Dokumentation der Praktischen Arbeit zur Prüfung zum Mathematisch-technischen Softwareentwickler

17. Mai 2018

Christian Anders

Prüfungs-Nummer:	101 20534
Programmiersprache:	Java
Ausbildungsort:	Forschungszentrum Jülich

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Aufgabenanalyse

1.1 Problemanalyse

Im Folgenden soll ein Programm zur Bestimmung der kritischen Pfade und der Gesamtdauer eines Netzplans implementiert werden. Ein Netzplan ist ein Hilfsmittel zur Terminplanung von Projekten und besteht aus mehreren Vorgängen des Projekts, die nach ihren Abhängigkeiten und ihrer Reihenfolge verkettet sind. Mit Hilfe des Netzplans lassen sich die Gesamtdauer des Projektes und auch die Vorgänge bestimmen, die den Endtermin verzögern können. Ein Vorgang des Netzplans besteht in diesem Programm aus einer Vorgangsnummer, einer Vorgangsbezeichnung, einer Vorgangsdauer (D), den Vorgänger und Nachfolger Vorgängen. Die Start- und Endvorgänge enthalten keine Vorgänger bzw. Nachfolger. Außerdem wird zu jedem Vorgang ein frühester Anfangszeitpunkt (FAZ), ein frühester Endzeitpunkt (FEZ), ein spätester Anfangszeitpunkt (SAZ), ein spätesten Endzeitpunkt (SEZ), sowie ein Gesamtpuffer (GP) und ein freier Puffer (FP) berechnet. Anhand dieser Werte können anschließend die kritischen Pfade bestimmt werden. Ein kritischer Pfad ist eine Aneinanderreihung von kritischen Vorgängen, der an einem Startvorgang beginnt und an einem Endvorgang endet. Ein kritischer Vorgang wiederum ist ein Vorgang dessen Gesamt- und freier Puffer gleich Null ist. In einem Netzplan können mehrere kritische Pfade auftreten. Der Algorithmus zur Bestimmung der Werte FAZ, FEZ, SAZ, SEZ, GP und FP ist in drei Phasen gegliedert. In der ersten Phase werden, ausgehend von allen Startvorgängen, die Werte FAZ und FEZ=FAZ+D berechnet. Der FAZ jeden Vorgangs ist der FEZ seines Vorgängers. Hat ein Vorgang mehr als einen Vorgänger wird der späteste, also größte FAZ seiner Vorgänger übernommen. Der Start des Vorgangs wird dadurch möglich, dass alle Startvorgänge den FAZ=0 besitzen, da dieser als Startzeitpunkt für das Projekt angesehen wird. In der zweiten Phase werden ausgehend von allen Endvorgängen die Werte SEZ und SAZ=SEZ-D berechnet. Der SEZ jedes Vor-

gangs ist der SAZ seines Nachfolgers, wobei bei mehreren Nachfolgern der früheste, also kleinste SAZ übernommen wird. Der Start des Verfahrens wird dadurch möglich, dass bei allen Endvorgängen der SEZ gleich dem in der ersten Phase berechneten FEZ ist. In der dritten Phase wird für jeden Vorgang der GP aus GP=SAZ-FAZ=SEZ-FEZ und der FP aus dem kleinsten FAZ der Nachfolger minus dem eigenen FEZ berechnet. Da bei Endvorgängen der SEZ gleich dem FEZ ist und Endvorgänge keine Nachfolger haben, ist ihr GP und FP immer gleich Null. Im Programm werden die zweite und dritte Phase zu einer zusammengefasst, da dadurch der Netzplan nicht erneut durchlaufen werden muss. Zur Bestimmung des kritischen Pfades wird nun von jedem Startvorgang aus jeder mögliche Pfad zu einem Endvorgang abgelaufen und geschaut, ob dieser komplett aus kritischen Vorgängern besteht. Ist dies der Fall wird dieser ausgegeben. Es muss vor den Berechnungen geprüft werden, ob der Netzplan zusammenhängend ist oder ob sich Zyklen darin befinden. Mögliche Fehlerfälle eines Netzplan wären:

- kein Startvorgang
- kein Endvorgang
- nur ein Vorgang
- zyklischer Netzplan
- ein nicht zusammenhängender Netzplan
- ein Vorgang hat keinen Vorgänger oder keinen Nachfolger

Sonderfälle, die bei einem Netzplan auftreten können sind:

- nur ein Start- und ein Endvorgang
- mehrere Start- und/oder Endvorgänge
- besonders viele Vorgänge (Maximalbeispiel)
- nur eine Verbindung, die zwei Netzpläne verbindet

1.2 Zerlegung in Teilmodule

Das Programm lässt sich in die folgenden Module einteilen:

- Das Einlesen und Überprüfen der Vorgänge und des Titels aus einer Datei.
- Die Überführung der Eingangsdaten in eine geeignete Datenstruktur.
- Die Überprüfung des Netzplans.
- Die Berechnung aller kritischen Pfade und aller Größen, die dazu nötig sind. Diese wird durch rekursive Methoden umgesetzt.
- Die Ausgabe der Ergebnisse in einer Datei.
- Die Fehlerausgabe.

1.3 Aufbau der Eingabedatei

Die Eingabedaten liegen für jedes Testbeispiel in einer eigenen Datei vor. In der Datei können Kommentare und müssen Vorgänge und der Titel des Testfalls stehen. Jeder Zeile, in der ein Kommentar steht, muss ein '\\' voran gehen, wobei anschließend die ganze Zeile als Kommentar gesehen wird. Der Titel des Testfalls muss in einer Zeile stehen, die mit '\\+' beginnt. Dabei muss die Datei genau eine Titelzeile enthalten. Alle anderen Zeilen, die nicht mit '\\' oder '\\+' beginnen, enthalten Vorgänge. Es muss mindestens ein Vorgang in der Datei enthalten sein. Eine solche Zeile beinhaltet in der folgenden Reihenfolge die Vorgangsnummer, Vorgangsbezeichnung, Vorgangsdauer, Vorgänger und Nachfolger. Jedes dieser Attribute muss mit einem Semikolon vom nächsten getrennt sein, wobei Leerzeichen zwischen den Semikolons und den Attributen erlaubt sind. Semikolons die vor dem ersten oder hinter dem letzten Attribut stehen sind erlaubt, so lange vor oder hinter ihnen keine anderen Zeichen als Leerzeichen stehen. Gibt es mehrere Vorgänger oder Nachfolger, sind diese jeweils mit einem Komma voneinander getrennt. Vor oder hinter dem Komma sind Leerzeichen, aber keine anderen Zeichen erlaubt. Eine Vorgangsnummer ist ganzzahlig und ≥ 0 und darf nicht mehrmals in einer Datei vorkommen. Die Vorgangsbezeichnung darf keine Semikolons enthalten. Die Dauer eines Vorgangs ist eine Dezimalzahl mit maximal zwei Nachkommastellen die ≥ 0 sein muss. Sind mehr als zwei Nachkommastellen angegeben werden diese abgeschnitten. Die Vorgänger und Nachfolger dürfen nur Vorgangsnummern enthalten, die existieren. Diese sind dadurch ebenfalls ganzzahlig und ≥ 0. Außerdem darf kein Vorgang Vorgänger und zugleich Nachfolger

eines anderen Vorgangs sein und ein Vorgang auch nicht sein eigener Vorgänger oder Nachfolger. Start- bzw. Endvorgänge, die keine Vorgänger bzw. Nachfolger besitzen haben ein '-' anstelle der Zahl, wobei Leerzeichen vorkommen dürfen.

Durch diese Vorgaben können folgende Fehlerfälle auftreten:

- Die Eingabedatei existiert nicht.
- Die Eingabedatei enthält keine Vorgänge.
- Die Eingabedatei enthält keinen oder mehr als einen Titel.
- Ein Attribut enthält einen ungültigen Datentyp.
- Eine Zeile enthält mehr oder weniger als vier Semikolons mit weiteren Zeichen dahinter/davor.
- Eine Vorgangsnummer ist negativ oder kommt mehrmals vor.
- Die Vorgangsdauer ist negativ.
- Ein Vorgang hat sich selbst als Vorgänger oder Nachfolger.
- Ein Vorgänger oder Nachfolger enthält eine ungültige Vorgangsnummer.
- Ein Vorgang ist Nachfolger eines anderen Vorgangs, dieser ist aber nicht Vorgänger des anderen Vorgangs.

1.4 Aufbau der Ausgabedatei

Die Ausgabe erfolgt in einer Datei. Ist eine Exception aufgetreten wird diese zusammen mit dem Inhalt der Eingabedatei in der Ausgabedatei dokumentiert. Ist dies nicht der Fall, steht der Titel des Testfalls in der ersten Zeile. Anschließend folgt für die Übersicht eine Leerzeile und in einer weiteren Zeile die Attributsbezeichnungen mit einem Semikolon von einander getrennt. Diese sind: Vorgangsnummer, Vorgangsbezeichnung, D,FAZ,FEZ,SAZ,SEZ,GP und FP. Darunter folgen so viele Zeilen wie es Vorgänge gibt, die jeweils pro Zeile die Werte eines Vorgangs mit einem Semikolon getrennt beinhalten. Nach diesen Zeilen folgt erneut eine Leerzeile. Anschließend drei Zeilen, die den Anfangsvorgang, Endvorgang und die Gesamtdauer des Projektes beinhalten. Nach einer weiteren Leerzeile eine Zeile in der 'kritischer Pfad' steht. In der darauffolgenden Zeile wird der kritische Pfad ausgegeben. Die Vorgänge des kritischen Pfads sind mit dem String '- ⟩' von einander getrennt.

Kapitel 2

Verfahrensbeschreibung

2.1 Überführung in eine logische Datenstruktur

Die Klasse 'Vorgang' dient dazu einen Vorgang zu Speichern. Sie enthält eine String Variable um die Bezeichnung des Vorgangs zu speichern. Außerdem gibt es 7 Variablen vom Typ Double, die die Werte FAZ, FEZ, SAZ, SEZ, D, GP und FP speichern. Darüber hinaus gibt es eine Integer und eine Boolsche Variable zur Speicherung der Vorgangsnummer und ob dieser Vorgang kritisch ist oder nicht. Die beiden Attribute Vorgänger und Nachfolger werden in zwei Integer Feldern gespeichert. Um die Informationen eines Vorgangs in einem String zusammenzufassen wird die Methode toString() überschrieben. Die beiden Methoden istStartvorgang() und istEndvorgang() überprüfen, ob der Vorgang ein Start- bzw. Endvorgang ist und geben dementsprechend einen Boolean zurück. Für jedes Attribut werden Getter und Setter zur Verfügung gestellt. Von den Settern ausgenommen sind die Vorgangsnummer, Vorgangsbezeichnung, Dauer, Vorgänger und Nachfolger, da diese einmalig mit dem Konstruktor gesetzt werden.

2.2 Das Einlesen der Daten

Der formale Aufbau der Eingabedatei ist genau festgelegt (siehe Abschnitt 1.3). Zu Beginn des Einlesens werden alle in der Aufgabenanalyse beschriebenen Fehlerfälle überprüft und gegebenenfalls wird eine EingabeException mit einer Fehlermeldung geworfen, die den aufgetretenen Fehler beschreibt. Die Fehlermeldung wird zusammen mit dem Inhalt der Eingabedatei in die Ausgabedatei geschrieben und das Programm wird beendet, da eine korrekte Bearbeitung nicht möglich ist. Anderenfalls werden die Vorgänge zeilen-

weise eingelesen.

Die Verarbeitung der Eingabedatei erfolgt zeilenweise. Für jede Zeile wird geprüft, ob sie mit '\\' oder '\\+' beginnt. Ist ersteres der Fall wird diese Zeile ignoriert und es wird zur nächsten Zeile gegangen. Enthält der Anfang der Zeile die Zeichenfolge '\\+', wird der Titel eingelesen und in einem String gespeichert. Leerzeichen die vor oder hinter dem Titel stehen werden dabei weggelassen. Für jede andere Zeile wird versucht einen Vorgang einzulesen. Es werden die Vorgangsnummer, die Vorgangsbezeichnung, die Vorgangsdauer, die Vorgänger und die Nachfolger aus dem String entnommen und in ihren Datentyp geparst. Anschließend wird ein Objekt vom Typ Vorgang erzeugt und dem Konstruktor die Werte für die Vorgangsnummer, die Vorgangsbezeichnung, die Dauer, die Vorgänger und die Nachfolger übergeben. Der neu erzeugte Vorgang wird dann in einer HashMap gespeichert. Den Schlüssel bildet dabei die Vorgangsnummer. Ist dies für alle Zeilen/Vorgänge geschehen, gibt die Methode die erstellte HashMap, die alle Vorgänge enthält, zurück.

2.3 Algorithmus zur Bestimmung des kritischen Pfades

Wie bereits in der Aufgabenanalyse unter 1.1 beschrieben, ist der Algorithmus in drei Phasen eingeteilt, wobei die zweite und dritte Phase in einen Programmschritt zusammengefasst sind. Die Methoden des Programms funktionieren rekursiv. Der rekursive Algorithmus für die erste Phase wird für jeden Endvorgang aufgerufen, bekommt diesen also im Funktionsaufruf übergeben. Anschließend wird geprüft, ob das Attribut FAZ ungleich -1 ist. Ist dies nicht der Fall, wird die Funktion mit dem Vorgänger als Übergabeparameter erneut aufgerufen. Gibt es mehrere Vorgänger wird dies für jeden von diesen gemacht. Der Algorithmus läuft nun bis zu einem Startvorgang durch und stoppt da, da der FAZ bei einem Startvorgang gleich Null ist. Es ist auch möglich, dass die Rekursion bei einem Vorgang stoppt, der bereits berechnet wurde. So werden keine Pfade mehrmals durchlaufen. Auf dem rekursiven Rückweg werden nun alle FAZ bzw. FEZ Werte berechnet, bis man schließlich beim Endvorgang wieder ankommt. Der FAZ eines Vorgangs ist der FEZ des Vorgängers. Gab es zuvor mehrere Vorgänger, wird der späteste, also größte FEZ für die Berechnung benutzt. Ist man beim Endvorgang wieder angekommen, werden von diesem SEZ=FEZ und SAZ=SEZ-D berechnet. Die zweite Methode, die die zweite und dritte Phase zusammenfasst, wird von jedem Startvorgang aus gestartet. Es wird überprüft, ob das Attribut SEZ ungleich -1 ist. Wenn nicht, wird für jeden Nachfolger die Methode erneut aufgerufen und der Nachfolgervorgang der Methode übergeben. Ist der SEZ ungleich -1, ist man an einem Endvorgang angekommen und die Rekursion stoppt. Es ist auch möglich, dass die Rekursion bei einem Vorgang stoppt, der bereits berechnet wurde. So werden keine Pfade mehrmals durchlaufen. Auf dem rekursiven Rückweg werden

anschließend alle Werte (SAZ,SEZ,GP,FP,kritisch) berechnet. Besaß ein Vorgang mehrere Nachfolger, wird der frühste, also kleinste SAZ der Nachfolger für die Berechnung genutzt. In einem letzten Schritt, wird der Netzplan erneut von allen Startvorgängen aus rekursiv durchlaufen. Während des Durchlaufs wird geprüft, ob der aktuelle Vorgang kritisch ist. Ist das der Fall geht die Methode rekursiv weiter. Andernfalls wird die Rekursion an der Stelle beendet und es wird zum vorherigen Vorgang, der kritisch war, zurückgekehrt. Von dort aus wird für jeden Nachfolger der gleiche Schritt gemacht. Die Rekursion endet, wenn man an einem Endvorgang angekommen ist.

