NOM : PECATTE **Prénom :** Timothée **Jury :**

Algèbre ← Entourez l'épreuve → **Analyse**

Sujet choisi : 907. Algorithmique du texte : exemples et applications.

Autre sujet :

0. Introduction, définitions basiques :

Σ un alphabet fini

Déf 1 : (Mots) Σ^*. On écrit $w \in \Sigma^*$ comme $w = w_1 \dots w_n$ avec $w_i \in \Sigma$

Déf 2 : Soit $w \in \Sigma^*$. On dit que $x \in \Sigma^*$ est :
- un préfixe de w ssi $\exists z \in \Sigma^*$ tq $w = xz$
- un suffixe de w ssi $\exists z \in \Sigma^*$ tq $w = zx$
- un facteur de w ssi $\exists y, z \in \Sigma^*$ tq $w = yxz$

Ex 3 : $w = abab$. $Pref(w) = \{\varepsilon, a, ab, aba, abab\}$
$Suf(w) = \{\varepsilon, b, ab, bab, abab\}$
$Fact(w) = \{\varepsilon, a, b, ab, ba, aba, bab, abab\}$

Déf 4 : (Taille) $|w| = n$ ssi $w = w_1 \dots w_n$
↳ w se code en $\Theta(n \log |\Sigma|)$, soit $\Theta(n)$ si Σ est supposé fixé.

I. Recherche d'un motif dans un texte :

I.1. Problème :

- SEARCH(m, t) :
  - Entrées : un motif u et un texte $t \in \Sigma^*$, $|u| = m$, $|t| = n$
  - Sortie : réponse à "$u \in Fact(t)$?"

Rmq 5 : Variante en-ligne

I.2. Algorithme naïf :

Algo 6.
```
POUR i ALLANT DE 1 À n-m+1 FAIRE
  | j ← 1
  | TANT QUE (j ≤ m et t_{i+j} = u_j) FAIRE
  |   L j ← j+1
  | SI (j = m+1) FAIRE
  L   L report d'une occurrence en i
```

Prop 7. Cet algorithme résout SEARCH(u, t) en temps $O(n.m)$

Prop 8 : Dans un modèle de Bernoulli, cet algorithme tourne en moyenne en $O(n)$.

I.3. Algorithme parallèle : SHIFT-AND

- On considère qu'une opération élémentaire est le "ET" bit à bit sur des registres machines de taille p ⤳ utile que si $m \leq p$

Déf 9 : (Masque) Étant donné u un mot, on définit l'entier $B_c(u)$, pour $c \in \Sigma$ par
le $i^{ème}$ bit de $B_c(u) = 1$ ssi $u_j = c$

Ex 10 : $u = ananas$. $B_a(u) = \overline{101010}^2$, $B_n(u) = \overline{010100}^2$, $B_s(u) = \overline{000001}^2$

Algo 11. SHIFT-AND(u, t)
```
PRE-CALCUL
 | POUR c ∈ Σ FAIRE B[c] ← 0^m
 L POUR j=1...m FAIRE B[u_j] ← B[u_j] | 0^{m-j}10^{j-1}
RECHERCHE
 | D ← 0^m
 | POUR i=1...n FAIRE
 |  | D ← ((D << 1) | 0^{m-1}1) & B[t_i]
 L  L SI D & 10^{m-1} ≠ 0^m ALORS report d'une occurrence en i
```

Prop 12 : SHIFT-AND résout SEARCH(u, t) en temps $O(n)$.

I.4. Algorithme KMP :

Déf 13 : (Bordure) Étant donné un mot $u \in \Sigma^*$, on dit que w est une bordure de u ssi w est un préfixe de u, et w est un suffixe de u, et $|w| < |u|$ (ie $w \neq u$).

