Κεφάλαιο 3ο: Οι φυσικοί και οι ακέραιοι αριθμοί

3.1 Οι αριθμοί στο *Mathematica*

Το *Mathematica* δεν αναγνωρίζει το 7. σαν κάποιο ακέραιο αριθμό. Με την Head μπορούμε να δούμε την επικεφαλίδα ενός αριθμού.

```
Head[7.]
```

```
Real
```

Υπάρχουν πολλές άλλες δυνατότητες για αριθμούς όπως Integer, Rational, Complex π.χ

```
{Head[3], Head[1/3], Head[0.3], Head[0.3 + i]}
```

```
{Integer, Rational, Real, Complex}
```

Οι πράξεις εκτελούνται με διαφορετικό τρόπο σε κάθε περίπτωση. Έτσι όταν δουλεύομε με ακέραιους αριθμούς πρέπει κάθε φορά να σιγουρευόμαστε ότι οι μεταβλητές μας και οι σταθερές μας είναι ακέραιοι αριθμοί στις εμπλεκόμενες σχέσεις π.χ όταν θελήσουμε να ορίσουμε το παραγοντικό για ακεραίους θα γράψουμε:

```
paragontiko[n_Integer] := (n)!
```

Με το n_Integer αναγκάζουμε την μεταβλητή n να παίρνει τιμές με επικεφαλίδα Integer δηλ. μόνο ακέραιες τιμές -αλλιώς δεν θ προκύψει αποτέλεσμα.

```
{paragontiko[4], paragontiko[4.], (4.5)!}
```

```
{paragontiko[4], paragontiko[4.], 52.3428}
```

3.2 Οι Διαιρέτες ενός ακεραίου αριθμού

Η συνάρτηση Divisors βρίσκει όλους τους θετικούς διαιρέτες ενός ακεραίου.

```
Divisors[-72]
```

```
{1, 2, 3, 4, 6, 8, 9, 12, 18, 24, 36, 72}
```

Αν θέλουμε μόνο τους πρώτους διαιρέτες απ' αυτούς τότε γράφουμε

```
Select[%, PrimeQ]
```

```
{2, 3}
```

Με το PrimeQ ελέγχουμε αν ένας αριθμός είναι πρώτος ή όχι:

```
{PrimeQ[12], PrimeQ[-3], PrimeQ[7.]}
```

```
{False, True, False}
```

Απο την απάντηση βλέπουμε ότι το 7. και το 12 δεν είναι πρώτοι ενώ το -3 θεωρείται πρώτος!

3.3 Παραγοντοποίηση ενός ακεραίου αριθμού

Αν θέλουμε τους πρώτους παράγοντες μαζί με τους εκθέτες τους στο ανάπτυγμα ενός ακεραίου τότε γράφουμε

```
FactorInteger[140]
```

```
{{2, 2}, {5, 1}, {7, 1}}
```

Το {2,2) σημαίνει 2^2 και όμοια το {5, 1} σημαίνει 5^1 στην ανάλυση του 140. Για αρνητικούς παίρνουμε μπροστά και το {-1,1} για παράδειγμα

```
FactorInteger[-140]
```

```
{{-1, 1}, {2, 2}, {5, 1}, {7, 1}}
```

Τώρα θα ορίσουμε μια συνάρτηση ώστε να εμφανίζεται μονάχα ο μέγιστος πρώτος διαιρέτης ενός ακεραίου. Για τον σκοπό αυτό κατασκευάζουμε την maxPrimeDivisor

```
maxPrimeDivisor[x_Integer] := First[Last[FactorInteger[x]]]
```

```
maxPrimeDivisor[14]

7
```

Η χρήση των First και Last φαίνεται στα παραδείγματα που ακολουθούν.

```
u = {1, 5, 7, 7, 8, -1}

{1, 5, 7, 7, 8, -1}
```

```
Last[u](* το πρώτο από δεξιά*)

-1
```

```
First[u](* το πρώτο από αριστερά*)

1
```

3.4 Η συνάρτηση Flatten

Ένας άλλος τρόπος για την εύρεση των πρώτων διαιρετών θα ήταν να απομονώσουμε τους πρώτους που βρίσκονται μέσα στην λίστα FactorInteger. Αυτό μπορεί να γίνει είτε με αναστροφή(Transpose) του πίνακα FactorInteger είτε διαλέγοντας μόνο τις πρώτες συντεταγμένες από κάθε ζευγάρι του FactorInteger με την χρήση της Part.

