Processamento de Linguagens

Trabalho Prático nº2 (Flex)
XML to Dot

Relatório de Desenvolvimento

Cesário Perneta	João Gomes	Tiago Fraga
a73883	a74033	a74092

May 3, 2018

Abstract

Neste relatório será abordado a resolução e verificação do segundo trabalho prático da unidade curricular de Processamento de Linguagens. Será possível encontrar a forma como abordamos o problema proposto bem como uma explicação detalhada do código elaborado.

Contents

Chapter 1

XML to Dot

O nosso grupo teve como enunciado o *XML to Dot*, neste enunciado foi nos proposto que fizessemos uma análise aos ficheiros *.xml* fornecidos. Depois de efetuada esta análise, foi nos pedido que desenvolvessemos um programa em *Flex* de modo a gerar um grafo de dependências entre os elementos de um documento anotado em *XML*. Para gerar o Grafo tivemos de escrever linhas do tipo *Dot-graphviz*, criando assim um ficheiro com a extensão *.dot*.

1.1 Estrutura de Dados

Começamos por definir uma estrutura de dados capaz de guardar todas as *tags* do ficheiro *XML* que continham dependências de modo a construir o grafo corretamente.

```
typedef struct doc {
    Lista* tag_principal;
}Document;
```

Esta é a estrutura capaz de guardar o apontador para o inicio da lista de *tags* principais, importante para a criação do grafo de dependências. Esta estrutura possui só um array de apontadores.

```
typedef struct lista{
    char* principal;
    struct array* filho;
    struct lista* next;
}Lista;
```

Esta é a estrutura principal da nossa implementação, contêm a *tag* principal, ou seja, guarda a *tag* que possui sub-elementos, contêm ainda uma estrutura auxiliar explicada a seguir e o apontador para a próxima posição da lista.

```
typedef struct array{
    char* tag;
    struct array* next;
}Array;
```

Esta estrutura contêm todos os sub-elementos relativos a *tag* principal em causa, ou seja, tem o nome da *tag* e o apontador para a próxima posição da tabela.

```
// cria uma nova estrutura capaz de armazenar uma tag secundária.
Array* newSecundaryTag(char* nome);

//cria uma nova estrutura capaz de armazenar uma tag principal.
Lista* newPrincipalNode(char* nome);

//inicia a estrutura com os apontadores para a Lista de tags principais.
Document* initDoc();

//adiciona uma nova tag principal.
int adicionaPrincipalNode(Document* doc,char* nome);

//adiciona uma nova tag secundária.
int adicionaSecudaryNode(Document* doc, char* principal, char* nome);

//verifica se a tag principal já está armazenada.
int verificaPrincipal(Document* doc, char* nome);

//verifica se a tag secundária já está armazenada.
int verificaSecundario(Document* doc, char* principal, char* nome);
```

Apresentamos aqui as funções importantes para manipular as estruturas criadas.

```
int imprimeLista(Document* doc){
    int fd = open("graph.dot",O_CREAT|O_WRONLY,0666);
    write(fd,"digraph tp2{",strlen("digraph tp2{"));
    write(fd,"\n",1);
    Lista* lista = doc->tag_principal;
    while(lista){
        Array* array = lista->filho;
        while(array){
            write(fd,lista->principal,strlen(lista->principal));
            write(fd,"->",strlen("->"));
            write(fd,array->tag,strlen(array->tag));
            write(fd,"\n",1);
            array = array->next;
        }
        lista = lista->next;
    }
    write(fd,"}",1);
    close(fd);
    return 0;
}
```

Salientamos esta função,visto ser, a função que trata da criação e da escrita no ficheiro *.dot*. Esta trata de percorrer a estrutura depois de criada de modo a imprimir de acordo com as regras de um ficheiro *.dot* e criando assim o grafo a apresentar.

