UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

Curso 2022/2023

Guion de prácticas

Language3

class Language

Abril de 2023

Índice

1. Definición del problema

Como ya sabemos, las prácticas tienen como objeto principal trabajar con textos escritos en diferentes idiomas. Así pues, vamos a desarrollar un conjunto de aplicaciones sobre ficheros de texto que nos permitan averiguar automáticamente el idioma en el que está escrito un texto.

Más concretamente, a la llegada de un nuevo texto, en un idioma desconocido, se quiere hallar el language al que más se ajusta (de entre un conjunto de language candidatos) el nuevo texto para realizar nuestra predicción. En esta práctica vamos a dar un paso más en nuestro proceso de predicción.

De momento, seguimos utilizando ficheros languages, sin preocuparnos de los textos de los que proceden. De esta forma, el problema de la predicción de un texto lo vamos a reducir a resolver el siguiente problema:
Dado un language, L_x de idioma desconocido, y un conjunto de languages de referencia, $L_1, L_2, ..L_i$, se quiere calcular la distancia de L_x a cada uno de los languages $L_1, L_2, ..L_i$ y determinar aquel con menor distancia, para asignarle su idioma al language de idioma desconocido.
Para identificar el language más próximo, es necesario definir una medida de distancia entre dos languages, esto es, medir el parecido entre dos languages. Los detalles se muestran en sección 4.
Los distintos languages que van a intervenir, en nuestro proceso de predicción, se van a almacenar todos en memoria dinámica; esto nos va a permitir introducirnos en la gestión dinámica de memoria de forma práctica, inicialmente fuera de una clase.

2. Arquitectura de las prácticas

Como ya se indicó en la práctica anterior, la práctica `Language` se ha diseñado por etapas, las primeras contienen estructuras más sencillas, sobre las cuales se asientan otras estructuras más complejas y se van completando con nuevas funcionalidades.

En `Language3` se amplía la funcionalidad de la clase `Language`, bloque **C'** de la Figura 1, el resto de clases permanecen iguales.

C Language.cpp
: Se amplía la clase con la implementación del método `getDistance()`. La componente privada de datos de la clase Language es la misma que en Language2, un vector estático de objetos `BigramFreq`. Esto supone cambios mínimos en la clase.

La carga práctica de Language3 consiste en la gestión de memoria dinámica para el almacenamiento de los languages que intervienen en el programa y en la incorporación y tratamiento de nuevos parámetros a la función `main()`, lo que implica un nuevo tratamiento de los argumentos de main.

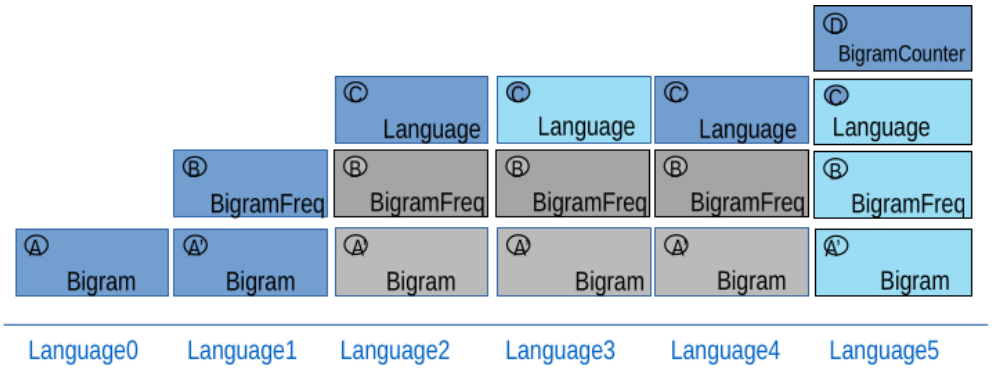

Figura 1: Arquitectura de las prácticas de MP 2023. Los cambios esenciales en las clases (cambio en estructura interna de la clase) se muestran en azul intenso; los que solo incorporan nuevas funcionalidades en azul tenue. En gris se muestran las clases que no sufren cambios en la evolución de las prácticas.

3. Objetivos

El desarrollo de esta práctica `Language3` persigue los siguientes objetivos:

- Practicar con punteros y memoria dinámica fuera de una clase.
- Practicar con herramientas como el depurador de NetBeans y `valgrind` para rastrear errores en tiempo de ejecución.
- Profundizar en el paso de parámetros a `main()` desde la línea de comandos (parámetros alternativos, optativos y por defecto).

4. Cálculo de distancia entre languages

Ya sabemos que un fichero language es un fichero que contiene una lista de bigramas identificados como frecuentes en un determinado idioma. Los bigramas más frecuentes, se encuentran arriba en el orden y los menos frecuentes al final de la lista. Indicar que, todos los bigramas que figuran en un language tienen una frecuencia mínima de 1, esto es, han aparecido alguna vez en el texto origen.

