Presenting an Algorithm to Find the Minimum Edit Distance

David Gries*
Bill Burkhardt

88-903
March 1988

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This work was supported by the NSF under grant DCR-8320274. The work was done while the second author, an undergraduate at Colgate, was spending the summer at Cornell under the NSF's program for Research Experience for Undergraduates.

Presenting an Algorithm to Find the Minimum Edit Distance

David Gries and Bill Burkhardt[1]

Computer Science Department, Cornell University

August 1987

0. Introduction

Article [0] describes a neat algorithm for finding the minimum set of editing operations to transform a given sequence of characters A (= $[A_0, A_1, ..., A_{\#A-1}]$) into a given sequence of characters B, where an editing operation is a deletion of a character from A or the insertion into A of a character of B. If k is the minimum number of editing operations required, which is bounded above by $\#A+\#B$, then the algorithm takes time proportional to $k * min(\#A, \#B)$, so the worst-case time is $O(\#A*\#B)$. The expected time, given certain distributional assumptions, is $k^2 + min(\#A, \#B)$. Thus, the algorithm is efficient and compares favorably with the algorithm used for *DIFF* in UNIX. Further, it is trivial to modify the algorithm to terminate upon detecting that more than a given number K of edit operations are required.

In [0], the algorithm is described almost completely in terms of an example. Our purpose here is to give another, more rigorous and precise, description.

We first present an algorithm to determine just the minimum *number* of editing operations required to transform A into B; we then discuss modifying it to also determine a corresponding editing sequence.

1. Specification

Consider the following function m, defined for $-1 \le r \le \#A$ and $-1 \le c \le \#B$:

$$(0)\quad m(r,c) = \begin{cases} -1 \le r \le 0 \ \vee\ -1 \le c \le 0 & \rightarrow r+c \\ 0 < r \le \#A \ \wedge\ 0 < c \le \#B \ \wedge\ A[r-1] = B[c-1] & \rightarrow m(r-1, c-1) \\ 0 < r \le \#A \ \wedge\ 0 < c \le \#B \ \wedge\ A[r-1] \ne B[c-1] & \rightarrow min(m(r-1,c), m(r,c-1)) + 1 \end{cases}$$

We claim that, for $0 \le r$ and $0 \le c$, $m(r,c)$ is the minimum number of editing operations needed to transform $A[0..r-1]$ into $B[0..c-1]$. (The values $m(r,-1)$ and $m(-1,c)$ are needed only to dispense more easily with boundary conditions.) This can be seen as follows. First, to transform $A[0..-1]$ (i.e. the empty sequence) into $B[0..c-1]$ requires inserting the first c characters of B, which is c editing operations, and $m(0,c) = c$. And, to transform $A[0..r-1]$ into $B[0..-1]$ requires deleting all characters of $A[0..r-1]$, which is r editing operations, and $m(r,0) = r$.

Now consider $m(r,c)$ for $0 < r$ and $0 < c$. If $A[r-1] = B[c-1]$, transforming $A[0..r-1]$ into $B[0..c-1]$ is the same as transforming $A[0..r-2]$ into $B[0..c-2]$; this is reflected in the second line of definition (0). Finally, suppose, that $A[r-1] \ne B[c-1]$. Then the minimum number of editing operations is performed in one of the following two ways:

[1] This work was supported by the NSF under grant DCR-8320274. The work was done while the second author, an undergraduate at Colgate, was spending the summer at Cornell under the NSF's program for Research Experience for Undergraduates.

transform $A[0..r-2]$ into $B[0..c-1]$ and delete $A[r-1]$, or
transform $[0..r-1]$ into $B[0..c-2]$ and append $B[c-1]$.

In this case, the third line of (0) gives the minimum number of editing operations.

Since $m(r,c)$ is indeed the minimum number of editing operations needed to transform $A[0..r-1]$ into $B[0..c-1]$, our problem is only to develop an algorithm to compute $m(\#A,\#B)$, which we now do. In doing so, we rely solely on definition (0) of m and forget completely about its interpretation.

1. Analysis

One approach to calculating $m(\#A,\#B)$ is to begin with $m(0,0)=0$ and calculate $m(r,c)$ for increasing values of r and c until $m(\#A,\#B)$ has been calculated. In the worst case, this can take time $\#A*\#B$, but, by calculating as few of the m values as possible, we hope to do better on the average. We now analyze m to unearth some facts. Consider m to be a matrix $m(-1..\#A,-1..\#B)$. Our calculations will deal with the *diagonals* of the matrix, as shown in Fig. 0.

