TRAVAUX PRATIQUES XXV

Retour sur trace

A Autour du problème des n dames

A.1 Applications directes du cours

Un fichier `dames.ml` est fourni pour ce TP. Il contient grosso modo fonction `solution_n_dames` vue en cours (légèrement modifiée). Il s'agit de la version où l'on manipule des listes. Ici, on cherche à construire la liste des coordonnées (ligne,colonne) de placement des dames (alors qu'en cours on ne stockait qu'une seule des deux coordonnées, l'autre était implicite).

1. Prenez le temps de bien lire `soluble_n_dames` et vérifiez que vous la comprenez bien.
2. Étant donnés (i, j) et (k, l) deux coordonnées de cases, comment tester très rapidement s'ils sont sur une même ligne ? colonne ? diagonale ?
   (Pour la diagonale, cherchez une quantité simple qui est invariante le long d'une diagonale descendante, et une autre invariante le long d'une ascendante.)
   Écrire une fonction `en_prise (i,j) (k,l)` qui teste si deux coordonnées de case correspondent à des dames en prise.
3. En déduire une fonction `conflit_derniere : (int*int) list -> bool` qui prend en argument une listes de coordonnées (qui correspondent aux dames déjà placées, de la dernière à la première) et qui teste si la tête de cette liste est en prise avec l'une des coordonnées de la queue de la liste.
   Autrement dit, elle teste si la tête de la liste a créé une prise.
4. Vérifiez que `soluble_n_dames` fonctionne.

Ce programme teste l'existence d'une solution. En général, on peut vouloir résoudre 4 variantes :

- Tester s'il existe au moins une solution.
- Exhiber une solution (quelconque) s'il en existe une (et signaler qu'il n'en existe pas dans le cas contraire).
- Compter le nombre de solutions.
- Déterminer la liste de toutes solutions.

5. Écrire une fonction `solution_n_dames : int -> (int*int) list` qui étant donné un entier n, renvoie une solution du problème des n dames.
6. **a.** Existe-t-il une solution pour le problème des n dames pour $n = 2$? Pour $n = 3$? Pour $n = 5$? Afficher la disposition des dames sur l'échiquier, s'il en existe une (une fonction d'affichage est fournie).
   b. Et pour $n = 20$? (Et pour $n = 30$? Et pour $n = 100$?) Que remarquez-vous ?

A.2 Variantes du problèmes des n dames

7. Écrire une fonction `compte_n_dames : int -> int` qui étant donné un entier n, calcule le nombre de solutions au problème des n dames.
   Indication : On ne veut plus utiliser d'exception mais continuer jusqu'au bout. Quand vous trouvez une solution, augmentez un « compteur de solutions ».
8. Écrire une fonction `toutes_solutions_n_dames : int -> (int*int) list list` qui étant donné un entier n renvoie la liste de toutes les solutions au problème des n dames.
   Remarque : on peut passer par une référence de listes, ou Faire sans en codant astucieusement. Pour les plus à l'aise, essayez donc la deuxième option !

B Cavalier d'Euler

Cette section se traite en C.

Définition du problème

Aux échecs, un cavalier est une pièce qui peut faire des déplacements "en L", de deux cases selon un axe et une case selon l'autre (donc deux cases verticalement puis une case horizontalement, ou deux cases horizontalement puis une case verticalement).

Donnons un exemple des cases atteignables par un cavalier sur un échiquier standard (de taille 8×8) :

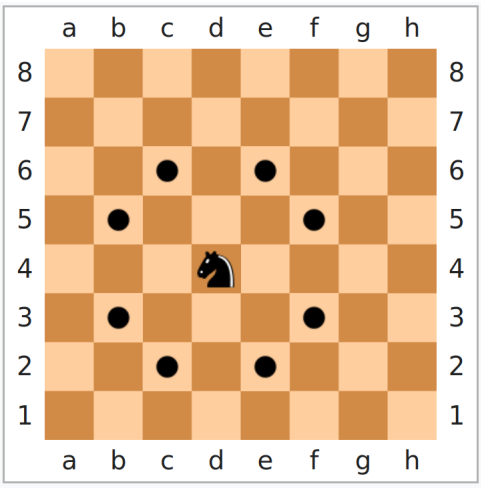

FIGURE XXV.1 – Les huit déplacements possibles pour un cavalier.

Bien sûr, si le cavalier est dans un coin ou sur un bord, son nombre de déplacements possible est limité par les bords de l'échiquier (il ne peut pas en sortir).

Définition 1

Problème du cavalier :
Étant donné un échiquier de taille $n \times m$ avec $n, m \in \mathbb{N}^*$ et un cavalier posé sur une case quelconque (i_0, j_0) de l'échiquier, faire parcourir au cavalier toutes les cases de l'échiquier de sorte qu'il ne repasse jamais deux fois par la même case.

Certaines variantes imposent que le cavalier finisse en plus sur sa case de départ, mais nous ne nous intéresserons pas à cette contrainte ici.

