Algoritmos y Estructuras de Datos I

CASO DE ESTUDIO – PARTE I

VERONICA ECHEZARRETA

Contenido

Introducción

Se realizó una simulación de una cadena de supermercados y su funcionamiento, ya sea agregando productos a la empresa, y a su vez, estos mismos productos a cada sucursal, así como eliminando productos de cada sucursal o listando los productos existentes en cada sucursal junto con su stock, etc.

Problema planteado

Al simular el funcionamiento de una cadena de supermercados, se requieren distintas funcionalidades:

- Incorporar un nuevo producto a una sucursal del supermercado.
- Agregar stock a un producto existente en una sucursal.
- Simular la venta de un producto en una sucursal (reducir el stock de un producto existente). De no haber stock suficiente para la venta en esa sucursal, deberá indicarse la lista de sucursales que tengan el stock suficiente, ordenada por cantidad de producto.
- Eliminar productos que ya no se venden (por no ser comercializados más) en todas las sucursales del supermercado.
- Dado un código de producto, indicar las existencias de este en todas las sucursales, ordenada por sucursal.
- Listar todos los productos registrados, en una sucursal, ordenado por nombre de producto, presentando además su stock.
- Listar todos los productos registrados, ordenados por ciudad, barrio, y nombre de producto, presentando además su stock.

Análisis de alternativas

Alternativa 1

La primera alternativa supone el uso de arboles binarios de búsqueda para almacenar los productos en la empresa.

También se utilizan estos mismos para almacenar las sucursales en la empresa, y estas sucursales, a su vez, emplean árboles para almacenar los productos en cada una de ellas.

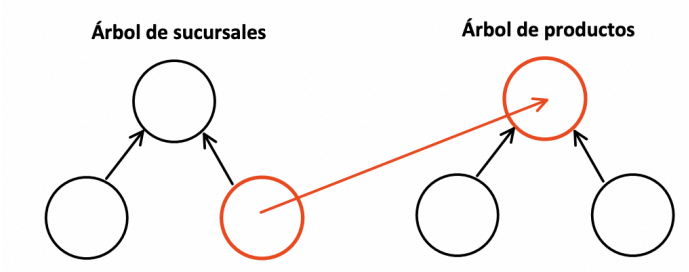
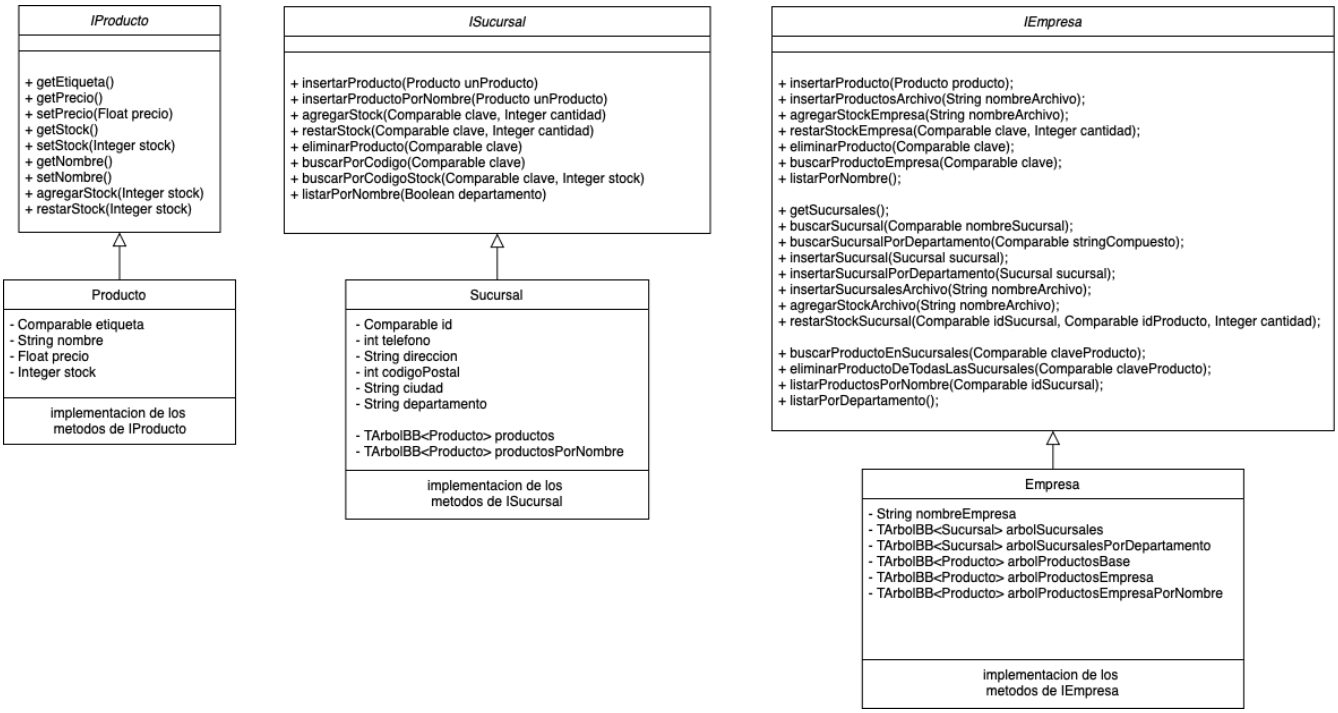
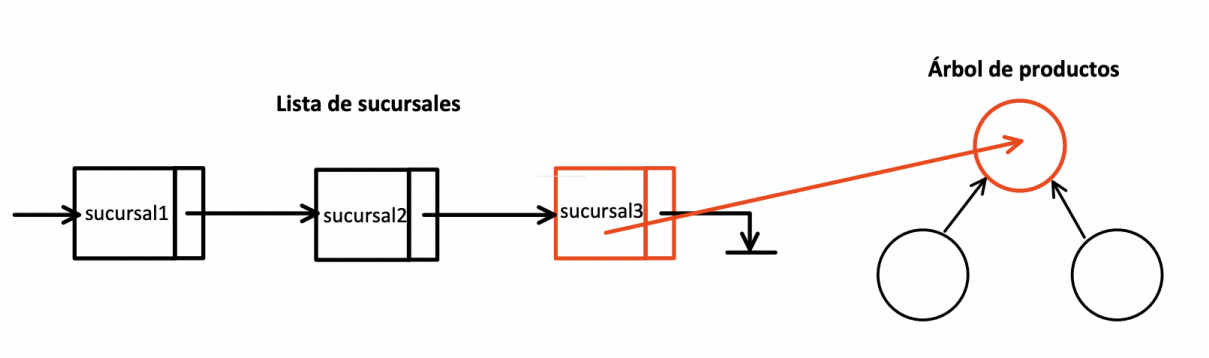


