JFP'25

27 mai 2025

Informations importantes

Codage des entiers

Comme il s'agit de (dé)sérialiser certaines données dans des fichiers, il est nécessaire de préciser comment certains types y sont stockés. Les *entiers* sont en petit-boutisme (little-endian). C'est-à-dire que les octets de poids faibles sont les premiers en mémoire. Autrement dit un nombre comme `0xDEADBEEF` est enregistré à partir de l'index i ainsi :

...	0xEF	0xBE	0xAD	0xDE	...
	i	$i+1$	$i+2$	$i+3$	

Format des images natives

Les images manipulées dans la suite sont purement monochromes, donc ne contenant que des pixels noirs ou blancs, même si le format natif supporte des couleurs ou des niveaux de gris.

Programmes

Dans chaque épreuve qui suit il s'agit d'écrire un programme qui doit pouvoir s'exécuter de l'une des deux façons suivantes :

```
prog image-source.ext image-destination.ext
prog image-source.ext image-destination.ext
```

`prog` étant un programme permettant de gérer un format particulier d'image (l'un des formats JFP) en s'appuyant sur le format JFPRAW. JFPRAW est le format pivot du concours. C'est-à-dire que pour n'importe quel format X, le programme correspondant de l'épreuve propose les deux transformations :

— RAW→X et,
— X→RAW.

Attention, le programme doit prendre soin de vérifier que l'image donnée comme source est bien au format attendu.

On notera qu'il est peut-être plus simple d'écrire d'abord les décodeurs que les codeurs. D'ailleurs pour avancer à l'étape N+1, il est nécessaire de posséder le décodeur de l'étape N.

1 Format RAW

Ce programme est chargé de transformer une image au format PNG en une image au format JFPRAW ou l'inverse.

Quelques images de tests vous sont données : ...

1.1 Le format JFPRAW

Un fichier JFPRAW est structuré ainsi :

octets	nom	valeur	commentaire
0 à 5	header	`JFPRAW`	identificateur de format
6 à 9	width	*entier*	largeur de l'image
10 à 13	height	*entier*	hauteur de l'image
14 à 17	data size	*entier*	taille des données de codage (en octets)
18 à 18+data size-1	data	*octets*	données de codage

La suite d'octets dénommée *data* représente la suite de bits 0/1 (noir/blanc) des pixels de l'image parcourue depuis le point en haut à gauche, de la gauche vers la droite, de haut en bas, jusqu'au point en bas à droite.

Ainsi pour une image de taille (3x4) comme :

0	0	1
1	0	1
0	1	1
0	0	1

La suite des bits de l'image est $B_{i\in[0,12)}$ est = 001101011001.

Cette suite est stockée dans une suite d'octets (unité de lecture/écriture en mémoire comme sur fichier), il faut la compléter afin d'obtenir une longueur multiple de 8. On complètera la suite par autant de bits 0 que nécessaire. Ainsi la suite stockée sera $S_{i\in[0,16)}$ est = 001101011001**0000** où **0000** sont les bits de complétion (padding).

La suite de bits est stockée dans une suite d'octets $O_{i\in[0,2)}$ où O_i a pour valeur le nombre $S_{8*i+7}...S_{8*i+1}S_{8*i}$ en base 2. Pour notre exemple O_0 = 10101100 soit en base 16 : 0xAC ou en base 10 : 172. Et O_1 = 00001001.

Ainsi pour l'image donnée en exemple le contenu du fichier (qui sera de longueur 20) devra être :

'J'	'F'	'P'	'R'	'A'	'W'	0x03	0x00	0x00	0x00	0x04	0x00	0x00	0x00
0	1	2	3	4	5	6	7	8	9	10	11	12	13

				0x02	0x00	0x00	0x00	0xAC	0x09
				14	14	16	17	18	19

2 Format RLE

Écrire un programme permettant de coder ou décoder le format JFPRAW en format JFPRLE.

2.1 Le format JFPRLE

RLE est l'acronyme de Run-Length Encoding. L'idée est de coder une suite de valeurs sous la forme de couples (nombre,valeur) de sorte qu'une suite comme *aaaaabbb* soit représentée comme $(5, a)(3, b)$.

