Nom : Kany Daphné Sujet : 9 Jury : Page : 1/3

Ne rien écrire hors du cadre.

Leçon 9 : Algorithmique du texte. Exemples et Applications
Niveau : MPII / MPI

Un auteur envoie son manuscrit à un éditeur. Il commence par compresser le texte (I). A la réception, l'éditeur compare le livre à d'autres pour vérifier le plagiat (II). Il peut ensuite chercher un motif dans le texte (III).

Notation 1 : On prend les conventions de slicing python sur les listes.

I - Compression

I.1 - Encodage par lettre

Définition 1 (Algorithme de compression) : Un algo. de compression sans perte est la donnée d'une fonction $f : \Sigma^* \to \Sigma^*$ bijective (il existe un unique décodage).

Algorithme 2 : Algorithme de Huffman :

1 - Prétraitement : On construit un arbre binaire $\mathcal{A}_H$ dont les feuilles sont les lettres $a \in \Sigma$.

2 - Compression : On code chaque lettre par la suite de 0 et 1 de son chemin dans $\mathcal{A}_H$ (gauche : 0, droite : 1) de la racine à la feuille correspondante.

3 - Décompression : On décode une suite de 0 et de 1 en parcourant le chemin correspondant dans $\mathcal{A}_H$.

Prétraitement 3 :
- créer une forêt F d'arbres binaires réduits à (a, f_a)
- Tant que F contient plus d'un arbre :
  - Extraire de F les arbres A_1 et A_2 de plus petites fréquences (($f_{A_1} \leq f_{A_2}$ à la racine
  - Insérer un arbre de racine $f_{A_1} + f_{A_2}$ ayant pour fils A_1 et A_2 de F
- Renvoyer le seul arbre de F

Théorème 4 : L'arbre construit par l'algo 3 minimise $S = \sum_{c \in \Sigma} f_c d_c$ où d_c est la profondeur de c dans l'arbre et f_c la fréquence de c dans le texte.

Exercice 5 : Quelle structure pour F dans le prétraitement 3 ? Quelle est alors la complexité ?

I.2 - Encodage par séquences

Ici, on associe un code à une séquence de lettres (ou motif).

Idée 6 : L'algorithme LZW détermine un codage dans un dictionnaire d au fur et à mesure de la lecture du texte. On associe

Algorithme 6 : LZW

Entrée : s
Initialement, les lettres de Σ sont codées par un entier (ex : code ASCII)
$m = ''$
Tant qu'il reste du texte s à coder :
  Retirer de s le plus long préfixe w présent dans d
  Ajouter le codage de w à la fin de m
  $w' \leftarrow w$ concaténé à la première lettre de s
  Ajouter un nouveau codage pour w' dans d
retourner m

Remarque 7 : Pour la décompression, il n'est pas nécessaire de transmettre le dictionnaire : on peut le reconstruire à la volée.

Application 8 : Le format de fichier .zip comprend un dossier en utilisant entre autre les algorithmes de Huffman et LZW.

II - Comparaison

II.1 - Plus Longue Sous Suite Commune (PLSSC)

Problème 8 : Soit $x, y \in \Sigma^*$. Une plus longue sous suite commune à x et y, notée PLSSC(x,y) est $\operatorname{argmin} \{ k \mid \exists i,j \; x[i,i+k] = y[j,j+k] \}$

Application 9 : Ce problème correspond au fait de trouver des morceaux d'ADN commun.

Exemple 10 : $x =$ 'AACGTAT' $y =$ 'GTAAGC'
PLSSC$(x,y) =$ 'GTA'

Algorithme M (Brute force) : On cherche toutes les séquences de x dans y → complexité exponentielle.

Algorithme 12 : (programmation dynamique)
On considère les sous pb $c_{ij} = |PLSSC(x[:i], y[:j])|$
Alors $c_{ij} = \begin{cases} 0 \text{ si } i=0 \text{ ou } j=0 \\ c_{i-1,j-1} + 1 \text{ si } x_i = y_j \\ \max(c_{i,j-1}, c_{i-1,j}) \text{ si } x_i \neq y_j \end{cases}$
$\Rightarrow$ Complexité en $O(|x||x||y|)$

Remarque 13 : Pour obtenir la valeur de la séquence, on stocke d'où on vient.

II - 2) Distance entre deux chaînes

Définition 14 : Une distance sur un ensemble E est une applicat° $d : E^2 \to \mathbb{R}^+$ tq :
$d(x,y) = d(y,x)$ (symétrie)
$d(x,y) = 0 \Rightarrow x = y$ (séparation)
$d(x,y) \leq d(x,z) + d(z,y)$ (inégalité triangulaire)

Définition 15 (Distance de Hamming) : La distance de Hamming entre deux chaînes de caractères de même taille est le nombre de caractères distincts.

