Informe proyecto Organización Computadoras UNS

Comisión 21

Integrantes: Feuilles Juan Cruz Lu:100598

Delgado Romina Soledad Lu:104218

Objetivos:

El objetivo principal del proyecto es desarrollar una aplicación en lenguaje C que cumpla con los siguientes requerimientos, evaluar las cantidad de apariciones de palabras dentro de archivos de tipo texto “.txt” mediante la implementación los TDA Lista, Multiset, y un CuentaPalabras

Implementaciones

TDA Lista:

El tda lista es una lista simplemente enlazada, sin centinela.Donde cada celda está implementada de manera que contenga un elemento (struct elemento) y la celda siguiente, siendo elemento un par (int,char*) es decir que contenga un entero y una cadena de caracteres.

Contiene las siguiente funciones:

lista t *lista_crear() Crea una lista vacía y la devuelve.

int lista_insertar(lista t *l, elemento t elem, unsigned int pos) Inserta el elemento elem en la posición pos de la lista.

elemento t *lista_eliminar(lista t *l, unsigned int pos) Elimina el elemento de la posición pos de la lista.

elemento t *lista_elemento(lista t *l, unsigned int pos) Devuelve un puntero al elemento que ocupa la posición pos de la lista.

int lista_ordenar(lista t *l, funcion comparacion t comparar) Dada la lista l y la función comparar ordena la lista de acuerdo al criterio de dicha función.

unsigned int lista_cantidad(lista t *l) Devuelve la cantidad de elementos de la lista l.

int lista vacia(list t lista) Devuelve verdadero (≠ 0) si la lista está vacía, y falso(= 0) si la lista contiene al menos un elemento.

TDA Multiset:

Es una colección sin orden establecido que acepta elementos repetidos.Implementado mediante un árbol Trie, donde cada elemento contiene una cantidad de apariciones (palabra) y un puntero a un arreglo de árboles siguientes, que el arreglo contenga veintiséis elementos (siguientes[26]) no es casual ya que que cada lugar representa una letra del alfabeto respectivamente, y que una posición esté libre significa que la letra correspondiente no esté, esto se calcula como que la posición en el arreglos es igual a pos=int(char)-97.

Ejemplo siguientes[0]!=NULL significa “a” esta presente
siguientes[26]!=NULL significa “z” esta presente
obs:siempre se usan minúsculas.

Contiene las siguientes funciones:

multiset t *multiset crear() Crea un multiset vacío de palabras y lo devuelve.

void multiset insertar(multiset t *m, char *s) Inserta la palabra s al multiset m.

int multiset cantidad(multiset t *m, char *s) Devuelve la cantidad de repeticiones de la palabra s en el multiset m.

lista_t multiset elementos(multiset t *m, int (*f)(elemento t,elemento t)) Devuelve una lista de tipo lista_t con todos los elementos del multiset m y la cantidad de apariciones de cada uno. Como no era necesario que la lista esté ordenada se quitó de los parámetros tanto del .c como el .h la función **int (*f)(elemento t,elemento t)** que se usaba para establecer la prioridad del orden en caso de requerirlo.

void insertarPreOrdenAux(multiset_t *m, lista_t *l, char *palabra, int nivel) Funcion auxiliar para recorrer el multiset almacenando palabras en la lista l, siguiendo el patrón del PreOrden por ser esta la manera de recorrer un árbol más eficiente a criterio personal.

void multiset eliminar(multiset t **m) Elimina el multiset m liberando el espacio de memoria reservado. Luego de la invocación m debe valer NULL. Para esta función se implementaron dos funciones auxiliares:

void eliminar_aux(multiset_t *m,void(destruir_elemento(void*))) Hace un recorrido del multiset invocando la función destruir elemento a elemento.
void destruir_elemento(void*e) Libera el espacio ocupado por el elemento e.

CuentaPalabras:

Es la aplicación en si, no un TDA, la idea general es que dado un directorio que contenga archivos de texto .txt, se los analice según su contenido de palabras y su cantidad de apariciones, en general se aprovecha el manejo de elementos repetidos que ofrece Multiset permitiendo tener un contador de apariciones y usando el ordenar de lista, a partir de eso se generan dos archivos **"cadauno.out.txt"** y **"totales.out.txt"** que también se muestran en pantalla para que el usuario pueda comprobarlos directamente, no se mezclaron los archivos de entrada y salida en un mismo directorio para no generar errores y que se tomen archivos de salida como entradas durante la ejecución o reejecucion de la aplicación.

Contiene las siguientes funciones:

comparacion_resultado_t funcion_comparacion(elemento_t *elem1, elemento_t *elem2) Compara dos elementos **elm1** y **elm2**, recordar que el elemento es un par (int, char*) siendo "int" cantidad de apariciones y "char*" palabra, como primera prioridad se establece la cantidad de apariciones, si es la misma se considera el orden alfabético de las palabras a través de la función "**strcmp**" perteneciente a la librería string.h

void lista_destruir(lista_t*l) Utiliza de manera iterativa el eliminar del lista para eliminar y liberar el espacio de todo lo que contenga la lista l.

void help() Muestra un mensaje para guiar al usuario en el uso de la aplicacion.

void nombres_archivos(char * c,lista_t* l) Accede al directorio de entrada c y almacena los nombres de los archivos .txt en la lista l.

void lista_a_archivo(FILE* salida,lista_t *l) Pasa el contenido de la lista l al archivo salida.

void mostrar_arc(FILE* f1) Muestra en pantalla el contenido de un archivo.

void salida_total(FILE* f1,lista_t* nombres,char*dir,char*dir2) A partir de la lista nombres, que contiene los nombres de los archivos de entrada, el directorio dir, se copia en f1 las palabras de todos los archivos de entrada ordenadas según su su frecuencia de aparición o alfabéticamente como corresponda, dir2 es un auxiliar para no perder dir en el proceso de creación de f1 (totales.out.txt)

void salida_cada_uno(FILE* f1, lista_t* nombres,char*dir,char* dir2) A partir de la lista nombres y los directorios, almacena en f1(cadauno.out.txt) las palabras ordenadas según apariciones/alfabeto pero con la salvedad de separando el orden de cada archivo.

void menu() Muestra el menú de operación de la aplicación, dando tres opciones, presionado “0” se sale, con presionar “1” se ejecuta el orden de cada archivo por separado , y con “2” el orden de todos los archivos juntos.

void programa(char* directorio) A partir del directorio de entrada, programa maneja la dinámica de la aplicación ,llevando a cabo cada operación que especifique el usuario.

int main(int argc, char* argv[]) Recibe los parámetros de entrada, optando por ejecutar lo que el usuario necesite o ayuda en caso de considerarlo necesario según las especificaciones del proyecto.

Consideraciones:

Palabra: se considera a toda cadena de caracteres que contenga el alfabeto en minúscula sin la letra “ñ”.

Lista: Por no contar con un destruir, se implementó uno por fuera del TDA dentro de cuentapalabras.c para ser usado dentro del mismo cuando sea necesario, para hacer más legible el código,de manera similar se agrega una función comparación necesaria para ordenar las listas usadas.

Multiset: Se quitó la función **int(*f)(elemento t,elemento t)** pasada por parámetros en **multiset_elementos(...)** porque era necesario solo en caso que la salida de **multiset_elementos** sea ordenada, la cátedra quitó ese requisito pero no modificó de header del enunciado por lo que para hacer menos engorroso el uso de multiset_elementos se quitó.

CuentaPalabras: Para los archivos de salida no se tomó en cuenta el directorio de entrada proporcionado por el usuario, dado que a nuestro criterio no se especificaba la ruta de salida en el enunciado, consideramos usar la ruta por defecto del compilador que es la misma donde se almacena el programa generado. Tampoco se mencionaba nada de mostrar los archivos generados por pantalla, si bien esto fue una función de testeo decidió dejarse dado que suma a la hora de chequear el contenido de salida y hace más amigable la experiencia con el usuario, pero en caso de molestar simplemente se borra la línea o se comenta la línea.

Guia usuario:

Identificar donde se encuentra cuentapalabras.exe.
Accede con comando cd si se encuentra en SO windows
ejemplo:

```
C:\Users\Juani>cd C:\Users\Juani\Documents\WorkSpaceCodeBlock\ProyectoOc1\cuentapalabras\bin\Debug

C:\Users\Juani\Documents\WorkSpaceCodeBlock\ProyectoOc1\cuentapalabras\bin\Debug>cuentapalabras.exe
```

Cuando se encuentre en cuentapalabras.exe espacio por medio copie los parámetros de entrada del programa “-h” opcional es la ayuda y el directorio de los archivos de texto a procesar

ejemplo:

```
bug>cuentapalabras.exe -h C:\Users\Juani\Documents\WorkSpaceCodeBlock\ProyectoOc1\cuentapalabras\archivos
```

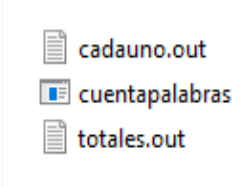
Lo próximo en ver será.

```
----------///*Bienvenido-a-la-Guia-de-usuario*////----------


El siguiente programa permite analisis palabras almacenadas en archivos dados por el usuario.
Recuerde que debe proporcionar directorio donde se encuentren los archivos a analizar.
Ejemplo: cuentapalabras.exe h- directorio,  h- es opcional y directorio apunta a la carpeta donde almacena los archivos.
Cuando ingrese un directorio utilice barras de separador ejemplo c:\disco\archivos



 Para salir ingrese 0


 Para analizar palabras por archivo separados ingrese 1


 Para analizar palabras de todos los archivos ingrese 2
```

Si ejecuta opción 1 o 2 verá en pantalla lo siguiente

```
1
 1  Archivo1.txt
 1  tres
 2  dos
 2  uno
 1  Archivo2.txt
 1  dos
 1  tres
 2  treinta
 1  Archivo3.txt
 1  doce
 1  ocho
 1  tres
 2  treinta
 1  Archivo4.txt
 1  dos
 1  treinta
 2  ocho

Para salir ingrese 0


Para analizar palabras por archivo separados ingrese 1


Para analizar palabras de todos los archivos ingrese 2

2
 1  doce
 2  uno
 3  ocho
 3  tres
 4  dos
 5  treinta

Para salir ingrese 0
```

En caso de necesitar los archivos de salida luego ingresar a la opción 1 y 2 aparecerán los siguientes archivos en la misma carpeta donde está la aplicación.



Con 0 se sale de la aplicación

```
0


 La aplicacion se ejecuto con exito!!!

C:\Users\Juani\Documents\WorkSpaceCodeBlock\ProyectoOc1\cuentapalabras\bin\Debug>
```