2. Problème d'algorithmique

11 - Une procédure récursive s'écrit sans problème : si $k = 1$, elle renvoie 0 ; sinon, elle calcule l'indice i du nœud dont le poids p_i est minimal parmi les $k-1$ premiers nœuds : si le k-ème nœud est de poids strictement inférieur à p_i, elle renvoie $k-1$; sinon, elle renvoie i. Remarquons au passage qu'en cas d'égalité, cette fonction renvoie le plus petit indice donnant un nœud de poids minimal.

```
let rec indice_du_min F k = match k with
  1 -> 0
  | _ -> let i=indice_du_min F (k-1) in
    if F.table.(k-1).poids < F.table.(i).poids then
      k-1
    else
      i ;;
```

12 - L'analyse est une nouvelle fois élémentaire :

- on note k le nombre d'arbres de la forêt ;
- on calcule l'indice i de la racine de poids minimal : $i =$ `indice_du_min T k` ;
- si $i \neq k-1$, on échange les nœuds d'indices i et $k-1$;
- on calcule l'indice j de la racine "d'avant dernier" plus petit poids : $j =$ `indice_du_min T` $(\mathtt{k}-1)$;
- si $j \neq k-2$, on échange les nœuds d'indices j et $k-2$.

S'il y a des égalités entre les poids, la fonction `indice_du_min` choisit bien les racines de plus petits indices.

L'échange de deux nœuds se fait par le biais de la fonction auxiliaire :

```
let echange T i j = let N = T.(i) in
  T.(i) <- T.(j) ;
  T.(j) <- N ;;
```

ce qui donne :

```
let deux_plus_petits F = let k = F.nb_arbres in let i = indice_du_min F k in
  if i <> k-1 then echange F.table i (k-1) ;
  let j = indice_du_min F (k-1) in
    if j <> k-2 then echange F.table j (k-2) ;;
```

13 - L'assemblage se fait sans problème :

- on note k et n les nombres d'arbres et de nœuds de la forêt ;
- on place les deux plus petits arbres en dernières positions, i.e. aux positions $k-2$ et $k-1$;
- on modifie les champs `nb_arbres` et `nb_noeuds` (on ajoute un nœud et on regroupe deux arbres) ;
- on copie dans la case `F.table.(n)` le contenu de la case `F.table.(k-2)` : ce nœud ne sera plus une racine dans le nouvel arbre ;
- on définit le nouveau nœud, que l'on stocke dans la case `F.table.(k-2)` : ses fils gauche et droit sont respectivement les nœuds d'indices $k-1$ et n et son poids est la somme des poids de ses fils.

```
let assemblage F = let k = F.nb_arbres and n = F.nb_noeuds in
   deux_plus_petits F ;
   F.nb_arbres <- k-1 ;
   F.nb_noeuds <- n+1 ;
   F.table.(n) <- F.table.(k-2) ;
   F.table.(k-2) <- { lettre = `\000` ;
                      poids = F.table.(k-1).poids+F.table.(n).poids ;
                      fg = k-1 ;
                      fd = n } ;;
```

14 - Supposons qu'un nœud interne N n'ait qu'un fils N'. En "remontant" N' à la place de N, on obtient un nouvel arbre $\mathcal{A}'$ tel que $e(\mathcal{A}') < e(\mathcal{A})$. En effet, notons $\mathcal{F}$ l'ensemble des feuilles de $\mathcal{A}$ appartenant à l'arbre enraciné en N. Une feuille f de $\mathcal{A}$ étant également une feuille de $\mathcal{A}'$, notons $h_{\mathcal{A}}(f)$ et $h_{\mathcal{A}'}(f)$ les hauteurs de f respectivement dans $\mathcal{A}$ et $\mathcal{A}'$. Nous avons alors :

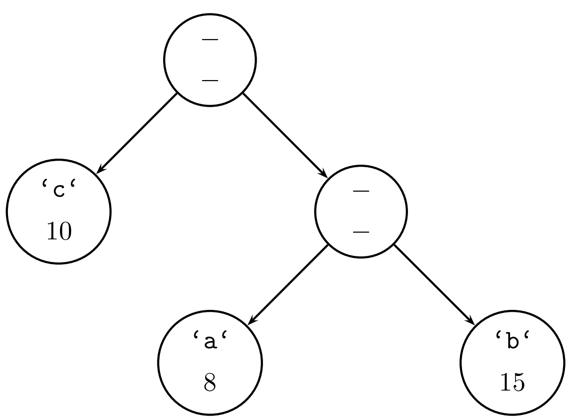
$$\forall f,\ h_{\mathcal{A}'}(f) = \begin{cases} h_{\mathcal{A}}(f) - 1 & \text{si } f \in \mathcal{F} \\ h_{\mathcal{A}}(f) & \text{sinon} \end{cases}$$

