Junioraufgabe 2: Container

J2.1 Lösungsidee

In dieser Aufgabe gibt es mehrere unterschiedlich schwere Container, von denen wir allerdings nicht wissen, wie schwer diese sind. Wir haben lediglich als Information eine Liste von paarweisen Wägungen gegeben, die jeweils angeben, ob ein Container schwerer als ein anderer ist. Weiterhin ist bekannt, dass jeder Container mindestens einmal zusammen mit einem anderen Container auf der Containerpaarwaage war. Nun soll man herausfinden, ob man ausgehend von diesen Informationen sicher sagen kann, welcher Container der schwerste ist.

Zur Veranschaulichung bietet es sich an, diese Wägungen wie in Abbildung J2.1 in einem gerichteten azyklischen Graphen[2] zu modellieren, wobei für jeden Container ein Knoten reserviert wird. Hierbei bedeutet ein Pfeil von einem Container A zu einem Container B, dass A mit B auf der Waage war und A schwerer als B ist.

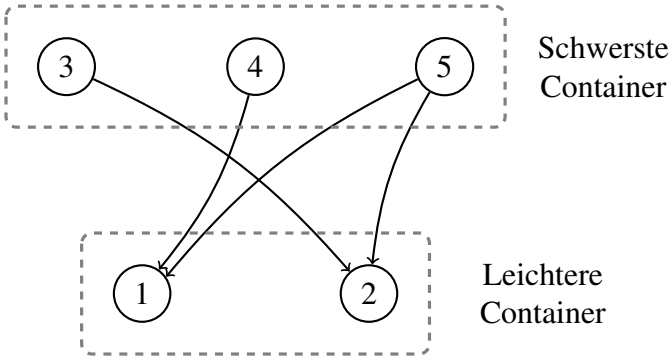

Abbildung J2.1: Darstellung der Abhängigkeiten im Beispiel `container0.txt`

Zunächst fällt einem auf, dass in diesem Beispiel die Container 1 und 2 nicht die schwersten sein können, denn Container 1 ist leichter als die Container 4 und 5 und Container 2 ist leichter als die Container 3 und 5. Allgemein gilt, dass ein Container, der bei einer Wägung leichter als irgendein anderer Container ist, auf jeden Fall nicht der schwerste sein kann.

Es gilt also, dass es zu einem schwersten Container nie einen schwereren geben darf. Anhand der gegebenen Vergleiche kann man alle Container sammeln, die diese Eigenschaft erfüllen, nämlich dass es jeweils keinen als schwerer bekannten Container gibt. Wir bezeichnen diese Container im Folgenden als *Kandidaten*. Hier gibt es zwei Fälle:

- Entweder es gibt nur genau einen möglichen Kandidaten. Dann muss dieser Container der schwerste sein, denn jeder andere Container war zumindest einmal der leichtere bei einer Wägung.
- Andernfalls gibt es mehrere Kandidaten. Hier ist nicht eindeutig, welcher dieser Kandidaten der schwerste Container ist. Ein Beispiel zeigt bereits die Abbildung J2.1: Die Container 3, 4 und 5 sind bei keiner Wägung leichter als ein anderer Container. Allerdings müssen diese nach Aufgabenstellung unterschiedlich schwer sein. Nachdem aber zwischen den Containern 3, 4 und 5 keine Wägungen vorgenommen wurden, kann man nicht wissen, welcher der schwerste ist.

[2] Für eine Erklärung, was ein gerichteter Graph ist, siehe `https://de.wikipedia.org/wiki/Gerichteter_Graph`. Ein azyklischer Graph ist ein Graph ohne Zyklen. Für mehr Informationen zu Zyklen siehe `https://de.wikipedia.org/wiki/Graph_(Graphentheorie)#Teilgraphen,_Wege_und_Zyklen`.

Algorithmische Umsetzung

Ausgehend von den vorherigen Überlegungen genügt es somit zur Lösung der gestellten Aufgabe, die Container zu finden, die in keiner Wägung leichter waren. Effizient lässt sich das beispielsweise wie in Algorithmus 1 realisieren, der als Eingaben die Container (containers) sowie die Wägungen (relations, jeweils Paare der Form $(\text{larger}, \text{smaller})$) entgegennimmt.

Algorithmus 1 Bestimmen möglicher Kandidaten mit größtem Gewicht

```
procedure FINDCANDIDATES(containers, relations)
    candidates ← containers
    for (larger, smaller) in relations do
        candidates ← candidates \ {smaller}
    end for
    return candidates
end procedure
```

Die Idee ist hier, dass wir zunächst davon ausgehen, dass noch keine Wägungen durchgeführt wurden. Somit kann jeder Container der schwerste Container sein und gilt als Kandidat. Danach wird jede Wägung überprüft: Ist hierbei ein Container leichter als ein anderer, ist er kein möglicher Kandidat mehr und wird aus der Liste der Kandidaten, falls noch darin vorhanden, entfernt. Falls anschließend nur noch ein Element in der Kandidatenliste enthalten ist, so ist eindeutig, welcher Container der schwerste ist. Nämlich eben dieser letzte Kandidat, der noch in der Liste ist. Falls noch mehrere Elemente in der Kandidatenliste enthalten sind, so ist nicht eindeutig feststellbar, welcher Container der schwerste Container ist.