Vamos a definir una medida de distancia entre dos objetos de tipo language. Para ello, indicar que es importante especificar el orden relativo de los languages, a la hora de calcular la distancia, $distance(L_1, L_2)$. Esto es así, porque la medida no es simétrica, como luego mostraremos. Sea L_1 el language cuyo idioma puede ser desconocido (o no) y L_2 es el language con el que se quiere medir el parecido.

La distancia se calcula de la siguiente forma:

$$distance(L_1, L_2) = \frac{\sum_{bigram_i(L_1)} |rank^{L_1}_{bigram_i(L_1)} - rank^{L_2}_{bigram_i(L_1)}|}{size(L_1) * size(L_1)}$$

donde, $bigram_i(L_j)$ es el i-ésimo bigrama de language $L_j, j \in \{1,2\}$, y $rank^{L_k}_{bigram_i(L_j)}$ es la posición en el orden (rank) del i-ésimo bigrama del language $L_j, j \in \{1,2\}$ en el language L_k. Al tratarse de restas de valores, se utiliza la función $||$, valor absoluto, para que no se compensen unos términos con otros.

Como se puede observar, los valores de frecuencia no intervienen en el cálculo de la distancia, tan solo sirven para establecer el $rank$ de un bigrama, posición dentro del orden de los bigramas de un language. Para los valores de $rank$, se considera 0 la primera posición, 1 la segunda y así sucesivamente.
Dado un bigrama determinado de L_1, $bigram_i(L_1)$, que se encuentra en una posición, $rank^{L_1}_{bigram_i(L_1)}$, este se busca en L_2 obteniendo así su posición, $rank^{L_2}_{bigram_i(L_1)}$.
Sin embargo, si el bigrama no se encuentra en L_2, se le asigna el máximo valor, $rank^{L_2}_{bigram_i(L_1)} = size(L_1)$. Dicho de otra manera, si un bigrama determinado en L_1 no aparece en L_2, hay una penalización. Esta penalización disminuye conforme el bigrama es menos frecuente en el language origen, ver tabla 4 donde existen numerosos bigramas no presentes en L_2.

Propiedades de la distancia.

- $distance(L_1, L_2) \in [0, 1]$. La distancia es un número real que toma valores en el intervalo $[0, 1]$; está normalizada por el cuadrado de la longitud de L_1.
- Toma el valor cero para $distance(L_1, L_1)$ ya que, coinciden todos los bigramas en las mismas posiciones dentro del orden, luego el numerador siempre es cero.
- La medida de distancia tal como se ha definido no es simétrica. $distance(L_1, L_2) \neq distance(L_2, L_1)$.
- La distancia de un objeto language L_1 a otro objeto language cualquiera L_2 no está definida. Como puede ver en los comentarios del método `getDistance()` en `Language.h`, se debe lanzar una excepción `std::invalid_argument` en el método `getDistance()` si el objeto `*this` no contiene ningún bigrama.

Veamos con dos ejemplos, cómo se calcula la distancia y cómo interpretar que dos language sean más o menos parecidos. Los cálculos que aquí se muestran se han realizado con ficheros language: `prep_xx.bgr`, `prep_sp.bgr` y `prep_en.bgr`, respectivamente L_1, L_2, L_3.

Sea L_1, el language que se muestra a continuación, cuyo idioma es desconocido y tiene una longitud de 10 bigramas.

```
1  MP-LANGUAGE-T-1.0
2  unknown
3  10
4  en 32611
5  de 32547
6  la 29086
7  es 28902
8  ue 28493
9  er 27375
10 qu 26242
```

```
ra 24083
éq 1
úa 1
```

Disponemos de otro language L_2, cuyo idioma es conocido, español, con el que queremos medir la distancia.

```
MP-LANGUAGE-T-1.0
spanish
15
ue 36965
de 34134
en 33872
es 32751
qu 32497
er 27440
os 25559
la 23147
do 21658
an 21541
zl 1
ói 1
úa 1
úf 1
úh 1
```

Vamos a desglosar la distancia, con los valores que aporta cada uno de los bigramas de L_1.
El valor de la distancia, $distance(L_1, L_2) = 0{,}21$, proviene del valor $sum/100$. Los detalles se encuentran en la siguiente tabla. De los 10 bigramas de

	$bigram$	$rank(L_1)$	$rank(L_2)$	$\lvert rank(L_1) - rank(L_2) \rvert$	$\in L_2$
	en	0	2	2	true
	de	1	1	0	true
	la	2	7	5	true
	es	3	3	0	true
	ue	4	4	0	true
	er	5	5	0	true
	qu	6	4	2	true
	ra	7	10	3	false
	éq	8	10	2	false
	úa	9	12	3	true
sum				21	

Cuadro 1: Cálculo desglosado de la distancia $distance(L_1, L_2)$.

