Reporte tarea 5 - Programación y algoritmos

Erick Salvador Alvarez Valencia

Centro de Investigación en Matemáticas, A.C.,
erick.alvarez@cimat.mx

Resumen En el presente reporte se analizará las modificaciones que se realizaron al analizador de texto de la tarea anterior. El enfoque fue mejorar la sintaxis del código haciendo más legible al mismo, esto se logra haciendo uso de estructuras, las cuales permiten almacenar un conjunto de datos, haciendo su tratamiento más fácil. Otra modificación importante en el analizador fue una serie de mátricas de calidad que fueron añadidas, esto anterior nos permite evaluar el qué tan buena fue la clasificación realizada. Finalmente se analizarán una serie de correcciones que se hicieron a unos códigos propuestos para la parte dos de la tarea.

Keywords: Lenguaje C, clasificador, texto, metrica, calidad, estructuras.

1. Descripción

Partimos del programa desarrollado en la tarea anterior, el cual realiza una clasificación de palabras en base a qué clase corresponden, al final, el algoritmo genera una lista de bloques, la cual se obtiene dividiendo los libros de dos autores diferentes. Dicha lista describe el número de bloque, la clase original del bloque y, la clase a la cual fue clasificada el bloque, el siguiente paso es aplicar una métrica para saber qué tan buena o mala fue la clasificación, ya que por sí sola, la lista no nos dice mucho.
Existen una serie de métricas que nos ayudan a lo anterior, todas ellas trabajan con la cantidad de positivos, negativos, verdaderos positivos y verdaderos negativos que fueron obtenidos en el análisis y clasificación.

Para trabajar más cómodamente se definió una estructura, la cual contiene todo lo necesario para realizar la clasificación, de esta forma podemos tener un código más ordenado y más facil de escalar. Dicha estructura también contiene un puntero a función, el cual se puede apuntar hacia alguna de las funciones que evaluan la calidad. Con esto anterior logramos de una manera sencilla y rápida la elección de la métrica que más se ajuste a lo que queremos.

Otro punto a destacar es que se modificó el código de la función *separate*, ahora genera archivos de 5000 palabras y no caracteres, por lo mismo se producirán menos archivos a la hora de generar la clasificación.

La estructura generada se llama $FOBJ$ y contiene las matrices generadas de las clases 1 y 2, los conjuntos de prueba y entrenamiento de cada clase, así como variables que indican la cardinalidad de los conjuntos anteriores y una puntero a una función el cual puede apuntar a cualquiera de las funciones que generan la métrica de calidad. El algoritmo que se usó para generar un nuevo objeto es el siguiente:

Algorithm 1 Algoritmo para generar un nuevo objeto de tipo $FOBJ$.

1: **procedure** MemObjAlloc($FrecC1, FrecC2, TryingSet1, TryingSet2, F1sz, F2sz, NoWords$)
2: $newObj \leftarrow$ Puntero de tipo $FOBJ$.
3: $newObj \leftarrow$ Dar memoria al objeto.
4: $newObj \Rightarrow FrecC1 \leftarrow CopiarMatriz(FrecC1)$.
5: $newObj \Rightarrow FrecC2 \leftarrow CopiarMatriz(FrecC2)$.
6: $newObj \Rightarrow TryingSet1 \leftarrow CopiarVector(TryingSet1)$.
7: $newObj \Rightarrow TryingSet2 \leftarrow CopiarVector(TryingSet2)$.
8: $newObj \Rightarrow F1sz \leftarrow F1sz$.
9: $newObj \Rightarrow F2sz \leftarrow F2sz$.
10: $newObj \Rightarrow NoWords \leftarrow NoWords$.
11: $newObj \Rightarrow QualityMetric \leftarrow FuncAccuracy$.
12: **return** $newObj$

El código generado para clonar un objeto ya existente es muy parecido al código anterior, solamente que los datos de origen provienen de un objeto de origen. Hay que tener en cuenta que la copia de las matrices y vectores se haga por valor y no por referencia, ya que esto prodría generar problemas posteriores.
De la misma forma el código de la función que libera un objeto se muestra a continuación:

Algorithm 2 Algoritmo para liberar la memoria de un objeto de tipo $FOBJ$.

1: **procedure** FreeObjMem(Obj)
2: **if** $Obj \Rightarrow FrecC1$!= $NULL$ **then**
3: $BorrarMatriz(Obj \Rightarrow FrecC1)$.
4: **if** $Obj \Rightarrow FrecC2$!= $NULL$ **then**
5: $BorrarMatriz(Obj \Rightarrow FrecC2)$.
6: **if** $Obj \Rightarrow TryingSet1$!= $NULL$ **then**
7: $BorrarVector(Obj \Rightarrow TryingSet1)$.
8: **if** $Obj \Rightarrow TryingSet2$!= $NULL$ **then**
9: $BorrarVector(Obj \Rightarrow TryingSet2)$.
10: $newObj \Rightarrow F1sz \leftarrow 0$.
11: $newObj \Rightarrow F2sz \leftarrow 0$.
12: $newObj \Rightarrow NoWords \leftarrow 0$.
13: $newObj \Rightarrow QualityMetric \leftarrow NULL$.
14: **return** Obj

Las funciones que generan la métrica de calidad en base al resultado de la clasificación son muy sencillas de implementar, todas ellas trabajan con los valores obtenidos por la clasificación. De cualquier forma se muestra el pseudocódigo de una de ellas. Se usará la métrica *Sensitivity*.