Η Part[expr,i] δίνει το i μέρος της expr ενώ με Part[expr,i,j] παίρνουμε το j μέρος του i.

```
m = FactorInteger[15]
MatrixForm[m](* Ο m σε μορφή πίνακα*)
anastrofos = Transpose[m](*αναστροφή του m *)
First[anastrofos](*Παίρνουμε την 1 η γραμμή *)


{{3, 1}, {5, 1}}
```

$$\begin{pmatrix} 3 & 1 \\ 5 & 1 \end{pmatrix}$$

```
{{3, 5}, {1, 1}}
```

```
{3, 5}
```

Άλλος τρόπος με την χρήση της Part

```
Part[m, {1}](* η πρώτη γραμμή του m *)
Part[m, {1}, {1}]
(* το στοιχείο της 1 ης γραμμής, 1 ης στήλης *)
Part[m, All, {1}](* η 1 η στήλη του m-
  αυτή φυσικά περιέχει τους πρώτους που μας ενδιαφέρουν *)


{{3, 1}}
```

```
{{3}}
```

```
{{3}, {5}}
```

Άρα οι πρώτοι διαιρέτες είναι το 3 και το 5. Αν δεν μας αρέσουν οι πολλές αγκύλες μπορούμε να τις απλοποιήσουμε με την χρήση της Flatten.

```
Flatten[%]

{3, 5}
```

```
? Flatten
```

```
Flatten[list] flattens out nested lists.
  Flatten[list, n] flattens to level n. Flatten[
  list, n, h] flattens subexpressions with head h.
```

Δηλ. η Flatten[a] μας δίνει σε μια λίστα τα στοιχεία της α ανεξάρτητα σε ποιό επίπεδο αυτά βρίσκονται, ενώ η Flatten[a,2] μας δίνει μόνο τα στοιχεία έως και 2ου επιπέδου..Π.χ

```
Flatten[{1, 2, 3, {4, 5, 6}, {7, {8, 9}}, {{10}, {{11}, 12}}}]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

```
Flatten[{1, 2, 3, {4, 5, 6}, {7, {8, 9}}, {{10}, {{11}, 12}}}, 2]
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, {11}, 12}
```

Όλες οι παραπάνω πράξεις μπορούν να γραφούν με μία μόνο συνάρτηση:

```
diairetes[n_Integer] := Flatten[Part[FactorInteger[n], All, {1}]]
```

3.5 Οι πρώτοι αριθμοί

Αν ζητάμε να βρούμε τους πρώτους<=10 τότε

```
Select[Range[10], PrimeQ]
```

```
{2, 3, 5, 7}
```

ενώ αν ζητάμε τους 10 πρώτους στη σειρά τότε

```
Prime[Range[10]]
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
```

Η συνάρτηση Prime[x] παραπάνω μας δίνει το x -ιοστό πρώτο. Με ?Prime μπορούμε να μάθουμε πληροφορίες.

```
Prime[300]
```

```
1987
```

```
? Prime
```

```
Prime[n] gives the nth prime number.
```

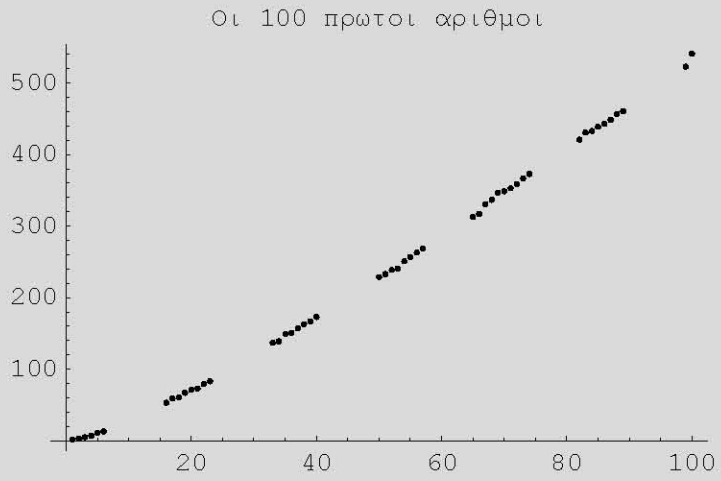
Οι 100 πρώτοι αριθμοί δίνονται με την μορφή πίνακα.

