Lab work nº 2

Relatório

Teoria Algorítmica da Informação

Gil Teixeira

João Relva

Vera Oliveira

Dezembro 2021

Índice

1 Introdução

Neste segundo trabalho prático da unidade curricular de Teoria Algorítmica da Informação, os alunos tinham como objetivo desenvolver um total de três programas, existindo também a possibilidade de realização de um exercício bónus.

O primeiro programa deveria receber dois ficheiros de texto – um ficheiro de referência (que representa um determinado idioma) e um ficheiro alvo (texto a ser analisado). O objetivo seria obter o número de bits necessários para comprimir o texto alvo, usando o modelo calculado a partir de um determinado texto de referência. O desafio do programa seguinte seria construir um sistema que permitisse reconhecer o idioma de um determinado ficheiro alvo, após a aprendizagem através de um conjunto de textos escritos em diferentes linguagens. O último programa a implementar tinha como finalidade processar um texto que tivesse segmentos escritos em várias linguagens e indicar a localização de cada segmento no ficheiro a ser analisado e qual a linguagem em que estava escrito.

Relativamente ao exercício bónus o objetivo seria explorar a possibilidade de utilizar combinações de vários *"finite-context models"* para representar cada idioma.

Este relatório contém a explicação do código desenvolvido, os vários resultados obtidos e as conclusões finais.

O trabalho desenvolvido foi disponibilizado num repositório no GitHub.

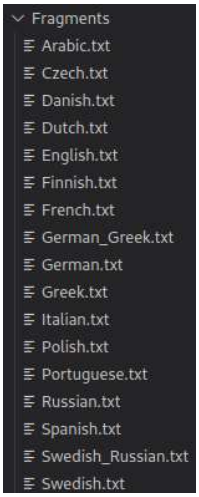
2 Desenvolvimento

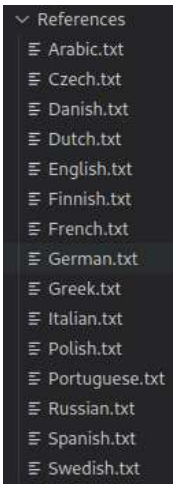
2.1 Design

Para o desenvolvimento deste trabalho continuámos a utilizar a linguagem de programação usada no trabalho anterior (Python) e todo o código foi implementado e testado em Linux.

Considerando todo o contexto do trabalho e as tarefas que deveriam ser implementadas, criámos quatro ficheiros *.py*: *fcm.py*, *lang.py*, *findlang.py* e *locatelang.py*. O *fcm.py,* responsável pela criação do modelo finito de contextos e que foi implementado no trabalho anterior, serviu de base para a realização deste segundo trabalho (devido a uma questão de simplicidade e conveniência decidimos reformular totalmente este programa). O *lang.py* retorna o número de bits necessários para comprimir um texto *target*, o *findlang.py* permite reconhecer a linguagem em que foi escrito determinado texto e o *locatelang.py,* após fornecido como input um texto com segmentos escritos em vários idiomas, indica a localização de um determinado segmento e a linguagem em que foi escrito.

O Projeto contém duas pastas – *References* e *Fragments*, que incluem, respetivamente, textos de referência para determinadas linguagens e textos alvo que serão analisados. Os nomes dos ficheiros (tanto os de referência como os fragmentos) correspondem à linguagem a que estão associados, como se poderá verificar nas imagens seguintes.





As instruções para a execução dos vários programas estão presentes no ficheiro "*README"* do repositório do projeto (link na secção *"Project Repository"*).

2.2 Implementação

Esta secção apresenta os principais passos dados na implementação dos vários programas. Apesar de o programa *fcm.py* ter sofrido uma reimplementação considerámos que não seria de interessante colocar os detalhes do seu desenvolvimento neste relatório sendo um assunto já abordado no relatório anterior.

