FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Prof. Dipl.-Inf. Ingrid Scholl

Dipl.-Ing. Norbert Kutscher

Michel Erbach, B.Sc.

Kathrin Petters, B.Sc.

Marcel Stüttgen, M.Eng.

Algorithmen und Datenstrukturen Praktikum

Studiengänge Informatik und Wirtschaftsinformatik

SS 2018

Inhaltsverzeichnis

0 Einleitung

Willkommen zum Praktikum „Algorithmen und Datenstrukturen“. In diesem Praktikum werden eine Auswahl der theoretisch vermittelten Datenstrukturen und einige Algorithmen aus der Vorlesung mit der Programmiersprache C++ implementiert. Insgesamt werden fünf Praktika zu den folgenden Themen gestellt:

1. Verkettete Listen
2. Ringpuffer und Binärer Suchbaum
3. Performanz-Analyse von Sortierverfahren
4. Algorithmen zu Graphen
5. Verlustfreie Kompression

Jede Praktikumsaufgabe ist sehr umfangreich, so dass diese in Heimarbeit von Ihnen vorbereitet werden muss. Falls Sie während ihrer Programmierung zu Hause Fragen haben sollten, stehen Ihnen folgende Möglichkeiten ohne Wertung der Reihenfolge offen:

- Tutoren und Mitarbeiter während der anderen Praktikatermine aufsuchen oder zu Beginn des eigenen Praktikumtermins fragen,
- Lerngruppen bilden (wir sind gerne dabei behilflich, Gruppen zu vernetzen),
- Fragen im ILIAS-Forum stellen und diskutieren,
- Kommilitonen fragen (selbst vernetzt oder via Forum),
- die Sprechstunde von Frau Prof. Scholl nutzen (Mittwoch von 11:00 - 12:00 Uhr, Raum G303).

Wir wünschen Ihnen ein erfolgreiches Semester und viel Spaß mit „Algorithmen und Datenstrukturen“!

0.1 Team

Für die Betreuung des Praktikums stehen Ihnen während des Semesters folgende Ansprechpartner zur Verfügung:

- Professorin

Prof. Ingrid Scholl

scholl@fh-aachen.de
+49 241 6009 52177
Raum G 303
Sprechstunde: Mittwoch 11:00-12:00 Uhr, Raum G303

- Mitarbeiter

Dipl.-Ing. Norbert Kutscher

kutscher@fh-aachen.de
+49 241 6009 52174
Raum E 151
Sprechstunde: nach Vereinbarung

Kathrin Petters, B.Sc.

petters@fh-aachen.de
+49 241 6009 52241
Raum G 309
Sprechstunde: nach Vereinbarung

Michel Erbach, B.Sc.

erbach@fh-aachen.de
+49 241 6009 52100
Raum H 215
Sprechstunde: nach Vereinbarung

Marcel Stüttgen, M.Eng.

stuettgen@fh-aachen.de
+49 241 6009 52206
Raum G 301
Sprechstunde: nach Vereinbarung

Zudem werden wir von einigen Studierenden als Tutoren unterstützt, die Ihnen selbstverständlich auch zur Verfügung stehen.

0.2 Zeitplan und Termine

Die Tabelle 1 zeigt alle Termine der jeweiligen Gruppen inklusive der Raumzuordnung. Beachten Sie bitte, dass aufgrund der Feiertage im Mai/Juni zwei Freitagstermine auf **Dienstag 15:45-18:45 Uhr** in den **Räumen E141, E143, E145 und E146** verschoben wurden.

	1a-e	2a-d	3a-e	4a-d
Raum	1a: F104 1b: F106 1c: F111 1d: E141 1e: E143	2a: F104 2b: F106 2c: F111 2d: E141	3a: F104 3b: F106 3c: F111 3d: E145 3e: E143	4a: F104 4b: F106 4c: F111 4d: E145
P1	17.04.	20.04.	24.04.	27.04.
P2	08.05.	04.05.	15.05.	18.05.
P3	22.05.	25.05.	05.06.	08.06.
P4	12.06.	15.06.	19.06.	22.06.
P5	26.06.	29.06.	03.07.	06.07.
Ersatztermin	10.07.	13.07.	10.07.	13.07.

Tabelle 1: Terminplanung

ACHTUNG: Durch die vielen Feiertage im Sommersemester kommt es zwei mal zu einem dreiwöchigen anstellen vom sonst üblichen 14-tägigen Rhythmus.

0.3 „Spielregeln“

- Die Zuteilung zu den Räumen wird nach Ende des Anmeldeverfahren durch einen Mitarbeiter vorgenommen.
- Die Bearbeitung der Praktikumsaufgaben erfolgt in 2er-Gruppen, im Sonderfall, z.B. bei ungerader Anzahl an Studierenden innerhalb einer Praktikumsgruppe, auch in einer 3er-Gruppe.
- Um ein Testat für eine Praktikumsaufgabe zu erhalten, sind Sie in der Lage...
  - ... jede Zeile des Codes zu erklären. Lösungen, die nicht erklärt werden können, werden nicht akzeptiert.
  - ... die verwendeten Datenstrukturen und Algorithmen auf Papier mit neuen Testdaten zu skizzieren und Laufzeitkomplexitäten zu berechnen und zu begründen.
  - ... vor Ort das Programm so anzupassen, dass es mit neuen Testdaten on-the-fly getestet werden kann.
  - ... den Debugger sinnvoll einzusetzen und die Speicherbelegung der erzeugten Daten nachzuvollziehen.
  - ... pünktlich zu Beginn des Praktikumtermins zu erscheinen.
  - ... eine Entwicklungsumgebung Ihrer Wahl (z.B. VisualStudio oder Eclipse für Windows-Plattformen, Qt-Creator oder makefiles für Linux-Plattformen, o.Ä.) einzusetzen und ein Projekt 'ADS-Praktikum' zuzüglich Unterprojekten pro Aufgabe zu erstellen.
  - ... eine im Optimalfall vollständig funktionierende Lösung einen Tag vor dem Praktikumstermin per Ilias hochzuladen. Sollten noch kleinere Fehler vorhanden sein, können diese während des Praktikumtermins mit unserer Unterstützung behoben werden. Sie müssen nicht zwangsweise jede Aufgabe einzeln hochladen, sie können z.B. auch Ihre Quelldateien zippen und dann fortlaufend aktualisieren.
- Gelbe/Rote Karten - Regel: Jeder Teilnehmer darf sich im Laufe des Semester ein „Foul“ erlauben (z.B. unentschuldigtes Fehlen, Aufgabe nicht rechtzeitig fertig, etc), welches dann mit der gelben Karte geahndet wird. Jedes weitere „Foul“ führt zur roten Karte und damit zum Platzverweis (Ausschluss aus dem Praktikum). Ein unentschuldigter Fehltermin muss nachgeholt werden.
- Plagiat-Regel: Das Kopieren von nicht selbst programmierten Source-Code (auch in Subteilen) ist nicht erlaubt, denn **Kopieren ist nicht gleich Kapieren!** Ein offensichtlicher Täuschungsversuch führt zum Ausschluss aus dem Praktikum.
- Für alle Praktikumstermine Ihrer Gruppe besteht grundsätzlich **Anwesenheitspflicht** (Ausnahme: s.u.). Sollten Sie verhindert sein, z.B. aus schwerwiegenden Gründen oder Krankheit, so ist der zuständige Praktikumsbetreuer **vorher**, z.B. per Email, darüber in Kenntnis zu setzen. Im Krankheitsfall ist zusätzlich ein ärztliches Attest einzureichen bzw. nachzuzeigen. Sollte dies

nicht erfolgen, handelt es sich um ein unentschuldigtes Fehlen (Ausnahmen: z.B. Schneechaos bei Bus/ Bahn). „Rückstände“ in der Bearbeitung der Aufgaben sind selbstständig bis zum nächsten Termin aufzuarbeiten und zu Begin des Praktikumtermins vorzuzeigen!

- Sie möchten vorarbeiten? Gerne! Fleiß und Einsatz sollen belohnt werden! Sie können Aufgaben im Voraus bearbeiten und diese dann während des regulären Praktikums ihren Betreuern vorführen. Bitte haben Sie jedoch Verständnis, dass diese angehalten sind, zunächst alle „regulären“ Aufgaben zu testieren, so dass Sie ggf. etwas warten müssen, bis sich ein geeignetes Zeitfenster findet. Sollten Sie vorweg ein Testate erfolgreich erhalten, befreit Sie dies von der Anwesenheitpflicht für den entsprechenden regulären Termin.

Nach all diesen vielen Erstinformationen wünschen wir Ihnen ein erfolgreiches Semester und viel Spaß mit „Algorithmen und Datenstrukturen”!

1 Praktikum 1 : Erstellen Sie ein C++ Programm zur Speicherung von Tweets

Sie erstellen ein kleines Twitter-Programm. In diesem Programm lesen Sie kleine Tweets von der Tastatur ein. Diese werden mit Datum und Uhrzeit in einer verketteten Liste gespeichert. Alle bisherigen Tweet-Nachrichten sollen nach der Eingabe angezeigt werden.
In der Aufgabenstellung erhalten Sie Lösungshinweise und Testbeispiele. Vervollständigen Sie die gegebenen Programmteile und fügen Sie weitere Tests hinzu. Gehen Sie unbedingt in der Reihenfolge der Aufgabenstellung vor.
Die Aufgabenstellung gliedert sich in drei große Arbeitsschritte. Zu jedem Arbeitsschritt legen Sie bitte ein Projekt an, damit Sie die einzelnen Entwicklungsschritte im Praktikum erklären und zeigen können.

1.1 Aufgabenstellung

1. Der erste Teil der Aufgabe besteht darin, dass Sie eine Klasse bereit stellen, die die Datenstruktur einer dynamisch, doppelt verketteten Liste zur Verfügung stellt. Um die Aufgabe zunächst zu vereinfachen gehen wir von der Speicherung von integer-Werten (als key) aus.
   Zur Speicherung der Daten benötigen wir eine Klasse Node. Die Klasse List dient zur Verwaltung der Datenstruktur und die Klasse Node zu Speicherung der Inhalte.
   Wir benötigen folgende Dateien:

Node.h	Headerdatei der Klasse Node
Node.cpp	c++ Quelle der Klasse Node
List.h	Headerdatei der Klasse List
List.cpp	c++ Quelle der Klasse List
main.cpp	Testprogramm für die Klasse List

   Diese Dateien liegen als Download auf dem ILIAS-Server (Aufgabe1a - zip-archivierte Dateien). Kopieren Sie die Dateien in ein neues C++ Projekt. Verändern Sie **nicht** die Headerdateien und auch nicht die Datei "Node.cpp". Fügen Sie den fehlenden Code in die Quelldateien ein (List.cpp oder main.cpp).

   In der C++ Quelle der Klasse List sind einige Methoden nur allgemein sprachlich in der zu erfüllenden Funktion beschrieben. Erzeugen Sie den notwendigen C++ Quellcode.
   Die main.cpp enthält ein Testprogramm mit dem die Methoden der Klasse geprüft werden können. Dort können Sie auch weitere Testbeispiele einfügen.

2. Der zweite Teil der Aufgabe besteht darin, dass Sie die Klasse List und die Klasse Node je zu einer Template Klasse umbauen.
   Gehen Sie wie folgt vor:

Kopieren Sie alle Dateien in ein neues Projekt. Anschließend wird der Programmteil von Node.cpp in die Datei Node.h kopiert (unterhalb der Klassendefinition). Anschließend kopieren Sie den Programmteil von List.cpp in List.h und löschen dann die Dateien Node.cpp und List.cpp.

Das Programm ist nach diesem Umbau lauffähig!

Jetzt bauen Sie beide Klassen zu Template Klassen um.
Die überladenen Operatoren werden wie normale Methoden zu Template-Methoden umgesetzt. Eine Besonderheit sind die friend-Operatoren. Hier ist eine zusätzliche Template-Deklarierung notwendig. Aus diesem Grund sind diese Teile des Quellcodes nochmals vorgegeben (Downloadbereich ILIAS-Server: Aufgabe1b).

3. Der letzte Teil der Aufgabe besteht nun darin, die ursprüngliche Aufgabe zu realisieren. Kopieren Sie dazu wieder alle Datein vom 2. Projekt in ein neues Projekt außer die Datei main.cpp. Dafür fügen Sie die Datei tweet.h und main.cpp ein (Downloadbereich ILIAS-Server: Aufgabe1c).

2 Praktikum 2: Ringpuffer und Binärbaum

2.1 Aufgabenstellung

Herzlichen Glückwunsch. Sie wurden als Systemarchitekt bei „Data Fuse Inc.“, einem führenden Anbieter von BigData- und Swarm-Informationslösungen, eingestellt. In der Entwicklungsabteilung „advanced distributed systems“ sind Sie nun für die Konzeption und Realisierung von experimentellen Lösung zuständig. Zeigen Sie, dass Sie kreative Lösungen finden und Algorithmen in ein neues Zusammenspiel bringen können.

