Genominformatik

Lecture notes for a course in the Sommersemester 2014

Stefan Kurtz

April 7, 2014

Contents

1 Fast computation of the Unit Edit Distance **1**
1.1 Properties of the unit edit distance 1
1.2 A lazy evaluation strategy 3
1.3 Diagonalwise computation 4
1.4 An output sensitive algorithm 5

2 Chaining of Matches **13**
2.1 Anchor based sequence comparison 13
2.2 The match chaining problem 14
2.3 A simple chaining algorithm 16
2.4 An efficient match chaining algorithm 18

3 Suffix Trees **25**
3.1 Motivation 25
3.2 The Concept of Suffix Trees 25
3.3 An Informal Introduction to Suffix Trees 26
3.4 A Formal Introduction to Suffix Trees 27
3.5 The Role of the Sentinel Character 28
3.6 The Size of Suffix Trees 29
3.7 The Write Only Top Down Suffix Tree Construction 30
3.8 Suffix Tree Applications 32
3.8.1 Searching for Exact Patterns 32
3.8.2 Approximate Matching of Patterns against a suffix tree 33
3.8.3 Minimal Unique Substrings 36
3.8.4 Maximal Unique Matches 38
3.8.5 All-pairs suffix-prefix matching 39
All-pairs suffix-prefix matching with minimum length 40

Maximum all-pairs suffix-prefix matching 42
3.8.6 Maximal Repeats 46

4 Enhanced Suffix Arrays 51
4.1 Basic Notions 51
4.2 From Suffix Trees to Suffix Arrays 53
4.3 Constructing the lcp-table in Linear Time 53
4.4 A Linear Time Suffix Tree Construction 55
4.5 Algorithms based on Lcp-intervals 57
4.5.1 A simple linear time algorithm for finding supermaximal repeats . . . 57
4.5.2 Computation of maximal unique matches 58
4.6 Exact Pattern Matching on Suffix Arrays 59
4.6.1 A simple method using two binary searches for each pattern character 59
4.6.2 The optimized method of Manber and Myers 61
4.7 Implementation details for suffix arrays 63

5 Fragment Assembly 65
5.1 Sequencing technologies 65
5.2 Shotgun sequencing 65
5.3 Problems in fragment assembly 68
5.3.1 Base calling errors 68
5.3.2 Chimera fragments 69
5.3.3 Vector contamination (only in Sanger Sequencing) 69
5.3.4 Unknown orientation 69
5.3.5 Repeats in the target sequence 70
5.3.6 Insufficient coverage 71
5.4 Models for fragment assembly 72
5.4.1 The shortest common superstring model 72
5.4.2 The reconstruction model 73
5.4.3 Uniform distribution of start positions 75
5.5 Algorithms and data structures for fragment assembly 78
5.5.1 Computation of Overlaps 80
Efficiency of Overlap Computations 82
5.5.2 Representation of Overlaps 83
5.5.3 The Greedy algorithm 88

6 Modeling Context in Sequences 93
6.1 Events, Probabilities and Probabilistic Model Analysis 93
6.1.1 Maximum Likelihood estimation 94
6.1.2 Conditional, joint, and marginal probabilities 95
6.1.3 Prior- and posterior probability 96
6.1.4 Comparing models using Bayes' theorem 96
6.2 `CpG`-Islands 98

6.3 Markov chains . . . 98
6.3.1 Modeling the begin and end states . . . 100
6.3.2 Determining the transition matrix . . . 100
6.3.3 Answering question 1 . . . 101
6.4 Hidden Markov Models . . . 103
6.4.1 The Viterbi Algorithm . . . 105
6.4.2 The Forward Algorithm . . . 108
6.4.3 The Backward-Algorithm . . . 109
6.4.4 Comparison of the three tables . . . 109
6.4.5 Posterior probabilities . . . 110
6.4.6 Training the parameters . . . 110
6.4.7 The Baum-Welch Algorithm . . . 112
6.5 Determining Protein Domains using HMMs . . . 114
6.5.1 Profile hidden Markov models . . . 115
pHMM construction from a multiple alignment . . . 116
Sequence alignment and database searching with pHMMs . . . 119
6.5.2 Profile HMM collections for sequence annotation and classification . 120
Pfam database . . . 120
TIGRFAM database . . . 121
SMART database . . . 121
SUPERFAMILY database . . . 121
CATH Protein structure classification database . . . 121
PANTHER classification system . . . 122

