25. Bundeswettbewerb Informatik, 2. Runde

Lösungshinweise und Bewertungskriterien

Allgemeines

Es ist immer wieder bewundernswert, wie viel an Ideen und Wissen, an Fleiß und Durchhaltevermögen in den Einsendungen zur zweiten Runde eines Bundeswettbewerbs Informatik steckt. Um aber die Allerbesten für die Endrunde zu bestimmen, müssen wir Ihre Arbeit kritisch begutachten und hohe Anforderungen stellen. Von daher sind Punktabzüge die Regel und Bewertungen über das Soll (5 Punkte) hinaus die Ausnahme, insbesondere bei den schwierigen Aufgaben dieses Jahres. Lassen Sie sich davon nicht entmutigen! Wie auch immer ihre Einsendung bewertet wurde: Allein durch die Arbeit an den Aufgaben und den Einsendungen hat jede Teilnehmerin und jeder Teilnehmer einiges dazu gelernt; den Wert dieses Effektes sollten Sie nicht unterschätzen.

Bevor Sie sich in die Lösungshinweise vertiefen, lesen Sie doch bitte kurz die folgenden Anmerkungen zu Einsendungen und den beiliegenden Unterlagen durch.

Terminprobleme Einige Einsender gestehen ganz offen, dass Ihnen die Zeit zum Einsendeschluss hin knapp geworden ist, worauf wir bei der Bewertung natürlich keine Rücksicht nehmen können. Abiturienten macht der Terminkonflikt mit der Abiturvorbereitung Probleme. Der ist für eine erfolgreiche Teilnahme sicher nicht ideal. In der zweiten Jahreshälfte läuft aber die 2. Runde des Mathewettbewerbs, dem wir keine Konkurrenz machen wollen. Also bleibt uns nur die erste Jahreshälfte. Aber: Sie haben etwa vier Monate Bearbeitungszeit für die 2. Runde. Rechtzeitig mit der Bearbeitung der Aufgaben zu beginnen ist der beste Weg, Konflikte mit dem Abitur zu vermeiden.

Dokumentation Es ist sehr gut nachvollziehbar, dass Sie Ihre Energie bevorzugt in die Lösung der Aufgaben, die Entwicklung ihrer Ideen und die Umsetzung in Software fließen

lassen. Doch ohne eine gute Beschreibung der Lösungsideen, eine übersichtliche Dokumentation der wichtigsten Komponenten Ihrer Programme, eine gute Kommentierung der Quellcodes und eine ausreichende Zahl sinnvoller Beispiele (die die verschiedenen bei der Lösung des Problems zu berücksichtigenden Fälle abdecken) ist eine Einsendung wenig wert. Bewerterinnen und Bewerter können die Qualität Ihrer Einsendung nur anhand dieser Informationen vernünftig einschätzen. Mängel können nur selten durch gründliches Testen der eingesandten Programme ausgeglichen werden – wenn diese denn überhaupt ausgeführt werden können: Hier gibt es häufig Probleme, die meist vermieden werden könnten, wenn Lösungsprogramme vor der Einsendung nicht nur auf dem eigenen, sondern auch einmal auf einem fremden Rechner getestet würden. Insgesamt sollte die Erstellung des schriftlichen Materials die Programmierarbeit begleiten oder ihr teilweise sogar vorangehen: Wer nicht in der Lage ist, Idee und Modell präzise zu formulieren, bekommt keine saubere Umsetzung in welche Programmiersprache auch immer hin.

Bewertungsbögen Kein Kreuz in einer Zeile bedeutet, dass die genannte Anforderung den Erwartungen entsprechend erfüllt wurde. Vermerkt wird also in der Regel nur, wenn davon abgewichen wurde – nach oben oder nach unten. Ein Kreuz in der Spalte „+“ bedeutet Zusatzpunkte, ein Kreuz unter „–“ bedeutet Minuspunkte für Fehlendes oder Unzulängliches. Die Schattierung eines Feldes bedeutet, dass die entsprechenden Plus- bzw. Minuspunkte in der Regel nicht vergeben wurden.[1]

Lösungshinweise Bei den folgenden Erläuterungen handelt es sich um Vorschläge, nicht um die einzigen Lösungswege, die wir gelten ließen. Wir akzeptieren in der Regel alle Ansätze, die die gestellte Aufgabe vernünftig lösen und entsprechend dokumentiert sind. Einige Dinge gibt es allerdings, die – unabhängig vom gewählten Lösungsweg – auf jeden Fall diskutiert werden müssen. Zu jeder Aufgabe gibt es deshalb einen Abschnitt, indem gesagt wird, worauf bei der Bewertung letztlich geachtet wurde, zusätzlich zu den grundlegenden Anforderungen an Dokumentation und Quellcode.

[1] Ausnahmen bestätigen die Regel: Bei Aufgabe 3 ist beim 5. Bewertungspunkt (“äquivalente Formeln geeignet behandelt”) die Minus-Spalte fälschlicherweise schattiert; bei diesem Punkt konnten Plus- und Minuspunkte vergeben werden.

Aufgabe 1: Faltpolygone

1.1 Lösungsidee

Teil 1: Operationen

Diese Aufgabe behandelt das klassische Origami – diesmal jedoch digital. Ein (zunächst) rechteckiges Polygon soll entlang von (geraden) Linien gefaltet werden. Es ergibt sich quasi ein Stapel von (nicht zwangsläufig rechteckigen) Polygonen. Anschließend sollen Schnitte durch den Polygonstapel definiert werden können. Durch einen Schnitt zerfällt der Polygonstapel in zwei Teile. Die Auswirkungen werden aber erst nach dem Ausfalten ganz sichtbar, wenn also zumindest einer der beiden Teile durch Umkehrung der Faltungen (in umgekehrter Faltungsfolge) wieder „entstapelt“, also in ein planes Papierstück zurückverwandelt wird.

Die Aufgabenstellung nennt also schon die wichtigsten Operationen:

Falten Ein Faltpolygon wird entlang einer Geraden gefaltet (der Teil auf der einen Seite der Geraden soll dazu auf die andere Seite gespiegelt werden); es entsteht ein neues Faltpolygon.

Schneiden Ein Faltpolygon wird entlang einer Geraden durchgeschnitten; es entstehen mindestens zwei neue Faltpolygone.

Ausfalten Die letzte Faltung des Faltpolygons wird zurückgenommen.

Die Aufgabenstellung definiert Faltpolygone als die beim Falten, Schneiden und Ausfalten entstehenden Objekte. Sowohl geschichtete als auch ausgefaltete Polygonstapel fallen unter diesen Begriff und sinnvollerweise auch das ursprüngliche Blatt. Damit das Ausfalten funktionieren kann, muss immer die jeweils letzte Faltung gespeichert werden oder aus der Darstellung des Faltpolygons berechnet werden können.

Selbstverständlich muss auch eine Erzeuge-Operation definiert sein, die eine Faltpolygon-Instanz mit den Angaben zum initialen Papierbogen generiert. Nicht ganz so selbstverständlich ist die Visualisierung des Faltpolygons, insbesondere (aber nicht nur) des vollständig ausgefalteten. Eine solche Operation wird für Teil 4 benötigt; die Visualisierung ist aber eher nicht als Operation eines abstrakten Datentyps zu sehen. Mit Hilfe einer Funktion, die entscheidet, ob ein Faltpolygon vollständig ausgefaltet ist, und der obigen Operation „Ausfalten“ lässt sich das vollständige Ausfalten des Faltpolygons realisieren, das auch als eigenständige Operation zur Verfügung gestellt werden kann.

Teil 2: Darstellung im Rechner

Wir wollen hier einen Ansatz betrachten, der Polygon-basiert arbeitet und zwei Zustände des Faltpolygons gespeichert hält. Alternativen sind kantenbasierte Algorithmen sowie Algorithmen, die nur einen Zustand oder alle Zwischenzustände festhalten. Ob Polygone oder Kanten,

geometrische Objekte sollten auch als solche behandelt werden. Nicht geeignet sind rein pixelbasierte Ansätze, bei denen die geometrischen Berechnungen auf die Manipulation von Pixelmengen zurückgeführt werden müssen.

Die rechteckige Ausgangsform soll als Polygon abgespeichert werden oder anders gesagt als Liste von Kanten, wobei eine Kante ein Array von zwei Punkten ist und ein Punkt wiederum eine Struktur bestehend aus zwei Zahlwerten X und Y. Durch diese Struktur ist der Algorithmus nicht auf die vorgegebene rechteckige Ausgangsform festgelegt, sondern kann mit beliebigen polygonalen Ausgangsformen arbeiten. (Dies ist schon eine Erweiterung!) Bereits mit dieser Datenstruktur ist eine Visualisierung der Ausgangsform möglich, indem man durch die Linien läuft und diese zeichnet.

Knicke werden in diesem Ansatz als eine Liste von Linien verwaltet. Zusätzlich wird die aktuelle Situation – nach Anwendung aller bisherigen Knicke – als Liste von Polygonen verwaltet. In der Ausgangssituation enthält die Polygonliste genau das Ausgangspolygon, die Knickliste ist leer. Wird vom Benutzer ein Knick hinzugefügt, wird jedes Polygon der Polygonliste darauf überprüft, ob es vom Knick verändert wird. Hierzu sei o.B.d.A. festgelegt, dass immer alle Punkte links unterhalb des Knicks unverändert bleiben, während die Punkte rechts oberhalb des Knicks gespiegelt werden. Nun gibt es für ein Polygon drei Möglichkeiten:

1. Es liegt komplett links unterhalb des Knicks. In diesem Fall bleibt das Polygon unverändert erhalten.
2. Es liegt komplett rechts oberhalb des Knicks. In diesem Fall wird das komplette Polygon gespiegelt, wobei der Knick die Spiegelachse ist. Das Ausgangspolygon wird als leeres Polygon erhalten (warum dies sinnvoll ist, wird bei der Betrachtung des Schnittvorgangs ersichtlich!), das Spiegelpolygon wird zur bisherigen Liste hinzugefügt.
3. Der Knick schneidet das Polygon. In diesem Fall wird das Polygon in zwei Teilpolygone zerteilt. Jede Kante, die den Knick schneidet, wird dabei in Teilkanten zerlegt, die komplett links unter oder rechts über dem Knick liegen. Die Kanten werden in zwei Polygone einsortiert. Abschließend werden die Knickkanten als Abschnitte des Knicks in beide Polygone eingefügt. Das Polygon links unterhalb des Knicks ersetzt das bisherige Polygon, das Polygon rechts oberhalb wird gespiegelt und zur bisherigen Liste hinzugefügt.

Interessanterweise gilt dabei, dass – wenn man die Polygonliste jedes Mal rückwärts durchläuft – innerhalb der Polygonliste eine Reihenfolge beibehalten wird. Das erste Polygon in der Liste entspricht immer dem Teil des Papiers, der auf dem Tisch liegt, das letzte Polygon entspricht immer dem oben liegenden Teil. Durch diese Reihenfolge ist es bei der Visualisierung später recht einfach, eine Darstellung mit verdeckten Kanten zu ermöglichen.

