Estructuras de Datos - Segundo Parcial

Ejercicio 1:

a) En el lenguaje Java, defina las clases necesarias para implementar un mapeo de elementos de tipo genérico K en elementos de tipo genérico V usando una tabla de hash abierto, para ello defina sólo atributos y encabezados de operaciones. Asuma que posee la clase Lista<E> que implementa la interfaz PositionList<E> de acuerdo a [GT] por medio de una lista doblemente enlazada (la definición de la interfaz lista se da abajo y corresponde con la vista en clase y a la dada en [GT]).
b) Esriba en el lenguaje Java el código del constructos de la clase Mapeo definida en el inciso (a).
c) Escriba en el lenguaje Java el código de las operaciones Put (insertar) y Remove (remover) para el mapeo definido en el inciso (a).
d) Escriba en el lenguaje Java una operación de consulta para la clase Mapeo que reciba otro mapeo B y devuelva verdadero si B está contenido en el mapeo receptor del mensaje y falso en caso contrario. Un mapeo B está contenido en un mapeo a si todo par (x,B(x)) en B existe en A. Implementar las operaciones auxiliares de la clase mapeo que utilice para resolver este inciso.
e) Calcule el orden del tiempo de ejecución de la solución del inciso (d) asumiendo que la función de hash distribuye las claves uniformemente. Justificar adecuadamente aclarando el orden de las operaciones de mapeo y lista usadas.

Ejercicio 2:

a) Defina en el lenguaje Java las clases NodoABB y ArbolABB para representar un nodo de un árbol binario de búsqueda y un árbol binario de búsqueda (ABB), respectivamente. Defina sólo atributos y encabezados de operaciones. Implemente **solamente** las operaciones que vaya a usar en la solución del inciso (e). De considerar necesaria la definición de otra clase o interfaz, defina solamente atributos y signaturas de operaciones; si va a usar alguna operación de estas clases adicionales en la solución del inciso (e), deberá implementar la operación usada. Implemente el comparador asociado.
b)Escriba el código Java del constructor de la clase árbol binario de búsqueda definido en le inciso (a).
c) Defina la interfaz para implementar una cola con prioridades genérica.
d) Defina la estructura de datos para implementar una cola con prioridades como la definida en el inciso (c) en términos de un Heap.
e) Agregue un método a la clase ABB que dado un entero K realice lo siguiente: Se deben eliminar del ABB *los K-ésimos nodos con rótulo más pequeño.* Como estructura auxiliar sólo puede utilizar la cola con prioridad definida en c (sin implementar las operaciones de la CCP utilizadas). ¿De qué tipo será la clave de las entradas de la cola?, ¿resulta conveniente utilizar el mismo comparador que utilizó para ABB?, de lo contrario implemente uno conveniente.
f) ¿Puede implementar el método solicitado en el inciso e) sin utilizar NINGUNA estructura auxiliar? De la estrategia para resolver el problema.
g) Calcule el orden del tiempo de ejecución de la solución asumiendo que el ABB está balanceado especificando claramente el orden del tiempo de ejecución de las operaciónes de la cola con prioridades.

Ejercicio 3:

a) Dada la interfaz Graph<K,V> definida en [GT] para modelar grafos no dirigidos:

- vertices()
- edges()
- adyacentsEdges(v)
- opposite(v,e)
- endVertices(e)
- areAdjacent(v,w)
- replace(v,x)
- insertVertex(x)
- insertEdge(v,w,x)
- removeVertex(v)
- removeEdge(e)

Defina usando Java todas las clases necesarias para representar un grafo no dirigido utilizando la representación de *lista de adyacencias.* Los rótulos de los vértices del grafo son de tipo genérico V. Los rótulos de los arcos del grafo son de tipo genérico E. No implemente las operaciones de las clases; es decir, defina sólo los atributos y signaturas de las operaciones.

b) Implemente en el lenguaje Java la operación removeEdge(e) (eliminar un arco e) para las definiciones propuestas en el inciso (a).

c) Implemente en el lenguaje Java la operación removeVertex(v) (eliminar un vértice v) para las definiciones propuestas en el inciso (a).

d) Dado un grafo G, y tres vértices v1, v2 y v3 se desea hallar un camino de v1 a v3 que pase por v2. Resolver este problema en Java, y exclusivamente en término de las operaciones de los TDAs usados, asumiendo que posee las implementaciones completas de las clases grafo (definida en (a)), lista, pila, cola y mapeo. Use las que necesite.

Observación: Debe devolver el camino hallado.

PositionList<E>:
- size()
- isEmpty()
- first()
- last()
- next(p)
- prev(p)
- addFirst(e)
- addLast(e)
- addAfter(p,e)
- addBefore(p,e)
- remove(p)
- set(p,e)
- iterator()

ColaConPrioridades<K,V>
- size()
- isEmpty()
- min()
- insert(key,value)

- removeMin()

Map<K,V>
- size()
- isEmpty()
- get(k)
- put(k,v)
- remove(k)
- keys()
- values()
- entries()