Analyse d'algorithme : HLIN508

Aucun document autorisé.

1 Arbre de soustractions

1.1 Définition

1.1.1 Les types

Les sommets de l'arbre peuvent être de deux types :
- des feuilles
- des noeuds binaires

Les feuilles sont étiquetées par un entier

Les noeuds binaires ont exactement deux fils, qui peuvent chacun être une feuille ou un noeud binaire

1.1.2 Les méthodes

Les prédicats appliqués à un sommet, ils renvoient un booléen :
- `bool sommet::F?()` renvoie `vrai` si et seulement si le sommet auquel on applique la méthode est une feuille
- `bool sommet::B?()` renvoie `vrai` si et seulement si le sommet auquel on applique la méthode est un sommet binaire

les constructeurs
- `sommet::sommet(int)` construit une feuille étiquetée par l'entier passé en paramètre
- `sommet::sommet(sommet*, sommet*)` construit un noeud binaire dont le fils gauche (rep. droit) a pour adresse le premier (resp. deuxième) pointeur passé en paramètre

les autres méthodes ne sont définies chacunes que pour un type de noeud :

- `int sommet::VA()` n'est définie que pour les feuilles, et renvoie le nombre qui étiquete la feuille

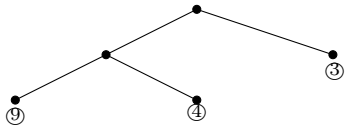
  Les deux autres méthodes ne sont définies que pour les sommets binaires

- `sommet* sommet::FG()` renvoie un pointeur sur le fils gauche du sommet avec lequel est invoqué la méthode
- `sommet* sommet::FD()` renvoie un pointeur sur le fils droit du sommet avec lequel est invoqué la méthode

1.2 Évaluation

Chacun des noeuds internes représente le signe *moins*

Ainsi l'arbre suivant



représente l'expression
$((9 - 4) - 3)$
et s'évalue à 2

Question 1 Écrire une méthode `int sommet::Eval()` qui renvoie la valeur que représente l'arborescence enracinée dans le sommet sur lequel est invoqué la méthode.
Indiquer (sans justification) sa classe de complexité.

1.3 Construction

On dispose d'un tableau de caractères dans lequel est écrite l'expression arithmétique, à raison d'un caractère par case du tableau.
Les caractères que peut contenir un tableau sont donc les chiffres, le signe $'-'$ et les parenthèses, soit l'ensemble de caractères {-, (,), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.

1.3.1 Spécifications de l'expression arithmétique contenue dans le tableau

Ce sont les mots générés par la grammaire
- d'alphabet terminal l'ensemble de caractères : {-, (,), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- d'alphabet non terminal l'ensemble $\{E_{xpression}, N_{ombre}, C_{hiffre}\}$
- d'axiome $E_{xpression}$
- et de règles de production
  - $E_{xpression} \rightarrow (E_{xpression} - E_{xpression}) \mid N_{ombre}$
  - $N_{ombre} \rightarrow N_{ombre}C_{hiffre} \mid C_{hiffre}$
  - $C_{hiffre} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

En d'autres termes
- un nombre est composé de chiffres (au moins un)
- une expression est
  - soit un nombre
  - soit la soustraction, placée entre parenthèses, de deux expressions
    On dira alors que le signe moins *correspond à* la parenthèse ouvrante.

1.3.2 Les méthodes de la `struct tab` qui contient le tableau :

— `int tab::PREM()` qui renvoie le premier indice du tableau

— `bool tab::NB?(int)` qui renvoie vrai si et seulement si la case du tableau dont l'indice est passé en paramètre contient un chiffre.

— `bool tab::PO?(int)` qui renvoie vrai si et seulement si la case du tableau dont l'indice est passé en paramètre contient une parenthèse ouvrante.

La méthode suivante n'est définie que si l'indice passé en paramètre n'est pas le dernier indice du tableau.

— `int tab::SUIV(int)` renvoie l'indice suivant l'indice passé en paramètre

La méthode suivante n'est définie que si la case du tableau dont l'indice est passé en paramètre contient une parenthèse ouvrante.

— `int tab::PM(int)` renvoie l'indice suivant celui du signe moins qui correspond à la parenthèse ouvrante dont l'indice est passé en paramètre.

La méthode suivante n'est définie que si la case du tableau dont l'indice est passé en paramètre contient un chiffre

— `int tab::NB(int)` renvoie la valeur du nombre correspondant à la suite consécutive de chiffres qui commence à l'indice passé en paramètre

Question 2 Donner (sans justification) la classe de complexité de chacune de ces méthodes. Vous indiquerez quel paramètre vous utilisez pour mesurer cette complexité.

Question 3 Écrire une méthode récursive `sommet* tab::CSTR(int` *deb*`)`
— qui construit l'arbre qui correspond à la sous formule commençant à l'indice *deb* dans le tableau de la struct sur laquelle est invoquée la méthode et
— qui renvoie un pointeur sur la racine de cet arbre.
L'appel initial de cette méthode recursive sur une `struct` T sera : $T \rightarrow$ `CSTR(PREM())`

Question 4 Si la sous formule commençant à l'indice *deb* contient p signes moins, quel est le nombre total d'appels à `CSTR` (y compris le premier) ? Justifier.

Question 5 Soit un arbre complet de hauteur h, dont chaque feuille est étiquetée par un nombre de un chiffre.
— quel est le nombre p de noeuds internes ?
— quelle est la longueur n de la formule correspondante ?

Question 6 À quelle classe de complexité Θ, fonction de la longueur de la formule, appartient `CSTR` ? Justifier.

Question 7 Imaginer une façon d'obtenir que la complexité de `CSTR()` devienne linéaire.
On ne demande pas d'écrire un algorithme, seulement de donner des indications pour sa construction.