TD - n° 1
(Correction)

Algorithmes pour les jeux

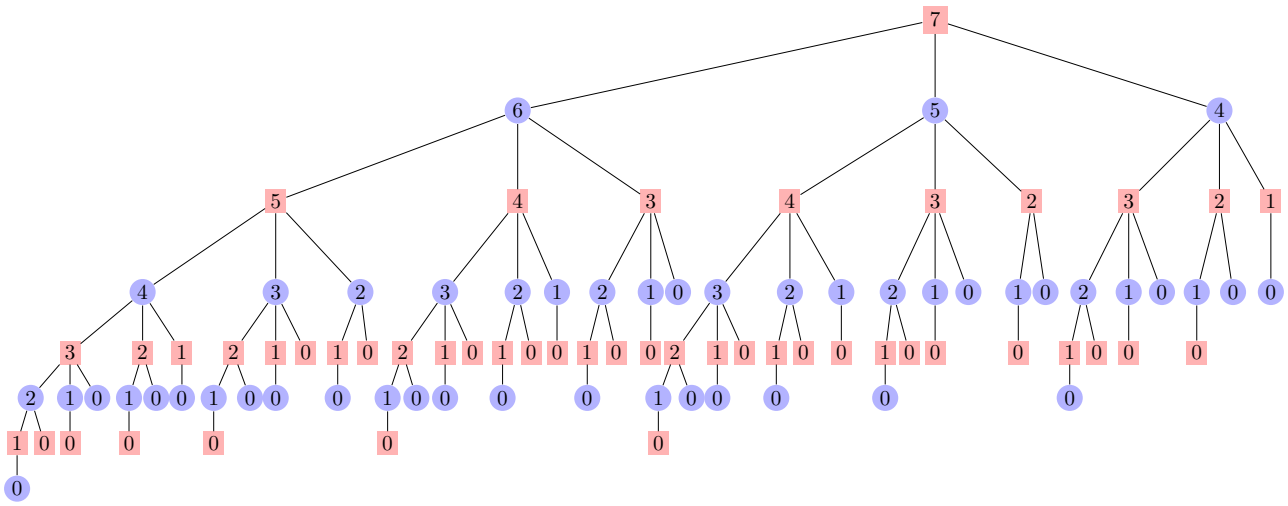
Questions sur le jeu de Nim

On s'intéresse ici à la modélisation d'une des nombreuses variantes du jeu de Nim, dans laquelle on dispose au départ d'un ensemble de N allumettes et où chaque joueur peut tour à tour enlever $1, 2$ ou 3 allumettes. Le joueur perdant est celui qui ramasse la dernière allumette.

1. Développez l'arbre de recherche entier pour $N = 7$.

Correction :

On peut représenter l'arbre complet de la façon suivante :



Les noeuds ami (i.e. max) figurent ici en rouge et les noeuds ennemi (i.e. min) en bleu.

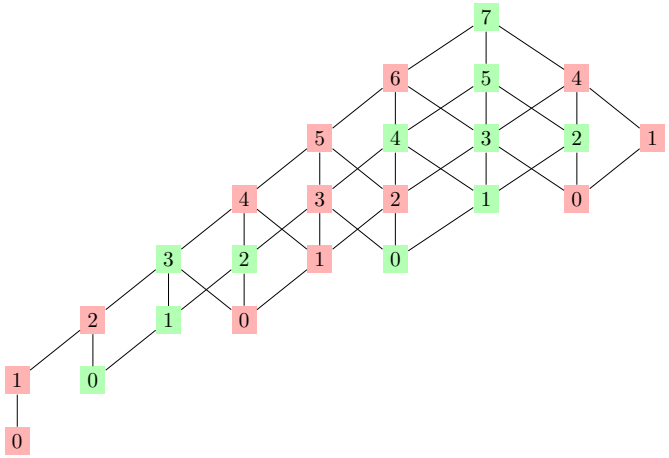
Note : il est généralement d'usage de relier les arcs issus d'un noeud min par un arc de cercle, mais je n'ai pas encore trouvé la façon de faire ceci de façon simple en latex avec le module tikz.

2. Existe-t-il des stratégies gagnantes pour ce jeu ?

Correction : Pour savoir s'il existe une stratégie gagnante, il faut faire tourner l'algorithme vu en cours. On étiquette chaque noeud feuille gagnante (i.e. 1 pour Ami) et perdant (i.e. 0 pour Ennemi) et on fait remonter les estimations vers la racine en étiquetant chaque noeud interne Ami gagnant si l'un de ses fils est lui même gagnant et en perdant si tous ses fils sont perdants. On procède inversement pour les noeud Ennemi.

Dans l'arbre ci-dessous, les noeuds gagnants sont étiquetés en vert et les perdants en rouge.

