

Gestão e Tratamento de Informação

1º semestre

Resolução do Mini-Projecto 2 - Grupo 14

Frederico Sabino
António Lopes
Francisco DuarteNº73239
Nº73721
Nº73838

Solutions for Exercise 1

Question 1.1 - Solution in text file 1_1.xq

```
declare function local:find-politician-pairs($doc) {  
  let $politicians := $doc//politician  
  let $politicians_bigrams := local:computeBigrams($politicians)  
  let $number_of_pols := fn:count($politicians)  
  
  let $politicians_jaccard := (for $position in 1 to $number_of_pols  
    return <politician name="{ $politicians[$position]/@name}">  
      {if($position = $number_of_pols)  
        then  
          <jaccard value="0" />  
        else  
          (for $positionfront in $position + 1 to  
            $number_of_pols  
            return  
              <jaccard  
value="{local:computeJaccard($politicians_bigrams[$position],  
$politicians_bigrams[$positionfront])}"  
              poll="{ $politicians_bigrams[$position]/@name}"  
              party1="{ $politicians_bigrams[$position]/@party}"  
              pol2="{ $politicians_bigrams[$positionfront]/@name}"  
              party2="{ $politicians_bigrams[$positionfront]/@party}" />  
            )  
          }  
        </politician>  
      )  
  )  
  
  let $thresh := 0.5  
  let $final_politicians := <pairs> {(for $jaccard in $politicians_jaccard//jaccard  
    where $jaccard/@value >= 0.5  
    return  
      <pair>  
        <politician name="{ $jaccard/@poll}"  
party="{ $jaccard/@party1}" />  
        <politician name="{ $jaccard/@pol2}"  
party="{ $jaccard/@party2}" />  
        </pair>  
      )} </pairs>  
  )  
  return $final_politicians;  
};
```

```

declare function local:computeJaccard($politician1, $politician2) {
  let $number_equal_bigrams := fn:count(fn:distinct-values(for $bigram1 in
    $politician1//bigram, $bigram2 in $politician2//bigram
      where fn:compare($bigram1/@value, $bigram2/@value) = 0
      return
        $bigram1/@value
    ))
  let $number_bigrams_pol1 := fn:count(fn:distinct-values(for $bigram1 in
    $politician1//bigram
      return $bigram1/@value))
  let $number_bigrams_pol2 := fn:count(fn:distinct-values(for $bigram2 in
    $politician2//bigram
      return $bigram2/@value))
  let $all_bigrams := (($number_bigrams_pol1 - $number_equal_bigrams) +
    ($number_bigrams_pol2 - $number_equal_bigrams) + $number_equal_bigrams)
  return $number_equal_bigrams div $all_bigrams;
};

declare function local:computeBigrams($politicians) {
  let $pol_bigrams := (for $politician in $politicians
    return
      <politician name="{ $politician/@name}" party="{ $politician/@party}">
        {for $position in 1 to fn:string-length($politician/@name)
          return
            if($position = 1)
            then
              <bigram value="{fn:concat('#', fn:substring($politician/@name,
                $position, 1))}" />
            else
              if($position = fn:string-length($politician/@name))
              then
                (<bigram value="{fn:substring($politician/@name, $position -
                  1, 2)}" />,
                  <bigram value="{fn:concat(fn:substring($politician/@name,
                    $position, 1), '#')}" />)
              else
                <bigram value="{fn:substring($politician/@name, $position -
                  1, 2)}" />
            }
          </politician>)
    return $pol_bigrams;
  };

  local:find-politician-
  pairs(doc("file:///afs/ist.utl.pt/users/2/1/ist173721/GTI/Proj3/Politicians.xml"));

