NOM : PRENOM :

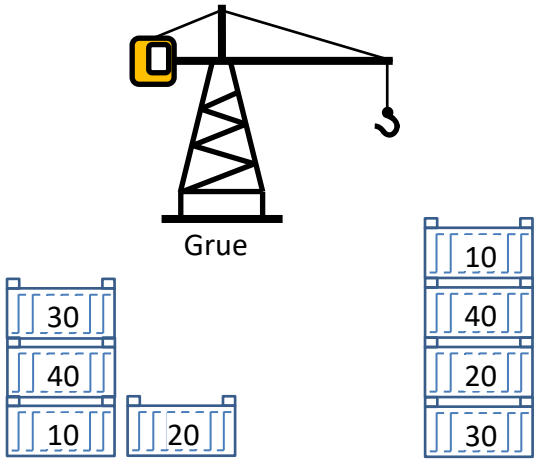
CONTROLE INTELLIGENCE ARTIFICIELLE (I4RSD11) - 15/03/2023

- Durée 1h15
- Documents non autorisés. Seule une **fiche de révision A4 recto-verso** est autorisée.
- Le sujet doit être rendu avec la copie, vous pouvez vous servir des schémas fournis.

EXERCICE 1 : ALGORITHME A* (9 PT)

Dans un port, quatre containers identiques mais de poids différents sont initialement disposés en 2 piles de 2 containers (fig.1 A). On désire réarranger ces containers en une seule pile (fig. 1 B).

Pour cela, on dispose d'une grue, qui peut soulever un container situé en haut d'une pile et **le placer au sol ou bien l'empiler au-dessus d'un autre**.
Chaque action est du type « *poser X sur Y* » avec
$X \in \{10, 20, 30, 40\}$ et $Y \in \{\text{sol}, 10, 20, 30, 40\}$.
L'aire de manutention est limitée et n'accepte que **3 piles au maximum** => les 4 containers ne peuvent jamais se trouver posés sur le sol simultanément.
Le **coût** de l'action « *poser X sur Y* » est en première approximation **égal** au **poids du container** *X*.
On veut trouver une séquence d'actions qui permet de passer de la situation A à la situation B en **minimisant le poids total des containers déplacés.**



A. Situation **initiale** B. Situation **finale**

Fig. 1

Pour définir la fonction *h*, on adopte d'abord les définitions suivantes.
Dans une situation *u* donnée, on distingue les containers *bien-placés* et ceux *mal-placés,* relativement à la situation finale B. Un container ***bien-placé*** est défini récursivement :
- soit en *u* il est posé au sol et il est aussi au sol en B ; sa position exacte au sol n'est pas importante.
- soit en *u* il repose sur un container *bien-placé*, et il repose sur le même container en B.

Un container ***mal-placé*** est un container qui n'est pas *bien-placé*. Pour pouvoir le déplacer il faut d'abord dépiler tous les containers situés au-dessus de lui. De ce fait, il faut repérer dans chaque pile le plus bas container *mal-placé* C, en lisant la pile de **de bas en haut** : C et tous les containers situés au-dessus de lui sont *mal-placés*. Dans ce problème, on choisit ***h*(*u*) = somme des poids de tous les containers mal-placés** sur l'ensemble des piles de *u*.

1.1 Fonctions *g* et *h*

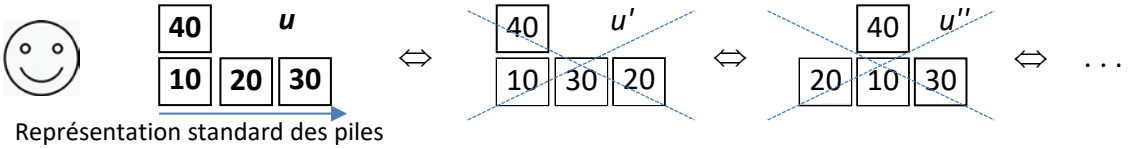
a. Rappelez la définition (itérative ou récursive) de la fonction ***g*(*u*)**. (0,5 pt)
b. Montrer, en détaillant votre calcul, que ***h*(A) = 100**. (0,5 pt)
c. Démontrer que *h* est ***coïncidente***. (0,5 pt)
d. Montrer que *h* est ***minorante***. (2,5 pt)

1.2. Tracer l'arbre des situations générées et développées par A* (page 3). (5 pt)

On respectera impérativement les conventions suivantes :
- on génèrera les successeurs d'un état en évaluant les actions réalisables "poser X sur Y" de la gauche vers la droite **par valeur croissante** de X, et pour un X donné, **par valeur Y croissante** {sol,10,20,30,40}.
- On numérotera les états u_i au fur et à mesure de leur apparition : u_0(=A), u_1, u_2, …. u_n (=B) et **on mentionnera** obligatoirement à côté de chaque situation u_i **les valeurs de *f*(u_i), *h*(u_i) et *g*(u_i)**.
- Deux nœuds contenant la même situation **ne doivent pas figurer dans l'arbre** : on évitera de créer un nœud s'il existe déjà un nœud contenant la même situation avec une évaluation égale ou inférieure.