NB : Pour alléger la représentation, l'arbre de jeu est ici représenté sous la forme d'un graphe.



3. En prenant comme heuristique $h(n) = N - n$ pour $n > 1$, $h(1) = -\infty$ et $h(0) = +\infty$ pour Ami, et $h(1) = +\infty$ et $h(0) = -\infty$ pour Ennemi, simulez une partie entière pour $N = 10$ entre deux I.A., l'une voyant à 2 demi-coups, et l'autre à 4 demi-coups.

Correction :

Faire faire les deux développements possibles aux étudiants, en alternant dans les rangs :

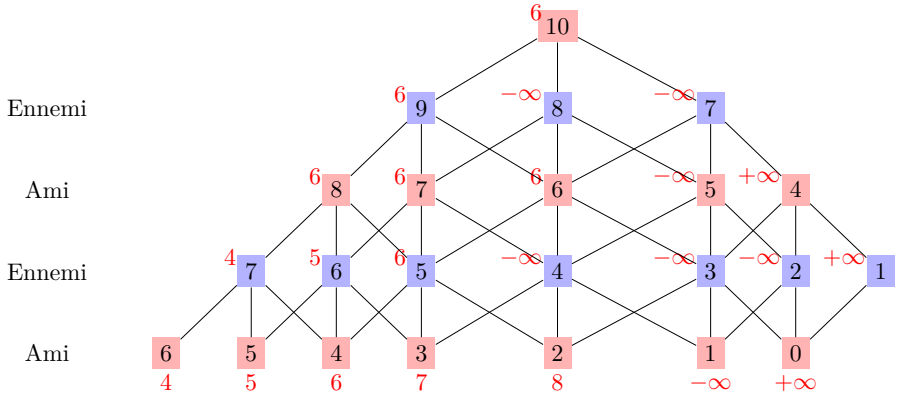
- L'un fait un développement en prenant pour le joueur ami l'IA à 4 demi coups et pour le joueur ennemi, l'IA à 2 demi coups.
- Le voisin fait l'inverse : ami à 2 demi coups et ennemi, à 4 demi coups.

Ca les oblige à se concentrer sans regarder ce que fait le voisin.

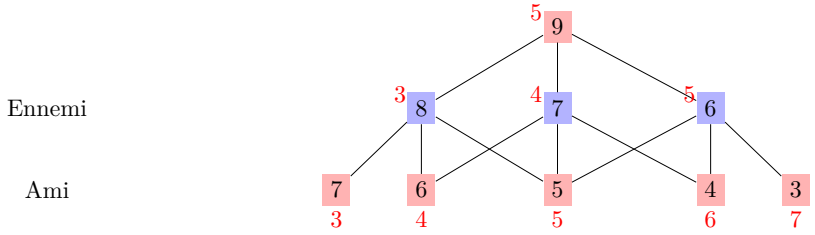
Insister sur le fait qu'après chaque coup joué par un joueur, il faut recommencer un nouvel arbre dont la racine correspond à l'état résultant du coup précédent. Les étudiants sont flemmards et ont tendance à tout vouloir faire sur le même arbre.

Exemple :

Avec le joueur Ami à 4 demi coups, le premier coup de jeu donne lieu à l'arbre suivant :

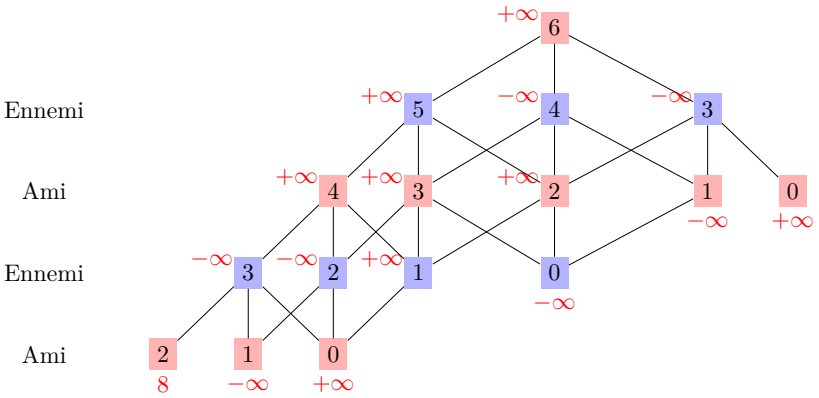


La valeur remontée à la racine est 6 et correspond au coup qui consiste a prendre une allumette. Le joueur Ami joue donc ce coup et le joueur ennemi lance son IA (avec seulement 2 demi-coups) à partir de l'état 9. Notons que pour le joueur ennemi les rôles sont inversés. Le joueur Ami (du début) devient le joueur ennemi, et lui devient le joueur Ami.



Le joueur ennemi en déduit donc le meilleur coup à jouer pour lui des de prendre 3 allumettes.

