Telepaartie

Lösungsidee

Da bei günstigen Telepaartien recht "schnell" ein Behältnis leer zu sein scheint (konvergiert rasch), verwenden wir eine **Brute-Force**. Zunächst stellen wir fest, dass Telepaartien symmetrisch sind: telepaartie(a, b) = telepaartie(b, a). Somit ergeben sich immer nur 3 mögliche Telepaartien: Behälter 1 & 2, 1 & 3 oder 2 & 3. Um also für eine Biberverteilung b die LLL(b) zu berechnen, müssen wir schließlich 3^n Möglichkeiten betrachten - da n meist relativ gering ist, wird im Normalfall eine sehr gute Laufzeit erzielt.

Ähnlich gehen wir beim zweiten Aufgabenteil vor: L(n) = maximale LLL(b) für alle b mit insgesamt n Bibern. Also generieren wir nun alle Möglichkeiten, n Biber auf 3 Behälter zu verteilen. Diese gehen wir dann alle durch und bestimmen wie oben beschrieben für jede die LLL(b). Von diesen wählen wir dann das Maximum = L(n).

Umsetzung

Etwas anders als in der Aufgabenstellung gefordert nur mit einem und nicht mit zwei Programmen umgesetzt, siehe Verwendung.

Bibliotheken

- `java.util.ArrayList`: Dynamische Liste wird als Zwischenspeicher für die iterative Brute-Force verwendet
- `java.util.Arrays`: Nützliche Array-Helfermethoden. Verwendet, um `equals` und `hashCode` zu implementieren.
- `java.util.Iterator`: Schnittstelle, die von `String.join` gebraucht und von [BiberIterator] implementiert wird.

Klassen

`Biberverteilung`

Speichert eine Biberverteilung mithilfe eines `int`-Arrays (da die Länge = 3 ja schon von Anfang an feststeht). Liefert eine Methode `telepaartie`, die zwei Indices von Behälter nimmt, eine Telepaartie durchführt und eine neue [Biberverteilung] zurückgibt. Falls die Telepaartie einen der beiden Behälter leeren sollte, wird `null` zurückgegeben.

`Biberverteilung$BiberIterator`

Eine Nutzklasse, um die Java-Funktion `String.join` nutzen zu können, die eine `Iterable<String>` nimmt und bei `Biberverteilung.toString` zum Einsatz kommt. Außerdem kann die Biberverteilung so `Iterable<String>` implementieren und leichter durchgegangen werden.

`Telepaartie`

Die Hauptklasse. Enthält die `main`-Methode, lädt die Aufgabe aus den Argumenten - bei genau einem Argument wird L(n) berechnet, bei dreien wird die LLL der [Biberverteilung] berechnet. Enthält die brute-force Methoden die dies tun:

- `berechneL`: Nimmt ein n (`int`) und probiert alle Möglichkeiten aus, n auf drei Behälter zu verteilen: Mit 2 geschachtelten `for`-Schleifen werden zunächst in den ersten Behälter k = n-2 bis 1 Biber getan, dann in den zweiten bis zu n-k-1 - also maximal einer weniger als noch übrig sind - schließlich muss ja auch für den letzten noch min. ein Biber bleiben. In diesen werden einfach die übrigen Biber getan. Für n <= 2 gibt die Methode einfach 0 zurück, da alle Möglichkeiten direkt einen leeren Behälter enthalten würden.
- `berechneLLL`: Nimmt eine Biberverteilung und berechnet iterativ (mit einer jeweils auf 3^n wachsenden ArrayList) alle Möglichkeiten, Telepaartien durchzuführen - so lange, bis ein Behälter geleert werden kann, was meist recht früh der Fall ist. Anders als bei "Nummernmerker" ist hier eine "depth-first-search" nicht zielführend, da einige Pfade sehr lange sein können (bspw. 2000 Telepaartien) und schließlich sowieso verworfen werden müssen, da irrwitzige kürzere (bspw. 8 Telepaartien) existieren. Daher nutzen wir hier einen iterativen "level-order-tree-traversal" - der Baum wird Ebene für Ebene abgesucht, sodass wir auf diese kurzen Lösungen zuerst stoßen und nicht riskieren, lange, unnütze Pfade zu untersuchen. Dafür ist zwar der Speicherverbrauch höher, doch dies spielt bei den relativ kleinen Aufgaben (siehe Beispiele) keine Rolle, vor Allem, da die Telepaartie schnell konvergiert ((erfahrungsgemäß) ist nach ziemlich wenigen Schritten schon ein Gefäß leer).

