PROJEKTDOKUMENTATION

AUSBILDUNG ZUM FACHINFORMATIKER F.
ANWENDUNGSENTWICKLUNG

Abschlussprüfung Winter 2017
IHK Berlin

Auszubildender:
Nils Diefenbach
Ident-Nummer:
3590244

Ausbilder:
Sebastian Freundt
GA FINANCIAL SOLUTIONS GMBH

13. September 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Abkürzungsverzeichnis

- **GLEIS**: Global Legal Entity Identifier System
- **CLI**: Command Line Interface
- **CSV**: Comma-separated Values Format
- **DLD**: Damerau-Levenshtein Distanz
- **IB**: Interactive Brokers, Datenanbieter 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 computergestütztes 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 GmbH 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 gegebenen 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 Kabushiki-gaisha“ ergibt.

Bei der GA Financial Solutions GmbH werden Zuordnungen zwischen Symbologien aktuell ad hoc mit Heuristiken und in 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 zugeteilt 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 sind, sind folgende Themen explizit nicht Teil der Aufgabe:

- Datenintegrität – Die Überprüfung des Datenformats ist nicht Teil des 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. Projektplanung

3.1. Projektphasen

Dem Autor standen 70 Arbeitsstunden zur Umsetzung des Projektes zur Verfügung. Vor Beginn des Projekts wurden diese auf die, für die Softwareentwicklung typischen, Phasen verteilt. Eine ausführlichere Aufteilung der Stunden kann der Tabelle "Detaillierte Zeitplanung" (5) entnommen werden.

3.2. Abweichungen vom Projektantrag

Ursprünglich wurde im Projektantrag eine Einbindung des Algorithmus in Python genannt. Diese wurde, nach gründlicherer Recherche und in Absprache mit der Abteilung Datenverarbeitung, jedoch auf einen späteren Zeitpunkt verlegt und nicht Teil des Pflichtenhefts. Grund hierfür ist, dass die C-Implementierung eine höhere Priorität hat als die Python Bibilothek, der Einbindungsprozess letzterer jedoch komplexer ist als zuvor angenommen. Deshalb wurde beschlossen die 7 Stunden, welche der Python-Implementierung zugewiesen wurden, stattdessen der Optimierung des C-Programms zuzuweisen.

3.3. Ressourcenplanung

In der Ressourcenplanung finden sich sämtliche Einheiten aufgelistet, die für die Realisierung des Projektes benötigt wurden. Hierbei ist zu erwähnen, dass mit „Einheiten"´ Hard- und Softwareressourcen, sowie personelle Ressourcen gemeint sind. Die Auflistung der Ressourcen kann im Abschnitt C.1 des Anhangs eingesehen werden.

3.4. Entwicklungsprozess

Für die Umsetzung des Projekts wurde vom Autor ein agiler Entwicklungsprozess in Form von Kanban ausgewählt. Ausschlaggebend hierfür war die Tatsache, dass es bei Kanban keine festen Phasenlängen gibt und Prioritäten sofort angepasst werden, sobald neue Informationen (z.B. Abschluss von Arbeitspaketen, Verzögerungen im Ablauf) hinzukommen.

Das Kanbanboard wurde hierbei in fünf Bahnen aufgeteilt, welche repräsentativ für die fünf Stufen, in welche der Autor die Implementierungsphase unterteilt hat, stehen:

- **Offen** – Arbeitspakete in dieser Spalte befinden sich noch nicht in der Entwicklung, und warten darauf bearbeitet zu werden.
- **Testentwicklung** – Es werden Tests geschrieben, welche alle Anforderungen an den erforderlichen Code zur Fertigstellung des Arbeitspakets überprüfen.
- **Implementierung** – Das Arbeitspaket wird bearbeitet und der dazugehörige Code erstellt. Hierbei wird Code geschrieben, der den Anforderungen des zuvor geschriebenen Tests gerecht wird.

- **Verifizierung** – Das Arbeitspaket wurde fertiggestellt und muss nun von der Abteilung Datenverarbeitung abgesegnet werden.
- **Erledigt** – Arbeitspakete in dieser Spalte sind fertiggestellt, vom Auftraggeber überprüft und unterschrieben worden.

