Travail sur les entiers et leurs diviseurs

Présentation de la série d'algorithmes :

Il s'agit de construire consécutivement un algorithme qui recherche les diviseurs d'un entier naturel, puis un algorithme qui détermine si un nombre est premier et enfin un algorithme qui recherche les nombres parfaits inférieurs à un entier naturel donné. Le premier algorithme est réutilisé dans les deux suivants.

1. l'Algorithme de recherche des diviseurs d'un entier en langue naturelle :

```
1  DONNEES
2          n nombre entier # n est l'entier dont on va chercher les diviseurs
3  AUTRES VARIABLES
4          i,s nombres entiers
5  TRAITEMENT
6          donner à s la valeur de la racine carrée de n
7          afficher "recherche des diviseurs de ", n
8          donner à i la valeur 1
9          tant que (i<s)
10                 si le reste de la division euclidienne de n par i est nul alors
11                         afficher i
12                         afficher n/i
13                 donner à i la valeur i+1
14         si le reste de la division euclidienne de n par s est nul alors
15                 afficher s
16 SORTIE
17         afficher "Recherche des diviseurs terminée"
```

2. Questionnement possible :

a. Faire construire cet algorithme aux élèves en langage naturel ou les faire exécuter l'algorithme à la main (on pourra questionner les élèves sur le test d'arrêt le plus efficace sqrt(n), n/2, n...).
A chaque diviseur d de n différent de $\sqrt{n}$, correspond un autre diviseur $\frac{n}{d}$.
Si un entier n possède un diviseur d strictement supérieur à $\sqrt{n}$, alors le diviseur qui lui correspond est nécessairement inférieur à $\sqrt{n}$.
Ainsi, il suffit de chercher les diviseurs d de n inférieurs ou égaux à $\sqrt{n}$ et leur correspondant pour trouver tous les diviseurs de n, ce qui limite le nombre d'itérations de la boucle. Mais il faut alors gérer à part le cas où $\sqrt{n}$ est un diviseur de n (lignes 14 et 15).
On peut ainsi donner l'algorithme aux élèves avec comme test d'arrêt i<n ou i<s en ligne 9 , i<n/2 est un test d'arrêt correct dès que n>5 puisque n/2> sqrt(n). Dans le cas i<n, il faut retirer les lignes 12, 14 et 15 pour qu'il n'y ait pas de doublons. Dans le cas i<n/2, il sera plus difficile d'éviter les doublons dans la liste de diviseurs.

b. Faire programmer cet algorithme.

c. Faire étudier comment on peut transformer cet algorithme pour compter le nombre de diviseur (*ceci permet de préparer l'algorithme suivant pour savoir si un nombre est premier.*)

d. Demander aux élèves comment on pourrait faire pour renvoyer tous les affichages à la phase de SORTIE (*ceci va déboucher sur l'utilisation d'une liste d'entiers qui peut amener un travail de tri si l'on désire afficher les diviseurs dans l'ordre croissant (**voir en annexe**).*)

3. l'Algorithme test de primalité d'un entier en langue naturelle :

```
DONNEES
        n nombre entier # n est l'entier dont on va chercher les diviseurs
        i nombre entier
        s nombre entier
        c nombre entier
TRAITEMENT
        donner à s la valeur racine carrée de n
        donner à i la valeur 1
        donner à c la valeur 0
        tant que (i<s)
                si le reste de la division euclidienne de n par i est nul alors
                        donner à c la valeur c+2
                donner à i la valeur i+1
        si le reste de la division euclidienne de n par s est nul alors
                donner à c la valeur c+1
SORTIE
        si c=2 alors
                afficher n ," est un nombre premier "
        sinon
                afficher n ," n'est pas un nombre premier car il possède ", c ," diviseurs"
```

4. Questionnement possible :

a. Faire transformer l'algorithme initial pour compter le nombre de diviseurs, et ensuite décider si le nombre est premier ou non. (on pourra remarquer que l'affichage des diviseurs ralentit l'algorithme).

b. Lorsque c atteint la valeur 3, il est certain que le nombre n'est pas premier. Demander aux élèves comment on peut arrêter la boucle dans ce cas. (*on modifie la condition du tant que en :* `tant que (i<s) et (c<=2)`).

