Partiel d'Algorithmique et Programmation

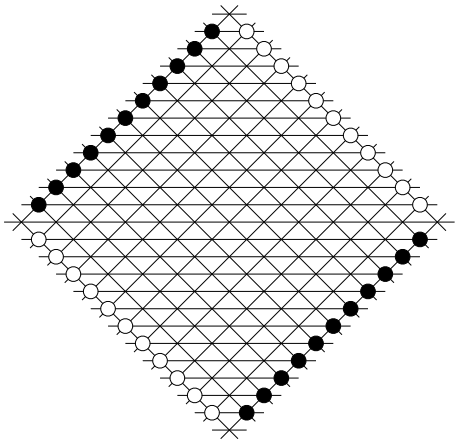
Math 312

29 octobre 2008

Durée du partiel : 3 heures
Tous les documents statiques sont autorisés.
Aucun outil dynamique (calculatrice, pda, téléphone, etc) n'est admis.

Le partiel consiste en un gros problème. Chaque question demande l'écriture d'une fonction ou procédure pour une action particulière. Vous êtes libres de les rédiger en langage algorithmique ou en C, essayez toutefois de ne pas trop mélanger les deux. Vous pouvez utiliser ensuite ces fonctions dans la suite du problème même si vous n'avez pas réussi à les écrire. Les questions dans chaque partie sont de difficulté croissante. En outre, les deux premières parties sont indépendantes.

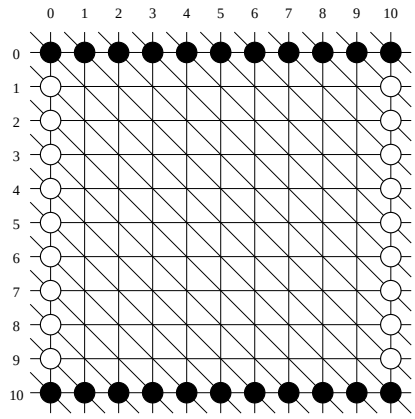
Le barème est donné à titre indicatif, il est susceptible de varier. La qualité de la présentation et des choix algorithmiques sera prise en compte dans l'évaluation.



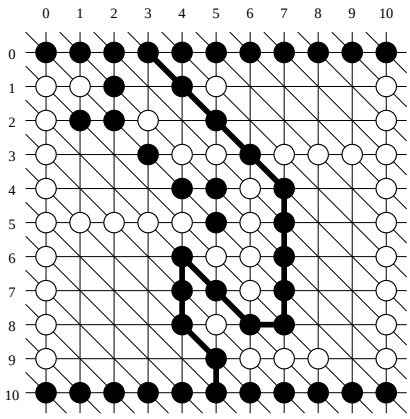
Problème : Jeu de Hex

Hex est un jeu de réflexion pour 2 joueurs qui se joue sur une grille triangulaire (formée de triangles bien orientés, voir figure). La règle du jeu est très simple :

Le joueur blanc pose des pions blancs, le joueur noir pose des pions noirs. Le joueur blanc commence. Chaque joueur pose à son tour un pion sur une intersection libre. Les pions une fois posés ne sont ni retirés ni déplacés. La grille a quatre bords, deux bords opposés sont pour le joueur blanc et les deux autres sont pour le joueur noir (voir la position de départ de la figure). Le joueur qui parvient à former un chemin continu qui rejoint ses deux bords a gagné la partie.



Position de départ



Le joueur noir gagne la partie

Dans l'exemple, le joueur noir a gagné. Le joueur blanc a perdu : l'existence d'un chemin gagnant pour une couleur empêche l'existence d'un chemin gagnant pour l'autre couleur.

Partie 1 : Modélisation du jeu (6,5pts)

Nous réalisons un programme qui permet à 2 joueurs de jouer au jeu de Hex. Dans cette partie, nous ne cherchons pas à savoir quel est le joueur gagnant. Ce sera l'objet de la partie suivante.

Chaque intersection de la grille est indicée par son indice de ligne puis de colonne (voir figure). Nous utilisons un tableau carré à deux dimensions pour modéliser la grille : une case du tableau est une intersection de la grille. Le jeu de Hex peut se jouer sur une grille carrée de taille quelconque (11 pour l'exemple). Nous désignons par la constante `T` la taille du tableau à deux dimensions `Grille`. En C, `T` et `Grille` seraient définis de la manière suivante :

```
#define T 11
int Grille[T][T];
```

Un pion blanc sera modélisé par l'entier `1` et un pion noir par l'entier `2`. L'absence de pion sera modélisé par l'entier `0`. Voici, par exemple, l'état du tableau `Grille` correspondant au deux situations précédentes (les indices de lignes et de colonnes ne font pas partie du tableau).

