AUTOMATISIERTE ERSTELLUNG VON NETZPLÄNEN

29. August 2018 | Christian Peters | Institut für Kernphysik

ARBEITEN IM INSTITUT

- CLAS12 Project
  - Teilchendetektor der Thomas Jefferson National Facility, Newport News Virginia, U.S.A.
  - Elektronen werden beschleunigt und zur Kollision gebracht
  - Teilchenbahnen werden durch Driftkammer registriert
- Starke Strahlung in der Driftkammer $\rightarrow$ Defekte möglich
- Erkennung diser Defekte mithilfe von Algorithmen der künstlichen Intelligenz
  - Deep Learning, Convolutional Neural Networks
  - Charakteristische Muster werden gesucht und erkannt
  - Resultat: *Fault Detector*

WAS IST EIN NETZPLAN?

- Terminplanung großer Projekte sehr komplex
  - Viele einzelne Vorgänge, die untereinander *vernetzt* sind
  - Abhängigkeiten oft stark verzweigt
  - Per Hand kaum aufzulösen
- Wie lange Dauert ein Projekt?
  - Welche Vorgänge dürfen sich nicht verzögern?
  - Was sind die *kritischen Pfade* durch ein Projekt?
  - Wo kann effektiv Zeit gespart werden?
- Technisches Hilfsmittel: Netzplan
  - Verkettung aller Vorgänge nach ihren Abhängigkeiten
  - Basis für automatisierte Berechnungen der relevanten Größen
- Formal: Gerichteter azyklischer Graph
  - Vorgänge bilden Knoten des Graphen
  - Abhängigkeiten legen die Kanten fest

EIGENSCHAFTEN VON VORGÄNGEN

- Zu Beginn spezifiziert:
  - Eindeutige Vorgangsnummer
  - Lesbare Vorgangsbezeichnung
  - Dauer
  - Vorgänger und Nachfolger
- Zu berechnen:
  - Frühester und spätester Anfangszeitpunkt (FAZ und SAZ)
  - Frühester und spätester Endzeitpunkt (FEZ und SEZ)
  - Gesamtpuffer
    - Spielraum, der das Projektende nicht gefährdet
  - Freier Puffer
    - Spielraum, der die früheste Abarbeitung der Nachfolger nicht gefährdet

ALGORITHMISCHE KONSTRUKTION EINES NETZPLANS

1. Einlesen der Vorgänge
2. Initialisierung des Netzplans
   - Sind Vorgänger und Nachfolger konsistent?
   - Hängt der Graph zusammen?
   - Ist der Graph zyklenfrei?
3. Vorwärtsrechnung
   - Berechnung von FAZ und FEZ
4. Rückwärtsrechnung
   - Berechnung von SAZ und SEZ
5. Zeitreserven bestimmen
   - Berechnung von GP und FP

