**CARÁTULA**

**ALUMNO:** ESQUIVEL GRADOS LUIS GERMÁN

**CÓDIGO**: 17200154

**FACULTAD**: FISI

**ESCUELA**: INGENIERÍA DE SISTEMAS

**CURSO**: ALGORÍTMICA III

**TEMA**: Algoritmos Voraces Y Conjuntos

**DOCENTE**: AUGUSTO CORTEZ VÁSQUEZ

**FECHA DE PRESENTACIÓN**: 02/05/2019

El TAD Conjunto creado está basado en el TAD Lista Enlazada.

El TAD Lista Enlazada se muestra a continuación:

|  |
| --- |
| Lista Enlazada |
| Atributos |
| -Nodo<T> cab |
| -Nodo<T> ult |
| Métodos |
| +getCab(); |
| +getUlt(); |
| +setCab(Nodo<T> cab); |
| +setUlt(Nodo<T> ult); |
| +agregar(T info); |
| +listar(); |
| +recuperar(); |
| +salvar(); |
| +buscar(T infoBuscada); |
| +eliminar(T info); |

Descripción de los métodos:

getCab():

Devuelve el primer nodo de la lista

getUlt():

Devuelve el último nodo de la lista

setCab(Nodo<T> cab):

Establece el primer nodo de la lista

setUlt(Nodo<T> ult):

Establece el último nodo de la lista

agregar(T info):

Agrega un elemento al final de la lista.

listar():

Devuelve una cadena que contiene la información de los nodos de la lista.

recuperar(String archivo):

Recupera en la lista el archivo cuyo nombre se pase por parámetro.

salvar(String archivo):

Salva la lista en el archivo cuyo nombre se pase por parámetro.

buscar(T infoBuscada):

Devuelve el nodo que tiene la información buscada

eliminar(T info):

Elimina el nodo que tiene la información buscada

Con esta información, podemos pasar a definir el TAD Conjunto:

TAD conjunto extiende de Lista Enlazada:

|  |
| --- |
| Conjunto |
| Atributos |
| -Nodo<T> cab |
| -Nodo<T> ult |
| Métodos |
| **+asigna(Conjunto B)** |
| **+combinacion(Conjunto B)** |
| **+union(Conjunto B)** |
| **+igual(Conjunto B)** |
| **+diferencia(Conjunto B)** |
| **-eliminarConjunto(Nodo<T> nodo)** |
| **+esMiembro(T info)** |
| **+eliminarTodo()** |
| +getCab(); |
| +getUlt(); |
| +setCab(Nodo<T> cab); |
| +setUlt(Nodo<T> ult); |
| +agregar(T info); |
| +listar(); |
| +recuperar(); |
| +salvar(); |
| +buscar(T infoBuscada); |
| +eliminar(T info); |

asigna(Conjunto B):

Asigna el contenido del conjunto B al Conjunto

combinacion(Conjunto B):

Devuelve la combinación del Conjunto con el conjunto B

union(Conjunto B):

Devuelve la unión del Conjunto con el conjunto B

+igual(Conjunto B):

Verifica si el Conjunto y el conjunto B son iguales

diferencia(Conjunto B):

Devuelve la diferencia del Conjunto con el conjunto B

eliminarConjunto(Nodo<T> nodo):

Elimina el nodo y todos los nodos hijos del nodo que se le pase por parámetro.

esMiembro(info):

Verifica si el nodo que se le pasa como referencia es miembro del conjunto.

eliminarTodo():

Elimina el conjunto.

Especificación e implementación de las funciones:

**asigna(Conjunto B):**

*Especificación:*

Entrada: B: Conjunto, A: Conjunto (conjunto que llama la función)

Salida: ----------------

Precondición: True

Postcondición: Conjunto A = conjunto B

*Implementación:*

Funcion **asigna(Conjunto A, Conjunto B):**

**INICIO**

Elimina el conjunto A

Para cada elemento de B hacer

Insertar en A

finPara

**FIN**

**esMiembro(T info):**

*Especificación:*

Entrada: info: T (cualquier tipo de dato), A: Conjunto (conjunto que llama la función)

Salida: b: boolean

Precondición: T es el tipo de dato del Conjunto A

Postcondición: b ≡ info € A

*Implementación:*

Funcion **esMiembro(Conjunto A, T info)**

**INICIO**

nodo <- buscar(info)

si nodo no es nulo entonces

b <- verdad

sino

b<-falso

finSi

retorna b

**FIN**

**union(Conjunto A, Conjunto B):**

*Especificación:*

Entrada: B: Conjunto, A: Conjunto (conjunto que llama la función)

Salida: C: Conjunto

Precondición: A y B: Conjuntos compatibles

Postcondición: C = A U B

*Implementación:*

Funcion **union(Conjunto A, Conjunto B)**

**INICIO**

Para cada elemento de A hacer

Insertar en C

finPara

Para cada elemento de B hacer

Si el elemento no es miembro de C

Insertar en C

finSi

finPara

**FIN**

**combinacion (Conjunto A, Conjunto B):**

*Especificación:*

Entrada: B: Conjunto, A: Conjunto (conjunto que llama la función)

Salida: C: Conjunto

Precondición: A y B: Conjuntos compatibles y disjuntos

Postcondición: C = A U B

*Implementación:*

Funcion **combinación (Conjunto A, Conjunto B)**

**INICIO**

Para cada elemento de A hacer

Insertar en C

finPara

Para cada elemento de B hacer

Insertar en C

finPara

**FIN**

**interseccion (Conjunto A, Conjunto B):**

*Especificación:*

Entrada: B: Conjunto, A: Conjunto (conjunto que llama la función)

Salida: C: Conjunto

Precondición: A y B: Conjuntos compatibles

Postcondición: C = A ^ B

*Implementación:*

Funcion **interseccion (Conjunto A, Conjunto B)**

**INICIO**

Para cada elemento de A hacer

Si el elemento es miembro de B entonces

Insertar en C

finSi

finPara

**FIN**

**igual (Conjunto A, Conjunto B):**

*Especificación:*

Entrada: B: Conjunto, A: Conjunto (conjunto que llama la función)

Salida: b: booleano

Precondición: A y B: Conjuntos compatibles

Postcondición: b ≡ A = B

*Implementación:*

Funcion **igual (Conjunto A, Conjunto B)**

**INICIO**

AuxA <- A.cab

Mientras auxA!=nulo y ¬fallo

auxB <- B.cab

mientras auxB != nulo y ¬coincidencia

si auxB.info != auxA.info entonces

coincidencia<- true

sino

auxB <- auxB.sgte

finSi

finMientras

si auxB != nulo entonces

fallo<- true

finSi

auxA <- auxA.sgte

finMientras

**FIN**