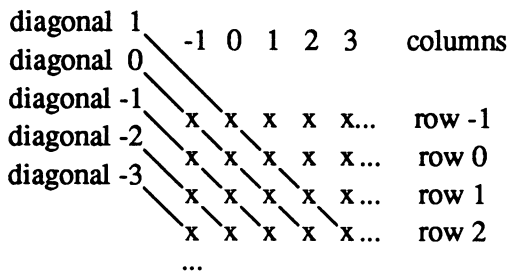


Figure 0. Matrix m with diagonals

Row r, column c, and diagonal d are related by the formula $d=c-r$, or $c=r+d$. We denote the matrix element at row r and diagonal d by

$$m(r \mid d).$$

Thus, $m(r \mid d)$ is the element $m(r,r+d)$.

With these preliminaries, we have the following facts. The reader may wish to look at the small example in Fig. 1 as a means of reaffirming understanding of these facts.

	−1	0	1	2	3	A
row −1	−2	−1	0	1	2	
row 0	−1	0	1	2	3	
row 1	0	1	2	1	2	a
row 2	1	2	3	2	1	b
row 3	2	3	2	3	2	c
$\#B$			c	a	b	

Figure 1. The matrix m for A = (a b c) and B = (c a b)

(1) **Lemma.** The elements of diagonal d have the same parity as d: $even(d) \equiv even(m(r \mid d))$.

Proof. The theorem holds trivially for $-1 \le r \le 0$ or $-1 \le c \le 0$ and is proved inductively on $r+c$ for $r > 0$ and $c > 0$, using the fact that definition (0) defines $m(r \mid d)$ as either $m(r-1 \mid d)$ or one more than one of its adjacent values horizontally to the left and vertically above. □

(2) **Lemma.** Adjacent values of m differ by at most 1, and each diagonal is nondecreasing and increases by at most 2 with each element. □

Proof. The proof is relegated to the Appendix.

(3) **Corollary.** The first two values on diagonal d are $abs(d)$–2 and $abs(d)$, and the values on d increase by exactly 0 or 2 with each element.

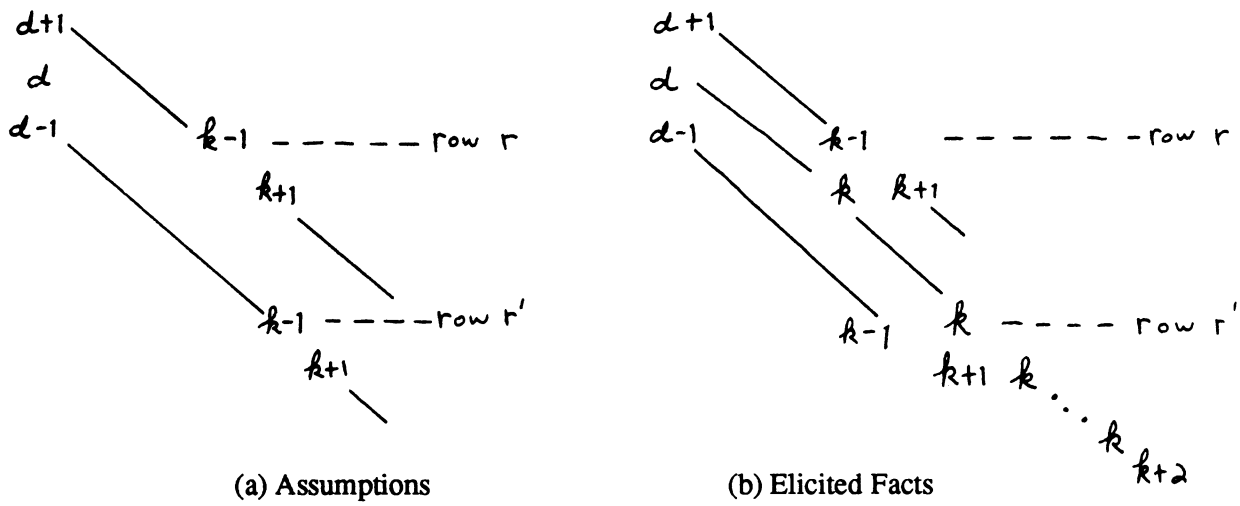
Proof. That the first two values are as stated comes directly from the definition of m. By Lemma (2) the increase with each element is 0, 1, or 2; and by Lemma (1) the increase cannot be 1. □

(4) **Corollary.** Adjacent values of m differ by exactly 1.

