Algorithmique Avancée, Complexité, calculabilité

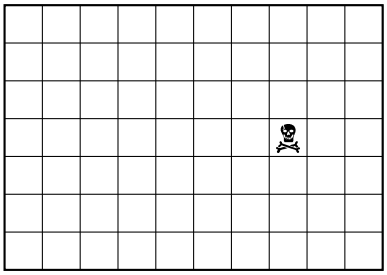
TP n° 2 : Programmation dynamique

Université de Lille-1 / FIL

2013-2014

Description du problème

Dans ce TP, on s'intéresse au jeu suivant. On prend une tablette de chocolat, c'est-à-dire une grille de m colonnes et n lignes, dans laquelle un des carrés est marqué d'un signe spécial. C'est le `carré de la mort`, et on note (i, j) ses coordonnées. Pour fixer les idées, le carré en haut à gauche est le carré de coordonnées $(0, 0)$, tandis que le carré en bas à droite est le carré de coordonnées $(m-1, n-1)$.

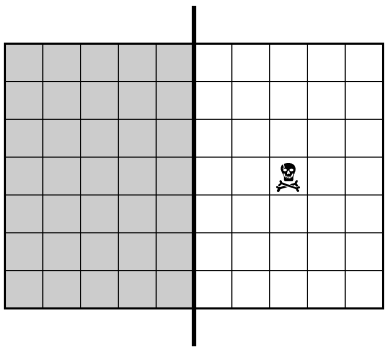


$(m, n) = (10, 7)$ et $(i, j) = (7, 3)$

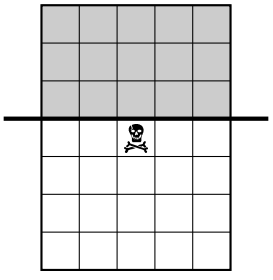
Le jeu se joue de la façon suivante. Le premier joueur prend la tablette, la casse en deux (horizontalement ou verticalement), et donne la partie qui contient le carré mortel à l'autre joueur (la partie restante ne sert plus à rien et peut être consommée). L'autre joueur procède de même. Au fur et à mesure que la partie progresse, la tablette rétrécit, et fatalement au bout d'un moment il ne reste plus que le carré de la mort tout seul. Celui qui reçoit des mains de l'autre le carré de la mort a perdu.

Exemple

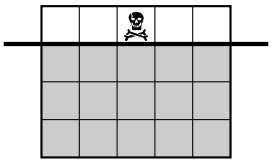
Par exemple, le joueur 1 commence par casser verticalement la tablette en deux moitiés égales :



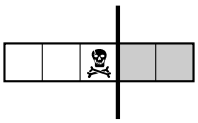
Là-dessus, le joueur 2 décide de la casser horizontalement après la 3^{e} ligne :



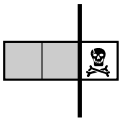
Le joueur 1 décide lui-aussi de la casser horizontalement après la première ligne :



Le joueur deux casse verticalement le long de la 3e colonne :



Et pour finir, le joueur un casse verticalement le long de la 2e colonne, et gagne car il donne le carré de la mort.



Objectif du TP

Le but du jeu est de mettre au point un programme qui joue selon une stratégie optimale, c'est-à-dire qui ne commet *jamais* d'erreur. S'il peut gagner, alors il joue le coup qu'il faut pour gagner. S'il ne peut pas gagner, alors il joue la meilleure défense possible (il essaie de retarder sa défaite).

En effet, dans un jeu comme celui-ci (où il n'y a pas de match nul) on est forcément dans l'une de ces deux situations :

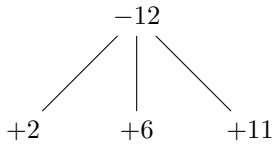
1. Soit le joueur qui commence peut, en jouant correctement, gagner quoi que fasse l'autre.
2. Soit le joueur qui commence, quoi qu'il fasse, va perdre si l'autre joue correctement.