```

Question 1.2 - Solution in text file 1_2.xq

```

declare function local:processClusterList($clusters) {
let $processedCluster :=
(
    for $position in 1 to fn:count($clusters)
    return local:processCluster($clusters, $position)
)

let $removedElementDuplicates :=
(
    for $elem in $processedCluster
    return local:removeDuplicatedElements($elem)
)

let $removedClusterDuplicates := local:removeDuplicatedClusters($removedElementDuplicates)

return
    if (deep-equal($clusters, $removedClusterDuplicates))
    then $removedClusterDuplicates
    else local:processClusterList($removedClusterDuplicates)
};

declare function local:removeDuplicatedClusters($clusterList) {
    for $position in 1 to count($clusterList)
    where not(some $nodeInSeq in subsequence($clusterList, $position+1) satisfies
local:sequence-node-equal-any-order($nodeInSeq/politician,
$clusterList[$position]/politician))
    return $clusterList[$position]
};

declare function local:sequence-node-equal-any-order($seq1, $seq2) as xs:boolean {
    let $diff1 := (for $elem1 in $seq1
        where not(local:is-node-in-sequence($elem1, $seq2))
        return $elem1)
    let $diff2 := (for $elem2 in $seq2
        where not(local:is-node-in-sequence($elem2, $seq1))
        return $elem2)
    return empty(($diff1, $diff2))
};

declare function local:removeDuplicatedElements($cluster) {
<cluster>
{for $position in 1 to count($cluster/politician)
where not(local:is-node-in-sequence($cluster/politician[$position],
subsequence($cluster/politician, $position+1)))
return $cluster/politician[$position]}
</cluster>
};

declare function local:is-node-in-sequence($node as node()?, $seq as node()* ) as
xs:boolean {
    some $nodeInSeq in $seq satisfies deep-equal($nodeInSeq,$node)
};

(: returns a new cluster as a result of joining with the rest of the clusters :)
declare function local:processCluster($clusters, $clusterPosition) {
    let $list := (
        for $position in ((1 to $clusterPosition - 1),($clusterPosition+1 to
fn:count($clusters)))
        return
            if(local:isSimiliarClusters($clusters[$position],
$clusters[$clusterPosition]))
            then
                <cluster>
                {
                    (for $politician in $clusters[$position]//politician
                    return $politician,
                    for $politician in $clusters[$clusterPosition]//politician
                    return $politician)
                }
                </cluster>
            else()
    )
    return
        if(empty($list))

```

```

        then $clusters[$clusterPosition]
        else $list
    };

(: a cluster is similar with another if it has at least one equal element :)
declare function local:isSimilarClusters($cluster1, $cluster2) {
let $commonElements :=
    (for $element1 in $cluster1//politician, $element2 in $cluster2//politician
     where deep-equal($element1, $element2)
     return $element1)
return not(empty($commonElements))
};

<clusters>{local:processClusterList(doc("file:///afs/ist.utl.pt/users/2/1/ist173721/GTI/Pr
oj3/pairs.xml")//pair)}</clusters>;

(:local:processClusterList((<pair><politician name="a" party="PSD"/><pair name="b"
party="PS"/><politician name="c" party="PSD"/></pair>,<pair><politician name="b"
party="PS"/><politician name="c" party="PSD"/></pair>,<pair><politician name="d"
party="PSD"/><politician name="e" party="PSD"/></pair>)):)

```

Question 1.3 - Solution in text file 1_3.xq

```

declare function local:cleanPoliticians($politiciansList, $clusters) {
<politicians>
{
let $cleanedPoliticians := (
for $politician in $politiciansList//politician
return
    let $occurrenceInCluster := (
for $cluster in $clusters//cluster
return
    if (some $nodeInSeq in $cluster//politician satisfies deep-
equal($nodeInSeq, $politician))
    then local:getPoliticianFromCluster($cluster)
    else ())
return
    if (empty($occurrenceInCluster))
    then $politician
    else $occurrenceInCluster
)
return local:distinct-nodes($cleanedPoliticians)
}
</politicians>
};

declare function local:getPoliticianFromCluster($cluster) {
<politician
name="{
let $maxLength := max(for $politicianName in $cluster//politician/@name
return string-length($politicianName))
let $biggestName := (for $politician in $cluster//politician
where string-length($politician/@name) = $maxLength
return $politician/@name)
return $biggestName[1]
}" party="{
let $minLength := min(for $politicianParty in $cluster//politician/@party
return string-length($politicianParty))
let $shortestParty := (for $politician in $cluster//politician
where string-length($politician/@party) = $minLength
return $politician/@party)
return $shortestParty[1]
}"/>
};

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
};

(:
let $politicians := (
<politicians>
<politician name="A" party="P" />

