Projet bloc 2 : Algorithmique et coût en opérations

1. Place dans la progression

Pré requis :

- Affectation, boucles for et while, instruction conditionnelle
- Manipuler des listes indexées
- Utilisation de bibliothèques

Notions du programme abordées :

- Thème algorithmique :
  Notion de coût linéaire et quadratique
  Parcours séquentiel d'un tableau
- Traitement de données en tables :
  Importer des données depuis un fichier csv

Objectif :

- Prendre en compte l'efficacité d'un algorithme et coder dans ce sens afin d'obtenir le « meilleur » algorithme

2. Introduction

Un ***algorithme*** est une suite finie, claire et ordonnée d'instructions utilisée pour effectuer une tâche.

- Recette de cuisine
- Monter un meuble en kit
- Donner un itinéraire
- ……

Activité 1
Chaque algorithme permet d'effectuer une tâche.
Pour chaque tâche existe-t-il un seul algorithme ? Donner un ou des exemples.

.....

.....

.....

Un site Web propose 3 itinéraires pour se rendre de Bordeaux à Paris.
Que peut signifier « le meilleur itinéraire » ?

- .....
- .....
- .....
- .....

Conclusion :
Pour comparer des algorithmes **qui effectuent la même tâche**, il faut un critère pour en mesurer l'efficacité.

Activité 2

Soient les fonctions : $f_4(x) = 1 + x + x^2 + x^3 + x^4$ $g4(x)=(x^5 - 1)/(x - 1)$

Montrer que $f_4(x) = g_4(x)$

.....

Décomposer $f_4(7)$ en une suite d'opérations arithmétiques élémentaires (+,-,/,*).

.....

$f_4(7) = 1 + 7 + 7 * 7 + 7 * 7 * 7 + 7 * 7 * 7 * 7$

Combien d'opérations élémentaires sont nécessaires pour calculer $f_4(7)$? .....

10 opérations

Combien d'opérations élémentaires sont nécessaires pour calculer $g_4(7)$? .....

7 opérations

Conclusion

Quel critère de performance peut-on énoncer pour comparer l'efficacité de ces 2 algorithmes ?

.....

Activité 3

On étudie le nombre d'opérations nécessaires au calcul de $f_n(7)$ et $g_n(7)$.

Compléter le tableau suivant en utilisant les fonctions Operations_fn et Operations_gn du programme Polynome.py.

La fonction Operations_fn(5) donne le nombre d'opérations nécessaires au calcul de $f_5(7)$.

n	5	10	15	20	25
Nbre opérations fn	15	55	120	210	325
Nbre opérations gn	8	13	18	23	28

Afficher 2 courbes, en utilisant la fonction Trace(c1,c2) qui montre le nombre d'opérations de chaque algorithme en fonction de n.

c1 est une liste contenant le nombre d'opérations nécessaires au calcul de $f_5(7)$, $f_{10}(7)$, …, $f_{25}(7)$.

c2 est une liste contenant le nombre d'opérations nécessaires au calcul de $g_5(7)$, $g_{10}(7)$, …, $g_{25}(7)$.

A quel type de courbes semblent appartenir ces 2 courbes ?

……

Conclusion :

L'algorithme Poly_f a un coût **linéaire**. Si la taille des données est multipliée par 10, le nombre d'opérations est multiplié par 10.

L'algorithme Poly_g a un coût **quadratique**. Si la taille des données est multipliée par 10, le nombre d'opérations est multiplié par 100.

Revenir aux programmes

Activité 4

Soit la fonction $h_4(x)= (((x + 1) * x + 1) * x + 1) * x + 1$, On admet que $h_4(x) = f_4(x)$

Ecrire un programme, en utilisant une boucle for, retournant le résultat de $h_4(x)$ pour toute valeur de x

Programme algo_Horner

Pour aller plus loin

Afin de généraliser, $h_n(x) = (((x + 1) * x + 1)$ ……..)* $x + 1) * x + 1$, écrire le nouveau programme.

n fois

Programme algo_Horner2

Conclusion :

L'algorithme d'Horner a un coût ……. linéaire

3. Cours

Algorithme et complexité
On dispose d'un algorithme A effectuant un traitement sur des données D dont on peut mesurer la taille.

On veut pouvoir *estimer* le temps de calcul nécessaire à A pour traiter une donnée de taille n, que l'on appelle la *complexité en temps* dans le but de pouvoir prédire le comportement général de A notamment quand n devient très grand.

