

GYMNASIUM OTTOBRUNN

Oberstufenjahrgang 2017/19

Seminarfach Softwareentwicklung

Seminararbeit

36. Bundeswettbewerb Informatik

Runde 2

Aufgabe 1 und 3

Verfasser: Jonas Fritsch

Seminarleiter: StD Peter Brichzin

Bewertung: Punkte

Unterschrift des Seminarleiters:

INHALTSVERZEICHNIS

1	EINLEITUNG	4
2	AUFGABE 1 - "Die Kunst der Fuge"	5
2.1	Aufgabenstellung	5
2.2	Auslegung der Aufgabe	5
2.3	Lösungsidee	6
2.3.1	Verstehen des Problems	6
2.3.2	Der Algorithmus	13
2.4	Implementierung	21
2.5	Optimierungsmöglichkeiten	23
2.6	Beispiele	23
3	AUFGABE 3 - "Quo vadis, Quax?"	24
3.1	Aufgabenstellung	24
3.2	Lösungsidee	24
3.3	Teilaufgabe (a)	24
3.4	Teilaufgabe (b)	24
3.5	Teilaufgabe (c)	24
3.5.1	Implementierung	24
3.5.2	Optimierungsmöglichkeiten	24
3.5.3	Beispiele	24
3.6	Teilaufgabe (d)	24
4	FAZIT	25
5	ABBILDUNGSVERZEICHNIS	26
6	LITERATURVERZEICHNIS	27
7	CD-VERZEICHNIS	28
8	ANHANG	29
8.1	Aufgabe 1	29
8.1.1	Quelltext	29
8.1.2	Installationshinweise	29

8.2	Aufgabe 3	29
8.2.1	Quelltext	29
8.2.2	Installationshinweise	29
9	ERKLÄRUNG DES VERFASSERS	30

1 EINLEITUNG

Der aus drei Runden bestehende Bundeswettbewerb Informatik (BwInf) ist einer von vier Bundesweiten Informatikwettbewerben. Mit dem 36. BwInf wurde für die erste Runde mit **1463** Teilnehmern eine so hohe Teilnahme wie seit 1993 nicht mehr geschafft [1]. Von den ursprünglich 1463 Teilnehmern der ersten Runde, haben es **828** Teilnehmende geschafft sich für die zweite Runde zu qualifizieren [2]. Von diesen haben jedoch nur noch **154** Teilnehmer der zweiten Runde Lösungen für zwei Aufgaben ihrer Wahl eingereicht. Dies liegt nicht zuletzt an der stark steigenden Komplexität der Aufgaben gegenüber der ersten Runde. Zu der letzten, dritten Runde des Bundeswettbewerbs Informatik werden schlussendlich nur noch **25** Teilnehmer eingeladen. Das sind im Falle des BwInfs von 2017 nur rund **1,7%** der ursprünglichen Teilnehmer.

Im Folgenden wird eine eingereichte Lösung zu der ersten und dritten Aufgabe der zweiten Runde des 36. Bundeswettbewerbs Informatik erläutert.

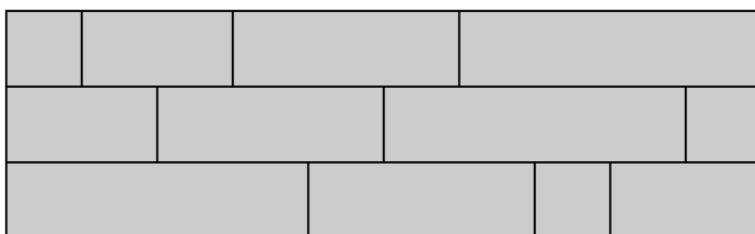
2 AUFGABE 1 - "Die Kunst der Fuge"

2.1 Aufgabenstellung

Ilona besitzt einen riesigen Haufen Holzklötzchen: Diese haben alle dieselbe Höhe und Tiefe, aber verschiedene Längen.

Ilona möchte eine Mauer bauen. Jede Reihe der Mauer soll aus n Klötzchen bestehen, die die Längen 1 bis n haben und lückenlos aneinander liegen. Die Stellen zwischen den Klötzchen heißen Fugen. Ilona möchte, dass in der fertigen Mauer niemals zwei Fugen übereinander liegen, selbst wenn sich mehrere Reihen dazwischen befinden. Außerdem soll ihre Mauer möglichst hoch sein.

Für $n = 4$ gelingt es ihr recht schnell, eine Mauer mit drei Reihen zu bauen:



Aufgabe

Hilf Ilona, indem du ein Programm schreibst, das nach Eingabe von n eine nach ihren Vorgaben konstruierte, möglichst hohe Mauer ausgibt. Für $n = 10$ sollte dein Programm eine Mauer der Höhe 6 ausgeben können. Wie hoch werden die Mauern deines Programms für größere n ?

2.2 Auslegung der Aufgabe

Die grundlegende Aufgabe ist, eine Mauer mit möglichst vielen Reihen, die aus verschieden länglichen Klötzchen bestehen, zu bauen.

Die Längen der Klötzchen in einer Reihe sind durch eine Variable n definiert. Die Variable n ($n \in \mathbb{N} \setminus \{0\}$) definiert dabei die Längen und auch die Gesamtanzahl der Klötzchen pro Reihe. Gesamtanzahl heißt, dass bei zum Beispiel $n = 5$ jede Reihe aus 5 einzelnen Klötzchen besteht. Jede Reihe setzt sich dann aus einem 1er, einem 2er, einem 3er, einem 4er und einem 5er Klotz zusammen.

Der Trick bei der Aufgabe ist nun, dass niemals zwei Fugen **übereinanderliegen**

dürfen. Eine Fuge ist die Lücke zwischen zwei Klötzchen (siehe Abbildung 1).

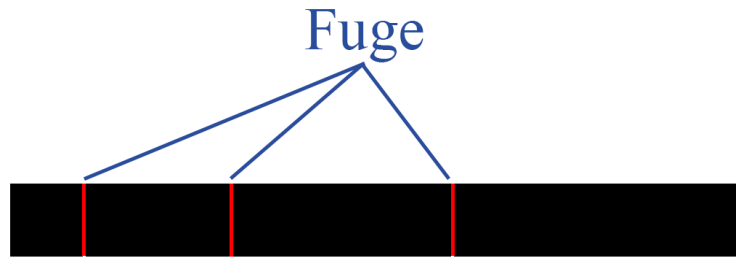


Abbildung 1: Darstellung der 3 Fugen einer Reihe mit 4 Klötzchen

Als Hilfestellung wurde außerdem erwähnt, dass eine Mauer für $n = 10$ eine maximale Höhe von 6 hat.

