Estructura de Datos – Primer Cuatrimestre – Segundo parcial – 13-06-2017

Ejercicio 1

a) Defina el encabezado de la interfaz Tree<E> y las signaturas de las operaciones positions() , children(p) ,parent(p) ,addAfter(p,rb,e).

b) Complete las clases TNodo y Árbol para implementar un árbol general, cuyos rótulos son de tipo genérico E utilizando la representación de colección de hijos y referencia al padre.

º Para la clase TNodo , defina los atributos implemente el o los constructores y escriba las signaturas pero no implemente el cuerpo de las consultas y comandos triviales.

º Para la clase Árbol , defina los atributos implemente e solamente los constructores y los métodos createRoot() y addLastChild(p,e), no defina las demás consultas ni comandos ni implemente su cuerpo.

c) Implemente en java una consulta que reciba un árbol de enteros e imprima por pantalla los rótulos de sus nodos ordenados en forma ascendente en base a la cantidad de hijos de cada nodo. Es decir, considere que un nodo A es mayor que un nodo B si la cantidad de hijos de A es mayor que la cantidad de hijos de B.
Para resolver este ejercicio utilice una cola con prioridad sin implementarla, pero si deberá implementar los métodos auxiliares utilizados.

d) Justificando adecuadamente,indique el orden del tiempo de ejecucion de los procedimiento solución al inciso (c) asumiendo que las operaciones del tipo E son de tiempo constante,e indicando apropiadamente el orden del tiempo de ejecucion de las operaciones utilizadas del TDALista y TDAColaConPrioridad.(Detallando la estructura subyacente en el lenguaje natural).

Estructura de Datos – Primer Cuatrimestre – Segundo parcial – 13-06-2017

Ejercicio 2

a) Defina usando Java las interfaces , Vertex<V> y Edge<E> para representar un grafo no dirigido de acuerdo a [GT] (cuyas operaciones se listan abajo). Los rótulos de los vértices del grafo son de tipo genérico V. Los rótulos de los arcos del grafo son de tipo genérico E.

b) Defina las clases Vertice<V,E> y Arco<V,E> que implementan las interfaces Vertex<V> y Edge<E> definidas en (a) utilizando la representación de lista de adyacencias. Complete el encabezado de las clases, defina solo atributos y signaturas de las operaciones. Implemente el o los constructores pero no implemente los cuerpos de las operaciones.

c) Defina la clase Grafo<V,E> que implemente la interfaz definida en (a) utilizando la representación de lista de adyacencias. Complete el encabezado de las clases y defina los atributos. Implemente unicamente el o los constructores y las operaciones insertVertex(x)(inserta un vértice) y removeVertex(x)(Eliminar un vértice asumiendo que ya han sido eliminado previamente todos sus arcos adyacentes) para los incisos anteriores.

d) Agregue una operación a la clase Grafo<V,E> definida en (c) que dado un vértice v1 haga un recorrido DFS (Depth First Search – Búsqueda en profundidad) comenzando en el vértice v1 y retorne un mapeo cuyas claves serán los rótulos de los vértices del grafo y el valor asociado será la cantidad de arcos adyacentes a cada vértice. En caso que el grafo tenga rótulos repetidos, debe almacenar en el mapeo unicamente aquel vértice que tenga mayor cantidad de arcos adyacentes.

Resolver el problema en Java, recuerde que esta agregando un método a la clase Grafo, por lo tanto tiene total acceso a su estructura, si utiliza operaciones del TDAGrafo no debe implementarlas, pero si debe implementar los métodos auxiliares utilizados. Asumiendo que posee las implementaciones completas de las clase lista, pila, cola, y mapeo, use las que necesite.

PositionList<E>	PriorityQueue<K,V>	Tree<E>	Graph<V,E>	Map<K,V>	Diccionary<K,V>
Size()	Size()	Size()	Vertices()	Size()	Size()
isEmpty()	IsEmpty()	isEmpty()	edges()	isEmpty()	isEmpty()
first()	min()	iterator()	IncidenEdges(v)	find(k)	values()
last()	insert(k,v)	positions()	opposite(v,e)	findAll(k)	keys()
next(p)	removeMin()	replace(v,e)	endVertices(e)	insert(k,v)	entries()
prev(p)		root()	areAdyacent(v,w)	remove(e)	put(k,v)
addFirst(e)		parent(v)	replace(v,x)	entries()	remove(k)
addLast(e)		children(v)	replace(e,x)		get(k)
addAfter(p,e)		isInternal(v)	insertVertex(x)		
addBefore(p,e)		isExternal(v)	insertEdge(v,w,x)		
remove(p)		isRoot(v)	removeVertex(v)		
set(p,e)			removeEdge(e)		
iterator()					
positions()					