Faculté d'ingénierie et de technologie – Génie informatique

Algorithmique et programmation

Damien Berthet & Vincent Labatut

TP 14 | propriétés arithmétiques

Présentation

Le but de ce TP est d'approfondir les tableaux et fonctions, en manipulant des notions issues de l'arithmétique. Nous allons en particulier travailler avec les diviseurs d'un nombre entier, que nous représenterons sous forme de tableau. Pour en simplifier le traitement, nous nous limiterons à des entiers possédant `N` diviseurs (où `N` est une constante).

1 Diviseurs d'un nombre

Exercice 1

Définissez la constante `N` sous forme de macro. Écrivez une fonction `void calcule_diviseurs(int n, int diviseurs[N], int *d)` qui identifie tous les diviseurs de l'entier `n` (y compris `n` lui-même) et les range dans le tableau diviseurs, dans l'*ordre croissant*. La fonction renvoie par adresse le nombre de divisieurs identifiés, grâce au paramètre `d`. On supposera que $n \geq 2$. Testez votre fonction depuis la fonction `main`, en affichant le tableau obtenu.

exemple : pour `n=6`, on obtient le tableau `{1,2,3,6}` et `d=4`.

Exercice 2

Pour tout entier naturel $n \geq 2$, on note $s(n)$ la somme des diviseurs *propres* de n, i.e. la somme de tous ses diviseurs sauf n lui-même.

exemples :

- $s(6) = 1 + 2 + 3 = 6$;
- $s(10) = 1 + 2 + 5 = 8$;
- $s(12) = 1 + 2 + 3 + 4 + 6 = 16$;
- $s(p) = 1$, pour tout nombre premier p.

Écrivez une fonction `int additionne_diviseurs_propres(int n)` qui calcule la somme des diviseurs propres de l'entier `n`.

Exercice 3

On notera $\sigma(n)$ la somme des diviseurs de n, i.e. la somme de tous ses diviseurs (y compris n lui-même).

exemples :

- $\sigma(6) = 1 + 2 + 3 + 6 = 12$;
- $\sigma(10) = 1 + 2 + 5 + 10 = 18$;
- $\sigma(12) = 1 + 2 + 3 + 4 + 6 + 12 = 28$;
- $\sigma(p) = 1 + p$, pour tout nombre premier p.

Écrivez une fonction `int additionne_diviseurs(int n)` qui calcule la somme des diviseurs de l'entier `n`. Vous devez utiliser la fonction `additionne_diviseurs_propres`.

Exercice 4

Écrivez une fonction `void affiche_sommes_diviseurs_propres(int max)` qui affiche :

- Sur une première ligne : la liste des entiers `n` compris entre 2 et `max` ;
- Sur la deuxième ligne : les valeurs de $s(n)$ correspondantes.

On utilisera cette fonction avec un paramètre `max` inférieur ou égal à 59, de sorte que les nombres $s(n)$ aient au plus 2 chiffres. **Attention :** vous ferez en sorte que les nombres n et $s(n)$ soient alignés.

exemple : l'appel `affiche_diviseurs_propres(25)` produit l'affichage suivant :

```
n    : 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
s(n) : 01 01 03 01 06 01 07 04 08 01 16 01 10 09 15 01 21 01 22 11 14 01 36 06 16
```

2 Nombres parfaits, amicaux et sublimes

Exercice 5

On appelle *abondance* d'un nombre $n \geq 2$ la valeur $a(n) = s(n) - n$.

exemples :

- L'abondance de 6 est : $a(6) = s(6) - 6 = 6 - 6 = 0$;
- L'abondance de 10 est : $a(10) = s(10) - 10 = 8 - 10 = -2$;
- L'abondance de 12 est : $a(12) = s(12) - 12 = 16 - 12 = 4$.

Écrivez une fonction `int calcule_abondance(int n)` qui calcule l'abondance du nombre passé en paramètre. Vous devez utiliser `additionne_diviseurs_propres`.

Exercice 6

Un entier $n \geq 2$ est dit *parfait* s'il est égal à la somme de ses diviseurs propres $s(n)$, i.e. : $n = s(n)$. Autrement dit, un entier est parfait si son abondance est nulle, i.e. : $a(n) = 0$.

exemples :

- 6 est parfait, car : $s(6) = 6$;
- 10 n'est *pas* parfait, car : $s(10) \neq 10$;
- 28 est parfait, car : $s(28) = 1 + 2 + 4 + 7 + 14 = 28$.

Écrivez une fonction `int est_parfait(int n)` qui renvoie 1 si l'entier `n` est parfait, et 0 sinon.

Exercice 7

Deux entiers $n, p \geq 2$ sont dits *amicaux* lorsque la somme des diviseurs propres de chacun des deux est égal à l'autre nombre, c'est-à-dire s'ils vérifient *à la fois* $s(n) = p$ *et* $s(p) = n$.

exemple : les nombres 220 et 284 sont amicaux, car on a :

- $s(220) = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$
- $s(284) = 1 + 2 + 4 + 71 + 142 = 220$

Écrivez une fonction `int sont_amicaux(int n, int p)` qui retourne 1 si les entiers `n` et `p` sont amicaux, et 0 sinon.

Exercice 8

Un entier $n \geq 2$ est dit *sublime* lorsque le nombre de ses diviseurs et la somme de ses diviseurs sont tous les deux des nombres parfaits.

exemple : 12 est un nombre sublime, car $\sigma(12) = 1 + 2 + 3 + 4 + 6 + 12$, et on a donc à la fois :

- 6 diviseurs, et 6 est un nombre parfait ;
- $\sigma(12) = 28$, et 28 est aussi un nombre parfait.

Écrivez une fonction `int est_sublime(int n)` qui détermine si le nombre `n` est sublime. La fonction renvoie 1 si c'est le cas, et 0 sinon.

Remarque : on n'a identifié à ce jour que deux nombres sublimes.

3 Nombres abondants et déficients

Exercice 9

On appelle nombre *abondant* un nombre $n \geq 2$ dont l'abondance est *strictement positive* : $a(n) > 0$. Au contraire, on appelle nombre *déficient* un nombre dont l'abondance est *strictement négative* : $a(n) < 0$.

exemples :

- 10 est un nombre déficient, car $a(10) = -2$;
- 12 est un nombre abondant, car $a(12) = 14$.

Écrivez les fonctions `int est_abondant(int n)` et `int est_deficient(int n)` qui indiquent si le nombre `n` passé en paramètre est respectivement abondant ou déficient. Chaque fonction doit renvoyer 1 si c'est le cas (nombre abondant ou déficient) et 0 si ce n'est pas le cas.

Exercice 10

Écrivez une fonction `void affiche_nombres(int max, int mode)` qui reçoit en paramètres une valeur maximale `max` et un mode de fonctionnement `mode`. Ce mode a trois valeurs possibles : -1, 0 et $+1$. En fonction du mode, la fonction doit réaliser l'une des trois tâches suivantes :

- Pour -1 : afficher tous les entiers déficients compris entre 2 et `max` ;
- Pour 0 : afficher tous les entiers parfaits compris entre 2 et `max` ;
- Pour $+1$: afficher tous les entiers abondants compris entre 2 et `max`.

exemple : l'appel `affiche_nombres(100,1)` affiche la liste des nombres abondants inférieurs à 100 :

```
 1. 12
 2. 18
 3. 20
 4. 24
 5. 30
 6. 36
 7. 40
 8. 42
 9. 48
10. 54
11. 56
12. 60
13. 66
14. 70
15. 72
16. 78
17. 80
18. 84
19. 88
20. 90
21. 96
```