2.3 Lösungsidee

2.3.1 Verstehen des Problems

Um die Problemstellung der Aufgabe genauer zu verstehen, bietet es sich erstmal an, Mauern für kleinere n auf Papier zu zeichnen. Beginnt man mit $n = 1$, fällt auf, dass ganz gleich wie viele Reihen übereinanderliegen, es nie zu einer Fugenüberlappung kommen kann, da eine Reihe mit einem einzelnen Klötzchen keine Fuge besitzt. Für $n = 1$ kann also theoretisch eine **unendlich** hohe Mauer gebaut werden.

Für $n = 2$ ist es auch noch recht einfach, eine Mauer zu bauen. Denn es gibt pro Reihe nur 2 mögliche Reihenfolgen von Klötzchen. Entweder kommt zuerst der 1er oder der 2er Klotz. Eine der zwei möglichen Lösungen ist in Abbildung 2 zu sehen. Es gibt keine Möglichkeit, noch ein Klötzchen in einer neuen dritten Reihe zu platzieren, ohne eine Fuge doppelt zu besetzen.

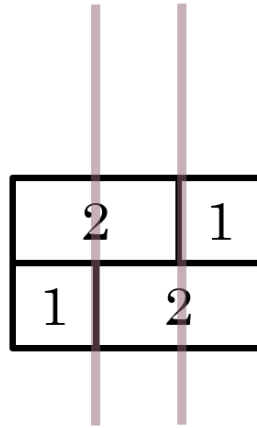


Abbildung 2: Eine der zwei möglichen Lösungen für $n = 2$. Die roten Striche zeigen an, dass die jeweilige Fuge bereits besetzt ist.

Auch für $n = 3$ gibt es eine Mauer mit einer maximalen Höhe von 2 (siehe Abbildung 3). Hier wird es jedoch schon komplexer, da es nun für eine Reihe nicht mehr $2! = 2 \times 1 = 2$, sondern $3! = 3 \times 2 \times 1 = 6$ mögliche Klötzchen-Reihenfolgen gibt. Außerdem tritt bei $N = 3$ das erste Mal auf, dass eine **mögliche Fugenstelle nicht besetzt wird**.

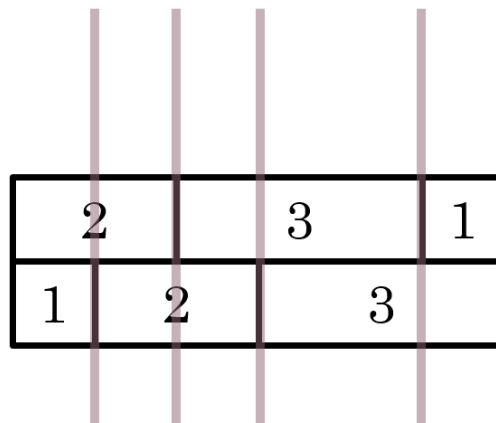


Abbildung 3: Eine mögliche Lösung für $n = 3$. Zu beachten ist, dass in der Mauer eine **mögliche Fugenstelle**, dort wo es eine größere Lücke zwischen den zwei letzten besetzten Fugenstellen gibt, **unbesetzt bleibt**.

Geht man noch weiter und bildet eine Mauer für $n = 4$ ($4! = 24$ mögliche Klötzchen Anordnungen pro Reihe), so kann man erstmals eine Mauer der Höhe 3 bilden (siehe Abbildung 4). Hier ist auch wieder jede einzelne Fugenstelle besetzt.

	4			3		1		2
2		3				4		1
1	2		3				4	

Abbildung 4: Eine mögliche Lösung für $n = 4$

Die Komplexität der Aufgabe steigt mit höheren n stark an. Während man bei $n = 2$ $2!$ Möglichkeiten pro Reihe und 2 Reihen hatte ($2! \times 2 = 4$ unterschiedliche Mauern), gibt es bei $n = 4$ bereits $4! \times 3 = 72$ verschiedene Mauern. Allgemein gibt es also für eine Mauer $n! \times h$ Lösungen, wobei h die Anzahl der übereinanderliegenden Reihen ist.

Es ist logisch, dass zwischen der Anzahl der maximal möglichen Reihen h und der Anzahl der Klötzchen pro Reihe n eine Abhängigkeit existiert.

Allgemein kann man sagen, dass die **maximale Mauerhöhe** dann erreicht ist, wenn **nicht** mehr **genug Fugenstellen in der Mauer frei sind**, um eine **weitere Reihe zu bilden**.

Somit lässt sich die Formel

$$h = \frac{f_{Mauer}}{f_{Reihe}}$$

ableiten, wobei f_{Mauer} die maximale Anzahl an Fugen in der Mauer und f_{Reihe} die Anzahl der Fugen, die eine Reihe besetzt, beschreibt.

Um nun auf f_{Mauer} schließen zu können, braucht man nur die Länge der Mauer. Da jede Reihe der Mauer gleich lang ist, ist die Länge der Mauer gleich der Länge einer Reihe. Die Länge einer Reihe definiert sich wiederum durch die Menge und Breite ihrer Klötzchen. Das heißt also, die Länge einer Reihe/der Mauer ist nach der **Gaußschen Summenformel**

$$1 + 2 \dots + n = \sum_{k=1}^n k = \frac{n(n+1)}{2}$$

Um nun aber von der Länge der Mauer auf f_{Mauer} zu kommen, muss man noch 1 von der Länge der Mauer abziehen. Dies wird offensichtlich, wenn man zu einer Mauer eine Reihe aus lauter 1er Klötzchen bildet (siehe Abbildung 5). Da das Ende des letzten Klötzchens nicht als Fuge gilt, wird von der Länge der Reihe noch 1 abgezogen.

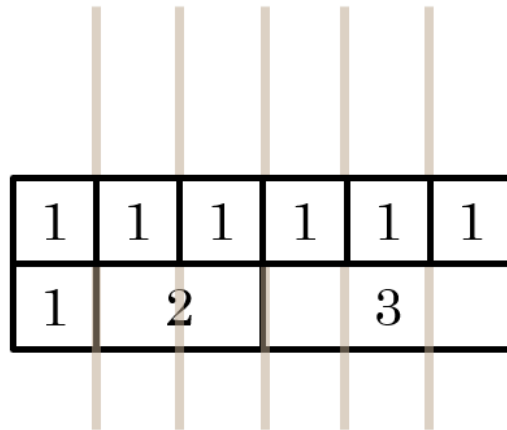


Abbildung 5: Eine Mauer der Länge 6 besitzt 5 mögliche Fugenstellen.