- Teil 1 - Backup-Rotation mittels Ringpuffer
- Teil 2 - Binärbaum - Datenhaltung und Operation
- Teil 3 - Backup-Rotation mit Ringpuffer und Binärbäumen (Zusatzaufgabe)

2.2 Backup mittels Ringpuffer

Die Erstellung von Backups im industriellen Umfeld ist äußerst komplex und eine Sicherung kann schnell mehrere Terabyte umfassen. Eine aufwändige Langzeitarchivierung ist jedoch nicht immer erforderlich, da die Änderungsrate der Daten zu hoch und damit zu teuer ist. Man bedient sich hier des Tricks der Ring-Sicherung, bei der nur ein begrenzter Horizont von Backupzyklen vorgehalten wird und ältere Sicherungen gelöscht oder überschrieben werden.
Schreiben Sie ein Programm zur Erstellung und Verwaltung von Backups mittels Ringpuffer. Erstellen Sie hierfür eine Hauptklasse **Ringpuffer**, die die Verwaltung des Rings und damit der verknüpften Backups ermöglicht. Diese Klasse realisiert alle erforderlichen Operationen auf der Datenstruktur (z.B. neue Elemente hinzufügen, Inhalte ausgeben, Suchen, usw.). Der eigentliche Ring besteht aus insgesamt 6 Knoten der Klasse **Ringnode**, die untereinander ähnlich einer verketten Liste verknüpft sind. Um die Alterung eines Datensatzes zu simulieren/kennzeichnen (siehe Bild 1), besitzt jeder Ringnode das Attribut OldAge (Aktuellste: ’0’, der Älteste ’5’). Mit der Speicherung eines neuen Ringnodes ersetzen Sie immer den Ältesten. Verwenden Sie die im Bild (1) vorgegebenen Klassen und erweitern Sie diese bei Bedarf. *Beachten Sie bitte die Lösungshinweise unten, da diese weitere Details zur Implementierung geben!*
Der Benutzer soll über die Konsole folgende Möglichkeiten haben:

- Neuen Datensatz eingeben. Dieser besteht aus Daten für das Backup (SymbolicData:string) sowie einer Beschreibung (Description:string)
- Suchen nach Backup-Daten. Auf der Konsole sollen die Informationen OldAge, Beschreibungs- und Datentext des betreffenden Nodes ausgegeben werden. Sonst eine Fehlermeldung.
- Alle Backup-Informationen ausgeben. Liste aller Backups, Format siehe Beispiel

```
                                // Beispiel: Menü der Anwendung
==================================
 OneRingToRuleThemAll v0.1, by Sauron Schmidt
==================================
1) Backup einfuegen
2) Backup suchen
3) Alle Backups ausgeben
?>

--------------------------------------------------------------------------------
?> 1                            // Beispiel: neuer Datensatz
+Neuen Datensatz einfuegen
Beschreibung ?>  erstes Backup
Daten ?> echtWichtig1
+Ihr Datensatz wurde hinzugefuegt.

--------------------------------------------------------------------------------
?> 2                            // Beispiel: suche Datensatz
+Nach welchen Daten soll gesucht werden?
?> echtWichtig1
+ Gefunden in Backup: OldAge 0, Beschreibung: erstes Backup, Daten: echtWichtig1

?> 2                            // Beispiel 2: suche Datensatz
+Nach welchen Daten soll gesucht werden?
?> megaWichtig1
+ Datensatz konnte nicht gefunden werden.

--------------------------------------------------------------------------------
?> 3      // Beispiel: Ausgabe aller Backups nachdem weitere Daten eingegeben wurden
OldAge: 0, Descr: sechstes Backup, Data: 0118999
--------------------------
OldAge: 1, Descr: fuenftes Backup, Data: 1337
--------------------------
OldAge: 2, Descr: viertes Backup, Data: 007
--------------------------
OldAge: 3, Descr: drittes Backup, Data: 789
--------------------------
OldAge: 4, Descr: zweites Backup, Data: 456
--------------------------
OldAge: 5, Descr: erstes Backup, Data: echtWichtig1
```

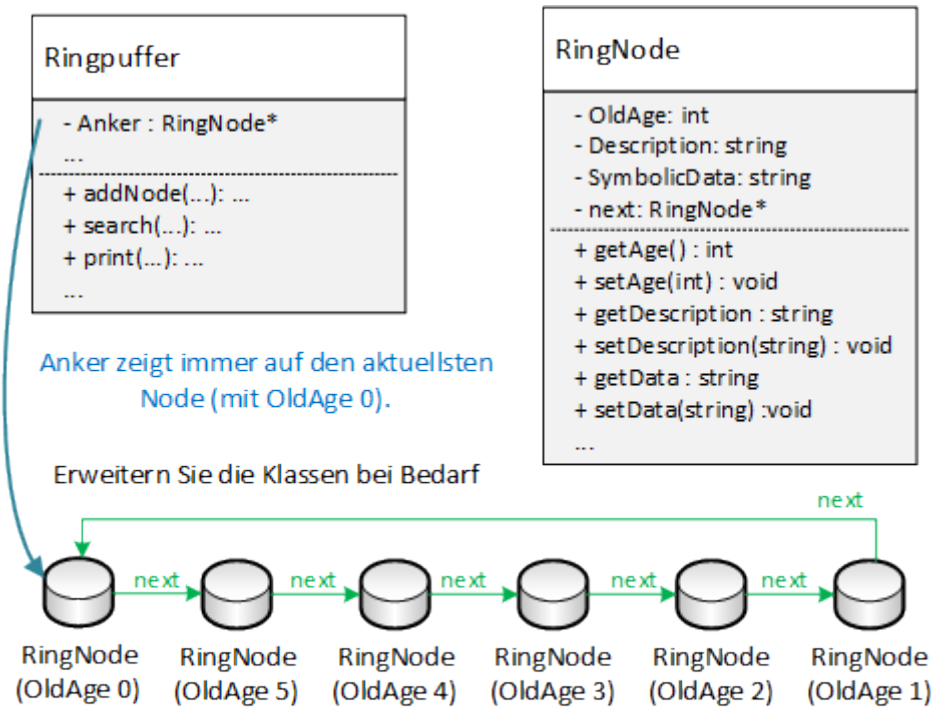


Abbildung 1: Datensicherung mittels Ringpuffer

2.2.1 Lösungshinweise

- Nutzen Sie Ihr Wissen über die verkettet Liste, da sich die Datenstrukturen stark ähneln.
- Achten Sie darauf, dass Sie die Operationen in der richtigen Reihenfolge vornehmen.
- Sie müssen die OldAge-Informationen der Nodes immer aktualisieren und den neuen Node richtig positionieren.
- In der Abbildung 1 folgt auf OldAge 0 - next - OldAge 5. Es wird damit die Schreibrichtung angezeigt, und dass der Node mit OldAge5 als erster überschrieben werden soll.
- Erweitern Sie die Klassen mit Attributen und Methoden, wenn es wirklich erforderlich ist.

Implementierung Klasse **Ringpuffer**

- Erstellen Sie die Klasse Ringpuffer als "übergeordnete" (keine Vererbung gemeint) Klasse für die Kontrolle über den eigentlichen Ring.
- Methoden der Klasse Ringpuffer sollen sich nur um das Handling der Datennodes/Ring kümmern und keine Nutzereingaben fürs Menü verarbeiten (z.B. Menü ausgeben, Menüauswahl einlesen, etc.). Schreiben Sie für die Darstellung des Menüs eigene Methoden oder realisieren Sie dies in der main() selbst.
- Der Datenring besteht aus 6 Nodes der Klasse RingNode und ist von außen nicht direkt zu erreichen, sondern nur über die Klasse Ringpuffer.

- Der Anker des Ringpuffers zeigt immer auf den aktuellsten Ringnode (OldAge=0).
- Zu Beginn ist Ihr Ring noch leer und der Anker zeigt auf "null". Sie können Ihren Ring jetzt auf zwei Arten aufbauen (Ihre Entscheidung):
  - Variante A (statisch): Sie legen zu Beginn schon 6 leere Nodes im Ring an und schreiben dann die ersten 6 Nutzerdaten fortlaufend in die vorbereiteten Nodes. Erst bei der siebten Eingabe ersetzen Sie den "ältesten"Node und fahren dann wie oben beschrieben fort. Bei dieser Variante müssen Sie dafür sorgen, dass leere Nodes bei der Suche oder Ausgabe nicht berücksichtigt werden. Beispiel: nur zwei Datensätze wurden vom Nutzer eingegeben, dann dürfen bei einer Ausgabe auch nur die beiden Datensätze angezeigt werden.
  - Variante B (dynamisch): Sie lassen den Ring mit jeder Eingabe dynamisch wachsen, bis die maximale Anzahl von 6 Nodes erreicht ist. Erst wenn der Benutzer einen neuen Datensatz speichern will, legen Sie einen neuen Node an und fügen ihn dem Ring hinzu. Ist die maximale Anzahl von 6 erreicht, identifizieren Sie den ältesten Node, ersetzen ihn durch den neuen Node und verfahren weiter wie oben beschrieben. Bei dieser Variante müssen Sie sich um die Einhaltung der Obergrenze (6 Nodes) sowie um korrekte Einfüge-Operationen (umbiegen der next-Pointer) kümmern.
- Bei der ersten Einfüge-Operation hat Ihr neuer Node den Attributwert OldAge=0 und es muss keine Aktualisierung vorgenommen werden, da es noch keine weiteren Einträge gibt.
- Ab der zweiten Einfüge-Operation muss das OldAge Attribut aller schon bestehenden Nodes um 1 erhöht werden, da es einen neuen/aktuelleren Node mit dem Attributwert OldAge=0 gibt und damit alle anderen Nodes "älter"werden.
- Ist der Ring voll besetzt und Sie wollen einen Node einfügen, müssen Sie Ihren ältesten Node identifizieren (OldAge=5), sich die Referenzen vom Vorgänger und Nachfolger merken, ihn löschen und an seiner Stelle den neuen Node platzieren. Anschließend müssen die Referenzen vom Vorgänger und auf den Nachfolger umgestellt werden (ähnlich verkette Liste).
- Wenn Sie den ältesten Node ersetzen, so müssen Sie in richtig entfernen (inkl. Änderung der betroffen Pointer). Sie dürfen nicht einfach die neuen Attributwerte in den alten Node schreiben.
- Erweitern Sie die Klasse bei Bedarf (Stichworte z.B. friendly, getter/setter).

Implementierung Klasse **Ringnode**

- Die Klasse Node ist der "'dumme" Datencontainer, in dessen Attribute die Daten der aktuellen Sicherung geschrieben werden.
- Einzige Intelligenz ist der "'next" Pointer auf den nächsten Node.
- Ringnode muss eine eigene Klasse sein und darf nicht einfach als Struct realisiert werden.
- Erweitern Sie die Klasse bei Bedarf (Stichworte z.B. friendly, getter/setter).

2.3 Binärbaum - Datenhaltung und Operation

Ihnen liegt ein großer Datensatz aus der letzten Zielgruppenanalyse vor. Um diesen auswerten zu können, müssen Sie ihn zunächst in einer entsprechenden Datenstruktur aufbereiten. Hierzu haben Sie sich für eine einfache Variante eines Binär-Bäums entschieden.
Entwickeln Sie eine Haupt-Klasse **„Tree“**, die als übergeordnete Klasse (keine Vererbung gemeint) die Kontrolle über den Baum hat und für alle Operationen verantwortlich ist (z.B. Node hinzufügen, Node löschen, Node suchen, Node ausgeben, usw.). Der eigentliche Baum besteht aus **“TreeNodes“**, die jeweils mindestens die angegebenen Datenattribute (Abbildung 2) sowie die erforderlichen Referenzen auf den links/rechts folgenden TreeNode bzw. Nullpointer, besitzen. Den erforderlichen Aufbau der Klassen können Sie der Abbildung 2 entnehmen. Sie können die Klassen erweitern, wenn es sinnvoll erscheint (Stichwort z.B. Getter/Setter, friendly). Wo ein TreeNode im Tree platziert wird, entscheidet sich anhand des Attributs “NodePosID“. Dieser Integer-Wert errechnet sich aus den Attribut-Werten “Alter“, “PLZ“ und “Einkommen“ des Nodes.

$$Alter(int) + PLZ(int) + Einkommen(double) = Positionsindikator(int)$$

Um den chonologischen Ablauf der Einfüge- und Löschoperationen später nachvollziehen zu können, benötigt Sie für jeden TreeNode eine Seriennummer (ID). Nutzen Sie hierfür das Integer-Attribut “NodeID“ des TreeNodes, das Sie für jeden neuen Node inkrementiert speichern. Beachten Sie, dass die NodeID eine fortlaufende Seriennummer ist und nicht mit der NodePosID zu verwechseln ist.
Sie müssen den Baum nach Operationen **nicht** ausbalancieren, aber nach z.B. Löschvorgängen wieder korrekt funktionsfähig machen. *Beachten Sie die Lösungshinweise.*

