Somketens

Pythagoras jaargang 61 nummer 6
juni 2022

door **Matthijs Coster**

1 SomKetens

Het artikel over Somketens verscheen in Pythagoras juni 2022. Het idee is dat je een eindige verzameling S kiest, bijvoorbeeld kwadraten, en dat je de getallen $\{1, 2, 3, \ldots, N\}$ kiest, voor zekere N. Vervolgens maak je een rangschikking van deze getallen, zo dat de som van twee opeenvolgende getallen steeds een element van S is. De notatie is met "$-$" tussen de getallen. Bijvoorbeeld voor $N = 15$ en de kwadraten vinden we de oplossing $8 - 1 - 15 - 10 - 6 - 3 - 13 - 12 - 4 - 5 - 11 - 14 - 2 - 7 - 9$. Deze oplossing is (op volgorde na) uniek.

1.1 Opmerking vooraf

Bij het plaatsen van de zgn. puntjes op de i kwam ik tot de conclusie dat dit best wel een complex programma is geworden. De vraag is dan ook voordehandliggend of dit programma geschikt is om gepubliceerd te worden in Pythagoras. Mocht je commentaar hebben, stuur een email naar matthijs@pyth.eu. Zo lang ik niets verneem zal ik wat eenvoudigere programma's plaatsen. Gaarne inclusief opleiding die je volgt.

1.2 Artikel in Pythagoras

Een van de opgaven van de eerste ronde van de Wiskunde Olympiade luidde als volgt:

Zet de getallen 1 t/m 15 op een rij, zo dat twee getallen naast elkaar steeds optellen tot een kwadraat. Wat is de uitkomst als we het eerste en laatste getal bij elkaar optellen?

In dit artikel beschrijven we een programma geschreven in Python waarmee je dit probleem kunt oplossen. Maar het kan nog veel meer. In plaats van kwadraten kun je ook kiezen voor derde machten of driehoeksgetallen (dat zijn de sommen $1 + 2 + 3 + \cdots + n$. We gaan uit van een willekeurige (eindige) verzameling S. Je wilt de getallen $N = \{1, 2, 3, \ldots, n\}$, voor zekere N rangschikken zo dat elk van de sommen van twee opeenvolgende getallen een element is van S. De notatie werkt met "$-$" tussen de getallen. Bijvoorbeeld voor $n = 6$ en $S = \{7, 8\}$ vinden we de oplossing $1 - 6 - 2 - 5 - 3 - 4$. Deze oplossing is (op volgorde na) uniek.

Vooraf

Om te beginnen bepalen we de verzameling S. Dit gebeurt in Programma 1. Uiteraard kun je lukraak een aantal getallen kiezen, die samen S vormen, maar je kunt er ook voor kiezen om de derde machten in S te plaatsen. Ook is mogelijk om de Fibonacci getallen in S te plaatsen, of een combinatie van verschillende soorten getallen.

Verschillende getallen

De uitkomst van het programma dat we schrijven is een reeks $n_1 - n_2 - \cdots - n_k$, waarbij $\{n_1, n_2, \ldots, n_k\}$ een bepaalde permutatie is van de getallen $\{1, 2, 3, \ldots, k\}$. We merken op dat de getallen n_1 en n_k 'e'en buur hebben en de overige getallen twee buren.

De volgende opsplitsing van de getallen $\{1, 2, 3, \ldots, k\}$ ligt dus erg voor de hand:

- Geisoleerde getallen. Verzameling (C_0, variabele-notatie C0, aantal is c_0, variabele-notatie c0). Een getal k is bevat in C_0 dan is er geen enkel getal $m \in N$ te vinden met $k + m \in S$
- Getallen met slechts één buur. Verzameling (C_1, variabele-notatie C1 aantal is c_1, variabele-notatie c1). k is bevat in de verzameling C_1 als er precies één getal $m \in N$ bestaat met $n + m \in S$.
- De overige getallen hebben twee of meer buren. (Deze getallen zijn niet bevat in een verzameling met specifieke naam)

We gaan eerst uitzoeken voor elk getal of het betreffende getal bevat is in C_0 of C_1. Het is duidelijk zelfs als $c_0 = 1$, dan kun je geen keten maken. Ook moet c_1 beperkt zijn (tot hooguit 2).

Dit alles controleren we in Programma 2. Dan volgt Programma 3. Hier gaan we aan de slag met getallen die precies twee buren hebben. We beginnen met de getallen $\{1, 2, 3, \ldots, N\}$, die allen een eigen keten van lengte 1 vormen. Alle getallen worden geplaatst in Sin. Successievelijk worden ketens aan elkaar gekoppeld. Er ontstaan middengetallen (Mid) en randgetallen (Ran). De administratie is best wel lastig als twee ketens worden gekoppeld. Uiteindelijk blijft er één keten over met 2 randgetallen en verder middens.

De precieze uitwerking staat beschreven in het programma zelf.

Voorbeeld

We nemen een nog eenvoudiger voorbeeld dan de kwadraten. We kiezen de driehoeksgetallen (de sommen $1 + 2 + 3 + \cdots + n$). $N = \{1, 2, 3, \ldots, 10\}$. Het eerste dat we gaan doen is voor de getallen 1 t/m 10 nagaan of er een single tussen zit, en welke getallen op de rand terecht gaan komen.

$S = \{1, 3, 6, 10, 15\}$. Het volgende element zou 21 zijn, maar $9 + 10 = 19$ is de grootste som die gemaakt kan worden.

