

Corrigé pour serveur UPS par JL. Lamard (jean-louis.lamard@prepas.org)

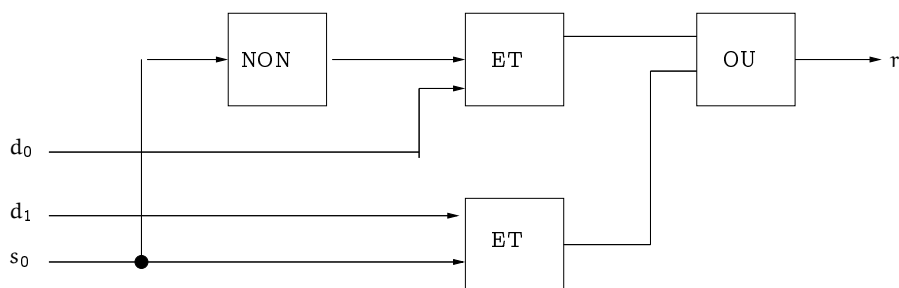
1. Problème de logique.

Question 1.

Il suffit de remplacer la porte "ET" du circuit synthétisant la constante 0 par une porte "OU" pour synthétiser la constante 1.

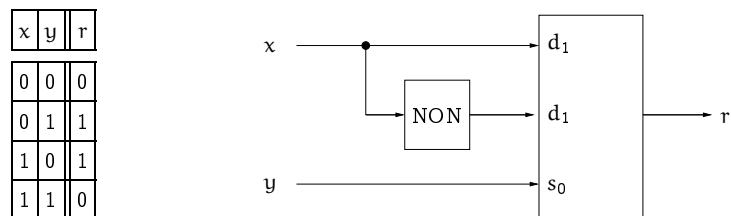
Questions 2 et 3.

Si $s_0 = 0$ alors $r = d_0$ et sinon $r = d_1$ donc $r = \overline{s_0}d_0 + s_0d_1 = (\overline{s_0} \wedge d_0) \vee (s_0 \wedge d_1)$



Questions 4 et 5.

On reconnaît la fonction "xor" : $r = x \oplus y = \overline{x}y + x\overline{y} = (\overline{x} \wedge y) \vee (x \wedge \overline{y})$



Question 6.

On peut réaliser la fonction logique ou de la même manière que dans la question 4 en fixant les entrées d_0, d_1, d_2 et d_3 respectivement à 0, 1, 1, 1. □

Pour un concentrateur M_1 on a vu que (avec les notations classiques du et et ou logique) : $r = \overline{s_0}d_0 + s_0d_1$.

Donc si $s_0 = x, d_1 = x$ et $d_0 = y$ il vient $r = \overline{x}y + xx = \overline{x}y + x$ qui est bien égal $y + x$ car r vaut 0 si et seulement si $x = 0$ et $\overline{x}y = 0$ i.e. si et seulement si $x = y = 0$.

Ainsi on peut bien réaliser la fonction logique ou avec un seul concentrateur M_1 (sans aucune autre porte, mais avec un nœud duplicateur ce qui est bien permis par l'énoncé) en envoyant x sur s_0 et d_1 et y sur d_0 . □

Questions 7 et 8.

Il est immédiat que toute fonction booléenne à n variables peut être réalisée à l'aide d'un concentrateur M_n :

Il suffit pour cela d'envoyer x_1 sur l'entrée s_0, \dots, x_n sur l'entrée s_{n-1} puis de fixer les entrées d_i de manière adéquat. De manière précise d_k avec $k = \sum_{j=0}^{n-1} y_j 2^j$ est fixée à $f(y_0, y_1, \dots, y_{n-1})$ en notant $(y_0, y_1, \dots, y_{n-1})$ l'une des 2^n distributions de vérité de (x_1, x_2, \dots, x_n) .

Pour l'exemple de la question 4, la fonction est nulle sauf pour les distributions $(1, 0, 1), (0, 1, 1)$ et $(1, 1, 1)$ qui correspondent en binaire à (attention la lecture des bits se fait de droite à gauche) 5, 6 et 7. Il suffit donc de fixer les entrées de 0 à 4 à la valeur 0 et les entrées de 5 à 7 à la valeur 1.

Question 9.

