Aufgabe 2: Alles Käse

Teilnahme-Id: 65438

Bearbeiter/-in dieser Aufgabe:
Raphael Gaedtke

10. April 2023

Inhaltsverzeichnis

1 Lösungsidee

1.1 Aufgabenteil a: Ein vollständiger Käsequader

Ob und wie eine Menge von Scheiben zu einem vollständigen Quader zusammengesetzt werden kann, wird durch die umgekehrte Berechnung überprüft:
Für bestimmte Seitenlängen eines Quaders wird ermittelt, ob und wie es möglich ist, dass das Zerschneiden dieses Körpers zur gegebenen Menge von Scheiben führt. Um das Problem laufzeitsparend zu lösen, muss dann noch eine möglichst kleine, aber für die Korrektheit der Lösung ausreichend große Menge von zu prüfenden Maßen des Quaders ausgewählt werden.

1.1.1 Überprüfen eines gegebenen Quaders

Es seien nun eine Menge M von noch nicht abgeschnittenen Käsescheiben und ein Quader Q mit den drei ganzzahligen, positiven Seitenlängen x, y und z mit $x \leq y \leq z$ gegeben. Es muss dann unter den Käsescheiben eine ausgewählt werden, die zuerst abgeschnitten werden soll:

1. Enthält M noch eine Käsescheibe mit den Seitenlängen z und y, so wird diese vom Quader abgeschnitten.
2. Ist dies nicht der Fall und enthält M noch eine Käsescheibe mit den Seitenlängen z und x, so wird diese vom Quader abgeschnitten.
3. Ist dies ebenfalls nicht der Fall und enthält M noch eine Käsescheibe mit den Seitenlängen y und x, so wird diese vom Quader abgeschnitten.
4. Ist keiner der obigen drei Fälle eingetreten, so kann Q genau dann durch Zerschneiden in die in vorgegebenen Scheiben zerlegt werden, wenn die Menge M leer ist und der Quader das Volumen 0 hat.

Ist nicht der vierte Fall eingetreten, wird die Fallunterscheidung für den durch das Abschneiden einer Scheibe neu entstandenen Quader mit der durch das Entfernen eines Elements neu entstandenen Menge M' von Käsescheiben wiederholt. In einer separaten Liste wird die Reihenfolge der abgeschnittenen Scheiben gespeichert, weil diese ja eine korrekte Zerlegung des Quaders darstellt. Die Benennungen der Seitenlängen des Quaders Q müssen nach einem Schritt gegebenenfalls vertauscht werden, damit die Bedingung $x \leq y \leq z$ weiterhin gilt.
Abgebrochen wird dieser Algorithmus, wenn der Quader Q ein Volumen von 0 hat, die Menge M leer ist oder keine der vorhandenen Scheiben mehr einer Seitenfläche des Quaders entspricht. Dass der Körper Q kann genau dann aus der Menge von Käsescheiben M zusammengesetzt werden kann, wenn beim Abbruch dieses Algorithmus sowohl das Volumen des Quaders, als auch die Anzahl der Elemente in der Menge von Käsescheiben 0 ist, wird im Folgenden bewiesen:

Theorem 1. *Der oben beschriebene Algorithmus terminiert immer in einer endlichen Anzahl von Schritten und endet genau dann mit einem Quader mit Volumen 0 und einer leeren Menge von Käsescheiben, wenn Q sich aus den Scheiben in M vollständig zusammensetzen lässt.*

Beweis. In jedem Schritt wird eine Käsescheibe vom Quader abgeschnitten und damit genau eine seiner drei Seitenlängen um genau 1 dekrementiert - also erreicht nach einer endlichen Zahl von Schritten eine Seitenlänge den Wert 0. Dann hat aber auch das Volumen des Quaders, das sich als das Produkt der drei Seitenlängen berechnen lässt, den Wert 0.
Dies ist eine Abbruchbedingung des Algorithmus, womit dieser Abbruch immer nach einer endlichen Anzahl von Schritten geschieht.

Gibt es eine Zerlegung von Q in die Scheiben von M und gibt es in M eine Scheibe mit den Seitenlängen z und y, dann muss es korrekt sein, diese abzuschneiden - wegen $z, y \geq x$ ist dies die größte Seitenfläche des Quaders, die durch Abschneiden einer anderen Scheibe verkleinert werden würde. Dann könnte die Scheibe mit den Seitenlängen z und y aber niemals abgeschnitten werden, da sich die Seitenlängen des Quaders nur verringern können.
Analog folgt, dass bei Nichtexistenz einer Scheibe mit den Seitenlängen z und y eine mit den Seitenlängen z und x abgeschnitten werden sollte - wegen $z \geq x, y$ ist z nämlich die maximale Seitenlänge, die sich durch das Abschneiden einer Scheibe mit den Seitenlängen y und x verringern würde. Dann würde eine Scheibe mit den Seitenlängen z und x aber bei Abbruch des Algorithmus übrig bleiben.

Gibt es aber nur eine Scheibe mit den Seitenlängen x und y, so ist es eindeutig, dass diese abgeschnitten werden muss. Die nächste abzuschneidende Scheibe ist also immer eindeutig bestimmt - und wenn eine Zerlegung von Q in die Scheiben in M existiert, dann wird diese eindeutige Abfolge auch vom Algorithmus ausgeführt.
Ähnliche Aussagen gelten, wenn genau zwei der drei Seitenlängen gleich sind und daher nur zwei echt verschiedenen Seitenflächen abgeschnitten werden können. Sind alle drei Seitenlängen gleich, so sind der Quader würfelförmig und die Maße einer abzuschneidenden Scheibe offensichtlich eindeutig.

Wird beim Ausführen des Algorithmus aber ein Quader mit Volumen 0 mit einer dann leeren Menge von Käsescheiben erreicht, so existiert auch eine korrekte Zerlegung von Q in die Käsescheiben von M - schließlich wird in jedem Schritt eine Scheibe aus M von Q abgeschnitten, sodass am Ende ein leerer Quader existiert, und jede einzelne Scheibe wurde genau einmal verwendet.

Damit sind die Aussagen, dass der Algorithmus mit einer leeren Menge von Käsescheiben und einem leeren Quader terminiert bzw. dass sich Q in die Scheiben in M zerlegen lässt, äquivalent - es wurde nämlich gezeigt, dass jede dieser beiden Aussagen aus der jeweils anderen folgt. □

1.1.2 Auswählen zu überprüfender Quader

Für den in Kapitel 1.1.1 beschriebenen Algorithmus wird die Menge M der Käsescheiben vom Anwender vorgegeben, die zu überprüfenden Quader Q müssen aber vom Programm selbst bestimmt werden. Hierzu werden einige Beobachtungen gemacht, wobei nun vorausgesetzt wird, dass sich Q aus den Scheiben in M zusammensetzen lässt:

1. Das Volumen V des Quaders Q ist durch die Menge M der Scheiben fest vorgegeben - diese können ebenfalls als Quader betrachtet werden, bei denen die dritte Seitenlänge den Wert 1 hat. Die Summe ihrer Volumina entspricht aber dem Volumen des Quaders, weil sich die Scheiben in M ohne Überstände oder Lücken zu Q zusammensetzen lassen.