Vervolgens gaan we alle buren bepalen:

n	buren van n	stukje keten
1	2, 5, 9	
2	1, 4, 8	
3	7	$3 - 7$
4	2, 6	$2 - 4 - 6$
5	1, 10	$1 - 5 - 10$
6	4, 9	$4 - 6 - 9$
7	3, 8	$3 - 7 - 8$
8	2, 7	$2 - 8 - 7$
9	1, 6	$1 - 9 - 6$
10	5	$10 - 5$

We zien dat 3 en 10 aan de rand komen. Vervolgens gaan we kijken naar getallen met precies twee buren. Mocht er een keten zijn, dan vormen dat getal met die twee buren een deel van de keten. deze stukjes keten bevatten twee eindpunten. Bovendien kunnen getallen die precies twee buren hebben eventueel ook op de rand voorkomen (maar dat geldt slechts voor maximaal twee

getallen). Maar in dit voorbeeld is dat niet van toepassing, omdat 3 en 10 al de rand vormen. Nu kunnen we de stukjes keten aan elkaar knopen. Bijvoorbeeld geldt dat voor 4, 5 en 6.

We bekijken in een nieuwe tabel stukjes keten en de buren van de betreffende stukjes keten (In het artikel is de layout beter):

Stukken keten
$1-5-10$
$2-4-6-9-1$
$3-7-8-2$

Nu is het een kwestie om de drie deelketens met elkaar te verbinden. We vinden: $3-7-8-2-4-6-9-1-5-10$.

Op de website staat het programma. Daar wordt in de uitleg ook gekeken naar ingewikkeldere situaties. Maar het basisidee blijft hetzelfde: Je bouwt een totale keten op door steeds meer deelketens met elkaar te verbinden. Het programma biedt ook de mogelijkheid om te tellen hoeveel verschillende ketens er zijn, of om na te gaan dat er ook cycles bestaan (hierbij zijn er geen begin- en eindpunten).

Opgaven

- Voer de opdracht uit van de Wiskunde Olympiade
- Laat S de verzameling van getallen van de vorm n^2+1. Bepaal het kleinste aantal getallen waarvoor er een keten bestaat.
- Toon aan dat als S uit even getallen bestaat er geen keten bestaat. In principe kan het voorkomen dat alle getallen twee of meer buren hebben.
- Gebruik de programmatuur om een keten te vinden voor de volgende gevallen:
- Fibonacci getallen, (Het is niet zo moeilijk om meerdere ketens te vinden.)
- Derde machten, (dit is niet meer met de hand te doen!)

1.3 Programma

1.3.1 Belangrijk vooraf

Bij het lezen van de tekst zou er mogelijk verwarring kunnen ontstaan over het gebruik van C_0, C_1 enerzijds en Sin en Ran anderzijds. Ik zal het verschil hier nogmaals beschrijven. C_0 is de verzameling getallen die geen buren heeft. C_1 is de verzameling getallen met precies één buur. Daarnaast wordt bij elke taak (NuDoen) een keten opgebouwd. Aanvankelijk zijn er N verschillende mini-ketentjes, bestaande uit één element. De getallen 1 t/m N zijn aan het begin bevat in Sin. Maar daarna zullen ketens worden samengevoegd. Twee losse elementen (getallen in Sin) vormen een keten van lengte 2. Elke keten van lengte 2 of groter heeft twee eindpunten (randen). Deze getallen zijn bevat in Ran. Als de keten langer dan 2 is, dan bevat de keten ook middens (getallen zijn bevat in Mid). Het uiteindelijke doel is om één keten te verkrijgen met precies twee getallen in Ran en alle andere getallen in Mid. Omgekeerd als voor een verzameling geldt dat er twee eindpunten en voor de rest middens zijn, dan hoef je niet één keten te hebben! Namelijk er kunnen ook één of meer cycli ontstaan (een rij getallen waarbij het eerste en laatste getal hetzelfde zijn).

Ik heb er voor gekozen om bij het maken van ketens uit deelketens tegelijkertijd de verbindingen te verwijderen. uit de verzameling van alle verbindingen (C). Dat klinkt heel ingewikkeld, maar in de praktijk betekent dit dat C alleen bestaat uit verbindingen die de ketens kunnen laten groeien.

Als een getal in een keten twee buren heeft, dan komt dit getal helemaal niet meer voor in C.

Elke stap die wordt gezet betekent dat het aantal ketens wordt verkleind. Dus na een eindig aantal stappen houd je nog maar één keten over, en dat is dan precies de oplossing!

1.3.2 Notaties

De keten in wording bestaat uit drie soorten elementen: singles (Sin), randen (Ran) en middels (Mid).

- Ketens: Bevat de verschillende ketens met de getallen 1 t/m N.
- Sin: De getallen 1 t/m N worden in deze verzameling geplaatst.
- Ran: Om diverse redenen die verderop worden besproken plakken we getallen (of rijen getallen) aan elkaar. De randen van een rij getallen is bevat in Ran.
- Mid: De getallen in een keten die niet tot de randen behoren behoren tot Mid. Er is echter één uitzondering die later wordt besproken.

Er wordt een uitgebreide administratie bijgehouden van deze drie basisverzamelingen.

Op basis van S en N wordt er een tabel C geconstrueerd. In C staan per getal de connecties. Dit aantal connecties kan 0, 1, 2 of groter zijn. Juist de kleine waarden (0 en 1) van het aantal connecties is van belang om te bepalen of er uberhaupt een oplossing bestaat. Als er ook maar eenmaal een aantal connecties 0 (C_0) voorkomt dan wil dat zeggen dat er een getal k is dat met geen enkel ander getal een som s vormt dat een element is van S. Het zoeken naar een oplossing kan worden gestaakt!

Als het aantal connecties 1 (C_1) is dan is er maar één buur. Dat betekent dat het getal aan het begin of einde moet worden geplaatst. Maar daar zijn slechts twee plaatsen. Dus als het aantal malen dat 1 connectie voorkomt meer dan 2 is, dan is er geen oplossing.

Om na te gaan of er uberhaupt een oplossing kan bestaan tellen we het aantal keren dat 0 en 1 connectie voorkomt m.b.v. de formule isolatiewaarde$ = 3*c_0 + c_1.$Als <tt>isolatiewaarde</tt>$ > 2$, dan is er geen oplossing.

