Plán semestra

Dnes: *editačná vzdialenosť, najdlhšia spoločná podpostupnosť*

Do pondelka: výber článku na prezentáciu

Streda 17.4.: *zlepšenia výpočtu editačnej vzdialenosti*

Štvrtok 18.4.: prednáška nebude

Streda 24.4.: *približné výskyty vzorky, lokálne podobnosti, bioinformatika*

Štvrtok 25.4.: *zostavovanie DNA sekvencií, najkratšie spoločné nadslovo*

Streda 1.5.: sviatok

Štvrtok 2.5.: *viacnásobné zarovnanie, opakujúce sa sekvenčné motívy*

Streda 8.5.: sviatok

Štvrtok 9.5.: prezentácie

Streda 15.5.: prezentácie

Štvrtok 16.5.: prezentácie

New topic: Approximate occurrences, similar strings

Differences in texts occurs due to typos, experimental errors, transmission errors, deliberately introduced,. . .

So far: Pattern matching with Hamming distance $\leq k$ in $O(nk)$

Today: edit distance between two strings

Edit distance, Levenshtein distance (editačná vzdialenosť)

Edit operations: $(u, v \in \Sigma^*, a, b \in \Sigma)$

– insertion (inzercia) $uv \to uav$

– deletion (delécia) $uav \to uv$

– substitution (substitúcia) $uav \to ubv$

Edit distance $d_E(S, T) =$

shortest sequence of edit operations that changes S to T

Example:

$S =$ema ma mamu, $T =$mama sa ma, $d_E(S, T) = 5$

`ema␣ma␣mamu` (delete e) `ma␣ma␣mamu` (delete space)

`mama␣mamu` (substitute m->s) `mama␣samu` (insert space)

`mama␣sa␣mu` (substitute u-a) `mama␣sa␣ma`

Edit distance as an alignment

Sequence alignment (zarovnanie):

insert gaps ($-$) to S and T to get matrix with 2 rows

— column with a gap (insertion or deletion): cost 1

— column with two different symbols (substitution): cost 1

— column with equal symbols: cost 0

Example:

```
ema␣ma␣ma-mu
-ma-ma␣sa␣ma
100100010101
```

Problem: compute $d_E(S, T)$ for two input strings S and T

(note $d_H(S, T)$ trivially in $O(n)$ time)

