Contrôle Continu n°2

Analyse d'Algorithmes et Programmation

20 avril 2018

Durée : 2 heures et 30 minutes

Les seuls documents autorisés sont les notes de cours.
Toute recherche sur Internet entraînera une note de 0.
Le barème est indicatif.

Consignes (il est toujours utile de les lire)

L'ensemble des points obtenus donnera la note finale. Toute réponse partielle sera corrigée et rapportera — éventuellement — des points. Cependant, si votre programme ne fonctionne pas (pour n'importe quelle raison), il serait appréciable que cela soit précisé. Essayez autant que possible de vous passer des fonctions `Python` telles que `min` ou `index`, etc. Tout abus sera évidemment puni.

Vos réponses sont à envoyer **par courriel** à l'adresse `alexandre.gelin@uvsq.fr`, sous la forme d'un fichier d'extension `.ipynb` ou `.py` et dont l'intitulé sera votre prénom ou votre nom. Le courriel doit avoir pour sujet "`CC2`" et contenir **un unique fichier** au format "`alex.ipynb`" (ici, il faut remplacer `alex` par **votre** nom). Toute déviance d'une de ses règles sera durement pénalisée.

Je dois recevoir le courriel **avant la fin des 2 heures et 30 minutes** de composition. Tout retard sera aussi pris en compte dans la notation.

Exercice 1 — Algorithmes de tri [8 points]

On s'intéresse dans cet exercice à différentes manières de trier un tableau. Afin de pouvoir reproduire les expériences, on utilise un ensemble de 100 tableaux aléatoires mais reconstructibles. Pour cela, on utilise la fonction `random.seed(s)` qui initialise le générateur aléatoire avec une graine `s`. Ainsi, bien que nos tableaux soient aléatoires, il est toujours possible de reconstruire le même tableau. On construit nos tableaux grâce à la fonction `initTabs()`. Les autres fonctions données en annexe sont là pour vous aider alors étudiez-les bien.

Question 1.1 [2 points]
Implémenter l'algorithme de tri par insertion et tester ses performances sur l'ensemble des tableaux.

Question 1.2 [2 points]
Implémenter l'algorithme de tri fusion et tester ses performances sur l'ensemble des tableaux.

Question 1.3 [3 points]
La meilleure façon d'utiliser le tri fusion est de l'appliquer *jusqu'à un certain point* puis d'appliquer un tri plus simple pour les petites instances, comme le tri par insertion par exemple.

Écrire une procédure `triMixte(tab,p,r,b)` qui commence par appliquer le tri fusion puis le tri par insertion dès que la longueur du tableau `tab` est inférieure à `b`. Pour simplifier, on choisira pour `b` une puissance de 2. Tester ses performances sur l'ensemble des tableaux.

Indication : Il sera peut-être nécessaire de réécrire le tri par insertion en utilisant `p` et `r` en entrée.

Question 1.4 [1 point]
Trouver le `b` optimal qui minimise le temps de calcul pour les tableaux construits.

Exercice 2 — Arbre de Fibonacci [10 points]

Question 2.1 [2 points]

Écrire un algorithme qui calcule les n premiers termes de la suite de Fibonacci et les stocke dans un tableau de taille n. Pour rappel, cette suite est définie par $F_0 = 0$, $F_1 = 1$ et $\forall n \geq 2, \quad F_n = F_{n-1} + F_{n-2}$.

Construire le tableau des 32 premiers termes puis supprimer le terme apparaissant en double pour obtenir un tableau de 31 éléments distincts.

Question 2.2 [2 points]

En utilisant la structure de `Node` donnée, écrire une fonction `inserer(self,key)` qui insère le nœud (à créer) de clé `key` dans l'**arbre binaire de recherche** dont la racine est `self`.

Rappel : Pour créer un arbre, il suffit de créer un premier nœud puis de considérer ce nœud comme la racine.

Question 2.3 [2 points]

En choisissant **soigneusement** l'ordre dans lequel vous insérez les éléments, construisez un arbre dont les 31 nœuds sont les 31 termes **distincts** du tableau construit à la question 2.1 **ET** dont la hauteur est minimale. Vous pourrez vous servir de la fonction `recherche_modifiee` pour vérifier la hauteur de votre arbre.

Question 2.4 [2 points]

Afin de vérifier que votre arbre construit est bien un arbre binaire de recherche, comparer le résultat retourné par un certain parcours (qu'il faudra implémenter) de l'arbre avec le tableau initial.

Question 2.5 [2 points]

Écrire un programme qui calcule la hauteur totale de votre arbre. On pourra encore une fois se servir de la fonction `recherche_modifiee`.

Exercice 3 — Pivot de Gauss [8 points]

Question 3.1 [8 points]

Implémenter le pivot de Gauss dans le cadre de la résolution d'un système linéaire $Ax = b$.

Vérifier votre programme sur la matrice A et le vecteur b donnés :

$$\begin{pmatrix} 2 & 5 & 13 & 29 & 47 \\ 3 & 11 & 23 & 43 & 67 \\ 7 & 19 & 41 & 61 & 79 \\ 17 & 37 & 59 & 73 & 89 \\ 31 & 53 & 71 & 83 & 97 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 19 \\ 38 \\ 74 \\ 114 \\ 144 \end{pmatrix}.$$

Il est judicieux de regarder la façon dont sont construites ces entrées avant de se lancer dans la résolution du problème. La fonction `copy.deepcopy` donnée en annexe permet de recopier `A0` et `b0` en `A` et `b` afin de modifier ces derniers sans toucher aux entrées du problème. Il suffit de mettre cette ligne au début du programme.

Indication : On pourra utiliser des sous-routines telles que

- `choixPivot` qui renvoie la colonne du pivot (choisi de valeur absolue maximale);
- `echangerLignes` qui échange deux lignes de la matrice;
- `transvection` qui réalise l'opération $L_k \leftarrow L_k - cL_i$.

EXERCICE 1

In [ ]:
```
import random,time

def initTabs():
    Tabs=[]
    for i in range(100):
        random.seed(i)
        Tabs.append(random.sample(range(0,100000),1024))
    return Tabs

t0=time.time()
Tabs=initTabs()
print(time.time()-t0)
for tab in Tabs:
    print(tab[0], end=" ")
print()

def isSorted(tab):
    n=len(tab)-1
    while n>0 and tab[n]>=tab[n-1]:
        n-=1
    if n==0:
        return "T" #True
    return "F" #False

for tabs in Tabs:
    print(isSorted(tab), end="")
print()
```

EXERCICE 2

In [ ]:
```
class Node:
    def __init__(self,key):
        self.left = None
        self.right = None
        self.key = key

    def inserer(self,key):
        #TO BE COMPLETED

    def parcours(self):
        #TO BE COMPLETED

    def recherche_modifiee(self,key,h=0):
        if key<self.key:
            if self.left:
                return self.left.recherche_modifiee(key,h+1)
            else:
                return None
        elif key>self.key:
            if self.right:
                return self.right.recherche_modifiee(key,h+1)
            else:
                return None
        else:
            return h
```

EXERCICE 3

In [ ]:
```
A=[[2,5,13,29,47],[3,11,23,43,67],[7,19,41,61,79],[17,37,59,73,89],[31,53,71
,83,97]]
b=[19,38,74,114,144]
#A,b=[copy.deepcopy(v) for v in [A0,b0]]
```