We can jiggle the parameters of the method so that a count of one results in a register value of one and then the random rounding procedure has no effect on the first count. The function

$$v = \log(n + 1)/\log(2)$$

has just the property we want. This formula is of course independent of the base of the logarithms.

A General Solution

The class of functions that I have used and analyzed are the functions

$$v(n) = \log(1 + n/a)/\log(1 + 1/a)$$

where the parameter a controls both the maximum count that can be held in the register and the expected error. The constant $\log(1 + 1/a)$ in the denominator serves only to force $n = 1$ to correspond to $v = 1$ so that the random procedures have no effect on the first count and counts of 0 and 1 are represented exactly. It is in this way that good relative accuracy is preserved for small counts. The maximum value n that can be represented using the parameter a can be calculated from the inverted formula

$$n_v = a((1 + 1/a)^v - 1).$$

The expected error in the estimated value of n after n counts can be calculated from the formula

$$\sigma^2 = n(n - 1)/2a.$$

This formula can be proved by induction on n.

Let us inspect the performance of this method using the parameter $a = 30$. The largest value that can be represented in an 8-bit counter is about 130,000. The standard deviation σ is approximately equal to $n/8$ which implies that the relative error is nearly independent of n and that 95 percent of the time the relative error will be less than 24 percent. Larger values of a will lead to smaller maximum counts and, of course, to smaller relative error.

There is no need to compute any of the logarithms or powers during the counting process. A table containing the 255 values of Δ can be precomputed by the formula $\Delta_j = (a/(a + 1))^j$ and accessed when a new count is made. The random number generator can be of the simplest sort and no great demands are made on its properties.

The distribution of errors is somewhat asymmetric for small counts, but as n becomes larger, the distribution closely approximates the normal distribution. In the example above where $a = 30$, the normal error curve gives a useful estimate of the error distribution for counts greater than about 20.

Received June 1975; revised December 1977

Programming Techniques
S.L. Graham, R.L. Rivest Editors

An Analysis of Algorithms for the Dutch National Flag Problem

Colin L. McMaster
University of California

Solutions to the Dutch National Flag Problem have been given by Dijkstra [1] and Meyer [3]. Dijkstra starts with a simple program and arrives at an improved program by refinement. Both of the algorithms given by Dijkstra are shown to have an expected number of swaps which is $\frac{2}{3}N + \theta(1)$ and that these values differ at most by $\frac{1}{3}$ of a swap and asymptotically by $\frac{1}{4}$ of a swap. The algorithm of Meyer is shown to have expected swap complexity $\frac{5}{9}N$.

Key Words and Phrases: algorithmic analysis, Dutch National Flag Problem, refinement, structured programming

CR Categories: 4.0, 5.24, 5.25, 5.3

Introduction

Dijkstra [1] has defined a problem which he calls the Problem of the Dutch National Flag. It may be stated as follows. There is a row of N buckets numbered from 1 to N. The buckets are arranged in numerical order with bucket 1 on the left and bucket N on the right. Each bucket contains exactly one pebble and each pebble is

Research sponsored by Joint Services Electronics Program Grant F 44620-76-C-0100 and National Science Foundation Grant MCS 76-15036.

Author's address: Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

colored either red, white or blue. The problem is to arrange the pebbles in the buckets so that

(i) all of the red pebbles are to the left of all of the white pebbles and all of the blue pebbles, and
(ii) all of the white pebbles are to the left of all of the blue pebbles.

A procedure to solve the problem must use the following primitive operations:

buck(i): a function which returns the color of the pebble in bucket *i*.
swap(i, j): a procedure which exchanges the pebble in bucket *i* with the pebble in bucket *j*.

In addition, any solution procedure must satisfy the following requirements:

(1) It must handle any of the 3^N possible initial configurations of pebbles ($N \geq 0$),
(2) It must use only a fixed number of variables each of which may not take on much more than N different values, and
(3) The buck operation is considered so time consuming that it must not be applied to the same pebble twice.

Dijkstra also specifies that "regarding programs of the same degree of complication, the one that needs (on the average) the fewer swaps is to be preferred" [1, p. 112].

