

PROJEKTDOKUMENTATION

AUSBILDUNG ZUM FACHINFORMATIKER F.
ANWENDUNGSENTWICKLUNG

Abschlussprüfung Winter 2017 IHK Berlin

Auszubildender:

Nils Diefenbach

Ident-Nummer:

3590244

Ausbilder:

Sebastian Freundt

GA FINANCIAL SOLUTIONS GMBH

8. September 2017

Inhaltsverzeichnis

1	Einleitung	5
1.1	Projektumfeld	5
1.2	Projektziel	5
1.3	Projektbegründung	5
1.4	Projektschnittstellen	6
1.5	Projektabgrenzung	6
2	Analysephase	7
2.1	Ist-Analyse	7
2.2	Wirtschaftlichkeitsanalyse	7
2.2.1	Make-or-Buy Entscheidung	7
2.2.2	Projektkosten	9
2.2.3	Amortisationsdauer	9
2.3	Nutzwertanalyse	9
2.4	Anwendungsfälle	9
2.5	Lastenheft / Fachkonzept	9
2.6	Zwischenstand	9
3	Entwurfsphase	10
3.1	Zielplattform	10
3.2	Algorithmusdesign	10
3.3	Architekturdesign	10
3.4	Entwurf der Benutzeroberfläche	12
3.5	Datenmodell	13
3.6	Geschäftslogik	13
3.7	Pflichtenheft / Datenverarbeitungskonzept	13
4	Implementierungsphase	14
4.1	Entwicklungsvorbereitung	14
4.2	Implementierung der Datenstruktur	14
4.2.1	HashMap Implementation	14
4.2.2	Dynamische Array Implementation	14
4.3	Implementierung der Benutzeroberfläche	14
5	Abnahmephase	15
5.1	Code-Flight mit der Abteilungsleitung	15
5.2	Präsentation des Programms	15
5.3	Zwischenstand	15
6	Einfuehrungsphase	16

7	Dokumentation	17
7.1	Nutzerdokumentation	17
7.2	Entwicklerdokumentation	17
8	Fazit	19
8.1	Soll- /Ist-Vergleich	19
8.2	Post Mortem	19
8.3	Ausblick	19
9	Anhang	20
9.1	Analysephase	20
9.1.1	Vergleich: DLD Aufwand vs QGram Aufwand mit Fuzzy Join .	20
9.1.2	Vergleich: Theoretischer DLD vs QGram Zeitvergleich	20
9.1.3	Lastenheft	20
9.1.4	Use-Case QGJoin	21
9.2	Entwurfsphase	21
9.2.1	Nutzwertanalyse: Programmiersprachen	21
9.2.2	Nutzwertanalyse: Datentyp der Menge Q	21
9.2.3	Pseudocode: Algorithmus	24
9.2.4	Aktivitätsdiagramm: Programmlogik	24
9.2.5	Analyse: Verteilung der Qgramme	24
9.2.6	Datenmodell	24
9.2.7	Pflichtenheft	24
9.3	Implementierungsphase	24
9.3.1	Quellcodeauszug: Dynamische Array	24
9.3.2	Quellcodeauszug: Hashtable Implementation	25
9.3.3	Auszug: Hilfeausgabe des Kommandozeilenprogramms	26
9.4	Dokumentationsphase	26
9.4.1	Auszug: Entwicklerdokumentation	26
9.4.2	Auszug: Endnutzerdokumentation	27

- **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

1 Einleitung

1.1 Projektumfeld

Die GA Financial Solutions GmbH ist ein 5-köpfiger Finanzdienstleister im Herzen Berlins. Die Firma spezialisiert sich auf computer-gestütztes Handeln von Futures, Aktien und anderen Derivaten. Direkte Kunden gibt es zur Zeit nicht; 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.

1.2 Projektziel

Ziel ist es, eine machinell erstellte Verknüpfung zwischen zwei Symbolen für Derivate von verschiedenen Anbietern zu bewerten und auf Basis einer Metrik mit erhöhter Trefferquote zu assoziieren.

