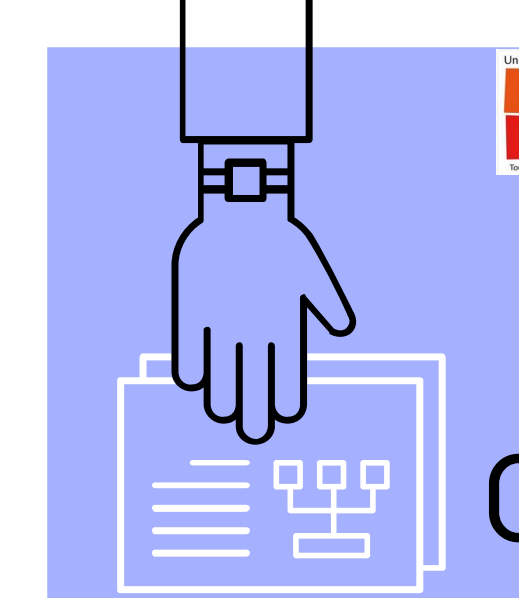
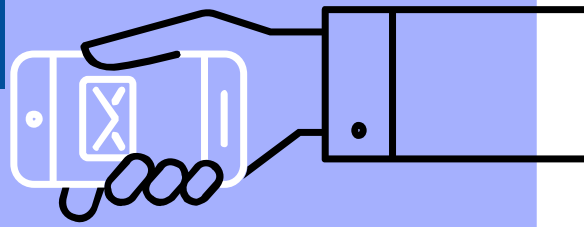
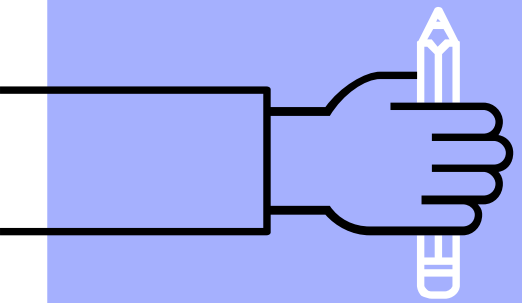
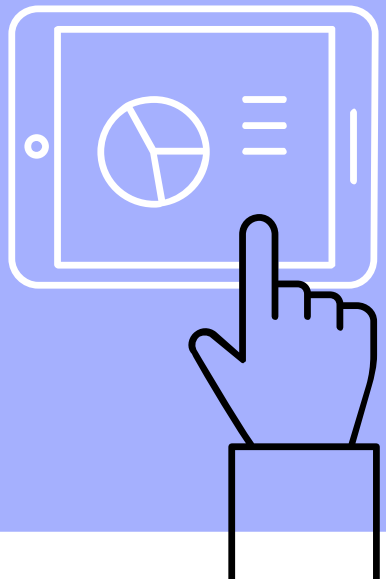