Schnitte werden in diesem Ansatz auf Teilschnitte reduziert. Ein Teilschnitt ist dabei ein Schnitt durch ein Polygon in der Polygonliste. Nun muss dieser Teilschnitt auf das Ausgangspolygon übertragen und dort hinzugefügt werden. Das Polygon, das diesen Teilschnitt beinhaltet, ist durch eine Abfolge an Faltoperationen an die Stelle im aktuellen Faltpolygon gekommen, an der es sich im Moment befindet. Würde man das Polygon entsprechend wieder

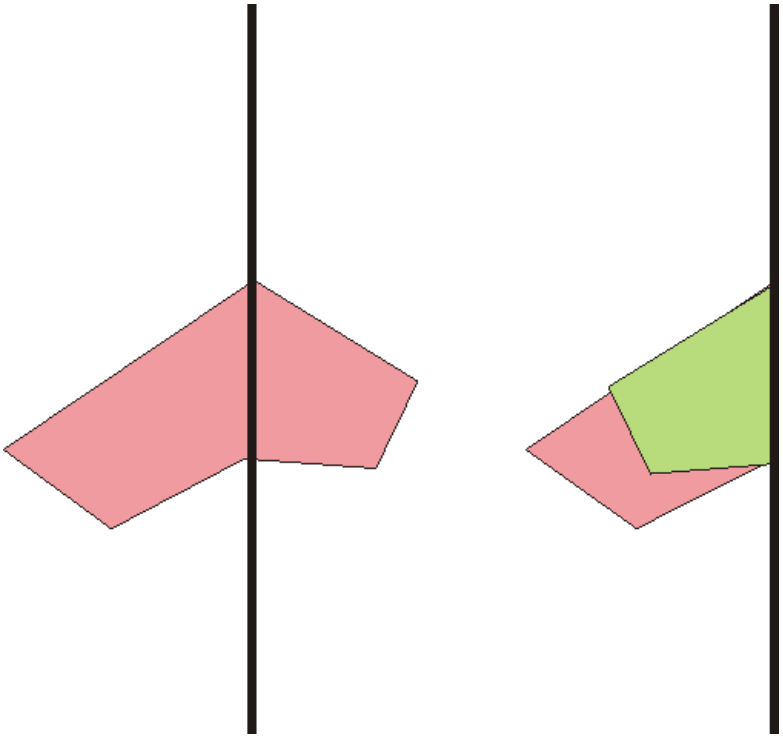

Abbildung 1: Ein Polygon und die daraus resultierenden Polygone nach dem Falten.

zurückfalten, käme es wieder an seiner ursprünglichen Stelle heraus. Entsprechend muss der Teilschnitt nur wie eine Kante zurückgefaltet werden und kommt dann auf seiner Position im Ausgangspolygon heraus. Um das Zurückfalten zu realisieren, ist es erforderlich, eine „Knickhistorie“ des Polygons zu kennen. Dazu reicht es aber aus, für jedes Polygon den Knick, durch das es entstanden ist, und das Polygon, aus dem es entstanden ist, mitzuhalten und diese Informationen rückwärts durchzugehen. Um in der Historie keine Inkonsistenzen zu erzeugen, ist es erforderlich, dass im Fall 2 beim Spiegeln (siehe oben) das alte Polygon als leeres Polygon erhalten bleibt – oder besser gesagt: als Polygon ohne Kanten, aber mit Historieninformationen.

1.2 Teil 4: Anzeige

Auch wenn das explizit in der Aufgabenstellung nicht gefordert wird, lässt sich ein ausgefaltetes Faltpolygon ohne grafische Darstellung nicht zeigen. Die oben schon angesprochene Visualisierung kommt hier also zur Anwendung. Die Anzeige sollte die Faltungen nachvollziehbar machen, z.B. mit Knicklinien; schön ist aber, wenn die Faltlinien auch weggelassen werden können. Die Folge der Operationen soll mit wenigen Schneideoperationen auskommen. Damit wird in der Aufgabenstellung beinahe belanglos ein kniffliger Punkt angesprochen. Es kann (mit großem Aufwand) bewiesen werden, dass für jedes beliebige Faltpoly-

gon nur eine einzige Schneide-Operation nötig ist. Hier soll nur klar werden, dass es für ein Faltpolygon mehrere erzeugende Operationenfolgen geben kann; offensichtlich ist, dass zwei zueinander achsensymmetrische Schnitte durch eine Faltung entlang der Symmetrieachse und einen Schnitt wirkungsgleich ersetzt werden können.

1.3 Bewertungshinweise

In Teil 1 sind die Mindestanforderungen offensichtlich: Falten, Schneiden und (einfaches) Ausfalten sind die zentralen Operationen eines Faltpolygons. Auch wenn das Interesse am Aussehen des vollständig ausgefalteten Faltpolygons groß ist, genügt es nicht, nur vollständiges Ausfalten als Operation zur Verfügung zu stellen; dies schränkt die Möglichkeiten des Datentyps unnötig ein. Insgesamt sollte die Idee eines abstrakten Datentyps zu ihrem Recht kommen; die Beschreibungen sollten noch keine Terminologie enthalten, die spezifisch für ein Programmier-Paradigma oder gar eine Programmiersprache ist.

Zu Teil 2: Wie schon oben angedeutet gibt es mehrere Möglichkeiten der Realisierung. Theoretisch ist es denkbar, an Stelle von Polygonen lediglich Kanten zu betrachten. Dieses Vorgehen ist jedoch sehr fragwürdig. Die Komplexität der Aufgabe und die Anzahl der zu beachtenden Fälle steigt hierbei stark an, das Programm wird komplizierter und somit kommt es in der Realisierung häufiger zu Fehlern. Ansätze, die sich nur einen Zustand (Startpolygon oder gefaltetes Polygon) merken, sind machbar, führen aber zu erhöhtem Rechenaufwand. Lösungen, die jeden Zwischenschritt abspeichern, sind ebenfalls realisierbar, bringen jedoch keinen wirklichen Vorteil, gemessen am vermehrten Speicherbedarf. Nicht sinnvoll ist ein rein pixelbasierter Ansatz. Die bei der Entwicklung des von der Aufgabenstellung eingeforderten Darstellungs-Vorschlags getroffenen Entwurfsentscheidungen sind natürlich zu begründen.

Zu Teil 3: Hier werden Stärken oder Schwächen der Realisierung berücksichtigt, die häufig schon aus den Vorgaben in den Teilen 1 und 2 resultieren. Der zentrale Aspekt einer Realisierung ist das Schneiden und Spiegeln von Kanten oder Flächen mit bzw. entlang einer (unendlich langen) Linie.

Zu Teil 4: Um nachvollziehbare Beispiele darstellen zu können, sollten Faltpolygone in jedem Zustand visualisiert werden können; für die drei Pflichtbeispiele sollten zumindest ausgewählte Zwischenzustände auch dokumentiert sein. Eine Ausgabe in Postscript oder ein anderes Grafikformat ist genauso gut (oder sogar besser) wie eine Anzeige am Bildschirm. Nicht akzeptabel ist jedoch eine reine Textausgabe wie etwa eine Liste von Koordinaten, die man miteinander verbinden muss, oder Ähnliches. Idealerweise sind in Zeiten von GUIs auch die Knicke und Schnitte grafisch eingebbar – also per Mausklick. Wenn man hier Koordinaten raten und eintippen muss, ist das wenig sinnvoll. Um Schnitte vernünftig platzieren zu können, sollte man die gefaltete Form sehen können. Wurde hier ein Hidden-Line-Algorithmus realisiert – was bei geschickter Datenhaltung kein wirklich großer Mehraufwand ist – wenn man also nur die Kanten sieht, die man auch bei einem Blatt Papier sehen würde, oder wenn die Darstellung wirklich gelungen ist, gibt es entsprechende Bonuspunkte. Zum Thema „Anzahl der nötigen Schneideoperationen“ sollte eine gute Lösung sich sinnvoll äußern.

Aufgabe 2: Bidoku

2.1 Problemstellung

Ein Bidoku ist ein vom bekannten Sudoku inspiriertes Denkspiel für Informatiker. In einem 16x16-Raster sollen Nullen und Einsen verteilt werden. Dabei ist das Raster in 16 Zeilen, 16 Spalten und 16 4x4-Unterquadrate unterteilt. Das Bidoku ist gelöst, wenn in jeder der Spalten und Zeilen und in jedem der Quadrate genau 8 Einsen und 8 Nullen stehen. Zu Beginn sind bereits so viele Zahlen eingetragen, dass eindeutig festgelegt ist, wie die anderen Felder belegt werden müssen. *Eindeutig festgelegt* bedeutet natürlich *nicht*, dass für den Spieler offensichtlich ist, was eingetragen werden muss. Dieser muss die Lücken durch Überlegen und Zählen nacheinander füllen.

Bidokus scheinen, wie bereits der Name sagt, stark mit Sudokus zusammenzuhängen. Bei den Lösungsstrategien gibt es allerdings deutliche Unterschiede. Übliche Sudoku-Techniken wie das Führen von Kandidatenlisten oder das Erarbeiten von Teillösungen sind im binären Fall gar nicht oder nur begrenzt anwendbar. Es müssen also neue Überlegungen angestellt werden.

Unser Hauptziel besteht darin, Bidokus zu generieren. Dabei müssen wir darauf achten, dass die generierten Bidokus *eindeutig* sind. Deshalb stellt sich die Frage, wie viele Zahlen wir vorverteilen möchten. Geben wir zu wenige Zahlen vor oder platzieren sie schlecht, so besitzt das Bidoku mehr als eine Lösung; setzen wir aber zu viele Zahlen, so ist das Lösen des Bidokus keine Herausforderung mehr und für den Spieler nicht spannend. Hier müssen wir eine Balance finden. Es kann uns auch passieren, dass wir die Zahlen so ungünstig platziert haben, dass man das Bidoku gar nicht mehr lösen kann. Mit solchen Bidokus würden wir unsere Rätselfreunde schnell vergraulen.

Außerdem fordert die Aufgabenstellung, dass der Benutzer den Schwierigkeitsgrad des Bidokus vor dem Erstellen wählen kann. Dabei ist zunächst einmal gar nicht klar, welche Bidokus besonders einfach und welche schwieriger sind: Wovon hängt der *Schwierigkeitsgrad* eines Bidokus ab?

Zum Abschluss wird in der Aufgabenstellung noch angeregt, dass man bei der Erstellung der Bidokus auch ästhetische Kriterien berücksichtigt. Hier kann man z.B. versuchen, symmetrische Bidokus zu erstellen. Wir werden später sehen, dass es je nach Lösungsansatz schwierig sein kann, die Form zu beeinflussen, die von den gesetzten Zahlen gebildet wird. Da dieser Aufgabenteil eher als Erweiterung zu sehen ist, werden wir nur kurz aufzeigen, wie man „schöne“ Bidokus erzeugen könnte.

2.2 Schwierigkeitsgrad

Die Wahl des Schwierigkeitsbegriffs wirkt sich darauf aus, wie wir unser Bidoku generieren können. Deshalb sollte man sich zu Beginn ein wenig Gedanken darüber machen, wie man

1	1	1	-	-	1	0	1	-	0	1	0	0	0	-	0
0	-	1	1	-	0	1	1	1	1	-	1	0	0	0	-
1	1	0	0	0	0	0	1	0	0	-	1	1	1	1	1
0	0	-	0	1	0	0	-	0	0	1	1	-	1	1	1
1	0	1	0	1	1	-	-	1	0	0	-	1	0	-	0
1	-	1	0	1	0	0	0	0	-	0	0	1	1	1	-
0	1	-	0	1	1	1	0	0	1	1	1	0	-	0	0
1	-	0	0	0	1	-	0	1	1	-	1	0	1	1	0
-	0	1	0	1	1	1	-	1	0	1	0	-	0	0	-
1	0	1	-	1	-	1	1	-	1	0	0	0	0	-	0
1	0	-	1	0	0	0	1	0	1	0	-	1	1	1	1
0	0	1	0	-	0	0	0	1	-	1	1	1	0	1	1
-	1	0	1	1	0	0	-	0	1	-	0	-	1	0	-
0	1	0	-	0	1	-	0	-	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0	1	0	0	1	1	-	0	0
0	-	0	1	0	-	1	0	1	0	-	1	1	0	0	1

Abbildung 2: Ein ganz einfaches Bidoku mit 48 freien Feldern.

die Schwierigkeit einteilen möchte. Am einfachsten ist es, den Schwierigkeitsgrad eines Bidokus in der Anzahl der fehlenden Zahlen zu messen. Aber auch andere Schwierigkeitsbegriffe sind denkbar: Die Schwierigkeit kann daran gemessen werden, wie viele Beobachtungen der Lösende kombinieren muss, um neue Zahlen einzusetzen. Man könnte auch in Betracht ziehen, wie kompliziert die Beobachtungen selbst sein müssen. In dieser Lösung werden wir uns darauf beschränken, eine Lösungsstrategie für die erstgenannte Variante zu entwickeln. Auf andere Möglichkeiten, die Schwierigkeit zu messen, werden wir nur kurz eingehen.

