PROJEKTDOKUMENTATION

AUSBILDUNG ZUM FACHINFORMATIKER F.
ANWENDUNGSENTWICKLUNG

Abschlussprüfung Winter 2017
IHK Berlin

Auszubildender:
Nils Diefenbach
Ident-Nummer:
3590244

Ausbilder:
Sebastian Freundt
GA FINANCIAL SOLUTIONS GMBH

10. September 2017

/home/freundt/business/GA_Logo.jpeg

Inhaltsverzeichnis

/home/freundt/business/GA_Logo.jpeg

1 Abkürzungsverzeichnis

- **GLEIS**: Global Legal Entity Identifier System
- **CLI**: Command Line Interface
- **CSV**: Comma-separated Values Format
- **DLD**: Damerau-Levenshtein Distanz
- **IB**: Interactive Brokers, Daten Anbieter und FI Broker
- **TDD**: Test-Driven Development
- **ERM**: Entity-Relationship-Model
- **UML**: Unified-Modelling-Language
- **HPC**: High Performance Computing

2 Einleitung

2.1 Projektumfeld

Die GA Financial Solutions GmbH ist ein fünfköpfiger Finanzdienstleister im Herzen Berlins, welcher sich auf computer-gestütztes Handeln von Futures, Aktien und anderen Derivaten spezialisiert hat. Direkte Kunden gibt es zur Zeit nicht und die Firma finanziert sich durch Investoren und den Gewinn ihrer Strategien. Die Rolle des Auftraggebers und Kunden übernimmt im Rahmen des Projektes Sebastian Freundt, stellvertretend für die Abteilung Datenverarbeitung.

2.2 Projektziel

Ziel ist es, ein Kommandozeilenprogramm zu schreiben, welches mehrere Derivatsymbole von verschiedenen Anbietern anhand einer Metrik vergleicht und übereinstimmende Strings ausgibt.

2.3 Projektbegründung

Die GA Financial Solutions handelt Wertpapiere aller Art auf eigene Rechnung. Dabei ist es marktüblich, dass Referenzdaten, Preisdaten und Ausführungen von unterschiedlichen Dienstleistern bezogen werden. So stehen ca. 223 Mio. handelbaren Wertpapieren rund 500.000 mögliche Herausgeber an ca. 1.200 Börsen weltweit gegenüber. Jede einzelne Börse bzw. jeder einzelne Makler im Markt unterstützt jedoch nur einen Bruchteil der Papiere.

In Ermangelung eines weltweiten Standards, der jedem Marktteilnehmer in jeder Handelsphase gerecht wird, werden unterschiedliche Symbologien benutzt, deren kleinster gemeinsamer Nenner stets die Klartextbezeichnung des begebenen Papiers zusammen mit der Klartextbezeichnung des Herausgebers ist. Erschwert wird die Problematik überdies noch durch unterschiedliche Transliterationsverfahren, so führt z.B. die NASDAQ die Firma „Panasonic Corp.“, die GLEIS-Datenbank[1] jedoch unter "パナソニック株式会社", was transskribiert wiederum „Panasonikku Kabushikigaisha“ ergibt.

Bei der GA Financial Solutions werden Zuordnungen zwischen Symbologien aktuell ad-hoc mit Heuristiken und Handarbeit bewerkstelligt. Der Zeitaufwand ist entsprechend hoch, die Fehlerrate ebenfalls. Es verbietet sich geradezu, die bisherigen Heuristiken systematisch zur Zuordnung aller Herausgeber aller Wertpapiere zu benutzen.

Das zu entwickelnde Tool soll vor Allem den Zeitaufwand und die benötigte menschliche Komponente, und somit Fehlerquellen, reduzieren, so dass diese Ressourcen entsprechend anderen Aufgaben zu geteilt werden können.

[1] Global Legal Entity Identifier System

2.4 Projektschnittstellen

Das Programm wird als Kommandozeilenprogramm angeboten, welches Daten per Pipe oder unter Angabe eines Dateipfads annimmt und diese im .csv Format über STDOUT ausgibt. Dieser Ansatz der Resultatausgabe wird explizit gewünscht, da sich das Programm auf diese Art mühelos in die bisherige Toolchain der Firma einbinden lässt. Das Programm wird vor Allem in der Abteilung für Datenverarbeitung zum Einsatz kommen.

Als technische Schnittstelle ist insbesondere das Datensammelsystem zu nennen, welches täglich unsere Datenbanken mit neuen Datensätzen füttert. Die Ergebnisse dieses Systems liegen im CSV-Format auf unseren Servern, bevor sie am Ende des Tages komprimiert werden.Eine direkte Anbindung des zu entwickelnden Projekts an das Datensammelsystem findet nicht statt. Wichtig ist allerdings, dass das Programm .CSV-Dateien unterstützt, welche vom System bereitgestellt werden.

Mittel und Genehmigung des Projekts stellt die Abteilung Datenverarbeitung, vertreten durch Sebastian Freundt.

2.5 Projektabgrenzung

Während die Hauptziele des Projekts das Vergleichen mehrerer hunderttausend Symbole und die effiziente Gestaltung dieser Vergleiche ist, sind folgende Themen explizit nicht Teil der Aufgabe:

- Datenintegrität - Die Überprüfung des Datenformats ist nicht Teil unseres Aufgabenfeldes.
- Datenverifizierung - Die Prüfung ob **eingehende** Daten korrekt sind, fällt nicht in das Aufgabenfeld.
- Datenbereitstellung - Die Bereitstellung der verarbeiteten Daten (z.B. als Datenbank o.Ä.) ist ebenfalls nicht Teil des Projekts.

3 Analysephase

3.1 Ist-Analyse

Die GA Financial Solutions GmbH holt sich täglich aktuelle Symbole von sechs verschiedenen Anbietern ab: Interactive Brokers, Google, Yahoo, Morningstar, Bloomberg und GLEIS. Täglich ändern sich diese Symbole um einen gewissen Prozentsatz und es kommen im Durchschnitt etwa 1,67 Millionen Instrumente hinzu.

Zur Zeit werden diese Symbole ad hoc mittels Heuristiken und Handarbeit verglichen - dabei wird meist nur eine kleine Teilmenge benutzt; gefundene Ergebnisse werden nach Abschluss eines Projekts wieder verworfen, da es sich meist um spezialisierte Werte handelt.

