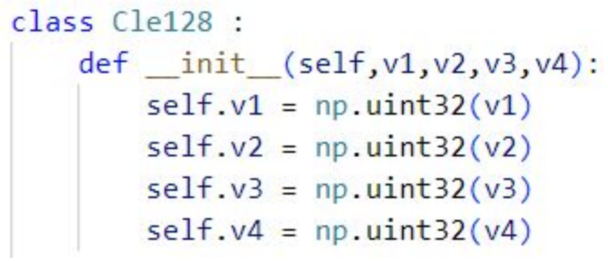
Présentation Projet ALGAV

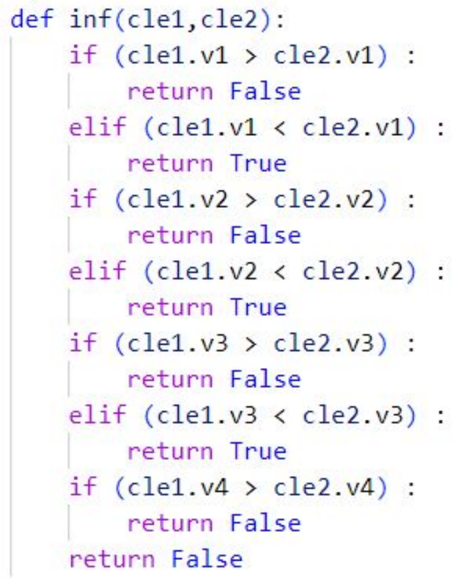
Laura RATTANAVAN
Julien FANG

2023/2024

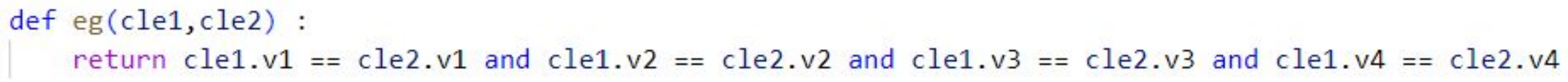
Échauffement



```python
class Cle128 :
    def __init__(self,v1,v2,v3,v4):
        self.v1 = np.uint32(v1)
        self.v2 = np.uint32(v2)
        self.v3 = np.uint32(v3)
        self.v4 = np.uint32(v4)
```

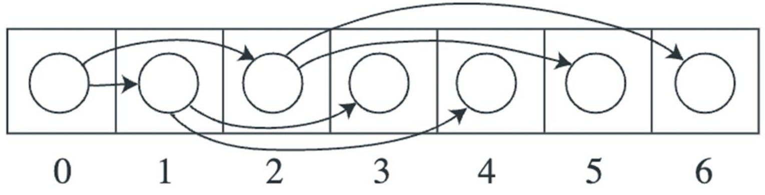


```python
def inf(cle1,cle2):
    if (cle1.v1 > cle2.v1) :
        return False
    elif (cle1.v1 < cle2.v1) :
        return True
    if (cle1.v2 > cle2.v2) :
        return False
    elif (cle1.v2 < cle2.v2) :
        return True
    if (cle1.v3 > cle2.v3) :
        return False
    elif (cle1.v3 < cle2.v3) :
        return True
    if (cle1.v4 > cle2.v4) :
        return False
    return False
```



```python
def eg(cle1,cle2) :
    return cle1.v1 == cle2.v1 and cle1.v2 == cle2.v2 and cle1.v3 == cle2.v3 and cle1.v4 == cle2.v4
```

