UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Estructura de Datos y Análisis de Algoritmos

Experiencia 2: Listas enlazadas

Andrés Felipe Muñoz Bravo
19.646.487-5

Profesor:
Roberto Gonzales Ibañes

Santiago - Chile
2016

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

Se ha pedido realizar un programa de codificado muy parecido al código Cesar, la diferencia radica en que el alfabeto utilizado para la codificación está compuesto solamente por las letras que componen al texto. Estos textos son leídos por el programa a través de un archivo de texto, los cuales pasan por una serie de algoritmos para luego entregar el resultado correspondiente (codificado o decodificado).

Para llevar a cabo este programa se ha trabajado con listas enlazadas circulares, utilizando el lenguaje de programación C, con el paradigma de programación imperativo procedural.

CAPÍTULO 2. MARCO TEORICO

TDA: Los tipos de datos abstractos, son conjunto de datos a los cuales se le asocian operaciones, como por ejemplo de TDA están las listas circulares doblemente enlazadas. En este laboratorio esta lista guardara en cada posición un carácter. Ejemplo de operaciones asociadas a este TDA son, agregar un elemento a la lista, remover un elemento de la lista, etc.

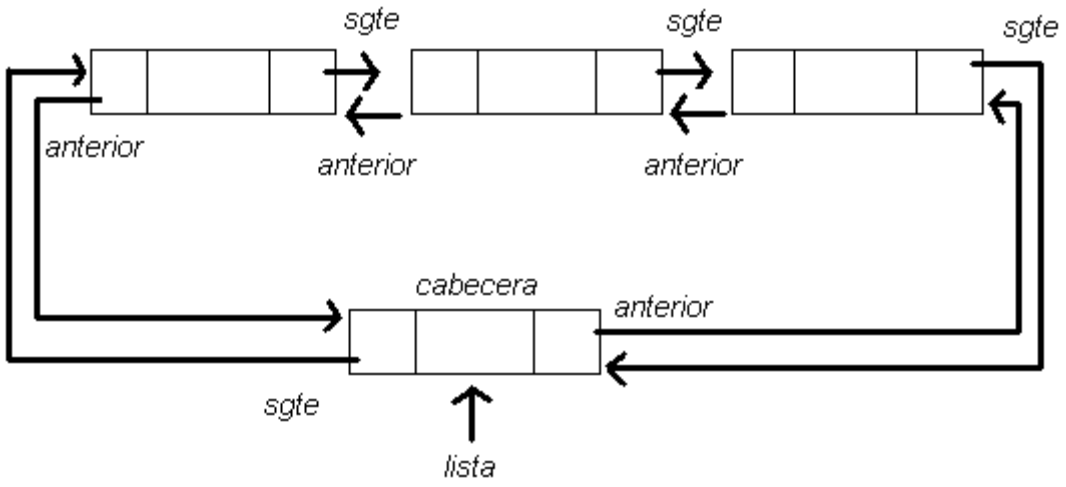

Ilustración 1: Lista circular doblemente enlazada.

CAPÍTULO 3. DESCRIPCIÓN DE LA SOLUCIÓN

Para el desarrollo de este programa es utilizada la herramienta sublime text 3, el cual es un editor de texto. La principal ventaja de este editor es que utiliza colores para las diferentes palabras reservadas del lenguaje de programación C, de esta forma ayuda al programador revisar la sintaxis programada. A continuación se explicara detalladamente las estructuras y las principales funciones del programa realizado.

2.1 TECNICAS

2.1.1 Ordenamiento de la lista de caracteres:

Para ordenar una lista se utilizó el algoritmo de ordenamiento “bubblesort”, debido a que su implementación es muy fácil. Este es sin duda el peor algoritmo de ordenamiento dentro de los más conocidos.

2.1.2 Listas:

La lista implementada es una lista circular doblemente enlazada, algunas funciones implementadas para el manejo de estas son:

- Add: Para agregar un elemento al final de la lista
- Find: Para buscar una letra dentro de la lista
- getLetter: Para obtener una letra de la lista, según una posición.
- showListFL: Para mostrar el contenido de la lista.
- moveLeft: Para mover todos los datos de la lista una posición a la izquierda.
- listcpy: Para copiar una lista.

2.2 ESTRUCTURAS

Para el programa se utilizan dos estructuras (Nodo y List) que se muestran a continuación:

```
typedef struct Nodo
{
    char letter;
    struct Nodo* next;
    struct Nodo* back;
}Nodo;

typedef struct List
{
    int length;
    Nodo* first;
    Nodo* last;

}List;
```

Ilustración 2:Estructuras

Estas estructuras nos ayudan a implementar el TDA lista circular doblemente enlazada.

Dentro de la estructura de nodo se encuentran definido 3 datos, primero un carácter letter, luego un puntero a un nodo siguiente, y finalmente un puntero a un nodo anterior.

