Junioraufgabe 2

33.Bundeswettbewerb Informatik 2014/'15

Der Script-Tim

Inhaltsverzeichnis

1 Problem

Das Problem besteht darin, beliebige Zahlen p,q und x zu ermitteln, für die gilt:

$$x, p, q > 1 \tag{1}$$

$$p \neq q \tag{2}$$

$$10 < (p + q) \leq 20 \tag{3}$$

$$\log_{10}(p * x) + \log_{10}(q * x) = 5 \tag{4}$$

3 und 4 können je nach Schwierigkeitsstufe abweichen.

2 Lösungsidee

Von Anfang an war mir klar, dass $\frac{p}{q}$ und $\frac{a}{b}$ durch einen Faktor - ich nenne ihn im Folgenden x - zusammenhängen;
$\frac{p}{q} \xrightarrow[*x]{*x} \frac{a}{b}$
Somit ist $a \equiv p * x$ und $b \equiv q * x$. Folglich reduziert sich die Menge an Variablen auf p, q und x.

Ich könnte es mir natürlich einfach machen und schreiben

for $AlzahlDerBrueche$ **do**
  $bruchOK =$ **false**
  repeat
    $p \leftarrow rand()$ mod $100 + 1$
    $q \leftarrow rand()$ mod $100 + 1$
    $x \leftarrow rand()$ mod $100 + 1$
    if $p > 1$ **and** $q > 1$ **and** $p \neq q$ **and** $\log_{10} p * x + \log_{10} q * x == 5$ **and** $untergr >$
    $p + q \leq obergr$ **then**
      $bruchOK =$ **true**
    end if
  until $bruchOK ==$ **true**
  [...]//Ausgabe des Bruches
end for
return 0

Dieser Lösungsweg wäre zwar korrekt, aber sehr ineffizient und Brain-AFK (dieses Wort wollte ich unbedingt irgendwo unterbringen). Im Folgenden werde ich versuchen, die Aufgabe etwas intelligenter zu lösen.

2.1 Schwierigkeit Leicht

2.1.1 p erzeugen

Zunächst wird p zufällig bestimmt

$$p \in \{\mathbb{N} \mid < 10\}$$

2.1.2 q erzeugen

Für q bleibt dann der Bereich zwischen 1 und $10 - p$ (Da $p < 10$ kann p maximal 9 und q somit minimal 1 sein).

$$q \in \{\mathbb{N} \mid \leq 10 - p\}$$

2.1.3 Passenden Faktor *x* ermitteln

Wir haben nun zwei Zahlen p und q. Für beide muss gelten $*x < 100$. Wir ermitteln zunächst, wie groß ein Faktor x maximal sein darf. Dafür müssen wir wissen, ob $p < q$ oder $p > q$. Dann teilen wir 100 durch den größeren Wert.

$$p > q \rightarrow \mathrm{x_{max}} = \lfloor \frac{100}{p} \rfloor$$

$$p < q \rightarrow \mathrm{x_{max}} = \lfloor \frac{100}{q} \rfloor$$

Folglich gilt dann auch, dass der kleinere Wert mal $\mathrm{x_{max}} < 100$ ist.

$$p > q \rightarrow q * \mathrm{x_{max}} < 100$$

$$p < q \rightarrow p * \mathrm{x_{max}} < 100$$

Da der größere Wert nie über 50 kommt, ist $\mathrm{x_{max}}$ immer ≥ 2. Wir kommen nun x völlig gefahrlos definieren als

$$x \in \{\mathbb{N} \mid \in [2 \ldots \mathrm{x_{max}}]\}$$

Nun haben wir p, q und x. Da $a \equiv p * x$ und $b \equiv q * x$ gilt, können wir a und b nun errechnen.

