Plán semestra

Dnes: *zlepšenia výpočtu editačnej vzdialenosti*

Zajtra (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

Rôzne:

DÚ opravená, výber článkov väčšinou uzavretý

Utorok 23.4.: nezabudnite sa ísť pozrieť na ŠVK

Edit distance, Levenshtein distance (editačná vzdialenosť)

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

– insertion (inzercia) $uv \rightarrow uav$

– deletion (delécia) $uav \rightarrow uv$

– substitution (substitúcia) $uav \rightarrow ubv$

Edit distance $d_E(S, T) =$

shortest sequence of edit operations that changes S to T

Example:

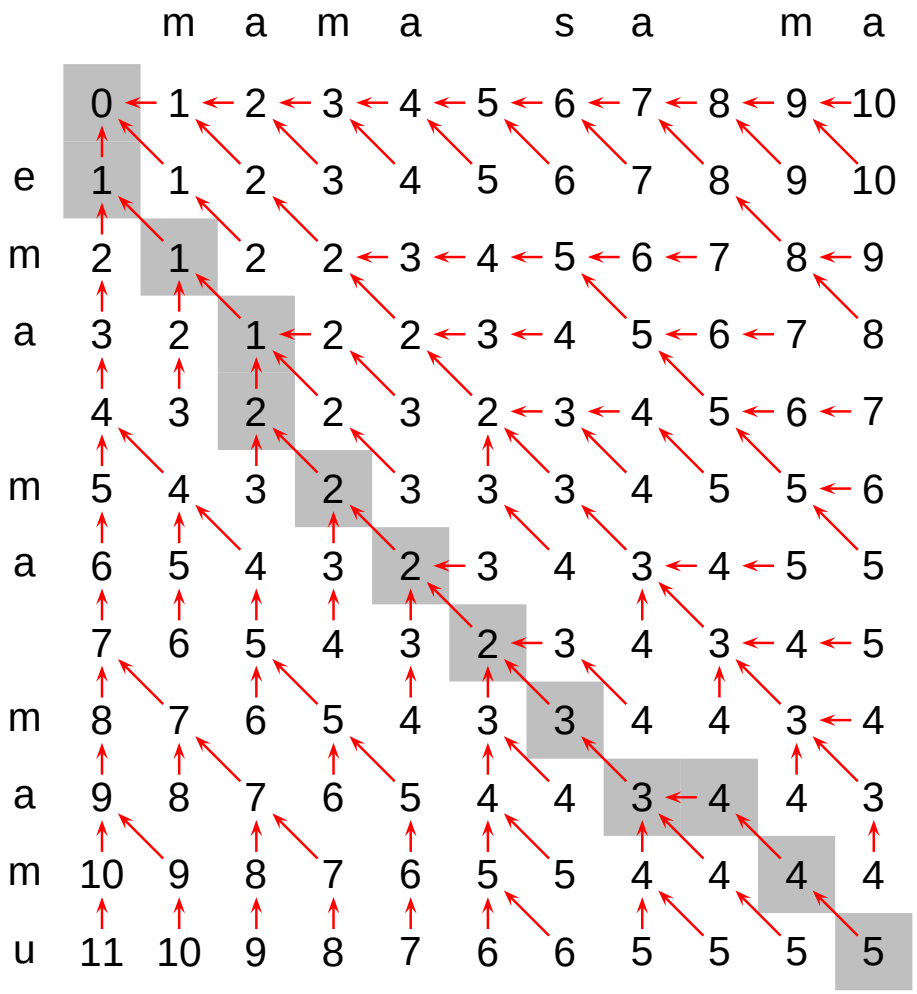
$S =$ema ma mamu, $T =$mama sa ma, $d_E(S, T) = 5$

