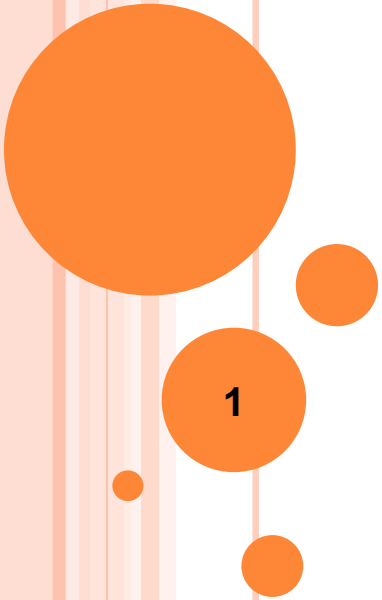
# Introduction au projet
## Codage de Huffman





Sujet conçu avec Jean-Marc Fouet

# CODAGE

- Supposons que nous voulions coder cette phrase.
- Première solution : le code ASCII

| Symbole | Code | Binaire |
|---|---|---|
| S | 83 | 01010011 |
| u | 117 | 01110101 |
| p | 112 | 01110000 |
| o | 111 | 01101111 |
| s | 115 | 01110011 |
| … | … | ... |
| . | 46 | 00101110 |

# Bilan de la solution ASCII

- Puisqu'on sait que chaque caractère est codé sur 8 bits, on n'a pas besoin de séparateur
- Le message est :

  01010011011110101011100000111000 0 ... 00101110

- Il occupe 47*8=376 bits

# CODAGE FIXE

- On peut arriver à un codage plus économique, en remarquant que les seuls caractères utilisés sont S, u, p, o, s, n, blanc, q, e, v, l, i, c, d, r, t, h, a, et point, soit 19 caractères.

| Symbole | Code | Binaire |
|---|---|---|
| S | 0 | 00000 |
| u | 1 | 00001 |
| p | 2 | 00010 |
| o | 3 | 00011 |
| s | 4 | 00100 |
| … | … | ... |
| . | 18 | 10010 |

# Bilan de cette solution

- Le message devient :

  00000000010001000010000011 ... 10010

- Il n'occupe plus que 47*5 = 235 bits.

# CODAGE OPTIMAL

- Etudions la fréquence d'apparition des caractères dans la phrase, et trions-les par ordre décroissant.

| Symbole | Fréquence | Numéro | Binaire |
|---|---|---|---|
| o | 6 | 0 | 0 |
| blanc | 6 | 1 | 1 |
| s | 5 | 2 | 10 |
| e | 5 | 3 | 11 |
| … | … | … | |
| a | 1 | 17 | 10001 |
| . | 1 | 18 | 10010 |

# Bilan de cette solution

- Avantage : les caractères les plus fréquents ont un code plus court
- Le message devient :

