Guion de Prácticas

Estructuras de Datos [ED]

Curso 2021 – 2022

Proyecto 3: Árboles

Objetivos de Aprendizaje

Al final de la unidad el alumno será capaz de:

1. Crear distintas clases de árboles dependiendo del tipo de estructura que queramos manejar
2. Implementar montículos binarios.

Proyecto

Durante la unidad, el alumno creará un proyecto compuesto por los siguientes elementos:

- La clase **BSTNode** que implementa un nodo de un árbol de búsqueda binario
- La clase **BSTree** que implementa un árbol de búsqueda binario utilizando la clase BSTNode
- La clase **AVLNode** que implementa un nodo de un árbol AVL
- La clase **AVLTree** que implementa un árbol AVL utilizando la clase AVLNode
- La interface **EDPriorityQueue** donde se definen los métodos de una cola de prioridad
- La clase **EDBinaryHead** que implementa un montículo binario

Cada una de las clases contendrá los siguientes métodos. Se especifica la cabecera de los métodos para la clase genérica.

Árboles de búsqueda binarios

Clase BSTNode

Crear una clase que defina las propiedades de un nodo de un árbol de búsqueda binaria, así como los métodos para gestionar dichas propiedades.

- ***public BSTNode(T clave)***
  Constructor, que recibe como parámetro el contenido que se le debe asignar a la propiedad info (contenido) del nodo. También inicializará el hijo izquierdo y el derecho del nodo a ***null***. Se entiende que cuando se crea un nodo es un nodo hoja
- ***public void setInfo(T clave)***
  Asignar un nuevo contenido al nodo
- ***public T getinfo()***
  Devuelve el contenido del nodo
- ***public void setLeft(BSTNode<T> nodo)***
  Asigna el nodo que se pasa como parámetro al subarbol izquierdo
- ***public void setRight(BSTNode<T> nodo)***
  Asigna el nodo que se pasa como parámetro al subárbol derecho
- ***public BSTNode<T> getLeft()***
  Devuelve el subárbol izquierdo
- ***public BSTNode<T> getRight()***
  Devuelve el subárbol derecho
- ***public String toString()***
  Devuelve la propiedad info (contenido) del nodo en forma de cadena

Clase BSTree

Crear una clase que defina las propiedades de un árbol de búsqueda binaria, así como los métodos para gestionar dicho árbol.

- ***public BSTree()***
  Constructor que inicializa la raíz a ***null***
- ***public int addNode(T clave)***
  Método que invoca a otro método recursivo. Devolverá -2 si la clave es null, -1 si la clave ya existe 0 si lo añade correctamente
  - ***private int addNodeR(BSTNode<T> nodo, T clave)***
    Inserta un elemento en el árbol y devuelve 0
- ***public BSTNode<T> searchNode(T clave)***
  Método invoca a otro recursivo. Devuelve el nodo completo si lo encuentra y ***null*** si no lo encuentra o bien la clave o el árbol son ***null***
  - ***private BSTNode<T> searchNodeR(BSTNode<T> nodo, T info)***
    Busca un elemento en el árbol y devuelve ***null*** si no lo encuentra y el nodo completo si lo encuentra

- ***public String preOrder()***
  Devuelve el recorrido en ***pre orden*** de un árbol de izquierda a derecha en una cadena separados sus elementos por tabuladores
- ***public String postOrder()***
  Devuelve el recorrido en ***post orden*** de un árbol de izquierda a derecha en una cadena separados sus elementos por tabuladores
- ***public String inOrder()***
  Devuelve el recorrido en ***in orden*** de un árbol de izquierda a derecha en una cadena separados sus elementos por tabuladores
- ***public int removeNode(T clave)***
  Método que invoca a otro recursivo. Devolverá -2 si la clave es ***null*** o el árbol es ***null***, -1 si la clave no existe y 0 si lo borra, actualizando la raíz del árbol
  - ***private BSTNode<T> removeNodeR(BSTNode<T> nodo, T clave)***
    Borrar el elemento pasado como parámetro y devuelve el nodo que tiene que ser asignado al padre del borrado

Árboles AVL

Clase AVLNode

Crear una clase que defina las propiedades de un nodo de un árbol AVL, así como los métodos para gestionar dichas propiedades.

- ***public AVLNode(T clave)***
  Constructor, que recibe como parámetro el contenido que se le debe asignar a la propiedad info (contenido) del nodo. También inicializará el hijo izquierdo y el derecho del nodo a ***null*** y la altura y el factor de balance a cero. Se entiende que cuando se crea un nodo es un nodo hoja
- ***public void setInfo(T clave)***
  Asignar un nuevo contenido al nodo
- ***public T getinfo()***
  Devuelve el contenido del nodo
- ***public void setLeft(AVLNode<T> nodo)***
  Enlaza el nodo que se pasa como parámetro al subárbol izquierdo
- ***public void setRight(AVLode<T> nodo)***
  Enlaza el nodo que se pasa como parámetro al subárbol derecho
- ***public AVLNode<T> getLeft()***
  Devuelve el subárbol izquierdo
- ***public AVLNode<T> getRight()***
  Devuelve el subárbol derecho
- ***public int getBFHeight()***
  Devuelve la altura de un árbol
- ***public int getBF()***
  Devuelve el factor de balance de un árbol

- ***public void updateHeight()***
  Asigna un valor al factor de balance y otro a la altura en función de la altura
- ***public String toString()***
  Muestra el contenido del nodo junto con el factor de balance

Clase AVLTree

Crear una clase que defina las propiedades de un árbol AVL, así como los métodos para gestionar dicho árbol.

