Module EA4 – Éléments d'Algorithmique

Dominique Poulalhon
dominique.poulalhon@liafa.univ-paris-diderot.fr

Université Paris Diderot
L2 Informatique, Math-Info et EIDD
Année universitaire 2013-2014

Interrogation n° 3 *facultative* lundi 5 mai

Inscription obligatoire sur Didel avant ce soir

mardi 20 mai de 12h à 15h

Documents autorisés

uniquement une feuille A4 manuscrite

(recto-verso)

mardi 20 mai de 12h à 15h

Documents autorisés

uniquement une feuille A4 manuscrite
(recto-verso)

Autrement dit :

tout autre document (manuscrit, imprimé ou électronique) *interdit*

matériel électronique *éteint et rangé*

problème

étant donné un fichier `texte` de taille ℓ, construire un fichier `compresse(texte)` de taille strictement inférieure à ℓ tel que `texte` puisse être reconstruit à partir de `compresse(texte)`

problème

étant donné un fichier `texte` de taille ℓ, construire un fichier `compresse(texte)` de taille strictement inférieure à ℓ tel que `texte` puisse être reconstruit à partir de `compresse(texte)`

(aparté)

il s'agit ici de compression sans perte, contrairement à ce qu'on a vu avec le hachage, ou, surtout, ce qu'on peut autoriser pour la compression de données perceptibles (image, son, video) pour laquelle on utilise le fait que l'oeil (ou l'oreille) ne peut pas capter tous les détails

problème

étant donné un fichier `texte` de taille ℓ, construire un fichier `compresse(texte)` de taille strictement inférieure à ℓ tel que `texte` puisse être reconstruit à partir de `compresse(texte)`

Malheureusement, un simple comptage montre :

Théorème

Aucun algorithme ne peut compresser tous les fichiers possibles.

problème

étant donné un fichier `texte` de taille ℓ, construire un fichier `compresse(texte)` de taille strictement inférieure à ℓ tel que `texte` puisse être reconstruit à partir de `compresse(texte)`

Malheureusement, un simple comptage montre :

Théorème

Aucun algorithme ne peut compresser tous les fichiers possibles.

Alors ?

de quoi part-on ?

d'un `texte` dont chaque caractère est codé en `ASCII` (sur un octet), ou éventuellement en `unicode` (4 octets pour l'`UTF32`)

de quoi part-on ?

d'un `texte` dont chaque caractère est codé en `ASCII` (sur un octet), ou éventuellement en `unicode` (4 octets pour l'`UTF32`)

un octet par caractère pour un alphabet de 26 lettres ?

de quoi part-on ?

d'un `texte` dont chaque caractère est codé en `ASCII` (sur un octet), ou éventuellement en `unicode` (4 octets pour l'`UTF32`)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules, $26 \times 2 = 52$

de quoi part-on ?

d'un `texte` dont chaque caractère est codé en `ASCII` (sur un octet), ou éventuellement en `unicode` (4 octets pour l'`UTF32`)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules, $26 \times 2 = 52$
- accentuées ou non, $> 12 \times 2 = 24$

de quoi part-on ?

d'un `texte` dont chaque caractère est codé en `ASCII` (sur un octet), ou éventuellement en `unicode` (4 octets pour l'`UTF32`)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules, $26 \times 2 = 52$
- accentuées ou non, $> 12 \times 2 = 24$
- des caractères de ponctuation, espaces... > 10

de quoi part-on ?

d'un `texte` dont chaque caractère est codé en `ASCII` (sur un octet), ou éventuellement en `unicode` (4 octets pour l'`UTF32`)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules, $26 \times 2 = 52$
- accentuées ou non, $> 12 \times 2 = 24$
- des caractères de ponctuation, espaces... > 10
- des chiffres 10

de quoi part-on ?

d'un `texte` dont chaque caractère est codé en `ASCII` (sur un octet), ou éventuellement en `unicode` (4 octets pour l'`UTF32`)

un octet par caractère pour un alphabet de 26 lettres ?

- majuscules ou minuscules, $26 \times 2 = 52$
- accentuées ou non, $> 12 \times 2 = 24$
- des caractères de ponctuation, espaces... > 10
- des chiffres 10

soit une centaine de caractères, nécessitant donc 7 bits chacun

si chaque caractère est codé avec le même nombre de bits

constat

tous ces caractères n'ont pas du tout la même fréquence dans un texte écrit en français, par exemple

Code de longueur variable

constat

tous ces caractères n'ont pas du tout la même fréquence dans un texte écrit en français, par exemple

corollaire

la probabilité uniforme sur les textes de longueur donnée n'est pas pertinente, et il est peut-être possible de compresser les textes « réels » (au détriment des « textes » n'ayant aucun sens)

Code de longueur variable

constat

tous ces caractères n'ont pas du tout la même fréquence dans un texte écrit en français, par exemple

corollaire

la probabilité uniforme sur les textes de longueur donnée n'est pas pertinente, et il est peut-être possible de compresser les textes « réels » (au détriment des « textes » n'ayant aucun sens)

idée

donner des mots de code de longueur *variable* aux lettres en fonction de leur *fréquence* dans le texte considéré : (très) courts pour les lettres fréquentes, plus longs pour les lettres rares (et ignorer les caractères qui n'apparaissent pas)

Codage...

