
Dokumentation der Praktischen Arbeit
zur Prüfung zum
Mathematisch-technischen Softwareentwickler

Thema: Automatisierte Erstellung von Netzplänen

16. Mai 2018

Christian Peters

Prüfungs-Nummer: 20613

Programmiersprache: Java

Ausbildungsort: Institut für Kernphysik
Forschungszentrum Jülich

Inhaltsverzeichnis

1	Einleitung	1
1.1	Was ist ein Netzplan?	1
1.2	Kritische Pfade	3
1.3	Ziel dieser Arbeit	3
2	Verfahrensbeschreibung	5
2.1	Datenstrukturen	5
2.1.1	Logische Repräsentation eines Vorgangs	5
2.1.2	Logische Repräsentation eines Netzplans	6
2.2	Einlesen der Vorgänge	7
2.2.1	Format der Eingabedatei	7
2.2.2	Algorithmus zum Einlesen	8
2.3	Konstruktion des Netzplans	8
2.3.1	Initialisierung	8
2.3.2	Vorwärtsrechnung	10
2.3.3	Rückwärtsrechnung	11
2.3.4	Berechnung der Zeitreserven	11
2.4	Auffinden der kritischen Pfade	11

2.5	Ausgabe der Ergebnisse	11
3	Entwicklerdokumentation	13
4	Benutzeranleitung	15
5	Testdokumentation	17
6	Ausblick	19
A	Abweichungen und Ergänzungen zum Vorentwurf	21

Kapitel 1

Einleitung

Die Kunst einer erfolgreichen zeitlichen Planung von Projekten ist es, die einzelnen Schritte bis zum Ziel optimal aufeinander abzustimmen. Kommt es in einer Kette von Aufgaben auch nur an einer Stelle zu Verzögerungen, kann dies das gesamte Projekt gefährden. Damit Projektmanager auch in schwierigen Situationen stets den Überblick behalten können, gibt es einige Werkzeuge, die bei der Planung großer Projekte von besonderer Bedeutung sind. Eines dieser Werkzeuge soll nun im Rahmen dieser Arbeit näher untersucht werden: Der Netzplan.

1.1 Was ist ein Netzplan?

Ein Netzplan dient der Darstellung aller Abhängigkeiten zwischen den einzelnen Vorgängen eines Projektes. Ein Vorgang innerhalb eines Projektes lässt sich hierbei als einzelner Knoten in einer verzweigten Kette aus weiteren Vorgängen auffassen, die alle zusammen wiederum das gesamte Projekt repräsentieren. Die Abhängigkeiten unter den Vorgängen werden hierbei durch Pfeile ausgedrückt: Ein Vorgänger besitzt einen Pfeil hin zu seinem Nachfolger. Jeder Vorgang verfügt hierbei über bestimmte Eigenschaften, die ihn eindeutig beschreiben:

- **Vorgangsnummer:** Eine eindeutige Identifikationsnummer des Vorgangs
- **Vorgangsbezeichnung:** Eine lesbare Bezeichnung des Vorgangs
- **Dauer:** Wie lange dauert die Ausführung des Vorgangs?

- **Vorgänger:** Welche anderen Vorgänge müssen zuerst ausgeführt werden, bevor der aktuelle Vorgang ausgeführt werden kann?
- **Nachfolger:** Welche anderen Vorgänge warten auf eine Abarbeitung des aktuellen Vorgangs?
- **Frühester Anfangszeitpunkt (FAZ):** Wann kann frühestens mit der Bearbeitung des Vorgangs begonnen werden?
- **Spätester Anfangszeitpunkt (SAZ):** Wann muss spätestens mit der Bearbeitung begonnen werden, damit der Zeitplan nicht gefährdet wird?
- **Frühester Endzeitpunkt (FEZ):** Wann kann der Vorgang frühestens abgeschlossen werden?
- **Spätester Endzeitpunkt (SEZ):** Wann muss der Vorgang spätestens abgeschlossen sein, damit der Zeitplan nicht gefährdet wird?
- **Gesamtpuffer (GP):** Wie viel Spielraum liegt zwischen dem frühesten und dem spätesten Anfangszeitpunkt?
- **Freier Puffer (FP):** Wie viel Spielraum existiert, wenn kein nachfolgender Vorgang durch den aktuellen Vorgang verzögert werden soll?

