**MEMORIA ESTRATEGIAS DE PROGRAMACIÓN Y ESTRUCTURAS DE DATOS 2021/2022**

**Preguntas teóricas de la sección 2.1**

1. ¿Qué debería devolver la operación size() de CollectionIF para un array disperso?

Tal y como está implementado la operación size() en la clase Collection, comenzamos respondiendo a esta cuestión indicando que debería ser en todo caso un número entero.

Ahora podemos preguntarnos, qué debería devolver en cada implementación. En el caso de SparseArraySequence - ¿Debemos devolver el número de posiciones que tenemos ocupadas en la lista? ¿O debemos devolver el mayor Index de todos los IndexedPair?

A mi parecer, size() debe funcionar como un contador de los pares indexados que estamos guardando en la lista, y por lo tanto, debe devolver el número de pares que hemos guardado en nuestra estructura de datos y no el mayor de los índices del tipo IndexedPair que guardamos en la lista. También podemos aquí devolver el tamaño de la lista.

En el caso de la implementación del árbol, size() debe funcionar como un contador de los nodos que están ocupados, y no de los nodos de soporte sobre los nodos hijos.

Si devolviéramos el número de nodos, estuvieran ocupado o no, no estamos devolviendo verdaderamente el tamaño de la estructura.

En ambas implementaciones, si size() devolviera otro valor que no fuera el número de pares indexados que tiene nuestra estructura de datos, estaríamos desvirtualizando el concepto de array disperso.

1. ¿Qué operaciones de SparseArrayIF deben tenerse en cuenta para que el código de size() ya implementado en Collection sirva para un array disperso? Razona tus respuestas.

Aquí debemos tener en cuenta que en la implementación de varios métodos, tenemos que actualizar size().

En el método set (int pos, E elem), debemos tener en cuenta dos aspectos de cara a la variable size() de collection:

* Si el elemento ya existe, no se produce su actualización ya que el tamaño de la estructura no cambia al elemento ser “machacado” por el nuevo valor
* Si el elemento no existe, entonces, debemos actualizar size() e incrementarla, size()

En el método delete(int pos), también debemos tener en cuenta si el elemento que queremos eliminar existe o no. Si existe, entonces tenemos que decrementar el tamaño de size(). En caso contrario, como devolvemos un null, no tenemos que actualizar esta variable.

1. ¿A qué clase debería extender una implementación de SparseArrayIF: Collection o Sequence? Razona tu respuesta.

Si extendemos la implementación desde Sequence, nos obliga a ofrecer una implementación a los métodos públicos de la clase padre. Esta opción no nos interesa ya que no queremos usar todos los métodos de la clase, sólo estamos interesados en un subconjunto de ellos.

En este caso, es mejor utilizar la funcionalidad de una secuencia, creando un objeto de tipo secuencia dentro de mi clase SpasrseArraySequence. De esta forma, puedo ofrecer al exterior los métodos oportunos (sin ofrecer todos necesariamente). Además, puedo valerme de la funcionalidad de la secuencia sin tener que preocuparme de cómo funciona esta clase internamente – algo que es muy difícil y tedioso si heredamos directamente de Sequence.

Por lo tanto, internamente, usamos una secuencia, aunque mi clase no es una secuencia, simplemente nos aprovechamos de tener uno internamente para delegar en ese objeto interno las operaciones.

Además, ya existen implementaciones muy interesantes dentro de SequenceIF que nos permite guardar objetos de diferentes maneras (list, Stack y Queue)

Preguntas teóricas de la sección 2.2

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

La secuencia más óptima para esta implementación sería una lista, ya que esta nos facilita el guardado de los elementos de pares indexados. Este tipo de secuencia nos permite agregar en una posición concreta (facilitando operaciones posteriores), eliminar datos, etc.

Además, su recorrido no es destructivo. Podemos consultar un elemento y que este no sea eliminado de su estructura.

Si usamos una pila, contamos con la desventaja de que sólo podemos trabajar con el elemento que está en la cima de la pila. Además, este recorrido es destructivo por lo que después de consultar el elemento de la pila que nos interesa, hemos perdido las posiciones que estaban encima de él.

En el caso de elegir una cola como tipo de secuencia, nos encontramos con una dificultad similar a la pila, y es que sólo podemos trabajar con el elemento que está al final de este tipo de secuencia y además nos volvemos a encontrar con un recorrido destructivo, que no nos interesa en este caso.

2. ¿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 de almacenamiento de los pares en la secuencia es el orden ascendente del índice. Esto nos permite hacer más eficientes otras operaciones, como el delete – si buscamos un número concreto y lo hemos encontramos, no necesitamos seguir recorriendo los elementos, de igual forma que el get.

Si lo almacenamos sin ningún tipo de orden, para poder luego ordenarnos, necesitamos hacer una función que ordene los elementos, aumentando los costes y la complejidad del programa.

Al final, es mejor hacer que en la inserción ya se haga directamente con los índices ordenados, ya que aunque esta operación sería más compleja que si la hacemos sin orden, ordenarlos nos facilita otros métodos más adelante.

1. ¿Afectaría el orden a la eficiencia de alguna operación de Collection? Razona tu respuesta.

No, ya que ninguna operación de Collection lleva implícita una búsqueda de elementos o cualquier operación donde el orden en el que se encuentra el elemento nos afecte.

Preguntas teóricas de la sección 2.3

1. Utilizando papel y lápiz inserta elementos en los índices 0,1,2,3… En vista del árbol obtenido, ¿cómo podrías recorrer el árbol para encontrar los elementos por orden creciente de índice? Razona tu respuesta.