Ähnlich lässt sich auch f_{Reihe} berechnen. Um wissen zu wollen, wie viele freie Fugen eine Reihe mit n Klötzchen besetzen wird, zieht man einfach wieder 1 von der Anzahl der Klötzchen pro Reihe (n) ab (siehe Abbildung 6).

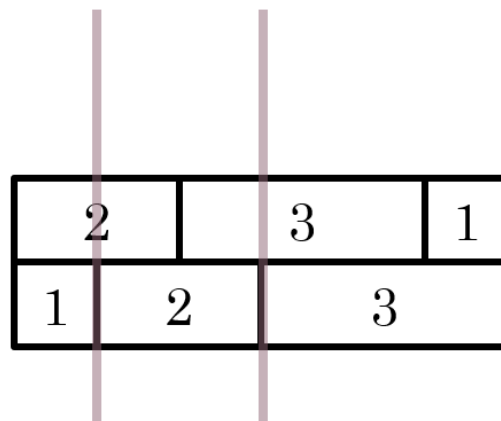


Abbildung 6: Eine Reihe bei $n = 3$ besetzt $n - 1 = 2$ freie Fugen.

Daraus ergibt sich die fertige Formel für die Anzahl der Reihen h in einer maximal hohen Mauer in Abhängigkeit von n .

$$h = \frac{\frac{n \times (n+1)}{2} - 1}{n - 1} = \frac{n}{2} + 1$$

Der Graph der daraus folgenden Funktion $h(n) = \frac{n}{2} + 1$ verläuft dabei linear (siehe Abbildung 7).

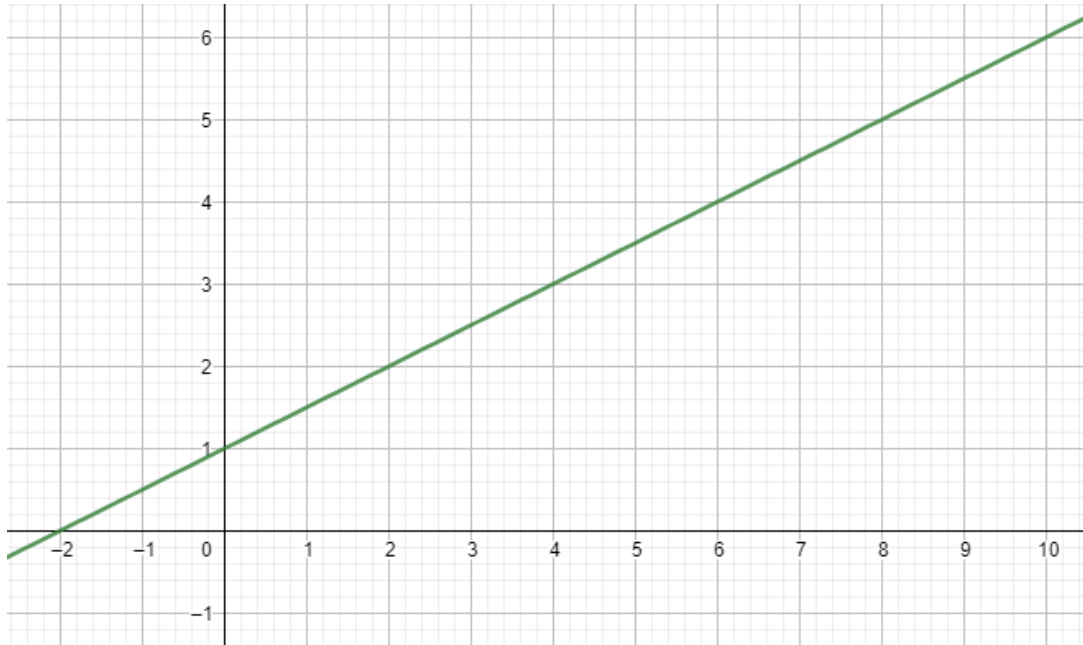


Abbildung 7: Graph der linearen Funktion $h(n) = \frac{n}{2} + 1$

Setzt man nun für $n = 10$ ein, kommt der Funktionswert 6 raus der mit der in der Aufgabenstellung angegebenen Höhe für eine Mauer von $n = 10$ übereinstimmt. Testet man allerdings die maximale Mauerhöhe für zum Beispiel $n = 3$, so kommt als Wert ungefähr 2,5 raus. Dies macht allerdings keinen Sinn, da die fertige Mauer nur aus vollen Reihen bestehen muss und keine halbfertige Mauer besitzen kann.

Diese Komma-Werte kommen bei allen ungeraden n vor. Der Zusammenhang ist dabei, dass wie beim vorherigen Beispiel $n = 3$ (siehe Abbildung 3) eine Fuge unbesetzt bleibt. Diese Fuge kann zwar mit einem weiteren Klotz besetzt werden, die Reihe könnte aber nicht mehr vollständig gebaut werden. Da unsere Mauerhöhe also eine natürliche Zahl sein muss, umgeben wir den Funktionsterm noch mit Abrundungsklammern:

$$h(n) = \lfloor \frac{n}{2} + 1 \rfloor$$

Der daraus resultierende Graph ist in Abbildung 8 zu sehen.

