Principes de fonctionnement des machines binaires

2019/2020

Pierluigi Crescenzi
Université de Paris, IRIF

Université de Paris

INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

cnrs dépasser les frontières

- Tests et examens
  - CC : résultat des tests en TD / TP (**cette semaine** et semaine 10)
  - E0 : partiel (samedi 26 octobre)
  - E1 : examen mi décembre
  - E2 : examen fin juin
- Notes finales
  - Note session 1 : 25% CC + 25% E0 + 50% E1
  - Note session 2 : max(E2, 33% CC + 67% E2)
- Rappel
  - Pas de note $\Rightarrow$ pas de moyenne $\Rightarrow$ pas de semestre
- Site web
  - `moodlesupd.script.univ-paris-diderot.fr`

- Numération et arithmétique
- Numération et arithmétique en machine
- **Numérisation et codage (texte, images)**
- **Compression**, cryptographie, contrôle d'erreur
- Logique et calcul propositionnel
- Circuits numériques

- La transmission électronique nécessite parfois l'encodage de fichiers contenant du «binaire» en suite de caractères alphabétiques
  - uuencode, Base64, quoted-printable

- La transmission électronique nécessite parfois l'encodage de fichiers contenant du «binaire» en suite de caractères alphabétiques
  - uuencode, Base64, quoted-printable
- uuencode
  - L'idée est de transformer chaque suite de 24 bits en suite de 4 caractères
    - Chaque sous-mot de 6 bits ($2^6 = 64$) est codé en le caractère de code ASCII $x + 32$

USASCII code chart

Bits $b_7 b_6 b_5$	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 Row ↓ Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

- La transmission électronique nécessite parfois l'encodage de fichiers contenant du «binaire» en suite de caractères alphabétiques
  - uuencode, Base64, quoted-printable
- uuencode
  - L'idée est de transformer chaque suite de 24 bits en suite de 4 caractères
    - Chaque sous-mot de 6 bits ($2^6 = 64$) est codé en le caractère de code ASCII $x + 32$
- Exemple
  - 000011110000111100001111

USASCII code chart

$b_7 b_6 b_5$ → / Bits / $b_4 b_3 b_2 b_1$ ↓ Row / Column →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

- La transmission électronique nécessite parfois l'encodage de fichiers contenant du «binaire» en suite de caractères alphabétiques
  - uuencode, Base64, quoted-printable
- uuencode
  - L'idée est de transformer chaque suite de 24 bits en suite de 4 caractères
    - Chaque sous-mot de 6 bits ($2^6 = 64$) est codé en le caractère de code ASCII $x + 32$
- Exemple
  - 000011110000111100001111
    - $000011 \Rightarrow 32 + 3 \Rightarrow$#

USASCII code chart

$b_7 b_6 b_5$ →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits ↓	Column →	0	1	2	3	4	5	6	7
b_4 b_3 b_2 b_1	Row ↓								
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

- La transmission électronique nécessite parfois l'encodage de fichiers contenant du «binaire» en suite de caractères alphabétiques
  - uuencode, Base64, quoted-printable
- uuencode
  - L'idée est de transformer chaque suite de 24 bits en suite de 4 caractères
    - Chaque sous-mot de 6 bits ($2^6 = 64$) est codé en le caractère de code ASCII $x + 32$
- Exemple
  - 000011110000111100001111
    - $000011 \Rightarrow 32 + 3 \Rightarrow$#
    - $110000 \Rightarrow 32 + 48 \Rightarrow$P

USASCII code chart

$b_7 b_6 b_5$ → / Bits ↓	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 / Row ↓ Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

- La transmission électronique nécessite parfois l'encodage de fichiers contenant du «binaire» en suite de caractères alphabétiques
  - uuencode, Base64, quoted-printable
- uuencode
  - L'idée est de transformer chaque suite de 24 bits en suite de 4 caractères
    - Chaque sous-mot de 6 bits ($2^6 = 64$) est codé en le caractère de code ASCII $x + 32$
- Exemple
  - 000011110000111100001111
    - $000011 \Rightarrow 32 + 3 \Rightarrow$#
    - $110000 \Rightarrow 32 + 48 \Rightarrow$P
    - $111100 \Rightarrow 32 + 60 \Rightarrow$\

USASCII code chart

Bits $b_7 b_6 b_5$	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 Row ↓ Column →	0	1	2	3	4	5	6	7
0 0 0 0 0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0 2	STX	DC2	"	2	B	R	b	r
0 0 1 1 3	ETX	DC3	#	3	C	S	c	s
0 1 0 0 4	EOT	DC4	$	4	D	T	d	t
0 1 0 1 5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0 6	ACK	SYN	&	6	F	V	f	v
0 1 1 1 7	BEL	ETB	'	7	G	W	g	w
1 0 0 0 8	BS	CAN	(	8	H	X	h	x
1 0 0 1 9	HT	EM	)	9	I	Y	i	y
1 0 1 0 10	LF	SUB	*	:	J	Z	j	z
1 0 1 1 11	VT	ESC	+	;	K	[	k	{
1 1 0 0 12	FF	FS	,	<	L	\	l	|
1 1 0 1 13	CR	GS	-	=	M	]	m	}
1 1 1 0 14	SO	RS	.	>	N	^	n	~
1 1 1 1 15	SI	US	/	?	O	_	o	DEL

- La transmission électronique nécessite parfois l'encodage de fichiers contenant du «binaire» en suite de caractères alphabétiques
  - uuencode, Base64, quoted-printable
- uuencode
  - L'idée est de transformer chaque suite de 24 bits en suite de 4 caractères
    - Chaque sous-mot de 6 bits ($2^6 = 64$) est codé en le caractère de code ASCII $x + 32$
- Exemple
  - 000011110000111100001111
    - $000011 \Rightarrow 32 + 3 \Rightarrow$#
    - $110000 \Rightarrow 32 + 48 \Rightarrow$P
    - $111100 \Rightarrow 32 + 60 \Rightarrow$\
    - $001111 \Rightarrow 32 + 15 \Rightarrow$/

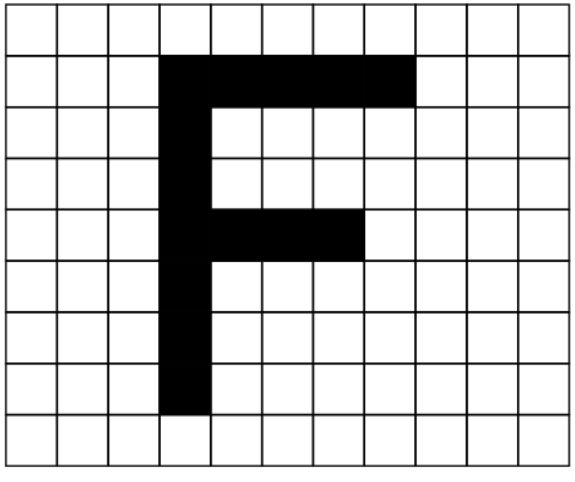
USASCII code chart

Bits $b_7 b_6 b_5$ →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 b_3 b_2 b_1 ↓ Row / Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

- Deux types d'images numériques
  - Matricielles
    - Description sous la forme d'une matrice dans laquelle chaque élément correspond à un point de l'image
      - La description est discrète (résolution)
  - Vectorielles
    - Description « idéale » sous la forme d'opérations géométriques appliquées à des formes décrites à l'aide d'équations mathématiques
      - Exemple : certaines polices de caractères

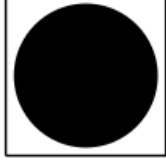
- Images matricielles (bitmap/raster)
  - On décompose l'image en points
    - Un point est un « atome » de la représentation
  - La valeur d'un point est la composition des valeurs des caractéristiques intéressantes (en général la couleur)
    - Les valeurs aussi sont discrètes
  - Un tel point est appelé **pixel** : **pic**ture **el**ement

- Images matricielles (bitmap/raster)
  - On décompose l'image en points
    - Un point est un « atome » de la représentation
  - La valeur d'un point est la composition des valeurs des caractéristiques intéressantes (en général la couleur)
    - Les valeurs aussi sont discrètes
  - Un tel point est appelé **pixel** : **pic**ture **el**ement
  - Exemple
    - Image de taille 9×11
      - 99 pixels
    - Noir et blanc
      - Valeur d'un point 0/1



- Codage des couleurs
  - Il existe de nombreuses façons de coder les couleurs numériquement
    - Le **noir et blanc**
    - Les **niveaux de gris**
      - Palette permettant de représenter différents gris, du noir au blanc (16 gris ou 256 gris sont courant)
    - En **RGB/RVB**
      - Système additif qui compose une couleur par addition de couleurs primitives (rouge, vert, bleu)
    - En **HSB/TSV**
      - Système de coordonnées dans un espace contenant des couleurs
  - Il existe des modes avec une valeur de transparence
    - Canal alpha

- Comment coder une couleur RGB ?
  - Un nombre pour le R, un pour le V, un pour le B
    - Mode arithmétique [0.0,1.0] (avec des flottants)
    - Mode pourcentage 0%-100% (des entiers suffisent)
    - Mode numérique, un nombre sur n-bits
  - Le nombre est ensuite codé en vue de son stockage

B	0	0	0	255	0	255	255	255
G	0	0	255	0	255	0	255	255
R	0	255	0	0	255	255	0	255
								

- Caractéristiques d'une image

- Caractéristiques d'une image
  - **Definition** : les dimensions de la matrice qui représente l'image
    - Combien de lignes et colonnes
      - C'est exprimé en pixels comme un produit de deux nombres (1024×768 px)

- Caractéristiques d'une image
  - **Definition** : les dimensions de la matrice qui représente l'image
    - Combien de lignes et colonnes
      - C'est exprimé en pixels comme un produit de deux nombres (1024×768 px)
  - **Resolution** : un pixel doit-il être représenté par une grande surface ou une petite surface ?
    - Elle indique la finesse de représentation de l'image
      - C'est en général exprimé sous la forme d'une densité linéaire
        - Nombre de pixel par pouce / point par pouce / dot-per-inch / pixel-per-inch
    - Il peut y avoir deux densités, une horizontale et une verticale

- Caractéristiques d'une image
  - **Definition** : les dimensions de la matrice qui représente l'image
    - Combien de lignes et colonnes
      - C'est exprimé en pixels comme un produit de deux nombres (1024×768 px)
  - **Resolution** : un pixel doit-il être représenté par une grande surface ou une petite surface ?
    - Elle indique la finesse de représentation de l'image
      - C'est en général exprimé sous la forme d'une densité linéaire
        - Nombre de pixel par pouce / point par pouce / dot-per-inch / pixel-per-inch
    - Il peut y avoir deux densités, une horizontale et une verticale
  - **Poids** : le nombre de bits utilisés pour coder une image
    - Sans compression c'est le produit de la définition par le nombre de bits utilisés pour coder un pixel

- PGM : un format de fichier codant une image en gris
  - Particulièrement inefficace en terme de poids mais très simple
  - L'image est décrite par la suite de nombre représentant les caractéristiques suivantes
    - Les caractères magiques P2
    - La définition (largeur puis hauteur)
    - La valeur maximale du niveau de gris
    - Les niveaux de gris, de gauche à droite puis de haut en bas

