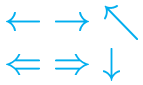
# Algorithmen und Datenstrukturen (Th. Ottmann und P. Widmayer)

## Folien: Spezielle Sortierverfahren

**Autor: Sven Schuierer**

Institut für Informatik
Georges-Köhler-Allee
Albert-Ludwigs-Universität Freiburg

# 1 Überblick

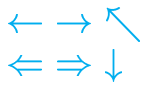


## Überblick

## Radixsort

## Forward-Radixsort

# 2 Radixsort



Nutzt Zahldarstellung der Schlüssel anstelle von Vergleichen

Schlüssel $k$ = Wort über Alphabet mit $m$ Elementen

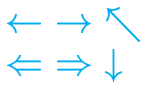
$$k = (k_l, k_{l-1}, \ldots, k_1, k_0)_m = \sum_{i=0}^{l} k_i \, m^i$$

$$k_i \in \{0, \ldots, m-1\}$$

$m = 10$ Dezimalzahlen
$m = 2$ Binärzahlen
$m = 26$ Zeichenketten über $\{a, \ldots, z\}$

Funktion: $z_m(k, i) = k_i$

# Radix-exchange-sort



Sortieren durch rekursive Aufteilung nach höchstwertigen Ziffern (Bits)

Radix-exchange-sort

Input: Folge $F$ von binären Schlüsseln mit Bits an Positionen $0, \ldots, l$, Bitposition $b \leq l$

Output: Nach Bitpositionen $\leq b$ sortierte Folge $F$

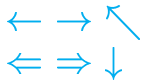
```
1 if |F| = 1 or b < 0
2   then return F
3   else /* Sortiere F nach Bit b (Quicksort) */
4        F_0 = {k ∈ F | z_2(b, k) = 0}
5        F_1 = {k ∈ F | z_2(b, k) = 1}
6        return Radix-exchange-sort(F_0, b − 1) +
7               Radix-exchange-sort(F_1, b − 1)
```

Beispiel: $F = (6, 7, 4, 2, 3)$
$= (110, 111, 100, 010, 011)$

| | 110 | 111 | 100 | 010 | 011 |
|---|---|---|---|---|---|
| $b = 2$ | 011 | 010 | 100 | 111 | 110 |
| $b = 1$ | 011 | 010 | 100 | 111 | 110 |
| $b = 0$ | 010 | 011 | | 110 | 111 |

Problem: Informationsgewinn pro Operation nur ein Bit

# Radix-exchange-sort



Definition

Die unterscheidenen Präfixe einer Menge $\{a_1, \ldots, a_n\}$ von Zeichenketten sind die kürzesten Präfixe von $\{a_1, \ldots, a_n\}$, die paarweise verschieden sind.

Der unterscheidende Präfix einer Zeichenkette $a_i$, die Präfix einer anderen Zeichenkette ist, ist $a_i$ selbst.

Beispiel:

SOCKEL
SOCRATES
SODA
SOFORT
SOFA
SORT
SORTIEREN

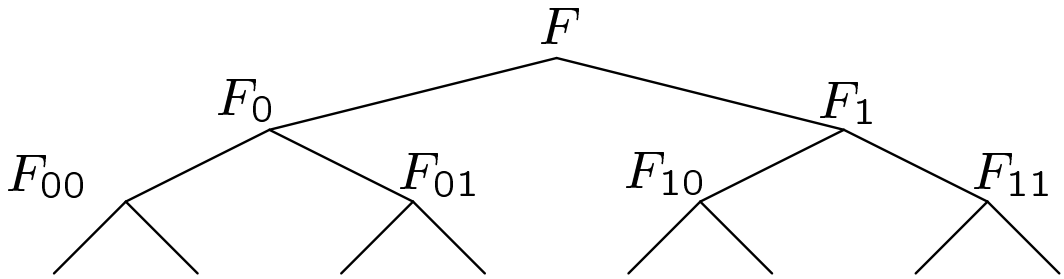
$s_i =$ unterscheidender Präfix von $a_i$