Dynamic programming for $d_E(S, T)$

Let $m = |S|$, $n = |T|$, indexing from 1: $S[1..m]$, $T[1..n]$

Let $A[i, j] = d_E(S[1..i], T[1..j])$

Compute $A[i, j]$ for $0 \le i \le m$, $0 \le j \le n$

Example:

```
12345678901
ema␣ma␣mamu
mama␣sa␣ma
```

$A[3, 4] = 2$

Dynamic programming for $d_E(S, T)$

Let $m = |S|$, $n = |T|$, indexing from 1: $S[1..m]$, $T[1..n]$

Let $A[i, j] = d_E(S[1..i], T[1..j])$

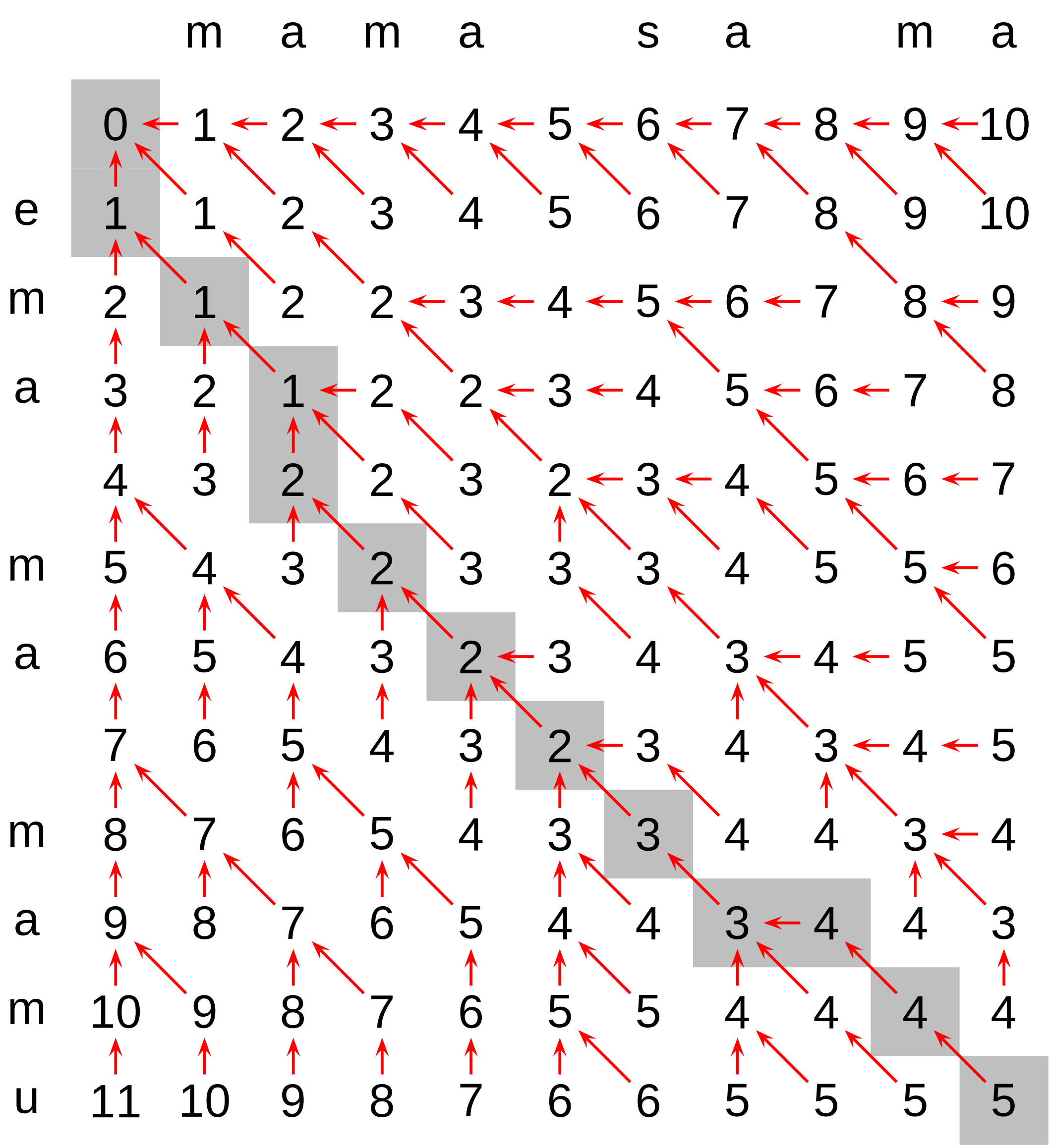
Compute $A[i, j]$ for $0 \leq i \leq m$, $0 \leq j \leq n$

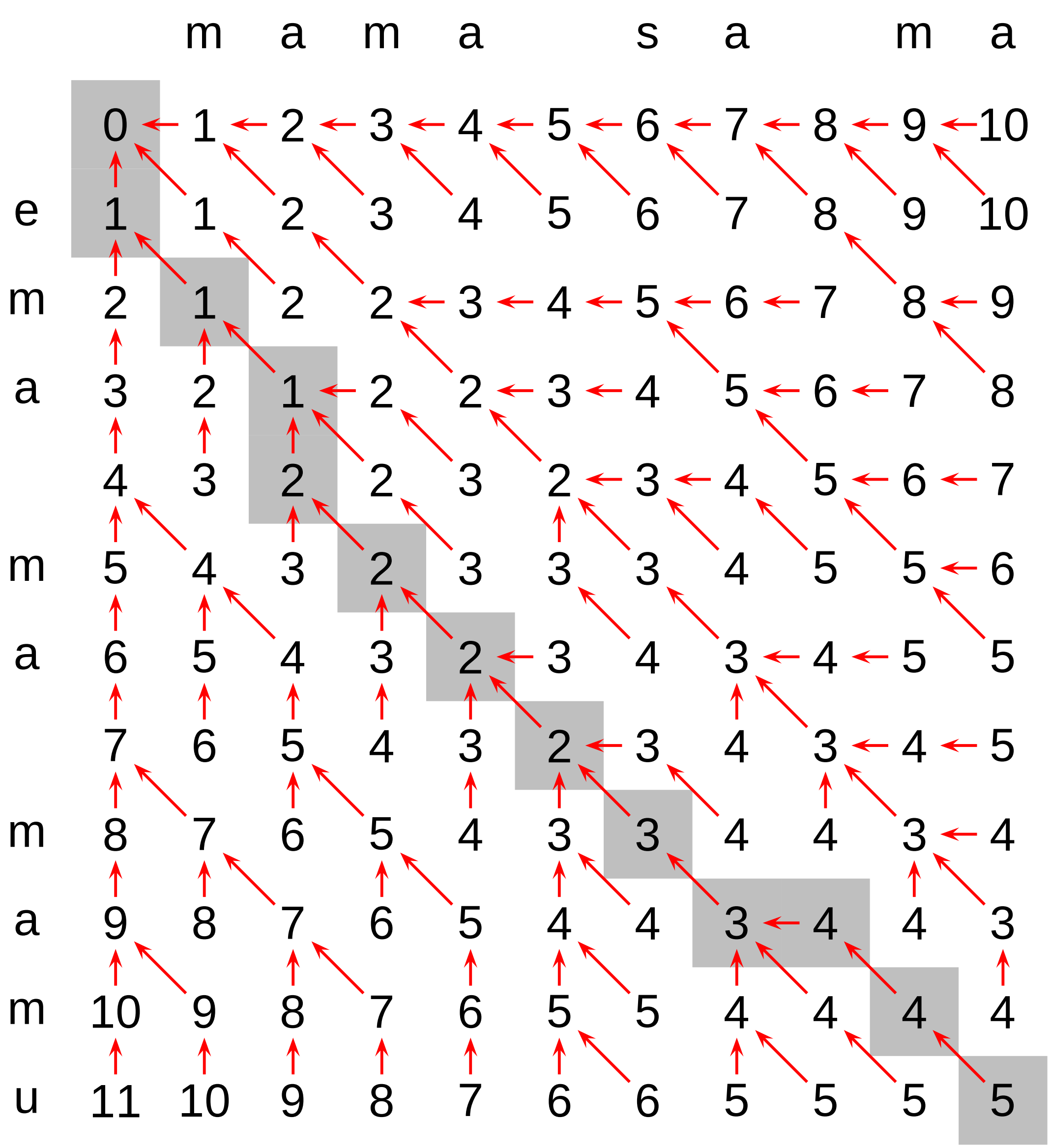
$$A[i, j] = \min \begin{cases} A[i-1, j-1] + c(S[i], T[j]) \\ A[i-1, j] + 1 \\ A[i, j-1] + 1 \end{cases}$$

