DM - Algorithmique - Master 1 Bio-informatique

2025 - 2026

Rev. 1

1 Organisation de la relecture de vos travaux

La relecture et l'évaluation des vos traveaux suivront les règles suivantes :

1. le projet peut être fait en groupe, il n'y a pas de limite de taille de groupe ;
2. vous pouvez m'envoyer autant de version de votre travail à relire (que je peux humainement relire) ;.
3. pour chaque version que vous m'envoyez, je doit vous envoyer :
   - une critique détaillée (revue de code et commentaire) de votre travail ;
   - une note temporaire sur 20 de votre travail ;
4. la critique envoyée doit contenir des suggestions de corrections ou améliorations qui doit vous permettre, si corrigée, d'obtenir une notre d'au moins 20/20 ;
5. la note temporaire ne peut qu'augmenter. Même si vous introduisez des erreurs dans une nouvelle version, la note ne peut pas diminuer ;
6. la note finale sera égale au max entre 20/20 et la dernière note temporaire obtenue par la dernière version que vous avez envoyée ;
7. vous pouvez vous inspirer de n'importe quel code source éxistant, à condition de citer vos sources ;
8. vous pouvez réutiliser du code source existant à condition qu'il soit sous license libre et que vous citez les auteurs ;
9. l'utilisation des IA génératives n'est pas interdit. Si vous utilisez une IA générative, il est impératif de la citer, de donner les requêtes réalisées et leurs réponses, puis de justifier la validité et le bon fonctionnement des réponses obtenus et enfin de justifier leurs intégrations dans le code.
10. une date de fin d'envoi sera discutée en classe. Il s'agit d'une date à partir de laquelle, le processus de relecture/notation ne sera plus garantie.

2 Exercices

Exercice 1

Dessiner la mémoire des programmes 12, 13, 14, 15, 16, 17 et 18 de l'exercice 5 du cours.

N'hésitez pas à juste prendre une photo d'un document écrit à la main sur feuille de papier.

Exercice 2

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 3

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 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 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 et optionnel) En utilisant éventuellement une pile, proposez une nouvelle version, non récursive, de ce même programme.

Exercice 4

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.