Uncompleted and uncorrected.

On a problem from Aho, Hopcroft and Ullman.

This text records my experience with a problem to which Ir.W.H.J.Feijen drew my attention. He suggested that I should tackle it and try to construct the most beautiful solution I could think of. (In addition, he told me, that Aho, Hopcroft and Ullman classified the problem as very easy.) As it was a problem I had never thought about before, I decided to accept the challenge.

Given a sequence of M elements A(1) through A(M) we can form "a subsequence of length s " from it by removing M-s elements and retaining the remaining s elements in their original order. Given two sequences A(1) through A(M) and B(1) through B(N) , it is asked to make a program --assuming that each element has, say, an integer value-- the maximum length of a common subsequence, in other word: the maximum length of a sequence that is both a subsequence of the A's and of the B's . So much for the problem. (As usual I shall identify the order from left to right with that of increasing subscript value.)

An obvious way to try to solve this problem is via "induction" over one of the lengths. With our final relation

R: k = the maximum length of a subsequence common to A(1) through A(M) and to B(1) through B(N)

we could try an invariant relation

P(k, n): k = the maximum length of a subsequence common to A(1) through A(M) and to B(1) through B(n)

which is easily initialized for small n (=0 or =1). Because

$$(P(k, n) \text{ and } n = N) \Rightarrow R \quad ,$$

we should then try to increase n under invariance of P . I am slightly repelled by the asymmetry of this attack; yet it seems a simple thing to try first and it may, at least, give me some familiarity with the problem.

Suppose that P(k, n) holds for a certain value of n . After n:= n + 1 we have P(k, n-1) . The question then becomes: how changes k when we extend B(1) through B(n-1) with B(n) ? This extension will increase k

when the "new" element $B(n)$ can be used in a longer subsequence, i.e. iff we can find an $A(i) = B(n)$ with $i >$ the ordinal number of the rightmost element of a subsequence of length k from $A(1)$ through $A(M)$ that is common to $B(1)$ through $B(n-1)$. This means that we have to know the leftmost position of the rightmost element of such a common subsequence. If we don't find such an element, k remains constant: it may, however, move the leftmost position of such a rightmost element. We get the same argument as in EWD591 --I am getting a suspicion why Feijen posed this problem to me-- and we add to our invariant relation:

for $1 \leq j \leq k$, $m(j)$ = the <u>minimal</u> value, such that $A(m(j))$ is the rightmost element of a subsequence of length j from $A(1)$ through $A(M)$ that also occurs in $B(1)$ through $B(n)$.

Similarly to EWD591 we observe that $m(j)$ is an increasing function:

$$1 \leq i < j \leq k \;\Rightarrow\; m(i) < m(j) \quad .$$

Let us now consider the adjustment of the array m. If

$$(\underline{A}\ i:\ 1 \leq i \leq M:\ A(i) \neq B(n)) \quad ,$$

the array m can remain unchanged. (This reflects that, to start with, we could have pruned the sequences by removing all elements with values that don't occur in the other sequence.)

Suppose, next, that the equation $A(i) = B(n)$ has <u>one</u> solution for i, and let j satisfy $m(j-1) < i < m(j)$. In that case, a sequence of $j-1$ elements, taken from $A(1)$ through $A(m(j-1))$ matches one taken from $B(1)$ through $B(n-1)$, and, as $A(i) = B(n)$, we have found a common subsequence of length j with $A(i)$ as its rightmost element. The adjustment needed can then be done by $m:(j)= i$. In order to capture the case $j = 1$ as well, we can extend the array variable at the low side with $m(0) = 0$. In the case that the only solution of $A(i) = B(n)$ satisfies $i > m(k)$, a longer match is possible: k should be increased by 1 and m could be adjusted by m:hiext(i). In order to treat this case just as the others, I suggest that we extend the array m beforehand with $m(k+1) = M + 1$; this is the normal coding trick, it is this time as if we extend the array 'A temporarily with the value $B(n)$, kind of "forcing the match".