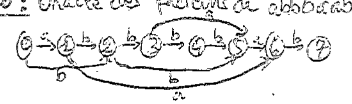
Appli 14. Périodes d'un mot

$\boxed{\text{DEV 1}}$ Algorithme KMP, preuve de la correction, de la terminaison et de la complexité

I.5. Algorithme à Oracle :

Déf 15 : (Oracle des facteurs) Étant donné $u = u_1 \dots u_m$, on construit l'automate suivant
- L'ensemble des états est $\{q_0, \dots, q_m\}$
- On a m transitions de la forme $q_{i-1} \xrightarrow{u_i} q_i$
- On a, pour chaque $i > 0$, une fonction de lien suffixiel, définie par récurrence par :
  - $S(0) = \theta$
  - $k := S(i-1)$, et on répète les étapes suivantes :
  - ST_1 : si $k = \theta$, $S(i) \leftarrow 0$
  - ST_2 : si $k \neq \theta$, et $(q_k, u_i, \varkappa) \notin \delta$, on rajoute (k, u_i, i) à δ, et on $k \leftarrow S(k)$
  - ST_3 : si $k \neq \theta$, et $(q_k, u_i, j) \in \delta$ pour un certain j, on s'arrête et on pose $S(i) \leftarrow j$.

Ex 16 : Oracle des facteurs de $abbbcab = u$



abc est reconnu mais abc $\notin Fact(u)$

Prop 17 : Soit $\mathcal{A}$ l'oracle des facteurs de u.
Alors $Fact(u) \subseteq \mathcal{L}(\mathcal{A})$

Rmq 18 : $S(i)$ = état d'arrivée du plus long suffixe qui est aussi facteur

Algo 19. BOM(u,t)

PRECALCUL
  $\delta \leftarrow$ fonction de transition de l'oracle des facteurs de $\tilde{u}$
RECHERCHE
  pos $\leftarrow 0$
  TANT QUE pos $\leq n-m$ FAIRE
    $q \leftarrow$ état initial de l'automate
    $j \leftarrow m$
    TANT QUE $j>0$ ET $q \neq \emptyset$ FAIRE
      $q \leftarrow \delta(q, t_{pos+j})$
      $j \leftarrow j-1$
    Si $j=0$ ALORS report d'une occurence à la position pos $+1$
    pos $\leftarrow$ pos $+ j + 1$

Prop 20. BOM(u,t) résout SEARCH(u,t) en temps $O(m.n)$ en pire cas, et en temps $O(\frac{n \log(m)}{m})$ dans un modèle de Bernoulli

I.6. Algorithme de Karp-Rabin:

Algo 21. KR(u,t)
  $a \leftarrow h(u)$
  POUR $i = 1 \dots n-m+1$ FAIRE
    Si $h(t[i \dots i+m-1]) = a$ ALORS
      Si $t[i \dots i+m-1] = u$ ALORS report d'une occurence à la pos i

Prop 22: KR(u,t) résout SEARCH(u,t) en temps $O(m.n)$ en pire cas

II. Indexation:

Soit t un texte fixé dans la suite.

II.1. Problèmes:

Pb 23. (Appartenance) Soit $u \in \Sigma^*$, trouver le plus long préfixe de $u \in$ Fact(t)

Pb 24. (Position) Soit $u \in$ Fact(t), déterminer la position de sa première (ou dernière) occurence dans t.

Pb 25. (#Occurences) Soit $u \in$ Fact(t), déterminer le nombre d'occurence de u dans t.

Pb 26. (Positions) Soit $u \in$ Fact(t), déterminer la liste des positions des occurences de u dans t.

Ex 27. $t = ababbabc$, $u = ab$
  Pos$(u,t) = \{1,3,6\}$, #Occurences $=3$, FirstPos $=1$, LastPos $=6$

II.2. Table des suffixes:

Def 28. La table des suffixes de t est un tableau contenant tous les suffixes de t par ordre lexicographique.

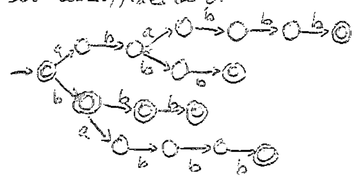
Prop 29: Cette table peut être construite en temps $O(n \log n)$

Prop 30: Étant donné la table des suffixes:
- Appartenance se résout en temps $O(m + \log n)$
- Position 〃 〃 〃 〃 $O(m + \log n)$
- #occurences 〃 〃 〃 〃 $O(m + \log n)$
- Positions 〃 〃 〃 〃 $O(m + \log n + \#occurences)$

II.3. Arbre des suffixes:

Def 31: (Trie) Le Trie d'un texte est un automate arborescent défini par:
- Les facteurs de t sont les états
- La fonction de transition δ est définie par $\delta(u,a) = ua$ si ua est un facteur de t.
- Les états terminaux sont les suffixes de t.