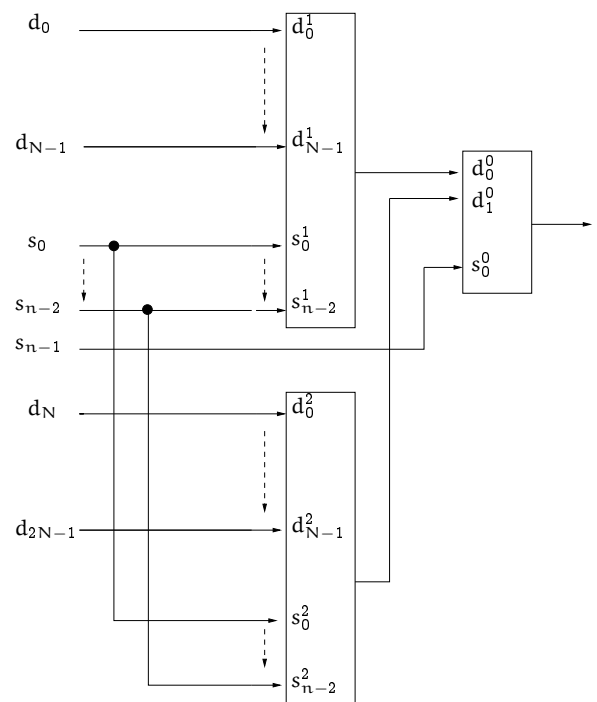
En remarquant, en liaison avec la question précédente, que :

$$f(x_1, x_2, \dots, x_n) = \overline{x_n} \underbrace{f(x_1, \dots, x_{n-1}, 0)}_{f_1(x_1, x_2, \dots, x_{n-1})} + x_n \underbrace{f(x_1, \dots, x_{n-1}, 1)}_{f_2(x_1, x_2, \dots, x_{n-1})}$$

ou (ce qui revient au même) en notant que (avec les notations de l'énoncé bas page 2) :

$$k = \sum_{j=0}^{n-1} x_j 2^j = \sum_{j=0}^{n-2} x_j 2^j \text{ ou } \sum_{j=0}^{n-2} x_j 2^j + 2^{n-1} \text{ suivant que } x_n = 0 \text{ ou } 1, \text{ on obtient, compte tenu}$$

de l'équation logique du concentrateur M_1 , le schéma de montage suivant :



Question 10.

Il vient immédiatement que $\alpha_n = 2\alpha_{n-1} + 4$ donc (suite arithmético-géométrique fixant -4) $\alpha_n + 4 = 2^{n-1}(\alpha_1 + 4)$ soit $\alpha_n = 4(2^n - 1)$.

2. Problème d'algorithmique et de programmation.

Première partie : fonctions de base pour l'algorithme de Huffman.

Question 11.

```
let indice_du_min F k =
  if (k > F.nb_noeuds) or (k <= 0)
  then failwith "Erreur dans indice_du_min"
  else let min = ref 0 in
    for i = 1 to k-1 do
      if F.table.(i).poids < F.table.(!min).poids then min := i else ()
    done;
    !min
;;
indice_du_min : Forêt -> int -> int = <fun>
```

Question 12.

```
#let échange v i j =
  if i <> j then let temp = v.(i) in
    v.(i) <- v.(j);
    v.(j) <- temp
  else ()
;;
échange : 'a vect -> int -> int -> unit = <fun>

#let deux_plus_petits F =
  if F.nb_arbres < 2 then failwith "Erreur dans deux_plus_petits !"
  else
    let min = indice_du_min F F.nb_arbres in
      échange F.table (F.nb_arbres - 1) min;
    let min = indice_du_min F (F.nb_arbres - 1) in
      échange F.table (F.nb_arbres - 2) min
;;
deux_plus_petits : Forêt -> unit = <fun>
```

Question 13.

L'algorithme est clair :

