Entraînement : concevoir une fonction récursive

Les solutions peuvent être rédigées en **pseudo-code, python, C, C++ ou Java**. La syntaxe du langage n'a pas d'importance tant que celle-ci reste **cohérente** et **compréhensible**. (Dans les exemples, les solutions sont données en pseudo-code).

Niveau 1

Grille d'évaluation

A (20)	L'algorithme est récursif et répond au problème donné. La complexité calculée est correcte (cohérente avec l'algorithme).
B (16)	La partie récursive de l'algorithme est correcte mais il y des erreurs sur les cas d'arrêts ou cas particuliers OU sur la complexité calculée.
C (11)	La partie récursive de l'algorithme est correcte mais il y a des erreurs sur les cas d'arrêts ou cas particuliers ET sur la complexité.
D (8)	Erreur sur la partie récursive de l'algorithme ou cas d'arrêt complètement absents.
E (1)	Algorithme non récursif ou récursif sans cas d'arrêts et faux.

Exercice 1.
Les coefficients binomiaux $\binom{n}{k}$ sont des nombres mathématiques qui donnent le nombre de façon de choisir k objets parmi n. Ils vérifient en particulier les propriétés suivantes :

- $\binom{n}{n} = \binom{n}{0} = 1$,
- si $k < 0$ ou $k > n$, $\binom{n}{k} = 0$,
- Pour $n > 0$, $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

(1) Donner un algorithme **récursif** qui calcule le coefficient binomial $\binom{n}{k}$. Votre fonction devra retourner un résultat (et donc terminer) quels que soient les entiers n et k passés en paramètre.

(2) Quelle est la complexité de votre algorithme ?

Solution

```
Binomimal
Input : deux entiers n et k
Procédé :
    Si n = k ou k = 0 :
        Retourner 1
    Si k < 0 ou k > n:
        Retourner 0
    Retourner Binomal(n-1,k-1) + Binomial(n-1, k)
```

Complexité $O(2^n)$.

Exercice 2 (Partiel 2017-2018)**.**

(1) Donner un algorithme **récursif** qui calcule le produit $a \times b$ de deux entiers donnés en utilisant **uniquement** des additions ou soustractions. Votre algorithme doit donner un résultat correct pour tous les entiers positifs ou négatifs.

(2) Donner la complexité de votre algorithme.

Solution
(Une solution possible)

```
Produit
Input : deux entiers a et b
Procédé :
    Si a = 0 ou b = 0 :
        Retourner 0
    Si a < 0 :
        Retourner -Produit(-a,b)
    Si b < 0 :
        Retourner -Produit(a,-b)
    Si a > b :
        Retourner Produit(b,a)
    Retourner b+Produit(a-1,b)
```

Complexité $O(a)$.
Question donnée au partiel 1 2017-2018, résultats obtenus :

A	B	C	D	E
47.6%	42.9%	9.5%	0%	0%

Exercice 3 (Partiel 2018-2019)**.**

(1) Donner un algorithme **récursif** qui calcule le nombre de chiffre (en base 10) d'un nombre entier positif. On considère que 0 s'écrit avec un chiffre. Par exemple, pour 16546, on renvoie 5.

(2) Donner la complexité de votre algorithme.

Solution
(Une solution possible)

```
Chiffres
Input : un entier n positif
Procédé :
    Si n < 10 :
        renvoyer 1
    renvoyer 1 + Chiffres(n/10)
```

Complexité $O(\log(n))$.
Question donnée au partiel 1 2018-2019, résultats obtenus :

A	B	C	D	E
45%	50%	5%	0%	0%

Niveau 2 (bonus en partiel)

Grille d'évaluation

A (20)	L'algorithme est récursif et répond au problème donné.
B (16)	Le principe récursif est en partie correct mais il y a des erreurs dans son implantation et / ou dans les cas particuliers et conditions d'arrêt.
C (11)	Un algorithme récursif est proposé qui tente de répondre à la question posée même s'il ne répond pas aux spécifications du problème.

Exercice 4.
Écrire une fonction qui prend en paramètre une chaîne de caractère ainsi qu'un entier k et retourne la liste de toutes les façon de choisir k lettres dans la chaîne.

Par exemple, si je lance ma fonction sur `"chat"` et 2, elle devra renvoyer : `[ "ch", "ca", "ct", "ha", "ht", "at"]`. Remarquez que l'ordre des des lettres reste le même que dans la chaîne originale. L'ordre des mots dans la liste n'a pas d'importance. La fonction ne s'occupera pas de supprimer les éventuels doublons et on retournera donc toujours une liste de taille $\binom{n}{k}$ où n est la taille de la chaîne.