Als het aantal connecties 2 is (of meer) dan zijn er twee mogelijkheden. Het betreffende getal wordt omringd door twee getallen of het getal staat aan de rand, maar dan moet daar wel ruimte zijn.

Als het aantal connecties groter wordt dan twee dan zijn er meerdere mogelijkheden om twee connecties te kiezen. In het programma gooien we connecties weg. Hierdoor worden bepaalde connecties geforceerd. Door maar voldoende connecties weg te gooien forceer je dat er precies één oplossing overblijft.

We maken gebruik van een lijst TeDoen. Hierin staan taken die achtereenvolgens worden uitgevoerd als NuDoen. Zo'n taak bevat een lijst connecties die wordt weggegooid. Er zijn steeds drie mogelijkheden: - (a) er is geen oplossing - (b) er is precies één oplossing - (c) er zijn (wellicht) meerdere oplossingen

In het laatste geval wordt de lijst TeDoen uitgebreid met extra taken, door bepaalde connecties weg te gooien.

1.4 Overzicht van de programmatuur

- In Programma 1 wordt de S bepaald
- In Programma 2 wordt een ondergrens bepaald voor N.
- In Programma 3 wordt de daadwerkelijke berekening van het bepalen van de keten uitgevoerd.

1.4.1 Gebruik van verbose

Dit is slechts een schets. Mogelijk is de praktijk toch nog anders.

Met verbose kun je de hoeveelheid uitvoer varieren, van zeer beperkt (0) tot zeer gedetailleerd (8). We maken het volgende onderscheid:

- 0: silent, eerste oplossing wordt weergegeven, en uiteindelijk aantal oplossingen
- 1: alleen oplossingen worden geteld en todo per 100 weergegeven
- 2: alle oplossingen worden afgedrukt
- 3: tevens informatie over nieuw toegevoegd
- 4: tevens informatie over isolated
- 5: tevens Ran2
- 6: Geheel Ran en Sin
- 7: tevens Chains
- 8: Tevens stappen

1.5 Diverse opmerkingen

In sommige gevallen zijn er geen oplossingen. Enkele voorbeelden: - S bestaat uit louter even getallen - S bestaat uit louter n-vouden + k, met $n > 2$

1.6 Kleine toelichting op Programma 1.

1.6.1 (a)

Mset is een verzameling van verzamelingen die ik zelf al heb uitgetest (17 in totaal).

1.6.2 (b)

Hier kun je je eigen functie kiezen in de variabele x.

1.6.3 (c)

Breuken: Ik ben bezig geweest met het zoeken van breuken f zo dat de verzameling $S = \{a, af, af^2, \ldots af^k\}$ een correcte Startverzameling oplevert. Hierbij is \$af = \$ round$(a * f)$ en \$afˆk = \$ round$(af^{k-1} * f)$.

1.6.4 (d)

Maak hier je finale keuze voor S

1.6.5 Opmerking:

In juni 2022 heb ik de gehele software nog herschreven. Ik heb vervolgens de derde machten opnieuw proberen uit te voeren. Dit heb ik gestaakt, omdat de huidige software daar een week voor nodig zou hebben. Ik ben er van overtuigd dat het mogelijk is om de software slimmer te maken, waardoor de snelheid aanzienlijk omhoog gaat. Mocht iemand zich daartoe willen zetten, dan wil ik hem / haar ook wel op weg helpen. Ik denk niet dat ik het zelf ga doen. Er liggen nog zoveel andere projecten te wachten...

[2]:
```
# Programma 1: Bepalen van S

import math

k = 1

# (a)
Mset = [("Leeg", 0, []), #0
    ("Driehoeksgetallen", 9, [3, 6, 10, 15, 21, 28, 36, 45, 55]), #1
    ("Langwerpige getallen +1", 12, [3, 7, 13, 21, 31, 43, 57, 73, 91, 111,␣
 ↪133]), #2
    ("Kwadraten", 15, [4, 9, 16, 25, 36, 49, 64, 81, 100, 121]), #3
    ("Kwadraten +1", 16, [5, 10, 17, 26, 37, 50, 65, 82, 101]), #4
    ("Kwadraten -1", 20, [3, 8, 15, 24, 35, 48, 63, 80, 99, 120]), #5
    ("2x Kwadraten +1", 31, [3, 9, 19, 33, 51, 73, 99, 129, 163, 201]), #6
    ("Derde machten", 0, [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]), #7
    ("Oneven derde machten", 0, [1, 27, 125, 343, 729, 1331, 2197, 3375, 4913,␣
 ↪6859]), #8
    ("Vierde machten", 0, [1, 16, 81, 256, 625, 1296, 2401, 4096, 6561,␣
 ↪10000]), #9
    ("Binom(n,3)", 0, [1, 4, 10, 20, 35, 56, 84, 120, 165, 220]), #10
    ("Binom(n,4)", 0, [1, 5, 15, 35, 70, 126, 210, 330, 495, 715, 1001, 1365]),␣
 ↪#11
    ("Binom(n,5)", 0, [1, 6, 21, 56, 126, 252, 462, 792, 1287, 2002, 3003,␣
 ↪4368]), #12
    ("Maximaal", 2, [3, 5, 9, 13, 21, 29, 43, 61, 89, 125]), #13
    ("Maximaal", 2, [3, 5, 9, 13, 19, 29, 41, 61, 85, 125, 173, 253]), #14
    ("Maximaal", 2, [3, 5, 9, 13, 21, 29, 39, 61, 93, 125, 189, 253]), #15
    ("Maximaal", 2, [3, 5, 9, 13, 21, 27, 45, 57, 93, 117, 189, 237, 381]), #16
    ("Exponentieel", 0, [1, 2, 4, 7, 12, 20, 32, 51, 81, 129, 205, 326, 518,␣
 ↪822, 1304, 2069]), #17 #16
]

# (b)
# Maken van een nieuwe verzameling S
S = []
V = 10
for xx in range(V):
    x = xx + 1
```

```
    #T.append((x*(x+1)*(x+2)*(x+3)*(x+4))//120) # Binom(n,5)
    S.append(x*x+x+1) # Langwerpige getallen +1
#print("Langwerpige getallen +1", S)

# (c)

# f = (p,q) = p/q
# a_1 = 1, en vervolgens a_n = Ceiling(f*a_n-1)

f = 46/29
a = 1
#print (1,a)
F = [1]
for ii in range(15):
    #i = ii + 2
    a = math.ceil(f*a)
    #print(i,a)
    F.append(a)
#print(F)
#S = F

# (d)
Keuze = 7
(txt, n, S) = Mset[Keuze]
print(txt)
print("Gemaakte keuze =", S)
```

```
Derde machten
Gemaakte keuze = [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

1.7 Kleine toelichting op Programma 2.

Mogelijk moet je de ondergrens (N) en bovengrens nog aanpassen.

isolatiewaarde is een getal dat wordt berekend uit het aantal singles en randen: isolatiewaarde $= 3c_0 + c_1$.

Om tot een oplossing te komen moet isolatiewaarde < 3 zijn. Uiteraard is het dan nog niet vanzelfsprekend dat er een oplossing is. Bestudeer maar het voorbeeld van de derde machten. Maar zo lang isolatiewaarde > 2 is wordt N met 1 opgehoogd.

[3]:
```
# Programma 2: Bepalen van C en nagaan of er een oplossing bestaat
N = 3
BOVENGRENS = 1000
isolatiewaarde = 3
while (isolatiewaarde > 2) & (N < BOVENGRENS):
    c0 = 0
    c1 = 0
    N += 1
    OC = []
```

