

# Lösung Aufgabe 5 „Wichteln“

## Lösungsidee

Wir verwenden einen optimierenden Algorithmus. Die Schüler werden in einer Reihenfolge durchgegangen. In dieser werden zuerst erste Wünsche erfüllt, dann Zweite, und schließlich Dritte.

Diese Reihenfolge verbessern wir solange möglich, indem wir immer zwei Schüler vertauschen, die neue Reihenfolge ausprobieren, und beibehalten falls diese besser ist.

## Korrektheit

Wenn eine Reihenfolge nicht mehr verbesserbar ist, muss sie die eine optimale Reihenfolge sein.

Es bleibt zu zeigen, dass jede optimale Geschenkeverteilung auch als Reihenfolge darstellbar ist.

Bei einer optimalen Geschenkeverteilung wird jedem Schüler:

- Der 1. Wunsch
- Oder der 2. Wunsch
- Oder der 3. Wunsch
- Oder kein Wunsch

erfüllt.

Damit nun einem Schüler sein erster Wunsch erfüllt wird, brauchen wir ihn nur an eine Stelle zu stellen, wo das Geschenk noch nicht vergeben sein wird.

Analog können wir auch zweite und dritte Wunsch erfüllen, indem wir den betreffenden Schüler an einer Stelle in der Reihenfolge einsetzen, wo sein erster Wunsch bzw. sein erster und sein zweiter Wunsch schon vergeben sein werden (da Schüler, die weiter vorne in der Reihenfolge stehen, diese erhalten).

## Komplexität

- Das Ausprobieren einer Reihenfolge ist in linearer Laufzeit  $O(n)$  möglich.
- Alle möglichen Tausche sind  $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$
- Somit ergibt sich eine **kubische Komplexität von  $O(n^3)$**

## Umsetzung

Implementierung in der modernen und performanten Programmiersprache Go.

## Kompilieren

`go build` (erzeugt `a5-Wichteln`) oder `go build main.go` (erzeugt `main`)

## Verwendung

`go run main.go <pfad>` oder `./main <pfad>`

Beispiel: ./main beispieldaten/wichteln1.txt

## Ausgabe

Lösung:

<In Textrichtung Ausgabe der erhaltenen Geschenke getrennt mit Komma und  
↪ Leerzeichen, aufsteigend nach Nummer des erhaltenden Schülers>

Erfüllte Wünsche: <Anzahl erfüllte 1. Wünsche>, <Anzahl erfüllte 2. Wünsche>,  
↪ <Anzahl erfüllte 3. Wünsche>

Zeit verstrichen: <Verstrichene Zeit in Sekunden> s

## Bibliotheken

- `fmt`: Ausgabe, Formattierung
- `io/ioutil`: Einlesen der Datei
- `os`: Programmargumente
- `regexp`: Regulärer Ausdruck zum Extrahieren der Zahlen
- `strconv`: String/Integer-Konversion
- `strings`: Auftrennen des Dateiinhalts nach Zeilen
- `time`: Zeitmessung

## Typen

### Verteilung

- Wert als vergleichbare Zahl: Anzahl erfüllter Wünsche so kodiert, dass zwei Verteilungen über ihren Wert vergleichbar sind
- Erhaltene Geschenke als „Slice“ mit `[Schülernummer - 1]` = Geschenknummer
- Vergebene Geschenke als „Slice“ mit `[Geschenknummer - 1]` = `true` wenn vergeben, sonst `false`

## Eingabe

Das erste Programmargument ist der Pfad zur Aufgabendatei. Diese wird gelesen und an Zeilenumbrüchen aufgetrennt. Die Wünsche jedes Schülers werden mithilfe eines einfachen regulären Ausdrucks als Strings extrahiert und zu Zahlen konvertiert. Schließlich erhält man eine Slice mit `[Schülernummer-1]` = 3-er-Array{Geschenknummer 1. Wunsch - 1, Geschenknummer 2. Wunsch - 1, Geschenknummer 3. Wunsch - 1}

## Verarbeitung

Eine Funktion probiert die aktuelle Reihenfolge aus, indem zuerst erste, dann zweite, und schließlich dritte Wünsche in der Reihenfolge erfüllt werden. Hierfür werden vergebene und erhaltene Geschenke mit einer Verteilung nachgehalten. Der Wert wird als Zahl (`uint64`) zur Basis  $n + 1$  mit  $n$  = Anzahl Schüler dargestellt. Hierbei stehen erste Wünsche an erster Stelle, 2. an 2. und 3. an 3.:  $wert = (n + 1)^2$ .