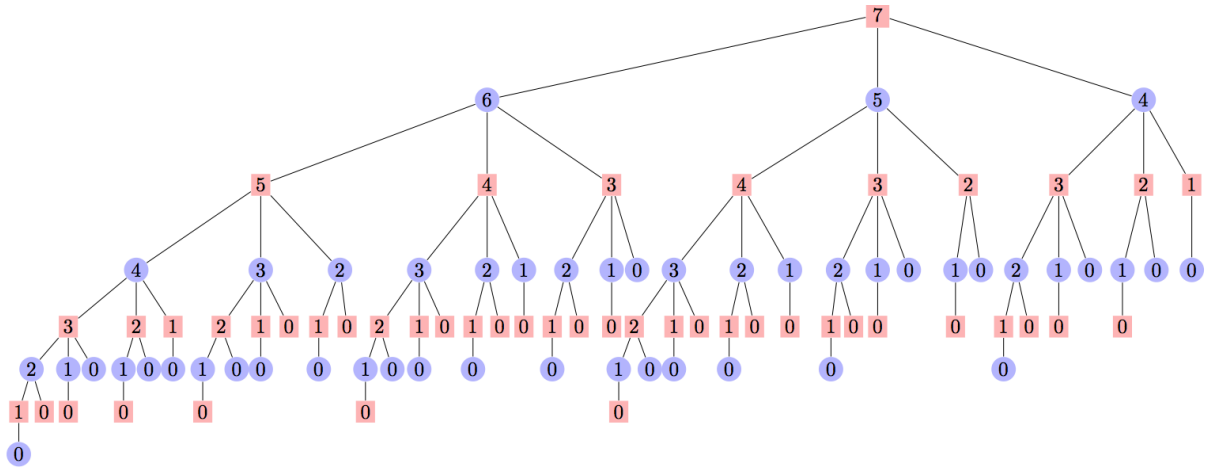
Le joueur ami repart alors de cet état :



Comme la valeur $+\infty$ est remontée à la racine il sait à ce stade qu'il a une stratégie gagnante et qu'il doit enlever une allumette pour jouer en 5.

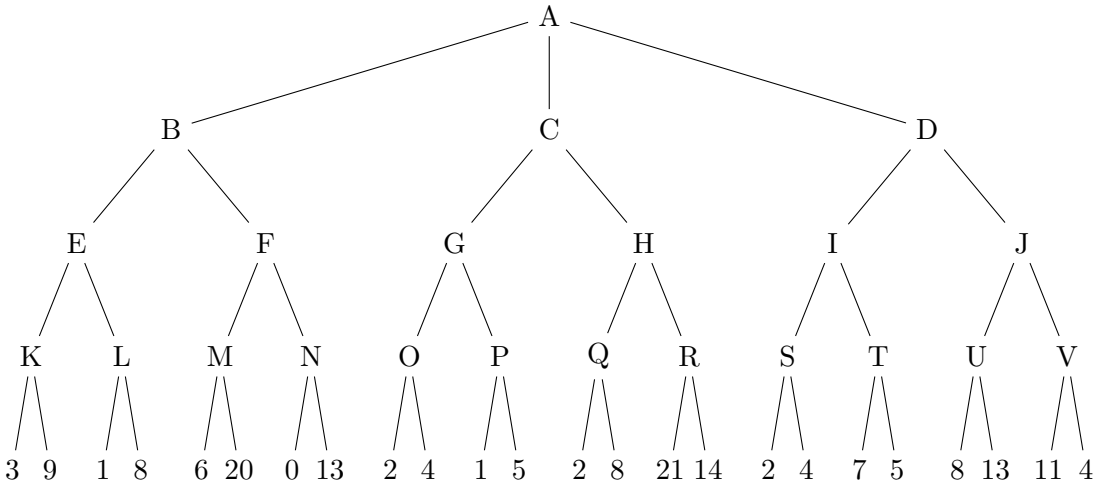
Le joueur Ennemi pourra alors jouer n'importe quoi, la réponse sera de jouer le complémentaire à 4 pour ramener l'adversaire dans l'état où il ne reste plus qu'une allumette.

