TP 2 : algorithme Minimax

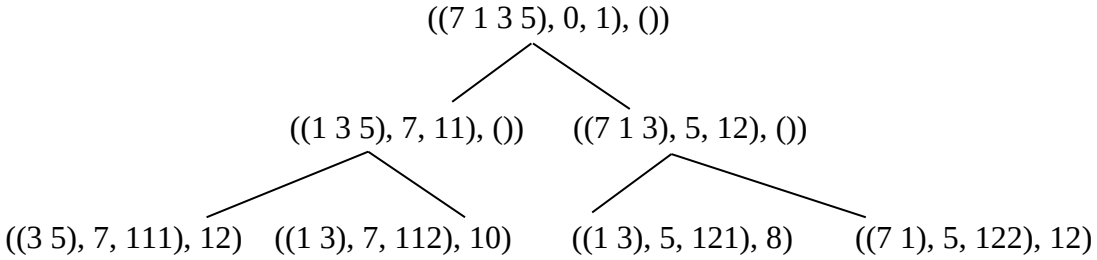
Exercice 3 :

On s'intéresse à un jeu de cartes à deux joueurs. *n* cartes (*n* pair) sont alignées sur une table, le jeu consiste pour chaque joueur à prendre une carte d'une extrémité. La prise d'une carte génère un gain pour le preneur correspondant à la valeur de la carte. Le jeu se termine lorsque toutes les cartes ont été prises. Chaque joueur calcule la somme de ses gains et annonce le résultat. Le vainqueur est celui qui obtient le plus fort résultat. On se propose d'évaluer une situation de jeu pour le joueur MAX en utilisant l'algorithme Minimax de l'exercice 2 et en considérant qu'un état est terminal lorsqu'il reste deux cartes sur la table. Pour cela il faut générer l'arbre de jeu correspondant puis appliquer l'algorithme à une profondeur égale à la profondeur de l'arbre généré.

Structure d'un nœud de l'arbre de jeu

noeud non valué	(*informations*)
nœud valué	(*informations valeur*)
informations	(*situation gains numéro*)
situation	(v_1 v_2 ...v_n), les vi sont les valeurs des cartes
gains	somme des prélèvements du joueur MAX (racine de l'arbre)
numéro	numéro du nœud, pour le kième fils d'un noeud *n* --> (numéro (*n*) * 10) + *k*

Exemple d'arbre de jeu :



Représentation de l'arbre ci-dessus :

((((7 1 3 5) 0 1)) ((((1 3 5) 7 11)) (((3 5) 7 111) 12) (((1 3) 7 112) 10)) ((((7 1 3) 5 12)) (((1 3) 5 121) 8) (((7 1) 5 122) 12)))

Représentation de l'arbre ci-dessus (indentée pour meilleure lisibilité) :

```
(
(((7 1 3 5) 0 1))
        (
        (((1 3 5) 7 11))
                (((3 5) 7 111) 12)
                (((1 3) 7 112) 10)
        )
        (
        (((7 1 3) 5 12))
                (((1 3) 5 121) 8)
                (((7 1) 5 122) 12)
        )
)
```

Questions

1. Définir les fonctions unaires **situation**, **gain** et **numero** qui s'appliquent à un nœud et retournent respectivement la situation du nœud, la somme des prélèvements du joueur MAX pour ce nœud et le numéro du nœud.

2. Définir la fonction unaire **genere_arbre** qui s'applique a un nœud *n* et construit l'arbre de jeu de racine *n*.

3. Donner la valeur minimax pour chaque exécution suivante en utilisant *genere_arbre* et la fonction *minimax* de l'exercice 2.

   nœud initial : (((7 1 2 3 5 8) 0 1))
   profondeur : 4

   noeud initial : (((7 1 2 4 6 3 5 8) 0 1))
   profondeur : 6

   nœud initial : (((9 1 3 6 2 7 4 3 5 8) 0 1))
   profondeur : 8

Pour définir vos fonctions, **vous devez utiliser** les fonctions définies dans les précédents exercices et les fonctions suivantes définies dans le fichier *JeuEtud.txt.*

car(<liste>) : retourne le premier élément de la liste passée en argument.
der(<liste>) : retourne le dernier élément de la liste passée en argument.

cdr(<liste>) : retourne la liste passée en argument privée de son premier élément.
tsd (<liste>) : retourne la liste passée en argument privée de son dernier élément.

cons(<arg> <liste>) : retourne la liste passée en second argument augmentée du premier argument.
sup(<arg> <liste>) : retourne la liste passée en second argument privée du premier argument.

etat_terminal?(<nœud>) : retourne vrai si le nœud passé en argument correspond à un état terminal.
heuristique(nœud>) : retourne la valeur heuristique du nœud passé en argument.

nœuds_fils(<nœud>) : retourne la liste des nœuds fils du nœud passé en argument (commence toujours par la prise de la carte de gauche).