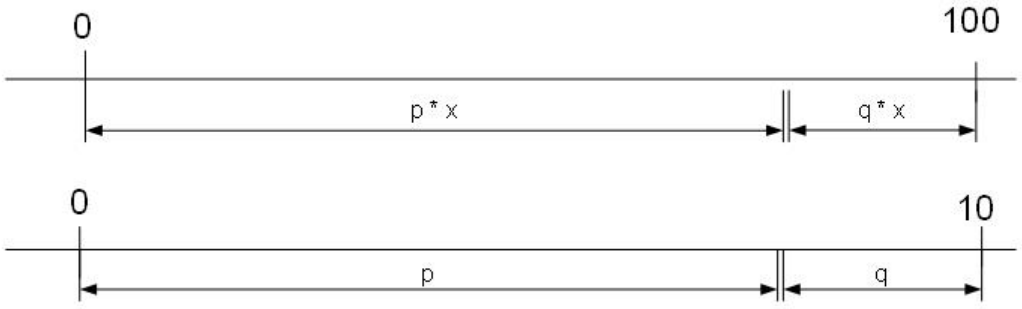

Abbildung 1: Leicht

2.2 Schwierigkeit Mittel + Schwer

Bei den Schwierigkeiten Mittel und Schwer soll die „Länge“ des Bruches 5 sein, d.h. ein Wert muss < 100 und der andere > 100 sein. Da Mittel und Schwer sich nur durch die p+q-Summen-Begrenzungen unterscheiden, definiere ich die Formeln allgemein und verwende die schwierigkeitsspezifischen Variablen *obergrenze* und *untergrenze*.

2.2.1 *p* erzeugen

Ebenfalls wird p zufällig erzeugt:

$$p \in \{\mathbb{N} \mid < obergrenze\}$$

2.2.2 *q* erzeugen

q wird dann ebenfalls so erzeugt, dass $p + q$ maximal *obergrenze*:

$$q \in \{\mathbb{N} \mid \leq obergrenze - p\}$$

2.2.3 Passenden Faktor *x* ermitteln

Wir müssen nun einen Faktor ermitteln, der multipliziert mit p und q einen Wert > 100 und einen Wert < 100 ergibt. Dafür teile ich 100 durch den größeren Wert und runde auf:

$$p > q \rightarrow \mathrm{x}_{\min} = \lceil \frac{100}{p} \rceil$$

$$q > p \rightarrow \mathrm{x}_{\min} = \lceil \frac{100}{q} \rceil$$

Durch das Aufrunden ist sichergestellt, dass $groessererWert * \mathrm{x}_{\min} > 100$. Theoretisch könnte $groessererWert * \mathrm{x}_{\min} = 100$, dies ist aber so unwahrscheinlich, dass die dadurch hervorgerufene erneute Ausführung des Algorithmus nicht als ineffizient gelten kann.

Wir haben nun einen minimalen Faktor $\mathrm{x}_{\min}$ ermittelt. Der maximale Faktor $\mathrm{x}_{\max}$ ist der kleinste Wert , für den gilt, dass $kleinererWert * (\mathrm{x}_{\max} + 1)$ größer gleich 100 ist. Deshalb gilt:

$$p > q \rightarrow \mathrm{x}_{\max} = \lfloor \frac{100}{q} \rfloor$$

$$q > p \rightarrow \mathrm{x}_{\max} = \lfloor \frac{100}{p} \rfloor$$

Der Faktor x ist dann

$$x \in \{\mathbb{N} \mid \in [\mathrm{x}_{\min} \ldots \mathrm{x}_{\max}]\}$$

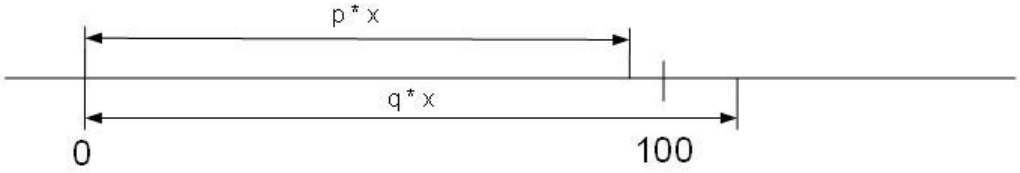

Abbildung 2: Beispielhaft hier: Schwer

3 Umsetzung in ein Programm

3.1 Wahl der Programmiersprache

Für die Umsetzung der Lösungsidee in ein Programm habe ich die Programmiersprache C++ gewählt.

