DS 2

Option informatique, deuxième année

Julien Reichert

Exercice 1 : Logique et calcul des propositions

Vous avez été sélectionné(e) pour participer au jeu “Cherchez les Clés du Paradis". Le jeu se déroule en trois épreuves, au cours desquelles vous devez collecter des clés vertes. À l'issue de chacune d'entre elles, vous passez à l'épreuve suivante en cas de succès et êtes éliminé(e) en cas d'échec. Jean-Pierre Pendule, le célèbre animateur, vous accueille pour la première épreuve. Il vous explique la règle du jeu. Devant vous, deux boîtes et sur chacune d'entre elles une inscription. Chacune des boîtes contient soit une clé verte, soit une clé rouge. Vous devez choisir l'une des boîtes : si le résultat est une clé rouge, alors vous quittez le jeu, si c'est une clé verte vous êtes qualifié(e) pour l'épreuve suivante.

Jean-Pierre Pendule dévoile les inscriptions sur chacune des boîtes et vous affirme qu'elles sont soit vraies toutes les deux, soit fausses toutes les deux :

— sur la boîte 1, il est écrit “Une au moins des deux boîtes contient une clé verte" ;
— sur la boîte 2, il est écrit “Il y a une clé rouge dans l'autre boîte".

Dans toute cette partie, on note P_i la proposition affirmant qu'il y a une clé verte dans la boîte i.

Question 1.1 : Donner une formule de la logique des propositions représentant la phrase écrite sur la boîte 1.

Question 1.2 : Donner de même une formule de la logique des propositions pour l'inscription de la boîte 2.

Question 1.3 : Donner une formule représentant l'affirmation de l'animateur. Simplifier cette formule de sorte à n'obtenir qu'une seule occurrence de chaque P_i.

Question 1.4 : Quel choix devez-vous faire pour continuer le jeu à coup sûr ?

Avec les mêmes règles du jeu, l'animateur vous propose deux nouvelles boîtes portant les inscriptions suivantes :

— sur la boîte 1, il est écrit “Il y a une clé rouge dans cette boîte, ou bien il y a une clé verte dans la boîte 2" ;
— sur la boîte 2, il est écrit “Il y a une clé verte dans la boîte 1".

Question 1.5 : Donner une formule de la logique des propositions pour chaque affirmation.

Question 1.6 : Sachant qu'encore une fois les deux affirmations sont soit vraies toutes les deux, soit fausses toutes les deux, donner le contenu de chaque boîte. En déduire votre choix pour remporter la deuxième clé verte.

Jean-Pierre Pendule vous dévoile une troisième boîte et vous explique les règles du jeu. Dans une des boîtes se cache la clé qui vous permet de remporter la victoire finale. Dans une autre boîte se cache une clé rouge qui vous fait tout perdre. La dernière boîte est vide. Encore une fois, chacune des boîtes porte une inscription :

— sur la boîte 1, il est écrit “La boîte 3 est vide" ;
— sur la boîte 2, il est écrit “La clé rouge est dans la boîte 1" ;
— sur la boîte 3, il est écrit “Cette boîte est vide".

L'animateur affirme que l'inscription portée sur la boîte contenant la clé verte est vraie, celle portée par la boîte contenant la clé rouge est fausse. L'inscription affichée sur la boîte vide est aussi vraie.

Question 1.7 : Donner une formule de la logique des propositions pour chaque inscription.

Question 1.8 : Donner une formule de la logique des propositions synthétisant l'information que vous apportée l'animateur.

Question 1.9 : En supposant que la clé verte est dans la boîte 2, montrer par l'absurde que l'on aboutit à une incohérence.

Question 1.10 : Donner alors la composition des trois boîtes.

Exercice 2 : Structures de données et algorithmes

À travers l'étude d'un jeu de société, ce sujet s'intéresse aux mouvements de robots, qui possèdent des capacités limitées de localisation. Avec le développement de la robotique, plusieurs problèmes de ce type font l'objet de nombreuses recherches : parcours minimum pour examiner une surface donnée, stratégies collectives avec plusieurs robots en interaction proche, nombre de robots nécessaires pour que tous les points d'une surface avec obstacles soient accessibles, etc.