```
first100prwtoi = Table[{n, Prime[n]}, {n, 100}]
```

```
{{1, 2}, {2, 3}, {3, 5}, {4, 7}, {5, 11}, {6, 13}, {7, 17}, {8, 19},
 {9, 23}, {10, 29}, {11, 31}, {12, 37}, {13, 41}, {14, 43},
 {15, 47}, {16, 53}, {17, 59}, {18, 61}, {19, 67}, {20, 71},
 {21, 73}, {22, 79}, {23, 83}, {24, 89}, {25, 97}, {26, 101},
 {27, 103}, {28, 107}, {29, 109}, {30, 113}, {31, 127}, {32, 131},
 {33, 137}, {34, 139}, {35, 149}, {36, 151}, {37, 157}, {38, 163},
 {39, 167}, {40, 173}, {41, 179}, {42, 181}, {43, 191}, {44, 193},
 {45, 197}, {46, 199}, {47, 211}, {48, 223}, {49, 227},
 {50, 229}, {51, 233}, {52, 239}, {53, 241}, {54, 251}, {55, 257},
 {56, 263}, {57, 269}, {58, 271}, {59, 277}, {60, 281},
 {61, 283}, {62, 293}, {63, 307}, {64, 311}, {65, 313},
 {66, 317}, {67, 331}, {68, 337}, {69, 347}, {70, 349}, {71, 353},
 {72, 359}, {73, 367}, {74, 373}, {75, 379}, {76, 383}, {77, 389},
 {78, 397}, {79, 401}, {80, 409}, {81, 419}, {82, 421}, {83, 431},
 {84, 433}, {85, 439}, {86, 443}, {87, 449}, {88, 457}, {89, 461},
 {90, 463}, {91, 467}, {92, 479}, {93, 487}, {94, 491}, {95, 499},
 {96, 503}, {97, 509}, {98, 521}, {99, 523}, {100, 541}}
```

Η γραφική παράσταση των παραπάνω σημείων είναι:

```
ListPlot[first100prwtoi, PlotLabel -> "Οι 100 πρωτοι αριθμοι"]
```




```
- Graphics -
```

3.6 Η Ευκλείδια Διαίρεση

Είναι γνωστό ότι αν α, β είναι ακέραιοι με β μη μηδενικός τότε υπάρχουν μοναδικοί ακέραιοι q=Quotient[α,β](το πηλίκο της διαίρεσης) m=Mod[α,β] (το υπόλοιπο) έτσι ώστε α=qβ+m. Το q θα μπορούσε να γραφεί συναρτήσει των α,β ως εξής $q=\text{sgn}\beta\times\left[\frac{\alpha}{|\beta|}\right]$ όπου με sgnβ συμβολίζουμε το πρόσημο $\frac{\beta}{|\beta|}$ του β και με [...] το ακέραιο μέρος ενός πραγματικού αριθμού. Βέβαια το ακέραιο μέρος στο *Mathematica* έχει την έννοια του μέρους που βρίσκεται αριστερά απο το δεκαδικό μέρος του π.χ

```
IntegerPart[-Pi]
```

```
-3
```

Ας μελετήσουμε λίγο την συνάρτηση Mod[α,β]. Ως το υπόλοιπο της διαίρεσης του α με το β πρέπει να είναι πάντα θετικό ή 0 και μικρότερο της απόλυτης τιμής του β.

```
Mod[-31, 5]
```

```
4
```

Το πηλίκο όμως μπορεί να είναι αρνητικό.

```
Quotient[-31, 5]
```

```
-7
```

Φυσικά, τα νούμερα που βρήκαμε επαληθεύουν την ταυτότητα της διαίρεσης.

```
-31 == 5 Quotient[-31, 5] + Mod[-31, 5]

True
```

Γενικά το *Mathematica* μας δίνει δυνατότητες να ελένξουμε(για απλές τουλάχιστον περιπτώσεις) αν ισχύουν κάποιες ταυτότητες π.χ

```
Sum[k^2, {k, 1, n}] == n (n + 1) (2 n - 1) / 6

1/6 n (1 + n) (1 + 2 n) == 1/6 n (1 + n) (-1 + 2 n)
```

Δεν έβγαλε true. Άρα μάλλον εμείς έχουμε γράψει στο δεξιό σκέλος της παραπάνω ισότητας λανθασμένα ή το *Mathematica* αδυνατεί να μας το αποδείξει . Ας το διορθώσουμε το 2n-1 σε 2n+1:

```
Sum[k^2, {k, 1, n}] == n (n + 1) (2 n + 1) / 6

True
```

Θα πρέπει να θυμούμαστε πάντα ότι το *Mathematica* μπορεί να κάνει πράξεις και να χειρίζεται σύμβολα αλλά δεν μπορεί να κάνει μαθηματικά δηλ. δεν μπορεί να μας αποδείξει μια ταυτότητα π.χ με επαγωγή. Απλώς μπορούμε να διαπιστώσουμε αν πράγματι ισχύει η ταυτότητα δοκιμάζοντας την με κάποιους αριθμούς n χωρίς βέβαια αυτό να αποτελεί απόδειξη. Π.χ για n απο ένα μέχρι το 10 βλέπουμε ότι η παραπάνω ταυτότητα αληθεύει