2. In der Menge der (nun als zweidimensional betrachteten) Käsescheiben M gibt es immer eine Scheibe, deren Flächenhinalt ein Teiler des Volumens V ist. In einer korrekten Zerlegung von Q in Käsescheiben gibt es schließlich eine, die als erstes abgeschnitten wird und deren Fläche somit eine Seitenfläche von Q ist. Das Volumen eines Quaders ist aber das Produkt des Flächeninhalts einer Seitenfläche und der Höhe auf dieser Seitenfläche, der Flächeninhalt einer jeden Seitenfläche ist also ein Teiler des Volumens V.

3. Da die Seitenlängen von Q durch das Abschneiden von Scheiben nur kleiner werden können, muss die größte Seitenlänge von Q mindestens so groß sein wie die größte Seitenlänge einer Scheibe.

4. Da die Seitenlängen von Q und daher auch die Seitenflächen von Q durch das Abschneiden von Scheiben nur kleiner werden können, muss die größte Seitenfläche von Q mindestens so groß sein wie die größte Scheibe.

Aus diesen Beobachtungen ergibt sich eine Methode zum Auffinden einer Menge von möglichen Quadern, die auf jeden Fall einen enthält, der sich in die Scheiben in M zerlegen lässt, insofern dies überhaupt möglich ist:
Zunächst wird durch alle Scheiben in M iteriert und dabei das Volumen V des Quaders, die maximale Seitenlänge einer Scheibe und die maximale Fläche einer Scheibe berechnet. Anschließend werden alle Scheiben ein zweites Mal durchgegangen.
Für jede Scheibe wird überprüft, ob ihre Fläche ein Teiler von V ist. Ist dies der Fall, ergibt sich ein Quader mit den Seitenlängen der Scheibe und der für das Erreichen des Volumens V nötigen Höhe. Hat dieser eine ausreichend große maximale Seitenlänge und Seitenfläche, so wird der in Kapitel 1.1.1 beschriebene Algorithmus für diesen ausgeführt.

1.1.3 Laufzeitüberlegungen

Definition 1. *Es sei im Folgenden $n \in \mathbb{N}$ die Anzahl der vorgegebenen Käsescheiben und $m \in \mathbb{N}$ die Anzahl dieser Käsescheiben nach Entfernen aller Duplikate.*

Für die Vorberechnungen zum Volumen des Gesamtquaders und den maximalen Seitenlängen und Flächeninhalten der Scheiben müssen alle n Scheiben durchgegangen werden. Die Laufzeit für die Vorberechnungen liegt also in $O(n)$.

Für das Überprüfen eines einzelnen Quaders muss maximal n-mal überprüft werden, welches die nächste Scheibe in M ist. Diese Scheibe muss anschließend aus M entfernt werden. Beide Operationen könenn in $O(\log n)$ ausgeführt werden, wenn M als Set oder ähnliche Datenstruktur gespeichert wird. Außerdem muss für jeden Test eines Quaders eine Kopie von M erstellt werden, da aus dieser Elemente gelöscht werden. Die Laufzeit dafür liegt in $O(\mid M \mid) = O(n)$.
Die Laufzeit für einen einzelnen Test liegt also in $O(n + n \log n)$. Es müssen aber maximal m Tests ausgeführt werden, weil Scheiben mit den gleichen Maßen ununterscheidbar sind. Die Laufzeit für das gesamte Verfahren liegt also in $O(n + m(n + n \log n)) = O(n + nm + nm \log n) = O(nm \log n)$.
Es muss allerdings hinzugefügt werden, dass diese Laufzeit eine ungenaue obere Schranke ist und in der Praxis selten erreicht werden kann: Dafür müssten nämlich die Flächeninhalte aller m unterschiedlichen Scheiben das Volumen V teilen (sonst würden nicht m Tests durchgeführt werden) und auch jeder Test zum Abschneiden fast aller Teile führen. Aufgrund der polynomiellen Laufzeit liegt das Problem aber auf jeden Fall in der Komplexitätsklasse P.

1.1.4 Laufzeitoptimierungen

In der Praxis lässt sich eine weitere Verbesserung der Laufzeit durchführen, die auf die theoretische Analyse in Kapitel 1.1.3 keinen Einfluss hat:
Problematisch ist hier die Tatsache, dass in jedem einzelnen Test in $O(n)$ eine Kopie von M erstellt wird, obwohl vielleicht nur ein Element aus M entfernt wird, bevor der Test abbricht. Um dies zu umgehen, wird als weitere Vorberechnung eine Kopie von M gespeichert. Im Folgenden wird beim Löschen eines Elements aus M das gelöschte Element in konstanter Zeit an eine Liste angehängt, in denen alle so gelöschten Elemente verwaltet werden.
Diese können nach Abbrechen des Tests jeweils in $O(\log n)$ in M eingefügt werden, da M als Set verwaltet wird. Dies geschieht genau dann, wenn die Zahl k der gelöschten Elemente so klein ist, dass diese k Operationen voraussichtlich weniger lange dauern als das Kopieren der gesamten Datenstruktur aus der in der Vorberechnung gespeicherten Version von M. Da die Laufzeit dafür in $O(n)$ liegt, wird der beschriebene Schritt also ausgeführt, wenn $k \log n < n$ gilt.
Außerdem lässt sich die Anzahl der Elemente in M minimal verringern, wenn nicht jedes einzelne Element, sondern nur alle unterschiedlichen Elemente in M gespeichert werden. In einer separaten Map wird für jedes Element gespeichert, wie oft es in M enthalten sein sollte. Eine solche Map kann Zugriffe und Änderungen ebenfalls in $O(\log n)$ ermöglichen, weshalb sich die theoretische Laufzeitkomplexität nicht ändert.
Das tatsächliche Programm kann aber geringfügig schneller werden, da so nur $m \leq n$ Elemente gespeichert werden.