2.2.1 lang.py

Como já foi referido anteriormente, o objetivo deste programa seria retornar o número de bits necessários para comprimir um determinado texto. O programa recebe como *inputs*: um texto de referência de uma linguagem, um texto *target*, um valor para o *k* e um valor para o *alpha*. Primeiramente é criada a tabela de contextos para o texto de referência, que irá conter todos os contextos e a contagem dos caracteres que aparecem a seguir a cada contexto. De seguida, o texto *target* é percorrido de acordo com a janela deslizante (dependendo do valor de k), e utilizando a tabela de contextos para o texto de referência já criada, é aplicada a seguinte fórmula (tendo em conta que C(t | ri)):

$$P(e|c) \approx \frac{N(e|c) + \alpha}{\sum_{s \in \Sigma} N(s|c) + \alpha|\Sigma|}, \qquad -\log P(e|c),$$

A soma de todos os logaritmos corresponde ao número absoluto de bits necessários para comprimir o texto *target* utilizando determinado texto de referência. Após a obtenção do número de bits é feita uma normalização a esse resultado de forma a reduzir o mesmo para um no intervalo entre 0 e 1.

2.2.2 findlang.py

Este programa recebe como *inputs*: um ficheiro *target*, um valor de *k* e um valor de *alpha*. Tem como objetivo identificar em que linguagem foi escrito um determinado texto *target* e a principal função é a *guessLang()* que faz o seguinte: através do programa anterior (*lang.py*), calcula o número de bits necessários para comprimir o texto alvo, utilizando cada um dos textos de referência presentes na pasta *References*, e guarda os valores dos bits num dicionário (valores normalizados), associando cada valor à sua linguagem. O valor mínimo nesse dicionário irá corresponder à linguagem em que está escrito o texto *target*.

2.2.3 locatelang.py

Este programa tinha como objetivo processar ficheiros com segmentos escritos em várias linguagens e identificar a que linguagens pertenceriam esses mesmo segmentos, indicando também a sua localização. O programa deveria retornar a posição do início de cada segmento assim como a linguagem do mesmo.

Esta aplicação recebe como *input* apenas um ficheiro *target*. Como primeira abordagem decidimos criar o perfil de complexidade para cada língua de referência. De seguida procedemos à suavização de cada perfil de complexidade através de uma *moving average* (janela deslizante com tamanho igual ao valor de *k*, valor esse que decidimos definir diretamente no código assim como o *alpha*, calculando a média a cada desvio. Esse novo perfil de complexidade já "suavizado" foi guardado num *array*. O próximo passo passou pela definição de um *threshold* (*log2(cardinalidade_target)/2*) que foi usado posteriormente para identificar as regiões que estariam abaixo do valor do mesmo. Como já foi mencionado, o objetivo final seria guardar as coordenadas dessas regiões e respetivos idiomas.

Devido a alguns problemas de implementação não conseguimos realizar da forma mais correta esta última parte que foi referida, que seria identificar o início dos vários segmentos identificando as respetivas linguagens.

Após a identificação das várias regiões situadas abaixo do *threshold*, em que foram guardadas num array todas as posições que as mesmas ocupam no ficheiro *target,* deparámo-nos com as seguintes questões:

- Existem várias linguagens a serem detetadas que não constam no ficheiro *target*
- Existem regiões identificadas que coincidem em várias linguagens
- As posições das regiões identificadas não estão a ser identificadas de forma correta

Resumidamente, a nossa aplicação está a guardar todas as coordenadas do texto alvo identificadas abaixo do threshold, para todas as linguagens de referência.

3 Resultados

Este seção apresenta todos os resultados obtidos através da execução dos vários programas implementados. Estes testes foram realizados num *laptop* com um processador Intel Core i7-4558U e 8GB de RAM dentro de uma máquina virtual com o sistema operativo Kali Linux 5.14.16.

3.1 Inputs/outputs

3.1.1 lang.py

Referência	Fragmento	K	Alpha
Portuguese.txt	Portuguese.txt	3	1

```
└─$ python3 lang.py References/Portuguese.txt Fragments/Portuguese.txt 3 1

With: References/Portuguese.txt

Bits to compress Fragments/Portuguese.txt: 269648
Bits Normalized: 0.3259403284381734

-------------------------EXECUTION TIME-------------------------
0:00:03.529708
```

Referência	Fragmento	K	Alpha
Portuguese.txt	Portuguese.txt	5	0.001

```
└─$ python3 lang.py Fragments/Portuguese.txt  Fragments/Portuguese.txt 5 0.001

With: Fragments/Portuguese.txt

Bits to compress Fragments/Portuguese.txt: 125597
Bits Normalized: 0.15

-------------------------EXECUTION TIME-------------------------
0:00:00.363284
```

