Äquivalente Charakterisierung von Dreieckszahlen und deren effiziente Implementierung in Python

Abstract:
Eine *Dreieckszahl* ist bekanntlich eine natürliche Zahl, die die Summe von k aufeinander folgenden natürlichen Zahlen beginnend bei 1 ist. Nach einem bekannten Beispiel vollständiger Induktion gilt

$$\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2} \tag{1}$$

für alle $n \in \mathbb{N}$, sodass sich Dreieckszahlen auch via (1) charakterisieren lassen. Mathematisch betrachtet ist damit das Problem gelöst. Betrachten wir jedoch die Fragestellung, ob eine gegebene Zahl $m \in \mathbb{N}$ eine Dreieckszahl ist, so können wir dieses Problem mit (1) algorithmisch nur durch eine Iteration lösen. Dies ist für wachsendes m jedoch unnötig langsam, weshalb wir in diesem Artikel ausgehend von (1) eine äquivalente Charakterisierung von Dreieckszahlen beweisen werden, welche in konstanter Zeit in Python implementiert wird.

Teil 1: Die äquivalente Charakterisierung

Wir beginnen mit einer Präzisierung der oben angerissenen Problemstellung. Es sei $n \in \mathbb{N}$. Dann ist n gemäß (1) genau dann eine Dreieckszahl, wenn folgendes erfüllt ist:

$$\exists l \in \mathbb{N}\colon n = \frac{1}{2} \cdot l \cdot (l+1). \tag{2}$$

Es handelt sich bei (2) um eine quadratische Gleichung in l, die nur natürliche Zahlen als Lösung zulässt. Daher können wir äquivalent umformen und erhalten zunächst

$$(2) \Leftrightarrow \exists l \in \mathbb{N}\colon l^2 + l - 2n = 0$$

und durch Anwendung der p-q-Formel und Berücksichtigung der Tatsache, dass nur positive Lösungen in Frage kommen, folgendes Resultat:

$$(2) \Leftrightarrow \exists l \in \mathbb{N}\colon l = -\frac{1}{2} + \sqrt{\frac{1}{4} + 2n} \in \mathbb{N}.$$

Wie wir sehen, hängt die Beantwortung der Frage nur vom Term $-\frac{1}{2} + \sqrt{\frac{1}{4} + 2n}$ und damit nur von n ab. Wir können diesen Term noch vereinfachen und erhalten

$$-\frac{1}{2} + \sqrt{\frac{1}{4} + 2n} \in \mathbb{N} \Leftrightarrow \sqrt{1+8n} - 1 \in 2\mathbb{N}. \tag{3}$$

Damit haben wir folgendes Ergebnis bewiesen.

Satz.
Es sei $n \in \mathbb{N}$ und $\Delta \subset \mathbb{N}$ die Menge aller Dreieckszahlen. Dann gilt:

$$n \in \Delta \Leftrightarrow \sqrt{1+8n} - 1 \in 2\mathbb{N}. \tag{4}$$

Wir überzeugen uns an ein paar Beispielen von der Wirkung der Charakterisierung (4):

n	1	10	28	45	2	9	33	127
$\sqrt{1+8n}-1$	2	8	14	18	$\sqrt{17}-1$	$\sqrt{73}-1$	$\sqrt{265}-1$	$3\sqrt{113}-1$
n Dreieckszahl?	ja	ja	ja	ja	nein	nein	nein	nein

Teil 2: Effiziente Umsetzung in Python

Zunächst wollen wir uns die „klassische“ Implementierung eines Algorithmus anschauen, die das in Teil 1 skizzierte Problem löst.

```
1  def isDreieckszahlKlassisch(n):
2      dreieckszahl = 0
3      zaehler = 1
4      while dreieckszahl <= n:
5          if dreieckszahl == n:
6              return True
7          dreieckszahl += zaehler
8          zaehler += 1
9      return False
```

In grober Abschätzung benötigt dieser Algorithmus $\mathcal{O}(n)$ Laufzeit, in jedem Fall ist aber evident, dass die Laufzeit nicht $\mathcal{O}(1)$ ist. Wir benutzen nun obigen Satz bzw. genauer (3), um einen Algorithmus in $\mathcal{O}(1)$ anzugeben.

```
1  import math
2
3  def isDreieckszahlVerbessert(n):
4      x = -1/2 + math.sqrt(1/4 + 2*n)
5      return x % 1 == 0.0
```

Der Beweis der Korrektheit dieses Algorithmus ist wegen (3) bzw. (4) offensichtlich. Da die Implementierung der Wurzel durch das Standardpaket in konstanter Zeit arbeitet und alle weiteren verwendeten Operationen (Addition, Multiplikation, Modulo) ebenfalls in konstanter Zeit arbeiten, ist die Laufzeit dieses Algorithmus $\mathcal{O}(1)$ und damit arbeitet dieser Algorithmus wie behauptet in konstanter Zeit.