e3a 2015 - PSI 1
durée 4 heures - calculatrices interdites

Exercice 1

But de l'exercice

Le jeu d'echec se joue sur un échiquier, c'est à dire sur un plateau de 8×8 cases. Ces casessont référencées de $a1$ à $h8$ (voir figures).
Une pièce, appelée le cavalier, se déplace suivant un "L" imaginaire d'une longueur de deux cases et d'une largeur d'une case.
Exemple (figure 1) : un cavalier situé sur la case $d4$ atteint, en un seul déplacement, une des huits cases $b5, c6, e6, f5, f3, e2, c2$ ou $b3$.
Dans toute la suite de l'exercice, on appellera **case permise** toute case que le cavalier peut atteindre en un déplacement à partir de sa position.

Le but de cet exercice est d'écrire un programme faisant parcourir l'ensemble de l'échiquier à un cavalier **en ne passant sur chaque case qu'une et une seule fois**

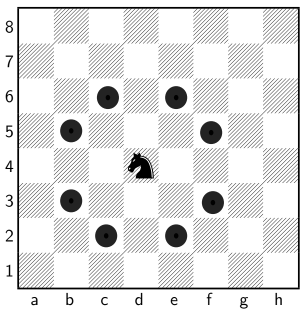

FIGURE 1
déplacements permis

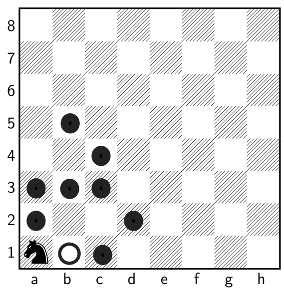

FIGURE 2
un exemple

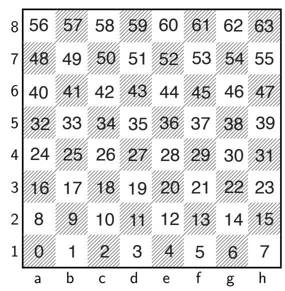

FIGURE 3
numérotation

Motivation et méthode retenue

Une première idée est de faire parcourir toutes les cases possibles à un cavalier en listant à chaque déplacement les cases parcourues. Lorsque celui-ci ne peut plus avancer, on consulte le nombre de cases parcourues.

- Si ce nombre est égal à $64 = 8 \times 8$, alors le problème est résolu.
- Sinon, il faut revenir en arrière et tester d'autres chemins.

1. **Exemple** : on considère le parcours suivant d'un cavalier démarrant en $a1$ (figure 2)

$$a1, b3, c1, a2, c3, b5, a3, c4, d2$$

Avec ce début de parcours, au déplacement suivant :

(a) le cavalier va en $b1$. Peut-il accomplir sa mission ?

(b) le cavalier ne va pas en $b1$. Peut-il accomplir sa mission ?

Il convient donc dans la résolution du problème proposé d'éviter de se retrouver dans la situation repérée en cette première question.

Dans tout ce qui suit, nous nommerons **coordonnées** d'une case la liste d'entiers $[i, j]$ où i représente le numéro de ligne et j le numéro de colonne (tous deux compris entre 0 et 7). Par exemple, la case $b3$ a pour coordonnées $[2, 1]$.
D'autre part, les cases sont numérotées de 0 à 63 en partant du coin gauche comme indiqué en figure 3.
Nous appellerons *indice* d'une case, l'entier $n \in [|0, 63|]$ ainsi déterminé. $b3$ a, par exemple, un indice égal à 17.

2. Ecrire une fonction `indice` qui prend en argument la liste des coordonnées d'une case est renvoie son indice. Ainsi, `indice([2,1])` doit être égal à 17.
3. Ecrire une fonction *coord* qui à l'indice n d'une case associela liste $[i, j]$ de ses coordonnées. Ainsi `coord(17)` doit être égal à $[2, 1]$.
4. On considère la fonction Python `CasA` suivante :

```
def CasA(n):
    Deplacements=[[1,-2],[2,-1],[2,1],[1,2],[-1,2],[-2,1],[-2,-1],[-1,-2]]
    L=[]
    i,j=Coord(n)
    for d in Deplacements:
        u=i+d[0]
        v=j+d[1]
        if u>=0 and u<8 and v>=0 and v<8:
            L.append(Indice([u,v]))
    return(L)
```

