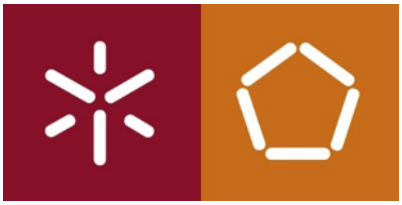


Universidade do Minho

Gramáticas na Compreensão de Software (4º ano de Curso)

Trabalho Prático

Relatório de Desenvolvimento

Mestrado Integrado em Engenharia Informática

Realizado por
GRUPO 1
João da Cunha e Costa, a84775
Luís Miguel Arieira Ramos, a83930

7 de janeiro de 2021

Resumo

Este relatório vai abordar todo o processo da realização do trabalho prático no âmbito unidade curricular de Gramáticas na Compreensão de Software do 4º ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho. Este trabalho aborda principalmente a criação de uma DSL para apoio ao professor, que permita escolher um recurso de aprendizagem adequado a determinado aluno para um determinado conceito.

Conteúdo

1 Introdução

1.1 Enquadramento e Contextualização

Este trabalho surgiu no âmbito da unidade curricular de Gramáticas na Compreensão de Software, do 4º ano do Mestrado Integrado em Engenharia Informática. Este trabalho aborda principalmente a criação de uma DSL para apoio ao professor, que permita escolher um recurso de aprendizagem adequado a determinado aluno para um determinado conceito. Para a resolução deste trabalho recorreu-se ao uso de expressões regulares estudadas nas aulas práticas, bem como a criação de uma gramática de atributos (GA), capaz de receber o ficheiro de input com os respetivos dados, aplicar o algoritmo necessário e indicar ao utilizador quais as melhores opções.

1.2 Problema e Objetivo

Os principais objetivos deste trabalho prático são logo dados no início do documento fornecido aos alunos. Por nossas palavras, os principais pontos expostos são os seguintes:

- Maior experiência com o programa ANTLR bem como a programação em Java;
- Praticar a escrita de GA, isto é, gramáticas de atributos, de acordo com o domínio do problema;
- Pôr em prática o uso expressões regulares (ERs), que servem para delinear partes de um certo documento que seguem um específico padrão;

O problema específico descrito neste enunciado cumpre todos estes objetivos.

1.3 Resultados ou Contributos

Quanto ao resultado deste trabalho, o grupo revelou-se satisfeito com o mesmo. O nosso programa é capaz de ler o ficheiro de input dado pelo utilizador, interpretar qual foi a query feita e aplicar o respetivo algoritmo de modo a obter a resposta para essa mesma query.

Conseguimos generalizar o nosso código de modo a ser permitido ao utilizador definir quais os valores dos fatores de decisão que prentende usar. Caso não seja dado nenhum valor, será usado um valor por omissão. Tudo isto será explicado em maior promenor no resto do relatório.

1.4 Estrutura do Relatório

Iremos explorar cada tópico mais aprofundadamente no resto deste relatório:

- Análise e Especificação: abordamos o problema do enunciado de maneira mais informal e específica, enunciando os objetivos a cumprir e explicamos os dados, pedidos e relações do programa;
- Conceção/desenho da Resolução: explicamos em maior detalhe as estruturas de dados e os algoritmos usados na nossa solução;
- Codificação e Testes: apresentamos os nossos resultados e testes que realizamos para garantir a qualidade do nosso código;
- Conclusão: avaliação final do nosso trabalho.

2 Análise e Especificação

2.1 Descrição informal do problema

Depois de explicarmos formalmente o que o problema pede na introdução, vamos agora aprofundar o nosso raciocínio imediato do que era para fazer depois de lermos o enunciado.

Ao observarmos o problema dado, definimos as seguintes tarefas:

- Definir o formato do ficheiro que irá conter a informação dos alunos, recursos de aprendizagem e as respetivas querys;
- Criar uma gramática de atributos capaz de receber e interpretar o ficheiro de input, ou seja, todos os recursos de aprendizem e todos os alunos registados;
- Definir que estruturas de dados usar de modo a tornar o programa o mais eficiente e simples possível;
- Criar um algoritmo capaz de resolver a prosposta descrita no enunciado, ou seja, um algoritmo que seja capaz de identifcar quais os recursos mais apropriados para um determinado aluno que esteja disposto a aprender um determinado conceito;

2.2 Especificação do Requisitos

2.2.1 Dados

Definido o formato do ficheiro de input que irá conter os dados, detetamos algumas caracteristicas no texto que teriamos de interpretar:

```
AL numeroAluno , idade , caracteristicas
AL A84900 , 18 , [raivoso , lento]
```

