Dernier exemple

Alain Cournier

Le problème

- Nous souhaitons calculer le quotient et le reste d'une division Euclidienne d'un entier naturel ***a*** par un autre entier naturel non nul ***b***.

- **Contrainte :** nous **n'utiliserons pas** dans le code de notre algorithme les caractères * ou **/**.

- (Au niveau assembleur, la division n'est pas une opération élémentaire)

Comprendre le problème

- Soit ***a*** et ***b*** deux entiers naturels tels que ***b*** ≠ 0.
- On recherche les deux uniques entiers naturels ***q*** et ***r*** tels que :
  - ***a*** = ***b*******q*** + ***r***
  - Avec ***r*** < ***b***
  - On appelle ***q*** le *quotient* et ***r*** le *reste*.

Premier algorithme

- Le principe de ce premier algorithme consiste à compter le nombre de fois où il est possible de soustraire l'entier naturel ***b*** à notre entier naturel ***a***.

- **But du jeu :**
  - Après ***q*** tours de boucle, on doit avoir retiré ***q*** fois la valeur de ***b*** à ***a***.
  - On s'arrête dès que cette soustraction devient impossible dans les entiers naturels

Algorithme itératif

Algo DivModIt

Données : ***a***, ***b*** deux entiers naturels // ***b*** ≠ 0

Résultats : ***q***, ***r*** deux entiers naturels // ***r*** < ***b*** et ***a*** = ***b***.***q*** + ***r***

DebutCode

q ← 0; ***r*** ← ***a***; // Pos 1

Tant que ***b*** ≤ ***r*** faire // Pos 2

q ← ***q***+1; ***r*** ← ***r***-***b*** // Pos 3

Fin Tant que //Pos 4

FinCode

Exécution de notre algorithme

tour	a	b	q	r
0	17	3	0	17
1			1	14
2			2	11
3			3	8
4			4	5
5			**5**	**2**

Remarques : le bon résultat

- Prop1(***a***,***b***,***q***,***r***) ≡ ***a***=***b***.***q***+***r***
- En Pos 1, Pos 2, Pos 3, Pos 4 : Prop1(***a***,***b***,***q***,***r***) est vraie
- En Pos 2, on sait de plus que ***b*** ≤ ***r*** ce qui permet d'être sûr que le résultat de la soustraction ***r***-***b*** sera un entier naturel.
- En Pos 4, on sait de plus que ***r*** < ***b***

  (le test b ≤ r a répondu faux pour la première fois)

- Donc, si notre algorithme renvoie un résultat, c'est le bon

Remarques : terminaison

- A chaque tour de boucle, on sait que l'on soustraira à ***r*** la valeur de ***b***.
- Or, ***b*** est un entier naturel non nul.
- Donc, à chaque tour de boucle la valeur de ***r*** décroit strictement.
- On entre dans la boucle que si $b \leq r$ donc en Pos 3, on a $r-b \geq 0$. Donc, ***r*** reste un entier naturel (il l'est au début car ***a*** est un entier naturel).
- Donc, au cours de notre exécution les valeurs de ***r*** décrivent une suite d'entiers naturels strictement décroissante et minorée par 0.
- Donc cette suite est finie. Ce qui entraine la sortie de la boucle « tant que » en un nombre fini d'itérations.

Complexité

- La valeur de ***q*** compte le nombre de tours de boucle.
- Chaque tour de boucle contient une soustraction et une addition (coûts addition/soustraction similaires).
- La complexité de notre algorithme est donc proportionnelle à ***a***/***b*** opérations. Dans le pire des cas, ***b*** = 1.
- Nous obtenons donc une complexité en Θ(***a***) opérations (d'additions).
- Version récursive ?

Algorithme récursif

Algo DivModRec

Données : ***a***, ***b*** deux entiers naturels // ***b*** ≠ 0

Résultats : ***q***, ***r*** deux entiers naturels // ***r*** < ***b*** et ***a*** = ***b***.***q*** + ***r***

DebutCode

Si ***b*** > ***a*** alors ***q*** ← 0; ***r*** ← ***a*** // Pos 1

Sinon

DivModRec(***a***-***b***, ***b***, ***q***, ***r***); // Pos 2

q ← ***q***+1; // Pos 3

Fin Si // Pos 4

FinCode

Exécution de notre algorithme

a	b	q	r
17	3		
14	3		
11	3		
8	3		
5	3		
2	**3**	**0**	**2**

Exécution de notre algorithme

a	b	q	r
17	3	**5**	**2**
14	3	4	2
11	3	3	2
8	3	2	2
5	3	1	2
2	**3**	**0**	**2**

Remarques

- Cet algorithme récursif (terminal) est strictement équivalent à l'algorithme itératif précédent

- En Pos 1, Prop1(***a***,***b***,***q***,***r***) et ***r*** < ***b*** est vraie

- Prop1(***a***-***b***,***b***,***q***,***r***) et ***r*** < ***b*** en Pos 2 implique Prop1(***a***,***b***,***q***,***r***) et ***r*** < ***b*** en Pos 3

- En faisant une récurrence sur la profondeur des appels récursifs on obtient que Prop1(***a***,***b***,***q***,***r***) et ***r*** < ***b*** est vraie en Pos 4.

