

UNITE IN3ST02

TD2 - Enoncé

1- OBJECTIFS

- Savoir déterminer un invariant de boucle et construire méthodiquement un algorithme à partir de cet invariant
- savoir établir la preuve d'un programme simple
- savoir calculer la complexité en temps d'algorithmes itératifs ou récursifs simples

Les premiers algorithmes considérés sont simples afin de pouvoir se focaliser sur les principes et la méthode.

Les exercices non traités en séance doivent l'être en travail personnel.

2- TRAVAIL A REALISER

Pour chacun des problèmes suivants, il est demandé de produire dans l'ordre :

1. un invariant (qu'il soit énoncé de façon formelle ou informelle), avec les conditions initiales et finales associées
2. une solution algorithmique commentée par l'invariant afin de fournir des éléments de preuve en ligne (fondamental !); la solution est attendue en langage algorithmique ou dans un langage de programmation quelconque, l'important étant la logique et non la syntaxe.
3. un calcul justifié de la complexité

2.1- Tri pairs-impairs

Soit $t[0..n-1]$ un tableau d'entiers. On veut le trier de façon que tous les nombres pairs, s'il y en a, se retrouvent « à droite » (i.e. aux indices supérieurs) et tous les nombres impairs, s'il y en a, se retrouvent « à gauche » (i.e. aux indices inférieurs). Ecrire un programme P de complexité *linéaire*, qui trie sur place (pas de tableau auxiliaire), et réponde à l'énoncé suivant :

($n > 0$) { P } ($t[0..i-1]$ impairs \wedge $t[i..n-1]$ pairs)

2.2- Drapeau hollandais

Même type d'exercice que le précédent. Soit un tableau $t[0..n-1]$ dont les éléments sont des couleurs à valeur dans {bleu, blanc, rouge}. Le nombre d'éléments d'une couleur donnée est quelconque, éventuellement nul. Le tableau peut être tricolore, bicolore ou monocolore. On veut trier le tableau de façon que tous les bleus, s'il y en a, soient « à gauche », tous les blancs, s'il y en a, soient « au milieu », tous les rouges, s'il y en a, soient « à droite ». Ecrire un programme P de complexité *linéaire*, qui trie sur place, et réponde à l'énoncé suivant :

($n > 0$) { P } ($t[0..b-1]$ bleus \wedge $t[b..r-1]$ blancs \wedge $t[r..n-1]$ rouges)

Ce problème est dû à l'algorithmicien hollandais Dijkstra.

2.3- Suppression des occurrences d'un élément dans un tableau

Soit à supprimer toutes les occurrences d'un élément d'un tableau. Par exemple, supprimer tous les espaces dans une chaîne de caractères.

Soit $t[0..n-1] = [e_0, e_1, \dots, e_{n-1}]$ un tableau contenant n éléments $[e_0, e_1, \dots, e_{n-1}]$. On note « $[e_0, e_1, \dots, e_{n-1}] - e$ » le tableau t duquel tous les éléments e ont été enlevés. Ecrire un programme P de complexité *linéaire*, qui trie sur place, et réponde à l'énoncé suivant :

$$((n > 0) \wedge (t[0..n-1] = [e_0, e_1, \dots, e_{n-1}])) \{ P \} (t[0..j-1] = [e_0, e_1, \dots, e_{n-1}] - e)$$

2.4- Racine carrée entière

La racine carrée entière (i.e. partie entière de la racine carrée) d'un entier naturel n , notée $rce(n)$, est définie par :

$$rce(n) \text{ entier naturel et } rce(n)^2 \leq n < (rce(n)+1)^2$$

Ecrire un programme P calculant $rce(n)$ sans passer par les réels (et donc sans utiliser la fonction standard racine carrée), de complexité *meilleure que* $\Theta(n^{1/2})$, répondant à l'énoncé suivant :

$$(n \geq 0) \{ P \} (d = rce(n))$$

Quelle est la complexité de cet algorithme ?