```
    for nn in range(N):
        n = nn + 1
        NC = []
        for s in S:
            t = s-n
            if (t > 0) & (t <= N) & (not(2*n == s)):
                NC.append(t)
        OC.append(NC)
        lnc = len(NC)
        if lnc == 0:
            c0 += 1
        if lnc == 1:
            c1 += 1
    isolatiewaarde = 3*c0 + c1
    if ((c0 < 5) & (c1 < 5)) | (N % 25 == 0):
        print(N,c0,c1)
RN = N
print("Ready")
```

```
4 4 0
5 3 2
6 2 4
25 0 20
50 1 30
75 1 34
100 0 38
125 0 30
150 0 30
175 0 45
200 0 19
215 0 4
216 0 4
217 0 4
218 0 4
225 0 11
250 0 35
275 0 22
293 0 4
294 0 3
295 0 2
Ready
```

2 Kleine toelichting op Programma 3.

2.1 Globale beschrijving van de werking van het programma

Getallen met 1 buur komen zonder meer aan de rand. Bij twee buren zijn er twee mogelijkheden: in het midden met precies die twee buren of aan de rand met een van beide buren. Bij meerdere buren moeten één of twee van de buren gekozen worden als daadwerkelijke buren. Om zorgvuldig te zijn worden al deze mogelijkheden achter elkaar geplaatst in een lijst TeDoen. De eerste mogelijkheid is [], daarna volgen restricties, die in de loop van de berekening naar voren komen. Per NuDoen wordt er gestart met N ketens. In de loop van het programma worden deelketens samengevoegd tot grotere ketens. Als er uiteindelijk één keten overblijft dan is dit een oplossing. Ook kan onderweg blijken dat isolatiewaarde te groot is. In NuDoen staan een aantal paren (t,u) die buren zijn. Deze verbindingen worden geforceerd verbroken.

Bij het aan elkaar plakken van twee ketens moet ook de administratie worden bijgewerkt. Alle getallen in een keten (behalve de randgetallen) zitten in Mid. De getallen die zelf een keten vormen zitten in Sin, de randgetallen zitten in Ran. Zo kan het gebeuren als een single wordt geplakt aan een andere single, dan ontstaan twee randgetallen, als een single aan een randgetal wordt geplakt dan wordt het randgetal middel en de single wordt rand. Zo zijn er nog enkele andere plakbewegingen, die voornamelijk in de stappen 0 t/m 2 worden uitgevoerd.

Bij grotere aantallen getallen kunnen het aantal taken dat moet worden uitgevoerd erg groot worden. In het programma wordt per GEDAANVIEW (10000) uitgevoerde taken aangegeven hoeveel taken nog in het verschiet liggen. Dat maakt de huidige methode suboptimaal. Mogelijk vind jij een snellere methode. Laat het weten!

Op het ogenblik kost het mijn simpele laptop meer dan een dag rekenen om na te gaan dat er voor 302 (derde machten) geen oplossingen zijn.

2.2 variabelen:

- 339 verbose wordt gebruikt t.b.v. hoeveelheid informatie. Met deze variabele kan de informatie die per stap wordt gegeven worden aangepast. Er is nog wel verbetering mogelijk.
- 340 CYCLE Staat normaal op 0. Als je wilt dat begin en eindpunt samenvallen, dan CYCLE = 1.
- 341 START kleinste waarde van N die wordt onderzocht
- 342 RANGE aantal opeenvolgende waarden dat moet worden getest.
- 343 GEDAANVIEW Bij grote berekeningen kan door deze Constante te gebruiken gezien worden dat de berekeningen voortduren. (Moet nog worden geintegreerd in verbose.)
- 344 oplossingen aantal oplossingen dat is gevonden. Momenteel alleen maar 0 of 1. Met een kleine aanpassing kan het ook groter worden.
- 348 N Er wordt getracht een Somketen voor {1..N} te vinden.
- 351 c0 aantal geisoleerde getallen
- 352 c1 aantal getallen met één buur
- 353 OC (OrigineelConnecties) Alle buren per getal. Let op OC[0] bevat de buren van 1 en i.h.a. OC[k] bevat de buren van k+1.
- 358 NC Alle buren voor het getal N. (dit is een tijdelijke variabele, na regel 323 wordt deze variabele niet meer gebruikt.)
- 370 isolatiewaarde bepaalt of er uberhaupt een oplossing bestaat.
