

Annexe Documents "Mathematica"

Est indiqué à gauche de chaque titre la référence du paragraphe illustré du texte principal

II.1. Fonction moins et sg signum

```
moins[x_, y_] := If[x ≥ y, x - y, 0]
{moins[3, 5], moins[6, 2]}
```

```
{0, 4}
```

```
sg[x_] := moins[1, moins[1, x]]
{sg[0], sg[1], sg[3]}
```

```
{0, 1, 1}
```

II.6. Plus petit entier $y \leq z$, s'il existe, tel que: $A[x, y] = 0$, sinon $z+1$.

6.2. Quotient élémentaire de la division euclidienne

```
A[{a_, b_}, q_] := moins[a + 1, b (q + 1)]
```

$$\text{quotél}[a_, b_] := \sum_{1=0}^a \prod_{q=0}^1 \text{sg}[A[\{a, b\}, q]]$$

```
{quotél[3, 2], quotél[4, 0], quotél[29, 7], quotél[381, 17]}
```

```
{1, 5, 4, 22}
```

immédiat

6.3. Racine carrée entière par défaut

```
A[a_, q_] := moins[(a + 1), (q + 1)2]
```

$$\text{racdéfé1}[a_] := \sum_{1=0}^a \prod_{q=0}^1 \text{sg}[A[a, q]]$$

```
{racdéfé1[13], racdéfé1[321]}
```

```
{3, 17}
```

Immédiat

```
racdéfé1[5874]
```

```
⌘Aborted
```

Aucune réponse au bout de 5 minutes

6. Autre exemple : nombre de chiffres en écriture décimale d'un nombre entier naturel

```

A[n_, k_] := moins[(n + 1), 10k]
nbchif[n_] :=  $\sum_{l=0}^n \prod_{k=0}^l \text{sg}[\mathbf{A}[n, k]]$ 
{nbchif[3], nbchif[14], nbchif[99], nbchif[100]}
{1, 2, 2, 3}
nbchif[2624]
$Aborted

```

Aucune réponse au bout de 6 minutes

III.2. Fonctions primitives récursives, Itération à l'aide de Nest

Signum en primitive récursivité

```

Sg[0] = 0
Sg[x_Integer /; x > 0] := 1
{Sg[0], Sg[1], Sg[3]}
{0, 1, 1}

```

Prédécesseur en itération pure à 2 arguments

```

h[{x_, y_}] := {y, y + 1}
pd[n_] := Nest[h, {0, 0}, n][[1]]
{pd[0], pd[1], pd[2]}
{0, 0, 1}

```

Différence positive en itération pure à un argument

```

Moins[x_, y_] := Nest[pd, x, y]
{Moins[3, 1], Moins[1, 3]}
{2, 0}

```

Factorielle en itération pure à deux arguments

```

g[{u_, v_}] := {(v + 1) × u, v + 1}
fact[n_] := Nest[g, {1, 0}, n][[1]]
{fact[0], fact[5], fact[12]}
{1, 120, 479001600}

```

Quotient de la division euclidienne en itération pure à trois arguments

```

c[{q_, a_, b_}] := {q + Sg[moins[(a + 2), b × (q + 1)]], a + 1, b}
quotit[a_, b_] := Nest[c, {0, 0, b}, a][[1]]
{quotit[3, 2], quotit[4, 0], quotit[29, 7], quotit[381, 17]}
{1, 4, 4, 22}

```

Relativement long (10 secondes)

Quotient de la division euclidienne en itération à deux arguments et un paramètre

```

ct[{q_, a_}, b_] := {q + If[b × (q + 1) ≤ a + 1, 1, 0], a + 1}
Quotit[a_, b_] := Nest[ct[#, b] &, {0, 0}, a][[1]]
{Quotit[3, 2], Quotit[4, 0], Quotit[29, 7], Quotit[381, 17]}
{1, 4, 4, 22}

```

Immédiat

III.7.4. Surexponentielle de base 2 [exponentielle de base 2 composée k fois avec elle même agissant sur 1]

```

exp2[x_] := 2x
surexp2[k_] := Nest[exp2, 1, k]
{surexp2[0], surexp2[1], surexp2[3], surexp2[4]}
{1, 2, 16, 65536}

