Algorithmique des arbres

Codes, Arbres pondérés, Codage de Huffman

L2 Mathématique et Informatique

Outline

Compression d'un texte, sans perte de données

Objectif : Représenter une suite d'octets *(le texte)* par une suite d'octets plus courte *(le compressé)* de sorte que le texte reconstruit à partir du compressé soit identique à l'original.

Idée générale : Substituer un mot à une lettre !

Idée 1 : $\rightsquigarrow$ chaque lettre est codée par un mot de même longueur

Idée 2 : $\rightsquigarrow$ les lettres sont codées par des mots de longueurs variables :

LES LETTRES APPARAISSANT LE PLUS GRAND NOMBRE DE FOIS SONT ASSOCIÉES À DES MOTS LES PLUS COURTS POSSIBLES.

LES LETTRES APPARAISSANT PEU DE FOIS SONT ASSOCIÉES À DES MOTS POTENTIELLEMENT LONGS.

$\Longrightarrow$ Souvent, on gagne beaucoup ; rarement, on perd.

Longueur d'un texte compréssé

Propriété :

Si une lettre a est une lettre du texte apparaissant $nb(a)$ fois dans le texte et est associé à un mot de longueur $l(a)$, alors la longueur du texte compressé est

$$L(\text{compressé}) = \sum_{a \in texte} nb(a) \cdot l(a) \ .$$

Objectif : Minimiser la quantité $L(\text{compressé})$.

Exemple

Exemple :

Imaginons un texte de 1000 caractères dont le nombre d'occurences des lettres est donné par :

Lettre	a	b	c	d	e	f
Nombre d'occurence	350	330	20	160	90	50

Codage 1 :

Lettre	a	b	c	d	e	f
Codage	000	001	010	011	100	101

$\Longrightarrow L(\text{compressé}) = 3000$ bits

Codage 2 :

Lettre	a	b	c	d	e	f
Codage	0	11	10000	101	1001	10001

$\Longrightarrow L(\text{compressé}) = 350 \cdot 1 + 330 \cdot 2 + 20 \cdot 5 + 160 \cdot 3 + 90 \cdot 4 + 50 \cdot 5 = 2200$ bits

Cela représente un gain d'un peu plus de 25%

Outline

Notion d'alphabet

Définition :

Soit A un ensemble, appelé un alphabet.
Un mot sur un alphabet A est une suite finie d'éléments de A.

Exemples :

- Avec l'alphabet $A = \{0, 1, \cdots, 9\}$, on construit des nombres :

$$2381,\ 0023,\ \cdots$$

- Avec l'alphabet $A = \{0, 1\}$, on construit des mots dits binaires :

$$00100111$$

- Avec l'alphabet $A = \{a, b, \cdots, z\}$, on construit des "mots classiques" :

$$\text{code},\ \text{huffman},\ \cdots$$

Notion de code

Définition :

Soit A un ensemble, appelé alphabet.

Un code sur un alphabet A est un ensemble C de mots construits sur A tels qu'une concaténation de mots de C ne peut être séparée en mots de C que d'une unique façon :

Pour tous entiers n et p, pour tous mots $x_1, \cdots, x_n, y_1, \cdots, y_p \in C$,

$$x_0 x_1 \dots x_n = y_0 y_1 \dots y_p \Longrightarrow \begin{cases} p = n \\ \forall i \in [\![1; n]\!] \ , \ x_i = y_i \end{cases} \quad (1)$$

Définition :

Un code est dit binaire s'il est construit sur un alphabet à deux éléments. *(On identifiera alors ces deux lettres aux entiers* 0 *et* 1*).*

Exemples

- $C_0 = \{001, 100, 010\}$ est un code : tous ses mots sont de même longueur
- $C_1 = \{1, 01, 001, \ldots, 0^n1, n \in N\}$ est un code : 1 est séparateur
- $C_2 = \{00, 1, 10\}$ est un code, mais le délai de décodage est non borné :
  - $\rightsquigarrow$ attendre le prochain 1 ou la fin
  - $\rightsquigarrow$ découper la séquence précédant le 1 de la gauche vers la droite

$$1001000101110 = 1|00|10|00|10|1|1|10$$

$$100000000\ldots = 1|00|00|00|00|\ldots \text{ ou } 10|00|00|00|0\ldots \text{ ?}$$

Contre-exemple

- $C_3 = \{01, 1, 10, 11, 100\}$ n'est pas un code :

$$10011 = 100|11 = 10|01|1 \ .$$

Code préfixe

Définition :

Un code préfixe est un ensemble C de mots sur un alphabet A dans lequel aucun mot du code n'est préfixe d'un autre mot du code.

Remarque : Un code préfixe devrait plutôt s'appeler un code sans préfixe, mais ce n'est pas la terminologie qui a été retenue...

Contre-exemple :

$C_2 = \{00, 1, 10\}$ est un code, mais il n'est pas préfixe :
$\rightsquigarrow$ 1 est un préfixe de 10 C_2 est un code suffixe !

Exemple :

$C_4 = \{00, 01, 100, 1010, 1011\}$ est un code préfixe.

Code préfixe et arbre binaire

Propriété :

L'ensemble des codes préfixes sur un alphabet à deux lettres est en bijection avec l'ensemble des arbres binaires.

Preuve :

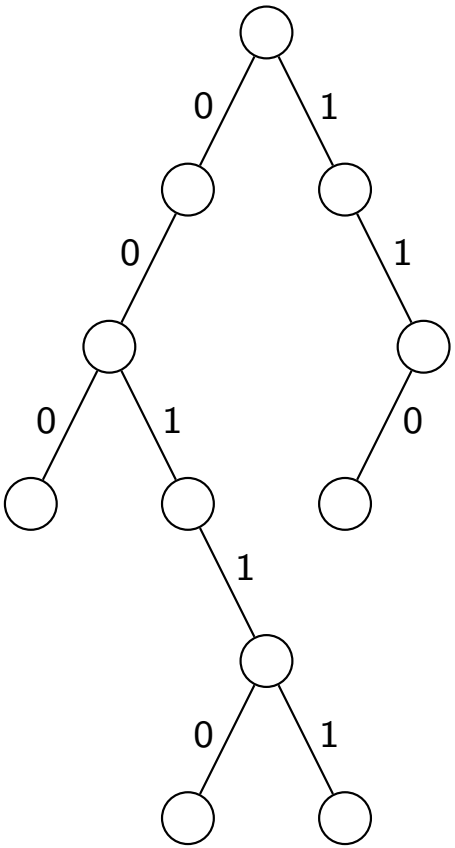
- Etant donné un arbre binaire, on étiquette ses arêtes :
  - ⇝ 0 pour une arête gauche ;
  - ⇝ 1 pour une arête droite ;

  on retrouve le code préfixe parcourant l'arbre binaire de la racine à chaque feuille.

- Etant donné un code préfixe, on ajoute successivement dans un arbre binaire initialement vide une branche pour que chaque mot se retrouve codé comme un chemin de la racine à une nouvelle feuille (0 pour une arête gauche, 1 pour une arête droite)

Exemples 1/2

Exemple :

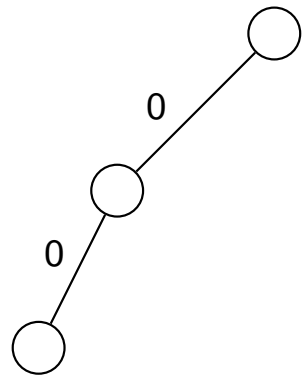


$\rightsquigarrow \qquad \mathrm{C} = \{000; 00110; 00111; 110\}$

Code binaire 2/2

Exemple :

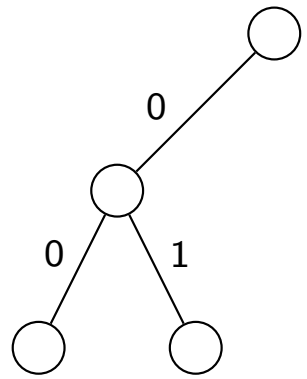
$$\mathrm{C}_4 = \{00; 01; 100; 1010; 1011\} \quad \rightsquigarrow$$



Code binaire 2/2

Exemple :

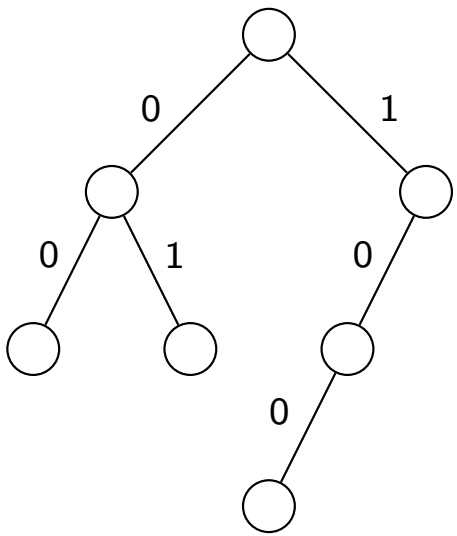
$$\mathrm{C}_4 = \{00; 01; 100; 1010; 1011\} \quad \rightsquigarrow$$



Code binaire 2/2

Exemple :

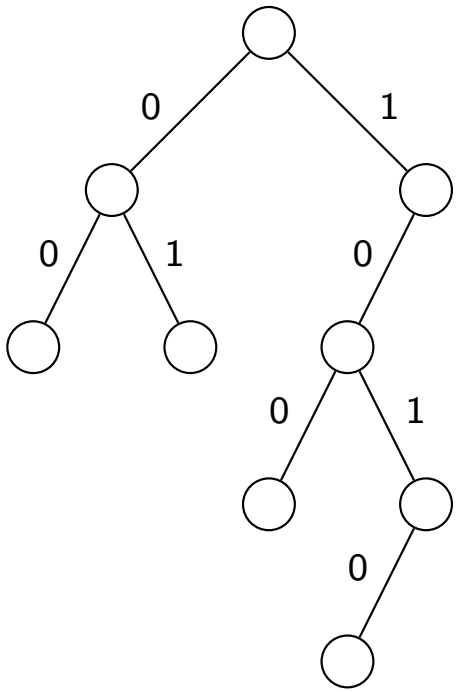
$C_4 = \{00; 01; 100; 1010; 1011\}$ $\rightsquigarrow$



Code binaire 2/2

Exemple :

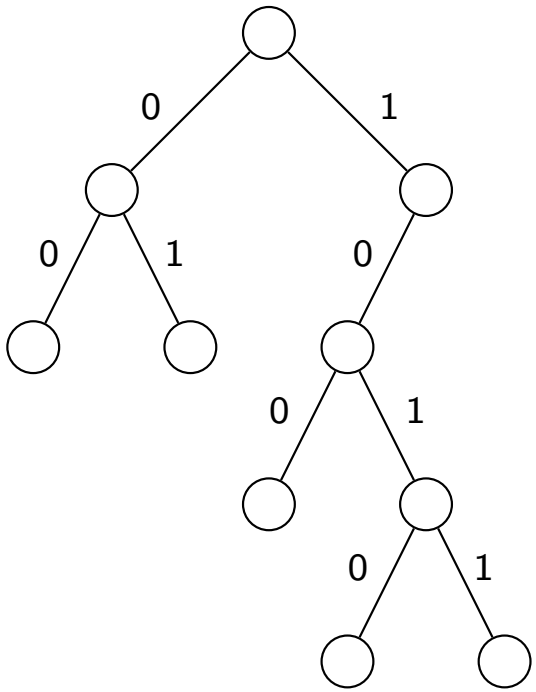
$$\mathrm{C}_4 = \{00; 01; 100; 1010; 1011\} \quad \rightsquigarrow$$



Code binaire 2/2

Exemple :

$$\mathrm{C}_4 = \{00; 01; 100; 1010; 1011\} \quad \rightsquigarrow$$



Code binaire et factorisation d'un message

Propriété :

Un code préfixe sur un alphabet à deux lettres est un code : la factorisation d'une suite de mot construite à partir du code préfixe est unique.

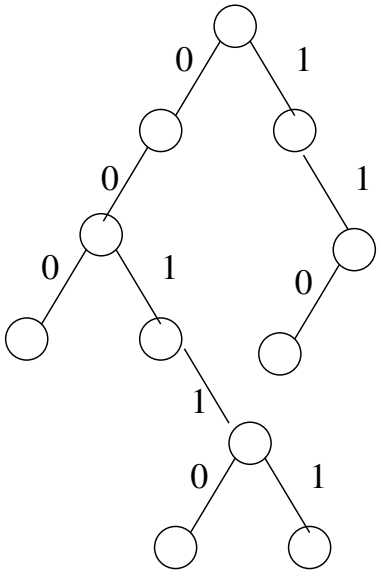
Preuve :

Parcours de l'arbre binaire associé au code binaire à partir de la racine : on sépare en arrivant à une feuille et on repart à la racine.

1100000011000110

↓

110|000|00110|00110



C={000,00110,00111,110}

Code binaire et factorisation d'un message

Propriété :

Un code préfixe est un code.

Preuve :

• Si l'alphabet associé au code possède N lettres, on associe au code préfixe un arbre N-aire (tous les nœuds ont exactement N enfants, éventuellement `NULL`)

• Parcours de l'arbre associé au code à partir de la racine :
on sépare en arrivant à une feuille et on repart à la racine.

Outline

Généralités

Objectif : Représenter une suite d'octets *(le texte)* par une suite d'octets plus courte *(le compressé)* de sorte que le texte reconstruit à partir du compressé soit identique à l'original.

$\Longrightarrow$ Compression de données sans perte

Le codage de Huffman est optimal parmi les codes qui substituent un mot à une lettre : il minimise un *"cout"*.

Application : le fax, la compressions de certaines images

Remarque : Il existe des méthodes de compression sans perte de donnée beaucoup plus efficace que l'algorithme de Huffman.

Outline

Rappel

Rappel :

Si une lettre a est une lettre d'un texte à compresser apparaissant $nb(a)$ fois et est associé à un mot de longueur $l(a)$, alors la longueur du texte compressé vaut

$$L(\text{compressé}) = \sum_{a \in texte} nb(a) \cdot l(a) \ .$$

Implémentons cette formule sur un arbre !

Arbre pondéré et cout

Etant donné un arbre A :

- $\rightsquigarrow$ on se donne une fonction de poids p définies sur les feuilles de A et à valeurs positives ;
- $\rightsquigarrow$ le poids d'un nœud interne est la somme des poids de ses enfants ;

- $\rightsquigarrow$ le coût d'une feuille est le produit de son poids par son niveau ;
- $\rightsquigarrow$ le coût d'un arbre est la somme des coûts de ses feuilles :

$$\text{Cout}(A) = \sum_{f \text{ feuille de A}} \mathit{poids}(f) \cdot \mathit{niveau}(f)$$

Arbre pondéré et cout

Etant donné un arbre A :

- $\rightsquigarrow$ on se donne une fonction de poids p définies sur les feuilles de A et à valeurs positives ;
  $p(f) =$ nombre d'occurrence de l'étiquette de la feuille dans le texte à compresser)
- $\rightsquigarrow$ le poids d'un nœud interne est la somme des poids de ses enfants ;

- $\rightsquigarrow$ le coût d'une feuille est le produit de son poids par son niveau ;
  niveau = longueur du mot codant l'étiquette de la feuille
- $\rightsquigarrow$ le coût d'un arbre est la somme des coûts de ses feuilles :

$$\text{Cout}(A) = \sum_{f \text{ feuille de A}} poids(f) \cdot niveau(f)$$

$$Cout(A) \longleftrightarrow L(\text{compréssé})$$

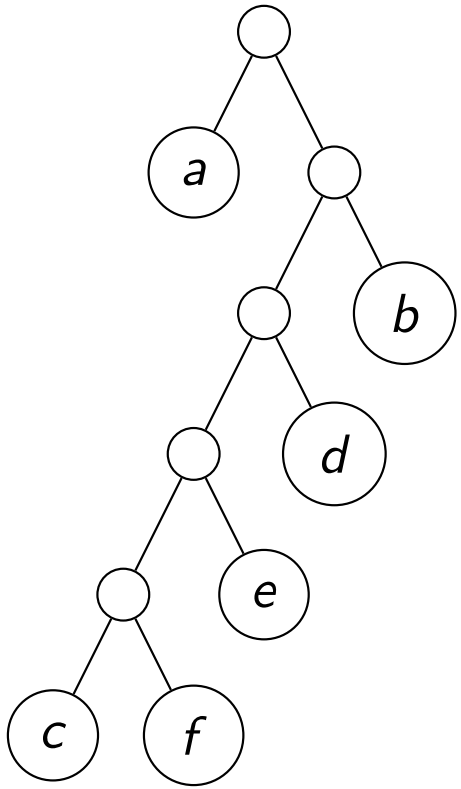
Recherche de minimisation du cout

Pour une fonction de poids donné, il existe des arbres de coût minimum. Ces arbres sont localement complets (tous les nœuds ont 0 ou 2 enfants).