- utiliser une *table de hachage* pour stocker les mots de code des différents caractères
- coder le texte par simple *concaténation* des codes des caractères qui le composent

Codage...

- utiliser une *table de hachage* pour stocker les mots de code des différents caractères
- coder le texte par simple *concaténation* des codes des caractères qui le composent

en ASCII

```
>>> texte = 'abracadabra'
>>> ascii = { 'a' : '01100001', 'b' : '01100010',
'c' : '01100011', 'd' : '01100100', 'r' : '01110010' }
>>> ''.join(ascii[c] for c in texte)
'01100001011000100111001001100001011000110110000101100
10001100001011000100111001001100001'
>>> len(''.join(ascii[c] for c in texte))
88
```

Codage...

avec un code « mini-ASCII »

```
>>> texte = 'abracadabra'                    # longueur 11
>>> dico = { 'a' : '001', 'b' : '010', 'c' : '011',
             'd' : '100', 'r' : '101' }
>>> ''.join(dico[c] for c in texte)
'00101010100101100110000101010100 1'         # longueur 33
```

Codage...

avec un code « mini-ASCII »

```
>>> texte = 'abracadabra'                    # longueur 11
>>> dico = { 'a' : '001', 'b' : '010', 'c' : '011',
             'd' : '100', 'r' : '101' }
>>> ''.join(dico[c] for c in texte)
'00101010100101100110000101010101001'        # longueur 33
```

avec un code de longueur variable

```
>>> huffman = { 'a' : '0', 'b' : '10', 'r' : '110',
                'c' : '1110', 'd' : '1111' }
>>> ''.join(huffman[c] for c in texte)
'01011001110011110101100'                    # longueur 23
```

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '00101010100101100110000101010100 1'
```

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'

texte_decode = 'a
```

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

```
texte_encode = '001010101001011001100001010101001'

texte_decode = 'ab
```

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

`texte_encode` = '001010101001011001100001010101001'

`texte_decode` = 'abr

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

`texte_encode` = '00101010100101100110000101010100 1'

`texte_decode` = 'abra

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

`texte_encode` = '00101010100101100110000101010100 1'

`texte_decode` = 'abrac

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

`texte_encode` = '00101010100101100110000101010100 1'

`texte_decode` = 'abraca

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

`texte_encode` = '00101010100101100110000101010101001'

`texte_decode` = 'abracad

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

`texte_encode` = '00101010100101100110000101010100 1'

`texte_decode` = 'abracada

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

`texte_encode` = '00101010100101100110000101010101001'

`texte_decode` = 'abracadab

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

`texte_encode` = '00101010100101100110000101010101001'

`texte_decode` = 'abracadabr

avec un code « mini-ASCII », c'est facile

caractère	a	b	c	d	r
mot de code	001	010	011	100	101

`texte_encode` = '00101010100101100110000101010100 1'

`texte_decode` = 'abracadabra'

avec un code « mini-ASCII », c'est facile

```
>>> code = ''.join(dico[c] for c in texte)
>>> decoupe = [ code[i:i+3] for i in range(0,33,3) ]
['001', '010', '101', '001', '011', '001', '100',
'001', '010', '101', '001']
>>> inverse = { v : k for (k,v) in dico.items() }
{ '011': 'c', '010': 'b', '100': 'd', '001': 'a',
'101': 'r' }
>>> ''.join(inverse[elt] for elt in decoupe)
'abracadabra'
```

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '010110011100111101011 00'

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0101100111001111010110 0'
```

```
texte_decode = 'a
```

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

```
texte_encode = '0 10110011100111101011100'