Kapitel 3

Programmbeschreibung

Das Programm ist nach dem Drei-Schichten und dem MVC (Model-View-Control)-Modell entwickelt worden. Die Klasse Vorgang übernimmt die Speicherung der Daten. Die Vorgangsnummer wird in als Integer und die Vorgangsbezeichnung als String gespeichert. Die Werte für den FAZ, FEZ, SAZ, SEZ, GP und FP werden in Variablen vom Typ Double gespeichert. Desweiteren werden zwei Integer Felder für die Vorgänger und Nachfolger und ein Boolean angelegt. Der Boolean speichert, ob ein Vorgang kritisch ist oder nicht.

Die Klassen Eingabe und Ausgabe stellen die Präsentationsschicht dar. Sie regeln das Einlesen aus der Eingabedabei bzw. stellen die ermittelten Ergebnisse für den Benutzer in einer Textdatei dar. Zu jeder Eingabedatei wird eine Ausgabedatei mit dem gleichen Namen und der Endung „.out“ erstellt, sofern nicht zuvor mit dem Übergabeparameter beim Aufruf der jar Datei der Parameter -o benutzt wurde. Der Parameter -o gibt an, dass ein Pfad für die Ausgabedatei angegeben wurde diese für die Ausgabe genutzt werden soll.

Die Klasse Netzplanerstellung entspricht der Anwendungslogik. Sie stellt die Schnittstelle zwischen Datenspeicherung und Ein- und Ausgabe dar. Sie enthält die Methoden zur Bestimmung des kritischen Pfades. Ihre run()-Methode wird von der Main-Klasse aufgerufen und steuert den gesamten Ablauf des Programms.

Zusätzlich gibt es noch zwei Klassen EingabeException und NetzplanException, die von RuntimeException abgeleitet sind. Die Klasse EingabeException ist dabei für Fehler bei der Eingabe und die Klasse NetzplanException für logische Fehler im Netzplan verantwortlich.

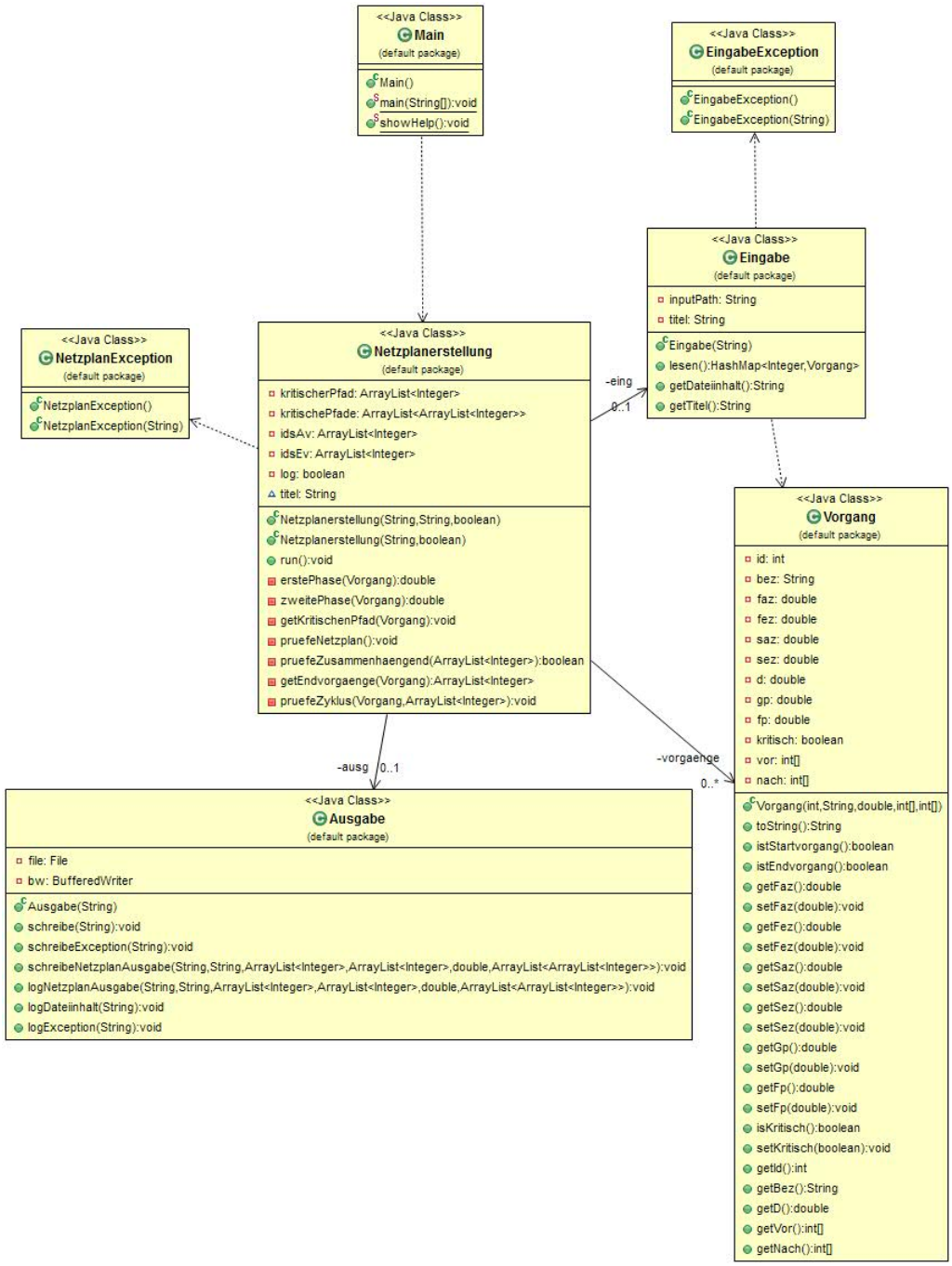

Abbildung 3.1: UML-Klassendiagramm

3.1 Die Klassenstrukturen

3.1.1 Die Klasse Eingabe

Die Klasse Eingabe dient zum Einlesen der Daten aus einer Eingabedatei und überprüft diese auf formale und inhaltliche Fehler.

Die Attribute der Klasse Eingabe

-**inputPath: String** Enthält den Pfad der Eingabedatei.

-**titel: String** Enthält nach dem Extrahieren den Titel des Testbeispiels.

Methoden der Klasse Eingabe

+**Eingabe(inputPath: String)** Der Konstruktor erzeugt eine Instanz der Klasse Eingabe und der übergebene Pfad zur Eingabedatei wird gespeichert.

+**lesen()** Liest die Eingabedatei ein und überprüft sie auf mögliche Fehler (siehe dazu Abschnitt 1.3).

+**getTitel(): int** Gibt den Titel des Testbeispiels zurück.

3.1.2 Die Klasse Vorgang

Diese Klasse repräsentiert einen Vorgang mit den übergebenen Werten. Die Vorgangsnummer, Vorgangsbezeichnung, die Dauer, die Vorgänger und die Nachfolger werden gespeichert. Die Vorgänger und Nachfolger werden in den beiden Feldern durch ihre Ids repräsentiert. Zusätzlich werden Variablen für den FAZ, FEZ, SAZ, SEZ, GP, FP und ob der Vorgang kritisch ist oder nicht angelegt. Der FAZ und SEZ werden mit -1 initialisiert.

Die Attribute der Klasse Vorgang

-**id: int** Enthält die Vorgangsnummer

-**bez: String** Enthält die Vorgangsbeschreibung

-**d: double** Enthält die Vorgangsdauer

-**faz: double** Enthält den frühesten Anfangszeitpunkt

-**fez: double** Enthält den frühesten Endzeitpunkt

-**saz: double** Enthält den spätesten Anfangszeitpunkt

-**sez: double** Enthält die spätesten Endzeitpunkt

-**gp: double** Enthält den Gesamtpuffer

-**fp: double** Enthält den freien Puffer

-**vor: int[]** Enthält die Vorgänger dieses Vorgangs

-**nach: int[]** Enthält die Nachfolger dieses Vorgangs

-**kritisch: boolean** Enthält die Information, ob der Vorgang kritisch ist oder nicht

Methoden der Klasse Vorgang

+**Vorgang(id: int, bez: String, d: double, vor: int[], nach: int[])** Der Konstruktor erzeugt eine Instanz der Klasse Vorgang mit einer Vorgangsnummer, Vorgangsbezeichnung,

Vorgangsdauer, seinen Vorgängern und Nachfolgern.

+toString(): String Gibt einen String zurück, der alle Informationen eines Vorgangs beinhaltet.

+istStartvorgang(): boolean Gibt die Information zurück, ob der Vorgang ein Startvorgang ist oder nicht.

+istEndvorgang(): boolean Gibt die Information zurück, ob der Vorgang ein Endvorgang ist oder nicht.

+getter und setter Es werden für alle Attribute Getter und Setter zur Verfügung gestellt. Von den Settern ausgenommen sind die Vorgangsnummer, Vorgangsbezeichnung, Vorgangsdauer, Vorgänger und Nachfolger. Diese werden einmalig mit dem Konstruktor gesetzt.

3.1.3 Die Klasse Netzplanerstellung

Diese Klasse gibt den kritischen Pfad und die Gesamtdauer eines Netzplanes aus, um mögliche Verbesserungen für eine schnellere Abarbeitung des Projektes aufzuzeigen. Die Berechnung findet in zwei rekursiven Methoden statt. Außerdem werden die Strings für die Ausgabe erstellt.

Die Attribute der Klasse Netzplanerstellung

-**kiritscherPfad: ArrayList¡Integer¿** Enthält den aktuellen kritischen Pfad.

-**kritischePfade: ArrayList¡Integer¿** Enthält eine Liste aller bereits ermittelten kritischen Pfade.

-**idsAv: ArrayList¡Integer¿** Enthält die Vorgangsnummern aller Startvorgänge.

-**idsEv: ArrayList¡Integer¿** Enthält die Vorgangsnummern aller Endvorgänge.

-**vorgaenge: HashMap¡Integer,Vorgang¿** Enthält alle Vorgänge des Netzplans. Der Schlüssel ist die Vorgangsnummer jedes Vorgangs.

-**eing: Eingabe** Dient als Schnittstelle zur Eingabe.

-**ausg: Ausgabe** Dient als Schnittstelle zur Ausgabe.

Methoden der Klasse Netzplanerstellung

+**Netzplanerstellung(inputPath: String, outputPath: String, boolean log)** Der Konstruktor erzeugt eine Instanz der Klasse Netzplanerstellung. Die Variable inputPath enthält den Pfad der Eingabedatei. Die Variable outputPath den Pfad der Ausgabedatei. Der boolsche Wert log gibt an, ob es eine Ausgabe in der Konsole geben soll.

+**run()** Die Methode steuert den Ablauf des Programms. Sie leitet zunächst das Einlesen der Eingabedaten ein. Tritt ein Fehler in den Eingangsdaten auf, wird dieser abgefangen und die Daten der Eingabedatei zusammen mit dem aufgetretenen Fehler in die Ausgabedatei geschrieben. Ist 'log' gleich true, werden die Informationen zusätzlich in der Konsole ausgegeben. Anschließend wird das Programm mit dem Exitcode 1 beendet. Tritt kein Fehler in den Eingabedaten auf, wird der eingelesene Netzplan überprüft. Tritt ein Fehler im Netzplan auf, werden wie bei einem Fehler in den Eingabedaten diese zusammen mit dem aufgetretenden Fehler in die Ausgabedatei geschrieben und das Programm beendet. Tritt kein Fehler auf, wird die erste Phase für alle Endvorgänge aufgerufen. Nachdem dies geschehen ist, sind der FAZ und der FEZ jedes Vorgangs berechnet. Anschließend wird die zweite Phase des Algorithmus für alle Startvorgänge aufgerufen. Ist dies erfolgt, sind alle fehlenden Werte für jeden Vorgang berechnet. Daraufhin werden die kritischen Pfade ermittelt. Abschließend erfolgt das Erstellen der Ausgabestrings und das Schreiben in die Ausgabedatei und ggf. Konsole.

+**erstePhase(v: Vorgang): double** Diese Methode berechnet rekursiv mit Hilfe des Backtracking-Verfahrens die Werte FAZ und FEZ. Sie ruft sich selbst solange mit den Vorgängern des aktuellen Vorgangs auf, bis sie an einem Startvorgang angekommen ist.

+**zweitePhase(v: Vorgang): double** Diese Methode berechnet rekursiv mit Hilfe des Backtracking-Verfahrens alle anderen Werte, also SAZ,SEZ,GP,FP und ob der aktuelle Vorgang kritisch ist. Sie ruft sich selbst solange mit den Nachfolgern des aktuellen Vorgangs auf, bis sie an einem Endvorgang angekommen ist.

+**getKritischenPfad(v: Vorgang)** Ermittelt ausgehend von einem Startvorgang alle kritischen Pfade von diesem Startvorgang aus und speichert diese in der Variablen kritischePfade.

+**pruefeNetzplan()** Diese Methode überprüft mit Hilfe der Methoden pruefeZusammenhaen-

gend und pruefeZyklus den Netzplan auf seine inhaltliche Richtigkeit (siehe dazu 1.1).

+pruefeZusammenhaengend(endvorgaenge: ArrayList¡Integer¿): boolean Diese Methode überprüft den Netzplan darauf, ob dieser zusammenhängt.

+getEndvorgaenge(v: Vorgang): ArrayList¡Integer¿ Diese Methode berechnet zu einem Startvorgang rekursiv alle Endvorgaenge, die von diesem Startvorgang aus erreicht werden können und gibt diese zurück.

+pruefeZyklus(v: Vorgang, list: ArrayList¡Integer¿) Diese Methode überprüft rekursiv den Netzplan auf Zyklen.

3.1.4 Die Klasse Ausgabe

Die Klasse Ausgabe dient zum Ausgeben der Ergebnisse in die Ausgabedatei.

Die Attribute der Klasse Ausgabe

-bw: BufferedWriter BufferedWriter zum Schreiben in die Ausgabedatei.

Methoden der Klasse Ausgabe

+Ausgabe(outputPath: String) Der Konstruktor erzeugt eine Instanz der Klasse Ausgabe. Zusätzlich wird die Ausgabedatei mit dem outputPath erzeugt. Existiert bereits eine gleichnamige Datei, wird diese überschrieben.

