When we are finished we find an optimal alignment just by following the traceback from the maximum score:

\[
(1 - f')y \quad 1 - f'z \quad 1 - f'x
\]

The traceback points at the cell that led to the maximal score:

\[
(1 - f')y \quad 1 - f'z \quad 1 - f'x
\]

Clearly infeasible to enumerate all possible alignments.

We want to find an optimal alignment between two sequences by using dynamic programming. This 'filling in the table' is called dynamic programming (Bellman 1955).

When filling in the table, we record the traceback from each cell. For example, in the table below, the traceback from cell (2, 2) is to the cell above and to the cell on the left.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

The traceback from (2, 2) is to the cell above (1, 2) and to the cell on the left (2, 1). This is because an optimal alignment between \(X = x_1x_2x_3\) and \(Y = y_1y_2y_3\) with a score of \(j\) consists of:

- \(x_1\) matched with \(y_1\) (match), or
- \(x_1\) matched with a gap (gap), or
- \(y_1\) matched with a gap (gap).

The score of an alignment with one gap is \(j - \Delta\), and the score of an alignment with two gaps is \(j - 2\Delta\). The score of a match is \(j + \beta\), and the score of a gap is \(-\gamma\).

So an optimal alignment can be computed by using the values in the current row.

Observation 1:

\[
(1 - f')y \quad 1 - f'z \quad 1 - f'x
\]

The value in the current cell is the maximum score of any combination of alignments.

If \(D_{i,j}\) is the maximum score for a global alignment of \(X_i\) and \(Y_j\), then \(D_{i+1,j+1}\) is the maximum score of any combination of alignments of \(X_i\) and \(Y_j\) with a gap, or

\[
D_{i+1,j+1} = \max\left( D_{i,j+1}, D_{i+1,j} \right)
\]

Observation 2:

\[
(1 - f')y \quad 1 - f'z \quad 1 - f'x
\]

The values \((1 - f')y\), \((1 - f')z\), and \((1 - f')x\) are the maximum scores for any combination of alignments of \(X_i\) and \(Y_j\) with one gap, two gaps, or no gaps, respectively.

These values can be computed by using the values in the current row and the values in the current column.

\[
(1 - f')y \quad 1 - f'z \quad 1 - f'x
\]

The value in the current cell is the maximum score of any combination of alignments of \(X_i\) and \(Y_j\) with one gap, two gaps, or no gaps, respectively.

If \(D_{i,j}\) is the maximum score for a local alignment of \(X_i\) and \(Y_j\), then \(D_{i+1,j+1}\) is the maximum score of any combination of alignments of \(X_i\) and \(Y_j\) with a gap, or

\[
D_{i+1,j+1} = \max\left( D_{i,j+1}, D_{i+1,j} \right)
\]
Implementing global alignment: Initialization

Upper border: position \( \alpha \) represents the alignment of \( \alpha \) to the empty prefix of \( \alpha \).

With simple linear gap costs, the score is \(-5\).

The traceback pointer at \( \alpha \) points to \( \alpha \).

The left-hand border is similar.

Hence we initialize the borders as follows:

```java
for(int i=1;i<=n;i++)
    F[i][0]=-d*i;
    B[i][0]=newTraceback2(i-1,0);

for(int j=1;j<=m;j++)
    F[0][j]=-d*j;
    B[0][j]=newTraceback2(0,j-1);
```

Implementing global alignment: Filling in the matrix

Position \( \alpha \) may be reached from \( \alpha \) with a match, adding \( \alpha \) to the score; from \( \alpha \) with a gap in \( \alpha \), subtracting \( \alpha \) from the score; or from \( \alpha \) with a gap in \( \alpha \), subtracting \( \alpha \) from the score.

The traceback \( \alpha \) points to the source of the maximal resulting score \( \alpha \).

Thus:

```java
for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
        int s = score[seq1.charAt(i-1)][seq2.charAt(j-1)];
        int val = max(F[i-1][j-1]+s,F[i-1][j]-d,F[i][j-1]-d);
        F[i][j]=val;
        if(val==F[i-1][j-1]+s)
            B[i][j]=newTraceback2(i-1,j-1);
        else if(val==F[i-1][j]-d)
            B[i][j]=newTraceback2(i-1,j);
        else if(val==F[i][j-1]-d)
            B[i][j]=newTraceback2(i,j-1);
```

The start \( \alpha \) of the traceback is cell \( \alpha \).

The traceback is recorded in a matrix \( \alpha \) with the same shape as \( \alpha \).

```
  | A | V | A | H | M | V | d |
--|--|--|--|--|--|--|--|
  |   |   |   |   |   |   |   |
```

Filling in the \( \alpha \) matrix for global alignment of \( \alpha \) — HEMAGREE and \( \alpha \) — PARMAE.
Local alignment of

HEAGAWGHEE

and

PAWHEAE

(Smith-Waterman)

A subsequence of

AWSGA

must be aligned with a subsequence of

AW-HEE

Requirement: the expected score of a random match must be negative. If the score of a random match extension were positive, then any local alignment could be profitably extended to a "better" (but probably biologically meaningless) one.

