Algorithmen und Datenstrukturen

Einführung:

Zielstellung ♦ Organisatorisches

Algorithmen ♦ Datentypen und -strukturen

Inhalte

- Typische, grundlegende Algorithmen
  - auf Sequenzen und Matrizen
  - auf Bäumen und Graphen
  - auf Mengen

  } **Datenstrukturen**
- Algorithmische Paradigmen
  - Teile und Herrsche (Divide and Conquer)
  - Dynamische Programmierung
  - Greedy-Algorithmen
- **Verwendete Sprachen:**
  - **Python**
  - Pseudocode

Vorausgesetzte Kenntnisse

- Inhalte des Kurses *Grundlagen der Programmierung*
  - Programmierung in Python
    - Kontrollstrukturen
    - Funktionen/Prozeduren
  - Algorithmisches Denken
    - Entwurf einfacher Algorithmen
    - Korrektheit, Terminieren, Effizienz
    - Iteration und Rekursion
  - Graphen
    - Repräsentation von Graphen
    - Abstand von Knoten (Brute-Force, BFS, DFS)
  - Prinzipieller Aufbau einer Rechners (Speicher – Prozessor)

Vorlesung – Übung

- **Vorlesung**
  - Vermittlung der **Konzepte** und **Algorithmen**
  - Voraussetzung für erfolgreiche Teilnahme an den Übungen und der Prüfung
- **Übung**
  - Vertiefung von Teilen des Vorlesungsstoffs, „Training“
  - Finden von Lösungsansätzen
  - Implementieren von Algorithmen

Programmieraufgaben

- *Implementieren Sie ausgewählte Algorithmen und Datenstrukturen zu Hause selbstständig!*
- Wöchentlich ab der dritten Woche (ab 22.04.)
  - Details auf Moodle
  - jeweils eine Woche Bearbeitungszeit (1. Aufgabe: zwei Wochen)
  - Upload der Lösungen auf **GIT.UP**
  - Testfälle beachten, Ein- und Ausgabeformat genau beachten!
  - ➢ *Die Testsuite benutzen, die mit der Aufgabe bereitgestellt wird!*
- GIT.UP: Managementsystem für Software-Projekte auf Basis der Versionsverwaltung GIT

Ablauf der Vorlesungen

- **Folien**
  - enthalten alle Begriffe, Definitionen, Aussagen und einige Beispiele und Erklärungen
  - aber bei weitem nicht alle relevanten Informationen

- **Tafel**

  Viele Beispiele, Überlegungen und Erklärungen werden schrittweise an der Tafel entwickelt.

  Schreiben Sie mit! Das wird Ihnen helfen!

Ablauf der Übungen

- Übungsaufgaben, die Sie vorher zu Hause lösen
- Vorstellung Ihrer Lösungen (in kleinen Schritten)

  Bonuspunkte *für die Klausur durch das Vorstellen von Lösungsschritten*
  - *Wie findet man Ansätze zur Lösung?*
  - *Kreative Diskussion von Varianten*
- *und das Wichtigste ...*
  Diskussion Ihrer **Fragen** zum Vorlesungsstoff

  Trauen Sie sich, Fragen zu stellen und Beiträge zu den Übungsaufgaben zu präsentieren!!!

Leistungserfassung

- **Klausur**
  - nach dem Vorlesungszeitraum
  - bestimmt die Modulnote
- Bonuspunkte
  - für substanzielle Beiträge während der Übungen
  - zur Notenverbesserung von bestandenen Klausuren
- **Prüfungsnebenleistung** zum Abschluss des Moduls: Mindestens 8 Programmieraufgaben müssen erfolgreich bearbeitet und fristgerecht auf den Abgabe-Server geladen werden