```
Table[Sum[k^2, {k, 1, n}] == n (n + 1) (2 n + 1) / 6, {n, 1, 10}]

{True, True, True, True, True, True, True, True, True, True}
```

3.7 Η εικασία του Goldbach

Το 1742 ο Christian Goldbach διατύπωσε την εικασία ότι κάθε άρτιος θετικός ακέραιος μεγαλύτερος του 4 μπορεί να γραφεί σαν άθροισμα δύο πρώτων αριθμών. Το *Mathematica* δεν μπορεί βέβαια να δώσει την απόδειξη της εικασίας αυτής, μπορεί όμως να μας βοηθήσει να βρούμε ζεύγη πρώτων αριθμών που το άθροισμα τους είναι δεδομένο. Παρακάτω ορίζουμε την συνάρτηση goldbach για αυτόν το σκοπό. Θα πρέπει να αναφέρουμε ότι δεξιά του /; μπαίνουν οι συνθήκες που πρέπει οποσδήποτε να ικανοποιούνται. Με EvenQ[n] ελέγχουμε αν ο n είναι άρτιος και με Positive[n] αν είναι θετικός. Για την συνάρτηση For θα πούμε αργότερα κάποια πράγματα. Για τώρα μπορείται να μάθετε πληροφορίες με το ?For.

```
goldbach[n_Integer /; EvenQ[n] && Positive[n] && n > 4] :=
 Module[{a = {}, i},
  For[i = 2, Prime[i] ≤ n / 2, i++, If[PrimeQ[n - Prime[i]],
    a = Append[a, {Prime[i], n - Prime[i]}]]]; a]
```

```
goldbach[2000]

{{3, 1997}, {7, 1993}, {13, 1987}, {67, 1933}, {127, 1873},
 {139, 1861}, {199, 1801}, {211, 1789}, {223, 1777}, {241, 1759},
 {277, 1723}, {307, 1693}, {331, 1669}, {337, 1663}, {373, 1627},
 {379, 1621}, {421, 1579}, {433, 1567}, {457, 1543}, {541, 1459},
 {547, 1453}, {571, 1429}, {577, 1423}, {601, 1399}, {619, 1381},
 {673, 1327}, {709, 1291}, {751, 1249}, {769, 1231},
 {787, 1213}, {829, 1171}, {877, 1123}, {883, 1117},
 {907, 1093}, {937, 1063}, {967, 1033}, {991, 1009}}
```

Το a είναι η λίστα που περιέχει όλα τα δυνατά ζεύγη πρώτων στα οποία η 1η συντεταγμένη είναι <= της 2ης. Αρχικά είναι κενή (a={ })και κάθε φορά που βρίσκουμε ένα ζευγάρι πρώτων με άθροισμα n το επισυνάπτουμε με την Append στην a. Το Module δηλώνει ένα σύνολο "τοπικών" (local)εντολών δηλ. που εκτελούνται μόνο όταν καλείται η συνάρτηση goldbach. Το Module ξεκινάει πάντα με την δήλωση των "τοπικών μεταβλητών" και των αρχικών τιμών που πιθανόν να έχουν. Στο παραπάνω Module οι τοπικές μεταβλητές είναι η a και η i.

Module[{x, y, ... }, expr] specifies that occurrences of the symbols x, y, ... in expr should be treated local. Module[{x = x0, ... }, expr] defines initial values for x, ...

3.8 Ο μέγιστος κοινός διαρέτης και το ελάχιστο κοινό πολλαπλάσιο

Με GCD παίρνουμε τον μέγιστο κοινό διαιρέτη κάποιων ακεραίων. Π.χ

```
{GCD[24, 14, 8], GCD[-24, 14, 8], GCD[24., 14, 8]}
```

```
GCD::exact : Argument 24.` at position 3 is not an exact number.
```

```
{2, 2, GCD[8, 14, 24.]}
```

Βλέπουμε ότι το GCD δεν δουλεύει για πραγματικούς και βγάζει μήνυμα λάθους. Το GCD[x,y,z] είναι πάντα ίσο με GCD[GCD[x,y],z] και μπορούμε να το διαπιστώσουμε εύκολα:

```
{GCD[x, y, z] == GCD[GCD[x, y], z], GCD[x, y] == GCD[y, Mod[x, y]]}
```

```
{True, GCD[x, y] == GCD[y, Mod[x, y]]}
```

Παρατηρήστε ότι για την μια ταυτότητα δίνει True ενώ για την άλλη(που είναι και αυτή αληθινή) δεν δίνει απάντηση!
Για διδακτικούς λόγους θα ορίσουμε τώρα το GCD[x,y,z] ως τον μεγαλύτερο απο τους κοινούς διαρέτες των x,y,z. Ας βρούμε για παράδειγμα το GCD[24,14,8]. Βρίσκουμε πρώτα ποιοί διαιρέτες είναι οι κοινοί χρησιμοποιώντας την συνάρτηση Intersection(κοινή τομή):

```
Intersection[Divisors[24], Divisors[14], Divisors[8]]
```

```
{1, 2}
```

Στην συνέχεια παίρνουμε το τελευταίο στοιχείο που εμφανίζεται στην τομή. Εδώ όπως βλέπουμε συμπίπτει με το μεγαλύτερο στοιχείο της τομής(δηλ. το 2) που είναι και ο ζητούμενος Μ.Κ.Δ.

```
Last[%]
```

```
2
```

Μπορούμε να βάλουμε όλα τα παραπάνω μέσα σε μια και μόνο συνάρτηση:

```
megkoinosDiairetis[m_Integer, n_Integer] :=
 Last[Intersection[Divisors[m], Divisors[n]]]
```

```
megkoinosDiairetis[24, 14]
```

```
2
```

Μπορεί επίσης να δοθεί και ένας αναδρομικός ορισμός της GCD χρησιμοποιώντας την ταυτότητα:

```
GCD[m, n] == GCD[n, Mod[m, n]]
```

```
GCD[m, n] == GCD[n, Mod[m, n]]
```

Ο αναδρομικός ορισμός θα δοθεί μετην myGCD στην επόμενη ενότητα. Όπως ήδη αναφέραμε το *Mathematica* αδυνατεί να αποδείξει την ταυτότητα παρόλο που για κάθε συγκεκριμένες τιμές ισχύει π.χ.

```
GCD[24, 14] == GCD[24, Mod[24, 14]]

True
```

Το ελάχιστο κοινό πολλαπλάσιο κάποιων αριθμών βρίσκεται με το LCM.

```
LCM[-2, -3, -5]

30
```

Φυσικά μπορούμε να το ορίσουμε χρησιμοποιώντας το GCD πού ήδη είδαμε ως εξής:

```
myLCM[m_, n_] := Abs[m * n] / GCD[m, n]
```

```
myLCM[-2, -3]

6
```

Το Abs δίνει την απόλυτη τιμή.

3.9 Αναδρομικοί ορισμοί στο *Mathematica*

Κάθε αναδρομικός ορισμός αποτελείται απο τις αρχικές(οριακές)συνθήκες και απο το κυρίως σώμα. Π.χ

```
myGCD[m_, 0] := m
myGCD[m_, n_] := myGCD[n, Mod[m, n]]
```

```
myGCD[24, 14]

2
```

Χρειάζεται ιδιαίτερη προσοχή όταν ορίζουμε αναδρομές. Έτσι για παράδειγμα αν θα τρέξουμε το myGCD[4.,2] θα πέσουμε σε ατέρμονα loop!(Φταίει ότι το 4. δεν θεωρείται ακέραιος απο το *Mathematica*) . Για να σταματήσει να τρέχει το πρόγραμμα πατάμε Alt +) Θα έπρεπε τα m,n να δηλωθούν ως ακέραιοι: myGCD[m_Integer,0]:=m και myGCD[m_Integer,n_Integer]:= myGCD[n,Mod[m,n]]. Ένας ισοδύναμος τρόπος για να δηλωθούν τα m και n ως ακέραιοι είναι ο παρακάτω

```
Clear[myGCD]
myGCD[m_, 0] /; IntegerQ[m] := m
myGCD[m_, n_] /; IntegerQ[n] && IntegerQ[m] :=
 myGCD[n, Mod[m, n]]
```

```
{myGCD[24, 14], myGCD[24., 14]}

{2, myGCD[24., 14]}
```

Το /; δηλώνει τις συνθήκες που πρέπει να ικανοποποιούνται απαραίτητα για να ισχύει ο ορισμός. Το IntegerQ[n] δίνει true αν το n είναι ακέραιος. Το && είναι το σύμβολο της σύζευξης στο *Mathematica*. Το myGCD[24.,14]δεν υπολογίσθηκε διότι το 24. είναι Real.

Ένα άλλο παράδειγμα αναδρομικού ορισμού μπορεί να δοθεί για την κατασκευή μιας ακολουθίας. Π.χ η ακολουθία 1,1,2,3,5,.. που κάθε όρος της είναι το άθροισμα των δύο προηγούμενων λέγεται ακολουθία του Fibonacci και δίνεται στο *Mathematica* με την χρήση της ενσωματωμένης συνάρτησης Fibonacci π.χ.

