Universidad Católica del Uruguay

Algoritmos y Estructuras de Datos I

PROYECTO INDIVIDUAL – PARTE I

DIEGO KLEINMAN

1. Introducción

En éste informe se explica el diseño de un sistema software para gestionar productos de la cadena de supermercados “Grandeza y Elegancia ANte Todo” (G.E.AN.T), se analizan dos alternativas posibles y se discuten las características referentes a la implementación del sistema.

Este sistema incluye operaciones como las ventas, incorporaciones de productos, agregado de stock a productos ya existentes, listados de productos de acuerdo a una variedad de parámetros y eliminación de productos.

1.1 Problema planteado

La empresa G.E.AN.T me encarga construir un sistema para su cadena de supermercados, este encargo incluye una serie de requerimientos (acciones que mi sistema debe ser capaz de realizar).

Yo como programador debo analizar alternativas para la implementación del sistema en base a mis conocimientos de estructuras de datos e implementar la alternativa que crea más correcta para el caso planteado.

1.1.1 Requisitos de la empresa para el sistema de software

- Venta de un producto en una sucursal (de no haber stock suficiente debe indicarse una lista de sucursales que tengan el stock necesario, ordenadas por cantidad de producto).
- Agregar stock a un producto en una sucursal.
- Dado un código de producto indicar las existencias totales en la cadena de supermercados.
- Dado un código de producto indicar las existencias del mismo ordenadas por sucursal.
- Listar todos los productos registrados ordenados por nombre, presentando su stock.
- Listar todos los productos registrados en una sucursal, ordenados por nombre y presentando su stock.
- Listar todos los productos registrados ordenados por ciudad, presentando su stock en esa ciudad.
- Listar todos los productos registrados ordenados por barrio, presentando su stock en ese barrio.
- Incorporar un nuevo producto a la cadena de supermercados (a todas las sucursales).
- Incorporar un nuevo producto a una sucursal específica.
- Eliminar un producto de la cadena de supermercados.
- Eliminar un producto de una cierta sucursal.

1.1.2 Algunas consideraciones importantes

El sistema software se hizo tomando en cuenta la magnitud de la empresa G.E.AN.T.

Además, cabe destacar que en la elección de la estructura del sistema software se priorizó siempre la eficiencia a la hora de realizar los requerimientos pedidos por encima del gasto de memoria que el propio sistema pueda llegar a tener.

Estos puntos se ampliarán en la sección de “Análisis de alternativas” a la hora de referirse al cálculo del costo de memoria.

Las operaciones no primitivas en orden de relevancia descendente serían :

- Venta
- Agregar stock
- Listados
- Incorporar producto
- Eliminar producto

Las ventas probablemente se realizarían en “pequeñas” cantidades y con mucha repetición, siendo ésta la operación no primitiva más usada.

Agregar stock posiblemente se haría muchas veces pero normalmente serán operaciones que impliquen grandes cantidades de productos “de una vez” y no sean operaciones tan frecuentes como las ventas.

Los listados serían operaciones que se podrían hacer diaria, semanal o mensualmente, y por ende, menos frecuentes que las anteriores.

Incorporar producto seguramente se haría pocas veces, al principio cuando inicializamos el sistema y luego esporádicamente cuando se comercialicen productos nuevos.

Eliminar producto sería la operación menos frecuente, se realizaría cuando un producto deje de producirse o se cambie de proveedor para un cierto lote de productos.

Las operaciones primitivas en orden de relevancia descendente serían:

- Búsqueda
- Inserción
- Eliminación

Para casi todas las operaciones no primitivas voy a necesitar hacer búsquedas, es la operación primitiva más usada. Las inserciones serán usadas por varios de los métodos del sistema y las eliminaciones serán las menos usadas.

En base a estas frecuencias diagramaré mi solución y optaré por una u otra alternativa a la hora de implementar el sistema software.

2. Análisis de alternativas

En esta sección procederé a realizar las descripciones de las alternativas contempladas para la realización del proyecto, las presentaré textualmente y luego mostraré explicaciones gráficas de cada alternativa. Además, analizaré los costos de memoria de cada una.

2.1 Consideraciones básicas para los análisis de los costos de memoria

2.1.1 Consideraciones generales

Para calcular espacio de memoria ocupada voy a suponer que tengo 10mil productos y 100 sucursales distintas, las sucursales no coinciden en barrio ni en ciudad unas con las otras. Además considero que todos los productos se venden en todas las sucursales y supongo que se implementará el programa en un sistema operativo de 64 bits.

Además cabe destacar que al ingresar los datos de sucursales proporcionados por el cuerpo docente se tomaron códigos postales como barrios.

2.1.2 Tipos de datos utilizados en el sistema software

- String(con un máximo de 16 caracteres) : códigos de productos,etiquetas de productos, sucursales,barrios y ciudades, direcciones de sucursal.
- String(con un máximo de 8 caracteres) : teléfono de sucursal
- Double : Precios de producto
- Arrays de un elemento (de tipo int) : Stock de Producto

Todo lo que se realiza con el tipo de dato “String” se podría implementar con arrays de cierta cantidad de elementos, esto nos ahorraría memoria, pero a su vez también nos dificultaría más las operaciones a realizar en el programa.
Teniendo en cuenta que el espacio en memoria generalmente no es un problema en el año 2020 (o al menos no se compara frente a los problemas que nos pueden generar los tiempos de ejecución), prefiero el uso de String por sobre los arrays de Char.

2.1.3 Datos básicos acerca de la memoria y los tipos de datos utilizados

- Una variable de tipo String de X caracteres ocupa como mínimo ((X*2) + 38 bytes + n) , siendo n el menor número necesario para que el espacio total de la variable en bytes sea múltiplo de 8. [1]
  → Una variable de tipo String de 16 caracteres ocupa 72 bytes
  → Una variable de tipo String de 8 caracteres ocupa 56 bytes.
- Una variable de tipo double ocupa 8 bytes
- Una array de enteros con X elementos ocupa ((X*4) + 12 bytes + n), siendo n el menor número necesario para que el espacio total de la variable en bytes sea múltiplo de 8.
  → Un array de un elemento (de tipo entero) ocupa 16 bytes
- Una referencia a un objeto en un sistema operativo de 64 bits ocupa 8 bytes

2.2 Alternativa no implementada

2.2.1 Estructura básica

Para esta alternativa se utilizarían la listas simplemente enlazadas como única estructura de datos en el sistema.

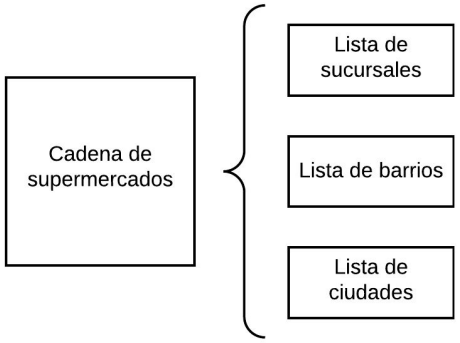
En este sistema habrían varios elementos relacionados entre sí:

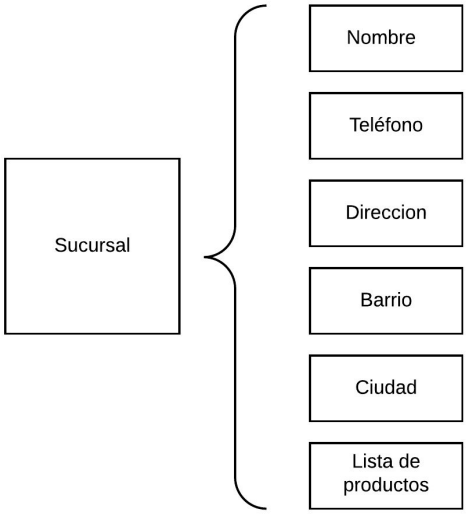
- Producto (tendría: nombre, código, precio, stock).
- Sucursal (tendría: nombre, teléfono, dirección, barrio, ciudad, Lista de productos).
- Lista de productos (tendría: Nodos con un código de producto como su etiqueta y con un objeto Producto como su dato).
- Lista de sucursales (tiene: Nodos con un nombre de sucursal como su etiqueta y con un objeto Sucursal como su dato)
- Lista de ciudades (tendría: Nodos con un nombre de ciudad como su etiqueta y con un objeto lista de productos (código-stock) como su dato).
- Lista de barrios (tendría: Nodos con un nombre de barrio como su etiqueta y con un objeto lista de productos (código-stock) como su dato).
- Cadena de supermercados (tendría: Lista de sucursales, Lista de ciudades, Lista de barrios).