```

Immédiat

IV.3.5. Addition décimale par la technique enseignée à l'école primaire

Calculs préparatoires

Inversion de l'ordre d'une liste

```

inv[u_] := Table[u[[Length[u] + 1 - k]], {k, Length[u]}]
inv[{1, 3, 5}]
{5, 3, 1}

```

Prolongement à la longueur l d'une liste par des 0 à droite

```

pro[l_, u_] := Table[If[k ≤ Length[u], u[[k]], 0], {k, l}]
pro[5, {1, 3, 6}]
{1, 3, 6, 0, 0}

```

Addition de deux listes de longueur éventuellement différente

```

Som[u_, v_] := pro[Max[Length[u], Length[v]], u] +
pro[Max[Length[u], Length[v]], v]
Som[{1, 2}, {5, 3, 8}]
{6, 5, 8}

Som[{1, 0, 6}, {7}]
{8, 0, 6}

```

Transformation d'une liste en prolongeant d'une unité à droite et en ajoutant à chaque terme les dizaines de celui qui est placé à sa gauche, "technique dite des retenues"

```

x = {31, 12, 34}
x = pro[Length[x] + 1, x]
k = 1
While[k ≤ Length[x],
x[[k]] = x[[k]] + If[k > 1, Quotient[x[[k - 1]], 10], 0]; k = k + 1]

x
{31, 15, 35, 3}

```

Transformation d'une liste en ne conservant que les termes modulo 10

```

k = 1
While[k ≤ Length[x], x[[k]] = Mod[x[[k]], 10]; k = k + 1]
x
1
{1, 5, 5, 3}

```

Addition de deux listes de chiffres interprétées comme des nombres entiers en écriture décimale

```

Addéc[u_, v_] :=
(s = Som[inv[u], inv[v]]; s = pro[Length[s] + 1, s]; k = 1;
While[k ≤ Length[s], s[[k]] =
s[[k]] + If[k > 1, Quotient[s[[k - 1]], 10], 0]; k = k + 1];
k = 1; While[k ≤ Length[s], s[[k]] = Mod[s[[k]], 10]; k = k + 1];
inv[s])

```

```

Addéc[{9, 8, 1}, {9, 4, 0, 1}]

```

```

981 + 9401

```

```

{1, 0, 3, 8, 2}

```

```

10382

```

```

Addéc[{4, 5, 8}, {7, 9, 6, 4}]

```

```

458 + 7964

```

```

{0, 8, 4, 2, 2}

```

```

8422

```

```

Addéc[{5, 9, 8, 3, 9, 7, 5, 6, 3, 1, 0, 6}, {8, 1, 0, 6, 4, 3, 9}]

```

```

598397563106 + 8106439

```

```

{0, 5, 9, 8, 4, 0, 5, 6, 6, 9, 5, 4, 5}

```

```

598405669545

```

Immédiat

V.2. Seconde récursivité

Le recours à une programmation avec des "-1", ce que certains appellent une "récurrence descendante", permet, au moins dans certains cas, de programmer la deuxième récursivité.

V.3.1. Coefficients du binôme

Méthode 2-récursive

```

cb[0, k_] := If[k > 0, 0, 1]

```

```

cb[n_Integer /; n > 0, k_] := Sg[k] × cb[n - 1, k - 1] + cb[n - 1, k]

```

```
{cb[2, 1], cb[3, 2], cb[4, 2]}
{2, 3, 6}
```

```
cb[21, 9]
$Aborted
```

Aucune réponse après plus d'une minute

```
Quotient[21!, 9! × 12!]
293930
```

Immédiat

```
cb[256, 5]
$RecursionLimit::reclim : Recursion depth of 256 exceeded.
```

La méthode limitée à 256[] retombe par là même dans un algorithme primitif récursif

Méthode primitive récursive avec listes et Nest

```
Table[If[And[1 < k, k ≤ Length[l]], l[[k]] + l[[k - 1]], 1],
      {k, Length[l] + 1}]
{1}

g[l_] := Table[If[And[1 < k, k ≤ Length[l]],
                  l[[k]] + l[[k - 1]], 1], {k, Length[l] + 1}]

lcb[n_] := Nest[g, {1}, n]
lcb[6]
{1, 6, 15, 20, 15, 6, 1}
```

Presque immédiat

```
Cb[n_, k_] := lcb[n][[k + 1]]
{Cb[2, 1], Cb[3, 2], Cb[4, 2]}
{2, 3, 6}
```