L_1, 7 se encuentran en L_2 y en las mismas posiciones o muy similares en el orden de los dos languages.

Por otro lado, disponemos de otro language L_3, cuyo idioma es inglés. Queremos realizar los mismos cálculos para conocer la distancia.

```
MP-LANGUAGE-T-1.0
english
15
he 4078
th 3915
in 2331
er 2136
an 1841
ou 1736
it 1526
nd 1461
at 1398
re 1384
on 1352
kh 1
kr 1
oz 1
où 1
yl 1
```

De los 10 bigramas de L_1, 9 no se encuentran en L_2 lo cual puede observarse en la penúltima columna como un valor de penalización que decrece conforme bajamos en posiciones en el orden de L_1. Finalmente,

	$bigram$	$rank(L_1)$	$rank(L_3)$	$\lvert rank(L_1) - rank(L_3) \rvert$	$\in L_3$
	`en`	0	10	10	false
	`de`	1	10	9	false
	`la`	2	10	8	false
	`es`	3	10	7	false
	`ue`	4	10	6	false
	`er`	5	3	2	true
	`qu`	6	10	4	false
	`ra`	7	10	3	false
	`éq`	8	10	2	false
	`úa`	9	10	1	false
sum				52	

Cuadro 2: Cálculo desglosado de $distance(L_1, L_3)$.

el valor de la distancia, $distance(L_1, L_2) = 0{,}52$. A tenor de las dos distancias calculadas, la conclusión es que, L_1 es más próximo a L_2 que a L_3. Dado el language L_1 el language de menor distancia sería L_2, y se le asignaría como idioma *spanish*.

5. Práctica a entregar

Para la elaboración de la práctica, dispone de una serie de ficheros que se encuentran en `Language3_nb.zip` o en el repositorio de `git`. El proyecto nuevo se llama `Language3`. Recupere y ubique los ficheros proporcionados en los directorios adecuados.

- Implementar el método getDistance() declarado en el nuevo fichero **Language.h**.

- El módulo **main.cpp** tiene por objetivo leer diferentes ficheros language `*.bgr`, con el fin de calcular la distancia del primer language a cada uno de los otros y hallar el language con mínima/máxima distancia, según se haya especificado en la línea de comandos `-min` o bien `-max`.

Un ejemplo de llamada al programa desde un terminal podría ser:

```
Linux> dist/Debug/GNU-Linux/language3 [-t min | max] <file1.bgr> <file.bgr>
[<file2.bgr> ... <filen.bgr>]
```

Notación: Los `[]` indican que no es obligatorio, es opcional y `|` indica alternativa; en este caso: el literal `min` o bien `max`.

1. Todos los parámetros se pasan al programa desde la línea de órdenes.

2. `language3` calcula la distancia de <file1.bgr> a cada uno de los languages que se encuentran en la lista de ficheros suministrada en los parámetros del main.

3. El primer parámetro que puede aparecer es el literal `-t` seguido bien por la cadena `min` o por `max`. Caso de que se especifique `-t min`, el programa identifica el language de menor distancia de todos y se determina el idioma que le corresponde. Caso de que se especifique `-t max`, el programa identifica el language de máxima distancia de todos y se determina el idioma al que corresponde.

4. Si no se introducen los parámetros `-t xxx`, el programa se ejecuta de forma equivalente a si se hubiesen introducido `-t min` (opción por defecto).

5. El programa recibe dos o más ficheros language (el número de ficheros es indeterminado).

6. El programa debe de almacenar la lista de languages contenidos en los ficheros `*.bgr` en un vector dinámico de objetos language. Debe hacer una gestión correcta de la memoria; reservar, usar y liberar.

5.1. Ejemplos de ejecución

El fichero languague3_nb.zip contiene un conjunto de ficheros de prueba, algunos de los cuales se detallan aquí.

Ejemplo 1: Faltan argumentos para la ejecución del programa. Se necesitan al menos dos ficheros language.

```
Linux>language3
Linux>language3 ../Books/30bigrams.bgr
```

La salida del programa sería la siguiente:

```
Error, run with the following parameters:
language3 [-t min|max] <file1.bgr> <file2.bgr> [ ... <filen.bgr>]
```

Ejemplo 2: Algún parámetro no es válido. En el primer caso el párametro `-f` no es válido. En el segundo caso no es válido `-t MIN`, ya que debe ser `-t min`.

```
Linux>language3 -f ../Books/30bigrams.bgr ../Books/30bigrams.bgr
Linux>language3 -t MIN ../Books/30bigrams.bgr ../Books/30bigrams.bgr
```