ce qui donne :

$$e(\mathcal{A}') = e(\mathcal{A}) - \sum_{f \in \mathcal{F}} poids(f) < e(\mathcal{A})$$

puisque $\mathcal{F}$ est non vide : nous avons prouvé par contraposée que les nœuds internes d'un arbre optimal ont tous deux fils. Nous dirons d'un tel arbre qu'il est *complet*.

Dans l'exemple proposé, un seul nœud interne a un fils unique : le gain est égal à $8 + 15 = 23$.



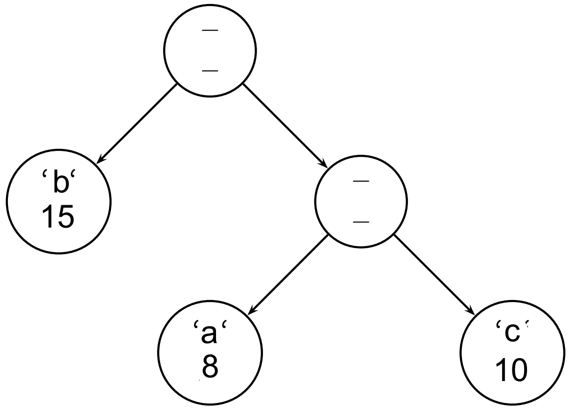
L'arbre $\mathcal{B}$_ex

15 - En échangeant les feuilles f_1 et f_2, nous obtenons un arbre $\mathcal{A}'$ pour lequel :

$$e(\mathcal{A}') = e(\mathcal{A}) - \underbrace{(h_{\mathcal{A}}(f_1) - h_{\mathcal{A}}(f_2))}_{>0}(poids(f_1) - poids(f_2))$$

Comme $\mathcal{A}$ est optimal, $e(\mathcal{A}') \geq e(\mathcal{A})$ et donc $poids(f_1) \leq poids(f_2)$.

On obtient directement, dans l'exemple proposé, un gain égal à 56-51 = 5.



L'arbre $\mathcal{C}$_ex

16,17 - Nous pouvons répondre en même temps aux question 16 et 17 : l'ensemble des arbres complets de même feuilles que $\mathcal{A}$ est un ensemble fini. Choisissons donc un arbre $\mathcal{A}'$ minimal parmi ceux-ci. Il est alors clair que $\mathcal{A}'$ est un arbre minimal : si $\mathcal{B}$ est un arbre possédant les mêmes feuilles que $\mathcal{A}'$, en "remontant" les nœuds qui sont fils uniques, on peut construire un arbre complet $\mathcal{B}'$ de mêmes feuilles que $\mathcal{A}'$ et $e(\mathcal{A}) \leq e(\mathcal{B}') \leq e(\mathcal{B})$. Soit alors f_1' une feuille de $\mathcal{A}$ de hauteur maximale. Si $f_1 \neq f_1'$, on échange ces deux feuilles : l'arbre est toujours optimal et complet (car $poids(f_1) \leq poids(f_1')$) et f_1 est une feuille de hauteur maximale. Comme l'arbre est complet, le père de f_1 possède alors deux fils : par maximalité de la hauteur de f_1, le second fils est également une feuille f_2'. En échangeant cette feuille et f_2, on obtient un nouvel arbre complet qui reste optimal, puisque $poids(f_2) \leq poids(f_2')$. Nous avons ainsi montré l'existence d'un arbre optimal (complet) de mêmes feuilles que $\mathcal{A}$ dans lequel f_1 et f_2 sont sœurs et de hauteur maximale.

18 - Notons n le père de f_1 et f_2 et $\mathcal{F}$ l'ensemble des feuilles de $\mathcal{A}$ distinctes de f_1 et f_2. Nous avons directement :

$$e(\mathcal{A}) = \sum_{f \in \mathcal{F}} h(f)poids(f) + (h(n)+1)(poids(f_1) + poids(f_2)) = e(\mathcal{B}) + p_1 + p_2.$$

19 - Supposons que $\mathcal{A}$ est optimal et soit $\mathcal{B}'$ de mêmes feuilles que $\mathcal{B}$. Notons $\mathcal{A}'$ l'arbre obtenu en ajoutant f_1 et f_2 pour filles à la feuille n de $\mathcal{B}'$. $\mathcal{A}'$ possède alors les mêmes feuilles que $\mathcal{A}$ et la question précédente donne :

$$e(\mathcal{B}') = e(\mathcal{A}') - p_1 - p_2 \geq e(\mathcal{A}) - p_1 - p_2 = e(\mathcal{B})$$

ce qui prouve que $\mathcal{B}$ est optimal.