Le temps effectif pris par une implémentation d'un algorithme dépend fortement du système sur lequel il tourne : architecture matérielle, système d'exploitation, langage de programmation. La mesure de la complexité d'un algorithme se veut indépendante de ces détails d'implémentation (même si en pratique ils sont fondamentaux). Elle se mesure à un plus haut niveau d'abstraction, et se définit comme le *nombre d'opérations élémentaires effectuées par l'algorithme*. Les instructions élémentaires couramment prises en compte sont :

- les opérations arithmétiques,
- la lecture ou l'affectation d'une variable (incluant l'affectation d'un compteur de boucle) ou d'un élément d'un tableau (ou liste ou dictionnaire),
- un test booléen.

Différents cas de complexité de l'algorithme

- Si la complexité est proportionnelle à la taille n des données, on dit que l'algorithme est de **complexité linéaire.**
- Si la complexité est proportionnelle à n^2, on dit que l'algorithme est de **complexité quadratique.**

Remarques :

- On ne s'intéresse pas au nombre exact d'instructions élémentaires, mais à son ordre de grandeur,
- Pour des données de grande taille, on privilégie les algorithmes linéaires plutôt que quadratiques.

4. Exercices d'application

Jusqu'à présent nous avons comptabilisé le coût en opération de certains algorithmes mais n'avons pas comptabilisé l'affectation en mémoire de variables ou les tests.

Exercice 1
Voici une fonction calculant votre moyenne sachant que vous avez eu trois notes (n1, n2, n3) avec trois coefficients différents (a, b, c) et indiquant votre moyenne et si vous dépassez 10.

Ex : J'ai obtenu un 10 (coef 4), un 12 (coef 3) et un 15 coef 2
Print (moyenne(10,12,15, 4,3,2)) donne la moyenne coefficientée.

```
def moyenne(n1, n2, n3, a, b, c) :
    x1 = n1*a
    x2 = n2*b
    x3 = n3*c
    somme_coeff = a + b + c
    moyenne = (x1 + x2 + x3 )/somme_coeff
    if moyenne > 10 :
        return (moyenne)
```

Quantifier le coût de cet algorithme en justifiant : 14
Nombre d'opérations élémentaires : 8
Nombre d'affectations : 5
Nombre de tests : 1

Exercice 2 : Travail sur un algorithme de traitement d'image
Vous disposez du programme pomme.py.

a. Que fait ce programme ?

.....

b. Précisez l'utilité des deux boucles for ?

.....

c. Donner un ordre de grandeur du coût de ce programme en justifiant.

.....

d. Si on traite une image de taille 600 * 600, par quel facteur est multiplié le coût ?

.....

5. Prolongement

Les ordinateurs sont utilisés pour travailler sur de très nombreuses données comme dans le cas de la recherche d'un gêne de 1 000 paires de bases dans un fragment (4 millions de paires de bases) d'un génome humain contenant au total plus de 3 400 Milliards de paires de bases.

Exercice 1 : Recherche d'un motif de taille k dans un texte (sans césure) de longueur n.
On cherche la présence du motif « NSI » k = 3 dans le fichier texte.csv, n = 486.

a. Proposer en langage naturel, un algorithme de recherche du motif « NSI ».

b. Vous disposez du programme recherche_NSI.py qui lit le fichier texte.csv.
Compléter ce programme pour qu'il parcourt toutes les lignes et qu'il affiche le nombre de fois et la position du mot (on parle d'occurrence)
On pourra utiliser la fonction len().
Ne pas utiliser les fonctions index ou in.
Ne pas utiliser les slices.

c. Estimer le coût en nombre d'instructions élémentaires en fonction de n et de k.

d. Quel est le type de complexité de cet algorithme ?

e. Si on travaille sur un motif génétique de taille 1000 que l'on veut trouver dans un fragment de 4 millions, estimer le coût .

f. Pour aller plus loin : Modifier votre programme pour chercher une chaîne de k caractères

Conclusion :
Algorithme naïf : Rechercher un motif de taille k dans un texte de taille n a **un coût de l'ordre de k * n** (type quadratique)

