TP algorithmique 1 - PGCD et fractions - 2nde 4

I Division

a. Comme en primaire

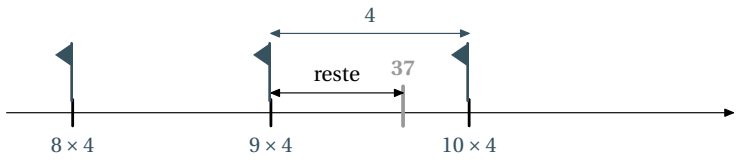
Vous voulez diviser 37 par 4 : à l'école primaire, vous avez appris à vous poser la question :

En 37 combien de fois 4 ?

Puis vient le temps du calcul :

- *une fois 4, 4 : trop petit...*
- *2 fois 4, 8 : trop petit...*
- *3 fois 4, 12 : trop petit...*
- ...
- *9 fois 4, 36 : trop petit...*
- *10 fois 4, 40 : Ah ! Trop grand !*

« Il y va » donc 9 fois mais on n'y est pas encore : il me reste un petit pas à faire de 36 jusqu'à 37.



Comment traduire cette division à l'aide d'une somme et d'un produit ?
Comment s'appelle chaque membre de cette division ?

b. Parlons à l'ordinateur

Imaginez maintenant que vous devez expliquer ce calcul à un ordinateur en l'imaginant comme un petit lutin se promenant sur la droite des nombres.

Vous pourriez commencer par lui dire :

« tu pars de 0 et tu vas compter tes enjambées »,

puis

« tu avances de 4 unités tant que tu n'auras pas dépassé 37 ».

Enfin,

« dès que tu as dépassé 37, tu t'arrêtes : le quotient est le nombre d'enjambées moins une ».

Il ne reste plus qu'à lui dire dans sa langue... Nous étudierons principalement deux langues informatiques cette année : le XCAS et le OCAML.

Par exemple, en XCAS, cela donne :

```
quotient(dividende,diviseur):={
enjambees:=0;
 while(enjambees*diviseur<=dividende){
  enjambees:=enjambees+1
 }
return(enjambees-1)
}
```

Écrivez un script qui renvoie cette fois le reste de la division euclidienne de deux nombres.

c. Parlons à un ordinateur plus malin

Les mathématiques, c'est pour les fainéants... Le problème, c'est que pour pouvoir se payer le luxe d'être fainéant, il faut pas mal réfléchir !
Par exemple, je suis malin et passablement fainéant. Je dois diviser a par b donc savoir « en a combien de fois b ». Je me dis :

Ben déjà, si a est plus petit que b, en a il y va zéro fois : le quotient vaut donc zéro

Puis je me dis :

Si je recule de b, il ira une fois de moins b dans a. Cela signifie que quotient(a,b)=1+quotient(a-b,b)

Eh ben, avec ça, un ordinateur intelligent va se débrouiller...
On lui demande le quotient de 37 par 4 :
- est-ce que 37 est plus petit que 4 ? Non, donc je garde en mémoire que quotient(37,4)=1+quotient(37-4,4) et je m'occupe de quotient(33,4) ;
- est-ce que 33 est plus petit que 4 ? Non, donc je garde en mémoire que quotient(33,4)=1+quotient(33-4,4) et je m'occupe de quotient(29,4) ;
- etc.
- est-ce que 5 est plus petit que 4 ? Non, donc je garde en mémoire que quotient(5,4)=1+quotient(5-4,4) et je m'occupe de quotient(1,4) ;
- est-ce que 1 est plus petit que 4 ? Oui, donc je sais que quotient(1,4)=0 ;
- donc je vais pouvoir calculer quotient(5,4) puis quotient(9,4) puis etc.

