# 6.6 Sequence Alignment

# String Similarity

How similar are two strings?

- `ocurrance`
- `occurrence`

First model the problem…

Q. How can we measure the distance?

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| o | c | u | r | r | a | n | c | e | - |
| o | c | c | u | r | r | e | n | c | e |

6 mismatches, 1 gap

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| o | c | - | u | r | r | a | n | c | e |
| o | c | c | u | r | r | e | n | c | e |

1 mismatch, 1 gap

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| o | c | - | u | r | r | - | a | n | c | e |
| o | c | c | u | r | r | e | - | n | c | e |

0 mismatches, 3 gaps

TU Delft

# String Similarity

## How similar are two strings?

- **`ocurrance`**
- **`occurrence`**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| o | c | u | r | r | a | n | c | e | - |
| o | c | c | u | r | r | e | n | c | e |

6 mismatches, 1 gap

## First model the problem…

Q. How can we measure the distance?

A. Idea: best of all possibilities

- Penalty for mismatches
  (depending on characters)
- Penalty for gaps

Minimize total penalty

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| o | c | - | u | r | r | a | n | c | e |
| o | c | c | u | r | r | e | n | c | e |

1 mismatch, 1 gap

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| o | c | - | u | r | r | - | a | n | c | e |
| o | c | c | u | r | r | e | - | n | c | e |

0 mismatches, 3 gaps

TU Delft

# Edit Distance

Applications.

- Basis for Unix diff.
- Speech recognition.
- Spelling suggestions in document editor.
- Computational biology.

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty $\delta$; mismatch penalty $\alpha_{pq}$ of chars p and q.
- Cost = sum of gap and mismatch penalties.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| C | T | G | A | C | C | T | A | C | C | T |
| C | C | T | G | A | C | T | A | C | A | T |

$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| - | C | T | G | A | C | C | T | A | C | C | T |
| C | C | T | G | A | C | - | T | A | C | A | T |

$2\delta + \alpha_{CA}$

# Sequence Alignment

Goal: Given two strings $X = x_1 x_2 \dots x_m$ and $Y = y_1 y_2 \dots y_n$ find *alignment M* of minimum cost.

Def. An alignment M is a set of ordered pairs $x_i$-$y_j$ such that each item occurs in at most one pair and no crossings.

Def. The pair $x_i$-$y_j$ and $x_{i'}$-$y_{j'}$ cross if $i < i'$, but $j > j'$.

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta}_{\text{gap}}$$

# Sequence Alignment

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i\,:\,x_i \text{ unmatched}} \delta + \sum_{j\,:\,y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Q. `CTACCG` vs. `TACATG`.

Suppose all α=1 for all mismatches and δ=1, what then is cost of M = {$x_2$-$y_1$, $x_3$-$y_2$, $x_4$-$y_3$, $x_5$-$y_4$, $x_6$-$y_6$} ?

A.

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | |
|---|---|---|---|---|---|---|
| C | T | A | C | C | G | |
| | T | A | C | A | T | G |
| | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ |

# Sequence Alignment

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i\,:\,x_i \text{ unmatched}} \delta + \sum_{j\,:\,y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Q. `CTACCG` vs. `TACATG`.

Suppose all $\alpha=1$ for all mismatches and $\delta=1$, what then is cost of $M = \{x_2\text{-}y_1, x_3\text{-}y_2, x_4\text{-}y_3, x_5\text{-}y_4, x_6\text{-}y_6\}$ ?

A. 3

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | | $x_6$ |
| C | T | A | C | C | - | G |
| - | T | A | C | A | T | G |
| | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $y_6$ |

# Sequence Alignment: Problem Structure

**Def.** OPT(i, j) = min cost of aligning strings $x_1\ x_2 \ldots x_i$ and $y_1\ y_2 \ldots y_j$.

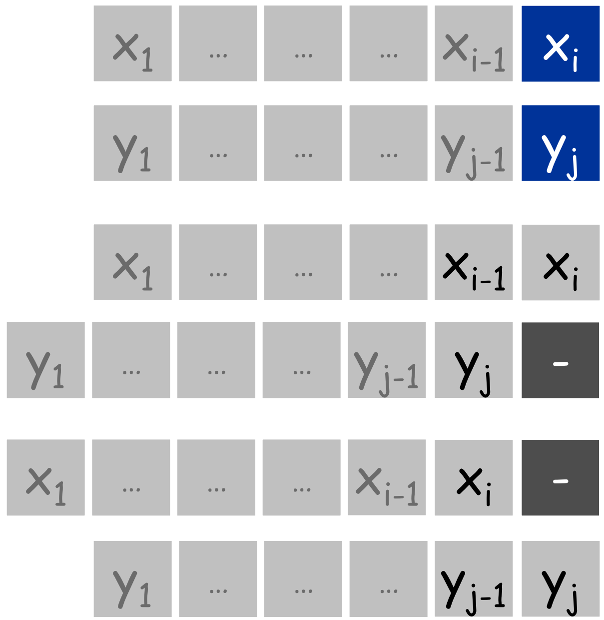
**Q.** How to define OPT recursively? What are the cases? (1 min)

| | | | | | |
|---|---|---|---|---|---|
| $x_1$ | … | … | … | $x_{i-1}$ | $x_i$ |
| $y_1$ | … | … | … | $y_{j-1}$ | $y_j$ |

