Mastermind - partie 2

Exercice 1. Préliminaire

Lors d'un coup au mastermind, la personne qui doit deviner propose une solution `a,b,c,d`. Son adversaire lui répond alors par un couple d'entier `(bien,mal)` le nombre de couleurs bien et mal placées.

La difficulté pour programmer un ordinateur qui joue au mastermind est de tenir compte de cette information. Lors du TD précédant nous avons écrit une fonction `bien_mal(a,b,c,d, A,B,C,D)` nous rendant les nombres bien et mal, il s'agit donc de l'utiliser.

Considérons donc que le coup `a,b,c,d, bien,mal`. Je me pose la question de savoir si `A,B,C,D` pourait être la solution.

a. Montrer que la `A,B,C,D` peut être la solution *si et seulement si* `bien_mal(a,b,c,d, A,B,C,D) = bien,mal`

On appelle coup_correct : $\left([|0,5|]^4 \times [|0;4|]^2\right) \times [|0,5|]^4 \to \{\text{vrai}, \text{faux}\}$ qui a `((a,b,c,d, bien,mal), A,B,C,D)` répond vrai *ssi* `A,B,C,D` peut être la solution de `a,b,c,d, bien,mal`.

Supposont maintenant que nous disposons d'un ensemble de coups $c_1, c_2, \ldots, c_n$.

b. Montrer que la combinaison `A,B,C,D` peut être la solution *ssi* Pour tout $i \leq n$: coup_correct(c_i, A, B, C, D) est vrai. On note correcte$(A, B, C, D, (c_1, \ldots, c_n))$ cette fonction.

On munit l'ensemble des combinaisons de l'ordre lexicographique (*ie* l'ordre du dictionnaire) :

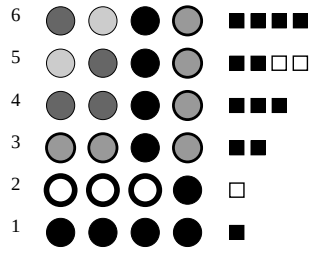
$(a, b, c, d) < (e, f, g, h)$ *ssi* $a < e$ ou $\Big(a = e$ et $b < f$ ou $\Big(b = f$ et $[\ldots]\Big)\Big)$

c. Quel est l'élément minimal de cet ensemble ? On le note x_0

On suppose que x_1 a été joué et note $c_1 = (x_1, \text{bien}_1, \text{mal}_1)$ le premier coup. Soit $n > 1$, on suppose que $c_1, \ldots, c_{n-1}$ aient été définis. On note x_n la combinsaison minimale telle que correcte$(A, B, C, D, (c_1, \ldots, c_{n-1}))$ soit vrai et on pose $c_n = (x_n, \text{bien}_n, \text{mal}_n)$ le $n^{\text{ième}}$ coup.

d. Montrer que la suite x_n est croissante et stationnaire. Soit x sa limite. Montrer que x est la solution du mastermind.

Exercice 2. Et maintenant du Maple



Le but de cet exercice est d'implémenter en Maple les résultats vu dans le premier exercice.

Un exemple est fournis sur le dessin ci contre. La première réponse montre qu'il y a un 0 de bien placé. La combinaison suivante ayant un unique 0 est 0,1,1,1. L'adversaire répondant qu'il y a une boule de mal placée, la combinaison possible suivante est 2,0,2,2, etc... Au bout de 6 coups, l'adversaire trouve la solution.

a. Écrire une fonction `coup_suivant := proc( a,b,c,d )` qui prend en argument 4 nombres compris entre 0 et 5 et qui rend la suite de quatre entiers correspondants à l'élément suivant selon l'ordre lexicographique.

b. Écrire une fonction `coup_correct := proc( A,B,C,D a,b,c,d, bien,mal)` qui rend `true` *ssi* `A,B,C,D` peut être la solution.

Une partie est constituée de la liste des coups joués. Par la suite on stockera la partie dans une liste [[a,b,c,d,bien,mal],[a',b',c',d', bien', mal'], [a", b", c", d", bien", mal"],...]. Par exemple, la partie ci dessus est avant la ligne 5 : [[2,0,3,3, 3,0],[2,0,2,2, 2,0], [0,1,1,1, 0,1],[0,0,0,0, 1,0]]

c. Écrire une fonction `correcte := proc(A,B,C,D, partie)` qui rend `true` si la combinaison `A,B,C,D` pourrait être la solution de `partie`.

d. En s'inspirant des lignes ci-dessous, implémenter l'algorithme en maple

```
mon_coup <- 0,0,0,0;
nombre_de_coup_joues <- 0;        partie <- []

tantque nombre_de_coup_joues < 10 faire
   dire à l'humain: je joue la combinaison mon_coup
   ligne := readline(-1); bon,mal := op(scanf(ligne,"%d %d"));
   si bon = 4 alors
      j'ai gagné!
   sinon
      ajouter le coup courant à la partie.
      tantque mon_coup ne pourrait pas être la solution faire
         si coup_suivant existe
             alors  mon_coup = coup_suivant(mon_coup);
             sinon écrire ``il n'y a pas de solution!''.
      finfaire
   nombre_de_coup_joues = nombre_de_coup_joues + 1
finfaire
```

Exercice 3. Pour aller plus loin

Sur des exemples précis, cet algorithme semble trouver la bonne réponse mais on peut se poser différentes questions sur la viabilité (nombre de coups en moyenne, dans le pire cas,etc...). Le nombre de combinaison n'étant que de 6^4, il est possible de toutes les étudier.

a. Modifier votre programme pour le faire jouer tout seul sur une combinaison fixés `a,b,c,d` et qu'il vous rende le nombre de coups qu'il a mis pour trouver

b. Utiliser la question précédente pour faire des statistiques sur le nombre de coups que met cet algorithme à trouver la solution (étudier la moyenne, le pire cas, ...)

c. Refaire les même statistiques mais en remplaçant la combinaison de départ par une nouvelle combinaison (par exemple 4,4,3,1) et étudier les performances. Est-ce mieux ?