Script generated by TTT

Title: Seidl: GAD (28.04.2015)

Date: Tue Apr 28 13:45:14 CEST 2015

Duration: 147:50 min

Pages: 39

RAM: Speicher

- unbeschränkt viele Speicherzellen (words) $S[0], S[1], S[2], \ldots$, von denen zu jedem Zeitpunkt nur endlich viele benutzt werden
- beliebig große Zellen führen zu unrealistischen Algorithmen

$\Rightarrow$ Jede Speicherzelle darf bei Eingabelänge n eine Zahl mit $O(\log n)$ Bits speichern.
(Für konstant große Zellen würde man einen Faktor $O(\log n)$ bei der Rechenzeit erhalten.)

$\Rightarrow$ gespeicherte Werte stellen polynomiell in Eingabelänge n beschränkte Zahlen dar (sinnvoll für Array-Indizes; bildet auch geschichtliche Entwicklung $4 \to 8 \to 16 \to 32 \to 64$ Bit ab)

Begrenzter Parallelismus:

- sequentielles Maschinenmodell, aber
- Verknüpfung logarithmisch vieler Bits in konstanter Zeit

RAM: Befehle

Annahme:

- Jeder Befehl dauert genau eine Zeiteinheit.
- Laufzeit ist Anzahl ausgeführter Befehle

Befehlssatz:

- Registerzuweisung:
  $R_i := c$ (Konst. an Register), $R_i := R_j$ (Register an Register)
- Speicherzugriff:
  $R_i := S[R_j]$ (lesend), $S[R_j] := R_i$ (schreibend)
- Arithmetische / logische Operationen:
  $R_i := R_j \text{ op } R_k$ (binär: op $\in \{+, -, \cdot, \oplus, /, \%, \wedge, \vee, <, \leq, =, \geq, >\}$),
  $R_i := \text{op } R_j$ (unär: op $\in \{-, \neg\}$)
- Sprünge:
  jump x (zu Adresse x), jumpz x R_i (bedingt, falls $R_i = 0$),
  jumpi R_j (zu Adresse aus R_j)

Das entspricht Assembler-Code von realen Maschinen!

Maschinenmodell

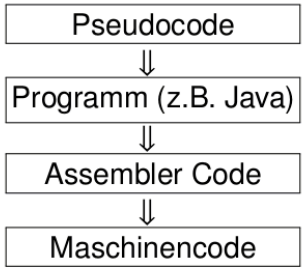
RAM-Modell

- Modell für die ersten Computer
- entspricht eigentlich der Harvard-Architektur (separater Programmspeicher)
- Random Access Stored Program (RASP) Modell entspricht der von Neumann-Architektur und hat große Ähnlichkeit mit üblichen Rechnern

Aber: Speicherhierarchie erfordert ggf. Anpassung des Modells

$\Rightarrow$ Algorithm Engineering, z.B. External-Memory Model

Pseudocode / Maschinencode



- Assembler/Maschinencode schwer überschaubar
- besser: Programmiersprache wie Pascal, C++, Java, ...
- oder: informal als Pseudocode in verständlicher Form

$a := a + bc \quad \Rightarrow \quad R_1 := R_b * R_C; \quad R_a := R_a + R_1$

R_a, R_b, R_c: Register, in denen a, b und c gespeichert sind

if (C) I else J ⇒ eval(C); jumpz sElse R_c; trans(I); jump sEnd; trans(J)

eval(C): Befehle, die die Bedingung C auswerten und das Ergebnis in Register R_c hinterlassen
trans(I), trans(J): übersetzte Befehlsfolge für I und J

sElse, sEnd: Adresse des 1. Befehls in trans(J) bzw. des 1. Befehls nach trans(J)

Laufzeitanalyse / worst case

Berechnung der worst-case-Laufzeit:

