The pattern matching problem.

The problem that is solved in this chapter is a very famous one, that has been tackled independently by many programmers. Yet we hope that our treatment gives some pleasure to even those of my readers who considered themselves thoroughly familiar with the problem and its various solutions.

We consider as given two sequences of values

$p(0), p(1), \ldots, p(N-1)$ with $N \geq 1$ and

$x(0), x(1), \ldots, x(M-1)$ with $M \geq 0$

(usually M is regarded as being many times larger than N). The question to be answered is: how many times occurs the "pattern", as given by the first sequence, in the second sequence?

Using $(\underline{N}\ i: 0 \leq i < m: B(i))$

to denote "the number of different values of i in the range $0 \leq i < m$, for which $B(i)$ holds", a more precise description of the final relation R that is to be established is

R: $\quad count = (\underline{N}\ i: 0 \leq i \leq M - N: match(i))$

where the function $match(i)$ is given by

for $0 \leq i \leq M - N$: $\quad match(i) = (\underline{A}\ j: 0 \leq j < N: p(j) = x(i + j))$

for $i < 0$ $\underline{or}$ $i > M - N$: $\quad match(i) = false$.

(To define $match(i) = false$ for those further values of i, thus making it a total function, is a matter of convenience.)

If we take as invariant relation

P1: $\quad count = (\underline{N}\ i:\ 0 \leq i < r:\ match(i))\ \underline{and}\ r \geq 0$

we have one which is trivially established by "count, r:= 0, 0" and, furthermore, is such that $\quad (P1\ \underline{and}\ r > M - N) \Rightarrow R$.

(The "matter of convenience" referred to above is that now the above inequality will do the job.) This gives a sketch for the program:

count, r:= 0, 0;

$\underline{do}\ r \leq M - N \rightarrow$ "increase r under invariance of P1 " $\underline{od}$,

and the reader is invited to work out for himself the refinement in which r is always increased by 1; worst case the time taken by the execution of that program will be proportional to $M * N$.

Depending on the pattern, however, much larger increases of r seem sometimes possible: if, for instance, the pattern is (1, 2, 3, 4, 5) and match(r) has been found to hold, "count, r:= count + 1, r + 5" would leave P1 invariant! Considering the invariant relation

P2: $\quad (\underline{A}\ j:\ 0 \leq j < k:\ p(j) = x(r + j))\ \underline{and}\ 0 \leq k \leq N$

(which can be expected to play a role in the repetitive construct computing match(r)), we can investigate, what we can gain by taking that relation outside the repetitive construct, i.e. we consider:

count, r, k:= 0, 0, 0;

$\underline{do}\ r \leq M - N \rightarrow$ "increase r under invariance of P1 $\underline{and}$ P2 " $\underline{od}$

(relation P2 being vacuously satisfied by $k = 0$).

In view of the validity of relation P2 and the formula for match(r) ,

the most natural thing to start the repeatable statement with, is to try to determine match(r) : as the truth of match(r) can be concluded from P2 and k = N we prescribe that k be increased as long as is necessary and possible:

$$\underline{\text{do}}\ k \neq N\ \underline{\text{cand}}\ p(k) = x(r + k) \rightarrow k := k + 1\ \underline{\text{od}} \quad , \qquad (1)$$

upon termination of which --and termination is guaranteed-- we have

$$P2\ \underline{\text{and}}\ (k = N\ \underline{\text{cor}}\ p(k) \neq x(r + k))$$

from which we can conclude that match(r) = (k = N) . Thus it is known, whether increasing r by 1 should be accompanied by "count:= count + 1" or not. We would like to know, by how much r can be increased without further increase of count and without taking any further x-values into account. (The taking into account of x-values is done in statement (1): to do so is its specific purpose! Here we are willing to exploit only properties of the --constant-- pattern.)

If $k = 0$, we conclude (because $N > 0$) that match(r) = false : the relation P1 then justifies an increase of r by 1 --leaving P1 invariant by leaving count unchanged-- but P2 does not justify any higher increase of r ; and $k = 0$ --making P2 vacuously true-- is maintained.

For general k , however, there is the following argument. Define for $0 < i \leq k \leq N$ the boolean function

$$dif(i, k) = (\underline{E}\ j: 0 \leq j < k - i;\ p(j) \neq p(i + j)) \quad .$$

From this it follows that dif(k, k) = false . If, however, dif(i, k) = true, we conclude --because $0 \leq i + j < k$-- on account of the truth of P2

$$(\underline{E}\ j: 0 \leq j < k - i:\ p(j) \neq x(r + i + j)) \quad ,$$

that is: dif(i, k) => non match(r + i) . Therefore, the variable "count" needs no further adjustments --besides the one on account of the value of match(r)-- when r is ~~de~~increased by d(k) , where d(k) is the minimum solution for i with $0 < i \leq k$ of the equation dif(i, k) = false , or:

$$(\underline{A}\ j: 0 \leq j < k - i: p(j) = p(i + j)) \quad . \quad (2)$$

The fact that d(k) is a solution of (2) implies

$$(\underline{A}\ j: 0 \leq j < k - d(k): p(j) = p(d(k) + j))$$

which, with P2 amounts to

$$(\underline{A}\ j: 0 \leq j < k - d(k): p(j) = x(r + d(k) + j))$$

and as a result --besides the adjustment of "count" as implied by the value of match(r) -- both P1 and P2 are kept invariant by "r, k:= r + d(k), k - d(k)".