Although this problem does not appear to have any important applications, it is nonetheless interesting for the following reasons. Because the problem is simple and easily understood, the reader is encouraged to develop a solution before proceeding and to observe the development process. Complete solutions in the form of programs can be presented and their efficiency analyzed. Furthermore, the following analysis shows that both the method and the results of the application of refinement can be deceptive.

Dijkstra imposes strong constraints upon the problem and outlines a simple argument which leads to the following "general situation" [1, p. 112]. There exist four types of buckets: buckets known to contain red pebbles, buckets known to contain white pebbles, buckets known to contain blue pebbles, and buckets containing uninspected pebbles. These are arranged in four contiguous regions and are kept track of by using the three variables r, w, and b with

$1 \leq k < r$: the pebble in the kth bucket is red,
$r \leq k \leq w$: the pebble in the kth bucket is uninspected,
$w < k \leq b$: the pebble in the kth bucket is white,
$b < k \leq N$: the pebble in the kth bucket is blue.

All of the solutions presented will be based on this observation.

This paper presents an analysis of the expected number of swaps required by three algorithms for this problem. Section 1 considers the problem when only two colors are involved and presents an algorithm which is shown to be optimal. Section 2 presents a simple solution to the three-color problem given by Dijkstra and analyzes its average performance. Section 3 presents Dijkstra's refined solution and an analysis of its average performance. Section 4 presents and analyzes the algorithm given by Meyer.

In order to analyze the expected number of swaps required by these algorithms, we shall assume that the possible initial configurations of pebbles are equally likely. Hence, for the problem of Section 1, each of the possible initial configurations has probability 2^{-N}, and for the problem of Sections 2, 3, and 4, each of the possible initial configurations has probability 3^{-N}.

1. The Solution to the Two-Color Problem

Consider the problem defined above when only two colors are involved, say red and white (this problem is identical to the one posed by Knuth [2, 5.22–33]). The solution to this simplified problem is the basis of the algorithm of Section 4. The idea is to move from left to right (using variable r) until a white pebble is encountered and then to move from right to left (using variable w) until a red pebble is found. These pebbles are then swapped and the algorithm continues. A Pascal procedure to accomplish this is

```
procedure monacan_national_flag;
  var r, w : integer; colr, colw : color;
begin
  r := 1; w := N;
  while w >= r do
    begin
      colr := buck(r);
      while (colr=red) and (w>r) do
        begin
          r := r + 1;
          colr := buck(r)
        end;
      if r < w then
        begin
          colw := buck(w);
          while (colw< >red) and (w>r+1) do
            begin
              w := w - 1;
              colw := buck(w)
            end;
          if colw = red then swap(r, w)
        end;
      r := r + 1;
      w := w - 1
    end
end
```

The expected behavior of this algorithm is given in the following theorem.

THEOREM 1. *Let $F_2(N)$ be the expected number of swaps used by the two color algorithm above when there are N uninspected pebbles. Then*

$$F_2(N) = \tfrac{1}{4}(N - 1).$$

PROOF. Let $G_2(N)$ be the expected number of swaps used by the two color algorithm above when there are $N - 1$ uninspected pebbles and the pebble in bucket r is known to be white. Then

$$F_2(N) = \tfrac{1}{2} F_2(N-1) + \tfrac{1}{2} G_2(N)$$

since bucket r contains a red pebble with probability $\frac{1}{2}$ and contains a white pebble with probability $\frac{1}{2}$. Now,

$$G_2(N) = \tfrac{1}{2} G_2(N-1) + \tfrac{1}{2}(1 + F_2(N-2))$$

since with probability $\frac{1}{2}$ bucket w contains a white pebble and no swap is made and with probability $\frac{1}{2}$ bucket w contains a red pebble and one swap is made. This system of recurrence relations reduces to:

$$F_2(N) = \tfrac{1}{4} + F_2(N-1).$$

The initial condition is $F_2(1) = 0$. □

Proof that this algorithm is optimal follows in the following theorem.