1.1.5 Eindeutigkeit einer Lösung

Theorem 2. *Das Zusammenfügen einer Menge von Scheiben zu einem vollständigen Käsequader ist nicht notwendigerweise eindeutig.*

Beweis. Die Aussage wird durch Angabe eines entsprechenden Beispiels gezeigt. Dafür habe eine von drei Scheiben die Seitenlängen 1 und 2 und zwei weitere seien quadratisch mit einer Seitenlänge von 1.
Diese drei Scheiben lassen sich zu einem Quader mit den Seitenlängen 1, 1 und 4 zusammensetzen, von dem dann die beiden quadratischen Scheiben abgeschnitten werden. Die drei Scheiben lassen sich aber auch zu einem Quader mit den Seitenlängen 1, 2 und 2 zusammenfügen: Von diesem wird dann zuerst die nicht quadratische Scheibe abgeschnitten, bevor das verbleibende Käsestück in die beiden gleich großen Quadrate zerlegt wird. □

1.2 Aufgabenteil b: Eine fehlende Scheibe

Die oben beschriebene Lösung kann für weitere Fragestellungen erweitert werden. In diesem Kapitel wird der Fall behandelt, dass Antje vor lauter Hunger schon eine (und zwar genau eine) Scheibe aufgegessen hat, die dann in der Menge M nicht vorkommt, für das Zusammensetzen eines vollständigen Quaders Q aber wahrscheinlich benötigt wird.

1.2.1 Alternative Lösung für Aufgabenteil a

Es wird zunächst ein alternativer Algorithmus für Aufgabenteil a angegeben, der zwar eine schlechtere praktische Laufzeit hat, dafür aber für die hier untersuchte Variante des Problems erweitert werden kann.

Dafür wird erst von der zuletzt übrig bleibenden Scheibe ausgegangen, ihre Maße ergeben einen Quader mit Höhe 1. An diesen Quader werden dann nach und nach weitere Scheiben angefügt, was den in Kapitel 1.1.1 beschriebenen Algorithmus quasi „umkehrt“.

Es sei also eine Menge M von noch nicht angefügten Käsescheiben und ein Quader Q mit den drei ganzzahligen, positiven Seitenlängen x, y und z mit $x \leq y \leq z$ gegeben. Es muss dann unter den Käsescheiben eine ausgewählt werden, die als nächstes angefügt werden soll:

1. Enthält M noch eine Käsescheibe mit den Seitenlängen y und x, so wird diese an den Quader angefügt.
2. Ist dies nicht der Fall und enthält M noch eine Käsescheibe mit den Seitenlängen z und x, so wird diese an den Quader angefügt.
3. Ist dies ebenfalls nicht der Fall und enthält M noch eine Käsescheibe mit den Seitenlängen z und y, so wird diese an den Quader angefügt.
4. Ist keiner der obigen drei Fälle eingetreten, so kann Q genau dann durch Zerschneiden in die in vorgegebenen Scheiben zerlegt werden, wenn die Menge M leer ist.

Danach wird die Fallunterscheidung für den durch das Abschneiden einer Scheibe neu entstandenen Quader mit der durch das Entfernen eines Elements neu entstandenen Menge von Käsescheiben wiederholt. In einer separaten Liste kann die Reihenfolge der angefügten Scheiben gespeichert werden, weil diese ja eine korrekte Zerlegung des Quaders darstellt.
Der Korrektheitsbeweis dieses Algorithmus ist analog zum dem des in Aufgabenteil a eingesetzten und auch die theoretische Laufzeitkomplexität ist gleich. In Aufgabenteil a wird allerdings die andere Methode eingesetzt, weil dort nur Quader überprüft werden, die das korrrekte Volumen und eine vorhandene Scheibe als Seitenfläche haben.
Hier muss allerdings jede vorhandene Scheibe (bis auf Duplikate) als zuletzt übrig bleibende Scheibe getestet werden.

1.2.2 Erweiterung des neuen Algorithmus

Der in Kapitel 1.2.1 beschriebene Algorithmus lässt sich nun für den Fall erweitern, dass genau eine Scheibe fehlt.
Es sei dafür nun ein vorgegebener Quader Q und die Menge M der bekannten, noch hinzuzufügenden Scheiben gegeben. Die Seitenlängen von Q seien weiterhin x, y, z mit $x \leq y \leq z$, außerdem sei bekannt, ob in der Zerlegung von Q schon eine nicht in M vorkommende Scheibe verwendet wurde. Es wird dann der folgende Algorithmus schrittweise durchgeführt:

1. Enthält M noch eine Käsescheibe mit den Seitenlängen y und x, so wird diese an den Quader angefügt und der Schritt mit dem so entstandenen Quader Q' und der Menge M' nach Entfernen der Scheibe x, y entstandenen Menge M' wiederholt (Dies ist wie im bekannten Algorithmus).
2. Ist keine Scheibe mit den Seitenlängen x und y in M enthalten, während noch keine Scheibe, die nicht Element von M ist, an Q angefügt wurden, so wird eine Scheibe mit den Seitenlängen x und y als von Antje gegessene Scheibe an Q angefügt.
   Mit dem so entstandenen Quader und der Menge M wird der Schritt wiederholt.
3. Die ersten beiden Fälle werden ebenfalls mit den anderen Seitenflächen ausgeführt.
4. Ist M leer und wurde schon eine fehlende Scheibe an Q angefügt, so wurde eine Möglichkeit zur Zerlegung von Q gefunden.
5. Wurde schon eine von Antje gegessene Scheibe an Q angefügt, während M nicht leer ist und keine Scheibe aus M an Q angefügt werden kann, so wird die Ausführung dieses Schritts abgebrochen, weil sich so keine Zerlegung von Q finden kann.

Der Korrektheitsbeweis dieses Algorithmus ist wiederum analog zu dem in Kapitel 1.1.1 (tatsächlich werden hier theoretisch alle Möglichkeiten der Zerlegung durchgegangen. Da aber das Abschneiden größerer Scheiben priorisiert wird, entsteht trotzdem die vorherige Reihenfolge).
Es ergibt sich also eine Art „Tiefensuche“, bei der alle Möglichkeiten zur Zerlegung des Quaders mit Hinzufügen einer unbekannten Scheibe durchgerechnet werden. Ein Frame der Tiefensuche ist charakterisiert

durch die Menge der verbleibenden Scheiben und den Quader Q.
Diese Tiefensuche kann laufzeitsparend umgesetzt werden, indem ein Quader und eine Menge M verwaltet werden, bei denen die Scheiben beim Bewegen im Tiefensuchbaum einzeln hinzugefügt bzw. entfernt werden.

