1. Preguntas Teóricas Implementaciones Stock

1.1. StockSequence

1.1.1. ¿Qué tipo de secuencia sería la más adecuada para devolver el resultado de la operación listStock(prefix)? ¿Qué consecuencias tendría el uso de otro tipo de secuencias? Razona tu respuesta.

La secuencia más adecuada para la implementación del método listStock(prefix) es la cola (Queue), porque la única función que tenemos que hacer es enlistar productos mediante el prefix, pero una vez los enlistes, no hay que manipularlos, no necesitas que se queden ahí , para nada. Que vayan desapareciendo una vez los proporciones al usuario hace que te ahorres bastante rendimiento. La lista serviría pero no sería tan eficiente porque todos los elementos quedarían ahí, sería mas eficiente la lista si después enlistar quisiésemos actualizar esos productos en concreto o cualquier otra función. La pila es la que sería menos eficiente en cualquier caso, para enlistar y luego para manipular, no sería nada eficiente.

1.1.2. ¿Qué tipo de secuencia sería la adecuada para realizar esta implementación? ¿Por qué? ¿Qué consecuencias tendría el uso de otro tipo de secuencias?

Para realizar esta implementación, una secuencia adecuada sería una lista(List). Esto se debe a que se necesita una estructura de datos que permita facilidad acceder a elementos de la secuencia y manipularlos. Si para actualizar el número de elementos tenemos que volver a crear otra secuencia auxiliar para almacenar y la otra para editar gastamos mucha eficiencia. Una lista no suele ser más eficiente que una pila o una cola, pero para esta implementación, tenemos que usar 3 colas o 3 pilas para que funcione frente a una sola lista; la lista acaba siendo más eficiente.

1.1.3. ¿Cuál sería el orden adecuado para almacenar los pares en la secuencia? ¿Por qué? ¿Qué consecuencias tendría almacenarlos sin ningún tipo de orden?

El orden adecuado para almacenar los pares es alfabéticamente por el nombre de los índices. Porque no tiene sentido que una persona busque un producto por el número de unidades que tenga, sino por el propio producto en sí. Alfabéticamente es un buen orden también porque facilita las operaciones a la hora de la implementación, por ejemplo, en el método listStock habría que ordenar la secuencia cada vez que se llame a este, sino necesitarías infinitos recursos para resolver esto. Además operamos con String y tenemos operaciones como CompareTo y equals que nos facilitan esta tarea.

1.1.4. ¿Afectaría el orden a la eficiencia de alguna operación prescrita por StockIF? Razona tu respuesta.

Como ya he adelantado anteriormente, afectaría en la eficiencia a la operación de listStock. Lo vemos sencillamente con ejemplo:

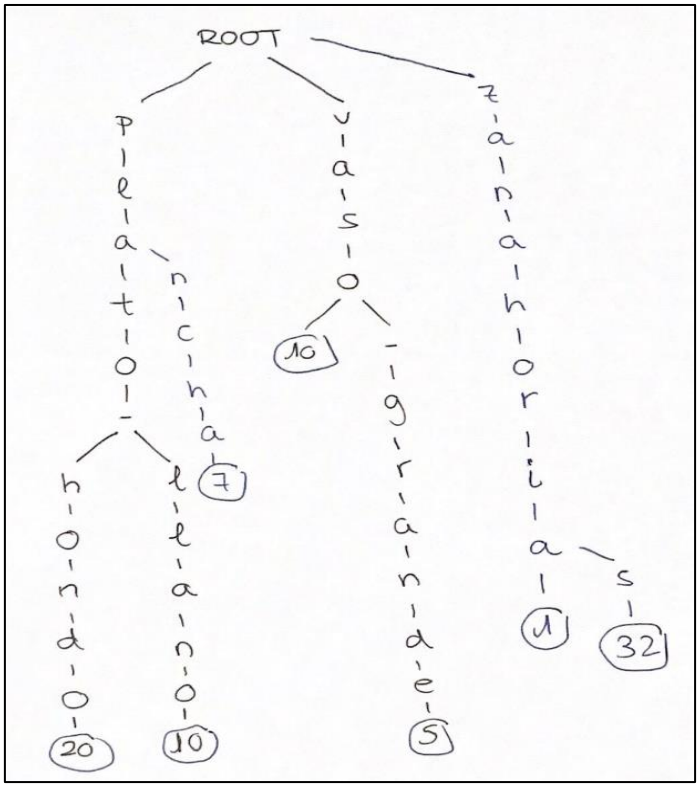
-Nosotros declaramos que prefix = “man”.
-Nuestra lista es: arándanos, bananas, cebollas, comino, manzanas, mangos, pimientos…
-La lista, buscará por la primera letra que coincida con “m”, en una lista auxiliar de forma iterativa se irán añadiendo los elementos que contengan esta letra. Se dejará de buscar en la primera lista cuando se llegué a la última palabra con “m” para no tener que seguir buscando cuando es innecesario y la lista puede ser inmensa. Una vez se completa la primera letra se repetirá el proceso con la letra a y luego con la n. Una vez prefix se haya leído imprimará los elementos que se han ido insertando en la cola.