- 375 TeDoen Lijst van taken die bekeken moeten worden. TeDoen bestaat uit lijsten. Deze lijsten (taken) bevatten paren (t,u) die buren zijn. Als een taak wordt uitgevoerd (NuDoen

)dan worden alle verbindingen tussen buren t en u verwijderd.
- 376 gedaan ten behoeve van de statistieken. Als een taak uit TeDoen wordt verwijderd, dan wordt gedaan met 1 verhoogd.
- 380 NuDoen standaard het eerste element van TeDoen
- 385 C Bevat een kopie van OC. In de loop van het programma onderdeel (NuDoen) worden er wijzigingen aangebracht in C. Daarom is het belangrijk dat OC in tact blijft.
- 386 Ketens Aanvankelijk zijn dit N lijsten met steeds één element, maar in de loop van het programma worden deze lijsten achter elkaar geplaatst, tot er uiteindelijk één lijst overblijft (als dat mogelijk is).
- 387 Sin De verzameling van singles. Deze getallen zijn als losse getallen bevat in Ketens.
- 388 Ran De verzameling van randgetallen. Deze getallen zijn dus eindpunten van de lijsten in Ketens. Per definitie is het aantal even!
- 389 Mid De lijst van middens. Deze getallen hebben altijd twee buren in Ketens.
- 390 C0 verzameling van geisoleerde getallen.
- 391 C1 verzamelingen van getallen met één buur.

2.3 procedures:

2.3.1 1. PrintC

Afdrukken buren Invoer: C, text C is een connectietabel, text is een toelichting

2.3.2 2. PrintK

Afdrukken keten Invoer: K, text K is een lijst ketens, text is een toelichting

2.3.3 3. ReverseK

Van achter naar voren plaatsen van keten Invoer: K K is een keten

2.3.4 4. SortK

Sorteren van keten (eerste element van keten is kleiner dan laatste), en vervolgens van Ketens.

2.3.5 5. RemoveC

Verwijdert uit alle elementen van C een zeker element *t*

2.3.6 6. RemoveKbe

Als er een keten K (met lengte groter dan 2) is dan moet in het geval dat er een connectie is tussen K[0] en K[len(K)-1] deze connectie worden verwijderd.

2.3.7 7. Join

Samenvoegen van twee ketens tot één keten. Tevens wordt de administratie t.a.v. de verbindingen bijgewerkt. Invoer: K K1, K2 zijn twee ketens Uitvoer: K, een keten

2.3.8 8. Stap0

Telt aantal geisoleerde getallen en randgetallen (zoals in programma 2). Er wordt tevens een eerste stap gezet tot het aan elkaar plakken van ketens. Uitvoer: veranderingen, is een positief getal als Ketens is aangepast.

2.3.9 9. Stap1

Samenvoegen van alle ketens waarvan eindpunt één buur heeft. Uitvoer: veranderingen, is een positief getal als Ketens is aangepast.

2.3.10 10. Stap2

Samenvoegen van geisoleerde getallen met precies twee buren. Uitvoer: veranderingen, is een positief getal als Ketens is aangepast.

2.3.11 11. Stap3

Werkt nog niet.

2.3.12 12. Stappen012

Uitvoeren van de stappen Stap0, Stap1 en Stap2. Uitvoer: (isolatiewaarde, veranderingen), is een paar getallen, veranderingen is een positief getal als Ketens is aangepast. Als isolatiewaarde > 2 dan is er geen oplossing.

[53]:
```
# Programma 3: bepalen van somketen
# procedures:
# 1. PrintC: Afdrukken verbindingen
# 2. PrintK: Afdrukken keten
# 3. ReverseK: Van achter naar voren plaatsen van keten
# 4. SortK: Sorteren van keten (eerste element van keten is kleiner dan␣
 ↪laatste)
# 5. RemoveC: Verwijdert uit alle elementen van C een zeker element t
# 6. RemoveKbe: Verwijdert Connectie K-begin - K-einde, indien van toepassing
# 7. Join: Voegt twee ketens K1 en K2 samen. Voorwaarden zijn dat
#          het laatste getal in K1 en het eerste getal van K2 buren zijn
#          en dat K1 en K2 verschillend zijn
# 8. Stap0: Telt aantal geisoleerde getallen
# 9. Stap1: Samenvoegen van alle ketens waarvan eindpunt 'e'en buur heeft
# 10. Stap2: Samenvoegen van singles met slechts twee buren
# 11. Stap3: Voor de toekomst
# 12. Stappen012: Voert Stap0, Stap1 en Stap2 achtereenvolgens uit.

def PrintC(C, text):
    lc = len(C)
    for i in range(lc):
        if len(C[i]) > 0:
            print(text, i+1, ":", len(C[i]), C[i])
```