Considérons la fonction de poids suivante :

$$\begin{cases} p(a) = 350 \\ p(b) = 330 \\ p(c) = 20 \\ p(d) = 160 \\ p(e) = 90 \\ p(f) = 50 \end{cases}$$

Considérons l'arbre A suivant :



Alors :

$$Cout(A) = p(a) + 2p(b) + 3p(d) + 4p(e) + 5\Big(p(c) + p(f)\Big) = 2200$$

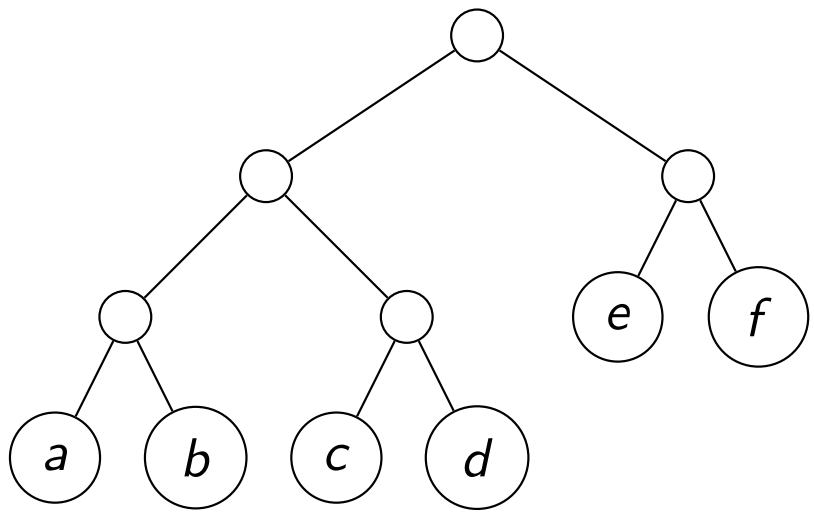
Recherche de minimisation du cout

Pour une fonction de poids donné, il existe des arbres de coût minimum. Ces arbres sont localement complets (tous les nœuds ont 0 ou 2 enfants).

Considérons la fonction de poids suivante :

$$\begin{cases} p(a) = 350 \\ p(b) = 330 \\ p(c) = 20 \\ p(d) = 160 \\ p(e) = 90 \\ p(f) = 50 \end{cases}$$

Considérons l'arbre A suivant :



Alors :

$$Cout(A) = 3\Big(p(a) + p(b) + p(c) + p(d)\Big) + 2\Big(p(e) + p(f)\Big) = 2860$$

Outline

Principe du codage de Huffman

Le code de Huffman est un code préfixe de coût minimal construit grâce à un arbre pondéré de coût minimum : les feuilles représentent les lettres, leur poids est le nombre d'occurence dans le texte.

- Une première passe du texte détermine le nombre d'occurence de chaque lettre.
- On construit un arbre pondéré de coût minimum.
- On construit le codage de chaque lettre : l'étiquette de la branche qui mène de la racine à la feuille contenant la lettre.
- Une deuxième passe construit le texte compréssé en remplaçant chaque lettre par son codage.

Principe du codage de Huffman

Le code de Huffman est un code préfixe de coût minimal construit grâce à un arbre pondéré de coût minimum : les feuilles représentent les lettres, leur poids est le nombre d'occurence dans le texte.

- Une première passe du texte détermine le nombre d'occurence de chaque lettre.
- On construit un arbre pondéré de coût minimum.

  $\rightsquigarrow$ *En utilisant une file de priorité !*
- On construit le codage de chaque lettre : l'étiquette de la branche qui mène de la racine à la feuille contenant la lettre.
- Une deuxième passe construit le texte compréssé en remplaçant chaque lettre par son codage.

Rappel : type abstrait de données `File de priorité`

Définition :

Soit C un ensemble quelconque et E un ensemble ordonné.
Une file de priorité est un ensemble de couples $(\texttt{cle},\texttt{priorite}) \in C \times E$, sur lequel on souhaite pouvoir réaliser les opérations suivantes :

- $\texttt{maximum} : \texttt{File} \longrightarrow C \times E$ ou $\texttt{minimum} : \texttt{File} \longrightarrow C \times E$
- $\texttt{inserer} : \texttt{File} \times C \times E \longrightarrow \texttt{File}$;
- $\texttt{supprimer} : \texttt{File} \longrightarrow \texttt{File}$.

Les opérations `inserer` et `supprimer` se font en $\mathcal{O}(\log_2 n)$, où n désigne le nombre d'éléments dans la file de priorité.

Rappel : type abstrait de données `File de priorité`

Définition :

Soit C un ensemble quelconque et E un ensemble ordonné.
Une file de priorité est un ensemble de couples $(\texttt{cle},\texttt{priorite}) \in C \times E$, sur lequel on souhaite pouvoir réaliser les opérations suivantes :

- ~~`maximum : File` $\longrightarrow C \times E$~~ ou `minimum : File` $\longrightarrow C \times E$
- `inserer : File` $\times C \times E \longrightarrow$ `File` ;
- `supprimer : File` $\longrightarrow$ `File`.

Les opérations `inserer` et `supprimer` se font en $\mathcal{O}(\log_2 n)$, où n désigne le nombre d'éléments dans la file de priorité.

Construction de l'arbre pondéré de cout minimum

A ce stade, on suppose connu :

1. le nombre n de lettres contenu dans le texte à compresser ;
2. le nombre d'occurence `nb_occ(l)` de chaque lettre `l` du texte.

Algorithme de création de l'arbre de Huffman :

- Créer une file F de priorité vide.
- Pour chaque lettre l du texte :

```
a = alloueNoeud(l)
inserer(F, a, nb_occ(l))
```

- Pour `i` allant de 1 à `n−1` :

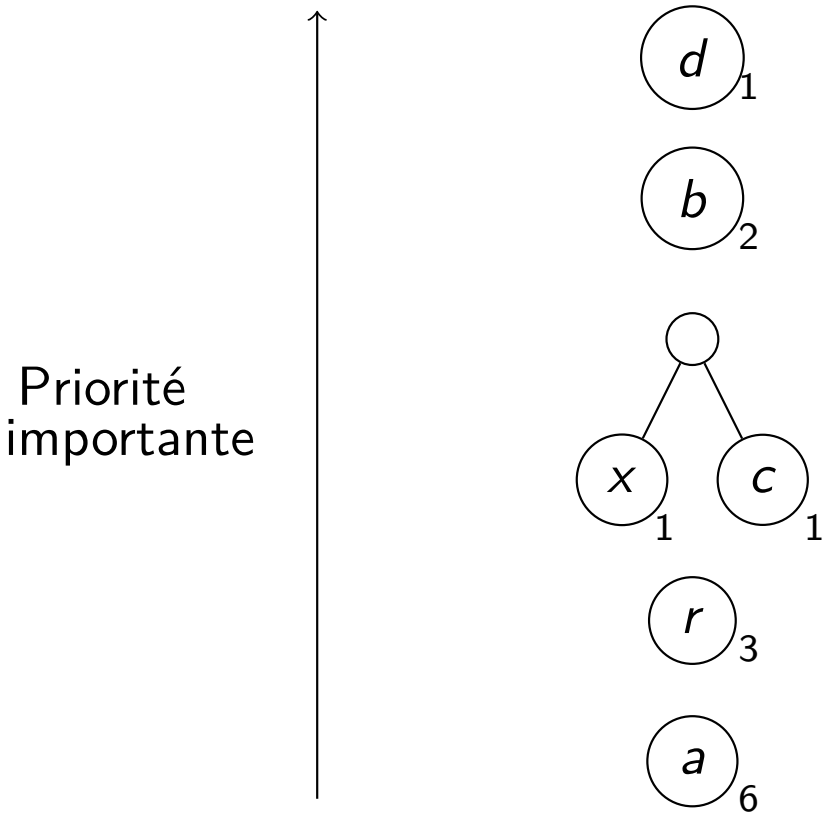
```
b = alloueNoeud()
cle_1, priorite_1 = minimum(F)
cle_2, priorite_2 = minimum(F)
b->fils_gauche = cle_1
b->fils_droit = cle_2
inserer(F, b, priorite_1 + priorite_2)
```

Exemple

Texte à compresser : "xabracadabrara"

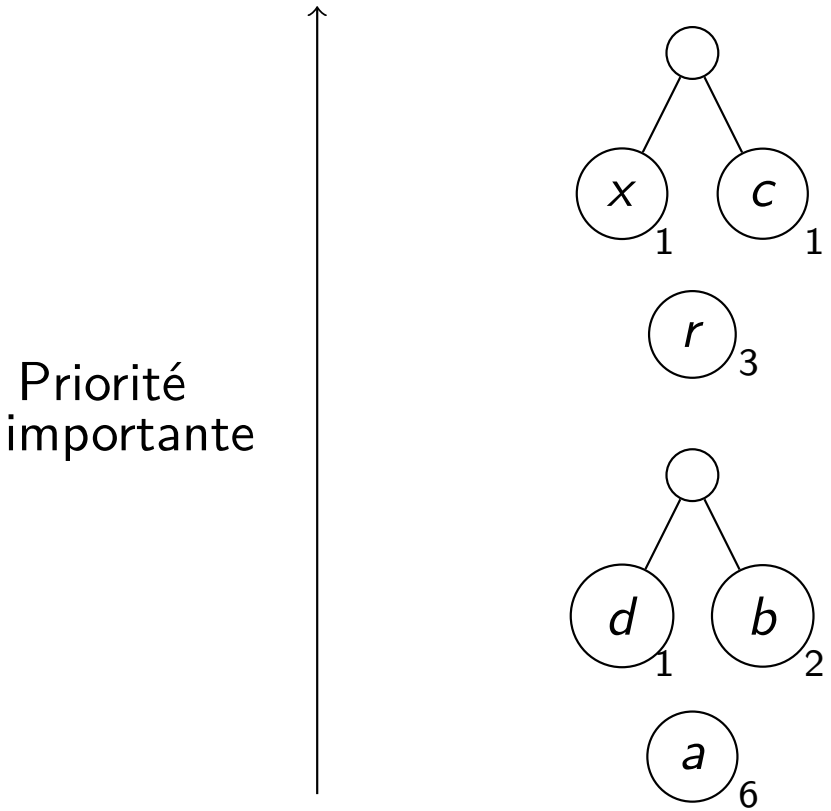
Exemple

Texte à compresser : "xabracadabrara"



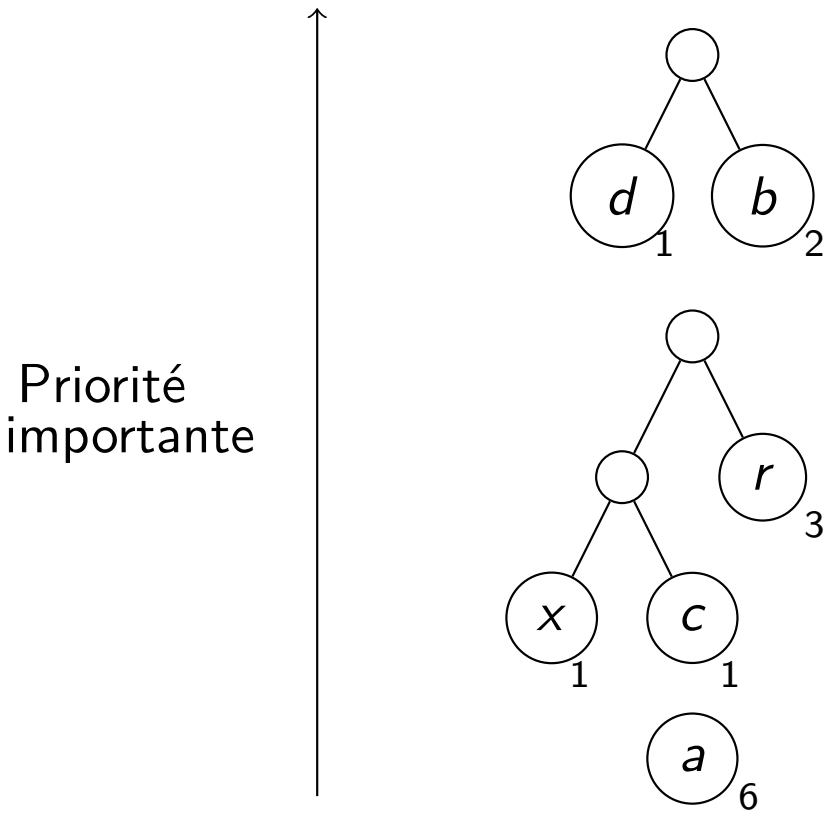
Exemple

Texte à compresser : "xabracadabrara"



Exemple

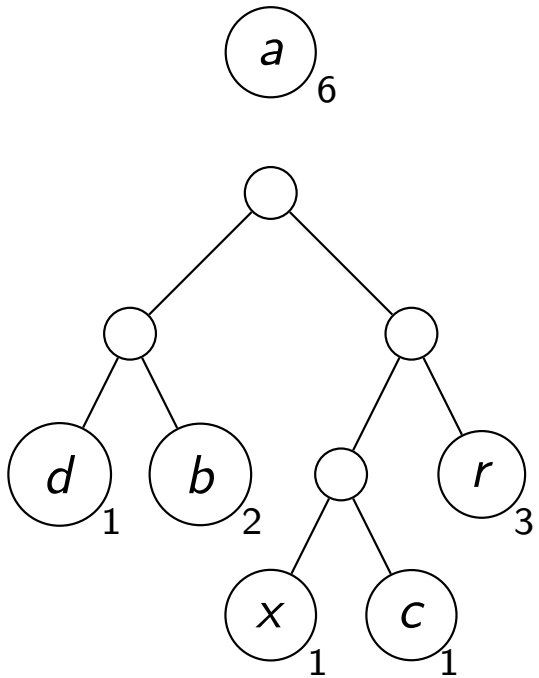
Texte à compresser : "xabracadabrara"



Exemple

Texte à compresser : "xabracadabrara"

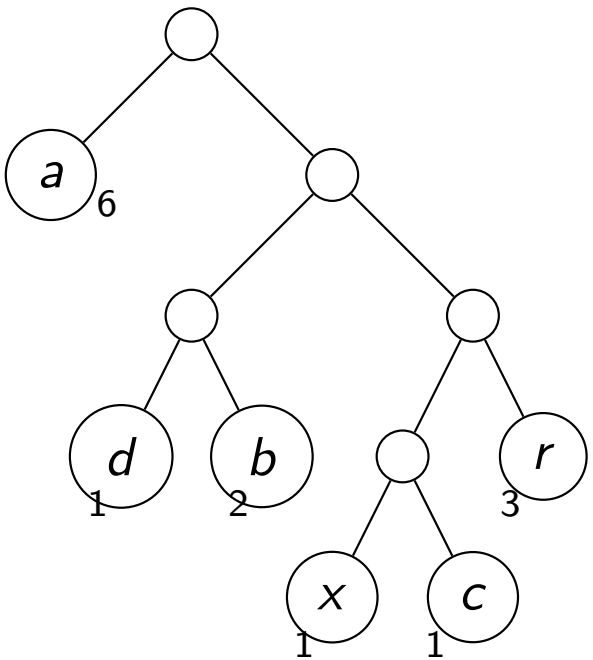
Priorité importante



Exemple

Texte à compresser : "xabracadabrara"

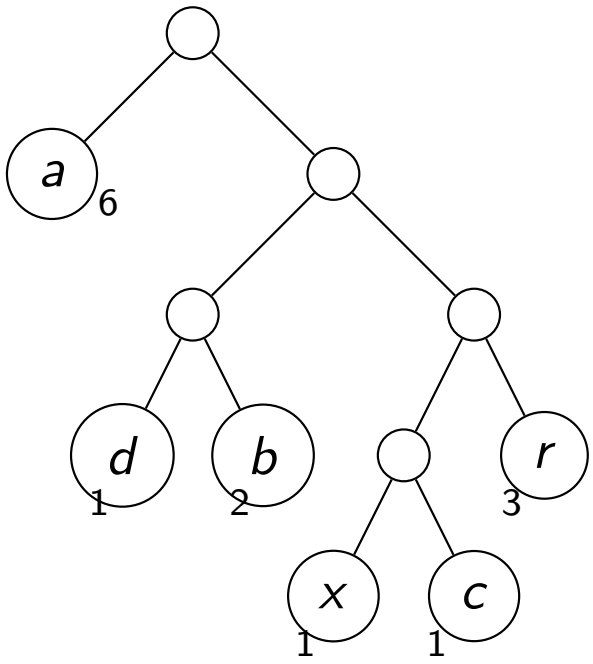
Priorité importante



Exemple

Texte à compresser : "xabracadabrara"

Priorité importante



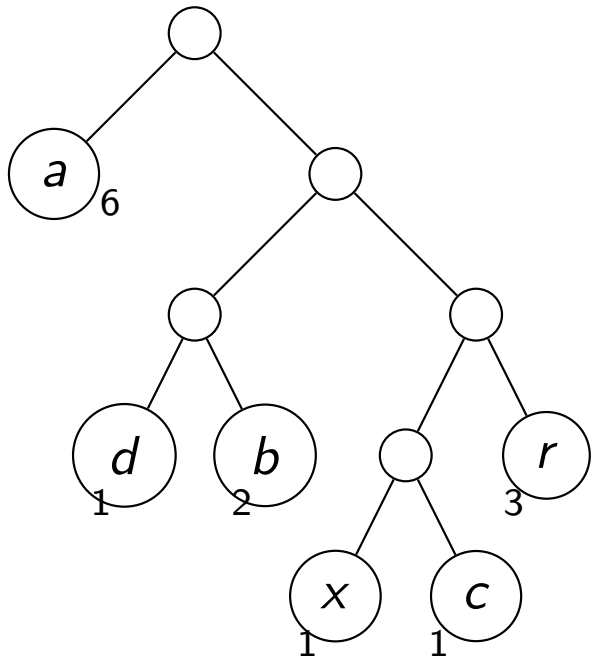
Codage :

a :	0	d :	100
b :	101	r :	111
c :	1101	x :	1100

Exemple

Texte à compresser : "xabracadabrara"

Priorité importante



Codage :

a :	0	d :	100
b :	101	r :	111
c :	1101	x :	1100

Compressé :

1100|0|101|111|0|1101|0|100|0|101|111|0|1

32 bits contre 14 · 8 = 112 bits

Exercice

Imaginons un texte de 1000 caractères dont le nombre d'occurences des lettres est donné par :

Lettre	a	b	c	d	e	f
Nombre d'occurence	350	330	20	160	90	50

Donner un arbre pondéré de cout minimal de Huffman pour ce texte et comparer avec le codage 2 du transparent 5.

