

# 34. BUNDESWETTBEWERB INFORMATIK

RUNDE 1  
01.09. - 30.11.2015

## Aufgabe 3

### Flaschenzug

25. Oktober 2015

**Eingereicht von:** *Der Skript-Tim*

Tim Hollmann  
ich@tim-hollmann.de

**Verwaltungs-Nr.:** 34.00003

Ich versichere hiermit, die vorliegende Arbeit ohne unerlaubte fremde Hilfe entsprechend den Wettbewerbsregeln des Bundeswettbewerb Informatik angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

*Tim Hollmann*, den 25. Oktober 2015

## INHALTSVERZEICHNIS

<b>Inhaltsverzeichnis</b>	<b>2</b>
<b>1 Lösungsidee</b>	<b>3</b>
1.1 Verallgemeinerung . . . . .	3
1.2 Naiver Algorithmus . . . . .	3
1.3 Dynamische Programmierung . . . . .	3
<b>2 Umsetzung</b>	<b>4</b>
2.1 Memo . . . . .	4
2.2 Rekursive Funktion . . . . .	5
2.3 Kompat und Kommandozeilenparameter . . . . .	6
<b>3 Beispiele</b>	<b>7</b>
3.1 Anwendung auf gegebene Beispiele . . . . .	7
3.2 Eigene Beispiele . . . . .	7
3.3 Beispielhafte Ausgabe . . . . .	8
<b>4 Quelltext</b>	<b>8</b>
<b>Abbildungsverzeichnis</b>	<b>14</b>

# 1 LÖSUNGSIDEE

## 1.1 VERALLGEMEINERUNG

Beim vorliegenden Problem handelt es sich um ein erweitertes Partitionierungsproblem<sup>12</sup>, bei dem es gilt, die Anzahl der Darstellungen einer natürlichen Zahl  $N$  als Summe von  $|K|$  natürlichen Summanden zu ermitteln. Zudem dürfen bestimmte Summanden  $i$  einen bestimmten Wert  $K_i$  nicht überschreiten. Man könnte schreiben:

*Die Anzahl der möglichen Kombinationen,  $N$  Flaschen auf  $|K|$  Behälter aufzuteilen, entspricht der Anzahl der wahren Aussagen für*

$$\sum_{i=1}^{|K|} \in [0 \dots K_i] = N$$

## 1.2 NAIVER ALGORITHMUS

Die Lösung dieses Problems wird anhand eines Beispieles verdeutlicht:  $K = \{A, B, C\}$

$$[0 \dots A] + [0 \dots B] + [0 \dots C] = N$$

Dieses Problem könnte sich durch einen naiven Algorithmus folgender Art lösen lassen

```

1 for(int a = 0; a <= A; ++a){
2     for(int b = 0; b <= B; ++b){
3         for(int c = 0; c <= C; ++c){
4             if (a + b + c == N) counter ++; //Aussage wahr?
5         }
6     }
7 }
```

Dieser Alorithmus besitzt eine Laufzeitkomplexität  $\mathcal{O}(K) = \prod_{i=1}^{|K|} (K_i+1)$ ; das ist schlecht. Die folgende Bearbeitung der Aufgabe widmet sich deshalb (wie so oft) der Optimierung des naiven Algorithmus.

## 1.3 DYNAMISCHE PROGRAMMIERUNG

Wie beim naiven Algorithmus inkrementiere ich den ersten Behälter  $K_1$  von 0 bis  $A$ . Die Anzahl möglicher Lösungen entspräche dann der Summe von

$$0 \dots B + 0 \dots C = N - 0$$

$$0 \dots B + 0 \dots C = N - 1$$

$$0 \dots B + 0 \dots C = N - 2$$

$$\vdots$$

$$0 \dots B + 0 \dots C = N - A$$

<sup>1</sup><https://de.wikipedia.org/wiki/Partitionierungsproblem>

<sup>2</sup>evtl. auch ein spezielles Teilsummenproblem (<https://de.wikipedia.org/wiki/Teilsummenproblem>).

