Devoir d'informatique 3

D.Malka – MPSI 2016-2017 – Lycée Saint-Exupéry

13.01.2017

Durée de l'épreuve : 2h00.

L'usage de la calculatrice n'est pas autorisé.

L'énoncé de ce devoir comporte 3 pages.

- *Si, au cours de l'épreuve, vous repérez ce qui vous semble être une erreur d'énoncé, signalez le sur votre copie et poursuivez votre composition en expliquant les raisons des initiatives que vous êtes amené à prendre.*
- *Il ne faudra pas hésiter à formuler des commentaires. Le barème tiendra compte de ces initiatives ainsi que des qualités de rédaction de la copie.*
- *La numérotation des exercices doit être respectée. Les résultats doivent être systématiquement encadrés.*
- *Les pages doivent être numérotées de la façon suivante : n°page courante/nombre total de pages.*

Données pour tous les exercices.

On rappelle que :
- La méthode `append()` ajoute l'argument en queue de tableau. Appel : `l.append(elt)`.
- La fonction `len()` renvoie un entier égal à la longueur du tableau passé en argument. Appel : `len(L)`.
- La fonction `float()` convertit l'argument en flottant. Appel `float(n)`.
- La fonction `int()` convertit l'argument en un entier. Si l'argument est un flottant, la fonction `int()` renvoie la partie entière de ce nombre. Appel `int(x)`.
- La fonction `range(a,b,p)` génère la liste suivante : $[a, a+p, a+2p, \dots, a+jp, \dots, b-p]$. L'appel `range(a,b)` est équivalent à `range(a,b,1)` et l'appel `range(b)` est équivalent à `range(0,b,1)`.
- La fonction `linspace(a,b,N)` génère la liste suivante : $[a, a+p, a+2p, \dots, a+jp, \dots, a+(N-1)p]$ avec $p = \frac{b-a}{N}$.
- Slicing : l'expression `l[i:j]` renvoie la sous-liste comprenant les éléments d'indice i inclus à j exclu de la liste `l`.

Exercice 1 – Exponentiation naïve

On redonne l'algorithme d'exponentation naïve de k par un entier naturel n.

1. Montrer que le programme termine.
2. Montrer que le programme est correcte.
3. Evaluer sa complexité en temps.

```
1 def expo(k,n):
2     p=1
3     c=n
4     while c>0:
5         p=p*k
6         c=c-1
7     return p
```

Figure 1 – Algorithme 1

Exercice 2 – Tranche de somme minimale d'un tableau

Soit T un tableau d'entiers relatifs. La tranche de somme minimale de T est le sous-tableau de T dont la somme des éléments est minimale. Par exemple, la tranche de somme minimale du tableau $[12, -4, -5, 2, -1]$ est $[-4, -5]$ (la somme vaut alors -9). Les tranches $[-5, 2, -1]$ ou encore $[12, -4]$ ne sont pas de somme minimale. Il arrive qu'il existe plusieurs tranches de somme minimale au sein d'un tableau.

1. Déterminer la tranche de somme minimale du tableau $[-5, -2, 9, -4, -4, 3]$.
2. On propose, figure 2, un premier algorithme naïf déterminant la tranche de somme minimale d'un tableau.

```
1  def tranche_min(T):
2      n=len(T)
3      g=0;d=0
4      s_min=T[0]
5      for i in range(n):
6          for j in range(i,n):
7              s=somme(T,i,j)
8              if s<s_min:
9                  s_min=s;g=i;d=j
10     return T[g:d+1],s_min
```

FIGURE 2 – Algorithme 2

2.1 Ecrire la fonction `somme` qui calcule et renvoie la somme des éléments d'indices i à j inclus d'un tableau T passé en arguments. Que vaut la complexité en temps de cette fonction ?

2.2 Expliquer de manière claire et concise le principe de l'algorithme fig.2.

2.3 Evaluer sa complexité.

3. On propose, figure 3, un second algorithme.

```
1  def tranche_min_2(T):
2      n=len(T)
3      g=0;d=0;
4      s_min=T[0]
5      k=0;j=0;
6      s=0
7      while j<n:
8          if s+T[j]<T[j]:
9              s=s+T[j]
10         else:
11             s=T[j]
12             k=j
13         if s<s_min:
14             g=k
15             d=j
16             s_min=s
17         j=j+1
18     return T[g:d+1],s_min
```

FIGURE 3 – Algorithme 3

3.1 Appliquer l'algorithme fig.3 au tableau $[-5, -2, 9, -4, -4, 3]$. On représentera en mémoire les sous-tableaux `T[k:j+1]` et `T[g:d+1]` en plus des variables.

3.2 Montrer que l'algorithme est correct. On pourra montrer que les propositions suivantes :
Le sous tableau `T[k:j+1]` est la tranche finissant en `j` de somme minimale.
Le sous tableau `T[g:d+1]` est la tranche de somme minimale du sous tableau `T[0:j+1]`.
sont des invariants de boucle.

3.3 Les deux algorithmes précédents ont été appliqués à des mêmes tableaux de différentes tailles n sur une même machine. La durée de traitement a été mesurée. Commenter et expliquer précisément les résultats.

n	10	100	1000	3000
algo 2	$81\,\mu s$	$18\,ms$	$13\,s$	$384\,s$
algo 3	$6\,\mu s$	$30\,\mu s$	$250\,\mu s$	$846\,\mu s$

Exercice 3 – Suite ultimement périodique

On considère un système dynamique discret, c'est-à-dire une suite d'éléments d'un ensemble E définie par le relation $x_{n+1} = f(x_n)$ dont $x_0 \in E$ est le premier terme et $f : E \to E$. On dit qu'une telle suite d'élément est ultimement périodique de période p s'il existe un entier N tel que $\forall\, n \geq N$, $x_{n+p} = x_n$.

1. Dans un premier temps, on suppose que la suite est périodique c'est-à-dire que $N = 0$. Ecrire une fonction qui détermine et renvoie la période p de la suite à partir de la fonction f et du premier terme x_0.

2. A présent, la suite est seulement ultimement périodique. L'algorithme de Floyd que nous étudions :
   — prend en entrée la fonction $f : E \to E$ et l'élément $x_0 \in E$ définissant une suite récurrente ultimement périodique ;
   — renvoie un entier t tel que $x_{2t} = x_t$.

```
1 def Floyd(f,x0):
2     t=1
3     x=f(x0);y=f(f(x0))
4     while y!=x:
5         x=f(x)
6         y=f(f(y))
7         t=t+1
8     return t
```

2.1 Si f et x_0 définissent la suite d'éléments $x_0, x_1, x_2 \ldots$, quelles sont les éléments contenus dans les variables x et y à la $(t-1)^{ieme}$ itération ?

2.2 Montrer alors que si la fonction f définit une suite d'éléments ultimement périodique, l'algorithme termine. **On n'utilisera pas pas la méthode du variant de boucle.**

2.3 Montrer que la valeur renvoyée est alors un multiple de la période p de la suite.

2.4 L'algorithme de Floyd peut-il démontrer l'ultime périodicité d'une suite ?