Diagrama de clases



Alternativa 2

La segunda alternativa supone también el uso de arboles binarios de búsqueda para almacenar los productos en la empresa, pero para las sucursales se emplearían listas en vez de árboles, y para los productos árboles.



Algoritmos

Métodos de Producto

Atributos:

- Comparable etiqueta
- String nombre
- Float precio
- Integer stock

Agregar stock al producto

Lenguaje natural:
Se crea un método que toma como parámetro un entero al cual será sumado al stock actual del producto.

Precondiciones:
- El parámetro debe ser un entero

Postcondiciones:
- El stock debe haber cambiado dependiendo del entero tomado

Pseudocódigo:

```
Producto.agregarStock(Integer stock) : int    O(1)
COM
    this.stock += stock                       O(1)
    devolver this.stock                       O(1)
FIN
```

Restar stock al producto

Lenguaje natural:
Se crea un método que toma como parámetro un entero al cual será restado al stock actual del producto.

Precondiciones:
- El parámetro debe ser un entero

Postcondiciones:
- El stock debe haber cambiado dependiendo del entero tomado

Pseudocódigo:

```
Producto.restarStock (Integer stockARestar) : int          O(1)
COM
    SI (stockARestar > this.stock) ENTONCES                    O(1)
        devolver -1                                            O(1)
    SINO SI (stockARestar = this.stock) ENTONCES               O(1)
        this.stock ← this.stock – stockARestar                 O(1)
```

```
        devolver 0          //Al ser el mismo monto, devuelve 0        O(1)
    SINO
        this.stock ← this.stock – stockARestar                       O(1)
        devolver this.stock                                          O(1)
    FIN SI
FIN
```

Métodos de Sucursal

Atributos:

- Comparable id
- int telefono
- String direccion
- int codigoPostal
- String ciudad
- String departamento

- TArbolBB<Producto> productos
- TArbolBB<Producto> productosPorNombre

Insertar Producto

Lenguaje natural:
Se inserta un producto al árbol de productos tomando como etiqueta el código de este.

Precondiciones:
- Debe ser un producto valido
- El árbol de productos no puede ser nulo

Postcondiciones:
- El árbol de productos tendrá un elemento más

Pseudocódigo:

Sucursal.insertarProducto (Producto producto) : void	**O(N)**
COM	
TElementoAB<Producto> p ← nuevo TElementoAB(producto.etiqueta, producto)	O(1)
productos.insertar(p)	O(N)
FIN	

Insertar Producto por nombre

Lenguaje natural:
Se inserta un producto al árbol de productos tomando como etiqueta el nombre de este.

Precondiciones:
- Debe ser un producto valido
- El árbol de productos no puede ser nulo

Postcondiciones:
- El árbol de productos tendrá un elemento más

Pseudocódigo:

Sucursal.insertarProductoPorNombre (Producto producto) : void	**O(N)**
COM	

```
    TElementoAB<Producto> p ← nuevo TElementoAB(producto.nombre, producto)    O(1)
    productos.insertar(p)                                                      O(N)
FIN
```

Agregar stock

Lenguaje natural:
Se busca si el producto ya existe en el árbol de productos. Si existe, se le agrega stock mediante un parámetro de entero al mismo. Sino devuelve falso.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El stock debe haber cambiado dependiendo del entero tomado

Pseudocódigo:

```
Sucursal.agregarStock (Comparable clave, Integer cantidad) : Boolean    O(N)
COM
    TElementoAB<Producto> p ← productos.buscar(clave)                   O(N)

    SI (p <> nulo) ENTONCES                                             O(1)
        p.datos.agregarStock(cantidad)                                  O(1)
        devolver true                                                   O(1)
    SINO
        devolver false                                                  O(1)
    FIN SI
FIN
```

