Projet

Informatique et Science du Numérique

Exemple : Compression de fichier par codage de Huffman

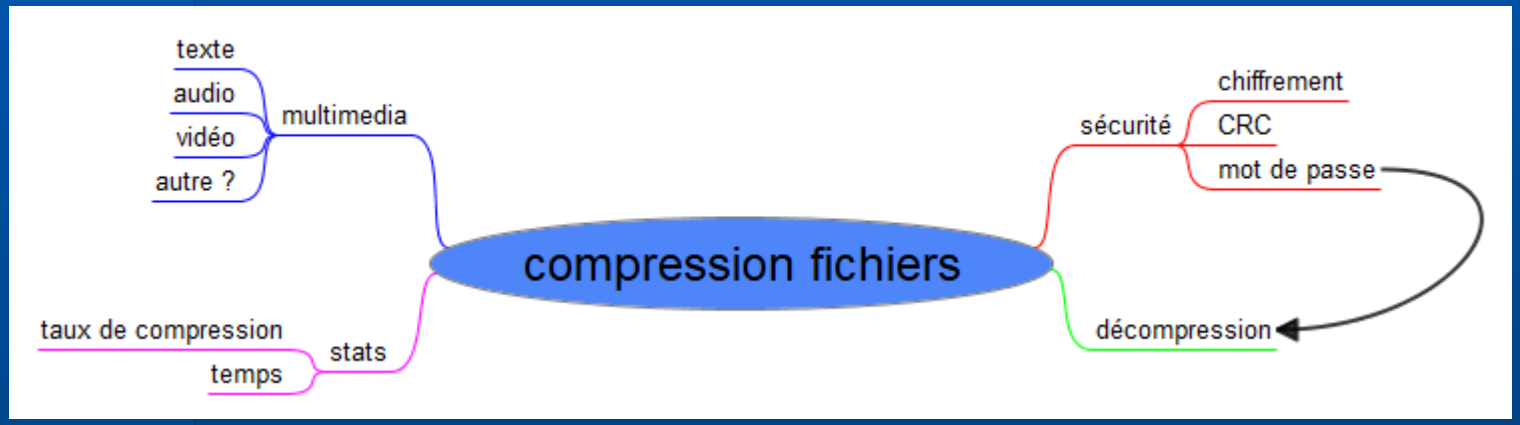
Les 4 fondamentaux de l'informatique

1. Données
2. Algorithme
3. Langage
4. Machine

Projet

Informatique et Science du Numérique

Avant projet



1. Faisabilité
2. Périmètre
3. SMART ?

Projet

Informatique et Science du Numérique

Présentation de la méthode du codage de Huffman

Recoder les données qui ont une occurrence très faible sur une longueur binaire supérieure à la moyenne, et recoder les données très fréquentes sur une longueur binaire très courte.

1. Calcul des fréquences des caractères du texte
2. Construction d'un arbre binaire (voir ci-dessous)
3. Coder en parcourant l'arbre depuis le haut en allant vers la lettre :
   - mouvement à gauche est codé 0
   - un mouvement à droite est codé 1

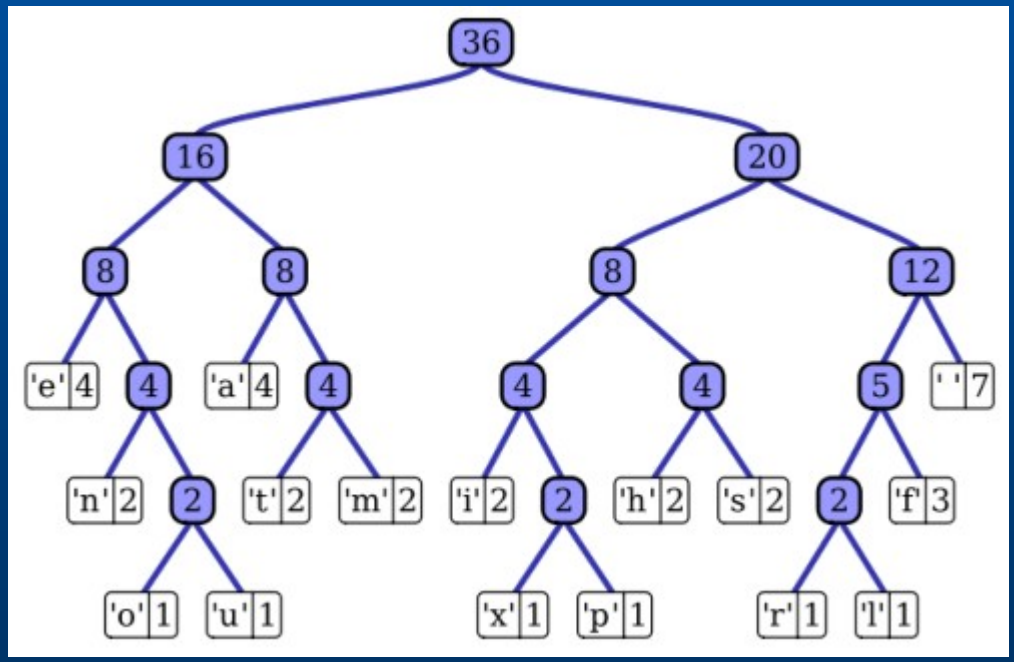
Projet

Informatique et Science du Numérique

Arbre de Huffman

Un exemple d'arbre obtenu avec la phrase
« this is an example of a huffman tree »