Laufzeit

Als Eingaben des Algorithmus haben wir die Menge der Container (containers) sowie die gegebenen Vergleiche (relations). Der vorgestellte Algorithmus besteht im Wesentlichen aus zwei Phasen:

1. Zunächst nehmen wir an, dass alle Container potenziell die schwersten sein können und erstellen eine entsprechende Menge. Dann entfernen wir für jeden gegebenen Vergleich maximal ein Element von den Kandidaten.

   Die Laufzeit hängt also hierfür vor allem davon ab, welche Datenstruktur man für die Verwaltung einer Menge M verwendet. Die bekanntesten sind Hashtabellen[3], balancierte Binärbäume[4] oder Bitarrays[5]. Im Regelfall weist die Hashtabelle für die üblichen Operationen wie Finden, Einfügen oder Löschen eines Elementes eine konstante Laufzeit auf, diese kann aber im seltenen Fall auch $\mathcal{O}(|M|)$ sein. Ein balancierter Suchbaum hat für diese Operationen stets eine Laufzeitkomplexität von $\mathcal{O}(\log(|M|))$. Ein Bitarray (also ein Array von Bits, dessen i-tes Bit angibt, ob das mit i indizierte Element in M enthalten ist) hat garantiert konstante Laufzeit, setzt allerdings voraus, dass man zu jedem Element schnell einen zugehörigen Index finden kann. Dies trifft beispielsweise dann zu, wenn die Container bereits aufsteigend mit den natürlichen Zahlen $(1, 2, 3, \ldots)$ benannt worden

[3] `https://de.wikipedia.org/wiki/Hashtabelle`
[4] `https://de.wikipedia.org/wiki/Balancierter_Baum`
[5] `https://de.wikipedia.org/wiki/Bitkette`

sind. Eine solche Nummerierung gibt ein Eingabeformat wie das der Beispieleingaben bereits vor.

Wir nehmen im Folgenden an, dass wir die üblichen Operationen auf Mengen in konstanter Zeit absolvieren können. Damit erhalten wir für das Erzeugen der Menge einen Laufzeitaufwand von $\mathcal{O}(|\text{containers}|)$ und für die Entfernung der Elemente einen Laufzeitaufwand von $\mathcal{O}(|\text{relations}|)$.

2. Anschließend muss man noch überprüfen, ob die Menge der Kandidaten die Mächtigkeit 1 hat, also nur noch ein Element enthalten ist und das Ergebnis eindeutig ist. Andernfalls sollte die Mächtigkeit ggf. ausgegeben werden. Das geschieht in konstanter Zeit.

Zusammengefasst erhalten wir also eine Laufzeit von $\mathcal{O}(|\text{containers}| + |\text{relations}|)$.

Anmerkung: Wir gehen in unserer Lösung davon aus, dass alle gegebenen paarweisen Wägungen konsistente Abhängigkeiten verursachen. Beispielsweise darf für zwei Container A und B nicht sowohl „A schwerer als B“ als auch „B schwerer als A“ gewogen worden sein. In diesem Fall hätte man in dem Abhängigkeitsgraphen einen Zyklus, den man erkennen und damit dem Nutzer eine entsprechende Warnung ausgeben könnte.

J2.2 Ein anderer Lösungsweg

Anstatt die Eigenschaft auszunutzen, dass ein schwerster Container in einer Wägung nie der leichtere ist, kann man die Eigenschaft auch expliziter überprüfen: Wir verwenden hierbei, dass Gewichtsrelationen transitiv sind, d. h. wenn für drei Container A, B und C sowohl „A größer als B“ als auch „B größer als C“ gemessen wird, muss auch „A größer als C“ gelten. Können wir also mithilfe der transitiven Eigenschaft für einen Container zeigen, dass dieser schwerer als alle anderen ist, haben wir den schwersten Container gefunden.