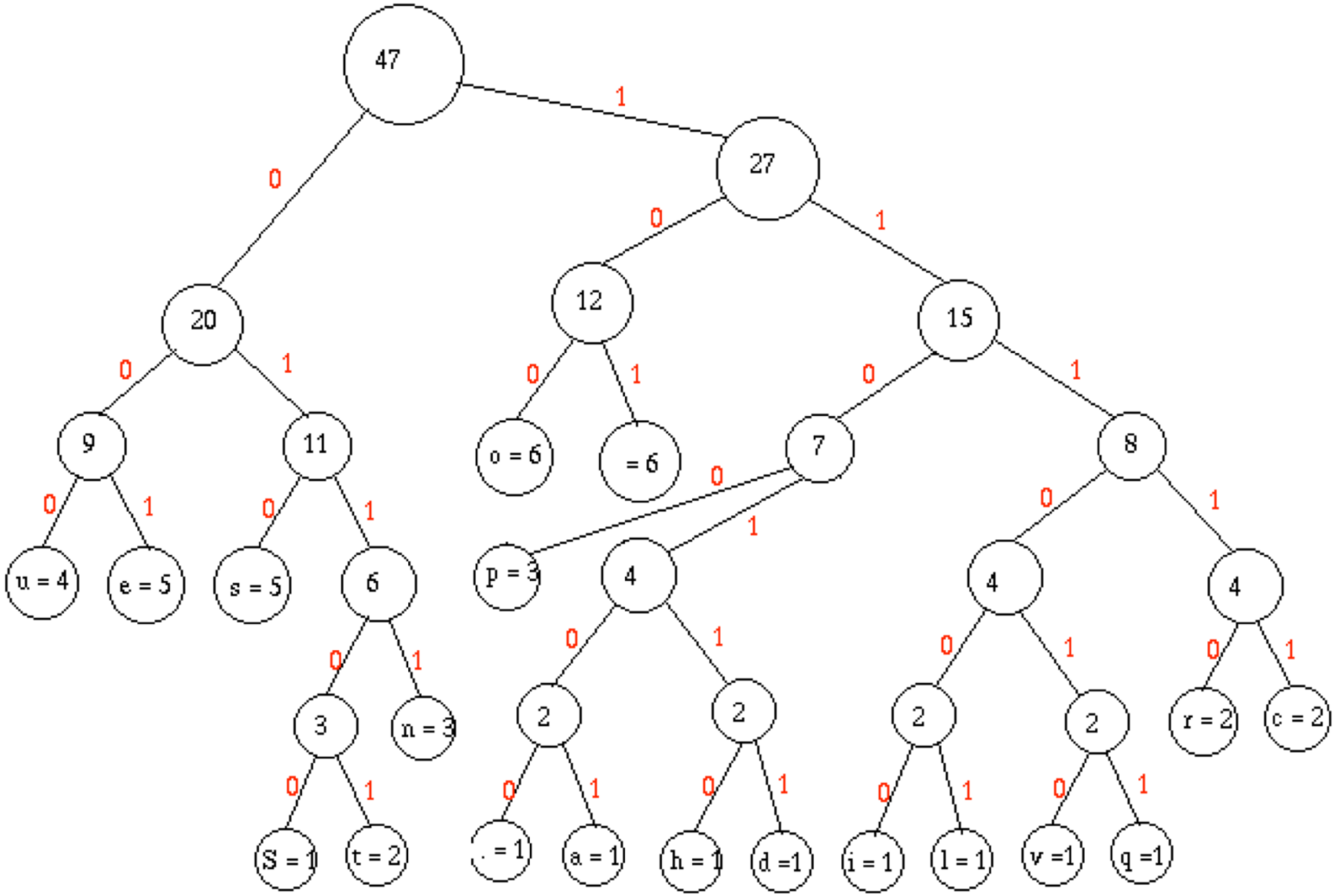
  10101001011010100110101101110 0 ... 10010

- Le message n'occupe plus que :
  (6*1) + (6*1) + (5*2) + (5*2) + (4*3) + (3*3) + (3*3) + (2*3) + (2*4) + (2*4) + 6*4 + 3*5 = 123 bits.
- Mais nous ne pouvons plus le décoder : il nous manque un séparateur
- Il est difficile de trouver un séparateur en binaire qui ne peut être confondu avec aucun caractère, ni avec une succession de caractères. Par exemple, 10011 n'est pas un caractère, mais pourrait très bien être "ue", ou "blanc o o blanc"

# La méthode de Huffman

- La solution consiste à ce qu'aucun code ne soit préfixe d'un autre.
- Nous pouvons garder "0" pour "o", puisqu'aucun autre code ne commence par "0 ».
- En revanche, nous ne pouvons pas garder "1" pour "blanc". Mais nous pouvons prendre "10", à condition de nous interdire qu'un autre code commence par "10".
- Il est clair que nous ne pouvons pas prendre "11" pour "s", à moins de nous interdire des codes commençant par "11", ce qui est impossible puisque nous avons déjà interdit "10xxx".

# REPRÉSENTATION DU CODE PAR UN ARBRE



# Arbre de Huffman

- Les nœuds internes sont des triplets

  (poids fils-gauche fils-droit)
- Les feuilles sont des couples

  (caractère poids)
- Le poids est un entier qui représente la fréquence d'un caractère dans une feuille, ou la somme des fréquences de tous les caractères regroupés par un nœud interne.

# Générer la table de codage

- Pour trouver le code associé à un caractère, on parcourt l'arbre et on associe à chaque caractère représenté dans une feuille le chemin depuis la racine jusqu'à cette feuille

# Résultat

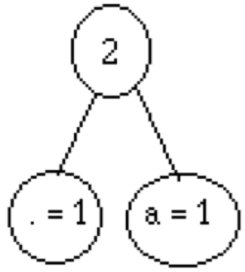
- Les caractères les plus fréquents ont bien les codes les plus cours

| Symbole | Binaire |
|---|---|
| o | 100 |
| blanc | 101 |
| s | 010 |
| e | 001 |
| … | |
| a | 110101 |
| . | 110100 |