1.2 Filtagem da Informação

Neste capítulo apresentamos como filtramos o texto de modo a guardar a informação pretendida nas estruturas criadas. Vamos mostrar e explicar as codinções de contexto e expressões regulares utilizadas.

```
%x TAG FECHO_TAG TAG_QUANDO
```

Esta instrução contêm o nome das condições de contexto utilizadas, importante para o funcionamento das mesmas.

```
<INITIAL>{
[a-zA-Z0-9]+          {;}
\<[?].*[?]\>          {;}
\<[!].*\>             {;}
\<quando.*\/          {BEGIN TAG_QUANDO;}
\<obs\/\>             {;}
\<                    {BEGIN TAG;}
\<\/                  {BEGIN FECHO_TAG;}
}
```

Esta condição de contexto serve para filtrar tudo o que não pretendemos apanhar e iniciar as condições de contexto em que guardamos as *tags* principais e secundárias. Em seguida temos a explicação de cada expressão regular.

```
[a-zA-Z0-9]+   {;}
```

Com esta expressão regular conseguimos eliminar o conteúdo dentro de cada *tag*, por exemplo

<NIF>987653210</NIF >, neste caso eliminamos **987653210**.

```
\<[?].*[?]\>   {;}
```

Com esta expressão regular eliminamos todas as *tags* que contenham o caracter , ou seja são removidas todas as instruções de processamento, visto não interessarem para o nosso grafo.

```
\<[!].*\>      {;}
```

Por último eliminamos todos os comentários que possam surgir no ficheiro visto também não terem interesse.

```
\<quando.*\/   {BEGIN TAG_QUANDO;}
```

Quando apanhamos uma *tag* começada por **quando** damos inicio a condição de contexto evidenciada **TAG_QUANDO** e explicada posteriormente.

```
\<obs\/\>      {;}
```

Esta expressão regular é referente ao ficheiro **processos.xml** visto quando a *tag* **obs**, quando não possui qualquer conteúdo apresentar esta representação **<obs>** e então decidimos remover para evitar situações de erro na criação do Grafo.

```
\<   {BEGIN TAG;}
```

Quando é apanhado um sinal de < começamos a condição de contexto **TAG** explicada posteriormente.

```
\<\/    {BEGIN FECHO_TAG;}
```

Por último neste bloco inicial, iniciamos a condição de contexto **FECHO_TAG**, quando apanhamos uma *tag* de fecho do ficheiro **XML**.

```
<TAG>{
[a-zA-Z0-9]+                    {tag = strdup(yytext);}
[a-zA-Z0-9]+?[_][a-zA-Z0-9]+    {tag = strdup(yytext);}
[ ].*[=].*\>{

    hierarquia[nivel] = strdup(tag);
    nivel++;
    adicionaPrincipalNode(doc,tag);
    BEGIN INITIAL;
}
\>{
    if(strcmp(tag,"obs")==0 || strcmp(tag,"pai")==0 || strcmp(tag,"mae")==0  ){
    BEGIN INITIAL;
    }else{
        hierarquia[nivel] = strdup(tag);
        nivel++;
        adicionaPrincipalNode(doc,tag);
        BEGIN INITIAL;
    }
}
\>.*\<\/.*\>{
        adicionaSecudaryNode(doc,hierarquia[nivel-1],tag);
        BEGIN INITIAL;
    }
}
```

Depois do bloco inicial, temos a condição de contexto **TAG**, vai ser neste bloco que vamos começar a adicionar as *tags* a estrutura criada.

```
[a-zA-Z0-9]+                    {tag = strdup(yytext);}
[a-zA-Z0-9]+?[_][a-zA-Z0-9]+    {tag = strdup(yytext);}
```

Estas expressões regulares são para apanhar os nomes das *tags* e por sua vez guardar esse nome numa vaŕiável.

```
[ ].*[=].*\> {
```

```
    hierarquia[nivel] = strdup(tag);
    nivel++;
    adicionaPrincipalNode(doc,tag);
    BEGIN INITIAL;
}
```

