25.03 Algoritmos y Estructuras de Datos

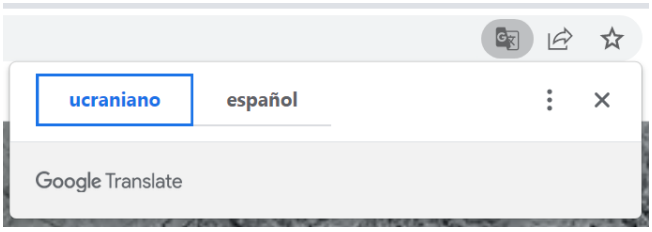
Level 2: Lequel?

Basado en CS106B Assignment 2. Fun with collections.

Introducción

En esta práctica vamos a hacer *language identification*.

El problema consiste en identificar el lenguaje de un texto, tal como lo hace tu navegador cuando visitas una página que está en un idioma diferente al de tu sistema operativo:



En este ejemplo el navegador reconoció que estás visitando una página en ucraniano, y te propone traducirla al español.

Perfiles de trigramas

Para la identificación del lenguaje vamos a seguir el enfoque de *perfiles de trigramas*, que consiste en determinar todas las sub-cadenas de tres caracteres de un texto y contar sus frecuencias. Veamos un ejemplo con una frase corta:

```
"TANTA TINTA TONTA ATENTA"
```

El perfil de trigramas de la frase es:

```
"NTA": 4     "ANT": 1     " TO": 1     " AT": 1
"TA ": 3     " TI": 1     "TON": 1     "ATE": 1
"A T": 2     "TIN": 1     "ONT": 1     "TEN": 1
"TAN": 1     "INT": 1     "A A": 1     "ENT": 1
```

O sea, `"NTA"` aparece cuatro veces, `"TA "`, tres veces, y así sucesivamente. Cada una de estas sub-cadenas es un trigrama (en términos más generales, un n-grama con $n = 3$).

Veamos el perfil de trigramas de un texto más extenso (del artículo “Manzana” de Wikipedia):

```
" de": 598   " la": 318   " co": 230   "s d": 196
"de ": 462   "en ": 285   "la ": 229   "nte": 194
"as ": 345   "es ": 272   "ent": 228   " ma": 190
"os ": 333   " en": 252   "el ": 227       …
```

La primera entrada revela que el artículo contiene muchas palabras que comienzan con `" de"`. También hay muchas palabras que terminan en `"de "`, `"as "`, `"os "`, `"en "` y `"es "`, lo cual probablemente se corresponde con tu experiencia cotidiana con el español.

Veamos también el perfil de trigramas del artículo “Apple" en inglés:

```
" th": 381   "ppl": 246   " in": 206   " ap": 191
"the": 355   " an": 217   " of": 205   "nd ": 187
"he ": 288   "ed ": 214   "and": 201   "es ": 186
"ple": 254   "app": 206   "ing": 197      ...
```

Y el de “Epal” en malayo (en malayo, epal significa manzana):

```
"an ": 443   " da": 182   "kan": 166   " ep": 133
"ang": 223   "epa": 180   "al ": 155   "eng": 130
"ng ": 216   " me": 180   " se": 141   "men": 120
"pal": 185   " di": 167   "nga": 135      ...
```

¿Notas que las frecuencias relativas de los perfiles de los diferentes lenguajes son muy distintos? Esto significa que podemos identificar el lenguaje de un texto siguiendo dos sencillos pasos: primero obtenemos su perfil de trigramas, y luego determinamos el lenguaje al cual el perfil más se asemeja.

Unicode

Antes de escribir nuestro programa, debemos definir exactamente qué es un carácter.

En C y C++ estamos acostumbrados a utilizar el tipo de datos `char` como carácter, pero en el mundo existen muchísimos sistemas de escritura. Los $2^8 = 256$ posibles valores de un byte no son, de hecho, suficientes para abarcar toda la variedad de posibles caracteres.

La versión 15.0 del estándar Unicode, por ejemplo, define 149,186 caracteres de 161 sistemas de escritura y varios conjuntos de símbolos (mira la figura 1).