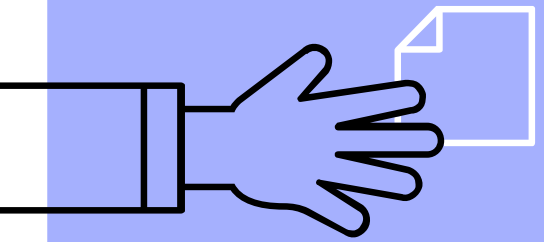
# Codage Huffman

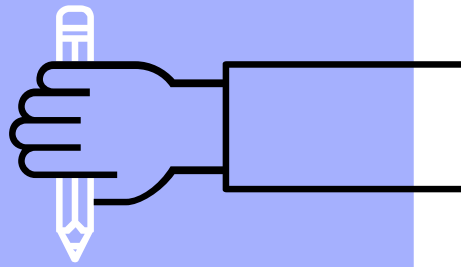
## Projet PIM





# 1.
# Algorithme





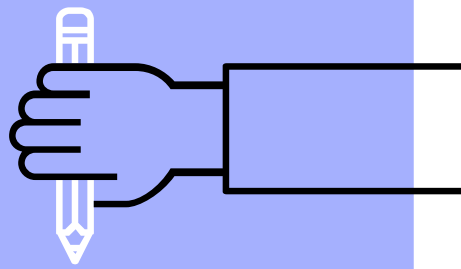
```
==> Creation de l'abre

==> Arbre :
(42)
 \--0--(17)
  |      \--0--(8)
  |      |      \--0--(4)'x'
  |      |      \--1--(4)'m'
  |      \--1--(9)
  |      |      \--0--(4)
  |      |      |      \--0--(2)'l'
  |      |      |      \--1--(2)'
'
  |      |      \--1--(5)' '
 \--1--(25)
  |      \--0--(10)
  |      |      \--0--(5)'t'
  |      |      \--1--(5)
  |      |      |      \--0--(2)
  |      |      |      |      \--0--(1)':'
  |      |      |      |      \--1--(1)
  |      |      |      |      |      \--0--(0)''
  |      |      |      |      |      \--1--(1)'d'
  |      |      |      \--1--(3)'p'
  |      \--1--(15)'e'

==> Table de Huffman :
x : 000
m : 001
l : 0100

 : 0101
  : 011
t : 100
: : 10100
 : 101010
d : 101011
p : 1011
e : 11


==> Creation du fichier compresse
==> Fin
```

```
abennagh@n7-ens-lnx042:~/1A/pim/CD07/src/src$ ./decompresser -b test.txt.hff
=> Liste des octets représentant les caractères : Success
=> Liste Parcours Infixe : Success
=> Reconstruction de l'arbre : Success
=> Liste Correspondance Code Huffman et Character : Success
=> Ecriture dans le fichier décompressé : Success
 File has been successfuly decoded !
 You may consult it in test.txt
Huffman Table :
'x' --> 000
'm' --> 001
'l' --> 0100
'
' --> 0101
' ' --> 011
't' --> 100
':' --> 10100
'' --> 101010
'd' --> 101011
'p' --> 1011
'e' --> 11
abennagh@n7-ens-lnx042:~/1A/pim/CD07/src/src$
```

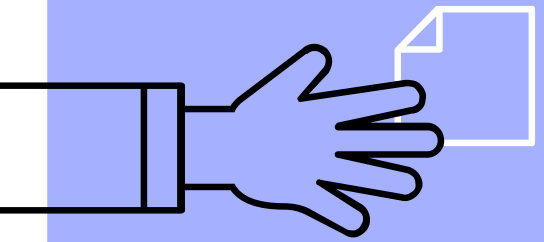
Fichier décompressé

```
abennagh@n7-ens-lnx042:~/1A/pim/CD07/src/src$ xxd test.txt
00000000: 6578 656d 706c 6520 6465 2074 6578 7465  exemple de texte
00000010: 203a 0a65 7865 6d70 7465 2074 656d 7065   :.exempte tempe
00000020: 7465 206c 6578 656d 650a                 te lexeme.
abennagh@n7-ens-lnx042:~/1A/pim/CD07/src/src$ xxd test.txt.hff
00000000: 0878 6d6c 0a20 743a 6470 6565 19c9 7e33  .xml. t:dpee..~3
00000010: 69ba f731 3745 c66e 6e66 f9b4 c675       i..17E.nnf...u
```

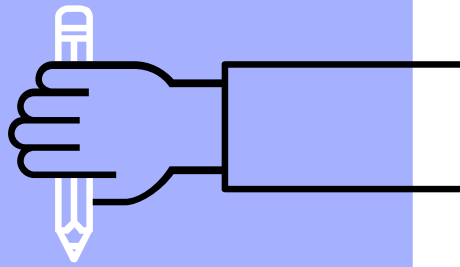
Fichier compressé



# 2.
# Architecture en Modules

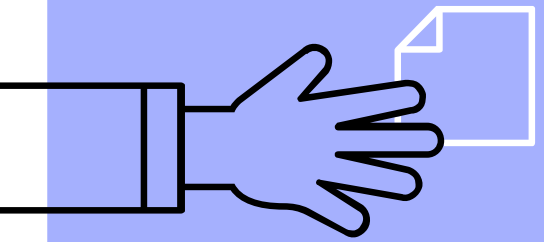


- *Utilisation du module LCA vu en TP*
- *Utilisation d'un module tree permettant de manipuler un type T_tree représentant l'arbre de Huffman*