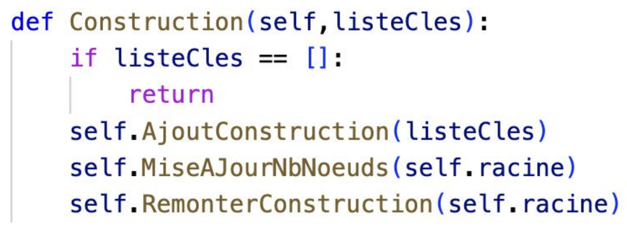
Structure Tas Min : Arbre



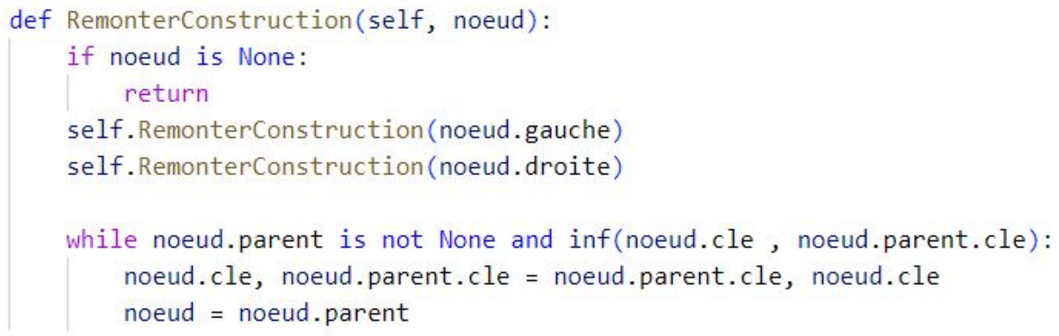
```python
class Noeud:
    def __init__(self,cle):
        self.cle = cle
        self.nbNoeuds = 1
        self.parent = None
        self.gauche = None
        self.droite = None


class TasMinArbre :
    def __init__(self):
        self.racine = None


def Construction(self,listeCles):
    if listeCles is None:
         return
    self.AjoutConstruction(listeCles)
    self.MiseAJourNbNoeuds(self.racine)
    self.RemonterConstruction(self.racine)
```

```python
def AjoutConstruction(self,listeCles):
    self.racine = Noeud(listeCles[0])
    self.AjoutConstructionRec(self.racine,listeCles,0)


def AjoutConstructionRec(self, noeud, listeCles, position):
    if noeud is None or position >= len(listeCles):
        return
    gauche_position = 2 * position + 1
    droite_position = 2 * position + 2

    if gauche_position < len(listeCles):
        noeud.gauche = Noeud(listeCles[gauche_position])
        noeud.gauche.parent = noeud
        self.AjoutConstructionRec(noeud.gauche, listeCles, gauche_position)

    if droite_position < len(listeCles):
        noeud.droite = Noeud(listeCles[droite_position])
        noeud.droite.parent = noeud
        self.AjoutConstructionRec(noeud.droite, listeCles, droite_position)
```

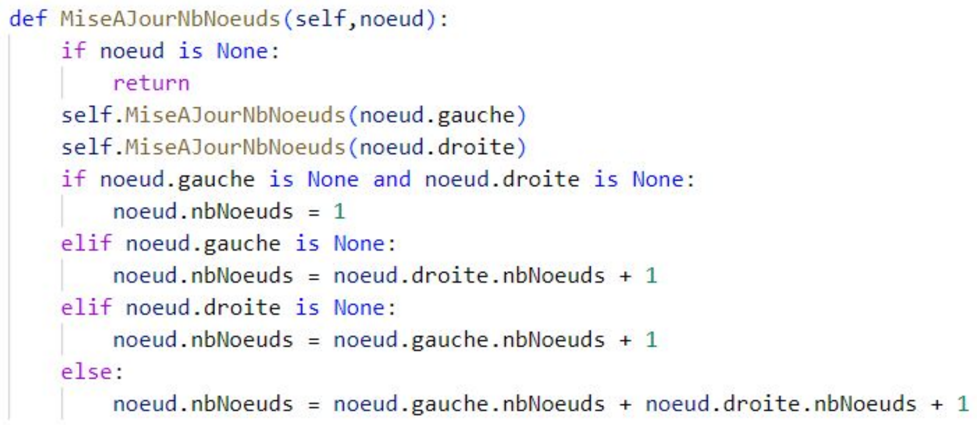
Structure Tas Min : Arbre



```
def Construction(self,listeCles):
    if listeCles == []:
        return
    self.AjoutConstruction(listeCles)
    self.MiseAJourNbNoeuds(self.racine)
    self.RemonterConstruction(self.racine)
```

