ÉPREUVE COMMUNE DE TIPE 2011 - Partie D

TITRE :

PARCOURS DU CAVALIER SUR L'ÉCHIQUIER

Temps de préparation : ……2 h 15 minutes
Temps de présentation devant les examinateurs : …….10 minutes
Dialogue avec les examinateurs : ……..10 minutes

GUIDE POUR LE CANDIDAT :

Le dossier ci-joint comporte au total : 12 pages
Document principal (12 pages, dont celle-ci) ; annexe : 0 page

Travail **suggéré** au candidat :

- Le candidat peut, au choix, proposer une synthèse du texte, ou mettre en avant un algorithme, une démonstration. Dans tous les cas il veillera à bien comprendre ce qu'il expose et cherchera à apporter un éclairage personnel sur le texte.

- Les algorithmes pourront être écrits en Caml ou en Pascal et devront être complétés en fonction des différentes orientations qui sont données ou en fonction d'idées propres.

Attention : s*i le candidat préfère effectuer un autre travail sur le dossier, il lui est* ***expressément recommandé*** *d'en informer le jury avant de commencer l'exposé.*

CONSEILS GENERAUX POUR LA PREPARATION DE L'EPREUVE :

* Lisez le dossier en entier dans un temps raisonnable.
* Réservez du temps pour préparer l'exposé devant les examinateurs.

- Vous pouvez écrire sur le présent dossier, le surligner, le découper … mais tout sera à remettre aux examinateurs en fin d'oral.
- En fin de préparation, rassemblez et ordonnez soigneusement TOUS les documents (transparents, *etc.*) dont vous comptez vous servir pendant l'oral, ainsi que le dossier, les transparents et les brouillons utilisés pendant la préparation. En entrant dans la salle d'oral, vous devez être prêt à débuter votre exposé.
- A la fin de l'oral, vous devez remettre aux examinateurs le présent dossier. Tout ce que vous aurez présenté aux examinateurs pourra être retenu en vue de sa destruction.

Parcours du cavalier sur l'échiquier

1. Introduction

Nous allons nous intéresser au déplacement du Cavalier sur l'échiquier. L'objectif est de passer une fois et une seule sur toutes les cases de l'échiquier. On numérotera chacune des cases traitées : on commencera par le nombre entier 1 pour terminer par le nombre correspondant à la taille maximale de l'échiquier.

Ce problème du cavalier, appelé également problème du Cavalier d'*Euler*, du nom du mathématicien suisse *Leonhard Euler* (1707-1783), est un célèbre casse-tête de logique dont les premières solutions connues sont présentées par deux Indiens *Ratnakara et Rudrata* dans un ouvrage écrit en 850. C'est à cette même époque que *al-Adli ar-Rumi*, très grand joueur d'échecs arabe, en donne également une solution.

Au début du 17^{ième} siècle, *Pierre Rémond de Montmort* pose les bases d'une réflexion mathématique reprise et développée par *Euler*. En 1759 il en propose une réelle version scientifique.

Pour traiter ce problème il n'est nul besoin de savoir jouer aux échecs. Rappelons brièvement le fonctionnement du Cavalier qui a un déplacement particulier en forme de *L*. Ce *L* peut être un *L* renversé verticalement ou un *L* renversé horizontalement voire un *L* renversé verticalement et horizontalement.

Illustrons simplement les différentes règles du déplacement du Cavalier sur l'échiquier. Si ce Cavalier se trouve en ligne i et en colonne j, il y aura au maximum 8 destinations possibles qui seront :

- ligne i+2, colonne j+1
- ligne i+1, colonne j+2
- ligne i–1, colonne j+2
- ligne i–2, colonne j+1
- ligne i–2, colonne j–1
- ligne i–1, colonne j–2
- ligne i+1, colonne j–2
- ligne i+2, colonne j–1

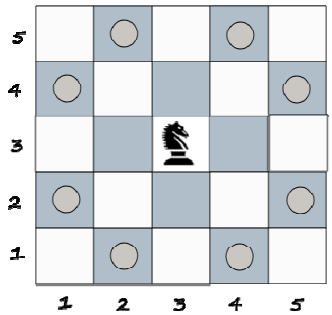


Figure 1

Si on utilise un échiquier d'ordre 5×5 (5 lignes, 5 colonnes), la case comportant un cavalier désigne la position du Cavalier à l'instant t et les ronds les cases possibles pour le Cavalier à l'instant t+1 (voir *figure 1*). Une seule case devra être choisie.

