3. Schlangenjagd

3.1 Einführung und Motivation

In diesem Programmierpraktikum schicken wir Sie auf die Schlangenjagd – auf die Jagd nach Zeichenschlangen um genau zu sein. Als Zeichenschlangen bezeichnen wir dabei räumlich angeordnete Zeichenketten. Zeichenschlangen können in vielerlei Gestalt auftreten. Abbildung 1 stellt verschiedene Ausprägungen der Zeichenkette „FERNUNIVERSITÄTINHAGEN“ dar.

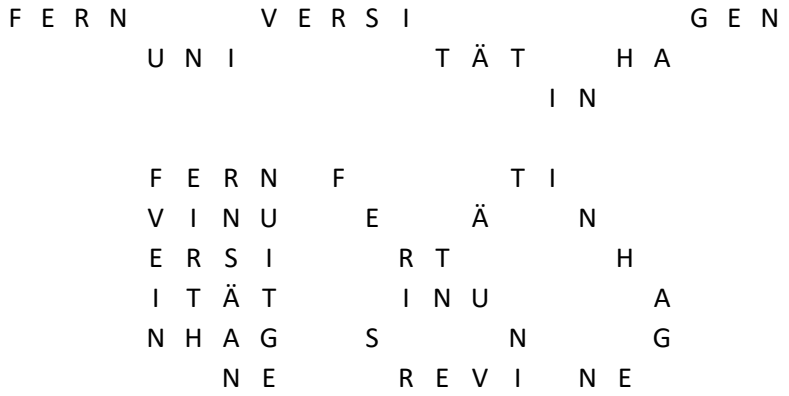


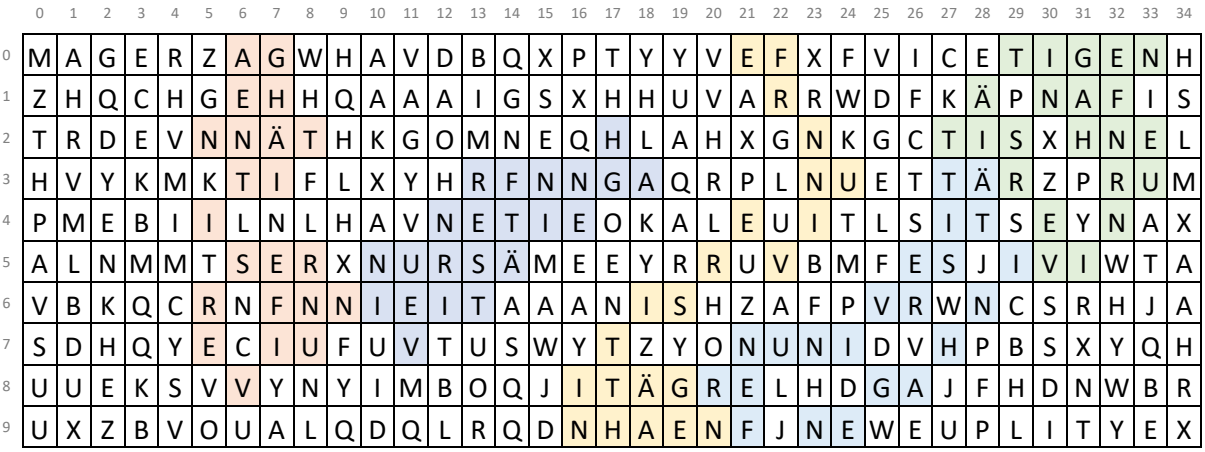
Abbildung 1: Zeichenketten in unterschiedlicher Anordnung

Freistehende Zeichenschlangen sind trotz möglicherweise ungewohnter Anordnung relativ einfach zu identifizieren. Deutlich schwieriger wird es, wenn sich eine unbekannte Anzahl solcher Zeichenschlangen zwischen einer Menge weiterer Zeichen versteckt hält.

Für die Schlangensuche betrachten wir einen Zeichendschungel aus rechteckig in Zeilen und Spalten angeordneten Feldern. Jedes Feld enthält ein einzelnes Zeichen, aber nicht jedes Feld bzw. Zeichen innerhalb des Dschungels lässt sich notwendigerweise einer Zeichenschlange zuordnen. Zur Veranschaulichung betrachten wir den Zeichendschungel in Abbildung 2, in dem sich mehrere Zeichenschlangen „FERNUNIVERSITÄTINHAGEN“ versteckt halten. Die Schlangen sind gut getarnt, so dass die manuelle Suche vermutlich eine lange Zeit in Anspruch nehmen würde. Die Verstecke der Schlangen werden daher direkt auf der nächsten Seite in Abbildung 3 aufgedeckt.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
0	M	A	G	E	R	Z	A	G	W	H	A	V	D	B	Q	X	P	T	Y	Y	V	E	F	X	F	V	I	C	E	T	I	G	E	N	H
1	Z	H	Q	C	H	G	E	H	H	Q	A	A	A	I	G	S	X	H	H	U	V	A	R	R	W	D	F	K	Ä	P	N	A	F	I	S
2	T	R	D	E	V	N	N	Ä	T	H	K	G	O	M	N	E	Q	H	L	A	H	X	G	N	K	G	C	T	I	S	X	H	N	E	L
3	H	V	Y	K	M	K	T	I	F	L	X	Y	H	R	F	N	N	G	A	Q	R	P	L	N	U	E	T	T	Ä	R	Z	P	R	U	M
4	P	M	E	B	I	I	L	N	L	H	A	V	N	E	T	I	E	O	K	A	L	E	U	I	T	L	S	I	T	S	E	Y	N	A	X
5	A	L	N	M	M	T	S	E	R	X	N	U	R	S	Ä	M	E	E	Y	R	R	U	V	B	M	F	E	S	J	I	V	I	W	T	A
6	V	B	K	Q	C	R	N	F	N	N	I	E	I	T	A	A	A	N	I	S	H	Z	A	F	P	V	R	W	N	C	S	R	H	J	A
7	S	D	H	Q	Y	E	C	I	U	F	U	V	T	U	S	W	Y	T	Z	Y	O	N	U	N	I	D	V	H	P	B	S	X	Y	Q	H
8	U	U	E	K	S	V	V	Y	N	Y	I	M	B	O	Q	J	I	T	Ä	G	R	E	L	H	D	G	A	J	F	H	D	N	W	B	R
9	U	X	Z	B	V	O	U	A	L	Q	D	Q	L	R	Q	D	N	H	A	E	N	F	J	N	E	W	E	U	P	L	I	T	Y	E	X

Abbildung 2: Zeichendschungel mit mehreren versteckten Zeichenschlangen „FERNUNIVERSITÄTINHAGEN“



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
0	M	A	G	E	R	Z	A	G	W	H	A	V	D	B	Q	X	P	T	Y	Y	V	E	F	X	F	V	I	C	E	T	I	G	E	N	H
1	Z	H	Q	C	H	G	E	H	H	Q	A	A	A	I	G	S	X	H	H	U	V	A	R	R	W	D	F	K	Ä	P	N	A	F	I	S
2	T	R	D	E	V	N	N	Ä	T	H	K	G	O	M	N	E	Q	H	L	A	H	X	G	N	K	G	C	T	I	S	X	H	N	E	L
3	H	V	Y	K	M	K	T	I	F	L	X	Y	H	R	F	N	N	G	A	Q	R	P	L	N	U	E	T	T	Ä	R	Z	P	R	U	M
4	P	M	E	B	I	I	L	N	L	H	A	V	N	E	T	I	E	O	K	A	L	E	U	I	T	L	S	I	T	S	E	Y	N	A	X
5	A	L	N	M	M	T	S	E	R	X	N	U	R	S	Ä	M	E	E	Y	R	R	U	V	B	M	F	E	S	J	I	V	I	W	T	A
6	V	B	K	Q	C	R	N	F	N	N	I	E	I	T	A	A	A	N	I	S	H	Z	A	F	P	V	R	W	N	C	S	R	H	J	A
7	S	D	H	Q	Y	E	C	I	U	F	U	V	T	U	S	W	Y	T	Z	Y	O	N	U	N	I	D	V	H	P	B	S	X	Y	Q	H
8	U	U	E	K	S	V	V	Y	N	Y	I	M	B	O	Q	J	I	T	Ä	G	R	E	L	H	D	G	A	J	F	H	D	N	W	B	R
9	U	X	Z	B	V	O	U	A	L	Q	D	Q	L	R	Q	D	N	H	A	E	N	F	J	N	E	W	E	U	P	L	I	T	Y	E	X

Abbildung 3: Zeichendschungel mit fünf gefundenen Zeichenschlangen „FERNUNIVERSITÄTINHAGEN“

Suchprobleme wie das der Schlangensuche können von einem Computer in der Regel schneller als von einem Menschen gelöst werden – deutlich bequemer ist das natürlich auch. Zudem lassen sich mit Hilfe von entsprechenden Algorithmen flexibel weitere Problemeigenschaften berücksichtigen. Vor der Formulierung eines solchen Algorithmus muss das Problem zunächst detaillierter untersucht und ausformuliert werden.

Bisher wurde für die Anordnung der Zeichenschlangen angenommen, dass sich aufeinanderfolgende Schlangenglieder in senkrecht, waagerecht oder diagonal angrenzenden Feldern befinden bzw. diesen zugeordnet sind. Diesen Zusammenhang wollen wir folgend verallgemeinern. Für die Suche werden neben dem Zeichendschungel eine oder mehrere Schlangenarten vorgegeben, die sich in diesem Dschungel versteckt halten können. Eine Schlangenart wird einerseits durch eine bestimmte Zeichenkette und andererseits durch eine Nachbarschaftsstruktur charakterisiert, die eine zulässige Struktur zwischen Schlangengliedern bzw. den zugeordneten Feldern vorgibt. Eine Nachbarschaftsstruktur bezieht sich also nicht auf den gesamten Dschungel und seine Felder, sondern kann abhängig von der Schlangenart variieren.

Halten sich mehrere Zeichenschlangen im Dschungel versteckt, so besteht die Möglichkeit, dass sich zwei Schlangen schneiden oder überdecken. Für diesen Fall ist zu klären, ob nur eine oder beide Schlangen für die Schlangensuche gezählt werden sollen. Wir legen dies eindeutig fest, indem wir jedem Feld einen nichtnegativen, ganzzahligen Wert für seine Verwendbarkeit zuweisen. Nimmt die Verwendbarkeit für alle Felder den Wert eins an, so wird nach Schlangen gesucht, die gleichzeitig ohne Überschneidungen im Dschungel auffindbar sind. Bei Werten größer als eins können sich Schlangen in den entsprechenden Feldern überschneiden. Es wäre auch möglich, dass zwei Schlangen derselben Art durch dieselben Felder verlaufen, dahingehend also nicht eindeutig sind.

Durch die Möglichkeit von Überschneidungen und einer beschränkten Verwendbarkeit von Feldern ist die Suche nach „allen Schlangen“ keine eindeutige Zielsetzung mehr. Wir führen als Erweiterung ein Punktesystem ein, bei dem für jede gefundene Schlange einer Art und für jedes dabei verwendete Feld eine jeweils individuell festgelegte Anzahl an Punkten vergeben

wird. Das Ziel der Suche besteht folglich darin, eine Menge von Schlangen mit einer möglichst hohen Gesamtpunktzahl zu finden. Die gefundene Schlangenmenge stellt dann die Lösung einer Probleminstanz bestehend aus einem Dschungel und den zugehörigen Schlangenarten dar. Zudem soll für jede Instanz ein Zeitlimit festgelegt werden, das die zulässige Suchzeit begrenzt.