texte_decode = 'a
```

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10110011100111101011 00'

`texte_decode` = 'ab

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 1100111001111010 1100'

`texte_decode` = 'ab

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 1100111001111010110 0'

`texte_decode` = 'ab

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 1100111100111101011100'

`texte_decode` = 'abr

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 01110011110101100'

`texte_decode` = 'abra

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110011110101100'

`texte_decode` = 'abra

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 11100111110101100'

`texte_decode` = 'abra

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110011111010100'

`texte_decode` = 'abra

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110011111010100'

`texte_decode` = 'abrac

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0111101011 00'

`texte_decode` = 'abraca

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 11110101100'

`texte_decode` = 'abraca

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 11110101100'

`texte_decode` = 'abraca

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 111101011100'

`texte_decode` = 'abraca

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 11110101100'

`texte_decode` = 'abracad

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0101100'

`texte_decode` = 'abracada

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 101100'

`texte_decode` = 'abracada

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 101100'

`texte_decode` = 'abracadab

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 10 1100'

`texte_decode` = 'abracadab

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 10 1100'

`texte_decode` = 'abracadab

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 10 1100'

`texte_decode` = 'abracadabr

avec un code de longueur variable, c'est plus difficile

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

`texte_encode` = '0 10 110 0 1110 0 1111 0 10 110 0'

`texte_decode` = 'abracadabra'

pourquoi avons-nous réussi ?

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

pourquoi avons-nous réussi ?

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110

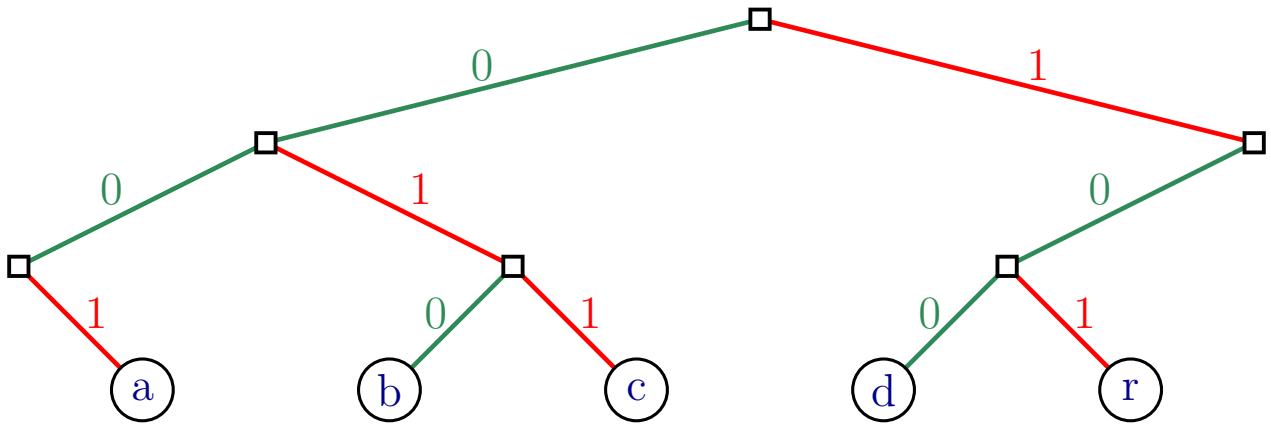
car aucun mot de code n'est préfixe d'un autre mot de code

c'est la définition d'un code préfixe

autre formulation

les mots du code sont les (codes des) feuilles d'un arbre binaire

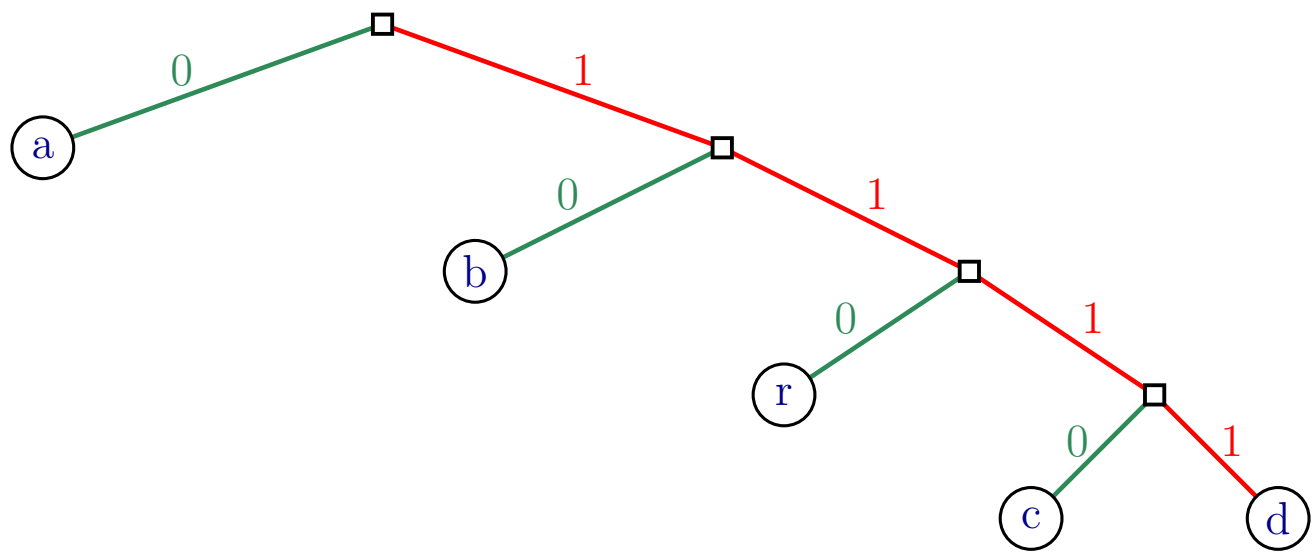
caractère	a	b	c	d	r
mot de code	001	010	011	100	101



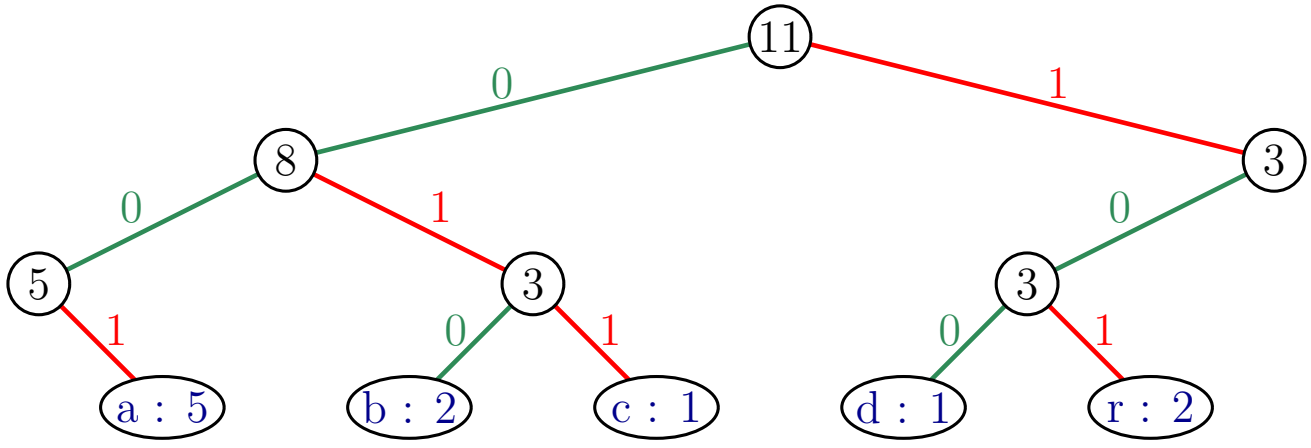
autre formulation

