DM - Algorithmique - Master 1 Bio-informatique

2022 - 2023

Rev. 1

Exercice 1

Donnez la définition d'une file. Illustrez votre définition par un exemple.

Exercice 2

La notation polonaise inverse est une notation particulière qui permet de réaliser des calculs mathématiques sans avoir recours à des parenthèses. Il consiste à écrire les opérations en commençant par donner les opérandes d'un calcul avant de donner l'opérateur.

Par exemple, pour faire 3 + 5, on écrit en notation polonaise inverse :

$$3 \;\; 5 \;\; +\,.$$

Ainsi, une séquence de nombres et d'opérateurs se lie de gauche à droite. À chaque fois qu'on lit un opérateur +, -, *, /, que l'on notera X, on remplace, dans la séquence, l'opérateur X et les deux derniers nombres a et b qui le précède par le résultat du calcul de aXb .

Par exemple, si l'on veut calculer l'expression suivante :

$$((3+5)*(6-2))+4*(5-(7+2))$$

il suffit d'écrire :

$$3 \;\; 5 \;\; + \;\; 6 \;\; 2 \;\; - \;\; * \;\; 4 \;\; 5 \;\; 7 \;\; 2 \;\; + \;\; - \;\; * \;\; +.$$

Le calcul s'effectuera ainsi :

$$\begin{array}{c} 3 \;\; 5 \;\; + \;\; 6 \;\; 2 \;\; - \;\; * \;\; 4 \;\; 5 \;\; 7 \;\; 2 \;\; + \;\; - \;\; * \;\; + \\ \rightarrow \\ 8 \;\; 6 \;\; 2 \;\; - \;\; * \;\; 4 \;\; 5 \;\; 7 \;\; 2 \;\; + \;\; - \;\; * \;\; + \\ \rightarrow \\ 8 \;\; 4 \;\; * \;\; 4 \;\; 5 \;\; 7 \;\; 2 \;\; + \;\; - \;\; * \;\; + \\ \rightarrow \\ 32 \;\; 4 \;\; 5 \;\; 7 \;\; 2 \;\; + \;\; - \;\; * \;\; + \\ \rightarrow \\ 32 \;\; 4 \;\; 5 \;\; 9 \;\; - \;\; * \;\; + \\ \rightarrow \\ 32 \;\; 4 \;\; -4 \;\; * \;\; + \\ \rightarrow \\ 32 \;\; -16 \;\; + \\ \rightarrow \\ 16. \end{array}$$

La notation polonaise inverse, pour un ordinateur, permet de réaliser un calcul sans avoir à implémenter une grammaire qui détecte les parenthèses ouvrantes et les parenthèses fermantes. En effet, une utilisation judicieuse des piles permet de résoudre le problème. Cette notation était donc utilisée dans les vielles calculatrices, comme la HP48G : `https://www.hpcalc.org/details/3937`.

Proposez un programme `calcul_polonaise_inverse(p)` qui prends une pile contenant des nombres et des opérateurs arithmétiques ('+', '*', '-', '/') et qui renvoie le résultat du calcul de la pile p lorsque p est lue de haut en bas et qu'elle est en notation polonaise inverse. Si l'expression n'est pas un calcul valide, la fonction renvoie alors None.

Pour cela, vous devez utiliser uniquement les piles données à l'aide des primitives suivantes :

```
def creer_pile():
  # retourne une pile vide.

def empiler( p, e ):
  # empile dans la pile p l'élément e.

def depiler( p ):
  # la fonction dépile l'élément situé en haut de la pile p
  # et le renvoie.

def taille( p ):
  # renvoie le nombre d'éléments contenus dans la pile p.
```

Dans cet exercice, il n'est pas autorisé d'utiliser des listes, tuples et dictionnaires du langage python.

Exercice 3

Proposez une implémentation des dictionnaires en utilisant uniquement une partie des primitives des piles précédentes qui sont les suivantes :

```
def creer_pile()
def empiler( P, e )
def depiler( P )
```

Vous noterez que cette fois-ci, la primitive `taille_pile()` n'est plus disponible.

