Functional Programming War and Peace Project

Group G

Biegler Simon

Comberlato Bampi Julio A.

FH University of Applied Sciences TECHNIKUM WIEN

Technology

- Language: F#
  - F# was chosen thanks to the integration of various principles of functional programming:
    - Variables are immutable by default (not constant variables have to be declared
    - Ease of usage of partial application
    - Functions are considered as values (usage of functions as function arguments)
    - Pipeline operator „|>“ allowing to easily and understandably chain function calls
    - Multitude of different collection types available
      - https://fsharp.github.io/fsharp-core-docs/reference/fsharp-collections.html
    - Efficiency of array operations and implementation of multiple helpful functions
      - https://fsharp.github.io/fsharp-core-docs/reference/fsharp-collections-listmodule.html
    - Further documentation: https://learn.microsoft.com/en-us/dotnet/fsharp/ https://fsharp.org/
  - .NET SDK: it was chosen to use the last .NET version available on Linux: .NET 8.0 https://dotnet.microsoft.com/en-us/download/dotnet/8.0

Techniques

- Sequences
  - Used for lazy loading of large ordered collections. Sequences are used in the developed program to efficiently parse and store the book’s content.
- List.fold
  - The function „fold“ is equivalent to the C++ „accumulate“ function. For parallelization F# provides a parallel implementation of fold for arrays.
- Lists addition + List.rev
  - The addition of an element at the beginning of a list has a constant time complexity of O(1), while insertion at the end of a list of length n has a time complexity of O(n). Therefore for the addition of multiple elements in a list the chosen approach was to insert elements at the beginning and then to revert the list (time complexity of O(n))
- Sets
  - Sets are used for containing various unique values. This allows for efficient search operations via the „contains“ function, being more efficient then the search in a traditional array or list
- List.indexed
  - The function List.indexed can be used to create a List of tuples from a non empty list, the tuples being: (index, value)
- List.pairwise
  - The function List.pairwise can be used to generate a List of tuples from a non empty list, its elements are: (eln, eln+1). This function was used to efficiently calculate the distance between the matches in a chapter

Approach

- The program’s workflow can be divided in 4 phases:

  1) „Reading“ the book’s content, as well as the lists of war and peace terms.
     1) Parsing of book’s content as a sequence, parsing of terms as a set
  2) Transformation of the book context in an appropriate structure for computation
     1) The book content is stored in a 2D list, every entry of the outer list represents a chapter, every entry of the inner list represents a word of the chapter.
  3) Parsing of chapters’ contents to find matches with the given keywords
     1) For the determination of a chapter’s topic the indexes of the words contained in the specified keywords set are saved to be used to calculate the density.
  4) Calculation of terms density to evaluate the chapter’s topic
     1) For every chapter the density of the war and peace terms are calculated and compared to assess its content.
  5) Output of results in a .txt file for simple comparison with reference

Topic evaluation

- Density
  - The determination of a chapter’s topic is based upon the density of the keywords related to the different topics. The higher a topic’s density, the more likely it is that the analysed chapter is related to the given topic.
    - Multiple approaches were tried:
      - Density based upon average relative distance
      - Density based upon average distance
      - Density as number of matches for each chapter
- Sanitisation
  - Before the parsing of the book’s content trimming and sanitation are executed:
    - Content’s trimming:
      - The provided .txt file was trimmed at the beginning and ad the end in order to avoid processing the books metadata. This was achieved by the usage of Seq.takeWhile and Seq.skipWhile
    - Content’s sanitation, following sanitisation’s techniques were implemented:
      - Removal of non alphanumeric characters from the input
      - Consideration of punctuation symbols as words

Testing

- Normal unit testing
  - Specific functionalities of the code (like the splitting of a string list in chapters) were too complicated to test with property based testing: it is possible to write an appropriate generator, however not method for the verification of a successful chapter splitting without hardcoded values was found. Therefore it was decided to use „normal“ unit tests for specific functions. E.g.:

- Property testing
  - While not every functionality was tested with property based testing, generators were created to implement property testing whenever possible. The generators were implemented with the library „FsCheck“ (https://fscheck.github.io/FsCheck/), the implemented generators were used to create sequences of words, in order to test the splitting of sentences in words, as well as the counting of matches with specific keywords.

Results

The success of a given approach was evaluated based upon the number of chapters evaluated differently in comparison to the provided reference.

Following results were achieved:

Density function	Sanitation Y/N	Differences
Density based upon avg relative distance	Y	54
Density based upon avg relative distance	N	24
Density as number of matched keyword	Y	39
Density as number of matched keyword	N	9