Exercici del tauler d'escacs i les 8 reines

Informació sobre aquest típic problema de programació la podeu trobar per exemple a:

https://es.wikipedia.org/wiki/Problema_de_las_ocho_reinas

En aquest trencaclosques, es tracta d'ubicar **8 reines** o **dames** en un tauler d'escacs (de **8 x 8**) sense que es matin entre elles. Es a dir, no pot haver-hi dues reines que estiguin en la mateixa **fila**, **columna** o **diagonal**. Definirem un **taulell** per un **vector** de **8** elements, on a l'element **0** del vector posem la **fila** (de **1** a **8**) on es troba la reina **1**, a l'element següent (segon del vector), posem la **fila** (de **1** a **8**) on es troba la reina **2**, i així successivament, de forma que, cada element del vector reflecteix la fila on es troba la reina corresponent al tauler. La reina es trobarà en la columna que indiqui el seu número (o desplaçament al vector+1), i a la fila que indiqui el número del vector a la seva posició. Amb aquesta estructura de dades, ens n'assegurem que dues reines o dames no poden coincidir a la mateixa fila ni columna.

Convé saber que hi ha més d'una solució possible per resoldre el problema (en un tauler de **8 x 8**, hi ha **92** solucions diferents). Aquest és un típic problema de **backtracking** (metodologia que no veurem), però que donat que hi ha moltes solucions, es pot resoldre desordenant un vector (**int t = {1,2,3,4,5,6,7,8}** per exemple) amb un mètode **shuffle()**, que haurem de programar, i testejant el resultat fins que alguna de les llistes obtingudes aleatòriament compleixi amb els requeriments del problema. La funció de test l'anomenarem **correcte(t)**, li passarem un vector **t** contenint un **tauler** i retornarà **true** si aquest compleix les especificacions i **false** si no.

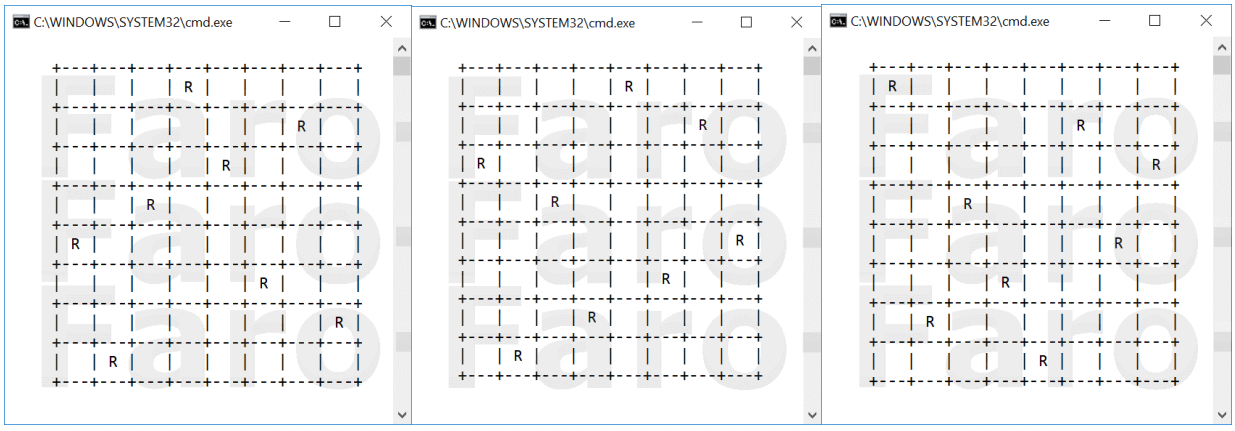
Es demana també que donada una solució representada per un vector **t** de valors que compleix amb els requisits del problema, programeu una funció **dibuixaTauler(t)** que dibuixa les **8** reines, representades per una **R** dintre d'un tauler dibuixat amb caràcters **ASCII**. Aquesta és probablement la part més complicada del problema, ja que la resta esdevé trivial, només hem de plantejar la funció de comprovació d'una determinada solució que mirarà si dos **reines** no es troben a la mateixa diagonal, ja que per la definició del tauler, hem exclòs que puguin estar a la mateixa fila o columna.

A la pàgina: http://elvex.ugr.es/decsai/algorithms/slides/5%20Backtracking.pdf hi ha un **PDF** que ens pot orientar sobre el problema de les **8** reines i com fer per determinar si dos reines es troben a la mateixa diagonal.