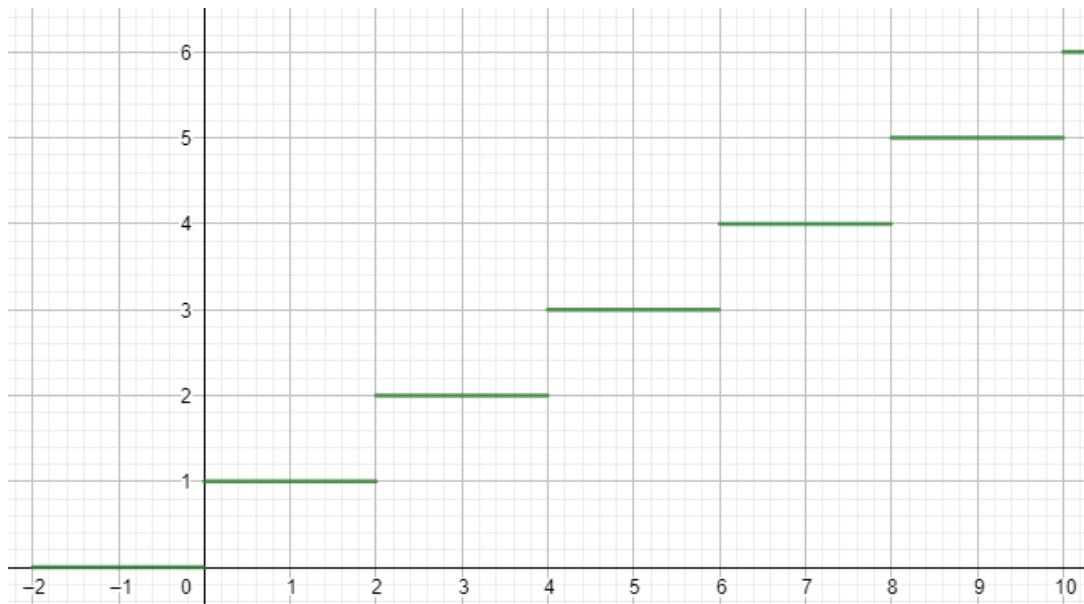


Abbildung 8: Graph der Funktion $h(n) = \lfloor \frac{n}{2} + 1 \rfloor$

Somit gibt uns die Funktion $h(n)$ mit der Definitionsmenge $\mathbb{D}_h = \mathbb{N} \setminus \{0; 1\}$ und der Wertemenge $\mathbb{W}_h = \mathbb{N} \setminus \{0\}$ die jeweilige maximale Mauerhöhe (h) in Abhängigkeit von n an. Wobei in der Definitionsmenge auch 1 ausgeschlossen ist, weil für $n = 1$ aus logischer Sicht eine unendlich hohe Mauer und nicht nur eine Mauer der Höhe 1 gebildet werden kann.

Die naivste Lösung für die Aufgabe wäre per Brute-Force so lange Klötzchen aneinanderzureihen und Reihen aufeinanderzustapeln, bis man eine fertige Mauer der maximalen Höhe erreicht hat.

Für die in der Aufgabenstellung festgelegte Mindestanforderung $n = 10$, gäbe es bereits

$$f(10) = 10!^6 \approx 2,28e+39$$

unterschiedliche Mauerkombinationen bei der maximalen Mauerhöhe von 6. Der Graph der allgemeinen Formel

$$f(n) = n!^{\lfloor \frac{n}{2} + 1 \rfloor}$$

ist in Abbildung 9 zu sehen.

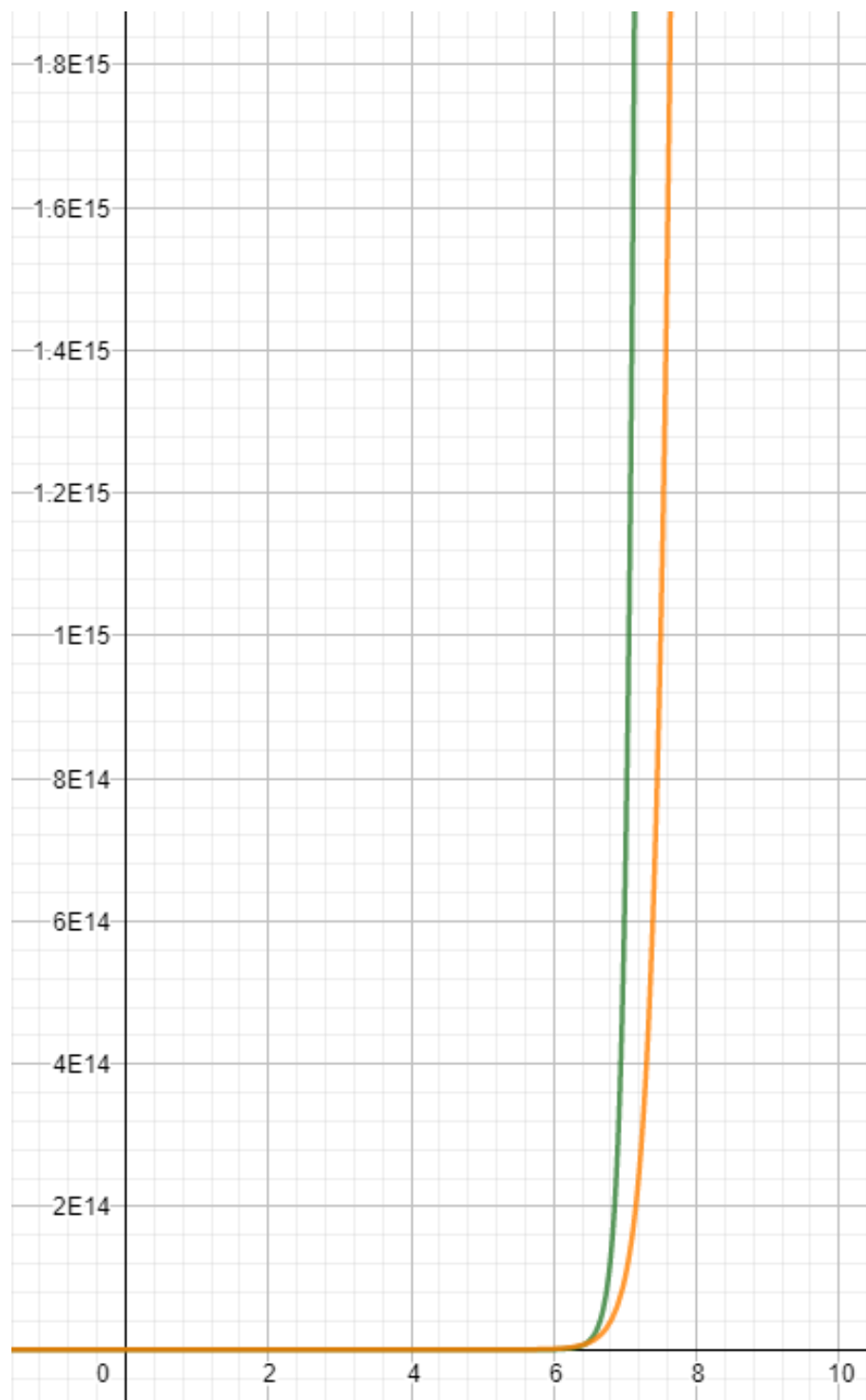


Abbildung 9: Der grüne Graph zeigt die extreme Steigung der Funktion $f(n) = n!^{\lfloor \frac{n}{2} + 1 \rfloor}$. Zum Vergleich der orangen-farbige Graph der Funktion $g(x) = 100^x$.

Die Anzahl der verschiedenen möglichen Mauern für n steigt somit mehr als **exponentiell**.