3.2 Programmablauf

3.2.1 Dateneingabe

Bevor überhaupt irgend ein Bruch generiert werden kann, müssen dem Programm zuerst folgende Daten bekannt sein:

1. **Anzahl der Brüche**: Wie viele Brüche überhaupt erzeugt werden sollen und
2. **Schwierigkeitsstufe**: Schwierigkeitsstufe im Sinne der Angabe.

Um diese Daten dem Programm mitzuteilen, habe ich verschiedene Möglichkeiten eingeräumt:

- **Übergeben von Parameter in der Kommandozeile**; da es sich hierbei um eine Kommandozeilenanwendung handelt, liegt die Möglichkeit nahe, die benötigten Daten über Kommandozeilenparameter zu übergeben. Hierzu verwenden Sie

  ```
  > Junioraufgabe2.exe <Anzahl> <Schwierigkeit>
  ```

  Für 'Anzahl' können Sie jede beliebige Zahl über Null ($\in \mathbb{N}$) einsetzen; für 'Schwierigkeit' sind möglich 'leicht','mittel' und 'schwer'.

- **Eingaben der Daten im Programm**; sind keine Kommandozeilenparameter übergeben oder sind diese formal unkorrekt (z.B. nicht numerisch usw.), werden die Daten direkt im Programm durch den Benutzer eingegeben.(Fallback)

```
62    int anzahlBrueche = 0;
63    int schwierigkeit = 0;
64
65    //Datenquelle: Befehlszeilenargumente übergeben?
```

```
66      bool kommandozeilenparameterOK = false;
67
68      if (argc == 3){
69          stringstream sstream;
70          sstream.clear();
71          sstream << argv[1];
72          sstream >> anzahlBrueche;
73
74          if (anzahlBrueche <= 0){
75              cout << "Fehler: Anzahl der Br[ue]che '" << anzahlBrueche << "' entspricht
                    nicht den Anforderungen.\nGehe [ue]ber zu manueller Dateneingabe...";
76          }else{
77              sstream.clear();
78              sstream << argv[2];
79              string schwierigkeitsstufe;
80              sstream >> schwierigkeitsstufe;
81
82              schwierigkeit = 0;
83
84              if (schwierigkeitsstufe == "leicht" || schwierigkeitsstufe == "Leicht" ||
                    schwierigkeitsstufe == "1"){
85                  schwierigkeit = 1;
86              }
87
88              if (schwierigkeitsstufe == "mittel" || schwierigkeitsstufe == "Mittel" ||
                    schwierigkeitsstufe == "2"){
89                  schwierigkeit = 2;
90              }
91
92              if (schwierigkeitsstufe == "schwer" || schwierigkeitsstufe == "Schwer" ||
                    schwierigkeitsstufe == "3"){
93                  schwierigkeit = 3;
94              }
95
96              if (schwierigkeit == 0){
97                  cout << "Fehler: Schwierigkeitsstufe '" << schwierigkeitsstufe << "'
                        nicht bekannt. Verwenden Sie 'leicht', 'mittel' oder 'schwer'";
98                  cout << "\nGehe zu manueller Dateneingabe [ue]ber...";
99              }else{
100                 kommandozeilenparameterOK = true;
101             }
102         }
103
104     }
105
106     if (!kommandozeilenparameterOK){
107         //Datenquelle: Benutzer
108         //Anzahl der zu erzeugenden Brüche
109         cout << "\nAnzahl der zu erzeugenden Br[ue]che:";
110
111         do{
112             cin >> anzahlBrueche;
```

```
113         }while(anzahlBrueche < 1);
114
115         //Schwierigkeitsgrad
116         cout << "\n Schwierigkeit: \n";
117         cout << " +--------------------------------------+\n";
118         cout << " | [1] Leicht                           |\n";
119         cout << " | [2] Mittel                           |\n";
120         cout << " | [3] Schwer                           |\n";
121         cout << " +--------------------------------------+\n";
122
123         do{
124             cout << "\n:";
125             cin >> schwierigkeit;
126         }while(schwierigkeit != 1 && schwierigkeit != 2 && schwierigkeit != 3);
127     }else{
128         cout << "\n============================================";
129         cout << "\n Kommandozeilenparameter erhalten:";
130         cout << "\n Anzahl der Br[ue]che: " << anzahlBrueche;
131         cout << "\n Schwierigkeit: " << schwierigkeit << " ";
132         switch(schwierigkeit){
133             case 1:
134                 cout << "(leicht)";
135                 break;
136             case 2:
137                 cout << "(mittel)";
138                 break;
139             case 3:
140                 cout << "(schwer)";
141                 break;
142             default:
143                 cout << "\n\nUNBEKANNTER FEHLER AUFGETRETEN.BEENDE";
144                 return 0;
145         }
146
147         cout << "\n============================================";
148     }
```