+schreibe(s: String) Die Methode schreibt den übergebenen String in die Ausgabedatei.

+schreibeException(exception: String) Die Methode schreibt eine Exception in die Ausgabedatei.

+schreibeNetzplanAusgabe(titel: String, vorgaenge: String, av: ArrayList¡Integer¿, ev: ArrayList¡Integer¿, gesamtD: double, kp: ArrayList¡ArrayList¡Integer¿¿) Die Methode schreibt die Ausgabe des Netzplans in die Ausgabedatei.

+logDateiinhalt(s: String) Die Methode schreibt den Dateiinhalt, der im String s enthalten ist in die Konsole.

+logException(exception: String) Die Methode schreibt eine Exception in die Konsole.

+logNetzplanAusgabe(titel: String, vorgaenge: String, av: ArrayList¡Integer¿, ev: ArrayList¡Integer¿, gesamtD: double, kp: ArrayList¡ArrayList¡Integer¿¿) Die Methode schreibt die Ausgabe des Netzplans in die Konsole.

3.1.5 Die Klasse EingabeException

Die Attribute der Klasse EingabeException

keine

Methoden der Klasse EingabeException

+EingabeException(s: String) Der Konstruktor erzeugt eine Instanz der Klasse EingabeException, die von der Klasse RuntimeException abgeleitet ist. Es wird der Super-Konstruktor aufgerufen, dem die Fehlermeldung als Zeichenkette übergeben wird.

3.1.6 Die Klasse NetzplanException

Die Attribute der Klasse NetzplanException

keine

Methoden der Klasse NetzplanException

+NetzplanException(s: String) Der Konstruktor erzeugt eine Instanz der Klasse NetzplanException, die von der Klasse RuntimeException abgeleitet ist. Es wird der Super-Konstruktor aufgerufen, dem die Fehlermeldung als Zeichenkette übergeben wird.

3.1.7 Die Klasse Main

Die Klasse Main startet das Programm.

Die Attribute der Klasse Main

keine

Methoden der Klasse Main

+**main(args: String[])** Die Methode erzeugt ein Objekt der Klasse Netzplanerstellung und führt dessen *run()*-Methode aus.

3.2 Programmablauf – Sequenzdiagramm

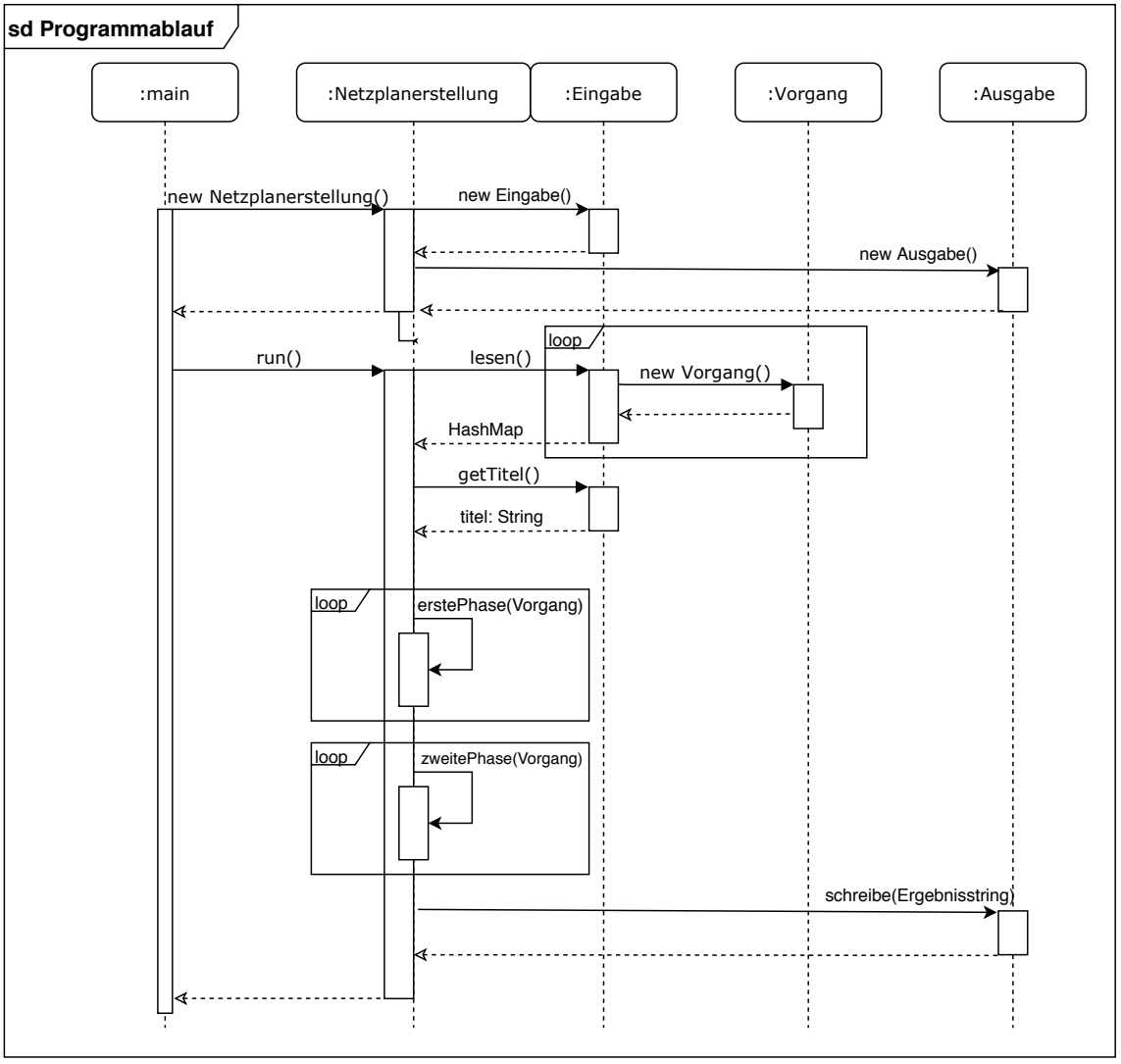

Abbildung 3.2: Das UML-Sequenzdiagramm

3.3 Programmablauf – Nassi-Shneiderman-Diagramme

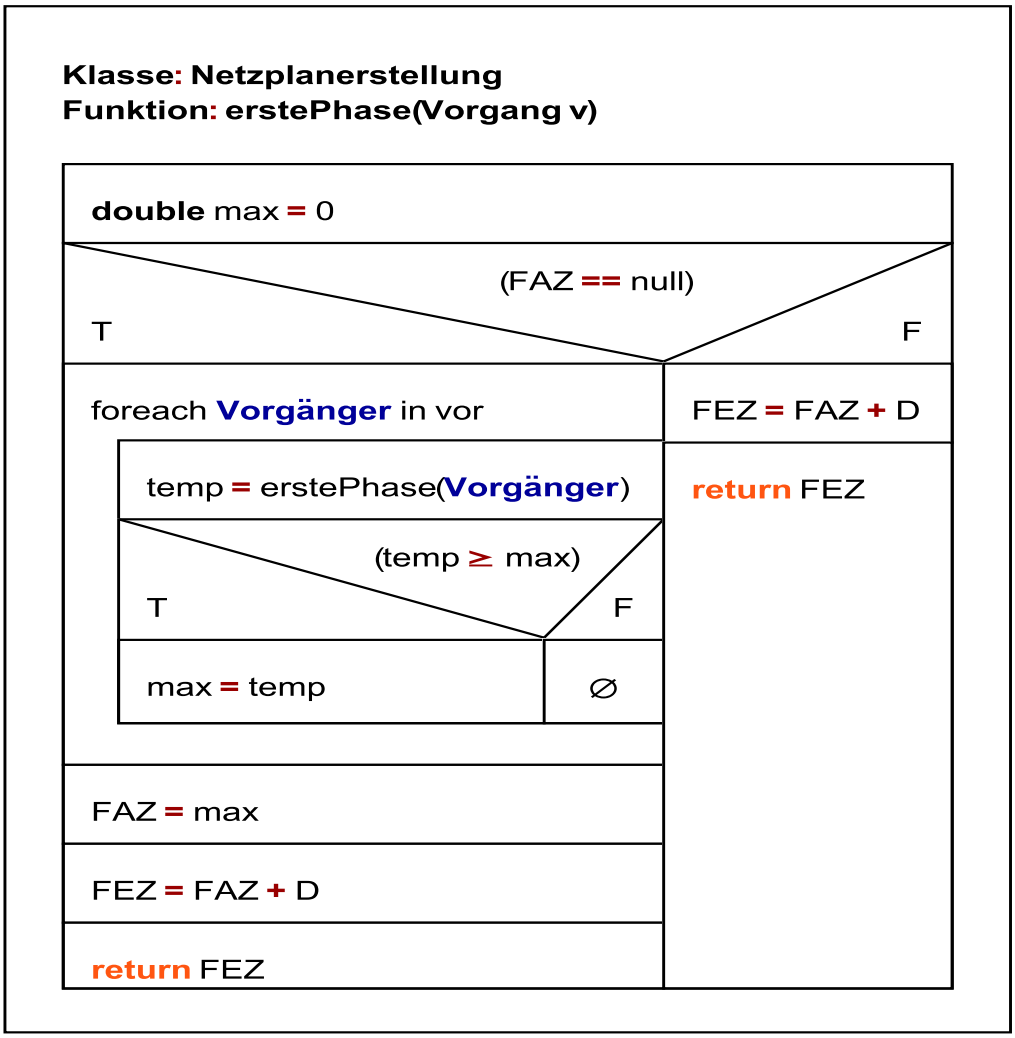

Abbildung 3.3: Die Methode erstePhase(Vorgang v)

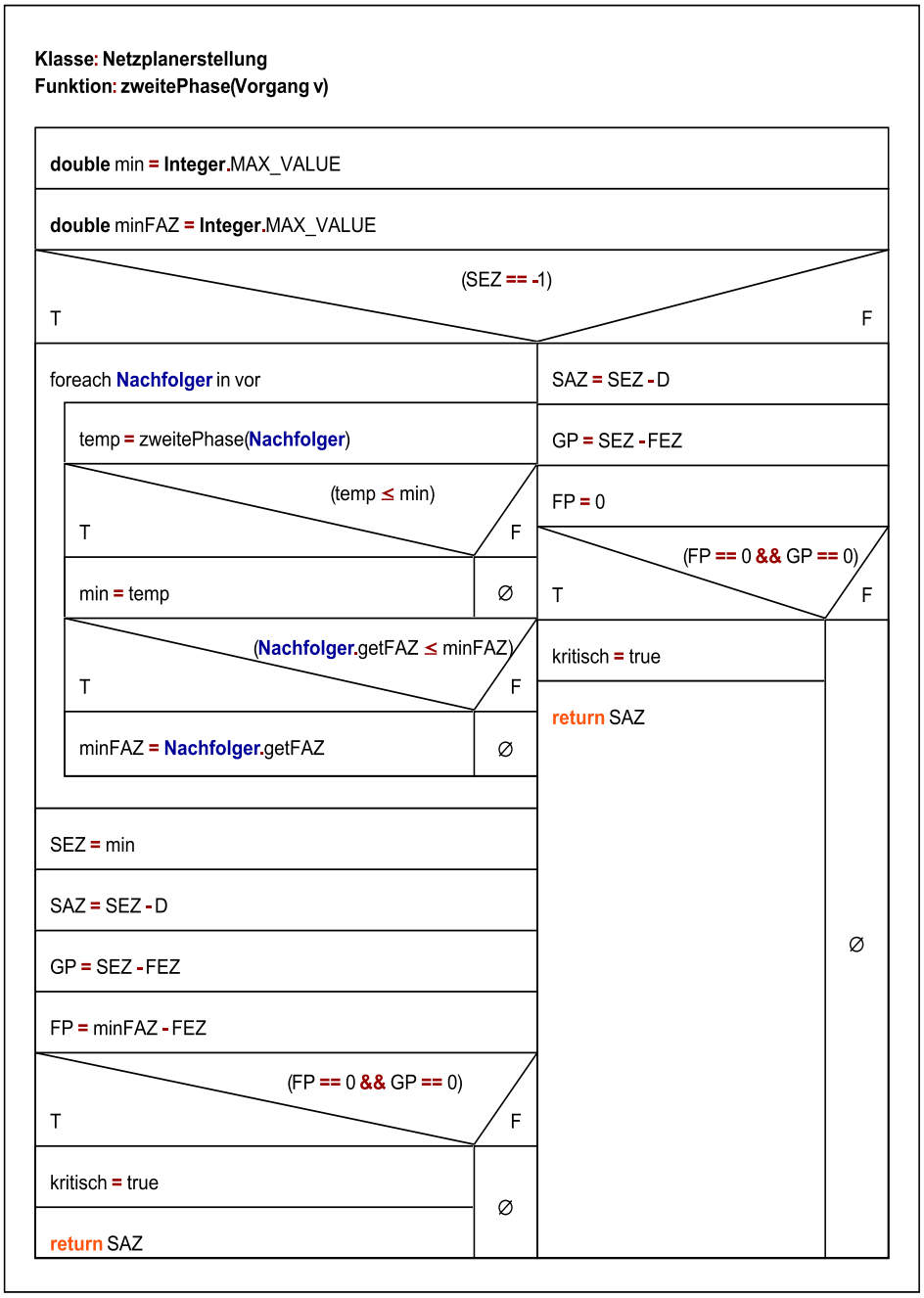

Abbildung 3.4: Die Methode zweitePhase(Vorgang v)

