Práctica 20

Datos generales:

Nombre de la Práctica	Listas ligadas
Nombre de la carrera	Ingeniería de Software
Nombre de la materia	Estructuras de Datos
Número y nombre de Unidad(es) temática(s)	III. Estructuras lineales.
Docente que imparte la materia	Aldonso Becerra Sánchez
Fecha de entrega para los alumnos	7-cotubre-2021
Fecha de entrega con extensión y penalización	8-octubre-2021
Fecha de elaboración	6-octubre-2021

Objetivo de la Práctica	Profundizar con las operaciones en las listas enlazadas.
Tiempo aproximado de realización	3 horas
Introducción	La memoria dinámica es un elemento importante en el manejo de información abundante donde no se sabe de antemano cuantos datos son los requeridos, por tanto solventa las limitaciones de la memoria estática. Las listas enlazadas permiten la manipulación de la memoria dinámica a través de la liga de nodos sucesivos.

Referencias que debe consultar el alumno (si se requieren):

Referencia 1:

1. Cairo, Osvaldo; Guardati, Silvia. Estructura de Datos, Tercera Edición. McGraw-Hill, México, Tercera Edición, 2006.

Referencia 2:

2. Mark Allen Weiss. Estructura de datos en Java. Ed. Addison Wesley.

Referencia 3:

3. Joyanes Aguilar, Luis. Fundamentos de Programación. Algoritmos y Estructuras de Datos. Tercera Edición, 2003. McGraw – Hill.

Actividades que debe realizar el alumno:

Actividad inicial:

Generar el reporte en formato IDC.

Actividad 1:

Primero genere la **Introducción**.

Actividad 2:

Realizar la clase ListaEncadenadaHash (lista diccionario) utilizando la clase NodoHash:

| clave | valor | dirMemDer |

NodoHash

donde clave es un valor de referencia utilizado para realizar las operaciones sobre la lista (como si fuera una llame primaria o identificador único), y valor es el contenido real del nodo (el dato); dirMemDer es el campo que indica el nodo que viene por delante en la lista.

De acuerdo a esto debe definir los siguientes métodos:

a) boolean insertar(Object clave, Object valor). Este método inserta los datos al final (se permiten duplicados en el campo de valor, pero NO en el campo de clave). Si la clave ya se encuentra en la lista, el valor anterior será substituido por el nuevo valor con la misma clave.

b) Object eliminar(Object clave). Se usa el campo de clave para borrar el elemento. Este método regresa el contenido real del campo (valor). Regresa null si no se localiza.

c) Object eliminarValor(Object valor). Se usa el campo de valor para borrar el elemento. Este método regresa el contenido real del campo (valor). Regresa null si no se localiza.

d) Object buscar(Object clave). Se utiliza para buscar un elemento ubicado por su clave. El método regresa el contenido (valor) en caso que se encuentre, null en caso contrario.

e) Object buscarValor(Object valor). Se utiliza para buscar un elemento ubicado por su valor. El método regresa el contenido (valor) en caso que se encuentre, null en caso contrario.

f) boolean substituir(Object clave, Object valorNuevo). Substituye un elemento de la lista localizado mediante la clave por el nuevo valor. Regresa un booleano si pudo hacerlo.

f) boolean substituirValor(Object valor, Object valorNuevo). Substituye un elemento de la lista localizado mediante la valor por el nuevo valor. Regresa un booleano si pudo hacerlo.

g) void imprimir(). Imprime la lista diccionario completo.

h) void imprimirClaves(): Imprime las claves.

i) void imprimirValores(): Imprime los valores.

j) ListaLigada aArreglos(): Regresa la lista diccionario como una lista que contiene dos arreglos, un arreglo es la lista de claves, y el otro es un arreglo que tiene la lista de valores de la lista diccionario.

k) ListaLigada aListas(): Regresa la lista diccionario como una lista que contiene dos listas ligadas, una es la lista de claves, y la otra es una lista que tiene los elementos con los valores.

l) Tabla2D aTabla(): Regresa una matriz con la estructura de la lista diccionario.

clave1 valor1

clave 2 valor2

……..

m) void vaciar(): Vacía el diccionario.

n) Object obtener(Object clave): Obtiene el valor de la clave especificada.

o) boolean vacia(): Indica si el diccionario está vacío.

p) boolean agregarLista(ListaEncadenadaHash lista2): Agrega todos los elementos de la lista pasada como argumento a la lista actual. Las claves no deben repetirse.

q) int cantidadElementos(): Regresa el tamaño de la lista.

r) boolean agregarArreglos(ArregloDatos arregloClaves, ArregloDatos arregloValores): Agrega todos los elementos del arreglo pasado como argumento, donde hay arreglos paralelos de clave y valor respectivamente. Las claves no deben repetirse.

s) boolean agregarListas(ListaEncadenada listaClaves, ListaEncadenada listaValores): Agrega todos los elementos de la lista pasada como argumento, donde hay listas paralelas de clave y valor respectivamente. Las claves no deben repetirse.

t) boolean agregarTabla(Tabla2D tabla): Agrega todos los elementos de la matriz pasada como argumento, donde la primera columna tiene las claves y a segunda columna tiene los valores. Las claves no deben repetirse y debe validar las dimensiones de la matriz.

Haga el programa (actividad 2, la cual es el **Desarrollo** del programa, junto con la captura de pantalla del programa funcionando).

Actividad 3:

Pruebe el funcionamiento del programa de la actividad 2 con todo y sus capturas de pantalla.

Actividad 4:

Realice la sección de **Código agregado** (diagrama de clases UML).

Actividad 5:

Realice la sección de **Pre-evaluación** (use los lineamientos establecidos).

Actividad 6:

Finalmente haga las **Conclusiones**.

Actividad 7:

Enviar en http://ingsoftware.reduaz.mx/moodle

Archivo anexo que se requiere para esta tarea (opcional):

Dudas o comentarios: a7donso@gmail.com