



# Introduction à l'Algorithmique II

CHAPITRE

CHAÎNE DE CARACTÈRES

COMPRESSION DE DONNÉES : CODAGE DE HUFFMAN

A.U: 2021 - 2022

Animé par: Dr. ibrahim GUELZIM  
Email: algo.smia.fsa@gmail.com

# Codage de l'information

- Codage de texte :
  - Code ASCII : { A-Z ou a-z } sur 8 bits

Exemple 1:

- TXT = "AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDTD"
- 35 caractères
- Codage :  $35 * 8 = 280$  bits

– Amélioration 1:

- Alphabet **restreint**  $\Sigma : \{ A, C, D, E, T \}$ 
    - $|\Sigma| = 5$
    - $2^2 < 5 \leq 2^3$
- 3 bits pour coder
- Longueur **FIXE** pour chaque code

Exemple 1: Codage ::  $35 * 3 = 105$  bits

- Codage à Longueur Fixe

# Codage de l'information

## – Amélioration 2:



- Astuce : " Coder le symbole le plus fréquent sur un nombre minimal de bits "

Exemple 1:

- TXT = "AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDTD"
- Fréquences d'apparitions des lettres :

- A : 15
- E : 9
- C : 6
- D : 4
- T : 1

- Comment coder ?
- Exemple : !

- A sur 1 bit                    Ex : "0"
- E sur 2 bits                   Ex : "01"
- C sur 3 bits                   Ex : "001"
- D sur 4 bits                   Ex : "0101"
- T sur 5 bits                   Ex : "00100"

- TxtCode = "00100010100001010100001010100000100100100100..."

# Codage de l'information

## – Amélioration 2

### Définition Préfixe :

- Soit  $\Sigma$  un alphabet fini = ensemble fini de symboles
  - Exemples d'alphabet :  $\Sigma_1 = \{0, 1\}$ ,  $\Sigma_2 = \{a, b, c, \dots, z\}$
- On appelle  $\Sigma^*$  ("sigma-étoile") l'ensemble de toutes les chaînes (**mots**) de longueur finie utilisant les symboles de l'alphabet  $\Sigma$ . (Ex: "01001", abca")
- La chaîne (mot) **vide** de longueur **zéro**, notée  $\epsilon$  appartient aussi à  $\Sigma^*$
- La longueur de la chaîne  $x$  (notée  $|x|$ ) est le nombre de symboles dans  $x$ .
- La concaténation de deux mots  $x$  et  $y$  est notée  $xy$  ( ou  $x.y$  ou  $x + y$ ) et  $|xy| = |x| + |y|$
- Une chaîne  $w$  est un **préfixe** d'une chaîne  $x$ , si  $\exists y \in \Sigma^* / x = wy$   
→ Une chaîne  $x$  est préfixe d'elle même (  $x = x\epsilon$  )

# Codage de l'information

## – Amélioration 2:

- Idée :

" Coder le symbole le plus fréquent sur un nombre minimal de bits "

- Rappel code : {A : "0", E : "01", C : "001", D : "0101", T : "00100"}

- Décoder:

(TxtCode="00100010100001010100001010100000100100100...")

- 001 : "0"."01" → "AE"

OU : "001" → "C"

- 0101 : "01"."01" → "EE"

OU : "0101" → "D"

→ PK **Ambiguïté ?** (plusieurs façons d'interpréter la même donnée)

→ un symbole est un **préfixe** d'un autre

→ E est préfixe de D : "D" = "EE"

→ C est préfixe de T : "T" = "CAA"