Some care is needed --and it is here that I tend to disagree with the "very easy" of A., H., and U. (but our standards may differ; alternatively I might be working on a clumsy solution)-- in the case that $A(i) = B(n)$ admits more than one solution. When establishing a j such that

$$m(j-1) < i < m(j)$$

it is essential for the justification of the adjustment $m:(j)= i$ that the value $m(j-1)$ still refers to a match with respect to $B(1)$ through $B(n-1)$, i.e. if we update the array m elementwise, we must do so in the order of decreasing subscript value. The adjustment of $m(j)$ may be done for values of j satisfying $m(j-1) < i \leq m(j)$; in the following program such a value of j is found by means of the usual binary search. Without declarations the following program would do:

```
n:= 0; m:= (0, 0, M+1);
do n ≠ N → n:= n + 1; x:= B(n);
          i, j := M, m.hib;
          do i ≠ 0 → if A(i) ≠ x → skip
                     [] A(i) = x → l:= m.lob;
                                   do l + 1 ≠ j → h:= (l + j) div 2;
                                                  if m(h) < i → l:= h
                                                   [] i ≤ m(h) → j:= h
                                                  fi
                                   od;
                                   m:(j)= i
                     fi;
                     i:= i - 1
          od;
          do m.high ≠ M + 1 → m:hiext(M + 1) od
do;
k:= m.hib - 1
```

Note how, when $A(i) = B(n)$ has more solutions for i, the starting area for the binary search shrinks. It is an algorithm of the type $M*N*(\log \text{something})$, and, being the only solution I have found so far, it is also the most beautiful one I have found so far.

* * *

The time-consuming part of the above process is the scan (proportional to M) in order to find the solutions for i of the equations $A(i) = x$, because it has to be done N times. A preliminary sorting of the A-values (with their original positions) which is of the order $M*(\log \text{something})$ allows us to replace the scan proportional to M by a binary search proportional to $\text{Log } M$, and the resulting algorithm should be of the order $(M+N)*(\log \text{something})$. At the moment I don't feel inclined to code that.

* * *

Although I don't expect to find a faster algorithm, for the sake of its potential elegance I would like to do at least a preliminary investigation of a more symmetric solution, with a kind of induction on k.

Let $V(k)$ be the set of all pairs (m, n), such that $A(1)$ through $A(m)$ and $B(1)$ through $B(n)$ are a pair of "shortest" sequences containing a common subsequence of length k, "shortest" in the sense that decreasing either m or n or both would result in a pair no longer containing a common subsequence of length k. The initialization is no problem, because $V(0) = \{(0, 0)\}$. Suppose that from a nonempty $V(k)$ we can deduce $V(k+1)$. If $V(k+1)$ is empty, we have found k; otherwise we proceed (after $k := k + 1$, etc.)

If the set $V(k)$ contains the pairs $(m1, n1)$ and $(m2, n2)$, it is not difficult to prove that

$$((m1, n1) \neq (m2, n2)) \Rightarrow (m1 \neq m2 \text{ and } n1 \neq n2) \qquad \text{and}$$

$$m1 < m2 \Rightarrow n1 > n2 \quad ,$$

i.e. the pairs can be uniquely ordered in the order of increasing m; they are then also ordered in the order of decreasing n. This is the best I can say about the pairs (m, n) in the set $V(k)$, and we could have a look whether it can be exploited in the construction of $V(k+1)$. In order to avoid notational confusion, $V(k+1)$ will be described as composed of pairs (a, b) in exactly the same way as $V(k)$ is composed of pairs (m, n). Let us build up the pairs (a, b) in the order of increasing "a" (i.e. decreasing "b"). For "a" we have to investigate the sequence $m1+1, \ldots, M$ in that order, where $m1$ is the smallest m-value. Let $\tilde{a}$ be the current value under investigation, let mi be the largest m-value, satisfying $mi < \tilde{a}$. Then we search for the smallest b, such that $A(\tilde{a}) = B(b)$ and $b > ni$, where (mi, ni) is a pair

from $V(k)$. So, as $\tilde{a}$ increases, the lower bound for the corresponding b decreases. On the other hand, because we have to find the b's in decreasing order, the upper bound of the area to be searched, as given by

$$b < \min\ (N+1,\ \text{the } b\text{'s found so far})$$

moves down as $V(k+1)$ grows. It is not clear at all that this strategy reduces the number of comparisons of A-values with B-values, I don't feel tempted to code it, and give up.