NOM : PRENOM :

Pour diminuer l'espace de recherche de A*, **on ne générera pas des situations équivalentes, c-à-d contenant les mêmes piles à une permutation près. La position relative des piles n'a pas d'effet sur l'optimalité.** Pour éviter de générer de telles situations, on conviendra d'adopter une **représentation « *standard* », en ordonnant les piles par base croissante**, ex :



Représentation standard des piles (par base croissante)

fig. 2 : situations équivalentes (à proscrire)

EXERCICE 2 : ALGORITHME MINMAX/NEGAMAX ET COUPURES ALPHA-BETA (5 PT)

L'arbre donné ci-après (page 4) est un arbre de jeu de profondeur 5 coups. Les feuilles de cet arbre ont été évaluées en convention Minmax (selon le point du vue du premier joueur, celui qui joue à la racine).

2.1 Complétez le label Minmax des nœuds internes de l'arbre. Quel est le meilleur coup à la racine ?

2.2 Complétez (avec une autre couleur ou sur un côté différent) le label Negamax de tous les noeuds.

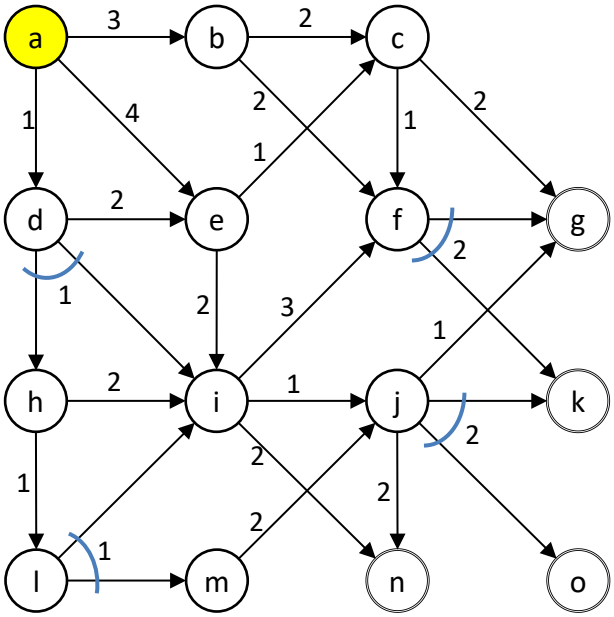
2.3 Si on applique l'algorithme Alpha-Beta à ce jeu, quelles sont les coupures réalisées ?

2.4 En arrivant à u_{41}, on constate qu'il s'agit de la même situation que celle du nœud u_1. Comment évaluer u_{41} ? Cela change-t-il la valeur du meilleur coup ? Cela change-t-il des coupures alpha-beta ?

EXERCICE 3 : ALGORITHME AO* (6 PT)

Appliquer l'algorithme AO* à l'hypergraphe ci-dessous associé à la décomposition du problème **a**.
Dessiner le sous-graphe solution de côut total minimal dans le cadre à droite.
Indiquer le coût de décomposition de ***a*** mais aussi celui de tous les sous-problèmes de la solution.
Les valeurs de l'heuristique estimant le coût de résolution sont données dans le tableau suivant :

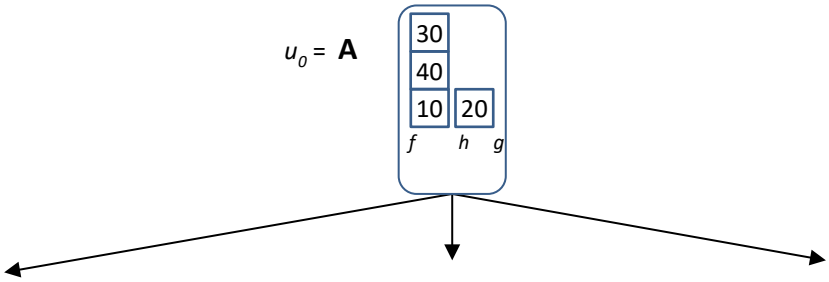
U	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
h(*U*)	5	3	2	4	2	2	0	3	1	1	0	3	2	0	0



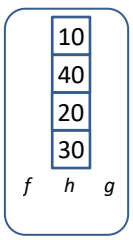
NOM : PRENOM :

ARBRE EXPLORE PAR A*

u_0 = **A**



B



NOM : PRENOM :

ARBRE MINMAX(profondeur 5 coups)