Proof. This follows from lemmas (2) and (1). □

Let us now discuss calculating an element of m, given previously calculated values. It will be advantageous to calculate a value along a diagonal d as follows. Assume the following about the diagonals $d-1$ and $d+1$ (see Fig. 2a):

(5) (*a*) $m(r \mid d+1)=k-1$,
(*b*) $m(r+1 \mid d+1)=k+1$ (or $r=\#A$ or $r+d=\#B$),
(*c*) $m(r' \mid d-1)=k-1$,
(*d*) $m(r'+1 \mid d-1)=k+1$ (or $r=\#A$ or $r+d=\#B$).



(a) Assumptions (b) Elicited Facts

Figure 2. Assumptions of Lemma 5 and the facts elicited from them

We now elicit from these assumptions several facts, which lead to the calculation of the last row for diagonal d for which the m-value is k (see Fig. 2b). First, consider the value $m(r+1 \mid d)$. Since adjacent values differ by 1, because $m(r \mid d+1) = k-1$, it can be only $k-2$ or k. Because $m(r+1 \mid d+1) = k+1$, it can be only k or $k+2$. Hence, $m(r+1 \mid d)=k$. Similarly, $m(r' \mid d)=k$. Letting $t=max(r+1, r')$; we have $m(t \mid d)=k$.

Now, suppose $m(t+1 \mid d)=k$. Because $m(t \mid d+1) \geq k+1$ and $m(t+1 \mid d-1) \geq k+1$ and diagonals are nondecreasing, by definition (0) we have $A[t]=B[t+d]$. Similarly, under the condition $m(t+1 \mid d)=k$, we have $m(t+2 \mid d)=k$ iff $A[t+1]=B[t+1+d]$, and so on. Thus, for each additional member $m(t+j \mid d)$ of diagonal d, $m(t+j \mid d)=k$ iff the previous element $m(t+j-1 \mid d)$ equals k and $A[t+j]=B[t+j+d]$.

We have outlined the proof of the following important lemma, which shows how to calculate the position of the last element with value k on diagonal d given the positions on the adjacent diagonals of the last elements with value $k-1$, and very efficiently.

(6) **Lemma.** Suppose (5) holds. Let $t=max(r+1, r')$. Let

(7) $\quad t'=MIN(j: t \leq j: j=\#A \vee j+d=\#B \vee A[j] \neq B[j+d])$.

Then $m(t' \mid d)=k$ and $m(t'+1 \mid d)$, if it exists, equals $k+2$. □

2. The essence of the algorithm

Suppose we have the situation given in Fig. 3. Given is a set S of alternating diagonals on which the last positions with value $k-2$ are known. Further, the last position of the value $k-1$ on each adjacent diagonal is known. Then Lemma (6) shows us how to calculate the last position of the value k on all the diagonals of S. Doing this and adding 1 to k leaves the situation of Fig. 3 true, but with k being one more than it was. Thus, we envision an algorithm that at iteration k calculates the last position of the value k on all diagonals. By Lemma 3, we need only consider the diagonals in the range $-k-1..k+1$. Further, we have $m(-1,k)=k-1$ and $m(k,-1)=k-1$.

