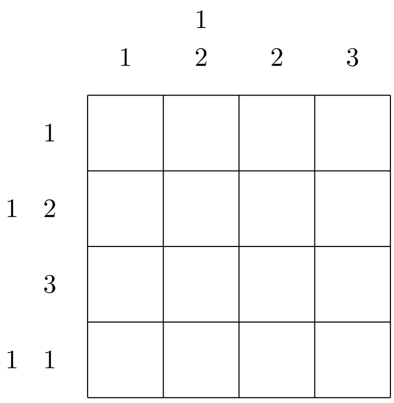
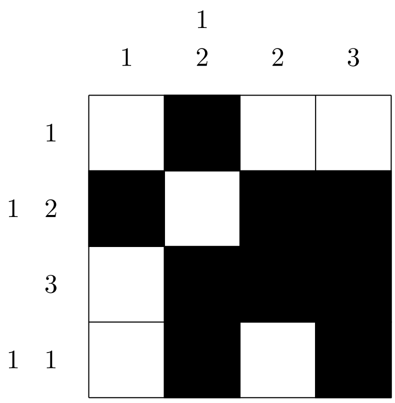
MPSI option info - 30 mars 2022

Résolution des Hanjies

Stéphane Legros Année scolaire 2021-2022

Le jeu de Hanjie est un casse-tête japonais, qui se joue sur une grille de taille $n \times p$ (en général, $n = 20$ et $p = 15$). À chaque ligne et chaque colonne de la grille est associée une suite d'entiers non nuls. Le jeu consiste à retrouver le dessin, obtenu en noircissant certaines cases de la grilles, qui est codé par ces suites d'entiers de la façon suivante : la suite $n_1, \ldots, n_k$ associée à une ligne (ou à une colonne) signifie que la ligne (ou la colonne) est constituée d'un bloc (éventuellement vide) de cases blanches, puis d'un bloc de n_1 cases noires, puis d'un bloc non vide de cases blanches, puis d'un bloc de n_2 cases noires et ainsi de suite jusqu'à un bloc de n_k cases noires suivi par un bloc (éventuellement vide) de cases blanches. Les lignes et colonnes seront numérotées de 0 à $n-1$ et de 0 à $p-1$. Voici un exemple de Hanjie, suivi de sa solution :





Pour manipuler ces grilles, nous utiliserons le type :

`type hanjie = {grille: int array array ; lignes: int list array ; colonnes: int list array}`

avec les conventions suivantes :

- le champ `grille` est une matrice de taille $n \times p$ dont l'entrée d'indice (i, j) contient 0 (resp. 1) si l'on sait que la case (i, j) est blanche (resp. noire) et 2 si l'on ne connaît pas encore sa couleur ;
- le champ `lignes` (resp. `colonnes`) est un tableau de longueur n (resp. p) dont l'entrée d'indice i contient la liste des entiers associée à la i-ème ligne (resp. colonne) (nous numéroterons les lignes de 0 à $n-1$ et les colonnes de 0 à $p-1$).

Nous appellerons `grille initiale` une grille de type `hanjie` dont les entrées du champ `grille` sont toutes égales à 2. Une telle grille sera dite *valide* si elle est admet une et une seule solution, i.e. s'il existe une et une seule façon de colorier la grille en accord avec les listes associées aux lignes et colonnes. Le but de ce TP est d'écrire une fonction `resoudre : hanjie -> hanjie list` qui, appliquée à une grille initiale, renvoie la liste (éventuellement vide) des grilles solutions. Vous pourrez testez vos fonctions en utilisant les hanjies $(h_i)_{1 \leq i \leq 5}$ ainsi que la fonction `dessiner`, que vous trouverez dans le fichier `Hanjie.oml`.

Pour résoudre ce problème, en notant H l'hanjie étudié, nous utiliserons l'analyse suivante :

(a) au début du calcul, nous créons deux tableaux L et C de longueurs respectives n et p : l'entrée $L.(i)$ (resp. $C.(i)$) contient la liste des solutions *a priori* possible pour la i-ème ligne (resp. la i-ème colonne). Ainsi, l'exemple ci-dessus donnera :

```
L.(0)=[[|1;0;0;0|];[|0;1;0;0|];[|0;0;1;0|];[|0;0;0;1|]]
      ⋮
L.(3)=[[|1;0;1;0|];[|1;0;0;1|];[|0;1;0;1|]]
      ⋮
C.(3)=[[|1;1;1;0|];[|0;1;1;1|]]
```

Nous effectuons ensuite en boucle, tant que la situation évolue, les opérations suivantes :

(b) pour chaque i entre 0 et $n-1$ et pour chaque j entre 0 et $p-1$ tels que `H.grille.(i).(j)` vaut 2 et tels que toutes les solutions possibles stockées dans $L.(i)$ donnent la même couleur à la case d'indice (i,j), nous affectons cette couleur à la case `H.grille.(i).(j)`.