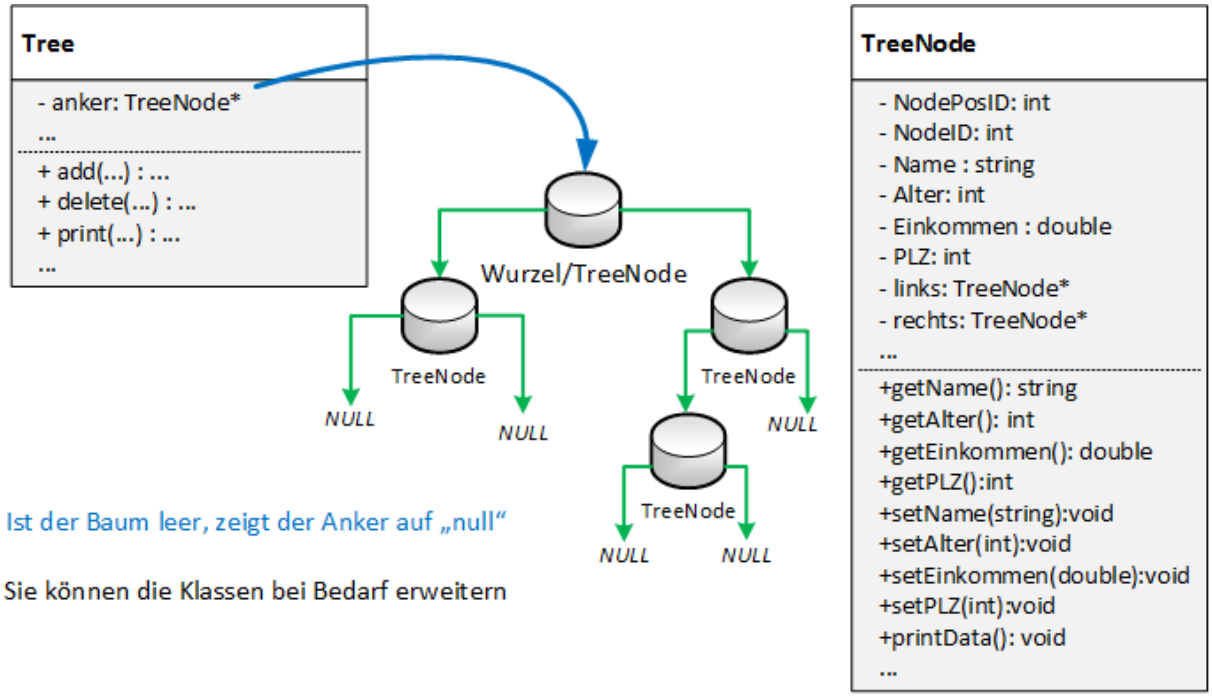


Abbildung 2: Beispiel, Aufbau der Baumstruktur und Klassen

Der Benutzer soll über ein Menü mit der Datenstruktur arbeiten können und folgende Möglichkeiten haben (beachten Sie die Lösungshinweise 2.3.4):

- Hinzufügen neuer Datensätze als ***Benutzereingabe***
- Hinzufügen neuer Datensätze aus einer ***CSV Datei***
- Löschen eines vorhandenen Datensatzes anhand der PositionsID.
- Suchen eines Datensatzes anhand des Personennamens.
- Anzeige des vollständigen Baums nach 'Preorder'.

```
==================================    // Beispiel: Menü der Anwendung
Person Analyzer v19.84, by George Orwell
==================================
1) Datensatz einfuegen, manuell
2) Datensatz einfuegen, CSV Datei
3) Datensatz löschen
4) Suchen
5) Datenstruktur anzeigen
?>
--------------------------------------------------------------------------------
?> 1                           // Beispiel: manuelles Hinzufügen eines Datensatzes
+ Bitte geben Sie die den Datensatz ein
Name ?> Mustermann
Alter ?> 1
Einkommen ?> 1000.00
PLZ ?> 1
+ Ihr Datensatz wurde eingefügt
--------------------------------------------------------------------------------
?> 5                         // Beispiel: Anzeigen eines Trees mit mehreren Einträgen
ID | Name       | Alter | Einkommen |  PLZ  | Pos
---+------------+-------+-----------+-------+-------
0  |  Mustermann|      1|       1000|      1|1002
3  |        Hans|      1|        500|      1|502
5  |     Schmitz|      1|        400|      2|403
4  |     Schmitt|      1|        500|      2|503
1  |      Ritter|      1|       2000|      1|2002
2  |      Kaiser|      1|       3000|      1|3002
--------------------------------------------------------------------------------
?> 4                           // Beispiel:  Datensatz suchen
+ Bitte geben Sie den zu suchenden Datensatz an
```

```
Name ?> Schmitt
+ Fundstellen:
NodeID: 4, Name: Schmitt, Alter: 1, Einkommen: 500, PLZ: 2, PosID: 503
--------------------------------------------------------------------------------
?> 3                          // Beispiel: Datensatz löschen
+ Bitte geben Sie den zu löschenden Datensatz an
PosID ?> 502
+ Datensatz wurde gelöscht.
--------------------------------------------------------------------------------
?> 5                            // zum Vergleich nochmal Struktur angezeigt
ID | Name       | Alter | Einkommen |  PLZ  | Pos
---+------------+-------+-----------+-------+-------
0  |  Mustermann|      1|       1000|      1|1002
4  |     Schmitt|      1|        500|      2|503
5  |     Schmitz|      1|        400|      2|403
1  |      Ritter|      1|       2000|      1|2002
2  |      Kaiser|      1|       3000|      1|3002
--------------------------------------------------------------------------------
?> 2                            // Beispiel: CSV Import
+ Möchten Sie die Daten aus der Datei "ExportZielanalyse.csv" importieren (j/n) ?> j
+ Daten wurden dem Baum hinzugefügt.
```

Beantworten Sie folgende Fragen nach erfolgreicher Implementierung:

- Die NodeID eignet sich nicht als Positionsindikator. Warum? Wie würde sich das auf den Baum auswirken?
- Was würde passieren, wenn die NodePosID mehrfach vergeben wird?
- Welche Interpretation lassen die sortierten Daten später zu? Was könnten Sie aus den Zusammenhängen schließen?
- Kennen Sie eine Datenstruktur, die sich für eine solche Aufgabe besser eignen würde?

2.3.1 Lösungshinweise

2.3.2 Klassen und Attribute

- Der Baum “Tree“ und die Knoten/Blätter “TreeNodes“ sind je eigenständige Klassen, die nicht untereinander vererben.
- Die NodeID ist eine fortlaufende Seriennummer für jeden Node, die beim Anlegen eines neuen Datensatzes einmalig vergeben wird und die Einfüge-Reihenfolge nachvollziehbar macht. Die

NodePosID ist damit nicht zu verwechseln, denn diese wird aus dem Alter, dem Einkommen und der PLZ des Datensatzes errechnet und dann für die Platzierung im Baum genutzt.

- Nutzen Sie die Datenkapselung und schützen Sie alle Attribute vor Zugriff. Erstellen Sie Setter/Getter, wenn es sinnvoll ist.
- Der Tree besitzt einen Poiner "Anker" auf den ersten TreeNode. Ist der Baum leer, muss dieser Pointer auf Null zeigen.
- Alle TreeNodes haben zwei Zeiger vom Typ TreeNode, die auf die nachfolgenden rechten/linken TreeNodes verweisen. Gibt es keine Nachfolger, so müssen die Referenzen auf Null oder einen festgelegten Nullpointer verweisen.

2.3.3 Tree und Tree-Operationen

- Das Einfügen eines neuen Nodes erfolgt anhand der errechneten Positions-ID (NodePosID). Laufen Sie ab Wurzel die bestehenden Nodes ab und vergleichen Sie die Positions-ID, in deren Abhängigkeit dann in den linken oder rechten Teilbaum gewechselt wird. Sie können davon ausgehen, dass der Benutzer nur gültige Werte eingibt, die nicht zu einer Mehrfachvergabe der PositionsID führen.
- Bei der Suchfunktion soll nach dem Namen einer Person gesucht werden und die Node-Position sowie die vorhandenen Daten ausgegeben werden. Beachten Sie hierbei, dass „Name" kein eindeutiges Schlüsselelement ist und es mehrere Datensätze mit dem Personennamen z.B. „Schmitt" geben kann. Sie müssen alle passenden Einträge finden und ausgeben. Überlegen Sie, ob hier eine rekursive oder iterativer Vorgehensweise besser ist.
- Löschen eines vorhandenen Datensatzes ist die anspruchsvollste Operation im Binärbaum. Haben Sie die Position des TreeNodes im Tree ausgemacht, müssen Sie auf folgende 4 Fälle richtig reagieren. Der zu löschende TreeNode...

  1. ... ist die Wurzel.
  2. ... hat keine Nachfolger.
  3. ... hat nur einen Nachfolger (rechts oder links).
  4. ... hat zwei Nachfolger.

  Denken Sie daran, dass Sie bei einer Löschoperation auch immer die Referenz des Vorgängers auf die neue Situation umstellen müssen.
- Die erwartete Funktionalität der Methoden aus Bild 2 ergibt sich aus der Benennung. Erweitern Sie die Klassen um eigene Methoden und Attribute wenn es sinnvoll ist.
- Sie müssen den Baum **nicht** ausbalancieren.

2.3.4 Eingabe der Daten

- Manuelle Eingabe - Wie im Schaubild oben zu erkennen soll der Benutzer über die Konsole einen neuen Datensatz anlegen können. Dabei werden die benötigten Positionen der Reihe nach als Benutzereingabe aufgenommen.
- CSV-Datei - Um einen schnelleren Import zu ermöglichen, soll der Benutzer auch eine CSV Datei einlesen können.
  - Die CSV Datei "ExportZielanalyse.csv" liegt im gleichen Verzeichnis wie das Programm. Sie müssen dem Benutzer keine andere Datei zur Auswahl geben oder eine Eingabe für den Dateinamen realisieren. Fragen Sie lediglich ob die Datei mit dem Namen wirklich importiert werden soll.
  - Jede Zeile der Datei stellt einen Datensatz dar, der seine Elemente mittels Semikolon trennt. (Hilfestellung: `https://www.c-plusplus.net/forum/281529-full` und `https://de.wikipedia.org/wiki/CSV_(Dateiformat)`)
  - Beachten Sie, dass Sie die gelesenen Werte in den richtigen Zieldatentyp übertragen müssen.
  - Die Reihenfolge der Spalten in der CSV Datei, entspricht der manuellen Eingabe (siehe Beispiel oben).
  - Der CSV Import darf die bereits vorhandenen, manuell eingetragenen, Datensätze nicht überschreiben, sondern nur den Tree damit erweitern.

2.3.5 Ausgaben und Benutzerinteraktion

- Übernehmen Sie das Menü aus dem Beispiel oben.
- Fehlerhafte Eingaben im Menü müssen abgefangen werden.
- Bei fehlgeschlagenen Operationen wird eine Fehlermeldung erwartet.
- Die Ausgabe des gesamten Baums (alle TreeNode Daten, siehe Beispiel) soll nach der 'Pre-Order' Reihenfolge erfolgen.
- Formatieren Sie die Ausgaben sinnvoll (siehe Beispiel).

2.4 Backup in hochverfügbaren Datenstrukturen (Zusatzaufgabe)

Situation: Das von Ihnen geschaffene, fragile Mass-Data-System (MDS) ist das Herz, die Seele und die Einnahmequelle des Unternehmens. Laut internem Service-Level-Agreement, muss das System extrem hochverfügbar sein und soll trotzdem in die Rotation der Datensicherung aufgenommen werden. Durch die geforderte Verfügbarkeit und der hohen Datenänderungsrate, ist ein "cold" Backup (abschalten und sichern) oder ein "hot/online" Backup (Stück für Stück sichern, wenn TreeNodes ohne Zugriff sind) nicht möglich. Sie müssen daher einen Snapshot (alles sofort) realisieren.

- Überlegen Sie: Warum ist keine der anderen Sicherungsarten passend? Warum passt das hot/online Backup auch nicht?

