Lycée La Prat's
Classe de PTSI

Jeudi 16 janvier 2014

Épreuve d'Informatique 2

Correction

Exercice 1 (Cours – 8 points)

1) Comme ce n'était pas précisé par l'énoncé, on pouvait donner l'algorithme qui répond par un booléen ou celui qui répond par un indice.

`http://www.normalesup.org/~dconduche/prepas/PTSI/AlgoClassiques/RechercheMot_Texte.py`

2) `http://www.normalesup.org/~dconduche/prepas/PTSI/AlgoClassiques/Dichotomie_fonction.py`

Exercice 2 (Exponentiation rapide – 6 points)

1) La fonction retourne le contenu de la variable locale `res`.

Comme `x` est un entier, elle est de type entier

2) Cas $x = 3$ et $n = 3$:

`x`	`n`	`res`	
3	3	1	avant le 1er passage (non demandé)
9	1	3	à la fin du 1er passage
81	0	27	à la fin du 2e passage

Comme `n=0`, c'est le dernier passage dans la boucle.

Cas $x = 2$ et $n = 10$:

`x`	`n`	`res`	
2	10	1	avant le 1er passage (non demandé)
4	5	1	à la fin du 1er passage
16	2	4	à la fin du 2e passage
256	1	4	à la fin du 3e passage
256^2	0	1024	à la fin du 4e passage

Comme `n=0`, c'est le dernier passage dans la boucle.

On pouvait aussi donner les résultats « sans effectuer les calculs », ce qui donne une meilleure vue de ce que fait l'algorithme :

`x`	`n`	`res`	
3	3	1	avant le 1er p.
3^2	1	3	à la fin du 1er p.
$(3^2)^2$	0	3×3^2	à la fin du 2e p.

`x`	`n`	`res`	
2	10	1	avant le 1er passage
2^2	5	1	à la fin du 1er passage
2^4	2	2^2	à la fin du 2e passage
2^8	1	2^2	à la fin du 3e passage
2^{16}	0	$2^2 \times 2^8$	à la fin du 4e passage

3) $\boxed{\text{La fonction } \texttt{Mystere} \text{ semble calculer } x^n}$

Notons x_k, n_k et res_k les valeurs de x, n et res lors du k-ième passage dans la boucle.
Pour $k=0$, x_0, n_0 et res_0 sont les valeurs initiales de x, n et res ($res=1$).
Il semblerait que $x^n \times res$ soit constant égal à $x_0^{n_0}$. Nous allons donc vérifier que

$$\mathcal{H}_k : \quad x_k^{n_k} \times res_k = x_0^{n_0}$$

est un invariant de boucle, vrai à la fin du k-ième passage dans la boucle pour tout $k \in \mathbb{N}$.

- $\underline{\mathcal{H}_0}$ est vraie par hypothèse : $res_0 = 1$ donc $x_0^{n_0} \times res_0 = x_0^{n_0}$.
- $\underline{\mathcal{H}_k \Longrightarrow \mathcal{H}_{k+1}}$: Supposons $\mathcal{H}_k$ vraie.
  Si $\underline{n_k = 1[2]}$, c'est-à-dire si $n_k = 2p+1$ est impair, alors $res_{k+1} = res_k \times x_k$ et $n_{k+1} = p = \dfrac{n_k - 1}{2}$.
  De plus $x_{k+1} = x_k^2$. Donc en remplaçant :
  $$x_{k+1}^{n_{k+1}} \times res_{k+1} = (x_k^2)^{\frac{n_k-1}{2}} \times res_k \times x_k = x_k^{n_k+1-1} \times res_k = x_0^{n_0} \qquad \text{(d'après } \mathcal{H}_k)$$
  Donc $\mathcal{H}_{k+1}$ est vraie dans ce cas là.
  Sinon, $n_k = 2p$ est pair, $res_{k+1} = res_k$ (pas de changement), $n_{k+1} = p = \dfrac{n_k}{2}$ et $x_{k+1} = x_k^2$. D'où
  $$x_{k+1}^{n_{k+1}} \times res_{k+1} = (x_k^2)^{\frac{n_k}{2}} \times res_k = x_k^{n_k} \times res_k = x_0^{n_0} \qquad \text{(d'après } \mathcal{H}_k)$$
  Donc $\mathcal{H}_{k+1}$ est vraie là aussi.
  Ainsi, quel que soit le résultat du branchement (if), $\mathcal{H}_{k+1}$ est vraie.
- Conclusion : par récurrence, $\boxed{\forall k \geqslant 0 \qquad x_k^{n_k} \times res_k = x_0^{n_0}}$

