252-0027

Einführung in die Programmierung

Übungen

Woche 7: Hoare-Logik, Problem Solving

Timo Stucki
Departement Informatik
ETH Zürich

Organisatorisches

- Mein Name: *Timo Stucki*
- Bei Fragen: *tistucki@student.ethz.ch*
  - Mails bitte mit «[EProg25]» im Betreff
- Neue Aufgaben: **Dienstag Abend** (im Normalfall)
- Abgabe der Übungen bis **Dienstag Abend (23:59)** Folgewoche
  - Abgabe immer via Git
  - Lösungen in separatem Projekt auf Git

Discord: timostucki

Webseite

- **timostucki.com**

Weakest Precondition II

Weakest Precondition

- Die **schwächste Vorbedingung (weakest precondition)** ist die schwächste Vorbedingung, die die Postcondition impliziert.
  - Falls die Postcondition `{ true }` ist, so ist `{ true }` die schwächste Vorbedingung. Alles impliziert die Postcondition, also insbesondere auch die schwächste Bedingung `true`.
  - Falls die Postcondition `{ false }` ist, so ist `{ false }` die schwächste Vorbedingung. Nur `{ false }` impliziert die Postcondition, demensprechend ist es die schwächste (und einzige) Vorbedingung.
- Die vorgestellten Regeln fürs Rückwärtsschliessen ergeben direkt die schwächsten Vorbedingungen.

If-Anweisungen und Aussagen

- **Ziel:** Regel für Tripel {P} `if (b)` S_1 `else` S_2 {Q}

- **Beobachtungen**
  - Ausführung von S_1 wenn b hält
  - Ausführung von S_2 wenn ¬b hält
  - P hält in beiden Fällen vor der Ausführung
  - Q muss in beiden Fällen nachher gelten

```
{P}
  if (b) {
    S1;
  } else {
    S2;
  }
{Q}
```

Hoare-Logik-Regel für if-Anweisungen

- Das Tripel $\{P\}\ \texttt{if (b)}\ S_1\ \texttt{else}\ S_2\ \{Q\}$ ist gültig, genau dann wenn
  1. $\{P \wedge b\}\ S_1\ \{Q\}$ gültig ist
  2. $\{P \wedge \neg b\}\ S_2\ \{Q\}$ gültig ist

Vorgehen für if-Anweisungen

- **Situation:** Entscheide, ob {P} `if (b)` S_1 `else` S_2 {Q} gültig ist

- **Empfohlenes Vorgehen:**
  1. Wende bekannte Regeln wie rechts gezeigt an (auf S_1 und S_2)
  2. Zeige notwendige Implikationen
     1. $P \wedge b \implies R_1$
     2. $P \wedge \neg b \implies R_2$

```
if (b) {
  {P ∧ b}
  {R1}
    S1;
  {Q}
} else {
  {P ∧ ¬b}
  {R2}
    S2;
  {Q}
}
```

Beispiel 1: Gültiges Tripel

Zu zeigen: Gültigkeit von $\{true\}$ `if (x < 0) { y = -x; } else { y = x;}` $\{y \geq 0\}$

1. Regeln anwenden:

```
if (x < 0) {
  {true ∧ x < 0}
  {-x ≥ 0}
    y = -x;
  {y ≥ 0}
} else {
  {true ∧ ¬(x < 0)}
  {x ≥ 0}
    y = x;
  {y ≥ 0}
}
```

2. Implikationen zeigen:
   1. $(x < 0) \implies (-x \geq 0)$ ✓
   2. $(x \geq 0) \implies (x \geq 0)$ ✓

Beispiel 2: Ungültiges Tripel

Geändert

Zu zeigen: Gültigkeit von $\{true\}$ `if (x < 0) { y = -x; } else { y = x;}` $\{y \geq 0\}$

1. Regeln anwenden:

```
if (x < 0) {
  {true ∧ x < 0}
  {-x ≥ 0}
    y = -x;
  {y ≥ 0}
} else {
  {true ∧ ¬(x < 0)}
  {x ≥ 0}
    y = x;
  {y ≥ 0}
}
```

Entsprechende Unterschiede

2. Implikationen zeigen:
   1. $(x < 0) \Longrightarrow (-x \geq 0)$ ✓
   2. $(x \geq 0) \Longrightarrow (x \geq 0)$ ✗

Hält nicht mehr (Gegenbeispiel: $x == 0$)

Beispiel 3: Ohne else-Zweig

Zu zeigen: Gültigkeit von $\{x == y \land x \mathrel{!}= 0\}$ `if (a > 1) { a = a * x; }` $\{a \mathrel{!}= y\}$

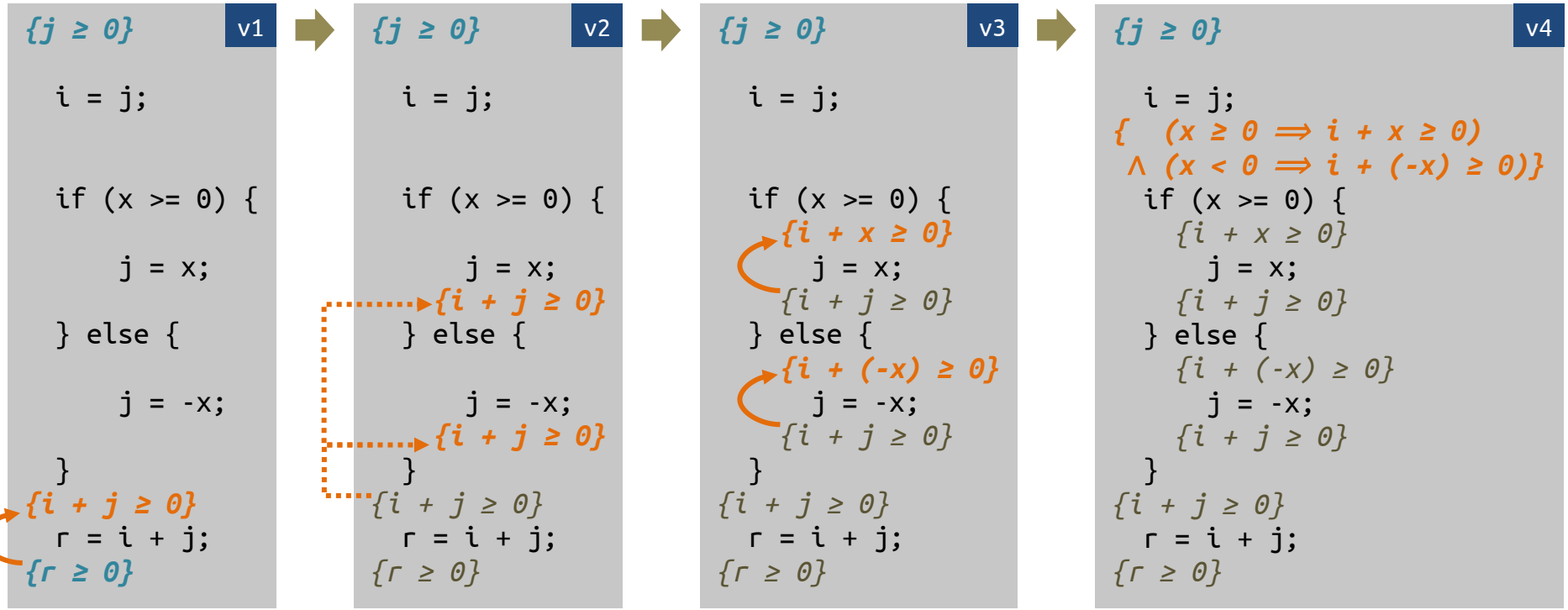
1. Regeln anwenden:

```
if (a > 1) {
  {x == y ∧ a > 1}
  {a * x ≠ y}
    a = a * x;
  {a ≠ y}
} else {
  {x == y ∧ ¬(a > 1)}
  {a ≠ y}
}
```

Leerer else-Block entspricht fehlendem else-Block

2. Implikationen zeigen:
   1. ✓ $(x == y \land x \mathrel{!}= 0 \land a > 1) \Longrightarrow (a * x \mathrel{!}= y)$
   2. ✗ $(x == y \land x \mathrel{!}= 0 \land a \leq 1) \Longrightarrow (a \mathrel{!}= y)$

Beispiel Schritt für Schritt



v4

v5

```
{j ≥ 0}
{  (x ≥ 0 ⟹ j + x ≥ 0)
 ∧ (x < 0 ⟹ j + (-x) ≥ 0) }
  i = j;
{  (x ≥ 0 ⟹ i + x ≥ 0)
 ∧ (x < 0 ⟹ i + (-x) ≥ 0) }
  if (x >= 0) {
    {i + x ≥ 0}
      j = x;
    {i + j ≥ 0}
  } else {
    {i + (-x) ≥ 0}
      j = -x;
    {i + j ≥ 0}
  }
{i + j ≥ 0}
  r = i + j;
{r ≥ 0}
```

Dann noch zu zeigen:

$$(j \geq 0) \implies \big((x \geq 0 \implies j + x \geq 0) \wedge (x < 0 \implies j + (-x) \geq 0) \big)$$

Fallunterscheidung (zwei Konjunkte):

1. Wenn $j \geq 0$ und $x \geq 0$, dann $j + x \geq 0$ ✓
2. Wenn $j \geq 0$ und $x < 0$, dann $j + (-x) \geq 0$ ✓

Weakest Precondition mit Verzweigung

Schwächste Vorbedingung - Beispiel

```
{                }

  if (x >= y){
    y = x
  }

{y >= x}
```

Weakest Precondition – Mit Verzweigung

{ }

```
if (x<5) {
        y = x*x;
} else {
        y = x+1;
}
```

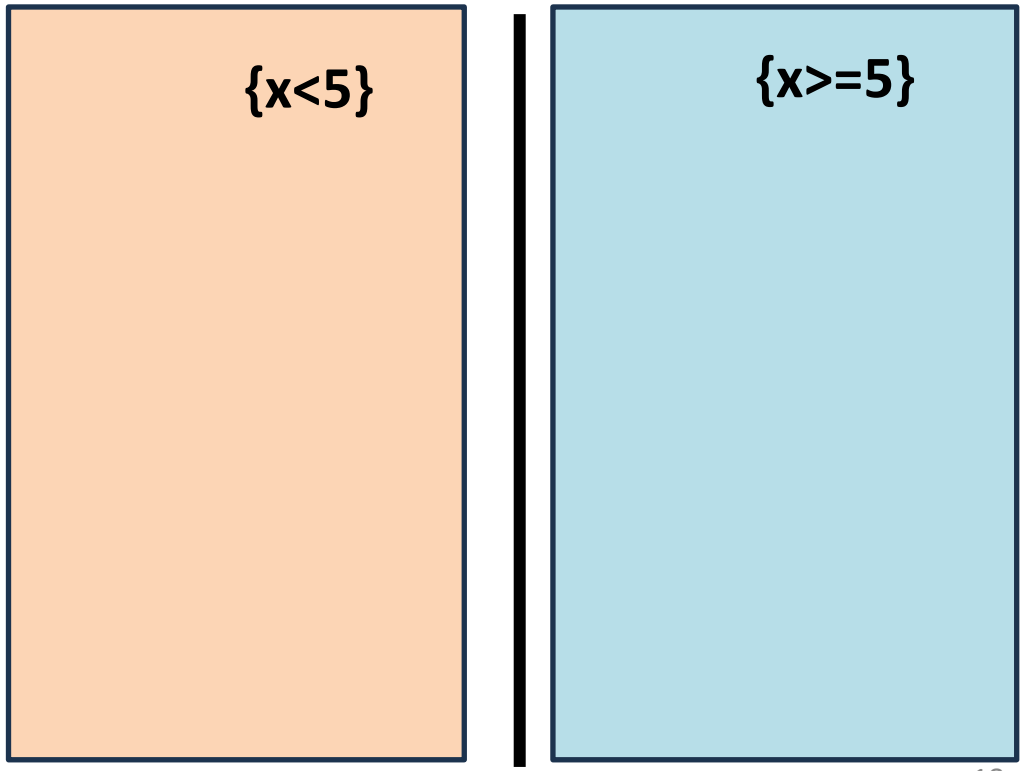
{y>9}

Weakest Precondition – Mit Verzweigung

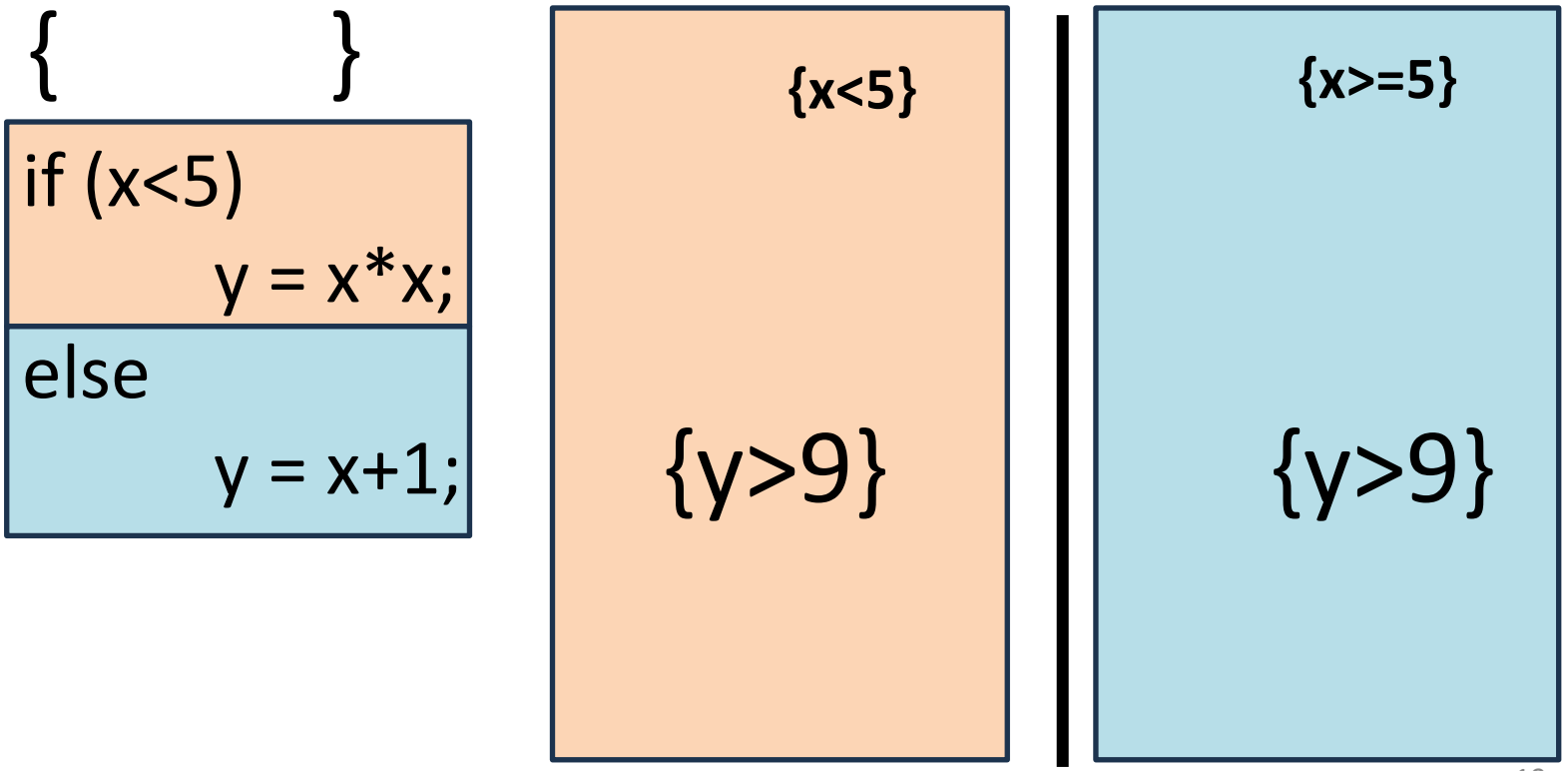
{ }

```
if (x<5)
        y = x*x;
else
        y = x+1;
```