Kombinieren Sie jetzt die beiden oberen Teilaufgaben zu einem Gesamtsystem, welches den Binärbaum auf Befehl in einem Ringpuffer „backupt". Die Funktionalität des Binärbaums soll wie in Teilaufgabe 2 erhalten bleiben. Die Abbildung 3 zeigt den gedachten Aufbau. Vereinen Sie die Menüstruktur der bisherigen beiden Teilaufgaben sinnvoll. Beispiel:

```
1  ===================================
2  HPDS v0.1, by Speedy Gonzales
3  ===================================
4  Backupsteuerung:
5  1) Backup einfuegen
6  2) Backup suchen
7  3) Alle Backups ausgeben
8  ----------------------------
9  Aktueller Baum:
10 4) Datensatz einfuegen
11 5) Datensatz löschen
12 6) Suchen
13 7) Datenstruktur anzeigen
14 ?>
```

2.4.1 Lösungshinweise

2.4.2 Anpassen des Ringpuffers

- Nutzen Sie jetzt das alte RingNode „SymbolicData" Attribut für die Referenz auf den zu speichernden Tree. Somit entfällt dieses Attribut bei der Nutzereingabe sowie Ausgabe.
- Die Suche nach Daten soll sich über alle Trees erstrecken. Wie bei den Trees, ist das zu suchende Attribut jetzt der Personenname. Geben Sie das Suchergebnis aus allen Trees, unter Angabe des Fundortes in welchem Backup, auf der Konsole aus.
- Passen Sie die Ringklassen für Operationen und Parameter vom Typ „Tree" bzw. „Tree*" an.

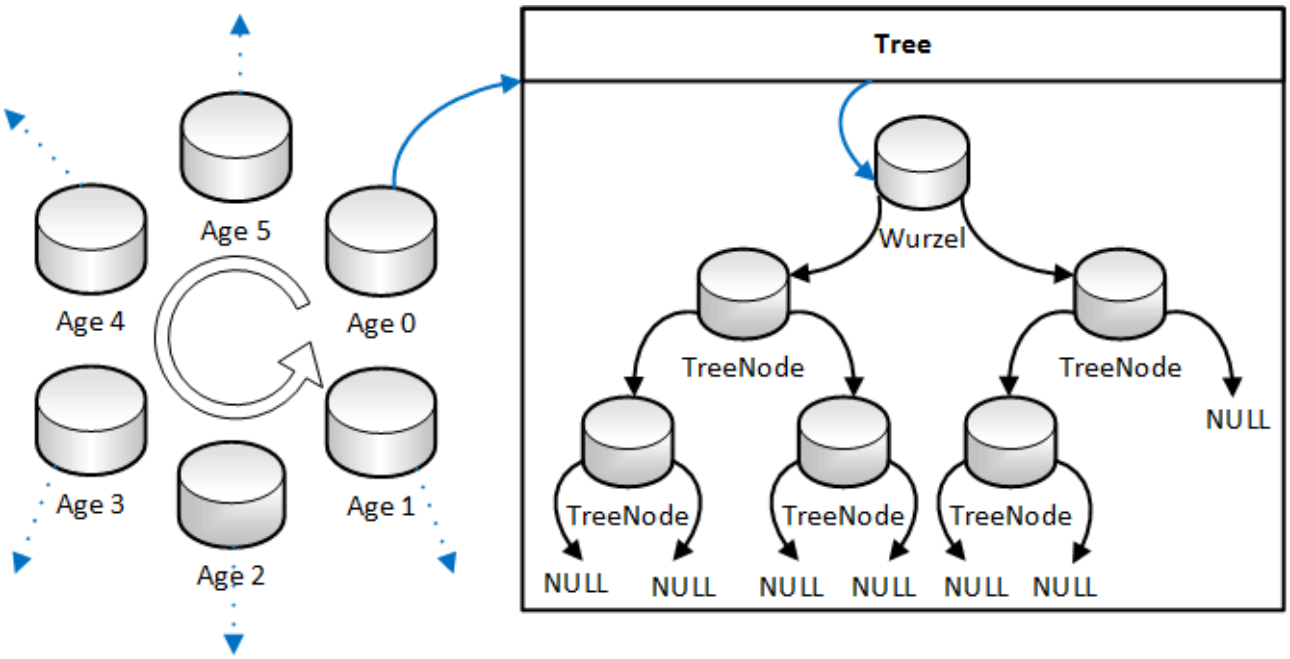


Abbildung 3: Im Ring gesicherter Snapshot der Baumstruktur

2.4.3 Sicherung/Kopie des Trees

- Beachten Sie, dass wenn ein Objekt Referenzen auf Andere als Attribute besitzt, diese nicht einfach kopieren/zuweisen können, da der Verweis auf die Speicherstelle bleibt. Ändern Sie Dinge im aktuellen Baum, hätte das Auswirkung auf die Bäume im Backup. Sie müssen die Bäume jeweils in neuen Speicher überführen (Stichwort: CopyConstructor).
- Überlegen Sie, ob die Überführung/Kopie der DatenNodes mit einer rekursiven oder iterativen Methode bewerkstelligt werden soll.

3 Praktikum 3: Sortierverfahren / Open Multi-Processing (OpenMP)

Ihre Vorgesetzten bei der Firma "Data Fuse Inc." sind begeistert von Ihren Fähigkeiten! Da die Verarbeitungsgeschwindigkeit der enormen Datenmengen weiter optimiert werden muss wurden Sie beauftragt, ein Framework zur Messung von Laufzeiten zu entwickeln. Mit Hilfe dieses Programms sollen Sie anschließend die Ausführungsdauer verschiedener Sortieralgorithmen sowie im zweiten Teil die Laufzeit von Matrixmultiplikationen quadratischer Matrizen in Abhängigkeit einer Problemgröße n untersuchen und auswerten. Mit Hilfe von OpenMP sollen die Zeiten gemessen und die Matrixmultiplikationen (nicht die Sortieralgoritmen!) parallelisiert werden. Die Ergebnisse sollen dann im Praktikum präsentiert und diskutiert werden.

3.1 Aufgabenstellung

1. Erstellen Sie das Hauptprogramm, in dem Sie die zu messenden Algorithmen aufrufen und die Ergebnisse der Zeitenmessungen in Textdateien schreiben. Sie können hierzu die Vorlage aus ILIAS benutzen, dürfen aber auch gerne eine eigene Lösung erstellen.

2. Erstellen Sie eine eigene Algorithmenbibliothek, bestehend aus der Header-Datei *MyAlgorithms.h* und implementieren Sie die folgenden Algorithmen in der zugehörigen cpp-Datei *MyAlgorithms.cpp* sowie im eigenen Namespace *MyAlgorithms*:

   - Heapsort
   - Mergesort
   - Quicksort
   - Shellsort mit der **Hibbard Folge** ($H_i = 2H_{i-1} + 1$)

3. Erweitern Sie ihre Algorithmenbibliothek indem Sie die schulbuchmäßige Matrixmultiplikation für quadratische Matrizen implementieren. Implementieren Sie sowohl die spalten- als auch die zeilenweise Speicherung/Verarbeitung der Daten:

   - Matrix-Multiplikation Column Major
   - Matrix-Multiplikation Row Major

4. Parallelisieren Sie die Berechnung der Matrizen durch hinzufügen der passenden OpenMP-Präprozessor-Direktive (siehe Codebsp. 2) vor der äußeren for-Schleife der Matrix-Multiplikation.

5. Messen Sie die Ausführungszeiten folgender Sortieralgorithmen:

   - Heapsort, $n = 1000 : 1000 : 1000000$
   - Mergesort, $n = 1000 : 1000 : 1000000$
   - Quicksort, $n = 1000 : 1000 : 1000000$
   - Shellsort, $n = 1000 : 1000 : 1000000$

sowie die folgenden Matrix-Multiplikationen:

- MatrixMul RowMajor 1 Thread, $n = 2 : 1 : 799$, $n = 800 : 11 : 1800$
- MatrixMul RowMajor max Threads, $n = 2 : 1 : 799$, $n = 800 : 11 : 1800$
- MatrixMul ColMajor 1 Thread, $n = 2 : 1 : 799$, $n = 800 : 11 : 1800$
- MatrixMul ColMajor max Threads, $n = 2 : 1 : 799$, $n = 800 : 11 : 1800$

in Abhängigkeit der Problemgröße n. Initialisieren Sie dabei Ihre Datenstrukturen vor jeder Messung mit Zufallszahlen (Sortieralgorithmen: Integer, Matrixmultiplikation: Double). Für *max* setzen Sie die Anzahl der **Kerne** (nicht Threads!) ihrer CPU ein.

Hinweis: Kompilieren Sie Ihr Projekt vor der Messung unbedingt im *RELEASE* Modus und verwenden Sie das Compilerflag *-O3* um eine maximale Performance zu erhalten. Zusätzlich sollten Sie alle unnötigen Konsolenausgaben für die Messungen deaktivieren, da diese sehr viel Zeit kosten.

6. Stellen Sie ihre Messergebnisse unter Zuhilfennahme von MATLAB, Octave oder GNUPLOT grafisch dar (Beispiele: siehe 3.7)! Entsprechen die Messergebnisse den Erwartungen (z.B. bzgl. O-Notation)? Achten Sie bei den Plots auf aussagekräftige Achsenbeschriftungen und eine vernünftige Legende!

7. Beachten Sie unbedingt die Lösungshinweise (3.2) sowie die Testläufe (3.3) und **planen Sie genügend Zeit für die Messungen ein**!

8. Alle Beispiele (Textausgaben, Codevorlagen, Plots,...) dienen der Illustration und dürfen gerne entsprechend Ihren eigenen Vorstellungen angepasst werden.

3.2 Lösungshinweise

3.2.1 Matrixmultiplikation: Aufbau der Matrizen und Allokation im Speicher

Gegeben sei eine quadratische Matrix $A \in \mathbb{R}^{n \times n}$ mit n Zeilen und n Spalten. Ihre Dimension in y-Richtung, bzw. die Anzahl der Zeilen, wird als "leading dimension y (ldy)" bezeichnet, die Dimension in x-Richtung, bzw die Anzahl der Spalten, als "leading dimension x (ldx)". Da wir nur quadratische Matrizen betrachten gilt $ldx = ldy = ld = n$. Zusätzlich „denken" wir in C/C++ und indizieren die Matrixelemente ab 0 (Abb. 4).

$$A = \underbrace{\left.\begin{bmatrix} a_{0,0} & \dots & a_{0,n-1} \\ \vdots & \vdots & \vdots \\ a_{n-1,0} & \dots & a_{n-1,n-1} \end{bmatrix}\right\} n,\ ldy}_{n,ldx}$$

Abbildung 4: Quadratische Matrix der Größe n

Um diese zweidimensionale Datenstruktur möglichst effizient in C/C++ speichern und verarbeiten zu können sollten die Daten unbedingt hintereinander und zusammenhängend im Speicher abgelegt werden. Die zweidimensionale $(n \times n)$-Matrix soll deswegen auf ein eindimensionales Array L mit $n * n$ Elementen abgebildet werden, wobei entweder alle Spalten oder alle Zeilen der Matrix der Reihe nach aneinander gehängt werden können (Abb. 5):

$$L = \begin{bmatrix} a_{0,0} \\ \vdots \\ a_{n-1,0} \\ \vdots \\ \vdots \\ a_{0,n-1} \\ \vdots \\ a_{n-1,n-1} \end{bmatrix}$$

$$A_{i,j} \rightarrow L[i + j * ld]$$

(a) spaltenorientierte Speicherung

$$L = \begin{bmatrix} a_{0,0} \\ \vdots \\ a_{0,n-1} \\ \vdots \\ \vdots \\ a_{n-1,0} \\ \vdots \\ a_{n-1,n-1} \end{bmatrix}$$

$$A_{i,j} \rightarrow L[i * ld + j]$$

(b) zeilenorienterte Speicherung

Abbildung 5: eindimensionales Array L

Mit diesem Array L soll im Programm gearbeitet werden. Dazu muss die zweidimensionale Indizierung der Matrix $A \in \mathbb{R}^{n \times n}$ in die passende eindimensionale Indizierung des Arrays L übersetzt werden. Ein Indexpaar (i, j) wird dabei wie in Abb. 5 übersetzt. Nutzen Sie die C++-Klasse ***vector*** zur Realisierung des linearen Arrays. Machen Sie sich mit den Member-Methoden dieser Klasse vertraut!

3.2.2 Matrixmultiplikation : Formel

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} * b_{kj} \quad mit\ i = 0\ ...\ n-1,\ j = 0\ ...\ n-1 \tag{1}$$

$$\underbrace{\begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}}_{\text{C}} = \underbrace{\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}}_{\text{A}} \times \underbrace{\begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}}_{\text{B}}$$

Abbildung 6: Matrixmultiplikation $C = A \times B$ für $n = 4$

3.2.3 OpenMP Beispiele

```
#include <iostream>
#include <omp.h>

int main(int, char*[]) {

    int num_procs = omp_get_num_procs();
    omp_set_num_threads(num_procs);

    int threads = 0;
    int id = 0;

    #pragma omp parallel private (id)
    {
     threads = omp_get_num_threads();
     id = omp_get_thread_num();
     std::cout << "hello world from thread: " << id << " out of ";
     std::cout << threads << "\n";
    }
    return 0;
}
```

Listing 1: OpenMP Hello World

```
#include <iostream>
#include <omp.h>

int main(int, char*[]) {

    int num_procs = omp_get_num_procs();
    omp_set_num_threads(num_procs);

    int arraysize=1000000000;
    int* myarray = new int[arraysize];

    int i;
    #pragma omp parallel for
    for (i=0;i<arraysize;i++) {
        myarray[i] = i;
    }

    return 0;
}
```

Listing 2: OpenMP Beispiel zur Parallelisierung einer for-Schleife

3.3 Testläufe

3.3.1 Sortierverfahren

Testen Sie die Sortierverfahren zunächst mit folgenden Zahlen: 98 44 30 22 64 63 11 23 8 18.