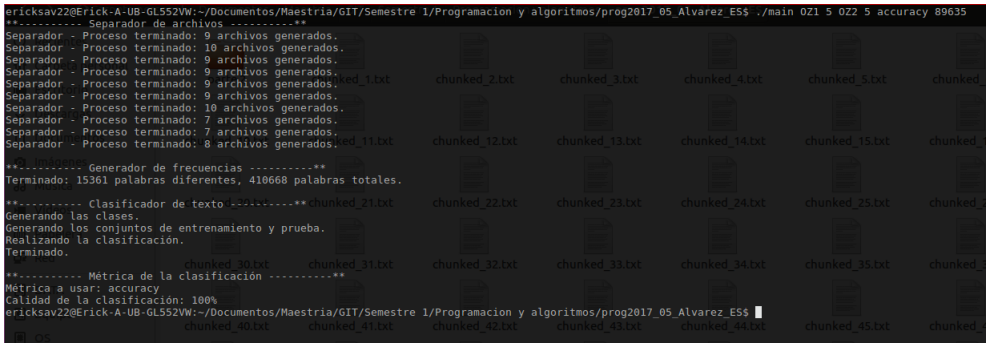
Algorithm 3 Algoritmo que genera la métrica de calidad *Sensitivity*.

1: **procedure** SensitivityMetric(P, N, TP, TN)
2: $FN \leftarrow N - TN$.
3: $metric \leftarrow TP/(TP + FN)$.
4: **return** $metric$

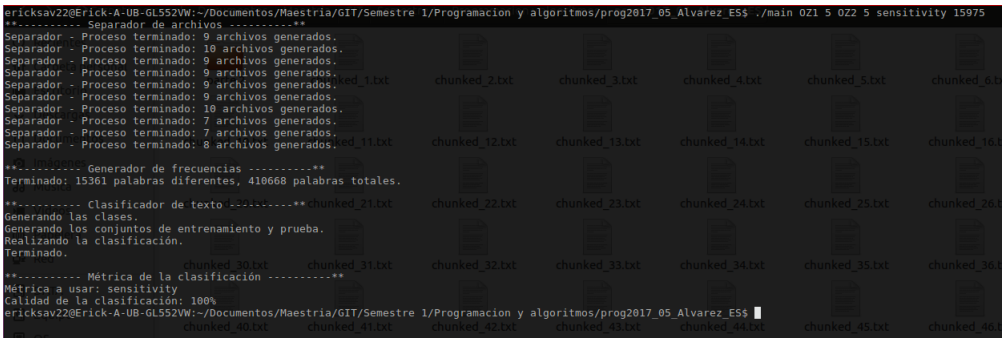
Así como esa se hicieron las demás funciones, las cuales reciben la misma cantidad de parámetricos, esto para que al usarlas con el puntero del objeto, no presentara errores por diferentes parámetros.

2. Descripción de las ejecuciones

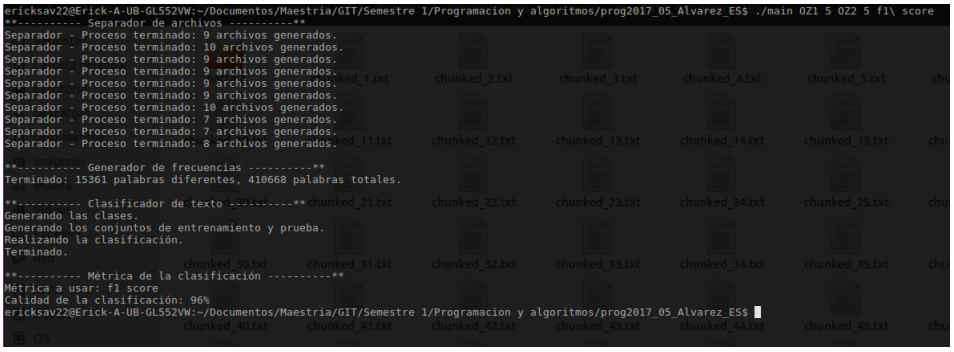
Se realizaron varias ejecuciones del programa, dentro de las cuales el enfoque fue el análisis de los valores P, T, TP y TN generados, así como su correcta implementación en las métricas de calidad. Se pudo comprobar que dichos valores coincidían con los resultados obtenidos en el archivo $clasified.txt$ que genera el software. A continuación se muestran algunos de los resultados de las ejecuciones empleando diferentes métricas de calidad y diferentes semillas:



(a) Figura 1. Resultado de la ejecución del programa usando la métrica *accuracy*, así como una semilla predefinida.



