The Saddleback Search

The origin of this algorithm is unknown; its name has been invented by David Gries. It solves a problem that can be stated in many variations; we shall first solve it in one of its straightforward versions and then discuss several variations.

We are given an integer function f of two natural arguments that is increasing in both its arguments and takes on the value F at least once. Saddleback Search has to locate such an occurrence, more precisely, the occurrence with the smallest value of the first argument. Because f is increasing in both its arguments, this is at the same time the occurrence with the largest value of the second argument. Thus we are lead to the following formal specification

|[F: int; f(i,j: 0≤i ∧ 0≤j) <u>array</u> <u>of</u> int

  {(<u>A</u> i,j,ii,jj: 0≤i<ii ∧ 0≤j<jj: f.i.j < f.ii.j ∧ f.i.j < f.i.jj) ∧

  f.X.Y = F ∧ (<u>A</u> i,j: 0≤i<X ∨ j>Y: f.i.j ≠ F)}

 ; |[x,y: int

   ; Saddleback Search

    {R: x,y = X,Y}

   ]|

 ]|

The analogy with the Linear Search now suggests to approach X from below and Y from above, i.e. to iterate with the invariant P, given by

$$P: \quad 0 \le x \le X \ \wedge \ y \ge Y \quad .$$

Our first approximation of Saddleback Search keeps the analogy to the Linear Search as close as possible:

"establish P"
; <u>do</u> $x < X \rightarrow x := x+1 \{P\} \ [\!] \ y > Y \rightarrow y := y-1 \{P\}$ <u>od</u> $\{R\}$.

We note that this program differs in two respects from the corresponding approximation —see EWD930— of the Linear Search: firstly —on account of the conjunct $y \ge Y$— the initialisation cannot be done independently of f ("$y := +\infty$" is not acceptable) and secondly the repetition here is nondeterministic.

In order to relate our inequalities involving X and Y to f, we observe the

<u>Lemma</u> $\quad x \le X \ \wedge \ y \ge Y \ \equiv$
$\quad (\underline{A}\, i,j : 0 \le i < x \ \vee \ j > y : f.i.j \ne F)$

<u>Proof</u>
$$\begin{array}{l} x \le X \ \wedge \ y \ge Y \\ \Rightarrow \{(\underline{A}\, i,j : 0 \le i < X \ \vee \ j > Y : f.i.j \ne F)\} \\ \quad (\underline{A}\, i,j : 0 \le i < x \ \vee \ j > y : f.i.j \ne F) \\ \Rightarrow \{ f.X.Y = F \ \wedge \ X \ge 0 \} \\ \quad X \ge x \ \wedge \ Y \le y \end{array}$$

(End of Proof)

With the above Lemma in our hands we now tackle the guards of the repetition, strengthened by the invariant:

$P \wedge x < X$
$= \{$arithmetic and definition of $P\}$
$P \wedge x+1 \le X \wedge y \ge Y$
$= \{$Lemma$\}$
$P \wedge (\underline{A}i,j: 0 \le i < x+1 \vee j > y: f.i.j \ne F)$
$= \{$definition of P, Lemma and predicate calculus$\}$
$P \wedge (\underline{A}j: 0 \le j \le y: f.x.j \ne F)$
$\Leftarrow \{f$ is increasing in its second argument$\}$
$P \wedge f.x.y < F$,

and

$P \wedge y > X$
$= \{$arithmetic and definition of $P\}$
$P \wedge x \le X \wedge y-1 \ge Y$
$= \{$Lemma$\}$
$P \wedge (\underline{A}i,j: 0 \le i < x \vee j > y-1: f.i.j \ne F)$
$= \{$definition of P, Lemma and predicate calculus$\}$
$P \wedge (\underline{A}i: i \ge x: f.i.y \ne F)$
$\Leftarrow \{f$ is increasing in its first argument$\}$
$P \wedge f.x.y > F$.