Verwendung

Berechnen der LLL(a, b, c): `java -jar Biberverteilung.jar a b c`

Berechnen der L(n): `java -jar Biberverteilung.jar n`

Beispiele

Verteilungen

`(2, 4, 7)`

```
1 LLL(2, 4, 7)=2
```

`(3, 5, 7)`

```
1 LLL(3, 5, 7)=3
```

`(80, 64, 32)`

```
1 LLL(80, 64, 32)=2
```

Maximale Leerlauflängen (`L(n)`)

n=10

```
L(1)=0
L(2)=0
L(3)=1
L(4)=1
L(5)=1
L(6)=2
L(7)=2
L(8)=2
L(9)=2
L(10)=2
```

n=100

```
L(1)=0
L(2)=0
L(3)=1
L(4)=1
L(5)=1
L(6)=2
L(7)=2
L(8)=2
L(9)=2
L(10)=2
L(11)=3
L(12)=3
L(13)=3
L(14)=3
L(15)=4
L(16)=3
L(17)=3
L(18)=3
L(19)=4
L(20)=3
L(21)=4
L(22)=4
L(23)=5
L(24)=4
L(25)=5
L(26)=4
L(27)=6
L(28)=4
L(29)=5
L(30)=5
L(31)=5
```

```
L(32)=4
L(33)=6
L(34)=4
L(35)=5
L(36)=4
L(37)=5
L(38)=5
L(39)=6
L(40)=4
L(41)=5
L(42)=5
L(43)=6
L(44)=5
L(45)=7
L(46)=5
L(47)=5
L(48)=5
L(49)=6
L(50)=6
L(51)=7
L(52)=5
L(53)=6
L(54)=6
L(55)=6
L(56)=5
L(57)=7
L(58)=5
L(59)=6
L(60)=6
L(61)=7
L(62)=6
L(63)=7
L(64)=5
L(65)=6
L(66)=7
L(67)=6
L(68)=5
L(69)=7
L(70)=6
L(71)=6
L(72)=5
L(73)=6
L(74)=6
L(75)=7
L(76)=6
L(77)=7
L(78)=6
L(79)=6
```

```
L(80)=5
L(81)=8
L(82)=6
L(83)=7
L(84)=6
L(85)=7
L(86)=6
L(87)=7
L(88)=6
L(89)=7
L(90)=7
L(91)=7
L(92)=6
L(93)=8
L(94)=6
L(95)=8
L(96)=6
L(97)=7
L(98)=7
L(99)=8
L(100)=7
```

Quellcode

Biberverteilung.java

```java
package appguru;

import java.util.Arrays;
import java.util.Iterator;

/**
 *
 * @author lars
 */

// Speichert eine Biberverteilung
public class Biberverteilung implements Iterable<String> {

    public int[] biber; // Array der Biber-Anzahlen = Behälter

    // Konstruktor
    public Biberverteilung(int[] biber) {
        this.biber = biber;
    }

```

```java
    // Telepaartie: Erzeugt veränderte Kopie der Biberverteilung, gibt null zurück wenn die Telepaartie einen Behälter leert
    public Biberverteilung telepaartie(int b1, int b2) {
        if (biber[b2] == biber[b1]) { // Telepaartie leert b1 / b2
            return null; // null zurückgeben
        }
        // Array kopieren
        int[] copied = new int[biber.length];
        System.arraycopy(biber, 0, copied, 0, biber.length);
        // Falls Inhalt von b1 > Inhalt b2, vertausche b1 und b2, sodass b1 immer weniger Biber hat
        if (biber[b1] > biber[b2]) {
            int b1_backup = b1;
            b1 = b2;
            b2 = b1_backup;
        }
        // Ziehe von b2 die Biber von b1 ab (b2 hat ja mehr)
        copied[b2] -= copied[b1];
        // Verdoppple die Anzahl der Biber in b1 (dem mit weniger)
        copied[b1] *= 2;
        return new Biberverteilung(copied); // Gebe eine neue Biberverteilung mit dem kopierten Array zurück
    }

    @Override
    public String toString() {
        // Gibt eine Biberverteilung aus, Anzahlen mit Kommas getrennt
        return "(" + String.join(", ", this) + ")";
    }

    // Iterator, um String.join nutzen & Anzahlen als Strings durchgehen zu können
    class BiberIterator implements Iterator<String> {

        public int i;

        @Override
        public boolean hasNext() {
            return i < biber.length;
        }

        @Override
        public String next() {
            return Integer.toString(biber[i++]);
        }

    }

    @Override
```

```
66      public Iterator<String> iterator() {
67          return new BiberIterator(); // Iterator erzeugen & zurückgeben
68      }
69
70      @Override
71      public boolean equals(Object object) {
72          // Prüft, ob zwei Biberverteilungen gleich sind; nutzt hauptsächlich Arrays.equals
73          if (object.getClass() != Biberverteilung.class) {
74              return false;
75          }
76          Biberverteilung b = (Biberverteilung) object;
77          return Arrays.equals(b.biber, this.biber);
78      }
79
80      @Override
81      public int hashCode() {
82          // Generiert einen Hash
83          return Arrays.hashCode(biber);
84      }
85  }
```