Referência	Fragmento	K	Alpha
English.txt	Portuguese.txt	2	1

```
└─$ python3 lang.py References/English.txt Fragments/Portuguese.txt 2 1

With: References/English.txt

Bits to compress Fragments/Portuguese.txt: 589875
Bits Normalized: 0.713018842179082

-------------------------EXECUTION TIME-------------------------
0:00:05.526452
```

Referência	Fragmento	K	Alpha
English.txt	Portuguese.txt	4	1

```
└─$ python3 lang.py References/English.txt Fragments/Portuguese.txt 4 1


With: References/English.txt


Bits to compress Fragments/Portuguese.txt: 697739
Bits Normalized: 0.8433997983488347


----------------------EXECUTION TIME----------------------
0:00:06.548662
```

Referência	Fragmento	K	Alpha
Russian.txt	English.txt	3	1

```
└─$ python3 lang.py References/Russian.txt Fragments/English.txt 3 1


With: References/Russian.txt


Bits to compress Fragments/English.txt: 1579411
Bits Normalized: 0.9591691154171689


----------------------EXECUTION TIME----------------------
0:00:07.453320
```

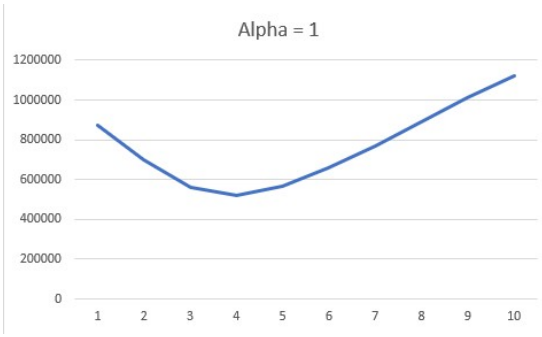
Referência	Fragmento	K	Alpha
Russian.txt	English.txt	3	0.1

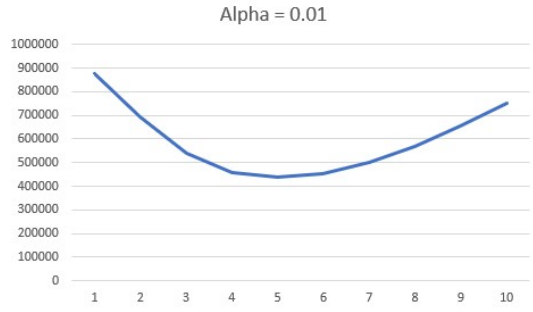
```
└─$ python3 lang.py References/Russian.txt Fragments/English.txt 3 0.1


With: References/Russian.txt


Bits to compress Fragments/English.txt: 1532275
Bits Normalized: 0.9305436947684799


----------------------EXECUTION TIME----------------------
0:00:07.873435
```

Referência	Fragmento	K	Alpha
Dutch.txt	Greek.txt	1	0.1

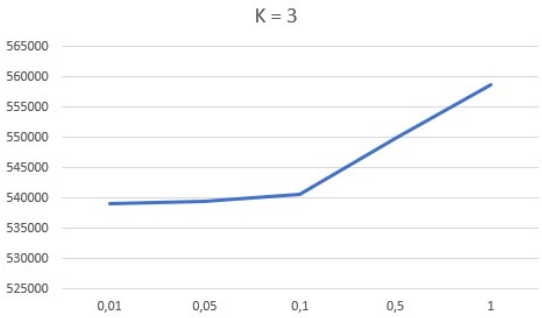
```
└─$ python3 lang.py References/Dutch.txt Fragments/Greek.txt 1 0.1

With: References/Dutch.txt

Bits to compress Fragments/Greek.txt: 4319223
Bits Normalized: 1.0750031767997836

----------EXECUTION TIME----------
0:00:00.630122
```