- AL: tag que indica que toda a informação seguinte estará relacionada com um aluno. Todos os campos estão separados por uma virgula, sendo assim mais fácil de interpretar.
- numeroAluno: campo que contêm o número de aluno. De notar que este campo só aceita números de alunos válidos, ou seja, que respeitem o respetivo formato $AXXXXX$ ou $PGXXXXX$.
- idade: campo que contêm a idade do aluno.
- caracteristicas: campo que contêm uma lista de caracteristicas referentes ao aluno. De notar que é obrigatório a lista conter pelo menos um elemento.

```
RA id , tipo , titulo , intervaloIdade , conceitos , caracteristicas
RA 1 , Livro, "Data Scientist" , (18/35) , [Data Science, SQL, pandas,
   Python] , [lento, raivoso, mediano]
```

- RA: tag que indica que toda a informação seguinte estará relacionada com um recurso de aprendizagem. Todos os campos estão separados por uma virgula, sendo assim mais fácil de interpretar.
- tipo: campo que contêm o tipo de recurso.
- titulo: campo que contêm o título do recurso em questão. De referir que o título inicia e termina com o Caractere ".

- intervaloIdade: campo que contêm o intervalo de idades apropriado.
- conceitos: campo que contêm uma lista de conceitos a que o determinado recurso se aplica. De notar que é obrigatório a lista conter pelo menos um elemento
- caracteristicas: campo que contêm uma lista de características referentes ao recurso. De notar que é obrigatório a lista conter pelo menos um elemento.

2.2.2 Pedidos

O principal pedido está bem explícito no enunciado deste problema. Dado um número de aluno e um conceito, o programa ser capaz de indicar, dos recursos disponíveis, quais são os três ideais para o respetivo aluno. De modo a tornar o programa mais interativo, o grupo adicionou um campo na query, opcional, em que o utilizador poderá escolher o número de recursos ideais a serem apresentados.

```
QUERY numeroAluno conceito numTop
QUERY A84900 JavaScript 5
```

- QUERY: tag que indica que toda a informação seguinte estará relacionada com um pedido do utilizador.
- numeroAluno: campo que contêm o número do aluno.
- conceito: campo que contêm o conceito que o aluno irá aprender.
- numTop: campo que contêm o número de opções que serão apresentadas.

De notar que o campo **numTop** é opcional. Quando este não existe, o programa assume o número 3, como valor *default*.

Como forma de tornar o programa ainda mais interativo, e mais realista possível, o grupo adicionou também uma linha, opcional, que permite ao utilizador escolher os valores dos fatores de decisão. De notar que caso o utilizador opte por não definir os seus próprios valores, serão usados valores *default* estudados e definidos pelo grupo.

```
FATORES fatorCaracteristicas fatorIdade
FATORES 1.7 0.4
```

- FATORES: tag que indica que toda a informação seguinte estará relacionada com os valores dos fatores de decisão.
- fatorCaracteristicas: campo que contêm o valor do fator das caracteristicas.
- fatorIdade: campo que contêm o valor do fator da idade.

De modo a tornar o progama o mais eficiente e rápido possível, o grupo optou por inserir o pedido do utilizador na primeira linha do respetivo ficheiro de input, dando assim possibilidade de correr o algoritmo sem ter de guardar toda a informação que o ficheiro contêm.

3 Gramática

Com recurso à ferramenta ANTLR, iniciou-se a escrita de uma gramática de atributos capaz de interpretar o ficheiro de input. Começamos por definir o formato geral do ficheiro.

```
basedados  : fatores?
             query
             aluno+
             recursoAprendizagem+
           ;
```

A primeira linha poderá conter **fatores**, ou não. De seguida recebe uma **query**. Depois disso, o programa recebe no mínimo um **aluno**. Por fim, recebe no mínimo um re**curso de aprendizagem**.

```
fatores : FATORES fatorC fatorI
        ;
```

Contêm a tag **FATORES** e os dois valores, respetivamente, fator características e fator idade.

```
query : QUERY numAluno conceito numTop?
      ;
```

Contêm a tag **QUERY**, número do aluno, o conceito e poderá conter, ou não, o número de recursos a serem mostrados ao utilizador.

```
aluno : AL numAluno VIRGULA idade VIRGULA caracteristicas
      ;
```

Contêm a tag **AL**, número do aluno, idade e as características.

```
recursoAprendizagem : RA id  VIRGULA tipo
                      VIRGULA titulo VIRGULA idadeIntervalo
                      VIRGULA conceitos VIRGULA caracteristicas
                    ;
```