Somit besteht zur Zeit keine firmenweite Datenbank, welche die Symbole bereits verglichen und sortiert bereitstellt.

Es besteht ein theorethischer Lösungsansatz auf Basis der Damerau-Levenshtein Distanz[2](kurz: DLD). Dieser sieht vor, eine M x M Matrix zu generieren, wobei M sämtliche Symbole aller Anbieter darstellt. Dies resultiert in einem circa 23 PetaBytes großem Datenobjekt im Arbeitsspeicher unserer Systeme, und führt erwartungsgemäß zu Speicherüberläufen. In Rechenaufrufen ausgedrückt bedeutet dies, dass etwa
1.038.901.988.746.226[3]DLD Aufrufe benötigt werden um die Daten miteinander zu vergleichen und die Datenbank zu erstellen. Das DLD-Matrix-Verfahren schafft hierbei etwa 1.340.000 Aufrufe pro Sekunde (CPU-Zeit). Daraus ergibt sich eine Initiallaufzeit von etwa 6,15 Jahren auf einem 4-Kern-Rechner bei voller Nutzlast (dies exkludiert die täglich hinzukommenden Daten während der Berechnungszeit). Ist diese Datenbank erstellt, werden durch die neu hinzukommenden Daten circa 16 Milliarden DLD-Aurfufe täglich benötigt um die Datenbank zu aktualisieren. Pro Tag entspricht dies weiteren 33 Minuten Rechenzeit auf einem 4-Kerner.

Gewünscht ist eine performantere und Ressourcen-schonendere Implementierung des obigen Prozesses, sodass die Daten auf einem firmeninternen 4-Kerner zeitnah ausgerechnet werden können. Hierbei ist es nicht unbedingt erforderlich die, Damerau-Levenshtein-Distanz zu verwenden. Desweiteren soll der Zeitaufwand der ad-hoc Vergleichue der Ergebnisse drastisch reduziert werden.

3.2 Wirtschaftlichkeitsanalyse

3.2.1 Make-or-Buy Entscheidung

Ein Tool, welches das oben beschriebene Problem löst gibt es zur Zeit nur mit dem ressourcenintensiven DLD-Matrix-Verfahren. Ein Projekt, welches ein ähnliches Pro-

[3]Die Damerau-Levenshtein Distanz[**?**] beschreibt die minimale Anzahl an Operationen welche benötigt wird um einen String A in einen zu vergleichenden String B zu verwandeln

[3]Eine Billiarde achtunddreißig Billionen neunhundertein Milliarden neunhundertachtundachtzig Millionen siebenhundertsechsundvierzigtausendzweihundertsechsundzwanzig

blem löst, ist das Open-source Projekt fuzzy-join[4]. Dieses Tool vergleicht Daten jedoch nur in eine Richtung (One-To-Many), während unser Use-Case ein bidirektionales Verfahren benötigt (Many-To-Many).

Somit stellte sich die Frage nach einer optimalen Vorgehensweise zur Umsetzung des Projekts. Hierzu wurde vom Autor eine Nutzwertanalyse2 erstellt, um diversen Lösungswege zu evaluieren.
Diese wurden anhand der folgenden Kriterien bewertet:

- **Projektkosten** - Wie hoch sind diePersonal-, Server- und/oder Softwarekosten der jeweiligen Lösung?
- **Nachhaltigkeit** - Gibt es externe Abhängigkeiten? Wenn ja, wie viele und besteht ein Risiko des Ausfalls dieser Abhängigkeiten? Wie gut lässt sich das Programm in unseren Arbeitsprozess eingliedern?
- **Flexibilität** - Wie leicht ist das Programm erweiterbar? Besteht die Möglichkeit in Zukunft weitere Parameter anzugeben um die Ergebnisse anzupassen? Wenn ja, wie sehr beeinträchtigt diese Erweiterung die Performanz des Programms?
- **Unterhaltungskosten** - Neben den Initialen Projektkosten, wie hoch sind die laufenden Kosten für die Lösung? Fallen Mietkosten für Server an? Wie hoch ist der Überwachungsaufwand der benötigt wird um die Funktionsfähigkeit der Lösung zu überprüfen?

3.2.2 Projektkosten

Als nächstes wurden die Projektkosten berechnet. Hierfür wurden pauschale Beträge für Mitarbeiterstundenlöhne, sowie Kosten für Räumlichkeiten, Strom und sonstige Utensilien (Papier, Drucker, Programme, Rechner etc) veranschlagt. Diese können in der Tabelle **??** eingesehen werden. Es wurden hierfür ein Stundenlohn von 5,50€ für den Autoren, sowie ein Stundenlohn von 35€ für jeden weiteren Mitarbeiter angesetzt. Nutzungskosten der Räumlichkeiten, Rechner, Software und weiterer Utensilien wurden mit 20€ pro Stunde kalkuliert.

3.2.3 Amortisationsdauer

3.3 Anwendungsfälle

Um eine Übersicht über die Anwendungsfälle zu bekommen hat der Autor sich zunächst mit der Abteilung für Datenverarbeitung zusammengesetzt und evaluiert. Aus diesem Meeting erstellte der Autor ein Use-Case-Diagramm, um diese Fälle zu visualisieren und zu verdeutlichen. Das erstellte Diagramm wurde im Anhang8.1.3 beigefügt.

[4]https://github.com/dgrtwo/fuzzyjoin

3.4 Lastenheft / Fachkonzept

Ein Lastenheft wurde von der Abteilung Datenverarbeitung, vertreten durch Sebastian Freundt, erstellt und dem Autor vorgelegt. Ein Auszug dieses Dokumentes befindet sich im Anhang auf Seite X.

