Nim

June 5, 2021

1 Nim

1.1 Het oorspronkelijke spel

Spelers A en B spelen Nim tegen elkaar. Er liggen 20 lucifers op tafel. Om de beurt mogen A en B 1 of 2 lucifers wegnemen. Degene die de laatste lucifer(s) wegneemt, die verliest. Wat is de beste strategie voor de spelers om te winnen?

1.2 Variant met een NimSet

De spelregels worden enigszins aangepast. Vooraf spreken A en B af dat ze slechts c lucifers mogen wegnemen voor $c \in$ NimSet, waarbij $NimSet$ een eindige verzameling is. Degene die de laatste lucifer(s) wegneemt wint (of verliest). Bij het traditionele spel is NimSet = {1,2}. Maar nu gaan we NimSet varieren. We spreken af dat A altijd begint. A en B beginnen met N lucifers. Wie wint wanneer. Opmerking: Als $1 \in$ NimSet dan zal altijd één van beiden winnen, zo niet dan kan in principe ook remise voorkomen, en dat maakt het Nimspel ingewikkelder. Daarom nemen we aan dat $1 \in$ NimSet. A begint elk spelletje. A en B zijn aan elkaar gewaagd, ofwel ze zijn superslim.

We onderscheiden de twee gevallen: 1. Degene die de laatste lucifer(s) wegneemt verliest. (Geval 1) 2. Degene die de laatste lucifer(s) wegneemt wint. (Geval 2)

Hoe kunnen A en B een strategie bedenken om te winnen? We werken van klein naar groot. We kijken alleen naar Geval 1. We lossen het probleem achtereenvolgens op voor $N = 1, 2, 3, \ldots$. Voor $N = 1$, is duidelijk dat A verliest.

1.3 Het basisprogramma

We gaan uit van NimSet. In het specifieke voorbeeld kiezen we NimSet = { 1, 4, 9, 10 }. Als $N = 1$ dan verliest A, immers A moet 1 lucifer weghalen. Hij neemt dus de laatste lucifer weg. Als $N = 2$, dan wint A. Door 1 lucifer weg te halen ontstaat er een situatie waarbij B zal verliezen.

NimWinst	2	4	5	7	9	10	11	12	13	15	...
NimVerlies	1	3	6	8	14	19	21	26	32	34	...

Het wordt steeds ingewikkelder om na te gaan of je zult winnen of verliezen. Tussen 26 en 32 zul je als beginspeler altijd winnen als volgt:

- 27 betekent winst want 27 - 1 = 26 betekent verlies
- 28 betekent winst want 28 - 9 = 19 betekent verlies
- 29 betekent winst want 29 - 10 = 19 betekent verlies

- 30 betekent winst want 30 - 9 = 21 betekent verlies
- 31 betekent winst want 31 - 10 = 21 betekent verlies

Je kunt ook in een hele lange rij met nullen en enen bijhouden of A wint (0) of verliest (1). Zo'n lijst (L) ziet er als volgt uit:

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	…
L	1	0	1	0	0	1	0	1	0	0	0	0	0	1	0	0	…

Mogelijk helpt je een dergelijke voorstelling om het onderstaande beter te begrijpen.

In de praktijk blijkt alleen NimVerlies van belang om bij te houden. De waarden van N die niet behoren tot NimVerlies behoren automatisch tot NimWinst. Per waarde van N controleren we of één van de waarden $N - c$ is bevat in NimVerlies, waarbij $c \in$ NimSet. In dat geval is N bevat in NimWinst, zo niet dan is N bevat in NimVerlies. In het laatste geval voegen we N toe aan NimVerlies.

We gingen uit van de eindige verzameling NimSet. Laat M de grootste waarde van NimSet. Of een zekere N zal worden toegevoegd aan NimVerlies is uitsluitend afhankelijk van de laatste M waarden van NimVerlies. Die laatste M waarden kunnen ieder afzonderlijk al dan niet bevat zijn in NimVerlies. Er zijn hooguit 2^M verschillende mogelijkheden of een nieuwe waarde al dan niet in NimVerlies terecht komt. Dat betekent dat hooguit na een lengte van 2^M zal de rij waarden van NimVerlies in een herhaling terecht komen. Meestal zal de bovengrens van 2^M niet worden bereikt. We zijn uiteraard geïnteresseerd in de lengte van de periode, die we aanduiden met PeriodeLen. Daarnaast kan in principe in het begin nog een paar onregelmatigheden zitten. Daarna ontstaat een herhalend patroon. Hoe groot is dat eerste onregelmatige deel? De lengte noteren we met StartLen. Een voorbeeld waarbij StartLen groter dan 0 is, is NimSet = {1, 6, 9}.