- 1/ on place les deux racines de plus petit poids en dernières positions (indices $a-1$ et a) de la partie réservée aux racines,
- 2/ on recopie la case d'indice $a-1$ en la case d'indice n (où n est le nombre de nœuds) qui contenait initialement le nœud vide,
- 3/ on modifie la case d'indice $a-1$ en lui attribuant la lettre '-', le poids somme des poids des deux racines de plus petit poids au départ, comme fils gauche le nœud d'indice a et comme fils droit celui d'indice n ,
- 4/ il reste à décrémenter le nombre de racines et à incrémenter le nombre de nœuds.

```
#let assemblage F =
  let a = F.nb_arbres-1 and n = F.nb_noeuds in
  deux_plus_petits F;
  F.table.(n) <- F.table.(a-1);
  F.table.(a-1) <- {lettre = '-';
    poids = F.table.(a).poids + F.table.(a-1).poids;
    fg = a;
    fd = n};
  F.nb_arbres <- a;
  F.nb_noeuds <- n+1
;;
assemblage : Forêt -> unit = <fun>

Vérification avec l'exemple donné (on prend ici MAX=10) :

let forêt1 =
  let table_forêt1 = make_vect MAX noeud_vide in
  table_forêt1.(0) <- {lettre = 'g'; poids = 10; fg = -1; fd = 6};
  table_forêt1.(1) <- {lettre = 'a'; poids = 20; fg = 4; fd = 3};
  table_forêt1.(2) <- {lettre = 'b'; poids = 13; fg = -1; fd = -1};
  table_forêt1.(3) <- {lettre = 'e'; poids = 9; fg = 5; fd = -1};
  table_forêt1.(4) <- {lettre = 'f'; poids = 15; fg = -1; fd = -1};
  table_forêt1.(5) <- {lettre = 'c'; poids = 12; fg = -1; fd = -1};
  table_forêt1.(6) <- {lettre = 'd'; poids = 8; fg = -1; fd = -1};
  {nb_arbres = 3; nb_noeuds = 7; table = table_forêt1}
;;

assemblage forêt1;;
- : unit = ()

forêt1;;
- : Forêt =
{nb_arbres = 2; nb_noeuds = 8;
table =
[|{lettre = 'a'; poids = 20; fg = 4; fd = 3};
 {lettre = '-'; poids = 23; fg = 2; fd = 7};
 {lettre = 'g'; poids = 10; fg = -1; fd = 6};
 {lettre = 'e'; poids = 9; fg = 5; fd = -1};
 {lettre = 'f'; poids = 15; fg = -1; fd = -1};
 {lettre = 'c'; poids = 12; fg = -1; fd = -1};
 {lettre = 'd'; poids = 8; fg = -1; fd = -1};
 {lettre = 'b'; poids = 13; fg = -1; fd = -1};
 {lettre = '\000'; poids = 0; fg = -1; fd = -1};
 {lettre = '\000'; poids = 0; fg = -1; fd = -1}]|}
```

Deuxième partie : propriétés pour l'algorithme de Huffman.

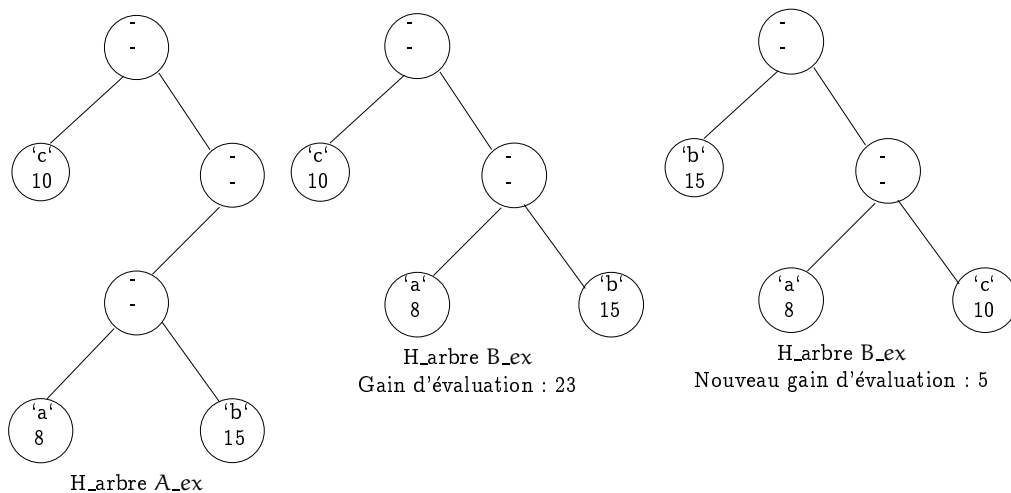
Question 14.

Soit un H_arbre A optimal. Supposons que A possède un noeud interne n admettant un seul fils m (pouvant être une feuille ou un noeud interne). Alors en remplaçant n par m on obtient un H_arbre ayant les mêmes feuilles et d'évaluation strictement plus petite (la hauteur a diminué d'une unité pour toutes les feuilles issues de n donc le gain d'évaluation est $p > 0$ où p est la somme des poids (strictement positifs) des feuilles issues du noeud n). Contradiction avec le fait que A soit minimal.

Ainsi un arbre de Huffman optimal est complet.

Question 15.

Immédiat sinon en échangeant les feuilles f_1 et f_2 on obtiendrait un arbre B de mêmes feuilles que A et d'évaluation strictement plus petite, ce qui est en contradiction avec le fait que A soit optimal.



Question 16 et 17.

Soit un H_arbre A et deux feuilles f_1 et f_2 de plus petits poids avec $\text{poids}(f_1) \leq \text{poids}(f_2)$. Soit B un H_arbre optimal de même feuilles (donc complet).