Immédiat

```
Cb[21, 9]
293930
```

Immédiat

```
Cb[256, 5]
```

```
8809549056
```

Immédiat

V.3.2. Fonction d'Ackermann

Calcul à l'aide d'un seul programme par récurrence descendante

```
Ac[0, x_Integer /; x ≥ 0] = x + 1
Ac[n_Integer /; n > 0, 0] := Ac[n - 1, 1]
Ac[n_Integer /; n > 0, x_Integer /; x > 0] :=
  Ac[n, x] = Ac[n - 1, Ac[n, x - 1]]
```

```
1 + x
```

```
{Ac[0, 5], Ac[1, 0], Ac[2, 1], Ac[3, 1],
  Ac[3, 2], Ac[3, 3], Ac[3, 6], Ac[4, 0]}
```

```
{6, 2, 5, 13, 29, 61, 509, 13}
```

Immédiat

```
Ac[4, 1]
```

```
$RecursionLimit::reclim : Recursion depth of 256 exceeded.
```

```
$Aborted
```

La méthode limitée à 256[] retombe par là même dans un algorithme primitif récursif

Ackermann calculée par une suite de fonctions primitives récursives

```
Ac0[x_] = x + 1
```

```
Ac1[x_] := Nest[Ac0, 1, x + 1]
```

```
1 + x
```

```
Ac2[x_] := Nest[Ac1, 1, x + 1]
```

```
Ac3[x_] := Nest[Ac2, 1, x + 1]
```

```
Ac4[x_] := Nest[Ac3, 1, x + 1]
```

```
{Ac0[5], Ac1[0], Ac2[1], Ac3[1], Ac3[2], Ac3[3], Ac3[6], Ac4[0]}
```

```
{6, 2, 5, 13, 29, 61, 509, 13}
```

10 secondes

```
Ac4[1]
```

```
$Aborted
```

Aucune réponse après plus de 5 minutes.

$Ac4[1] = Ac3[Ac3[1]] = Ac3[13]$

On montre facilement $A3[x] = 2^{x+3} - 3$

Ack3[x_] := 2^{x+3} - 3

Ack3[13]

65533

Immédiat

V.3.3. Fonction d'Ackermann-bis

Fonction $h[k, x]$ auxiliaire d'initialisation de $Ab[k+1, x, 0]$

h[n_, x_] := If[n < 1, 1, 0] × x + If[1 < n, 1, 0]

{h[0, x], h[1, 3], h[3, 5]}

{x, 0, 1}

Calcul à l'aide d'un seul programme par "récurrence descendante"

Ab[0, x_, y_] := y + 1

Ab[n_Integer /; n > 0, x_, 0] := h[n - 1, x]

**Ab[n_Integer /; n > 0, x_Integer /; x ≥ 0,
y_Integer /; y > 0] := Ab[n - 1, x, Ab[n, x, y - 1]]**

Ab[0, x, y]

1 + y

{Ab[1, 3, 5], Ab[1, 24, 18]}

{8, 42}

{Ab[2, 3, 5], Ab[2, 4, 7]}

{15, 28}

{Ab[3, 2, 4], Ab[3, 3, 5]}

{16, 243}

{Ab[4, 3, 2], Ab[4, 2, 3]}

{27, 16}

Ab[4, 3, 3]

\$RecursionLimit::reclim : Recursion depth of 256 exceeded.

La méthode limitée à 256[] retombe encore, par là même, dans un algorithme primitif récursif

Ackermann-bis calculée par une suite de fonctions primitives récursives

```

Ab0[x_, y_] := y + 1
Ab1[x_, y_] := Nest[Ab0[x, #] &, x, y]
Ab2[x_, y_] := Nest[Ab1[x, #] &, 0, y]
Ab3[x_, y_] := Nest[Ab2[x, #] &, 1, y]
Ab4[x_, y_] := Nest[Ab3[x, #] &, 1, y]

{Ab1[3, 5], Ab1[24, 18]}
{8, 42}

{Ab2[3, 5], Ab2[4, 7]}
{15, 28}

{Ab3[2, 4], Ab3[3, 5]}
{16, 243}

{Ab4[3, 2], Ab4[2, 3]}
{27, 16}

Ab4[3, 3]
$Aborted