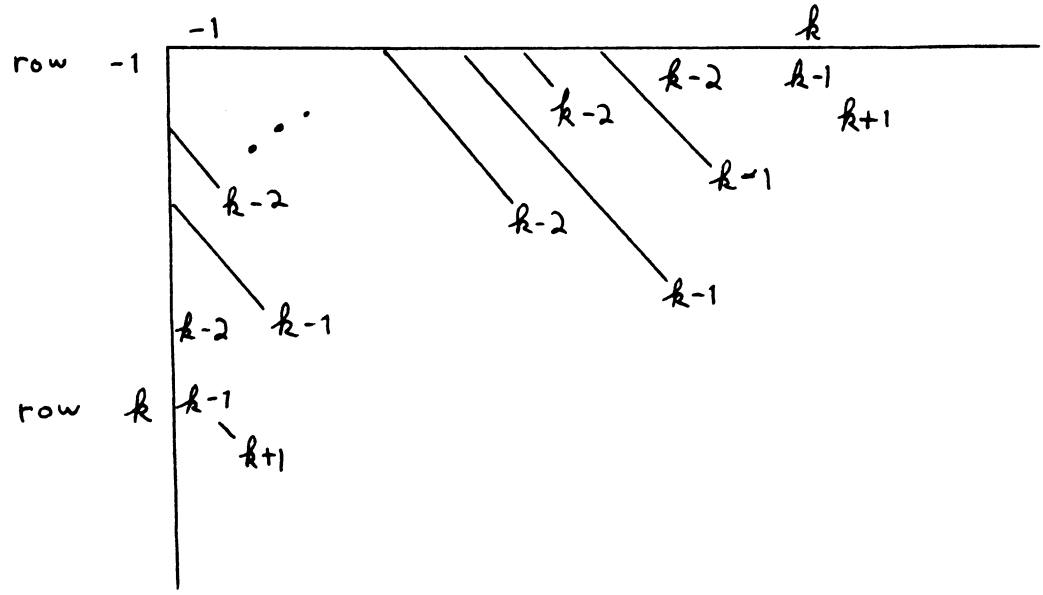


Figure 3. The scheme at an iteration

Note that the complete matrix m is not needed. Instead, for each diagonal d, we need to know only the row number of the last value $k-1$ or $k-2$ (depending on whether $k+d$ is odd or even, since all values on a diagonal have the same parity). Thus, we use an array $e[-(\#A+1)..\#B+1]$, where $e[d]$ contains this row number, and the space used by the algorithm will be proportional to $max(\#A, \#B)$.

Let us now be more rigorous. The algorithm uses an integer variable k, whose bounds are given by invariant $P0$:

$$P0:\ 0 \le k \le max(\#A, \#B)+1$$

Array $e[-(\#A+1)..\#B+1]$ contains in $e[d]$ a row number for which a value on diagonal d is known; more specifically:

$$P1a:\ \forall(d:\ -k<d<k \wedge even(k+d):\ m(e[d] \mid d)=k-2 \wedge m(e[d]+1 \mid d)=k)$$
$$P1b:\ \forall(d:\ -k<d<k \wedge odd(k+d):\ m(e[d] \mid d)=k-1 \wedge m(e[d]+1 \mid d)=k+1)$$

Thus, we plan the algorithm as a loop of the form

(8) $k := 0;$
{invariant: $P0 \wedge P1a \wedge P1b$}
do ... $\rightarrow$ Establish $(P1a \wedge P1b)^k_{k+1}$; $k := k+1$ **od**

$P0$, $P1a$, and $P1b$ are initially established by $k := 0$, since the ranges of the quantifications are then empty.

We now show how to establish $(P1a \wedge P1b)^k_{k+1}$ under the assumption that $P1a$ and $P1b$ hold and that all subscripts remain in bounds —boundary cases will be treated later. First, we show that $P1b$ implies $P1a^k_{k+1}$, so nothing need be done to establish $P1a^k_{k+1}$:

$$\begin{aligned}
& P1a^k_{k+1} \\
=\ & \forall(d:\ -(k+1)<d<k+1 \wedge even(k+1+d):\ m(e[d] \mid d)=k+1-2 \wedge m(e[d]+1 \mid d)=k+1) \\
=\ & \quad \{\text{arithmetic; for } -k=d \text{ and } k=d,\ k+1+d \text{ is not even}\} \\
& \forall(d:\ -k<d<k \wedge odd(k+d):\ m(e[d] \mid d)=k-1 \wedge m(e[d]+1 \mid d)=k+1) \\
=\ & P1b
\end{aligned}$$

Now consider $P1b^k_{k+1}$:

$$\begin{aligned}
& P1b^k_{k+1} \\
=\ & \quad \{arithmetic\} \\
& \forall(d:\ -k \le d \le k \wedge even(k+d):\ m(e[d] \mid d)=k \wedge m(e[d]+1 \mid d)=k+2)
\end{aligned}$$

We can calculate new values $e[d]$ to satisfy this by using Lemma 6, provided we know, for each diagonal adjacent to one of the pertinent diagonals d, the position of the last element in that diagonal that equals $k-1$, i.e. if we know