Restar stock

Lenguaje natural:
Se busca si el producto ya existe en el árbol de productos. Si existe, se le resta stock mediante un parámetro de entero al mismo. Sino devuelve falso.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El stock debe haber cambiado dependiendo del entero tomado

Pseudocódigo:

```
Sucursal.restarStock (Comparable clave, Integer cantidad) : Boolean    O(N)
COM
    TElementoAB<Producto> p ← productos.buscar(clave)                   O(N)

    SI (p <> nulo) ENTONCES                                             O(N)
        int stockFinal ← p.datos.restarStock(cantidad)                  O(1)
        SI (stockFinal = -1) ENTONCES                                   O(1)
```

```
            devolver false                                          O(1)
        SINO SI (stockFinal = 0) ENTONCES                           O(1)
            devolver true                                           O(1)
        SINO
            devolver true                                           O(1)
        FIN SI
    SINO
        devolver false                                              O(1)
    FIN SI
FIN
```

Eliminar producto

Lenguaje natural:
Se busca un producto en el árbol de productos de la sucursal mediante una clave dada por parámetro. Si ese producto existe en ese árbol, se elimina, sino se devuelve false.

Precondiciones:

- La clave por la que se busca debe ser una clave válida

Postcondiciones:

- El árbol de productos tendrá un elemento menos o seguirá con la misma cantidad de elementos.

Pseudocódigo:

```
Sucursal.eliminarProducto (Comparable clave) : Boolean              O(N)
COM
    TElementoAB<Producto> p ← productos.buscar(clave)               O(N)

    SI (p <> nulo) ENTONCES                                         O(1)
        productos.eliminar(clave)                                   O(N)
        devolver true                                               O(1)
    SINO
        imprimir("El producto que desea eliminar no se encuentra en la sucursal")   O(1)
        devolver false                                              O(1)
    FIN SI
FIN
```

Buscar Producto

Lenguaje natural:
Se recorre el árbol de productos de la sucursal en busca de un producto que coincida con la clave dada por parámetro. Si coincide, se devuelve el dato de ese producto, sino devuelve nulo.

Precondiciones:

- La clave por la que se busca debe ser una clave válida

Postcondiciones:

- El árbol de productos quedará intacto

Pseudocódigo:

```
Sucursal.buscarPorCodigo (Comparable idProducto) : Producto                O(N)
COM
    TElementoAB<Producto> unProducto ← productos.buscar(idProducto)        O(N)

    SI (unProducto <> nulo) ENTONCES                                       O(1)
        devolver unProducto.dato                                           O(1)
    SINO
        devolver nulo                                                      O(1)
    FIN SI
FIN
```

Buscar Producto y tomar el stock

Lenguaje natural:

Se recorre el árbol de productos en busca de un producto que coincida con la clave dada por parámetro, y a su vez, que el stock que toma por parámetro sea menor al stock del producto. Si coincide, se devuelve el dato de ese producto, sino devuelve nulo.

Precondiciones:

- La clave por la que se busca debe ser una clave válida

Postcondiciones:

- El árbol de productos quedará intacto

Pseudocódigo:

```
Sucursal.buscarPorCodigoStock (Comparable idProducto, Integer stock) : Producto    O(N)
COM
    TElementoAB<Producto> unProducto ← productos.buscar(idProducto)                O(N)

    SI (unProducto <> nulo AND unProducto.datos.stock > stock) ENTONCES            O(1)
        devolver unProducto.dato                                                   O(1)
    SINO
        devolver nulo                                                              O(1)
    FIN SI
FIN
```

Listar productos por nombre

Lenguaje natural:

Se recorre el árbol de productos en inorden para poder listar los productos por nombre en orden alfabético.

Precondiciones:

- Debe recorrerse en un árbol que tenga los productos agregados por nombre (la etiqueta debe ser el nombre del producto).

Postcondiciones:

- El árbol de productos deberá estar inalterado

Pseudocódigo:

```
Sucursal.listarPorNombre (Boolean departamento) : void                    O(N)
COM
    SI (NOT productosPorNombre.esVacio()) ENTONCES                        O(1)
        SI(NOT departamento) ENTONCES                                     O(1)
            imprimir("Los productos existentes en la sucursal " + id + " son:")   O(1)
        SINO
            imprimir("Los productos existentes de la sucursal " + departamento +
                    ciudad + codigoPostal + id + " son:")                 O(1)
        FIN SI
        listarPorNombreImplementacion(productosPorNombre.raiz)            O(1)
    SINO
        imprimir("La lista de sucursales esta vacia")                     O(1)
    FIN SI
FIN

Sucursal.listarPorNombreImplementacion(TElementoAB<Producto> producto) : void    O(N)
COM
    SI(producto.hijoIzq <> nulo) ENTONCES                                 O(1)
        listarPorNombreImplementacion(producto.hijoIzq)                   O(1)
    FIN SI

    imprimir(producto.etiqueta + producto.datos.stock)                    O(1)

    SI(producto.hijoDer <> nulo) ENTONCES                                 O(1)
        listarPorNombreImplementacion(producto.hijoDer)                   O(1)
    FIN SI
FIN
```

Métodos de Empresa

Atributos:

- String nombreEmpresa
- TArbolBB<Sucursal> arbolSucursales
- TArbolBB<Sucursal> arbolSucursalesPorDepartamento
- TArbolBB<Producto> arbolProductosBase
- TArbolBB<Producto> arbolProductosEmpresa
- TArbolBB<Producto> arbolProductosEmpresaPorNombre