- Beispielausgabe Heap Sort:

```
Heap Sort:
Durchlauf 0: 98  44  30  22  64  63  11  23  8  18
percDown(63) Durchlauf 5: 98  44  30  22  64  63  11  23  8  18
percDown(64) Durchlauf 4: 98  44  30  22  64  63  11  23  8  18
percDown(22) Durchlauf 3: 98  44  30  23  64  63  11  22  8  18
percDown(30) Durchlauf 2: 98  44  63  23  64  30  11  22  8  18
percDown(44) Durchlauf 1: 98  64  63  23  44  30  11  22  8  18
percDown(98) Durchlauf 0: 98  64  63  23  44  30  11  22  8  18
Heap Sort:
Durchlauf 0: 98  64  63  23  44  30  11  22  8  18
percDown(18) Durchlauf 9: 64  44  63  23  18  30  11  22  8  98
percDown(8) Durchlauf 8: 63  44  30  23  18  8  11  22  64  98
percDown(22) Durchlauf 7: 44  23  30  22  18  8  11  63  64  98
percDown(11) Durchlauf 6: 30  23  11  22  18  8  44  63  64  98
percDown(8) Durchlauf 5: 23  22  11  8  18  30  44  63  64  98
percDown(18) Durchlauf 4: 22  18  11  8  23  30  44  63  64  98
percDown(8) Durchlauf 3: 18  8  11  22  23  30  44  63  64  98
percDown(11) Durchlauf 2: 11  8  18  22  23  30  44  63  64  98
percDown(8) Durchlauf 1: 8  11  18  22  23  30  44  63  64  98
8  11  18  22  23  30  44  63  64  98
Finished in    0.062 sec
```

- Beispielausgabe Mergesort

```
MergeSort(0,9)
MergeSort(0,4)
MergeSort(0,2)
MergeSort(0,1)
MergeSort(0,0)
MergeSort(1,1)
Merge(0,0,1): 44 98 30 22 64 63 11 23 8 18
MergeSort(2,2)
Merge(0,1,2): 30 44 98 22 64 63 11 23 8 18
MergeSort(3,4)
MergeSort(3,3)
MergeSort(4,4)
Merge(3,3,4): 30 44 98 22 64 63 11 23 8 18
Merge(0,2,4): 22 30 44 64 98 63 11 23 8 18
```

```
MergeSort(5,9)
MergeSort(5,7)
MergeSort(5,6)
MergeSort(5,5)
MergeSort(6,6)
Merge(5,5,6): 22 30 44 64 98 11 63 23 8 18
MergeSort(7,7)
Merge(5,6,7): 22 30 44 64 98 11 23 63 8 18
MergeSort(8,9)
MergeSort(8,8)
MergeSort(9,9)
Merge(8,8,9): 22 30 44 64 98 11 23 63 8 18
Merge(5,7,9): 22 30 44 64 98 8 11 18 23 63
Merge(0,4,9): 8 11 18 22 23 30 44 63 64 98
```

- Beispielausgabe Quicksort

```
QuickSort(a,0,9)
pivot = a[4] = 64
Result: 18 44 30 22 8 63 11 23 64 98
QuickSort(a,0,7)
pivot = a[3] = 22
Result: 18 11 8 22 30 63 44 23 64 98
QuickSort(a,0,2)
pivot = a[1] = 11
Result: 8 11 18 22 30 63 44 23 64 98
QuickSort(a,4,7)
pivot = a[5] = 63
Result: 8 11 18 22 30 23 44 63 64 98
QuickSort(a,4,6)
pivot = a[5] = 23
Result: 8 11 18 22 23 30 44 63 64 98
QuickSort(a,5,6)
pivot = a[5] = 30
Result: 8 11 18 22 23 30 44 63 64 98
QuickSort(a,8,9)
pivot = a[8] = 64
Result: 8 11 18 22 23 30 44 63 64 98
```

- Beispielausgabe Shellsort

```
Shell Sort (H = 2H+1)
8  11  18  22  23  30  44  63  64  98
Finished in    10 sec
```

3.3.2 Matrixmultiplikation

Testdaten Matrixmultiplikation:

$$\underbrace{\begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}}_{\text{C}} = \underbrace{\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}}_{\text{A}} \times \underbrace{\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}}_{\text{B}}$$

- Beispielausgabe Matrixmultiplikation

```
spaltenweise
******A******
1
3
2
4
******B******
5
7
6
8
******C******
19
43
22
50

zeilenweise
******A******
1
2
3
4
******B******
5
6
7
8
******C******
19
22
43
50
```

3.4 OpenMP Compiler-Einstellungen

3.4.1 Linux g++

Aktivierung von OpenMP unter Linux durch Compilerflag **-fopenmp** und linken gegen OpenMP Library **-lgomp**. Beispielaufruf für g++:

```
g++ main.cpp MyAlgorithms.h MyAlrogithms.cpp −o praktikum3 −fopenmp −lgomp
```

3.4.2 Windows Visual Studio

Aktivierung von OpenMP unter Visual Studio:
https://msdn.microsoft.com/de-de/library/fw509c3b.aspx

```
Öffnen Sie das Dialogfeld Eigenschaftenseiten des Projekts.
Erweitern Sie den Knoten Konfigurationseigenschaften.
Erweitern Sie den Knoten C/C++.
Wählen Sie die Eigenschaftenseite Sprache aus.
Ändern Sie die Eigenschaft OpenMP−Unterstützung.
```

3.4.3 Windows DevC++

Aktivierung von OpenMP unter DevC++:

```
Tools −> Compiler Options
Add the following commands when calling the compiler: −fopenmp
Add the following commands when calling the linker: −lgomp
```

3.5 Format der TXT-Dateien

Die Messungen sollten tabulatorgetrennt spaltenweise abgespeichert werden, damit sie möglichst einfach geplottet werden können:

- 1.Spalte: Problemgröße
- 2.Spalte: Berechnungsdauer

Beispiel:

```
...
986000  6.3498632997e−02
987000  6.3852430001e−02
988000  6.3209023996e−02
...
```

3.6 Beispiele zum Plotten mit MATLAB / GNUPLOT

- MATLAB
  Erzeugen Sie im selben Ordner, indem sich Ihre Messungen befinden, eine M-Skript-Datei mit einem beliebigen Namen, z.B. *make_plots.m*:

```
clear ; clc ; close  all ;

fid =fopen('quicksort.txt');
data=textscan(fid,'%d %f');
fclose ( fid ) ;
x=data{1};
quicksort_y=data{2};

fid =fopen('mergesort.txt');
data=textscan(fid,'%d %f')
fclose ( fid ) ;
mergesort_y=data{2};

fid =fopen('heapsort.txt');
data=textscan(fid,'%d %f');
fclose ( fid ) ;
heapsort_y=data{2};

fid =fopen('shellsort.txt');
data=textscan(fid,'%d %f');
fclose ( fid ) ;
shellsort_y =data{2};



figure ;
title ('sorting algorithms');
xlabel('n [-]');
ylabel('t [s]');
hold on;
plot(x,quicksort_y);
plot(x,mergesort_y);
plot(x,heapsort_y);
plot(x, shellsort_y );
legend('quicksort','mergesort','heapsort','shellsort','Location','northwest');
hold  off ;
```

  Führen Sie das Skript anschließend aus:

```
>> make_plots
```

- GNUPLOT
  Erzeugen Sie im selben Ordner, indem sich Ihre Messungen befinden, eine Datei mit einem beliebigen Namen, z.B. *plots.gnu*:

```
reset
set autoscale x
set autoscale y
set xlabel "n [-]"
set ylabel "t [s]"
set key top left

plot \
"quicksort.txt" with linespoints title 'Quicksort',\
"mergesort.txt" with linespoints title 'Mergesort',\
"shellsort.txt" with linespoints title 'Shellsort',\
"heapsort.txt" with linespoints title 'Heapsort',\
```

  Starten Sie nun Gnuplot, wechseln Sie in das korrekte Verzeichnis, und fuehren Sie das Skript wie folgt aus:

```
$ cd 'pfad-zum-gnuplot-skript'
$ load "plots.gnu"
```

  Weiterführende Befehle zu GNUPLOT findet man z.B. hier:
  http://gnuplot.sourceforge.net/docs_4.0/gpcard.pdf

3.7 Beispielplots

Die Plots sollten, natürlich in Abhängigkeit der verwendeten CPU, in etwa so aussehen (in den Abbildungen wurden die Legenden anonymisiert um die Ergebnisse nicht vorweg zu nehmen):

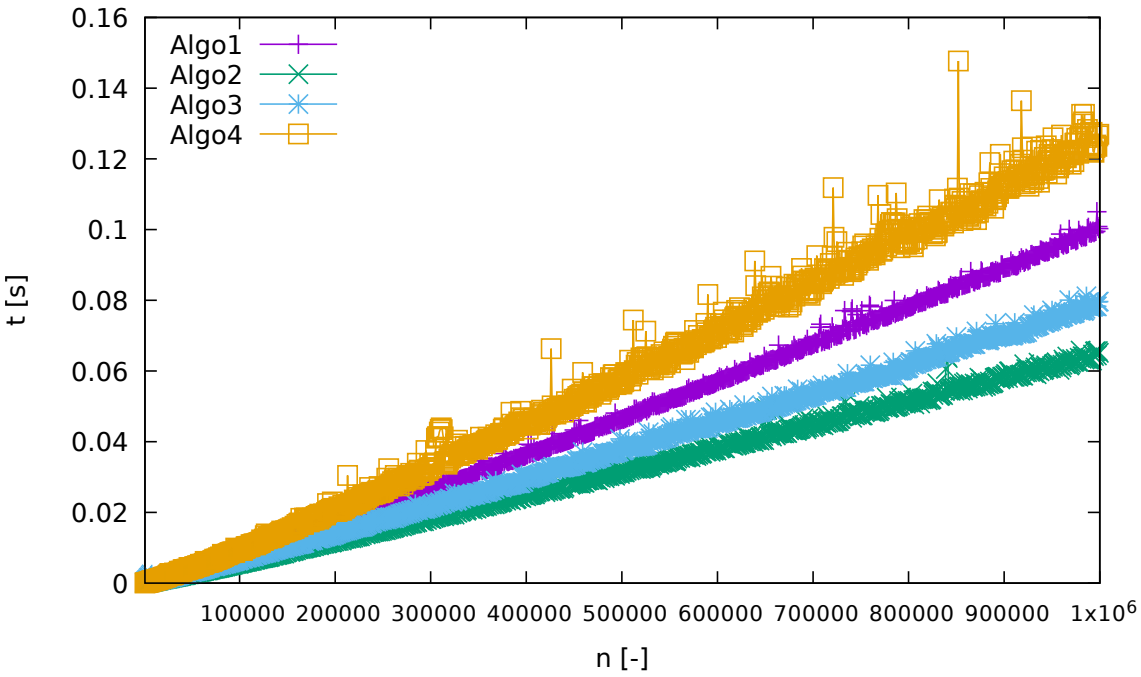


Abbildung 7: Laufzeitvergleich der Sortieralgorithmen

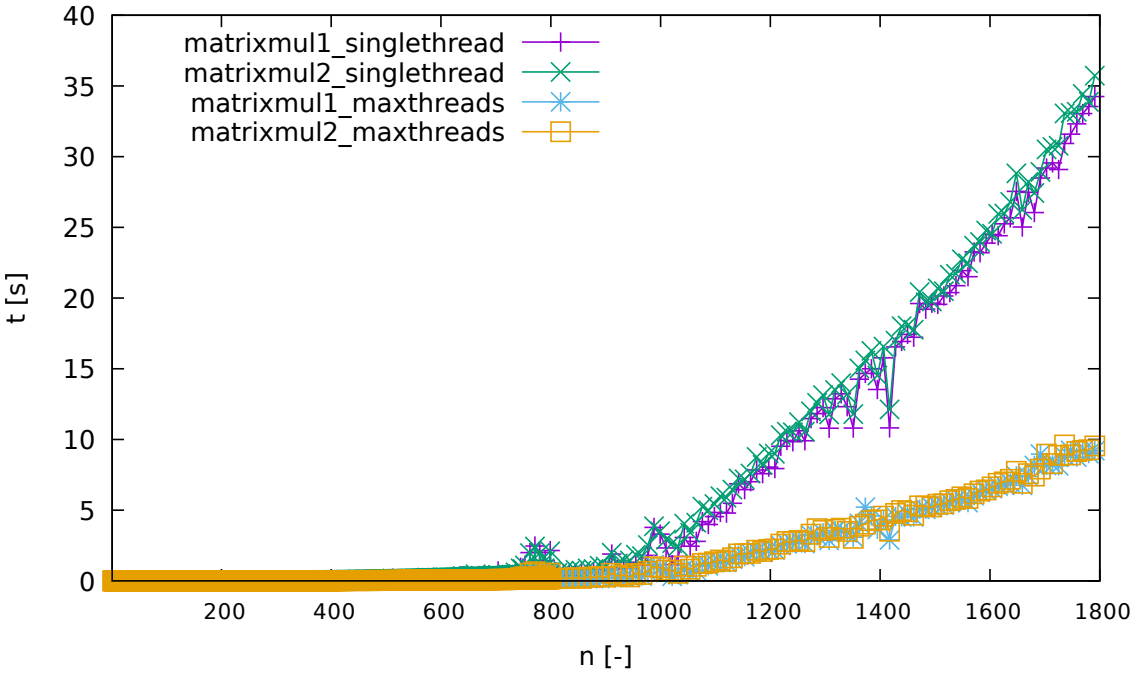


Abbildung 8: Laufzeitvergleich der Matrixmultiplikation (1 vs 4 Threads)

4 Praktikum 4 : Graphen

In dieser Aufgabe ist erneut Ihre Kompetenz als Systemarchitekt bei „Data Fuse“ gefragt. Die verschiedenen Standorte sollen mit neuen Routern verbunden werden und Sie sollen testen, ob die geplanten Router und deren Verbindungen ausreichen um alle Standorte zu verbinden. Ein weitere Test wird sein, dass die Entfernungen gemessen werden sollen.