Beispiele für unterschiedlich schwierige Bidokus sind in den Abbildungen 2, 3, 4 und 5 zu sehen. Sie wurden mit Hilfe der Überlegungen der nächsten Abschnitte erstellt.

2.3 Bidokus erzeugen

Wie erzeugt man ein Bidoku? Dafür gibt es einige recht verschiedene Möglichkeiten, und tatsächlich können gute Lösungen der vorliegenden Aufgabe aus verschiedenen Ansätzen entstehen. Wir können nur einen Ausschnitt detailliert beschreiben und wollen daher in den folgenden Abschnitten einen Überblick über die verschiedenen Ansätze zur Erzeugung geben, die uns in den Sinn gekommen sind.

Die Bidokus, die wir generieren wollen, müssen die Kriterien der *Lösbarkeit* und der *Eindeutigkeit* erfüllen. Wie man ein Bidoku löst und auf Eindeutigkeit prüft, möchten wir erst in Abschnitt 2.4 näher besprechen. Einige der Möglichkeiten zur Bidoku-Erzeugung basieren allerdings darauf, dass man diese Dinge prüfen kann. Daher werden wir vorerst davon ausge-

0	1	-	0	-	1	1	0	-	1	1	-	1	-	-	0
-	1	-	0	0	-	1	1	0	1	0	-	1	-	1	-
0	1	0	1	-	1	0	-	-	1	1	1	0	0	1	-
1	1	1	1	0	0	1	0	0	-	0	0	1	-	-	0
0	-	1	0	1	-	-	-	-	1	-	1	0	-	1	0
-	0	0	0	-	1	0	1	1	0	0	-	-	1	1	1
1	1	-	-	-	-	-	1	1	-	0	1	0	0	-	0
1	1	1	-	1	0	0	0	0	0	-	0	0	1	1	-
1	-	1	0	0	-	0	0	-	0	0	1	-	0	1	1
-	1	-	0	0	1	0	-	1	0	0	1	0	1	0	0
-	0	-	0	1	0	1	-	1	1	0	-	0	1	-	1
0	-	0	-	1	1	-	1	-	1	-	0	-	0	0	0
-	-	1	1	1	-	0	1	1	0	1	-	0	0	0	-
0	0	-	1	-	0	1	0	0	-	1	0	1	-	0	1
-	0	0	1	1	1	0	0	1	0	1	0	1	-	0	1
1	-	1	1	-	0	0	-	-	0	1	0	0	0	0	1

Abbildung 3: Es fehlen noch 73 Zahlen.

0	1	-	0	-	-	0	1	-	-	-	-	1	-	-	0
-	0	-	1	1	-	1	1	1	1	-	-	-	-	0	-
-	-	1	0	-	0	1	-	-	1	-	1	1	0	-	-
1	0	0	-	0	0	0	0	1	-	0	0	-	-	-	1
1	-	-	0	0	-	-	-	-	1	-	0	0	-	-	1
-	0	0	0	-	0	-	-	1	1	1	-	-	-	1	1
0	1	-	-	-	-	-	0	-	-	0	0	0	0	0	1
1	1	1	-	1	-	1	1	1	0	-	0	0	-	-	-
1	-	1	-	0	-	1	0	-	-	1	1	-	-	-	0
-	0	-	-	0	0	0	-	0	1	-	-	-	1	-	1
-	1	1	1	-	-	1	-	1	0	1	-	0	-	-	0
-	-	0	-	0	0	-	1	-	0	1	1	1	1	1	1
-	-	1	0	0	-	1	-	0	1	1	-	0	0	0	-
1	1	-	0	-	0	0	1	0	-	-	0	0	-	-	1
-	-	0	-	0	1	0	1	1	0	-	0	1	-	1	1
0	-	1	0	-	-	-	-	-	1	1	0	1	1	1	0

Abbildung 4: Ein mittelschweres Bidoku mit 110 freien Feldern

0	-	-	0	-	-	0	0	-	-	-	-	1	-	-	1
-	-	-	1	1	-	-	0	0	-	1	-	-	-	0	-
-	-	1	1	-	1	1	-	1	-	1	0	-	1	-	-
0	0	0	-	1	0	0	0	0	-	1	-	-	-	-	-
0	-	1	1	0	-	-	-	-	1	-	1	1	-	-	-
-	1	1	1	-	1	1	1	1	-	-	-	-	-	0	0
0	-	-	-	-	-	-	1	-	-	0	0	1	0	0	0
0	-	0	-	0	-	1	1	1	1	-	1	0	0	-	-
0	-	0	-	0	-	1	-	-	-	0	-	-	-	0	0
-	0	-	-	1	0	1	-	0	-	-	-	-	1	0	-
-	-	-	1	-	1	0	-	0	-	-	-	1	1	-	1
0	-	-	-	1	1	1	0	1	1	-	0	-	-	0	-
-	-	1	1	0	-	1	-	1	1	-	-	0	-	0	-
1	0	-	0	-	0	0	-	0	-	-	0	0	-	-	1
-	-	0	-	1	0	-	1	1	-	-	0	0	-	-	0
1	-	-	0	-	0	-	-	-	0	-	0	0	0	-	-

Abbildung 5: Ein schwieriges Bidoku. Mehr als die Hälfte (131) aller Felder sind frei.

hen, dass wir bereits über entsprechende „Unterprogramme" verfügen, deren Realisierung wir dann später klären. Diese leisten das folgende:

Sei ein 16x16 Raster gegeben, in dem beliebig viele (also evtl. auch gar keine) Nullen und Einsen eingetragen sind, die die Bidoku-Regeln nicht verletzen. Wir sprechen dann von einem „Bidoku-Kandidat". Dann gehen wir davon aus, dass wir Unterprogramme besitzen, die die folgenden Fragen beantworten können:

- Ist dieser Bidoku-Kandidat lösbar?
- Ist dieser Bidoku-Kandidat eindeutig?
- Wie sieht eine Lösung für diesen Bidoku-Kandidat aus?

Außerdem werden wir eine Prozedur verwenden können, die vollständig ausgefüllte Bidoku-Kandidaten generiert.

Zunächst gehen wir auf das Erzeugen vollständig ausgefüllter Kandidaten kurz ein. Im Folgenden besprechen wir dann (u.a.) darauf aufbauende Generierungsverfahren.

Ein vollständig gelöstes Bidoku generieren

Schon in diesem Schritt sind verschiedene Varianten denkbar.

Geschicktes Setzen von Zahlen Die erste Idee greift dem nächsten Abschnitt (Generierung 1) vor. Dort erzeugen wir ein Bidoku, indem wir so lange Zahlen in ein leeres Raster einfügen, bis wir ein eindeutiges Bidoku erhalten. Dabei prüfen wir nach jedem Schritt

0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

Abbildung 6: Zwei zulässige und vollständig gelöste Bidokus

auf Eindeutigkeit des Bidokus. Wenn wir ein vollständig gelöstes Bidoku erhalten wollen, können wir diesen Test einfach sparen und solange Zahlen einfügen, bis kein leeres Feld übrig ist. Dafür benötigen wir nur ein Unterprogramm, was nach jedem Setzen testet, ob das Bidoku weiterhin lösbar ist.

Systematisches Erzeugen und Permutieren Es ist nicht so schwierig, sich ein gelöstes Bidoku zu überlegen, wenn man gezielt dafür sorgt, dass die Regeln eingehalten werden. Zum Beispiel stellt man fest, dass man ein zulässiges, vollständig gelöstes Bidoku erhält, wenn man das 16x16 Raster alternierend mit Nullen und Einsen auffüllt, d.h. wenn man z.B. zeilenweise durchläuft und immer abwechselnd eine Eins und eine Null einfügt. Dieses Bidoku und ein weiteres zulässiges und vollständig gelöstes Bidoku sieht man in Abbildung 6.

Hat man ein solches vollständig gelöstes Bidoku, so kann man daraus andere Bidokus erstellen. Das funktioniert so: Wenn man darauf achtet, dass man keine Regeln missachtet, darf man bestimmte Zahlenpaare *vertauschen*. Wir nehmen uns als Beispiel eine Null und eine Eins, die beide in der gleichen Spalte liegen. Die Null sei in Zeile i, die Eins in Zeile j. Wir vertauschen diese beiden Zahlen. Dann suchen wir uns ein weiteres Pärchen in einer anderen Spalte, das aus einer Null in Zeile j und einer Eins in Zeile i besteht. Außerdem verlangen wir, dass das neue Pärchen die gleichen Unterquadrate trifft wie unser erstes Pärchen, genauer: die zweite Null ist im gleichen Unterquadrat wie die erste Eins, und die zweite Eins liegt im gleichen Unterquadrat wie die erste Null. Jetzt vertauschen wir auch die Null und die Eins unseres zweiten Pärchens. Dann verletzt das neue Bidoku keine Regeln, denn:

- Das alte Bidoku verletzte keine Regel.

- An der Anzahl der Nullen und Einsen in den Spalten haben wir nichts verändert.
- Für zwei Zeilen haben wir die Anzahl der Einsen und Nullen durch das erste Vertauschen geändert, dieses haben wir aber durch das zweite Vertauschen rückgängig gemacht.
- Liegen alle Zahlen im gleichen Unterquadrat, so hat sich für die Unterquadrate nichts geändert. Lagen die erste Null und die erste Eins in verschiedenen Unterquadraten, so hat sich für diese beiden Unterquadrate die Anzahl der Nullen und Einsen verändert. Dieses haben wir aber durch das zweite Vertauschen rückgängig gemacht.

Auf diese Weise kann man weitere Permutationsregeln aufstellen. Um zu einer vernünftigen Lösung der Aufgabe zu gelangen, muss man zwei Probleme lösen:

- Für jede Permutationsregel muss sichergestellt sein, dass die Bidoku-Regeln im Anschluss nicht verletzt sind.
- Man muss sicher sein, dass man durch wiederholtes Anwenden der Permutationsregeln nach bestimmter Zeit jedes (oder fast jedes) Bidoku erzeugen kann, so dass die entstehenden Bidokus ausreichend interessant sind.

Generierung 1: „So viele Zahlen einfügen, bis das Bidoku eindeutig ist“

Hier beginnt man mit einem leeren Bidoku und versucht dann, geschickt Zahlen einzufügen. Zu Beginn hat man die freie Wahl, wo man setzt und was man setzt, doch mit jeder gesetzten Zahl schränkt man die Lösung des Bidokus ein. Irgendwann gelangt man zu einer Situation, in der es nur noch eine verbleibende Möglichkeit gibt, die restlichen Zahlen einzufügen (d.h. die Lösung des Bidokus ist eindeutig), also bricht man ab und lässt die restlichen Felder frei. Außerdem stößt man zwischendurch auf Felder, die man nur mit einem Wert belegen darf, da das Bidoku sonst unlösbar wird oder weil man direkt eine Bidoku-Regel verletzen würde. Diese kann man auch frei lassen, da ihre Belegung implizit feststeht.

