

Teil 1: Einführung

Die Random-Access-Maschine

Die Random-Access-Maschine

Zur Präzisierung der Begriffe **Rechenzeit** und **Speicherbedarf** benötigt man ein präzises Rechenmodell.

Statt Turing-Maschinen, verwendet man häufig ein anderes Rechnermodell.

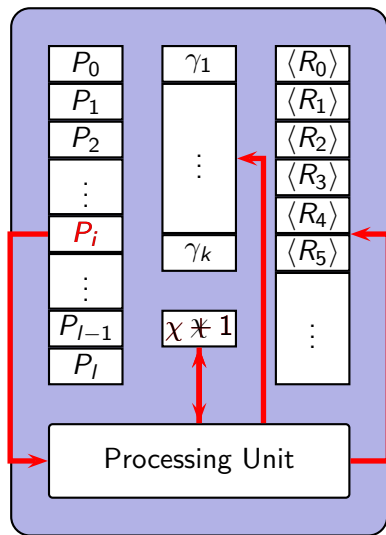
Die Random-Access-Maschine

Eine **Random-Access-Maschine (RAM)** (mit k Registern) besteht aus

- k Registern $\gamma_0, \dots, \gamma_k$.
- Einer unendlichen Zahl von Speicherzellen $R_i, i \in \mathbb{N}$.
- Einem Programm P_0, \dots, P_l , bestehend aus **Befehlen**
- Einem Befehlszähler χ .

Jedes Register und jede Speicherzelle enthält eine natürliche Zahl. Die Inhalte werden mit $\langle X \rangle$ bezeichnet.

Die Funktionsweise der RAM



In jedem Schritt tut die RAM folgendes:

- 1 Lese den Inhalt i von χ .
- 2 Erhöhe χ um eins.
- 3 Lade Befehl P_i .
- 4 Führe den Befehl aus.

Bei der Ausführung des Befehls werden unter Umständen die Inhalte der Register, der Speicherzellen und des Befehlszählers geändert. Nicht jedoch das Programm.

Der Befehlssatz der RAM - Transportbefehle

- $reg \leftarrow i$ Speichere den Wert i im Register reg
- $reg \leftarrow op$ Kopiere den Inhalt von op nach reg
- $op \leftarrow reg$ Kopiere den Inhalt reg nach reg
- $mop \leftarrow reg$ Kopiere den Inhalt reg in die Speicherzelle R_{i+reg}

Legende

- reg Ein beliebiges Register
- i Eine natürliche Zahl
- op Ein Operand der Form reg oder R_i
- mop Ein modifizierter Operand der Form $\langle R_{i+\langle \gamma_j \rangle} \rangle$
- co Ein Vergleichs-Operator aus $\{=, \neq, >, <, \geq, \leq\}$
- ao Ein Arithmetischer Operator aus $\{+, -, \cdot, \div, \text{ mod } \}$

Der Befehlssatz der RAM - Sprungbefehle

goto i Setze den Befehlszähler c auf i

if $reg\ rel\ 0$ **then goto** i Falls der Vergleich stimmt, setze den Befehlszähler auf i

Legende

reg Ein beliebiges Register

i Eine natürliche Zahl

op Ein Operand der Form reg oder R_i

mop Ein modifizierter Operand der Form $\langle R_i + \langle \gamma_j \rangle \rangle$

co Ein Vergleichs-Operator aus $\{=, \neq, >, <, \geq, \leq\}$

ao Ein Arithmetischer Operator aus $\{+, -, \cdot, \div, \text{mod}\}$

Der Befehlssatz der RAM - Arithmetische und Indexbefehle

$reg \leftarrow reg \text{ ao } op$ Arithmetische Operation auf reg und op

$reg \leftarrow reg \pm i$ Addiere/Subtrahiere Konstante zu/von reg

Achtung! Technische Einschränkung!

Bei den Arithmetischen Operationen und den Indexbefehlen, müssen beide Vorkommen von reg das selbe Register bezeichnen!

Legende

reg Ein beliebiges Register

i Eine natürliche Zahl

op Ein Operand der Form reg oder R_i

mop Ein modifizierter Operand der Form $\langle R_{i+\langle \gamma_j \rangle} \rangle$

co Ein Vergleichs-Operator aus $\{=, \neq, >, <, \geq, \leq\}$

ao Ein Arithmetischer Operator aus $\{+, -, \cdot, \div, \text{ mod } \}$

Beispiel: Die Berechnung von 2^n mit zwei Registern

P_0 : $\gamma_1 \leftarrow R_0$

Lade n aus R_0 in den Zähler γ_1

P_1 : $\gamma_2 \leftarrow 1$

Initialisiere γ_2 mit $1 = 2^0$

P_2 : **if** $\gamma_1 = 0$ **then goto** 6

Ende, falls der Zähler 0 ist

P_3 : $\gamma_2 \leftarrow \gamma_2 \cdot 2$

Multipliziere γ_2 mit 2

P_4 : $\gamma_1 \leftarrow \gamma_1 - 1$

Senke den Zähler um 1

P_5 : **goto** 2

Wiederhole

P_6 : $R_1 \leftarrow \gamma_2$

Ergebnis wird nach R_1 geschrieben

Die Mächtigkeit der RAM

Die RAM ist in folgenden Hinsichten unrealistisch:

- Es stehen unendlich viele Speicherzellen zur Verfügung
- Jedes Register und jede Speicherzelle (und jeder Befehl) kann eine beliebige natürliche Zahl speichern (d.h. bereits unendlicher Speicher pro Zelle)

Dennoch spiegelt das Modell sehr gut die Eigenschaften eines „natürlichen“ Computers wieder.

Um die Effizienz von Algorithmen auf diesem Rechnermodell messen zu können benötigen wir Kostenmaße. Im Folgenden führen wir zwei gebräuchliche ein.

Die RAM: Kostenmaße

Einheitskostenmaß (uniforme Kosten)

- Jeder Speicherzugriff kostet eine Einheit.
- Jede Ausführung eines Kernbefehles kostet eine Einheit.

Dieses Maß ist realistisch, wenn die vorkommenden Zahlen jeweils in eine Speicherzelle eines realen Computer passen (Wortgröße)

Logarithmisches Kostenmaß

- Jeder Kernbefehl kostet eine Einheit.
- Die Kosten für alle anderen Argumente sind proportional zur **Länge** der Binärdarstellung der Zahlen, d.h.

$$L(n) = \begin{cases} 1 & \text{falls } n = 0 \\ \lfloor \log n \rfloor + 1 & \text{sonst.} \end{cases}$$

Die RAM: Kostenmaße

Kosten für die Bereitstellung der Operanden

Operand	Einheitskosten	Logarithmische Kosten
i	0	0
reg	0	0
$\langle R_i \rangle$	1	$L(i)$
$\langle R_{i+\gamma_j} \rangle$	1	$L(i) + L(\langle \gamma_j \rangle)$

Kosten für die Kernbefehle

Befehl	Einheitskosten	Logarithmische Kosten
Transportbefehl	1	$1 + L(m)$
Sprungbefehl	1	$1 + L(k)$
Arithmetischer Befehl	1	$1 + L(m_1) + L(m_2)$
Indexbefehl	1	$1 + L(\langle \gamma_j \rangle) + L(i)$

Beispiel: Die Berechnung von 2^n

	Uniform	Log.	Ausführungen
0: $\gamma_1 \leftarrow R_0$	2	$1 + L(0) + L(\langle R_0 \rangle)$	1
1: $\gamma_2 \leftarrow 1$	1	$1 + L(1)$	1
2: if $\gamma_1 = 0$ then goto 6	1	$1 + L(6)$	$n + 1$
3: $\gamma_2 \leftarrow \gamma_2 \cdot 2$	1	$1 + L(\langle \gamma_2 \rangle) + L(2)$	n
4: $\gamma_1 \leftarrow \gamma_1 - 1$	1	$1 + L(\langle \gamma_1 \rangle) + L(1)$	n
5: goto 2	1	$1 + L(2)$	n
6: $R_1 \leftarrow \gamma_2$	2	$1 + L(1) + L(\langle \gamma_2 \rangle)$	1

Einheitskosten: $4n + 6 \in O(n)$

Logarithmische Kosten: $\Theta(n^2)$

Beispiel: Die Berechnung von 2^n

Wie ergeben sich die logarithmischen Kosten?

0:	$\gamma_1 \leftarrow R_0$	$1 + 1 + \Theta(\log(n))$
1:	$\gamma_2 \leftarrow 1$	$1 + 1$
2:	if $\gamma_1 = 0$ then goto 6	$(n + 1)(1 + 3)$
3:	$\gamma_2 \leftarrow \gamma_2 \cdot 2$	$3n + \Theta(n^2)$
4:	$\gamma_1 \leftarrow \gamma_1 - 1$	$2n + \Theta(\sum_{i=1}^n \log(i))$
5:	goto 2	$1 + 2$
6:	$R_1 \leftarrow \gamma_2$	$1 + 1 + \Theta(n)$

Kosten für Zeile 4:

$$\begin{aligned}
 & \underbrace{2 + \Theta(\log(n))}_{1. \text{ Runde}} + \underbrace{2 + \Theta(\log(n-1))}_{2. \text{ Runde}} \cdots + \underbrace{2 + \Theta(\log(1))}_{n\text{-te Runde}} \\
 & = 2n + \Theta\left(\sum_{i=1}^n \log(i)\right)
 \end{aligned}$$

Beispiel: Die Berechnung von 2^n

Wie ergeben sich die logarithmischen Kosten?