- PGM : un format de fichier codant une image en gris
  - Particulièrement inefficace en terme de poids mais très simple
  - L'image est décrite par la suite de nombre représentant les caractéristiques suivantes
    - Les caractères magiques P2
    - La définition (largeur puis hauteur)
    - La valeur maximale du niveau de gris
    - Les niveaux de gris, de gauche à droite puis de haut en bas

```
P2
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

- PGM : un format de fichier codant une image en gris
  - Particulièrement inefficace en terme de poids mais très simple
  - L'image est décrite par la suite de nombre représentant les caractéristiques suivantes
    - Les caractères magiques P2
    - La définition (largeur puis hauteur)
    - La valeur maximale du niveau de gris
    - Les niveaux de gris, de gauche à droite puis de haut en bas

```
P2
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11
0  3  0  0  0  0  0  7  7  7  7  0  0 11
0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

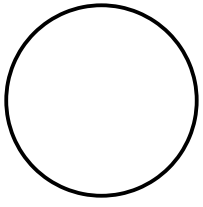
- Les images vectorielles : SVG
  - Un langage de description d'image indépendant de la résolution

- Les images vectorielles : SVG
  - Un langage de description d'image indépendant de la résolution

```svg
<svg width="200"
 height="200"
 xmlns="http://www.w3.org/2000/svg">
 <desc>Exemple SVG</desc>
 <circle cx="100"
 cy="100"
 r="50"
 fill="none"
 stroke="black"
 stroke-width="2" />
</svg>
```

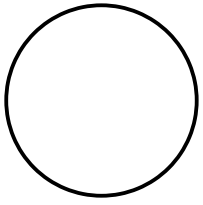
- Les images vectorielles : SVG
  - Un langage de description d'image indépendant de la résolution

```
<svg width="200"
 height="200"
 xmlns="http://www.w3.org/2000/svg">
 <desc>Exemple SVG</desc>
 <circle cx="100"
 cy="100"
 r="50"
 fill="none"
 stroke="black"
 stroke-width="2" />
</svg>
```



- Les images vectorielles : SVG
  - Un langage de description d'image indépendant de la résolution

```
<svg width="200"
 height="200"
 xmlns="http://www.w3.org/2000/svg">
 <desc>Exemple SVG</desc>
 <circle cx="100"
 cy="100"
 r="50"
 fill="none"
 stroke="black"
 stroke-width="2" />
</svg>
```

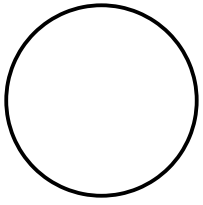


  - Un langage puissant

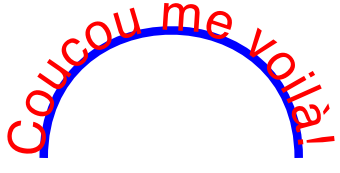
```
<svg width="2000" height="2000"
 xmlns="http://www.w3.org/2000/svg"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 version="1.1">
 <desc>Exemple SVG</desc>
 <path id="monChemin"
 d="M 150 150 C 150 50 300 50 300 150"
 fill="none" stroke="blue"
 stroke-width="5" />
 <text font-family="Helvetica"
 font-size="30" fill="red">
 <textPath xlink:href="#monChemin">Coucou me voil&#x00E0;!
 </textPath>
 </text>
</svg>
```

- Les images vectorielles : SVG
  - Un langage de description d'image indépendant de la résolution

```
<svg width="200"
 height="200"
 xmlns="http://www.w3.org/2000/svg">
 <desc>Exemple SVG</desc>
 <circle cx="100"
 cy="100"
 r="50"
 fill="none"
 stroke="black"
 stroke-width="2" />
</svg>
```



  - Un langage puissant




```
<svg width="2000" height="2000"
 xmlns="http://www.w3.org/2000/svg"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 version="1.1">
 <desc>Exemple SVG</desc>
 <path id="monChemin"
 d="M 150 150 C 150 50 300 50 300 150"
 fill="none" stroke="blue"
 stroke-width="5" />
 <text font-family="Helvetica"
 font-size="30" fill="red">
 <textPath xlink:href="#monChemin">Coucou me voil&#x00E0;!
 </textPath>
 </text>
</svg>
```

Compression

- Code compresseur
  - Il s'agit ici d'obtenir une représentation plus compacte
    - En vue de **transmission** (économie de bande passante)
    - En vue de stockage (économie d'espace)
  - Deux types de compression
    - **Conservative** ou **sans perte** : le message originel peut être reconstruit à l'identique en inversant la fonction
      - Ce type de compression est recherché avec du texte par exemple
    - **Non conservative** ou **avec perte** : le message originel n'est pas reconstruit à l'identique, mais un message similaire est obtenu à l'inversion
      - Souvent le cas des images et des sons : il n'est pas nécessaire que des détails quasi-invisibles-sensibles soient conservés
      - Cette compression permet d'obtenir de très bons taux de compression

- **Quotient de compression**
  - $Q = \text{volume initial}/\text{volume final}$
    - Plus une compression sera forte, plus le quotient de compression sera lui aussi élevé
- **Taux de compression** : deux façons habituelles de le définir
  - $T = \text{volume final}/\text{volume initial}$
    - Plus le taux de compression est faible, plus la taille du fichier compressé résultant est faible
      - $T = 1/Q$
  - $T = 1 - \text{volume final}/\text{volume initial}$
    - Plus le taux de compression est élevé, plus la taille du fichier compressé résultant est faible
    - $T = 1 - 1/Q$

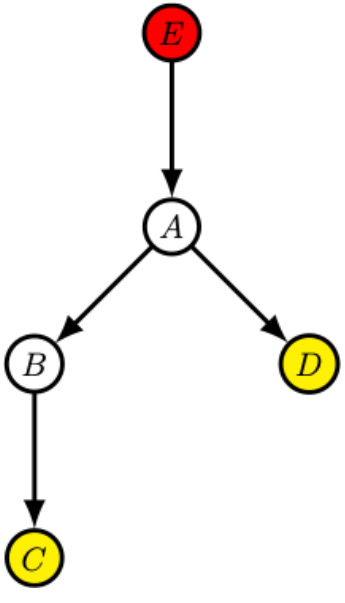
- Idée
  - Coder avec des mots courts les lettres les plus fréquentes
  - Coder avec des mots longs les lettres les moins fréquentes
  - Exemple
    - Trois lettres A, B, C : A très fréquente, B moyennement et C rare
    - On peut utiliser $\tau(A) = \texttt{0}$, $\tau(B) = \texttt{10}$ et $\tau(C) = \texttt{11}$
      - Ainsi $\tau(AAABAAABBAAAAC) = \texttt{00010000101000011}$: 18 bits
      - Un codage ordinaire sur 2 bits / caractère aurait donné 28 bits

- Fréquence des caractères dans Wikipédia en français

Lettre	Frequence	Lettre	Frequence
e	12,10	p	2,49
a	7,11	g	1,23
i	6,59	b	1,14
s	6,51	v	1,11
n	6,39	h	1,11
r	6,07	f	1,11
t	5,92	q	0,65
o	5,02	y	0,46
l	4,96	x	0,38
u	4,49	j	0,34
d	3,67	k	0,29
c	3,18	w	0,17
m	2,62	z	0,15

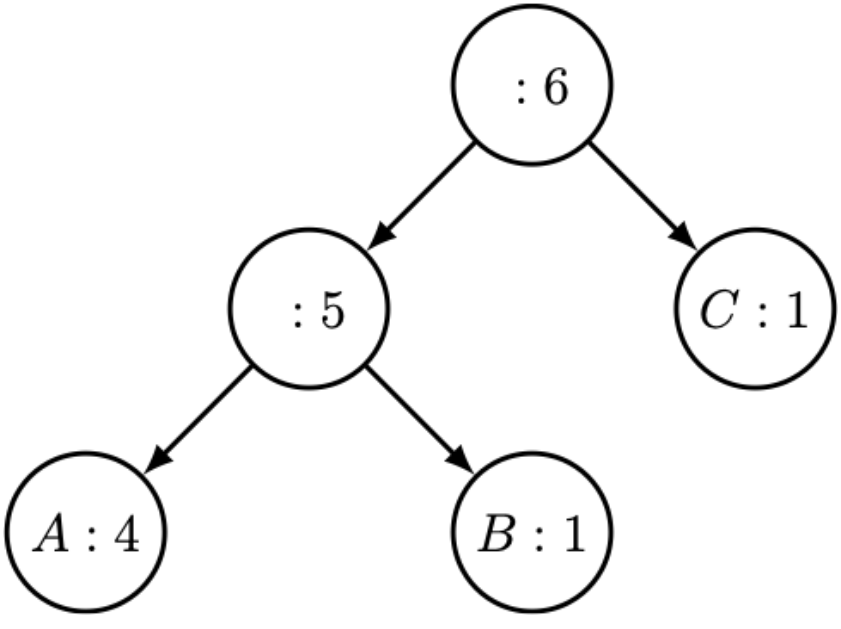
- Comment obtenir un codage à partir des fréquences des lettres ?
  - On fabrique un arbre
    - C'est une structure dans laquelle on trouve des nœuds
      - Un **nœud** permet de désigner d'autres nœuds
      - Un nœud qui ne désigne rien est une **feuille**
      - Un nœud sans ascendant est appelé **racine**

- Exemple
  - A, B, C, D, E : nœuds
  - E désigne A, A désigne B et D, B désigne C
  - E : racine
  - C, D : feuilles



- On fabrique un arbre **binaire** (avec uniquement des nœuds à deux descendants)
  - Chaque feuille représente une lettre et sa fréquence associée (ou son nombre d'occurrences)
  - Chaque nœud représente la somme des fréquences des nœuds qu'il désigne

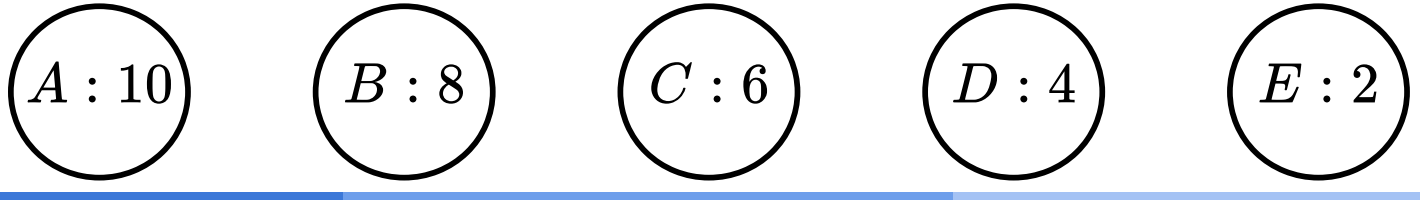
- Exemple
  - A : fréquence 4
  - B : fréquence 1
  - C : fréquence 1



