Aufgabe 1
Die Kunst der Fuge
Dokumentation

Kamal Abdellatif

1 Obere Schranke

Die Höhe einer Mauer von beliebigem n ist nach oben beschränkt. Eine Mauer ist so breit wie die Summe der Längen ℓ aller Steine in einer Reihe. Da sich zwischen den Steinen nur die Reihenfolge ändert, entspricht die Breite jeder Reihe der Breite w der Mauer.

$$w = \sum_{\ell=1}^{n} \ell = \frac{1}{2} n(n+1) \tag{1.1}$$

In jeder Reihe befinden sich genau $n-1$ Fugen, da auf jeden der n Steine in der Reihe genau eine Fuge folgt, mit Ausnahme des letzten Steins. Besitzt die Mauer eine Höhe h, so ist die Anzahl aller Fugen in der gesamten Mauer $h(n-1)$.
Eine Fuge nimmt entlang der Breite der Mauer eine bestimmte Position ein. Jede mögliche Position ist eine *Spalte*, wobei die i-te Spalte den Abstand i zum linken Mauerrand besitzt. Liegt auf einer Spalte eine Fuge, so ist die Spalte *besetzt*.

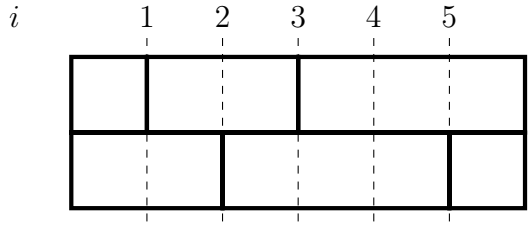

Abbildung 1: Spalten einer Mauer

Eine Mauer der Breite w besitzt $w-1$ Spalten, da die erste Spalte Distanz 1 und die letzte Spalte Distanz $w-1$ zum linken Rand aufweist. Hat eine Mauer keine sich überschneidenden Fugen, so kann jede Spalte von maximal einer Fuge belegt sein. Mit jeder hinzukommenden Reihe werden $n-1$ unbelegte Spalten mit Fugen besetzt. Ist es

nicht mehr möglich, eine Reihe hinzuzufügen, ohne dass eine Überschneidung entsteht, ist die höchst mögliche Mauer erreicht. Diese maximale Höhe ergibt sich somit aus dem Quotienten aus der Anzahl aller Spalten und der Anzahl von Fugen pro Reihe.

$$\begin{aligned} h(n-1) &\leq w-1 \\ &\leq \frac{1}{2}n(n+1)-1 \\ h &\leq \frac{\frac{1}{2}n(n+1)-1}{n-1} = \frac{\frac{1}{2}n(n-1)+n-1}{n-1} \\ h &\leq \frac{1}{2}n+1 \end{aligned} \tag{1.2}$$

Nur für gerade n nimmt diese obere Schranke ganzzahlige Werte an. Die Schranke ist genau dann erreicht, wenn jede Spalte durch genau eine Fuge belegt ist. Für ungerade n ist die Schranke nie ganzzahlig, sodass sie nie erreicht werden kann. Demnach müssen im ungeraden Fall immer Spalten offen bleiben.

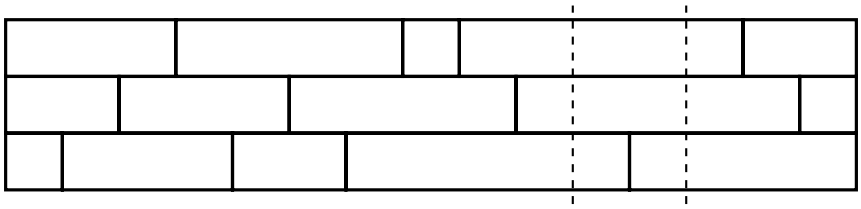

Abbildung 2: Offene Spalten bei ungeraden n

Die größte erreichbare Höhe h_{max} für ungerade n ist gleich der größten ganzen Zahl unter der Schranke. Aus Gleichung (1.2) ergibt sich

$$h_{max} = \frac{1}{2}(n-1)+1 = \frac{1}{2}n+1 \tag{1.3}$$

Wird diese Höhe erreicht, so ist die Anzahl r der noch offenen Spalten die Differenz aus der Anzahl aller Spalten $w-1$ und der Anzahl besetzter Spalten $h_{max}(n-1)$.