$erfuellteErsteWuensche + (n+1) \cdot erfuellteZweiteWuensche + erfuellteDritteWuensche$ .  
Der Vergleich zweier Verteilungen wird somit zu einem einfachen Zahlenvergleich.

Wir beginnen mit der Einlesereihenfolge der Schüler als Startreihenfolge.

Dann probieren wir solange Swaps (Täusche) aus, bis keiner der möglichen Swaps mehr zu einer Verbesserung des Wertes der aktuellen Reihenfolge führt. Einen Swap machen wir rückgängig, wenn sich herausstellt, dass dieser zu keiner Verbesserung geführt hat. Insgesamt ergeben sich 3 geschachtelte Schleifen.

## Ausgabe

Zunächst wird die Verteilung komplettiert: Schüler, die keinen Wunsch erfüllt bekommen haben, erhalten die erstbesten freien Geschenke (in Reihenfolge der Geschenkenummern). Hierfür gehen wir die erhaltenen Geschenke nach Schüler durch, finden leer ausgehende Schüler, und gehen dann die Geschenke durch, anfangend nach den schon durchgegangenen Geschenken, bis wir ein noch nicht vergebenes finden. Dieses erhält der Schüler.

Schließlich geben wir „einfach“ die erhaltenen Geschenke aus. Diese trennen wir in einer Zeile mit Leerzeichen und Komma, sonst durch Zeilenumbrüche. Hierbei sorgen wir für Zeilen, die nicht länger als 80 Zeichen werden.

Die Anzahl der erfüllten Wünsche extrahieren wir aus dem Wert der Verteilung mittels Division mit Rest und geben diese zusammen mit der verstrichenen Zeit ebenfalls aus.

## Quellcode

### main.go

```
1 package main
2
3 import (
4     "fmt"
5     "io/ioutil"
6     "os"
7     "regexp"
8     "strconv"
9     "strings"
10    "time"
11 )
12
13 // Verteilung - Wert als vergleichbare Zahl, von Schülern erhaltene Geschenke,
14 // ↪ vergebene Geschenke
15 type Verteilung struct {
16     wert                uint64
17     erhalteneGeschenke []uint
18     vergebeneGeschenke []bool
```

```

18 }
19
20 func main() {
21     // Zeitmessung
22     nanos := time.Now().UnixNano()
23     // Eingabe
24     if len(os.Args) != 2 {
25         println("Verwendung: <pfad>")
26         return
27     }
28     // Einlesen
29     text, err := ioutil.ReadFile(os.Args[1])
30     if err != nil {
31         panic(err)
32     }
33     lines := strings.Split(string(text), "\n")
34     _anzahl, err := strconv.Atoi(lines[0])
35     if err != nil {
36         panic(err)
37     }
38     // Anzahl Schüler
39     anzahl := uint(_anzahl)
40     // Basis für schnellen Vergleich
41     basis := uint64(anzahl + 1)
42     // Schüler: [Nummer - 1] = {Nummer 1. Wunsch - 1, Nummer 2. Wunsch - 1,
43     ↪ Nummer 3. Wunsch - 1}
44     schueler := make([][3]uint, anzahl)
45     for s := uint(0); s < anzahl; s++ {
46         var wunsch [3]uint
47         // Regulärer Ausdruck: Alle Zahlen (= Wünsche) aus der Zeile
48         ↪ extrahieren
49         wunschStr :=
50         ↪ regexp.MustCompile("[0-9]+").FindAllString(lines[s+1], 3)
51         for w := 0; w < 3; w++ {
52             // Wünsche einlesen
53             wunschW, err := strconv.Atoi(wunschStr[w])
54             if err != nil {
55                 panic(err)
56             }
57             wunschW--
58             wunsch[w] = uint(wunschW)
59         }
60         schueler[s] = wunsch
61     }
62     // Verfahren

