001 Problem Solving: Rummy Numbers (Solution)

Maximilian Noppel[1]

[1]**max [at] vspace.one**

Zusammenfassung

SPOILER ALERT: Dieses Dokument enthält meine Lösung!

Wir lösen wieder ein Teilproblem von Rummy. Im Vergleich zum 005 CodeGolfing geht es dieses mal nur darum eine Lösung implementieren. Die Länge des Sourcecodes ist dabei egal. Außerdem schauen wir uns dieses mal nicht die Farben der Karten sondern die Zahlen an.

Schwierigkeit: 

EINFÜHRUNG

Rummy ist ein relativ einfaches Kartenspiel. Jede Karte hat ein Zahl von 1 bis 13 und eine von vier Farben (schwarz, blau, geld und rot)[1]. Das Ziel vom Rummy ist es Karten zu Reihen aus mindestens drei Karten zusammenzulegen. Dabei müssen entweder alle Karten die gleich Farbe haben und aufsteigenden Zahlen oder alle Karten die gleiche Zahl aber unterschiedliche Farben. Keine Karte darf dabei übrigbleiben oder in einer Zweier-Reihe sein. Beim letzten CodeGolfing (005) haben wir dieses Problem für Karten gelöst, die alle die gleiche Zahl hatten. Wir mussten also nur Dreier/Viererreihen aus verschieden-farbigen Karten bilden.

Nun lösen wir das Problem für Zahlen, die alle die gleiche Farbe haben. Hier können wir deutlich längere Reihen bilden. Zum Beispiel `[1] [2] [3] [4] [5] [6] [7] ...`. Außerdem können wir nach der höchsten Zahl (13) wieder von vorne anfangen. Die Reihe `[12] [13] [1]` ist also eine gültige Reihe.

AUFGABE

Die Aufgabe ist also wie folgt: Gegeben seien 13 Zahlen zwischen 0 und 100. Jede dieser Zahlen representiert die entsprechenden Anzahl an Karten. Die erste Zahl gibt also die Anzahl an einsern, die zweite die Anzahl an zweiern und so weiter an. Diese 13 Zahlen werden über die Kommandozeile an das Programm übergeben. Für Python beispielsweise so:

```
$ python your_script.py 4 9 34 75 10 90 45 23 7 76 25 19 95
```

Oder für C Programme so

```
$ ./main 4 9 34 75 10 90 45 23 7 76 25 19 95
```

Das Programm soll ausgeben ob die gegeben Karten ordnungsgemäß abgelegt werden können. Also ob es eine Ablage gibt, so dass keine Karte alleine oder in einer Zweireihe liegen bleibt. Wie die Ablage tatsächlich aussieht ist nicht interessant. Natürlich muss jede Karte muss abgelegt werden.

Das Programm signalisiert das Ergebnis *Ablage möglich* über den Rückgabewert 0 und *Ablage nicht möglich* über einen Wert $\neq 0$, z.B. -1

UPLOAD

Da ich keine Lust haben meinen Code am Ende wieder so zu verunstalten und da beim letzten CodeGolfing die Beteiligung an der Abschlussveranstaltung so gering war, dürft ihr die Aufgabe nach Belieben lösen und eure Lösung ins Wiki packen. Link: `https://wiki.vspace.one/doku.php?id=treffen:problemsolving001`

[1] Eigentlich gibt es noch zwei Joker, aber die ignorieren wir hier mal.

MEINE LÖSUNG

Sei $k \in \mathbb{R}$ die mögliche Anzahl an Zahlen. Im Fall von Rummy eben konstant $k = 13$. Unsere Eingabe ist dann $n = \mathbb{R}^k$. Wir notieren n_0, wenn wir die Anzahl an Karten mit einer eins darauf meinen, n_1 wenn wir die Anzahl an Karten mit einer zwei meinen, und so weiter. In der Informatik fängt man mit der 0 an zu zählen. Zur Einfachheit nortieren wir n_{i-1} wenn wir eigentlich $n_{i-1 \mod k}$ meinen. Das bedeutet n_{i-1} ist die Karte links von n_i und n_{i+1} die Karte rechts von n_i. Für $k = 13$ liegt also n_{12} links von n_0.