Buscar Producto

Lenguaje natural:
Se recorre el árbol de productos de la empresa en busca de un producto que coincida con la clave dada por parámetro. Si coincide, se devuelve el dato de ese producto, sino devuelve nulo.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El árbol de productos quedará intacto

Pseudocódigo:

Empresa.buscarProducto(Comparable idProducto) : Producto	**O(N)**
COM	
TElementoAB<Producto> unProducto ← arbolProductosBase.buscar(idProducto)	O(N)
SI (unProducto <> nulo) ENTONCES	O(1)
devolver unProducto.dato	O(1)
SINO	
devolver nulo	O(1)
FIN SI	
FIN	

Insertar Producto

Lenguaje natural:
Se inserta un producto al árbol de productos tomando como etiqueta el código de este. Se busca el producto en el árbol de productos, si no existe, se agrega al mismo.

Precondiciones:
- Debe ser un producto valido
- El árbol de productos no puede ser nulo

Postcondiciones:
- El árbol de productos tendrá un elemento más

Pseudocódigo:

```
Empresa.insertarProducto(Producto producto) : void                    O(N)
COM
    TElementoAB<Producto> unProducto ← nuevo TElementoAB(producto.etiqueta, producto)|O(1)
    Producto p ← buscarProducto(producto.etiqueta)                    O(N)

    SI (p = nulo) ENTONCES                                            O(1)
        arbolProductosBase.insertar(unProducto)                       O(N)
    FIN SI
FIN
```

Insertar productos desde un archivo

Lenguaje natural:
Se lee un archivo tomado por parámetro, y se "parte" cada línea por comillas. Luego se crean productos tomando como etiqueta la primera parte de la línea y el nombre como la segunda parte de la línea. También, se le agrega el precio desde la tercera parte de la línea. Por ultimo, se agrega este producto al árbol de productos llamando al método "insertarProducto" visto anteriormente.

Precondiciones:
- Debe ser un producto valido
- El árbol de productos no puede ser nulo

Postcondiciones:
- El árbol de productos tendrá un elemento más

Pseudocódigo:

```
Empresa.insertarProductosArchivo(String nombreArchivo) : void          O(N^2)
COM
    String[] línea ← ManejadorArchivosGenerico.leerArchivo(nombreArchivo)   O(N)

    PARA (String data en línea) HACER                                  O(N)
        String[] datos = data.split("\"");                             O(1)
        Producto producto ← nuevo Producto(datos[0], datos[1])         O(1)
        producto.precio(datos[2])                                      O(1)
        insertarProducto(producto)                                     O(N)
    FIN PARA
FIN
```

Agregar stock a los productos de la empresa desde un archivo

Lenguaje natural:
Se lee un archivo tomado por parámetro, y se "parte" cada línea por comas. Se toma la primera parte de la línea como la etiqueta del producto y se busca si el producto ya existe en el árbol de productos. Si existe, se crea un nuevo producto tomando como etiqueta el id del producto anteriormente creado, y el dato también. Esto se hace para que cada sucursal tenga una instancia de ese producto, pero con diferentes stocks. Luego, se le agrega stock llamando al método "agregarStock" del producto.

Precondiciones:

- Cada atributo del producto debe estar cuidadosamente visto cuando se “parte” cada línea del archivo. Las partes de las líneas deben ser validos para poder ser usados posteriormente

Postcondiciones:
- Los productos tendrán sus stocks modificados
- El árbol de productos no debe tener ni un elemento menos ni mas

Pseudocódigo:

Empresa.agregarStockEmpresa(String nombreArchivo) : void	**$O(N^3)$**
COM	
String[] línea ← ManejadorArchivosGenerico.leerArchivo(nombreArchivo)	O(N)
PARA (String data en línea) HACER	O(N)
String[] datos = data.split(",")	O(1)
Comparable idProducto ← datos[0]	O(1)
int stock ← datos[1]	O(1)
Producto producto ← buscarProducto(idProducto)	O(N)
SI (producto <> nulo) ENTONCES	O(1)
Producto p ← nuevo Producto(idProducto, producto.nombre)	O(1)
TElementoAB<Producto> unProducto ← nuevo TElementoAB(p.etiqueta, p)	O(1)
arbolProductosEmpresa.insertar(elemProducto)	O(N)
TElementoAB<Producto> pNombre = nuevo TElementoAB(p.nombre, p);	O(1)
arbolProductosEmpresaPorNombre.insertar(pNombre);	O(N)
p.agregarStock(stock)	O(N)
FIN SI	
FIN PARA	
FIN	

Restar stock a un producto

Lenguaje natural:
Se busca si el producto ya existe en el árbol de productos. Si existe, se le resta stock mediante un parámetro de entero al mismo.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El stock debe haber cambiado dependiendo del entero tomado

Pseudocódigo:

Empresa.restarStockEmpresa(Comparable clave, Integer cantidad) : Boolean	**O(N)**
COM	
TElementoAB<Producto> elemProducto ← arbolProductosEmpresa.buscar(clave)	O(N)
SI (elemProducto <> nulo) ENTONCES	O(1)
int stockFinal ← elemProducto.dato.restarStock(cantidad)	O(1)
SI (stockFinal <> -1) ENTONCES	O(1)

```
            devolver true                                                    O(1)
        SINO
            devolver false                                                   O(1)
        FIN SI
    SINO
        devolver false                                                       O(1)
    FIN SI
FIN
```

Eliminar un producto de la empresa

Lenguaje natural:
Se busca un producto en el árbol de productos de la empresa mediante una clave dada por parámetro. Si ese producto existe en ese árbol, se elimina, sino se devuelve false.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El árbol de productos tendrá un elemento menos o seguirá con la misma cantidad de elementos.

