
Programmieren, Algorithmen und Datenstrukturen II

Praktikum 5: Operatorüberladung und Abstrakte Datentypen

Sommersemester 2019

Prof. Dr. Arnim Malcherek

Allgemeine Hinweise zum Praktikum:

- Bereiten Sie die Aufgaben unbedingt zu Hause oder in einem freien Labor vor. Das beinhaltet:
 - Entwurf der Lösung
 - Codieren der Lösung in einem Qt-Creator-Projekt
- Die Zeit während des Praktikums dient dazu, die Lösung testen zu lassen sowie eventuelle Korrekturen vorzunehmen.
- Das Praktikum dient auch zur Vorbereitung der praktischen Klausur am Ende des Semesters. Versuchen Sie also in Ihrem eigenen Interesse, die Aufgaben selbstständig nur mit Verwendung Ihrer Unterlagen bzw. Ihres bevorzugten C++-Buches und ohne Codefragmente aus dem Netz zu lösen.
- Die Lösung wird nur dann testiert, wenn
 - sie erklärt werden kann bzw. Fragen zur Lösung beantwortet werden können.
 - das Programm ablauffähig und die Lösung nachvollziehbar ist.
 - die Hinweise oder Einschränkungen aus der Aufgabenstellung befolgt wurden.
- Zur Erinnerung hier noch einmal die Regeln des Praktikums, die schon in der Vorlesung besprochen wurden:
 - Sie arbeiten in 2er Gruppen.
 - Ein Testat gibt es nur zum jeweiligen Termin.
 - Abschreiben und Kopieren ist verboten.
 - Es gibt keine Noten. Die Bewertung ist lediglich erfolgreich / nicht erfolgreich.
 - Das Praktikum ist Zulassungsvoraussetzung für die Klausur. Hierfür müssen alle fünf Praktikumsübungen testiert sein.

Lernziele:

- Sie sind in der Lage abstrakte Datentypen als Klassen aufzubauen und solche Datentypen einzusetzen.
- Sie können Operatoren überladen.
- Sie können grundlegende Graphen-Algorithmen aus der Vorlesung sinnvoll anwenden.

Aufgabe 1

Ersetzen Sie die zentrale Ablage `vector <Booking*> allBookings` durch eine sortierte doppelt verkettete Liste. Sortierkriterium soll die Reisennummer (`travel.id` sein). Erstellen Sie dazu eine geeignete Klasse `List`. Verwenden Sie nicht die fertigen Datentypen aus der STL oder Qt, sondern Ihre selbst erstellte Klasse `List` nach dem Vorbild aus der Vorlesung.

Überladen Sie für die Klasse `List` den Operator `[]`, so dass Sie auch für die Liste einen Indexzugriff ausführen können. Bei Zugriff auf einen ungültigen Index soll eine geeignete Ausnahme geworfen werden.

Verallgemeinern Sie Ihre Klasse `List` zu einem Klassentemplate und verwenden Sie in Ihrem Programm die Klasse `List <Booking*>`. Falls Sie bisher die Funktion `at()` statt des Operators `[]` verwendet haben, können Sie diese zusätzlich implementieren. Auf diese Art und Weise müssen Sie am Rest Ihres Programmes nichts ändern.

Aufgabe 2

Das Reisebüro erhält recht häufig Beschwerden über falsche oder fehlende Buchungen. Die Mitarbeiter möchten daher Prüfungen in der Software haben, deren Ergebnisse sie auf mögliche fehlerhafte Buchungen hinweisen. Um diese Prüfungen zu ermöglichen, wurden den Buchungen Anordnungsbeziehungen hinzugefügt, mit denen sie in eine topologische Ordnung gebracht werden können. Dazu wurden in der Datei `bookings5.txt` neue Spalten am Ende eingefügt, in denen die Nummern der Vorgängerbuchungen eingetragen sind. Die Anzahl dieser Spalten variiert von 0 (keine Vorgängerbuchung) bis 2 (zwei Vorgängerbuchungen). (Mehr als zwei zusätzliche Spalten können nicht auftreten.)

Beispiel (i):

```
F|25|167.01|20150301|20150301|4|1|Norbert Eilenfeldt|FRA|VIE|Austrian Airlines|A
R|26|65.01|20150301|20150302|4|1|Norbert Eilenfeldt|Wien-Schwechat Flughafen|Wien-Schwechat Flughafen|Sext|Haftpflicht|25
H|27|199.99|20150301|20150302|4|1|Norbert Eilenfeldt|Hotel Kummer|Wien|0|25
F|28|167.01|20150302|20150302|4|1|Norbert Eilenfeldt|VIE|FRA|Austrian Airlines|A|26|27
```

In diesem Beispiel sind das Hotel und die Mietwagenbuchung vom Hinflug (Nummer 25) abhängig. Der Rückflug (Nummer 28) ist sowohl vom Hotel (Nummer 27) als auch von der Mietwagenbuchung (Nummer 26) abhängig.