(9) $$\forall(d:\ -(k+1) \le d \le k+1 \wedge odd(k+d):\ m(e[d] \mid d)=k-1 \wedge m(e[d]+1 \mid d)=k+1)$$

Splitting the range of (9) into the three parts $d=-(k+1)$, $-(k+1)<d<k+1$, and $d=k+1$ yields

$$\begin{aligned}
& \forall(d:\ -k<d<k \wedge odd(k+d):\ m(e[d] \mid d)=k-1 \wedge m(e[d]+1 \mid d)=k+1) \ \wedge \\
& m(e[-k-1] \mid -k-1)=k-1 \wedge m(e[-k-1]+1 \mid -k-1)=k+1) \ \wedge \\
& m(e[k+1] \mid k+1)=k-1 \wedge m(e[k+1]+1 \mid k+1)=k+1))
\end{aligned}$$

The first conjunct of this formula is $P1b$, which is assumed to be true. Consider the second conjunct. Since, by definition (0) of m,

$m(r \mid -r) = m(r,0) = r$
$m(r-1 \mid -r) = m(r-1,-1) = r-2$

the second conjunct is satisfied by $e[-k-1] = k$. Similarly, the third conjunct is satisfied by $e[k+1] = -1$. Hence, the last two conjuncts can be established by $e[k-1], e[k+1] := -k-1, -1$. Hence, $P\,1b^k_{k+1}$ is established by algorithm (10), where the loop simply computes each of the new $e[d]$ using Lemma (6). Since (10) leaves $P\,1a^k_{k+1}$ invariantly true, (10) is the desired refinement of Establish $(P\,1a \wedge P\,1b)^k_{k+1}$.

(10) Establish $(P\,1a \wedge P\,1b)^k_{k+1}$:
$e[-k-1], e[k+1] := k, -1;$
$d := -k;$
do $d \le k \rightarrow$ Calculate new $e[d]$:
var $t := max(e[d+1]+1, e[d-1]);$
$e[d] := MIN(j:\ t \le j:\ j = \#A$ **cor** $j+d = \#B$ **cor** $A[j] \neq B[j+d]);$
$d := d+2$
od

3. The complete algorithm to calculate m(#A , #B).

The previous section gives the essence of the algorithm, but it fails to deal with boundary problems and termination. In this section, we modify the algorithm consisting of (8) and (10) to deal with these problems. Remember, throughout this section, the basic notion is the same, and we are only taking care of these problems.

We are interested only in determining $m(\#A, \#B)$, and not in computing all values of m. Consider Fig. 4. Suppose the last value on diagonal $\bar{d}+1$ is known, i.e. it is known that $e[\bar{d}+1] + \bar{d} + 1 = \#B$ and the last m-value in diagonal $\bar{d}+1$ is $k-1$. Then there is no need to calculate more values on diagonals above $\bar{d}$. Similarly, we can use a value $\underline{d}$. Thus, the range $\underline{d}..\bar{d}$ denotes the set of diagonals whose last value is not known, and thus bounds the band of diagonals to be considered.

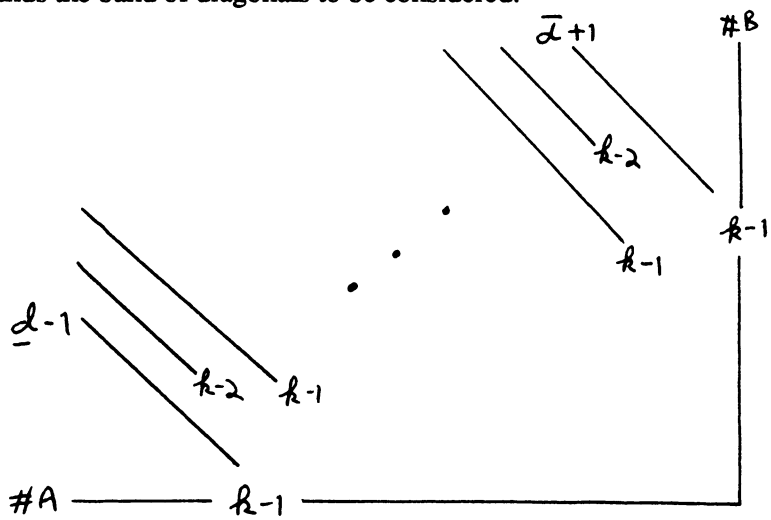


Figure 4. Bounding the diagonals under consideration

We now modify the invariants of the algorithm to take $\bar{d}$ and $\underline{d}$ into account. First, k, $\bar{d}$ and $\underline{d}$ are bounded as follows:

$P0$: $0 \le k \le max(\#A, \#B)+1$,
$-\#A \le \underline{d} \le \#B-\#A+1 \ \wedge \ (-k < \underline{d} \Rightarrow even(k) = even(\underline{d}))$,
$\#B-\#A-1 \le \bar{d} \le \#B \ \wedge \ (k > \bar{d} \Rightarrow even(k) = even(\bar{d}))$

Next, in the original $P1$, variable k indicated the band of diagonals for which values of e had been calculated. We now use $\underline{d}$ and $\bar{d}$ to indicate this:

$P1a$: $\forall(d: max(-k, \underline{d}-1) < d < min(k, \bar{d}+1) \wedge even(k+d): m(e[d] \mid d) = k-2 \wedge m(e[d]+1 \mid d) = k)$
$P1b$: $\forall(d: max(-k, \underline{d}-1) < d < min(k, \bar{d}+1) \wedge odd(k+d): m(e[d] \mid d) = k-1 \wedge m(e[d]+1 \mid d) = k+1)$

We also want to state that the values at the ends of the diagonals d referred to in $P1a$ and $P1b$ are unknown. Defining

$lastrow(d) = (e[d] = \#A)$ and
$lastcol(d) = (e[d]+d = \#B)$

and assuming that unassigned values of e have the value -1 (for example), we can state this as

$P2$: $\forall(d: \underline{d} \le d \le \bar{d}: \neg lastrow(d) \wedge \neg lastcol(d))$

As long as $k \le \bar{d}$, we are essentially in the same situation as in the previous section with respect to the upper bound of the band of diagonals to be calculated. However, if $k > \bar{d}$, then the value $e[\bar{d}+1]$ is known; further, the corresponding column number is $\#B$, so that the boundary has been reached (see Fig. 4). A similar statement is to be made for $\underline{d}$, so that we have

$P3a$: $k > \bar{d} \Rightarrow lastcol(\bar{d}+1) \wedge m(e[\bar{d}+1] \mid d+1) = k-1$
$P3b$: $-k < \underline{d} \Rightarrow lastrow(\underline{d}-1) \wedge m(e[\underline{d}-1] \mid \underline{d}-1) = k-1)$

By $P3$ and $P0$, the value $m(\#A, \#B)$, which is in diagonal $\#B-\#A$, is known iff $\#B-\#A < \underline{d}$ or $\bar{d} < \#B-\#A$. Hence, execution should continue as long as $\underline{d} \le \#B-\#A \le \bar{d}$. Upon termination, from $P3$ we have $m(\#A, \#B) = k-1$.

It is easy to see that the initialization in algorithm (11) below establishes all the invariants. Comparing the body of the loop of (11) with the algorithm of the previous section, we see that the following modifications have occurred. First, the calculations of $e[-k-1]$ and $e[k+1]$ have been guarded so that they occur only when necessary. Second, the loop bounds in the calculation of the new $e[d]$ have been changed to reflect the use of variables $\bar{d}$ and $\underline{d}$ in defining the band of diagonals still to be processed. Third, $\bar{d}$ and $\underline{d}$ are changed as required whenever an element of the last row or column of m is determined.

```
(11)  k, d, d̄ := 0, -#A, #B;
      do d ≤ #B-#A ≤ d̄ →
         {Calculate e[k+1] and e[-k-1] if necessary:}
            if d̄ ≥ k  then e[k+1] := -1;
            if d ≤ -k  then e[-k-1] := k;
         {Calculate the new e[d] in the band d...d̄:}
            d := max(-k, d);
            do d ≤ min(k, d̄) → var t := max(e[d+1]+1, e[d-1]);
                               e[d] := MIN(j: t≤j: j=#A cor j+d=#B cor A[j]≠B[j+d]);
                               {Update d̄ and d:}
                                  if lastcol(d) then d̄ := d-1;
                                  if lastrow(d) then d := d+1;
                               d := d+2
            od;
         k := k+1
      od
```