Wichtig bei der Angabe der Vorgänger und Nachfolger ist, dass ein Netzplan keine zyklischen Abhängigkeiten enthalten darf (sonst könnte sich die Dauer eines Projektes bis ins Unendliche hinziehen). Zudem muss es stets mindestens einen Vorgang ohne Vorgänger (einen *Startvorgang*) und mindestens einen Vorgang ohne Nachfolger (einen *Endvorgang*) geben.

Da zu Beginn der Planungsphase im Normalfall noch nicht alle aufgelisteten Attribute bekannt sind (i.d.R. nur Vorgangsnummer, Vorgangsbezeichnung, Dauer, sowie Vorgänger und Nachfolger), ist es die Aufgabe des Projektmanagers, die restlichen Eigenschaften so zu bestimmen, dass der Zeitplan eingehalten wird. An dieser Stelle kommt die Rolle des Netzplans zum tragen: Durch die Verkettung der einzelnen Vorgänge mit ihren Vorgängern und Nachfolgern können die fehlenden Attribute (FAZ, SAZ, FEZ, SEZ, GP, FP) systematisch berechnet werden. Hieraus ergibt sich auch unmittelbar die Gesamtdauer des Projektes: Sie entspricht entweder dem FEZ der Endvorgänge oder ist undefiniert, falls dieser nicht eindeutig ist.

1.2 Kritische Pfade

In jedem Projekt gibt es immer auch Vorgänge, deren Abarbeitung an besonders enge Fristen geknüpft ist und daher keinerlei Spielraum erlaubt. Solche Vorgänge, bei denen entsprechend der obigen Notation $GP = 0$ und $FP = 0$ gilt, werden im Folgenden als *kritische Vorgänge* bezeichnet. In jedem Projekt gibt es mindestens eine Kette von Vorgängen beginnend bei einem Startvorgang hin zu einem Endvorgang, die ausschließlich aus kritischen Vorgängen besteht. Eine derartige Konstellation wird auch als *kritischer Pfad* bezeichnet, da eine reibungslose Abarbeitung dieser Vorgänge Grundvoraussetzung für die Einhaltung des Zeitplans ist. Der Projektmanager sollte also schon zu Beginn alle kritischen Pfade identifizieren und besonders auf die termingerechte Abarbeitung der zugehörigen Vorgänge achten. Auch bei dieser Aufgabe hilft ihm der Netzplan, der sich systematisch nach kritischen Pfaden durchsuchen lässt.

1.3 Ziel dieser Arbeit

Das Ziel dieser Arbeit ist es nun, die Erstellung von Netzplänen sowie die Suche nach kritischen Pfaden zu automatisieren. Das zu diesem Zweck zu entwickelnde Programm soll in der Lage sein, die einzelnen Vorgänge eines Projektes aus einer Eingabedatei einzulesen und hieraus einen korrekten Netzplan zu konstruieren. Die fehlenden Eigenschaften der Vorgänge (FAZ, SAZ, FEZ, SEZ, GP, FP) sollen dazu systematisch errechnet werden. Anschließend soll der Netzplan auf kritische Pfade durchsucht werden, die Ergebnisse der Suche, sowie die vollständige Beschreibung aller Vorgänge und ihrer Fristen inklusive der Gesamtdauer des Projektes, sollen abschließend in einer Ausgabedatei gespeichert werden.

Kapitel 2

Verfahrensbeschreibung

2.1 Datenstrukturen

Die Grundlage eines jeden Programms ist durch die Art der verwendeten Datenstrukturen gegeben. Damit im weiteren Verlauf Netzpläne effizient erzeugt werden können, ist eine solide logische Repräsentation unerlässlich. Die grundlegende Darstellung der wichtigsten Elemente dieses Programms soll nun im Folgenden näher beschrieben werden.