THEOREM 2. *Any algorithm for the two color problem using swaps requires at least* $\frac{1}{4}(N-1)$ *swaps on the average.*

PROOF. Consider the $\binom{N}{k}$ initial configurations of pebbles with k red pebbles and $N - k$ white pebbles, with $0 \le k \le N$. There are $\binom{k}{i}\binom{N-k}{i}$ of these configurations which can achieve the final configuration using i swaps, but cannot achieve the final configuration using less than i swaps. Hence, for these initial configurations, the best possible average number of swaps is

$$\binom{N}{k}^{-1} \sum \binom{k}{i}\binom{N-k}{i} i = (k(N-k))/N.$$

Therefore over all possible bucket arrangements, the minimal average number of swaps is

$$2^{-N} \sum \binom{N}{k} \frac{(k(N-k))}{N} = \tfrac{1}{4}(N-1).$$ □

Although this algorithm is simple, it is best possible and provides the key to the solution in Section 4.

2. A Simple Solution to the Three-Color Problem

We now return to the three-color Dutch National Flag Problem. Recall the general situation of four regions delineated by the variables r, w, and b. Dijkstra's first solution examines one bucket at a time and places its pebble in an appropriate bucket. There are essentially two choices for the bucket to be inspected: the one at the red boundary (pointed to by r) and the one at the white boundary (pointed to by w). The latter choice leads to a better algorithm, so the algorithm will move from right to left across the uninspected area. A Pascal procedure for the algorithm is

```
procedure dutch_national_flag;
  var r, w, b : integer;
begin
  r := 1; w := N; b := N;
  while w >= r do
    case buck(w) of
      red   : begin swap(r, w); r := r + 1 end;
      white : w := w - 1;
      blue  : begin swap(w, b); w := w - 1; b := b - 1 end
    end
end
```

We now state the easily proved Theorem 3.

THEOREM 3. *Let* $F_{RL}(N)$ *be the expected number of swaps used by the right-to-left algorithm above when the uninspected area has N buckets. Then*

$$F_{RL}(N) = \tfrac{2}{3} N.$$

Although the solution to the problem is simple, the algorithm is far from optimal.

3. A More Refined Solution

The solution presented in the previous section is quite simple. Dijkstra attempts to enhance its performance by some simple refinements. To this end, he first observes that when all of the pebbles are red, the algorithm uses N swaps when none is necessary. This consideration leads to the following ideas: (i) it may be advantageous to examine two buckets at a time, instead of just one (although this seems to increase the number of cases to be handled from 3 to 9); (ii) the solution should attempt to move the red boundary at r as far to the right as possible without swapping; and (iii) similarly, the solution should attempt to move the white boundary at w as far to the left as possible without swapping. If these ideas are implemented, then the pebble at r is either white or blue and the pebble at w is either red or blue. As long as $r < w - 1$, there are only four cases. The refined solution in Pascal is

```
procedure dutch_national_flag;
  var r, w, b : integer; colr, colw : color;
begin
  r := 1; w := N; b := N;
  while w >= r do
    begin
      colr := buck(r);
      while (colr = red) and (w>r) do
        begin
          r := r + 1;
          colr := buck(r)
        end;
      if r < w then
        begin
          colw := buck(w);
          while (colw=white) and (w>r+1) do
            begin
              w := w - 1;
              colw := buck(w)
            end;
```

```
            case colw of
              red   : begin swap(r, w); r := r + 1 end;
              white : w := w - 1;
              blue  : begin swap(w, b); w := w - 1;
                            swap(r, w); b := b - 1 end
            end
          end;
        if colr = blue then begin swap(w, b); b := b - 1 end;
        w := w - 1
      end
end
```

The expected behavior of this algorithm is given in Theorem 4.

THEOREM 4. *Let $F_R(N)$ be the expected number of swaps used by the refined algorithm when the uninspected area has N buckets. Then*

$$F_R(N) = \tfrac{2}{3}N - \tfrac{1}{4} + \tfrac{1}{4}(-\tfrac{1}{3})^N.$$

PROOF. Let $R_w(N)$ be the expected number of swaps required by the refined algorithm when the uninspected area contains $N - 1$ buckets and bucket r is known to contain a white pebble. Similarly, let $R_b(N)$ be the expected number of swaps required when the uninspected area contains $N - 1$ buckets and bucket r is known to contain a blue pebble. Then

$$F_R(N) = \tfrac{1}{3}F_R(N-1) + \tfrac{1}{3}R_w(N) + \tfrac{1}{3}R_b(N).$$

The terms on the right are the result of obtaining a red, a white, or a blue pebble, respectively, in bucket r. Now consider the function R_w:

$$R_w(N) = \tfrac{1}{3}(1 + F_R(N-2)) + \tfrac{1}{3}R_w(N-1) + \tfrac{1}{3}(2 + F_R(N-2)).$$

