THE LINEAR SEARCH REDISCOVERED

In a recent paper Dijkstra and Feijen (1989) derive an unusual program for linear searching. The authors ask their readers the following question: “Did you know this program for *The Bounded Linear Search*? We did not.”

I did indeed. I derived it at Caltech around 1973 and published it in a textbook (Brinch Hansen 1985). I found it by trying to write an abstract program for searching an ordered array without initially specifying a particular search method!

I will restate the problem which I solved in Pascal using different variable names. Suppose that n integers are stored in non-decreasing order in the array elements A[1], A[2], ... A[n]. Find an element A[i] (if it exists) which has a given value x.

My starting point was the following abstract program:

```
consider all elements;
while more than one element do
  begin
    partition interval;
    choose subinterval
end;
examine final element
```

Initially the algorithm considers all elements in the interval from 1 to n. The interval is gradually reduced until there is only one element left. The final element is then examined to determine if it holds a solution.

The invariant of the loop can be stated as follows: If the problem has any solutions at least one of them can be found in the current interval from i to m, where

$$1 \le i \le m \le n$$

We can now refine some of the program pieces:

P. Brinch Hansen, The linear search rediscovered. *Structured Programming 11*, (1990), 53–55.

```
consider all elements:
  i := 1; m := n

more than one element:
  i < m

examine final element:
  found := A[i] = x
```

If the interval from i to m holds more than one element, we use a particular search algorithm to find an element k which divides the interval into a left interval from i to k and a right interval from k + 1 to m, where

$$1 \leq i \leq k < m \leq n$$

In choosing one of the subintervals, there are three cases to consider:

1. If A[k] < x there is no solution in the left interval.
2. If A[k] = x there is a solution in the left interval.
3. If A[k] > x there is no solution in the right interval.

We can now define yet another program piece:

```
choose subinterval:
  if A[k] ≥ x
    then {choose left interval} m := k
    else {choose right interval} i := k + 1
```

The only program piece that depends on a particular search method is the one which partitions the current interval.

If we cut the current interval in half we have a **binary search**:

```
i := 1; m := n;
while i < m do
  begin
    k := (i + m) div 2;
    if A[k] ≥ x then m := k
    else i := k + 1
  end;
found := A[i] = x
```

If the left interval consists of a single element onIy, we have a **linear search**:

```
i:= 1; m := n;
while i < m do
  begin
    k:= i;
    if A[k] ≥ x then m := k
    else i := k + 1
  end;
found := A[i] = x
```

By eliminating the superfluous variable k, you get the following (unexpected) solution:

```
i := 1; m := n;
while i < m do
  if A[i] ≥ x then m := i
  else i := i + 1
```

If you replace the condition A[i] ≥ x by A[i] = x, the algorithm finds the first element (if any) which equals x in an unordered array. Finally, if you write this version of the algorithm using parallel assignment and guarded commands, you obtain the algorithm which I published (without any explanation) in Brinch Hansen (1985):

```
i, m := 1, n;
do i < m →
  if A[i] = x → m := i []
    not (A[i] = x) → i := i + 1
  fi
od
found := A[i] = x
```

References

Dijkstra, E. W, and Feijen, W. H. J. 1989. The Linear Search Revisited. *Structured Programming 10*, 1, 5–8.

Brinch Hansen, P. 1985. *On Pascal Compilers.* Prentice-Hall, Englewood Cliffs, NJ, 283.