Pseudocódigo:

```
Empresa.eliminarProducto(Comparable clave) : Boolean                         O(N)
COM
    TElementoAB<Producto> p ← arbolProductosEmpresa.buscar(clave)            O(N)

    SI (p <> nulo) ENTONCES                                                  O(1)
        arbolProductosEmpresa.eliminar(clave)                                O(N)
        devolver true                                                        O(1)
    SINO
        imprimir("El producto que desea eliminar no se encuentra en la empresa")  O(1)
        devolver false                                                       O(1)
    FIN SI
FIN
```

Buscar producto en la empresa

Lenguaje natural:
Se recorre el árbol de productos de la empresa en busca de un producto que coincida con la clave dada por parámetro. Si coincide, se devuelve el dato de ese producto, sino devuelve nulo.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El árbol de productos quedará intacto

Pseudocódigo:

```
Empresa. buscarProductoEmpresa (Comparable idSucursal) : Producto                O(N)
COM
    TElementoAB<Producto> elemProducto ← arbolProductosEmpresa.buscar(clave)     O(N)

    SI(elemProducto <> nulo) ENTONCES                                            O(1)
        devolver elemProducto.datos                                              O(1)
    SINO
        devolver nulo                                                            O(1)
    FIN SI
FIN
```

Listar productos por nombre

Lenguaje natural:

Se recorre el árbol de productos en inorden para poder listar los productos por nombre en orden alfabético.

Precondiciones:

- Debe recorrerse en un árbol que tenga los productos agregados por nombre (la etiqueta debe ser el nombre del producto).

Postcondiciones:

- El árbol de productos deberá estar inalterado

Pseudocódigo:

```
Empresa.listarPorNombre() : void                                                 O(N)
COM
    SI (NOT arbolProductosEmpresaPorNombre.esVacio()) ENTONCES                   O(1)
        listarPorNombreImplementacion(arbolProductosEmpresaPorNombre.raiz)       O(1)
    FIN SI
FIN

Empresa.listarPorNombreImplementacion (TElementoAB<Producto> producto) : void   O(N)
COM
    SI(producto.hijoIzq <> nulo) ENTONCES                                        O(1)
        listarPorNombreImplementacion(producto.hijoIzq)                          O(1)
    FIN SI

    imprimir(producto.etiqueta + producto.datos.stock)                           O(1)

    SI(producto.hijoDer <> nulo) ENTONCES                                        O(1)
        listarPorNombreImplementacion(producto.hijoDer)                          O(1)
    FIN SI
FIN
```

Buscar una sucursal

Lenguaje natural:
Se recorre el árbol de sucursales de la empresa en busca de una sucursal que coincida con la clave dada por parámetro. Si coincide, se devuelve el dato de esa sucursal, sino devuelve nulo.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El árbol de sucursales quedará intacto

Pseudocódigo:

Empresa.buscarSucursal(Comparable idSucursal) : Sucursal	**O(N)**
COM	
TElementoAB<Sucursal> elemSucursal ← arbolSucursales.buscar(clave)	O(N)
SI(elemSucursal <> nulo) ENTONCES	O(1)
devolver elemSucursal.datos	O(1)
SINO	
devolver nulo	O(1)
FIN SI	
FIN	

Buscar una sucursal por departamento

Lenguaje natural:
Se recorre el árbol de sucursales por departamento de la empresa en busca de una sucursal que coincida con la clave dada por parámetro. Si coincide, se devuelve el dato de esa sucursal, sino devuelve nulo.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El árbol de sucursales quedará intacto

Pseudocódigo:

Empresa.buscarSucursalPorDepartamento(Comparable idSucursal) : Sucursal	**O(N)**
COM	
TElementoAB<Sucursal> elemSucursal ← arbolSucursalesPorDepartamento.buscar(clave)	O(N)
SI(elemSucursal <> nulo) ENTONCES	O(1)
devolver elemSucursal.datos	O(1)
SINO	
devolver nulo	O(1)
FIN SI	
FIN	

Insertar una sucursal

Lenguaje natural:
Se inserta una sucursal al árbol de sucursales tomando como etiqueta el código de esta. Se busca la sucursal en el árbol de sucursales, si no existe, se agrega al mismo.

Precondiciones:
- Debe ser una sucursal valida
- El árbol de sucursales no puede ser nulo

Postcondiciones:
- El árbol de sucursales tendrá un elemento más

Pseudocódigo:

Empresa.insertarSucursal (Sucursal sucursal) : void	**O(N)**
COM	
TElementoAB<Sucursal> elemSucursal ← nuevo TElementoAB(sucursal.id, sucursal)	O(1)
Sucursal s ← buscarSucursal(sucursal.id)	O(N)
SI (s = nulo) ENTONCES	O(1)
arbolSucursales.insertar(elemSucursal)	O(N)
FIN SI	
FIN	

Insertar una sucursal por departamento

Lenguaje natural:
Se inserta una sucursal al árbol de sucursales por departamento tomando como etiqueta el departamento de esta. Se busca la sucursal en el árbol de sucursales, si no existe, se agrega al mismo.