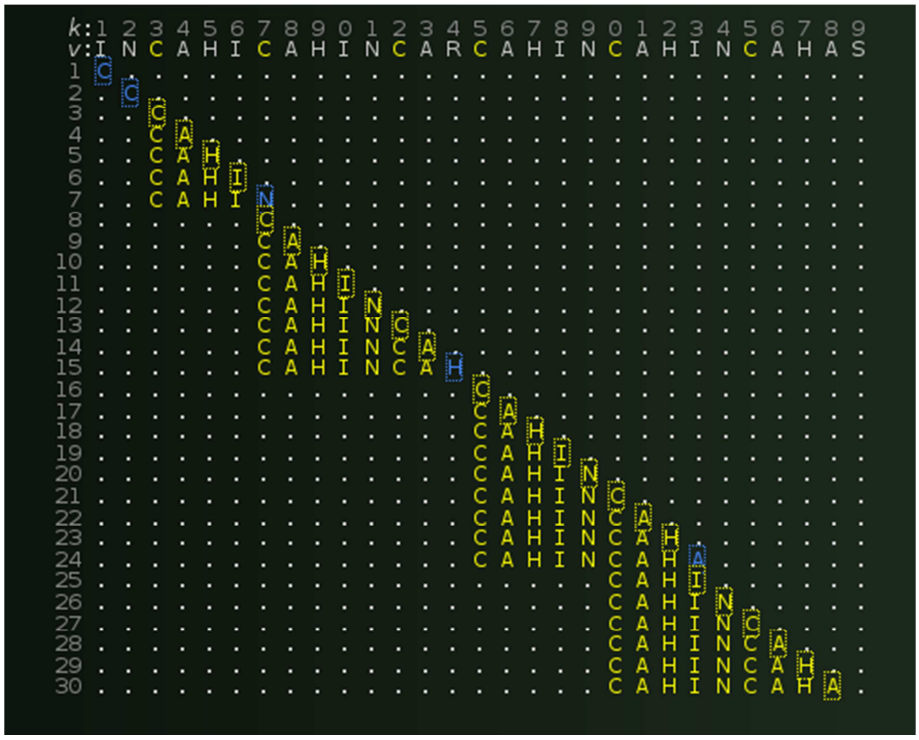
Exercice 2 :

Montrer le programme permettant de trouver toute suite de caractères (répétitifs) de longueur k dans un texte de longueur n (n > k)

Approfondissement :

Cet algorithme est particulièrement utile dans la recherche de motif répétitif présent dans l'ADN qui est constitué de 4 « lettres » correspondant aux nucléotides. Ainsi la recherche d'un motif correspondant à un fragment de gêne permet de savoir si le « gène recherché » est présent ou non dans le génome.

Par exemple le motif est CAHINCAHA et le texte est en v.



Dans un premier temps :

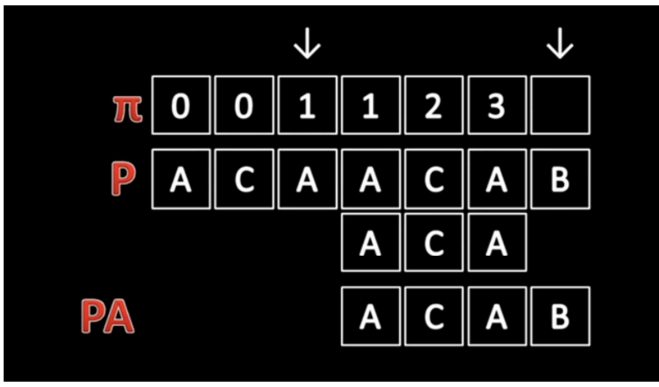
```python
def kmp(pattern, text):
    """

    """

    # 1) Construct the failure array
    failure = get_failure_array(pattern)

    # 2) Step through text searching for pattern
    i, j = 0, 0  # index into text, pattern
    while i < len(text):
        if pattern[j] == text[i]:
            if j == (len(pattern) - 1):
                return True
            j += 1

        # if this is a prefix in our pattern
        # just go back far enough to continue
        elif j > 0:
            j = failure[j - 1]
            continue
        i += 1
    return False
```

Mais dans le motif il y donc on a des répétitions, donc on crée un tableau failure de ce type :

	1	2	3	4	5	6	7	8	9
u	C	A	H	I	N	C	A	H	A
B	0	0	0	0	0	1	2	3	0

```python
def get_failure_array(pattern):
    """
    Calculates the new index we should go to if we fail a comparison
    :param pattern:
    :return:
    """
    failure = [0]
    i = 0
    j = 1
    while j < len(pattern):
        if pattern[i] == pattern[j]:
            i += 1
        elif i > 0:
            i = failure[i-1]
            continue
        j += 1
        failure.append(i)
    return failure
```

Pour comprendre la création de cette listefailure : https://www.youtube.com/watch?v=HaAu5ZGj6fc



Pour aller plus loin sur la compréhension de l'algo :
http://zanotti.univ-tln.fr/ALGO/I31/KMP.html
https://fr.wikipedia.org/wiki/Algorithme_de_Knuth-Morris-Pratt

Complexité k

On crée une liste d'entiers dans la fonction get_failure_array permettant de voir combien de lettres du suffixe sont présents et se répètent.
https://fr.wikipedia.org/wiki/Algorithme_de_Knuth-Morris-Pratt

Dans le programme, la recherche du motif dans text va nécessiter une comparaison lettre à lettre avec association du motif répétitif.

Complexité n

Dans la fonction kmp on balaie le texte et lorsque le motif est présent il envoie vrai,
sinon il retourne en se déplaçant de - 1 pour vérifier que le motif ne reprend pas au j - 1.

Conclusion :

L'algorithme de Knuth-Morris-Pratt **un coût de l'ordre de k + n** (type linéaire)

Synthèse