The first term on the right arises from a red pebble in bucket w, requiring one swap. The second term reflects the fact that we are skipping over white pebbles at bucket w. The last term results from a blue pebble in bucket w, requiring two swaps. Simplifying, we obtain

$$R_w(N) = 1 + \tfrac{2}{3}F_R(N-2) + \tfrac{1}{3}R_w(N-1).$$

In a similar way, we obtain for R_b:

$$\begin{aligned} R_b(N) &= \tfrac{1}{3}(2 + F_R(N-2)) + \tfrac{1}{3}R_b(N-1) + \tfrac{1}{3}(3 + F_R(N-2)) \\ &= \tfrac{5}{3} + \tfrac{2}{3}F_R(N-2) + \tfrac{1}{3}R_b(N-1). \end{aligned}$$

Now let $G_R(N) = R_w(N) + R_b(N)$. Then we obtain the system of recurrence relations

$$\begin{aligned} F_R(N) &= \tfrac{1}{3}F_R(N-1) + \tfrac{1}{3}G_R(N) \\ G_R(N) &= \tfrac{8}{3} + \tfrac{4}{3}F_R(N-2) + \tfrac{1}{3}G_R(N-1). \end{aligned}$$

This system reduces to

The initial conditions are $F_R(1) = \frac{1}{3}$ and $F_R(2) = \frac{10}{9}$. □

It is now evident that the refined algorithm has an expected number of swaps which is of the same rate of growth as the same quantity for the right-to-left algorithm. The difference between these values is $\frac{1}{4} - \frac{1}{4}(-\frac{1}{3})^N$. For $N \geq 0$, this quantity attains a maximum of $\frac{1}{3}$ when $N = 1$. As N approaches infinity, this quantity approaches $\frac{1}{4}$ so that independent of the number of buckets, the average saving which the refinement achieves is no more than $\frac{1}{3}$ of a swap.

The ideas which guide the refinement process are sound; however, one important feature still remains common to the two algorithms. In both algorithms, situations occur when a pebble whose color has not been established is swapped into a new bucket. Since this pebble might later have to be returned to a bucket in the same region as the pebble's initial bucket, the algorithm will execute two swaps when none is necessary; these swaps dominate the running times of the algorithms.

4. Meyer's Solution

In this section, we shall consider a solution which is shorter and clearer than the refined solution and avoids the problem of swapping uninspected pebbles. As a starting point, recall the two-color algorithm of Section 1. The uninspected area is scanned from left to right (using variable r). Once again, red pebbles are skipped until a nonred pebble is found. When a nonred pebble is encountered, then scanning begins from right to left (using variable w). White pebbles are skipped until a nonwhite pebble is found. If the nonwhite pebble in bucket w is blue, then it is swapped with the pebble in bucket b (this pebble must be white unless $b = w$) and scanning continues. When a red pebble is encountered at bucket w, it is swapped with the nonred pebble in bucket r. If, in addition, this nonred pebble (previously in bucket r) is blue, then it is swapped with the pebble in bucket b. Notice that at no time will an uninspected pebble be swapped.

This algorithm was developed by Meyer [3]. A Pascal procedure implementing the idea is

```
procedure dutch_national_flag;
  var r, w, b : integer; colr, colw : color;
begin
  r := 1; w := N; b := N;
  while w >= r do
    begin
      colr := buck(r);
      while (colr=red) and (w>r) do
        begin
          r := r + 1;
          colr := buck(r)
        end;
```

```
    if r < w then
      begin
        colw := buck(w);
        while (colw< >red) and (w>r+1) do
          begin
            if colw = blue then begin swap(w, b); b := b - 1 end;
            w := w - 1;
            colw := buck(w)
          end;
        if colw = blue then begin swap(w, b); b := b - 1 end;
        if (colw=red) or (colr< >white) then swap(r, w)
      end;
    if colr = blue then begin swap(w, b); b := b - 1 end;
    r := r + 1;
    w := w - 1
  end
end
```

This program is quite similar to the program of Section 1. Three statements have been added, one to handle the occurrence of a blue pebble while scanning from left to right and two to handle the occurrence of a blue pebble while scanning from right to left. The expected behavior of this algorithm is given in Theorem 5.

