Algorithmen und Datenstrukturen 1

Felix Ichters, Lukas Dzielski*

Sommersemester 2023

Begleitmaterial zur Vorlesung 'Algorithmen und Datenstrukturen'.

*Universität Heidelberg

Inhaltsverzeichnis

1 Beweise

1.0.1 Obervation

1.0.2 Links, etc

1.0.3 Statements

Ein **Statement/ Aussage** ist ein Mathematischer Ausdruck der entweder wahr oder Fallsch ist.

Beispiel:

- $2 \in \{x \in \mathbb{R} | x < 5\}$ (wahr)
- $3^2 + 5^2 = 8^2$ (falsch)

Dabei werden Ausdrüche wie $0 < x < 1$ verwendet um Mengen zu definieren.

$$A = \{x \in \mathbb{R} | 0 < x < 1\}$$

Wichtig ist hierbei der **Wahrheitswert** eines offenen Ausdrucks $0 < x < 1$ hängt von gewähltem x ab. Also ist

- $x = 1/2$ (wahr)
- $x = 5$ (falsch)

Die **Domäne** ist hierfür wichtig zu beachten. Für $\mathbb{N}$, gibt es kein x s.t. $0 < x < 1$, aber es gibt welche für $\mathbb{R}$

1.0.4 Formal mathimatical Proofs

Ein **formaler mathematischer Beweise** besteht aus einer nummerierten Sequent von **wahren Aussagen**. Jede Aussage in einem Beweis ist ein **Annahme** oder **folgt** aus vorherigen Aussagen dirch Ableitungsregel/ Inferenzregel (rule of inference). Die Letzte Aussage ist die die wir bewiesen haben.
$\Rightarrow$ Offene Aussagen können in Beweisen nicht auftreten!

Beispiel einer Inferenzregel: set definition rule

Wenn ein Element in einer Menge ist, dann können wir definierende Eienschaften ableiten. Andererseits, wenn es die definierende Eigenschaften erfüll, dann können wir ableiten das dasd Element in der Menge ist.

1.0.5 set definition rule: Beispiel

Definiere $C = \{x \in \mathbb{R} | x < 2\}$
($x < 2 \wedge x \in \mathbb{R}$ ist die definierende Eigenschaft). Dabei gibt es zeiw Möglichkeiten für die Ableitung

Möglichkeit 1

1. $a \in C$
2. $a < 2 \wedge a \in \mathbb{R}(1; defC)$

Möglichkeit 2

1. $b < 2 \wedge b \in \mathbb{R}$
2. $b \in C(1; defC)$

Jede Aussage in dem Bewies hat eine Nummer. Wir begründen wie wir eine Aussage ableiten, zb $(1; defC)$ bedeutet wir leiten die aktuelle Aussage aus Aussage 1 mit der Definition von C und der set definition rule ab.

Bemerkung: $\wedge b \in R$ wird oft ausgelassen, wenn Kontext es zulässt.

1.0.6 Makrosteps in Proof

Problem: Schauen wir uns folgenden Bewwis an:
(ass = Annahme (assumption) und prop = Eigenschaft(property))

$$PROOFHERE$$

Ist das ein akzrptabler Beweis? Akzeptanz von Makro-Schritten wie "**prop** $\mathbb{R}$"hängt von der Zielgruppe ab!
Welche Eigenschaft von $\mathbb{R}$ wurde benutzt?

1.1 Einfache Beweistechniken

Beweis durch beispiel **Beispiel:** Zeigen Sie es gibt einePrimzahl zwischen 80 und 90.
Idee: Zeugen angeben für Primzahl (p) für die die Aussage gilt.

Beweis. Wähle $p = 83$ □

Ist das ausreichend?
Eigenflich, **NEIN**. Wie müssen noch **zeigen** das 83 tatsächlich eine Primzahl ist.
Das können wir tun in dem wir alle Teiler ausprobieren.

1.1.1 Wiederlegen von Behauptungen

Behauptung: Nimm an n ist eine Primzahl größer als 1. Dann ist $2^n - 1$ ebenfalls eine Primzahl.
Können Sie die Behauptung beweisen? Try hard $\cdots$
Wenn Sie es nicht können, dann sollen Sie darüber nachednken die Behauptung zu wiederlegen.
Eine Primzahl n für die $2^n - 1$ nicht prim ist, ist genug!
Das Gegenbeispiel ist $n = 11$ da 11 prim ist aber

$$2^{11} - 1 = 2047 = 23 \cdot 89$$

keine Primzahl ist!

1.2 Beweisen und verwenden von Für Alle Aussagen

1.2.1 Inferenzregel für definierte Beziehungen

The definition rule

Angenommen, es wurde eine Beziehung definiert. Wenn die Beziehung gilt (in irgendweinem Beweisschritt oder Annahme), dann kann die definierende Eigenschaft abgeleitet werden. Andererseits, wenn die definierende Eigenschaft gilt, dann kann die Beziehung abgeleitet werden.