Het is natuurlijk nog wel de vraag hoe je StartLen en PeriodeLen berekent.

1.4 Berekening van StartLen en PeriodeLen

Als in NimVerlies twee opeenvolgende en identieke intervallen te vinden zijn van een lengte groter dan M dan heb je de periode gevonden. Maar in principe kan de periode kleiner zijn dan M. Om te controleren dat twee intervallen identiek zijn moeten de elementen van NimVerlies in de twee intervallen op een afstand PeriodeLen van elkaar vandaan liggen, maar ook de elementen van NimWinst. We maken gebruik van een slim trucje. We gaan kijken naar opeenvolgende verschillen van de elementen van NimVerlies. We bekijken eerst even het voorbeeld van hierboven met NimSet = { 1, 4, 9, 10 }.

Verschil	2	3	2	6	5	2	5	6	2	3	2	6	5	2	5	6

Nu is het een kwestie van zoeken naar een rij verschillen met som minimaal M zo dat deze rij zich oneindig vaak herhaalt. In het voorbeeld hier boven is dat de verzameling PeriodeDD = {2, 3, 2, 6, 5, 2, 5, 6}. (We geven verschillen (in variabelennamen) consequent aan met DD om het programma enigszins leesbaar te houden.) De som van deze verschillen is 31, dus PeriodeLen = 31.

De procedure CheckPeriodeDD controleert of de verschillen die voorkomen in PeriodeDD minimaal tweemaal terugkeren in NimDD. Hierbij kan het voorkomen dat een aantal verschillen vooraan in

de rij worden overgeslagen, omdat hier nog sprake is van onregelmatigheden. Bijvoorbeeld in het geval $SetNim = \{1, 6, 9\}$:

Verschil	2	2	3	5	2	3	2	3	2	3	2	3	2	3	2	3

Het eerste deel is onregelmatig, en dat moet dus worden overgeslagen. We moeten uiteindelijk vinden StartDD = {2, 2, 3, 5} en PeriodeDD = {2, 3}. In de praktijk gaat het om een minderheid van alle gevallen. In het programma maken we tevens gebruik van StartDDLen = 4, het aantal elementen van StartDD en PeriodeDDLen = 2, het aantal elementen van PeriodeDD.

1.5 Alle verzamelingen NimSet doorlopen

Naar mate M groter wordt neemt ook het aantal verzamelingen NimSet toe. Bijvoorbeeld als $M = 10$, dan geldt $1, 10 \in$ NimSet. De tussen liggende getallen $2 \ldots 9$ kunnen al dan niet voorkomen. Daarvoor zijn $2^8 = 256$ mogelijkheden.

1.6 Records

Hieronder staat een overzicht van spelletjes met bijzondere verzamelingen NimSet met de grootste PeriodeLen voor M (de grootste waarde van NimSet.

NimSet	**PeriodLen**	**PeriodeVerschillen**
{1}	2	{2}
{1, 2}	3	{3}
{1, 2, 3}	4	{4}
{1, 3, 4}	7	{2, 5}
{1, 4, 5}	8	{2, 6}
{1, 5, 6}	11	{2, 2, 7}
{1, 4, 6, 7}	13	{2, 3, 5, 3}
{1, 4, 7, 8}	25	{2, 3, 6, 3, 2, 9}
{1, 4, 8, 9}	17	{2, 2, 3, 7, 3}
{1, 4, 9, 10}	33	{2, 2, 3, 8, 3, 2, 2, 11}
{1, 2, 6, 9, 10, 11}	39	{3, 4, 8, 4, 3, 5, 7, 5}
{1, 6, 11, 12}	61	{2, 2, 3, 2, 8, 5, 2, 2, 5, 8, 2, 3, 2, 2, 13}

Kortom je kunt nog vele spelletjes Nim spelen!

1.7 Opdrachten

1. Werk op papier uit bij NimSet = {1, 6, 9} dat StartDDLen groter dan 0 is.
2. Maak een programma voor Geval 2.
3. Wat gebeurt er als NimSet niet 1 bevat? Er kunnen nu ook remises optreden. Bijvoorbeeld als er precies 1 lucifer ligt. Als je dreigt te verliezen, dan is een remise een betere optie. Werk bijvoorbeeld op papier uit het geval dat NimSet = {2, 3}. Probeer het programma aan te passen.
4. In BerekenNimPeriode Staan twee while-loops in elkaar. De buitenste while-loop is er om StartDDLen te vergroten. Daarbinnen is een while-loop om PeriodeDDLen te vergroten. Dat

deel van het programma is heel omslachtig en tijdrovend. Bedenk een efficientere manier om StartDD en PeriodeDD te berekenen.