New interpretation of

\( \% \) & \( \& \)

\( \% \) & \( \& \)

the maximal score for an alignment between a suffix of

HEAGAWGHEE

and a suffix of

PAWHEAE

Implementing local alignment: Initialization

Upper border: position \( \) & \( \)

represent the alignment of a suffix of

HEAGAWGHEE

to an empty sequence. An empty match, with score 0, is the best we can do (provided gap scores have negative nums). Then \( \) & \( \)

is the start of a new local alignment, and the traceback pointer at \( \) & \( \)

points nowhere.

The left-hand border is similar. Hence we initialize the border cells to 0 and the traceback to null (this requires no action in Java).

Implementing local alignment: Filling in the matrix

Position \( \)

may be reached from nowhere, with score 0, because we can always start a new local alignment; from \( \)

with a match, adding \( \)

\( \) & \( \)

\( \)

\( \)

\( \)

\( \) with a gap in \( \)

\( \)

subtracting \( \)

\( \)

\( \) with a gap in \( \)

\( \)

subtracting \( \)

\( \)

\( \) with a gap in \( \)

\( \)

subtracting \( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)

\( \)
Implementing repeated matches:

Initialization

Left-hand border:

Position \( (i, j) \) represents the best alignment of an empty subsequence of \( T \) to a subsequence of \( S \). This must have score 0.

The traceback pointer at \( (i, j) \) points nowhere.

Filling in the matrix

Position \( (i, j) \) may be reached from \( (i-1, j) \) by letting \( T \) be unmatched by any part of \( S \), keeping the old score; or from \( (i, j-1) \) by completing a match whose score is at least \( 8 \), subtracting \( 8 \) from that score.

Position \( (i, j) \) for \( 9 \) may be reached from \( (i-1, j) \), because we start a new local alignment, keeping the old score; from \( (i, j-1) \) with a match, adding \( 9 \) to the score; or from \( (i, j) \) with a gap in \( S \), subtracting \( 9 \) from the score.

As always, the traceback points to the source of the maximal resulting score.

Let

\[
\max_j(i-1) = \begin{cases} 
9 & \text{if } \max_j(i-1) \text{ is greater than } \max_j(i-1) \text{ and maximal; otherwise } 0.
\end{cases}
\]

This gives:

\[
\text{for}(i=1; i<=n; i++) \{ \\
\text{int } \max_j = \max_j(i-1); \\
\text{int } \text{val} = \max(\text{F}[i][0], \text{F}[i-1][j-1]+\text{s}, \text{F}[i-1][j]-d, \text{F}[i][j-1]-d); \\
\text{F}[i][j] = \text{val}; \\
\text{if } (\text{val} == \text{F}[i][0]) \\
\text{B}[i][j] = \text{new Traceback2}(i, 0); \\
\text{elseif } (\text{val} == \text{F}[i-1][j-1]+\text{s}) \\
\text{B}[i][j] = \text{new Traceback2}(i-1, j-1); \\
\text{elseif } (\text{val} == \text{F}[i-1][j]-d) \\
\text{B}[i][j] = \text{new Traceback2}(i-1, j); \\
\text{elseif } (\text{val} == \text{F}[i][j-1]-d) \\
\text{B}[i][j] = \text{new Traceback2}(i, j-1); \\
\text{\} \\
\}
\]

The start of the traceback is \( (\max J, \max I) \).

Overlap matches between

\[
\text{HEAGAWGHEE} \\
\text{PAWHEAE}
\]

A prefix or suffix of \( T \) may be aligned with a prefix or suffix of \( S \), keeping the old score.

Overlap matches between \( T \) and \( S \):

Filling in the matrix

The traceback pointer at \( (f^T, f^S) \) points to the source of the maximal resulting score.

The traceback pointer at \( (f^T, f^S) \) points to the source of the maximal resulting score.

Overlap matches between

\[
\text{HEAGAWGHEE} \\
\text{PAWHEAE}
\]

A prefix or suffix of \( T \) must be aligned with a prefix or suffix of \( S \), keeping the old score.

Overlap matches between \( T \) and \( S \):

Filling in the matrix
Reducing the space consumption of global alignment

All algorithms require time to fill in the tables and space to store the tables. However, column $\%$ depends only on column $\%$. So only two columns of $\%$ and the traceback need to be stored at the same time.

Hence we can compute the optimal global alignment and traceback using only space $O(n \log n)$.

How can we compute the optimal local alignment?

We can use the same approach by finding the optimal global alignment of the sequences and the traceback.

Reducing the space consumption of local alignment

Fill in $\%$ using only space $O(1)$. Keep track of the starting point $\%' and the ending point $\%' of the local alignment with highest score.

The result is the optimal local alignment between $X$ and $Y$. We can determine $\%$ while filling in $\%$.

How do we reconstruct the optimal global alignment?

When $\% < \%$ or $\% < \%$, use the standard algorithm (in this case it uses little space anyway).

Otherwise, let $\% = \%$ and assume the optimal alignment passes through $\% = \%$. We can determine $\%$ while filling in $\%$. Recursively determine the optimal global alignment between $A$ and $B$; and the optimal global alignment between $C$ and $D$.

The result is the concatenation of $\%$ and $\%$.