Algorithmique avancée – Examen Réparti 2
UPMC — Master d'Informatique —
Janvier 2016 – durée 2h

Les seuls documents autorisés sont les polys de cours, ainsi que la copie double personnelle.
Le barême donné est indicatif.

1 QCM [5 points]

Cocher une seule réponse par question, et argumentez votre choix.

Q1. La récurrence $T(n) = T(n/3) + \Theta(1)$ a pour solution
A) $T(n) = \Theta(n)$
B) $T(n) = \Theta(n \log n)$
C) $T(n) = \Theta(\log n)$
Donner un exemple d'algorithme qui a cette complexité.
sol C : recherche trichotomique dans un tableau trié

Q2. En utilisant un splay-tree de n clés, on peut réaliser un tri des n clés
A) en temps moyen $O(n \log n)$
B) en temps amorti $O(n \log n)$
C) en temps pire $O(n \log n)$
sol B splay-tree avec remontee à la racine de l'ememement cherché (inséré, supprimé) à chaque operation

Q3. Soit $T = (ab)^{1000}$ le texte constitué de la répétition de mille fois ab. Pour compresser T, la meilleure méthode est
A) Huffmann statique
B) LZW
C) Shannon-Fano
sol B LZW qui va stocker des repetitions de ab Avec Huffman : $a : 0$, $b : 1$***, d'où un texte compressé de 2000 bits, donc 250 octets.***
Avec LZW, le texte se décompose en $a - b - (ab) - (ab)a - (ba) - (ba)b - (ab)^2 - (ab)^2a - (ba)^2 - (ba)^2b - (ab)^3 - (ab)^3a - (ba)^3 - (ba)^3b - \ldots - (ab)^k - (ab)^ka - (ba)^k - (ba)^kb - \ldots$***. On a besoin de :***
- 1 code pour le premier a***,***
- 1 code pour le premier b***,***
- pour les 1998 caractères suivants, 4 codes pour chaque facteur $(ab)^k.(ab)^ka.(ba)^k.(ba)^kb$***. Chacun des facteurs est de longueur*** $8k+2$***, le nombre de facteurs est majoré par*** n ***tel que :*** $\sum_{k=1}^{n-1}(8k+2) \leq 1998 \leq \sum_{k=1}^{n}(8k+2)$***. La valeur de*** n ***est 23 et le nombre de codes nécessaires pour le texte est majoré par*** $2 + 4 * 23 = 94$***. Avec LZW, 94 octets suffisent pour compresser*** $(ab)^{1000}$***.***

Q4. On représente un code préfixe complet par
A) les feuilles d'un trie binaire
B) les noeuds internes d'un trie binaire
C) les feuilles et les noeuds internes d'un trie binaire
sol A feuilles car code prefixe

Q5. Dans la compression JPEG, l'image est découpée en blocs de 8×8 pixels. Si l'on découpe en blocs de 16×16, le nombre de multiplications effectuées lors de la transformée cosinus discrète matricielle est
A) divisé par 2
B) multiplié par 2
C) multiplié par 4

sol B car sur des blocs de taille B on a $2 \times B^3$ ***multiplications pour calculer . Sur une image de taille I, le nombre de multilications est*** $I/(B * B) * 2B^3 = 2IB$***, donc multiplié par 2 si B passe de 8 à 16***

2 Exercice [5 points]

a) On considère un texte formé de 5 lettres a, b, c, d, e, de fréquences respectives $1, 1, 1, 2, 3$.
Donner 2 arbres de Huffmann construits sur ce texte : l'un de hauteur 3 et l'autre de hauteur 4.
sol (((a. b) . (c.d)). e) et ((((a. b) .c).d).e)