```
    return

def PrintK(K, text):
    lk = len(K)
    if lk < 1:
        print(text, ":", "-")
        return
    if lk < 2:
        print(text, ":", K[0])
        return
    k = ""
    for i in range(lk-1):
        k = k + str(K[i]) + " - "
    k = k + str(K[lk-1])
    print(text, ":", k)
    return

def ReverseK(K):
    lk = len(K)
    if lk < 2:
        return K
    CK = []
    RK = []
    # kopie maken
    for k in K:
        CK.append(k)
    for i in range(lk):
        RK.append(CK[-1])
        CK.pop()
    return RK

def SortK():
    changes = 1
    while changes > 0:
        changes = 0
        for K in Ketens:
            #PrintK(K,changes)
            if K[-1] < K[0]:
                changes += 1
                RK = ReverseK(K)
                Ketens.append(RK)
                Ketens.remove(K)
    Ketens.sort()
    return

def RemoveC(t):
    # Deze procedure wordt aangeroepen als
```

```
    # t ergens in een keten is geplaatst.
    # bij alle buren van t wordt t verwijderd
    for c in C:
        if t in c:
            u = C.index(c)+1
            c.remove(t)
            lc = len(c)
            if lc == 0:
                if u in Ran:
                    C1.append(u)
                    Ran.remove(u)
                    Mid.append(u)
                elif u in Sin:
                    C0.append(u)
                    Sin.remove(u)
                    Mid.append(u)
            #if lc == 1:
    return

def RemoveKbe(K):
    # t = K[0] u = K[len(K)-1]
    # Als t + u in S dan moet t - u worden verwijderd.
    if len(K) > 2:
        t = K[0]
        u = K[-1]
        if t + u in S:
            c = C[t-1]
            if u in c:
                c.remove(u)
                lc = len(c)
                if lc == 0:
                    if t in Ran:
                        C1.append(t)
                        Ran.remove(t)
                        Mid.append(t)
            c = C[u-1]
            if t in c:
                c.remove(t)
                lc = len(c)
                if lc == 0:
                    if u in Ran:
                        C1.append(u)
                        Ran.remove(u)
                        Mid.append(u)
    return

def Join(K1, K2):
```

```
    K = []
    if verbose > 10:
        print("Join", Ketens)
        print("Join", K1, K2)
    for k in K1:
        K.append(k)
    for k in K2:
        K.append(k)
    # Administratie
    if len(K1) == 1:
        Sin.remove(K1[0])
        Ran.append(K1[0])
    else:
        t = K1[-1]
        Ran.remove(t)
        Mid.append(t)
        for u in C[t-1]:
            C[u-1].remove(t)
        C[t-1] = []
    if len(K2) == 1:
        Sin.remove(K2[0])
        Ran.append(K2[0])
    else:
        t = K2[0]
        Ran.remove(t)
        Mid.append(t)
        for u in C[t-1]:
            C[u-1].remove(t)
        C[t-1] = []
    if (len(K1) == 1) & (len(K2) == 1):
        t = K1[0]
        u = K2[0]
        C[t-1].remove(u)
        C[u-1].remove(t)
    if verbose > 10:
        print("Join", K)
    return K

def Stap0():
    # Telt aantal geisoleerde en randgetallen
    # plakt tevens stukken keten aan elkaar (als er geen alternatief is)
    for nn in range(N):
        c = C[nn]
        n = nn + 1
        lc = len(c)
        if (lc == 0) & (n in Sin) & (n not in C0):
            C0.append(n)
```

```
        if (lc == 0) & (n in Ran) & (n not in C1):
            C1.append(n)
        if (lc == 1) & (n in Sin) & (n not in C1):
            # n is een single en heeft slechts een buur
            # we verlengen de keten waarop de buur ligt met n
            C1.append(n)
            # we gaan uit van de twee buren t en u
            t = n
            u = c[0]
            # Twee ketens worden geselecteerd
            # Kt bevat t en Ku bevat u
            for K in Ketens:
                if t in K:
                    Kt = K
            for K in Ketens:
                if u in K:
                    Ku = K
            # Mogelijk moeten we Ku omdraaien
            if Ku[-1] == u:
                KuR = ReverseK(Ku)
            else:
                KuR = Ku
            K = Join(Kt, KuR)
            Ketens.append(K)
            Ketens.remove(Kt)
            Ketens.remove(Ku)
            if verbose > 9:
                print("Step0", Ketens)
    return

def Stap1():
    global CYCLE
    # Plakt stukken keten aan elkaar als een randgetal
    # precies een enkele andere buur heeft
    changes = 0
    for nn in range(N):
        c = C[nn]
        n = nn + 1
        lc = len(c)
        if (lc == 1) & (n in Ran):
            t = n
            u = c[0]
            if len(C1) < 2 - 2*CYCLE:
                # We maken wel een extra taak aan voor het geval t een van de␣
 ↪twee
                # randgetallen is. Derhalve wordt in die taak de verbinding t -␣
 ↪u
```

```
                # verwijderd.
                NieuweTaak = []
                for nd in NuDoen:
                    NieuweTaak.append(nd)
                NieuweTaak.append([t,u])
                TeDoen.append(NieuweTaak)
            for K in Ketens:
                if t in K:
                    Kt = K
            for K in Ketens:
                if u in K:
                    Ku = K
            if Kt == Ku:
                # Dit betekent een ongewenste cycle
                CO.append(u)
            else:
                if Kt[0] == t:
                    KtR = ReverseK(Kt)
                else:
                    KtR = Kt
                if Ku[-1] == u:
                    KuR = ReverseK(Ku)
                else:
                    KuR = Ku
                changes += 1
                K = Join(KtR, KuR)
                if verbose > 8:
                    print("Stap1 Join", KtR, KuR)
                Ketens.append(K)
                Ketens.remove(Kt)
                Ketens.remove(Ku)
                if verbose > 9:
                    PrintK(Ketens, "Stap1")
    if verbose > 8:
        print("Stap1", changes)
    return changes

def Stap2():
    global CYCLE
    # Plakt stukken keten aan elkaar als een geisoleerd getal
    # precies twee andere buren heeft
    changes = 0
    for nn in range(N):
        c = C[nn]
        n = nn + 1
        lc = len(c)
        if (lc == 2) & (n in Sin):
```

