Informatik II: Algorithmen und Datenstrukturen SS 2017

Vorlesung 4a, Dienstag, 16. Mai 2017

(Assoziative Felder aka Maps)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

- Organisatorisches
  - Erfahrungen mit dem ÜB3 — O-Notation
  - Kommunikation mit Tutor — RZ-Kürzel
  - Update Jenkins — Python checkstyle
- Inhalt
  - Assoziative Felder — Definition + Beispiele
  - Anwendungsbeispiel — MapCountingSort
  - Bibliotheken dafür — in Python / Java / C++
  - ÜB4: Implementieren Sie ein assoziatives Feld + benutzen Sie es für eine konkrete Anwendung (Wortfrequenzen)

Erfahrungen mit dem ÜB3 1/3

UNI FREIBURG

- Zusammenfassung / Auszüge
  - Für die meisten gut machbar (besser als das ÜB2)
  - "Sogar für mathe-phobe Studierende gut lösbar"
  - Beispiele in der VL seien einfacher als auf dem ÜB
  - Aufgabe 3 hat Spaß gemacht
  - Wie zeigt man, dass eine Funktion **nicht** in Ω(...) ist?
  - "Ich kann die Landau Notation langsam nicht mehr sehen"
  - "Diese Landau-Notation ist wirklich eine feine Sache"
  - "Unsicherheit: Wann hat man etwas komplett bewiesen"
  - Bereitstellung der Videoaufnahmen großer Luxus!
  - Rhythmus der Vorlesung dürfe gerne schneller werden

Erfahrungen mit dem ÜB3 2/3

- Wie zeigt man g ≠ Ω(f)
  - Zum Beispiel $n \neq \Omega(n^2)$ … also $g(n) = n$, $f(n) = n^2$
  - Zur Wiederholung, noch einmal die Definition von Ω(f)

    $\{ g : \mathbf{N} \to \mathbf{R} \mid \exists n_0 \in \mathbf{N} \;\; \exists C>0 \;\; \forall n \geq n_0 \;\; g(n) \geq C \cdot f(n) \}$
  - Da steht $\exists n_0 \in \mathbf{N}$ und $\exists C>0$
  - Also nehmen wir doch mal an, dass die existieren, und versuchen, das zu einem Widerspruch zu führen

$\forall n \geq n_0 : \; \underbrace{n \geq C \cdot n^2}_{g(n) \quad g(n)} \Leftrightarrow 1 \geq C \cdot n \Leftrightarrow \frac{1}{C} \geq n$

$\forall n \geq n_0 : \; n \leq \underbrace{\frac{1}{C}}_{\text{Konstante}}$ ↯

Erfahrungen mit dem ÜB3 3/3

- Laufzeiten der Programme von Aufgabe 3
  - Funktion **quad(n)** … Rückgabewert n^2

    Anzahl Durchläufe der inneren Schleife = Endwert von "result" = $n + (n-1) + \ldots + 1 = n \cdot (n+1)/2 = \mathbf{\Theta(n^2)}$
  - Funktion **id(n)** … Rückgabewert n

    Anzahl der Durchläufe der inneren Schleife = Summe aller Zähler in "counts" = **n** … weil: in der ersten Schleife wird n mal genau einer der Zähler um genau 1 erhöht
  - Funktion **rev(n)** … Rückgabewert [n, …, 1]

    Die Laufzeit der Funktion index auf einem Feld der Größe m ist **nicht** konstant, sondern bis zu Θ(m)

    → Gesamtlaufzeit hier $\mathbf{\Theta(n^2)}$, trotz einfacher Schleife

Kommunikation mit Tutor

- Erinnerung
  - Sie können Ihrem Tutor / Ihrer Tutorin bei Fragen oder Problemen gerne eine Mail schreiben
  - Geben Sie dabei bitte das Kürzel von Ihrem RZ-Account mit an (Initialen + Zahl)
  - Ihr Tutor / Ihre Tutorin findet Sie dann leichter im System wieder

Jenkins Update

- Für den Python checkstyle
  - Die neue Version prüft nun auch die Benennung von Funktions- und Variablennamen
  - Außerdem muss die Importreihenfolge jetzt alphabetisch sein + Systemheader müssen vor anderen stehen
  - Lokal bekommt man diese Änderung mit:

    pip install pep8-naming==0.4.1 flake8-import-order==0.12
  - Auf Jenkins ist das schon installiert

    Keine Sorte: das ist alles Standard und keine große Sache

Assoziative Felder 1/10

UNI FREIBURG

- Definition
  - Verwalten einer Menge von n Elementen, jedes mit einem eindeutigen Schlüssel S aus einer beliebigen Menge U