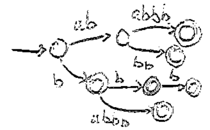
Ex 32: Trie$(ababbb)$



Prop 33: Le Trie des suffixes peut être construit en temps et en espace $O(n^2)$

Def 34: L'arbre des suffixe est obtenu en supprimant les nœuds de degré 1 non-terminaux du Trie des suffixes.

Ex 35: $T_c(ababbb)$:



Prop 36: T_c peut être stocké en $O(|u|)$.

Def 37. (lien suffixe) $\forall a \in \Sigma$, $ls(az) = z$ pour chaque état z de l'automate

DEV 2 Algorithme d'Ukkonen pour construire T_c en temps linéaire en-ligne

Def 38: $maxc(z) = \max\{|z| : z \in x^{-1} Suff(t)\}$
  $minc(x) = \min\{|z| : z \in x^{-1} Suff(t)\}$

Prop 39: first-pos$(x) = |t| - maxc(x) - |x|$
  last-pos$(x) = |t| - maxc(x) - |x|$

Prop 40: $T_c(t)$ pouvant être construit en $O(n)$, on peut répondre à Appartenance, Position, #occurrences, pour un mot x, en temps $O(|x| \times \log|A|)$ et en mémoire $O(n)$.

Prop 41: Si les états terminaux de $T_c(t)$ sont annotés par l'indice du suffixe correspondant, la liste des positions des occurrences de x peut se résoudre en temps $O(|x| \times \log|A| + \#\text{occurrences})$

III. Compression : codage de Huffman

III. 1. Codages préfixes

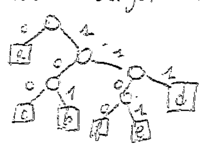
Déf 42: Un codage $\sigma : \Sigma \to A^*$ est dit préfixe si $\forall a \neq b \in \Sigma$, $\sigma(a)$ n'est pas préfixe de $\sigma(b)$.

Ex 43:

Σ	a	b	c	d	e	f
σ	0	101	100	111	1101	1100

Déf 43: (Arbre d'un codage) L'arbre d'un codage est défini comme le Trie des codes $\{\sigma(a), a \in \Sigma\}$

Ex 44:



Prop 45. L'arbre d'un codage préfixe optimal est complet.

Déf 46. (Coût) Pour $c \in \Sigma$, soit $f(c)$ la fréquence de c dans le texte, alors $|\sigma| = \sum_{c \in \Sigma} f(c)\, |\sigma(c)|$

III. 2. Construction glouton d'un codage préfixe optimal :

Algo 47: HUFFMAN(Σ) (avec $n = |\Sigma|$)

```
Q ← file de priorité vide
POUR i = 1...n FAIRE  INSERER(Q, Σi, f(Σi))
POUR i = 2...n FAIRE
  | z ← créer un nouveau noeud
  | gauche[z] ← EXTRAIRE-MIN(Q)
  | droite[z] ← EXTRAIRE-MIN(Q)
  | INSERER(Q, z, f(gauche[z]) + f(droite[z]))
retourner EXTRAIRE-MIN(Q)
```

Lemme 48: Soit un ensemble de caractère et $x, y \in \Sigma$ les deux caractères les moins fréquents. Il existe un codage optimal préfixe dans lequel $|\sigma(x)| = |\sigma(y)|$ et $\sigma(x), \sigma(y)$ ne diffèrent que par leur dernier bit

Lemme 49: La construction d'un codage préfixe optimal vérifie la propriété de sous-structure optimale.

Prop 50: L'algorithme HUFFMAN(Σ) produit un codage préfixe optimal.

IV. La notion de similitude : un aperçu

IV. 1. Plus longue sous-séquence commune :

Déf 51: (Sous-séquence) Pour $u, v \in \Sigma^*$, on dit que u est un sous-mot/une sous-séquence de v s'il existe $(i_j)_{j \leq |u|}$ tq $u_j = v_{i_j}$ $\forall j \leq |u|$.