Ce sujet porte sur la résolution de la situation pratique du jeu Ricochet Robots créé par Alex Randolph en 1999. Ce jeu se déroule sur un plateau de 16×16 cases, avec 4 robots et des murs. À chaque mouvement, un des robots se déplace, dans une des quatre directions, jusqu'à ce qu'il rencontre un obstacle (un mur ou autre robot). Le but est de trouver le nombre de coups minimal pour déplacer un robot particulier de son point de départ jusqu'à une case précise du plateau. Un exemple en 8 coups est donné figure 1.

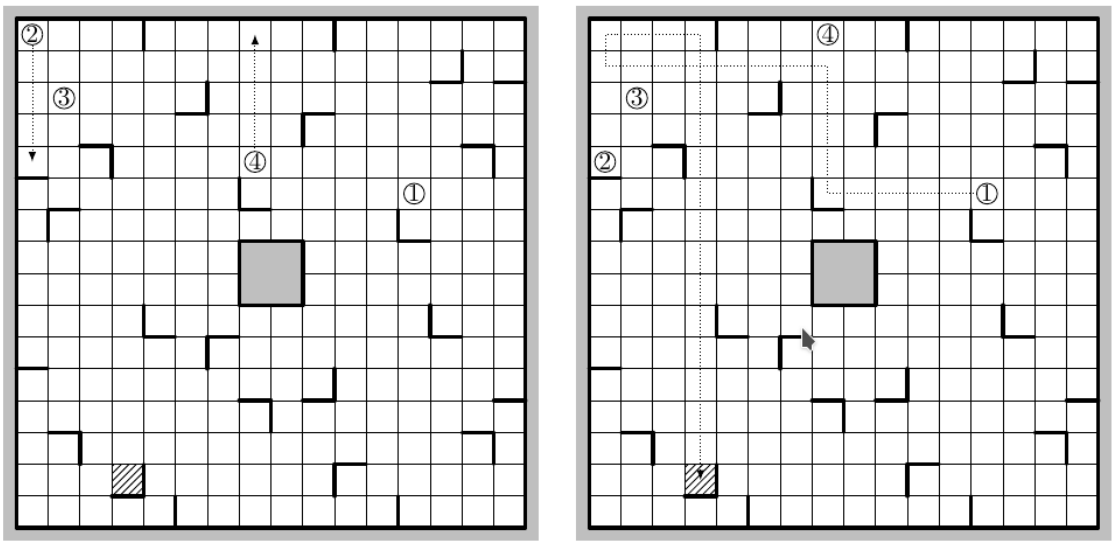

FIGURE 1 – Le jeu des robots : le but est d'amener le robot 1 sur la case hachurée. À gauche : deux déplacements des robots 2 et 4 ; à droite : six déplacements du robot 1. Le jeu est résolu en 8 mouvements (solution optimale)

On rappelle la définition des fonctions suivantes :

- `Array.copy : 'a array -> 'a array` telle que l'appel `Array.copy v` renvoie un nouveau tableau contenant les valeurs contenues dans `v` ;
- `Array.make : int -> 'a -> 'a array` telle que l'appel `Array.make n x` renvoie un nouveau tableau de longueur `n` initialisé avec des éléments égaux à `x` ;
- `Array.make_matrix : int -> int -> 'a -> 'a array` telle que l'appel `Array.make_matrix p q x` renvoie une nouvelle matrice à `p` lignes et `q` colonnes initialisée avec des éléments égaux à `x`.

Déplacement d'un robot dans une grille

On considère pour le moment une grille sans robots du jeu Ricochet Robots. Notons N le nombre de cases par ligne et colonne de la grille (16 dans le jeu originel). **Dans les fonctions demandées, on supposera que N est une variable globale notée `n` en Caml**. On numérote chaque case par un couple (a, b) de $[|0, N-1|]^2$, correspondant à la ligne a et à la colonne b. On numérote également les lignes horizontales et verticales séparant les cases à l'aide d'un entier de $[|0, N|]$, de sorte que la case (a, b) est délimitée par les lignes horizontales a (au dessus) et $a+1$ (en dessous), de même que par les lignes verticales b (à gauche) et $b+1$ (à droite) (figure 2).