On peut représenter l'arbre complet de la façon suivante :

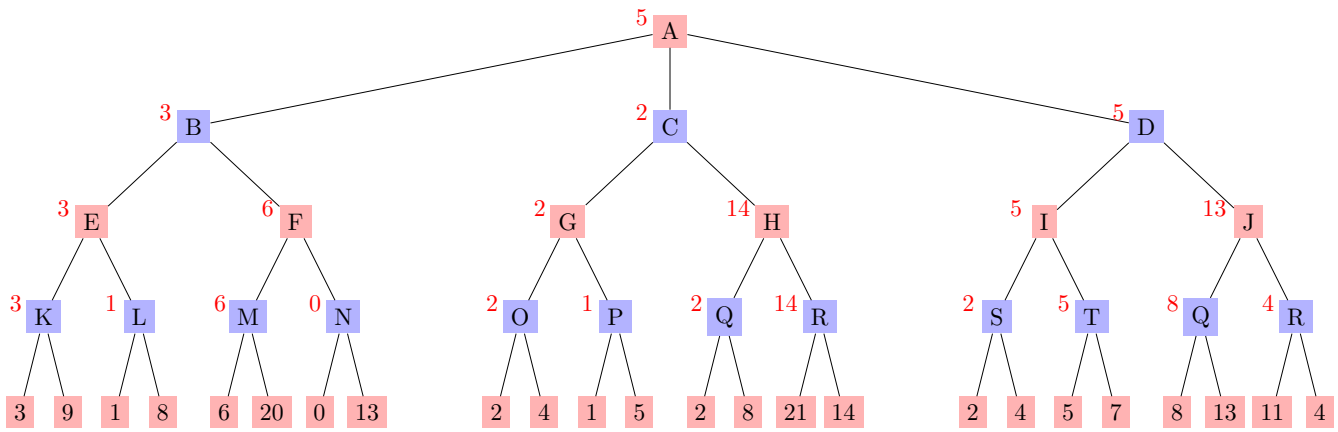


Exercice 1 Algorithmes Minimax et $\alpha\beta$

1. Appliquez rapidement Minimax sur l'arbre de jeu ci-dessous pour déterminer le coup qui semble le plus prometteur pour le premier joueur.



Correction :



2. Appliquez l'algorithme $\alpha\beta$ sur ce même arbre.
Vous redévelopperez complètement sur une feuille séparée l'arbre exploré. Indiquez pour chaque coupe effectuée, s'il s'agit d'une coupe α ou β et s'il s'agit d'une coupe de surface ou d'une coupe profonde.

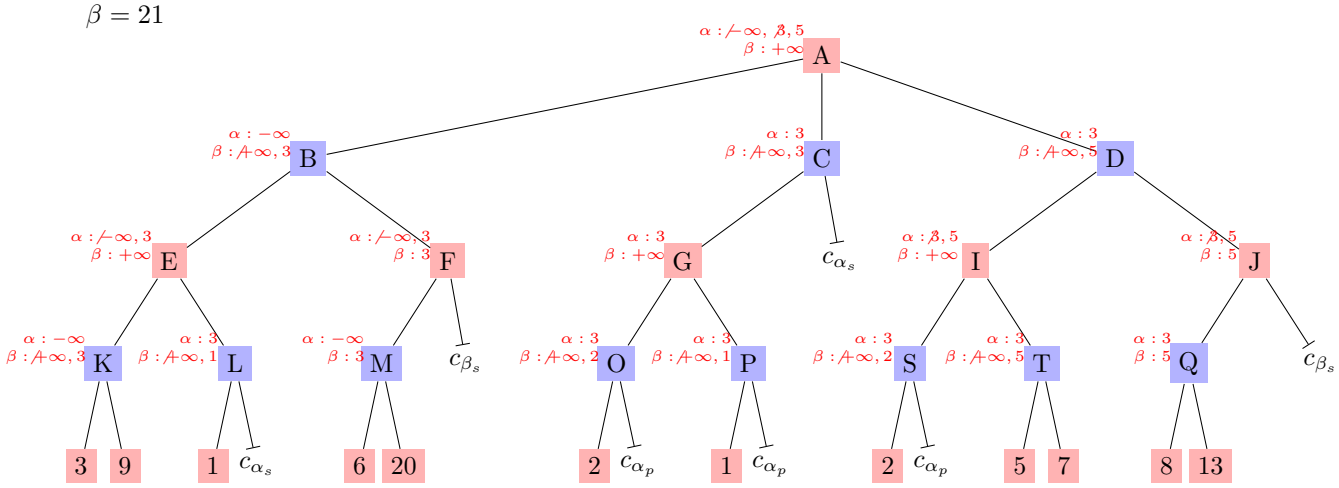
Correction : Les étudiants sont paresseux... ils ne veulent pas se fatiguer à recopier l'arbre.

Expliquer que c'est bien un parcours en profondeur de gauche à droite que l'on fait et qu'il est préférable de refaire le graphe au fur et à mesure du parcours pour bien comprendre comment cela marche.

Il est important de leur faire rédiger la solution de façon à ce que l'on puisse comprendre à postériori leur raisonnement.

Le mieux est de leur faire marquer systématiquement au niveau de chaque noeud les valeurs des paramètres α et β. Lors de la mise à jour d'une valeur, faire en sorte de rayer la valeur mise à jour de façon à ce qu'on l'on puisse malgré tout comprendre quelles ont été les valeurs successives

Exemple :
$\alpha = -\not\infty, \not 2, \not 7, 15$
$\beta = 21$



nature des coupes :
c_{α_s} : coupe-α de surface.
c_{β_s} : coupe-β de surface.
c_{α_p} : coupe-α profonde.