Bei dieser Vorgehensweise muss man auf zwei Dinge achten: Zum einen darf man keine Situation erzeugen, in der das Bidoku nicht mehr lösbar ist. Nach dem Setzen einer Zahl sollte man also sichergehen, dass weiterhin eine Lösung existiert. Wir können es uns leicht machen und sagen, dass wir unser ,,Unterprogramm” aufrufen, das uns sagt, ob unser Bidoku weiterhin lösbar ist. Wenn nicht, dann hätten wir die letzte Zahl nicht setzen dürfen, also entfernen wir sie und versuchen unser Glück mit einer anderen Zahl oder an einer anderen Stelle. Außerdem muss man wissen, wann man genug Zahlen eingefügt hat, d.h. ab wann die eingefügten Zahlen die noch freien Felder eindeutig bestimmen. Hier benutzen wir unser anderes Unterprogramm, welches die Eindeutigkeit prüfen kann. Sobald wir die Rückmeldung bekommen, dass nur noch eine gültige Lösung existiert, können wir die Erzeugung stoppen, denn wir haben ein eindeutiges, lösbares Bidoku erzeugt.

Generierung 2: „Aus einem vollständig gelösten Bidoku Zahlen entfernen“

Alternativ kann man zunächst ein vollständig gelöstes Bidoku generieren, um dann eine gewünschte Anzahl von Zahlen wieder zu entfernen. *Vollständig gelöst* ist ein Bidoku, wenn keine Felder leer sind. In dieser Situation existiert offensichtlich eine Lösung, und auch durch das Entfernen von Zahlen wird sich daran nichts ändern. Wir erhalten also die Garantie, dass das durch das Löschen entstehende Bidoku in jedem Fall lösbar ist und kennen sogar die Lösung, denn diese besteht aus den gelöschten Zahlen.

Allerdings muss man darauf achten, dass die Lösung nach jedem Löschen auch eindeutig bleibt. Wir könnten nach jedem Löschen unser Unterprogramm aufrufen, das testet, ob weiterhin ein eindeutiges Bidoku vorliegt. Alternativ kann man aber auch nur mit dem Unterprogramm auskommen, dass auf Lösbarkeit testet: Nachdem wir eine Zahl gelöscht haben, setzen wir die Zahl ein, die dort nicht stand und testen dann, ob das entstandene Bidoku lösbar ist. Wenn dies der Fall ist, gibt es zwei Lösungen für das Bidoku ohne die Zahl, da das Bidoku auch mit der Zahl, die ursprünglich in dem Feld stand, lösbar war.

Generierung 3: Besonders geschicktes Setzen von Zahlen

Hier versucht man, ein Bidoku ,,greedy” zu erzeugen, d.h. man setzt wie bei der ersten Methode aus Abschnitt 2.3 nacheinander Zahlen und versucht, ein vollständiges Bidoku zu erzeugen. Man möchte aber kein Unterprogramm aufrufen, dass die Lösbarkeit überprüft. Stattdessen leitet man sich aus den Regeln für Bidokus her, wie man Felder füllen darf. Dabei sollte man die Felder möglichst in einer vorgegebenen Reihenfolge belegen. Geht man z.B. spaltenweise vor, so kann man die ersten zwei Spalten beliebig setzen, solange man die Bedingung einhält, dass jede Spalte acht Nullen und acht Einsen enthält. Für die dritte und vierte Spalte muss man dann noch beachten, dass man die Bedingungen für die ersten vier Blöcke einhält.

Für die weiteren Spalten werden die Nebenbedingungen für das randomisierte Setzen schwieriger, und evtl. kann man auch nicht mehr vollständig sicher sagen, dass man keinen Widerspruch erzeugt. Daher kann es passieren, dass man bei den letzten Spalten merkt, dass es keine gültige Belegung mehr gibt. In diesem Fall löscht man alle Felder und beginnt von vorn.

Eine Lösung, die auf einem solchen Ansatz beruht, muss auf zwei Dinge achten:

- Ein Fehlschlag darf ,,nicht zu oft” auftreten, so dass man in akzeptabler Zeit ein vollständiges Bidoku erhält.
- Wie in Abschnitt 2.3 (Methode 2) darf man keine zu langweiligen Bidokus erstellen, d.h. um zu erreichen, dass man nur lösbare Bidokus erstellt bzw. nicht zu viele Fehlschläge hat, darf man nicht zu rigorose Regeln verwenden. Man muss die Randomisierung so gestalten, dass es prinzipiell möglich ist, dass jedes (oder fast jedes) Bidoku entsteht.

2.4 Bidokus lösen und auf Eindeutigkeit prüfen

Im Kopf lösen: Ein Ansatz mit „oberen und unteren Schranken"

Oft scheint es beim Lösen eines Bidokus so zu sein, dass man entweder sehr viel weiß (z. B. die komplette Belegung einer Zeile) oder gar nichts (bzw. nicht genug, um auch nur eine Zahl eintragen zu können).

Der erste Fall liegt vor, wenn man eine Zeile, Spalte oder ein Quadrat findet, in dem bereits acht Nullen oder acht Einsen stehen. Man kann dann die leeren Felder mit der jeweils anderen Zahl füllen. Unsere einfachste Strategie ist es also, immer die Anzahlen der Zahlen in jeder Zeile, Spalte und in jedem Quadrat zu prüfen.

Sobald wir in eine Situation kommen, in der wir mit dieser einfachen Regel keine Zahlen mehr eintragen können, müssen wir uns etwas Komplizierteres ausdenken. Die Aufgabenstellung gibt uns ein Beispiel, wie man das anstellen kann. Dort werden die Spalten 9 bis 12 des vorgegebenen Bidokus angesehen. Die entsprechenden Unterquadrate sind links in Abbildung 7 abgebildet. Jetzt stellt man folgende Konstellation fest: Im unteren Unterquadrat sind bereits sieben Einsen, d.h. es kann nur noch eine Eins eingefügt werden. In der neunten Spalte (in der Abbildung der zweiten) gibt es bereits sechs Nullen, d.h. in dieser Spalte können nur noch maximal zwei Nullen eingefügt werden. Für den Schnitt der Spalte mit dem Unterquadrat bedeutet dies, dass nur zwei der drei leeren Felder mit Nullen aufgefüllt werden können – das dritte muss also mit der letzten verbleibenden Eins des unteren Unterquadrats aufgefüllt werden. Wir wissen zwar nicht, welches der drei Felder mit einer Eins gefüllt wird, aber wir wissen, dass die Eins auf jeden Fall im Schnitt von Spalte und Quadrat liegt. Daher folgt, dass der Rest des Quadrats mit Nullen gefüllt werden kann. Dies ist in der Abbildung als zweites dargestellt. Dabei stehen die neu hinzugefügten Nullen in Klammern, um sie von den vorgegebenen zu unterscheiden. Wenn man ein weiteres Mal überlegt, so stellt man außerdem fest, dass man nicht nur über das Unterquadrat, sondern auch über die Spalte etwas erfahren hat. Da in dem Schnitt maximal eine Eins stehen kann, müssen die beiden verbleibenden Nullen der Spalte ebenfalls im Schnitt stehen. Dadurch sind alle Nullen der Spalte verbraucht, folglich können wir die außerhalb des Schnittes liegenden freien Felder mit Einsen auffüllen. Dies sieht man in im dritten Teil von Abbildung 7.

Für unser erfolgreiches Vorgehen suchen wir jetzt eine Verallgemeinerung. Uns hat geholfen, dass wir den Schnitt von einer Spalte mit vielen Nullen und einem Unterquadrat mit vielen Einsen betrachtet haben. Genauer betrachtet kann man feststellen:

- Die Anzahl der fehlenden Nullen in der Spalte war kleiner als die Anzahl der fehlenden Zahlen im Schnitt von Unterquadrat und Spalte, also müssen die restlichen Felder des Schnittes Einsen enthalten.
- Dadurch haben wir aber alle noch fehlenden Einsen des Quadrats verteilt, also konnten wir den Rest des Quadrats (außerhalb des Schnitts) mit Nullen auffüllen.

0	-	0	-
-	1	0	1
-	0	-	1
-	0	1	-
-	-	0	0
-	-	0	-
-	1	0	-
0	0	0	-
0	0	-	-
1	0	-	-
1	-	-	-
0	0	1	-
1	-	-	-
1	-	-	1
-	1	1	1
0	-	1	0

0	-	0	-
-	1	0	1
-	0	-	1
-	0	1	-
-	-	0	0
-	-	0	-
-	1	0	-
0	0	0	-
0	0	-	-
1	0	-	-
1	-	-	-
0	0	1	-
1	-	(0)	(0)
1	-	(0)	1
(0)	1	1	1
0	-	1	0

0	(1)	0	-
-	1	0	1
-	0	-	1
-	0	1	-
-	(1)	0	0
-	(1)	0	-
-	1	0	-
0	0	0	-
0	0	-	-
1	0	-	-
1	(1)	-	-
0	0	1	-
1	-	(0)	(0)
1	-	(0)	1
(0)	1	1	1
0	-	1	0

Abbildung 7: Erste Schritte in der Lösung des Beispiels aus der Aufgabe

Sei „Einsen(Quadrat *i*)" die Anzahl der Einsen, die bereits in Unterquadrat *i* stehen (wir nummerieren die Unterquadrate spaltenweise von rechts nach links von 1 bis 16 durch)), sei „Nullen(Spalte *j*)" die Anzahl der Nullen in Spalte *j* und sei „Frei(Spalte *j*, Quadrat *i*)" die Anzahl der freien Felder im Schnitt von Quadrat *i* und Spalte *j*. Mit Hilfe dieser Abkürzungen kann man nun darstellen, welche Bedingungen in unserem Beispiel erfüllt sind. Wir formulieren dazu jeweils den Aussageteil der beiden Feststellungen um (das, was wir daraus schlussfolgern, nicht).

- 8 - Nullen(Spalte 9) $<$ Frei(Spalte 9, Quadrat 12)
- 8 - Einsen(Quadrat 12) $\leq$ Frei(Spalte 9, Quadrat 12) - (8 - Nullen(Spalte 9))

Die obere Aussage ist relativ direkt übernommen (da es maximal 8 Nullen geben darf, gibt der linke Teil die noch fehlenden Nullen an). Die zweite Feststellung besagt, dass im Quadrat weniger Einsen fehlen als im Schnitt von Quadrat und Spalte aufgrund der Belegung der Spalte eingefügt werden müssen. Das entspricht der Aussage, dass wir alle fehlenden Einsen „verteilt" haben. Da diese beiden Bedingungen gelten, können wir im Beispiel den Rest von Quadrat mit Nullen auffüllen. Unsere Überlegungen hingen nicht davon ab, mit welcher Spalte und welchem Quadrat wir genau überlegt haben, d.h. wir können die gleichen Schlussfolgerungen *immer* anstellen, wenn die beiden Bedingungen erfüllt sind.