    Terminologie: Schlüssel = **Key**, Element = **Value**
  - Uns interessieren insbesondere folgende Operationen:

    insert(key, value) Einfügen von value mit Schlüssel key

    lookup(key) Ist key ϵ S und wenn ja mit welchem Wert

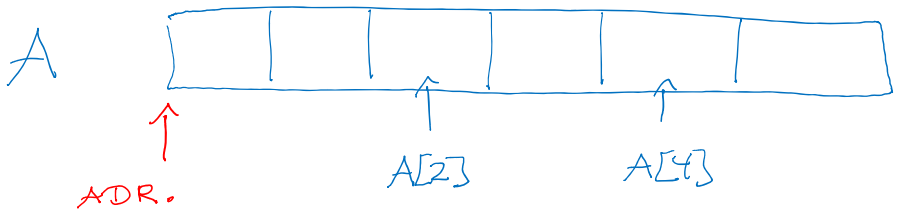
    erase(key) Falls key ϵ S, dann das Elemente löschen

    Wir schauen uns heute erstmal nur "insert" und "lookup" an
  - Terminologie: ein assoziatives Feld heißt in den gängigen Programmiersprachen meistens **Map** oder **Dictionary**

Assoziative Felder 2/10

- Anwendungsbeispiel 1
  - Das normale Feld ist ein Spezialfall mit $S = \{0, \ldots, n-1\}$
  - Der Schlüssel von einem Element ist dann gerade die Position in dem Feld, in dem Fall genannt "Index"
  - Dann bekommt man

Laufzeit insert:	$\Theta(1)$	besser geht's nicht
Laufzeit lookup:	$\Theta(1)$	besser geht's nicht
Platzverbrauch:	$\Theta(n)$	besser geht's nicht



Assoziative Felder 3/10

- Anwendungsbeispiel 2
  - Datensätze aller Informatik Studierenden mit Matrikelnummer

    1234567 Studi X
    3276432 Studi Y
    2334523 Studi Z

  - Die Matrikelnummer ist eindeutig pro Datensatz, von daher ist das ein geeigneter Schlüssel
  - Die Schlüsselmenge S (Matrikelnummern der Infostudis) ist viel kleiner als die Menge U aller möglichen Matrikelnummern

    U = {0, …, 9.999.999} **...** bei bis zu 7-stelligen Nummern

Assoziative Felder 4/10

EINGABE:

12, 17, 17, 12, 12, 4, 4, 4, 12,
4, 4, 17, 4, 4, 12

- Anwendungsbeispiel 3
  - Die verschiedenen Zahlen aus der Eingabe zu einem Sortieralgorithmus, und wie oft sie jeweils vorkommen

12	5 mal
17	3 mal
4	7 mal

    U = {Menge aller möglicher Zahlen}
  - Die Schlüsselmenge S ist hier ebenfalls viel kleiner als U

    Dazu schreiben wir nachher zusammen ein Programm

Assoziative Felder 5/10

- Anwendungsbeispiel 4
  - Worthäufigkeiten in einem gegebenen Text

umfang	kommt 16 mal vor
zuständig	kommt 4 mal vor
fachprüfungsausschuss	kommt 88 mal vor

  - Ähnlich wie Beispiel 3, nur dass die Schlüssel jetzt Zeichenketten sind, dazu mehr in der Vorlesung morgen
  - Auf dem ÜB4 sollen gerade Worthäufigkeiten berechnet werden, und zwar für die Prüfungsordnungen der Informatik

Assoziative Felder 6/10

- Anwendungsbeispiel 5
  - Wie Beispiel 3, nur mit einem kleineren U und abstrakten Elementen, als handliches Beispiel für die nächsten Folien

    5 Element 1
    14 Element 2
    12 Element 3

  - Die Schlüssel kommen aus der Menge U = {0, …, 19}
  - Die genau Beschaffenheit der Elemente (Values) wird auf den nächsten Folien keine Rolle spielen: es sind einfach nur Daten, die an der betreffenden Stelle gespeichert werden

Assoziative Felder 7/10

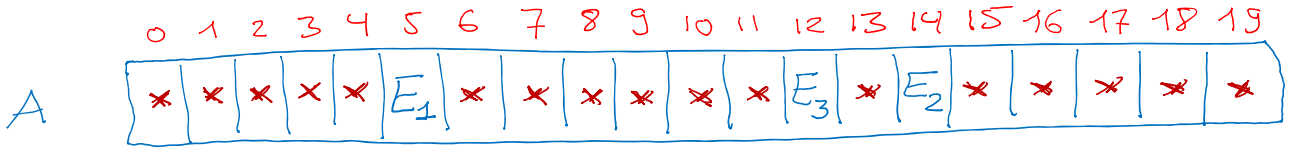
5 : E_1
14 : E_2
12 : E_3
$U = \{0, \ldots, 19\}$