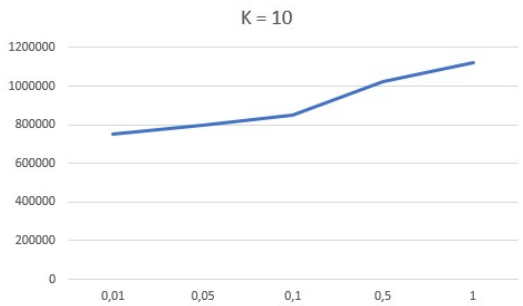
Os gráficos seguintes representam o número de *bits* necessários para comprimir o fragmento para cada valor de *k*, para *alpha* igual a 1 e 0.01 respetivamente. Foi utilizado o mesmo texto de referência (*English.txt*, exemplo de referência localizado na pasta *References*) e o mesmo texto target (*English.txt*, exemplo de *target* localizado na pasta *Fragments*) para a construção dos gráficos. Como é possível observar, numa fase inicial o número de *bits* necessários para comprimir o ficheiro alvo diminuem, mas a partir de *k* igual a 4 ou 5 (dependendo do valor de *alpha*) já começam a aumentar novamente.





Os gráficos seguintes representam o número de bits necessários para comprimir o fragmento, mas para cada valor de *alpha*, para *k* igual a 3 e 10 respetivamente. Foi também utilizado o mesmo de texto de referência e de *target* assim como no exemplo anterior. É possível observar que à medida que os valores de *alpha* aumentam o número de *bits* necessários para comprimir o ficheiro também aumentam, tendo uma subida mais acentuada a partir do *alpha* igual a 0.1.





Durante a realização dos testes verificou-se ainda que para línguas de referência e fragmentos mais semelhantes (por exemplo, espanhol e português) o número de *bits* necessários para comprimir esses fragmentos não são tão elevados comparativamente com línguas de referência e fragmentos mais díspares (por exemplo, português e árabe).

3.1.2 findlang.py

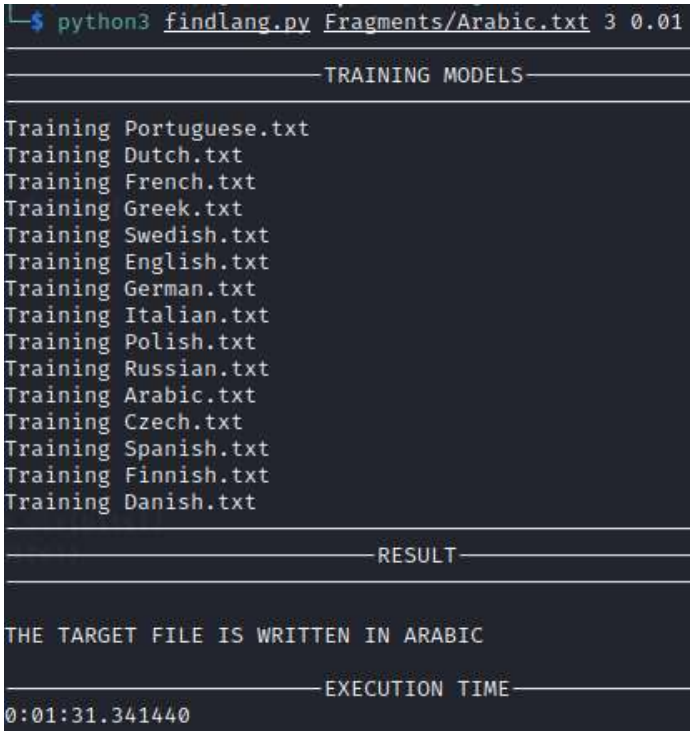
Fragmento	K	Alpha
Portuguese.txt	1	1

```
└─$ python3 findlang.py Fragments/Portuguese.txt 1 1
_____________________TRAINING MODELS_____________________
Training Portuguese.txt
Training Dutch.txt
Training French.txt
Training Greek.txt
Training Swedish.txt
Training English.txt
Training German.txt
Training Italian.txt
Training Polish.txt
Training Russian.txt
Training Arabic.txt
Training Czech.txt
Training Spanish.txt
Training Finnish.txt
Training Danish.txt
_____________________RESULT_____________________

THE TARGET FILE IS WRITTEN IN PORTUGUESE

_____________________EXECUTION TIME_____________________
0:01:01.464177
```