# Sequence Alignment: Problem Structure

Def. OPT(i, j) = min cost of aligning strings $x_1\ x_2\ .\ .\ .\ x_i$ and $y_1\ y_2\ .\ .\ .\ y_j$.

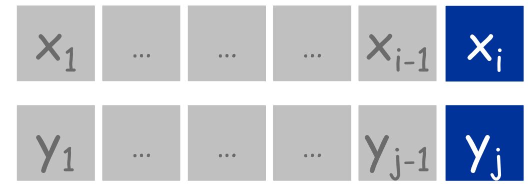
- Case 1: OPT matches $x_i$ with $y_j$.
- Case 2a: OPT leaves $x_i$ unmatched.
- Case 2b: OPT leaves $y_j$ unmatched.



# Sequence Alignment: Problem Structure

Def. OPT(i, j) = min cost of aligning strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.

- Case 1: OPT matches $x_i$ with $y_j$.
  - pay mismatch for $x_i$ with $y_j$ + min cost of aligning $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_{j-1}$



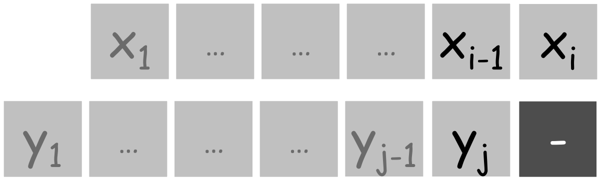

- Case 2a: OPT leaves $x_i$ unmatched.

- Case 2b: OPT leaves $y_j$ unmatched.

# Sequence Alignment: Problem Structure

Def. OPT(i, j) = min cost of aligning strings $x_1\ x_2 \ldots x_i$ and $y_1\ y_2 \ldots y_j$.

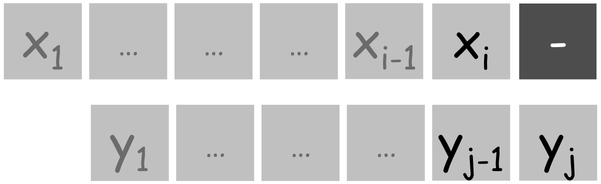
- Case 1: OPT matches $x_i$ with $y_j$.
  - pay mismatch for $x_i$ with $y_j$ + min cost of aligning $x_1\ x_2 \ldots x_{i-1}$ and $y_1\ y_2 \ldots y_{j-1}$
- Case 2a: OPT leaves $x_i$ unmatched.
  - pay gap for $x_i$ and min cost of aligning $x_1\ x_2 \ldots x_{i-1}$ and $y_1\ y_2 \ldots y_j$
- Case 2b: OPT leaves $y_j$ unmatched.



# Sequence Alignment:  Problem Structure

Def.  OPT(i, j) = min cost of aligning strings $x_1\ x_2\ .\ .\ .\ x_i$ and $y_1\ y_2\ .\ .\ .\ y_j$.

- Case 1:  OPT matches $x_i$ with $y_j$.
  - pay mismatch for $x_i$ with $y_j$ + min cost of aligning $x_1\ x_2\ .\ .\ .\ x_{i-1}$ and $y_1\ y_2\ .\ .\ .\ y_{j-1}$
- Case 2a:  OPT leaves $x_i$ unmatched.
  - pay gap for $x_i$ and min cost of aligning $x_1\ x_2\ .\ .\ .\ x_{i-1}$ and $y_1\ y_2\ .\ .\ .\ y_j$
- Case 2b:  OPT leaves $y_j$ unmatched.
  - pay gap for $y_j$ and min cost of aligning $x_1\ x_2\ .\ .\ .\ x_i$ and $y_1\ y_2\ .\ .\ .\ y_{j-1}$




Q. What to do if one of the strings (subproblems) is empty?

# Sequence Alignment:  Problem Structure

Def.  OPT(i, j) = min cost of aligning strings $x_1\ x_2\ .\ .\ .\ x_i$ and $y_1\ y_2\ .\ .\ .\ y_j$.