[42]:
```
# Nim
# Invoer: NimSet
# Uitvoer: PeriodeDD en StartDD

import math

def BerekenVolgendeVerlies(NimSet, NimVerlies):
    L = len(NimVerlies)
    LaatsteVerlies = NimVerlies[L-1]
    VolgendeVerlies = LaatsteVerlies
    OK = True
    while OK == True:
        VolgendeVerlies += 1
        OK = False
        for i in NimSet:
            if (VolgendeVerlies > i) & (VolgendeVerlies-i in NimVerlies):
                OK = True
    return VolgendeVerlies

def CheckPeriodeDD(PeriodeDD, NimDD, StartDDLen, M):
    PeriodeLen = 0
    for v in PeriodeDD:
        PeriodeLen += v
    num = math.ceil(M / PeriodeLen)
    if num < 2:
        num = 2
    NimDDLen = len(NimDD)
    PeriodeDDLen = len(PeriodeDD)
    #print(StartDDLen, PeriodeDDLen, NimDDLen, M, num)
    if NimDDLen < StartDDLen + num * PeriodeDDLen:
        return False
    ok = 0
    for n in range(num):
        for p in range(PeriodeDDLen):
            if PeriodeDD[p] == NimDD[StartDDLen + p + n*PeriodeDDLen]:
                ok += 1
    return ok == num * PeriodeDDLen

NimVerlies = [1]
NimDD = []
NimWinst = [] # Wordt niet gebruikt
NimRemise = [] # Wordt met deze settings niet gebruikt
# Voorbeelden van NimSet
NimSet = [1, 8, 15, 16]
NimSet = [1, 4, 9, 10]
```

```
NimSet = [1, 6, 9]
NimSet = [1, 4, 5, 10]
LenNimSet = len(NimSet)
M = NimSet[LenNimSet-1]
VolgendeVerlies = 1
LaatsteVerlies = 1
while VolgendeVerlies < 2*M:
    VolgendeVerlies = BerekenVolgendeVerlies(NimSet, NimVerlies)
    NimDD.append(VolgendeVerlies - LaatsteVerlies)
    NimVerlies.append(VolgendeVerlies)
    LaatsteVerlies = VolgendeVerlies
PeriodeGevonden = False
while not PeriodeGevonden:
    VolgendeVerlies = BerekenVolgendeVerlies(NimSet, NimVerlies)
    NimDD.append(VolgendeVerlies - LaatsteVerlies)
    NimVerlies.append(VolgendeVerlies)
    LaatsteVerlies = VolgendeVerlies
    NimDDLen = len(NimDD)
    StartDD = []
    StartDDLen = 0
    PeriodeDD = []
    PeriodeDDLen = 0
    while (not PeriodeGevonden) & (StartDDLen + 2 * PeriodeDDLen < NimDDLen) & \
          (NimDDLen - StartDDLen >= 4):
        while (not PeriodeGevonden) & (StartDDLen + 2 * PeriodeDDLen <␣
 ↪NimDDLen):
            PeriodeDD.append(NimDD[StartDDLen + PeriodeDDLen])
            PeriodeDDLen += 1
            PeriodeGevonden = CheckPeriodeDD(PeriodeDD, NimDD, StartDDLen, M)
        if not PeriodeGevonden:
            StartDD.append(NimDD[StartDDLen])
            StartDDLen += 1
            PeriodeDD = []
            PeriodeDDLen = 0
if StartDDLen > 0:
    print(NimSet, ":", PeriodeDD, "Start", StartDD)
else:
    print(NimSet, ":", PeriodeDD)
print("Ready")
```

```
[1, 4, 5, 10] : [3] Start [2, 6]
Ready
```

