UV IA41 – Automne 2007
Concepts fondamentaux en Intelligence Artificielle et langages dédiés

Examen Final
Jeudi 24 Janvier 2008, de 8h à 10h, en salle P108, site de Sévenans

Coefficient : 40%.
Aucun document autorisé.

MERCI d'utiliser une copie différente pour chaque partie.

Remarques et conseils :
- Lisez **attentivement** chaque question avant d'y répondre.
- Indiquer clairement sur votre copie le numéro de l'exercice avant d'y répondre.
- Expliquez autant que possible vos choix lors de la définition d'un prédicat.
- Le barême défini ci-après est susceptible d'être modifié.

Partie I : Questions de logiques (6 points)

Exercice 1 : Questions de cours (3 points)

1. Qu'est-ce qu'une clause ?
2. Quelle différence existe-t'il entre une clause standard et une clause de Horn ?
3. Quels sont les deux mécanismes fondateurs de PROLOG qui lui permettent de donner plusieurs réponses lors d'une requête ?

Exercice 2 : Résolution d'un ensemble de clauses (3 points)

Démontrez la validité du raisonnement suivant à l'aide du principe de résolution :
- Tout homme est un primate.
- Les dauphins ne sont pas des primates.
- Il y a des dauphins qui sont intelligents.
- Donc on peut ne pas être un homme et être intelligent.

Partie II : Problème de satisfaction de contraintes (7 points)

Exercice 3 : Le problème des n reines (7 points)

Le problème des n reines consiste à placer n reines sur un échiquier de dimension n x n sans qu'aucune d'entre elles ne soit en prise avec une autre. Deux reines ne peuvent pas être alignées horizontalement, verticalement ou diagonalement.

On propose de modéliser une solution de ce problème par une liste des coordonnées des n reines, de la forme : [[X1,Y1], [X2,Y2], ..., [Xn,Yn]], avec Xi, Yi les coordonnées de la reine numéro i.

Pour réussir à positionner n reines sur un échiquier de dimension n x n, on constate que l'on doit obligatoirement disposer une reine par colonne pour éviter qu'elles ne soient en prises verticalement (on peut également faire la même constatation pour l'alignement horizontal). Cette constatation permet de simplifier l'énumération des solutions, puisque l'on peut se restreindre aux solutions pour lesquelles la reine numéro i est dans la colonne i.

1) Définir un prédicat « buildFirst(+N,-L) » qui construit une liste de N couples (Xi,Yi) tels que Xi=N-i+1 et Yi est indéfini.

Exemple : ?- buildFirst(4,L).

L=[[4,_],[3,_],[2,_],[1,_]]

2) Définir un prédicat « buildAll(+N,-L) » qui énumère toutes les valeurs possibles pour les lignes des N reines de la liste L. Cette énumération porte donc sur les valeurs Y des couples de coordonnées de la liste L, sans modifier la colonne X, et sans tenir compte des contraintes de prise entre reines.

3) Définir un prédicat « checkAll(+L) » qui vérifie que toutes les reines de la liste L ne sont pas en prises les unes avec les autres.

4) Ecrire enfin le prédicat « nqueen(+N,-L) » qui fournit les solutions au problème des n reines.

Partie III : PROLOG, MinMax et A* (7 points)

Exercice 4 : Algorithme d'Euclide en PROLOG (2 points)

On souhaite écrire une fonction calculant le PGCD (Plus Grand Commun Diviseur) de deux nombres entiers strictement positifs *a* et *b* (avec $a > b$). Pour cela on utilise l'algorithme d'Euclide décrit ci-dessous :

« Soit *a* et *b* deux entiers strictement positifs tel que $a > b$. On calcule le reste de la division entière de *a* par *b* que l'on note *r*. Si *r* est nul alors le PGCD entre *a* et *b* est *b*, sinon le PGCD correspond au PGCD entre *b* et *r*. Ainsi le PGCD correspond au dernier reste non nul. »

Exemple :

PGCD(28, 6)

Le reste de la division entière de 28 par 6 est 4, on calcul donc le PGCD entre 6 et 4.

PGCD(6, 4)

Le reste de la division entière de 6 par 4 est 2, on calcul donc le PGCD entre 4 et 2.

PGCD(4, 2)

Le reste de la division entière de 4 par 2 est 0, le PGCD est donc 2.

