UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos
Cursada 2022

Práctica 4
Árboles Generales

Objetivos

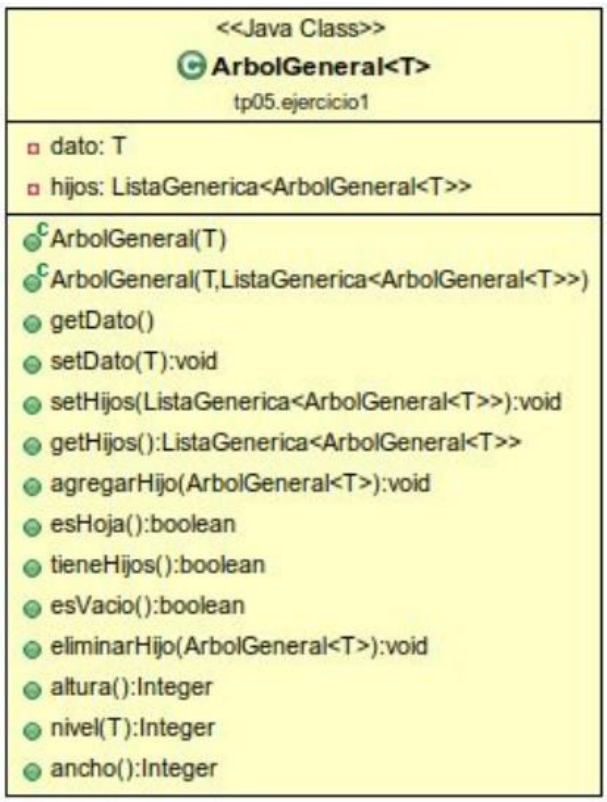
- Modelar un árbol n-ario
- Realizar recorridos sobre árboles generales
- Implementar ejercicios de aplicación sobre árboles generales.

Importante: Se recomienda continuar trabajando dentro del proyecto AyED y generar paquetes y subpaquetes para esta práctica.
Descargue el archivo **tp04_ag.zip** El archivo zip descargado desde la página de la cátedra no es un proyecto eclipse, por lo tanto:

1. Descomprima el archivo zip.
2. Sobre la carpeta **src** de su proyecto AyED haga click con el botón derecho del mouse y seleccione la opción *Import > FileSystem*.
3. Haga click en "Browse", busque la carpeta descomprimida y seleccione la carpeta **tp04_ag** (haga click para que aparezca el check seleccionado).
4. Haga click en el botón finalizar.

Ejercicio 1

Considere la siguiente especificación de la clase **ArbolGeneral** (con la representación de **Lista de Hijos**)



Nota: la clase **Lista** es la utilizada en la práctica 2.

El constructor **ArbolGeneral(T dato)** inicializa un árbol que tiene como raíz un nodo general. Este nodo tiene el dato pasado como parámetro y una lista vacía.

El constructor **ArbolGeneral(T dato, ListaGenerica<ArbolGeneral<T>> hijos)** inicializa un árbol que tiene como raíz un nodo general. Este nodo tiene el dato pasado como parámetro y tiene como hijos la lista pasada como parámetro.

El método **getDato():T** retorna el dato almacenado en la raíz del árbol.

El método **getHijos():ListaGenerica<ArbolGeneral<T>>**, retorna la lista de hijos de la raíz del árbol.

El método **agregarHijo(ArbolGeneral<T> unHijo)** agrega unHijo a la lista de hijos del árbol

El método **eliminarHijo(ArbolGeneral<T> unHijo)** elimina unHijo del árbol.

El método **tieneHIjos()** devuelve verdadero si la lista de hijos del árbol no es null y tampoco es vacía

El método **esVacio()** devuelve verdadero si el dato del árbol es null y además no tiene hijos.

Los métodos **altura(), nivel(T) y ancho()** se resolverán en el ejercicio 4.

Analice la implementación en JAVA de las clases **ArbolGeneral** brindadas por la cátedra.

Ejercicio 2

a) ¿Qué recorridos conoce para recorrer en profundidad un árbol general? Explique brevemente.

b) ¿Qué recorridos conoce para recorrer por niveles un árbol general? Explique brevemente.

c) ¿Existe alguna diferencia entre los recorridos preorden, postorden, inorden para recorrer los árboles generales respecto de los árboles binarios? Justifique su respuesta.

d) ¿Existe alguna noción de orden entre los elementos de un árbol general? Justifique su respuesta.

e) En un árbol general se define el grado de un nodo como el número de hijos de ese nodo y el grado del árbol como el máximo de los grados de los nodos del árbol. ¿Qué relación encuentra entre los Árboles Binarios sin tener en cuenta la implementación? Justifique su respuesta.

Ejercicio 3

Implemente en la clase **ArbolGeneral<T>** los siguientes métodos:

public ListaGenerica<T> numerosImparesMayoresQuePreOrden (Integer n)

Método que retorna una lista con los elementos impares del árbol receptor que sean mayores al valor "n" pasado como parámetro, recorrido en preorden.

public ListaGenerica<T> numerosImparesMayoresQueInOrden (Integer n)

Método que retorna una lista con los elementos impares del árbol receptor que sean mayores al valor "n" pasado como parámetro, recorrido en inorden.

public ListaGenerica<T> numerosImparesMayoresQuePostOrden (Integer n)

Método que retorna una lista con los elementos impares del árbol receptor que sean mayores al valor "n" pasado como parámetro, recorrido en postorden.

public ListaGenerica<T> numerosImparesMayoresQuePorNiveles()

Método que retorna una lista con los elementos impares del árbol receptor que sean mayores al valor "n" pasado como parámetro, recorrido por niveles.

