PRiME COMA Computational Mathematics

18 november 2014

Oplossingen

Dit bestand bevat de oplossingen van de vraagstukken, uitgewerkt in *Sage*. De code is niet de elegantste noch de snelste code om de problemen op te lossen maar is gericht naar doorzichtigheid en volstaat om de coma binnen de gegeven tijd op te lossen.

1 The root of life, the universe and everything ($\sqrt{2}$ punten)

De waarde kan bepaald worden door een lus te schrijven, of eleganter:

```
sqrt(reduce(lambda x,y:  y+sqrt(x), [42,41..1])).n(digits=70)
```

Numerieke waarde

```
1.75793275661800453270881963821813852765309402164591977725585366439111
```

2 Een $500 vraagstuk ($\sqrt{3}$ punten)

Paul Erdös said about the Collatz conjecture: "Mathematics may not be ready for such problems." He also offered $500 for its solution.

Een lus volstaat weer om het probleem op te lossen:

```
jaren = 0;
dochters=3141594;
While[dochters > 1,
    if dochters% 2==0:
        dochters /= 2
    else:
        dochters = dochters*3+1
    jaren++;
]
jaren
```

Numerieke waarde

```
76
```

3 De slimste wiskundige ter wereld ($\sqrt{5}$ punten)

Beschouw een permutatie π. Dan geldt dat er voor iedere k ofwel een even aantal of een oneven aantal permutaties nodig zijn om k terug op zichzelf af te sturen:

$$\prod_{k=1}^{14}(2+(N_k \bmod 2)) = 2^n 3^m, \quad m+n=14 \tag{1}$$

Nu geldt dat gezien 14 een even getal is dat ook de machten n en m even moeten zijn. (want π moet een even aantal cykels met een even aantal elementen bevatten). Voor een bepaalde π zal het product dus een van de volgende vormen aannemen: 2^{14}, $2^{12}3^2$, $2^{10}3^4$, ..., $2^4 3^{10}$, $2^2 3^{12}$ of 3^{14}. We kunnen dit opdelen in 3 mogelijkheden:

- π bestaat enkel uit even cykels: $(1) = 2^{14}$
- π bestaat enkel uit oneven cykels: $(1) = 3^{14}$
- π bestaat uit een combinatie van even en oneven cykels: $(1) = 36M_\pi$

met M_π een constante die afhangt van de permutatie in kwestie.
Er geldt uit symmetrie dat er evenveel permutaties zijn bestaande uit enkel even cykels als permutaties die bestaan uit enkel oneven cykels. Eveneens geldt er dat:

$$\left(4^{2^{14}} + 4^{3^{14}}\right) \mod 1299 = 2$$

en dat

$$4^{36} \mod 1299 = 1.$$

Daaruit volgt ook dat $4^{36M_\pi} \mod 1299 = 1 \ \forall \pi$. Alles tezamen mogen we besluiten dat de som te herleiden is tot:

$$\left[\sum_{\pi\in S_{14}} 4^{\prod_{k=1}^{14}(2+(N_k \bmod 2))}\right] \bmod 1299 = \left[\sum_{\pi\in S_{14}} 1\right] \bmod 1299 = 14! \bmod 1299$$

Numerieke waarde

```
648
```

4 Zatte funties

($\sqrt{7}$ punten)

Beschouw de vectorruimte van de veeltermen over de reële getallen. Iedere operatie die toegepast wordt op de functie (zijnde een veelterm) is een lineaire operatie en kan dus gerepresenteerd worden door een matrix. Het totaal van alle operaties is dan de vermenigvuldiging van de matrices van de apparte operaties. Gezien matrix vermenigvuldiging enorm snel gaat (door overgang op een andere basis waarbij de matrix zo goed mogelijk wordt omgezet tot een diagonaal/Jordan-matrix kan het vermenigvuldigen van matrices met zichzelf even snel gaan als het kwadrateren van de diagonaal elementen). Hiermee voorkomt men overflows en/of lange berekeningstijden.

```
R = Integers(10^25)

def m_ afleider(deg):
    m= matrix(R, deg+1, deg+1)
    for i in range(deg):
         m[i,i+1] = i+1
    return m

def m_ translator(deg, a):
    m = Matrix(R, deg+1, deg+1)
    for i in [0..deg]:
         for j in [0..i]:
             m[j,i] = (-a)^(i-j)*binomial(i,j)
    return m

def m_ draaier(deg):
    m = Matrix(R, deg+1, deg+1)
    for i in [0..deg]:
         m[i,i] = 2*(i% 2)-1
    return m

afleider = m_ afleider(17)
translator_ r2 = m_ translator(17, 2)
translator_ l3 = m_ translator(17, -3)
draaier = m_ draaier(17)
afleider[17,0] = 1

for i in range(17):
```

```
    translator_ r2[i, 17] = 0
    translator_ r2[17,i] = 0

verband = translator_ l3*draaier*translator_ r2*afleider

def nde_ functie(f, n):
    f = R[x](f)
    fn = verband^n*vector(R, f.coeffs()+[0]*(17-f.degree()))
    return R[x](sum([R(fn[i])*R[x](x^i) for i in range(18)]))

% time
(nde_ functie(sum([x^i for i in range(18)]), 2^(19*53)+2081))(1)
```