Un fichier JFPRLE est structuré comme suit :

octets	nom	valeur	commentaire
0 à 5	header	`JFPRLE`	identificateur de format
6 à 9	width	*entier*	largeur de l'image
10 à 13	height	*entier*	hauteur de l'image
14 à 17	data size	*entier*	taille des données de codage (en octets)
18 à 18+data size-1	data	*octets*	données de codage

La suite d'octets dénommée *data*, $D_{i\in[0,data\ size)}$, représente le codage RLE d'une image. Avec :

- $D_i \leq 127$, pour coder une suite de longueur D_i de pixels noirs ;
- $D_i > 127$, pour coder une suite de longueur $D_i - 128$ de pixels blancs.

Les précautions suivantes vous sont imposées afin de garantir l'unicité du codage :

1. aucune suite de longueur 0 ne doit être codée ; ce qui exclut $D_i = 0$ ou $D_i = 128$.
2. lorsqu'une suite de pixels uniformes (de valeur x) est de longueur supérieure à 127, il faut la décomposer ainsi :

$$\text{code}((l, x)) = \text{sequence}_{i\in[0,\frac{l}{127})}((127, x)).(l\%127, x)$$

Ainsi (300,x) sera codé (127,x)(127,x)(46,x).

3 Format HUF

Écrire un programme permettant de coder ou décoder le format JFPRAW en format JFPHUF. Ce format propose d'essayer de compresser le codage RAW à l'aide d'un code de Huffman.

3.1 Le codage de Huffman

La description informelle qui suit devrait suffire à reconstruire l'algorithme.

On se donne un mot, par exemple `cababaaadda` que l'on cherche à coder de façon optimale (en un sens non précisé ici).

3.1.1 Calcul des nombres d'occurrence

On construit tout d'abord la suite ordonnée par leur poids des couples (l, p) pour toutes les lettres du mot où :

- l est une lettre et,
- p son poids calculé initialement comme le nombre d'occurence de cette lettre dans le mot à coder.

Pour notre exemple, on a donc : $(c, 1)$, $(b, 2)$, $(d, 2)$, $(a, 6)$.

Si on considère un couple (l, p) comme un arbre réduit à une feuille, on obtient alors une forêt ordonnée :

$$(c, 1) \qquad (b, 2) \qquad (d, 2) \qquad (a, 6)$$

Afin de garantir l'unicité l'ordre est :

- celui des poids croissants ;
- en cas de poids égaux, les symboles dans leur ordre naturel (ici b $<$ d).

3.1.2 Fusion d'arbres

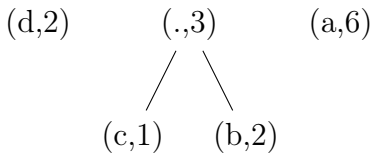
La fusion de deux arbres consiste en la création d'un arbre constitué d'une racine et de deux fils (gauche et droit) que sont les arbres initiaux. La lettre portée par la racine n'a pas d'importance, on la notera en utilisant le symbole . (point). Le nombre d'occurence de la racine est la somme des nombres d'occurrence portés par les deux arbres initiaux.

3.1.3 Itération de la fusion

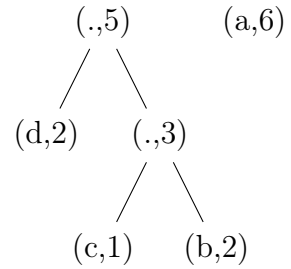
L'algorithme consiste en :

— l'itération de la fusion en un arbre des **deux premiers** arbres de la forêt selon l'ordre ;
— les deux arbres fusionnés disparaissent de la forêt ;
— l'arbre obtenu est inséré dans la forêt selon l'ordre des poids et le plus à «gauche» possible.

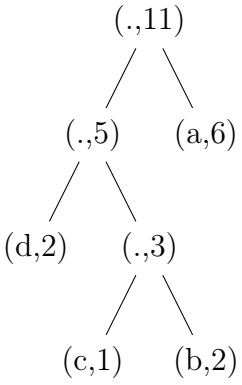
Lorsqu'il ne reste plus qu'un seul arbre, l'algorithme s'arrête et l'arbre obtenu est l'arbre dit de Huffman (du nom de son inventeur). La première fusion donne :



Il y a trois arbres, on fusionne et ordonne à nouveau :



Il reste deux arbres, on itère donc encore une fois :



On a trouvé un arbre de Huffman pour le mot `cababaaadda`. En général, il existe plusieurs arbres de Huffman, mais les consignes précédentes permettentd'assurer qu'un seul arbre convient ici.