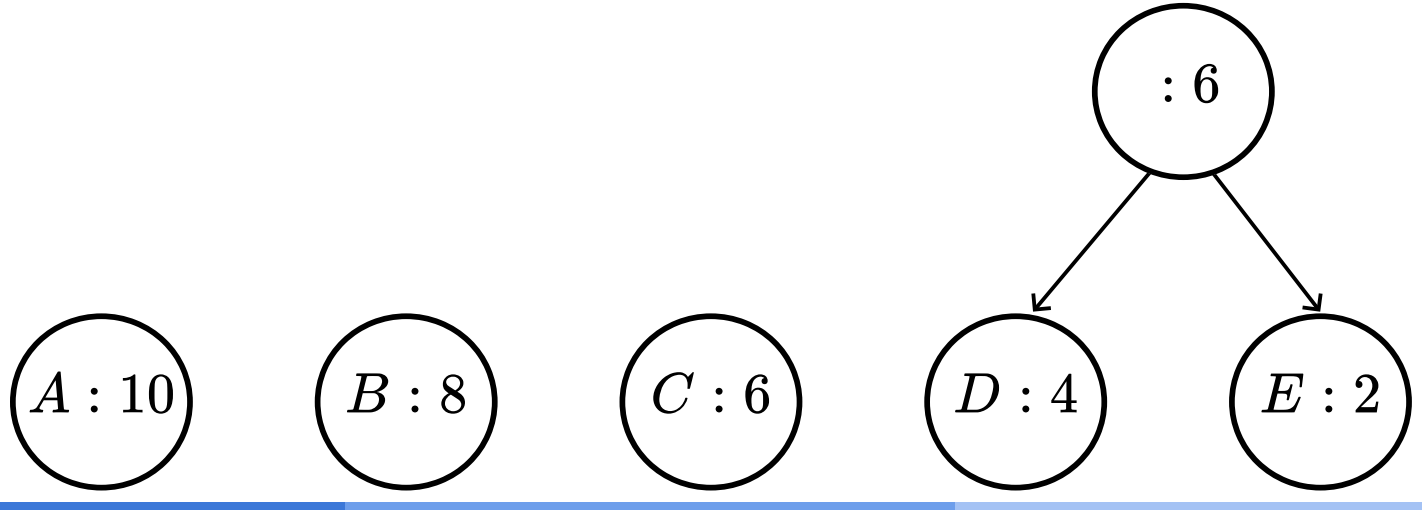
- Algorithme
  - On part de l'ensemble des arbres réduits aux simples lettres pondérées par leur fréquence d'apparition dans le texte
  - À chaque étape, on sélectionne deux arbres dont les fréquences des racines sont les plus petites et on fabrique un arbre dont le nœud racine désigne les deux arbres sélectionnés et dont la fréquence est simplement la somme des fréquences/occurrences

- Exemple : fréquences {A:10,B:8,C:6,D:4,E:2}

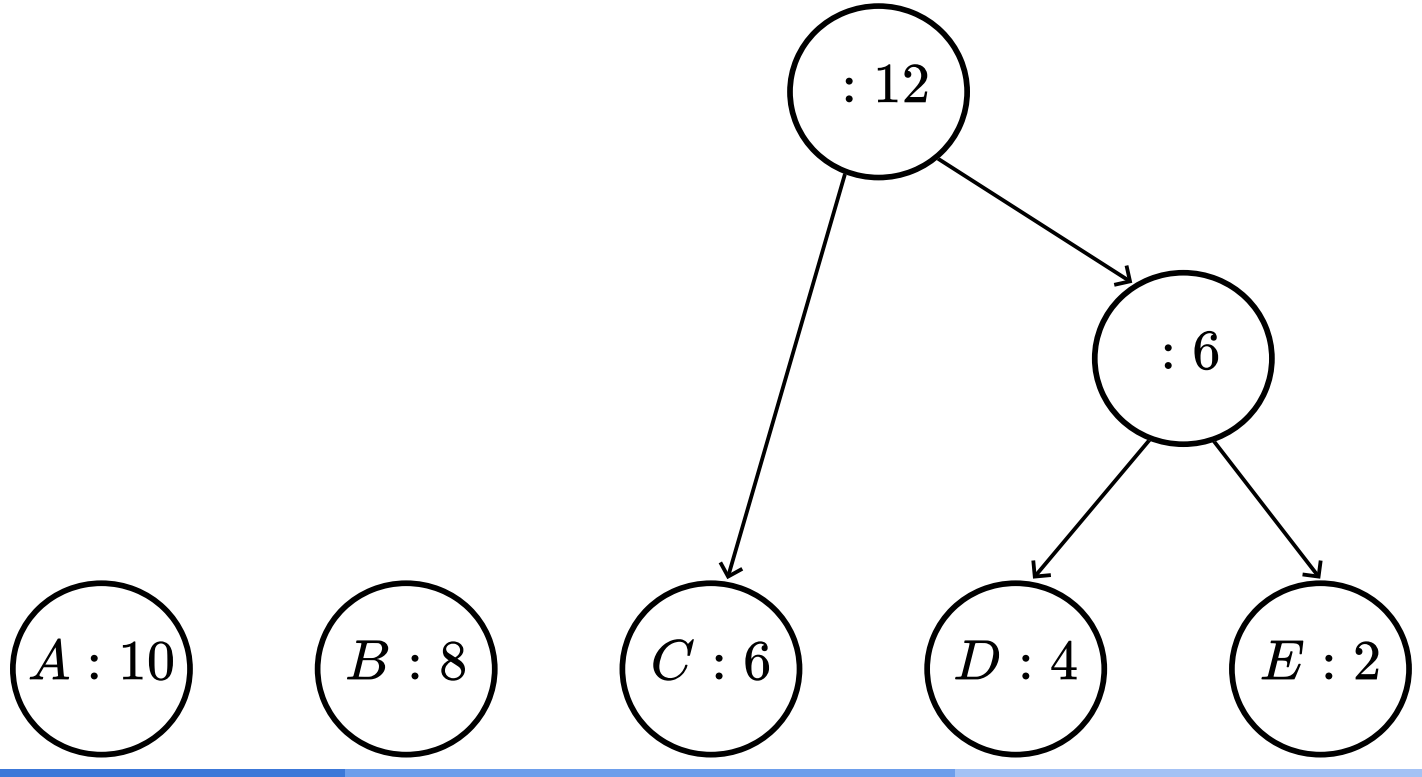
- Exemple : fréquences {A:10,B:8,C:6,D:4,E:2}



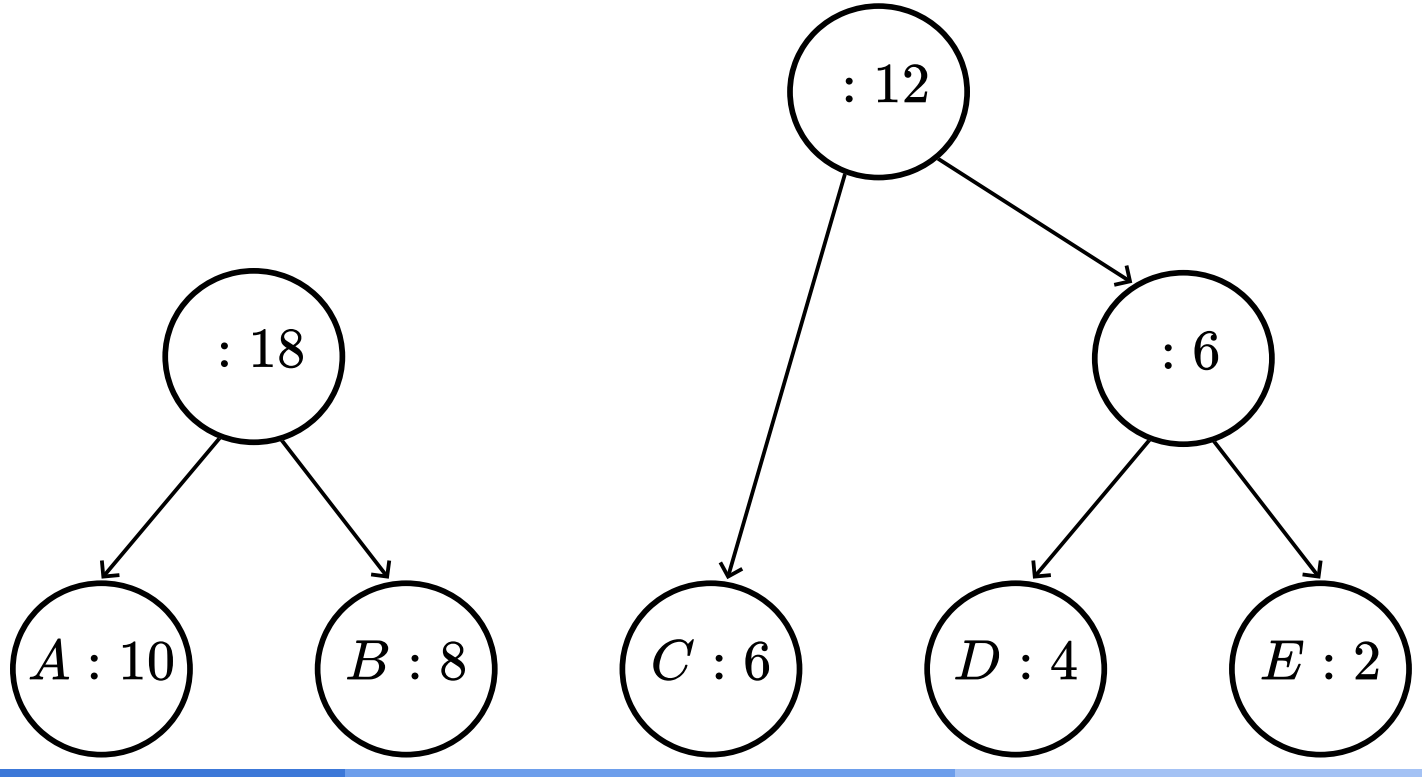
- Exemple : fréquences {A:10,B:8,C:6,D:4,E:2}



