I. Implémentation sous Python et premiers exercices

Nous implémentons les piles à partir des listes Python dans la question 1.

Ensuite, on agit sur les piles qu'en utilisant les fonctions propres à cette structure de données programmées dans la question 1. L'utilisation des listes autrement que pour simuler les piles est interdite dans ce premier exercice.

1. Créer les fonctions suivantes : pile_vide(), empiler(), depiler(), est_vide(), nombre_elements(), lire_sommet(), vider_pile(), afficher_pile().
2. Ecrire une fonction depilerK(*p*,*k*) permettant de dépiler *k* éléments d'une pile *p* sans les renvoyer (ou tous les éléments si la pile en contient moins de *k*).
3. Ecrire une fonction estDansLaPile(*p*,*x*) qui renvoie True si la pile contient l'élément *x*, False sinon, sans modifier la pile.

II. Notation polonaise inversée (Reverse Polish Notation)

La notation polonaise inversée (RPN) permet d'écrire des expressions arithmétiques sans utiliser de parenthèses. Elle se différencie de la notation polonaise par l'ordre des termes, les opérandes y étant présentés avant les opérateurs et non l'inverse.

Exemples : "(3 + 1) × (2 + 4)" s'écrit "3 1 + 2 4 + ×"
"5 × (2 + 3)" s'écrit "2 3 + 5 ×" ou "5 2 3 + ×"

Ecrire un interpréteur d'expressions en RPN.
L'expression est une chaine de caractères contenant des nombres et des opérateurs séparés par des espaces. L'interpréteur lit la chaine de gauche à droite, stocke les nombres dans une pile, et lorsqu'il rencontre un opérateur, récupère les deux derniers nombres de la pile, effectue l'opération et stocke le résultat dans la pile. L'interpréteur renvoie le résultat de l'expression si celle-ci est correcte, un message d'erreur dans le cas contraire.

On pourra utiliser les méthodes de chaines de caractères split et isdigit.

chaine.split() découpe la chaine de caractère en liste de chaines (en utilisant par défaut l'espace comme caractère de séparation).

In [1]: 'vive les piles !'.split() → Out[1]: ['vive','les','piles','!']

chaine.isdigit() renvoie True si la chaine représente un nombre, False sinon.

III. Un algorithme de recherche en profondeur

Une matrice labyrinthe est une matrice dont deux coefficients sont des 2 et tous les autres des 0 ou des 1.

Deux coefficients a_{ij} et a_{kl} d'une telle matrice sont dit adjacents si :

$$|j - l| + |i - k| = 1$$

On appelle chemin toute suite de coefficients différents de 1 adjacents, les coefficients étant identifiés par leurs indices de ligne et de colonne. Un chemin solution est un chemin dont le premier terme est un 2 et le dernier est l'autre 2.

Exemples :

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 2 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 2 & 1 \end{pmatrix}$$

La matrice de gauche admet un chemin solution, pas celle de droite.

Ecrire une fonction solution, qui prend en paramètre une matrice labyrinthe, représentée par une liste de listes, et recherche un chemin solution.
Cette fonction renverra la matrice en notant le chemin solution avec des 3 à la place des 0 s'il en existe un. Sinon, elle renverra un message.

L'utilisation d'une pile pour stocker le chemin en cours permettra de revenir en arrière en dépilant lorsqu'on est bloqué. Un élément de la pile sera un tuple contenant les indices des coefficients (ligne et colonne). Afin de ne pas revisiter les mêmes cases, on remplacera les 0 par des 8 par exemple dans les cases déjà visitées ne faisant pas partie du chemin solution.

Exercice supplémentaire : Programmer une solution par récursivité. Dans ce cas, le programmeur n'a pas à se soucier de la gestion de la pile.