Quel est le poids de ces arbres (poids $(H) = \sum f(\nu)\text{prof}(\nu)$; ν feuille de H) ?
sol 18
Que signifie ce poids au niveau de la compression ?
sol *c'est la taille du texte compressé*
b) On considère la suite (g_k) définie par $g_0 = g_1 = g_2 = 1$ et pour $k \geq 3, g_k = g_{k-1} + g_{k-2}$.
Le texte T_k contient $k+1$ lettres qui ont pour fréquences $g_0, \ldots, g_k$.
Ecrire un algorithme qui construit un arbre de Huffman de hauteur k pour T_k.
sol *A := arbrebin*(g_0, g_1) *; Pour i = 2 à k A := arbrebin*(A, g_i) *; return A*
Montrer par récurrence que le poids d'un arbre de Huffmann pour T_k est $g_{k+4} - 3$ (on rappelle que $\sum_0^k g_i = g_{k+2}$).
sol vrai pour k= 2 : poids = 5 = 8-3 On travaille sur l'arbre de Huffmann H_k ***de hauteur*** k ***On suppose vrai pour k et on montre vrai pour*** $k+1$
le poids de H_k ***est*** $g_{k-1} + 2g_{k-2} + 3g_{k-3} + \cdots + kg_0$ ***(se démontre facilement par récurrence) On a*** $H_{k+1} =$ ***arbrebin***(H_k, g_{k+1}) ***donc le poids de*** H_{k+1} ***est égal à*** g_{k+1} ***plus le poids de*** H_k ***dans lequel toutes les profondeurs sont augmentées de 1*** $= g_{k+1} + poids(H_k) + g_0 + \cdots + g_k$***. Par hypothèse de récurrence on a donc*** $poids(H_{k+1}) = g_{k+1} + g_{k+4} - 3 + g_0 + \cdots + g_k = g_{k+4} - 3 + g_{k+3} = g_{k+5} - 3$

3 Problème [10 points]

On considère ici une méthode de compression qui utilise le fait que certains mots sont absents du texte à compresser. On se place dans un alphabet binaire {0, 1}. Le principe de la compression est le suivant : si on sait par exemple que le facteur 1011 n'apparaît pas dans le texte, cela veut dire qu'à chaque fois que le facteur 101 est présent, il sera suivi par 0, et l'on peut donc omettre ce 0.
Soit un texte T, on note $F(T) = \{w \in \{0,1\}^*; \exists A, B \in \{0,1\}^* \text{ avec } T = AwB\}$ l'ensemble des *facteurs* de T. Un ensemble de *non-facteurs* de T, $NF(T)$ est un ensemble de mots qui ne sont pas dans T, c'est-à-dire un sous-ensemble (fini ou non) du complémentaire de $F(T)$. Cet ensemble peut être donné en extension ou décrit par une propriété. Par exemple pour $T = 0101$, on peut avoir $NF(T) = \{00; 11\}$ ou alors $NF(T) =$ l'ensemble des mots contenant deux lettres 1 à la suite.

Question 1. Soit $T = 100101$. Que vaut $F(T)$? Donner deux ensembles $NF(T)$ différents pour T.
$F(T) = \{0, 1, 01, 10, 00, 001, 010, 100, 101, 0010, 0101, 1001, 10010, 00101, 100101\}$ ***et on peut prendre*** $NF(T) = \{011, 110\}$ ***ou*** $NF(T) =$***l'ensemble des mots qui ne contiennent pas deux 1 à la suite ou ...***

On suppose que l'ensemble $NF(T)$ des non-facteurs du texte T est fourni ; on compresse T dans un parcours de gauche à droite, en éliminant toutes les lettres pouvant être déduites des lettres précédentes à l'aide de $NF(T)$: supposons que l'on a déjà compressé le préfixe v de T ($T = vv'$). S'il existe dans $NF(T)$ un mot $u = u'x$, $x \in \{0,1\}$, tel que u' est un suffixe de v, alors la lettre suivant v est nécessairement $\bar{x}$ (si x vaut 0 alors $\bar{x}$ vaut 1, et inversement). Cette lettre est inutile dans le texte compressé car elle peut être déduite grâce à $NF(T)$. On élimine de cette manière toutes les lettres pouvant être déduites des lettres précédentes à l'aide de $NF(T)$.
Par exemple la compression du texte $T = 01101010$ avec $NF(T) = \{00; 111; 1011\}$ aboutit au texte compressé $TC = 01$, après avoir traité la suite des préfixes de T, comme montré ci-dessous .

préfixe v	TC	mot dans $NF(T)$
0	0	00
01	0	—
011	01	111
0110	01	00
01101	01	1011
011010	01	00
0110101	01	1011
01101010	01	

Question 2. Appliquer la méthode à la compression du texte $T = 01101010$ avec $NF(T) = \{00; 0111\}$. Faire le même type de tableau que ci-dessus.

v	TC	mot dans $NF(T)$
0	0	00
01	0	—
011	01	0111
0110	01	00
01101	01	—
011010	010	00
0110101	010	—
01101010	0100	

Question 3. Décrire, en pseudo-code, un algorithme de compression suivant la méthode précedente Vous expliciterez les primitives que vous utilisez en donnant leur spécification. Quelle est la complexité de cet algorithme (préciser le coût des primitives) ?

