DESCRIPCIÓN DE LAS ESTRUCTURAS DE DATOS Y ALGORITMOS

Estructuras de datos

References

Para representar las referencias hemos utilizado dos arraylists, uno llamado *keys* que representa las celdas que son referenciadas, y otro arraylist llamado *values* que cada elemento de este arraylist contiene otro arraylist con las celdas que referencian a una celda de *keys*, y la conexión entre *keys* y *values* es el índice, de manera que la celda en la posición 0 de *keys* es la celda referenciada en las celdas del arraylist de la posición 0 de *values*.

Es una estructura idéntica a la de un Hashmap, pero se pueden modificar las claves en caso de que se añada o se borre una fila o columna.

Document

Para identificar cada documento hemos utilizado un string con su nombre, y para representarlo usamos un Hashmap que contiene todas las sheets pertenecientes a este documento, donde cada *key* de este map es el identificador de la sheet, y cada *value* es el objeto sheet.

Sheet

Hemos utilizado un int para identificar a la hoja y un string que es el nombre que le da el usuario a la hoja.

Además la hoja está formada por un arraylist de arraylist de celdas debido a la propiedad redimensionable de esta que nos viene perfecto para cuando añadimos o eliminamos una fila/columna. La hoja tiene un tamaño mínimo de 1x1 (una celda) y un tamaño máximo de 255x255 por practicalidad (hemos considerado que hacerla infinita sería inviable).

Las funciones *addRow* y *Addcolumn* tienen coste constante respecto al tamaño del array que va desde donde añadimos al final (n), entonces O(n) para el update de las variables fila y columna de cada celda y luego el add de la row o de la columna que es de tiempo constante amortizado, esto significa O(m) siendo m el número de elementos que añadimos. Finalmente al actualizar las referencias recorremos toda la matriz, coste O(n^2*ref) siendo n el tamaño de rows y columnas (mismo tamaño para el peor caso) y ref son las referencias que tienen cada celda de la matriz.

Las funciones *deleteColumn* y *deleteRow* también tienen el mismo coste, está vez es recorrer la matriz entera porque también actualizamos las referencias, O(n^2*ref).

Ambas funciones tienen un coste adicional debido a tener que actualizar la variable row y column de cada celda pero es una variable necesaria para recorrer apropiadamente la matriz en las demás funciones y para actualizar correctamente las referencias.

DeleteContBloq, copyContBloq y moveContBloq recorren las celdas de las que queremos eliminar, copiar o mover y para cada una eliminar/añadir/mover todas las referencias que hacía, coste O(n*ref) siendo ref las referencias de cada celda.

Las demás funciones tienen un tiempo constante o lineal respecto al tamaño de las celdas que manejamos en dichas funciones.

Cell

Para representar una celda hemos utilizado dos enteros *row* y *column* que representan la fila y la columna en la que se encuentra la cell dentro de su sheet. También tiene como parámetro un string llamado content, el cual tendrá el contenido de la celda.

Function

La clase function es una clase abstracta que contiene como parámetro un ArrayList<Parameters> al cual podrán acceder todas las subclases, y una función llamada getValue() que también heredan todas ellas.

El primer parametro del Arraylist es el nombre de la función del usuario, los demás parámetros son los que necesita dicha función.
Por ejemplo si queremos aplicar la función de Reemplazar texto, la función necesitará como parámetros el texto original, el substring a reemplazar, y el substring por el cual queremos reemplazarlo. Por lo tanto el arraylist tendrá tres elementos, en la posición 0 estará “replaceText”, en la 1 el texto original, en la 2 el texto que substring que queremos reemplazar y en 3 el substring por el cual tenemos que reemplazarlo.
De esta manera todas las subclases tendrán su propio arraylist con los parámetros necesarios para aplicar la función.

La función getValue() es definida en esta clase, pero implementada en cada una de las subclases de Function. De hecho, la implementación en cada subclase es concreta e independiente en cada una de ellas, lo que quiere decir que getValue retorna el resultado de aplicar la función correspondiente para los parámetros indicados. Por ejemplo, si queremos hacer la media de un conjunto de números, llamaremos a la función getValue de Function, y esta se ejecutará en la subclase correspondiente (en este caso la clase Mean), y devolverá el resultado de aplicar la media a los valores indicados. Este último paso es automático, y sirve para cualquier función de la clase Function.

Cabe remarcar que hemos definido esta clase como abstracta ya que estas permiten crear un objeto y dejan que las subclases decidan qué clase instanciar. En otras palabras, las subclases son responsables de crear la instancia de la clase. Y como hemos visto en el párrafo anterior, esto nos servirá para que cada tipo de función tenga su propio código, y que el usuario simplemente decida que tipo de función quiere aplicar y esta realizará el cálculo correspondiente.