	0	1	2	3	4	5	6	7	8	9	10
0	2	2	2	2	2	2	2	2	2	2	2
1	1	0	0	0	0	0	0	0	0	0	1
2	1	0	0	0	0	0	0	0	0	0	1
3	1	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	0	0	0	1
5	1	0	0	0	0	0	0	0	0	0	1
6	1	0	0	0	0	0	0	0	0	0	1
7	1	0	0	0	0	0	0	0	0	0	1
8	1	0	0	0	0	0	0	0	0	0	1
9	1	0	0	0	0	0	0	0	0	0	1
10	2	2	2	2	2	2	2	2	2	2	2

Position de départ

	0	1	2	3	4	5	6	7	8	9	10
0	2	2	2	2	2	2	2	2	2	2	2
1	1	1	2	0	2	1	0	0	0	0	1
2	1	2	2	1	0	2	0	0	0	0	1
3	1	0	0	2	1	1	2	1	1	1	1
4	1	0	0	0	2	2	1	2	0	0	1
5	1	1	1	1	1	2	1	2	0	0	1
6	1	0	0	0	2	1	1	2	0	0	1
7	1	0	0	0	2	2	1	2	0	0	1
8	1	0	0	0	2	1	2	2	0	0	1
9	1	0	0	0	0	2	1	1	1	0	1
10	2	2	2	2	2	2	2	2	2	2	2

Le joueur noir gagne la partie

1) Initialiser (1pt) Écrire une procédure qui initialise la grille de jeu à la position de départ. Les en-têtes de cette procédure sont, selon qu'on adopte une description algorithmique ou l'écriture en C :

procédure Initialiser(Grille)

```
void initialiser (int G[T][T]);
```

2) Afficher (1pt) Pour afficher la grille de jeu, nous ne dessinons pas les lignes de la grille triangulaire mais juste des symboles représentant les pions. Un pion blanc est représenté par le caractère `'B'`, un pion noir par le caractère `'N'`, l'absence de pion par le caractère `'.'`. Écrire une procédure qui affiche la grille de jeu. Elle a pour en-tête :

procédure afficher(Grille)

```
void afficher (int G[T][T]);
```

Voici, par exemple, l'affichage correspondant aux deux situations précédentes.

```
N N N N N N N N N N N
B . . . . . . . . . B
B . . . . . . . . . B
B . . . . . . . . . B
B . . . . . . . . . B
B . . . . . . . . . B
B . . . . . . . . . B
B . . . . . . . . . B
B . . . . . . . . . B
B . . . . . . . . . B
N N N N N N N N N N N
```

Position de départ

```
N N N N N N N N N N N
B B N . N B . . . . B
B N N B . N . . . . B
B . . N B B N B B B B
B . . . N N B N . . B
B B B B B N B N . . B
B . . . N B B N . . B
B . . . N N B N . . B
B . . . N B N N . . B
B . . . . N B B B . B
N N N N N N N N N N N
```

Le joueur noir gagne la partie

3) Jouer un coup (2pts) Écrire une procédure qui demande à un joueur un numéro de ligne et un numéro de colonne correspondant à une case où il peut jouer. La procédure redemande un numéro de ligne et un numéro de colonne, tant que le joueur ne joue pas sur une case valide. Une case est valide si elle correspond à une case de la grille et si elle n'est pas occupée par un autre pion. L'en-tête de cette procédure peut être :

procédure jouercoup(i,j,Grille)

```
void jouercoup (int * i, int * j, int G[T][T]);
```

Les variables désignées par les deux premiers paramètres de la procédure contiennent à la fin de l'execution de la procédure le numéro de ligne et de colonne donnés par le joueur.

4) Enregistrer le coup (1pt) Écrire une procédure qui utilise la procédure `jouercoup` pour demander à un joueur un numéro de ligne et de colonne, et qui met un pion du joueur sur la grille. Cette procédure à un paramètre `joueur` qui vaut `1` ou `2` selon que c'est le joueur blanc ou noir qui joue. Cette procédure peut être utilisée pour faire jouer le joueur blanc ou le joueur noir, selon le paramètre `joueur`. L'en-tête de cette procédure peut être :

procédure enregistrercoup(joueur,Grille)

```
void enregistrercoup (int joueur, int G[T][T]);
```

5) Programme principal (1,5pts) Écrire le programme principal (en C `main`) qui permet de faire jouer deux joueur au jeu de Hex. Le programme doit initialiser la grille, l'afficher, puis faire jouer alternativement le joueur blanc et le joueur noir jusqu'à ce que toutes les cases de la grille soient remplies. Après chaque coup, la grille est affichée. On utilise les procédures définies aux questions précédentes. Il n'est pas demandé de déterminer si un joueur a gagné.