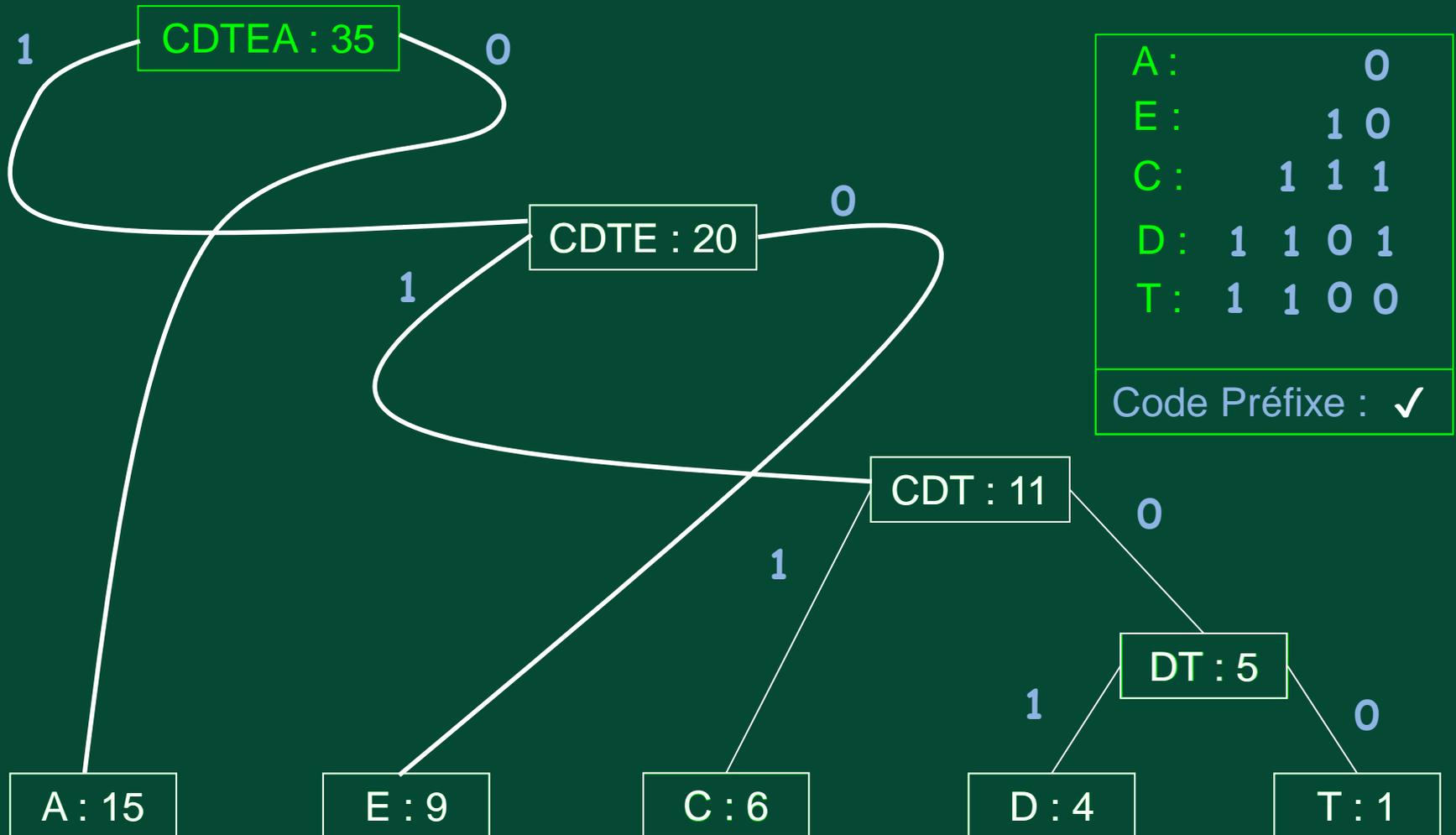
Solution pour éviter l'ambiguïté :

- Proposer un codage où aucun symbole n'en soit un préfixe d'un autre.
- Codage **Préfixe** → Codage de Huffman

# Codage de Huffman

- Application : Exemple 1

TXT = "AEAEEAAEEEAEEEAACCCAAACCCDDDT" (35 car)