1.3 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 500000 mögliche Herausgeber an ca. 1200 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-Datenbank¹ jedoch unter "パナソニック株式会社", was transskribiert wiederum „Panasonikku Kabushiki-gaisha“ 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 entsprechend Ressourcen entsprechend anderen Aufgaben zu geteilt werden können. Ein Faktor, der in einem kleinen Betrieb wie diesem essenziell ist.

¹Global Legal Entity Identifier System

1.4 Projektschnittstellen

Das Tool wird als Kommandozeilenprogramm, welches Daten per Pipe (") annimmt und diese im .csv Format über STDOUT ausgibt. Dies ist vor allem nützlich für die Prototyping phase, welche oft höchst-agile Einbindungen verschiedener Tools erfordert, ohne vorher viel Bootstrapping betreiben zu müssen. Die Nutzung dieses Tools findet hierbei vor Allem in der Abteilung Datenverarbeitung statt.

Als technische Schnittstelle ist vor Allem unser 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. Jedoch findet keine direkte Anbindung des zu entwickelnden Projekts an das Datensammelsystem statt. Es ist nur wichtig, dass das Programm .CSV Dateien unterstützt, welche vom System bereitgestellt werden.

Mittel und Genehmigung des Projekts stellt die Abteilung Datenverarbeitung, vertreten durch Sebastian Freundt, welchem auch am Ende des Projekts dieses präsentiert wird.

1.5 Projektabgrenzung

Während mehrere hundert-tausend Symbole miteinander verglichen und die effiziente Gestaltung dieser Vergleiche eine Priorität ist, so sind folgende Themen explizit nicht Teil der Aufgabe dieses Projektes:

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

2 Analysephase

2.1 Ist-Analyse

Die GA Financial Solutions holt sich täglich aktuelle Symbole von sechse verschiedenen Anbietern ab: Interactive Brokers[?], Google, Yahoo, Morningstar, Bloomberg und GLEIS. Diese Symbole ändern sich zu einem gewissen Prozentsatz täglich und im Schnitt kommen etwa 1.67 Millionen Instrumente hinzu.

Zur Zeit werden diese Symbol ad hoc mit Heuristiken und Handarbeit verglichen - dabei wird meist immer nur eine kleine Teilmenge verglichen, und die gefunden Ergebnisse wieder verworfen, da es sich meist um spezialisierte Werte handelt.

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

Es besteht ein theorethischer Lösungsansatz auf Basis der Damerau-Levenshtein Distanz²: er sieht vor, eine $M \times M$ 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, wie man erwarten kann, zu Speicherüberläufen. In Rechenaufrufen ausgedrückt bedeutet dass etwa $1.038.901.988.746.226^3$ DLD Aufrufe benötigt werden um die Daten miteinander zu vergleichen und die Datenbank zu erstellen, wobei das DLD-Matrix Verfahren etwa 1.340.000 Aufrufe pro Sekunde (CPU Zeit) schafft. Daraus ergibt sich eine Initial-Laufzeit von etwa 6.15 Jahren auf einem 4-Kern Rechner bei voller Nutzlast (dies exkludiert die täglich hinzukommenden Daten während der Berechnung). Ist diese Datenbank erstellt, werden durch die neu hinzukommenden Daten circa 16 Milliarden DLD Aufrufe täglich benötigt um die Datenbank zu aktualisieren. Dies entspricht weiteren 33 Minuten Rechenzeit auf einem 4-Kerner pro Tag.

Gewünscht ist eine performantere, ressourcen-schonendere Implementierung des obigen Prozesses, sodass die Daten auf einem Firmeninternen 4-Kerner zeitnah ausgerechnet werden können. Hierbei ist es nicht erforderlich die, dem derzeitigen Prozess zugrunde liegende, Damerau-Levenshtein Distanz zu verwenden. Zudem soll erzieht werden, dass die menschliche Komponente aus dem Vergleichsverfahren eliminiert wird, somit also keine Heuristiken zum Symbol-Vergleich benötigt werden.