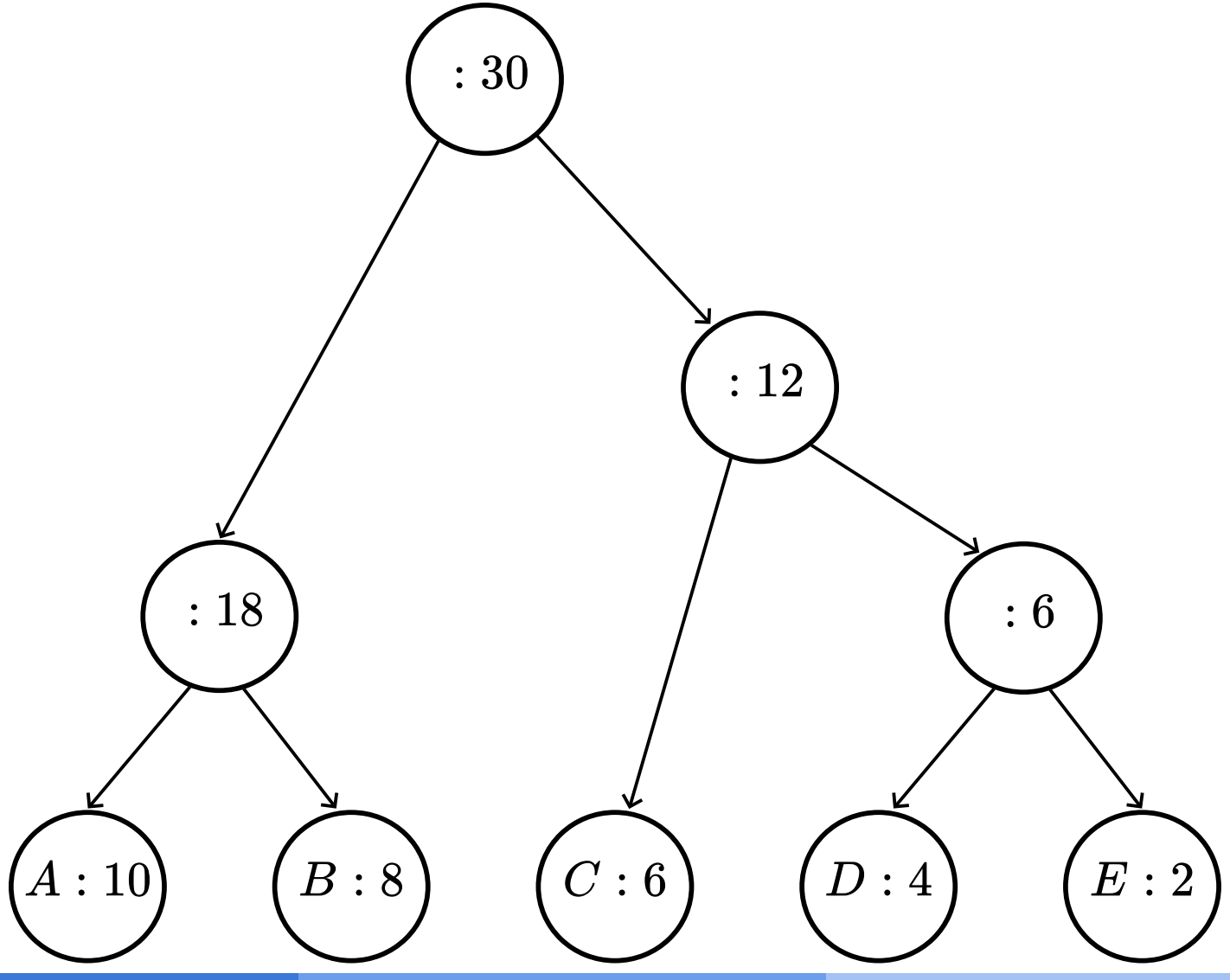
- Exemple : fréquences {A:10,B:8,C:6,D:4,E:2}



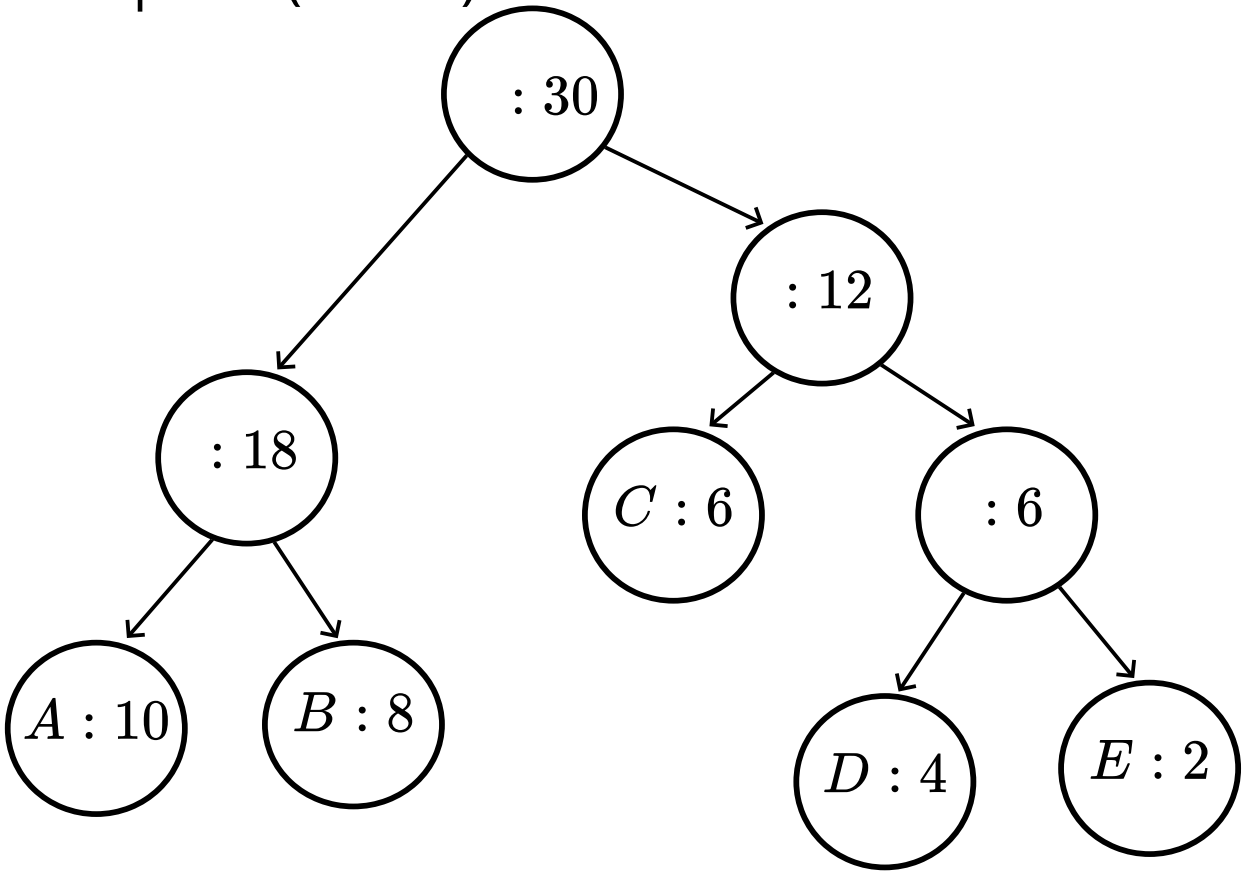
- Exemple : fréquences {A:10,B:8,C:6,D:4,E:2}



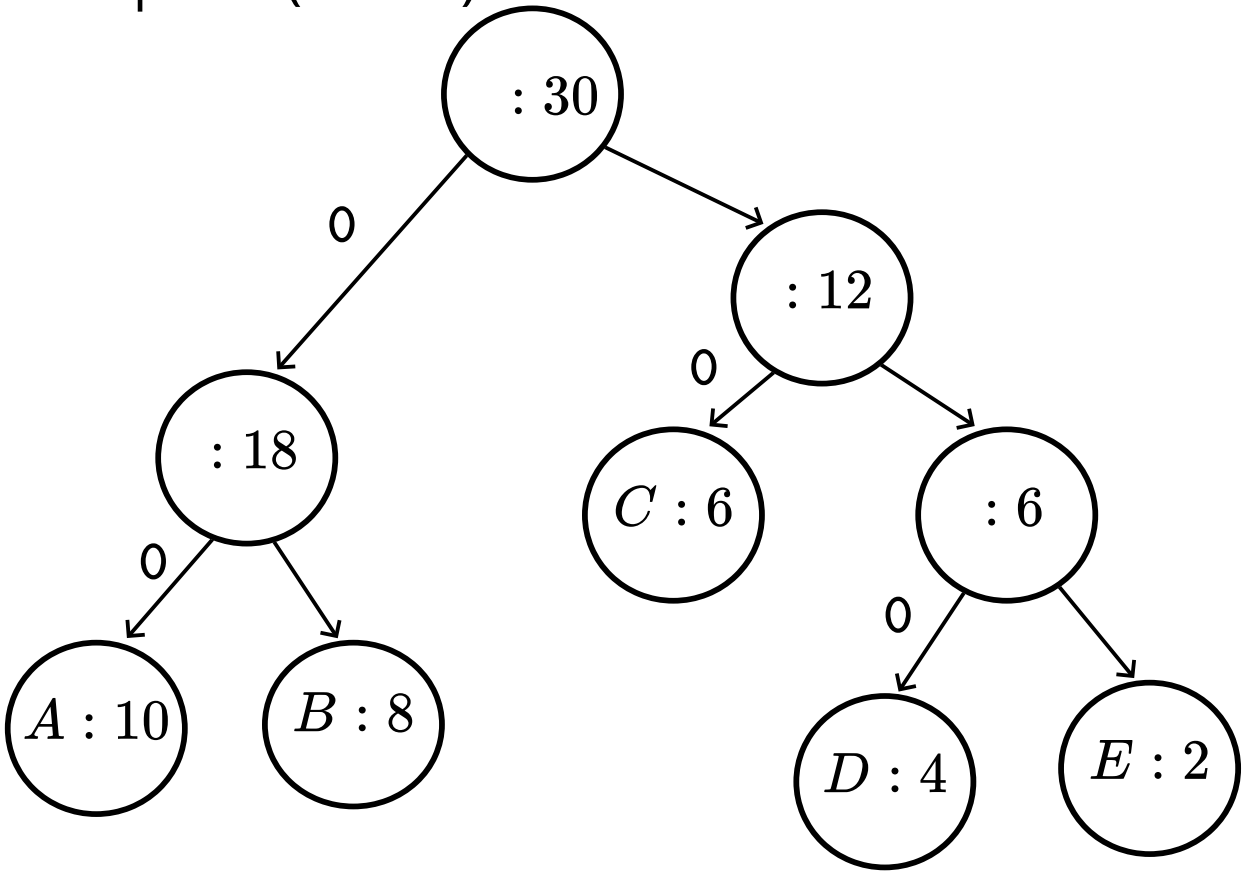
- Exemple : fréquences {A:10,B:8,C:6,D:4,E:2}



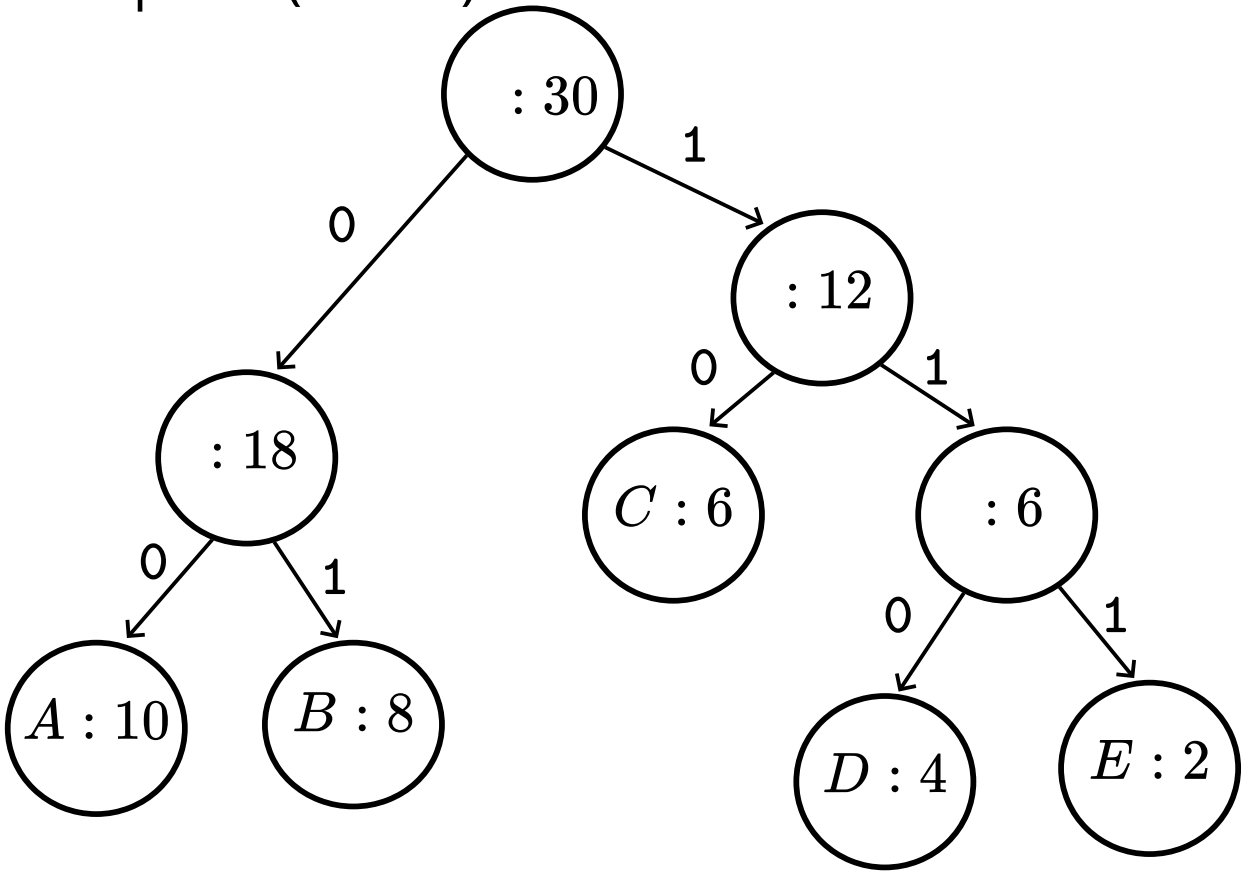
- On va utiliser l'**arbre de Huffman** pour coder les lettres
  - On étiquette chaque paire de branche descendante l'une par 0 (gauche) et l'autre par 1 (droite)



