Dokumentation der Praktischen Arbeit zur Prüfung zum Mathematisch-technischen Softwareentwickler

23. Mai 2019

Jakob Rockenbauch

Prüfungs-Nummer: 30 6511 142 18688

Programmiersprache: golang

Inhaltsverzeichnis

1. Aufgabenanalyse

1.1. Analyse

In der Aufgabe geht es darum schemenhafte Karten auf Basis von vorgegebenen Kennwerten zu erstellen. Dabei werden Staaten als Kreise dargestellt und die Fläche der Kreise ist proportional zum Kennwert. Die Schwierigkeit liegt hierbei eine möglichst überschneidungsfreie Darstellung zu finden und gleichzeitig die urspünglichen Lage- und Nachbarschaftsbeziehungen möglichst beizubehalten.

1.2. Eingabeformat

Das Programm erwartet den Pfad zu einer Datei oder Verzeichniss enthält als Kommandozeilenparameter. Falls der übergebene Pfad ein Verzeichniss ist werden alle Dateien in diesem Verzeichniss die mit .txt enden als Eingaben verwendet und hintereinander abgearbeitet. Falls nur eine Datei übergeben wird muss sie mit .txt enden.

Die Eingabedateien werden nach folgenden Formatsvorgaben eingelesen. Bei Abweichungen von der Vorgabe ist das Verhalten undefiniert oder eine Fehlermeldung wird ausgegeben.

1.2.1. Formatsforgaben

Kommentare

Alle Zeilen die mit einem oder mehreren # Zeichen beginnen werden als Kommentare ignoriert.

Titel

Die erste Zeile wird als Titel des Kennwertes gepeichert.

Staaten

Die darauf folgenden Zeilen beschreiben die Staaten die in der Karte dargestellt werden. Sie müssen aus einer ID, einem Kennwert, einem Längengrad und einem Breitengrad bestehen. Die einzelnen Werte müssen durch eine beliebige Anzahl von Whitespace getrennt sein. Die ID muss ein string sein, der Kennwert eine ganze positive Zahl und Längengrad und Breitengrad eine Fließkommazahl. Es werden so lange Staaten eingelesen bis eine Zeile ein : Symbol enthält.

Nachbarschaften

Alle weiteren Zeilen müssen mit der ID eines Staates gefolgt von einem Doppelpunkt(:), gefolgt von einer Liste von IDs bestehen. Diese Zeilen bilden die Nachbarschaftsbeziehungen der Staaten ab. Nachbarschaften sind symetrisch das heißt wenn z.B. NL ein Nachbar von D ist auch D ein Nachbar von NL. Es reicht jedoch wenn in der Eingabe die Nachbarschaft nur in eine richtung enthalten ist. Die Rückrichtung gilt dann automatisch auch.

1.2.2. Beispiel

```
Bierkonsum
# Staat Bierkonsum Laengengrad Breitengrad
D   8692     10.0     51.3
NL  1156      5.3     52.2
B    781      4.8     50.7
L     80      6.1     49.8
F   2077      2.8     47.4
CH   440      8.2     46.9
A    945     14.2     47.6
CZ  1573     15.3     49.8
PL  3724     18.9     52.2
DK   360      9.6     56.0
# Nachbarschaften
D: NL B L F CH A CZ PL DK
NL: B
B: L F
F: CH
CH: A
A: CZ
CZ: PL
```

1.3. Ausgabeformat

Die Ausgabe erfolgt in eine Datei oder meherere Dateien. Die Ausgabe ist speziell auf das Programm gnuplot ab Version 5 zugeschnitten. Diese Dateien haben den gleichen Namen wie die Eingabedateien doch eine .gnu Endung anstatt der .txt Endung.

1.3.1. Beschreibung

Das Format der Ausgabedateien ist wie folgt:

```
reset
set xrange [<xmin>:<xmax>]
set yrange [<ymin>:<ymax>]
set size ratio 1.0
set title "<titel>, Iteration: <nr>"
unset xtics
unset ytics
$data << EOD
<Liste aus <xpos> <ypos> <radius> <ID> <numericID>>
EOD
plot \
'$data' using 1:2:3:5 with circles lc var notitle, \
'$data' using 1:2:4:5 with labels font "arial,9" tc var notitle
```

Dabei werden die <Tags> wie folgt ersetzt.