Das Augenmerk liegt hierbei vor allem auf der Spalte „Verifizierung"´. Dieser bereichert den Entwicklungsprozess um einen formellen, fest vorgesehenen Zeitpunkt zur Feedback-Kommunikation. So können frühzeitig, möglicherweise notwendige Anpassungen geäußert und unter Umständen noch umgesetzt werden. Wichtig hierbei ist, dass die Überprüfung nicht im Rahmen eines Meetings geschieht, da sonst der Entwicklungsprozess gestört wird. Ein Feedback zu den einzelnen Arbeitspaketen wird ausschließlich über das Kanbanboard kommuniziert.

4. Analysephase

4.1. 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 Instrumentsymbole hinzu.

Zur Zeit werden diese Symbole ad hoc mittels Heuristiken und in Handarbeit verglichen. Dabei wird meist nur eine kleine Teilmenge benutzt und 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 theoretischer Lösungsansatz auf Basis der Damerau-Levenshtein-Distanz[2] (kurz: DLD). Dieser sieht vor, eine M $\times$ M -Matrix zu generieren, wobei M sämtliche Symbole aller Anbieter darstellt. Dies resultiert in einem ca. 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 ca. 16 Milliarden DLD-Aufrufe täglich benötigt, um die Datenbank zu aktualisieren. Pro Tag entspricht dies weiteren 33 Minuten Rechenzeit auf einem 4-Kerner.

Gewünscht wurde eine performantere und ressourcenschonendere Implementierung des obigen Prozesses, sodass die Daten auf einem firmeninternen 4-Kerner zeitnah ausgerechnet werden können. Hierbei war es nicht unbedingt erforderlich, die Damerau-Levenshtein-Distanz zu verwenden. Des weiteren sollte der Zeitaufwand der ad hoc-Vergleiche der Ergebnisse drastisch reduziert werden.

4.2. Wirtschaftlichkeitsanalyse

4.2.1. 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 Problem löst, ist das Open-Source-Projekt *fuzzy-join*[4]. Dieses Tool vergleicht Daten jedoch nur in

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

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

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

eine Richtung (*One-To-Many*), während das Abschlussprojektszenario 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 Nutzwertanalyse (Anhang Abschnitt 2) erstellt, um diverse Lösungswege zu evaluieren. Diese wurden anhand der folgenden Kriterien bewertet:

- **Projektkosten** – Wie hoch sind die Personal-, Server- und/oder Softwarekosten der jeweiligen Lösung?
- **Nachhaltigkeit** – Gibt es externe Abhängigkeiten? Wenn ja, wie viele und besteht das Risiko eines technischen 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?

4.2.2. 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 im Anhang (Abschnitt 1) 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.

4.2.3. Amortisationsdauer

Nach einer Einschätzung der Abteilung für Datenverarbeitung wird zur Zeit jede Woche ein Mitarbeiter etwa 10 Stunden pro Woche benötigt, um Symboliken miteinander zu vergleichen. Dies ergibt, mit dem in der Projektkostenkalkulation verwendeten Stundenlöhnen und Kosten für Räumlichkeiten, Hard- und Software also

$$10\frac{\text{h}}{\text{Woche}} \times 55\frac{€}{\text{h}} = 550\frac{€}{\text{Woche}}$$

Der Autor verspricht sich durch das Programm eine Beschleunigung des Vergleichsprozesses um etwa 30%, da der größte zeitliche Anteil vom manuellen Vergleich beansprucht wird. Im Optimalfall ist eine Verbesserung von 80% zu erwarten, womit der

Prozess noch etwa zwei Stunden dauern würde. Eine Verbesserung von 55% wird vom Autor als realistisch eingeschätzt[5].

Anhand dieser Werte lassen sich folgende Amortisationsdauern errechnen:

Bester Fall:

$$\frac{3.883{,}75€}{550\frac{€}{\text{Woche}} \times 80\%} \approx 9\text{Wochen}$$

Wahrscheinlichster Fall:

$$\frac{3.883{,}75€}{550\frac{€}{\text{Woche}} \times 55\%} \approx 13\text{Wochen}$$

Schlechtester Fall:

$$\frac{3.883{,}75€}{500\frac{€}{\text{Woche}} \times 30\%} \approx 24\text{Wochen}$$

4.3. 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 (Abschnitt C.2).