Nun ist hierbei zwar zu beachten, dass es durchaus mehrere mögliche unterschiedliche Lösungen geben kann und das bei der Gesamtmenge unterschiedlicher Mauern auch auf die Reihenfolge von Reihen geachtet wird (siehe Abbildung 10), welche für die Aufgabenstellung keinen Unterschied macht.

2	3	1
1	2	3

1	2	3
2	3	1

Abbildung 10: Obwohl beide Mauern aus den selben zwei Reihen bestehen, werden sie als unterschiedliche Mauern angesehen.

Weiterhin ist zu beachten, dass es für Mauern mit ungeradem n mehr unterschiedliche Lösungen gibt, als für die wieder darauffolgende Mauer mit geradem n .

Dies liegt eben daran, dass bei ungeraden n immer mindestens eine Fugenstelle frei bleibt, was wiederum zu mehr richtigen Mauervariationen führt.

Trotzdem würde das Finden einer Lösung für $n = 10$, selbst wenn ein Computer nur **1 Millisekunde** für die Permutation einer **gesamten Mauer** benötigen würde, die maximal erwartete Algorithmuslaufzeit einer BwInf-Lösung bei weitem überschreiten.

Damit ist eine naive Brute-Force Methode als Lösung ausgeschlossen und es muss nach einem effizienteren Algorithmus gesucht werden.

2.3.2 Der Algorithmus

Grundlegend gibt es zwei unterschiedliche Wege eine Mauer, wie in der Aufgabenstellung zu bauen.

Der offensichtlichere ist die Bauart **'von unten nach oben'**. Bei dieser Bauart setzt man zuerst die Klötzchen zu einer Reihe zusammen. Hat man eine Reihe fertig, beginnt man mit der zweiten Reihe und so weiter (siehe Abbildung 11). Man setzt Reihe auf Reihe (von unten nach oben) bis man die Maximalhöhe erreicht hat.

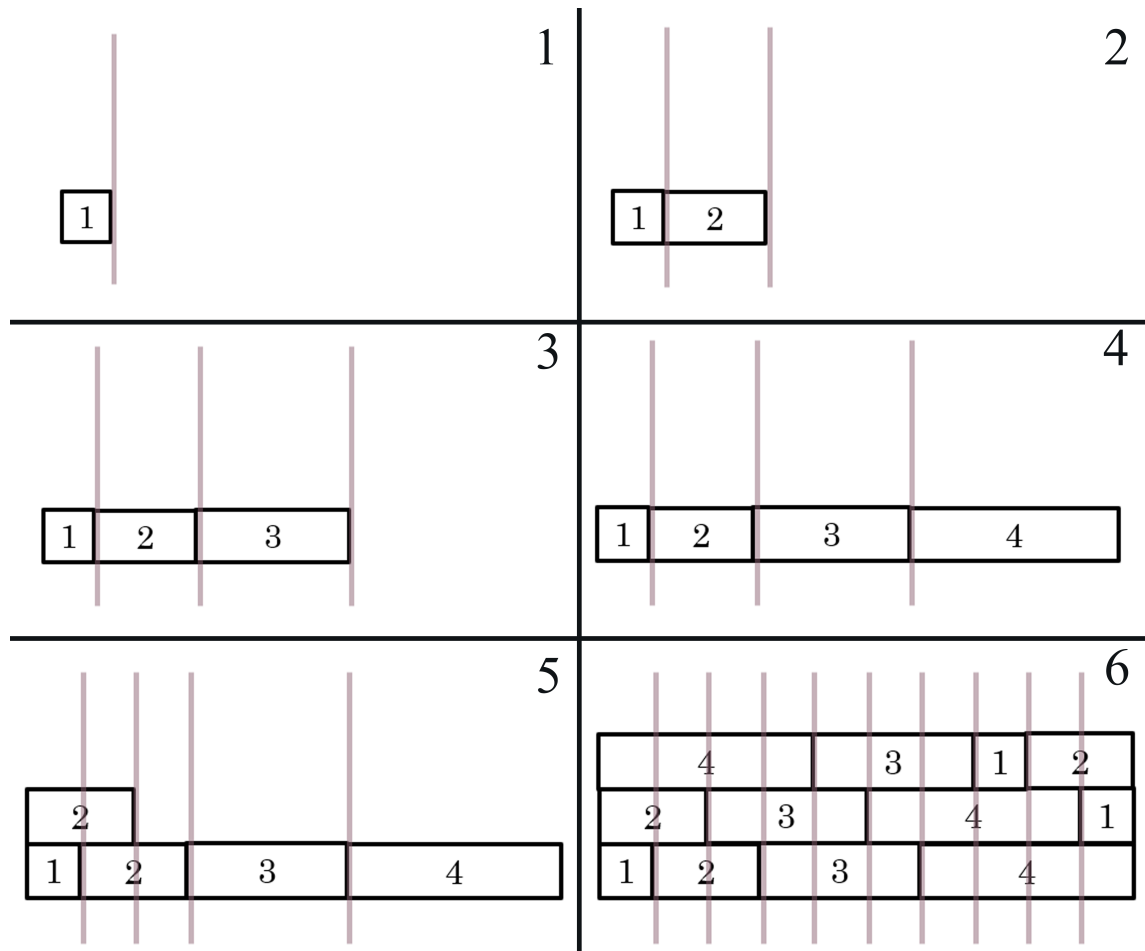


Abbildung 11: Der Bau einer Mauer nach der **'von unten nach oben'** Methode. Es wird Reihe nach Reihe gebaut (1-5), bis die maximale Höhe erreicht wurde (6).

Eine andere Möglichkeit ist, wenn man die Anzahl der Reihen (h) für eine Mauer der maximalen Höhe bereits weiß, die Mauer **'von links nach rechts'** aufzubauen. Man kann die Aufgabe auch so formulieren, dass das **Ziel ist, möglichst alle Fugenposition in der Mauer zu füllen**. Nun bietet es sich an, anstatt wie bei **'von unten nach oben'** immer einen Klotz zu setzen und **dann** zu gucken, ob die jeweilige Fugenposition noch frei ist, von den **Fugenpositionen auszugehen** und diese jeweils zu versuchen **mit einem Klotz zu füllen** (siehe Abbildung 12).



Abbildung 12: Der Bau einer Mauer nach der **'von links nach rechts'** Methode. Es wird jeweils immer die nächste freie Fuge angeschaut und überprüft, welcher Klotz in welcher Reihe diese Fuge füllen kann. Dieser Klotz wird dann gesetzt (1-5). Dies wiederholt sich solange, bis die letzte Fuge erreicht wurde und die Mauer fertig ist (6).