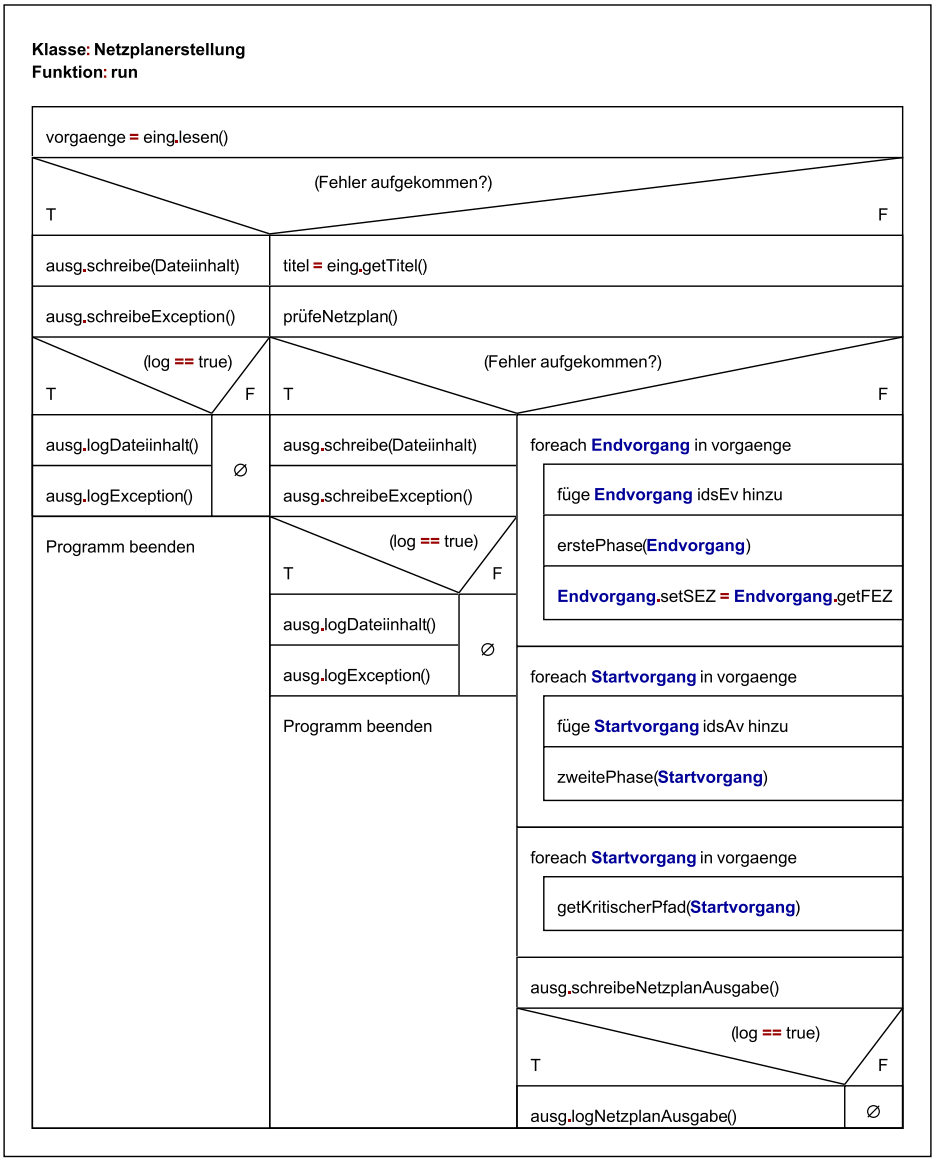

Abbildung 3.5: Die Methode run()

Kapitel 4

Testdokumentation

Die folgenden Testbeispiele sollen zeigen, dass das Programm einwandfrei funktioniert. Es wurden Beispiele gewählt, die alle Programmzweige abdecken. Es wurden Black-Box- und White-Box-Tests durchgeführt. Die Testbeispiele sind in Normalfälle, Sonderfälle und Fehlerfälle eingeteilt.
Normalfälle haben eine gültige Eingabedatei und werden fehlerfrei bearbeitet. Sonderfälle ebenso, allerdings stellen sie Besonderheiten bei der Eingabe oder Verarbeitung dar. Diese werden ebenfalls korrekt abgearbeitet. Die Fehlerfälle stellen formale oder logische Fehler bei der Dateneingabe dar und führen zum Ende des Programms mit einer entsprechenden Fehlermeldung.

Die Eingabedateien sind, nach Normal-, Sonder- und Fehlerfällen getrennt, im Ordner Beispiele zu finden. Die dazugehörigen Ausgabedateien mit dem gleichen Namen und der Endung „.out“ werden erzeugt und im gleichen Ordner gespeichert.

Folgende Eingabedateien wurden getestet:

- Normalfälle
  - IHK1.in
  - IHK2.in
  - IHK3.in
  - IHK5.in
  - Normal_GroßerVerzweigterNetzplan.in
  - Normal_MehrereStartUndEndvorgänge.in

- Sonderfälle
  - Sonder_Zusammenhängend.in
  - Sonder_MaximalBeispiel.in
  - Sonder_NurStartUndEndvorgang.in
  - Sonder_VieleAbzweigungen
- Fehlerfälle
  - IHK4.in
  - Fehler_DatentypDauer.in
  - Fehler_DatentypID.in
  - Fehler_DatentypNachfolger.in
  - Fehler_DatentypVorgänger.in
  - Fehler_GleicheID.in
  - Fehler_GleicherVorgängerNachfolger.in
  - Fehler_IDAlsNachfolger.in
  - Fehler_IDAlsVorgänger.in
  - Fehler_IDInNachfolgerNichtExistent.in
  - Fehler_IDInVorgängerNichtExistent.in
  - Fehler_KeinAnfangsvorgang.in
  - Fehler_KeinEndvorgang.in
  - Fehler_KeineVorgänge.in
  - Fehler_MehrereTitel.in
  - Fehler_MehrereVorgängerOhneKommata.in
  - Fehler_NachfolgerVorgängerPassenNicht.in
  - Fehler_NegativeDauer.in
  - Fehler_NegativeID.in
  - Fehler_NetzplanNichtZusammenhängend.in
  - Fehler_VorgängerNachfolgerPassenNicht.in
  - Fehler_ZuVieleSemikolons.in
  - Fehler_ZuWenigeSemikolons.in
  - Fehler_Zyklus.in

Für die Beschreibung der Testfälle wird davon ausgegangen, dass bereits die korrigierte Version der IHK-Prüfung vorliegt. Da diese dennoch zum Teil weitere Fehler enthält, ist jedem betroffenen Beispiel eine Anmerkung hinzugefügt.

4.1 Normalfälle

4.1.1 IHK1.in

Dieses Beispiel ist das erste Beispiel aus der Aufgabenstellung der IHK. Die korrekte Bearbeitung des Programms konnte anhand der Musterlösung in der Aufgabenstellung geprüft werden. Wichtige Punkte der Arbeitsweise des Algorithmus werden an diesem Beispiel im Folgenden dargestellt. Bei den anderen Beispielen wird dann jeweils der Unterschied oder die Besonderheit zum 'Normalfall' beschrieben.

Nachdem die Log-Bedingung und der Dateipfad der Eingabe- und ggf. der Ausgabedatei dem Programm beim Aufruf als Parameter übergeben wurden, wird in der main-Methode der Main-Klasse ein Objekt vom Typ Netzplanerstellung erstellt. Das heißt, der Konstruktor der Klasse Netzplanerstellung wird mit dem boolean log und dem Eingabe- und ggf. auch Ausgabepfad als Parameter aufgerufen. Die Klasse Netzplanerstellung erstellt im Konstruktor jeweils ein Eingabe und Ausgabe Objekt und speichert den boolean log ab. Anschließend wird die *run()*-Methode der Klasse Netzplanerstellung ausgeführt. Dort wird die Methode *lese()* der Klasse Eingabe ausgeführt. Kann die Eingabedatei nicht gefunden werden, wird ein entsprechender Fehler ausgegeben. Die Methode liest die Eingabedaten zeilenweise ein, wobei Leerzeilen und Zeilen die mit '\\' beginnen ignoriert werden. Da die zweite Zeile der Eingabedatei mit '\\+' beginnt, wird diese als Titel erkannt. Es wird geprüft ob bereits ein Titel eingelesen wurde. Wenn nicht, wird der Titel ohne Leerzeichen am Anfang und Ende gespeichert. Bei allen anderen Zeilen wird ein Vorgang eingelesen. Als erstes werden Leerzeichen am Anfang und Ende der Zeile entfernt. Anschließend wird die Zeile an ihren Semikolons geteilt und es wird geprüft, ob fünf Teilstrings entstanden sind. Ist dies nicht der Fall, stehen in der Zeile zu wenig oder zu viele Semikolons und es wird ein Fehler ausgegeben. Sind genau fünf Teilstrings entstanden, werden die Leerzeichen am Anfang und Ende des ersten Teilstrings entfernt und es wird versucht den übrig gebliebenen String zu einem Integer zu parsen. Funktioniert dies nicht, wird ein Fehler ausgegeben, da die Vorgangsnummer nicht den richtigen Datentyp besitzt. Ist der Datentyp richtig, wird überprüft ob die Id negativ ist oder bereits vorkam. Tritt einer der Fälle ein wird ebenfalls ein entsprechender Fehler geworfen. Die Vorgangsbezeichnung, also der zweite Teilstring, wird nach dem Entfernen der Leerzeichen am Anfang und am Ende übernommen. Mit dem dritten Teilstring, also der Dauer des Vorgangs wird wie mit der ID verfahren, mit dem Unterschied, dass diese zu Double geparst wird. Außerdem wird die Dauer auf die zweite Nachkommastelle abgeschnitten. Beim Teilstring für die Vorgänger werden ebenfalls erst die Leerzeichen am Anfang und am Ende des Strings entfernt. Anschließend wird geprüft ob der Teilstring nur noch ein minus enthält, was bedeutet, dass der Vorgang keine Vorgänger besitzt und damit ein

Startvorgang ist. Ist dies nicht der Fall, wird der Teilstring an den Kommata getrennt, sodass für jeden Vorgänger erneut Teilstrings entstehen. Für jeden Vorgänger werden nun die Leerzeichen am Anfang und am Ende entfernt. Sind danach immer noch Leerzeichen enthalten, kann ein möglicher Tippfehler dafür der Fall sein. In diesem Fall wird ein Fehler ausgegeben. Anschließend wird versucht den String des Vorgängers in einen Integer zu Parsen. Wie bei der Vorgangsnummer und der Dauer, wird ein Fehler ausgegeben, wenn der Datentyp nicht stimmt. Desweiteren wird geprüft, ob der Vorgang sein eigener Vorgänger ist und dann ein Fehler ausgegeben. Mit dem letzten Teilstring, also den Nachfolgern, wird wie mit den Vorgängern verfahren. Sind keine Fehler aufgetreten, wird ein Objekt der Klasse Vorgang mit allen Parametern erstellt und in einer HashMap mit der Vorgangsnummer als Schlüssel gespeichert. Nachdem alle Zeilen durchlaufen wurden, wird überprüft ob die Liste von Vorgängern oder Nachfolgern eine ID besitzt, die nicht eingegeben wurde und ob jede Nachfolger-Vorgänger Beziehung stimmt. Außerdem wird überprüft, ob in der gesamten Datei ein Titel stand und ob überhaupt Vorgänge enthalten waren. Enthielt die Datei keinen Titel wird ein Fehler für diesen ausgegeben und das Programm beendet sich. Da die Eingabedatei dem vorgegebenen Format entspricht, kann das Beispiel fehlerfrei verarbeitet werden.

Eingabedatei:

```
//****************************************
//+ Installation von POI Kiosken
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; Planung des Projekts; 1; -; 2,4
2; Beschaffung der POI-Kioske; 25; 1; 3
3; Einrichtung der POI-Kioske; 10; 2; 6
4; Netzwerk installieren; 6; 1; 5
5; Netzwerk einrichten; 1; 4; 6
6; Aufbau der POI Kioske; 2; 3,5; 7
7; Tests und Nachbesserungen der POI Kioske; 1; 6; -
```

Da das Einlesen der Daten fehlerfrei verlaufen ist, kann die *run()*-Methode nun den Netzplan mit Hilfe der Methode *pruefeNetzplan()* auf Richtigkeit prüfen. Diese Methode kontrolliert erst, ob es weniger als zwei Vorgänge, keinen Start- oder Endvorgang, oder einen Vorgang gibt, der Start- und Endvorgang ist. Tritt einer dieser Fälle ein, gibt das Programm eine entsprechende Fehlermeldung aus. Anschließend wird kontrolliert ob ein Zyklus in dem Netzplan vorhanden ist. Dazu wird der Netzplan von allen Endvorgängen zu den Startvorgängen durchlaufen und bei jedem Vorgang geprüft ob dieser in dem gerade betrachtetenden Pfad bereits Nachfolger eines anderen Vorgangs war. Ist, wie in die-

sem Beispiel, kein Zyklus im Netzplan enthalten, wird anschließend überprüft ob der Netzplan zusammenhängt. Dazu wird für jeden Startvorgang die Menge der erreichbaren Endvorgängen ermittelt. Da es im Beispiel nur einen Start- und Endvorgang gibt, wird ermittelt, dass der Startvorgang mit der ID 1 den Endvorgang mit der ID 7 besitzt:

```
1 -> [7]
```

In diesem Fall muss der Netzplan zusammenhängen und es wird True zurück gegeben. Gibt es mehr als nur einen Startvorgang, wird die Überprüfung komplexer. Der Grundgedanke des Verfahrens ist es, dass nur Startvorgänge die in mehr als einem Endvorgang enden, zwei Netzpläne miteinander verbinden können. Gibt es also mehr als einen Startvorgang, aber alle Startvorgänge enden nur in einem Endzustand, müssen alle Startvorgänge in dem selben Endvorgang enden, da sonst der Netzplan nicht verbunden ist. Für den Fall, dass es Startvorgänge gibt, die in mehreren Endvorgängen enden, wird ein Startvorgang ausgewählt. Zusammen mit diesem wird dann für jeden anderen Startvorgang überprüft, ob diese jeweils über mindestens einen gemeinsamen Endvorgang miteinander verbunden sind. Trifft dies zu, können nun auch die Endzustände des zweiten Startvorgangs erreicht werden. So ergibt sich am Ende eine Menge aller erreichbaren Endvorgänge. Diese muss gleich der Menge aller existierenden Endvorgänge sein, sonst ist der Netzplan nicht zusammenhängend.

Enthält der Netzplan, wie im Beispiel, keinen Fehler, wird von jedem Endvorgang aus die Methode *erstePhase(Vorgang v)* aufgerufen. Diese wird für jeden Vorgänger des aktuellen Vorgangs erneut aufgerufen bis der aktuelle Vorgang ein Startvorgang oder ein bereits behandelter Vorgang ist. Nun werden auf dem rekursiven Rückweg die Werte FAZ und FEZ berechnet, wobei bei mehreren Vorgängern immer der größere FEZ übernommen wird.