La salida del programa sería de nuevo la siguiente:

```
Error, run with the following parameters:
language3 [-t min|max] <file1.bgr> <file2.bgr> [ ... <filen.bgr>]
```

Ejemplo 3: Se busca hallar la distancia mínima entre `quijote.bgr` y otros 4 language suministrados.

```
Linux>language3 ../Books/quijote.bgr ../Books/fortunata.bgr
      ../Books/aliceWonder.bgr ../Books/lesMiserables.bgr
      ../Books/BodasdeSangre_FedericoGarciaLorca.bgr
```

La salida del programa da un mensaje similar a este:

```
Distance to ../Books/fortunata.bgr: 0.0554647
Distance to ../Books/aliceWonder.bgr: 0.250935
Distance to ../Books/lesMiserables.bgr: 0.22352
Distance to ../Books/BodasdeSangre_FedericoGarciaLorca.bgr: 0.13272
Nearest language file: ../Books/fortunata.bgr.
Identifier of the nearest language: spanish
```

Ejemplo 4: Se dispone del fichero language `prep_xx.bgr`, y se quiere conocer la distancia a varios languages cuyo idioma es conocido, a saber, inglés, francés, español y alemán (orden respectivo en el que figuran en la línea de comandos).

```
Linux>language3 -t max ../Books/prep_xx.bgr ../Books/prep_en.bgr
      ../Books/prep_fr.bgr ../Books/prep_sp.bgr ../Books/prep_ge.bgr
```

La salida es como sigue:

```
Distance to prep_en.bgr: 0.52
Distance to prep_fr.bgr: 0.39
Distance to prep_sp.bgr: 0.21
Distance to prep_ge.bgr: 0.4
Farthest language file: prep_en.bgr.
Identifier of the farthest language: english
```

6. Código para la práctica

A. Language.h

```
#ifndef LANGUAGE_H
#define LANGUAGE_H


#include <iostream>
#include "BigramFreq.h"

/**
 * @class Language
 * @brief It defines a model for a given language. It contains a vector of
 * pairs Bigram-frequency (objects of the class BigramFreq) and an identifier
 * (string) of the language.
 */
class Language {
public:

    /**
     * @brief Base constructor. It builds a Language object with "unknown" as
     * identifier, and an empty vector of pairs Bigram-frequency.
     */
    Language();

...
    /**
     * @brief Gets the distance between this Language object (\f$L_1\f$) and
     * the given one @p otherLanguage (\f$L_2\f$).
     * The distance between two Languages \f$L_1\f$ and \f$L_2\f$ is
     * calculated in the following way:
     * \f$d = \frac{ \sum_{bigram_i(L_1)} | rank_{bigram_i(L_1)}^{L_1} -
     * rank_{bigram_i(L_1)}^{L_2} | }{size(L_1) * size(L_1)}\f$,
     * where \f$bigram_i(L_j)\f$ is the bigram \f$i\f$ of the Language \f$L_j,
     * j \in \{1, 2\}\f$ and \f$rank_{bigram_i(L_j)}^{L_k}\f$ is the ranking
     * of the bigram \f$i\f$ of the Language \f$L_j, j \in \{1, 2\}\f$ in the
     * Language \f$L_k\f$.
     * The rank of a bigram is the position in which it
     * appears in the list of BigramFreq. We consider 0 as the
     * first position (rank equals to 0). When calculating
     * \f$rank_{bigram_i(L_1)}^{L_2}\f$, if the bigram \f$bigram_i(L_1)\f$
     * does not appears in the Language \f$L_2\f$ we consider that the rank
     * is equals to the size of Language \f$L_1\f$.
```

```
     * Query method
     * @param otherLanguage A Language object. Input parameter
     * @pre The list of bigrams of this and otherLanguage should be ordered in
     * decreasing order of frequency. This is not checked in this method.
     * @throw Throws a std::invalid_argument exception if the implicit object
     * (*this) is empty, that is, it does not have any bigram.
     * @return The distance between this Language object and the given
     * one @p otherLanguage.
     */
    double getDistance(Language otherLanguage);
...
private:
    static const int DIM_VECTOR_BIGRAM_FREQ = 2000; ///< The capacity of the array _vectorBigramFreq
    std::string _languageId; ///< language identifier
    BigramFreq _vectorBigramFreq[DIM_VECTOR_BIGRAM_FREQ]; ///< array of BigramFreq
    int _size; ///< Number of elements in _vectorBigramFreq

    /**
     * @brief Sets the number of BigramFreq objects
     * @param size The size to set in this object
     */
    void setSize(int size);

    static const std::string MAGIC_STRING_T; ///< A const string with the magic string for text files
};
#endif /* LANGUAGE_H */
```