- $T(I)$ sei worst-case-Laufzeit für Konstrukt I
- $T(\text{elementare Zuweisung}) = O(1)$
- $T(\text{elementarer Vergleich}) = O(1)$
- $T(\textbf{return } x) = O(1)$
- $T(\textbf{new Typ}(\ldots)) = O(1) + O(T(\text{Konstruktor}))$
- $T(I_1; I_2) = T(I_1) + T(I_2)$
- $T(\textbf{if } (C)\ I_1\ \textbf{else}\ I_2) = O(T(C) + \max\{T(I_1), T(I_2)\})$
- $T(\textbf{for}(i = a; i < b; i++)\ I) = O\left(\sum_{i=a}^{b-1}(1 + T(I))\right)$
- $T(e.m(\ldots)) = O(1) + T(ss)$, wobei ss Rumpf von m

Beispiel: Vorzeichenausgabe

Funktion signum(x)
Eingabe : Zahl $x \in \mathbb{R}$
Ausgabe : -1,0 bzw. 1 entsprechend dem Vorzeichen von x

```
if x < 0 then
  return -1
if x > 0 then
  return 1
return 0
```

Wir wissen:

$T(x < 0) = O(1)$
$T(\textbf{return } -1) = O(1)$
$T(\textbf{if } (C)\ I) = O(T(C) + T(I))$

Also: $T(\textbf{if } (x < 0)\ \textbf{return } -1) = O(1) + O(1) = O(1)$

Beispiel: Vorzeichenausgabe

Funktion signum(x)
Eingabe : Zahl $x \in \mathbb{R}$
Ausgabe : -1,0 bzw. 1 entsprechend dem Vorzeichen von x

if $x < 0$ **then**	
⌊ **return** *-1*	$O(1)$
if $x > 0$ **then**	
⌊ **return** *1*	$O(1)$
return *0*	$O(1)$

$O(1 + 1 + 1) = O(1)$

Beispiel: Minimumsuche

Funktion minimum(A, n)
Eingabe : Zahlenfolge in $A[0], \ldots, A[n-1]$
n: Anzahl der Zahlen
Ausgabe : Minimum der Zahlen

```
min = A[0];                                   O(1)
for (i = 1; i < n; i++) do                    O(Σ_{i=1}^{n-1} (1 + T(l)))
    if A[i] < min then min = A[i];              O(1)
return min                                    O(1)
```

$$O\left(1 + \left(\sum_{i=1}^{n-1} 1\right) + 1\right) = O(n)$$

Beispiel: BubbleSort

Sortieren durch Aufsteigen

Vertausche in jeder Runde in der (verbleibenden) Eingabesequenz (hier vom Ende in Richtung Anfang) jeweils zwei benachbarte Elemente, die nicht in der richtigen Reihenfolge stehen

Beispiel

5	10	19	1	14	3
5	10	19	1	3	14
5	10	1	19	3	14
5	1	10	19	3	14
1	5	10	19	3	14

1	5	10	3	19	14
1	5	3	10	19	14
1	**3**	5	10	19	14
1	3	**5**	10	14	19
1	3	5	**10**	**14**	**19**

Beispiel: Sortieren

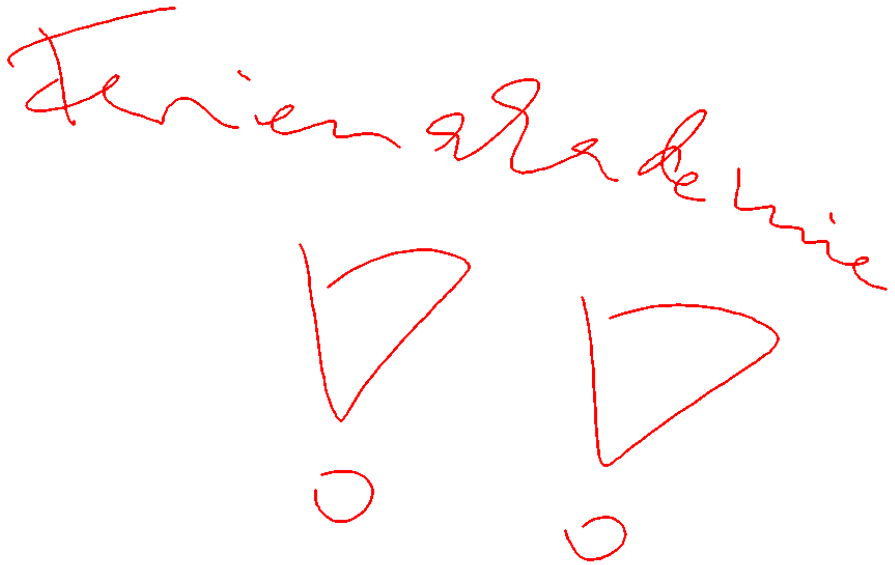
Prozedur BubbleSort(A, n)
Eingabe : n: Anzahl der Zahlen
$A[0], \ldots, A[n-1]$: Zahlenfolge
Ausgabe : Sortierte Zahlenfolge A

```
for (i = 0; i < n − 1; i++) do                O(Σ_{i=0}^{n-2} T(l1))
    for (j = n − 2; j ≥ i; j--) do              O(Σ_{j=i}^{n-2} T(l2))
        if A[j] > A[j + 1] then                   O(1 + T(l3))
            x = A[j];                               O(1)
            A[j] = A[j + 1];                        O(1)
            A[j + 1] = x;                           O(1)
```