Notons qu'un tel arbre existe bien car l'ensemble des évaluations des H_arbres de mêmes feuilles que A est un ensemble non vide d'entiers naturels donc admet un plus petit élément.

- Supposons que f_1 ne soit pas de hauteur maximale dans B et soit alors f'_1 une feuille de B de hauteur maximale.

La contribution des feuilles f_1 et f'_1 à l'évaluation de B est $h(f_1)\text{poids}(f_1) + h(f'_1)\text{poids}(f'_1)$. En échangeant ces deux feuilles on obtient un arbre B' pour lequel la contribution est $h(f_1)\text{poids}(f'_1) + h(f'_1)\text{poids}(f_1)$.

Donc $e(B) - e(B') = (h(f_1) - h(f'_1))(\text{poids}(f_1) - \text{poids}(f'_1)) \geq 0$ puisque f_1 est de poids minimal et f'_1 de hauteur maximale.

Comme B est optimal, cette quantité est nulle. Ainsi B' est également optimal et f_1 en est bien une feuille de hauteur maximale.

Ainsi il existe toujours un arbre optimal C (soit B soit B') dont f_1 est une feuille de hauteur maximale.

- Comme C est complet (car optimal) le père de f_1 admet un autre fils f'_2 qui est un nœud de même hauteur que f_1 donc maximale donc qui est une feuille (sinon il admettrait une feuille descendante de hauteur strictement plus grande). Si f_2 est différente de f'_2 , le même raisonnement que précédemment montre qu'en échangeant les feuilles f_2 et f'_2 on obtient un arbre C' encore optimal.
- En conclusion, il existe toujours un arbre optimal de mêmes feuilles que A dans lequel f_1 et f_2 sont deux feuilles sœurs et de hauteur maximale.

Question 18.

Il est immédiat que $e(B) = e(A) - (p_1 + p_2)$.

Question 19.

- Supposons A non optimal et soit A' de mêmes feuilles et d'évaluation strictement plus petite. Soit B' obtenu à partir de A' par coupe de f_1 et f_2 . Alors B' a les mêmes feuilles que B et pour évaluation $e(A') - (p_1 + p_2) < e(B)$ d'après la question précédente. Ce qui prouve que B n'est pas optimal.
- Supposons B non optimal et soit B' de mêmes feuilles et d'évaluation strictement plus petite. Soit A' obtenu à partir de B' en remplaçant la feuille n par un nœud ayant pour fils les feuilles f_1 et f_2 . Alors A' a les mêmes feuilles que A et pour évaluation $e(A') = e(B') + (p_1 + p_2) < e(A)$ d'après la question 18. Ce qui prouve que A n'est pas optimal.
- Ainsi A est optimal si et seulement si B l'est.

Troisième partie : l'algorithme de Huffman.

Question 20.

Le seul découpage possible est (100)(00)(11)(011) donc le texte est "face".

Question 21.

On vérifie facilement que le codage donné en exemple est préfixe.

Le fait que le codage soit préfixe implique clairement que l'application $C : T^* \rightarrow C(T)$ est injective donc que le décodage est possible !

Sinon par exemple avec "a" codé en 0, "b" codé en 1 et "c" en 01, les textes "ab" et "c" auraient le même code !

Question 22.

Il est immédiat que la longueur de $C(T)$ est égale à $e(A)$ où A est le H_arbre représentant le codage.

Question 23.

Soit A l'arbre produit par l'algorithme de Huffman à partir de la forêt constituée des feuilles des caractères "occurencés" du texte T. Pour prouver que le codage associé à cet arbre est optimal, il suffit, compte tenu de la question précédente, de prouver que A est optimal parmi les H_arbres ayant pour feuilles les feuilles qui constituent la forêt de départ.

On raisonne pour cela par récurrence sur le nombre k de feuilles de la forêt de départ.

La propriété est évidemment vraie pour $k = 2$. Supposons qu'elle soit vraie pour $k \leq n - 1$ (avec $n \geq 3$) et considérons une forêt F de n feuilles.

Soient f_1 et f_2 les deux feuilles de plus petit poids de F (au sens de la question 12) et soit F' la forêt de $n - 1$ feuilles composée des mêmes feuilles que F sauf f_1 et f_2 remplacée par une unique feuille α de poids $\text{poids}(f_1) + \text{poids}(f_2)$.

Par hypothèse de récurrence l'arbre A' obtenu à partir de la forêt F' par l'algorithme de Huffman est optimal parmi les arbres ayant pour feuilles les feuilles constituant F' .

Or la première étape de l'algorithme de Huffman appliqué à la forêt F consiste à remplacer les feuilles f_1 et f_2 par un nœud ayant f_1 et f_2 pour fils.