Coupe-α : la cause de la coupe est la valeur de α qui est supérieure à la valeur remontée pour β.
Coupe-β : la cause de la coupe est la valeur de β qui est inférieure à la valeur remontée pour α.
Coupe de surface : la valeur à l'origine de la coupe vient du parent direct
Coupe de profonde : la valeur à l'origine de la coupe vient d'un ancêtre du parent direct

Correction : Même graphe que précédemment en inversant les rôles et signes de α et β sur les noeuds min.

3. Proposez un ordre de développement des fils de chaque noeud permettant de minimiser le nombre de noeuds développés par $\alpha\beta$.

Correction : Ordonner les fils min d'un noeud max par valeurs décroissante, et les fils max d'un noeud min par valeurs croissantes.

Exercice 2 Iterative Deepening $\alpha\beta$

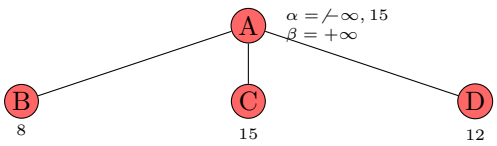
On considère l'algorithme $\alpha\beta$, dans sa version où il exécute une recherche à profondeur incrémentale. On suppose que lors de la première exploration de chaque nouveau niveau de l'arbre, les différents fils d'un même nœud sont explorés suivant l'ordre lexicographique.

1. Détailler le fonctionnement de cet algorithme sur le graphe décrit implicitement dans le tableau suivant :

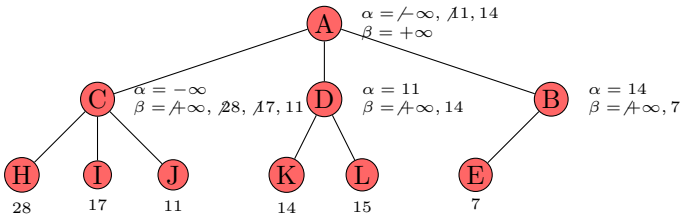
noeud n	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
$h(n)$	13	8	15	12	7	55	19	28	17	11	14	15	10	11	
$succ(n)$	B C D	E F G	H I J	K L	M N	O P	Q R S	T U	V W	X Y	Z A1	A2 A3	a b	c d	
noeud n	O	P	Q	R	S	T	U	V	W	X	Y	Z	A1	A2	A3
$h(n)$	20	13	10	11	6	14	30	18	19	12	10	14	12	15	8
$succ(n)$	e	f	g	h i	j k	l	m n	o	p	q r s	t u	v w	x1 x2	y1 y2	z1 z 2
noeud n	a	b	c	d	e	f	g	h	i	j	k	l	m	n	
$h(n)$	30	12	8	15	19	3	7	17	12	10	6	14	12	8	
noeud n	o	p	q	r	s	t	u	v	w	x1	x2	y1	y2	z1	z2
$h(n)$	10	8	12	8	15	6	7	14	19	6	14	15	16	12	10

Correction :

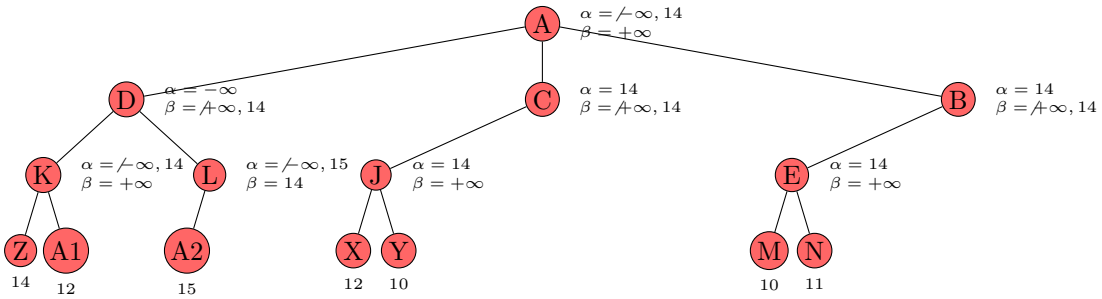
ProfMax = 1 :