Par exemple, si on pose un cavalier sur la case (1,1) d'un échiquier de taille 5×5 (en numérotant les cases comme dans une matrice et à partir de 0), une solution possible du problème du cavalier est la suivante :

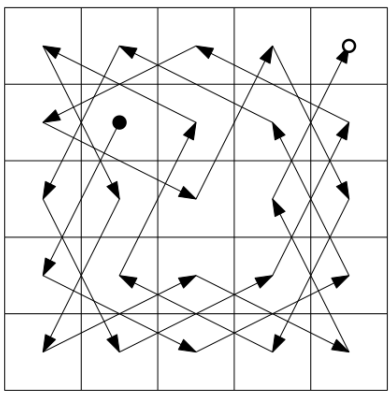

FIGURE XXV.2 – Un parcours du cavalier sur un échiquier 5×5.

Représentation informatique

Une solution au problème du cavalier sera représentée par une matrice de taille $n \times m$ telle que :

— La case de coordonnées (i_0, j_0) contient 0.

— Une case de coordonnées quelconque (i, j) contient un entier k si (i, j) est la k-ième case sur laquelle le cavalier est allé (sa case initiale étant la 0-ième).

Autrement dit, on représente les déplacements successifs du cavalier par une suite de cases $(c_0, c_1, ..., c_{n\times m-1})$. La case c_k de coordonnées (i_k, j_k) contient l'entier k, sachant que la case c_0 est la case initiale du cavalier.

Par exemple, une solution du parcours d'un cavalier partant de la case $(0, 0)$ sur un échiquier de taille 3×4 serait :

Terminal
```
0 3 6 9
7 10 1 4
2 5 8 11
```

9. Dessiner (sur une feuille et avec un stylo !) le parcours du cavalier correspondant à cette solution.

Résolution par retour sur trace

Si vous souhaitez vous entraîner à résoudre le problème en autonomie, arrêtez vous là dans la lecture de l'énoncé et répondez simplement à la question ouverte suivante :

10. Résoudre le problème du cavalier par la méthode du retour sur trace. Vérifier la complexité de votre solution avec la dernière question du sujet. Vous travaillerez en C.

Pour vous guider un peu :

Un plateau est une matrice, donc un "tableau" de "tableau" (la "tableau" des lignes). Ces "tableaux" seront en fait des pointeurs vers des zones allouées, donc un plateau est un `int**` : `plateau[i][j]` est l'entier correspondant à la case (i, j).

11. Implémenter une fonction `int** cree_plateau(int nb_lgn, int nb_col)` qui crée un plateau à `nb_lgn` lignes et `nb_col` colonnes.
Demandez-moi de l'aide si vous bloquez !!

12. Écrire une fonction `void free_plateau(int** plateau, int nb_lgn)` qui libère un plateau.

13. Implémenter une fonction `void affiche(int const** )` qui affiche un plateau.
(Bonus :) affichez les entiers de sorte à ce qu'ils soient alignés pour faciliter la lecture (on considérera des entiers entre 0 et 100 uniquement). Par exemple, la solution sur l'échiquier $(3, 4)$ précédent devrait s'afficher comme suit :

Terminal
```
0  3  6  9
7  10 1  4
2  5  8  11
```

Une manière d'implémenter le retour sur trace pour ce problème est de, tout au long de l'algorithme, faire en sorte d'une case du plateau contiennne (-1) si le cavalier n'est pas encore passé dessus et k s'il est passé dessus au k-ième coup.

Une solution partielle du problème contenant les k premiers mouvemments du cavalier possède $k+1$ cases remplies (la case initiale contenant 0 et toutes les cases visitées après chaque mouvement, numérotées dans l'ordre). Lorsqu'on revient en arrière, il **faudra** remettre la case à -1.

14. À la main, calculer les coordonnées des 8 cases que l'on peut théoriquement atteindre en 1 mouvement de cavalier depuis la case (i, j).

15. Écrire une fonction `bool disponible (int const** plateau, int nb_lgn, int nb_col, int x, int y)` qui renvoie `true` si et seulement si la case (x, y) du plateau existe et est disponible.

16. En déduire une fonction récursive `bool retour_trace(int** plateau, int lgn, int col, int i, int j)` qui prend en argument un plateau et ses dimensions (dans les cases du plateau sont marqués des (-1) ou des k comme détaillé précédemment) ainsi que la coordonnée (i, j) d'une case où le cavalier vient d'atterir (elle ne contient donc pas (-1)) et teste s'il est possible de résoudre le problème depuis cette situation.
On pourra faire un gros `if-then-else` au sein du code pour gérer les 8 déplacements possibles. On peut aussi faire plus propre en précalculant les vecteurs correspondant à ces 8 déplacements : le plus importants est d'avoir un code lisible (!!!!).

17. En déduire une fonction `bool cavalier(int nb_lgn, int nb_col, int i0, int j0)` qui prend en argument les dimensions d'un plateau et les coordonnées de la case initiale et renvoie `true` si et seulement si il existe un parcours du cavalier d'Euler depuis cette case initiale qui passe une et une seule fois par chaque case.

18. Jusqu'à quelle valeur de n la recherche d'un parcours sur un échiquier $n \times n$ (à partir de la case $(0, 0)$, disons) prend-elle un temps raisonnable ?
Pour peu que vous ne soyez pas dans un cas pathologique[1]*, le* 8×8 *depuis* $(0, 0)$ *devrait finir en 2-3 min.*

1. Cf TIPE de votre camarade de spé.