

Didaktische Hinweise

Programmiersprachenunabhängige Darstellung von Algorithmen

Zielgruppe

Der Leitfaden zur programmiersprachenunabhängigen Darstellung von Algorithmen in Form von Struktogrammen richtet sich an Schülerinnen und Schüler in der Einführungsphase. Er kann aber natürlich auch für jüngere Schülerinnen und Schüler verwendet werden, die bereits über die entsprechenden inhaltlichen Voraussetzungen verfügen.

Voraussetzungen

Die meisten algorithmischen Probleme, die in der Einführungsphase gelöst werden, sind so überschaubar, dass eine vorherige Planung als Struktogramm nicht notwendig ist. Struktogramme werden hier daher mit dem Ziel eingeführt, grundlegende Strategien in der Algorithmik unabhängig von einer konkreten Programmiersprache herauszuarbeiten. Die Schülerinnen und Schüler sollten daher bereits mehr als eine Programmiersprache kennengelernt haben. Zumindest innerhalb der Lerngruppe sollte mehr als eine Sprache bekannt sein. Weiterhin werden verschiedene Beispiele aus dem Bereich Zeichenkettenverarbeitung verwendet, so dass die Schülerinnen und Schüler in mindestens einer Programmiersprache mit der Zeichenkettenverarbeitung vertraut sein sollten.

Lernziele

Mithilfe des Leitfadens zur programmiersprachenunabhängigen Darstellung von Algorithmen kann der Entwurf von Struktogrammen erlernt werden, der in Niedersachsen als standardisierte Darstellung von Algorithmen verwendet wird (vgl. [1] und [2]). Eine solche programmiersprachenunabhängige Darstellung erscheint dann besonders sinnvoll, wenn man mehr als eine Programmiersprache kennengelernt hat und sich unabhängig von der verwendeten Sprache über Algorithmen unterhalten möchte. Die Struktogramme werden daher vor dem Hintergrund eingeführt, mindestens zwei Programmiersprachen zu vergleichen und Gemeinsamkeiten und Unterschiede herauszuarbeiten. Die Schülerinnen und Schüler sollten nach dieser Einheit grundlegende algorithmische Arbeitsweisen von sprachspezifischen Eigenheiten unterscheiden können. Abschließend wird reflektiert, wann der Einsatz von Struktogrammen hilfreich sein kann.

Didaktische Hinweise

Es soll deutlich werden, dass der Entwurf von Algorithmen unabhängig von der konkreten Programmiersprache erfolgen kann. Für Probleme im Bereich Zeichenkettenverarbeitung lässt sich dies besonders gut zeigen, da die Operationen, die sowohl die textbasierten als auch die grafischen Programmiersprachen hierzu anbieten, sehr ähnlich sind. Beispiele aus dem Bereich Zeichnen oder Spiele eignen sich hier weniger, da die grafischen Sprachen wie *Scratch*¹ oder *Snap!*² nach dem Prinzip der Turtlegrafik arbeiten und viele hilfreiche Operationen, wie das Abprallen vom Rand oder das Abfragen des Berührens von anderen Objekten oder Farben zur Verfügung stellen, die in

¹ Scratch ist ein Projekt der Scratch Foundation in Zusammenarbeit mit der Lifelong Kindergarten Group des MIT Media Lab. Es ist kostenlos unter <https://scratch.mit.edu> erhältlich.

² Snap! wird von der University of California, Berkeley zur Verfügung gestellt: <https://snap.berkeley.edu>

Processing³ erst selbst programmiert werden müssten. Processing bietet hingegen fertige Methoden zum Zeichnen geometrischer Objekte, die in den grafischen Sprachen meist selbst erstellt werden müssen. Ein Vergleich würde hier daher den Eindruck erwecken, die Sprachen wären unterschiedlich mächtig und den Blick auf die Gemeinsamkeiten und algorithmischen Grundstrukturen verstellen. Darauf sollte geachtet werden, wenn der Vergleich der Programmiersprachen in einem anderen Kontext als der Zeichenkettenverarbeitung initiiert wird.