Für das Beispiel sieht dies wie folgt aus:

```
starte-erstePhase(Vorgang7)
  starte-erstePhase(Vorgang6)
     starte-erstePhase(Vorgang3)
        starte-erstePhase(Vorgang2)
           starte-erstePhase(Vorgang1)
           beende-erstePhase(Vorgang1)->Startvorgang. FAZ=0, FEZ=1
        beende-erstePhase(Vorgang2)->FAZ=1, FEZ = 26
     beende-erstePhase(Vorgang3)->FAZ=26, FEZ=36
     starte-erstePhase(Vorgang5)
        starte-erstePhase(Vorgang4)
           starte-erstePhase(Vorgang1)
           beende-erstePhase(Vorgang1)->Startvorgang. FAZ=0, FEZ=1
        beende-erstePhase(Vorgang4)->FAZ=1, FEZ=7
     beende-erstePhase(Vorgang5)->FAZ=7, FEZ=8
  beende-erstePhase(Vorgang6)->FAZ=36, da 36>8, FEZ=38
beende-erstePhase(Vorgang7)->FAZ=38, FEZ=39
```

Anschließend wird für jeden Startvorgang die Methode *zweitePhase(Vorgang v)* aufgerufen. Diese wird für jeden Nachfolger des aktuellen Vorgangs erneut aufgerufen, bis der aktuelle Vorgang ein Endvorgang oder ein bereits behandelter Vorgang ist. Nun werden auf dem rekursiven Rückweg alle restlichen Werte, also SAZ, SEZ, GP, FP und kritisch berechnet. Bei mehreren Nachfolgern wird immer der kleinere SEZ übernommen. Für das Beispiel sieht dies wie folgt aus:

```
starte-zweitePhase(Vorgang1)
  starte-zweitePhase(Vorgang2)
    starte-zweitePhase(Vorgang3)
      starte-zweitePhase(Vorgang6)
        starte-zweitePhase(Vorgang7)
        beende-zweitePhase(Vorgang7)->Endvorgang. SEZ=39, SAZ=38, GP=0, FP=0
      beende-zweitePhase(Vorgang6)->SEZ=38, SAZ=36, GP=0, FP=0
    beende-zweitePhase(Vorgang3)->SEZ=36, SAZ=26, GP=0, FP=0
  beende-zweitePhase(Vorgang2)->SEZ=26, SAZ=1, GP=0, FP=0
  starte-zweitePhase(Vorgang4)
    starte-zweitePhase(Vorgang5)
      starte-zweitePhase(Vorgang6)
      beende-zweitePhase(Vorgang6)->Behandelt. SEZ=38, SAZ=36, GP=0, FP=0
    beende-zweitePhase(Vorgang5)->SEZ=36, SAZ=35, GP=28, FP=28
  beende-zweitePhase(Vorgang4)->SEZ=35, SAZ=29, GP=28, FP=0
beende-zweitePhase(Vorgang1)->SEZ=1, SAZ=0, GP=0, FP=0
```

Sind alle Werte für alle Vorgänge berechnet, ruft die *run()*-Methode die rekursive Methode *getKritischenPfad(Vorgang v)* für jeden Startvorgang auf. Diese berechnet die kritischen Pfade des Netzplans und speichert sie in der Liste kritischePfade. Für das Beispiel entsteht der folgende kritische Pfad:

```
1->2->3->6->7
```

Abschließend erstellt die *run()*-Methode Strings für die Vorgänge, berechnet Informationen wie die Gesamtdauer und übergibt diese der Methode *schreibeNetzplanAusgabe()* der Klasse Ausgabe, die diese in die Ausgabedatei schreibt.

Ausgabedatei:

```
Installation von POI Kiosken

Vorgangsnummer; Vorgangsbezeichnung; D; FAZ; FEZ; SAZ; SEZ; GP; FP
1; Planung des Projekts; 1.0; 0.0; 1.0; 0.0; 1.0; 0.0; 0.0
2; Beschaffung der POI-Kioske; 25.0; 1.0; 26.0; 1.0; 26.0; 0.0; 0.0
3; Einrichtung der POI-Kioske; 10.0; 26.0; 36.0; 26.0; 36.0; 0.0; 0.0
4; Netzwerk installieren; 6.0; 1.0; 7.0; 29.0; 35.0; 28.0; 0.0
5; Netzwerk einrichten; 1.0; 7.0; 8.0; 35.0; 36.0; 28.0; 28.0
6; Aufbau der POI Kioske; 2.0; 36.0; 38.0; 36.0; 38.0; 0.0; 0.0
7; Tests und Nachbesserungen der POI Kioske; 1.0; 38.0; 39.0; 38.0; 39.
0.0; 0.0

Anfangsvorgang: 1
Endvorgang: 7
Gesamtdauer: 39.0

Kritischer Pfad:
1. 1->2->3->6->7
```

4.1.2 IHK2.in

Dieses Beispiel ist ebenfalls der Aufgabenstellung der IHK zu entnehmen. Es unterscheidet sich zum ersten Beispiel darin, dass es keine Verzweigungen des Netzplans gibt, also jeder Vorgang nicht mehr als einen Vorgänger bzw. Nachfolger hat.

Eingabedatei:

Anmerkung: Die Eingabedatei des zweiten Beispiels enthält zwei Fehler, die korrigiert wurden. In dem originalen Beispiel besitzt der Vorgang 6 den Vorgänger 7. Da dieser nicht existiert wurde dieser in der Eingabedatei in den 5. Vorgang geändert. Außerdem steht im fünften Vorgang ein Semikolon als letztes Zeichen. Das Programm akzeptiert dieses wenn hinter diesem keine anderen Zeichen als Leerzeichen folgen.

```
//****************************************
//+ Beispiel 2 - Wasserfallmodell
//****************************************
```

```
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; Problemanalyse; 2; -; 2
2; Grobplanung; 3; 1; 3
3; Feinplanung; 3; 2; 4
4; Implementierung; 10; 3; 5
5; Testphase; 5; 4; 6;
6; Einsatz und Wartung; 5; 5; -
```

Da der Netzplan ebenfalls nur einen Start- und Endvorgang und auch keine Zyklen besitzt, ändert sich der Programmablauf nicht, außer das durch die fehlenden Verzweigungen das Programm schneller arbeitet, da es nicht so viele Pfade ablaufen muss. Für das Beispiel gibt es einen kritischen Pfad, der wie folgt aussieht:

```
1->2->3->4->5->6
```

Ausgabedatei:

```
Beispiel 2 - Wasserfallmodell

Vorgangsnummer; Vorgangsbezeichnung; D; FAZ; FEZ; SAZ; SEZ; GP; FP
1; Problemanalyse; 2.0; 0.0; 2.0; 0.0; 2.0; 0.0; 0.0
2; Grobplanung; 3.0; 2.0; 5.0; 2.0; 5.0; 0.0; 0.0
3; Feinplanung; 3.0; 5.0; 8.0; 5.0; 8.0; 0.0; 0.0
4; Implementierung; 10.0; 8.0; 18.0; 8.0; 18.0; 0.0; 0.0
5; Testphase; 5.0; 18.0; 23.0; 18.0; 23.0; 0.0; 0.0
6; Einsatz und Wartung; 5.0; 23.0; 28.0; 23.0; 28.0; 0.0; 0.0

Anfangsvorgang: 1
Endvorgang: 6
Gesamtdauer: 28.0

Kritischer Pfad:
1. 1->2->3->4->5->6
```

4.1.3 IHK3.in

Das folgende Beispiel ist ein Netzplan mit mehreren Start- und Endvorgängen.

Eingabedatei:

Anmerkung: Die Eingabedatei des dritten Beispiels enthält zwei Fehler. In dem originalen Beispiel steht im fünften Vorgang ein Semikolon als letztes Zeichen. Das Programm akzeptiert dieses wenn hinter diesem keine anderen Zeichen als Leerzeichen folgen. Desweiteren ist in der Ausgabedatei der SEZ des 8. Vorgangs falsch. Dieser muss 12 und nicht 13 betragen.

```
//*****************************************
//+ Beispiel 3
//*****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; Kaffee mahlen; 1; -; 2
2; Kaffee in Trichter füllen; 1; 1; 4
3; Wasser kochen; 4; -; 4,7
4; Kaffee brühen; 6; 2,3; 5
5; Kaffee trinken; 3; 4; -;
6; Tee in Beutel füllen; 1; -; 7
7; Tee aufsetzen; 5; 3,6; 8
8; Tee trinken; 3; 7; -
```

Das es in diesem Beispiel mehr als einen Startvorgang gibt, ändert sich die Vorgehensweise bei der Überprüfung der Zusammengehörigkeit des Netzplans. Die Ermittlung der erreichbaren Endvorgänge für jeden Startvorgang ergibt in diesem Fall die folgende Liste:

```
1 -> [5]
3 -> [5, 8]
6 -> [8]
```

In diesem Fall wird als erstes der Startvorgang mit der ID 3 betrachtet, da dieser in mehr als einem Endvorgang endet. Nun ist die Menge der erreichbaren Endvorgänge [5, 8]. Da kein weiterer Startvorgang existiert, der in mehr als einem Endvorgang endet, wird nun überprüft, ob die Menge der erreichbaren Endvorgängen gleich der Menge aller Endvorgänge ist. Da dies der Fall ist, ist der Netzplan zusammenhängend und es wird kein Fehler geworfen. Die ermittelten kritischen Pfade sind für dieses Beispiel:

```
3->4->5
3->7->8
```

Ausgabedatei:

```
Beispiel 3

Vorgangsnummer; Vorgangsbezeichnung; D; FAZ; FEZ; SAZ; SEZ; GP; FP
1; Kaffee mahlen; 1.0; 0.0; 1.0; 2.0; 3.0; 2.0; 0.0
2; Kaffee in Trichter füllen; 1.0; 1.0; 2.0; 3.0; 4.0; 2.0; 2.0
3; Wasser kochen; 4.0; 0.0; 4.0; 0.0; 4.0; 0.0; 0.0
4; Kaffee brühen; 6.0; 4.0; 10.0; 4.0; 10.0; 0.0; 0.0
5; Kaffee trinken; 3.0; 10.0; 13.0; 10.0; 13.0; 0.0; 0.0
6; Tee in Beutel füllen; 1.0; 0.0; 1.0; 3.0; 4.0; 3.0; 3.0
7; Tee aufsetzen; 5.0; 4.0; 9.0; 4.0; 9.0; 0.0; 0.0
8; Tee trinken; 3.0; 9.0; 12.0; 9.0; 12.0; 0.0; 0.0

Anfangsvorgang: 1,3,6
Endvorgang: 5,8
Maximale Gesamtdauer (da mehrere Endzustaende): 13.0

Kritische Pfade:
1. 3->4->5
2. 3->7->8
```

4.1.4 IHK5.in

Dieses Beispiel ist ebenfalls der Aufgabenstellung der IHK zu entnehmen. Es unterscheidet sich zum ersten Beispiel nur darin, dass es größer und komplexer ist.

Eingabedatei:

Anmerkung: Die Eingabedatei des zweiten Beispiels enthält einen Fehler, die korrigiert wurden. In dem originalen Beispiel besitzt der Vorgang 4 den Nachfolger 8, der Vorgang 8 jedoch nicht den Vorgänger 4, sondern den Vorgang 3, der wiederum nicht den Nachfolger 8 besitzt. Aus diesem Grund wurde in der Eingabedatei der Nachfolger des 8. Vorgangs von 3 zu 4 geändert. Dadurch ändert sich die Ausgabe des Beispiels, weshalb die hier angegebene Ausgabe von der des IHK-Beispiels abweicht.

```
//****************************************
//+ Beispiel 5 IT-Installation
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; Infrastrukturbedarf ermitteln; 1; -; 2,3
2; Arbeitsplatzbedarf ermitteln; 2; 1; 5
3; Netzwerkplan entwerfrn; 1; 1; 4,7
4; Peripheriebedarf ermitteln; 1; 3; 8
5; Hardware PC + Server beschaffen; 4; 2; 6,7,9
6; Software beschaffen; 2; 5; 12
7; Netzwerkzubehör beschaffen; 5; 3,5; 11
8; Peripherie beschaffen; 1; 4; 13
9; Hardware PC + Server aufbauen; 6; 5; 10
10; Server installieren; 3; 9; 12,13,15
11; Netzwerk aufbauen; 5; 7; 12,13
12; PC-Image anlegen; 1; 6, 10,11; 16
13; Peripherie anschließen; 1; 8,10,11; 14
14; Netzwerkplan dokumentieren; 2; 13; 17
15; Server-Image anlegen; 1; 10; 17
16; PC-Remote installieren; 1; 12; 17
17; Gesamtdokumenation erstellen; 3; 14,15,16; -
```

Da der Netzplan ebenfalls nur einen Start- und Endvorgang und auch keine Zyklen besitzt, ändert sich der Programmablauf nicht, außer das durch den größeren Netzplan das Programm mehr Pfade durchlaufen muss. Für das Beispiel gibt es einen kritischen Pfad, der wie folgt aussieht:

```
1->2->5->7->11->13->14->17
```

Ausgabedatei:

```
Beispiel 5 IT-Installation

Vorgangsnummer; Vorgangsbezeichnung; D; FAZ; FEZ; SAZ; SEZ; GP; FP
1; Infrastrukturbedarf ermitteln; 1.0; 0.0; 1.0; 0.0; 1.0; 0.0; 0.0
2; Arbeitsplatzbedarf ermitteln; 2.0; 1.0; 3.0; 1.0; 3.0; 0.0; 0.0
3; Netzwerkplan entwerfrn; 1.0; 1.0; 2.0; 6.0; 7.0; 5.0; 0.0
4; Peripheriebedarf ermitteln; 1.0; 2.0; 3.0; 15.0; 16.0; 13.0; 0.0
5; Hardware PC + Server beschaffen; 4.0; 3.0; 7.0; 3.0; 7.0; 0.0; 0.0
```

```
6; Software beschaffen; 2.0; 7.0; 9.0; 16.0; 18.0; 9.0; 8.0
7; Netzwerkzubehör beschaffen; 5.0; 7.0; 12.0; 7.0; 12.0; 0.0; 0.0
8; Peripherie beschaffen; 1.0; 3.0; 4.0; 16.0; 17.0; 13.0; 13.0
9; Hardware PC + Server aufbauen; 6.0; 7.0; 13.0; 8.0; 14.0; 1.0; 0.0
10; Server installieren; 3.0; 13.0; 16.0; 14.0; 17.0; 1.0; 0.0
11; Netzwerk aufbauen; 5.0; 12.0; 17.0; 12.0; 17.0; 0.0; 0.0
12; PC-Image anlegen; 1.0; 17.0; 18.0; 18.0; 19.0; 1.0; 0.0
13; Peripherie anschließen; 1.0; 17.0; 18.0; 17.0; 18.0; 0.0; 0.0
14; Netzwerkplan dokumentieren; 2.0; 18.0; 20.0; 18.0; 20.0; 0.0; 0.0
15; Server-Image anlegen; 1.0; 16.0; 17.0; 19.0; 20.0; 3.0; 3.0
16; PC-Remote installieren; 1.0; 18.0; 19.0; 19.0; 20.0; 1.0; 1.0
17; Gesamtdokumenation erstellen; 3.0; 20.0; 23.0; 20.0; 23.0; 0.0; 0.0

Anfangsvorgang: 1
Endvorgang: 17
Gesamtdauer: 23.0

Kritischer Pfad:
1. 1->2->5->7->11->13->14->17
```

4.1.5 Normal_GroßerVerzweigterNetzplan.in

Dieses Beispiel besteht aus 16 Vorgängen und mehreren Verzweigungen. Es zeigt, dass das Programm bei größteren und komplexeren Verzweigungen korrekt arbeitet.

