Algorithme Min-Max pour le calcul du gain

On considère un graphe biparti $G = (S, \mathcal{A})$ associé à un jeu à deux joueurs A et B.

L'ensemble S des n sommets est une union disjointe $S = S_A \cup S_B$. Les joueurs jouent alternativement : les sommets $s \in S_A$ correspondent aux états où c'est au tour du joueur A de jouer. Une partie s'arrête lorsqu'un joueur se trouve sur un sommet sans successeur. L'ensemble $\mathcal{A}$ des m arêtes vérifie $\mathcal{A} \subset (S_A \times S_B) \cup (S_B \times S_A)$. On note $s \to t$ pour signifier $(s,t) \in \mathcal{A}$.

On note T l'ensemble des positions terminales, c'est-à-dire les sommets s sans successeur. Une partie s'arrête lorsqu'elle aboutit à une position terminale.

A chaque position finale s est associé un entier $e(s)$ représentant le gain obtenu par le joueur A associé à cette position finale. On considère que le jeu est à somme nulle (le gain obtenu par le joueur B est l'opposé du gain obtenu par le joueur A), c'est-à-dire que le gain $e(s)$ est un entier strictement positif si le joueur A a gagné et $e(s)$ est un entier strictement positif si le joueur B a gagné, et $e(s)$ vaut 0 en cas de partie nulle.

A chaque étape, chaque joueur joue au mieux : le joueur A choisit le sommet maximisant son gain, et le joueur B choisit le sommet minimisant le gain du joueur A.

On suppose ici le graphe **acyclique** : autrement dit, toute partie est finie.

Le graphe G est codé par **le dictionnaire des listes d'adjacence** et **la partition est stockée par un dictionnaire** P dont les clés sont les sommets $s \in S$ et dont la valeur $P[s]$ vaut 0 si $s \in S_A$ et $P[s]$ vaut 1 si $s \in S_B$.

Les gains sont codés par un **dictionnaire** E dont les clés sont des sommets et dont la valeur d'une clé s est le gain $e(s)$ associé à la position s. Au départ, on connaît uniquement les gains associés aux positions finales.

1) Définir une fonction `max(L)` qui renvoie la valeur maximale d'une liste L non vide d'entiers.

On écrit de même une fonction `min(L)` qui pourra être utilisée dans la suite.

2) Évaluation du gain par l'algorithme min-max par un parcours en profondeur

On suppose connu initialement le dictionnaire E dont les clés sont les sommets terminaux $s \in T$ et dont les valeurs sont les gains connus $e(s)$ associés aux positions finales.

On définit par induction la fonction e pour tout sommet $s \in S \setminus T$ par :

$$e(s) = \begin{cases} \max\{e(t),\, t \in G[s]\} \text{ si } s \in S_A \\ \min\{e(t),\, t \in G[s]\} \text{ si } s \in S_B \end{cases}$$

Écrire une fonction `Gain(G,P,E,s0)` qui prend comme arguments

- les dictionnaires G et P codant respectivement le graphe et la partition

- le dictionnaire E définissant les gains uniquement pour les sommets terminaux

- un sommet initial s_0 (c'est-à-dire une clé de G)

complète le dictionnaire E codant les gains des descendants de $s_0 \in S$ (s_0 inclus), et renvoie $e(s_0)$.

On utilisera une fonction **auxiliaire récursive** `traite(s)` qui définit $E[s]$ si ce n'est pas déjà le cas.

3) Écrire une **version itérative avec une pile** basée sur le principe suivant :

- la pile est intialement défini à $[s_0]$
- tant que la pile n'est pas vide :
    - on récupère le premier élément de la pile, mais sans le dépiler, avec l'instruction : `s = pile.head()`
    - si $E[s]$ est déjà défini, on dépile s de la pile
    - sinon :
        - on ajoute à la pile les successeurs de s **qui ne sont pas déjà** des clés du dictionnaire E
        - dans le cas où aucun successeur de s n'a été ajouté à la pile à l'étape précédente :

            on calcule $E[s]$ et on dépile s de la pile