Algorithmen und Datenstrukturen (ESE)
Entwurf, Analyse und Umsetzung von Algorithmen (IEMS)
WS 2014 / 2015

Vorlesung 4, Donnerstag 13. November 2014
(Mittlere Laufzeit, Assoziative Arrays aka Maps)

Junior-Prof. Dr. Olaf Ronneberger
Image Analysis Lab
Institut für Informatik
Universität Freiburg

[basiert auf der Veranstaltung von Prof. Dr. Hannah Bast, Lehrstuhl für Algorithmen und Datenstrukturen, mit Folien aus „Informatik 2" von Prof. Dr. Matthias Teschner]

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Ihre Erfahrungen mit dem Ü3 (O-Notation)
- Laufzeiten
  - Beste, schlechteste und mittlere Laufzeit
- Assoziative Arrays aka Maps aka = also known as
  - Nutzen anhand eines typischen Beispieles
  - Alternative dazu: Sortieren
  - **Übungsaufgabe:** die weltweit häufigsten Ortsnamen finden ... einmal mit "Maps", und einmal mit Sortieren
- Benutzung von std::sort, und std::map

Erfahrungen mit dem Ü3 (O-Notation)

UNI FREIBURG

- Zusammenfassung / Auszüge (Stand 13.11. 9:30)
  - Ärger mit SVN Server → Abgabe verlängert bis heute Abend
  - Zeitaufwand war bei den meisten weniger als 4 Stunden
  - ziemlich kniffelig, hat mich viel Zeit gekostet die Groß-O-Notation zu verstehen.
  - Bitte wieder mehr programmieren
  - Bitte mehr Beweise, weitere Übungen dazu
  - Tutorat für dieses Blatt?
  - Klausuraufgabe war die einfachste → die müssen sie ja auch unter Klausurbedingungen lösen können
  - Gruppenabgaben?
    - Nein. Falls Latex zu aufwändig ist, einfach handschriftlich und einscannen.
    - Offensichtlich identische Lösungen werden ab jetzt **beide** mit 0 Punkten bewertet.
    - Probieren Sie immer zuerst die Aufgaben alleine zu lösen. Das schult das analytische Denkvermögen.

Feedback von Ihrem/r Tutor/in

UNI FREIBURG

- Machen Sie dazu svn update
  - ... spätestens bevor Sie ein neues Übungsblatt bearbeiten
  - Sie bekommen dann (hoffentlich) eine Datei

    uebungsblatt_XY/feedback-tutor.txt
  - Dort sehen Sie unter anderem
    - warum wo wie viel Punkte abgezogen
    - Verbesserungsvorschläge für Code / Beweise
    - Antworten auf Fragen aus Ihrer erfahrungen.txt
    - Und was Ihrem/r Tutor/in noch alles so einfällt ...

Verschiedene Laufzeiten

- Laufzeit hängt nicht immer ausschließlich von der Größe des Problems ab, sondern auch von der Beschaffenheit der Eingabemenge
- Daraus ergeben sich
  - beste Laufzeit
    beste Laufzeitkomplexität für eine Eingabeinstanz der Größe n
  - schlechteste Laufzeit
    schlechteste Laufzeitkomplexität für eine Instanz der Größe n
  - mittlere oder erwartete Laufzeit
    gemittelte Laufzeitkomplexität für alle Eingabeinstanzen der Größe n

Beispiel 1 (if Abfrage)

- Eingabe: Feld a mit n Elementen $a[i] \in \mathbb{N} \quad 1 \leq a[i] \leq n$
- Ausgabe: Feld a mit n Elementen mit $a[0] \neq 1$

`if (a[0]==1)`	O(1)	O(1)	?
`then`		O(1)	
`  a[0]:=2;`	O(1)		
`else`			
`  for i:=0 to n-1 do`	O(n)	O(n)	
`    a[i]:=2;`	O(1)		

- beste Laufzeit: O(1) + O(1) = O(1)
- schlechteste Laufzeit: O(1) + O(n) = O(n)