```

```

60 // Reihenfolge, in der Schüler Wünsche erhalten
61 reihenfolge := make([]uint, anzahl)
62 // Probiert eine Reihenfolge, gibt "Wert" der resultierenden
   ↳ Verteilung als Zahl zurück
63 probiereReihenfolge := func() (verteilung Verteilung) {
64     // Wert
65     verteilung.wert = 0
66     // Vergebene Geschenke: [Geschenk] = Vergeben?
67     verteilung.vergebeneGeschenke = make([]bool, anzahl)
68     // Erhaltene Geschenke: [Schüler] = Geschenk
69     verteilung.erhalteneGeschenke = make([]uint, anzahl)
70     stelle := basis * basis
71     for i := 0; i < 3; i++ {
72         erfuellteWuensche := uint64(0)
73         for _, s := range reihenfolge {
74             // Schüler in Reihenfolge durchgehen
75             wunsch := schueler[s][i]
76             if verteilung.erhalteneGeschenke[s] != 0 ||
   ↳ verteilung.vergebeneGeschenke[wunsch] {
77                 // Schüler schon "abgespeist" oder
   ↳ Geschenk schon vergeben
78                 continue
79             }
80             // Zähler erhöhen, Geschenk zuordnen & als
   ↳ vergeben markieren
81             erfuellteWuensche++
82             verteilung.erhalteneGeschenke[s] = wunsch + 1
83             verteilung.vergebeneGeschenke[wunsch] = true
84         }
85         verteilung.wert += stelle * erfuellteWuensche
86         stelle /= basis
87     }
88     return
89 }
90 // Startreihenfolge: 0 bis anzahl aufsteigend
91 for i := range reihenfolge {
92     reihenfolge[i] = uint(i)
93 }
94 // Startvergleichswerte für die Reihenfolge
95 besteVerteilung := probiereReihenfolge()
96 for {
97     verbesserung := false
98     for i := range reihenfolge {
99         for j := range reihenfolge[i+1:] {
100             // Probiere "swaps"

```

```

101         reihenfolge[i], reihenfolge[j] =
↪ reihenfolge[j], reihenfolge[i]
102         // Werte nach Swap
103         andereVerteilung := probiereReihenfolge()
104         if andereVerteilung.wert >
↪ besteVerteilung.wert {
105             // Bessere Verteilung! Aktualisieren
106             besteVerteilung = andereVerteilung
107             verbesserung = true
108         } else {
109             // Ansonsten: Swap wieder rückgängig
↪ machen
110             reihenfolge[i], reihenfolge[j] =
↪ reihenfolge[j], reihenfolge[i]
111         }
112     }
113 }
114 // ...solange die Reihenfolge verbessert werden kann
115 if !verbesserung {
116     break
117 }
118 }
119
120 // Beste Verteilung komplettieren: "Leer ausgehenden" (kein Wunsch
↪ erfüllt) Schülern erstbeste Geschenke geben
121 var i uint
122 for s, g := range besteVerteilung.erhalteneGeschenke {
123     if g == 0 {
124         // Kein Wunsch erfüllt, kein Geschenk zugeordnet
125         for {
126             if !besteVerteilung.vergebeneGeschenke[i] {
127                 // Erstbestes übriges Geschenk
↪ erhalten
128                 besteVerteilung.erhalteneGeschenke[s]
↪ = i + 1
129                 i++
130                 break
131             }
132             i++
133         }
134     }
135 }
136
137 // Ausgabe
138 fmt.Println("Lösung:")