les mots du code sont les (codes des) feuilles d'un arbre binaire

caractère	a	b	c	d	r
mot de code	0	10	1110	1111	110



Comparaison de taux de compression



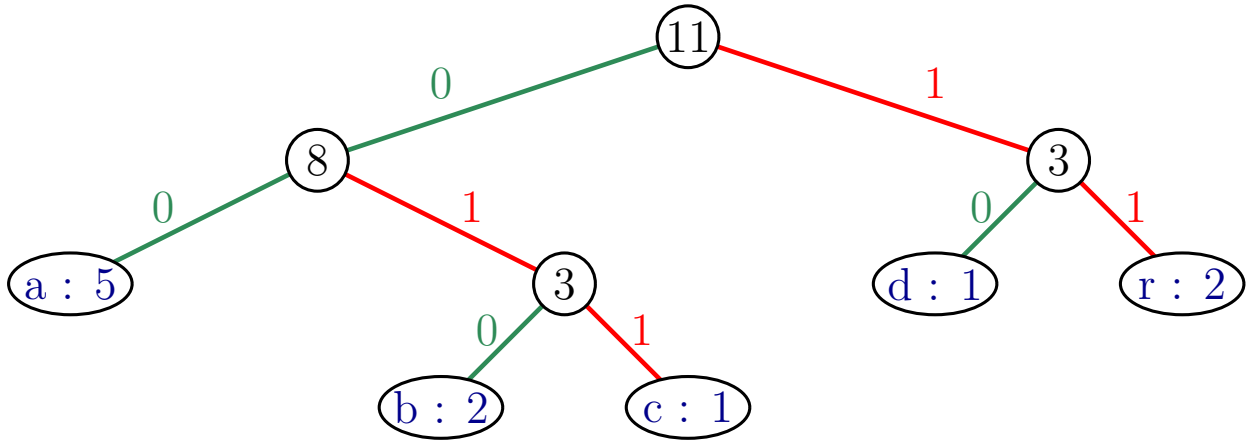
pour chaque caractère c, soit

- $n(c)$ le nombre d'occurrences de c dans le texte
- $h(c)$ la longueur du code de c (*i.e.* sa hauteur dans l'arbre)

longueur totale du texte codé :

$$\sum_c n(c)h(c) = 33$$

Comparaison de taux de compression



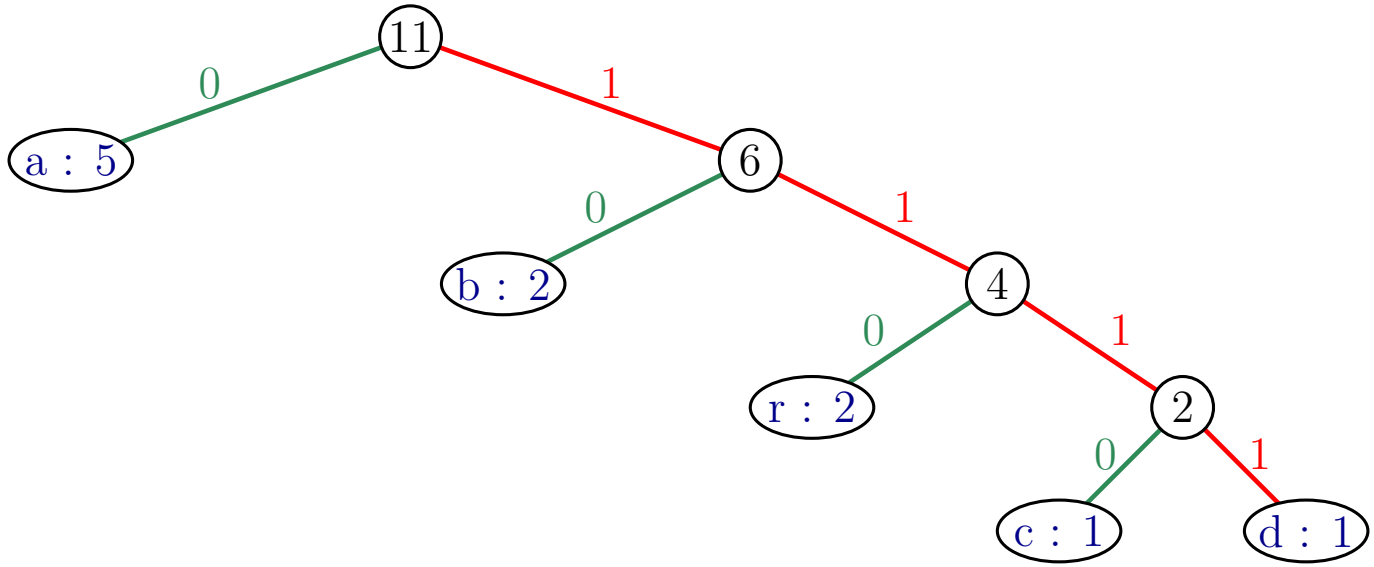
pour chaque caractère c, soit

- $n(c)$ le nombre d'occurrences de c dans le texte
- $h(c)$ la longueur du code de c (*i.e.* sa hauteur dans l'arbre)

longueur totale du texte codé :

$$\sum_c n(c)h(c) = 25$$

Comparaison de taux de compression



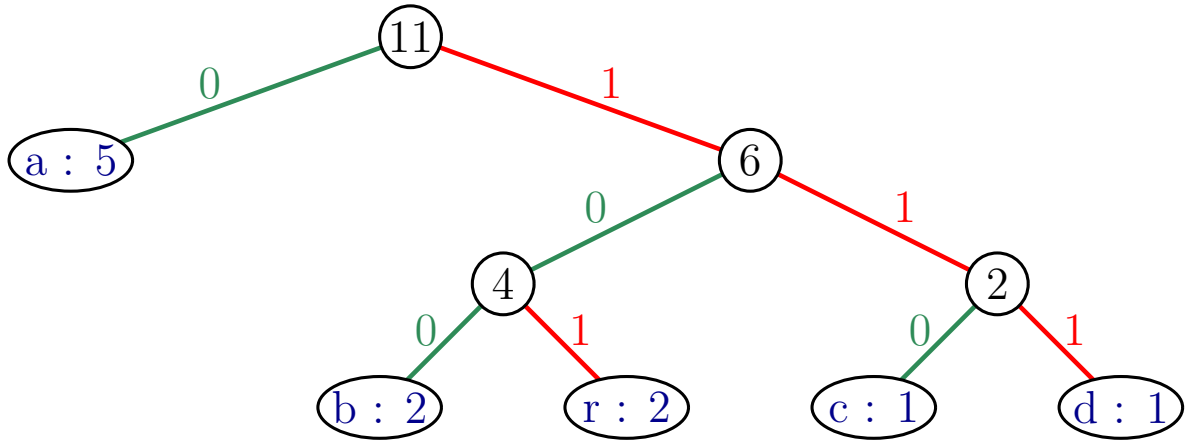
pour chaque caractère c, soit

- $n(c)$ le nombre d'occurrences de c dans le texte
- $h(c)$ la longueur du code de c (*i.e.* sa hauteur dans l'arbre)

longueur totale du texte codé :

$$\sum_c n(c)h(c) = 23$$

Comparaison de taux de compression



pour chaque caractère c, soit

- $n(c)$ le nombre d'occurrences de c dans le texte
- $h(c)$ la longueur du code de c (*i.e.* sa hauteur dans l'arbre)

longueur totale du texte codé :

$$\sum_c n(c)h(c) = 23$$

Comment ces deux derniers codes sont-ils obtenus ?

codage de Huffman

construire l'arbre de manière *gloutonne* :