- Realisierung 1
  - Feld A der Größe |U|, für jeden Schlüssel i steht an A[i] das zugehörige Element, sonst ein spezieller Wert ×
  - Dann bekommen wir:

GUT Laufzeit insert: $\Theta(1)$ ... d.h. konstante Zeit

GUT Laufzeit lookup: $\Theta(1)$

FURCHTBAR Platzverbrauch: $\Theta(|U|)$ ... wäre $\Theta(|S|)$ (# Elemente)



Assoziative Felder 8/10

5 : E_1
14 : E_2
12 : E_3

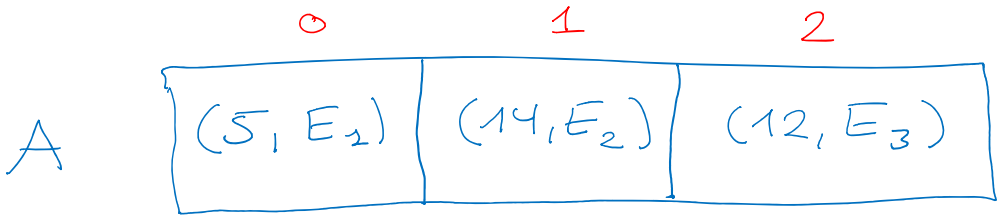
- Realisierung 2
  - Feld der Größe |S| = Anzahl Schlüssel, an Stelle i steht einfach das i-te Elemente zusammen mit seinem Schlüssel
  - Dann bekommen wir:

ohne nachzuprüfen, ob Element schon drin steht

GUT Laufzeit insert: $\Theta(1)$ ... einfach am Ende anzeigen

SCHLECHT Laufzeit lookup: $\mathcal{O}(|S|)$ ... kann am Anfang stehen, aber auch am Ende

GUT Platzverbrauch: $\Theta(|S|)$



Assoziative Felder 9/10

5 : E_1
14 : E_2
12 : E_3

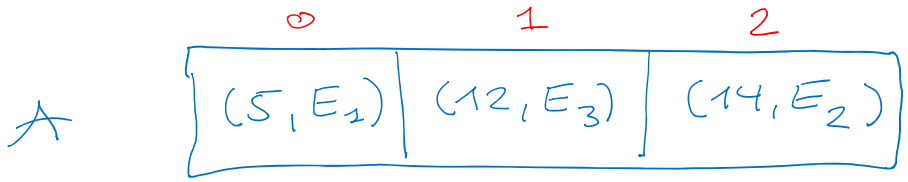
- Realisierung 3
  - Wie Realisierung 2, aber sortiert

    Dann geht lookup schneller (mit binärer Suche), aber insert langsamer (die Sortierung muss gewahrt bleiben)
  - Dann bekommen wir:

    Laufzeit insert: $\Theta(\log |S|)$ + Einfügen in ein Feld — kann bis zu $\Theta(|S|)$ kosten

    OK Laufzeit lookup: $\Theta(\log |S|)$ ... binäre Suche

    GUT Platzverbrauch: $\Theta(|S|)$



Assoziative Felder 10/10

- Realisierung 4 (Wunsch)
  - Wir hätten gerne eine Datenstruktur, mit der wir für eine Menge von n Elementen mit beliebigen Schlüsseln aus einer beliebigen Menge U bekommen:

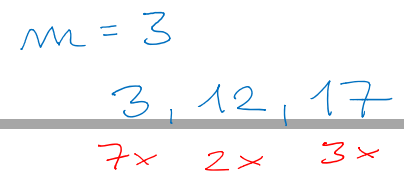
    Laufzeit insert: $\Theta(1)$

    Laufzeit lookup: $\Theta(1)$ |s|

    Platzverbrauch: $\Theta(n)$
  - Das wäre das Optimum, den besser geht es nicht
  - Mit einer sog. "Hash Map" kriegt man das tatsächlich hin !

    Dazu morgen mehr, heute erstmal, wie man das **benutzt**

MapCountingSort 1/3

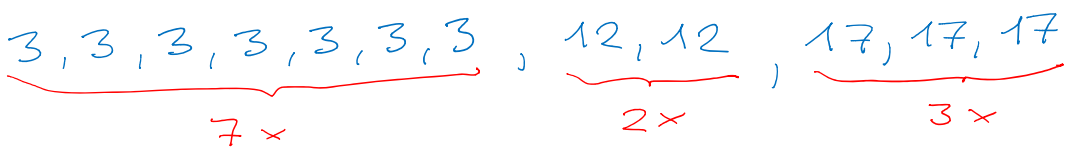


- Unterschied zu CountingSort
  - Mit CountingSort: sortieren mit Zeit O(n) und Platz O(m) wenn die n Zahlen aus dem Bereich 1..m waren
  - Mit einer Map können wir das jetzt verallgemeinern zu:

    n **beliebige** Zahlen, aber höchstens m verschiedene

    Beispiel: 17 17 3 3 17 3 3 3 3 12 12 3

    Lösung: 3,3,3,3,3,3,3 ; 12,12 ; 17,17,17

  - Das programmieren wir jetzt zusammen in Python und benutzen dabei Pythons assoziatives Feld (= Dictionary)