Si la lista no está ordenada, habría que ordenarla en cada proceso de búsqueda, pero además, alfabéticamente, nos da eficiencia para que, una vez no haya más palabras con esa letra, deje de buscar en posiciones inferiores. Por último si prefix es null, una forma adecuada de que se enliste el Stock es por orden alfabético.

1.2. StockTree

1.2.1. Utilizando papel y lápiz inserta más pares producto-unidades en el árbol del ejemplo. Compara al menos dos formas de colocar los hijos de un nodo y comenta qué ventajas tendría cada una de ellas. Razona tu respuesta.

Insertando pares producto-unidades observamos varias cosas. Los productos se dividen en letras y cada letra corresponde a un Nodo de tipo Inner. Las unidades de estos productos se insertan en Nodos de tipo Info. Como yo lo planteo es que vamos a recorrer el árbol de izquierda a derecha comparando las letras. Podemos insertar producto-unidades de dos formas. Perteneciendo a un Node Inner existente porque ya existe parte de la cadena o podemos insertar directamente en la raíz. Lo que está claro es que una vez coincida la cadena ya no tendremos que volver a la derecha, sino, seguir el recorrido por abajo.



1.2.2. La operación listStock(String prefix) nos indica que la secuencia de elementos de tipo StockPair que devuelve ha de estar ordenada de forma creciente según los productos. ¿Alguna de las opciones de colocar los hijos de un nodo de la pregunta anterior resulta más conveniente para mejorar la eficiencia de esta operación? Razona tu respuesta.

Claro, yo lo he dado por sentado pero creo que como en el caso de Sequence, el árbol tiene que estar ordenado alfabéticamente. ¿Por qué?
Lo que interesa es que el árbol, haga una lista de todos los hijos de un nodo de forma horizontal, y comparando esos nodos ordenados alfabéticamente seleccionara cual es el ideal para seguir haciendo su recorrido hacia abajo. Una vez baje otra vez hará ese recorrido horizontal con los hijos, y así, sucesivamente. Nos podemos encontrar dos casos, que las letras coincidan y siga bajando por el árbol hasta llegar al recorrido, pero si no coincide puede parar de recorrer el árbol o crear un Nodo nuevo de tipo Inner, respetando el orden alfabético. Creo que esa es la forma más eficiente.

2. Preguntas teóricas sobre costes

2.1. Defina el tamaño del problema y calcule el coste asintótico temporal en el caso peor de las operaciones updateStock y retrieveStock para ambas implementaciones.

Primeramente tenemos StockSequence.

En retrievestock tenemos un while por lo tanto Orden O(n) y luego tenemos sentencias condicionales pero no aumentan el orden de complejidad porque son todas las operaciones constantes O(1).
Lo mismo pasa en updatestock, tenemos un while que hace que el peor caso sea orden O(n) y dentro del while hay sentencias condicionales pero sus operaciones son todas de orden constante O(1).

“n” en este caso es el número de elementos que recorrer, por lo tanto, el peor caso, sería recorrer absolutamente todos los elementos hasta llegar al final, que eso pasaría si tuviésemos que manipular el elemento que alfabéticamente es el mayor de todos.

Ahora tenemos StockTree.

En la implementación de StockTree, el costo asintótico temporal en el peor caso de retrieveStock y updateStock es O(n), donde n es la longitud de la cadena p. Esto se debe a que se realiza una búsqueda a lo largo del árbol hasta encontrar el nodo correspondiente, lo que requiere una cantidad de operaciones proporcional a la longitud de la cadena p. Lo que pasa es que el árbol tenemos más elementos porque lo que en lista un nodo es una palabra, en árbol, todas las letras de esa palabra, son cada una de ellas, un nodo diferente. Por lo tanto vamos a tener más.

2.2. Compare el coste asintótico temporal obtenido en la pregunta anterior con los costes empíricos obtenidos. ¿Coincide el coste calculado con el coste real?

No he tenido tiempo para realizar el coste asintótico temporal real por lo tanto no lo puedo comparar con el teórico. Pero mediante los juegos de pruebas proporcionados por el equipo docente se observan varias características:

- La secuencia es menos costosa que el árbol.

- Ha medida que insertamos más elementos evidentemente crece más el coste, pero crece más en el árbol por lo que he explicado, porque lo que es un nodo en la lista, son muchos más en el árbol por la división en letras.

- Cuando se añaden pocos elementos ambos costes son similares porque la diferencia de cantidad de elementos en ambas no es muy grande.