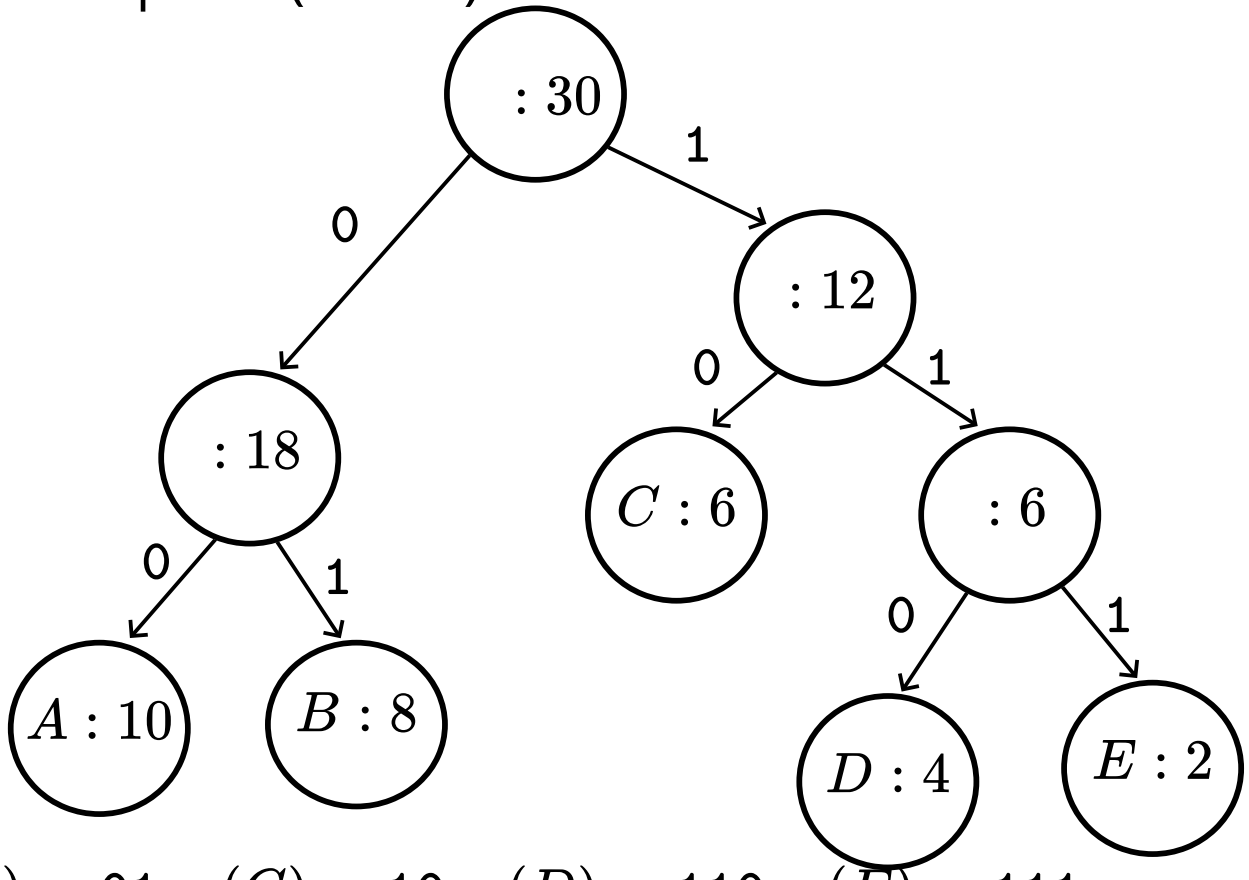
- On va utiliser l'**arbre de Huffman** pour coder les lettres
  - On étiquette chaque paire de branche descendante l'une par 0 (gauche) et l'autre par 1 (droite)



- On va utiliser l'**arbre de Huffman** pour coder les lettres
  - On étiquette chaque paire de branche descendante l'une par 0 (gauche) et l'autre par 1 (droite)



- On va utiliser l'**arbre de Huffman** pour coder les lettres
  - On étiquette chaque paire de branche descendante l'une par 0 (gauche) et l'autre par 1 (droite)

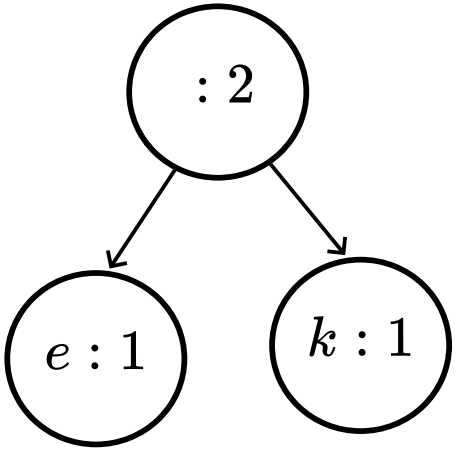


- Code
  - $\tau(A) = 00$, $\tau(B) = 01$, $\tau(C) = 10$, $\tau(D) = 110$, $\tau(E) = 111$

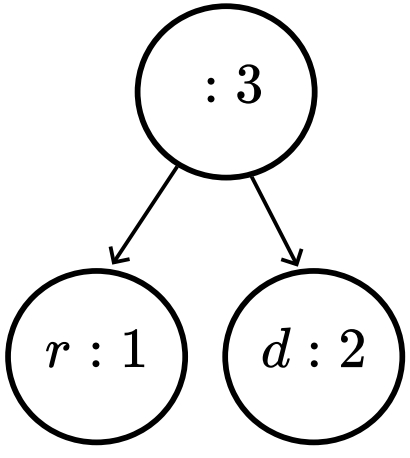
- Exemple : avadakedavra

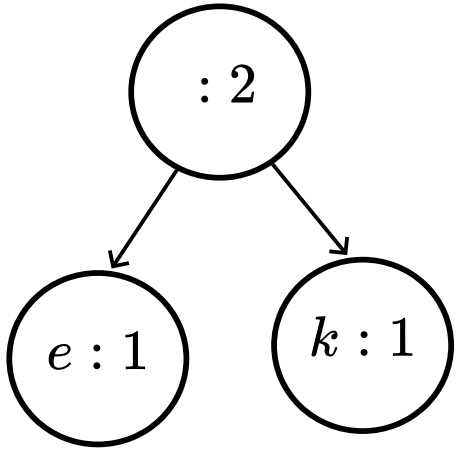
- Exemple : avadakedavra
  - Fréquences : {a:5,d:2,e:1,k:1,r:1,v:2}

- Exemple : avadakedavra
  - Fréquences : {a:5,d:2,e:1,k:1,r:1,v:2}
  - Arbre de Huffman

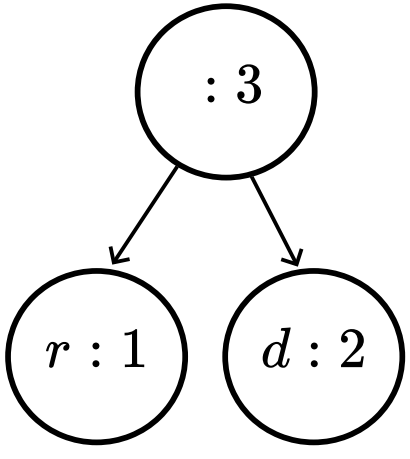


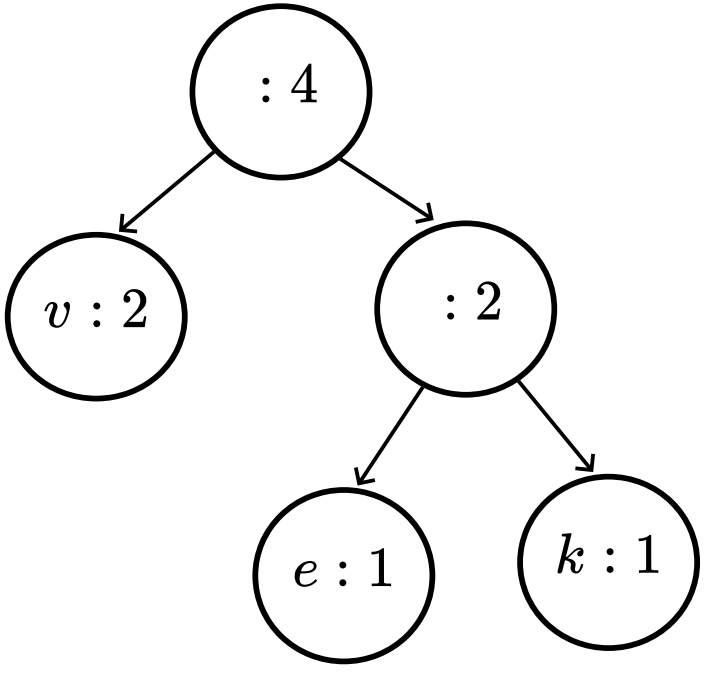
- Exemple : avadakedavra
  - Fréquences : {a:5,d:2,e:1,k:1,r:1,v:2}
  - Arbre de Huffman