3.2 Regelwerk

Die folgenden Definitionen und Regeln fassen die bisherigen Beschreibungen der Problemstellung zusammen.

Dschungel: Ein Zeichendschungel wird durch eine Matrix mit einer vorgegebenen Anzahl an Zeilen und Spalten repräsentiert. Für jede Kombination aus einer Zeile und einer Spalte ergibt sich ein Feld, das ein einzelnes Zeichen enthalten kann.

Schlangenarten: Eine Schlangenart wird durch eine Zeichenkette und eine Nachbarschaftsstruktur definiert. Die Nachbarschaftsstruktur gibt mögliche Anordnungen für Zeichenschlangen im Dschungel vor. Für jeden Dschungel ist eine Menge von möglichen Schlangenarten vorgegeben.

Nachbarschaftsstrukturen: Jeder Schlangenart ist eine Nachbarschaftsstruktur zugeordnet. Diese definiert den räumlichen Bezug zwischen den Feldern von zwei aufeinanderfolgenden Schlangengliedern einer Schlange im Zeichendschungel. Folgende Nachbarschaftsstrukturen werden vorgegeben:

> **Distanz-Nachbarschaft:** Ein Feld liegt in der Nachbarschaft eines anderen Feldes, wenn es sich mit einer bestimmten Anzahl von Zügen auf jeweils angrenzende Felder in senkrechter, waagerechter oder diagonaler Richtung erreichen lässt. Die maximale Anzahl von Zügen wird durch einen Distanzwert vorgegeben. Ein Feld liegt nicht in seiner eigenen Nachbarschaft.
>
> **Sprung-Nachbarschaft:** Ein Sprung entspricht einer vorgegebenen Anzahl an Zügen in senkrechter und waagerechter Richtung, wofür zwei Distanzwerte vorgegeben werden. Die Richtungen können dabei in beliebiger Reihenfolge gewählt werden. Nur die am Ende erreichten Felder, nicht die dabei durchquerten Felder, gehören der Nachbarschaft an.

Ausprägungen der oben beschriebenen Nachbarschaftsstrukturen werden in Abbildung 4 für verschiedene Distanzparameter dargestellt. Die grün hinterlegten Felder (X) liegen in der Nachbarschaft des orangenen Feldes (O).

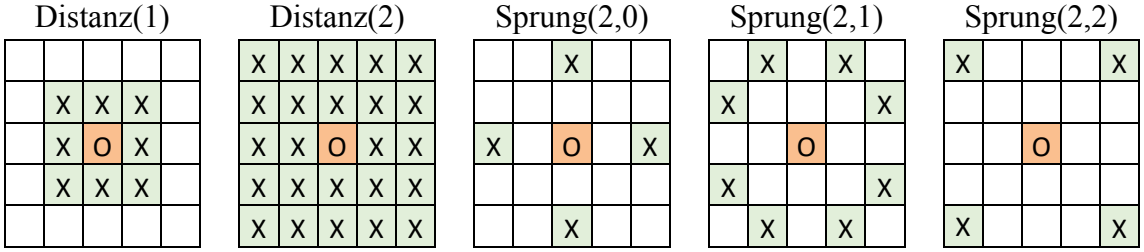


Abbildung 4: Mögliche Ausprägungen von Nachbarschaftsstrukturen

Schlangen: Zeichenschlangen bestehen aus einer geordneten Menge von Schlangengliedern, die jeweils einem Feld im Dschungel zugeordnet sind. Die Zeichenfolge der zugeordneten Felder muss mit der Zeichenkette der jeweiligen Schlangenart übereinstimmen. Weiterhin müssen die Felder aufeinanderfolgender Schlangenglieder entsprechend der für die Schlangenart vorgegebenen Nachbarschaftsstruktur angeordnet sein.

Verwendbarkeit von Feldern: Jedem Feld im Dschungel wird ein nichtnegativer ganzzahliger Wert für dessen maximale Verwendbarkeit zugewiesen. Dieser Wert gibt also vor, wie viele Schlangen dieses Feld maximal überdecken bzw. wie viele Schlangenglieder diesem Feld zugeordnet werden dürfen.

Abbildung 5 zeigt in der ersten Spalte einen Zeichendschungel mit drei unterschiedlichen Angaben zur Verwendbarkeit der Felder in der zweiten Spalte. Nicht relevante Felder sind grau hinterlegt. Im Zeichendschungel sind zwei Schlangenarten „SCHLANGE“ und „DSCHUNGEL“, jeweils mit Distanz(1)-Nachbarschaft, versteckt. Die Spalten drei und vier enthalten zulässige Lösungen, die hinsichtlich der Anzahl der gefundenen Schlangen maximal sind. Die gefundenen Schlangen sind orange markiert und bei mehrfacher Verwendung der Felder dunkler bzw. stärker gesättigt. Im letzten Fall wurde unter Lösung 1 zweimal dieselbe Schlange „SCHLANGE“ gefunden, was bei der gegebenen Verwendbarkeit zulässig ist.

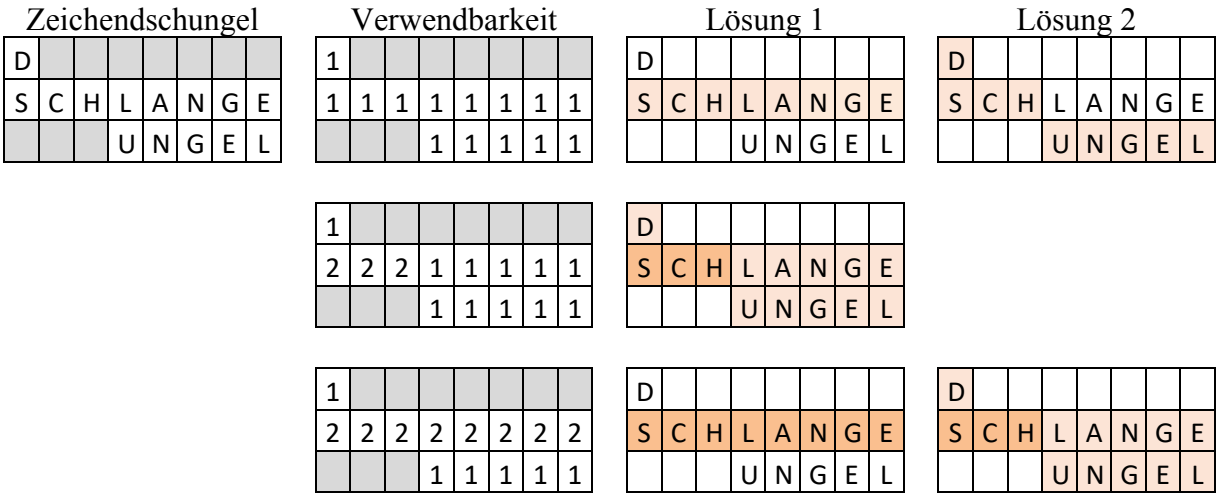


Abbildung 5: Beispiele für die Verwendbarkeit von Feldern

Punkte: Bei der Schlangensuche kann für jede gefundene Schlange eine bestimmte Punktzahl erreicht werden, die von der jeweiligen Schlangenart abhängig ist. Darüber hinaus werden mit jedem zugeordneten Schlangenglied Punkte für einzelne Felder vergeben. Die jeweilige Punktzahl sei dabei – genau wie bei der Verwendbarkeit von Feldern – ein nichtnegativer ganzzahliger Wert. Das Ziel der Schlangensuche besteht darin, so viele Punkte wie möglich zu erreichen.

Wir erweitern das erste Beispiel aus Abbildung 5 durch die Angabe von Punkten für einzelne Felder und Schlangenarten. Für die Schlangenart „SCHLANGE“ sollen acht Punkte, für die Schlangenart „DSCHUNGEL“ neun Punkte vergeben werden. Die Punkte für die einzelnen Felder werden in Abbildung 6 angegeben.

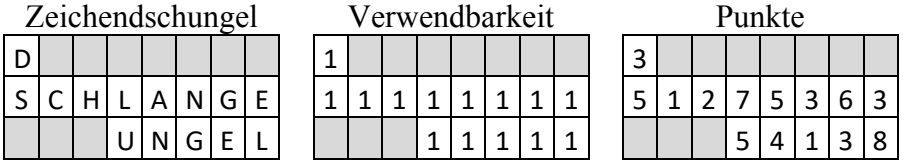


Abbildung 6: Beispiel für die Punkte verwendeter Felder

Für die beiden alternativen Lösungen aus Abbildung 5 werden folgende Gesamtpunktzahlen erreicht, wobei sich der erste Summand auf die dazugehörige Schlangenart und alle weiteren auf die überdeckten Felder beziehen:

- „SCHLANGE“: $8 + (5 + 1 + 2 + 7 + 5 + 3 + 6 + 3) = 40$
- „DSCHUNGEL“: $9 + (3 + 5 + 1 + 2 + 5 + 4 + 1 + 3 + 8) = 41$

Wenn Sie das Beispiel genauer untersuchen, werden Sie mehr als eine Lösung „SCHLANGE“ finden. Die erreichte Punktzahl ist jedoch in jedem Fall nicht größer als die der oben angegebenen Lösung. Die Lösung „DSCHUNGEL“ ist in diesem Fall also optimal.

Probleminstanz: Eine Probleminstanz besteht aus einem Dschungel und einer Menge von zugehörigen Schlangenarten. Jedem Dschungelfeld ist ein Zeichen, ein Wert für die Verwendbarkeit und eine Punktzahl zugeordnet. Die Schlangenarten geben vor, nach welchen Zeichenketten im Dschungel gesucht werden soll, wobei die jeweilige Nachbarschaftsstruktur zulässige Anordnungen definiert. Jeder Schlangenart ist, genau wie den Feldern, eine Punktzahl zugeordnet. Ein Zeitlimit gibt vor, wie lange im Dschungel nach einer Lösung gesucht werden darf.

Lösung: Die Lösung zu einer Probleminstanz besteht aus einer Menge von Schlangen. Für eine zulässige Lösung müssen die Schlangen in der Abfolge und Anordnung ihrer Glieder den Vorgaben der Schlangenart entsprechen und zusammen die Bedingung für die Verwendbarkeit einzelner Felder einhalten. Die Güte einer Lösung bestimmt sich aus der Summe aller Punkte für die gefundenen Schlangen und die dabei überdeckten Felder.

3.3 Aufgabenstellung

Aufbauend auf dem obigen Regelwerk soll in diesem Programmierpraktikum die folgende Aufgabenstellung erfüllt werden:

Entwerfen und implementieren Sie ein Programm, das in vorgegebenen Probleminstanzen nach einer möglichst guten Lösung suchen kann. Darüber hinaus sollen einfache Probleminstanzen erzeugt und gefundene Lösungen überprüft und bewertet werden können. Der Fokus soll dabei auf der Schlangensuche liegen.

1. Für die **Schlangensuche** ist ein geeigneter Suchalgorithmus zu entwerfen und zu implementieren. Der Algorithmus soll auf Basis einer vorgegebenen Probleminstanz, bestehend aus einem Dschungel und Schlangenarten, innerhalb eines vorgegebenen Zeitlimits eine zulässige Lösung finden. Die Lösung sollte dabei eine möglichst hohe Gesamtpunktzahl für die Schlangenarten und überdeckten Felder erreichen.
2. Ein **Dschungelgenerator** für die Erzeugung einfacher Probleminstanzen ist zu entwickeln. Vorgegeben sind die gewünschte Größe des Dschungels, eine Auswahl an

Zeichen und eine Menge von Schlangenarten. Für jede Schlangenart soll eine bestimmte Anzahl an Schlangen im Dschungel verteilt werden. Verbleibende Felder sind mit weiteren zulässigen Zeichen aufzufüllen. Die geforderte Funktionalität des Dschungelgenerators wird im Gegensatz zur Schlangensuche auf einfache Probleminstanzen beschränkt, in denen sich die erzeugten Schlangen nicht überschneiden und jedes Feld maximal einmal verwendet werden soll.