Damit haben wir im Prinzip (endlich) eine relativ allgemeine Regel gefunden, mit deren Hilfe man zusätzliche Felder auffüllen kann. Wir wollen noch ein paar zusätzliche Merkmale festhalten, um den Nutzen der Regel zu erhöhen. Zum einen stellen wir fest, dass das Überprüfen der oberen Bedingung überflüssig ist. Da (8 - Einsen(Quadrat 12)) immer positiv ist, kann die untere Bedingung nicht gelten, wenn die obere nicht gilt. Als zweites erinnern wir uns,

0	1	1	(1)	(1)	0	(0)	1	0	(1)	0	(0)	0	(1)	1	(0)
(1)	0	0	(0)	(0)	0	1	1	(1)	1	0	1	(1)	(0)	0	1
(1)	0	0	0	(1)	0	1	(0)	(1)	0	(1)	1	0	1	0	(1)
1	1	(0)	1	1	(1)	(0)	(0)	(0)	0	1	(0)	0)	(0)	(1)	1
1	(0)	(1)	1	0	0	1	(1)	(0)	(1)	0	0	(1)	0	1	0
(0)	1	0	(0)	0	0	(1)	0	(1)	(1)	0	(1)	1	0	(1)	(1)
(0)	0	1	1	0	(1)	(0)	(1)	(1)	1	0	(1)	1	(0)	0	(0)
1	(0)	(1)	(0)	(1)	1	(1)	0	0	0	0	(1)	1	0	(1)	(0)
(1)	(1)	0	(1)	0	0	(0)	1	0	0	(1)	(1)	0	(1)	0	(1)
0	1	(1)	(1)	0	0	(0)	(1)	1	0	(1)	(0)	(1)	0	0	(1)
1	(1)	0	(0)	1	(0)	1	0	1	(1)	(1)	(0)	(0)	1	0	0
0	0	0	0	(1)	1	1	(1)	0	0	1	(0)	(1)	(1)	(1)	(0)
0	(0)	(1)	(0)	1	(1)	(1)	(1)	1	(1)	(0)	(0)	(1)	(0)	0	(0)
(0)	0	1	1	(1)	(1)	0	(0)	1	(0)	(0)	1	(0)	1	(0)	1
0	1	(0)	1	0	(1)	(0)	(0)	(0)	1	1	1	0	1	(1)	(0)
(1)	1	(1)	(0)	0	(1)	(0)	(0)	0	(0)	1	0	(0)	1	(1)	1

Abbildung 8: Die vollständige Lösung des Bidokus aus der Aufgabenstellung.

dass wir im Beispiel auch direkt die Spalte füllen konnten. Durchläuft man alle Überlegungen noch einmal mit dem vertauschten Argument für Quadrat und Spalte, so erhält man folgende analoge Bedingungen (von denen ebenfalls die obere überflüssig ist):

- 8 - Einsen(Quadrat 12) $<$ Frei(Spalte 9, Quadrat 12)
- 8 - Nullen(Spalte 9) $\leq$ Frei(Spalte 9, Quadrat 12) - (8 - Einsen(Quadrat 12))

Die zweite Bedingung ist aber eigentlich nichts anderes als die zweite Bedingung oben, denn beide kann man umformen zu

- (8 - Einsen(Quadrat 12)) + (8 - Nullen(Spalte 9)) $\leq$ Frei(Spalte 9, Quadrat 12)

Das bedeutet: Wann immer diese eine Bedingung erfüllt ist, kann man das entsprechende Quadrat außerhalb des Schnitts mit Nullen und die entsprechende Spalte außerhalb des Schnittes mit Einsen auffüllen.

Glücklicherweise funktioniert dieser Trick nicht nur für jede Kombination von Spalten mit Quadraten, sondern auch für jede Kombination von Zeilen mit Quadraten. Außerdem können sich die Rollen von Einsen und Nullen vertauschen. Damit haben wir eine sehr nette und allgemeine Regel gefunden.[2]

Das Beispiel aus der Aufgabe lässt sich allein durch Anwendung der beiden beschriebenen Regeln lösen. Dabei benötigt man die komplizierte Regel sogar nur dreimal. Die vollständige Lösung des Aufgabenbeispiels ist in Abbildung 8 zu sehen.

[2]Es sei noch angemerkt, dass das $\leq$ auch in ein Gleichheitszeichen verwandelt werden kann. Der Term auf der linken Seite kann niemals kleiner werden als der auf der rechten Seite.

Nach dieser langen Herleitung wollen wir noch kurz eine dritte Möglichkeit erwähnen, die dem Lösenden auch auf dem Papier zur Verfügung steht: die *Hypothese*. Das bedeutet nur, dass man für ein Feld, dessen richtige Belegung man nicht kennt, eine Belegung rät und dann ausprobiert, ob sich dadurch Konsequenzen ergeben, die einen Widerspruch zu den Bidoku-Regeln darstellen. Wenn man solche Widersprüche recht schnell finden kann (das Bidoku erst fast komplett lösen und dann merken, dass es doch nicht passt, ist eher nicht hilfreich), dann kann man die entsprechende Belegung für das Feld ausschließen und weiß, dass die andere Zahl die richtige ist.

Am Computer lösen: Ein rekursiver Ansatz mit Tiefensuche

Bidokus sind ziemlich groß – zumindest, wenn man versuchen möchte, ein Bidoku nur durch „Ausprobieren" zu lösen. Wenn ein vorgegebenes Bidoku genau zur Hälfte gefüllt ist, d.h. 128 Felder sind noch leer, dann gibt es genau $2^{128} \approx 3,4 \cdot 10^{38}$ Möglichkeiten, die leeren Felder mit Nullen und Einsen zu füllen. Sind noch weniger Felder gefüllt, so gibt es noch mehr Möglichkeiten. Umso erstaunlicher ist es, dass man auch mit einem Ansatz, der auf Ausprobieren basiert, durchaus gute Erfolge erzielen kann.

Zur Lösung des Bidokus durchläuft man die noch freien Felder in irgendeiner Reihenfolge. Für jedes Feld testet man, welche Belegungen aufgrund der Bidoku-Regeln möglich sind (d.h. man überprüft, ob in die Spalte, die Zeile und das Quadrat jeweils noch eine Null und eine Eins hineinpassen). Ist eine mögliche Belegung ausgeschlossen, wählt man die andere. Ansonsten sucht man sich eine der beiden Möglichkeiten beliebig aus und ruft sich anschließend rekursiv auf. Bringt der rekursive Aufruf ein positives Ergebnis, d.h. ist das Bidoku weiterhin lösbar, so hat man eine gültige Belegung gewählt, d.h. man bleibt bei der Belegung. Andernfalls weiß man, dass man einfach die andere Belegung einsetzen kann (zumindest, wenn man vorher immer sichergestellt hat, dass man ein lösbares Bidoku hat). Die Rekursion bricht in zwei Fällen ab:

- Man hat es geschafft, alle leeren Felder zu füllen, d.h. das Bidoku war lösbar und man hat eine Lösung gefunden.
- Für ein Feld würden beide Belegungen Bidoku-Regeln verletzen und sind deshalb unmöglich. In diesem Fall hat man auf einer höheren Rekursionsebene einen „Fehler" gemacht oder das Bidoku war nicht lösbar.

Im ersten Fall kann man direkt das ganze Programm beenden, im anderen Fall kehrt man nur zur nächsthöheren Rekursionsebene zurück.

Trotz der großen Anzahl von möglichen Belegungen aller 256 Felder kann man mit dieser einfachen rekursiven Methode oft sehr schnell testen, ob ein Bidoku lösbar ist, und eine Lösung ausgeben. Das liegt daran, dass Bidokus sich nur sehr schwer unlösbar machen lassen; um eine Situation zu erzeugen, in der keine Lösung mehr existiert, müssen daher meistens bereits so viele Zahlen eingefügt sein, dass die verbleibende Rekursionstiefe sehr niedrig ist. Meistens muss man also nur wenige Ebenen zurückgehen, um einen „Fehler" zu korrigieren.

Allerdings sollte man sich davor hüten, die Felder in einer Reihenfolge zu belegen, die viel im Raster herumspringt, da das Setzen von Feldern an vielen verschiedenen Orten schneller unlösbare Bidokus erzeugt. Wenn man diese rekursive Lösungsvariante zusammen mit dem in Abschnitt 2.3 (Generierung 1) beschriebenen Verfahren benutzt, so sollte man beim Erzeugen des Bidokus ebenfalls darauf achten, dass man keine zu verteilten Felder nacheinander belegt, sondern lieber spalten- oder zeilenweise durchlaufen. Man kann auf diese Weise tatsächlich nette, eindeutig lösbare Bidokus erzeugen, denen allerdings in der Regel anzusehen ist, in welcher Ecke die Erzeugung begonnen hat (denn dort werden mehr Felder gesetzt).

Um die Tiefensuche zu beschleunigen, kann man nach dem Setzen von Feldern versuchen, die Folgen, die das Setzen hatte, direkt zu berechnen. Dazu kann man die beiden Regeln verwenden, die im letzten Abschnitt beschrieben worden sind, wobei die zweite Regel einen größeren Zusatzaufwand mit sich bringt. Hat man die Regeln jedoch beide implementiert, so ist die Tiefensuche eigentlich nichts anderes als die Realisierung eines menschlichen Spielers, der immer fleißig Regeln anwendet und auf eine Hypothese zurückgreift, sobald er nichts Neues mehr schließen kann.

Bidokus auf Eindeutigkeit prüfen

Hier können wir es uns leicht machen, denn unsere Tiefensuche, egal ob um zusätzliche Regeln erweitert oder nicht, kann bei geeigneter Anpassung leicht überprüfen, ob zwei Lösungen existieren. Dazu muss man nur zählen, wie oft das erste der beiden Abbruchkriterien erreicht wurde (und erst beim zweiten Mal die Suche abbrechen). Da unser menschlicher Spieler im Prinzip nichts anderes ist als eine Tiefensuche mit zusätzlichen Regeln, kann man auch diesen Ansatz direkt so umwandeln, dass man Bidokus auch auf Eindeutigkeit prüfen kann.

2.5 Erweiterungen

Muster und Symmetrien

Die Realisierung dieser Erweiterung kann beliebig kompliziert aussehen. Wenn man sein Bidoku rekursiv erzeugt und dabei vorwärts vorgeht (d.h. man setzt solange Ziffern, bis man ein eindeutig lösbares Bidoku erhält), dann kann man Muster oder Symmetrien erzeugen, indem man die Reihenfolge beeinflusst, in der man die Felder belegt. Dadurch beeinflusst man die Form der vorbelegten Felder. Man stößt allerdings auf zwei Probleme: Zum einen weiß man nicht vorher, wann der Prozess abbricht, und muss damit rechnen, dass die gewünschte Form hinterher schlecht zu erkennen ist, weil zu viele Felder besetzt werden mussten, um das Bidoku eindeutig zu machen. (Wenn man lieber mehr Felder besetzten möchte, kann man dies problemlos tun, da mehr besetzte Felder an der eindeutigen Lösbarkeit nichts ändern.) Zum anderen haben Belegungsreihenfolgen, die sehr stark im Raster herumspringen, den Nachteil, dass schnell unlösbare Bidokus entstehen, deren Unlösbarkeit man aber nur schwer erkennen

0	0	1	1	1	1	0	1	1	1	0	1	0	0	0	0
0	1	0	1	1	0	0	0	1	1	1	1	1	0	0	0
1	-	-	1	0	0	0	1	0	0	0	1	1	-	-	1
0	-	-	1	1	1	0	1	0	0	0	0	0	-	-	1
1	1	-	-	1	1	1	0	0	0	0	0	-	-	1	0
1	1	-	-	1	1	0	1	0	0	0	0	-	0	0	1
0	1	-	-	1	1	0	-	-	1	1	1	-	-	0	0
0	1	1	-	-	0	0	-	-	1	1	-	-	1	0	1
1	0	1	-	-	1	1	-	-	1	0	-	-	0	0	0
1	0	1	-	-	0	1	-	-	1	0	-	-	0	0	0
1	1	0	-	-	0	0	-	-	1	0	-	1	1	1	1
0	0	0	0	-	-	-	0	1	-	-	0	1	1	1	1
0	1	1	1	0	0	1	0	0	1	1	0	0	0	1	1
0	0	1	0	1	0	1	0	0	0	1	0	1	1	1	1
1	0	1	0	0	1	1	0	0	0	1	1	0	1	1	0
1	0	1	0	0	1	1	1	1	0	1	1	0	0	0	0

Abbildung 9: Ein „W“.

kann (bzw. nur durch das Durchprobieren vieler Möglichkeiten). Ein rein rekursiver Algorithmus ist für komplizierte Belegungsreihenfolgen daher oft zu langsam.