7 Gene Structure Prediction 123
7.1 Simple approaches to gene prediction . . . 124
7.2 Eukaryotic gene structure . . . 124
7.3 Genscan's model . . . 125
7.4 Spliced Alignments Using ESTs . . . 129
7.4.1 The Set of States . . . 131
7.4.2 The Transition Probabilities . . . 131
7.4.3 The Output Weights . . . 132
7.4.4 Optimal Spliced Alignment . . . 133
7.4.5 The Dynamic Programming Recurrences . . . 134
7.5 Consensus Spliced Alignments . . . 135

8 RNA secondary structure prediction 141
8.1 Minimizing the Free Energy . . . 142
8.2 Independent Base Pairs . . . 143
8.3 Structures with Loops . . . 146
8.4 Improving the efficiency of interior loop computation . . . 152
8.5 Representation of Secondary Structures . . . 154

9 De novo detection of LTR retrotransposons 157
9.1 Features of LTR retrotransposons used for de novo prediction 157
9.2 Algorithms and workflow underlying *LTRharvest* 159

Bibliography 163

CHAPTER 1

Fast computation of the Unit Edit Distance

1.1 Properties of the unit edit distance

From previous lectures we know that the edit distance of two strings u and v of length m and n can be computed in $O(mn)$ time using the following recurrence:

$$E_\delta(i,j) = \begin{cases} 0 & \text{if } i=0 \text{ and } j=0 \\ E_\delta(i-1,0)+\delta(u[i]\to\varepsilon) & \text{if } i>0 \text{ and } j=0 \\ E_\delta(0,j-1)+\delta(\varepsilon\to v[j]) & \text{if } i=0 \text{ and } j>0 \\ \min\left\{ \begin{array}{l} E_\delta(i-1,j)+\delta(u[i]\to\varepsilon) \\ E_\delta(i,j-1)+\delta(\varepsilon\to v[j]) \\ E_\delta(i-1,j-1)+\delta(u[i]\to v[j]) \end{array} \right\} & \text{if } i>0 \text{ and } j>0 \end{cases}$$

In this section, we restrict to the case that δ is the unit cost function, *i.e.* all insertions, deletions and mismatches have cost 1. A match has cost 0. The simplified recurrence for table E_δ is shown in Figure 1.1. An evaluated matrix is given in Figure 1.2.

The restriction to the unit cost function leads to some properties, which we exploit later: Consecutive entries in E_δ-columns, E_δ-rows, and E_δ-diagonals differ by at most one. Additionally, the entries in E_δ-diagonals are non-decreasing. See Figure 1.3 for a graphical explanation. These facts are formally stated in the following lemma, for which we do not give a proof (this can be found in [Ukk85].

Lemma 1 Table E_δ has the following properties:

1. $E_\delta(i-1,j)-1 \leq E_\delta(i,j) \leq E_\delta(i-1,j)+1$ for all i, j, $1 \leq i \leq m$, $0 \leq j \leq n$.

2. $E_\delta(i,j-1)-1 \leq E_\delta(i,j) \leq E_\delta(i,j-1)+1$ for all i, j, $0 \leq i \leq m$, $1 \leq j \leq n$.

Figure 1.1: The simplified recurrence for table E_δ for the unit cost function.

$$E_\delta(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0 \\ E_\delta(i-1,0) + 1i & \text{if } i > 0 \text{ and } j = 0 \\ E_\delta(0,j-1) + 1j & \text{if } i = 0 \text{ and } j > 0 \\ \min \left\{ \begin{array}{l} E_\delta(i-1,j) + 1 \\ E_\delta(i,j-1) + 1 \\ E_\delta(i-1,j-1) + \\ \quad (\text{if } u[i] = v[j] \text{ then } 0 \text{ else } 1) \end{array} \right\} & \text{if } i > 0 \text{ and } j > 0 \end{cases}$$

Figure 1.2: The edit distance matrix E_δ for $u = bcacd$ and $v = dbadad$, assuming the unit cost function.

			d	b	a	d	a	d
	$E_\delta(i,j)$	0	1	2	3	4	5	6
	0	0	1	2	3	4	5	6
b	1	1	1	1	2	3	4	5
c	2	2	2	2	2	3	4	5
a	3	3	3	3	2	3	3	4
c	4	4	4	4	3	3	4	4
d	5	5	4	5	4	3	4	4