Grundidee
Die Grundidee meines Algorithmes ist, dass man für jeden Stapel nur zwei Stapel nach links und zwei Stapel nach rechts anschauen muss. Auf diese 5 Werte wenden wir dann Bedingungen an, die für jeden Stapel halten müssen. Im Folgenden betrachten wir also erstmal nur einen Stapel n_i und seine vier Nachbarn $n_{i-2}, n_{i-1}, n_{i+1}$ und n_{i+2}.

Die Grundidee ist, dass jeder Stapel seine Karten *loswerden muss*. Dazu hat jeder Stapel drei verschiedene Möglichkeiten.

- Er kann nach links anküpfen. Also beiden Stapel links von sich nutzen. Hier kann er $min(n_{i-1}, n_{i-2})$ Karten loswerden.
- Analog für rechts.
- Sollte noch Karten übrig bleiben, müssen diese als eine Dreierreihe mit n_i also Zentrum abgelegt werden.

Wenn danach immer noch Karten übrigbleiben kann der Stapel nicht abgelegt werden und die Ausgabe muss $\neq 0$ sein.

Formalisierung
Im nächsten Schritt formalisieren wir diese Idee. Wir definieren die Anzahl an Karten die n_i nach links los werden kann als

$$minL_i := \min(n_{i-1}, n_{i-2})$$

und analog

$$minR_i := \min(n_{i+1}, n_{i+2})$$

Wenn wir diese Karten abziehen blieben noch $n_i - minL_i - minR_i$ Karten übrig. Aber auch von unseren direkten Nachbarn n_{i-1} und n_{i+1} sind nun weniger Karten übrig. Um genau zu sein $n_{i-1} - minL_i$ und $n_{i+1} - minR_i$. Wir können also noch maximal

$$\min(n_{i-1} - minL_i, n_{i+1} - minR_i)$$

Karten als Dreireihe anlegen.

Wenn also einer oder mehrere Stapel existieren, die wir nicht komplett verteilen können, ist eine Ablage nicht möglich und wir müssen $\neq 0$ ausgeben. Andern falls ist eine Ablage möglich.

Mathematisch kann das über den Exist-Quantor $\exists$ ausdrücken. $\exists i : ...$ bedeutet soviel wie "*Es existiert ein i, so dass ...*". $\exists$ können wir auch als logischen Ausdruck interpretieren. Wenn ein i mit der entsprechenden Eigenschaft existiert, dann ist der Ausdruck 1, andernsfalls 0.

Unsere finale formale Lösung ist also:

$$\exists i : n_i - minL_i - minR_i > \min(n_{i-1} - minL_i, n_{i+1} - minR_i)$$

Übersetzung in Python
Die obige Formel lässt sich sehr einfach in Python Sourcecode übersetzen:

```
1  import sys
2  import numpy as np
3
4  def solve():
5      n = np.array(sys.argv[1:], dtype=int)
6      assert(len(n) == 13)
7
8      for i in range(13):
9          minl = min(n[(i-1) % 13], n[(i-2) % 13])
10         minr = min(n[(i+1) % 13], n[(i+2) % 13])
11         if n[i % 13] - minl - minr > min(n[(i-1) % 13]-minl, n[(i+1) % 13]-minr):
12             return 1
13
14     return 0
```

In den Zeilen 1+2 importieren wir lediglich zwei gängige Bibliotheken. *sys* ist eine Python-Standardbibliothek während *numpy* extra installiert werden muss. Die Zeile 5 parst lediglich die Kommandozeilenargumente in ein Array mit 13 Elementen. Zeile 6 wirft einen Fehler falls weniger als 13 Zahlen übergeben wurden. Danach wird für jeden Stapel n_i überprüft ob die Bedingung gilt. In den Zeilen 9 und 10 werden die beiden Werte $minL_i$ und $minR_i$ erzeugt. Sollte ein Stapel n_i gefunden werden, auf den die Bedingung zutrifft wird direkt 1 zurückgeben (Zeilen 11+12). Sollte das auf keinen einzigen Stapel zutreffen, ist eine Ablage möglich und es wird 0 zurückgegeben (Zeile 14).