CIS 575. Introduction to Algorithm Analysis
Material for February 14, 2024

Dutch National Flag Algorithm: the Solution

1 Top-Down Solution

Following our general guidelines, we may first attempt a top-down solution. For that purpose, given an array $A[1..n]$ we may do a case analysis on the color of $A[n]$, the last element:

Blue Easy: that element is where it should be; now just recursively rearrange $A[1..n-1]$.

Red A bit more involved, but not hard: swap $A[n]$ with $A[1]$; since $A[1]$ is now red, it is where it should be and we just need to recursively rearrange $A[2..n]$.

White This is the tricky case! We can only put that element at its proper spot *later*, and to remember that we need stack space that may be be linear in n, violating our requirement that the algorithm is *in-place*.

It is, however, possible to write a *tail*-recursive algorithm that solves the Dutch National Flag problem. As that algorithm may require a little effort to understand, let us instead directly look at an equivalent *iterative* algorithm.

2 Iterative Solution

Given an input array $A[1..n]$, and with r (w,b) the current count of red (white, blue) items, our goal is to maintain the **loop invariants**

1. $r + w + b \leq n$
2. for all i with $1 \leq i \leq r$ we know that $A[i]$ is red
3. for all i with $n - b < i \leq n$ we know that $A[i]$ is blue
4. for all i with $n - b - w < i \leq n - b$ we know that $A[i]$ is white

Graphically, what we know about A is:

$1 \quad r$	$r+1 \quad n-b-w$	$n-b-w+1 \quad n-b$	$n-b+1 \quad n$
Red	???	White	Blue

We can clearly *establish* the invariant by giving all of r,w,b the initial value 0 (as then the claims about certain elements being of a certain color are all *vacuously true*).

We will be done when the area containing "???" is empty, that is, when $r + w + b = n$.

To *maintain* the loop invariant, while making progress, we examine the last unknown element: with $k = n - b - w$, we do a case analysis on the color of $A[k]$:

Red swap it with $A[r + 1]$, and add 1 to r

Blue swap it with $A[n - b]$, and add 1 to b

White just add 1 to w.

We have developed the iterative algorithm

```
DutchFlagIter(A[1...n])
  r ← 0; w ← 0; b ← 0
  while r + w + b < n
    k ← n − b − w
    if A[k] is red
      A[k] ↔ A[r + 1]; r ← r + 1
    else if A[k] is blue
      A[k] ↔ A[n − b]; b ← b + 1
    else
      w ← w + 1
  return r,w,b
```

This algorithm is obviously **in-place** with running time **proportional to n**.

Example Consider an array with 6 elements, 2 of each color:

Blue1, White1, Red1, Red2, Blue2, White2

The algorithm will then behave as follows, with the items whose colors have already been detected in boldface, and the item currently examined in italics:

1	2	3	4	5	6	r	w	b
Blue1	White1	Red1	Red2	Blue2	*White2*	0	0	0
Blue1	White1	Red1	Red2	*Blue2*	**White2**	0	1	0
Blue1	White1	Red1	*Red2*	**White2**	**Blue2**	0	1	1
Red2	White1	Red1	*Blue1*	**White2**	**Blue2**	1	1	1
Red2	White1	*Red1*	**White2**	**Blue1**	**Blue2**	1	1	2
Red2	**Red1**	*White1*	**White2**	**Blue1**	**Blue2**	2	1	2
Red2	**Red1**	**White1**	**White2**	**Blue1**	**Blue2**	2	2	2

We see that even though Red1 was initially left of Red2, we end up with Red1 being to the right of Red2. This shows that the algorithm we have developed, despite its many other good qualities, is *not stable*; a "stable" algorithm would keep the order of "equivalent" items.