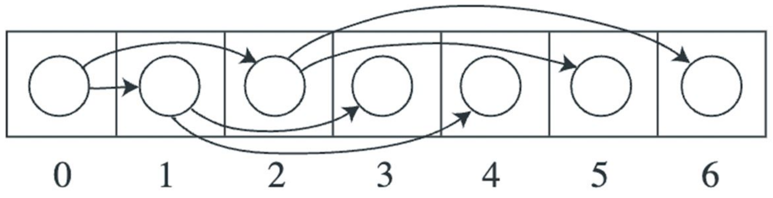


```
def RemonterConstruction(self, noeud):
    if noeud is None:
        return
    self.RemonterConstruction(noeud.gauche)
    self.RemonterConstruction(noeud.droite)

    while noeud.parent is not None and inf(noeud.cle , noeud.parent.cle):
        noeud.cle, noeud.parent.cle = noeud.parent.cle, noeud.cle
        noeud = noeud.parent
```



```
def MiseAJourNbNoeuds(self,noeud):
    if noeud is None:
        return
    self.MiseAJourNbNoeuds(noeud.gauche)
    self.MiseAJourNbNoeuds(noeud.droite)
    if noeud.gauche is None and noeud.droite is None:
        noeud.nbNoeuds = 1
    elif noeud.gauche is None:
        noeud.nbNoeuds = noeud.droite.nbNoeuds + 1
    elif noeud.droite is None:
        noeud.nbNoeuds = noeud.gauche.nbNoeuds + 1
    else:
        noeud.nbNoeuds = noeud.gauche.nbNoeuds + noeud.droite.nbNoeuds + 1
```

Structure Tas Min : Tableau

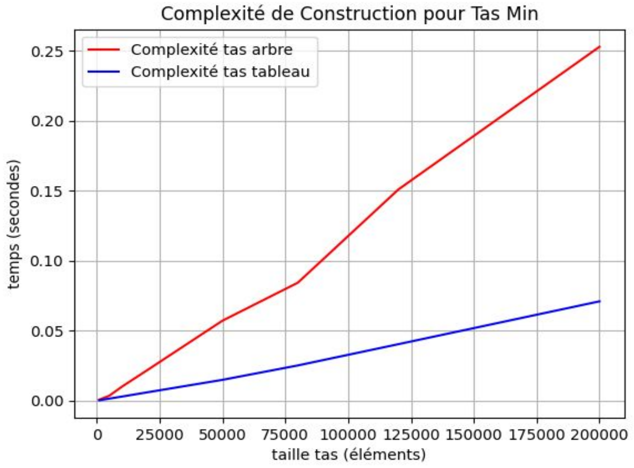


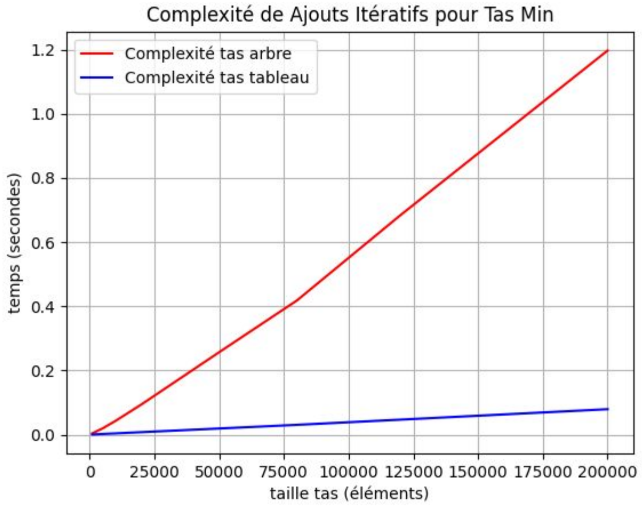
```python
class TasMinTableau:
    def __init__(self):
        self.tas=[]
```