Donc $x^n \times res = x_0^{n_0}$ est bien un invariant de boucle pour la fonction mystère.
Terminaison : La variable `n` est de type entier positif, sa valeur décroît strictement après chaque passage dans la boucle (`n//2`) donc `n` atteint 0 et $\boxed{\text{La boucle } \texttt{while} \text{ se termine}}$
Correction : Comme on divise n par 2 à chaque étape, la boucle se termine sur la valeur $n=0$. En sortie de boucle l'invariant s'écrit donc :

$$x_k^{n_k} \times res_k = x_k^0 \times res_k = res_k = x_0^{n_0}$$

Donc la variable `res` contient à ce moment là x^n. Nous venons de prouver que

$$\boxed{\text{La fonction } \texttt{Mystere} \text{ calcule } x^n}$$

Une version modifiée de la fonction `Mystere` qui affiche la valeur des variables, un compteur du nombre de passage dans la bouble, et l'invariant : DST2_ex2.py

4) Si $n = 2^k$, à chaque passage dans la boucle le test sera faux et la nouvelle valeur de n sera $\dfrac{n}{2}$, sauf au dernier passage. La boucle sera donc exécutée $k+1$ fois[1].
Il n'y a que des opérations élémentaires dans la boucle (pas de seconde boucle, ou d'opération sur des types composés). Donc la complexité est en O(nombre de passage dans la boucle), ici $O(k)$.
Comme $n = 2^k$, $k = \dfrac{\ln n}{\ln 2}$ et par conséquent $\boxed{\text{La complexité est en } O(\ln n)}$
Si n n'est pas de la forme 2^k, ce n'est pas grave. Il est encadré par des puissances de 2 : $2^k \leqslant n \leqslant 2^{k+1}$ (comme $10^3 \leqslant 5973 \leqslant 10^4$ en base 10). Donc la complexité sera encadrée par k et $k+1$. Or, lorsque $k \to +\infty$, $k \sim k+1$, donc par encadrement $k \sim k+1 \sim \dfrac{\ln n}{\ln 2}$. On pouvait raisonner « comme si » n était une puissance de 2.

On remarque d'ailleurs à la question 2 que l'on peut améliorer la fonction en supprimant le dernier calcul de `x` : la condition de sortie de boucle devient `n>1`, et juste avant le `return` (mais hors de la boucle) on rajoute `if n%==1 : res=res*x` :

1. dans ce genre de cas, toujours tester pour n petit : est-ce que ça marche pour $k=0$? $k=1$?

```
def Mystere(x,n) : # x et n sont des entiers
    res=1
    while n>1 :
        if n%2==1 :
            res=res*x
        n=n//2
        x=x*x
    if n%==1 :
        res=res*x
    return res
```

Exercice 3 (Méthode des rectangles – 6 points)

1) Il y a plusieurs syntaxes possibles :

`import math` puis `math.cos`

`import math as m` puis `m.cos` (simple abréviation)

`from math import cos,sqrt` puis `cos` (pour n'importer que les fonctions `cos` et `sqrt`).

Exemple de programme :

```
from math import cos,sqrt
def f(x):#x dans [-1,1] donc cos >= 0
    return sqrt(cos(x))
```

2) La fonction suivante convient :

```
def Rectangle(f,a,b,n):
    aire=0      #contient l'aire entre a et x_i
    pas=(b-a)/n #écart entre x_i et x_(i+1)
    for i in range(n):
        aire=aire+pas*f(a+i*pas)
    return aire
```

Le rectangle i est de largeur $\dfrac{b-a}{n}$ et de hauteur $f(x_i)$, donc son aire est $\dfrac{b-a}{n} \times f(x_i)$.
La formule pour approcher l'intégrale à l'aide de la *méthode des rectangles* est donc

$$\sum_{i=0}^{n-1} \frac{b-a}{n} \times f\left(a + i \times \frac{b-a}{n}\right)$$

Vous le reverrez en cours de mathématiques sous le nom de somme de Riemann.

DST2_ex3.py