Relatório do trabalho da disciplina de Processamento de Linguagens

Documentação do trabalho prático

Roberto Filipe Manso Barreto - 21123

Henrique Monteiro Cartucho - 21122

Licenciatura de Engenharia de Sistemas Informáticos

Novembro de 2021

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Índice

1. Introdução

1.1. Estrutura do documento

Este documento refere-se à descrição e documentação do trabalho prático realizado pelos alunos Roberto Filipe Manso Barreto e Henrique Monteiro Cartucho da turma de licenciatura de engenharia de sistemas informáticos. Este documento contém a descrição do pensamento e do desenvolvimento do problema e dificuldades do mesmo.

1.2. Resumo

Inicialmente foi proposto pelo professor da disciplina 3 enunciados dos quais seria necessário escolher um destes, foi então escolhido o enunciado C, com este enunciado foi iniciado um projeto cujo objetivo consiste em desenvolver um programa que utilizando expressões regulares seja capaz de interpretar um ficheiro XML e converter este em ficheiros HTML e LATEX. Este projeto tem como grande objetivo explorar expressões regulares e *lexer* utilizando a biblioteca *ply* na linguagem *python*.

2. Lexer

2.1.Tokens

Para desenvolver o *lexer* foi então criado o ficheiro *xml_lexer,* neste ficheiro foi criada a classe XMLex, nesta classe foram criados os tokens: CTAGS, OTAGS, TRASH, TEXT.

- CTAGS → *Close tags,* este token está encarregue de detetar e ler tags de fecho (ex: *</def>*)
- OTAGS → *Open tags*, este token está encarregue de detetar e ler tags de abertura (ex: *<def>*)
- TRASH → *Trash* (Lixo), este token está encarregue de detetar tudo que não seja de interesse para o programa (ex: *<usg type="style">*)
- TEXT → Text (texto), este token está encarregue de detetar e ler todo o texto que existe entre tag de abertura e tag de fecho (ex: *<orth>*Achafundar*</orth>*)

Para ignorar *new lines* que não são de interesse para o programa foi utilizado o token *t_ignore e atribuído o caracter "\n",* ignorando assim todos os caracteres *"\n".*

2.2.Funções de Reconhecimento

Para aplicar as expressões regulares ao *tokens* criados e realizar operações com os *tokens* lidos foram criadas funções para cada *token* criado. A ordem como estas funções são apresentadas neste documento não é a ordem pela qual as funções estão no código do projeto, estas estão organizadas de forma diferente de forma que a solução desenvolvida seja de simples interpretação nesta documentação.

2.2.1. Função de abertura de tags

Para a função de abertura de tags foi criada a expressão regular "*<[^ >]+(>?)(\n?)*":

- "<" → reconhecer o início de qualquer *tag* de fecho, estas *tags* de fecho sempre iniciam com "*<*" (ex: "*<def>*").
- "*[^ >]+*" → reconhecer qualquer caracter, existindo sempre pelo menos 1 caracter ("+"), até se encontrar o caracter "*>*" ou o caracter " "(espaço). O caracter ">" indica sempre a finalização da escrita do nome da *tag* (ex: "*<def>*"), o caracter " "(espaço) é utilizado para quando existe alguma informação extra na *tag*, esta informação não seja reconhecida pois não é importante para o bom funcionamento do projeto, geralmente esta informação encontra-se separada no dome da tag por " "(espaço).
- "(>?)(\n?") → reconhecer o caracter final da *tag* (">") e *new line* ("\n") se estes existirem ("?").

Esta função após reconhecer a abertura de *tag* inicialmente é executado o método strip para remover quaisquer espaços brancos desnecessários inclusive *new line*, após isso são removidos os caracteres "<" e ">", conseguindo então obter apenas o nome da tag.

Com isto é possível então reconhecer as tags de abertura do documento XML. Para descobrir se o conteúdo de cada tag reconhecida é importante, foi decidido criar um switcher para as tags, mas visto que a linguagem python não dispõe de um switcher foi necessário utilizar outros métodos para desenvolver este switcher, criando assim o ficheiro tag_switcher.

