CH08-320201

Algorithms and Data Structures

ADS

Lecture 10

Dr. Kinga Lipskoch

Spring 2019

Counting Sort: Problem Statement

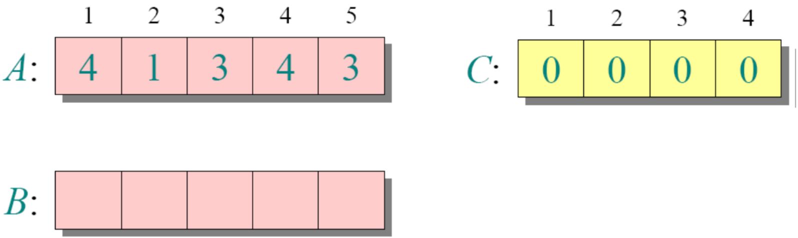
- Input: $A[1...n]$, where $A[j] \in \{1, 2, ..., k\}$.
- Output: $B[1...n]$, which is a sorted version of $A[1...n]$.
- Auxiliary storage: $C[1...k]$.

Counting Sort

```
for i := 1 to k do
  C[i] := 0
for j := 1 to n do
  C[A[j]] := C[A[j]] + 1
  // C[i] = |{key = i}|
for i := 2 to k do
  C[i] := C[i] + C[i - 1]
  // C[i] = |{key <= i }|
for j := n downto 1 do
  B[C[A[j]]] = A[j]
  C[A[j]] = C[A[j]] - 1
```

Counting Sort: Example (1)

Loop 1:



for $i \leftarrow 1$ **to** k
  do $C[i] \leftarrow 0$

Counting Sort: Example (2)

Loop 2:

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

B:					

for $j \leftarrow 1$ **to** n
  do $C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright\ C[i] = |\{\text{key} = i\}|$

Counting Sort: Example (3)

Loop 3:

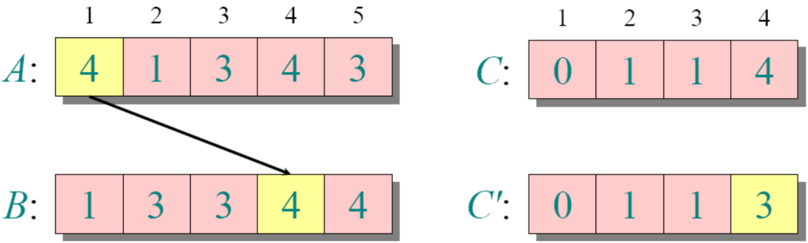
	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C':	1	1	3	5

for $i \leftarrow 2$ **to** k
  do $C[i] \leftarrow C[i] + C[i\text{–}1]$ $\triangleright$ $C[i] = |\{\text{key} \leq i\}|$

Counting Sort: Example (4)

Loop 4:



for $j \leftarrow n$ **downto** 1

  do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$

    $C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort: Asymptotic Analysis (1)

$\Theta(k)$

```
for i := 1 to k
    do C[i] := 0
```

$\Theta(n)$

```
for j := 1 to n
    do C[A[j]] := C[A[j]] + 1
```

$\Theta(k)$

```
for i := 2 to k
    do C[i] := C[i] + C[i-1]
```

$\Theta(n)$

```
for j := n downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] -1
```

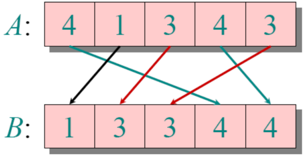
$\Theta(n + k)$

Counting Sort: Asymptotic Analysis (2)

- If $k = O(n)$, then Counting Sort takes $\Theta(n)$ time.
- Comparison sorting takes $\Omega(n \lg n)$ time.
- Counting Sort is not a comparison sort, not a single comparison between elements occurs.

Stable Sorting

- Definition:
  Stable sorting algorithms maintain the relative order of records with equal keys (i.e., values).
- Thus, a sorting algorithm is stable, if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list.
- Is Counting Sort stable?



Radix Sort: Motivation

- Counting Sort is less efficient when processing numbers from a large range, i.e., k is large.
- Can we find an algorithm that efficiently sorts n numbers for large k?