(b) Figura 2. Resultado de la ejecución del programa usando la métrica *sensitivity*, así como una semilla predefinida.



(c) Figura 3. Resultado de la ejecución del programa usando la métrica $f1score$, así como una semilla aleatoria.

Cabe mencionar que también se realizaron ejecuciones bajo el comando de *valgrind* y se demostró que el programa trabaja adecuadamente con la memoria solicitada para las estructuras y las matrices, toda ella la libera.

3. Conclusiones del programa

Queda concluir el programa funcionó según lo esperado, obtuvo los valores P, T, TP y TN de forma correcta, por lo cual pudo brindar una función de métrica de calidad adecuada.

De la misma forma, las funciones que trabajan con la estructura hicieron lo esperado, y pese a que no se utilizó la función *clone_obj*, esta quedó implementada para usarla en cualquier parte del programa que se necesite.

Una mejora que se puede proponer al programa es guardar los valores P, T, TP y TN por si se necesitan posterior a la clasificación y la aplicación de la métrica.

4. Parte 2 - Corrección de los programas

Los códigos relacionados con los programas que se describirán fueron incluidos en la carpeta '02', así como el archivo README.txt asociado a ellos que muestra como compilarlos y ejecutarlos.

1. **Programa 1:** Este programa se compone de 3 archivos, podemos darnos cuenta que en el archivo *apuntadores.h* se está haciendo la definición de la función, pero también se hace la implementación de la misma. Aunque C permite esto anterior, no es lo que se acostumbra hacer. Lo correcto sería hacer sólo la definición en el archivo *.h* y la implementación en el *.c* para tener un código más ordenado. Otro error que se presenta es en la función $imprimeVector$, la cual debe recibir un apuntador a enteros, pero en cambio, recibe un entero, lo cual genera un error de compilación. Un tercer error se presenta al momento de llamar a la función $printf$ pero no se está incluyendo la librería *stdio.h*

2. **Programa 2:** En este programa encontraremos los siguientes errores:

   a) En el archivo *apuntadores.c*, se quiere imprimir algunos datos con la función $printf$, sin embargo no se está importando la librería *stdio.h*

   b) En el último $printf$ de la función swap, se quiere imprimir las direcciones de memoria de las variables x e y, sin embargo al usar los corchetes se está imprimiendo su valor en la posición cero, lo cual genera una incompatibilidad.

   c) La función *smax* recibe la copia de las variables que se pasan en el *main* más no la dirección de memoria de las mismas, por lo que si se pretende hacer un swap, esto no resultará.

   d) La función *swap* al hacer el intercambio en las variables, lo hace como direcciones de memoria, además de hacer un $x = y$ y posteriormente un $y = x$, hay que notar que no se está usando la variable auxiliar. Lo que generará que no se produzca el intercambio de manera correcta.

   e) En la función *smax* no se cumplirá la condicion de acuerdo a como se están pasando los valores de las variables.

3. **Programa 3:** En este programa encontraremos los siguientes errores:

a) Se está pidiendo memoria para un arreglo de nueve enteros, pero al llamar a las funciones $llenaVector$ y $calculaModa$ se está pasando un tamaño de diez, lo que puede producir errores de acceso a la memoria.

b) En la función $llenaVector$ se quiere imprimir la direccioń de memoria de cada valor x_i, y para ello no se necesita hacer el casting a entero.

c) No es necesario generar un vector de tipo *double* si sólamente se guardarán las repeticiones de cada valor aleatorio calculado, con un vector de enteros es suficiente.

d) En la función $calculaModa$ se está retornando la frecuencia del elemento y no la moda en sí.

4. **Programa 4:** Los errores de este programa son:
   a) Se quiere pedir memoria para una matriz de *char* mediante la función $umatriz$ pero la variable mat no es un puntero de punteros a *char*, lo cual provocará un error de conversión.

   b) En el archivo $apuntadores.c$ se están usando las funciones para pedir y liberar memoria, pero no se incluyó la librería $stdlib.h$.

   c) En la función $freeMat$ se quiere liberar memoria para una matríz, pero se está pasando como argumento un puntero a *char*, se esperaría que se pase un puntero de punteros, esto genera un error de conversión.

   d) En la función $umatriz$ se quiere pedir memoria para crear una matríz de *char*, pero se está creando una variable que es sólo un puntero a *char* y no un puntero de punteros. De la misma forma cuando se pide memoria para cada elemento mat_i no se está haciendo casting de lo que regresa la función $malloc$.

   e) Un error importante es que al pedir memoria para la matríz, se le está diciendo a la función $malloc$ que reserve para un tamaño de un char sin signo por el número de filas, cuando debería ser el tamaño de un apuntador a char sin signo por el número de filas.

   f) No se está liberando memoria en la función $main$ de la matriz que fue creada.

5. **Programa 5:** Para este programa sólo fueron implementadas las funciones para requerir y liberar memoria de los objetos $DMATRIZ$ y $DVECTOR$, así como la función que realiza una multiplicación matriz - vector.