Por otro lado se encuentra la estructura List la cual representara una lista, dentro de ella se encuentra el largo (cantidad de nodos en su interior), un puntero al primer nodo de la lista, y un puntero al último nodo de la lista.

2.3 FUNCIONES

2.3.1 **Codificar:**

int encode(): Esta función es utilizada para codificar un texto de un archivo de texto, para ello se realizan los siguientes pasos.

- Leer el archivo de texto y guardarlo dentro de un string
- Crear 4 listas :
    - Lista que contiene el texto donde cada posición es un carácter de él.
    - Lista que contiene las letras del texto sin repetir y ordenadas alfabéticamente
    - Lista que contiene los mismos datos de la lista anterior.
    - Lista para guardar el texto codificado.
- Recorrer la primera lista, buscar la letra en la segunda lista y luego obtener la posición, se agrega a la cuarta lista la letra contenida en la tercera lista en la posición encontrada.
- Formar un string con la lista del texto codificado para posteriormente depositarlo en un archivo de texto.

2.3.2 Decodificar:

Int decode(): Esta función es utilizada para decodificar un texto de un archivo de texto, para ellos se procede a utilizar casi el mismo algoritmo anterior.

- Leer el archivo de texto y guardarlo dentro de un string junto con el alfabeto y el desfase.
- Crear 4 listas :
    - 1) Lista que contiene el texto a decodificar, donde cada posición es un carácter de él.
    - 2) Lista que contiene las letras del alfabeto sin repetir y ordenadas alfabéticamente.
    - 3) Lista que contiene los mismos datos de la lista anterior.
    - 4) Lista para guardar el texto decodificado.
- Recorrer la primera lista, buscar la letra en la tercera lista y luego obtener la posición, se agrega a la cuarta lista la letra contenida en la segunda lista en la posición encontrada.
- Formar un string con la lista del texto codificado para posteriormente depositarlo en un archivo de texto.

CAPÍTULO 4. ANÁLISIS DE LOS RESULTADOS

A continuación se muestran todas las funciones del programa con su respectivo orden:

Función	T(n)	Orden
createNodo	6	O(1)
createList	5	O(1)
add	12	O(1)
invert	13n+7	O(n)
delete	n+9	O(n)
deleteList	n^2+10n+8	O(n^2)
deleteSpaceLast	n^2+9n	O(n^2)
find	2+2n	O(n)
findPosition	5+2n	O(n)
getLetter	2+2n	O(n)
isVowel	n^2+12n+74	O(n^2)
isConsonant	n^2+12n+269	O(n^2)
showListFL	2n+5	O(n)
showListLF	2n+5	O(n)
moveLeft	2n+3	O(n)
moveRight	2n+3	O(n)
stringToList	14n+7	O(n)
listToString	3n+6	O(n)
createListEncode	7n^2+18n+17	O(n^2)
listcpy	13n+7	O(n)
compareList	2n+2	O(n)
encode	n^3+30^2+201n+103	O(n^3)
decode	3n^3+57n^2+397n+127	O(n^3)
showTitle	6	O(1)
bubbleSort	5n^2+3n+2	O(n^2)
pause	3	O(1)

Tabla 1: Tiempo y orden de funciones.

Como se puede apreciar las funciones con mayor orden son encode y decode. Estas funciones son las principales del programa anteriormente descritas con su respectivo algoritmo. Analizando estas funciones, se llega a la conclusión que se puede pasar de O(n^3) a O(n^2), si se cambian las funciones para verificar si una letra es consonante o vocal, ya que las funciones implementadas en este programa para hacer dicha verificación tienen O(n^2), las cuales son utilizadas dentro de un bucle dentro de las funciones encode y decode.

Al hacer el cambio anteriormente descrito, el programa mejoraría su eficiencia en gran medida al momento de codificar o decodificar un texto muy grande.

CAPÍTULO 5. CONCLUSIÓN

En esta segunda entrega se cumplió el objetivo completamente, y se aprendió a implementar y utilizar listas en el lenguaje de programación C. Al igual que el laboratorio anterior, lo importante fue pensar desde un inicio la solución para poder implementarla sin ningún problema. En general no se presentaron grandes problemas.

Una de las falencias del programa es que no funciona perfectamente para textos muy grandes (con más de 50000 caracteres), debido a que lee una cierta cantidad caracteres por línea. Esto se podría mejorar en el futuro, haciendo dinámica esa lectura dependiendo de la cantidad de caracteres en la línea a leer.

Otra cosa que se podría mejorar es que el usuario ingrese los nombres de los archivos a trabajar, lo cual el programa debiera primero revisar si es posible trabajar con ellos, para luego proceder con la plena ejecución y entrega de resultados.

CAPÍTULO 6. REFERENCIAS

Departamento de Ingeniería Informática USACH. (2016). Enunciado Laboratorio 2. 2016: USACH.

DCC Universidad de Chile (2007). *Tipos de datos abstractos*. (2016) https://users.dcc.uchile.cl/~bebustos/apuntes/cc30a/TDA/.