In Abbildung 9 sehen Sie ein Beispiel, wie so ein Graph aufgebaut sein könnte, die Knoten stellen die Router an den Standorten dar, die Kanten die Verbindungen mit einem Gewicht als fiktive Entfernung. Hier lässt sich schnell erkennen, dass alle Standorte verbunden sind, mit der minimalen Entfernung wird es auch hier auf den ersten Blick schon schwieriger.

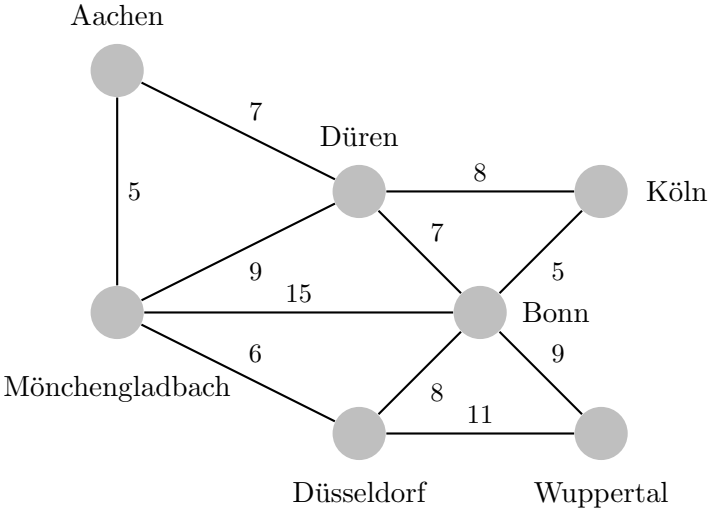


Abbildung 9: Beispiel Unternehmens Standorte mit Verbindungen

4.1 Aufgabenstellung

1. Beschäftigen Sie sich mit der Tiefen- und Breitensuche.
   Was ist der Unterschied zwischen den beiden Suchen?

   Implementieren Sie anschließend die rekursive Tiefensuche und die iterative Breitensuche.
   Was ist der Unterschied zwischen rekursiv und iterativ?

   Als Beispielgraph stehen ihnen drei Graphen Dateien als „.txt “ zur Verfügung. Sie finden den Graph dort als Kantenliste aufgebaut. Die erste Zeile gibt die Knotenzahl m an, alle folgenden Zeilen die Kanten in der Form: Startknoten $\rightarrow$ Endknoten $\rightarrow$ Gewicht, wobei die Knoten mit $0 \ldots (m-1)$ nummeriert sind. Der Graph ist ungerichtet und für diese Aufgabe können Sie das Gewicht noch ignorieren.

   Hinweis: Nutzen Sie das „besucht “-Flag der Knotenklasse.

2. Beschäftigen Sie sich mit der Thematik des minimalen Spannbaums. Kennen Sie die Bedeutung der Algorithmen nach Prim und Kruskal im Bezug auf den minimalen Spannbaum. Seien Sie in

der Lage die beiden Algorithmen zu erklären und die Unterschieden zu verdeutlichen.

Implementieren Sie anschließend die Algorithmen nach Prim und Kruskal. Mit diesen können Sie einen Minimalen Spannbaum auf zwei verschiedene Weisen erstellen. Geben Sie die Gesamtkosten der beiden Algorithmen aus und vergleichen Sie diese.

Auch hierfür steht Ihnen wieder eine Datei „graph.txt“ zur Verfügung. Die erste Zeile enthält wieder die Knotenzahl, alle weiteren enthalten die Kanten mit Start und Endknoten sowie einem Gewicht. Beachten Sie folgende Notation: Startknoten → Endknoten → Gewicht.

Die Implementierung kann ebenfalls in eine Beispieldatei integriert oder anhand der in der Vorlesung besprochenen API gemacht werden. In dem Template müssen nur die Algorithmen implementiert werden, die Übrigen Funktionalitäten sind bereits gegeben. Die API aus der Vorlesung stellt ebenfalls schon das gesamte Gerüst zur Verfügung.

4.2 Lösungshinweise

Für die Lösung steht Ihnen ein Framework zur Verfügung, was die Elemente der bisherigen Praktika nutzt. Dort sind die Stellen markiert, in denen Sie Ihre Algorithmen integrieren sollen. Es reicht, diese Lücken zu füllen, eine weitere Ergänzung ist nicht nötig, aber auch nicht verboten, wenn es Ihnen hilft.

Zudem folgen in den nächsten Abschnitten ein paar allgemeine Hinweise zu dem Algorithmen, die Ihnen die Programmierung erleichtern sollen. Beachten Sie, dass hier nur Hinweise gegeben werden. Gegebene Beispiele müssen erweitert oder überarbeitet und können nicht so übernommen werden.

4.2.1 Graph- und Knoten-Klasse

Um Ihnen die Arbeit zu erleichtern, wird hier ein Graph bereitgestellt. Er orientiert sich an den ersten Aufgaben und das meiste daraus sollte Ihnen bekannt sein. Ein Knoten besteht wie bei den Listen und Bäumen aus einem Wert als „Namen“ und bekommt nun noch zusätzlich die Attribute „besucht“ und „Distanz“. Wozu Sie diese verwenden können, wird in den nächsten Abschnitten erklärt.

Der Graph ist grundsätzlich aufgebaut wie ein Baum in Aufgabe 2. Zusätzlich werden aber mehr als 2 Nachbarn zugelassen und ein Knoten kann auch eine Verbindung wieder zum Elternknoten haben. Da es sich in diesen Aufgaben um ungerichtete Graphen handelt, wird jede Kante als Hin- und Rückkante angelegt. Jeder Knoten enthält daher eine Liste von Kanten. Eine Kante besteht dabei aus einem Zielknoten und für den zweiten Aufgabenteil zusätzlich noch aus einem Gewicht.

Ebenfalls wird ihnen in dieser Klasse bereits das verarbeiten der Datei zur Verfügung gestellt. Den Aufruf der Datei müssen Sie jedoch noch integrieren, da dies von Ihrer Plattform abhängt.

4.2.2 Tiefensuche

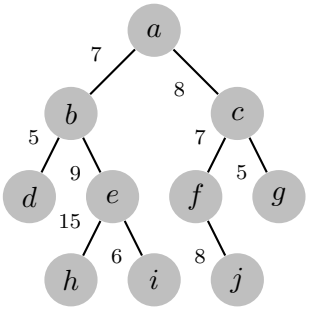
Die Tiefensuche wurde Ihnen in der Vorlesung vorgestellt. Sie sucht zunächst solange in die Tiefe, bis kein weiterer Knoten als Kind mehr folgt und geht dann wieder eine Ebene höher. Zur Implementierung

einer rekursiven Suche, ist in Algorithmus 1 der Pseudocode dargestellt. Bei einer Tiefensuche, werden alle erreichten Knoten als besucht markiert, gibt es Knoten, die nicht durch Tiefensuche erreicht werden können, so gibt es keine Verbindung zum Startknoten.

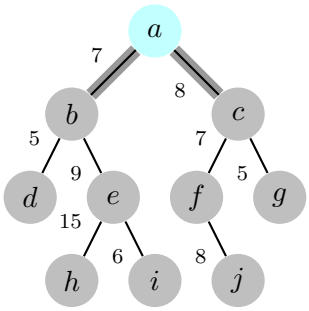
Algorithm 1: Tiefensuche rekursiv

```
marked[1..|V|] = false
Function DFS(G,v)
    marked[v] = true
    for ∀w ∈ G.adj(v) do
        if marked(w) == false then
            DFS(G,w)
```

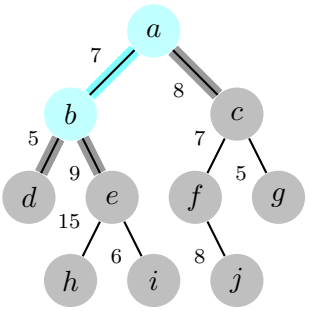
Die folgenden Grafiken zeigen die Tiefensuche exemplarisch an einem Baum, für einen Graphen funktioniert diese auf die gleiche Weise. Es muss nur berücksichtigt werden, ob ein Knoten bereits besucht wurde.



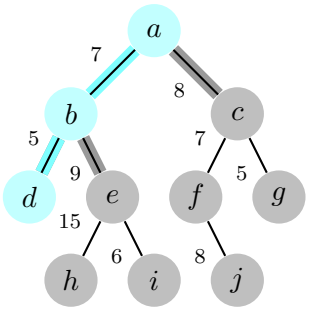
Original



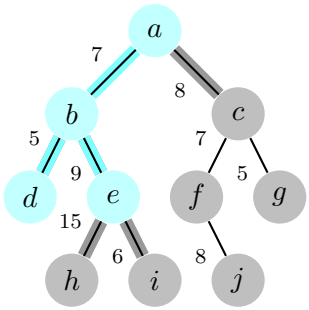
Schritt 1



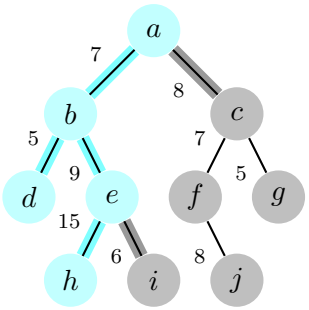
Schritt 2



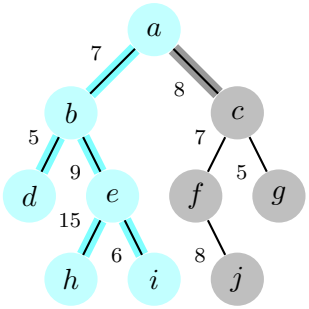
Schritt 3



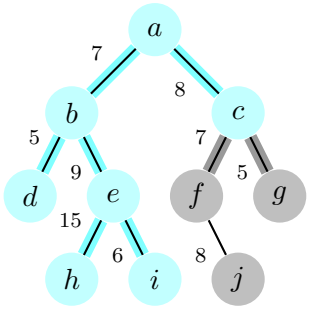
Schritt 4



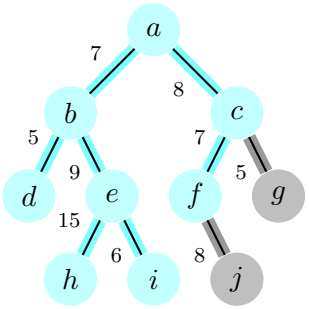
Schritt 5



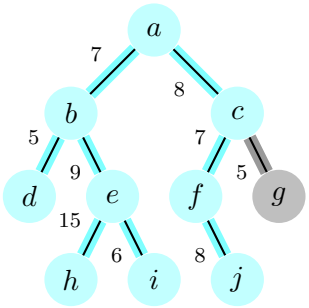
Schritt 6



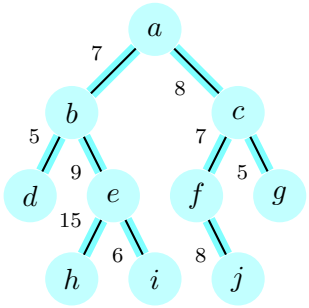
Schritt 7



Schritt 8



Schritt 9



Endergebnis

4.2.3 Breitensuche

Im Gegensatz zur Tiefensuche wird hier zunächst in der Breite gesucht. Das bedeutet zu einem Knoten werden zunächst alle Kinder untersucht, bevor man die Kindes-Kinder betrachtet.