Precondiciones:
- Debe ser una sucursal valida
- El árbol de sucursales por departamento no puede ser nulo

Postcondiciones:
- El árbol de sucursales por departamento tendrá un elemento más

Pseudocódigo:

Empresa.insertarSucursalPorDepartamento (Sucursal sucursal) : void	**O(N)**
COM	
String stringCompuesto;	
stringCompuesto = sucursal.departamento + ", " + sucursal.ciudad + ", "	
+ sucursal.codigoPostal + ", " + sucursal.id	O(1)
TElementoAB<Sucursal> elemSucursal ← nuevo TElementoAB(stringCompuesto ,sucursal)	O(1)
Sucursal s ← buscarSucursal(sucursal.id)	O(N)
SI (s = nulo) ENTONCES	O(1)
arbolSucursalesPorDepartamento.insertar(elemSucursal)	O(N)

```
    FIN SI
FIN
```

Insertar sucursales desde un archivo

Lenguaje natural:
Se lee un archivo tomado por parámetro, y se "parte" cada línea por comas. Luego se crean sucursales tomando como id la primera parte de la línea, el teléfono como la segunda parte de la línea, la dirección como la tercera parte, el código postal como la cuarta parte, la ciudad como la quinta parte y el departamento como la sexta parte de la linea. Por ultimo, se agrega esta sucursal al árbol de sucursales llamando al método "insertarSucursal" visto anteriormente, o se llama al método "insertarSucursalPorDepartamento".

Precondiciones:
- Deben ser sucursales validas
- El árbol de sucursales no puede ser nulo

Postcondiciones:
- El árbol de sucursales tendrá al menos un elemento más

Pseudocódigo:

Empresa.insertarSucursalesArchivo (String nombreArchivo) : void	**$O(N^2)$**
COM	
String[] línea ← ManejadorArchivosGenerico.leerArchivo(nombreArchivo)	O(N)
PARA (String data en línea) HACER	O(N)
String[] datos = data.split(",");	O(1)
int stock ← datos[1]	O(1)
int cp ← datos[3]	O(1)
Sucursal sucursal ← nueva Sucursal(data[0], tel, data[2], cp, data[4], data[5])	O(1)
insertarSucursal(sucursal)	O(N)
insertarSucursalPorDepartamento(sucursal)	O(N)
FIN PARA	
FIN	

Agregar stock a las sucursales desde un archivo

Lenguaje natural:
Se lee un archivo tomado por parámetro, y se "parte" cada línea por comas. Se toma la primera parte de la línea como la etiqueta de la sucursal y la segunda parte de la línea como la etiqueta del producto. Se busca si la sucursal ya existe en el árbol de sucursales. Si existe, se busca si el producto ya existe en el árbol de productos. Si existe, se crea un nuevo producto tomando como etiqueta el id del producto anteriormente creado, y el dato también. Esto se hace para que cada sucursal tenga una instancia de ese producto, pero con diferentes stocks. Luego, se agrega ese producto a la sucursal y se le agrega stock llamando al método "agregarStock" de la sucursal.

Precondiciones:

- Cada atributo del producto y la sucursal debe estar cuidadosamente visto cuando se “parte” cada línea del archivo. Las partes de las líneas deben ser validos para poder ser usados posteriormente

Postcondiciones:
- Los productos tendrán sus stocks modificados
- El árbol de productos tendrá mas elementos, dependiendo de cuantos productos haya
- El árbol de sucursales no se verá alterado

Pseudocódigo:

Empresa.agregarStockArchivo (String nombreArchivo) : void	**O(N^3)**
COM	
String[] línea ← ManejadorArchivosGenerico.leerArchivo(nombreArchivo)	O(N)
PARA (String data en línea) HACER	O(N)
String[] datos = data.split(",")	O(1)
Comparable idSucursal ← datos[0]	O(1)
Comparable idProducto ← datos[1]	O(1)
int stock ← datos[2]	O(1)
Sucursal sucursal ← buscarSucursal(idSucursal)	O(N)
SI (sucursal <> nulo) ENTONCES	O(1)
Producto producto ← buscarProducto(idProducto)	O(N)
Producto p ← nuevo Producto(idProducto, producto.nombre)	O(1)
sucursal.insertarProducto(p)	O(N)
sucursal.insertarProductoPorNombre(p)	O(N)
sucursal.agregarStock(idProducto, stock)	O(N)
FIN SI	
FIN PARA	
FIN	

Restar stock a las sucursales (venta de un producto)

Lenguaje natural:
Se busca si el producto ya existe en el árbol de sucursales. Si existe, se le resta stock mediante un parámetro de entero al mismo. Si el stock que se pasa por parámetro es mayor al stock del producto, se busca el resto de sucursales y se verifica si estas sucursales tienen stock suficiente o no para poder restárselo.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El stock debe haber cambiado dependiendo del entero tomado

Pseudocódigo:

Empresa.restarStockSucursal(Comparable idSucursal, Comparable idProducto, Integer cantidad)) : Boolean	**O(N^3)**
COM	

```
    Sucursal sucursal ← buscarSucursal(idSucursal)                    O(N)
    Boolean hayStock ← false                                          O(1)

    SI (sucursal <> nulo) ENTONCES                                    O(1)
        hayStock ← sucursal.restarStock(idProducto, cantidad)         O(N)

        SI (NOT hayStock) ENTONCES                                    O(1)
            buscarProductoEnSucursalesStock(idProducto, cantidad)     O(N)
        FIN SI
    FIN SI
    devolver hayStock                                                 O(1)
FIN
```

Buscar productos en el resto de sucursales si el stock que se quiere restar es mayor del stock que tiene el producto en una sucursal dada por parámetro

Lenguaje natural:

Se recorre el árbol de sucursales en inorden para poder buscar los productos en cada sucursal ordenadas por id, y se busca si el stock del producto es mayor al del stock pasado por parámetro.