$$\begin{aligned} r &= (w-1) - h_{max}(n-1) \\ &= \left(\frac{n(n+1)}{2}-1\right) - \frac{n+1}{2}(n-1) \\ &= \frac{(n+1)(n-1)+n-1}{2} - \frac{(n+1)(n-1)}{2} \\ &= \frac{1}{2}(n-1) \qquad (2 \nmid [TODOBEAUTIFY]n) \end{aligned} \tag{1.4}$$

Für eine Mauer ungeraden n ohne Überschneidungen müssen mindestens $r = \frac{1}{2}(n-1)$ Spalten offen bleiben.

2 Backtrakcing

2.1 Vorbetrachtung

Das Problem wird mit Backtracking gelöst. Es wird davon ausgegangen, dass die nach der oberen Schranke (1.2) gegebene größtmögliche Höhe $h = h_{max}$ immer erreichbar ist. Um eine Mauer n zu lösen, wird von einer leeren Mauer der Dimensionen $w \times h$ ausgegangen. Ziel des Backtracking ist es, diese Mauer Stein für Stein zu füllen, ohne dass sich Fugen überschneiden.
Eine Mauer $W = \{R_1, \ldots, R_h\}$ besteht aus h Reihen R. Während der Konstruktion der Mauer sind die Reihen nur teilweise gefüllt. Eine Reihe $R = \{b_1, \ldots, b_k\}$ enthält $k \leq n$ Steine einer bestimmten Länge b. Ein Stein b_i ist der i-te Stein von links innerhalb der Reihe. Nur die Steine $\Sigma = \{1, \ldots, n\}$ können in einer Reihe $R \subseteq \Sigma$ enthalten sein. Jede Reihe R besitzt eine Länge $\ell(R)$. Diese Länge ergibt sich aus der Summe aller in R enthaltenen Steinlängen.

$$\ell(R) = \sum_{b_i \in R} b_i \tag{2.1}$$

Die Reihe endet in der ℓ-ten Spalte. Diese Spalte wird als *Endpunkt* der Reihe bezeichnet.

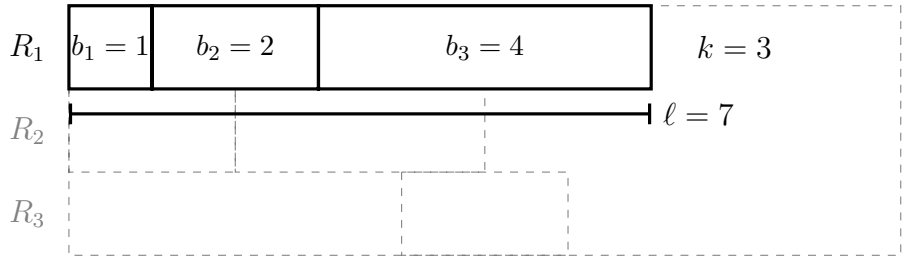

Abbildung 3: Bezeichnungen

2.2 Gerade Fälle

In der Lösung einer Mauer mit geraden n ist jede Spalte durch genau eine Fuge besetzt. Um diese Lösung zu konstruieren, werden die Spalten nacheinander von links nach rechts mit Fugen aufgefüllt. Mit jedem Backtracking-Schritt wird eine Spalte gefüllt. Dazu wird ein Stein direkt an das rechte Ende einer Reihe angelegt.

