Script generated by TTT

Title: Seidl: GAD (05.05.2015)

Date: Tue May 05 13:53:47 CEST 2015

Duration: 139:36 min

Pages: 62

Zufallsvariable

Beispiel

Wir ziehen aus einem Poker-Kartenspiel mit 52 Karten (13 von jeder Farbe) eine Karte.

Wir bekommen bzw. bezahlen einen bestimmten Betrag, je nachdem welche Farbe die Karte hat, z.B. 4 Euro für Herz, 7 Euro für Karo, −5 Euro für Kreuz und −3 Euro für Pik.

Wenn wir ein As ziehen, bekommen wir zusätzlich 1 Euro.

$\Omega = \{\heartsuit A, \heartsuit K, \ldots, \heartsuit 2, \diamond A, \diamond K, \ldots, \diamond 2, \clubsuit A, \clubsuit K, \ldots, \clubsuit 2, \spadesuit A, \spadesuit K, \ldots, \spadesuit 2\}$.

X sei der Geldbetrag den wir bekommen bzw. bezahlen.

$W_X = \{-5, -4, -3, -2, 4, 5, 7, 8\}$

$\Pr[X = -3] = \Pr[\spadesuit K] + \ldots + \Pr[\spadesuit 2] = 12/52 = 3/13$

Zufallsvariable

Definition

Für einen Wahrscheinlichkeitsraum mit Ergebnismenge Ω nennt man eine Abbildung $X : \Omega \mapsto \mathbb{R}$ (numerische) Zufallsvariable.

Eine Zufallsvariable über einer endlichen oder abzählbar unendlichen Ergebnismenge heißt diskret.

Der Wertebereich diskreter Zufallsvariablen

$$W_X := X(\Omega) = \{x \in \mathbb{R} \mid \exists \omega \in \Omega \text{ mit } X(\omega) = x\}$$

ist ebenfalls endlich bzw. abzählbar unendlich.

Schreibweise: $\Pr[X = x] := \Pr\left[X^{-1}(x)\right] = \sum_{\omega \in \Omega \mid X(\omega) = x} \Pr[\omega]$

Zufallsvariable

Beispiel

Wir ziehen aus einem Poker-Kartenspiel mit 52 Karten (13 von jeder Farbe) eine Karte.

Wir bekommen bzw. bezahlen einen bestimmten Betrag, je nachdem welche Farbe die Karte hat, z.B. 4 Euro für Herz, 7 Euro für Karo, −5 Euro für Kreuz und −3 Euro für Pik.

Wenn wir ein As ziehen, bekommen wir zusätzlich 1 Euro.

Zufallsvariable

Beispiel

Wir ziehen aus einem Poker-Kartenspiel mit 52 Karten (13 von jeder Farbe) eine Karte.

Wir bekommen bzw. bezahlen einen bestimmten Betrag, je nachdem welche Farbe die Karte hat, z.B. 4 Euro für Herz, 7 Euro für Karo, −5 Euro für Kreuz und −3 Euro für Pik.

Wenn wir ein As ziehen, bekommen wir zusätzlich 1 Euro.

$\Omega = \{\heartsuit A, \heartsuit K, \ldots, \heartsuit 2, \diamond A, \diamond K, \ldots, \diamond 2, \clubsuit A, \clubsuit K, \ldots, \clubsuit 2, \spadesuit A, \spadesuit K, \ldots, \spadesuit 2\}$.

X sei der Geldbetrag den wir bekommen bzw. bezahlen.

Effizienz | Erwartete Laufzeit

Zufallsvariable

Beispiel

Wir ziehen aus einem Poker-Kartenspiel mit 52 Karten (13 von jeder Farbe) eine Karte.

Wir bekommen bzw. bezahlen einen bestimmten Betrag, je nachdem welche Farbe die Karte hat, z.B. 4 Euro für Herz, 7 Euro für Karo, −5 Euro für Kreuz und −3 Euro für Pik.

Wenn wir ein As ziehen, bekommen wir zusätzlich 1 Euro.

$\Omega = \{\heartsuit A, \heartsuit K, \ldots, \heartsuit 2, \diamond A, \diamond K, \ldots, \diamond 2, \clubsuit A, \clubsuit K, \ldots, \clubsuit 2, \spadesuit A, \spadesuit K, \ldots, \spadesuit 2\}$.

X sei der Geldbetrag den wir bekommen bzw. bezahlen.

$W_X = \{-5, -4, -3, -2, 4, 5, 7, 8\}$

$\Pr[X = -3] = \Pr[\spadesuit K] + \ldots + \Pr[\spadesuit 2] = 12/52 = 3/13$

H. Seidl (TUM) GAD SS'15 100

Effizienz | Erwartete Laufzeit

Zufallsvariable

Beispiel

Wir ziehen aus einem Poker-Kartenspiel mit 52 Karten (13 von jeder Farbe) eine Karte.