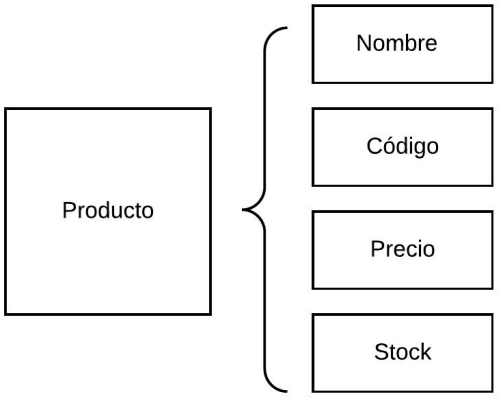
[1] Dato obtenido de la página: https://www.javamex.com/tutorials/memory/string_memory_usage.shtml

2.2.2 Diagramas explicativos

Procederé a mostrar diagramas explicativos de la estructura que tendría el programa:



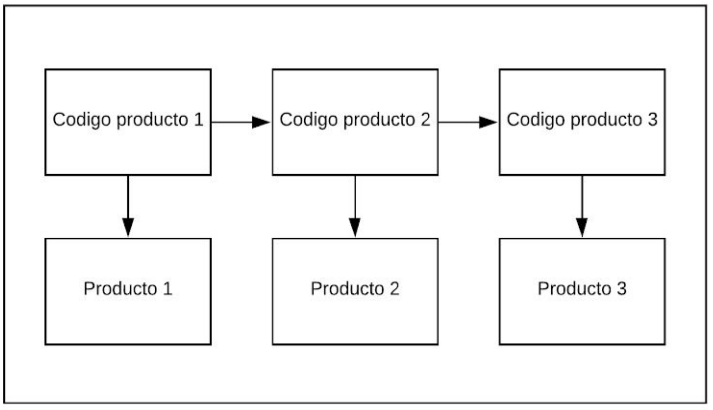




Una lista de productos se compone de nodos que tienen como etiqueta al código del producto y como dato al objeto producto.
Los nodos forman una lista de encadenamiento simple.
(No pueden haber productos repetidos en la lista)

Nodo producto X = Nodo(código producto X, producto X)

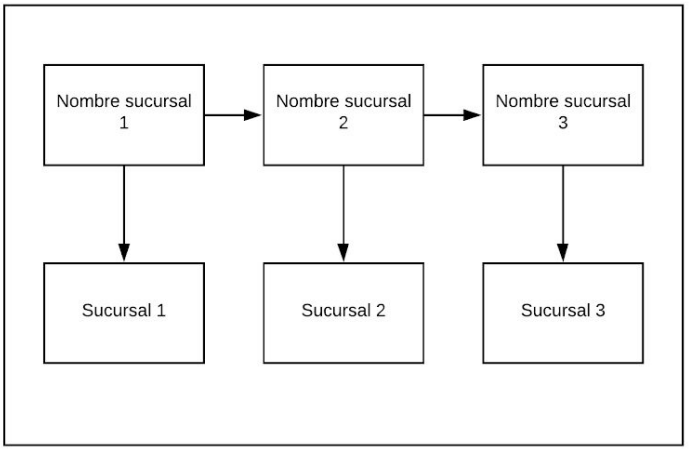
Lista de productos



Una lista de sucursales se compone de nodos que tienen como etiqueta al nombre de cada sucursal y como dato al objeto sucursal.
Los nodos forman una lista de encadenamiento simple.
(No pueden haber sucursales repetidas en la lista)

Nodo sucursal X = Nodo(nombre sucursal X, sucursal X)

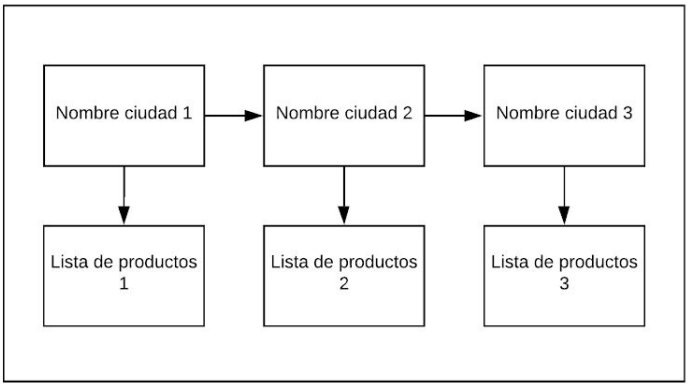
Lista de sucursales



Una lista de ciudades se compone de nodos que tienen como etiqueta al nombre de cada ciudad y como dato una lista de productos.
Los nodos forman una lista de encadenamiento simple.
(No pueden haber ciudades repetidas en la lista)
Las listas de productos tendrian nodos con etiqueta = codigo de producto y dato= stock de ese producto en ese barrio.

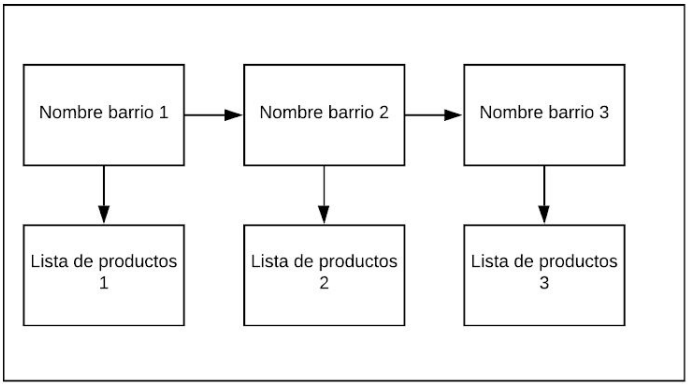
Nodo ciudad X = Nodo(nombre ciudad X, lista de productos)

Lista de ciudades



La estructura de la "lista de barrios" es análoga a la de la "lista de ciudades"
(No pueden haber barrios repetidos en la lista).
Las listas de productos tendrian nodos con etiqueta = codigo de producto y dato= stock de ese producto en ese barrio.

Lista de barrios



2.2.3 Análisis de costo de memoria

Considerando 10 mil productos y 100 sucursales obtengo el siguiente resultado:

Memoria total utilizada por esta alternativa ≈ 452 MB

Los cálculos específicos se encuentran en el Anexo bajo la sección "*Análisis de costo de memoria de alternativa no implementada*".

2.3 Alternativa implementada

2.3.1 Estructura básica

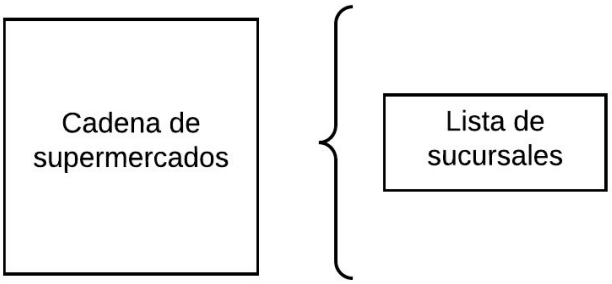
Para esta alternativa se utilizarán listas simplemente enlazadas para guardar la información referente a las sucursales y se utilizarán Árboles Binarios de Búsqueda para guardar la información referente a los productos.

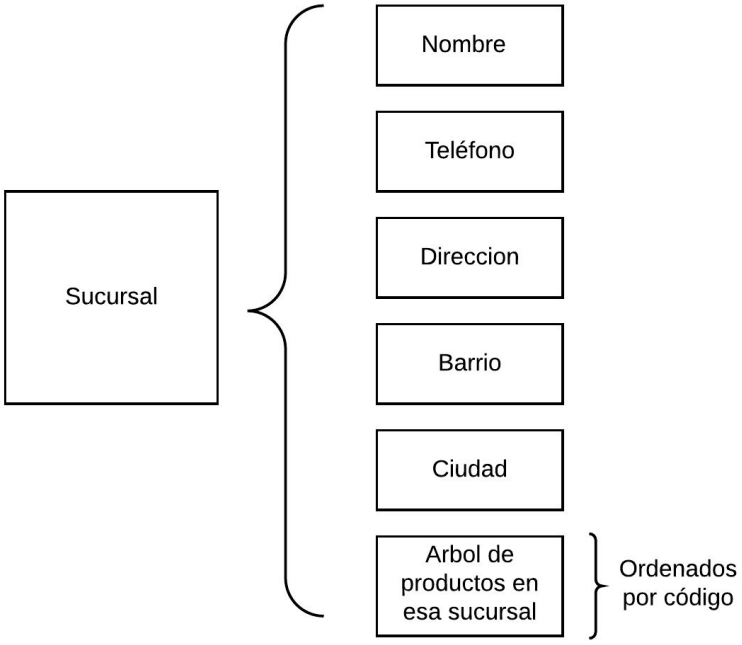
En este sistema habrán varios elementos relacionados entre sí:

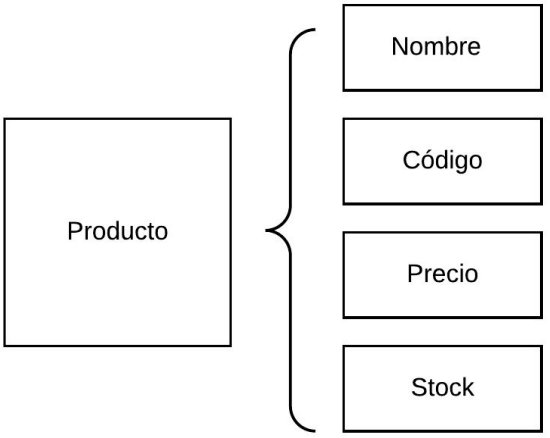
- Producto (tiene: nombre, código, precio, stock)
- Sucursal (tiene: Nombre, teléfono, dirección, barrio, ciudad, Árbol de productos).
- Lista de sucursales (tiene: Nodos con un nombre de sucursal como su etiqueta y con un objeto Sucursal como su dato)
- Árbol de productos (tiene: TElementosAB con un código de producto como su etiqueta y con un objeto Producto como su dato)
- Cadena de supermercados (tiene: Lista de sucursales)