Outline

Principe du décodage

Pour décoder un message codé par l'algorithme de Huffman, il faut avoir accès

- au codage de chaque lettre

ou

- à l'arbre de Huffman

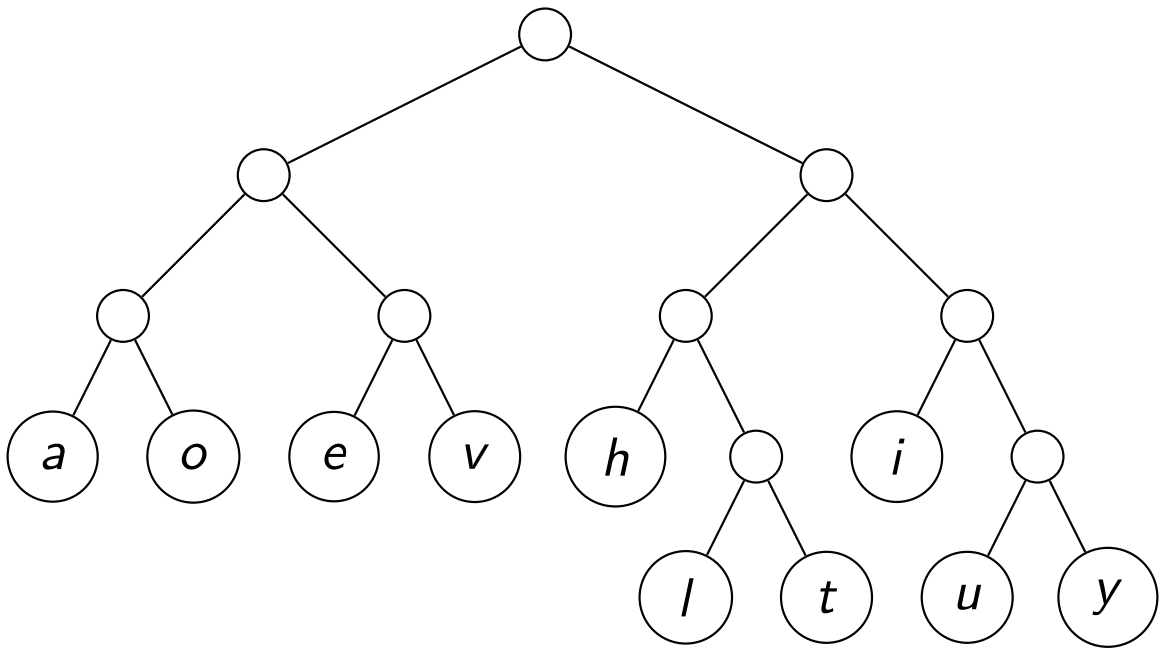
Puisque le code de Huffman est un code préfixe, on peut déconcaténer le messsage de manière unique !

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :

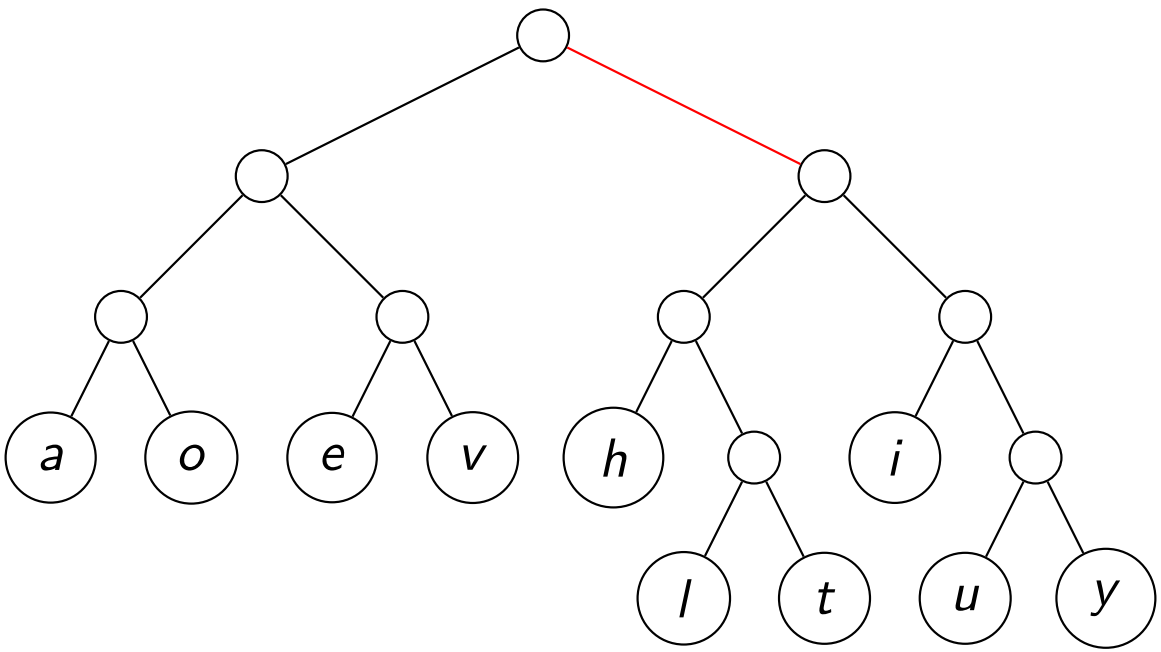


Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



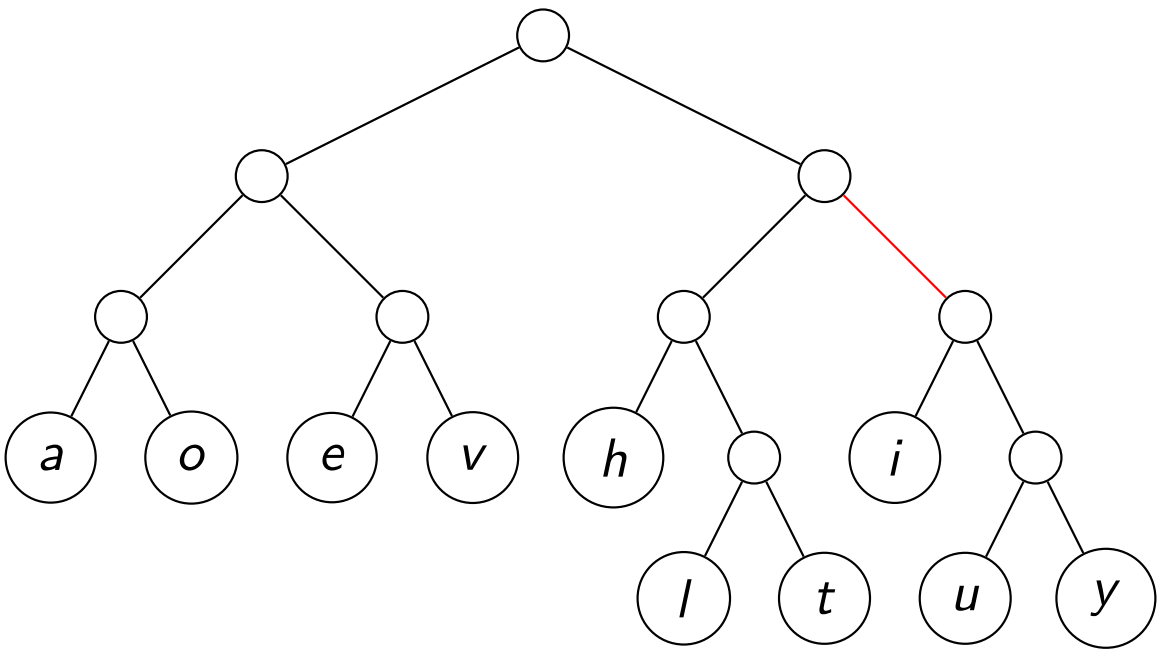
Texte décompréssé :

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



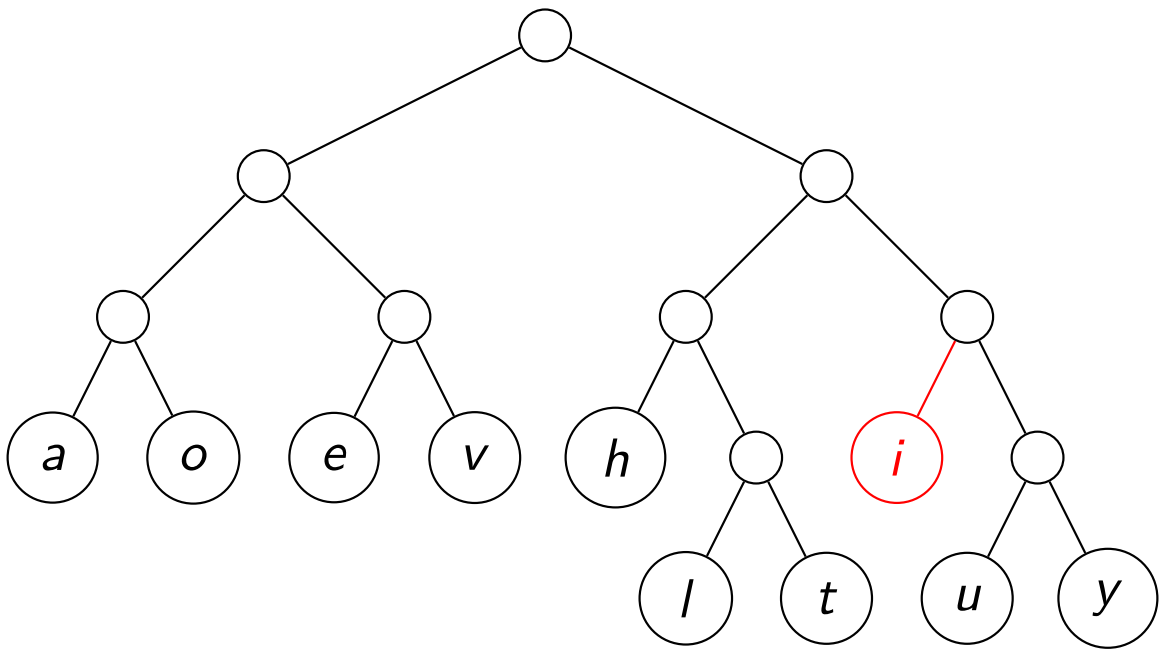
Texte décompréssé :