Erzeugt man sein Bidoku, indem man aus einem vollständig gelösten Bidoku Zahlen löscht, so kann man durch die Reihenfolge der Felder, deren Inhalt man zu löschen versucht, eine gewünschte Form annäherungsweise erreichen. Es kann passieren, dass sehr viele Felder ,,übersprungen” werden, weil das Löschen des Inhalts hier die Eindeutigkeit zerstören würde. Dadurch kommt man an das gewünschte Ergebnis evtl. nicht nahe genug heran. Ab einer gewissen Anzahl von zu löschenden Feldern dauert es bei einem rein rekursiven Algorithmus außerdem recht lange, zu überprüfen, ob die Eindeutigkeit verloren geht. Abbildung 9 zeigt ein Beispiel, in dem ein Muster durch das Löschen relativ weniger Zahlen erzeugt wurde. In Abbildung 10 wurden sehr viele Zahlen gelöscht, dafür handelt es sich nicht um eine besonders spannende Symmetrie.

Wenn man sein Programm so umbauen kann, dass es auch kleinere Raster mit angepassten Regeln füllen kann, so kann man sich symmetrische Bidokus aus kleineren zusammensetzen. Beispiele hierfür kann man in Abbildung 11 und in Abbildung 12 sehen. Ein Spezialfall des Zusammensetzens ist das Spiegeln: Hier erstellt man einfach einen Teil des Bidokus (mit zusätzlichen Regeln) und spiegelt diesen dann, um das komplette Bidoku zu erhalten. Leider geht durch das Zusammensetzen im Allgemeinen die Eindeutigkeit verloren: Auch wenn die Lösung für die kleinen Raster eindeutig ist, wenn man für diese die zusätzlichen Regeln anwendet, kann das gesamte Bidoku noch weitere Lösungen enthalten, die sich um die zusätzlichen Regeln nicht kümmern. Dies ist auch in den beiden Beispielen der Fall.

Es lassen sich noch viele weitere Möglichkeiten finden, symmetrische oder andere ,,ästhetisch schöne” Bidokus zu generieren. Die hier vorgestellten Möglichkeiten waren eher natürliche

-	-	-	-	-	-	-	-	1	0	1	1	1	0	0	0
-	-	-	-	-	-	-	1	1	1	1	0	1	0	1	1
-	-	-	1	1	0	-	-	0	1	1	0	1	0	0	1
-	-	-	-	-	-	0	-	0	0	0	0	1	1	0	0
-	-	1	-	-	0	-	-	0	0	1	1	1	1	1	0
-	-	-	-	-	-	1	1	1	1	1	1	1	1	0	0
-	0	-	-	-	-	0	-	0	1	1	0	1	0	0	0
-	-	-	-	0	-	-	-	0	0	0	0	0	1	1	0
-	0	-	-	-	0	-	-	0	1	0	1	1	0	0	0
-	-	-	-	-	-	-	1	1	0	0	1	0	1	1	1
-	-	1	1	-	0	0	-	1	0	1	1	0	0	1	1
-	-	-	-	0	0	0	-	0	1	0	0	0	0	1	1
-	-	1	-	-	0	-	-	1	1	0	0	0	1	1	0
-	-	-	-	0	-	0	-	0	0	1	0	0	0	0	1
-	-	-	-	-	-	0	1	1	1	0	1	0	1	0	1
-	-	-	-	0	0	0	-	1	0	0	1	0	1	1	1

Abbildung 10: Hier wurde versucht, möglichst viele Zahlen auf der linken Seite zu löschen.

-	-	-	0	0	-	-	-	-	-	-	1	1	-	-	-
-	-	1	0	1	1	-	-	-	-	0	1	1	0	-	-
-	1	0	1	1	0	1	-	-	0	1	0	0	1	1	-
0	1	0	1	0	1	0	1	1	0	1	0	0	1	1	0
0	1	1	0	1	0	0	0	1	1	0	1	0	1	0	1
-	0	0	1	0	1	1	-	-	1	0	1	1	0	1	-
-	-	0	0	1	1	-	-	-	-	0	0	0	0	-	-
-	-	-	1	1	-	-	-	1	-	-	1	1	-	-	-
-	-	-	0	0	-	1	1	-	-	-	1	0	-	-	-
-	-	1	1	0	1	-	-	-	0	1	0	0	-	-	-
-	0	0	1	0	0	0	-	-	1	1	0	1	0	0	-
0	1	1	0	1	0	0	1	1	0	1	0	1	0	0	1
0	1	0	1	0	1	0	1	1	0	1	0	0	1	1	0
-	1	0	1	1	0	1	-	-	0	1	0	0	1	1	-
-	-	1	0	1	1	-	-	-	-	0	1	1	0	-	-
-	-	-	0	0	-	-	-	-	-	-	1	1	-	-	-

Abbildung 11: Dieses Bidoku wurde aus zwölf 4x4 Rastern und einem 8x8 Raster (in der Mitte) zusammengesetzt. Sieht man sich die vollständig gefüllten Zeilen an, sieht man, welche Regeln für die kleineren Raster verwendet wurden.

1	0	0	1	1	1	0	0	0	1	1	0	1	0	0	1
0	1	1	-	-	-	-	0	1	-	-	-	-	0	0	1
0	0	0	1	-	-	-	0	1	-	-	-	0	1	1	1
0	-	1	1	1	-	-	0	1	-	-	0	0	0	-	1
1	-	-	0	1	0	-	1	0	-	1	1	1	-	-	0
1	-	-	-	0	0	1	1	0	0	0	1	-	-	-	0
0	-	-	-	-	0	1	1	0	0	0	-	-	-	-	0
1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	0
0	1	1	0	1	0	1	0	0	0	0	0	1	1	1	1
1	-	-	-	-	0	0	0	1	1	1	-	-	-	-	1
0	-	-	-	1	0	1	1	1	1	1	0	-	-	-	1
0	-	-	1	1	0	-	1	1	-	0	0	0	-	-	1
1	-	0	0	0	-	-	1	0	-	-	1	1	1	-	0
0	1	0	0	-	-	-	1	0	-	-	-	1	0	0	0
1	1	1	-	-	-	-	0	0	-	-	-	-	1	1	0
1	1	0	1	0	1	0	0	1	0	0	1	0	1	1	0

Abbildung 12: Ein Bidoku, das aus vier 8x8 Rastern zusammengesetzt wurde.

Erweiterungen der bereits vorhandenen Algorithmen, aber man kann sich sicher auch neue Generierungsalgorithmen überlegen, die gezielt symmetrische Bidokus erzeugen und dabei auch berücksichtigen, dass evtl. nicht jedes Feld frei gelassen werden kann (und für dieses Problem eine passende Lösung finden).

Generell muss sich eine solche Erweiterung an den folgenden Kriterien messen lassen:

- Bleibt die Eindeutigkeit des Bidokus erhalten?
- Sind Bidokus mit verschiedenen Schwierigkeitsgraden weiterhin erzeugbar?
- Wie gut kann man das Muster / die Symmetrie / die gewünschte Form erkennen?
- Handelt es sich eher um einen Zusatz zur bisherigen Lösung oder um ein Verfahren, dass auf zusätzlichen Erkenntnissen beruht?

2.6 Bewertungskriterien

- Schwächen der verwendeten Verfahren werden offensichtlich, wenn alle von der Einsendung generierten Bidokus starke Ähnlichkeiten aufweisen und somit nicht interessant zu lösen sind. Dies ist z.B. der Fall, wenn die Bidokus grundsätzlich links gefüllt und rechts leer sind oder eine immer gleiche Form mit verschiedenen Zahlen gefüllt wird. Außerdem müssen die generierten Bidokus lösbar sein, und das eindeutig.
- Die Aufgabenstellung verlangt eine wählbare Schwierigkeit des generierten Bidokus. Es müssen also Bidokus von unterschiedlicher Schwierigkeit erzeugt werden können.

Für besonders durchdachte Kriterien für den Schwierigkeitsgrad eines Bidokus gibt es hier Extrapunkte. Zum Beispiel kann nicht nur die Anzahl der fehlenden Zahlen berücksichtigt werden, sondern auch die Komplexität der Überlegungen, die man benötigt, um die freien Felder zu füllen.

- Entscheidend für die Effizienz des gesamten Verfahrens ist die Effizienz des verwendeten Algorithmus zum Lösen von Bidokus. Ohne logische Strategien wie in Abschnitt 2.4 beschrieben ist diese nicht zu erzielen. Ein reines Backtracking / Tiefensuche ist nicht intelligent genug. Generell sollte der Algorithmus in der Lage sein, ein Bidoku in wenigen Minuten zu erzeugen.
- Entscheidend für die Bewertung der Lösung ist, ob die generierten Bidokus der zentralen Anforderung der eindeutigen Lösbarkeit entsprechen. Da dies mit Beispielen allein nur schwer zu demonstrieren ist, sollte besonders begründet sein, warum das gewählte Verfahren diese Anforderung erfüllt. Gute Beispiele sind natürlich dennoch wichtig, einige sollten idealerweise von Hand zu lösen sein. Eine interessante theoretische Frage ist, wie viele Felder in einem Bidoku gegebener Größe mindestens belegt sein müssen, damit es (eindeutig) lösbar ist.
- Die Erzeugung symmetrischer und auf andere Weise ästhetisch schöner Bidokus wird als Erweiterung gewertet. Besonders hoch zu bewerten sind perfekt symmetrische Bidokus, die nicht aus Spiegelungen hervorgegangen sind. Bidokus, in denen sogar die eingetragenen Zahlen oder die Lösung symmetrisch ist, sind schwierig zu generieren und verdienen besondere Anerkennung.

Aufgabe 3: Folgenreiche Spielereien

Diese Aufgabe war sehr offen formuliert und erfordert daher zunächst eine präzisere Festlegung auf die umzusetzende Funktionalität und auf die zu erfüllenden (eigenen) Anforderungen. Das Kernthema der Aufgabe an sich sollte hingegen klar sein: die automatische Generierung von Folgen, nach denen dann anhand von Startgliedern gesucht werden kann.

3.1 Anforderungen und Beschränkungen

Auch wenn im Aufgabentext die Verbindung zur Online-Enzyklopädie der Zahlenfolgen hergestellt wird, ist dies mehr als Demonstration des absolut Möglichen denn als Vorbild für eine Lösung zu verstehen – schließlich soll unser Folgenlexikon vollständig automatisch erstellt werden und muss auf menschliche Vervollständigung und Pflege verzichten. Außerdem soll sich eine Lösung hier auf das Erzeugen von Folgen ausschließlich mithilfe von Rekursionsformeln bzw. Kombination von Elementarfolgen durch Operatoren beschränken. Algorithmische Beschreibungen für Folgen (wie sie etwa für die Folge der Primzahlen benötigt würden) sind demnach außen vor. Es sollte also erkannt werden, dass nur ein sehr beschränkter Teil aller möglichen Folgen betrachtet werden kann.