$$O\left(\sum_{i=0}^{n-2}\sum_{j=i}^{n-2} 1\right)$$

Beispiel: Sortieren

$$\begin{aligned}
\sum_{i=0}^{n-2}\sum_{j=i}^{n-2} 1 &= \sum_{i=0}^{n-2}(n-i-1) \\
&= \sum_{i=1}^{n-1} i \\
&= \frac{n(n-1)}{2} \\
&= \frac{n^2}{2} - \frac{n}{2} \\
&= O\left(n^2\right)
\end{aligned}$$



Beispiel: Sortieren

$$\begin{aligned}\sum_{i=0}^{n-2}\sum_{j=i}^{n-2} 1 &= \sum_{i=0}^{n-2}(n-i-1)\\ &= \sum_{i=1}^{n-1} i\\ &= \frac{n(n-1)}{2}\\ &= \frac{n^2}{2}-\frac{n}{2}\\ &= O\left(n^2\right)\end{aligned}$$

Beispiel: Binäre Suche

Prozedur BinarySearch(A, n, x)

Eingabe : n: Anzahl der (sortierten) Zahlen
$A[0],\ldots,A[n-1]$: Zahlenfolge
x: gesuchte Zahl

Ausgabe : Index der gesuchten Zahl

$\ell = 0;$	$O(1)$
$r = n-1;$	$O(1)$
while $(\ell \le r)$ **do**	$O(\sum_{i=1}^{k} T(I))$
$\quad m = \lfloor (r+\ell)/2 \rfloor;$	$O(1)$ ↑
$\quad$ **if** $A[m] == x$ **then return** $m;$	$O(1)$
$\quad$ **if** $A[m] < x$ **then** $\ell = m+1;$	$O(1)$
$\quad$ **else** $r = m-1;$	$O(1)$
return -1	$O(1)$

$$O\left(\sum_{i=1}^{k} 1\right) = O(k)$$

Beispiel: Binäre Suche

Aber: Wie groß ist die Anzahl der Schleifendurchläufe k?

Größe des verbliebenen Suchintervalls $(r-\ell+1)$ nach Iteration i:

$$\begin{aligned} s_0 &= n\\ s_{i+1} &\le \lfloor s_i/2 \rfloor\end{aligned}$$

Bei $s_i < 1$ endet der Algorithmus.

$$\Rightarrow \quad k \le \log_2 n$$

Gesamtkomplexität: $O(\log n)$

Beispiel: Bresenham-Algorithmus

Algorithmus Bresenham1: zeichnet einen Kreis

$x = 0;\quad y = R;$	$O(1)$
$\text{plot}(0, R); \text{plot}(R, 0); \text{plot}(0, -R); \text{plot}(-R, 0);$	$O(1)$
$F = \frac{5}{4} - R;$	$O(1)$
while $x < y$ **do**	$O(\sum_{i=1}^{k} T(I))$
if $F < 0$ **then**	
$F = F + 2 * x + 1;$	
else	
$F = F + 2 * x - 2 * y + 2;$	alles $O(1)$
$y = y - 1;$	
$x = x + 1;$	
$\text{plot}(x, y); \text{plot}(-x, y); \text{plot}(-y, x); \text{plot}(-y, -x);$	
$\text{plot}(y, x); \text{plot}(y, -x); \text{plot}(x, -y); \text{plot}(-x, -y);$	