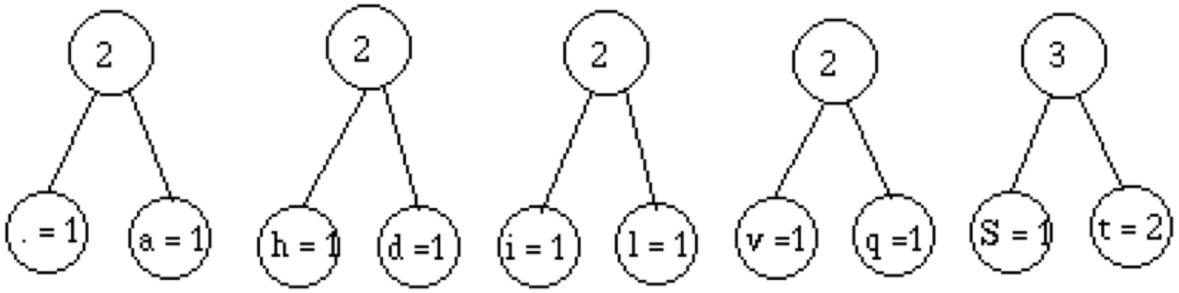
- La représentation en arbre nous garantit qu'aucun code n'est préfixe d'un autre
- Taille du message codé : 185 bits (et on peut décoder)
- On peut démontrer que ce codage est optimal dans le cas général

# COMMENT CONSTRUIRE L'ARBRE DE HUFFMAN ?

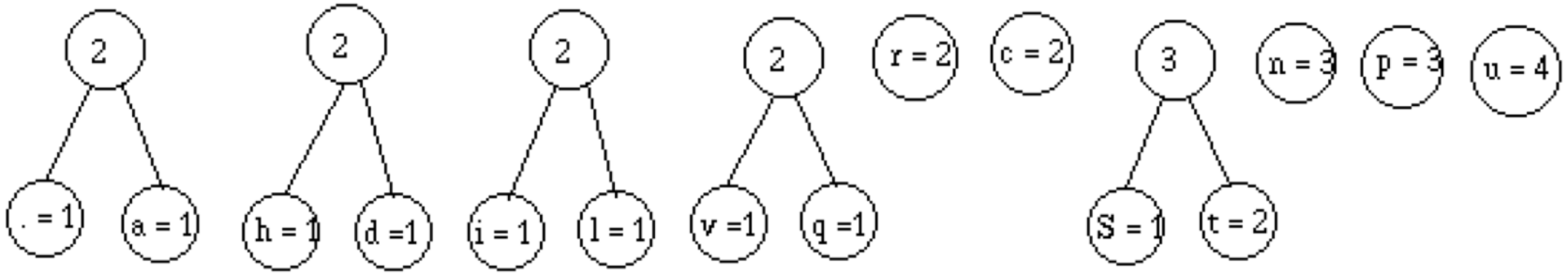
- On part de la table des fréquences en commençant par les plus faibles
- Je regroupe le point et le a :



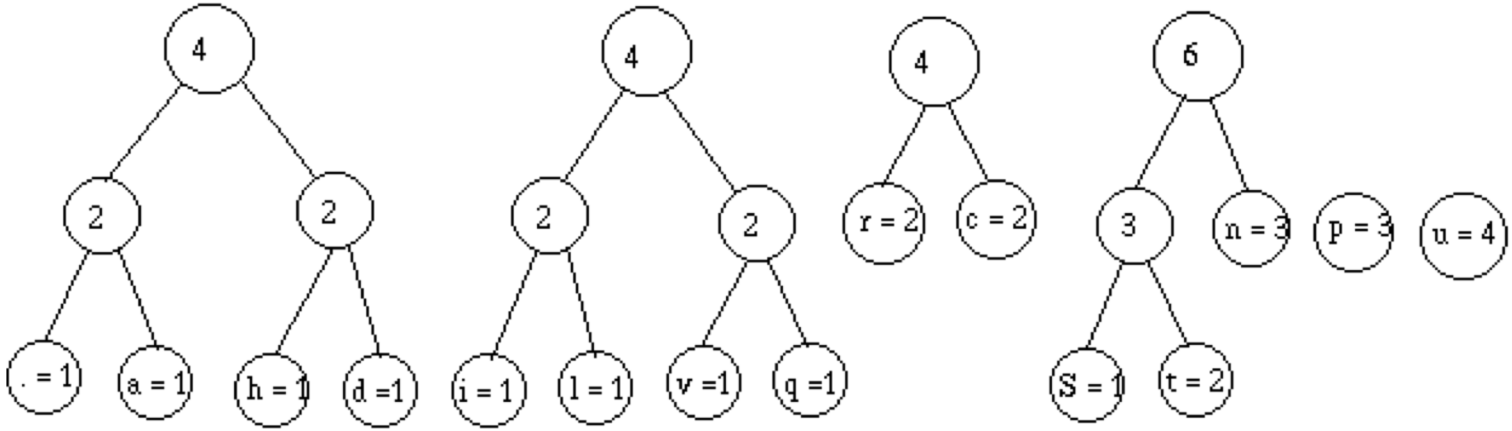

- Et les autres petits, deux par deux :