Tag	Inhalt
<xmin>	Kleinster X-Wert im Darstellungsbereich
<xmax>	Größter X-Wert im Darstellungsbereich
<ymin>	Kleinster Y-Wert im Darstellungsbereich
<ymax>	Größter Y-Wert im Darstellungsbereich
<titel>	Der eingelesene Titel
<nr>	Anzahl der Iterationen
<Liste aus ....>	Liste aus Staaten, je Zeile ein Staat.
<xpos>	X-Koordinate des Kreismittelpunktes
<ypos>	Y-Koordinate des Kreismittelpunktes
<ID>	ID des Staates
<numericID>	Forlaufende numerische ID (dient zur Färbung)

1.3.2. Beispiel

```
reset
set xrange [-33.86357381155017:85.07416473609788]
```

```
set yrange [-2.173515497514847:116.76422305013321]
set size ratio 1.0
set title "Bierkonsum, Iteration: 1000"
unset xtics
unset ytics
$data << EOD
16.75122941552248 63.89562847551619 29.676315947556564 D 0
-23.041037681301283 71.42772638479772 10.822536130248883 NL 1
-20.220018532838196 51.912423031589974 8.895608153727649 B 2
-10.217085354402032 46.657852455652204 2.8470501736687086 L 3
-11.701867227882623 29.77460395147962 14.506691505134297 F 4
9.429475402706576 28.287342791648197 6.676924501791871 CH 5
25.645069779242746 25.449506312958032 9.78511719613052 A 6
46.45098906863463 33.77447230655959 12.624508777350401 CZ 7
65.6494359636268 59.43715463606391 19.42472877247108 PL 8
-3.546190833310119 93.28328965373458 6.039505452538444 DK 9
EOD
plot \
'$data' using 1:2:3:5 with circles lc var notitle, \
'$data' using 1:2:4:5 with labels font "arial,9" tc var notitle
pause -1
```

1.4. Speziallfälle

Folgende Spezialfälle wurden identifizeirt und sind getestet.

- Nur ein Staat
- TODO

1.5. Fehlerfälle

Im Falle eines Fehler durch falsche Eingaben wird eine Fehlermeldung ausgegeben und die Lösungsfindung für die entsprechende Eingabe abgebrochen.

1.6. Vereinfachungen

Gegenüber der Realität unserer Erde wurden hierbei eine Reihe von vereinfachten Ahnnahmen getroffen.

1. Die Karte ist unendlich groß.

2. Die Karte ist flach und nicht gewölbt.
3. Der Abstand zwischen zwei Längengraden ist immer gleich groß.
4. Der Abstand benachbarter Längengrade ist gleich dem Abstand benachbarter Breitengrade
5. Daraus folgend sind Abstände nicht auf einer Kugel berechnet.

2. Verfahrensbeschreibung

2.1. Übersicht

Zuerst werden die Staaten und Nachbarschaftsbeziehungen aus der Eingabedatei eingelesen. Danach wird mithilfe eines iterativen Algorithmus mit einer festen Anzahl von Iterationen eine Lösung gesucht. Diese wird dann in eine für gnuplot aufbereitete Form in eine Ausgabedatei geschrieben.

2.2. Einlesen

Das Einlesen geschieht mit einem speziell auf das Eingabeformat abgestimmten Parser der die Eingabedatei nur genau einmal durchläuft. Der Parser ist so geschrieben das er die Eingabe nicht nur aus Dateien lesen kann sondern ist über das io.Reader interface aus der golang Standardbibliothek erweiterbar.

2.3. Algorithmus

Der Algorithmus besteht aus einer festen Anzahl von Iterationen. In jeder Iterationen werden die Mittelpunkte der Staaten verschoben. Diese Verschiebungen geschehen in Abhängigkeit von den aktuellen Überschneidungen sowie den Abständen von Nachbarstaaten. Dafür werden zuerst für jeden Staat Abstoßungskräfte durch Überschneidungen und Anziehungskräfte durch Nachbarn berechnet, dann diese pro Staat addiert und schlussendlich alle Staaten proportional zur berechneten Kraft verschoben. Diese Verschiebung ist gedämpft um die echten Lagebeziehungen möglichst beizubehalten und um Schwingungseffekte zu verringern.