PRÜFEN AUF ZUSAMMENHANG

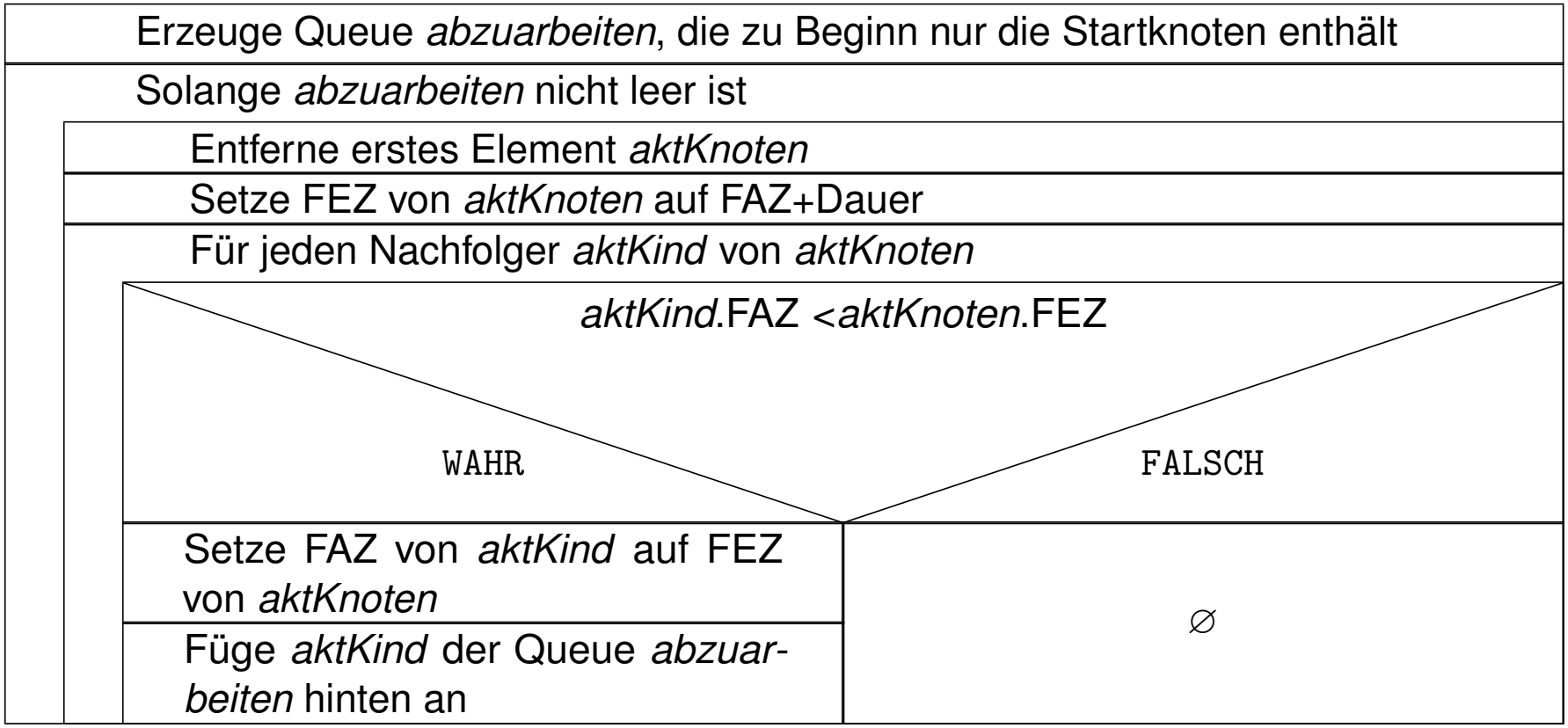
1. Erzeuge Adjazenzmatrix A des Graphen
   - Gibt an, welche Vorgänge wie voneinander abhängen
   - Element a_{ij} ist 1, falls Vorgang i Vorgänger von Vorgang j ist, ansonsten 0
2. Erzeuge symmetrische Version A' von A
   - Richtung der Kanten ist egal für das Zusammenhängen des Graphen
   - Setze $a_{ji} = 1$, falls $a_{ij} = 1$
3. Traversiere nun den Graphen auf Basis von A'
   - Bleiben Knoten übrig, war der Graph nicht zusammenhängend!

PRÜFEN AUF ZYKLEN

1. Erzeuge beginnend bei jedem Startknoten (Knoten ohne Vorgänger) eine Expansion des Graphen
   - D.h. erzeuge sukzessive alle möglichen Pfade durch den Graph
2. Teste in jedem Schritt der Expansion, ob ein Knoten doppelt in einem Pfad vorkommt
   - In diesem Fall wurde ein Zyklus erkannt!

VORWÄRTSRECHNUNG

- Berechne FAZ und FEZ aller Vorgänge
- Aktualisiere die Werte entlang der Abhängigkeiten



RÜCKWÄRTSRECHNUNG

- Berechne SAZ und SEZ aller Vorgänge
- Vorgehensweise analog zur Vorwärtsrechnung