2.3.2 Diagramas explicativos

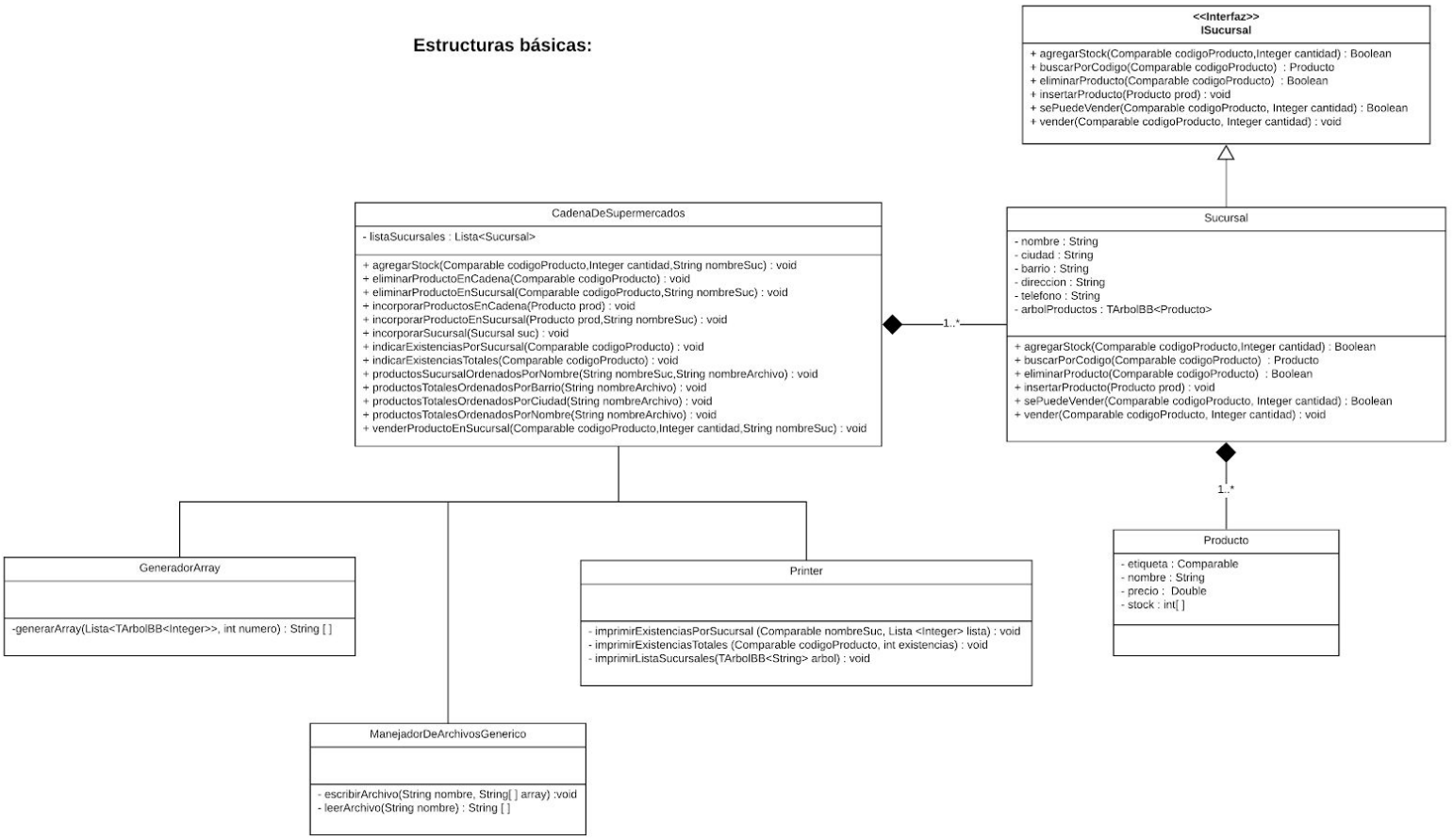
Procederé a mostrar diagramas explicativos de la estructura que tendría el programa:



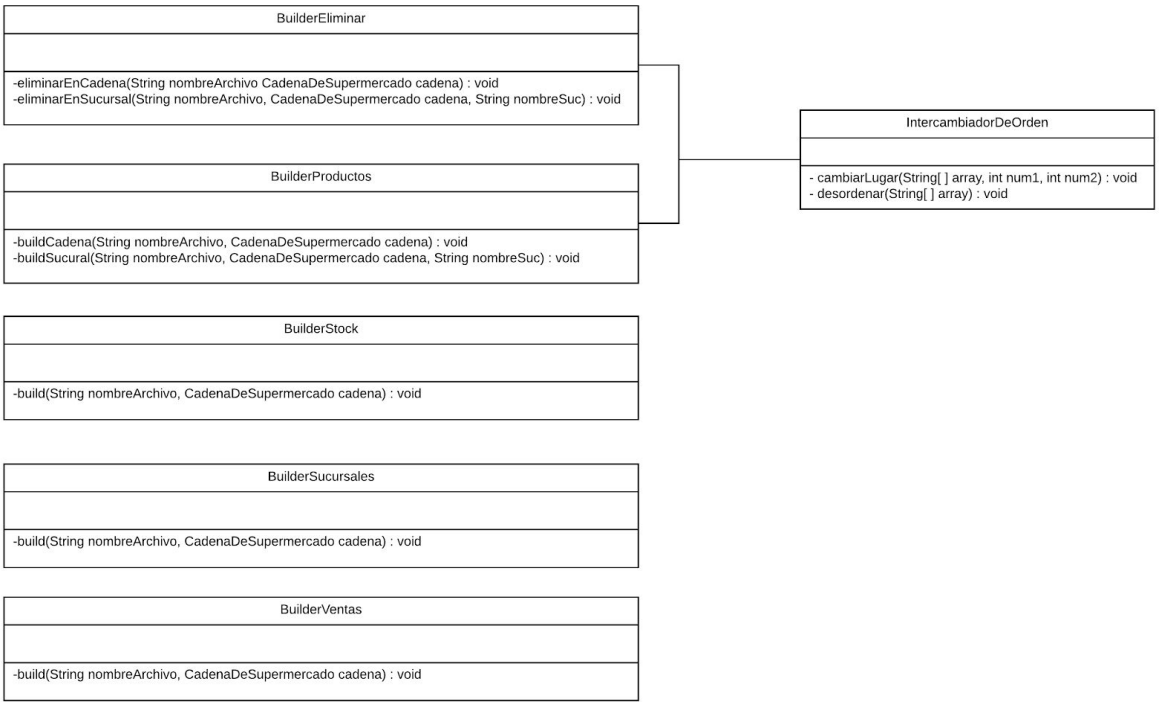




Estructuras básicas:



Builders:



2.3.3 Análisis de costo de memoria

Considerando 10 mil productos y 100 sucursales obtengo el siguiente resultado:

Memoria total utilizada por esta alternativa ≈ 260 MB

Los cálculos específicos se encuentran en el Anexo bajo la sección "*Análisis de costo de memoria de alternativa implementada*".

2.4 Comparativa de costos de memoria de ambas alternativas

Para la alternativa no implementada el costo total de memoria (con la suposiciones planteadas) del sistema software asciende a los 452 MB.

Para la alternativa implementada el costo total de memoria (con la suposiciones planteadas) del sistema software asciende a los 260 MB.

En el caso de la alternativa implementada se ocupa menos memoria ya que pese a que un árbol de productos ocupa más memoria de por sí que una lista de productos (teniendo ambos la misma cantidad de productos) por tener dos referencias en cada TElementoAB frente a una única referencia de la lista por cada Nodo, en esta alternativa se optó por no usar lista de ciudades y de barrios, lo que efectivizó bastante el sistema en cuanto a gasto de memoria.

2.5 Algoritmos

2.5.1 Métodos de la alternativa no implementada

En esta sección presentaré una tabla que muestra cómo serían las precondiciones, postcondiciones y lenguaje natural de los métodos requeridos por la empresa G.EAN.T. en el caso hipotético de que yo implementara esta alternativa.

Método	Descripción del método
Venta de un producto en una sucursal	**Precondiciones:** Ninguna **Postcondiciones:** En caso de que se haga la venta: El producto modifica su estado (reduce su stock). **Lenguaje natural:** Este método se encarga de simular la venta de un producto en una sucursal, reduce su stock si se puede vender o te devuelve los locales en donde la venta sería posible Ejecuto la operación primitiva “buscar” del TDA LISTA en la “lista de sucursales” de la cadena de supermercados para buscar la sucursal en la cual quiero hacer la venta. Si no encuentro la sucursal donde quiero realizar la venta finalizo la operación. Si encuentro la sucursal ejecuto la misma operación “buscar” en la “lista de productos” asociada a esa sucursal. Si encuentro el producto que deseo vender chequeo que tenga el stock necesario, si lo tengo reduzco su stock y finalizo la operación. Si no tengo el stock necesario o no encuentro el producto en la sucursal donde quiero realizar la venta, recorro una por una las sucursales de la “lista de sucursales” de la cadena de supermercados y chequeo todo lo anterior, todas las que tengan stock suficiente de ese producto para las venta las voy agregando por “inserción directa”(de menor a mayor en cuanto a stock) en una lista a devolver al finalizar el método.
Agregar stock a un producto en una cierta sucursal	**Precondiciones:** Ninguna **Postcondiciones:** En caso de que el producto se encuentre en la “lista de productos” de la sucursal en la cual quiero agregar el stock: El producto modifica su estado (aumenta su stock). **Lenguaje natural:** Este método agrega stock a los productos de una cierta sucursal. Ejecuto la operación primitiva “buscar” del TDA LISTA en la “lista de sucursales” de la cadena de supermercados. Si encuentro la sucursal deseada ejecuto la misma operación “buscar” en la “lista de productos” asociada a esa sucursal. Si encuentro el producto al cual deseo agregar stock, modifico su atributo stock. Si no encuentro la sucursal o el producto finalizo la operación.

