

# Bundeswettbewerb Mathematik 2022 Runde 1

## Aufgabe 1

Sei  $k_n$  die Anzahl an Nüssen am  $n$ -ten Tag.

Wir wissen das  $k_0$ , die Anzahl, bevor noch Nüsse dazukommen, gleich 2022 ist.

Wir wissen weiterhin, dass  $k_1 - k_0 = 2$ ,  $k_2 - k_1 = 4$ ,  $k_3 - k_2 = 6$ , ... ist.

Damit können wir folgende Formel aufstellen:

$$k_n = k_{n-1} + 2n = 2022 + \sum_{i=1}^n (2i) = 2022 + 2 * \sum_{i=1}^n i$$

Mithilfe der Summenformel von Gauss ergibt dies:

$$= 2022 + 2 * \frac{n(n+1)}{2} = 2022 + n(n+1)$$

Damit wir die Nüsse gleichmäßig auf die 5 aufteilen könne, muss  $k_n \equiv 0 \pmod{5}$  sein.

Also muss gelten:

$$2022 + n(n+1) \equiv 0 \pmod{5} \Leftrightarrow n(n+1) \equiv 3 \pmod{5}$$

Wir unterscheiden nun 5 Fälle für  $n$ , nach modulo 5, und untersuchen  $n(n+1)$ :

$n \pmod{5}$	$(n+1) \pmod{5}$	$(n(n+1)) \pmod{5}$
0	1	0
1	2	2
2	3	$6 \equiv 1$
3	4	$12 \equiv 2$
4	0	0

Dabei fällt auf, dass es keinen Fall gibt, in dem  $n(n+1) \equiv 3 \pmod{5}$  ist, wie es für die Aufteilung auf 5 notwendig wäre.

Damit kann es niemals einen Tag  $n$  geben, an dem die Nüsse gleichmäßig auf alle 5 Eichhörnchen aufgeteilt werden können.

■

## Aufgabe 2

Sei  $n$  die Anzahl an Mitteldreiecken  $+ 1$ , also die Anzahl an eingezeichneten Dreiecken inklusive dem gleichseitigen.

Wir stellen zunächst fest, dass die Verbindung der Seitenmitten eines gleichseitigen Dreiecks wieder ein gleichseitiges Dreieck hervorbringt, da das Dreieck symmetrisch aufgebaut sein muss, nämlich genau die Symmetrieachsen des äußeren gleichseitigen Dreiecks. Das einzige Dreieck, was dies erfüllt, ist wieder das gleichseitige Dreieck.

Weiterhin liegen die Mitten des inneren Dreiecks wieder auf den Verbindungen des äußeren Dreiecks mit den gegenüberliegenden Eckpunkten.

Seien  $A_1$ ,  $B_1$  und  $C_1$  die Eckpunkte des äußersten Dreiecks, und  $M$  der Mittelpunkt des Dreiecks (An dieser Stelle sei angemerkt, dass im gleichseitigen Dreieck Inkreis-, Umkreis- und Mittenschnittpunkt zusammen fallen; somit ist es egal, welchen dieser wir an dieser Stelle betrachten)

Wir bezeichnen nun die Mitten jeweils mit  $A_2$  (die Mitte von  $[B_1C_1]$ ),  $B_2$  (die Mitte von  $[A_1C_1]$ ) und  $C_2$  (die Mitte von  $[A_1B_1]$ )

Die Mitten der Seiten  $[A_2B_2]$ ,  $[A_2C_2]$  und  $B_2C_2$  bezeichnen wir analog mit  $C_3$ ,  $B_3$  und  $A_3$ .

Dies setzen wir so lange fort, bis wir  $A_n$ ,  $B_n$  und  $C_n$  eingezeichnet haben.

Wir können nun diese Anordnung von Dreiecken als Graphen  $G_n$  bezeichnen. Dafür wollen wir zunächst eine Verbindungsvorschrift erstellen, die Angibt, welche Knoten miteinander Verbunden sind.

Mit folgenden Punkten ist jeder Knoten  $X_i$  bis auf den Mittelpunkt verbunden, sofern diese Existieren:

- Allen anderen  $Y_i$ , da sie bereits ein Dreieck bilden
- Allen anderen  $X_n$  und  $M$ , da diese alle auf einer Geraden liegen
- Allen  $Y_{i+1}$ , da die benachbarten auf den Dreiecksseiten liegen, die an dieser Ecke angrenzen
- Allen  $Y_{i-1}$ , da  $Y_i$  entweder die Mitte darstellt

Ein Dreieck in diesem Graphen zeichnet sich nun dadurch aus, dass wir drei Eckpunkte suchen, die verbunden sind, und nicht auf einer Geraden liegen.

Drei Punkte liegen in unserem Graphen auf einer Geraden, wenn:

- Sie alle vom selben 'type' sind (z.B. alle vom Type  $A_n$ ), und optional dem Mittelpunkt
- Sie der Form  $X_i$ ,  $Y_i$ ,  $Z_{i-1}$  sind, also zwei Eckpunkte und dessen Mittelpunkt

Wir wollen dieses Problem nun mithilfe eines Computerprogramms lösen. Ich verwende in der finalen Implementierung python, da es keine Limitationen in vielen Bereichen hat.

Wir beginnen aber zunächst aber mit einigen Definitionen.

Wir nennen die Funktion, die entscheidet, ob zwei Knoten verbunden sind, 'connected', und die Funktion, ob drei Punkte auf einer Geraden liegen, 'on\_one\_line'.

Wir können uns eine Menge  $M_n$  aller Eckpunkte erzeugen, sie enthält alle  $A_i$ ,  $B_i$  und  $C_i$  mit  $0 < i \leq n$  und  $M$ .

Wir können weiterhin eine Menge  $G_n$  erzeugen, die alle Verbindungen als  $\{X_n, Y_n\}$  darstellt, erzeugen. Diese Menge ist:

$$\{\{x, y\} | x \in M_n \wedge y \in M_n \wedge \text{connected}(x, y)\}$$

Aus dieser Menge können wir nun die Menge aller Dreiecke  $D_n$  heraus konstruieren, auf folgende Weise:

$$D_n = \{\{x, y, z\} | \{x, y\} \in G_n \wedge \{x, z\} \in G_n \wedge \{y, z\} \in G_n \wedge \text{on\_one\_line}(x, y, z)\}$$

Damit haben wir alle Bausteine für ein Program, was uns die Gesamtanzahl an anzeigbaren Dreiecke angibt. Die Anzahl ist einfach der Betrag von  $D_n$ .

Folgender Pseudo-Code implementiert dies:

```

Funktion connected(X, Y)
{
  Falls (X[0] == Y[0]) → Wahr
  Falls (X[0] == 'M' oder Y[0] == 'M') → Wahr
  Falls (Betrag(x[1] - Y[1]) ≤ 1) → Wahr
  Sonst → Falsch
}

Funktion on_one_line(X, Y, Z)
{
  Wenn Betrag({X[0], Y[0], Z[0]} - {'M'}) == 1 → Wahr
  s = [X[1], Y[1], Z[1]]
  s.sort()
  Wenn s[0] == s[1] == s[2] - 1 → Wahr
  Sonst → Falsch
}

n = Eingabe('n: ')
K_n = {A_i | i ∈ ℕ ∧ i ≤ n} + {B_i | i ∈ ℕ ∧ i ≤ n} + {C_i | i ∈ ℕ ∧ i ≤ n} + {'M'}
G_n = {{x, y} | x ∈ K_n ∧ y ∈ K_n ∧ connected(x, y)}
D_n = {{x, y, z} | {x, y} ∈ G_n ∧ {x, z} ∈ G_n ∧ {y, z} ∈ G_n ∧ on_one_line(x, y, z)}
Ausgabe(Betrag(D_n))

```

Wir wollen nun diesen Code in python umsetzen (zu finden ab dem 09.03.2022 unter [https://github.com/uuk0/BuMa2022/blob/main/runde%201/Aufgabe2\\_final.py](https://github.com/uuk0/BuMa2022/blob/main/runde%201/Aufgabe2_final.py))

---

Listing 1: Python code

---

```
import itertools

n = int(input("N:"))

G = set()
D = set()

def connected(a, b):
    if (a[0] == "M" or b[0] == "M"):
        return True

    if (abs(a[1] - b[1]) <= 1):
        return True

    return False

def on_one_line(a, b, c):
    if len({a[0], b[0], c[0]} - {"M"}) == 1:
        return True

    x, y, z = sorted((a[1], b[1], c[1]))
    if (x == y == z - 1):
        return True

    return False

vertices = {"M", 0}

for i in range(1, n + 1):
    for c in ("A", "B", "C"):
        vertices.add((c, i))

for a, b in itertools.combinations(vertices, 2):
    if connected(a, b):
        G.add((min(a, b, key=lambda e: e[1]), max(a, b, key=lambda e: e[1])))

for a, b, c in itertools.combinations(vertices, 3):
    if on_one_line(a, b, c):
        continue

    a, b, c = sorted((a, b, c), key=lambda e: e[1])
    D.add((a, b, c))

print(len(D))
```