Beispiel: Für Mengen A und B, definiere A ist **Teilmenge** von B, $A \subseteq B$, wenn **für alle** x **mit** $x \in A : x \in B$. Mit anderen Worten:
$A \subseteq B$ if and only if (iff) $\forall x((x \in A) \to (x \in B))$ ist wahr.

Möglichkeit 1:

1.	$A \subseteq B$	(ass 2)
2.	für alle x s.t.$x \in A : x \in B$	(1, def $\subseteq$)

Möglichkeit 2:

1.	für alle x s.t.$x \in A : x \in B$	(1, def $\subseteq$)
2.	$A \subseteq B$	(ass 2)

1.2.2 Inferenzeregel für all Aussagen

Sei $\mathfrak{P}$ eine Formel. Beispielsweise steht $\mathfrak{P}(x)$ für $x \in A$ und $\mathfrak{Q}(x)$ steht für $x \in B$. Dann kann "**für alle x s.t.** $x \in A : x \in B$äls "**für alle x s.t.** $\mathfrak{P}(x) : \mathfrak{Q}(x)$"geschrieben werden.

Regeln um $\forall$ Aussagen zu beweisen (pr $\forall$)

Um Aussagen der Form "**für alle** x **in s.t.** $\mathfrak{P}(x) : \mathfrak{Q}(x)$", zu beweisen nimmt man an x sei **beliebig gewähtes Element (eigenvariable)** s.t $\mathfrak{P}(x)$ wahr ist. Dann zeige man $\mathfrak{Q}(x)$ ist wahr.

Generaliesirungen z.B. "für alle x, y s.t. $\mathfrak{P}(x, y) : \mathfrak{Q}(x, y)$"möglich

Ein Beispiel Sei $C = \{x \in \mathbb{R} | x < 1\}$ und $D = \{x \in \mathbb{R} | x < 2\}$. Zeige $C \subseteq D$!

$$ProofHere!$$

Wie können wir "für alle x s.t $\mathfrak{P}(x) : \mathfrak{Q}(x)$"wiederlegen?

Bemerkungen

- Durch Einrückung kennzeochnen wir **Teilbeweise** die von einer ANnahme wie SSei $x \in C$ beliebigäbhängen.
- Eine Annahme hat keine Begründung.
- Teilbeweise 2-4 basieren auf der Annahme in 1
- Schritte aus 1-4 können nicht in Begründungen auftauchen, sobald der Teilbeweis fertig ist (d. h, **nach pr** $\forall$ **in 5**)
- Wir schreiben oft "für alle $x \in C : x \in D$ßtatt "für alle $x s.t. x \in C : x \in D$"

1.3 Verwenden von für alle Aussagen

1.3.1 Inferenzeregel um all Aussagen zu verwenden

Die Regel um $\forall$ Aussagen in Beweisen zu verwenden (us $\forall$)

Wenn wir wissen das eine Aussage "für alle x s.t $\mathcal{P}(x) : \mathcal{Q}(x)$"wahr ist und wir $\mathcal{P}(t)$ berets als einen Schritt für eine Variable t im Beweis haben, dann können wir $\mathcal{Q}(t)$ ableiten.

Beispiel

1.	$t \in A$	
2.	für alle x s.t.$x \in A : x \in B$	
3.	$t \in B$	$(1, 2; us\forall)$

Beispiel

1.	$\lvert a\rvert < \lvert b\rvert$	
2.	für alle x s.t.$\lvert x\rvert \leq \lvert y\rvert : x^2 \leq y^2$	
3.	$a^2 < b^2$	$(1, 2; us\forall)$

Beispiel Seiesn A, B, C Mengen. Zeige $\subseteq$ ist transitiv, d. h, zeige $A \subseteq B$ und $B \subseteq B$, dann $A \subseteq C$.

Annahme: A, B, C Mengen

1. $A \subseteq B$
2. $B \subseteq C$

Zeige: $A \subseteq C$

1. $x \in A$ beliebig

1.4 Beweisen und verwenden von Oder Aussagen

1.4.1 Inverenzregeln u oder Aussagen zu verwenden

Die Regel um $\vee$ Aussagen in Beweisen anzuwenden (us $\vee$)

Wenn wir wissen das "$\mathfrak{P}$ oder $\mathfrak{Q}$"wahr ist und wir bewiesen können das $\mathfrak{R}$ wahr ist wenn $\mathfrak{Q}$ gilt sowie das $\mathfrak{R}$ wahr ist wenn $\mathfrak{P}$ gilt, dann können wir ableiten das $\mathfrak{R}$ gilt.

$\Rightarrow$ Dies nennt man auch Fallunterscheidung!

Definition 1 Gegeben Mengen A und B, die Vereinigung von A und B, $A \cup B = \{x | x \in A \text{ oder } x \in B\}$.

Beispiel Für Mengen A, B, C, wenn $A \subseteq C$ und $B \subseteq C$, dann $(A \cup B) \subseteq C$.