GetFunctionFactory

La *Factory* es un patrón de diseño creacional para crear diferentes objetos del mismo tipo, en vez de hacer una llamada directa a la constructora de cada objeto, y con un método de registro para añadir nuevos objetos sin romper el principio de Open/Close (abierto para extensiones pero cerrado para modificaciones). Para guardar todas las clases hemos utilizado un Hashmap llamado *instances* donde cada clave es el nombre de cada función, y el valor es la clase de la función. En consecuencia habrá un elemento en el map por cada función implementada en el programa.

Uso de Strings para el contenido de celda:

Para el contenido de las celdas hemos decidido que todo contenido será de tipo String, ya que permite representar cualquier tipo de carácter, y a diferencia de Char, puede representar y almacenar más de un carácter. Todos los parámetros de entrada son Strings, pero cuando hacemos los cálculos necesarios en las funciones, convertimos los parámetros (que vienen dados por un array de String), y calculamos el resultado que será retornado también en formato String. Este último paso solo se aplica a las funciones que no son de tipo texto, ya que para estas no hará falta convertir el String pasado por parámetro.

Algoritmos

Referenciación absoluta

Hemos decidido usar el método de referenciación absoluta para cuando agregamos y eliminamos filas o columnas. La referencia absoluta es cuando una referencia permanece constante sin importar a dónde se copie la fórmula que la contiene, o cuando se agregan filas o columnas que puedan modificar el identificador de la celda referenciada.

Por ejemplo, si en una función se referencia la celda B2, y añadimos una columna entre la A y la B, la celda referenciada pasaría a ser C2, y en nuestro caso, la función que apunta a la celda (que ahora es C2) dejaría de apuntarla y pasaría a apuntar a la nueva celda B2, que se ha añadido al añadir toda una columna.

Funciones de búsqueda, reemplazo y ordenación

Para la búsqueda en un bloque usamos la función de la clase string *contains* que usa el algoritmo Boyer Moore que tiene de coste O(nm) siendo n el tamaño del contenido de la celda dónde buscamos y m el tamaño del substring que estamos buscando, pero este coste solo ocurre cuando todos los carácteres son iguales, normalmente el coste es más cercano a O(n). Esto hay que hacerlo i veces, siendo i el número de celdas donde buscamos. Así que el coste total es O(i*nm) en el peor caso, O(i*n) es lo más común.

El coste de *replaceContBloq* llama a *searchContBloq* ya que también tiene que buscar substrings solo que este también los reemplaza que es tiempo constante. Así que el tiempo peor también es O(i*nm) siendo i el número de celdas donde queremos reemplazar. Además este tiene que actualizar las referencias de las celdas reemplazadas así que el coste sería O(i*m + ref *i) siendo ref las referencias de cada celda donde reemplazar.

Por último el coste de *sortContBloq* que utiliza la función sort() de arraylist que utiliza el mergesort con coste O(nlog(n)) donde n es el número de celdas que queremos ordenar. Además hacemos n operaciones *set* para cambiar el contenido de las celdas siendo n también el número de celdas cuyo coste es O(n). También tenemos que actualizar las referencias cuyo coste es el número de celdas donde hemos sorteado por sus referencias O(n*log(n) + n*ref).

Funciones implementadas

Funciones	**Coste**	**Justificación**
Truncate	O(1)	Constante ya que todas las operaciones usadas son constantes (incluido BigDecimal)
Absolut	O(1)	Constante ya que la multiplicación es constante
Floor	O(1)	Constante ya que *Math.floor()* es constante
Increment	O(1)	Constante ya que la suma es una operación constante
Mean	O(n), tal que n es el número de elementos del vector	Cálculo de la media de los valores del vector = n (para n elementos)
Identity	O(1)	Constante ya que devuelve la entrada
Median	O(n log(n)), tal que n es el número de elementos del vector	Coste de ordenar el vector con el algoritmo Collections.sort()
Variance	O(n), tal que n es el número de elementos del vector	Cálculo de la media de los valores del vector = n (para n elementos) + cálculo de la varianza una vez obtenida la media = n (para n elementos)

Covariance	O(n), tal que n es el número de elementos de un vector	Cálculo de la media de los dos vectores (que tienen el mismo tamaño) = 2n + cálculo de la resta de cada elemento y la media = n
StandardDeviation	O(n), tal que n es el número de elementos del vector	Cálculo de la media de los valores del vector = n (para n elementos) + cálculo de la covarianza una vez obtenida la media = n (para n elementos)
PearsonCorrelation	O(n), tal que n es el número de elementos del vector X y del vector Y	Cálculo de las medias de los vectores X e Y dentro del bucle que recorre todos los elementos del vector X
ReplaceText	O(1)	Constante ya que *text.ReplaceAll()* es constante
LengthText	O(1)	Constante ya que *String.length()* es constante
ElementExtraction	O(n + m), donde n es el tamaño de la fecha y m el tamaño del índice	Cálculo de que índice recorriendo cada carácter de la fecha = n + *date.substring()* = m
DayOfWeek	O(1)	Constante ya que funciones usadas son constantes