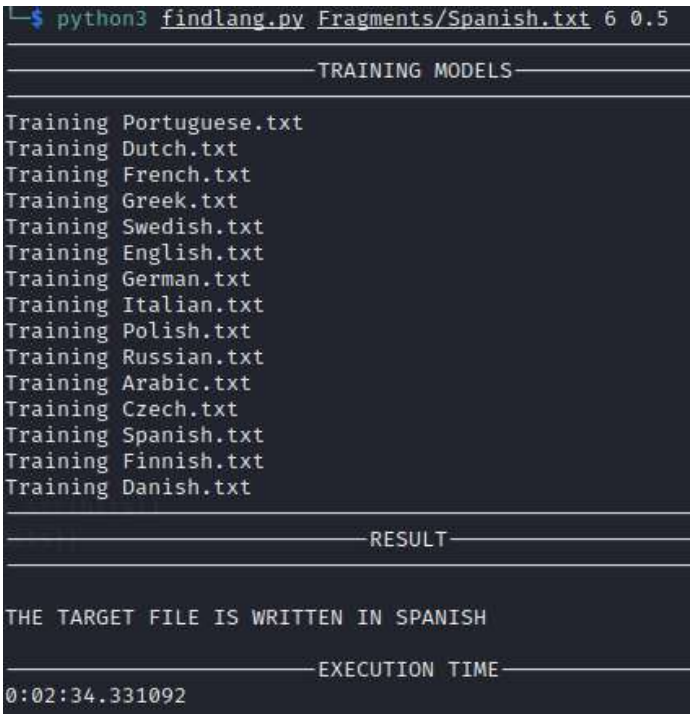
Fragmento	K	Alpha
Swedish.txt	3	1

```
└─$ python3 findlang.py Fragments/Swedish.txt 3 1
_____________________TRAINING MODELS_____________________
Training Portuguese.txt
Training Dutch.txt
Training French.txt
Training Greek.txt
Training Swedish.txt
Training English.txt
Training German.txt
Training Italian.txt
Training Polish.txt
Training Russian.txt
Training Arabic.txt
Training Czech.txt
Training Spanish.txt
Training Finnish.txt
Training Danish.txt
_____________________RESULT_____________________

THE TARGET FILE IS WRITTEN IN SWEDISH

_____________________EXECUTION TIME_____________________
0:01:38.640975
```

Fragmento	K	Alpha
Arabic.txt	3	0.01



Fragmento	K	Alpha
Arabic.txt	6	0.5



Relativamente aos testes realizados a este programa observou-se que todas as linguagens em que estão escritos os exemplos de textos existentes foram detetadas com sucesso (para valores de k menores ou iguais a 6 e para qualquer valor de alpha).

3.1.3 locatelang.py

Devido ao facto de este programa não estar implementado da forma esperada e apenas guardar todas as coordenadas do texto identificadas abaixo do *threshold* definido, colocámos aqui apenas um exemplo para um pequeno texto *target* escrito em Inglês e Francês (*English_French.txt*).

```
Checking French ...
[21, 22, 33, 34, 46, 47, 55, 59, 60, 62, 63, 64, 65, 66, 100, 101, 115, 116, 117, 118, 119, 120, 123, 124, 125, 126, 127, 128, 129, 140,
141, 167, 171, 172, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 191, 192, 193, 195, 200, 201, 202, 203, 210, 211, 215, 219, 22
0, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 256, 257, 258, 273, 283, 284, 285, 286, 287, 288, 302, 303, 312, 313, 316, 322, 323,
 324, 325, 338, 359, 360, 366, 367, 368, 369, 370, 371, 372, 376, 381, 392, 393, 394, 395, 396, 397, 398, 404, 405, 406, 412, 413, 414, 4
18, 419, 420, 421, 422, 423, 426, 427, 428, 429, 430, 431, 432, 433, 497, 498, 499, 500, 501, 502, 503, 504, 509, 515, 517, 524, 553, 554
, 555, 556, 593, 594, 600, 601, 634, 635, 636, 637, 638, 640, 641, 642, 643, 644, 645, 652, 657, 658, 659, 660, 661, 662, 683, 684, 685,
686, 687, 688, 689, 690, 691, 692, 693, 694, 717, 718, 719, 720, 721, 723, 724, 729, 730, 740, 742, 749, 750, 751, 762, 763, 764, 765, 76
6, 767, 778, 779, 780, 787, 788, 789, 790, 800, 801, 802, 804, 805, 809, 811, 820, 827, 861, 862, 863, 864, 865, 866, 867, 874, 875, 876,
 878, 879, 880, 882, 883, 884, 885, 907, 908, 909, 916, 917, 918, 919, 949, 951, 959, 960, 961, 962, 963, 964, 971, 972, 973, 974, 975, 9
76, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002,
1003, 1004, 1005, 1006, 1007, 1008, 1009, 1014, 1015, 1016, 1017, 1018, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030,
 1033, 1034, 1035, 1036, 1041, 1042, 1043, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1059, 1060, 1061, 1062, 1063
, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1073, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1088, 1090, 109
```

```
Checking English ...
[1, 2, 3, 4, 6, 7, 8, 9, 10, 13, 14, 15, 18, 19, 20, 21, 22, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 43, 44, 45, 46,
47, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 84, 8
5, 86, 87, 93, 94, 95, 96, 97, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 115, 116, 117, 118, 119, 120, 121, 123, 124, 125, 12
6, 127, 128, 129, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 146, 147, 148, 149, 150, 151, 153, 154, 155, 158, 159, 160, 161, 162,
 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 1
91, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 207, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223
, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250,
251, 252, 253, 254, 255, 256, 258, 259, 260, 261, 262, 263, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 28
1, 282, 283, 284, 285, 286, 287, 288, 289, 290, 298, 299, 300, 302, 303, 304, 305, 308, 309, 310, 311, 312, 316, 317, 318, 322, 323, 324,
 325, 326, 327, 328, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 350, 351, 352, 353, 360, 361, 3
```