Supposons maintenant que $\mathcal{B}$ est optimal et soit $\mathcal{A}'$ un arbre optimal de mêmes feuilles que $\mathcal{A}$, tel que f_1 et f_2 soient sœurs (un tel arbre existe d'après la question 17). Notons $\mathcal{B}'$ l'arbre obtenu en simplifiant $\mathcal{A}'$ par coupe de f_1 et f_2. Nous avons une nouvelle fois :

$$e(\mathcal{A}') = e(\mathcal{B}') + p_1 + p_2 \geq e(\mathcal{B}) + p_1 + p_2 = e(\mathcal{A})$$

et $\mathcal{A}$ est optimal, puisque son évaluation est inférieure ou égale à celle d'un arbre optimal de mêmes feuilles.

20 - La seule façon de décoder le texte est : $1000011011 = (100)(00)(11)(011)$ donc $T = face$.

21 - On vérifie facilement que le codage donné en exemple est préfixe. Si C est un codage préfixe, l'application $C : T \mapsto C(T)$ est injective : c'est le moins que l'on puisse demander à un codage ! En effet, si $C(T) = C(T')$

avec $T \neq T'$, en prenant le premier caractère qui différencie T de T', on obtient deux lettres distinctes dont l'un des code est préfixe de l'autre.

L'autre intérêt est qu'il est possible, au moyen de l'arbre défini dans la suite du problème, de décoder le message $C(T)$ “sans retard” : on part de la racine de l'arbre et on descend à gauche ou à droite selon que le caractère courant de $C(T)$ est un 0 ou un 1. Dès que l'on arrive à une feuille, la lettre contenue dans cette feuille est le premier caractère de T. On remonte à la racine et on répète l'opération jusqu'à lecture complète de $C(T)$.

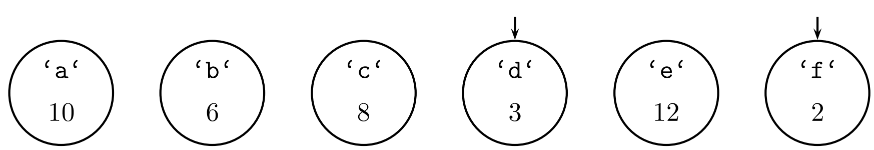
22 - $h(f)$ est exactement le nombre de caractères du code de la lettre associée à f, donc l'évaluation du H_arbre associé à un code préfixe C est la longueur du texte codé $C(T)$.

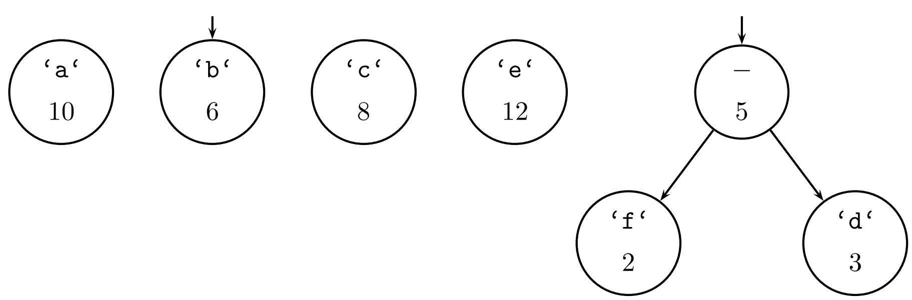
23 - Montrons la propriété par récurrence sur la taille de l'alphabet.