Nachmeldung Leistungen

- *Sollten bestandene Leistungen noch nicht in Puls verbucht sein, dann gilt:*
- Leistungen aus den Sommersemestern 2022 und 2023 können **auf Antrag** in diesem Semester verbucht werden
  - Prüfungsnebenleistung Vorlesung (Programmieraufgaben)
  - Prüfungsnebenleistung Übung (keine Leistung erforderlich)
- Voraussetzung: Zulassung als Teilnehmer(in) in Puls
- Voraussetzung: Meldung durch persönliche Nachricht in **Moodle** an mich **bis zum 5. Mai 2024**

Moodlekurs

- Zuordnung zu den **Übungsgruppen** hier!
- Bereitstellung der Lehrmaterialien
- Hinweise zu den Programmieraufgaben
- **Ankündigungen**
- Inhaltliche Fragen im „**Fragen-Forum**“ auf Moodle
  - Nur die Tutoren antworten hier!
  - Zum Diskutieren gibt es das **Studentenforum**.

PULS

1. Belegen

- Bitte Modulnummer ohne –x nehmen.
  Nur wer im **SS 2019** oder früher AuD bereits belegt hatte, verwendet das Modul mit –x !!!
- Vorlesung + Übung (genau eine Gruppe;
  Gruppenzuordnung läuft aber über Moodle)

2. Prüfungsanmeldung

- mindestens acht Werktage vor dem Prüfungstermin (also vor dem Klausurtermin)
- Sie werden von uns zugelassen.
- Ohne Zulassung keine Klausurteilnahme!!!

Algorithmen und Datenstrukturen

Einführung:

Algorithmen ♦ Datentypen und -strukturen

Algorithmen

- Kern zur Lösung von Problemen und Aufgaben mit den Mitteln der Informatik
- **Programme** realisieren Algorithmen (in einer bestimmten Programmiersprache).
- Algorithmen beschreiben die Problemlösung *unabhängig* von Programmiersprachen.
- Dabei werden im Allgemeinen **Daten** verarbeitet.
  - **Eingabedaten** $\xrightarrow{\textit{Abbildung}}$ **Ausgabedaten**
  - **Anweisungsfolge**

Probleme und Algorithmen

- **Spezifikation des Problems**
  - **Eingabe:** Daten, die dem Algorithmus als Eingabedaten gegeben werden
  - **Ausgabe:** Daten, die der Algorithmus aus den Eingabedaten berechnet und damit die Problemstellung beantwortet
- **Algorithmus** beschreibt, *wie* die Eingabedaten in zugehörige Ausgabedaten transformiert werden

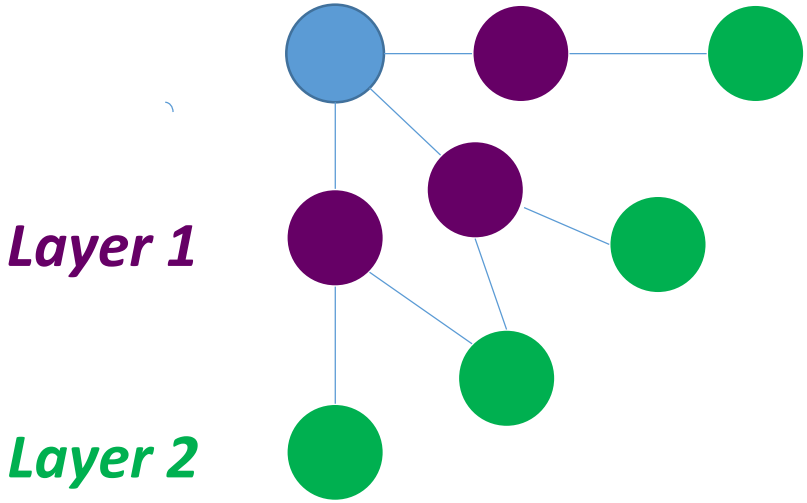
Beispiele

- schriftliche Addition, Multiplikation, Division, ...
- Euklidischer Algorithmus
- Abstand von Knoten in Graphen
- ...

- Algorithmen *unabhängig* von Quelle der Daten
  → **Annahme:** Daten liegen im Speicher vor.
- Algorithmen *abhängig* von Repräsentation der Daten

Wdh.: Breitensuche in Graphen

- zuerst alle Nachbarn eines Knotens bestimmen (*Layer 1*)
- dann für alle Knoten aus *Layer 1* alle (neuen) Nachbarn bestimmen (*Layer 2*)
- usw.