Dabei gilt es dann immer, die entstehende Gleichung  $[0 \dots B] + [0 \dots C] = N - X$  zu lösen. Dazu bietet sich wieder das selbe Durchinkrementieren (diesmal vom  $B$ ) an. Schnell wird deutlich, dass sich eine Lösung dazu rekursiv definieren lässt:

```

1  int anzahl( Behältermenge K, Flaschenanzahl N ){
2      // Nur ein Behälter:
3      if (|K| == 1) return (K[0] >= N) ? 1 : 0;
4
5      // Mehrere Behälter:
6      Behälter x = K[0]; // Einen Behälter abspalten
7      Behältermenge rest = K - x;
8
9      for( i = 0; i <= x ) // Abgespaltenen Behälter iterieren
10     {
11         counter += anzahl( rest, N - i);
12     }
13
14     return counter;
15 }

```

Ist lediglich ein Behälter übrig, also die maximale Tiefe der Rekursion erreicht, kommt es lediglich darauf an, ob der übrige Behälter groß genug ist, die gegebene Anzahl Flaschen aufzunehmen (`return (K[0] >= N)? 1 : 0`).

Bis jetzt besitzt dieser *neue* Algorithmus trotz aller Rekursivität im Hinblick auf Laufzeitkomplexität keinen nennenswerten Unterschied zum naiven Algorithmus. Den Unterschied werden Abbruchbedingungen machen; es kann im Vorherein abgesehen werden, ob eine tiefere Rekursion zielführend ist. So zum Beispiel macht es keinen Sinn, eine tiefere Rekursion durchzuführen, wenn die Summe der Behälterkapazitäten nicht ausreicht, um die Flaschenanzahl aufzunehmen. Auch  $i$  darf nie größer als die übrige Flaschenanzahl sein; sonst wird die Anzahl der Flaschen negativ (`anzahl( rest, N - i)`).

Ein weiterer erheblich leistungssteigernder Faktor liegt in der Natur des sogenannten *dynamischen Programmierens*. Neben Rekursion zielt diese auf das Speichern und Wiederverwenden bereits erreichten Wissens ab.<sup>3</sup> Konkret im Algorithmus bedeutet das, dass der Rückgabewert der Funktion in Verbindung mit den übergebenen Parametern gespeichert, und bei jedem Aufruf der Funktion zunächst nach den aktuellen Parametern im Speicher gesucht und - falls gefunden - direkt daraus zurückgegeben wird.

## 2 UMSETZUNG

Die Umsetzung erfolgte in C++ (Standard C++-11) unter Visual C++ 2015.

### 2.1 MEMO

Die konkrete Implementierung des Memos sieht so aus

<sup>3</sup>[https://de.wikipedia.org/wiki/Dynamische\\_Programmierung](https://de.wikipedia.org/wiki/Dynamische_Programmierung)

```
17 std::map<std::tuple<std::vector<behaelter>, uint>, bigInt> memo;
```

Das `tuple`-Objekt besteht aus den Argumenten der rekursiven Funktion (siehe **2.2**); der Liste der Behälterkapazitäten `std::vector<behaelter>` (`behaelter` als `typedef` für `int`) und der Behälteranzahl `N` `uint` (`unsigned int`). Es stellt den Schlüssel dar, unter dem in der `memo`-Map der Rückgabewert der Funktion als `bigInt` (`unsigned long long int`) abgelegt wird.

## 2.2 REKURSIVE FUNKTION

```
176 auto Application::anzahl(const std::vector<behaelter>& z, uint ←
    z_summe, uint n) -> bigInt
177 {
178
179     // Suche im Memo
180     std::tuple<std::vector<behaelter>, int> t(z, n);
181
182     if (memo.find(t) != memo.end())
183         return memo.at(t);
184
185     // Nur ein Behälter
186     if (z.size() <= 1)
187     {
188         int i = ((z[0] >= n) ? 1 : 0);
189         memo[t] = i;
190         return i;
191     }
192
193     // Bei mehreren Behältern den ersten herausnehmen (und den rest in ←
        temporärem Vektor speichern)
194     std::vector<behaelter> temp_b;
195     for (int i = 1; i < z.size(); i++) { temp_b.push_back(z[i]); }
196
197     // Neue Kapazitätensumme
198     int temp_zustand_summe = z_summe - z[0];
199
200     bigInt s = 0;
201     // Ersten Behälter durchiterieren und rekursiv für übrige Behälter ←
        aufrufen
202     for (int i = 0; i <= z[0] && n - i >= 0; i++) {
203         if (temp_zustand_summe < n - i) continue;
204         s += anzahl(temp_b, temp_zustand_summe, n - i);
205     }
206
207     memo[t] = s; // Ergebnis speichern
208
209     return s;
210 }
```

Zusätzlich zu der Behälterliste  $K_z$  und der Flaschenanzahl  $N_n$  führt die Funktion noch einen weiteren Parameter mit: `int z_summe`. Dieser gibt lediglich die Summe der Behälterkapazitäten in  $z$  an. So ist es nicht nötig, diese vor Gebrauch in Zeile **203** jedes mal neu zu errechnen, denn sie ergibt sich sehr einfach durch Abzug des zu inkrementierenden Behälters (Z.198).

