BookAnalyser

Ein Projekt in Funktionaler Programmierung / Felix Kainz

FH University of Applied Sciences
TECHNIKUM WIEN

Project Structure

- Includes.cpp
  - Hier befinden sich alle inkludierten Header und verwendeten Entities
- MapReduce.cpp:
  - Hier befindet sich die main
- PureFunctions.cpp:
  - Hier befinden sich alle Funktionen die vollständig pure sind
- ImpureFunctions.cpp
  - Hier befinden sich alle Funktionen die nicht pure sind

Ablauf:

- 1. Einlesen des Buches

```
ifstream inputFile(filename);
vector<vector<string>> chapters;
chapters.push_back(vector<string>());

while(std::getline(inputFile, line) && !bookIsOver){
    if(line.length() == 0){
        continue;
    }

    if( bookHasStarted ){
        if(line.compare(0, 7, "CHAPTER") == 0){
            currentchapter++;
            chapters.push_back(vector<string>());
        } else {
            chapters[currentchapter].push_back(line);
        }
    }

}

inputFile.close();

return chapters;
```

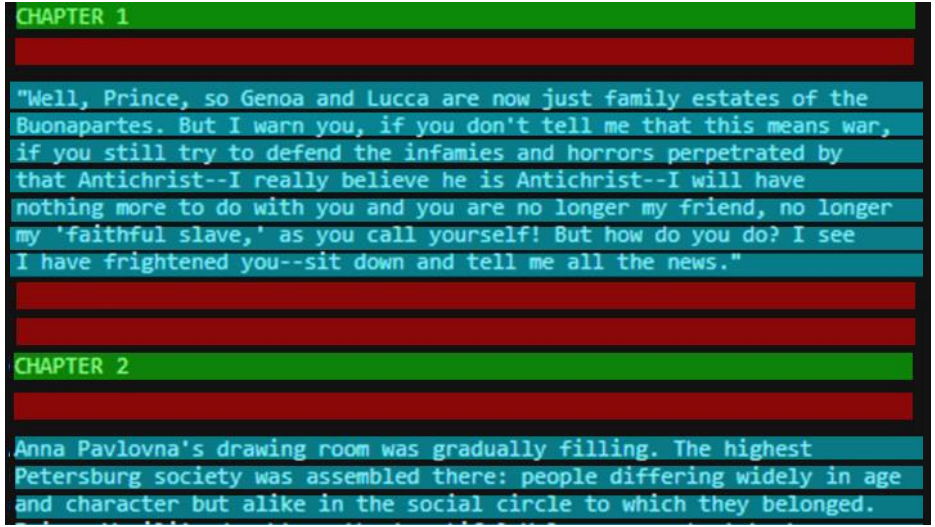


Image: InpureFunctioncs.cpp - ReadBookFromFile

Ablauf:

- 2. Einlesen der Wortlisten

```
afraid
anguish
armed
barbwire
battle
```

- 3. Mappen der Wortlisten

```
["afraid", 1]
["anguish", 1]
["armed", 1]
["barbwire", 1]
["battle", 1]
```

Konvertierung zu einer map um:

- Effiziente Suche nach Keys
- Wenn Key >= 1 dann vorhanden
- Wenn Key = 0 dann nicht vorhanden

```
auto MapVector = [](const vector<string> Vector)
    map<string, int> Mapping;
    for(string Word : Vector){
        Mapping[Word]++;
    }
    return Mapping;
};
```