Mittlere Laufzeit

- Mittlere Laufzeit wird durch die Mittelung der notwendigen Schritte für alle Eingabeinstanzen der Größe n ermittelt.
- Jedes Element a[i] kann n Werte annehmen.
- n Elemente: $n \cdot n \cdot \ldots \cdot n = n^n$ verschiedene Eingabeinstanzen der Größe n
- a[i]=1 in n^{n-1} Instanzen
- a[i]≠1 in $n^n - n^{n-1} = n^{n-1} \cdot (n-1)$ Instanzen
- Gesamtschritte:
  n^{n-1} Instanzen · 1 Schritt + n^{n-1} · (n-1) Instanzen · n Schritte
  $= n^{n-1} + n^n \cdot (n-1)$
- Mittlere Laufzeit: (Schritte / Instanzen)

$$\frac{n^{n-1} + n^n \cdot (n-1)}{n^n} = \frac{1}{n} + n - 1 = \mathcal{O}(n)$$

Beispiel 2 (Binär-Addition)

- Eingabe: n-stellige Dualzahl a
- Ausgabe: n-stellige Dualzahl a+1
- Zahl der Schritte des Algorithmus ergibt sich aus der Zahl der "0"-"1"- bzw. "1"-"0"-Wechsel

Problemgröße n	Eingabe	Operation	Ausgabe	Zahl der Schritte
10	1011100100	+1	1011100101	1
4	1011	+1	1100	3
8	11111111	+1	00000000	8 (=n)

- beste Laufzeit: 1 Schritt = O(1)
- schlechteste Laufzeit: n Schritte = O(n)

Mittlere Laufzeit

- ermittle die durchschnittliche Schrittzahl für alle Eingabeinstanzen der Größe n

n=1		
Eingabe	Ausgabe	Schritte
0	1	1
1	0	1

Durchschnitt $\frac{1+1}{2} = 1$

n=2		
Eingabe	Ausgabe	Schritte
00	01	1
01	10	2
10	11	1
11	00	2

Durchschnitt $\frac{1+2+1+2}{4} = \frac{3}{2}$

n=3		
Eingabe	Ausgabe	Schritte
000	001	1
001	010	2
010	011	1
011	100	3
100	101	1
101	110	2
110	111	1
111	000	3

Durchschnitt $\frac{7}{4}$

Mittlere Laufzeit $2 - \frac{1}{2^{n-1}} = \mathcal{O}(1)$

Mittlere Laufzeit Fallunterscheidung

- Fallunterscheidung für Instanzen der Größe n

n-stellige Eingabe	Instanzen	Zahl der Schritte
xxx…x0	2^{n-1}	1
xxx…x01	2^{n-2}	2
xxx…x011	2^{n-3}	3
…	…	…
x0111…1	$2 = 2^1$	n-1
0111…1	$1 = 2^0$	n
111…1	1	n

- Mittlere Laufzeit (Schritte / Instanzen):

$$\frac{1 \cdot 2^{n-1} + 2 \cdot 2^{n-2} + \cdots + (n-1) \cdot 2^1 + n \cdot 2^0 + n \cdot 1}{2^{n-1} + 2^{n-2} + \cdots + 2^1 + 2^0 + 1} = \frac{\sum_{i=1}^{n} (i \cdot 2^{n-i}) + n}{\sum_{i=0}^{n-1} (2^i) + 1}$$

Zwischenrechnung

$$\frac{\sum_{i=1}^{n}(i \cdot 2^{n-i}) + n}{\sum_{i=0}^{n-1}(2^i) + 1} = ?$$

$1 \cdot 2^3 + 2 \cdot 2^2 + 3 \cdot 2^1 + 4 \cdot 2^0$

Nenner:

$$\sum_{i=0}^{n-1}(2^i) + 1 \quad = \quad 2^n - 1 + 1 \quad = \quad 2^n$$

geometrische Reihe

intuitiv klar

Zähler:

$$\sum_{i=1}^{n}(i \cdot 2^{n-i}) + n \quad = \quad 2 \cdot \sum_{i=1}^{n}(i \cdot 2^{n-i}) - \sum_{i=1}^{n}(i \cdot 2^{n-i}) + n$$