3.1.4 Étiquettage

On étiquette les arcs de l'arbre de Huffman à l'aide des symboles 0 et 1, en distinguant les deux fils. La convention est 0 pour l'arc menant au fils gauche et 1 pour l'autre :

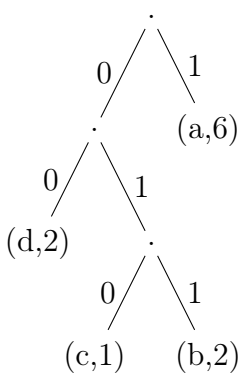


FIGURE 1 – Un arbre de Huffman

3.1.5 Codage

Le codage d'une lettre du mot initial est le mot formé par les étiquettes du chemin menant de la racine à la feuille correspondant à la lettre.

lettre :	c	a	b	a	b	a	a	a	d	d	a
codage :	010	1	011	1	011	1	1	1	00	00	1

Donc le codage du mot `cababaaadda` est la suite de bits : `0101011101111100001`.

3.1.6 Décodage

Le décodage s'effectue en lisant la suite de bits et en parcourant en conséquence l'arbre (en partant de la racine jusqu'à atteindre une feuille puis en repartant de la racine).

- Partant de la racine, on lit 0, on se déplace à gauche, on lit 1 on va à droite, on lit 0 on va à gauche, on tombe sur la feuille 'c'.
- On remonte à la racine en continuant à lire la suite. On lit 1 on va à droite et on tombe sur la feuille 'a'.
- On remonte à la racine en continuant à lire la suite. On lit 0, puis, 1, puis 1, on tombe sur la feuille 'b'.
- On continue ainsi jusqu'à épuisement de la suite de bits.

3.1.7 Cas dégénérés

Dans le cas particulier où le texte ne comprend qu'une seule lettre, l'arbre est réduit à un seul nœud. Dans ce cas, il n'y a pas besoin de parcourir l'arbre car on sait qu'il n'y a qu'une seule lettre ; celle contenue dans la racine. Il n'y a pas besoin non plus de coder quoi que ce soit ; on verra comment cela peut être réalisé dans la section suivante.

3.2 Le format JFPHUF

Important : pour le format JFPHUF l'alphabet à considérer dans notre codage d'image est celui des octets (les «lettres» à lire sont des paquets de 8 bits). On fait donc des comptage d'occurence sur les octets qui codent la séquence de bits du format JFPRAW (le champ data du format).

Ce format

index	nom	valeur	commentaire
0 à 5	header	`JFPHUF`	identificateur de format
6 -à 9	width	*entier*	largeur de l'image
10 à 13	height	*entier*	hauteur de l'image
14 à 17	size	*entier*	taille des données de codage (en octets)
18 à x	tree	*tree*	codage de l'arbre de Huffman
$x+1$ –	code	*h-code*	codage de Huffman de l'image RAW

3.2.1 Sérialisation de l'arbre

Le type *tree* est structuré ainsi :

index	nom	valeur	commentaire
18 à 21	n	*entier*	taille en octets du codage de l'arbre
22 à ...	nodes	*node*	le parcours préfixe de l'arbre

Le codage consiste à :

- coder un nœud par l'octet 0x00 ; suivi du codage de ses deux fils gauche et droit dans l'ordre ;
- coder une feuille par l'octet 0xFF suivi de l'écriture de l'octet qui représente le symbole porté par la feuille.

Ainsi le codage de l'arbre de la figure **??** sera codé par la suite de 11 octets suivants :

```
0x00 0x00 0xFF d 0x00 0xFF c 0xFF b 0xFF a
```

3.2.2 Codage de Huffman

Le type *h-code* est structuré ainsi :

index	nom	valeur	commentaire
$22+n$ à $25+n$	bcount	*entier*	longueur en bits du codage de Huffman
$26+n$ à	bcode	*octet*	octets stockant le code de Huffman

3.2.3 Exemple

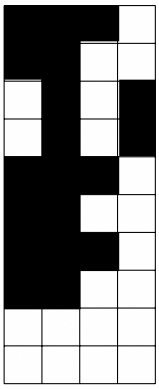


FIGURE 2 – `image9.png`

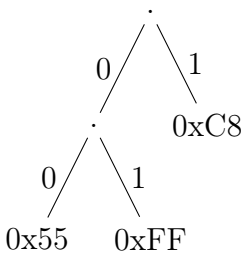
La suite de bits (RAW) correspondant à l'image (de taille 4x10) de la figure **??** est : `00010011.10101010.00010011.00010011.11111111`.