Wie groß ist Anzahl Schleifendurchläufe k? $O\left(\sum_{i=1}^{k} 1\right) = O(k)$

Beispiel: Bresenham-Algorithmus

- Betrachte dazu die Entwicklung der Werte der Funktion

$$\varphi(x, y) = y - x$$

- Anfangswert: $\varphi_0(x, y) = R$
- Monotonie: verringert sich pro Durchlauf um mindestens 1
- Beschränkung: durch die **while**-Bedingung $x < y$ bzw. $0 < y - x$

$\Rightarrow$ maximal R Runden

Effizienz | Laufzeitanalyse

Beispiel: Bresenham-Algorithmus

Algorithmus Bresenham1: zeichnet einen Kreis

$x = 0;\quad y = R;$	$O(1)$
$\text{plot}(0, R); \text{plot}(R, 0); \text{plot}(0, -R); \text{plot}(-R, 0);$	$O(1)$
$F = \frac{5}{4} - R;$	$O(1)$
while $x < y$ **do**	$O(\sum_{i=1}^{k} T(I))$
if $F < 0$ **then**	
$F = F + 2 * x + 1;$	
else	
$F = F + 2 * x - 2 * y + 2;$	alles $O(1)$
$y = y - 1;$	
$x = x + 1;$	
$\text{plot}(x, y); \text{plot}(-x, y); \text{plot}(-y, x); \text{plot}(-y, -x);$	
$\text{plot}(y, x); \text{plot}(y, -x); \text{plot}(x, -y); \text{plot}(-x, -y);$	

Wie groß ist Anzahl Schleifendurchläufe k? $O\left(\sum_{i=1}^{k} 1\right) = O(k)$

H. Seidl (TUM) GAD SS'15 88

Effizienz | Laufzeitanalyse

Beispiel: Bresenham-Algorithmus

- Betrachte dazu die Entwicklung der Werte der Funktion

$$\varphi(x, y) = y - x$$

- Anfangswert: $\varphi_0(x, y) = R$
- Monotonie: verringert sich pro Durchlauf um mindestens 1
- Beschränkung: durch die **while**-Bedingung $x < y$ bzw. $0 < y - x$

$\Rightarrow$ maximal R Runden

H. Seidl (TUM) GAD SS'15 89

Beispiel: Bresenham-Algorithmus

Algorithmus Bresenham1: zeichnet einen Kreis

$x = 0;\quad y = R;$	$O(1)$
$\text{plot}(0, R); \text{plot}(R, 0); \text{plot}(0, -R); \text{plot}(-R, 0);$	$O(1)$
$F = \frac{5}{4} - R;$	$O(1)$
while $x < y$ **do**	$O(\sum_{i=1}^{k} T(I))$
if $F < 0$ **then**	
$F = F + 2 * x + 1;$	
else	
$F = F + 2 * x - 2 * y + 2;$	alles $O(1)$
$y = y - 1;$	
$x = x + 1;$	
$\text{plot}(x, y); \text{plot}(-x, y); \text{plot}(-y, x); \text{plot}(-y, -x);$	
$\text{plot}(y, x); \text{plot}(y, -x); \text{plot}(x, -y); \text{plot}(-x, -y);$	

Wie groß ist Anzahl Schleifendurchläufe k? $O\left(\sum_{i=1}^{k} 1\right) = O(k)$

Beispiel: Fakultätsfunktion

Funktion fakultaet(n)

Eingabe : $n \in \mathbb{N}_+$

Ausgabe : $n!$

if $(n == 1)$ **then**	$O(1)$
return 1	$O(1)$
else	
return $n * fakultaet(n-1)$	$O(1 + \ldots?)$

- $T(n)$: Laufzeit von fakultaet(n)
- $T(1) = O(1)$ $\leq c$
- $T(n) \leq T(n-1) + O(1)$ c

$\Rightarrow T(n) = O(n)$

$T(n) \cdot c \cdot w \in O(n)$

Übersicht

Average Case Complexity

Uniforme Verteilung:
(alle Instanzen gleichwahrscheinlich)

$$t(n) = \frac{1}{|\mathcal{I}_n|} \sum_{i \in \mathcal{I}_n} T(i)$$

Tatsächliche Eingabeverteilung kann in der Praxis aber stark von uniformer Verteilung abweichen.