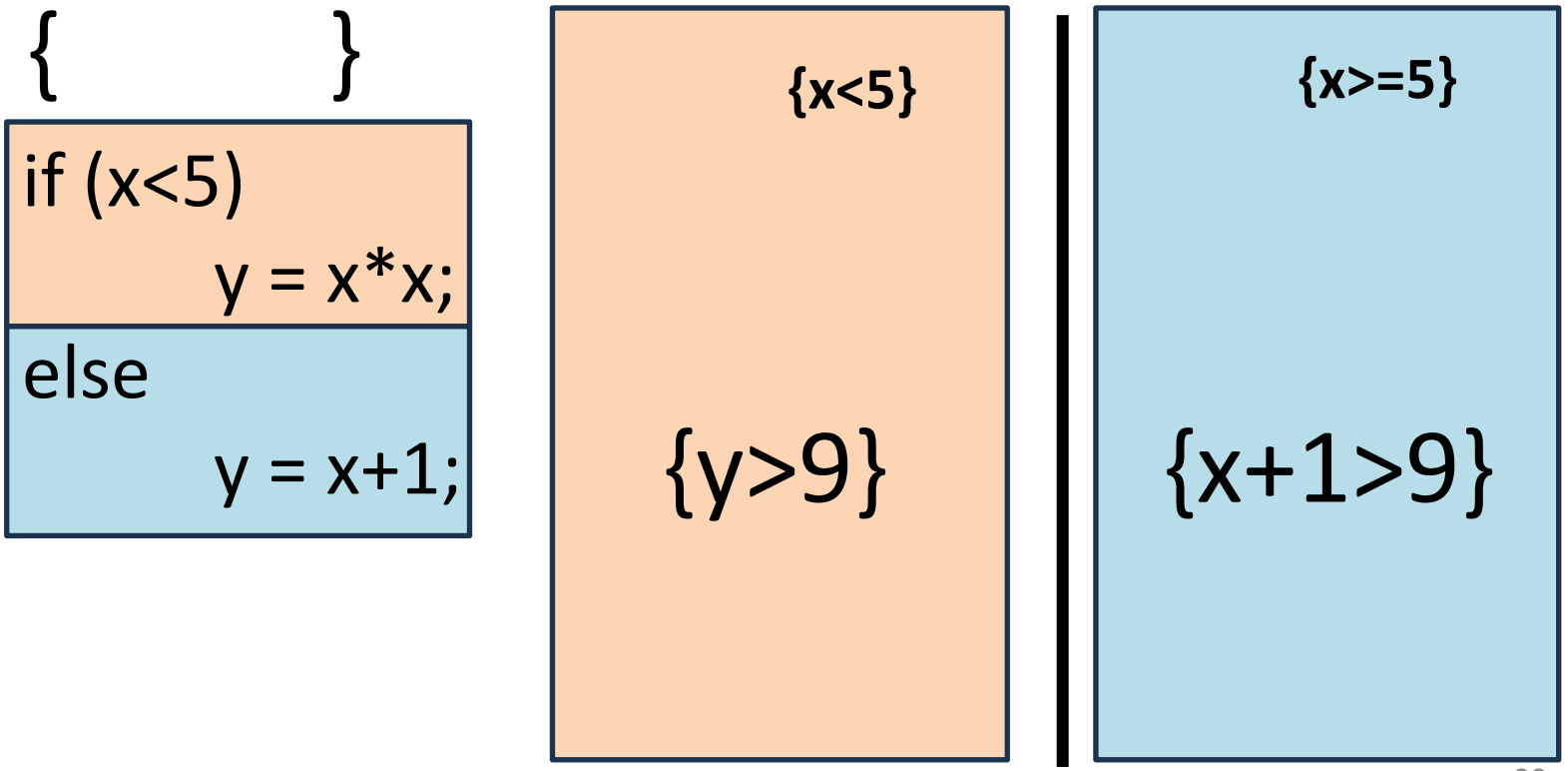
{y>9}



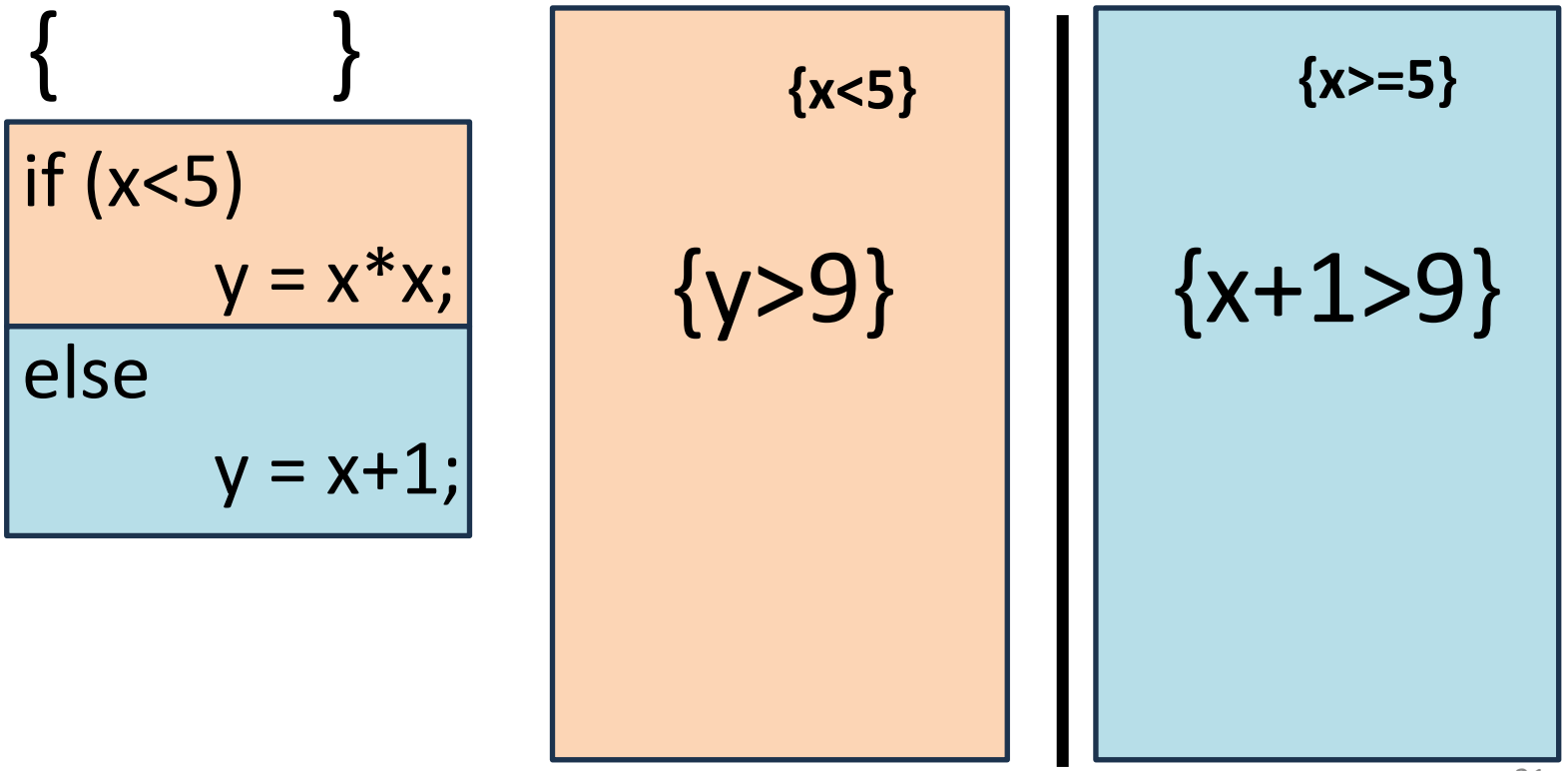
Weakest Precondition – Mit Verzweigung



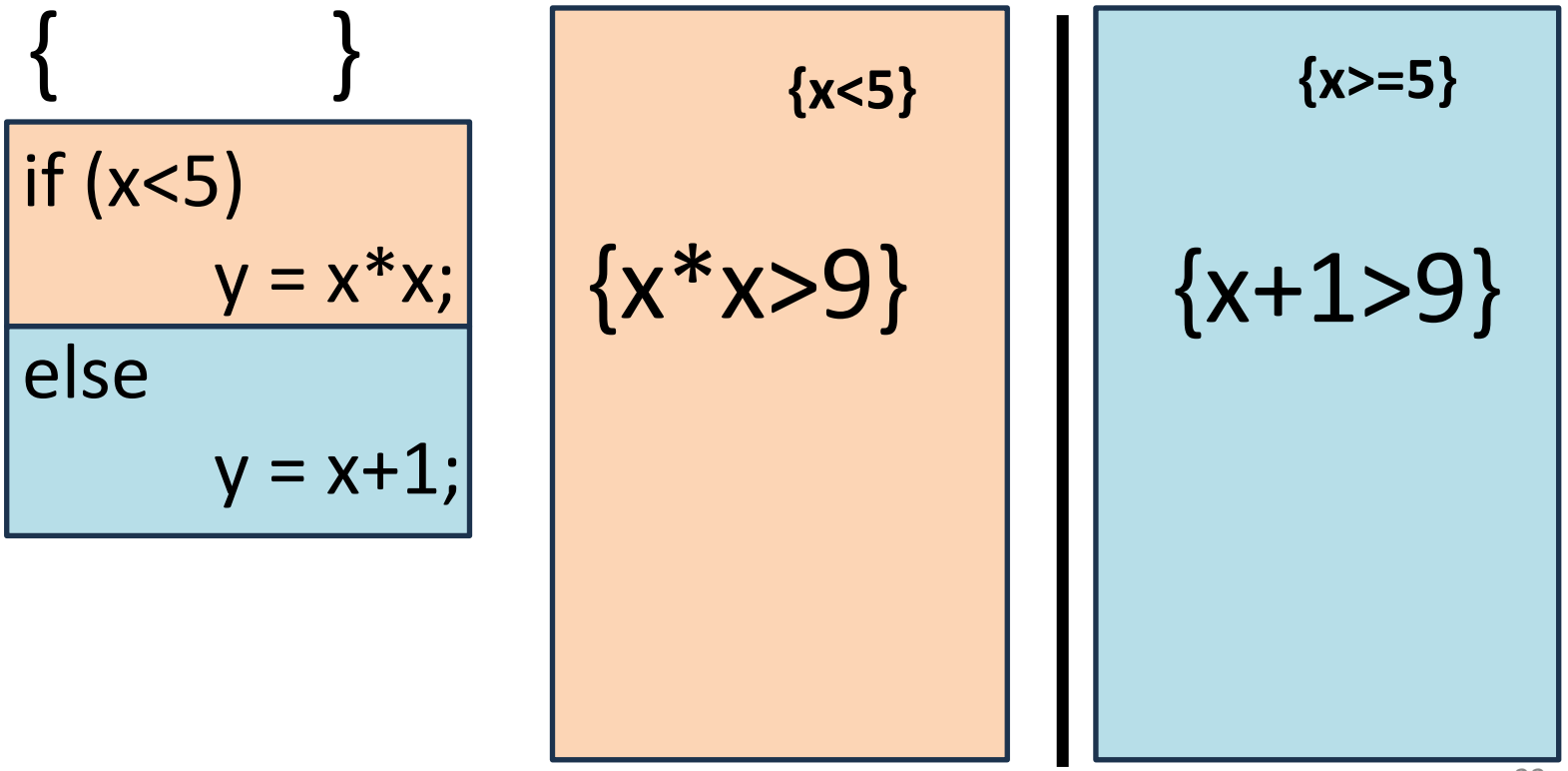
Weakest Precondition – Mit Verzweigung



Weakest Precondition – Mit Verzweigung



Weakest Precondition – Mit Verzweigung



{(**x<5** && x*x>9) || (**x>=5** && x+1>9)}

{ }

```
if (x<5)
        y = x*x;
else
        y = x+1;
```

{x<5}

{x*x>9}

||

{x>=5}

{x+1>9}

Weakest Precondition - Prüfungsbeispiel

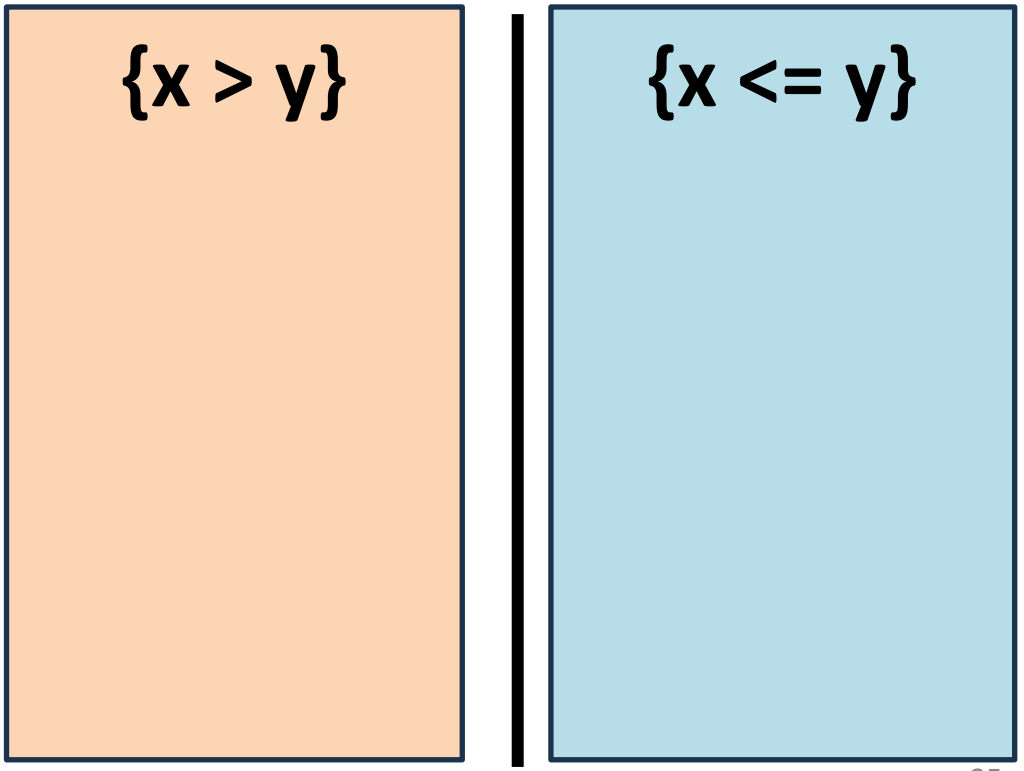
```
{              }
 if (x > y)
        z = x – y;
 else
        z = y – x;

{  z > 0  }
```

Weakest Precondition - Prüfungsbeispiel

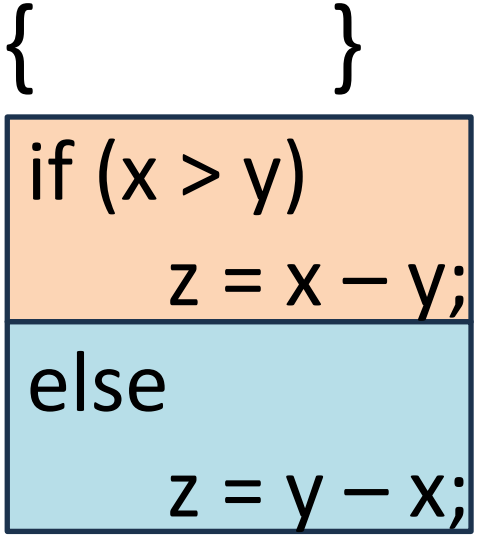
{ }

```
if (x > y)
        z = x – y;
else
        z = y – x;
```

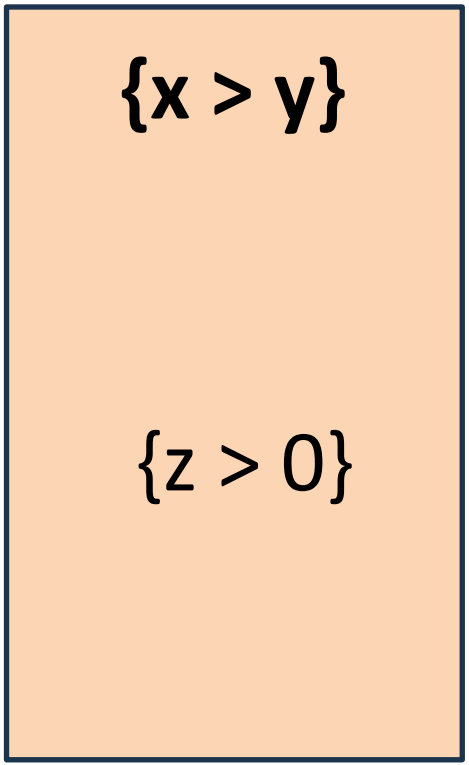
{ z > 0 }

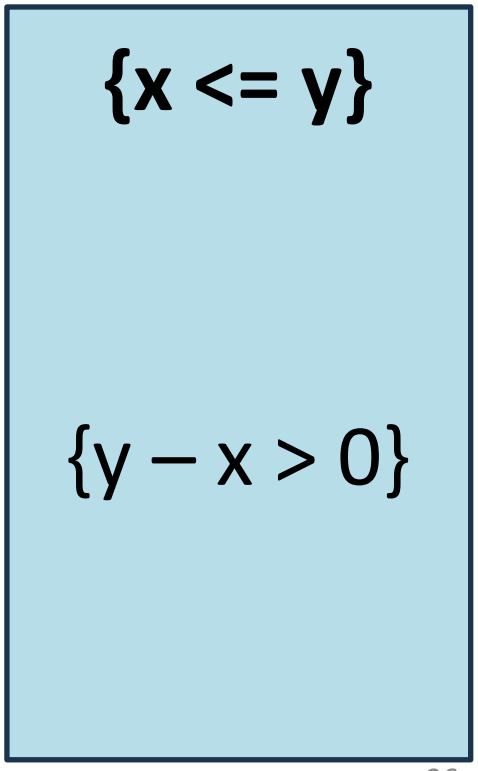


Weakest Precondition - Prüfungsbeispiel



{ z > 0 }



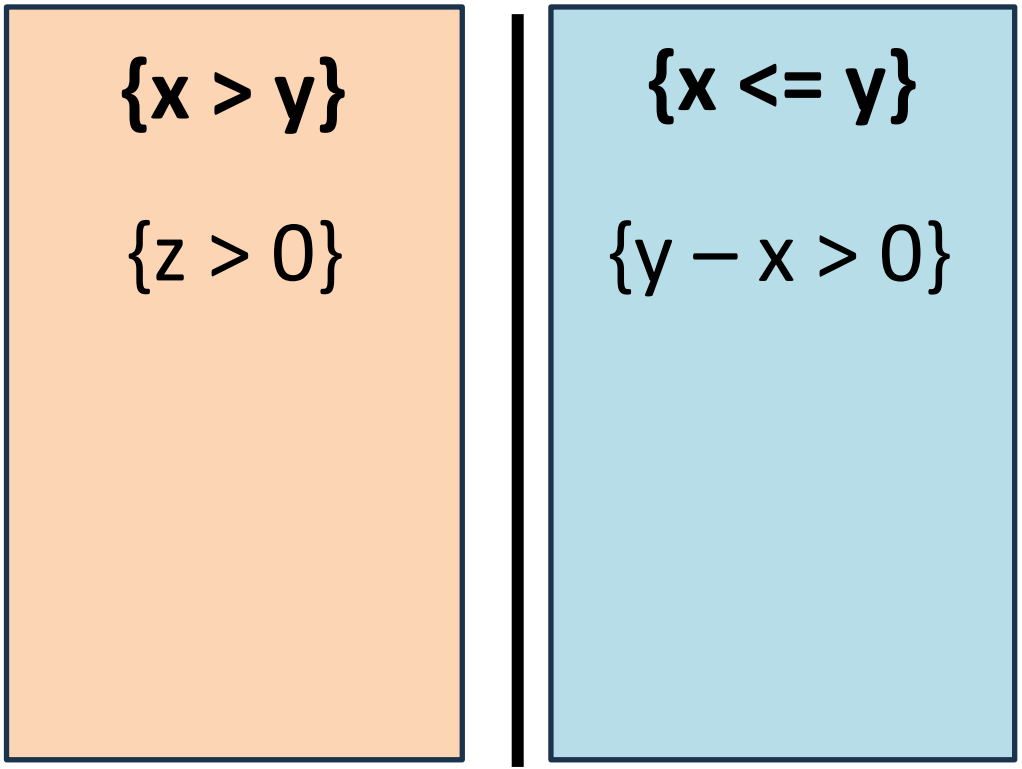


Weakest Precondition - Prüfungsbeispiel

{ }

```
if (x > y)
        z = x – y;
else
        z = y – x;
```

{ z > 0 }

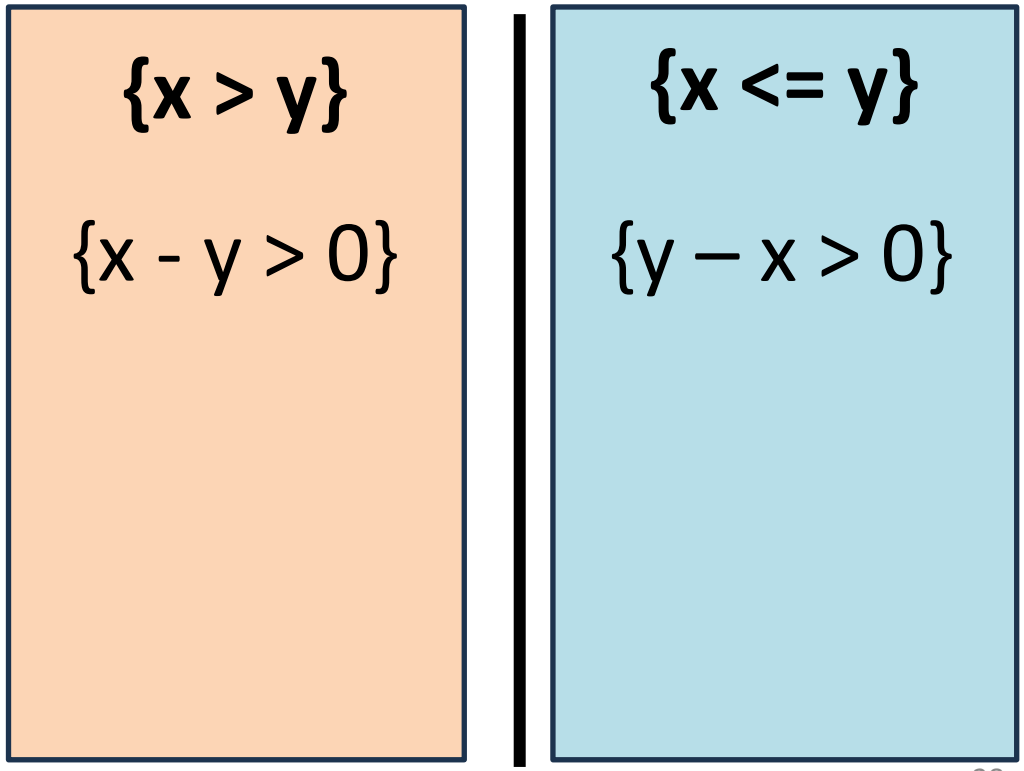


Weakest Precondition - Prüfungsbeispiel

{ }

```
if (x > y)
        z = x – y;
else
        z = y – x;
```

{ z > 0 }

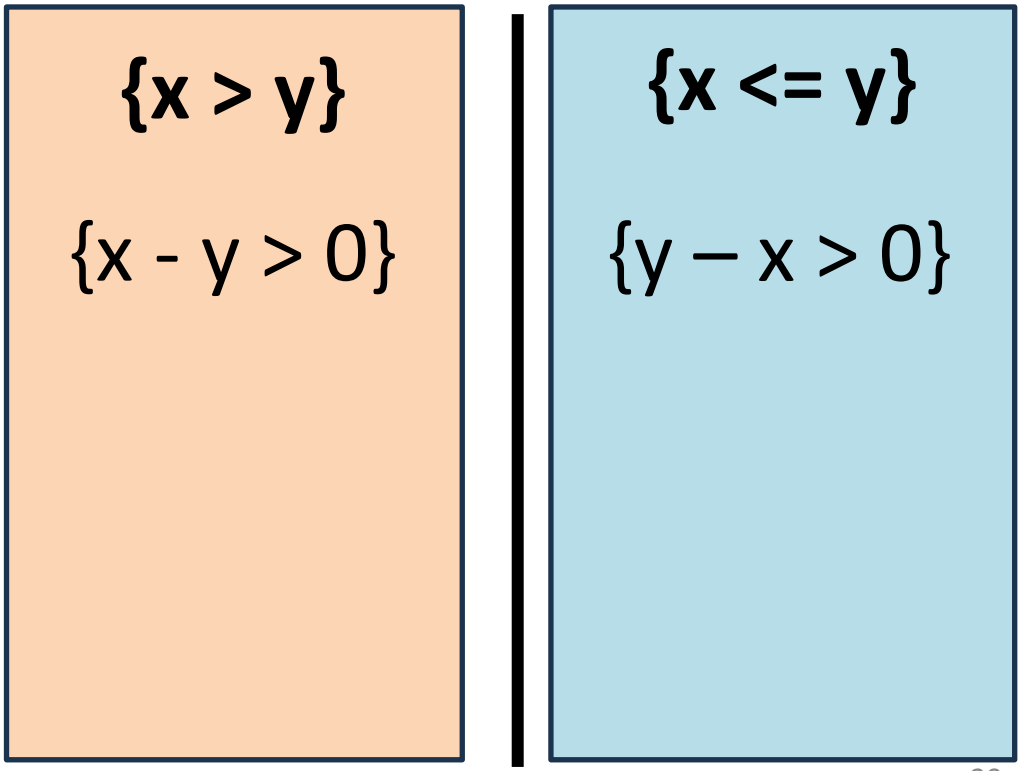


Weakest Precondition - Prüfungsbeispiel

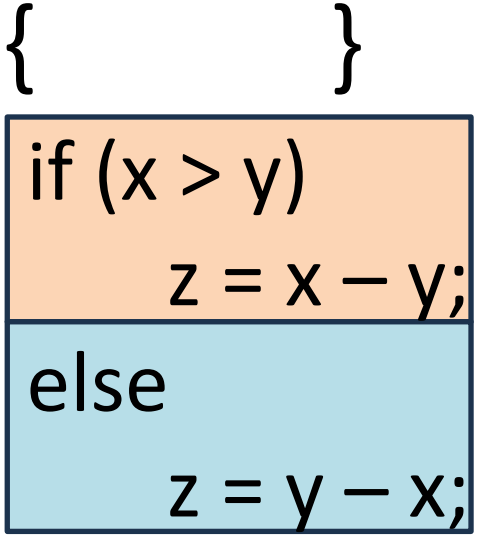
{ }

```
if (x > y)
        z = x – y;
else
        z = y – x;
```

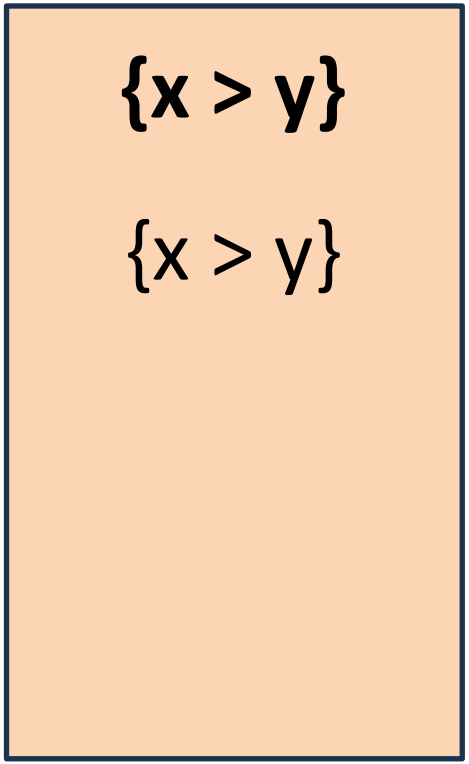
{ z > 0 }

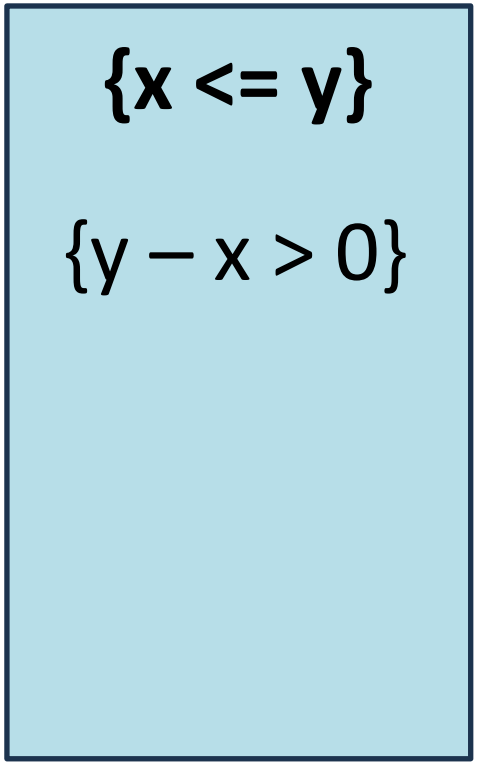


Weakest Precondition - Prüfungsbeispiel



{ z > 0 }





{((x > y) && (x > y)) || ((x <= y) &&(y > x))}

{ }

if (x > y)
    z = x – y;

else
    z = y – x;

{ z > 0 }

{x > y}

{x > y}

{x <= y}

{y > x}

Hoare Triple

Hoare Tripel – Prüfungsbeispiel HS18

```
{ b > c }
 if (x > b) {
     a = x;
} else {
     a = b;
}
{ a > c }
```

Hoare Tripel – Prüfungsbeispiel HS18

```
{ true }
 if (x > y) {
     y = x;
} else {
     y = -x;
}
{ y >= x}
```

Hoare Tripel – Prüfungsbeispiel HS18

```
{ x > 0 }
    y = x * x;
    z = y / 2;
{ z > 0 }
```

Hoare Logik FAQ

- Müssen wir unsere Ausdrücke vereinfachen?
  - Solange die Lösung **logisch äquivalent** ist, ist sie korrekt.

