Science of Computer Programming 7 (1986) 243-248
North-Holland

SMALL PROGRAMMING EXERCISES 13

M. REM

Department of Mathematics and Computing Science, Eindhoven University of Technology, 5600 MB Eindhoven, Netherlands

Many computing problems may be formulated as computations on graphs. The graph, usually a directed one, then represents the essential structure of the problem.

This time we have again two graph problems, both of which have been composed by R. H. Mak. In our representation of graphs the vertices are identified by distinct natural numbers. In the first exercise these numbers play an essential role: for each vertex we have to determine which of its ancestors has the least number.

A directed graph is called *equidistant* if for each pair (j, k) of vertices all paths from j to k have the same number of arcs. Obviously, equidistance implies acyclicity. We are requested to check whether a given graph is equidistant. In order to simplify this exercise somewhat, it is given that all vertices are reachable from vertex 0. Both problems allow solutions that are linear in the number of vertices and arcs.

Exercise 32: Least ancestors

Given is a directed graph G. Let

$$R(k) = \{j \mid G \text{ has a path from vertex } k \text{ to vertex } j\}.$$

Notice that $k \in R(k)$. Vertex k is called an *ancestor* of vertex j if $j \in R(k)$. Graph G is represented by its successor sets, as explained in *Small Programming Exercises* 5.

We have to solve S in

```
|[N, M: int; {N ≥ 1 ∧ M ≥ 0}
  b(j: 0≤j≤N), e(i: 0≤i<M): array of int;
  {SUC(G, b, e)}
  |[a(j: 0≤j<N): array of int;
    S
    {(Aj: 0≤j<N: a(j) = (MIN k: 0≤k<N ∧ j∈R(k): k))}
  ]|
]|
```

Exercise 33: Checking the equidistance of a digraph

With R as defined above, the functional specification reads

```
|[N, M: int; {N ≥ 1 ∧ M ≥ 0}
  b(j: 0≤j≤N), e(i: 0≤i<M): array of int;
  {SUC(G, b, e) ∧ (Aj: 0≤j<N: j ∈ R(0))}
  |[eq: bool;
    S
    {eq ≡ (G equidistant)}
  ]|
]|
```

Solution of Exercise 30 (the flag of Alphanumerica)

The reader is advised to consult the problem description in *Small Programming Exercises* 12. The functional specification of the iterative version *IS* is

```
|[N: int; {N ≥ 0}
  |[c: cursor; {c.x = 0 ∧ c.y = 0}
    fl(i, j: 0≤i<2^N ∧ 0≤j<N): array of (blank, star);
    IS
    {fl(i, j: 0≤i<2^N ∧ 0≤j<N)   contains the flag of Alphanumerica}
  ]|
]|
```

We introduce variables m and n and maintain the following invariant:

$c.x = 0 \wedge 0 \leq c.y \leq N$
$\wedge\ m = 2^{N-c.y} \wedge n = 2^{c.y}$
$\wedge$ ($fl(i, j: 0 \leq i < 2^N \wedge 0 \leq j < c.y)$ contains the top $c.y$ rows of the flag of Alphanumerica).

If $m = 1$ we have $c.y = N$ and the postcondition is satisfied. For $m \neq 1$ the next row to be printed is, according to the specification of the flag, a sequence of n groups, each containing $m/2$ blanks followed by $m/2$ stars.

Thus, we find

```
IS:  |[m, n: int; m,n := 2 ↑ N, 1
     ; do m ≠ 1
       → |[i: int; i,m := 0, m/2
          ; do i ≠ n
            → |[j: int;
                 j := 0 ; do j ≠ m → bl; j := j+1 od
                ;j := 0; do j ≠ m → st; j := j+1 od
               ]|
               i := i+1
            od
          ]|
       ;nl; n := 2 * n
```

```
  od
]|
```