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



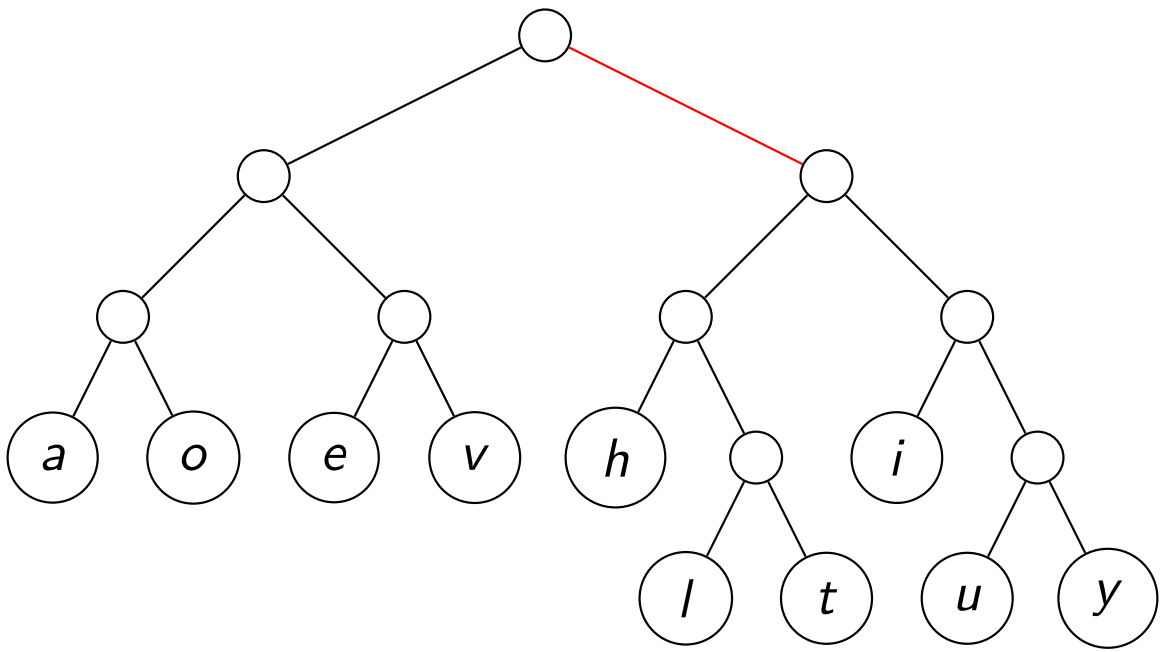
Texte décompréssé : i

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



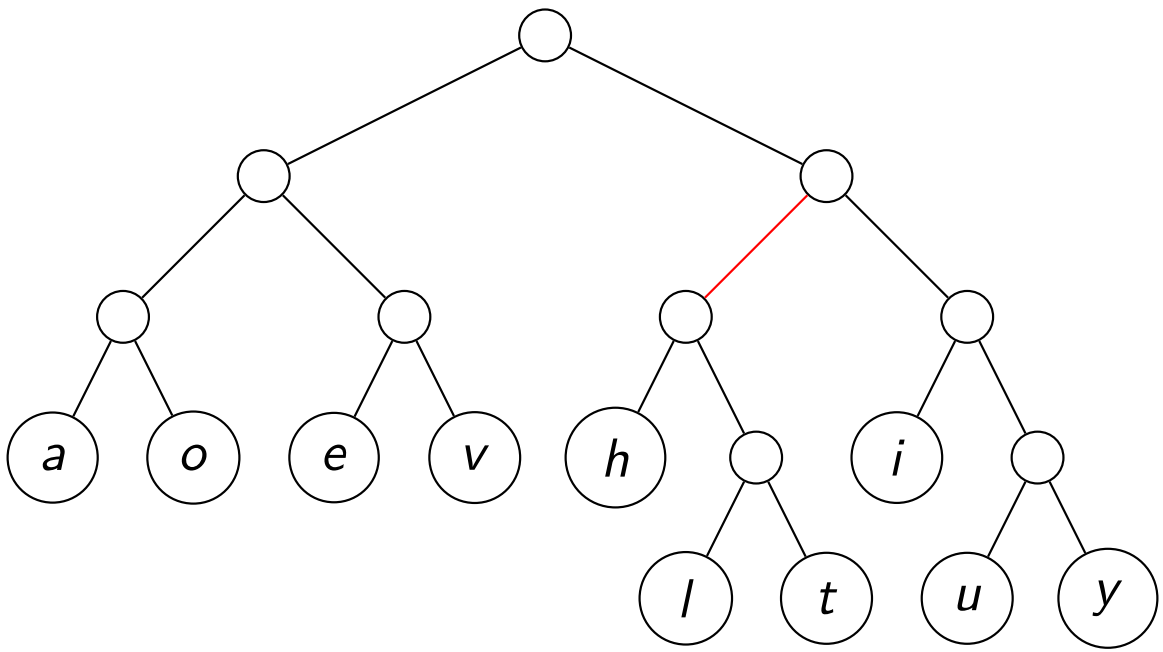
Texte décompréssé : i

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



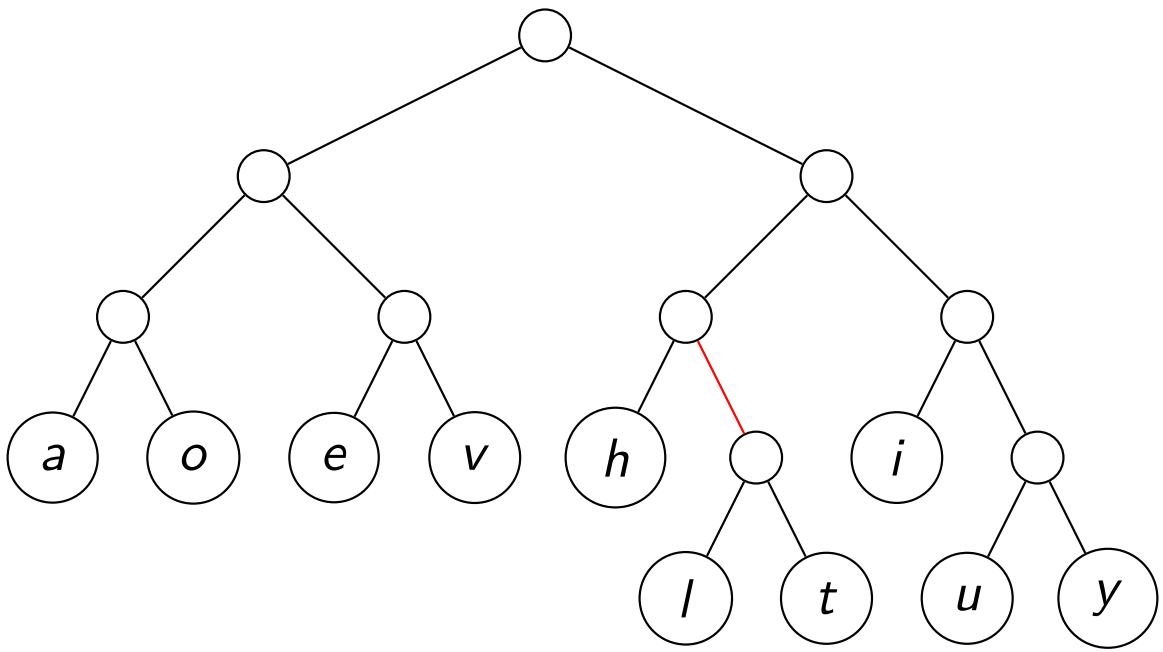
Texte décompréssé : i

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



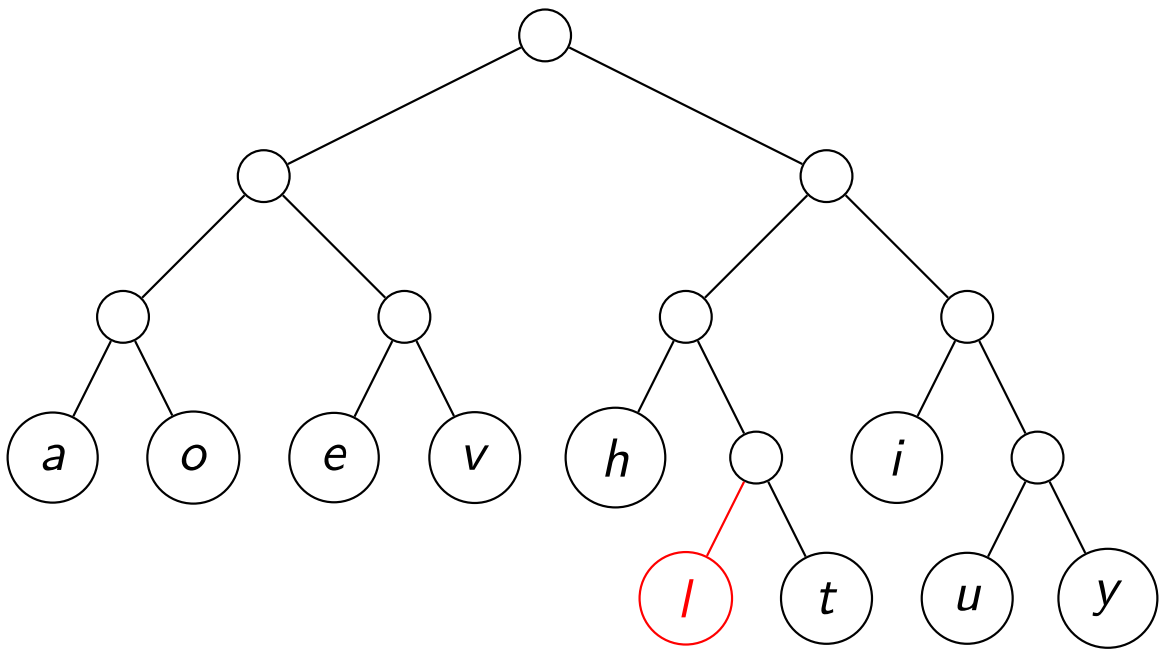
Texte décompréssé : i

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



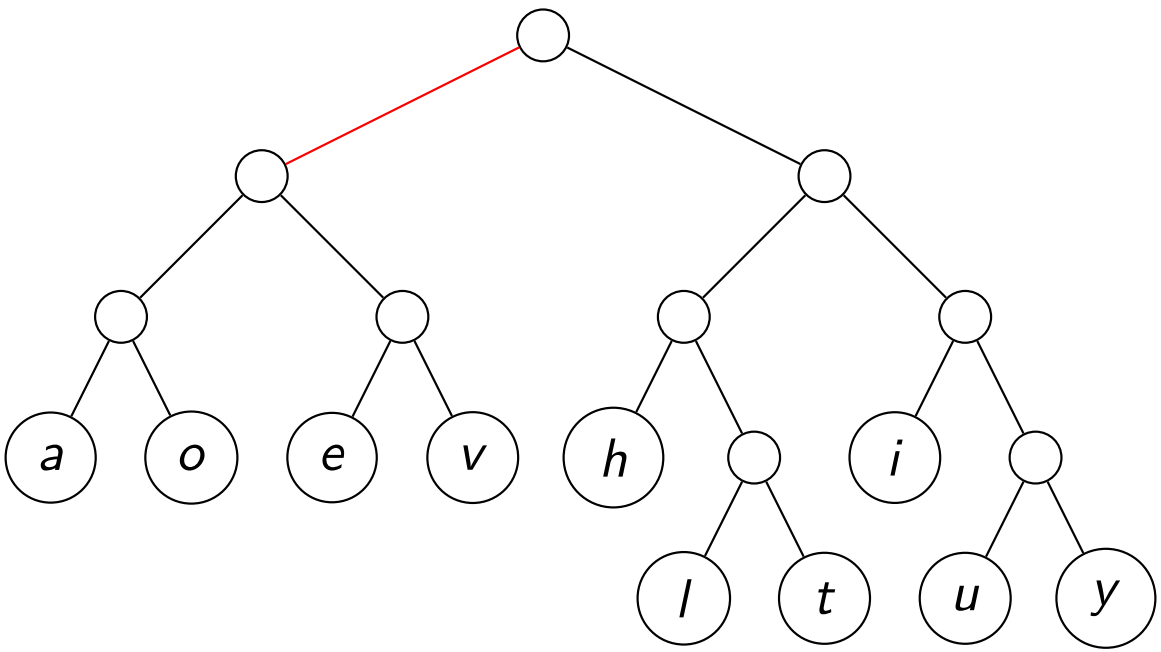
Texte décompréssé : i l

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



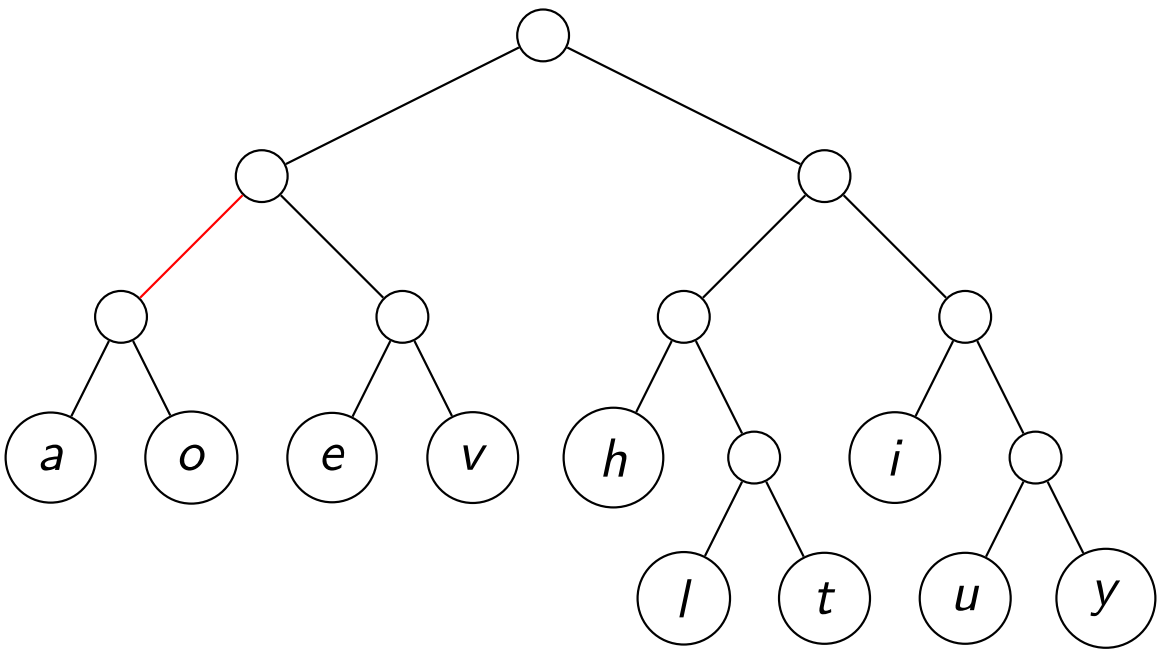
Texte décompréssé : i l

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



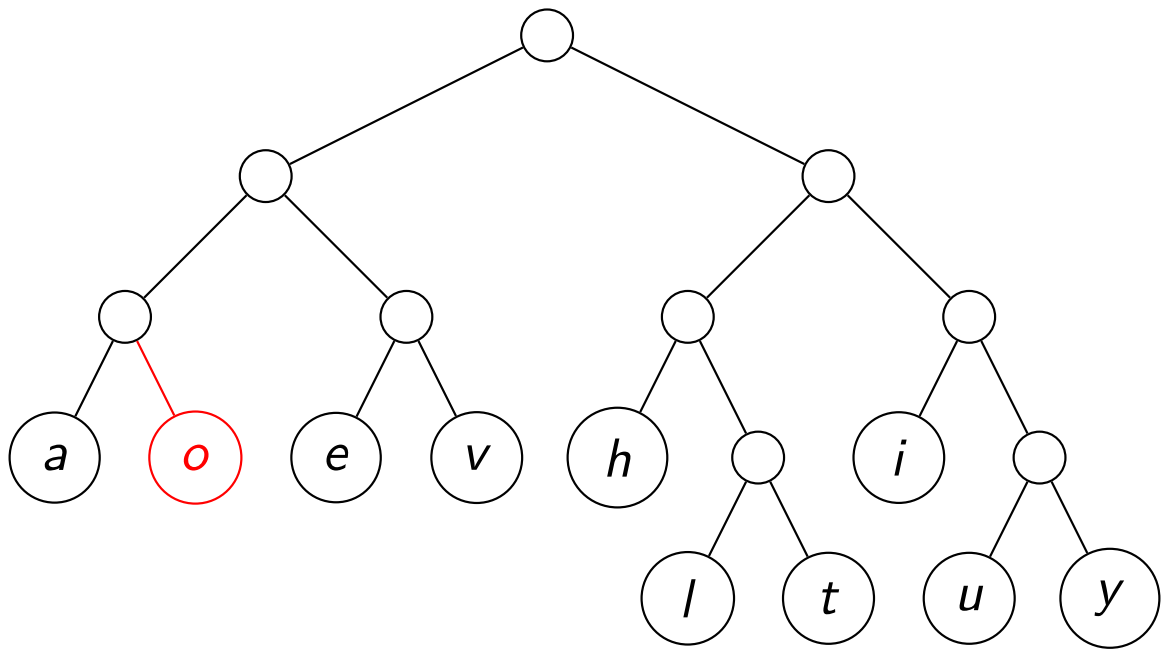
Texte décompréssé : i l

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



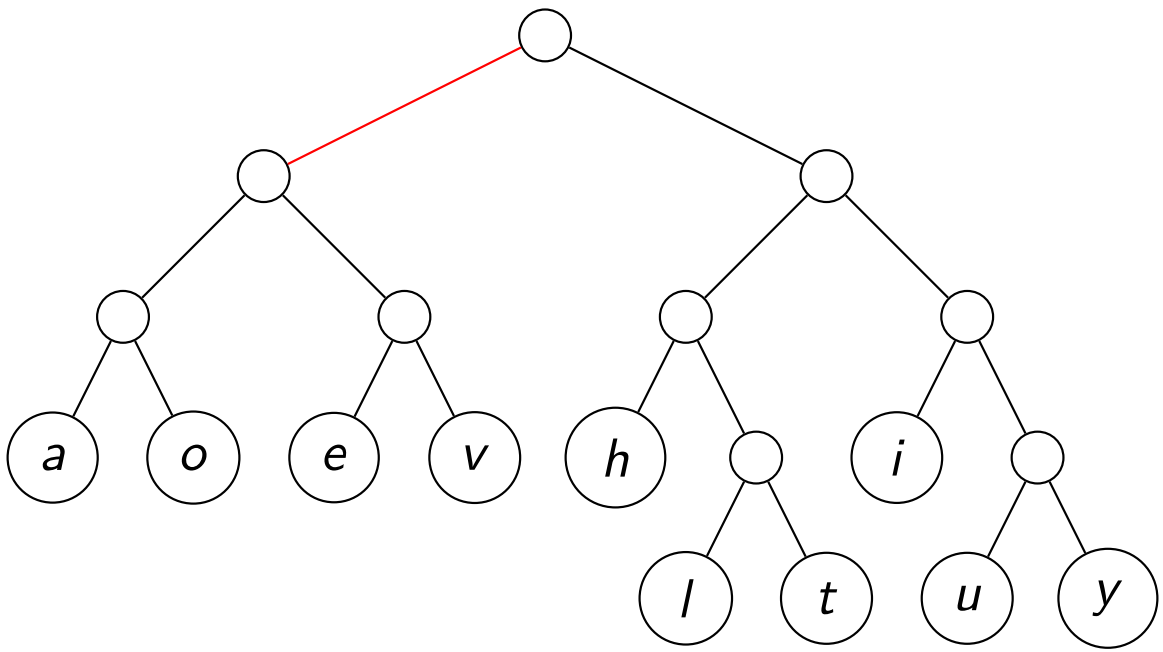
Texte décompréssé : i l o

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



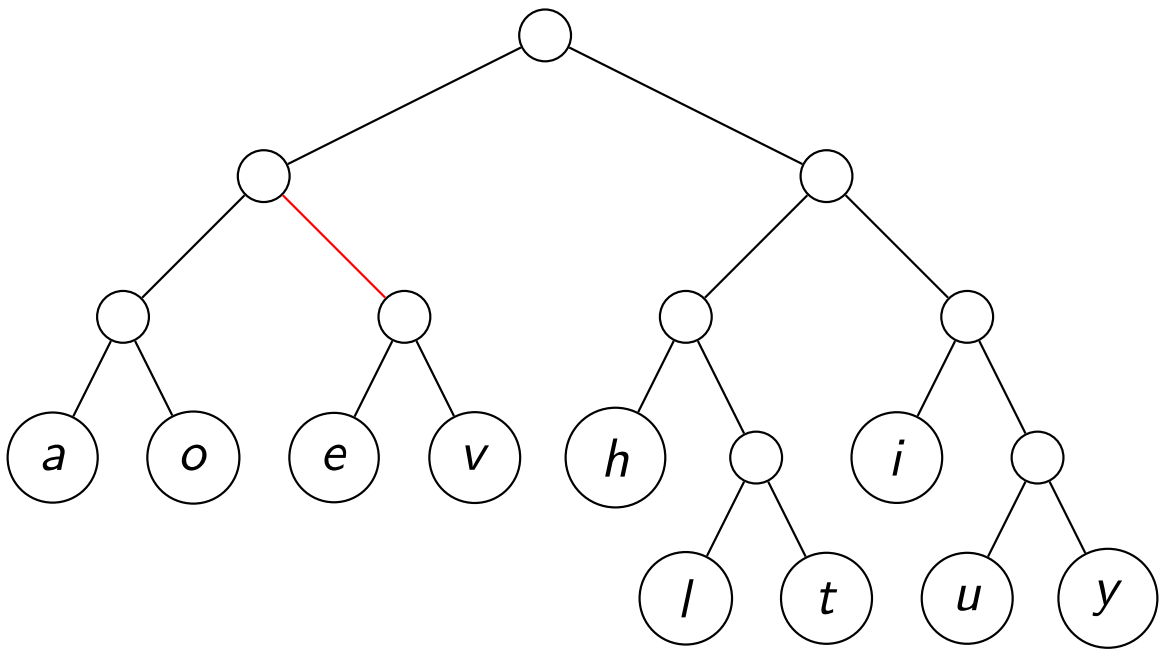
Texte décompréssé : i l o

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



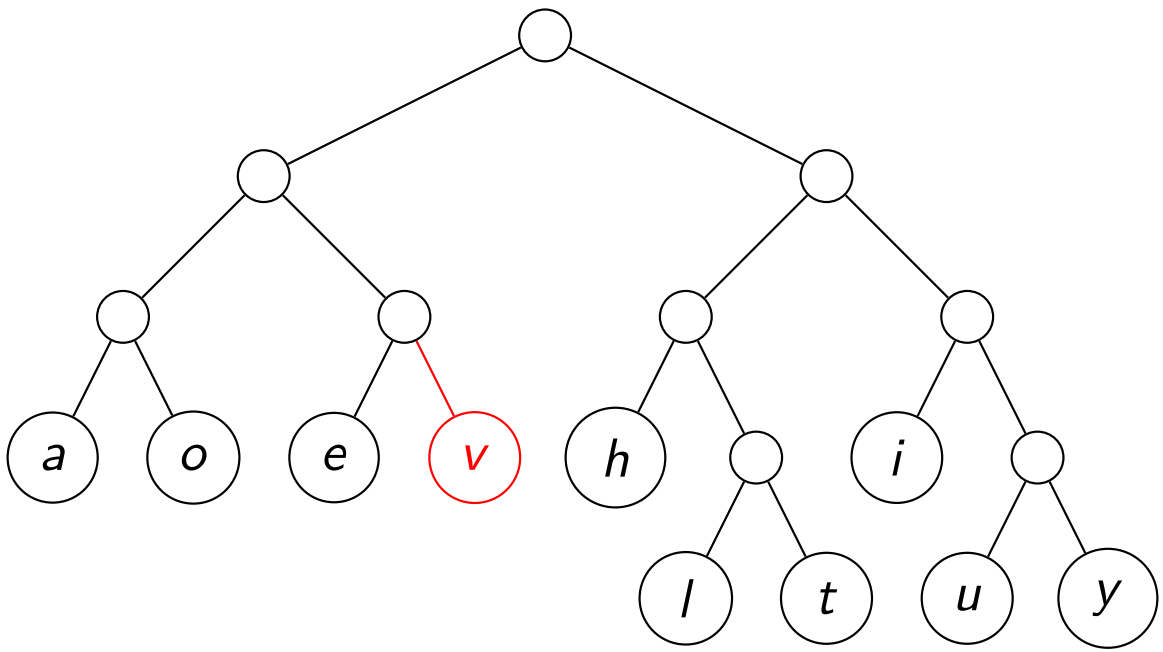
Texte décompréssé : i l o

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



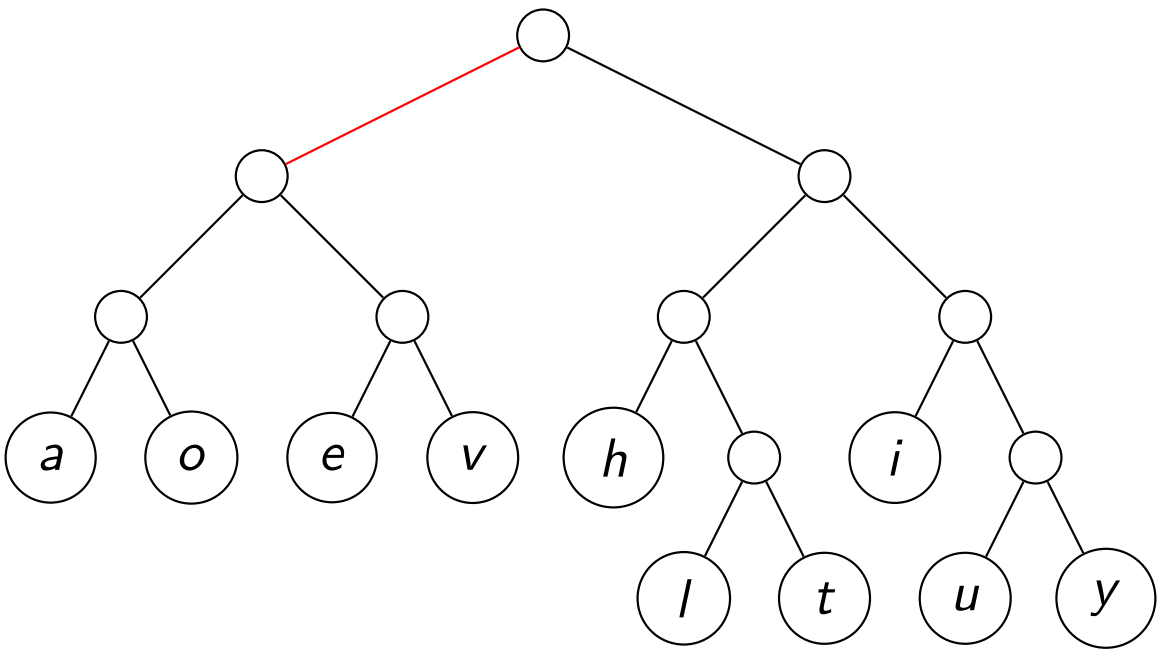
Texte décompréssé : i l o v

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



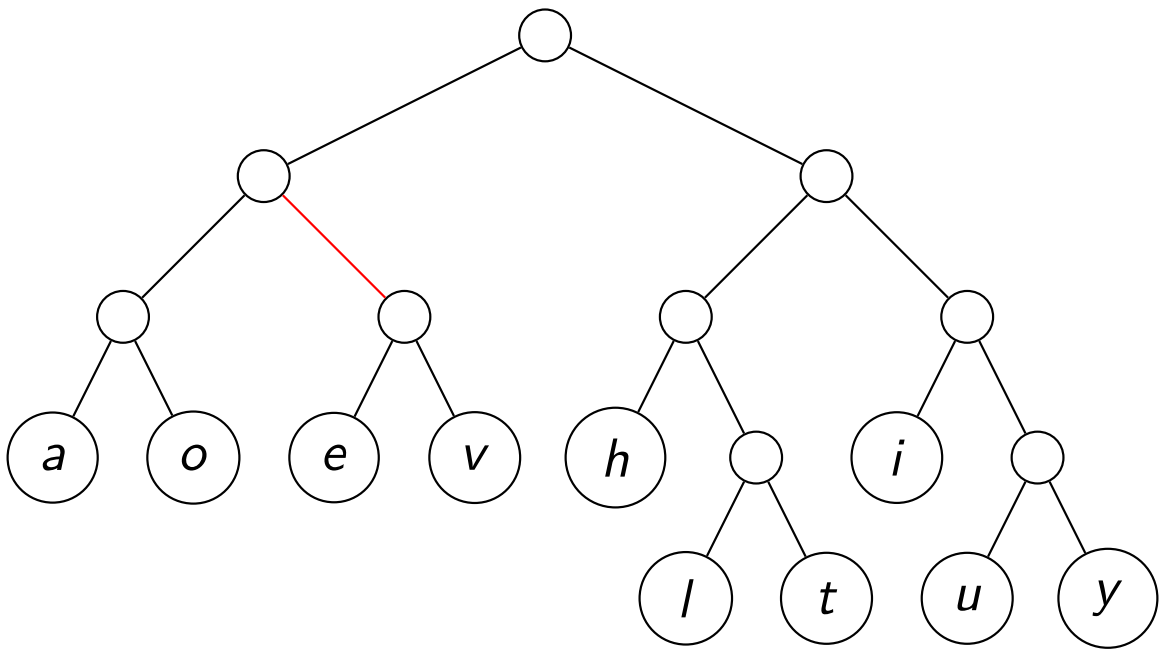
Texte décompréssé : i l o v

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