Radix Sort: History

- The 1880 U.S. census took almost 10 years to process.
- Herman Hollerith (1860-1929) prototyped a punched-card technology.
- His machines, including a "card sorter", allowed the 1890 census total to be reported in 6 weeks.
- He founded the Tabulating Machine Company in 1911, which merged with other companies in 1924 to form International Business Machines (IBM).

Radix Sort: Idea

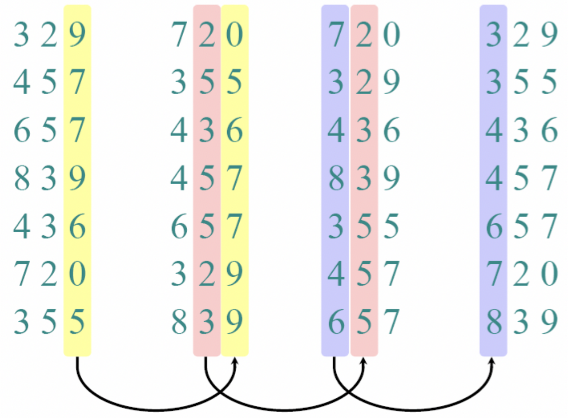
- Holleriths idea was to use a digit-by-digit sort.
- He sorted on most significant digit first.
- However, it requires us to keep one sequence for each digit, which then gets sorted recursively.
- It is more efficient to sort on least significant digit first.
- This idea requires a stable sorting algorithm.

Radix Sort: Pseudocode

RADIX-SORT(A, d)

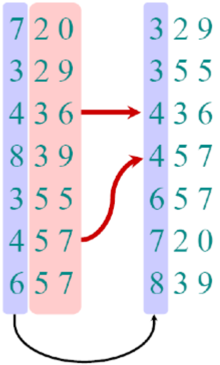
```
1  for i = 1 to d
2      use a stable sort to sort array A on digit i
```

Radix Sort: Example



Radix Sort: Correctness

- Induction on digit position:
- Only one digit: trivial.
- Assume that the numbers are sorted by their low-order $t-1$ digits.
- Sort on digit t:
  - Two numbers that differ in digit t are correctly sorted.
  - Two numbers equal in digit t are put in the same order as the input, i.e., correct order.



Radix Sort: Asymptotic Analysis

- Use Counting Sort as stable sorting algorithm.
- Sort n computer words of b bits each.
- Each word can be viewed as having b/r base-2^r digits.
- Example: 32-bit word
  - $r = 8$: $d = b/r = 4$ passes of counting sort on base-2^8 digits
  - $r = 16$: $d = b/r = 2$ passes of counting sort on base-2^{16} digits
- How many passes should we make?

Radix Sort: Choosing r (1)

- Counting Sort takes $\Theta(n+k)$ time to sort n numbers in the range from 0 to $k-1$.
- If each b-bit word is broken into r-bit pieces, each pass of Counting Sort takes $\Theta(n+2^r)$ time.
- Since there are b/r passes, we have:

$$T(n,b) = \Theta\left(\frac{b}{r}\left(n+2^r\right)\right)$$

- Choose r to minimize $T(n,b)$.

Radix Sort: Choosing r (2)

$$T(n,b) = \Theta\left(\frac{b}{r}\left(n + 2^r\right)\right)$$

- Increasing r means fewer passes, but when $r >> \lg n$ the time grows exponentially.
- We do not want $2^r > n$, but there is no harm asymptotically in choosing r as large as possible subject to this constraint.
- Choosing $r = \lg n$ implies $T(n,b) = \Theta(bn/\lg n)$.
- For numbers in the range from 0 to $n^d - 1$, we have $b = dr = d \lg n$, i.e., Radix Sort runs in $\Theta(dn)$ time.

Radix Sort: Conclusions

- In practice, Radix Sort is fast for large inputs, as well as simple to code and maintain.
- Example (32-bit numbers, i.e., $b = 32$, and $n = 2000$):
  - dn: At most $d = 3$ passes when sorting 2000 numbers.
  - $n \lg n$: Merge Sort and Quicksort do at least $\mathit{ceiling}(\lg 2000) = 11$ passes.