Exercices : Recherche d'un motif dans une chaîne de caractères

Prérequis

- Notion de complexité
- Les listes Python
- Structure de données et de contrôle

Objectifs

- Produire des algorithmes pour la recherche d'un motif dans un texte.
- Comparer les différents algorithmes (complexité temporelle)

Exercices :

Pour chaque exercice discuter en fonction des solutions proposées:

- le choix des structures de données
- l'aspect plus ou moins générique du code
- la terminaison des boucles
- l'efficacité en terme de nombre d'instructions effectuées pour obtenir le résultat

Contexte

On représente un brin d'ADN par une chaîne de caractères qui peut contenir quatre bases différents :

- 'A'(Adénine),
- 'C'(Cytosine),
- 'G'(Guanine)
- 'T'(Thymine)

Chaque base possède une base complémentaire avec laquelle elle peut s'associer...

Exo 1 :

Écrire une fonction `brinComplementaire` qui étant donné un brin d'ADN calcule et renvoie son brin complémentaire, le brin complémentaire est calculé avec les règles suivantes : 'A' et 'T' sont complémentaires, et 'C' et 'G' sont complémentaires

Exemple :

- brinComp("A") renvoie "T"
- brinComp("AAGT") renvoie "TTCA".

Exo 2 :

Écrire une fonction `masseMolaire` qui calcule la masse molaire d'une séquence ADN passée en argument. Chaque lettre a une masse donnée :'A'(135 g/mol) ;'T'(126 g/mol) ;'G'(151 g/mol) ;'C'(111 g/mol). La masse totale est la somme des masses des lettres de la séquence.

Exemple :

masseMolaire("AGATC")renvoie (135 + 151 + 135 + 126 + 111) g/mol

Exo 3 :

Écrire une fonction `estADN` qui prend en argument une chaîne de caractères et renvoie `True` si cette chaîne correspond à un brin d'ADN et qui renvoie `False` sinon.

Exemple :

- estADN("TTGAC") et estADN("GCAATAG") renvoient `True`
- estADN("AMOG") et estADN("CaTg") renvoient `False`

Jeux de tests pour évaluer expérimentalement la complexité temporelle

Un module pour évaluer le temps d'exécution d'une fonction en fonction du jeu de données. Attention ce temps est dépendant de la machine et des processus en cours.

In [1]:
```
import time
```

Un exemple d'utilisation

In [4]:
```
debut = time.time()
brinComplementaireNaif("AACT")
print(f"Durée = {time.time() - debut : 1.2e}")
```

```
Durée =  9.75e-05
```

Le problème : Faire varier la taille (n) du jeu de données pour tracer une représentation garphique de $t = f(n)$ avec t le temps d'exécution. Nous allons donc construire un jeu de données avec n classes contenant chacune un nombre de brins fixés dont les lettres seront tirées au hasard. La première classe contient de brins de deux lettres. On construit les brins de la classe suivante en multipliant par deux le nombre de lettres. S'il y a 10 classes les brins de la dernière classe contiennent 2^{10} lettres

In [5]:
```
import random as rd
import matplotlib.pyplot as plt
bases = ['A', 'T', 'C', 'G']
```

In [6]:
```
nombre_brins = 10
def jeux_donnees(classe, nombre_brin):
    """
    Commentaires....
    """
    data = []
    for i in range(1, classe + 1):
        brin_classe = []
        for j in range(nombre_brin):
            brin = ""
            for k in range(2**i):
                brin += bases[rd.randint(0,3)]
            brin_classe.append(brin)
        data.append(brin_classe)
    return data
```

Création du jeux de données

In [7]:
```
data = jeux_donnees(4, 2)
print(data)
```

```
[['GA', 'CC'], ['CAAT', 'ACCG'], ['GTCTAAAG', 'TGAGATAC'], ['GAAGGGGCTATGACGC
', 'GGAGCTAAGTTGACAG']]
```

Question : Décrire la structure obtenue pour data

Nous allons maintenant créer un jeux de données beaucoup plus conséquent **Attention il faut patienter un peu et surtout ne pas assayer de l'afficher**.

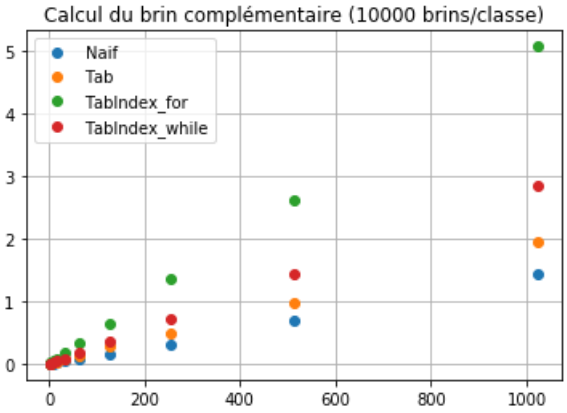
In [8]:
```
data = jeux_donnees(10, 10000)
```

