

Prácticas de SAR

Sistemas de Almacenamiento y Recuperación de información

Práctica 3: El Mono Infinito

El mono infinito

Descripción del problema



Figura: monkey at work

*El teorema del mono infinito afirma que un mono pulsando teclas **al azar** sobre un teclado durante **un periodo de tiempo infinito** casi seguramente podrá escribir finalmente cualquier libro que se halle en la Biblioteca Nacional de Francia.*

Objetivo de la práctica

Ya que no tenemos tanto tiempo, crearemos un programa en python que procese un documento y que utilice la información extraída de él para ayudar al mono a escribir su libro.

Ejercicio

Tarea

Escribir 2 programas en python:

❶ Creador del índice (**SAR_p3_monkey_indexer.py**):

- El programa recibe dos argumentos (el nombre de dos ficheros):
 - ❶ A partir del primer fichero crea un índice
 - ❷ Guarda el índice en disco utilizando el segundo nombre

❷ Mono infinito evolved (**SAR_p3_monkey_evolved.py**):

- Recibe el nombre de un fichero como argumento:
 - ❶ Carga el índice de disco
 - ❷ Genera 10 frases

¿Qué debo hacer? (Monkey Indexer)

Tokenización:

- La separación entre frases vendrá dada por “.”, “;”, “!”, “?” o por dos saltos de línea.
- Se eliminarán todos los símbolos no alfanuméricos.
- Los tokens serán las palabras del documento en minúsculas.
- Se añadirá un símbolo especial “\$” que indique inicio y final de frase.

Creador de índices:

- El índice se guardará como una tabla hash (diccionario de python).
- Las claves del diccionario serán los tokens del documento más la palabra especial “\$” .
- Cada entrada del diccionario contendrá:
 - El número total de veces que ha aparecido el token.
 - Una lista con todas los tokens que han aparecido en el documento después de la clave (incluido “\$”) y el número de veces que ha sucedido.
- La lista de sucesores debe estar ordenada por el número de apariciones del par de tokens.

“spam.txt”:

Egg and Bacon;

Egg, sausage and Bacon;

Egg and Spam;

Spam Egg Sausage and Spam;

Egg, Bacon and Spam;

Egg, Bacon, sausage and Spam;

Spam, Bacon, sausage and Spam;

Spam, Egg, Spam, Spam, Bacon and Spam;

Spam, Spam, Spam, Egg and Spam;

Spam, Spam, Spam, Spam, Spam, Spam, Spam, Spam, baked beans, Spam, Spam, Spam and Spam;

Lobster Thermidor aux crevettes with a Mornay sauce, garnished with truffle pate, brandy and a fried egg on top and Spam

¿Qué debo hacer?

Índice para "spam.txt":

```
$ (11, [(5, 'spam'), (5, 'egg'), (1, 'lobster')])
a (2, [(1, 'mornay'), (1, 'fried')])
and (12, [(9, 'spam'), (2, 'bacon'), (1, 'a')])
aux (1, [(1, 'crevettes')])
bacon (6, [(2, 'sausage'), (2, 'and'), (2, '$')])
baked (1, [(1, 'beans')])
beans (1, [(1, 'spam')])
brandy (1, [(1, 'and')])
crevettes (1, [(1, 'with')])
egg (9, [(3, 'and'), (2, 'sausage'), (2, 'bacon'), (1, 'spam'), (1, 'on')])
fried (1, [(1, 'egg')])
garnished 1, [(1, 'with')])
lobster (1, [(1, 'thermidor')])
mornay (1, [(1, 'sauce')])
on (1, [(1, 'top')])
pate (1, [(1, 'brandy')])
sauce (1, [(1, 'garnished')])
sausage (4, [(4, 'and')])
spam (27, [(11, 'spam'), (9, '$'), (3, 'egg'), (2, 'bacon'), (1, 'baked'),
(1, 'and')])
thermidor (1, [(1, 'aux')])
top (1, [(1, 'and')])
truffle (1, [(1, 'pate')])
with (2, [(1, 'truffle'), (1, 'a')])
```

¿Cómo se inicia una frase?

- Se elige como palabra inicial '\$'.

¿Cómo se elige cada siguiente palabra?

- Las palabras siguientes se eligen sucesivamente de forma “**aleatoria ponderada**” entre las sucesoras de la palabra actual teniendo en cuenta el número de veces que ha aparecido.

¿Cuándo se termina una frase?

- Un número máximo de palabras por frase, 25 en nuestro caso.
- La palabra elegida es la palabra “final” especial (\$).

Cosas útiles

pickle

```
import pickle

def save_object(object, file_name):
    with open(file_name, 'wb') as fh:
        pickle.dump(object, fh)

def load_object(file_name):
    with open(file_name, 'rb') as fh:
        obj = pickle.load(fh)
    return obj
```

librería random

```
import random
```

```
random.randint(a, b)
```

"Return a random integer N such that $a \leq N \leq b$."

```
random.choice(seq)
```

"Return a random element from the non-empty sequence seq. If seq is empty, raises IndexError."

Ampliación

Se proponen como ampliación:

- Realizar la indexación teniendo en cuenta grupos de tres palabras (trigramas) que aparecen en las frases.
 - Cada entrada del diccionario será una tupla (w_1 , w_2) que enlazará con las palabras vistas detrás de w_1 y w_2 .
- Realizar la generación de frases teniendo en cuenta el nuevo índice.
 - ¿Cuál es la primera palabra de cada frase?
 - ¿Y la segunda?