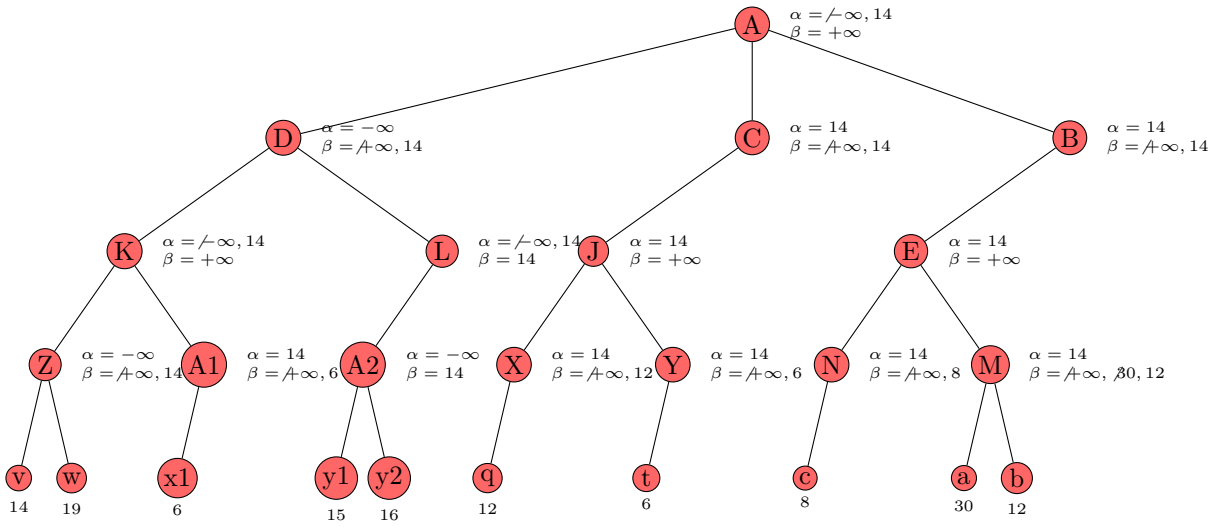
ProfMax = 2 :



ProfMax = 3 :



ProfMax = 4 :



2. Comment gérer en pratique le réordonnancement des fils tout en limitant la consommation mémoire ?

Correction :

Pour gérer en pratique le réordonnancement des fils, il ne faut pas garder tout l'arbre en mémoire sinon cela prendrait trop de temps à trier à chaque fois, et surtout le traitement en mémoire de tout l'arbre prendrait au final trop de place...

On peut utiliser une table de hashage avec oubli en cas de collision pour stocker les noeuds déjà visités et leur valeur heuristique associée. Cela permet de maitriser la mémoire allouée.

Mieux encore, on peut ne stocker que la valeur de hashage de l'état considéré, sans garder l'état lui-même. Ainsi on gagne énormément de place (les plateaux ne sont pas en mémoire, seul un entier long l'est). D'accord, on peut se planter en pensant revoir un noeud déjà vu, mais ce n'est pas grave dans la mesure où la valeur heuristique du noeud ne sert qu'à ordonner les fils. Cela ne remet pas en cause la valeur minimax finale, qui reste, elle toujours bonne.

Autre optimisation possible : ne pas garder la valeur heuristique de chaque noeud (ce qui oblige à ordonner tous les fils à chaque fois), mais garder les 2 ou 3 meilleurs coups pour chaque noeud... Ainsi on n'a plus rien à faire pour ordonner les fils... Et les fils 4, 5, ... ben on les explore au hasard :).

Tout le problème vient maintenant de pouvoir associer une valeur de hashage facilement à un état. Ce sera vu en cours.

En gros : A chaque position de chaque pièce sur le plateau on associe un entier aléatoire, mais stocké en mémoire. La valeur de hashage du plateau est le xor logique de toutes les valeurs associées aux positions de chaque pièces. Ainsi, un mouvement sur le plateau (l'exemple est pris sur les échecs) se fait en faisant très peu d'opération (un xor pour enlever la piece et un nouveau xor pour la mettre à son arrivée).

Exercice 3 Etude analytique de l'algorithme Iterative Deepening $\alpha\beta$

On suppose qu'une machine met T_h secondes pour évaluer la valeur heuristique d'une configuration de jeu.

1. Expliquez pourquoi on peut, dans les questions suivantes, négliger le temps de création des noeuds internes de l'arbre de recherche pour estimer le coût total des différents algorithmes.