- Müssen wir unsere Herleitung zeigen?
  - Solange dies **nicht explizit gefragt** wird, ist dies nicht nötig.

- Müssen Gegenbeispiele alle Variablen enthalten, wenn nur eine davon relevant ist?
  - Wenn z.B. y != 0 immer bedeutet, das aus einer Precondition nicht die Postcondition folgt, dann ist dies ausreichend.

Umfrage

1 min, keine offenen Fragen ;)

https://docs.google.com/forms/d/e/1FAIpQLSeFu1As2BrAUauMFlklRiz0DMPt2MeEo6p40vxVUYjRagrmFw/viewform?usp=dialog

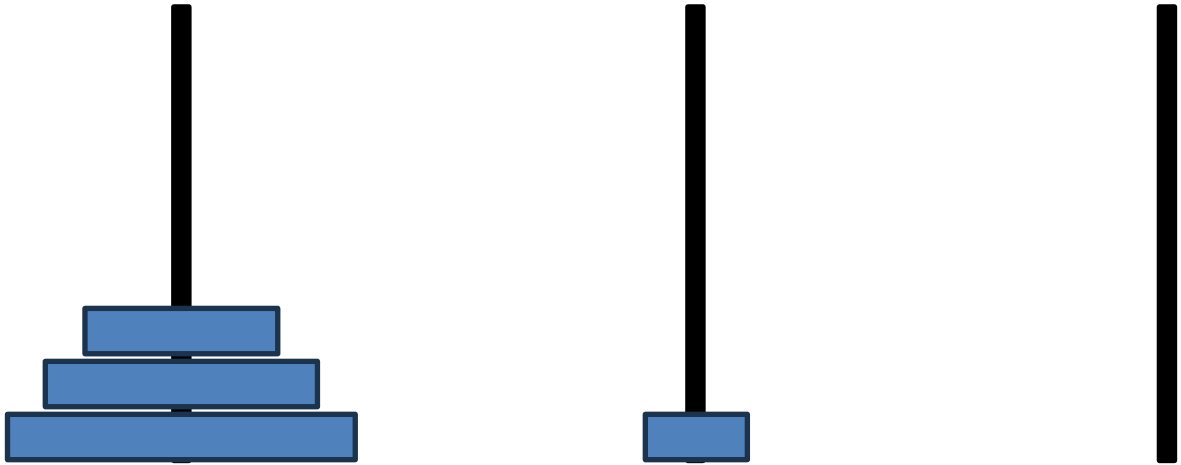
Towers of Hanoi

Towers of Hanoi - Beschreibung

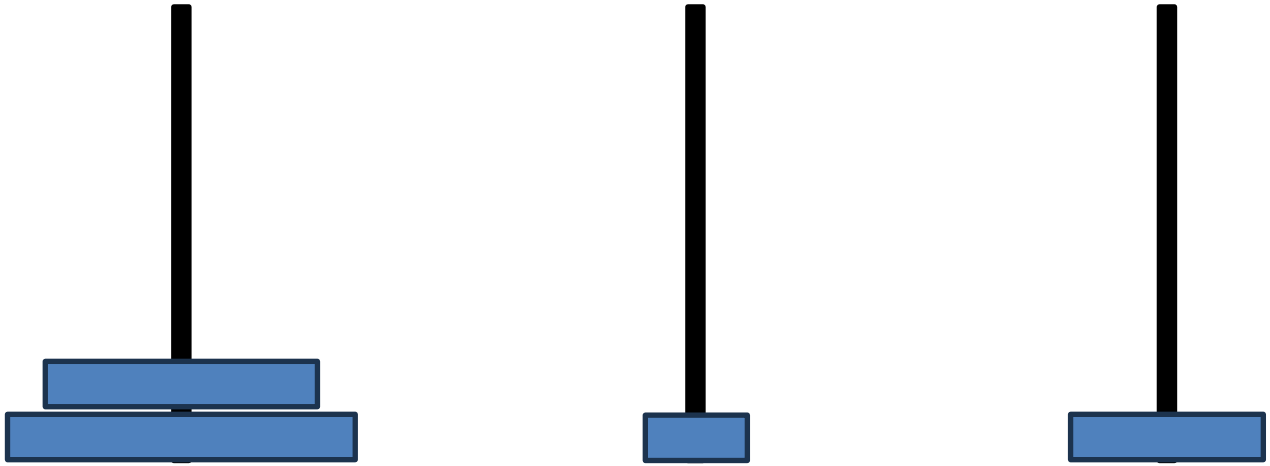
Das Türme von Hanoi-Problem besteht aus drei Stäben und einer Anzahl von unterschiedlich grossen Scheiben, die zu Beginn auf einem Stab gestapelt sind, wobei die groesste Scheibe unten und die kleinste oben liegt. Ziel ist es, alle Scheiben von einem Ausgangsstab auf einen Zielstab zu bewegen, wobei nur eine Scheibe auf einmal bewegt werden darf und niemals eine groessere Scheibe auf eine kleinere gelegt werden darf.