La suite des octets obtenus à partir de cette séquence est :

`0xC8 0x55 0xC8 0xC8 0xFF`

En effet : 0xC8 = 0b11001000, 0x55 = 0b01010101 et 0xFF = 0b11111111.

L'arbre de Huffman obtenu par comptage des occurrences des octets est :



Cet arbre doit alors être codé comme suit :

`0x00 0x00 0xFF 0x55 0xFF 0xFF 0xFF 0xC8`

Et la suite des bits constituants l'image brute est codée comme la séquence de bits : 1.00.1.1.01. Ce qui constitue l'octet 0x59.

3.2.4 Cas dégénéré

On a déjà fait remarquer que dans le cas où le texte n'est constitué que d'une seule lettre, l'arbre est dégénéré.

Codage dégénéré. Le codage de Huffman aura un nombre de bits (bcount) égal à 0 et le code sera donc de longueur nulle (bcode réduit à rien).

Décodage dégénéré. Le décodage consiste à reconstituer la séquence initiale d'octets à l'aide :

— de la lettre portée par l'arbre réduit à une feuille ;
— de la taille de l'image.

4 Format QUA

Écrire un programme permettant de coder ou décoder le format JFPRAW en format JFPQUA. Ce format propose d'essayer de compresser le codage RAW à l'aide d'un arbre quaternaire (quad-tree).

Attention à lire attentivement les détails !

4.1 Les arbres quaternaires

On considère la plupart du temps une image comme une matrice ; mais il existe de nombreuses autres possibilités. Parmi les autres, il y a les arbres quaternaires. Un arbre binaire est constitué de nœuds ayant toujours 2 nœuds fils, un arbre quaternaire est constitué de nœuds ayant toujours 4 fils. Un arbre binaire peut être utilisé pour coder une liste (structure linéaire), un arbre quaternaire code une structure bidimensionnelle [1].

4.1.1 Division récursive d'une image

Une image carré de taille $L = 2^l$ peut être découpée récursivement l fois en 4 quadrants (nord-ouest, nord-est, sud-ouest, sud-est) :

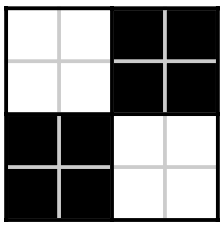
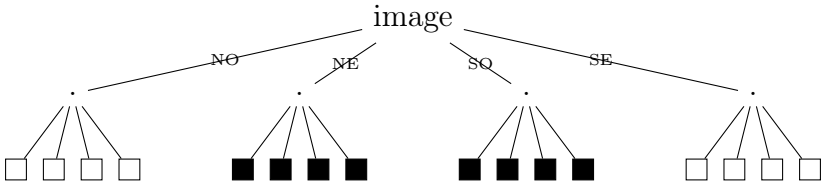


FIGURE 3 – Une image 4x4

On peut alors construire l'arbre quaternaire suivant qui la réprésente :



1. Les arbres octaux – oct-tree codent des structures tri-dimensionnelles

4.1.2 Propriété de l'arbre

Une propriété intéressante est que la taille du carré unitaire blanc ou noir ne compte pas. Cette représentation fonctionne si le carré blanc et le carré noir ont comme taille nxn, pour tout $n > 0$. Autrement dit, cette représentation est aussi valable pour représenter l'image suivante :

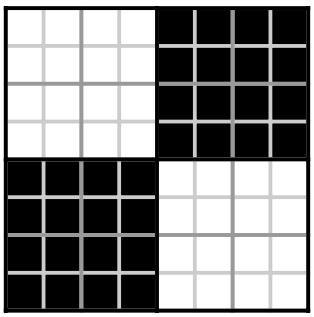


FIGURE 4 – Une image 8x8

Cette propriété peut alors être utilisée pour réduire l'arbre.