Wir bekommen bzw. bezahlen einen bestimmten Betrag, je nachdem welche Farbe die Karte hat, z.B. 4 Euro für Herz, 7 Euro für Karo, −5 Euro für Kreuz und −3 Euro für Pik.

Wenn wir ein As ziehen, bekommen wir zusätzlich 1 Euro.

$\Omega = \{\heartsuit A, \heartsuit K, \ldots, \heartsuit 2, \diamond A, \diamond K, \ldots, \diamond 2, \clubsuit A, \clubsuit K, \ldots, \clubsuit 2, \spadesuit A, \spadesuit K, \ldots, \spadesuit 2\}$.

X sei der Geldbetrag den wir bekommen bzw. bezahlen.

$W_X = \{-5, -4, -3, -2, 4, 5, 7, 8\}$

$\Pr[X = -3] = \Pr[\spadesuit K] + \ldots + \Pr[\spadesuit 2] = 12/52 = 3/13$

H. Seidl (TUM) GAD SS'15 100

Effizienz | Erwartete Laufzeit

Zufallsvariable

Beispiel

Wir ziehen aus einem Poker-Kartenspiel mit 52 Karten (13 von jeder Farbe) eine Karte.

Wir bekommen bzw. bezahlen einen bestimmten Betrag, je nachdem welche Farbe die Karte hat, z.B. 4 Euro für Herz, 7 Euro für Karo, −5 Euro für Kreuz und −3 Euro für Pik.

Wenn wir ein As ziehen, bekommen wir zusätzlich 1 Euro.

$\Omega = \{\heartsuit A, \heartsuit K, \ldots, \heartsuit 2, \diamond A, \diamond K, \ldots, \diamond 2, \clubsuit A, \clubsuit K, \ldots, \clubsuit 2, \spadesuit A, \spadesuit K, \ldots, \spadesuit 2\}$.

X sei der Geldbetrag den wir bekommen bzw. bezahlen.

$W_X = \{-5, -4, -3, -2, 4, 5, 7, 8\}$

$\Pr[X = -3] = \Pr[\spadesuit K] + \ldots + \Pr[\spadesuit 2] = 12/52 = 3/13$

H. Seidl (TUM) GAD SS'15 100

Erwartungswert

Definition

Für eine diskrete Zufallsvariable X ist der Erwartungswert definiert als

$$\mathbb{E}[X] := \sum_{x \in W_X} x \cdot \Pr[X = x] = \sum_{\omega \in \Omega} X(\omega) \cdot \Pr[\omega]$$

sofern $\sum_{x \in W_X} |x| \cdot \Pr[X = x]$ konvergiert (absolute Konvergenz).

Erwartungswert

Definition

Für eine diskrete Zufallsvariable X ist der Erwartungswert definiert als

$$\mathbb{E}[X] := \sum_{x \in W_X} x \cdot \Pr[X = x] = \sum_{\omega \in \Omega} X(\omega) \cdot \Pr[\omega]$$

sofern $\sum_{x \in W_X} |x| \cdot \Pr[X = x]$ konvergiert (absolute Konvergenz).

Bei endlicher Ereignismenge und gleichwahrscheinlichen Ereignissen entspricht der Erwartungswert dem Durchschnitt:

$$\mathbb{E}[X] = \sum_{x \in W_X} x \cdot \Pr[X = x] = \sum_{\omega \in \Omega} X(\omega) \cdot \frac{1}{|\Omega|} = \frac{1}{|\Omega|} \sum_{\omega \in \Omega} X(\omega)$$

Erwartungswert

Beispiel

(Beispiel wie zuvor)

$$\begin{aligned}\mathbb{E}[X] &= 4 \cdot \frac{12}{52} + 5 \cdot \frac{1}{52} + 7 \cdot \frac{12}{52} + 8 \cdot \frac{1}{52} \\ &\quad + (-5) \cdot \frac{12}{52} + (-4) \cdot \frac{1}{52} + (-3) \cdot \frac{12}{52} + (-2) \cdot \frac{1}{52} = \boxed{\frac{43}{52}}\end{aligned}$$

Wir bekommen also im Erwartungswert $\frac{43}{52}$ Euro pro gezogener Karte.

Grundlagen zu diskreter Wahrscheinlichkeitstheorie findet man z.B. in folgendem Buch:

Th. Schickinger, A. Steger
Diskrete Strukturen – Band 2
(Wahrscheinlichkeitstheorie und Statistik)
Springer-Verlag, 2001.