Statt Processing kann natürlich auch eine andere Programmiersprache verwendet werden. Die grundlegenden Zeichenkettenoperationen und algorithmischen Kontrollstrukturen sollten in jeder im Unterricht eingesetzten Programmiersprache zu finden sein. Da die Schülerinnen und Schüler im Unterricht in der Regel nur objektorientierte Sprachen mit imperativem Kern verwenden, sind diese gemeint, wenn im Leitfaden nur allgemein von Programmiersprachen gesprochen wird.

Im Idealfall haben alle Schülerinnen und Schüler der Lerngruppe bereits mit beiden betrachteten Programmiersprachen gearbeitet. Wenn die Schülerinnen und Schüler parallel mit verschiedenen Sprachen gearbeitet haben, kann diese Einheit aber auch der Zusammenführung dienen. Die Aufgaben sollten dann von Teams oder Kleingruppen bearbeitet werden, in denen Anwender*innen beider Programmiersprachen vertreten sind.

Die Einführung erfolgt hier bewusst erst, nachdem die Schülerinnen und Schüler bereits einige Erfahrungen mit der Implementierung von Algorithmen gesammelt haben. Das experimentelle Vorgehen ist für die Schülerinnen und Schüler wichtig. Sie benötigen die direkte Rückmeldung, ob ihr Programm funktioniert, und müssen sich erst einen Überblick über den Werkzeugkasten verschaffen, den eine Programmiersprache bietet. Dieses Vorgehen soll auch nicht durch den Entwurf von Struktogrammen ersetzt werden. Struktogramme sollen vielmehr ein zusätzliches Instrument darstellen, um Erfahrungen zu strukturieren oder Ideen schematisch darzustellen, um sich darüber austauschen zu können. Als Motivation kann hier die Kommunikation mit Mitschüler*innen (Programmierer*innen) dienen, die möglicherweise eine andere Programmiersprache verwenden.

Die Darstellung der Struktogramme orientiert sich an der Schreibweise, die in den ergänzenden Hinweisen zum Kerncurriculum für die Sek II (s. [2]) zu finden ist. Da die Schüler*innen noch nicht zwingend mit eigenen Blöcken oder Methoden gearbeitet haben, wird hier jedoch davon ausgegangen, dass ein Struktogramm jeweils für ein vollständiges Programm und nicht für eine einzelne Operation erstellt wird. Auf die Kopfzeile mit dem Namen des Struktogramms könnte auch jeweils verzichtet werden.

Die Aufgaben

Die Aufgaben 1 bis 3 dienen der Erarbeitung der Gemeinsamkeiten und Unterschiede zwischen den bereits verwendeten Programmiersprachen.

In den Aufgaben 4, 5 und 6 wird das Erstellen von Struktogrammen geübt. Der Algorithmus für die Caesar-Verschlüsselung ist den Schülerinnen und Schülern vermutlich bekannt. Ansonsten bietet sich hier ein anderer Algorithmus an, den die Schülerinnen und Schüler bereits kennen, damit sie sich auf die korrekte Darstellung der algorithmischen Strukturen in einem Struktogramm konzentrieren können. In Aufgabe 6 wird auf Tätigkeiten aus dem Alltag zurückgegriffen. Dabei wird zum einen

³ Die Programmierumgebung Processing wurde 2001 von Ben Fry und Casey Reas initiiert. Nähere Informationen finden Sie unter <https://processing.org/>

deutlich, dass einem auch in anderen Kontexten algorithmische Strukturen begegnen. Zum anderen fällt es den Schülerinnen und Schülern dabei vermutlich leichter, sich von einer konkreten Programmiersprache zu lösen. Da die Formulierung einer Alltagsstätigkeit als Algorithmus ungewohnt ist, ist die Herausforderung dabei, die algorithmischen Strukturen herauszuarbeiten.

In Aufgabe 7 und 8 soll ein Struktogramm als erster Entwurf für ein unbekanntes Problem erstellt werden. An die Implementierung in der selbst gewählten Programmiersprache sollte sich eine Reflexionsphase über die Vor- und Nachteile der verschiedenen Programmiersprachen in Bezug auf das jeweilige Problem anschließen. Dabei wird deutlich, dass die Umsetzung mancher Probleme in den verschiedenen Sprachen sehr ähnlich ist (z. B. Aufgabe 8), während es bei anderen Problemen Unterschiede gibt (z. B. Aufgabe 7). Diese sind meist auf Operationen zurückzuführen, die nur in einer Sprache als fertige Blöcke oder Methoden zur Verfügung stehen, während sie in der anderen selbst implementiert werden müssen. Der Grund können aber auch Konzepte wie der Einsatz von Objekten oder das Erstellen von eigenen Operationen sein, die in den beiden Sprachen unterschiedlich weit beherrscht werden. Diese Unterschiede können sich im Verlauf der Kompetenzentwicklung bezüglich des Programmierens in der jeweiligen Sprache relativieren.

