

Übungsserie 4

Datenstrukturen & Algorithmen

Universität Bern
Frühling 2018



Übungsserie 4

- > Lineares Sortieren, Stabilität von Sortieralgorithmen
 - > 6 theoretische Aufgaben
 - > 1 praktische Aufgabe
 - > Poolstunde: Montag 17:00-18:00
-

Theoretische Aufgaben

- > **Aufgabe 1 & 2**
 - Countingsort
 - > **Aufgabe 3**
 - Radixsort
 - > **Aufgabe 4**
 - Stabilität
 - > **Aufgabe 5**
 - Bucket-Sort
 - > **Aufgabe 6**
 - Radixsort Variante
-

Theoretische Aufgaben

> Aufgabe 1

- Countingsort auf Papier durchspielen
- Abbildung 8.2 im Buch

> Aufgabe 2

- Eingabefeld: $[a_1, \dots, a_n]$ $a_i \in [0, 1, \dots, k]$
- „Wie viele Zahlen liegen im Intervall $[a, \dots, b]$?“
- Antwort in $O(1)$!
- Einmaliger Vorverarbeitungsschritt in $\Theta(n + k)$

- *Tipps: Was beinhaltet das C-Array in Countingsort?*
- *Buch s. 195*

Theoretische Aufgaben

> Aufgabe 3

- Radixsort auf Papier durchspielen
- Stabilität beachten!
 - {{HUT}, {GUT}} nach dem letztem Zeichen sortieren:
 - {{HUT}, {GUT}}
 - ~~{{GUT}, {HUT}}~~

> Aufgabe 5

- Bucketsort auf Papier durchspielen
 - Abbildung 8.4 im Buch
-

Aufgabe 4

- > **a)** Insertionsort, Mergesort, Heapsort, Quicksort stabil?
 - Definition Stabilität: Relative Reihenfolge bleibt gleich
[..., A, ..., A, ...] → sortieren → [A, A,]
 - Pseudocode studieren
 - Antwort in einem Satz begründen

- > **b)** Wie kann ein Sortieralgorithmus stabilisiert werden?
 - *Tipp: Index im Input-Array in Vergleich miteinbeziehen!*

Aufgabe 6

> **Input** Array von ganzen Zahlen mit **variabler** Länge, z.B:

[1,
23,
444,
2,
4444,
3]

— Die Gesamtanzahl der Stellen aller Zahlen ist bekannt (n)




— D.h.: Ist m_i die Anzahl Zahlen mit i Ziffern, gilt: $\sum_i m_i \cdot i = n$

> Finde Sortieralgorithmus mit Worstcase $O(n)$

Aufgabe 6

> Radixsort?

— Für Radixsort müssen alle Zahlen gleich viele Ziffern haben!

			
3 2 9	7 2 0	7 2 0	3 2 0
	<small>IA</small>	<small>IA</small>	<small>IA</small>
4 5 7	3 5 5	3 2 9	3 5 5
	<small>IA</small>	<small>IA</small>	<small>IA</small>
6 5 7	4 3 6	4 3 6	4 3 6
	<small>IA</small>	<small>IA</small>	<small>IA</small>
8 3 9	4 5 7	8 3 9	4 5 7
	<small>IA</small>	<small>IA</small>	<small>IA</small>
4 3 6	6 5 7	3 5 5	6 5 7
	<small>IA</small>	<small>IA</small>	<small>IA</small>
7 2 0	3 2 9	4 5 7	7 2 0
	<small>IA</small>	<small>IA</small>	<small>IA</small>
3 5 5	8 3 9	6 5 7	8 3 9

Aufgabe 6

> Radixsort mit **Padding**?

[1,		[0001,
23,		0023,
444,		0444,
2,	→	0002,
4444,		4444,
3]		0003]

> **Worstcase**

- Eine Zahl mit $n/2$ Ziffern, $n/2$ Zahlen mit einer Ziffer,
also $m = \frac{n}{2} + 1$ Zahlen
- Nach Padding: Alle Zahlen haben $d = \frac{n}{2}$ Ziffern
→ Laufzeit: $\Omega(d \cdot m) = \Omega(n^2)$

Aufgabe 6

- > **Lösungsansatz** Zahlen mit mehr Ziffern sind grösser (obda erste Ziffer $\neq 0$)
 - Zahlen nach Länge gruppieren
 - Zahlengruppen einzeln sortieren