Erwartungswert

Beispiel

Münze werfen, bis sie zum ersten Mal Kopf zeigt

Zufallsvariable k: Anzahl der Versuche

k ungerade: Spieler bezahlt etwas an die Bank
k gerade: Spieler bekommt etwas von der Bank

Zufallsvariable X: Gewinnbetrag der Bank

Erwartungswert

Beispiel

Münze werfen, bis sie zum ersten Mal Kopf zeigt

Zufallsvariable k: Anzahl der Versuche

k ungerade: Spieler bezahlt etwas an die Bank
k gerade: Spieler bekommt etwas von der Bank

Zufallsvariable X: Gewinnbetrag der Bank

Variante 1: Spieler bezahlt / bekommt k Euro
$\mathbb{E}[X]$ existiert (absolute Konvergenz)

Variante 2: Spieler bezahlt / bekommt 2^k Euro
$\mathbb{E}[X]$ existiert nicht (keine Konvergenz)

Variante 3: Spieler bezahlt / bekommt $\frac{2^k}{k}$ Euro
$\mathbb{E}[X]$ existiert nicht (Konvergenz, aber keine absolute)

Erwartungswert zusammengesetzter Zufallsvariablen

Satz (Linearität des Erwartungswerts)

Für Zufallsvariablen $X_1, \ldots, X_n$ und

$$X := a_1 X_1 + \ldots + a_n X_n$$

mit $a_1, \ldots, a_n \in \mathbb{R}$ gilt

$$\mathbb{E}[X] = a_1 \mathbb{E}[X_1] + \ldots + a_n \mathbb{E}[X_n].$$

Interessant ist für uns vor allem der einfache Fall:

$$X := X_1 + \ldots + X_n$$

mit

$$\mathbb{E}[X] = \mathbb{E}[X_1] + \ldots + \mathbb{E}[X_n].$$

Effizienz | Erwartete Laufzeit

Erwartungswert

Beispiel

Münze werfen, bis sie zum ersten Mal Kopf zeigt

Zufallsvariable k: Anzahl der Versuche

k ungerade: Spieler bezahlt etwas an die Bank
k gerade: Spieler bekommt etwas von der Bank

Zufallsvariable X: Gewinnbetrag der Bank

Variante 1: Spieler bezahlt / bekommt k Euro
$\mathbb{E}[X]$ existiert (absolute Konvergenz)

Variante 2: Spieler bezahlt / bekommt 2^k Euro
$\mathbb{E}[X]$ existiert nicht (keine Konvergenz)

Variante 3: Spieler bezahlt / bekommt $\frac{2^k}{k}$ Euro
$\mathbb{E}[X]$ existiert nicht (Konvergenz, aber keine absolute)

H. Seidl (TUM) GAD SS'15 103

Effizienz | Erwartete Laufzeit

Erwartungswert zusammengesetzter Zufallsvariablen

Satz (Linearität des Erwartungswerts)

Für Zufallsvariablen $X_1, \ldots, X_n$ und

$$X := a_1 X_1 + \ldots + a_n X_n$$

mit $a_1, \ldots, a_n \in \mathbb{R}$ gilt

$$\mathbb{E}[X] = a_1 \mathbb{E}[X_1] + \ldots + a_n \mathbb{E}[X_n].$$

Interessant ist für uns vor allem der einfache Fall:

$$X := X_1 + \ldots + X_n$$

mit

$$\mathbb{E}[X] = \mathbb{E}[X_1] + \ldots + \mathbb{E}[X_n].$$

H. Seidl (TUM) GAD SS'15 104

Erwartungswert

Beispiel

Münze werfen, bis sie zum ersten Mal Kopf zeigt

Zufallsvariable k: Anzahl der Versuche

k ungerade: Spieler bezahlt etwas an die Bank
k gerade: Spieler bekommt etwas von der Bank

Zufallsvariable X: Gewinnbetrag der Bank

Variante 1: Spieler bezahlt / bekommt k Euro
$\mathbb{E}[X]$ existiert (absolute Konvergenz)

Variante 2: Spieler bezahlt / bekommt 2^k Euro
$\mathbb{E}[X]$ existiert nicht (keine Konvergenz)

Variante 3: Spieler bezahlt / bekommt $\frac{2^k}{k}$ Euro
$\mathbb{E}[X]$ existiert nicht (Konvergenz, aber keine absolute)

Erwartungswert zusammengesetzter Zufallsvariablen

Satz (Linearität des Erwartungswerts)

Für Zufallsvariablen $X_1, \ldots, X_n$ und

$$X := a_1 X_1 + \ldots + a_n X_n$$

mit $a_1, \ldots, a_n \in \mathbb{R}$ gilt

$$\mathbb{E}[X] = a_1 \mathbb{E}[X_1] + \ldots + a_n \mathbb{E}[X_n].$$

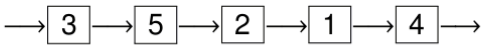
Interessant ist für uns vor allem der einfache Fall:

$$X := X_1 + \ldots + X_n$$

mit

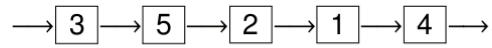
$$\mathbb{E}[X] = \mathbb{E}[X_1] + \ldots + \mathbb{E}[X_n].$$

Beispiel: Suche in statischer Liste



- gegeben: Liste mit Elementen $1, \ldots, m$
- search(i): lineare Suche nach Element i ab Listenanfang
  - s_i Position von Element i in der Liste ($1 \hat{=}$ Anfang)
  - p_i Wahrscheinlichkeit für Zugriff auf Element i