Contêm a tag **RA**, o id, o tipo, a descrição, o intervalo de idades, os conceitos e as características.

```
caracteristicas : LPARENTRETO
                  caracteristica (VIRGULA caracteristica)*
                  RPARENTRETO
                ;
```

Contêm a tag **LPARENTRETO**, uma ou mais características e a tag **RPARENTRETO**.

```
conceitos : LPARENTRETO
            conceito (VIRGULA conceito)*
            RPARENTRETO
          ;
```

Contêm a tag **LPARENTRETO**, um ou mais conceitos e a tag **RPARENTRETO**.

```
titulo : ASPA
         titulo_aux (VIRGULA titulo_aux)*
         ASPA
       ;
```

Contêm a tag **ASPA**, um ou mais titulos e a tag **ASPA**.

```
descricoes : LPARENT
             NUM BARRA NUM
             RPARENT
           ;
```

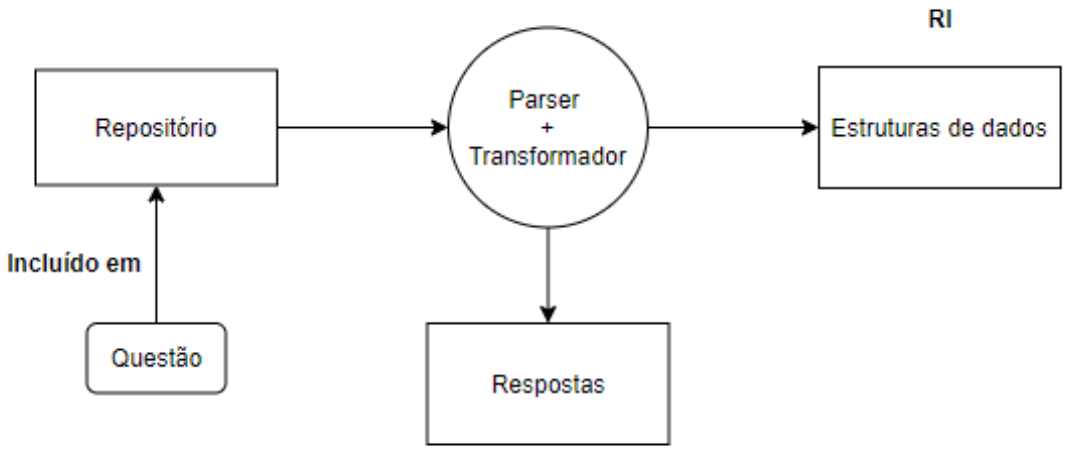
Contêm a tag **LPARENT**, a tag **NUM**, a tag **BARRA**, a tag **NUM** e a tag **RPARENT**.

```

idade : NUM
      ;

id : NUM
   ;

numTop : NUM
       ;

numAluno : NUMALUNO
         ;

fatorC : DOUBLE
       ;

fatorI : DOUBLE
       ;
```

```
//LEXER


RA: [rR][aA]
  ;

FATORES: [fF][aA][tT][oO][rR][eE][sS]
       ;

QUERY: [qQ][uU][eE][rR][yY]
     ;

AL: [aA][lL]
  ;

NUM: [1-9][0-9]*
   ;

DOUBLE: [0-9]+([\.][0-9]+)?
      ;

NUMALUNO:([aA]|[pP][gG])[1-9][0-9][0-9][0-9][0-9]
        ;

TEXTO: [a-zA-Z0-9\-\&\#\.\:]+
     ;


VIRGULA:       ',';
PONTO:         '.';
PONTOVIRGULA: ';';
LPARENT:       '(';
RPARENT:       ')';
LPARENTRETO:   '[';
RPARENTRETO:   ']';
ASPA:          '"';
BARRA:         '/';


WS: ('\r'? '\n' | ' ' | '\t')+ ->skip;
```

4 Conceção/desenho da Resolução

4.1 Esquema



Como o esquema demonstra, a aplicação envia para o parser+transformador o repositório com toda a informação dos alunos e recursos, e ainda inclui a questão feita pelo utilizador. O parser+transformador recorre às estruturas de danos para guardar a informação que necessita e no final do processo apresenta as respostas ao utilizador.

