Université de Marne-la-Vallée
Institut Gaspard-Monge

Licence d'informatique et d'I.U.P. — Algorithmique

Examen de janvier 1998, durée : 3 h.

Seule la consultation des supports de cours, travaux dirigés et travaux pratiques est autorisée pendant le contrôle. Chacune des dix questions compte pour deux points.

Recherche dans une liste de chaînes

Le but de cet exercice est d'écrire un algorithme efficace de recherche d'une chaîne de caractères dans une liste de chaînes.

On considère une liste de n chaînes de caractères mémorisée dans une table T. La longueur maximale des chaînes $T[0]$, $T[1]$, $\ldots$, $T[n-1]$ est notée M. On note $\leq$ l'ordre lexicographique et on suppose que

$$T[0] \leq T[1] \leq \ldots \leq T[n-1].$$

On considère une chaîne x de longueur m ($|x| = m$).

1) Écrire une fonction en langage C qui réalise la recherche dichotomique de x dans T en utilisant comme opération élémentaire de comparaison la comparaison de caractères. Donner une majoration asymptotique de son temps d'exécution ? Combien de comparaisons de caractères effectue-t-elle au maximum ?

Pour deux chaînes u et v, on note $lpc(u, v)$ la longueur du plus long préfixe commun à u et v (par exemple, $lpc(\texttt{aababacb}, \texttt{aabacb}) = 4$).

Pour chaque sous-intervalle $[i, j]$ de $[0, n-1]$ qui intervient lors de la recherche dichotomique on note $LPC[i, j] = lpc(T[i], T[j])$.

2) Soit $[d, f]$ un sous-intervalle de $[0, n-1]$ tel que $d+1 < f$. On note

$$k = \lfloor (d+f)/2 \rfloor, \quad ld = lpc(T[d], x), \quad lf = lpc(T[f], x),$$

On suppose à la fois que $T[d] < x < T[f]$ et que $ld \leq lf$.
Comment se comparent x et $T[k]$ suivant que $lf < LCP[k, f]$, $lf > LCP[k, f]$ ou $lf = LCP[k, f]$? Quelle est la valeur de $lpc(T[k], x)$ dans chacun de ces trois cas ?

3) Déduire de la question **2** une modification de l'algorithme de recherche de x dans T. On supposera calculées les valeurs $LPC[d, f]$ pour tous les intervalles $[d, f]$ qui interviennent pendant la recherche.

Montrer que le nombre de comparaisons de caractères effectuées pendant la recherche est inférieur à $M + \lceil \log_2 n \rceil$ (on pourra considérer les valeurs successives de $\max(ld, lf)$, où ld et lf sont définis dans la question **2**).

4) Pour $n = 6$, dessiner l'arbre (induit par la relation d'inclusion) des intervalles $[d, f]$, $d+1 < f$, qui interviennent dans l'algorithme de recherche.

Donner une majoration du nombre de ces intervalles pour n quelconque.

La question qui suit est indépendante des précédentes.

5) Écrire une implémentation en langage C de la fonction *lpc* qui satisfait les deux propriétés :

- elle utilise `memcmp` comme seule opération de comparaison de chaînes ;
- elle exécute au plus un nombre logarithmique d'appels à `memcmp`.

Préciser le nombre maximal d'appels à `memcmp`. On rappelle le prototype :

```
int memcmp(const void *, const void *, size_t);
```

et le fait que la valeur de `memcmp(x, y, k)` est celle de `strcmp` appliquée aux préfixes de longueur k des chaînes x et y.

Graphes

On considère un graphe valué $G = (S, A, v)$: S est l'ensemble fini des sommets, A est l'ensemble des arcs, et la valuation est la fonction $v \ : A > \mathbb{Z}$. Le coût d'un chemin $(x_0, x_1, \ldots, x_k)$ de longueur k est la somme des valuations des arcs qui le composent : $c(x_0, x_1, \ldots, x_k) = v(x_0, x_1) + v(x_1, x_2) + \ldots + v(x_{k-1}, x_k)$. On note $d(x_0, x_k) = \min\{c(x_0, x_1, \ldots, x_k) \mid (x_0, x_1, \ldots, x_k) \text{ chemin de } G\}$, valeur qui appartient à $\mathbb{Z} \cup \{-\infty\}$.

Les sommets s et t étant fixés, on étudie la détermination d'un chemin de coût minimal d'origine s et d'extrémité t, ainsi que le calcul de $d(s, t)$. On suppose que le graphe ne possède pas de boucle (arc de la forme (x, x)), et que pour tout sommet x de G il existe un chemin de s vers x et un chemin de x vers t.

6) Montrer qu'il existe un chemin de coût minimal d'origine s et d'extrémité t si et seulement si le graphe G ne possède pas de circuit (cycle) de coût strictement négatif.

Pour calculer $d(s, t)$ on dispose d'une fonction h, donnée *a priori*, qui est telle que, pour chaque sommet x de G, $h(x)$ est une évaluation par défaut de $d(x, t)$ ($h(x) \leq d(x, t)$). L'algorithme A^* ci-dessous associe à certains sommets x de G une approximation $F[x]$ de $d(s, t)$ et une approximation $D[x]$ de $d(s, x)$.

7) Appliquer l'algorithme A^* au graphe G_0 de la figure ci-dessous ($s = 1$, $t = 7$) avec la fonction h définie par $h(1) = 3$, $h(2) = 5$, $h(3) = 2$, $h(4) = 6$, $h(5) = 4$, $h(6) = 0$, $h(7) = 0$. On donnera la suite des valeurs prises par la variable x pendant l'exécution de l'algorithme, et les valeurs successives de l'approximation associée à chaque sommet du graphe.

Est-ce que tout sommet du graphe G est ajouté à l'ensemble E pendant l'exécution de l'algorithme ? Est-ce qu'un sommet de G peut être ajouté plusieurs fois à l'ensemble E ?

8) Montrer que, à chaque itération pendant le test $x \neq t$ pour le sommet x lui-même, on a l'inégalité $F[x] \leq d(s, t)$.

Que dire de l'algorithme A^* lorsque simultanément la valuation v est positive ou nulle (pour tout arc (x, y), $v(x, y) \geq 0$) et la fonction h est nulle (pour tout sommet x, $h(x) = 0$) ?

9) Indiquer comment la variable E peut être implémentée pour obtenir un algorithme rapide.

Quel est, avec cette implémentation, le temps d'exécution de l'algorithme A^*.

10) Indiquer comment l'algorithme A^* peut mémoriser un chemin de coût minimal de s à t.

`Algorithme` A^*`(graphe` G`, sommets` s `et` t`, fonction` h`)` $\{$

  `pour tout sommet` x `de` G `faire` $D[x] \longleftarrow +\infty$`;`

  $D[s] \longleftarrow 0$`;` $F[s] \longleftarrow h(s)$`;` $E \longleftarrow \{s\}$`;` $x \longleftarrow s$`;`

  `tant que (`$x \neq t$`) faire` $\{$

    `pour tout successeur` y `de` x `faire`

      `si (`$D[y] > D[x] + v(x,y)$`)` $\{$

        $D[y] \longleftarrow D[x] + v(x,y)$`;`

        $F[y] \longleftarrow D[y] + h(y)$`;`

        $E \longleftarrow E \cup \{y\}$`;`

      $\}$

    $E \longleftarrow E - \{x\}$`;`

    $x \longleftarrow$ `un sommet de` E `pour lequel` $F[x]$ `est minimal;`

  $\}$

  `retour` $D[t]$`;`

$\}$