Dado un código de producto indicar las existencias totales en la cadena de supermercados	**Precondiciones:** Ninguna **Postcondiciones:** Ninguna **Lenguaje natural:** Este método indica el stock total de un producto en toda la cadena de supermercados. Inicializo una variable “stockTotal” en 0. Recorro sucursal por sucursal de la “lista de sucursales” de la cadena de supermercados. En cada una ejecuto la operación primitiva “buscar” del TDA LISTA en su “lista de productos”. Si encuentro el producto sobre el cual deseo contabilizar su stock agrego el stock de ese producto a la variable “stockTotal” y paso a la siguiente sucursal, si no encuentro el producto paso a la siguiente sucursal. Una vez recorridas todas las sucursales devuelvo el “stockTotal” calculado.
Dado un código de producto indicar las existencias del mismo ordenadas por sucursal	**Precondiciones:** Ninguna **Postcondiciones:** Ninguna **Lenguaje natural:** Este método indica el stock de un producto en cada sucursal de la cadena de supermercados. Inicializo una nueva lista “resultado”. Recorro sucursal por sucursal de la “lista de sucursales” de la cadena de supermercados. En cada una ejecuto la operación primitiva “buscar” del TDA LISTA en su “lista de productos”. Si encuentro el producto al cual deseo contabilizar su stock instancio un nuevo nodo (en su atributo etiqueta coloco el nombre de la sucursal y en su dato el stock del producto en esa sucursal) y lo agrego a la lista resultado, luego paso a la siguiente sucursal y repito el procedimiento, si no encuentro el producto paso a la siguiente sucursal. Una vez recorridas todas las sucursales devuelvo la lista resultado.
Listar todos los productos registrados ordenados por nombre y presentando su stock	**Precondiciones:** Ninguna **Postcondiciones:** Ninguna **Lenguaje natural:** Este método genera una lista de productos totales de la cadena de supermercados ordenados por nombre y con su stock correspondiente. Inicializo una nueva lista “resultado”. (Esta lista tendrá nodos con etiqueta = nombre de producto y dato = stock del producto) Recorro sucursal por sucursal de la “lista de sucursales” de la cadena de supermercados. En cada una accedo a su “lista de productos”, chequeo producto por producto si el nombre del producto no está ya en la lista resultado utilizando la operación primitiva “buscar” del TDA LISTA, si no está, instancio un nuevo nodo (de la forma que mencione arriba) y lo inserto mediante una inserción directa en la lista resultado(inserto ordenado por nombre). Si el nombre ya se encuentra en la lista agrego a su stock el stock del producto que estoy recorriendo. Al finalizar el recorrido de los productos de la última sucursal de la “lista de sucursales” devuelvo la lista resultado.

Listar todos los productos registrados en una sucursal, ordenados por nombre y presentando además su stock	**Precondiciones:** Ninguna **Postcondiciones:** Ninguna **Lenguaje natural:** Este método genera una lista de productos de una cierta sucursal ordenados por nombre y con su stock correspondiente en esa sucursal. Inicializo una nueva lista "resultado". (Esta lista tendrá nodos con etiqueta = nombre de producto y dato = stock del producto) Ejecuto la operación primitiva "buscar" del TDA LISTA en la "lista de sucursales" de la cadena de supermercados para buscar la sucursal sobre la cual quiero listar los productos. Si no encuentro la sucursal retorno la lista vacía. Si la encuentro accedo a su "lista de productos" y la recorro producto por producto, para cada producto instancio un nuevo nodo (de la forma que mencione arriba) y lo inserto mediante una inserción directa en la lista resultado(inserto ordenado por nombre). Al finalizar de recorrer todos los productos de la sucursal devuelvo la lista resultado.
Listar todos los productos registrados ordenados por ciudad, presentando además su stock en esa ciudad	**Precondiciones:** Ninguna **Postcondiciones:** La "lista de ciudades" de la cadena de supermercados cambia su estado, cada una de sus ciudades cambia su estado ya que sus "listas de productos" se ven modificadas. **Lenguaje natural:** Este método genera una lista de productos de una cierta ciudad con su stock correspondiente en esa ciudad. Recorro ciudad por ciudad la "lista de ciudades" de la cadena de supermercados. Para cada ciudad recorro la "lista de sucursales" de la cadena de supermercados. En cada sucursal que encuentre que esté situada en esa ciudad específica recorro su lista de productos, chequeo producto por producto si el nombre del producto no está ya en la "lista de productos" asociada a la ciudad en cuestión (utilizando la operación primitiva "buscar" del TDA LISTA), si no está, instancio un nuevo nodo (con etiqueta= nombre de producto y dato= stock del producto en esa ciudad) y lo inserto mediante la operación primitiva "insertar" del TDA LISTA en la "lista de productos" de la ciudad en cuestión. Si el nombre del producto ya se encuentra en la "lista de productos" de la ciudad, agrego a su stock el stock del producto que estoy recorriendo. Repito el procedimiento para cada ciudad de la "lista de ciudades" de la cadena de supermercados.

Listar todos los productos registrados ordenados por barrio, presentando además su stock en ese barrio	**Precondiciones:** Ninguna **Postcondiciones:** La “lista de barrios” de la cadena de supermercados cambia su estado, cada uno de sus barrios cambia su estado ya que sus “listas de productos” se ven modificadas. **Lenguaje natural:** Este método genera una lista de productos de un cierto barrio con su stock correspondiente en ese barrio. Recorro barrio por barrio la “lista de barrios” de la cadena de supermercados. Para cada barrio recorro la “lista de sucursales” de la cadena de supermercados. En cada sucursal que encuentre que esté situada en ese barrio específico recorro su lista de productos, chequeo producto por producto si el nombre del producto no está ya en la “lista de productos” asociada al barrio en cuestión (utilizando la operación primitiva “buscar” del TDA LISTA), si no está, instancio un nuevo nodo (con etiqueta= nombre de producto y dato= stock del producto en esa ciudad) y lo inserto mediante la operación primitiva “insertar” del TDA LISTA en la “lista de productos” del barrio en cuestión. Si el nombre del producto ya se encuentra en la “lista de productos” del barrio, agrego a su stock el stock del producto que estoy recorriendo. Repito el procedimiento para cada barrio de la “lista de barrios” de la cadena de supermercados.
Incorporar un nuevo producto a la cadena de supermercados (a todas las sucursales)	**Precondiciones:** El producto debe tener sus datos correctos (código,descripción,precio) **Postcondiciones:** En caso de que el producto efectivamente se incorpore: Las “listas de productos” asociadas a las sucursales cambian su estado, incorporan un nuevo producto. **Lenguaje natural:** Este método incorpora un nuevo producto a la cadena de supermercados. Instancio el producto a insertar. Recorro sucursal por sucursal de la “lista de sucursales” de la cadena de supermercados. En cada sucursal chequeo si el producto a insertar ya se encuentra en la “lista de productos” asociada a esa sucursal (mediante la operación primitiva “buscar” del TDA LISTA). Si el producto ya está en la “lista de productos” de la sucursal imprimo un error en pantalla y finalizo el método. Si no ésta, creo un nuevo nodo (con etiqueta = código de producto a incorporar y dato= producto a incorporar) y ejecuto la operación “insertar” del TDA LISTA sobre la “lista de productos” con el nodo recientemente creado. Repito la operación hasta terminar de recorrer la “lista de sucursales” de la cadena de supermercados. Finalizo la operación.

Incorporar un nuevo producto a una sucursal específica	**Precondiciones:** El producto debe tener sus datos correctos (código,descripción,precio) **Postcondiciones:** En caso de que el producto efectivamente se incorpore: La “lista de productos” asociada a la sucursal en la cual se incorpora cambia su estado, incorpora un nuevo producto. **Lenguaje natural:** Este método incorpora un nuevo producto a una sucursal específica de la cadena de supermercados. Instancio el producto a insertar. Ejecuto la operación primitiva “buscar” del TDA LISTA en la “lista de sucursales” de la cadena de supermercados. Si encuentro la sucursal deseada ejecuto la misma operación “buscar” en la “lista de productos” asociada a esa sucursal. Si el producto ya está en la “lista de productos” de la sucursal imprimo un error en pantalla y finalizo el método. Si no ésta, creo un nuevo nodo (con etiqueta = código de producto a incorporar y dato= producto a incorporar) y ejecuto la operación “insertar” del TDA LISTA sobre la “lista de productos” de la sucursal, con el nodo recientemente creado. Finalizo la operación.
Eliminar un producto de la cadena	**Precondiciones:** Ninguna **Postcondiciones:** En caso de que el producto efectivamente se elimine: Las “lista de productos” asociadas a las sucursales en la cuales se encontraba el producto cambian su estado, dejan de tener un producto. **Lenguaje natural:** Este método elimina un producto a la cadena de supermercados. Recorro sucursal por sucursal la “lista de sucursales” de la cadena de supermercados. Para cada sucursal ejecuto la operación “eliminar” del TDA LISTA sobre su “lista de productos” con el código del producto que quiero eliminar. Finalizo la operación.
Eliminar un producto de una cierta sucursal	**Precondiciones:** Ninguna **Postcondiciones:** En caso de que el producto efectivamente se elimine: La “lista de productos” asociada a la sucursal en la cual se encontraba el producto cambia su estado, deja de tener un producto. **Lenguaje natural:** Este método elimina un producto de una sucursal específica de la cadena de supermercados. Ejecuto la operación primitiva “buscar” del TDA LISTA para buscar la sucursal donde hacer la eliminación. Si no la encuentro finalizo la operación. Si la encuentro ejecuto la operación “eliminar” del TDA LISTA sobre su “lista de productos” con el código del producto que quiero eliminar. Finalizo la operación.

2.5.2 Métodos de la alternativa implementada

En la alternativa implementada las listas tienen una referencia al último Nodo e insertan al final en 0(1), no debo chequear que los elementos no estén repetidos ya que al elaborar las listas a partir de árboles esto ya está controlado, el insertar de árbol no inserta elementos repetidos.

Para agregar sucursales a la lista de sucursales de la cadena de supermercados si debo controlar que no existan aún en ésta ya que ésta operación se hace directamente desde archivo o de forma manual(por ejemplo si tuviera un árbol de sucursales éste control no sería necesario).

Para los cálculos de orden de tiempo de ejecución se toma N como la cantidad de sucursales y M como la cantidad de productos. Se pondrá en color rojo el peor caso posible, en color azul el mejor y en negro las cosas que sean constantes.

En los pseudocódigos omití los try catch.

