Algorithmique IV (UE41) - Licence d'Informatique

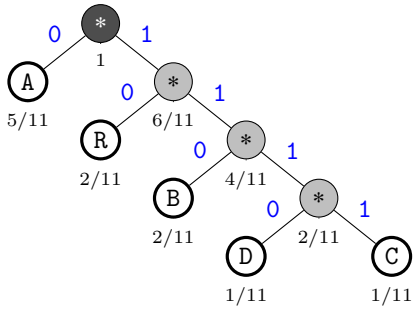
Contrôle terminal (Session 1 - Mai 2025)

Cette correction est bien plus plus détaillée que ce que j'exigeais dans les réponses, mais il fallait *a minima*, que les arguments clefs soient évoqués pour obtenir des points.

Présentation du problème

Le *codage de Huffman* est un algorithme de compression de texte. Chaque symbole $x_i \in \mathscr{A}$ d'un mot $x = x_1 x_2 \dots x_n$ où $\mathscr{A} = \{a_1, a_2, \dots, a_q\}$ est un alphabet fini, est codé par une séquence binaire $h(x_i)$ où la fonction $h : \mathscr{A} \to \{0,1\}^*$ est déterminée à partir des fréquences des symboles dans le texte : plus a est fréquent, plus son code $h(a)$ est court. Le codage de x est la concaténation des séquences $h(x_i)$.

Le code de Huffman $h(a)$ du symbole a est la séquence formée par les étiquettes binaires des branches — 0 à gauche et 1 à droite —, sur le chemin qui relie la racine d'un arbre binaire valué, à la feuille a :



a	$p_a \approx$	$h(a)$
A	5/11	0
B	2/11	110
C	1/11	1111
D	1/11	1110
R	2/11	10

Fig. À gauche : arbre des symboles `A,B,C,D,R` basé sur leur fréquence dans `ABRACADABRA`. À droite : code de Huffman.

Exemple : l'encodage du mot `CAR` est la séquence binaire `1111010` obtenue par la concaténation $\texttt{1111} \cdot \texttt{0} \cdot \texttt{01}$.

Principe de construction de l'arbre de Huffman

Le principe est très simple :

(*a*) On calcule la fréquence de chacun des q symboles de $\mathscr{A}$. Exemple : $\mathscr{A} = \{\texttt{A}, \dots \texttt{Z}\}$ avec le texte $x = \texttt{ABRACADABRA}$ dans la table. Seuls les symboles dans le texte seront codés.

(*b*) On initialise une forêt avec des arbres réduits à leur racine constituée par chacun des symboles de $\mathscr{A}$ présent dans le texte et valué par sa fréquence (ce sont les 5 feuilles cerclées de gras dans la figure).

(*c*) Tant qu'il reste au moins 2 arbres :

- chercher les deux dont la racine a la plus petite valuation (`C` et `D` dans notre exemple)
- créer un nœud père dont le fils gauche est celui qui a la plus petite fréquence, et le fils droit la deuxième plus petite fréquence, le valuer par la somme de leurs fréquences ($\frac{2}{11} = \frac{1}{11} + \frac{1}{11}$ pour `C` et `D`, les pères sont distingués sur fond gris par un symbole $\notin \mathscr{A}$, une $*$ ici).

Le sujet porte sur la construction de l'arbre de Huffman et la compression/décompression d'un texte grâce à cette méthode.

Préliminaires (6 pts)

Question 1. Soit $c : \mathscr{A} \to \{0,1\}^*$ un encodage des symboles de $\mathscr{A}$. Pour que l'on puisse décoder $c(a)$ en a pour tout $a \in \mathscr{A}$, il faut nécessairement que c soit injective. Montrez que l'encodage de Huffman h est injectif.

Réponse. Par construction, les symboles présents dans le texte sont rangés dans les feuilles de l'arbre et leurs codes sont définis par les chemins (uniques) qui les relient à la racine. Si deux chemins sont distincts, ils se séparent nécessairement sur un nœud interne et les étiquettes étant distinctes suivant la branche empruntée, leurs séquences binaires sont distinctes. La fonction h est donc injective.

Question 2. Supposons que c encode les symboles de $\mathscr{A}$ par des séquences de même longueur ℓ. Exprimez formellement cette propriété. Quelle est la longueur minimale pour que c soit injective ? Comparez la longueur totale du texte `ABRACADABRA` encodé par un encodage c de longueur minimale à celle de ce même texte encodé par h avec $\mathscr{A} = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{R}\}$.