1.2.3 Laufzeitüberlegungen

Die Berechnung muss im schlechtesten Fall für m zuletzt übrig gebliebene Scheiben durchgeführt werden. Die Verwaltung der Menge M benötigt dabei für das Hinzufügen bzw. Entfernen einer Scheibe eine Laufzeit von jeweils $O(\log n)$, während das Abschneiden bzw. Anfügen einer Scheibe an den Quader in konstanter Zeit verläuft (diese Betrachtungen sind analog zu denen in 1.1.3. Auch die in Aufgabenteil a beschriebenen Laufzeitoptimierungen können in dem Maße erneut angewandt werden, wie sie das Verwalten von Duplikaten von Scheiben betreffen).
Allerdings ergeben sich in einem Schritt möglicherweise mehrere Möglichkeiten, eine nächste Scheibe anzufügen. Dies ist nur der Fall, wenn in einem vorherigen Schritt noch keine unbekannte Scheibe hinzugefügt wurde, und es gibt dann auch nur drei Möglichkeiten für die neue Scheibe. Von einem Tiefensuchframe ohne vorherige unbekannte Scheibe aus werden also bis zu vier (statt vorher einem) Tests folgender Quader durchgeführt: Drei mit unbekannten Scheiben, einer mit einer vorliegenden Scheibe. Danach verläuft dieser Test allerdings wie in Kapitel 1.2.1.
Die Laufzeit von vorher $O(nm \log n)$ sollte sich also nur um einen konstanten Faktor erhöhen.

1.2.4 Eindeutigkeit einer Lösung

Die Eindeutigkeit einer Lösung ist auch in dieser Erweiterung keinesfalls gegeben. Hierfür genügt schon die Angabe desselben Beispiels wie in Kapitel 1.1.5 - schließlich könnte an dieses Beispel ja einfach eine neue Seitenfläche als unbekannte Scheibe angefügt werden.
Ein „stärkeres“ Beispiel für die Uneindeutigkeit der Lösung gibt aber die Betrachtung er Beispiele in Kapitel 3.

1.3 Aufgabenteil b: Mehrere fehlende Scheiben

Antje könnte nicht nur eine, sondern auch mehrere Scheiben gegessen haben - schließlich hatte sie schon vor dem Telefonat Appetit. Laut den Eingabedateien auf der BwInf-Webseite dauerte dieses aber so lange, dass sie Millionen von Käsescheiben mit präziser Millimeterdicke abschneiden konnte.
Dieser Fall soll hier untersucht werden, allerdings mit der Vereinfachung, dass die Maße des zu erreichenden Quaders Q schon vorgegeben sind.
Diese Angabe ist sinnvoll, weil keine Rückschlüsse auf das Volumen des Quaders oder die Anzahl der Käsescheiben gezogen werden können.
Andererseits ist es realistisch, dass Antje die Maße des ursprünglichen Quaders kennt. Ein wertvolles Indiz hierfür ist die Tatsache, dass Antjes Käse überhaupt als Quader vorliegt. Käselaibe haben in der Regel nämlich eine runde Form, was einerseits auf einer in der Anfgangszeit der Käseherstellung begründeten Tradition, andererseits in diversen Vorteilen dieser Form bei Käseherstellung und Käsepflege begründet ist [1].
Wird ein solcher runder Laib nun z.B. an der Käsetheke oder im Hofladen geschnitten und verkauft, so resultieren daraus keilförmige Käsestücke. Es ist aber auch nicht vorstellbar, warum Antje einen keilförmigen Käse vor ihrem Telefonat zu einem Quader zugeschnitten haben sollte.
Daraus lässt sich folgern, dass Antje den Käse in abgepackter Form erstanden hat (Für die Fabrikanten von abgepacktem Käse, der in der Regel in Massenproduktion hergestellt wird, hat die Quaderform Vorteile bei Lagerung und Transport von tonnenweise Käse). Die Verpackung liefert aber Informationen über die Maße des Käsequaders, mit denen Antje ihr Programm füttern kann.

1.3.1 Lösung des Problems

Das Problem wird durch eine Kombination der beiden vorherigen Ansätze gelöst: Dazu wird zunächst eine „Tiefensuche“ wie in Kapitel 1.2.2 durchgeführt. Ein Frame ist dann charakterisiert durch einen aktuellen Quader W und die Menge der noch abzuschneidenden Scheiben M.
Die Tiefensuche geht aber nicht von einer einzelnen Scheibe, sondern wie der Algorithmus in Kapitel 1.1.1 vom vorgegebenen Quader Q aus, von dem nach und nach Scheiben abgeschnitten werden, wobei

[1] https://www.tilsiter.ch/blog/warum-sind-kaeselaibe-rund zuletzt abgerufen am 06.04.2023

die kleineren Quader W erreicht werden.
Wie bei der Tiefensuche im vorherigen Kapitel können unbekannte Scheiben abgeschnitten werden. Diese sind aber nicht beschränkt durch eine vorgegebene Anzahl, sondern durch die Tatsache, dass die noch unbekannten Scheiben ein Volumen haben müssen, dass maximal so groß ist wie die Differenz zwischen dem Volumen von W und der Summen aller Volumina von Scheiben in M.
Die Korrektheit dieses Algorithmus ergibt sich analog zu den Lösungen in den vorherigen Kapiteln.

1.3.2 Laufzeitüberlegungen

Die Laufzeit dieses Verfahrens lässt sich nur durch extrem ungenaue, obere Schranken angeben - es könnten für einen Quader mit den Seitenlägen q_1, q_2, q_3 etwa alle Kombinationen von Seitenlängen mit jeweils allen Kombinationen von verbleibenden Scheiben durchgegangen werden, und das mit einem logarithmischen Faktor für das Verwalten der Scheiben. Es verbliebe eine Laufzeit mit einer Laufzeitomplexität in $O(q_1 \cdot q_2 \cdot q_3 \cdot 2^{|M|} \cdot \log |M|)$.
Auch wenn diese obere Schranke enorm ungenau ist, ist die Laufzeit dieses Verfahren deutlich schlechter als die der vorhergehenden Algorithmen. Der Grund dafür ist die Tatsache, dass die Tiefensuche viel mehr Fälle untersuchen muss, weil mehr als eine Scheibe hinzugefügt werden kann. Außerdem liegt die Laufzeitkomplexität des Programms nicht mehr in der Klasse der polynomiellen Laufzeitkomplexitäten.

2 Umsetzung

Die Lösungsidee wird in C++ umgesetzt.

2.1 Aufgabenteil a: Ein vollständiger Käsequader

2.1.1 Speichern der Daten und Ausführung des Algorithmus

Die Scheiben werden als Paare von Long Integern `pair<long long, long long>` gespeichert. Die beiden Einträge dieser Paare sollen dabei immer aufsteigend geordnet sein.
Die für die Scheiben verwendeten Datenstrukturen sind dann für das Speichern der unterschiedlichen Scheiben ein Container des Typs `set<pair<long long, long long>>` und ein Container bzw. eine Map des Typs `map<pair<long long, long long>, long long>`, in dem für jedes Paar im Set festgehalten wird, wie viele dieser Scheiben noch in M enthalten sind.
Von diesen Strukturen gibt es im Programm zwei Kopien, von denen eine diejenige ist, auf der der Algorithmus operiert, und eine, die zum Wiederherstellen der Daten nach dem Test eines Quaders kopiert werden kann.