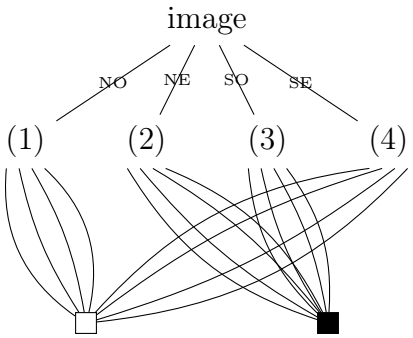
4.1.3 Réduction de l'arbre

L'arbre peut être réduit en :

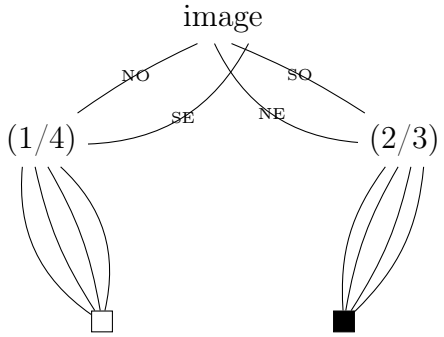
- identifiants les nœuds identiques (c'est-à-dire qui représentent une même sous-image) ;
- supprimant les nœuds qui ont 4 fils identiques et en prolongeant l'arc entrant vers «l'unique» fils.

Cette réduction construit un graphe acyclique dirigé à une racine.

Identification. Par exemple, dans l'arbre précédent, on a de nombreux nœuds qui désigne le nœud blanc (idem pour le noir). On fusionne donc tous ces nœuds en un seul :



On peut itérer l'identification car les nœuds (1) et (4) désignent la même sous-image, idem pour (2) et (3) :

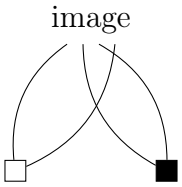


Suppression. On constate que :

— le nœud (1/4) a 4 fils qui désignent 4 sous-images identiques (un carré blanc) ;.

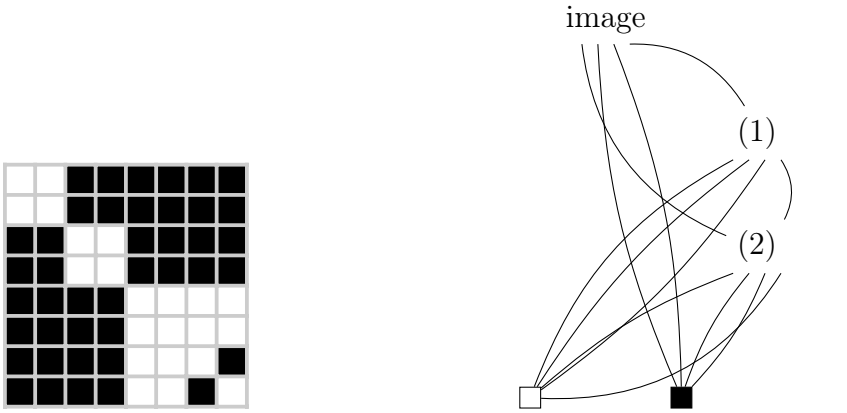
— le nœud (2/3) a 4 fils qui désignent 4 sous-images identiques (un carré noir).

On supprime ces nœuds en faisant pointer leur arc entrant vers l'«unique» fils :



Itération. On itère ces opérations tant que possible.

Exemple.



(a) Une image 8x8 (b) Son graphe réduit associé

FIGURE 5 – Une image et son quadtree

On remarque que :

— le nœud (2) représente l'image : NO-Blanc, NE-Noir, SO-Noir, SE-Blanc ;
— le nœud (1) a pour son quadrant SE l'image codée par (2) ;
— le nœud racine a pour son quadrant NO l'image codée par (2).

Attention, pour les deux derniers cas, la taille de l'image codée par (2) n'est pas la même ! Dans le premier cas, l'image est obtenue en descendant deux fois dans l'arbre (racine $\rightarrow$ (1) $\rightarrow$ (2)) alors que pour le second cas on est descendu une seule fois fois (racine $\rightarrow$ (2)).

4.1.4 Codage

Le codage consiste à construire le graphe réduit et à le sérialiser. Le format est précisé en section suivante.