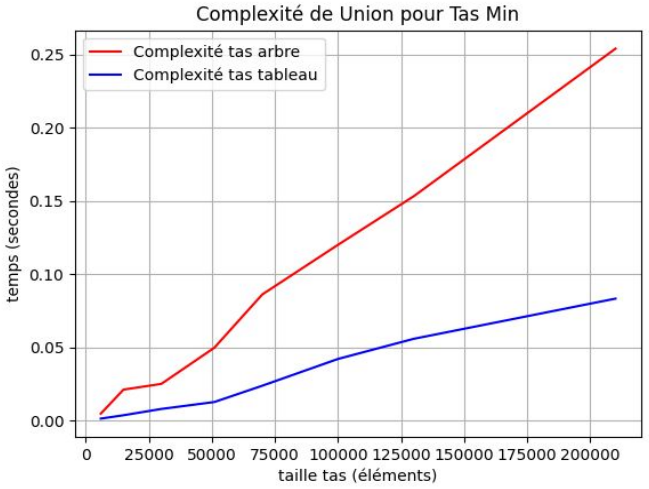
```python
def RemonterConstruction(self, indice):
    taille = len(self.tas)
    while True :
        gauche = 2 * indice + 1
        droite = 2 * indice + 2
        indice_min = indice

        if gauche < taille and inf(self.tas[gauche] , self.tas[indice_min]):
            indice_min = gauche

        if droite < taille and inf(self.tas[droite] , self.tas[indice_min]):
            indice_min = droite

        if indice_min != indice:
            self.tas[indice], self.tas[indice_min] = self.tas[indice_min], self.tas[indice]
            indice = indice_min
        else:
            break
```

```python
def Construction(self, listeCles):
    self.tas = listeCles
    taille = len(self.tas)
    for i in range(taille // 2, -1, -1):
        self.RemonterConstruction(i)
```

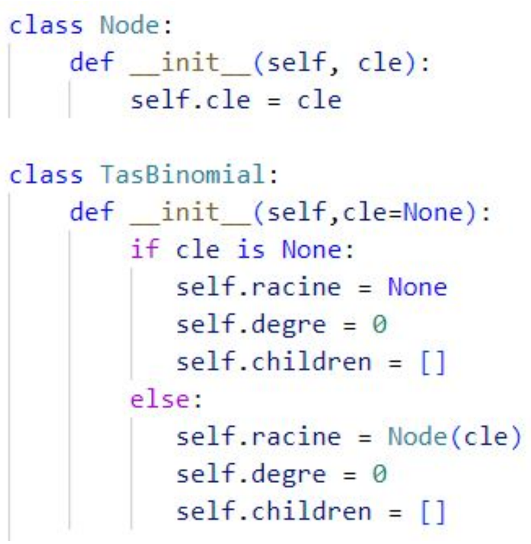
Graphes des structures Tas Min



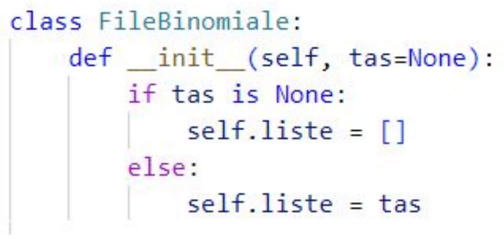




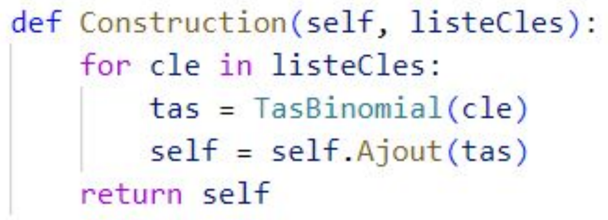
Structure Tas Binomial et File Binomiale



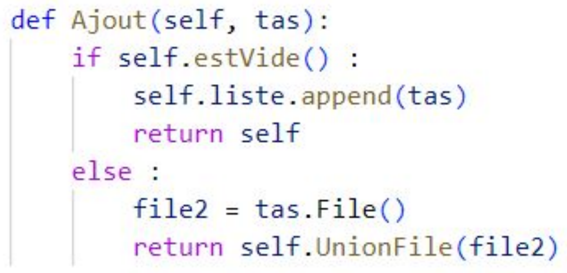
```python
class Node:
    def __init__(self, cle):
        self.cle = cle

class TasBinomial:
    def __init__(self,cle=None):
        if cle is None:
            self.racine = None
            self.degre = 0
            self.children = []
        else:
            self.racine = Node(cle)
            self.degre = 0
            self.children = []
```



```python
class FileBinomiale:
    def __init__(self, tas=None):
        if tas is None:
            self.liste = []
        else:
            self.liste = tas
```