3. Implementieren Sie Funktionalität zur **Prüfung der Zulässigkeit** und **Bewertung** von Lösungen. Für die Zulässigkeit müssen alle gegebenen Bedingungen erfüllt sein. Anschließend kann die für die Lösung erreichte Punktzahl berechnet werden.

Folgende **Anforderungen** werden im Rahmen des Programmierpraktikums an Sie gestellt:

- Sie beschäftigen sich bei der Lösung der Programmieraufgabe mit der Schwierigkeit des Algorithmenentwurfs und kommen dabei insbesondere mit der Komplexität des Lösungsraums in Kontakt. Da der Lösungsraum in Abhängigkeit von der Probleminstanz sehr groß werden kann, muss die Suche nach einer möglichst guten Lösung heuristisch gesteuert werden. Der Entwurf dieser Heuristiken erfordert eine intensive Auseinandersetzung mit der Problemstellung.
- Unter Anwendung der Prinzipien der objektorientierten Programmierung ist der Softwareentwurf so zu konzipieren, dass das Lösungsverfahren durch Konfiguration und Implementierung von Erweiterungspunkten anpassbar ist. Dadurch soll die Möglichkeit geschaffen werden, verschiedene heuristische Ansätze experimentell zu untersuchen und das Lösungsverfahren an unterschiedliche Probleminstanzen anzupassen.
- Bei der Implementierung des Lösungsverfahrens ist dessen Effizienz zu berücksichtigen, um eine möglichst gute Lösung oder sogar eine optimale Lösung in möglichst kurzer Zeit zu ermitteln. Dazu sind zum einen geeignete Datenstrukturen zur Speicherung von Laufzeitinformationen zu verwenden. Andererseits ist bei der Implementierung des Algorithmus selbst und bei der Lösung von Unterproblemen sowie bei der Anwendung von Heuristiken auf eine möglichst zeiteffiziente Abarbeitung zu achten.
- Dem Prinzip eines guten Softwareentwurfs folgend, sind die Bestandteile des Lösungsverfahrens durch geeignete Testklassen abzusichern. Zur Ausführung der Tests wird ein verbreitetes Rahmenwerk zum Komponententest verwendet.
- Es ist eine Dateneingabe und -ausgabe auf der Basis von XML-Dokumenten zu realisieren.
- Vorgegebene oder selbst erzeugte Probleminstanzen sowie gefundene Lösungen sollen zur Visualisierung textbasiert ausgegeben werden können.
- Das Lösungsverfahren ist in ein Hauptprogramm einzubetten. Die Konfiguration des Programms soll vollständig über eine einzulesende Eingabedatei und durch Kommandozeilenparameter durchführbar sein.

Nachfolgend wird die Aufgabenstellung weiter konkretisiert. Verschiedene Hinweise zur Lösung und Vorgehensweise werden in den Abschnitten 3.4 und 3.5 gegeben, während in Kapitel 4 Hinweise zur Durchführung des Programmierpraktikums zu finden sind.

3.4 Konkretisierung der Aufgabenstellung

Im Programmierpraktikum ist ein Lösungsverfahren für die Suche nach Zeichenschlangen zu konzipieren und zu implementieren. Das Lösungsverfahren soll den in Abschnitt 3.3 genannten funktionalen und nicht-funktionalen Anforderungen genügen. Für die Suche sind verschiedene Schlangenarten mit Zeichenketten, Nachbarschaftsstrukturen und Punkten sowie die Verwendbarkeit und Punkte einzelner Felder im Dschungel zu berücksichtigen. Die gefundene Lösung muss unter Berücksichtigung der gegebenen Regeln und den Vorgaben der Probleminstanz zulässig sein. Die Gesamtpunktzahl ist zu maximieren.

Das zu entwickelnde Lösungsverfahren ist in ein Hauptprogramm einzubetten, das eine Probleminstanz aus einer Eingabedatei einliest, gegebenenfalls das Lösungsverfahren startet sowie für die zu lösende Probleminstanz den Dschungel, die gefundenen Zeichenschlangen und die erreichten Punkte ausgibt. Ein Dschungelgenerator soll die Erzeugung einfacher Probleminstanzen anhand von Vorgaben zu Dschungel und Schlangenarten erlauben. Es soll möglich sein, selber gefundene wie auch vorgegebene Lösungen auf ihre Zulässigkeit zu prüfen und anschließend zu bewerten.

Die Anforderungen an die zu implementierenden Bestandteile werden im Folgenden genauer beschrieben. Das erfolgreiche Bestehen des Programmierpraktikums erfordert zwingend die vollständige Umsetzung der geforderten Funktionalitäten.

3.4.1 Datenmodell

Zunächst ist eine programminterne Repräsentation für die Probleminstanz und die gesuchte Lösung zu entwickeln. Wir bezeichnen diese folgend auch als das Datenmodell des Programms. Die Probleminstanz besteht aus dem Zeichendschungel und zugehörigen Schlangenarten. Die Lösung enthält die gefundenen Schlangen der angegebenen Schlangenarten.

Das Regelwerk aus Abschnitt 3.2 muss im Datenmodell geeignet abgebildet werden. Mögliche Kandidaten für Entitätstypen sind der Dschungel und dessen Felder, Schlangenarten mit zugeordneten Nachbarschaftsstrukturen sowie die Schlangen und ihre Glieder. Im Programm lassen sich die Entitätstypen jeweils als Klassen implementieren. Attribute geben mögliche Eigenschaften vor. Beziehungen zwischen den Klassen müssen ebenfalls geeignet berücksichtigt werden.

Das UML-Diagramm in Abbildung 7 zeigt eine mögliche Strukturierung der Daten. Für die Implementierung in Java kann es sinnvoll sein, das Modell weiter zu modifizieren oder auch zu erweitern. Zudem müssen geeignete Datentypen und -strukturen für Attribute und Beziehungen festgelegt werden. Dabei sind insbesondere die Anforderungen des Lösungsverfahrens für eine effiziente Ausführung zu berücksichtigen.

Die Dateneingabe und Datenausgabe erfolgt XML-basiert. Die Probleminstanz und Lösung werden in einem XML-Dokument innerhalb des Wurzelelements `<Schlangenjagd>` definiert. Die Elemente `<Dschungel>` und `<Schlangenarten>` definieren die Probleminstanz. Innerhalb des Elements `<Schlangen>` soll die gefundene Lösung in Form mehrerer Elemente vom Typ `<Schlange>` gespeichert werden.

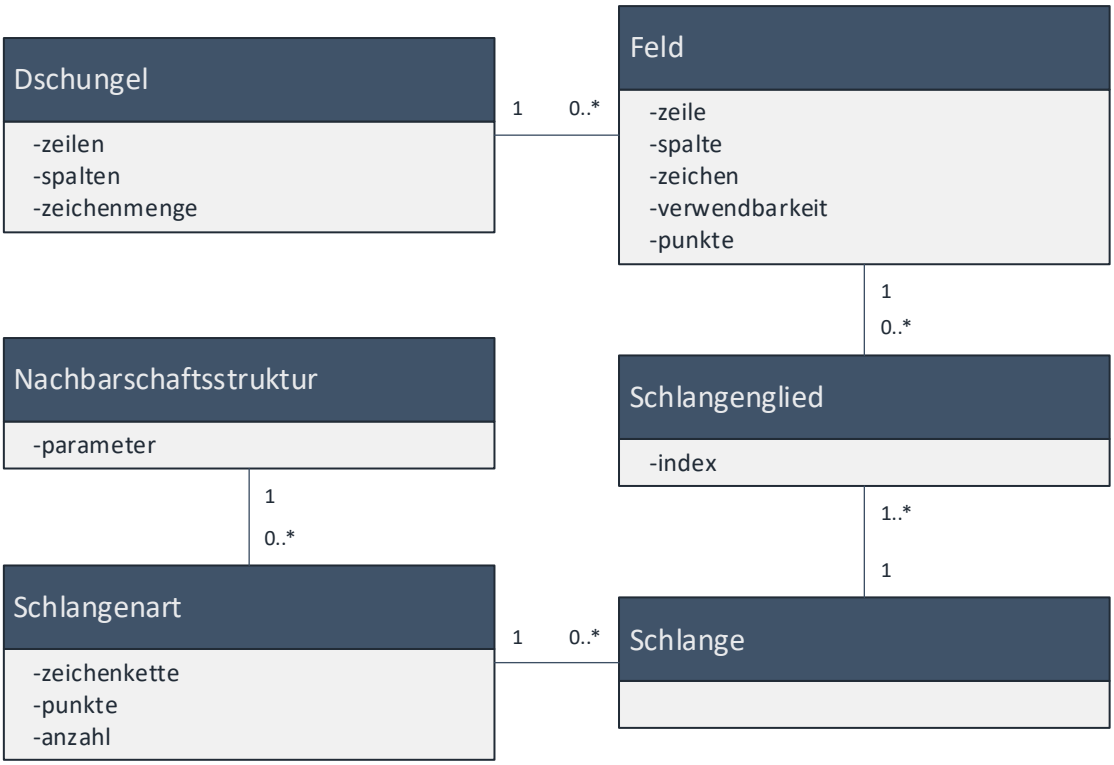


Abbildung 7: Mögliche Struktur des Datenmodells

Im Element <Zeit> wird unter <Vorgabe> die maximal zulässige Laufzeit Ihres Programms zur Lösung der Probleminstanz vorgegeben. Die Zeiteinheit wird über ein Attribut in Millisekunden „ms“, Sekunden „s“, Minuten „min“, Stunden „h“, oder Tagen „d“ angegeben. Die tatsächliche Laufzeit soll nach erzeugter Lösung unter <Abgabe> gespeichert werden.

Der <Dschungel> enthält Attribute für die Anzahl der Zeilen und Spalten sowie für die unabhängig von den Schlangenarten im Dschungel zulässigen Zeichen. Felder im Dschungel können über Kindelemente vom Typ <Feld> definiert werden. Sie enthalten als Inhalt das zugehörige Zeichen und Attribute für die Zeile und Spalte, die Verwendbarkeit und die erreichbaren Punkte. Zusätzlich soll ein Feld durch ein id-Attribut eindeutig identifiziert werden. Der Wert dieses Attributs wird unter Berücksichtigung von Zeilennummer z, Spaltennummer s und Spaltenanzahl S eindeutig als

$$\text{„F<Nummer>“ mit <Nummer>} := z \cdot S + s$$

festgelegt. Beachten Sie, dass die Zeilen- und Spaltennummerierung bei 0 beginnt. Für die Verwendbarkeit und die Punktzahl wird jeweils ein Standardwert von „1“ vorgegeben. Ein fehlendes Feld ist implizit leer mit einer Verwendbarkeit und Punktzahl von „0“.

Eine <Schlangenart> innerhalb von <Schlangenarten> wird durch ein id-Attribut im Format „A<Nummer>“ eindeutig identifiziert, wobei <Nummer> einer in aufsteigender Reihenfolge zugeordneten Zahl $0, 1, 2, \ldots$ entspricht. Ein weiteres Attribut legt die Anzahl der erreichbaren Punkte mit einem Standardwert von „1“ fest. Das letzte Attribut gibt die Anzahl der im Dschungel versteckten Schlangen an. Diese Anzahl muss nicht mit der Anzahl der Schlangen in einer optimalen Lösung übereinstimmen. Als Kindelemente werden die

<Zeichenkette> und die <Nachbarschaftsstruktur> definiert. Letztere enthält als Attribut den Typ der Nachbarschaftsstruktur und als Inhalt <Parameter> für dessen Ausprägung. Mögliche Nachbarschaftsstrukturen sind gemäß des Regelwerks in Abschnitt 3.2 „Distanz“ und „Sprung“.