```

```

        <politician name="B" party="PS" />
        <politician name="D" party="PS" />
        <politician name="F" party="OLA" />
    </politicians>
)
:)

let                                     $politicians                               :=
doc("file:///afs/ist.utl.pt/users/2/1/ist173721/GTI/Proj3/Politicians.xml")

(
let $clusters := (
<clusters>
    <cluster>
        <politician name="AB" party="PSX" />
        <politician name="A" party="P" />
        <politician name="B" party="PS" />
        <politician name="AC" party="x" />
    </cluster>
    <cluster>
        <politician name="D" party="PS" />
        <politician name="DRE" party="y" />
        <politician name="E" party="PS" />
    </cluster>
</clusters>
)
:)

let $clusters := doc("file:///afs/ist.utl.pt/users/2/1/ist173721/GTI/Proj3/clusters.xml")
return local:cleanPoliticians($politicians, $clusters)

```

Solutions for Exercise 2

Question 2.1

GreatPoliticians(p):- Review(p,_,_,score), score>4
 GreatParties(p, pt):- Review(p,_,_,score), Politician(p,pt), score>4

Question 2.2

Entre as duas queries não podem existir relações de "containment" ou equivalência visto que ambas têm uma aridade diferente, a query Q1 tem aridade 2 e a Q2 tem aridade 1.

Se considerarmos a query Q1'(n):- Q1(n,t) neste caso podemos verificar que existe uma relação de equivalência entre as queries Q1'(n) e

Q2(p), estas queries são equivalentes. Depois de feito o "unfold" as condições/restrições a que as queries estão sujeitas são as mesmas.

$Q1'(n) \sqsubseteq Q2(n)$
 $Q2(n) \sqsubseteq Q1'(n)$
 $Q1'(n) \equiv Q2(n)$

Question 2.3

Q3('Joaquim Silva', pt):- Review(p,date,topic,score),
Review(p,date',topic',score'),Politician(p,pt), score>4, score'>4, p='Joaquim Silva'

Solutions for Exercise 3

Para a solução dos problemas 3.1 e 3.2 considera-se $x = \text{HEPOCARTES}$ e $y = \text{EPOCRATES}$

Question 3.1

Jaro:
tamanho de $x = 10$
tamanho de $y = 9$
 $\text{gap} = g = \min(10, 9) / 2 = 4$, arredondado para “baixo”
 $t = 2$
 $C = \text{EPOCARTES} = 9$
 $\text{Jaro}(x, y) = 1/3 * (9/9 + 9/10 + (9 - 2/2)/9) = 251/270 = 0.9296$ aprox.

Question 3.2

Jaccard:
 $B_x = \{\#H, HE, EP, PO, OC, CA, AR, RT, TE, ES, S\# \}$
 $B_y = \{\#E, EP, PO, OC, CR, RA, AT, TE, ES, S\# \}$
 $J(x, y) = |B_x \cap B_y| / |B_x \cup B_y| = 6/15 = 2/5 = 0.4$

Question 3.3

Os resultados obtidos em 3.1 e 3.2 permitem concluir que as duas strings são bastante semelhantes, quer tendo em conta o valor obtido pelo Jaro que é facilmente visível que 9 em 10 caracteres existem em x e em y , quer tendo em conta o valor obtido pelo Jaccard que nos indica que do subset de bigramas de x e y praticamente metade ocorre em ambos. Assim, é possível comparar ambas as métricas na medida em que o Jaro tem uma melhor performance quando a comparar strings curtas, como é o caso, e, comparativamente ao Jaccard, quando existem transposições lida melhor com as mesmas visto que quando ocorre uma transposição no Jaccard são menos elementos que ocorrem nos dois subsets e portanto a semelhança é menor; no Jaccard a semelhança é computada tendo em conta apenas os n -gramas comuns dos subsets em relação aos possíveis n -grams dos subsets.