4.1.5 Décodage

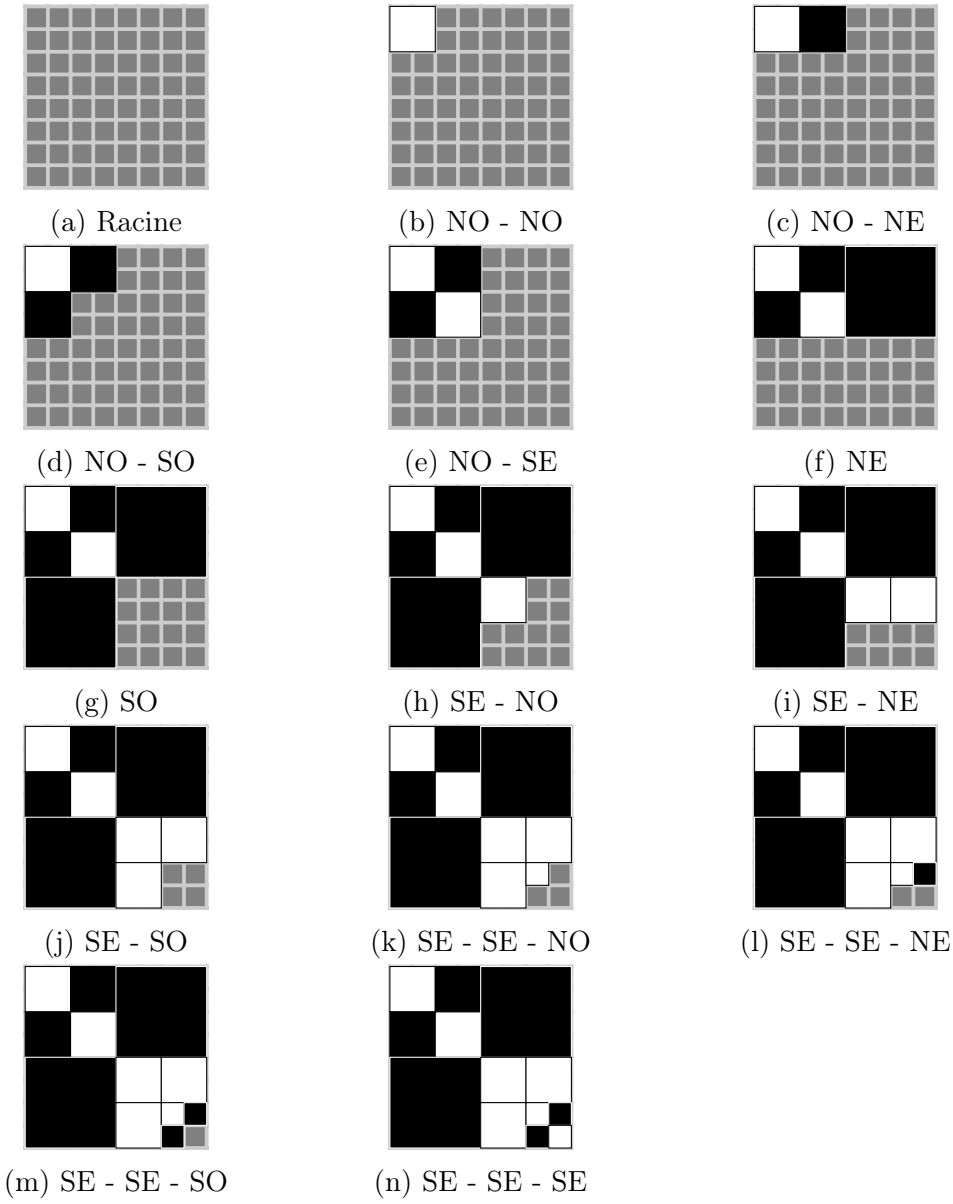
Le décodage consiste à désérialiser le graphe puis à reconstituer l'image. La reconstitution de l'image procède récursivement.

À partir du graphe de l'exemple précédent, on construit une image de la taille indiquée (ici 8x8).

Puis on appelle une procédure qui reçoit un nœud n et une image i et qui :

— si n est le nœud blanc ou noir remplit l'image en blanc ou noir ;
— rappelle la procédure sur le nœud NO et la sous-image constituée du quadrant NO. Le quadrant est de taille la moitié de l'image originale (moitié de la taille $\Rightarrow$ quart de la surface) ;
— idem pour les partis NE, SO et SE.