Wdh.: Breitensuche (Markierung)

Eingabe: ungerichteter, schlingenfreier Graph $G = (V,E)$
in Adjazenzlisten-Repräsentation, $u \in V$

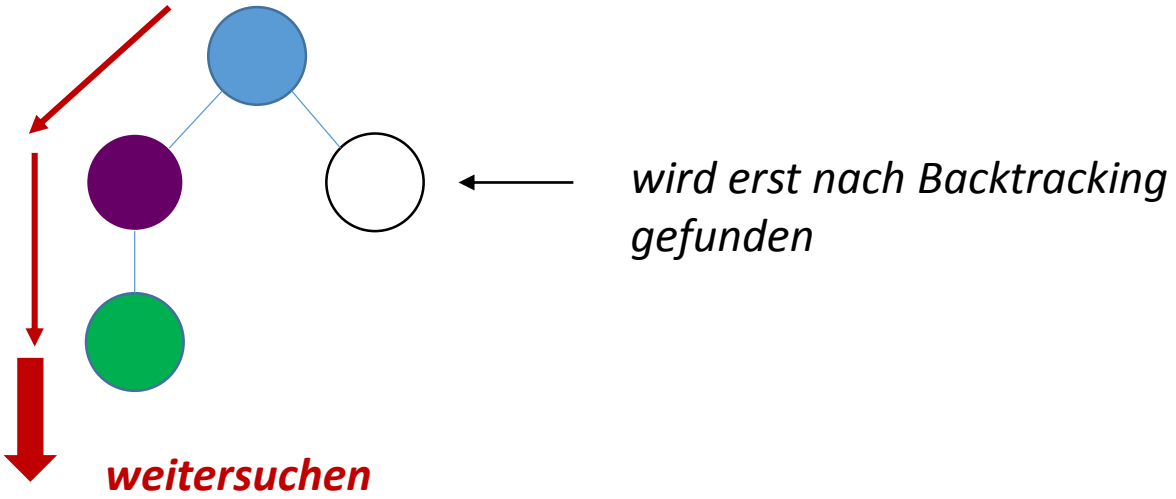
```
Q ⟵ leere Warteschlange          # für Knoten, die noch unmarkierte
                                 # Nachbarn haben könnten
für alle i ∈ V
        mark[i] ⟵ 0
mark[u] ⟵ 1
enqueue(Q,u)
solange Q nicht leer ist
        j ⟵ dequeue(Q)           # j wird untersucht
        für alle k in adj[j]
                falls mark[k] = 0
                        mark[k] ⟵ 1
                        enqueue (Q,k)   # neuer Knoten gefunden
```

Datentypen und Algorithmen

- Algorithmen manipulieren Daten
- Daten sind dabei von einem bestimmten **Datentyp**
  - *am Bsp.:* Adjazenzlisten; Warteschlange
  - bedingen Korrektheit und Effizienz der Algorithmen
- Manipulation der Daten durch für die gewählten Datentypen spezifischen **Operationen**
  - *am Bsp. Liste:* Zugriffsoperation adj[*j*]
  - *am Bsp. Schlange:* enqueue, dequeue

Wdh.: Tiefensuche in Graphen

- von jedem gefundenen Knoten sofort einen neuen Nachbarn suchen
- erst, wenn so kein neuer Knoten gefunden werden kann, zurückgehen zum zuletzt gefundenen Knoten, der noch weitere Nachbarn haben kann: **Backtracking**



Wdh.: Tiefensuche (Markierung)

Eingabe: ungerichteter, schlingenfreier Graph $G = (V,E)$
in Adjazenzlisten-Repräsentation, $u \in V$

für alle $i \in V$
    $mark[i] \longleftarrow 0$
$S \longleftarrow$ leerer Stack
$mark[u] \longleftarrow 1$
push(S,u)
solange S nicht leer ist
    $akt \longleftarrow$ **top**(S)
    falls $k \in$ adj$[akt]$ existiert für das $mark[k] = 0$
        $mark[k] \longleftarrow 1$
        push(S,k)
    sonst
        pop(S)

