- The search time of BDM and BOM is $\mathcal{O}(n(\log_\sigma m)/m)$, which is optimal on *average*. (BNDM is optimal only when $m \leq w$.)
- MP and KMP are optimal in the *worst case*.
- There are also algorithms that are optimal in *both* cases. They are based on similar techniques, but they will not be described them here.

Karp–Rabin

The Karp–Rabin algorithm uses a *hash function* $H : \Sigma^* \to [0..q) \subset \mathbb{N}$ for strings. It computes $H(P)$ and $H(T[j..j+m))$ for all $j \in [0..n-m]$.

- If $H(P) \neq H(T[j..j+m))$, then we must have $P \neq T[j..j+m)$.
- If $H(P) = H(T[j..j+m)$, the algorithm compares P and $T[j..j+m)$ in brute force manner. If $P \neq T[j..j+m)$, this is a *collision*.

A good hash function has two important properties:

- Collisions are rare.
- Given $H(a\alpha)$, a and b, where $a, b \in \Sigma$ and $\alpha \in \Sigma^*$, we can quickly compute $H(\alpha b)$. This is a called *rolling* or *sliding window* hash function.

The latter property is essential for fast computation of $H(T[j..j+m))$ for all j.

The hash function used by Karp–Rabin is

$$H(c_0 c_1 c_2 \ldots c_{m-1}) = (c_0 r^{m-1} + c_1 r^{m-2} + \cdots + c_{m-2} r + c_{m-1}) \bmod q$$

This is a rolling hash function:

$$H(\alpha) = (H(a\alpha) - a r^{m-1}) \bmod q$$
$$H(\alpha b) = (H(\alpha) \cdot r + b) \bmod q$$

which follows from the rules of *modulo arithmetic*:

$$(x + y) \bmod q = ((x \bmod q) + (y \bmod q)) \bmod q$$
$$(xy) \bmod q = ((x \bmod q)(y \bmod q)) \bmod q$$

The parameters q and r have to be chosen with some care to ensure that collisions are rare.

- The original choice is $r = \sigma$ and q is a large *prime*.
- Another possibility is $q = 2^w$, where w is the machine word size, and r is a small prime ($r = 37$ has been suggested). This is faster in practice, because it avoids slow modulo operations.
- The hash function can be *randomized* by choosing q or r randomly. Furthermore, we can change q or r whenever a collision happens.

Algorithm 1.17: Karp-Rabin

Input: text $T = T[0 \ldots n)$, pattern $P = P[0 \ldots m)$
Output: position of the first occurrence of P in T

(1) Choose q and r; $s \leftarrow r^{m-1} \bmod q$
(2) $hp \leftarrow 0; ht \leftarrow 0$
(3) *for* $i \leftarrow 0$ *to* $m-1$ *do* $hp \leftarrow (hp \cdot r + P[i]) \bmod q$ // $hp = H(P)$
(4) *for* $j \leftarrow 0$ *to* $m-1$ *do* $ht \leftarrow (ht \cdot r + T[j]) \bmod q$
(5) *for* $j \leftarrow 0$ to $n-m-1$ *do*
(6) *if* $hp = ht$ *then if* $P = T[j \ldots j+m)$ *then* return j
(7) $ht \leftarrow ((ht - T[j] \cdot s) \cdot r + T[j+m]) \bmod q$
(8) *if* $hp = ht$ *then if* $P = T[j \ldots j+m)$ *then* return j
(9) return n

On an integer alphabet:

- The worst case time complexity is $\mathcal{O}(mn)$.
- The average case time complexity is $\mathcal{O}(m+n)$.

Karp–Rabin is not competitive in practice, but hashing can be a useful technique in other contexts.

2. 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 2.1: Let $A =$ `Lewensteinn` and $B =$ `Levenshtein`. Then $ed(A, B) = 3$.