Figure 1.3: The differences of consecutive entries in matrix E_δ. Along vertical and horizontal edges the differences are either -1, 1, or 1. Along diagonal edges the differences are either 0 or 1.

$$\begin{tikzcd}
(i-1,j-1) \arrow[r, "\{-1{,}0{,}1\}"] \arrow[d, "\{-1{,}0{,}1\}"'] \arrow[dr, "\{0{,}1\}"] & (i-1,j) \arrow[d, "\{-1{,}0{,}1\}"] \\
(i,j-1) \arrow[r, "\{-1{,}0{,}1\}"'] & (i,j)
\end{tikzcd}$$

3. $E_\delta(i-1,j-1) \leq E_\delta(i,j) \leq E_\delta(i-1,j-1)+1$ for all $i, j, 1 \leq i \leq m, 1 \leq j \leq n$.

From the properties stated in Lemma 1 we can conclude the following:

Lemma 2 For all $i, j, 1 \leq i \leq m, 1 \leq j \leq n$, the following property holds: If $E_\delta(i-1,j-1) \leq E_\delta(i,j-1)$ and $E_\delta(i-1,j-1) \leq E_\delta(i-1,j)$, then

$$E_\delta(i,j) = E_\delta(i-1,j-1) \iff u[i] = v[j].$$

Here is a graphical explanation of the Lemma:

$$\begin{tikzcd}
E_\delta(i-1,j-1) \arrow[r, "\leq"] \arrow[d, "\leq"'] \arrow[dr] & E_\delta(i-1,j) \arrow[d] \\
E_\delta(i,j-1) \arrow[r] & E_\delta(i,j) = E_\delta(i-1,j-1) \iff u[i] = v[j]
\end{tikzcd}$$

Proof: When we apply the assumptions to the simplified recurrence, we obtain

$$\begin{aligned}
E_\delta(i,j) &= \min\left\{\begin{array}{l} E_\delta(i,j-1)+1, \\ E_\delta(i-1,j)+1, \\ E_\delta(i-1,j-1) + (\text{if } u[i] = v[j] \text{ then } 0 \text{ else } 1) \end{array}\right\} \\
&= E_\delta(i-1,j-1) + (\text{if } u[i] = v[j] \text{ then } 0 \text{ else } 1)
\end{aligned}$$

$$\begin{aligned}
\text{Hence, } E_\delta(i,j) = E_\delta(i-1,j-1) &\iff (\text{if } u[i] = v[j] \text{ then } 0 \text{ else } 1) = 0 \\
&\iff u[i] = v[j].
\end{aligned}$$

Lemma 3 For all $i, j, 1 \leq i \leq m, 1 \leq j \leq n$, the following property holds:

$$E_\delta(i,j) = \begin{cases} E_\delta(i-1,j-1) & \text{if } u[i] = v[j] \\ 1 + E_\delta(i-1,j) & \text{else if } E_\delta(i-1,j) < E_\delta(i-1,j-1) \\ 1 + \min\{E_\delta(i,j-1), \\ \qquad\qquad E_\delta(i-1,j-1)\} & \text{otherwise} \end{cases}$$

The proof of Lemma 3 is left as an exercise.

1.2 A lazy evaluation strategy

Due to the previous lemma, we do not have to evaluate E_δ completely. Whenever the characters $u[i]$ and $v[j]$ are identical, the corresponding edge in the edit distance graph is minimizing. Hence it suffices to evaluate an entry along this edge. If $u[i] \neq v[j]$, then we additionally have to test if $E_\delta(i-1,j) < E_\delta(i-1,j-1)$ holds. If so, then we can evaluate $E_\delta(i,j)$ without computing $E_\delta(i,j-1)$. Thus matrix E_δ can be evaluated in a lazy strategy. A request to evaluate $E_\delta(m,n)$ then triggers the computation of all necessary values in E_δ in a band around the main diagonal. The smaller $E_\delta(m,n)$, the smaller the band.

Figure 1.4 shows an example of an edit-distance matrix evaluated by the lazy strategy.

The details of this approach are left for an exercise.

Figure 1.4: Entries in E_δ for u = `FREIZEIT` and v = `ZEITGEIST`, evaluated by the lazy strategy. 55 of the 90 entries (*i.e.* 61%) are evaluated.