Aquí podemos hacerlo de dos formas:

* Realizar cualquier tipo de recorrido y posteriormente, crearnos una función que nos ordene los elementos por orden creciente.
* Realizar el recorrido en anchura, para que directamente salgan ya los elementos ordenados por orden creciente.

En la implementación de la práctica hemos usado el recorrido en anchura.

1. Dado que podemos localizar el nodo que contiene el par indexado con el índice x mediante la búsqueda descrita anteriormente, ¿es necesario almacenar un par indexado en los nodos o bastaría con almacenar únicamente el elemento a indexar? Razona tu respuesta.

Sería suficiente con almacenar únicamente el elemento a indexar, que tal y como dice el enunciado, podemos localizarlo fácilmente porque sabemos el número de su índice.

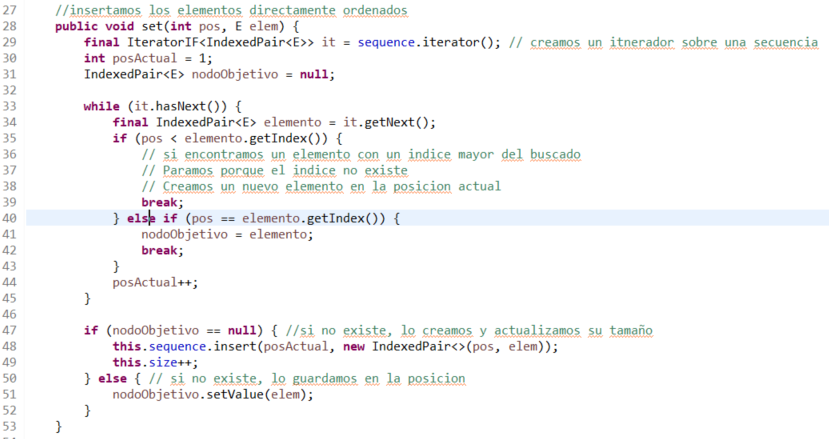
Sin embargo, para el método indexIterator, que espera que demos una lista de los índices que están asociados a un valor, y el problema es que para ello tenemos que recorrer el árbol, y esto no puede ser del todo fácil y asignar a cada nodo a un índice. Por lo tanto, indexedPair nos hace la vida más fácil

Preguntas teóricas de la sección 6 1.

1. Defina el tamaño del problema y calcule el coste asintótico temporal en el caso peor de las operaciones set y get para ambas implementaciones.

En el caso de la secuencia, el caso peor es cuando tenemos que ir al final de la secuencia para insertar/consultar. Probaremos mirando un índice igual o mayor que los ya existentes para medir siempre el caso peor.

En el caso de SparseArraySequence el código de set es el siguiente:



Hasta la línea 32 todos son llamadas de coste constante (incializar variables primitivas o estructuras de datos vacías, solicitar iterador de la secuencia)

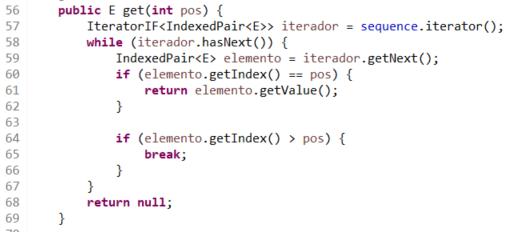
Desde la línea 33 a la línea 45 tenemos un bucle while, con una condición de salida que se comprueba en tiempo constante. Todo el cuerpo del bucle es de coste constante (comparar la variable pos con el índice del elemento y actualizar el valor de la variable nodoObjetivo), por lo tanto, el peor caso es cuando tenemos que recorrer el bucle hasta el final porque no llegamos a salir con el break. El coste asintótico temporal del bucle es de O(n), donde n es el número de elementos que tenemos en la lista.

Como el bucle se ha ejecutado hasta el final, y no ha encontrado la posición que hemos pasado por parámetro entre los pares indexados en nuestra lista, nos vamos a la opción que nodoObjetivo no exista, hay que insertar el par indexado en la posición posActual, por lo que es una operación de coste O(posActual).

Finalmente, las demás operaciones son de coste constante.

El coste asintótico temporal por lo tanto es T(n), siendo n el número de pares indexados que existen en la estructura, es decir, es lineal con respecto al número de pares indexados que existe en la estructura.

En el caso de SparseArraySequence el código de get es el siguiente:



La línea 47 tiene un coste constante, ya que simplemente pedimos el iterador de la secuencia.

De la línea 58 a la línea 67 tenemos un bucle while, con coste 0(n), siendo n el número de pares indexados existentes en nuestra estructura, por lo tanto, este coste es lineal respecto al número de pares indexados de nuestro array disperso.

Finalmente, la línea 68 es de coste constante.

Aquí el caso peor es cuando tenemos que recorrer la lista entera porque nuestro elemento no existe y llegar a la línea 68 para que nos devuelva un null.

Por lo tanto, es de coste O(n), siendo n el número de inexedpair de nuestra estructura.

* En el caso del árbol binario, el coste de get/set es independiente del número de nodos del árbol, solo depende el número de bist que se necesiten para representar el índice ficticio en cuestión. En concreto, si un número requiere 20 bits para representarse, el pero caso es que el elemento exista y haya que bajar los 20 niveles del árbol para encontrarlo, o que el nodo exista y esté vacío. Es decir, cuando no podemos abandonar la búsqueda de forma prematura aún sabiendo que no está.

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