Aufgabe 3: Telepaartie

Team-ID: 00587

Team-Name: Doge.NET

Bearbeiter dieser Aufgabe:
Johannes von Stoephasius

23. November 2019

Inhaltsverzeichnis

1 Lösungsidee

1.1 Definitionen

Zustand Ein Zustand ist definiert als Menge von Behältern, wobei jedem Behälter eine nichtnegative ganze Zahl zugeordnet werden kann, die der Anzahl an Bibern im Gefäß entspricht.

Weiter können die Behälter untereinander getauscht werden, da die Konstellation die selbe bleibt. Deshalb werden die Biber-Anzahlen eines Zustands immer nur im sortierten Zustand betrachtet, wobei hier aufsteigende Sortierung verwendet wird.

Man bezeichne die Menge aller Zustände mit Gesamtbiberzahl n als $\mathbb{S}_n$.

Endzustand Ein Endzustand ist jeder Zustand, der genau einen leeren Eimer enthält.

Sind weniger, also keine, leere Eimer enthalten, so ist der Zustand kein Endzustand laut der Aufgabenstellung.

Sind mehr enthalten, so ist der Zustand nur durch Operationen auf einen anderen Endzustand zu erhalten, und somit nicht relevant. Die Ermittlung aller Endzustände wird in Abschnitt 1.4 erläutert.

Man bezeichne die Menge aller Endzustände mit Gesamtbiberzahl n als $\mathbb{E}_n$. Dabei gilt: $\mathbb{E}_n \subseteq \mathbb{S}_n$

Ursprungszustand Ein Ursprungszustand von einem Zustand x, ist jeder Zustand der mit einem einzelnen Telepaartieschritt zum Zustand x wird. Die Ermittlung aller Ursprungszustände eines Zustands wird in Abschnitt 1.3 erläutert.

Die Menge an Ursprungszuständen von x kann geschrieben werden als $origin(x)$, mit $origin : \mathbb{S}_n \to \mathbb{S}_n$.

Generation Eine Generation ist eine Menge an unterschiedlichen Zuständen.

1.2 Kernidee

Die Grundidee der Lösung basiert nicht auf der Idee, alle nicht-Endzustände zu ermitteln und diese optimal zu lösen, sondern invers alle Endzustände zu ermitteln und diese invers auf alle ihre Ursprungszustände zurückzuführen, diese Ursprungszustände wieder auf ihre eigenen Ursprungszustände zurückzuführen usw., wobei konstant überprüft wird, ob es nicht "Abkürzungen" im Sinne bereits gefundener Zustände gibt.

1.3 Ermittlung aller Ursprungszustände

Zum ermitteln der Ursprungszustände $origin(s)$ eines Zustands $s \in \mathbb{S}_n$ wird jede Biber-Anzahl mit jeder anderen Biber-Anzahl verglichen. Ist dabei bei einem Vergleich zweier Anzahlen die erste Anzahl größer als 0 und durch 2 teilbar, dann ist es möglich, dass auf diese beiden Behälter eine Telepaartie angewendet wurde. Um diese umzukehren wird die Anzahl im ersten Behälter addiert und die Differenz zum 2. Behälter addiert. Diese Überprüfung wird für alle Kombinationen zweier Biber-Anzahlen durchgeführt, bis am Ende alle Ursprungszustände gefunden wurden.

1.3.1 Begründung

Seien $a_0, a_1, b_0, b_1 \in \mathbb{N}$ die zwei Biberanzahlen, wobei a_0 und b_0 die Anzahlen vor der Telepaartie repräsentieren, und a_1 und b_1 die danach. Sei weiterhin o.B.d.A. $a_0 < b_0$.

