ASI3 10 novembre 2015

Algorithmique et bases de la programmation

Durée : *1h30*
Documents autorisés : **AUCUN**

Remarques :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.
- N'utilisez pas de crayon à papier sur votre copie.

1 QCM (5 points)

Répondez au qcm ci joint (à rendre avec votre copie). La note de chaque question peut varier de -1 à $+1$. La note à l'exercice (0 au minimum) est la moyenne des notes de chaque question multipliée par le nombre de points de l'exercice.

Soit B le nombre de bonnes réponses à une question et soit M le nombre de mauvaises réponses à cette même question ($B + M$ est égale au nombre de réponses à la question). Chaque bonne réponse cochée rapporte $1/B$ points et chaque mauvaise réponse $-1/M$ points.

2 Recherche d'un élément (5 points)

Écrire une fonction, *recherche*, qui détermine en $O(log_2(n))$ le plus **grand** indice d'un élément (dont on est sûr de l'existence) dans un tableau d'entiers t de n éléments triés ordre croissant. Il peut y avoir des doublons (ou plus) dans le tableau.

— **fonction** recherche (t : **Tableau**[1..MAX] **d'Entier** ; n : **NaturelNonNul** ; element : **Entier**) **: NaturelNonNul**
⌊**précondition(s)** $\exists\ 1 \leq i \leq n$, t[i] = element

Solution proposée :

```
fonction rechercheDichotomique (t : Tableau[1..MAX] d'Entier ; n : NaturelNonNul ; element : Entier) : NaturelNonNul
    ⌊précondition(s)  ∃ 1 ≤ i ≤ n / t[i] = element
    Déclaration  a,b,m resultat : NaturelNonNul
debut
    a ← 1
    b ← n
    tant que a < b faire
        m ← (a + b) div 2
        si t[m] = element alors
            si t[m+1] = element alors
                a ← m+1
            sinon
                a ← m
            finsi
        sinon
            si t[m] < element alors
                a ← m + 1
            sinon
                b ← m - 1
            finsi
        finsi
    fintantque
    resultat ← b
    retourner resultat
fin
```

3 Documents et $n-grams$ (10 points)

Un $n-gram$ est une succession de n caractères. Pour un alphabet donné (une liste de caractères), l'ensemble des $n-grams$ forme ce que l'on appelle une base. Par exemple pour l'alphabet composé des lettres *'a'*, *'b'* et *'c'*, la base de taille 2 est composée des $2-grams$: *"aa"*, *"ab"*, *"ac"*, *"ba"*, *"bb"*, *"bc"*, *"ca"*, *"cb"* et *"cc"*.

Dans le domaine de l'analyse de documents (textes) les lettres en minuscule sont l'alphabet de la base. On peut alors représenter un document d pour une base b donnée par des valeurs x_i ($i \in 1..|b|$), telles que la valeur x_i est le nombre d'occurrences du $n-gram$ g_i dans les mots du document d.

3.1 Analyse (2 points)

Donnez l'analyse descendante de l'opération *representationDUnDocument* qui permet de calculer, à partir d'une base (une liste de $n-grams$), le nombre d'occurrences de chaque $n-gram$ dans un document (représenté par une chaîne de caractères). Vous utiliserez uniquement les boites proposées par la figure 1, tel que :

documentEnMots permet d'obtenir la liste des mots formant un document ;

initialisationNbOccurences associe la valeur 0 à chaque $n-gram$ d'une liste de $n-grams$;

additionnerNbOccurencesDesNGramsPourUnMot incrémente le nombre d'occurrences des $n-grams$ présents dans un mot.