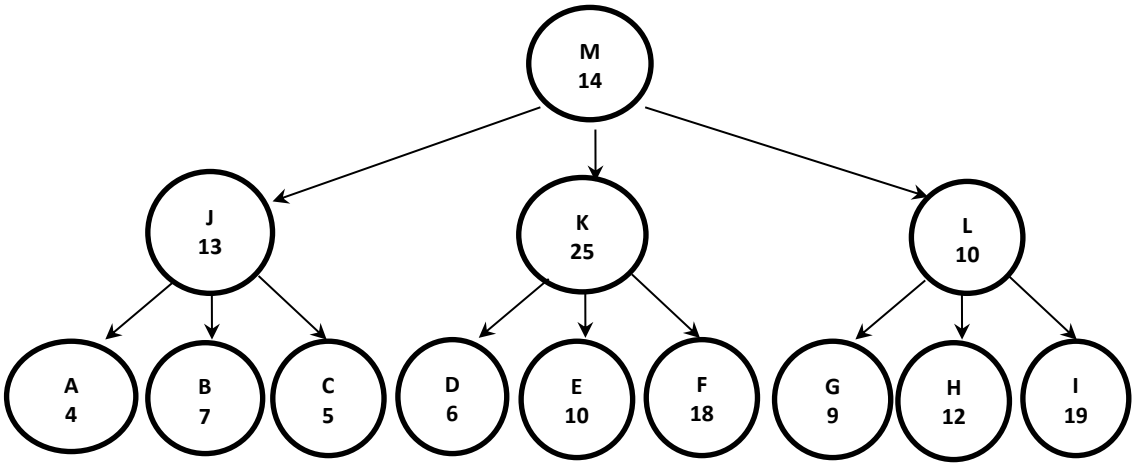
Ejercicio 4

Implemente en la clase **ArbolGeneral** los siguientes métodos

a) **public int altura(): int** devuelve la altura del árbol, es decir, la longitud del camino más largo desde el nodo raíz hasta una hoja.
b) **public int nivel(T dato)** devuelve la profundidad o nivel del dato en el árbol. El nivel de un nodo es la longitud del único camino de la raíz al nodo.
c) **public int ancho(): int** la amplitud (ancho) de un árbol se define como la cantidad de nodos que se encuentran en el nivel que posee la mayor cantidad de nodos.

Ejercicio 5

El esquema de comunicación de una empresa está organizado en una estructura jerárquica, en donde cada nodo envía el mensaje a sus descendientes. Cada nodo posee el tiempo que tarda en transmitir el mensaje.



Se debe devolver **el mayor promedio** entre todos los valores promedios de los niveles.

Para el ejemplo presentado, el promedio del nivel 0 es 14, el del nivel 1 es 16 y el del nivel 2 es 10. Por lo tanto, debe devolver 16.

a) Indique y justifique qué tipo de recorrido utilizará para resolver el problema.
b) Implementar en una clase AnalizadorArbol, el método con la siguiente firma:

public int devolverMaximoPromedio (ArbolGeneral<AreaEmpresa>arbol)

Donde **AreaEmpresa** es una clase que representa a un área de la empresa mencionada y que contiene la identificación de la misma representada con un **String** y una tardanza de transmisión de mensajes interna representada con **int**.

Ejercicio 6

Se dice que un nodo n es ancestro de un nodo m si existe un camino desde n a m.
Se dice que un nodo n es descendiente de un nodo m si existe un camino desde m a n.
Implemente un método en la clase ArbolGeneral con la siguiente firma:

public Boolean esAncestro(T a, T b): devuelve true si el valor a es ancestro del valor b.

El cual determine si un valor a es ancestro de un valor b.

Ejercicio 7

Sea una red de agua potable, la cual comienza en un caño maestro y la misma se va dividiendo sucesivamente hasta llegar a cada una de las casas.
Por el caño maestro ingresan "x" cantidad de litros y en la medida que el caño se divide, de acuerdo con las bifurcaciones que pueda tener, el caudal se divide en partes iguales en cada una de ellas. Es decir, si un caño maestro recibe 1000 litros y tiene por ejemplo 4 bifurcaciones se divide en 4 partes iguales, donde cada división tendrá un caudal de 250 litros.
Luego, si una de esas divisiones se vuelve a dividir, por ej. en 5 partes, cada una tendrá un caudal de 50 litros y así sucesivamente hasta llegar un lugar sin bifurcaciones.

Se debe implementar una clase **RedDeAguaPotable** que contenga el método con la siguiente firma:

public double minimoCaudal(double caudal)

que calcule el caudal de cada nodo y determine cuál es el mínimo caudal que recibe una casa. Asuma que la estructura de caños de la red está representada por una variable de instancia de la clase RedAguaPotable y que es un ArbolGeneral.

Extendiendo el ejemplo en el siguiente gráfico, al llamar al método minimoCaudal con un valor de 1000.0 debería retornar 25.0