2.3.1. Berechnung der Kräfte

Zur Berechnung der Kräfte wird über jeden Staat(a) iteriert und der Abstand zu jedem anderen Staat(b) berechnet, hierbei ist der kürzeste Abstand von Kreisaußenkante zu Kreisaußenkante gemeint und nicht der Abstand der Mittelpunkte. Falls die Kreise sich über-

schneiden ist der Abstand negativ. Abhängig davon ob die Länder benachbart sind wird dann eine Kraft auf den Staat berechnet. Die Stärke der Kraft(f) abhängig vom Abstand(d) berechnet sich bei Nachbarstaaten wie folgt:

$$f(d) = d$$

Dies sorgt dafür das Nachbarn sich berühreren. Bei nicht Nachbarschaftsbeziehungen:

$$f(d) = \begin{cases} 1,2 \cdot d & \text{falls } d < 0 \\ 0 & \text{sonst} \end{cases}$$

Dies sorgt dafür das nicht benachbarte Länder sich nicht überschneiden. Die Richtung der Kraft ist hierbei immer vom Mittelpunkt des Staates a in Richtung des Mittelpunktes Staat b. Bei einer negativen Stärke wirkt die Kraft in die umgekehrte Richtung. Da die Kraft von b auf a auch gleichzeitig eine gleichstarke Kraft von a auf b bewirkt, werden alle Kräfte halbiert damit die Abstände durch sie nicht doppelt kompensiert werden.

2.3.2. Anwendung der Kräfte

Nachdem alle Kräfte berechnet wurden wird über alle Staaten iteriert und auf jeden die berechnete Kraft(f) angewendet. Dies geschieht durch eine Verschiebung(v):

$$v = c \cdot f, c\epsilon(0,1)$$

Wobei c ein Dämpfungsfaktor zwischen 0 und 1 ist. Die Dämpfung sorgt dafür das die Lagebeziehungen nicht sicht nicht zu stark ändern und soll Schwingungseffekte verhindern.

2.4. Komplexität

Dadurch das für jeden Staat der Abstand zu jedem anderen Staat berechnet werden muss, liegt die Komplexität des Algorithmus bei $O(n)^2$ wobei n die Anzahl der Staaten ist.

2.5. Ausgabe

Die Ausgabe ist getrennt von dem Algorithmus und bereitet die Ergebnisse für gnuplot auf.

3. Programmbeschreibung

3.1. Packages

Das Gesamte Programm ist in 4 Packages aufgeteilt. Es gibt ein Package model in dem die Datenstrukturen definiert sind. Außerdem gibt es jeweils für die Eingabe, den Algorithmus und die Ausgabe ein eigenes Package. Diese Packages sind untereinander unabhänging und loose über die im model Package definierten Datenstrukturen gekoppelt. Außerdem gibt es natürlich das main Package das in der main() Methode alles verbindet.

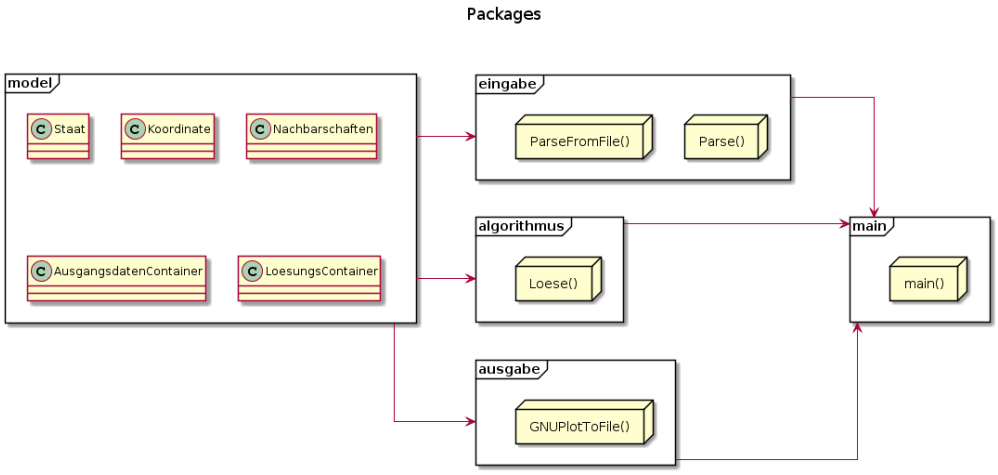

Abbildung 3.1.: Übersicht Packages

3.1.1. Model

Das Package Model enthält alle benötigten Klassen. Es wurde versucht die Klassenhirarchien möglichst simpel zu gestalten.