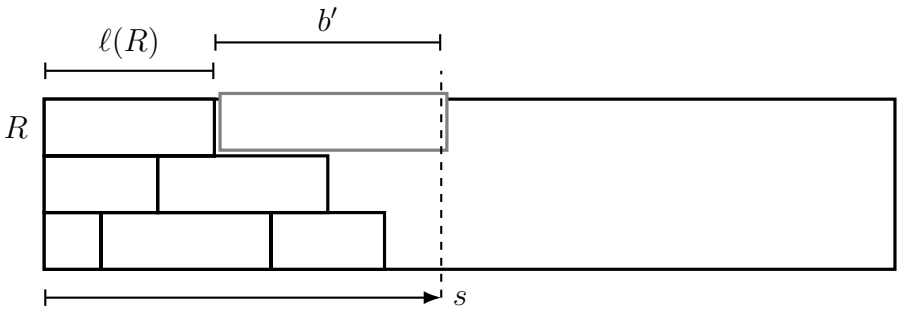

Abbildung 4: Anlegen eines Steins

Soll eine Reihe R zu der Spalte s erweitert werden, muss ein Stein bestimmter Länge b' hinzugefügt werden. Da die Reihe nach Anfügen des Steins in Spalte s endet, ist der neue Stein genau so lang sein wie die Differenz zwischen s und dem aktuellem Ende der Reihe ℓ. Dabei darf b' noch nicht in R enthalten sein, da es sonst zu einer Wiederholung kommen würde.

$$b' = s - \ell(R) \qquad (b' \in \Sigma \backslash R) \tag{2.2}$$

Nach Anfügen des Steins hat sich in der R die Anzahl k von Steinen um eins erhöht und die Länge um b' vergrößert. Da die Lücke s nun durch eine Fuge gefüllt ist, wird als nächstes die darauf folgende Spalte $s + 1$ gefüllt. Dieser Vorgang wird als *Schritt* bezeichnet.

$$R' = R \cup \{b'\} \qquad \ell' = \ell + b' \qquad s' = s + 1 \tag{2.3}$$

Durch diese Vorgehensweise wird jede Spalte durch genau eine Fuge besetzt und in keiner Reihe wird ein Stein wiederholt. Wurden alle Reihen vollständig gefüllt, so ist eine Lösung gefunden worden.
Das Backtracking öffnet einen neuen Suchzweig für jede mögliche Reihe, die zum Füllen der nächsten Spalte gewählt werden kann. Eine Reihe R kann genau dann gewählt werden, wenn der benötigte Stein b' noch nicht in R enthalten ist. Ist keine Reihe mehr möglich, so muss der gesamte Suchzweig verworfen werden.
Das Backtracking geht in Tiefensuche vor und wird rekursiv implementiert. Jeder Rekursionsschritt besteht aus drei Schritten:

(1) Der zu untersuchende Schritt wird ausgeführt: Der Stein wird an die entsprechende Reihe angefügt und die aktuelle Spalte inkrementiert.

(2) Alle Folgeschritte werden untersucht. Dazu wird durch alle möglichen Reihen iteriert. Der Erfolg des aktuellen Schrittes hängt davon ab, ob mindestens eine davon erfolgreich war.

(3) Die unternommenen Änderungen werden rückgängig gemacht.

Sobald eine Lösung gefunden wurde, wird die Suche abgebrochen. War eine der Möglichkeiten erfolgreich, so werden weitere Möglichkeiten außer acht gelassen, und der Erfolg wird sofort zurückgegeben.
Für den vollständigen Algorithmus in Pseudocode siehe Anhang (1).

2.3 Ungerade Fälle

Für ungerade n ist nicht mehr gegeben, dass jede Spalte durch genau eine Fuge besetzt wird. Genauer, nach Gleichung (1.4) bleiben in einer maximalen Lösung genau $r = \frac{1}{2}(n-1)$ Spalten offen.
Um das Backtracking auf alle Fälle zu erweitern, wird die Möglichkeit hinzugefügt, bestimmte Spalten auszulassen. Ist es in einem ungeraden Fall während des Backtracking nicht möglich, die direkt folgende Spalte s zu füllen, so kann diese Spalte *übersprungen* werden. Dazu wird nicht versucht, die Spalte s zu füllen, sondern die darauf folgende Spalte $s+1$.

