Trabalho sobre Web Semântica (valendo 3,0 da nota da A2).

Professor: Mark Douglas Jacyntho

Trabalho **em grupo de no máximo 4 alunos**.

Enviar o **código fonte** para o e-mail **markjacyntho@gmail.com**

O código fonte tem que ser enviado em um arquivo .zip nomeado da seguinte forma:

<nome completo do(s) aluno(s)>.zip, onde

Exemplo: FulanoDasCoves_CiclanoDaSilva.zip

Semantic Crawler

(Integração de dados coletados da Web de Dados)

Fundamentação teórica:

Navegar pela Web é uma tarefa comum para nós humanos. A página corrente que estamos vendo contém links para outras páginas e nós clicamos nestes links e navegamos para outra página, que pode conter outros links, e assim sucessivamente.

Agentes de software também podem navegar pela Web. Talvez o exemplo mais comum seja os Web *crawlers* dos sites de busca (p.ex. google). Na Web de Dados (*Web of Linked Data*), há uma grande quantidade de fontes de dados (*datasets*), bem com um número ainda maior de links entre estas fontes de dados. Um link nada mais é do que a propriedade de uma tripla onde o sujeito reside em uma fonte de dados e o objeto reside em outra fonte de dados. Enfim, um RDF-link é definido pela propriedade da tripla (sujeito-propriedade-objeto). Neste contexto, um agente de software pode descobrir novos dados seguindo estes RDF-links e navegando de uma fonte de dados para outra, e depois para outra, e assim por diante.

Por exemplo, suponha que nós queiramos saber mais sobre o tenista *Roger Federer*. Podemos dereferenciar a URI `http://dbpedia.org/resource/Roger_Federer` e parte do documento RDF retornado é mostrado abaixo:

```
1: <rdf:Description
1a:     rdf:about="http://dbpedia.org/resource/Roger_Federer">
2:   <owl:sameAs rdf:resource= "http://rdf.freebase.com/ns/
2a:                         guid.9202a8c04000641f800000000019f525"/>
3: </rdf:Description>
4: <rdf:Description
4a:     rdf:about="http://mpii.de/yago/resource/Roger_Federer">
5:    <owl:sameAs
5a:     rdf:resource="http://dbpedia.org/resource/Roger_Federer"/>
6: </rdf:Description>
```

Agora, usando o link `owl:sameAs` da linha 2, podemos dereferenciar o URI `http://rdf.freebase.com/ns/guid.9202a8c04000641f800000000019f525`, que nos leva para outro documento RDF que contém mais fatos (triplas) sobre *Roger Federer*. Uma vez feito o download deste outro documento RDF, podemos repetir o mesmo procedimento e descobrir novos dados sobre ele, e assim sucessivamente.

O mesmo vale para o link da linha 4. Em geral, temos mais de um link para seguir. Para seguir todos os links temos que escolher entre duas estratégias: busca em profundidade (*depth-first search*) e busca em largura (*breadth-first search*).

Na busca em profundidade, primeiramente nós ignoramos o segundo link da linha 4 e seguimos apenas o link da linha 2 para obter um novo documento RDF. Além disso, nós seguiremos qualquer outro link encontrado neste novo documento para obter outro novo documento, e assim sucessivamente, aprofundando cada vez mais. Continuaremos aumentando a profundidade da busca até não haver mais link para navegar. Neste ponto, nós voltamos um nível atrás e seguimos outro link neste nível. Portanto, apenas quando não há mais links para seguir, a partir do link da linha 2, é que iremos para o link da linha 4, repetindo o mesmo processo de aprofundamento. Perceba que primeiro navegamos em profundidade antes de explorar um link no mesmo nível (mesmo documento).

Já na busca em largura, ocorre o oposto. Primeiro exploramos todos os links do mesmo nível (documento), para depois explorar o nível abaixo. Ou seja, depois de seguir o link da linha 2, mesmo achando novos links no novo documento, nós voltaremos para seguir o link da linha 4.

Enunciado:

Este trabalho consiste em criar um agente de software, usando Jena, que navega pela Web de Dados para coletar e integrar "tudo" que foi dito sobre um dado recurso, seguindo os links definidos pela propriedade **`owl:sameAs`**, de acordo com a estratégia **busca em profundidade**[1]. Para tal, o agende deve receber como parâmetros de entrada o *URI do recurso* e *um grafo RDF (um model do Jena)* que será preenchido com as triplas coletadas. Por exemplo, no caso do *Roger Federer*, poderíamos começar pela DBpedia passando como parâmetro o URI `http://dbpedia.org/resource/Roger_Federer`.

A partir do URI passado como parâmetro, o agente de software teria que executar três passos:

1. Dereferenciar o URI e obter um documento RDF que descreve o recurso definido pelo URI.

2. Coletar os fatos sobre o recurso, ou seja, coletar todas as triplas que satisfazem ao seguinte padrão e adicioná-las ao grafo passado como parâmetro:

   ```
   <URI> <algumaPropriedade> <algumValor> .
   ```

3. Para navegar e obter novos dados, encontrar todas triplas que satisfaçam os dois padrões a seguir, e o conjunto de *recursoSujeito* e *recursoObjeto* são os links a serem seguidos, ou seja, novos URIs a serem dereferenciados (voltar ao passo 1 para cada novo URI):

   ```
   <URI> owl:sameAs <recursoObject> .

   <recursoSujeito> owl:sameAs <URI> .
   ```

O primeiro passo dereferencia o URI obtendo o documento RDF asssociado.

O segundo passo obtém os dados. O agente coleta todas as triplas (fatos) sobre o recurso identificado pelo URI, ou seja, as triplas que têm o URI como sujeito.