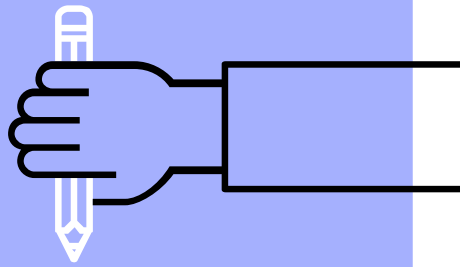
# 3. Structure de données

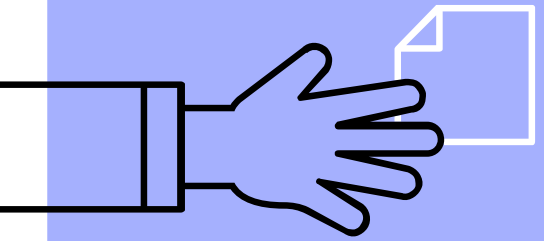
Module Tree



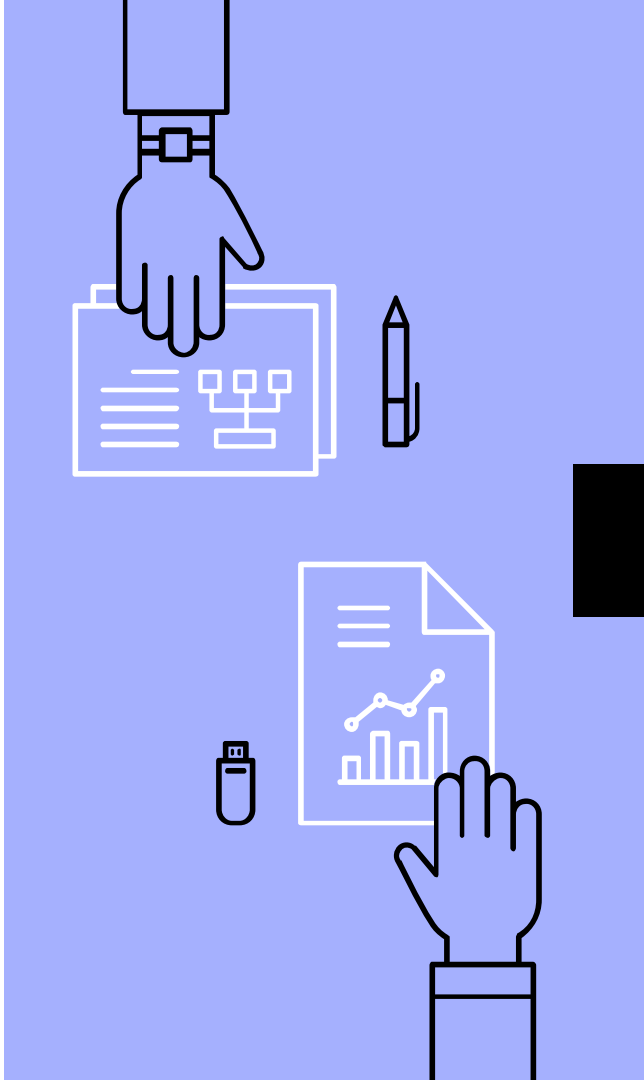
```
TYPE T_Tree  EST POINTEUR VERS T_Node

TYPE T_Node EST UN ENREGISTREMENT
        Left : P_Node
        Right : P_Node
        Data : T_data
        id : T_id
FIN ENREGISTREMENT
```

# 4. Principaux Algorithmes

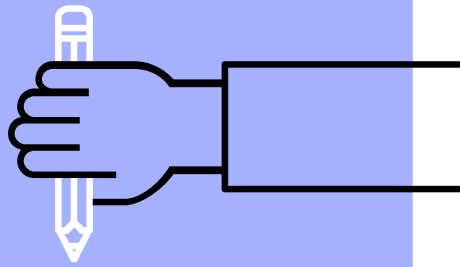


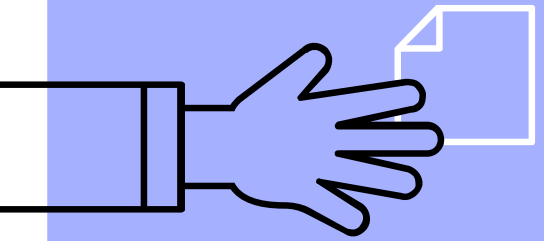


- Affichage de l’arbre
- Codage Huffman d’une clé
- Parcours Infixe d’un arbre
- Construction d’un arbre à partir d’un parcours infixe et une liste de clés
- Construction de l’arbre de Huffman à partir de feuilles dont les clés représentent les caractères et les données leur fréquence
- Contrainte : Écriture en octets dans le fichier

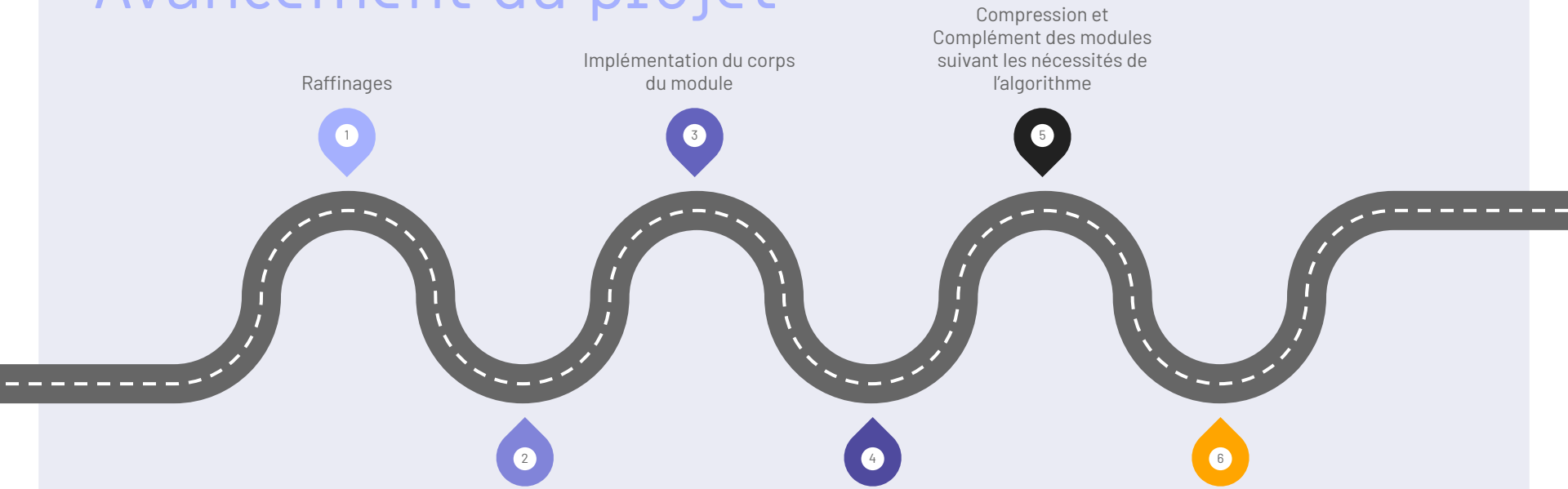


# 5. Avancement du projet

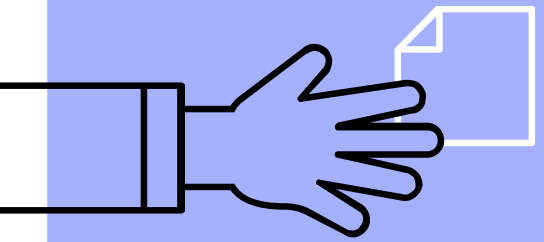


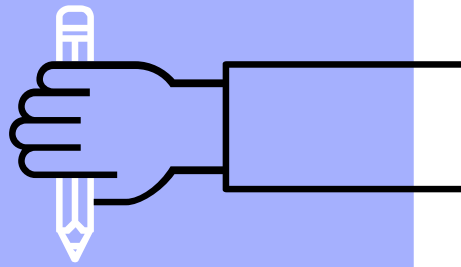


# Avancement du projet



# Compression





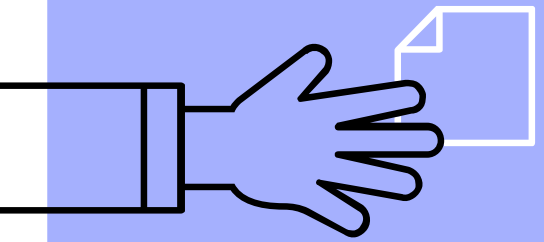
**R0 : Compresser un fichier txt**

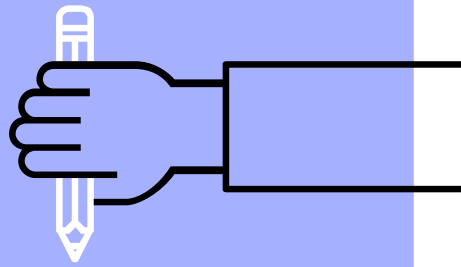
**R1 : Comment “Compresser un fichier txt” ?**

- Lire un fichier txt  Liste_Octet : in
- Compter le nombre d’occurence de chaque caractère  Liste_Octet : in out LCA_Integer_Octet
- Construire l’arbre de Huffman Liste_Tree : out LCA_Integer_Tree; Liste_Octet : in out
- Enregistrer le Codage de Huffman  Code : in out Unbounded_String

```
==> Creation de l'abre

==> Arbre :
(42)
 \--0--(17)
 |      \--0--(8)
 |      |      \--0--(4)'x'
 |      |      \--1--(4)'m'
 |      \--1--(9)
 |      |      \--0--(4)
 |      |      |      \--0--(2)'l'
 |      |      |      \--1--(2)'
'
 |      |      \--1--(5)' '
 \--1--(25)
 |      \--0--(10)
 |      |      \--0--(5)'t'
 |      |      \--1--(5)
 |      |      |      \--0--(2)
 |      |      |      |      \--0--(1)':'
 |      |      |      |      \--1--(1)