Plus précisément, à un moment donné, le quadruplet (m, n, i, j) définit entièrement l'état actuel du jeu (la « configuration »). A chaque configuration on associe une *valeur*, qui exprime à quel point la position est favorable au joueur dont c'est le tour. La valeur d'une configuration est l'entier $+k$ si le joueur dont c'est le tour peut gagner en maximum k coups, qui que fasse l'autre (il peut gagner plus vite si l'autre joue mal) ; la valeur est l'entier $-k$ si le joueur dont c'est le tour ne peut pas gagner mais qu'il peut survivre au moins k coups face à un adversaire optimal (mais il peut éventuellement gagner si son adversaire commet des erreurs).

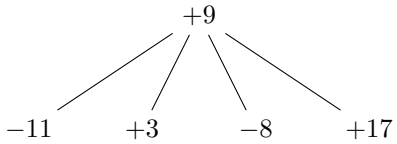
La première étape du TP est de programmer une fonction qui calcule la valeur d'une position.

Méthode

Pour programmer cela, on va utiliser la méthode de la programmation dynamique. En effet, la valeur d'une position peut se calculer à partir des valeurs de positions « plus petites ». Disons donc que si on peut passer de la position A à la position B en une « coupe », alors B est un *successeur* de A. Considérons une position qui a 3 successeurs, dont les valeurs sont respectivement $+2, +6$ et $+11$.

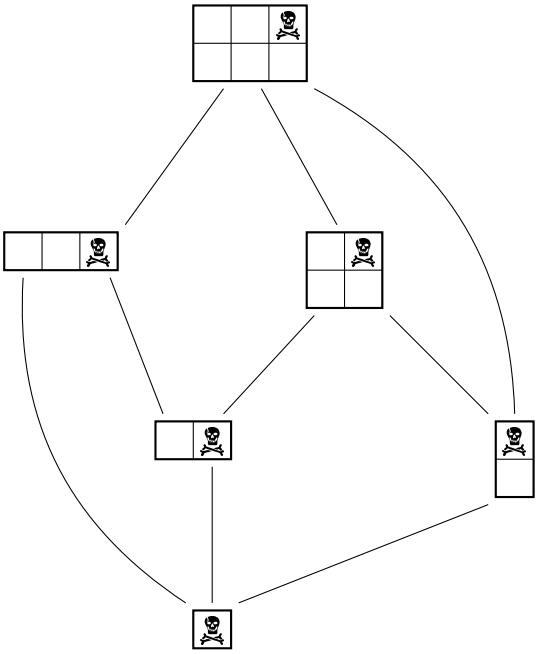


Cela signifie qu'il y a 3 coupes possibles, mais que toutes les coupes offrent une configuration gagnante à l'adversaire. La configuration de départ est donc perdante, et sa valeur est -12. En effet, en choisissant le 3e successeur, l'adversaire ne pourra pas gagner en moins de 11 coups, et donc comme ça on pourra « tenir » 12 coups en tout avant de rendre l'âme.



Maintenant considérons une configuration qui a 4 successeurs de valeurs $-11, +3, -8$ et $+17$. Cette configuration est gagnante, parce qu'en choisissant (par exemple) la première coupe, on place l'adversaire dans une situation perdante. Mais pour gagner le plus vite possible, il vaut mieux choisir le 3e successeur, car alors on est sûr de gagner en 9 étapes maximum. La valeur de la configuration de départ est donc $+9$.

Question 1 : Déterminez à la main les valeurs des positions suivantes, et faites-les figurer sur votre compte-rendu.



Question 2 : Donnez une formule mathématique permettant de calculer la valeur d'une position à partir des valeurs de ses successeurs. Il peut être utile de distinguer le cas où un des successeurs au moins à une valeur négative du cas où ils sont tous positifs.

Programmation de l'évaluation d'une position

Version naïve

Le plus simple, a priori, consiste à écrire une fonction récursive $f(m, n, i, j)$ qui renvoie la valeur de la configuration décrite par ses arguments en s'appelant récursivement pour évaluer les successeurs. Dans l'idéal, vous écrirez un programme qui prend 4 arguments sur la ligne de commande et qui affiche le résultat.

Question 3 : Codez une version naïve de cette fonction, qui ne mémorise pas ses résultats. Testez-là sur les configurations de petite taille (3×2) du début du TP. Combien de temps met votre code à résoudre l'instance $(10, 7, 7, 3)$ (celle qui figure sur la première page du sujet ?) Et $(10, 7, 5, 3)$? Comment expliquer la différence ? La complexité de cette procédure est-elle polynomiale ou exponentielle (justifiez) ?