Ça paraît plus long mais ce sale boulot de gestion de la mémoire est fait par des logiciels intelligents. Nous pouvons nous contenter de donner juste à l'ordinateur un cas simple et un moyen pour aller d'un cas compliqué vers le cas simple.
Regardez plutôt !

```
quotient_malin(dividende,diviseur):={
 if(dividende<diviseur)
   then{0}
   else{1+quotient_malin(dividende-diviseur,diviseur)}
}
```

OCAML parle à peu près le même langage :

```
let rec quotient_malin(dividende,diviseur)=
   if dividende<diviseur
     then 0
     else 1+quotient_malin(dividende-diviseur,diviseur)
```

Écrivez un script qui renvoie cette fois le reste malin de la division euclidienne de deux nombres.

II PGCD

a. Diviseur

Le PéGéCéDé de deux nombres, c'est leur plus grand diviseur commun.
D'abord, qu'est-ce qu'on appelle un diviseur dans ce contexte ?
Par exemple, 4 divise 80 car le reste de la division de 80 par 4 vaut 0 : « la division tombe juste » comme vous disiez en CM1.
On peut aussi dire que 4 divise 32 : 4 est donc un *diviseur commun* de 32 et 80. Mais est-ce le plus grand ?

b. Euclide

Euclide, c'est un savant grec qui a eu beaucoup de bonnes idées il y a 2300 ans. On utilise encore largement ses résultats au lycée.
Bon, vous voulez connaître le plus grand diviseur commun de deux nombres a et b.
Est-ce qu'il en existe au moins un ?
L'idée d'Euclide a été d'effectuer la division...euclidienne de a par b. Soit a le plus grand des deux.

$$a = b \times q + r \qquad \text{avec} \quad 0 \leqslant r < b$$

Appelons d un diviseur commun de a et b.
Pourquoi d est aussi un diviseur commun de b et r ?
Et vice-versa et dans l'autre sens ?
Donc chercher le PGCD de a et b revient à chercher celui de b et r.
Quel est l'avantage ? Qu'est-ce qui se passe au bout d'un moment ? Y a-t-il un cas simple ? Peut-on utiliser notre méthode de fainéant ?
Testez vos conjectures sur la machine.

III Opérations sur les nombres rationnels

a. Fraction simplifiée

Quelle différence existe-t-il entre $\frac{7}{3}$ et $\frac{21}{9}$?
Quel rôle peut jouer le PGCD dans cette histoire ?
On va créer un programme simplifiant les fractions.
Au lieu d'écrire un nombre rationnel sous la forme $\frac{\text{numérateur}}{\text{dénominateur}}$, on va l'entrer dans la machine sous la forme `[numérateur,dénominateur]`.
Ainsi $\frac{21}{9}$ s'écrira `[21,9]`.
On peut alors créer un programme qui simplifiera une fraction donnée.
Par exemple sur XCAS :

```
simp(Fraction):={
a:=Fraction[0]; // le premier terme de la fraction
b:=Fraction[1]; // le second
  if(b==0)
    then{"On peut pas diviser par 0"}
    else{[a/pgcd(a,b),b/pgcd(a,b)]}
}:;
```

Des commentaires ?
Sur OCAML, on procède à quelques aménagements.

Pour prévenir toute division par zéro on va créer un opérateur traitant les exceptions qu'on nommera `Division_par_zero` et qui nous servira de garde-fou :

```
# exception Division_par_zero;;
```

On invoquera cette exception avec la commande `raise` : étant donné votre niveau d'anglais, cette commande est naturelle !...
On peut créer un opérateur qui simplifie les fractions :

```
# let simp([a;b]) =
 if b=0  then raise Division_par_zero
 else [a/pgcd(a,b);b/pgcd(a,b)];;
```

b. Opérations sur les fractions

On crée ensuite une somme simplifiée de fractions :

```
# let som([a;b],[c;d]) =
  if b=0 or d=0 then raise Division_par_zero else
  simp([a*d+b*c;b*d]);;
```

Par exemple :

```
# som([2;3],[1;6]);;
- : int list = [5; 6]
```

Bon : occupez-vous de la multiplication, du calcul de l'inverse et de la division.
Quand tout sera prêt, comment entrer

$$1 + \frac{2 + \frac{3}{4}}{1 - \frac{5}{6}}$$

et

$$3 + \frac{1}{2 + \frac{1}{1 + \frac{1}{2 + \frac{1}{6}}}}$$

.