Annahme: A, B, C Mengen

1. $A \subseteq C$
2. $B \subseteq C$

Zeige: $A \cup B = \{x | x \in A \text{ oder } x \in B\}$

1.	Sei $x \in A \cup B$ beliebig	
2.	$x \in A$ oder $x \in B$	(1; def $\cup$)
3.	Fall 1: Annahme $x \in A$	
4.	für alle $t \in A$: $t \in C$	(ass 1; def $\subseteq$)
5.	$x \in C$	(3, 4; us $\forall$)
6.	Fall 2: Annahme $x \in B$	
7.	für alle $t \in B$: $t \in C$	(ass 2; def $\subseteq$)
8.	$x \in C$	(6, 7; us $\forall$)
9.	$x \in C$	(2, 3-8; us $\vee$)
10.	für alle $x \in A \cup B$: $x \in C$	(1-9; pr $\forall$)
11.	$(A \cup B) \subseteq C$	(10; def $\subseteq$)

Erweiterte Definitionsregel(def^2)

Wenn eine Aussage $\mathfrak{P}$ die definierende Eigenschaft von einer Definition ist, ist es zulässigen $\mathfrak{P}$ zu verwenden oder zu beweisen **ohne** $\mathfrak{P}$ selbst als Schritt anzuführen. Als Begründung für den abgeleiteten Schritt gibt man die Definition und nicht die Regel um $\mathfrak{P}$ zu verwenden oder zu beweisen.

$\Rightarrow$ kürzere Beweise (mit ausgelassenen Details)

Beispiel

1.	$a \in M$	
2.	$M \subseteq N$	
3.	$a \in N$	(1, 2; def^2 $\subseteq$)

Warnung: Später verwenden wir def und def^2 synonym!

1.4.2 Inverenzregeln um oder Aussagen zu beweisen

Beweisregel für $\vee$ (pr $\vee$)

Wenn $\mathfrak{P}$ als Schritt in einem Beweis etabliert wurde, dann kann "$\mathfrak{P}$ oder $\mathfrak{Q}$"als neue Zeile geschrieben werden. Symmetrisch, wenn $\mathfrak{Q}$ als Schritt in einem Beweis etabliert wurde, dann kann "$\mathfrak{P}$ oder $\mathfrak{Q}$"als neue Zeile geschrieben werden.

Beispiel Für Mengen A, B, C: wenn $A \subseteq B$ oder $A \subseteq C$, dann $A \subseteq B \cup C$

Annahme: A, B, Mengen

1. $A \subseteq B$ oder $A \subseteq C$

Zeige: $A \subseteq (B \cup C)$

1.	Sei $x \in A$ beliebig	
2.	$A \subseteq B$ oder $A \subseteq C$	(ass 1)
3.	Fall 1: Annahme $A \subseteq B$	
4.	$x \in B$	(1, 3; def $\subseteq$)
5.	$x \in B$ oder $x \in C$	(4; pr $\vee$)
6.	Fall 2: Annahme $A \subseteq C$	
7.	$x \in C$	(1, 6; def $\subseteq$)
8.	$x \in B$ oder $x \in C$	(7; pr $\vee$)
9.	$x \in B$ oder $x \in C$	(2, 3-8; us $\vee$)
10.	$x \in (B \cup C)$	(9; def $\cup$)
11.	für alle $x \in A$: $x \in (B \cup C)$	(1-10; pr $\forall$)
12.	$A \subseteq (B \cup C)$	(11; def $\subseteq$)

1.4.3 Generalisierung oder Inferenzregeln

Regel um $\vee$ in Beweisen zu verwenden (us $\vee$) final

Wenn wir wissen das "$\mathfrak{P}_1 oder \mathfrak{P}_2$ oder $\cdots \mathfrak{P}_n$"wahr ist und wir beweisen das $\mathfrak{R}$ in allen fällen nicht zu einem Wiederspruch führt, dann können wir ableiten das $\mathfrak{R}$ wahr ist.

Regel um $\vee$ in Beweisen zu beweisen (us $\vee$ final)

Wir können "$\mathfrak{P}_1$ oder $\mathfrak{P}_2$ oder $\mathfrak{P}_n$äls Schritt in einem Beweis aufführen falls wir einen von $\mathfrak{P}_1$ bis $\mathfrak{P}_n$ im Beweis etabliert haben.

1.5 Beweisen und verwenden von Und Aussagen

Regel um $\wedge$ zu verwenden (us $\wedge$)

Wenn "$\mathfrak{P}$ und $\mathfrak{Q}$ëin Schritt in einem Beweis ist, dann können wir sowohl $mathfrakP$ als auch $\mathfrak{Q}$ als Schritt aufführen.