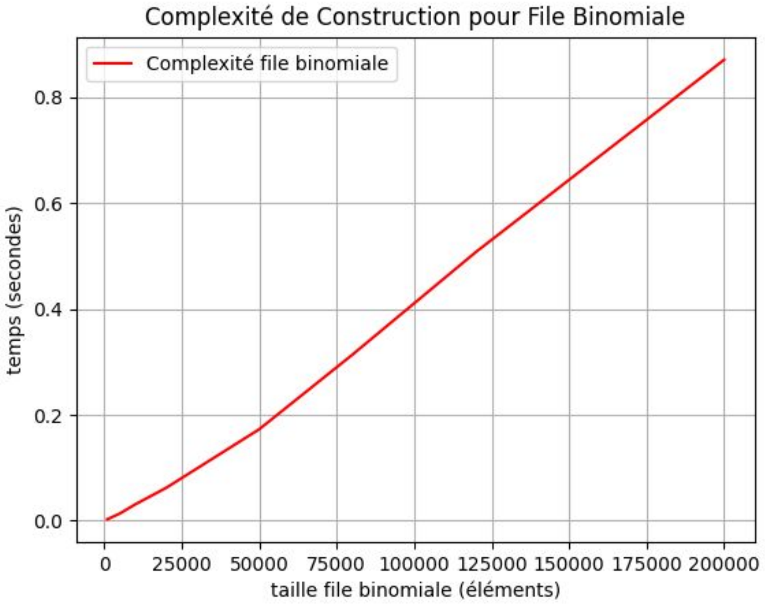


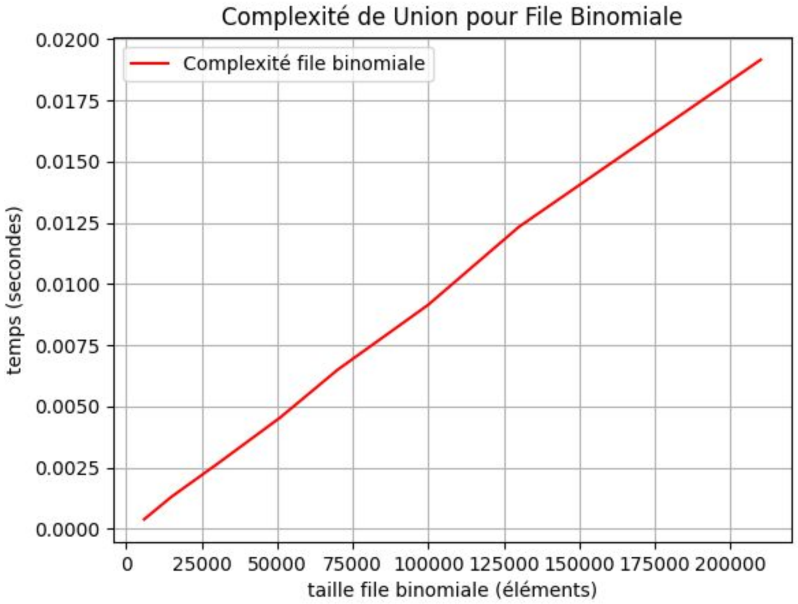
```python
def Construction(self, listeCles):
    for cle in listeCles:
        tas = TasBinomial(cle)
        self = self.Ajout(tas)
    return self
```