[5] Die Zeiten wurden anhand von Kalkulationen der theorethischen Rechenvorgänge unter Einbezug der Schreib-, und Lesegeschwindigkeiten des Speichers, sowie Geschwindigkeit der CPU geschätzt.

5. Entwurfsphase

5.1. Zielplattform

Wie dem Abschnitt 1 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 und bietet eine ideale Schnittstelle, um effizient im Speicher agieren zu können. Zur Überprüfung dieser Einschätzung wurden weitere Sprachen anhand einer Nutzwertanalyse mit der Sprache C verglichen. Die wichtigste Rolle spielten hierbei folgende Kriterien:

- **Performanz** – Geschwindigkeit der Sprache in der High-Performance-Computing (HPC) Domäne
- **Speichereffizienz** – Der Fußabdrucks 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 Voraussetzungen** – Anzahl und Aufwand der Voraussetzungen, um die Sprache zu nutzen. Auch hier ist weniger besser.

Die Ergebnisse dieses Vergleichs wurden in der Nutzwertanalyse präsentiert. Sie ist dem Anhang (Abschnitt 3) dieses Dokumentes beigefügt.

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

5.2. 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, wobei **Q** die Länge des Teilstrings ist. Dieser Code wurde in Form eines Aktivitätsdiagramm (Anhang 3) visualisiert. Der zugrunde liegende Pseudocode des Diagramms ist ebenfalls im Anhang (Listing 1) einsehbar.

5.3. Architekturdesign

Das Projekt besteht aus zwei Header-Dateien und einer main.c-Datei. Die Header-Dateien beinhalten zum einen die algorithmusspezifischen Funktionen zum Vergleichen von Strings, zum anderen eine Open-Source-Hashmap-Implementierung.

Auf Programmebene wurde die Logik in zwei Abschnitte unterteilt:

- **Vorbereitung**
  Hier werden Daten aus der ersten Datei geladen und benötigte Datenstrukturen im Speicher angelegt. 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 Q-Gramme, welche sich aus den jeweils einzelnen Strings in **S** generieren lässt

- **Ausführung**
  - **Variable I** = Die Menge, welche die gleiche Größe hat wie **S** und die Anzahl der Q-Gramme zählt, die 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 Q-Gramme in der aktuell ausgelesenen Zeile
  - **Variable M** = Eine Menge der Indizes von Strings, welche auf die Q-Gramme **T** am besten passen

Mit Ausnahme von **Q** und **M** konnten 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. Diese benötigt zwar mehr Speicherplatz, verfügt jedoch auch über einen O(1)-Suchaufwand - dies bot einen entscheidenden Geschwindigkeitsvorteil beim späteren Vergleich der Q-Gramme. Auch hier wurde wiederum eine Nutzwertanalyse (Tabelle 4) durchgeführt, um sicherzustellen, dass die persönlichen Annahmen des Autors mit den technischen Gegebenheiten übereinstimmen. Hierzu wurden zwei in Frage kommende Datenstrukturen nach folgenden Kriterien bewertet:

1. **Schnelligkeit Suchbefehl** – Wie hoch ist die Komplexität des Suchbefehls 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 (5-Bit ASCII) begrenzt[6]. 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 (Abbildung 4) dargestellt, welcher die Anzahl Q-Gramme auf der X-Achse, der Anzahl der Strings in der diese vorkommen auf der Y-Achse als Quantil gegenüber stellt.

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 testgetriebenen 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.

5.4. 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.

Die bevorzugte Oberfläche der Endnutzer ist die Kommandozeile, daher wird auf eine grafische Oberfläche verzichtet.

5.5. 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 Anhang (Abbildung 5) eingesehen werden.

[6] Maximale Anzahl an Schlüsselworten einer Hashmap im Kontext dieses Projekts = Länge des Alphabets$^{\text{Q-Grammlänge}}$

Es ist klarzustellen, dass die GA Financial Solutions GmbH es bevorzugt nicht mit Datenbanksystemen, wie zum Beispiel MySQL oder MongoDB, zu arbeiten. Stattdessen werden komprimierte CSV-Dateien verwendet, welche meist nur aus drei Spalten (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.