2.1.1 Logische Repräsentation eines Vorgangs

Ein einzelner Vorgang kann als Objekt dargestellt werden, welches die in 1.1 beschriebenen Attribute *Vorgangsnummer*, *Vorgangsbezeichnung*, *Dauer*, *Vorgänger*, *Nachfolger*, *FAZ*, *SAZ*, *FEZ*, *SEZ*, *GP*, *FP* enthält und einen Zugriff auf diese ermöglicht. Hierbei werden jedem Attribut die folgenden Definitionsbereiche zugeordnet, welche später vom Anwender eingehalten werden müssen:

- **Vorgangsnummer:** Element der ganzen Zahlen \mathbb{Z} , welches im gesamten Projekt *eindeutig* sein muss.
- **Vorgangsbezeichnung:** Nicht leere Zeichenkette, welche das Semikolon nicht enthalten darf¹.
- **Dauer:** Element der natürlichen Zahlen \mathbb{N} . Dies bedeutet, dass keine negativen Dauern und auch keine nicht existenten Dauern zugelassen sind.

¹Dies hängt mit dem Eingabeformat zusammen, was an späterer Stelle erläutert wird.

- **Vorgänger:** Liste gültiger Vorgangsnummern.
- **Nachfolger:** Liste gültiger Vorgangsnummern.
- **FAZ, SAZ, FEZ, SEZ, GP und FP:** Elemente der natürlichen Zahlen \mathbb{N} *inklusive* der 0.

Mehrere Vorgänge, welche zusammen ein Projekt bilden, können somit in einer Liste aus einzelnen Vorgangsobjekten abgelegt werden.

2.1.2 Logische Repräsentation eines Netzplans

Ein Netzplan kann formal als gerichteter azyklischer Graph beschrieben werden, wobei die Knotenmenge durch die Menge aller Vorgänge des Projektes gegeben ist und die Kantenmenge durch die Vorgänger-Nachfolger-Relation unter den Vorgängen beschrieben werden kann. Ein Hilfsmittel, welches zusätzlich zu der Liste aller Vorgänge des Projekts verwendet wird, um den Netzplan als Graph darzustellen, ist die sogenannte *Adjazenzmatrix*. Diese Matrix A besitzt für jeden Knoten des Graphen sowohl eine Zeile als auch eine Spalte und hat daher bei n Knoten die Dimension $n \times n$. Existiert eine direkte Verbindung von Knoten i nach Knoten j (dies ist anschaulich der Fall, wenn Vorgang j Nachfolger von Vorgang i ist), so ist der jeweilige Eintrag a_{ij} der Adjazenzmatrix 1, ansonsten 0. Anhand der Adjazenzmatrix lässt sich auch leicht erkennen, ob ein Knoten Start- oder Endknoten ist: Da ein Startknoten i keinen Vorgänger hat, gilt für die entsprechenden Einträge der i 'ten Spalte: $a_{ki} = 0 \forall 1 \leq k \leq n$. Für jeden Endknoten j gilt analog $a_{jk} = 0 \forall 1 \leq k \leq n$.

Es sei an dieser Stelle schon angemerkt, dass Vorgänge in ihrer Nummerierung nicht zwingend die Zahlen 1 bis n (oder später im Rechner 0 bis $n - 1$) belegen müssen. Daher wird zusätzlich eine Abbildung $g : \text{Vorgangsnummern} \rightarrow \mathbb{N}$ eingeführt, welche die Vorgangsnummern lückenlos auf die ersten n natürlichen Zahlen abbildet. Dementsprechend bezeichnet g^{-1} die zugehörige Umkehrabbildung, welche eine Zahl von 1 bis n auf die entsprechende ursprüngliche Vorgangsnummer abbildet.

2.2 Einlesen der Vorgänge

2.2.1 Format der Eingabedatei

Das Format, in dem die Daten dem Programm zugeführt werden, soll hier anhand des folgenden Beispiels erläutert werden:

```
//*****  
//+ 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 Nachbesserung der POI Kioske; 1; 6; -
```

Abbildung 2.1: Beispiel einer Eingabedatei

Eine Datei besteht aus einer ungeordneten Menge an Kommentar- und Datenzeilen. Leerzeilen sind *nicht* zugelassen. Kommentarzeilen lassen den Anwender Anmerkungen hinzufügen, welche vom Programm nicht beachtet werden. Sie beginnen stets mit einem `//`. Eine besondere Art der Kommentarzeile ist die *Überschriftzeile*, da sie im Gegensatz zu den normalen Kommentarzeilen vom Programm besonders berücksichtigt wird. Sie beginnt immer mit einem `//+` und kann an beliebiger Stelle innerhalb der Datei auftauchen. Die Zeichenkette, welche auf das einleitende `//+` folgt, beschreibt das jeweilige Projekt und darf nicht leer sein. Pro Datei muss es *genau eine* Überschriftzeile geben. Im gegebenen Beispiel sind die Zeilen 1, 3 und 4 klassische Kommentarzeilen und werden vom Programm nicht beachtet. Zeile 2 ist die Überschriftzeile.

