Estructuras de Datos

UNIVERSIDAD DE BURGOS

Autores:

Jorge Navarro González

Mario Ubierna San Mamés

Índice de la complejidad de los métodos:

put:

```
21      public V put(R row, C column, V value) {
22          if (mapa.containsKey(row)) {
23              if (!mapa.get(row).containsKey(column)) {
24                  elementos++;
25              }
26          } else if (!mapa.containsKey(row)) {
27              mapa.put(row, new HashMap<C, V>());
28              elementos++;
29          }
30          return mapa.get(row).put(column, value);
31      }
```

- En este método tenemos que todos y cada una de las interacciones que se realiza en el mapa es O(log n) ya que en recorrer las filas tardamos O(log n) y en recorrer las columnas O(log m), pero al no estar concatenados, aplicamos la regla de la suma, esto significa que nos quedamos con la O() mayor, en este caso se corresponde con O(log n).

remove:

```
35      @Override
36      public V remove(R row, C column) {
37          elementos--;
38          return mapa.get(row).remove(column);
39      }
```

- En este método sucede algo parecido al anterior método, ya que en recorrer las filas tardamos O(log n) y en recorrer las columnas O(log m), por lo que en el peor de los casos tarda O(log n + log m) es igual O(log nm).

get:

```
43      @Override
44      public V get(Object row, Object column) {
45
46          if (mapa.get(row) == null) {
47              return null;
48          } else {
49              return mapa.get(row).get(column);
50          }
51      }
```

- En este método tenemos que acceder a las filas tardamos O(log n), acceder a las columnas O(log m), por lo que en el peor de los casos tendríamos O(log n + log m) es igual a O(log nm).

containsKeys:

```
53      @Override
54      public boolean containsKeys(Object row, Object column) {
55
56          if (mapa.containsKey(row) && mapa.get(row).containsKey(column)) {
57              return true;
58          } else {
59              return false;
60          }
61      }
```

- En este caso debemos tener en cuenta que no hay las mismas filas que columnas, por lo que en recorrer las filas tardamos O(log n), en recorrer las columnas tardamos O(log m), por lo tanto tenemos O(log n + log n + log m) es igual a O(log n^2 * m).

containsValue:

```
64      @Override
65      public boolean containsValue(V value) {
66
67          for (R a : mapa.keySet()) {
68              if (mapa.get(a).containsValue(value)) {
69                  return true;
70              }
71          }
72          return false;
73      }
```

- En este caso al tener un bucle for each, éste tiene una complejidad de n, la complejidad de get es O(log m), por lo que aplicamos la regla de la multiplicación y tenemos que la complejidad del método es O(n * log m).

row:

```
76      @Override
77      public Map<C, V> row(R rowKey) {
78
79          return mapa.get(rowKey);
80      }
```

- Como ya hemos mencionado antes, acceder a las filas tardamos O(log n), por lo que la complejidad es O(log n).

column:

```
82      @Override
83      public Map<R, V> column(C columnKey) {
84
85          Map<R, V> map = new HashMap<R, V>();
86          for (R a : mapa.keySet()) {
87              if (mapa.get(a).containsKey(columnKey)) {
88                  map.put(a, mapa.get(a).get(columnKey));
89              }
90          }
91          return map;
92      }
```

- En este caso el bucle externo es O(n), lo que hay dentro del bucle ya lo he mencionado en anteriores métodos y la complejidad seria O(log n + log m + 1 + log n + log m), por lo que la complejidad del método sería O(n* log n + log m + 1 + log n + log m) es igual a O(n* log n^2 * m^2).

cellSet:

```
129     @Override
130     public Collection<es.ubu.lsi.edat.pract08.Table.Cell<R, C, V>> cellSet() {
131
132         List<Cell<R, C, V>> lista = new ArrayList<>(this.size());
133         Iterator<R> it = mapa.keySet().iterator();
134         while (it.hasNext()) {
135             R next = it.next();
136             Iterator<C> it2 = mapa.get(next).keySet().iterator();
137             while (it2.hasNext()) {
138                 C next2 = it2.next();
139                 lista.add(new Celda<R, C, V>(next, next2, mapa.get(next).get(next2)));
140             }
141         }
142         return Collections.unmodifiableCollection(lista);
143     }
```

- Tenemos dos bucles while, los cuales tienen una complejidad de O(nm), además dentro del primer bucle tenemos O(n) por el keySet que hay dentro, además hay que añadir el keySet que tenemos antes del bucle por tanto con todo ello nos quedará una complejidad de O(n + nm^2), pero también tenemos un get que tiene una complejidad de log n y otro con complejidad log m porque uno recorre la fila y otro la columna por tanto la complejidad total será de O(n + nm^2 * lognm).

size:

```
145     @Override
146     public int size() {
147
148         return elementos;
149     }
```

- En este caso se realizan operaciones básicas, cuya complejidad es O(1).

isEmpty:

```
151     @Override
152     public boolean isEmpty() {
153
154         return elementos == 0;
155     }
```

- Al igual que en el método anterior, hacemos una operación básica en O(1).

clear:

```
157     @Override
158     public void clear() {
159
160         mapa = new HashMap<R, Map<C, V>>();
161         elementos = 0;
162     }
163
164 }
```

- En este caso también realizamos las operaciones básicas en O(1).