5.6. Geschäftslogik

Zur Veranschaulichung der bisherigen Geschäftslogik, wurde ein Aktivitätsdiagramm (Abbildung 1) 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. Hinzu 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. Nach Einbindung des Programms durch die Abteilung Datenverarbeitung ergibt sich eine neue, in Abbildung 2 ersichtliche, Geschäftslogik.

5.7. Pflichtenheft

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 Pflichtenhefts befindet sich im Anhang (Abschnitt C.3).

6. Implementierungsphase

6.1. 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 dazugehörige Funktionen betrachtet und als Arbeitspaket beschrieben.

6.2. Implementierung der Datenstrukturen

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

6.2.1. Implementierung der Hashmap

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

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

6.2.2. Implementierung des Dynamisch wachsenden Arrays

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 dem Array interagieren zu können.

7. 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 vorgestellt.

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

7.1. Code-Flight mit der Abteilungsleitung

Um die Qualität des Codes zu gewährleisten, wurde für das Projekt die Abnahme über einen Code-Flight[7] 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

7.2. Präsentation des Programms

Nach dem Code-Flight mit dem Abteilungsleiter wurde für das gesamte Team der Abteilung Datenverarbeitung ebenfalls eine Präsentation 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. Abschließend wurde ein Post Mortem gehalten und die gewonnenen Erkenntnisse in Bezug auf die Arbeit mit Kanban und der testgetriebenen Entwicklung erläutert.

[7] zu deutsch „Programm-Flug"´ - bei dieser Technik erklärt eine Person dem Zuhörer laut den Code und beschreibt was dessen Funktion ist.

8. 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.

8.1. 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 Aufruf erzeugt eine kurze Beschreibung des Programms und dessen Parameter.

8.2. 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), kann mit Hilfe von Dokumentationstools ausgelesen und später als eine lokale Webseite mit automatisch generierten Verlinkungen ausgegeben werden.

9. Fazit

9.1. Soll- / Ist-Vergleich

Rückblickend kann festgehalten werden, dass alle im Pflichtenheft abgesprochenen Funktionen und Anforderungen eingehalten worden sind und zur vollsten Zufriedenheit des Auftraggebers geliefert wurden. Hierbei spielt die Entscheidung, die Einbindung des C-Codes in Python zu streichen, sicherlich eine elementare Rolle. Wie man erkennen kann, wurden die freigewordenen Stunden effektiv ausgenutzt um den Anforderungen des Projekts gerecht zu werden. Die von der IHK vorgegebenen 70 Stunden wurden dabei konsequent eingehalten.

9.2. Post Mortem

Vor allem während der Entwurfsphase konnte der Autor von seinen Kenntnissen aus dem Projektmanagement profitieren. Besonders bei den Nutzwertanalysen waren diese hilfreich. Weiter erwies sich die Anwendung der testgetriebenen Entwicklung als große Unterstützung. So wurden lästige Segmentation Faults frühzeitig erkannt und die Funktionen (und somit das gesamte Programm) konnten zügiger fertiggestellt werden. Auch das genutzte Kanban-Board erwies sich als großartiges Werkzeug, um Prozesse und deren Fortschritt zu dokumentieren. Kurze Kommunikations- und Entscheidungswege ermöglichten eine agile Entwicklung des Projekts und werden dem Autor auch weiterhin als ein wichtiger Baustein produktiver Arbeitsumgebungen in Erinnerung bleiben.

9.3. Ausblick

Punkte, welche im Lastenheft genannt, aber nicht mit in das Pflichtenheft genommen wurden, stehen nun natürlich auf der Liste der weiteren Schritte. Da der Code gut dokumentiert und leserlich geschrieben wurde, ist eine Cythoneinbindung nun effizient umsetzbar. Durch die bereitgestellten Tests können ebenfalls schnell Python Unittests erstellt und somit die Einbindung des C-Codes effektiv und verlässlich auch in Python überprüft werden.