```
            # We gaan nu uit van het stukje keten u - t - v
            t = n
            u = c[0]
            v = c[1]
            if len(C1) < 2 - 2*CYCLE:
                # Er worden twee nieuwe taken aangemaakt. In beide gevallen zal
                # t een randpunt zijn
                NieuweTaak = []
                for nd in NuDoen:
                    NieuweTaak.append(nd)
                NieuweTaak.append([t,u])
                TeDoen.append(NieuweTaak)
                NieuweTaak = []
                for nd in NuDoen:
                    NieuweTaak.append(nd)
                NieuweTaak.append([t,v])
                TeDoen.append(NieuweTaak)
            for K in Ketens:
                if t in K:
                    Kt = K
            for K in Ketens:
                if u in K:
                    Ku = K
            for K in Ketens:
                if v in K:
                    Kv = K
            if not(Ku == Kv):
                changes += 1
                if Ku[0] == u:
                    KuR = ReverseK(Ku)
                else:
                    KuR = Ku
                if Kv[-1] == v:
                    KvR = ReverseK(Kv)
                else:
                    KvR = Kv
                K1 = Join(KuR, Kt)
                K = Join(K1, KvR)
                Ketens.append(K)
                Ketens.remove(Kt)
                Ketens.remove(Ku)
                Ketens.remove(Kv)
                if verbose > 9:
                    PrintK(Ketens, "Stap2")
    if verbose > 8:
        print("Stap2", changes)
    return changes
```

```
# Einde Stap2

def Stap3():
    # Hier gaan we de getallen met precies twee buren verbinden
    return

def Stappen012():
    # Hier worden de stappen 0, 1 en 2 achter elkaar uitgevoerd
    verandering = 0
    Stap0()
    c1 = len(C1) + 2*CYCLE
    c0 = len(C0)
    isolatiewaarde = 3*c0 + c1
    if isolatiewaarde < 3:
        verandering1 = Stap1()
        verandering2 = Stap2()
        Stap0()
        c0 = len(C0)
        c1 = len(C1) + 2*CYCLE
        isolatiewaarde = 3*c0 + c1
        verandering = verandering1 + verandering2
    return (isolatiewaarde, verandering)

# Hier begint het hoofdprogramma

#- 0: silent, eerste oplossing wordt weergegeven, en uiteindelijk aantal␣
 ↪oplossingen
#- 1: alleen oplossingen worden geteld en gedaan per 10000 weergegeven
#- 2: alleen oplossingen worden geteld en TeDoen per 100 weergegeven
#- 3: informatie over nieuw toegevoegd geen oplossingen worden afgedrukt
#- 4: alle oplossingen worden afgedrukt
#- 5: tevens informatie over nieuw toegevoegd
#- 6: tevens informatie over isolated
#- 7: tevens Ran2
#- 8: Geheel Ran en Sin
#- 9: tevens Ketens
#- 10: Tevens stappen
verbose = 3
CYCLE = 0 # vervang 0 door 1 en er zijn geen eindpunten meer toegestaan
START = 305
RANGE = 1
GEDAANVIEW = 10000
oplossingen = 0
#while oplossingen == 0:
for NN in range(RANGE):
    oplossingen = 0
    N = NN + START
```

```
    if verbose > 3:
        print("*** N =", N)
    c0 = 0 # aantal singles
    c1 = 2 * CYCLE # aantal randen
    OC = [] # OverzichtConnecties
    for nn in range(N):
        # In deze for-loop wordt OC berekend
        # Tevens worden c0 en c1 berekend.
        n = nn + 1
        NC = []
        # Niet vergeten om eerst Programma 1 en 2 uit te voeren!
        for s in S:
            t = s-n
            if (t > 0) & (t <= N) & (not(2*n == s)):
                NC.append(t)
        OC.append(NC)
        lnc = len(NC)
        if lnc == 0:
            c0 += 1
        if lnc == 1:
            c1 += 1
    isolatiewaarde = 3*c0 + c1
    if (verbose > 0) & (c0 < 5) & (c1 < 5) & (isolatiewaarde > 2):
        print(N,"Status: Geisoleerd =", c0, "Een buur =", c1)
    if (isolatiewaarde < 3) & (oplossingen == 0):

        TeDoen = [[]]
        gedaan = 0
        printgedaan = GEDAANVIEW-1

        while (len(TeDoen)  > 0) & (oplossingen == 0):
            NuDoen = TeDoen[0]
            TeDoen.pop(0)
            gedaan += 1
            if (len(NuDoen) > 0) & (verbose > 7):
                print("NuDoen:", NuDoen)
            C = []
            Ketens = []
            Sin = []
            Ran = []
            Mid = []
            C0 = []
            C1 = []
            runs = 0
            restart = 0
            # Hier onder worden Sin en Ketens geinitialiseerd
            for nn in range(N):
```

```
            # We beginnen met N ketens
            # Alle n worden toegevoegd aan Sin
            n = nn+1
            Sin.append(n)
            Ketens.append([n])
        # We maken een Kopie van OC en plaatsen die in C
        for oc in OC:
            # Hier wordt een kopie gemaakt van OC
            # C = OC is onjuist, want dan wijzigt OC mee
            c = []
            for o in oc:
                c.append(o)
            C.append(c)
        # Hier worden extra verbindingen verwijderd, conform NuDoen
        for nd in NuDoen:
            t = nd[0]
            u = nd[1]
            C[u-1].remove(t)
            C[t-1].remove(u)

        Stap0()
        c1 = len(C1) + 2*CYCLE
        c0 = len(C0)
        isolatiewaarde = 3*c0 + c1

        if isolatiewaarde < 3 :
            veranderingen = 1
            while (isolatiewaarde < 3) & (veranderingen > 0):
                (isolatiewaarde, veranderingen) = Stappen012()
            if gedaan > printgedaan:
                printgedaan += GEDAANVIEW
                print("gedaan", gedaan, "tedoen", len(TeDoen))
        if isolatiewaarde < 3 :
            SortK()
            if (len(Sin) == 0) & (len(Ran) <= 2):
                oplossingen = 1
                PrintK(Ketens, "Oplossing")
            elif verbose > 5:
                PrintK(Ketens, "Geen oplossing")
                PrintC(C, "Geen oplossing")
                Ran.sort()
                Mid.sort()
                print("Geen oplossing, Sin", Sin)
                print("Geen oplossing, Ran", Ran)
                print("Geen oplossing, Mid", Mid)
                # In feite is het programma nog niet af.
                # Hier moet nog meer code onder, om nog meer verbindingen
```