‘a’ est codé 010
‘x’ est codé 10010



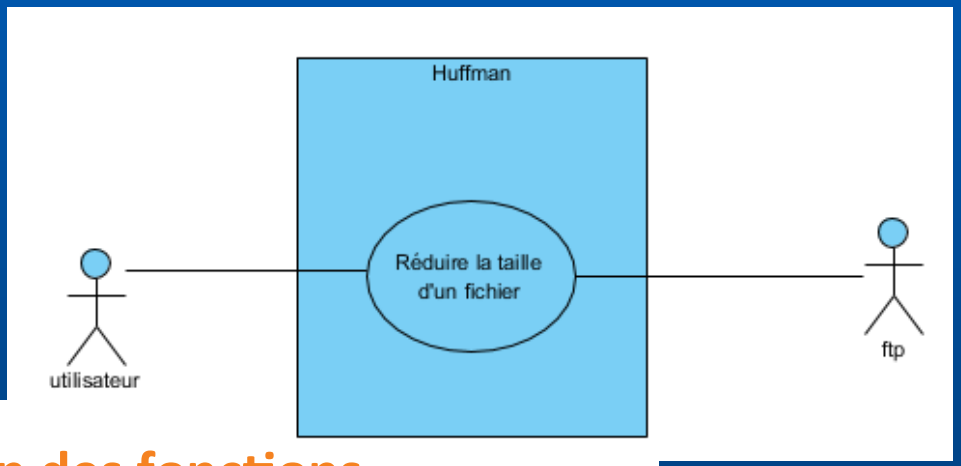
Projet

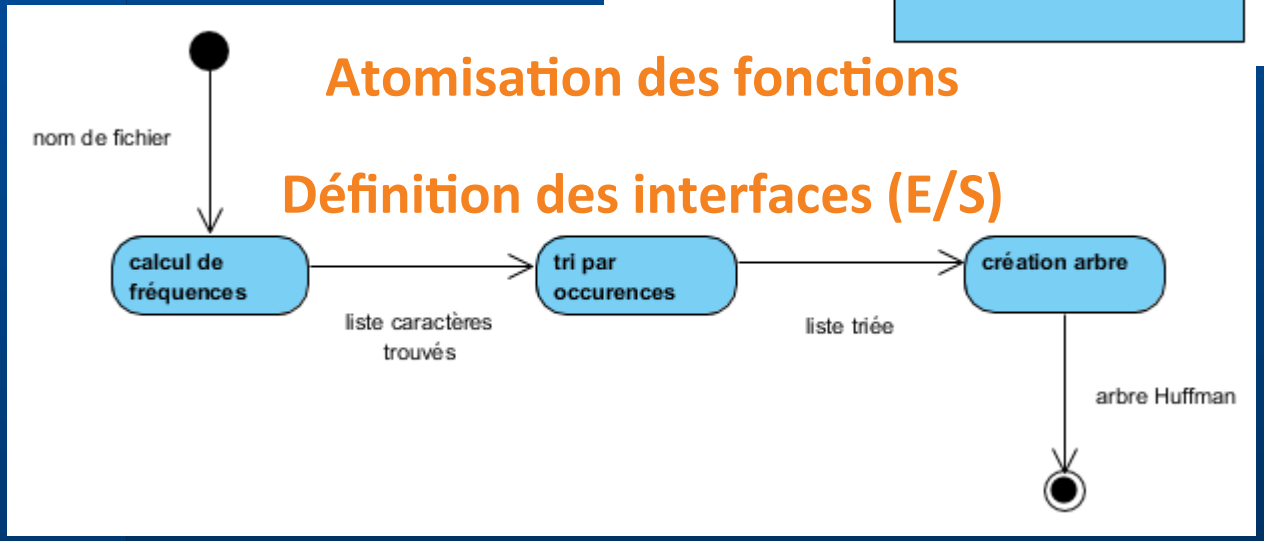
Informatique et Science du Numérique

Modélisation UML

Objectif du projet

Fonctionnalités du programme



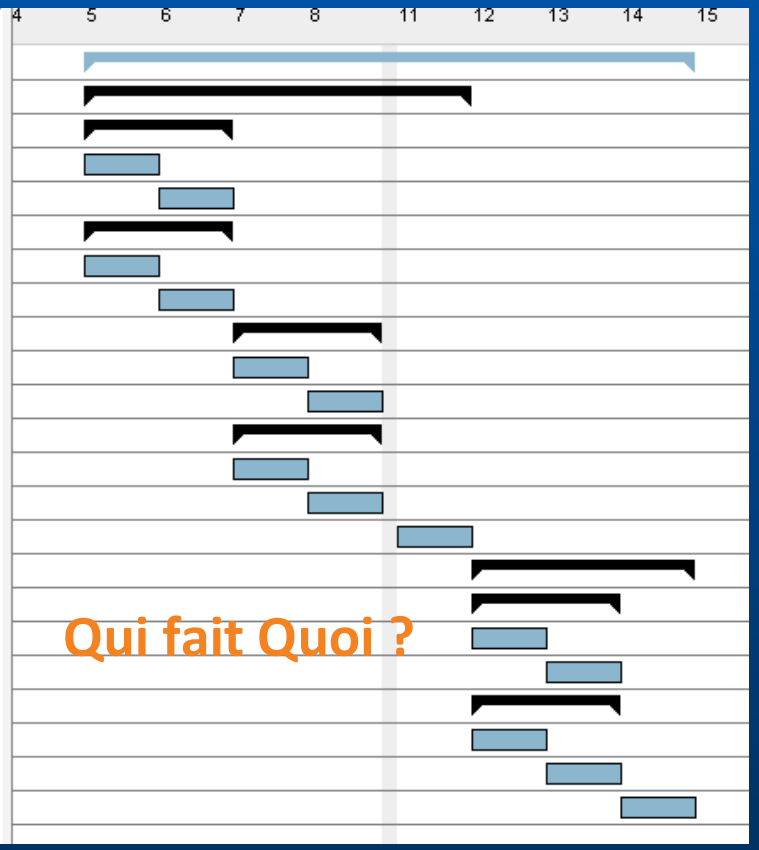


Projet

Informatique et Science du Numérique

Planification

Nom	Date de début	Date de fin
compression fichier	05/05/20	14/05/20
compression	05/05/20	11/05/20
calcul fréquences	05/05/20	06/05/20
dev fonction	05/05/20	05/05/20
tests unitaires	06/05/20	06/05/20
tri occurrences	05/05/20	06/05/20
dev fonction	05/05/20	05/05/20
tests unitaires	06/05/20	06/05/20
création arbre	07/05/20	08/05/20
dev fonction	07/05/20	07/05/20
tests unitaires	08/05/20	08/05/20
enregistrement fichier	07/05/20	08/05/20
dev fonction	07/05/20	07/05/20
tests unitaires	08/05/20	08/05/20
tests intégration	11/05/20	11/05/20
décompression	12/05/20	14/05/20
lecture fichier	12/05/20	13/05/20
dev fonction	12/05/20	12/05/20
tests unitaires	13/05/20	13/05/20
décodage fichier	12/05/20	13/05/20
dev fonction	12/05/20	12/05/20
tests unitaires	13/05/20	13/05/20
tests intégration	14/05/20	14/05/20



Projet

Informatique et Science du Numérique

Algorithme

```
Procédure tri_à_bulles_optimisé(T : Tableau d'éléments)
Variables
   i, j : entier
   taille : entier := taille(T)
   permutation : booléen := vrai

Début
   i := 0

   Tantque permutation faire
      permutation := faux
      pour j allant de 0 à taille – i – 1
         si T[j+1] < T[j] alors
            échanger(T[j+1], T[j])
            permutation := vrai
      i := i + 1
Fin
```

Scénario nominal

Optimisation performance

Projet

Informatique et Science du Numérique

Implémentation

```
def bubble_sort(sorted : list) -> list:
    """
    fonction tri a bulle
    permutation des éléments 2 à 2 en faisant remonter la plus grande valeur en fin de vecteur

    arguments:
            sorted – liste à trier

    retour:
            liste triée en place
    """
    permut, total = True, 0

    while permut:
        permut = False
        # le dernier élément permuté est forcément bien placé
        for i in range(len(sorted) - total -1):
            if sorted[i] > sorted[i+1]:
                permut = swap(i, i+1)          # renvoie True si permutation
        total += 1

    return sorted
```

Modèle de la boite noire

Justification du code

Gestion des bugs

Projet

Informatique et Science du Numérique

Tests unitaires

```
assert bubble_sort([]) == []
assert bubble_sort([1]) == [1]
assert bubble_sort([3, 2, 1]) == [1, 2, 3]
assert bubble_sort(['c', 'b', 'a']) == ['a', 'b', 'c']
assert bubble_sort(['c', 1, True]) == None
assert bubble_sort("cba") == None
assert bubble_sort(321) == None
```

Pertinence des tests
Taux de couverture

Modifier l'implémentation en conséquence

Projet

Informatique et Science du Numérique

Tests de performance

```
# coding: utf-8

import time
import random
Import bsort

size = [10, 100, 200, 500, 1000, 2000, 5000, 10_000, 20_000, 50_000]

# génération nombres aléatoires
for i in range(size):
    sorted = [] # Liste vide
    for j in range(i):
        sorted.append(random.randint(0, 100))

    start = time.clock()
    bsort.bubble_sort(sorted)
    ellapse = time.clock() - start

    # format pour import tableur
    print("{1}, {2}".format(i, .ellapse))
```

Tests de performance

volumétrie/vitesse

Projet

Informatique et Science du Numérique

Tests de performance

Temps d'exécution en $O(n^2)$