3.2 Folgen generieren

Wichtig und ausschlaggebend für den weiteren Lösungsweg ist die Frage: Welche Typen von Folgen können und sollen zu erzeugen und damit zu finden sein?

Rekursionsfolgen

Eine Folge $\langle a_n \rangle$ nennt man dann rekursiv definiert, wenn sich bei k gegebenen Anfangswerten der Wert eines Folgegliedes a_i aus den k voran stehenden Folgegliedern $a_{i-k}, \ldots, a_{i-1}$ berechnet.

Aber: was bedeutet „berechnen“, bzw. welche Möglichkeiten soll die Lösung hier vorsehen? An dieser Stelle müssen sinnvolle Einschränkungen getroffen werden. Zum einen können ausschließlich solche Rekursionsvorschriften verwendet werden, bei denen $k \leq 2$ ist, sich die Berechnung also nur aus den maximal zwei letzten Folgegliedern ergibt, wie zum Beispiel bei der Fibonacci-Folge. Zum anderen können die Operatoren eingeschränkt werden, die auf diese letzten Glieder anzuwenden sind. Sie werden im Folgenden durch $\circ$ symbolisiert. Beispiele seien die Grundrechenarten oder Potenzen. Kombiniert mit Konstanten K_x sowie der Position des aktuellen Folgegliedes n ergeben sich daraus verschiedenste Schemata, die zur

Generierung von Folgen verwendet werden können. Ein einfaches allgemeines Schema wäre beispielsweise:

$$a_n = (a_{n-1} \circ K_1) \circ (a_{n-2} \circ K_2) \circ K_3 \circ n \text{ mit Startwerten } a_0 = K_4,\ a_1 = K_5.$$

Das Schema wie hier im Voraus festzulegen erleichtert die spätere Implementierung wesentlich, da nur noch über die zu belegenden Stellen für Konstanten und Operatoren iteriert werden muss, stellt aber auf der anderen Seite eine Beschränkung dar, die erkannt und gerechtfertigt werden sollte.

Eine umfangreichere Lösung besteht darin, die Rekursionsformeln als Wörter einer formalen Sprache aufzufassen und systematisch aus Syntaxregeln (Ableitungsregeln) für diese Sprache aufzubauen: Ausgehend von

$$a_n = < TERM >$$

sowie Ableitungsregeln

<TERM>	:=	(<TERM> <OP> <TERM>) | <KONST> | <SEQ_MEM>
<OP>	:=	+ | - | · | ...
<SEQ_MEM>	:=	n | a_0 | ... | a_{n-k}
<KONST>	:=	1 | 2 | ...

lassen sich beliebige Folgen-Formeln erzeugen. Die Klammerung der Ausdrücke ergibt sich dabei aus der Reihenfolge der Ableitungen. Die Tiefe möglicher Ableitungen kann entweder grundsätzlich begrenzt werden, oder aber man lässt sie inkrementell wachsen (bis zu einem Benutzerabbruch).

Die Berechnung dieser Formeln lässt sich nun relativ einfach realisieren: Fasst man die Ableitung einer Formel als Baum auf:

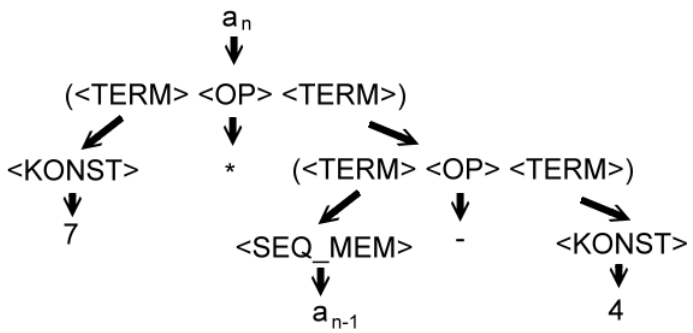

Abbildung 13: Beispielhafte Ableitung von $a_n = 7 \cdot (a_{n-1} - 4)$

so kann der Wert eines Folgengliedes durch rekursives Berechnen der Äste ermittelt werden: Konstanten und vorangehende Glieder sind bekannt, und anhand der Operatoren wird der Wert an einer „Verzweigung“ (<TERM> <OP> <TERM>) durch Anwendung auf die Äste der Operanden festgestellt. Der Wert eines Folgegliedes a_n ergibt sich also durch rekursive Berechnung aller Knoten des Ableitungsbaumes.

Solche Ableitungsbäume sind auch hilfreich, um die Erzeugung strukturell identischer Formeln zu vermeiden: $a_n = 7 \cdot (a_{n-1} - 4)$ und $a_n = (a_{n-1} - 4) \cdot 7$ haben den gleichen Ableitungsbaum und können als identisch aufgefasst werden.

Elementarfolgen

Beispiele Mit elementaren Folgen, oder kurz Elementarfolgen, ist nichts anderes gemeint als grundlegende und daher recht einfach zu beschreibende Folgen, die benutzt werden, um sich daraus komplexere Folgen durch Kombination und Anwendung von Operatoren „zusammenzubauen“. Der genaue Aufbau einer Elementarfolge ist also wiederum selbst zu definieren. Einige sinnvolle Beispiele seien hier angegeben:

- konstante Folgen ($a_n = x$)
- Fakultätsfolge ($a_n = a_{n-1} \cdot n, \;\; a_1 = 1$)
- Folge der Präfixsummen ($a_n = a_{n-1} + n, \;\; a_0 = 0$)
- alternierende Folge ($a_n = (-1)^n$)
- Folge der positiven (geraden/ungeraden) natürlichen Zahlen ($a_n = n$, $a_n = 2n$, $a_n = 2n+1$)

Wie wir sehen sind dies also einfache Beispiele von Rekursionsformeln selbst. Weitere elementare Folgen wären beispielsweise auch die der Quadratzahlen, jedoch lassen sich diese bereits recht einfach durch Kombination zusammensetzen.

Kombination von Elementarfolgen Die gewählten Elementarfolgen werden nun durch Operatoren miteinander kombiniert. Auch wenn das nicht durch die Aufgabe vorgegeben ist, kann dies grundsätzlich als gliedweise Operation interpretiert werden. Die Kombination wird nun erreicht, indem ausgehend von einer Folge jeweils durch Anwendung eines Operators und einer weiteren Folge eine neue Folge erzeugt wird, die ihrerseits dann weiter kombiniert werden kann:

$$\langle b \rangle = \langle a \rangle \circ \langle c \rangle$$

Abermals ergibt sich ein Ableitungsbaum. Dabei ist wieder darauf zu achten, dass nicht ein und dieselbe Folge auf mehreren Wegen kombiniert wird. Wie schon bei den Rekursionsformeln können beliebige Kombinationen als Ableitungen dieses allgemeinen Bildungsgesetzes aufgefasst und implementiert werden.

Explizite Folgen

Letztlich kann eine Folge auch durch eine nicht-rekursive Berechnungsformel für jedes Folgeglied angegeben werden. Beispiel: $a_n = n^2$. In der Aufgabenstellung ist dieses Formel-Prinzip nicht erwähnt, kann aber zusätzlich behandelt werden. Die Vorgehensweise bei der Aufstellung solcher Formeln ist ähnlich wie bei den anderen Formel-Prinzipien, bei den Ableitungsregeln für rekursive Formeln oben ist nur bei der Regel für <SEQ_MEM> auf die rekursiven Elemente zu verzichten.

3.3 Folgen speichern

Grundsätzlich wäre denkbar, erst bei einer Suchanfrage mit der Generierung von Folgenformeln nach den obigen Mustern zu beginnen und jede generierte Formel daraufhin zu testen, ob sie zu den eingegebenen Anfangsgliedern passt. Das führt jedoch zu immer wieder gleichen Berechnungen. Es ist also sinnvoll, erst einmal eine Reihe von Formeln zu generieren und die sich daraus ergebenden Folgen geeignet abzuspeichern, so dass eine effiziente Suche ermöglicht wird.

Da es unendlich viele Folgen gibt, muss diese Aufbauphase beschränkt werden, z.B. durch die Länge oder syntaktische Komplexität der generierten Folgen. Vermieden werden sollte nach Möglichkeit, dass beliebig viele Formeln gespeichert werden, die alle die gleiche Folge ergeben. Am Beispiel von Elementarfolgen: $P(\mathbf{2})$, $\mathbf{2} * P(\mathbf{1})$, $P(\mathbf{1}) + P(\mathbf{1})$, $\mathbf{3} * P(\mathbf{1}) - P(\mathbf{1})$ ergeben alle die gleiche Folge. In diesem Beispiel ist es naheliegend, nur die einfachste Formel $P(\mathbf{2})$ zu speichern bzw. die Generierung der anderen Formeln von vornherein zu vermeiden. Diese Frage nach einer „einfachsten“ Formel für eine Folge ist aber nicht einfach zu entscheiden. Außerdem ist es interessant, mehrere Formeln für die gleiche Folge ausgeben zu können, wenn sie genügend unterschiedlich sind.

Im Hinblick auf die Suche sollten die Formeln mit einer Reihe von Anfangsgliedern verbunden sein. Hier bietet sich an, die Anfangsglieder in einen Suchbaum zu organisieren.

3.4 Folgen suchen

Eine Anfrage besteht aus einer Reihe von Folgengliedern. Entlang dieser Reihe wird der Suchbaum durchlaufen, und die am erreichten Knoten abgespeicherten Formeln können ausgegeben werden. Es kann natürlich der Fall sein, dass für die Anfrage-Reihe noch keine genau passende Formel existiert. Dieser Fall sollte nicht einfach mit einer negativen Antwort erledigt werden. Mehrere mögliche Formeln könnten in geeigneter Sortierung ausgegeben werden. Falls sich mit wenigen weiteren Anfangsgliedern entscheiden lässt, welche Folge gemeint ist, kann der Benutzer zur Eingabe weiterer Glieder aufgefordert werden. Falls es gar nichts Passendes gibt, kann das System weitere Formeln generieren, in das Lexikon einbauen und direkt

mit der Anfrage-Reihe abgleichen; der Benutzer sollte aber kontrollieren können, wie lange diese Generierungsphase dauert. Hier wird die Idee eines Online-Algorithmus verfolgt.

Möglicherweise sind die eingegebenen Glieder nicht die Anfangsglieder einer gespeicherten Folge, werden aber doch durch die Folgen-Formel beschrieben. Im einfachsten Fall sind Anfrage-Reihe und gespeicherte Folge nur um eine Position versetzt; z.B. kann man die Fibonacci-Folge sowohl mit $0, 1, 1, 2, 3$ als auch mit $1, 1, 2, 3, 5$ beginnen lassen. Eine solche leichte Verschiebung kann abgefangen werden. Als Erweiterung ist es zu sehen, wenn eine freiere Suche in der Folge möglich ist; bei rekursiven Formeln kann eine freie Suche so realisiert werden, dass für jede Formel geprüft wird, ob die eingegebene Zahlenreihe ihr genügt.

3.5 Folgen ausgeben

Die Ausgabe der gefundenen Folgen sollte mindestens die korrekte mathematische Formel zu Berechnung beliebiger Folgeglieder enthalten, sowie ggf. die benutzten Startwerte einer Rekursion. Neben Korrektheit der Formel ist auch auf Sinnhaftigkeit der Formeldarstellung zu achten: Produkte mit Nullfaktoren, die eventuell bei der Verwendung eines festen allgemeinen Formelschemas auftreten können, sollten beispielsweise grundsätzlich ebenso wenig in der späteren Formel auftauchen wie unnötige Klammerungen. Zusätzlich sollte die Antwort auf eine Formelsuche noch einige weitere berechnete Folgenglieder enthalten.