- placer les couples (`caractère`, `fréquence`) dans une file de priorité
- tant que la file de priorité contient plusieurs éléments
  - extraire les deux éléments de fréquence minimale
  - créer un nœud binaire dont ces éléments sont les fils
  - insérer le nœud dans la file de priorité
- retourner l'unique élément de la file de priorité – la racine de l'arbre de code

Rappel : files de priorité

Comparaison des différentes implémentations vues en cours :

	liste (chaînée)		??
	non triée	triée	
insertion	$\Theta(1)$	$\Theta(n)$	
extraction du minimum	$\Theta(n)$	$\Theta(1)$	

Rappel : files de priorité

Comparaison des différentes implémentations vues en cours :

	liste (chaînée)		tas
	non triée	triée	
insertion	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$
extraction du minimum	$\Theta(n)$	$\Theta(1)$	$\Theta(\log n)$

Déroulement de l'algorithme

File : (a : 5) (b : 2) (r : 2) (c : 1) (d : 1)

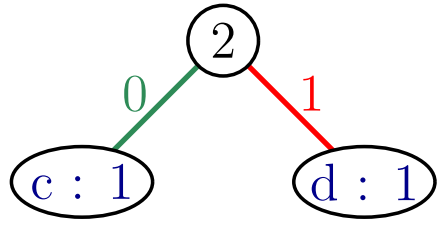
Déroulement de l'algorithme

File : (a : 5) (b : 2) (r : 2)

(c : 1) (d : 1)

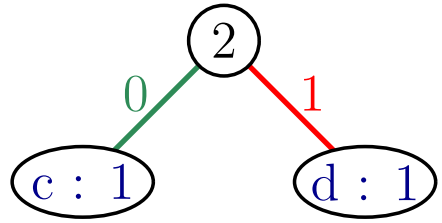
Déroulement de l'algorithme

File : a : 5 b : 2 r : 2



Déroulement de l'algorithme

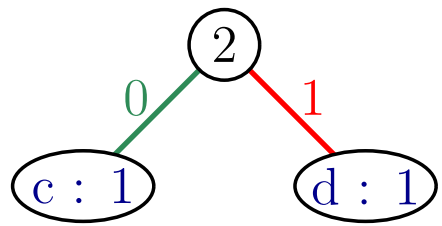
File : a : 5 b : 2 r : 2



Déroulement de l'algorithme

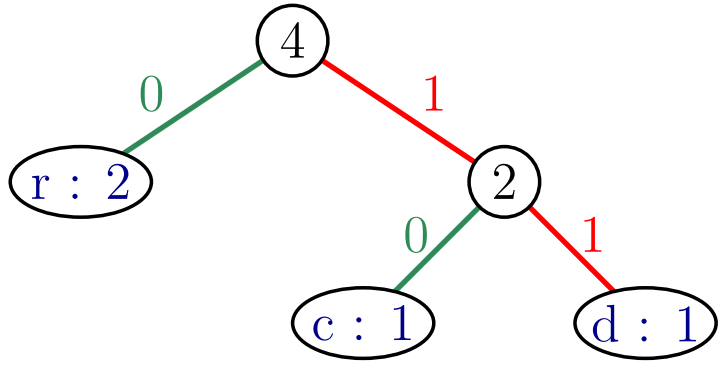
File : a : 5 b : 2

r : 2



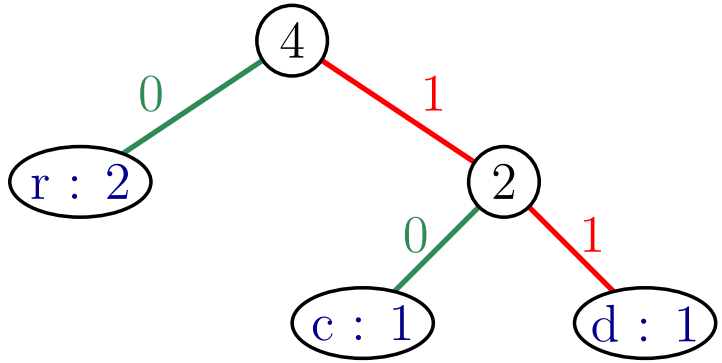