Para criar o switcher foi criada a classe TagSwitcher e utilizado um dicionário no qual as suas chaves são o nome das tags que foram identificadas na leitura dos documentos como importantes e os seus valores são apontadores para funções que correspondem a cada tag. Nesta classe foi também criada a variável currentTag que guarda o nome da tag atual, para guardar este nome de tag é verificado ao ler uma tag de abertura se esta existe nas keys do dicionário, pois caso exista, isto indica que esta tag é importante, após a verificação é atribuído á variável currentTag o nome da tag reconhecida. É importante guardar o nome da tag de abertura pois com esta informação é possível moldar o documento gerado de forma diferente para cada tag diferente. Para realizar o switch das tags e executar a função necessária foi criado o método switch, este método obtém do dicionário a função correspondente à tag atual e executa esta função.

2.2.2. Função de fecho de tags

Para a função de fecho de *tags* foi criada a expressão regular "*</[^>]+>(\n?)*":

- "</" → reconhecer o início de qualquer *tag* de fecho, estas *tags* de fecho sempre iniciam com "*</*" (ex: "*</def>*")
- "*[^>]+*" → reconhecer qualquer caracter existindo sempre pelo menos 1 caracter("+") até se encontrar o caracter ">", este caracter indica sempre que o nome da *tag* acabou de ser escrito (ex: "*</def>*")
- ">(\n?)" → reconhecer o caracter final da *tag* (">") e se existir ("?"), reconhecer também o caracter "\n"

Esta função após reconhecer um fecho de *tag* inicialmente é executado o método strip para remover quaisquer espaços brancos desnecessários inclusive *new line*, após isso são removidos os caracteres "<" e ">", conseguindo então obter apenas o nome da tag que está a realizar o fecho. Após isto é verificado se a tag de fecho está contida nas chaves do TagSwitcher, pois existem definições de palavras que contém outras tags dentro da tag de definição, como por exemplo a tag "quote", com isso foi necessário criar uma variável chamada currentdef que é uma string à qual é acrescentado o texto da definição parte por parte conseguindo assim juntar o texto de cada tag dentro das definições de palavras em uma só string. Visto isto é necessário guardar as definições caso existam várias definições então foi acrescentado a função write_nl_def que acrescenta um new line à definição atual quando a tag "\def" é detetada, mas para detetar quando as definições terminam de vez é utilizada a tag de palavra("orth"), quando esta tag é reconhecida a função que esta executa termina a definição da palavra anterior pois não irá existir mais definições desta palavra.

2.2.3. Função de reconhecimento de lixo

Para a função reconhecimento de lixo foi inicialmente criada a expressão regular "*[^><]+>*":

- "*[^><]+*" → reconhecer qualquer caracter existindo sempre pelo menos 1 caracter até se encontrar o caracter ">" ou "<", estes caracteres indicam sempre que o nome da *tag* acabou de ser escrito ou então iniciado (ex: *<usg type="dom">*)
- ">" → reconhecer o caracter final da tag para assim este não ter de ser removido de outros tokens de forma lógica.

Este token permitiu não necessitar de logica para ser removido qualquer tipo de lixo encontrado entre o final do nome da tag e o caracter ">", conseguindo assim uma melhor performance do programa.

2.2.4. Função de reconhecimento de texto

Para a função de reconhecimento de texto foi criada a expressão regular "[^<]+":

- "[^<]+" → reconhecer qualquer caracter existindo obrigatoriamente pelo menos 1 caracter até que se encontre o caracter "<", pois o caracter "<" indica o inicio de uma tag seja esta de abertura ou de fecho, sendo assim o texto seria encontrado até ao seguinte caracter "<".

Com esta função foi então possível obter todo o texto necessário de cada tag. Para este texto ser então utilizado é chamada a função switch do TagSwitcher que neste momento começou a receber por parâmetro o texto a ser escrito, quando a função obtida ao consultar a chave que contém a tag atual, a sua função correspondente é obtida e executada colocando nos seus parâmetros o texto reconhecido sendo assim este é direcionado para a tag de importância.Caso a tag atual seja a tag de fecho de uma definição, a função referente a esta não recebe qualquer valor por parâmetro então caso a tag atual seja a tag de fecho de definição não é colocado nenhum valor nos seus parâmetros.