Réponse. Aucune difficulté particulière :

$$\forall a \in \mathscr{A} \quad |c(a)| = \ell.$$

ou encore

$$\exists \ell \in \mathbb{N}\ \forall a \in \mathscr{A} \quad |c(a)| = \ell$$

si l'on souhaitait inclure la longueur ℓ dans l'expression.

On sait qu'une condition nécessaire pour qu'une application $f : X \to Y$ entre deux ensembles finis de cardinaux respectifs n et m soit injective est $n \leqslant m$. Dans notre cas, puisque l'encodage est de longueur constante ℓ, on a $c : \mathscr{A} \to \{0,1\}^\ell$ et il faut donc que

$$|\mathscr{A}| \leqslant 2^\ell.$$

La fonction logarithme étant croissante, on en déduit que

$$\log_2(|\mathscr{A}|) \leqslant \ell.$$

Le cardinal étant un nombre entier, on conclut que la longueur minimale est $\ell := \lceil \log_2(|\mathscr{A}|) \rceil$, c'est-à-dire le plus petit entier supérieur à $\log_2(|\mathscr{A}|)$.

Pour $x :=$ `ABRACADABRA`, l'alphabet est constitué de 5 lettres, on a donc $\ell = 3$, ce qui donne un encodage du texte x de longueur 33 puisque $|x| = 11$. Avec l'encodage de Huffman, le texte codé est

`01101001111011100110100`

qui n'est plus que de longueur 23.

Question 3. Montrez que l'injectivité d'un encodage c de longueur minimale constante ℓ est une condition suffisante pour décoder un texte. Montrez que cette condition n'est plus suffisante en général si c est de longueur variable. Exhibez un contre-exemple avec un texte x et un encodage c de votre choix sur l'alphabet $\mathscr{A}$ de l'exemple.

Réponse. En effet, il suffit de segmenter le texte codé en mots de longueur ℓ et de décoder chaque mot à l'aide de c^{-1} puisque c est injective. En revanche si le code est de longueur variable, quand bien même c est injective, on ne peut plus déterminer quelle segmentation réaliser, par exemple pour le code de longueur variable définit par $c($`A`$) :=$ `0`, $c($`R`$) :=$ `1`, $c($`C`$) :=$ `01`, qui est manifestement injectif, l'encodage du mot `CAR` est

$$\overset{\texttt{C}}{\texttt{01}}.\overset{\texttt{A}}{\texttt{0}}.\overset{\texttt{R}}{\texttt{1}} = \overset{\texttt{C}}{\texttt{01}}.\texttt{01} = \texttt{0}.\overset{\texttt{A}}{\texttt{1}}.\texttt{0}.\texttt{1} = \texttt{0}.\texttt{1}.\texttt{01}$$

et peut être décodé en `CC`, `ARAR` ou encore `ARC`. L'injectivité de c ne suffit donc pas pour décoder le texte.

Question 4. Plus généralement, justifiez qu'aucun des codes de Huffman n'est le préfixe d'un autre code pour un alphabet $\mathscr{A}$ quelconque. Expliquez comment décoder un texte encodé par h à l'aide de l'arbre.

Réponse. Par construction, chaque symbole a de $\mathscr{A}$ est situé dans une feuille de l'arbre de Huffman. Tout préfixe strict de $h(a)$ suit donc un chemin issu de la racine qui s'achève nécessairement sur un nœud interne codant le symbole $*$. Un code de Huffman ne peut-être qu'un préfixe donc et ne code donc jamais un symbole de Aucun préfixe ne s'a

Le décodage s'ensuit immédiatement : la séquence binaire codant le texte x est lue de gauche à droite en suivant simultanément le chemin correspondant dans l'arbre en partant de la racine ; à chaque fois que le chemin atteint une feuille, on décode le symbole et on repart de la racine jusqu'à avoir consommé toute la séquence binaire.

Exemple : l'encodage du texte $x =$ `BAR` pour le code de Huffman en introduction est la séquence binaire

$$\overset{\texttt{B}}{\texttt{110}}.\overset{\texttt{A}}{\texttt{0}}.\overset{\texttt{R}}{\texttt{10}} = \texttt{110010}$$

En partant de la racine, on suit donc deux branches droites puis une gauche ($\overrightarrow{\texttt{110}}\texttt{010}$), ce qui aboutit à la feuille `B`. On repart donc de la racine en suivant une branche gauche ($\texttt{110}\overrightarrow{\texttt{0}}\texttt{010}$) et on aboutit à la feuille `A`. On repart de la racine, on suit une branche droite puis une gauche ($\texttt{110010}$) pour finir le décodage sur la feuille `R`.