Beispiel: Suche in statischer Liste



- gegeben: Liste mit Elementen $1, \ldots, m$
- search(i): lineare Suche nach Element i ab Listenanfang
  - s_i Position von Element i in der Liste ($1 \hat{=}$ Anfang)
  - p_i Wahrscheinlichkeit für Zugriff auf Element i

Erwartete Laufzeit der Operation search(i) mit zufälligem i:

$$\mathbb{E}[T(\text{search}(i))] = O\left(\sum_i p_i s_i\right)$$

Beispiel: Suche in statischer Liste

- gegeben: Liste mit Elementen $1, \ldots, m$
- search(i): lineare Suche nach Element i ab Listenanfang
  - s_i Position von Element i in der Liste (1≙Anfang)
  - p_i Wahrscheinlichkeit für Zugriff auf Element i

Erwartete Laufzeit der Operation search(i) mit zufälligem i:

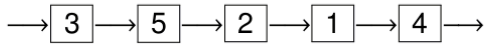
$$\mathbb{E}[T(\text{search}(i))] = O\left(\sum_i p_i s_i\right)$$

Erwartete Laufzeit $t(n)$ für n Zugriffe bei statischer Liste:

$$t(n) = \mathbb{E}[T(n \times \text{search}(i))] = n \cdot \mathbb{E}[T(\text{search}(i))] = O\left(n \sum_i p_i s_i\right)$$

Effizienz | Erwartete Laufzeit

Beispiel: Suche in statischer Liste



- gegeben: Liste mit Elementen $1, \ldots, m$
- search(i): lineare Suche nach Element i ab Listenanfang
  - s_i Position von Element i in der Liste (1≙Anfang)
  - p_i Wahrscheinlichkeit für Zugriff auf Element i

Erwartete Laufzeit der Operation search(i) mit zufälligem i:

$$\mathbb{E}[T(\text{search}(i))] = O\left(\sum_i p_i s_i\right)$$

Erwartete Laufzeit $t(n)$ für n Zugriffe bei statischer Liste:

$$t(n) = \mathbb{E}[T(n \times \text{search}(i))] = n \cdot \mathbb{E}[T(\text{search}(i))] = O\left(n \sum_i p_i s_i\right)$$

H. Seidl (TUM) GAD SS'15 105

Effizienz | Erwartete Laufzeit

Beispiel: Suche in statischer Liste

Optimale Anordnung?

⇒ wenn für alle Elemente i, j mit $p_i > p_j$ gilt, dass $s_i < s_j$, d.h. die Elemente nach Zugriffswahrscheinlichkeit sortiert sind

o.B.d.A. seien die Indizes so, dass $p_1 \geq p_2 \geq \ldots \geq p_m$

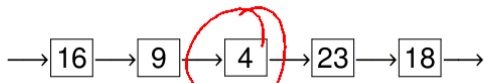
- Optimale Anordnung: $s_i = i$
- Optimale erwartete Laufzeit: $\text{opt} = \sum_i p_i \cdot i$

Einfach: wenn die Zugriffswahrscheinlichkeiten bekannt sind
⇒ optimale erwartete Laufzeit durch absteigende Sortierung nach p_i

Problem: was wenn die Wahrscheinlichkeiten p_i unbekannt sind?

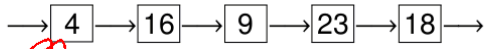
Effizienz | Erwartete Laufzeit

Beispiel: Suche in selbstorganisierender Liste



Move-to-Front Rule:
Verschiebe nach jeder erfolgreichen Suche das gefundene Element an den Listenanfang

Bsp.: Ausführung von search(4) ergibt



Beispiel: Suche in selbstorganisierender Liste

Erwartete Laufzeit $t(n)$ bei dynamischer Liste:

$$\mathbb{E}[T(\text{search}(i))] = O\left(\sum_i p_i \cdot \mathbb{E}[s_i]\right)$$

Satz

Ab dem Zeitpunkt, wo auf jedes Element mindestens einmal zugegriffen wurde, ist die erwartete Laufzeit der search-Operation unter Verwendung der Move-to-Front Rule höchstens $2 \cdot \text{opt}$.

Beispiel: Suche in selbstorganisierender Liste

Beweis.

Betrachte zwei feste Elemente i und j

t_0 Zeitpunkt der letzten Suchoperation auf i oder j

- bedingte Wahrscheinlichkeit: $\Pr[A \mid B] = \dfrac{\Pr[A \wedge B]}{\Pr[B]}$
- $\Pr[C \mid (C \vee D)] = \dfrac{\Pr[C \wedge (C \vee D)]}{\Pr[C \vee D]} = \dfrac{\Pr[C]}{\Pr[C \vee D]}$
- $\Pr[\text{search}(j) \text{ bei } t_0 \mid \text{search}(i \vee j) \text{ bei } t_0] = \dfrac{p_j}{p_i + p_j}$
- mit Wsk. $\dfrac{p_i}{p_i + p_j}$ steht i vor j und mit Wsk. $\dfrac{p_j}{p_i + p_j}$ steht j vor i

Beispiel: Suche in selbstorganisierender Liste

Beweis.