```

```

139 // Erste Zeile der Lösung beginnt mit dem Geschenk, das der erste
    ↪ Schüler erhält
140 zeile := strconv.Itoa(int(besteVerteilung.erhalteneGeschenke[0]))
141 for _, g := range besteVerteilung.erhalteneGeschenke[1:] {
142     geschenk := strconv.Itoa(int(g))
143     if len(zeile)+len(geschenk)+2 > 80 {
144         // Passt nicht mehr in Zeile: Maximale Zeilenlänge von
            ↪ 80
145         fmt.Println(zeile)
146         // Neue Zeile beginnen
147         zeile = ""
148     } else {
149         // In gleicher Zeile anhängen mit trennendem Komma
150         zeile += ", "
151     }
152     // Zugeordnetes Geschenk zur Ausgabe hinzufügen
153     zeile += geschenk
154 }
155 if zeile != "" {
156     // Letzte Zeile ausgeben
157     fmt.Println(zeile)
158 }
159 // Wert der Verteilung (in erfüllten Wünschen) & verstrichene Zeit
    ↪ ausgeben
160 // Dafür Rückrechnung:
161 // - wert/basis² = erfüllte 1. Wünsche
162 // - (wert/basis) % basis = erfüllte 2. Wünsche
163 // - wert % basis = erfüllte 3. Wünsche
164 fmt.Printf(`
165 Erfüllte Wünsche: %d, %d, %d
166
167 Zeit verstrichen: %v s
168 `, besteVerteilung.wert/(basis*basis), (besteVerteilung.wert/basis)%basis,
    ↪ besteVerteilung.wert%basis, float32(time.Now().UnixNano()-nanos)/1e9)
169 }

```

## Beispiele

### wichteln1.txt

Lösung:

6, 2, 1, 3, 5, 4, 7, 10, 9, 8

Erfüllte Wünsche: 6, 0, 2

Zeit verstrichen: 0.00040208 s

**wichteln1.txt**

Lösung:

6, 2, 1, 3, 5, 4, 7, 10, 9, 8

Erfüllte Wünsche: 6, 0, 2

Zeit verstrichen: 0.00040208 s

**wichteln2.txt**

Lösung:

4, 5, 6, 1, 2, 3, 7, 8, 9, 10

Erfüllte Wünsche: 3, 0, 0

Zeit verstrichen: 0.000151697 s

**wichteln3.txt**

Lösung:

2, 20, 29, 8, 1, 3, 5, 12, 4, 28, 6, 9, 14, 23, 26, 30, 11, 7, 16, 10, 19, 13  
27, 15, 17, 18, 22, 21, 24, 25

Erfüllte Wünsche: 15, 6, 1

Zeit verstrichen: 0.001349349 s

**wichteln4.txt**

Lösung:

4, 21, 14, 9, 28, 12, 3, 16, 10, 19, 11, 20, 7, 23, 2, 25, 1, 27, 17, 13, 22,  
↪ 26  
15, 6, 30, 24, 18, 8, 5, 29

Erfüllte Wünsche: 15, 4, 3

Zeit verstrichen: 0.002028563 s

**wichteln5.txt**

Lösung:

5, 6, 7, 18, 27, 2, 4, 9, 25, 16, 14, 13, 19, 15, 10, 8, 26, 23, 20, 3, 1, 21  
22, 11, 24, 28, 30, 12, 17, 29



Erfüllte Wünsche: 13, 1, 7

Zeit verstrichen: 0.001637866 s

#### **wichteln6.txt**

Lösung:

1, 27, 38, 23, 30, 21, 37, 45, 55, 51, 9, 33, 86, 26, 2, 42, 28, 39, 15, 35, 6  
4, 31, 14, 84, 12, 74, 7, 78, 25, 54, 5, 18, 19, 34, 53, 36, 48, 43, 87, 32, 83  
29, 22, 47, 85, 44, 50, 16, 52, 40, 65, 24, 56, 62, 57, 59, 46, 13, 81, 60, 75  
61, 67, 10, 80, 64, 66, 68, 69, 70, 90, 3, 71, 49, 72, 58, 11, 17, 73, 8, 63,  
↪ 76  
77, 79, 41, 20, 82, 88, 89

Erfüllte Wünsche: 37, 3, 21

Zeit verstrichen: 0.010158948 s

#### **wichteln7.txt**

Lösung:

484, 1, 5, 146, 367, 380, 2, 980, 3, 866, 571, 309, 868, 73, 854, 660, 6, 551  
304, 861, 255, 722, 143, 103, 977, 616, 994, 352, 80, 960, 602, 44, 9, 14, 15  
347, 949, 594, 85, 683, 753, 23, 24, 25, 31, 181, 968, 318, 182, 467, 705, 773  
630, 843, 390, 531, 743, 455, 109, 471, 149, 235, 696, 697, 730, 885, 234, 879  
138, 18, 997, 561, 402, 116, 686, 526, 545, 891, 919, 798, 480, 831, 956, 121  
399, 644, 677, 175, 490, 501, 932, 836, 475, 613, 560, 126, 260, 549, 156, 129  
321, 233, 857, 313, 206, 188, 13, 769, 589, 341, 620, 177, 647, 63, 35, 435,  
↪ 910  
32, 847, 801, 817, 325, 151, 82, 631, 715, 682, 316, 168, 711, 921, 899, 694  
655, 106, 429, 246, 202, 790, 783, 128, 37, 33, 893, 579, 894, 39, 506, 108,  
↪ 124  
40, 105, 781, 487, 96, 731, 607, 922, 671, 733, 962, 420, 42, 453, 75, 450,  
↪ 469  
46, 734, 48, 495, 49, 167, 684, 577, 189, 902, 923, 964, 52, 889, 752, 265,  
↪ 947  
580, 564, 963, 991, 54, 492, 698, 60, 488, 364, 489, 566, 639, 45, 499, 707,  
↪ 338  
593, 288, 818, 337, 710, 383, 605, 973, 708, 64, 281, 74, 290, 562, 10, 212,  
↪ 651  
934, 68, 22, 70, 76, 57, 72, 356, 604, 209, 799, 161, 512, 838, 606, 633, 78,  
↪ 86  
535, 888, 427, 534, 884, 999, 548, 615, 89, 91, 335, 98, 155, 441, 100, 628

1000, 556, 447, 574, 656, 110, 306, 621, 612, 115, 350, 256, 458, 998, 892,  
↪ 623

211, 302, 250, 20, 841, 597, 117, 118, 539, 638, 830, 87, 376, 834, 409, 410  
120, 758, 229, 353, 950, 414, 310, 137, 59, 139, 299, 756, 858, 803, 140, 142  
157, 7, 144, 485, 918, 127, 145, 170, 222, 472, 354, 174, 176, 326, 286, 476  
135, 426, 203, 395, 796, 179, 180, 193, 190, 674, 745, 198, 95, 336, 208, 134  
718, 938, 215, 805, 672, 217, 274, 152, 986, 939, 218, 220, 221, 224, 227, 663  
12, 164, 975, 247, 196, 936, 465, 849, 125, 183, 438, 989, 371, 625, 614, 965  
228, 675, 36, 553, 28, 896, 231, 845, 687, 543, 113, 751, 424, 807, 542, 132  
658, 421, 204, 873, 619, 236, 640, 150, 725, 797, 239, 237, 387, 862, 820, 241  
554, 191, 439, 301, 244, 245, 666, 249, 253, 112, 257, 259, 261, 263, 806, 111  
267, 305, 741, 343, 270, 271, 851, 690, 540, 945, 197, 55, 714, 88, 537, 272  
275, 43, 122, 277, 280, 65, 169, 673, 283, 84, 251, 21, 903, 287, 289, 765,

