Handout Bucket Sort

Group members: Cuiting Huang, Zhe Cheng, Hongyang Ye

Authors affiliation: Northeastern University

Country location: China

E-mail address: huang.cui@northeastern.edu, cheng.zhe1@northeastern.edu, ye.ho@northeastern.edu

I Key Points

A. The sorting algorithm comparing two elements can be categorized as comparison sort.

B. Linear sort can be faster than comparison sort, with $O(n)$.

C. The critical assumption of bucket sort is:

The data is as evenly distributed as possible.

D. Designing the mapping function of bucket sort is of vital importance.

E. The lower bound of comparison sort is $O(nlog(n))$*. Linear sort can beat this bound because it does not compare.*

F. The limits of bucket sort are:

Can perform badly if all the inputs fall into one bucket.

Extra space for buckets is needed.

II Summary of comparison sorts

	Average Time Complexity	Best Case	Worst Case	Space Complexity
Insertion sort	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$
Bubble sort	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge sort	$O(nlog(n))$	$O(nlog(n))$	$O(nlog(n))$	$O(n)$

III Bucket sort

To guarantee the time complexity $O(n)$ of bucket sort

A. Assumption of bucket sort

The data set is generated by a random process, which distributes elements independently over the interval.

Only the elements are as evenly distributed as possible, the time complexity of bucket sort would be $O(n)$.

B. Basic Idea

What's bucket? A bucket is a container with a certain volume, bucket represents an interval.

1) First step: Creating buckets and determining the range of each bucket. There are many different ways to determine the number of buckets. One rule we can take into account is that it is best to have elements evenly distributed in each bucket. Here we will introduce two buckets splitting principle.

a) Simple Bucket Splitting:

$$\mathrm{bucketNum} = \max/\mathrm{k} - \min/\mathrm{k} + 1$$

$$\mathrm{f(n)} = (\mathrm{int})\ (\mathrm{array[i]} - \min) * 1/\mathrm{k}$$

b) Reduced Bucket Splitting: If the range of the data set is large, with using simple buckets splitting, bucketNum will be extremely overflowing, there will be a bunch of empty buckets in the middle.

$$\mathrm{bucketNum} = (\max - \min)/(\max - \min) * \mathrm{array.length}$$

$$\mathrm{f(n)} = (\mathrm{array[i]} - \min)/(\max - \min) * \mathrm{array.length}$$

2) Second step: Traverse the data set and place the elements into each corresponding bucket. Here we will use a mapping function, each element will be mapped to the nth bucket. The mapping function the key of making our algorithm more efficient.

3) Third Step: Sort the elements inside each bucket separately. You can use whatever comparison sort you like.

4) Last Step: Traverse over all buckets which are not empty and put all elements back to the original data set in turn.

Notice that, bucket sort is the algorithm out-of-place, to avoid wasting space, the number of buckets should be just enough, neither overflowing nor too little.

C. Sorting character strings

In addition to the numerical array, bucket sort can be used dealing with String array. For example, there is a string array with lowercase letters uppercase letters and digits.

$$\text{Input data set D, a, F, 99, B, c, A, z}$$

All lowercase letters are required to precede uppercase letters, but no order is required between themself. Then put the digits behind the letters. By using bucket sort, we can separate upper and lower case letters and digits.

First, traverse the array, count the number of elements of three types.

$$\mathrm{lowerNum} = 3, \mathrm{upNum} = 4, \mathrm{digitNum} = 1$$

Next, calculate the starting index of each type.

$$\text{lowerIndex} = 0, \text{upIndex} = \text{lowerNum}, \text{digiIndex} = \text{lowerNum} + \text{upNum}$$

Then, according to the index interval, put elements into the corresponding bucket, the interval of each bucket is $[\text{Index}, \text{Index} + \text{Num} - 1]$

$\text{bucket lowercase}[0-2]$: a, v, c

$\text{bucket uppercase}[3-6]$: D, F, B, A

$\text{bucket digit}[7]$: 99

Last, write back to the original array. Done!

D. Bonus

Animation Bucket_sort https://youtu.be/vt1YX_ndHMk

Animation Bucket_sort for string array https://youtu.be/Ggf4NUe7bCg

IV Pseudo-code

A. How to implement one

1. $let\ B[0...n1]\ be\ a\ new\ array$
2. *n = A.length*
3. *for i = 0 to n – 1*
4. $make\ B[i]\ an\ empty\ list$
5. *for i = 0 to n – 1*
6. $insert\ A[i]\ into\ list\ B[nA[i]]$
7. *for i = 0 to n – 1*
8. $Sort\ list\ B[i]\ with\ insertion\ sort$
9. $Concatenate\ the\ lists\ B[0],\ B[1],\ ...,\ B[n-1]\ together\ in\ order$

B. Practices

There will be n/m elements in each bucket, where m represents number of buckets. Nonetheless, programmers often utilize length of input array as the number of buckets.

To completely avoid comparisons, we should find ways to have up to 1 element in 1 bucket.

C. Time Complexity

In an average case, the time complexity is like as such:

$$O(N) + O(M * (N/M) * \log(N/M))$$

$$= O(N) + O(N * \log(N/M))$$

If comparison avoidance is done perfectly, the time complexity is: $O(n)$

D. Space Complexity

Space complexity really depends on how the buckets are presented.

1)array

2)Singly linked list

In worst cases, we will have $O(N * M)$ space complexity.

Space complexity increases when

1)input array elements increases

2)Number of buckets increases

V Pros and cons

A. Pros of bucket sort

Can sort in linear time if the data set satisfies the assumption: the data set conforms to uniform distribution.

B. Cons of bucket sort

Can perform badly if all the inputs fall into one bucket, and requires extra space for buckets.

VI Real-world application

A. When to use bucket sort?

The data set has some specific features.

B. When to use comparison sort?

The data set does not satisfy the assumptions of linear sort.

We do not have much space for sorting.