Koordinate

Die Klasse Koordinate beschreibt einen Vektor im R^2. Sie besitzt eine Reihe von Helfermethoden, die einfache Operationen wie die Streckung oder Addition von Koordinaten

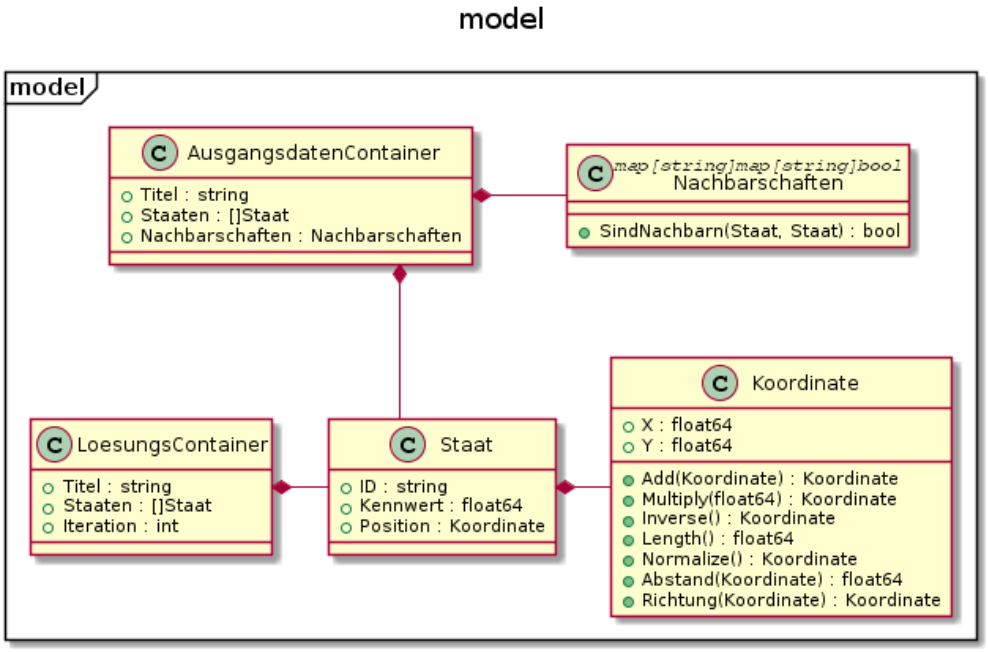

Abbildung 3.2.: Package model

abbilden.

Als Datentyp für die Felder wurde float64 gewählt um eine möglichst große Genauigkeit zu haben. Außerdem entahlten einige Methoden NaN Checks nach den Operationen und liefern im Zweifel einen Nullvektor zurück um numerische Instabilitäten zu vermeiden.

Staat

Die Klasse Staat ist die Datenstruktur für einen Staat und besteht aus einer string als ID, einem float64 als Kennwert und einer Koordinate als Position.

Nachbarschaften

Die Klasse Nachbarschaften beschreibt die Nachbarschaften zwischen den Staaten. Sie ist ein typedef auf den Type map[string]map[string]bool, und beschreibt somit eine 2-dimensionale boolsche Set mit string Keys. Durch das Ausnutzen von go's zero value erlaubt uns dies sehr einfache Prüfungen von Nachbarschaftsbeziehungen. Mehr Informationen dazu lassen sich hier finden. Die Methode SindNachbarn(Staat, Staat) prüft ob zwei Staaten Nachbarn sind. Dies ist dank der Verwendung einer Map in $O(1)$ möglich.

AusgangsdatenContainer

Die Klasse AusgangsdatenContainer fasst alle zur Lösung des Problems notwendigen Informationen zusammen. Sie dient als Schnittstellendatentyp zwischen Eingabe und Algorithmus.

LoesungsContainer

Die Klasse LoesungsContainer fasst die Lösung zusammen. Sie dient als Schnittstellendatentyp zwischen Algorithmus und Ausgabe.

3.1.2. Eingabe

Das Package Eingabe enthält Funktionen um das spezifizierte Eingabeformat zu parsen. Dafür stellt es zwei Funktionen zu Verfügung. Die Funktionen liefern alle einen AusgangsdatenContainer und einen error zurück.

Parse

Die Methode Parse() erwartet ein Objekt des Interfaces io.Reader und liefert ein Objekt der Klasse AusgangsdatenContainer und ein Objekt des Interfaces error zurück. Falls beim Einlesen kein Fehler aufgetreten ist, wird nil als error zurück geliefert. Durch das io.Reader Interface ist die Parse Funktion nicht auf einen festen Eingabetypen festgelegt und somit leicht für verschiedene Datentypen verwendbar.

ParseFromFile

Die Methode ParseFromFile erwartet als Parameter den Pfad zu einer Datei. Sie versucht dann diese Datei zu öffnen und seinen Inhalt mithilfe der Parse() Funktion einzulesen. Sie liefert ein Objekt der Klasse AusgangsdatenContainer und ein Objekt des Interfaces error zurück. Der error ist heirbei entweder ein Fehler der beim öffnen der Datei passiert oder ein Fehler beim Einlesen der Datei mit der Parse() Funktion.