Betrachte nun nur ein festes Element i

- Definiere Zufallsvariablen $X_j \in \{0, 1\}$ für $j \neq i$:

$$X_j = 1 \quad \Leftrightarrow \quad j \text{ vor } i \text{ in der Liste}$$

- Erwartungswert:

$$\begin{aligned} \mathbb{E}[X_j] &= \cancel{0 \cdot \Pr[X_j = 0]} + \cancel{1 \cdot} \Pr[X_j = 1] \\ &= \Pr[\text{letzte Suche nach } i \,/\, j \text{ war nach } j] \\ &= \frac{p_j}{p_i + p_j} \end{aligned}$$

Beispiel: Suche in selbstorganisierender Liste

Beweis.

- Listenposition von Element i: $1 + \sum_{j \neq i} X_j$
- Erwartungswert der Listenposition von Element i:

$$\begin{aligned} \mathbb{E}[s_i] &= \mathbb{E}\left[1 + \sum_{j \neq i} X_j\right] \\ &= 1 + \mathbb{E}\left[\sum_{j \neq i} X_j\right] = 1 + \sum_{j \neq i} \mathbb{E}\left[X_j\right] \\ \mathbb{E}[s_{i,\text{MTF}}] &= 1 + \sum_{j \neq i} \frac{p_j}{p_i + p_j} \end{aligned}$$

Beispiel: Suche in selbstorganisierender Liste

Beweis.

Erwartete Laufzeit der search-Operation:

$$\begin{aligned}
\mathbb{E}[T_{\text{MTF}}] &= \sum_i p_i \left(1 + \sum_{j \neq i} \frac{p_j}{p_i + p_j}\right) \\
&= \sum_i \left(p_i + \sum_{j \neq i} \frac{p_i p_j}{p_i + p_j}\right) = \sum_i \left(p_i + 2 \sum_{j<i} \frac{p_i p_j}{p_i + p_j}\right) \\
&= \sum_i p_i \left(1 + 2 \sum_{j<i} \frac{p_j}{p_i + p_j}\right) \leq \sum_i p_i \left(1 + 2 \sum_{j<i} 1\right) \\
&\leq \sum_i p_i \cdot (2i - 1) < \sum_i p_i \cdot 2i = 2 \cdot \text{opt}
\end{aligned}$$

$2(i-1)+1$

□

Übersicht

Sequenzen

Sequenz: lineare Struktur

$$s = \langle e_0, \ldots, e_{n-1} \rangle$$

(Gegensatz: verzweigte Struktur in Graphen, fehlende Struktur in Hashtab.)

Klassische Repräsentation:

- (Statisches) Feld / Array:
  direkter Zugriff über $s[i]$
  - Vorteil: Zugriff über Index, homogen im Speicher
  - Nachteil: dynamische Größenänderung schwierig
- Liste:
  indirekter Zugriff über Nachfolger / Vorgänger
  - Vorteil: Einfügen / Löschen von Teilsequenzen
  - Nachteil: kein Zugriff per Index, Elemente über Speicher verteilt

Sequenzen

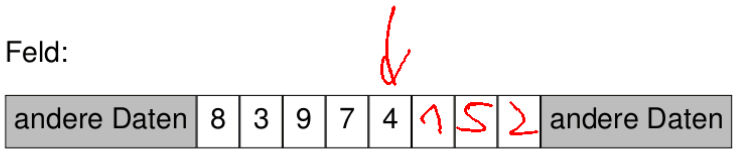
Operationen:

- $\langle e_0, \ldots, e_{n-1} \rangle[i]$ liefert Referenz auf e_i
- $\langle e_0, \ldots, e_{n-1} \rangle.\text{get}(i) = e_i$
- $\langle e_0, \ldots, e_{i-1}, e_i, \ldots, e_{n-1} \rangle.\text{set}(i, e) = \langle e_0, \ldots, e_{i-1}, e, \ldots, e_{n-1} \rangle$
- $\langle e_0, \ldots, e_{n-1} \rangle.\text{pushBack}(e) = \langle e_0, \ldots, e_{n-1}, e \rangle$
- $\langle e_0, \ldots, e_{n-1} \rangle.\text{popBack}() = \langle e_0, \ldots, e_{n-2} \rangle$
- $\langle e_0, \ldots, e_{n-1} \rangle.\text{size}() = n$

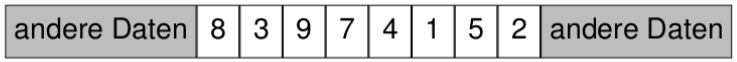
Sequenz als Feld

Problem: beschränkter Speicher

- Feld:



- pushBack(1), pushBack(5), pushBack(2):



- pushBack(6): voll!

