

Algorithmique

Partiel n° 1 (P1)

INFO-SUP (s1)
EPITA

4 Jan. 2016 - 10 :00 (D.S. 307009.33 BW)

Consignes (à lire) :

- Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
 - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
 - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
 - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
 - Aucune réponse au crayon de papier ne sera corrigée.
 - La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
 - Le code :**
 - Tout code doit être écrit dans le langage Python (pas de C, CAML, ALGO ou autre).
 - **Tout code Python non indenté ne sera pas corrigé.**
 - Tout ce dont vous avez besoin (fonctions, méthodes) est indiqué en **annexe** !
 - Durée : 2h00
-



Des arbres

Exercice 1 (Un peu de cours... – 5 points)

Soit l'arbre $B = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 010, 011, 101, 110, 111, 0001, 0101\}$.

1. Représenter graphiquement l'arbre B en donnant comme étiquette aux noeuds leur numéro d'ordre hiérarchique.
2. Quels sont (en ordre hiérarchique) les noeuds externes de l'arbre B ?
3. Quelle est la longueur de cheminement interne de l'arbre B ?
4. Quelle est la profondeur moyenne externe de l'arbre B ?
5. Quelle est la particularité d'un arbre localement complet ?

Des listes

Exercice 2 (Type abstrait → Python – 2 points)

Soit l'opération *change* définie pour le type abstrait *Liste itérative* de la manière suivante :

OPÉRATIONS

$change : Liste \times \text{Élément} \times \text{Élément} \rightarrow Liste$

AXIOMES

$longueur(change(\lambda, x, y)) = longueur(\lambda)$

$1 \leq i \leq longueur(\lambda) \ \& \ x = i\text{ème}(\lambda, i) \Rightarrow i\text{ème}(change(\lambda, x, y), i) = y$

$1 \leq i \leq longueur(\lambda) \ \& \ x \neq i\text{ème}(\lambda, i) \Rightarrow i\text{ème}(change(\lambda, x, y), i) = i\text{ème}(\lambda, i)$

AVEC

$\lambda : Liste$

$i : Entier$

$x, y : \text{Élément}$

Implémenter cette opération à l'aide d'une fonction **Python**. La fonction doit modifier la liste en place (la liste en paramètre est modifiée, aucune autre liste ne doit être créée).

Exercice 3 (ALGO → Python – 4 points)

Soit la fonction **test**, qui utilise les opérations du type abstrait *Liste itérative* :

```
fonction test(Liste L) : boolean
variables
    entier i
    boolean b
debut
    b ← vrai
    i ← 1
    tant que i < longueur(L) faire
        si ième(L, i) > ième(L, i+1) alors
            b ← faux
        fin si
        i ← i + 1
    fin tant que
    retourne b
fin
```

1. Que fait la fonction **test** ?
2. Écrire une version **Python** de cette fonction, si possible plus optimale que la version **ALGO** présentée ci-dessus.

Exercice 4 (Tri fusion – 2,5 + 5 + 2,5 points)

1. Écrire la fonction `partition` qui sépare une liste en deux listes de longueurs quasi identiques (à 1 près) : une moitié dans chaque liste.

Exemples d'application :

```
1 >>> partition([15, 2, 0, 4, 5, 8, 2, 3, 12, 25])
2 ([15, 2, 0, 4, 5], [8, 2, 3, 12, 25])
3 >>> partition([5,3,2,8,7,1,5,4,0,6,1])
4 ([5, 3, 2, 8, 7], [1, 5, 4, 0, 6, 1])
```

2. Écrire la fonction `merge` qui fusionne deux listes triées en ordre croissant en une seule liste triée.

Exemple d'application :

```
1 >>> merge([1,5,8], [2,3,4,8])
2 [1, 2, 3, 4, 5, 8, 8]
```

3. Pour trier une liste L, on procède (récursivement) de la façon suivante :

- ▷ Une liste de longueur < 2 est triée.
- ▷ Une liste de longueur ≥ 2 :
 - on partitionne la liste L en deux sous-listes L1 et L2 de longueurs quasi identiques (à 1 près) ;
 - on trie récursivement les deux listes L1 et L2 ;
 - enfin, on fusionne les listes L1 et L2 en une liste triée.

Utiliser les deux fonctions précédentes (quelles soient écrites ou non) pour écrire la fonction `mergesort` qui trie en ordre croissant une liste (pas en place : la fonction construit une nouvelle liste qu'elle retourne).

Exemple d'application :

```
1 >>> mergesort([5,3,2,8,7,1,5,4,0,6,1])
2 [0, 1, 1, 2, 3, 4, 5, 5, 6, 7, 8]
```

Annexe : Fonctions et méthodes autorisées

Vous pouvez utiliser la méthode `append` et la fonction `len` sur les listes :

```
1 >>> help(list.append)
2 Help on method_descriptor:   append(...)
3     L.append(object) -> None -- append object to end of L
4
5 >>> help(len)
6 Help on built-in function len in module builtins:   len(...)
7     len(object)
8     Return the number of items of a sequence or collection.
```

Vous pouvez également utiliser la fonction `range`. Rappels :

```
1 >>> for i in range(10):
2     ...     print(i, end=' ')
3     0 1 2 3 4 5 6 7 8 9
4
5 >>> for i in range(5, 10):
6     ...     print(i, end=' ')
7     5 6 7 8 9
```