3. Approximate String Matching

Often in applications we want to search a text for something that is similar to the pattern but not necessarily exactly the same.

To formalize this problem, we have to specify what does “similar” mean. This can be done by defining a similarity or a distance measure.

A natural and popular distance measure for strings is the edit distance, also known as the Levenshtein distance.
Edit distance

The edit distance $ed(A, B)$ of two strings $A$ and $B$ is the minimum number of edit operations needed to change $A$ into $B$. The allowed edit operations are:

- S Substitution of a single character with another character.
- I Insertion of a single character.
- D Deletion of a single character.

**Example 3.1:** Let $A = \text{Lewensteinn}$ and $B = \text{Levenshtein}$. Then $ed(A, B) = 3$.

The set of edit operations can be described with an edit sequence: $\text{NNSNNINNNND}$ or with an alignment: $\text{Lewens-teinn}$ $\text{Levenshtein-}$

In the edit sequence, N means No edit.
There are many variations and extension of the edit distance, for example:

- **Hamming distance** allows only the substitution operation.

- **Damerau–Levenshtein distance** adds an edit operation:
  - T **Transposition** swaps two adjacent characters.

- With **weighted edit distance**, each operation has a cost or weight, which can be other than one.

- Allow insertions and deletions (indels) of **factors** at a cost that is lower than the sum of character indels.

We will focus on the basic Levenshtein distance.
Computing Edit Distance

Given two strings $A[1..m]$ and $B[1..n]$, define the values $d_{ij}$ with the recurrence:

$$d_{00} = 0,$$
$$d_{i0} = i, \; 1 \leq i \leq m,$$
$$d_{0j} = j, \; 1 \leq j \leq n,$$
and

$$d_{ij} = \min \left\{ \begin{array}{ll}
    d_{i-1,j-1} + \delta(A[i], B[j]) \\
    d_{i-1,j} + 1 & 1 \leq i \leq m, 1 \leq j \leq n, \\
    d_{i,j-1} + 1
  \end{array} \right\}$$

where

$$\delta(A[i], B[j]) = \begin{cases} 
1 & \text{if } A[i] \neq B[j] \\
0 & \text{if } A[i] = B[j]
\end{cases}$$

**Theorem 3.2:** $d_{ij} = ed(A[1..i], B[1..j])$ for all $0 \leq i \leq m$, $0 \leq j \leq n$. In particular, $d_{mn} = ed(A, B)$. 
Example 3.3: $A = \text{ballad}, B = \text{handball}$

\[
\begin{array}{|c|ccccccccc|}
\hline
 d & h & a & n & d & b & a & l & l & l \\
\hline
 b & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
 a & 1 & 1 & 2 & 3 & 4 & 4 & 5 & 6 & 7 \\
 l & 2 & 2 & 1 & 2 & 3 & 4 & 4 & 5 & 6 \\
 l & 3 & 3 & 2 & 2 & 3 & 4 & 5 & 4 & 5 \\
 l & 4 & 4 & 3 & 3 & 3 & 4 & 5 & 4 & 5 \\
 a & 5 & 5 & 4 & 4 & 4 & 4 & 4 & 4 & 5 \\
 d & 6 & 6 & 5 & 5 & 4 & 5 & 5 & 5 & 6 \\
\hline
\end{array}
\]

$ed(A, B) = d_{mn} = d_{6,8} = 6.$
Proof of Theorem 3.2. We use induction with respect to \( i + j \). For brevity, write \( A_i = A[1..i] \) and \( B_j = B[1..j] \).

**Basis:**
\[
\begin{align*}
    d_{00} &= 0 = \text{ed}(\epsilon, \epsilon) \\
    d_{i0} &= i = \text{ed}(A_i, \epsilon) \quad (i \text{ deletions}) \\
    d_{0j} &= j = \text{ed}(\epsilon, B_j) \quad (j \text{ insertions})
\end{align*}
\]

**Induction step:** We show that the claim holds for \( d_{ij}, 1 \leq i \leq m, 1 \leq j \leq n \). By induction assumption, \( d_{pq} = \text{ed}(A_p, B_q) \) when \( p + q < i + j \).