[44]:
```
# Nim
# Invoer: MM
# Uitvoer: PeriodeDD en StartDD voor alle (unieke) NimSet met M <= MM
```

```
import math
import numpy

def BerekenVolgendeVerlies(NimSet, NimVerlies):
    L = len(NimVerlies)
    LaatsteVerlies = NimVerlies[L-1]
    VolgendeVerlies = LaatsteVerlies
    OK = True
    while OK == True:
        VolgendeVerlies += 1
        OK = False
        for i in NimSet:
            if (VolgendeVerlies > i) & (VolgendeVerlies-i in NimVerlies):
                OK = True
    return VolgendeVerlies

def CheckPeriodeDD(PeriodeDD, NimDD, StartDDLen, M):
    PeriodeLen = 0
    for v in PeriodeDD:
        PeriodeLen += v
    num = math.ceil(M / PeriodeLen)
    if num < 2:
        num = 2
    NimDDLen = len(NimDD)
    PeriodeDDLen = len(PeriodeDD)
    #print(StartDDLen, PeriodeDDLen, NimDDLen, M, num)
    if NimDDLen < StartDDLen + num * PeriodeDDLen:
        return False
    ok = 0
    for n in range(num):
        for p in range(PeriodeDDLen):
            if PeriodeDD[p] == NimDD[StartDDLen + p + n*PeriodeDDLen]:
                ok += 1
    return ok == num * PeriodeDDLen

def BerekenNimPeriode(NimSet):
    NimVerlies = [1]
    NimDD = []
    LenNimSet = len(NimSet)
    M = NimSet[LenNimSet-1]
    VolgendeVerlies = 1
    LaatsteVerlies = 1
    while VolgendeVerlies < 2*M:
        VolgendeVerlies = BerekenVolgendeVerlies(NimSet, NimVerlies)
        NimDD.append(VolgendeVerlies - LaatsteVerlies)
        NimVerlies.append(VolgendeVerlies)
        LaatsteVerlies = VolgendeVerlies
```

```
    PeriodeGevonden = False
    while not PeriodeGevonden:
        VolgendeVerlies = BerekenVolgendeVerlies(NimSet, NimVerlies)
        NimDD.append(VolgendeVerlies - LaatsteVerlies)
        NimVerlies.append(VolgendeVerlies)
        LaatsteVerlies = VolgendeVerlies
        NimDDLen = len(NimDD)
        StartDD = []
        StartDDLen = 0
        PeriodeDD = []
        PeriodeDDLen = 0
        while (not PeriodeGevonden) & (StartDDLen + 2 * PeriodeDDLen <␣
↪NimDDLen) & \
              (NimDDLen - StartDDLen >= 4):
            while (not PeriodeGevonden) & (StartDDLen + 2 * PeriodeDDLen <␣
↪NimDDLen):
                PeriodeDD.append(NimDD[StartDDLen + PeriodeDDLen])
                PeriodeDDLen += 1
                PeriodeGevonden = CheckPeriodeDD(PeriodeDD, NimDD, StartDDLen,␣
↪M)
            if not PeriodeGevonden:
                StartDD.append(NimDD[StartDDLen])
                StartDDLen += 1
                PeriodeDD = []
                PeriodeDDLen = 0
    if StartDDLen > 0:
        return [PeriodeDD, StartDD]
    else:
        return [PeriodeDD]

MM = 10
PS_Set = []
kM = numpy.power(2,MM)
cnt = 0
for k in range(kM):
    NimSet = [1]
    c = 2
    while k > 0:
        if k % 2 == 1:
            NimSet.append(c)
        k = k // 2
        c += 1
    PS = BerekenNimPeriode(NimSet)
    PeriodeLen = 0
    for p in PS[0]:
        PeriodeLen += p
    if not PS in PS_Set:
```

```
        cnt += 1
        if len(PS) == 2:
            print(cnt, NimSet, ": (", PeriodeLen, ")", PS[0], "Start:", PS[1])
        else:
            print(cnt, NimSet, ": (", PeriodeLen, ")", PS[0])
        PS_Set.append(PS)
print("Ready")
```

```
1 [1] : ( 2 ) [2]
2 [1, 2] : ( 3 ) [3]
3 [1, 2, 3] : ( 4 ) [4]
4 [1, 4] : ( 5 ) [2, 3]
5 [1, 3, 4] : ( 7 ) [2, 5]
6 [1, 2, 3, 4] : ( 5 ) [5]
7 [1, 4, 5] : ( 8 ) [2, 6]
8 [1, 2, 3, 4, 5] : ( 6 ) [6]
9 [1, 6] : ( 7 ) [2, 2, 3]
10 [1, 2, 6] : ( 7 ) [3, 4]
11 [1, 3, 6] : ( 9 ) [2, 2, 5]
12 [1, 2, 4, 6] : ( 8 ) [3, 5]
13 [1, 5, 6] : ( 11 ) [2, 2, 7]
14 [1, 4, 5, 6] : ( 9 ) [2, 7]
15 [1, 2, 4, 5, 6] : ( 10 ) [3, 7]
16 [1, 2, 3, 4, 5, 6] : ( 7 ) [7]
17 [1, 4, 7] : ( 8 ) [2, 3, 3]
18 [1, 6, 7] : ( 12 ) [2, 2, 8]
19 [1, 4, 6, 7] : ( 13 ) [2, 3, 5, 3]
20 [1, 3, 4, 6, 7] : ( 10 ) [2, 8]
21 [1, 2, 5, 6, 7] : ( 11 ) [3, 8]
22 [1, 2, 3, 4, 5, 6, 7] : ( 8 ) [8]
23 [1, 8] : ( 9 ) [2, 2, 2, 3]
24 [1, 3, 8] : ( 11 ) [2, 2, 2, 5]
25 [1, 2, 3, 8] : ( 9 ) [4, 5]
26 [1, 4, 8] : ( 12 ) [2, 3, 2, 5]
27 [1, 5, 8] : ( 13 ) [2, 2, 2, 7]
28 [1, 2, 3, 5, 8] : ( 10 ) [4, 6]
29 [1, 2, 3, 5, 6, 8] : ( 11 ) [4, 7]
30 [1, 7, 8] : ( 15 ) [2, 2, 2, 9]
31 [1, 4, 7, 8] : ( 25 ) [2, 3, 6, 3, 2, 9]
32 [1, 3, 4, 7, 8] : ( 11 ) [2, 9]
33 [1, 6, 7, 8] : ( 13 ) [2, 2, 9]
34 [1, 2, 6, 7, 8] : ( 12 ) [3, 9]
35 [1, 4, 6, 7, 8] : ( 14 ) [2, 3, 9]
36 [1, 2, 3, 5, 6, 7, 8] : ( 13 ) [4, 9]
37 [1, 2, 3, 4, 5, 6, 7, 8] : ( 9 ) [9]
38 [1, 2, 9] : ( 10 ) [3, 3, 4]
39 [1, 2, 4, 9] : ( 11 ) [3, 3, 5]
```