- > Komplexität bestimmen
 - Verwende, dass die Summe aller Ziffern n ist.

$$\sum_i m_i \cdot i = n$$

Praktische Aufgabe

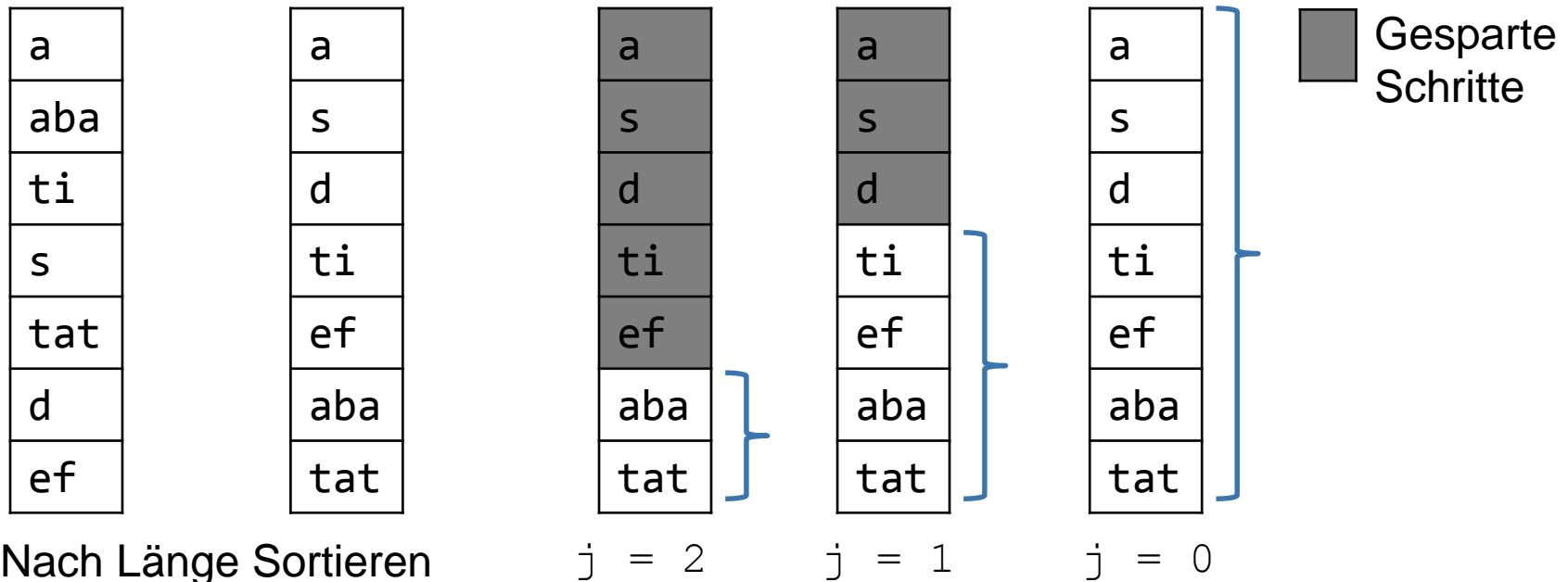
- > **Gegeben** Radixsort Implementation
 - Sortiert Zeichenketten **unterschiedlicher** Länge
 - Padding kürzerer Zeichenketten mit leerem Charakter

 - > **Teilaufgaben**
 - Komplexität der gegebenen Implementation abschätzen
 - Implementation beschleunigen
 - Beschleunigung experimentell bestätigen

 - > Ähnlich wie theoretische Aufgabe 6, aber nicht gleich!
-

Praktische Aufgabe

- > Komplexität von Radix Sort verbessern:
- **Zuerst** Wörter nach deren Länge sortieren (vgl. theoretische Aufgabe 6)
 - **Danach** Nur über Wörter mit Länge $> j$ iterieren
 - Nützlich, wenn nur wenige sehr lange Wörter



Praktische Aufgabe

- > Nach Länge sortieren in $O(n)$
 - > Countingsort!
 - C-Array erstellen:
 - $C[i] = \text{Anzahl Zeichenketten mit Länge } \geq i$
 - Information in C nutzen, um Radixsort nur auf relevante Elemente anzuwenden
-

RadixSort.java

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

} Input

char[] [] A: Array von Zeichenketten

A[i] : Zeichenkette i

A[i][j] : Zeichen j von
Zeichenkette i

d: maximale Zeichenkettenlänge

RadixSort.java

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

} **queues** Array von Listen

queues[0] //empty character ''

queues[1] //'a'

...

queues[26] //'z'

RadixSort.java

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

Wort i wird basierend auf Zeichen an Stelle j in die richtige Queue eingefügt

Bsp: $j = 2$

,abc' → queue[2]
,a' → queue[0]

Detail

queues[A[i][j] - 'a' + 1] ...