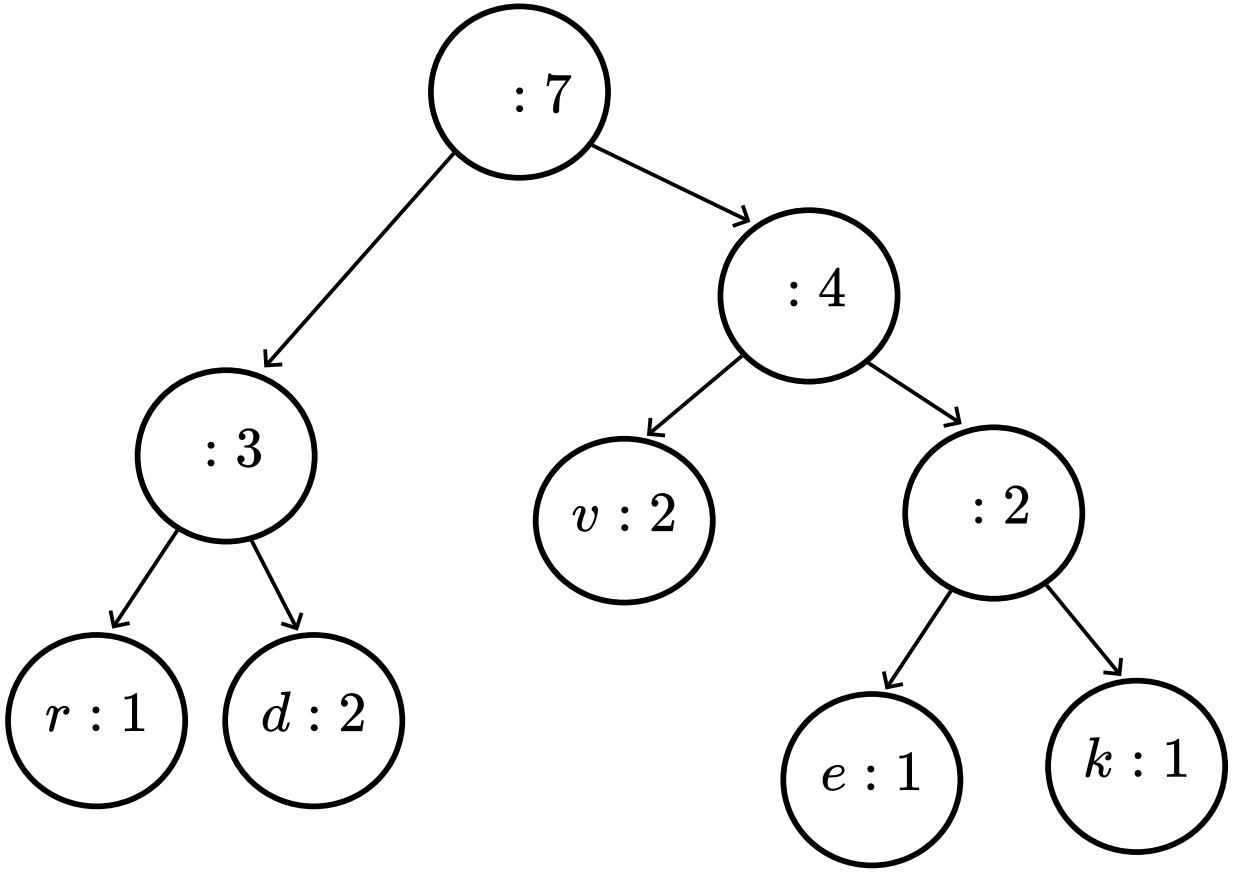


- Exemple : avadakedavra
  - Fréquences : {a:5,d:2,e:1,k:1,r:1,v:2}
  - Arbre de Huffman

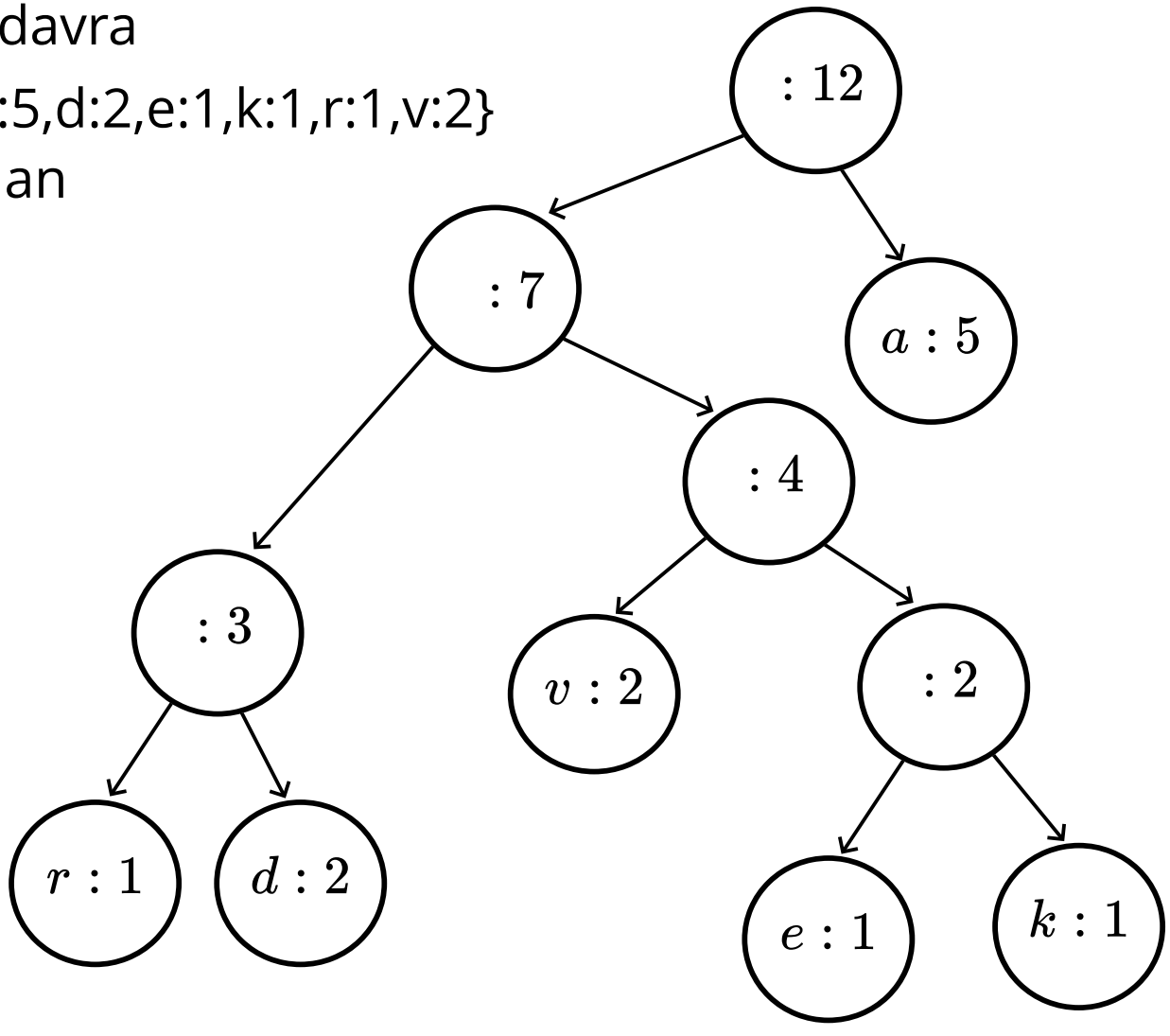




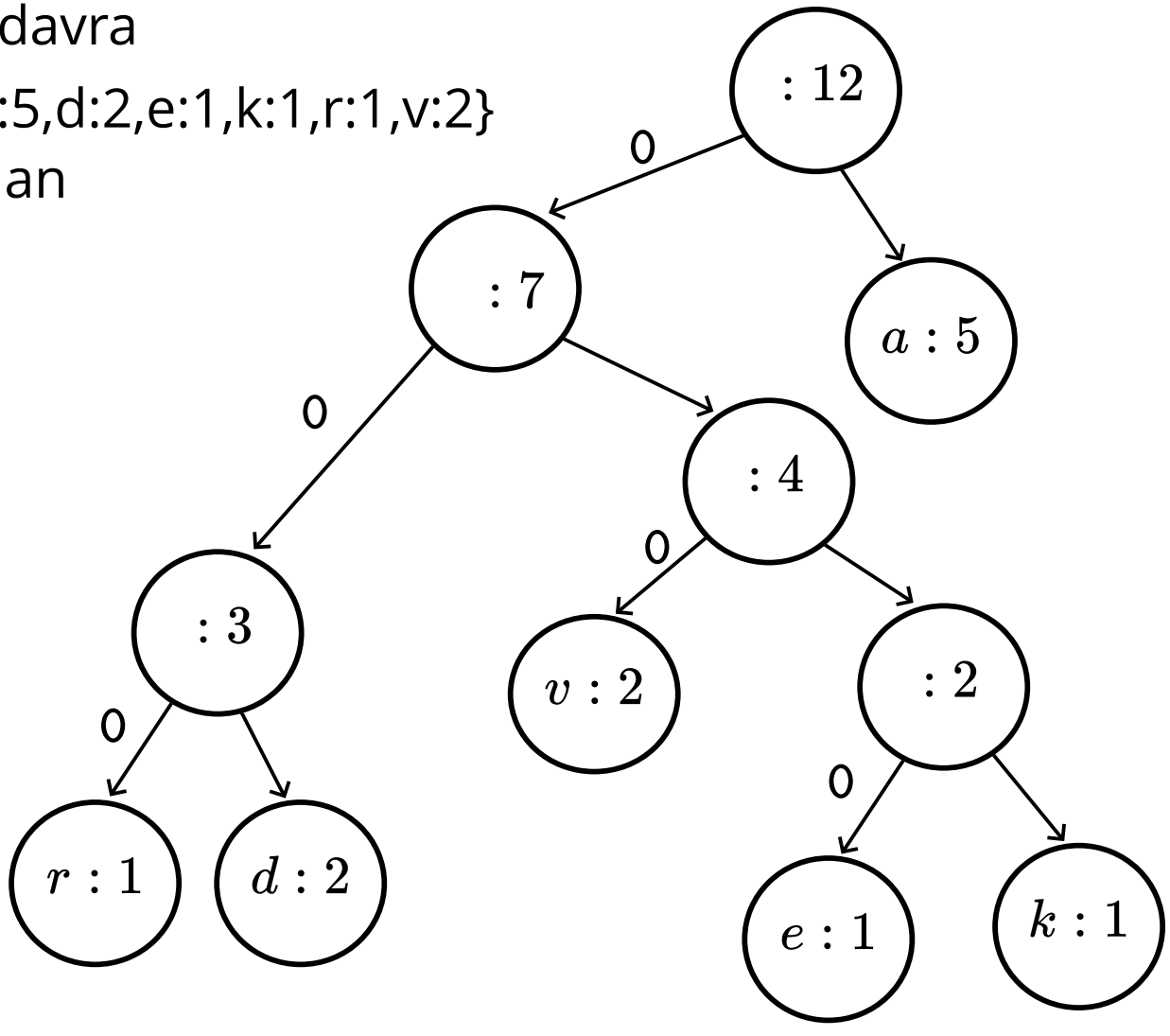
- Exemple : avadakedavra
  - Fréquences : {a:5,d:2,e:1,k:1,r:1,v:2}
  - Arbre de Huffman