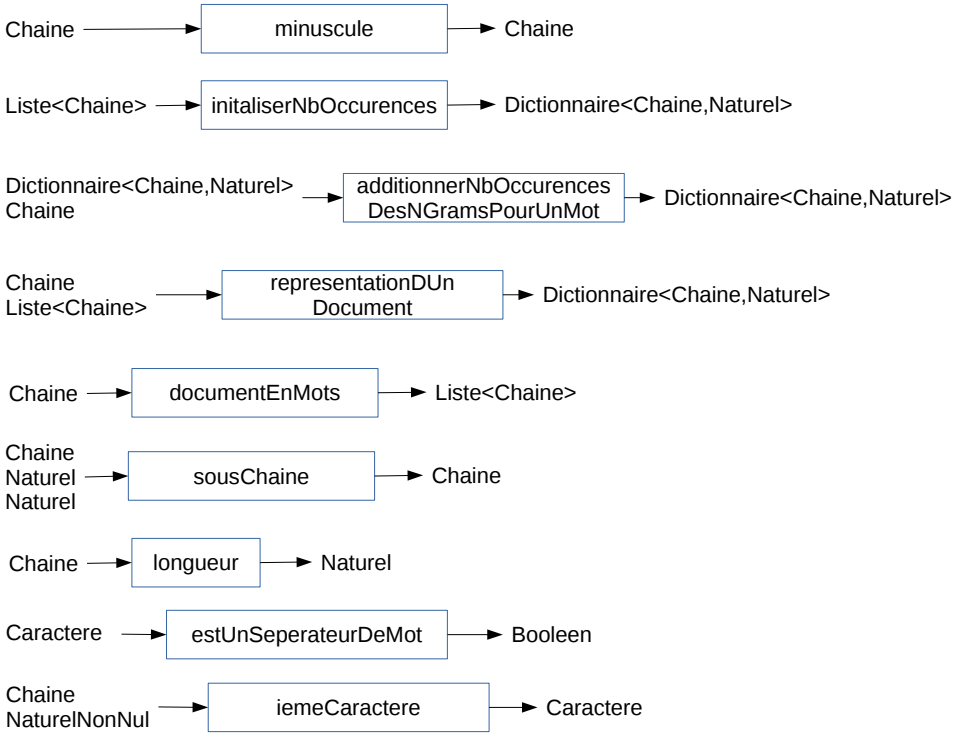
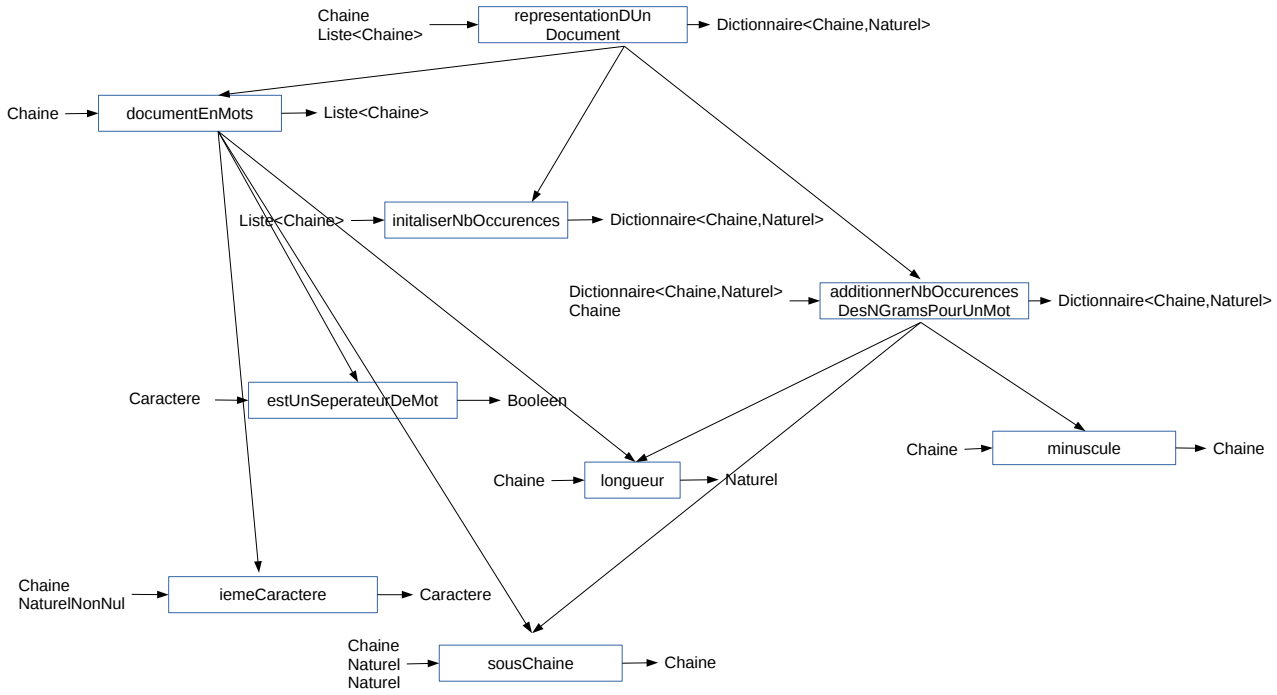


FIGURE 1 – Boites pour définir l'analyse descendante de *representationDUnDocument*

Solution proposée :



3.2 Conception détaillée (8 points)

On considère comme connues les fonctions suivantes :

- **fonction** longueur (uneChaine : **Chaine de caracteres) : Naturel**
- **fonction** iemeCaractere (uneChaine : **Chaine de caracteres**, iemePlace : **NaturelNonNul) : Caractere**

  ⌊**précondition(s)** $iemePlace \leq longueur(uneChaine)$
- **fonction** sousChaine (uneChaine : **Chaine de caracteres**, debut, fin : **NaturelNonNul) : Chaine de caracteres**
- **fonction** minuscule (ch : **Chaine de caracteres) : Chaine de caracteres**
- **fonction** documentEnMots (doc : **Chaine de caracteres) :** Liste<**Chaine de caracteres**>

3.2.1 Construction de la base (3 points)

Proposez l'algorithme de la fonction suivante qui calcule l'ensemble des $n-grams$ de longueur n pour un alphabet donné :

- **fonction** base (alphabet : Liste<**Caractere**>, n : **NaturelNonNul) :** Liste<**Chaine de caracteres**>

Solution proposée :

fonction base (alphabet : Liste<**Caractere**>, n : **NaturelNonNul) :** Liste<**Chaine de caracteres**>

