Counting sort

Class	Sorting algorithm
Data structure	Array
Worst case performance	$O(n+k)$
Best case performance	$O(n+k)$
Average case performance	$O(n+k)$

Counting sort (sometimes referred to as **ultra sort** or **math sort**[1]) is a sorting algorithm which (like bucket sort) takes advantage of knowing the range of the numbers in the array to be sorted (array *A*). It uses this range to create an array *C* of this length. Each index *i* in array *C* is then used to count how many elements in *A* have the value *i*; then counts stored in *C* can then be used to put the elements in *A* into their right position in the resulting sorted array. The algorithm was created by Harold H. Seward in 1954.

Characteristics of counting sort

Counting sort is a stable sort and has a running time of $\Theta(n+k)$, where n and k are the lengths of the arrays *A* (the input array) and *C* (the counting array), respectively. In order for this algorithm to be efficient, k must not be much larger than n.

The indices of *C* must run from the minimum to the maximum value in *A* to be able to index *C* directly with the values of *A*. Otherwise, the values of *A* will need to be translated (shifted), so that the minimum value of *A* matches the smallest index of *C*. (Translation by subtracting the minimum value of *A* from each element to get an index into *C* therefore gives a counting sort. If a more complex function is used to relate values in *A* to indices into *C*, it is a bucket sort.) If the minimum and maximum values of *A* are not known, an initial pass of the data will be necessary to find these (this pass will take time $\Theta(n)$; see selection algorithm).

The length of the counting array *C* must be at least equal to the range of the numbers to be sorted (that is, the maximum value minus the minimum value plus 1). This makes counting sort impractical for large ranges in terms of time and memory needed. Counting sort may for example be the best algorithm for sorting numbers whose range is between 0 and 100, but it is probably unsuitable for sorting a list of names alphabetically. However, counting sort can be used with radix sort to sort a list of integers whose range is too large for counting sort to be suitable alone.

Because counting sort uses key values as indexes into an array, it is not a comparison sort, and the $\Omega(n \log n)$ lower-bound for sorting is inapplicable.

Counting sort is commonly used within other sorting algorithms such as radix sort, since radix sort processes by counting occurrence of digits of a limited number of bits, thus limiting the size of the count table.

Tally sort

A well-known variant of counting sort is **tally sort**, where the input is known to contain no duplicate elements, or where we wish to eliminate duplicates during sorting. In this case the count array can be represented as a bit array; a bit is set if that key value was observed in the input array. Tally sort is widely familiar because of its use in the book *Programming Pearls* as an example of an unconventional solution to a particular set of limitations.[2]

The algorithm

A summary of the algorithm is as follows.

1. Find the highest and lowest elements of the set
2. Count the different elements in the set. (E.g. Set[4,4,4,1,1] would give three 4's and two 1's)
3. Store zero 0's, two 1's, zero 2's, zero 3's, and three 4's into the destination set - i.e. store as many i-th elements as its corresponding count holds.

C implementation

```
#include <stdlib.h>
void counting_sort(int array[], int size)
{
  int i, min, max;

  min = max = array[0];
  for(i = 1; i < size; i++)
  {
    if (array[i] < min)
      min = array[i];
    else if (array[i] > max)
      max = array[i];
  }

  int range = max - min + 1;
  int *count =(int*)malloc(range * sizeof(int));

  for(i = 0; i < range; i++)
    count[i] = 0;

  for(i = 0; i < size; i++)
    count[ array[i] - min ]++;
  int j, z = 0;
  for(i = min; i <= max; i++)
    for(j = 0; j < count[ i - min ]; j++)
      array[z++] = i;

  free(count);
}
```

Single-pass variant

Given two preconditions, it is possible to merge the counting and writing passes, thus reducing the complexity from O(N) to exactly N reads and writes of the array elements.[3] First, the sorting problem must be relaxed to allow a non-contiguous (i.e. sparse) output array, which suffices for iterating over keys in sorted order. Second, the computer must support virtual memory, with physical memory mapped in response to page faults.

