Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

Vorlesung 4, Montag, 15. November 2010

(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
  - Morgen ist wieder eine Übungsgruppe
  - Ihre Erfahrungen mit dem 3. Übungsblatt
- Assoziative Arrays
  - Wir werden an einem typischen Beispiel sehen, was das ist und wofür sie gut sind
  - Die Übungsaufgabe wird diesmal ein Programm sein, wo man assoziative Arrays gut gebrauchen kann

Ihre Erfahrungen mit dem 3. Ü-Blatt

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback
  - Ein sehr mathe-lastiges Übungsblatt
  - Die meisten fanden es diesmal vom Aufwand und Schwierigkeitsgrad her angemessen
  - "Ich konnte dieses Mal viel mehr mit der Vorlesung anfangen und das Blatt ist im Allgemeinen leichter gefallen. Falls jedoch alles falsch ist, war die Vorlesung nicht gut."
  - LaTeX hat für einige viel Zeit gekostet
  - Was muss man beweisen und was darf man einfach so behaupten? Zum Beispiel $2^n > n$ für alle $n \geq 1$
    - Gute Frage ohne einfache Antwort

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)
  - $A_{Paprika}$, $A_{Sellerie}$, $A_{Kohlrabi}$, A_{Tomate}, ...
  - Beispielanfrage: das Element mit Schlüssel Kohlrabi
  - Die Schlüssel können auch (beliebige) 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 verschiedene 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: 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 2: 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
- Problem
  - 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 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

UNI FREIBURG

- In der C++ STL bzw. in java/util ...
  - ... heißt das assoziative Array einfach **map**
  - Der Index heißt dort **key**, das Element heißt **value**
  - Eine map unterstützt u.a. die folgenden Operationen
    - insert(key, value) : Einfügen von Element value mit Schlüssel key
    - get(key): Zugriff auf das Element mit Schlüssel key
    - erase(key) : Löschen des Elementes mit Schlüssel key
- Effizienz
  - Hängt von der Implementierung ab
  - Das machen wir in der nächsten Vorlesung, da bauen wir uns unsere eigene map ... und analysieren sie dann

Vorsicht bei der STL map

UNI FREIBURG

- Sie hat 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["kohlrabi"] = 5;
    M["sellerie"] = -12;
    ```

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

    ```
    if (M.count("birne") > 0) ...  // Only asks if "birne" 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 C++ und Java

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

  http://download.oracle.com/javase/1.4.2/docs/api/java/util/Map.html