- Case 1:  OPT matches $x_i$ with $y_j$.
  – pay mismatch for $x_i$ with $y_j$ + min cost of aligning $x_1\ x_2\ .\ .\ .\ x_{i-1}$ and $y_1\ y_2\ .\ .\ .\ y_{j-1}$
- Case 2a:  OPT leaves $x_i$ unmatched.
  – pay gap for $x_i$ and min cost of aligning $x_1\ x_2\ .\ .\ .\ x_{i-1}$ and $y_1\ y_2\ .\ .\ .\ y_j$
- Case 2b:  OPT leaves $y_j$ unmatched.
  – pay gap for $y_j$ and min cost of aligning $x_1\ x_2\ .\ .\ .\ x_i$ and $y_1\ y_2\ .\ .\ .\ y_{j-1}$

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_1$ | … | … | … | $x_{i-1}$ | $x_i$ | - |
| | $y_1$ | … | … | … | $y_{j-1}$ | $y_j$ |

$$
OPT(i, j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}
$$

# Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, x₁x₂...xₘ, y₁y₂...yₙ, δ, α) {
   for i = 0 to m
      M[i, 0] = iδ
   for j = 0 to n
      M[0, j] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xᵢ, yⱼ] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
}
```

Q. How to prove correctness of such an algorithms?

# Proving Correctness of Dynamic Programming Approaches

Thm. **Sequence-Alignment** gives minimal cost of possible alignment.

Pf.

## Proving Correctness of Dynamic Programming Approaches

Thm. **Sequence-Alignment** gives minimal cost of possible alignment.

Pf. (by induction)

**Base**: by definition of cost: if m=0, we have n gap penalties; similar if n=0

**IH**: Suppose **Sequence-Alignment** gives the minimal cost of any possible alignment up to lengths i and j-1 and up i-1 and j.

**Step**: Let x and y of length i and j be given.

TU Delft

## Proving Correctness of Dynamic Programming Approaches

Thm. **Sequence-Alignment** gives minimal cost of possible alignment.

Pf. (by induction)

**Base**: by definition of cost: if m=0, we have n gap penalties; similar if n=0

**IH**: Suppose **Sequence-Alignment** gives the minimal cost of any possible alignment up to lengths i and j-1 and up i-1 and j.

**Step**: Let x and y of length i and j be given.

- Then there are three options for the alignment of the last characters:
- Case 1: OPT matches $x_i$ with $y_j$.
  - pay mismatch for $x_i$ with $y_j$ + min cost of aligning up to i-1 and j-1
- Case 2a: OPT leaves $x_i$ unmatched.
  - pay gap for $x_i$ and min cost of aligning up to i-1 and j
- Case 2b: OPT leaves $y_j$ unmatched.
  - pay gap for $y_j$ and min cost of aligning up to i and j-1
- The algorithm takes exactly the minimum of these three options.

With induction on both i and j, the theorem now follows.

# Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {
   for i = 0 to m
      M[i, 0] = iδ
   for j = 0 to n
      M[0, j] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
}
```

Q. What is time + space complexity?

# Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {
   for i = 0 to m
      M[i, 0] = iδ
   for j = 0 to n
      M[0, j] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
}
```

Q. What is time + space complexity? A. $\Theta(mn)$ time and space.

English words or sentences: m, n $\leq$ 10.

Computational biology: m = n = 100 000. 10 billions ops OK, but 10GB array?

# Sequence Alignment: Linear Space

Q. How to avoid quadratic space when only interested in the value? (1 min)

$$OPT(i, j) = \begin{cases} j\delta & \text{if i = 0} \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if j = 0} \end{cases}$$

TU Delft

# Sequence Alignment: Linear Space

Q. How to avoid quadratic space when only interested in the value?

A. We can calculate the optimal value in O(m + n) space and O(mn) time.

- Compute OPT(i, •) from OPT(i-1, •). Re-use space for "row i-1".
- No longer a simple way to recover alignment itself.

$$OPT(i, j) = \begin{cases} j\delta & \text{if i = 0} \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if j = 0} \end{cases}$$

TU Delft

# Sequence Alignment: Linear Space

Q. How to avoid quadratic space when only interested in the value?

A. We can calculate the optimal value in O(m + n) space and O(mn) time.

- Compute OPT(i, •) from OPT(i-1, •). Re-use space for “row i-1”.
- No longer a simple way to recover alignment itself.

Q*. How can we still get the solution as well?

# Sequence Alignment: Linear Space

Q. How to avoid quadratic space when only interested in the value?

A. We can calculate the optimal value in O(m + n) space and O(mn) time.

- Compute OPT(i, •) from OPT(i-1, •). Re-use space for "row i-1".
- No longer a simple way to recover alignment itself.

Q*. How can we still get the solution as well?

Theorem. [Hirschberg 1975] Optimal alignment in O(m + n) space and O(mn) time.

- Clever combination of divide-and-conquer and dynamic programming.
- Inspired by idea of Savitch from complexity theory.