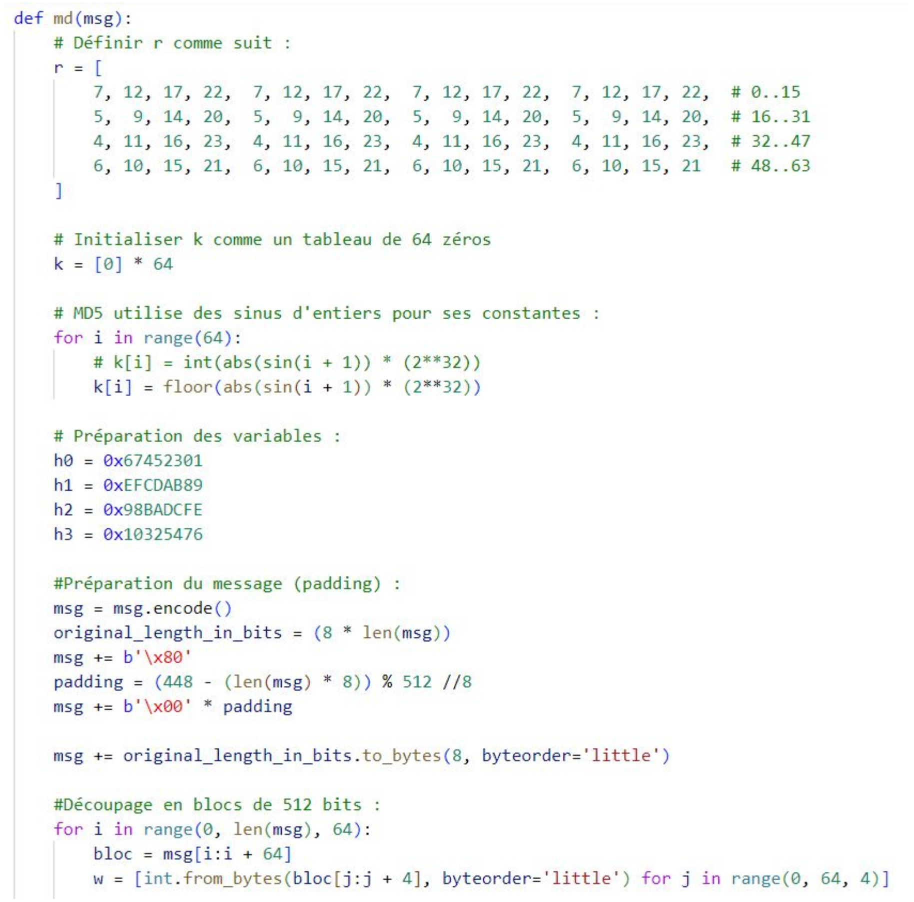
```python
def Ajout(self, tas):
    if self.estVide() :
        self.liste.append(tas)
        return self
    else :
        file2 = tas.File()
        return self.UnionFile(file2)
```