↪ 627

136, 474, 558, 293, 314, 565, 524, 214, 97, 294, 71, 437, 160, 449, 268, 240  
296, 416, 300, 311, 483, 315, 388, 898, 365, 322, 544, 324, 744, 864, 327, 205  
329, 443, 519, 508, 676, 629, 461, 591, 333, 330, 94, 331, 332, 133, 583, 342  
344, 940, 829, 466, 363, 345, 187, 349, 521, 511, 840, 498, 351, 357, 748, 652  
890, 800, 359, 924, 360, 875, 557, 362, 428, 497, 303, 941, 901, 525, 368, 162  
679, 653, 369, 457, 370, 373, 377, 379, 966, 372, 517, 382, 384, 185, 810, 178  
51, 440, 186, 386, 780, 19, 269, 394, 346, 276, 378, 398, 400, 403, 754, 913  
678, 114, 576, 844, 104, 404, 366, 166, 154, 417, 406, 412, 418, 505, 419, 423  
486, 943, 230, 603, 432, 430, 433, 786, 777, 883, 56, 470, 933, 552, 434, 252  
431, 436, 83, 361, 243, 509, 223, 291, 448, 662, 415, 451, 931, 720, 131, 452  
456, 867, 732, 689, 77, 460, 904, 959, 463, 200, 897, 464, 473, 477, 478, 298  
828, 479, 515, 688, 481, 491, 496, 422, 573, 503, 972, 927, 504, 81, 729, 41  
510, 147, 405, 514, 559, 848, 601, 216, 634, 516, 582, 536, 26, 242, 50, 522  
523, 201, 53, 355, 500, 527, 713, 774, 529, 445, 626, 778, 530, 816, 396, 192  
532, 446, 397, 219, 766, 292, 348, 680, 533, 538, 482, 860, 248, 541, 546, 93  
454, 547, 493, 62, 700, 101, 958, 870, 550, 567, 569, 572, 581, 584, 586, 706  
811, 442, 717, 297, 590, 592, 595, 407, 596, 920, 723, 312, 319, 598, 320, 599  
608, 254, 764, 609, 528, 375, 611, 328, 877, 617, 618, 173, 622, 226, 946, 624  
153, 792, 649, 635, 632, 739, 643, 462, 213, 232, 645, 374, 648, 650, 659, 808  
285, 518, 392, 661, 92, 664, 411, 755, 610, 770, 667, 668, 600, 669, 172, 323  
791, 494, 681, 691, 693, 699, 701, 984, 702, 703, 709, 967, 520, 692, 339, 712  
716, 4, 66, 719, 393, 141, 724, 878, 578, 641, 171, 881, 740, 727, 728, 912,

↪ 735

737, 988, 738, 742, 747, 996, 468, 69, 771, 749, 750, 761, 736, 408, 258, 762  
507, 27, 929, 763, 768, 587, 772, 67, 928, 148, 210, 775, 11, 779, 785, 788,

↪ 953

835, 804, 102, 789, 793, 282, 642, 38, 795, 29, 787, 646, 809, 784, 760, 194  
685, 812, 776, 308, 987, 637, 8, 17, 782, 926, 813, 389, 880, 262, 401, 159,

↪ 670

819, 340, 636, 821, 824, 502, 825, 826, 585, 832, 833, 358, 767, 839, 842, 295  
184, 563, 846, 225, 815, 850, 284, 746, 278, 852, 726, 853, 856, 859, 863, 413

865, 869, 908, 871, 130, 855, 872, 874, 695, 882, 909, 886, 827, 802, 887, 757  
895, 665, 900, 195, 905, 199, 906, 307, 264, 575, 207, 16, 907, 914, 279, 381  
915, 822, 266, 814, 30, 704, 119, 916, 334, 917, 925, 163, 930, 974, 34, 935  
657, 937, 942, 425, 513, 568, 944, 238, 969, 654, 948, 90, 951, 952, 954, 837  
955, 158, 957, 317, 123, 273, 961, 99, 970, 971, 385, 976, 61, 978, 979, 444  
981, 165, 982, 983, 459, 985, 876, 990, 47, 570, 721, 588, 107, 911, 79, 992  
993, 759, 823, 391, 58, 555, 995, 794

Erfüllte Wünsche: 541, 126, 55

Zeit verstrichen: 20.312572 s

### **wichteln8.txt**

Zusätzliches Beispiel:

5  
2 3 1  
5 4 1  
3 2 5  
1 4 2  
2 5 4

Lösung:

2, 5, 3, 1, 4

Erfüllte Wünsche: 4, 0, 1

Zeit verstrichen: 0.000129278 s