Dans tout ce document nous utiliserons des échiquiers d'ordre quelconque. Le nombre de lignes et le nombre de colonnes ne sont pas obligatoirement identiques.

Nous analyserons deux types de parcours du déplacement du Cavalier que l'on appellera des « rondes ». Une ronde consiste pour un Cavalier à parcourir toutes les cases de l'échiquier sans passer deux fois par la même case. On étudiera des rondes cycliques et des rondes non cycliques.

Par définition une ronde cyclique est une ronde où la dernière position du Cavalier sur l'échiquier permet de repartir vers la case initiale. Par exemple, pour un échiquier d'ordre 8×8, le problème d'Euler consiste à placer le Cavalier sur une case quelconque et à lui faire parcourir les 63 autres cases en exactement 63 sauts consécutifs. Cette ronde du Cavalier d'Euler sera dite cyclique si le Cavalier peut aller de la case finale à la case initiale en un seul coup.

On peut se poser une première question : de telles rondes sont-elles possibles sur un échiquier d'ordre 3×3, 4×4, 5×5, 4×5, 4×6, 7×5, … ?

En fonction de la nature de l'échiquier on pourra dénombrer plusieurs milliers voire plusieurs milliards de solutions. Par exemple pour un échiquier d'ordre 8×8 il y a plus de 120 000 000 000 solutions de parcours cycliques. Par contre, dans certaines conditions il n'y en aura aucune. Le problème du cavalier est beaucoup plus complexe que celui du fou ou celui des reines. Il demande un temps considérable pour être résolu du fait qu'il engendre un très grand nombre de combinaisons possibles.

Attention ! Nous nous intéresserons uniquement aux règles du déplacement du Cavalier telles que nous les connaissons habituellement au jeu d'échec. Mais, il existe de nombreuses variantes du déplacement du Cavalier qui ne font qu'augmenter le nombre de solutions.

Nous allons regarder, à présent, quelques méthodes afin de traiter ce problème de ronde du Cavalier. Les programmes et les algorithmes présentés seront uniquement sous forme de pseudo-code.

2. Résultats théoriques

a) Cas des échiquiers d'ordre M×N avec comme produit un résultat impair

Nous pouvons remarquer qu'un échiquier d'ordre 3×3 n'admet aucune solution, la case centrale du damier ne pouvant jamais être atteinte.

Nous nous intéressons dans ce paragraphe uniquement aux rondes cycliques d'échiquiers d'ordre M×N ayant comme produit N×M un nombre impair. Nous allons montrer qu'il n'existe aucune solution dans ce cas.

La démonstration prendra en compte la notion d'invariant.

Définition :

> *Un invariant est une propriété qui se conserve au cours et après une opération de nature physique, chimique ou mathématique.*

Dans notre problème de parcours du Cavalier on considérera qu'un invariant est une propriété qui ne dépend pas du parcours considéré.

Le raisonnement s'effectuera sur les couleurs de cases (blanches ou noires) de l'échiquier. On peut remarquer que la couleur de la case où se trouve le Cavalier après le coup numéro p n'est fonction que de la parité du nombre p. Il est clair que cette couleur change à chaque saut du Cavalier. Comme la solution doit être cyclique, la dernière case atteinte par le cavalier devrait avoir une couleur différente de celle de départ. Or comme le produit de N par M est impair, la dernière case atteinte a donc la même couleur que celle de la première case. C'est une contradiction et l'invariant n'est donc pas respecté.