Graphes de la structure File Binomiale

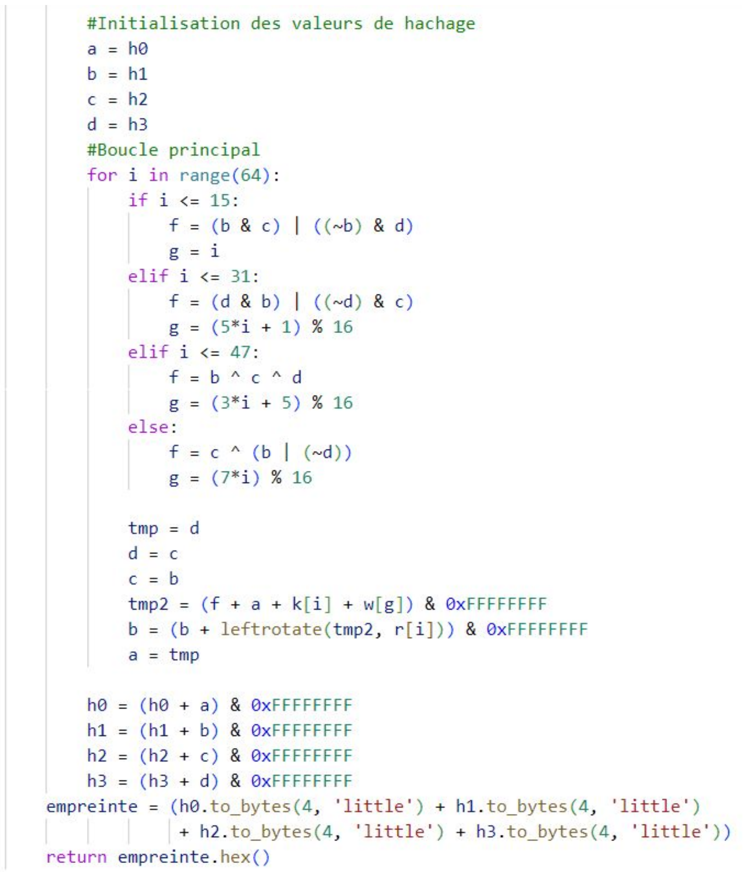




Hachage MD5

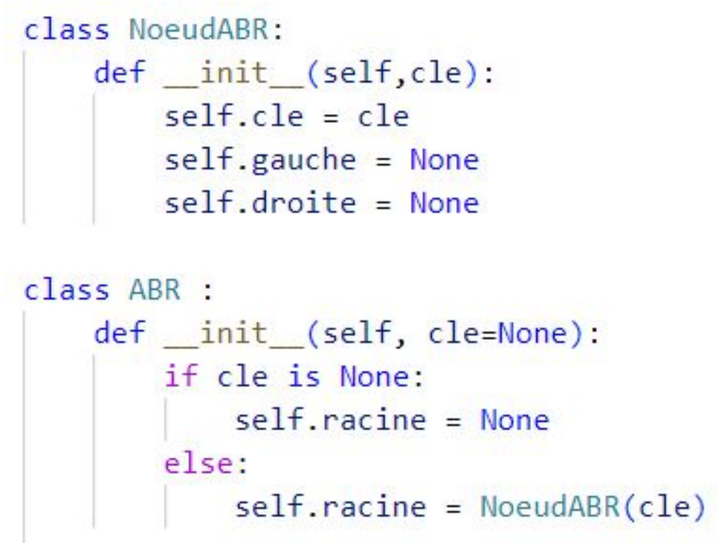


```python
def md(msg):
    # Définir r comme suit :
    r = [
        7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  # 0..15
        5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  # 16..31
        4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  # 32..47
        6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21   # 48..63
    ]

    # Initialiser k comme un tableau de 64 zéros
    k = [0] * 64

    # MD5 utilise des sinus d'entiers pour ses constantes :
    for i in range(64):
        # k[i] = int(abs(sin(i + 1)) * (2**32))
        k[i] = floor(abs(sin(i + 1)) * (2**32))

    # Préparation des variables :
    h0 = 0x67452301
    h1 = 0xEFCDAB89
    h2 = 0x98BADCFE
    h3 = 0x10325476

    #Préparation du message (padding) :
    msg = msg.encode()
    original_length_in_bits = (8 * len(msg))
    msg += b'\x80'
    padding = (448 - (len(msg) * 8)) % 512 //8
    msg += b'\x00' * padding

    msg += original_length_in_bits.to_bytes(8, byteorder='little')

    #Découpage en blocs de 512 bits :
    for i in range(0, len(msg), 64):
        bloc = msg[i:i + 64]
        w = [int.from_bytes(bloc[j:j + 4], byteorder='little') for j in range(0, 64, 4)]
```



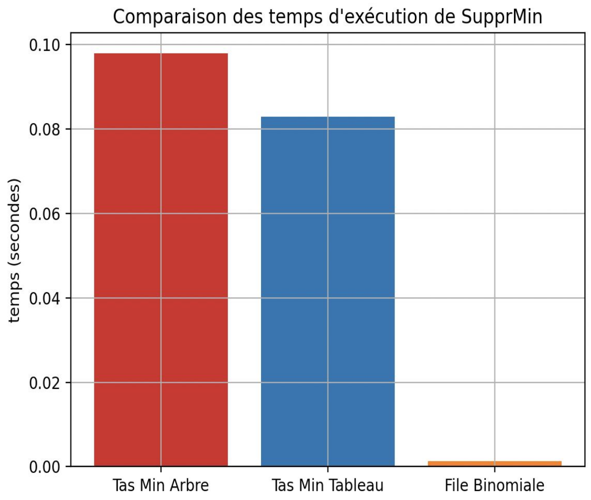
```python
        #Initialisation des valeurs de hachage
        a = h0
        b = h1
        c = h2
        d = h3
        #Boucle principal
        for i in range(64):
            if i <= 15:
                f = (b & c) | ((~b) & d)
                g = i
            elif i <= 31:
                f = (d & b) | ((~d) & c)
                g = (5*i + 1) % 16
            elif i <= 47:
                f = b ^ c ^ d
                g = (3*i + 5) % 16
            else:
                f = c ^ (b | (~d))
                g = (7*i) % 16

            tmp = d
            d = c
            c = b
            tmp2 = (f + a + k[i] + w[g]) & 0xFFFFFFFF
            b = (b + leftrotate(tmp2, r[i])) & 0xFFFFFFFF
            a = tmp

        h0 = (h0 + a) & 0xFFFFFFFF
        h1 = (h1 + b) & 0xFFFFFFFF
        h2 = (h2 + c) & 0xFFFFFFFF
        h3 = (h3 + d) & 0xFFFFFFFF
    empreinte = (h0.to_bytes(4, 'little') + h1.to_bytes(4, 'little')
                 + h2.to_bytes(4, 'little') + h3.to_bytes(4, 'little'))
    return empreinte.hex()
```