(a = 2a – a)

$$= 1 \cdot 2^n + 2 \cdot 2^{n-1} + 3 \cdot 2^{n-2} + \cdots + (n-1) \cdot 2^2 + n \cdot 2^1$$
$$- 1 \cdot 2^{n-1} - 2 \cdot 2^{n-2} - \cdots - (n-2) \cdot 2^2 - (n-1) \cdot 2^1 - n \cdot 2^0 + n$$

$$= 1 \cdot 2^n + 1 \cdot 2^{n-1} + 1 \cdot 2^{n-2} + \ldots \quad + \quad 1 \cdot 2^2 \quad + \quad 1 \cdot 2^1 \quad - n + n$$

$$= 2^n + 2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2^1 + 2^0 \quad - 2^0 \quad = \quad 2^{n+1} - 1 \quad - 2^0$$

Ergebnis

- Mittlere Laufzeit (Schritte / Instanzen)

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

$$\lim_{n\to\infty}\left(2 - \frac{1}{2^{n-1}}\right) = 2 \qquad \Rightarrow \mathcal{O}(1)$$

Normale vs. Assoziative Arrays

UNI FREIBURG

- Normales Array
  - Zugriff auf eine (große) Anzahl von gleichartigen Elementen über einen fortlaufenden **Index**

    A_0, A_1, A_2, A_3, A_4, A_5, ...
  - Beispielanfrage: das Element mit Index 3 A[3]
- Assoziatives Array
  - Zugriff auf eine (große) Anzahl von gleichartigen Elementen über einen beliebigen sog. **Schlüssel** (Key)
  - A_{Europa}, $A_{Amerika}$, A_{Asien}, A_{Afrika} , ...
  - Beispielanfrage: das Element mit Schlüssel Amerika A["Amerika"]
  - Die Schlüssel können auch (nicht fortlaufende) Zahlen sein

Und wofür braucht man das?

UNI FREIBURG

- In der Praxis ...
  - ... gibt es kaum ein größeres Programm wo man nicht irgendwo ein assoziatives Array braucht
- Unser Beispiel für diese Vorlesung
  - Eine Liste von Infos zu Ländern (von www.geonames.org)

countryInfo.txt

ISO	ISO3	ISO-Numeric	fips	Country	Capital	Area(in sq km)	Population	Continent	tld	CurrencyCode	CurrencyName
AD	AND	20	AN	Andorra	Andorra la Vella	468	84000	EU	.ad	EUR	Euro
AE	ARE	784	AE	United Arab Emirates	Abu Dhabi	82880	4975593	AS	.ae	AED	Dirham
AF	AFG	4	AF	Afghanistan	Kabul	647500	29121286	AS	.af	AFN	Afghani
AG	ATG	28	AC	Antigua and Barbuda	St. John's	443	86754	NA	.ag	XCD	Dollar
AI	AIA	660	AV	Anguilla	The Valley	102	13254	NA	.ai	XCD	Dollar
AL	ALB	8	AL	Albania	Tirana	28748	2986952	EU	.al	ALL	Lek
AM	ARM	51	AM	Armenia	Yerevan	29800	2968000	AS	.am	AMD	Dram
AO	AGO	24	AO	Angola	Luanda	1246700	13068161	AF	.ao	AOA	Kwanza
AQ	ATA	10	AY	Antarctica		14000000	0	AN	.aq		
AR	ARG	32	AR	Argentina	Buenos Aires	2766890	41343201	SA	.ar	ARS	Peso
AS	ASM	16	AQ	American Samoa	Pago Pago	199	57881	OC	.as	USD	Dollar
AT	AUT	40	AU	Austria	Vienna	83858	8205000	EU	.at	EUR	Euro
AU	AUS	36	AS	Australia	Canberra	7686850	21515754	OC	.au	AUD	Dollar
AW	ABW	533	AA	Aruba	Oranjestad	193	71566	NA	.aw	AWG	Guilder
AX	ALA	248		Aland Islands	Mariehamn		26711	EU	.ax	EUR	Euro
AZ	AZE	31	AJ	Azerbaijan	Baku	86600	8303512	AS	.az	AZN	Manat
BA	BIH	70	BK	Bosnia and Herzegovina	Sarajevo	51129	4590000	EU	.ba	BAM	Marka
BB	BRB	52	BB	Barbados	Bridgetown	431	285653	NA	.bb	BBD	Dollar
BD	BGD	50	BG	Bangladesh	Dhaka	144000	1,56E+08	AS	.bd	BDT	Taka
BE	BEL	56	BE	Belgium	Brussels	30510	10403000	EU	.be	EUR	Euro
BF	BFA	854	UV	Burkina Faso	Ouagadougou	274200	16241811	AF	.bf	XOF	Franc
BG	BGR	100	BU	Bulgaria	Sofia	110910	7148785	EU	.bg	BGN	Lev
BH	BHR	48	BA	Bahrain	Manama	665	738004	AS	.bh	BHD	Dinar
BI	BDI	108	BY	Burundi	Bujumbura	27830	9863117	AF	.bi	BIF	Franc
BJ	BEN	204	BN	Benin	Porto-Novo	112620	9056010	AF	.bj	XOF	Franc
BL	BLM	652	TB	Saint Barthelemy	Gustavia	21	8450	NA	.gp	EUR	Euro