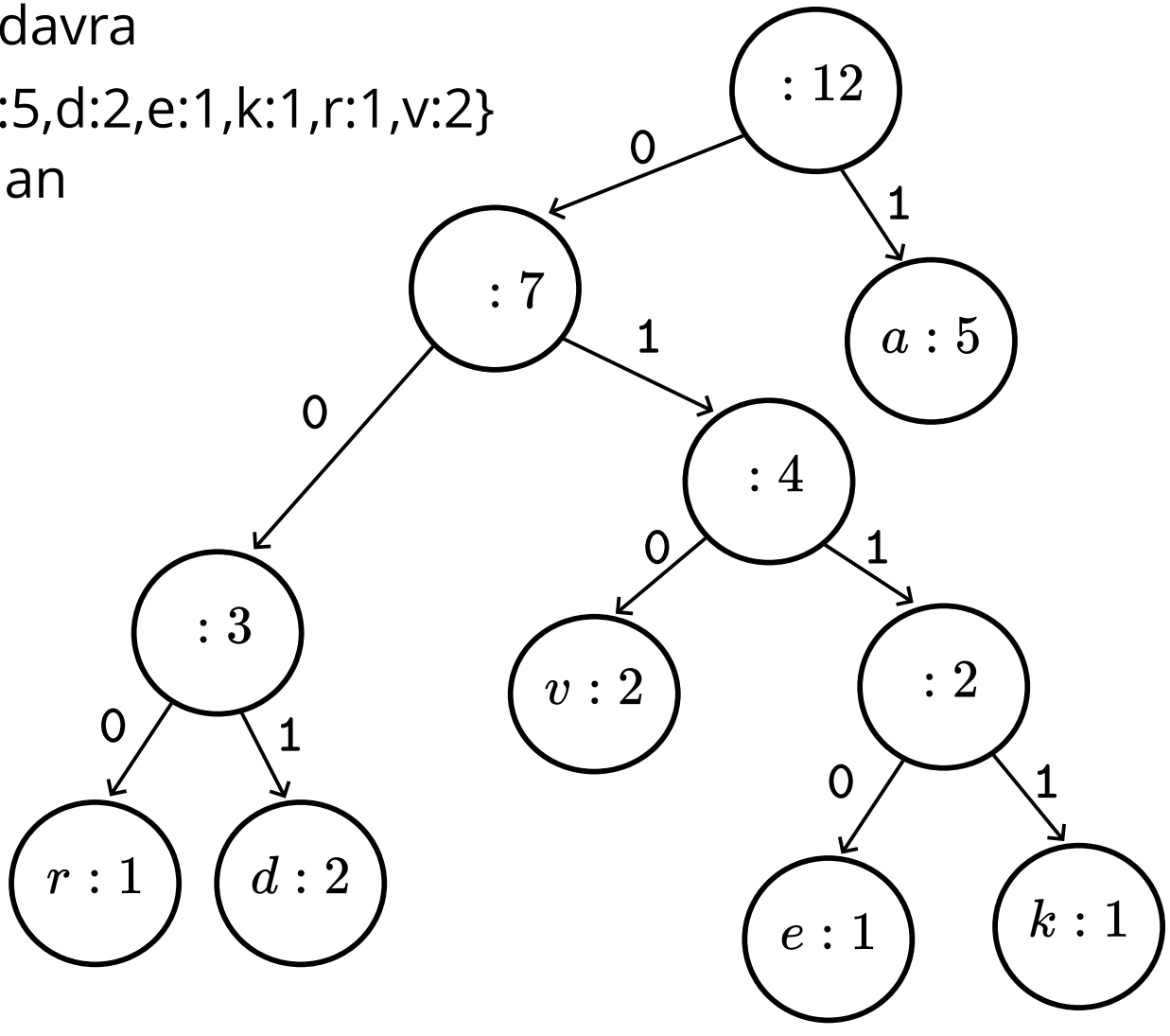
- Exemple : avadakedavra
  - Fréquences : {a:5,d:2,e:1,k:1,r:1,v:2}
  - Arbre de Huffman



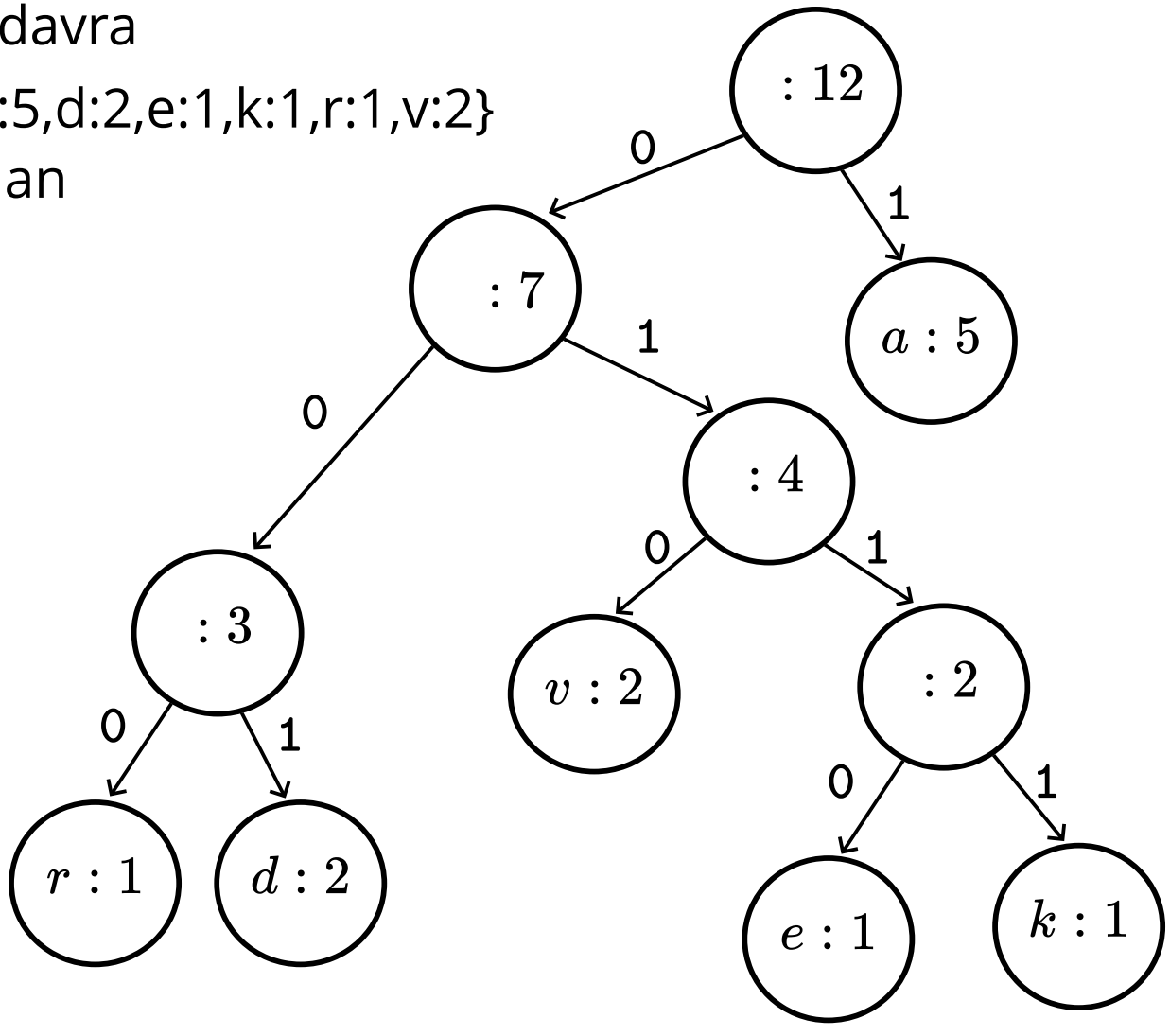
- Exemple : avadakedavra
  - Fréquences : {a:5,d:2,e:1,k:1,r:1,v:2}
  - Arbre de Huffman



- Exemple : avadakedavra
  - Fréquences : {a:5,d:2,e:1,k:1,r:1,v:2}
  - Arbre de Huffman



- Exemple : avadakedavra
  - Fréquences : {a:5,d:2,e:1,k:1,r:1,v:2}
  - Arbre de Huffman
  - Code
    - $\tau(a) = 1$
    - $\tau(d) = 001$
    - $\tau(e) = 0110$
    - $\tau(k) = 0111$
    - $\tau(r) = 000$
    - $\tau(v) = 010$



- Exemple : avadakedavra (fréquences : {a:5,d:2,e:1,k:1,r:1,v:2})
  - Code : $\tau(a) = 1$, $\tau(d) = 001$, $\tau(e) = 0110$, $\tau(k) = 0111$, $\tau(r) = 000$ et $\tau(v) = 010$

- Exemple : avadakedavra (fréquences : {a:5,d:2,e:1,k:1,r:1,v:2})
  - Code : $\tau(a) = 1$, $\tau(d) = 001$, $\tau(e) = 0110$, $\tau(k) = 0111$, $\tau(r) = 000$ et $\tau(v) = 010$
  - Mot codé :

a	v	a	d	a	k	e	d	a	v	r	a
1	010	1	001	1	0111	0110	001	1	010	000	1

- Exemple : avadakedavra (fréquences : {a:5,d:2,e:1,k:1,r:1,v:2})
  - Code : $\tau(a) = 1$, $\tau(d) = 001$, $\tau(e) = 0110$, $\tau(k) = 0111$, $\tau(r) = 000$ et $\tau(v) = 010$
  - Mot codé :

a	v	a	d	a	k	e	d	a	v	r	a
1	010	1	001	1	0111	0110	001	1	010	000	1

  - Nombre de bits
    - $1 + 3 + 1 + 3 + 1 + 4 + 4 + 3 + 1 + 3 + 3 + 1 = 28$ bits

- Exemple : avadakedavra (fréquences : {a:5,d:2,e:1,k:1,r:1,v:2})
  - Code : $\tau(a) = 1$, $\tau(d) = 001$, $\tau(e) = 0110$, $\tau(k) = 0111$, $\tau(r) = 000$ et $\tau(v) = 010$
  - Mot codé :

a	v	a	d	a	k	e	d	a	v	r	a
1	010	1	001	1	0111	0110	001	1	010	000	1

  - Nombre de bits
    - $1 + 3 + 1 + 3 + 1 + 4 + 4 + 3 + 1 + 3 + 3 + 1 = 28$ bits
    - Équivalente : $5 \cdot 1 + 2 \cdot 3 + 1 \cdot 4 + 1 \cdot 4 + 1 \cdot 3 + 2 \cdot 3 = 28$ bits

- Exemple : avadakedavra (fréquences : {a:5,d:2,e:1,k:1,r:1,v:2})
  - Code : $\tau(a) = 1$, $\tau(d) = 001$, $\tau(e) = 0110$, $\tau(k) = 0111$, $\tau(r) = 000$ et $\tau(v) = 010$
  - Mot codé :

a	v	a	d	a	k	e	d	a	v	r	a
1	010	1	001	1	0111	0110	001	1	010	000	1

  - Nombre de bits
    - $1 + 3 + 1 + 3 + 1 + 4 + 4 + 3 + 1 + 3 + 3 + 1 = 28$ bits
    - Équivalente : $5 \cdot 1 + 2 \cdot 3 + 1 \cdot 4 + 1 \cdot 4 + 1 \cdot 3 + 2 \cdot 3 = 28$ bits
  - Code de longueur fixe : $\lceil \log_2(5) \rceil = 3 \Rightarrow 12 \cdot 3 = 36$ bits