Par contre de tels échiquiers peuvent avoir une ou plusieurs solutions avec des parcours non cycliques. La *figure 2* montre une solution pour un échiquier 5×5.

5	7	12	17	22	5
4	18	23	6	11	16
3	13	8	25	4	21
2	24	19	2	15	10
1	1	14	9	20	3
	1	2	3	4	5

Figure 2

b) Cas des échiquiers d'ordre N×M sans parcours cyclique

Tous les échiquiers offrant des solutions de parcours non cycliques ne possèdent pas forcément, en leur sein, de parcours cycliques.

Le théorème de *Allen Schwenk* permet de montrer qu'un parcours cyclique du Cavalier ne peut être réalisé que sur un échiquier d'au moins six cases de côté.

Théorème de Schwenk

Pour tout échiquier M × N tel que M soit inférieur ou égal à N, il existe un parcours cyclique du cavalier, à moins qu'une des conditions suivantes (ou plus) ne soit vraie :

- *M et N sont impairs et M×N différent de 1 ;*

- *M ∈{1, 2, 4} et M×N différent de 1 ;*
- *M = 3 et N ∈{4, 6, 8}.*

Démonstration

Condition 1 *(M et N sont impairs; M et N différents de 1)*

La première condition a été démontrée au paragraphe précédent.

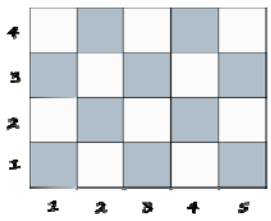
Condition 2 *(M ∈{1, 2, 4}; M et N différents de 1)*

Pour le cas M=1 et M=2 il est facile de constater que chaque case de l'échiquier ne pourra être atteinte excepté le cas où M et N valent 1. Un échiquier d'ordre 1×1 n'offrant que peu d'intérêt.

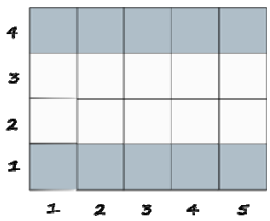
Le cas M=4 est moins facile à démontrer. On va commencer par définir deux ensembles pour un échiquier donné. Le premier ensemble nommé N_1 sera constitué uniquement avec les cases noires et le second nommé B_1 avec les cases blanches. Quelle que soit la valeur de N, le produit 4×N sera toujours pair. Les deux ensembles N_1 et B_1 auront le même nombre d'éléments.

Prenons un second échiquier mais avec une coloration complétement différente. Les cases noires seront définies uniquement pour les lignes 1 et 4 et les cases blanches pour les lignes 2 et 3. Nous définirons également deux ensembles, l'ensemble N_2 pour les cases noires et l'ensemble B_2 pour les cases blanches.

Par exemple, si on prend un échiquier d'ordre 4×5 voici les deux représentations possibles :



Echiquier X



Echiquier Y

Figure 3

Supposons qu'il existe un parcours du Cavalier cyclique qui respecte l'alternance des couleurs de l'échiquier pour chaque déplacement du Cavalier. Et ceci quel que soit l'échiquier choisi (*échiquier X* ou *échiquier Y*).

Il est nécessaire qu'il y ait l'alternance des couleurs sur l'échiquier *Y*. Il est évident que le cavalier passe obligatoirement d'une case de N_2 à une case de B_2 et que, d'autre part, comme il doit visiter chaque case, il doit aussi passer d'une case de B_2 à une case de N_2 car sinon il devrait ensuite parcourir deux cases consécutives ou plus de N_2, ce qui est impossible.

Soit la case (1,1) la case de départ. Cette case appartient à la fois aux ensembles N_1 et N_2. Pour respecter l'alternance de couleur dans le déplacement, la case suivante choisie appartiendra nécessairement à B_1 et à B_2. Puis de nouveau à N_1 et N_2 et ainsi de suite. On constate rapidement que les ensembles N_1 et N_2 ont les mêmes éléments comme par ailleurs les ensembles B_1 et B_2. Mais par construction les ensembles N_1 et N_2 ne sont pas égaux, ils n'ont qu'une partie de leurs éléments en commun. Le raisonnement est identique pour les ensembles B_1 et B_2. Nous arrivons donc à une contradiction.