Question 5. Définissez formellement la fréquence p_i du symbole a_i de $\mathscr{A}$ dans le texte $x = x_1 x_2 \dots x_n$.

Réponse. C'est le ratio entre le nombre d'occurrences du symbole a_i et la longueur $n := |x|$ du texte x.

$$\forall i \in [\![1, |\mathscr{A}|]\!] \quad p_i := \frac{\#\{j \in [\![1,n]\!] \mid x_j = a_i\}}{n}.$$

Question 6. Écrivez un algorithme `Frequences(x)` qui renvoie la fréquence d'apparition des symboles de $\mathscr{A}$ dans x, c'est-à-dire la liste

$$[p_1, p_2, \dots, p_q].$$

Vous utiliserez sans l'écrire, la fonction `ord(a)` qui renvoie l'ordre du symbole a dans $\mathscr{A}$ définie par $\forall i \in [\![1, |\mathscr{A}|]\!]$ `ord`$(a_i) = i$.

Réponse. L'alphabet est considéré implicitement comme une variable globale, on pouvait bien sûr en faire un paramètre de l'algorithme.

```
ALGORITHME Frequences(x):liste
DONNEES
   x: mot
VARIABLES
   F: liste de réels
   n: entier
DEBUT
   Allouer(F, |A|, 0)
   n ← |x|
   i ← 1
   TQ i <= n FAIRE
      F[ord(x[i])] ← F[ord(x[i])] + 1/n
      i ← i + 1
   FTQ
   RENVOYER F
FIN
```

Question 7. Calculez sa complexité en temps.

Réponse. La taille n de l'instance à traiter est bien sûr la longueur $|x|$ du texte à encoder. Comme souvent dans ce type d'algorithmes, la taille q de l'alphabet $\mathscr{A}$ est considéré comme une constante. Il n'y a pas de meilleur ou de pire des cas, toutes les instances de taille n ont le même coût. L'allocation coûte $\Theta(q)$, les deux instructions avant la boucle et le retour de `F` ont un coût constant $\Theta(1)$, le coût de la mise à jour de `F` dans la boucle est en $\Theta(1)$ et il y en a n. Le coût total est donc

$$\Theta(q) + \Theta(1) + n.\Theta(1) = \Theta(n)$$

Construction de l'arbre avec un tas minimal (12 pts)

Un *tas* maximal (resp. minimal) est un tableau T dont l'arbre binaire associé est partiellement ordonné : la valuation de chaque nœud est supérieure (resp. inférieure) à ses fils. Le tas minimal constitué par les symboles de $\mathscr{A}$ présents dans le texte et valué par leur fréquence, fournit le symbole de fréquence minimale en $\Theta(1)$ puisqu'il est à la racine de l'apo.

Un tableau T est initialisé avec les références des quadruplets (a, p, g, d) pour chaque symbole a de $\mathscr{A}$ présent dans x où p est sa fréquence, g et d sont les références vers des quadruplets. La référence d'un quadruplet $q = (a, p, g, d)$ est codée $\gg q$, la référence vide $\boxslash$.

Dans la suite vous utiliserez sans les écrire les algorithmes suivants et le tableau `F` :

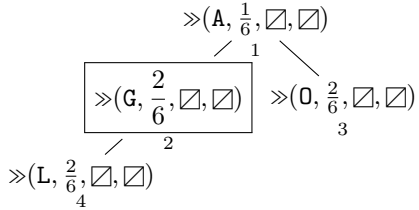
- `Tamiser(@T, ipere, ifin)` : tamise le sous-arbre d'indice `ipere` jusqu'à l'indice `ifin`.
- `Entasser(@T)` : transforme un tableau T en tas minimal.
- `ord(x)` : renvoie l'ordre du symbole x, i.e. $\texttt{ord}(a_i) = i$.
- `F` : fréquences des symboles de $\mathscr{A}$ où `F[i]` est la fréquence de a_i, calculées par l'algorithme `Frequences`.

Question 8. Illustrez la transformation du tableau T en tas pour le texte $x =$ `ALGOGO`.