In Aufgabe 9 wird ein grobes Struktogramm für ein komplexeres Problem angeboten. Je stärker dieses verfeinert und konkretisiert wird, desto deutlicher wird, dass die Programmiersprachen-unabhängigkeit bei der konkreten Umsetzung an ihre Grenzen stößt. So stellen manche Programmiersprachen z. B. eine Operation, die überprüft, ob ein Zeichen, eine Ziffer ist, direkt zur Verfügung, während dies in anderen Sprachen z. B. anhand des ASCII-Codes selbst getestet werden muss. An dieser Stelle eignet es sich ggf., falls noch nicht geschehen, das Erstellen von eigenen Blöcken bzw. Methoden einzuführen.

Weiterhin gibt es hier unterschiedliche Möglichkeiten, der algorithmischen Verfeinerung. So können Variablen eingeführt werden, um sich beispielsweise zu merken, ob das vorherige Zeichen eine Ziffer war. Oder diese wird jedes Mal direkt getestet, wobei auf die Ränder geachtet und zuerst geprüft werden muss, ob es überhaupt ein vorheriges Zeichen gibt. Eine mögliche Implementierung enthält die Musterlösung zu Aufgabe 12 im Leitfaden zur Zeichenkettenverarbeitung.

Am geschicktesten lässt sich diese Aufgabe mithilfe eines endlichen Automaten lösen, so dass diese Aufgabe zu gegebener Zeit ggf. wieder aufgegriffen werden kann.

In Aufgabe 10 wird abschließend reflektiert, wann der Einsatz von Struktogrammen hilfreich ist. In den Aufgaben 7 und 8 haben viele Schülerinnen und Schüler vermutlich die Erfahrung gemacht, dass ein Struktogramm, wenn das Ziel die Implementierung ist, nicht benötigt wird. Wenn die Grundidee bereits klar ist, bietet die direkte Implementierung hier viel einfacher die Möglichkeiten, durch die Rückmeldungen des Systems, Schwachstellen im Algorithmus aufzudecken und zu verbessern. Bei komplexeren Problemen wie in Aufgabe 9 oder von Schüler*innen die Schwierigkeiten haben einen Ansatz zu finden, wird ein Struktogramm hingegen möglicherweise als hilfreich empfunden. Diese Einschätzung kann aber individuell sehr verschieden sein. Manche sehen algorithmische Konzepte durch die Darstellung als Struktogramm klarer, andere durchschauen diese auch durch die direkte Erfahrung beim Implementieren. In diesem Fall ist es dann eher ein Werkzeug zur Kommunikation und Dokumentation, das vor allem mit Blick auf eine Abiturprüfung relevant wird.

Zu den Aufgaben liegen Musterlösungen bei. Für die korrekte Ausführbarkeit wird keine Garantie übernommen. Auch für Folgeschäden, die sich aus der Anwendung der Quelltexte für die Musterlösungen oder Beispiele des Leitfadens oder durch eventuelle fehlerhafte Angaben ergeben, wird keine Haftung oder juristische Verantwortung übernommen.

Literatur

[1] Niedersächsisches Kultusministerium (Hrsg.) (2014). Kerncurriculum für die Schulformen des Sekundarbereichs I Schuljahrgänge 5 – 10. Informatik. Hannover: Unidruck

[2] Niedersächsisches Kultusministerium (Hrsg.) (2021) Ergänzende Hinweise zum Kerncurriculum Informatik für die gymnasiale Oberstufe am Gymnasium, an der Gesamtschule sowie für das Kolleg.
<https://cuvo.nibis.de/cuvo.php?p=download&upload=260> [Datum des Zugriffs: 18.10.2022]

Lizenz

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht kommerziell - Keine Bearbeitungen 4.0 International Lizenz](#).