EXPLICACIÓ DE LES ESTRUCTURES DE DADES I ALGORISMES

1. Perquè una LinkedList?

La característica principal del nostre programa és l'important ús que li donem a una estructura de dades que ens proporciona Java: les LinkedList.

La LinkedList ens permet fer interseccions o eliminacions en temps constant utilitzant iteradors, encara que l'accés als elements és seqüencial.

El principal benefici del qual nosaltres ens aprofitem és la reutilització d'iteradors existents per insertar, eliminar o modificar cel·les (amb contingut o sense).

Un altre benefici, encara que menys habitual, que trobem amb les LinkedList és que quan vulguem afegir o eliminar la primera posició de la llista (o de les primeres), ja que aquesta operació amb LinkedList triga un temps O(1). Amb una altra estructura de dades com podria ser ArrayList, el fet d'haver de moure totes les posicions un lloc faria que el temps fos O(n), provocant una notable diferencia de temps comparat amb la estructura escollida per nosaltres.

Un "inconvenient" que trobem de les LinkedList en el nostre programa seria que si hem d'emmagatzemar una fulla molt gran (amb continguts), els punters a l'element següent i l'anterior s'han de guardar, provocant una major sobrecarga, però si aquesta fulla que ha de ser molt gran, encara no hi té continguts, no ha de reservar un espai de memoria que si hauria de reservar si s'utilitza una altra estructura com les ArrayList, les quals reserven tant espai en memoria com necessiten abans de res, per tant, això pot ser un inconvenient per nosaltres però creiem que és millor utilitzar les LinkedList que no una altra estructura en la qual ja haguem d'estar reservant espai en memoria abans d'haver fet res.

2. Capa de presentació

A la capa de dades trobarem totes les vistes necessàries perquè l'usuari pugui obtenir una experiència agradable quan utilitzi el programa. Podem trobar diferents vistes com serien les següents:

- **DocumentView:** Vista principal del programa, des d'on podrem instanciar i obrir nous documents.
- **SheetView:** Vista de la fulla de càlcul, des d'on es podrà treballar amb aquesta i fer totes les operacions que calguin.
- **ErrorView:** Frame que mostra un missatge d'error en pantalla.
- **SheetListView:** Component que s'afegeix a la SheetView que mostra en una llista el nom les fulles existents del document.
- **SheetMenuBarView:** Component que s'afegeix a la SheetView que mostra una barra d'eines amb la que interactuar.
- **TableListener:** Expansió de la classe de swing TableModelListener. Encarregat mitjançant esdeveniments de llegir les modificacions de la taula, llegir les fórmules i comunicar amb el controlador les operacions a fer.
- **TableModel:** Expansió de la classe de swing AbstractTableModel. Guarda les dades de la taula per poder visualitzar-les.
- **SheetPane:** Expansió de la classe de swing JPanel. Encarregat de definir com es presenta la taula.

3. Estructures a les classes principals de la capa de domini

3.1. Document

Aquesta classe és la més general, aquí tindrem tots els documents que ha creat l'usuari i s'identificaran per nom.

L'estructura de dades que hem escollit és la LinkedList, hem utilitzat aquesta estructura per la seva facilitat en termes de cost a l'hora d'esborrar o afegir valors, també l'hem escollit per la flexibilitat que ens dóna en crear-les en termes d'espai d'emmagatzematge, ja que si o haguéssim pensat amb un vector hauríem de donar un espai o crear-lo d'una determinada mida i potser mai ompliríem tot aquest espai que havien demanat al principi. Per aquest motiu la LinkedList vam veure que era millor opció per implementar en el nostre treball.

3.2. Sheet

Aquesta classe fa referència al lloc on treballarem amb les cel·les, el que seria la nostra taula de l'Excel. Les fulles s'identifica pel seu nom i trobem que té una Linkedlist de files que és on es trobaran totes les cel·les que tinguem en una fulla. D'altra banda, també trobem dos rang de cel·les provisionals per si s'han de fer operacions amb rang, com serien les operacions estadístiques. Un rang seria per realitzar les operacions estadístiques que només necessiten treballar amb un sol rang com serien la mitjana, la mediana, la variància i la desviació estàndard i després les que treballen amb dos rangs com serien la covariància i la correlació de Pearson.