Image: PureFunctioncs.cpp - MapVector

4. Evaluation

- 1. Iteriere über jedes Kapitel
- 2. Mache einen neuen Thread
  - Teile die Kapitel in Wörter
  - Evaluiere das Kapitel
  - Speicher dein Ergebnis
- 3. Warte bis alle Threads fertig sind
- 4. Sortiere alle Ergebnisse

```
o EvaluateAllChapters = [](const Book& Book, const map<string, int>& PeaceMapping, const map<string, int>& WarMapping )
mutex mtx;
vector<ChapterEvaluation> EvaluatedChapters;

// remove the prelude before the first chapter starts
auto BookView = Book | std::views::all | std::views::drop(1);
int i = 0;

// vector of all threads
vector<thread> activethreads = {};

ranges::for_each(BookView.begin(), BookView.end(), [&](const vector<string>& chapter){
    int Threadnumber = ++i;
    // add a thread for each chapter
    activethreads.emplace_back([&EvaluatedChapters, &mtx, &chapter, &PeaceMapping, &WarMapping, Threadnumber]() {
        // Tokenize them
        Chapter TokenizedChapter = SplitChapterIntoWords(chapter);

        // Evaluate
        ChapterEvaluation result = EvaluateChapter(TokenizedChapter, PeaceMapping, WarMapping);

        // add index and safe the result
        result.chapterIndex = Threadnumber;
        lock_guard<mutex> lock(mtx);
        EvaluatedChapters.emplace_back(result);
    });
});

// wait for all to finish
for (auto& thread : activethreads) {
    thread.join();
}

// sort it
vector<ChapterEvaluation> EvaluatedChaptersSorted = sortEvaluations(EvaluatedChapters);

return EvaluatedChaptersSorted;
```

Image: PureFunctioncs.cpp - EvaluateChapters

Teile die Kapitel in Wörter

"Well, Prince, so Genoa and Lucca are now just family estates of the

- Schaue dir jeden Character an
- Wenn der Charakter ein Buchstabe ist, dann hänge ihn an das letzte Wort an
- Wenn der Charakter ein Leerzeichen ist, dann speichere dir das letzte Wort und starte ein neues Wort

```
auto SplitStringIntoWords = [](string WordLine) -> Line {
    Line Words;

    string currentWord = string();

    for_each(WordLine.begin(), WordLine.end(), [&](char letter){
        if(letter == '\'' || IsALetter(letter)){
            currentWord.push_back(letter);
        }
        else if(letter == ' '){
            Words.push_back(currentWord);
            currentWord = string();
        }
    });

    Words.push_back(currentWord);
```

Image: PureFunctioncs.cpp - SplitStringIntoWords

"Well, Prince, so Genoa and Lucca are now just family estates of the

Evaluiere das Kapitel

- Besteht aus zwei Teilen:

1. Filtere alle Wörter

- Gehe durch jedes Wort

- Schaue ob es in einer Map ist

- Wenn ja, dann speichere es in einer Liste

Optional:
Entweder Peace Term oder War Term oder keines von beidem

```cpp
vector<int> FoundPeaceTerms = { };
vector<int> FoundWarTerms = { };
int iterator = 1;

auto evaluateWord = [&](Word word){
    optional<int> Result = Filter(word, PeaceTerms, WarTerms);
    if(Result.has_value()){
        if(*Result){
            FoundWarTerms.push_back(iterator);
        } else {
            FoundPeaceTerms.push_back(iterator);
        }
    }
    iterator++;
};

auto evaluateLine = [&](Line line){
    for_each(line.begin(), line.end(), evaluateWord);
};

for_each(Chapter.begin(), Chapter.end(), evaluateLine);

return make_pair(FoundPeaceTerms, FoundWarTerms);
```

Image: PureFunctioncs.cpp - FilterChapter

```cpp
auto Filter = [](const string& Word, const map<string, int>& PeaceTerms, const map<string, int>& WarTerms) -> optional<int> {
    if(PeaceTerms.find(Word) != PeaceTerms.end()) return 0;
    if(WarTerms.find(Word) != WarTerms.end()) return 1;
    return nullopt;
```

Image: PureFunctioncs.cpp - Filter

Evaluiere das Kapitel

- 2. Berechne die Häufigkeit

Einfaches zählen der Elemente von beiden Listen

Das Thema mit mehr Elemente = das Thema des Kapitels

```
auto PairVectors = FilterChapter(Chapter, PeaceMapping, WarMapping);

int PeaceDistance = avrgDistance(PairVectors.first);
int WarDistance = avrgDistance(PairVectors.second);
bool isWarChapter = WarDistance > PeaceDistance;

ChapterEvaluation Result{
    0,
    Chapter,
    PairVectors.first,
    PairVectors.second,
    isWarChapter,
};
return Result;
```