Utilisation du jeu de données

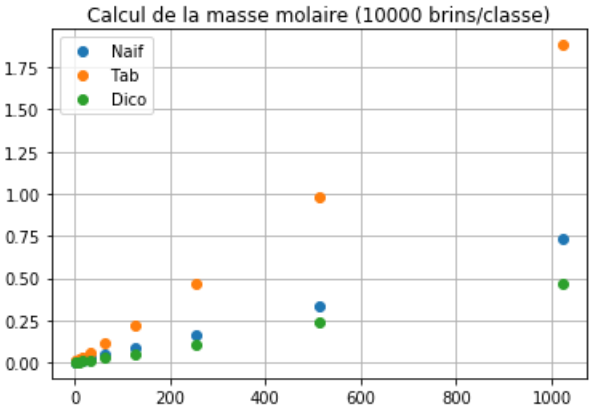
In [ ]:
```
def exploiter(f, jeux_donnees):

    temps       = []
    taille_brin = []
    for elt in jeux_donnees:
        taille_brin.append(len(elt[0]))
        debut = time.time()
        for brin in elt:
            f(brin)
        temps.append(time.time() - debut)
    return taille_brin, temps
```

In [9]:
```
plt.plot(*exploiter(brinComplementaireNaif, data), "o", label='Naif')
plt.plot(*exploiter(brinComplementaireTab, data), "o", label='Tab')
plt.plot(*exploiter(brinComplementaireTabIndex, data), "o", label='TabIndex_fo
r')
plt.plot(*exploiter(brinComplementaireTabIndex_v2, data), "o", label='TabIndex
_while')
plt.grid()
plt.legend(loc='best')
plt.title("Calcul du brin complémentaire (10000 brins/classe)")
plt.show()
```



In [10]:
```
plt.plot(*exploiter(masseMolaireNaif, data), "o", label='Naif')
plt.plot(*exploiter(masseMolaireTab, data), "o", label='Tab')
plt.plot(*exploiter(masseMolaireDico, data), "o", label='Dico')
plt.grid()
plt.legend(loc='best')
plt.title("Calcul de la masse molaire (10000 brins/classe)")
plt.show()
```



Éléments de Correction

- Plusieurs solutions proposées pour un même exercice permettant de discuter :
  - de complexité peut-être testée expérimentalement avec des compteurs (ajout un compteur)
  - de code plus ou moins bien écrit
  - du remplacement d'une boucle for par une boucle while (avantage : on s'arrete au bon moment mais au détriment de bugs à la terminaison)
  - Montrer que asymptotiquement on est en $0(n)$ pour les algos proposés

In [ ]:

```
############################
####   Solutions Exo 1  #####
############################

def brinComplementaireNaif(brin):
    compteur = 0
    tab_brin    = []
    taille_brin = len(brin)
    for i in range(taille_brin):
        compteur += 1
        base = brin[i]
        if base   == 'A':
            tab_brin.append('T')
        elif base == 'T':
            tab_brin.append('A')
        elif base == 'C':
            tab_brin.append('G')
        else:
            tab_brin.append('C')
    return "".join(tab_brin), compteur

def brinComplementaireTab(brin):
    base_complement = ['A', 'T', 'A', 'C', 'G', 'C']
    tab_brin        = []
    for base in brin:
        index = base_complement.index(base)
        tab_brin.append(base_complement[index + 1])
    return "".join(tab_brin)

def brinComplementaireTabIndex(brin):
    base_complement  = [['A', 'T'], ['T', 'A'], ['C', 'G'], ['G', 'C']]
    taille_base_comp = len(base_complement)
    tab_brin         = []
    for base in brin:
        for i in range(taille_base_comp):
            cle = base_complement[i][0]
            if cle == base:
                valeur = base_complement[i][1]
                tab_brin.append(valeur)
    return "".join(tab_brin)

def brinComplementaireTabIndex_v2(brin):
    base_complement  = [['A', 'T'], ['T', 'A'], ['C', 'G'], ['G', 'C']]
    taille_base_comp = len(base_complement)
    tab_brin         = []
    for base in brin:
        cle = ""
        i = -1
        while cle != base:
            i += 1
            cle = base_complement[i][0]
        valeur = base_complement[i][1]
        tab_brin.append(valeur)
    return "".join(tab_brin)
```

In [3]:
```
############################
####   Solutions Exo 2  #####
############################

def masseMolaireNaif(brin):
    masse_molaire = 0
    for base in brin:
        if base == 'A':
            masse_molaire += 135
        elif base == 'T':
            masse_molaire += 126
        elif base == 'G':
            masse_molaire += 151
        else:
            masse_molaire += 111
    return masse_molaire

def masseMolaireTab(brin):
    masse_molaire   = 0
    taille_brin     = len(brin)
    bases           = ['A', 'T', 'G', 'C']
    masses_molaires = [135, 126, 151, 111]
    for i in range(taille_brin):
        lettre = brin[i]
        index  = bases.index(lettre)
        masse_molaire += masses_molaires[index]
    return masse_molaire

def masseMolaireDico(brin):
    masse_molaire = 0
    dictionnaire  = {'A' : 135,'T': 126,'G': 151,'C': 111}
    for base in brin:
        masse_molaire += dictionnaire[base]
    return masse_molaire

############################
####   Solutions Exo 3 #####
############################
# À faire avec les algos du calepin Algorithme_KMP
```

In [ ]:
```
print(brinComplementaireNaif("AACG"))
print(brinComplementaireTab("AACG"))
print(brinComplementaireTabIndex("AACG"))
print(brinComplementaireTabIndex_v2("AACG"))
```

In [ ]:
```
brin = "AGATC"
print(masseMolaireNaif(brin))
print(masseMolaireTab(brin))
print(masseMolaireDico(brin))
```