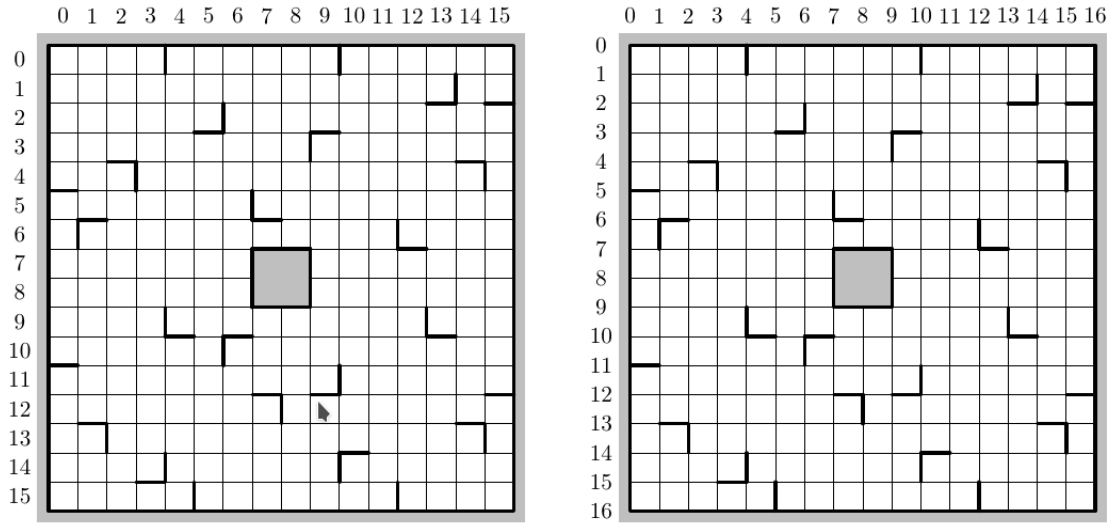

FIGURE 2 – À gauche : numérotation des cases par ligne/colonne ; à droite : numérotation des lignes horizontales et verticales

Pour représenter en Caml la grille avec ses obstacles, on se donne deux tableaux de taille N. Le premier contient les obstacles verticaux sur chacune des lignes, le second contient les obstacles horizontaux sur chacune des colonnes. Un obstacle est donné par le numéro de la ligne (verticale ou horizontale) auquel il appartient. Les obstacles sur une ligne (ou colonne) sont donnés sous la forme d'un tableau ordonné dans l'ordre croissant. Par exemple, la représentation du plateau de la figure 1 est donnée ci-après.

```
let obstacles_lignes = [| [|0; 4; 10; 16|]; [|0; 14; 16|]; [|0; 6; 16|];
   [|0; 9; 16|]; [|0; 3; 15; 16|]; [|0; 7; 16|]; [|0; 1; 12; 16|]; [|0; 7; 9; 16|];
   [|0; 7; 9; 16|]; [|0; 4; 13; 16|]; [|0; 6; 16|]; [|0; 10; 16|]; [|0; 8; 16|];
   [|0; 2; 15; 16|]; [|0; 4; 10; 16|]; [|0; 5; 12; 16|] |];;

let obstacles_colonnes = [| [|0; 5; 11; 16|]; [|0; 6; 13; 16|]; [|0; 4; 16|];
   [|0; 15; 16|]; [|0; 10; 16|]; [|0; 3; 16|]; [|0; 10; 16|]; [|0; 6; 7; 9; 12; 16|];
   [|0; 7; 9; 16|]; [|0; 3; 12; 16|]; [|0; 14; 16|]; [|0; 16|]; [|0; 7; 16|];
   [|0; 2; 10; 16|]; [|0; 4; 13; 16|]; [|0; 2; 12; 16|] |];;
```

Notez que les bordures de la grille sont considérées comme des obstacles. Ainsi, les entiers 0 et N sont présents dans les tableaux associés à chaque ligne/colonne.

Question 2.1 : Écrire une fonction `dichotomie a t` de signature `int -> int array -> int` telle que si `t` est un tableau d'entiers strictement croissants et `a` un élément supérieur ou égal au premier élément du tableau et strictement inférieur au dernier, la fonction renvoie l'unique indice `i` tel que `t.(i)` $\leq$ `a` $<$ `t.(i+1)`. La fonction doit avoir une complexité logarithmique en la taille du tableau.

On considère un robot positionné en (a, b), avec $0 \leq a, b < N$. Il peut se déplacer dans les quatre directions cardinales ouest/est/nord/sud.

Question 2.2 : Écrire une fonction `deplacements_grille (a,b)` de signature `int * int -> (int * int) array` fournissant les 4 cases atteintes par les déplacements en question, sous forme d'un tableau à 4 éléments (ouest/est/nord/sud). Si le robot ne peut pas bouger dans une direction donnée (car il est contre un obstacle), on considérera que le résultat du déplacement dans cette direction est la case (a, b) elle-même. Les deux tableaux `obstacles_lignes` et `obstacles_colonnes` sont des variables globales.