Schlussendlich bleiben noch zwei Punkte, welche in Zukunft angegangen werden können, um das Programm weiter zu verfeinern:

1. Die Optimierung der Programmlogik

    - Verfeinerung der Hashmap: Da die maximale Anzahl an Schlüsselworten bekannt ist, kann die Hashmap dahingehend angepasst werden, dass exakt so viele „Felder"´ im Speicher reserviert werden wie nötig
    - Das Auswahlkriterium kann als Funktion implementiert und der Vergleichsfunktion als Pointer übergeben werden – so können Änderungen unkomplizierter realisiert werden

2. Weiteres fine-tuning des Q-Gramm Filters

A. Diagramme und Grafiken

A.1. Geschäftslogik

Abbildung 1: UML Diagramm der bisherigen Geschäftslogik

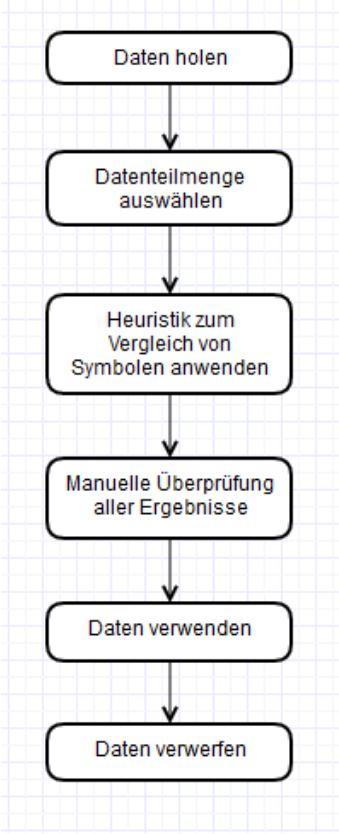


Abbildung 2: UML Diagramm der Geschäftslogik nach Fertigstellung des Projekts

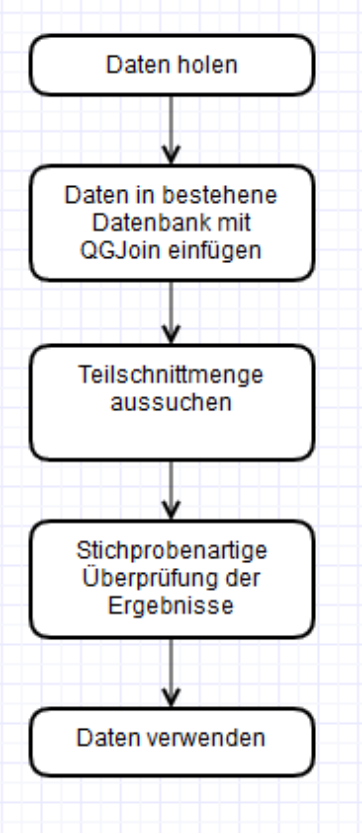


