ACT

Pierre Lague & Romain Lecleire

Compte Rendu TP2

Abstract

Ce second TP aborde un problème d'algorithmique connu : le jeu hexapawn. L'objectif est de familiariser les étudiants à réfléchir à différentes solutions tout en considérant le paradigme de programmation dynamique et son efficacité par rapport aux solutions naïves. Le langage de programmation utilisé est `Python 3`.

N.B. Dans la suite du TP, nous n'utilisons pas deepcopy() pour copier les tableaux car cela pose des problèmes de récursion (limite de récursivité atteinte sous python 3).

Q1 Donnez une formule mathématique permettant de calculer la valeur d'une configuration à partir des valeurs de ses successeurs

Avec la lecture du sujet et l'analyse des exemples, nous en avons déduit les formules mathématiques suivantes :

$$\begin{cases} \text{si } y_i > 0 \\ x \to (\max(y_i) + 1) \times (-1) \\ \text{si } y < 0 \\ x \to (\max(y_i) - 1) \times (-1) \\ \text{si } \exists y_i < 0 \text{ et } \exists y_i > 0 \\ x \to (\max(y_i < 0)\, x - 1) + 1 \end{cases}$$

Ces formules nous permettent de calculer la valeur d'une configuration à partir de la valeur de ses successeurs.

Q2 Codez une version naïve qui permet de retourner le nombre minimum de coup avec lesquels on peut gagner.

Pour représenter la structure du jeu nous avons décidé de stocker les configurations dans des tableaux qui nous pouvons parcourir. A une configuration est associée une valeur numérique utilisée dans les formules mathématiques présentées plus haut. La version naïve fera appel à la récursivité pour permettre de calculer tous les coups possibles pour tous les pions afin de "remonter" l'arbre des coups pour trouver la valeur finale de la configuration. (assemblage des solutions des sous-problèmes pour résoudre le problème principal).

Q3 Codez la version dynamique avec "mémoïzation" de la fonction d'évaluation.

La version dynamique avec "mémoïzation" va permettre de :

- gagner du temps en compléxité temporelle car le principe de programmation dynamique empêche les calculs redondants.
- stocker les valeurs des configurations dans table de hashage permettant un accès rapide aux valeurs des configurations si nécéssaires.

N.B. nous avions initialement pensé à stocker les valeurs dans un tableau a plusieurs dimensions, mais cela devient impraticable pour l'accès aux éléments (x, y, blanc, noir, valeur_de_config) et la compléxité spatiale.

Config/Version	Naïve	Memo
Config_3x4_1	0.0002	0.00075
Config_3x4_minus2	0.0022	0.0024
Config_4x4_11	5.065	0.73
Config_5x5_15	120++	47.72

Table 1: Tableaux de comparaison de temps (unités en secondes)

Comparaison de temps d'éxécution

Section 1: Appendix A : naive algorithm

Section 2: Appendix B : dynamic algorithm

Algorithm 1 Naive Algorithm

```
1: procedure NAIVE(current, bd)
2:     coor_pawns ← MAKEPAWNSLIST(bd, current)          ▷ List of current pawns's coordinates
3:     if CHECKLOOSE(current, bd) then
4:         return 0                                      ▷ Game over (lost)
5:     else
6:         next_value ← []
7:         for all (y, x) in coor_pawns do
8:             if current is "W" then
9:                 if bd[y − 1][x] is "- then            ▷ W Moves forward
10:                    bd[y][x] ← "−"
11:                    bd[y − 1][x] ← current
12:                    next_value.append(NAIVE("B", bd))   ▷ Recursive call for the next configuration
13:                end if
14:                if x > 0 and bd[y − 1][x − 1] is "B" then   ▷ W Attacks left
15:                    bd[y][x] ← "−"
16:                    bd[y − 1][x − 1] ← current
17:                    next_value.append(NAIVE("B", bd))   ▷ Recursive call for the next configuration
18:                end if
19:                if x < width − 1 and bd[y − 1][x + 1] is "B" then   ▷ W Attacks right
20:                    bd[y][x] ← "−"
21:                    bd[y − 1][x + 1] ← current
22:                    next_value.append(NAIVE("B", bd))   ▷ Recursive call for the next configuration
23:                end if
24:            else if current is "B" then
25:                if bd[y + 1][x] is "- then            ▷ B Moves forward
26:                    bd[y][x] ← "−"
27:                    bd[y + 1][x] ← current
28:                    next_value.append(NAIVE("W", bd))   ▷ Recursive call for the next configuration
29:                end if
30:                if x > 0 and bd[y + 1][x − 1] is "W" then   ▷ B Attacks left
31:                    bd[y][x] ← "−"
32:                    bd[y + 1][x − 1] ← current
33:                    next_value.append(NAIVE("W", bd))   ▷ Recursive call for the next configuration
34:                end if
35:                if x < width − 1 and bd[y + 1][x + 1] is "W" then   ▷ B Attacks right
36:                    bd[y][x] ← "−"
37:                    bd[y + 1][x + 1] ← current
38:                    next_value.append(NAIVE("W", bd))   ▷ Recursive call for the next configuration
39:                end if
40:            end if
41:        end for
42:        if LENGTH(next_value) = 0 then
43:            return 0
44:        else
45:            return GETVALUE(next_value)
46:        end if
47:    end if
48: end procedure
```