4 Entwurfsphase

4.1 Zielplattform

Wie aus dem vorherigen Abschnitt**??** entnommen werden kann, soll das Projekt als Kommandozeilenprogramm implementiert werden. Anhand der Anforderungen des Lastenhefts, welches Speicherperformanz sowie Geschwindigkeit fordert, fiel die Entscheidung der Programmiersprache recht schnell auf die Hochsprache C.Diese erlaubt effiziente Berechnungen bietet eine ideale Schnittstelle um effizient im Speicher agieren zu können. Zur Überprüfung dieser Einschätzung wurden diverse weitere Sprachen anhand einer Nutzwertanalyse mit der Sprache C verglichen. Die wichtigste Rolle spielten hierbei folgende Kriterien:

- **Performanz** - Geschwindigkeit der Sprache in der HPC Domäne
- **Speichereffizienz** - Der Fußabdruck der Sprache auf der Festplatte, sowie der Objekte im Speicher, soll möglichst gering sein
- **Dokumentation** - Bibliotheken und Module sollen gut und einsehbar dokumentiert sein.
- **Systemische Vorraussetzungen** - Anzahl und Aufwand der Vorraussetzungen, um die Sprache zu nutzen

Die Ergebnisse dieses Vergleichs wurden in der Nutzwertanalyse **??** präsentiert. Sie ist dem Anhang dieses Dokumentes beigefügt.

Letztendlich wurde an der urprünglichen Entscheidung, die Sprache C zu nutzen, festgehalten.

4.2 Algorithmusdesign

In Zusammenarbeit mit der Abteilung für Datenverarbeitung hat der Autor einen simplen Pseudocode für die grundlegende Logik des Algorithmus' entwickelt, welcher als Ausgangspunkt für die weitere Entwicklung genutzt wurde. Hierbei wurden sogenannte Q-Gramme verwendet - diese verstehen sich als Teilstring eines Strings der Länge Q. Dieser Code wurde in Form eines Aktivitätsdiagramm4 visualisiert. Der zugrunde liegende Pseudocode des Diagramms ist ebenfalls im Anhang8 einsehbar.

4.3 Architekturdesign

Das Projekt besteht aus zwei Header-Dateien und einer main.c. Die Header-Dateien beinhalten zum einen die algorithmusspezifischen Funktionen zum Vergleich von Strings, zum anderen eine open-source Hashmap-Implementierung. Ein Screenshot der Ordnerstruktur kann im Anhang8.1.6 eingesehen werden.

Auf Programmebene wurde die Logik in zwei Abschnitte unterteilt:

- **Vorbereitung**
  Hier werden Daten aus der ersten Datei geladen und benötigte Datenstrukturen im Speicher angefordert. Variablen werden initialisiert und der Q-Gramm-Pool erstellt.

- **Ausführung**
  Das zweite Datenset wird über einen File-Pointer referenziert und die Strings werden Zeile für Zeile ausgelesen. Die Q-Gramme jeder Zeile werden dann mit den Q-Grammen aus der Phase "Vorbereitung"verglichen. Erfüllt die Zeile die Anforderungen für Stringübereinstimmungen, wird sie über STDOUT ausgegeben.

Hieraus ergaben sich acht Subkomponenten, welche implementiert werden mussten:

- **Vorbereitung**:
  - **Variable S**: Die Menge aller Strings aus der ersten Liste bzw. Datei
  - **Variable Q**: Die Menge aller QGramme, welche sich aus den jeweils einzelnen Strings in S generieren lässt

- **Ausführung**
  - **Variable I**: Die Menge, welche die gleiche Größe wie S hat und die Anzahl der Qgramme zählt, welche für den jeweiligen Index eines Strings in S passen.
  - **Variable f**: Eine Referenz zur zweiten Liste, über die wir zu vergleichende Strings holen können
  - **Variable T**: Die Menge der Qgramme in der aktuell ausgelesenen Zeile
  - **Variable M**: Eine Menge der Indizes von Strings, welche auf die QGramme **T** am besten passen

Mit Ausnahme von Q und M können alle Datenstrukturen mit den geläufigen C Datentypen ausgedrückt werden.

Für die Strukturen Q und M mussten jedoch neue Datenstrukturen erstellt werden. Für Q bot sich hier eine Hashmap an, die zwar einen höheren Speicheraufwand hat, welcher jedoch durch die Beschränkung des Alphabets und der Länge der Q-Gramme abschätzbar ist[5]. Der daraus folgende O(1) Suchaufwand zum Vergleich von Q-Grammen ist ein unschätzbarer Vorteil in der späteren Ausführungsphase. Auch hier wurde wiederum eine Nutzwertanalyse4 durchgeführt, um sicherzustellen dass die persönlichen Annahmen des Autors mit den technischen Gegebenheiten übereinstimmen. Hierzu wurden verschiedene Datenstrukturen nach folgenden Kriterien bewertet:

1. **Schnelligkeit Such-Befehl** - Wie hoch ist die Komplexität des Such-Befehls der Datenstruktur?

2. **Speicheraufwand** - Wie hoch ist der maximale Speicheraufwand des Datentyps wenn die Menge aller möglichen Q-Gramme eingefügt wird?

3. **Schnelligkeit Einfügen-Befehl** - Wie hoch ist die Komplexität des Einfügen-Befehls der Datenstruktur?

Für M wird ein dynamisch wachsendes Array benötigt, da die Anzahl der passenden Q-Gramme je nach String variiert. Die Anzahl der möglichen Zeichen pro Q-Gramm von unserem verwendeten Zeichensatz ist durch die Kodierung (7-Bit ASCII) begrenzt. Um Einschätzen zu können welche Startgröße des Arrays sinnvoll ist um die Speicherzuweisungsaufrufe möglichst gering zu halten, wurde die Verteilung der Q-Gramme innerhalb des derzeitigen Datensatzes berechnet. Dies wurde anhand eines Graphen**??** dargestellt, welcher die Anzahl Strings auf X-Achse, der Anzahl Q-Gramme auf der Y-Achse als Quantil gegenüber.

Anhand dieser Analyse konnte eine optimale Startgröße festlegt werden, sodass in etwa der Hälfte der Fälle erwartet werden kann, dass das Array nicht vergrößert werden muss, und somit die Speicherzuweisungen minimiert werden.

Im Einklang mit der test-getriebenen Entwicklung wurde von Anfang an das Hauptaugenmerk auf die möglichst granulare Entwicklung des Codes gelegt. So wurden die Schritte des Algorithmus' in jeweils mindestens einen Test und eine dazugehörige Funktion unterteilt **??**. Auch wurden für jeden selbst erstellten Datentyp und die dazugehörigen Funktionen Tests geschrieben.

4.4 Entwurf der Benutzeroberfläche

Der GNU Codingstandard[**?**], sowie die POSIX Utility Guidelines[**?**] spezifizieren standardisierte Parameternamen und Funktionen eines Programms, sowie deren erwartete Funktion. Da es sich bei der Zielgruppe um erfahrene Linuxnutzer handelt, wurde das Kommandozeilenprogramm nach diesen Standards entwickelt. Ein Auszug dieser Vorgaben befindet sich im Anhang auf Seite X.