4.2 Estruturas de Dados

Em termos de estruturas de dados, usamos maioritariamente HashMaps, como suporte de memória para algumas das nossas funcionalidades:

- Começamos por criar duas variáveis `double fatorIdade` e `double fatorCaract` sendo que estes tomam como valor *default*, 0.5 e 1.5, respetivamente;
- Criou-se também uma variável do tipo string, denominada de `numAluno`, que irá conter o número do aluno em questão, sendo inicializada com valor *null*;
- Foram adicionados três inteiros, `idadeAluno`,`ntop` e `diferenca`, com o objetivo de guardar valores necessários na resolução do algoritmo;
- Para guardar características referentes ao aluno selecionado, foi criada uma lista `List<String> caracteristicasAluno`;
- Por fim, foram criados seis HashMaps, com objetivo de guardar toda a informação sobre os recursos escolhidos.

```
1  double fatorIdade      = 0.5;
2  double fatorCaract     = 1.5;
3  String numAluno        = null;
4  int idadeAluno         = 0;
5  int ntop               = 3;
6  int diferenca          = 0;
7  List<String>                    caracteristicasAluno = new ArrayList<>();
8  HashMap<Integer,String>         IdadeMap             = new HashMap<>();
9  HashMap<Integer,List<String>> TituloMap              = new HashMap<>();
10 HashMap<Integer,List<String>> ConceiMap              = new HashMap<>();
11 HashMap<Integer,List<String>> CaractMap              = new HashMap<>();
12 HashMap<Integer,String>         TipoMap              = new HashMap<>();
13 HashMap<Integer,Double>         top                  = new HashMap<>();
```

4.3 Algoritmos

O grupo definiu que o algoritmo de escolha seria feito através de um sistema de pontos que envolve as características e a idade, tanto do aluno como do recurso, e os conceitos que estão associados a cada recurso.

Em primeiro lugar, o algoritmo calcula a pontuação do recurso em análise através de uma fórmula definida pelo grupo.

São contadas todas as características do aluno que coincidem com as características do recurso e multiplica-se pelo seu factor :

```
pontuacao = númeroCaracteristicasCoincidentes * factorCaracteristicas
```

Em segundo lugar, caso a idade do aluno não pertença ao intervalo de idades do recurso, então o algoritmo verifica se a idade do aluno é maior que a idade maior do intervalo. Caso seja verdade, então calcula a diferença entre a idade do aluno e a idade máxima do intervalo. Caso contrário, fará a diferença entre a idade mínima e a idade do aluno. Depois de obtido o valor, este é multiplicado pelo seu factor e subtraído à pontuação geral:

```
pontuacao -= diferencaIdade * factorIdade
```

De modo a tornar a aplicação o mais realista possível e de modo a que se notasse mais diferença de idades, por exemplo, entre os 10 e 13 anos do que entre os 14 e 20 anos, o grupo definiu valores que serão multiplicados ao factorIdade, não podendo estes ser alterados pelo utilizador.

```
if idadeAluno <= 13  factorIdade * 1.2;
if 14 <= idadeAluno <=20 factorIdade;
if 21 <= idadeAluno <=50 factorIdade * 0.2;
if 51 <= idadeAluno factorIdade * 0.4;
```

Por último, calculada a pontuação do recurso, o algoritmo coloca no HashMap o id e a pontuação do respetivo recurso. Esse HashMap irá conter o os recursos mais apropriados ao aluno e terá o tamanho definido pelo utilizador.

Caso o HashMap já esteja totalmente preenchido, o algoritmo terá de decidir se o recurso é mais apropriado dos que os restantes recursos já guardados na estrutura.

Essa decisão é influenciada pelo conceito e pela pontuação. Para isso são separados, em dois TreeMaps, os recursos que contêm o conceito, daqueles que não tem.

```
TreeMap<String,Integer> topwithoutc = new TreeMap<String,Integer>();
TreeMap<String,Integer> topwithc    = new TreeMap<String,Integer>();
top.forEach((k,v)-> {
   if (ConceiMap.get(k).contains(query.conceit)) topwithc.put(pontuacao,id);
   else topwithoutc.put(pontuacao,id);
});
```

Depois é verificado se o recurso em questão contém, ou não, o conceito pedido.

Caso não contenha, então comparamos a pontuação do recurso com os recursos pertencentes ao TreeMap **topwithoutc**. Se a pontuação for maior, então o de menor pontuação será retirado do HashMap, e o recurso actual será adicionado.

Caso contenha, o algoritmo vai verificar o tamanho do TreeMap **topwithoutc**. Se não for vazio, então automaticamente o algoritmo dá prioridade ao recurso atual, sendo que o recurso do TreeMap **topwithoutc** com menor pontuação é retirado do HashMap, e o recurso atual é adicionado. Se for vazio, então compara-se a pontuação do recurso com os recursos pertencentes ao TreeMap **topwithc**. Se a pontuação for maior, então o de menor pontuação será retirado do HashMap, e o recurso actual será adicionado.