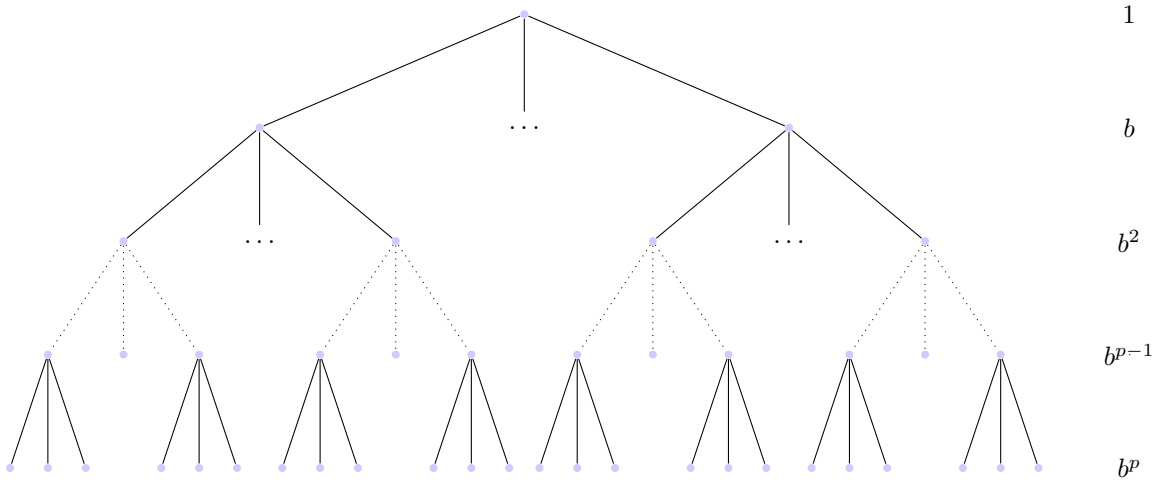
   Correction :

   On peut faire abstraction du temps mis pour les noeuds internes car ils sont en minorité par rapport aux feuilles (cela est d'autant plus vrai que le facteur de branchement est important) et parce que le temps de création des noeuds (interne ou feuilles) est souvent plus petit (un facteur d'échelle dans la plupart des jeux) que le temps d'évaluation des feuilles (le calcul de la fonction heuristique).

2. Donnez le temps nécessaire à MiniMax pour estimer la valeur de la racine d'un arbre de jeu dont le facteur de branchement est b et l'horizon est fixé à p (en demi-coups).

   Correction :

   Estimation du nombre de noeuds apparus lors d'un développement Minimax à profondeur p :



   Pour MiniMax, il faut estimer la valeur de b^p feuilles, soit un temps de $T = T_h.b^p$.
   Application numérique : $T = \frac{30^{10}}{10^6} = \frac{5.9049 \times 10^{14}}{10^6} = 5.9 \times 10^{14-6} = 5.9 \times 10^8$ (soit plus de 18 ans).

3. Donnez un encadrement du temps nécessaire à $\alpha\beta$ pour effectuer la même estimation.

   Correction :

   Dans le pire des cas, la complexité de $\alpha\beta$ est celle de $MiniMax$. Si l'on n'a pas de chance, il faut donc les 18 ans.
   Dans le meilleur des cas $T = T_h.b^{p/2}$.

   Application numérique : : $T = \frac{30^5}{10^6} = \frac{2.43 \times 10^7}{10^6} = 2.43 \times 10^{7-6} = 24.3s$
   ... à comparer avec les 18 ans !

4. Imaginons que l'on utilise un algorithme de type MiniMax à profondeur incrémentale, oubliant tout entre deux recherches successives. En toute généralité, quel serait le pourcentage de feuilles supplémentaires développées par un tel algorithme, par rapport à un MiniMax classique, pour arriver à la même profondeur, en fonction de b et de p ?

   Correction :

   Le nombre de feuilles évaluées par l'algorithme $Minimax$ classique est de b^p feuilles.
   Le nombre de feuilles évaluées par l'algorithme $MinimaxID$ est de : $b + b^2 + ... + b^p$.
   La différence est donc de $b + b^2 + ... + b^{p-1} = b.(1 + b + b^2 + \ldots + b^{p-2}) = b.\frac{(b^{p-1}-1)}{(b-1)}$.
   Le pourcentage recherché est donc de : $b.\frac{b^{p-1}-1}{(b-1).b^p} = \frac{b^{p-1}-1}{b^{p-1}.(b-1)} = \frac{(1-\frac{1}{b^{p-1}})}{(b-1)} = \frac{1}{(b-1)} - \frac{1}{b^{p-1}(b-1)}$

... qui tend vers $\frac{1}{b-1}$ lorsque p augmente.

Donc plus le facteur de branchement est grand, moins le temps perdu par ID est important.

5. On souhaite à présent comparer la version classique de l'algorithme $\alpha\beta$ avec sa version à profondeur incrémentale. Supposons qu'à profondeur égale, une version à profondeur incrémentale explore 50% de nœuds en plus que la taille de l'arbre critique (optimal) et qu'une version $\alpha\beta$ coupe la moitié des noeuds explorés par MiniMax (c'est une supposition réaliste : cela simule le fait que la version iterative deepening va utiliser les lancements précédents pour ordonner les fils et se rapprocher plus de l'arbre critique, alors que la version directe n'a pas de moyen simple de se rapprocher de l'arbre critique). Comparez les deux approches en termes de noeuds supplémentaires développés.