Venta de un producto en una sucursal:

Precondiciones: Ninguna
Postcondiciones: En caso de que se haga la venta,el producto modifica su estado (reduce su stock).
Lenguaje natural:
Este método se encarga de simular la venta de un producto en una sucursal, reduce su stock si se puede vender o devuelve los locales en donde la venta sería posible. Busca si el producto se puede vender en la sucursal deseada, si no es posible la venta recorre toda la lista de sucursales y en cada una chequea que sea posible la venta, las sucursales en las cuales se pueda vender el producto se van agregando a un árbol de salida (Se crean elementos con etiqueta=stock de producto y dato= nombre/es de sucursal/es con ese stock)
Pseudocódigo:

- **Venta de un producto en una sucursal:**

Precondiciones: Ninguna
Postcondiciones: En caso de que se haga la venta,el producto modifica su estado (reduce su stock).
Lenguaje natural:
Este método se encarga de simular la venta de un producto en una sucursal, reduce su stock si se puede vender o devuelve los locales en donde la venta sería posible.
Pseudocódigo:

Método de CadenaDeSupermercados:

De CadenaDeSupermercados venderProductoEnSucursal(Comparable codigo, entero cantidad, String suc) : void

```
COM
    output  = nuevo árbol                              O(1)
    aux = listaSucursales.buscar(suc)                  O(N)
    SI (aux <> nulo) ENTONCES                          O(1)
        SI(aux.dato.sePuedeVender(código,cantidad)) ENTONCES O(log M) , O(M)
            aux.dato.vender(código,cantidad)           O(log M) , O(M)
        SI NO
            actual = listaSucursales.primero           O(1)
            MIENTRAS (actual <> nulo) HACER   O(N)
                sucActual = actual.dato                O(1)
                SI(sucActual.sePuedeVender(código,cantidad)) entonces O(log M) , O(M)
                    elem = sucActual.arbolProductos.buscar(codigo) O(log M) , O(M)
                    elem2 = output.buscar(elem.dato.stock)  O(log M) , O(M)
                    SI(elem2 <> nulo) ENTONCES         O(1)
                        elem2.setDato(elem2.dato + ";" + sucActual.nombre)  O(1)
                    SI NO
                        nuevoElem = nuevo TElementoAB (elem.dato.stock,sucActual.nombre)    O(1)
                        output.insertar(nuevoElem)  O(log M) , O(M)
                FIN SI
                actual = actual.siguiente  O(1)
            FIN MIENTRAS
    FIN SI
    Printer.imprimirListaSucursales(output)  O(N)
FIN
```

O(log M) , O(M)

O(N*log M) , O(N*M)

Mejor caso: O(N*log M) ; peor caso: O(N*M)

Métodos de Sucursal:

De Sucursal sePuedeVender(Comparable etiqueta, entero cantidad) : boleano

```
COM
    nodo  arbolProductos.buscar(etiqueta)          O(log M) , O(M)
    SI (nodo <> nulo) ENTONCES                          O(1)
        aux  = nodo.dato                               O(1)
        SI (aux.stock >= cantidad) ENTONCES      O(1)
            RETORNAR VERDADERO                 O(1)
        SI NO
            RETORNAR FALSO                     O(1)
    FIN SI
    RETORNAR FALSO                                      O(1)
FIN
```

Mejor caso: O(log M) ; peor caso: O(M)

De Sucursal sePuedeVender(Comparable etiqueta, entero cantidad) : void

```
COM
    aux =  arbolProductos.buscar(etiqueta)          O(log M) , O(M)
    prod = aux.dato                                 O(1)
    prod.setStock(prod.stock - cantidad)                O(1)
FIN
```

Mejor caso: O(log M) ; peor caso: O(M)

De Printer imprimirListaSucursales(TArbolBB arbol) : void

```
COM
    SI(arbol.esVacio) ENTONCES     O(1)
        IMPRIMIR TEXTO                 O(1)
    SI NO
        listaSuc = arbol.inOrden           O(N)
        productosSuc = nuevo String     O(1)
        aux  = listaSuc.primero            O(1)
        MIENTRAS (aux <> nulo) HACER    O(N)
            productosSuc = aux.dato + ",stock :" + aux.etiqueta + "||"     O(1)
            aux = aux.siguiente     O(1)
        FIN MIENTRAS
        IMPRIMIR productosSuc   O(1)
 FIN
```

Orden total = O(N)

Agregar stock a un producto en una cierta sucursal:

Precondiciones: Ninguna
Postcondiciones: En caso de que el producto se encuentre en la "lista de productos" de la sucursal en la cual quiero agregar el stock, el producto modifica su estado (aumenta su stock).
Lenguaje natural:
Este método agrega stock a los productos de una cierta sucursal. Busca la sucursal en la cual agregar stock y ejecuta su método agregarStock, éste método busca el producto deseado y si existe en el árbol de productos de la sucursal le modifica el stock sumando la cantidad deseada.
Pseudocódigo:

```
De cadenaDeSupermercados agregarStock (Comparable codigo, entero cantidad, String suc) : void
    COM
        SI(cantidad > 0) ENTONCES  O(1)
            aux = listaSucursales.buscar(suc)  O(N)
            aux.dato.agregarStock(codigo,cantidad)  O(log M) , O(M)
        FIN SI
    FIN

Mejor caso: O(log M) ; Peor caso: O(M)
```

```
De Sucursal agregarStock(Comparable codigo, entero cantidad) : booleano
    COM
        elem = arbolProductos.buscar(etiqueta)  O(log M) , O(M)
        SI(elem == null) ENTONCES    O(1)
            RETORNAR FALSO          O(1)
        SI NO
            aux = elem.datos                        O(1)
            aux.setStock(aux.stock + cantidad)      O(1)
            RETORNAR VERDADERO                      O(1)
    FIN

Mejor caso: O(log M) ; Peor caso: O(M)
```

Dado un código de producto indicar las existencias totales en la cadena de supermercados:

Precondiciones: Ninguna
Postcondiciones: Ninguna
Lenguaje natural:
Este método indica el stock total de un producto en toda la cadena de supermercados. Se recorren todas las sucursales de la lista de sucursales y en cada una se busca el producto con la etiqueta ingresada, si se encuentra se suma su stock a una variable que se devuelve al finalizar la operación.
Pseudocódigo:

```
De cadenaDeSupermercados indicarExistenciasTotales(Comparable Etiqueta) : void
    COM
        result = 0 (nuevo entero)  O(1)
        actual = listaSucursales.primero  O(1)
        MIENTRAS(actual <> nulo) HACER  O(N)
            suc = actual.dato   O(1)
            aux = suc.arbolProductos.buscar(Etiqueta) O(log M), O(M)
            SI(aux <> nulo)
                result = result + aux.dato.stock      O(1)
            FIN SI
            actual = actual.siguiente       O(1)
        FIN MIENTRAS
        Printer.imprimirExistenciasTotales(Etiqueta,result)   O(1)
    FIN

Mejor caso: O(N*log M) ; Peor caso: O(N*M)
```

Dado un código de producto indicar las existencias del mismo ordenadas por sucursal:

Precondiciones: Ninguna
Postcondiciones: Ninguna
Lenguaje natural:
Este método indica el stock de un producto en cada sucursal de la cadena de supermercados. Se recorren todas las sucursales de la lista de sucursales buscando en cada una el producto con la etiqueta ingresada, si el producto se encuentra se genera un nuevo Nodo(nombreSuc,stockProd) y se agrega a una lista a devolver. Al final se imprime en consola la lista.
Pseudocódigo:

```
De cadenaDeSupermercados indicarExistenciasPorSucursal(Comparable etiqueta) : void
    COM
        result = nueva Lista O(1)
        actual = listaSucursales.primero O(1)
        MIENTRAS(actual <> nulo) HACER O(N)
            suc = actual.dato O(1)
            aux = suc.arbolProductos.buscar(etiqueta) O(log M), O(M)
            SI(aux <> nulo) O(1)
                nodo = nuevo Nodo (suc.nombre,aux.datos.stock) O(1)
                result.insertar(nodo) O(1)
            FIN SI
            actual = actual.siguiente O(1)
        FIN MIENTRAS
        SI(result.esVacia) ENTONCES  O(1)
            IMPRIMIR TEXTO  O(1)
        SI NO
            Printer.imprimirExistenciasPorSucursal(etiqueta,result)   O(N)
    FIN

Mejor caso: O(N*log M) ; Peor caso: O(N*M)

De printer imprimirExistenciasPorSucursal(Comparable codigo,Lista lista)
    COM
        aux = lista.primero   O(1)
        MIENTRAS(aux <> nulo) HACER  O(N)
            IMPRIMIR TEXTO   O(1)
            aux = aux.siguiente  O(1)
        FIN MIENTRAS
    FIN

Orden :  O(N)
```

Listar todos los productos registrados ordenados por nombre y presentando su stock

Precondiciones: Ninguna
Postcondiciones: Ninguna
Lenguaje natural:
Este método genera un archivo de productos totales de la cadena de supermercados ordenados por nombre y con su stock correspondiente. Se genera un árbol de salida y se recorren todos los productos de todas las sucursales, si el elemento recorrido ya está en el árbol de salida se le agrega el stock correspondiente, si no está se genera un nuevo elemento(nombreProd,stockProd) y se inserta al árbol. Al final se escribe un archivo txt con los productos del árbol recorridos en inOrden.
Pseudocódigo:

```
De cadenaDeSupermercados productosTotalesOrdenadosPorNombre(String nombreArchivo) : void
    COM
        aux = listaSucursales.primero  O(1)
        salida = nuevo TArbolBB  O(1)
        MIENTRAS(aux <> nulo) HACER  O(N)
            listaAux = aux.dato.arbolProductos.inOrden O(M)
            actual = listaAux.primero O(1)
            MIENTRAS (actual <> nulo) HACER O(M)
                prod = actual.dato O(1)
                elem2 = salida.buscar(prod.nombre) O(log M), O(M)
                SI(elem2 <> nulo) ENTONCES  O(1)
                    elem2.setDatos(elem2.datos + prod.stock) O(1)
                SI NO
                    elem3 = nuevo TElementoAB(prod.nombre,prod.stock) O(1)
                    salida.insertar(elem3) O(log M), O(M)
                actual = actual.siguiente O(1)
            FIN MIENTRAS
            aux = aux.siguiente  O(1)
        FIN MIENTRAS
        array = salida.inordenQueDevuelveArray O(M) (recorrido en inOrden que retorna array)
        ManejadorArchivosGenerico.escribirArchivo(ruta,array)  O(M) (lineas de archivo = cantidad de productos)
    FIN

Mejor caso: O( M^2 * N * log M) ; Peor caso: O( M^3 * N)
```