Par conséquent dans le cas d'échiquier d'ordre 4×N il ne peut y avoir de parcours cyclique de Cavalier.

Condition 3 *(M = 3 et N ∈{4, 6, 8})*

On peut vérifier la condition 3 en déterminant tous les parcours et en observant qu'aucun d'entre eux n'est cyclique, en commençant avec N = 4 puis N = 6 voire N = 8.

On peut remarquer dans le cas d'un échiquier d'ordre 3×4 qu'il existe bien des parcours non cycliques. En voici deux (*figure 4*) :

3	10	7	2	5
2	1	4	11	8
1	12	9	6	3
	1	2	3	4

3	8	11	6	3
2	1	4	9	12
1	10	7	2	5
	1	2	3	4

Figure 4

3. Heuristique de Warnsdorff

Une heuristique est un algorithme qui fournit rapidement (généralement en un temps polynomial) une solution pour un problème d'optimisation. Le plus souvent, cette solution n'est pas optimale. L'heuristique proposée en 1823 par le mathématicien allemand *Warnsdorff* permet de trouver une solution au problème du cavalier d'Euler en un temps linéaire. Cependant, il est possible dans certains cas que cette heuristique n'affiche aucun parcours alors que l'on peut montrer à l'aide d'autres méthodes de résolution qu'il en existe.

Le principe de cette heuristique est le suivant :

- à chaque case *c* de l'échiquier on affecte un poids égal au nombre de cases accessibles à partir de *c* ;
- lorsque l'on parcourt l'échiquier, on choisit les cases de poids minimal.

L'idée est donc de parcourir l'échiquier en partant de la périphérie et en se rapprochant du centre.

L'algorithme 1, ci-après, utilise deux matrices ayant la taille de l'échiquier :

- la matrice `parcours`, telle que `parcours[i,j]` vaut 0 si la case (*i*, *j*) n'a pas été atteinte par le Cavalier et vaut *n* si celui-ci est passé sur la case (*i*, *j*) à la $n^{ème}$ étape de son parcours ;
- la matrice `poids`, qui contient les poids de Warnsdorff.

Une procédure principale permet de choisir la position de départ, (*x*, *y*), d'initialiser les deux matrices `parcours` et `poids`, et d'appeler la procédure `parcourir` qui applique l'heuristique de manière itérative jusqu'à ce que l'échiquier soit complètement parcouru ou bien que l'heuristique (fonction `Warnsdorff`) n'arrive pas à trouver de solution :

```
Constantes
  M =   //dimension du premier axe de l'échiquier
  N =   //dimension du second axe de l'échiquier

Type
  MAT : matrice [M] [N] d'entiers ;
  DELTA : matrice [8] d'entiers ;
  POSITION : structure {ligne : entier ; colonne : entier} ;

Variables globales
  parcours : MAT ; // Matrice contenant l'indice du saut pour chaque case
                   // parcourue et 0 sinon
  poids    : MAT ; // Matrice contenant les poids de Warnsdorff

procedure principale (x : entier ; y : entier)
debut
   initialiserParcours ; // La matrice Parcours est initialisée avec des 0
   initialiserPoids ;
   parcours[x,y] = 1 ;
   parcourir(x,y)
fin procedure

procedure parcourir (x : entier ; y : entier)
   p : POSITION ;
   i : entier
debut
   i = 2 ;
   p.ligne = x ;
   p.colonne = y ;
   tant que (i <= M*N) faire
      p = Warnsdorff(p) ;
      si (p.ligne >= 0) alors
         parcours[p.ligne, p.colonne] = i ;
         i = i + 1
      sinon
         i = M*N + 1
      fin si
   fin tant que
   si (p.ligne >= 0) alors
      afficher(parcours)
   sinon
      écrire('Pas de solution trouvée.')
   fin si
fin procedure
```

```
fonction Warnsdorff (p : POSITION) : POSITION
   deltaLigne   : DELTA = [2, 1, -1, -2, -2, -1,  1,  2] ;
   deltaColonne : DELTA = [1, 2,  2,  1, -1, -2, -2, -1] ;
   pMin : POSITION ;
   i, iMin, poidsMin : entier ;
debut
   poidsMin = 9 ;
   de i = 0 à 7 faire
      pMin.ligne   = p.ligne   + deltaLigne[i] ;
      pMin.colonne = p.colonne + deltaColonne[i] ;
      si ((pMin.ligne >= 0) et (pMin.ligne < M) et (pMin.colonne >= 0)
           et (pMin.colonne < N) et (parcours[pMin.ligne, pMin.colonne] = 0
           et (poids[pMin.ligne, pMin.colonne] < poidsMin) )
      alors
         poidsMin = poids[pMin.ligne, pMin.colonne] ;
         iMin = i
      fin si
   fin faire
   si (poidsMin = 9) alors pMin.ligne = -1 ;
   retourner(pMin)
fin fonction
```