Implizites cast char → int

A[i][j] - 'a' = Index des Zeichens A[i][j]

RadixSort.java

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

Queues nach A zurück kopieren

Beispiel

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

} $A = \{\{wer\}, \{ab\}, \{a\}, \{b\}\}, \quad d = 3$

Beispiel

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

$A = \{\{wer\}, \{ab\}, \{a\}, \{b\}\}, \quad d = 3$

Nach $j = 2$ -ten Zeichen einordnen

queue[0] = {ab}, {a}, {b}

queue[1] = {}

...

queue[18] = {wer} // r = 18

Beispiel

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

$A = \{\{wer\}, \{ab\}, \{a\}, \{b\}\}, \quad d = 3$

Nach $j = 2$ -ten Zeichen einordnen

queue[0] = {ab}, {a}, {b}

queue[1] = {}

...

queue[18] = {wer} // r = 18

Zurückkopieren

$A = \{\{ab\}, \{a\}, \{b\}, \{wer\}\}$

Beispiel

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

$A = \{\{ab\}, \{a\}, \{b\}, \{wer\}\}$

Beispiel

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

$A = \{\{ab\}, \{a\}, \{b\}, \{wer\}\}$

Nach $j = 1$ -ten Zeichen einordnen

queue[0] = {a}, {b}

queue[1] = {}

queue[2] = {ab} //b = 2

queue[3] = {}

queue[4] = {}

queue[5] = {wer} //e = 5

...

Beispiel

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

$A = \{\{ab\}, \{a\}, \{b\}, \{wer\}\}$

Nach $j = 1$ -ten Zeichen einordnen

queue[0] = {a}, {b}

queue[1] = {}

queue[2] = {ab} //b = 2

queue[3] = {}

queue[4] = {}

queue[5] = {wer} //e = 5

...

Zurückkopieren

$A = \{\{a\}, \{b\}, \{ab\}, \{wer\}\}$

Beispiel

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

$A = \{\{a\}, \{b\}, \{ab\}, \{wer\}\}$

Nach $j = 0$ -ten Zeichen einordnen

queue[0] = {}

queue[1] = {a}, {ab} //a = 1

queue[2] = {b} //b = 2

queue[3] = {}

...

queue[23] = {wer} //w = 23

...

Beispiel

```
public static void radixSort(char[][] A, int d)
{
    // 27 queues for 26 characters plus 'empty' character
    LinkedList[] queues = new LinkedList[27];

    // for all positions from right to left
    for(int j=d-1; j>=0; j--)
    {
        // initialize empty queues
        for(int i=0; i<27; i++) queues[i] = new LinkedList();

        // place each character array in correct queue
        for(int i=0; i<A.length; i++)
        {
            if(j<A[i].length)
            {
                // characters 'a'-'z'
                queues[A[i][j]-'a'+1].addLast(A[i]);
            }
            else
            {
                // character array is shorter than current position.
                // place it in 'empty' queue. 'empty' queue is queue 0
                // to get lexicographically correct results, i.e., a<ab.
                queues[0].addLast(A[i]);
            }
        }

        // traverse queues
        int n = 0;
        for(int i=0; i<27; i++)
        {
            while(queues[i].size() > 0)
            {
                A[n] = (char[])queues[i].removeFirst();
                n++;
            }
        }
    }
}
```

$A = \{\{a\}, \{b\}, \{ab\}, \{wer\}\}$

Nach $j = 0$ -ten Zeichen einordnen

queue[0] = {}

queue[1] = {a}, {ab} //a = 1

queue[2] = {b} //b = 2

queue[3] = {}

...

queue[23] = {wer} //w = 23

...

Zurückkopieren

$A = \{\{a\}, \{ab\}, \{b\}, \{wer\}\}$

Fragen?

u^b

b
UNIVERSITÄT
BERN