Beipiel

1. $a < 1$ und $a \in A$
2. $a < 1$ (oder 2. $a \in A$) (1; us $\wedge$)

Regel um $\wedge$ zu beweisen (pr $\wedge$)

Um "$\mathfrak{P} und \mathfrak{Q}$ïn einem Beweis zu zeigen, zeige $\mathfrak{P}$ und zeige ebenfalls $\mathfrak{Q}$

Beispiel

i.	$\mathfrak{P}$	
	$\vdots$	
j.	$\mathfrak{Q}$	
	$\vdots$	
k.	$\mathfrak{P}$ und $\mathfrak{Q}$	(i, j; pr $\wedge$)

Beispiel: Zeige für Mengen A, B das gilt $A \cap B = B \cap A$

1.	Sei $x \in A \cap B$ beliebig	
2.	$x \in A$ oder $x \in B$	(1; def $\cap$)
3.	$x \in A$	(2; us $\wedge$)
4.	$x \in B$	(2; us $\wedge$)
5.	$x \in B$ und $x \in A$	(4, 3; pr $\wedge$)
6.	$x \in B \cap A$	(5; def $\cap$)
7.	für alle $x \in A \cap B : x \in B \cap A$	(1 - 6; pr $\forall$)
8.	$A \cap B \subseteq B \cap A$	(7; def $\subseteq$)
9.	$B \cap A \subseteq A \cap B$	(1 - 8; symnetry)
10.	$A \cap B \subseteq B \cap A$ und $B \cap A \subseteq A \cap B$	(8, 9; pr $\wedge$)
11.	$A \cap B = B \cap A$	(10; def $=$)

In 11, benutzen wir die Definition "=", d.h, $\mathbf{A = B}$ **iff** $\mathbf{A \subseteq B \wedge B \subseteq A}$

Symmetrieregel

Symmetrieregel

Wenn $\mathfrak{P}(A_1, B_1, \cdots)$ eine Aussage ist die bewiesen wurde für beliebige $A_1, B_1, \cdots$ in den Annahmen und Hypotesen, und falls $A_2, B_2, \cdots$ eine Permutatuion von $A_1, B_2, \cdots$ ist dann ist $\mathfrak{P}(A_2, B_2, \cdots)$ wahr. Dies lässt sich auch auf universelle Variable in für alle Aussagen übertragen, d.h., wenn für alle $A_1, B_1, \cdots : \mathfrak{P}(A_1, B_1, \cdots)$ wahr ist, dann ist für alle $A_2, B_2, \cdots : \mathfrak{P}(A_2, B_2, \cdots)$ ebenfalls wahr.

Beispiel von oben:

$$\begin{array}{cccc} A \cap & B \subseteq & B \cap & A \\ \downarrow & \downarrow & \downarrow & \downarrow \\ B \cap & A \subseteq & A \cap & B \end{array}$$

Tausche A durch B und B durch A

1.6 Theoreme verwednen

Regel für Substitutionen (subs)

Jeder Name oder Repräsentant eines mathematischen Objekts kann durch einen anderen Namen/ Repräsentanten des gleichen Objekts ersetzt werden. Gleiche Namen für unterschiedliche Objekte dürfen nicht verwendet werden.

Beispiel 1

1.	$A \cap B = C$	
2.	$A = D$	
3.	$D \cap B = C$	(1,2; subs)

Beispiel 2

1.	$x^2 + x = 6$	
2.	$x = y + 1$	
3.	$(y+1)^2 + y + 1 = 6$	(1,2; subs)

Theoremregel (thm)

um ein Theorem auf Schritte in einem Beweis anzuwenden, finde eine Aussage $\mathfrak{P}$ die äquivalent zur Aussage vom Theorem ist. Dann kann $\mathfrak{P}$ als neuer Schritt im Beweis aufgeführt werden oder durch Substitution verwendet werden um einen Schritt zu verändern.

- Dies ist eine Möglichkeit Lemmas in Beweisen zu verwenden
- Weitere Möglichkeiten $\rightarrow$ später

Beispiel: Zeuge für Mengen A, B, C: es gilt $A \cup (B \cup C) = (A \cup B) \cup C$

Annahmen: A, B, C Mengen

Zeige: $A \cup (B \cup C) = (A \cup B) \cup C$

1.	Sei $x \in (A \cup B) \cup C$ beliebig	
2.	$x \in (A \cup B)$ oder $x \in C$	(1; def $\cup$)
3.	Fall 1: $x \in A \cup B$	
4.	$x \in A$ oder $x \in B$	(3; def $\cup$)
5.	Fall 1a: $x \in A$	
6.	$x \in A \cup (B \cup C)$	(5; def $\cup$)
7.	Fall 1b: $x \in B$	
8.	$x \in B \cup C$	(7; def $\cup$)
9.	$x \in A \cup (B \cup C)$	(8, def $\cup$)
10.	$x \in A \cup (B \cup C)$	(4, 5-9; us $\vee$)
11.	Fall 2: $x \in C$	
12.	$x \in B \cup C$	(11; def $\cup$)
13.	$x \in A \cup (B \cup C)$	(12; def $\cup$)
14	$x \in A \cup (B \cup C)$	(2, 3 - 13; us $\vee$)
15	$(A \cup B) \cup C \subseteq A \cup (B \cup C)$	(1, 2 - 14; def $\subseteq$)
16	$C \cup (B \cup A) \subseteq (C \cup B) \subseteq A$	(15, Thm $X \cup Y = Y \cup X$)
17.	$A \cup (B \cup C) \subseteq (A \cup B) \cup C$	(16; symmetry (AC))
18.	$A \cup (B \cup C) = (A \cup B) \cup C$	(15, 17; def $=$)

1.6.1 Substitution anwenden

Beispiel

Beispiel...