Towers of Hanoi - Beispiel

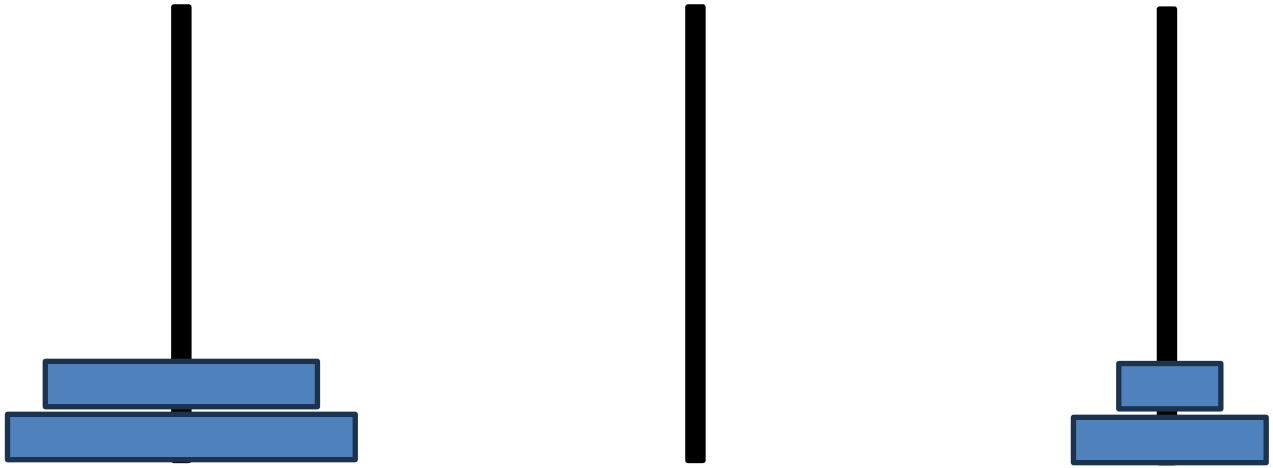
Towers of Hanoi - Beispiel



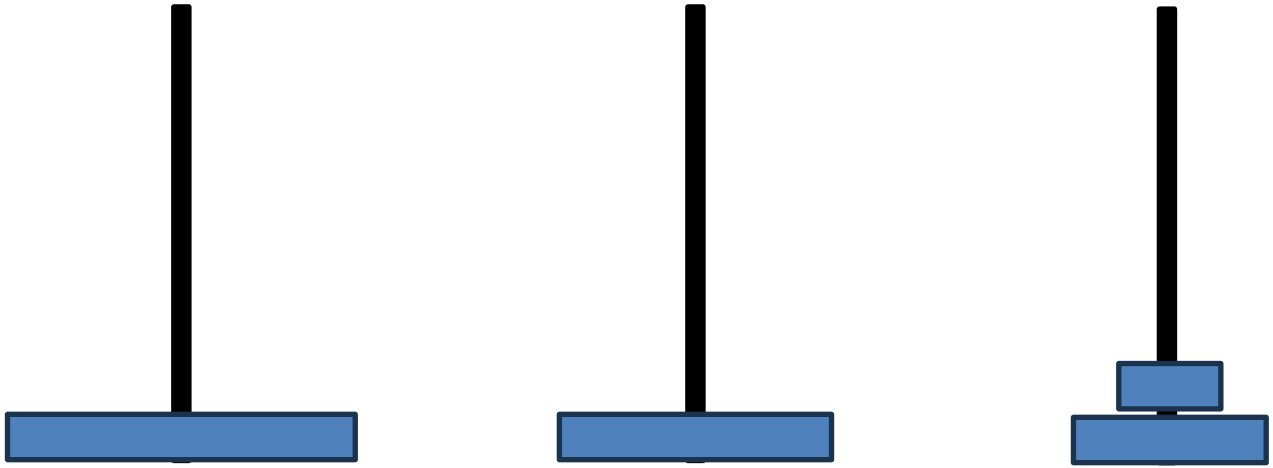
Towers of Hanoi - Beispiel



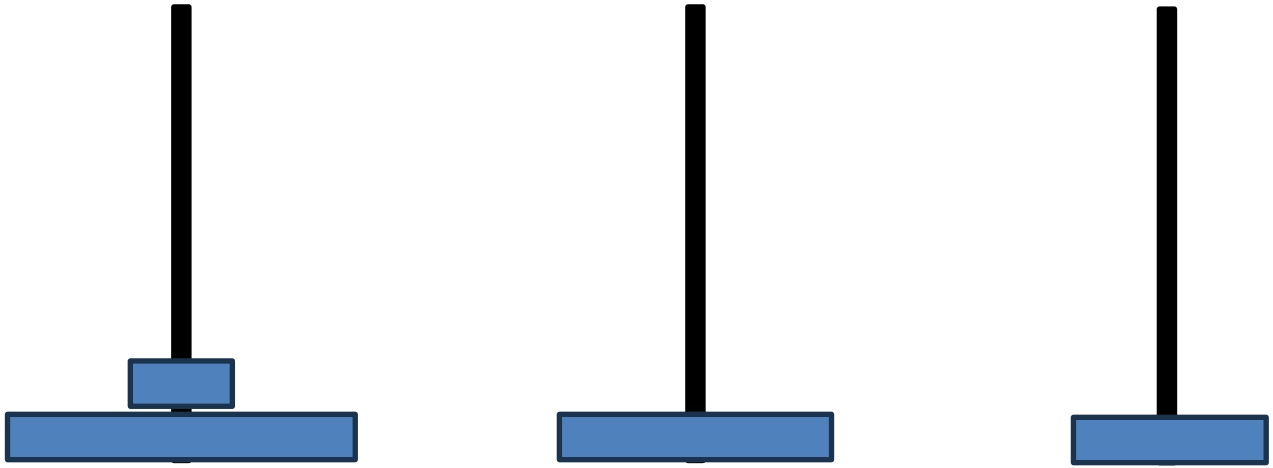
Towers of Hanoi - Beispiel



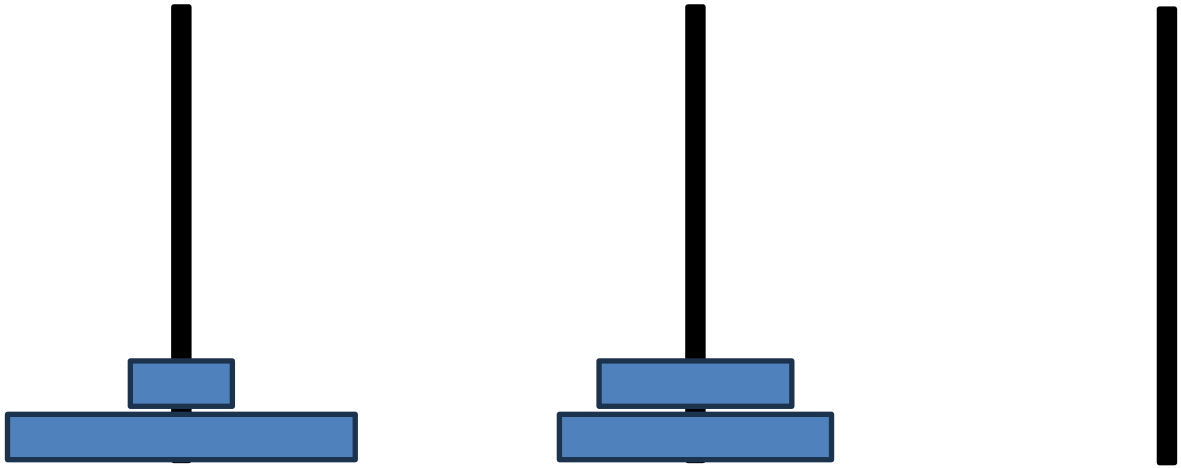
Towers of Hanoi - Beispiel



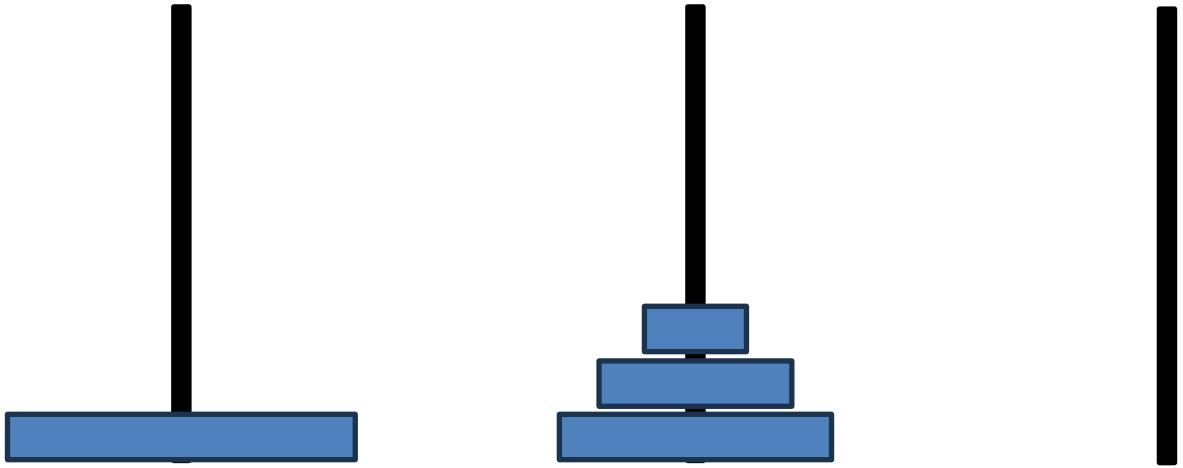
Towers of Hanoi - Beispiel



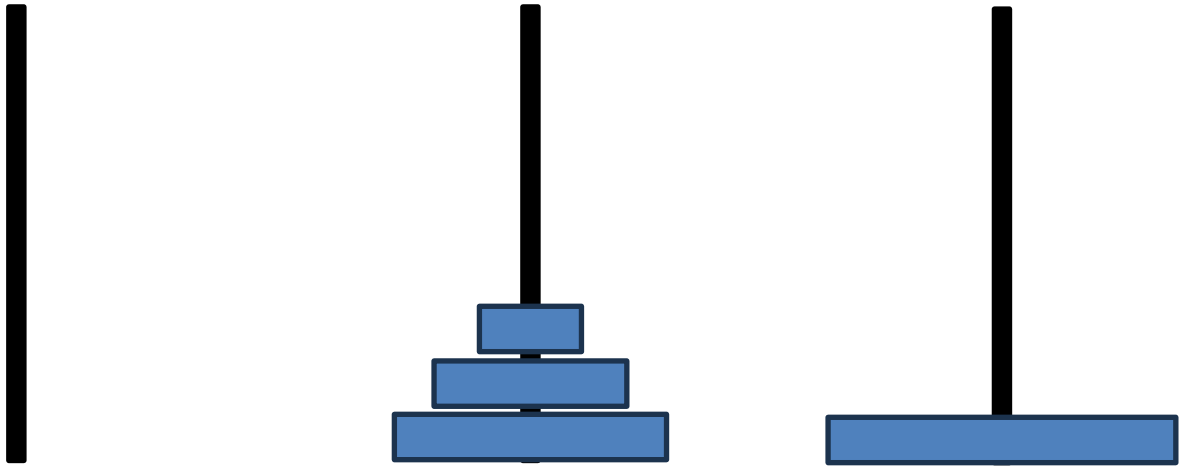
Towers of Hanoi - Beispiel



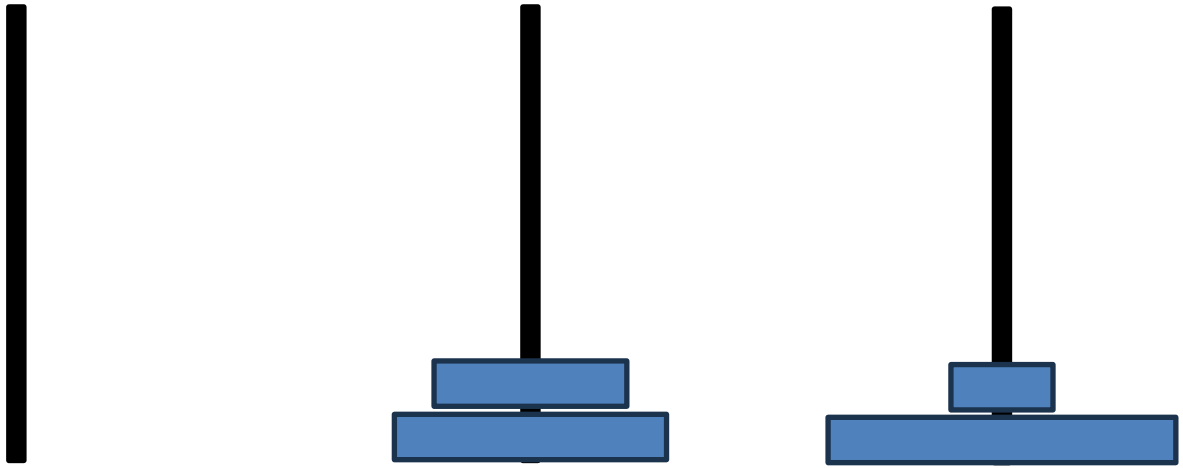
Towers of Hanoi - Beispiel



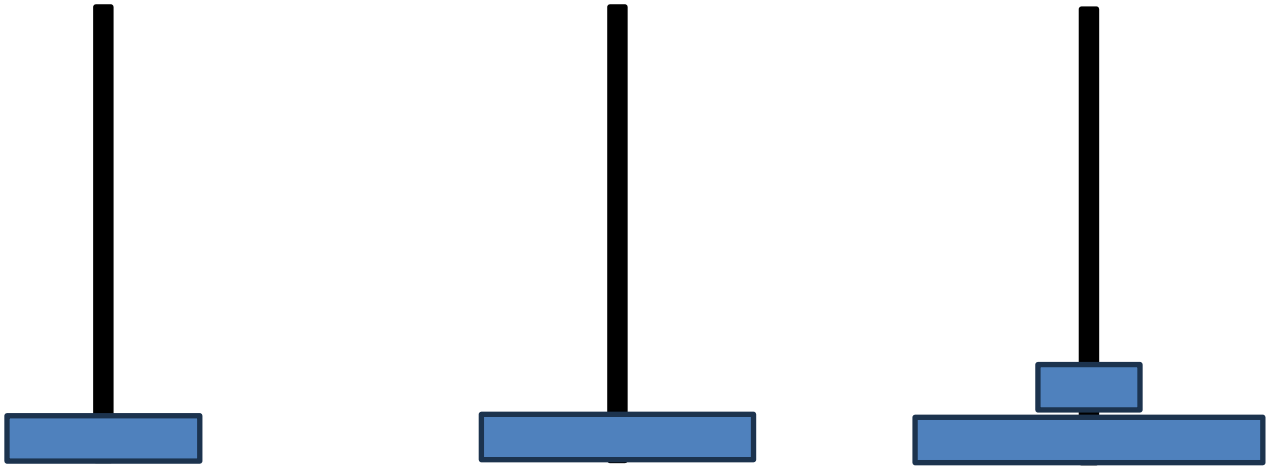
Towers of Hanoi - Beispiel



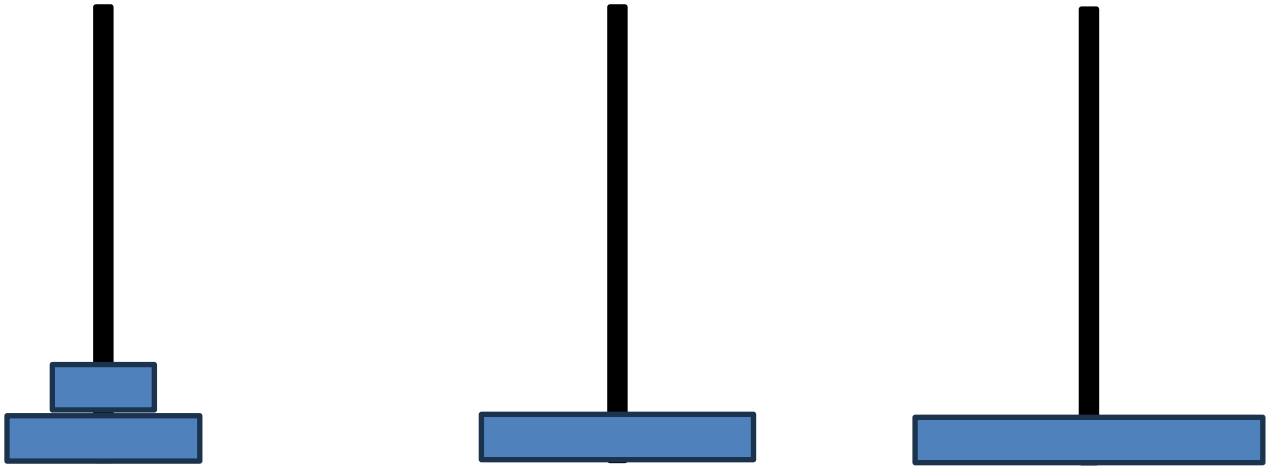
Towers of Hanoi - Beispiel



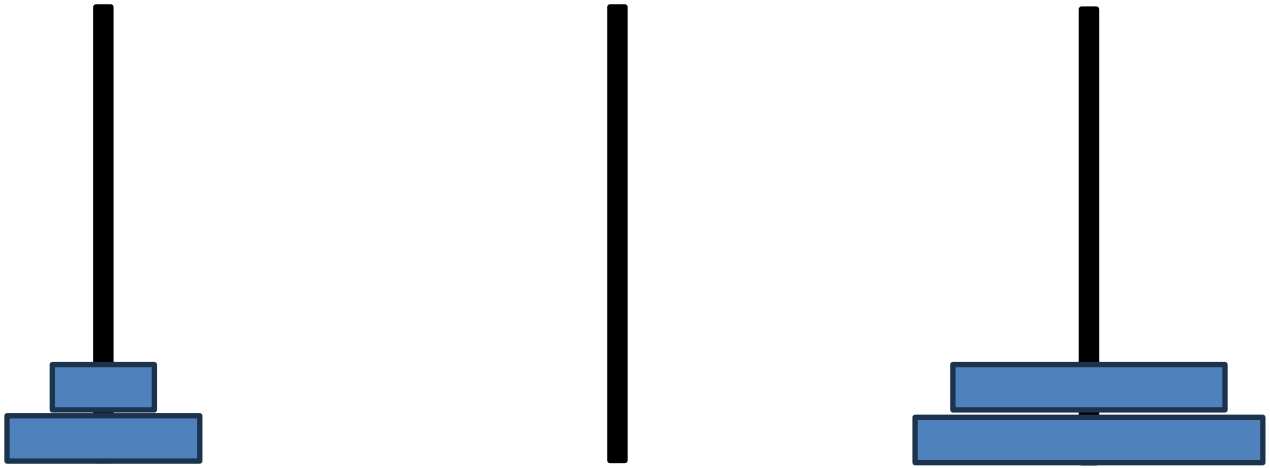
Towers of Hanoi - Beispiel



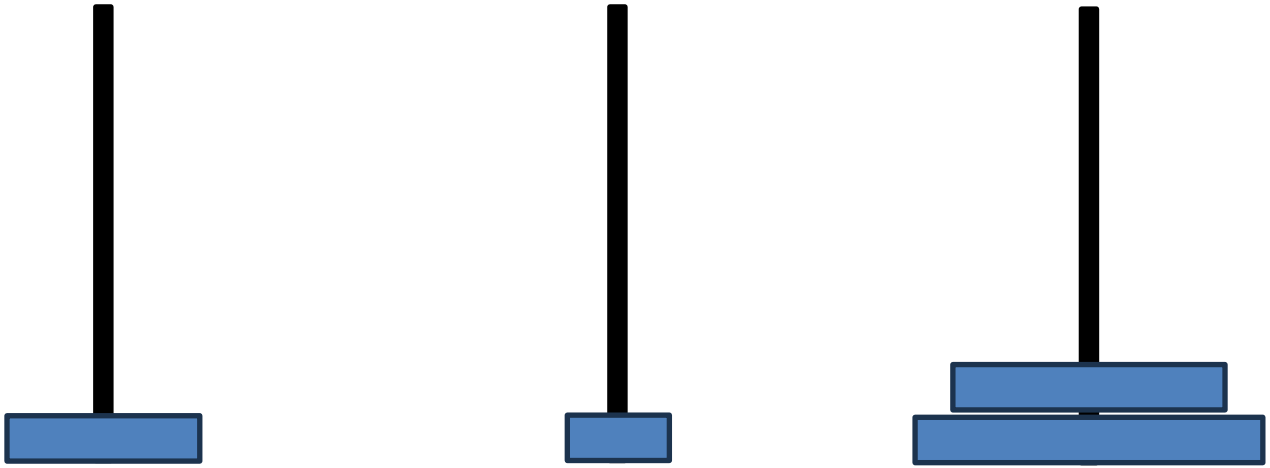
Towers of Hanoi - Beispiel



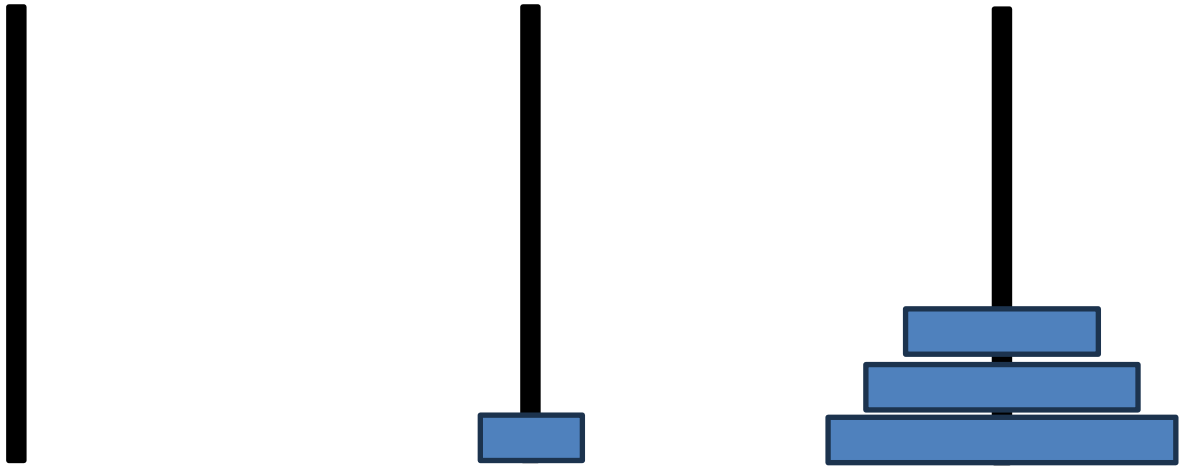
Towers of Hanoi - Beispiel



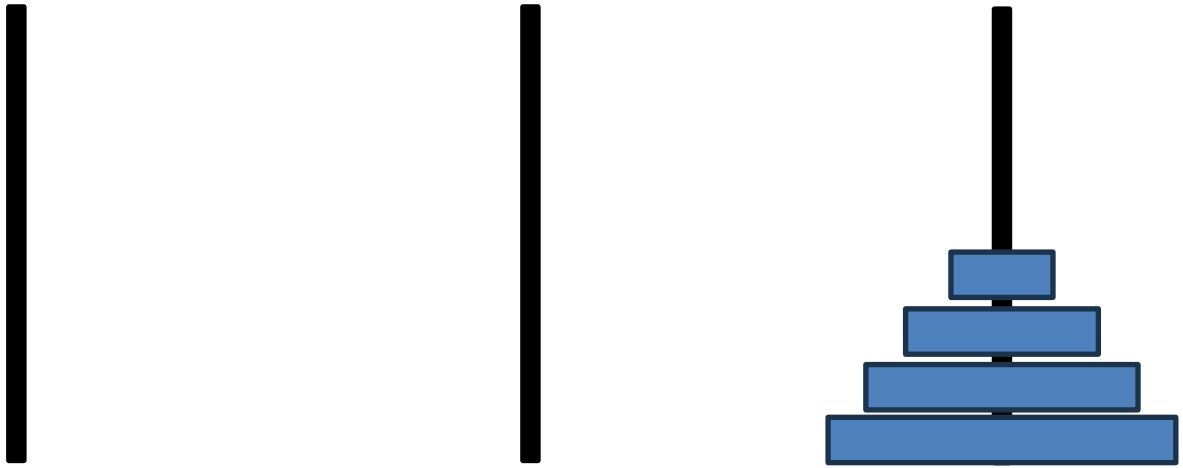
Towers of Hanoi - Beispiel

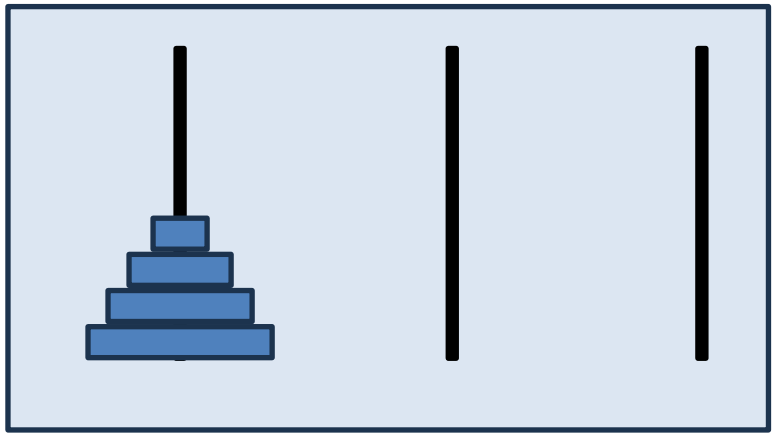


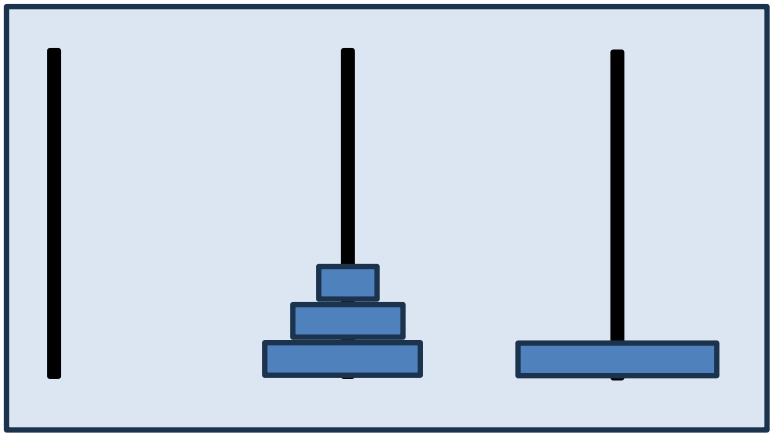
Towers of Hanoi - Beispiel



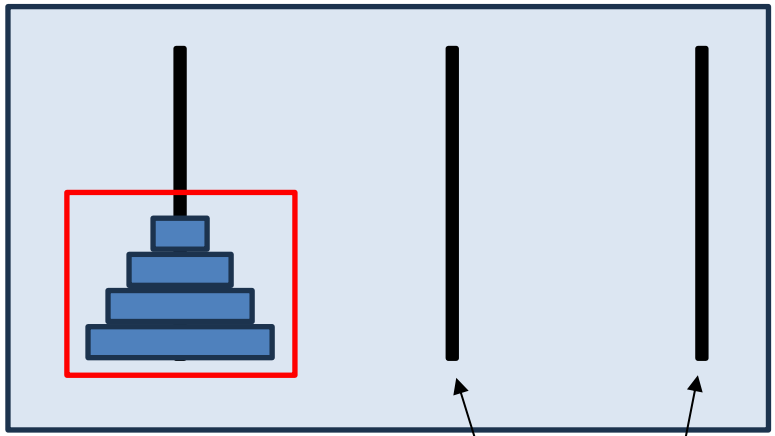
Towers of Hanoi - Beispiel

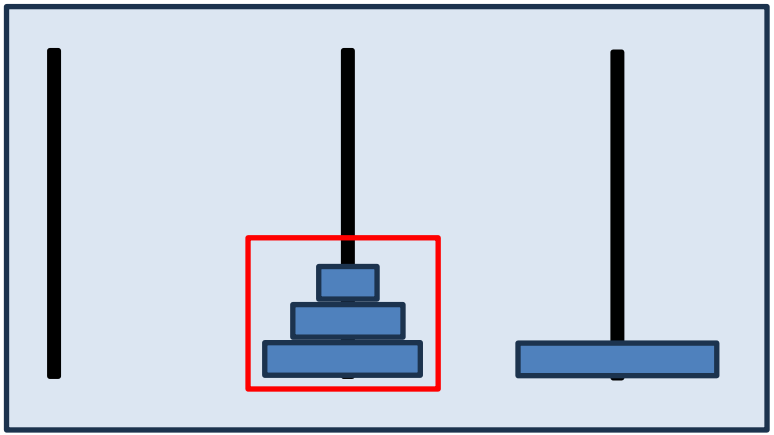




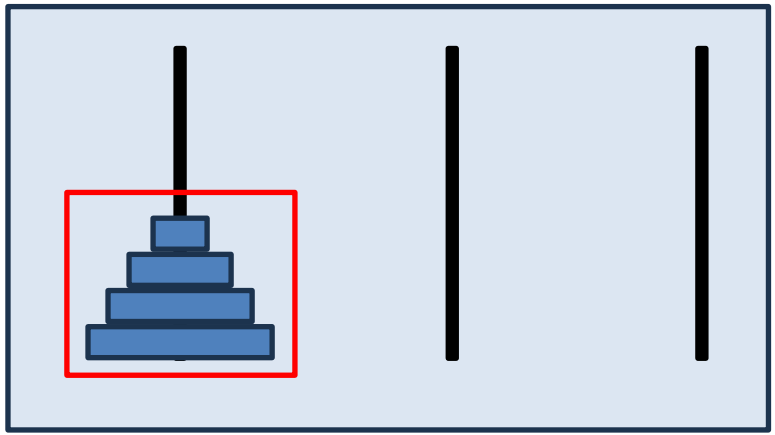


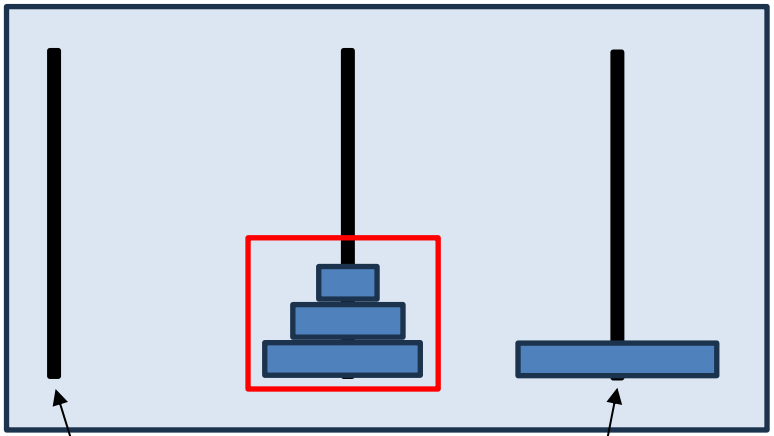
Was haben die beiden Konstellationen gemeinsam?



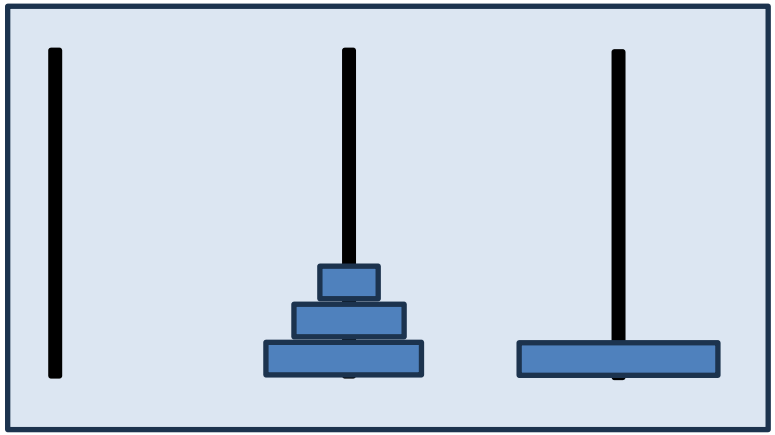


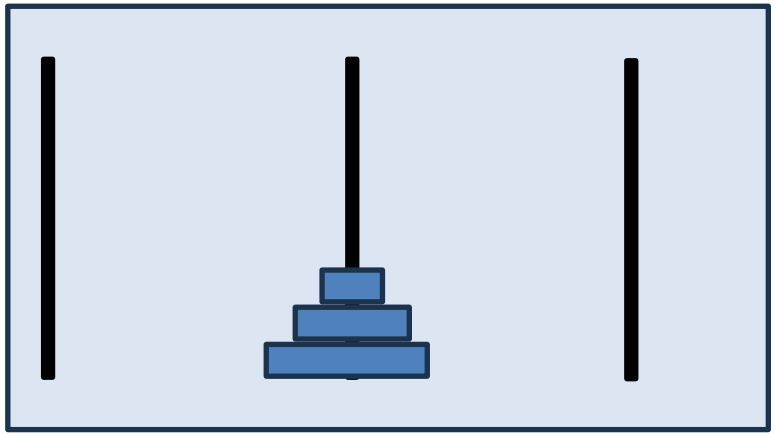
Wir können jede Scheibe in der roten Box auf den verbleibenden Stäben platzieren.



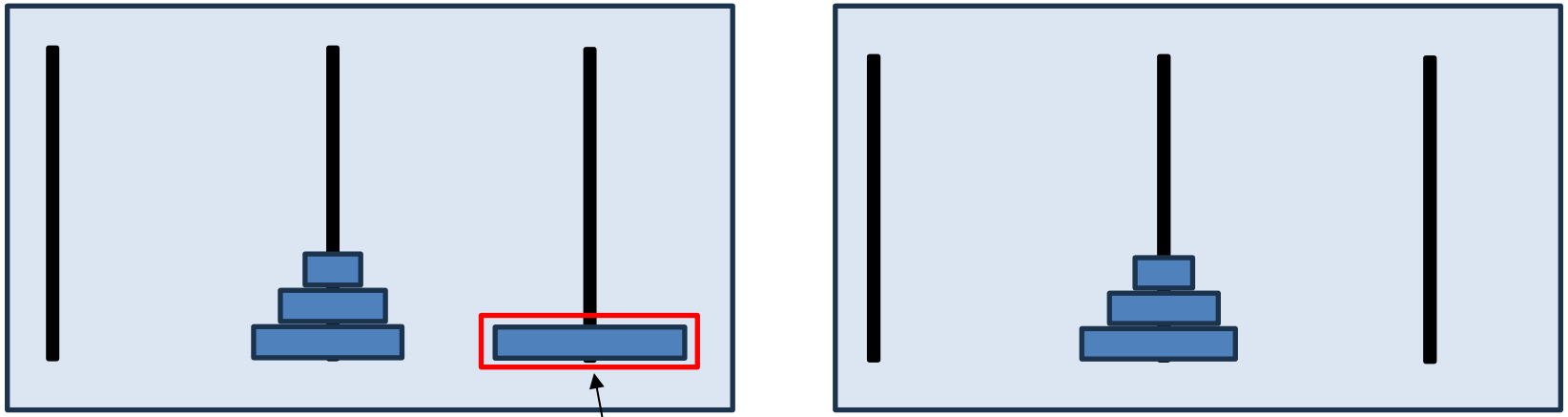


Wir können jede Scheibe in der roten Box auf den verbleibenden Stäben platzieren.