3.1.3. Algorithmus

Das Package Algorithmus enthält die Funktionen zum lösen des Problemes.

Loese

Die Funktion Loese ist die einzig öffentliche Funktion des Packages. Sie erhält einen AusgangsdatenContainer als Parameter und liefert einen LoesungsContainer zurück. Sie erstellt ein Objekt der Klasse algorithmus, initialisiert es und ruft dann in einer Schleife 1000 mal die iteration Methode auf ihm auf. Danach liefert sie die Loesung zurück.

algorithmus

Die Klasse algorithmus implementiert die Lösung des Problemes und speichert den aktuellen Stand der Lösung. Zusätzlich hat sie ein Feld für Nachbarschaftsbeziehungen und eins für die aktuell in dieser Iteration berechneten Kräfte. Die wichtigste Methode ist iteration(). Diese berechnet die nächste Iteration der Lösung.

algorithmus.iteration()

Die Methode Iteration ist der Kern des Algorithmus, hier werden die Kräfte berechnet und die Staaten verschoben.

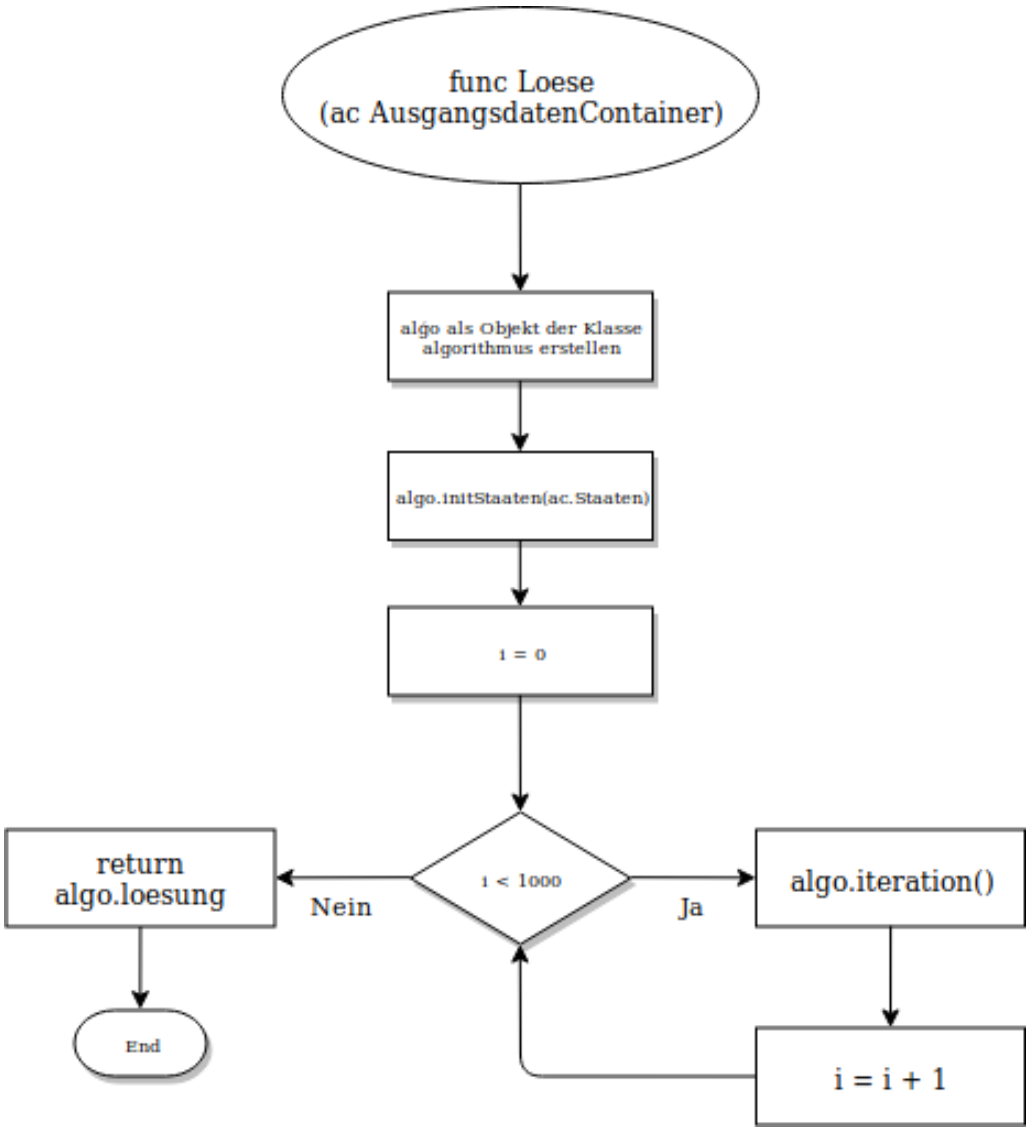

Abbildung 3.3.: Funktion Loese

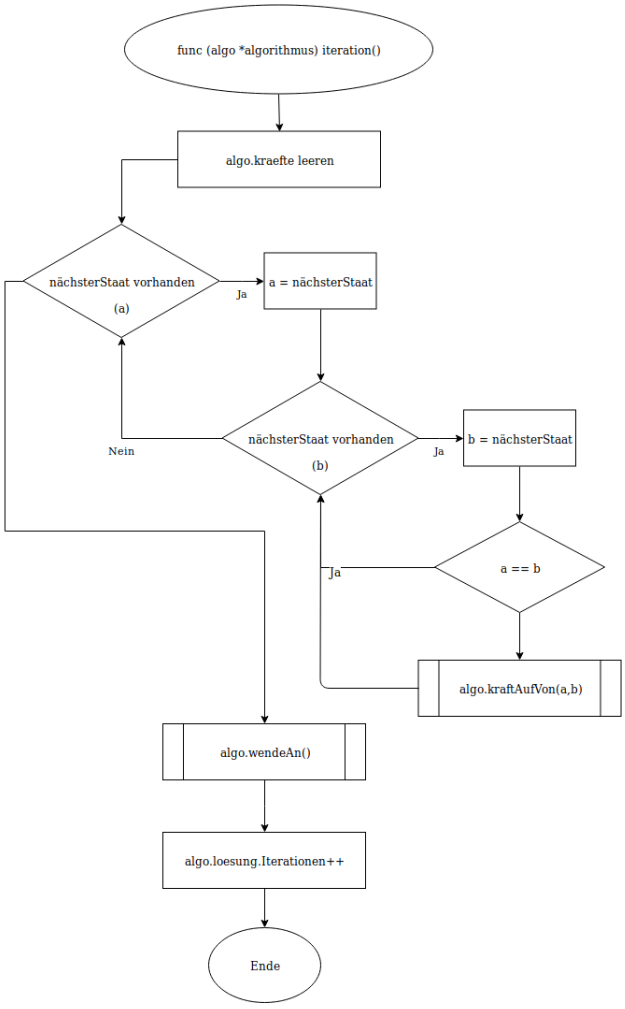

Abbildung 3.4.: Methode algorithmus.Iteration

algorithmus.kraftVonAuf()

In der Methode kraftVonAuf wird die Kraft von Staat b auf Staat a berechnet.