```
{Fibonacci[1], Fibonacci[2],
 Fibonacci[3], Fibonacci[4], Fibonacci[5]}

{1, 1, 2, 3, 5}
```

Ένας αναδρομικός ορισμός θα μπορούσε να είναι ο

```
Remove[myFib]
myFib[1] := 1;(*  πρώτη αρχική συνθήκη *)
myFib[2] := 1;  (*δεύτερη αρχική συνθήκη *)
myFib[n_Integer] := myFib[n - 1] + myFib[n - 2]
```

```
myFib[4]

3
```

Ο παραπάνω αναδρομικός ορισμός δεν είναι και ιδιαίτερα καλός διότι θα αναγκαστεί ο υπολογιστής να κρατησει πολλές φορές κάθε προηγούμενο όρο. Αυτό το διαπιστώνουμε χρησιμοποιώντας την εντολή Trace.

```
Trace[myFib[5]]

{myFib[5], myFib[5 - 1] + myFib[5 - 2],
 {{5 - 1, 4}, myFib[4], myFib[4 - 1] + myFib[4 - 2],
  {{4 - 1, 3}, myFib[3], myFib[3 - 1] + myFib[3 - 2],
   {{3 - 1, 2}, myFib[2], 1}, {{3 - 2, 1}, myFib[1], 1}, 1 + 1, 2},
  {{4 - 2, 2}, myFib[2], 1}, 2 + 1, 3}, {{5 - 2, 3}, myFib[3],
  myFib[3 - 1] + myFib[3 - 2], {{3 - 1, 2}, myFib[2], 1},
  {{3 - 2, 1}, myFib[1], 1}, 1 + 1, 2}, 3 + 2, 5}
```

Παρατηρούμε ότι το myFib[2] έχει εμφανιστεί 3 φορές στον υπολογισμο του myFib[5] όπως εύκολα βλέπουμε με το Trace[myFib[5],myFib[_]]

```
{Trace[myFib[5], myFib[_]],
 TimeConstrained[myFib[200], 1.5], Timing[Fibonacci[200]]}

{{myFib[5],
  {myFib[4], {myFib[3], {myFib[2]}, {myFib[1]}}, {myFib[2]}},
  {myFib[3], {myFib[2]}, {myFib[1]}}}, $Aborted,
 {0. Second, 280571172992510140037611932413038677189525}}
```

Ακόμα βλέπουμε ότι το myFib[200] είναι αδύνατον να υπολογιστεί σε λιγότερο απο 1.5 δευτερόλεπτα(αυτό σημαίνει το $Aborted) ενώ το Fibonacci[200] υπολογίστηκε σε 0. Second δηλ. σχεδόν σε μηδενικό χρόνο και να φανταστείται ότι το MyFib[200]=280571172992510140037611932413038677189525. Όλα αυτά δείχνουν καθαρά ότι ο ορισμός που δώσαμε δεν είναι τόσο γρήγορος. Για αυτό το λόγο επιλέγουμε ένα καλύτερο ορισμό τον newFib που υπολογίζει την ακολουθία από κάτω πρός τα πάνω δηλ. βρίσκουμε κάθε φορά την επόμενη τιμή ξεκινώντας απο τις αρχικές χωρίς επαναλήψεις υπολογισμών.

```
newFib[1] := 1;
newFib[n_] :=
 Module[{x = 0, y = 1}, Do[{x, y} = {y, x + y}, {n - 1}]; y]
```

```
Timing[newFib[200]]

{0. Second, 280571172992510140037611932413038677189525}
```

Τελειώνουμε με ένα παράδειγμα **κακού** αναδρομικού ορισμού. Οι κανόνες εκτελούνται με την σειρά που τους δίνομε. Δεν πρέπει να υπάρχουν επικαλύψεις των αρχικών συνθηκών όπως συμβαίνει π.χ με τον υπολογισμό του badgcd[0,0] ή να υπάρχει loop στο κυρίως σώμα της αναδρομής όπως συμβαίνει π.χ με τον υπολογισμό του badgcd[9,15]:= badgcd[Mod[9,15],15]] . Ένας κακός αναδρομικός ορισμός σίγουρα δεν θα δουλέψει!