Algorithme 1

4. Algorithme avec *backtracking* pour des parcours du Cavalier

a) Introduction

Ce type d'algorithmes est très intéressant et permet de trouver, le plus souvent, toutes les solutions, notamment les milliards de possibilités pour un échiquier 8×8.

Le retour sur trace, appelé aussi «marche arrière» a pour origine le terme anglais *backtracking* inventé par le mathématicien américain *D.H. Lehmer* dans les années 1950. Le *backtracking* consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage.

L'idée est la suivante : à partir d'une case donnée on va choisir une case libre (en respectant les règles du déplacement du Cavalier). On va utiliser ce principe jusqu'à trouver si possible une solution, mais le plus souvent nous allons arriver sur une position de blocage : plus de case libre. On reviendra au choix précédent afin de choisir une autre case disponible. Si cette situation ne donne toujours pas satisfaction on recommencera à choisir une autre case, et ainsi de suite.

L'utilisation la plus courante pour le traitement des algorithmes de recherche par *backtracking* s'effectue principalement en utilisant la programmation récursive. Le *backtracking* est intégré par défaut au mécanisme de récursion.

Nous allons nous servir de cette technique de programmation afin de trouver l'ensemble des solutions pour notre ronde du Cavalier.

b) Approche fonctionnelle

Il est possible d'utiliser une matrice pour représenter l'échiquier mais la structure de donnée la plus adaptée est une liste de couples, chaque couple (*i j*) représentant la case qui se trouve à l'intersection de la ligne *i* et de la colonne *j*. Les fonctions suivantes pourront utiliser l'une ou l'autre de ces structures de données pour représenter l'échiquier.

On admet que l'on dispose d'une fonction *cases_accessibles* ayant comme paramètres un couple, pos, représentant une position, et une liste de couples, *Ech*, représentant un échiquier, et qui retourne la liste des cases accessibles à partir de la position pos sur l'échiquier *Ech*. La fonction *nb_parcours* a elle aussi comme argument un couple *position* et une liste de couples *Echiquier* représentant respectivement la position de départ du Cavalier et l'échiquier sur lequel on souhaite connaître le nombre de "rondes". Cette fonction est récursive, et le principe de son héritage est le suivant : la fonction *cases_accessibles* donne la liste *CA* des cases accessibles à partir de *position*, et, pour chaque élément *c* de *CA*, la fonction *nb_parcours* est appelée récursivement avec comme arguments c et un nouvel échiquier dans lequel on a enlevé la case *position*, ce qui élimine toute possibilité de revenir sur une case déjà parcourue. Il ne reste plus qu'à faire la somme des nombres de rondes trouvées par chaque appel récursif. Le cas d'initialisation est alors trivial : c'est le cas où l'échiquier n'a plus qu'une seule case (la case *position* en l'occurrence) et la réponse retournée est évidemment 1. Finalement, dans une écriture qui reste impérative, la fonction *nb_parcours* peut s'écrire ainsi :

```
nb_parcours (position, Echiquier)
   si Echiquier n'a qu'une seule position
      retourner 1
   sinon
   début
      CA = cases_accessibles (position, Echiquier) ;
      nouvel_echiquier = supprimer (position, Echiquier) ;
      compteur = 0 ;
      Pour chaque élément c de CA, répéter
        compteur = compteur + nb_parcours (c, nouvel_echiquier) ;
      retourner compteur
   fin
fin fonction
```

Algorithme 2

Cette fonction peut facilement être adaptée pour donner la liste de toutes les rondes. Les deux choses auxquelles il faut penser sont :

- pour l'héritage, il ne faut pas oublier de mettre la position de départ en tête de toutes les rondes obtenues par appels récursifs ;
- pour l'initialisation, il ne faut pas commettre d'erreur de typage : la fonction doit retourner une liste de rondes, donc une liste de listes de positions.

Cela donne la fonction suivante :

```
parcours (position, Echiquier)
   Si Echiquier n'a qu'une seule position
      retourner la liste qui contient la liste qui contient position
   sinon
   début
      CA = cases_accessibles(position, Echiquier) ;
      Nouvel_echiquier = supprimer (position, Echiquier) ;
      Liste_de_rondes = () ;
      Pour chaque élément c de CA, répéter
         Liste_de_rondes = append (Liste_de_rondes,
                                   parcours (c, Nouvel_echiquier) ) ;
      Pour chaque élément Lr de Liste_de_rondes, répéter
         Mettre position en tête de Lr ;
      Retourner Liste_de_rondes
   Fin
fin fonction
```

Algorithme 3

où la fonction *append*, appelée avec deux listes `L1` et `L2` retourne la liste obtenue en regroupant les contenus de `L1` et de `L2`

c) Approche impérative

i. Chemin Hamiltonien : définitions de base

Soit X un ensemble quelconque et soit $X^{(2)}$ l'ensemble des ensembles à deux éléments $\{ x_1, x_2 \}$ avec $x_1, x_2 \in X$. Un graphe non orienté sur X est un couple $G = (X, V)$ où $V \subseteq X^{(2)}$. X est l'ensemble des sommets de G et V celui de ses arêtes. On écrira x_1x_2 l'arête $\{x_1, x_2\} \in V$, x_1 et x_2 étant ses extrémités.

Dans un graphe non orienté, une chaîne est une suite d'arêtes $(x_0x_1, x_1x_2, \dots, x_{n-1}x_n)$.

On appelle chemin hamiltonien une chaîne qui passe une fois et une seule par tous les sommets d'un graphe.

Un graphe qui admet un chemin hamiltonien est appelé graphe semi-hamiltonien.

Le chemin $x_1x_2 \dots x_nx_1$ est un cycle hamiltonien si $x_1x_2 \dots x_n$ est un chemin hamiltonien.

Un graphe qui admet un cycle hamiltonien est appelé hamiltonien.

ii. Échiquier vers graphe

Pour un échiquier carré de n cases de côté, on associera un sommet du graphe à chaque case de l'échiquier : le nombre de sommets du graphe sera donc n^2.

Comment seront alors reliés les sommets entre eux ?

Si une case de l'échiquier est identifiée par ses coordonnées *i* et *j*, un sommet (*i*, *j*) du graphe sera relié par une arête aux 8 sommets (*i*–1, *j*–2), (*i*+1, *j*–2), (*i*–2, *j*–1), (*i*+2, *j*–1), (*i*–1, *j*+2), (*i*+1, *j*+2), (*i*–2, *j*+1), (*i*+2, *j*+1). Cela signifie que, d'un sommet du graphe, il partira ou arrivera au maximum huit arêtes. Si on choisit la case se trouvant dans un des quatre coins d'un échiquier, il n'y a que deux arêtes reliant ce sommet à deux autres.

Construisons à partir de la case (1,1) une petite partie du graphe avec l'échiquier suivant :

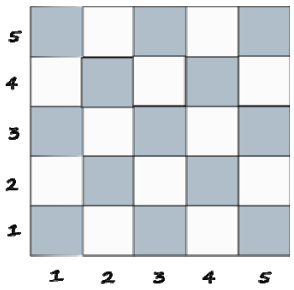


Figure 5

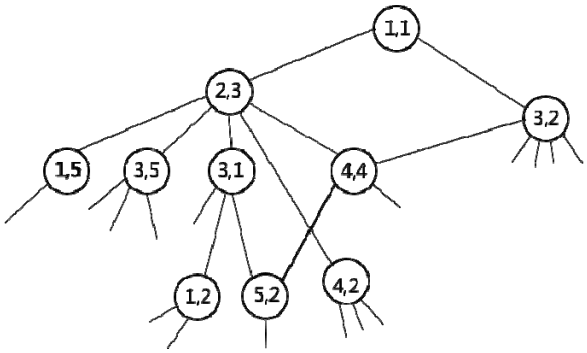


Figure 6

On constate que la *figure 6* n'est pas un arbre, mais un graphe. Il a été élaboré partiellement à partir de la *figure 5*. On peut remarquer l'existence, sur cette partie du graphe, de cycles entre sommets (par exemple le chemin (1,1), (2,3), (4,4), (3,2), (1,1))

Chercher un parcours du cavalier qui passe une fois et une seule par toutes les cases de l'échiquier, correspond à la recherche d'un chemin hamiltonien sur le graphe.

iii. Survol de l'algorithme

Le pseudo-code de cet algorithme est donné en *algorithme* 4. Les paramètres de l'algorithme sont :

- une matrice `ronde` dans laquelle on indiquera la valeur des cases parcourues ;
- une matrice `position_cavalier` indiquant les huit déplacements possibles du cavalier ;
- deux constantes M et N définissant les dimensions de l'échiquier ;
- une variable `nb_solutions` précisant le nombre de solutions trouvées.

La procédure récursive `parcourir()` permettra de réaliser la construction du chemin hamiltonien s'il existe. Nous pouvons remarquer que ce code ne donne qu'une seule solution et il serait assez simple de le modifier afin d'obtenir toutes les solutions.

iv. Algorithme

```
Constantes
  M=8  // la dimension du premier axe de l'échiquier
  N=8  // la dimension du second axe de l'échiquier

Variables globales
  ronde : matrice[M][N] de nombres entiers
  position_cavalier : matrice[8][2] de nombres entiers
  nb_solutions : entier = 0

procedure parcourir(n:entier, x:entier, y:entier)    // position suivante
k,X,Y:entier
  ronde [x][y] = n;
  Si (n < NM) alors
    pour k = 0 à 7 faire
      X = x + position_cavalier[k][0]
      Y = y + position_cavalier[k][1];
      si (X >= 0 et X < M et Y >= 0 et Y < N et ronde [X][Y] == 0)
           parcourir(n + 1, X, Y)
      fin si
    Fin du pour
  sinon si (n = NM) alors
            // affichage de la solution
         nb_solutions = nb_solutions +1;
         si (nb_solutions <= 1) affiche_solution() fin si
        fin si
  Fin si
  ronde[x][y] = 0;
fin procedure

procedure principale()
  i,j:entier;
  NM:entier = N*M
  position_cavalier={{2,1},{1,2},{-1,2},{-2,1},{-2,-1},{-1,-2},{1,-2},
                     {2,-1}}
  Pour i = 0 à M faire
    Pour j = 0 à N faire
     ronde[i][j] = 0
    Fin du pour
  Fin du pour
  parcourir(1, 0, 0);
fin procedure
```

Algorithme 4

d) Ronde cyclique du Cavalier

Il est relativement facile de chercher le nombre de rondes cycliques du Cavalier, voire de les afficher. Pour mettre en œuvre cette fonctionnalité sur les algorithmes précédents il suffit de mémoriser la case initiale du parcours et de tester si elle est accessible en un seul saut à partir de la dernière case de ce même parcours.