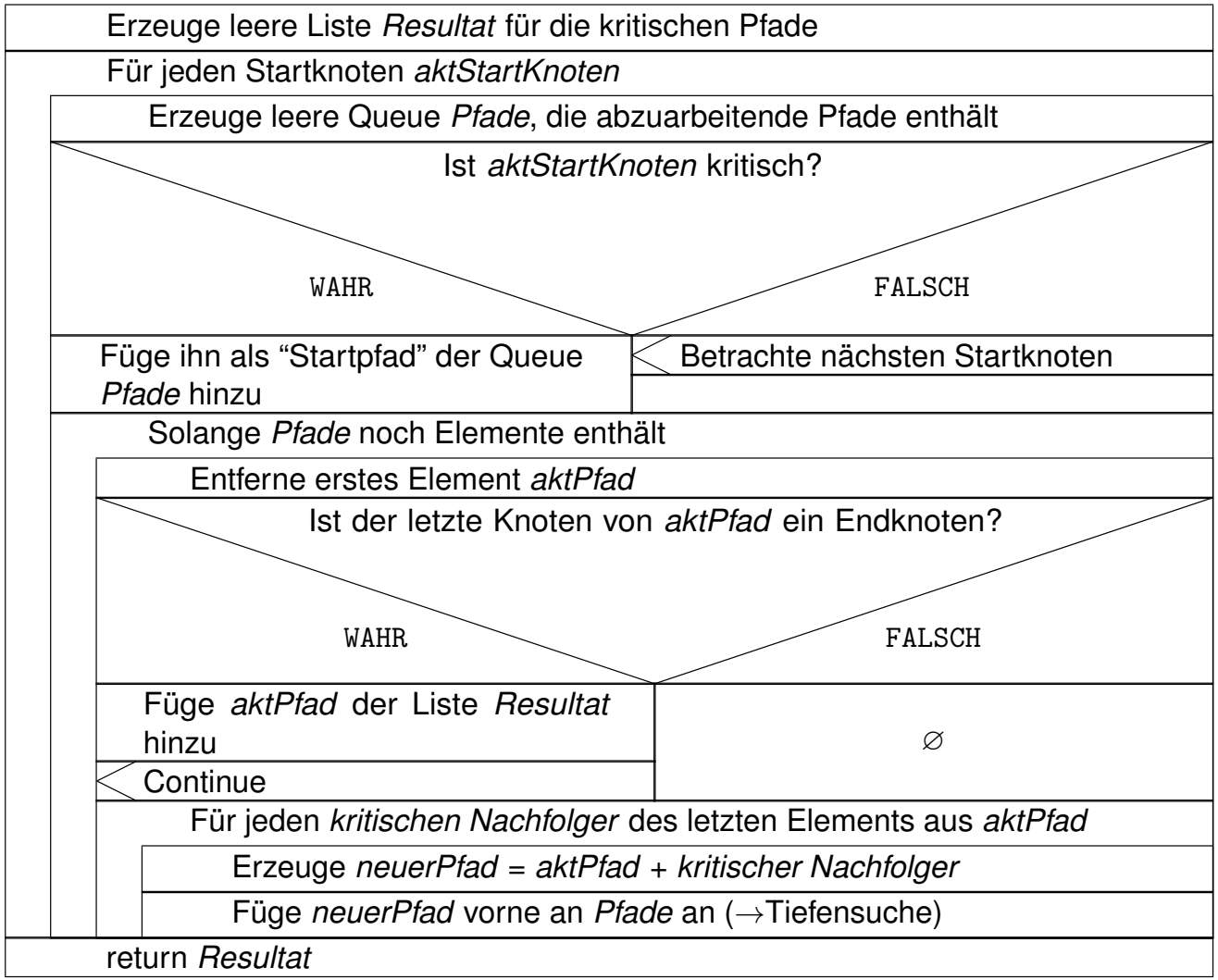
1. Beginne mit einer Queue aus den Endknoten (Knoten ohne Nachfolger)
   - Setze bei allen Endknoten $SEZ = FEZ$, da diese keinen Puffer haben
2. Arbeite die Queue wie bei der Vorwärtsrechnung ab
   - Diesmal setze $SAZ = SEZ - Dauer$
   - Aktualisiere diesmal die *SEZ* der Vorgänger