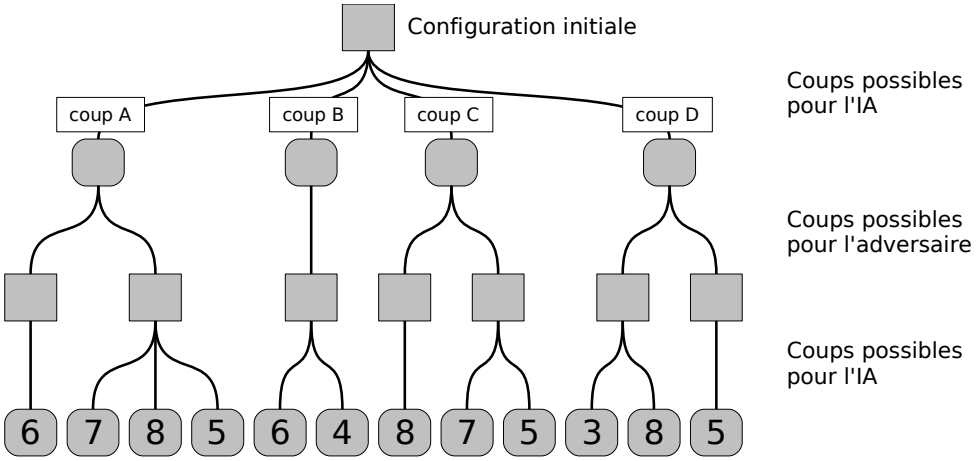
Donner le prédicat *pgcd* en PROLOG permettant de calculer le PGCD de deux nombres entiers strictement positifs à l'aide de l'algorithme d'Euclide.

Rappels de prédicats PROLOG :

is : affectation
=:= : test d'égalité arithmétique
=\= : test d'inégalité arithmétique
mod : reste de la division entière

Exercice 5 : Algorithme MinMax (2 points)

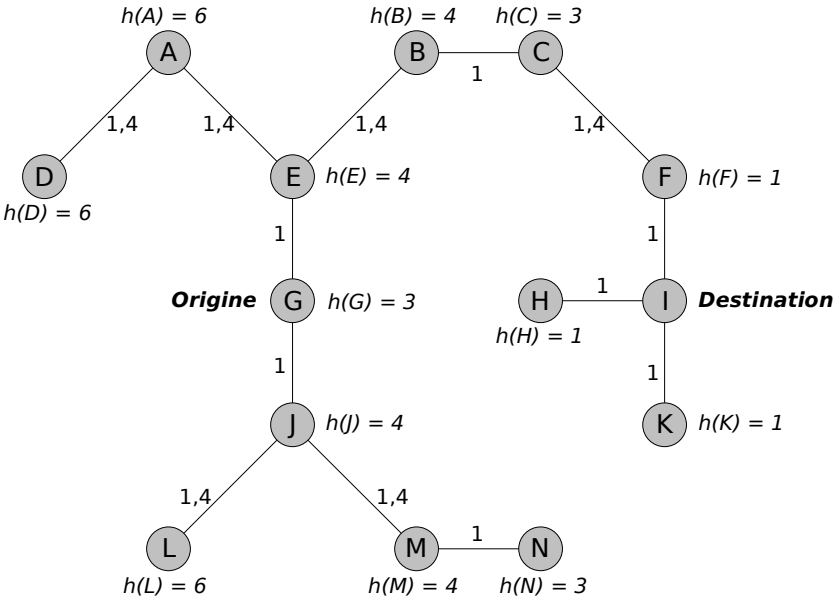
On souhaite réaliser une intelligence artificielle capable de jouer à un jeu à deux personnes. Pour cela on réalise un algorithme MinMax capable de trouver le meilleur coup à jouer en évaluant l'arbre de jeu sur une profondeur donnée. L'arbre représenté ci-dessous correspond à un arbre de jeu de profondeur 3 permettant à une IA de choisir son prochain coup. L'évaluation des configurations est indiquée sur chacune des feuilles de l'arbre.



En appliquant l'algorithme MinMax, déterminez la valeur du nœud racine et donnez le coup qu'effectuera l'IA.

Exercice 6 : Algorithme A* (3 points)

L'algorithme A* permet de calculer le plus court chemin entre deux points dans un graphe. La figure suivante représente un graphe où est effectué le parcours par A*. Sur chaque nœud est indiquée la valeur de l'heuristique (*h*) permettant d'estimer la distance de la destination. De plus, la distance effective entre les nœuds du graphe est indiquée sur les arcs. Le nœud d'origine est le nœud G et le nœud destination est le nœud I.



Après plusieurs itérations, les listes OUVERT (contenant les nœuds à explorer) et FERME (contenant les nœuds déjà explorés) contiennent les nœuds suivants :

Liste OUVERT	[	E ;	M ;	L]
heuristique (*h*) :		h(E) = 4	h(M) = 4	h(L) = 6
distance effective (*g*) :		g(E) = 1	g(M) = 2,4	g(L) = 2,4

Liste FERME	[	G ;	J]
heuristique (*h*) :		h(G) = 3	h(J) = 4

distance effective (g) : g(G) = 0 g(J) = 1

Pour chacun des nœuds explorés, la valeur de l'heuristique (h) et la valeur de la distance à l'origine (g) sont indiquées.

1) Quel est le contenu des listes OUVERT et FERME après l'exploration du meilleur nœud de OUVERT ? Il n'est pas nécessaire de donner les valeurs de h et g.
2) En observant le graphe, quels sont les nœuds qui ne seront pas explorés par l'algorithme A* (nœuds qui ne seront à aucun moment dans la liste OUVERT) ?