Sequenz als Feld

Problem:

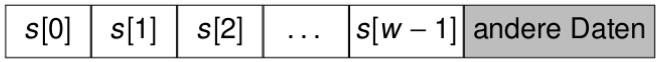
- Beim Anlegen des Felds ist nicht bekannt, wieviele Elemente es enthalten wird
- Nur Anlegen von statischen Feldern möglich (s = new ElementTyp[w])

Lösung: Datenstruktur für dynamisches Feld

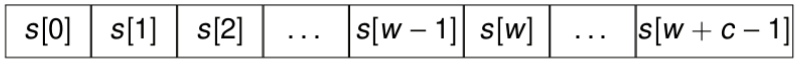
Dynamisches Feld

Erste Idee:

- Immer dann, wenn Feld s nicht mehr ausreicht: generiere neues Feld der Größe $w + c$ für ein festes c

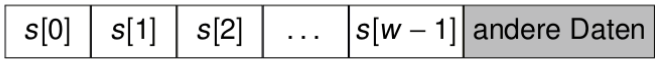


⇓ Kopieren in neues größeres Feld

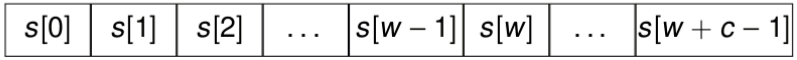


Dynamisches Feld

Zeitaufwand für Erweiterung: $\Theta(w)$



⇓ Kopieren in neues größeres Feld



Zeitaufwand für n pushBack Operationen:

- Aufwand von $\Theta(w)$ nach jeweils c Operationen (wobei w immer größer wird)
- Gesamtaufwand:

$$\Theta\left(\sum_{i=1}^{n/c} c \cdot i\right) = \Theta\left(n^2\right)$$

Dynamisches Feld

Bessere Idee:

- Immer dann, wenn Feld s nicht mehr ausreicht:
  generiere neues Feld der doppelten Größe $2w$

$s[0]$	$s[1]$	$\ldots$	$s[w-1]$	andere Daten

$\Downarrow$ Kopieren in neues größeres Feld

$s[0]$	$s[1]$	$\ldots$	$s[w-1]$	$s[w]$	$\ldots$	$s[2w-1]$

- Immer dann, wenn Feld s zu groß ist ($n \leq w/4$):
  generiere neues Feld der halben Größe $w/2$

Dynamisches Feld

Implementierung

Klasse **UArray** mit den Methoden:

- ElementTyp **get**(int i)
- **set**(int i, ElementTyp e)
- int **size**()
- void **pushBack**(ElementTyp e)
- void **popBack**()
- void **reallocate**(int new_w)

- ElementTyp& **[**int i**]**
  auch möglich, aber Referenz nur bis zur nächsten Größenänderung des Felds gültig

Dynamisches Feld

Implementierung

Klasse **UArray** mit den Elementen:

- $\beta = 2$ // Wachstumsfaktor
- $\alpha = 4$ // max. Speicheroverhead
- **w** $= 1$ // momentane Feldgröße
- **n** $= 0$ // momentane Elementanzahl
- **b** $=$ new ElementTyp$[w]$ // statisches Feld

1024

$b[0]$	$b[1]$	$b[2]$	$\ldots$	$b[w-1]$

Dynamisches Feld

Implementierung

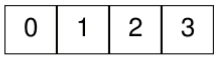
```
ElementTyp get(int i) {
  assert(0≤i<n);
  return b[i];
}

set(int i, ElementTyp e) {
  assert(0≤i<n);
  b[i] = e;
}

int size() {
  return n;
}
```

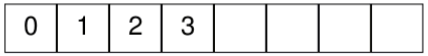
Dynamisches Feld

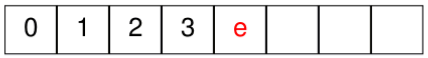
Implementierung

```
void pushBack(ElementTyp e) {
   if (n==w)
      reallocate(β*n);
   b[n]=e;
   n++;
}
```

n=4, w=4







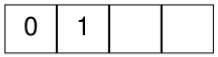
n=5, w=8

Dynamisches Feld

Implementierung

```
void popBack() {
   assert(n>0);
   n−−;
   if (α*n ≤ w ∧ n>0)
      reallocate(β*n);
}
```

n=3, w=8



n=2, w=4

Dynamisches Feld

Implementierung

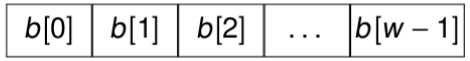
```
void reallocate(int new_w) {
   w = new_w;
   ElementTyp[ ] new_b = new ElementTyp[new_w];
   for (i=0; i<n; i++)
      new_b[i] = b[i];
   b = new_b;
}
```


Dynamisches Feld

Implementierung

Klasse **UArray** mit den Elementen:

- $\beta = 2$ // Wachstumsfaktor
- $\alpha = 4$ // max. Speicheroverhead
- $\mathbf{w} = 1$ // momentane Feldgröße
- $\mathbf{n} = 0$ // momentane Elementanzahl
- $\mathbf{b} =$ new ElementTyp$[w]$ // statisches Feld



Dynamisches Feld

Implementierung

```
void reallocate(int new_w) {
   w = new_w;
   ElementTyp[ ] new_b = new ElementTyp[new_w];
   for (i=0; i<n; i++)
      new_b[i] = b[i];
   b = new_b;
}
```


Dynamisches Feld

Wieviel Zeit kostet eine Folge von n pushBack-/popBack-Operationen?

