Instituto Politécnico Cávado do Ave

Curso engenharia de Sistemas Informáticos

Processamento de Linguagem

Autores

João Ricardo Nº 18845

João Rodrigues Nº 19431

Carlos Santos Nº 19432

Test Anything Protocol

Data: 22/11/2020

ÍNDICE

RESUMO

O *Test Anything Protocol* é um formato textual usado por ferramentasde testes unitários desenvolvidas para várias linguagens, desde o Perl ao C.

```
1..3
ok 1 - One is One
not ok 2 - Two is Three
#   Failed test 'Two is Three'
#   at subtests line 4.
#          got: '2'
#     expected: '3'
ok 3 - Four is Four
```

O que se pretende é o desenvolvimento de uma ferramenta que permita analisar este tipo de outputs.

Pretende-se que tenha as seguintes funcionalidades:

- Gerar, um relatório para cada ficheiro, que contenha um resumo de números de testes executados, número de testes com resultado positivo e percentagem de falhas;
- Gerar um HTML para cada um dos relatório, e que o apresente de forma visual (e colorida) quais os testes com sucesso;
- Dada uma pasta com vários ficheiros, em que cada um é um relatório independente, gerar páginas HTML interligadas em que é possível consultar visualmente o dados dos relatórios.

[1]

[1] https://testanything.org/

CONTEXTUALIZAÇÃO

Este documento foi realizado no contexto da unidade curricular Processamento de Linguagem, do Instituto Politécnico Cávado do Ave. Neste trabalho pretende-se criar uma ferramenta de análise dos ficheiros.

O modo da abordagem deste tipo de problema foi:

- Criar expressões regulares capazes de ler os ficheiros;
- Guardar os Dados de forma a serem manipulados mais tarde;
- Utlização do **HTML** e **CSS** para a página web com a informação detalhada relativa ao ficheiro(Nº de Testes, quais positivos, negativos, decrições,...);

CRIAÇÃO DE EXPRESSÕES REGULARES

Uma Expressão Regular é uma forma concisa e flexível de identificar cadeias de caracteres de interesse, como caracteres particulares, palavras ou padrões de caracteres. Estas Expressões Regulares são analisadas e um processador de expressão regulares examinam os textos e identificam as partes que coincidem.

Para a escolha das expressões foram feitas análises e sucessivas tentativas a ficheiros de testes.

```
1..3
ok 1
not ok 2
        # Subtest: Some subtests first
        not ok 1 - my subtest 1
                # Subtest: Some subtests second
                 ok 1 - my other subtest 1
                 not ok 2 - and another subtest 2
                 1..2
        ok 2 - Some subtests second
        ok 3 - some more subtests
        1..3
ok 3 - Some subtests first
```

Na figura abaixo mostra um expressão regular utilizada na leitura do total de testes executados.

```
def t_TOTAL_STAGES(t):
    r"[0-9 ]+..[0-9]+"
    return t
```

Esta função analisa e reconhece este tipo de de string **"1..2, 1..4, ..."**, assim, no ficheiro, o que tenta procurar é um número(com 1 ou mais algarismos), dois pontos(..) e por fim outro número(com 1 ou mais algarismos).

Através desta Expressão foi possível saber o número de Testes executados, neste caso (através da análise das duas imagens - Figura 1 e Figura 2), o output esperado será um total de 8 Testes.

Output obtido:

```
LexToken(TOTAL_STAGES,'1..3',1,0)
LexToken(TOTAL_STAGES,' 1..2',1,189)
LexToken(TOTAL_STAGES,' 1..3',1,264)
```

ARMAZENAMENTO DE DADOS

Os Dados para serem manipulados têm de ser guardados em estruturas de dados eficientes, tendo em conta que será necessário fazer cálculos estatísticos. Assim sendo os valores estarão guardados numa Classe

```
class Test:
        def _init_(self, result="tba", stage=0, description="tba", level=0,
        comment = ):
            self.result = result
            self.stage = stage
            self.description = description
            self.level = level
            self.comment = comment
```

Nesta classe são guardados os dados referentes aos ficheiros lidos.

- result - Se o teste foi bem efetuado(Ok ou Not Ok);
- stage - Número do teste(1,2,3...);
- description - Infomação acerca do resultado;
- level - Se é um Teste/Subteste;
- comment - Comentário relativo ao teste;

HTML E CSS

Um dos objetivos do projeto era fazer gerar um relatório de forma visual(colorida) dos testes com sucesso por um lado. Por outro, com uma pasta de vários ficheiros interligar os diferentes relatórios de forma que os fosse possível consultar.

```
writeFile(html_file, «span class="test"onmouseenter="viewDesc"+ str(i) +
"()"onmouseout="closeDesc"+ str(i) + "()"style=margin-left:"
+ str((test.level - 1) * 2) + "em;")
if test.result == "ok":
        writeFile(html_file, "color:green>")
else:
        writeFile(html_file, "color:red>")
```

Aqui esta o output do ficheiro **HTML**, como resultado da leitura dos ficheiros testes.

Figura 1: Página Web gerada

Esta é a página web gerada com as linguagens **CSS** e **HTML** proporcionando uma leitura melhor acerca dos dados de cada ficheiro, de forma individual ou em pasta, como input no programa.

CONCLUSÃO

Na nossa opinião foi muito interessante o desenvolvimento deste projeto, pois potenciou a experiência do desenvolvimento de Software. Assimilar os conteúdos da Unidade Curricular, desenvolver Capacidades de programação em *PYTHON*, e na linguagem de marcação *HTML*.

Sentimos que este projeto foi bastante exigente e fez com que nos dedicássemos mais e melhorar-mos as nossas capacidades.

Com este Trabalhos adquirimos inúmeras valias que nos serão úteis em futuros projetos.

Em suma, abordamos todos os assuntos lecionados e graças a isso conseguimos cumprir os objetivos propostos.