Organización de computadoras

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Segundo Cuatrimestre de 2017

Proyecto N° 1
Programación en lenguaje C

Propósito

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

Con este objetivo, se debe implementar:

- El TDA Lista Ordenada, que almacene elementos de tipo genérico, y ordene los mismos a través de un comparador diseñado para tales elementos.
- El TDA Trie, que almacena cadena de caracteres y un contador asociado a cada cadena, y que permite recuperar fácilmente tanto palabras como su correspondiente contador.
- Un programa principal, el cual debe tomar como parámetro de entrada un archivo de texto, recorrer el mismo, calcular la frecuencia de aparición de las palabras que lo componen y ofrecer un conjunto de operaciones de consulta y manipulación del trie.

1. TDA Lista Ordenada

Implementar un TDA Lista Ordenada en lenguaje C, cuyos elementos sean punteros genéricos. El orden de los elementos en la lista se especifica al momento de la creación, a través de una función de comparación. La lista debe ser simplemente enlazada sin centinelas. La implementación debe proveer las operaciones:

1. `lista_t crear_lista(int (*f)(void*,void*))` Crea y retorna una lista vacía. El orden de los elementos insertados estará dado por la función de comparación `int f(void*,void*)`. Se considera que la función `f` devuelve -1 si el orden del primer argumento es menor que el 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.

2. `int insertar(lista_t lista, void* elem)` Agrega el elemento `elem` en la posición correspondiente de la lista, de modo que la misma quede siempre ordenada. Retorna verdadero si procede con éxito, falso en caso contrario.

3. `int eliminar(lista_t lista, void* pos)` Elimina el elemento en la posición `pos`. Reacomoda la lista adecuadamente al eliminar en posiciones intermedias. Retorna verdadero si procede con éxito, falso en caso contrario. Si la posición no es válida, aborta con *exit status* `LST_POS_INV`.

4. `int size(lista_t lista)` Retorna la cantidad de elementos de la lista. Si la lista no está inicializada, el programa aborta con *exit status* `LST_NO_INI`.

5. `void* primer_pos(lista_t lista)` Retorna la primer posición de la lista. Si la lista es vacía, aborta con *exit status* `LST_VAC`.

6. `void* ultima_pos(lista_t lista)` Retorna la última posición de la lista. Si la lista es vacía, aborta con *exit status* `LST_VAC`.

7. `void* siguiente(lista_t lista, void* pos)` Retorna la posición siguiente a `pos` en la lista. Si `pos` es la última posición de la lista, o una posición inválida, aborta con *exit status* `LST_POS_INV`.

8. `void* buscar(lista_t lista, void* elem)` Busca y retorna la primer posición que contenga al elemento `elem` en la lista. En caso de que el elemento no exista en la lista, retorna `NULL`.

9. `void* recuperar(lista_t lista, void* pos)` Retorna un puntero al elemento en la posición `pos` de la lista. Si la posición no es válida, aborta con *exit status* `LST_POS_INV`.

10. `int destruir(lista_t lista)` Libera la memoria ocupada por la lista y le asigna `NULL`. Retorna verdadero en caso de éxito, falso en caso contrario. Si la lista no está inicializada, aborta con *exit status* `LST_NO_INI`.

Para la implementación, se debe considerar que los tipos `lista_t` y `celda_t` están definidos de la siguiente manera:

```
typedef struct lista_ordenada {
  unsigned int cantidad_elementos;
  celda_t primera_celda;
} * lista_t;

typedef struct celda {
  void * elemento;
  struct celda * proxima_celda;
} * celda_t;
```

2. TDA Trie

Implementar un TDA Trie en lenguaje C, cuyos nodos tienen como rótulo un *caracter* (`char`), y almacena adicionalmente un contador de tipo *entero* (`int`). El trie debe implementarse manteniendo referencia a un nodo raíz, y considerando cada nodo del árbol como una estructura que mantiene referencia al padre y una lista ordenada de nodos como hijos. La implementación debe proveer las operaciones:

1. `trie_t crear_trie()` Retorna un nuevo trie vacío, esto es, con nodo raíz que mantiene rótulo nulo y contador en cero.

2. `int insertar(trie_t tr, char* str)` Inserta el string `str` en el trie `tr`, inicializando el valor de contador asociado en uno. En caso de que el string ya se encuentre representado en el trie, aumenta el valor del contador asociado a dicho string en una unidad. Retorna verdadero si la inserción (o actualización) fue exitosa, falso en caso contrario.

3. `int pertenece(trie_t tr, char* str)` Retorna verdadero si el string `srt` pertenece al trie `tr`, falso en caso contrario.

4. `int recuperar(trie_t tr, char* str)` Retorna el entero asociado al string `str`, dentro del trie `tr`. Si el string no pertenece al trie, retorna STR_NO_PER.

5. `int size(trie_t tr)` Retorna la cantidad de palabras almacenadas en el trie `tr`.

6. `int eliminar(trie_t tr, char* str)` Elimina el string `str` dentro del trie `tr`, liberando la memoria utilizada. Retorna verdadero en caso de operación exitosa, y falso en caso contrario.

7. `int eliminar(trie_t tr)` Elimina todos los strings dentro del trie `tr`, liberando la memoria utilizada, quedando el trie vacío. Retorna verdadero en caso de operación exitosa, y falso en caso contrario.