Laut der Definition der Telepaartie gilt:

$$a_1 = 2a_0$$
$$b_1 = b_0 - a_0$$

Hierraus lässt sich herleiten:

$$\begin{aligned}
& a_1 = 2a_0 \\
\iff\quad & a_0 = \frac{a_1}{2} \\
& b_1 = b_0 - a_0 \\
\iff\quad & b_1 = b_0 - \frac{a_1}{2} \\
\iff\quad & b_0 = b_1 + \frac{a_1}{2} \\
& a_0 < b_0 \\
\iff\quad & \frac{a_1}{2} < b_1 + \frac{a_1}{2} \\
\iff\quad & 0 < b_1
\end{aligned}$$

Wichtig hierbei ist:

$$0 < b_1$$
$$a_0 = \frac{a_1}{2}$$
$$b_0 = b_1 + \frac{a_1}{2}$$

1.4 Generieren der Endzustände

Zur effizienten Findung aller Endzustände werden nicht erst alle möglichen Endzustände mit Duplikaten generiert und am Ende die Duplikate entfernt, sondern gleich nur Zustände berechnet, die nicht wiederholt aufteten werden.

Zur Simplifizierung der Rechnung werden alle Zustände mit Behälterzahl minus eins berechnet, die keine leeren Behälter besitzen. Danach wird an jedes dieser einfach eine null angehängt.

Der Algoritmus funktioniert, indem zuerst für einen Behälter alle möglichen Biberanzahlen ermittelt werden, für die gilt:

- Es ist möglich die restlichen Biber so aufzuteilen, dass jeder Behälter genausoviele oder weniger Biber enthält wie der vorherigen
- Es ist garantiert, dass die restlichen Behälter alle nicht leer sein müssen

Um zu garantieren, dass die Behälter absteigend befüllbar sind, müssen mindestens $\lceil \frac{<Anzahl\ Biber>}{<Anzahl\ Behälter>} \rceil$ Biber in den ersten Becher.

Um die nicht-Leerheit zu garantieren, müssen maximal $< Anzahl\ Biber > -(< Anzahl\ Behälter > +1)$ Biber in den ersten Becher. Somit kann mindestens ein Biber in jeden restlichen Behälter platziert werden.

Für den trivialen Fall, das nur ein Behälter vorhanden ist, müssen alle Biber in diesen.

Nun lassen sich alle für den ersten Becher mögliche Biberanzahlen bestimmen, und für diese jeweils rekursiv alle folgenden Biberanzahlen. Hierbei ist noch zu beachten, dass noch garantiert werden muss, das die Behälter absteigend voll sind. Dies ist trivialerweise umsetzbar, indem alle Biberanzahlen die größer der Biberanzahlen eines vorherigen Behälters sind eliminiert werden.

1.5 Hauptalgorithmus

Sei die Generation Gen_{i+1} definiert als alle unterschiedliche Ursprungszustände aller Elemente aus Gen_i, die in keiner vorherigen Generation $Gen_k, k < i$ enthalten sind.

Sei dabei Gen_0 als Spezialfall gleich der Menge aller Endzustände für Gesamtbiberzahl n.

$$Gen_0 := \mathbb{E}_n$$
$$Gen_{i+1} := \left\{ s \mid \underbrace{(\exists_{t \in Gen_i} : s \in origin(t))}_{\text{Alle Ursprungszustände der vorherigen Generation}} \wedge \underbrace{(\forall_{i_0 \in \mathbb{N}, i_0 \leq i} : s \notin Gen_{i_0})}_{\text{Keine bereits in vorherigen Generationen enthaltene Zustände}} \right\}$$

Der Algoritmus funktioniert dann, indem er nach und nach alle nicht-leeren Generationen ermittelt. Sei Gen_m die letzte nicht-leere Generation, so ist $LLL(n) = m$.

1.6 Beweis

Es existiert eine letzte nicht-leere Generation Gen_m. Weiterdem gilt $LLL(n) = m$.

1.6.1 Hilfssatz 1: Erreichen aller lösbaren Zustände

Wähle beliebig aber fest einen Zustand $s \in \mathbb{S}_n$. Ist der Zustand lösbar, also durch wiederholte Telepaartie zu einem Endzustand überführbar, so gibt es eine Generation Gen_i aus $(Gen_i)_{i \in \mathbb{N}}$ mit $s \in Gen_i$.

Trivialer Fall Gilt $s \in \mathbb{E}_n$, so ist s lösbar mit 0 Telepaartieschritten. Da $Gen_0 = \mathbb{E}_n$ gilt, gilt $s \in Gen_0$.