AaBbCc (latino), ΑαΒβΓγ (griego), АаБбВв (cirílico), ԱԲԳe (armenio), אבג (hebreo), آءؤ (árabe), ܒܓ (siríaco), ऄअआ (devanagari), অআই (bengalí), ਅਆਇ (gurmuji), ଅଆଇ (oriya), அஆஇ (tamil), అఆఇ (telugú), กขฃ (tailandés), ກຂຄ (laosiano), ༀཀྐྐཱཿ (tibetano), ကခဂ (birmano), ႠႡႢ (georgiano), ㄱㄲㄴ (hangul), ሀሁሂ (etíope), ᎣᎤᎥ (cherokee), ᠠᠡᠢ (mongol), ⠅⠣⠭ (braille), ⲗⲂⲄ (copto), 豈更車 (chino), ぁあぃ (japonés hiragana), ァアィ (japonés katakana), †⁋⁒ (puntuación general), €£₡ (divisas), ℃℅ℋ (similares a letras), ⅒⅔⅗ (formas numéricas), →↲⇈ (flechas), ∀∂∃ (operadores matemáticos), ⌀⌖⌘ (símbolos técnicos), ⒶⒷⒸ (alfanuméricos delimitados), ┌╌╌┐ (diseño de cajas), ▒▓▙ (elementos de bloque), ▸◈◒ (formas geométricas),

Fig. 1: Algunos caracteres de Unicode

Afortunadamente existe el estándar UTF-8, que permite codificar los caracteres Unicode en secuencias de bytes. En particular, codifica cada carácter Unicode en grupos de entre uno y cuatro bytes.

A programar

Empieza a partir del starter code que te damos junto con el enunciado.

Parte 1: Construye el perfil de trigramas

Tu primer tarea consiste en editar `Lequel.cpp` y completar la función:

```
TrigramProfile buildTrigramProfile(const Text &text)
```

Esta función recibe como parámetro una variable de tipo `Text` (una lista de líneas de texto). Puedes encontrar su definición en el archivo `Text.h`.

La función debe devolver un perfil de trigramas `TrigramProfile`: un mapa de trigramas (`string`) a frecuencias (`float`). La definición de `TrigramProfile` se encuentra en `Lequel.h`.

Para decodificar el UTF-8 que está contenido en el `string` de cada línea de texto, aprovecha el código comentado en `buildTrigramProfile`. El `wstring` resultante contiene en cada posición un carácter Unicode decodificado. `wstring` se utiliza igual que `string`.

El `wstring` construido a partir del `string` de cada línea de texto te permite obtener los trigramas, que debes almacenar en otro `wstring`. Para convertir este segundo `wstring` en un `string` (ya que `TrigramProfile` almacena los trigramas como `string`), utiliza el código comentado en `buildTrigramProfile`.

Si una línea de texto tiene menos de tres caracteres, ignórala.

Te preguntarás por qué almacenamos las frecuencias con `float` y no con `int`. Ocurre que a continuación normalizaremos las frecuencias para que tomen valores entre `0.0F` y `1.0F`.

Parte 2: Normaliza las frecuencias

Ahora tienes un `TrigramProfile` que contiene el perfil de trigramas del texto.

Para que el perfil sea independiente del largo del texto, debes normalizar sus frecuencias. Tomando prestada una técnica del álgebra lineal, normalizaremos las frecuencias de la siguiente manera:

- **Suma los cuadrados de todas las frecuencias.**
- **Divide cada frecuencia por la raíz cuadrada de este número.**

Veamos un ejemplo. Supongamos que tenemos el siguiente perfil de trigramas:

```
"aaa": 3
"bbb": 1
"ccc": 1
```

Primero calculamos el número $3^2 + 1^2 + 1^2 = 11$, y luego dividimos cada frecuencia por la raíz cuadrada de este número. Obtenemos:

```
"aaa": 0.904534
"bbb": 0.301511
"ccc": 0.301511
```

Ahora puedes editar `Lequel.cpp` y completar la función:

```
void normalizeTrigramProfile(TrigramProfile &trigramProfile)
```

Truco: para modificar los valores del `TrigramProfile`, escribe el `for` de la siguiente manera:

```
for (auto &element : trigramProfile)
{…}
```

La referencia `&element` te permitirá modificar el elemento in-situ.

