2I003 TD5 Listes

Exercice 1 – Polynômes

Le but de cet exercice est d'écrire un ensemble de fonctions qui permettent de manipuler des polynômes représentés en machine sous la forme d'une liste de monômes. Tout monôme de la forme ax^n est ainsi représenté par un couple d'entiers `(facteur, puissance)` de valeur (a, n). Un polynôme est alors une liste de monômes dans l'ordre des puissances de x décroissantes.

A titre d'exemple, le polynôme $5x^3 + 3x^2 + 1$ est représenté par la liste composée dans l'ordre des monômes $(5, 3)$, $(3, 2)$ et $(1, 0)$.

Question 1

On considère la fonction récursive `def RajouteMonome(p, m)` qui rajoute le monôme m à sa place dans le polynôme p. Si le polynôme p possède déja un monôme de puissance $m.puissance$, m n'est pas rajouté à p.

```
def RajouteMonome(p,m):
        if p==[]:
                return [m]
        if p[0].puissance==m.puissance:
                return p
        if p[0].puissance<m.puissance:
                L=p
                L.insert(0,m)
        else:
                L=RajouteMonome(p[1:],m)
                L.insert(0,p[0])
        return L
```

Evaluez la complexité de cette fonction dans le pire des cas et le meilleur des cas en fonction de la représentation de p (tableau ou liste simplement chaînée).

Question 2

On considère la fonction itérative `def AffichagePolynome(p)` qui affiche tous les monômes du polynôme p.

```
def AffichagePolynome(p):
        for m in p:
                print "a=", m.facteur, "puiss=", m.puissance
```

Evaluez la complexité de cette fonction pour les deux représentations de p.

Question 3

On considère la fonction `def DeriverPolynome(p)` qui construit récursivement le polynôme dérivé de p en dérivant le premier monôme (si il existe).

```
def DeriverPolynome(p):
        if (p==[] or p[0].puissance==0):
                return []
```

```
        m=p.pop(0)
        m.facteur=m.facteur*m.puissance
        m.puissance=m.puissance-1

        p = DeriverPolynome(p)
        p.insert(0,m)
        return p
```

Evaluez la complexité de cette fonction pour les deux représentations de p. Que peut-on faire pour améliorer la complexité si p est représentée sous la forme d'un tableau ?

Question 4

On considère la fonction `int EvalPolynome(p, x)` qui calcule la valeur du polynôme p en x.

```
def EvalPolynome(p, x):
        if (p==[]):
                return 0
        return p[0].facteur*x**(p[0].puissance)+EvalPolynome(p[1:],x)
```

Quelle est la complexité de cette fonction ?

Question 5

Soit $P(x)$ de la forme $P(x) = \sum_{i=0}^{n} a_i x^i$. On définit la suite de polynômes de Horner associée à P par $H_0^P(x) = a_n$, $H_1^P(x) = H_0(x)x + a_{n-1}, \cdots, H_n^P(x) = H_{n-1}^P(x)x + a_0$. Démontrer que $P(x) = H_n^P(x)$.

Question 6

On suppose dans cette question (uniquement) que le polynôme p possède tous ses puissances On considère la fonction récursive `def EvalHorner(p,x)` qui calcule la valeur du polynôme p en x en utilisant un schéma d'évaluation de Horner.

```
def EvalHorner(p,x):
        if (p==[]):
                return 0
        a0=p[len(p)-1].facteur
        L=p[0:len(p)-1]
        return EvalHorner(L,x)*x+a0
```

Démontrez sa terminaison et sa validité. Quelle est la complexité de cette nouvelle fonction ?

Question 7

Ecrire la fonction récursive `def SommePolynome(pol1, pol2)` qui effectue la somme des deux polynômes $pol1$ et $pol2$. Ces deux polynômes sont détruits par cette fonction.

Démontrez la terminaison et la validité de cette fonction. Quelle est sa complexité dans le pire des cas en fonction de la représentation choisie pour p (tableau ou liste chaînée) ?

Exercice 2 – Implantation d'une file dans un tableau circulaire

Une file est une structure de données pour laquelle les insertions se font en queue et les retraits en tête. On les désigne souvent pas l'acronyme FIFO (First-In, First-Out). Le but de cet exercice est d'étudier l'implantation d'une file dans un tableau géré de manière circulaire. La structure de données de type `File` que l'on utilise comporte 4 éléments définis de la manière suivante :

— Un tableau tab de n éléments.
— $tete$ désigne l'indice du tableau correspondant au premier élément dans la file. Au démarrage, $tete = 0$.
— $queue$ désigne la première place disponible dans le tableau. Au démarrage, $queue = 0$.