Beweis durch Widerspruch Angenommen $s \notin Gen_i$. Für alle $i = 1, 2, ...$

$$\begin{aligned} s \notin Gen_i &\iff \neg \left(\left(\exists_{t \in Gen_{i-1}} : s \in origin(t) \right) \wedge (\forall_{i_0 \in \mathbb{N}, i_0 < i} : s \notin Gen_{i_0}) \right) \\ &\iff \neg \left(\exists_{t \in Gen_{i-1}} : s \in origin(t) \right) \vee \neg (\forall_{i_0 \in \mathbb{N}, i_0 < i} : s \notin Gen_{i_0}) \\ &\iff \left(\forall_{t \in Gen_{i-1}} : s \notin origin(t) \right) \vee (\exists_{i_0 \in \mathbb{N}, i_0 < i} : s \in Gen_{i_0}) \end{aligned}$$

Angenommen es gilt $\exists_{i_0 \in \mathbb{N}, i_0 < i} : s \in Gen_{i_0}$. Wenn dies gilt, existiert ein i_0, für welches gilt: $s \in Gen_{i_0}$. Somit existiert eine Generation aus $(Gen_i)_{i \in \mathbb{N}}$ mit $s \in Gen_{i_0}$.

Somit können wir das Problem durch Redefinition $i := i_0$ reformulieren als:

$$s \notin Gen_i \iff \forall_{t \in Gen_{i-1}} : s \notin origin(t)$$

Dies ist nun zu zeigen:

$$s \notin Gen_i \iff \forall_{t \in Gen_i} : s \notin origin(t)$$

Bemerkung: $origin(origin(t)) = \left\{ s \mid \exists_{u \in origin(t)} : s \in origin(u) \right\}$

$$\begin{aligned} &\iff \forall_{t \in Gen_{i-1}} : s \notin origin(origin(t)) \\ &\iff \forall_{t \in Gen_{i-1}} : s \notin (origin \circ origin)(t) \\ &\iff \forall_{t \in Gen_{i-i}} : s \notin (\underbrace{origin \circ ... \circ origin}_{\text{i-mal verkettet}})(t) \\ &\iff \forall_{t \in \mathbb{E}_n} : s \notin (\underbrace{origin \circ ... \circ origin}_{\text{i-mal verkettet}})(t) \end{aligned}$$

Damit dies gilt, müsste s für keine Anzahl i an Telepaartieschritten zu einem Endzustand kommen. Somit müsste s also unlösbar sein. □

1.6.2 Korollar aus Hilfssatz 1: Maximale Mindestschrittzahl

Wenn $s \in Gen_i$ gilt, dann ist s in i oder weniger Telepaartieschritten zu einem Endzustand überführbar.

Beweis Wie aus Abschnitt 1.6.1 hervorgeht, kann ein Zustand $s \in \mathbb{S}_n$ nur lösbar sein, bzw. eine Generation Gen_i mit $s \in Gen_i$ existieren, wenn gilt:

$$\begin{aligned} & \forall_{t \in Gen_i} : s \notin origin(t) \iff \forall_{t \in \mathbb{E}_n} : s \notin (\underbrace{origin \circ ... \circ origin}_{\text{i-mal verkettet}})(t) \\ \iff \quad & \forall_{t \in Gen_i} : s \in origin(t) \iff \exists_{t \in \mathbb{E}_n} : s \in (\underbrace{origin \circ ... \circ origin}_{\text{i-mal verkettet}})(t) \end{aligned}$$

Da laut Definition von *origin* die i-fache Selbstverkettung von *origin* alle Zustände sind, von denen aus der Parameter mit weniger als oder genau i Telepaartieschritten erreicht werden kann ist, ist der Endzustand $t \in \mathbb{E}_n$ von s in weniger als oder genau i Schritten erreichbar. □

1.6.3 Hilfssatz 2: Eindeutigkeit

Für jeden lösbaren Zustand $s \in \mathbb{S}_n$ gilt, dass *genau ein* i existiert, für dass die Generation Gen_i mit $s \in Gen_i$ existiert.