Jede im Dschungel gefundene Schlange soll durch ein Element <Schlange> innerhalb von <Schlangen> repräsentiert werden. Eine Schlange referenziert über ein Attribut die jeweilige Schlangenart. Jede Schlange enthält eine geordnete Menge an Elementen <Schlangenglied>, die in korrekter Reihenfolge jeweils einem Feld mit passendem Zeichen zugeordnet sind.

Listing 1 zeigt die zugehörige Dokumenttypdefinition (DTD). Die angegebene DTD definiert die Dokumentstruktur, die von den Dokumenten unbedingt einzuhalten ist.

```
01 <!ELEMENT Schlangenjagd (Zeit?, Dschungel, Schlangenarten,
           Schlangen?)>
02 <!ELEMENT Zeit (Vorgabe, Abgabe?)>
03 <!ATTLIST Zeit
04     einheit (ms|s|min|h|d) "s"
05 >
06 <!ELEMENT Vorgabe (#PCDATA)>
07 <!ELEMENT Abgabe (#PCDATA)>
08 <!ELEMENT Dschungel (Feld*)>
09 <!ATTLIST Dschungel
10     zeilen CDATA #REQUIRED
11     spalten CDATA #REQUIRED
12     zeichen CDATA #REQUIRED
13 >
14 <!ELEMENT Feld (#PCDATA)>
15 <!ATTLIST Feld
16     id ID #REQUIRED
17     zeile CDATA #REQUIRED
18     spalte CDATA #REQUIRED
19     verwendbarkeit CDATA "1"
20     punkte CDATA "1"
21 >
22 <!ELEMENT Schlangenarten (Schlangenart+)>
23 <!ELEMENT Schlangenart (Zeichenkette, Nachbarschaftsstruktur)>
24 <!ATTLIST Schlangenart
25     id ID #REQUIRED
26     punkte CDATA "1"
27     anzahl CDATA #REQUIRED
28 >
29 <!ELEMENT Zeichenkette (#PCDATA)>
30 <!ELEMENT Nachbarschaftsstruktur (Parameter*)>
31 <!ATTLIST Nachbarschaftsstruktur
32     typ CDATA #REQUIRED
33 >
34 <!ELEMENT Parameter EMPTY>
35 <!ATTLIST Parameter
36     wert CDATA #REQUIRED
37 >
38 <!ELEMENT Schlangen (Schlange*)>
39 <!ELEMENT Schlange (Schlangenglied+)>
```

```
40 <!ATTLIST Schlange
41     art IDREF #REQUIRED
42 >
43 <!ELEMENT Schlangenglied EMPTY>
44 <!ATTLIST Schlangenglied
45     feld IDREF #REQUIRED
46 >
```

Listing 1: Dokumenttypdefinition

Wir betrachten als Beispiel in Abbildung 8 einen Zeichendschungel mit zwei Zeilen und vier Spalten, in dem die Zeichenschlange „FERNUNI“ gefunden wurde.

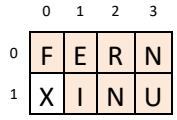


Abbildung 8: Zeichendschungel mit Zeichenschlange "FERNUNI"

Das XML-Dokument in Listing 2 enthält die zugehörige Probleminstanz und Lösung.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <!DOCTYPE Schlangenjagd SYSTEM "schlangenjagd.dtd" >
03 <Schlangenjagd>
04 <Zeit einheit="s">
05     <Vorgabe>60.0</Vorgabe>
06     <Abgabe>1.0</Abgabe>
07 </Zeit>
08 <Dschungel zeilen="2" spalten="4"
          zeichen="ABCDEFGHIJKLMNOPQRSTUVWXYZ">
09     <Feld id="F0" zeile="0" spalte="0" verwendbarkeit="1"
               punkte="1">F</Feld>
10     <Feld id="F1" zeile="0" spalte="1" verwendbarkeit="1"
               punkte="1">E</Feld>
11     <Feld id="F2" zeile="0" spalte="2" verwendbarkeit="1"
               punkte="1">R</Feld>
12     <Feld id="F3" zeile="0" spalte="3" verwendbarkeit="1"
               punkte="1">N</Feld>
13     <Feld id="F4" zeile="1" spalte="0" verwendbarkeit="1"
               punkte="1">X</Feld>
14     <Feld id="F5" zeile="1" spalte="1" verwendbarkeit="1"
               punkte="1">I</Feld>
15     <Feld id="F6" zeile="1" spalte="2" verwendbarkeit="1"
               punkte="1">N</Feld>
16     <Feld id="F7" zeile="1" spalte="3" verwendbarkeit="1"
               punkte="1">U</Feld>
17 </Dschungel>
18 <Schlangenarten>
19     <Schlangenart id="A0" punkte="1" anzahl="1">
20         <Zeichenkette>FERNUNI</Zeichenkette>
21         <Nachbarschaftsstruktur typ="Distanz">
22              <Parameter wert="1" />
23         </Nachbarschaftsstruktur>
24     </Schlangenart>
25 </Schlangenarten>
26 <Schlangen>
```

```
27      <Schlange art="A0">
28          <Schlangenglied feld="F0" />
29          <Schlangenglied feld="F1" />
30          <Schlangenglied feld="F2" />
31          <Schlangenglied feld="F3" />
32          <Schlangenglied feld="F7" />
33          <Schlangenglied feld="F6" />
34          <Schlangenglied feld="F5" />
35      </Schlange>
36 </Schlangen>
37 </Schlangenjagd>
```

Listing 2: Probleminstanz und Lösung als XML-Dokument

3.4.2 Lösungsverfahren

Der Lösungsraum kann in Abhängigkeit von der Probleminstanz sehr groß werden. Ein vollständiges Durchsuchen des Lösungsraums, d. h. ein Betrachten aller möglichen Lösungen, ist aus diesem Grund oft nicht mit vertretbarem Zeitaufwand möglich.

Implementieren Sie daher als Lösungsverfahren für die **Schlangensuche** einen auf Backtracking basierenden Algorithmus, der für eine gegebene Probleminstanz im Dschungel nach den angegebenen Schlangentypen sucht. Das übergeordnete Ziel ist die Maximierung der Gesamtpunktzahl. Das Lösungsverfahren muss unter Berücksichtigung der angegebenen Regeln auf beliebige Probleminstanzen anwendbar sein. Kann der Algorithmus innerhalb der angegebenen Bearbeitungszeit den vollständigen Lösungsraum nicht nach allen möglichen Lösungen absuchen, soll vorzeitig abgebrochen werden. Die beste zulässige Lösung, die bis zu diesem Zeitpunkt gefunden wurde, soll zurückgegeben werden.

Eine halbformale Beschreibung des zu entwickelnden Algorithmus ist in Listing 3 angegeben. Abweichende Implementierungen sind zulässig. Weitere Informationen und Beispiele zu Backtracking-Algorithmen sind in [1] zu finden.

```
01 METHODE sucheSchlange() {
02     WENN (aktuelle Punkte > bisher maximale Punkte) {
03         speichere Lösung
04     }
05     WENN (Zeitvorgabe erreicht) {
06         beende Suche und gebe Lösung zurück
07     }
08     erzeuge zulässige Startfelder
09     priorisiere und sortiere zulässige Startfelder
10     FÜR (Startfeld in zulässige Startfelder) {
11         bestimme zulässige Schlangenarten für Startfeld
12         priorisiere und sortiere zulässige Schlangenarten
13         FÜR (Schlangenart in Schlangenarten)
14             erzeuge neue Schlange mit Schlangenkopf
                       für Schlangenart
15             setze Schlangenkopf auf Startfeld
16             sucheSchlangenglied(Schlangenkopf)
17             entferne Schlangenkopf und Schlange
18         }
19     }
```

```
20 }
21
22 METHODE sucheSchlangenglied(vorherigesGlied) {
23     WENN (vorherigesGlied ist letztes Schlangenglied) {
24         sucheSchlange()
25         RUECKGABE
26     }
27     erzeuge zulässige Nachbarfelder für vorherigesGlied
28     priorisiere und sortiere zulässige Nachbarfelder
29     FÜR (Nachbarfeld in zulässige Nachbarfelder) {
30         erzeuge neues Schlangenglied
31         setze Schlangenglied auf Nachbarfeld
32         sucheSchlangenglied(Schlangenglied)
33         entferne Schlangenglied
34     }
35 }
```

Listing 3: Backtracking-Algorithmus für die Schlangensuche

Innerhalb des Algorithmus können an unterschiedlichen Stellen Entscheidungen über die weitere Suchrichtung bzw. Suchreihenfolge getroffen werden. Mögliche Suchrichtungen und deren Reihenfolge werden durch die Auswahl, Priorisierung und Sortierung von

- zulässigen Startfeldern,
- zulässigen Schlangenarten und
- zulässigen Nachbarfeldern

bestimmt. Für die Zulässigkeit müssen gegebenenfalls die Nachbarschaftsstruktur, die Verwendbarkeit von Feldern und die jeweiligen Zeichen beachtet werden. Die Priorisierung kann heuristisch über die Implementierung von Regeln erfolgen, die z. B. die Punkte und Verwendbarkeit von Schlangenarten bzw. Feldern berücksichtigen. Bei den im Programmierpraktikum vorgegebenen Probleminstanzen können Sie für den Entwurf einer solchen Heuristik davon ausgehen, dass die Punktzahl entweder für alle Felder gleich ist, oder für jedes Feld zufällig gewählt wurde. Bei zufälliger Wahl ist die Punktzahl die Realisierung einer diskret-gleichverteilten Zufallsvariable im Intervall $[0, 9]$, nimmt also mit gleicher Wahrscheinlichkeit die Werte $0, 1, \ldots, 9$ an.

Die Softwarearchitektur des Lösungsverfahrens soll dessen Anpassbarkeit gewährleisten, um verschiedene Regeln bzw. Heuristiken testen zu können. Wiederholung von gleichartigem Programmcode soll dabei vermieden werden. Entwerfen Sie eine entsprechende Softwarearchitektur unter Berücksichtigung der Prinzipien der objektorientierten Softwareentwicklung. Konzipieren Sie dazu entsprechende Schnittstellen oder abstrakte Klassen, deren Implementierung die Umsetzung der Heuristiken ermöglicht.

Bei der Entwicklung des Lösungsverfahrens ist darauf zu achten, dass die erzeugten Lösungen unter den Vorgaben der jeweiligen Probleminstanz zulässig sind. Das bedeutet insbesondere, dass eine Schlange die notwendige Anzahl an Schlangengliedern enthält und jedes Schlangenglied auf ein Feld verweist, das gemäß seiner Position in der Zeichenschlange ein entsprechendes Zeichen enthält. Aufeinanderfolgende Schlangenglieder müssen außerdem die durch die Schlangenart vorgegebene Nachbarschaftsstruktur einhalten. Schließlich dürfen

einzelne Felder durch alle in der Lösung enthaltenen Schlangen nicht häufiger als jeweils vorgegeben verwendet werden. Darüber hinaus ist das Lösungsverfahren hinsichtlich der in beschränkter Zeit erreichbaren Punkte zu optimieren.

Bei kleinen Probleminstanzen kann der Lösungsraum in vorgegebener Zeit vollständig untersucht werden. Der Algorithmus sollte in diesem Fall insbesondere hinsichtlich der Laufzeit so effizient wie möglich implementiert sein, so dass eine optimale Lösung in möglichst geringer Zeit gefunden wird. Dazu zählt beispielsweise, dass geeignete Datenstrukturen verwendet werden. Bei großen Probleminstanzen kann der Lösungsraum in beschränkter Zeit auch bei einer effizienten Implementierung in der Regel nicht vollständig untersucht werden. In diesem Fall muss die Suche in Richtung besonders aussichtsreicher Teile des Lösungsraums priorisiert werden. Bei Unterproblemen für die Priorisierung muss zwischen Verbesserung der Lösungsgüte und einer längeren Gesamtlaufzeit abgewogen werden.

3.4.3 Erzeugung von Probleminstanzen

Die Suche nach Schlangen setzt die Existenz von geeigneten Probleminstanzen voraus. Im Verlauf des Programmierpraktikums werden wir Ihnen Probleminstanzen bereitstellen, mit denen Sie Ihr Lösungsverfahren testen und Ihre Ergebnisse untereinander vergleichen können. Unabhängig davon sollen Sie selber einen **Dschungelgenerator** für die Erzeugung einfacher Probleminstanzen entwerfen und implementieren. Die Beschränkung auf einfache Probleminstanzen bedeutet, dass bei der geforderten Funktionalität zur Erzeugung von Schlangen Überschneidungen ausgeschlossen werden und jedes Feld maximal einmal verwendet werden darf. Für die Schlangensuche gilt diese Einschränkung nicht.

Bei der Erzeugung müssen Vorgaben über die enthaltenen Schlangenarten und Zeichen umgesetzt werden. Teil einer vorgegebenen Probleminstanz sind immer mindestens

- die Anzahl der Zeilen und Spalten des Dschungels,
- die für den Dschungel zulässigen Zeichen,
- eine oder mehrere Schlangenarten und
- die jeweils erwünschte Anzahl von Schlangen.

Fehlen Teile dieser Daten, ist eine Fehlermeldung auszugeben. Eine vollständige Probleminstanz enthält darüber hinaus auch den Vorgaben entsprechende Dschungelfelder und versteckte Schlangen. Fehlen Dschungelfelder, bezeichnen wir die Probleminstanz als unvollständig.

Bei der Erzeugung eines neuen Dschungels sind bereits vorhandene Dschungelfelder und eine dazu passende Lösung zunächst zu löschen bzw. folgend zu überschreiben. Existiert noch keine Vorgabe für ein Zeitlimit, so ist ein geeigneter Wert zu ergänzen. Ein XML-Dokument mit einer unvollständigen Probleminstanz wird beispielhaft in Listing 4 angegeben.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <!DOCTYPE Schlangenjagd SYSTEM "schlangenjagd.dtd" >
03 <Schlangenjagd>
04 <Dschungel zeilen="3" spalten="5"
           zeichen="ABCDEFGHIJKLMNOPQRSTUVWXYZ" />
```

```
05 <Schlangenarten>
06     <Schlangenart id="A0" anzahl="1">
07         <Zeichenkette>FERNUNI</Zeichenkette>
08         <Nachbarschaftsstruktur typ="Distanz">
09             <Parameter wert="1" />
10         </Nachbarschaftsstruktur>
11     </Schlangenart>
12     <Schlangenart id="A1" anzahl="1">
13         <Zeichenkette>HAGEN</Zeichenkette>
14         <Nachbarschaftsstruktur typ="Sprung">
15             <Parameter wert="2" />
16             <Parameter wert="1" />
17         </Nachbarschaftsstruktur>
18    </Schlangenart>
19 </Schlangenarten>
20 </Schlangenjagd>
```