   (a) Que renvoient `CasA(0)` et `CasA(39)`.
   (b) Expliquer en une phrase ce que fait cette fonction.
5. Ecrire une fonction `Init` ne prenant aucun argument et qui modifie deux variables globales `ListeCA` et `ListeCoups`. `ListeCoups` recevra la iste vide. `ListeCA` recebra une listede 64 éléments. Chaque élément `listeCA[n]` (pour $0 \leq n \leq 63$) devra contenir la liste des indices des cases qu'un cavalier peut atteindre en un coup à partir de la cas d'indice n.
6. Après exécution de la fonction `Init()`, la commande `ListeCA[n]` renvoie-t-elle $[5]$, $[10, 17]$, $[10, 17, 0]$, $[17, 0, 10]$, $[]$ ou une autre valeur ?
7. Au cours de la recherche, lorsqu'on déplace le cavalier vers la case d'indice n, cet indice n doit être retiré de la liste des *cases permises* à partir de la position n.
   Exemple : après exécution de la fonction `Init()`, la liste des cases permises depuis $b1$ est $[a3, c3, d2]$ et `ListeCA[1]=[16,18,11]`. La liste des cases permises depuis $a3$ est $[b5, c4, c2, b1]$ et `ListeCA[16]=[33,26,10,1]`.
   Puis, on choisit de commencer le parcours en posant le cavalier en $b1$. Cette case doit donc être retirée de la liste des cases permises de $a3$, $c3$ et $d2$. En particulier pour $a3$, la liste `ListeCA[16]` devient `[33,26,10]`.
   Cette méthode nous permet de détecter les blocages : le cavalier arrive sur la case d'indice n, n est alors retiré de toutes les listes `ListeCA[k]` pour toute case k permise pour n. Si dès lors l'une de ces listes devient vide, nous dirons que nous somme alors dans une *situation critique*, cela signifiera que la case d'indice k ne peut plus être atteinte que depuis la case d'indice n. Par conséquent,
   - si le cavalier se déplace sur une autre case que celle d'indice k, alors cette dernière ne pourra plus jamais être atteinte ;
   - si le cavalier se déplace sur la case d'indice k, il est bloqué pour le coup suivant. Soit la mission est accomplie, soit le cavalier n'a pas parcouru toutes les cases.

Le programme va réaliser la recherche en maintenant à jour la variable globale `ListeCoups` afin qu'elle contienne en permanence la liste des positions successives occupées par lecavalier au cours de ses tentatives de déplacement. Nous avons alors besoin d'écrire trois fonctions.

(a) Ecrire une fonction `OccupePosition` qui
- prend comme argument un entier n (indice d'une case), l'ajoute à la fin de la variable globale `ListeCoups`,
- puis enlève n de toutes les listes `ListeCA[k]` pour toutes les cases k permises depuis la case d'indice n,
- renvoie enfin la valeur `True` si nous sommes dans une situation critique et `False` sinon.

On pourra utiliser la méthode `remove` *qui permet de retirer d'une liste le premier élément égal à l'argument fouri. Si l'argument ne fait pas partie de la liste, une erreur sera retournée*

```
L=[1,2,3,4,5,6]
L.remove(2) # modifie L en [1,3,4,5,6]
L.remove(6) # provoque une erreur
```

(b) Ecrire une fonction `LiberePosition` qui ne prend pas d'argument et qui
- récupère le dernier élément n de la variable globale `ListeCoups` (i.e. l'indice de la dernière case jouée à l'aide de la fonction `OccupePosition`),
- puis l'enlève de `ListeCoups`,
- et enfin, qui ajoute n à toutes les listes `ListeCA[k]` pour toutes les cases d'indice k permises depuis la case d'indice n.

On pourra utiliser la méthode `pop` *qui renvoie le dernier élément d'une liste et le supprime de cette même liste.*

```
L=[1,2,3,4,2,5,2]
n=L.pop()  #n=2 et L=[1,2,3,4,2,5]
```

(c) Ecrire une fonction `TestePosition` d'argument un entier n (indice d'une case) qui :
- occupe la position d'indice n,
- vérifie si la situation est critique.
  Si c'est le cas, la fonction vérifiera si les 63 cases sont occupées et, dans ce cas renverra `True` pour indiquer que la rececherche est terminée. Si les 63 cases ne sont pas occupées, la fonction libérera la case d'indice n et renverra `False`.
  Dans le cas contraire, la fonction vérifiera avec `TestePosition` toutes les cases d'indice k jouables après celle d'indice n (on prendra garde à affecter une variable locale avec la liste `ListeCA[n]` puisque celle-ci risque d'être modifiée lors des appels suivants). La fonction retournera `True` dès que l'un des appels à `TestePosition` retourne `True` ou libérera la case d'indice n et retournera `False` sinon.