4. Extending the algorithm to compute the edit sequence.

The algorithm given in Sect. 3 calculates only the minimum number of edit operations to transform A into B. To modify it to compute a sequence of edit operations of minimum length as well, we do the following. With each element $e[d]$ maintain an array $s[d]$ of edit operations that transform $A[0..e[d]-1]$ into $B[0..e[d]+d-1]$. If $A[r-1]=B[c-1]$, then the editing operations needed to transform $A[0..r-1]$ into $B[0..c-1]$ is the same as those to transform $A[0..r-2]$ into $B[0..c-2]$. If not, then $A[0..r-1]$ can be transformed into $B[0..c-1]$ in one of two ways, as outlined in Sect. 1, and we need only record which way is used. Thus, for each assignment to e we have the following additions:

$e[-k-1] := k$	$s[-k-1] := [\,]$
$e[k+1] := -1$	$s[k+1] := [\,]$
$t := e[d+1]+1;\ e[d] := min(...)$,	$s[d] := s[d+1]$ ^ 'Delete $A[e[d]-1]$'
$t := e[d-1];\ e[d] := min(...)$,	$s[d] := s[d-1]$ ^ 'Insert $B[e[d]+d-1]$'

The statements 'Delete $A[e[d]-1]$' and 'Insert $B[e[d]+d-1]$' refer always to elements of the *original* sequences A and B. Detailed information on minimizing the space needed for the sequences $s[d]$ is given in [0]. For example, it should be clear that sequences only for alternate diagonals are needed.

5. Acknowledgements. Thanks go to Doug McIlroy for bringing paper [0] to our attention and for criticisms of a draft of this manuscript and to Gary Levin for a very careful and constructive reading.

6. References.

[0] Miller, W., and Myers, E.W. A file comparison algorithm. *Software—Practice and Experience 15*, 11 (November 1985), 1025-1040.

Appendix

(2) **Lemma.** Adjacent values of m differ by at most 1, and each diagonal is nondecreasing and increases by at most 2 with each element:

(a) $m(r-1,c-1) \le m(r,c) \le m(r-1,c-1)+2$,
(b) $m(r-1,c)-1 \le m(r,c) \le m(r-1,c)+1$, and
(c) $m(r,c-1)-1 \le m(r,c) \le m(r,c-1)+1$

Proof. The proof is by induction on $r+c$. The base cases $-1 \le r, c \le 0$ are trivial —note that (a) and (c) are inapplicable if $r=-1$ and that (a) and (b) are inapplicable if $c=-1$.

We first prove (a). If $A[r-1]=B[c-1]$, then by definition (0) $m(r-1,c-1) = m(r,c)$ and (a) holds. Suppose $A[r-1] \neq B[c-1]$, and assume that, by definition (0), $m(r,c)=m(r-1,c)+1$ (the case $m(r,c)=m(r,c-1)+1$ is similar and is omitted). We have, by induction using (c),

$$
\begin{aligned}
& m(r-1,c-1)-1 \le m(r-1,c) \le m(r-1,c-1)+1 \\
= \; & \{\text{Use assumption } m(r,c)=m(r-1,c)+1 \text{ and substitute for } m(r-1,c)\} \\
& m(r-1,c-1)-1 \le m(r,c)-1 \le m(r-1,c-1)+1 \\
= \; & \{\text{Add 1 to each expression}\} \\
& \text{(a)}.
\end{aligned}
$$

We now prove the first inequality of (b):

$$
\begin{aligned}
& m(r-1,c)-1 & \\
\le \; & m(r-1,c-1)+1-1 & \{\text{Induction, second inequality of (c)}\} \\
\le \; & m(r,c)+1-1 & \{\text{First inequality of (a)}\} \\
= \; & m(r,c) & \{\text{Arithmetic}\}
\end{aligned}
$$

We now prove the second inequality of (b). If $m(r,c)=m(r-1,c-1)$, then it follows, by induction, from the first inequality of (c):

$$
\begin{aligned}
& m(r,c) & \\
= \; & m(r-1,c-1) & \{\text{Assumption}\} \\
\le \; & m(r-1,c)+1 & \{\text{Induction, first inequality of (c)}\}
\end{aligned}
$$

If $m(r,c) \neq m(r-1,c-1)$, then by definition (0) $m(r,c)$ is the minimum of $m(r-1,c)+1$ and $m(r,c-1)+1$, so the second inequality of (b) holds.

The proof of (c) is similar to the proof of (b) and is omitted. □