On pourra utiliser la syntaxe python pour les listes et les chaînes de caractères :

Liste vide : `L <- [ ]`

Ajout : `L.append(valeur)`

Première lettre d'un mot : `u[0]`

Dernière lettre d'un mot : `u[-1]`

Concaténation de deux chaînes de caractère : `u + v`

La chaîne de caractère obtenue en copiant les lettres de `u` à partir de l'indice 1 : `u[1 :]`

La chaîne de caractère obtenue en copiant les lettres de `u` sauf la dernière : `u[ :-1]`

Solution

```
KParmiN
Input : une chaine de caractère s et un entier k
Procédé :
    Si k = 0 :
        Retourner [""] # liste contenant le mot vide
    Si s = "" :
        Retourner [] # liste vide (pas de résultat)
    resultat = []
    Pour e dans KParmiN(s[1 :], k-1) :
        resultat.append(s[0] + e)
    Pour e dans KParmiN(s[1 :], k) :
        resultat.append(e)
    Retourner resultat
```

Exercice 5 (Partiel 2017-18)**.**

Le *shuffle* entre deux mots u et v est la liste des mots obtenus en mélangeant entre elles les lettres de u et v tout en conservant l'ordre des lettres des mots de départ. Par exemple, les mots du shuffle de `"bob"` et `"cat"` sont exactement ceux de la liste suivante

bobcat bocbat bocabt bocatb bcobat bcoabt bcoatb bcaobt bcaotb bcatob cbobat cboabt cboatb cbaobt cbaotb cbatob cabobt cabotb cabtob catbob.

Remarquez que les lettres de "bob" et "cat" sont toujours dans le même ordre. Ainsi, le mot "tbobca" n'appartient pas au shuffle car les lettres de "cat" ne sont plus dans le bon ordre.

Voici quelques propriétés du shuffle :

— Si l'un des deux mots est vide, le shuffle ne contient qu'un seul élément : l'autre mot. (Si les deux sont vides, il ne contient que le mot vide "").

— Sinon, un mot du shuffle commence soit par la première lettre de u, soit par la première lettre de v, la suite est donnée par le shuffle des lettres restantes.

Implanter un algorithme récursif qui prend en paramètres deux mots `u` et `v` et retourne la liste des mots de leur shuffle.

On pourra utiliser la syntaxe python pour les listes et les chaînes de caractères.

Liste vide : `L <- [ ]`

Ajout : `L.append(valeur)`

Première lettre d'un mot : `u[0]`

Concaténation de deux chaînes de caractère : `u + v`

Le chaîne de caractère obtenue en copiant les lettres de `u` à partir de l'indice 1 : `u[1 :]`

Solution

```
Shuffle
Input : deux chaines de caractères u et v
Procédé :
    Si u = "":
        Retourner [v]
    Si v = "":
        Retourner [u]
    resultat <- []
    Pour e dans Shuffle(u[1:], v):
        resultat.append(u[0] + e)
    Pour e dans Shuffle(u,v[1:]):
        resultat.append(v[0] + e)
    Retourner resultat
```

Question donnée au partiel 1 2017-2018, résultats obtenus :

A	B	C	Non répondu
9.5%	9.5%	14.3%	66.7%

Exercice 6 (Partiel 2018-2019)**.**
Donner un algorithme qui prend en paramètre un tableau de taille n contenant des valeurs distinctes et renvoie les 2^n sous ensemble possible. Par exemple, si on donne en entrée : `[1,2,3]` , l'algorithme renverra `[ [1,2,3], [1,2], [1,3], [1], [2,3], [2], [3], [] ]` . Dans chaque tableau, les nombres seront dans l'ordre initial, par contre, l'ordre des sous-ensemble dans le résultat n'a pas d'importance.

Remarque : pour un tableau vide, on renvoie le tableau contenant l'unique sous-ensemble vide donc `[ [] ]` .

(Vous pouvez utiliser la syntaxe que vous voulez pour tronquer / concaténer des tableaux).

Solution

```
SousEnsemble
Input : un tableau t de taille n
Procédé :
    Si n = 0:
       Renvoyer [[]]
    Resultat <- []
    V <- SousEnsemble(t[1:])
    Pour e dans v:
        Resultat.append([t[0]] + e)
    Resultat.extend(V)
    Renvoyer Resultat
```

Question donnée au partiel 1 2018-2019, résultats obtenus :

A	B	C	Non répondu
5%	18%	50%	27%