Réponse. La fréquence des symboles `A` et `L` est de $\frac{1}{6}$, celles des symboles symboles `G` et `O` est de $\frac{2}{6}$. La valeur initiale du tableau T est donc

$$[\gg(\texttt{A}, \frac{1}{6}, \boxslash, \boxslash), \gg(\texttt{G}, \frac{2}{6}, \boxslash, \boxslash), \gg(\texttt{L}, \frac{1}{6}, \boxslash, \boxslash), \gg(\texttt{O}, \frac{2}{6}, \boxslash, \boxslash)].$$

et l'arbre associé est représenté ci-dessous. Le premier nœud d'indice $2 := \lfloor \frac{4}{2} \rfloor$ auquel l'algorithme de tamisage est appliqué pour transformer le tableau en tas est encadré :



Comme on peut le constater, la fréquence du fils gauche (symbole `L` de fréquence $\frac{2}{6}$) du nœud d'indice 2 (symbole `G` de fréquence $\frac{2}{6}$) n'étant pas strictement inférieure, aucun échange n'a lieu. L'algorithme de tamisage remonte ensuite au nœud d'indice 1 (symbole `A` de fréquence $\frac{1}{6}$) et encore une fois aucun échange n'a lieu puisque sa fréquence est inférieure à celle de ses deux fils `G` et `O`. Le tableau T avait déjà une structure de tas minimal.

Question 9. En rajoutant un élément à la fin d'un tas minimal, il perd son statut de tas si la valuation de cet élément est inférieure à celle de son père. On peut rétablir la propriété APO en faisant remonter cet élément le long du chemin qui le relie à la racine. Écrivez un algorithme `Remonter(@T,i)` qui restaure la propriété APO de T en remontant le nœud i le long de la branche qui relie la racine de l'arbre à i tant que c'est nécessaire.

Réponse. Il suffit de comparer la fréquence du nœud avec celle de son père et les échanger si nécessaire en recommençant avec le père tant que nécessaire :

```
ALGORITHME Remonter(@T,i)
DONNEES
   T: liste de quadruplets
   i: entier
VARIABLES
   pere: entier
DEBUT
   pere ← i DIV 2
   TQ (pere > 0) ET ((T[i]»p) < (T[pere]»p)) FAIRE
      Echanger(T,i,pere)
      i ← pere
      pere ← i DIV 2
   FTQ
FIN
```

Notez que le codage choisi pour accéder aux champs du quadruplet n'avait pas d'importance du moment que c'était compréhensible.

Question 10. Justifiez informellement que cet algorithme est correct et calculez sa complexité.

Réponse. Avant l'insertion de l'élément d'indice i à la fin du tableau T, ce dernier avait une structure de tas. Si la propriété APO n'est plus satisfaite après l'introduction du nœud d'indice i, c'est exclusivement entre lui et son père d'indice $p := \lfloor \frac{i}{2} \rfloor$. Le cas échéant, en échangeant les deux valeurs, la propriété est rétablie en i mais le problème est potentiellement déplacé sur le père p. On recommence donc la comparaison sur le nœud p et ainsi de suite, au pire jusqu'à la racine.

NB. Une récurrence finie permettrait de formaliser ce raisonnement. La justification de terminaison de l'algorithme se fait grâce à la condition $p > 0$ qui ne sera nécessairement plus satisfaite par un terme de la suite strictement décroissante des valeurs de p, si l'autre condition n'arrête pas la boucle entre temps.

Le nombre de comparaisons et d'échanges est borné supérieurement par $h-1$ si h désigne la profondeur de l'APO. Nous savons qu'un arbre binaire équilibré à q nœuds a pour profondeur $h := \lfloor \log_2(q) \rfloor$. Ici le nombre de nœuds est le nombre de symboles de l'alphabet $\mathscr{A}$. Dans le meilleur des cas (l'introduction du nouveau nœud ne viole pas la propriété APO), la complexité est en $\Theta(1)$), dans le pire des cas $\Theta(\log(q))$ qui est souvent confondu avec $\Theta(1)$ puisque la taille de l'alphabet est supposé constante.

Question 11. Écrivez un algorithme `Huffman(x):quadruplet` qui renvoie la racine de l'arbre de Huffman sur la base du texte x :

(1) Calculer les fréquences des symboles de l'alphabet.
(2) Initialiser T avec les références des quadruplets.
(3) Transformer T en tas minimal.
(4) Tant que le tas T n'est pas réduit à un élément :
   (a) Récupérer le 1er quadruplet minimal $qg \leftarrow T[1]$.
   (b) Le remplacer par le dernier et tamiser T.
   (c) Récupérer le 2ème quadruplet minimal $qd \leftarrow T[1]$.
   (d) Le remplacer par le dernier et tamiser T.
   (e) Rajouter le quadruplet `(*,qg[2]+qd[2],qg,qd)` à la fin de T.
   (f) Le remonter pour rétablir la propriété APO.
(5) Renvoyer le premier quadruplet du tas.

NB. Le tas T perd donc un élément en (a), un autre en (c) et en gagne un en (f).