$$c(a, b) = \begin{cases} 0 & \text{if } a = b \\ 1 & \text{if } a \neq b \end{cases}$$

		m	a	m	a		s	a		m	a
	0	1	2	3	4	5	6	7	8	9	10
e	1	1	2	3	4	5	6	7	8	9	10
m	2	1	2	2	3	4	5	6	7	8	9
a	3	2	1	2	2	3	4	5	6	7	8
	4	3	2	2	3	2	3	4	5	6	7
m	5	4	3	2	3	3	3	4	5		
a											
m											
a											
m											
u											

$$A[i,j] = \min \begin{cases} A[i-1,j-1] \\ \quad +c(S[i],T[j]) \\ A[i-1,j]+1 \\ A[i,j-1]+1 \end{cases}$$


m a m a s a m a
e m a m a m a m u



Alignment:

```
ema␣ma␣ma-mu
-ma-ma␣sa␣ma
```

Edit distance is a distance function (metrika)

$d_E(S, T) \geq 0$

$d_E(S, T) = 0 \iff S = T$

$d_E(S, T) = d_E(T, S)$ (symmetry)

$d_E(S, T) \leq d_E(S, X) + d_E(X, T)$ (triangle inequality)

Generalized edit distance

(zovšeobecnená editačná vzdialenosť)

Table of weights $w(a, b)$ for $a, b \in \Sigma \cup \{-\}$

For $a, b \in \Sigma$:

$w(a, -)$ cost of deletion, $w(-, b)$ cost of insertion

$w(a, b)$ cost of substitution from a to b, or cost of identity if $a = b$

Dynamic programming to find the alignment with lowest cost

– for some strange weights not the lowest sequence of operations

– not always a distance function

$$A[i,j] = \min \begin{cases} A[i-1,j-1] + w(S[i],T[j]) \\ A[i-1,j] + w(S[i],-) \\ A[i,j-1] + w(-,T[j]) \end{cases}$$

Longest common subsequence lcs

Najdlhšia spoločná podpostupnosť

Def: subsequence of a sequence $a_1, \ldots, a_n$ is sequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ such that $1 \leq i_1 < i_2 < \cdots < i_k \leq n$

$lcs(S, T) =$ length of the longest sequence which is a common subsequence of both S and T

Example:

$S =$ema ma mamu, $T =$mama sa ma, $lcs(S, T) = 7$

Also alignment (disallow substitutions):

```
ema␣ma␣---mamu
-ma-ma␣sa␣ma--
.**.***...**..
```

How to find using DP for generalized edit distance?

Program diff

Compare two files line by line, e.g. two versions of a source code.

