Institut für Informatik
Praktische Informatik und Bioinformatik
Prof. Dr. Ralf Zimmer
Markus Joppich
Yauheniya Zhdanovich
Felix Offensperger
algo2@bio.ifi.lmu.de

Wintersemester 2021/22

Exercises for Algorithmic Bioinformatics II

Assignment 1

Deadline: Monday, 01 Nov 2021, 09.42 AM
Hand-in online via Moodle (program source code + (tex code+pdf or hand-written pdf)).
If you have any questions, **please ask in time**.

Exercise 1 (Sequence alignments, 10P):

Remember what you learned about sequence alignments in the Programmierpraktikum (and reference the lecture slides).

(a) Explain the "Waterman trick". Show that an alignment reported by SW is optimal, i.e. can not be improved by extending.

(b) The dynamic programming (DP) recursion equation of the Needleman-Wunsch (NW) algorithm for arbitrary gap costs is defined as:

$$\begin{aligned} S_{i,j} &= \max(S_{i-1,j-1} + \alpha_{i,j},\, L_{i,j},\, O_{i,j}) \\ L_{i,j} &= \max_{1 \le k < j} (S_{i,j-k} - g_L(i,j,k)) \\ O_{i,j} &= \max_{1 \le k < i} (S_{i-k,j} - g_O(i,j,k)) \end{aligned}$$

where $\alpha_{i,j}$ denotes substitution costs and $g_{L|O}(i,j,k)$ costs for a gap of length k up to position (i,j).

Derive the running time of an alignment of two sequences (both length n) with the NW algorithm, assuming that $g_{L|O}(i,j,k)$ can be computed in constant time.

(c) Start with the NW equation and derive the recursion equation of the Gotoh algorithm for linear gap costs:

$$\begin{aligned} S_{i,j} &= \max(S_{i-1,j-1} + \alpha_{i,j},\, L_{i,j},\, O_{i,j}) \\ L_{i,j} &= \max(S_{i,j-1} - g_o, L_{i,j-1} - g_e) \\ O_{i,j} &= \max(S_{i-1,j} - g_o, O_{i-1,j} - g_e) \end{aligned}$$

where g_o and g_e denote the costs for gap-opening and gap-extension.

Derive the running time of an alignment of two sequences (both length n) with the Gotoh algorithm.

Hint: Linear gap costs do not depend on the current position in the matrix nor whether they are calculated for row or column.

Exercise 2 (Fibonacci, 15P):

The lecture has discussed three algorithms for computing Fibonacci numbers. These algorithms have different complexities. Please also hand in your easy-to-understand code.

(a) Implement the different algorithmic versions.

(b) Fibonacci numbers grow quite fast and become big. How many Fibonacci numbers can you compute with 32bit, 64bit and 128bit integers?

(c) Analyse your implementations. How many function calls are needed? How many additions, multiplications? How does that correlate with the actual runtime of the implementation? How is the growth of the Fibonacci numbers? Give estimations, proofs and visualizations as evidence for your findings!

(d) There is an "explicit" formula (Binet, 1843) for computing Fibonacci numbers. Prove this formula (by induction) and implement it. Then compare this version with your other implementations. How efficient is your implementation of this formula?

Exercise 3 (Alignments, 15P):

For a given pair of sequences of length n and m, there is an exponential number of possible alignments. The optimal alignments (for a given scoring function) is a subset of the possible alignments (maybe only one, "the" optimal alignment). Justify your answers, e.g. by explaining your source code.

(a) Compute *the number* of possible alignments with the recursive formula given in the lecture.

(b) Compute *the number* of alignments with dynamic programming.

(c) Compare the runtimes of both versions and justify your results.

(d) The lecture gives several theoretical estimations (e.g. Laquer Theorem) for the number of alignments of sequences of length n (with and without "mutual" gaps). Compare the estimations with the actual numbers computed above.