— $plein$ est un booléen vrai si le tableau est plein.

Question 1
Ecrire la fonction `def Ajouter(F, elem)` qui permet de rajouter un élément $elem$ la file F. Quelle est la complexité de cette fonction ?

Question 2
Dans quel cas la file est-elle vide ? En déduire le code de la fonction `def Retirer(F)` qui renvoie le premier élément de la file, -1 sinon. Quelle est la complexité de cette nouvelle fonction ?

Question 3
Proposer une autre structure bien adaptée pour une file. Que vaut alors la complexité de l'insertion et la suppression ? Avantages, inconvénients ?

Exercice 3 – Opérations de base sur des ensembles d'entiers

On considère deux ensembles d'entiers $A = \{a_0, \cdots, a_{n-1}\}$ et $B = \{b_0, \cdots, b_{m-1}\}$ tels que $a_0 < \cdots < a_{n-1}$ et $b_0 < \cdots < b_{m-1}$. Ces ensembles sont stockés dans des listes LA et LB. Le but de cet exercice est d'écrire les fonctions associées à des opérations de base sur les ensembles.

Question 1
Ecrire la fonction récursive `def insertionTrie(elem, R)` qui, pour une liste trié d'entiers tous différents, insère l'entier $elem$ à sa place, si il n'est pas dans R et retourne la liste obtenue. En déduire la fonction `def Ensemble(L)` qui renvoie un ensemble à partir des éléments de L (soit renvoie une liste croissante en enelemevant les doublons).

Quelle est la complexité de ces deux fonctions pour un tableu ou une liste simplement chaînée ?

Dans le pire des cas, on insère à la fin. La fonction d'insertion est donc en $\mathcal{O}(n)$ pour un tableau ou une liste chaînée. Dans le meilleur des cas, c'est en $\Omega(1)$ avec une insertion en tête. La fonction `Ensemble(L)` est donc en $\mathcal{O}(n^2)$ dans le pire des cas, et en $\Omega(n)$ sinon.

Question 2
Ecrire la fonction récursive `def Intersection(LA, LB)` qui renvoie la liste correspondant à $A \cap B$ triée en ordre croissant.

Question 3
Démontrez la terminaison et la validité de cette fonction.

Question 4
Evaluez la complexité de cette fonction, selon que les listes sont représentées dans un tableau ou une liste simplement chaînée.

Question 5
Ecrire la fonction récursive `def Union(LA, LB)` qui renvoie la liste correspondant à $A \cup B$ triée en ordre croissant.

Question 6
La différence symétrique de A et B est le sous-ensemble des éléments de A ou B qui ne sont pas communs à A et B. On a donc $A \Delta B = (A \cup B) \setminus (A \cap B)$.

Soient les ensembles $A = \{3, 6, 13, 15\}$ et $B = \{2, 6, 15, 18\}$. Que vaut $A \Delta B$?

Question 7
Que vaut $A \Delta B$ si $A = \emptyset$? Si $B = \emptyset$?

Question 8
On suppose dans cette question que $A \neq \emptyset$ et $B \neq \emptyset$ et on note $A' = A - \{a_0\}$ et $B' = B - \{b_0\}$. Montrez que :

$$A\Delta B = \begin{cases} (A'\Delta B') & \text{si } a_0 = b_0 \\ \{a_0\} \cup (A'\Delta B) & \text{si } a_0 < b_0 \\ \{b_1\} \cup (A\Delta B') & \text{si } a_0 > b_0 \end{cases}$$

Question 9
En déduire la fonction récursive `def DifferenceSym(LA, LB)` qui renvoie la liste correspondant à $A\Delta B$ triée en ordre croissant.

Question 10
Donner la liste chronologique des appels de la fonction `DifferenceSym(LA, LB)` pour les listes `LA=[3, 6, 13, 15]` et `LB=[2, 6, 15, 18]`.

Exercice 4 – Recherche du k-ième élément - Extrait de Juin 2011
Le but de cet exercice est d'étudier la sélection du k-ième élément dans une **liste** non triée à n éléments.
Dans tout l'exercice, on considère une liste L d'entiers **tous différents**. La liste possède n éléments $L[0], \ldots, L[n-1]$.
Pour tout entier $k \in \{1, \ldots, n\}$, le k-ième plus petit élément désigne l'élément de la liste qui possède exactement $k-1$ éléments strictement plus petits que lui dans la liste.