8. Afin de réduire notablement la complexité temporelle du programme, on part du proncipe qu'il faut tester en priorité les cases ayantv le moins de cases permises possibles. On appellera *valuation* d'une case d'indice n le nombre de cases permises pour cette case.

(a) Ecrire une fonction `valuation` qui prend comme argument un indice n de case en entrée et renvoie la valuation de cette case.

(b) Ecrire une fonction `Fusion` qui prend comme arguments deux listes A et B d'entiers entre 0 et 63 ; on suppose ces listes triées par ordre croissant de valuation de leurs éléments ; l'appel `fusion(A,B)` retourne comme valeur la liste fusionnée de tous les éléments de `A` et `B` triée par ordre croissant de valuation de ses éléments.

(c) Ecrire une fonction `TriFusion` qui prend en argument une liste `L` d'entiers compris entre 0 et 63 et qui retourne comme valeur la liste de tous les éléments de `L` triée par valuation croissante de ses éléments.

(d) Modifier la fonction `TestePosition` pour qu'elle agisse ainsi que l'on a décidé en début de question.

Exercice 2

1. Soit $A = \begin{pmatrix} 0 & 1 \\ y-4 & 2x \end{pmatrix} \in \mathcal{M}_2(\mathbb{R})$. Déterminer une condition nécessaire et suffisante pour que A soit diagonalisable dans $\mathcal{M}_2(\mathbb{R})$.
2. On note $E_1 = \{u \in \mathbb{R}^+ /\ u^2 \notin \mathbb{N}\}$ et E_2 son complémentaire dans $\mathbb{R}^+$. Prouver que E_2 est un ensemble dénombrable.
3. Soient $(\Omega, \mathcal{A})$ un espace probabilisable et f définie de $\mathbb{R}^+$ dans $\mathbb{R}$ par
$$\forall u \geq 0,\ f(u) = \begin{cases} 0 & \text{si } \ u^2 \notin \mathbb{N} \\ \frac{\lambda}{2^{u^2}} & \text{sinon} \end{cases}$$
Déterminer λ pour qu'il existe une probabilité $\mathbb{P}$ tel que f soit la loi de probabilité d'une variable aléatoire X définie sur Ω et à valeurs dans $\mathbb{R}^+$. Préciser $X(\Omega)$.
4. Déterminer $X^2(\Omega)$ et la loi de probabilité de X^2.
5. Déterminer l'espérance $\mathbb{E}(X^2)$ de la variable aléatoire X^2.
6. Déterminer la fonction génératrice de la variable aléatoire X^2. Retrouver alors la valeur de $\mathbb{E}(X^2)$ obtenue à la question précédente.
7. Soit Y une variable aléatoire définie sur Ω, indépendante de la variable aléatoire X et suivant la loi
$$\forall u \in \mathbb{R}^+,\ \mathbb{P}(Y = u) = \begin{cases} 0 & \text{si } \ u \notin \mathbb{N} \\ \frac{1}{2^{u+1}} & \text{sinon} \end{cases}$$
Soit alors Z la variable aléatoire définie sur Ω par $Z = X^2 + Y$. Déterminer la fonction génératrice de Z. En déduire sa loi de probabilité.
8. Déterminer enfin la probabilité pour que la matrice $A = \begin{pmatrix} 0 & 1 \\ Y-4 & 2X \end{pmatrix} \in \mathcal{M}_2(\mathbb{R})$ soit diagonalisable.

Exercice 3

On pose, lorsque cela est possible
$$f(x) = \int_1^{+\infty} \frac{dt}{t^x \sqrt{t^2 - 1}}$$

1. Déterminer l'ensemble de définition I de f.
2. En justifiant son existence, calculer $\int_0^{+\infty} \frac{dx}{e^x + e^{-x}}$.
3. Calculer $f(1)$. *On pourra utiliser l'application* $\varphi\ :\ u > 0 \mapsto \mathrm{ch}(u)$.
4. Calculer $f(2)$. *On pourra remarquer que la dérivée de* $x \mapsto \frac{\mathrm{sh}(x)}{\mathrm{ch}(x)}$ *est égale à* $x \mapsto \frac{1}{\mathrm{ch}^2(x)}$.
5. Vérifier que f est positive sur I.
6. Montrer que f est décroissante sur I.
7. Prouver que f est de classe C^1 sur I et préciser l'expression de $f'(x)$. Retrouver alors le résultat de la question précédente.
8. Soit $x \in I$. Démontrer la relation
$$f(x+2) = \frac{x}{x+1} f(x)$$
On pourra effectuer, en la justifiant, une intégration par parties.
9. Soit $p \in \mathbb{N}^*$. Donner l'expression de $f(2p)$ à l'aide de factorielles.