4 Conclusões

4.1 lang.py

- À medida que os valores de *k* aumentam, o número de *bits* necessários para comprimir um ficheiro diminuem, mas só até a um determinado valor (dependendo do valor de *alpha*, mas a rondar o *k* igual a 4 ou a 5) visto que os contextos, que serão analisados no texto de referência, passam a ser cada vez maiores, ou seja, não são tão frequentes ou até mesmo inexistentes, interferindo no cálculo dos *bits*.
- À medida que os valores de *alpha* aumentam, o número de *bits* necessários para comprimir um ficheiro aumentam também, sendo o aumentam mais notório em valores de *k* superiores.
- Quando um texto de referência e um fragmento são escritos na mesma linguagem o número de *bits* necessários para comprimir o fragmento é mais baixo.
- Quando um texto de referência e um fragmento são exatamente iguais os *bits* de compressão deverão ser perto de zero, especialmente para valores de *k* maior que 3 e *alpha* menor que 0.01.
- Quando as linguagens em que estão escritos os textos de referência e o fragmento são muito diferentes o número de *bits* necessários para comprimir o fragmento são maiores.
- Quanto maior o tamanho de um texto de referência menos serão os bits necessários para comprimir um texto target visto que existe um maior número de contextos que serão analisados.
- É percetível também que quanto mais pequeno o texto *target* for menos serão os bits necessários para o comprimir.

4.2 findlang.py

- O programa deteta qualquer linguagem (desde que exista o seu texto de referência) em que um determinado texto foi escrito seja para qualquer valor de *k* compreendido entre 0 e 6 (inclusive) e qualquer valor de *alpha*. Ainda assim prevê se que os valores ideais para *k* sejam entre 2 e 4 e que o *alpha* seja o menor possível.

4.3 locatelang.py

- Relativamente a este programa não conseguimos retirar muitas conclusões devido ao facto de não estar totalmente finalizado.
- Podemos referir que o tipo de metodologia implementada nesta aplicação não é “perfeita” e apresenta algumas limitações.
- É possível que várias linguagens sejam detetadas num texto *target* em que essas linguagens não estejam presentes.

- Certas posições de segmentos detetadas num idioma poderão coincidir com posições de segmentos detetadas num outro idioma.

Consideramos que todos os programas implementados poderão apresentar algum valor comercial. Dependendo do cliente e utilização final concluímos que existem várias vantagens em adquirir estes produtos. Tanto o lang.py, o findlang.py e o locatelang.py são úteis pois funcionam de forma rápida, eficaz e sem erros associados. Reconhecer a linguagem de um qualquer ficheiro de texto (principalmente com um tamanho elevado) de forma automática e perceber quantos bits são necessários para comprimir também um ficheiro de texto acaba por se tornar bastante útil em vários aspetos, até mesmo a um nível empresarial.

5 Project Repository

- GitHub Repository: https://github.com/bearkillerPT/TAI