40 [1, 3, 4, 9] : (12) [2, 5, 5]
41 [1, 2, 4, 5, 9] : (13) [3, 3, 7]
42 [1, 6, 9] : (5) [2, 3] Start: [2, 2, 3, 5]
43 [1, 2, 5, 7, 9] : (14) [3, 3, 8]
44 [1, 8, 9] : (16) [2, 2, 2, 10]
45 [1, 4, 8, 9] : (5) [5] Start: [2, 3, 2]
46 [1, 4, 5, 8, 9] : (12) [2, 10]
47 [1, 6, 8, 9] : (17) [2, 2, 3, 7, 3]
48 [1, 3, 6, 8, 9] : (14) [2, 2, 10]
49 [1, 2, 7, 8, 9] : (16) [3, 3, 10]
50 [1, 4, 7, 8, 9] : (15) [2, 3, 10]
51 [1, 2, 6, 7, 8, 9] : (13) [3, 10]
52 [1, 2, 3, 6, 7, 8, 9] : (14) [4, 10]
53 [1, 2, 3, 4, 5, 6, 7, 8, 9] : (10) [10]
54 [1, 10] : (11) [2, 2, 2, 2, 3]
55 [1, 3, 10] : (13) [2, 2, 2, 2, 5]
56 [1, 4, 10] : (11) [3, 2, 3, 3] Start: [2, 3, 2, 6]
57 [1, 2, 3, 4, 10] : (11) [5, 6]
58 [1, 5, 10] : (15) [2, 2, 2, 2, 7]
59 [1, 4, 5, 10] : (3) [3] Start: [2, 6]
60 [1, 3, 4, 5, 10] : (14) [2, 6, 6]
61 [1, 6, 10] : (16) [2, 2, 3, 2, 2, 5]
62 [1, 2, 6, 10] : (11) [3, 4, 4]
63 [1, 4, 6, 10] : (14) [2, 3, 2, 7]
64 [1, 2, 3, 4, 6, 10] : (12) [5, 7]
65 [1, 7, 10] : (17) [2, 2, 2, 2, 9]
66 [1, 4, 7, 10] : (11) [2, 3, 3, 3]
67 [1, 3, 4, 7, 10] : (19) [2, 6, 5, 6]
68 [1, 4, 6, 7, 10] : (16) [2, 3, 8, 3]
69 [1, 2, 3, 4, 6, 7, 10] : (13) [5, 8]
70 [1, 2, 6, 8, 10] : (16) [3, 4, 5, 4]
71 [1, 2, 5, 6, 8, 10] : (18) [3, 4, 7, 4]
72 [1, 2, 3, 4, 6, 7, 8, 10] : (14) [5, 9]
73 [1, 9, 10] : (19) [2, 2, 2, 2, 11]
74 [1, 4, 9, 10] : (31) [2, 3, 2, 6, 5, 2, 5, 6]
75 [1, 3, 4, 9, 10] : (13) [2, 5, 6]
76 [1, 6, 9, 10] : (33) [2, 2, 3, 8, 3, 2, 2, 11]
77 [1, 3, 6, 9, 10] : (15) [2, 2, 11]
78 [1, 4, 6, 9, 10] : (18) [2, 3, 2, 11]
79 [1, 3, 4, 6, 9, 10] : (19) [2, 5, 7, 5]
80 [1, 4, 5, 6, 9, 10] : (13) [2, 11]
81 [1, 4, 7, 9, 10] : (19) [2, 3, 3, 5, 3, 3]
82 [1, 8, 9, 10] : (17) [2, 2, 2, 11]
83 [1, 2, 8, 9, 10] : (17) [3, 3, 11]
84 [1, 3, 4, 8, 9, 10] : (18) [2, 5, 6, 5]
85 [1, 6, 8, 9, 10] : (18) [2, 2, 3, 11]
86 [1, 2, 4, 6, 8, 9, 10] : (14) [3, 11]
87 [1, 2, 3, 7, 8, 9, 10] : (15) [4, 11]