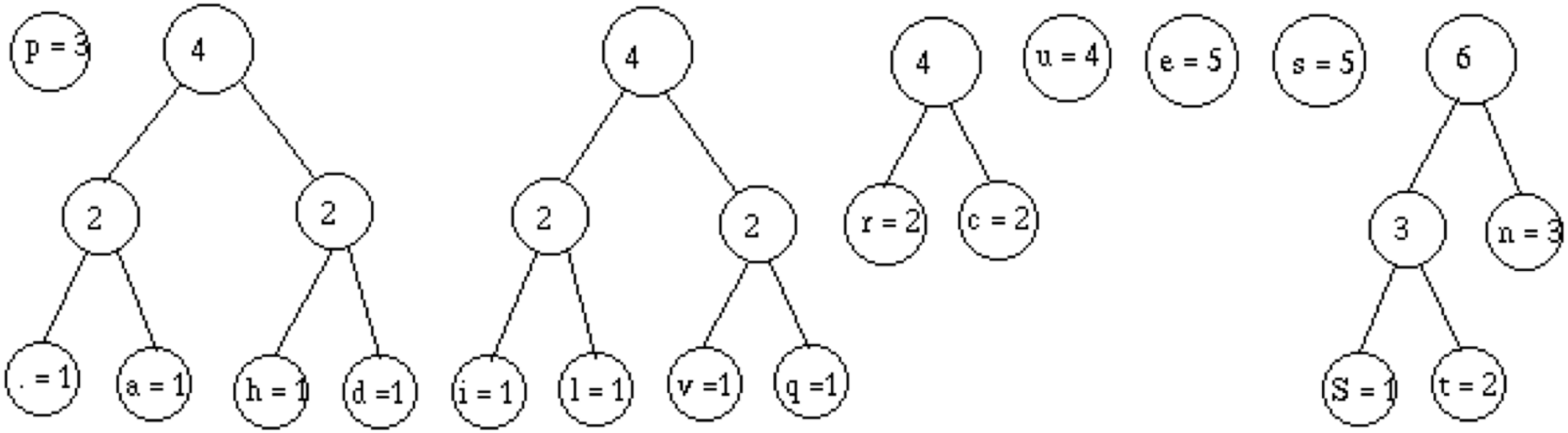
- Je trie :