Telepaartie.java

```
1   package appguru;
2
3   import java.util.ArrayList;
4
5   /**
6    *
7    * @author lars
8    */
9   public class Telepaartie {
10
11      // Berechnet LLL(start), wobei start eine Biberverteilung ist, iterativ
12      public static int LLL(Biberverteilung start) {
13          ArrayList<Biberverteilung> verteilungen = new ArrayList(); // Verteilungen
14          verteilungen.add(start);
15          for (int s = 1;; s++) { // Schritte zählen
16              // Verteilungen für den nächsten Schritt
17              ArrayList<Biberverteilung> neue_verteilungen = new ArrayList();
18              // Biberverteilungen durchgehen
19              for (Biberverteilung v : verteilungen) {
20                  // Alle Möglichkeiten, Telepaartien durchzuführen durchgehen
21                  for (int i = 0; i <= 1; i++) {
22                      for (int j = i + 1; j <= 2; j++) {
23                          Biberverteilung v_neu = v.telepaartie(i, j); // Telepaartie durchführen
24                          if (v_neu == null) { // Behälter geleert
25                              return s; // Anzahl benötigter Schritte zurückgeben, fertig!
```

```
                    }
                    neue_verteilungen.add(v_neu); // Neue Verteilung speichern
                }
            }
        }
        // Nächster Schritt, nächste Verteilungen
        verteilungen = neue_verteilungen;
    }
}

// Berechnet L(n) - Geht alle Kombinationen durch und findet die maximale LLL
public static int L(int n) {
    // Wenn n <= 2 ist sowieso immer ein Behälter leer
    if (n <= 2) {
        return 0;
    }
    // Ansonsten: Initialisiere "Rekordhalter" mit dem kleinstmöglichen Wert
    int max_schritte = Integer.MIN_VALUE;
    // Gehe alle Möglichkeiten durch, Gefäße zu befüllen, möglichst ohne verschiedene
        Anordnungen
    for (int i = 1; i < n - 1; i++) {
        for (int j = n - i - 1; j >= 1; j--) {
            int k = n - j - i;
            int schritte = LLL(new Biberverteilung(new int[]{i, j, k})); // LLL berechnen
            if (schritte > max_schritte) { // wurden mehr Schritte benötigt? (ist die LLL
                größer?)
                max_schritte = schritte; // neuer Rekordhalter
            }
        }
    }
    return max_schritte; // Rekordhalter zurückgeben
}

public static void main(String[] args) {
    try {
        if (args.length == 1) { // für ein Argument
            int n = Integer.parseInt(args[0]); // zu Zahl umwandeln
            if (n < 0) {
                System.out.println("Fehler: n < 0");
            } else {
                // L(n) berechnen und ausgeben
                if (n == 0) {
                    System.out.println("L(0)=0");
                }
                for (int i = 1; i <= n; i++) {
                    System.out.println("L(" + i + ")=" + L(i));
                }
            }
```

```
72            } else if (args.length == 3) { // falls drei Argumente gegeben sind
73                // lese Verteilung ein
74                int[] verteilung = new int[3];
75                boolean schon_null = false;
76                for (byte b = 0; b < 3; b++) {
77                    verteilung[b] = Integer.parseInt(args[b]); // zu Zahl umwandeln
78                    if (verteilung[b] <= 0) {
79                        schon_null = true; // ein Behälter ist schon <= null, fertig!
80                    }
81                }
82                Biberverteilung vert = new Biberverteilung(verteilung);
83                // für einen Behälter, der schon <= 0 ist direkt 0 ausgeben, sonst LLL berechnen
84                System.out.println("LLL" + vert + "=" + (schon_null ? "0":LLL(vert)));
85            } else {
86                // Instruktionen
87                System.out.println("Geben sie 1 (=n) oder 3 (=Biberverteilung) Argumente an.");
88            }
89        } catch (NumberFormatException ne) {
90            // Fehlermeldung
91            System.out.println("Keine korrekten Zahlen");
92        }
93    }
94
95 }
```