```
                    # te verwijderen
        print(gedaan)
        print(N, "!!Aantal oplossingen:", oplossingen)
print("Ready", N)
```

```
Oplossing : [95, 248, 264, 79, 137, 206, 10, 115, 228, 284, 59, 157, 186, 30,
34, 182, 161, 55, 288, 224, 119, 97, 246, 266, 77, 139, 204, 12, 15, 201, 142,
74, 269, 243, 100, 25, 191, 152, 64, 279, 233, 110, 106, 237, 275, 68, 148, 195,
21, 6, 2, 123, 220, 292, 51, 165, 178, 38, 305, 207, 136, 80, 263, 249, 94, 31,
185, 158, 58, 285, 227, 116, 9, 18, 198, 145, 71, 272, 240, 103, 113, 230, 282,
61, 155, 188, 28, 36, 180, 163, 53, 290, 222, 121, 4, 23, 193, 150, 66, 277,
235, 108, 17, 199, 144, 72, 271, 241, 102, 114, 229, 283, 60, 156, 187, 29, 35,
181, 162, 54, 289, 223, 120, 96, 247, 265, 78, 138, 205, 11, 16, 200, 143, 73,
270, 242, 101, 24, 192, 151, 65, 278, 234, 109, 107, 236, 276, 67, 149, 194, 22,
5, 3, 122, 221, 291, 52, 164, 179, 37, 27, 189, 154, 62, 281, 231, 112, 104,
239, 273, 70, 146, 197, 19, 8, 117, 226, 286, 57, 159, 184, 32, 93, 250, 262,
81, 135, 208, 304, 39, 177, 166, 50, 293, 219, 124, 1, 7, 20, 196, 147, 69, 274,
238, 105, 111, 232, 280, 63, 153, 190, 26, 99, 244, 268, 75, 141, 202, 14, 13,
203, 140, 76, 267, 245, 98, 118, 225, 287, 56, 160, 183, 33, 92, 251, 261, 82,
134, 209, 303, 40, 176, 167, 49, 294, 218, 125, 91, 252, 260, 83, 133, 210, 302,
41, 175, 168, 48, 295, 217, 126, 90, 253, 259, 84, 132, 211, 301, 42, 174, 169,
47, 296, 216, 127, 89, 254, 258, 85, 131, 212, 300, 43, 173, 170, 46, 297, 215,
128, 88, 255, 257, 86, 130, 213, 299, 44, 172, 171, 45, 298, 214, 129, 87, 256]
1
305 !!Aantal oplossingen: 1
Ready 305
```

2.4 Voorbeelden

Alleen de eerste en laatste regel worden getoond : - $\frac{1}{2}n(n+1)$, 1 t/m 9: $3-7-8-2-4-6-9-1-5$ - n^2+n+1, 1 t/m 12: $7-6-1-12-9-4-3-10-11-2-5-8$ - n^2+1, 1 t/m 16: $5-12-14-3-7-10-16-1-9-8-2-15-11-6-4-13$ - n^2-1, 1 t/m 20: $12-3-5-10-14-1-2-13-11-4-20-15-9-6-18-17-7-8-16-19$ - $2n^2+1$, 1 t/m 32: $1-18-15-4-29-22-11-8-25-26-7-12-21-30-3-16-17-2-31-20-13-6-27-24-9-10-23-28-5-14-19-32$ - $\frac{1}{6}n(n+1)(n+2)$, 1 t/m 78: $12-72-48-36-20-64-56-28-7-77-43-41-15-69-51-5-30-\cdots-13-71-49-35-21-63-57-27-29-55-65-19-37-47-73-11-24-60$ - $\frac{1}{24}n(n+1)(n+2)(n+3)$, 1 t/m 286: $39-171-159-51-279-216-114-96-234-261-69-141-189-21-105-\cdots-26-9-201-129-81-249-246-84-126-204-6-120-210-285-45-165$ - n^3-1, 1 t/m 309: $13-111-231-280-62-153-189-26-98-244-267-75-140-202-309-\cdots-299-43-172-170-45-297-214-128-87-255-256-86-129-213-298-44-171$ - n^3, 1 t/m 305: $95-248-264-79-137-206-10-115-228-284-59-157-186-30-34-182-161-55-288-224-119-97-246-266-77-139-204-12-\cdots-88-255-257-86-130-213-299-44-172-171-45-298-214-129-87-256$ - $\frac{1}{120}n(n+1)(n+2)(n+3)(n+4)$, 1 t/m 1716: $1001-286-506-1496-1507-495-792-1210-77-175-1112-890-397-\cdots--1539-463-824-1178-109-143-1144-858-429-1573-1430-572-715-1287$ - $(2n+1)^3$, 1 t/m 2069: $3-24-101-1230-967-364-1833-1542-655-676-1521-1854-$

343 − 988 − · · · − 1235 − 962 − 369 − 1828 − 1547 − 650 − 681 − 1516 − 1859 − 338 − 993 − 1204 − 127 - n^4, 1 t/m 6508: 29 − 52 − 573 − 5988 − 4012 − 2549 − 1547 − 5014 − 4986 − 1575 − 2521 − 4040 − 5960 − · · · − 1930 − 4631 − 5369 − 1192 − 2904 − 3657 − 6343 − 218 − 38 − 587 − 5974 − 4026 − 2535 − 1561 − 5000