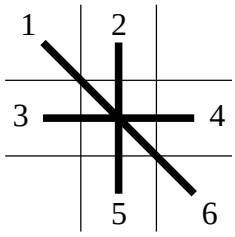
Partie 2 : Déterminer le gagnant (8pts)

Au jeu de Hex, il y a toujours un gagnant. Une grille qui ne contient aucune case vide possède toujours un chemin gagnant pour le joueur blanc ou le joueur noir, et il ne peut exister un chemin gagnant pour les deux joueurs.

Dans cette partie, nous allons déterminer, *pour une grille complètement remplie*, quel est le joueur qui a gagné. Si le joueur blanc gagne, le joueur noir ne peut pas gagner. Si le joueur blanc perd, c'est le joueur noir qui gagne. Il nous suffit donc de déterminer si le joueur blanc gagne ou non, pour connaître le vainqueur.

Pour determiner si le joueur blanc gagne, nous utilisons l'algorithme suivant. Nous remplaçons tous les `1` de la colonne `1` par des `-1`. Puis, nous propageons ces `-1` dans la grille de jeu sur les positions du joueur blanc : " un `1` devient `-1` s'il a un voisin `-1`". Si, une fois la propagation terminée, la colonne `T-2` contient un `-1`, le joueur blanc a gagné.

Nous remarquons qu'une case intérieure de la grille a exactement 6 voisines dans le jeu de Hex.



6) Coordonnées des cases voisines (1pt) Écrire une procédure qui affiche les indices des cases voisines d'une case intérieure donnée. L'en-tête de cette procédure peut être :

procédure casesvoisines(i,j)

```
void casesvoisines (int i, int j);
```

Par exemple, l'appel à `casesvoisines(3,5)` doit afficher

```
2,4  2,5  3,4  3,6  4,5  4,6
```

7) Initialisation de la propagation (0,5pts) Écrire une procédure qui remplace tous les `1` de la colonne `1` du tableau passé en paramètre par des `-1`. L'en-tête de cette procédure peut être :

procédure initialisationpropagation(Grille)

```
void initialisationpropagation (int G[T][T]);
```

8) Propagation partielle (2,5pts) Écrire une fonction qui propage les `-1` partiellement. Cette fonction parcourt toutes les cases intérieures de la grille, une seule fois, et transforme un `1` en `-1` si et seulement si ce `1` a un `-1` comme voisin. Cette fonction retourne un booléen vrai si au moins une case a été modifiée et faux sinon. L'en-tête de cette fonction peut être :

fonction propagationpartielle(Grille) → booléen

```
int propagationpartielle (int G[T][T]);
```

9) Propagation totale (2pts) Écrire une procédure qui propage totalement les `-1`, c'est-à-dire : jusqu'à ce que plus aucune propagation ne soit possible. Cette procédure appelle la fonction précédente jusqu'à ce qu'elle ne réalise plus de modification. L'en-tête de cette procédure peut être :

procédure propagationtotale(Grille)

```
void propagationtotale (int G[T][T]);
```

10) Test propagation (1pt) Écrire une fonction qui retourne `1` si la colonne `T-2` du tableau passé en paramètre contient un `-1` et retourne `0` sinon. L'en-tête de cette fonction peut être :

fonction testpropagation(Grille) → entier

```
int  testpropagation (int G[T][T]);
```

11) Afficher le gagnant (1pt) Écrire une procédure qui prend en paramètre une grille de jeu de Hex ne contenant aucune case vide et qui affiche "Victoire du joueur blanc" ou "Victoire du joueur noir".

procédure affichergagnant(Grille)

```
void affichergagnant (int G[T][T]);
```

Partie 3 : Un peu de réflexion (4,5pts)

12) Complexité facile (1pt) Déterminez l'ordre de grandeur du nombre d'opérations de la fonction `propagationpartielle` en fonction de la taille `T` du plateau. Expliquez.

13) Complexité de propagation totale (2pts) La procédure `propagationtotale` fait un nombre indéterminé d'appels à la fonction `propagationpartielle`. Proposez une majoration du nombre de ces appels, en fonction de la taille `T` du plateau. Déduisez-en une majoration de la complexité de la procédure `propagationtotale`. Justifiez vos réponses.

14) Test du gagnant en cours de partie (2,5pts) Dans la partie précédente, nous avons su déterminer le gagnant quand toutes les cases du plateau sont remplies. Très souvent, un joueur aura gagné la partie avant que toutes les cases ne soient remplies. Pour améliorer le jeu, on devrait donc tester le gagnant à chaque coup d'un joueur, mais ceci ralentirait largement le jeu. Justifier cette observation, et proposez une (ou des) méthodes pour tester la victoire d'un joueur en ralentissant le moins possible le jeu. (Inutile d'écrire des algorithmes précis)

Pour aller plus loin (Hors-sujet)

- Essayez de montrer qu'il y a toujours un gagnant.
- Si vous voulez jouer à Hex, commencez par jouer sur une grille de petite taille puis faites la grossir jusqu'à 11x11 cases intérieures ou plus.