More on Sorting

CSCI2100 Tutorial 6
Shangqi Lu

Adapted from the slides of the previous offerings of the course
Counting Sort

• Sort a set of integers in a small domain \([1, U]\)

Initialize array \(B\)

| 7 | 2 | 6 | 4 | 8 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

\[\text{A} \quad \text{B}\]

Scan through \(A\)

| 7 | 2 | 6 | 4 | 8 | 9 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |

\[\text{A} \quad \text{B}\]

Clear A and scan through \(B\)

| 2 | 4 | 6 | 7 | 8 | 9 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |

\[\text{A} \quad \text{B}\]
Counting Sort

• Modify the counting sort to solve a variant of the previous problem
  Sort objects in a small domain based on integer keys
    • E.g., Sort a set of records in database by their keys
A Toy Problem: Sorting a Multi-Set

• Problem Input:
  • A multi-set $S$ of $n$ integers (each in the range $[1, U]$) is given in an array of length $n$
  • The values of $n$ and $U$ are inside two registers

• Goal:
  • Arrange the elements of $S$ in non-decreasing order
Example

- $B$ acts as counters instead of flags

Initialize array $B$

```
  7 2 6 2 9 7 1 2 0 0 0 0 0 0 0 0 0
```

Scan through $A$

```
  7 2 6 2 9 7 1 2 1 3 0 0 0 1 2 0 1
```

Clear $A$ and scan through $B$

```
  1 2 2 2 6 7 7 9 1 3 0 0 0 1 2 0 1
```
Sorting Objects (in A Small Domain)

• Problem Input:
  • A multi-set $S$ of $n$ objects in an array
  • Each object is a key-value pair, where the $1^{st}$ position gives the key, $2^{nd}$ position gives the value
  • All keys are in the range $[1, U]$
  • Some keys of objects may be identical
  • The values of $n$ and $U$ are inside two registers

• Goal:
  • Arrange the elements of $S$ in non-decreasing order by key
Example

• Consider a multi-set $S$
  $S = \{(9,1), (7,2), \{2,4\}, \{6,5\}, \{2,6\}, \{7,7\}, \{1,8\}, \{2,9\}\}$

• Initially we will have the following array

  $\begin{array}{cccccccccccc}
  k_1 & v_1 & k_2 & v_2 & k_3 & v_3 & k_4 & v_4 & k_5 & v_5 & k_6 & v_6 & k_7 & v_7 & k_8 & v_8 \\
  9 & 1 & 7 & 2 & 2 & 4 & 6 & 5 & 2 & 6 & 7 & 7 & 1 & 8 & 2 & 9 \\
  \end{array}$

• Rearrange the elements so that their keys are sorted:

  $\begin{array}{cccccccccccc}
  k_1 & v_1 & k_2 & v_2 & k_3 & v_3 & k_4 & v_4 & k_5 & v_5 & k_6 & v_6 & k_7 & v_7 & k_8 & v_8 \\
  1 & 8 & 2 & 9 & 2 & 6 & 2 & 4 & 6 & 5 & 7 & 7 & 7 & 2 & 9 & 1 \\
  \end{array}$
Example

• What if we solve this problem by using the counting sort algorithm on multi-set?

Compute $B$

Clear $A$ and scan through $B$

The values for those keys are lost
Sorting Objects (in A Small Domain)

• Need to modify the counting sort algorithm on multi-set in order to work for this problem
Example

Compute $B$

\[
\begin{array}{cccccccccccc}
9 & 1 & 7 & 2 & 2 & 4 & 6 & 5 & 2 & 6 & 7 & 7 & 1 & 8 & 2 & 9 \\
\end{array}
\]

Solve the problem by computing the cumulative sum of $B$

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
1 & 4 & 4 & 4 & 5 & 7 & 7 & 8 \\
\end{array}
\]

If $B[i] \neq 0$, $B'[i]$ indicates the last index of a particular key in the final sorted array.

\[
\begin{array}{cccccccccccc}
1 & 8 & 2 & 9 & 2 & 6 & 2 & 4 & 6 & 5 & 7 & 7 & 7 & 2 & 9 & 1 \\
\end{array}
\]

The final sorted array
Example

Build up a new array $A'$ by repeating the following: for a key-value pair $(k, v)$ in $A$, copy it to the $B'[k]$-th position in $A'$

Decrement the value in $B'$ to ensure that it always point to a valid, empty position in $A'$
Example

The second iteration

9 1 7 2 2 4 6 5 2 6 7 7 1 8 2 9 1 4 4 4 4 5 7 7 7

A

B'

A'

9 1 7 2 2 4 6 5 2 6 7 7 1 8 2 9 1 4 4 4 4 5 6 7 7

A

B'

A'

