TEST DE PRIMALITÉ
Un doigt d'algorithmique...

Compétences mathématiques :

- Division euclidienne, définition d'un nombre premier

Compétences informatiques :

- test « SI », boucle « TANT QUE ». Notion d'efficacité d'un algorithme.

Prérequis :

- Avoir manipulé les tests et les boucles.

Partie 1: Mise au point théorique du test

Le problème

On veut créer une procédure qui teste si un entier donné est premier ou non.

1. Simplification du problème

Une première idée pour tester la primalité d'un nombre donné a est de le diviser successivement par tous les entiers supérieurs à 2 plus petits que lui.

a. Quand pourra-t-on conclure que le nombre est premier ?

b. Pourquoi suffit-il de diviser par les entiers inférieurs à $\sqrt{a}$?

2. Division euclidienne

XCAS dispose d'une fonction `reste(n,p)` qui renvoie le reste de la division euclidienne de n par p. En quoi peut-elle nous être utile pour résoudre notre problème ?

Partie 2: Analyse de l'algorithme

1. On rentre un nombre a : que faire si ce nombre est pair ? Pourquoi se pose-t-on cette question ? S'il n'est pas pair, on continue.
2. Si on prend les entiers dans l'ordre croissant, par lequel va-ton continuer à tester a d'après nos remarques faites à la question 1. ? Notons j cet entier. Tant que cet entier reste inférieur à $\sqrt{a}$, on va tester s'il divise a. Si oui, que fait-on ?
3. Sinon, regarde-t-on l'entier suivant ?
4. Si on finit par dépasser $\sqrt{a}$ sans que nos différents j n'aient divisé a : que peut-on en conclure ?

Partie 3: Écriture de l'algorithme

Nous allons appeler notre procédure `test_premier(a)`.

1. À quel sorte de test correspond le problème évoqué à la question 1 de la partie 2 ?
2. Quels autres tests ont été évoqués dans les questions suivantes ?
3. Écrivez la procédure en utilisant les mots-clés suivants : `tantque`, `return`, `si...alors...sinon`, `reste`.

Partie 4: Efficacité de l'algorithme

1. Testons notre test... sur 42, 43, 143, 12345678910111213141516171819.
2. Plus dur : 10000030019000057 est-il premier ?
3. Essayez de trouver un grand nombre premier (au moins 10 chiffres).
4. Comment trouver un grand nombre premier d'au moins 20 chiffres ? **XCAS** peut nous y aider avec la commande `nextprime(n)` qui donne le premier nombre premier qui suit n.

   Que donne `test_premier(nextprime(10^20))` ?

   Si vous trouvez le temps trop long, n'hésitez pas à cliquer sur `STOP` ...

   Notre procédure est en fait très peu efficace. Il faut savoir que la manipulation des nombres premiers de plus de 200 chiffres est à la base de la sécurité des messages sur internet, des transactions bancaires, etc.

   XCAS a son propre test de primalité noté `isprime`. Que donne :

```
isprime(nextprime(10^100))
```

Quel a été le temps de calcul. Il nous reste beaucoup de chemin à parcourir pour être capable d'être aussi efficace...

Partie 5: Application : recherche des nombres chanceux

Nouveau problème

Soit l'algorithme suivant :
- choisir un entier naturel ;
- l'élever au carré ;
- ajouter au résultat le nombre initialement choisi ;
- à ce dernier ajouter 41 ;
- lire ce dernier résultat.

Qu'observe-t-on ? Et si on prend un autre nombre que 41 ?

1. Écrire l'algorithme à l'aide de **XCAS**.
2. Si on part de 0, jusqu'à quand obtient-on des nombres premiers ? (On pourra utiliser notre test des parties précédentes ou `isprime`).
3. Essayez avec d'autres nombres premiers plus petits que 41.
4. Commentez ces procédures :

```
test_court_premier(a):={
si reste(a,2)==0 alors return(0);
sinon j:=3;
   tantque  j*j<=a faire
        si reste(a,j)==0 alors return(0);
        sinon j:=j+2;
        fsi
   ftantque
   return(1);
fsi
}:;
```

Puis

```
test_chanceux(X):={
local t;
t:=0;
tantque test_court_premier(chanceux(t,X))==1 faire t:=t+1;
ftantque
si t>=X-2 alors return(1); sinon return(0);
fsi
}:;
```

et enfin

```
recherche_chanceux(x):={
Liste:=NULL;
pour k de 2 jusque x faire
  si test_chanceux(k)==1 alors Liste:=Liste,k;
  fsi
fpour
Liste
}:;
```