Para la implementación, se deben considerar que los tipos `trie_t` y `nodo_t` están definidos de la siguiente manera:

```
typedef struct trie {
  nodo_t raiz;
} * trie_t;

typedef struct nodo {
  char rotulo;
  unsigned int contador;
  struct nodo * padre;
  lista_t hijos;
} * nodo_t;
```

3. Programa Principal

Implementar una aplicación de consola que le permita al usuario especificar un archivo de texto, compuesto por todo tipo de caracteres, a partir del cual se debe contabilizar la cantidad de apariciones de cada palabra en el archivo. Se considerá como palabra, a toda secuencia de caracteres S, tal que:

$$S =< c_1, \ldots, c_n >, n > 0$$
$$c_i \in \{\text{a,}\ldots\text{,z}\} \cup \{\text{á,é,í,ó,ú}\}, \forall i(0 < i <= n).$$

El programa debe ofrecer un menú de operaciones, con las que el usuario puede consultar y manipular el estado del trie:

1. Iniciar: permite la creación de un nuevo trie, a partir del ingreso de un nombre de archivo de texto.

2. Mostrar palabras: permite visualizar el listado de todas las palabras junto con la cantidad de apariciones de la misma.

3. Consultar: permite determinar si una dada palabra ingresada pertenece o no al archivo, y en consecuencia, cuántas veces esta se repite en el archivo.

4. Comienzan con: permite consultar cuántas palabras comienzan con una letra dada.
5. Es prefijo: permite consultar si una palabra ingresada es prefijo de otras almacenadas en el trie.
6. Porcentaje prefijo: dado un prefijo, indicar el porcentaje de palabras del trie que comienzan con él.
7. Salir: permite salir del programa.

Sobre la implementación

- Los archivos fuente principales se deben denominar **lista.c, trie.c y main.c** respectivamente. En el caso de las librerías, también se deben adjuntar los respectivos archivos de encabezados **lista.h y trie.h**, los cuales han de ser incluidos en los archivos fuente de los programas que hagan uso de las mismas.
- Es importante que durante la implementación del proyecto se haga un uso cuidadoso y eficiente de la memoria, tanto para la reservar (`malloc`), como para liberar (`free`) el espacio asociado a variables y estructuras.
- Se deben respetar con exactitud los nombres de tipos y encabezados de funciones especificados en el enunciado. Los proyectos que no cumplan esta condición quedarán automáticamente desaprobados.
- La compilación debe realizarse con el *flag* `-Wall` habilitado. El código debe compilar **sin advertencias** de ningún tipo.
- La copia o plagio del proyecto es una falta grave. Quien incurra en estos actos de deshonestidad académica, desaprobará automáticamente el proyecto.

Sobre el estilo de programación

- El código implementado debe reflejar la aplicación de las técnicas de programación modular estudiadas a lo largo de la carrera.
- En el código, entre eficiencia y claridad, se debe optar por la claridad. Toda decisión en este sentido debe constar en la documentación que acompaña al programa implementado.
- El código debe estar indentado, comentado, y debe reflejar el uso adecuado de nombres significativos para la definición de variables, funciones y parámetros.

Sobre la documentación

Los proyectos que no incluyan documentación estarán automáticamente desaprobados. La misma debe:

- Estar dirigida a usuarios finales y desarrolladores.

- Explicar detalladamente los programas realizados, incluyendo el diseño de la aplicación y el modelo de datos utilizado, así como toda decisión de diseño tomada, y toda observación que se considere pertinente.
- Incluir explicación de todas las funciones implementadas, indicando su prototipo y el uso de los parámetros de entrada y de salida (tanto dentro del código fuente como en la documentación del proyecto). Se espera que la explicación no sea una mera copia del código fuente, sino más bien una síntesis de lo implementado a través de diagramas, pseudocódigos, o cualquier representación que considere adecuada.
- En general, se deben respetar todas las consignas indicadas en la "Guía para la documentación de proyectos de software" entregada por la cátedra.

Sobre la entrega

Toda comisión que no cumpla con los requerimientos, estará automáticamente desaprobada. Los mismos son:

- Las comisiones estarán conformadas por 2 alumnos, y serán las que oportunamente registró y notificó la cátedra.
- La entrega del código fuente y la documentación se realizará a través de un archivo comprimido **zip** o **rar**, denominado ***PR1-Apellido1-Apellido2***, que debe incluir las siguientes carpetas:
  - **Fuentes**, donde se deben incorporar los archivos fuente ".c" y ".h" (ningún otro).
  - **Documentación**, donde se debe incorporar el informe del proyecto en formato PDF (ningún otro).
- El archivo comprimido debe enviarse por e-mail, respetando el siguiente formato:
  - **Para:** *federico.joaquin@cs.uns.edu.ar*
  - **Asunto:** *OC :: PR1 :: COM XX :: Apellido1 - Apellido2*
  - **Cuerpo del e-mail:**
    Se adjunta Proyecto Nº 1, de la comisión XX:
    Apellido, Nombre 1 - LU 1
    Apellido, Nombre 2 - LU 2
- El e-mail debe ser enviado con anterioridad al día **Martes 10 de Octubre de 2017**, a las **22:00 hs**. Se considerará como hora de ingreso, la registrada en el servidor de e-mail del DCIC.

Sobre la corrección

- La cátedra evaluará tanto el **diseño** e **implementación** como la **documentación** y **presentación** del proyecto, y el cumpliemiento de **todas** las condiciones de entrega.
- Tanto para compilar el proyecto, como para verificar su funcionamiento, se utilizará la máquina virtual "OCUNS" publicada en el sitio web de la cátedra.