$$S = \sum_{i=1}^{n} |s_i|$$

# Analyse von Radix-exchange-sort



## Aufteilung bei Radix-exchange-sort:




## Laufzeit von Radix-exchange-sort:

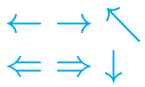
- konstanter Aufwand pro Aufteilungsschritt für $a_i$ (Bit $b$ testen, tauschen)
- # Aufteilungsschritte für $a_i = |s_i|$

$$T_{RES}(a_1, \ldots, a_n) = \sum_{i=1}^{n} |s_i| = O(n + S)$$

Anders:

$$\begin{aligned} & T(a_1, \ldots, a_n) \\ &= |\{(a_i, A) \mid a_i \text{ beteiligt an Aufteilungsschritt } A\}| \\ &= \sum_{i=1}^{n} |\{A \mid a_i \text{ beteiligt an Aufteilungsschritt } A\}| \end{aligned}$$

# Sortieren durch Fachverteilung



Idee: zuerst nach dem letzten Zeichen sortieren

Beispiel:

$F = 434, 528, 154, 176, 783, 204, 351, 218, 900$

Verteilen nach Ziffer 3:

| $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ |
|---|---|---|---|---|---|---|---|---|---|
| 900 | 351 | | 783 | 204<br>154<br>434 | | 176 | | 218<br>528 | |

Sammeln: 900, 351, 783, 434, 154, 204, 176, 528, 218

Verteilen nach Ziffer 2:

| $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ |
|---|---|---|---|---|---|---|---|---|---|
| 204<br>900 | 218 | 528 | 434 | | 154<br>351 | | 176 | 783 | |

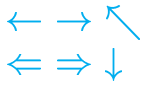
Sammeln: 900, 204, 218, 528, 434, 351, 154, 176, 783

Verteilen nach Ziffer 1:

| $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ |
|---|---|---|---|---|---|---|---|---|---|
| | 176<br>154 | 218<br>204 | 351 | 434 | 528 | | 783 | | 900 |

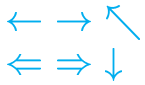
Sammeln: 154, 176, 204, 218, 351, 434, 528, 783, 900

# Sortieren durch Fachverteilung



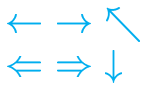
```
interface Decomposable {
  public int range ();
  /* Wertebereich der Zeichen: 0 .. range - 1 */

  int zeichen (int i);
  /* Zeichen an Stelle i im Bereich 0 .. range - 1;
     -1, falls das Objekt kein Zeichen an Stelle i
     besitzt */

  public abstract int laenge ();
  /* Anzahl der Zeichen des Objektes */
} // interface Decomposable


/* (Backward) Radix Sort */
class RadixSort {
  static void sort (Decomposable A[]){
     int n = A.length-1;
     int m = A[0].range();
     list fach [] = new list [m+1];

     /* Initialisiere die Faecher */
     for (int i = 0; i <= m; i++) {
       fach[i] = new list ();
     }

     /* Berechne die maximale Laenge einer
        Zeichenkette */
     int max = 1;
     for (int i = 2; i <= n; i++) {
       if (A[i].laenge() > A[max].laenge()) max = i;
     }
```

# Sortieren durch Fachverteilung



```
class RadixSort {
  static void sort (Decomposable A[]){

      ⋮

      /* Sortiere die Zeichenketten */
      for (int z = A[max].laenge () - 1; z >= 0; z--)
        {

        /* Verteile A[1] .. A[n] nach Zeichen z auf
           fach[1] .. fach[m]; fach[0] enthaelt
           Zeichenketten, die kein z-tes Zeichen
           haben */
        for (int i = 1; i <= n; i++) {
          fach [A[i].zeichen (z) + 1].append (A[i]);
        }

        /* Sammle die Zeichen wieder ein */
        int i = 1;
        for (int j = 0; j <= m; j++) {
          while (! fach[j].isEmpty ()) {
            A[i++] = fach[j].removeFirst ();
          }
        }
      }
  }
} // class RadixSort
```

# Sortieren durch Fachverteilung—Analyse



$l$ = Länge der Zeichenketten in Bits

Anzahl Durchläufe: $l/\log m$

Aufwand pro Durchlauf: $O(n+m)$

$\Rightarrow$ Gesamtaufwand: $O((n+m)\,l/\log m)$

Speicherplatz: $O(n+m)$

Vergleich:

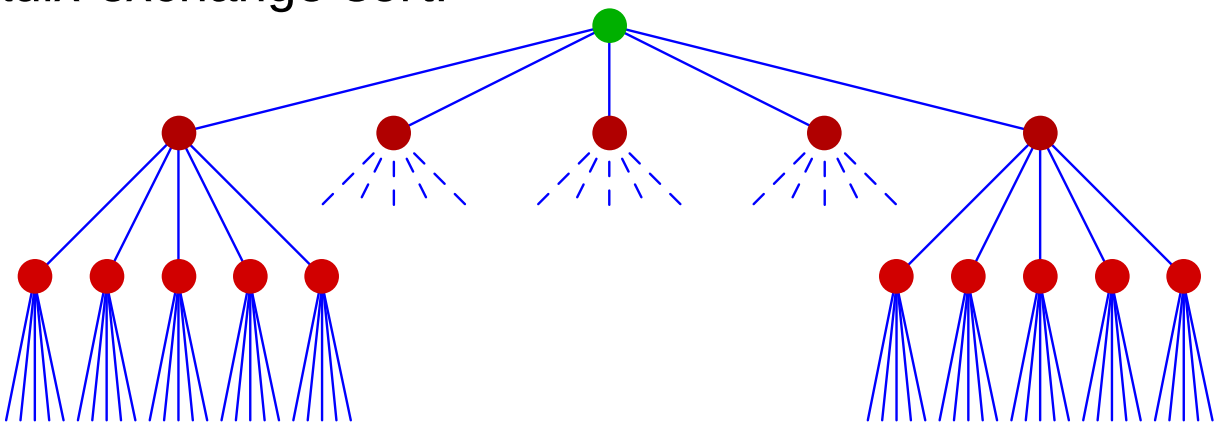
| | Operationen |
|---|---|
| Radix-exchange-sort | $O(n+S)$ |
| Fachverteilung | $O((n+m)\,l/\log m)$ |

$w$ = Maschinenwortlänge

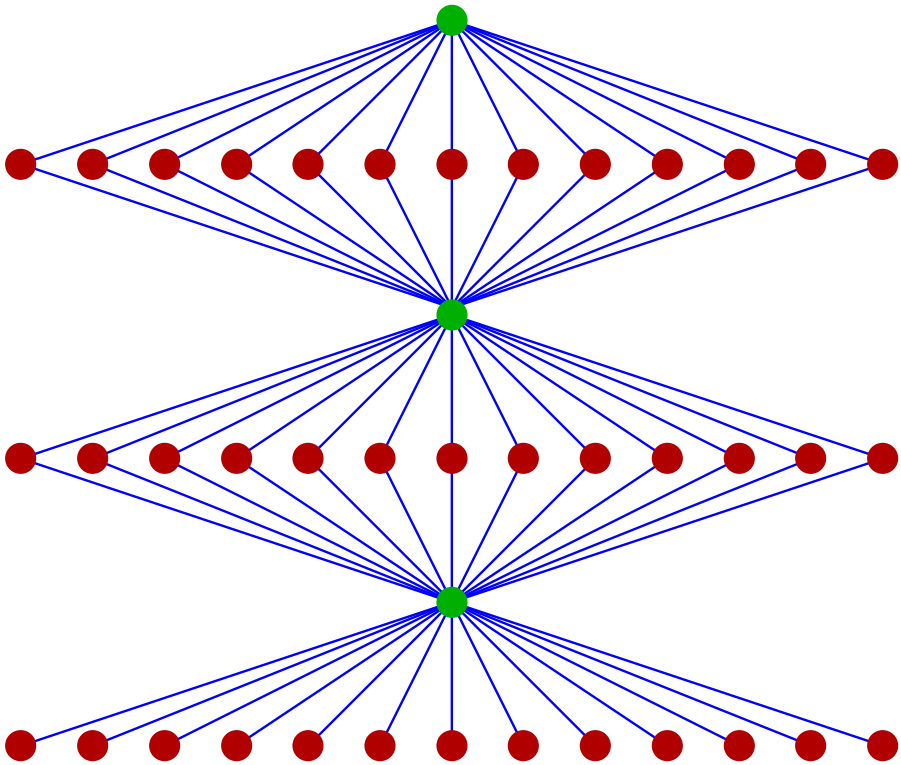
# Vergleich der Aufteilungen



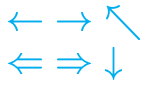
Radix-exchange-sort:



Sortieren durch Fachverteilung:



# 3 Forward-Radixsort



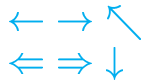
Idee:

- Verwendung von $m$ Fächern
- Fachverteilung nach höchstwertigen Ziffern
- Verteilung aller Schlüssel in einem Schritt
- Verwendung von Gruppen

Invariante: Nach dem $i$-ten Durchlauf sind die Elemente nach den ersten $i$ Zeichen sortiert

Gruppe $G$: aufeinanderfolgende Elemente, bei denen die ersten $i$ Zeichen gleich sind
fertig, falls $|G| = 1$
unfertig, falls $|G| \geq 2$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 120 | 125 | 168 | 163 | 165 | 309 | 307 | 352 | 412 | 410 |



## Forward-Radixsort

Input: Folge $A = (a_1, \ldots, a_n)$ von Elementen

Output: Sortierte Folge $F$

1 Gruppe[1] = $A$ /* beginne mit allen Elementen */

2 $i = 1$ /* beginne mit 1. Zeichen */

3 while es gibt eine unfertige Gruppe do

4 Durchlaufe unfertige Gruppen in aufsteigender Reihenfolge

5 Verteile Elemente nach dem $i$-ten Zeichen (mit Gruppennummer) auf die $m$ Fächer

6 Durchlaufe Fächer in aufsteigender Reihenfolge und sammele Elemente in ihren Gruppen

7 Durchlaufe die Gruppen:

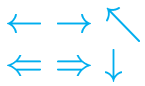
8 if $(z_m(a_j, i) \neq z_m(a_{j-1}, i))$ then

9 beginne neue Gruppe mit Nummer $j$

10 update Zeiger auf nächste unfertige Gruppe

11 $i = i + 1$ /* nächstes Zeichen */

# Forward-Radixsort



Beispiel: $F = 434, 528, 154, 176, 783, 204, 351, 218$

Verteilen nach Ziffer 1:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 176 | 218 | | | | | | | |
| | 154 | 204 | 351 | 434 | 528 | | 783 | | 900 |
| $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ |

Sammeln:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 154 | 176 | 204 | 218 | 351 | 434 | 528 | 783 | 900 |
| 1 | 1 | 3 | 3 | 5 | 6 | 7 | 8 | 9 |

Verteilen nach Ziffer 2:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 204(3) | 218(3) | | | | 154(1) | | 176(1) |
| $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ |

Sammeln:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 154 | 176 | 204 | 218 | 351 | 434 | 528 | 783 | 900 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# Analyse

$$S_{max} = \max\{|s_i| \mid 1 \leq i \leq n\}$$

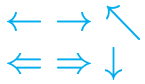
Laufzeit von Forward-Radixsort:

- konstanter Aufwand pro Aufteilungsschritt für $a_i$:
  — verteilen nach Ziffer $i$
  — sammeln nach Gruppennummer
  — Gruppennummer ändern
- # Iterationen für $a_i$: $|s_i|$

- # Iterationen ingesamt: $S_{max}/\log m$
- Aufwand pro Iteration: $m$

$$\begin{aligned} T_{FR}(a_1, \ldots, a_n) &= \sum_{i=1}^{n} |s_i|/\log m + m\, S_{max}/\log m \\ &= O(n + S/\log m + m\, S_{max}/\log m) \end{aligned}$$

# Forward-Radixsort



Vergleich:

| | Operationen |
|---|---|
| Radix-exchange-sort | $O(n + m \cdot S/\log m)$ |
| Fachverteilung | $O(n \cdot l/\log m + m \cdot l/\log m)$ |
| Forward Radixsort | $O(n + S/\log m + m \cdot S_{max}/\log m)$ |
| Quicksort | $O(n \log n\,(l/w))$ |

$w$ = Maschinenwortlänge, $S \leq n \cdot l$