Jump to: navigation, search

An **american flag sort** is an efficient, in-place variant of radix sort that distributes items into hundreds of buckets. Non-comparative sorting algorithms such as radix sort and american flag sort are typically used to sort large objects such as strings, for which comparison is not a unit-time operation.[1]

American flag sort iterates through the bits of the objects, considering several bits of each object at a time. For each set of bits, american flag sort makes two passes through the array of objects: first to count the number of objects that will fall in each bin, and second to place each object in its bucket. This works especially well when sorting a byte at a time, using 256 buckets. With some optimizations, it is twice as fast as quicksort for large sets of strings.[1]

The name comes by analogy with the Dutch national flag problem in the last step: efficiently partition the array into many "stripes".

Contents

[hide]

\* 1 Algorithm

o 1.1 Pseudocode

\* 2 Sample implementation in Python

\* 3 See also

\* 4 External links

\* 5 References

[edit] Algorithm

Sorting algorithms in general sort a list of objects according to some ordering scheme. In contrast to comparison-based sorting algorithms, such as quicksort, american flag sort can only sort integers (or objects that can be interpreted as integers). In-place sorting algorithms, including american flag sort, run without allocating a significant amount of memory beyond that used by the original array. This is a significant advantage, both in memory savings and in time saved copying the array.

American flag sort works by successively dividing a list of objects into buckets based on the first digit of their base-N representation (the base used is referred to as the radix). When N is 2, each object can be swapped into the correct bucket by using the Dutch national flag algorithm. When N is larger, however, objects cannot be immediately swapped into place, because it is unknown where each bucket should begin and end. American flag sort gets around this problem by making two passes through the array. The first pass counts the number of objects that belong in each of the N buckets. The beginning and end of each bucket in the original array is then computed as the sum of sizes of preceding buckets. The second pass swaps each object into place.

American flag sort is most efficient with a radix that is a power of 2, because bit-shifting operations can be used instead of expensive logarithms to compute the value of each digit. When sorting strings, it is typical to use a radix of 256, which corresponds by sorting character-by-character.[1]

[edit] Pseudocode

american\_flag\_sort(Array, Radix)

for each digit D:

# first pass: compute counts

Counts <- zeros(Radix)

for object X in Array:

Counts[digit D of object X in base Radix] += 1

# compute bucket offsets

Offsets <- [ sum(Counts[0..i]) for i in 1..Radix]

# swap objects into place

for object X in Array:

swap X to the bucket starting at Offsets[digit D of X in base Radix]

for each Bucket:

american\_flag\_sort(Bucket, Radix)

[edit] Sample implementation in Python

This example written in the Python programming language will perform american flag sort for any radix of 2 or greater. Simplicity of exposition is chosen over clever programming, and so the log function is used instead of bit shifting techniques.

def get\_radix\_val(x, digit, radix):

return int(floor(x / radix\*\*digit)) % radix

def compute\_offsets(a\_list, start, end, digit, radix):

counts = [0 for \_ in range(radix)]

for i in range(start, end):

val = get\_radix\_val(a\_list[i], digit, radix)

counts[val] += 1

offsets = [0 for \_ in range(radix)]

sum = 0

for i in range(radix):

offsets[i] = sum

sum += counts[i]

return offsets

def swap(a\_list, offsets, start, end, digit, radix):

i = start

next\_free = copy(offsets)

cur\_block = 0

while cur\_block < radix-1:

if i >= offsets[cur\_block+1]:

cur\_block += 1

continue

radix\_val = get\_radix\_val(a\_list[i], digit, radix)

if radix\_val == cur\_block:

i += 1

continue

swap\_to = next\_free[radix\_val]

a\_list[i], a\_list[swap\_to] = a\_list[swap\_to], a\_list[i]

next\_free[radix\_val] += 1

def american\_flag\_sort\_helper(a\_list, start, end, digit, radix):

offsets = compute\_offsets(a\_list, start, end, digit, radix)

swap(a\_list, offsets, start, end, digit, radix)

if digit == 0:

return

for i in range(len(offsets)-1):

american\_flag\_sort\_helper(a\_list, offsets[i], offsets[i+1], digit-1, radix)

def american\_flag\_sort(a\_list, radix):

for x in a\_list:

assert(type(x) == int)

max\_val = max(a\_list)

max\_digit = int(floor(log(max\_val, radix)))

american\_flag\_sort\_helper(a\_list, 0, len(a\_list), max\_digit, radix)