Déroulement de l'algorithme

File : a : 5 b : 2



Déroulement de l'algorithme

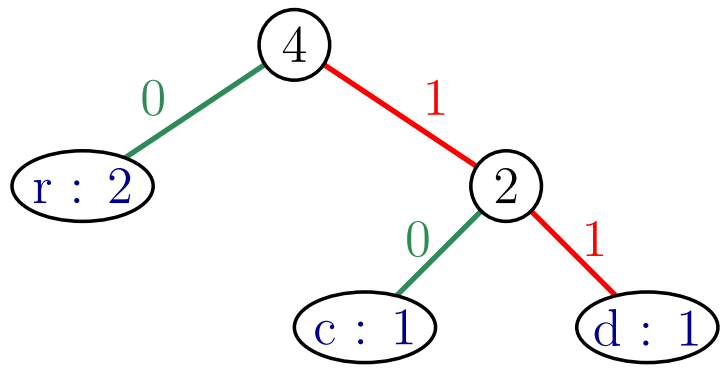
File : a : 5 b : 2



Déroulement de l'algorithme

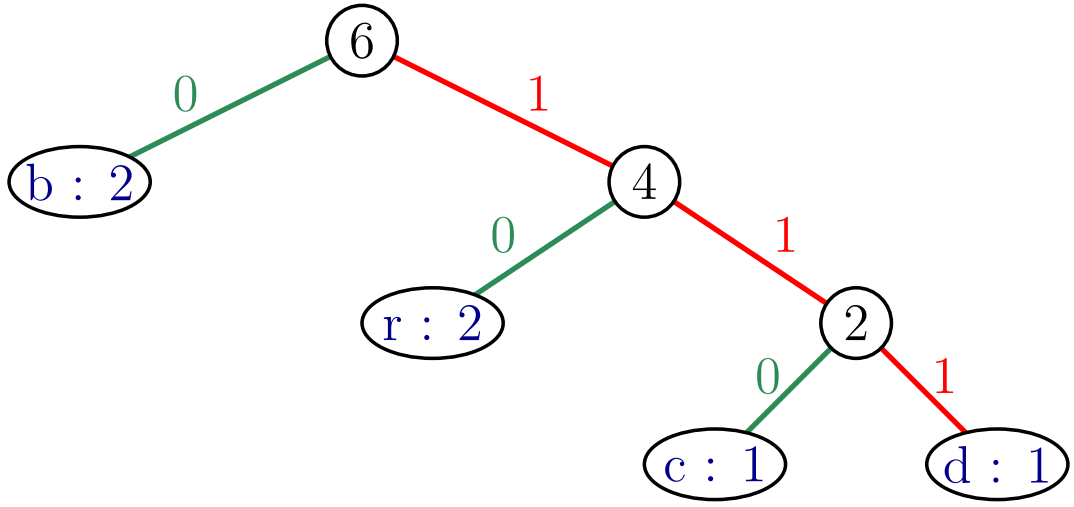
File : a : 5

b : 2



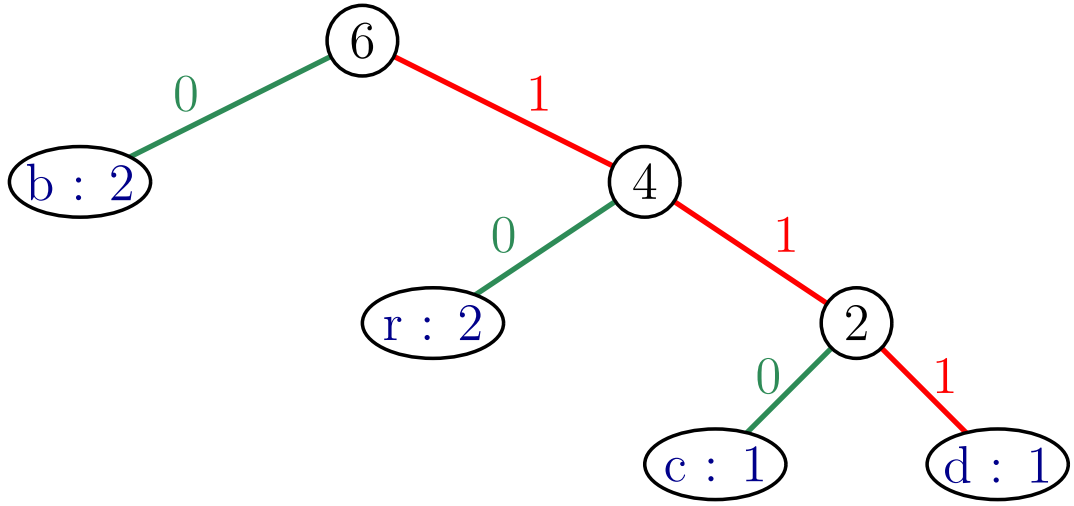
Déroulement de l'algorithme

File : a : 5

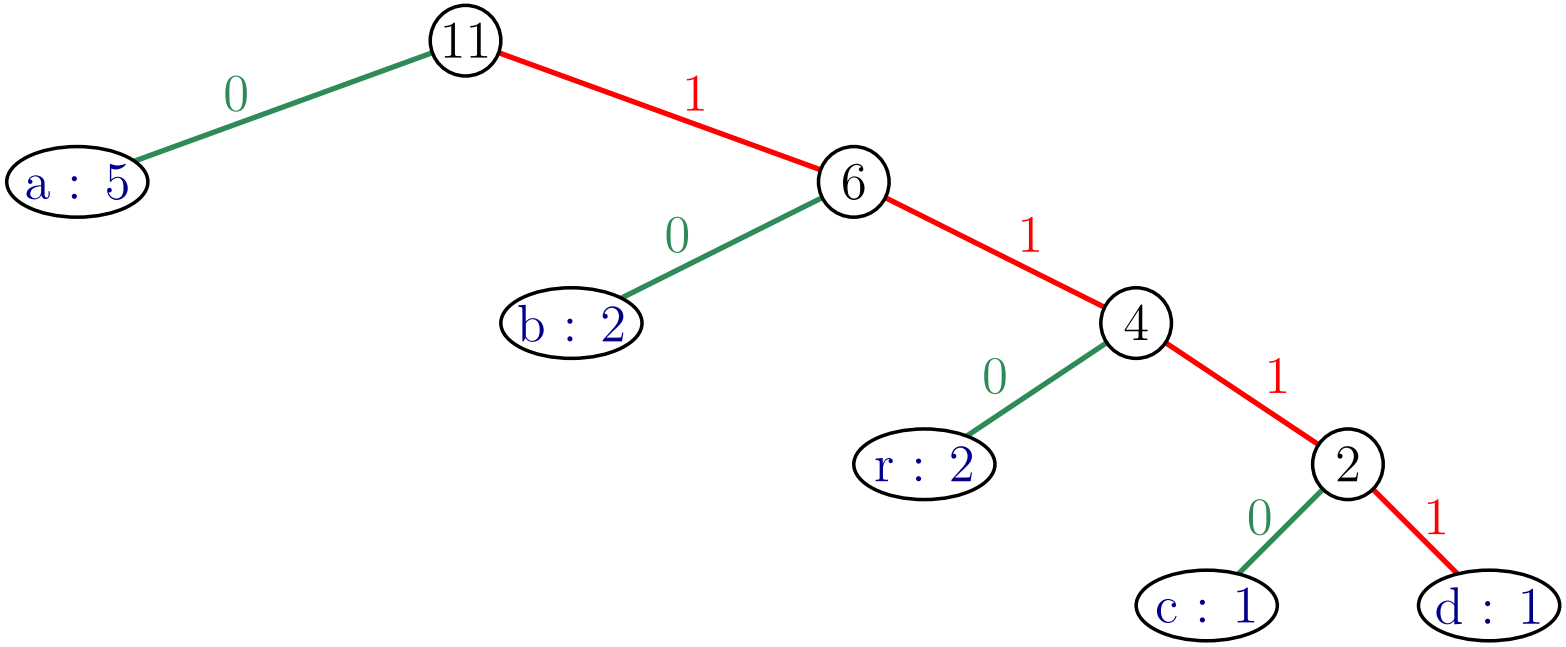


Déroulement de l'algorithme

File : a : 5



Déroulement de l'algorithme



Théorème

le code produit par l'algorithme de Huffman est optimal parmi les codes préfixes

Théorème

le code produit par l'algorithme de Huffman est optimal parmi les codes préfixes

Lemme

soit x *et* y *deux caractères de fréquence minimale, alors il existe un code préfixe optimal* φ *tel que* $\varphi(x) = w \cdot 0$ *et* $\varphi(y) = w \cdot 1$ *pour un même mot* w

Théorème

le code produit par l'algorithme de Huffman est optimal parmi les codes préfixes

Lemme

soit x *et* y *deux caractères de fréquence minimale, alors il existe un code préfixe optimal tel que* x *et* y *ont le même père dans l'arbre associé*

Théorème

le code produit par l'algorithme de Huffman est optimal parmi les codes préfixes

Lemme

soit x *et* y *deux caractères de fréquence minimale, alors il existe un code préfixe optimal tel que* x *et* y *ont le même père dans l'arbre associé*

Lemme