Da die bevorzugte Oberfläche der Endnutzer die Kommandozeile ist, wird keine grafische Oberfläche entwickelt.

4.5 Datenmodell

Aufgrund der Natur der Daten ist das Datenmodell für das Projekt trivial. Es besteht aus einer einspaltigen Tabelle, welche in Relation zu sich selbst steht. Aus diesen Gründen wird auf eine weitere Ausführung des Modells verzichtet - die UML-Grafik des Modells kann jedoch im Anhang7 eingesehen werden.

Es ist zu klarzustellen, dass die GA Financial Solution ungern Datenbanksysteme wie MySQL, MongoDB u.Ä. nutzt. Stattdessen werden komprimierte CSV-Dateien

[5]Maximale Anzahl an Schlüsselworten einer Hashmap im Kontext dieses Projekts = Länge des Alphabets hoch Q-Grammlänge

verwendet, welche meist nur aus drei Zeilen (Zeitstempel, Typ und Wert) bestehen. Dies ergibt sich aus der negativen Erfahrung mit Datenbankmanagementsystemen und der Vereinfachung des Workflows durch die Nutzung von CSV-Daten in Kombination mit der hauseigenen CLI-Toolchain.

4.6 Geschäftslogik

Zur Veranschaulichung der bisherigen Geschäftslogik, wurde eine Aktivitätendiagramm8.1.4 erstellt.Wie man dort erkennen kann, ist die Abholung der Daten von Anbietern automatisiert, jedoch nicht deren Verarbeitung. Diese geschieht bei Bedarf und erfordert, unter Anderem, die Vorsortierung der Daten in handhabbare Datensätze, welche einfacher und in angemessener Zeit von unseren 4-Kern-Maschinen berechnet werden können. Zudem kommt eine erforderliche manuelle Überprüfung der Ergebnisse. Die daraus entstehenden Daten werden schließlich für das derzeitig entwickelte Projekt genutzt und schlussendlich wieder verworfen, da die Datensätze hoch parametrisiert sind, und sich nicht zur generellen Nutzung innerhalb der Firma eignen.

4.7 Pflichtenheft / Datenverarbeitungskonzept

Abschließend zur Entwurfsphase wurde ein Pflichtenheft erstellt und dieses als Roter Faden für das Projekt der Abteilung für Datenverarbeitung vorgelegt. Wie im Projektmanagement üblich, baut dieses auf dem Lastenheft auf. Ein Auszug des Pflichtenhefts8.3.2 befindet sich im Anhang.

5 Implementierungsphase

5.1 Entwicklungsvorbereitung

Zunächst mussten einige organisatorische Bedingungen erfüllt werden, bevor die eigentliche Entwicklungsphase offiziell beginnen konnte. So wurden das Kanban Board mit den Aufgabenpaketen aus dem Pflichtenheft populiert. Hierzu wurden Tests und deren dazugehörigen Funktionen separat betrachtet und als Arbeitspaket beschrieben.

5.2 Implementierung der Datenstruktur

Wie bereits im vorherigen Abschnitt erwähnt, werden zwei spezielle Datenstrukturen für das Programm benötigt - Hashmaps und dynamisch wachsende Arrays.

5.2.1 Hashmap-Implementierung

Auf die eigene Implementierung einer Hashmap wurde verzichtet, stattdessen wurde eine frei verfügbare Bibliothek genutzt, welche eine Hashmap für String-Integer Paare zur Verfügung steht.

Der Code wurde weitgehend verbatim übernommen, mit der Ausnahme des Containertyps, welche die Hashmap speichert. Da ein Q-Gramm in mehreren Strings vorkommen kann, musste der Containertyp von einem Integer zu einen dynamisch wachsenden Array abgewandelt werden. Da letzteres ohnehin entwickelt werden muss, war die Komplexität dieser Anforderung gering.

5.2.2 Dynamische Array-Implementierung

C stellt nur simple Datentypen zur Verfügung, welche dem Nutzer erlauben komplexere Datenstrukturen abzubilden. Da ein dynamisch-wachsendes Array nicht in C zur Verfügung steht, musste eine Datenstruktur für dieses erstellt werden. Das Konzept hierbei ist recht einfach - sollte das Array voll sein, so wird beim nächsten Aufruf der Insert-Funktion die Länge des Array verdoppelt. Hierfür wurde eine neue Datenstruktur angelegt und drei Funktionen definiert, um mit der Datenstruktur interagieren zu können. Ein Auszug des Quellcodes dieser Datenstruktur ist im Anhang zu finden**??**.

6 Abnahmephase

Nach Abschluss der Implementierungsphase wurde das Projekt, wie zuvor mit der Abteilung für Datenverarbeitung abgesprochen, sowohl dem Abteilungsleiter im Einzelgespräch, als auch der gesamten Abteilung in Form einer Präsentation, präsentiert.

Das Hauptaugenmerk und die Vorgehensweise war in beiden Fällen jedoch unterschiedlich und wird in den folgenden Aschnitten erläutert.

6.1 Code-Flight mit der Abteilungsleitung

Um die Qualität des Codes zu gewährleisten wurde für das Projekt die Abnahme über einen Code-Flight[6] festgelegt. Diese einstündige Vorführung des Codes wurde mit dem Abteilungsleiter Sebastian Freundt im Vier-Augen-Gespräch vollzogen und gestaltete sich wie folgt:

- Demonstration des Programms über die Kommandozeile, mit Schritt-für-Schritt Erklärung der gewählten Parameter und deren Design-Entscheidung.
- Sichtung des Codes, angefangen mit der Hauptroutine.
- Erläuterung der gewählten Datenstrukturen und Begründung für die Nutzung dieser.
- Limitation des Algorithmus und des CLI-Programms.
- Weitere Schritte zur Verbesserung der Hauptprogramms.
- Mögliche Ansätze zur Verbesserung des Algorithmus.

6.2 Präsentation des Programms

Nachdem Code-Flight mit dem Abteilungsleiter, wurde ebenfalls eine Präsentation für das gesamte Team der Abteilung Datenverarbeitung gehalten. Diese beschränkte sich im Inhalt jedoch auf die Grundprinzipien des Algorithmus, sowie die Funktion und Nutzung des CLI-Programms. Unter anderem wurden verfügbare Parameter und maximal zulässige Dateigrößen erläutert.

