How it works?

In order to search for query sequences in a big .fasta "database" file, "alenhador" first reads the database and at the same time filters sequences who are not similar enough to the query. Sequences filtered are not stored in memory anymore, and that helps reducing the memory usage. After reducing the possible sequences to align, it does the Needleman–Wunsch algorithm on each sequence concurrently.

The Heuristic Filter

The project filters database sequences who are not similar to the query, so that alignments are only made with sequences who have a chance of being a match. First, it creates a profile of every sequence in the query and the database.

For a sequence S, the profile contais:
- The number of occurences of a word W in F;
- The sequence length;
- The method "float compare(otherProfile)";

float compare()

The method "profile.compare(otherProfile)" returns a floating point number 0 <= x <= 1.0, which indicates the chance of "profile" containing "otherProfile". The result is not perfect and precision may vary depending on the words length.

```
float SeqProfile::compare(SeqProfile* other){
    vector<float> comparations;
    for(pair<string, int> entry : other->wordCounts)
    {
        if(wordCounts.find(entry.first) != wordCounts.end()){
            float value = (float)wordCounts[entry.first]/entry.second;
            comparations.push_back(value);
        }
    }

    if(comparations.size() > 0){
        float result = 0.0;
        for(float comp : comparations){
            result += comp;
        }
        return result/other->length;
    }else{
        return 0.0;
    }
}
```

The Alignment algorithm

The project uses Needleman–Wunsch, an O(n^2) algorithm for sequence alignment that uses dynamic programming. The result is an optimal alignment. The alignment of each sequence is made concurrently. An absolute score for the alignment is returned along with the alignment itself and the relative score (score/len(subjectSequence)) is displayed too.

Overall process

```
for each querySeq in queryFasta:
        filteredSeqs = dbFasta.filterSeqsAccordingTo(querySeq)
        alignments = list()
        for each filteredSeq in filteredSeqs:
                alignment = NeedlemanWunsch(querySeq, filteredSeq)
                alignment.start()
                alignments.add(alignment)
        waitForAllAreFinished(alignments)
        printResults(alignments)
```

filterSeqsAccordingTo(querySeq)

To maximize the efficiency of this part of the process, it is divided in three parts (each separated and concurrent with the others): reading fasta database, profiling sequences, sorting the most similar.

```
numberOfProfilingThreads = max(std::thread::hardware_concurrency() - 2, 2)
detach [numberOfProfilingThreads] profiling threads
detach sorting thread
startReadingDB
while(sorting or profiling):
        wait
return filteredSeqs
```

The fasta reading thread (the main thread) reads the file and puts the readen sequences in a profiling queue. The profiling threads pop a sequence from the profiling queue, create of profile of it and the put it in a sorting queue. The sorting thread pops a profiled sequence from the sorting queue and tries to put it in the filteredSeqs list, which has a limited size, and sequences who cannot fit are discarted from the memory.

(For more detailed information, please read the README.md file at the repository)
(You can find example result here)