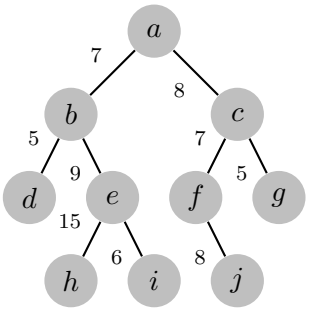
Auch hier können Sie sich wieder am Pseudocode aus der Vorlesung in Algorithmus 2 orientieren.

Algorithm 2: Breitensuche iterativ

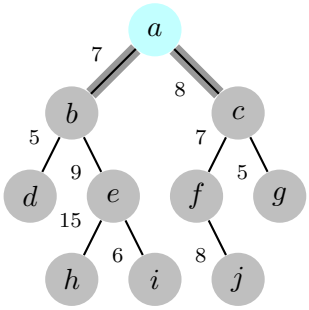
```
marked[1..|V|] = false
edgeTo[1..|V|] = 0
Function BFS(G,s)
    queue q
    q.enqueue(s)
    marked[s] = true while !q.isEmpty() do
        v = q.dequeue()
        for ∀w ∈ G.adj(v) do
            if marked[w] == false then
                edgeTo[w] = v
                marked[w] = true
                q.enqueue(w)
```

Analog zur Tiefensuche ist im folgenden auch die Breitensuche grafisch dargestellt. Auch hier kann

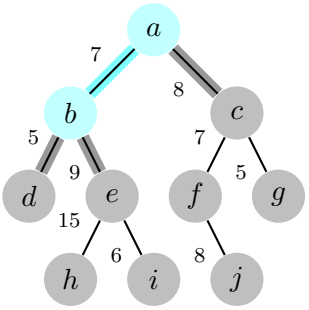
das Prinzip einfach auf einen Graphen übertragen werden.



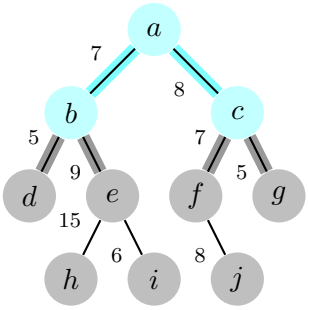
Original



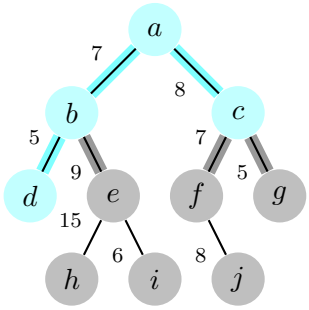
Schritt 1



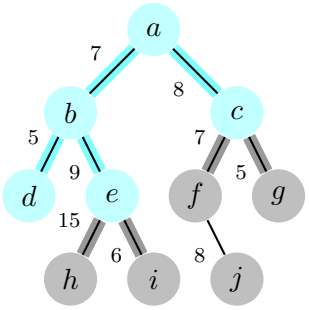
Schritt 2



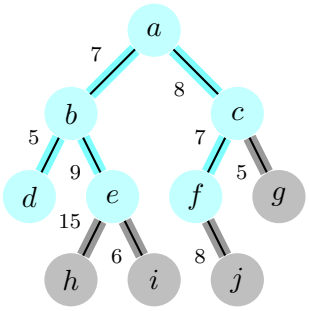
Schritt 3



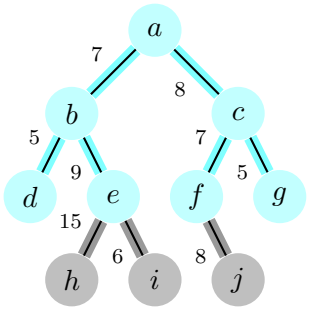
Schritt 4



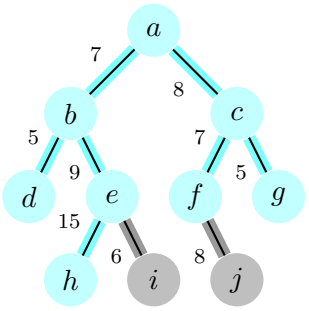
Schritt 5



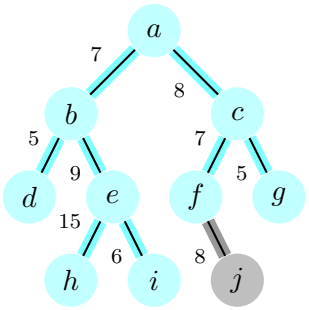
Schritt 6



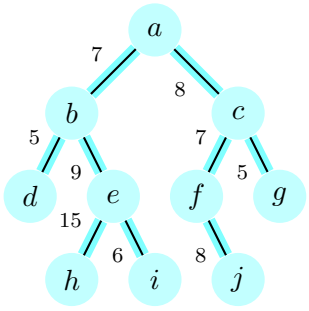
Schritt 7



Schritt 8



Schritt 9



Endergebnis

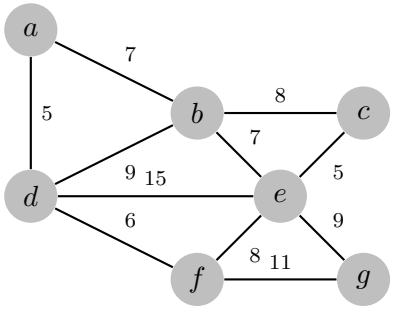
4.2.4 Prim

Der Prim Algorithmus dient der Erstellung eines Minimalen Spannbaum (Minimal-Spanning-Tree MST). Algorithmus 3 beschreibt ihn als Pseudocode.

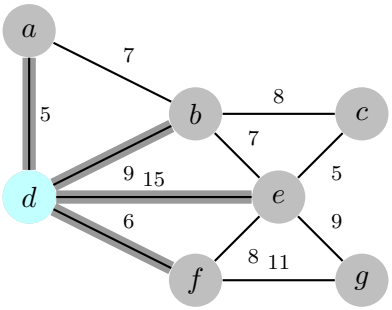
Algorithm 3: Prim Algorithmus

input : Graph G, Gewichtsfunktion w, Startknoten s
output : MST zu Graph G
parameter: $prev[u]$: Elternknoten von Knoten u im Spannbaum
$Adj[u]$: Adjazenzliste von u (alle Nachbarknoten)
$wert[u]$: Abstand von u zum entstehenden Spannbaum
$Q \leftarrow$ Prioritätswarteschlange
Schreibe alle Knoten V_G in Q
for *alle* $u \in Q$ **do**
    setze $wert[u] \leftarrow \infty$
    setze $prev[u] \leftarrow 0$
setze $wert[s] = 0$
while Q *nicht leer* **do**
    Wähle Knoten u aus Q mit $min(wert[u])$
    for $v \in Adj[u]$ **do**
        if $v \in Q$ *und* $w(u, v) < wert[v]$ **then**
            setze $prev[v] \leftarrow u$
            und $wert[v] \leftarrow w(u, v)$

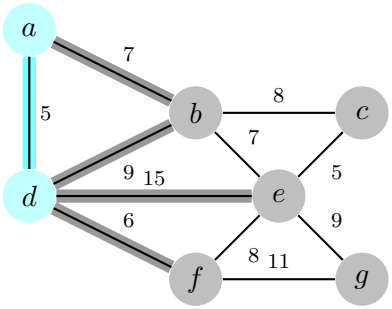
Ein Beispiel mit dem folgenden Graph beschreibt den Algorithmus wobei als Startknoten Knoten d verwendet wurde. Die grau eingefärbten Kanten sind die zur Auswahl stehenden und die mint gefärbten Kanten die dem MST hinzugefügten Kanten.



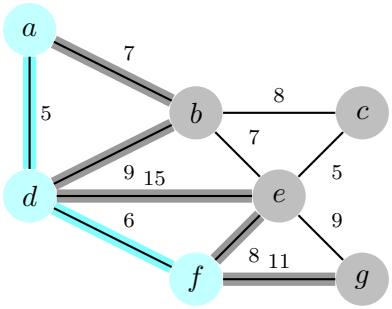
Originalgraph



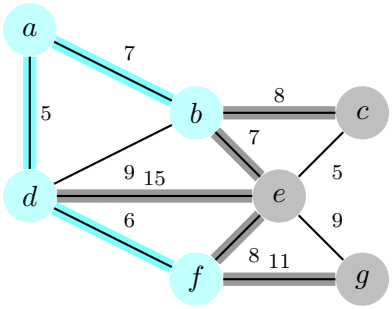
Startknoten d, erste Kantenauswahl



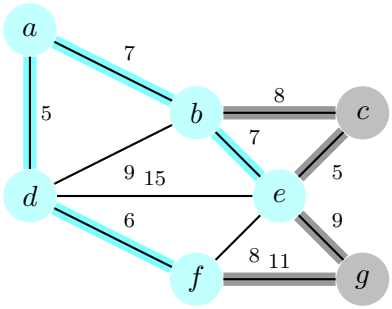
a hinzugefügt, neue Kantenauswahl



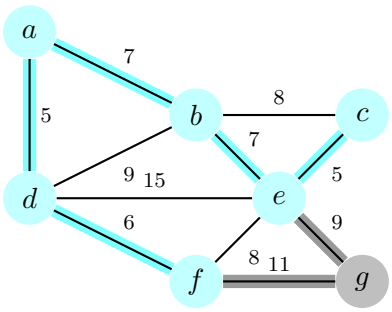
f hinzugefügt, neue Kantenauswahl



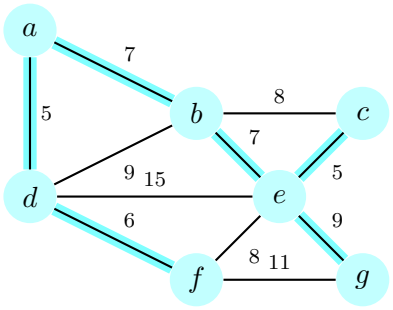
b hinzugefügt, neue Kantenauswahl



e hinzugefügt, neue Kantenauswahl



c hinzugefügt, neue Kantenauswahl



g hinzugefügt, MST vollständig

Der vollständige MST verbindet nun alle Knoten kostenminimal von Knoten *d* ausgehend. Die Kosten dieses MST betragen 39.

4.2.5 Kruskal

Dieser Algorithmus ist ebenfalls zum finden eines MST, arbeitet aber leicht anders als Prim. In diesem Fall wird kein Startknoten gewählt, sondern alle Kanten zu Beginn der Größe nach sortiert und immer die (dem Kantengewicht nach) kleinste Kante als nächste gewählt, sofern mindestens einer der Knoten noch nicht mit dem MST verbunden wurde. Auch hierzu sehen Sie in Algorithmus 4 den Pseudocode,

den Sie zur Implementierung verwenden können.

Algorithm 4: Kruskal Algorithmus

input : Graph G, Gewichtsfunktion w

output : MST zu Graph G

parameter: $E(1)$ Kantenmenge der in MST enthaltenen Kanten

$E(2)$ Kantenmenge der noch zu bearbeitenden Kanten

$E_1 \leftarrow 0$

$L \leftarrow E$

Sortiere die Kanten in L aufsteigend nach ihrem Kantengewicht.

while $L \neq 0$ **do**

  Wähle eine Kante $e \in L$ mit kleinstem Kantengewicht

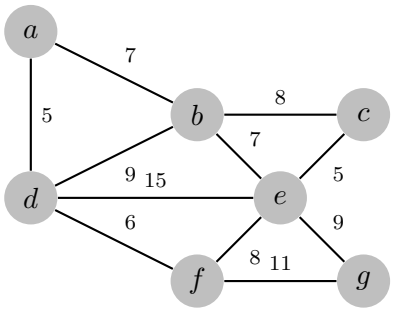
  Entferne die Kante e aus L

  if *Graph* $(V, E_1 \cup \{e\})$ *enthält keinen Kreis* **then**

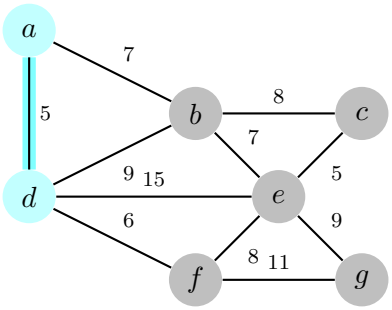
    $E_1 \leftarrow E_1 \cup \{e\}$

Die folgenden Abbildungen zeigen am gleichen Beispiel wie bei Prim den Ablauf des Algorithmus. Die Kanten wurden initial wie folgt sortiert:

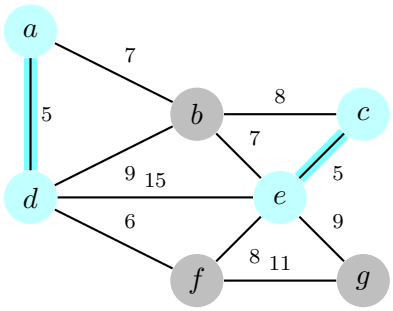
$(a, d, 5)$ $(c, e, 5)$ $(d, f, 6)$ $(a, b, 7)$ $(b, e, 7)$ $(b, c, 8)$ $(e, f, 8)$ $(b, d, 9)$ $(e, g, 9)$ $(f, g, 11)$ $(d, e, 15)$