BERECHNUNG DER ZEITRESERVEN

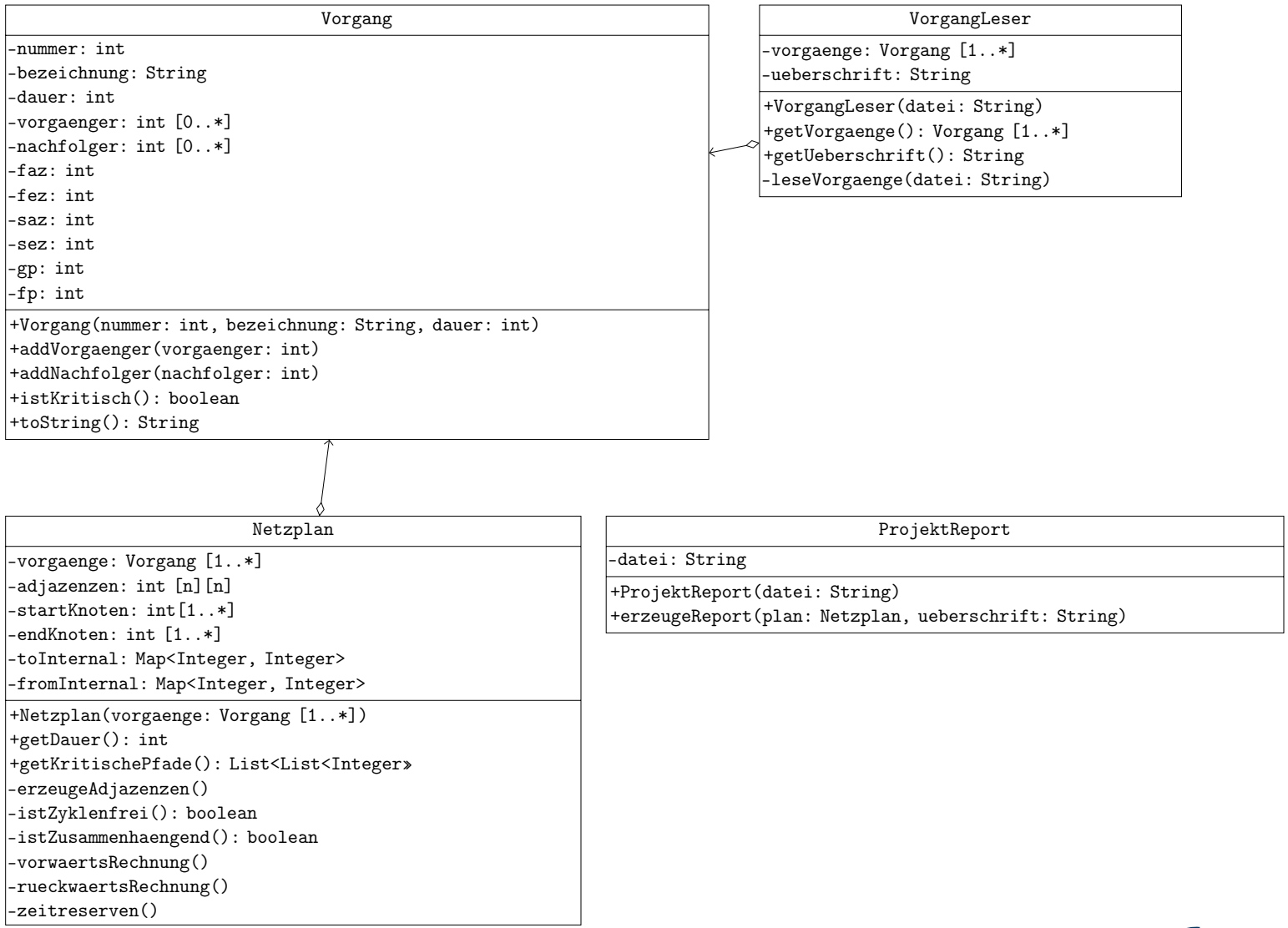
1. Durchlaufe alle Vorgänge
2. Setze $GP = SAZ - FAZ$ (alternativ $GP = SEZ - FEZ$)
3. Setze $FP = \min_{i \in \{Nachfolger\}} FAZ_i - FEZ$

$\rightarrow$ Verwende diese Werte als Grundlage für die Suche nach kritischen Pfaden!

AUFFINDEN DER KRITISCHEN PFADE



OBJEKTORIENTIERTE REALISIERUNG



AUSBLICK

- Implementierung der Präsentationsschicht
  - Bisher nur Logik und Datenhaltung
  - Lässt sich leicht auf der bestehenden Architektur aufsetzen
- Optimierung der Speicherung des Graphen
  - Speicherbedarf der Adjazenzmatrix wächst quadratisch mit der Anzahl der Knoten
  - Ist aber häufig nur dünn besetzt
  - Nutzung dieser Begebenheit, z.B. durch den Einsatz des Compressed Row Storage (CRS) Formats
- Unterstützung weiterer Ein- und Ausgabeformate, z.B. XML oder JSON
  - Kapselung der speziellen Ein- und Ausgabelogik hinter einer abstrakten Klasse
  - Wahl des Formats wird so zur Laufzeit möglich (Konfiguration mit konkreter Strategie)
  - Implementierung durch Einsatz des *Strategy Pattern*