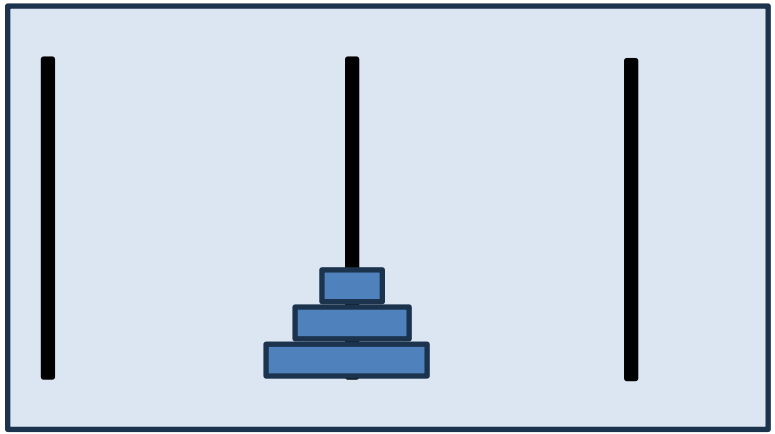


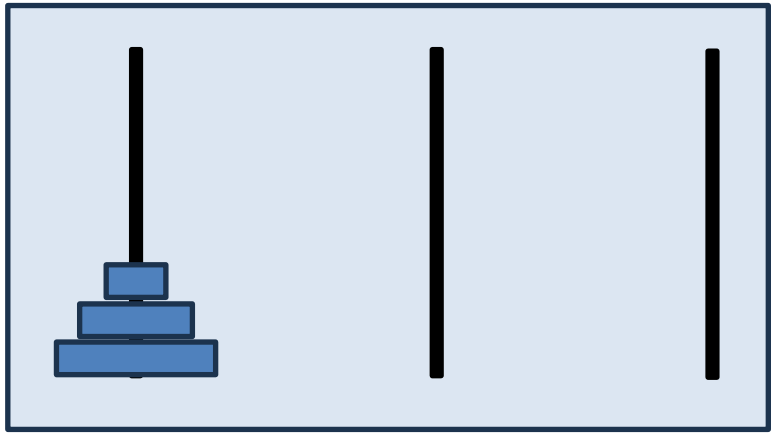
Können die Konstellationen gleich gelöst werden?



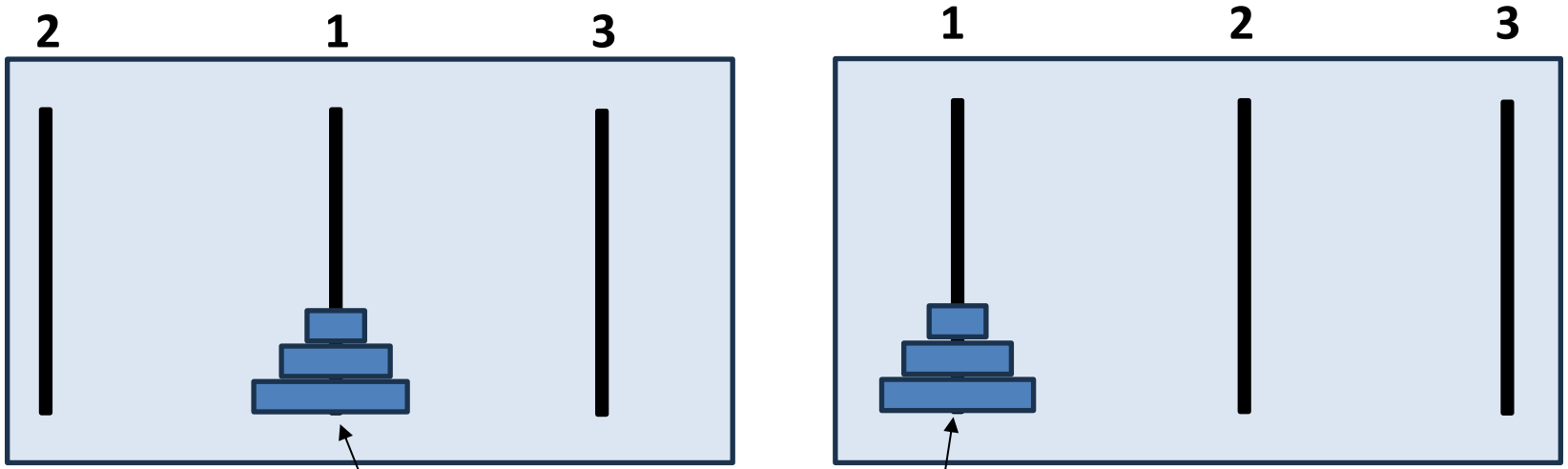
Diese Scheibe ändert das Problem nicht. Es ist die grösste Scheibe und wir können sie deshalb ignorieren und nur die verbleibenden Scheiben betrachten.

Reduktion



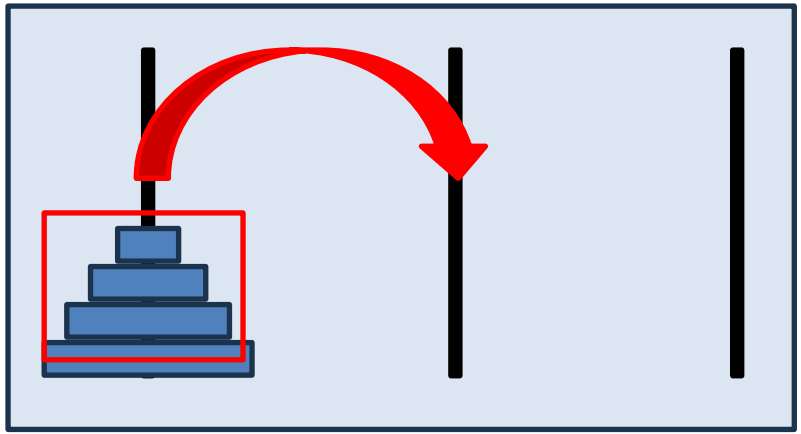


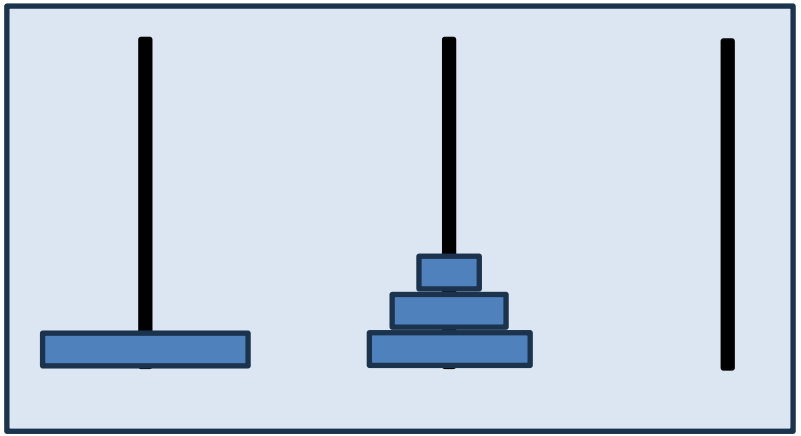
Können die Konstellationen gleich gelöst werden?



Reduktion

Das ist das gleiche Problem, wenn wir die Stäbe anders nummerieren.

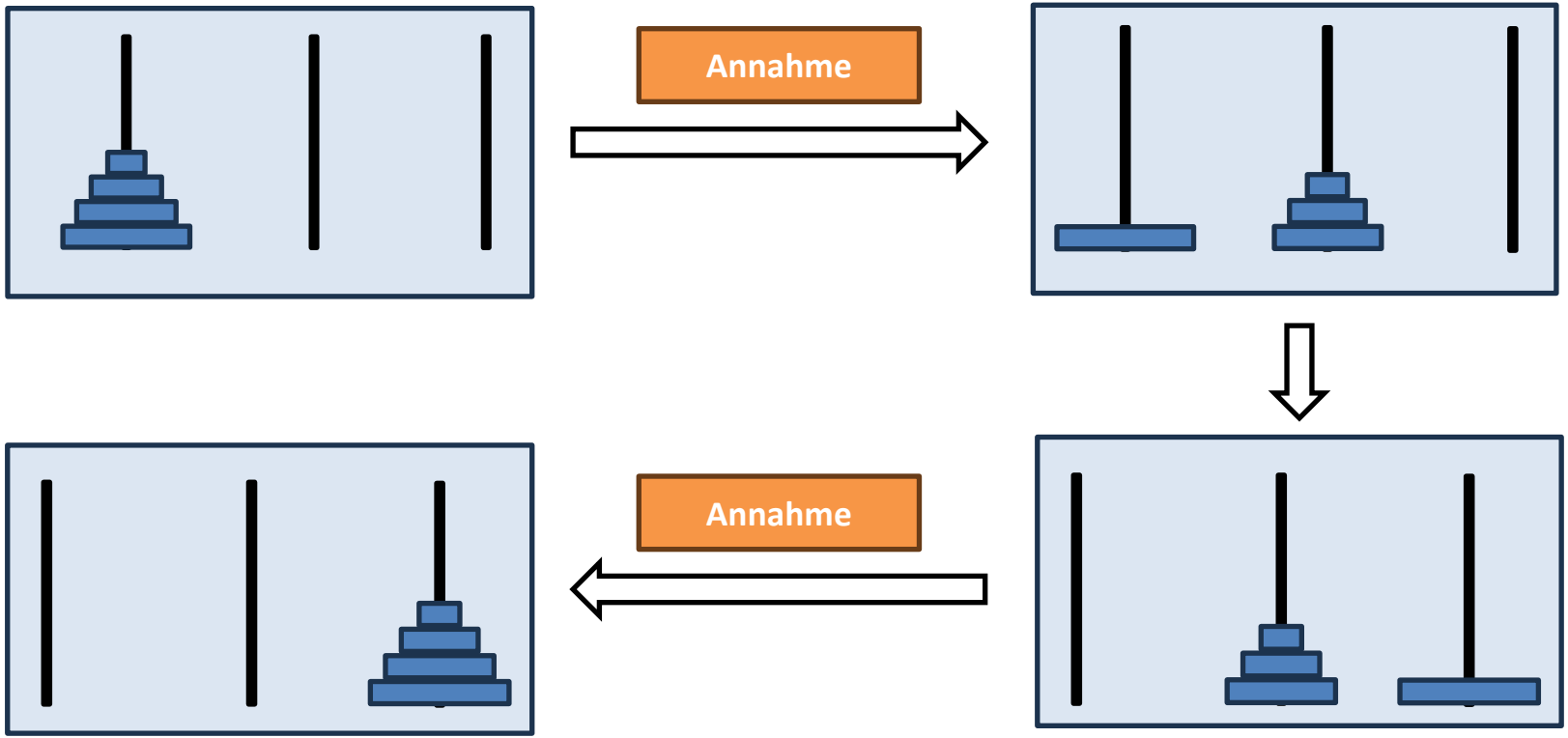




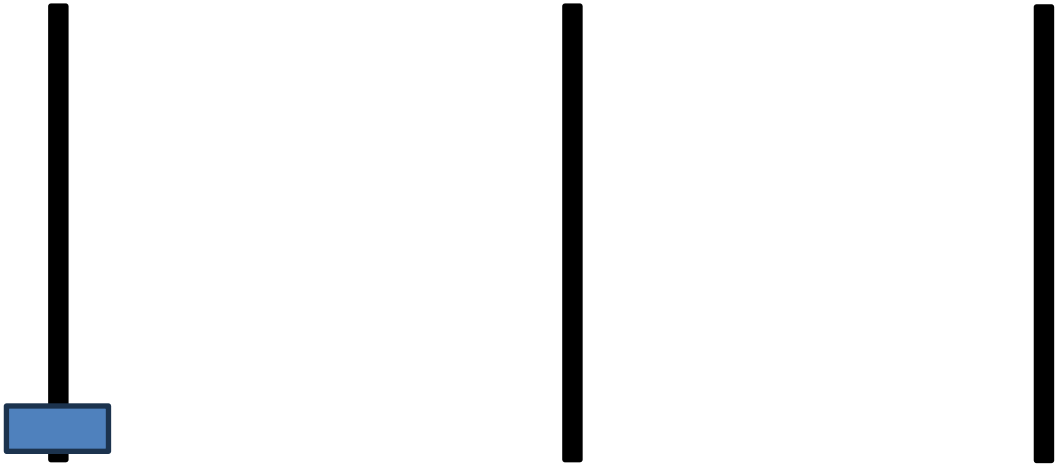
Wir nehmen nun an, wir wissen bereits wie wir einen Stapel von Grösse drei auf einen “freien” Stab bewegen können, wenn wir zwei “freie” Stäbe haben.

Annahme

Wir können das Problem jetzt lösen!

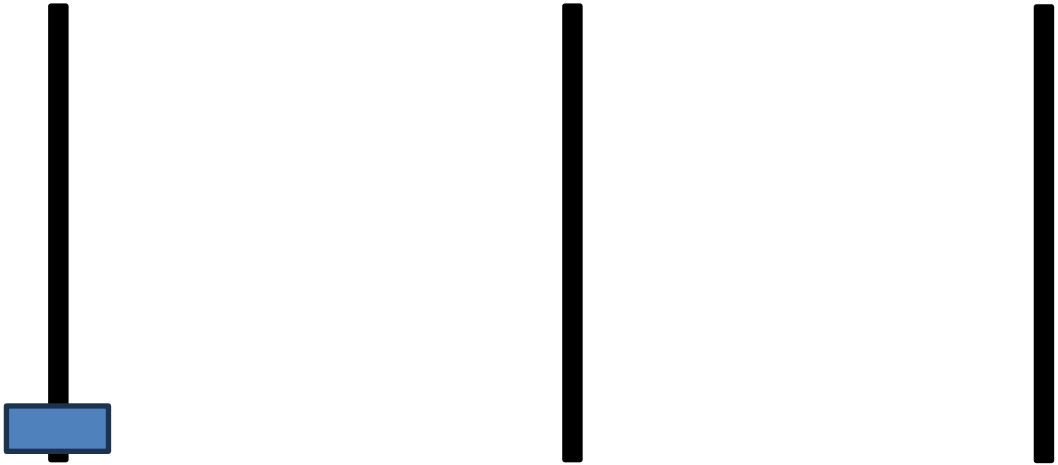


Stimmt die Annahme?



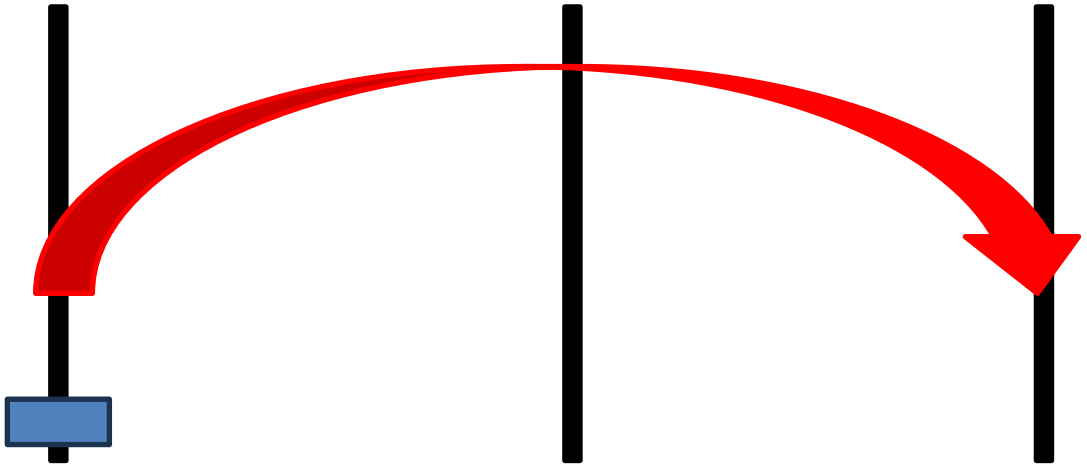
Nur in einem Fall!

Stimmt die Annahme?

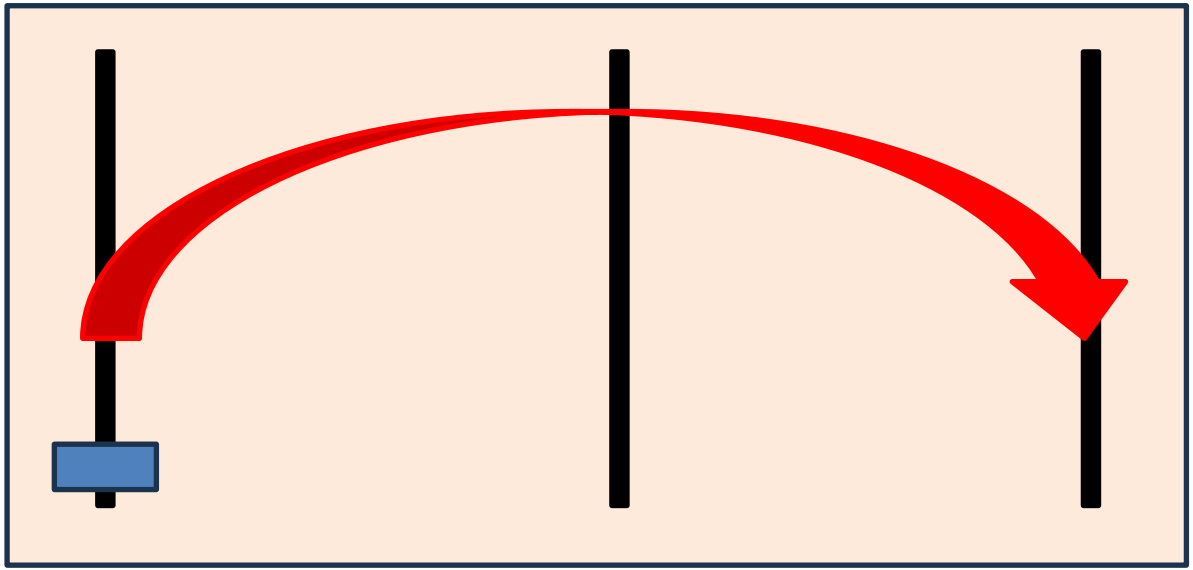


Das reicht aber!