A:	0
E:	1 0
C:	1 1 1
D:	1 1 0 1
T:	1 1 0 0

Code Préfixe : ✓

# Codage de Huffman

- Comment coder texte de longueur **NC** ?
  - Créer un tableau de **symboles** TS[] contenant les **symboles alphanumériques** ainsi que leurs **fréquences** d'apparition.
  - **Trier** TS[] par ordre **décroissant** des fréquences d'apparition
  - Un nœud est **orphelin** s'il ne possède pas de père.
  - Construire l'arbre **binnaire** de Huffman de la façon suivante:
    1. Les symboles du TS **trié** dans un ordre **décroissant** constituent les **feuilles** de l'arbre (de gauche à droite)
    2. Créer un père des deux nœuds **orphelins** ayant les plus petites fréquences :
      - i. Donner au nœud père une fréquence égale à la **somme** des fréquences de ses deux fils.
      - ii. La nouvelle valeur du symbole est la **concaténation** du fils droit au fils gauche { Remarque :  $F_q(FG) \geq F_q(FD)$  }
      - iii. Affecter à la branche reliant le père au fils gauche le symbole **binnaire** "1" et au fils droit "0"
    3. Si la fréquence du père est égale à **NC** → **Arrêt**  
Sinon, aller à l'étape 2
  - Le **symbole binnaire** de chaque caractère est construit en écrivant de **droite à gauche** la valeur **binnaire** du chemin allant de la **feuille vers la racine**.

# Codage de Huffman :: Pseudo Code 1 / 11

- Comment ?

I. Lire le texte à coder et construire l'alphabet relative

II. Calculer les fréquences d'apparitions des lettres (symboles)

III. Affecter un code binaire (succession de 0,1) à chaque symbole

IV. Vérifier la justesse (En : correctness) de l'algorithme

- Coder un texte alphanumérique en binaire
- Décoder le texte binaire en texte alphanumérique

- Outils :

Ranger les symboles dans un tableau de sorte que chaque élément contienne:

- Symbole alphanumérique
- Fréquence du symbole
- Symbole binaire

→ Sol : Structure

# Codage de Huffman :: Pseudo Code 2 / 11

- Construire la structure utilisée :

AS:	Fq:
BS :	

- Structure Symb

Début

AS : Chaîne de caracteres

# Alphanumeric String

Fq : Entier

# Frequence d appartition dans le texte

BS : Chaîne de caracteres

# Binary String

Fin

- Pour utiliser un tableau de structure :

Variable TS[] : Structure Symb

AS:	Fq:								
BS :		BS :		BS :		BS :		BS :	

- Procédé (coder le contenu d'un fichier) :

- Mettre le fichier dans une variable considérée comme chaîne
- Parcourir le fichier lettre par lettre pour construire l'alphabet
  - Si la lettre **existe** dans l'alphabet, **incrémenter** sa fréquence par **1**
  - Sinon, **l'ajouter** à l'alphabet en lui **affectant** la fréquence **1**
  - Réalisation : Ecrire une fct qui réalise les points ci-dessus

```

TXT ← LireToutFichier(NomFich) # re-écrire la fonction si besoin
Pour i de 0 à Len(TXT) Faire # construction du Tableau des Symboles TS
    pos ← SymbInTS(TXT[ i ], TS)
    Si pos = -1 alors
        TS[ Len(TS) ] ← ConstrSymb(TXT[ i ])
    Sinon
        TS[ pos ].Fq ← TS[ pos ].Fq + 1
    FinSi
FinPour

```

## Récapitulatif du corps du Pseudo code

```

CopTS ← TriDec(TS), NbLetAlph ← Len(TS), NbSymbPart ← NbLetAlph - 1 # initialisation
Pour i de 0 à NbLetAlph-2 Faire
    AjSymbBinChSymb(TS, CopTS[ NbSymbPart ].AS, "0") # Fils droit
    AjSymbBinChSymb(TS, CopTS[ NbSymbPart - 1 ].AS, "1") # Fils gauche

```

```

CopTS[ NbSymbPart - 1 ].AS ← CopTS[ NbSymbPart - 1 ].AS + CopTS[ NbSymbPart ].AS
CopTS[ NbSymbPart - 1 ].Fq ← CopTS[ NbSymbPart - 1 ].Fq + CopTS[ NbSymbPart ].Fq