6.3 Zwischenstand

[6] zu deutsch P̈rogramm-Flug"

7 Dokumentation

Die Dokumentation ist aufgeteilt in zwei Bestandteile: Die Nutzerdokumentation in Form einer CLI-Dokumentation, sowie die Entwicklerdokumentation in Form von Dokumentationsblöcken im Quellcode. Die gesamte Dokumentation wurde auf Englisch, der Sprache des Unternehmens, verfasst.

7.1 Nutzerdokumentation

Da der Funktionsumfang des Programms recht klein ist, wird auf ein handelsübliches Dokument verzichtet. Die Dokumentation findet ausschließlich über die Kommandozeile mit dem Aufruf des Programms unter Zusatz des Hilfe Parameters ("help"bzw. "h") statt. Dieser Aufrauf erzeugt eine kurze Beschreibung des Programms und dessen Parameter.

7.2 Entwicklerdokumentation

Die Entwicklerdokumentation des Codes findet über einen Dokumentationsblock oberhalb der Signatur jeder Funktion statt. Dieser Block beinhaltet eine kurze Beschreibung der funktionsweise der Routine, sowie die Namen und erwarteten Datentypen der Funktionsparameter. Die Formatierung ist ReStructuredText(kurz ReST), und kann mit Hilfe von Dokumentationstools ausgelesen und später als eine lokale Webseite mit automatisch generierten Verlinkungen ausgegeben werden.

8 Fazit

Kommt noch

8.1 Soll- /Ist-Vergleich

Kommt noch

8.2 Post Mortem

Kommt noch

8.3 Ausblick

Kommt noch
Weitere Schritte:
Python einbindung
Optimierung des Algorithmusfilters
Optimierung des Rechenaufwands des Algorithmus

/home/freundt/business/GA_Logo.jpeg

Abbildung 1: Use-Case Diagramm des QGJoin Programms

Abbildung 2: UML Diagramm der bisherigen Geschäftslogik

9 Anhang

9.1 Diagramme und Grafiken

9.1.1 Vergleich: DLD Aufwand vs QGram Aufwand mit Fuzzy Join

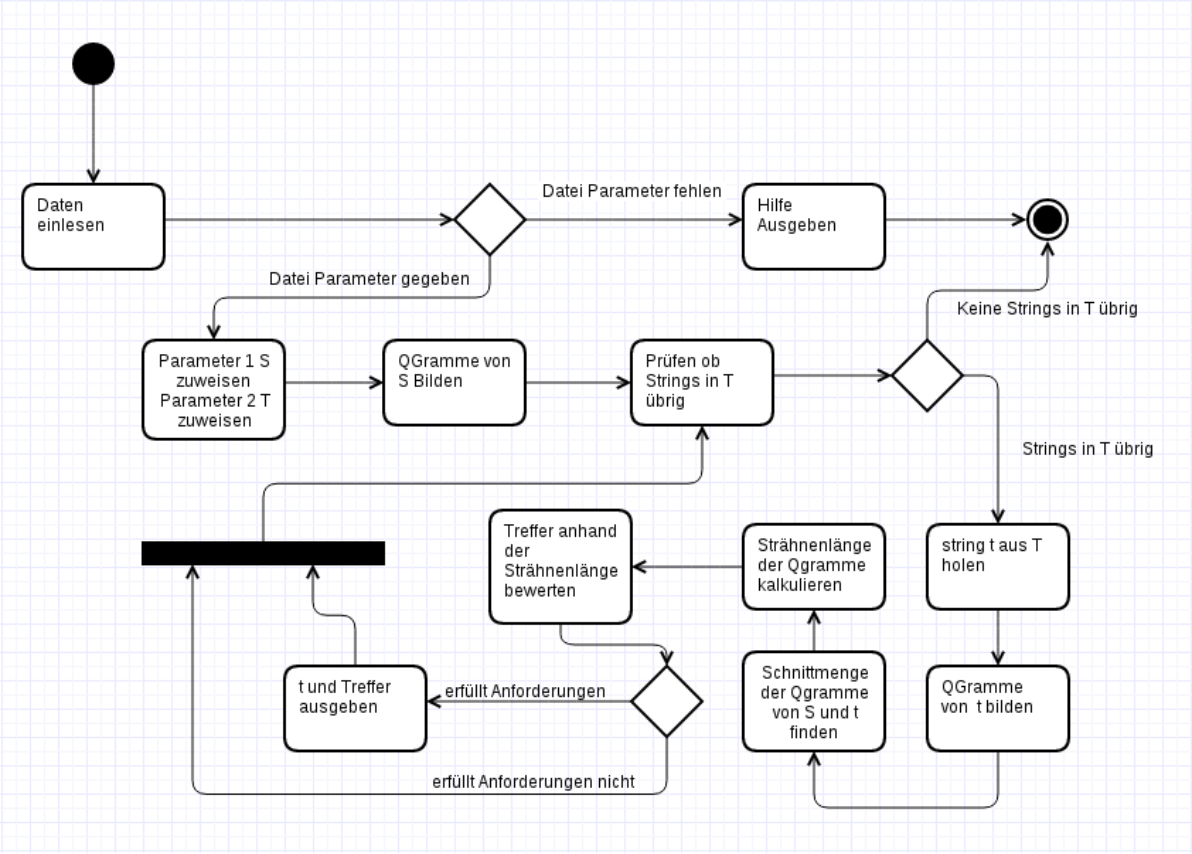
9.1.2 Vergleich: Theoretischer DLD vs QGram Zeitvergleich

