Übung Algorithmen und Datenstrukturen

Sommersemester 2016

Patrick Schäfer, Humboldt-Universität zu Berlin

Agenda

1. Vorstellen des vierten Übungsblatts
2. Vorbereitende Aufgaben für das vierte Übungsblatt

Allgemeine (vergleichsbasierte) Sortierverfahren

Wikipedia: „Allgemeine Sortierverfahren basieren auf dem paarweisen Vergleich der zu sortierenden Elemente. Bei der Komplexitätsanalyse wird davon ausgegangen, dass der Aufwand zum Vergleich zweier Elemente konstant ist.“

- Welche Algorithmen aus der Vorlesung sind (keine) allgemeine Sortierverfahren?

1. Selection Sort
2. Insertion Sort
3. Bubble Sort
4. Merge Sort
5. Quick Sort
6. Bucket Sort

Untere Schranke für vergleichsbasierte Sortierverfahren

Ein vergleichsbasiertes Sortierverfahren kann nicht schneller sein als:

$$\Omega(n \log n)$$

Bei Sortierverfahren, die *nicht* auf Vergleichen beruhen, kann ein linearer Anstieg der benötigten Zeit mit der Anzahl der zu sortierenden Elemente erreicht werden. Beispiele:

BucketSort: $O(m \cdot (n + k))$

mit k := Größe des Wertbereichs und
m := Anzahl der Stellen.

	Comps worst case	avg. case	best case	Additional space	Moves (wc / ac)
Selection Sort	$O(n^2)$		$O(n^2)$	$O(1)$	$O(n)$
Insertion Sort	$O(n^2)$		$O(n)$	$O(1)$	$O(n^2)$
Bubble Sort	$O(n^2)$		$O(n)$	$O(1)$	$O(n^2)$
Merge Sort	$O(n*\log(n))$		$O(n*\log(n))$	$O(n)$	$O(n*\log(n))$
QuickSort	$O(n^2)$	O(n*log(n)	O(n*log(n)	$O(\log(n))$	$O(n^2)$ / $O(n*\log(n))$
BucketSort (m= …)	$O(m*(n+k))$			$O(n+k)$	

Aufgabe (Schreibtischtest, lexikographische Ordnung)

Führen Sie einen Schreibtischtest für den Algorithmus Bubblesort für die folgenden Eingabe-Arrays durch. Geben Sie das Array A nach jedem Durchlauf der Zeilen 5-6 an.

1. $S = [5,2,7,5,5,7]$

   Ordnung: natürliche Ordnung auf den natürlichen Zahlen

2. $S = [ogr, rvv, obb, rov, vgb]$

   Ordnung: lexikographische Ordnung auf Zeichenketten.

Lexikographische Ordnung für Zeichenketten aus Σ^m (wobei Σ ein geordnetes Alphabet ist): Es gilt $a_1, \ldots, a_m < b_1, \ldots, b_m$ 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 $j < i$.

Algorithmus Bubblesort(A)

Input: Array A

Output: Sortiertes Array A

1: **repeat**
2:     swapped := **false**;
3:     **for** $i := 1$ **to** $n - 1$ **do**
4:         **if** $A[i] > A[i+1]$ **then**
5:             swap($A[i], A[i+1]$);
6:             swapped := **true**;
7:         **end if**
8:     **end for**
9: **until not** swapped
10: **return** A;

Vertauscht benachbarte Elemente, falls der Vorgänger größer ist.
Am Ende jeder Iteration der REPEAT-Schleife (Zeile 1-9) steht das Maximum am Ende des Arrays.
https://www.youtube.com/watch?v=lyZQPjUT5B4

Aufgabe (Schreibtischtest, Algorithmenanalyse)

- Führen Sie einen Schreibtischtest für den Algorithmus PositionSort für das folgenden Eingabe-Array durch. Geben Sie nach jedem Durchlauf der for-Schleife mit Laufvariablen *i* das Array *B* an. Gehen sie davon aus, dass *B* mit den Werten 0 initialisiert ist.

  $$S = [5, 2, 7, 5, 5, 7]$$

  Ordnung: natürliche Ordnung auf den natürlichen Zahlen

- Erläutern Sie den Algorithmus.
- Analysieren Sie die Laufzeit von PositionSort in Abhängigkeit von n.