Numerieke waarde

```
934013314081112170171158O
```

5 I used to be a banker but I lost interest ($\sqrt{11}$ punten)

We kunnen de operaties beschrijven met volgende matrices A, B en C, die werken op de kolommatrix $(R1, R2, R3)^T$ met daarop bedrag op de drie rekeningen. Specifiek hebben we:

$$A = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \quad C = \epsilon \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Een rekening beheren voor een totale duur van N jaar komt overeen met het specifieren van een 'woord' in de letters A, B en C dat N letters telt, bv.

$$w(A, B, C) = ABCCCABB \ldots B.$$

Merk eerst op dat $CA = AC$ en $CB = BC$ zodat we alle optredens van de matrix C vooraan kunnen groeperen. We kunnen dus zonder verlies aan algemeenheid stellen dat het optimum gerealiseerd wordt met een woord van de vorm

$$w(A, B, C) = C^x w(A, B)$$

Beschouw nu de commutator $T = A^1 B^1 AB$, dus T voldoet aan $BAT = AB$. We zullen de matrix T gebruiken om koppels A, B van volgorde te verwisselen. (Dit is de groepstheoretische commutator en die bestaat omdat A en B inverteerbaar zijn: hun

determinant is 1; deze verschilt van de matrix-gewijze commutator $AB - BA$ die we verder niet gebruiken.) Merk op dat

$$T = \begin{bmatrix} 1 & 0 & ab \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

en we kunnen meteen narekenen dat $TA = AT$ en $TB = BT$. We claimen:
CLAIM. $w(A, B) = B^j A^i T^s$, waarin i het aantal keer is dat A voorkomt, j het aantal keer dat B voorkomt en s het aantal koppels (A, B) in het woord w waarbij de A voor de B komt. (I.h.b. is $s \leq ij$).
Voorbeeld: $ABAB = BATAB = BAABT = BABATT = BBAATTT = B^2A^2T^3$, waarbij de koppels gegeven zijn door ABAB, ABAB, ABAB.

Bewijs. We bewijzen dit met inductie op het aantal koppels van letters B, A dat voorkomt in het woord $w(A, B)$ waarbij A voor de B komt. (Niet noodzakelijk vlak ervoor.) Onderstel voor de inductiebasis dat dit aantal 0 is, dan is $w(A, B)$ van de vorm $B^j A^i$ en is de claim triviaal waar. Stel nu dat het gestelde bewezen is tot aan $s - 1$ dergelijke koppels. Neem nu een willekeurig koppel AB waarbij de A bovendien vlak voor de B komt. Vervang dit koppel door BAT en merk op dat vermits T commuteert met zowel A als B we de T helemaal achteraan kunnen laten. Het overgebleven woordt heeft precies een koppel A, B minder dan het vorige woord; dus kunnen we de inductiehypothese toepassen en we zijn klaar:

$$\cdots AB \cdots \to \cdots BAT \cdots \to (\cdots BA \cdots)T \overset{IH}{\to} (B^i A^j T^{s-1})T = B^i A^j T^s.$$

Gevolg. Elk woord $w(A, B, C)$ van lengte N kan geschreven worden als een woord $C^x B^y A^z T^t$, met $x + y + z = N$ en $0 \leq t \leq yz$.
Opmerking. Dit is eigenlijk een stelling over de discrete Heisenberggroep $H_3(\mathbb{Z})$. We hadden ook A, B als canonieke volgorde kunnen nemen in plaats van B, A, dat verandert niks aan de stelling maar de matrix T verandert wel, namelijk de ab rechtsboven verandert in $-ab$.
Het totaal bedrag op de drie spaarrekeningen, vertrekkend van de begintoestand met 1 euro op rekening 3 is dus gegeven door $\epsilon^x(abt + by + 1)$. Vermits dit lineair is in t, wordt het gemaximaliseerd door t zo groot mogelijk te kiezen dus $t = yz$. [Dit komt overeen met eerst alle A's uitvoeren en daarna pas alle B's. We hadden ook de andere keuze kunnen nemen en elk woord schrijven in de vorm $w(A, B) = C^x A^y B^z T^t$ maar dan zou T er anders uitzien en zouden we uiteindelijk de uitdrukking $\epsilon^x(abyz - abt + by + 1)$ vinden, die maximaal wordt als t minimaal is, dus $t = 0$.]
Dit leidt uiteindelijk tot de uitdrukking:

$$\epsilon^N \frac{abyz + by + 1}{\epsilon^{y+z}}, \text{ met } y + z \leq N.$$

We zouden verleid kunnen zijn deze uitdrukking analytisch uit te werken, maar vermits we enkel in discrete oplossingen geinteresseerd zijn, en we niet meteen kunnen garanderen dat het discreet optimum ook redelijk dicht bij het continu optimum ligt, is het eenvoudiger om gewoon alle ruwweg N^2 mogelijkheden af te gaan:

```
a = 0.1
b = 0.2
epsilon = 1+0.05
n = 100
record = 1
xyz = {0,0,0}
For[y=1,y<=n,y++,
    For[z=1, z<=n-y,z++,
        currentValue = (a*b*y*z + b*y + 1)/(epsilon^(y+z))
            If[currentValue > record,
            record = currentValue;
            xyz = {n-y-z, y, z};
            Print[record*epsilon^n];
        ]
    ]
]
```