Dann

$$t(n) = \sum_{i \in \mathcal{I}_n} p_i \cdot T(i)$$

Aber: meist schwierig zu berechnen!

Beispiel: Binärzahl-Inkrementierung

Prozedur increment(A)

Eingabe : Array A mit Binärzahl in $A[0]\dots A[n-1]$,
in $A[n]$ steht eine 0

Ausgabe : inkrementierte Binärzahl in $A[0]\dots A[n]$

```
i = 0;
while (A[i] == 1) do
    A[i] = 0;
    i = i + 1;
A[i] = 1;
```

Durchschnittliche Laufzeit für Zahl mit n Bits?

Binärzahl-Inkrementierung: Analyse

- $\mathcal{I}_n$: Menge der n-Bit-Instanzen
- Für die Hälfte (also $\frac{1}{2}|\mathcal{I}_n|$) der Zahlen $x_{n-1}\dots x_0 \in \mathcal{I}_n$ ist $x_0 = 0$
  $\Rightarrow$ 1 Schleifendurchlauf
- Für die andere Hälfte gilt $x_0 = 1$.
  Bei diesen gilt wieder für die Hälfte (also $\frac{1}{4}|\mathcal{I}_n|$) $x_1 x_0 = 01$
  $\Rightarrow$ 2 Schleifendurchläufe
- Für den Anteil $(\frac{1}{2})^k$ der Zahlen gilt $x_{k-1}x_{k-2}\dots x_0 = 01\dots1$
  $\Rightarrow$ k Schleifendurchläufe

Durchschnittliche Anzahl Schleifendurchläufe:

$$t(n) = \frac{1}{|\mathcal{I}_n|}\sum_{i\in\mathcal{I}_n} T(i) = \frac{1}{|\mathcal{I}_n|}\sum_{k=1}^{n}\frac{|\mathcal{I}_n|}{2^k}\cdot k = \sum_{k=1}^{n}\frac{k}{2^k} \stackrel{?}{=} O(1)$$

Binärzahl-Inkrementierung: Abschätzung

Lemma

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

Binärzahl-Inkrementierung: Abschätzung

Lemma

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

Beweis

Induktionsanfang:

Für $n = 1$ gilt: $\sum_{k=1}^{1}\frac{k}{2^k} = \frac{1}{2} \le 2 - \frac{1+2}{2^1}$

Binärzahl-Inkrementierung: Abschätzung

Lemma

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

Beweis

Induktionsanfang:

Für $n = 1$ gilt: $\sum_{k=1}^{1} \frac{k}{2^k} = \frac{1}{2} \leq 2 - \frac{1+2}{2^1}$ ✓

Induktionsvoraussetzung:

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

Binärzahl-Inkrementierung: Abschätzung

Beweis.

Induktionsschritt: $n \to n+1$

$$\sum_{k=1}^{n+1} \frac{k}{2^k} = \left(\sum_{k=1}^{n} \frac{k}{2^k}\right) + \frac{n+1}{2^{n+1}}$$

Binärzahl-Inkrementierung: Abschätzung

Beweis.

Induktionsschritt: $n \to n+1$

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

Binärzahl-Inkrementierung: Abschätzung

Beweis.

Induktionsschritt: $n \to n+1$

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

□

Binärzahl-Inkrementierung: Abschätzung

Lemma

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

Beweis

Induktionsanfang:

Für $n = 1$ gilt: $\sum_{k=1}^{1} \frac{k}{2^k} = \frac{1}{2} \leq 2 - \frac{1+2}{2^1}$ ✓

Binärzahl-Inkrementierung: Abschätzung

Beweis.

Induktionsschritt: $n \to n+1$

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

□

Zufallsvariable

Definition

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

Effizienz | Erwartete Laufzeit

Zufallsvariable

Definition

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

H. Seidl (TUM) GAD SS'15 98

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.

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.

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.