Préliminaires et algorithme naïf

Question 1
Soit $L = [2, 5, 10, 7, 23, 1, 14, 12]$. Quel est le plus petit élément ? Quel est le 5-ième plus petit élément de cette liste ? Et le 8-ième ?

Question 2
On suppose **uniquement dans cette question** que la liste L est **triée** en ordre croissant. Où se trouve le k-ième plus petit élément de la liste L ? En déduire la complexité d'un algorithme qui retourne cet élément si la liste est représenté par un tableau. Même question si la liste est représentée par une liste simplement chaînée.

Question 3
Quelle est la complexité minimale dans le pire des cas pour trier par comparaison une liste de n éléments ? Citer un algorithme de cette complexité si la liste est représentée par un tableau. Même question si la liste est représentée par une liste simplement chaînée.

Question 4
En déduire une première méthode pour obtenir le k-ième plus petit élément d'une liste non triée à n éléments. Quelle en est la complexité pour un tableau ou une liste simplement chaînée ?

Un algorithme général
Soit U une liste et x un entier appartenant à U. `PARTITION(U, x)` renvoie un triplet sous la forme d'une liste $[U_1, p, U_2]$ où p est un entier et U_1 et U_2 sont deux listes telles que :
- U_1 contient tous les éléments de U strictement plus *petits* que x,
- U_2 contient tous les éléments de U strictement plus *grands* que x,
- $p = |U_1| + 1$ (où $|U_1|$ est le nombre d'éléments de U_1).

On remarque que x est le p-ième plus petit élément de la liste U.
Par exemple :
`PARTITION` $([2, 5, 10, 7, 23, 1, 14, 12], 7) = [[2, 5, 1], 4, [10, 23, 14, 12]]$
`PARTITION` $([2, 5, 10, 7, 23, 1, 14, 12], 1) = [[], 1, [2, 5, 10, 7, 23, 14, 12]]$

Voici la définition de la fonction `PARTITION(U,x)` :

```
def PARTITION(U,x):
  n=len(U) ; U1=[] ; U2=[] ; p=1 ; i=0
  while i<n:
    if U[i]<x:
      U1.append(U[i]) ; p=p+1
    elif U[i]>x:
      U2.append(U[i])
    i=i+1
  return [U1,p,U2]
```

Question 5
Quel est le nombre d'itérations de la boucle **while** ? En supposant que la complexité de la fonction `len` est en $\Theta(1)$, quelle est la complexité de `PARTITION(U,x)` si les listes sont représentées par des tableaux surdimensionnés de n éléments ? Même question si les listes sont représentées par des listes simplement chaînées.

Question 6
Quelle est la proprié qu'il faut démontrer pour obtenir la validité de `PARTITION(U,x)`.
Soit maintenant la définition de la fonction `RECHERCHE(L,k)` qui renvoie le k-ième plus petit élément de la liste L pour $k \in \{1, \ldots, |L|\}$:

```
def RECHERCHE(L,k):
  pivot=L[0]
  [L1,p,L2]=PARTITION(L,pivot)
  if p==k:
    return pivot
  elif p>k:
    return RECHERCHE(L1,k)
  else:
    return RECHERCHE(L2,k-p)
```

Question 7
Exécuter l'appel de `RECHERCHE([3, 5, 23, 10, 7 , 1, 14, 12],5)` en donnant l'ensemble des appels récursifs. Les appels à `PARTITION` seront mentionnés avec leur résultat, mais non déroulés.

Question 8
En supposant la validité et la terminaison de `PARTITION`, prouver la validité et la terminaison de `RECHERCHE(L,k)`.

Dans les 3 questions qui suivent, on suppose que toutes les listes manipulées sont représentées sous la forme de tableaux surdimensionnés de n éléments.

Question 9
Quel est, pour une liste L de n élements, le nombre minimum d'appels à `PARTITION` pour un appel à `RECHERCHE(L, k)` ? Donnez un cas pour lequel cette valeur est atteinte. En déduire une évaluation de la complexité dans le meilleur des cas pour `RECHERCHE`.

Question 10
Quel est, pour une liste L de n éléments, le nombre maximum d'appels à `PARTITION` pour un appel à `RECHERCHE(L, k)` ? Donnez un cas où cette valeur est atteinte. En déduire une évaluation de la complexité dans le pire des cas pour `RECHERCHE`.

Question 11
Soit k un entier fixé. On exécute `RECHERCHE(L,k)` sur une liste de taille n :
- avec quel genre de liste fait-on le plus grand nombre d'appels à `PARTITION` ?
- combien fait-on d'appels à `PARTITION` dans ce cas ?
- en déduire la complexité de `RECHERCHE(L,k)` dans le pire cas.