- ***public AVLTree()***
  Constructor que inicializa la raíz a ***null***
- ***public int addNode(T info)***
  Método que invoca a otro método recursivo. Devolverá -2 si la clave es null, -1 si la clave ya existe 0 si lo añade correctamente
  - ***private AVLNode<T> addNodeR(AVLNode<T> nodo, T clave)***
    Inserta un elemento en el árbol y devuelve el nodo actualizado
- ***public AVLNode<T> searchNode(T info)***
  Método invoca a otro recursivo. Devuelve el nodo completo si lo encuentra y ***null*** si no lo encuentra o bien la clave o el árbol son ***null***
  - ***private AVLNode<T> searchNodeR(AVLNode<T> nodo, T info)***
    Busca un elemento en el árbol y devuelve ***null*** si no lo encuentra y el nodo completo si lo encuentra
- ***private AVLNode<T> updateAndBalanceIfNecesary(AVLNode<T> nodo)***
  Actualiza el factor de balance y la altura y devuelve el nodo actualizado tras aplicar alguna de las rotaciones si es necesario
- ***private AVLNode<T> sinleLeftRotation(AVLNode<T> nodo)***
  Realiza la rotación simple a la izquierda y devuelve el nodo actualizado
- ***private AVLNode<T> sinleRightRotation(AVLNode<T> nodo)***
  Realiza la rotación simple a la derecha y devuelve el nodo actualizado
- ***private AVLNode<T> doubleLeftRotation(AVLNode<T> nodo)***
  Realiza la rotación doble a la izquierda y devuelve el nodo actualizado
- ***private AVLNode<T> doubleRightRotation(AVLNode<T> nodo)***
  Realiza la rotación doble a la derecha y devuelve el nodo actualizado
- ***public int removeNode(T info)***
  Método que invoca a otro recursivo. Devolverá -2 si la clave es ***null*** o el árbol es ***null***, -1 si la clave no existe y 0 si lo borra, actualizando la raíz del árbol
  - ***private AVLNode<T> RemoveNodeR(AVLNode<T> nodo, T clave)***
    Borrar el elemento pasado como parámetro y devuelve el nodo que tiene que ser asignado al padre del borrado
- ***public String preOrder()***
  Devuelve el recorrido en ***pre orden*** de un árbol de izquierda a derecha en una cadena separados sus elementos por tabuladores

- ***public String postOrder()***
  Devuelve el recorrido en ***post orden*** de un árbol de izquierda a derecha en una cadena separados sus elementos por tabuladores
- ***public String inOrder()***
  Devuelve el recorrido en ***in orden*** de un árbol de izquierda a derecha en una cadena separados sus elementos por tabuladores

Colas de prioridad

Interface PriorityQueue

Crear una interface genérica que defina los métodos de cualquier cola de prioridad.

- *public int add(T elemento)*
- *public T poll()*
- *public int remove(T elemento)*
- *public boolean isEmpty()*
- *public void clear()*
- *public int cambiarPrioridad(int pos, T elemento)*
- *public String toString()*

Clase BinaryHeap

Crear una clase para implementar las colas de prioridad mediante un montículo de mínimos. No se admiten claves repetidas.

1. Implementar los métodos del interface cola de prioridad
   - ***public BinaryHeap(int tam)***
     Constructor. Crea e inicializa el vector de tamaño ***tam*** e inicializa el número de elementos del vector a cero
   - ***public int add(T clave)***
     Añade un elemento a la cola de prioridad. Devuelve -2 si la clave es ***null***, -1 si no hay espacio suficiente en el vector y 0 si lo añade correctamente
   - ***public T poll()***
     Obtiene el elemento con mayor prioridad. Devuelve ***null*** si la cola está vacía
   - ***public int remove(T clave)***
     Elimina un elemento especificado de la cola de prioridad. Devuelve -2 si la clave es ***null*** o la cola está vacía, -1 si la clave no existe y 0 si lo inserta correctamente
   - ***public boolean isEmpty()***
     Devuelve ***true*** si la cola de prioridad está vacía y ***false*** el otro caso
   - ***public void clear()***
     Borrar todos los elementos de la cola de prioridad
   - ***public int cambiarPrioridad(int pos, T clave)***

Cambia la prioridad de un elemento que se encuentra en la posición ***pos*** por la que se pasa como parámetro

- ***public String toString()***
  Devuelve una cadena con los elementos de la cola de prioridad separados por tabuladores. El último no lleva tabulador

2. Implementar dos nuevos métodos en la clase:
   - ***private void filtradoAscendente(int pos)***
     Realiza un filtrado ascendente a partir de la posición pasada como parámetro
   - ***private void filtradoDescendente(int pos)***
     Realiza un filtrado descendente a partir de la posición pasada como parámetro

Consideraciones finales

- Todas las clases, propiedades y métodos escritas en el proyecto debe ser documentadas correctamente por el alumno utilizando **JavaDoc**.
- Todos los métodos diseñados deben ser evaluados por el alumno mediante una exhaustiva batería de pruebas positivas y pruebas negativas haciendo uso de **JUnit**.

Fecha de Entrega

Una vez completado el proyecto, deberá subirse al Campus Virtual en las fechas que se designen. Pasado ese período no se aceptará la entrega del proyecto de ningún modo.