1.7 Beweisen und verwenden von Implikationen

1.7.1

1.8 law of Excludet Middle

1.9 Equivalenz und iff Aussagen

1.9.1 Inferenzeregel um if-then Aussagen zu beweisen

Regel um Implikationen zu verwenden (us $\rightarrow$) (oder modus ponens (MP))

Wenn $\mathfrak{P}$ und if $\mathfrak{P}, then \mathfrak{Q}$SSchritte in einem Beweis sind, dann können wir $\mathfrak{Q}$ ableiten und als Schritt schreiben.

i.	$\mathfrak{P}$	
	$\vdots$	
j.	if $\mathfrak{P}$, then $\mathfrak{P}$	
j+1	$\mathfrak{Q}$	(i, j; us $\rightarrow$)

Beispiel

1.	if $x < 2$, then $x \in A$	
2.	$x < 2$	
3.	$x \in A$	(1, 2; us $\rightarrow$)

1.9.2 Beweisen und verwenden von Äquivalenzen

Beweisen von Äquivalenzen (pr $\leftrightarrow$)

Um zu zegen "$\mathfrak{P}$ ist äquivalent zu $\mathfrak{Q}$"nimm zuerst an $\mathfrak{P}$ gilt und zeige $\mathfrak{Q}$, und dann nimm $\mathfrak{Q}$ an und zeige $\mathfrak{P}$

Verwenden von Äquivalenzen (pr $\leftrightarrow$)

Eine Aussage darf durch eine äquivalente Aussage ersetzt werden.

1.9.3 iff Aussagen

- $\mathfrak{P}$ iff $\mathfrak{Q}$ gilt genau dann wenn $\mathfrak{P} \leftrightarrow \mathfrak{Q}$ ist wahr
- Zeige $\mathfrak{P}$ iff $\mathfrak{Q}$: beweise ïf $\mathfrak{P}$, then $\mathfrak{Q}$ünd ïf $\mathfrak{Q}$, then $\mathfrak{P}$"
- ïf $\mathfrak{P}$, then $\mathfrak{Q}$"wird typischerweise bewiesen durch Annahme von $\mathfrak{P}$ und Ableitung von $\mathfrak{Q}$
- Oder ïf $\mathfrak{P}$, then $\mathfrak{Q}$"Kontraposition beweisen:
  Zeige ïf $\mathfrak{P}$, then $\mathfrak{Q}$": Annahme von $\neg\mathfrak{Q}$ um Herleitung $\neg\mathfrak{P}$

1.10 Beweis durch Wiederspruch

1.10.1 Das Ableitungsschema

Idee: Nimm die Negation von $\mathfrak{P}$ an und leite einen Wiederspruch her!

$\vdots$

i. $\mathfrak{Q}$

$\vdots$

j. Annahme $\neg\mathfrak{P}$ (um einen Wiederspruch zu erhalten)

$\vdots$

k. $\neg\mathfrak{Q}$ (Wiederspruch zu $\mathfrak{Q}$ bei i.)

k+1. $\mathfrak{P}$

Beispiel Zeige: $\forall x \in \mathbb{R} \wedge x \in [0, \pi/2] : sinx + cos \geq 1$

1.
2.
3.

1.11 Existenzaussagen

1.11.1 Verwendung

Existenzaussagen verwenden (us $\exists$)

Um eine Aussage "$\mathfrak{P}(j)$ füe ein $1 \leq j \leq n$"in einem Beweis zu verwenden, schreibe "Wähle $1 \leq j_0 \leq n$ s.t. $\mathfrak{P}(j_0)$". Das definiert das Symbol j_0. Beide Ausdrücke $1 \leq j_0 \leq n$ und $\mathfrak{P}(j_0)$ können später im Beweis verwendet werden.

Beispiel: Für $i = 1, 2, \cdots, 10$,definiere $A_i = \{t \in \mathbb{R} | 0 < t < \frac{1}{i}\}$

1.11.2 beweisen (leichte Variante)

1.11.3 Erweiterte Existenzaussagen

Negierung

Induktion

Induktionsprinzipien

How dows it work behind the scene?

Mathematische Induktion

Mathematische Induktion(Beispiel)

2 Einführung

2.1 Algorithmenanalyse

2.1.1 Asymptotische Algorithmenanalyse

Zeitkomplexität Unter Zeitkomplexität eines Problems wird die Anzahl der Rechenschritte die ein Algorithmus zur Lösung des Problems benötigt, in Abhängigkeit der Länge der Eingabe. ...