```
badgcd[a_, 0] := a
badgcd[0, b_] := b
badgcd[a_, b_] := badgcd[Mod[a, b], b]
```

```
badgcd[24, 15]
```

```
$IterationLimit::itlim : Iteration limit of 4096 exceeded.
```

```
Hold[badgcd[Mod[9, 15], 15]]
```

Άσκηση: Προσπαθήστε να διορθώσετε όλα τα σφάλματα του παραπάνω αναδρομικού ορισμού.

3.10 Επίλυση Διοφαντικών εξισώσεων

Όταν λέμε διοφαντική εξίσωση εννοούμε μια συνηθισμένη γραμμική εξίσωση όπου οι άγνωστοι μπορούν να παάρουν μόνο ακέραιες λύσεις. Π.χ ax+by=c. Σε αυτέ τις περιπτώσεις για να υπάρχει λύση θα πρέπει απαραίτητα ο Μ.Κ.Δ των συντελεστών των αγνώστων να διαιρεί το c. Δηλ θα πρέπει με ορολογία του *Mathematica* το Mod[c,GCD[a,b]]=0. Π.χ για c=GCD[a,b] η εξίσωση ax+by=GCD[a,b] έχει σίγουρα μια ακέραια λύση {m,n} δηλ. am+bn=GCD[a,b]. Το *Mathematica* μας δίνει την δυνατότητα να βρούμε συγχρόνως το GCD[a,b] και μια ειδική λύση {m,n} της παραπάνω εξίσωσης. Η συνάρτηση για αυτό το σκοπό είναι η ExtendedGCD. π.χ

```
ExtendedGCD[12, 7, 245]
```

```
{1, {3, -5, 0}}
```

Δηλαδή ΜΚΔ=1 και η διοφαντική εξίσωση 12m+7n+245r=1 έχει μια λύση {m,n,r}={3,-5,0}. Φυσικά ίσως να υπάρχουν και άλλες ακέραιες λύσεις. Με την χρήση της If μπορούμε να τσεκάρουμε αν μια διοφαντική εξίσωση έχει λύση. Ακολουθούν κάποια παραδείγματα.

```
If[Mod[16, GCD[12, 8]] == 0,
 " η εξίσωση έχει ακέραια λύση", "καμμία ακέραια λύση"]
```

```
η εξίσωση έχει ακέραια λύση
```

```
If[Mod[16, GCD[12, 3, 15]] == 0,
 " η εξίσωση έχει ακέραια λύση", "καμμία ακέραια λύση"]
```

```
καμμία ακέραια λύση
```

Με άλλα λόγια η 16=12m+8n έχει ακέραιες λύσεις ενώ η 16=12x+3y+15z δεν έχει.
Ειδικά όταν έχουμε δυο αγνώστους - την x και y-τότε μπορούμε εύκολα να βρούμε την γενική λύση ώς

εξής: Ορίζουμε κατ'αρχήν d=GCD[a,b]. Έστω ότι d/c και m,n είναι μια ακέραια λύση της am+bn=d. Πολλαπλασιάζουμε και τα δύο μέλη με τον ρητό c/d και παίρνουμε $ax_0 + by_0 = c$ όπου $x_0 = (c/d)\,m$ και $y_0 = (c/d)\,n$. Τώρα μπορούμε να πάρουμε μια γενική ακέρια λύση σύμφωνα με τον τύπο $x = x_0 + bt$, $y = y_0 - at$ όπου t είναι μια παράμετρος με ακέραιες τιμές. Όλα τα παραπάνω υλοποιούνται ορίζοντας την συνάρτηση diofantine.

```
diofantine[a_Integer, b_Integer,
  c_Integer](* λύση της ax+by=c *) :=
 Module[{m, n}, Clear[t]; ({d, {m, n}} = ExtendedGCD[a, b];
   (*Τα m και n ικανοποιούν την  am+bn=
     d όπου d είναι ο ΜΚΔ των a,b *)If[Mod[c, d] == 0,
    {(c / d) m + b t, (c / d) n - a t}, "δεν έχει ακέραια λύση"])]
```

```
{diofantine[12, 7, 245], diofantine[12, 8, 245]}
```

```
{{735 + 7 t, -1225 - 12 t}, δεν έχει ακέραια λύση}
```

Με άλλα λόγια η 12x+7y=245 έχει λύσεις τις {735+7 t,-1225-12 t} ενώ η 12x+8y=245 δεν έχει.

