Programación II

Trabajo práctico final: Texto Predictivo

Agustín López

Facultad de Ciencias Exactas Ingeniería y Agrimensura

1 Lectura y procesamiento de archivos en C

El objetivo principal del programa escrito en C se centró en dos tareas principales:

1. **Leer archivos de la persona proporcionada como argumento:**
   En primer lugar, el programa ejecuta el comando `ls Textos<nombre>.txt`, almacenando su salida en un archivo auxiliar `archivo.txt`. Esta acción tiene como fin conocer los nombres y la cantidad de textos escritos por la persona. Posteriormente, emplea un bucle while para leer línea por línea el archivo, procesando cada texto de manera individual en cada iteración.

2. **Agregar la entrada sanitizada al archivo `Entradas/<nombre>.txt`:**
   En cada iteración del bucle, se ejecuta la función `agregar_entrada`, la cual incorpora al archivo `Entradas/<nombre>.txt` el contenido sanitizado del texto actual.

Una vez finalizado el bucle, se invoca al programa escrito en Python, el cual se encuentra listo para operar con el archivo `Entradas/<nombre>.txt`.

En el programa en C, no hubo decisiones particularmente relevantes, ya que como el objetivo era bastante claro y directo, no logré encontrar muchas alternativas para abordarlo.

2 Algoritmo de predicción en Python

El programa en Python se diseñó con el propósito principal de trabajar con tres archivos distintos: la entrada, las frases incompletas y la salida.

1. **Carga inicial de datos:**
   La primera acción consistió en cargar las oraciones del archivo de entrada y las frases incompletas en dos listas de strings separadas, donde cada string es una linea del archivo.

2. **Lógica central en la función `completar_frases`:**
   Esta función posee la lógica central y la decisión más desafiante del programa: el algoritmo para determinar la palabra más probable. Después de investigar, el algoritmo que me pareció mejor para abordar este problema fue el modelo de bigramas. El algoritmo comienza procesando el texto de entrada y lo convierte en un diccionario. En este diccionario, las claves son tuplas de todos los pares de palabras consecutivas, y el valor es la cantidad de ocurrencias de esas palabras consecutivas. Por ejemplo:

   quiza no sea el vino
   quiza no sea el postre
   quiza no sea
   no sea nada

   Quedaria representado como:

   ```
   {("quiza", "no"): 3, ("no", "sea"): 4, ("sea", "el"): 2,
   ("el", "vino"): 1, ("el", "postre"): 1, ("sea", "nada"): 1}
   ```

   Luego, el programa trabaja con el archivo de frases incompletas línea por línea, identifica las palabras anteriores y posteriores al guion bajo para compararlas con el diccionario de bigramas y utiliza un diccionario donde las claves son las palabras candidatas y los valores son números enteros que representan la posibilidad de ser la palabra correcta. Si una palabra candidata aparece despues de la palabra anterior al guión bajo o antes de la palabra siguiente al guión bajo se le asigna 1 punto por cada ocurrencia de esto. Si una palabra candidata aparece entre la palabra anterior y la posterior al guion bajo, se le asigna 10 puntos ya que las probabilidades de que sea la palabra correcta son mucho mas altas. Y por ultimo, si no hay suficiente información para predecir la palabra de esta manera, el programa selecciona la palabra que aparece con mayor frecuencia en el texto de entrada.

   Por ejemplo veamos que pasa con estas lineas siguiendo el ejemplo anterior:

```
esta manera _ no sea la muerte
maldito sea _ guru
```

En el primer caso, el algoritmo busca las tuplas donde "manera" es la primera componente y "no" es la segunda. Encuentra que "no" aparece 3 veces después de "quiza", por lo que asigna 3 puntos de posibilidad. Esta palabra se elige finalmente.

De manera similar, en el segundo ejemplo, busca las tuplas donde "sea" es la primera componente y encuentra que después de "sea", "el" aparece 2 veces y "nada" 1 vez. Con esta información, asigna puntajes de 2 para "el" y 1 para "nada". La palabra elegida es "el" por tener el puntaje más alto.

```
esta manera quiza no sea la muerte
maldito sea el guru
```

3. **Alternativas:**
   Encontrar el algoritmo que consideré adecuado resultó ser más complicado de lo esperado y fue la parte que más tiempo me llevó. Mientras buscaba soluciones, consideré algunas ideas que no desarrollé porque noté que quizás iba a complicar innecesariamente el programa.

   Por ejemplo, pensé en crear listas de las palabras anteriores y posteriores a aquella que se quería predecir, y luego compararlas de alguna manera con el archivo de entrada. Sin embargo, para obtener palabras candidatas, hubiera necesitado analizar prácticamente cada una de las palabras del archivo de entrada. También consideré inicialmente el uso de trigramas, donde la palabra candidata sería la del medio de la tríada, pero pronto encontré dificultades para asignar puntajes a las palabras candidatas. Aunque me pareció que habría alguna manera de hacerlo de esa manera, decidí replantearlo utilizando bigramas. Finalmente, consideré que fue una decisión acertada, ya que el enfoque con bigramas me resultó más intuitivo en la implementación del algoritmo.