Réponse. Dans l'algorithme `Huffman`, l'entier t permet de connaître la taille du tas au fur et à mesure des créations des nœuds pères. Pour éviter de parcourir la table des fréquences inutilement pour déterminer la taille initiale du tableau T pour son allocation, à savoir le nombre de symboles différents présents dans le texte, on suppose que t est calculé par l'algorithme `Frequences`. L'algorithme auxiliaire `Init` initialise le tableau T avec les t références de quadruplets associés aux symboles du texte et renvoie sa taille t.

NB. Le symbole `#` code la référence vide ⍁ dans l'algorithme `Init`.

```
ALGORITHME Huffman(x):quadruplet
DONNEES
   x: mot
VARIABLES
   T: liste de réf. de quad.
   t: entier
   qg,qd: réf. de quad.
DEBUT
   t ← Init(T,x)
   Entasser(T)
   TQ (t > 1) FAIRE
      qg ← T[1]
      T[1] ← T[t]
      t ← t - 1
      Tamiser(T,1,t)
      qd ← T[1]
      T[1] ← T[t]
      t ← t - 1
      Tamiser(T,1,t)
      t ← t + 1
      T[t] ← »(*,(qg»p)+(qd»p),qg,qd)
      Remonter(T,t)
   FTQ
   RENVOYER(T[1])
FIN
```

```
ALGORITHME Init(@T,x):entier
DONNEES
   T: liste de réf. de quad.
   x: mot
VARIABLES
   F: liste de réels
   i,t: entiers
DEBUT
   (F,t) ← Frequences(x)
   Allouer(T,t)
   t ← 0
   i ← 1
   POUR TOUT a € A FAIRE
      SI (F[ord(a)] > 0) ALORS
         t  ← t + 1
         T[t] ← »(a,F[i],#,#)
      FSI
      i ← i + 1
   FPOUR
   RENVOYER t
FIN
```

Question 12. Calculez sa complexité.

Réponse. Commençons par l'algorithme `Init`. Le calcul des fréquences a un coût de $\Theta(n)$, avec $n = |x|$. L'allocation du tableau a un coût de $\Theta(t) = \Theta(1)$ puisque $t \leqslant q$ et que q est supposée constant. L'initialisation de ses valeurs a un coût de $\Theta(q) = \Theta(1)$. Le coût global de cet algorithme est donc $\Theta(n)$.

Passons à l'algorithme `Huffman`. Comme nous l'avons vu en cours, l'algorithme `Entasser` a une complexité linéaire en la taille du tableau à transformer en tas, donc en $\Theta(t) = \Theta(1)$ ici. L'extraction et la mise à jour de la racine ont une complexité en $\Theta(1)$ et le tamisage en $\Theta(\log(t))$ dans le pire des cas. La création du nœud père a un coût de $\Theta(1)$ et sa «remontée» $\Theta(\log(t))$ dans le pire des cas. Chaque passage dans la boucle a donc un coût de $\Theta(\log(t))$. Le coût total de la boucle dans le pire des cas est donc [1]

1. ce calcul a été effectué à plusieurs reprises en cours et en TD.

$$\sum_{i=1}^{t} \Theta(\log(i)) = \Theta(t\log(t)) = \Theta(1).$$

Finalement la complexité globale dans le pire des cas de l'algorithme `Huffman` est

$$\Theta(n).$$

NB. On peut bien entendu donner une information plus précise sur le coût de cet algorithme en conservant les différentes complexités dépendant de la taille de l'alphabet, comme s'il elle n'était pas considérée comme une constante. Cet perte d'information est justifiée concrètement par le fait que la compression en général n'a d'intérêt que lorsqu'on a une très grande quantité d'information à transmettre/stocker, et la taille de l'alphabet est réellement négligeable par rapport à la taille du texte.

Question 13. Expliquez, sans écrire d'algorithme, comment utiliser T pour créer la fonction h et pour reconstituer un texte ainsi compressé.

Réponse. Une fois que l'arbre de Huffman a été construit, il suffit de parcourir toutes ses branches qui sont stockées dans les quadruplets de T. Pour ce faire, on part de la racine et on descend systématiquement sur la gauche à chaque nœud rencontré. Quand une feuille est atteinte, on remonte jusqu'au premier nœud qui possède une bifurcation sur la droite qui n'a pas encore été explorée et on emprunte cette branche droite en reprenant la même règle «toujours à gauche». Les codes des symboles sont obtenus en conservant à jour durant la visite de l'arbre, la séquence binaire codant la position courante dans l'arbre qui est rangée dans la table à chaque fois qu'une feuille est atteinte. La méthode de décompression du texte a été fournie dans l'énoncé.