2.3.Gerar ficheiros HTML e LATEX

Para gerar ficheiros HTML e LATEX foi necessário criar o ficheiro files_generator, neste ficheiro encontra-se a classe FilesGenerator, esta classe no seu construtor recebe por parâmetro a letra atual do dicionário e atribui o seu valor à variável header, de seguida verifica se o caminho desejado para colocar os ficheiros gerados existe, caso este não exista é criado, após a verificação são abertos os ficheiros html e latex no modo write para caso estes ficheiros não existam serem criados e caso existam todo o seu conteúdo ser reescrito, estes ficheiros são gerados com o nome do ficheiro ser a letra atual de leitura.

2.3.1. Gerar ficheiros HTML

A linguagem de escrita HTML tem um padrão igual a XML, funcionando esta também com tags, sendo então de fácil aprendizagem. Foi então decidido que o título da página seria a letra atual de leitura e este receberia a tag "*h1*", a definição da palavra seria um paragrafo recebendo então a tag "*p*" e toda a restante informação seria organizada por tópicos recebendo então a tag "*li*".

Um ficheiro HTML começa sempre com a tag *hmtl* seguida da tag *body* escrevendo então todo o ficheiro dentro da tag *body*. Com a estrutura do documento decidida foram desenvolvidas funções para cada tag XML que seria necessário ser escrita no ficheiro HTML, estas funções recebem por parâmetro a string a ser escrita e é executada no ficheiro html criado a função write que vai acrescentar texto a este ficheiro, esta função recebe por parâmetro a string a ser acrescentada ao ficheiro, para cada tipo de tag XML é então no inicio da string aberta a correspondente tag HTML, de seguida é colocado com a ajuda da formatação de strings o valor recebido na função e por fim fechada a tag HTML. Estas funções são então executadas e todo o ficheiro é escrito, por fim este ficheiro é terminado, fechando as tags body e html e fechado ficando assim este guardado.

2.3.2. Gerar ficheiros LATEX

Para gerar ficheiros LATEX é utilizado um formato específico para latex, para isto foi necessário consultar documentação disponibilizada pelo website overleaf. Para decidir a estrutura do documento foi decidido que a letra de leitura seria associada ao "\title", cada palavra seria "\section" e os restantes seriam "\item" que seriam iniciados com "\begin{itemize}" e terminados com "\end{itemize}". Utilizando as funções para escrever os ficheiros html, é escrito os ficheiros LATEX ao memso tempo, mas existiram alguns contratempos como por exemplo não é possível escrever "_" diretamente, sendo necessário criar uma função que substituí os "_" por "\textunderscore". Para escrever os valores recebidos é utilizado a formatação de strings e a concatenação de strings conseguindo assim utilizar o caracter "{"sem ativar a formatação de strings. Por fim para terminar o ficheiro é escrito no ficheiro "\end{document}" e é fechado o ficheiro ficando então este guardado.

Para receber então os valores na classe FilesGenerator esta é instanciada na classe TagSwitcher e as funções da classe FilesGenerator são executadas assim que as funções da classe TagSwitcher são executadas, assegurando assim que cada função é executada para a tag correta.

2.4.Função initialize

Para obter o nome de todos os ficheiros necessários para ler foi importado da biblioteca “OS” a função walk que percorre todos os ficheiros em um caminho e devolve o nome de cada um, este nome é guardado em um array e é passado por parâmetro para a classe lexer ao instanciar a mesma. Quando a função initialize é executada então, o lexer é instanciado e um loop é realizado com o array de nomes de ficheiros, este loop a cada iteração primeiramente cria um novo ficheiro na classe TagSwitcher passando por parâmetro apenas o primeiro caracter do nome do ficheiro utilizando slicing para assim este ser criado com o header já colocado, de seguida o ficheiro é lido e o seu conteúdo é guardado em uma variável que posteriormente é passada por parâmetro para o input do lexer, após isso é executada a função de terminar o ficheiro do TagSwitcher e caso ainda exista mais ficheiros o loop é repetido.

3. Bibliografia

- Documentation - Overleaf, Online LaTeX Editor | 20-11-2021
  - https://www.overleaf.com/learn