THEOREM 5. *Let $F_M(N)$ be the expected number of swaps used by the above algorithm when there are N uninspected pebbles. Then*

$$F_M(N) = \tfrac{5}{9}N,\ N > 1;\ F_M(1) = \tfrac{1}{3}.$$

PROOF. Let $M_w(N)$ and $M_b(N)$ be the expected number of swaps used by the above algorithm when the uninspected area has $N - 1$ buckets and the pebble in bucket r is known to be white and blue, respectively. Then

$$F_M(N) = \tfrac{1}{3}F_M(N-1) + \tfrac{1}{3}M_w(N) + \tfrac{1}{3}M_b(N).$$

The three terms on the right are obtained by a red pebble, a white pebble, and a blue pebble, respectively, in bucket r. Consider the function M_w:

$$M_w(N) = \tfrac{1}{3}(1 + F_M(N-2)) + \tfrac{1}{3}M_w(N-1) + \tfrac{1}{3}(1 + M_w(N-1)).$$

Once again, the three terms on the right arise from a red pebble, a white pebble and a blue pebble, respectively. Simplifying,

$$M_w(N) = \tfrac{2}{3} + \tfrac{1}{3}F_M(N-2) + \tfrac{2}{3}M_w(N-1).$$

Similarly for M_b:

$$M_b(N) = \tfrac{1}{3}(2 + F_M(N-2)) + \tfrac{1}{3}M_b(N-1) + \tfrac{1}{3}(1 + M_b(N-1)) = 1 + \tfrac{1}{3}F_M(N-2) + \tfrac{2}{3}M_b(N-1).$$

Now let $G_M(N) = M_w(N) + M_b(N)$. Then we obtain the system of recurrence relations:

$$F_M(N) = \tfrac{1}{3}F_M(N-1) + \tfrac{1}{3}G_M(N)$$

$$G_M(N) = \tfrac{5}{3} + \tfrac{2}{3}F_M(N-2) + \tfrac{2}{3}G_M(N-1).$$

This system reduces to

$$F_M(N) = \tfrac{5}{9} + F_M(N-1).$$

The initial condition is $F_M(2) = \frac{10}{9}$. □

It may be argued that this same solution can be obtained by the refinement process if one observes that, rather than moving the white boundary as far to the left as possible without swapping, the solution should attempt to move the white boundary to the left until a red pebble is encountered, skipping white pebbles and swapping blue pebbles.

Refinements might also be applied to this program to eliminate redundant swaps. One such refinement would be to replace each of the three occurrences of "*swap(w, b)*" by "**if** $w < b$ **then** *swap(w, b)*." However, this refinement only reduces the expected number of swaps by $1 + (\frac{1}{8}(N-8))(\frac{2}{3})^N$, for $N > 1$. Another suggestion might be to look at more than just two pebbles at a time. Although this may improve the efficiency of the algorithm, the complexity of the resulting program increases rapidly with the number of pebbles simultaneously scanned.

Conclusion

In spite of its apparent simplicity, the Problem of the Dutch National Flag exemplifies several interesting points about program development. The improvement in efficiency resulting from a refinement can be deceptively small. A negligible efficiency gain may be obtained at the cost of increased program complexity, more complicated proof of correctness and lengthier documentation. A mathematical analysis of the proposed algorithms with respect to a precise model can be used to expose these tradeoffs. In addition, the way in which refinement may best be applied to an existing program is not always obvious; in this case, a solution with a nontrivial improvement in efficiency could have been obtained if a different set of observations had been made. As in the present problem, it may be more effective to consider a simpler problem and extend its solution, than to refine an existing program.

Acknowledgments. I would like to thank Arthur Gill for his support and encouragement, and Steven Meyer for many provocative discussions.

Received December 1976; revised January 1978

References

1. Dijkstra, E.W. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, N.J., 1976.
2. Knuth, D.E. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
3. Meyer, S.J. A failure of structured programming. Zilog Corp., Software Dept. Technical Report No. 5, Cupertino, CA., February, 1978.