2.2 Wirtschaftlichkeitsanalyse

2.2.1 Make-or-Buy Entscheidung

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

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

³Eine Billiarde achtunddreißig Billionen neunhundertein Milliarden neunhundertachtundachtzig Millionen siebenhundertsechszvierzigtausendzweihundertsechszundzwanzig

blem löst, ist das Open-source Projekt fuzzy-join⁴. Jedoch vergleicht dieses Tool Daten nur in eine Richtung (One-To-Many), während unser Use-Case ein Bi-direktionales Verfahren benötigt (Many-To-Many).

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

blieb nur die Ausweichung auf Alternativen, zum Beispiel die Auslagerung in einen Cloud-Dienst. Dies wäre jedoch nach unseren Erfahrungen mit dieser Art Berechnung nicht kompatibel und Anpassungen des DLD Verfahrens nötig gewesen. Die vollständige Liste der Alternativen lautet wie folgt:

- **Projektkosten**
Größtes Kriterium bei der Entwicklung des Projektes waren logischerweise die potentiellen Entwicklungskosten. Diese wurden anhand Branchentypischer Stundenlöhne (im Falle der externen Entwicklung), bzw. der Preislisten der Cloud Computing Anbieter unseres Vertrauens (MassiveGrid, Amazon Web Services) berechnet und verglichen.
- **Nachhaltigkeit**
Inwiefern die Lösung weitere Integration mit anderen Services und unserer Toolchain erlaubt, wurde mit diesem Kriterium bewertet.
- **Flexibilität**
Bei der Untersuchung dieses Kriteriums lag das Augenmerk vor Allem auf der Möglichkeit der Anpassung der Software und der Ergebnisse während der Entwicklung, sowie nach Abschluss der Entwicklung (z.B. über Parameter).
- **Unterhaltskosten**
Dieses Kriterium beschreibt die laufenden Kosten nachdem das Projekt umgesetzt worden ist.

2.2.2 Projektkosten

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 €

⁴<https://github.com/dgrtwo/fuzzyjoin>

2.2.3 Amortisationsdauer

2.3 Nutzwertanalyse

2.4 Anwendungsfälle

Ein ungefähre Übersicht der Anwendungsfälle des Programms wurde mit Hilfe eines Use-Case-Diagramms im zuge der Analysephase erstellt. Sie befindet sich im Anhang auf Seite X und lässt alle benötigten Funktionen aus Endanwendersicht erkennen.

2.5 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.

2.6 Zwischenstand

3 Entwurfsphase

3.1 Zielplattform

Wie aus der Sektion Projektziel entnommen werden kann, soll das Projekt als Kommandozeilenprogramm implementiert werden. Aufgrund der Anforderungen aus dem Lastenheft, welches Speicherperformanz sowie Geschwindigkeit fordert, fiel die Entscheidung der Programmiersprache recht schnell. C ist eine Hochsprache, welche effizient und extrem schnell die geforderten Berechnungen ausführen kann, und dem programmierer ein ideales Interface bietet, effizient im Speicher zu agieren. Um jedoch persönliche Erfahrungen und Ansichten zu verifizieren, wurden diverse weitere Sprachen anhand einer Nutzwertanalyse verglichen. Die wichtigste Rolle spielten hierbei folgende Kriterien:

- Performanz - Geschwindigkeit der Sprache in der HPC Domäne
- Speichereffizienz - Der Fußabdruck der Sprache, sowie deren Objekte im Speicher soll möglichst gering sein
- Dokumentation - Bibliotheken und Module sollten 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, zu finden im Anhang auf Seite X, präsentiert.

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

3.2 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. Dieser wurde vom Autor in Form eines Aktivitätsdiagramms, einsehbar auf Seite X im Anhang, visualisiert. Der Pseudo code ist ebenfalls auf Seite X im Anhang einsehbar. Hierbei wurden sogenannte QGramme verwendet - diese verstehen sich als Teilstring eines Strings der Länge Q.

3.3 Architekturdesign

Das Projekt besteht aus 2 Header-Dateien und einer main.c. Die Header Dateien beinhalten zum einen die Algorithmus-spezifischen Funktionen zum Vergleich von Strings, zum anderen eine open-source Hashmap implementation.

