Prof. Dr. J. Giesl
D. Cloerkes, S. Dollase, D. Meier

Aufgabe 3 (Rekursion): (14 Punkte)

Auch in dieser Aufgabe soll eine fraktale Struktur mit Hilfe der Klasse `Canvas` gezeichnet werden. Diesmal geht es um Gosper-Kurven. Eine solche Kurve 0. Ordnung besteht aus einer geraden Linie. Kurven n-ter Ordnung für $n > 0$ werden gebildet, indem man sieben Gosper-Kurven $(n-1)$-ter Ordnung kombiniert. Dabei unterscheidet man zwischen Links-Kurven und Rechts-Kurven.
Eine Gosper-Links-Kurve n-ter Ordnung wird wie folgt konstruiert:

- Zunächst wird eine Gosper-Links-Kurve $(n-1)$-ter Ordnung gezeichnet
- Dann ein $-60°$ Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung
- Ein $-120°$ Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung
- Ein $60°$ Knick
- Eine Gosper-Links-Kurve $(n-1)$-ter Ordnung
- Ein $120°$ Knick
- Zwei Gosper-Links-Kurven $(n-1)$-ter Ordnung
- Ein $60°$ Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung
- Die nächste Kurve setzt dann im Winkel von $-60°$ an.

Gosper-Rechts-Kurven n-ter Ordnung werden ähnlich erzeugt:

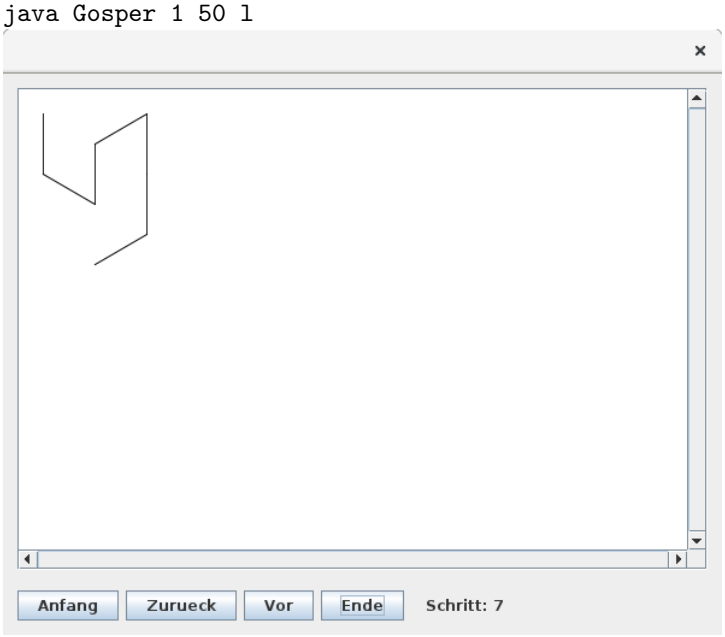
- Nach der vorhergehenden Kurve wird $60°$ nach rechts angesetzt.
- Dann folgt eine Gosper-Links-Kurve $(n-1)$-ter Ordnung
- Ein $-60°$ Knick
- Zwei Gosper-Rechts-Kurven $(n-1)$-ter Ordnung
- Ein $-120°$ Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung
- Ein $-60°$ Knick
- Eine Gosper-Links-Kurve $(n-1)$-ter Ordnung
- Ein $120°$ Knick
- Eine Gosper-Links-Kurve $(n-1)$-ter Ordnung
- Ein $60°$ Knick
- Eine Gosper-Rechts-Kurve $(n-1)$-ter Ordnung

Positive Winkel bedeuten eine Ecke im Uhrzeigersinn, negative Winkel eine Ecke entgegen dem Uhrzeigersinn. Wie in der vorigen Aufgabe dürfen Sie in dieser Aufgabe keine Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt.
Implementieren Sie die statischen Methoden `gosperLinks` und `gosperRechts` in der Klasse `Gosper`, die im Moodle-Lernraum verfügbar ist. Diese sollen folgende Parameter erhalten:

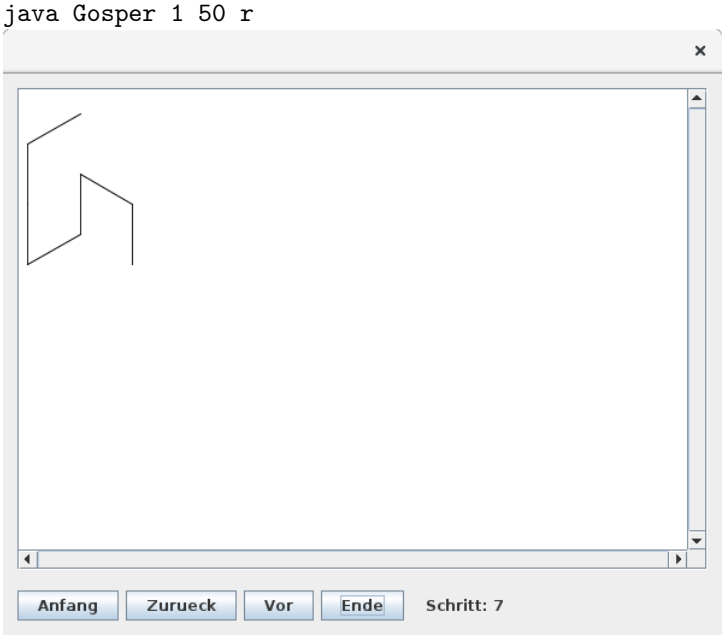
- eine Referenz `c` auf ein `Canvas` Objekt
- eine `int`-Zahl, welche die gewünschte Ordnung der Kurve angibt.
- eine `int`-Zahl, welche die Länge einer Gosper-Kurve 0. Ordnung angibt.

Diese Methode soll eine Gosper-Kurve der spezifizierten Ordnung zeichnen.
Zum Testen Ihrer Implementierung enthält die Klasse `Gosper` schon eine `main`-Methode. Das Programm bekommt bis zu drei Parameter. Der erste gibt die Ordnung der Kurve an, der zweite die Länge der Kurven 0. Ordnung aus denen sie zusammengesetzt werden soll. Der dritte Parameter gibt an, ob es eine Links-Kurve (`l`) oder eine Rechts-Kurve (`r`) sein soll. Aus der `main`-Methode wird die Methode `gosperLinks` bzw. `gosperRechts` entsprechend aufgerufen. Sie können Ihre Implementierung mit folgenden Aufrufen testen (darunter finden Sie Abbildungen, die Sie als Ergebnis zu diesen Aufrufen erhalten sollten):