MapCountingSort 2/3

- Algorithmus am Beispiel

  Beispiel: 17 17 3 3 17 3 3 3 3 12 12 3

  – **Schritt 1:** Mit einer Map die Anzahl Vorkommen zählen

  Beispiel: 17: 3 mal, 3: 7 mal, 12: 2 mal

  – **Schritt 2:** Diese Anzahlen nach den Werten sortieren

  Beispiel: (3, 7), (12, 2), (17, 3)

  Zum Sortieren in ein Feld von Paaren (Wert, Anzahl) schreiben, dieses Feld dann nach den Werten sortieren

  – **Schritt 3:** Dann die Ausgabe schreiben wie gehabt

  Beispiel: 3, 3, ..., 3, 12, 12, 17, 17, 17

MapCountingSort 3/3

- Laufzeitanalyse
  - Die Laufzeit ist $\Theta(n \cdot M + m \cdot \log m)$, wobei M = die durchschnittliche Kosten einer Map Operation

    Schritt 1: $\Theta(n \cdot M)$ n Map Operationen

    Schritt 2: $\Theta(m \cdot \log m)$ z.B. mit MergeSort

    Schritt 3: $\Theta(n)$ die n Werte ausgeben
  - Das ist gut, wenn M klein und $m \ll n$
  - Insbesondere: linear, wenn $M = O(1)$ und $m = O(n / \log n)$

$\Rightarrow m \cdot \log m \leq n$

$m \leq n / \log n \leq m$

$n / \log n \leq m \leq \log n$

Rechenbeispiel für n und m: $n = 2^{40} \approx 10^{12} = 1$ TERA

$\Rightarrow n / \log n = \frac{1 \text{ TERA}}{40}$.

Ass. Felder in Python, Java, C++ 1/5

- Python ... da heißt ein assoziatives Feld **Dictionary**
  - Grundoperationen

	keine Typinformation nötig
map = {}	Erzeuge leere Map
map[key] = value	Einfügen von key mit Wert value
value = map[key]	Wert für key
if key in map: …	Fragen, ob key enthalten ist
del map[key]	Löschen von key
list(maps.items())	Alle key, value Paare als Feld

Ass. Felder in Python, Java, C++ 2/5

- Java ... da heißt ein assoziatives Feld **Map**
  - Grundoperationen

	K = key type, V = value type
Map<K, V> map = …	Erzeuge leere Map
map.put(key, value);	Einfügen von key mit Wert value
value = map.get(key);	Wert für key
if (map.containsKey(key)) …	Fragen, ob key enthalten ist
map.remove(key);	Löschen von key
map.entrySet();	Alle key, value Paare

Ass. Felder in Python, Java, C++ 3/5

- C++ ... da heißt ein assoziatives Feld ebenfalls **Map**
  - Grundoperationen

	K = key type, V = value type
std::map<K, V> map;	Erzeuge leere Map
map[key] = value;	Einfügen von key mit Wert value
value = map[key];	Wert für key
if (map.count(key)) …	Fragen, ob key enthalten ist
map.erase(key)	Löschen von key
for (auto item : map) …	Iteration über alle key, value Paare

Ass. Felder in Python, Java, C++ 4/5

- C++
  - Achtung bei folgendem Nebeneffekt:

    `if (map[key]) ...` — Fügt key mit Wert 0 ein, falls key bisher nicht in map war

    `if (map.count(key) > 0) ...` — Fragt nur, ob key in map ist

  - Das ist gefährlich, kann aber auch nützlich sein, z.B.

    `map[key]++;` — Erhöht den Wert von key; falls key noch nicht in map, wird vorher mit 0 intialisiert

Ass. Felder in Python, Java, C++ 5/5

- Effizienz
  - Hängt von der Implementierung ab; effizient sind

    Hashtabellen Vorlesungen 4b, 5a, 5b

    Suchbäume Vorlesungen 8a und 8b
  - In den diversen Programmiersprachen:

    Java: java.util.HashMap und java.util.TreeMap

    C++11: std::unordered_map und std::map

    Python: ist dem Compiler überlassen

Literatur / Links

- Assoziative Arrays
  - In Mehlhorn/Sanders:

    4 Hash Tables and Associative Arrays (das führt schon weiter)
  - In Wikipedia

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

    http://en.wikipedia.org/wiki/Associative_array
  - In Python, Java, C++

    http://docs.python.org/tutorial

    http://docs.oracle.com/javase

    http://www.cplusplus.com/reference