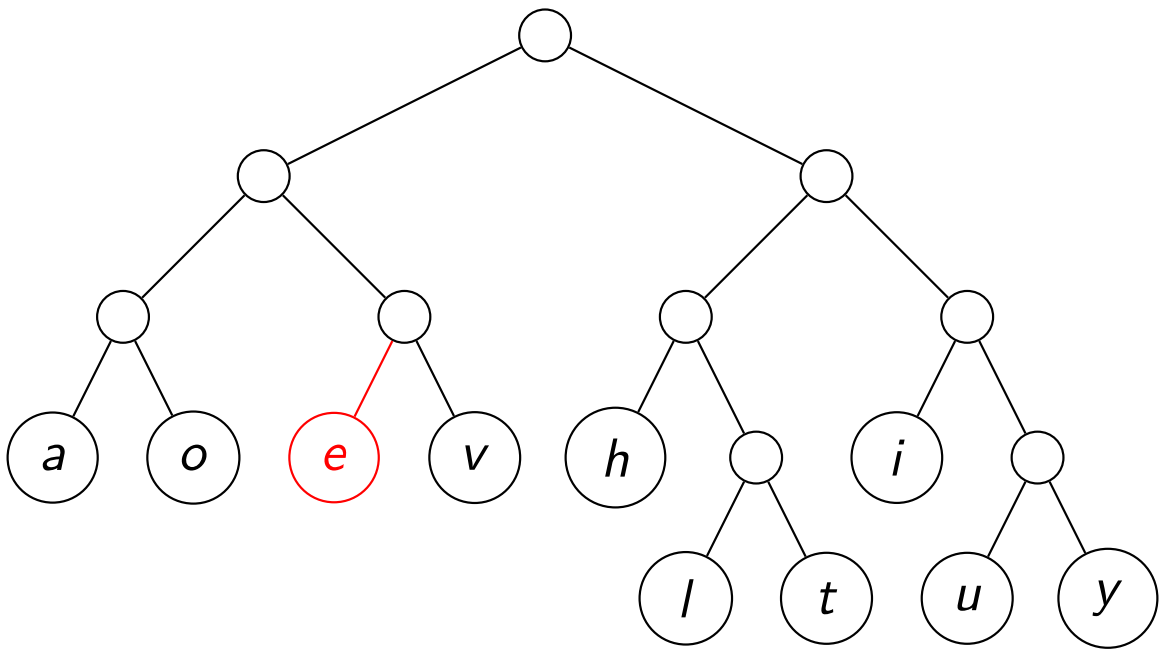
Texte décompréssé : i l o v

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



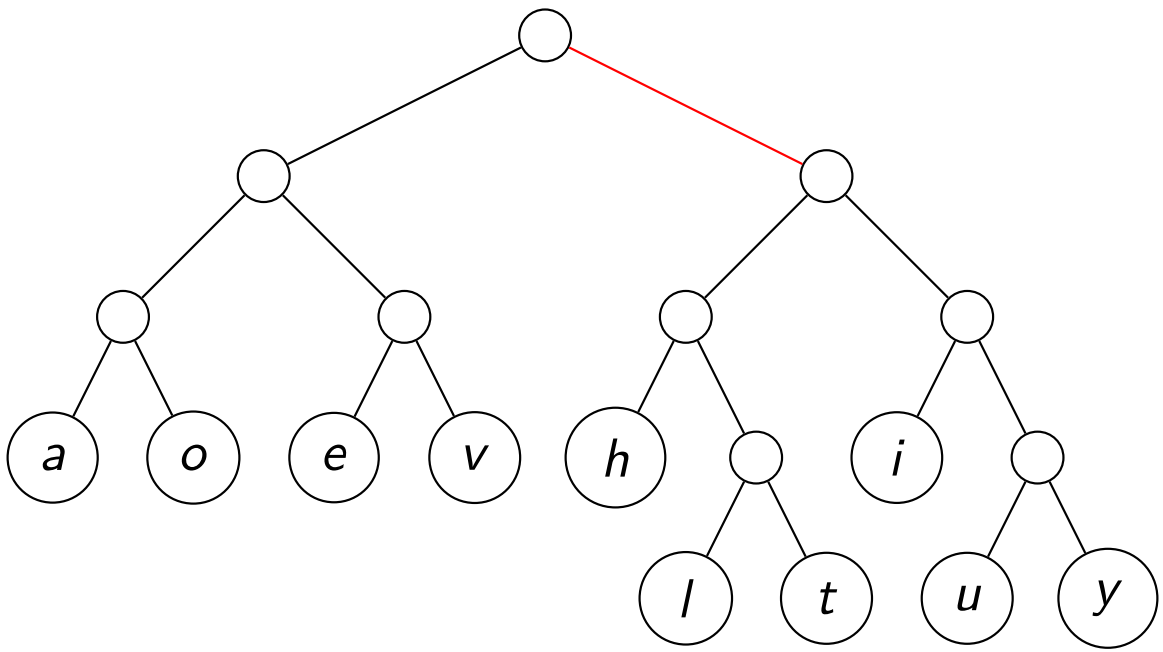
Texte décompréssé : i l o v e

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



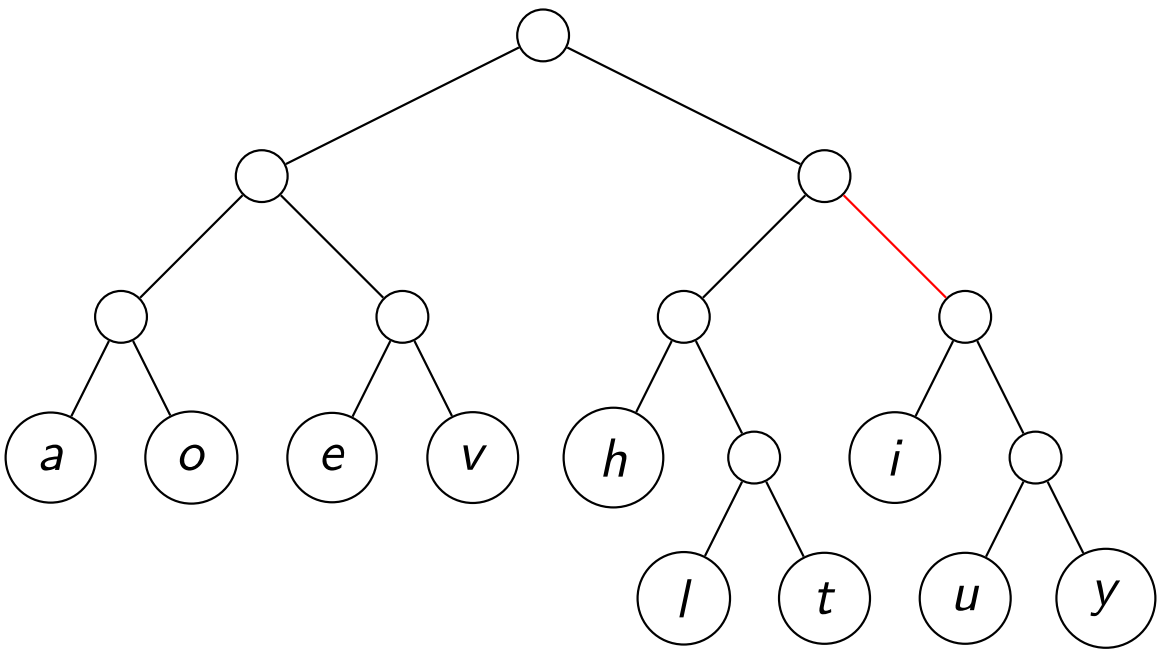
Texte décompréssé : i l o v e

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



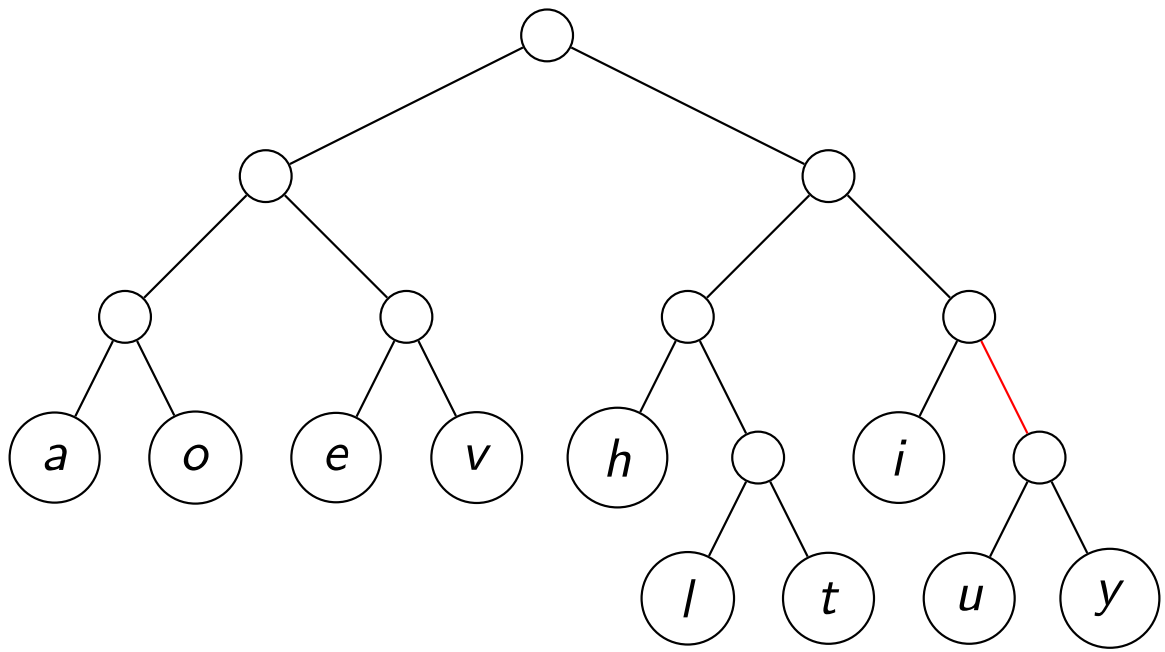
Texte décompréssé : i l o v e

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



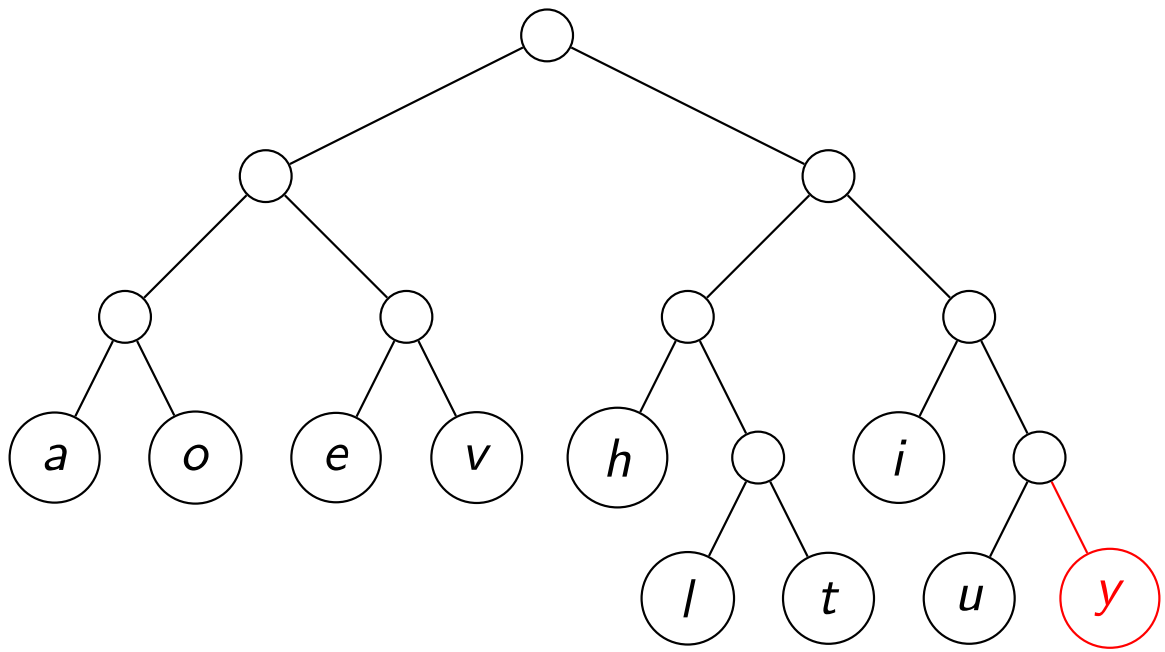
Texte décompréssé : i l o v e

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



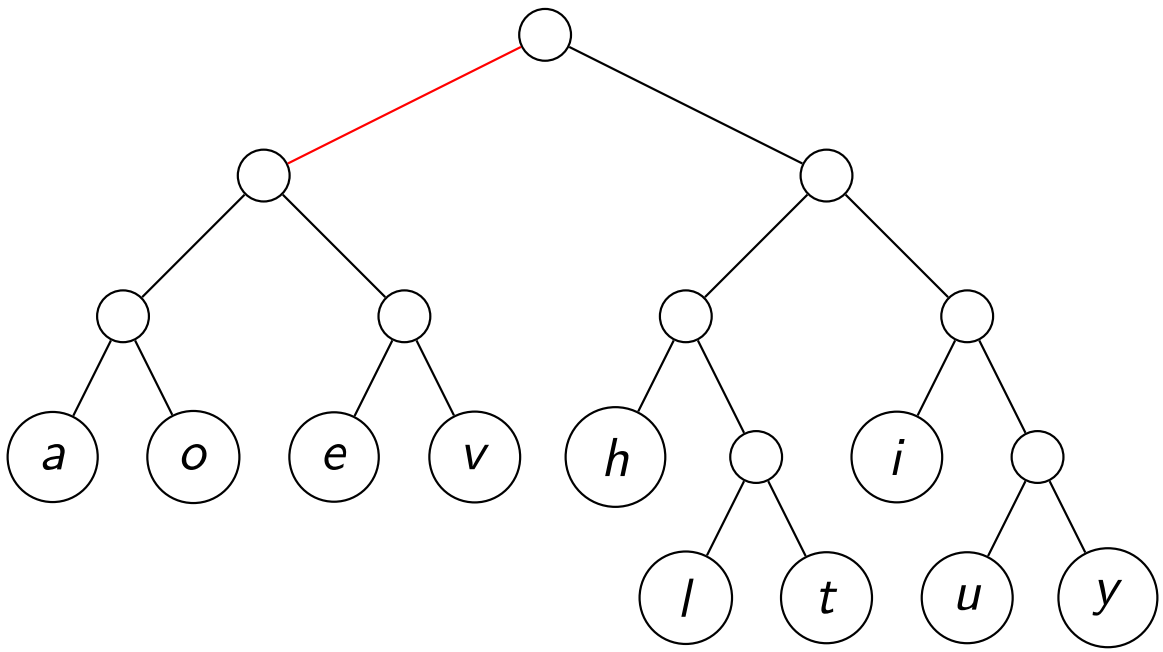
Texte décompréssé : i l o v e y

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



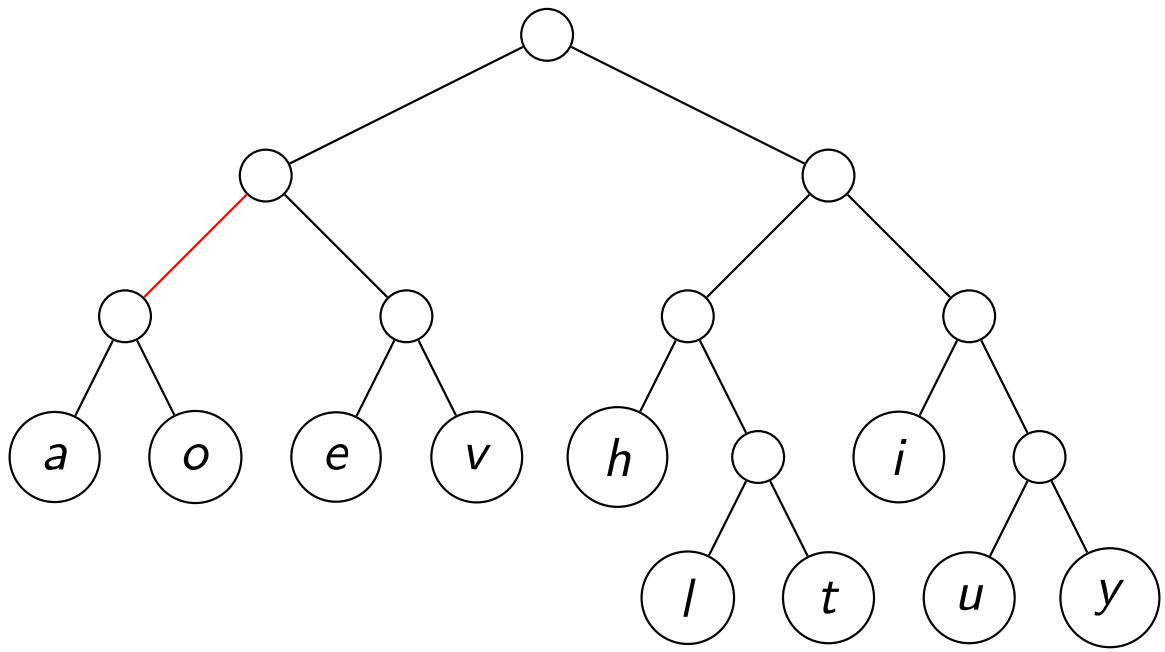
Texte décompréssé : i l o v e y

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



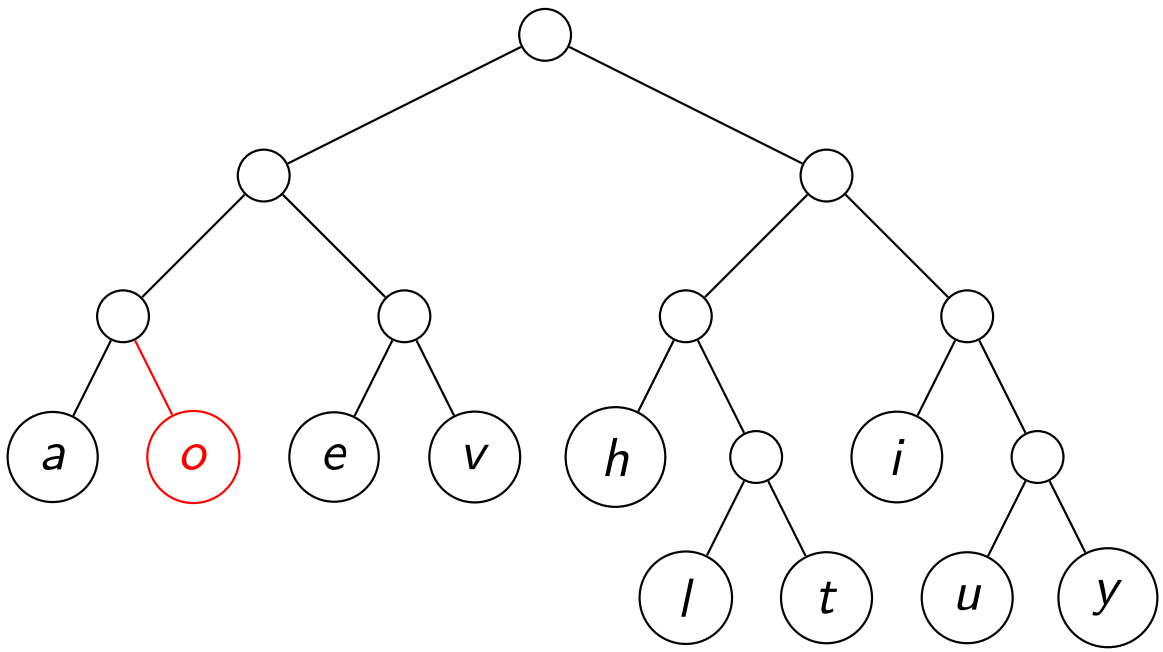
Texte décompréssé : i l o v e y

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



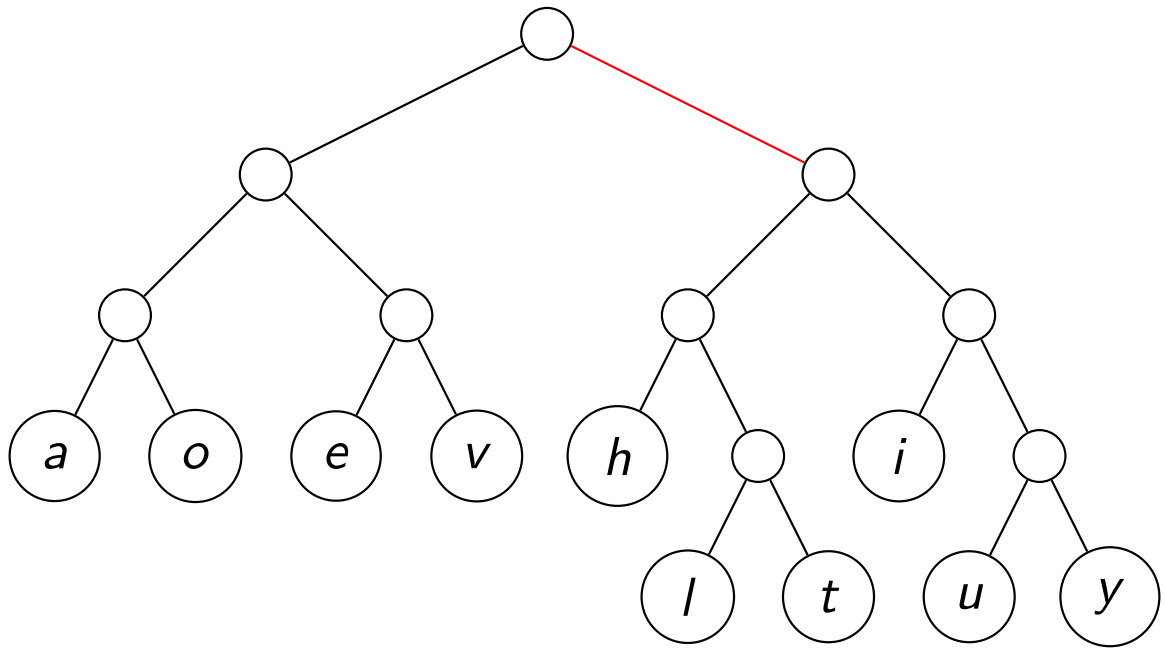
Texte décompréssé : i l o v e y o

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



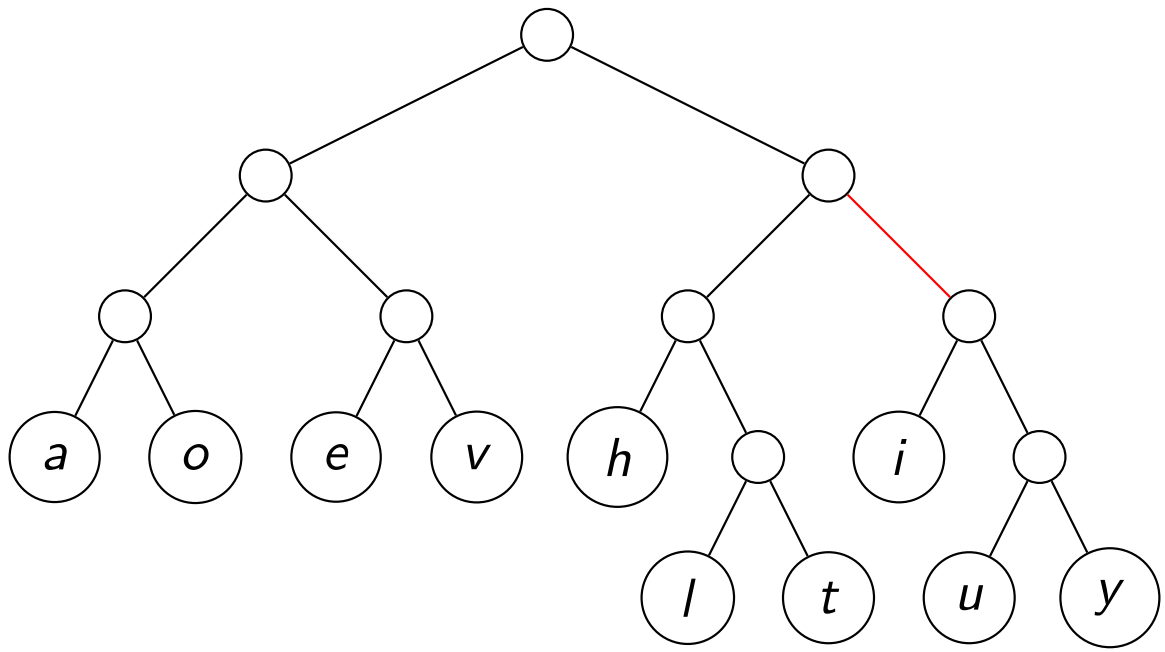
Texte décompréssé : i l o v e y o

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



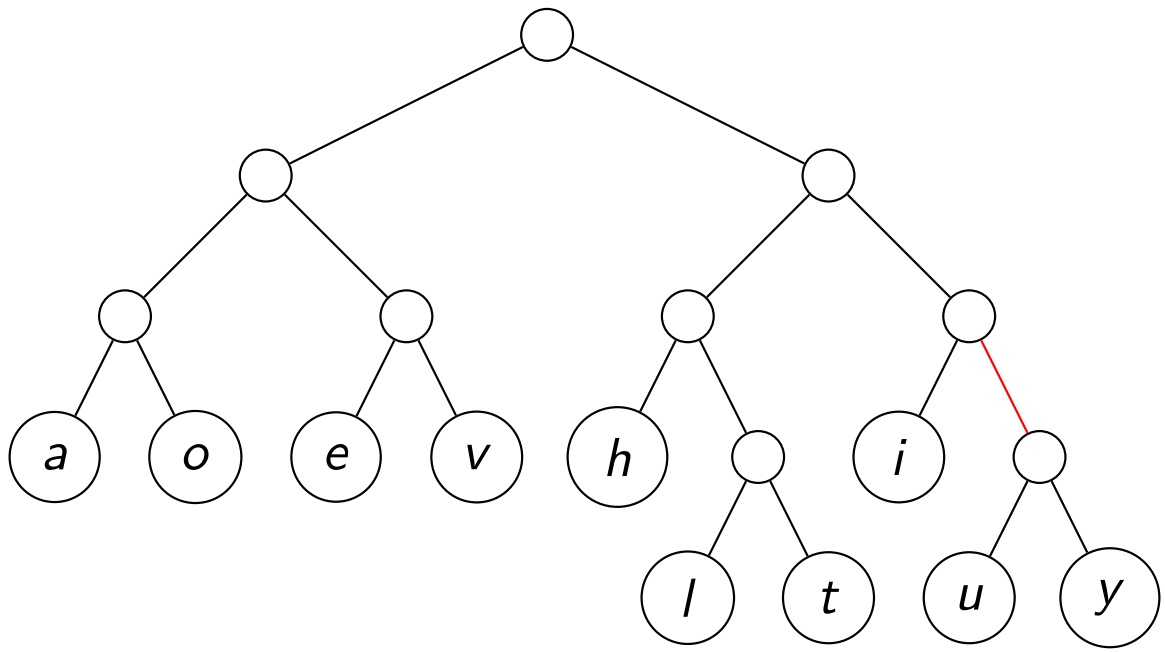
Texte décompréssé : i l o v e y o

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



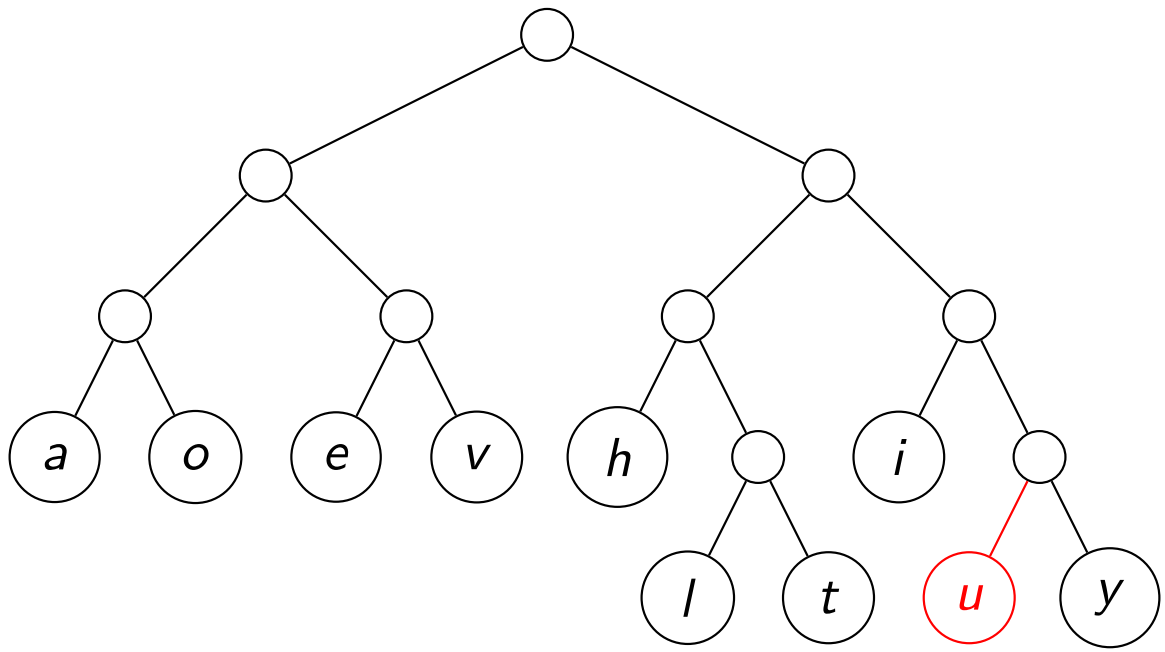
