

Dynamic Programming Algorithms

Minimum Edit Distance

CS 336: Design and Analysis of Algorithms
© Konstantin Makarychev

How to measure distances between strings?

- **Hamming Distance:** Distance between a and b is the number of positions where a and b differ i.e.,

$$d_H(a, b) = |\{i: a_i \neq b_i\}|$$

- **Pros:** Easy to compute. Works well in many applications.
- **Cons:** Doesn't work in some other applications:
 - Text editing: Typos “Hamming Distance” vs “Hmming Distance”.
 - Bioinformatics: DNA mutations.

How to measure distances between strings?

- **Edit or Levenshtein distance:** The distance between a and b equals the minimum number of deletions, insertions, and substitutions required to transform a to b .

Applications:

- Text processing: Distance from “Hamming Distance” to “Hmming Distance” equals 1.
- Bioinformatics: Edit distance is a good similarity measure for DNA strands.

```

26 namespace kaldl {
27
28 template<class T>
29 int32 LevenshteinEditDistance(const std::vector<T> &a,
30                               const std::vector<T> &b) {
31     // Algorithm:
32     // - write A and B for the sequences, with elements a_0 ...
33     // - let |A| = M and |B| = N be the lengths, and have
34     // - elements a_0 ... a_{M-1} and b_0 ... b_{N-1}.
35     // - We are computing the recursion
36     // - E(m, n) = min( E(m-1, n-1) + (1-delta(a_{m-1}, b_{n-1})),
37     // - E(m-1, n) + 1,
38     // - E(m, n-1) + 1).
39     // - Where E(m, n) is defined for m = 0..M and n = 0..N and out-of-
40     // - bounds quantities are considered to be infinity (i.e. the
41     // - recursion does not visit them).
42
43     // We do this computation using a vector e of size N+1.
44     // The outer iterations range over m = 0..M.
45
46     int M = a.size(), N = b.size();
47     std::vector<int32> e(N+1);
48     std::vector<int32> e_tmp(N+1);
49     // initialize e.
50     for (size_t i = 0; i < e.size(); i++)
51         e[i] = i;
52     for (int32 m = 1; m <= M; m++) {
53         // computing E(m, .) from E(m-1, .)
54         // handle special case n = 0:
55         e_tmp[0] = e[0] + 1;
56
57
58     }
59 }

```

```

47 int M = a.size(), N = b.size();
48 std::vector<int32> e(N+1);
49 std::vector<int32> e_tmp(N+1);
50 // initialize e.
51 for (size_t i = 0; i < e.size(); i++)
52     e[i] = i;
53 for (int32 m = 1; m <= M; m++) {
54     // computing E(m, .) from E(m-1, .)
55     // handle special case n = 0:
56     e_tmp[0] = e[0] + 1;
57
58     for (int32 n = 1; n <= N; n++) {
59         int32 term1 = e[n-1] + (a[m-1] == b[n-1] ? 0 : 1);
60         int32 term2 = e[n] + 1;
61         int32 term3 = e_tmp[n-1] + 1;
62         e_tmp[n] = std::min(term1, std::min(term2, term3));
63     }
64     e = e_tmp;
65 }
66 return e.back();
67 }

```

Edit Distance

Given: Two strings a_1, \dots, a_n and b_1, \dots, b_m .

Goal: Find the minimum number of operations – deletions, insertions, and substitutions – required to transform a to b .

Let $K[i, j]$ be the edit distance between prefixes a_1, \dots, a_i and b_1, \dots, b_j .

We need to find $K(n, m)$.

How to compute $K(i, j)$?

Given: Two strings a_1, \dots, a_n and b_1, \dots, b_m .

Goal: Find the minimum number of operations – deletions, insertions, and substitutions – required to transform \mathbf{a} to \mathbf{b} .

Base case: $K[i, 0] = i$ and $K[0, j] = j$.

Why?

How to compute $K(i, j)$?

Given: Two strings a_1, \dots, a_n and b_1, \dots, b_m .

Goal: Find the minimum number of operations – deletions, insertions, and substitutions – required to transform \mathbf{a} to \mathbf{b} .

Need a recursive formula for $K[i, j]$:

Option A: $a_i = b_j$, then we don't need to change a_i .

Option B: a_i is replaced with b_j .

Option C: a_i is deleted.

Option D: b_j is inserted.

How to compute $K(i, j)$?

Given: Two strings a_1, \dots, a_n and b_1, \dots, b_m .

Goal: Find the minimum number of operations – deletions, insertions, and substitutions – required to transform \mathbf{a} to \mathbf{b} .

Need a recursive formula for $K[i, j]$:

Option A: $a_i = b_j$, then we don't need to change a_i .

$$K[i, j] = K[i - 1, j - 1]$$

Option B: a_i is replaced with b_j .

Option C: a_i is deleted.

Option D: b_j is inserted.

How to compute $K(i, j)$?

Given: Two strings a_1, \dots, a_n and b_1, \dots, b_m .

Goal: Find the minimum number of operations – deletions, insertions, and substitutions – required to transform \mathbf{a} to \mathbf{b} .

Need a recursive formula for $K[i, j]$:

Option A: $a_i = b_j$, then we don't need to change a_i .

Option B: a_i is replaced with b_j .

$$K[i, j] = K[i - 1, j - 1] + 1$$

Option C: a_i is deleted.

Option D: b_j is inserted.

How to compute $K(i, j)$?

Given: Two strings a_1, \dots, a_n and b_1, \dots, b_m .

Goal: Find the minimum number of operations – deletions, insertions, and substitutions – required to transform \mathbf{a} to \mathbf{b} .

Need a recursive formula for $K[i, j]$:

Option A: $a_i = b_j$, then we don't need to change a_i .

Option B: a_i is replaced with b_j .

Option C: a_i is deleted.

$$K[i, j] = K[i - 1, j] + 1$$

Option D: b_j is inserted.

How to compute $K(i, j)$?

Given: Two strings a_1, \dots, a_n and b_1, \dots, b_m .

Goal: Find the minimum number of operations – deletions, insertions, and substitutions – required to transform \mathbf{a} to \mathbf{b} .

Need a recursive formula for $K[i, j]$:

Option A: $a_i = b_j$, then we don't need to change a_i .

Option B: a_i is replaced with b_j .

Option C: a_i is deleted.

Option D: b_j is inserted.

$$K[i, j] = K[i, j - 1] + 1$$

Algorithm for Edit Distance

```
for all  $i, j$ :  
  if  $(a_i = b_j)$   
    a.  $K[i, j] = K[i - 1, j - 1]$   
  else  
     $K[i, j] = \min\{$   
      b.  $K[i - 1, j - 1] + 1,$   
      c.  $K[i - 1, j] + 1,$   
      d.  $K[i, j - 1] + 1\}$ 
```

Edit Distance

- The running time of the algorithm is $O(nm)$.
 - The running time is almost optimal.
 - In practice, people sometimes use approximate methods for computing edit distance.
-

Questions?