Listing 4: Unvollständige Probleminstanz als XML-Dokument

Nach der Vervollständigung einer Probleminstanz mit Hilfe des Dschungelgenerators soll diese wieder als XML-Dokument gespeichert werden. Eine zu Listing 4 passende vollständige Probleminstanz wird in Listing 5 angegeben.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <!DOCTYPE Schlangenjagd SYSTEM "schlangenjagd.dtd">
03 <Schlangenjagd>
04 <Dschungel zeilen="3" spalten="5"
           zeichen="ABCDEFGHIJKLMNOPQRSTUVWXYZ">
05     <Feld id="F0" zeile="0" spalte="0" verwendbarkeit="1"
               punkte="1">R</Feld>
06     <Feld id="F1" zeile="0" spalte="1" verwendbarkeit="1"
               punkte="1">N</Feld>
07     <Feld id="F2" zeile="0" spalte="2" verwendbarkeit="1"
               punkte="1">R</Feld>
08     <Feld id="F3" zeile="0" spalte="3" verwendbarkeit="1"
               punkte="1">G</Feld>
09     <Feld id="F4" zeile="0" spalte="4" verwendbarkeit="1"
               punkte="1">F</Feld>
10     <Feld id="F5" zeile="1" spalte="0" verwendbarkeit="1"
               punkte="1">H</Feld>
11     <Feld id="F6" zeile="1" spalte="1" verwendbarkeit="1"
               punkte="1">U</Feld>
12     <Feld id="F7" zeile="1" spalte="2" verwendbarkeit="1"
               punkte="1">N</Feld>
13     <Feld id="F8" zeile="1" spalte="3" verwendbarkeit="1"
               punkte="1">E</Feld>
14     <Feld id="F9" zeile="1" spalte="4" verwendbarkeit="1"
               punkte="1">A</Feld>
15     <Feld id="F10" zeile="2" spalte="0" verwendbarkeit="1"
               punkte="1">N</Feld>
16     <Feld id="F11" zeile="2" spalte="1" verwendbarkeit="1"
               punkte="1">I</Feld>
17     <Feld id="F12" zeile="2" spalte="2" verwendbarkeit="1"
               punkte="1">A</Feld>
18     <Feld id="F13" zeile="2" spalte="3" verwendbarkeit="1"
               punkte="1">H</Feld>
```

```
19      <Feld id="F14" zeile="2" spalte="4" verwendbarkeit="1"
                punkte="1">E</Feld>
20  </Dschungel>
21  <Schlangenarten>
22      <Schlangenart id="A0" punkte="1" anzahl="1">
23          <Zeichenkette>FERNUNI</Zeichenkette>
24          <Nachbarschaftsstruktur typ="Distanz">
25              <Parameter wert="1" />
26          </Nachbarschaftsstruktur>
27      </Schlangenart>
28      <Schlangenart id="A1" punkte="1" anzahl="1">
29          <Zeichenkette>HAGEN</Zeichenkette>
30          <Nachbarschaftsstruktur typ="Sprung">
31              <Parameter wert="2" />
32              <Parameter wert="1" />
33          </Nachbarschaftsstruktur>
34      </Schlangenart>
35  </Schlangenarten>
36  </Schlangenjagd>
```

Listing 5: Vervollständigte Probleminstanz als XML-Dokument

Die Erzeugung eines Dschungels entsprechend der gegebenen Vorgaben erfolgt in zwei Schritten:

1. Zufällige Verteilung der Schlangen im Dschungel.
2. Belegung der verbleibenden Felder mit jeweils zufällig gewählten Zeichen.

Für die Verteilung der Schlangen im Dschungel kann ein Vorgehen ähnlich zu dem in Listing 3 skizzierten Backtracking-Algorithmus für die Schlangensuche angewendet werden. Nach Auswahl einer Schlangenart wird eine entsprechende Schlange im Dschungel angeordnet. Dieses Vorgehen wird solange wiederholt, bis die gewünschte Anzahl an Schlangen jeder Art erreicht ist. Das Startfeld für eine Schlange soll zufällig mit gleicher Wahrscheinlichkeit unter den bisher noch nicht belegten Feldern ausgewählt werden. Dasselbe gilt für die folgenden Schlangenglieder in der jeweils zulässigen Nachbarschaft. Während der gliedweisen Anordnung einer Schlange kann es vorkommen, dass keine zulässigen Nachbarfelder mehr gefunden werden. Dieser Fall tritt insbesondere dann auf, wenn schon viele Felder durch andere Schlangen belegt wurden. Einzelne Schlangenglieder oder auch vollständige Schlangen müssen dann zurückgenommen werden (Backtracking). Die dadurch frei gewordenen Felder stehen anschließend wieder für Schlangen in anderer Anordnung zur Verfügung.

Wenn alle Schlangenarten in der gewünschten Anzahl im Dschungel vorhanden sind, sollen die verbleibenden leeren Felder mit Zeichen belegt werden. Für jedes Feld ist zufällig ein Zeichen aus der vorgegebenen Zeichenmenge mit gleicher Wahrscheinlichkeit auszuwählen.

Da der Dschungelgenerator keine Überschneidungen von Schlangen vorsieht, soll die Verwendbarkeit aller Felder auf den Standardwert „1“ und die Punkte pro Feld auf den Standardwert „1“ gesetzt werden. Erweiterungen des Dschungelgenerators können beim Testen und Verbessern Ihres Lösungsverfahrens für die Schlangensuche hilfreich sein, werden aber später bei der Bewertung Ihrer Abgabe nicht berücksichtigt.

3.4.4 Zulässigkeit und Bewertung von Lösungen

Einmal erzeugte Lösungen müssen vor ihrer Bewertung auf ihre Zulässigkeit geprüft werden. Eine Lösung ist zulässig, wenn die folgenden vier Bedingungen erfüllt sind:

- Für jede Schlange stimmen die Anzahl an Schlangengliedern und die Anzahl an Zeichen in der Zeichenkette der jeweiligen Schlangenart überein.
- Für jedes Schlangenglied stimmen das Zeichen im zugeordneten Feld und das dem Schlangenglied zugeordnete Zeichen in der Zeichenkette der jeweiligen Schlangenart überein.
- Die Anzahl der einem Feld zugeordneten Schlangenglieder überschreitet nicht die Verwendbarkeit des Feldes.
- Die Beziehungen zwischen aufeinanderfolgenden Schlangengliedern aller Schlangen entsprechen der durch die jeweilige Schlangenart vorgegebenen Nachbarschafts-struktur.

Abbildung 9 zeigt beispielhaft Verstöße gegen die drei letztgenannten Bedingungen. Im Zeichendschungel wurden zwei Schlangenarten „SCHLANGE“ (orange) und „DSCHUNGEL“ (blau) mit Sprung(2,1)-Nachbarschaftsstruktur angeordnet. Felder mit Überschneidung und damit mehrfacher Verwendung sind grün gekennzeichnet, auftretende Fehler rot umrahmt.

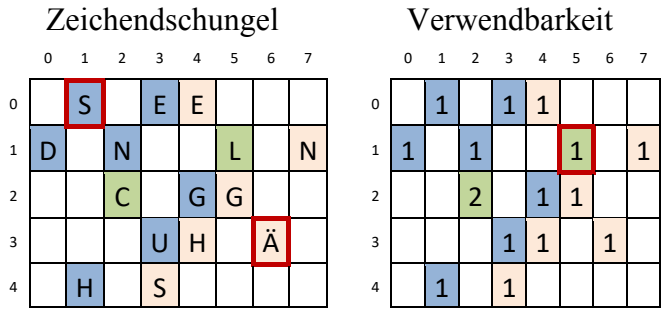


Abbildung 9: Mögliche Fehlerarten in einer Lösung

Das Feld (0,1) befindet sich nicht in der Nachbarschaft von Feld (1,0). Feld (1,5) wird von beiden Schlangen verwendet, hat jedoch nur eine Verwendbarkeit von eins. Feld (3,6) enthält das falsche Zeichen.

Bei der Prüfung auf Zulässigkeit sollen die Fehlerarten und die Häufigkeit des Auftretens bestimmt werden. Jede Schlange mit der falschen Anzahl an Schlangengliedern, jedes Schlangenglied mit dem falschen Zeichen, jedes Schlangenglied außerhalb der Nachbarschaft des vorangehenden Gliedes und jede Überschreitung der Verwendbarkeit eines Feldes zählen dabei als ein Fehler.

Bei der Bewertung einer Lösung werden die Punkte für alle gefundenen Schlangen und die Punkte für einzelnen Schlangengliedern zugeordneten Feldern addiert. Einzelne Felder können also je nach Verwendung mehrfach gezählt werden. Ein Beispiel für die Berechnung wurde bereits für die Lösung in Abbildung 6 gegeben.

3.4.5 Darstellung von Probleminstanzen und Lösungen

Die zuvor definierte XML-Repräsentation der Daten ermöglicht das standardisierte Einlesen und Speichern von Probleminstanzen und Lösungen. Für den menschlichen Betrachter ist dieses Format jedoch sehr unübersichtlich. Um einen schnellen Überblick über die Probleminstanzen und die gefundenen Lösungen zu erhalten, soll daher Funktionalität für die Darstellung in Form einer übersichtlichen Textausgabe implementiert werden.

Für die Probleminstanz sollen mindestens, falls vorhanden, folgende Daten auf die Konsole ausgegeben und übersichtlich dargestellt werden:

- Anzahl der Zeilen und Spalten des Dschungels sowie die verwendeten Zeichen,
- Schlangenarten mit Zeichenkette, Nachbarschaftsstruktur, Punkten und Anzahl,
- Dschungelfelder mit (1) Zeichen (2) Verwendbarkeit (3) Punkten.

Darüber hinaus soll die Lösung mit allen gefundenen Schlangen ausgegeben werden:

- Schlangenart mit Zeichenkette und Nachbarschaftsstruktur,
- Hervorhebung der im Zeichendschungel angeordneten Schlange,
- Zeile und Spalte der aufeinanderfolgenden Schlangenglieder.

Bei der Darstellung der Schlange im Dschungel können nicht verwendete Felder beispielsweise als Punkt repräsentiert werden, um die relevanten Felder mit Schlangengliedern hervorzuheben.

3.4.6 Hauptprogramm

Das Hauptprogramm wird über drei Kommandozeilenparameter gesteuert. Der erste Parameter (`ablauf`) steuert den Ablauf des Hauptprogramms und wird weiter unten genau spezifiziert. Der zweite Parameter (`eingabe`) identifiziert eine einzulesende XML-Datei. Die Datei muss eine Probleminstanz bzw. die zu deren Erzeugung notwendigen Parameter und optional eine Lösung der gegebenen Probleminstanz enthalten. Der dritte Parameter (`ausgabe`) gibt optional eine Ausgabedatei an, in die Probleminstanzen und Lösungen erneut im XML-Format gespeichert werden.

Für den Ablaufparameter `ablauf` wird folgende Festlegung getroffen:

- „`l`“ (lösen): Für eine gegebene Probleminstanz wird nach einer neuen Lösung gesucht (siehe Unterabschnitt 3.4.2) und bei Angabe einer Ausgabedatei gespeichert.
- „`e`“ (erzeugen): Eine neue Probleminstanz wird auf Basis der gegebenen Parameter erzeugt (siehe Unterabschnitt 3.4.3) und bei Angabe einer Ausgabedatei gespeichert.
- „`p`“ (prüfen): Die Zulässigkeit der gegebenen Lösung wird überprüft. Bei Unzulässigkeit werden die Art und Anzahl der verletzten Bedingungen in der Konsole ausgegeben (siehe Unterabschnitt 3.4.4).
- „`b`“ (bewerten): Die Gesamtpunktzahl der Lösung wird unabhängig von der Zulässigkeit berechnet und in der Konsole ausgegeben (siehe Unterabschnitt 3.4.4).
- „`d`“ (darstellen): Die Probleminstanz und die zugehörige Lösung werden in der Konsole dargestellt (siehe Unterabschnitt 3.4.5).

Die Parameterwerte sollen miteinander kombinierbar sein. Für die Parameterbelegung