Nachdem die Daten übergeben wurden, wird je nach Schwierigkeitsstufe ein od. mehrere Brüche erzeugt:

3.2.2 Schwierigkeit Leicht

```
159         for(unsigned int y = 1; y <= anzahlBrueche; y++){
160             p = 0; q = 0; x = 0;
161             do{
162                 //P erzeugen
163                 p = (rand() % 9) +1;
164
165                 //Q erzeugen
166                 do{
167                     q = (rand() % (10-p)) +1;
168                 }while(p + q > 10 || p == q);
```

```
169
170
171                 //X erzeugen
172                 if (p > q){
173                     x = (rand() % (((int)(100/p))-1))+2;
174                 }else{
175                     x = (rand() % (((int)(100/q))-1))+2;
176                 }
177             }while(!(q > 0 && p > 0 && p != q && p+q <= 10 && x >= 2 && ((int)(log10((
                    double)(p*x))) +1) + ((int)(log10((double)(q*x))) +1) == 4));
178             ausgeben(y, p, q, x, 10, 0, 4);
179         }
```

Hierbei wird genau nach **2.1** vorgegangen:

$$p \in \{\mathbb{N} \mid < 10\}$$

```
162                 //P erzeugen
163                 p = (rand() % 9) +1;
```

$$q \in \{\mathbb{N} \mid \leq 10 - p\}$$

```
165                 //Q erzeugen
166                 do{
167                     q = (rand() % (10-p)) +1;
168                 }while(p + q > 10 || p == q);
```

Hierbei habe ich gleich eine Schleife eingebaut, die dafür sorgt, dass gilt $p \neq q$ und die Obergrenze von 10 eingehalten wird. Die Umsetzung der Zuweisung von x sieht so aus:

$$p > q \rightarrow \mathrm{x_{min}} = \lceil \frac{100}{p} \rceil$$

$$q > p \rightarrow \mathrm{x_{min}} = \lceil \frac{100}{q} \rceil$$

```
171                 //X erzeugen
172                 if (p > q){
173                     x = (rand() % (((int)(100/p))-1))+2;
174                 }else{
175                     x = (rand() % (((int)(100/q))-1))+2;
176                 }
```

Sind nun die Werte p,q und x ermittelt, erfolgt sicherheitshalber noch eine 'Endabnahme'; die Werte werden noch einmal daraufhin überprüft, die notwendigen Bedingungen (vgl.1) zu erfüllen:

```
}while(
    !(
        q > 0
        && p > 0
        && p != q
        && p+q <= 10
```

```
        && x >= 2
        && ((int)(log10((double)(p*x))) +1)
          + ((int)(log10((double)(q*x))) +1) == 4
    )
);
```

Zum Ermitteln der Bruch-Länge verwende ich hier die Logarithmusfunktion zur Basis 10