Precondiciones:

- Debe recorrerse en un árbol que tenga las sucursales agregadas por id (la etiqueta debe ser el id de la sucursal).

Postcondiciones:

- El árbol de productos deberá estar inalterado
- El árbol de sucursales deberá estar inalterado

Pseudocódigo:

Empresa.buscarProductoEnSucursalesStock(Comparable claveProducto, Integer stock) : void|O(N^2)

```
COM
    SI (NOT arbolSucursales.esVacio()) ENTONCES                                   O(1)
        buscarProductoEnSucursalesStockImplementacion(arbolSucursales.raiz, claveProducto, stock)
                                                                                  O(N^2)
    SINO
        imprimir("La lista de sucursales esta vacia")                             O(1)
    FIN SI
FIN
```

Empresa.buscarProductoEnSucursalesStockImplementacion(TElementoAB<Sucursal> elemSucursal, Comparable claveProducto, Integer stock) : void **O(N^2)**

```
COM
    SI (elemSucursal.hijoIzq <> nulo) ENTONCES                                    O(1)
        buscarProductoEnSucursalesStockImplementacion(elemSucursal.hijoIzq, claveProducto,
stock)                                                                            O(1)
    FIN SI

    elemSucursal.datos.buscarPorCodigoStock(claveProducto, stock)                 O(N)

    SI (elemSucursal.hijoDer <> nulo) ENTONCES                                    O(1)
```

```
        buscarProductoEnSucursalesStockImplementacion(elemSucursal.hijoDer, claveProducto,
stock)                                                                          O(1)
    FIN SI
FIN
```

Buscar productos en sucursales

Lenguaje natural:
Se recorre el árbol de sucursales en inorden para poder buscar los productos en cada sucursal ordenadas por id, y se imprime el stock al lado del id del producto.

Precondiciones:
- Debe recorrerse en un árbol que tenga las sucursales agregadas por id (la etiqueta debe ser el id de la sucursal).

Postcondiciones:
- El árbol de productos deberá estar inalterado
- El árbol de sucursales deberá estar inalterado

Pseudocódigo:

```
Empresa.buscarProductoEnSucursales (Comparable claveProducto) : Boolean        O(N^2)
COM
    Boolean seEncontro ← false                                                  O(1)
    SI (NOT arbolSucursales.esVacio()) ENTONCES                                 O(1)
        seEncontro ← buscarProductoEnSucursalesImplementacion (arbolSucursales.raiz,
claveProducto)                                                                  O(N^2)
    SINO
        imprimir("La lista de sucursales esta vacia")                           O(1)
    FIN SI
    devolver seEncontro                                                         O(1)
FIN
```

```
Empresa.buscarProductoEnSucursalesImplementacion(TElementoAB<Sucursal> elemSucursal,
Comparable claveProducto, Integer stock) : Boolean                              O(N^2)
COM
    Boolean seEncontro ← false                                                  O(1)
    SI (elemSucursal.hijoIzq <> nulo) ENTONCES                                  O(1)
        seEncontro ← buscarProductoEnSucursalesImplementacion (elemSucursal.hijoIzq,
claveProducto) OR seEncontro                                                    O(1)
    FIN SI

    Producto p ← elemSucursal.datos.buscarPorCodigo(claveProducto)              O(N)
    seEncontro ← seEncontro OR (p <> nulo)                                      O(1)

    SI (elemSucursal.hijoDer <> nulo) ENTONCES                                  O(1)
        seEncontro ← buscarProductoEnSucursalesImplementacion (elemSucursal.hijoDer,
claveProducto) OR seEncontro                                                    O(1)
    FIN SI
FIN
```

Eliminar productos de todas las sucursales

Lenguaje natural:
Se recorre el árbol de sucursales en inorden. Se busca si el producto existe en cada sucursal, y si existe se elimina.