PositionSort(S)

```
n := |S|;
B := neues Array der Länge n
for i = 1 .. n do
    pos := 1;
    for j = 1 .. i − 1 do
        if S[j] ≤ S[i] then
            pos := pos+1;
        end if
    end for
    for j = i + 1 .. n do
        if S[j] < S[i] then
            pos := pos+1;
        end if
    end for
    B[pos] := S[i];
end for
return B;
```

Aufgabe (Stabilität)

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

Entscheiden Sie, ob die folgenden Verfahren stabil sind. Falls das Verfahren nicht stabil ist, geben Sie eine (bitte möglichst kleine) Instanz als Gegenbeispiel an. Begründen Sie auf jeden Fall Ihre Entscheidung.

1. Position Sort
2. Selection Sort
3. Insertion Sort

Algorithmus SelectionSort(A)

Input: Array A
Output: Sortiertes Array A

```
1:  n := |A|;
2:  for i := 1 to n − 1 do
3:     min_pos := i;
4:     for j := i + 1 to n do
5:        if  A[min_pos] > A[j]  then
6:           min_pos := j;
7:        end if
8:     end for
9:     swap(A[min_pos], A[i]);
10: end for
11: return A;
```

- Bestimmt das kleinste Element im Rest-Array (Zeile 4-8).
- Am Ende jeder Iteration der FOR-Schleife (Zeile 1-9) steht das Minimum am Angang des Rest-Arrays.
- https://www.youtube.com/watch?v=Ns4TPTC8whw

Algorithmus InsertionSort(A)

Input: Array A
Output: Sortiertes Array A

```
1:  n := |A|;
2:  for i := 2 to n do
3:     j := i;
4:     key := A[j];
5:     while (A[j − 1] > key) and (j > 1) do
6:        A[j] := A[j − 1];
7:        j := j − 1;
8:     end while
9:     A[j] := key;
10: end for
11: return A;
```

- Fügt das i-te Element sortiert in das Array ein.
- Am Ende jeder Iteration der FOR-Schleife (Zeile 2-10) ist das Array bis zum i-ten Element sortiert.
- https://www.youtube.com/watch?v=ROalU379l3U

PositionSort(S)

```
1:  n := |S|;
2:  B := neues Array der Länge n
3:  for i = 1 .. n do
4:     pos := 1;
5:     for j = 1 .. i − 1 do
6:        if S[j] ≤ S[i] then
7:           pos := pos+1;
8:        end if
9:     end for
10:    for j = i + 1 .. n do
11:       if S[j] < S[i] then
12:          pos := pos+1;
13:       end if
14:    end for
15:    B[pos] := S[i];
16: end for
17: return B;
```

- Zählt Anzahl der Elemente, die kleiner sind.
- Am Ende jeder Iteration der FOR-Schleife (Zeile 3-16) steht das aktuelle Element an einer Position, so dass alle kleineren Elemente links und alle größeren Elemente rechts liegen.

Aufgabe (Sortierung spezieller Arrays)

Beweisen oder widerlegen Sie: Es gibt ein Sortierverfahren, welches ein Array von n beliebigen (vergleichbaren) Elementen in Laufzeit $\mathrm{O}(n)$ sortiert, falls …

1. … im Array nur Zahlen zwischen 1 und 1000 vorkommen.
2. ... in einem ursprünglich sortierten Array zwei beliebige Zahl vertauscht wurden.
3. … die ersten drei Viertel des Arrays bereits sortiert sind.