ECE430.217 Data Structures

Radix sort

Byoungyoung Lee
https://compsec.snu.ac.kr
byoungyoung@snu.ac.kr

Lecture slides were prepared based on Douglas W. Harder's notes (**dwharder@alumni.uwaterloo.ca).**

Outline

This topic covers **radix sort**:

- **Bucket sort** is $\Theta(n + m)$
- Is it possible to capitalize on the linear behaviour?
- Consider sorting digits

Motivation: Problem of Bucket Sort

Suppose we want to sort 10 digit numbers with repetitions

- We could use bucket sort, but this would require the use of 10^{10} buckets
- With one byte per counter, this would require 9 GiB

This may not be very practical…

Radix Sort: Idea

Consider the following scheme

- Given the numbers

  16 31 99 59 27 90 10 26 21 60 18 57 17

- If we first sort the numbers based on their last digit only, we get:

  90 10 60 31 21 16 26 27 57 17 18 99 59

- Now sort according to the first digit:

  10 16 17 18 21 26 27 31 57 59 60 90 99

- The resulting sequence of numbers is a sorted list

Radix Sort: Algorithm

Algorithm of Radix Sort:

- Suppose we are sorting decimal numbers
- Create an array of 10 queues
- For each digit, starting with the least significant
    - Place the i-th number into the bin corresponding to the current digit
    - Remove all numbers from the bins in the order of bins

Radix Sort

Suppose that two n-digit numbers are equal for the first m digits:

$$a = a_n a_{n-1} a_{n-2} \cdots a_{n-m+1} \boldsymbol{a_{n-m}} \cdots a_1 a_0$$

$$b = a_n a_{n-1} a_{n-2} \cdots a_{n-m+1} \boldsymbol{b_{n-m}} \cdots b_1 b_0$$

where $\boldsymbol{a_{n-m}} < \boldsymbol{b_{n-m}}$

For example, 103574 < 103892 because 1 = 1, 0 = 0, 3 = 3 but 5 < 8

Then, on iteration $n - m$, a will be placed in a lower bin than b

When they are taken out, a will precede b in the list

For all subsequent iterations, a and b will be placed in the same bin, and will therefore continue to be taken out in the same order

Therefore, in the final list, a must precede b

Example 1: Sorting decimal numbers

Sort the following decimal numbers:

86 198 466 709 973 981 374 766 473 342

Note that we will interpret 86 as 086

Example 1

Next, create an array of 10 queues:

0				
1				
2				
3				
4				
5				
6				
7				
8				
9				

Example 1: 3rd digit

Push according to the 3rd digit:

086 198 466 709 973 981 374 766 473 342

0				
1	98**1**			
2	34**2**			
3	97**3**	47**3**		
4	37**4**			
5				
6	08**6**	46**6**	76**6**	
7				
8	19**8**			
9	70**9**			

and dequeue: 98**1** 34**2** 97**3** 47**3** 37**4** 08**6** 46**6** 76**6** 19**8** 70**9**

Example 1: 2^{nd} digit

Enqueue according to the 2^{nd} digit:

981 342 973 473 374 086 466 766 198 709

0	7**0**9			
1				
2				
3				
4	3**4**2			
5				
6	4**6**6	7**6**6		
7	9**7**3	4**7**3	3**7**4	
8	9**8**1	0**8**6		
9	1**9**8			

and dequeue: 7**0**9 3**4**2 4**6**6 7**6**6 9**7**3 4**7**3 3**7**4 9**8**1 0**8**6 1**9**8

Example 1: 1st digit

Enqueue according to the 1st digit:

709 342 466 766 973 473 374 981 086 198

0	086			
1	198			
2				
3	342	374		
4	466	473		
5				
6				
7	709	766		
8				
9	973	981		

and dequeue: **0**86 **1**98 **3**42 **3**74 **4**66 **4**73 **7**09 **7**66 **9**73 **9**81

Example 1

The numbers are now sorted:

086 198 342 374 466 473 709 766 973 981

The next example uses the binary representation of numbers, which is even easier to follow

Example 2: Sorting binary numbers

Sort the following base 2 numbers:

`1111 11011 11001 10000 11010 101 11100 111 1011 10101`

First, interpret each as a 5-bit number:

`01111  11011  11001  10000  11010  00101  11100  00111  01011  10101`

Next, create an array of two queues:

0								
1								

Example 2

Place the numbers

`01111 11011 11001 10000 11010 00101 11100 00111 01011 10101`

into the queues based on the 5th bit:

0	10000	11010	11100					
1	01111	11011	11001	00101	00111	01011	10101	

Remove them in order:

`10000 11010 11100 01111 11011 11001 00101 00111 01011 10101`

Example 2

Place the numbers

10000 11010 11100 01111 11011 11001 00101 00111 01011 10101

into the queues based on the 4th bit:

0	10000	11100	11001	00101	10101			
1	11010	01111	11011	00111	01011			

Remove them in order:

10000 11100 11001 00101 10101 11010 01111 11011 00111 01011

Example 2

Place the numbers

10000 11100 11001 00101 10101 11010 01111 11011 00111 01011

into the queues based on the 3rd bit:

0	10000	11001	11010	11011	01011			
1	11100	00101	10101	01111	00111			

Remove them in order:

10000 11001 11010 11011 01011 11100 00101 10101 01111 00111

Example 2

Place the numbers

10000 11001 11010 11011 01011 11100 00101 10101 01111 00111

into the queues based on the 2nd bit:

0	10000	00101	10101	00111				
1	11001	11010	11011	01011	11100	01111		

Remove them in order:

10000 00101 10101 00111 11001 11010 11011 01011 11100 01111

Example 2

Place the numbers

`10000 00101 10101 00111 11001 11010 11011 01011 11100 01111`

into the queues based on the 1st bit:

0	00101	00111	01011	01111				
1	10000	10101	11001	11010	11011	11100		

Remove them in order:

`00101 00111 01011 01111 10000 10101 11001 11010 11011 11100`

Example 2

The numbers are now sorted:

```
00101  00111  01011  01111  10000  10101  11001  11010  11011  11100
```

This required $5n$ enqueues and dequeues

– In this case, it $n = 10$

Run-time analysis

Recall: Bucket sort is $\Theta(n + m)$ where m is the number of buckets

- In fact, this m corresponds to the maximum number

In radix sort,

- the number of iterations would be:
  - $\lfloor \log_{10}(m) \rfloor + 1$ digits for decimal numbers
  - $\lfloor \log_2(m) \rfloor + 1$ bits for binary numbers
- each iteration takes $\Theta(n)$

Run time of radix sort is therefore $\Theta(n \ln(m))$

- For this to be faster than previous sorting algorithms, it must be true that $\ln(m) < \ln(n)$ or $m << n$
- Therefore, it is only truly useful if we are sorting lists of relatively small range of numbers

Run-time analysis

The following table summarizes the run-times of bucket sort for sorting n numbers in the range $0, \ldots, m-1$

Case	Run Time	Comments
Worst	$\boldsymbol{\Theta}(n \ln(m))$	No worst case
Average	$\boldsymbol{\Theta}(n \ln(m))$	
Best	$\boldsymbol{\Theta}(n \ln(m))$	No best case

It requires $\Theta(n)$ memory for the queues

Summary

Radix sort uses bucket sort on each digit of a set of numbers

- Interesting in theory, less useful in practice
- Useful only if sorting numbers with significant duplication
- The idea is used elsewhere

References

[1] Donald E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd Ed., Addison Wesley, 1998, §5.2.3, p.144-8.

[2] Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, McGraw Hill, 1990, Ch. 7, p.140-9.

[3] Weiss, *Data Structures and Algorithm Analysis in C*++, *3rd Ed.*, Addison Wesley, §7.5, p.270-4.

References

Wikipedia, http://en.wikipedia.org/wiki/Radix_sort

[1] Donald E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd Ed., Addison Wesley, 1998, §5.2.3, p.144-8.

[2] Cormen, Leiserson, and Rivest, *Introduction to Algorithms*, McGraw Hill, 1990, Ch. 7, p.140-9.

[3] Weiss, *Data Structures and Algorithm Analysis in C++*, *3rd Ed.*, Addison Wesley, §7.5, p.270-4.