Complexité

- Le pire des cas pour la profondeur des appels récursifs est ***b***=1.

- **Conséquence :** la complexité dans le pire des cas est seulement dépendante de la valeur de la variable ***a***. Deux cas : ***a*** positif ou nul.

- T(0) = 3 (vous pouvez mettre la constante que vous souhaitez)
- T(*n*+1) = 6 + T(*n*)
  - T(*n*) est le temps d'exécution de l'algorithme quand le premier paramètre ***a*** vaut *n*

Complexité

- $T(n+1) = 6 + T(n) = 2*6 + T(n-1) = 3*6 + T(n-2)$
- On veut démontrer que après i remplacements (quel que soit i)
  - $T(n+1) = i*6 + T(n-i+1)$

  - Vrai pour 0 remplacement car $T(n+1) = 0*6 + T(n-0+1)$
  - On suppose que la formule est vraie pour k remplacements donc :
  - $T(n+1) = k*6 + T(n-k+1)$ et on effectue un remplacement de plus
  - $T(n+1) = k*6 + 6 + T(n-k)$ nous avons effectué $k+1$ remplacement et
  - $T(n+1) = (k+1)*6 + T(n-(k+1)+1)$ la formule est donc exacte

Complexité

- On a démontré que après i remplacements (quel que soit i)
  - $T(n+1) = i*6 + T(n-i+1)$

- Or on connait T(0), on cherche donc i tel que $n-i+1 = 0$. Soit $i=n+1$
  - $T(n+1) = (n+1)*6 + T(0) = 6(n+1) + 3$
  - Si $\boldsymbol{a} = n+1$, on obtient $6.\boldsymbol{a}+3$
  - D'où, complexité en $\Theta(\boldsymbol{a})$ opérations

Question : Peut-on faire mieux ?

- Si $b > a$ le résultat est simple à obtenir $q = 0$ et $r = a$

- Sinon supposons que je sache diviser a par $2.b$ et que le résultat soit :
  - $a = (2b).q' + r'$
  - Comment calculer q et r ? Il faut distinguer deux cas :
    - $r' < b$: $a = (2b).q' + r' = b.(2q') + r'$, donc avec $q = 2q'$ et $r = r'$, on a $a = b.q + r$ avec $r < b$
      (en particulier, puisque r' est un entier naturel, r est aussi un entier naturel)
    - $r' \geq b$: mais $r' < 2b$. Donc $0 \leq r'-b < b$.
      $a = (2b).q' + r' = b.(2q') + r' = b.(2q')+b+r'-b = b.(2q'+1) + r'-b$
      Donc, avec $q = 2q'+1$ et $r = r'-b$, on a $a=b.q+r$ avec $0 \leq r < b$
      (en particulier, r reste un entier naturel)

Question : Peut-on faire mieux ?

- Si $b > a$ le résultat est simple à obtenir $q = 0$ et $r = a$

- Sinon supposons que je sache diviser a par $2.b$ et que le résultat soit :
  - $a = (2b).q' + r'$
  - Comment calculer q et r ? Il faut distinguer deux cas :
    - $r' < b$: $a = (2b).q' + r' = b.(2q') + r'$, donc avec $q = 2q'$ et $r = r'$, on a $a = b.q + r$ avec $r < b$
      (en particulier, puisque r' est un entier naturel, r est aussi un entier naturel)
    - $r' \geq b$: mais $r' < 2b$. Donc $0 \leq r'-b < b$.
      $a = (2b).q' + r' = b.(2q') + r' = b.(2q')+b+r'-b = b.(2q'+1) + r'-b$
      Donc, avec $q = 2q'+1$ et $r = r'-b$, on a $a=b.q+r$ avec $0 \leq r < b$
      (en particulier, r reste un entier naturel)

Je triche puisque j'utilise $2*b$

Question : Peut-on faire mieux ?

- Je triche puisque j'utilise 2*$\boldsymbol{b}$

  - Non !
  - Car 2*$\boldsymbol{b}$ = $\boldsymbol{b}$+$\boldsymbol{b}$

Question : Peut-on faire mieux ?

- Peut-on commencer ? Oui si on connait ***a*** et ***b***

- Quand finir ? Quand ***b*** > ***a*** !