		Z	E	I	T	G	E	I	S	T
	0	1	2	3	4	5	6	7	8	
F	1	1	2	3	4	5	6	7	8	
R	2	2	2	3	4	5	6	7	8	
E	3	3	2	3	4	5	5	6	7	
I			3	2	3	4	5	5	6	
Z				3	3	4	5	6	6	
E							4	5	6	
I								4	5	
T										5

1.3 Diagonalwise computation

Next we consider which band around the main diagonal of matrix E_δ is actually evaluated. This then leads to an output-sensitive algorithm for computing the unit edit distance. That is, an algorithm, whose running time depends on the computed distance value. The smaller this value, the faster it runs. The description is based on the edit graph $G(u, v)$ of u and v. Recall that this edit graph has three kinds of edges: insertion edges, deletion edges, and replacement edges. The insertion and deletion edges have cost 1. A replacement edge is labeled with an edit operation $\delta(\alpha \to \beta)$. If $\alpha = \beta$, then the replacement edge has cost 0. If $\alpha \neq \beta$, then the replacement edge has cost 1.

Definition 1 Let $d \in \mathbb{N}_0$. A *d-path* is a path in $G(u, v)$ which contains exactly d edges with cost 1. That is, a d-path has cost d. A d-path that begins at $(0, 0)$ is an initial d-path. A d-path that does not begin at $(0, 0)$ is an internal d-path. To simplify the description, a d-path is initial if not stated otherwise.

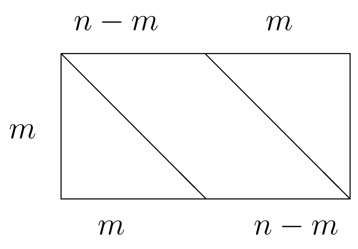
Definition 2 Let $h \in [-m, n]$. The forward diagonal h ($\searrow$) consists of all pairs (i, j), $0 \leq i \leq m$, $0 \leq j \leq n$, satisfying $j - i = h$.

By definition, $(0, 0)$ is on diagonal 0, and (m, n) is on diagonal $n - m$. Hence it is clear that any path from $(0, 0)$ to (m, n) must cross the diagonal band between diagonal 0 and diagonal $n - m$, as shown in Figure 1.5.

Lemma 4 A d-path must end on a diagonal $h \in D_d := \{-d, -d + 1, \ldots, d - 1, d\}$.

Proof: We prove the claim by induction on d.

Figure 1.5: The diagonal band from diagonal 0 (left) to diagonal $n-m$ (right)



- Case $d=0$: A 0-path only has edges of cost 0. As the start node $(0,0)$ is on diagonal 0, the path ends on diagonal $0 \in D_d = D_0 = \{0\}$.
- Suppose the claim hold for $d>0$. Consider a $(d+1)$-path. It contains at least one edge with cost 1, and thus can be split into 3 parts:

  part 1: maximal prefix which is a d-path. By assumption this path ends on diagonal $h \in D_d$.

  part 2: either a horizontal edge from diagonal h to $h+1$, or a vertical edge from diagonal h to $h-1$, or a diagonal edge with cost 1 on diagonal h.

  part 3: an internal 0-path on diagonal $h+1$, or $h-1$, or h, depending on part 2.

Hence the $(d+1)$-path ends on diagonal

$$\begin{aligned}
h' \in\ & \{d'+1 \mid d' \in D_d\} \cup \\
& \{d'-1 \mid d' \in D_d\} \cup \\
& \{D_d\} \\
=\ & \{-d+1, -d+1+1, \ldots, d-1+1, d+1\} \cup \\
& \{-d-1, -d+1-1, \ldots, d-1-1, d-1\} \cup \\
& \{-d, -d+1, \ldots, d-1, d\} \\
=\ & \{-(d+1)+2, -(d+1)+3, \ldots, (d+1)-1, d+1\} \cup \\
& \{-(d+1), -(d+1)+1, \ldots, (d+1)-3, (d+1)-2\} \cup \\
& \{-(d+1)+1, -(d+1)+2, \ldots, (d+1)-2, (d+1)-1\} \\
=\ & \{-(d+1), -(d+1)+1, -(d+1)+2, \ldots, \\
& \ (d+1)-2, (d+1)-1, d+1\} \\
=\ & D_{d+1}
\end{aligned}$$

1.4 An output sensitive algorithm

Definition 3 A d-path of maximal length ending on diagonal h is a maximal d-path on h. □

The idea of the algorithm is to compute how far we come in the edit graph using d edges of cost 1. More precisely,

$$\text{for all } d = 0, 1, \ldots \text{ and all } h, -d \leq h \leq d$$

compute the endpoint of a maximal d-path on h. Now recall that (m, n) is on diagonal $n - m$. Hence, if d is minimal such that (m, n) is the endpoint of a maximal d-path on $n - m$, then we have $edist_\delta(u, v) = E_\delta(m, n) = d$.