Diese **'von links nach rechts'** Bauart ist dadurch möglich geworden, dass man schon im vornherein die Anzahl an Reihen weiß, die für eine maximale Mauer benötigt wird.

Aber auch bei dieser Bauart bleibt die Frage, was passiert, wenn mehr als ein Klotz die nächste Fuge füllen kann. Dies passiert nämlich bei fast jeder Fuge (siehe Abbildung 13).

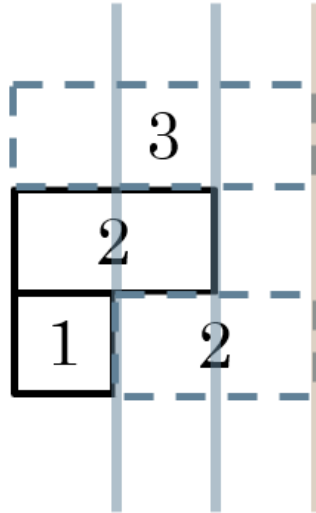


Abbildung 13: Zum Besetzen der dritten Fuge (orange markiert) bei $n = 4$ kann entweder ein 3er Klötzchen in einer neuen Reihe, oder ein 2er Klötzchen in der untersten Reihe platziert werden.

Wählt man ein zufälliges Klötzchen aus, so kann es schnell dazu kommen, dass eine Reihe soweit in 'Rückstand' gerät, dass kein möglicher Klotz mehr lang genug wäre, um die Reihe fortzuführen (siehe Abbildung 14).

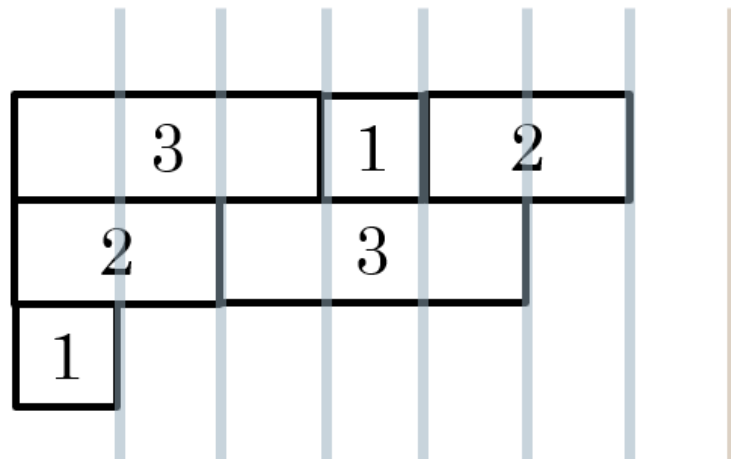


Abbildung 14: In der dargestellten Mauer für $n = 4$, gibt es für die **unterste Reihe keine Möglichkeit mehr ein Klötzchen zu platzieren**. Da die obereren Reihen alle Fugen schon besetzt haben und die nächste freie Fuge **6 Längen** von der untersten Reihe entfernt ist, würde diese Reihe ein **6er Klötzchen** benötigen. Bei $n = 4$ gibt es jedoch maximal **4er Klötzchen**.

Es ist also wichtig immer darauf zu achten, dass die kürzeste Reihe nicht den An-

schluss verliert.

Die Lösung für dieses Problem ist bei dem Fall, dass wenn verschiedene Klötzchen mehrerer Reihen eine Fuge befüllen können, immer die **kürzeste Reihe zu bevorzugen**.

Mit diesem Algorithmus kommt man bis zu $n = 8$. Dort tritt ein weiteres Problem auf (siehe Abbildung 15).

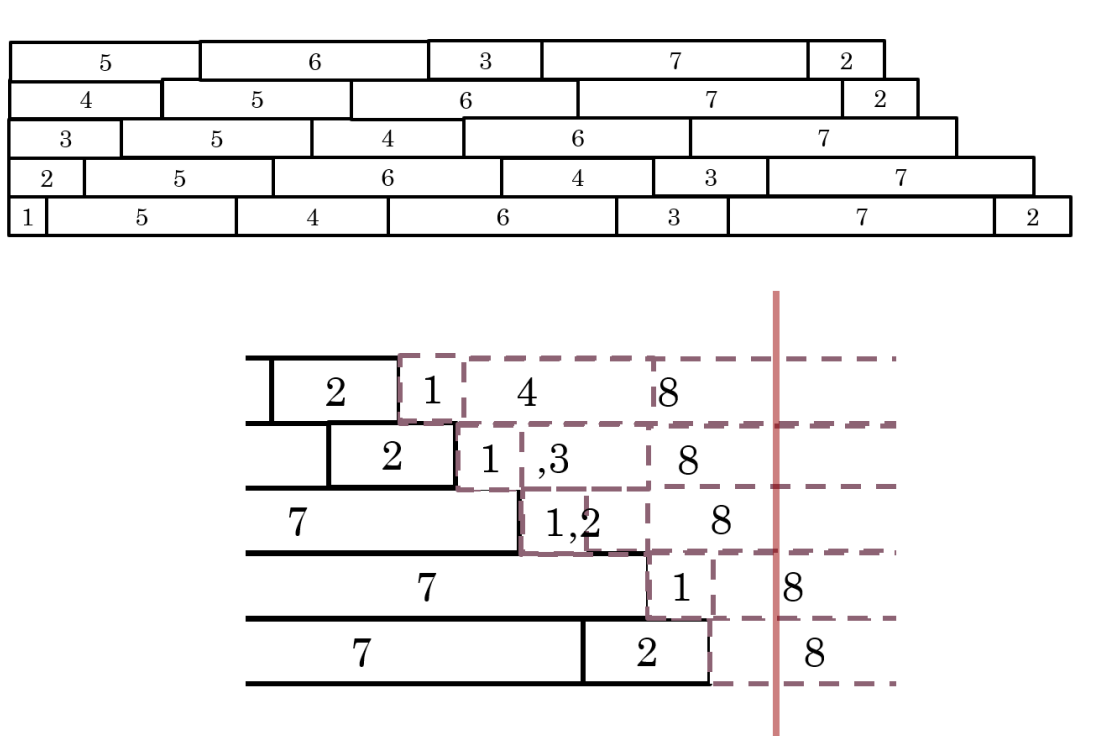


Abbildung 15: Bei $n = 8$ tritt das Problem auf, dass die **29 Fuge** (rot markiert), durch **kein einziges, mögliches Klötzchen besetzt werden kann**. Da n jedoch eine **gerade Zahl** ist, wissen wir, dass für eine Mauer der maximalen Höhe **alle Fugen besetzt werden müssen**. Das untere Bild zeigt nochmal den hinteren Ausschnitt der Mauer mit allen noch möglichen Klötzchen für jede Reihe an. Keines davon würde die nächste Lücke füllen.

