Nom :
Prénom :

ASWS – Apprentissage Symbolique et Web Sémantique

Examen Réparti 5I802 1ère session du 12 nov 2014

2 heures

CORRIGÉ

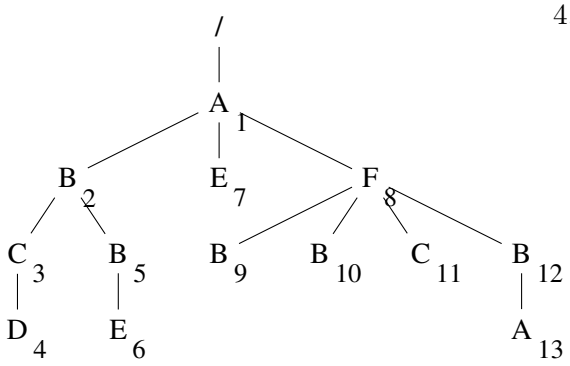
Documents autorisés

Les téléphones mobiles doivent être éteints et rangés dans les sacs. Le barème sur 20 points (11 questions) n'a qu'une valeur indicative.

1 XML/XPath (5 pts)

Soit donné l'arbre XML suivant où chaque noeud est identifié par un attribut `@id` l'identifiant de la racine est égal à 0:

Arbre XML:



Trois expressions XPath qui retournent la liste des noeuds $4, 6, 7, 9, 10, 11, 13$ (feuilles de l'arbre):

1. $/descendant :: *[not(child :: *)]/attribute :: id$ (syntaxe étendue)
2. $// * [not(*)]/@id$ (syntaxe abrégée)
3. $/descendant :: *[not(*/*)]/child :: */@id$ (syntaxe mixte)

Question 1 (3 points)

Donnez pour chaque expression XPath ci-dessous la liste des identifiants des noeuds retournés.

1. $/descendant :: B/descendant :: */@id$

   Solution: $3, 4, 5, 6, 13$

2. $/descendant :: B[child :: *]/@id$

   Solution: $2, 5, 12$

3. $/descendant :: */child :: *[1]/@id$

   Solution: $2, 3, 4, 6, 9, 13$

4. $/descendant :: */following - sibling :: B[1]/@id$

   Solution: $5, 10, 12$

5. $/descendant :: E[not(following - sibling :: *)]/@id$

Solution: 6

6. $/descendant :: *[child :: E[not(following - sibling :: *)]]/@id$

Solution: 5

Question 2 (2 points)
Donnez pour chaque séquence de noeuds ci-dessous une expression XPath (syntaxe étendue ou abrégée) qui la calcule:

1. $6, 7$

Solution: $/descendant :: E/@id$

2. $2, 9, 10$

Solution: $/descendant :: B[following - sibling :: *]/@id$

3. $5, 7, 10$

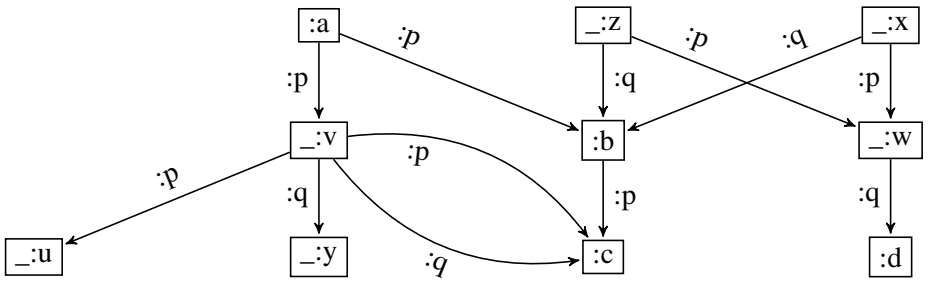
Solution: $/descendant :: */child :: *[2]/@id$

4. $1, 2, 8$

Solution: $/descendant :: B/parent :: */@id$

2 Graphes RDF (4 pts)

Soit le graphe RDF $g1$ suivant:



On suppose que les noeuds et les propriétés appartiennent à l'espace de nom `http://exemple.org/`. Chaque noeud est étiqueté par son identifiant ou par une lettre précédée par '_:' s'il s'agit d'un noeud blanc. Les arcs sont étiquetés par les types des propriétés.