Correction :

Si l'on suppose que l'algorithme $\alpha\beta$ (non incrémental) coupe en pratique la moitié des noeuds explorés par MiniMax, cela revient en pratique à dire qu'il explore en gros la moitié des feuilles (puisque c'est le facteur dominant dans le coût) et que donc il lui faut :

$$T_{\alpha\beta(p)} = \frac{T_h.b^p}{2}$$

A.N : $T_{\alpha\beta(p)} = 2.95 * 10^8$ (soit plus de 9 ans).

Si l'on suppose que $\alpha\beta$ incrémental explore 50% de nœuds en plus que la taille de l'arbre critique... chaque appel à une profondeur i doit normalement conduire à l'évaluation de $\frac{3}{2}.b^{i/2}$ feuilles.

Remarque : on pourrait être tenté de ne faire la somme que de 1 à $p/2$ mais cela pose en pratique un problème car on a bien p itérations et les exposants sont rationnels. En fait $b^{i/2} = (\sqrt{b})^i$.

On a donc

$$T_{\alpha\beta ID(p)} = T_h.\frac{3}{2}.\sqrt{b}.(1 + ... + b^{\frac{p-2}{2}} + b^{\frac{p-1}{2}})$$

Soit

$$T_h.\frac{3}{2}.\sqrt{b}.\frac{\sqrt{b}^p - 1}{\sqrt{b} - 1}$$

A.N. : $T_{\alpha\beta ID(p)} = 45s$

Si on fait le rapport des deux on obtient :

$$\mathrm{T}_{\alpha\beta ID(p)} \over T_{\alpha\beta(p)} = \frac{T_h.\frac{3}{2}.\sqrt{b}.\frac{\sqrt{b}^p-1}{\sqrt{b}-1}}{\frac{T_h.b^p}{2}} = \frac{3.\sqrt{b}.\frac{\sqrt{b}^p-1}{\sqrt{b}-1}}{b^p} = \frac{3.(\sqrt{b}^p-1)}{\sqrt{b}^{2p-1}.(\sqrt{b}-1)} = \frac{3}{\sqrt{b}-1}.(\frac{1}{\sqrt{b}^{p-1}} - \frac{1}{\sqrt{b}^{2p-1}}) = O(\frac{1}{b^{p/2}})$$

Exercice 4 Jeux avec tirage de dés

On désire adapter le principe de la recherche $\alpha\beta$ au cas des jeux à deux joueurs dont les mouvements possibles dépendent d'un tirage de dés, effectué avant que chaque joueur ne décide quoi jouer (par exemple le jeux des petits chevaux).

1. Adaptez l'algorithme MiniMax pour ce problème.

   Correction : La solution de ce problème est un algo connu sous le nom de expectimax Au lieu d'avoir des niveaux Ami/Ennemi/Ami/Ennemi dans l'arbre de recherche, on tire réellement les dés et ensuite on lance une recherche sur un arbre du type Ami/Dés (Ennemi)/Ennemi/Dés (Ami)/Ami/Dés(Ennemi)/Ennemi/...

   Entre Ami et Ennemi, il y a a chaque fois un tirage de dés qui peut être vu comme un troisième joueur.

   Pour prendre en compte tous les tirages de dés, il faut faire par exemple la moyenne des coups des fils à chaque niveau simulant le tirage de dés (c'est bien cela "expectimax").

2. Comment transformer votre algorithme pour avoir des solutions optimistes ou pessimistes par rapport aux tirages possibles de dés ?

   Correction : Pour être optimiste, on peut avoir plutot le MAX plutot que la moyenne (j'ai de la chance, les dés vont maximiser mes gains (Max est utilisé à la fois pour la simulation des dés Ami mais aussi pour les dés ennemis).

   Pour être pessimiste, on prendre le Min.

3. Est-il encore possible de trouver des stratégies gagnantes ?

   Correction : On peut bien entendu toujours trouver une stratégie gagnante (une stratégie qui prend tous les cas des tirages de dés en compte en fait, comme si les dés étaient du coté ennemi).

4. Peut-on (sous quelles conditions) adapter l'algorithme $\alpha\beta$ pour ce problème ?

   Correction : Pour adapter les coupes alpha-beta, il faut imposer des bornes à la fonction heuristique.

   Prenons le cas où alpha = 9 dans en arrivant à un noeud DES qui a dix branches filles (on tire un dés à 10 faces). Si on sait par exemple que toutes les valeurs heuristiques sont entre -10 et 10, et si la premiere branche nous remonte par exemple un -10, on sait que la valeur minimax du noeud étudié sera entre -10 et 8 (la somme de toutes les valeurs des fils est entre -100 (si les 9 autres sont à -10) et 80 (si les 9 autres sont à +10). Dans ce cas la valeur du noeud ne peut être plus grande que alpha. Donc on coupe.