Image: PureFunctioncs.cpp - EvaluateChapter

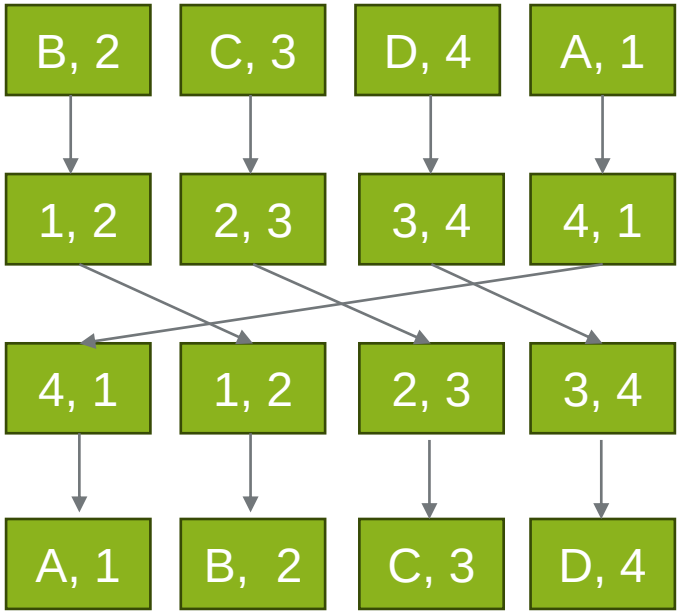
4. Evaluation

- 1. Iteriere über jedes Kapitel
- 2. Mache einen neuen Thread
  - Teile Kapitel in Wörter
  - Evaluiere das Kapitel
  - Speicher dein Ergebnis
- 3. Warte bis alle Threads fertig sind
- 4. Sortiere alle Ergebnisse

```
o EvaluateAllChapters = [](const Book& Book, const map<string, int>& PeaceMapping, const map<string, int>& WarMapping )
mutex mtx;
vector<ChapterEvaluation> EvaluatedChapters;

// remove the prelude before the first chapter starts
auto BookView = Book | std::views::all | std::views::drop(1);
int i = 0;

// vector of all threads
vector<thread> activethreads = {};

ranges::for_each(BookView.begin(), BookView.end(), [&](const vector<string>& chapter){
    int Threadnumber = ++i;
    // add a thread for each chapter
    activethreads.emplace_back([&EvaluatedChapters, &mtx, &chapter, &PeaceMapping, &WarMapping, Threadnumber]() {
        // Tokenize them
        Chapter TokenizedChapter = SplitChapterIntoWords(chapter);

        // Evaluate
        ChapterEvaluation result = EvaluateChapter(TokenizedChapter, PeaceMapping, WarMapping);

        // add index and safe the result
        result.chapterIndex = Threadnumber;
        lock_guard<mutex> lock(mtx);
        EvaluatedChapters.emplace_back(result);
    });
});

// wait for all to finish
for (auto& thread : activethreads) {
    thread.join();
}

// sort it
vector<ChapterEvaluation> EvaluatedChaptersSorted = sortEvaluations(EvaluatedChapters);

return EvaluatedChaptersSorted;
```

Image: PureFunctioncs.cpp - EvaluateChapters

Sortieren alle Ergebnisse:



Std::sort sortiert die die Evaluations basierend auf Ihrer Kapitelnummer, damit sie wieder in der richtigen Reihenfolge sind.

Problem:

Kapitelevaluieren sind const und können nur kopiert werden und nicht verschoben

Lösung:

Wir sortieren Indexe, und erstellen erst ganz am Ende eine neue Sammlung

Ergebnisse

```
1   Chapter 1: peace-related
2   Chapter 2: peace-related
3   Chapter 3: peace-related
4   Chapter 4: peace-related
5   Chapter 5: peace-related
6   Chapter 6: peace-related
7   Chapter 7: war-related
8   Chapter 8: peace-related
9   Chapter 9: peace-related
10  Chapter 10: war-related
11  Chapter 11: peace-related
12  Chapter 12: peace-related
13  Chapter 13: war-related
```

Image: ResultFolder/Results.txt

```
1   Program took 0.251952 seconds
```

Image: ResultFolder/TimeResult.txt

Danke für Ihre Aufmerksamkeit!

Github:
https://github.com/Felix-beep/MapReduce