Eingabedatei:

```
//****************************************
//+ Normal_GroßerVerzweigterNetzplan
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 10; -; 2,3,4,5,6
2; B; 5; 1; 7
3; C; 4; 1; 7,11
4; D; 8; 1; 8
5; E; 7; 1; 12
```

```
6; F; 5; 1; 9
7; G; 5; 2,3; 10
8; H; 2; 4; 11
9; I; 5; 6; 12,13
10; J; 8; 7; 14
11; K; 10; 3,8; 14,15
12; L; 5; 5,9; 15
13; M; 5; 9; 15
14; N; 9; 10,11; 16
15; O; 20; 11,12,13; 16
16; P; 10; 14,15; -
```

Da der Netzplan ebenfalls nur einen Start- und Endvorgang und auch keine Zyklen besitzt, ändert sich der Programmablauf nicht, außer das durch den größeren Netzplan das Programm mehr Pfade durchlaufen muss. Für das Beispiel gibt es einen kritischen Pfad, der wie folgt aussieht:

```
1->4->8->11->15->16
```

Ausgabedatei:

```
Normal_GroßerVerzweigterNetzplan

Vorgangsnummer; Vorgangsbezeichnung; D; FAZ; FEZ; SAZ; SEZ; GP; FP
1; A; 10.0; 0.0; 10.0; 0.0; 10.0; 0.0; 0.0
2; B; 5.0; 10.0; 15.0; 23.0; 28.0; 13.0; 0.0
3; C; 4.0; 10.0; 14.0; 16.0; 20.0; 6.0; 1.0
4; D; 8.0; 10.0; 18.0; 10.0; 18.0; 0.0; 0.0
5; E; 7.0; 10.0; 17.0; 18.0; 25.0; 8.0; 3.0
6; F; 5.0; 10.0; 15.0; 15.0; 20.0; 5.0; 0.0
7; G; 5.0; 15.0; 20.0; 28.0; 33.0; 13.0; 0.0
8; H; 2.0; 18.0; 20.0; 18.0; 20.0; 0.0; 0.0
9; I; 5.0; 15.0; 20.0; 20.0; 25.0; 5.0; 0.0
10; J; 8.0; 20.0; 28.0; 33.0; 41.0; 13.0; 2.0
11; K; 10.0; 20.0; 30.0; 20.0; 30.0; 0.0; 0.0
12; L; 5.0; 20.0; 25.0; 25.0; 30.0; 5.0; 0.0
13; M; 5.0; 20.0; 25.0; 25.0; 30.0; 5.0; 5.0
14; N; 9.0; 30.0; 39.0; 41.0; 50.0; 11.0; 0.0
15; O; 20.0; 30.0; 50.0; 30.0; 50.0; 0.0; 0.0
16; P; 10.0; 50.0; 60.0; 50.0; 60.0; 0.0; 0.0
```

```
Anfangsvorgang: 1
Endvorgang: 16
Gesamtdauer: 60.0

Kritischer Pfad:
1. 1->4->8->11->15->16
```

4.1.6 Normal_MehrereStartUndEndvorgänge.in

Dieses Beispiel besitzt mehrere Start- und Endvorgänge und mehrere kritische Pfade. Es zeigt, dass das Programm bei mehreren kritischen Pfaden und mehreren Start- und Endvorgängen korrekt arbeitet.

Eingabedatei:

```
//*****************************************
//+ Normal_MehrereStartUndEndvorgänge
//*****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 1; -; 3;
2; B; 1; -; 4,5
3; C; 2; 1; 6,7
4; D; 4; 2; 7
5; E; 3; 2; 8,9
6; F; 1; 3; -
7; G; 5; 3,4; -
8; H; 8; 5; 10
9; I; 2; 5; 10
10; J; 3; 8,9; -
```

Ausgabedatei:

```
Normal_MehrereStartUndEndvorgänge

Vorgangsnummer; Vorgangsbezeichnung; D; FAZ; FEZ; SAZ; SEZ; GP; FP
```

```
1; A; 1.0; 0.0; 1.0; 0.0; 1.0; 0.0; 0.0
2; B; 1.0; 0.0; 1.0; 0.0; 1.0; 0.0; 0.0
3; C; 2.0; 1.0; 3.0; 1.0; 3.0; 0.0; 0.0
4; D; 4.0; 1.0; 5.0; 1.0; 5.0; 0.0; 0.0
5; E; 3.0; 1.0; 4.0; 1.0; 4.0; 0.0; 0.0
6; F; 1.0; 3.0; 4.0; 3.0; 4.0; 0.0; 0.0
7; G; 5.0; 5.0; 10.0; 5.0; 10.0; 0.0; 0.0
8; H; 8.0; 4.0; 12.0; 4.0; 12.0; 0.0; 0.0
9; I; 2.0; 4.0; 6.0; 10.0; 12.0; 6.0; 6.0
10; J; 3.0; 12.0; 15.0; 12.0; 15.0; 0.0; 0.0

Anfangsvorgang: 1,2
Endvorgang: 6,7,10
Maximale Gesamtdauer (da mehrere Endzustaende): 15.0

Kritische Pfade:
1. 1->3->6
2. 1->3->7
3. 2->4->7
4. 2->5->8->10
```

4.2 Sonderfälle

4.2.1 Sonder_Zusammenhängend.in

Das folgende Beispiel ist ein Netzplan mit mehreren Start- und Endvorgängen. Der erstellte Netzplan wird jedoch nur durch einen speziellen Pfad verbunden. Es soll verdeutlicht werden, wie genau ein Netzplan auf Zusammengehörigkeit geprüft wird.

Eingabedatei:

```
//***********************************************
//+ Normal_Zusammenhängend      Viele Start- und Endvorgänge,
 zwei Netzpläne die nur über einen Vorgang verbunden sind.
//***********************************************
```

```
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A1; 1; -; 7
2; A2; 1; -; 8
3; A3; 1; -; 8
4; A4; 1; -; 11
5; A5; 1; -; 9
6; A6; 1; -; 16
7; A; 1; 1; 10
8; B; 1; 2,3; 10,11
9; C; 1; 5; 11,15,16
10; D; 1; 7,8; 12
11; E; 1; 8,4,9; 13,14
12; E1; 1; 10; -
13; E2; 1; 11; -
14; E3; 1; 11; -
15; E4; 1; 9; -
16; E5; 1; 9,6; -
```

Graph:

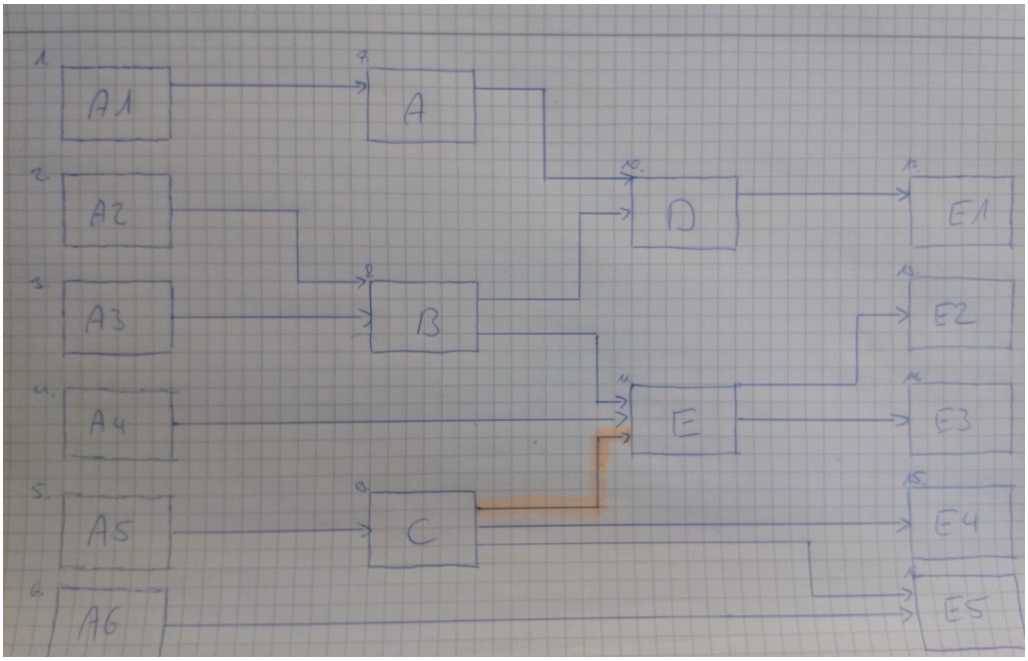

Abbildung 4.1: Graph zum Testbeispiel Normal_Zusammenhängend.in

Das es in diesem Beispiel mehr als einen Startvorgang gibt, ergibt die Ermittlung der erreichbaren Endvorgänge für jeden Startvorgang eine Liste der Länge 6 und dem folgenden Inhalt:

```
1 -> [12]
2 -> [12, 13, 14]
3 -> [12, 13, 14]
4 -> [13, 14]
5 -> [13, 14, 15, 16]
6 -> [16]
```

In diesem Fall wird als erstes der Startvorgang mit der ID 2 betrachtet, da dieser der erste ist, der in mehr als einem Endvorgang endet. Nun ist die Menge der erreichbaren Endvorgänge [12, 13, 14]. Nun wird weiter zu Startvorgang 3 gegangen und geschaut ob einer der Endvorgänge in denen dieser endet sich in der Liste der erreichbaren Endvorgänge befindet. Dies ist hier der Fall, da sowohl 12 und 13 als auch 14 erreichbar sind. Nun würde jeder Endvorgang von Startvorgang 3 der Liste hinzugefügt werden, der noch nicht enthalten ist. Da die beiden Listen jedoch identisch sind, bleibt die Menge der erreichbaren Endvorgänge [12, 13, 14]. Nun wird mit Startvorgang 4 genauso verfahren. Da dieser aber ebenfalls keine neuen Endvorgänge enthält, wird fortgefahren. Startvorgang 5 enthält ebenfalls die Endvorgänge 13 und 14, die in der Liste der erreichbaren Endvorgänge enthalten sind. Daher werden seine weiteren Endvorgänge, also 15 und 16 der Liste hinzugefügt, wodurch sich diese auf [12, 13, 14, 15, 16] erweitert. Da es keine weiteren Startvorgänge mit mehr als einem Endvorgang gibt, wird nun überprüft ob die Menge der erreichbaren gleich der Menge aller Endvorgänge ist. Da dies der Fall ist, ist der Netzplan zusammenhängend. Würde die Verbindung zwischen Vorgang 9 (C) und Vorgang 11 (E), die im Graphen markiert ist, fehlen, würde die Liste des Startvorgangs 5 nur 15 und 16 enthalten. Diese würden nicht der Liste der erreichbaren Endvorgänge hinzugefügt werden und das Programm würde bei der Überprüfung der Menge der erreichbaren Endvorgänge mit der aller Endvorgänge einen Fehler werfen, da der Netzplan nicht zusammenhängen würde.

Die ermittelten kritischen Pfade sind für dieses Beispiel alle möglichen Pfade, die in Endzuständen enden, da für jeden Vorgang nur eine Dauer von 1 benutzt wurde:

```
1->7->10->12
2->8->10->12
2->8->11->13
2->8->11->14
3->8->10->12
```

```
3->8->11->13
3->8->11->14
5->9->11->13
5->9->11->14
5->9->15
5->9->16
```

Ausgabedatei:

```
Normal_Zusammenhängend      Viele Start- und Endvorgänge,
 zwei Netzpläne die nur über einen Vorgang verbunden sind.

Vorgangsnummer; Vorgangsbezeichnung; D; FAZ; FEZ; SAZ; SEZ; GP; FP
1; A1; 1.0; 0.0; 1.0; 0.0; 1.0; 0.0; 0.0
2; A2; 1.0; 0.0; 1.0; 0.0; 1.0; 0.0; 0.0
3; A3; 1.0; 0.0; 1.0; 0.0; 1.0; 0.0; 0.0
4; A4; 1.0; 0.0; 1.0; 1.0; 2.0; 1.0; 1.0
5; A5; 1.0; 0.0; 1.0; 0.0; 1.0; 0.0; 0.0
6; A6; 1.0; 0.0; 1.0; 1.0; 2.0; 1.0; 1.0
7; A; 1.0; 1.0; 2.0; 1.0; 2.0; 0.0; 0.0
8; B; 1.0; 1.0; 2.0; 1.0; 2.0; 0.0; 0.0
9; C; 1.0; 1.0; 2.0; 1.0; 2.0; 0.0; 0.0
10; D; 1.0; 2.0; 3.0; 2.0; 3.0; 0.0; 0.0
11; E; 1.0; 2.0; 3.0; 2.0; 3.0; 0.0; 0.0
12; E1; 1.0; 3.0; 4.0; 3.0; 4.0; 0.0; 0.0
13; E2; 1.0; 3.0; 4.0; 3.0; 4.0; 0.0; 0.0
14; E3; 1.0; 3.0; 4.0; 3.0; 4.0; 0.0; 0.0
15; E4; 1.0; 2.0; 3.0; 2.0; 3.0; 0.0; 0.0
16; E5; 1.0; 2.0; 3.0; 2.0; 3.0; 0.0; 0.0

Anfangsvorgang: 1,2,3,4,5,6
Endvorgang: 12,13,14,15,16
Maximale Gesamtdauer (da mehrere Endzustaende): 4.0

Kritische Pfade:
1. 1->7->10->12
2. 2->8->10->12
3. 2->8->11->13
4. 2->8->11->14
5. 3->8->10->12
```

```
6. 3->8->11->13
7. 3->8->11->14
8. 5->9->11->13
9. 5->9->11->14
10. 5->9->15
11. 5->9->16
```

4.2.2 Sonder_MaximalBeispiel.in

Bei diesem Sonderfall ist der Netzplan wie das Wasserfallmodell aufgebaut und hat 1000 Vorgänge. Die Berechnungen für diesen Netzplan benötigen etwa eine Sekunde. Da ein Projekt in der Realität nur sehr unwahrscheinlich mehr als 1000 Vorgänge haben wird, ist dieses Beispiel als Maximalbeispiel angenommen.

Eingabedatei:
Die Eingabedatei ist in diesem Fall wegen ihrer Größe nicht in das Dokument eingebunden. Sie kann aber im Ordner 'Tests' eingesehen werden.

Ausgabedatei:
Die Ausgabedatei ist in diesem Fall wegen ihrer Größe nicht in das Dokument eingebunden. Sie kann aber im Ordner 'Tests' eingesehen werden.

4.2.3 Sonder_NurStartUndEndvorgang.in

Bei diesem Sonderfall hat der Netzplan keine weiteren Vorgänge außer einen Start- und Endvorgang. Es gibt also nur einen Pfad, der zwingend ein kritischer Pfad ist.