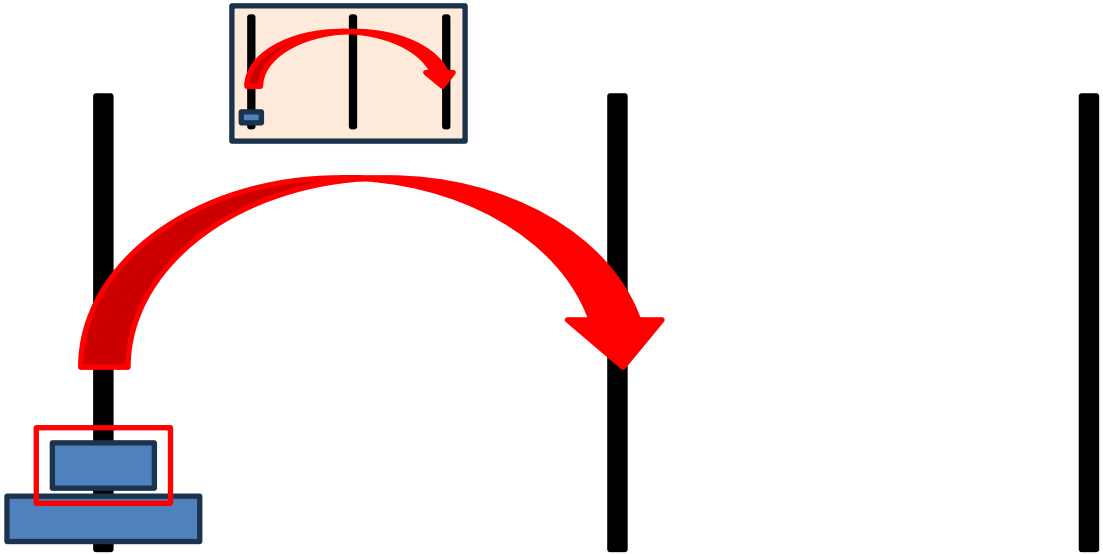
Towers of Hanoi – Step by Step



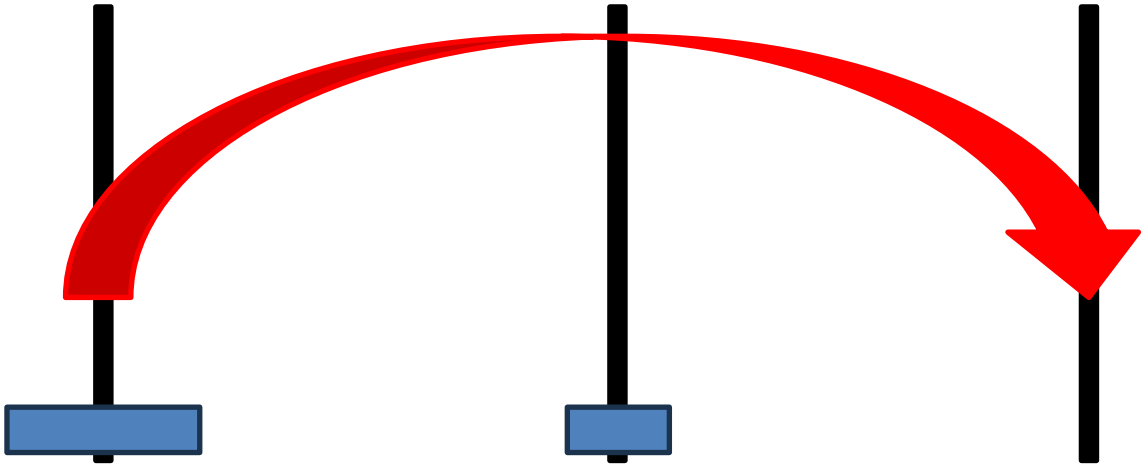
Towers of Hanoi – Step by Step



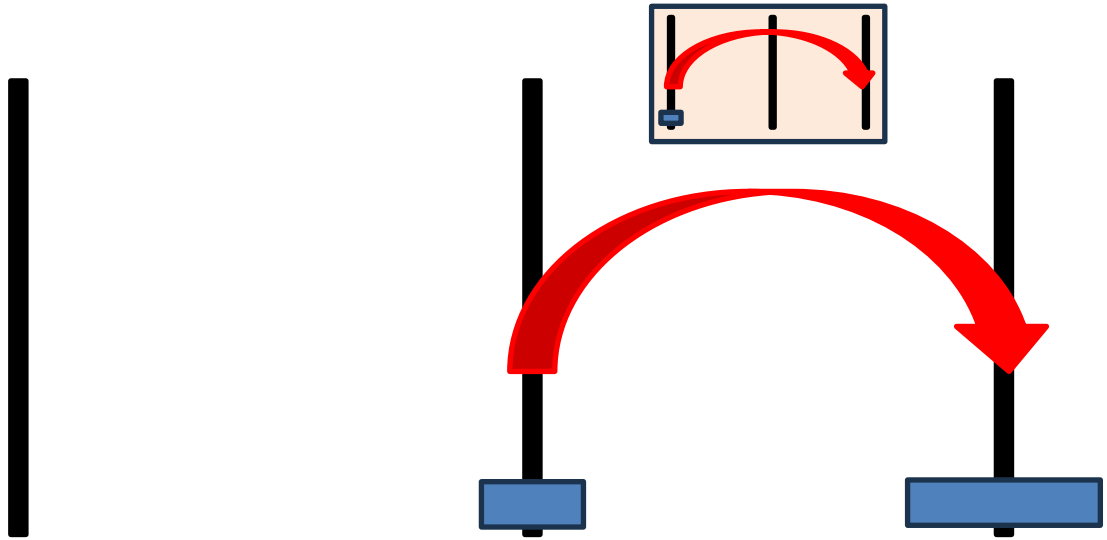
Towers of Hanoi – Step by Step



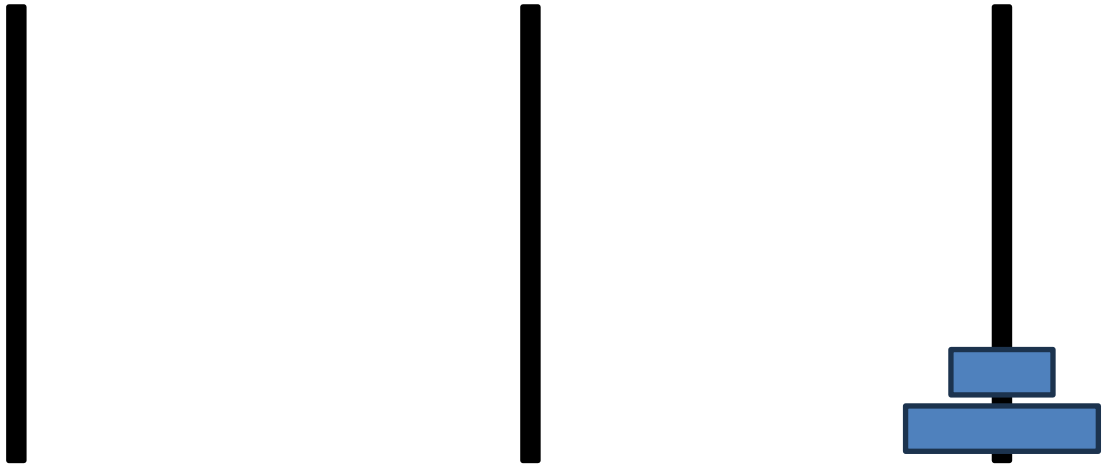
Towers of Hanoi – Step by Step



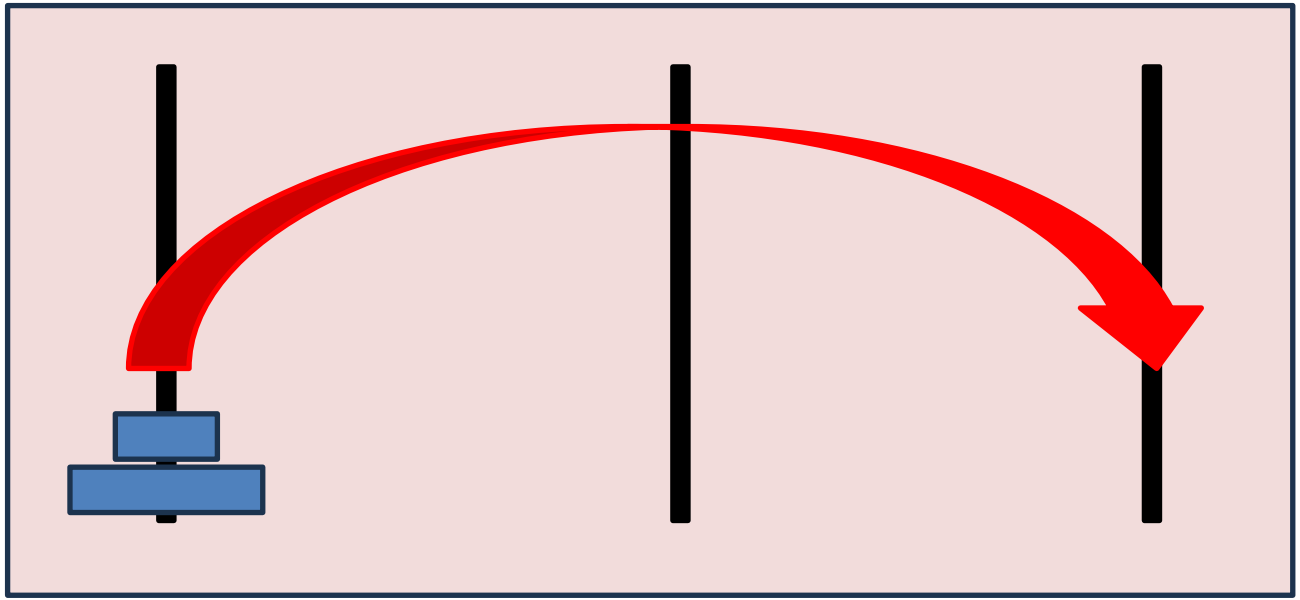
Towers of Hanoi – Step by Step



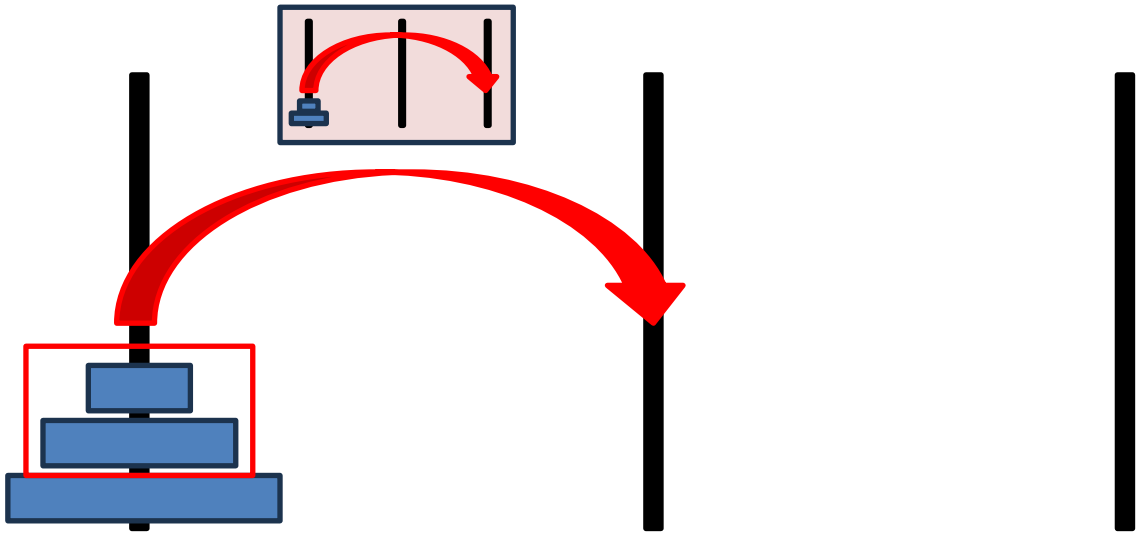
Towers of Hanoi – Step by Step



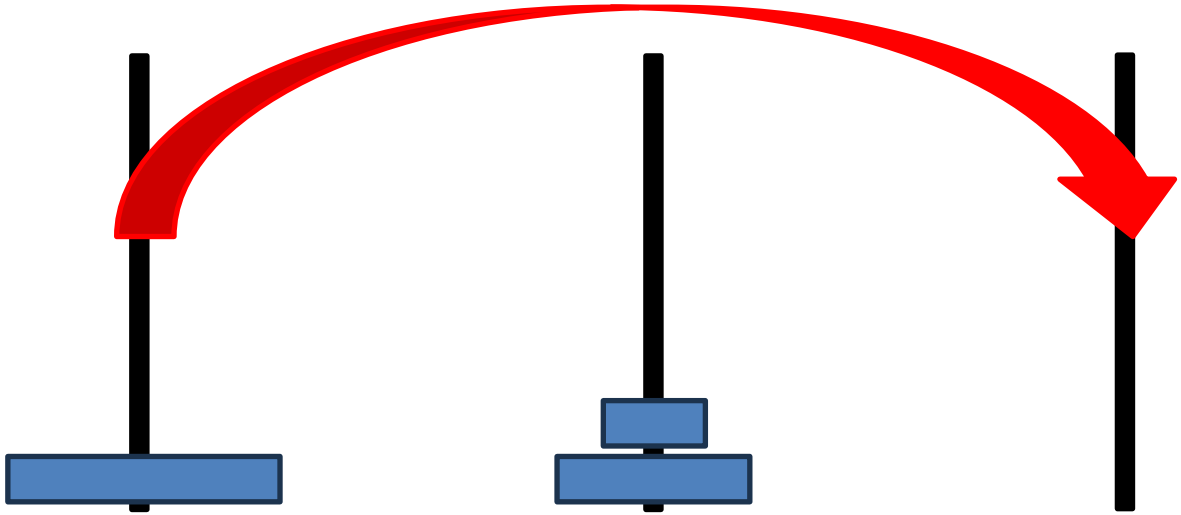
Towers of Hanoi – Step by Step



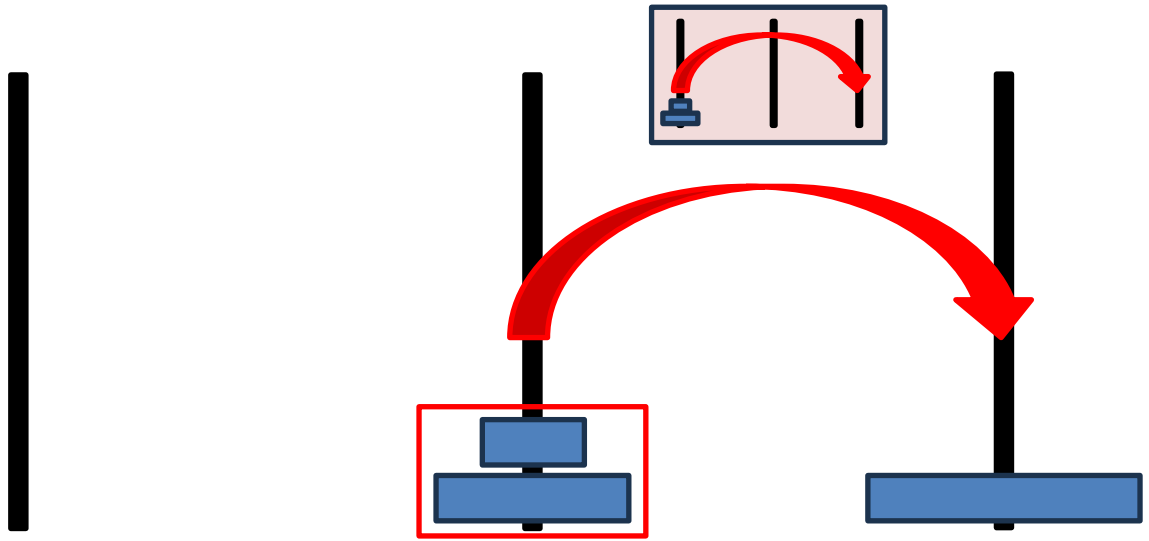
Towers of Hanoi – Step by Step



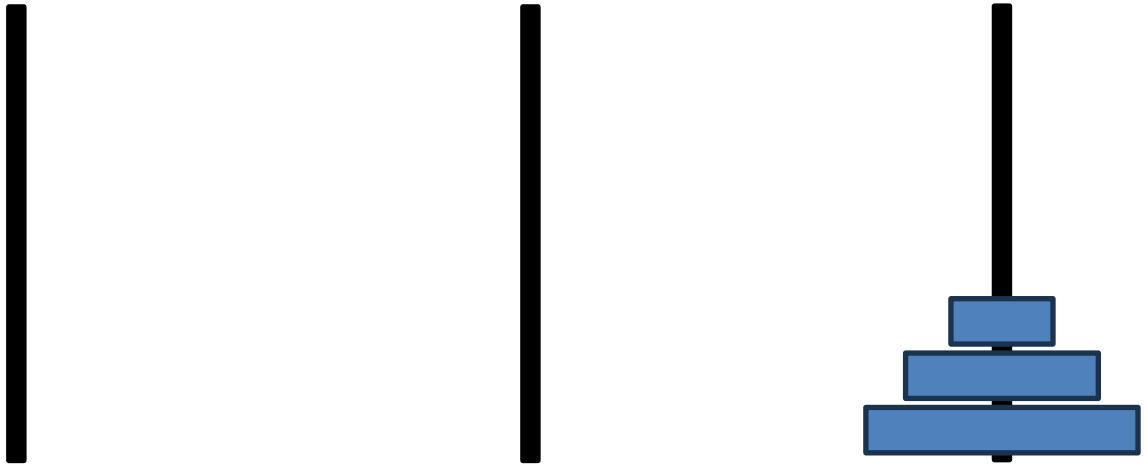
Towers of Hanoi – Step by Step



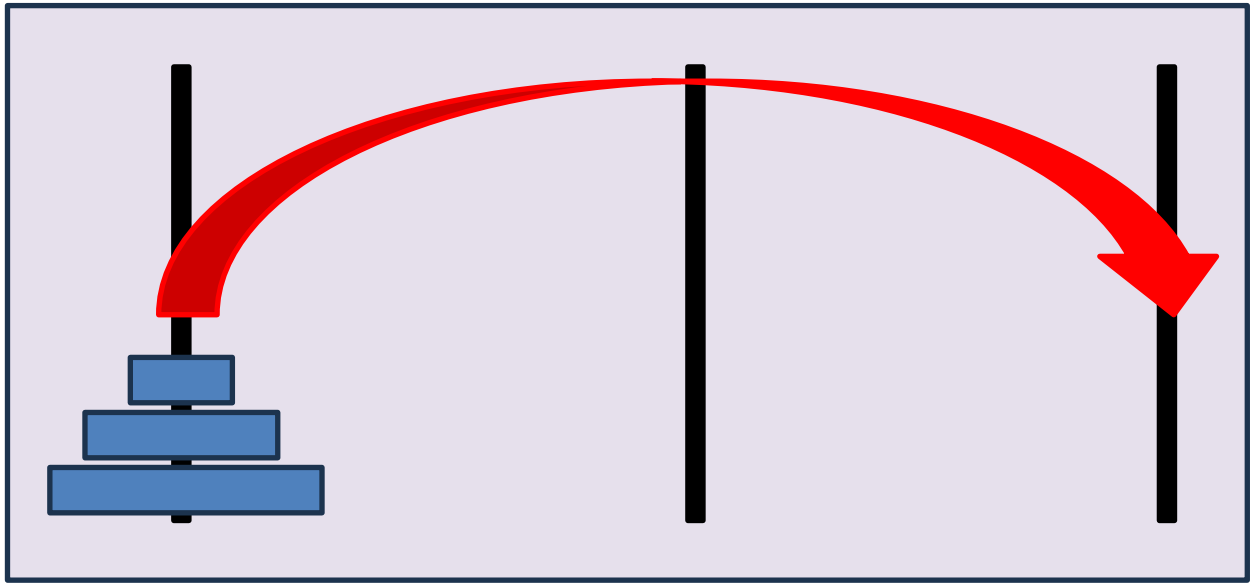
Towers of Hanoi – Step by Step



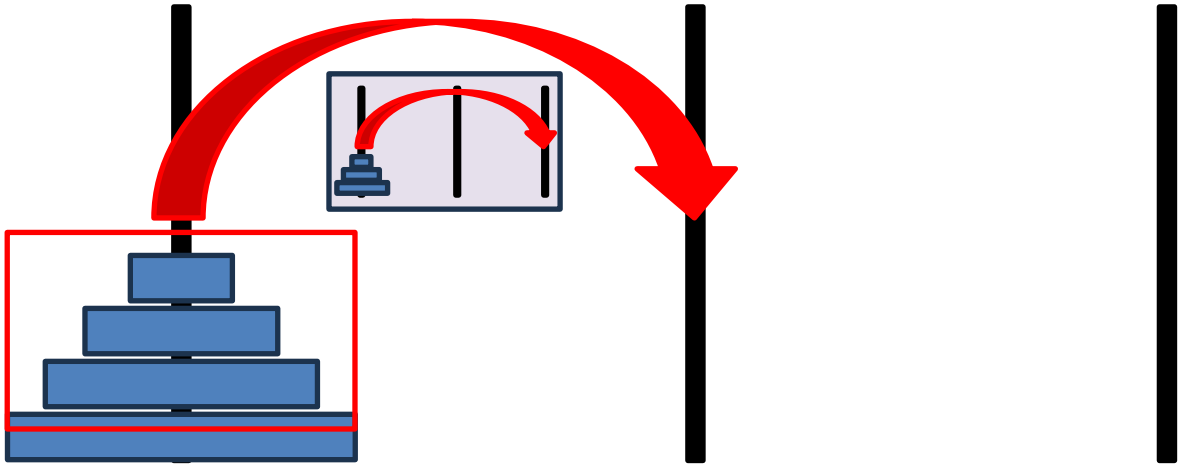
Towers of Hanoi – Step by Step



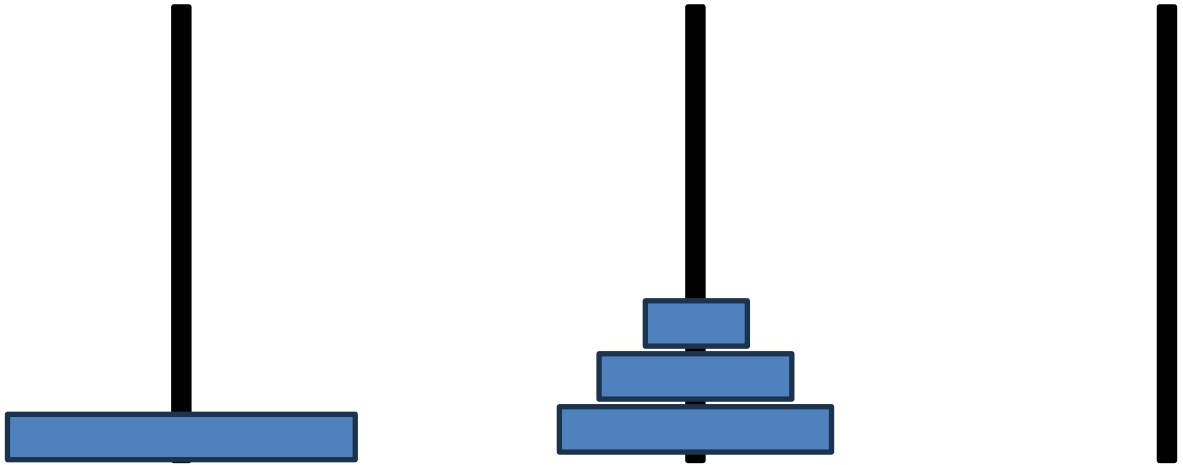
Towers of Hanoi – Step by Step



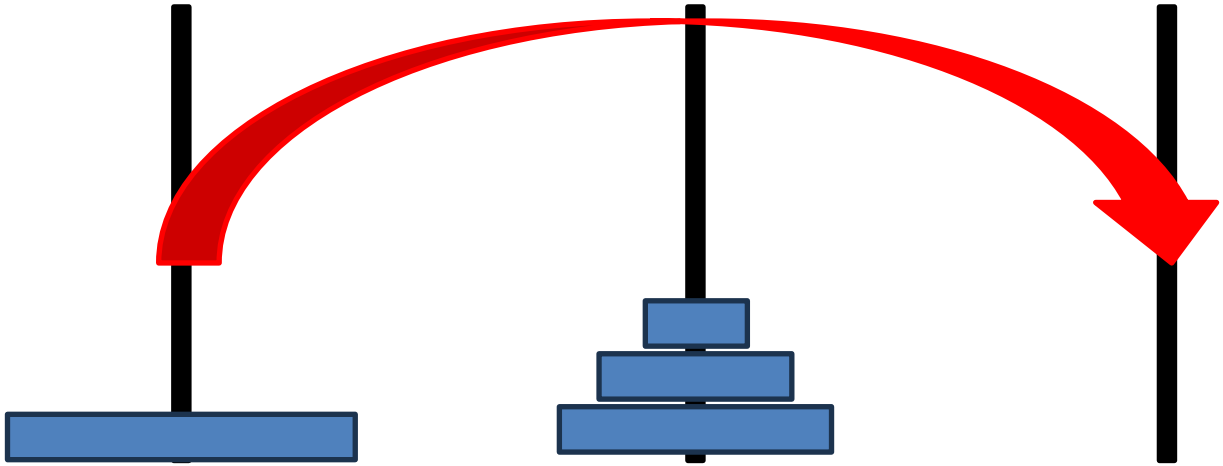
Towers of Hanoi – Step by Step



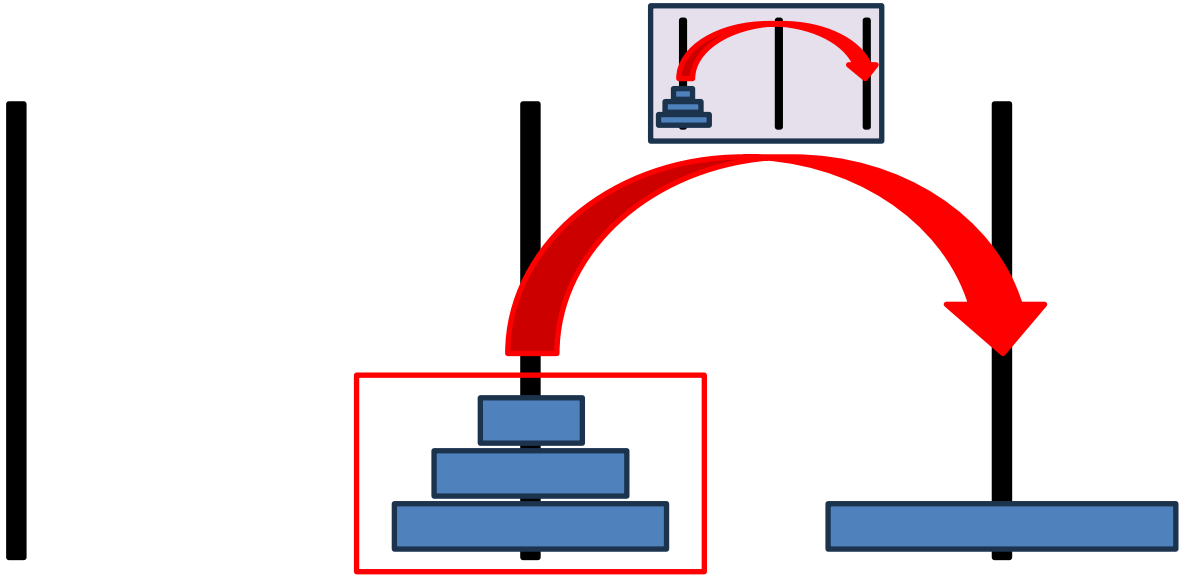
Towers of Hanoi – Step by Step



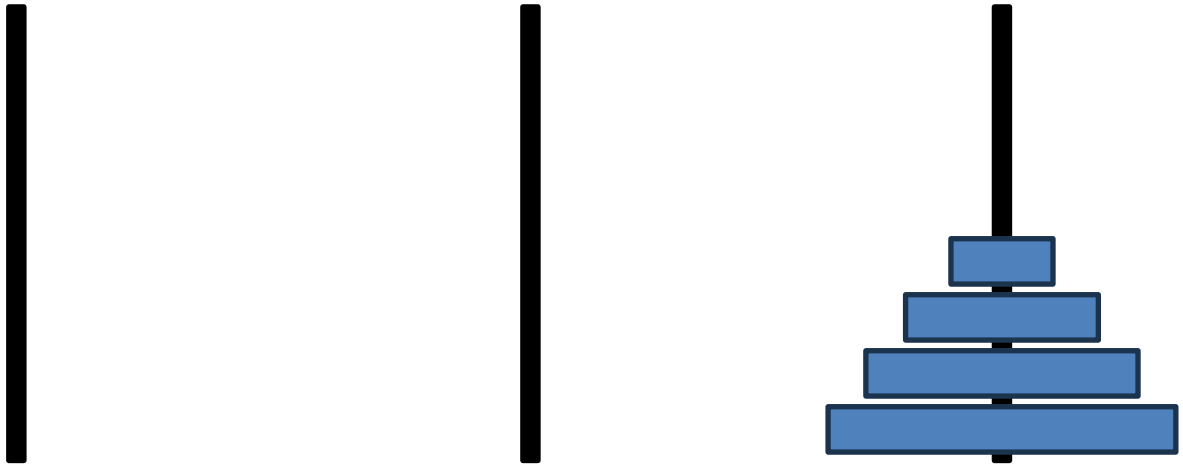
Towers of Hanoi – Step by Step



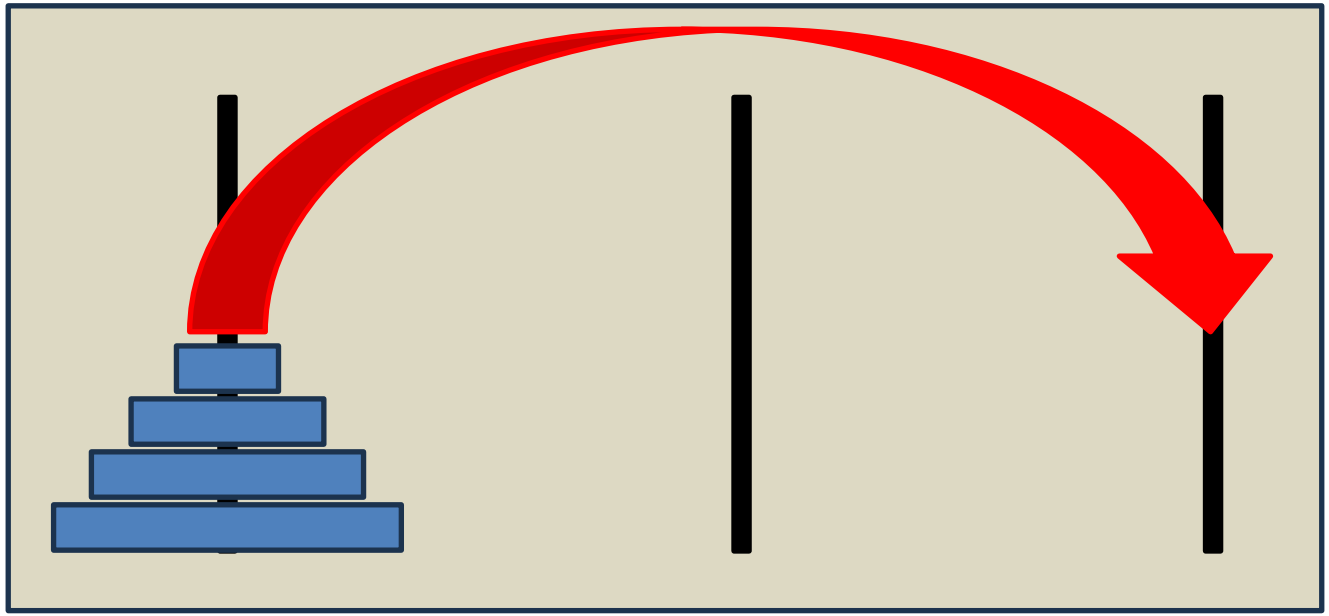
Towers of Hanoi – Step by Step



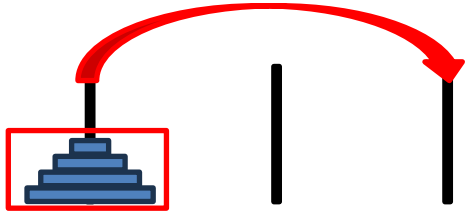
Towers of Hanoi – Step by Step



Towers of Hanoi – Step by Step



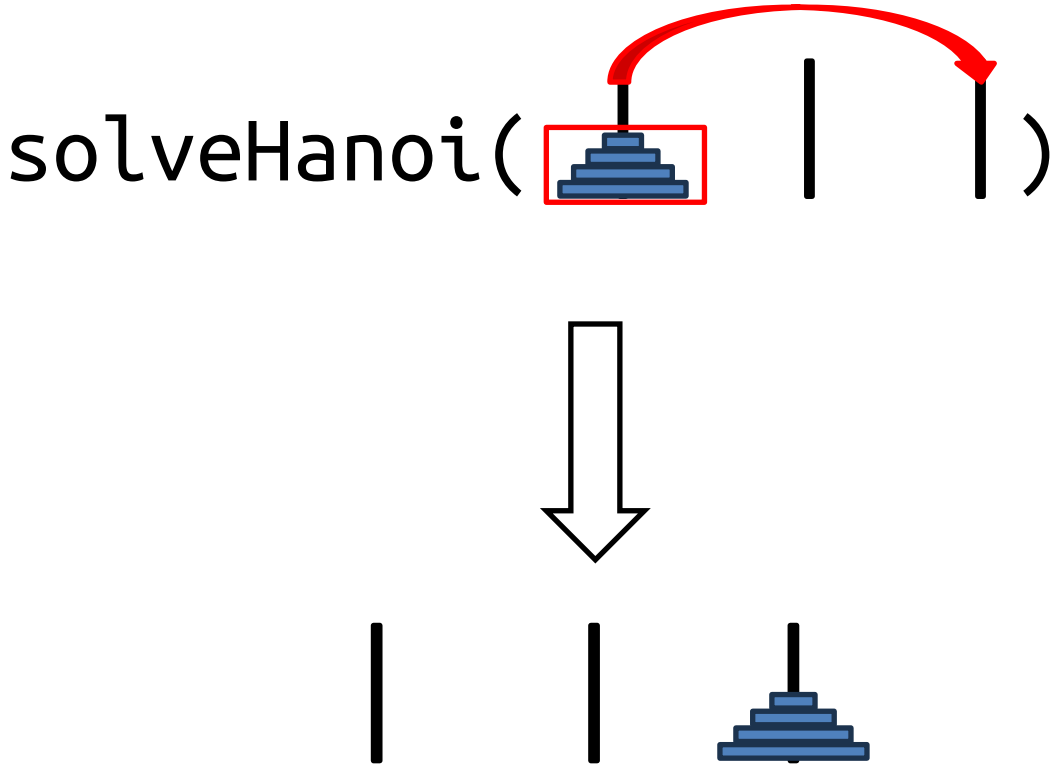
Towers of Hanoi – Methoden

Was soll `solveHanoi(`  `)` können?

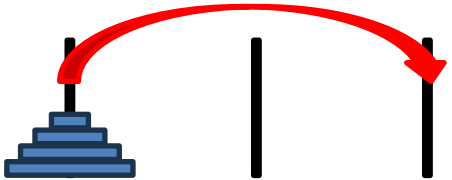
Input: Hanoi Konstellation mit zwei “freien” Stäben und ein (Teil-)stapel, der auf einen “freien” Stab bewegt werden soll.

- Ein Stab ist frei, wenn der zu bewegende Stapel nur kleinere Scheiben enthält, als die Scheiben auf dem Stab.

Towers of Hanoi – Methoden