9.1.3 Use-Case QGJoin

9.1.4 Geschäftslogik Alt

9.1.5 Aktivitätsdiagramm: Programmlogik

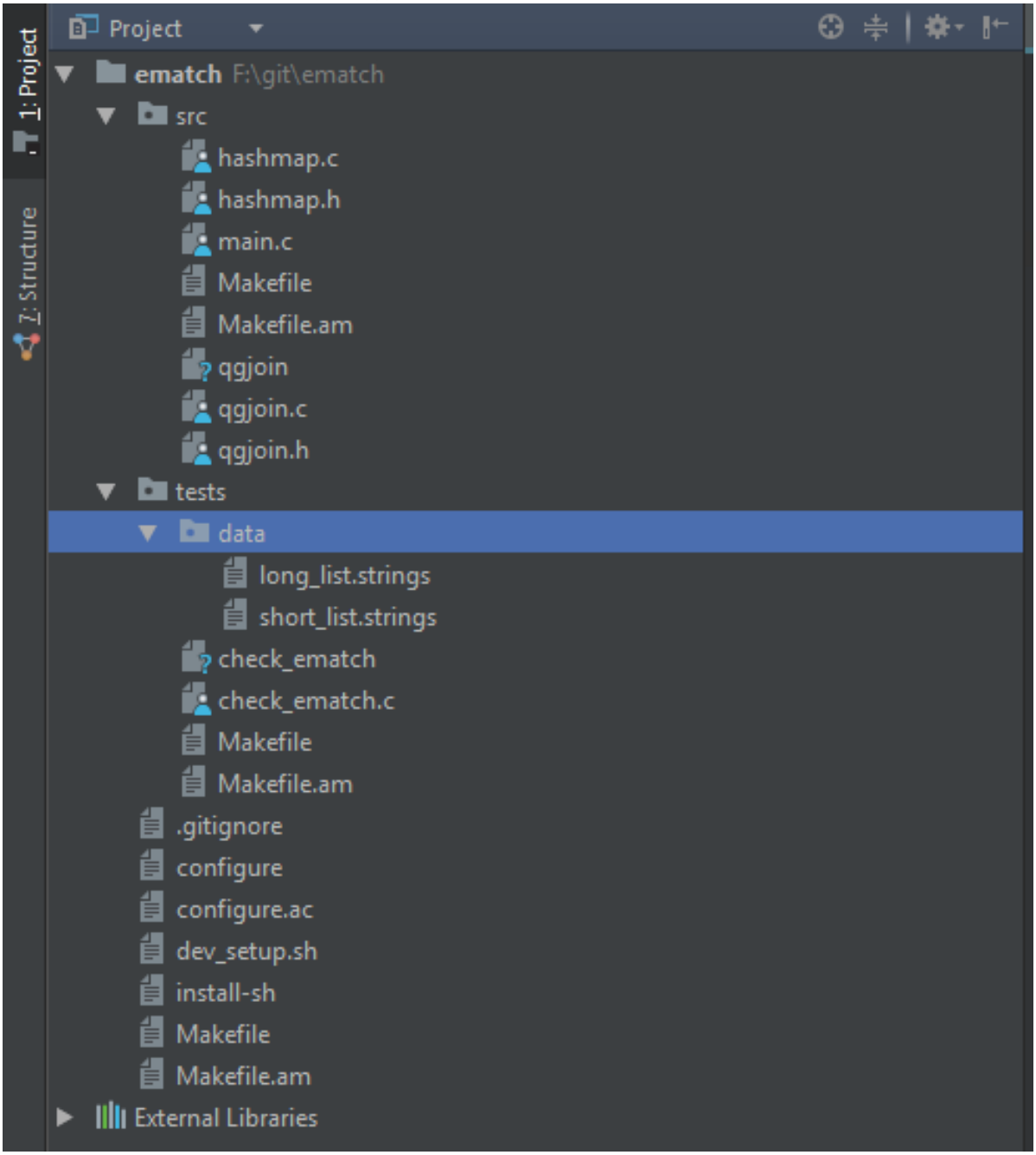
Abbildung 3: Programmlogik als UML Aktivitätsdiagramm



```
/home/freundt/business/GA_Logo.jpeg
```

