TD 6 - Récursivité

Objectif : Savoir concevoir / dérécursiver / analyser un algorithme récursif.

Exercice 1 - Soit A un ensemble fini et A^* l'ensemble des mots formés sur A. On donne les fonctions de base suivantes :

ε : le mot vide. C'est un élément de A^*
premier : $A^* \setminus \{\varepsilon\} \to A$ renvoie le premier élément d'un mot. premier(example) = e
dernier : $A^* \setminus \{\varepsilon\} \to A$ renvoie le dernier élément d'un mot. dernier(example) = e
saufpremier : $A^* \setminus \{\varepsilon\} \to A$: le mot excepté le premier élément. saufpremier(example) = xample
saufdernier : $A^* \setminus \{\varepsilon\} \to A$: le mot excepté le dernier élément. saufpremier(example) = exampl
estvide : $A^* \to \{vrai, faux\}$: vrai ssi le mot est vide. estvide(ε) = $vrai$, estvide(example) = $faux$.
longueur : $A^* \to \mathbb{N}$: le nombre d'éléments du mot. longueur(example) = 7

Donner un algorithme récursif de la fonction Egal telle que Egal(P,Q) renvoie vrai si et seulement si les deux mot P et Q sont égaux.

Exercice 2 - Le PGCD de deux entiers naturels peut être calculé selon l'algorithme d'Euclide, qui s'écrit sous forme récursive comme suit. Écrire une version itérative de cet algorithme.

Algorithme 1 : Calcul récursif du PGCD

```
début
    /* ENTRÉES : Deux entier m et n */
    /* SORTIE : PGCD(m,n) */
    si m = 0 alors retourner n
    sinon
        retourner PGCD(n mod m, m)
fin
```

Exercice 3 - L'algorithme 2 représente une forme récursive non terminale où l'appel récursif est inclus dans une composition. Contrairement à l'exemple du cours, il n'exécute pas une action mais retourne un résultat. Sa version dérécursivée est l'algorithme 3.

a) L'algorithme 4 représente un cas particulier de l'algorithme 2. Ré-écrivez l'algorithme générique 3 d'une manière spécifiquement adaptée à l'algorithme 4

b) Testez votre version itérative sur le nombre 3.

c) Comparez la complexité de la version itérative de la question précédente avec celle de l'algorithme 5

Algorithme 2 : Une fonction récursive non terminale f

début
- /* ENTRÉE : x, SORTIE : résultat $f(x)$ */
- **si** $condition1(x)$ **alors retourner** $f_1(x)$
- ...
- **sinon si** $condition\_p(x)$ **alors retourner** $f_p(x)$
- **sinon**
  - **retourner** $g\Big(f\big(T(x)\big),x\Big)$

fin

Algorithme 3 : Expression itérative de la même fonction f

début
- /* ENTRÉE : x, SORTIE : résultat $f(x)$ */
- $P \leftarrow$ pilevide
- empile$((x,"appel"),P)$
- **tant que** *not* estvide(P) **faire**
  - $(y,etat) \leftarrow$ sommet(P) ; depile(P)
  - **si** $etat =$ "$appel$" **alors**
    - **si** $condition1(y)$ **alors** $res \leftarrow f_1(y)$
    - ...
    - **sinon si** $condition\_p(y)$ **alors** $res \leftarrow f_p(y)$
    - **sinon**
      - empile$((y,"retour"),P)$
      - empile$((T(y),"appel"),P)$
  - **si** $etat =$ "$retour$" **alors**
    - $res \leftarrow g(res,y)$
- **retourner** res

fin

Algorithme 4 : Fact_Rec

début
- /* ENTRÉES : Un nombre n ; SORTIE : La factorielle de n */
- **si** $n \leq 1$ **alors**
  - Retour 1
- **sinon**
  - Retour $n*$ Fact_Rec$(n-1)$

fin

Algorithme 5 : Fact_Iter

début
- /* ENTRÉES : Un nombre n ; SORTIE : La factorielle de n */
- $f \leftarrow 1$
- **pour** $i = 2$ *à* n **faire**
  - $f \leftarrow f*i$
- Retour f

fin

Exercice 4 - Complexité de la recherche d'un élément majoritaire dans un vecteur

Soit un vecteur V de n éléments. On dit qu'un élément x est *majoritaire* s'il y a strictement plus de $n/2$ éléments égaux à x.

a) L'algorithme 6 divise le vecteur en deux presque-moitiés V_1 et V_2. S'il existe un élément majoritaire dans V, il est majoritaire dans au moins une des deux moitiés. On cherche donc les éléments majoritaires de V_1 et V_2 (s'ils existent), on calcule le nombre total d'occurrences de chacun dans V et on en déduit si un des deux est majoritaire dans V.

   Exprimer par une formule de récurrence la complexité dans le pire cas $C(n)$ de l'algorithme 6, sachant que pour un vecteur V de taille n, occurrence(x,V) a une complexité de n comparaisons.

b) En approchant n par un nombre de la forme 2^p et en posant $x(p) = C(2^p)$, exprimer $x(p)$ en fonction de p puis $C(n)$ en fonction de n.

   Il peut être utile de passer par les séries génératrices et d'utiliser le résultat suivant : la suite u de série génératrice $U(z) = \frac{1}{(1-\alpha z)^2}$ a pour terme courant $u(p) = \alpha^p(p+1)$.

Algorithme 6 : Recherche d'élément majoritaire selon le principe "diviser pour régner" : fonction *majoritaire*.

```
début
    /* ENTRÉE : un vecteur d'entiers V de taille n*/
    /* SORTIES : l'élément majoritaire et son nombre d'occurrences s'il existe, (∅,0) sinon*/
    si n = 1 alors retourner (V(1),1)
    sinon
        (x, n_x) ← majoritaire(V(1 → ⌊n/2⌋))
        (y, n_y) ← majoritaire(V(⌊n/2⌋ + 1 → n))
        si n_x ≠ 0 alors
            n_x ← n_x + occurrence(x, V(⌊n/2⌋ + 1 → n))
        si n_y ≠ 0 alors
            n_y ← n_y + occurrence(y, V(1 → ⌊n/2⌋))
        si n_x > n/2 alors retourner (x, n_x)
        sinon si n_y > n/2 alors retourner (y, n_y)
        sinon retourner (∅, 0)
fin
```