3.3. Row

Pel que fa a la classe Row, s'utilitza per emmagatzemar les cel·les amb les quals treballarem i farem operacions. Les cel·les amb les que treballem s'emmagatzemen en una LinkedList de cel·les.

3.4. Cell

La classe Cell fa referència al lloc on emmagatzemem tots els valors amb els quals anem treballant. En aquesta classe hem decidit afegit una LinkedList en la qual guardarem quines són les cel·les referenciades que té una cel·la en concret, per després poder saber amb quines cel·les interactuava una cel·la en el cas que s'hagi esborrat una fila o una columna s'hagi modificat el contingut.

Com que hi ha diferents tipus de valors que podem emmagatzemar a les cel·les, hem creat diferents classes que s'estenen de la classe cel·la (serien els fills).

- **DataCell:** Cel·la en la qual el seu contingut és una data.
- **NumCell:** Cel·la en la qual el seu contingut és un double.
- **StringCell:** Cel·la en la qual el seu contingut és un String.
- **FormulaCell:** Cel·la en la qual el seu contingut és una fórmula i conté una LinkedList amb totes les cel·les que són referenciades en aquella fórmula.
    - **FormulaCellDate:** Cel·la en la qual el seu contingut és el resultat d'una fórmula de tipus data.
    - **FormulaCellNum:** Cel·la en la qual el seu contingut és el resultat d'una fórmula de tipus double.
    - **FormulaCellString:** Cel·la en la qual el seu contingut és el resultat d'una fórmula de tipus string.

3.5. Rank

La classe Rank, fa referència a un conjunt de cel·les, conté una LinkedList de cel·les per saber quines cel·les formen part d'aquest rang i un atribut per saber el nombre de columnes i files del rang que té el rang.

4. Capa de dades

A la capa de dades trobem de primera un controlador de persistència que està connectat al controlador de domini, i l'únic que fa aquest controlador de persistència és connectar la capa de domini amb la capa de dades.

Dintre la capa de dades trobem tres classes més que serien les classes CSV, ControladorLoad i el ControladorWrite.

- **CSV:** És la classe més general de la capa de dades, i en aquesta classe trobarem tots aquells mètodes que ens serviran per trobar els diferents fitxers on s'han d'emmagatzemar les dades o per trobar els fitxer d'on hem d'agafar les dades per carregar-les al programa.
- **ControladorLoad:** Aquesta classe contindrà la funció amb la qual llegirem el contingut dels fitxers, per posteriorment carregar tota aquesta informació llegida a la capa de presentació.
- **ControladorWrite:** Aquesta classe contindrà la funció amb la qual escriurem les dades ens els diferents fitxers csv que tingui el programa.

5. Algorismes a comentar:

5.1. Operacions

Nom de l'operació	**Descripció**	**Cost**	**Justificació**
Average	Calcula la mitjana	O(n)	Recorrem una LinkedList linealment per obtenir els valors
Median	Calcula la mediana	O(n)	Recorrem una LinkedList linealment per obtenir els valors

Variance	Calcula la variància	O(n)	Recorrem una LinkedList linealment per obtenir els valors
Covariance	Calcula la covariància	O(n)	Recorrem dues LinkedList linealment per obtenir els valors
Standard Deviation	Calcula la desviació estàndard	O(n)	Recorrem una LinkedList linealment per obtenir els valors
Pearson	Calcula la correlació de Pearson	O(n)	Recorrem dues LinkedList linealment per obtenir els valors
Day	Indica quin dia es donada una data	O(1)	Hem utilitzat una llibreria general de Java. No sabem la seva implementació però intuïm que fa un recorregut seqüencial, cosa que ens fa pensar que el cost sigui constant.
Month	Indica quin mes es donada una data	O(1)	Hem utilitzat una llibreria general de Java. No sabem la seva implementació però intuïm que fa un recorregut seqüencial, cosa que ens fa pensar que el cost sigui constant.