Nach den nötigen Vorberechnungen, die in einer `for`-Schleife durchgeführt werden, werden die Käsescheiben in einer weiteren `for`-Schleife durchgegangen. Für die Quader, für die die in Kapitel 1.1.2 beschriebenen Bedingungen gelten, wird der in Kapitel 1.1.1 beschriebene Test durchgeführt.
Dafür werden in einer `while`-Schleife die nötigen Scheiben abgeschnitten, an einen Container des Typs `vector<pair<long long>>` angehängt und aus dem Set der noch nicht abgeschnittenen Scheiben entfernt. Ist die Menge der noch nicht abgeschnittenen Scheiben leer oder lässt sich keine weitere Scheibe abschneiden, so wird die `while`-Schleife abgebrochen.
Wurden dann alle Scheiben, abgeschnitten, so wird die Lösung mithilfe der Liste der abgeschnittenen Scheiben ausgegeben. Ist dies nicht der Fall, so werden die Elemente der Liste der gelöschten Scheiben entweder wieder in das Set für die Menge M eingefügt oder dieses Set mit der Sicherheitskopie überschrieben.

2.1.2 Verwaltung des Käsequaders

Um den Käsequader während der Tests zu verwalten, wird eine eigene Klasse namens `cuboid` definiert. Diese hat die Seitenlängen des Quaders als private Membervariablen.
Unter den öffentlichen Memberfunktionen gibt es eine, die bei Übergabe des Sets der noch nicht abgeschnittenen Scheiben die größte auswählt, die auch Seitenfläche des Quaders ist und die Seitenlängen des Quaders aktualisiert.

2.2 Aufgabenteil b: Eine fehlende Scheibe

Die Verwaltung des Käsequaders erfolgt mit minimalen Anpassungen genau wie in Kapitel 2.1. Neu ist nur die Umsetzung der „Tiefensuche“, die über Möglichkeiten von Quadern und Scheibenmengen iteriert. Eine funktionale, rekursive Tiefensuche wäre allerdings nicht praktisch umsetzbar - bei großen Beispielen mit mehr als 10^6 Käsescheiben könnte dies in einen Stack Overflow münden. Daher wird diese Tiefensuche mithilfe eines eigenen Stacks durchgeführt, der Äquivalente von Tiefensuchframes verwaltet. Diese enthalten jeweils eine Angabe über die Tatsache, ob schon eine unbekannte Scheibe eingefügt wurde, welches die letzte angefügte Scheibe war und wie viele der folgenden Möglichkeiten schon ausprobiert wurden.
Der Käsequader und die Menge M der Käsescheiben werden außerhalb dieser Tiefensuche verwaltet und durch Hinzufügen und Entfernen von Scheiben aktuell gehalten. Ihre Verwaltung erfolgt ebenfalls analog zu der in Kapitel 2.1.
Zusammenfassend werden also in einer `for`-Schleife alle Möglichkeiten für die zuletzt übrig bleibende Scheibe durchgegangen, bevor eine jede solche Scheibe mithilfe einer durch einen `stack` und eine `while`-Schleife umgesetzte Tiefensuche untersucht werden.

2.3 Aufgabenteil b: Mehrere fehlende Scheiben

Wie bereits in Kapitel 1.3.1 beschrieben, ist die Lösung dieses Problems eine Kombination der beiden vorherigen Ansätze. Dies spiegelt sich auch in der Umsetzung wider, die lediglich die vorherigen Ansätze miteinander vereint und um das Speichern des verbleibenden Volumens von Scheiben in M ergänzt.

3 Beispiele

3.1 Aufgabenteil a: Ein vollständiger Käsequader

Ausgabe für die Eingabedatei kaese1.txt:

Die angegebene Menge von Scheiben laesst sich zu einem Quader mit den Seitenlaengen 6, 6 und 6 zusammensetzen.
Dabei muessen die Scheiben in folgender Reihenfolge vom Quader abgeschnitten werden:
6 x 6
6 x 6
4 x 6
4 x 6
4 x 6
3 x 6
3 x 3
3 x 3
3 x 4
2 x 4
2 x 4
2 x 4

Ausgabe für die Eingabedatei kaese2.txt:

Die angegebene Menge von Scheiben laesst sich zu einem Quader mit den Seitenlaengen 2, 1000 und 1000 zusammensetzen.
Dabei muessen die Scheiben in folgender Reihenfolge vom Quader abgeschnitten werden:
2 x 1000
2 x 1000
2 x 998
998 x 999
998 x 999

Ausgabe für die Eingabedatei kaese3.txt:

Die angegebene Menge von Scheiben laesst sich zu einem Quader mit den Seitenlaengen 1000, 1000 und 10 zusammensetzen.

Dabei muessen die Scheiben in folgender Reihenfolge vom Quader abgeschnitten werden:
1000 x 1000
1000 x 1000
1000 x 1000
7 x 1000
999 x 1000
6 x 1000
998 x 1000
5 x 998
998 x 999
4 x 999
4 x 997
4 x 998
4 x 998
4 x 995
995 x 997
995 x 997
2 x 997
2 x 994
2 x 996
2 x 993
2 x 995
992 x 995
992 x 995

Ausgabe für die Eingabedatei kaese4.txt:

Die angegebene Menge von Scheiben laesst sich zu einem Quader mit den Seitenlaengen 210, 210 und 210 zusammensetzen.
Dabei muessen die Scheiben in folgender Reihenfolge vom Quader abgeschnitten werden:
210 x 210
(Die folgenden 549 Zeilen wurden aus Platzgründen weggelassen)

Ausgabe für die Eingabedatei kaese5.txt:

Die angegebene Menge von Scheiben laesst sich zu einem Quader mit den Seitenlaengen 2730, 3570 und 2310 zusammensetzen.
Dabei muessen die Scheiben in folgender Reihenfolge vom Quader abgeschnitten werden:
2730 x 3570
(Die folgenden 6847 Zeilen wurden aus Platzgründen weggelassen)

Ausgabe für die Eingabedatei kaese6.txt:

Die angegebene Menge von Scheiben laesst sich zu einem Quader mit den Seitenlaengen 39270, 510510 und 30030 zusammensetzen.
Dabei muessen die Scheiben in folgender Reihenfolge vom Quader abgeschnitten werden:
39270 x 510510
(Die folgenden 90329 Zeilen wurden aus Platzgründen weggelassen)

Ausgabe für die Eingabedatei kaese7.txt:

Die angegebene Menge von Scheiben laesst sich zu einem Quader mit den Seitenlaengen 510510, 510510 und 510510 zusammensetzen.
Dabei muessen die Scheiben in folgender Reihenfolge vom Quader abgeschnitten werden:
510510 x 510510
(Die folgenden 1529902 Zeilen wurden aus Platzgründen weggelassen)

Laufzeit:

Für jede der betrachteten Eingabedateien benötigte das Programm auf einem durchschnittlichen Computer weniger als 30 Sekunden Laufzeit.