Precondiciones:
- La clave por la que se busca debe ser una clave válida

Postcondiciones:
- El árbol de productos tendrá un elemento menos o seguirá con la misma cantidad de elementos.
- El árbol de sucursales se vera inalterado

Pseudocódigo:

```
Empresa.eliminarProductoDeTodasLasSucursales (Comparable claveProducto) : Boolean     O(N^2)
COM
    Boolean seEncontro ← false
    SI (NOT arbolSucursales.esVacio()) ENTONCES                                       O(1)
        seEncontro ← eliminarProductoDeTodasLasSucursalesImplementacion(arbolSucursales.raiz,
claveProducto)
                                                                                      O(N^2)
    SINO
        imprimir("La lista de sucursales esta vacia")                                 O(1)
    FIN SI
    devolver seEncontro
FIN

Empresa.eliminarProductoDeTodasLasSucursalesImplementacion (TElementoAB<Sucursal>
elemSucursal, Comparable claveProducto, Integer stock) : Boolean                      O(N^2)
COM
    Boolean seEncontro ← false
    SI (elemSucursal.hijoIzq <> nulo) ENTONCES                                        O(1)
        seEncontro ← eliminarProductoDeTodasLasSucursalesImplementacion(elemSucursal.hijoIzq,
claveProducto) OR seEncontro                                                          O(1)
    FIN SI

    seEncontro ← elemSucursal.datos.eliminarProducto(claveProducto, stock) OR seEncontro O(N)

    SI (elemSucursal.hijoDer <> nulo) ENTONCES                                        O(1)
        seEncontro ← eliminarProductoDeTodasLasSucursalesImplementacion(elemSucursal.hijoDer,
claveProducto) OR seEncontro                                                          O(1)
    FIN SI

    devolver seEncontro
FIN
```

Listar productos por nombre

Lenguaje natural:
Se recorre el árbol de productos en inorden de una sucursal dada por parámetro para poder listar los productos por nombre en orden alfabético.

Precondiciones:
- Debe recorrerse en un árbol que tenga los productos agregados por nombre (la etiqueta debe ser el nombre del producto).

Postcondiciones:
- El árbol de sucursales deberá estar inalterado
- El árbol de productos deberá estar inalterado

Pseudocódigo:

```
Empresa.listarProductosPorNombre (Comparable idSucursal) : void                    O(N)
COM
    Sucursal sucursal ← buscarSucursal(idSucursal)                                 O(N)
    sucursal.listarPorNombre(false)      //Se pasa un boolean en false para que no busque por
departamento                                                                       O(N)
FIN
```

Listar productos por departamento, ciudad, código postal, id de sucursal

Lenguaje natural:
Se recorre el árbol de sucursales por departamento, ciudad, código postal y id (ya que pueden haber mas sucursales con el mismo departamento, misma ciudad y código postal) en inorden para poder listar los productos por nombre en orden alfabético.

Precondiciones:
- Debe recorrerse en un árbol que tenga las sucursales agregadas por departamento, ciudad, código postal y id (la etiqueta debe ser el departamento, ciudad, código postal y id), y cada una de estas sucursales debe tener un arbol de productos agregados por nombre (la etiqueta debe ser el nombre del producto).

Postcondiciones:
- El árbol de sucursales deberá estar inalterado
- El árbol de productos deberá estar inalterado

Pseudocódigo:

```
Empresa.listarPorDepartamento () : void                                  O(N^2)
COM
    SI (NOT arbolSucursalesPorDepartamento.esVacio()) ENTONCES           O(1)
        listarPorDepartamentoImplementacion (arbolSucursales.raiz)       O(N^2)
    SINO
        imprimir("La lista de sucursales esta vacia")                    O(1)
    FIN SI
FIN
```

Empresa.listarPorDepartamentoImplementacion(TElementoAB<Sucursal> elemSucursal): void **$O(N^2)$**

```
COM
    SI (elemSucursal.hijoIzq <> nulo) ENTONCES                          O(1)
        listarPorDepartamentoImplementacion (elemSucursal.hijoIzq)      O(1)
    FIN SI

    elemSucursal.datos.listarPorNombre(true)    //Se pasa un boolean en true para que busque por
departamento                                                            O(N)

    SI (elemSucursal.hijoDer <> nulo) ENTONCES                          O(1)
        listarPorDepartamentoImplementacion (elemSucursal.hijoDer)      O(1)
    FIN SI
FIN
```

Selección y justificación de alternativa a implementar

Se eligió la alternativa 1, que consiste en tener un árbol de sucursales y que las mismas tengan un árbol de productos cada una, ya que consideré que recorrer una lista o un árbol con pocos elementos eran más o menos igual de eficientes, y ya que para los productos se utilizaban arboles también, decidí seguir por el mismo camino con las sucursales.

Se crearon interfaces de Productos (IProducto), Sucursales (ISucursal) y Empresa (IEmpresa), luego se tomaron las mismas clases e interfaces de TArbolBB y TElementoAB que utilizamos anteriormente en clase.

Conclusiones

- Es difícil agrupar la información en una única estructura y esperar que el acceso a la información sea óptimo para todos los casos de uso (ejemplo: sucursales ordenadas por nombre vs. sucursales ordenadas por departamento, ciudad y código postal).
- Puede aprovecharse la existencia de punteros a objetos para poder mantener diferentes estructuras (en este caso varios ABB con diferentes criterios de ordenación) en memoria sin la necesidad de duplicar los objetos en sí, evitando malgastar memoria.
- El uso de claves compuestas para los árboles ayuda a que la información pueda ser ordenada en un árbol por varios criterios anidados, donde el primer atributo se repite entre varios objetos.

Guía del usuario

El programa consta de dos partes: en la primera se ejecutan los métodos mismos de la empresa, en la segunda, se ejecutan los métodos de la empresa con las sucursales.

Para que el programa corra más rápido, si se quiere, se comentan los métodos que no se utilicen en el momento y se descomentan los que si se quieran utilizar. Por ejemplo, si se quieren ejecutar los métodos de la primera parte, se comentan los métodos de la segunda parte, y viceversa.