```
ablauf=ldpb
```

sind die Funktionen zur Lösung der Probleminstanz („`l`“), der Darstellung von Probleminstanz und Lösung („`d`“), der Prüfung der Zulässigkeit („`p`“) sowie der Bewertung der Lösung („`b`“) in der vorgegebenen Reihenfolge auszuführen.

Für die Parameter `eingabe` und `ausgabe` wird durch einen String jeweils ein Dateipfad vorgegeben. Entspricht die eingelesene XML-Datei nicht den Vorgaben der DTD, wird das Programm mit einer Fehlermeldung beendet. Soll eine neue Probleminstanz oder Lösung ohne Angabe einer Ausgabedatei erzeugt werden, erfolgt ebenfalls eine Fehlermeldung.

Ein vollständiger Parameteraufruf kann wie folgt aussehen:

```
ablauf=ld eingabe=res\sj1.xml ausgabe=res\sj1_loesung.xml.
```

3.4.7 API-Modus

Im API-Modus soll die angebotene Funktionalität durch ein weiteres Programm aufgerufen und in dessen Adressraum ausgeführt werden können. Implementieren Sie dazu die in Listing 6 angegebene Schnittstelle *`SchlangenjagdAPI`* in der in Listing 7 angegebenen Einstiegsklasse *`Schlangenjagd`*. Beachten Sie, dass keine Änderungen an der Klassen- und Methodendefinition sowie der Paketzuordnung vorgenommen werden dürfen.

```
01 package de.fernuni.kurs01584.ss23.hauptkomponente;
02
03 import java.util.List;
04
05 public interface SchlangenjagdAPI {
06     public boolean loeseProbleminstanz(String xmlEingabeDatei,
               String xmlAusgabeDatei);
07     public boolean erzeugeProbleminstanz(String xmlEingabeDatei,
               String xmlAusgabeDatei);
08     public enum Fehlertyp {
09         GLIEDER, ZUORDNUNG, VERWENDUNG, NACHBARSCHAFT
10     }
11     public List<Fehlertyp> pruefeLoesung(String
               xmlEingabeDatei);
12     public int bewerteLoesung(String xmlEingabeDatei);
13
14     public String getName();
15     public String getMatrikelnummer();
16     public String getEmail();
17 }
```

Listing 6: Schnittstellenspezifikation der Hauptkomponente