88 [1, 4, 7, 8, 9, 10] : (16) [2, 3, 11]
89 [1, 2, 3, 4, 6, 7, 8, 9, 10] : (16) [5, 11]
90 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] : (11) [11]
91 [1, 6, 11] : (12) [2, 2, 3, 2, 3]
92 [1, 2, 6, 11] : (12) [4, 3, 5] Start: [3]
93 [1, 3, 6, 11] : (14) [2, 2, 5, 5]
94 [1, 2, 4, 6, 11] : (13) [3, 5, 5]
95 [1, 2, 5, 6, 11] : (17) [3, 4, 3, 7]
96 [1, 4, 5, 6, 11] : (19) [2, 7, 3, 7]
97 [1, 3, 4, 5, 6, 11] : (16) [2, 7, 7]
98 [1, 4, 7, 11] : (18) [2, 3, 3, 2, 8]
99 [1, 2, 6, 7, 11] : (4) [4] Start: [3, 5]
100 [1, 4, 6, 7, 11] : (18) [2, 3, 5, 5, 3]
101 [1, 8, 11] : (7) [2, 3, 2] Start: [2, 2, 2, 3, 7]
102 [1, 4, 5, 8, 11] : (3) [3] Start: [2, 7]
103 [1, 3, 4, 5, 8, 11] : (6) [6] Start: [2, 7]
104 [1, 6, 8, 11] : (7) [2, 3, 2] Start: [2, 2, 3, 2, 5]
105 [1, 4, 7, 8, 11] : (29) [2, 3, 9, 3, 2, 10]
106 [1, 2, 6, 9, 11] : (20) [3, 4, 3, 5, 5]
107 [1, 4, 8, 9, 11] : (17) [2, 3, 2, 5, 5]
108 [1, 10, 11] : (20) [2, 2, 2, 2, 12]
109 [1, 4, 10, 11] : (7) [5, 2] Start: [2, 3, 2, 7]
110 [1, 6, 10, 11] : (21) [2, 2, 3, 2, 7, 5]
111 [1, 2, 6, 10, 11] : (12) [3, 4, 5]
112 [1, 3, 6, 10, 11] : (7) [2, 5] Start: [2, 2, 5, 7]
113 [1, 2, 4, 6, 10, 11] : (20) [3, 5, 7, 5]
114 [1, 5, 6, 10, 11] : (16) [2, 2, 12]
115 [1, 2, 5, 6, 10, 11] : (31) [3, 4, 8, 4, 3, 9]
116 [1, 4, 7, 10, 11] : (34) [2, 3, 3, 6, 3, 3, 2, 12]
117 [1, 4, 6, 7, 10, 11] : (17) [2, 3, 9, 3]
118 [1, 2, 4, 6, 7, 10, 11] : (21) [3, 5, 8, 5]
119 [1, 3, 4, 6, 7, 10, 11] : (14) [2, 12]
120 [1, 8, 10, 11] : (21) [2, 2, 2, 3, 9, 3]
121 [1, 3, 8, 10, 11] : (18) [2, 2, 2, 12]
122 [1, 4, 5, 8, 10, 11] : (3) [3] Start: [2, 7, 6]
123 [1, 2, 6, 8, 10, 11] : (28) [3, 4, 5, 4, 3, 9]
124 [1, 2, 5, 6, 8, 10, 11] : (16) [3, 4, 9]
125 [1, 4, 7, 8, 10, 11] : (31) [2, 3, 9, 3, 2, 12]
126 [1, 2, 9, 10, 11] : (18) [3, 3, 12]
127 [1, 4, 9, 10, 11] : (19) [2, 3, 2, 12]
128 [1, 6, 9, 10, 11] : (19) [2, 2, 3, 12]
129 [1, 2, 6, 9, 10, 11] : (39) [3, 4, 8, 4, 3, 5, 7, 5]
130 [1, 2, 5, 6, 9, 10, 11] : (34) [3, 4, 8, 4, 3, 12]
131 [1, 2, 4, 5, 6, 9, 10, 11] : (15) [3, 12]
132 [1, 4, 7, 9, 10, 11] : (20) [2, 3, 3, 12]
133 [1, 2, 6, 7, 9, 10, 11] : (4) [4] Start: [3, 5, 8]
134 [1, 4, 6, 7, 9, 10, 11] : (17) [2, 3, 12]
135 [1, 2, 3, 8, 9, 10, 11] : (16) [4, 12]

```
136 [1, 2, 6, 8, 9, 10, 11] : ( 19 ) [3, 4, 12]
137 [1, 2, 3, 4, 7, 8, 9, 10, 11] : ( 17 ) [5, 12]
138 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] : ( 12 ) [12]
Ready
```