Pour implémenter le dictionnaire, vous implémenterez les primitives suivantes :

```
def creer_dictionnaire():
    # Créé et renvoie un nouveau dictionnaire
def inserer_association( D, cle, valeur ):
    # Ajoute une nouvelle association ('cle','valeur') dans le
    # dictionnaire D.
    # S'il existait déjà une association ('cle', valeur1) dans D,
    # alors cette association est remplacée par ('cle','valeur').
def supprimer_association( D, cle ):
    # Supprime l'assocation de D qui à pour cle 'cle'.
def appartient_au_dictionnaire( D, cle ):
    # Renvoie Vraie si 'cle' est une clé du dictionnaire D.
```

```
def obtenir_valeur( D, cle ):
    # Renvoie la valeur associée à la clé dans le dictionnaire D.
```

Exercice 4

Le but de cet exercice est d'écrire un programme récursif qui renvoie toutes les partitions de n. Une partition de n est une liste d'entier ordonné du plus grand au plus petit telle que la somme des entiers fassent n.

Par exemple, les partitions de 5 sont :

$$[5], [4, 1], [3, 2], [3, 1, 1], [2, 2, 1], [2, 1, 1, 1], [1, 1, 1, 1, 1].$$

En fait, on peut remarquer que si l'on prend une partition de n et que l'on retire le premier élément a de cette partition, alors on obtient une partition de $n - a$.

Plus précisément, si je retire le premier entier de chaque partition de 5, j'obtiens :

$$[], [1], [2], [1, 1], [2, 1], [1, 1, 1], [1, 1, 1, 1],$$

qui sont respectivement :

- pour $a = 5$, toutes les partitions de $5 - a = 0$ dont les éléments sont plus petits que l'entier 5 retiré : [] ;
- pour $a = 4$, toutes les partitions de $5 - a = 1$ dont les éléments sont plus petits que l'entier 4 retiré : [1] ;
- pour $a = 3$, toutes les partitions de $5 - a = 2$ dont les éléments sont plus petits que l'entier 3 retiré : $[2], [1, 1]$;
- pour $a = 2$, toutes les partitions de $5 - a = 3$ dont les éléments sont plus petits que l'entier 2 retiré : $[2, 1], [1, 1, 1]$;
- pour $a = 1$, toutes les partitions de $5 - a = 4$ dont les éléments sont plus petits que l'entier 1 retiré : $[1, 1, 1, 1, 1]$.

1. Proposez une fonction récursives $partitions\_bornee(n, b)$ qui prends en paramètre deux entiers n et b et qui renvoie toutes les partitions de n dont les éléments sont plus petits que b.
2. Proposez une fonction $partitions(n)$ qui prends en paramètre un entier n et qui renvoie une liste de toutes les partitions de n.

Exercice 5

Dans cet exercice on suppose que vous disposez d'une bibliothèque contenant l'API des listes simplement chaînées suivante :

```
def create_list():
    """
    Créer et retourne une liste vide.
    """
def create_cell(value, successeur):
    """
    Créer et retourne une cellule contenant la valeur
```

```
    'valeur' et le successeur 'successeur'.
    """
def change_cell(cell, valeur, successeur):
    """
    Change les attributs 'valeur' et 'successeur' de la cellule
    'cell' passée en paramètre.
    """
def push_front(l, e):
    """
    Ajoute l'élément 'e' au début de la liste 'l'.
    Cette fonction créé une nouvelle cellule et l'insère en
    début de liste.
    """
def remove_front(l):
    """
    Retire le premier élément de la liste si il existe.
    """
def first_cell( l ):
    """
    Retourne la première cellule de la liste 'l' et None
    si la liste est vide.
    """
def next_cell( cell ):
    """
    Retourne le successeur de la cellule 'cell'.
    """
def value_cell( cell ):
    """
    retourne la valeur de la cellule 'cell'.
    """
```

Proposez un programme *tri_selection*(*liste*) qui prend en paramètre une liste simplement chaînée contenant des entiers et qui la trie à l'aide du tri sélection.

Dans ce programme il est interdit d'utiliser les listes python et tout autre structure de donnée native de Python telle que les dictionnaires et les tuples. Seule l'API des listes simplement chaînées doit être utilisée.