```

```

NbSymbPart ← NbSymbPart - 1

```

```

InsererElem( CopTS, NbSymbPart ) # Repositionnement de l'élément à la position NbSymbPart

```

```

FinPour
TxtCode ← CodeTxt(TXT, TS)
Ecrire ("Text Codé par Huffman : ", TxtCode)
TxtDecode ← DecodeBin(TxtCode, TS)
Ecrire (Text Décodé : ", TxtDecode)

```

# Codage de Huffman :: Pseudo Code 3 / 11

- écrire une fct qui
  - renvoie -1 si le caractère à chercher n'appartient pas au tableau de symboles de l'alphabet
  - sinon renvoie la position du caractère dans le tableau,

Fonction `SymbInTS`(car : caractere, TS[] : structure Symb)

Début

Pour i de 0 à Len(TS)-1 Faire # Len(TS) : longueur du tableau TS (analogie / chaine)

    Si car = TS[i].AS alors

        retourner i

    FinSi

FinPour

retourner -1                   # cas où si car ne fait pas partie de TS[]

Fin

## Codage de Huffman :: Pseudo Code 4 / 11

- Écrire une fonction qui **construit** (crée et initialise) un symbole

Fonction **ConstrSymb** ( $C$  : caractère) : Structure Symb

variable  $Sb$  : Structure Symb

Début

$Sb.AS \leftarrow C$

$Sb.Fq \leftarrow 1$

Fin

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAEEAAACCCAACCCDDDDTD"



- TS [ ]** ( Tableau vide au départ : de Longueur 0 )

AS:	Fq:								
BS :		BS :		BS :		BS :		BS :	

# Codage de Huffman :: Pseudo Code 5 / 11

- Remarques:

- Trier TS[] par ordre décroissant / Fréquence des symboles

Variable NbLetAlph : Entier

NbLetAlph  $\leftarrow$  Len(TS)

- Créer une copie de TS[] pour construire l'arbre de Huffman :

Variable CopTS [] : structure Symb

CopTS  $\leftarrow$  TS      # on suppose que cette affectation est permise

# Codage de Huffman : Construction Tableau Symboles

- Exemple 1

"AEAAEEAAAEAAAEEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS :		BS :		BS :	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS :		BS :		BS :	

# Codage de Huffman :: Pseudo Code 6 / 11

Fonction `AjSymbBinChSymb`( `TS[]` : Structure Symb, `ch` : chaine de caractère, `SB` : caractère)

# Ajoute Le Symbole Binaire `SB` aux éléments de la chaine `ch` appartenant au tableau `TS`.

Début

`Lg` ← `Len`( `ch` )

Pour `i` de `0` à `Lg-1` Faire

`pos` ← `SymbInTS`( `ch[i]`, `TS` )     # On suppose que ts les car de `ch` ∈ l'alphabet

`TS[ pos ].BS` = `SB` + `TS[ pos ].BS`

FinPour

Fin

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAAEEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS :		BS : 1		BS : 0	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS :		BS :		BS :	

AjSymbBinChSymb(TS, CopTS[ NbSymbPart ].AS, "0")

# Fils droit

AjSymbBinChSymb(TS, CopTS[ NbSymbPart - 1 ].AS, "1")

# Fils gauche

# Codage de Huffman :: Pseudo Code 7 / 11

- Remarques et commentaires:

# Concaténation du champ AS en utilisant l'opérateur +

$\text{CopTS}[\text{NbSymbPart} - 1].\text{AS} \leftarrow \text{CopTS}[\text{NbSymbPart} - 1].\text{AS} + \text{CopTS}[\text{NbSymbPart}].\text{AS}$

# Incrémentation du champ Fq en utilisant l'opérateur +

$\text{CopTS}[\text{NbSymbPart} - 1].\text{Fq} \leftarrow \text{CopTS}[\text{NbSymbPart} - 1].\text{Fq} + \text{CopTS}[\text{NbSymbPart}].\text{Fq}$