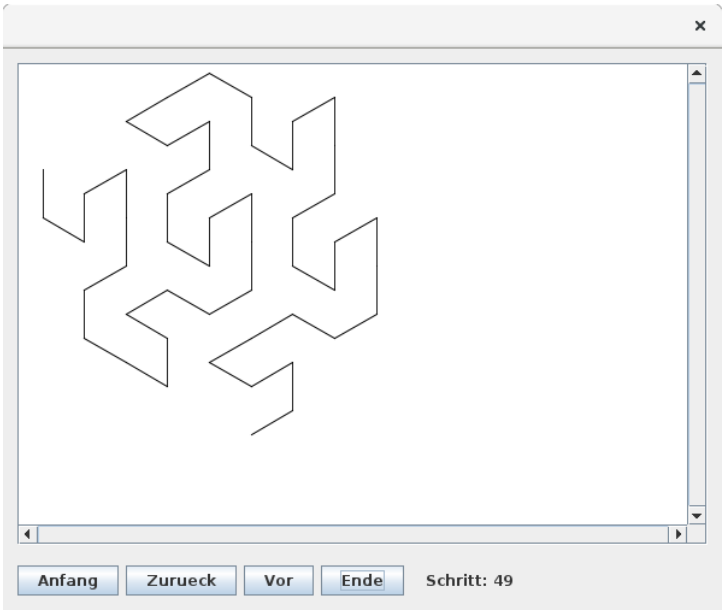
- `java Gosper 1 50 l`



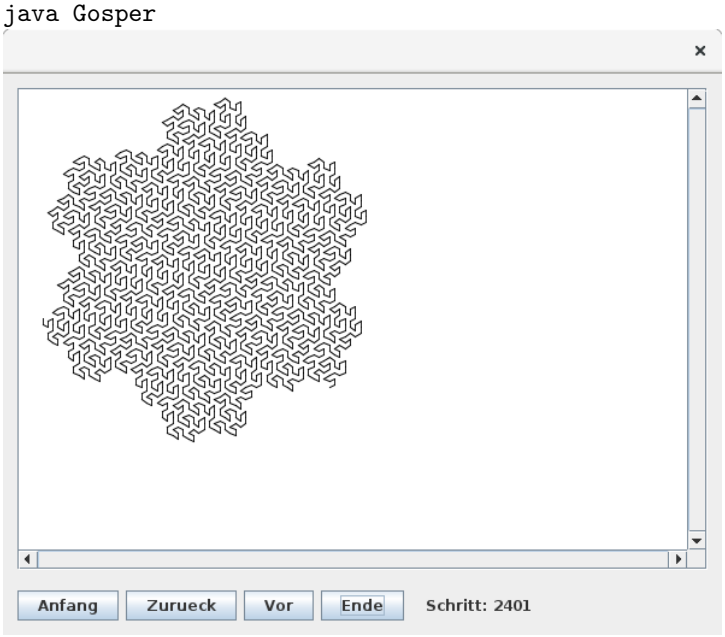
- `java Gosper 1 50 r`



- `java Gosper 2 40 l`



- `java Gosper`



Hinweise:

- Klicken Sie einmal auf die Schaltfläche Ende, um das Ergebnis anzuzeigen.
- Mit den Schaltflächen Vor und Zurueck können Sie die Zeichnung schrittweise auf- bzw. abbauen. Der Ablauf entspricht dabei dem Ablauf Ihres Programms. Die Schaltflächen Anfang und Ende springen zum Anfang bzw. Ende des Ablaufs.

Aufgabe 6 (Rekursive Datenstrukturen): (36 Punkte)

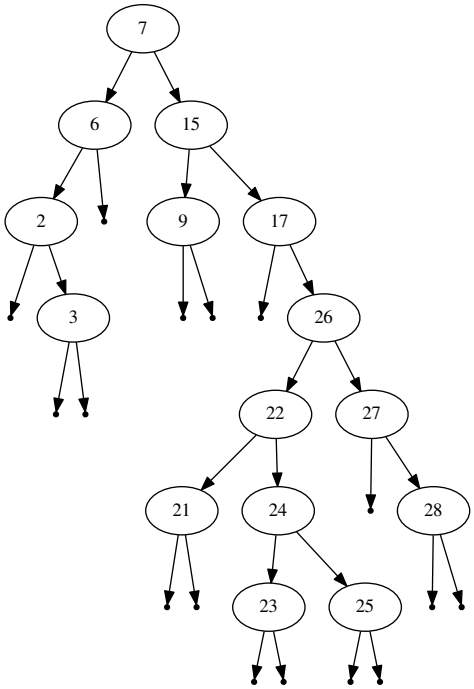
In dieser Aufgabe sollen einige rekursive Algorithmen auf sortierten Binärbäumen implementiert werden.
Aus dem Moodle-Lernraum können Sie die Klassen `Tree` und `TreeNode` herunterladen. Die Klasse `Tree` repräsentiert einen Binärbaum, entsprechend der Klasse Baum auf den Vorlesungsfolien. Einzelne Knoten des Baumes werden mit der Klasse `TreeNode` dargestellt. Alle Methoden, die Sie implementieren, sorgen dafür, dass in dem Teilbaum `left` nur Knoten mit kleineren Werten als in der Wurzel liegen und in dem Teilbaum `right` nur Knoten mit größeren Werten.
Um den Baum zu visualisieren, ist eine Ausgabe als dot Datei bereits implementiert. In dieser einfachen Beschreibungssprache für Graphen steht eine Zeile `x -> y;` dafür, dass der Knoten `y` ein Nachfolger des Knotens `x` ist. In Dateien, die von dem vorgegebenen Code generiert wurden, steht der linke Nachfolger eines Knotens immer vor dem rechten Nachfolger in der Datei. Optional können Sie mit Hilfe der Software Graphviz, wie unten beschrieben, automatisch Bilder aus dot Dateien generieren.
Die Klasse `Tree` enthält außerdem eine `main` Methode, die einige Teile der Implementierung testet.