Um dennoch eine Lösung zu bekommen, wird **Backtracking** verwendet. Backtracking erlaubt es, **zur letzten Entscheidung zurückzugehen** und eben **eine andere** zu **treffen**. Mit Entscheidung ist die Wahl eines Klötzchens um eine freie Fuge zu besetzen, wenn es mehrere Möglichkeiten gäbe gemeint (siehe Abbildung 16).

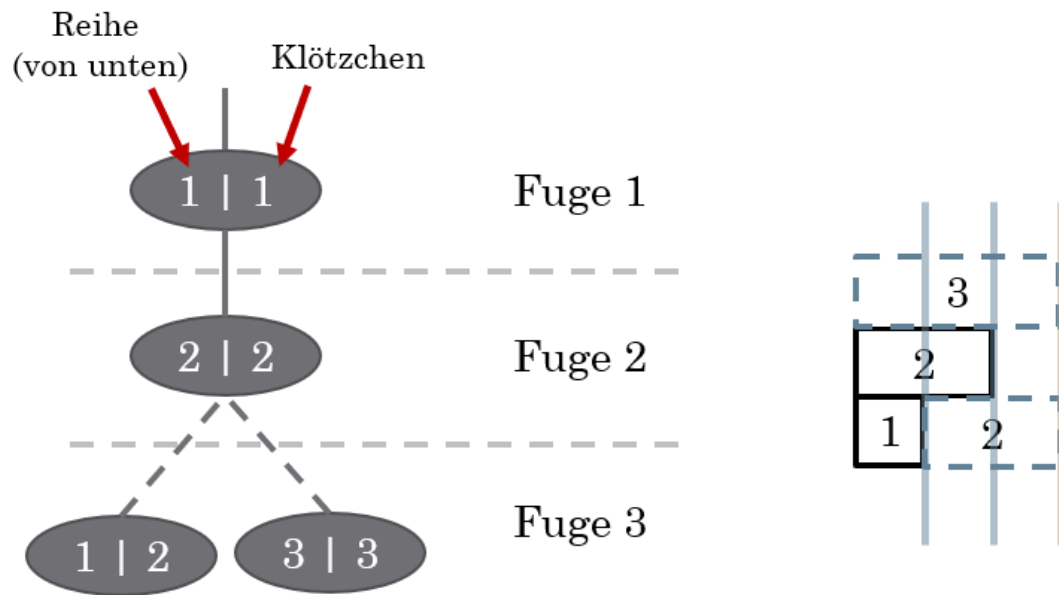


Abbildung 16: Bei einer Mauer von $n = 4$ tritt das vorher schon besprochene Problem auf, dass es zwei mögliche Klötzchen gibt, die die **3 Fuge** besetzen können. Der Baum auf der linken Seite veranschaulicht die möglichen Klötzchen pro Fuge. Die **Tiefe des Baumes** entspricht dabei einer **jeweiligen Fugenposition** und die **Knoten dieser Tiefe** stehen für die **möglichen Klötzchen**.

Wählen wir nun das Klötzchen der kürzesten Reihe (das 3er Klötzchen in der obersten, neuen Reihe) und **gehen davon aus, dass die 4te Fuge nun nicht mehr gefüllt werden kann** (siehe Abbildung 17),

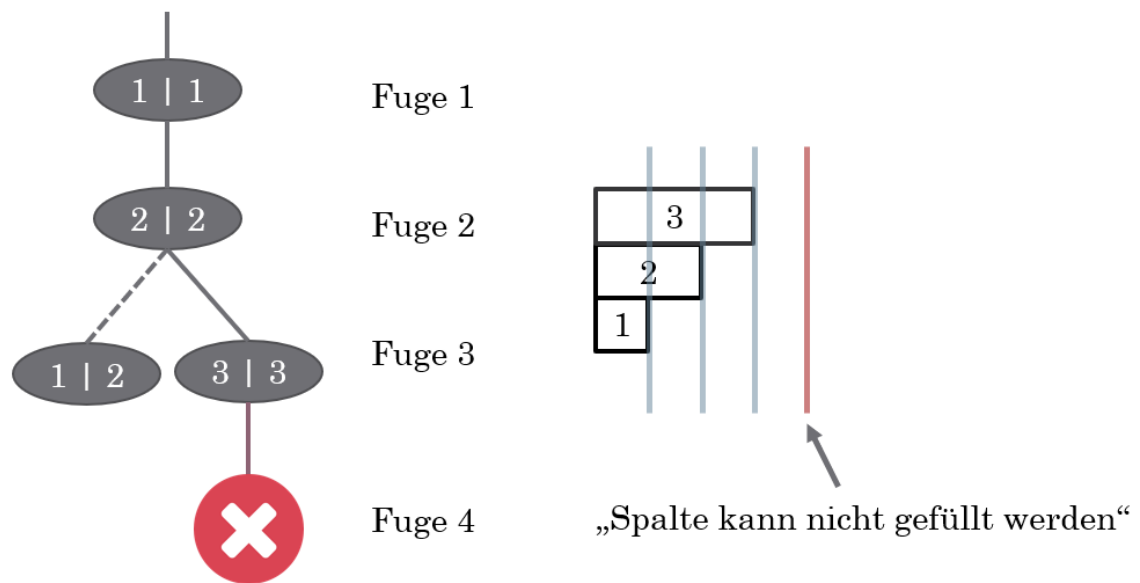


Abbildung 17: Angenommen nach der Entscheidung vom einsetzen des 3er Klötzchens, kann die 4te freie Fuge nicht besetzt werden. Theoretisch könnte in der untersten Reihe ein 3er Klotz die 4te Fuge füllen. Zur Vereinfachung der Darstellung wird dies jedoch außer Acht gelassen.

so gehen wir **einen Schritt zurück, zu unserer letzten Entscheidung** und **wählen dort das andere Klötzchen** (siehe Abbildung 18).



Abbildung 18: Nach dem "Trial and error"Prinzip wird beim Backtracking zur letzten Entscheidung zurückgegangen (oberes Bild) und ein anderes Klötzchen ausgewählt (unteres Bild). Dann wird mit dem Algorithmus fortgefahren.

Damit ist der grundlegende Algorithmus vollständig.