Eingabedatei:

```
//****************************************
//+ Sonderfall - Nur ein Start- und Endvorgang
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; Problemanalyse; 2; -; 2
2; Grobplanung; 3; 1; -
```

Ausgabedatei:

```
Sonderfall - Nur ein Start- und Endvorgang

Vorgangsnummer; Vorgangsbezeichnung; D; FAZ; FEZ; SAZ; SEZ; GP; FP
1; Problemanalyse; 2.0; 0.0; 2.0; 0.0; 2.0; 0.0; 0.0
2; Grobplanung; 3.0; 2.0; 5.0; 2.0; 5.0; 0.0; 0.0

Anfangsvorgang: 1
Endvorgang: 2
Gesamtdauer: 5.0

Kritischer Pfad:
1. 1->2
```

4.2.4 Sonder_VieleAbzweigungen.in

Bei diesem Sonderfall hat der Netzplan viele Verzweigungen.

Eingabedatei:

```
//****************************************
//+ Sonder_VieleAbzweigungen- Jeder zu Jedem ohne Zyklus
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 1; -; 2,3,4,5,6,7,8,9,10
2; B; 1; 1,11,12,13,14,15,16,17,18; -
3; C; 1; 1,11,12,13,14,15,16,17,18; -
4; D; 1; 1,11,12,13,14,15,16,17,18; -
5; E; 1; 1,11,12,13,14,15,16,17,18; -
6; F; 1; 1,11,12,13,14,15,16,17,18; -
7; G; 1; 1,11,12,13,14,15,16,17,18; -
8; H; 1; 1,11,12,13,14,15,16,17,18; -
9; I; 1; 1,11,12,13,14,15,16,17,18; -
10; J; 1; 1, 11,12,13,14,15,16,17,18; -
11; K; 1; -; 2,3,4,5,6,7,8,9,10
12; L; 1; -; 2,3,4,5,6,7,8,9,10
```

```
13; M; 1; -; 2,3,4,5,6,7,8,9,10
14; O; 1; -; 2,3,4,5,6,7,8,9,10
15; P; 1; -; 2,3,4,5,6,7,8,9,10
16; Q; 1; -; 2,3,4,5,6,7,8,9,10
17; R; 1; -; 2,3,4,5,6,7,8,9,10
18; S; 1; -; 2,3,4,5,6,7,8,9,10
```

Ausgabedatei: Die Ausgabedatei ist in diesem Fall wegen ihrer Größe nicht in das Dokument eingebunden. Sie kann aber im Ordner 'Tests' eingesehen werden.

4.3 Fehlerfälle

Fehler die zum Abbruch des Programms führen wurden bereits in Abschnitt 1.3 beschrieben. Anhand eines Black-Box-Tests wird, geprüft, ob die Fehler erkannt und die richtige Fehlermeldung in die Ausgabedatei geschrieben wird. Um die Ausgabe etwas zu verkürzen, wird die Ausgabe der Eingangsdatei hier in der Dokumentation nicht mit angegeben. Sie steht dennoch in der Ausgabedatei.

4.3.1 IHK4.in

Dieses Beispiel ist ebenfalls der Aufgabenstellung der IHK zu entnehmen. Es ist fehlerhaft und enthält einen Zyklus, der sich durch die Vorgänge 3 und 4 zieht.

Eingabedatei:

```
//****************************************
//+ Beispiel 4 mit Zyklus
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; Flughafen planen; 24; -; 2
2; Flughafen bauen; 24; 1; 3
3; Baumängel erkennen; 1; 2; 4
4; Baumängel beseitigen; 6; 3; 3,5
5; Flughafenbau abnehmen und genehmigen; 1; 4; 6
```

```
6; Flugbetrieb aufnehmen; 1; 5; -
```

Das der Netzplan einen Zyklus enthält, erkennt das Programm bereits bei der Überprüfung der Eingabedaten, da kein Vorgang den gleichen Vorgänger und Nachfolger besitzen darf. Für eine genauere Beschreibung der Überprüfung auf Zyklen ist in Abschnitt 4.3.23 zu finden.

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Der Vorgang mit der Vorgangsnummer 4 hat einen gleiche
```

4.3.2 Fehler_DatentypDauer.in

Eingabedatei:

```
//*********************************************************
//+ Fehler_DatentypDauer - Character in Dauer von Vorgang 2
//*********************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 21; -; 2,3
2; B; A; 1; 4
3; C; 7; 1; 6
4; D; 3; 2; 5,6
5; E; 5; 4; 7
6; F; 8; 3,4; 7
7; G; 2; 5,6; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Die Dauer des Vorgangs mit der Vorgangsnummer 2
ist keine Zahl.
```

4.3.3 Fehler_DatentypID.in

Eingabedatei:

```
//*****************************************************
//+ Fehler_DatentypID - Dezimalzahl in Vorgangsnummer
//*****************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 1; -; 2
2; B; 1; 1; 3
3; C; 1; 2; 4,5,6
4.5; D; 2; 3; 12
5; E; 1; 3; 10
6; F; 5; 3; 7,8,9
7; G; 5; 6; 10
8; H; 2; 6; 10
9; I; 3; 6; 10
10; J; 4; 5,7,8,9; 11
11; K; 1; 10; 12
12; L; 1; 11; 13;
13; M; 4; 12; 14;
14; N; 1; 13; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Vorgangsnummer des 4. Vorgangs ist nicht ganzzahlig.
```

4.3.4 Fehler_DatentypNachfolger.in

Eingabedatei:

```
//*****************************************************
//+ Fehler_DatentypNachfolger - Dezimalzahl in Vorgangsnummer
//*****************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
```

```
1; A; 1; -; 2
2; B; 1; 1; 3
3; C; 1; 2; 4,5,6
4.5; D; 2; 3; 12
5; E; 1; 3; 10
6; F; 5; 3; 7,8,9
7; G; 5; 6; 10
8; H; 2; 6; 10
9; I; 3; 6; 10
10; J; 4; 5,7,8,9; 11
11; K; 1; 10; 12
12; L; 1; 11; 13;
13; M; 4; 12; 14;
14; N; 1; 13; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Vorgangsnummer eines Nachfolgers des 1. Vorgangs
ist keine ganzzahlige Zahl.
```

4.3.5 Fehler_DatentypVorgänger.in

Eingabedatei:

```
//******************************************************
//+ Fehler_DatentypVorgänger - Dezimalzahl in Vorgangsnummer
//******************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 1; -; 2
2; B; 1; 1; 3
3; C; 1; 2; 4,5,6
4.5; D; 2; 3; 12
5; E; 1; 3; 10
6; F; 5; 3; 7,8,9
7; G; 5; 6; 10
```

```
8; H; 2; 6; 10
9; I; 3; 6; 10
10; J; 4; 5,7,8,9; 11
11; K; 1; 10; 12
12; L; 1; 11; 13;
13; M; 4; 12; 14;
14; N; 1; 13; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Vorgangsnummer eines Vorgängers des Vorgangs
mit der Vorgangsnummer 3 ist keine ganzzahlige Zahl.
```

4.3.6 Fehler_GleicheID.in

Eingabedatei:

```
//************************************************************
//+ Fehler_GleicheID - Vorgangsnummer (12) mehrfach vorhanden
//************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 1; -; 2
2; B; 1; 1; 3
3; C; 1; 2; 4,5,6
4; D; 2; 3; 12
5; E; 1; 3; 10
6; F; 5; 3; 7,8,9
7; G; 5; 6; 10
8; H; 2; 6; -
9; I; 3; 6; 10
12; J; 4; 5,7,8,9; 11
11; K; 1; 10; 12
12; L; 1; 11; 13;
13; M; 4; 12; 14;
14; N; 1; 13; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Der Vorgang mit der Vorgangsnummer 12
existiert mehrmals.
```

4.3.7 Fehler_GleicherVorgängerNachfolger.in

Eingabedatei:

```
//*********************************************************************
//+ Fehler_GleicherVorgängerNachfolger - Gleiche Zahlen in Vorgänger
und Nachfolger
//*********************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 25; -; 4,5,8
2; B; 8; -; 3
3; C; 5; 2; 2
4; D; 9; 1; 6
5; E; 21; 1; 7
6; F; 9; 3,4; 7
7; G; 6; 5,6; 9
8; H; 15; 1; 10
9; I; 2; 7; 10
10; J; 1; 8,9; 11
11; K; 2; 10; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Der Vorgang mit der Vorgangsnummer 3 hat einen
gleichen Vorgänger und Nachfolger.
```

4.3.8 Fehler_IDAlsNachfolger.in

Eingabedatei:

```
//************************************************************
//+ Fehler_IDAlsNachfolger - Vorgangsnummer (4) als Nachfolger
//************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 3; -; 2,3,4
2; B; 8; 1; 6
3; C; 2; 1; 6
4; D; 10; 3; 4
5; E; 13; 4; 6
6; F; 4; 2,3,5; 7,8
7; G; 1; 6; 9
8; H; 2; 6; 10
9; I; 2; 7; 10
10; K; 1; 8,9; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Der Vorgang mit der Vorgangsnummer 4 ist sein
eigener Nachfolger.
```

4.3.9 Fehler_IDAlsVorgänger.in

Eingabedatei:

```
//************************************************************
//+ Fehler_IDAlsVorgänger - Vorgangsnummer (4) als Vorgänger
//************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 3; -; 2,3,4
2; B; 8; 1; 6
3; C; 2; 1; 6
```

```
4; D; 10; 4; 5
5; E; 13; 4; 6
6; F; 4; 2,3,5; 7,8
7; G; 1; 6; 9
8; H; 2; 6; 10
9; I; 2; 7; 10
10; K; 1; 8,9; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Der Vorgang mit der Vorgangsnummer 4 ist
sein eigener Vorgänger.
```

4.3.10 Fehler_IDInNachfolgerNichtExistent.in

Eingabedatei:

```
//*************************************************************
//+ Fehler_IDInNachfolgerNichtExistent - Vorgangsnummer (12)wird als
Nachfolger adressiert, ist aber nicht da
//*************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 1; -; 3;
2; B; 1; -; 4,5
3; C; 2; 1; 6,7
4; D; 4; 2; 7
5; E; 3; 2; 8,9
6; F; 1; 3; 11
7; G; 5; 3,4; 11
8; H; 8; 5; 10
9; I; 2; 5; 12
10; J; 3; 8,9; 11
11; K; 1; 6,7,10; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Der Vorgang mit der Vorgangsnummer 12 existiert nicht
```

4.3.11 Fehler_IDInCorgängerNichtExistent.in

Eingabedatei:

```
//*****************************************************************
//+ Fehler_IDInNachfolgerNichtExistent - Vorgangsnummer (12) wird als
Vorgänger adressiert, ist aber nicht da
//*****************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 1; -; 3;
2; B; 1; -; 4,5
3; C; 2; 1; 6,7
4; D; 4; 2; 7
5; E; 3; 2,12; 8,9
6; F; 1; 3; -
7; G; 5; 3,4; -
8; H; 8; 5; 10
9; I; 2; 5; 10
10; J; 3; 8,9; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Der Vorgang mit der Vorgangsnummer 12 existiert nicht
```

4.3.12 Fehler_KeinAnfangsvorgang.in

Eingabedatei:

```
//****************************************
```

```
//+ Fehler_keinAnfangsvorgang - kein Anfangsvorgang
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 2; 4; 2,3
2; B; 14; 1; 4,5,6
3; C; 1; 1; 12
4; D; 1; 2; 7
5; E; 2; 2; 7
6; F; 1; 2; 7
7; G; 1; 4,5,6; 8,9
8; H; 5; 7; 11
9; I; 2; 7; 10
10; J; 2; 9; 11;
11; K; 1; 8,10; 12
12; L; 7; 3,11; 13
13; M; 1; 12; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
NetzplanException: Der Netzplan enthält keinen Startvorgang.
```

4.3.13 Fehler_KeinEndvorgang.in

Eingabedatei:

```
//****************************************
//+ Fehler_KeinEndvorgang 13 - kein Endvorgang
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 2; -; 2,3
2; B; 14; 1; 4,5,6
3; C; 1; 1; 12
4; D; 1; 2; 7
5; E; 2; 2; 7
6; F; 1; 2; 7
```

```
7; G; 1; 4,5,6; 8,9
8; H; 5; 7; 11
9; I; 2; 7; 10
10; J; 2; 9; 11;
11; K; 1; 8,10,13; 12
12; L; 7; 3,11; 13
13; M; 1; 12; 11
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
NetzplanException: Der Netzplan enthält keinen Endvorgang.
```

4.3.14 Fehler_KeineVorgänge.in

Eingabedatei:

```
//****************************************
//+ Fehler_KeineVorgänge - keine Daten
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Die Datei enthält keine Vorgänge.
```

4.3.15 Fehler_MehrereTitel.in

Eingabedatei:

```
//*************************************************************
```

```
//+ Fehler_MehrereTitel - Mehrere Titelzeilen
//*************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 3; -; 2,3,4
2; B; 8; 1; 6
//+ Fehlerbeispiel 16 - Mehrere Titelzeilen
3; C; 2; 1; 6
4; D; 10; 3; 5
5; E; 13; 4; 6
6; F; 4; 2,3,5; 7,8
7; G; 1; 6; 9
8; H; 2; 6; 10
9; I; 2; 7; 10
10; K; 1; 8,9; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Es ist mehr als ein Titel für das
Testbeispiel angegeben
```

4.3.16 Fehler_MehrereVorgängerOhneKommata.in

Eingabedatei:

```
//*****************************************************************
//+ Fehler_MehrereVorgängerOhneKommata - Mehrere Zahlen in Vorgänger
ohne Kommata
//*****************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 1; -; 2
2; B; 1; 1; 3
3; C; 1; 2; 4,5,6
4; D; 2; 3; 12
5; E; 1; 3; 10
6; F; 5; 3; 7,8,9
```

```
7; G; 5; 6; 10
8; H; 2; 6; -
9; I; 3; 6 4; 10
12; J; 4; 5,7,8,9; 11
11; K; 1; 10; 12
12; L; 1; 11; 13;
13; M; 4; 12; 14;
14; N; 1; 13; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Vorgänger der Vorgangsnummer 9enthalten ein
Leerzeichen. Evlt. ein fehlendes Komma?
```

4.3.17 Fehler_NegativeDauer.in

Eingabedatei:

```
//************************************************************
//+ Fehler_NegativeDauer - Negative Zahl in Dauer
//************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 10; -; 2,3,4,5,6
2; B; 5; 1; 7
3; C; -4; 1; 7,11
4; D; 8; 1; 8
5; E; 7; 1, 12
6; F; 5; 1; 9
7; G; 5; 2,3; 10
8; H; 2; 4; 11
9; I; 5; 6; 12,13
10; J; 8; 7; 14
11; K; 10; 3,8; 14,15
12; L; 5; 5,9; 15
13; M; 5; 9; 15
```

```
14; N; 9; 10,11; 16
15; O; 20; 11,12,13; 16
16; P; 10; 14,15; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Die Dauer des Vorgangs mit der Vorgangsnummer 3
ist nicht größter gleich Null.
```

4.3.18 Fehler_NegativeID.in

Eingabedatei:

```
//************************************************************
//+ Fehler_NegativeID - Negative Zahl in Vorgangsnummer
//************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 10; -; 2,3,4,5,6
2; B; 5; 1; 7
-3; C; 4; 1; 7,11
4; D; 8; 1; 8
5; E; 7; 1, 12
6; F; 5; 1; 9
7; G; 5; 2,3; 10
8; H; 2; 4; 11
9; I; 5; 6; 12,13
10; J; 8; 7; 14
11; K; 10; 3,8; 14,15
12; L; 5; 5,9; 15
13; M; 5; 9; 15
14; N; 9; 10,11; 16
15; O; 20; 11,12,13; 16
16; P; 10; 14,15; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Die ID des 3 Vorgangs ist negativ.
```

4.3.19 Fehler_NetzplanNichtZusammenhängend.in

Dieses Testbeispiel enthält einen nicht Zusammenhängenden Netzplan. Eine genauere Beschreibung der Überprüfung ist in Abschnitt 4.2.1 zu finden.

Eingabedatei:

```
//****************************************
//+ Fehler_NetzplanNichtZusammenhängend - Netz nicht zusammenhängend
//****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 10; -; 2,3,4
2; B; 5; 1; 7
3; C; 4; 1; 7,11
4; D; 8; 1; 8
5; E; 7; -; 12
6; F; 5; -; 9
7; G; 5; 2,3; 10
8; H; 2; 4; 11
9; I; 5; 6; 12,13
10; J; 8; 7; 14
11; K; 10; 3,8; 14,15
12; L; 5; 5,9; -
13; M; 5; 9; -
14; N; 9; 10,11; 16
15; O; 20; 11,12,13; 16
16; P; 10; 14,15; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
NetzplanException: Der Netzplan ist nicht zusammenhängend.
```

4.3.20 Fehler_VorgängerNachfolgerPassenNicht.in

Eingabedatei:

```
//*****************************************************************
//+ Fehler_VorgängerNachfolgerPassenNicht - Vorgänger und Nachfolger
passen nicht zusammen (Vorgänger fehlt)
//*****************************************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 21; -; 2,3
2; B; 5; 1; 4
3; C; 7; 1; 6
4; D; 3; 2; 5,6
5; E; 5; 4; 7
6; F; 8; 3; 7
7; G; 2; 5,6; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Der Vorgang 6 ist Nachfolger von 4. 4 ist aber
nicht Vorgänger von Vorgang 6.
```

4.3.21 Fehler_ZuVieleSemikolons.in

Eingabedatei:

```
//*****************************************
//+ Fehler_ZuVieleSemikolons - Zuviele Semikolons
//*****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 3; -; 2
2; B; 9; 1; 3
3; C; 12; 2; 4
4; D; 18; 3; 5
5; E; 8; 4; 6
```

```
6; F; 8; 5; 7,8,9
7; G; 30; 6; 10
8; H; 3; 6; 10; 6
9; I; 2; 6; 12
10; J; 10; 7,8; 11
11; K; 5; 10; 12
12; L; 4; 9,11; 12
13; M; 1; 12; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Semikolonfehler im 8 Vorgang.
```

4.3.22 Fehler_ZuWenigeSemikolons.in

Eingabedatei:

```
//*****************************************
//+ Fehler_ZuWenigeSemikolons - Zu wenige Semikolons
//*****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 3; -; 2
2; B; 9; 1; 3
3; C; 12; 2; 4
4; D; 18; 3; 5
5; E; 8; 4, 6
6; F; 8; 5; 7,8,9
7; G; 30; 6; 10
8; H; 3; 6; 10, 6
9; I; 2; 6; 12
10; J; 10; 7,8,11
11; K; 5; 10; 12
12; L; 4; 9,11; 12
13; M; 1; 12; -
```

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
EingabeException: Semikolonfehler im 5 Vorgang.
```

4.3.23 Fehler_Zyklus.in

Dieses Testbeispiel enthält einen Zyklus, der durch die Vorgänge 6,9,12,15 geht.

Eingabedatei:

```
//*****************************************
//+ Fehler_Zyklus - Zyklus
//*****************************************
//Vorgangsnummer; Vorgangsbezeichnung; Dauer; Vorgänger; Nachfolger
1; A; 10; -; 2,3,4,5,6
2; B; 5; 1; 7
3; C; 4; 1; 7,11
4; D; 8; 1; 8
5; E; 7; 1; 12
6; F; 5; 1,15; 9
7; G; 5; 2,3; 10
8; H; 2; 4; 11
9; I; 5; 6; 12,13
10; J; 8; 7; 14
11; K; 10; 3,8; 14,15
12; L; 5; 5,9; 15
13; M; 5; 9; 15
14; N; 9; 10,11; 16
15; O; 20; 11,12,13; 6,16
16; P; 10; 14,15; -
```

Bei der Methode *pruefeZyklus(v: Vorgang, list: ArrayList¡Integer)* wird für jeden Endvorgang aufgerufen. Da es in diesem Beispiel nur einen Endvorgang gibt, wird die Methode allein mit diesem aufgerufen. Von dem Endvorgang aus fügt er sich selbst der Liste hinzu und ruft nun für jeden Vorgänger die Methode auf, und übergibt den Nachfolger und die Liste. Dieser Überprüft nun, ob er in der Liste steht. Wenn nicht fügt er alle seine Nachfolger die noch nicht in der Liste stehen der Liste hinzu. Die Liste beinhaltet also alle Vorgänge, die auf dem aktuellen Pfad bereits erreichbar sind. Anschließend wird wieder

für jeden Vorgänger die Methode aufgerufen und der Vorgänger und die Liste übergeben. Für den Pfad (Rückwärts) 16,15,12,9 sieht die Liste wie folgt aus:

```
16 -> []
15 -> [6, 16]
12 -> [6, 16, 15]
5 -> [6, 16, 15, 12]
9 -> [6, 16, 15, 12, 13]
```

Als nächstes wird die Methode mit dem Vorgang 6 aufgerufen. Dieser überprüft, ob er bereits in der Liste steht, was der Fall ist. Aus diesem Grund wird an der Stelle ein Fehler ausgegeben und das Programm beendet.

Ausgabedatei:

```
************************************
Es ist ein Fehler aufgetreten.
************************************
NetzplanException: Der Netzplan enthält einen Zyklus.
Betroffen ist der Vorgang mit der Vorgangsnummer 6
```

Kapitel 5

Zusammenfassung und Ausblick

Das Programm wurde dazu entwickelt, einen Netzplan einzulesen und dessen kritischen Pfade und die Gesamtdauer eines Projektes zu bestimmen, um mögliche Verbesserungen für eine schnellere Abarbeitung aufzuzeigen. Der Netzplan ist dabei durch die Angabe von Vorgängen definiert, die wiederum aus einer Vorgangsnummer, einer Vorgangsbeschreibung, einer Vorgangsdauer, Vorgängern und Nachfolgern bestehen. Start- und Endvorgänge besitzen keine Vorgänger bzw. Nachfolger. Desweiteren hat ein Vorgang die Werte FAZ, FEZ, SAZ, SEZ, GP, FP und kritisch, die bestimmt werden müssen um die kritischen Pfade und die Gesamtdauer ermitteln zu können. Nun werden zuerst mittels eines Backtracking-Verfahrens von allen Endvorgängen ausgehend die Werte FAZ und FEZ bestimmt. Anschließend werden ebenfalls mit einem Backtracking-Verfahren von allen Startvorgängen ausgehend die restlichen Werte bestimmt. Als dritter Schritt werden die kritischen Pfade und die Gesamtdauer des Netzplans ermittelt. Abschließend erfolgt eine Ausgabe der Start- und Endvorgänge, der Gesamtdauer und der kritischen Pfade in eine Ausgabedatei.

Da die Erstellung größerer Netzpläne ohne graphische Veranschaulichung sehr umständlich ist, könnte das Programm mit einer GUI erweitert werden. In dieser müsste es die Möglichkeit geben einen Netzplan zu entwerfen und anschließlich berechnen zu können. Die GUI müsste den erstellen Netzplan in eine Eingabedatei die den in Abschnitt 1.3 beschriebenen Formalien entspricht umwandeln und anschließend das Programm mit dieser ausführen.

Eine weitere Verbesserung könnte bei der Ausgabe des Programms vorgenommen werden. Diese könnte zusätzlich zu der bisherigen Ausgabedatei eine graphische Darstellung der Ausgabe liefern, in der alle berechneten Werte stehen und die kritischen Pfade kennt-

lich gemacht sind.

Desweiteren ließe sich das Programm so erweitern, dass es versucht den erstellten Netzplan zu verbessern. Mit Blick auf die Realität, gibt es Projekte, in denen es erstrebenswert ist, den Terminplan zu finden, dessen Gesamtdauer am geringsten ist und in denen Puffer eine unbedeutende Rolle spielen. Dem entgegengesetzt kann es Projekte geben, in denen es wünschenswert ist, Puffer in der Terminplanung zu haben und die Gesamtdauer des Projektes eine geringere Rolle spielt. Zusätzlich sind nicht in jedem Projekt alle Vorgänge von anderen Causalabhängig, weshalb sie im Netzplan auch zu einem anderen Zeitpunkt bearbeitet werden können. Durch eine entsprechende Umstrukturierung des Netzplanes, könnte so dieser entweder besonders viele oder besonders wenige kritische Pfade enthalten. Je weniger kritische Pfade es gibt, desto mehr Puffer gäbe es in der Terminplanung, je mehr desto schneller könnte das Projekt fertiggestellt werden.

Anhang A

Abweichungen und Ergänzungen zum Vorentwurf

Das implementierte Programm weicht bei der Bestimmung der kritischen Pfade vom Vorentwurf ab. In diesem wird in der Methode *zweitePhase(v: Vorgang)* der aktuelle Vorgang der ArrayListe 'kritischerPfad' hinzugefügt, sollte der Vorgang kritisch sein. Dies führt jedoch dazu, dass die Liste eine Aneinanderreihung der kritischen Vorgänge enthält. Für den Fall, dass es nur einen kritischen Pfad gibt, funktioniert dieses Verfahren. Da es aber auch vorkommen kann, dass es mehr als einen kritischen Pfad gibt, führt diese Methode zu einem Fehler. Aus diesem Grund wird ein kritischer Vorgang in dieser Methode nicht mehr einer Liste hinzugefügt sondern nur der boolean kritisch auf true gesetzt. Anschließend wird eine eigene Methode aufgerufen, die rekursiv durch den Netzplan geht und alle kritischen Pfade ermittelt. Diese werden dann in der Liste 'kritischePfade' gespeichert. Die Liste 'kritischerPfad' dient nur noch als Hilfsvariable für die Methode. Auf diese Weise ist es möglich auch mehr als einen kritischen Pfad zu erkennen und zu speichern.

Anhang B

Benutzeranleitung

B.1 Systemvoraussetzungen

B.1.1 Installierte Programme

Zum Ausführen des Programms wird eine Java Laufzeitumgebung der Version JavaSE-1.8 oder höher vorausgesetzt. Das mitgelieferte Skript runall.bat setzt eine installierte Windows-Version mit installierter Bash voraus. Da das Programm als .zip-Archiv ausgeliefert wird, muss ein Programm zum Entpacken zur Verfügung stehen.

B.1.2 Hardware

Für das Programm wird keine besondere Hardware benötigt.

B.2 Verzeichnisstruktur

Entpacken Sie die Datei Netzplanersteller.zip. Danach finden Sie die folgende Verzeichnisstruktur vor:

```
NetzplanErsteller
├─bin
├─src
└─dist
```

```
├── lib
├── doc
├── Tests
├── runall.bat
└── Dokumentation.pdf
```

Der Ordner *src* enthält den Quellcode. Die Datei Netzplanerstellung.jar befindet sich in dem Ordner *dist*. Das Verzeichnis *lib* beinhaltet die verwendeten Bibliotheken. Die Entwicklerdokumentation als Javadoc befindet sich im Ordner *doc*. Der Ordner *Tests* beinhaltet alle mitgelieferten Testfälle mitsamt Ausgabedateien. Darüber hinaus enthält der Ordner NetzplanErsteller das Batch-Skript *runall.bat*. Dieses Skript führt alle Testfälle im Ordner Tests nacheinander aus. Die Datei *Dokumentation.pdf* beinhaltet die Dokumentation. Im Ordner *bin* finden sich die Kompilierten .class Dateien. Im Ordner *Tests* befinden sich die folgenden Ein- und Ausgabedateien:

```
NetzplanErsteller
├── ...
├── Tests
│   ├── IHKTestfall1.in
│   ├── IHKTestfall1.out
│   ├── IHKTestfall2.in
│   ├── IHKTestfall2.out
│   ├── IHKTestfall3.in
│   ├── IHKTestfall3.out
│   ├── IHKTestfall4.in
│   ├── IHKTestfall4.out
│   ├── IHKTestfall5.in
│   └── IHKTestfall5.out
└── ...
```

Diese Testfälle dienen der Demonstration der Funktionalität und der Überprüfung von Fehler- und Sonderfällen.

B.3 Programmaufruf über die Kommandozeile

Das Programm muss über die Kommandozeile aufgerufen werden. Führen Sie dazu den Befehl
java -jar raetselLoeser.jar [-h] [-l] [-o AUSGABEDATEI] [-t] EINGABEDATEI
aus um das Programm zu starten. Durch Angabe von -h können Sie weitere Hilfen zu den verfügbaren Parametern erhalten. Die Option -l zeigt Logging-Informationen in der

Konsole an, sodass die Programmausführung besser nachvollzogen werden kann. Wenn Sie den Pfad oder den Namen der Ausgabedatei selbst definieren möchten, so kann dazu die Option -o verwendet werden. Wenn diese Option nicht genutzt wird, benutzt das Programm den Pfad der Eingabedatei und ersetzt im Dateinamen das .in durch .out. Die Option -t gibt die Möglichkeit nur den Dateinamen, ohne Dateipfad anzugeben. In diesem Fall geht das Programm davon aus, dass sich die Eingabedatei im Ordner Tests befindet. Abschließend muss eine Eingabedatei angegeben werden. Die Angabe der Eingabedatei ist nicht optional und wird zur Programmausführung zwingend benötigt.

B.4 Ausführung der Testbeispiele

Die Datei *runall.bat* führt alle Testfälle im Ordner *Tests* aus. Dabei werden alle Dateien als Eingabedateien betrachtet die auf .in enden. Die zugehörigen Ausgabedateien enden dann auf .out und befinden sich im selben Ordner. Bereits existierende Ausgabedateien werden überschrieben. Das Skript lässt sich mit einem Doppelklick starten.

Anhang C

Entwicklungsumgebung

Programmiersprache	:	Java
Compiler	:	Javac 1.8.0_171-b11
Rechner	:	Intel(R) Core(TM) i5-2500 CPU @3.3GHz
Betriebssystem	:	Windows 7 Professional, 64bit