Parte 3: Implementa la similitud coseno

Ahora llegamos al paso en el que determinamos cuán parecido es el perfil del texto al perfil de un lenguaje.

Para ello utilizaremos la similitud coseno, que se calcula de la siguiente forma:

- **Para cada trigrama presente tanto en el perfil del texto (perfil P_T) como en el perfil del lenguaje (perfil P_L), multiplicamos las frecuencias de ese trigrama en P_T y P_L, y sumamos todos los números encontrados de esta manera.**

Veamos un ejemplo. Supongamos que tenemos el siguiente perfil de trigramas:

```
Perfil 1:
  "aaa": 0.333
  "bbb": 0.667
  "ccc": 0.667

Perfil 2:
  "bbb": 0.333
  "ccc": 0.667
  "ddd": 0.667
```

Ambos perfiles tienen en común los trigramas `"bbb"` y `"ccc"`. La similitud coseno es por tanto:

```
(Producto de las frecuencias "bbb") + (producto de las frecuencias "ccc")
= (0.667 * 0.333) + (0.667 * 0.667)
= 0.667
```

Las frecuencias de `"aaa"` y `"ddd"` no son relevantes porque estos trigramas no aparecen en ambos perfiles.

Edita `Lequel.cpp` y completa la función:

```
float getCosineSimilarity(TrigramProfile &textProfile, TrigramProfile &languageProfile)
```

Parte 4: Identifica el lenguaje de un texto

Sólo falta unir todo el trabajo que hiciste en las partes anteriores.

En `Lequel.cpp`, completa la función:

```
string identifyLanguage(const Text &text, LanguageProfiles &languageProfiles)
```

En `languageProfiles` recibes una lista de elementos de tipo `LanguageProfile` (definido en `Lequel.h`).

Cada elemento de tipo `LanguageProfile` contiene un `languageCode` (una cadena que identifica el lenguaje), y el perfil de trigramas del lenguaje.

Lo que debes hacer en `identifyLanguage` es calcular el perfil de trigramas de tu texto, normalizar sus frecuencias, y luego iterar sobre todos los lenguajes que recibes en `languageProfiles`.

Para cada lenguaje calcula la similitud coseno entre el perfil del texto y el perfil del lenguaje. Guarda el `languageCode` que produce máxima similitud coseno.

En el `return`, devuelve el `languageCode` de máxima similitud coseno.

Parte 5: Explora y evalúa

¡Acabas de implementar un sistema de identificación de lenguaje! Ahora debes preguntarte: ¿Qué tan bien funciona? ¿Cuándo acierta? ¿Cuándo falla? Estas preguntas son importantes en vista de la importancia política y cultural de los lenguajes.

Para ello considera las preguntas en el archivo `README.md` (del starter code). Respóndelas en el mismo archivo.

Debes entregar este archivo junto a tu código.

Tips

- **Usa las pruebas.** Te proveemos muchas pruebas que puedes usar en cada etapa de tu desarrollo para saber si estás haciendo las cosas bien.
- **Usa las referencias de C++ (con &) siempre que sea posible.** Evita hacer copias innecesarias que enlentezcan tu código.
- **Pregúntanos.** Si hay algo que no comprendes, pregunta inmediatamente y no pierdas el tiempo. ¡Esto es especialmente importante en este trabajo!

Bonus Points

Más cosas para hacer:

- **Aprende más sobre la similitud coseno.** Lee el artículo Understanding cosine similarity and its application.
- **Si el texto a analizar es muy grande, se producirá un cuello de botella computacional.** ¿Cuál es y cómo podría resolverse?
- **Añade más lenguajes.** Crea otro target, usa la función `getTextFromFile` (de `Text.h`) para leer un corpus (un documento de texto grande en cierto lenguaje), procésalo en una función `buildLanguageProfile` y escríbelo a disco con `writeCSV` (`CSVData.h`). Modifica el archivo `languagecode_names_es.csv` y añade tus perfiles a la carpeta `trigrams`. Algunos idiomas que puedes añadir: guaraní, catalán, asturiano, e incluso ¡lenguaje C, lenguaje C++ y Python!