2.4 Implementierung

Das Programm wurde in der Sprache C# als Konsolen Applikation mit dem .NET Framework 4.7.1 geschrieben. Das in Abbildung 19 gezeigte Klassendiagramm zeigt die Grundstruktur des Programms.

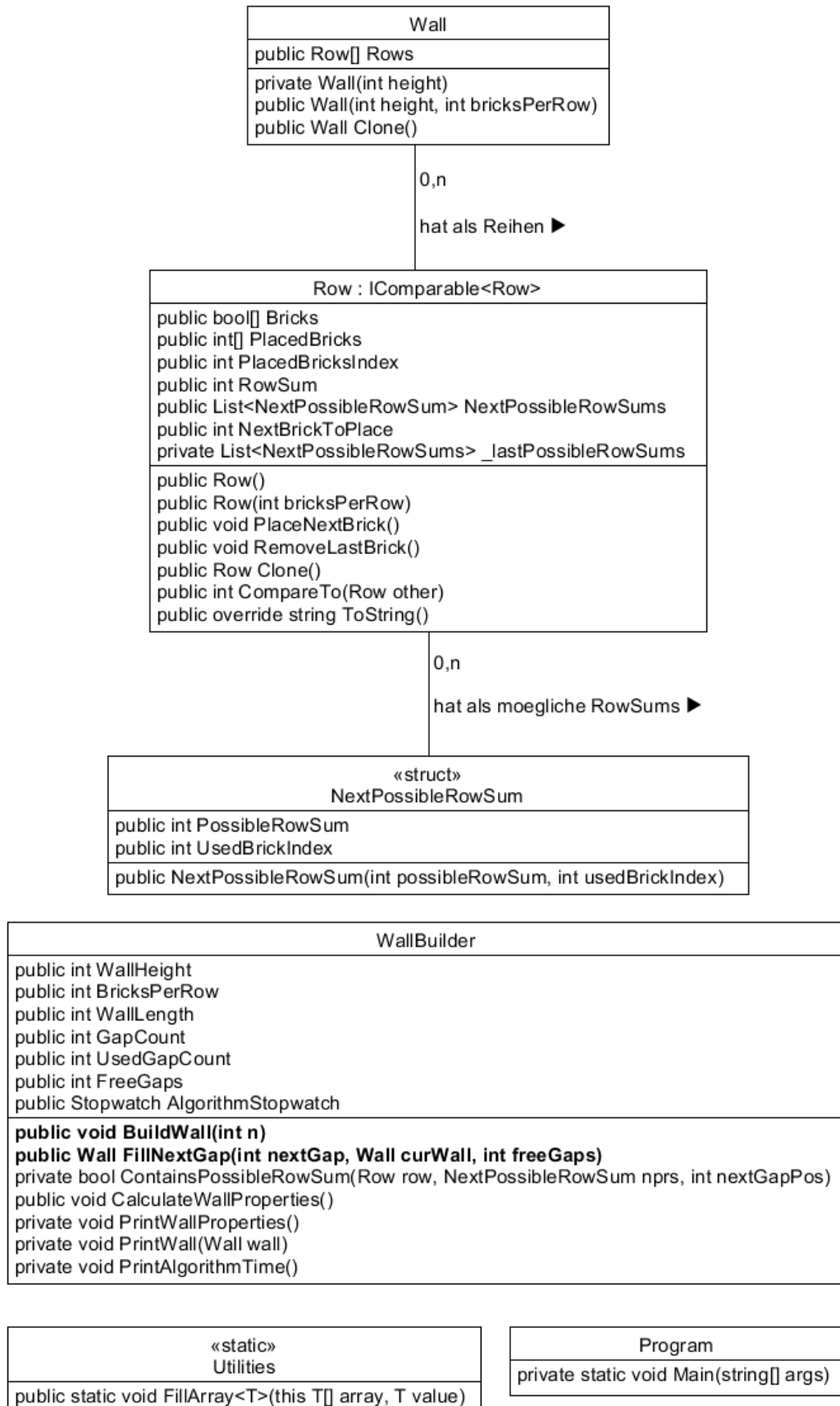


Abbildung 19: Erweitertes Klassendigramm des Programms. Die wichtigsten zwei Methoden des Algorithmus wurden in der Klasse *WallBuilder* hervorgehoben.

2.5 Optimierungsmöglichkeiten

2.6 Beispiele

3 AUFGABE 3 - "Quo vadis, Quax?"

3.1 Aufgabenstellung

3.2 Lösungsidee

3.3 Teilaufgabe (a)

3.4 Teilaufgabe (b)

3.5 Teilaufgabe (c)

3.5.1 Implementierung

3.5.2 Optimierungsmöglichkeiten

3.5.3 Beispiele

3.6 Teilaufgabe (d)

4 FAZIT

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempus consectetur lorem, imperdiet dignissim est auctor a. Vivamus convallis, leo et iaculis egestas, nunc massa porttitor tellus, id faucibus urna justo eget massa. Praesent quis feugiat odio. Nullam quis mattis enim. Fusce volutpat odio in enim sodales venenatis. Mauris consequat.

5 ABBILDUNGSVERZEICHNIS

Alle enthaltenen Abbildungen wurden mithilfe von folgenden Tools selbstständig erstellt:

- Microsoft PowerPoint
- Grafikrechner - GeoGebra
- UMLet
- Adobe Photoshop CC 2018

6 LITERATURVERZEICHNIS

Buchquellen:

- [1] Leslie Lamport, *TEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.
- [2] Leslie Lamport, *TEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.
- [3] Leslie Lamport, *TEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.

PDF-Scans:

- [1] Leslie Lamport, *TEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.
- [2] Leslie Lamport, *TEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.

Internetquellen:

- [1] Leslie Lamport, *TEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.
- [2] Leslie Lamport, *TEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.
- [3] Leslie Lamport, *TEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd edition, 1994.

7 CD-VERZEICHNIS

8 ANHANG

8.1 Aufgabe 1

8.1.1 Quelltext

8.1.2 Installationshinweise

8.2 Aufgabe 3

8.2.1 Quelltext

8.2.2 Installationshinweise

9 ERKLÄRUNG DES VERFASSERS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tempus consectetur lorem, imperdiet dignissim est auctor a. Vivamus convallis, leo et iaculis egestas, nunc massa porttitor tellus, id faucibus urna justo eget massa. Praesent quis feugiat odio. Nullam quis mattis enim. Fusce volutpat odio in enim sodales venenatis. Mauris consequat.

Ich erkläre hiermit, dass ich die Seminararbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benützt habe.

....., den

Ort

Datum

.....

Unterschrift des Verfassers