Abstrakte Datentypen (ADT)

- **Datentypen** sind definiert durch
  - die Menge der darstellbaren **Werte** und
  - die ausführbaren **Operationen**
- **ADT**: Abstraktion von der Art, wie die Werte gespeichert und die Operationen ausgeführt werden
  - nur, welche Operationen erlaubt sind
  - *unabhängig von Realisierung in Programmiersprachen*
- Bedeutend beim implementierungsunabhängigen Entwurf von Algorithmen

Datenstrukturen

- Implementierungen eines ADT
  - feste Darstellung der Werte
  - Realisierung der Operationen (in einer Programmiersprache)
- verschiedene Implementierungen eines ADT
- Datenstrukturen mit den gleichen Operationen (*insbes. verschiedene Implementierungen eines ADT*) können in Algorithmen gegeneinander ausgetauscht werden.
- Implementierung kann die Laufzeit eines Algorithmus beeinflussen, der den ADT verwendet

Spezifikation eines ADT

Queue (Warteschlange) – informal

- **Beschreibung:** FIFO-Sequenz
- **Wertebereich:** Menge aller endlichen Folgen von Elementen eines gewissen Grundtyps
- **Operationen:**

Wdh.: Breitensuche (Markierung)

Eingabe: ungerichteter, schlingenfreier Graph $G = (V,E)$
in Adjazenzlisten-Repräsentation, $u \in V$

```
Q ⟵ leere Warteschlange        # für Knoten, die noch unmarkierte
                               # Nachbarn haben könnten
für alle i ∈ V
        mark[i] ⟵ 0
mark[u] ⟵ 1
enqueue(Q,u)
solange Q nicht leer ist
        j ⟵ dequeue(Q)         # j wird untersucht
        für alle k in adj[j]
                falls mark[k] = 0
                        mark[k] ⟵ 1
                        enqueue (Q,k)   # neuer Knoten gefunden
```

Queue (Warteschlange) – informal

- **Beschreibung:** FIFO-Sequenz
- **Wertebereich:** Menge aller endlichen Folgen von Elementen des Grundtyps
- **Operationen:**

Operation	gibt zurück	Verhalten
empty()	Queue	erzeugt leere Warteschlange
isEmpty(*Q*)	bool	entscheidet, ob *Q* leer ist
enqueue(*Q*,*x*)	Queue	fügt *x* der Queue *Q* hinzu
dequeue(*Q*)	**???**	löscht Frontelement aus *Q* und gibt es zurück

Queue (Warteschlange) – informal

Schnittstelle/Interface

Operation	gibt zurück	Verhalten
empty()	Queue	erzeugt leere Warteschlange
isEmpty(*Q*)	bool	entscheidet, ob *Q* leer ist
enqueue(*Q*,*x*)	Queue	fügt *x* der Queue *Q* hinzu
delete(*Q*)	Queue	löscht Frontelement aus *Q*
front(*Q*)	Grundtyp	liefert Wert des Frontelements

dequeue(*Q*) ≙ front(*Q*); delete(*Q*)

Formale Spezifikation eines ADT-Interface

type Σ =

sorts *verwendete Typen*

functions *Namen, Definitions- und Wertebereiche in der Form*

$f : A \rightarrow B$

end.

Die Bedeutung der Symbole ergibt sich erst durch eine Interpretation.

Interface ADT Queue

type Queue =

sorts T, bool, q

functions

empty: $\rightarrow$ q

isEmpty : q $\rightarrow$ bool

enqueue: q $\times$ T $\rightarrow$ q

delete: q $\rightarrow$ q

front: q $\rightarrow$ T

end.