Il en découle que A' est obtenu à partir de A en simplifiant A par coupe de f_1 et f_2 .

Comme A' est optimal, la question 19 prouve que A est bien optimal. \square

Question 24.

On peut facilement appliquer l'algorithme de Huffman à la main. On peut aussi demander au calculateur de faire le travail :

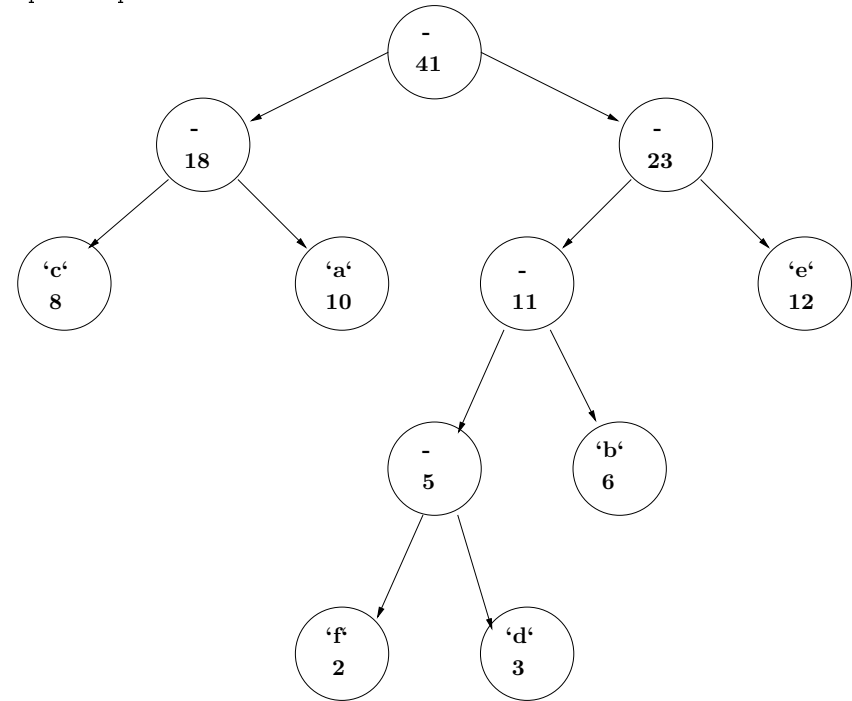
```
let Huffam F =
  while F.nb_arbres >= 2 do assemblage F done;
  F
;;
Huffam : Forêt -> Forêt = <fun>

let MAX = 13;;

let forêt =
  let table_forêt = make_vect MAX noeud_vide in
  table_forêt.(0) <- {lettre = 'a'; poids = 10; fg = -1; fd = -1};
  table_forêt.(1) <- {lettre = 'b'; poids = 6; fg = -1; fd = -1};
  table_forêt.(2) <- {lettre = 'c'; poids = 8; fg = -1; fd = -1};
  table_forêt.(3) <- {lettre = 'd'; poids = 3; fg = -1; fd = -1};
  table_forêt.(4) <- {lettre = 'e'; poids = 12; fg = -1; fd = -1};
  table_forêt.(5) <- {lettre = 'f'; poids = 2; fg = -1; fd = -1};
  {nb_arbres = 6; nb_noeuds = 6; table = table_forêt}
;;

Huffam forêt;;
- : Forêt =
{nb_arbres = 1; nb_noeuds = 11;
 table =
[[{lettre = '-'; poids = 41; fg = 1; fd = 10};
 {lettre = '-'; poids = 18; fg = 3; fd = 8};
 {lettre = '-'; poids = 11; fg = 4; fd = 7};
 {lettre = 'c'; poids = 8; fg = -1; fd = -1};
 {lettre = '-'; poids = 5; fg = 5; fd = 6};
 {lettre = 'f'; poids = 2; fg = -1; fd = -1};
 {lettre = 'd'; poids = 3; fg = -1; fd = -1};
 {lettre = 'b'; poids = 6; fg = -1; fd = -1};
 {lettre = 'a'; poids = 10; fg = -1; fd = -1};
 {lettre = 'e'; poids = 12; fg = -1; fd = -1};
 {lettre = '-'; poids = 23; fg = 2; fd = 9};
 {lettre = '\000'; poids = 0; fg = -1; fd = -1};
 {lettre = '\000'; poids = 0; fg = -1; fd = -1}]]]
```

Ce qui correspond à l'arbre suivant :



Lequel correspond au codage suivant de l'alphabet pondéré :

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

_____ FIN _____