Because the minimum solution of (2) depends on k and p only, we find:

```
begin glocon p, N, x, M; virvar count; privar r, k; pricon d;
   "initialize d";
   count, r, k := 0, 0, 0;
   do r ≤ M - N →
        do k ≠ N cand p(k) = x(r + k) → k:= k + 1 od;
        if k = N → count:= count + 1; r, k := r + d(k), k - d(k)
         [] 0 < k < N → r, k := r + d(k) , k - d(k)
         [] k = 0 → r:= r + 1
        fi
   od
end
```

The only job left is the initialization of the array variable d , i.e. to establish for each k satisfying $1 \leq k \leq N$ the minimum solution for i of (2) . The Linear Search Theorem tells us that we should try i-values in increasing order. It pays, however, to realize that this minimum value for i has to be determined for a whole sequence of k-values. Let $k1 > k2$ and let $d(k1)$ be the minimum solution for i of (2) with $k = k1$. From

$$(\underline{A}\ j: 0 \leq j < k1 - d(k1): p(j) = p(d(k1)+ j))\ \underline{\text{and}}\ k1 > k2$$

follows:

$$(\underline{A}\ j: 0 \leq j < k2 - d(k1): p(j) = p(d(k1)+ j))$$

i.e. for $k = k2$, $d(k1)$ is also a solution for i of (2) , but not necessarily the smallest! From that we conclude that $d(k)$ is a monotonically non-decreasing function of k. And the algorithm therefore investigates increasing values of i , each time deciding whether for one or more k-values $i = d(k)$ can be concluded (should be established). More precisely: let $j(i)$ for given value of i be the maximum value $\leq N - i$, such that

$$(\underline{A}\ j: 0 \leq j < j(i): p(j) = p(i + j)) \quad ;$$

then $d(k) = i$ for all k such that $k - i \leq j(i)$ (or: $k \leq j(i)+ i$), for which no solution $d(k) < i$ exists. As the values of i will be tried in increasing order and, as soon as existing as minimal solution, will be recorded in the monotonically non-decreasing function d , the condition is

$$d.hib < k \leq j(i)+ i$$

and we get the following program:

```
"initialize d":
begin glocon p, N; virvar d; privar i;
   d vir int array, i vir int := (1), 0;
   do d.hib ≠ N →
      begin glocon p, N; glovar d, i; privar j;
         j vir int := 0; i:= i + 1;
         do j < N - i cand p(j) = p(i + j) → j:= j + 1 od;
         do d.hib < j + i → d:hiext(i) od
      end
   od
end
```

Exercise 1. Give a formal correctness proof for the above initialization. (End of exercise 1.)

Exercise 2. With "r, k := r + d(k), r ÷ d(k)" for $0 < k$, our algorithm adjusts r and k without changing r + k. Investigate the slight gain that is possible for $0 < k < N$ if it is known that the x-values are two-valued. (End of exercise 2.)

Remark. Our final algorithm is one, the execution time of which I regard as to grow proportional to M + N. Once one has set oneself the goal to find --if possible-- an algorithm with such a performance, its actual development does not seem to require much more than the usual care; the crucial point seems the refusal to be satisfied (without further investigation) with the obvious M * N -algorithm, the development of which I have left as an exercise to the reader. A slight reformulation of the problem, however, enables us to recognize also here a general design principle, which might be called "The

Search for the Small Superset." Suppose that we had not been asked to count the number of matches, but to generate the sequence of r-values, for which match(r) holds.

When a program has to generate the members of a set A , there are (roughly) only two situations. Either we have simple, straightforward "successor function" by means of which a next member of A can be generated --and then the whole set can be trivially generated by means of repeated application of that successor function-- or we do not have a function like that. In the latter case, the usual technique is to generate the members of a set B instead, where

a) each member of A is a member of B as well

b) there exists a generator for successive members of B

c) there exists a test whether a member of B belongs to A as well.

The algorithm then generates and inspects all members of B in turn.

If this technique is to lead to a satisfactory performance, three conditions should be satisfied:

1) the members of set B should be reasonably efficient to generate

2) the test whether an element of B belongs to A as well --particularly in the case that it does <u>not</u>, for, usually, B is an order of magnitude larger than A -- should be reasonably efficient

3) set B should not be unnecessarily large.

The trained problem solver, aware of the above, will consciously look for a smaller set B than the obvious one. In this example, the set of <u>all</u> r-values satisfying $0 \leq r \leq M - N$ is the obvious one. Note, that in the

previous chapter "An exercise attributed to R.W.Hamming" the replacement of the set "qq" by the much smaller set "qqq" was another application of the principle of The Search for the Small Superset. And besides "taking a relation outside the repetitive construct" this illustrates the second strategical similarity between the solutions presented in the current and in the previous chapter. (End of remark.)