Find (approximation of) lcs(S,T) where symbols are lines of the files.

```
1522a1540
>     my $last_line = undef;
1525,1526c1543,1544
<     foreach my $gtf_line (@$transcript) {
<       # printf STDERR Dumper($gtf_line);
---
>     for(my $line_num = 0; $line_num<@$transcript; $line_num++)
>       my $gtf_line = $transcript->[$line_num];
```

Hunt, Szymanski 1977

$Z[i] =$ list of j s.t. $T[j] = S[i]$ in decreasing order

$Z = Z[1]Z[2]\ldots Z[m]$

Example:

```
  12345678901
S:ema␣ma␣mamu
T:mama␣sa␣ma
```

$Z[1]_e = \emptyset$

$Z[2]_m = 9, 3, 1$

$Z[3]_a = 10, 7, 4, 2$

$Z[4] = 8, 5$

$Z[5]_m = 9, 3, 1$

$Z[6]_a = 10, 7, 4, 2$

$Z[7] = 8, 5$

$Z[8]_m = 9, 3, 1$

$Z[9]_a = 10, 7, 4, 2$

$Z[10]_m = 9, 3, 1$

$Z[11]_u = \emptyset$

$Z = 9, 3, 1, 10, 7, 4, 2, 8, 5, 9, 3, 1, 10, 7, 4, 2, 8, 5, 9, 3, 1, 10, 7, 4, 2, 9, 3, 1$

Convert to another problem

Lemma: lcs(S,T)=lis(Z)

lis(Z): length of the longest increasing subsequence of Z

$Z_{i_1} < Z_{i,2} < \cdots < Z_{i,k}$ where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$

lis(Z) contains j from $Z[i] \iff$ lcs(S,T) contains $S[i] = T[j]$

Example:

```
  12345678901
S:ema␣ma␣mamu    ema␣ma␣---mamu
T:mama␣sa␣ma     -ma-ma␣sa␣ma--
```

$;9,3,\underline{1};10,7,4,\underline{2};8,5;9,\underline{3},1;10,7,\underline{4},2;8,\underline{5};\underline{9},3,1;\underline{10},7,4,2;9,3,1;$

Problems

Let $r = |Z|$

- How to compute Z efficiently? (as a function of m, n, r, σ)
  - Small alphabet (e.g. $\Sigma = \{1, \ldots, n+m\}$)
  - Large alphabet
  - Symbols = lines in files, as in diff
- How to find lis(Z) efficently?

A simple dynamic programming algorithm for lis(Z)

$A[i] =$ l.i.s. of $Z_1 \ldots Z_i$ that ends with Z_i

Example:

$Z = 9, 3, 1, 10, 7, 4, 2, 8, 5, 9, 3, 1 \ldots$

$A = 1, 1, 1, 2, 2, 2, 2, 3, 3, 4, 3, 1, \ldots$

Another dynamic programming algorithm for lis(Z)

Alg 2: $A[i,j] =$ min. x such that x can be the last element in an increasing subsequence of $Z_1 \ldots Z_i$ of legth j

Example:

$i = 9$, $Z = 9, 3, 1, 10, 7, 4, 2, 8, 5, \ldots$

$A[9, 0] = -\infty$

$A[9, 1] = 1$ i.s. (1)

$A[9, 2] = 2$ i.s. $(1, 2)$

$A[9, 3] = 5$ i.s. $(1, 2, 5)$

$A[9, i] = \infty$ for $i \geq 4$

```
A[0,0] = −infinity; A[1..r] = infinity;
for(int i=1; i<=r; i++) {
  A[i,0] = −infinity; A[0,i] = infinity;
}
for(int i=1; i<=r; i++) {
  for(int j=1; j<=r; j++) {
    if(A[i−1,j−1] < Z[i])
      A[i,j] = min(Z[i], A[i−1,j]);
    else
      A[i,j] = A[i−1,j]
  }
}
return highest k s.t. A[r,k]<infinity;
```

```
A[0] = -infinity; A[1..r] = infinity;
for(int i=1; i<=r; i++) {
    find smallest k s.t. A[k]>=Z[i] (bin. search);
    A[k] = Z[i];
}
return highest k s.t. A[k]<infinity;
```

```
A[0] = -infinity; A[1..r] = infinity;
B[0] = 0;
for(int i=1; i<=r; i++) {
    find smallest k s.t. A[k]>=Z[i] (bin. search);
    A[k] = Z[i];
    B[k] = new node;
    B[k]->index = i; B[k]->next = B[k-1];
}
find highest k s.t. A[k]<infinity;
v = B[k];
while(v) {
    print v->index; v = v->next;
}
```