Am Schluss dieser Aufgabe sollte der Aufruf `java Tree t1.dot t2.dot` eine Ausgabe der folgenden Form erzeugen. Die Zahlen sind teilweise Zufallszahlen.

```
Aufgabe b): Zufaelliges Einfuegen
Baum als DOT File ausgegeben in Datei t1.dot
Aufgabe a): Suchen nach zufaelligen Elementen
17 ist enthalten
19 ist nicht enthalten
12 ist nicht enthalten
15 ist enthalten
12 ist nicht enthalten
13 ist nicht enthalten
3 ist enthalten
17 ist enthalten
2 ist enthalten
15 ist enthalten
26 ist enthalten
9 ist enthalten
18 ist nicht enthalten
29 ist nicht enthalten
Aufgabe c): geordnete String-Ausgabe
tree[2, 3, 6, 7, 9, 15, 17, 21, 22, 23, 24, 25, 26, 27, 28]
Aufgabe d): Suchen nach vorhandenen Elementen mit Rotation.
Baum nach Suchen von 15, 3 und 23 als DOT File ausgegeben in Datei t2.dot
```

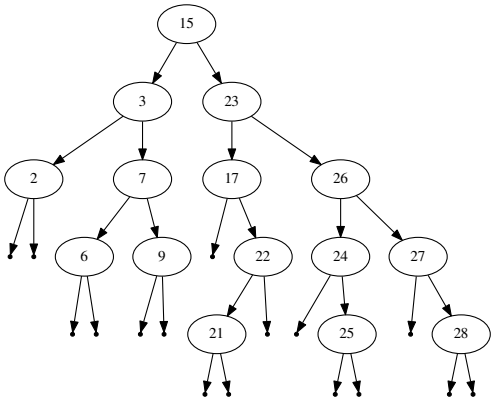
Falls Sie anschließend mit `dot -Tpdf t1.dot > t1.pdf` und `dot -Tpdf t2.dot > t2.pdf` die dot Dateien in PDF umwandeln[1], sollten Sie Bilder ähnlich zu den Folgenden erhalten.

[1] Sie benötigen hierfür das Programm Graphviz



Listing 1: t1.dot

```
digraph {
graph [ordering="out"];
7 -> 6;
6 -> 2;
null0[shape=point]
2 -> null0;
2 -> 3;
null1[shape=point]
3 -> null1;
null2[shape=point]
3 -> null2;
null3[shape=point]
6 -> null3;
7 -> 15;
15 -> 9;
null4[shape=point]
9 -> null4;
null5[shape=point]
9 -> null5;
15 -> 17;
null6[shape=point]
17 -> null6;
17 -> 26;
26 -> 22;
22 -> 21;
null7[shape=point]
21 -> null7;
null8[shape=point]
21 -> null8;
22 -> 24;
24 -> 23;
null9[shape=point]
23 -> null9;
null10[shape=point]
23 -> null10;
24 -> 25;
null11[shape=point]
25 -> null11;
null12[shape=point]
25 -> null12;
26 -> 27;
null13[shape=point]
27 -> null13;
27 -> 28;
null14[shape=point]
28 -> null14;
null15[shape=point]
28 -> null15;
}
```



Listing 2: t2.dot

```
digraph {
graph [ordering="out"];
15 -> 3;
3 -> 2;
null0[shape=point]
2 -> null0;
null1[shape=point]
2 -> null1;
3 -> 7;
7 -> 6;
null2[shape=point]
6 -> null2;
null3[shape=point]
6 -> null3;
7 -> 9;
null4[shape=point]
9 -> null4;
null5[shape=point]
9 -> null5;
15 -> 23;
23 -> 17;
null6[shape=point]
17 -> null6;
17 -> 22;
22 -> 21;
null7[shape=point]
21 -> null7;
null8[shape=point]
21 -> null8;
null9[shape=point]
22 -> null9;
23 -> 26;
26 -> 24;
null10[shape=point]
24 -> null10;
24 -> 25;
null11[shape=point]
25 -> null11;
null12[shape=point]
25 -> null12;
26 -> 27;
null13[shape=point]
27 -> null13;
27 -> 28;
null14[shape=point]
28 -> null14;
null15[shape=point]
28 -> null15;
}
```