Datenzeilen liefern dem Programm die notwendigen Informationen zu den einzelnen Vorgängen eines Projekts. Jede Datenzeile beschreibt genau einen Vorgang und besteht aus fünf Elementen, welche jeweils durch ein Semikolon getrennt werden: *Vorgangsnummer*, *Vorgangsbezeichnung*, *Dauer*, *Vorgänger* und *Nachfolger*². Die Eingaben zu Vorgangsnummer, Vorgangsbezeichnung und Dauer müssen hierbei mit den Definitionsbereichen der in 2.1.1 beschriebenen Attribute eines Vorgangs übereinstimmen. Die Elemente Vorgänger und Nachfolger werden jeweils durch gültige Vorgangsnummern angegeben,

²Dies ist auch der Grund, weshalb im Definitionsbereich der Vorgangsbezeichnung keine Semikola zugelassen sind.

welche durch Kommata getrennt aufgezählt werden. Doppelte Vorgänger oder Nachfolger sind nicht erlaubt. Hat ein Vorgang keinen Vorgänger bzw. keinen Nachfolger, so steht an dieser Stelle das Zeichen –. Es handelt sich in diesem Fall um einen Start- oder Endvorgang. Jedes Projekt muss mindestens einen Start- und einen Endvorgang enthalten.

2.2.2 Algorithmus zum Einlesen

Der Algorithmus, der zum Einlesen der Eingabedatei implementiert wurde, iteriert im Wesentlichen über alle Zeilen und prüft, um was für eine Art der Zeile es sich handelt. Je nach Art der Zeile werden die Eingaben validiert, stimmt etwas nicht, so wird ein Fehler geworfen. Während der Iteration wird sukzessive eine Liste mit gelesenen Vorgängen befüllt, die am Ende zurückgegeben wird. Diese Liste wird im folgenden Struktogramm (siehe Abbildung 2.2), welches einen Überblick über die wesentlichen Gesichtspunkte des Algorithmus gibt, als *Resultat* bezeichnet.

2.3 Konstruktion des Netzplans

Um aus der eingelesenen Liste aus Vorgangsobjekten nun den Netzplan konstruieren zu können, sind einige Schritte notwendig, die an dieser Stelle näher beschrieben werden.

2.3.1 Initialisierung

Während der Initialisierungsphase wird zunächst geprüft, ob alle Grundvoraussetzungen erfüllt sind, damit es sich um einen gültigen Netzplan handelt. Hierzu wird zunächst die Adjazenzmatrix erzeugt und währenddessen die Konsistenz der Vorgänger - Nachfolger - Relation sichergestellt.

Erzeugung der Adjazenzmatrix

Um die Adjazenzmatrix zu erzeugen, wird jede Vorgangsnummer einer Spalte in der Matrix zugeordnet. Hierzu wird die Abbildung g verwendet, die bereits bei der Beschreibung des Netzplans erläutert wurde. Man erhält g , indem man ein assoziatives Feld anlegt, wobei jeder Vorgangsnummer die zugehörige Position in der Liste aus Vorgängen zugeordnet wird. Für jeden Vorgang wird nun in der zugehörigen Zeile der Adjazenzmatrix in der Spalte jedem seiner Nachfolger eine 1 in der Matrix eingetragen. Hierbei wird auch