Question 2.3 : Écrire une fonction `matrice_deplacements ()`, de type `unit -> (int * int) array array array` produisant une matrice `m` telle que `m.(a).(b)` contienne le tableau des déplacements possibles pour un robot depuis la case (a, b), et ce pour tous $0 \leq a, b < N$. Donner la complexité de création de la matrice.

On cherche maintenant à intégrer les positions d'autres robots dans le déplacement d'un robot. On utilise la fonction précédente pour créer une matrice `mat_deplacements` que l'on considérera comme globale.

Question 2.4 : Écrire une fonction `modif t (a,b) (c,d)` de signature

```
(int * int) array -> int * int -> int * int -> unit
```

telle que si `t` est le tableau de taille 4 donnant les déplacements ouest/est/nord/sud d'un robot placé en (a, b) dans la grille ne contenant pas d'autres robots, et (c, d) la position d'un autre robot, alors la fonction modifie si nécessaire le tableau `t` en prenant en compte le robot en (c, d).

On s'intéresse maintenant au déplacement d'un robot situé en (a, b) dans la grille, avec d'autres robots éventuellement présents, dont les positions sont stockées dans une liste.

Question 2.5 : Déduire des questions précédentes une fonction `deplacements_robots (a,b) q` de signature

```
int * int -> (int * int) list -> (int * int) array
```

donnant les déplacements ouest/est/nord/sud d'un robot situé en (a, b) dans la grille, les positions des autres robots étant stockées dans la liste `q`. On ne modifiera pas la matrice `mat_deplacements` : on souhaite une copie modifiée de `mat_deplacements.(a).(b)`.

Question 2.6 : Si on suppose que la solution optimale demande au plus k mouvements, une solution possible pour résoudre le jeu Ricochet Robots consiste à générer toutes les suites possibles de k déplacements. Avec 4 robots en tout, estimer la complexité d'une telle approche (on utilisera la notation O).

La suite du problème a pour objet de proposer une solution plus efficace pour la résolution du jeu Ricochet Robots.

Quelques fonctions utilitaires

Une fonction de tri

Question 2.7 : Écrire une fonction `insertion x q` de signature `'a -> 'a list -> 'a list` prenant en entrée un élément `x` et une liste `q` triée dans l'ordre croissant, et renvoyant une liste triée dans l'ordre croissant, constituée des éléments de `q` et `x`.

Question 2.8 : En déduire une fonction `tri_insertion q` de signature `'a list -> 'a list` permettant de trier une liste dans l'ordre croissant.

Question 2.9 : Rappeler la complexité de ce tri dans le pire et le meilleur cas. Que peut-on dire de la complexité si dans la liste `q`, tous les éléments excepté peut-être un sont dans l'ordre croissant ?

Quelques fonctions sur les listes

Question 2.10 : Écrire une fonction `mem1 x q` de signature `'a -> ('a * 'b) list -> bool` testant l'appartenance d'un couple dont le premier élément est `x` dans la liste `q`.

Question 2.11 : Écrire une fonction `assoc x q` de signature `'a -> ('a * 'b) list -> 'b` renvoyant, s'il existe, l'élément `y` du premier couple `(x,y)` appartenant à la liste `q`.

Tables de hachage

Dans l'optique de résoudre le problème du jeu des robots, nous allons travailler sur un graphe dont les sommets seront étiquetés par les positions des robots. Le nombre de sommets possibles étant élevé, il est nécessaire d'utiliser une structure de données adaptée pour travailler sur ce graphe. Nous allons donc réaliser une structure de dictionnaire permettant, en particulier, de tester facilement si un sommet a déjà été vu ou non et d'associer un sommet à chaque sommet découvert.

Une structure de dictionnaire est un ensemble de couples (clé, élément), les clés (nécessairement distinctes) appartenant à un même ensemble K, les éléments à un ensemble E. La structure doit garantir les opérations suivantes :
- recherche d'un élément connaissant sa clé ;
- ajout d'un couple (clé, élément) ;
- suppression d'un couple connaissant sa clé.