[47]:
```python
# Nim
# De gebruikte procedures staan hier boven
# Uitvoer: uitsluitend de NimSet met langste periode per M

MM = 19
PS_Set = []
kM = numpy.power(2,MM)
cnt = 0
M0 = 0
PeriodeLenMax = 0
for k in range(kM):
    NimSet = [1]
    c = 2
    while k > 0:
        if k % 2 == 1:
            NimSet.append(c)
        k = k // 2
        c += 1
    NimSetLen = len(NimSet)
    M = NimSet[NimSetLen - 1]
    if M > M0:
        if PeriodeLenMax > 0:
            print(M0, NimSetMax, PeriodeLenMax, PSMax)
        M0 = M
        PeriodeLenMax = 0
    PS = BerekenNimPeriode(NimSet)
    PeriodeLen = 0
    for p in PS[0]:
        PeriodeLen += p
    if PeriodeLen > PeriodeLenMax:
        PeriodeLenMax = PeriodeLen
        NimSetMax = NimSet
        PSMax = PS[0]
print(M, NimSetMax, PeriodeLenMax, PSMax)
print("Ready")
```

```
2 [1] 2 [2]
3 [1, 2] 3 [3]
4 [1, 2, 3] 4 [4]
5 [1, 3, 4] 7 [2, 5]
6 [1, 4, 5] 8 [2, 6]
7 [1, 5, 6] 11 [2, 2, 7]
8 [1, 4, 6, 7] 13 [2, 3, 5, 3]
```

```
9 [1, 4, 7, 8] 25 [2, 3, 6, 3, 2, 9]
10 [1, 6, 8, 9] 17 [2, 2, 3, 7, 3]
11 [1, 6, 9, 10] 33 [2, 2, 3, 8, 3, 2, 2, 11]
12 [1, 2, 6, 9, 10, 11] 39 [3, 4, 8, 4, 3, 5, 7, 5]
13 [1, 6, 11, 12] 61 [2, 2, 3, 2, 8, 5, 2, 2, 5, 8, 2, 3, 2, 2, 13]
14 [1, 4, 6, 12, 13] 64 [2, 3, 2, 3, 11, 3, 2, 3, 2, 9, 5, 2, 3, 5, 9]
15 [1, 8, 13, 14] 73 [2, 2, 2, 3, 2, 10, 5, 2, 2, 2, 5, 10, 2, 3, 2, 2, 2, 15]
16 [1, 2, 6, 10, 13, 14, 15] 83 [3, 4, 4, 8, 4, 4, 3, 5, 4, 7, 5, 4, 3, 4, 5, 7,
4, 5]
17 [1, 8, 15, 16] 113 [2, 2, 2, 3, 2, 2, 10, 7, 2, 2, 2, 5, 2, 10, 2, 5, 2, 2,
2, 7, 10, 2, 2, 3, 2, 2, 2, 17]
18 [1, 4, 13, 17] 118 [2, 3, 2, 3, 2, 9, 5, 2, 3, 2, 3, 6, 5, 5, 2, 3, 5, 6, 5,
3, 2, 5, 5, 6, 3, 2, 3, 2, 5, 9]
19 [1, 4, 6, 17, 18] 144 [2, 3, 2, 3, 2, 3, 11, 5, 3, 2, 3, 2, 5, 9, 5, 2, 3, 2,
3, 5, 11, 3, 2, 3, 2, 3, 2, 9, 5, 5, 2, 3, 5, 5, 9]
20 [1, 4, 8, 11, 15, 18, 19] 237 [2, 3, 2, 5, 2, 14, 6, 3, 3, 10, 7, 3, 2, 5, 7,
9, 5, 2, 5, 2, 3, 17, 3, 2, 5, 2, 5, 9, 7, 5, 2, 3, 7, 10, 3, 3, 6, 14, 2, 5, 2,
3, 2, 20]
20 [1, 3, 4, 9, 16, 17, 19, 20] 253 [2, 5, 5, 2, 11, 10, 2, 6, 5, 2, 8, 13, 2,
6, 2, 5, 8, 10, 5, 6, 2, 5, 5, 13, 5, 5, 2, 6, 5, 10, 8, 5, 2, 6, 2, 13, 8, 2,
5, 6, 2, 10, 11]
Ready
```