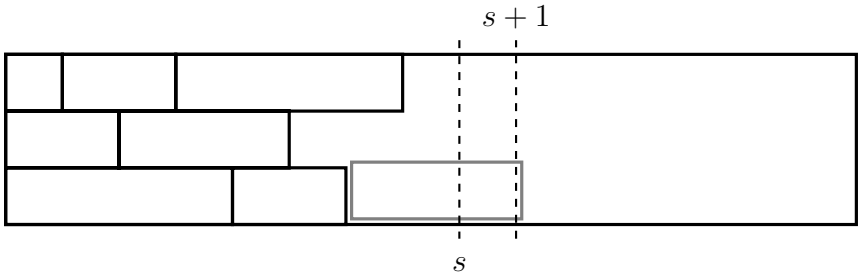

Abbildung 5: Überspringen einer Spalte

Ist es auch nicht möglich, die Spalte $s+1$ zu füllen, so wird diese ebenfalls übersprungen. Es ist demnach möglich, auch mehrere Spalten in einem Schritt zu überspringen. Sei z die Anzahl von übersprungenen Spalten in einem Schritt.
Während des Backtrackings soll r die Anzahl von Spalten darstellen, die noch übersprungen werden können. Anfänglich ist r die durch Gleichung (1.4) gegebene Insgesamtanzahl, insofern n ungerade ist. Handelt es sich um einen geraden Fall, so dürfen keine Spalten übersprungen werden, womit $r = 0$.

$$r = \begin{cases} 0 & (2 \mid n) \\ \frac{1}{2}(n-1) & (2 \nmid n) \end{cases} \tag{2.4}$$

Wurden z Spalten übersprungen, so verringert sich r um diese Zahl. Es können mindestens 0 und maximal r Spalten in einem Schritt übersprungen werden. Da nun nicht immer die direkt folgende Spalte als betrachtet wird, müssen die Gleichungen (2.2) und (2.3) für den Übergang zum nächsten Schritt erweitert werden.

$$b' = s + z - \ell(R) \qquad (b' \in \Sigma \backslash R \ \wedge \ 0 \leq z \leq r) \tag{2.5}$$

$$R' = R \cup \{b'\} \qquad \ell' = \ell + b' \qquad s' = s + z + 1 \qquad r' = r - z \tag{2.6}$$

Ein Schritt setzt sich nun zusammen aus einer Reihe $R \in W$ und einer Anzahl von Sprüngen $z \in Z$ mit $Z = \{0, \ldots, r\}$. Die Menge aller Folgeschritte ist die Menge von allen Kombinationen $W \times Z$. Dazu werden die Iterationen über die Reihen und über die Sprunglängen geschachtelt. Die äußere Iteration geht über z aufsteigend. Es werden demnach immer zuerst die Möglichkeiten mit der geringsten Anzahl von Sprüngen betrachtet. Für den erweiterten Algorithmus siehe Anhang (2).

2.4 Suchheuristik

Mit dem Backtracking wird ein Suchbaum abgelaufen. Jeder Knoten in diesem Baum stellt einen Füllzustand der Wand dar, während ein Schritt von einem Zustand zum nächsten eine Kante ist. Ist kein Schritt von einem Knoten aus mehr möglich, so handelt es sich um eine *Sackgasse*. Ist die Mauer vollständig gefüllt, so ist der zugehörige Zustands-Knoten eine *Lösung*. Alle Blätter des Baums sind entweder Sackgassen oder Lösungen.

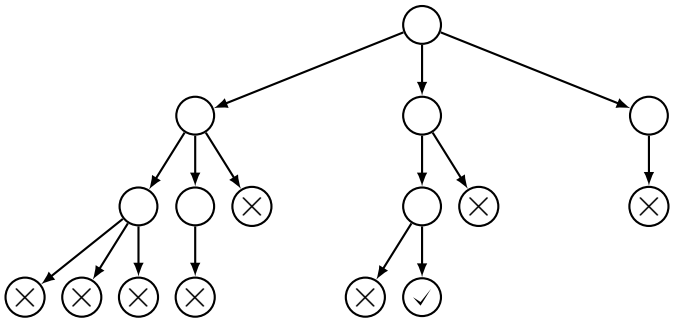

Abbildung 6: Suchbaum

Das Backtracking geht den gesamten Baum ab, bis eine Lösung gefunden wurde. Die Größe des Baums, der bis zur ersten gefundenen Lösung betrachtet werden musste, ist maßgebend für die Laufzeit der Suche. Dabei ist die Reihenfolge entscheidend, in der die einzelnen Möglichkeiten abgelaufen werden.