Abbildung 3: Programmlogik als UML Aktivitätsdiagramm

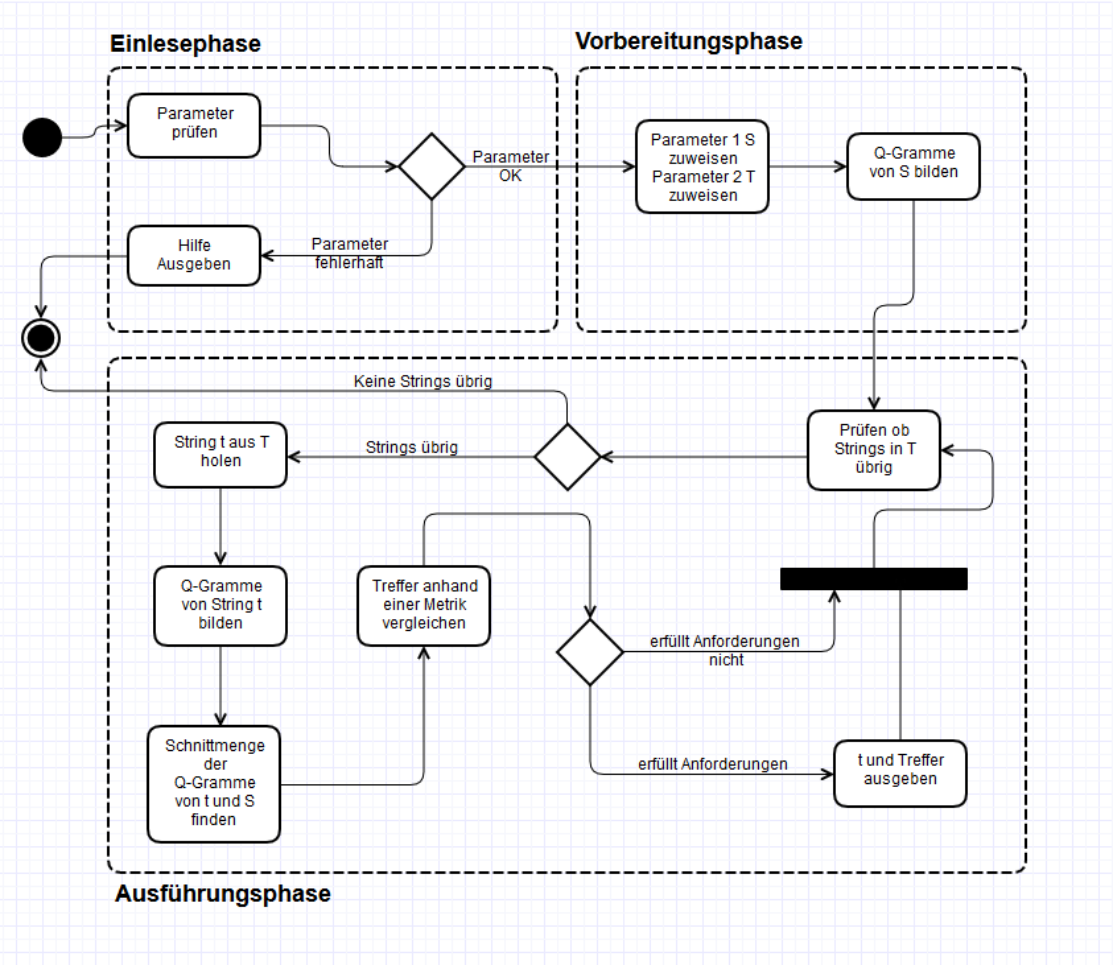


Abbildung 4: Verteilung der QGramme im zu bearbeitenden Datensatz

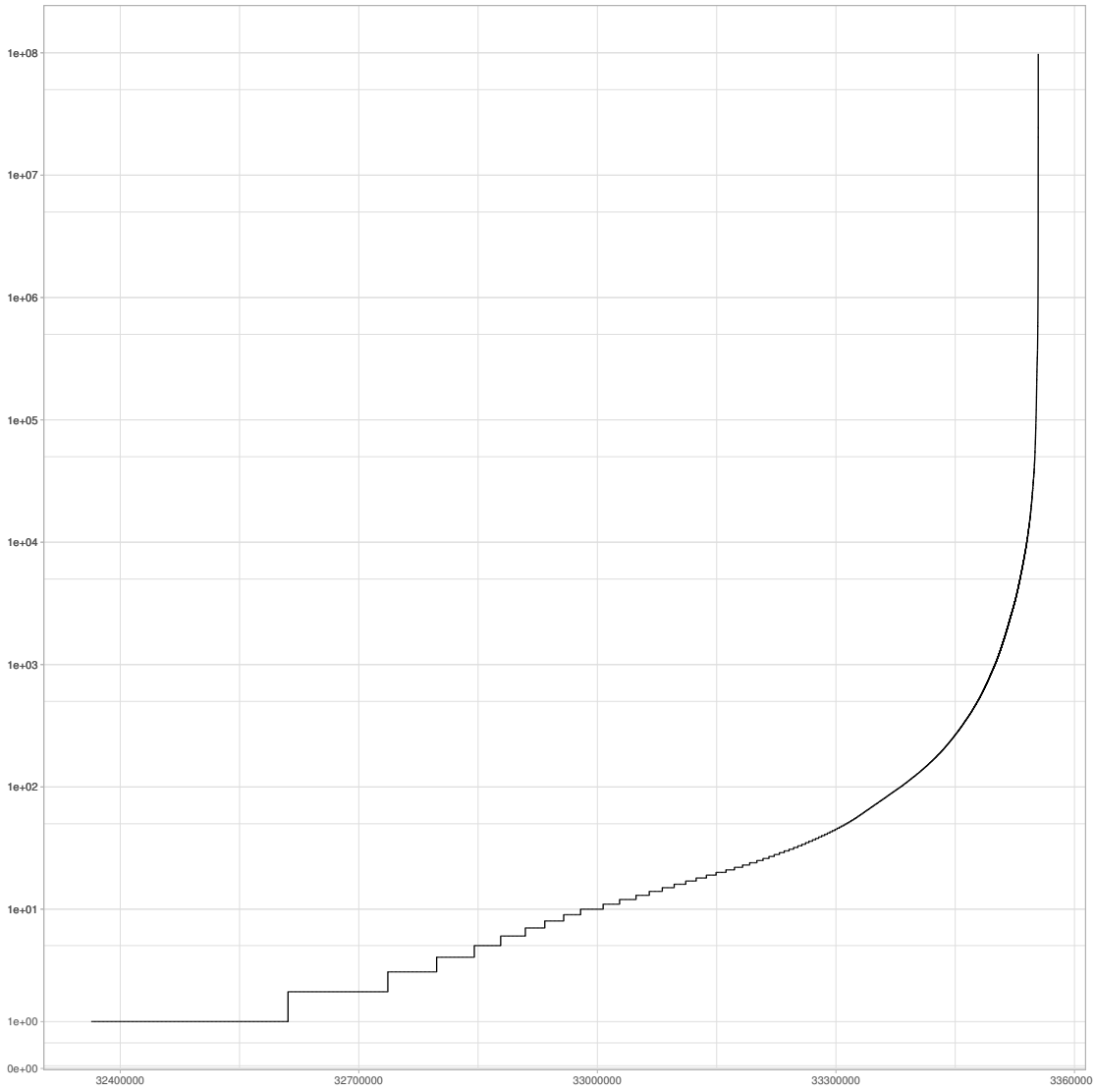
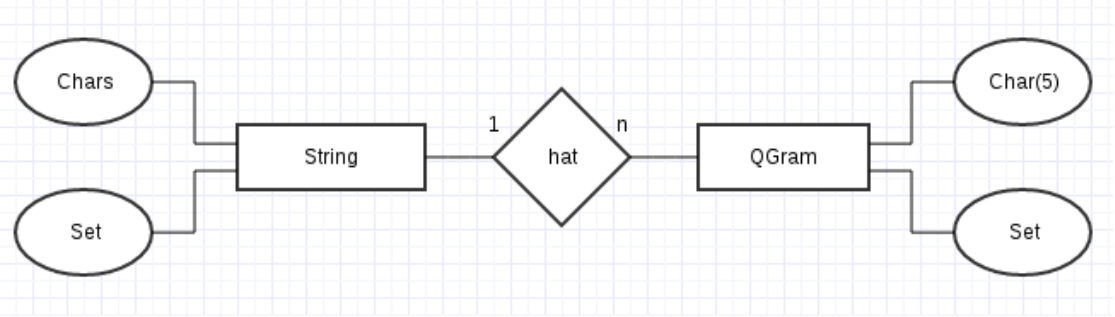


Abbildung 5: UML Darstellung des Datenmodells



B. Tabellen

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 €**

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
Flexibilität	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**

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**

Tabelle 4: Nutzwertanalyse der Datentypen für Q

Kriterien	Gewicht	Hashmap		Binärbaum	
		Bewertung	Gesamt	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**

Tabelle 5: Detaillierte Zeitplanung

Analysephase		**9h**
1. Analyse des Ist-Zustands		3h
1.1 Fachgespräch mit der Datenverarbetungsabteilung	*1h*	
1.2 Prozessanalyse	*2h*	
2. Make or BuyËntscheidung und Wirtschaftlichkeitsanalyse		1h
3. Erstellen eines Üse-Case"Diagramms		2h
4. Erstellen eines Lastenhefts mit der Abteilung Datenverarbeitung		3h
Entwurfsphase		**26h**
1. Prozessentwurf		2h
2. Algorithmusentwurf		14h
3. Entwurf von C Datenstrukturen		5h
3.1 Speicherarme Q-Gramm Liste	*3h*	
3.2 Selbst-ausgleichender Binär Baum	*2h*	
4. Entwurf der Kommandozeilenoberfläche		2h
5. Erstellen eines UML Komponentendiagramms der Anwendung		4h
6. Erstellen des Pflichtenhefts		4h
Implementierungsphase		**21h**
1. Unittests erstellen		5h
2. C Datenstrukturen Implementieren		8h
3. Algorithmus Logik in C Implementieren		8h
Abnahmephase		**2h**
1. Code-Flight mit Abteilungsleitung		1h
2. Präsentation		1h
Dokumentationsphase		**9h**
1. Erstellen der Nutzerdokumentation als Man Page		2h
2. Erstellen der Projektdokumentation		7h
Pufferzeit		2h
1. Puffer		3h
Gesamt		**70h**