$$\log_{10}(Zahl) + 1 = \text{Anzahl der Stellen von } Zahl$$

Sind nicht alle Bedingungen erfüllt (was sie aber in fast allen Fällen sein sollten), wird der Algorithmus wiederholt. Werden alle Bedingungen erfüllt, wird der Bruch durch die (übrigens einzige Funktion neben der *main*-Funktion bei dieser Aufgabe) Funktion *ausgeben* in einer improvisierten Tabelle ausgegeben. Parallel zur Ausgabe wird die p-q-Summe und die Bruchlänge ermittelt und dem Benutzer zur manuellen Überprüfung ausgegeben.

```
178             ausgeben(y, p, q, x, 10, 0, 4);
```

```
14  void ausgeben(int y, int p, int q, int x, int obergr, int untergr, int bruchLaenge){
15  cout << "\n";
16  //Nummer
17  cout << "| ";
18  if ((int)(log10((double)(y))) +1 == 4){ cout << ""; }
19  if ((int)(log10((double)(y))) +1 == 3){ cout << "0"; }
20  if ((int)(log10((double)(y))) +1 == 2){ cout << "00"; }
21  if ((int)(log10((double)(y))) +1 == 1){ cout << "000"; }
22  cout << y;
23  //a/b
24  cout << " | ";
25  if ((((int)(log10((double)(p*x))))+1) + (((int)(log10((double)(q*x))))+1) == 3){ cout
        << " "; }
26  if ((((int)(log10((double)(p*x))))+1) + (((int)(log10((double)(q*x))))+1) == 4){ cout
        << " "; }
27  cout << p*x;
28  cout << "/";
29  cout << q*x;
30  //Faktor x
31  cout << " | ";
32  if (x < 10){ cout << " "; }
33  cout << x;
34  //p-q-Summe
35  cout << "    | ";
36  if ((((int)(log10((double)(p))))+1) + (((int)(log10((double)(q))))+1) == 2){ cout <<
        " "; }
37  if ((((int)(log10((double)(p))))+1) + (((int)(log10((double)(q))))+1) == 3){ cout <<
        " "; }
38  cout << p << "/" << q;
39  //Check: p-q-Summe
40  cout << " | ";
41  cout << ((p+q > untergr && p+q <= obergr) ? "OK [" : "FAIL [");
42  if (p+q < 10 ){ cout << "0"; }
```

```
43 cout << p+q;
44 //Check: BruchLänge
45 cout << "] | ";
46 int laenge = (((int)(log10((double)(p*x))))+1) + (((int)(log10((double)(q*x))))+1);
47 cout << ((laenge == bruchLaenge) ? "OK [" : "FAIL [");
48 cout << laenge;
49 cout << "]  |";
50 }
```

Diese Schritte werden für jeden zu erzeugenden Bruch wiederholt.

3.2.3 Schwierigkeiten Mittel + Schwer

Da sich Schwierigkeit Mittel und Schwer nur durch die Begrenzungen für die p-q-Summe unterscheiden, verwende ich auch für beide den selben Code und repräsentiere die Untergrenze und Obergrenze der p-q-Summme durch die Variablen *untergr* und *obergr*.

```
181         obergr = 0; untergr = 0;
182         if (schwierigkeit == 2){//Mittel
183             untergr = 10;
184             obergr = 20;
185         }else{ //Schwer
186             obergr = 30;
187             untergr = 20;
188         }
189 
190         for(unsigned int y = 1; y <= anzahlBrueche; y++){
191             p = 0; q = 0; x = 0;
192             do{
193                 //P erzeugen
194                 p = (rand() % (obergr - 1)) +1;
195 
196                 //Q erzeugen
197                 do{
198                     q = (rand() % (obergr - p)) +1;
199                 }while(p == q || p + q > obergr || p+q <= untergr);
200 
201                 //X ermitteln
202                 if (p > q){
203                     int differenz = ((int)(100/q)) - (((int)(100/q)) +1);
204                     x = (((int)(100/p))+1) + (rand() % differenz) +1;
205                 }else{
206                     int differenz = ((int)(100/q)) - (((int)(100/q)) +1);
207                     x = (((int)(100/p))+1) + (rand() % differenz) +1;
208                 }
209 
210             }while(!(q > 0 && p > 0 && p != q && p+q <= obergr && p+q > untergr && x
                    >= 2 && (((int)(log10((double)(p*x)))) +1) + (( (int) (log10((double)(
                    q*x)))) +1) == 5));
211             ausgeben(y, p, q, x, obergr, untergr, 5);
212         }
```

Der Code ist beinahe selbsterklärend und deckt sich auch teilweise mit dem aus **3.2.2**;

1. Setzen von *untergr* und *obergr* je nach Schwierigkeitsstufe [Z.181 - 188]

2. Erzeugung von p nach 2.2.1 [Z.194]

3. Erzeugung von q nach 2.2.2 [Z.197 - 199]; hierbei verwende ich wieder eine Schleife, um $p \neq q$ und die Summenbeschränkungen einzuhalten.