3.2 Aufgabenteil b: Eine fehlende Scheibe

Als Eingabedateien für diese Aufgabe wurden die Eingabedateien für Aufgabenteil a verwendet, in jeder dieser Dateien aber (willkürlicherweise) die dritte Scheibe entfernt.

Ausgabe für die Eingabedatei missing1.txt:

Der gesuchte Quader hat die Seitenlaengen 6, 6 und 6.
Die Scheiben muessen dabei in der folgenden Reihenfolge abgeschnitten werden:

6 x 6
6 x 6
4 x 6
4 x 6
4 x 6
3 x 6
3 x 3
3 x 3
3 x 4 (Diese Scheibe hatte Antje schon vorher gegessen)
2 x 4
2 x 4
2 x 4

Ausgabe für die Eingabedatei missing2.txt:

Der gesuchte Quader hat die Seitenlaengen 2, 1000 und 1000.
Die Scheiben muessen dabei in der folgenden Reihenfolge abgeschnitten werden:

2 x 1000
2 x 1000
2 x 998
998 x 999
998 x 999 (Diese Scheibe hatte Antje schon vorher gegessen)

Ausgabe für die Eingabedatei missing3.txt:

Der gesuchte Quader hat die Seitenlaengen 9, 1000 und 1001.
Die Scheiben muessen dabei in der folgenden Reihenfolge abgeschnitten werden:

9 x 1000 (Diese Scheibe hatte Antje schon vorher gegessen)
1000 x 1000
1000 x 1000
7 x 1000
999 x 1000
6 x 1000
998 x 1000
5 x 998
998 x 999
4 x 999
4 x 997
4 x 998
4 x 998
4 x 995
995 x 997

995 x 997
2 x 997
2 x 994
2 x 996
2 x 993
2 x 995
992 x 995
992 x 995

Ausgabe für die Eingabedatei missing4.txt:

Der gesuchte Quader hat die Seitenlaengen 210, 210 und 210.
Die Scheiben muessen dabei in der folgenden Reihenfolge abgeschnitten werden:

210 x 210
(Die folgenden 549 Zeilen wurden aus Platzgründen weggelassen)

Ausgabe für die Eingabedatei missing5.txt:

Der gesuchte Quader hat die Seitenlaengen 2310, 2730 und 3570.
Die Scheiben muessen dabei in der folgenden Reihenfolge abgeschnitten werden:

2730 x 3570
(Die folgenden 6847 Zeilen wurden aus Platzgründen weggelassen)

Ausgabe für die Eingabedatei missing6.txt:

Der gesuchte Quader hat die Seitenlaengen 30030, 39270 und 510510.
Die Scheiben muessen dabei in der folgenden Reihenfolge abgeschnitten werden:

39270 x 510510
(Die folgenden 6847 Zeilen wurden aus Platzgründen weggelassen)
(Die folgenden 90329 Zeilen wurden aus Platzgründen weggelassen)

Ausgabe für die Eingabedatei missing7.txt:

Der gesuchte Quader hat die Seitenlaengen 510510, 510510 und 510510.
Die Scheiben muessen dabei in der folgenden Reihenfolge abgeschnitten werden:

510510 x 510510
(Die folgenden 1529902 Zeilen wurden aus Platzgründen weggelassen)

Laufzeit und Eindeutigkeit:

Für jede der betrachteten Eingabedateien benötigte das Programm auf einem durchschnittlichen Computer weniger als 45 Sekunden Laufzeit. Interesant ist hier die Beobachtung, dass das Programm für die dritte Beispieleingabe einen Quader findet, der anders ist als die Zerlegung aus Aufgabenteil a (Und die fehlende Scheibe auch durch eine andere ersetzt hat).

3.3 Aufgabenteil b: Mehrere fehlende Scheiben

Konstruktion der Eingabedateien:

Um die Ergebnisse des Programms sinnvoll darzustellen, wird die Eingabedatei kaese1.txt von der BwInf-Website modifiziert. Gespeichert werden diese in den Eingabedateien manyMissing1.txt bis manyMissing6.txt.
Im Vergleich zur Ausgangsdatei fehlen 0, 1, 3, 6, 9 oder 12 der ursprünlich 12 Scheiben. Die übergebenen Maße entsprechen denen, die die vorherigen Programme für diese Eingabedatei finden (also ist der Quader

ein Würfel mit der Seitenlänge 6).

Ausgabe für die Eingabedatei manyMissing1.txt:

Die Scheiben werden in der folgenden Reihenfolge vom Quader abgeschnitten:

6 x 6
6 x 6
4 x 6
4 x 6
4 x 6
3 x 6
3 x 3
3 x 3
3 x 4
2 x 4
2 x 4
2 x 4

Ausgabe für die Eingabedatei manyMissing2.txt:

Die Scheiben werden in der folgenden Reihenfolge vom Quader abgeschnitten:

6 x 6
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
4 x 6
4 x 6
4 x 6
3 x 6
3 x 3
3 x 3
3 x 4
2 x 4
2 x 4
2 x 4

Ausgabe für die Eingabedatei manyMissing3.txt:

Die Scheiben werden in der folgenden Reihenfolge vom Quader abgeschnitten:

6 x 6
6 x 6
4 x 6
5 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
3 x 6
4 x 6
2 x 4
2 x 4
2 x 4 (Diese Scheibe hatte Antje schon aufgegessen)
3 x 4
1 x 3 (Diese Scheibe hatte Antje schon aufgegessen)
3 x 3

Ausgabe für die Eingabedatei manyMissing4.txt:

Die Scheiben werden in der folgenden Reihenfolge vom Quader abgeschnitten:

6 x 6

6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
3 x 6
3 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
4 x 6
2 x 4
2 x 4
4 x 4 (Diese Scheibe hatte Antje schon aufgegessen)
1 x 4 (Diese Scheibe hatte Antje schon aufgegessen)
1 x 3 (Diese Scheibe hatte Antje schon aufgegessen)
3 x 3

Ausgabe für die Eingabedatei manyMissing5.txt:

Die Scheiben werden in der folgenden Reihenfolge vom Quader abgeschnitten:

6 x 6
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
2 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
2 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
4 x 6
1 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
1 x 3 (Diese Scheibe hatte Antje schon aufgegessen)
1 x 3 (Diese Scheibe hatte Antje schon aufgegessen)
3 x 4

Ausgabe für die Eingabedatei manyMissing6.txt:

Die Scheiben werden in der folgenden Reihenfolge vom Quader abgeschnitten:

6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)
6 x 6 (Diese Scheibe hatte Antje schon aufgegessen)

Laufzeit:

Für die angegebenen (kleinen) Testfälle liegt die Laufzeit des Programms auf einem durchschnittlichen Computer unter einer Sekunde. Für Fälle mit größeren Seitenlängen des Quaders (diese sind eher ausschlaggebend als die Anzahl der vorgegebenen Scheiben) wird das Ergebnis in der Regel aber nicht mehr nach einer praktikablen Zeit ausgegeben.