The endpoint of a maximal d-path is defined in terms of the *front* of a diagonal:

Definition 4 For any $d \in \mathbb{N}_0$ and any $-d \leq h \leq d$ define

$$\mathit{front}(h, d) = \max\{i \mid 0 \leq i \leq m, E_\delta(i, h+i) = d\}.$$

That is, the endpoint of a maximal d-path on a particular diagonal h is given as the row number of the endpoint.

Lemma 5 Let $d_{\min} := \min\{d \in \mathbb{N}_0 \mid \mathit{front}(n-m, d) = m\}$. Then $d_{\min}$ is the unit edit distance of u and v.

Proof: Let $d = edist_\delta(u, v)$. We have $d = E_\delta(m, n) = E_\delta(m, (n-m)+m)$. Hence

$$\mathit{front}(n-m, d) = \max\{i \mid 0 \leq i \leq m, E_\delta(i, n-m+i) = d\} = m.$$

Thus $d \geq d_{\min}$. Now suppose $d > d_{\min}$. We have $\mathit{front}(n-m, d_{\min}) = m$ which implies

$$d_{\min} = E_\delta(m, (n-m)+m) = E_\delta(m, n) = d.$$

This is a contradiction, which implies that the assumption $d > d_{\min}$ was wrong. Hence $d = d_{\min}$.

We will now develop recurrences for computing *front*. Consider the case $d = 0$. A maximal 0-path ends on (i, i) where $i = |lcp(u, v)|$ and $lcp(u, v)$ is the longest common prefix of u and v. Hence we derive

$$\mathit{front}(0, 0) = |lcp(u, v)| \tag{1.1}$$

Example 1 For $u =$ `aabaa` and $v =$ `aaaba` we have $lcp(u, v) =$ `aa` and so $\mathit{front}(0, 0) = |lcp(u, v)| = 2$. This can be verified by the following matrix E_δ which only shows the 0-values.

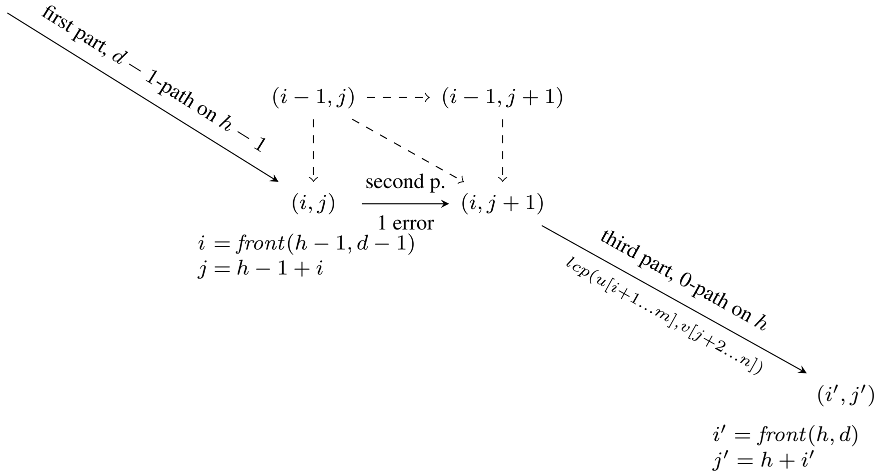
		a	a	a	b	a
	0					
a		0				
a			0			
b						
a						
a						

Now let $d > 0$ and consider a maximal d-path on h. There are three ways to split this path into three parts.

Case 1: Suppose the d-path on h consists of the following three parts (as shown in Figure 1.6):

- a maximal $(d-1)$-path on $h - 1$.

Figure 1.6: Case 1: Splitting of d-path into 3 parts (second part is horiz. edge)



- a horizontal edge from diagonal $h-1$ to diagonal h.
- an internal maximal 0-path on diagonal h.