- Exemple : avadakedavra (fréquences : {a:5,d:2,e:1,k:1,r:1,v:2})
  - Code : $\tau(a) = 1$, $\tau(d) = 001$, $\tau(e) = 0110$, $\tau(k) = 0111$, $\tau(r) = 000$ et $\tau(v) = 010$
  - Mot codé :

a	v	a	d	a	k	e	d	a	v	r	a
1	010	1	001	1	0111	0110	001	1	010	000	1

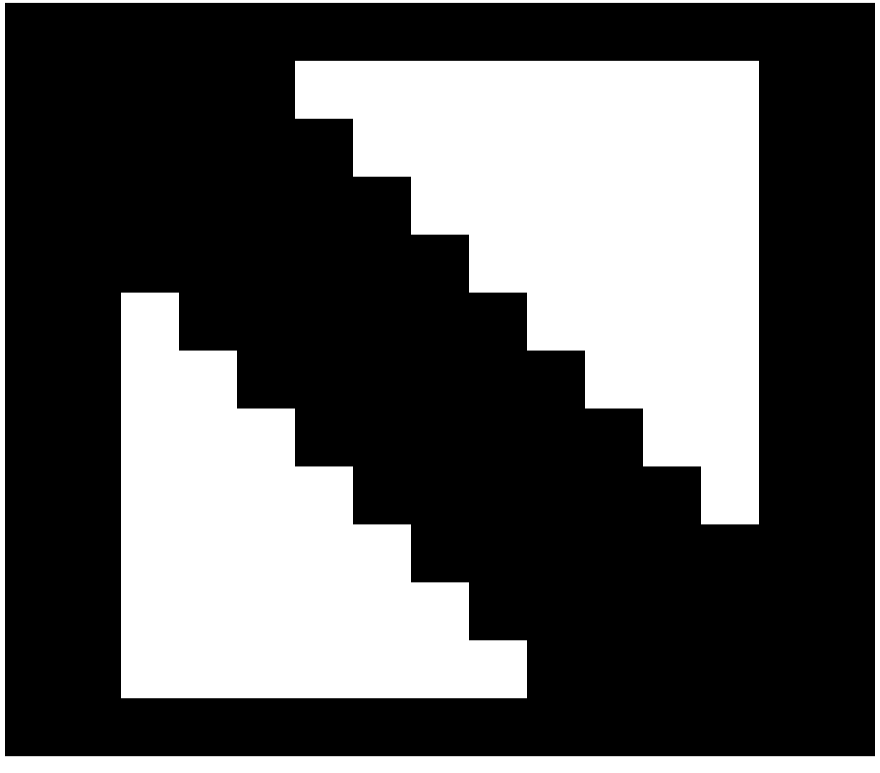
  - Nombre de bits
    - $1 + 3 + 1 + 3 + 1 + 4 + 4 + 3 + 1 + 3 + 3 + 1 = 28$ bits
    - Équivalente : $5 \cdot 1 + 2 \cdot 3 + 1 \cdot 4 + 1 \cdot 4 + 1 \cdot 3 + 2 \cdot 3 = 28$ bits
  - Code de longueur fixe : $\lceil \log_2(5) \rceil = 3 \Rightarrow 12 \cdot 3 = 36$ bits
  - Taux de compression
    - $28/36 \approx 0,77$, c'est-à-dire 77%
    - Ou $1 - 28/36 \approx 0,23$, c'est-à-dire 23%

- Il existe d'autres types de compression sans perte (vous les étudierez plus tard)
  - Par dictionnaire
  - Par contexte
- L'outil **zip** utilise par défaut l'algorithme DEFLATE qui est une variante du codage de Huffman
- L'outil **bzip2** utilise l'algorithme de Burrows-Wheeler ainsi qu'un codage de Huffman
- Les outils **compress** et **gzip** utilisent l'algorithme de Lempel-Ziv
- L'outil **lzip** utilise l'algorithme Lempel-Zip Markov chains

- La compression d'image recouvre de nombreuses techniques très différentes
  - Les meilleures compressions sont avec perte
  - Sur certaines images on peut vraiment gagner même sans perte
- Prenons les images fabriquées avec un ordinateur
  - Beaucoup d'entre elles ont de larges bandes uniformes
  - On peut exploiter cette propriété
- Le codage **RLE** (Run-Length Encoding) s'applique assez bien aux images en noir et blanc
  - On a pour chaque ligne une alternance de bandes de pixels noirs et pixels blancs
  - On va donc simplement coder la longueur de ces bandes
    - L'on commence toujours par le nombre de 0

- La compression d'image recouvre de nombreuses techniques très différentes
  - Les meilleures compressions sont avec perte
  - Sur certaines images on peut vraiment gagner même sans perte
- Prenons les images fabriquées avec un ordinateur
  - Beaucoup d'entre elles ont de larges bandes uniformes
  - On peut exploiter cette propriété
- Le codage **RLE** (Run-Length Encoding) s'applique assez bien aux images en noir et blanc
  - On a pour chaque ligne une alternance de bandes de pixels noirs et pixels blancs
  - On va donc simplement coder la longueur de ces bande

- Exemple



```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Pour chaque ligne, on va coder la suite des longueurs
  - Par convention, l'on commence toujours par le nombre de 0

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Pour chaque ligne, on va coder la suite des longueurs
  - Par convention, l'on commence toujours par le nombre de 0

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Ligne 1 : 0,15 (2 nombres)

- Pour chaque ligne, on va coder la suite des longueurs
  - Par convention, l'on commence toujours par le nombre de 0

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Ligne 1 : 0,15 (2 nombres)
- Ligne 2 : 0,15 (2 nombres)

- Pour chaque ligne, on va coder la suite des longueurs
  - Par convention, l'on commence toujours par le nombre de 0

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0
0 0 1 1 1 1 0 0 0 0 0 0 1 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Ligne 1 : 0,15 (2 nombres)
- Ligne 2 : 0,15 (2 nombres)
- Ligne 3 : 15 (1 nombre)

- Pour chaque ligne, on va coder la suite des longueurs
  - Par convention, l'on commence toujours par le nombre de 0

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0
0 0 1 0 0 0 0 0 0 1 1 1 1 0 0
0 0 1 1 0 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 0 1 1 0 0
0 0 1 1 1 1 0 0 0 0 0 0 1 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Ligne 1 : 0,15 (2 nombres)
- Ligne 2 : 0,15 (2 nombres)
- Ligne 3 : 15 (1 nombre)
- Ligne 4 : 5,8,2 (3 nombres)

- Pour chaque ligne, on va coder la suite des longueurs
  - Par convention, l'on commence toujours par le nombre de 0

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Ligne 1 : 0,15 (2 nombres)
- Ligne 2 : 0,15 (2 nombres)
- Ligne 3 : 15 (1 nombre)
- Ligne 4 : 5,8,2 (3 nombres)
- Ligne 5 : 6,7,2 (3 nombres)

- Pour chaque ligne, on va coder la suite des longueurs
  - Par convention, l'on commence toujours par le nombre de 0

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0
0 0 1 0 0 0 0 0 0 1 1 1 1 0 0
0 0 1 1 0 0 0 0 0 0 1 1 1 0 0
0 0 1 1 1 0 0 0 0 0 0 1 1 0 0
0 0 1 1 1 1 0 0 0 0 0 0 1 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Ligne 1 : 0,15 (2 nombres)
- Ligne 2 : 0,15 (2 nombres)
- Ligne 3 : 15 (1 nombre)
- Ligne 4 : 5,8,2 (3 nombres)
- Ligne 5 : 6,7,2 (3 nombres)
- Ligne 6 : 7,6,2 (3 nombres)
- Ligne 7 : 8,5,2 (3 nombres)
- Ligne 8 : 2,1,6,4,2 (4 nombres)
- Ligne 9 : 2,2,6,3,2 (4 nombres)
- Ligne 10 : 2,3,6,2,2 (4 nombres)
- Ligne 11 : 2,4,6,1,2 (4 nombres)
- Ligne 12 : 2,5,8 (3 nombres)
- Ligne 13 : 2,6,7 (3 nombres)
- Ligne 14 : 2,7,6 (3 nombres)
- Ligne 15 : 15 (1 nombre)
- Ligne 16 : 0,15 (2 nombres)
- Ligne 17 : 0,15 (2 nombres)

- Codage RLE

0,15,0,15,15,5,8,2,6,7,2,7,6,2,8,5,2,2,1,6,4,2,2,2,6,3,2,2,3,6,2,2,2,4,6,1,2,,2,5,8,2,6,7,2,7,6,15,0,15,0,15

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Codage RLE

0,15,0,15,15,5,8,2,6,7,2,7,6,2,8,5,2,2,1,6,4,2,2,2,6,3,2,2,3,6,2,2,2,4,6,1,2,,2,5,8,2,6,7,2,7,6,15,0,15,0,15

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- 47 nombres entre 0 et 15

- Codage RLE

0,15,0,15,15,5,8,2,6,7,2,7,6,2,8,5,2,2,1,6,4,2,2,2,6,3,2,2,3,6,2,2,2,4,6,1,2,,2,5,8,2,6,7,2,7,6,15,0,15,0,15

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- 47 nombres entre 0 et 15
  - Il nous faut 4 bits par nombre

- Codage RLE

0,15,0,15,15,5,8,2,6,7,2,7,6,2,8,5,2,2,1,6,4,2,2,2,6,3,2,2,3,6,2,2,2,4,6,1,2,,2,5,8,2,6,7,2,7,6,15,0,15,0,15

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- 47 nombres entre 0 et 15
  - Il nous faut 4 bits par nombre
- $188 = 47 \times 4$ bits

- Codage RLE

0,15,0,15,15,5,8,2,6,7,2,7,6,2,8,5,2,2,1,6,4,2,2,2,6,3,2,2,3,6,2,2,2,4,6,1,2,,2,5,8,2,6,7,2,7,6,15,0,15,0,15

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- 47 nombres entre 0 et 15
  - Il nous faut 4 bits par nombre
- $188 = 47 \times 4$ bits
  - Au lieu de $15 \times 17 = 255$ bits

- Codage RLE

0,15,0,15,15,5,8,2,6,7,2,7,6,2,8,5,2,2,1,6,4,2,2,2,6,3,2,2,3,6,2,2,2,4,6,1,2,,2,5,8,2,6,7,2,7,6,15,0,15,0,15

```
P2
15 17
1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 0 0 0 0 0 0 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 1 0 0 0
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- 47 nombres entre 0 et 15
  - Il nous faut 4 bits par nombre
- $188 = 47 \times 4$ bits
  - Au lieu de $15 \times 17 = 255$ bits
- Taux de compression
  - $188/255 \approx 0,74$

- RLE est un **codage par répétition**
  - Cette technique est employée pour le fax
    - On y utilise d'abord le RLE afin d'obtenir une suite de nombres
    - Puis on utilise Huffman pour coder la suite de nombres
- Les autres types de compression d'images (avec ou sans perte) font appel à des techniques trop avancées pour être décrites ici
  - Ils reposent sur l'exploitation de particularités des images
    - .gif, .png pour les images numériques
    - .jpeg pour les photographies