Texte décompréssé : i l o v e y o

Exemple

Décodons le texte suivant :

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

sachant qu'il a été codé en utilisant l'arbre de Huffman suivant :



Texte décompréssé : i l o v e y o u

Outline

Structure utilisée pour stocker l'arbre de Huffman

Fait : Si le nombre de feuilles est N, l'arbre de Huffman comprend $2N - 1$ nœuds.

On représente donc l'arbre de Huffman par un tableau de $2N - 1$ cases : les nœuds sont chainés par indice.

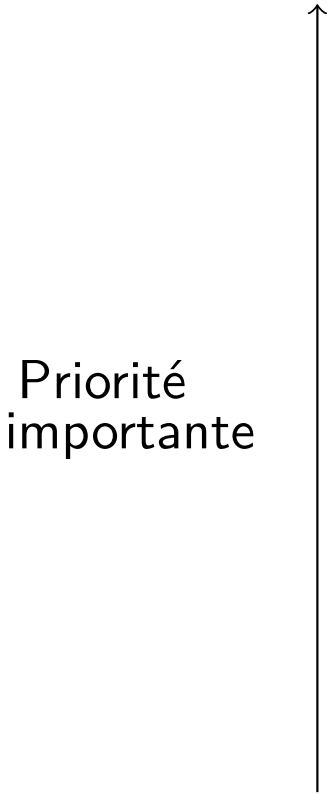
```
typedef struct {
    char lettre;
    int occurence;
    int gauche,droit;
} NoeudHuffman;
```

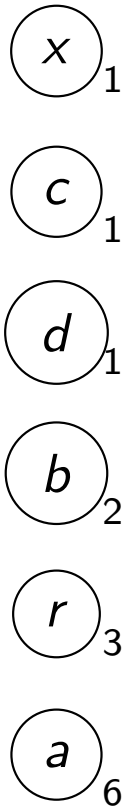
Construction de l'arbre

- Les premier nœuds de l'arbre contiennent les lettres et sont triées par ordre croissant
- En remarquant que les nœuds internes sont crées par poids croissant, le tableau est utilisé comme deux files de priorité :
  - $\rightsquigarrow$ la première contient la suite triée des feuilles ;
  - $\rightsquigarrow$ la seconde contient celle des nœuds internes.
- Les deux nœuds de poids minimum sont donc parmi les deux premières feuilles ou les deux premiers nœuds internes non traitées.

Exemple

Texte à compresser : "xabracadabrara"



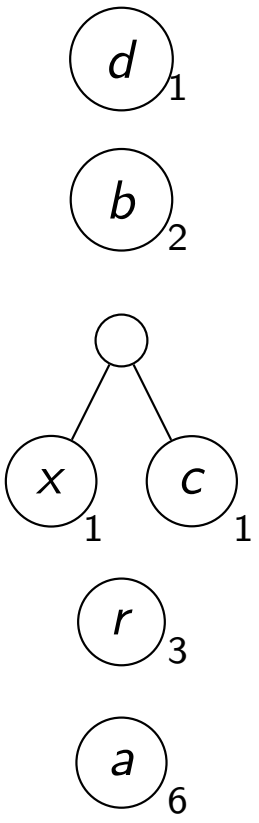


	lettre	Poids	G	D
→ 0	x	1	-	-
1	c	1	-	-
2	d	1	-	-
3	b	2	-	-
4	r	3	-	-
5	a	6	-	-
→ 6				
7				
8				
9				
10				

Exemple

Texte à compresser : "xabracadabrara"

Priorité importante

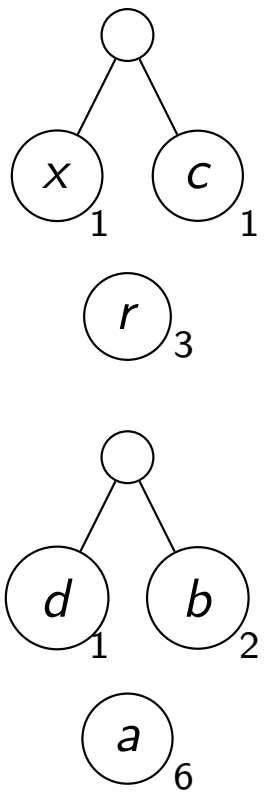


	lettre	Poids	G	D