Suppose that the maximal $(d-1)$-path on $h-1$ ends in (i,j), *i.e.* $i = \mathit{front}(h-1,d-1)$ and $j = h-1+i$. Then the maximal path on diagonal h ends in some node (i',j') where $i' = \mathit{front}(h,d)$ and $j' = h+i'$. The length of the maximal 0-path on diagonal h is the length of $\mathit{lcp}(u[i+1\ldots m], v[j+2\ldots n]) = \mathit{lcp}(u[i+1\ldots m], v[h+i+1\ldots n])$. Hence we conclude

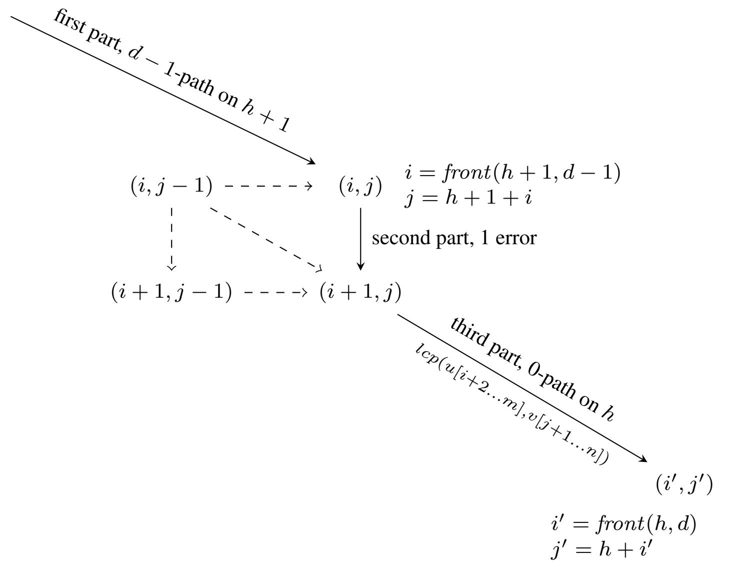
$$\mathit{front}(h,d) = i + |\mathit{lcp}(u[i+1\ldots m], v[h+i+1\ldots n])| \tag{1.2}$$

Case 2: Suppose the d-path on h consists of the following three parts (as shown in Figure 1.7):

- a maximal $(d-1)$-path on $h+1$.
- a vertical edge from diagonal $h+1$ to diagonal h.
- an internal maximal 0-path on diagonal h.

Suppose that the maximal $(d-1)$-path on $h+1$ ends in (i,j), *i.e.* $i = \mathit{front}(h+1,d-1)$ and $j = h+1+i$. Then the internal maximal 0-path on diagonal h ends in some node (i',j') where $i' = \mathit{front}(h,d)$ and $j' = h+i'$. The length of the maximal 0-path on diagonal h is the

Figure 1.7: Case 2: Splitting of d-path into 3 parts (second part is vertical edge)



length of $lcp(u[i+2\ldots m], v[j+1\ldots n]) = lcp(u[i+2\ldots m], v[h+i+2\ldots n])$. Hence we conclude

$$front(h,d) = i+1+|lcp(u[i+2\ldots m], v[h+i+2\ldots n])| \tag{1.3}$$

Case 3: Suppose the d-path on h consists of the following three parts (as shown in Figure 1.8):

- a maximal $(d-1)$-path on h.
- a diagonal edge with weight 1 on diagonal h.
- an internal maximal 0-path on diagonal h.