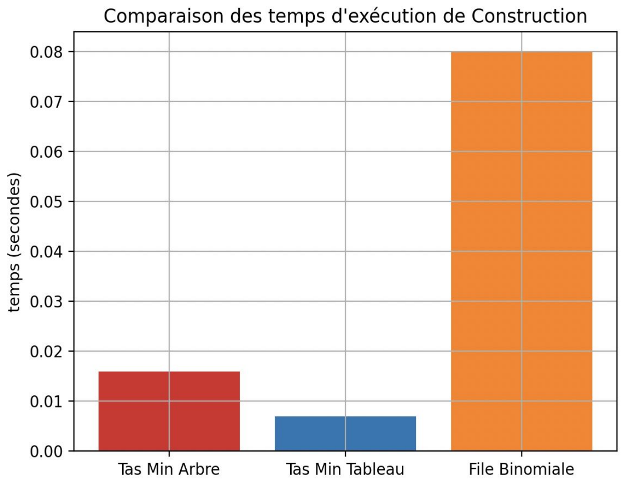
Arbre binaire de recherche

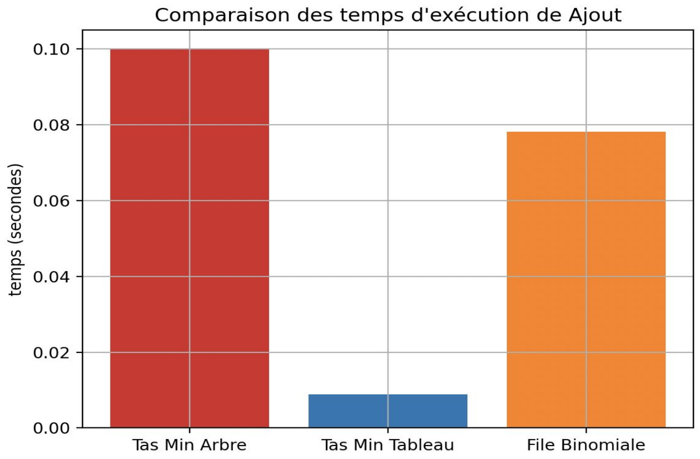


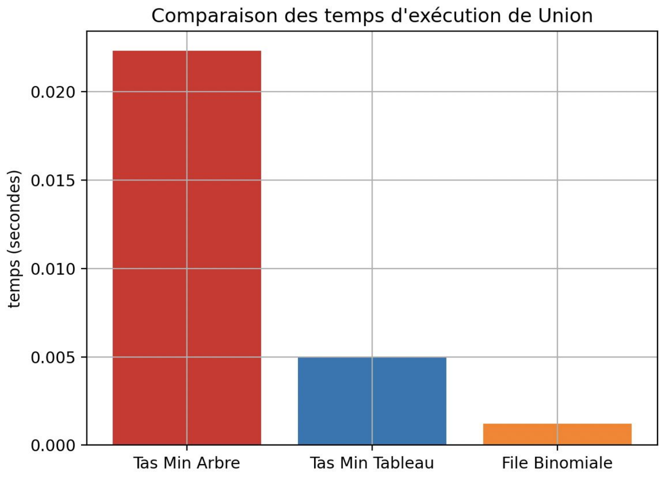
```python
class NoeudABR:
    def __init__(self,cle):
        self.cle = cle
        self.gauche = None
        self.droite = None

class ABR :
    def __init__(self, cle=None):
        if cle is None:
            self.racine = None
        else:
            self.racine = NoeudABR(cle)
```

Étude expérimentale


Comparaison des temps d'exécution de SupprMin
temps (secondes)
Tas Min Arbre
Tas Min Tableau
File Binomiale


Comparaison des temps d'exécution de Construction
temps (secondes)
Tas Min Arbre
Tas Min Tableau
File Binomiale


Comparaison des temps d'exécution de Ajout
temps (secondes)
Tas Min Arbre
Tas Min Tableau
File Binomiale


Comparaison des temps d'exécution de Union
temps (secondes)
Tas Min Arbre
Tas Min Tableau
File Binomiale