Nombre:	LU:	Comisión: 2	Cant. hojas: 3

Estructuras de Datos - Segundo Parcial

Licenciatura en Ciencias de la Computación – Ingeniería en Computación – Ingeniería en Sistemas de Información
Universidad Nacional del Sur – 30/5/2024

Observaciones generales:

- **REALICE LOS EJERCICIOS EN HOJAS SEPARADAS**.
- Lea todo el ejercicio antes de comenzar a desarrollarlo.
- Numere y ponga su nombre a todas las hojas. Indique cuántas hojas entrega.
- Recuerde que se evalúan correctitud, eficiencia y legibilidad de sus soluciones.

A

Importante: Para resolver este examen utilice las interfaces presentadas en clase. Al final del examen se encuentra un recordatorio de los métodos que cada una provee.

Ejercicio 1:

a) Agregue un método a la clase DicccionarioHashAbierto<K,V> tal que reciba una clave *k* y un entero *e* y retorne verdadero si y solo si en el diccionario que recibe el mensaje hay al menos *e* entradas con clave *k*. El método debe lanzar la excepción InvalidKeyException en caso de que la clave sea inválida. Recuerde que cuenta con total acceso a la estructura. Si utiliza otro método del TDADiccionario deberá implementarlo como así también los método auxiliares.

```
public class DiccionaroHashAbierto<K,V> implements Dictionary<K,V>{
 protected static final float fc=0.7f;
 protected PositionList<Entrada<K,V>>[] buckets;
 protected int n;
 protected int N;
 ...}
```

b) Indique el orden del tiempo de ejecución de su solución. Justifique adecuadamente.

Ejercicio 2:

a) Agregue un método a la clase Arbol<E> con la siguiente signatura: `public iterable<E> convertirAhoja(Position<E> p) throws InvalidPositionException` tal que elimine del árbol receptor del mensaje todos los nodos que sean necesarios para convertir la posición *p* en hoja. El método debe retornar un iterable con todos los elementos de los nodos eliminados. Si *p* ya es una hoja debe retornar un iterable vacío. Recuerde que cuenta con total acceso a la estructura. Si utiliza otro método del TDAArbol deberá implementarlo como así también los método auxiliares.

```
public class Arbol<E> implements Tree<E>{
 protected TNodo<E> raiz;
 protected int cant;
 ...}
```

b) Indique el orden del tiempo de ejecución de su solución. Justifique adecuadamente.

Ejercicio 3:

a) Escriba un método que reciba un árbol de caracteres y retorne un mapeo cuyas claves sean cada una de las vocales y sus respectivos valores la cantidad de veces que aparece cada vocal en el árbol. Para resolver este ejercicio debe implementar un único recorrido en posorden y sin usar los iteradores del árbol (ni iterator ni positions). Considere solo letras en minúscula. Asuma que cuenta con los TDAs Arbol y Mapeo totalmente implementados. Si utiliza métodos auxiliares debe implementarlos.

b) Indique el orden del tiempo de ejecución de su solución. Justifique adecuadamente.

PositionList<E>:	Tree<E>	Dictionary<K,V>:	Map<K,V>:
size( )	size()	size()	size()
isEmpty()	isEmpty()	isEmpty()	isEmpty()
first()	iterator()	find(k)	get(k)
last()	positions()	findAll(k)	put(k, v)
next(p)	replace(v, e)	insert(k,v)	remove(k)
prev(p)	root()	remove(e)	keys()
addFirst(e)	parent(v)	entries()	values()
addLast(e)	children(v)		entries()
addAfter(p, e)	isInternal(v)		
addBefore(p, e)	isExternal(v)		
reove(p)	isRoot(v)		
set (p, e)	createRoot(e)		
iterator()	addFirstChild(p,e)		
positions()	addLastChild(p,e)		
	addBefore(p,rb,e)		
	addAfter(p,lb,e)		
	removeExternal(p)		
	remoceInternal(p)		
	removeNode(p)		

HOJA 1

EJERCICIO 1 A

```
public boolean alMenosEconK(K key, int e) throws InvalidKeyException {
    if (key == null) {
        throw new InvalidKeyException("Clave nula");
    }
// SE ASUMIRÁ QUE SE INGRESARÁ UN ENTERO POSITIVO COMO PARÁMETRO.
    int hash = hash(key);
    int contador = 0;
    PositionList<Entry<K,V>> bucket = buckets[hash];
    Iterator<Entry<K,V>> entriesIterator = bucket.iterator();
    Entry<K,V> entry ;
    while (entriesIterator.hasNext() && contador < e) {
        entry = entriesIterator.next();
        if (entry.getKey().equals(key)) {
            ++contador;
        }
    }

    return (contador >= e);
}

private int hash(K key) {
    return (Math.abs(key.hashCode()) % N);
}
```

B) EN EL MÉTODO alMenosEconK, SE RECORRE EL BUCKET CORRESPONDIENTE AL VALOR HASH ASOCIADO A LA CLAVE key. ESTE RECORRIDO SE HARÁ n VECES, EN EL PEOR CASO, DONDE n ES LA CANTIDAD DE ENTRADAS QUE CONTIENE ESTE BUCKET.
POR LO TANTO EL ORDEN DEL TIEMPO DE EJECUCIÓN ES O(n).