Listar todos los productos registrados en una sucursal, ordenados por nombre y presentando además su stock

Precondiciones: Ninguna
Postcondiciones: Ninguna
Lenguaje natural:
Este método genera un archivo de productos de una cierta sucursal ordenados por nombre y con su stock correspondiente en esa sucursal. Se busca la sucursal deseada, se devuelve a partir de su árbol de productos un nuevo árbol con los productos ordenados por nombre y se genera un archivo salida en txt con los productos recorridos en inOrden.
Pseudocódigo:

```
De cadenaDeSupermercados productosSucursalOrdenadosPorNombre(String suc, String nombre) : void
    COM
        aux = listaSucursales.buscar(suc) O(N)
        salida = aux.dato.arbolProductos.inordenQueDevuelveArbolPorNombre() O(M*log M), O(M^2)  (Recorre el arbol y va insertando en un arbol salida)
        array = salida.inordenQueDevuelveArray()  O(M)
        ManejadorArchivos.escribirArchivo(ruta,array)  O(M)
    FIN

Mejor caso: O(máximo(N,M)) ;  Peor caso: O(máximo(N,M^2))
```

Listar todos los productos registrados ordenados por ciudad, presentando además su stock en esa ciudad

Precondiciones: Ninguna
Postcondiciones: Ninguna
Lenguaje natural:
Este método genera un archivo con Ciudades y los productos que se comercializan en ellas con su stock en las mismas. Se recorre sucursal por sucursal de la lista de sucursales, en cada una se chequea si la ciudad de la misma no está aún en la lista de salida, si no está se recorren todos los productos de la sucursal, generando un árbol por nombre y se inserta un nodo a la lista salida con etiqueta= nombre ciudad y dato = árbol de productos de sucursal recorrida ordenados por nombre, si la ciudad ya está en la lista de salida se van recorriendo los productos de la sucursal y agregando stock a los del árbol del nodo correspondiente en la lista de salida. Al final se manda todo a un archivo txt.
Pseudocódigo:

```
De cadenaDeSupermercados productosTotalesOrdenadosPorCiudad(String rutaArchivo)
    COM
        contadorLineas = 0 (nuevo entero)     O(1)
        aux = listaSucursales.primero      O(1)
        lista = nueva Lista de TArbolBB       O(1)
        MIENTRAS(aux <> nulo) HACER      O(N)
            ciudadActual = lista.buscar(aux.dato.ciudad)   O(N)
            listaAux = aux.dato.arbolProductos.inOrden()   O(M)
            SI (ciudadActual <> nulo) ENTONCES   O(1)
                nodoActual = listaAux.primero          O(1)
                MIENTRAS (nodoActual <> nulo) HACER    O(M)
                    prod = nodoActual.dato       O(1)
                    elem = ciudadActual.dato.buscar(prod.nombre)  O(log M) , O(M)
                    SI(elem <> nulo) ENTONCES   O(1)
                        elem.setDatos(elem.datos + prod.getStock)  O(1)
                    SI NO
                        contadorLineas = contadorLineas + 1    O(1)
                        elem2 = nuevo TElementoAB(prod.nombre,prod.stock)   O(1)
                        ciudadActual.dato.insertar(elem2)   O(log M) , O(M)
                    nodoActual = nodoActual.siguiente  O(1)
                FIN MIENTRAS
            SI NO
                contadorLineas = contadorLineas + 1  O(1)
                arbol = nuevo TArbolBB    O(1)
                nodoActual = listaAux.primero   O(1)
                MIENTRAS (nodoActual <> nulo) HACER  O(M)
                    prod = nodoActual.dato   O(1)
                    elem3 = nuevo TElementoAB(prod.nombre,prod.stock)  O(1)
                    SI (arbol.buscar(prod.nombre) == nulo) ENTONCES   O(log M) , O(M)
                        arbol.insertar(elem3)  O(log M) , O(M)
                        contadorLineas = contadorLineas + 1  O(1)
                    FIN SI
                    nodoActual = nodoActual.siguiente O(1)
                FIN MIENTRAS
                ciudadNueva = nuevo Nodo(aux.dato.ciudad, arbol)   O(1)
                lista.insertar(ciudadNueva) O(N)
            aux = aux.siguiente   O(1)
        FIN MIENTRAS
        array = GeneradorArray.generarArray(lista,contadorLineas) O(N*M) (Método que recibe una lista de arboles y un contador y genera un array)
        ManejadorArchivosGenerico.escribirArchivo(ruta, array)  O(N*M)
    FIN

Mejor caso : O(máximo(N^2,N*M*log M)) ; Peor caso: O(máximo(N^2,N*M^2))
```

Listar todos los productos registrados ordenados por barrio, presentando además su stock en ese barrio

Precondiciones: Ninguna
Postcondiciones: Ninguna
Lenguaje natural:
Este método genera un archivo con barrios y los productos que se comercializan en ellos con su stock en los mismos. Es un método análogo al anterior
Pseudocódigo:
El pseudocódigo es análogo al anterior pero en vez de usar el atributo ciudad se usa el atributo barrio. Los órdenes de tiempo de ejecución son idénticos

Incorporar un nuevo producto a la cadena de supermercados (a todas las sucursales)

Precondiciones:
El producto debe tener sus datos correctos (código,descripción,precio)
Postcondiciones:
En caso de que el producto efectivamente se incorpore:
Las "listas de productos" asociadas a las sucursales cambian su estado, incorporan un nuevo producto.
Lenguaje natural:
Este método incorpora un nuevo producto a la cadena de supermercados. Recorre las sucursales una por una y en cada una chequea si el producto está, si no está lo incorpora.
Pseudocódigo:

```
De cadenaDeSupermercados incorporarProductoEnCadena(Producto prod) : void
    COM
        actual = listaSucursales.primero  O(1)
        MIENTRAS(actual <> nulo) HACER  O(N)
            suc = actual.dato   O(1)
            SI (suc.buscarPorCodigo(prod.etiqueta) == null) ENTONCES  O(log M), O(M)
                suc.insertarProducto(prod.clonar())   O(log M), O(M)
            FIN SI
            actual = actual.siguiente  O(1)
        FIN MIENTRAS
    FIN

Mejor caso: O(N* log M) ;  Peor caso: O(N*M)
```

```
De Sucursal insertarProducto(Producto prod)
    COM
        prodAux = buscarPorCodigo(prod.etiqueta)  O(log M), O(M)
        SI (prodAux <> nulo) ENTONCES   O(1)
            aux = nuevo TElementoAB(prod.etiqueta,prod)  O(1)
            arbolProductos.insertar(aux) O(log M), O(M)
        FIN SI
    FIN

Mejor caso: O(log M) ;  Peor caso: O(M)
```

Incorporar un nuevo producto a una sucursal específica

Precondiciones:
El producto debe tener sus datos correctos (código,descripción,precio)
Postcondiciones:
En caso de que el producto efectivamente se incorpore:
La "lista de productos" asociada a la sucursal en la cual se incorpora cambia su estado, incorpora un nuevo producto.
Lenguaje natural:
Este método incorpora un nuevo producto a una sucursal específica de la cadena de supermercados. Se busca la sucursal y si se encuentra se inserta el producto.
Pseudocódigo:

```
De cadenaDeSupermercados incorporarProductoEnSucursal(Producto prod, String suc)
    COM
        aux = listaSucursales.buscar(suc) O(N)
        aux.dato.insertarProducto(prod) O(log M) , O(M)
    FIN

Mejor caso: O(máximo(N,log M)) ;  Peor caso: O(máximo(N,M))
```

Eliminar un producto de la cadena

Precondiciones: Ninguna
Postcondiciones:
En caso de que el producto efectivamente se elimine:
Las "lista de productos" asociadas a las sucursales en la cuales se encontraba el producto cambian su estado, dejan de tener un producto.
Lenguaje natural:
Este método elimina un producto a la cadena de supermercados. Recorre sucursal por sucursal y eliminar el producto de cada una.
Pseudocódigo:

```
De cadenaDeSupermercados eliminarProductoEnCadena(Comparable etiqueta) : void
    COM
        actual = listaSucursales.primero  O(1)
        MIENTRAS(actual <> nulo) HACER  O(N)
            suc = actual.dato  O(1)
            SI (suc.arbolProductos.buscar(etiqueta) <> nulo) ENTONCES  O(log M), O(M)
                suc.eliminarProducto(etiqueta)  O(log M), O(M)
            FIN SI
            actual = actual.siguiente;  O(1)
        FIN MIENTRAS
    FIN

Mejor caso: O(N*log M) ;  Peor caso O(N*M)
```

Eliminar un producto de una cierta sucursal

Precondiciones: Ninguna
Postcondiciones:
En caso de que el producto efectivamente se elimine:
La “lista de productos” asociada a la sucursal en la cual se encontraba el producto cambia su estado, deja de tener un producto.
Lenguaje natural:
Este método elimina un producto de una sucursal específica de la cadena de supermercados.
Busca la sucursal en la cual eliminar y elimina el producto.
Pseudocódigo:

```
De cadenaDeSupermercados eliminarProductoEnSucursal(Comparable codigo, String suc) : void
    COM
        aux = listaSucursales.buscar(suc) O(N)
        aux.dato.eliminarProducto(codigo) O(log M), O(M)
    FIN

Mejor caso: O(máximo(N,log M)) ; Peor caso: O(máximo(N,M))
```

2.6 Tabla comparativa de ordenes de tiempo de ejecución

Siendo N la cantidad de sucursales y M la cantidad de productos

(Las explicaciones de como llegue a los valores de orden en el caso de la alternativa no implementada están en el anexo bajo la sección "*Cálculo de ordenes de tiempo de ejecución para alternativa no implementada*".

