Junioraufgabe 2 - Container

Team-ID: 00166
Team-Name: ez

Bearbeitet von Florian Bange

21. November 2022

Inhaltsverzeichnis

1 Definition des Problems

Um das Problem zu loesen, wird es zunaechst formell definiert.

Sei

$$C = \{c_1,\ c_2,\ \ldots,\ c_n\}$$

die Menge mit $|C| = n$ an n Containern, wobei fuer alle $a,\ b \in C$ mit $a \neq b$ gilt, dass a und b nicht das gleiche Gewicht haben.

Gegeben sind Paare $(a,\ b)$ mit $a, b \in C$, wobei a schwerer ist als b als eine Relation $R \subseteq C \times C$, wobei

$$C \times C := \{(a,\ b)\ :\ a, b \in C\}.$$

Gesucht ist nun ein $c \in C$, welches schwerer ist als alle anderen Container:

$$\forall c_i \in C : (c_i \neq c \implies c > c_i)$$

Dies soll allerdings ausschliszlich durch die Paare aus R ermittelt werden.

2 Loesungsidee / Loesung

Um das soeben definierte Problem zu loesen, sei L die Menge, die alle Elemente von C enthaelt, welche auf keiner rechten Seite eines Paares in R stehen:

$$L := \{x \in C \mid \forall c \in C : (c,\ x) \notin R\}$$

Bzw. ist dies die Menge, welche alle Container enthaelt, von denen man weisz, dass sie nicht leichter als irgendein anderer Container sind.

Diese Menge kann einfach erhalten werden, indem von der Menge aller Container, alle Container entfernt werden, welche auf einer rechten Seite eines Paares stehen.
Die Menge aller Container kann erhalten werden, da jeder Container mindestens einmal gewogen wurde und somit mind. einmal in einem Paar aus R vorkommt. (siehe Algorithm 1) Nun gilt fuer alle $c \in L$,

Algorithm 1 Pseudocode zum erhalten aller poteniell schwersten Container

```
1:  procedure getContainers((a_1, b_1), ..., (a_m, b_m))
2:      A ← new empty set
3:      R ← new empty set
4:
5:      for i = 1, 2, ..., m do
6:          A.add(a_i)
7:          A.add(b_i)
8:
9:          R.add(b_i)
10:     end for
11:
12:     H ← new empty set
13:     Add all elements of A to H
14:     Remove all elements of R from H
15:
16:     return H
17: end procedure
```

dass c ein potentiell schwerster Container ist.
Bzw. formell: Fuer jedes $c \in L$ gilt, dass kein Container existiert, der nach Ungleichungen aus R schwerer ist als c.

Dies ist korrekt, da keine Kombination von Ungleichungen aus R erzeugt werden kann, bei der c ganz

rechts steht (weil c niemals rechts steht).

Das Ergebnis des definierten Problems, ist nun abhaengig von der Groesze / Laenge von L.

1. Fuer $|L| = 1$ ist der schwerste Container eindeutig.
2. Fuer $|L| > 1$ ist der schwerste Container nicht eindeutig und es existiert nur eine Menge an moeglichen Containern.
3. $|L| = 0$ ist nicht moeglich, da dafuer kein $x \in C$ existieren duerfte, sodass $\forall c \in C : (c,\ x) \notin R$ gilt. Bzw.

$$\neg\exists x \in C : \forall c \in C : (c,\ x) \notin R$$
$$\Leftrightarrow \forall x \in C : \exists c \in C : (c,\ x) \in R$$

   Dies ist allerdings fuer den schwersten Container z, welcher eindeutig existiert, nicht moeglich, da kein $c \in C$ existiert mit $c > z$.

3 Implementierung / Umsetzung

Die soeben beschriebene Loesungsidee wurde in Java 8 implementiert.

Dabei wurden die Paare als Liste von (selbstimplementierten) Paaren dargestellt.
Die Mengen aller Container und aller Container auf einer rechten Seite eines Paares wurden als HashSets implementiert, sodass keine doppelten Element vorkommen.
Dementspraechend wurde die Menge an allen Containern, welche auf keiner rechten Seite eines Paares sind, ebenfalls als HashSet implementiert, sodass andere HashSets einfach vollstaendig hinzugefuegt, bzw. entfernt werden koennen.

4 Laufzeit- und Platzkomplexitaet

Sei die Eingabe Laenge n definiert als Anzahl der Paare.

Die Laufzeitkomplexitaet dieses Algorithmus liegt in $\mathcal{O}(n)$, denn es werden ausschliezlich ein Mal alle Paare durchlaufen und des Weiteren ein Mal ein HashSet zu einem anderen hinzugefuegt und eines von einem anderen entfernt. Wobei alle HashSets eine Groesze in $\mathcal{O}(n)$ haben und diese Operationen bei HashSets (aufgrund des Hashings) linear sind.

Die Platzkomplexitaet liegt ebenfalls in $\mathcal{O}(n)$, da (wie bereits erwaehnt) eine Liste aller Paare existiert und mehrere HashSet mit einer maximalen Groesze von n existieren.

5 Beispiele

5.1 Beispieleingabe 1 - container0

Der schwerste Container ist nicht eindeutig. Die potentiell schwersten Container sind 3, 4 und 5.

5.2 Beispieleingabe 2 - container1

Der schwerste Container ist 4.

5.3 Beispieleingabe 3 - container2

Der schwerste Container ist nicht eindeutig. Die potentiell schwersten Container sind 1 und 3.

5.4 Beispieleingabe 4 - container3

Der schwerste Container ist nicht eindeutig. Die potentiell schwersten Container sind 5 und 7.

5.5 Beispieleingabe 5 - container4

Der schwerste Container ist 5.

5.6 Eigenes Beispiel 1

Eingabe:

1. 9 8
2. 8 7
3. 7 6
4. 6 5
5. 5 4
6. 4 3
7. 3 2
8. 2 1

Ausgabe:
Der schwerste Container ist 9.

Diese Ausgabe ist korrekt, da sich folgende Ungleichung ergibt:

$$9 > 8 > 7 > 6 > 5 > 4 > 3 > 2 > 1.$$

6 Quellcode

6.1 Funktion zum berechnen der potentiell schwersten Container

```java
/**
 * Function for getting all containers that are potentially the heaviest,
 * given a list of pairs of containers
 * where the first container is heavier than the second one
 *
 * @param pairs Given pairs (c_1, c_2) of containers where c_1 > c_2
 * @return A HashSet of potential heaviest containers
 */
private static Set<Integer> getPotentialHeaviestContainers(List<Pair<Integer, Integer>> pairs) {
    // Fill HashSets with 1) all containers 2) containers on the right (of the pairs)
    Set<Integer> allContainers = new HashSet<>(); // 1)
    Set<Integer> rightSights = new HashSet<>(); // 2)

    for (Pair<Integer, Integer> pair : pairs) {
        int a = pair.getFirst();
        int b = pair.getSecond();

        allContainers.add(a);
        allContainers.add(b);

        rightSights.add(b);
    }

    // Create a new HashSet containing all containers and remove all containers that
    // are on any right side, such that the new HashSet contains all containers
    // that are never on a right side and hence cannot be lighter than any other container
    Set<Integer> biggest = new HashSet<>(allContainers);
    biggest.removeAll(rightSights);

    return biggest;
}
```