 |      |      |      |      |      \--0--(0)''
 |      |      |      |      |      \--1--(1)'d'
 |      |      |      \--1--(3)'p'
 |      \--1--(15)'e'

==> Table de Huffman :
x : 000
m : 001
l : 0100

 : 0101
  : 011
t : 100
: : 10100
 : 101010
d : 101011
p : 1011
e : 11


==> Creation du fichier compresse
==> Fin
```

# Décompression





**R0: Décompresser un fichier .hff**

**R1: Comment " Décompresser un fichier .hff"?**

- Lire un fichier .hff -> File_name: Chaîne de caractères
- Enregistrer les octets représentant les caractères -> Lca_Char : out T_LCA
- Stocker la suite du fichier dans une variable -> Suite : Chaîne de caractères
- Enregistrer le parcours infixe -> Lca_parcours : out T_LCA
- Reconstruire l'arbre de Huffman -> Lca_Char, Lca_parcours : in T_LCA , Created_tree: out T_Tree
- Enregistrer le codage de Huffman correspondant à chaque caractère -> Lca_Char_Code : out T_LCA, Created_Tree: in T_tree
- Ecrire le texte décodé dans un fichier texte -> Lca_Char_Code : in T_LCA

```
abennagh@n7-ens-lnx042:~/1A/pim/CD07/src/src$ ./decompresser -
=> Liste des octets représentant les caractères : Success
=> Liste Parcours Infixe : Success
=> Reconstruction de l'arbre : Success
=> Liste Correspondance Code Huffman et Character : Success
=> Ecriture dans le fichier décompressé : Success
 File has been successfuly decoded !
 You may consult it in test.txt
Huffman Table :
'x' --> 000
'm' --> 001
'l' --> 0100
'
' --> 0101
' ' --> 011
't' --> 100
':' --> 10100
'' --> 101010
'd' --> 101011
'p' --> 1011
'e' --> 11
```

FIN