Erste Idee:

- einzelne Operation kostet $O(n)$
- Schranke kann nicht weiter gesenkt werden, denn reallocate-Aufrufe kosten jeweils $\Theta(n)$

$\Rightarrow$ also Gesamtkosten für n Operationen beschränkt durch $n \cdot O(n) = O(n^2)$

Dynamisches Feld

Wieviel Zeit kostet eine Folge von n pushBack-/popBack-Operationen?

Zweite Idee:

- betrachtete Operationen sollen direkt aufeinander folgen
- zwischen Operationen mit reallocate-Aufruf gibt es immer auch welche ohne

$\Rightarrow$ vielleicht ergibt sich damit gar nicht die n-fache Laufzeit einer Einzeloperation

Dynamisches Feld

Wieviel Zeit kostet eine Folge von n pushBack-/popBack-Operationen?

Zweite Idee:

- betrachtete Operationen sollen direkt aufeinander folgen
- zwischen Operationen mit reallocate-Aufruf gibt es immer auch welche ohne

$\Rightarrow$ vielleicht ergibt sich damit gar nicht die n-fache Laufzeit einer Einzeloperation

Lemma

Betrachte ein anfangs leeres dynamisches Feld s.

Jede Folge $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ von pushBack- und popBack-Operationen auf s kann in Zeit $O(n)$ bearbeitet werden.

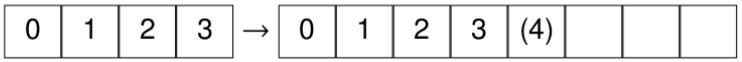
Dynamisches Feld

$\Rightarrow$ nur durchschnittlich konstante Laufzeit pro Operation

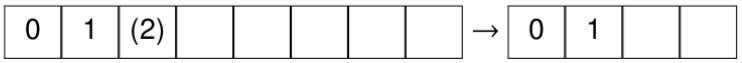
- Kosten teurer Operationen werden mit Kosten billiger Operationen verrechnet.
- Man nennt das dann amortisierte Kosten bzw. amortisierte Analyse.
- In diesem Beispiel hätten wir also eine amortisierte Laufzeit von $O(1)$ für die pushBack- und die popBack-Operation.

Dynamisches Feld: Analyse

- Feldverdopplung:



- Feldhalbierung:



- nächste Verdopplung: nach $\geq n$ pushBack-Operationen
- nächste Halbierung: nach $\geq n/2$ popBack-Operationen

Dynamisches Feld: Analyse

Formale Verrechnung: Zeugenzuordnung

- reallocate kann eine Vergrößerung oder Verkleinerung sein
- reallocate als Vergrößerung auf n Speicherelemente: es werden die $n/2$ vorangegangenen pushBack-Operationen zugeordnet
- reallocate als Verkleinerung auf n Speicherelemente: es werden die n vorangegangenen popBack-Operationen zugeordnet

$\Rightarrow$ kein pushBack / popBack wird mehr als einmal zugeordnet

Dynamisches Feld: Analyse

- Idee: verrechne reallocate-Kosten mit pushBack/popBack-Kosten (ohne reallocate)
  - Kosten für pushBack / popBack: $O(1)$
  - Kosten für reallocate(k*n): $O(n)$
- Konkret:
  - $\Theta(n)$ Zeugen pro reallocate(k∗n)
  - verteile $O(n)$-Aufwand gleichmäßig auf die Zeugen
- Gesamtaufwand: $O(m)$ bei m Operationen

Dynamisches Feld: Analyse

Kontenmethode

- günstige Operationen zahlen Tokens ein
- teure Operationen entnehmen Tokens
- Tokenkonto darf nie negativ werden!

Dynamisches Feld: Analyse

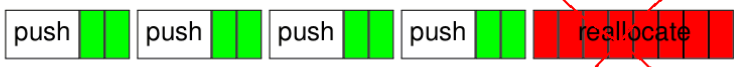
Kontenmethode

- günstige Operationen zahlen Tokens ein
  - pro pushBack 2 Tokens
  - pro popBack 1 Token
- teure Operationen entnehmen Tokens
  - pro reallocate(k∗n) −n Tokens
- Tokenkonto darf nie negativ werden!
  - Nachweis über Zeugenargument

Dynamisches Feld: Analyse

Tokenlaufzeit (Reale Kosten + Ein-/Auszahlungen)

- Ausführung von pushBack / popBack kostet 1 Token
  - Tokenkosten für pushBack: 1+2=3 Tokens
  - Tokenkosten für popBack: 1+1=2 Tokens
- Ausführung von reallocate(k∗n) kostet n Tokens
  - Tokenkosten für reallocate(k∗n): n−n=0 Tokens