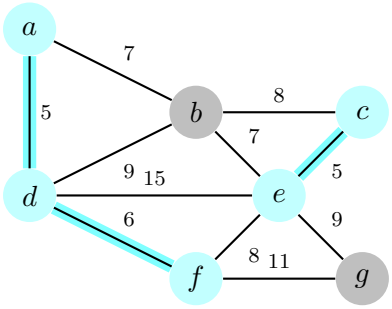
Originalgraph



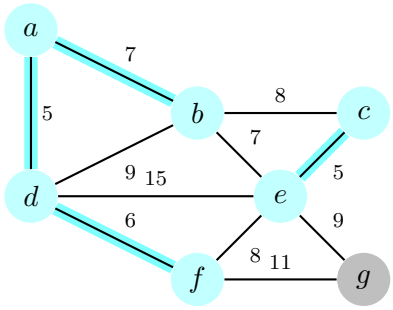
Kante $(a, d, 5)$ hinzugefügt



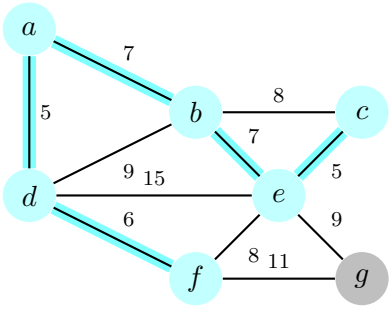
Kante $(c, e, 5)$ hinzugefügt



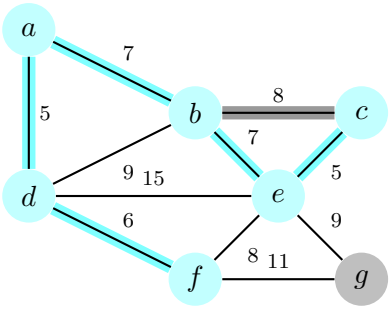
Kante $(d, f, 6)$ hinzugefügt



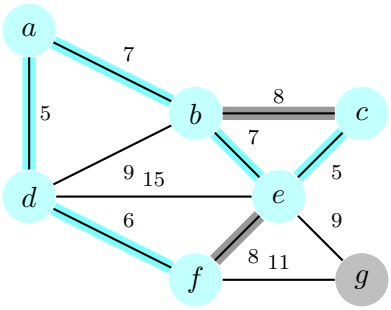
Kante $(a, b, 7)$ hinzugefügt



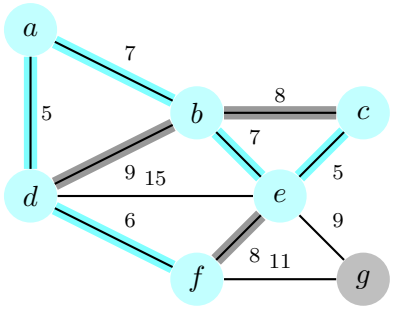
Kante $(b, e, 7)$ hinzugefügt



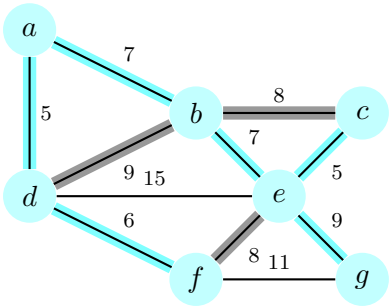
Kante $(b, c, 8)$ verwerfen



Kante $(e, f, 8)$ verwerfen



Kante $(b, d, 9)$ verwerfen



Kante $(e, g, 9)$ hinzufügen

Nachdem Knoten g hinzugefügt wurde ist der MST vollständig und es müssen keine weiteren Kanten mehr betrachtet werden. Die Gesamtkosten berechnen sich durch die Summe aller verwendeten Kante. In diesem Fall wurden exakt die gleichen Kanten wie beim Prim Algorithmus verwendet und somit beträgt die Summe des MST auch hier 39.

5 Praktikum 5 : Verlustfreie Kompression von Daten

Da Sie bisher so hervorragend gearbeitet haben, werden Sie mit der Aufgabe betraut sämtliche Firmendaten zu archivieren. Um den Speicherbedarf zu reduzieren, sollen die zu sichernden Dateien komprimiert werden. Dabei ist ein Komprimierungsverfahren anzuwenden, welches verlustfrei die Daten komprimiert.

Gedacht ist, zur Komprimierung, die Huffman-Codierung einzusetzen. Wie Ihnen bekannt ist werden dabei die Zeichen/Daten in der Datei untersucht und zunächst die Häufigkeit der Vorkommen von einzelnen Zeichen/Daten festgestellt. Danach wird über Bottom-Up, mit Hilfe der Wahrscheinlichkeiten, ein Binärbaum erzeugt. Mit Hilfe dieses Binärbaumes kann dann der komprimierte Code erzeugt werden.

Da auf den Servern sehr viele Bilddateien gespeichert sind, sollen diese als erstes zur Archivierung aufbereitet werden.

5.1 Aufgabenstellung

1. In diesem Praktikum wird Ihnen der gesamte Code vorgegeben - Sie müssen keine Anpassungen vornehmen (Der Beispiel-Code ist nur unter Windows lauffähig. Linux-User müssten auf unseren Terminal-Rechnern arbeiten, oder selbst Anpassungen an der CImg-Library vornehmen).
2. Laden Sie den Code in Ihr Visual-Studio und führen Sie ihn aus; beobachten Sie das Verhalten und erklären Sie die Funktionsweise des verwendeten Algorithmus.
3. Die "Implementierungsanweisungen" dienen hier nur dem Verständnis - Sie sollen die Arbeitsweise des Codes und des Algorithmus später erklären können.
4. Beantworten Sie die gestellten Fragen - schriftlich - vor Ihrer Abgabe.

Fragen zur Aufgabe

1. Welchen Zweck erfüllt die Huffman-Codierung?
2. Sie haben die Folge "abaac". Wie wenden Sie Huffman an? Welcher Baum und Code für die Zeichen entsteht? Wie würde die Folge als Code aussehen?
3. Erstellen Sie die Codetabelle für "ads hat viel zu bieten" und geben Sie den Text danach auf der Seite http://huffman.ooz.ie/ ein. Warum unterscheiden sich die Codes?
4. Wie groß ist die Einsparung des ersten Beispiels (abaac)? Zeigen Sie schriftlich, wie Sie zu diesem Wert kommen.
5. Zeigen Sie beispielhaft, wie Sie den ermittelten Code von "abaac" wieder dekodieren.

6. Sieht der Huffman-Code immer gleich aus?
7. Erklären Sie: Was ist das Codewort und was das Codebuch?
8. Wo findet man das Codebuch in dem gegebenen Programm?
9. Wir haben in dem Beispiel nur ein Bild mit Grauwerten. Wäre die Anwendung auf ein RGB-(Farb)-Bild möglich und bewerten Sie den Umfang.
10. Sie erhalten mit dieser Aufgabe eine Abbildung des Histogramms (hist.bmp). Welche Informationen können sie aus dem Histogramm ziehen?
11. Wie ist die Huffman-Codierung in Bezug auf die Effizienz und Geschwindigkeit, im Vergleich mit anderen Kompressionsverfahren, zu bewerten (googlen Sie mal) . Für welche Anwendung eignet er sich nicht?

5.2 Theorie

In dieser Aufgabe sollen Sie auf Bildern arbeiten. Zum besseren Verständnis, wird ihnen nun kurz erklärt, wie ein Bild aufgebaut ist und was ein Histogramm eines Bildes ist.

Mit einem Bild ist in diesem Kontext ein Farbbild gemeint. Das bedeutet, dass es aus drei Farbkanälen besteht, einem roten, einem grünen und einem blauen. Diese sogenannten RGB-Bilder kann man sich als drei übereinander gelegte Matrizen vorstellen. Jeder Pixel erhält aus jeder Matrix einen Wert für Rot, Grün und Blau. Diese Werte zusammen gemischt ergeben dann den Farbwert. Dargestellt werden diese Werte als Tripel z.B. durch $(255, 0, 0)$ (Rot). Das Tripel $(0, 0, 0)$ stellt schwarz dar und $(1, 1, 1)$ entsprechend weiß.

Wandelt man ein Bild nun um in ein Grauwertbild, so reduziert man das Bild auf eine einzelne Matrix in der Helligkeitsstufen gespeichert werden. Ein Grauwertbild enthält also keine Farbinformationen mehr, die einzelnen Pixel werden als Graustufen dargestellt.

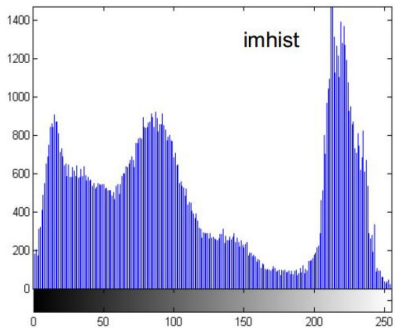
Ein Histogramm stellt nun die Häufigkeit jedes Farbwertes oder Grauwertes in einem Bild dar. Das bedeutet für jeden Wert (i.d.R 255 Werte) wird aufsummiert, wie oft er im Bild vorkommt. Man kann darüber den Kontrast eines Bildes analysieren.

Zur Bearbeitung steht Ihnen die aus GIP bekannte CImg Library zur Verfügung.

5.3 Verständnis - Implementierungsanweisung

1. Sie haben die Aufgabe eine Bilddatei einzulesen. Benutzen Sie dazu die Library CImg, die Sie bereits aus dem GIP-Praktikum kennen.

   Zur Vereinfachung der Aufgabenstellung nehmen wir ein Grauwert-Bild. Für dieses Bild ermitteln Sie zuerst die Auftrittswahrscheinlichkeiten der Grauwerte des Bildes.



Im zweiten Schritt schreiben Sie einen Algorithmus, der den sogenannten Huffman-Code zu jedem Grauwert liefert. Der Huffman-Code kann durch einen Binärbaum erzeugt werden. Dabei werden alle Wahrscheinlichkeiten als Blätter eines Baumes abgelegt. Der Binärbaum entsteht, indem immer die zwei kleinsten Wahrscheinlichkeiten aufaddiert werden und die entstehende Summe in einem neuen Eltern-Knoten (Bottom-Up) in dem Baum eingefügt werden. Als linker Nachfolger wird der größere Summand zugeordnet. Der Algorithmus zum Aufbau des Binärbaumes endet, wenn ein Knoten mit der Summe 1.0 als Wurzelknoten eingefügt wird. Die Pfade von der Wurzel zu den Blättern liefern dann die Codewörter für die Grauwerte. Dabei erhalten häufig vorkommende Grauwerte einen kurzen Code und selten vorkommende Grauwerte ein langes Codewort.

Zur Speicherung verwenden Sie eine binäre Datei. Zuerst werden die Häufigkeiten der Blätter - größensortiert - in die Datei geschrieben.

Danach speichern sie den durch die Grauwerte entstehen Huffman-Bitstrom in die Datei.

2. Sie erstellen ein Programm welches die von Ihnen erzeugte Datei einliest und das ursprüngliche Bild wieder herstellt.

Lesen Sie zunächst die gespeicherten Häufigkeiten aus der Datei und erzeugen daraus wieder den Binärbaum. Jetzt können sie den Bitstrom auslesen und mit Hilfe des Binärbaumes wieder die ursprünglichen Grauwerte erzeugen.

Zeigen Sie Ergebnis auf dem Bildschrim als Bild an. Verwenden Sie dazu wieder die CImg-Library.

5.4 Verständnis - Implementierungsanweisung - Lösungshinweise

- Definieren Sie zunächst eine geeignete Datenstruktur für den Knoten des Binärbaumes.
- Erzeugen Sie eine Prioritätswarteschlange mit den Blattknoten des Binärbaumes. Verwenden Sie hierzu die STL.
- Entnehmen Sie immer zwei Elemente aus der Prioritätswarteschlange und erzeugen einen neuen Knoten und fügen den neuen Knoten in die Prioritätswarteschlange ein.
- Wiederholen Sie den letzten Schritt, bis nur noch 1 Knoten in der Prioritätswarteschlange vorhanden ist. Geben Sie die Adresse als Wurzelknoten zurück.
- Verwenden Sie als Datenstruktur zur Speicherung der Codewörter zu jedem Grauwert eine map aus der STL. Welche Datenstruktur sind hier der primäre und der sekundäre Schlüssel, was bedeuten diese in Bezug auf die Codewörter?
- Traversieren Sie den Baum Bottum-Up und erzeugen Sie für jedes Blatt bzw. für jeden Grauwert einen Vektor vom Datentyp bool. Ein "bool"repräsentiert 1 Bit.
- Immer 8 dieser bool-Werte speichern Sie bitweise in unsign char ab. Diese werden dann in die Datei geschrieben.