```
m = diofantine[12, 7, 245]
m = m /. t → 100;
x = First[m]
y = Last[m]
12 x + 7 y == 245
```

```
{735 + 7 t, -1225 - 12 t}
```

```
1435
```

```
-2425
```

```
True
```

Το /. είναι σύμβολο της αντικατάστασης. Με m=m/.t→100 αντικαταστήσαμε την τμε 100 στο τύπο του m και το αποτέλεσμα (το ζευγάρι 1435, -2425 μπήκε στο m. Στην συνέχεια βρίσκουμε τα αντίστοιχα x,y και κάνουμε την δοκιμή. Το ερωτηματικό ; στην m=m/.t→100; σημαίνει ότι δεν θέλουμε να "εκτυπώσουμε" το m στην οθόνη μας αλλά απλώς να γίνουν οι πράξεις "σιωπηλά".

3.11 Λύνοντας εξισώσεις με modulus

Στην θεωρία αριθμών συναντάμε εξισώσεις και συστήματα με modulus. Για παράδειγμα 2x-3y+z=11(mod 5). Εδώ ζητάμε ακέραιες λύσεις για τα x,y,z που παίρνουν τιμές 0,1,2,3 και 4. Τέτοιες γραμμικές εξισώσεις αντιμετωπίζονται με την χρήση της LinearSolve ή της Solve όπως βλέπουμε στα παραδείγματα. Αν η εξίσωση ή οι εξισώσεις δεν είναι γραμμικές τότε θα πρέπει να χρησιμοποίσουμε την Solve.

```
(*Μια λύση της εξίσωσης 2 x-3 y+z=11 (mod 5) *)
a = {{2, -3, 1}}; b = {11}; LinearSolve[a, b, Modulus → 5]

{3, 0, 0}
```

```
Solve[2 x - 3 y + z == 11 && Modulus == 5, {x}]
(* ζητάμε η εξίσωση να λυθεί ως προς το x *)

{{Modulus → 5, y → 3 + 4 x + 2 z}}
```

```
Solve[2 x^2 - 3 y + z == 11 && Modulus == 5, {y}]
(* ΜΗ γραμμική. Εδώ ζητάμε η εξίσωση να λυθεί ως προς y*)

{{Modulus → 5, y → 3 + 4 x^2 + 2 z}}
```

Αν χρησιμοποιούμε την έκδοση 5 του *Mathematica* τότε θα μπορούσαμε να αντικαταστήσουμε την Solve με την Reduce και να γράφαμε.

```
Reduce[2 x^2 - 3 y + z == 11, {x, y, z}, Modulus -> 5]
```

Ειδικά σε περιπτώσεις της μορφής a*x-b=0(mod m) θα μπορούσαμε απλά να γράφαμε Solve[ax==b && Modulus==m, x] για να βρούμε το x. Άλλος τρόπος θα ήταν να πολλαπλασιάζαμε και τα δυο μέλη της a*x=b(mod m) με το d= a^{-1}(mod m) δηλ. με τον αντίστροφο του a . Στο *Mathematica* αυτό βρίσκεται με την συνάρτηση PowerMod δηλαδή d=PowerMod[a,-1,m]. Οπότε το ζητούμενο x=d*b(mod m). Για να λύσουμε για παράδειγμα την 5x=3(mod 7) γράφουμε

```
d = PowerMod[5, -1, 7];(* εύρεση του 5^-1 ( mod 7) *)
x = Mod[d 3, 7](* x= d*3 (mod 7)  *)

2
```

Τελειώνουμε με την περίπτωση που έχουμε σύστημα γραμμικών εξισώσεων με διαφορετικά mod και ένα άγνωστο π.χ $x \equiv 0 \bmod 4$, $x \equiv 1 \bmod 9$, και $x \equiv 2 \bmod 121$. Τότε μπορούμε να χρησιμοποίσουμε την Chinese-Remainder. Π.χ γράφοντας ChineseRemainder[{0, 1, 2}, {4, 9, 121}] όπου η λίστα {0,1,2} είναι οι σταθερές και {4,9,121} τα modulus παίρνουμε την λύση x=244

```
<<NumberTheory`NumberTheoryFunctions`
ChineseRemainder[{0, 1, 2}, {4, 9, 121}]

244
```

Προσέξτε ότι χρειάστηκε να καλέσουμε πρώτα το πακέτο <<NumberTheory`NumberTheoryFunctions` πριν καλέσουμε την ChineseRemainder.