SCREENSHOT OF DIRECTORY LAYOUT

Auf programmatischer Ebene wurde das Programm in zwei Abschnitte unterteilt:

- **Vorbereitung**
Hier werden Daten aus der ersten Datei geladen und benötigte Datenstrukturen im Speicher angefordert. Variablen werden initialisiert und der QGram Pool erstellt.
- **Ausführung**
Das zweite Set an Daten wird über einen File Pointer referenziert und die Strings werden Zeile für Zeile ausgelesen, ihre QGramme mit denen im Schritt Vorbereitung erstellten verglichen und bei einer genügenden Übereinstimmung über STDOUT ausgegeben.

Hieraus ergaben sich 8 Sub-komponenten, welche einer Implementierung bedurften:

- **Vorbereitung:**
 - **S:** Eine Menge aller Strings aus der ersten Liste bzw Datei
 - **Q:** Eine Menge aller QGramme, welche sich aus denjeweils einzelnen Strings in S generieren lassen
- **Ausführung**
 - **I:** Eine Menge, welche die gleiche Größe wie S ist und die Anzahl der Qgramme zählt, welche für den jeweiligen Index eines Strings in S passen.
 - **f:** Eine Referenz zur zweiten Liste, von der wir zur vergleichende Strings holen können
 - **T:** Eine Menge der Qgramme in der zur Zeit ausgelesenen Zeile
 - **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 - prinzipiell besteht hier zwar ein höherer Speicheraufwand, dieser ist jedoch abschätzbar, da die maximale Anzahl an möglichen Key-Value Einträgen begrenzt ist. Der darausfolgende $O(1)$ Suchaufwand zum Vergleich von Qgramme ist ein unschätzbarer Vorteil in der späteren Ausführung. Auch hier wurde wiederum eine Nutzwertanalyse durchgeführt, um sicherzustellen dass die persönlichen Annahmen des Autors mit den technischen Gegebenheiten übereinstimmen. Hierzu wurden verschiedene Datenstrukturen nach folgenden Kriterien bewertet:

1. Speicheraufwand

Da laut Vorgabe maximal 32GB zur Ausführung des Programms zur Verfügung stehen, sollte der Speicherverbrauch natürlich so gering wie möglich sein. Da jedoch die maximale Datengröße aufgrund der 7-Bit ASCII Kodierung der Strings bekannt ist, ist dieses Kriterium am geringsten gewichtet worden.

2. Schnelligkeit Einfügen-Befehl

Das Erstellen der Datenstruktur sollte der größte rechnerische Aufwand im ersten Schritt des Programms sein. Deshalb sollte hier möglichst effizient gearbeitet und der Aufwand entsprechend optimiert sein.

3. Schnelligkeit Such-Befehl

Einen Wert aus der Datenstruktur abzufragen sollte nach Möglichkeit in Richtung von Aufwand $O(1)$ gehen, da dies mehrfach während des Stringvergleichs gebraucht wird. Deshalb ist dieses Kriterium auch am höchsten Gewichtet worden, da hier die größten Engpässe entstehen können.

Für M wird eine dynamisch wachsende Array benötigt, da die Anzahl der passenden QGramme je nach String variiert. Da die Anzahl der möglichen Zeichen pro QGramm von unserem verwendeten Zeichensatz begrenzt ist (7-Bit ASCII), konnte eine begründete Annahme über die Verteilung der QGramme in der Menge der Strings gemacht werden. Anhand dieser Analyse konnte eine optimale Startgröße festgelegt werden, sodass in etwa der Hälfte der Fälle erwartet werden kann, dass das Array nicht vergrößert werden muss, und somit die Speicherzuweisungen minimiert werden.

Eine größere Ansicht des Graphen befindet sich im Anhang auf Seite X.

In Einklang mit der test-getriebenen Entwicklung wurde von Anfang an ein 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 selbsterstellten Datentypen und dazugehörige Funktionen ebenfalls zunächst ein Test geschrieben.

3.4 Entwurf der Benutzeroberfläche

Der GNU Codingstandard⁵, sowie die POSIX Utility Guidelines⁶ 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 auch entwickelt. Ein Auszug dieser Vorgaben befindet sich im Anhang auf Seite X.