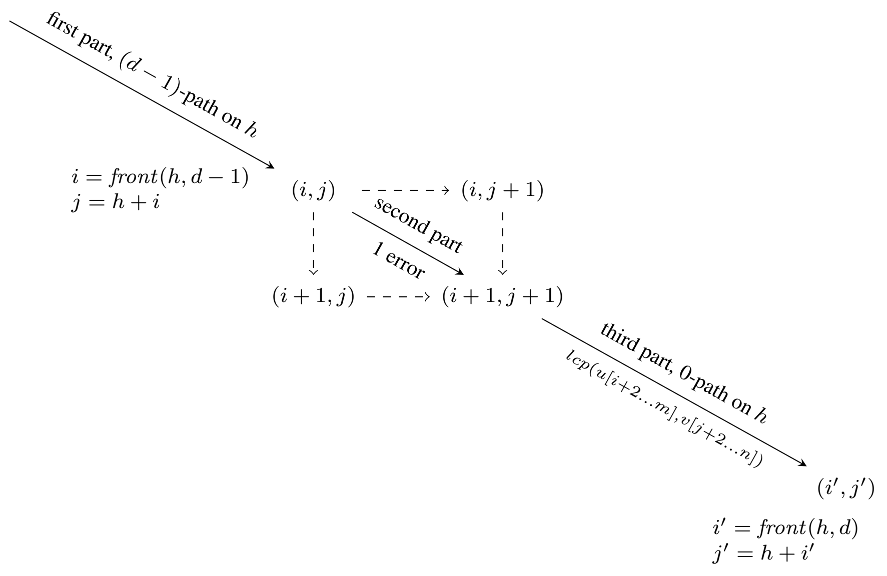
Suppose that the maximal $(d-1)$-path on h ends in (i, j), *i.e.* $i = front(h, d-1)$ and $j = h+i$. Then the internal maximal 0-path on diagonal h ends in some node (i', j') where $i' = front(h, d)$ and $j' = h + i'$. The length of the internal maximal 0-path on diagonal h is the length of $lcp(u[i+2\ldots m], v[j+2\ldots n] = lcp(u[i+2\ldots m], v[h+i+2\ldots n])$. Hence we conclude

$$front(h,d) = i+1+|lcp(u[i+2\ldots m], v[h+i+2\ldots n])| \tag{1.4}$$

Now, in Equation (1.2)

$$front(h,d) = i+|lcp(u[i+1\ldots m], v[h+i+1\ldots n])|$$

Figure 1.8: Case 3: Splitting of d-path into 3 parts (second part is diagonal edge)



substitute the term i by ℓ, where $\ell = front(h-1, d-1)$, leading to

$$front(h, d) = \ell + |lcp(u[\ell+1 \ldots m], v[h+\ell+1 \ldots n])| \tag{1.5}$$

Furthermore, in Equation (1.3) and (1.4)

$$front(h, d) = i + 1 + |lcp(u[i+2 \ldots m], v[h+i+2 \ldots n])|$$

substitute the term $i+1$ by ℓ, where $\ell = front(h+1, d-1) + 1$ in Equation (1.3), and $\ell = front(h, d-1) + 1$ in Equation (1.4), to obtain

$$front(h, d) = \ell + |lcp(u[\ell+1 \ldots m], v[h+\ell+1 \ldots n])| \tag{1.6}$$

Note that the right hand side of each of the two Equations (1.5), (1.6) are identical. The Equations only differ in how ℓ is defined. Since all three cases can occur at the same time, we compute the maximal value for ℓ and obtain the following recurrence combining all three equations:

$$\begin{aligned} front(h, d) &= \ell + |lcp(u[\ell+1 \ldots m], v[h+\ell+1 \ldots n])| \\ &\text{where } \ell = \max \left\{ \begin{array}{l} front(h-1, d-1) \\ front(h+1, d-1) + 1 \\ front(h, d-1) + 1 \end{array} \right\} \end{aligned} \tag{1.7}$$

The complete pseudo code for computing the unit edit distance is shown in Algorithm 1.

Algorithm 1 Greedy DP algorithm for computing the unit edit distance

Input: sequences u and v, δ is the unit cost function
Output: $edist_\delta(u,v)$

```
1:  function $fastedist(u,v)$
2:      $(m,n) := (|u|,|v|)$
3:      for $d := 0$ upto $\max\{n,m\}$ do
4:          for $h := -d$ upto $d$ do
5:              Compute $front(h,d)$ according to Equation (1.7)
6:          end for
7:          if $-d \leq n-m \leq d$ and $front(n-m,d) = m$ then
8:              return $d$
9:          end if
10:     end for
11: end function
```

Let $e := edist_\delta(u,v)$. For each $d \in [0,e]$, the algorithm computes a front of width $2 \cdot d + 1$. Hence the total number of front values is

$$\begin{aligned}\sum_{d=0}^{e}(2d+1) &= e+1+2\sum_{d=0}^{e} d\\ &= e+1+2\frac{(e+1)\cdot e}{2}\\ &= e+1+(e+1)\cdot e\\ &\in O(e^2)\end{aligned}$$