2.5- Plus longue suite constante

Soit $t[0..n-1]$ un tableau d'entiers. On veut déterminer la première plus longue suite de valeurs constantes. Ecrire un programme P de complexité *linéaire* répondant à l'énoncé suivant :

$$(n \geq 0) \{ P \} (S(i, j, n))$$

où $S(i, j, n) = \ll t[i..j] \text{ est la première plus longue suite constante de } t[0..n-1] \gg$

2.6- Sous-chaîne

On notera :

$s[0 .. n-1]$ une chaîne s de n caractères indicée de 0 à $n-1$

$s[i .. j]$ la sous-chaîne de s indicée de i à j

$s[i .. i+j] = t[k .. k+j]$ pour signifier que les deux sous-chaînes $s[i .. i+j]$ et $t[k .. k+j]$ sont égales élément par élément

Soit à déterminer si une chaîne de caractères $s[0 .. ns-1]$ est une sous-chaîne d'une chaîne de caractères $t[0 .. nt-1]$.

Exemple : "unix" est une sous-chaîne de "abuuniunixeunix"

Ecrire un programme P de complexité $O(ns.nt)$ répondant à l'énoncé suivant :

$$(ns > 0 \wedge nt > 0) \{ P \} (S(i, ns, nt))$$

où $S(i, ns, nt) = \ll \text{si } s[0 .. ns-1] \text{ est une sous-chaîne de } t[0 .. nt-1] \text{ alors } t[i .. i+ns-1] \text{ est la première sous-chaîne de } t[0 .. nt-1] \text{ égale à } s[0 .. ns-1], \text{ sinon } i = -1 \gg$

Nota. La recherche d'une sous-chaîne dans une chaîne (string matching) est un problème d'importance dans le vaste domaine du traitement de texte. L'algorithme le

plus efficace pour des applications usuelles et celui de Boyer-Moore. C'est cet algorithme qui, en version simplifiée ou complète, est généralement implémenté dans les éditeurs de texte pour la commande « search » ou « substitute ». Son efficacité repose sur le fait qu'il utilise les informations déduites de chaque échec local pour éliminer le plus grand nombre possible de positions à vérifier. A cette fin, il pré-traite le motif à rechercher : il pré-calcule deux fonctions (tables de sauts) qui fourniront le nombre de positions à sauter dans le texte avant de poursuivre une recherche. Le coût de l'algorithme peut être sub-linéaire car il n'a pas besoin de vérifier chacun des caractères du texte. Cet algorithme demande en pire cas $3n$ comparaisons de caractères pour une recherche de motif non périodique.

Pour en savoir plus sur les algorithmes d'appariement de chaînes :

www-igm.univ-mlv.fr/~lecroq/string/index.html

2.7- Puissance entière

Soit à calculer x^n , avec x réel et n entier naturel. Ecrire un programme P non récursif, de complexité meilleure que $\Theta(n)$ répondant à l'énoncé suivant :

$$(x \text{ réel} \wedge n \text{ entier naturel}) \{ P \} (y = x^n)$$

Pour cela, on s'appuiera sur la propriété suivante :

$$x^{2k} = (x^2)^k$$

$$x^{2k+1} = x * (x^2)^k$$

Quel est la complexité de cet algorithme ?

2.8- Sous-tableau de somme maximale

Soit $t[0..n-1]$ un tableau d'entiers relatifs. On veut déterminer le premier sous-tableau de somme maximale. Le résultat est constitué de trois valeurs : l'indice de début et l'indice de fin de ce sous-tableau, et la valeur de cette somme. Ecrire un programme P non récursif, de complexité *linéaire* répondant à l'énoncé suivant :

$$(n > 0) \{ P \} (S(i, j, s, n))$$

où $S(i, j, s, n) = \langle t[i..j] \text{ est le premier sous-tableau de somme maximale, } s, \text{ de } t[0..n-1] \rangle$