Beweis Ist der Zustand s lösbar, so existiert laut Abschnitt 1.6.1 ein i mit $s \in Gen_i$. Aufgrund der Kondition $\forall_{i_0 \in \mathbb{N}, i_0 < i} : s \notin Gen_{i_0}$ in der Definition von Gen_i gilt, dass keine Generation $Gen_j, j < i$ aus $(Gen_i)_{i \in \mathbb{N}}$ existiert, die s enthält. Andersherum gibt es auch keine späteren Generationen $Gen_k, k > i$ mit $s \in Gen_k$, da für diese dann ein $i_0 = i$ mit $s \in Gen_{i_0}$ existieren würde, was gegen die Definition von Gen_i verstößt. □

1.6.4 Hilfssatz 3: Minimalität der Schritte

Für jeden lösbaren Zustand $s \in \mathbb{S}_n$ mit $s \in Gen_i$ gilt, dass $i = LLL(s)$.

Beweis Der Fall $LLL(s) > i$ wird vom Korollar Abschnitt 1.6.2 wiederlegt.

Somit wäre nur noch zu zeigen das $LLL(s) < i$ nicht gilt.

Damit $LLL(s) < i$ gilt, müsste es eine Schrittfolge geben, um s mit $k < i$ Schritten in einen Endzustand zu überzuführen.

Die Generationen Gen_j mit $0 \leq j < i$ enthalten zusammen alle Elemente von allen j-fachen Selbstverkettungen von $origin$, also jeden Zustand der in genau $i - 1$ oder weniger Schritten zu einem Endzustand überführbar ist. Mit der Eindeutigkeit der Generationen (siehe Abschnitt 1.6.3) verbunden, ist also $LLL(s) < i$ und $s \in Gen_j$ äquivalent.

Ist nun $s \in Gen_i$, gilt laut Abschnitt 1.6.3, dass s in keiner anderen Generation aus $(Gen_i)_{i \in \mathbb{N}}$, also auch keiner Generation Gen_j enthalten ist. Da $LLL(s) < i \iff s \in Gen_j$ gilt, und da $s \notin Gen_j$ gilt, gilt auch $\neg(LLL(s) < i) \iff LLL(s) \geq i$.

Da $LLL(s) \geq i$ gilt, gilt $LLL(s) < i$ nicht. □

1.6.5 Hilfssatz 4: Garantie der Leerheit

Die Serie $(Gen_i)_{i \in \mathbb{N}}$ ist bis inklusive zu einem Index m in keinem Element leer. Nach diesem Index ist sie in jedem Element leer.

Beweis Angenommen $Gen_i =$.

$$\begin{aligned} Gen_{i+1} :&= \{s \mid (\exists_{t \in Gen_i} : s \in origin(t)) \wedge (\forall_{i_0 \in \mathbb{N}, i_0 \leq i} : s \notin Gen_{i_0})\} \\ &= \{s \mid \underbrace{(\exists_{t \in \{\}} : s \in origin(t))}_{\substack{\text{In der leeren Menge} \\ \text{existieren keine Elemente.} \\ \text{Erst Recht keine, die} \\ \text{die Kondition erfüllen}}} \wedge (\forall_{i_0 \in \mathbb{N}, i_0 \leq i} : s \notin Gen_{i_0})\} \\ &= \{\} \end{aligned}$$

Somit gilt $Gen_i = \implies Gen_{i+1} =$.

Wäre vor einem Index m eine Generation leer, müssten somit auch folgende Generationen leer sein. Somit wäre m redefinierbar als der Index wo das erste leere Element vorkommt.

Da nur eine endliche Menge an Zuständen $s \in \mathbb{S}_n$ existiert, und da alle lösbaren Zustände, welche Teilmenge aller Zustände sind, laut Abschnitt 1.6.3 eindeutig genau einer Generation angehören, ist auch die Menge an nicht-leeren Generationen endlich.

Da endlich viele nicht-leeren Generationen enthalten sein müssen, und da die Serie nicht zwischendrin leere Generationen enthalten kann, muss sie alle nicht-leeren Generationen bis zu einem Index m haben, und alle leeren ab diesem. □

1.6.6 Beweis

Laut Abschnitt 1.6.5 existiert eine letzte, nicht-leere Generation Gen_m.