3 Fogen, Felder und Listen

Folgen spielen in der Informatik eine Überragende Rolle. Das sieht man schon an der Vielzahl von Begriffen: Folge, **Feld**, Schlange, **Liste**, Datei, Stapel, Zeichenkette, Log, $\cdots$. Wir unterscheiden:

- **abstrakter** Begriff [2, 3, 5, 7, 9, 11, $\cdots$] - Mathe
- Funktionalität (stack, $\cdots$) - Softwaretechnik
- Repräsentation - Algorithmik

Anwendungen $\cdots$

Form Follows Function

Operation	LIST	SLIST	UArray	CArray	explanation
$[\cdot]$	n	n	1	1	
$\|\cdot\|$	1*	1*	1	1	not with inter-list
FIRST	1	1	1	1	
LAST	1	1	1	1	
INSERT	1	1*	n	n	InsertAfter only
REMOVE	1	1*	n	n	RemoveAfter only
PUSHBACK	1	1	1*	1*	amortized
PUSHFRONT	1	1	n	1*	amortized
POPBACK	1	n	1*	1*	amortized
POPFRONT	1	1	n	1*	amortized
CONCAT	1	1	n	n	
SPLICE	1	1	n	n	
FINDNEXT	n	n	n^*	n^*	cache-efficient

3.1 Verkettete Listen

Eine verkettete Liste ist eine häufig verwendete Art von Datenstruktur, bei der jedes Element in der Liste Informationen über das nächste Element enthält. Es gibt verschiedene Arten von verketteten Listen, darunter einfach verkettete Listen und doppelt verkettete Listen.

3.1.1 Doppelt verkettete Listen

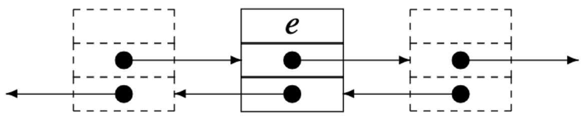
Bei doppelt verketteten Listen enthält jedes Listenelement Informationen sowohl über das vorherige als auch das nächste Element. Dies ermöglicht eine effiziente Navigation in beide Richtungen innerhalb der Liste.

In der Vorlesung haben wir eine Implementierung von doppelt verketteten Listen betrachtet. Jedes Listenelement wird durch die Klasse Ïtem"repräsentiert. Ein Item enthält zwei Handles (Zeiger) auf die vorherigen und nächsten Elemente in der Liste sowie ein Element selbst, das den eigentlichen Datenwert enthält.

Hier ist die Definition der Klasse Ïtem":

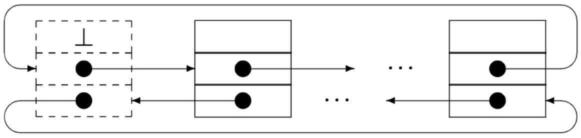
Listenglieder (Items)

```
Class Handle = Pointer to Item
Class Item of Element // one link in a doubly linked list
    e : Element
    next : Handle
    prev : Handle
    invariant next→prev = prev→next = this
```



Probleme:

- **Vogränger des ersten** Listenelements?
- **Nachfolger des letzten** Listenelements?



Trick: Dummy-Header Der Trick mit dem Dummy-Header ist ein nützliches Werkzeug zur Verbesserung der Implementierung und Verwendung von Listenstrukturen. Er steigert Lesbarkeit, Effizienz und Eleganz des Codes und erleichtert das Testen. Durch Vermeidung von Sonderfällen und Aufrechterhaltung der Invariante der verketteten Liste gewährleistet der Dummy-Header reibungslose und zuverlässige Listenoperationen.
Der Dummy-Header ist eine Technik zur Vereinfachung der Implementierung von Listenstrukturen. Durch Verwendung eines speziellen Listenelements am Anfang und Ende der Liste können viele Sonderfälle vermieden werden. Er dient als Platzhalter und Verweis für den Anfang und das Ende der Liste.
Der Dummy-Header bietet folgende Vorteile bei der Verwendung von Listen:

+ Die Invariante der verketteten Liste wird immer erfüllt. Der Dummy-Header stellt sicher, dass es immer einen Vorgänger für das erste Element und einen Nachfolger für das letzte Element gibt.
+ Durch Vermeidung vieler Sonderfälle wird die Implementierung einfacher, lesbarer und eleganter. Es müssen keine zusätzlichen Bedingungen für leere Listen oder Listen mit einem einzigen Element überprüft werden.
+ Der Zugriff auf das erste und letzte Element der Liste ist effizient und erfordert keine aufwändige Traversierung der gesamten Liste.
+ Der Dummy-Header erleichtert das Testen der Listenimplementierung, da die speziellen Fälle von leeren Listen und Listen mit einem einzigen Element bereits abgedeckt sind.
- Der zusätzliche Speicherplatz, der für den Dummy-Header benötigt wird. Dieser Faktor ist in der Regel vernachlässigbar, besonders bei längeren Listen.