Terminado o algoritmo para todo os recursos, ficam no HashMap os ids dos recursos mais apropriados, sendo apenas necessário apresentar a resposta ao utlizador de forma simples e agradável.

5 Codificação e Testes

5.1 Alternativas, Decisões e Problemas de Implementação

A implementação inicial foi relativamente fácil e simples. Os problemas surgiram quando foi necessário ordenar a estrutura de dados que guardava os melhores recursos para determinada query. Inicialmente optamos por utilizar um HashMap para guardar estes dados, no entanto tivemos dificuldade em ordenar o mesmo de acordo com a pontuação do recurso. A solução para este problema foi a alteração da estrutura de dados para um TreeMap, o qual por padrão se encontra ordenado.

Provavelmente o aspeto mais relevante a ser definido pelo grupo foi como determinar se um recurso é mais adequado do que outro para dado aluno aprender certo conceito. Optamos portanto por definir um sistema de pontos, por cada característica que o aluno tivesse em comum com o recurso eram somados certos pontos e, dependendo da faixa etária do aluno e da diferença da idade do mesmo para a idade alvo do recurso, eram descontados tantos pontos quanto mais díspar era a diferença. O valor dos pontos somados ou descontados estão associados a fatores, os quais podem ser ajustados pelo utilizador.

Por último, também decidimos tratar de alguns erros que poderiam acontecer como, por exemplo, dar um número de aluno que não exista na base de dados. Quando isso acontece, surge uma mensagem alertar o utilizador. Alguns do erros são tratados na parte léxica, como por exemplo, o número de aluno ter obrigatoriamente o formato AXXXXX/PGXXXXX.

5.2 Testes realizados e Resultados

Para testar a viabilidade da nossa solução, fizemos testes com exemplos de ficheiros com variados inputs e com vários tamanhos. Iremos agora listar os testes que pretendemos fazer com cada ficheiro de input (todos os ficheiros estarão disponíveis do zip enviado com o resto do trabalho):

- **in.txt** - Exemplo simples que demonstra o resultado obtido;
- **in_top.txt** - Exemplo simples com adição de um valor definido pelo utilizador, de modo obter no resultado o determinado valor de recursos mais adequados;
- **in_com_fatores.txt** - Exemplo simples com adição dos valores dos factores definidos pelo utlizador;
- **in_100.txt** - Exemplo com grande quantidade de recursos, de modo a verificar a eficiência do algoritmo;
- **in_aluno_nao_existe.txt** - Exemplo simples com um aluno que não existe na base de dados.

Concluímos assim que obtivemos bons resultados, sendo que o nosso código se comportou da maneira esperada em todos estes testes.

De seguida apresentamos um exemplo do *output* da aplicação, sendo que o objetivo era determinar quais os três recursos ideias para o aluno **A82500** aprender **JavaScript**.

```
Aluno A82500 com 20 anos e com as seguintes caracteristicas: [comunicativo, social]

Os 3 melhores recursos para o aluno A82500 aprender o conceito JavaScript são:

1:
Recurso com id 6
Tipo: Video
Titulo: [Front-End Engineer]
Público Alvo: dos 17 aos 25 anos
Caracteristicas: [produtivo, empreendedor, criativo]
Conceitos: [Front-End, HTML, CSS, JavaScript]
--------------------------------------------------------------------------------
2:
Recurso com id 2
Tipo: Curso online
Titulo: [Curso de programacao sobre JavaScript]
Público Alvo: dos 17 aos 25 anos
Caracteristicas: [inovador, inteligente, autonomo]
Conceitos: [JavaScript]
--------------------------------------------------------------------------------
3:
Recurso com id 12
Tipo: Curso online
Titulo: [Create Video Games with Phaser.js]
Público Alvo: dos 16 aos 23 anos
Caracteristicas: [calculista, distraido, problemas]
Conceitos: [Phaser.js, JavaScript, Animacoes, Game Physics, Programacao]
--------------------------------------------------------------------------------
```

6 Conclusão

Em suma, neste trabalho, o grupo considera que aprofundou muito o seu conhecimento sobre as ferramentas usadas na unidade curricular. Foi dada a oportunidade de aperfeiçoar o uso da ferramenta ANTLR e expandir o conhecimento sobre gramáticas de atributos (GA). Por fim, com este trabalho, o objetivo futuro será implementar um sistema capaz de reconhecer vários tipos de querys distintas, ajudando assim o utlizador em outras decisões.