```
01 package de.fernuni.kurs01584.ss23.hauptkomponente;
02
03 public class Schlangenjagd implements SchlangenjagdAPI {
04     // TODO: Implementierung von Schnittstelle und Programm-
                Einstieg
05 }
```

Listing 7: Einstiegsklasse der Hauptkomponente

Die Methode `boolean loeseProbleminstanz(String,String)` liest die vorgegebene Eingabedatei und startet das Lösungsverfahren für die Schlangensuche. Die gefundene Lösung wird in der Ausgabedatei gespeichert. Die Methode liefert den Wert `true`, wenn mindestens eine Schlange gefunden wurde und ansonsten `false`.

Die Methode `boolean erzeugeProbleminstanz(String,String)` liest die vorgegebene Eingabedatei, erzeugt eine neue Probleminstanz auf Basis der gegebenen Parameter und speichert die Probleminstanz in der vorgegebenen Ausgabedatei. Bei Erfolg wird der Wert `true` und ansonsten `false` zurückgegeben.

Die Felder der Aufzählung (Enumeration) `Fehlertyp` repräsentieren jeweils eine Verletzung der in Unterabschnitt 3.4.4 gegebenen Bedingungen für die Zulässigkeit einer Lösung:

- `GLIEDER`: Eine Schlange besteht nicht aus der richtigen Anzahl von Schlangengliedern.
- `ZEICHEN`: Ein Schlangenglied ist einem Feld mit einem falschen Zeichen zugeordnet.
- `VERWENDUNG`: Ein Schlangenglied ist einem bereits maximal verwendeten Feld zugeordnet.
- `NACHBARSCHAFT`: Ein Schlangenglied befindet sich nicht in der Nachbarschaft des jeweils vorherigen Schlangengliedes.

Die Methode `List<Fehlertyp> pruefeLoesung(String)` liest die Probleminstanz und Lösung aus der gegebenen Datei ein und überprüft die Lösung auf Zulässigkeit. Dabei werden sowohl die Art als auch die Häufigkeit der verletzten Bedingungen ermittelt. Als Rückgabewert liefert die Methode eine Liste der gefundenen Einzelfehler.

Die Methode `int bewerteLoesung(String)` liest die Probleminstanz und Lösung aus der gegebenen Datei ein und berechnet die erreichte Punktzahl. Die Berechnung erfolgt unabhängig von der Zulässigkeit der Lösung.

Implementieren Sie schließlich die Methoden `getName()`, `getMatrikelnummer()` und `getEmail()` so, dass Ihre persönlichen Daten zurückgegeben werden.

3.4.8 Testbarkeit

Bei der Implementierung von Datenmodell und Lösungsverfahren sind Methoden bzw. Algorithmen zur Lösung von unterschiedlichen Unterproblemen zu entwickeln. Einfache Unterprobleme entstehen beispielsweise bei der Bestimmung von zulässigen Start- oder Nachbarfeldern. Kapseln Sie die Lösung von Unterproblemen in separaten Methoden bzw. Klassen. Bei der Kapselung ist dafür zu sorgen, dass die Lösung von Unterproblemen individuell getestet werden kann.

Entwickeln Sie für alle auftretenden Unterprobleme geeignete Testklassen, durch welche die Korrektheit der implementierten Methoden sichergestellt wird. Die Testklassen müssen auf Basis von Unit Tests entwickelt werden. Achten Sie darauf, dass Ihre Implementierung alle Testfälle besteht.

3.5 Empfohlenes Vorgehen

Einige Schritte der Problemanalyse und Modellierung wurden in Abschnitt 3.4 bereits vorweggenommen, um Ihnen einen schnelleren Einstieg in die Aufgabenstellung des Programmierpraktikums zu ermöglichen. Für Ihre eigene Implementierung kann es sinnvoll oder sogar notwendig sein, von der vorgeschlagenen Strukturierung der Daten in Abbildung 7 oder dem Ablauf des Lösungsalgorithmus in Listing 3 abzuweichen. Damit bieten sich Ihnen viele Freiheitsgrade hinsichtlich der Optimierung von Laufzeit und Speicherbedarf sowie der im Mittel erreichbaren Lösungsgüte bei großen Probleminstanzen. Dabei sollten Sie aber immer auf die Korrektheit Ihres Lösungsverfahrens und die Einhaltung der nach außen hin sichtbaren Schnittstellen achten. Dazu gehören insbesondere die DTD für die XML-Dokumente, in denen Probleminstanzen und Lösungen bereitgestellt und gespeichert werden, sowie die Kommandozeilenparameter und API-Methoden. Wir empfehlen Ihnen das folgende Vorgehen:

1. Versuchen Sie zunächst eine gegebene Probleminstanz (z. B. Abbildung 2) manuell zu lösen. Denken Sie über Ihr Vorgehen bei der Identifizierung möglicher Startpositionen für die Suche nach Schlangen der vorgegebenen Arten nach. Überlegen Sie weiter, wie sich ausgehend von diesen Startpositionen die Felder für nachfolgende Zeichen in einer Zeichenschlange auffinden lassen. Unterteilen Sie Ihre Vorgehensweise dabei in möglichst kleinteilige Einzeloperationen, die sich später in Programmcode abbilden lassen. Notieren Sie sich mögliche Klassen und Attribute, die dabei berücksichtigt werden müssen.
2. Entwerfen Sie ein geeignetes Datenmodell in Form eines UML-Klassendiagramms. Vergleichen Sie Ihr Modell mit der Struktur, die in Abbildung 7 vorgeschlagen wird. Konkretisieren Sie dabei die Typen der Attribute und die Art der Beziehung von Klassen bzw. daraus abgeleiteten Objekten untereinander. Implementieren Sie das Datenmodell anschließend in Java.
3. Konzipieren Sie anschließend den Ablauf des Lösungsverfahrens. Identifizieren Sie dafür die zu lösenden Unterprobleme. Skizzieren Sie bereits in dieser Phase die Lösung von auftretenden Unterproblemen. Achten Sie darauf, dass das Datenmodell eine effiziente Lösung der Unterprobleme unterstützt.
4. Implementieren Sie die Eingabe- und Ausgabekomponenten, durch welche die XML-Dateien in den beschriebenen Formaten eingelesen und ausgegeben werden können. Beim Lesen einer Eingabedatei sind einerseits die eingelesenen Informationen in das Datenmodell Ihres Programms zu überführen. Andererseits muss die Objektstruktur des Datenmodells für die Übertragung in das Ausgabeformat geeignet aufbereitet werden.
5. Realisieren Sie die textbasierte Ausgabe von Probleminstanzen und Lösungen. Testen Sie die Korrektheit der Ausgaben anhand der zur Verfügung gestellten Beispielinstanzen.
6. Implementieren Sie die Algorithmen für die zu lösenden Unterprobleme. Die dazu erforderlichen Testfälle sollen bereits während der Implementierung der Algorithmen zur Verfügung stehen. Achten Sie bei der Entwicklung der Algorithmen unbedingt darauf, dass diese effizient ausgeführt werden können.

7. Entwickeln Sie den Backtracking-Algorithmus. Analog zur Implementierung der Algorithmen zur Lösung der Unterprobleme ist auch der Backtracking-Algorithmus so effizient wie möglich zu implementieren.

4. Hinweise zur Durchführung

Im Folgenden werden Vorgaben und Hinweise zur erfolgreichen Erstellung des Programms beschrieben. Außerdem werden Rahmenbedingungen, die bei der Implementierung zu beachten sind, dargestellt.

4.1 Allgemeine Vorgaben

In der Konzeption, der Programmierung und der Durchführung sind folgende Systeme und Bibliotheken in den genannten Versionen explizit erlaubt:

- Runtime: Java Platform, Standard Edition (OpenJDK) in Version 19 [2]
- Entwicklungsumgebung: Eclipse IDE 2023-03
- XML Parser: JDOM in Version 2.0.6.1
- Unit Test Framework: JUnit in Version 5.9.2.

Die Nutzung anderer fremder Klassenbibliotheken ist nicht zulässig. Alle nötigen Bibliotheken werden in einem Musterprojekt zur Verfügung gestellt. Verwenden Sie zur Implementierung Ihrer Lösung ausschließlich dieses Musterprojekt. Im Musterprojekt sind bereits Pakete und Klassen vorgegeben, die beim Softwareentwurf als Leitlinien dienen.

Bevor Sie die Anwendung zur Bewertung abgeben, sind Integrationstests durchzuführen. Das bedeutet, dass Sie zusätzlich zu den Unit Tests komponentenübergreifend testen müssen. In der einfachsten Form kann das manuell erfolgen, indem Sie die geforderten Funktionen einzeln überprüfen. Geben Sie Ihr Programm erst dann ab, wenn Sie sicher sind, alle Anforderungen erfüllt zu haben.

Es ist ebenfalls darauf zu achten, dass das Programm unabhängig vom Ort der Ablage auf der Festplatte geöffnet und kompiliert werden kann. Kopieren Sie dazu vor der Abgabe das komplette Projekt an eine andere Stelle der Festplatte Ihres Computers. Versuchen Sie nun, die Software zu öffnen und zu kompilieren. Führen Sie dann alle Unit Tests erneut aus.

4.2 Dokumentation und Richtlinien

Die Dokumentation des Programms erfolgt im Programmcode mittels Javadoc. Ein Tutorial zu Javadoc ist in [3] zu finden. Kommentare sollen einen fachlichen, logischen und konzeptionellen Charakter haben. Es ist zwingend erforderlich, alle öffentlichen Klassen, Schnittstellen, deren Parameter und Rückgabewerte sowie alle öffentlichen Variablen von Klassen und Schnittstellen zu kommentieren. Es ist ebenfalls notwendig, bei der Implementierung komplexer Logik, wie beispielsweise bei der Implementierung des Backtracking-Algorithmus, im Programmcode Java-Kommentare in Bezug auf das logische Vorgehen einzufügen. Das ermöglicht Dritten ein einfacheres Verständnis, hilft beim Nachvollziehen der Überlegungen des Entwicklers und vermeidet so Missverständnisse.

Analog zum eigentlich Programm sind alle Testfälle so zu kommentieren, dass klar erkennbar ist, wofür der jeweilige Testfall verwendet werden kann.

4.3 Komponentenentwurf

Im Programmierpraktikum sollen Sie ein Programm entwickeln, das unter anderem ein Lösungsverfahren für die Suche nach Zeichenschlangen in einem Zeichendschungel anbietet. Das Programm sollte so entwickelt werden, dass es objektorientierten Standards genügt. Das bedeutet, dass das Programm aus mehreren entkoppelten Komponenten besteht, die in definierten Beziehungen zueinander stehen.

Das Programm besteht typischerweise aus den folgenden Komponenten:

- Hauptkomponente: Ermöglicht über die Kommandozeile oder die Programmierschnittstelle (API) das Starten der Anwendung und den Aufruf der angebotenen Funktionalität.
- Dateiverarbeitungskomponente: Ermöglicht das Lesen und Schreiben von XML-Dateien sowie die Datenbereitstellung.
- Algorithmenkomponente: Beinhaltet die Algorithmen zur Erzeugung von Probleminstanzen und zu deren Lösung.
- Modellkomponente: Bildet die Daten für Probleminstanzen und zugehörigen Lösungen in geeigneten Strukturen ab und bietet Funktionen zur Datenverarbeitung und -modifikation an.
- Darstellungskomponente: Ermöglicht die textbasierte Ausgabe von Probleminstanzen und Lösungen.
- Testkomponente: Enthält die einzelnen Testfälle.

Grundsätzlich ist zu empfehlen, vor der Programmierung einen groben Architekturentwurf zu erstellen. Das bedeutet, dass Sie die einzelnen Komponenten zum Beispiel unter Verwendung von UML zeichnen. Stellen Sie auch die Beziehungen der einzelnen Komponenten zueinander dar. Anschließend definieren Sie die Funktionen und Eigenschaften jeder Komponente. Beginnen Sie erst dann mit der Implementierung, wenn Sie alle Funktionen des Systems einer Komponente zugeordnet haben. Es ist sinnvoll, zunächst für jede Komponente ein separates Paket in Eclipse anzulegen. Die Testkomponente stellt einen Sonderfall dar. Sie wird in der Ordnerstruktur des Projekts unter `test` von den übrigen Komponenten unter `src` getrennt. Komponententests werden den Paketen der zu testenden Komponenten zugeordnet. Nachdem Sie das durchgeführt haben, importieren Sie benötigte Pakete und enthaltene Klassen, wie Sie es in der Architektur vorgesehen haben. Dadurch wird das Programm strukturierter, so dass auch bei fortschreitender Implementierung die Komponententrennung und der Überblick bestehen bleibt. Um Ihnen den Einstieg zu erleichtern, sind die Komponenten in Form von Paketen zusammen mit einigen Klassengerüsten und Importbeziehungen bereits im Musterprojekt vorhanden.

4.3.1 Hauptkomponente

Die Hauptkomponente soll den zentralen Einstiegspunkt der Anwendung realisieren. Die Main-Methode des Hauptprogramms ist über drei Kommandozeilenparameter zu steuern. Der erste Parameter steuert den Ablauf des Hauptprogramms. Die weiteren Parameter geben die Pfade

zu Ein- und Ausgabedateien in Form von XML-Dokumenten an. Eine detaillierte Beschreibung der einzelnen Parameter ist in Unterabschnitt 3.4.6 zu finden.

Falls ein Fehler bei der Parametereingabe auftritt, ist das dem Anwender durch eine entsprechende Fehlermeldung zu signalisieren. Bei der Fehlermeldung ist keine technische Meldung auszugeben, sondern der Benutzer so detailliert wie möglich über den Fehler zu informieren. Guter Stil ist es, bei der Ausgabe von Fehlertexten dem Benutzer mögliche Lösungen anzubieten (z. B. „Bitte geben Sie eine Eingabedatei mit Hilfe des Parameters eingabe=BeispielDatei.xml an.“).

4.3.2 Dateiverarbeitung