0:	$\gamma_1 \leftarrow R_0$	$1 + 1 + \Theta(\log(n))$
1:	$\gamma_2 \leftarrow 1$	$1 + 1$
2:	if $\gamma_1 = 0$ then goto 6	$(n + 1)(1 + 3)$
3:	$\gamma_2 \leftarrow \gamma_2 \cdot 2$	$3n + \Theta(n^2)$
4:	$\gamma_1 \leftarrow \gamma_1 - 1$	$2n + \Theta(\sum_{i=1}^n \log(i))$
5:	goto 2	$1 + 2$
6:	$R_1 \leftarrow \gamma_2$	$1 + 1 + \Theta(n)$

Kosten für Zeile 3:

$$\begin{aligned}
 & \underbrace{3 + \Theta(\log(1))}_{1. \text{ Runde}} + \underbrace{3 + \Theta(\log(2))}_{2. \text{ Runde}} + \dots + \underbrace{3 + \Theta(\log(2^{n-1}))}_{n\text{-te Runde}} \\
 & = 3n + \Theta\left(\sum_{i=0}^{n-1} \log(2^i)\right) = 3n + \Theta\left(\sum_{i=0}^{n-1} i\right) = 3n + \Theta(n^2)
 \end{aligned}$$

Beispiel: Die Berechnung von 2^n

Wie ergeben sich die logarithmischen Kosten?

0:	$\gamma_1 \leftarrow R_0$	$1 + 1 + \Theta(\log(n))$
1:	$\gamma_2 \leftarrow 1$	$1 + 1$
2:	if $\gamma_1 = 0$ then goto 6	$(n + 1)(1 + 3)$
3:	$\gamma_2 \leftarrow \gamma_2 \cdot 2$	$3n + \Theta(n^2)$
4:	$\gamma_1 \leftarrow \gamma_1 - 1$	$2n + \Theta(\sum_{i=1}^n \log(i))$
5:	goto 2	$1 + 2$
6:	$R_1 \leftarrow \gamma_2$	$1 + 1 + \Theta(n)$

Gesamtzeit:

$$T(n) \in 13 + 9n + \Theta(n) + \Theta(n^2) + \Theta(\log n) + \Theta\left(\sum_{i=1}^n \log i\right) = \Theta(n^2)$$

Erweiterungen der RAM

1 Allgemeine Adress-Substitution

- ▶ Jede Speicherzelle kann wie ein Register benutzt werden
- ▶ Auf RAM simulierbar
- ▶ Kostengewinn ist ein konstanter Faktor

2 RASP - Random Access Stored Programm

- ▶ RAM mit Programmspeicher
- ▶ Programmspeicher ist schreibbar
- ▶ Jedes RASP-Programm mit $O(f(n))$ Laufzeit kann durch ein RAM-Programm in $O(f(n))$ simuliert werden.

Kosten der Simulation verschiedener Rechenmodelle

Simulierte Maschine	Simulierende Maschine			
	1-Band TM	k -Band TM	RAM	RASP
1-Band TM	$O(T(n))$	$O(T(n))$	$O(T(n) \log T(n))$	$O(T(n) \log T(n))$
k -Band TM	$O(T^2(n))$	$O(T(n))$	$O(T(n) \log T(n))$	$O(T(n) \log T(n))$
RAM	$O(T^3(n))$	$O(T^2(n))$	$O(T(n))$	$O(T(n))$
RASP	$O(T^3(n))$	$O(T^2(n))$	$O(T(n))$	$O(T(n))$

Höhere Programmiersprachen

Anstelle der relativ unübersichtlichen RAM-Programme verwendet man üblicherweise höhere Programmiersprachen, wie z.B. Pascal, C, C++ oder Java.

Wir halten uns dabei im Allgemeinen an die folgenden Konventionen:

- Verzicht auf Ein- und Ausgabekonventionen und Datendeklarationen
- Wir verwenden alle möglichen Datentypen, d.h.
 - ▶ *Grundtypen*: Integer, Real, Boolean etc.
 - ▶ *Zusammengesetzte Typen*: Felder, Listen, Graphen, Strukturen etc.
- Grundlegenden Anweisungen:
 - ▶ **if A then B else C**
 - ▶ **repeat A until B**
 - ▶ **while A do B**
- Wohldefinierte und präzise, umgangssprachlich formulierte Anweisungen sind erlaubt.
 - ▶ Wichtige Einschränkung: Die Anweisung muss **leicht** in RAM-Code übersetzbar sein.

Datenstrukturen auf RAMs

- **Felder** $A[1..n] = A[1], A[2], \dots, A[n]$
 - ▶ BA = Basisadresse
 - ▶ $\langle R_{BA} \rangle = n$; Länge des Feldes
 - ▶ Speicherung von $A[i]$ in R_{BA+i}
- Schlangen, Keller (Stacks), Listen und Bäume
 - ▶ Die Routinen müssen im RAM-Code vorliegen.