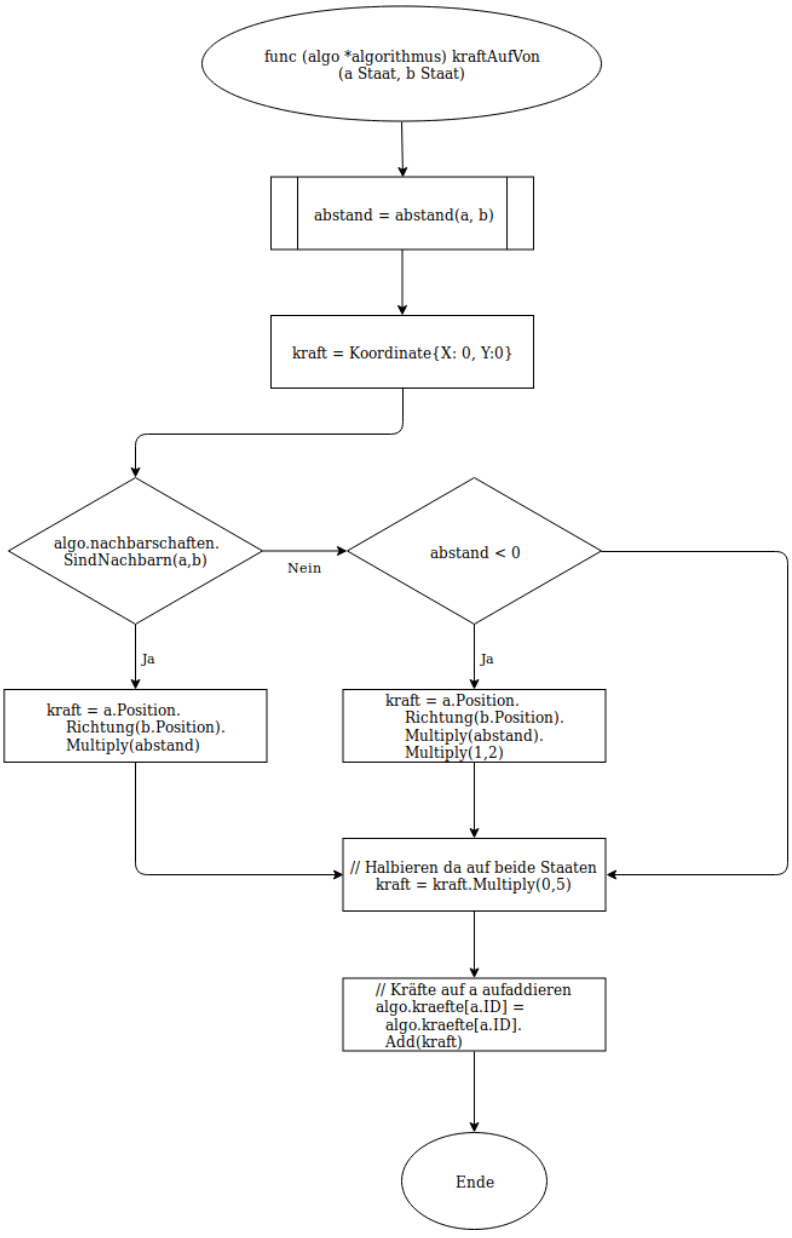

Abbildung 3.5.: Methode algorithmus.kraftVonAuf

algorithmus.wendeAn()

Die Methode wendeAn() wendet die berechneten Kräfte auf die Staaten an.

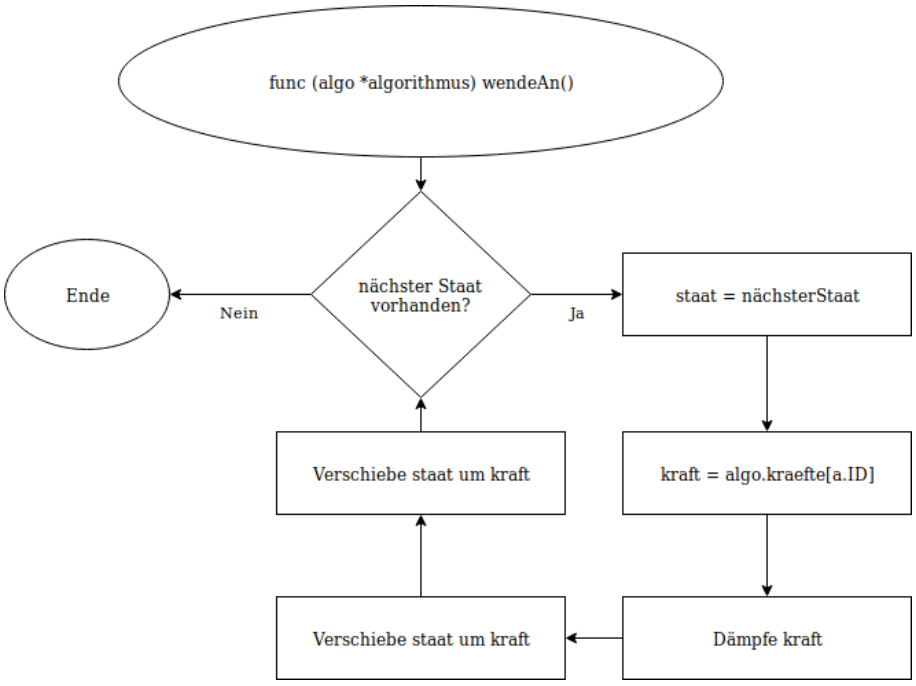

Abbildung 3.6.: Methode algorithmus.wendeAn

3.1.4. Ausgabe

Das Package ausgabe enthält die nötigen Funktionen um die Lösung für gnuplot aufzubereiten und in eine Ausgabedatei zu schreiben.