Solutions for Exercise 4

Question 4

Phase 1 (v computation):

$$v = \{ \begin{array}{l} \{C1, C3, C5\}, \{C1, C4, C5\}, \{C2, C3, C5\}, \{C2, C4, C5\}, \\ \{C1, C3\}, \{C1, C4\}, \{C2, C3\}, \{C2, C4\}, \{C1, C5\}, \{C2, C5\}, \{C3, C5\}, \{C4, C5\}, \\ \{C1\}, \{C2\}, \{C3\}, \{C4\}, \{C5\} \end{array} \}$$

Phase 2 (ζ computation with path finding):

$$\zeta = \{ \begin{array}{l} \{C1, C4, C5\}, \\ \{C1, C4\}, \{C2, C3\}, \{C1, C5\}, \{C4, C5\}, \\ \{C1\}, \{C2\}, \{C3\}, \{C4\}, \{C5\} \end{array} \}$$

Phase 3 (Covers found from ζ):

Covers:

C1: $\{\{C1, C4, C5\}, \{C2, C3\}\}$
C2: $\{\{C1, C4\}, \{C2, C3\}, \{C1, C5\}\}$
C3: $\{\{C1, C4\}, \{C2, C3\}, \{C4, C5\}\}$

...

Other covers are possible with $\{CN\}$ but involves more candidate sets thus they won't be selected by the algorithm

The algorithm selects C1 because it has the smallest number of candidate sets.

Phase 4 (Schema Mapping Expression as an SQL Query):

$$\{\{C1, C4, C5\}, \{C2, C3\}\}$$

```
SELECT name, opinion, party
FROM Politician p, Speech s
WHERE p.NameP = s.nameP
UNION
SELECT name, opinion, NULL
FROM Journalist j, Article a
WHERE j.NameJ = a.NameJ
```

Solutions for Exercise 5

Question 5.1

A dificuldade do processo de mapeamento de esquemas (schema mapping) deve-se sobretudo à heterogeneidade das fontes de dados.

Os dados existentes nas diversas fontes podem não estar organizados da mesma forma daquela pretendida no esquema mediador e até podem estar organizados de forma diferente entre eles. Por este motivo é preciso que o mapeamento tenha em conta certos detalhes, mesmo com o conhecimento das correspondências. Pode acontecer haver dados incompletos numa fonte, como por exemplo numa tabela com nome, número de telefone e morada, pode acontecer um registo não ter número de telefone, neste caso o mapeamento deve ter em conta esta possibilidade e possivelmente tentar preencher os detalhes em falta.

(No processo de mapeamento é necessário definir as transformações necessárias a aplicar aos dados vindos das fontes para que estes estejam no formato pretendido do esquema mediador). Durante este processo pode ainda ser preciso combinar dados de múltiplos atributos (através de operações como joins, operações matemáticas...). Esta combinação pode não ser trivial de encontrar pois é necessário analisar grandes conjuntos de dados dos dois esquemas para se chegar a uma combinação correcta. Pode ainda existir mais de uma forma de combinar os dados para chegar ao objectivo pretendido. Por exemplo para calcular o número de encomendas total feitas numa loja online podem existir uma ou mais formas de chegar ao valor pretendido dependendo dos dados disponíveis, vamos assumir que é possível chegar ao valor somando o número de encomendas feitas por cada cliente ou então somar o número de encomendas feitas a cada armazém. O mapeamento deve ter estas alternativas em conta e escolher a melhor.

Question 5.2

O propósito destas dimensões é permitir a análise da qualidade de dados segundo diferentes perspectivas (dimensões). Ou seja, o objectivo é fornecer ferramentas, neste caso métricas, que permitem analisar os dados de forma mais objectiva e de certa forma quantificar a sua qualidade segundo diferentes perspectivas como a completude, consistência, precisão, etc.