5. l'Algorithme de recherche des nombres parfaits en langue naturelle :

Remarque *:*
Un nombre parfait est un entier naturel qui est égal à la somme de ses diviseurs propres (ie : ses diviseurs différents de lui-même, un nombre premier n'a qu'un diviseur propre : 1).
Par exemple : 6 est parfait car $6 = 1+2+3$.

```
DONNEES
        l nombre entier # limite du domaine de recherche de 1 à l
        n nombre entier # n est l'entier dont on va chercher les diviseurs
        i nombre entier
        s nombre entier
        sdp nombre entier
TRAITEMENT
        donner n à la valeur 2
        tant que n<l
                donner à sdp la valeur 1
                donner à i la valeur 2
                donner à s la valeur racine carrée de n
                tant que i<s
                        si le reste de la division euclidienne de n par i est nul alors
                                donner à sdp la valeur sdp+i
                                donner à sdp la valeur sdp+n/i
                donner à i la valeur i+1
                si le reste de la division euclidienne de n par s est nul alors
                        donner à sdp la valeur sdp+s
                si sdp=n alors
                                afficher n, "est parfait"
                n=n+1
SORTIE
        afficher "Recherche terminée"
```

6. Questionnement possible :

a. Faire construire l'algorithme en langage naturel en remarquant que pour chaque entier n inférieur à l, il va falloir déterminer tous ses diviseurs pour faire la somme de ceux différents de 1 et de n.

b. Faire programmer cet algorithme.

c. Demander combien il y a de nombres parfaits inférieurs à 20000.

Point historique sur les nombres parfaits :
Les nombres parfaits étaient déjà connus par l'école pythagoricienne. Les quatres premiers nombres parfaits sont connus depuis le début de notre ère. Il a fallu attendre le XVième siècle pour voir apparaître un cinquième nombre parfait : 33 550 336.

Les nombres de la forme $2^{p-1}(2^p-1)$ avec 2^p-1 premier sont parfaits. On appelle nombres de Mersenne les nombres de la forme 2^p-1 *(Marin Mersenne, 1588-1648)*.
Ainsi chaque fois que l'on découvre un nombre de Mersenne premier, on découvre simultanément un nombre parfait.

Actuellement, on connaît seulement une quarantaine de nombres parfaits.

Quelques questions qui restent ouvertes :

— Actuellement, les mathématiciens ne connaissent aucun nombre parfait impair, et ne savent pas s'il en existe ou non.

— De plus, les mathématiciens ne savent pas si les nombres parfaits pairs sont en nombre infini.

ANNEXE : procédure de tri de la liste de diviseurs

On considère que l'on a modifié l'algorithme 1, en remplaçant les commandes afficher i, afficher n/i et afficher s par ajouter i à la liste l, ajouter n/i à la liste l et ajouter s à la liste s.

La liste l obtenue contient un nombre d' éléments noté nd : l[0], l[1], ... l[nd-1] qui correspondent aux diviseurs de n, mais qui ne sont pas ordonnés.

Pour trier cette liste on peut utiliser l'algorithme suivant (appelé *tri à bulles*). Cet algorithme parcourt plusieurs fois la liste et à chaque passage fait "remonter" le nombre le plus grand de la zone non trié, un peu comme des bulles qui remonteraient à la surface (la plus grosse remonte la première, puis celle légèrement moins grosse ...).

```
DONNEES
l liste d'entiers # éléments à trier
nd entier # nombre d'éléments de la liste
AUTRES VARIABLES
i, j, k ,m entiers
TRAITEMENT
m=nd-1
tant que m > 0
        donner à j la valeur 0
        pour i de 1 jusqu'à m
                si l[i-1]>l[i] alors
                        donner à k la valeur de l[i]
                        donner à l[i] la valeur de l[i-1]
                        donner à l[i-1] la valeur de k
                        donner à j la valeur de i
        donner à m la valeur de j
SORTIE
#affichage de la liste
pour i de 0 jusqu'à nd-1
        afficher l[i]," "
```