Examen de

λ-calcul, typage et algorithmique

Durée : 2 h

Tous documents autorisés

Ce sujet comporte 2 pages.

Les algorithmes demandés pourront être écrits dans le langage de programmation de votre choix.

1 λ-calcul typé

1. [1pt] Typer le terme suivant :

   ```
   fun f g x -> f (fun y -> g y x) x
   ```

2. [2pt] Rappeler la règle de typage du `let` : `let` x `=` e_1 `in` e_2 et proposer une règle pour le `let rec` : `let rec` f `=` e_1 `in` e_2.

3. [3pt] Donner les règles de typage associés aux types suivants :

   ```
   type 'a t1 = { x : 'a; y : int };;
   type t2 =
       X
     | Y of t2 t1;;
   ```

2 Sous-tableau de somme maximale

Soit un tableau d'entiers a_1, a_2, ..., a_n. On cherche à trouver $1 \leq i < j \leq n$ tels que la somme $\sum_{k=i}^{j} a_k$ soit maximale. Par exemple, dans le tableau $[1, -2, 1, 2, -1, 3, -2, 1]$, on a $i = 3$ et $j = 6$ pour une somme de $1 + 2 - 1 + 3 = 5$.

1. [3pt] Donner un algorithme naïf pour calculer i et j, puis estimer sa complexité temporelle.
2. [3pt] En utilisant la technique « diviser pour régner », proposer un algorithme de résolution. Indications :
   - Diviser le problème sur le tableau a en deux sous-problèmes sur des tableaux a' et a''.
   - Calculer la solution au problème sur le tableau total a en considérant trois cas :
     (a) la solution pour a est celle de a' ;
     (b) la solution pour a est celle de a'' ;
     (c) la solution est à cheval sur la frontière entre a' et a''.

   Donner la complexité de l'algorithme obtenu.

3. [3pt] Soit le tableau c des cumuls de a, $c_1 = a_1$, $c_2 = c_1 + a_2$, ..., $c_n = c_{n-1} + a_n$.
   (a) Tracer le graphe de c pour l'exemple $a = [1, -2, 1, 2, -1, 3, -2, 1]$.
   (b) $\forall k < k'$, que vaut $c_{k'} - c_k$? En déduire les i et j recherchés.
   (c) Donner un algorithme de complexité linéraire pour les calculer.

3 Distance d'édition

On définit la distance d'édition $d(s_1, s_2)$ entre deux chaînes de caractères s_1 et s_2 comme le **nombre minimal de suppressions, insertions ou substitutions** de caractères nécessaire pour transformer l'une en l'autre. Par exemple, la distance $d($`"karpov"`,`"kasparov"`$)$ est égale à 3, car on peut passer de `"karpov"` à `"kasparov"` en 3 opérations d'édition :
- substitution de `'r'` par `'s'` ;
- insertion de `'a'` ;
- insertion de `'r'`.

On indexera de 1 à n les caractères d'une chaîne de longueur n et on notera $s[1..i]$ la sous-chaîne de caractères constituée des i premiers caractères de s (préfixe) : *e.g.* $s_1[1..4] =$`"karp"` avec $s_1 =$`"karpov"`. Pour deux chaînes de caractères s_1 et s_2 données, on notera $d(i, j)$ la distance d'édition $d(s_1[1..i], s2[1..j])$, et $d(0, j)$ (resp. $d(i, 0)$) la distance d'édition entre la chaîne vide et $s_2[1..j]$ (resp. $s_1[1..i]$ et la chaîne vide).

Ce problème possède la propriété de *sous-structure optimale*, c'est-à-dire qu'on peut calculer la solution optimale pour les sous-chaînes $s_1[1..i]$ et $s_2[1..j]$ à partir des solutions sur les préfixes suivants :
- $s_1[1..i-1]$ et $s_2[1..j]$ (insertion) ;
- $s_1[1..i]$ et $s_2[1..j-1]$ (suppression) ;
- $s_1[1..i-1]$ et $s_2[1..j-1]$ (substitution).

On considère dans la suite deux chaînes de caractères s_1 et s_2 de longueurs respectives n_1 et n_2.

1. [1pt] Donner, en les justifiant, les valeurs $d(s_1, s_1)$ et $d(s_1, s_2)$, dans le cas particulier où s_1 et s_2 n'ont aucun caractère en commun.
2. [1pt] Donner, en les justifiant, les valeurs de $d(0, j)$ et $d(i, 0)$, $\forall j \leq n_2$ et $\forall i \leq n_1$.
3. [2pt] Écrire l'équation de récurrence qui permet de calculer $d(i, j)$ en fonction de $d(i-1, j)$, $d(i, j-1)$ et $d(i-1, j-1)$.
4. [3pt] Donner un algorithme de type **programmation dynamique** pour calculer $d(s_1, s_2)$ et calculer ses complexités temporelles et spatiales.
5. *Questions subsidiaires :* Comment, à partir de l'algorithme précédent, retrouver la séquence d'opérations à effectuer pour transformer s_1 en s_2 ? Cette séquence est-elle unique en générale ?