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

Dynamic programming for computing $d_E(S,T)$



Subproblem:

$$A[i,j] =$$

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

Recurrence:

$$A[i,j] =$$

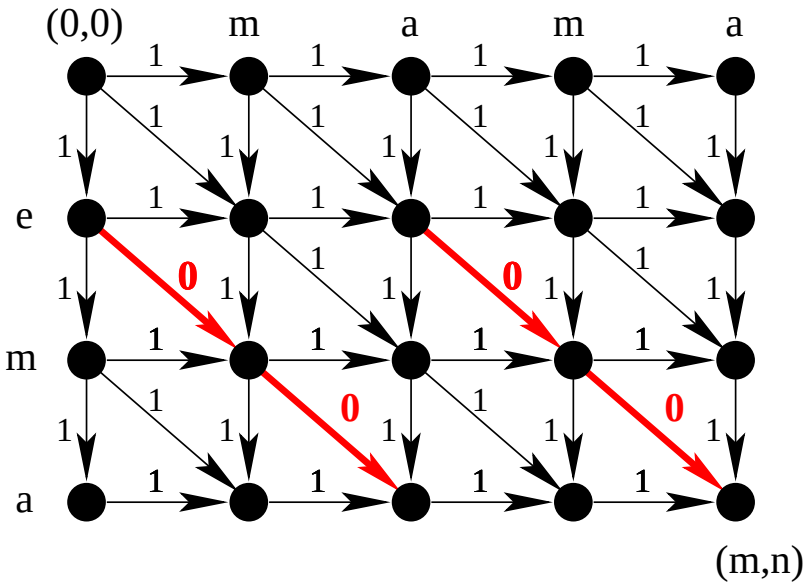
$$\min \begin{cases} A[i-1,j-1] + c(S[i], T[j]) \\ A[i-1,j] + 1 \\ A[i,j-1] + 1 \end{cases}$$

Alignment:

```
ema␣ma␣ma-mu
-ma-ma␣sa␣ma
```

Dynamic programming in graph representation

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



Each alignment – path from $(0,0)$ to (m,n)

Alignment with lowest cost – path with lowest cost (shortest path)

Shortest paths in directed acyclic graphs

- Sort vertices of the graph topologically,
  i.e. $(v_i, v_j) \in E$ implies $i < j$
- Shortest path from s to t by dynamic programming
  $A[u]$: shortest path from s to u
  $A[u] = \min_{v:(v,u)\in E} A[v] + w(v,u)$
- Running time $O(|V| + |E|)$
- Similarly longest path (in general graphs NP-hard)

Running time of dynamic programming

$O(nm)$ on strings of lengths n and m

How long does it takes?

(straightforward implementation, random sequences of length n, ordinary desktop couple of years ago)

n	time
100	0.0008s
1,000	0.08s
10,000	8s
100,000	13 minutes (*)
1,000,000	22 hours (*)
10,000,000	3 months (*)
100,000,000	25 years (*)

Improvements of dynamic programming

Hunt-Szymanski algorithm for LCS: $O(m + n + r \log r)$

where $0 \leq r \leq mn$, $r = |\{(i, j) : S[i] = T[j]\}|$

Hirschberg's algorithm: linear memory

Four Russians technique: slightly improve running time

Ukkonen's algorithm: fast for highly smiliar strings

Hirschberg algorithm 1975

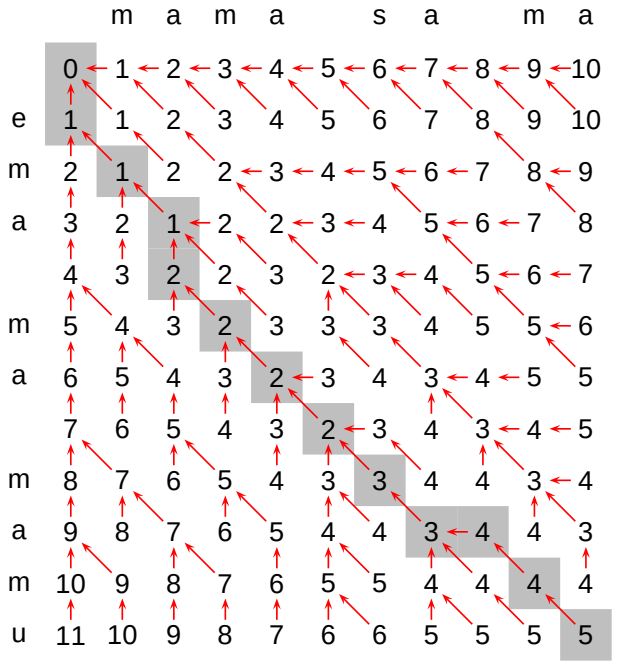
Basic dynamic programming for edit distance:

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

$O(nm)$ time

$O(nm)$ memory

Goal: linear memory $O(n+m)$



```
 1  optA(l1, r1, l2, r2) { // align S[l1..r1] and T[l2..r2]
 2      if(r1—l1 <= 1 ||  r2—l2 <=1)
 3          solve using dynamic programming
 4      else {
 5          k=(r—l+1)/2;
 6          for (i=0; i<=k; i++)
 7              compute A[i,∗] from A[i—1,∗]
 8          for (i=k+1; i<=r—l+1; i++)
 9              compute A[i,∗], B_k[i,∗] from A[i—1,∗], B_k[i—1,∗]
10          k2=B_k[r1—l1—1,r2—l2—1];
11          optA(l1, l1+k—1, l2, l2+k2—1);
12          optA(l1+k, r2, l2+k2, r2);
13      }
14  }
```

Four Russians technique

Arlazarov, Dinic, Kronrod, Faradzev, 1970

Goal: improve running time of DP from $O(mn)$

For simplicity, assume $|S| = |T| = n$, $\sigma = 2$

Divide A into square blocks $t \times t$, overlapping by one column/row

Lemma: Adjacent values in a row or column of A differ by at most 1

Four Russians technique

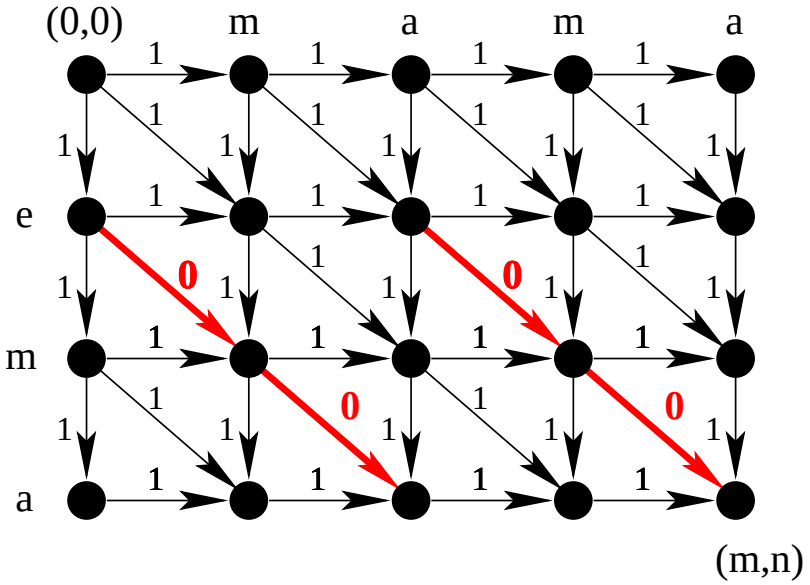
- Pre-computation for one block: $O(t^2)$
- Number of different blocks: 6^{2t-2}
- Overall precomputation $O(t^2 6^{2t})$
- A has n^2/t^2 blocks, each in $O(t)$ time
- Overall $O(n^2/t + t^2 6^{2t})$
- Set $t = \log_6(n)/2$, i.e. $n = 6^{2t}$
- Running time $O(n^2/\log n)$
- What about larger alphabets?

Ukkonen's algorithm 1985

Goal: make dynamic programming faster if $d_E(S, T)$ is small

Number diagonals of DP matrix so that $A[i, j]$ belongs to diagonal $j - i$.

Lemma: If $D = d_E(S, T)$, the shortest path in the graph representation of dynamic programming uses only diagonals from set $\{-D, \ldots, D\}$



Improvements of dynamic programming

Hunt-Szymanski algorithm for LCS: $O(m + n + r \log r)$

where $0 \leq r \leq mn$, $r = |\{(i, j) : S[i] = T[j]\}|$

Four Russians technique: $O(mn/\log m)$ for a constant σ

(precompute small squares, save time)

Hirschberg's algorithm: $O(mn)$ time but $O(m + n)$ memory

(compute where the paths crosses middle of the matrix,

divide and conquer)

Ukkonen's algorithm: $O(md)$ time where $d = d_E(S, T)$

Guess d, verify in a band of diagonals