The set of edit operations can be described with an *edit sequence* or with an *alignment*:

```
NNSNNNINNNND
Lewens-teinn
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 subtitution 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*:

$$\begin{aligned}
d_{00} &= 0, \\
d_{i0} &= i, \ 1 \le i \le m, \\
d_{0j} &= j, \ 1 \le j \le n, \ \text{and} \\
d_{ij} &= \min \begin{cases} d_{i-1,j-1} + \delta(A[i], B[j]) \\ d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \end{cases} \quad 1 \le i \le m, 1 \le j \le n,
\end{aligned}$$

where

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

Theorem 2.2: $d_{ij} = ed(A[1..i], B[1..j])$ for all $0 \le i \le m$, $0 \le j \le n$. In particular, $d_{mn} = ed(A, B)$.

Example 2.3: $A = \texttt{ballad}, B = \texttt{handball}$

d		h	a	n	d	b	a	l	l
	0	1	2	3	4	5	6	7	8
b	1	1	2	3	4	4	5	6	7
a	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	5	4
a	5	5	4	4	4	4	4	5	5
d	6	6	5	5	4	5	5	5	6

$ed(A, B) = d_{mn} = d_{6,8} = 6$.

Proof of Theorem 2.2. We use induction with respect to $i + j$. For brevity, write $A_i = A[1..i]$ and $B_j = B[1..j]$.

Basis:

$$d_{00} = 0 = ed(\epsilon, \epsilon)$$
$$d_{i0} = i = ed(A_i, \epsilon) \quad (i \text{ deletions})$$
$$d_{0j} = j = ed(\epsilon, B_j) \quad (j \text{ insertions})$$

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

The value $ed(A_i, B_j)$ is based on an *optimal edit sequence*. We have three cases depending on what the last edit operation is:

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

I: $ed(A_i, B_j) = ed(A_i, B_{j-1}) + 1 = d_{i,j-1} + 1$.

D: $ed(A_i, B_j) = ed(A_{i-1}, B_j) + 1 = d_{i-1,j} + 1$.

Since the edit sequence is optimal, the correct value is the minimum of the three cases, which agrees with the definition of d_{ij}. □

The recurrence gives directly a *dynamic programming* algorithm for computing the edit distance.

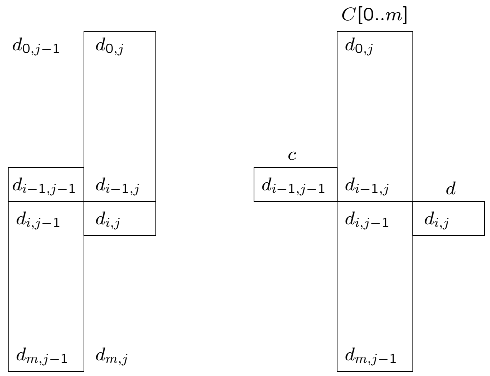
Algorithm 2.4: Edit distance
Input: strings $A[1..m]$ and $B[1..n]$
Output: $ed(A, B)$

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

The time and space complexity is $\mathcal{O}(mn)$.

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:



Algorithm 2.5: Edit distance in $\mathcal{O}(m)$ space
Input: strings $A[1..m]$ and $B[1..n]$
Output: $ed(A, B)$

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

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

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 2.6: $A = \texttt{ballad}, B = \texttt{handball}$

$$
\begin{array}{c|ccccccccccccccccc|}
d & & \texttt{h} & & \texttt{a} & & \texttt{n} & & \texttt{d} & & \texttt{b} & & \texttt{a} & & \texttt{l} & & \texttt{l} & \\
\hline
 & 0 & \Rightarrow & 1 & \Rightarrow & 2 & \Rightarrow & 3 & \Rightarrow & 4 & \rightarrow & 5 & \rightarrow & 6 & \rightarrow & 7 & \rightarrow & 8 \\
\texttt{b} & \downarrow & \Searrow & & \searrow & & \searrow & & \searrow & & \Searrow & & & & & & & \\
 & 1 & & 1 & \rightarrow & 2 & \rightarrow & 3 & \rightarrow & 4 & & 4 & \rightarrow & 5 & \rightarrow & 6 & \rightarrow & 7 \\
\texttt{a} & \downarrow & \searrow & \downarrow & \Searrow & & & & & & & & \Searrow & & & & & \\
 & 2 & & 2 & & 1 & \Rightarrow & 2 & \rightarrow & 3 & \rightarrow & 4 & & 4 & \rightarrow & 5 & \rightarrow & 6 \\
\texttt{l} & \downarrow & \searrow & \downarrow & & \downarrow & \Searrow & & \Searrow & & \searrow & & \searrow & \downarrow & \Searrow & & \searrow & \\
 & 3 & & 3 & & 2 & & 2 & \Rightarrow & 3 & \rightarrow & 4 & \rightarrow & 5 & & 4 & \rightarrow & 5 \\
\texttt{l} & \downarrow & \searrow & \downarrow & & \downarrow & \searrow & \downarrow & \Searrow & & \Searrow & & \searrow & & \searrow & \downarrow & \Searrow & \\
 & 4 & & 4 & & 3 & & 3 & & 3 & \Rightarrow & 4 & \rightarrow & 5 & & 5 & & 4 \\
\texttt{a} & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow & \downarrow & \searrow & & \Searrow & & & & & \Downarrow \\
 & 5 & & 5 & & 4 & & 4 & & 4 & & 4 & & 4 & \Rightarrow & 5 & & 5 \\
\texttt{d} & \downarrow & \searrow & \downarrow & & \downarrow & \searrow & \downarrow & \searrow & & \searrow & \downarrow & \searrow & \downarrow & \Searrow & & \Searrow & \Downarrow \\
 & 6 & & 6 & & 5 & & 5 & & 4 & \rightarrow & 5 & & 5 & & 5 & \Rightarrow & 6 \\
\hline
\end{array}
$$

There are 7 paths from $(0, 0)$ to $(6, 8)$ corresponding to, for example, the following edit sequences and alignments:

```
IIIINNNNDD   SNISSNIS   SNSSINSI
----ballad   ba-lla-d   ball-ad-
handball--   handball   handball
```