Question 3 (1 point)
Est-ce que l'expression Turtle représente un graphe équivalent au graphe $g1$?

```
@prefix : <http://exemple.org/> .
:a :p [ :q _:b0 ; :p _:b2 ; :p :e ; :q :e ] ; :p :b .
:b :p :e .
_:b3 :q :b; :p _:b1 .
[] :p _:b1; :q :b .
_:b1 :q :f .
```

Solution: oui ■ non □

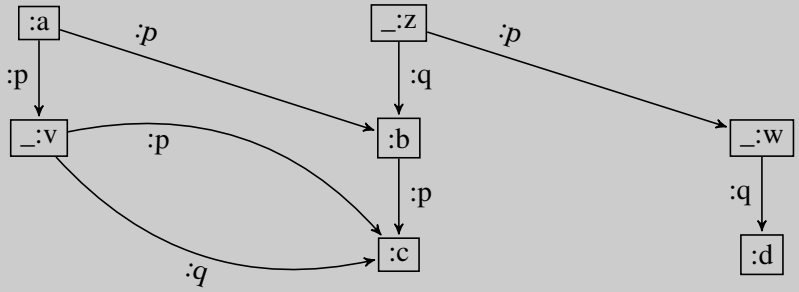
Question 4 (1 point)

Donnez l'instance minimale $core(g1)$ (forme normale) du graphe $g1$ en format Turtle et le *mapping* m pour l'obtenir.

Solution: $m : \_:y \rightarrow c, \_:x \rightarrow \_:z, \_:u \rightarrow c$:

Turtle:

```
@prefix : <http://exemple.org/> .
:a :p [ :p :e ; :q :e ] ; :p :b .
:b :p :e .
[] :q :b; :p :d .
:d :q :f .
```



Soit le schéma RDF/S $schema.ttl$:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>
@prefix : <http://exemple.org/> .
:A a rdfs:Class .
:B a rdfs:Class ; rdfs:subClassOf :A .
:p a owl:TransitiveProperty ; rdfs:domain :A ; rdfs:range :A .
:q a rdf:Property ; rdfs:subPropertyOf :p; rdfs:range :B .
```

Le graphe saturé $g1_{sat} = Rules_{RDF/S}(g1)$ correspond au graphe RDF/S qu'on obtient quand on applique l'ensemble des règles RDF/S vu en cours au graphe $g1$.

Question 5 (2 points)

Donnez pour chaque ressource identifiée `:a`, `:b`, `:c` et `:d`, les classes dont elles sont des instances dans le graphe saturé $g1_{sat}$. Remarque: la propriété `:p` *est transitive* et la propriété *`:q` est une sous-propriété de* `:p`

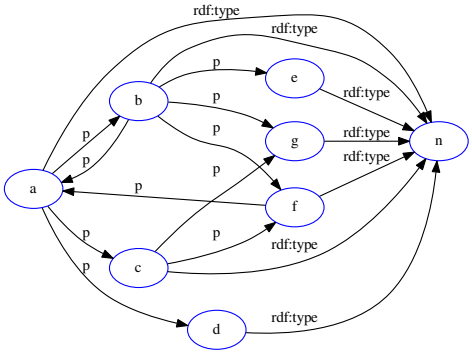
Solution:

```
:a      a       :A , rdfs:Resource .
:b      a       :B , :A , rdfs:Resource .
:c      a       :B , :A , rdfs:Resource .
:d      a       :B , :A , rdfs:Resource .
```

3 SPARQL (7 pts)

Soit le graphe dirigé RDF/S *graphe.ttl*:

```
@prefix : <http://exemple.org/> .
:a a :n; :p :b, :c, :d .
:b a :n; :p :a, :e, :f, :g .
:c a :n; :p :f, :g .
:d a :n .
:e a :n .
:f a :n; :p :a .
:g a :n .
```



Question 6 (1 point)

Donnez les requêtes SPARQL suivantes sur le graphe RDF/S *graphe.ttl*. Vous n'êtes pas obligé de spécifier les espaces de nom utilisés ni le graphe interrogé (expressions SELECT... WHERE ...).

1. Les noeuds qui n'ont pas de propriété `:p` (sans arc `:p` sortant).

Solution:

```
PREFIX : <http://exemple.org/>
SELECT ?x
FROM <graph.ttl>
WHERE { ?x a :n .
  OPTIONAL { ?x :p ?y . }
  FILTER (!bound(?y)) }
```

2. Les *cycles* de longueur 2. La requête retourne les couples de noeuds sur le cycle.

Solution:

```
SELECT DISTINCT ?x ?y
FROM <graph.ttl>
WHERE { ?x ?p ?y . ?y ?q ?x .
        FILTER (?y != ?x) }
```

Question 7 (1 point)

Définissez un ensemble de *règles forward* qui génèrent un triplet `:a :q :b` entre toutes les paires de noeuds `(:a,:b)` reliés par un chemin de propriétés `:p` dans *graphe.ttl* (fermeture transitive de la propriété `:p`).

Solution:

```
@prefix : <http://exemple.org/> .
```

```
    [trans1: (?x :p ?y) -> (?x :q ?y)]
    [trans2: (?x :q ?y)(?y :q ?z) -> (?x :q ?z)]
```

Question 8 (2 points)

Définissez une requête SPARQL qui génère un nouveau graphe *trans.ttl* qui contient le graphe *graphe.ttl* et un triplet `:a :q :b` entre toutes les paires de noeuds `(:a,:b)` reliés par un chemin de propriétés `:p` dans *graphe.ttl* (fermeture transitive de la propriété `:p`). Remarque: vous pouvez utiliser la fonction `BIND`.

Solution:

```
PREFIX : <http://exemple.org/>
CONSTRUCT { ?x ?p ?y . }
FROM <graph.ttl>
WHERE {
       { ?x ?p ?y . }
       UNION
       { ?x :p+ ?y .
         BIND ( :q as ?p ) }
}
```

Question 9 (1 point)

Soit le document `tree.ttl` et la requête SPARQL suivants:

```
@prefix : <http://exemple.org/> .
:a :p :b, :c .
:b :p :d .
:c :p :e, :f .
```

```
prefix : <http://exemple.org/>
select ?b (count(?m) as ?x)
  from <tree.ttl>
 where { :a :p* ?m .
         ?m :p+ ?b . }
 group by ?b
 order by ?b
```

Donnez le résultat de la requête et *expliquez en français* ce qu'elle calcule?

Solution:

b	x
:b	1
:c	1
:d	2
:e	2
:f	2

La requête retourne pour chaque noeud sa distance de la racine `:a`.

Question 10 (2 points)

Calculez pour chaque noeud dans le graphe *graphe.ttl*, le degré entrant (nombre d'arcs entrant) *indeg* et le degré sortant (nombre d'arcs sortants) *outdeg*. Le résultat est une table avec trois colonnes $(noeud, indeg, outdeg)$:

node	indeg	outdeg
:f	2	1
:d	1	0
:c	1	2
:b	1	4

```
| :g    | 2       | 0        |
| :a    | 2       | 3        |
| :e    | 1       | 0        |
```

Solution:

```
prefix : <http://exemple.org/>
select (?n as ?node)
       (count(distinct ?a) as ?indeg)
       (count(distinct ?b) as ?outdeg)
from <graph.ttl>
 where {
 optional { ?a :p ?n }
 optional { ?n :p ?b }
}
group by ?n
```

4 Triple Store (4 pts)

Le graphe *trans.ttl* (avec deux types de propriétés `:p` et `:q`) est stocké dans

1. une table $Triples(subject, prop, object)$ et
2. une table $Proptable(subject, p, q)$.

Toutes les URI sont stockées sous forme de chaînes et avec préfixe. Par exemple, la propriété `http://example.org/p` est stockée sous forme d'une chaîne `':p'` etc.

Question 11 (4 points)
Traduisez les requêtes SPARQL suivantes en requêtes SQL sur ces deux tables.

1.
```
prefix : <http://example.org>
select ?a ?c
 where { ?a :p ?b .
         ?b :q ?c . }
```

Solution:
SQL sur $Triples(subject, prop, object)$:

```
select t1.sujet, t2.objet
from Triples t1, Triples t2
where t1.objet = t2.sujet
and t1.prop=':p' and t2.prop=':q'
```

SQL sur $Proptable(subject, p)$:

```
select t1.sujet, t2.q
from Proptable t1, Proptable t2
where t1.p = t2.sujet
```

2.
```
prefix : <http://example.org>
select ?a ?b
 where { ?a :p ?b .
         OPTIONAL { ?b :q ?a } }
```

Solution: SQL sur $Triples(subject, prop, object)$:

```
select t1.sujet, t2.objet
from Triples t1 left outer join Triples t2
     on (t1.objet = t2.sujet and t1.sujet = t2.objet)
where t1.prop=':p' and t2.prop=':q'
```

SQL sur $Proptable(subject, p)$:

```
select t1.sujet, t2.q
from Proptable t1 left outer join Proptable t2
     on (t1.p = t2.sujet and t1.sujet = t2.q)
```

3.
```
prefix : <http://example.org>
select ?a
 where { ?a :p ?b .
         OPTIONAL { ?b :q ?a }
         FILTER (!bound(?b)) }
```

Solution:

SQL sur $Triples(subject, prop, object)$:

```
select t1.sujet
from Triples t1 outer join Triples t2
     on (t1.objet=t2.sujet and t1.sujet = t2.objet)
where t1.prop=':p'
  and t2.prop=':q'
  and t1.objet = NULL and t2.sujet = NULL;
```

SQL sur $Proptable(subject, p)$:

```
select t1.sujet, t2.q
from Proptable t1 left outer join Proptable t2
     on (t1.p = t2.sujet and t1.sujet = t2.q)
where t1.p = NULL and t2.sujet = NULL
```

La réponse est toujours vide.

4.
```
prefix : <http://example.org>
select ?a ?p ?b
 where { { ?a ?p ?b }
           MINUS
         { ?b ?p ?a } }
```

Solution:

SQL sur $Triples(subject, prop, object)$:

```
select *
  from Triples t1
 where (t1.objet, t1.prop, t1.sujet) not in (select *
                                               from Proptable)
```

SQL sur $Proptable(subject, p)$:

```
select *
  from Proptable t1
 where (t1.p,t1.sujet) not in (select t2.sujet, t2.p
                                 from Proptable t2)
   and (t1.q,t1.sujet) not in (select t2.sujet, t2.q
                                 from Proptable t2)
```