Die Definition der Listenklasse mit dem Dummy-Header lautet:

```
class List of Element {
    // Item h ist der Vorgänger des ersten Elements und der Nachfolger des letzten Elem
    // Pos. vor jedem eigentlichen Element

    // !help

    // Einfache Zugriffsfunktionen
    Function head: Handle; return Adresse von h
    Function isEmpty: {0, 1}; return h.next = head // ⟨⟩?
    Function first: Handle; assert ¬isEmpty; return h.next
    Function last: Handle; assert ¬isEmpty; return h.prev
}
```

```
Procedure splice(a,b,t : Handle) {
    //!help
}
```

Der rest sind Einzeiler (?)

```
// Moving elements around within a sequence.
// ⟨... , a, b, c ... , a', c', ...⟩ → ⟨... , a, c ... , a', b, c', ...⟩
Procedure moveAfter(b, a' : HANDLE) splice(b, b, a')
Procedure moveToFront(b : HANDLE) moveAfter(b, HEAD)
Procedure moveToBack(b : HANDLE) moveAfter(b, LAST)
```

Oder doch nicht? Speicherverwaltung! Die Speicherverwaltung in einer Programmiersprache kann potenziell sehr langsam sein. Ein Beispiel dafür ist eine Variable, die einmal erstellt wurde, wie zum Beispiel ein statisches Element in Java. Die *FREELIST* enthält ungenutzte Elemente, während *CHECKFREELIST* sicherstellt, dass sie nicht leer ist. In realen Implementierungen gibt es verschiedene Ansätze:

1. Ein naiver Ansatz, der jedoch eine gute Speicherverwaltung bietet.
2. Verfeinerte Konzepte zur Verwaltung der *FREELIST*, die über Klassen hinweg angewendet werden können und die Freigabe von Elementen ermöglichen.
3. Anwendungsspezifische Ansätze, zum Beispiel wenn man im Voraus weiß, wie viele Elemente insgesamt benötigt werden.

Items Löschen

```
// ⟨... , a, b, c, ...⟩ 7 → ⟨... , a, c, ...⟩
Procedure remove(b : HANDLE) moveAfter( b, freeList.head)
Procedure popFront remove(FIRST)
Procedure popBack remove(LAST)
```

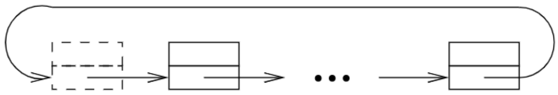
Elemente einfügen

Ganze (Teil)Listen Manipulieren

Suchen

Funktionalität ↔ Effizienz Wir betrachten folgendes Beispiel um Listenlängen zu verwalten. Neben den vorhandenen Elementen verfügt die Liste nun auch über eine zusätzliche Eigenschaft namens `SIZE`.
Problem: inter-list SPLICE geht nicht mehr in konstanter Zeit. Die Moral dabei ist das es keine perfekte Art, Listen zu implementieren gibt. Es hängt von den spezifischen Anforderungen und den gewünschten Operationen ab.

3.1.2 Einfach verkettete Listen



Vergleich mit doppelt verketteten Listen

- weniger Speicherplatz
- Platz ist oft auch Zeit
- eingeschränkter, z. B. kein remove
- merkwürdige Benutzerschnittstelle, z. B. removeAfter

Einfach verkettete Listen – Invariante?

Einfach verkettete Listen – SPLICE

Einfach verkettete Listen – PUSHBACK

Listen: Zusammenfassung, Verallgemeinerungen

Felder (Arrays)

Beschränkte Felder (Bounded Arrays)

3.2 Unbeschränkte Felder (Unbounded Arrays)

Unbeschränke Felder – Anwendungen

Unbeschränke Felder – Grundidee

Unbeschränke Felder mit teilweise ungenutztem Speicher

Kürzen

3.2.1 Amortisierte Komplexität unbeschr. Felder

Beweis: Konto-Methode (oder Versicherung)

3.3 Amortisierte Analyse – allgemeiner

Amortisierte Analyse – Diskussion

3.4 Stapel und Schlangen

3.5 Vergleich: Listen – Felder

Iterieren

Einfügen an zufälliger Position

Ausblick: Weitere Repräsentationen von Folgen

4 Hashing

...

4.1 Hashing mit verketteten Listen

Analyse

Zufällige

4.2 Universelles Hashing

idee: Nutze nur betrimmre

Eine einfache universelle Familie

m ei eine primzahl, $Key \subseteq \{0, \cdots, m-1\}^k$
Satz 3 Für $a = (a_1,)$

4.3 Hashing mit Linearer Suche (Linear Probing)

Offenees Hashing

Der einfache Teil

```
your
code
example
```

Remove

```
your
code
example
```

Verketten $\leftrightarrow$ Lineare Suche

Volllaufen: Verketten weniger empfindlich. Unbeschränktes **offenes** Hashing hat nur amortisiert konstante Einfügezeit.