...

...

Und wofür braucht man das?

UNI FREIBURG

- Uns interessieren die Spalten 5 (Land) und 9 (Kontinent)
- Welche Kontinente haben wie viele Länder?
- Wie würden Sie das lösen?
- Wir schauen uns zwei typische Lösungen dafür an
  - Mit Sortieren
  - Mit einem assoziativen Array aka Map

Lösung 1: Sortieren

UNI FREIBURG

- Idee
  - Wir sortieren die Zeilen (Spalte 5 und 9) nach Kontinent
  - Dann stehen alle Länder im selben Kontinent in einem Block hintereinander
  - Die Größe von jedem Block können wir dann zählen
  - Und dann den größten Block finden
- Nachteil
  - Sortieren braucht Zeit $\Theta(n \log n)$ und geht nicht in $O(n)$
  - Wir müssen **zweimal** über die ganzen Daten laufen
- Vorteil
  - Algorithmisch einfach
  - Kommandozeile: geht mit einfachen Unix/Linux-Befehlen

Einfache Lösung mit Unix-Befehlen

UNI FREIBURG

- Daten liegen als reine Text-Datei vor
  - Spalten getrennt mit Tabulator
  - Kommentarzeilen beginnen mit Doppelkreuz (#)
- **grep** wählt bestimmte Zeilen aus

```
grep –v ‘^#‘ countryInfo.txt
```

„nicht“ „Zeilenanfang“

- **cut** wählt bestimmte Spalten aus (Tab separiert)

```
cut –f5,9
```

„Spalte 5 und 9“

Einfache Lösung mit Unix-Befehlen

- **sort** sortiert Zeilen

```
sort -t '       ' -k2,2
```

„Spaltentrenner“ „Tab“ (einfügen mit CTRL-v TAB) Sortierschlüssel ist Spalte 2 bis 2

- **uniq** findet oder zählt eindeutige Schlüssel (Zeilen)

```
uniq -c
```

„zählen“

- **less** zeigt Datei Bildschirmweise an
- **head** gibt n Zeilen am Anfang der Datei aus

```
head -n30
```

Liste nach Kontinenten sortieren

UNI FREIBURG

```
grep -v '^#' countryInfo.txt | cut -f5,9 | sort -t'	' -k2,2 | less
```

```
Algeria	AF
Angola	AF
Benin	AF
Botswana	AF
Burkina Faso	AF
Burundi	AF
Cameroon	AF
Cape Verde	AF
Central African Republic	AF
Chad	AF
Comoros	AF
Democratic Republic of the Congo	AF
Djibouti	AF
```