The functional specification of the recursive version *RS* is

```
|[N: int; {N ≥ 0}
 |[k: int; {k = K ∧ 0 ≤ K ≤ N}
   c: cursor; {c.x = X ∧ c.y = Y ∧ 0 ≤ X ≤ 2^N − 2^K ∧ 0 ≤ Y ≤ N − K}
   fl(i, j: 0 ≤ i < 2^N ∧ 0 ≤ j < N): array of (blank, star);
   RS
   {fl(i, j: X ≤ i < X + 2^K ∧ Y ≤ j < Y + K)
    contains the flag of Alphanumerica}
 ]|
]|
```

The recursive nature of *RS* requires the functional specification to be strengthened. We extend the specification with the requirement that *RS* leaves the remainder of *fl* unchanged, for whose formulation we introduce *F* as the initial value of *fl*. We also add conditions on the final values of *k* and *c*. Our extended specification is

```
|[N: int; {N ≥ 0}
 |[k: int; {k = K ∧ 0 ≤ K ≤ N}
   c: cursor; {c.x = X ∧ c.y = Y ∧ 0 ≤ X ≤ 2^N − 2^K ∧ 0 ≤ Y ≤ N − K}
   fl(i, j: 0 ≤ i < 2^N ∧ 0 ≤ j < N): array of (blank, star);
   {(Ai, j: 0 ≤ i < 2^N ∧ 0 ≤ j < N: fl(i, j) = F(i, j))}
   RS
   {k = K ∧ c.x = X + 2^K ∧ c.y = Y
   ∧ (Ai, j: 0 ≤ i < X ∨ X + 2^K ≤ i < 2^N
            ∨ 0 ≤ j < Y ∨ Y + K ≤ j < N: fl(i, j) = F(i, j))
   ∧ (fl(i, j: X ≤ i < X + 2^K ∧ Y ≤ j < Y + K)
   contains the flag of Alphanumerica)}
 ]|
]|
```

Program *RS* is to print a flag of *k* rows. If $k = 0$ it is a **skip**. If $k \geq 1$ program *RS* prints row 0 (consisting of 2^{k-1} blanks followed by the same number of stars) and two flags of $k-1$ rows each:

```
RS:   if k = 0 → skip
      □ k ≥ 1 → k := k − 1
              ; |[M: int; M := 2 ↑ k
                ; |[j: int;
                    j := 0; do j ≠ M → bl; j := j + 1 od
                  ; j := 0; do j ≠ M → st; j := j + 1 od
                  ]|
                ; back(2 * M)
```

```
        ]|
      ; down; RS; RS; up
      ; k := k+1
fi
```

It is a nice solution, but it does require backspacing and forward and reverse line feed.

In both countries the flag rotated by 90° is easier to print. We skip their functional specifications and give possible solutions only. In ISA the rotated flag may be produced by printing the natural numbers up to 2^k in binary notation, the blank replacing 0 and the star replacing 1:

```
IS':  |[M, i := 2 ↑ k, 0
      ; do i ≠ M
        → |[j, n: int; j, n := i, 0
          ; do n ≠ k → if j mod 2 = 0 → bl
                       □ j mod 2 = 1 → st
                       fi
                     ; j, n := j div 2, n+1
          od
          ]|
        ; nl; i := i+1
        od
      ]|
```

In RSA the difference is even more striking. The recursive printing of the rotated flag does not require backspacing or line feed. Next to k, we assume the existence of another global variable, this one of type **string of** (*blank*, *star*). If s is such a string, expression $l(s)$ yields the length of s, command $pr(s)$ prints s, and (*blank*: s) denotes the result of concatenating a blank at the front of s.