Algorithm 2 Memoization Algorithm

```
1: procedure MEMOIZATION(current, bd)
2:     coor_pawns ← MAKEPAWNSLIST(bd, current)                ▷ List of current pawns
3:     bd_hash ← HASHVALUE(bd)                                ▷ Hash code of bd
4:     if bd_hash is in bd_dict then                          ▷ Hash code already exists
5:         return bd_dict[bd_hash]
6:     end if
7:     if CHECKLOOSE(current, bd) then                        ▷ Check if lost
8:         if bd_hash is not in bd_dict then
9:             bd_dict[bd_hash] ← 0
10:        end if
11:        return bd_dict[bd_hash]                            ▷ Game over (lost)
12:    else
13:        next_value ← []
14:        for all (y, x) in coor_pawns do
15:            if current is "W" then
16:                if bd[y − 1][x] is "- then                 ▷ W Moves forward
17:                    bd[y][x] ← "−"
18:                    bd[y − 1][x] ← current
19:                    next_value.append(MEMOIZATION("B", bd))   ▷ Recursive call for the next configuration
20:                end if
21:                if x > 0 and bd[y − 1][x − 1] is "B" then   ▷ W Attacks left
22:                    bd[y][x] ← "−"
23:                    bd[y − 1][x − 1] ← current
24:                    next_value.append(MEMOIZATION("B", bd))   ▷ Recursive call for the next configuration
25:                end if
26:                if x < width − 1 and bd[y − 1][x + 1] is "B" then   ▷ W Attacks right
27:                    bd[y][x] ← "−"
28:                    bd[y − 1][x + 1] ← current
29:                    next_value.append(MEMOIZATION("B", bd))   ▷ Recursive call for the next configuration
30:                end if
31:            else if current is "B" then
32:                if bd[y + 1][x] is "- then                 ▷ B Moves forward
33:                    bd[y][x] ← "−"
34:                    bd[y + 1][x] ← current
35:                    next_value.append(MEMOIZATION("W", bd))   ▷ Recursive call for the next configuration
36:                end if
37:                if x > 0 and bd[y + 1][x − 1] is "W" then   ▷ B Attacks left
38:                    bd[y][x] ← "−"
39:                    bd[y + 1][x − 1] ← current
40:                    next_value.append(MEMOIZATION("W", bd))   ▷ Recursive call for the next configuration
41:                end if
42:                if x < width − 1 and bd[y + 1][x + 1] is "W" then   ▷ B Attacks right
43:                    bd[y][x] ← "−"
44:                    bd[y + 1][x + 1] ← current
45:                    next_value.append(MEMOIZATION("W", bd))   ▷ Recursive call for the next configuration
46:                end if
47:            end if
48:        end for
49:        if LENGTH(next_value) = 0 then                     ▷ If no more moves
50:            if bd_hash is not in bd_dict then
51:                bd_dict[bd_hash] ← 0                       ▷ No move and not in bd_dict, so set to 0 in bd_dict
52:            end if
53:            return bd_dict[bd_hash]                        ▷ Game over (lost)
54:        else
55:            bd_dict[bd_hash] ← GETVALUE(next_value)        ▷ Add value to bd_dict and return the result
56:            return bd_dict[bd_hash]
57:        end if
58:    end if
59: end procedure
```