Auf eine grafische Nutzeroberfläche wird nicht gewünscht, da die bevorzugte Oberfläche der Endnutzer die Kommandozeile ist.

⁵<https://www.gnu.org/prep/standards/standards.html#User-Interfaces>

⁶http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap12.html

3.5 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. Ein Modell eines Strings sieht dann wie folgt aus:

aus diesen Gründen wird auf eine weitere Ausführung des Modells verzichtet.

Es ist zu klärifizieren, dass die GA Financial Solution kaum bis gar nicht Datenbanksysteme wie MySQL, MariaDB u.Ä. nutzt. Stattdessen werden komprimierte csv Dateien verwendet, welche meist nur aus 3 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.

3.6 Geschäftslogik

Die bisherige Geschäftslogik wird auf Seite X im Anhang dargestellt. Wie dort zu erkennen ist, ist die Abholung der Daten von Anbietern automatisiert, die Verarbeitung jedoch nicht. Diese geschieht bei Bedarf und erfordert u.A. die Vorsortierung der Daten in handhabbare Datensätze, welche einfacher und in angemessener Zeit von unseren 4 Kern Maschinen berechnet werden können. Zudem kommt noch eine erforderliche manuelle Überprüfung der Ergebnisse. Die Darausentstehenden Daten werden schließlich für das derzeitig entwickelte Projekt genutzt und schlussendlich wieder verworfen, da die Datensätze hoch parametrisiert sind, und nicht zur generellen Nutzung innerhalb der Firma eignen.

Im Anhang auf Seite X ist ein Komponentendiagramm vorzufinden, welche die unterschiedlichen Komponenten innerhalb der Geschäftslogik darstellt.

3.7 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, und erläutert die Lösungsansätze und die benötigten Mittel um das Projekt zu realisieren.

4 Implementierungsphase

4.1 Entwicklungsvorbereitung

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

4.2 Implementierung der Datenstruktur

Wie bereits in Sektion X erwähnt, werden zwei spezielle Datenstrukturen für das Programm benötigt - Hashmaps und dynamisch wachsende Arrays.

4.2.1 Hashmap Implementation

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

Der Code wurde weitestgehend verbatim übernommen, mit der Ausnahme des Containertyps, welches die Hashmap speichert. Da ein QGram in mehreren Strings vorkommen kann, musste der Containertyp von einem Integer zu einer dynamisch wachsenden Array abgewandelt werden. Da letztere ohnehin entwickelt werden muss, war die Komplexität dieser Anforderung gering.

4.2.2 Dynamische Array Implementation

C besitzt von Haus heraus nur simple Datentypen, welche dem Nutzer erlauben exponentiell komplexe Datenstrukturen zu erschaffen. Da auch eine dynamische wachsende Array nicht in ANSI-C zur Verfügung steht, musste eine Datenstruktur für diese erstellt werden. Das Konzept hierbei ist recht einfach - sollte die Array voll sein, so wird beim nächsten Aufruf der Insert Funktion die Länge der Array verdoppelt. Hierfür wurde eine neue Datenstruktur angelegt und 3 Funktionen definiert, um mit der Datenstruktur zu interagieren. Ein Auszug des Quellcodes dieser Datenstruktur ist auf Seite X im Anhang vorzufinden.

4.3 Implementierung der Benutzeroberfläche

5 Abnahmephase

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

Der Hauptaugenmerk und die Vorgehensweise war in beiden Fällen jedoch unterschiedlich und wird in den folgenden Sektionen genauer erläutert.

5.1 Code-Flight mit der Abteilungsleitung

Um die Qualität des Codes zu gewährleisten wurde anstatt einer stillen Abnahme, bei dem das Projekt übergeben, vom Auftraggeber überprüft und entweder abgenommen oder bemängelt wird, wurde für das Projekt die Abnahme über einen Code-Flight⁷ 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 die Begründung für die Nutzung dieser.
- Limitation des Algorithmus und des CLI Programms.
- Die nächsten, möglichen Schritte zur Verbesserung der Hauptprogramms.
- Mögliche Ansätze zur Verbesserung des Algorithmus.

5.2 Präsentation des Programms

Nachdem der Code-Flight mit dem Abteilungsleiter abgeschlossen wurde, 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.

