Algorithmen und Datenstrukturen (für ESE) WS 2011 / 2012

Vorlesung 3, Dienstag, 15. November 2011
(Assoziative Arrays)

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

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Übungsgruppen diese Woche!
  - Ihr Code muss kompilieren / Checkstyle, Ant, Jenkins
  - Machen Sie regelmäßig ein svn update
  - Zusammenarbeit, was ist erlaubt?
  - Ihre Erfahrungen mit dem 2. Übungsblatt
- Assoziative Arrays
  - Wir werden an einem typischen Beispiel sehen, was das ist und wofür die gut sind
  - **Übungsaufgabe:** populäre Anfragen in der Log-Datei einer Suchmaschine finden
    - einmal mit Sortieren, und einmal mit assoziativen Arrays

Übungsgruppen

UNI FREIBURG

- Fangen diese Woche an!
  - Termine und Ort, siehe Wiki
    - **Sebastian Sester**, Dienstag 14:15 Uhr, Raum 51-03-026
    - **Katja Faist**, Mittwoch 12:15 Uhr, Raum 51-00-031
    - **Manuel Bührer**, Donnerstag 12:15 Uhr, Raum 51-00-031

Code muss kompilieren!

UNI FREIBURG

- Wenn Ihr Code nicht kompiliert
  - ... ist es **nicht** die Aufgabe Ihres Tutors nach dem Fehler zu suchen, das kann nämlich ewig dauern
  - Ihr Tutor hat dann das Recht, Ihnen **0** Punkte zu geben
    - es ist nämlich außerdem extrem viel Arbeit, Code zu beurteilen, den man nicht mal kompilieren und laufen lassen kann
    - wenn er / sie nett ist, bittet er Sie vielleicht per Mail, selber den Fehler zu finden und mitzuteilen, wo der Fehler liegt, aber verlassen Sie sich nicht darauf
- Außerdem
  - Bitte checkstyle und ant beachten, das gibt sonst in Zukunft Punktabzug

SVN update

- Machen Sie svn update
  - ... spätestens bevor Sie ein neues Übungsblatt bearbeiten
  - Sie bekommen dann insbesondere
    - Feedback von Ihrem Tutor zu Ihrem letzten Übungsblatt, das steht in

      non-code/uebungsblatt_X/feedback-tutor.txt
    - eventuelle Updates, z.B. aktuell eine neue checkstyle_config.xml

Ist Zusammenarbeit erlaubt?

UNI FREIBURG

- Sie dürfen
  - ... gerne mit Ihren KomilitonInnen zusammen
    - über die Übungsaufgaben nachdenken
    - gemeinsam Lösungsstrategien entwerfen
- Aber Sie sollen
  - ... die Details dann jede/r für sich ausarbeiten
    - inbesondere Code und Beweise
  - Das hat einen einfachen Grund
    - sonst lernt man nichts
    - oder noch schlimmer: man denkt man hat es verstanden, hat es aber nicht verstanden

Ihre Erfahrungen mit dem 2. Ü-Blatt

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback Stand 15.11 15:42
  - Ü-Blatt sehr gut machbar / viel leichter als das letzte
  - Typischer Zeitaufwand 2 – 5h ... so darf es gerne bleiben
  - Vorlesung dazu war sehr gut verständlich
  - "System zum Hochladen ist gewöhnungsbedürftig"
  - "Komme mir vor wie wenn ich ein Tagebuch führe"
  - Probleme mit der Definition von O etc. wenn $f : N \to R$ negativ
    - **Lösung 1:** In den Definitionen $|...|$ verwenden
      - So steht es zum Beispiel in Wikipedia
    - **Lösung 2:** Wir betrachten nur f mit $\lim_{n\to\infty} f(n) > 0$
      - Am Anfang dürfen sie aber ruhig mal < 0 sein
  - Es gibt auch noch o und ω ... die brauchen wir hier aber nicht

Normale vs. Assoziative Arrays

- 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
- Assoziatives Array
  - Zugriff auf eine (große) Anzahl von gleichartigen Elementen über einen beliebigen sog. **Schlüssel** (Key)
  - A_{blau}, A_{gelb}, A_{orange}, A_{rot}, $A_{schwarz}$, ...
  - Beispielanfrage: das Element mit Schlüssel gelb
  - 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
- Ein typisches Beispiel
  - Gegeben eine lange Liste von Anfragen an eine Suchmaschine
  - Wir wollen wissen, welche Anfrage am häufigsten vorkommt
  - Man beachte, dass es theoretisch unendlich viele verschieden Anfragen gibt, von denen aber nur relativ wenige tatsächlich vorkommen
  - Wir werden uns **drei** Lösungen für dieses Problem anschauen: mit Sortieren, mit einem normalen Array und mit einem assoziativem Array

Lösung 1: Normales Array

UNI FREIBURG

- Idee
  - Wir merken uns für jede Anfrage, wie oft wir sie schon gesehen haben
  - Dabei können wir uns auch leicht zu jedem Zeitpunkt die bis dahin häufigste Anfrage merken
- Nachteil
  - Nehmen wir an, wir haben schon n verschiedene Anfragen gesehen
  - Wie finden wir dann für die nächste Anfrage heraus, ob es eine neue oder eine von diesen n ist?
  - Mit einem normalen Array geht das in $O(n)$ Zeit
  - Bei n verschiedenen Anfragen also insgesamt $\Theta(n^2)$ Zeit

Lösung 2: Sortieren

UNI FREIBURG

- Idee
  - Wir sortieren die Anfragen lexikographisch
  - Dann haben wir Blöcke von gleichen Anfragen
  - Die Größe von jedem Block können wir zählen
  - Und dann den größten Block finden
- Nachteil
  - Wir müssen einmal alles sortieren, das ist teuer und kostet außerdem Platz
  - Wir müssen zweimal über die ganzen Daten laufen
- Vorteil
  - Algorithmisch einfach + geht von der Kommandozeile aus mit einfachen Unix/Linux-Befehlen

Lösung 3: Assoziatives Array

UNI FREIBURG

- Idee
  - Ein assoziatives Array von Zählern mit der Suchanfrage als Index
  - Und wie vorher merken wir uns zu jedem Zeitpunkt die bis dahin häufigste Suchanfrage
- Vorteil
  - Nehmen wir an wir können in Zeit $O(1)$ auf einen der schon vorhandenen Zähler zugreifen bzw. herausfinden, dass es für diesen Index noch keinen Zähler gibt
  - Dann hat unser Programm für eine Liste von n Suchanfragen Laufzeit $O(n)$

Effiziente assoziative Arrays 1/2

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)

Effiziente assoziative Arrays 2/2

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

Vorsicht bei den maps in C++

UNI FREIBURG

- Sie haben ein besonderes "feature"
  - 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["blau"] = 5;
    M["gelb"] = 0;
    ```

  - 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["lila"] >0) ...  // Inserts "lila" with value 0.
    ```

    ```
    if (M.count("lila") > 0) ...  // Only asks if "lila" is there.
    ```

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/