Ex 52: BCDB est un sous-mot de ABCBABD

Pb 53. (PLSC) Étant donné $u, v \in \Sigma^*$, trouver une plus longue sous séquence commune à u et v

Prop 54: PLSC peut être résolu par programmation dynamique en temps $O(|u| \cdot |v|)$

IV. 2. Distance d'édition :

Pb 55: Étant donné $u, v \in \Sigma^*$, et des opérations sur des mots de Σ^*. Trouver le nombre minimal d'opération pour passer de u à v

Pb 56: Variante de Pb 55 avec opérations pondérées

Ex 57: Opérations : ajouter une lettre, supprimer une lettre, changer une lettre. On peut passer de algorithme à altruiste en 6 opérations

Prop 58: L'exemple 57 peut être résolu par prog. dyn en temps $O(|u| \cdot |v|)$.

Rem 59: L'alignement de séquence se réduit au pb 57

IV. 3. Recherche de motif approché :

Pb 60: Étant donné un motif u et un texte t, on cherche toutes les occurrences de u dans t ayant une distance d'édition d'au plus k, pour k fixé

Prop 61: Le Pb 60 peut être résolu par prog dyn. en temps $O(|u| \cdot |t|)$.

Réfs:
- Cormen : Huffman et KMP et prog dyn
- Navarro : Flexible Pattern matching in strings. Algo de recherche, comparatif, recherche approchée
- Crochemore : Text Algorithm. Algo d'Ukkonen. Huffman notamment.
- Crochemore : Algorithms on strings. Le reste
- Éléments d'algorithmique : chap 10

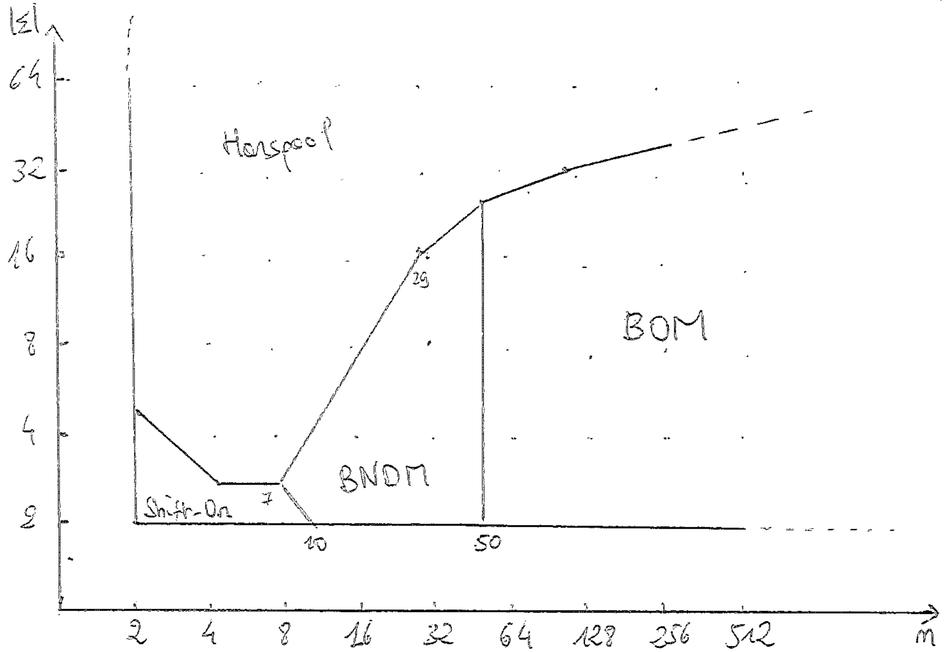


Figure 1: Comparatif des algorithmes de recherche

	B	D	C	A	B	A
A	0	0↑	0↑	↖1	←1	↖1
B	↖1	←1				
C			↖2	←2		
B					↖3	
D					↑3	3↑
A					3↑	↖4
B					↖4	4↑

Figure 2, Construction de la plus longue sous-séquence commune.