5.3 Zwischenstand

⁷zu deutsch Programm-Flug"

6 Einfuehrungsphase

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, über den Aufruf des Programms mit dem Parameter `h` bzw. `-help`, statt. Dieser Aufruf erzeugt eine kurze Beschreibung des Programms und dessen Parameter.

7.2 Entwicklerdokumentation

Die Entwicklerdokumentation ist etwas weitreichender gestaltet, jedoch bewusst minimalistisch gehalten. Die Dokumentation des Codes findet über einen Dokumentationsblock über der Signatur jeder Funktion statt. Dieser Block beinhaltet eine kurze Beschreibung, sowie die Namen und erwartete Datentypen der Funktionsparameter.

=

8 Fazit

läuft.

8.1 Soll- /Ist-Vergleich

Jeschacht: 80 Soll: 100

8.2 Post Mortem

War ganz gut, alles.

8.3 Ausblick

Weitere Schritte:

Python einbindung

Optimierung des Algorithmusfilters

Optimierung des Rechenaufwands des Algorithmus

9 Anhang

9.1 Analysephase

9.1.1 Vergleich: DLD Aufwand vs QGram Aufwand mit Fuzzy Join

9.1.2 Vergleich: Theoretischer DLD vs QGram Zeitvergleich

9.1.3 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 Kommandozeile 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. 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 Zielplattform 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.

9.1.4 Nutzwertanalyse: Projektentwicklung

Tabelle 2: Nutzwertanalyse Projektentwicklung

Kriterien	Gewicht	Externe Entwicklung Bewertung	Gesamt	Interne Entwicklung Bewertung	Gesamt
Projektkosten	45.00%	4	1.8	5	2.25
Nachhaltigkeit	15.00%	4	0.6	4	0.6
Flexibilität	5.00%	3	0.15	5	0.25
Unterhaltskosten	35.00%	4	1.4	4	1.4
Gesamt	100.00%		3.95		4.5

9.1.5 Use-Case QGJoin

9.2 Entwurfsphase

9.2.1 Nutzwertanalyse: Programmiersprachen

Tabelle 3: My caption

Kriterien	Gewicht	C Bewertung	C Gesamt	C++ Bewertung	C++ Gesamt	Bewe
Performanz	30.00%	5	1.5	5	1.5	4
Speichereffizienz	30.00%	5	1.5	5	1.5	3
Dokumentation	20.00%	5	1	4	0.8	4
Systemische Voraussetzungen	10.00%	5	0.5	5	0.5	3
Datenstrukturen	10.00%	3	0.3	4	0.4	5
Gesamt	100.00%		4.8		4.7	