C pseudocode

```
void single_pass_counting_sort(int *array, int size, int range, int
**output, int *next)
{
  int key, i;
  *output = ReserveAddressSpace(range*size*sizeof(int));
  for(key = 0; key < range; key++)
    next[key] = key*size;
  for(i = 0; i < size; i++)
    (*output)[next[array[i]]++] = array[i];
}

void iterate_over_sorted(int size, int range, int **output, int *next)
{
  int key, i;
  for(key = 0; key < range; key++)
    for(i = key*size; i < next[key]; i++)
      do_work((*output)[i]);
}
```

Arrays of 8-bit and 16-bit Elements

Counting Sort can be simplified when used to sort arrays of 8-bit or 16-bit elements. For these data types the count array is of reasonable size for today's computers - e.g. 256 counts of 32-bit each for arrays of 8-bit elements on 32-bit operating systems. For these data types the count arrays do not have to be dynamically allocated.[4] Sorting arrays of 8-bit and 16-bit elements using Counting Sort, significantly outperforms other algorithms due to small constants of the algorithm and its sequential memory access when reading and writing. Unsigned and signed 8-bit and 16-bit elements must be handled differently, since indexing using negative numbers is not supported in languages such as C++.

Parallel Implementations

Counting Sort is amenable to parallel implementations.[5] [6] Using the parallel_reduce pattern, the input array can be split into sub-arrays, which are processed in parallel, each generating its own count array. These count arrays are then merged to produce a single count array.

Notes

[1] Anthony Ralston, Edwin D. Reilly, David Hemmendinger, ed (2003). "Ultrasort". *Encyclopedia of Computer Science* (4th ed.). Wiley. pp. 1660–1661. ISBN 0-470-86412-5.

[2] Chapter 1 of Jon Bentley's *Programming Pearls* ISBN 0-201-10331-1.

[3] J. Wassenberg, W. Middelmann, P. Sanders: An Efficient Parallel Algorithm for Graph-Based Image Segmentation. In: Computer Analysis of Images and Patterns, LNCS Vol. 5702 (September 2009)

[4] V.J. Duvanenko, "In-Place Hybrid N-bit-Radix Sort", Dr. Dobb's Journal, November 2009, p. 6 (http://www.drdobbs.com/architecture-and-design/221600153)

[5] V.J. Duvanenko, "Parallel Counting Sort" Dr. Dobb's Journal, April 2010 (http://www.drdobbs.com/architecture-and-design/224700144)

[6] V.J. Duvanenko, "Parallel Counting Sort (Part 2)" Dr. Dobb's Journal, July 2010 (http://www.drdobbs.com/high-performance-computing/225900071)

References

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 8.2: Counting sort, pp. 168–170.
- Donald Knuth. *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Second Edition. Addison-Wesley, 1998. ISBN 0-201-89685-0. Section 5.2, Sorting by counting, pp. 75–80.
- Seward, Harold H. *Information sorting in the application of electronic digital computers to business operations* Masters thesis. MIT 1954.
- NIST's Dictionary of Algorithms and Data Structures: counting sort (http://www.nist.gov/dads/HTML/countingsort.html)

External links

- Analyze Counting Sort in an online Javascript IDE (http://tide4javascript.com/?s=Counting)
- Demonstration applet from Cardiff University (http://users.cs.cf.ac.uk/C.L.Mumford/tristan/CountingSort.html)
- Counting sort (http://www.24bytes.com/Count-Sort.html)
- Counting Sort Simulation Java Applet (http://users.cs.cf.ac.uk/C.L.Mumford/tristan/CountingSort.html)
- Efficient Counting Sort in Haskell (http://kks.cabal.fi/laskentalajittelu/CountingSortHaskell)

Article Sources and Contributors

Counting sort *Source*: http://en.wikipedia.org/w/index.php?oldid=402597887 *Contributors*: A5b, Aenar, Agabrielson, Albuseer, Altenmann, Ashawley, Barbarab111, Bayard, Billylikeswikis, Bolero, Booyabazooka, BrokenSegue, C. A. Russell, CJLL Wright, Cdiggins, Cyde, Cyhawk, DanielKO, Dcoetzee, Ducker, Duvavic1, Ed Fitzgerald, Fredrik, Frencheigh, Gadfium, Gamma, Giftlite, GregorB, Gwern, Herry43113, Hiihammuk, Jan Wassenberg, Jiuguang Wang, Joshuaali, K.A.Sayed, Kevin, Kyellan, Linuxbabu, Mav, Merendoglu, Oli Filth, Oskar Sigvardsson, Oğuz Ergin, Pako, Pce3@ij.net, RadioFan, Razimantv, Saurabhc, Scullder, SiobhanHansa, Smyth, Snaxe920, Stemonitis, Swift, TEcHNOpl, TWiStErRob, Thegeneralguy, Timwi, Torsten Will, Zedla, Zippedmartin, Zundark, 103 anonymous edits

License