Da die letzte nicht-leere Menge existiert, ist bekannt, dass alle nicht-leeren Generation einen Index kleiner oder gleich m haben.

Laut Abschnitt 1.6.4 gilt für alle $s \in Gen_i$: $LLL(s) = i$.

Da alle Generation mit mehr als null Zuständen Gen_i einen Index $i \leq m$ haben, ist die maximale LLL der maximale Index m.

Da die maximale LLL gleich dem Index m ist, ist $L(n) = m$.

2 Umsetzung

Zur Umsetzung haben wir den obigen Algorithmus in C# 8.0 mit .NET Core 3.0 implementiert.
Die Zustände werden in Form einer `public class` State gespeichert. Die `class` beinhaltet

1. `int Depthations` - Eine Property zur Errechnung der Generationsnummer des Zustands.

2. `State? Parent` - Der "Vater" des Zustands; also der Zustand, von der Zustand Ursprungszustand ist. Ist der Zustand ein Endzustand, so ist Parent `null`.

3. `int[] Buckets` - Die Biberanzahlen.

Die wichtigsten Methoden aus der `class` State sind `public IEnumerable<State> Origins()` und `private State ReverseTeelepartie(int first, int second)`, wobei `Origins()` alle Ursprungszustände des Zustands ermittelt, und `ReverseTeelepartie(int first, int second)` dabei intern einen neuen Zustand berechnet, der den Zustand vor der Telepaartie beschreibt.

Die allgemeine Berechnung erfolgt in der `public static class` Telepartie. Hier ist die wichtigste Methode `private static int LLLCore(int numberOfCups, int numberOfItems, State? goal, Action<string>? writeLine)`, die entweder für nur einen gegebenen Fall oder für eine Anzahl von Bibern die Anzahl von nötigen Operationen berechnet.

3 Beispiele

Im Folgenden wird das Programm immer mit Argumenten aufgerufen, um den Dialog mit dem CLI zu überspringen. Für mehr Informationen über die möglichen Parameter führen sie den Befehl `Telepaartie.CLI --help` aus.

Für die Verteilung 2, 4, 7 ist die Ausgabe:

```
./Telepaartie.CLI -l 2,4,7 -v
Starting iteration 2
FERTIG!
Man benötigt 2 Telepaartie-Schritte
Die Berechnung dauerte 0:00 Minuten.
```

Für die Verteilung 3, 5, 7 ist die Ausgabe:

```
./Telepaartie.CLI -l 3,5,7 -v
Starting iteration 3
FERTIG!
Man benötigt 3 Telepaartie-Schritte
Die Berechnung dauerte 0:00 Minuten.
```

Für die Verteilung 80, 64, 32 ist die Ausgabe:

```
./Telepaartie.CLI -l 80,64,32 -v
Starting iteration 2
FERTIG!
Man benötigt 2 Telepaartie-Schritte
Die Berechnung dauerte 0:00 Minuten.
```

```
./Telepaartie.CLI -c 3 -e 100 -v
Starting iteration 8

--------------

State (Depth:8) {31;32;37}
State (Depth:7) {5;31;64}
State (Depth:6) {10;31;59}
State (Depth:5) {10;28;62}
State (Depth:4) {20;28;52}
State (Depth:3) {8;40;52}
State (Depth:2) {16;32;52}
State (Depth:1) {32;32;36}
State (Depth:0) {0;36;64}

--------------