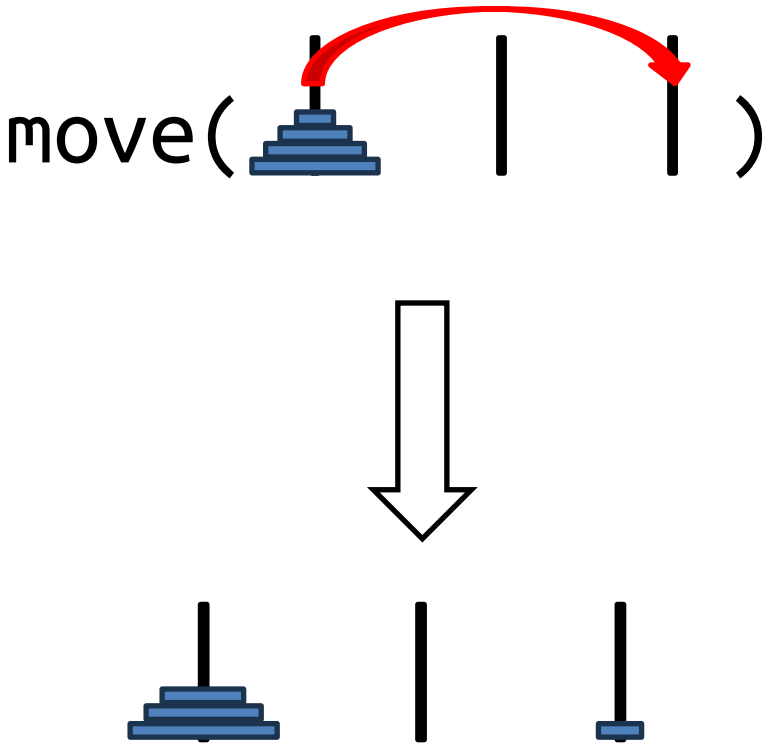
Towers of Hanoi – Methoden

Was soll `move(`  `)` können?

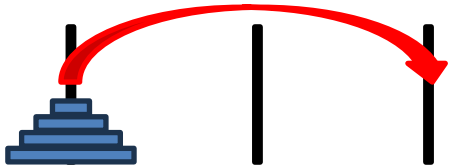
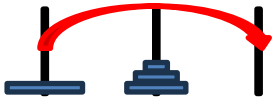
Input: Hanoi Konstellation mit mindestens einem “freien” Stab und eine Scheibe soll auf einen “freien” Stab bewegt werden.

- Ein Stab ist frei, wenn der zu bewegende Stapel nur kleinere Scheiben enthält, als die Scheiben auf dem Stab.

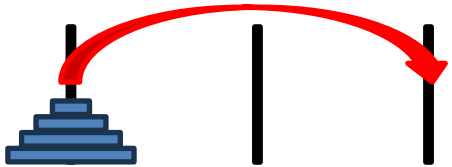
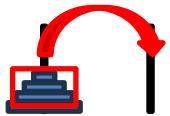
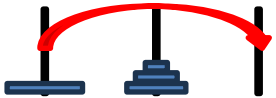
Towers of Hanoi – Methoden



Towers of Hanoi – Programm

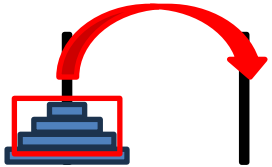
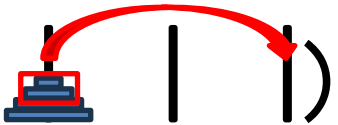
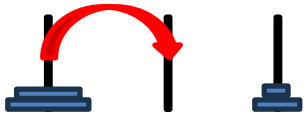
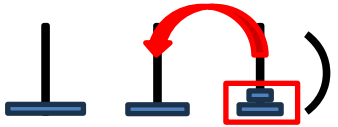
```
solveHanoi(  ) {
    solveHanoi(  )
    move(  )
    solveHanoi(  )
}
```

Towers of Hanoi – Programm

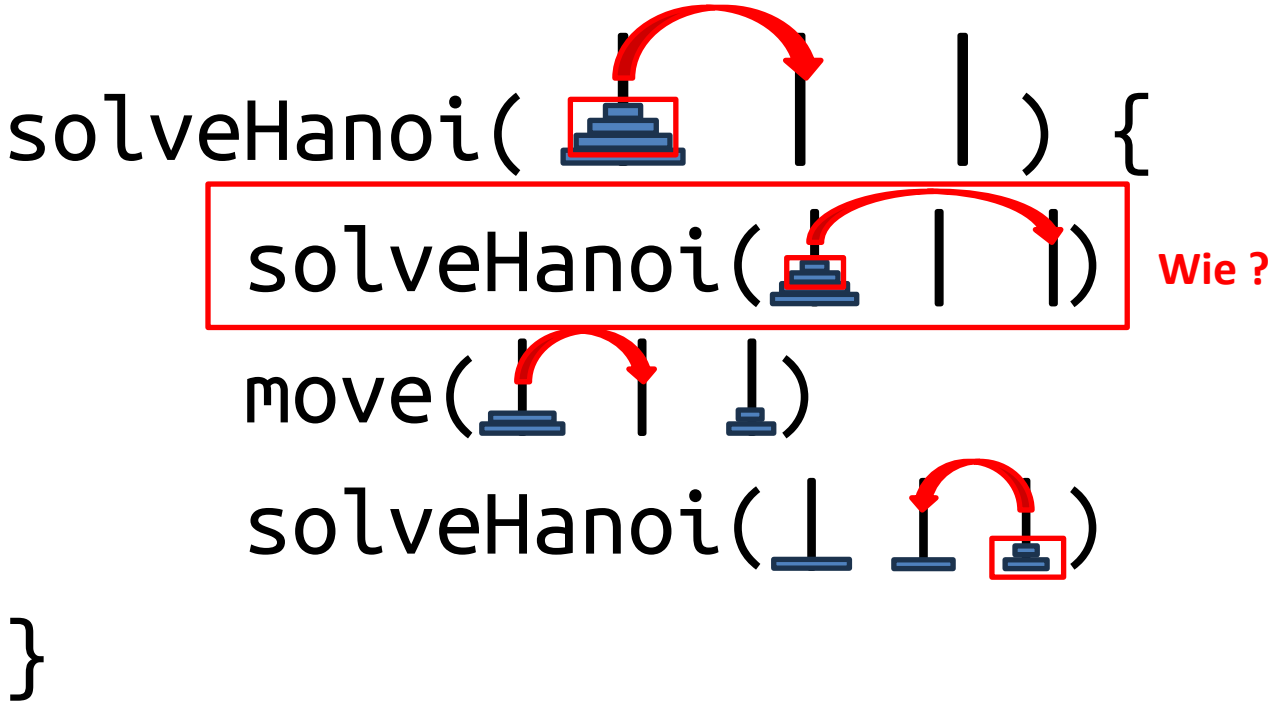
```
solveHanoi(  ) {
    solveHanoi(  )
    move(  )
    solveHanoi(  )
}
```

Wie ?

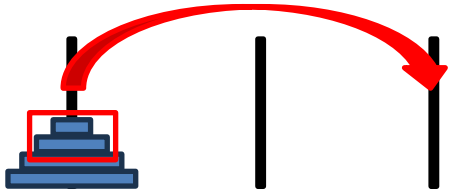
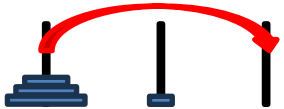
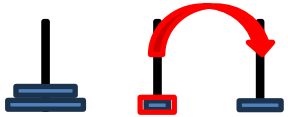
Towers of Hanoi – Programm

```
solveHanoi(  ) {
    solveHanoi(  )
    move(  )
    solveHanoi(  )
}
```

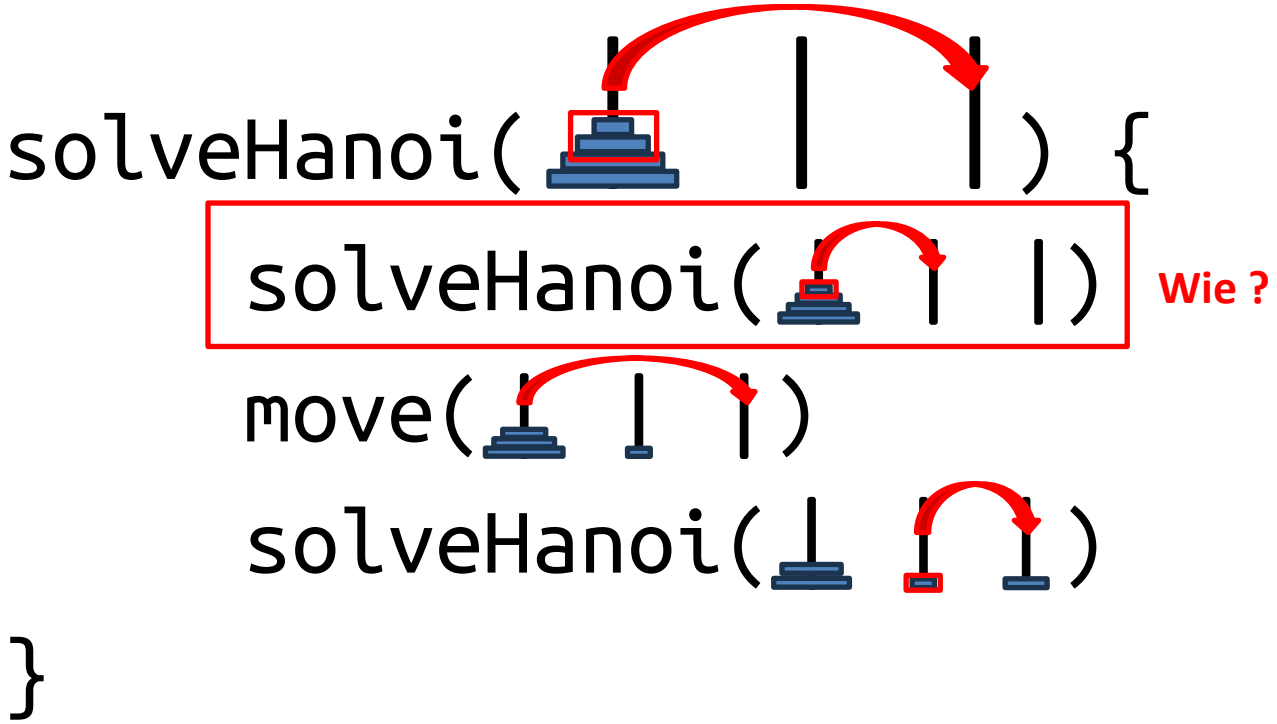
Towers of Hanoi – Programm



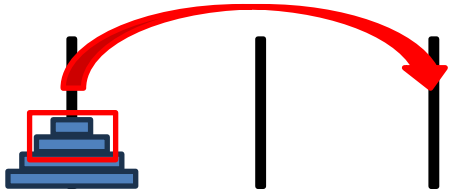
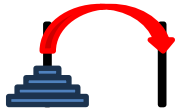
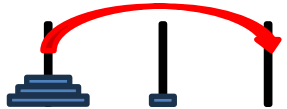
Towers of Hanoi – Programm

```
solveHanoi(  ) {
    solveHanoi(  )
    move(  )
    solveHanoi(  )
}
```

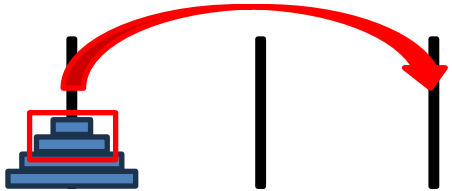
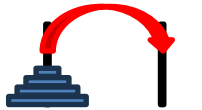
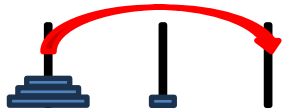
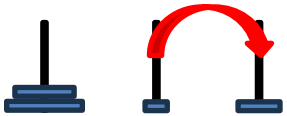
Towers of Hanoi – Programm



