Irene Bertolín Rico
Carla Cánovas Iglesias
Adrian Cristian Crisan
Alex Moreno Baeza

ESTRUCTURES DE DADES

Document

1. ArrayList<Full>

Conté un conjunt de fulles, l'hem representat amb un ArrayList de fulles, donat que nosaltres guardem les fulles com fulla 1, fulla 2, fulla 3... De manera que ens interessa només guardar el número de fulla, i al eliminar-ne una, a totes les fulles més grans els hi restem 1 a l'índex. Per tant, l'única cosa que realment emmagatzemem és la referència a la fulla que pertoca a la posició segons l'ordre de creació.

De manera que:

Afegir full	$O(1)$	
Eliminar full	$O(n)$	$\Theta(1)$
Obtenir full	$O(1)$	
Obtenir la mida d'aquest	$O(1)$	

Podem observar que eliminar una fulla té cost màxim lineal, això és degut al fet que les cel·les poden estar referenciades en altres fulls, i per això hem de comprovar aquestes referències. Però al considerar que n en mitjana no és excessivament gran podem considerar que en mitjana és una operació que es resol en temps constant.

2. HashMap<String, Full>

Opcionalment, si decidim que les fulles tenen nom, llavors enlloc d'utilitzar un vector, utilitzaríem un Hashmap, on la clau és el nom de la fulla i el segon paràmetre és la referència a aquesta. Com podem deduir no poden haver-hi dos noms repetits. El costs de les operacions es mantenen.

Com en el nostre cas, hem decidit usar un número per full en lloc d'un nom, **utilitzarem un ArrayList** com a estructura de dades per emmagatzemar els fulls.

Full

1. Matriu nXm de cel·les

Cost de crear un full	$O(n * m)$
Cost d'afegir una fila	$O(n)$
Cost d'afegir una columna	$O(m)$
Cost d'eliminar una fila	$O(n)$
Cost d'eliminar una columna	$O(m)$
Cost de modificar una cel·la	$O(1)$

Tot i que si considerem que un full té en mitjana 100 files per 100 columnes, llavors el cost passa a ser:

Cost de crear un full	Θ(1)
Cost d'afegir una fila	Θ(1)
Cost d'afegir una columna	Θ(1)
Cost d'eliminar una fila	Θ(1)
Cost d'eliminar una columna	Θ(1)
Cost de modificar una cel·la	Θ(1)

2. Hashmap de cel·les

Cost de crear un full	$O(1)$
Cost d'afegir una fila	$O(1)$
Cost d'afegir una columna	$O(1)$
Cost d'eliminar una fila	$O(1)$
Cost d'eliminar una columna	$O(1)$
Cost de modificar una cel·la	Θ(1)

Hem decidit utilitzar un HashMap, ja que com podem veure en eficiència en mitjana és similar, però l'avantatge que ens ofereix l'opció del Hashmap enfront de la matriu és que no necessitem definir la mida del full, el nombre de cel·les i columnes, donat que podem anar-les afegint en només si utilitzem una cel·la i per tant el cost en espai és menor.

Cel·la

Per la cel·la necessitem dos estructures: una per guardar les referències que fa la propia cel·la a altres cel·les i una altra per saber quines cel·les tenen referenciada aquesta.

1. ArrayList

Les operacions tindran cost O(n), sent n el nombre de referències, però donat que el nombre de referències en mitjana no és molt gran podem dir que tindran cost Θ(1).

2. HashSet

Les operacions tindran cost Θ(1).

3. TreeSet

TreeSet on totes les operacions tenen cost Θ(log n), sent n el nombre de referències, però donat que el nombre de referències en mitjana no és molt gran podem dir que tindran cost Θ(1).

Utilitzarem un HashSet donat que és més eficient per fer operacions com find(), per trobar una referència i perquè internament un HashSet està implementat amb un HashMap, estructura de dades que utilitzem en una altra classe, i d'aquesta manera, mantenim certa relació.