C. Auszüge

C.1. Ressourcen

Hardware

- Büroarbeitsplatz mit vernetztem Rechner
- Laptop

Software

- openSuSE 13.1 - Linux Distribution
- Atom IDE - Modulare, Open-Source Entwicklungsumgebung für eine Vielfalt an Sprachen
- LaTeX - Textsatzsystem zur Erstellung von Dokumenten
- GNU Compiler Collection (GCC) - Compilersammlung des GNU Projekts
- Git - Dezentralisierte Versionsverwaltung

Personal

- Mitarbeiter der Abteilung Datenverarbeitung - Festlegung der Anforderungen sowie Abnahme des Projektes
- Entwickler - Umsetzung des Projekts
- Anwendungsentwickler - Code Review

C.2. 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.

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. Die Systemvoraussetzungen des Programms darf mindestens 32GB Arbeitsspeicher und mindestens 200 GB Festplattenspeicher voraussetzen. Zwischenprozessliche Kommunikationswege dürfen nicht vorausgesetzt werden. 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.

C.3. Pflichtenheft

- Das Programm nimmt die Daten zum einen als Dateipfad, und zum anderen als Dateipfad oder Input über STDIN an.
- Die Ausführung des Programms mit dem Parameter h bzw help ruft einen Hilfetext zur Nutzung des Programms auf.
- Die Ausführung des Programms mit dem Parameter v bzw version ruft die derzeitige Versionsnummer des Programms auf.
- Das Programm wird auf und für openSuSE 13.1 entwickelt und kompiliert.

- Das Programm wird keine Form von Multithreading bzw Multiprocessing verwenden.
- Die zuerst, über einen Dateipfad angegebene Liste wird in den Speicher gelesen; die zweite kann entweder als Dateipfad oder über STDIN eingegeben werden und wird Zeile für Zeile geladen.
- Das Programm garantiert einen konstanten Speicherverbrauch, welcher abhängig von der ersten Datei ist.
- Das Programm wird keine Eingabeüberprüfung vollziehen - Dateigröße und Format müssen vom Nutzer erkannt und angepasst werden.
- POSIX und GNU Guidelines werden für die Parameternamen der Kommandozeile eingehalten und angewendet.
- Für jeder Routine, welche für das Programm geschrieben wird, wird es einen Test geben, der deren Funktion überprüft
- Das Programm vergleicht Strings mitfhilde einer 5-Bit ASCII Kodierung, gibt die Strings aber so aus wie sie eingegeben worden sind.
- Alle Funktionen werden über Dokumentationsblöcke dokumentiert; dies beinhaltet Datentypen der Parameter und funktionsweise der Routine.

D. Codeauszüge

Listing 1: Pseudocode des zu implementierenden Algorithmus

```
// Setup
if |S1| < |S2|
        S <- S1
        T <- S2
else
        S <- S2
        T <- S1
end

for each s from S with |s| >= 5
        i <- intern(s)
        Q <- 5grams(s)
        for each q from Q
                R(q) <- R(q) u i
        end
end

## Run
while read t from T with |t| >= 5
        Q <- 5grams(t)
        W <- 1

        I <- call zero_interns
        for each q from Q
                for each r from R(q)
                        I(r) += W
                end
                W <- W * 2
        end

        J <- call zero_streaks
        max <- 0
        for each s from S with |s| >= 5
                i <- intern(s)
                J(i) <- longest_streak(I(i))
                if (J(i) > max)
                        max = J(i)
                        M <- {s}
                else if (J(i) == max)
                        M <- M u s
                end
        end

        ## Optional max check
        if (max < |t| / 2)
                continue
        end

        ## Output
        for each s from M:
                output s, t, max
        end
end
```

Listing 2: 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 Qgrams.
    :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++;
    }
}
```

Listing 3: 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 fi
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>
```