Déclaration resultat,temp : Liste<**Chaine de caracteres**>

debut

```
resultat ← liste()
si n = 1 alors
    pour chaque car de alphabet
        insérer(resultat, longueur(resultat), caractereEnChaine(car))
    finpour
sinon
    temp ← base(alphabet,n-1)
    pour chaque car de alphabet
        pour chaque nGram de temp
            insérer(resultat, longueur(resultat), caractereEnChaine(car)+nGram)
```

```
        finpour
    finpour
  finsi
  retourner resultat
fin
```

3.2.2 Initialiser le nombre d'occurrences de $n-gram$ (1 point)

Proposez l'algorithme de la fonction suivante :

— **fonction** initialisationNbOccurences (nGrams : Liste<**Chaine de caracteres**>) **:** Dictionnaire<**Chaine de caracteres**,**Naturel**>

Solution proposée :

```
fonction initialisationNbOccurences (nGrams : Liste<Chaine de caracteres>) : Dictionnaire<Chaine de caracteres,Naturel>
  Déclaration  resultat : Dictionnaire<Chaine de caracteres,Naturel>
               nGram : Chaine de caracteres
debut
  resultat ← dictionnaire()
  pour chaque nGram de nGrams
    ajouter(resultat,nGram,0)
  finpour
  retourner resultat
fin
```

3.2.3 Occurrences des $n-grams$ présents dans un mot (2 points)

Proposez l'algorithme de la procédure suivante :

— **procédure** additionnerNbOccurencesDesNGramsPourUnMot (**E/S** nbOccurrences : Dictionnaire<**Chaine de caracteres**, **Naturel**>, **E** mot : **Chaine de caracteres**, n : **NaturelNonNul**)

⌊**précondition(s)** longueur(mot)≥n

Solution proposée :

```
procédure additionnerNbOccurencesDesNGramsPourUnMot (E/S nbOccurrences : Dictionnaire<Chaine de caracteres, Naturel>, E mot : Chaine de caracteres, n : NaturelNonNul)
  ⌊précondition(s)  longueur(mot)≥n
  Déclaration  nGram : Chaine de caracteres
               i : NaturelNonNul
debut
  pour i ←1 à longueur(mot)-n+1 faire
    nGram ← minuscule(sousChaine(mot,i,n+i-1))
    ajouter(nbOccurences, nGram, 1 + obtenirValeur(nbOccurences,nGram))
  finpour
fin
```

3.2.4 Occurrences des $n-grams$ présents dans un texte (2 points)

Proposez l'algorithme de la fonction suivante :

— **fonction** representationDUnDocument (doc : **Chaine de caracteres**, nGrams : Liste<Chaine>) **:** Dictionnaire<**Chaine de caracteres**,**Naturel**>

Solution proposée :

```
fonction representationDUnDocument (doc : Chaine de caracteres, nGrams : Liste<Chaine>) : Dictionnaire<Chaine de caracteres,Naturel>
  Déclaration  resultat : Dictionnaire<Chaine de caracteres,Naturel>
               n : NaturelNonNul
               mot : Chaine de caracteres
debut
```

```
    resultat ← initialisationNbOccurences(nGrams)
    n ← longueur(obtenirElement(nGrams,1))
    pour chaque mot de documentEnMots(doc)
        si longueur(mot)≥n alors
            additionnerNbOccurencesDesNGramsPourUnMot(resultat, mot, n)
        finsi
    finpour
    retourner resultat
fin
```

Annexe

Voici la conception préliminaire de quelques TAD.

Liste

- **fonction** liste **() :** Liste
- **fonction** estVide **(**uneListe : Liste**) : Booleen**
- **procédure** insérer **(E/S** uneListe : Liste, **E** position : **Naturel**, element : Element**)**

  ⌊**précondition(s)** $1 \leq position \leq longueur(uneListe) + 1$
- **procédure** supprimer **(E/S** uneListe : Liste, **E** position : **Naturel)**

  ⌊**précondition(s)** $1 \leq position \leq longueur(uneListe)$
- **fonction** obtenirElement **(**uneListe : Liste, position : **Naturel) :** Element

  ⌊**précondition(s)** $1 \leq position \leq longueur(uneListe)$
- **fonction** longueur **(**uneListe : Liste**) : Naturel**

Dictionnaire

- **fonction** dictionnaire **() :** Dictionnaire
- **procédure** ajouter **(E/S** unDictionnaire : Dictionnaire, **E** clef : Clef, element : Valeur**)**
- **procédure** retirer **(E/S** unDictionnaire : Dictionnaire, **E** clef : Clef**)**
- **fonction** estPresent **(**unDictionnaire : Dictionnaire, clef : Clef**) : Booleen**
- **fonction** obtenirValeur **(**unDictionnaire : Dictionnaire, clef : Clef**) :** Valeur

  ⌊**précondition(s)** estPresent(unDictionnaire, clef)
- **fonction** obtenirClefs **(**unDictionnaire : Dictionnaire**) :** Liste<Clef>
- **fonction** obtenirValeurs **(**unDictionnaire : Dictionnaire**) :** Liste<Valeur>