---

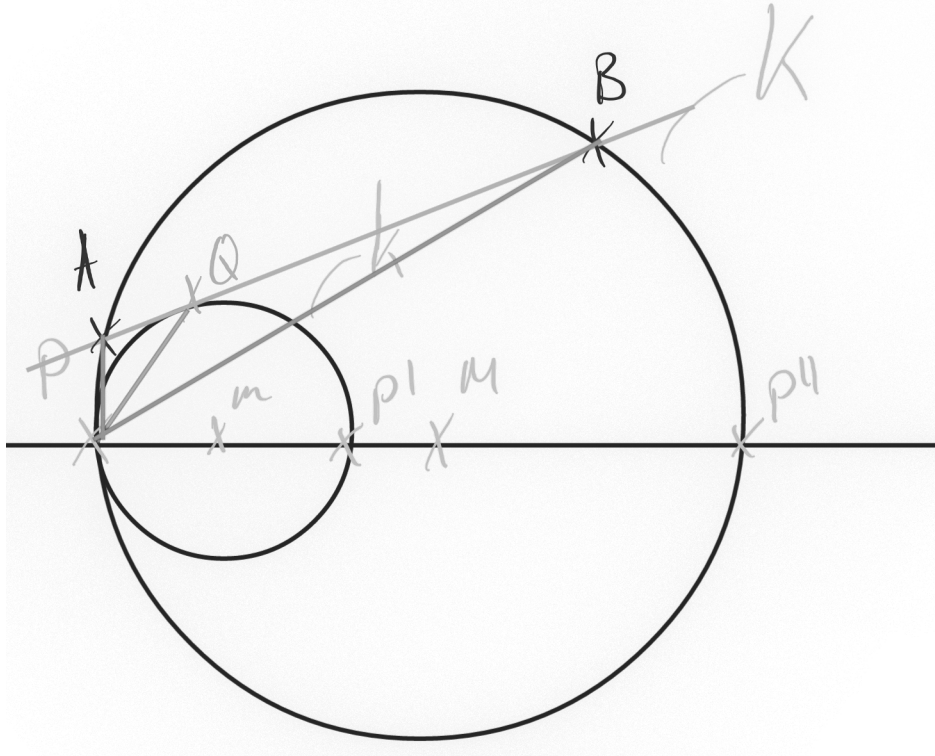
Führt man obiges Program für verschiedene Werte für  $n$  aus, ergeben sich folgende Werte:

$n$	$ D_n $
1	4
2	23
3	90
...	...
8	1985
9	2844
10	3919
...	...

Somit sind die gesuchten Werte  $n = 9$  mit 2844 einzeichnen-baren Dreiecken und insgesamt 8 Mitteldreiecke

### Aufgabe 3

Skizze: (für eine farbige Version siehe <https://github.com/uuk0/BuMa2022/>)



Wir wollen beweisen, dass  $\angle APQ = \angle QPB$  ist.

Da uns nur Winkel interessieren, interessieren uns nur die Seitenverhältnisse, also  $\frac{mP'}{MP''}$ , und der Winkel  $\alpha$  unter  $Q$ , also  $\angle QPP'$ .

O.b.d.A. können wir  $0^\circ < \alpha < 180^\circ$  setzen und  $\overline{MP''} = 1$ , ersteres aufgrund der Achsensymmetrie zur Horizontalen, zweiteres aufgrund der Ähnlichkeitsregeln.

**Aufgabe 4**

Wir stellen zunächst fest, dass der größte Teiler einer Zahl immer sie selbst ist. Das heist, sofern die Zahl selbst nicht durch drei Teilbar ist, ist  $a_k = k$ .

Wir definieren nun 
$$\phi(k) = \begin{cases} k & \text{wenn } k \bmod 3 \neq 0 \\ \phi(\frac{k}{3}) & \text{sonst} \end{cases}$$

Dabei stellt  $\phi(k)$  gleichzeitig  $a_k$  dar.

Wir stellen weiterhin fest, dass jede dritte Zahl durch drei Teilbar ist, jede neunte durch neun, jede 27 durch 27, und so weiter.

Eine Eigenschaft der Zahlen im Dreiersystem ist, dass sie alle 3 einen ‘roll-over‘ haben an der 1-er Stelle, alle  $3^2$  an der  $3^2$  Stelle, alle  $3^3$  an der  $3^3$  Stelle usw.

Damit steht an der 1-er Stelle alle 3 eine 1, an der 3-er Stelle alle  $3^2$ , an der  $3^2$  Stelle alle  $3^3$ , usw.

Weiterhin stellen wir fest, dass eine Summe von natürlichen Zahlen genau dann durch 3 teilbar ist, wenn  $\Sigma(a_n \bmod 3) \equiv 0 \pmod{3}$