## 2.3 KOMPILAT UND KOMMANDOZEILENPARAMETER

Der Quellcode wurde von mir unter Windows für x86 und x64, sowie unter Linux für x64 Bit kompiliert.

### Hinweis - Windows-Runtime

Standardmäßig erfordern die unter Visual Studio 2015 kompilierten Programme die „*Visual C++ Redistributable für Visual Studio 2015*“ - Laufzeitkomponenten. Falls Sie sich diese nicht herunterladen wollen, liegen die Programme auch in einer Version bei, die keine Laufzeitkomponenten benötigen, dafür aber Laufzeitfehler verursachen könnten. Sie liegen als `Win[Bit]Alternativ.exe` bei.

### Hinweis - Linux-Kompilat

Die unter Linux per gcc kompilierte Programmversion ist im Vergleich zu den Windows-Varianten erheblich leistungsschwächer. Dies könnte an der besseren Code-Optimierung bei Visual Studio liegen. Benutzen Sie deshalb nach Möglichkeit die Windows-Varianten; das Programm braucht unter Linux für das letzte Beispiel sogar das 10-fache (45 Sekunden statt ca. 4).

**Kommandozeilenparameter** Das Programm kann die Beispiel-Dateien nach Syntax der Materialvorlagen von `bundeswettbewerb-informatik.de` automatisch auslesen; verwenden Sie dazu den Kommandozeilenparameter `-f`.

```
> Flaschenzug.exe -help
```

```
-h -help -? Hilfe-Ausgabe mit Auflistung dieser Kommandozeilenparameter
```

```
-f -file [filename] Pfad/Dateiname der Datei mit Syntax der BwInf-Materialvorlagen
```

Ist keine Datei angegeben, diese kann nicht geöffnet, ausgelesen oder verwertet (da in anderer Syntax) werden, fällt das Programm in eine manuelle Eingabe der Behälterdaten zurück.

### 3 BEISPIELE

#### 3.1 ANWENDUNG AUF GEGEBENE BEISPIELE

Datei	$N$	$ K $	Verteilungsmöglichkeiten	Zeit
flaschenzug0.txt	7	2	2	0s
flaschenzug1.txt	5	3	13	0s
flaschenzug2.txt	10	3	48	0s
flaschenzug3.txt	30	20	6209623185136	0s
flaschenzug4.txt	3000	11	17361307039603189181	3,5s
flaschenzug5.txt	2000	18	14281469418584906228	4,2s

#### 3.2 EIGENE BEISPIELE

Datei	$N$	$ K $	Verteilungsmöglichkeiten	Zeit
eigen1.txt	1600	10	2047548734	0s
eigen2.txt	2000	10	0	0s
eigen3.txt	4500	30	7313278250444698418	0s
eigen4.txt	2000	30	14571855106722217088	7s

1 ▶ data ▶ eigen1.txt :

```
1600
10
290 280 50 160 130 230 20 110 300 70
```

1 ▶ data ▶ eigen2.txt :

```
2000
10
290 280 50 160 130 230 20 110 300 70
```

*Kommentar: Kapazität der Behälter nicht ausreichend für 2000 Flaschen.*

1 ▶ data ▶ eigen3.txt :

```
4500
30
240 290 70 120 10 290 40 10 170 260 300 90 90 40 20 220 10 130 100 300 10 260 190
70 300 220 100 80 290 280
```

1 ▶ data ▶ eigen4.txt :

```
2000
30
```

```
240 290 70 120 10 290 40 10 170 260 300 90 90 40 20 220 10 130 100 300 10 260 190
70 300 220 100 80 290 280
```

### 3.3 BEISPIELHAFTE AUSGABE

```
Administrator: Eingabeaufforderung - bin\FlaschenzugWinx64.exe -f data\flaschenzug4.txt
C:\Users\Tim Hollmann\Desktop\Aufgabe 3 Final>bin\FlaschenzugWinx64.exe -f data\flaschenzug4.txt

+=====+
| Flaschenzug - Aufgabe 3 |
+=====+
| Tim Hollmann <0003>    |
| @ 34.Bundeswettbewerb Informatik 2015/'16 |
+=====+

-----
Übersicht:
Flaschen:                3000

Behälter:
[1]      :5
[2]      :100
[3]      :100
[4]      :200
[5]      :500
[6]      :500
[7]      :600
[8]      :800
[9]      :800
[10]     :1000
[11]     :1000

Berechne Kombinationen. Bitte warten... fertig.
Kombinationen: 17361307039603189181
Drücken Sie [ENTER] zum Beenden ...
```