Bei Folgen, die aus Elementarfolgen kombiniert wurden, bietet es sich außerdem an, eben auf diese Zusammensetzung hinzuweisen. Mit Textbausteinen lassen sich so Antworten wie diese generieren:

Eine weitere gefundene Folge, die zu diesen Folgenwerten führt ist $\langle a \rangle$ = $\langle b \rangle$ + $\langle c \rangle$ - $\langle d \rangle$, wobei $\langle b \rangle$ die Folge der ganzen Zahlen, $\langle c \rangle$ die konstante Folge 7 und $\langle d \rangle$ die Folge der Präfixsummen ist.

Offensichtlich wurden hier die Blätter des Ableitungsbaumes aneinander gereiht und anschließend die Textbausteine für die Elementarfolgen angehängt.

Sinn der Antwort muss sein, dass der Anwender verständlich mitgeteilt bekommt, welche verschiedenen Möglichkeiten es gibt, die von ihm eingegebenen Werte als Folge mathematisch zu fassen, worin sich diese voneinander unterscheiden bzw. welche Auswirkungen die Wahl einer bestimmten Formel auf die weiteren Folgenglieder hat. Außerdem sollte er mathematische Konstrukte wie Summenreihen, geometrische/arithmetische Folgen, … als solche erkennen können.

Eine Verbalisierung der mathematischen Bildungsvorschrift macht dann Sinn, wenn diese nachvollziehbar, verständlich und in ihrer Komplexität mit der der Formel vergleichbar ist:

*Die Folge ergibt sich, indem von **der Fakultätsfolge** jedes Folgeglied **quadriert wird**.*

Dies kann erreicht werden, indem Textbausteine für jede Elementarfolge bzw. jeden Operator miteinander kombiniert werden.

Beispiele

Um die Fähigkeit seines Lösungs-Programms zu testen, gibt es eine Unzahl verschiedener Anfangsfolgen, an denen man sich versuchen kann. Sehr einfache Folgen sollten ohne weiteres erkannt werden; in besonders grundlegenden Fällen kann die Ausgabe einer Konstruktionsformel auch durch einen manuell eingegebene sprachliche Beschreibung ergänzt werden:

1, 2, 3, 4, 5, 6

Die Folge der natürlichen Zahlen $a_n = a_{n-1} + 1$

1, 3, 5, 7, 9 /
2, 4, 6, 8, 10

Die Folge der (un-)geraden Zahlen $a_n = a_{n-1} + 2$ *mit* $a_1 = 1$ / $a_1 = 2$

1, 1, 2, 3, 5, 8, 13

Die Fibonacci-Folge $a_n = a_{n-1} + a_{n-2}$ *mit* $a_1 = a_2 = 1$

1, 2, 4, 8, 16

Die Folge $a_n = 2^n$

0, 1, 2, 3, 6, 13

Die Folge der Präfixsummen $a_n = a_{n-1} + n$ *mit* $a_1 = 0$

1, 2, 6, 24, 120

Die Folge $a_n = n! = a_{n-1} \cdot n$ *mit* $a_1 = 1$

Auch einfache geometrische und arithmetische Folgen sollten erkannt werden können:

1, 4, 7, 10, 13
2, 6, 18, 54
1, 3, 7, 15, 31

Die Folgen $a_n = a_{n-1} + 3$ *mit* $a_1 = 1, a_n = a_{n-1} \cdot 3$ *mit* $a_1 = 2$, $a_n = a_{n-1} \cdot 2 + 1$

Schwieriger wird es schon, wenn größere Konstanten (die sich aber aus Faktoren oder Potenzen zusammensetzen lassen) oder Kombinationen von Folgen auftreten:

0, 20, 40, 60 ($a_n = a_{n-1} + 2 \cdot 2 \cdot 5$)
1, 128, 16384 ($a_n = a_{n-1} \cdot 2^7$)
49, 50, 52, 55, 61 ($< \mathtt{a} > = < \mathtt{b} > + \mathbf{7}^2$, $b_n = b_{n-1} + n$, $b_1 = 0$)

Auch Linearkombinationen sind denkbar:

$a_n = k_1 a_{n-1} + k_2 a_{n-2} + k_3 a_{n-3} + \ldots$

3.6 Besonderheiten, Erweiterungen und Probleme

Ein bisher nicht genanntes Problem, das sich aus dem Umgang mit Folgen ergibt, liegt in der Ganzzahligkeit der zu untersuchenden unendlichen Folgen. Dies ist insbesondere bei der Wahl der Operatoren (:, $\sqrt{x}$) zu berücksichtigen. Werden Operatoren verwendet, bei denen die Ergebnisse bezüglich ihrer Ganzzahligkeit nicht eindeutig sind, muss dies erkannt und geeignet behandelt werden (Gauss-Klammern o.Ä.).

Ein weiterer Aspekt, dem Aufmerksamkeit geschenkt werden sollte, besteht darin, dass bei den Berechnungen der rekursiven Folgen nach Möglichkeit für ein Folgeglied nicht noch einmal sämtliche Vorgängerglieder berechnet werden müssen. Dies kann erreicht werden, indem sämtliche Vorgängerglieder zwischengespeichert werden.

Die Aufgabe kann man erweitern, wenn man die zu verwendenden Folgen weiter verallgemeinert. Interessante Möglichkeiten ergeben sich etwa durch Einführung algorithmischer Elemente wie Fallunterscheidungen, Summenreihen beliebiger Ausdrücke oder zusätzlicher Operatoren (MIN(x, y), MAX(x, y), NEXTPRIME(x), ...), die bei der Erzeugung verwendet werden können.

Die freie Suche in den Folgen ist als Erweiterung schon erwähnt worden. Eine weitere Variante ist, dass die angegebenen Folgenglieder in der zu bestimmenden Folge nicht notwendigerweise direkt aufeinander folgen müssen, sondern eine anzugebende Anzahl an potentiell dazwischenliegenden unbekannten Gliedern möglich ist.

3.7 Bewertungskriterien

Auch wenn die Aufgabenstellung bezüglich der gewünschten Funktionalität nicht definitiv ist, gibt es doch grundsätzliche Ansprüche, denen Lösungen genügen müssen. Zum einen muss das Folgenlexikon vollständig automatisch erstellt werden können: nicht der Benutzer soll es sein, der Vorgaben über die mögliche Form von Folgen macht, sondern diese sollten im Programm verankert sein. Eine Lösung, die im Wesentlichen die manuelle Erstellung des Folgenlexikons vorsieht, geht an der Aufgabe deutlich vorbei und kann deshalb auch bei ansonsten hoher Qualität nicht hoch bewertet werden. Je freier der gewählte Ansatz zur Erzeugung von Folgen dabei ist, je mehr strukturell verschiedene Folgen also generiert werden können, desto höher ist die Leistung des Programms einzuschätzen. Da aber bei allen Ansätzen prinzipiell unendlich viele Folgen generiert werden können, muss die Menge der im Lexikon abgespeicherten Folgen sinnvoll begrenzt werden.

Die Erzeugung zueinander äquivalenter und ähnlicher Folgen-Formeln muss vermieden bzw. deren Ausgabe als vielfach gefundene Lösung unterbunden oder zumindest geeignet organisiert werden. Außerdem sollte das Problem der Ganzzahligkeit von Folgen erkannt und behandelt bzw. umgangen werden.

Die Folgen-Suche wiederum muss durch geeignete Speicherung effizient möglich sein; mit nicht eindeutigen Suchergebnissen muss geeignet umgegangen werden. Die Präsentation der

Suchergebnisse muss übersichtlich, einheitlich und verständlich erfolgen. Neben präziser mathematischer Korrektheit und angemessener Einfachheit können textuelle Ergänzungen oder Erklärungen eingebaut werden. In jedem Fall ist die gewählte Darstellung zu begründen.

Um die Fähigkeiten des Programms unter Beweis zu stellen, müssen mindestens fünf möglichst verschieden aufgebaute Anfangsfolgen gesucht werden. Die Beispiele sollten dazu dienen zu klären, warum die gefundenen Folgen gefunden werden konnten, bzw. welche Arten von Funktionen auffindbar sind; dies sollte im Einklang mit den selbst formulierten Anforderungen stehen. Gleichzeitig muss aber auch genau der negative Fall behandelt werden: welche Folgen lassen sich warum nicht erzeugen bzw. finden.

Perlen der Informatik – aus den Einsendungen

Allgemeines

Ein paar einleitende Gedanken … Gedanken trifft es gut: Das ist alles relativ zusammenhanglos.

Es gibt noch andere Lösungsansätze. Zum Beispiel könnte *Hier endete die Einsendung.*

Die Prozedur „pause“ sorgt dafür, dass das Programm sich nicht aufhängt, wenn es zu viel zu tun gibt. *Aha, eine Software-Suizid-Schutz-Maßnahme.*

Um das zu erreichen, muss `sort()` eine Funktion übergeben werden, die den Raumschiffoperator (`<=>`) anwendet.

Der Beweis dazu findet sich auf Wikipedia und die Gegenbeispiele, die den Satz widerlegen, in den Links.

MCI-Aspekte

Das Benutzer-Interface des Programms ist noch recht unfreundlich.

Diese Aufgabe bot eigentlich kaum Möglichkeiten, das Benutzer-Interface unverständlich zu gestalten, ich habe mich trotzdem im Rahmen des Verfügbaren bemüht.

Zum Starten der grafischen Benutzeroberfläche bitte „Ende“ eingeben.

Aufgabe 1

Die folgenden Faltpolygone sind alle unendlich groß, man sollte also nicht versuchen, sie nachzufalten!

… denn ich bin während der Entwicklung meines Programms häufig auf das Problem gestoßen, dass ich das zusammengefaltete Polygon nicht mehr gefunden habe.

Zudem habe ich noch keine Überprüfung eingebaut, ob die Schnittlinie […] sich selbst nicht schneidet.

Der Benutzer könnte theoretisch eine so große Zahl angeben, dass sie nicht in den Speicher seines Rechners passt.

Die Liste der Punkte wird gelehrt.

Die Entfaltfunktion dient also sowohl zum Entfalten als auch zum Wieder-Falten. Dies mag ein bisschen unlogisch klingen.

Dem armen Kontrolleur […] habe ich eine Funktion gegeben, um sich abzureagieren. Sie wird mit dem Schalter „Müll“ gestartet. Dabei wird in zufälliger Weise ein Schnitt oder eine Faltung vorgenommen, so dass das Stück Papier zusammengeknüllt auf dem Bildschirm herumliegt.

Eine gute Möglichkeit für Faltpolygone besteht darin, sie für Horoskope zu benutzen. Man lässt eine Testperson ein Faltpolygon erstellen und interpretiert dann die Ergebnisse. Mit ein paar mehr Faltvorgängen bleibt natürlich viel mehr Raum für Vermutungen über die Zukunft.

Aufgabe 2

Biodoku

vorbesetzte Lücken

`permutate_verysmall()` bringt die größten Veränderungen mit sich.

Die average-case-Laufzeit unter 50 leeren Feldern ist gleich null.

Nachdem man herausgefunden hat, dass es theoretisch 115792089237316195423570985008687907853269984665640564039457584007913129639936 Möglichkeiten gibt, ein Bidokufeld zu füllen, macht man sich erhebliche Gedanken über die Laufzeit.

Aufgabe 3

Bei der Aufgabe stellt sich zunächst einmal die Frage, was überhaupt das Problem ist.