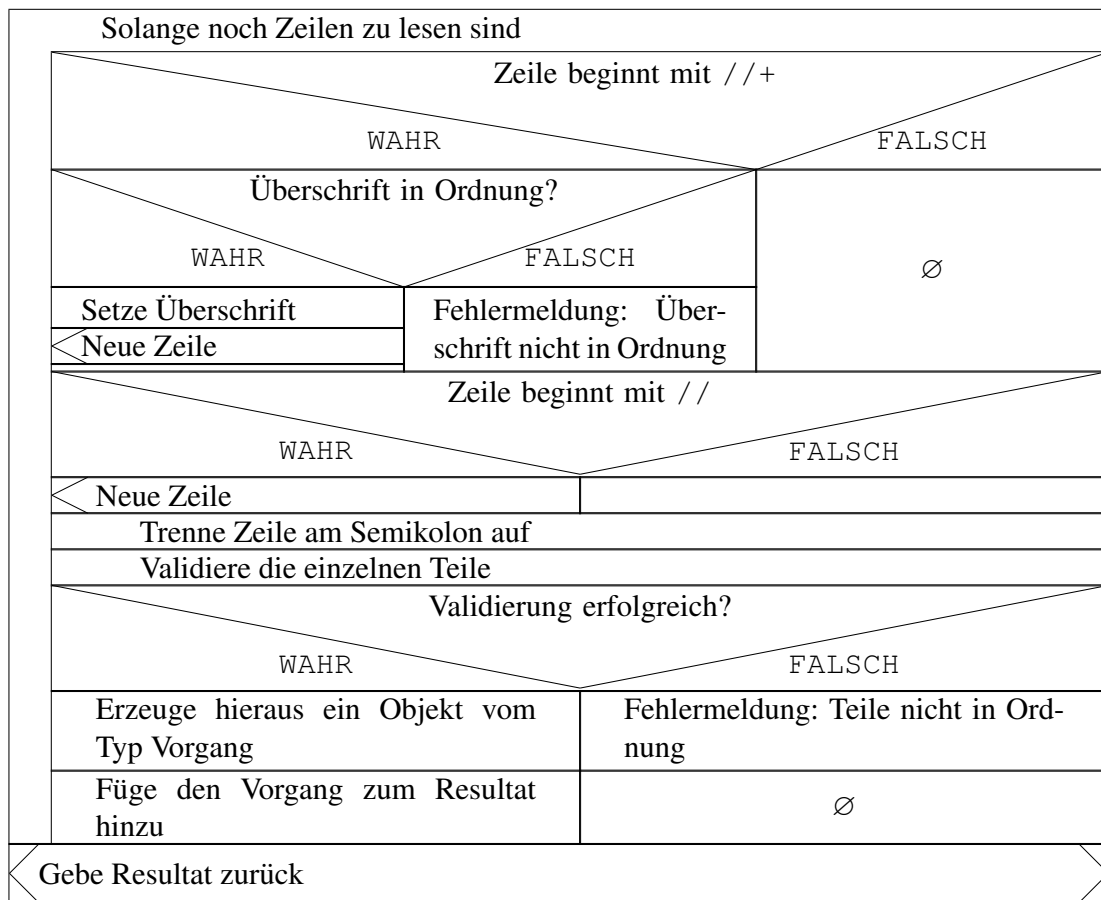


Abbildung 2.2: Algorithmus zum Einlesen der Eingabedatei

gleich abgeprüft, ob die Beziehung eines Vorgangs zu seinen Vorgängern und Nachfolgern konsistent ist: Ist z.B. ein Vorgang *a* als Vorgänger von Vorgang *b* eingetragen, hat diesen aber nicht als Nachfolger, so ist die Beziehung inkonsistent und der Netzplan kann nicht erzeugt werden. Das Programm wird in diesem Fall mit einem Fehler beendet.

Hängt der Graph zusammen?

Nachdem die Adjazenzmatrix erzeugt wurde und die Konsistenz der Beziehungen unter den Vorgängen sichergestellt ist, muss im nächsten Schritt geprüft werden, ob der Graph auch zusammenhängt, da es sich nur in diesem Fall um einen gültigen Netzplan handelt. Um dies zu bewerkstelligen, wird der bis dahin gerichtete Graph vorübergehend als ungerichtet aufgefasst. Beginnend bei einem beliebigen Knoten wird der Graph im nächsten Schritt traversiert. Bleiben nach dem Durchlaufen noch Knoten übrig, welche nicht be-

sucht wurden, war der Graph nicht zusammenhängend.

Um aus dem gerichteten Graph einen ungerichteten Graph zu erzeugen, kann einfach eine symmetrische Version der Adjazenzmatrix erstellt werden. Hierzu wird zu jedem Element $a_{ij} = 1$ das an der Diagonalen gespiegelte Element a_{ji} ebenfalls auf 1 gesetzt. Die Traversierung, welche im nächsten Schritt erfolgt, wird mittels einer Breitensuche durchgeführt.

