Informatique tronc commun
Devoir n° 01
Première partie, sur papier

03 décembre 2016

Consignes de rédaction

À chaque fois que l'on écrira du code `Python`, on indiquera les niveaux d'indentation par des traits verticaux placés à gauche du code, comme cela a été fait en cours.

On numérotera chaque ligne de chaque bloc de code `Python`.

Lorsque l'on vous demande d'écrire un invariant ou un variant, on fera systématiquement référence aux lignes du bloc de code étudié. Par exemple : « Un invariant pour la boucle `for` des lignes n° 42 à 1515 est [...] ».

Lorsque l'on vous demande de justifier des invariants ou des variants, on fera systématiquement référence à une ligne du bloc de code étudié. Par exemple : « juste avant la ligne n° 42, on a que [...] », ou « juste après la ligne n° 1515, on sait que [...] ».

Lorsque l'on écrit une fonction `Python`, on ne demande pas de vérifier que les arguments donnés sont corrects. Cependant, il sera apprécié d'indiquer les préconditions vérifiées par ces arguments dans la *docstring* de la fonction.

1 Appartenance à un disque

Q1 Écrire une fonction `appartient(a,c,r)` prenant en argument

- un couple de nombres `a` ;
- un couple de nombres `c` ;
- un nombre positif `r` ;

et renvoyant la valeur de vérité de « le point de coordonnées `a` est dans le disque fermé de centre de coordonnées `c` et de rayon `r` ».

2 Nombres parfaits

On appelle *nombre parfait* tout entier naturel non nul qui est égal à la somme de ses diviseurs stricts, c'est-à-dire de ses diviseurs autres que lui-même.

Par exemple, 26 n'est pas parfait, car ses diviseurs stricts sont 1, 2 et 13, et $1+2+13 = 16 \neq 28$. Mais 28 est parfait, car ses diviseurs stricts sont 1, 2, 4, 7 et 14, et $1+2+4+7+14 = 28$.

Q2 Écrire une fonction `Python` `parfait(n)` prenant en entrée un entier naturel non nul `n`, et renvoyant un booléen donnant la valeur de vérité de l'assertion « `n` est parfait ».

Q3 Écrire les éventuels variants et invariants permettant de montrer que cette fonction renvoie le bon résultat.

3 Plus petite factorielle supérieure à 123456789

Q4 Écrire un script `Python` permettant de calculer le plus petit entier naturel n tel que $n! > 123456789$.

Q5 Écrire les éventuels variants et invariants de boucle permettant de montrer que le code `Python` écrit précédemment donne le bon résultat.

Q6 Montrer que les variants et/ou invariants donnés à la question précédente sont bien des variants/invariants de boucle et justifier que le script écrit termine et donne le bon résultat.

4 Codes autocorrecteurs

On s'intéresse au problème du codage d'une suite de bits (représentée sous la forme d'un tableau de 0 ou 1), de manière à pouvoir réparer une erreur de transmission. On fixe un entier naturel non nul `k`. Un tableau de bits `b` sera codé avec un niveau de redondance `k` en répétant chaque bit $2k+1$ fois. Pour décoder un tableau avec un niveau de redondance `k`, on le découpe en blocs de $2k+1$ bits. Dans chaque bloc, on effectue un « vote » et l'on considère la valeur majoritaire.

Exemple : Avec $k = 2$ (et donc un niveau de redondance de 2), le tableau

$$b = [0, 1, 0]$$

sera codé en

$$c = [\underbrace{0,0,0,0,0}_{5 \text{ bits}}, \underbrace{1,1,1,1,1}_{5 \text{ bits}}, \underbrace{0,0,0,0,0}_{5 \text{ bits}}].$$

Imaginons qu'après transmission, le tableau reçu soit

$$c' = [\underbrace{0,0,0,1,0}_{1 \text{ erreur}}, \underbrace{0,1,1,0,1}_{2 \text{ erreurs}}, \underbrace{0,1,0,1,1}_{3 \text{ erreurs}}].$$

On le décode alors en

$$b' = [0, 1, 1].$$

Q7 Écrire une fonction `code(b,k)` renvoyant le tableau codant un tableau `b`, avec un niveau de redondance `k`.

Q8 Écrire une fonction `decode(c,k)` renvoyant le tableau décodant un tableau `c`, avec un niveau de redondance `k`.