4 Quellcode

4.1 Aufgabenteil a: Ein vollständiger Käsequader

Das verwendete Template:

```
#include <bits/stdc++.h>
#define ll long long
#define pll pair<ll, ll>
#define NOPAIR (pll) make_pair(-1,-1)
using namespace std;
```

Die verwendeten Variablen und ihre Initialisierung zu Beginn der `main()`-Funktion:

```
//Variablen zum Speichern der Eingabe:
    int n; //Anzahl der Scheiben
    int m; //Anzahl der Scheiben ohne Duplikate
    set<pll> slicesSafe; //Unveraenderliche Kopie der Menge von Scheiben
    map<pll, ll> countersSafe; //Unveraenderliche Kopie der Zaehler fuer Scheiben
    set<pll> slicesChange; //Veraenderliche Kopie der Menge von Scheiben
    map<pll, ll> countersChange; //Veraenderliche Kopie der Zaehler fuer Scheiben
    //In jedem Paar (a, b) ganzer Zahlen soll dabei b >= a gelten!
    ofstream fout; //Ausgabedatei

    readInput(n, m, slicesSafe, countersSafe, slicesChange, countersChange, fout);
    //Lesen der Eingabe

    //Variablen zum Speichern noetiger Vorberechnungen:
    ll volume; //Volumen des Quaders
    ll longestEdge; //Groesste Seitenlaenge einer Scheibe
    ll largestArea; //Groesste Flaeche einer Scheibe

    precompute(volume, longestEdge, largestArea, slicesSafe, countersSafe);
    //Ausfuehren der Vorberechnungen
```

Die Definition der Klasse für die Verwaltung des Quaders:

```
class cuboid //Klasse fuer die Quader
{
    private:
        ll x, y, z; //Seitenlaengen

    public:
        cuboid(ll w, ll b, ll h); //Konstruktor, nimmt Seitenlaengen entgegen
        void orderEdges(); //Vertausche Seitenlaengen zu x <= y <= z
        pll cutBiggestSlice(set<pll>& slices); //Schneidet groesste Seitenflaeche ab,
        //die auch in der Liste ist. Liefert abgeschnitte Flaeche oder NOPAIR zurueck
        ll longestEdge(); //Liefert laengste Seitenlaenge zurueck
        ll largestArea(); //Liefert maximalen Flaecheninhalt einer Seitenflaeche zurueck
};
```

Die Implementierung der Funktion `cutBiggestSlice()`:

```
pll cuboid::cutBiggestSlice(set<pll>& slices)
{
    if(slices.count({y, z}) > 0)
    {
        x--;
        orderEdges();
        return {y, z};
    }
    if(slices.count({x, z}) > 0)
    {
        y--;
        orderEdges();
        return {x, z};
    }
    if(slices.count({x, y}) > 0)
    {
        z--;
        orderEdges();
        return {x, y};
    }
    return NOPAIR;
}
```

Die Durchführung der Tests:

```
//Durchgehen aller unterschiedlicher Kanten:
for(pll slice : slicesSafe)
{
    ll area = slice.first*slice.second;
    if(volume%area == 0)
    {
```

```
        //Die Flaeche teilt das Volumen, slice kann eine Seitenflaeche sein!
        cuboid q(slice.first, slice.second, volume/area); //Initialisieren des Quaders Q
        vector<pll> deleted(0); //Abgeschnitte Scheiben
        if(q.largestArea() >= largestArea && q.longestEdge() >= longestEdge)
        {
            //Q ist gross genug, der Test wird durchgefuehrt
            while(!slicesChange.empty())
            {
                //Abschneiden einer weiteren Scheibe:
                pll cutted = q.cutBiggestSlice(slicesChange);
                if(cutted == NOPAIR)
                {
                    //Es konnte keine weitere Scheibe abgeschnitten werden
                    break;
                }
                deleted.push_back(cutted); //Hinzufuegen zur Liste der geloeschten Scheiben
                countersChange[cutted]--; //Entfernen der Scheibe
                if(countersChange[cutted] == 0)
                {
                    slicesChange.erase(cutted);
                }
            }

            if(slicesChange.empty())
            {
                //Es wurde eine gueltige Zerlegung gefunden
                //Ausgabe
                return 0;
            }
            else
            {
                //Es wurde keine gueltige Zerlegung gefunden:
                int k = deleted.size();
                if(k*log2(n) <= m)
                {
                    //Es ist schneller, die Elemente einzeln in die Liste einzufuegen
                    for(pll p : deleted)
                    {
                        if(countersChange[p] == 0)
                        {
                            slicesChange.insert(p);
                        }
                        countersChange[p]++;
                    }
                }
                else
                {
                    //Es ist schneller, die gesamte Liste zu kopieren
                    slicesChange = slicesSafe;
                    countersChange = countersSafe;
                }
            }
        }
    }
}

//Es wurde keine gueltige Zerlegung gefunden!
//Ausgabe
return 0;
```

4.2 Aufgabenteil b: Eine fehlende Scheibe

Das verwendete Template:

```
#include <bits/stdc++.h>
#define ll long long
#define pll pair<ll, ll>
#define tlll tuple<ll, ll, ll>
#define tblp tuple<bool, ll, pll>
using namespace std;
```

Die Verwaltung des Käsequaders wird hier nicht gezeigt, da sie der in Kapitel 4.1 sehr ähnlich ist. Im Folgenden eine Funktion zum Verwalten der Menge M:

```
bool available(pll slice, set<pll>& slices, map<pll, ll>& counters)
{
    //Kann die uebergebene Scheibe noch verwendet werden?
    if(slices.count(slice) > 0 && counters[slice] > 0)
    {
        return true;
    }
    return false;
}
```