Let \( E_{ij} \) be an edit sequence with the cost \( \text{ed}(A_i, B_j) \). Such an optimal edit sequence always exists. We have three cases depending on what the last operation symbol in \( E_{ij} \) is:

**N or S:** \( E_{ij} = E_{i-1,j-1}N \) or \( E_{ij} = E_{i-1,j-1}S \) and
\[
\text{ed}(A_i, B_j) = \text{ed}(A_{i-1}, B_{j-1}) + \delta(A[i], B[j]) = d_{i-1,j-1} + \delta(A[i], B[j]).
\]

**I:** \( E_{ij} = E_{i,j-1}I \) and \( \text{ed}(A_i, B_j) = \text{ed}(A_i, B_{j-1}) + 1 = d_{i,j-1} + 1. \)

**D:** \( E_{ij} = E_{i-1,j}D \) and \( \text{ed}(A_i, B_j) = \text{ed}(A_{i-1}, B_j) + 1 = d_{i-1,j} + 1. \)

One of the cases above is always true, and since the edit sequence is optimal, it must be one with the minimum cost, which agrees with the definition of \( d_{ij} \). \(\square\)
The recurrence gives directly a dynamic programming algorithm for computing the edit distance.

**Algorithm 3.4:** Edit distance
Input: strings $A[1..m]$ and $B[1..n]$
Output: $ed(A, B)$

(1) for $i \leftarrow 0$ to $m$ do $d_{i0} \leftarrow i$
(2) for $j \leftarrow 1$ to $n$ do $d_{0j} \leftarrow j$
(3) for $j \leftarrow 1$ to $n$ do
(4) for $i \leftarrow 1$ to $m$ do
(5) $d_{ij} \leftarrow \min\{d_{i-1,j-1} + \delta(A[i], B[j]), d_{i-1,j} + 1, d_{i,j-1} + 1\}$
(6) return $d_{mn}$

The time and space complexity is $O(mn)$. 
The space complexity can be reduced by noticing that each column of the matrix \((d_{ij})\) depends only on the previous column. We do not need to store older columns.

A more careful look reveals that, when computing \(d_{ij}\), we only need to store the bottom part of column \(j - 1\) and the already computed top part of column \(j\). We store these in an array \(C[0..m]\) and variables \(c\) and \(d\) as shown below:

\[
\begin{array}{cccc}
  & d_{0,j-1} & d_{0,j} & C[0..m] \\
  d_{i-1,j-1} & d_{i-1,j} & c & \\
  d_{i,j-1} & d_{i,j} & \quad & \\
  d_{m,j-1} & d_{m,j} & \quad & \\
\end{array}
\]
**Algorithm 3.5:** Edit distance in $O(m)$ space

Input: strings $A[1..m]$ and $B[1..n]$  
Output: $ed(A,B)$

1. for $i ← 0$ to $m$ do $C[i] ← i$
2. for $j ← 1$ to $n$ do
3.   $c ← C[0]; C[0] ← j$
4.   for $i ← 1$ to $m$ do
5.     $d ← \min\{c + \delta(A[i], B[j]), C[i - 1] + 1, C[i] + 1\}$
6.     $c ← C[i]$
7.     $C[i] ← d$
8. return $C[m]$

- Note that because $ed(A, B) = ed(B, A)$ (exercise), we can assume that $m \leq n$. 

105
It is also possible to find optimal edit sequences and alignments from the matrix $d_{ij}$.

An edit graph is a directed graph, where the nodes are the cells of the edit distance matrix, and the edges are as follows:

- If $A[i] = B[j]$ and $d_{ij} = d_{i-1,j-1}$, there is an edge $(i-1, j-1) \rightarrow (i, j)$ labelled with N.
- If $A[i] \neq B[j]$ and $d_{ij} = d_{i-1,j-1} + 1$, there is an edge $(i-1, j-1) \rightarrow (i, j)$ labelled with S.
- If $d_{ij} = d_{i,j-1} + 1$, there is an edge $(i, j-1) \rightarrow (i, j)$ labelled with I.
- If $d_{ij} = d_{i-1,j} + 1$, there is an edge $(i-1, j) \rightarrow (i, j)$ labelled with D.

Any path from $(0,0)$ to $(m,n)$ is labelled with an optimal edit sequence.
Example 3.6: \( A = \text{ballad}, B = \text{handball} \)

There are 7 paths from \((0,0)\) to \((6,8)\) corresponding to 7 different optimal edit sequences and alignments, including the following three:

<table>
<thead>
<tr>
<th>d</th>
<th>h</th>
<th>a</th>
<th>n</th>
<th>d</th>
<th>b</th>
<th>a</th>
<th>l</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>\Rightarrow</td>
<td>2</td>
<td>\Rightarrow</td>
<td>3</td>
<td>\Rightarrow</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>\Rightarrow</td>
<td>1</td>
<td>\Rightarrow</td>
<td>2</td>
<td>\Rightarrow</td>
<td>3</td>
<td>\Rightarrow</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>\Rightarrow</td>
<td>2</td>
<td>\Rightarrow</td>
<td>3</td>
<td>\Rightarrow</td>
<td>4</td>
<td>\Rightarrow</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>\Rightarrow</td>
<td>3</td>
<td>\Rightarrow</td>
<td>3</td>
<td>\Rightarrow</td>
<td>4</td>
<td>\Rightarrow</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>\Rightarrow</td>
<td>4</td>
<td>\Rightarrow</td>
<td>4</td>
<td>\Rightarrow</td>
<td>4</td>
<td>\Rightarrow</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>\Rightarrow</td>
<td>5</td>
<td>\Rightarrow</td>
<td>5</td>
<td>\Rightarrow</td>
<td>4</td>
<td>\Rightarrow</td>
</tr>
</tbody>
</table>
**Approximate String Matching**

Now we are ready to tackle the main problem of this part: approximate string matching.