Die Dateiverarbeitungskomponente ist für die Dateneingabe und Datenausgabe verantwortlich. Sie hat die Aufgabe, die XML-Eingabedatei einzulesen und anschließend entsprechend des Datenmodells in eine Objektstruktur abzubilden. Komponenten, welche Daten aus einer bestimmten Quelle nutzen wollen, müssen mit dem spezifischen Format der Datenquelle nicht vertraut sein. Ihnen werden die Daten im Format der Modellkomponente bereitgestellt. Durch diese Entkopplung kann später problemlos die Dateiverarbeitung ausgetauscht werden, ohne dass andere Komponenten ihren Programmcode anpassen müssen. Im Programmierpraktikum erfolgt die Dateneingabe und -ausgabe im XML-Dateiformat (siehe Unterabschnitt 3.4.1).

Die Dateiverarbeitungskomponente muss ebenfalls dafür sorgen, dass eine durch das Programm ermittelte Lösung und die dazugehörige Probleminstanz zurück in eine angegebene XML-Datei geschrieben werden.

Die im Listing 1 gezeigte DTD definiert die XML-Dokumentstruktur, in der die Probleminstanzen eingelesen bzw. die Ergebnisse ausgegeben werden. Verwenden Sie zur Implementierung der Dateneingabe- und Datenausgabefunktionalität entweder den DOM-Parser des javax.xml Pakets oder die im Musterprojekt bereits integrierte JDOM2-Bibliothek [4].

4.3.3 Algorithmenentwicklung

Implementieren Sie für die Nutzung der Algorithmenkomponente eindeutige Klassen und Methoden, die anderen Komponenten den Aufruf der angebotenen Funktionalität ermöglichen. Für die Austauschbarkeit von Algorithmen mit gleicher Funktionalität ist die Definition von Schnittstellen sinnvoll, die von funktionsgleichen Klassen implementiert werden. Dem für die Schlangenjagd verwendeten Suchalgorithmus sollen die zu lösende Probleminstanz und weitere Konfigurationsparameter übergeben werden. Die Rückgabe enthält dann die gefundene Lösung und gegebenenfalls weitere Informationen zu deren Beschreibung. Nutzen Sie zur Repräsentation der Probleminstanz und deren Lösung das Datenmodell der Modellkomponente. Über die Konfigurationsparameter können Heuristiken für die Lösung möglicher Unterprobleme und ein Abbruchkriterium vorgegeben werden.

Entwickeln Sie für alle auftretenden Unterprobleme geeignete Testklassen, durch die die Korrektheit der implementierten Methoden möglichst umfassend sichergestellt wird. Achten Sie darauf, dass Ihre Implementierung alle von Ihnen definierten Testfälle besteht.

Beachten Sie, dass das Lösungsverfahren so effizient wie möglich zu implementieren ist. Dazu zählt beispielsweise das Verwenden von geeigneten Datenstrukturen im Datenmodell sowie die speicher- und zeitschonende Konzeption und Implementierung der Algorithmen.

4.3.4 Darstellungskomponente

Die Darstellungskomponente spielt für die gegebene Aufgabenstellung nur eine untergeordnete Rolle. Sie dient primär der übersichtlichen Ausgabe von Probleminstanzen und Lösungen. Dennoch ist es für die Trennung von Verantwortlichkeiten, spätere Erweiterungen und alternative Benutzeroberflächen sinnvoll, diese Funktionalität in einer eigenen Komponente abzubilden.

4.3.5 Testbarkeit

In der Testkomponente werden die Tests für die einzelnen Komponenten implementiert. Bei der Implementierung des Lösungsverfahrens sind Algorithmen zur Lösung von Unterproblemen zu entwickeln. Kapseln Sie die Lösung von Unterproblemen in separate Methoden, Klassen und Schnittstellen. Bei der Kapselung muss sichergestellt werden, dass die Lösungen der Unterprobleme individuell getestet werden können.

Die einzelnen Tests werden typischerweise in Testklassen entwickelt. Jede Testklasse gehört immer zu einer fachlichen bzw. thematischen Einheit (Unit). Die Implementierung ist auf Basis des JUnit-Frameworks [5] durchzuführen. Die zugehörigen Bibliotheken sind bereits im Musterprojekt eingebunden.

Im Musterprojekt sind in der Testkomponente bereits einige Tests implementiert. Nutzen Sie die implementierten Tests, um Ihr Programm zu testen. Auf Basis der bereits enthaltenen Tests sind weitere Tests zu entwickeln. Mit zunehmender Größe der Anwendung können Sie so sicherstellen, dass sich stets alle Funktionen wie gewünscht verhalten.

4.4 Empfohlenes Vorgehen

Das in Abschnitt 3.5 beschriebene Vorgehen wird nachfolgend um technische Aspekte erweitert:

1. Beginnen Sie zunächst mit dem Softwareentwurf. Zeichnen Sie sich dafür die in Abschnitt 4.3 beschriebenen Komponenten und deren Beziehungen auf. Danach ordnen Sie die Funktionen des Programms den einzelnen Komponenten zu. Vernachlässigen Sie dabei zunächst die technischen Rahmenbedingungen.
2. Konzipieren und entwerfen Sie ebenfalls die verschiedenen Anwendungsfälle des Programms. Dazu gehören beispielsweise das Ermitteln, die Bewertung sowie das Anzeigen einer Lösung. Zeichnen Sie sich die einzelnen Anwendungsfälle auf, um dadurch ein Gefühl für die Komplexität und den Ablauf der einzelnen Schritte (Funktionen) zu bekommen. Als nächstes ordnen Sie die einzelnen Schritte den zuvor definierten Komponenten zu.
3. Konzipieren Sie ebenfalls vor der Implementierung mit Hilfe der Anwendungsfälle den Ablauf des Lösungsverfahrens. Identifizieren Sie dabei die zu lösenden Probleme bzw. Unterprobleme. Denken Sie bereits jetzt über mögliche algorithmische Lösungen nach.

Entwerfen Sie aufbauend auf diesen Überlegungen eine geeignete Architektur, die insbesondere den Anforderungen an Anpassbarkeit und Entkopplung genügt.

4. Zusätzlich zur Anwendungsfallkonzeption ist nun die technische Konzeption der Komponenten durchzuführen. Dazu gehören die Spezifikation von Schnittstellen für verwendete Heuristiken sowie der einzelnen Komponenten. Die Schnittstellen müssen zu diesem Zeitpunkt noch nicht endgültig festgelegt werden. Achten Sie darauf, die zuvor definierten Beziehungen zwischen den einzelnen Komponenten nicht zu verletzen.
5. Laden Sie das im Kursportal zur Verfügung gestellte Musterprojekt in die Entwicklungsumgebung. Im Projekt sind bereits die einzelnen Komponenten mit rudimentären Beziehungen zueinander eingefügt.
6. Bevor Sie Logik implementieren, sind zunächst einzelne Testfälle mit Hilfe des JUnit-Frameworks zu erstellen. Das hilft Ihnen, gezielt nur die Funktionen zu prüfen, an denen Sie gerade arbeiten. Außerdem werden dadurch Seiteneffekte durch andere Komponenten vermieden. Es ist nicht erforderlich, für jede Testinstanz alle nötigen Schritte des Anwendungsfalls durchzuführen. Lediglich die sich gerade in Entwicklung befindliche Funktionalität kann getestet werden. Zu Beginn erscheint dieses Vorgehen umständlich. Mit fortschreitender Entwicklung werden die Vorteile eines solchen Vorgehens sichtbar. Wichtig wird dieses Vorgehen insbesondere dann, wenn Sie Verbesserungen an bestehendem Programmcode vornehmen. Durch die Testfälle können Sie prüfen, ob die Logik der Systembausteine nach der Änderung identisch geblieben ist.
7. Implementieren Sie die Eingabe- und Ausgabefunktionen in der Datenverarbeitungskomponente. Beim Lesen einer Eingabedatei sind einerseits die eingelesenen Informationen in das Datenmodell zu überführen, andererseits müssen die im Modell abgebildeten Daten geeignet aufbereitet werden, um diese in das geforderte Ausgabeformat zu transformieren. Es ist sinnvoll, mit dem Datenmodell zu starten, um bereits erste Testdaten in der weiteren Entwicklung nutzen zu können.
8. Implementieren Sie die Methoden- bzw. Klassenrümpfe der zuvor identifizierten einzelnen Anwendungsfälle. Implementieren Sie ebenfalls die Aufrufreihenfolge der Methoden in Ihrem Programmcode bzw. in den einzelnen Komponenten des Programms. Alle Funktionen der einzelnen Anwendungsfälle sollen am Ende als Rumpf in Ihrem Programm implementiert sein. Die zuvor implementierten Datenklassen können so bereits in der Schnittstellenimplementierung verwendet werden. Nutzen und erweitern Sie während der Entwicklung kontinuierlich die zuvor implementierten Tests zur Überprüfung der gerade implementierten Validierungslogik.
9. Entwickeln Sie den Backtracking-Algorithmus. Beachten Sie, dass das Verfahren beendet wird, nachdem der Lösungsraum vollständig untersucht oder die vorgegebene Laufzeit überschritten wird. Achten Sie bei der Entwicklung der Algorithmen unbedingt darauf, dass diese effizient ausgeführt werden können. Der Zeitaufwand wird durch die Größe des untersuchten Lösungsraums bestimmt. Daher kann eine heuristische Bestimmung der weiteren Suchrichtung bzw. -reihenfolge notwendig sein. Schauen Sie sich zunächst den in Unterabschnitt 3.4.2 angegebenen Pseudocode an, bevor Sie mit der Programmierung beginnen. Überlegen Sie, wie Sie die einzelnen Schritte umsetzen können. Beginnen Sie erst mit der Implementierung, nachdem Sie den Algorithmus verstanden haben. Denken Sie bei der Implementierung stets an die Komponententrennung und an die

Austauschbarkeit der einzelnen Teilfunktionen des Algorithmus. Verwenden Sie dazu Schnittstellen oder abstrakte Klassen. Es empfiehlt sich, zunächst einige Testfälle für den Algorithmus zu entwickeln.

10. Nachdem der Algorithmus implementiert wurde, können Sie diesen aus der Hauptkomponente aufrufen, um Probleminstanzen zu lösen, und übergebene Lösungen auf Zulässigkeit zu überprüfen.
11. Im vorletzten Schritt müssen Sie alle zuvor getrennt entwickelten Komponenten integrieren. Dazu implementieren Sie die Anwendungsfälle vollständig. Achten Sie dabei unbedingt auf die Entkopplung der Komponenten.
12. Im letzten Schritt sind die Anforderungen der Hauptkomponente wie z. B. Konsoleneingabe mit Dateiübergabe zu implementieren und die zuvor implementierten und getesteten Anwendungsfälle anzubinden.

Führen Sie intensive Integrationstests durch. Korrigieren Sie dabei auftretende Fehler in den jeweiligen Komponenten. Falls Sie während der Entwicklung auf Fehler stoßen, sind direkt entsprechende Tests zu implementieren.

5. Ausblick

Der für die Schlangenjagd entwickelte Suchalgorithmus und das zugehörige Datenmodell lassen sich auf verschiedene andere Problemstellungen übertragen. Betrachten Sie dafür die beiden folgenden Beispiele.

Produktionsplanung: Für eine Fabrik soll ein Plan für die Herstellung verschiedener Produkte erstellt werden. Die Fabrikhalle besteht aus regelmäßig angeordneten Arbeitsplätzen, so dass ein Arbeitsplatz über seine Reihe und Position innerhalb dieser Reihe eindeutig identifiziert werden kann. An jedem Arbeitsplatz kann ein bestimmter Arbeitsschritt zu vorgegebenen Kosten durchgeführt werden. Zudem liegen Informationen über die jeweilige Auslastung und damit freie Kapazitäten vor. Die vorgegebenen Produkte können zu einem bestimmten Preis am Markt verkauft werden. Die Herstellung der Güter erfolgt nach einem bestimmten Arbeitsplan, bestehend aus einer Menge von aufeinanderfolgenden Arbeitsgängen. Zudem gibt es Vorgaben für den Transport zwischen einzelnen Arbeitsgängen bzw. Arbeitsplätzen. Unter dem Ziel der Gewinnmaximierung soll ein Produktionsplan aufgestellt werden, der herzustellende Produkte und zugehörige Mengen mit einer Zuordnung von Arbeitsgängen zu Arbeitsplätzen bestimmt.

Rundreise: Ausgehend von einem Startort ist eine Rundreise durch verschiedene Städte geplant. Die Distanzen zwischen den einzelnen Städten sind bekannt. Ziel ist es, eine Städtereihenfolge zu finden, bei der jede Stadt genau einmal besucht und die dabei zurückgelegte Gesamtdistanz minimiert wird.

Beide Probleme lassen sich im Datenmodell der Modellkomponente abbilden. Tabelle 2 stellt die Zusammenhänge zwischen den Problemen dar.

Tabelle 2: Abbildung verschiedener Probleme in einem verallgemeinerten Datenmodell

Schlangenjagd	Produktionsplanung	Rundreise
Zeile, Spalte	Reihe, Position	Städte
Dschungel	Fabrikhalle	Distanzmatrix
Feld	Arbeitsplatz	Verbindungsstrecke
Verwendbarkeit	Kapazität	-
Punkte	Variable Kosten	Distanz
Schlangenart	Produkt	-
Schlange	Arbeitsplan	Reiseplan
Schlangenglied	Arbeitsgang	Etappe
Nachbarschaftsstruktur	Transportbeziehung	Adjazente Verbindungs-strecken

Ein generischer und passend konfigurierter Suchalgorithmus kann Lösungen für alle drei genannten Probleme erzeugen. Bei der Konfiguration können Heuristiken für die Lösung von Teilproblemen bzw. Zielkriterien für die Steuerung der Suche vorgegeben werden. Spezialisierte Lösungsansätze für ähnliche Probleme werden in vertiefenden Lehrveranstaltungen des Lehrgebiets Unternehmensweite Softwaresysteme behandelt.

6. Literatur

[1] Saake, G., Sattler, K.-U. 2010. Algorithmen und Datenstrukturen: Eine Einführung mit Java. 4. Auflage, dpunkt.verlag, Heidelberg.

[2] Java Platform, Standard Edition & Java Development Kit Version 19 API Spezifikation. https://docs.oracle.com/en/java/javase/19/docs/api/index.html.

[3] Javadoc-Tutorial. 2023. https://www.oracle.com/technical-resources/articles/java/java-doc-tool.html.

[4] JDOM2-Bibliothek. 2023. http://www.jdom.org/.

[5] JUnit-Framework. 2023. https://junit.org/junit5/.