4. Erzeugung von x nach 2.2.3 [Z.202 - 208]

Auch hier wieder die „Endabnahme“ und die darauffolgende Ausgabe des Bruches über *ausgeben* [Z.210f.].

3.3 Beispiele

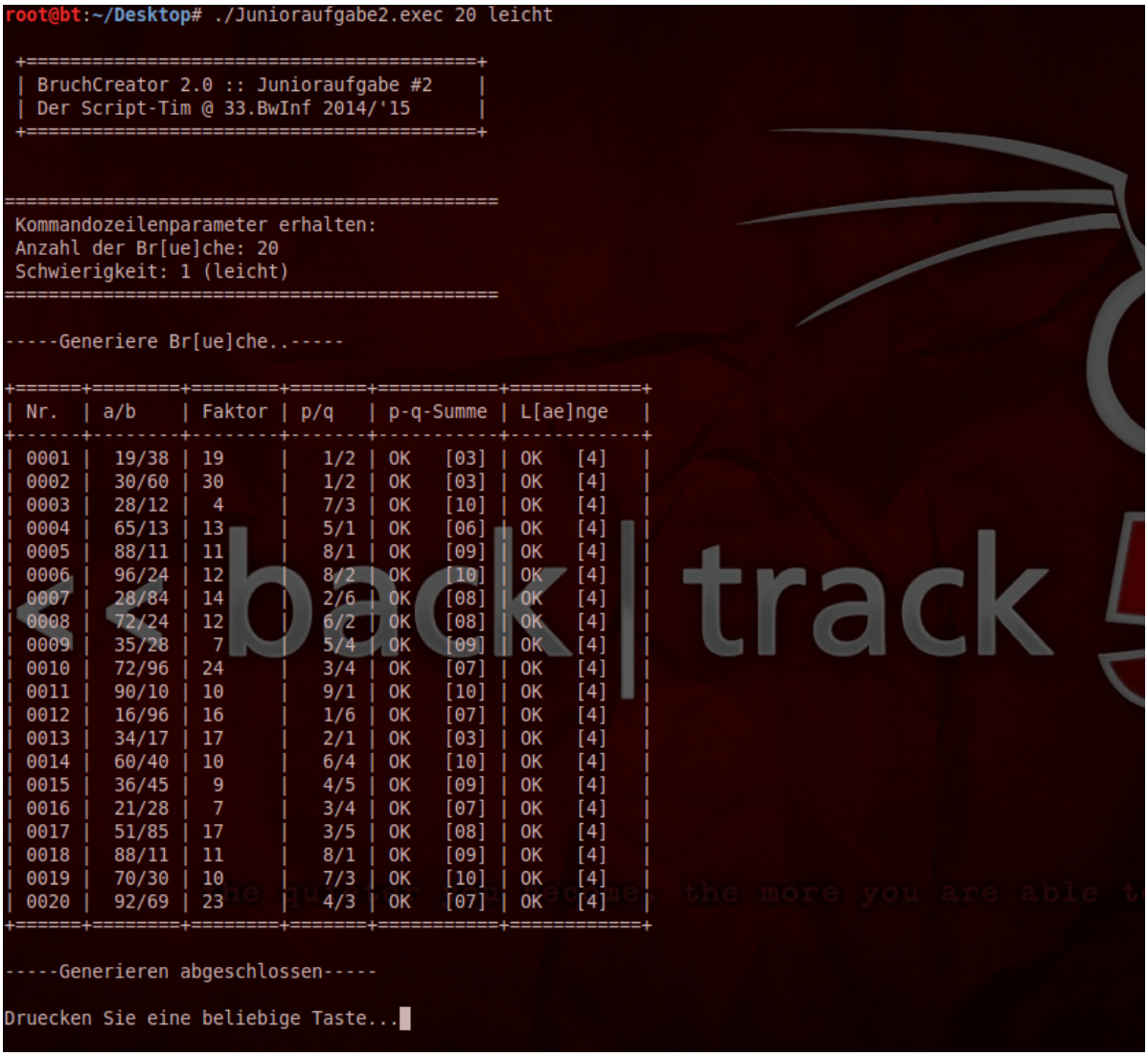

Abbildung 3: Ein Beispiel für 20 leichte Brüche

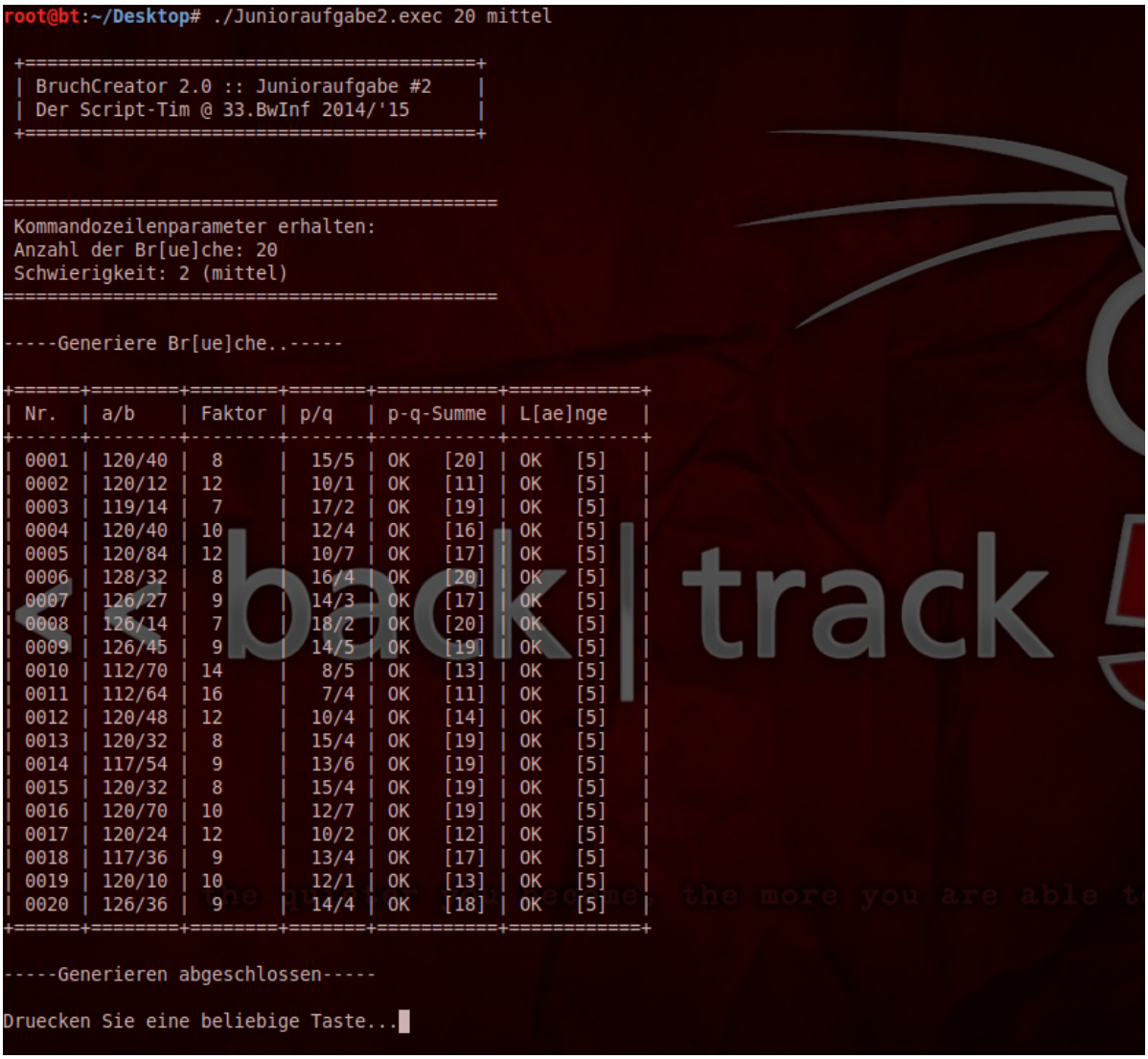