Cache: Lineare Suche besser. Vor allem **DOALL**

Zeit Abwägung: Kompliziert! Abhämgig von n, **Füllgrad**, **Elementgröße**,

5 Sortieren

Formaler

Gegeben: Elementfolge $s = [e_1, \cdots, e_n]$
Gesucht: $s' = [e'_1, \cdots, e'_n]$

- s' ist Permutation von s
- $e'_1 \leq \cdots \leq e'_n$ für eine **lineare Ordnung** $' \leq'$

Anwendungsbeispiele

- Allgemein: Vorverarbeitung
- Suche: **Telefonbuch** $\leftrightarrow$ unsortierte Liste
- Gruppieren (Alternative Hashing?)

Beispiel aus Kurs/ Buch

- Aufabau von Suchbäumen
- Kruskals MST- Algorithmus
- Verarbeitung von Intervallgraphen (z.B Hotelbuchungen)
- Rucksackproblem
- Scheduling, die schwersten Probleme zuerst
- Sekundärspeicheralgorithmen, z.B Datenbank-Join

Viele verwandte Probleme. Zum Beisoiel **Transposition** dünner Matrizen, **invertierten Index** aufbauen, Konversion zwischen Graphrepräsentationen.

Überblick

- Einfache Algorithmen/ kleine Datenmengen
- **Mergesort** - ein erster effizienter Algorithmus
- Eine passende **untere Schranke**
- **Quicksort**
- das Asuwahlproblem
- ganzzahlige Slüssel - jenseits der unteren Schranke

5.1 Einfache Sortieralgorithmen

Sortieren durch Einfügen (inster sort)

procedure MyAlgorithm(x)
    $y \leftarrow 0$
    for $i \leftarrow 1$ to 10 **do**
        if i is odd **then**
            $y \leftarrow y + x$
        end if
    end for
    return y
end procedure

Sentinels am Beispiel Sortieren durch Einfügen

procedure insertSort(a : Array [1..n] of Elements)
    for $i \leftarrow 2$ to n do **do**
        invariant $a[1] \leq \ldots \leq a[i-1] \cdots$
    end for
    return y
end procedure

Analyse

5.2 Sortieren durch Mischen

idee: Teile und Hersche

procedure mergSort($[e_1, \cdots, e_n]$: Sequence of Elements)
    if $n = 1$ **then**
        return $[e_1]$
    end if
end procedure

Mischen(merge)

Beispiel

Mischen

Analyse

$$T(n) = O(n) + T(\lceil n/w \rceil) + T(\lfloor n/s \rfloor)$$

Problem: Runderei

5.3 Untere Schranken

geht es schneller als $\Theta(nlogn)$?

Eine vergleisbasiere untere Schranke

Vergkeichsbasiertes Sortieren: Informationen über Elemente nur durch Zwei-Wege-Vergleich $e_i \leq e_j$?

Satz: Deterministische vergleichsbasierte Sortieralgorithmen brauchen

$$\mathbf{n} \ \mathbf{log} \ \mathbf{n} - O(n)$$

Vergleiche im schlechtesten Fall.

Beweis: Betrachte Eingaben, die Permutatuiónen von $1..n$ sin. Es gibt genau $n!$ solche Permutationen.

Baumbasierte Sortierer-Darstellung

Beweis

Baum der **Tiefe T** hat höchstens 2^T Blätter.

$$\Rightarrow 2^T \geq n!$$

$$\Leftrightarrow T \geq \log n! \geq \log(\frac{n}{e})^n = n \log n - n \log e = n \log n - O(n)$$

einfache Aproximation der Fakultät: $(\frac{n}{e})^n \leq n! \leq n^n$
Beweis für den **linken Teil:**

$$\ln n! = \sum_{2 \leq i \leq n} \ln i \geq \int_1^n \ln x \, dx = [x(\ln x - 1)]_{x=1}^{x=n} \geq n(\ln n - 1)$$

$$n! \geq e^{n(\ln n - 1)} = \frac{e^{n \ln n}}{e^n} = \frac{n^n}{e^n} = (\frac{n}{e})^n$$

Randomisierung, Mittlere Auführungszeit

Satz: immer noch $n \log n - O(n)$ Vergleiche.
Beweis: nicht hier.

Quicksort - erster Versuch

Idee: Teile-und-Hersche aber vergleichen mit mergsort ändersrum Leiste Arbeit **vor** rekursivem Aufruf

Quicksort - Analyse im schlechtesten Fall

Annahme: Pivot ist immer Minimum (oder Maximum) der Eingaben

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ \Theta(n) & \text{if } n \geq 2 \end{cases}$$

$\Rightarrow T(n) = \Theta(n + (n-1) + \cdots + 1) = \Theta(n^2)$

Schlechteser Fall: Beispiel

Quicksort - Analyse im besten Fall

6 Prioritätslisten

6.1 ...

...

7 Sortierte Listen

7.1 ...

...

8 Graphenreprarsentation

8.1 ...

...

9 Graphtravesierung

9.1 ...

...

10 Küreste Wege

10.1 . . .

. . .

11 Minimale Spannbäume

11.1 ...

...

12 Optimierung

12.1 ...

...