**Problem 3.7:** Given a text $T[1..n]$, a pattern $P[1..m]$ and an integer $k \geq 0$, report all positions $j \in [1..m]$ such that $ed(P, T[j - \ell...j]) \leq k$ for some $\ell \geq 0$.

The factor $T(j - \ell...j)$ is called an approximate occurrence of $P$.

There can be multiple occurrences of different lengths ending at the same position $j$, but usually it is enough to report just the end positions. We ask for the end position rather than the start position because that is more natural for the algorithms.
Define the values $g_{ij}$ with the recurrence:

\[
\begin{align*}
  g_{ij} &= \min \begin{cases} 
    g_{i-1,j-1} + \delta(P[i], T[j]), \\
    g_{i-1,j} + 1, & 1 \leq i \leq m, 1 \leq j \leq n. \\
    g_{i,j-1} + 1
  \end{cases} \\
  g_{i0} &= i, \quad 1 \leq i \leq m, \text{ and} \\
  g_{0j} &= 0, \quad 0 \leq j \leq n.
\end{align*}
\]

**Theorem 3.8:** For all $0 \leq i \leq m$, $0 \leq j \leq n$:

\[
g_{ij} = \min\{ed(P[1..i], T(j-\ell...j)) \mid 0 \leq \ell \leq j\}.
\]

In particular, $j$ is an ending position of an approximate occurrence if and only if $g_{mj} \leq k$. 
**Proof.** We use induction with respect to $i + j$.

**Basis:**

- $g_{00} = 0 = ed(\epsilon, \epsilon)$
- $g_{0j} = 0 = ed(\epsilon, \epsilon) = ed(\epsilon, T(j - 0..j))$ (min at $\ell = 0$)
- $g_{i0} = i = ed(P[1..i], \epsilon) = ed(P[1..i], T(0 - 0..0))$ ($0 \leq \ell \leq j = 0$)

**Induction step:** Essentially the same as in the proof of Theorem 3.2.
Example 3.9: $P = \text{match}, T = \text{remachine}, k = 1$

<table>
<thead>
<tr>
<th>g</th>
<th>r e m a c h i n e</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>m a t c h</td>
<td>1 1 1 0 1 1 1 1 1 1</td>
</tr>
<tr>
<td>m a t c h</td>
<td>2 2 2 1 0 1 2 2 2 2</td>
</tr>
<tr>
<td>m a t c h</td>
<td>3 3 3 2 1 1 2 3 3 3</td>
</tr>
<tr>
<td>m a t c h</td>
<td>4 4 4 3 2 1 2 3 4 4</td>
</tr>
<tr>
<td>m a t c h</td>
<td>5 5 5 4 3 2 1 2 3 4</td>
</tr>
</tbody>
</table>

One occurrence ending at position 6.
Algorithm 3.10: Approximate string matching
Input: text $T[1..n]$, pattern $P[1..m]$, and integer $k$
Output: end positions of all approximate occurrences of $P$

(1) for $i \leftarrow 0$ to $m$ do $g_{i0} \leftarrow i$
(2) for $j \leftarrow 1$ to $n$ do $g_{0j} \leftarrow 0$
(3) for $j \leftarrow 1$ to $n$ do
(4)   for $i \leftarrow 1$ to $m$ do
(5)      $g_{ij} \leftarrow \min\{g_{i-1,j-1} + \delta(A[i], B[j]), g_{i-1,j} + 1, g_{i,j-1} + 1\}$
(6)   if $q_{mj} \leq k$ then output $j$

- Time and space complexity is $\mathcal{O}(mn)$.
- The space complexity can be reduced to $\mathcal{O}(m)$ by storing only one column as in Algorithm 3.5.
Ukkonen’s Cut-off Heuristic

We can speed up the algorithm using the diagonal monotonicity of the matrix \((g_{ij})\):

A diagonal \(d, -m \leq d \leq n\), consists of the cells \(g_{ij}\) with \(j - i = d\).
Every diagonal in \((g_{ij})\) is monotonically increasing.

Example 3.11: Diagonals -3 and 2.