Abbildung 1: Invertiertes Konsolenfenster der Bearbeitung des Beispiels flaschenzug4.txt

## 4 QUELLTEXT

```
7 typedef unsigned int uint, behaelter;
8 typedef unsigned long long int bigInt;
9
10 struct Application
11 {
12
13     uint _N;
14     std::vector<behaelter> _behaelter;
15     std::string _filename;
16
17     std::map<std::tuple<std::vector<behaelter>, uint>, bigInt> memo;
18 }
```



```

19     auto main(const std::vector<std::string>& arguments) -> int;
20
21     auto loadFromFile(void) -> bool;
22     auto getFromUser(void) -> void;
23
24     auto anzahl(const std::vector<behaelter>& z, uint z_summe, uint n) ↔
        -> bigInt;
25
26 };

```

Listing 1: 3&gt;inc&gt;Application.hpp - Application-Headerdatei

```

1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <string>
5  #include <cstdlib>
6  #include <fstream>
7  #include <sstream>
8
9  #include "../inc/Application.hpp"
10
11 auto Application::main(const std::vector<std::string>& arguments) -> int
12 {
13
14     std::cout << "\n +=====+";
15     std::cout << "\n | Flaschenzug - Aufgabe 3 |";
16     std::cout << "\n +-----+";
17     std::cout << "\n | Tim Hollmann (0003) |";
18     std::cout << "\n | @ 34.Bundeswettbewerb Informatik 2015/'16 |";
19     std::cout << "\n +=====+\n\n";
20
21     // Kommandozeilenargumente auswerten
22     for (int i = 0; i < arguments.size(); ++i)
23     {
24
25         if (arguments[i] == "-h" || arguments[i] == "--help" || ↔
            arguments[i] == "-?")
26         {
27             // Hilfe-Ausgabe
28             std::cout << "Hilfe-Ausgabe: Kommandozeilenargumente\n";
29             std::cout << "\n-f --file <Dateiname>\tLiest die Datendatei ↔
                ein. Format von bundeswettbewerb-informatik.de";
30             return EXIT_SUCCESS;
31         }
32         else if (arguments[i] == "-f" || arguments[i] == "--file")
33         {
34             // Dateiname übergeben
35             if (i + 1 <= arguments.size() - 1)
36             {
37                 _filename = arguments[i + 1];
38                 i++;
39             }

```

```

40     else { // Dateiname nicht in übergeben
41         std::cout << "Fehler - Trotz Kommandozeilenparameter ↵
42             Dateiname nicht übergeben. Ignoriere. ";
43     }
44     else {
45         // Unbekanntes Kommandozeilenargument
46         std::cout << "Warnung - Unbekannter Kommandozeilenparameter ↵
47             '" << arguments[i] << "'. Ignoriere. ";
48     }
49 }
50
51 // Dateiname übergeben?
52 if (_filename.size() == 0)
53 {
54     // Keine Datei übergeben
55     std::cout << " \nFehler - Keine Datendatei übergeben. Gehe zu ↵
56         manueller Dateneingabe über.";
57     getFromUser();
58 }
59 else if (!loadFromFile())
60 {
61     // Fehler beim Laden / lesen aus der Datei
62     std::cout << "\nFehler beim Öffnen/Auslesen der Datendatei. Gehe ↵
63         zu manueller Dateneingabe über.";
64     getFromUser();
65 }
66
67 std::cout << ↵
68     "\n\n-----\nÜbersicht: ↵
69     \nFlaschen: \t\t" << _N << "\n\nBehälter:";
70
71 for (int i = 0; i < _behaelter.size(); i++) {
72     std::cout << "\n[" << i + 1 << "]\t:" << _behaelter[i];
73 }
74
75 std::cout << "\n\nBerechne Kombinationen. Bitte warten... ";
76
77 int summe_kapazitaeten = 0;
78 for (int i = 0; i < _behaelter.size(); i++) { summe_kapazitaeten += ↵
79     _behaelter[i]; }
80
81 bigInt s = anzahl(_behaelter, summe_kapazitaeten, _N);
82
83 std::cout << "fertig.\n\nKombinationen: " << s << "\n\nDrücken Sie ↵
84     [ENTER] zum Beenden ... ";
85
86 std::cin.get();
87
88 return EXIT_SUCCESS;
89 }