Hence we find ourselves invited to consider the second approximation with the (conditionally) strengthened guards

"establish P"
; $\underline{\text{do}}$ $f.x.y < F \rightarrow x := x+1 \{P\}$ $[]$ $f.x.y > F \rightarrow y := y-1 \{P\}$ $\underline{\text{od}}$

for which we have to check that, though the guards have been strengthened, the final conclusion R is still justified. Indeed:

$P \wedge f.x.y \ge F \wedge f.x.y \le F$
$= \{$arithmetic$\}$
$P \wedge f.x.y = F$
$= \{$definition of P and $X, Y\}$

$x, y = X, Y$.

Because f is increasing in both arguments, $x = 0 \wedge f.0.y \geq F \Rightarrow P$. Thus we arrive at a complete program for the Saddleback Search

$$\begin{array}{l} \quad x, y := 0, 0 \\ ;\ \underline{\text{do}}\ f.x.y < F \rightarrow y := y+1\ \underline{\text{od}}\ \{P\} \\ ;\ \underline{\text{do}}\ f.x.y < F \rightarrow x := x+1 \\ \quad [\!]\ f.x.y > F \rightarrow y := y-1 \\ \quad \underline{\text{od}}\ \{R\} \end{array}$$

Convergence of the first repetition is guaranteed by the fact that $f.x.y$ is increasing in its second component.

* * *

We observed at the beginning that the occurrence of F with the smallest value of the first argument is also that with the largest value of the second argument. Consequently we could also have defined X, Y as the solution of $x, y: (f.x.y = F)$ with the minimum value for $x - y$:

$$f.X.Y = F \ \wedge\ (\underline{A}\, i, j :\ 0 \leq i \ \wedge\ j \leq 0 \ \wedge\ i - j < X - Y :\ f.i.j \neq F) .$$

The disadvantage of this definition is that, for the proof of our Lemma, another appeal to f's double monotonicity would be required. It has the advantage that a first approximation —viz. to investigate values of $f.x.y$ for increasing values of $x - y$ — would have been a more direct analogue of the Linear Search. The approach has a further heuristic virtue.

Under the invariant $x-y \leq X-Y$ the search would continue until $x-y = X-Y$. But those two conditions do not imply $x,y = X,Y$! However

$$x,y = X,Y \quad \equiv \quad x-y = X-Y \ \wedge \ x \leq X \ \wedge \ y \geq Y$$

– a nice little theorem I did not know – and we are thus led to the stronger invariant $x \leq X \wedge y \geq Y$. (I did this derivation as well, and it was kind of nice; it was essentially the case analysis needed in the proof of the Lemma, that put me off.)

* * *

The first variation is Saddleback Count, which, instead of locating an occurrence, counts the number of occurrences. It does so in the order of increasing $x-y$. Formally specified

|[F: int ; f(i,j: 0≤i ∧ 0≤j) __array__ __of__ int
  {($\underline{A}$ i,j,ii,jj: 0≤i<ii ∧ 0≤j<jj: f.i.j < f.ii.j ∧ f.i.j < f.i.jj) ∧
   K = ($\underline{N}$ (i,j): 0≤i ∧ 0≤j : f.i.j = F)}
; |[k: int
  ; Saddleback Count
   {R: k = K}
  ]|
]| .

The invariant P is given by

P: k = ($\underline{N}$ (i,j): 0≤i < x ∨ j > y : f.i.j = F)

or, equivalently

$P: \quad k + (\underline{N}(i,j) : i \geq x \wedge 0 \leq j \leq y : f.i.j = F) = K$.

A solution for Saddleback Count is

$|[$ x, y : int
; x, y, k := 0, 0, 0 ; <u>do</u> f.x.y < F → y := y+1 <u>od</u> {P}
; <u>do</u> y ≥ 0 → <u>if</u> f.x.y < F → x := x+1
    ▯ f.x.y > F → y := y−1
    ▯ f.x.y = F → x, y, k := x+1, y−1, k+1
    <u>fi</u> {P}
  <u>od</u> {R}
$]|$ {R}

which, I trust, now requires no further explanation.

* * *

A next variation is that in the declaration of f the first argument is bounded by $0 \leq i < I$ and/or the second argument is bound by $0 \leq j < J$.

$0 \leq i < I$: this bound has no influence on the text of Saddleback Search; for Saddleback Count the guard $y \geq 0$ of the last repetition has to be replaced by the stronger $x < I \wedge y \geq 0$ so as to prevent "index out of bounds". The proper reformulation of the invariants is left as an exercise to the reader.

