fasebare_newual

Synopsis

The crate `needleman_wunsch` of the `fasebare_newual` package consists of two Rust modules:

- `fasta_multiple_cmp` provides functions to read biological sequences (DNA, RNA or proteins) in FASTA formatted files;
- `sequences_matrix` provides functions to build an alignment matrix of two sequences and to compute their similarity score, according to the Needleman-Wunsch algorithm (https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm).

The `Data` directory contains test data with artificial sequence data, and also true sequences extracted from the Genbank databank, in order to try the programs.

The `cargo` build system will build a standalone program to be invoked from the command line. The program has been built and run only with the Linux OS, but maybe it would run with other OS.

Motivation

The first aim of this package was for the author to learn the programming language Rust, and to apply it to a domain he knows a bit, Bioinformatics. The author's site (https://laurentbloch.net/MySpip3/-Rust-) gives some explanations of this approach (in French…).

These programs are intended for pedagogic use, if you use them for professional or scientific projects, it will be at your own risks.

Credits

These programs invoke the following crate: `simple_matrix`

To take into account the dependency to this package, the `Cargo.toml` file must be:

```
[package]
name = "needleman_wunsch"
version = "0.1.0"
authors = ["Laurent Bloch <lb@laurentbloch.org>"]
edition = "2018"

# See more keys and their definitions at
# https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
simple-matrix = "0.1"
```

The author found hints and inspiration from totof2000 (https://linuxfr.org/forums/programmationautre/posts/rust-lire-des-donnees-de-type-i8-depuis-un-fichier), Unknown (https://www.it-swarm-fr.com/fr/file-io/quelle-est-la-maniere-de-facto-de-lire-et-decrire-des-fichiers-dans-rust-1.x/1054845808/), Nicolas Memeint (https://docs.rs/simple-matrix/0.1.2/simple_matrix/).

Principle of operations (summary)

Usually biologists work about a sequence of interest, which we will name the *query sequence*, and they try to compare it with a batch of sequences, the *bank*, in order to select the sequences of the bank with the higher similarity scores.

The similarity scores between two sequences are computed according to the Needleman-Wunsch algorithm (https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm). This algorithm build an alignment matrix. One sequence has its letters placed horizontally on the top of the matrix, each letter on the top of a column. The second sequence has its letter placed vertically on the left of the matrix, each letter on the left of a row. One extra line is placed below the top sequence, and one extra column is placed on the right of the left sequence. Each cell of the matrix will contain the score of each individual pair of letters.

To fill the matrix, the program computes each score for each individual pair of letters according to one of three situations (definitions borrowed from Wikipedia):

- Match: The two letters at the current index are the same.
- Mismatch: The two letters at the current index are different.
- Gap: The best alignment involves one letter aligning to a gap in the other sequence.

So the algorithm needs two parameters to work: the value of the gap penalty, and the value of the mismatch penalty (or, alternatively, the value of the match bonus, which is the solution adopted for our program).

You could refer to the Wikipedia article (https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm) for further explanations and details.

To build and invoke the program:

For the developper, the command line to build the program is (from the base directory of the project):

```
cargo build
```

Then, you can invoke the program is as follows:

```
cargo run <path to the file of the query sequence>
          <path to the file of the sequences bank>
          <value of the match bonus>
          <value of the gap penalty (negative or zero)>
```

For instance, with test files from this repository:

```
cargo run Data/seq_orchid2.fasta Data/sequences_orchid.fasta 1.0 -0.5
```

To build an executable binary file proceed as follows:

```
cargo build --release
```

The executable file will be there:

```
./target/release/needleman_wunsch
```

Remember, with Rust, no runtime, so this executable is executable anywhere with your data.