DM - Algorithmique - Master 1 Bio-informatique

2021 - 2022

Rev. ~~1~~3

Exercice 1

Proposez un algorithme qui prend en paramètre une liste Python d'entiers l et qui renvoie vrai si les entiers paires sont triés par ordre croissant alors que les entiers impaires sont triés par ordre décroissant.

Déterminer la complexité dans le pire cas de votre algorithme.

Exercice 2

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

Exercice 3

Dans cet exercice, 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 ):
  # 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.

1. Proposez un algorithme $trier(pile)$ qui prend en paramètre une pile et qui la trie par ordre croissant, de haut en bas.
2. Proposez un algorithme $swap(p, k, n)$ qui prend en paramètre une pile p et deux entiers k et n et qui, parmi les n premiers éléments de la pile, permute les k premiers éléments avec les $n-k$ éléments suivant. Par exemple, si une pile p contient les éléments suivants : $[1, 2, 3, 4, 5, 6, 7, 8, 9]$, alors, après l'exécution de $swap(p, 3, 7)$, cette même pile sera égale à : $[4, 5, 6, 7, 1, 2, 3, 8, 9]$.

Exercice 4

Le but de cet exercice est d'écrire un programme récursif qui renvoie tous les mots bien parenthésés de taille n. Un mot w bien parenthésé est un mot contenant uniquement des parenthèses ouvrantes "(" et des parenthèses fermantes ")" qui

— contient autant de parenthèses ouvrantes que de parenthèses fermantes,

— pour tout préfixe u de w, le mot u contient ~~strictement~~ plus de parenthèses ouvrantes que de parenthèses fermantes.

Par exemple, (())() est bien parenthésé car il y a au total 3 parenthèses ouvrantes et fermantes. De plus, les préfixes de ce mot, à savoir : (, ((, (()), (()), (())(et (())(), contiennent toujours plus de parenthèses ouvrantes que de parenthèses fermantes.

1. Énumérez, à la main, les mots bien parenthésés de taille 1, 2, 3, 4, 5 et 6.
2. Proposez un programme $est\_bien\_parenthese(w)$ qui prend en paramètre un mot w et qui renvoie Vrai si le mot est bien parenthésé.
3. Un mot bien parenthésé w est soit le mot vide (w=""), soit un mot qui peut toujours s'écrire ~~, soit sous la forme uv, soit~~ sous la forme $(u)v$ où u et v sont deux mots bien parenthésés. Proposez un programme récursif qui prend en paramètre un entier n et qui renvoie la liste de tous les mots bien parenthésés de taille n.
4. (difficile et optionnel) Pourquoi l'algorithme proposé dans l'item précédent construit tous les mots bien parenthésés ?
5. (difficile) En utilisant éventuellement une pile, proposez une nouvelle version, non récursive, de ce même programme.

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_bulle*(*liste*) qui prend en paramètre une liste simplement chaînée contenant des entiers et qui la trie à l'aide du tri à bulle.

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.