Wie oben erwähnt, sind die meisten Zahlen zufällig bei jedem Aufruf neu gewählt. In jedem Fall aber sollten die obersten Knoten in der zweiten Grafik die Zahlen 3, 15 und 23 sein.

In dieser Aufgabe dürfen Sie *keine* Schleifen verwenden. Die Verwendung von Rekursion ist hingegen erlaubt.

a) Implementieren Sie Methoden zum Suchen nach einer Zahl im Baum.

Die Methode `simpleSearch` in der Klasse `Tree` prüft, ob eine Wurzel existiert (d.h., ob der Baum nicht leer ist). Falls er leer ist, wird sofort `false` zurück gegeben. Existiert hingegen die Wurzel, wird ihre Methode `simpleSearch` aufgerufen.

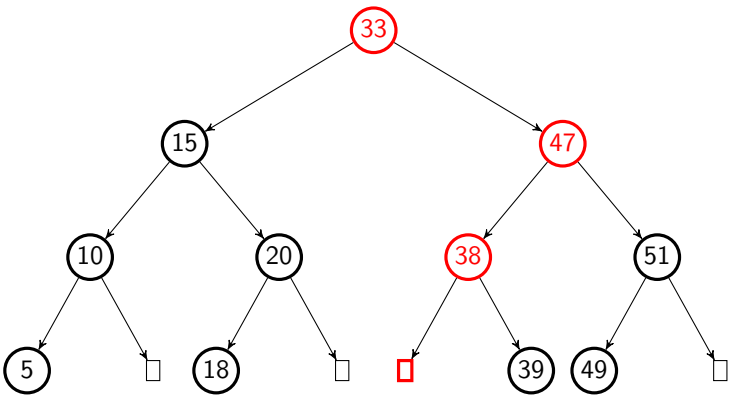
Die Methode `simpleSearch` der Klasse `TreeNode` durchsucht nun den Baum nach der übergebenen Zahl. Hat der aktuelle Knoten den gesuchten Wert gespeichert, soll `true` zurückgegeben werden. Andernfalls wird eine Fallunterscheidung durchgeführt. Da der Baum sortiert ist, wird nach Zahlen, die kleiner sind als der im aktuellen Knoten gespeicherte Wert, nur im linken Teilbaum weiter gesucht. Für Zahlen, die größer sind, muss nur im rechten Teilbaum gesucht werden. Trifft diese Suche irgendwann auf `null`, kann die Suche abgebrochen werden und es wird `false` zurückgegeben.

b) Implementieren Sie Methoden zum Einfügen einer Zahl in den Baum. Vervollständigen Sie dazu die

Methoden `insert` in den Klassen `TreeNode` und `Tree`.

In der Klasse `Tree` muss zunächst überprüft werden, ob eine Wurzel existiert. Falls nein, so sollte das neue Element als Wurzel eingefügt werden. Existiert eine Wurzel, dann wird `insert` auf der Wurzel aufgerufen. In der Klasse `TreeNode` wird zunächst nach der einzufügenden Zahl gesucht. Wird sie gefunden, braucht nichts weiter getan zu werden (die Zahl wird also kein zweites Mal eingefügt). Existiert die Zahl noch nicht im Baum, muss ein neuer Knoten an der Stelle eingefügt werden, wo die Suche abgebrochen wurde.

Wird zum Beispiel im folgenden Baum die Zahl 36 eingefügt, beginnt die Suche beim Knoten 33, läuft dann über den Knoten 47 und wird nach Knoten 38 abgebrochen, weil der linke Nachfolger fehlt. An dieser Stelle, als linker Nachfolger von 38, wird nun die 36 eingefügt.