Cabe destacar que en el caso de la alternativa implementada en el peor de los casos los árboles degeneran en listas, pero al desordenar los archivos previo a hacer incorporaciones o eliminaciones esto es muy poco probable. Por ende los órdenes "reales" estarían entre el peor y el mejor caso pero más cerca del mejor caso que del peor. De acuerdo a lo visto en clase los árboles quedarían bastante equilibrados con el randomizado de archivos, posiblemente tendrían cerca de un 39% menos de eficiencia que un árbol completamente balanceado.

Método	Alternativa no implementada	Alternativa implementada
Venta de un producto en una sucursal	O(máximo((N*M), N^3)	Peor caso:O(M*N) Mejor caso: O(N*log M)
Agregar stock a un producto en una sucursal	O(máximo(N,M))	Peor caso: O(M) Mejor caso: O(log M)
Dado un código de producto indicar las existencias totales en la cadena de supermercados	O(N*M)	Peor caso:O(M*N) Mejor caso:O(N*log M)
Dado un código de producto indicar las existencias del mismo en todas las sucursales ordenadas por sucursal	O(N*M)	Peor caso:O(M*N) Mejor caso:O(N*log M)

Listar todos los productos registrados ordenados por nombre y presentando su stock	O(M^4 * N)	Peor caso:(M^3 * N) Mejor caso:O(M^2 * N * log M)
Listar todos los productos registrados en una sucursal, ordenados por nombre y presentando además su stock	O(M^3)	Peor caso:O(máximo(N, M^2)) Mejor caso: O(máximo(N,M))
Listar todos los productos registrados ordenados por ciudad, presentando además su stock en esa ciudad	O(M^2 * N^2)	Peor caso:O(máximo(N^2 ,N* M^2)) Mejor caso: O(máximo(N^2 ,N*M*log M))
Listar todos los productos registrados ordenados por barrio, presentando además su stock en ese barrio	O(M^2 * N^2)	Peor caso:O(máximo(N^2 ,N* M^2)) Mejor caso: O(máximo(N^2 ,N *M*log M))
Incorporar un nuevo producto a todas las sucursales	O (N*M)	Peor caso: O (N*M) Mejor caso: O (N*log M)
Incorporar un nuevo producto a una sucursal específica	O(máximo(N,M))	Peor caso: O(máximo(N,M)) Mejor caso: O(máximo(N,log M))
Eliminar un producto de la cadena	O(N*M)	Peor caso: O(N*M) Mejor caso: O(N*log M)
Eliminar un producto de una sucursal	O(máximo(N,M))	Peor caso: O(máximo(N,M)) Mejor caso: O(máximo(N,log M))

3. Selección y justificación de alternativa a implementar

Como se puede apreciar en la tabla comparativa de órdenes de tiempo de ejecución, la alternativa implementada es ampliamente más eficiente que la no implementada, y con los cambios realizados en cuanto a quitar las listas de ciudades y barrios e ir generando la lista de salida en los propios métodos, se consigue un ahorro en costo de memoria.

Asumiendo que las sucursales no van a ser cientos de miles, no veo una necesidad clara de hacer un árbol de sucursales; considero que con tener una lista de sucursales es suficiente. La lista es una estructura simple y nos facilita mucho el trabajo a la hora de realizar el propio sistema.

Asimismo en el caso hipotético en el cual se usara un árbol de sucursales, los métodos que tienen que recorrer todas las sucursales no se verían beneficiados por este cambio, no modificarían su orden en tiempo de ejecución. Y los que tienen que hacer algo en una sucursal específica no se beneficiarían casi nada tampoco (ya que el O(N) de encontrar la sucursal es mucho menor que el orden del resto de las operaciones de estos métodos). No agregaría demasiada eficiencia y costaría más esfuerzo a la hora de la implementación.

En las operaciones de emitir listados de productos se utilizó el manejador de archivos para crear archivos txt con los listados. Observé que la consola era bastante más lenta a la hora de emitir los listados que el hecho de sacar los archivos por txt. Esto efectiviza bastante las emisiones de listados y además, los mismos quedan guardados para que el usuario los chequee directamente.

Es importante destacar que el sistema software cuenta con la función añadida de desordenar las líneas de los archivos txt de entrada para el caso de las operaciones de incorporar y eliminar productos, esto casi “asegura” que los árboles de productos no resulten demasiado desbalanceados luego de introducir o eliminar productos, esta funcionalidad es muy útil ya que el usuario puede utilizar archivos ordenados por código y directamente cargarlos al sistema, éste no disminuirá su eficiencia ni tendrá problemas a la hora de cargarlos. (En los demás métodos este desordenamiento no es necesario ya que los mismos no modifican la estructura de los árboles de productos de las sucursales).

Teniendo en cuenta los puntos planteados, la complejidad de la solución, su gasto en memoria y su eficiencia en tiempo de ejecución creo que la alternativa tomada es más que correcta para los requerimientos planteados.

4. Conclusiones

El producto cumple con todos los requerimientos planteados por la empresa y en algunos casos añade la posibilidad de realizar las operaciones tanto de forma manual como por archivos lo que efectivizaría los trabajos dentro de la cadena de supermercados. Creo que es un sistema conciso, confiable y que cumple con las demandas solicitadas. A su vez, en caso de querer agregar nuevas funcionalidades, esto se podría hacer sin demasiado problema. Por otra parte no ocupa demasiado espacio en memoria y realiza las operaciones con gran rapidez, cosa que es sumamente necesaria sobretodo cuando hablamos de una cadena de supermercados que además cuenta con varios establecimientos. Además tiene una interfaz de usuario simple y fácil de usar. Por lo que, desde mi punto de vista es una solución más que adecuada para el problema planteado.

5. Guía de usuario

La computadora donde se ejecute el sistema software debe tener instalada la JVM (Java virtual machine) y el programa NetBeans.

La instalación de JVM puede hacerse desde este link: https://www.java.com/es/download/

La instalación de NetBeans puede hacerse desde este link:
https://netbeans.apache.org/download/index.html (Se recomienda descargar la última versión LTS (Long Term Support)).

Abrir el proyecto en NetBeans y darle a ejecutar, aparecerá un menú con todas las opciones disponibles en el sistema software.

Recordar que para hacer operaciones en las cuales se utilizan archivos es sumamente importante respetar los formatos especificados en el propio menú de cada opción, de lo contrario el sistema no funcionará correctamente.

Todos los listados emitidos se encontrarán en la carpeta “Archivos” dentro del proyecto con los nombres que se asignen por parte del usuario.

Además en ésta carpeta se encuentra un archivo llamado “ventasPrueba” que sirve para simular ventas desde archivo y un archivo “elimPrueba” que sirve para eliminar productos desde archivo, estos archivos y los demás que se

encuentran en la carpeta están puestos a modo de ejemplo, la idea es que el usuario genere los suyos propios y ejecute las operaciones que requiera.

6. Anexo

6.1 Cálculo de ordenes de tiempo de ejecución para alternativa no implementada

Para los cálculos de orden en tiempo de ejecución se asumirá N como la cantidad de sucursales y M como la cantidad de productos. (Hay también N ciudades y N barrios).

- **Venta de un producto en una sucursal:**

  → El peor caso se da cuando encuentro la sucursal, encuentro el producto y NO tengo stock suficiente para la venta o directamente no encuentro el producto en esa sucursal:

Al hacer el "buscar" del TDA LISTA en la "lista de sucursales" tengo O(N), al hacer el "buscar" del TDA LISTA en la "lista de productos" tengo O(M), cuando veo que no hay stock suficiente debo recorrer la "lista de sucursales" una por una y en cada una chequear que se pueda hacer la venta (N veces O(M)), suponiendo el peor caso en el cual todas las sucursales restantes tengan el producto y lo puedan vender, debo elaborar una lista salida en orden por cantidad de producto (el ordenamiento por inserción directa tiene O(N^2)), entonces me queda : N veces O(M) + N veces O(N^2).

El orden final queda de la forma : O(máximo((N*M), N^3)

- **Agregar stock a un producto existente en una cierta sucursal:**

→ El peor caso es en el cual encuentro la sucursal (encuentre o no el producto):

Al hacer el "buscar" del TDA LISTA en la "lista de sucursales" tengo O(N), al hacer el "buscar" del TDA LISTA en la "lista de productos" tengo O(M), aumentar el stock es O(1)

El orden final queda de la forma : O(máximo(N,M))

- **Dado un código de producto indicar las existencias totales en la cadena de supermercados:**

Debo recorrer sucursal por sucursal de la “lista de sucursales” O(N), al hacer el “buscar” del TDA LISTA en la “lista de productos” tengo O(M), aumentar la variable stockTotal es O(1)

El orden final queda de la forma : O(N*M)

- **Dado un código de producto indicar las existencias del mismo en todas las sucursales ordenadas por sucursal:**

→ El peor caso es el caso en el cual el producto está en todas las sucursales

Recorro sucursal por sucursal en la “lista de sucursales” O(N), al hacer el “buscar” del TDA LISTA en la “lista de productos” tengo O(M), debo elaborar una lista salida por sucursal (sabiendo que las sucursales no se repiten cada dato se inserta al final de la lista salida con la referencia al último O(1)), entonces me queda :

N veces (O(M) * O(1))

El orden final queda de la forma : O(N*M)

- **Listar todos los productos registrados ordenados por nombre y presentando su stock**

→ El peor caso sería el caso en el cual todas las sucursales tengan todos los productos:

Debo recorrer sucursal por sucursal de la “lista de sucursales” O(N), insertando uno por uno los productos registrados en cada sucursal en la lista salida (Previamente tengo que chequear que el producto no esté ya en la lista de salida, O(M)) y en orden por nombre(inserción directa) O(M)*O(M^2), los órdenes se multiplican y queda de la forma:

O(N) * O(M) * O(M) * O(M^2)

El orden final queda de la forma: O(M^4 * N)

- **Listar todos los productos registrados en una sucursal, ordenados por nombre y presentando además su stock:**

→ El peor caso sería el caso en el cual la sucursal tenga M productos:

Al hacer el “buscar” del TDA LISTA en la “lista de sucursales” tengo O(N), insertando uno por uno los productos registrados en la sucursal en la lista salida y en orden por nombre(inserción directa) O(M)*O(N^2) (en este caso no tengo que chequear que los productos estén ya en al lista de salida ya que al incorporar productos a la lista de productos de una sucursal ya chequeo y no inserto repetidos), se desprecia el O(N) y los demás órdenes se multiplican y queda de la forma:

O(N) + O(M) * O(M^2)

El orden final queda de la forma: O(M^3)

- **Listar todos los productos registrados ordenados por ciudad, presentando además su stock en esa ciudad:**

→ El peor caso sería el caso en el cual todas las ciudades tengan todos los productos, además cada sucursal esté en una ciudad distinta (N ciudades)

Debo recorrer ciudad por ciudad de la “lista de ciudades” O(N), para cada ciudad debo recorrer sucursal por sucursal de la “lista de sucursales” O(N), y recorrer producto por producto de cada sucursal O(M). Luego insertar uno por uno los productos en cada “lista de productos” de la ciudad en cuestión (previamente debo chequear que el producto no este ya en la lista, O(M)), si no está se inserta en orden 1.

O(N) * O(N) * O(M) * O(M)

El orden final queda de la forma : O(M^2 * N^2)

- **Listar todos los productos registrados ordenados por barrio, presentando además su stock en ese barrio:**

→ Análogo al de ciudad, el peor caso sería el caso en el cual todos los barrios tengan todos los productos y además cada sucursal este en un distinto barrio.

El orden final queda de la forma : O(M^2 * N^2)

- **Incorporar un nuevo producto a todas las sucursales:**

Debo recorrer sucursal por sucursal O(N), y en cada buscar que el producto no esté ya agregado (O(M)) y si no está insertarlo (puedo insertarlo con referencia al ultimo O(1))

El orden final queda de la forma : O (N*M)

- **Incorporar un nuevo producto a una sucursal específica:**

→ El peor caso es el caso en el cual la sucursal a la cual quiero incorporar el producto es la última de la lista de sucursales
Análogo al anterior pero solo debo hacerlo en una sucursal

El orden final queda de la forma : O(máximo(N,M))

- **Eliminar un producto de la cadena:**

→ El peor caso es cuando el producto a eliminar se encuentra al final de las listas de productos de cada sucursal de la lista de sucursales

Recorro sucursal por sucursal de la lista de sucursales de la cadena de supermercados O(N), y en cada sucursal utilizo la operación primitiva “eliminar” del TDA Lista en la “lista de productos” tengo O(M).

El orden final queda de la forma: O(N*M)

- **Eliminar un producto de una cierta sucursal:**

→ El peor caso es cuando el producto a eliminar se encuentra al final de la lista de productos de la sucursal y la sucursal se encuentra al final de la lista de sucursales.

Busco la sucursal con la operación primitiva “buscar” del TDA Lista O(N), y en la sucursal utilizo la operación primitiva “eliminar” del TDA Lista en la “lista de productos” tengo O(M).

El orden final queda de la forma: O(máximo(N,M))

6.2 Análisis de costo de memoria de alternativa no implementada

- Memoria utilizada por un producto
  → Por ser un Object: 8 bytes
  → Nombre: 72 bytes
  → Código: 72 bytes
  → Precio: 8 bytes
  → Stock: 16 bytes
  Memoria total utilizada por un producto ≈ 176 bytes

- Memoria utilizada por un “Nodo producto” (Nodo con etiqueta = código de producto y dato = producto)
  → Por ser un Object: 8 bytes
  → Etiqueta: 72 bytes
  → Producto: 176 bytes
  → Referencia al siguiente: 8 bytes (En un SO de 64 bits)
  Memoria total utilizada por un “Nodo Producto” ≈ 264 bytes

- Memoria utilizada por una Lista de productos (Nodos con productos adentro)
  → (Memoria de un “Nodo producto”)* cantProductos + referencia al último producto de la lista + 8 bytes (Por ser un Object)
  → (264 bytes) * 10000 + 8 bytes + 8 bytes
  Memoria total utilizada por una Lista de productos ≈ 2,52 MB

- Memoria utilizada por una sucursal:
  → Por ser un Object: 8 bytes
  → Nombre: 72 bytes
  → Teléfono: 56 bytes
  → Dirección: 72 bytes
  → Barrio: 72 bytes
  → Ciudad: 72 bytes
  → Lista de productos: 2,52 MB
  Memoria total de una sucursal ≈ 2,52 MB

- Memoria utilizada por un "Nodo sucursal" (Nodo con etiqueta = nombre de sucursal y dato = sucursal)
  → Por ser un Object: 8 bytes
  → Etiqueta: 72 bytes
  → Sucursal: 2,52 MB
  → Referencia al siguiente: 8 bytes
  Memoria total utilizada por un "Nodo Sucursal" ≈ 2,52 MB

- Memoria utilizada por una Lista de sucursales:
  → (Memoria de un "Nodo sucursal")* cantSucursales + referencia a la última sucursal de la lista + 8 bytes (Por ser un Object)
  → ((2,52 MB) * 100) + 8 bytes + 8 bytes
  Memoria total utilizada por una Lista de sucursales ≈ 252 MB

- Memoria utilizada por una Lista de productos (etiqueta=código,dato=stock)
  → Por ser un Object: 8 bytes
  → Código: 72 bytes
  → Stock: 16 bytes
  → Referencia al siguiente : 8 bytes
  Memoria total utilizada por una Lista de productos de estas características ≈ (104 bytes)*10000 + 8 bytes(referencia la ultimo) + 8 bytes (Por ser un Object) = 1 MB

- Memoria utilizada por un "Nodo ciudad" (Nodo con etiqueta = nombre de ciudad y dato = lista de productos(etiqueta= código, dato=stock))

→ Por ser un Object: 8 bytes
→ Etiqueta: 72 bytes
→ Lista de productos: 1 MB
→ Siguiente: 8 bytes
Memoria total utilizada por un “Nodo Ciudad” ≈ 1 MB

- Memoria utilizada por una Lista de ciudades
  → (Memoria de un “Nodo ciudad”)* cantCiudades + referencia a la última ciudad de la lista + 8 bytes (Por ser un Object)
  → ((1 MB) * 100) + 8 bytes + 8 bytes
  Memoria total utilizada por una Lista de ciudades ≈ 100 MB

- Memoria utilizada por un “Nodo barrio” (Nodo con etiqueta = nombre de barrio y dato = lista de productos):
  → Análoga a “Nodo ciudad”
  Memoria total utilizada por un “Nodo barrio” ≈ 1 MB

- Memoria utilizada por una lista de barrios:
  → Análoga a la “lista de ciudades”
  Memoria total utilizada por una Lista de barrios ≈ 100 MB

Memoria total = 100 MB + 100 MB + 252 MB = 452 MB

6.3 Análisis de costo de memoria de alternativa implementada

- Memoria utilizada por un producto
  → Por ser un Object: 8 bytes
  → Nombre: 72 bytes
  → Código: 72 bytes
  → Precio: 8 bytes
  → Stock: 16 bytes
  Memoria total utilizada por un producto ≈ 176 bytes

- Memoria utilizada por un “TElementoAB producto” (TElementoAB con etiqueta = código de producto y dato = producto)
  → Por ser un Object: 8 bytes
  → Etiqueta: 72 bytes
  → Producto: 176 bytes

→ Referencia al hijoIzq: 8 bytes (En un SO de 64 bits)
→ Referencia al hijoDer: 8 bytes (En un SO de 64 bits)
Memoria total utilizada por un “TElementoAB Producto” ≈ 272 bytes

- Memoria utilizada por un TArbolBB de productos (TElementosAB con productos adentro)
  → (Memoria de un “TElementoAB producto”)* cantProductos + 8 bytes (Por ser un Object)
  → (272 bytes) * 10000 + 8 bytes
  Memoria total utilizada por un TArbolBB de productos ≈ 2,6 MB

- Memoria utilizada por una sucursal:
  → Por ser un Object: 8 bytes
  → Nombre: 72 bytes
  → Teléfono: 56 bytes
  → Dirección: 72 bytes
  → Barrio: 72 bytes
  → Ciudad: 72 bytes
  → Arbol de productos: 2,6 MB
  Memoria total de una sucursal ≈ 2,6 MB

- Memoria utilizada por un “Nodo sucursal” (Nodo con etiqueta = nombre de sucursal y dato = sucursal)
  → Por ser un Object: 8 bytes
  → Etiqueta: 72 bytes
  → Sucursal: 2,6 MB
  → Referencia al siguiente: 8 bytes
  Memoria total utilizada por un “Nodo Sucursal” ≈ 2,6 MB

- Memoria utilizada por una Lista de sucursales:
  → (Memoria de un “Nodo sucursal”)* cantSucursales + referencia a la última sucursal de la lista + 8 bytes (Por ser un Object)
  → ((2,6 MB) * 100) + 8 bytes + 8 bytes
  Memoria total utilizada por una Lista de sucursales ≈ 260 MB

Para ésta esta alternativa obvie el uso de listas de ciudades y barrios y directamente cree las salidas necesarias (de productos ordenados por ciudad o por

barrio) en los métodos correspondientes, por ende me ahorro ese espacio en memoria.

Memoria total = 260 MB