Numerieke waarde

```
276.63665781875352164972422324971757919422778543347776654503342
72624707
```

6 Poincarés boomgaard ($\sqrt{13}$ punten)

Er zijn veel verschillende manieren om dit probleem aan te pakken. Ik heb gekozen enkele lemmas uit de volgende paper toe te passen die het rekenwerk wat draagelijker maken voor de computer (maar het is zeker mogelijk zonder deze lemmas te werken):
http://www.uam.es/gruposinv/ntatuam/downloads/orchard.pdf
De code is geschreven in python en dus comptatibel met *Sage*.
We starten met een lijst van alle mogelijke elementen van $\mathrm{SL}_2(Z)$ op te stellen die i in het eerste kwadrant binnen een afstand r afbeelden. Deze matrices komen in de variabele `lijst` terecht.

```
max = 45# max iteratiewaarde voor a,b
R = 2500 # straal van de cirkel

for a in range(0,max):
    for b in range(0,max):
        for c in range(-max,max):
            for d in range(-max,max):
                if a*a+b*b+c*c+d*d <= R and
                a*d-b*c == 1 and a*c+b*d > 0 and a*a+b*b < c*c+d*d:
                    list.append(lijst,[[a,b],[c,d]])
```

Nu hebben we een lijst met matrices die i naar het eerste kwadrant sturen en allen in de boomgaard liggen maar bepaalde verschillende matrices sturen i naar hetzelfde punt. In deze oplossing gaan we het aantal matrices tellen i.p.v. het aantal punten want straks zullen we adhv de matrices bepalen welke punten zichtbaar zijn en welke niet.
We zeggen dat 2 matrices congruent zijn als ze i naar hetzelfde punt sturen en definieren een functie die een matrix en een lijst met matrices neemt en kijkt of deze lijst een matrix bevat die congruent is met de gegeven matrix. Deze functie zullen we dan gebruiken om de congruenten te filteren uit de lijst.

```
def congruent(mat1,lijst):
    for i in range(0,len(lijst)):
        if mat1[1][0]**2 + mat1[1][1]**2 == lijst[i][1][0]**2+
        lijst[i][1][1]**2 and mat1[0][0]*mat1[1][0] +
        mat1[0][1]*mat1[1][1] == lijst[i][0][0]*lijst[i][1][0]+
        lijst[i][0][1]*lijst[i][1][1]:
            return True
    return False

def removedups(lijstje):
    lijst2=[]
    for mat1 in lijstje:
        if not congruent(mat1,lijst2):
            lijst2.append(mat1)
    return lijst2
lijst = removedups(lijst)
```

Van deze lijst met matrices wensen we nu te bepalen welke een zichtbaar punt en welke een onzichtbaar punt bepaald. Hiervoor definieren we eerst een functie `divisors` die

alle delers behalve het getal zelf teruggeeft van een bepaald natuurlijk getal:

```
def divisors(n):
    list = []
    if n == 1:
        return [1]
    else:
        for x in range(1,int(n)):
            if (int(n) % x) == 0:
                list.append(x)
        return list
```

We zetten het aantal zichtbare bomen en onzichtbare bomen op 0 en overlopen de lijst met matrices. Voor iedere matrix bepalen we ofdat hij een zichtbare of onzichtbare boom creert en we updaten de variabelen. Het criterium voor een matrix die een zichtbare/onzichtbare boom creert kan je terug vinden onder Lemma 3.3 in de paper.

```
visPoints=0
nonVisPoints=0
for n inrange(0,len(lijst)):
    B = lijst[n][0][0]*lijst[n][1][0]+
    lijst[n][0][1]*lijst[n][1][1]
    D = lijst[n][1][0]**2 + lijst[n][1][1]**2
    delers = divisors(B)
    visible = True
    for m in range(0,len(delers)):
        b = delers[m]
        for a in range(1,b+1):
           if (b*b+1)%a ==0 and
           D*B*(b*b-a*a+1) == a*b*(D*D-B*B-1) and b!=B:
               visible=False
           if not visible:
               break
        if not visible:
           break
    if visible:
        visPoints=visPoints+1
    else:
        nonVisPoints=nonVisPoints+1
print(4*visPoints)
```

```
print(4*nonVisPoints)
```

Numerieke waarde

```
3600
```