<table>
<thead>
<tr>
<th>g</th>
<th>r e m a c h i n e</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>m</td>
<td>1 1 1 0 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>a</td>
<td>2 2 2 1 0 1 2 2 2 2 2 2</td>
</tr>
<tr>
<td>t</td>
<td>3 3 3 2 1 1 2 3 3 3 3 3</td>
</tr>
<tr>
<td>c</td>
<td>4 4 4 3 2 1 2 3 4 4 4 4</td>
</tr>
<tr>
<td>h</td>
<td>5 5 5 4 3 2 1 2 3 4 4 4</td>
</tr>
</tbody>
</table>
More specifically, we have the following property.

**Lemma 3.12:** For every $i \in [1..m]$ and every $j \in [1..n]$, $g_{ij} = g_{i-1,j-1}$ or $g_{ij} = g_{i-1,j-1} + 1$.

**Proof.** By definition, $g_{ij} \leq g_{i-1,j-1} + \delta(P[i], T[j]) \leq g_{i-1,j-1} + 1$. We show that $g_{ij} \geq g_{i-1,j-1}$ by induction on $i + j$.

The induction assumption is that $g_{pq} \geq g_{p-1,q-1}$ when $p \in [1..m]$, $q \in [1..n]$ and $p + q < i + j$. At least one of the following holds:

1. $g_{ij} = g_{i-1,j-1} + \delta(P[i], T[j])$. Then $g_{ij} \geq g_{i-1,j-1}$.
2. $g_{ij} = g_{i-1,j} + 1$ and $i > 1$. Then
   $$g_{ij} = g_{i-1,j} + 1 \geq g_{i-2,j-1} + 1 \geq g_{i-1,j-1}$$
3. $g_{ij} = g_{i,j-1} + 1$ and $j > 1$. Then
   $$g_{ij} = g_{i,j-1} + 1 \geq g_{i-1,j-2} + 1 \geq g_{i-1,j-1}$$
4. $g_{ij} = g_{i-1,j} + 1$ and $i = 1$. Then $g_{ij} \geq 0 = g_{i-1,j-1}$.
5. $g_{ij} = g_{i,j-1} + 1$ and $j = 1$. Then $g_{i1} \leq g_{i-1,0} + 1 = g_{i0}$. Thus this case is not possible. \[\square\]
We can reduce computation using diagonal monotonicity:

- Whenever the value on a diagonal $d$ grows larger than $k$, we can discard $d$ from consideration, because we are only interested in values at most $k$ on the row $m$.

- We keep track of the smallest undiscarded diagonal $d$. Each column is computed only up to diagonal $d$.

**Example 3.13:** $P = \text{match}, T = \text{remachine}, k = 1$

| g | r | e | m | a | c | h | i | n | e |
|---|---|---|---|---|---|---|---|---|
| m | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| t | 2 | 2 | 1 | 0 | 1 | 2 | 2 | 2 | 2 |
| c | 1 | 1 | 2 | 3 |   |   |   |   |   |
| h | 1 | 2 | 3 |   |   |   |   |   |   |
|   | 1 | 2 |   |   |   |   |   |   |   |
The position of the smallest undiscarded diagonal on the current column is kept in a variable \( top \).

**Algorithm 3.14**: Ukkonen’s cut-off algorithm  
Input: text \( T[1..n] \), pattern \( P[1..m] \), and integer \( k \)  
Output: end positions of all approximate occurrences of \( P \)  
(1) for \( i \leftarrow 0 \) to \( \min(k, m) \) do \( g_{i0} \leftarrow i \)  
(2) for \( j \leftarrow 1 \) to \( n \) do \( g_{0j} \leftarrow 0 \)  
(3) \( top \leftarrow \min(k + 1, m) \)  
(4) for \( j \leftarrow 1 \) to \( n \) do  
(5) \hspace{0.5cm} for \( i \leftarrow 1 \) to \( top \) do  
(6) \hspace{1cm} \( g_{ij} \leftarrow \min\{g_{i-1,j-1} + \delta(A[i], B[j]), g_{i-1,j} + 1, g_{i,j-1} + 1\} \)  
(7) \hspace{1cm} while \( g_{top,j} > k \) do \( top \leftarrow top - 1 \)  
(8) \hspace{1cm} if \( top = m \) then output \( j \)  
(9) \hspace{1cm} else \( top \leftarrow top + 1 \)
The time complexity is proportional to the computed area in the matrix \((g_{ij})\).

- The worst case time complexity is still \(O(mn)\).
- The average case time complexity is \(O(kn)\). The proof is not trivial.

There are many other algorithms based on diagonal monotonicity. Some of them achieve \(O(kn)\) worst case time complexity.