TD : Programmes d'arithmétique

2 heures

Rédigé par Pascal Delahaye

12 décembre 2017

I] Les nombres premiers

1. Erathostène : construction d'une liste de nombres premiers

   Construire un programme `Liste_prem(N)` qui renvoie tous les nombres premiers inférieurs ou égaux à N.
   Ce programme devra impérativement mettre en oeuvre l'algorithme d'Erathostène.

   L'idée est d'obtenir une liste de taille N+1 telle que L[k] = 1 si k est premier et L[k] = 0 sinon.

   On procédera de la façon suivante :

```
-> On crée une liste de taille N+1 ne contenant que des 1 : L = [1]*(N+1)
   L[0] = 0
   L[1] = 0

-> p = 2

-> Tant que p <= sqrt(N) faire :
     - eliminer les multiples de p inférieurs ou égaux à N
     - redefinir p en posant : p = le nombre premier suivant

-> On construit alors la liste des nombres premiers cherchés en ne gardant
   que les valeurs de k telles que L[k] = 1
```

   On construit alors la liste `Listprem = Liste_prem(10000)` contenant les nombres premiers inférieurs à 10000.
   Cette liste sera utilisée dans les programmes suivants.

2. Ulisation de la liste de nombres premiers trouvée :

(a) Test de primalité :

Nous avons vu en cours que si un nombre n est composé, alors il admet un diviseur premier inférieur ou égal à $\sqrt{n}$. En utilisant ce résultat, construire une fonction `Testprem(n)` qui teste si $n \geq 2$ est un nombre premier.

(b) Décomposition en facteurs premiers :

Construire une procédure `Decomp(n)` permettant de décomposer un entier $n \geq 2$ en facteurs premiers.
On retournera le résultat sous la forme d'une liste de couples (p, v_p) où p est premier et v_p est la valuation de n en p.

(c) Répartition des nombres premiers :

Construire une procédure `Graphe(N)` qui renvoie le graphe représentant le nombre de nombres premiers inférieurs à $n \in [\![2, N]\!]$ en fonction de n.

On pourra alors vérifier (en adaptant la procédure précédente) qu'au voisinage de $+\infty$, ce nombre est équivalent à $\frac{n}{\ln n}$

II] PGCD

1. PGCD et PPCM :

Construire une procédure `Pgcd_Ppcm(a,b)` qui renvoie le PGCD et le PPCM de deux entiers $a, b \in \mathbb{N}^*$.
Ce programme devra reposer sur l'algorithme d'Euclide.

2. BEZOUT :

Construire une procédure `Bezout(a,b)` qui renvoie un couple de Bezout associé aux deux entiers $a, b \in \mathbb{N}^*$.
Ce programme devra reposer sur l'algorithme de Bezout vu en cours.

Rappel de l'algorithme :
On construit les 3 suites (r_k), (u_k) et (v_k) telles que : $\begin{cases} r_0 = a \\ r_1 = b \end{cases}$, $\begin{cases} u_0 = 1 \\ u_1 = 0 \end{cases}$ et $\begin{cases} v_0 = 0 \\ v_1 = 1 \end{cases}$.

Et pour tout $k \in \mathbb{N}$:

- r_{k+2} est le reste de la division euclidenne de r_k par r_{k+1}.
- $u_{k+2} = u_k - qu_{k+1}$ où q est le quotient de la division euclidienne de r_k par r_{k+1}.
- $v_{k+2} = v_k - qv_{k+1}$ où q est le quotient de la division euclidienne de r_k par r_{k+1}.

Nous avons alors la relation : $u_k a + v_k b = r_k$ pour tout $k \in \mathbb{N}$.
Or, d'après Euclide, nous savons que le dernier reste non nul r_{n_0} est le PGCD de a et b.
Notre programme doit donc renvoyer le couple $(u_{n_0},\ v_{n_0})$

3. Equations Diophantiennes :

Construire une procédure `Diophant(a,b,c)` qui renvoie les solutions de l'équation diophantienne $ax + by = c$
Ce programme pourra utiliser les programmes `Pgcd_Ppcm` et `Bezout` précédents.

Principe de l'algorithme :

- On vérifie si le PGCD divise c
- Si oui :
  - On simplifie le problème en divisant a, b et c par le PGCD
  - On recherche une solution particulière à l'aide d'un couple de Bezout
  - On en déduit la forme des solutions

III] DIVERS

1. Nombres parfaits :

   On dit qu'un entier naturel est parfait lorsqu'il est égal à la somme de ses diviseurs positifs strictes.

   (a) Construire une procédure `Somme_div(n)` qui renvoie la somme des diviseurs de $n \in \mathbb{N}^*$.
   (b) En déduire une procédure `Test_parfait(n)` qui teste si le nombre $n \in \mathbb{N}^*$ est parfait.
   (c) En déduire une procédure `Liste_parfait(N)` qui renvoie la liste des nombres parfaits inférieurs à N.

2. Nombres amiables :

   On dit que deux entiers naturels a et b sont amiables lorsque la somme des diviseurs de l'un est égale à la somme des diviseurs de l'autre. Construire une procédure `Liste_amiable(N)` qui renvoie la liste des couples de nombres amiables inférieurs ou égaux à N.