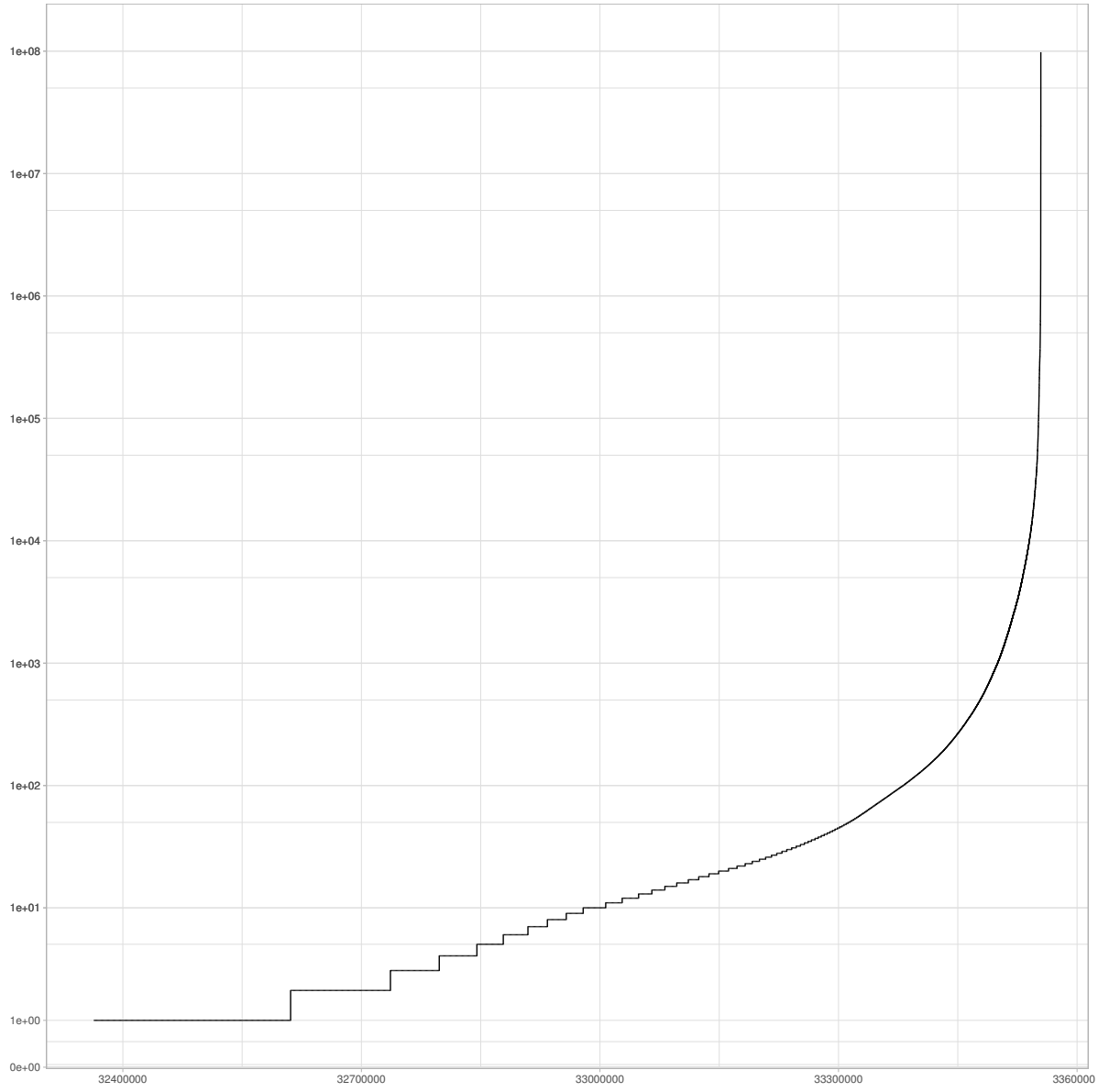
9.1.6 Projektstruktur

Abbildung 4: Übersicht der Projektstruktur.



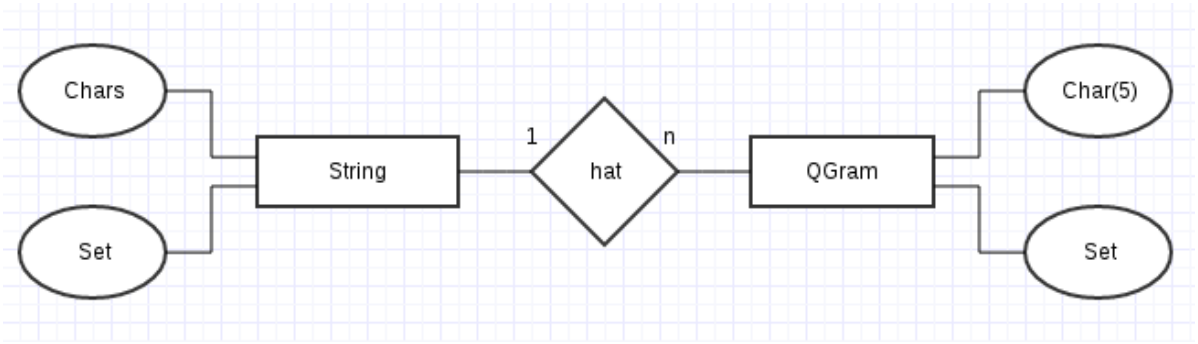
/home/freundt/business/GA_Logo.jpeg

Abbildung 5: Verteilung der QGramme im zu bearbeitenden Datensatz



/home/freundt/business/GA_Logo.jpeg

Abbildung 6: UML Darstellung des Datenmodells



9.1.7 Analyse: Verteilung der Qgramme

9.1.8 Datenmodell

9.2 Tabellen

9.2.1 Projektkosten

9.2.2 Nutzwertanalyse: Projektentwicklung

9.2.3 Nutzwertanalyse: Programmiersprachen

9.2.4 Nutzwertanalyse: Datentyp der Menge Q

9.3 Auszüge

9.3.1 Lastenheft

1. Spezifikationen des Algorithmus
   1.1. Der Algorithmus muss 2 Listen von Strings verarbeiten können, wobei eine dieser Listen garantiert in den Speicher passt und die zweite eine beliebige Größe haben kann.
   1.2. Der Algorithmus muss aus der ersten Liste sämtliche Strings finden, welche den Strings aus der zweiten Liste ähneln. Die gefundenen Strings sollen ausgegeben werden.
   1.3. Der Algorithmus ist in C zu programmieren.
   1.4. der Algorithmus verarbeitet 7-Bit ASCII Strings schreibungsunabhängig. Zeichensetzung ist beim Vergleich generell zu ignorieren; Ausnahmen bilden die Charaktere Bindestrich, Unterstrich und Leerzeichen.
2. Spezifikationen des Kommandozeilenprogramms
   2.1. Der Algorithmus muss über die Kommmandozeile ausführbar sein.
   2.2. Die Erste Liste wird immer als Dateiname übergeben.

/home/freundt/business/GA_Logo.jpeg

Tabelle 1: Projektkosten

Vorgang	Mitarbeiter	Stunden	Personal	Ressources	Gesamt
Entwicklungskosten	1	70	385.00 €	1,295.00 €	1,680.00 €
Fachgespräch	2	5	350.00 €	92.50 €	1,842.50 €
Code Review	2	2	140.00 €	37.00 €	317.00 €
Abnahme	2	0.5	35.00 €	9.25 €	44.25 €
				Projektkosten Gesamt:	**3,883.75 €**

/home/fre /bu ine s/GA_Logo.jpeg

Tabelle 2: Nutzwertanalyse Projektentwicklung

Kriterien	Gewicht	Externe Entwicklung		Interne Entwicklung		Dedizierter 12-Kerner		Cluster	
		Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt
Projektkosten	45.00%	4	1.8	5	2.25	4	1.8	3	1.35
Nachhaltigkeit	15.00%	4	0.6	4	0.6	1	0.15	1	0.15
Flexibilitt	5.00%	3	0.15	5	0.25	4	0.2	4	0.2
Unterhaltskosten	35.00%	4	1.4	4	1.4	2	0.7	1	0.35
Gesamt	**100.00%**		**3.95**		**4.5**		**2.85**		**2.05**

/home/fre[illegible]A_Logo.jpeg

Tabelle 3: Nutzwertanalyse der Programmiersprachen

Kriterien	Gewicht	C		C++		Java		Haskell	
		Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt	Bewertung	Gesamt
Performanz	30.00%	5	1.5	5	1.5	4	1.2	5	1.5
Speichereffizienz	30.00%	5	1.5	5	1.5	3	0.9	4	1.2
Dokumentation	20.00%	5	1	4	0.8	4	0.8	2	0.4
Systemische Vorraussetzungen	10.00%	5	0.5	5	0.5	3	0.3	3	0.3
Datenstrukturen	10.00%	3	0.3	4	0.4	5	0.5	5	0.5
Gesamt	**100.00%**		**4.8**		**4.7**		**3.7**		**3.9**

/home/freundt/business/GA_Logo.jpeg

Tabelle 4: Nutzwertanalyse der Datentypen für Q

Kriterien	Gewicht	Hashmap Bewertung	Gesamt	Binärbaum Bewertung	Gesamt
Speicheraufwand	25.00%	2	0.5	2	0.5
Schnelligkeit Daten Einfügen (Insert	20.00%	5	1	4	0.8
Schnelligkeit Auslesen (Look-up)	55.00%	5	2.75	4	2.2
Gesamt	**100.00%**		**4.25**		**3.5**

2.3. Die zweite Liste kann entweder als Dateiname oder über STDIN übergeben werden.

2.4. die GNU Standards für Kommandozeilenprogramme müssen eingehalten werden.