The effect of *RS'* is that a rotated Alphanumerican flag is printed that consists of $2^{k-l(s)}$ rows and $k-l(s)$ columns with each row extended by string s, resulting in a printed area of k columns. If string s is initially empty *RS'* prints, consequently, a rotated flag of 2^k rows.

```
RS':  if l(s) = k → pr(s); nl
      □ l(s) < k → |[t: string of (blank, star); t := s
                   ; s := (blank: t); RS'
                   ; s := (star: t); RS'
                   ]|
      fi
```

Solution of Exercise 31 (balanced segments)

We have to solve S in

```
|[N: int; {N ≥ 0}
  X(i: 0 ≤ i < N): array of int;
```

$$
\begin{array}{l}
|[r\colon \mathit{int};\\
\quad S\\
\quad \{r = (\mathbf{MAX}\ p, q\colon 0 \leqslant p \leqslant q \leqslant N \wedge (\mathbf{N}i\colon p \leqslant i < q\colon X(i) < 0)\\
\qquad\qquad\qquad\qquad = (\mathbf{N}i\colon p \leqslant i < q\colon X(i) > 0)\colon q - p)\}\\
\quad]|\\
]|
\end{array}
$$

Let, for $0 \leqslant j \leqslant N$, $E(j)$ denote the excess of negative numbers over positive numbers in $X(i\colon 0 \leqslant i < j)$:

$$E(j) = (\mathbf{N}i\colon 0 \leqslant i < j\colon X(i) < 0) - (\mathbf{N}i\colon 0 \leqslant i < j\colon X(j) > 0).$$

Notice that $E(0) = 0$ and that $-j \leqslant E(j) \leqslant j$. The postcondition may then be written as

$$r = (\mathbf{MAX}\ p, q\colon 0 \leqslant p \leqslant q \leqslant N \wedge E(p) = E(q)\colon q - p).$$

We obtain our invariant by replacing in the postcondition constant N by a variable:

$$
\begin{array}{ll}
P0\colon & 0 \leqslant n \leqslant N\\
& \wedge\ r = (\mathbf{MAX}\ p, q\colon 0 \leqslant p \leqslant q \leqslant n \wedge E(p) = E(q)\colon q - p).
\end{array}
$$

It may be initialized with $n, r = 0, 0$.

An increment of n by 1 requires determining the least p such that $E(p) = E(n+1)$. The value of $E(n+1)$ can be deduced from that of $E(n)$ as follows:

$$E(n+1) = \begin{cases} E(n)+1 & \text{if } X(n) < 0,\\ E(n) & \text{if } X(n) = 0,\\ E(n)-1 & \text{if } X(n) > 0. \end{cases}$$

We, therefore, record the value of $E(n)$:

$$P1\colon \quad e = E(n).$$

The numbers in the set $\{E(i) \mid 0 \leqslant i \leqslant n\}$ form an interval $[a, b]$ of integer numbers. In order to facilitate the search for the least p such that $E(p) = E(n+1)$, we record for each m in that interval the least p such that $E(p) = m$:

$$
\begin{array}{ll}
P2\colon & a = (\mathbf{MIN}\ i\colon 0 \leqslant i \leqslant n\colon E(i))\\
& \wedge\ b = (\mathbf{MAX}\ i\colon 0 \leqslant i \leqslant n\colon E(i))\\
& \wedge\ (\mathbf{A}m\colon a \leqslant m \leqslant b\colon f(m) = (\mathbf{MIN}\ i\colon 0 \leqslant i \leqslant n \wedge E(i) = m\colon i)).
\end{array}
$$

With P0 $\wedge$ P1 $\wedge$ P2 as our invariant, the solution becomes

$$
\begin{array}{ll}
S\colon & |[n, e, a, b\colon \mathit{int};\\
& \quad f(m\colon -N \leqslant m \leqslant N)\colon \mathit{array\ of\ int};\\
& \quad n, r, e, a, b := 0, 0, 0, 0, 0\\
& \quad ; f\colon (0) = 0\\
& \quad ; \mathbf{do}\ n \neq N\\
& \qquad \rightarrow \mathbf{if}\ X(n) < 0 \rightarrow e := e + 1
\end{array}
$$

```
    □X(n) = 0 → skip
    □X(n) > 0 → e := e − 1
    fi
;   if e = a − 1 → a := e; f:(e) = n + 1
    □a ≤ e ≤ b → skip
    □e = b + 1 → b := e; f: (e) = n + 1
    fi
;   r := r max (n + 1 − f(e))
;   n := n + 1
 od
]|
```

The execution time of our solution is proportional to N.