Länder pro Kontinent zählen

UNI FREIBURG

```
grep -v '^#' countryInfo.txt | cut -f9 | sort | uniq -c | sort -nr
```

```
58 AF
54 EU
52 AS
42 NA
27 OC
14 SA
 5 AN
```

Lösung 2: Assoziatives Array / Map

UNI FREIBURG

- Idee
  - Ein assoziatives Array mit dem Kontinent als Index
  - Der Wert ist entweder einfach ein Zähler, oder die Liste aller Länder in dem Kontinent
- Vorteil
  - Laufzeit O(n)
  - Vorausgesetzt wir können in Zeit O(1) auf ein Element des Arrays zugreifen, wie bei einem normalen Array
  - Dass das geht, zeigen wir in der nächsten Vorlesung
  - Aber erstmal sollen Sie verstehen, wozu und wie man assoziative Arrays benutzt !

Map in Java und C++ 1/3

UNI FREIBURG

- In Java und C++ heißen die assoziativen Arrays **map**
  - Der Index heißt dort **key**, das Element heißt **value**
  - Eine map unterstützt u.a. die folgenden Operationen
    - **Einfügen** von Element value mit Schlüssel key
      - in Java put(key, value) ... in C++ insert(key, value)
    - **Zugriff** auf das Element mit Schlüssel key
      - in Java get(key) ... in C++ operator[](key)
    - **Löschen** des Elementes mit Schlüssel key
      - in Java remove(key) ... in C++ erase(key)
    - **Fragen** ob Element mit Schlüssel key da ist
      - in Java containsKey(key) ... in C++ count(key)

A["abc"]

Map in Java und C++ 2/3

UNI FREIBURG

- Effizienz
  - Hängt von der Implementierung ab, es gibt insbesondere
    - In Java: java.util.HashMap und java.util.TreeMap
    - In C++: __gnu_cxx::hash_map und std::map
      - in C++11 ist die hash map std::unordered_map
  - Was das genau ist, sehen wir in der nächsten Vorlesung
    - da bauen wir uns unsere eigene **map**
    - und analysieren sie dann

Map in Java und C++ 3/3

UNI FREIBURG

- Vorsicht bei folgendem Feature der map in C++
  - Nehmen wir an, wir haben eine map mit Schlüsseln vom Typ std::string und Elementen vom Typ int

    ```
    std::map<std::string, int> M;
    ```

  - Dann kann man auf Elemente einfach mit dem [] Operator zugreifen wie bei einem normalen Array

    ```
    M["europe"] = 54;
    M["asia"] = 52;
    ```

  - Aber Vorsicht, wenn das Element vorher nicht in der map war, wird es durch [] angelegt, mit dem Defaultwert für den Typ von value, für int ist das 0.

    ```
    if (M["venus"] > 0) ...          // Inserts "venus" with value 0.
    if (M.count("venus") > 0) ...    // Only asks if "venus" is there.
    ```

Übungsblatt 4

UNI FREIBURG

- Ermitteln Sie die häufigsten Städtenamen der Welt
- Quelle „allCountries.txt“ von geonames.org (ca. 8 Millionen Einträge)
- per Sortieren und per Hashmap

Benutzung von std::sort und std::map

UNI FREIBURG

- Live im Editor…

Literatur / Links

- Assoziative Arrays
  - In Mehlhorn/Sanders:

    4 Hash Tables and Associative Arrays (das führt schon weiter)
  - In Cormen/Leiserson/Rivest

    12 Hash Tables (das ebenso)
  - In Wikipedia

    http://de.wikipedia.org/wiki/Assoziatives_Array

    http://en.wikipedia.org/wiki/Associative_array
- Map in Java und in C++

  http://download.oracle.com/javase/1.4.2/docs/api/java/util/Map.html
  - und ...HashMap bzw. ...TreeMap

  http://www.cplusplus.com/reference/stl/map/