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS :		BS : 1		BS : 0	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS :		BS :		BS :	

$$\begin{aligned} \text{CopTS}[\text{NbSymbPart} - 1].\text{AS} &\leftarrow \text{CopTS}[\text{NbSymbPart} - 1].\text{AS} + \text{CopTS}[\text{NbSymbPart}].\text{AS} \\ \text{CopTS}[\text{NbSymbPart} - 1].\text{Fq} &\leftarrow \text{CopTS}[\text{NbSymbPart} - 1].\text{Fq} + \text{CopTS}[\text{NbSymbPart}].\text{Fq} \end{aligned}$$

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAAEEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS :		BS : 1		BS : 0	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: DT	Fq: 5	AS: T	Fq: 1
BS :		BS :		BS :		BS :		BS :	

# Codage de Huffman : Construction Tableau Symboles

- Exemple 1

"AEAAEEAAAEAAAEEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS :		BS : 1		BS : 0	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: DT	Fq: 5
BS :		BS :		BS :		BS :	

# Ajouter "0" sur les symboles de la chaîne représentant le dernier élément de CopTS

# Codage de Huffman : Construction Tableau Symboles

- Exemple 1

"AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS :		BS : 01		BS : 00	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: DT	Fq: 5
BS :		BS :		BS :		BS :	

# Ajouter "0" sur les symboles de la chaine représentant le dernier élément de CopTS

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS : 1		BS : 01		BS : 00	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: DT	Fq: 5
BS :		BS :		BS :		BS :	

# Ajouter "1" sur les symboles de la chaîne représentant l'avant dernier élément de CopTS

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAAEEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS : 1		BS : 01		BS : 00	

- CopTS[] # copie Non triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: E	Fq: 9	AS: CDT	Fq: 11
BS :		BS :		BS :	

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"A E A A E E E A A E E E E A A A C C C A A A C C C D D D T D"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS : 1		BS : 01		BS : 00	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: CDT	Fq: 11	AS: E	Fq: 9
BS :		BS :		BS :	

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAEEAAACCCAAACCCDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS :		BS : 1		BS : 01		BS : 00	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: CDT	Fq: 11	AS: E	Fq: 9
BS :		BS :		BS :	

# Ajouter "0" sur les symboles de la chaîne représentant le dernier élément de CopTS

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"A E A A E E A A A E E E A A A E E E A A A C C C A A A C C C D D D T D"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS : 0		BS : 1		BS : 01		BS : 00	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: CDT	Fq: 11	AS: E	Fq: 9
BS :		BS :		BS :	

# Ajouter "0" sur les symboles de la chaîne représentant le dernier élément de CopTS

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAAEEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS : 0		BS : 1		BS : 01		BS : 00	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: CDT	Fq: 11	AS: E	Fq: 9
BS :		BS :		BS :	

# Ajouter "1" sur les symboles de la chaîne représentant l'avant dernier élément de CopTS

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS : 0		BS : 11		BS : 101		BS : 100	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: CDT	Fq: 11	AS: E	Fq: 9
BS :		BS :		BS :	

# Ajouter "1" sur les symboles de la chaîne représentant l'avant dernier élément de CopTS

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"A E A A E E A A A E E E A A A E E E A A A C C C A A A C C C D D D T D"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS : 0		BS : 11		BS : 101		BS : 100	

- CopTS[] # copie Non triée dans ordre décroissant de TS

AS: A	Fq: 15	AS: CDTE	Fq: 20
BS :		BS :	

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"A E A A E E E A A E E E E A A E E E E A A C C C A A C C C D D D T D"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS :		BS : 0		BS : 11		BS : 101		BS : 100	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: CDTE	Fq: 20	AS: A	Fq: 15
BS :		BS :	

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS : 0		BS : 0		BS : 11		BS : 101		BS : 100	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: CDTE	Fq: 20	AS: A	Fq: 15
BS :		BS :	

# Ajouter "0" sur les symboles de la chaîne représentant le dernier élément de CopTS

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAAEEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS : 0		BS : 10		BS : 111		BS : 1101		BS : 1100	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: CDTE	Fq: 20	AS: A	Fq: 15
BS :		BS :	

