Script generated by TTT

Title: Täubig: GAD (03.05.2012)

Date: Thu May 03 12:29:41 CEST 2012

Duration: 43:40 min

Pages: 24

Beispiel: Binärzahl-Inkrementierung

Lemma

$$\sum_{k=1}^{n} \frac{k}{2^k} \leq 2 - \frac{n+2}{2^n}$$

Beweis.

Induktionsanfang: Für $n = 1$ gilt:

$$\sum_{k=1}^{1} \frac{k}{2^k} = \frac{1}{2} \leq 2 - \frac{1+2}{2^1} \qquad \checkmark$$

Induktionsvoraussetzung: Für n gilt:

$$\sum_{k=1}^{n} \frac{k}{2^k} \leq 2 - \frac{n+2}{2^n}$$

Beispiel: Binärzahl-Inkrementierung

Beweis.

Induktionsschritt: $n \to n+1$

$$\begin{aligned}
\sum_{k=1}^{n+1} \frac{k}{2^k} &= \left(\sum_{k=1}^{n} \frac{k}{2^k}\right) + \frac{n+1}{2^{n+1}} \\
&\leq 2 - \frac{n+2}{2^n} + \frac{n+1}{2^{n+1}} \qquad \text{(laut Ind.vor.)} \\
&= 2 - \frac{2n+4}{2^{n+1}} + \frac{n+1}{2^{n+1}} = 2 - \frac{2n+4-n-1}{2^{n+1}} \\
&= 2 - \frac{n+3}{2^{n+1}} \\
&= 2 - \frac{(n+1)+2}{2^{n+1}}
\end{aligned}$$

$\square$

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

Beispiel: Suche in statischer Liste

$$\longrightarrow\boxed{3}\longrightarrow\boxed{5}\longrightarrow\boxed{2}\longrightarrow\boxed{1}\longrightarrow\boxed{4}\longrightarrow$$

- 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:

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

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 | Durchschnittliche Laufzeit

Beispiel: Suche in statischer Liste

$$\longrightarrow\boxed{3}\longrightarrow\boxed{5}\longrightarrow\boxed{2}\longrightarrow\boxed{1}\longrightarrow\boxed{4}\longrightarrow$$

- 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:

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

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. Täubig (TUM) | GAD | SS'12 | 100 / 631

Effizienz | Durchschnittliche Laufzeit

Beispiel: Suche in statischer Liste

$$\longrightarrow\boxed{3}\longrightarrow\boxed{5}\longrightarrow\boxed{2}\longrightarrow\boxed{1}\longrightarrow\boxed{4}\longrightarrow$$

- 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:

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

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. Täubig (TUM) | GAD | SS'12 | 100 / 631

Beispiel: Suche in statischer Liste

Optimale Anordnung?

$\Rightarrow$ 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
$\Rightarrow$ optimale erwartete Laufzeit durch absteigende Sortierung nach p_i

Problem: was wenn die Wahrscheinlichkeiten p_i unbekannt sind?

Beispiel: Suche in selbstorganisierender Liste

$$\longrightarrow \boxed{16} \longrightarrow \boxed{9} \longrightarrow \boxed{4} \longrightarrow \boxed{23} \longrightarrow \boxed{18} \longrightarrow$$

Move-to-Front Rule:

Verschiebe nach jeder erfolgreichen Suche das gefundene Element an den Listenanfang

Bsp.: Ausführung von search(4) ergibt

$$\longrightarrow \boxed{4} \longrightarrow \boxed{16} \longrightarrow \boxed{9} \longrightarrow \boxed{23} \longrightarrow \boxed{18} \longrightarrow$$

Beispiel: Suche in selbstorganisierender Liste

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

$$\mathbb{E}[T(1 \times \text{search}(i))] = O\left(\sum_i p_i \cdot \mathbb{E}[s_i]\right)$$

$$t(n) = \mathbb{E}[T(n \times \text{search}(i))] = O\left(\sum_{j=1}^{n} \sum_i p_i \cdot \mathbb{E}[s_i]\right)$$

Satz

Die erwartete Laufzeit für n search-Operationen bei Verwendung der Move-to-Front Rule ist maximal $2 \cdot \text{opt}$ für genügend große n.

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:

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

Beispiel: Suche in selbstorganisierender Liste

Beweis.

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

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

Beispiel: Suche in selbstorganisierender Liste

Beweis.

Erwartete Laufzeit für Operation t für genügend großes t:

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

□

Effizienz Durchschnittliche Laufzeit

Beispiel: Suche in selbstorganisierender Liste

Beweis.

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

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

H. Täubig (TUM) GAD SS'12 106 / 631

Beispiel: Suche in selbstorganisierender Liste

$$\longrightarrow \boxed{16} \longrightarrow \boxed{9} \longrightarrow \boxed{4} \longrightarrow \boxed{23} \longrightarrow \boxed{18} \longrightarrow$$

Move-to-Front Rule:

Verschiebe nach jeder erfolgreichen Suche das gefundene Element an den Listenanfang

Bsp.: Ausführung von search(4) ergibt

$$\longrightarrow \boxed{4} \longrightarrow \boxed{16} \longrightarrow \boxed{9} \longrightarrow \boxed{23} \longrightarrow \boxed{18} \longrightarrow$$

Übersicht

4 Datenstrukturen für Sequenzen
- Felder
- Listen
- Stacks und Queues
- Diskussion: Sortierte Sequenzen

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
- $\langle e_0, \ldots, e_{n-1} \rangle.\mathrm{get}(i) = e_i$
- $\langle e_0, \ldots, e_{i-1}, e_i, \ldots, e_{n-1} \rangle.\mathrm{set}(i, e) = \langle e_0, \ldots, e_{i-1}, e, \ldots, e_{n-1} \rangle$
- $\langle e_0, \ldots, e_{n-1} \rangle.\mathrm{pushBack}(e) = \langle e_0, \ldots, e_{n-1}, e \rangle$
- $\langle e_0, \ldots, e_{n-1} \rangle.\mathrm{popBack}() = \langle e_0, \ldots, e_{n-2} \rangle$
- $\langle e_0, \ldots, e_{n-1} \rangle.\mathrm{size}() = n$

Übersicht

4 Datenstrukturen für Sequenzen
- Felder
- Listen
- Stacks und Queues
- Diskussion: Sortierte Sequenzen

Sequenz als Feld

Problem:

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

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

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

$\Downarrow$ Kopieren in neues größeres Feld

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

Dynamisches Feld

Zeitaufwand für Erweiterung: $O(w+c) = O(w)$

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

$\Downarrow$ Kopieren in neues größeres Feld

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

Zeitaufwand für n pushBack Operationen:

- Aufwand von $O(w)$ nach jeweils c Operationen
- Gesamtaufwand:

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

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)
- int **size**()
- void **pushBack**(ElementTyp e)
- void **popBack**()
- void **reallocate**(int new_w)