Algorithmus 2 Finden aller definitiv leichteren Container mittels Tiefensuche

```
procedure FINDSMALLER(curr, relations, visited)
    if curr ∈ visited then
        return visited
    end if
    for neigh ∈ {x | (curr, x) ∈ relations} do
        visited ← FINDSMALLER(neigh, relations, visited)
        visited ← visited ∪ {neigh}
    end for
    return visited
end procedure
```

Eine mögliche Realisierung mittels Tiefensuche[6] beschreibt Algorithmus 2. Wir betrachten zunächst Container, von denen man ausgehend von der Information in relations direkt weiß, dass diese kleiner als der aktuelle Container sind. Für jedes kleinere Element kann man nun aufgrund der transitiven Eigenschaft der Größer-Relation rekursiv weitergehen und weitere Container sammeln. Bereits besuchte Container im rekursiven Aufruf lassen sich überspringen, um dadurch Laufzeit zu sparen. Wenn man für einen Container alle anderen Container in visited gesammelt hat, weiß man, dass dieser der schwerste ist. Die Laufzeitkomplexität der Tiefensuche

[6] `https://de.wikipedia.org/wiki/Tiefensuche`

ist bekanntlich $\mathcal{O}(|\text{containers}| + |\text{relations}|)$, unter der Voraussetzung, dass die Datenstruktur relations die leichteren Container zu einem Container effizient herausfinden kann.

Dieses Prozedere müssen wir für alle Container wiederholen. Im schlimmsten Fall hat man also eine Laufzeitkomplexität von $\mathcal{O}(|\text{containers}| \cdot (|\text{containers}| + |\text{relations}|))$, was zwar eine Verschlechterung gegenüber der anderen vorgestellten Lösungsidee bedeutet, aber trotzdem noch eine akzeptable Laufzeit ist.

J2.3 Beispiele

Die folgende Tabelle zeigt die Ergebnisse für die vorgegebenen Beispiele. Es ist jeweils angegeben, welche Container potenziell die schwersten nach dem Kriterium aus der Lösungsidee in Abschnitt J2.1 sind, sowie, ob man eindeutig den schwersten Container identifizieren konnte (es also nur einen Kandidaten gibt).

Beispiel	**Mögliche Kandidaten**	**Eindeutig?**
`container0.txt`	$\{3,4,5\}$	nein
`container1.txt`	$\{4\}$	ja
`container2.txt`	$\{1,3\}$	nein
`container3.txt`	$\{5,7\}$	nein
`container4.txt`	$\{5\}$	ja

Für die Eingaben `container0.txt` und `container1.txt` lassen sich die Ergebnisse leicht exemplarisch nachvollziehen. Wie bereits in Abbildung J2.1 dargestellt, kann man die Container 1 und 2 als schwerste Container ausschließen, da diese kleiner sind als andere. Zwischen den Containern 3, 4 und 5 ist allerdings keine Relation gegeben. Jeder von diesen könnte der schwerste sein, wodurch wir kein eindeutiges Ergebnis erhalten.

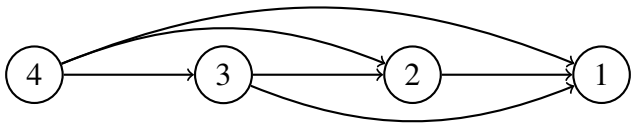

Abbildung J2.2: Darstellung der Abhängigkeiten im Beispiel `container1.txt`

Bei der zweiten Eingabe (Abbildung J2.2) sieht man hingegen, dass es für alle Container, bis auf einen, einen anderen Container gibt, der schwerer ist. Insbesondere wurde hier das Gewicht jedes Containers mit dem jedes anderen verglichen. Container 4 ist also eindeutig der schwerste.

J2.4 Bewertungskriterien

Die Bewertungskriterien vom Bewertungsbogen werden hier erläutert (Punktabzug in []).

- [−1] **Eindeutigkeit nicht korrekt festgestellt**
  Das Lösungsverfahren muss erkennen können, ob es mehrere Container gibt, die die schwersten sein können, und dann feststellen, dass die Lösung nicht eindeutig ist.
- [−1] **Lösungsverfahren anderweitig fehlerhaft**
  Insbesondere soll die Menge der Kandidaten für den schwersten Container – ob eindeutig oder nicht – keinen falschen Container enthalten.

  Falls schon bei *Eindeutigkeit nicht korrekt festgestellt* ein Punkt abgezogen wurde, so soll hier nicht erneut ein Punkt abgezogen werden.
- [−1] **Ergebnisse schlecht nachvollziehbar**
  Es muss ausgegeben werden, ob der schwerste Container eindeutig bestimmt werden kann, und, wenn ja, welcher dies ist. Lediglich auszugeben, ob der schwerste Container eindeutig bestimmt werden kann, genügt nicht. Falls die Lösung nicht eindeutig ist, so muss nicht die Menge an Kandidaten ausgegeben werden, sondern es genügt ein Hinweis, dass die Lösung nicht eindeutig ist.
- [−1] **Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele**
  Die Dokumentation soll Ergebnisse zu mindestens 4 der vorgegebenen Beispiele (`container0.txt` bis `container4.txt`) enthalten.