0	x	1	-	-
1	c	1	-	-
→ 2	d	1	-	-
3	b	2	-	-
4	r	3	-	-
5	a	6	-	-
→ 6	-	2	0	1
7				
8				
9				
10				

Exemple

Texte à compresser : "xabracadabrara"

Priorité importante

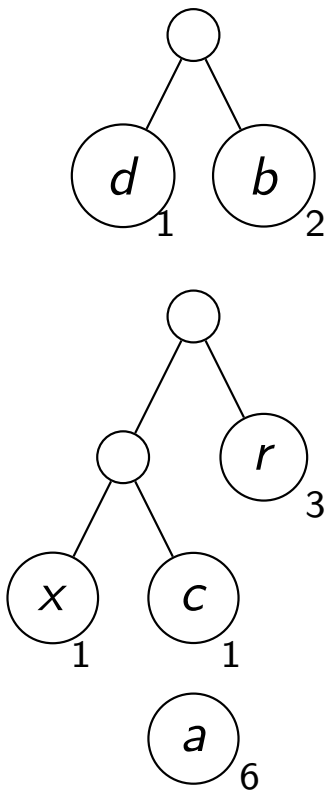


	lettre	Poids	G	D
0	x	1	-	-
1	c	1	-	-
2	d	1	-	-
3	b	2	-	-
→ 4	r	3	-	-
5	a	6	-	-
→ 6	-	2	0	1
7	-	3	2	3
8				
9				
10				

Exemple

Texte à compresser : "xabracadabrara"

Priorité importante

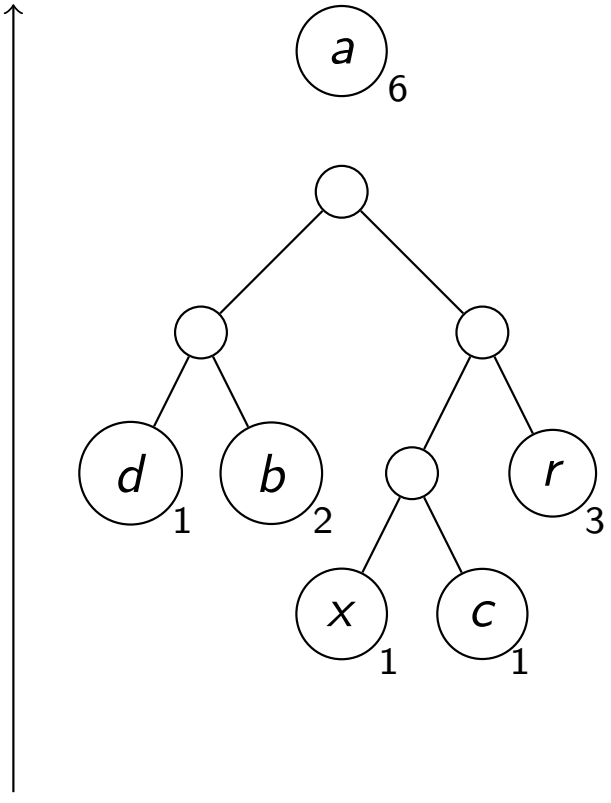


	lettre	Poids	G	D
0	x	1	-	-
1	c	1	-	-
2	d	1	-	-
3	b	2	-	-
4	r	3	-	-
→ 5	a	6	-	-
6	-	2	0	1
→ 7	-	3	2	3
8	-	5	6	4
9				
10				

Exemple

Texte à compresser : "xabracadabrara"

Priorité importante

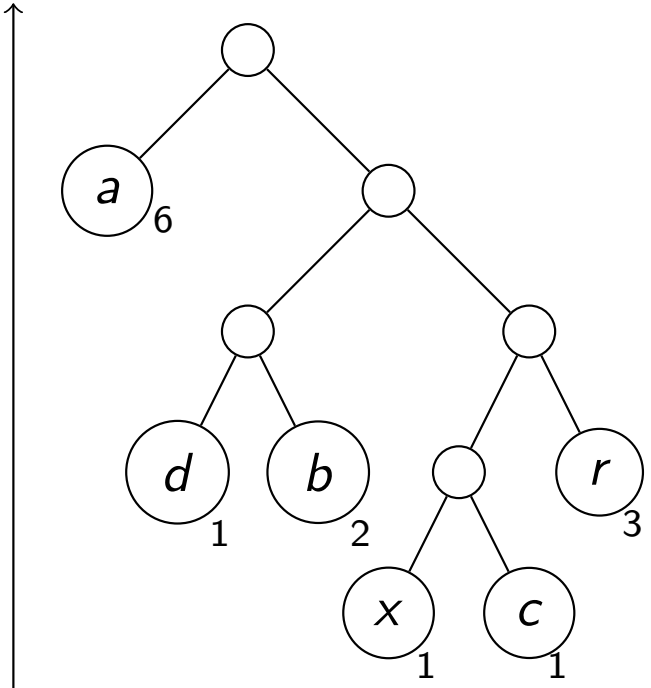


	lettre	Poids	G	D
0	x	1	-	-
1	c	1	-	-
2	d	1	-	-
3	b	2	-	-
4	r	3	-	-
→ 5	a	6	-	-
6	-	2	0	1
7	-	3	2	3
8	-	5	6	4
→ 9	-	8	7	8
10				

Exemple

Texte à compresser : "xabracadabrara"

Priorité importante



	lettre	Poids	G	D
0	x	1	-	-
1	c	1	-	-
2	d	1	-	-
3	b	2	-	-
4	r	3	-	-
5	a	6	-	-
6	-	2	0	1
7	-	3	2	3
8	-	5	6	4
9	-	8	7	8
10	-	14	5	9

Deux difficultés :

Deux problèmes se posent pour permettre le décodage :

- il faut transmettre le code utilisé ;
- si la longueur du compressé n'est pas multiple de 8, il ne faut pas prendre en compte tous les bits du dernier octet.

Transmission du code

- Un parcours préfixe de l'arbre de Huffman permet d'associer les mots du code et les lettres du texte :
  - ⇝ un bit 0 indique un nœud interne ;
  - ⇝ un bit 1 indique une feuille, que l'on fait suivre du code ASCII de la lettre associée.

