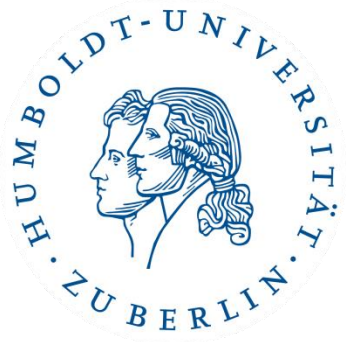
Übung Algorithmen und Datenstrukturen



Sommersemester 2017

Marc Bux, Humboldt-Universität zu Berlin

Allgemeine Sortierverfahren

Ein allgemeines Sortierverfahren ist ein Sortierverfahren, welches nur Wissen über die Anordnung der zu sortierenden Elemente erlangen kann, indem es Elemente vergleicht. Ansonsten können keine weiteren Eigenschaften der Elemente vorausgesetzt bzw. ausgenutzt werden.

Welche der aus der Vorlesung bekannten Sortierverfahren sind allgemeine Sortierverfahren?

The Sound of Sorting:
https://www.youtube.com/watch?v=kPRA0W1kECg

Agenda

1. Schreibtischtests zu Sortierverfahren
2. Stabilität von Sortierverfahren

Agenda

1. Schreibtischtests zu Sortierverfahren
2. Stabilität von Sortierverfahren

MergeSort

Führen Sie einen Schreibtischtest für den Algorithmus **Mergesort** aus der VL für das Eingabe-Array

$$A = [x,\ a,\ b,\ o,\ k,\ j,\ c,\ r,\ g]$$

durch, wobei Sie als Ordnung die alphabetische Ordnung auf den Buchstaben annehmen.

Geben Sie die Zwischenschritte in Form eines Graphen wie in der VL (Folie 7) an.

QuickSort

Führen Sie einen Schreibtischtest für den Algorithmus **Quicksort** aus der VL für das Eingabe-Array

$$A = [2,\ 10,\ 6,\ 7,\ 13,\ 4,\ 1,\ 12,\ 5,\ 9]$$

durch, wobei Sie als Ordnung die natürliche Ordnung auf den natürlichen Zahlen annehmen.

Als Pivot-Element wählen Sie das am weitesten rechts stehende Element des aktuellen Teil-Arrays.

Geben Sie den aktuellen Wert von A nach jeder Swap-Operation (Folie 25, Zeilen 14, 17) an. Unterstreichen Sie jeweils das in diesem Aufruf von **divide**(A, l, r) betrachtete Teil-Array $A[l..r]$.

BucketSort

In dieser Teilaufgabe nutzen wir den Algorithmus **Bucketsort**, um Arrays von Zeichenketten gleicher Länge über einem festen endlichen Alphabet Σ zu sortieren. Wir nehmen also an, dass die Einträge des übergebenen Arrays alles Elemente von Σ^m für eine bestimmte Zahl $m \in \mathbb{N}_{>0}$ sind, wobei Σ^m wie üblich die Menge aller Zeichenketten über Σ der Länge m ist. Wir nehmen weiterhin an, dass das Alphabet Σ linear geordnet ist, und dass die Ordnung auf den Zeichenketten die *lexikographische* Ordnung ist. Es gilt also: $a_1 \dots a_m < b_1 \dots b_m$ g.d.w. ein i mit $1 \leq i \leq m$ existiert, so dass $a_i < b_i$ und $a_j = b_j$ für alle $j < i$. Führen Sie einen Schreibtischtest für $\Sigma = \{0, 1, 2, 3\}$, $m = 3$, und das Array

$$A \;=\; [103,\; 202,\; 101,\; 231,\; 022,\; 031,\; 030,\; 233,\; 201]$$

durch.

Notieren Sie nach jedem Durchlauf der Schleife mit Laufvariable i den Inhalt des Arrays A. Markieren Sie wie in der VL (Folie 19) mit vertikalen Strichen, welche Elemente von A sich in dieser Iteration gemeinsam in einem Bucket befanden.

Agenda

1. Schreibtischtests zu Sortierverfahren
2. Stabilität von Sortierverfahren

Stabilität von Sortierverfahren

In dieser Aufgabe betrachten wir Arrays mit Einträgen des abstrakten Datentyps Element. Grundlage dafür bilden eine Menge K von Schlüsseln, eine Menge V von Werten und eine lineare Ordnung $\leq$ auf K. Ein Element e hat einen Schlüssel e.key $\in K$ und einen Wert e.val $\in V$. Wir notieren ein Element e auch als Paar (e.key, e.val). Ist beispielsweise $K = \{a, b\}$ und $V = \mathbb{N}$, dann schreiben wir $(a, 17)$ für ein Element e mit e.key = a und e.val = 17.

Sind e_1 und e_2 vom Typ Element, können wir e_1 und e_2 auf Basis ihrer Schlüssel vergleichen: Es gelte genau dann $e_1 \leq e_2$, wenn e_1.key $\leq e_2$.key ist. Ist beispielsweise $K = \mathbb{N}$, $V = \{a, b, c\}$ und $\leq$ die natürliche Ordnung auf $\mathbb{N}$, dann gilt $(2, \mathrm{c}) \leq (4, a)$.

Ein Sortierverfahren heißt stabil, wenn Elemente mit gleichen Schlüsseln nach der Sortierung in der gleichen Reihenfolge aufeinander folgen wie vor der Sortierung.

SelectionSort

Betrachten Sie den Algorithmus SelectionSort. Ist **SelectionSort** ein stabiles Sortierverfahren? Beweisen Sie Ihre Antwort.

SelectionSort(Array A)

Input: Array A von n Werten
Output: Sortiertes Array A.

```
1: for i = 0 ... n − 2 do
2:     for j = i + 1 ... n − 1 do
3:         if A[i] > A[j] then
4:             tmp = A[j];
5:             A[j] = A[i];
6:             A[i] = tmp;
7:         end if
8:     end for
9: end for
```

LinearSort

Betrachten Sie den Algorithmus LinearSort.

Linearsort(Array A)

Input: Array A von n ganzen Zahlen im Bereich von 1 bis z.
Output: Array A aufsteigend sortiert.

```
1:  Initialisiere Array B der Länge z, welches überall auf 0 gesetzt ist
2:  n ← |A|
3:  Initialisiere Array C der Länge n
4:  for i = 1 to n do
5:      B[A[i]] ← B[A[i]] + 1
6:  end for
7:  for j = 2 to z do
8:      B[j] ← B[j] + B[j − 1]
9:  end for
10: for i = n downto 1 do
11:     C[B[A[i]]] ← A[i]
12:     B[A[i]] ← B[A[i]] − 1
13: end for
14: return C
```

a) Erklären Sie, was bei **Linearsort** nach der ersten und nach der zweiten **for**-Schleife im Array B steht.

b) Analysieren Sie die Laufzeit von **Linearsort** in Abhängigkeit von n und z. Geben Sie also eine möglichst gute obere Schranke der Form $\mathcal{O}(f)$ an, wobei f nur von n und z abhängt.

c) Ist **Linearsort** ein stabiles Sortierverfahren? Begründen Sie Ihre Antwort.

Ausblick

- nächste Woche:
  - Aufgabenblatt 2 durchrechnen
- zu nächster Woche:
  - mit Aufgabenblatt 3 auseinandersetzen
  - falls noch nicht vorgerechnet: zum Vorrechnen eintragen (first-come-first-served):
    Montag: https://dudle.inf.tu-dresden.de/algodat22/
    Dienstag: https://dudle.inf.tu-dresden.de/algodat32/