheiro XML com informação sobre os CDs (i.e., manter apenas um elemento `<disc>` dos vários duplicados que forem encontrados).

2.2 – Utilizando o ficheiro XML contendo informação sobre quais os elementos duplicados reais, escreva uma XQuery que permita contar quantos dos duplicados reais correspondem a duplicados aproximados.

2.3 – Com base no código da função `local:distinct-nodes()`, escreva uma nova função XQuery que melhor se adapte à remoção de duplicados no ficheiro XML com informação sobre os CDs. Além de detectar os elementos duplicados exactos, a nova função deverá ainda usar uma função de semelhança entre cadeias de caracteres por forma a contabilizar os CDs cujos títulos sejam duplicados aproximados.

Desta forma, os CDs duplicados deverão corresponder ao elementos `<disc>` exactamente iguais, ou aos elementos `<disc>` com um descendente `<dtitle>` que seja um duplicado aproximado de outro elemento `<dtitle>` existente no ficheiro.

2.4 – Usando o ficheiro XML contendo informação sobre quais os elementos que são duplicados reais, compare a qualidade dos resultados obtidos nas alíneas 2.1 e 2.3. Para tal, deverá escrever uma XQuery que contabilize quantos dos duplicados detectados por cada um dos mecanismos são duplicados reais (i.e., medir a **precisão**), e quantos dos duplicados reais são detectados por cada um dos mecanismos (i.e., medir a **abrangência**).

Nota informativa sobre o comportamento dos operadores union, except e intersect

Segundo a especificação do W3C para a linguagem XQuery, os operadores de conjuntos *union*, *except* e *intersect* procedem à **eliminação dos elementos duplicados exactos** que se encontrem nos resultados. Contudo, esta eliminação é feita com base na **identidade dos nós XML** (i.e., com base nos identificadores únicos atribuídos pelo motor de XQuery a cada um dos nós XML, aquando da sua leitura ou criação) e não com base nos valores associados aos próprios nós ou aos seus descendentes.

Usando um mecanismo semelhante ao da função `local:distinct-nodes()`, é relativamente simples definir funções XQuery que correspondam a estes operadores, mas que apresentem uma semântica de eliminação de duplicados diferente (i.e., com base nos valores e na estrutura dos elementos). A Figura 2 apresenta o código XQuery para uma função *union* que retorne a união de duas sequências, eliminando os duplicados exactos.

```
declare function local:union ($arg1 as node()*, $arg2 as node()*) as node()* {
 let $arg := $arg1 union $arg2
 for $a at $apos in $arg
 let $before_a := fn:subsequence($arg, 1, $apos - 1)
 where every $ba in $before_a satisfies not(deep-equal($ba,$a))
 return $a
};
```

Figura 2 – Uma função XQuery que retorna a união de duas sequências, eliminando os duplicados exactos.