# Exact-Cover in Java

Here is the listing of a rather minimal implementation as a variant of Donald Knuths Algorithm X for the Exact-Cover problem in Java, explications below:

```
 1 import java.util.*;
 2 import java.util.function.*;
 3
 4 public class AlgX {
 5   Map<Integer,Set<Integer>> rs=new TreeMap<>(),cs=new TreeMap<>();
 6   Set<Integer> s=new TreeSet<>();
 7   static Consumer<Set<Integer>> c;
 8
 9   AlgX(int[][] a) {
10     for (int y=0;y<a.length;y++) for (int x=0;x<a[y].length;x++) if (a[y][x]!=0) {
11       cs.computeIfAbsent(x,i->new HashSet<>()).add(y);
12       rs.computeIfAbsent(y,i->new HashSet<>()).add(x);
13     }
14   }
15   AlgX(AlgX a,int y){
16     for (int i:a.rs.keySet()) rs.put(i,new HashSet<>(a.rs.get(i)));
17     for (int i:a.cs.keySet()) cs.put(i,new HashSet<>(a.cs.get(i)));
18     s.addAll(a.s); s.add(y); Set<Integer> r=new HashSet<>();
19     for (int c:rs.get(y)) {r.addAll(cs.get(c)); cs.remove(c);}
20     rs.keySet().removeAll(r); for (Set<Integer> c:cs.values()) c.removeAll(r);
21   }
22   void solve() {
23     if (cs.isEmpty()) c.accept(s); else
24     for (int y:Collections.min(cs.values(),new Comparator<Set<?>>() {
25       public int compare(Set<?> o1,Set<?> o2){return Integer.compare(o1.size(),o2.size());}
26     })) new AlgX(this,y).solve();
27   }
28 }
```

Explications:

In line 5 we instantiate the two maps *rs* and *cs* for the rows and columns of a matrix, where the columns represent the "universe" and the rows represent subsets of that universe. We are looking for a set of rows that covers the universe exactly once in each column.

The set *s* in line 6 collects the rows belonging to a solution, the Consumer *c* in line 7 gets called for each solution and has to be set to an appropriate instance before calling solve().

The constructor in lines 9-14 initializes the maps *rs* and *cs* from an int[][] containing the initial problem as matrix, as for example:

$$\begin{matrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{matrix}$$

The method *solve* in lines 22-28 prints the solution *s* if no more columns are found, otherwise calls recursively *solve* for a new AlgX constructed from *this* AlgX in lines 15-21 for each row *y* in the column with the minimal number of covered lines found by method *Collections.min* in line 24-26.

The constructor AlgX in lines 15-21 first makes a copy of *rs*, *cs* and *s* from the calling AlgX, adds the given row *y* to *s*, then for each column in row *y* collects all rows containing this column in a Set *r* and deletes this column from map *cs*.

Finally all rows in *r* are removed from map *rs* and in all remaining columns in *cs* the rows from *r* are removed.

This means, we reduce the original problem to a smaller problem with all rows and columns removed, that are already covered by the rows of the selected column y and if no more colums rest to cover, a solution is found.