```

```
State (Depth:8) {5;32;63}
State (Depth:7) {5;31;64}
State (Depth:6) {10;31;59}
State (Depth:5) {10;28;62}
State (Depth:4) {20;28;52}
State (Depth:3) {8;40;52}
State (Depth:2) {16;32;52}
State (Depth:1) {32;32;36}
State (Depth:0) {0;36;64}

--------------



FERTIG!
Man benötigt 8 Telepaartie-Schritte
Die Berechnung dauerte 0:00 Minuten.
```

```
./Telepaartie.CLI -c 4 -e 600 -v
Starting iteration 8

--------------

State (Depth:7) {1;117;191;291}
State (Depth:6) {2;117;191;290}
State (Depth:5) {4;117;189;290}
State (Depth:4) {4;173;189;234}
State (Depth:3) {8;173;189;230}
State (Depth:2) {8;16;230;346}
State (Depth:1) {16;16;222;346}
State (Depth:0) {0;32;222;346}

--------------

State (Depth:7) {3;109;198;290}
State (Depth:6) {6;106;198;290}
State (Depth:5) {6;184;198;212}
State (Depth:4) {12;178;198;212}
State (Depth:3) {12;20;212;356}
State (Depth:2) {8;24;212;356}
State (Depth:1) {16;16;212;356}
State (Depth:0) {0;32;212;356}

--------------

State (Depth:7) {23;94;202;281}
State (Depth:6) {23;187;188;202}
State (Depth:5) {14;23;187;376}
State (Depth:4) {9;28;187;376}
State (Depth:3) {9;56;159;376}
State (Depth:2) {9;103;112;376}
State (Depth:1) {18;103;103;376}
State (Depth:0) {0;18;206;376}

--------------
```

- Einige Ergebnisse der Kürze halber ausgelassen -

```
--------------

```

```
State (Depth:7) {7;122;193;278}
State (Depth:6) {7;85;122;386}
State (Depth:5) {14;85;115;386}
State (Depth:4) {28;85;115;372}
State (Depth:3) {56;57;115;372}
State (Depth:2) {56;58;114;372}
State (Depth:1) {56;56;116;372}
State (Depth:0) {0;112;116;372}

--------------

State (Depth:7) {6;31;202;361}
State (Depth:6) {12;31;196;361}
State (Depth:5) {12;31;165;392}
State (Depth:4) {24;31;165;380}
State (Depth:3) {24;62;134;380}
State (Depth:2) {24;72;124;380}
State (Depth:1) {48;48;124;380}
State (Depth:0) {0;96;124;380}

--------------

State (Depth:7) {1;89;221;289}
State (Depth:6) {2;88;221;289}
State (Depth:5) {4;88;221;287}
State (Depth:4) {8;88;221;283}
State (Depth:3) {16;80;221;283}
State (Depth:2) {32;64;221;283}
State (Depth:1) {64;64;189;283}
State (Depth:0) {0;128;189;283}

--------------

State (Depth:7) {5;31;193;371}
State (Depth:6) {5;31;178;386}
State (Depth:5) {10;31;178;381}
State (Depth:4) {10;62;147;381}
State (Depth:3) {20;52;147;381}
State (Depth:2) {40;52;127;381}
State (Depth:1) {40;52;254;254}
State (Depth:0) {0;40;52;508}

--------------


FERTIG!
Man benötigt 8 Telepaartie-Schritte
Die Berechnung dauerte 0:08 Minuten.
```

```
./Telepaartie.CLI -c 3 -e 5000 -v
Starting iteration 15

--------------

State (Depth:14) {125;1558;3317}
State (Depth:13) {250;1558;3192}
State (Depth:12) {250;1634;3116}
State (Depth:11) {500;1384;3116}
```

```
State (Depth:10) {500;1732;2768}
State (Depth:9) {1000;1232;2768}
State (Depth:8) {232;2000;2768}
State (Depth:7) {464;1768;2768}
State (Depth:6) {928;1304;2768}
State (Depth:5) {1304;1840;1856}
State (Depth:4) {552;1840;2608}
State (Depth:3) {1104;1840;2056}
State (Depth:2) {736;2056;2208}
State (Depth:1) {1472;1472;2056}
State (Depth:0) {0;2056;2944}

--------------

State (Depth:14) {125;1849;3026}
State (Depth:13) {250;1849;2901}
State (Depth:12) {500;1599;2901}
State (Depth:11) {1000;1099;2901}
State (Depth:10) {1000;1802;2198}
State (Depth:9) {396;1000;3604}
State (Depth:8) {792;1000;3208}
State (Depth:7) {208;1584;3208}
State (Depth:6) {416;1376;3208}
State (Depth:5) {832;960;3208}
State (Depth:4) {128;1664;3208}
State (Depth:3) {256;1536;3208}
State (Depth:2) {512;1536;2952}
State (Depth:1) {1024;1024;2952}
State (Depth:0) {0;2048;2952}

--------------
```