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

9.2.2 Nutzwertanalyse: Datentyp der Menge Q

9.2.3 Pseudocode: Algorithmus

9.2.4 Aktivitätsdiagramm: Programmlogik

Abbildung 1: Aktivitätsdiagramm der Programmlogik

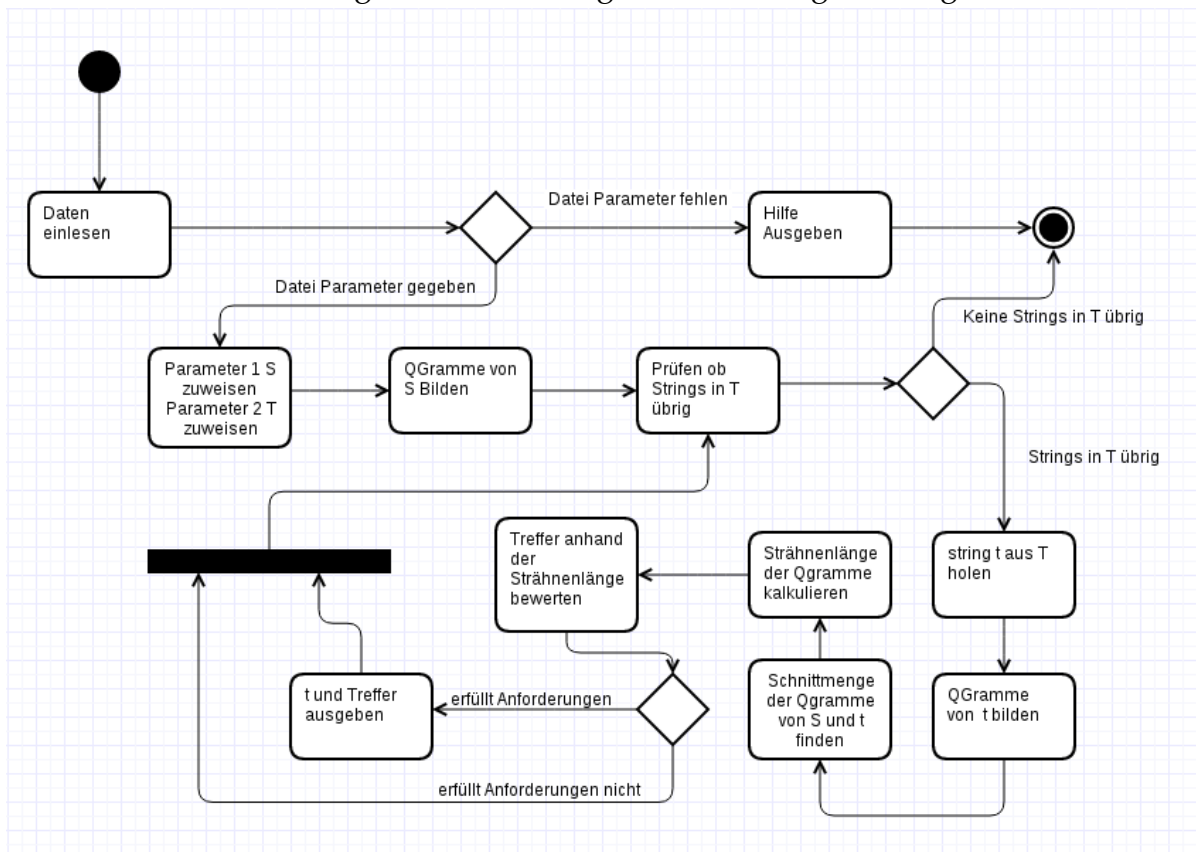


Abbildung 2: Verteilung der QGramme im zu bearbeitenden Datensatz

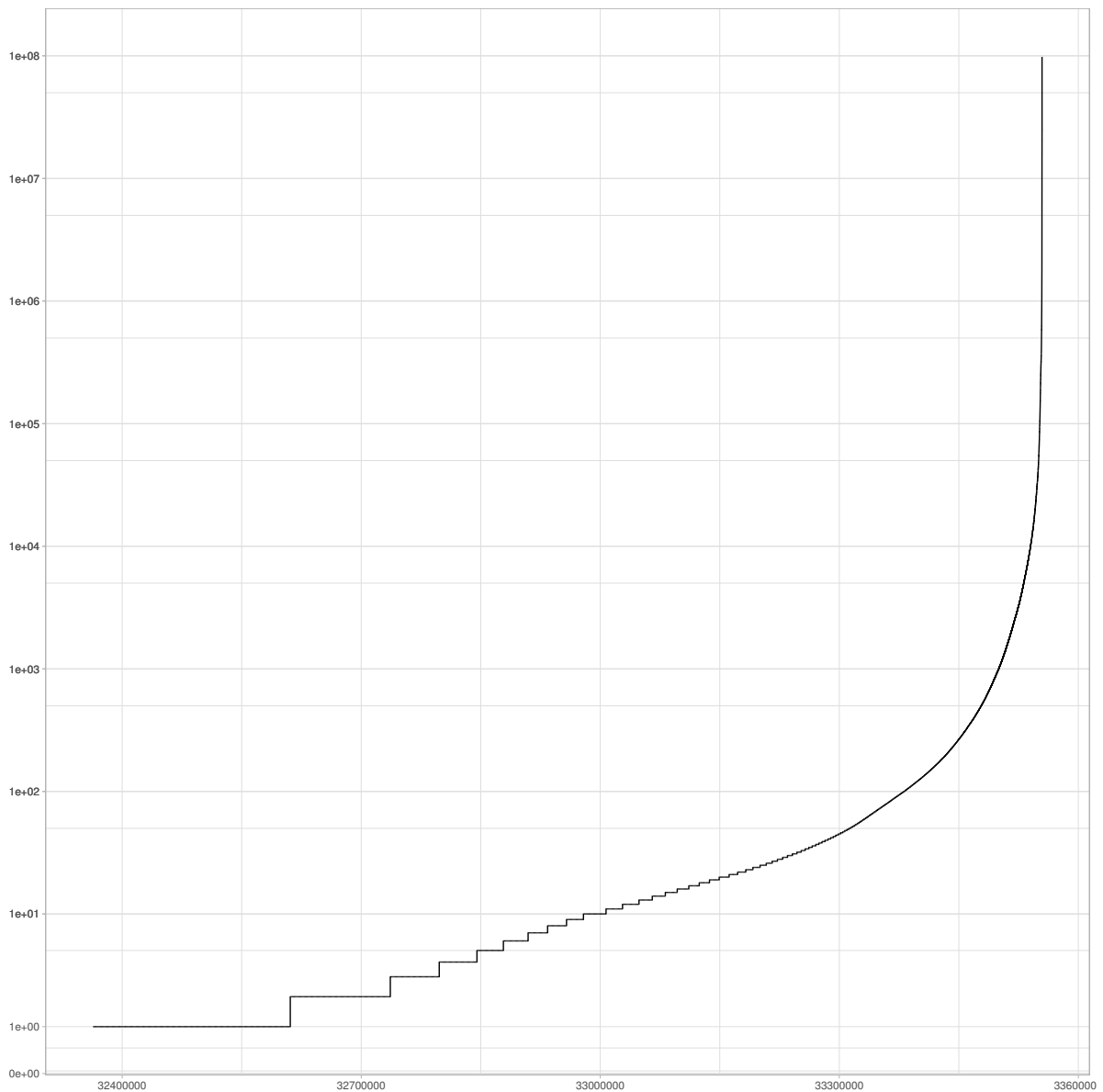
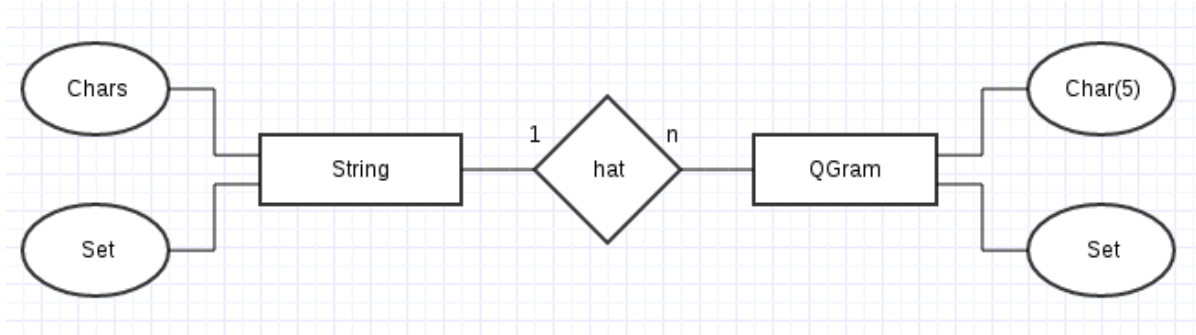


Abbildung 3: UML Darstellung des vorhandenen Datenmodells



9.2.5 Analyse: Verteilung der Qgramme

9.2.6 Datenmodell

9.2.7 Pflichtenheft

9.3 Implementierungsphase

9.3.1 Quellcodeauszug: Dynamische Array

Listing 1: 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;  
}
```

9.3.2 Quellcodeauszug: Hashtable Implementation

Listing 2: 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, key) == 0))
        return curr;

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

return MAP_FULL;
}

```

9.3.3 Auszug: Hilfeausgabe des Kommandozeilenprogramms

9.4 Dokumentationsphase

9.4.1 Auszug: Entwicklerdokumentation

Listing 3: 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++;  
    }  
}
```

9.4.2 Auszug: Endnutzerdokumentation