Fonction Compresser : Texte × Ensemble de Textes → (Entier × Texte)
Compresser(T, NF) renvoie la taille de T et le texte compressé à l'aide de NF
v ← vide ; y ← vide ; $\#v$ est le texte lu et y le texte compressé
while T ≠ vide **do**
  a ← prem(T) ;
  if ∀ u' ∈ Suff(v), **Non** estDans(u'0, NF) **et Non** estDans(u'1, NF) **then**
    y ← y.a ;
  end if
  v ← v.a ;
  T ← reste(T) ;
end while
return $(|v|, y)$

---

les fonctions prem(T) et reste(T) renvoient respectivement la première lette du texte T et le texte privé de sa première lettre. Le prédicat estDans(w,E) renvoie Vrai ssi le mot w est dans l'ensemble E. La fonction Suff(w) renvoie l'ensemble des suffixes du mot w.
Le texte a taille n***. On suppose que les primitives prem(T), reste(T), estDans(w,E) sont en*** $O(1)$ ***et que la taille du plus long mot de*** $NF(T)$ ***est*** k***. Alors Complexité :*** $O(nk)$***.***

Inversement, pour décompresser un texte TC à partir de l'ensemble de non facteurs $NF(T)$ qui a été utilisé lors de la compression, on construit la suite des préfixes du texte initial T : à partir de la première lettre de TC, on retrouve, à l'aide de $NF(T)$, les lettres qui suivent dans T ; quand on ne peut plus deviner la lettre suivante de T, on examine la prochaine lettre dans le texte compressé TC et l'on applique le même processus ; et ainsi de suite jusqu'à épuisement des lettres de TC.
Par exemple pour décompresser $TC = 01$ à l'aide de $NF(T) = \{00; 111; 1011\}$, on procède comme suit :

préfixe v	TC	mot dans $NF(T)$
0	01	
01	1	00
011	1	on ajoute le dernier bit
0110	ε	111
01101	ε	00
011010	ε	1011
0110101	ε	00
01101010	ε	1011

Question 4. Appliquer l'algorithme pour décompresser le texte TC obtenu en question 2, à l'aide de $NF(T) = \{00; 0111\}$.

préfixe v	TC	mot dans $NF(T)$
0	0100	
01	100	00
011	00	on ajoute 2eme bit de TC
0110	00	0111
01101	00	00
011010	0	on ajoute 3eme bit
0110101	0	00
01101010	ε	on ajoute dernier bit

Question 5. Expliquer pourquoi dans l'algorithme de décompression il faut aussi fournir la *taille* du texte initial T.

Pour l'unicité de la décompression, par exemple si $NF(T) = \{00, 111, 1011\}$ ***et le texte compressé est*** 01***, si on ne fournit pas la taille du texte*** T***,*** 01 ***peut se décompresser en*** 011 ***ou en n'importe quel texte de la forme*** 011010101010.....***.***

Question 6. Décrire, en pseudo-code, un algorithme de décompression suivant cette méthode, et donner sa complexité (en précisant le coût des primitives).

Fonction Décompresser : Entier × Texte × Ensemble de Textes → Texte
Décompresser(NF, y, n) renvoie le texte T décompressé à partir de y à l'aide de NF
$v \leftarrow$ vide ;
while $|v| < n$ **do**
  if $\exists u' \in \text{Suff}(v)$; estDans$(u'a, NF)$ avec $a \in \{0, 1\}$ **then**
    $v \leftarrow v.\bar{a}$
  else
    $b \leftarrow \text{prem}(y)$; $y \leftarrow \text{reste}(y)$;
    $v \leftarrow v.b$;
  end if
end while
return (v)

Complexité : $O(nk)$***.***

Question 7. Il s'agit maintenant de trouver une structure de données pour représenter l'ensemble $NF(T)$. Quelle est l'opération sur laquelle cette structure de données doit être efficace ?

Sol : Pour tout mot v***, il faut pouvoir trouver s'il existe un mot*** $u' \in$ ***Suff(***v***) tel que*** $u'a \in NF(T)$***, avec*** $a \in \{0, 1\}$***.***

Proposer une représentation de l'ensemble $NF(T)$; analyser la complexité des opérations sur cette structure.

Sol : on peut representer $NF(T)$ ***par un trie. Ensuite pour chaque suffixe*** u' ***de*** v ***on vérifie si*** $u'a$ ***est dans le trie. La hauteur du trie est la longueur*** k ***du plus long mot de*** $NF(T)$***. Donc pour chaque suffixe de taille au plus*** k***, on verifie s'il est dans le trie (au plus*** k ***comparaisos de bits).***
La complexité de l'algorithme en comparaisons de bits est donc $O(n \times k^2)$***.***