# Ajouter "1" sur les symboles de la chaine représentant l'avant dernier élément de CopTS

# Codage de Huffman : Construction Tableau Symboles

- Exemple

"AEAAEEAAAEAAAEAAACCCAAACCCDDDDTD"

AS: A	Fq: 15	AS: E	Fq: 9	AS: C	Fq: 6	AS: D	Fq: 4	AS: T	Fq: 1
BS : 0		BS : 10		BS : 111		BS : 1101		BS : 1100	

- CopTS[] # copie triée dans ordre décroissant de TS

AS: CDTEA	Fq: 35
BS :	

## Codage de Huffman :: Pseudo Code 8 / 11

- Écrire une fonction pour **coder** le texte **alphanumérique** en un code **binaire**
- À cette étape, l'**alphabet** est construit ainsi que les **symboles binaires**.

Fonction **CodeTxt**(Txt [] : caractere, TS[] : structure Symb) : Chaine de caractères

Variables codeHuff : chaine de caractères

i , pos : Entier

Début

codeHuff ← ""

Pour i de 0 à **Len**(Txt)-1 Faire

**pos** ← **SymbInTS**(Txt[ i ], TS)

    codeHuff ← codeHuff + TS[ **pos** ].BS

FinPour

return codeHuff

Fin

# Codage de Huffman :: Pseudo Code 9 / 11

## # Illustration:

Ecrire ("Text Codé par Huffman : ")

TxtCode  $\leftarrow$  CodeTxt(Text, TS)

Ecrire( "TxtCodé = ", TxtCode )

TxtDecode  $\leftarrow$  DecodeBin(TxtCode , TS)

Ecrire ("Text Décodé : ")

Ecrire (TxtDecode)

- Expl 1:

**TXT** = "AEAAEEAAEEEAEEEAACCCAAACCCDDDTD" (35 car)

Lettre	A	E	C	D	T
Fq	15	9	6	4	1
Binaire	"0"	"10"	"111"	"1101"	"1100"

TxtCodé =

"01000101000010101000010101000011111111000111111111011101110111001101"

## Codage de Huffman :: Décoder

- Expl 1: TXT= "AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDTD" (35 car)

Lettre	A	E	C	D	T
Fq	15	9	6	4	1
Binaire	"0"	"10"	"111"	"1101"	"1100"



- TxtCodé =

"0100010100001010100001010100001111111110001111  
111111011101110111001101"

# Codage de Huffman :: Décoder

- Expl 1: TXT= "AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDTD" (35 car)

Lettre	A	E	C	D	T
Fq	15	9	6	4	1
Binaire	"0"	"10"	"111"	"1101"	"1100"



- TxtCodé =

"0100010100001010100001010100001111111110001111  
111111011101110111001101"

- TxtDecodé =

"A"

# Codage de Huffman :: Décoder

- Expl 1: TXT= "AEAAEEAAAEAAEEAAAEAAACCCAAACCCDDDTD" (35 car)

Lettre	A	E	C	D	T
Fq	15	9	6	4	1
Binaire	"0"	"10"	"111"	"1101"	"1100"



- TxtCodé =

"0100010100001010100001010100001111111110001111  
111111011101110111001101"

- TxtDecodé =

"AEAAEEAAAEAAEEAAAC"



# Codage de Huffman :: Décoder

- Expl 1: TXT= "AEAAEEAAEEEAEEEAACCCAAACCCDDTD" (35 car)

Lettre	A	E	C	D	T
Fq	15	9	6	4	1
Binaire	"0"	"10"	"111"	"1101"	"1100"



- TxtCodé =

"01000101000010101000010101000011111111110001111  
1111111011101110111001101"

- TxtDecodé =

"AEAAEEAAEEEAEEEAACCCAAACCCD"



## Codage de Huffman :: Décoder

- Expl 1: TXT= "AEAAEEAAEEEAEEEAACCCAAACCCDDDTD" (35 car)

Lettre	A	E	C	D	T
Fq	15	9	6	4	1
Binaire	"0"	"10"	"111"	"1101"	"1100"



- TxtCodé =

"01000101000010101000010101000011111111110001111  
11111110111011101110111001101"