O terceiro passo é a navegação. Especificamente neste trabalho, apenas a propriedade `owl:sameAs` está sendo usada para encontrar links. Como resultado, estes links são todos os URIs que representam o recurso em questão (por exemplo, *Roger Federer*). Perceba que o URI poder ser tanto o sujeito quanto o objeto de uma tripla contendo `owl:sameAs`; se for o sujeito, o URI do objeto é o link a ser seguido, e se for o objeto, o URI do sujeito é o link a ser seguido. Toda vez que um link é seguido, o novo URI em questão deve ser dereferenciado, repetindo todos três passos com este novo URI. Isto se repete até que não haja mais links a serem seguidos (dica: use recursividade).

Importante: Na Web de Dados (*Web of Linked Data*) é muito comum a existência de links bidirecionais, ou seja, um *dataset* A tem um link para o *dataset* B, e B tem um link de volta para A. Como resultado, se coletarmos

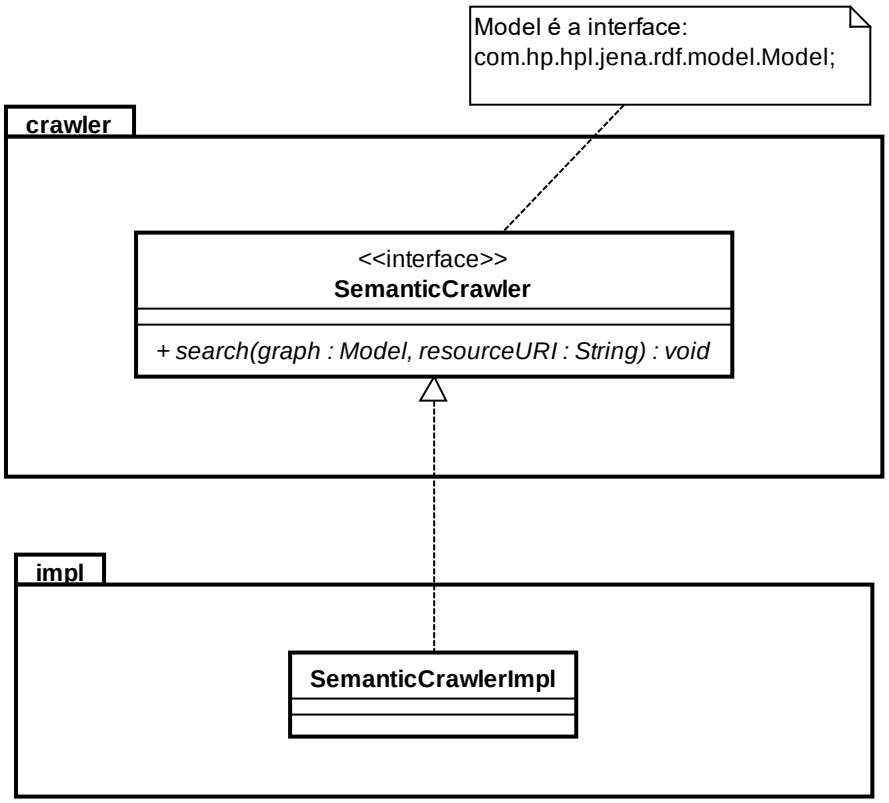
---

[1] Pseudocódigo recursivo de "busca em profundidade" em grafos pode ser encontrado em http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/Busca/busca.html#Prof (vide procedimento *`Busca-Prof(v: vértice)`*)

todos os URIs indiscriminadamente, teremos um loop infinito. Em outras palavras, é necessário armazenar os URIs que já foram visitados. Cada URI somente pode ser visitado (dereferenciado) apenas uma única vez.

Recursos anônimos (nós em branco): No passo 2, caso o valor (objeto) da tripla seja um nó em branco B_1, adicione também todas as triplas que tenham B_1 como sujeito. Se houver uma tripla que tenha B_1 como sujeito e outro nó em branco B_2 como objeto, repita os mesmos passos para o nó em branco B_2, adicionando todas as triplas que tenham B_2 como sujeito, e assim sucessivamente, até quando não houver mais nós em branco. Já no passo 3, se `<recursoObjeto>` ou `<recursoSujeito>` forem um nó em branco, também devem ser considerados, ou seja, tem que executar o passo 2 e adicionar todas triplas que tenham o referido nó em branco como sujeito. Perceba que não volta ao passo 1, pois não faz sentido dereferenciar um nó em branco, uma vez que a informação do nó em branco já esta presente no último arquivo RDF obtido.

O trabalho consiste em codificar o modelo abaixo:



Temos dois pacotes. O primeiro pacote (`crawler`) contém apenas a interface `SemanticCrawler`. Já o segundo pacote (`impl`) contém apenas a classe concreta `SemanticCrawlerImpl` que implementa a interface `SemanticCrawler`, ou seja, implementa o médodo `search(graph:Model, resourceURI: String): void`. A implementação deste método é o ponto principal deste trabalho e consiste em inserir todas as triplas do **passo 2**, explicado anteriormente, no grafo (*model*) passado como parâmetro.

Não precisa enviar classe de teste, ou seja, o teste não vale nota. Mas, obviamente, é preciso testar a sua implementação de alguma forma, mesmo que não valha nota. Para fazer seus testes pode usar qualquer URI. Comece com URIs que contenham menos informações, como por exemplo `http://dbpedia.org/resource/Zico`, para o processamento ser mais rápido.

Regras importantes:

1. Código com erros de compilação receberá nota zero.
2. Caso seja identificada cópia, ambos quem copiou e quem forneceu cópia receberão nota zero.
3. Trabalhos entregues com atraso receberão nota zero.