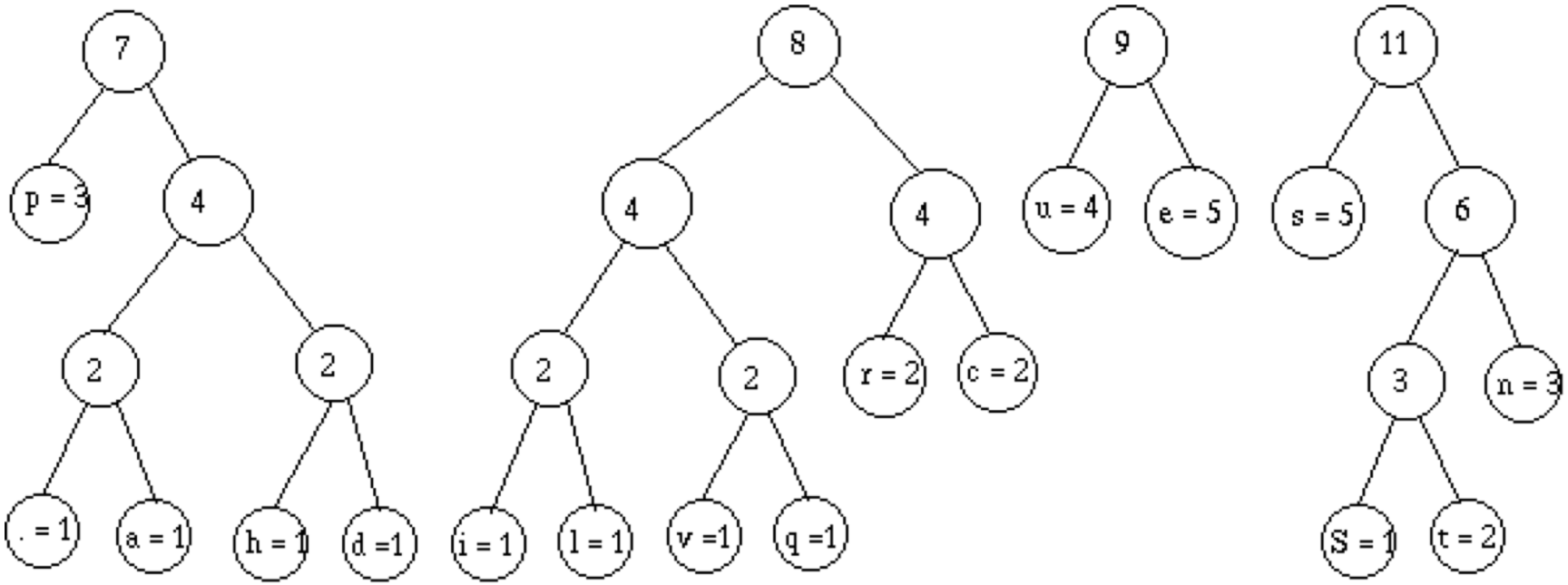

- Je regroupe deux par deux :




- Je retrie



- Je regroupe :




- Je trie, puis je regroupe, etc
- Jusqu'à obtenir un arbre unique avec tous les caractères en feuilles
- A chaque pas de l'algorithme on regroupe les deux premiers arbres

# Résumé des étapes

- Lire le texte à coder
  - ("S" "u" "p" "p" "o" "s" "o" "n" "s" " " "q" "u" ... ".")
- Calculer la fréquence de chaque caractère dans le texte
  - (("S" 1) ("u" 4) ("p" 3) ... ("." 1))
- Trier cette liste de couples (carac freq) suivant les fréquences croissantes
  - (("S" 1) ("q" 1) ("v" 1) ... ("o" 6))

# Résumé des étapes (2)

- Construire l'arbre
  - (47 (20 (9 ("u" 4) ("e" 5)) (11 ("s" 5) (" " 6))) (27 (12 .... ("c" 2)))))
- Cela demande de savoir
  - Fusionner deux arbres de Huffman (qui peuvent être des feuilles)
  - Insérer l'arbre résultat de la fusion dans la liste de couples (carac  fréq)
- Construire la table de codage (voir TD)
  - (("u" (0 0 0)) ("e" (0 0 1)) ... ("c" (1 1 1 1 1)))

# Résumé des étapes (3)

- Coder la phrase
  - (01100000110011001000 ... 110100)
- Et pour décoder ? (voir TD)
  - On utilise l'arbre plutôt que la table

# EXEMPLE D'EXÉCUTION

```
Entrez une liste de caractères à coder : ("a" "b" "r" "a" "c" "a" "d" "a" "b
Phrase :
(a b r a c a d a b r a)
Liste des fréquences :
((a 5) (r 2) (b 2) (d 1) (c 1))
Arbre :
(11 (a 5) (6 (b 2) (4 (2 (d 1) (c 1)) (r 2))))
Table de codage :
((a (0)) (b (1 0)) (d (1 1 0 0)) (c (1 1 0 1)) (r (1 1 1)))
Phrase codée compressée :
(0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0)
```

# INTERACTIONS AVEC L'UTILISATEUR

- Afficher à l'écran : fonction prédéfinie `display`

  Ex :

```
> (display "Bonjour")
Bonjour
> (display 'Bonjour)
bonjour
> (define a 'bonjour)
> (display a)
bonjour
```

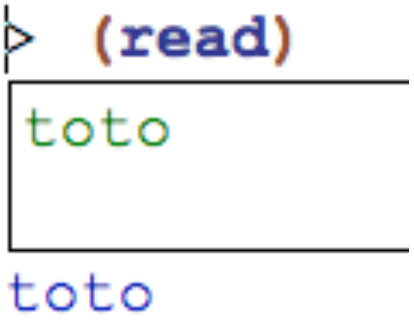
- Passer à la ligne : fonction prédéfinie `newline`

  Ex : `(newline)`

# INTERACTIONS AVEC L'UTILISATEUR

- Lire au clavier : fonction prédéfinie read

Ex :




- Utilisation du résultat de la fonction :

```
(let ((reponse (read)))
        (if (eq? reponse 'o)
            'toto
            'tata))
```

# SORTIR DU FONCTIONNEL

- Utilisation de la séquence : fonction prédéfinie `begin`

```
(define exemple
  (lambda ()
    (begin
      (display "Bonjour, souhaitez-vous continuer (o/n)")
      (newline)
      (let ((reponse (read)))
        (if (eq? reponse 'o)
            (exemple)
            (display "Au revoir !"))))))
```