- **Structure pour stocker un mot du code :**

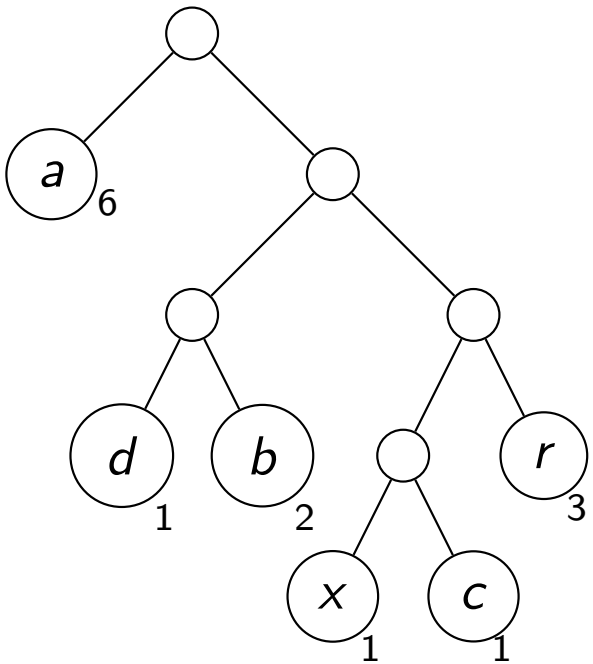
```
typedef struct {
    char *code;
    int nombreBit;
} Codage;
```

- **Structure pour stocker intégralement le code :**

```
#define NB_LETTRES 256
...
Codage Code[NB_LETTRES]
```

Exemple de transmission de code

Un arbre de Huffman :



Transmission du code associée :

`01[a]001[d]1[b]001[x]1[c]1[r]`

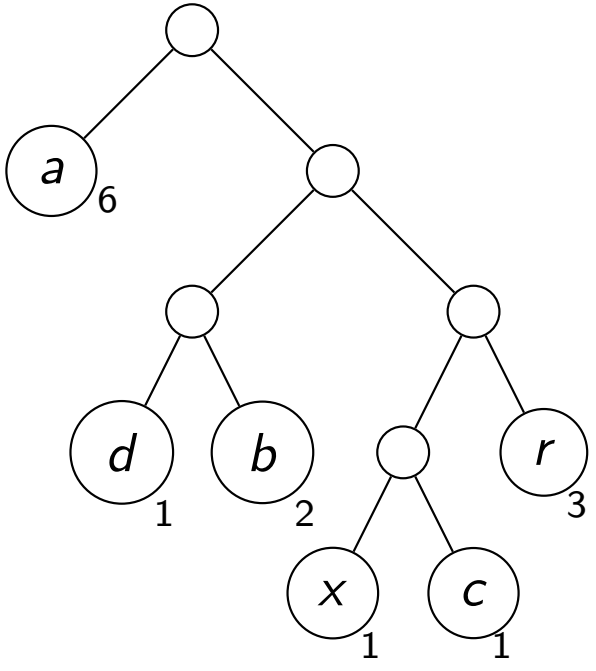
`[x]` désigne le code ascii de la lettre `x`.

Padding

Warning : Il peut y avoir des bits non valides dans dernier octet du fichier compressé.

- La longueur du compressé est égale au coût de l'arbre.
- Le codage de l'arbre utilise un bit par nœud.
- Le fichier compressé débute par 3 bits indiquant le nombre de bits invalides du dernier octet (de 0 à 7).

Exemple : *xabracadabrara*



Codage :

a : 0
b : 101
c : 1101
d : 100
r : 111
x : 1100

Calcul du padding :

compressé : 32 bits $\equiv 0[8]$
nombre de nœuds : 11 $\equiv 3[8]$
padding $= 8 - 0 - 3 - 3 = 2$

Entête du fichier :

010 - transmission du code

Fichier compressé :

`01001[a]001[d]1[b]001[x]1[c]1[r]`
`11000101111011010100010111011100`

Décompression

1. On lit le padding ;
2. On reconstruit l'arbre à partir de son parcours préfixe ;
3. On décode le texte initial :
   - on suit le chemin à partir de la racine ;
   - Lorsqu'on arrive à une feuille, on décode la lettre et on repart de la racine.
4. On oublie les bits de fin associé au padding.

Exercice : Sensibilité de l'algorithme d'Huffman aux erreurs

- On vient d'obtenir un code de Huffman en deux passes d'un texte initial :

  `1110001[A]1[O]01[E]1[V]001[H]01[L]1[T]01[I]01`
  `[U]1[Y]1101010001011010111110011110···0111011`

  Celui-ci contient le message : `iloveyou`

- On transmis ce fichier, mais une erreur a inversé le sixième bit du texte compressé :

  `1110001[A]1[O]01[E]1[V]001[H]01[L]1[T]01[I]01`
  `[U]1[Y]1101000001011010111110011110···0111011`

  Quel message contient désormais ce fichier ?

- *Pour les transmissions, on utilise des codes correcteurs d'erreurs.*

Outline

Inconvénients de l'algorithme de Huffman

- lire tout le texte avant de commencer la compression;
- transmettre le code utilisé.

Solutions :
• utiliser un code fixe;
• version adaptative de Huffman qui modifie le code au fur et à mesure de son utilisation.

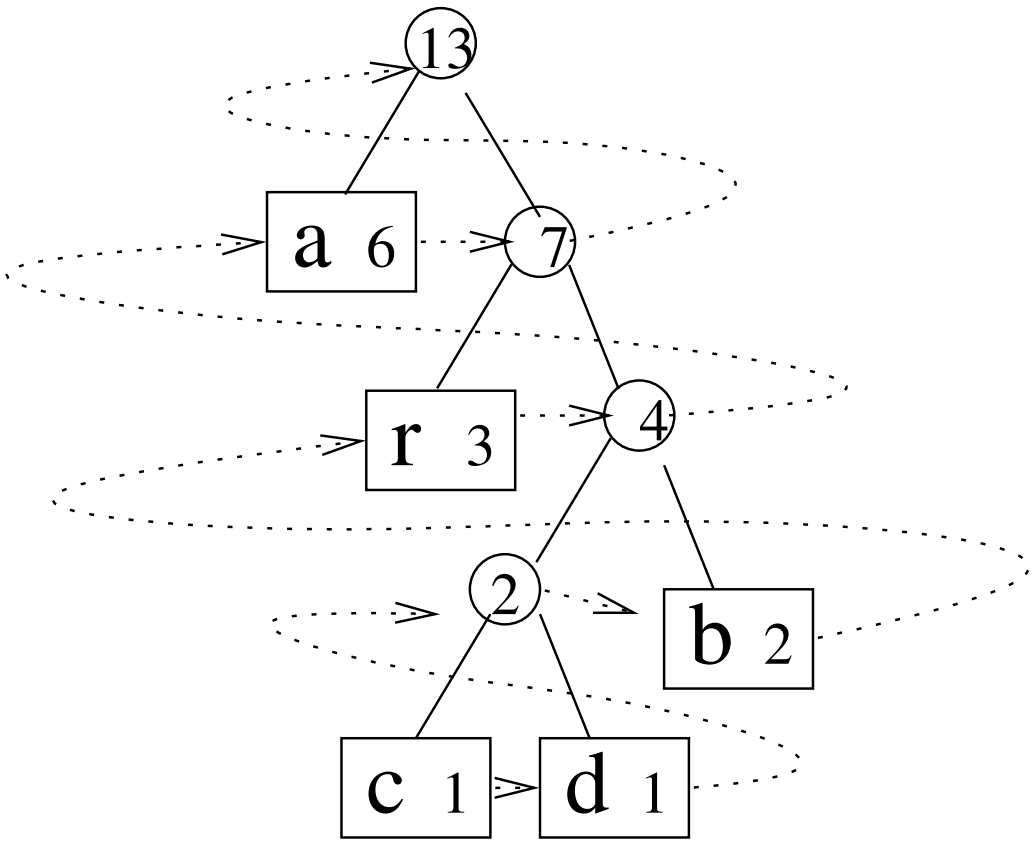
Compression : à la lecture d'une lettre du texte,

- on transmet son code actuel;
- on incrémente sa fréquence et on met à jour l'arbre.

Décompression: le décompresseur mime le compresseur pour disposer des mêmes données après avoir traité la même partie du texte.

Idée générale

Dans un arbre de Huffman, on peut ranger les nœuds dans une liste
- croissante pour les poids
- dans laquelle deux enfants du même nœud sont consécutifs.



Mise à jour dans l'algorithme d'Huffman adaptatif

A la lecture d'une lettre a du texte :

- *si la lettre a déjà été rencontrée :*

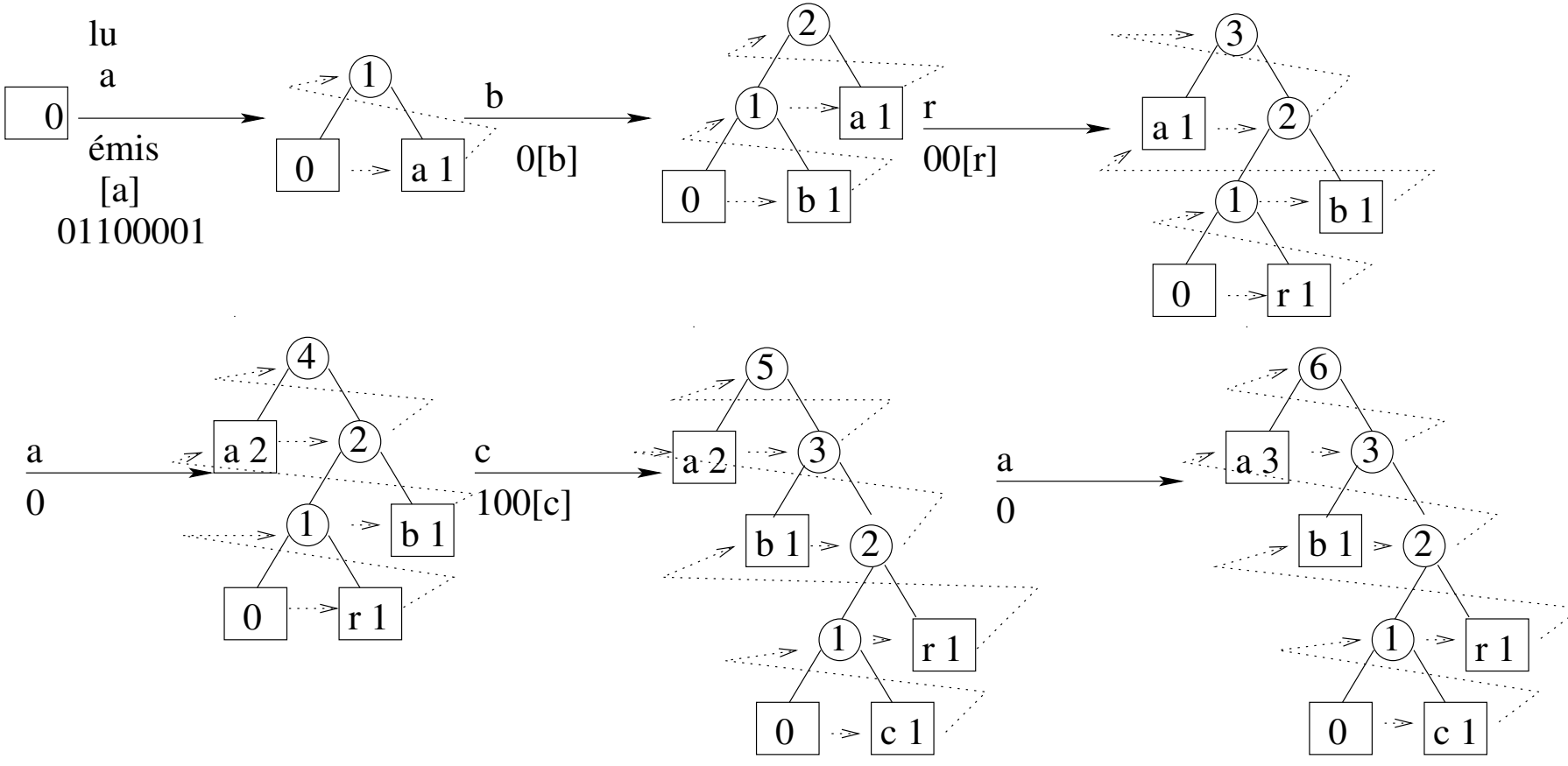
  on échange la feuile d'étiquette a avec le nœud de même poids le plus loin dans la liste, puis on incrémente son poids. On recommence avec le père, jusqu'à la mise à jour de la racine.

- *si la lettre n'a jamais été rencontrée :*

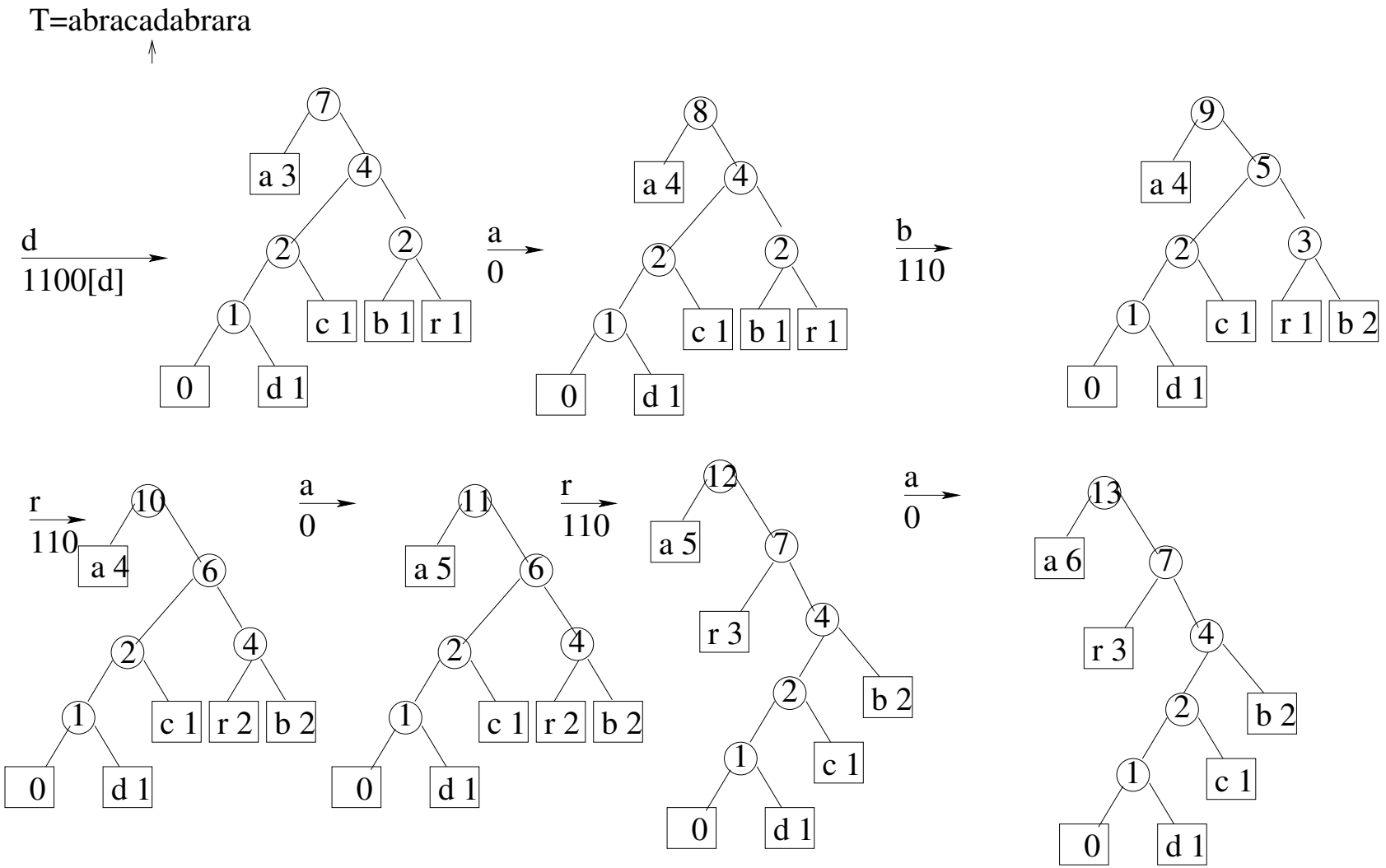
  on conserve dans l'arbre une feuille de poids nul qui représente toutes les nouvelles lettres. On code la lettre a par le code de la feuille de poids nul suivi du code ASCII de a. La feuille de poids nul est remplacée par un nœud ayant pour enfants la feuille de poids nul et une nouvelle feuille de poids 1 d'étiquette a. On poursuit comme dans le cas précédent en incrémentant le poids du nœud parent de 1.

Exemple 1/2

T=abracadabrara



Exemple 2/2


T=abracadabrara
d
1100[d]
a
0
b
110
r
110
a
0
r
110
a
0
7
a 3
4
2
2
1
c 1
b 1
r 1
0
d 1
8
a 4
4
2
2
1
c 1
b 1
r 1
0
d 1
9
a 4
5
2
3
1
c 1
r 1
b 2
0
d 1
10
a 4
6
2
4
1
c 1
r 2
b 2
0
d 1
11
a 5
6
2
4
1
c 1
r 2
b 2
0
d 1
12
a 5
7
r 3
4
2
b 2
1
c 1
0
d 1
13
a 6
7
r 3
4
2
b 2
1
c 1
0
d 1