Operation	gibt zurück
empty()	Queue
isEmpty(*Q*)	bool
enqueue(*Q*,*x*)	Queue
delete(*Q*)	Queue
front(*Q*)	Grundtyp

Interface ADT Boolean

type Boolean =

sorts bool

functions

t : $\rightarrow$ bool

f : $\rightarrow$ bool

not: bool $\rightarrow$ bool

and: bool $\times$ bool $\rightarrow$ bool

or: bool $\times$ bool $\rightarrow$ bool

end.

Interpretation

- Das **Verhalten** eines ADT wird festgelegt, indem
  - den Sorten konkrete Wertemengen und
  - den Funktionssignaturen konkrete Abbildungsvorschriften

  zugeordnet werden.

- Es gibt verschiedene formale und semi-formale Methoden, eine Interpretation anzugeben.

Interpretation für den ADT Boolean

- Zuordnung einer Wertemenge zur neu definierten Sorte

bool ↦ {true, false}

- Definition der Funktionen

t() = true

f() = false

not(true) = false, not(false)=true,

and(false, false) = and(false, true) = and(true, false) = false,

and(true, true) = true,

or(false, false) = false,

or(false, true) = or(true, false) = or(true, true) = true

Ausblick: Formale Spezifikation von Datentypen

- Festlegung des Verhaltens durch Gesetze (Axiomensystem)
- Beispiel Boolean:

  not(true) = false, not(false)=true,

  $\forall x.$ and(false, x) = false,

  $\forall x.$ and(true, x) = x,

  $\forall x\ \forall y.$ or(x, y) = not(and(not(x), not(y)))

Man kann nun beweisen, dass bis auf Isomorphie nur ein Datentyp existiert, der die Spezifikation erfüllt.

Ausblick: Gesetze Beispiel Queue

isEmpty(empty())

$\forall q\ \forall x.$ not(isEmpty(enqueue(q, x)))

$\forall q\ \forall x.$ isEmpty(q) $\Rightarrow$ isEmpty(delete(enqueue(q, x)))

$\forall q\ \forall x.$ isEmpty(q) $\Rightarrow$ front(enqueue(q, x)) = x

$\forall q\ \forall x.$ not(isEmpty(q)) $\Rightarrow$ front(enqueue(q, x)) = front(q)

Semi-formale Interpretation

Operation

pre: Bedingungen, die vor Ausführung der Operation erfüllt sein müssen (**Vorbedingungen**)

post: Bedingungen, die nach Ausführung der Operation garantiert sind (**Nachbedingungen**)

Semi-formale Interpretation: Queue

T: Datentyp des Grundtyps
bool: ADT Boolean
q: Menge aller endlichen Folgen von Elementen von **T** mit**:**

empty()
post: eine neue leere Queue ist erzeugt

isEmpty(*Q*)
post: true, falls *Q* keine Elemente enthält, sonst false

type Queue =

sorts T, bool, q

functions

empty: $\rightarrow$ q

isEmpty : q $\rightarrow$ bool

enqueue: q $\times$ T $\rightarrow$ q

delete: q $\rightarrow$ q

front: q $\rightarrow$ T

end.

Semi-formale Interpretation: Queue

enqueue(*Q*,*x*)

post: Der Schlange ***Q*** ist das Element ***x*** vom Typ **T** als neues letztes Element hinzugefügt.

delete(*Q*)

pre: isEmpty(*Q*) = false

post: Das erste Element ist aus *Q* entfernt.

front(*Q*)

pre: isEmpty(*Q*) = false

post: Das erste Element aus *Q* ist zurückgegeben. *Q* ist unverändert.

type Queue =

sorts T, bool, q

functions

empty: $\rightarrow$ q

isEmpty : q $\rightarrow$ bool

enqueue: q $\times$ T $\rightarrow$ q

delete: q $\rightarrow$ q

front: q $\rightarrow$ T

end.

Vom ADT zur Datenstruktur

Schnittstelle des Datentyps

Interpretation der Sorten- und Funktionssymbole

ADT

Verhalten des Datentyps

Implementierung

verwendbarer Datentyp

Datenstruktur