Application 16 : Dans un protocole réseau, on peut ajouter des bits de contrôle ayant une information redondante avec les autres bits (ex : bit de parité...). Certains messages sont donc invalides. On prend alors le message valide ayant la plus petite distance de Hamming.

Exemple 17 : $d_H('truc', 'troc') = 1$.

Définition 18 : La distance d'édition (ou de Levenstein) entre deux chaînes de caractère est le nombre minimal de transformations pour passer de l'une à l'autre parmi :
- $ins_{a,i}$: insertion de a à la position i
- $sub_{a,i}$: substition de la ième lettre par i
- sup_i : supression de la ième lettre

Application 19 : On peut utiliser cet algorithme pour détecter des mutations possibles dans les brins d'ADN, et ainsi essayer de les faire correspondre.

Application 20 : Dans les logiciels de traitement de texte, les correcteurs orthographiques cherchent dans un dictionnaire le mot le plus proche de celui mal orthographié pour proposer une correction.

Propriété 21 : La distance d'édition est une distance.

Algo 22 : (Prog dyn)
$lev(u.a, v.b) = \min(lev(u,v) + \mathbb{1}_{a \neq b}, lev(u, vb) + 1, lev(ua, v) + 1)$

Dev 1 : Correction de l'algorithme 22.

III - Recherche de motifs

III.1 - Recherche de motifs dans un texte

Problème : Soit $m \in \Sigma^*$, $t \in \Sigma^*$. m est-il un sous-mot de t ?

Remarque 23 : On pourrait vouloir le nombre d'occurrence, l'emplacement.

Exemple 24 : Ctrl + F dans un fichier.

Solution naïve 25 : On essaie toutes les positions possibles pour m dans t. $O(|m| \times |t|)$

Idée 26 : On peut essayer de décaler

Nom : Sujet : Jury : Page : /

III - Recherche de motifs

III.1) Recherche d'un motif dans un texte

Problème : Pour $m \in \Sigma^*$, $t \in \Sigma^*$, m est-il un sous mot de t ? Où ?

Remarque : On pourrait demander toutes les occurences, leur nombre !

Exemple : Ctrl + F

Solution naïve : On essaie toutes les positions de m dans t.
$\Rightarrow O(|m| * |t|)$

Idée : On peut essayer de décaler de plus de 1 quand on se trompe.

Dev 2 : Utilisation d'un automate pour une recherche de motif en temps linéaire.

~~Idée~~

Algorithme (Boyer - Moore) : Amélioration de l'idée naïve.

```
i = 0
Tant que i < |t| - |m| :
    Pour j allant de |m|-1 à 0 :
        Si t[i+j] ≠ m[j] :
            i = i + decalage[t[i+j]]
            break
    retourner true
```

décalage[a] : indice en partant de la fin de la dernière occurence de a dans m (|m| si a non présent)

Complexité : pire des cas $O(|t| * |m|)$
meilleur des cas où $m \notin t$: $O(|t|/|m|)$

Idée : Dans l'algorithme de Rabin Karp, on compare directement $t[i : i + |m|]$ avec m grâce à des fonctions de hash. Si les hash sont égaux, on vérifie. Sinon on incrémente i de 1.

Remarque : le hash défini par $h(t_0 \dots, t_{|m|-1}) = \sum_{j=0}^{|m|-1} B^{|m|-1-j} * t_j$ peut se calculer en $O(1)$ à partir du calcul précédent.

Complexité : pire des cas : $O(|m| \times |t|)$ (collisions à chaque fois).
meilleur des cas : $O(|t|)$

III.2) Analyse lexicale

Application : Lors de la compilation d'un programme C, le compilateur reconnaît des lexemes définis par un langage régulier.

Exemple : $L_{float} = (0-9)^+ . ! . (0-9)^*$

Algorithme : Pour savoir si un texte t contient la regexp e, on construit l'automate reconnaissant $\Sigma^* e \Sigma^*$ et on vérifie si $t \in L(A)$.

Application : grep suivi d'une regexp en norme POSIX et de noms de fichier affiche toutes les lignes des fichiers qui contiennent la regexp.

Application : En SQL, on peut comparer des attributs de type CHAR avec des regexp grâce au mot clef LIKE

Exemple :

```
SELECT Prenom FROM clients
WHERE prenom LIKE '%.on%.'
```

Ne rien écrire hors du cadre