Algorithme récursif

Algorithme DivModRecV2

Données : ***a***, ***b*** deux entiers naturels // ***b*** ≠ 0

Résultats : ***q***, ***r*** deux entiers naturels // ***r*** < ***b*** et ***a*** = ***bq*** + ***r***

DebutCode

Si ***b*** > ***a*** alors ***q*** ← 0; ***r*** ← ***a*** // Pos 1

Sinon

DivModRecV2(***a***, ***b***+***b***, ***q***, ***r***); ***q*** ← ***q***+***q***; // Pos 2

Si ***b*** ≤ ***r*** alors ***r*** ← ***r***-***b***; ***q***← ***q***+1 FinSi

Fin Si // Pos 4

FinCode

Exécution de notre algorithme

a	b	q	r
77	3		
77	6		
77	12		
77	24		
77	48		
77	**96**	**0**	**77**

Exécution de notre algorithme

a	b	q	r
77	3		
77	6		
77	12		
77	24		
77	48	2*0+1=1	77-48=29
77	**96**	**0**	**77**

Exécution de notre algorithme

a	b	q	r
77	3		
77	6		
77	12		
77	24	2*1+1=3	29-24=5
77	48	1	29
77	**96**	**0**	**77**

Exécution de notre algorithme

a	b	q	r
77	3	**2*12+1=25**	**5-3=2**
77	6	2*6=12	5
77	12	2*3=6	5
77	24	3	5
77	48	1	29
77	**96**	**0**	**77**

Exécution de notre algorithme

a	b	q	r
77	3	**25**	**2**
77	6	12	5
77	12	6	5
77	24	3	5
77	48	1	29
77	**96**	**0**	**77**

L’algorithme se termine

- Puisque la valeur de la variable ***b*** est un entier naturel non nul on a : 2****b*** > ***b***.

- Au fil des appels récursifs, les valeurs successives de la variable ***b*** forme une suite strictement croissante d’entiers naturels majorée par 2 fois la valeur de la variable ***a***.

- La longueur de la suite des valeurs de notre variable ***b*** est donc finie et notre algorithme se termine (test de fin de la récursivité).

Bon résultat

- On va faire une récurrence sur le nombre d'appel récursif
- P(*k*) : Si l'algorithme DivModRecV2(***a***,***b***,***q***,***r***) demande *k* appels récursifs alors ***a***=***bq***+***r*** et ***r*** < ***b***
- P(0) est vrai car dans ce cas on ne fait pas d'appel récursif, ***a*** < ***b*** donc ***q***=0 et ***r***=***a***. Ainsi, ***b***.***q***+***r*** = 0.***b***+***a***=***a*** : l'égalité est vérifiée et ***r*** < ***b***, donc P(0) est vraie

Bon résultat

- On fait une récurrence sur le nombre d'appels récursifs
- P(*k*) prouve P(*k*+1) :
  - Soit ***a*** et ***b*** deux entiers naturels avec ***b*** non nul tels que DivModRecV2(***a***,***b***,***q***,***r***) demande ***k***+1 appels récursifs.
  - Puisque *k*+1 >1, l'algorithme appelle récursivement DivModRecV2(***a***,***b***+***b***,q',r').
  - DivModRecV2(***a***,***b***+***b***,q',r') demande donc *k* appels récursifs et puisque notre propriété P(*k*) est supposée vraie, on a ***a*** = q'(***b***+***b***) + r'= 2q'***b***+r' et r' < 2***b***.
  - Deux cas doivent être étudié r' < ***b*** et r' ≥ ***b*** (mais r' < 2***b***)

Bon résultat

- **Cas 1 :** r’ est un entier naturel tel que r’ < ***b***. Dans ce cas, l’algorithme exécute l’instruction ***q*** ← q’+q’ (= 2q’) et ***r*** = r’.

  On obtient ***a*** = 2q’***b***+r’ = ***qb***+***r*** avec ***r*** un entier naturel tel que ***r*** < ***b***
  - P(k+1) est vérifiée

- **Cas 2 :** r’ ≥ ***b*** (mais r’ < 2***b***). Dans ce cas, l’algorithme exécute les instructions suivantes ***q*** ← q’+q’+1 (= 2q’+1) et ***r*** ← r’-***b***.

  On obtient alors les équations ***a*** = 2q’***b***+r’ = (***q***-1)***b***+(***r***+***b***) =***qb***-***b***+***r***+***b***=***qb***+***r*** avec 0 ≤ ***r*** = r’-***b*** < ***b***. Donc, ***r*** est un entier naturel tel que ***r*** < ***b***.
  - P(k+1) est vérifiée

- Par récurrence P(*n*) est vrai pour tout *n*

Complexité

- Pour connaitre la profondeur des appels récursifs, on cherche le plus petit entier k tel que $2^k \boldsymbol{b} > \boldsymbol{a}$ ce qui nous donne $k > \log(\boldsymbol{a}) - \log(\boldsymbol{b})$

- Comme le pire des cas est lorsque $\boldsymbol{b} = 1$ et que chaque appel récursif utilise au plus 20 opérations on obtient une complexité en $\Theta(\log(\boldsymbol{a}))$.

Comment faire en itératif ?

- SOS structures de donnée ?