- Einige Ergebnisse der Kürze halber ausgelassen -

```
--------------

State (Depth:14) {125;1658;3217}
State (Depth:13) {250;1658;3092}
State (Depth:12) {500;1408;3092}
State (Depth:11) {500;1684;2816}
State (Depth:10) {1000;1184;2816}
State (Depth:9) {1184;1816;2000}
State (Depth:8) {816;1816;2368}
State (Depth:7) {552;816;3632}
State (Depth:6) {816;1104;3080}
State (Depth:5) {288;1632;3080}
State (Depth:4) {576;1344;3080}
State (Depth:3) {768;1152;3080}
State (Depth:2) {768;1928;2304}
State (Depth:1) {1536;1536;1928}
State (Depth:0) {0;1928;3072}

--------------

State (Depth:14) {125;1403;3472}
State (Depth:13) {250;1403;3347}
State (Depth:12) {500;1153;3347}
State (Depth:11) {500;2194;2306}
State (Depth:10) {112;500;4388}
```

```
State (Depth:9) {112;1000;3888}
State (Depth:8) {112;2000;2888}
State (Depth:7) {224;1888;2888}
State (Depth:6) {448;1888;2664}
State (Depth:5) {896;1888;2216}
State (Depth:4) {1320;1792;1888}
State (Depth:3) {472;1888;2640}
State (Depth:2) {944;1888;2168}
State (Depth:1) {1224;1888;1888}
State (Depth:0) {0;1224;3776}

--------------



FERTIG!
Man benötigt 15 Telepaartie-Schritte
Die Berechnung dauerte 0:10 Minuten.
```

4 Quellcode

```
using System;
using System.Collections.Generic;
using System.Linq;

public static class Telepaartie
{
    private const string _separator = "--------------";

    public static int L(
        IEnumerable<int> goalBuckets,
        Action<string>? writeLine = null) //Zum finden der minimalen Anzahl an Operationen
            für einen Zustand
    {
        if (goalBuckets == null) throw new ArgumentNullException(nameof(goalBuckets));

        var goal = new State(goalBuckets);

        var numberOfCups = goalBuckets.Count();
        var numberOfItems = goalBuckets.Sum();

        return LLLCore(numberOfCups, numberOfItems, goal, writeLine);
    }

    public static int LLL(
        int numberOfCups = 3,
        int numberOfItems = 15,
        Action<string>? writeLine = null) //Zum finden der maximalen Anzahl der minimalen
            Anzahlen an Operationen für eine Anzahl
    {
        return LLLCore(numberOfCups, numberOfItems, null, writeLine);
    }

    private static int LLLCore(
        int numberOfCups,
        int numberOfItems,
        State? goal,
        Action<string>? writeLine)
```

```
    {
        HashSet<State> lastGen = new HashSet<State>(
            // Alle Endzustände bilden die nullte Generation
            State.AllEndingStates(numberOfCups, numberOfItems)
            .Select(x => new State(x)));

        HashSet<State> allStates = new HashSet<State>(lastGen);

        for (int i = 0; ; i++)
        {
            writeLine?.Invoke($"\rStarting iteration {i + 1}");

            HashSet<State> nextGen = new HashSet<State>(lastGen
                // Aktiviere Parallelisierung mit PLINQ
                .AsParallel()
                // Ermittle alle Ursprungzustände
                .SelectMany(x => x.Origins()));

            // Entferne Zustände die schon in vorherigen Generationen vorhanden sind
            lastGen.ExceptWith(allStates);

            // Falls die Operationsanzahl für nur einen Zustand festgestellt werden soll
            if (goal != null)
            {
                // Wenn das Element in der neuen Generation vorhanden ist, gebe den
                    Generationsindex, also die Anzahl zum lösen benötigter Telepaartien
                    zurück
                if (nextGen.Contains(goal)) return i + 1;
            }
            // Wenn die neue Generation die leere Menge ist
            else if (nextGen.Count == 0)
            {
                // Output
                if (writeLine != null)
                {
                    writeLine(Environment.NewLine);
                    foreach (var oldestChild in lastGen)
                    {
                        writeLine(Environment.NewLine + _separator + Environment.NewLine +
                            Environment.NewLine);

                        for (State? current = oldestChild; current != null; current =
                            current.Parent)
                        {
                            writeLine(current.ToString() + Environment.NewLine);
                        }
                    }

                    writeLine(Environment.NewLine + _separator + Environment.NewLine +
                        Environment.NewLine);
                }

                // Gebe den Generationsindex, also die Anzahl zum lösen benötigter
                    Telepaartien zurück
                return i + 1;
            }