```

```
85
86 auto Application::loadFromFile(void) -> bool
87 {
88
89     std::fstream datei(_filename.c_str(), std::ios::in);
90
91     if (!datei.good()) {
92         std::cout << "\nFehler beim Öffnen der Datei " << _filename << "\n"
93             << "\nStellen Sie sicher, dass diese am angegebenen Speicherort "
94             << "existiert.";
95         return false;
96     }
97
98     std::vector<std::string> file;
99
100     std::string zeile;
101     while (std::getline(datei, zeile, '\n'))
102         file.push_back(zeile);
103
104     datei.close();
105
106     try {
107         int n = std::stoi(file[0]);
108
109         int k = std::stoi(file[1]);
110
111         std::string behaelter_string = file[2];
112
113         std::stringstream data(behaelter_string);
114
115         std::vector<int> b;
116
117         std::string line;
118         while (std::getline(data, line, ' ')) {
119             b.push_back(std::stoi(line));
120         }
121
122         if (b.size() != k) {
123             std::cout << "Fehler - Die Anzahl der Behälter entspricht "
124                 << "nicht der angegebenen Anzahl in Zeile 2. Ignoriere Zeile 2 "
125                 << "(" << k << ") und benutze " << b.size() << " Behälter.";
126         }
127
128         _N = n;
129
130         for (int i = 0; i < b.size(); i++) {
131             _behaelter.push_back(b[i]);
132         }
133     }
134     catch (...){
135         std::cout << "\nWarnung - Fehler beim Einlesen der Datendatei "
136             << "aufgetreten. Stellen Sie sicher, dass deren Syntax der von "
```

```
        www.bundeswettbewerb-informatik.de entspricht. Gehe zu ↵
        manueller Dateneingabe über.";
133     return false;
134 }
135
136     return true;
137 }
138
139 auto Application::getFromUser(void) -> void
140 {
141
142     // Abfrage des Benutzers
143     std::cout << "\n\n===== \nManuelle ↵
        Eingabe:\n\n";
144
145     int n = 0;
146     do {
147         std::cout << "Flaschenanzahl N:";
148         std::cin >> n;
149     } while (n <= 0);
150
151     _N = n;
152
153     std::cout << "\n";
154
155     int k = 0;
156     do {
157         std::cout << "Anzahl der Behälter K:";
158         std::cin >> k;
159     } while (k <= 0);
160
161     std::cout << "\n\nEingabe der Behälterkapazitäten:";
162
163     for (int i = 1; i <= k; ++i)
164     {
165         std::cout << "\n";
166         int t = 0;
167         do {
168             std::cout << "Behälter " << i << " [ von " << k << " ]\t:";
169             std::cin >> t;
170         } while (t <= 0);
171         _behaelter.push_back(t);
172     }
173
174 }
175
176 auto Application::anzahl(const std::vector<behaelter>& z, uint ↵
        z_summe, uint n) -> bigInt
177 {
178
179     // Suche im Memo
180     std::tuple<std::vector<behaelter>, int> t(z, n);
181
```



```
182     if (memo.find(t) != memo.end())
183         return memo.at(t);
184
185     // Nur ein Behälter
186     if (z.size() <= 1)
187     {
188         int i = ((z[0] >= n) ? 1 : 0);
189         memo[t] = i;
190         return i;
191     }
192
193     // Bei mehreren Behältern den ersten herausnehmen (und den rest in ↔
194         temporärem Vektor speichern)
195     std::vector<behaelter> temp_b;
196     for (int i = 1; i < z.size(); i++) { temp_b.push_back(z[i]); }
197
198     // Neue Kapazitätssumme
199     int temp_zustand_summe = z_summe - z[0];
200
201     bigint s = 0;
202     // Ersten Behälter durchiterieren und rekursiv für übrige Behälter ↔
203         aufrufen
204     for (int i = 0; i <= z[0] && n - i >= 0; i++) {
205         if (temp_zustand_summe < n - i) continue;
206         s += anzahl(temp_b, temp_zustand_summe, n - i);
207     }
208
209     memo[t] = s; // Ergebnis speichern
210     return s;
211 }
```

Listing 2: 3▸src▸Application.cpp - Application-Hauptprogramm

## ABBILDUNGSVERZEICHNIS

- 1     Invertiertes Konsolenfenster der Bearbeitung des Beispiels `flaschenzug4.txt`   8

## LISTINGS

- 1      3 ▶ `inc ▶ Application.hpp` - Application-Headerdatei . . . . . 8  
2      3 ▶ `src ▶ Application.cpp` - Application-Hauptprogramm . . . . . 9