```

Aucune réponse après 4minutes

On montre facilement: $\text{Ab}[4, 3, 3] = \text{Ab}[3, 3] = 3^{27}$

3^{27}

7625597484987

Immédiat

VI. μ -récursivité et While

Nombre de chiffres en écriture décimale d'un entier naturel n

Sans test "borne"

```

nbchi[n_] := (k = 0; While[ $10^{\mathbf{k}} \leq \mathbf{n}$ , k = k + 1]; Print[k])

```

```
nbchi[3]; nbchi[14]; nbchi[99]; nbchi[100];
nbchi[2624]; nbchi[347210986549876190053]
```

```
1
2
2
3
4
21
```

Avec test "borne"

```
nbch[n_] := (k = 0; While[And[10k ≤ n, k ≤ n], k = k + 1]; Print[k])
```

```
nbch[3]; nbch[14]; nbch[99]; nbch[100];
nbch[2624]; nbch[347210986549876190053]
```

```
1
2
2
3
4
21
```

Immédiat dans les deux cas

VI.I.2. Quotient de la division euclidienne en \square -récursivité

Sans test "borne"

```
quotmur[a_, b_] :=
  (q = 0; While[b × (q + 1) ≤ a, q = q + 1]; Print[q])
```

```
quotmur[3, 2]
```

```
1
```

```
quotmur[29, 7]
```

```
4
```

```
quotmur[381, 17]
```

```
22
```

Immédiats

```
quotmur[4, 0]
```

```
$Aborted
```

Tourne sans fin

Avec test borne

```
quotmub[a_, b_] :=
  (q = 0; While[And[b × (q + 1) ≤ a, q ≤ a], q = q + 1]; Print[q])
```

```
quotmub[3, 2]
```

```
1
```

```
quotmub[4, 0]
```

```
5
```

```
quotmub[29, 7]
```

```
4
```

```
quotmub[381, 17]
```

```
22
```

Tous immédiats

VI.2.

Malgré le risque de tourner sans fin, **While** semble plus simple à programmer et par là même plus attractif pour l'utilisateur. Pour éviter ce risque, il y a toujours moyen, lorsque ça se termine, de transformer en la conjonction d'une minimisation bornée et d'une primitive récursivité par l'introduction d'un double test, le deuxième étant simplement un test de majoration. Aussi, je préconise pour ma part d'introduire un nouveau **While** à trois places [test, borne, expression ou sous-programme] : voir exemple du nombre de chiffres en écriture décimale et du quotient ci-dessus et encore d'autres exemples ci-dessous.

Primitive récursivité à l'aide de **While**

Le premier test disparaît, seul le test de borne subsiste, on a ainsi une nouvelle façon de faire de l'itération pure ou même de la primitive récursivité.

Exponentielle de base 2

```
exp2[n_] := (k := 0; u = 1; While[k < n, k = k + 1; u = u × 2]; Print[u])
```

```
exp2[10]
```

```
1024
```

Factorielle

La suite auxiliaire définie à l'aide de `u` et de `Nest` (itération pure) explicite le phénomène.

```
u[k_] := If[Mod[k, 2] == 0, Quotient[k, 2], 3 k - 1]
```

```
NestList[u, 5, 11]
```

```
{5, 14, 7, 20, 10, 5, 14, 7, 20, 10, 5, 14}
```

```
NestList[u, 3, 11]
```

```
{3, 8, 4, 2, 1, 2, 1, 2, 1, 2, 1, 2}
```

```
NestList[u, 7, 11]
```

```
{7, 20, 10, 5, 14, 7, 20, 10, 5, 14, 7, 20}
```

En supprimant le test de borne dans `term`, la suite `ter` ci-dessous, est partielle récursive, tourne sans fin pour 5, 7 mais elle fournit la réponse {nombre de pas,1} pour : 1, 2, 3, 4, 6 .

```
ter[n_] := (p = 0; k = n; While[k > 1, p = p + 1; If[Mod[k, 2] == 0,  
k = Quotient[k, 2], k = 3 k - 1]]; Print[{p, k}])
```

```
ter[1]
```

```
{0, 1}
```

```
ter[2]
```

```
{1, 1}
```

```
ter[2]
```

```
{1, 1}
```

```
ter[3]
```

```
{4, 1}
```

```
ter[4]
```

```
{2, 1}
```

```
ter[5]
```

```
$Aborted
```

```
ter[6]
```

```
{5, 1}
```

```
ter[7]
```

```
$Aborted
```