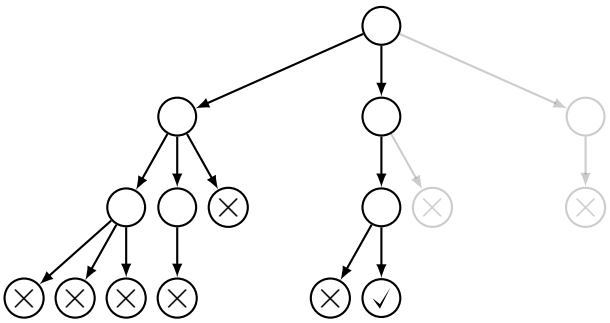

Abbildung 7: Betrachteter Teilbaum von links nach rechts

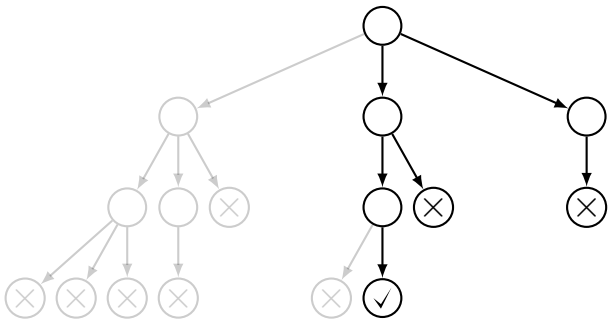

Abbildung 8: Betrachteter Teilbaum von rechts nach links

Um die Laufzeit der Suche zu verringern, wird eine Suchheuristik angewendet, welche die Reihenfolge der betrachteten Möglichkeiten beeinflusst. Eine solche Heuristik sollte die Schritte zuerst stellen, die eine höhere Warscheinlichkeit haben zielführend zu sein.

3 Anhang

Data: n *Eingabe für gerade n*

Result: solved W *vollständige Mauer ohne Überschneidungen*

$h \leftarrow \frac{1}{2}n + 1$ *Gl.(1.1)[TODO INDENTS]*
$w \leftarrow \frac{1}{2}n(n+1)$ *Gl. (1.3)*
$W \leftarrow \{\emptyset\}^h$
$s \leftarrow 1$
def putStone($R,\ b$)**:** *Rekursive Funktion*
    $s \leftarrow s + 1$
    $\ell(R) \leftarrow \ell(R) + b$
    $R \leftarrow R \cup \{b\}$
    if $s = w$**:**
        return True
    for R' **in** W**:**
        $b' \leftarrow s - \ell(R')$ *Gl. (2.2)*
        if $b' \notin R'$**:**
            if putStone($R',\ b'$)**:**
                return True
    $s \leftarrow s - 1$
    $R \leftarrow R \backslash \{b\}$
    $\ell(R) \leftarrow \ell(R) - b$
    return False

for R **in** W**:**
    putStone($R,\ 1$)

Algorithm 1: Backtracking für gerade n

Data: n *Eingabe für gerade n*

Result: solved W *vollständige Mauer ohne Überschneidungen*

$h \leftarrow \left\lfloor \frac{1}{2}n \right\rfloor + 1$ *Gl.(1.1)[TODO INDENTS]*
$w \leftarrow \frac{1}{2}n(n+1)$ *Gl. (1.3)*
$W \leftarrow \{\emptyset\}^h$
if $2 \mid n$**:**
    $r \leftarrow 0$
else:
    $r \leftarrow \frac{1}{2}(n-1)$
$s \leftarrow 1$
def putStone($R,\ b,\ z$)**:** *Rekursive Funktion*
    $s \leftarrow s + z + 1$
    $\ell(R) \leftarrow \ell(R) + b$
    $R \leftarrow R \cup \{b\}$
    if $s + z = w$**:**
        return True
    $r \leftarrow r - z$
    $Z \leftarrow \{0,\ \ldots,\ r\}$
    for $(R',\ z')$ **in** $W \times Z$**:**
        $b' \leftarrow s + z' - \ell(R')$ *Gl. (2.2)*
        if $b' \notin R'$**:**
            if putStone($R',\ b',\ z'$)**:**
                return True
    $s \leftarrow s - 1$
    $r \leftarrow r + z$
    $R \leftarrow R \backslash \{b\}$
    $\ell(R) \leftarrow \ell(R) - b$
    return False

for R **in** W**:**
    putStone($R,\ 1,\ 0$)

Algorithm 2: Backtracking für alle n