Hinweise:

Obwohl dem eigentlichen Einfügen eine Suche vorausgeht, ist es nicht sinnvoll, die Methode `simpleSearch` in dieser Teilaufgabe zu verwenden.

c) Schreiben Sie `toString` Methoden für die Klassen `Tree` und `TreeNode`.

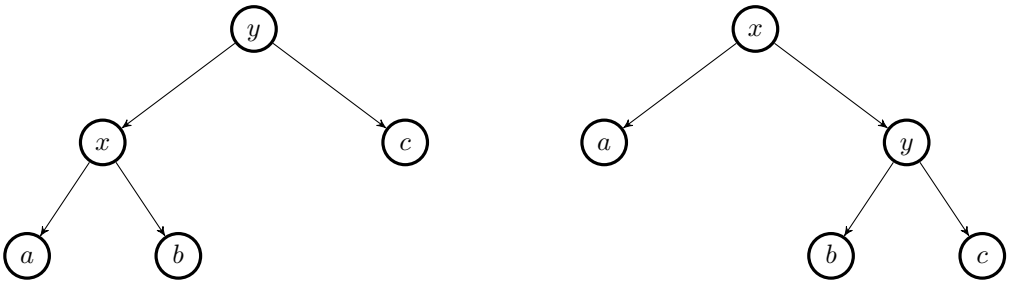
Die `toString` Methode der Klasse `TreeNode` soll alle Zahlen, die im aktuellen Knoten und seinen Nachfolgern gespeichert sind, aufsteigend sortiert und mit Kommas getrennt ausgeben. Ruft man beispielsweise `toString` auf dem Knoten aus dem Baum oben auf, der die Zahl 15 gespeichert hat, wäre die Ausgabe `5, 10, 15, 18, 20`.

Die `toString` Methode der Klasse `Tree` soll die Ausgabe `tree[5, 10, 15, 18, 20, 33, 38, 39, 47, 49, 51]` für das obige Beispiel erzeugen.

d) Implementieren Sie in dieser Teilaufgabe die Methoden `search` und `rotationSearch` in der Klasse `Tree` beziehungsweise `TreeNode`. Diese sollen einen alternativen Algorithmus zur Suche nach einem Wert im Baum implementieren.

Es ist sinnvoll, Elemente, nach denen häufig gesucht wird, möglichst weit oben im Baum zu speichern. Das kann realisiert werden, indem der Baum beim Aufruf der Suche so umstrukturiert wird, dass das gesuchte Element, falls es existiert, in der Wurzel steht und die übrige Struktur weitgehend erhalten wird. Da außerdem unbedingt die Sortierung erhalten bleiben muss, sollte ein spezieller Algorithmus verwendet werden.

Um einen Knoten eine Ebene im Baum nach oben zu befördern, kann die sogenannte *Rotation* verwendet werden. Soll im folgenden Beispiel x nach oben rotiert werden, wird die `left` Referenz des Vorgängerknotens y auf die `right` Referenz von x gesetzt. Anschließend wird die `right` Referenz von x auf y gesetzt. Das Ergebnis ist der rechts daneben gezeichnete Baum. Um im rechten Baum y nach oben zu rotieren, wird die Operation spiegelbildlich ausgeführt.



Diese Rotation kann nun so lange wiederholt werden, bis der Knoten mit der gesuchten Zahl in der Wurzel ist. Ist die gesuchte Zahl nicht enthalten, wird der Knoten, bei dem die Suche erfolglos abgebrochen wird, in die Wurzel rotiert.

Hinweise:

Die Signatur und Dokumentation der vorgegebenen Methoden geben Ihnen weitere Hinweise, wie die Rotation eines Knotens in die Wurzel rekursiv implementiert werden kann.

Aufgabe 7 (Deck 5): (Codescape)

Lösen Sie die Missionen von Deck 5 des Codescape Spiels. Ihre Lösung für die Codescape Missionen wird nur dann für die Zulassung gezählt, wenn sie Ihre Lösung vor der einheitlichen Codescape Deadline am Samstag, den 16.01.2021, um 23:59 Uhr abschicken.