(c) nous supprimons des possibilités stockées dans C celles qui ne correspondent pas aux valeurs modifiées à l'étape (b).

(d) nous effectuons l'opération (b) en échangeant les rôles des lignes et des colonnes.

(e) nous supprimons des possibilités stockées dans L celles qui ne correspondent pas aux valeurs modifiées à l'étape (d).

Nous reprenons en boucle les étapes (b)...(e) tant que des cases de $H.grille$ sont modifiées. Cette méthode ne permet pas de résoudre tous les Hanjies valides (elle échoue avec l'exemple h_5), mais elle semble suffisante pour résoudre les cent grilles de mon petit livre Marabout.

Définition : un tableau V à valeur dans $\{0,1\}$ sera dit *adapté* à une liste d'entiers non nuls $[n_1;n_2;\ldots;n_k]$ si V est de la forme :

$$[|\underbrace{0,\ldots,0}_{m_0},\underbrace{1,\ldots,1}_{n_1},\underbrace{0,\ldots,0}_{m_1},\underbrace{1,\ldots,1}_{n_2},\ldots,\underbrace{0,\ldots,0}_{m_{k-1}},\underbrace{1,\ldots,1}_{n_k},\underbrace{0,\ldots,0}_{m_k}|]$$

avec $m_0 \geq 0$, $m_1 > 0$, ...,$m_{k-1} > 0$ et $m_k \geq 0$. Un tel tableau sera codé par la liste $[i_1, i_2, \ldots, i_k]$ des indices des premiers 1 de chaque bloc. Ainsi, le tableau $[|0;0;1;1;1;0;1;1;0;1;1;1;0;0;1|]$ est adapté à la liste $[3;2;3;1]$ et est codé par la liste $[2;6;9;14]$.

1) Écrire une fonction

```
ajouter: int -> int list list -> int list list
```

qui ajoute un entier au début de toutes les listes d'une liste de listes d'entier.

Par exemple, l'appel `ajouter 1 [[2;3;4];[3;4]]` renverra la liste `[[1;2;3;4];[1;3;4]]`.

2) Écrire une fonction

```
crible: int array list -> int -> int -> int array list
```

qui, appliquée à une liste de tableaux d'entiers (de même longueur n) $[V_1;V_2;\ldots;V_k]$, à un entier i compris entre 0 et $n-1$ et à un entier a, renvoie la liste des V_j tels que $V_j.(i)$ est égal à a.

Ainsi, `crible [[|1;0;0;1|];[|0;1;0;1|];[|0;0;1;1|]] 2 0` renverra `[[|1;0;0;1|];[|0;1;0;1|]]`.

3) Écrire une fonction récursive

```
pos: int list -> int -> int -> int list list
```

qui, appliquée à une liste d'entiers $[n_1;n_2;\ldots;n_k]$, à un entier d et à un entier n, renvoie la liste des codes des tableaux de longueur n adaptés à $[n_1;n_2;\ldots;n_k]$ et dont l'indice du premier bloc est au moins égal à d. Ainsi, `pos [2;1] 4 9` renverra (à l'ordre près) la liste `[[4;7];[4,8];[5,8]]`.

4) Écrire une fonction

```
décoder: int list -> int list -> int -> int array
```

qui, appliquée à une liste l_1, à une liste l_2 et à un entier n, renvoie le tableau de longueur n adapté à l_1 et codé par l_2.

5) Écrire une fonction

```
possibilités : int list -> int -> int array list
```

qui, appliquée à une liste $[n_1, n_2, \ldots, n_k]$ et à un entier n renvoie la liste des tableaux V de longueur n adaptés à la liste $[n_1, n_2, \ldots, n_k]$.

6) Écrire une fonction

```
init: hanjie -> int array list array*int array list array
```

qui calcule le couple (L, C) associé à un Hanjie.

7) Écrire une fonction

```
test: int array list -> int -> int
```

qui, appliquée à une liste de tableaux d'entiers $[V_1; V_2; \ldots; V_k]$ et à un entier j renvoie :

- 0 si tous $V_i.(j)$ valent 0 ;
- 1 si tous $V_i.(j)$ valent 1 ;
- 2 sinon.

8) Écrire une fonction

```
test_lignes : hanjie -> int array list array -> int*int list
```

qui, appliquée à un hanjie H et au tableau L, effectue l'opération (b) et renvoie la liste des indices (i, j) tels que la case (i, j) a été modifiée. Écrire de même une fonction `test_colonnes` qui effectue l'opération (d).

9) Écrire une fonction

```
simplifie_colonnes : hanjie -> int array list array -> int*int list-> unit
```

qui, appliquée à un hanjie H, au tableau C et à la liste correspondant aux couples renvoyés par `test_lignes`, effectue l'opération (c). Écrire de même une fonction `simplifie_lignes` qui effectue l'opération (e).

10) Écrire une fonction

```
résolution : hanjie -> unit
```

qui applique l'algorithme de résolution décrit précédemment.