Year	Indica quin any es donada una data	O(1)	Hem utilitzat una llibreria general de Java. No sabem la seva implementació però intuïm que fa un recorregut seqüencial, cosa que ens fa pensar que el cost sigui constant.
DayOfWeek	Indica quin dia de la setmana es donada una data	O(1)	Hem utilitzat una llibreria general de Java. No sabem la seva implementació però intuïm que fa un recorregut seqüencial, cosa que ens fa pensar que el cost sigui constant.
Length	Calcula el tamany d'un String en una cel·la	O(n)	Recorre caràcter a caràcter un String de mida n
Search	Busca si existeix un String en una cel·la	O(n)	Recorre caràcter a caràcter un String per trobar si existeix l'String que estem buscant
Replace	Reemplaça un String per un altre en una cel·la	O(n)	Fa servir la funció Search descrita anteriorment que té cost O(n)
Sum	Suma el contingut de dues cel·les	O(1)	Ja es disposa dels valors per realitzar l'operació i

			per tant només s'haurà de realitzar l'operació de sumar
Rest	Resta el contingut de dues cel·les	O(1)	Ja es disposa dels valors per realitzar l'operació i per tant només s'haurà de realitzar l'operació de restar
Multiplication	Multiplica el contingut de dues cel·les	O(1)	Ja es disposa dels valors per realitzar l'operació i per tant només s'haurà de realitzar l'operació de multiplicació
Division	Divideix el contingut de dues cel·les	O(1)	Ja es disposa dels valors per realitzar l'operació i per tant només s'haurà de realitzar l'operació de divisió
Binary	Converteix un enter a binari	O(n)	Hem utilitzat una llibreria general de Java. Hem vist la seva implementació i hem calculat el seu cost en funció del bucle, cosa que ens fa pensar que el cost sigui O(n).
Hexadecimal	Converteix un enter a hexadecimal	O(n)	Hem utilitzat una llibreria general de Java. Hem vist la seva implementació i

			hem calculat el seu cost en funció del bucle, cosa que ens fa pensar que el cost sigui O(n).
Truncate	Trunca el valor d'una cel·la	O(1)	Ja es disposa del valor a truncar i per tant només s'haurà de retornar el valor truncat
Absolute	Calcula el valor absolut del valor d'una cel·la	O(1)	Ja es disposa del valor per realitzar l'operació i per tant només s'haurà de comprovar si aquest és positiu o negatiu. Si és positiu no s'haurà de fer res i si es negatiu s'haurà de canviar a positiu
Increase	Incrementa en 1 el valor d'una cel·la	O(1)	Ja es disposa del valor de la cel·la i per tant només s'haurà d'incrementar en 1 aquest valor

5.2. Cerca d'un element

La cerca d'un element en un rang seleccionat té cost O(n), ja que fem la cerca lineal, com que utilitzem una LinkedList amb totes les cel·les del rang i a més el rang no té perquè estar ordenat i pot ser que contingui cel·les que no siguin del tipus demanat (String) i, per tant, anirem recorrent tota la LinkedList fins a trobar l'element desitjat.

5.3. Reemplaçament d'un element

El reemplaçament d'un element en un rang seleccionat també té cost O(n), ja que es basa en la cerca d'un element, explicat anteriorment, per després fer la modificació d'aquest contingut, el qual és constant.

5.4. Algoritme de Merge Sort

Aquest algoritme l'utilitzem per ordenar un rang de cel·les ja sigui de forma ascendent o descendent i per qualsevol dels tres tipus de valors que tractem (double, string i dates). El cost d'aquesta operació és de $O(n^2 \log(n))$.

Per la implementació d'aquest algorisme d'ordenació, la principal estructura que hem utilitzat ha estat un vector per emmagatzemar tots els valors del rang.

PSEUDOCODI:

```
mergeSort (Vector<Double> sortVector, int l, int r, String Criterium, char checkRank) {
    if (l < r) {
        int m = (l+r)/2;
        mergeSortNum(sortVector, l, m, Criterium, checkRank);
        mergeSortNum(sortVector, m+1, r, Criterium, checkRank);
        if (Criterium.equalsIgnoreCase("Ascendant") || Criterium.equalsIgnoreCase("A")) {
            mergeAscNum(sortVector, l, m, r);
        }
        else mergeDescNum(sortVector, l, m, r);
    }
}
```

Aquest primer tros de codi anem fent recursivitat i també indiquem de quina manera volem que s'ordeni el rang de cel·les, ja sigui de forma ascendent o descendent.