Umsetzung der „Tiefensuche“ in der `main()`-Funktion:

```
stack<pll> order; //Reihenfolge der abgeschnittenen Scheiben
for(pll s : slices) //Scheibe, mit der gestartet wird
{
     cuboid q(s.first, s.second, 1);
     counters[s]--;
     order.push(s);

    //dfs(q, slices, counters, false, fout, order, n, countersSafe);
    stack<tblp> dfs;
    /*Stapel zum Verwalten der Tiefensuchen-Frames (speichert jeweils,
    ob schon die fehlende Scheibe hinzugefuegt wurde,
    in queries die Anzahl der untersuchten Folgemoeglichkeiten
    und zuletzt hinzugefuegte Scheibe)*/
    dfs.push({false, 0, {-1, -1}});
    while(dfs.size())
    {
        bool missing = get<0>(dfs.top());
        ll queries = get<1>(dfs.top());
        pll lastSlice = get<2>(dfs.top());
        dfs.pop();
        if(queries < 6)
        {
            dfs.push({missing, queries+1, lastSlice});
        }

        if((ll) order.size() == n+1) //Es wurden schon ausreichend viele Scheiben hinzugefuegt
        {
            //Ausgabe!

            exit(0);
        }

        //Seitenflaechen des Quaders:
        pll s1 = q.sliceK(1);
        pll s2 = q.sliceK(2);
        pll s3 = q.sliceK(3);

        //Versucht nacheinander, Scheiben abzuschneiden (in der Reihenfolge des Algorithmus)
        if(queries == 0 && available(s1, slices, counters))
        {
            //Kleinste Seitenflaeche muss hinzugefuegt werden
            q.addSlice(1);
            counters[s1]--;
            order.push(s1);
            dfs.push({missing, 0, s1});
        }
        else if(queries == 1 && !missing) //Kleinste Seitenflaeche als die von Antje gegessene
        {
            q.addSlice(1);
            counters[s1]--;
            order.push(s1);
            dfs.push({true, 0, s1});
        }

    //Die anderen Seitenflaechen folgen analog

        if(queries == 6) //Tiefensuche ueber diese Moeglichkeit ist beendet
        {
```

```
            q.cutSlice(lastSlice);
            counters[lastSlice]++;
            order.pop();
        }
    }

    counters[s]++;
}
```

4.3 Aufgabenteil b: Mehrere fehlende Scheiben

Die `main()`-Funktion

```
int main()
{
    //Variablen zum Speichern der Eingabe:
    int n; //Anzahl der Scheiben
    int m; //Anzahl der Scheiben ohne Duplikate
    set<pll> slices; //Veraenderliche Kopie der Menge von Scheiben
    map<pll, ll> counters; //Veraenderliche Kopie der Zaehler fuer Scheiben
    map<pll, ll> countersSafe; //Unveraenderliche Kopie der Zaehler fuer Scheiben
    //In jedem Paar (a, b) ganzer Zahlen soll dabei b >= a gelten!
    ofstream fout; //Ausgabedatei
    cuboid q(0, 0, 0); //Der Quader q
    ll sliceVolume; //Volumen aller verfuegbaren Scheiben

    readInput(n, m, slices, counters, countersSafe, fout, q, sliceVolume); //Lesen der Eingabe

    stack<pll> order; //Reihenfolge der abgeschnittenen Scheiben
    stack<tlpb> dfs;
    /*Stapel zum Verwalten der Tiefensuch-Frames, verwaltet die Anzahl
    der Folgemoeglichkeiten und die zuletzt hinzugefuegte Scheibe.
    Ausserdem wird angegeben, ob diese letzte Scheibe schon von Antje gegessen wurde*/
    dfs.push({0, NOPAIR, true}); //Zu Beginn wurde noch keine Scheibe hinzugefuegt
    order.push(NOPAIR);

    while(dfs.size())
    {
        ll queries = get<0>(dfs.top());
        pll lastSlice = get<1>(dfs.top());
        bool eaten = get<2>(dfs.top());
        dfs.pop();
        if(queries < 6)
        {
            dfs.push({queries+1, lastSlice, eaten});
        }

        if(q.volume() == 0)
        {
            //Der Quader wurde vollstaendig zerlegt => Ausgabe!
            exit(0);
        }

        //Seitenflaechen des Quaders:
        pll s1 = q.sliceK(1);
        pll s2 = q.sliceK(2);
        pll s3 = q.sliceK(3);

        //Versucht nacheinander, Scheiben abzuschneiden (in der Reihenfolge des Algorithmus)
        if(queries == 0 && available(s1, slices, counters))
        {
            //Groesste Seitenflaeche muss abgeschnitten werden
            q.cutSlice(1);
            counters[s1]--;
            sliceVolume -= s1.first*s1.second;
            order.push(s1);
            dfs.push({0, s1, false});
        }
        else if(queries == 1 && s1.first*s1.second <= q.volume()-sliceVolume)
        {
            //Groesste Seitenflaeche als von Antje gegessen
```

```
            q.cutSlice(1);
60          order.push(s1);
            dfs.push({0, s1, true});
62      }

64      if(queries == 2 && available(s2, slices, counters))
        {
66          //Zweitgroesste Seitenflaeche muss abgeschnitten werden
            q.cutSlice(2);
68          counters[s2]--;
            sliceVolume -= s2.first*s2.second;
70          order.push(s2);
            dfs.push({0, s2, false});
72      }
        else if(queries == 3 && s2.first*s2.second <= q.volume()-sliceVolume)
74      {
            //Zweitgroesste Seitenflaeche als von Antje gegessen
76          q.cutSlice(2);
            order.push(s2);
78          dfs.push({0, s2, true});
        }
80
        if(queries == 4 && available(s3, slices, counters))
82      {
            //Drittgroesste Seitenflaeche muss abgeschnitten werden
84          q.cutSlice(3);
            counters[s3]--;
86          sliceVolume -= s3.first*s3.second;
            order.push(s3);
88          dfs.push({0, s3, false});
        }
90      else if(queries == 5 && s3.first*s3.second <= q.volume()-sliceVolume)
        {
92          //Drittgroesste Seitenflaeche als von Antje gegessen
            q.cutSlice(3);
94          order.push(s3);
            dfs.push({0, s3, true});
96      }

98      if(queries == 6) //Tiefensuche ueber diese Moeglichkeit ist beendet
        {
100         q.addSlice(lastSlice);
            if(!eaten)
102         {
                counters[lastSlice]++;
104             sliceVolume += lastSlice.first*lastSlice.second;
            }
106         order.pop();
        }
108 }

110 //Es wurde keine gueltige Zerlegung gefunden! => AUsgabe
    fout.close();
112 return 0;
}
```