2.5. Die POSIX Utility Guidelines müssen eingehalten werden.

2.6. Das Programm muss unter openSuSE 13.1 laufen.

2.7. Alle für das Programm erforderliche Abhängigkeiten (z.B. Pakete, Bibliotheken) müssen für openSuSE 13.1 verfügbar sein.

2.8. Das Programm muss auf einem einzigen Kern lauffähig sein, und darf das laufen auf mehreren Kernen in einem Shared-Memory-Environment unterstützen.

2.9. Das Programm darf mit maximal 32GB Arbeitsspeicher arbeiten, und nicht mehr als 200 GB Festplattenspeicher zur Kalkulation verwenden. Zwischenprozessliche Kommunikationswege sind ausgeschlossen.

2.10. Das Programm gibt Fehler und Debug Daten über STDERR aus.

2.11. Errechnete Ergebnisse werden über STDOUT ausgegeben.

2.12. Die Zielplatform ist Intel x86-64 (intel64).

3. Spezifikationen der Qualitätssicherung

3.1. Der Algorithmus muss überprüfbar sein.

3.2. Unittests sind verfügbar gemacht worden.

4. Spezifikationen der Dokumentation

4.1. Sämtliche Funktionen müssen über einen Doc String dokumentiert werden

4.2. Die Doc Strings der Funktionen müssen auf Englisch geschrieben werden.

9.3.2 Pflichtenheft

9.3.3 Pseudocode: Algorithmus

Abbildung 7: Aktivitätsdiagramm der Programmlogik

Listing 1: Pseudocode des zu implementierenden Algorithmus

9.3.4 Quellcodeauszug: Dynamische Array

Listing 2: Auszug aus dem C Quellcode der dynamischen Array

```
typedef struct {
  int *array;
  size_t used;
  size_t size;
} Array;

/**
Insert given element into array.

    :param a: Array into which the given element is to be inserted.
    :type a: Struct Array ptr

    :param element: element to be looked up.
    :type element: int

    :return: void
    :return type: void
**/
void insertArray(Array *a, int element) {
  if (a->used == a->size) {
    a->size *= 2;
    a->array = (int *)realloc(a->array, a->size * sizeof(int));
  }
  a->array[a->used++] = element;
}
```

9.3.5 Quellcodeauszug: Hashtable Implementation

Listing 3: Auszug aus dem C Quellcode der dynamischen Array

```
/*
 * Hashing function for a string
 */
unsigned int hashmap_hash_int(hashmap_map * m, char* keystring){

    unsigned long key = crc32((unsigned char*)(keystring), strlen(keystri

        /* Robert Jenkins' 32 bit Mix Function */
        key += (key << 12);
        key ^= (key >> 22);
        key += (key << 4);
        key ^= (key >> 9);
```

```
        key += (key << 10);
        key ^= (key >> 2);
        key += (key << 7);
        key ^= (key >> 12);

        /* Knuth's Multiplicative Method */
        key = (key >> 3) * 2654435761;

        return key % m->table_size;
}

/*
 * Return the integer of the location in data
 * to store the point to the item, or MAP_FULL.
 */
int hashmap_hash(map_t in, char* key){
        int curr;
        int i;

        /* Cast the hashmap */
        hashmap_map* m = (hashmap_map *) in;

        /* If full, return immediately */
        if(m->size >= (m->table_size/2)) return MAP_FULL;

        /* Find the best index */
        curr = hashmap_hash_int(m, key);

        /* Linear probing */
        for(i = 0; i< MAX_CHAIN_LENGTH; i++){
                if(m->data[curr].in_use == 0)
                        return curr;

                if(m->data[curr].in_use == 1 && (strcmp(m->data[curr].key
                        return curr;

                curr = (curr + 1) % m->table_size;
        }

        return MAP_FULL;
}
```

9.3.6 Auszug: Entwicklerdokumentation

/home/freundt/business/GA_Logo.jpeg

Listing 4: Beispiel eines Docstrings für Entwickler

```
/**
Extract QGrams from given string to given array;

    :param s: string array which contains the string to be split into Qgr
    :type s: char ptr

    :param Q_of_s: Array in which to store the extracted Qgrams.
    :type Q_of_s: char ptr array

    :param len_of_s: length of s
    :type len_of_s: int

    :return: void
    :return type: void
**/
void populate_qgram_array(char *s, char *Q_of_s[], int len_of_s){
    int i = 0;
    while(i + Q_SIZE-1 < len_of_s){
        Q_of_s[i] = malloc(Q_SIZE+1);
        memcpy(Q_of_s[i], s+i, Q_SIZE+1);
        Q_of_s[i][Q_SIZE] = '\0';
        encode_qgram(Q_of_s[i]);
        i++;
    }
}
```

9.3.7 Auszug: Endnutzerdokumentation

Listing 5: Ausgabe der Endnutzerdokumentation über den –help parameter

```
nils@chantico:~/git/ematch/src> ./qgjoin --help
Usage: qgjoin [OPTION] LEFT_LIST RIGHT_LIST
   or: cat RIGHT_LIST | qgjoin LEFT_LIST
Compare two lists of strings using a Q-gram-based Fuzzy Join algorithm.
LEFT_LIST is expected to be a path to a file. It is read entirely into memory;
it is your job to make sure it fits. The RIGHT_LIST argument can either be a file
path, or input piped via STDIN.

LEFT_LIST              List of strings to be kept in memory.
                       Type: FILE PATH
RIGHT_LIST             List of strings to compare LEFT_LIST against.
                       Type: FILE PATH or STDIN

--help, -h             Display this help output
--version, -v          Display the version of the program.
nils@chantico:~/git/ematch/src>
```