- TxtDecodé =

"AEAAEEAAEEEAEEEAACCCAAACCCDDDTD" ✓

## Codage de Huffman :: Pseudo Code 10 / 11

- Écrire une fonction qui **décode** le texte **binaire** en **alphanumérique**
- À cette étape, l'**alphabet** est construit ainsi que les **symboles binaires**.
- Pour ce, écrire une fct qui vérifie si une chaîne binaire **BinCH** représente un symbole de l'alphabet (rangé dans le tableau **TS**)

Fonction **BSinTS** (BinCH [ ] : caractère, TS[ ] : structure Symb): Entier

Variable i : Entier

Début

Pour i de 0 à **Len**(TS) Faire

    Si BinCH = TS[i].BS

**retourner** ( i )

    FinSi

FinPour

**retourner** ( -1 )

Fin

# Codage de Huffman :: Pseudo Code 11 / 11

Fonction `DecodeBin(MotBin[] : caractere, TS[] : structure Symb) : chaine de caracteres`

Variables `TxtDecode[ ]` : caractere  
          `i, j, d` : Entier

Début

`TxtDecode = ""`

`i ← 0, d ← 1`

TantQue (`i < Len(MotBin)`) Faire

`j ← i + d`

`pos ← BSinTS(MotBin[i : j], TS)` # MotBin[i : j] est la chaine de la position i à j-1

    Si `pos = -1` # Symbole MotBin[i : j] NON trouvé

`d ← d + 1`

    Sinon # Symbole MotBin[i : j] trouvé

`TxtDecode ← TxtDecode + TS[pos].AS`

`i ← i + d`

`d ← 1`

FinTantQue

`retourner TxtDecode`

Fin

## Codage de Huffman :: Gain

- Expl 1: **TXT** = "AEAAEEAAEEEAEEEAACCCAAACCCDDDTD" (35 car)

Lettre	A	E	C	D	T
Fq	15	9	6	4	1
Binaire	"0"	"10"	"111"	"1101"	"1100"
Long Bin	1	2	3	4	4

- Longueur du message codé :  
 $15 \times 1 + 9 \times 2 + 6 \times 3 + 4 \times 4 + 1 \times 4 = 71$
- **TxtCodé** =  
"01000101000010101000010101000011111111000111111111011101110111001101"
- Longueur msg par codage **ASCII (8 bit)** :  $35 \times 8 = 280$
- **Evaluation de la compression** :
  - **Quotient de compression** :  $Q = (\text{Taille Initiale}) / (\text{Taille Finale}) = TI / TF$   
*Application Exemple 1 :  $Q1 = 280 / 71$*
  - **Taux de compression** :  $T = 1 / Q$   
*App :  $T1 = 1 / Q1 \sim 0,25$*
  - **Gain de compression** :  $G = 1 - T = (TI - TF) / TI$   
*App :  $G1 = 1 - T1 \sim 0,75$  (75 %)*

# Codage de Huffman :: Complexité

- Construction d'un tableau de  $K$  symboles (alphabet, muni des fréquences)
  - Après lecture du texte ou fichier de  $N$  caractères
  - Pour chaque lettre du fichier ( $N$  fois):
    - Insertion dans un tas  $(\text{Log } K)$  ou
    - Parcours du tas pour  $\text{màj}$   $(\text{Log } K)$
  - D'où : Complexité de la construction de l'alphabet est  $N \text{ Log}(K)$
- Pour la construction des Symboles Binaires (SB) :
  - Parcours de l'alphabet de longueur  $K$
  - Pour  $K-2$  itérations :
    - Suppression d'un élément du Tas  $(\text{Log } K)$
    - Réinsertion du nœud de fusion (père)  $(\text{Log } K)$
  - D'où : Complexité de la construction des SB est  $K \text{ Log}(K)$
- Par conséquent la complexité du codage de Huffman est :
  - $N \text{ Log}(K) + K \text{ Log}(K)$  (souvent  $N \gg K$ ) d'où
  - $T(\text{Huffman}) = N \text{ Log}(K)$