On obtient donc la séquence suivante de reconstitution (certaines étapes intermédiaires sont omises) :



(a) Racine (b) NO - NO (c) NO - NE

(d) NO - SO (e) NO - SE (f) NE

(g) SO (h) SE - NO (i) SE - NE

(j) SE - SO (k) SE - SE - NO (l) SE - SE - NE

(m) SE - SE - SO (n) SE - SE - SE

FIGURE 6 – Décodage d'une image

4.1.6 Cas des images de taille quelconque

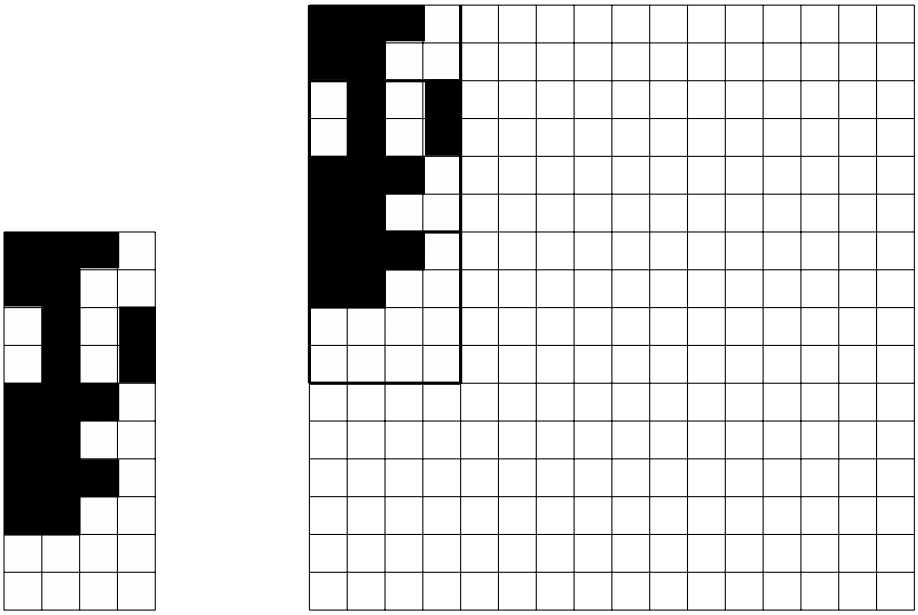
Dans la description précédente, on a considéré des images carrés dont la taille est une puissance de 2 (pour faciliter le raisonnement). Hélas, en général les images ne sont ni carré ni d'une taille puissance de 2.

Pour régler le problème il suffit d'étendre l'image initiale de sorte qu'elle soit

inscrite dans un carré ayant la bonne propriété. On considérera l'extension suivante :

— l'image initiale sera calée en haut à gauche d'une image carré dont la taille est une puissance de 2 ;
— la taille du carré sera la plus petite puissance de 2 possible ;
— l'image étendue le sera avec des pixels uniformément blancs.

Ainsi :



(a) Image 4x10 (b) Image 4x10 étendue en 16x16

FIGURE 7 – Extension d'une image

4.2 Le format JFPQUA

Un fichier de format JFPQUA est structuré ainsi :

index	nom	valeur	commentaire
0 à 5	header	`JFPQUA`	identificateur de format
6 à 9	width	*entier*	largeur de l'image
10 à 13	height	*entier*	hauteur de l'image
14 à 17	size	*entier*	taille des données de codage (en octets)
–	tree	*graphe*	codage du graphe réduit

Le type *graphe* est codé comme suit :

index	nom	valeur	commentaire
i à i+3	n	*entier*	nombre de nœuds du graphe
i+4 à i+7	root	*entier*	index du nœud racine
i+8 à i+23	node_0	*node*	nœud blanc
i+24 à i+39	node_1	*node*	nœud noir
⋮			
–	node_{n-1}	*node*	nœud

Attention, le codage du graphe contient toujours deux nœuds, le blanc et le noir ($n \geq 2$).

Le type *node* est codé comme suit :

index	nom	valeur	commentaire
i à i+3	no	*entier*	numéro du nœud du quadrant NO
i+4 à i+7	ne	*entier*	numéro du nœud du quadrant NE
i+8 à i+11	so	*entier*	numéro du nœud du quadrant SO
i+12 à i+15	se	*entier*	numéro du nœud du quadrant SE

Par convention, le nœud blanc a pour fils lui-même ; idem pour le nœud noir.

Le graphe de la figure **??** est donc ainsi codé comme suit :

index	index NO	index NE	index SO	index SE
0	0	0	0	0
1	1	1	1	1
2	3	1	1	4
3	0	1	1	0
4	0	0	0	3

Son nœud racine étant le 2.