Un cop fetes aquestes dues funcions, la funció **main** quedaria

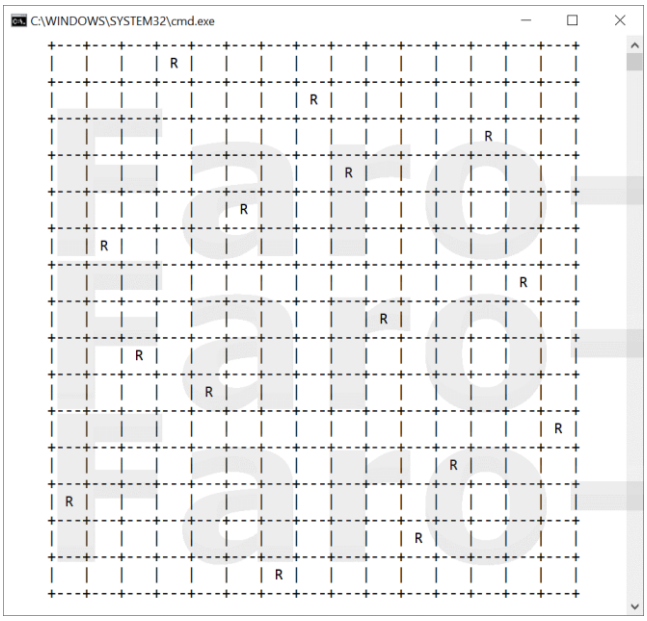
```
public static void main(String[] args) {
    int [] t = {1,2,3,4,5,6,7,8};   // Tauler
    while (!correcte(t))            // Mentre no sigui solució
        t = shuffle(t);             // Desordena el taulell
    dibuixaTauler(t);
}
```

S'indiquen alguns valors de tauler generats pel programa en successives execucions



Es demana també que exporteu el problema a qualsevol número de reines (a l'exemple de sota es mostra una possible solució per a un taulell de **15** reines). La idea seria fer que tant el dibuix del tauler, com l'algorisme de resolució sigui independent de la grandària del vector. A l'exemple, un cop modificar el programa, per dibuixar el taulell amb **15** reines, l'únic canvi al codi ha estat definir el vector inicial de **15** elements, i les funcions auxiliars no s'han modificat.

```
public static void main(String[] args) {
    int [] t = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};  // Tauler
    while (!correcte(t))     // Mentre no sigui solució
        t = shuffle(t);      // Desordena el taulell
    dibuixaTauler(t);
}
```



El problema de les **n** reines, no te solució per taulers més petits de **4 x 4** (amb **4** reines), i evidentment, a mesura que incrementem el nombre de files i columnes del taulell, el nombre de combinacions possibles s'incrementa considerablement, i la provabilitat de trobar aleatòriament una solució que compleixi amb els requeriments del problema es redueix (encara que també hi ha més solucions, el temps destinat a trobar una correcta pot ser considerable per a taulers grans). A la primera execució, per resoldre aquest tauler de **15x15** aleatòriament, s'han provat **122765** combinacions abans de trobar una correcta.

Sembla ser que hi ha un premi d'un milió de dolars pel que pugui resoldre un problema semblant amb **1000 reines** (en un taulell de **1000 x 1000** evidentment) amb un algorisme **polinomial**, i on algunes de les reines ja estarien col·locades. Més informació a:

https://www.meneame.net/m/cultura/problema-1000-reinas-millon-dolares-juego
http://ecodiario.eleconomista.es/viralplus/noticias/8601686/09/17/Si-resuelves-este-rompecabezas-ganaras-un-millon-de-dolares.html

Modificar l'aspecte de la nostra aplicació

Com a part opcional, es podria modificar la vista, és a dir, la percepció que té l'usuari de l'execució del nostre programa. Enlloc de que una **R** simbolitzi una reina, podem fer servir els caràcters unicode per representar-la (♕). També podríem fer servir aquests caràcters per fer requadres i els codis ANSI per dibuixar el tauler en colors. Fixeu-vos que l'únic mètode que hem de modificar és el que hem anomenat **dibuixaTauler(t)**. Mes informació a les pàgines:

https://es.wikipedia.org/wiki/S%C3%ADmbolos_de_ajedrez_en_Unicode
https://www.mclibre.org/consultar/htmlcss/html/html-unicode-simbolos.html#cajas