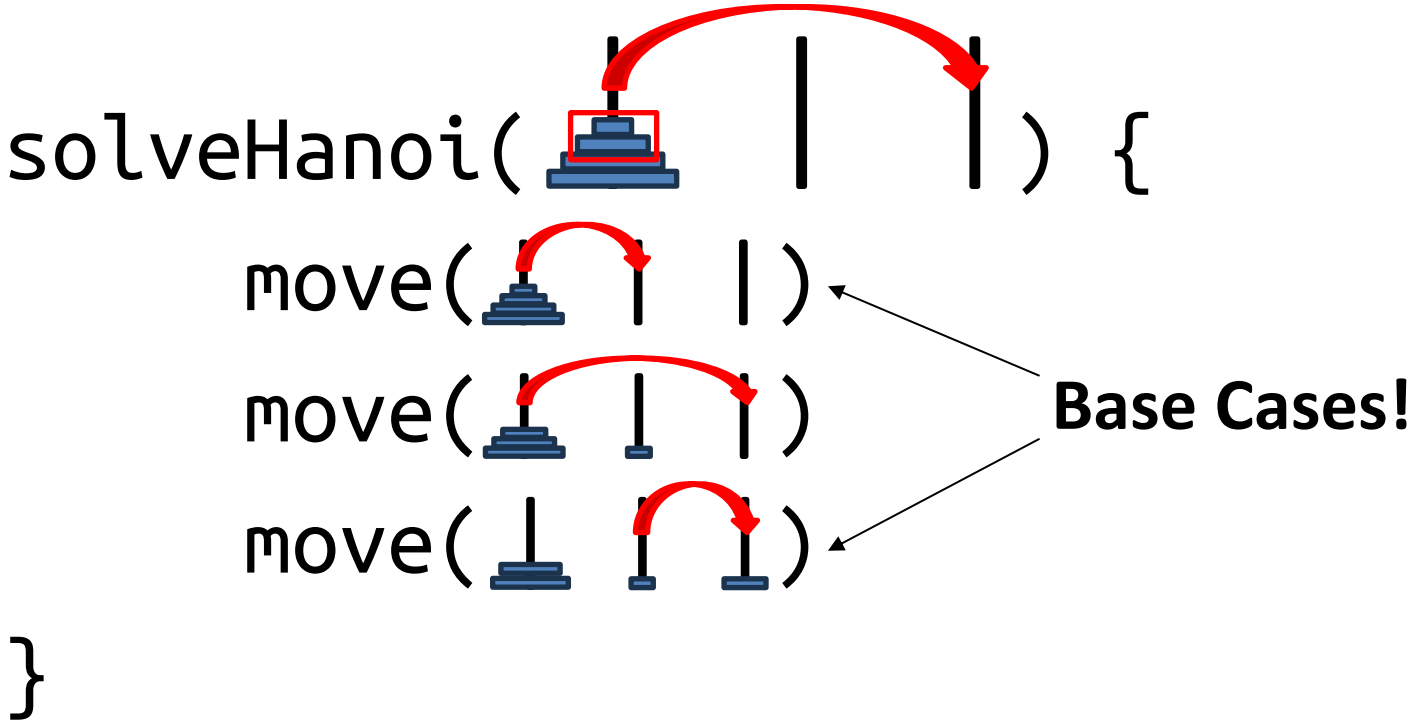
Towers of Hanoi – Programm

```
solveHanoi(  ) {
    move(  )
    move(  )
    solveHanoi(  )
}
```

Towers of Hanoi – Programm

```
solveHanoi(  ) {
    move(  )
    move(  )
    move(  )
}
```

Towers of Hanoi – Programm

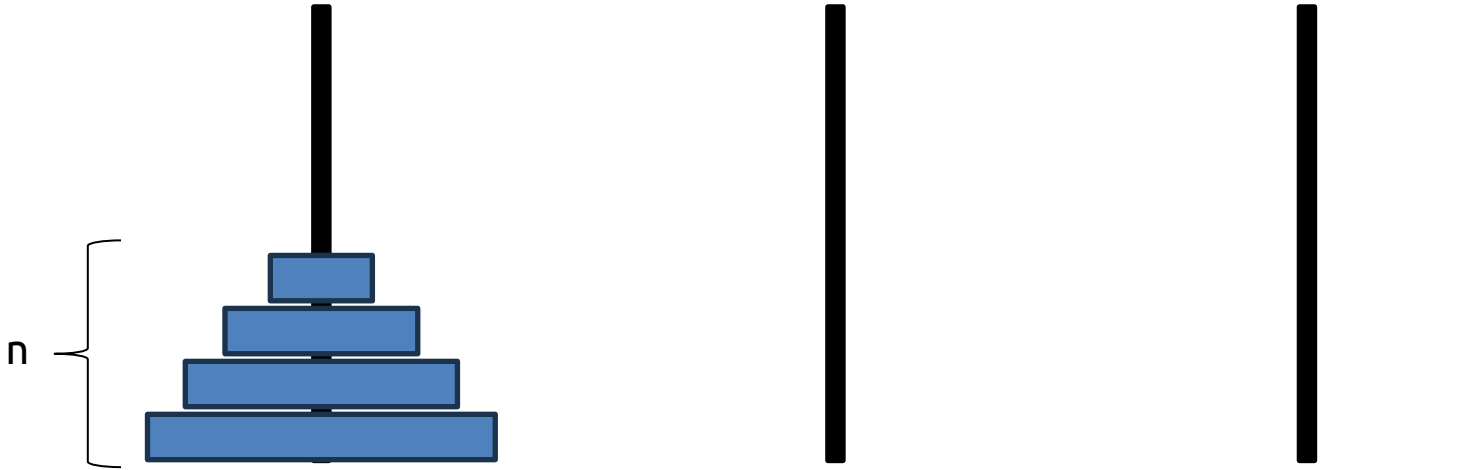


Towers of Hanoi – Als Programm

- Wir verstehen **wie** man das Problem lösen kann.
  - Das **wie** ist meistens der wichtigste Teil.

- **Zuerst:** Das Problem verstehen.
- **Dann:** Das Programm implementieren.

Towers of Hanoi – Als Programm



Towers of Hanoi – Als Programm

- **Methode 1:** `solveHanoi(int n, char stapelStab, char freierStab, char zielStab)`

- **Methode 2:** `move(int n, char stapelStab, char freierStab, char zielStab)`

Towers of Hanoi – Als Programm

```java
public class TowerOfHanoi {
    public static void move(int n, char stapelStab, char freierStab, char zielStab) {
        System.out.println("Scheibe " + n + " wurde von Stab " + stapelStab + " zu Stab " + zielStab + " bewegt.");
    }
    public static void solveHanoi(int n, char stapelStab, char freierStab, char zielStab) {
        if(n == 1) { // Base Case
            move(n, stapelStab, freierStab, zielStab);
        } else { // Step Case
            solveHanoi(n - 1, stapelStab, zielStab, freierStab);
            move(n, stapelStab, freierStab, zielStab);
            solveHanoi(n - 1, freierStab, stapelStab, zielStab);
        }
    }
    public static void main(String[] args) {
        int scheiben = 4;

        solveHanoi(scheiben, '1', '2', '3');
    }
}
```

Towers of Hanoi – Als Programm

- **Link zum Code:**
  https://lec.inf.ethz.ch/infk/eprog/2025/exercises/additionals/TowerOfHanoi.java

Permutation Check

Beispielsaufgabe

Gegeben sind die Integer Arrays `s` und `t`. Überprüfe, ob `t` eine Permutation von `s` ist.

Lösung 1: Generiere alle Permutationen von `s` und schaue ob `t` eine davon ist.

Beispielsaufgabe

Gegeben sind die Integer Arrays `s` und `t`. Überprüfe, ob `t` eine Permutation von `s` ist.

Lösung 1: Generiere alle Permutationen von `s` und schaue ob `t` eine davon ist.

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at TestingStuff/module.Permutation.permute(Permutation.java:16)
        at TestingStuff/module.Permutation.permute(Permutation.java:23)
        at TestingStuff/module.Permutation.permute(Permutation.java:23)
        at TestingStuff/module.Permutation.permute(Permutation.java:23)
        at TestingStuff/module.Permutation.permute(Permutation.java:23)
        at TestingStuff/module.Permutation.permute(Permutation.java:23)
```

Out of Memory?

- Java erhält für das Ausführen eines Programms eine **beschränkte** Menge Arbeitsspeicher (Memory).

- In der vorherigen Lösung generieren wir **O(n!) Permutationen**, welche alle Speicher benötigen.

- Irgendwann hat es **nicht mehr genug Speicher** und das Programm terminiert mit einem Laufzeit**fehler**.

Beispielsaufgabe

Gegeben sind die Integer Arrays `s` und `t`. Überprüfe, ob `t` eine Permutation von `s` ist.

Lösung 2: Generiere der Reihe nach alle Permutationen von `s` und überprüfe direkt nach dem Erstellen einer Permutation, ob sie gleich `t` ist.

Beispielsaufgabe

Gegeben sind die Integer Arrays `s` und `t`. Überprüfe, ob `t` eine Permutation von `s` ist.

Lösung 3: Prüfe ob `s` und `t` gleich lang sind und ob sie die gleichen Elemente enthalten.

Vorbesprechung

Discord: timostucki

Meine Aufgaben Ratings:

- **Best Case: Ihr macht alle Aufgaben (, wenn die Zeit reicht)**
- **Falls nicht: Ich mache Ratings zur Wichtigkeit der einzelnen Aufgaben**

- **(Keine offizielle Empfehlung, meine subjektive Meinung auf Basis meiner eigenen Erfahrung als Student in diesem Kurs)**
- **! Sagen nichts über die Schwierigkeit der Aufgaben aus !**

Wichtig für die Prüfung:
(Prüfungs- oder prüfungsähnliche Aufgaben)

Wichtig für euer Verständnis:
(Aber nicht in Prüfungsform)

Wichtig für tiefes Verständnis/ Zusatzaufgaben:
(Bspw. viele vom gleichen Typ)

Webseite

- **timostucki.com**

Aufgabe 1: Dreiecksmatrix

Die Klasse `Triangle` erlaubt die Darstellung von $Z \times S$ Dreiecksmatrizen (von `int` Werten). Z und S sind immer strikt grösser als 1 (d.h., > 1). Eine $Z \times S$ Dreiecksmatrize hat Z Zeilen $X_0, X_1, ..., X_{Z-1}$, wobei Zeile X_i genau $(i * (S-1)/(Z-1)) + 1$ viele Elemente hat. Dieser Ausdruck wird nach den Regeln für `int` Ausdrücke in Java ausgewertet. Für eine Dreiecksmatrix `D` ist $D_{i,j}$ das $(j+1)$-te Element in der $(i+1)$-ten Zeile. $D_{0,0}$ ist das erste Element in der ersten Zeile [die immer genau 1 Element hat]. Abbildung 1 zeigt Beispiele von Dreiecksmatrizen. Beachten Sie, dass es möglich ist, dass zwei (aufeinanderfolgende) Zeilen die selbe Anzahl Elemente haben.

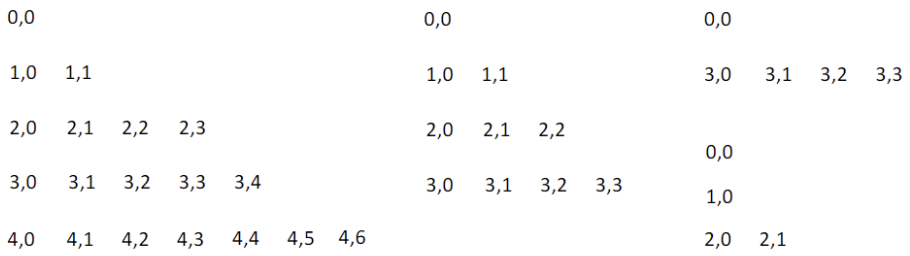


Abbildung 1: Beispiele von 5×7, 4×4, 2×4 und 3×2 Dreiecksmatrizen.

Aufgabe 1: Dreiecksmatrix

In der Datei `Triangle.java` finden Sie die Klasse `Triangle` mit einem Konstruktor `Triangle(int z, int s)`, der eine $z \times s$ Dreiecksmatrix erstellt. Dieser Konstruktor setzt die Werte aller Elemente auf 0. Vervollständigen Sie diese Klasse, so dass die folgenden Methoden unterstützt werden:

1. `int get(int i, int j)` gibt das Element $D_{i,j}$ zurück.
2. `void put(int i, int j, int value)` setzt das Element $D_{i,j}$ auf den Wert `value`.
3. `int[] linear()` liefert die Elemente in der kanonischen Reihenfolge (die Elemente jeder Zeile mit steigendem Index, und die Zeilen in steigender Reihenfolge).
4. `void init(int[] data)` ersetzt die Elemente von `D` durch die Werte in `data`. Sie dürfen annehmen, dass `data` genauso viele Elemente hat wie `D`. Die Methode setzt die Elemente von `D`, so dass die Folge `D.init(data); int[] y = D.linear();` in einen Array `y` resultiert für den `Arrays.equals(y, data)` den Wert `true` ergibt.
5. `void add(Triangle t)` Ein Aufruf `D.add(t)` addiert zu jedem Element $D_{i,j}$ den Wert von $t_{i,j}$, falls $t_{i,j}$ existiert. Falls $t_{i,j}$ nicht existiert, dann bleibt $D_{i,j}$ unverändert.

Tests finden Sie in der Datei "TriangleTest.java". Die Datei "TriangleGradingTest.java" enthält die Tests, welche wir bei der Prüfung für die Korrektur verwendet haben. Wir empfehlen, diese Tests erst zu verwenden, wenn Sie denken, dass Ihre Lösung korrekt ist, damit Sie sehen können, wie Sie bei einer Prüfung abgeschnitten hätten.

Aufgabe 2: Hoare Tripel

Welche dieser Hoare Tripel sind (un)gültig? Bitte geben Sie für ungültige Tripel ein Gegenbeispiel an. Die Anweisungen sind Teil einer Java Methode. Alle Variablen sind vom Typ `int` und es gibt keinen Overflow.

1. `{ x >= 0 || y >= 0 }  z = x * y;  { z > 0 }`
2. `{ x > 10 }  z = x % 10;  { z > 0 }`
3. `{ x > 0 }  y = x * x; z = y / 2;  { z > 0 }`
4. `{ x > 0 }  y = x * x; sum = y % (x + 1); { sum > 1 }`
5. `{ b > c }`

```
if (x > b) {
    a = x;
} else {
    a = b;
}
```

`{ a > c }`

Aufgabe 3: Weakest Precondition

Bitte geben Sie für die folgenden Programmsegmente die schwächste Vorbedingung (weakest precondition) an. Bitte verwenden Sie Java-Syntax. Alle Anweisungen sind Teil einer Java Methode. Alle Variablen sind vom Typ `int` und es gibt keinen Overflow.

1.

```
P: { ?? }
S:   if (x < 5) {
       y = x * x;
     } else {
       y = x + 1;
     }
Q: { y >= 9 }
```

2.

```
P: { ?? }
S:  if (x != y) {
      x = y;
    } else {
      x = y + 1;
    }
Q: { x != y }
```

Aufgabe 4: Blackbox Testing

Im letzten Übungsblatt haben Sie Testautomatisierung mit JUnit kennengelernt. In dieser Aufgabe sollen Sie nun Tests für eine Methode schreiben, deren Implementierung Sie nicht kennen. Dadurch werden Sie weniger durch möglicherweise falsche Annahmen beeinflusst, die bei einer Implementierung getroffen wurden. Sie müssen sich also überlegen, wie sich *jede* fehlerfreie Implementierung verhalten muss. Diesen Ansatz nennt man auch Black-Box Testing da die Details der Implementation verdeckt sind.

In Ihrem "U06"-Projekt befindet sich eine "blackbox.jar"-Datei, welche eine kompilierte Klasse `BlackBox` enthält. Den Code dieser Klasse können Sie nicht sehen, aber sie enthält eine Methode `void rotateArray(int[] values, int steps)`, welche Sie aus einer eigenen Klasse oder einem Unit-Test aufrufen können. Diese Methode "rotiert" ein `int`-Array um eine gegebene Anzahl Schritte.

Vereinfacht macht die Methode `rotateArray()` Folgendes: Eine Rotation mit `steps=1` bedeutet, dass alle Elemente des Arrays um eine Position nach rechts verschoben werden. Das letzte Element wird dabei zum ersten. Mit `steps=2` wird alles um zwei Positionen nach rechts rotiert, usw. Eine Rotation nach links kann mit einer negativen Zahl für `steps` erreicht werden. Das folgende Beispiel ist der erste, einfache Test, den Sie in der Datei "BlackBoxTest.java" finden:

```
int[] values = new int[] { 1, 2 };
int[] expected = new int[] { 2, 1 };
BlackBox.rotateArray(values, 1);
assertArrayEquals(expected, values);
```

Aufgabe 5: Levels

- **Bonus**

Achtung: Diese Aufgabe gibt Bonuspunkte (siehe "Leistungskontrolle" im `www.vvz.ethz.ch`). Die Aufgabe muss eigenhändig und alleine gelöst werden. Die Abgabe erfolgt wie gewohnt per Push in Ihr Git-Repository auf dem ETH-Server. Verbindlich ist der letzte Push vor dem Abgabetermin. Auch wenn Sie vor der Deadline committen, aber nach der Deadline pushen, gilt dies als eine zu späte Abgabe. Bitte lesen Sie zusätzlich die allgemeinen Regeln.

Nachbesprechung

Aufgabe 1: Präfixkonstruktion

Gegeben seien zwei Strings `s` und `t` und ein Integer `n` mit $n \geq 0$. Schreiben Sie ein Programm, das zurückgibt, ob `s` eine Konkatenation von maximal `n` vielen Präfixen von `t` ist.
Beispiele:

- `s = "abcababc"`, `t = "abc"`, `n = 4`: Das Programm sollte `true` zurückgeben, da `"abc"` und `"ab"` Präfixe von `t` sind und `s` eine Konkatenation von `"abc"`, `"ab"`, `"abc"` ist.
- `s = "abcbcabc"`, `t = "abc"`, `n = 4`: Das Programm sollte `false` zurückgeben, da `"bc"` kein Präfix von `t` ist.
- `s = "abab"`, `t = "abac"`, `n = 2`: Das Programm sollte `true` zurückgeben, da `"ab"` ein Präfix von `t` ist und `s` eine Konkatenation von `"ab"`, `"ab"` ist.

Implementieren Sie die Methode `isPrefixConstruction(String s, String t, int n)` in der Klasse `PrefixConstruction`. Die Methode hat drei Argumente: die beiden Strings `s` und `t` und der Integer `n`. Sie dürfen davon ausgehen, dass der Integer grösser oder gleich 0 ist. In der Datei "PrefixConstructionTest.java" finden Sie Tests.

Tipp: Lösen Sie die Aufgabe rekursiv.

Aufgabe 2: Weakest Precondition

Bitte geben Sie für die folgenden Programmsegmente die schwächste Vorbedingung (weakest precondition) an. Bitte verwenden Sie Java-Syntax. Alle Anweisungen sind Teil einer Java Methode. Alle Variablen sind vom Typ `int` und es gibt keinen Overflow.

1.
```
P: { ?? }
S:   m = n * 4;  k = m - 2;
Q: { n > 0 && k > 5 }
```

2.
```
P: { ?? }
S:   m = n * n;  k = m * 2;
Q: { k > 0 }
```

3.
```
P: { ?? }
S:   y = x + 3; z = y + 1;
Q: { z > 4 }
```

4.
```
P: { ?? }
S:   y = x + 1; z = y - 3;
Q: { z == 10 }
```

Beispiel

```
P: { ?? }
S:   a = b * 3; c = a + 1;
Q: { a > 0 && c < 5 }
```

Aufgabe 3: Wörter Raten

Das Programm "WoerterRaten.java" enthält Fragmente eines Rate-Spiels, welches Sie vervollständigen sollen. In dem Spiel wählt der Computer zufällig ein Wort w aus einer Liste aus und der Mensch muss versuchen, das Wort zu erraten. In jeder Runde kann der Mensch eine Zeichenfolge z (welche einen oder mehrere Buchstaben enthält) eingeben und der Computer gibt einen Hinweis dazu. Folgende Hinweise sind möglich:

1. w beginnt mit z
2. w endet mit z
3. w enthält z
4. w enthält nicht z

```
Tipp? e
Das Wort enthält nicht "e"!
Tipp? a
Das Wort endet mit "a"!
Tipp? j
Das Wort beginnt mit "j"!
Tipp? v
Das Wort enthält "v"!
Tipp? java
Das Wort ist "java"!
Glückwunsch, du hast nur 5 Versuche benötigt!
```

Aufgabe 4: Datenanalyse

In dieser Aufgabe werden Sie die Kelchblattlänge von Iris Blumen, welche auch Schwertlilien genannt werden, analysieren. Dazu verwenden Sie ein öffentlich zugängliches Dataset [1] welches die Längen vom Kelchblatt (Sepal Length) von jeweils 150 verschiedenen Iris Blumen enthält. Es gibt sehr viele Arten von Iris Blumen, insgesamt sind 285 Arten bekannt. Diese zu unterscheiden ist ein komplexer Prozess. Wir werden jedoch versuchen mittels der Länge des Kelchblattes drei Arten von Iris Blumen zu unterscheiden, in dem wir die gegebenen Daten analysieren. Wir werden uns auf die folgenden drei Iris Blumen konzentrieren:

- Iris setosa, auch Borsten-Schwertlilie genannt.
- Iris versicolor, auch Verschiedenfarbige Schwertlilie gennannt.
- Iris virginica, auch blaue Sumpfschwertlilie genannt.

Iris Versicolor **Iris Setosa** **Iris Virginica**