The computation of ℓ means to maximize over three values, which requires constant time. Moreover, the longest common prefix of $u[\ell+1\ldots m]$ and $v[h+\ell+1\ldots n]$ is computed. As shown in the previous case distinction, each such computation extends the path towards (m,n) by at least one step, thus either increasing the row or the column index (or both). Thus the entire computation of the longest common prefixes involves at most $O(m+n)$ pairwise character matches and $O(e^2)$ mismatches. Thus the longest common prefixes are computed in $O(m+n+e^2)$ time. This is also the running time of the complete algorithm. Thus the algorithm is output sensitive. The smaller the distance, the faster it runs. Each generation of $front$-values

$$front(-d,d), front(-d+1,d), \ldots, front(d-1,d), front(d,d)$$

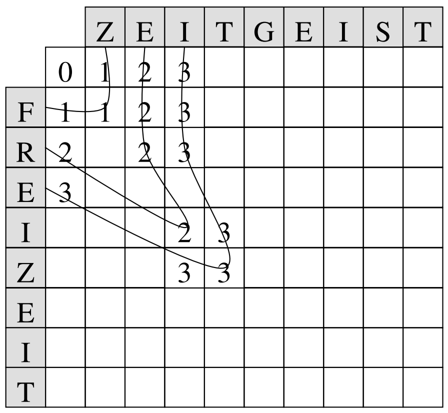
with $d > 0$ can be computed from the previous generation. Thus we only need to store two generations at any time, which requires $O(e)$ space. As we need to store the two sequences, we need $O(m+n)$ space. Note that we can easily modify the algorithm to also deliver an optimal alignment. As in the standard algorithm for computing optimal alignments, we mark in each entry which of the three front-values it depends on gave rise to the maximum value. Of course, we then cannot throw away a generation of front-values, which leads to a space requirement of $O(e^2)$.

See Figure 1.9 for an example of the values implicitly computed by this algorithm.

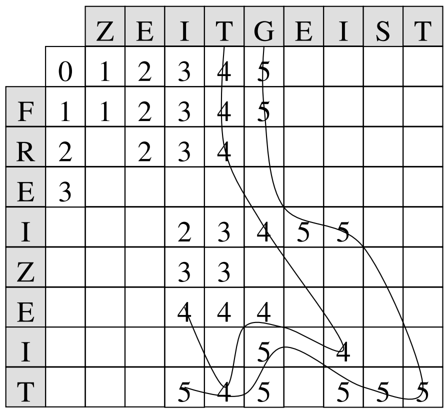
Figure 1.9: Left: Complete matrix E_δ and the values implicitly computed for $d = 3$ and $d = 5$. Right: front values computed by Algorithm 1. The front-value leading to the sought distance is shown in bold face.

		Z	E	I	T	G	E	I	S	T
	0	1	2	3	4	5	6	7	8	9
F	1	1	2	3	4	5	6	7	8	9
R	2	2	2	3	4	5	6	7	8	9
E	3	3	2	3	4	5	5	6	7	8
I	4	4	3	2	3	4	5	5	6	7
Z	5	4	4	3	3	4	5	6	6	7
E	6	5	4	4	4	4	4	5	6	7
I	7	6	5	4	5	5	5	4	5	6
T	8	7	6	5	4	5	6	5	5	5

complete distance matrix
$(m + 1) \cdot (m + 1) = 9 \cdot 10 = 90$ values



values implicitly computed for $d = 3$
e.g. $\mathit{front}(0,3) = 4, \mathit{front}(-1,3) = 5, \mathit{front}(-2,3) = 5$



values implicitly computed for $d = 5$. Since $m = 8$ and $n = 9$, we have $\mathit{front}(n - m, d) = \mathit{front}(1, 5) = 8$ and therefore $d = 5$.

h	d	$\mathit{front}(h,d)$
0	0	0
−1	1	1
0	1	1
1	1	0
−2	2	2
−1	2	4
0	2	2
1	2	1
2	2	0
−3	3	3
−2	3	5
−1	3	5
0	3	4
1	3	2
2	3	1
3	3	0
−4	4	8
−3	4	6
−2	4	6
−1	4	6
0	4	7
1	4	4
2	4	2
3	4	1
4	4	0
−5	5	8
−4	5	0
−3	5	8
−2	5	7
−1	5	8
0	5	8
1	**5**	**8**
2	5	4
3	5	4
4	5	1
5	5	0

36 values