PSEUDOCODI:

```
mergeNum (Vector<Double> sortVector, int l, int m, int r) {
    int n = r - l + 1;
```

```
    Vector<Double> aux = new Vector<>(n);
    int i = l;
    int j = m+1;
    int k = 0;
    while (i <= m && j <= r) {
      if (sortVector.get(i) <= sortVector.get(j)) {       //Si volem que s'ordeni ascendentment
      if (sortVector.get(i) >= sortVector.get(j)) { //Si volem que sordeni descendentment
        aux.add(k, sortVector.get(i));
        i++;
      }
      else {
        aux.add(k, sortVector.get(j));
        j++;
      }
      k++;
    }
    while (i <= m) {
      aux.add(k, sortVector.get(i));
      k++;
      i++;
    }
    while (j <= r) {
      aux.add(k, sortVector.get(j));
      k++;
      j++;
    }
    for (k = 0; k < n; ++k) sortVector.set(k+l, aux.get(k));
  }
```

Finalment fem l'ordenació del Merge Sort amb l'ajuda d'un vector auxiliar.

5.5. Copia i enganxa

L'algoritme que fem servir per copiar i enganxar un rang en una altra posició també té un cost de O(n) tant per copiar els valors del rang com per enganxar-los en un altre lloc.

L'estructura de dades que utilitzem per fer aquestes dues operacions també és la LinkedList, ja que no ens fa falta saber la mida del rang per anar-la omplint amb els valors.

Pel que respecta a l'operació de copiar l'únic que s'ha de fer és anar llegint els diferents valors del rang i afegir-los a una LinkedList provisional, cosa que fa que el cost d'aquesta operació sigui O(n).

D'altra banda, l'operació d'enganxar és molt similar, s'anirà recorrent la LinkedList provisional que hem omplert amb els valors del rang que volem enganxar en un altre lloc i començarà a enganxar els valors a partir de la primera cel·la que indiquem, cosa que fa que el cost d'aquesta operació també sigui O(n).

Hem afegit una LinkedList temporal per no perdre dades mentre es fa l'operació, ja que les posicions de les cel·les es poden solapar y a l'hora d'enganxar depenent de com recorrem la LinkedList es poden perdre valors d'algunes cel·les.

4.6 Esborrar/Afegir Fila/Columna

- **Esborrar fila/columna:** L'esborrar una columna, s'han de mirar les cel·les referenciades. Si hi ha alguna cel·la fórmula referenciada, en el contingut d'aquesta es posaria: "error", s'esborrarien les cel·les de la columna en qüestió en les LinkedList necessàries i s'enllaçaria la columna posterior a la que hem esborrat amb la posterior a l'esborrada.

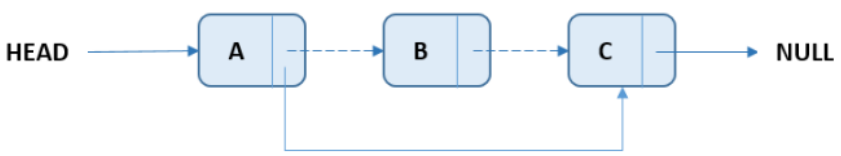

Figura 1: Esborrar un node en una linkedlist. Fotografia extreta de:
https://www.alphacodingskills.com/java/ds/java-delete-a-node-at-the-given-position-in-the-linked-list.php

- **Afegir fila/columna:** Quan s'afegeix una columna, s'enllaçarien els punters de la columna anterior a la nova amb la nova i els següents de la nova amb la columna posterior. Aquí deixem un exemple visual de com s'afegeix un node a una LinkedList. La columna funcionaria igual però afegint més nodes (cel·les).

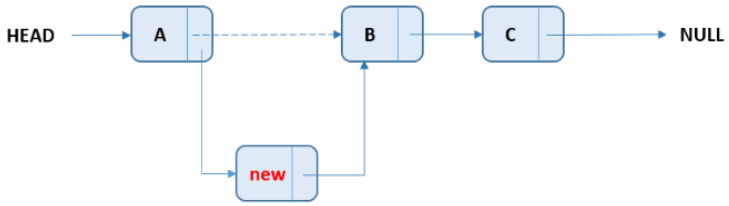

Figura 1: Afegir un node en una linkedlist. Fotografia extreta de:
https://www.alphacodingskills.com/java/ds/java-insert-a-new-node-at-a-given-position-in-the-linked-list.php