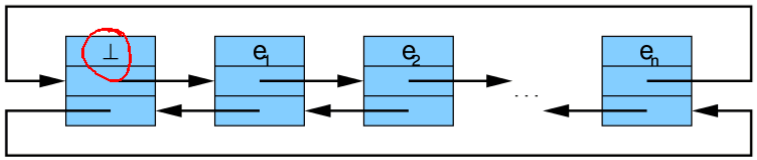
- Gesamtlaufzeit = $\mathcal{O}$(Summe der Tokenlaufzeiten)

Übersicht

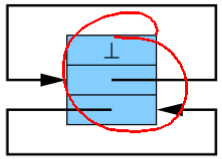
Doppelt verkettete Liste

Einfache Verwaltung:

durch Dummy-Element h ohne Inhalt ($\perp$):

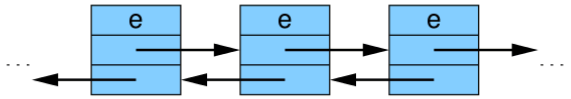


Anfangs:



Doppelt verkettete Liste

```
type Handle: Item<Elem>;

type Item<Elem> {
  Elem e;
  Handle next;
  Handle prev;
}

class List<Elem> {
  Item<Elem> h;      // initialisiert mit ⊥ und Zeigern auf sich selbst
  ... weitere Variablen und Methoden ...
}
```



Invariante:

next.prev == prev.next == this

Doppelt verkettete Liste

Zentrale statische Methode: splice(Handle a, Handle b, Handle t)

- Bedingung:
  - $\langle a, \ldots, b \rangle$ muss Teilsequenz sein (a=b erlaubt)
  - b nicht vor a (also Dummy h nicht zwischen a und b)
  - t nicht in Teilliste $\langle a, \ldots, b \rangle$, aber evt. in anderer Liste
- splice entfernt $\langle a, \ldots, b \rangle$ aus der Sequenz und fügt sie hinter Item t an

Für

$$\langle e_1, \ldots, a', a, \ldots, b, b', \ldots, t, t', \ldots, e_n \rangle$$

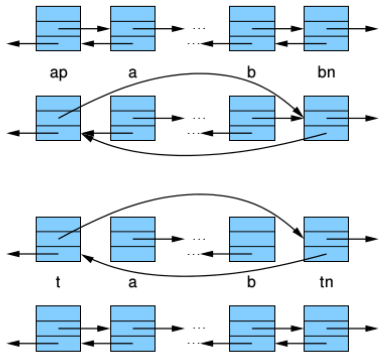
liefert splice(a,b,t)

$$\langle e_1, \ldots, a', b', \ldots, t, a, \ldots, b, t', \ldots, e_n \rangle$$

Doppelt verkettete Liste

Methoden

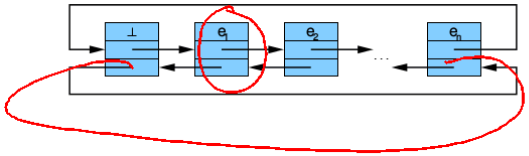
```
(static) splice(Handle a, b, t) {
  // schneide ⟨a,...,b⟩ heraus
  Handle ap = a.prev;
  Handle bn = b.next;
  ap.next = bn;
  bn.prev = ap;

  // füge ⟨a,...,b⟩ hinter t ein
  Handle tn = t.next;
  b.next = tn;
  a.prev = t;
  t.next = a;
  tn.prev = b;
}
```

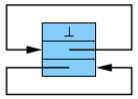


Doppelt verkettete Liste

Methoden

```
Handle head() {
  return h;
}

boolean isEmpty() {
  return (h.next == head());
}

Handle first() {
  return h.next;          // evt. h
}

Handle last() {
  return h.prev;          // evt. h
}
```





Datenstrukturen für Sequenzen | Listen

Doppelt verkettete Liste

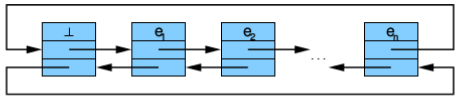
Methoden

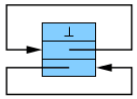
Handle head() {
  return h;
}

boolean isEmpty() {
  return (h.next == head());
}

Handle first() {
  return h.next;   // evt. h
}

Handle last() {
  return h.prev;   // evt. h
}



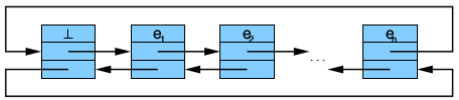


H. Seidl (TUM) | GAD | SS'15 | 142

Datenstrukturen für Sequenzen | Listen

Doppelt verkettete Liste

Methoden



```
(static) moveAfter (Handle b, Handle a) {
  splice(b, b, a);      // schiebe b hinter a
}

moveToFront (Handle b) {
  moveAfter(b, head());      // schiebe b ganz nach vorn
}

moveToBack (Handle b) {
  moveAfter(b, last());      // schiebe b ganz nach hinten
}
```

H. Seidl (TUM) | GAD | SS'15 | 143