```

```
            // Zur Sammlung aller bisher entdeckten Zustände die jetzige Generation
                hinzufügen.
            allStates.UnionWith(nextGen);
            // Die letzte Generation durch die jetzige ersetzen, um die nächste korrekt
                ausrechnen zu lassen
            lastGen = nextGen;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;

public class State : IEquatable<State>
{
    public int Depth => Parent == null ? 0 : (Parent.Depth + 1);

    public State? Parent { get; }

    public int[] Buckets { get; }

    private readonly int _hashCode;

    public State(IEnumerable<int> unsortedBuckets, State? parent = null)
    {
        if (unsortedBuckets.Any(x => x < 0)) throw new
            ArgumentException(nameof(unsortedBuckets));

        Buckets = unsortedBuckets.ToArray();
        Array.Sort(Buckets);

        Parent = parent;
        _hashCode = CalculateHashCode();
    }

    private State(int[] sortedBuckets, State? parent = null)
    {
        Buckets = sortedBuckets;
        Parent = parent;
        _hashCode = CalculateHashCode();
    }

    private State ReverseTeelepartie(int originalTarget, int originalSource)
    {
        int[] temp = new int[Buckets.Length];
        Buckets.CopyTo(temp, 0);

        temp[originalTarget] /= 2;
        temp[originalSource] += temp[originalTarget];
        Array.Sort(temp);

        return new State(temp, this);
    }

    public IEnumerable<State> Origins()
    {
        // Finden jeder Kombination
```

```
        for (int i = 0; i < Buckets.Length; i++)
        {
            for (int u = 0; u <= i; u++)
            {
                // Zulässige Werte rausfiltern
                if (Buckets[i] % 2 == 0 && Buckets[i] > 0)
                {
                    // und die bearbeitete Version zurückgeben
                    yield return ReverseTeelepartie(i, u);
                }

                if (Buckets[u] % 2 == 0 && Buckets[u] > 0)
                {
                    yield return ReverseTeelepartie(u, i);
                }
            }
        }
    }

    private int CalculateHashCode() =>
        Buckets.Aggregate(168560841, (x, y) => (x * -1521134295) + y);

    public static IEnumerable<List<int>> AllEndingStates(int numberOfCups, int
        numberOfItems)
    {
        foreach (var state in AllPossibleStates(numberOfCups - 1, numberOfItems,
            numberOfItems))
        {
            state.Add(0);
            yield return state;
        }
    }

    public static IEnumerable<List<int>> AllPossibleStates(int numberOfCups, int
        numberOfItems, int previousMax)
    {
        if (numberOfCups < 1) yield break;
        if (numberOfCups == 1) yield return new List<int> { numberOfItems };

        // Die Elementanzahl die mindestens dem aktuellen Behälter hinzugefügt werden muss
        int min = ((numberOfItems - 1) / numberOfCups) + 1;

        // Die Elementanzahl die maximal dem aktuellen Behälter hinzugefügt werden kann
        int max = Math.Min(previousMax + 1, numberOfItems);

        for (int i = min; i < max; i++)
        {
            // Finden aller Möglichen Kombinationen für den Rest der Biber und der
                Behälteranzahl -1
            foreach (var state in AllPossibleStates(numberOfCups - 1, numberOfItems - i, i))
            {
                state.Add(i);
                yield return state;
            }
        }
    }
}
```