10. Pour tout réel $x > 0$, on pose
$$\phi(x) = xf(x)f(x+1)$$
Prouver que $\phi(x+1) = \phi(x)$. Calculer $\phi(n)$ pour tout $n \in \mathbb{N}^*$.
11. En utilisant la question précédente, déterminer un équivalent de $f(x)$ quand $x \to 0^+$.
12. Vérifier que $\forall n \in \mathbb{N}^*,\ f(n)f(n+1) = \frac{\pi}{2n}$. En déduire que
$$f(n) \underset{\substack{n\to+\infty \\ n\in\mathbb{N}^*}}{\sim} \sqrt{\frac{\pi}{2n}}$$
13. En utilisant des parties entières, prouver que
$$f(x) \underset{x\to+\infty}{\sim} \sqrt{\frac{\pi}{2x}}$$
14. Déduire des questions précédentes le tableau des variations de f sur I et tracer sa courbe représentative dans un repère orthonormé.
15. Prouver que la fonction ϕ est constante sur $\mathbb{R}^{+*}$.

Exercice 4

Dans tout l'exercice, pour tout entier naturel k, on identifie polynôme de $\mathbb{R}_k[X]$ et fonction polynomiale associée pour la structure d'espace vectoriel normé.

1. Soit P un élément de $\mathbb{R}[X]$ unitaire (le terme de plus haut degré de P est égal à 1).
   (a) Soit $\alpha \in \mathbb{R}$. Montrer que $\forall z \in \mathbb{C},\ |z - \alpha| \geq |\mathrm{Im}(z)|$.
   (b) On suppose dans cette question que P est scindé sur $\mathbb{R}$. En utilisant une factorisation de P, montrer que
   $$\forall z \in \mathbb{C},\ |P(z)| \geq |\mathrm{Im}(z)|^{\deg(P)}$$
   où $\deg(P)$ désigne le degré du polynôme P.
   (c) On prend dans cette question $P(X) = X^3 + 1$.
   (a) Donner une factorisation de P dans $\mathbb{C}[X]$.
   (b) Trouver $z_0 \in \mathbb{C}$ tel que $|P(z_0)| < |\mathrm{Im}(z_0)|^{\deg(P)}$.
   (d) On suppose dans cette question que $\forall z \in \mathbb{C},\ |P(z)| \geq |\mathrm{Im}(z)|^{\deg(P)}$. Montrer que toutes les racines de P sont réelles. En déduire que P est scindé sur $\mathbb{R}$.
   (e) Enoncer clairement le résultat obtenu.
2. Soient q un entier naturel non nul et $(A_n)_{n\in\mathbb{N}}$ une suite de matrices trigonalisables de $\mathcal{M}_q(\mathbb{R})$ qui converge vers une matrice A. On appelle pour tout entier naturel n, P_n le polynôme caractéristique de A_n et P celui de la matrice A.
   (a) Donner le degré et le coefficient dominant de P_n.
   (b) Prouver que $\forall x \in \mathbb{R}$, $\lim_{n\to+\infty} P_n(x) = P(x)$.
   (c) En déduire que A est trigonalisable.
   (d) Qu'en conclut-on pour l'ensemble des matrices trigonalisables de $\mathcal{M}_q(\mathbb{R})$?
3. On prend dans cette question $q = 2$ et $A_n = \begin{pmatrix} 1 - \frac{1}{n} & 1 - \frac{\sin(n)}{n} \\ 0 & 1 + \frac{1}{n} \end{pmatrix}$ où n est un entier non nul.
   (a) Déterminer $A = \lim_{n\to+\infty} A_n$.
   (b) Etudier la diagonalisabilité des matrices A_n et A dans $\mathcal{M}_2(\mathbb{R})$.
   (c) Conclure.