Sabendo a partida que as *tags* com atributos são *tags* principais guardamos na estrutura sempre que as apanhamos. Usando uma estrutura auxiliar chamada **hierarquia** para saber em que nível se encontra de modo a gerarmos as dependências corretamente.

```
\> {
        if(strcmp(tag,"obs")==0 || strcmp(tag,"pai")==0 || strcmp(tag,"mae")==0  ){
        BEGIN INITIAL;
   }else{
        hierarquia[nivel] = strdup(tag);
        nivel++;
        adicionaPrincipalNode(doc,tag);
        BEGIN INITIAL;
    }
}
```

Aqui guardamos todas as *tags* principais que não possuem atributos, mantendo novamente a informação quanto a sua posição na hierarquia.

```
\>.*\<\/.*\> {
        adicionaSecudaryNode(doc,hierarquia[nivel-1],tag);
        BEGIN INITIAL;
    }
}
```

Nesta expressão regular captamos todas as *tags* secundárias, relativas a *tag* principal captada anteriormente e guardada no *array* hierarquia no nível anterior ao que estamos colocados.

```
<FECHO_TAG>{
[a-zA-Z0-9]+                    {tag = strdup(yytext);}
[a-zA-Z0-9]+?[_][a-zA-Z0-9]+    {tag = strdup(yytext);}
\>  {
        if(strcmp(tag,"obs")==0){
        BEGIN INITIAL;
        }else{
            nivel--;
            free(hierarquia[nivel]);
            if(nivel!=0){
                adicionaSecudaryNode(doc,hierarquia[nivel-1],tag);
            }
            BEGIN INITIAL;
        }
    }
}
```

Nesta condição de contexto fazemos o tratamento quanto ao fecho de *tags* do ficheiro.

```
\>  {
        if(strcmp(tag,"obs")==0){
        BEGIN INITIAL;
        }else{
            nivel--;
            free(hierarquia[nivel]);
            if(nivel!=0){
                adicionaSecudaryNode(doc,hierarquia[nivel-1],tag);
            }
            BEGIN INITIAL;
        }
    }
```

Quando uma *tag* fecha é adicionado como *tag* secundária a *tag* que está acima da mesma na hierarquia.

```
<TAG_QUANDO>{
\>              {
                adicionaSecudaryNode(doc,hierarquia[nivel-1],"quando");
                BEGIN INITIAL;
                }
}
```

Esta condição de contexto foi adicionada devido a *tag* **textless quando textgreater** presente no ficheiro **legenda.xml**.

```
int main(){
    doc = initDoc();
    tag = NULL;
    int i;
    for(i=0;i<20;i++){
        hierarquia[i] = NULL;
    }
    nivel=0;
    yylex();
    imprimeLista(doc);
    return 0;
}
```

Esta é a nossa função **main** que trata de iniciar a estrutura e o **array** hierarquia e imprimir para o ficheiro.

Chapter 2

Conclusão

Durante a realização deste trabalho prático, que foi o segundo desta unidade curricular, várias foram as etapas que tivemos de passar para chegar ao resultado final. Em comparaão com o trabalho anterior achamos que este foi mais complexo e trabalhoso, mas ao mesmo tempo foi bastante importante, visto que a sua elaboração permitiu melhorar a nossa capacidade de escrever Expressões Regulares (ER) e, a partir destas, desenvolver Processadores de Linguagem Regulares, tendo em vista a filtragem e transformação de textos. Para isto foi necessário aprender e desenvolver a nossa abilidade de gerar filtros de texto em **FLex** e assim desenvolver a solução necessária para resolver o enunciado escolhido. Infelizmente, ao contrário do trabalho anterior, não conseguimos resolver nenhum enunciado extra, devido a falta de tempo, para a resolução do mesmo. Concluindo, no geral estamos satisfeitos com o trabalho desenvolvido, tendo este se relevado bastante útil no aperfeiçoamento das nossas habilidades no que toca a gerar filtros de texto em **FLex**.