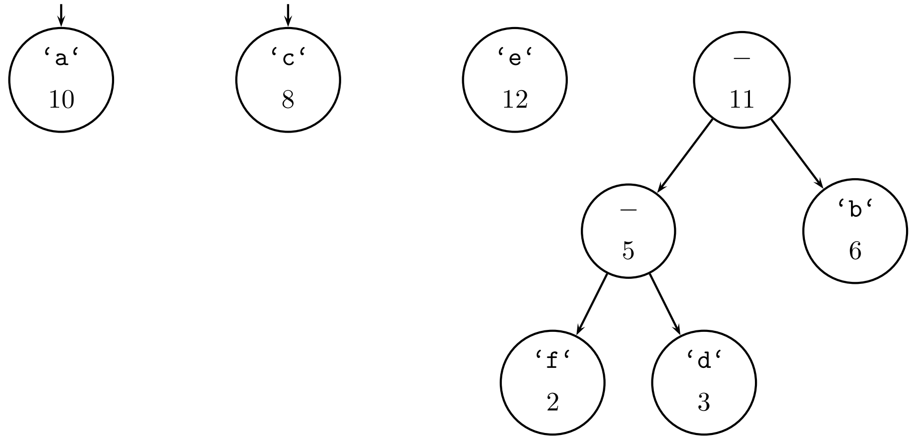
Si Σ est de cardinal 2, l'algorithme de Huffman donne clairement un arbre optimal.

Soit $n \geq 3$ et supposons que l'algorithme de Huffman conduise à un arbre optimal pour tout alphabet pondéré de cardinal $n-1$. Soit Σ un alphabet pondéré de cardinal n et notons $\mathcal{A}$ l'arbre construit par l'algorithme de Huffman quand on l'applique à Σ. Cet algorithme commence par “regrouper” deux lettres de plus petits poids, que nous noterons a_1 et a_2, en un nouveau caractère, que nous noterons α. La suite de l'algorithme revient alors à travailler sur le nouvel alphabet $\Sigma' = (\Sigma \setminus \{a_1, a_2\}) \cup \{\alpha\}$, en associant le poids $poids(a_1) + poids(a_2)$ à la nouvelle lettre α. Si nous notons f_1 et f_2 les feuilles de $\mathcal{A}$ associées aux caractères a_1 et a_2 (ces deux feuilles sont sœurs), l'arbre $\mathcal{B}$ obtenu en simplifiant $\mathcal{A}$ par coupe de f_1 et f_2 est exactement l'arbre construit par l'algorithme de Huffman appliqué à l'alphabet Σ'. Par hypothèse de récurrence, cet arbre est optimal : on en déduit que $\mathcal{A}$ est également optimal d'après la question 19.

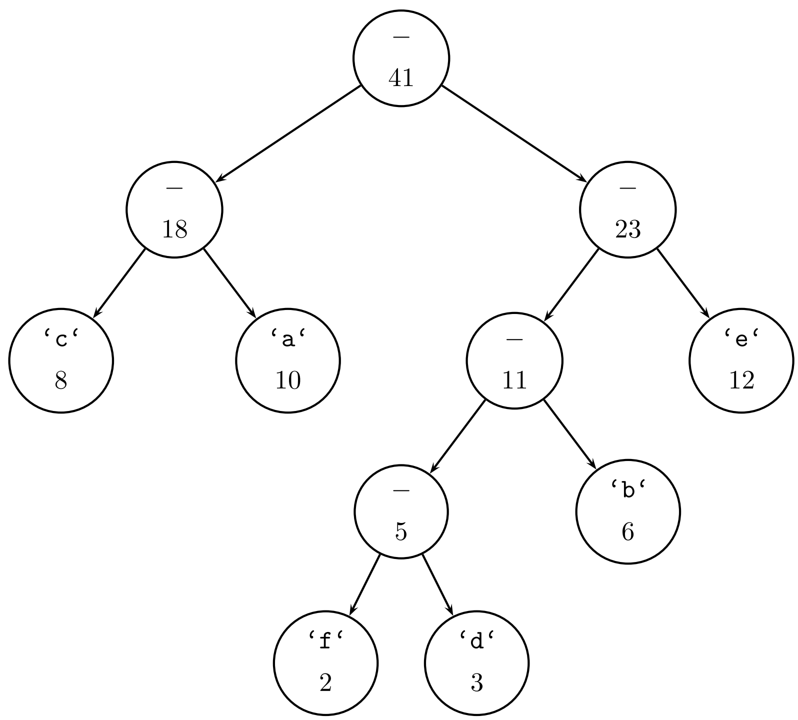
24 - Nous obtenons successivement les forêts suivantes (les deux arbres de plus petits poids sont repérés par une flèche) :







et ainsi de suite, jusqu'à obtenir l'arbre final :



Le codage de Huffman pour cet alphabet pondéré est donc :

caractère	code
‘a‘	01
‘b‘	101
‘c‘	00
‘d‘	1001
‘e‘	11
‘f‘	1000