7 2 9 1
Example

The third iteration

\[
\begin{array}{cccccccccccc}
9 & 1 & 7 & 2 & 2 & 2 & 4 & 6 & 5 & 2 & 6 & 7 & 7 & 1 & 8 & 2 & 9 \\
1 & 4 & 4 & 4 & 4 & 4 & 5 & 6 & 7 & 7 & 1 & 8 & 2 & 9 & 1 & 3 & 4 & 4 & 4 & 5 & 6 & 7 & 7 \\
\end{array}
\]
Example

The fourth iteration
Example

The fifth iteration

\[
\begin{array}{ccccccccccc}
9 & 1 & 7 & 2 & 2 & 4 & 6 & 5 & 2 & 6 & 7 & 7 & 1 & 8 & 2 & 9 \\
\end{array}
\]

\[
\begin{array}{ccccccccccc}
1 & 3 & 4 & 4 & 4 & 4 & 6 & 7 & 7 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
\end{array}
\]

\[
\begin{array}{ccccccccccc}
A & \quad & \quad & \quad & \quad & \quad & \quad & \quad & \quad & \quad & \quad & \quad & A'
\end{array}
\]

\[
\begin{array}{ccccccccccc}
A' & \quad & \quad & \quad & \quad & \quad & \quad & \quad & \quad & \quad & \quad & \quad & A'
\end{array}
\]
Example

The sixth and seventh iterations

```
9 1 7 2 2 4 6 5 2 6 7 7 1 8 2 9
```

```
1 2 4 4 4 4 4 4 6 7 7
```

```
A
```

```
B'
```

```
2 6 2 4 6 5 7 7 7 2 9 1
```

```
A'
```

```
1 8 2 6 2 4 6 5 7 7 7 2 9 1
```

```
A'
```

```
1 8 2 6 2 4 6 5 7 7 5 7 7 2 9 1
```

```
B'
```

```
1 2 3 4 5 6 7 8 9
```

```
1 2 4 4 4 4 4 4 5 7 7
```

```
A
```

```
B'
```

```
A'
```
Example

The eighth iteration

\[9 \ 1 \ 7 \ 2 \ 2 \ 4 \ 6 \ 5 \ 2 \ 6 \ 7 \ 7 \ 1 \ 8 \ 2 \ 9\]
\[0 \ 2 \ 4 \ 4 \ 4 \ 4 \ 5 \ 7 \ 7\]

\[1 \ 8 \ 2 \ 9 \ 2 \ 6 \ 2 \ 4 \ 6 \ 5 \ 7 \ 7 \ 7 \ 2 \ 9 \ 1\]
Algorithm 2

• Step 1 and 2:
  • Same as algorithm 1

• Step 3:
  • Compute the cumulative sum $B'$ of $B$

• Step 4
  • Create a new array $A'$.
  • For each pair $(k, v)$ in $A$
    • Copy it to the $B'[k]$-th position in $A'$
    • Decrease $B'[k]$ by 1
Time Complexity

• Step 1 and 2: Initializing B and scanning through A to compute B takes \( O(U + n) \) time
• Step 3: computing the cumulative sum \( B' \) takes \( O(U) \) time
• Step 4: scanning A and using \( B' \) to copy elements over into \( A' \) takes \( O(n) \) time
• Overall time complexity: \( O(n + U) \)
A Bonus Problem: Sorting Arbitrary Objects

• Problem Input:
  • A multi-set $S$ of $n$ objects in an array
  • Each object is a key-value pair, where the 1\textsuperscript{st} position gives the key, 2\textsuperscript{nd} position gives the value
  • The values of the keys can be very large

• Goal:
  • Arrange the elements of $S$ in non-decreasing order by key
Solution

- Apply merge sort to sort $S$
- Treat merge sort as a black box
- Replace the comparator of the merge sort

![Diagram](chart.png)

- $B$ is input to merge sort.
- The order between $e_1, e_2$ determines how merge sort processes.
- Our Comparator is used to compare $e_1, e_2$.
- Sorted $B$ is the output after sorting.

The order between $e_1, e_2$ determines how our Comparator works in the context of the algorithm.
Solution

• Our comparator compare two objects $e_1 = (k_1, v_1)$ and $e_2 = (k_2, v_2)$ as follows
  • If $k_1 < k_2$, then rule $e_1 < e_2$
  • If $k_1 > k_2$, then rule $e_1 > e_2$
  • If $k_1 = k_2$:
    • We can either rule $e_1 < e_2$ or $e_1 > e_2$
When to Call Our Comparator

• Remember we only do comparisons in merge operation
• For example:

\[
\begin{array}{cccc}
  i &  & j \\
  1 & 8 & 3 & 9 \\
  2 & 6 & 4 & 4 \\
\end{array}
\]
Time Complexity

• Merge sort takes $O(n \log n)$ times comparisons
• Cost of calling the comparator: $O(1)$
• Overall time complexity: $O(n \log n)$