Une structure de dictionnaire peut-être réalisée à l'aide d'une table de hachage. Cette table est implantée dans un tableau de w listes (appelées `alvéoles`) de couples (clé, élément). Ce tableau est organisé de façon à ce que la liste d'indice i contienne tous les couples (k, e) tels que $h_w(k) = i$ où $h_w : K \to [|0, w-1|]$ s'appelle `fonction de hachage`. On appelle w la `largeur de la table` de hachage et $h_w(k)$ le `haché` de la clé k.

Ainsi pour rechercher ou supprimer l'élément de clé k, on commence par calculer son haché qui détermine l'alvéole adéquate et on est alors ramené à une action sur la liste correspondante. De même pour ajouter un nouvel élément au dictionnaire on l'ajoute à l'alvéole indiquée par le haché de sa clé.

Nous commençons par nous doter d'une famille de fonctions h_w, pour les listes de couples de $[|0, N-1|]^2$. Un hachage naturel d'une liste comportant les couples $(a_i, b_i)_{0 \leq i < p}$ avec $0 \leq a_i, b_i < N-1$ est donnée par :

$$P_w(N) = \left(\sum_{i=0}^{p-1} (a_i + b_i N) N^{2i} \right) \text{ modulo } w$$

Autrement dit, on évalue le polynôme dont les coefficients sont donnés par les a_i et b_i en N, et on ne considère que le reste dans la division euclidienne par w. On rappelle qu'on a supposé que N est une variable globale.

Question 2.12 : Écrire une fonction récursive `hachage_liste w q` de signature `int -> (int * int) list -> int` calculant la quantité précédente.

On fixe désormais une largeur de hachage w. Un bon choix pour w serait par exemple un nombre premier ni trop petit, ni trop grand, comme 997. Pour les listes, on considérerait alors la fonction de hachage h_{997} donnée en Caml par `hachage_liste 997`. On définit en toute généralité le type suivant :

```
type ('a, 'b) table_hachage = {
   hache: 'a -> int;
   donnees: ('a * 'b) list array;
   largeur: int};;
```

Question 2.13 : Écrire une fonction `creer_table h w` de signature `('a -> int) -> int -> ('a, 'b) table_hachage` telle que `creer_table h w` renvoie une nouvelle table de hachage vide de largeur `w` munie de la fonction de hachage `h`.

Question 2.14 : Écrire une fonction `recherche t k` de signature `('a, 'b) table_hachage -> 'a -> bool` renvoyant un booléen indiquant si la clé k est présente dans la table t. On pourra utiliser les fonctions de la partie précédente.

Question 2.15 : Écrire une fonction `element t k` de signature `('a, 'b) table_hachage -> 'a -> 'b` renvoyant l'élément e associé à la clé k dans la table t, si cette clé est bien présente dans la table.

Question 2.16 : Écrire une fonction `ajout t k e` de signature `('a, 'b) table_hachage -> 'a -> 'b -> unit` ajoutant l'entrée (k, e) à la table de hachage t. On n'effectuera aucun changement si la clé est déjà présente.

Question 2.17 : Écrire enfin une fonction `suppression t k` de signature `('a, 'b) table_hachage -> 'a -> unit` supprimant l'entrée de la clé k dans la table t. On n'effectuera aucun changement si la clé n'est pas présente.

Exercice 3 : Graphes

On définira au préalable un type adapté pour les graphes en Caml.

Question 3.1 : Écrire une fonction qui prend en entrée un graphe et deux sommets et qui détermine s'il existe un chemin du premier au deuxième sommet. La fonction doit retourner une erreur si l'un des sommets n'existe pas dans le graphe.

Question 3.2 : Écrire une fonction qui prend en entrée un graphe et qui détermine si le graphe est fortement connexe.

Exercice 4 : Énigme de logique et d'arithmétique

La grille[1] ci-après est composée de 7 lignes de taille 7. Dans les cellules de chaque ligne et de chaque colonne, il faut placer deux cases noircies ainsi que chaque nombre parmi 2, 3, 4, 10 et 11 une et une seule fois. Les nombres au début de chaque ligne (resp. colonne) indiquent la somme des nombres qui sont entre les deux cases noircies de la ligne (resp. colonne) en question. La solution est unique (mais ce n'est pas une information à utiliser comme argument). Pour avoir la totalité des points, il faudra expliquer des passages non triviaux du raisonnement.

	15	15	28	?	13	13	3
18							
4							
0							
?							
11							
12							
15							

1. [Source : magazine "der Skatfreund" numéro 1 de 2019]