Überprüfen auf Zyklen

Da ein gültiger Netzplan keine Zyklen enthalten darf, muss nun geprüft werden, ob der vorliegende Graph auch azyklisch ist. Hierzu wird ausgehend von jedem Startknoten eine Expansion des Graphen gebildet (d.h. es werden alle möglichen Pfade durch den Graphen erzeugt). Sobald ein Pfad im nächsten Schritt der Expansion ein Element aufnehmen würde, welches bereits in diesem Pfad enthalten ist, wurde ein Zyklus entdeckt und das Programm kann mit einem Fehler beendet werden. Auch hier wird die zu diesem Zweck benötigte Traversierung mit einer Breitensuche durchgeführt.

2.3.2 Vorwärtsrechnung

Nachdem in den letzten Abschnitten sichergestellt wurde, dass aus der Liste an Vorgängen ein gültiger Netzplan erzeugt werden kann, müssen nun die noch fehlenden Einträge FAZ, SAZ, FEZ, SEZ, GP und FP für jeden Vorgang berechnet werden. In der Phase der Vorwärtsrechnung werden für jeden Vorgang die Werte FAZ und FEZ gesetzt, weiterhin kann nach dieser Phase auch die Dauer des Projektes abgelesen werden. Diese entspricht nämlich wie in Abschnitt 1.1 beschrieben dem FEZ der Endvorgänge, insofern dieser eindeutig ist.

Um die Vorwärtsrechnung durchzuführen, wird eine Liste mit abzuarbeitenden Knoten erzeugt, welche zunächst nur die Startknoten enthält. Solange diese Liste noch Elemente hat, wird jedes Element beginnend am Anfang der Liste einzeln bearbeitet. Hierzu wird zuerst der FEZ des Elements auf FAZ+Dauer gesetzt. Man beachte hierbei, dass Startknoten von Beginn an einen FAZ von 0 haben. Anschließend werden alle Nachfolger des Elements betrachtet. Der FAZ eines jeden Nachfolgers wird nun auf den FEZ des aktuellen Elements gesetzt, wenn sich der FAZ des Nachfolgers hierdurch vergrößert. Jeder Nachfolger wird dann hinten an die Liste der zu bearbeitenden Knoten angefügt und eine neue Iteration beginnt.

2.3.3 Rückwärtsrechnung

In der Phase der Rückwärtsrechnung werden nun für jeden Knoten die Werte SAZ und SEZ gesetzt. Hierbei wird zunächst auch wieder eine Liste mit abzuarbeitenden Knoten erzeugt, welche diese Mal nur die Endknoten enthält. Bei jedem Endknoten wird nun $SEZ = FEZ$ gesetzt, da Endknoten keinen Puffer haben. Die Liste wird nun wieder beginnend mit dem ersten Element abgearbeitet, bis sie leer ist. Hierbei wird zunächst der SAZ des aktuellen Elementes auf SEZ-Dauer gesetzt. Im Anschluss werden alle Vorgänger des Elementes betrachtet. Wurde der SEZ des Vorgängers noch nicht gesetzt oder würde er sich verringern, so wird dieser auf den SAZ des aktuellen Elements gesetzt. Anschließend wird wieder jeder Vorgänger hinten an die Liste angehängt und die Iteration beginnt von neuem.

2.3.4 Berechnung der Zeitreserven

Die Zeitreserven eines jeden Vorganges werden durch die Attribute GP und FP ausgedrückt. Um diese zu berechnen, werden alle Vorgänge durchlaufen. Hierbei wird für jeden Vorgang $GP = SAZ - FAZ = SEZ - FEZ$ gesetzt. Für den Wert FP gilt folgender Zusammenhang: $FP = \min_{i \in \{Nachfolger\}} FAZ_i - FEZ$. Dieser wird ebenfalls für jeden Vorgang ermittelt und gesetzt.

2.4 Auffinden der kritischen Pfade

2.5 Ausgabe der Ergebnisse

Kapitel 3

Entwicklerdokumentation

Programmiersprache	:	Java
Compiler	:	javac 1.8.0_161
Rechner	:	Intel Core i7-4790 CPU @ 3.60GHz, 16GB RAM
Betriebssystem	:	Ubuntu 16.04 LTS

Kapitel 4

Benutzeranleitung

Kapitel 5

Testdokumentation

Kapitel 6

Ausblick

Anhang A

Abweichungen und Ergänzungen zum Vorentwurf