GNUPlotToFile()

Die Funktion GNUPlotToFile schreibt die Ausgabedatei in dem an gnuplot ausgerichteten Format. Sie ruft dabei die Helferfunktion findExtrema auf um sicherzustellen das die xRange gleich groß wie die yRange ist. Falls es einen Fehler beim erstellen der Datei gibt wird dieser zurückgegeben.

findExtrema()

findExtrema ist eine Helferfunktion um sicherzustellen das die xRange gleich groß wie die yRange ist und alle Staaten vollständig im Bild enthalten sind. Dafür findet sie erst das kleinste Rechteck um alle Staaten herum und verlängert dann die kürzere Seite des Rechtecks gleichmäßig auf beiden Seiten um ein Quadrat zu erschaffen.

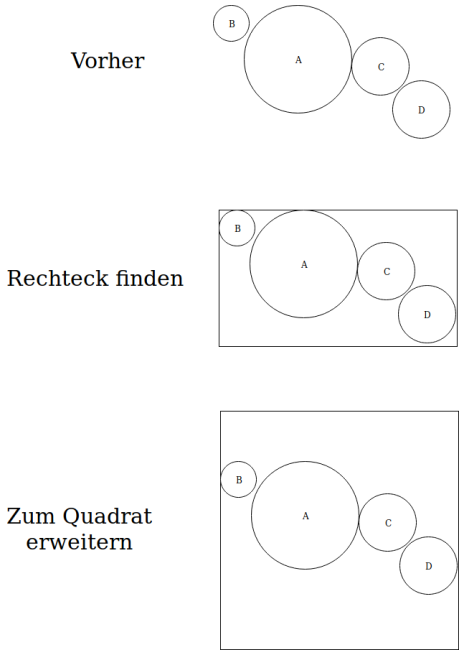

Abbildung 3.7.: Funktion findExtrema

4. Testdokumentation

Das Programm ist durch umfangreiche Unit Tests und durch die 3 Testbeispiele der Aufgabenstellung getestet.

5. Ausblick

5.1. Ausblick

Das Programm funktioniert, doch es sind noch viele weitere Verbesserungen denkbar und sinnvoll.

5.1.1. Abruchkriterium

Es wäre sinnvoll die Anzahl der Iterationen dynamisch früher abzubrechen falls ein gewisses Abruchkriterium erreicht ist. Denkbar wäre ein Abbruch z. B. wenn alle Kräfte sehr klein sind.

5.1.2. Voriteration

Auch eine Voriteration in der z.B. einfach alle Staaten weiter auseinander geschoben werden oder die durchschnittliche Größe der Kreise auf den durschnittlichen Abstand von Nachbarstaaten normiert werden könnte von Vorteil sein.

5.1.3. Direkte Anbindung an gnuplot

Es wäre denkbar gnuplot direkt aus dem Programm heraus aufzurufen und die Bilder zu erstellen.

5.1.4. Andere Ausgabeprogramme

Eine Anbindung an andere Grafikprogramme zum anzeigen der Bilder wäre möglich.

5.1.5. REST Service

Eine weitere Verbesserung der Bedienbarkeit wäre es das Programm in einen Service mit einer HTTP/REST Schnittstelle zu erweitern.

A. Abweichungen und Ergänzungen zum Vorentwurf

Die größten Abweichungen zur Vorlage liegen darin das die Abstoßungs- und Anziehungskräfte nicht getrennt berechnet werden sondern in einem Schritt.

B. Benutzeranleitung

C. Entwicklungsumgebung

Programmiersprache	:	golang
Compiler	:	go1.12.4 linux/amd64
Rechner	:	amd64
Betriebssystem	:	Ubuntu 18.04

D. Verwendete Hilfsmittel

D.1. Programme

- Editor: Visual Studio Code
- Layout: LaTeX
- Diagramme: https://www.planttext.com/
- Programmablaufpläne: https://draw.io
- Versionsverwaltung: git, https://github.com

D.2. Quellen

- https://godoc.org
- https://stackoverflow.com

E. Erklärung

Erklärung des Prüfungsteilnehmers / der Prüfungsteilnehmerin:

Ich erkläre verbindlich, dass das vorliegende Prüfprodukt von mir selbstständig erstellt wurde. Die als Arbeitshilfe genutzten Unterlagen sind der Arbeit vollständig aufgeführt. Ich versichere, dass der vorgelegte Ausdruck mit dem Inhalt der von mir erstellten digitalen Version identisch ist. Weder ganz noch in Teilen wurde die Arbeit bereits als Prüfungsleistung vorgelegt. Mir ist bewusst, dass jedes Zuwiderhandeln als Täuschugnsversuch zu gelten hat, der die Anerkennung des Prüfproduktes als Prüfungsleistung ausschließt

Bonn, den 23. Mai 2019

Ort und Datum

Unterschrift des Prüfungsteilnehmers

Anhang F

Aufgabenstellung