Version « dynamique »

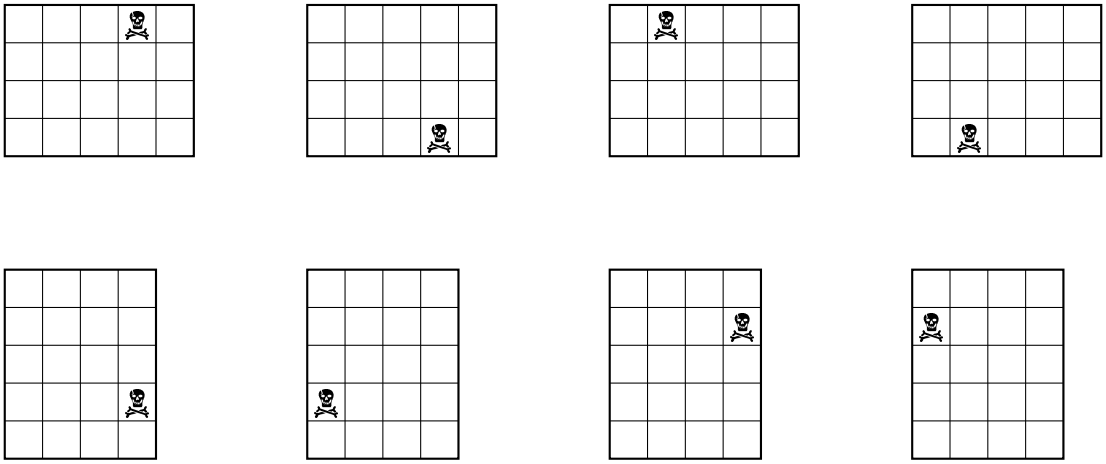
Pour éviter de répéter les calculs, il faut stocker le résultat du calcul dans un tableau. Ce qui complique un tout petit peu les choses, c'est qu'il faudrait que ce tableau soit indexé par (m, n, i, j). Plusieurs solutions sont possibles :

1. On pourrait faire un tableau à 4 dimensions
2. Ou bien on pourrait « coder » le quadruplet (m, n, i, j) en un seul entier (mais dans ce cas-là, il faut que le codage soit correct, et il faut le justifier !).

Question 4 : Déterminez les valeurs des configurations $(100, 100, 50, 50)$ et $(100, 100, 48, 52)$.
Question 5 : Déterminez toutes les paires (i, j) telles que la valeur de la configuration $(127, 127, i, j)$ soit égale à 127.
Question 6 : Donnez l'ordre de grandeur de la complexité de votre procédure en fonction de m et n. Justifiez.

Accélération

Il est possible d'accélérer sensiblement la procédure, en remarquant que le problème possède de nombreuses symétries. En effet, les 8 configurations suivantes ont la même valeur.



Question 7 Expliquez pourquoi toutes ces configurations ont la même valeur.
Question 8 Trouvez un moyen d'en tirer partie et incorporez-le à votre code. Est-ce que ça va plus vite ? (note : ça rend mon code environ 6 fois plus rapide).
Question 9 (question facultative) Utilisez les symétries pour réduire la complexité spatiale de votre programme. Quel est l'ordre de grandeur du gain ?

Programmation du moteur de jeu

Question 10 Écrivez un programme qui prend sur la ligne de commande les 4 arguments décrivant la configuration initiale, et... qui joue (on peut admettre qu'il commence). Quand c'est son tour il doit afficher son coup (dans un format intelligible par un être humain), mais aussi la configuration résultante de son coup, ainsi que son évaluation de cette configuration.

Quand c'est au tour de l'adversaire, il doit demander à l'adversaire d'entrer son coup, calculer la configuration résultante et afficher son évaluation.

Il doit évidemment détecter la fin de la partie.

Les interfaces un peu geek gagnent des points supplémentaires (genre les programmes d'échecs qui affichent en temps réel le nombre de configuration explorées et le nombre de coups envisagés d'avance).