[48]:
```python
# Nim
# De gebruikte procedures staan hier boven
# Uitvoer: uitsluitend NimSets met start groter dan 0

MM = 10
PS_Set = []
kM = numpy.power(2,MM)
cnt = 0
M0 = 0
PeriodeLenMax = 0
for k in range(kM):
    NimSet = [1]
    c = 2
    while k > 0:
        if k % 2 == 1:
            NimSet.append(c)
        k = k // 2
        c += 1
    NimSetLen = len(NimSet)
    PS = BerekenNimPeriode(NimSet)
    if len(PS) > 1:
        PeriodeLen = 0
        for p in PS[0]:
            PeriodeLen += p
```

```
        StartLen = 0
        for p in PS[1]:
            StartLen += p
        print(NimSet, PeriodeLen, StartLen, PS[0], PS[1])
print("Ready")
```

```
[1, 6, 9] 5 12 [2, 3] [2, 2, 3, 5]
[1, 4, 8, 9] 5 7 [5] [2, 3, 2]
[1, 4, 6, 8, 9] 5 7 [5] [2, 3, 2]
[1, 4, 10] 11 13 [3, 2, 3, 3] [2, 3, 2, 6]
[1, 4, 5, 10] 3 8 [3] [2, 6]
[1, 4, 5, 7, 10] 3 8 [3] [2, 6]
[1, 2, 6, 11] 12 3 [4, 3, 5] [3]
[1, 2, 6, 7, 11] 4 8 [4] [3, 5]
[1, 8, 11] 7 16 [2, 3, 2] [2, 2, 2, 3, 7]
[1, 4, 5, 8, 11] 3 9 [3] [2, 7]
[1, 3, 4, 5, 8, 11] 6 9 [6] [2, 7]
[1, 6, 8, 11] 7 14 [2, 3, 2] [2, 2, 3, 2, 5]
[1, 6, 9, 11] 5 12 [2, 3] [2, 2, 3, 5]
[1, 4, 10, 11] 7 14 [5, 2] [2, 3, 2, 7]
[1, 3, 6, 10, 11] 7 16 [2, 5] [2, 2, 5, 7]
[1, 4, 6, 10, 11] 7 14 [5, 2] [2, 3, 2, 7]
[1, 2, 6, 7, 10, 11] 4 8 [4] [3, 5]
[1, 4, 8, 10, 11] 7 14 [5, 2] [2, 3, 2, 7]
[1, 4, 5, 8, 10, 11] 3 15 [3] [2, 7, 6]
[1, 3, 4, 5, 8, 10, 11] 6 9 [6] [2, 7]
[1, 3, 6, 8, 10, 11] 7 16 [2, 5] [2, 2, 5, 7]
[1, 4, 6, 8, 10, 11] 7 14 [5, 2] [2, 3, 2, 7]
[1, 2, 6, 7, 9, 10, 11] 4 16 [4] [3, 5, 8]
Ready
```

[ ]: