PROYECTO
DE
PROGRAMACIÓN EN LENGUAJE C

Profesor: Leonardo De Mateis
Asistente: Gabriela Díaz

UNIVERSIDAD NACIONAL DEL SUR

10/10/2017

Giampietri Gonzalo – Zárate Tomás

1. Definición y especificación de requerimientos.

El desarrollo del proyecto consiste en implementar un programa que registre la frecuencia de aparición de palabras que forman parte de un archivo de texto.

El objetivo principal del proyecto, es implementar un programa que registre la frecuencia de aparición de palabras que forman parte de un archivo de texto.

Con el objetivo de cumplir los requerimientos, se implementó todo el sistema en lenguaje C: un programa principal y el siguiente conjunto de estructuras de datos: TDA Lista, TDA Lista Ordenada, TDA Trie

Aspectos técnicos y tecnologías empleadas.

Lenguaje/s y herramientas de Programación utilizado/S:

- C: C es un lenguaje de programación de propósito general e imperativo. Soporta programación estructurada y recursión.
- Compilador: gcc (GNU Compiler Collection).

Entorno de Desarrollo: Distribución GNU/Linux Devuan 32 bits.

Estructuras implementadas y utilizadas:

- **TDA Lista**: Lista cuyos elementos son punteros genéricos. Lista enlazada implementada sin centinelas.
- **TDA Lisa Ordenada**: Lista ordenada cuyos elementos son punteros genéricos. El orden de los elementos en la lista se especifica al momento de la creación, a través de una función de comparación.
- **TDA Trie**: El Trie es un árbol compuesto por nodos. Cada nodo tiene como rótulo un carácter y un contador de tipo entero. El trie mantiene referencia a un nodo raíz, considerando que cada nodo del árbol Trie como una estructura que mantiene referencia al padre y a una lista ordenada de nodos como hijos.

Libreas externas utilizadas:

- **string.h:** ésta librearía define un tipo de variable, una macro, y varias funciones para la manipulación de arreglos de caracteres.
- **stdio.h:** ésta librería define tres tipos de variables, algunas macros, y un conjunto de funciones para manipular la entrada y salida de datos.
- **stdlib.h:** ésta librería define cuatro tipos de variables, algunas macros, y un conjunto de funciones para implementar funciones generales.

2. Procesos y servicios ofrecidos por el sistema.

Programa Principal – evaluador

Evaluador contabiliza la cantidad de apariciones de cada palabra en un archivo de texto.

El programa se invoca a través de la consola. Recibe como argumento, por línea de comandos, el nombre de un archivo de texto compuesto por todo tipo de caracteres. Evaluador se invoca de la siguiente manera:

$evaluador <archivo_texto>

Operaciones ofrecidas por *evaluador*

Evaluador ofrece un menú de operaciones donde el usuario puede escoger que operación realizar. Las operaciones se cuentan enumeras del 1 al 6, de la siguiente manera:

1. Mostrar palabras: permite visualizar el listado de todas las palabras que posee el archivo, junto con la cantidad de apariciones de la misma.
2. Consular: permite determinar si una palabra ingresada pertenece o no al archivo, y en consecuencia, cuántas veces esta se repite en el archivo.
3. Comienzan con: permite consultar cuántas palabras comienzan con una letra dada.
4. Es prefijo: permite consultar si una palabra, ingresada por el usuario, es prefijo de otras almacenadas en el trie.
5. Porcentaje prefijo: dado un prefijo, indica el porcentaje de palabras del trie que comienzan con él.
6. Salir: permite salir del programa.

3. Documentación técnica

Operaciones TDA Lista

- TLista Crear_lista(): Crea y retorna una lista vacía.
- Int l_insertar(TLista l, TPosicion pos, TElemento e): Agrega un elemento a la lista l en la posición anterior a pos. Si pos es nulo, inserta a e en la primera posición. Retorna verdadero si procede con éxito, falso en caso contrario.
- Int l_eliminar(TLista l, TPosicion pos): Elimina un elemento en la posición pos de la lista. Retorna verdadero si procede con éxito, falso en caso contrario. Si la posición no es válida retorna LST_POS_INV.
- TElemento l_recuperar(TLista lista, TPosicion pos): Retorna el elemento correspondiente a la posición pos. Si la posición es POS_NULA, retorna ELE_NULO.

- TPosición l_ultima(TLista l): Retorna la última posición de la lista l. Si la lista está vacía retorna POS_NULA.
- TPosicion l_anterior(TLista l, TPosicion pos): Retorna la posición anterior a pos en la lista l. Si pos es la primera posición, retorna POS_NULA.
- TPosicion l_siguiente(TLista l, TPosicion pos): Retorna la posición siguiente a pos en la lista l.
- TPosicion l_primera (TLista l, TPosicion pos): Retorna la primera posición de la lista. Si la lista está vacía retorna POS_NULA.
- Int l_size(TLista l): Retorna la cantidad de elemento de la lista l.

Operaciones TDA Lista Ordenada

- TListaOrdenada Crear_lista_Ordenada(int (*f) (void *, void *): Crea y retorna una lista ordenda vacía. El orden de los elementos en la lista se especifica al momento de la creación, a través de una función de comparación int (*f) (void *, void *). La función f devuelve -1 si el orden del primer elemento es menor al orden del segundo, 0 si el orden es el mismo, y 1 si el orden del primer argumento es mayor que el orden del segundo.
- Int lo_insertar(TListaOrdenada l, TElemento e): Agrega un elemento elem a la lista l, de modo que quede siempre ordenada de forma ascendente. Retorna verdadero si procede con éxito, falso en caso contrario.
- Int lo_eliminar(TListaOrdenada l, TPosicion pos): Elimina el elemento en la posición pos de la lista. Retorna verdadero si procede con éxito, falso en caso contrario.
- TPosición lo_ultima(TListaOrdenada l): Retorna la última posición de la lista l.
- TPosicion lo_primera(TListaOrdenada l): Retorna la primera posición de la lista l.
- Int lo_size(TListaOrdenada l): Retorna la cantidad de elemento de la lista l.
- TPosicion lo_siguiente(TListaOrdenada lista, TPosicion pos): Retorna la posición siguiente a pos en la lista.

Operaciones TDA Trie

- TTrie crear_trie(): Retorna un nuevo trie vacío, esto es, con nodo raíz que mantiene rótulo nulo y contador en cero.
- Int tr_insertar(TTrie tr, char* str): Inserta el string str al trie, inicializando el valor de contador asociado en uno. En caso de que el string ya se encuentre en el trie, aumenta el valor del contador asociado a dicho string en una unidad. Retorna verdaderos si la inserción ha sido exitosa, y falso en caso contrario.
- Int tr_pertenece(TTrie tr, char* str): Retorna verdadero el string str pasado pertenece al trie, falso en caso contrario.

- Int tr_recuperar(TTrie tr, char* str): Retorna el entero asociado al string str, dentro del trie. Si el string no pertenece al trie, retorna STR_NO_PER.
- Int tr_size(TTrie tr): Retorna la cantidad de palabras del trie tr.
- Int tr_eliminar (TTrie tr, char* str): Elimina el string str dentro del trie, liberando la memoria utilizada. Retorna verdadero en caso de operación exitosa, y falso en caso contrario.

Funciones Auxiliares

TDA Trie

- TNodo tr_recuperarHijo_auxiliar(TNodo padre, char* s): Buscar y retorna el nodo hijo de un nodo padre pasado por parámetro que tenga como rótulo el primer caracter del string s. Si no lo encuentra, retorna NULL

- int tr_pertenece_auxiliar(TNodo padre, char* s,int longitud, TTrie tr): Busca al hijo del nodo padre pasado por paŕametro cuyo rótulo coincida con el primer carácter del string s. Si lo encuentra, sigue buscando recursivamente utilizando el mismo método, caso contrario retorna FALSE.
  Si llega a un nodo hoja y su rótulo coincide con el último carácter del string s retorna TRUE, y FALSE en caso contrario.

- TNodo tr_buscar_auxiliar(TTrie tr,char* str): Busca y retorna el nodo que contiene el ultimo caracter de str, si la palabra está en el trie tr. Si no lo encuentra retorna NULL.

- TPosicion tr_recuperarPos_auxiliar(TNodo nodo): Busca y retorna la posición del nodo en la lista de hijos de su padre.

- int tr_eliminar_auxiliar(TTrie tr, TNodo ultimo, int long original , int longitud): Se elimina la palabra cuyo último carácter es rótulo del nodo último pasado como parámero. Retorna TRUE si se ha eliminado la palabra con exito, FALSE en caso contrario. En el caso de que la palabra esté repetida o sea prefijo de otra, se descuenta el contador en 1 y se retorna TRUE.

Evaluador

- int ev_toLowerCase_auxiliar(int c): Retorna el entero correspondiente al código ASCII del caracter pasado por parámetro en minúscula. Se asume que previamente se controlo que el caracter c fuera una letra mayúscula entre A y Z.

- void ev_vaciar_auxiliar(char c[]): Vacía el arreglo de caracteres pasado por parámetro.

- int ev_porcentaje_auxiliar(TNodo nodo): Recorre el trie hacia abajo a través de los hijos del nodo pasado por por parámetro contando la cantidad de palabras de la cual es prefija la palabra que se pasa por parámetro en la función cascar "prefijos".

- TNodo ev_recuperarHijo_auxiliar(TNodo padre, char* s): Buscar y retorna el nodo hijo de un nodo padre pasado por parámetro que tenga como rótulo el primer caracter de s. Si no lo encuentra, retorna NULL

- TNodo ev_buscar_auxiliar(TTrie tr,char* str): Busca y retorna el nodo que contiene el ultimo caracter de str, si la palabra está en el trie. Si no lo encuentra retorna NULL.

- void ev_imprimir_auxiliar(TTrie tr , char c[]): Imprime la palabra pasada por parámetro, indicando cuánta veces se encuentra en el Trie tr.

- void ev_mostrar_auxiliar(TTrie tr,char buffer[], TNodo nodo): Recorre el Trie tr y concatena los rotulos de los nodos para formas las palabras que se encuentran en el Trie tr, para luego imprimirlas.

- int ev_comienzaCon_auxiliar(TNodo nodito,char c): Recorre el trie hacia abajo a través de los hijos del nodo nodito pasado por parámetro, contando cuantas palabras comienzan con el carácter c.

- void ev_liberar_auxiliar(TTrie tr, TNodo nodo): Libera recursivamente los espacios de memoria correspondiente a cada estructura.

4. Alcance y limitaciones

1. Se considera como palabra válida toda secuencia de caracteres tal que todos sus caracteres estén en el siguiente rango: a..z y A..Z.
2. Las opciones disponibles en el menú se distribuyen del número 1 al 6.
3. El programa finaliza si el usuario es ingresa un carácter no numérico en el menú de opciones con el siguiente mensaje: [RESPETE LAS OPCIONES. SOLO NÚMEROS DIFERENTES A 0. SE CIERRA EL PROGRAMA]
4. La especificación de un archivo de texto, al invocar al programa evaluador, es obligatoria. Si no se especifica el programa despliega el siguiente mensaje: < NO SE HA INGRESADO EL NOMBRE DE ARCHIVO >. Luego se cierra.
5. Si no se especifica un archivo de texto en la invocación del programa, éste muestra el mensaje: < NO ESE HA ENCONTRADO EL ARCHIVO >. Luego se cierra.
6. La invocación del programa especificando un archivo cuyo contenido es vacío no afecta a la funcionalidad del mismo.
7. Se considera que una palabra no es prefija de sí misma.
8. Se incorporaron y definieron las siguientes constantes especificadas para el desarrollo del proyecto:

Constante	*Valor*	*Significado*
FALSE	0	Valor lógico falso.
TRUE	1	Valor lógico verdadero.
LST_NO_INI	2	Intento de acceso inválido sobre lista sin inicializar.
LST_POS_INV	3	Intento de acceso a posición inválida en lista.
LST_VAC	4	Intento de acceso inválido sobre lista vacía.
TRI_NO_INI	5	Intento de acceso inválido sobre Trie sin inicializar.
STR_NO_PER	-1	String no perteneciente a trie.
POS_NULA	NULL	Posición nula.
ELE_NULO	NULL	Elemento nulo.

5. Conclusiones

Si bien en la especificación de requerimientos dados para el desarrollo del proyecto se establece que toda palabra es una secuencia de caracteres S (tal que S =< c1 , . . . , cn >, n > 0 , dónde ci ∈ {a,. . . ,z} ∪ {á, é, í, ó, ú}, ∀i(0 < i <= n)) , se decidió adoptar la política de sólo considerar a una palabra como una secuencia de caracteres que se encuentran entre a…z (A..Z). Esto se debe a complicaciones encontradas en la lectura y comparación de caracteres en el lenguaje C, ya que éste sólo reconoce aquellos caracteres pertenecientes al código ASCII original (de Estados Unidos).