X Es O(1) con respecto a la cantidad de elementos del diccionario

HOJA 2

Nota del autor (???): En el encabezado del método postOrden, se me olvidó agregar "throws InvalidPositionException", ya que este utiliza el método checkPosition, el cual puede arrojar la misma excepción. Una sutileza que pasó desapercibida.
Como dato adicional, en lugar de hacer un método recursivo que cuente la cantidad de elementos eliminados y lo retorne, podría haber hecho el método void, donde solo agregue los elementos eliminados a la lista, y luego desde el método convertirAhoja, consultar el size de la lista para restarle el valor al size del árbol, y hubiera sido más sencillo. Me compliqué la vida al pedo; los nervios del parcial, ustedes entienden.

EJERCICIO 2 A

```
public Iterable<E> convertirAhoja (Position<E> p) throws InvalidPosException {
    TNode<E> node = checkPosition(p);
DoublyLinkedList<E> elemEliminados = new DoublyLinkedList<>();
    int numEliminados = 0
    if (!isExternal(node)) {
        numEliminados = postOrden (node, p, elemEliminados, numEliminados);
        node.children = new DoublyLinkedList<>();
        cant= cant - numEliminados
    }
    return elemEliminados;
}
```

Hay que borrar la lista de hijos, no se pedía

```
private int postOrden(TNode<E> node, Position<E> p, DoublyLinkedList<E> list, int n){
    int num= n
    for (TNode<E> child : node.getChildren()) {
        num += postOrden(child, p, list, n  );
    }
    pos = checkPosition(p);
    if(pos != node) {
        list.addLast(node.element())
        ++num
    }
    return num;
}
```

```
private TNode<E> checkPosition (Position<E> p) throws InvalidPositionException {
    if (p == null) throw new InvalidPositionException("POSICIÓN NULA");
    if(isEmpty()) throw new InvalidPositionException("ARBOL VACÍO");
    TNode<E> node;
```

NOTA

(SIGUE →)

```
Try {
    node = (TNode<E>) p;
}
catch (ClassCastException e){
    Throw new InvalidPositionException("POSICIÓN INVÁLIDA");
}
return node;
}
```

B) EN EL MÉTODO convertirAhoja, SE LLAMA AL MÉTODO postOrden. ESTE MÉTODO, REALIZA UN RECORRIDO EN POSTORDEN A PARTIR DE UN NODO DADO. EN EL PEOR DE LOS CASOS, ESTE RECORRIDO COMENZARÁ DESDE LA RAÍZ, Y SE REPETIRÁ n VECES, DONDE n ES LA CANTIDAD DE NODOS DEL ÁRBOL. POR LO TANTO, EL ORDEN DEL TIEMPO DE EJECUCIÓN DE ESTE MÉTODO ES O(n), CONSIDERANDO ADEMÁS QUE EL RESTO DE MÉTODOS UTILIZADOS SE RESUELVEN EN TIEMPO CONSTANTE.

HOJA 3

Nota del autor (???): Esta solución cuenta efectivamente el número de vocales (si resulta que están presentes al menos una vez). Esto puede llegar a ser incorrecto, porque si una vocal no estuviera presente, lo correcto sería que dijera que dicha vocal se encuentra cero veces. ¿Cómo se podría solucionar? Simplemente agregando de antemano al mapeo 5 entradas correspondientes a cada vocal, y en todas su valor inicial será 0.

EJERCICIO 3 A

```
public Map<Character, Integer> contarVocales(Tree<Character> tree) {
    Map<Character, Integer> mapeo = new MapeoHashAbierto<>();
    try {
        postOrden(Tree, Tree.root(), mapeo);
    } catch (EmptyTreeException e) {
        System.out.println(e.getMessage());
    }
    return mapeo;
}


private void postOrden(Tree<Character> tree, Position<Character> node,
                       Map<Character, Integer> mapeo) {

    for(Position<Character> child : tree.children(node)) {
        postOrden(Tree, child, mapeo);
    }

    Integer contador; Character elem = node.element();
    Try {
        if(elem.equals('a') || elem.equals('e') || elem.equals('i') || elem.equals('o') ||
            elem.equals('u')) {
            contador = mapeo.get(elem);
            if(contador == null) {
                contador = 0;
            }
        } mapeo.put(elem, contador + 1);
    }
    catch (InvalidKeyException e) {
        System.out.println(e.getMessage());
    }
}
```

NOTA

(B) EN EL MÉTODO contarVocales, SE LLAMA A UN MÉTODO AUXILIAR LLAMADO postOrden, QUE COMO SU NOMBRE INDICA, SE HACE UN RECORRIDO RECURSIVO EN POSTORDEN, EMPEZANDO POR LA RAÍZ DEL ÁRBOL. ESTO SE HARÁ PARA CONTAR TODAS LAS VOCALES QUE TIENE EL ÁRBOL, CON LO CUAL ESTO SE REPETIRÁ n VECES, DONDE n ES EL TAMAÑO DEL ÁRBOL.

EN CADA RECORRIDO, SE LLAMAN A LOS MÉTODOS get() Y put() DE MapeoHashAbierto. ASUMIENDO UNA BUENA DISTRIBUCIÓN DENTRO DEL MAPEO, EL MÉTODO get() SE RESOLVERÁ DE FORMA CONSTANTE.

ASUMIENDO ADEMÁS QUE EL MAPEO TIENE UN TAMAÑO ADECUADO PARA ALMACENAR LAS, A LO SUMO, 5 ENTRADAS QUE SE AGREGARÁN, SIN TENER QUE HACER reHash, EL MÉTODO put() TAMBIÉN SE RESOLVERÁ DE FORMA CONSTANTE.

FINALMENTE, EL ORDEN DE TIEMPO DE EJECUCIÓN DEL MÉTODO contarVocales ES O(n), DONDE n ES EL TAMAÑO DEL ÁRBOL INGRESADO COMO PARÁMETRO.