$0 \leq j < J$: in Saddleback Search P is established by $x, y := 0, J-1$, in Saddleback Count by $x, y, k := 0, J-1, 0$.

The next variation to consider is a weakening of the monotonicity requirements on f from increasing to ascending.

Saddleback Search is also okay for an f that is ascending in its first and increasing in its second argument. If f is only given to be ascending in both arguments, the program for bounded second argument is still okay, but for unbounded second argument the establishment of P has to be effectuated by

$$x, y := 0, 0$$
$$;\ \underline{\text{do}}\ f.x.y \le F \rightarrow y := y+1\ \underline{\text{od}}$$

and f has to be such that this repetition converges, i.e., for increasing y, $f.0.y$ has to grow beyond F.

In the case of Saddleback Count we have in any case to insist that K exists (i.e. is finite), which is the case if both arguments are bounded from above or $f.x.y$ grows beyond F for increasing $x+y$. If, in addition, f is increasing in one of its arguments, its second one, say, it suffices to modify the third guarded command of the alternative construct from

$$f.x.y = F \rightarrow x, y, k := x+1, y-1, k+1$$

into

$$f.x.y = F \rightarrow x, k := x+1, k+1$$

(the original being an optimization of the latter which is valid if f is increasing in its first argument).

For Saddleback Count applied to an f given to be ascending in both unbounded arguments and $f.x.y$ growing beyond F for increasing $x+y$, we strengthen the invariant to $P \wedge Q$ with P (as before) given by

$$P: \quad k + (\underline{N}(i,j) : i \geq x \wedge 0 \leq j \leq y : f.i.j = F) = K$$

and Q given by

$$Q: \quad (\underline{A}\, i : x \leq i < z : f.i.y \leq F)$$

Invariant Q states the relevant property of z which has been introduced for the purpose of efficiency. Note that neither $x := x+1$ nor $y := y-1$ falsifies Q, nor $z := z\ \underline{max}\ x$.

```
|[ x, y, z : int
; x, y, z, k := 0,0,0,0 ; do f.x.y ≤ F → y := y+1 od {P ∧ Q}
; do y ≥ 0 →
     if f.x.y < F → x := x+1 {P ∧ Q}
     ▯ f.x.y > F → y := y-1 {P ∧ Q}
     ▯ f.x.y = F → z := z max x {Q': (A i: x ≤ i < z: f.i.y = F)}
          ; do f.z.y = F → z := z+1 od {Q' ∧ f.z.y ≠ F}
          ; y, k := y-1, k+z-x {P ∧ Q}
     fi
  od
]|
```

We finally mention that it may be worthwhile to replace the linear searches by logarithmic ones. We could replace, for instance,

$$\underline{\text{do}}\ f.x.y \le F \rightarrow y := y+1\ \underline{\text{od}}$$

by

$\underline{\text{if}}\ f.x.y > F \rightarrow \text{skip}$
$[]\ f.x.y \le F \rightarrow$
$\quad |[\ v: \text{int};\ v := 1\ \{f.x.y \le F \text{ and } v \text{ is power of } 2\}$
$\quad ;\ \underline{\text{do}}\ F \ge f.x.(y+v) \rightarrow v := 2*v\ \underline{\text{od}}$
$\quad \{f.x.y \le F < f.x.(y+v) \text{ and } v \text{ is power of } 2\}$
$\quad ;\ \underline{\text{do}}\ v \ne 1 \rightarrow v := v/2$
$\qquad\qquad \underline{\text{if}}\ f.x.(y+v) \le F \rightarrow y := y+v$
$\qquad\qquad []\ F < f.x.(y+v) \rightarrow \text{skip}$
$\qquad\qquad \underline{\text{fi}}$
$\quad \underline{\text{od}};\ y := y+1\ \{f.x.(y-1) \le F < f.x.y\}$
$\quad]|$
$\underline{\text{fi}}$

The other three linear searches can be treated similarly. Note that the worst case remains linear, viz. when in the original execution we have a long execution of alternations of $x := x+1$ and $y := y-1$.

prof. dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
United States of America

Austin 5 Sep. 1985