Abbildung 4: Ein Beispiel für 20 mittlere Brüche

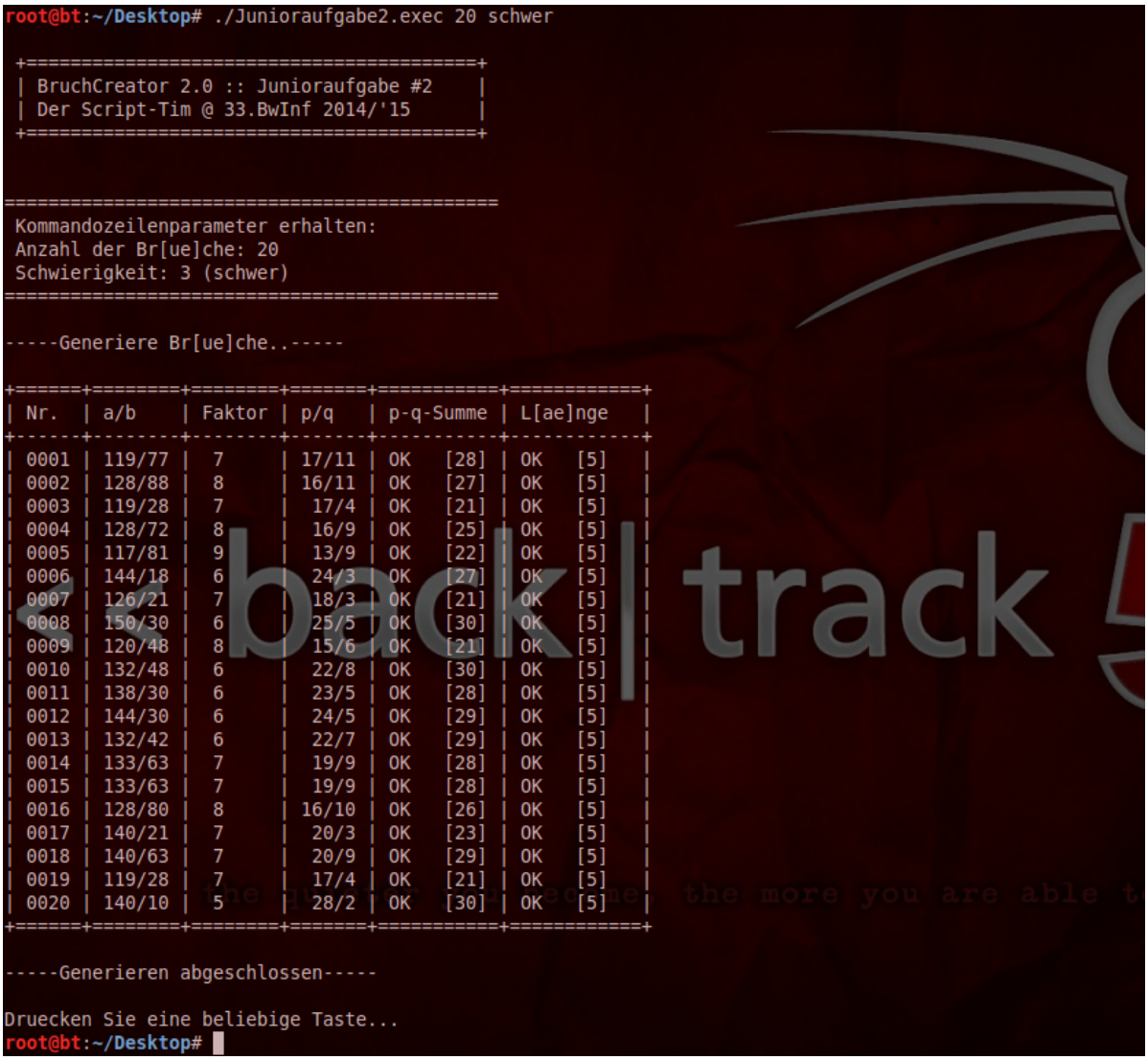

Abbildung 5: Ein Beispiel für 20 schwere Brüche