soit x *et* y *deux caractères ayant le même père dans un arbre de codage optimal* T *; soit* T' *obtenu à partir de* T *en supprimant* x *et* y*, et en associant un caractère* z *à la nouvelle feuille. Alors* T' *est un arbre de codage optimal.*

Points négatifs

- on n'a pas compté le coût de la transmission du dictionnaire
- l'algorithme nécessite un précalcul des fréquences, donc une prélecture du texte, ce qui n'est pas toujours possible

Points négatifs

- on n'a pas compté le coût de la transmission du dictionnaire
- l'algorithme nécessite un précalcul des fréquences, donc une prélecture du texte, ce qui n'est pas toujours possible

- si la langue du texte est connue, on peut utiliser une version *statique* avec un dictionnaire prédéfini... mais on perd nécessairement l'optimalité

Points négatifs

- on n'a pas compté le coût de la transmission du dictionnaire
- l'algorithme nécessite un précalcul des fréquences, donc une prélecture du texte, ce qui n'est pas toujours possible

- si la langue du texte est connue, on peut utiliser une version *statique* avec un dictionnaire prédéfini... mais on perd nécessairement l'optimalité
- il existe une version *adaptative* qui recalcule les fréquences au fur et à mesure de la lecture (et du codage) ; elle compresse encore mieux, *mais* la complexité en temps est bien plus grande puisque l'arbre doit être recalculé à chaque caractère