Beispiel (ii):

```
F|64|145.00|20150313|20150313|13|2|Roland Peitsch|MHG|HAM|Rhein-Neckar Air|A
R|65|765.11|20150313|20150321|13|2|Roland Peitsch|Hamburg Flughafen|Aarhus Flughafen|Europcar|Haftpflicht|64
H|66|424.04|20150313|20150317|13|2|Roland Peitsch|Thomas Hotel Spa & Lifestyle|Husum|0|64
H|67|413.03|20150317|20150321|13|2|Roland Peitsch|Radisson Blu Scandinavia|Aarhus|0|66
F|68|245.00|20150321|20150321|13|2|Roland Peitsch|AAR|BER|Scandinavian Airlines|A|65|67
F|69|165.00|20150321|20150321|13|2|Roland Peitsch|BER|MHG|Rhein-Neckar Air|A|68
```

Hinweis: Ihre Fehlerverarbeitung aus dem letzten Praktikum müssen Sie jetzt wieder etwas aufweichen: Für Hotels können zwischen 11 und 13 Spalten auftreten, für die anderen beiden Typen zwischen 12 und 14.

Überlegen Sie sich, wie Sie die zusätzlichen Informationen beim Einlesen speichern. Legen Sie anschließend für jede Reise einen gerichteten Graphen an. Die Knoten dieses Graphen sind Zeiger

auf die Buchungen, die Kanten sind durch die Abhängigkeiten gegeben. Erweitern Sie dazu Ihre Klasse `Travel` um ein geeignetes Attribut, in dem Sie diesen Graphen abspeichern können.

Nutzen Sie jetzt die in Moodle (Coding Kapitel 8) abgelegten Graphen-Algorithmen um die folgenden Prüfungen zu programmieren.

1. Prüfung auf gleichen Abflug- und Ankunftsort in der Methode `bool Travel::checkRoundtrip()`

Diese Methode soll mit Hilfe topologischer Sortierung prüfen, dass der Startflughafen des ersten Fluges der Reise gleich dem Zielflughafen des letzten Fluges der Reise ist. Falls die Bedingung erfüllt ist, wird `true` zurückgegeben, sonst `false`.

Beispiel:

- Flug von Frankfurt nach Paris am 25.11.2015
- Hotel gebucht in Paris vom 25.11.2015 bis zum 30.11.2015
- Weiterflug von Paris nach Wien am 30.11.2015
- Hotel gebucht in Wien vom 30.11.2015 bis zum 01.12.2015
- Flug von Wien nach Stuttgart am 01.12.2015

ergibt als Ergebnis `false`.

2. Methode `bool Travel::checkMissingHotel()`: Die gesamte Reise muss lückenlos mit Übernachtungsmöglichkeiten (entweder Hotels oder Übernachtflüge) abgedeckt sein.

Algorithmus:

- (a) Führen Sie eine topologische Sortierung durch
- (b) Im Ergebnis der topologischen Sortierung ignorieren Sie jetzt die Mietwagenbuchungen und prüfen, dass die Daten (`fromDate` und `toDate`) von Hotel- und Flugbuchungen entlang der Reihenfolge der topologischen Sortierung lückenlos sind.

Tipp: In der Beispielimplementierung der topologischen Sortierung aus der Vorlesung wurde ein Heap zum Sortieren verwendet. Entfernen Sie aus dem Heap in einer `while`-Schleife mit Hilfe der Funktion `pop()` so lange Einträge, bis er leer ist (`underflow exception`). In der `while`-Schleife prüfen Sie jedes Mal, ob das `toDate` des Vorgängers kleiner als das `fromDate` des Nachfolgers ist. Mietwagenbuchungen erkennen Sie durch `dynamic_cast` und überspringen sie mit `continue`.

Beispiel (i):

Anwendung der Prüfung auf

- Flug von Frankfurt nach Sao Paulo (über Nacht): Abflug 23.11.2015, Ankunft 24.11.2015
- Weiterflug von Sao Paulo nach Porto Alegre am 24.11.2015
- Hotelbuchung vom 24.11.2015 (egal wo)

ergibt als Ergebnis `true`.

Beispiel (ii):

Anwendung der Prüfung auf

- Flug von Frankfurt nach Sao Paulo (über Nacht): Abflug 23.11.2015, Ankunft 24.11.2015
- Weiterflug von Sao Paulo nach Porto Alegre am 25.11.2015
- Hotelbuchung vom 25.11.2015 (egal wo)

ergibt als Ergebnis `false`, da zwischen Ankunft des Fluges und der Hotelbuchung eine Lücke von einem Tag vorhanden ist.

3. `bool Travel::checkNeedlessHotel()`: Der Algorithmus dieser Prüfung ist derselbe wie für `checkMissingHotel`, nur dass Sie jetzt nicht nach Lücken in den Daten suchen sondern nach Überlappungen. Achten Sie auch darauf, dass am Ende der Reise keine überflüssige Hotelbuchung vorkommt.

4. `bool Travel::checkNeedlessRentalCar()`: Gleicher Algorithmus wie für Hotels, nur für Mietwagen. Hier müssen Sie Hotelbuchungen überspringen.

Die Logik Ihrer Prüfungen soll so sein, dass die Prüfungen **false** ergeben, falls ein Fehler auftritt und **true**, wenn alles in Ordnung ist. Geben Sie das Ergebnis in einem geeigneten UI aus (ein Beispiel finden Sie in Abbildung 1, Sie können aber auch ein anderes UI wählen z.B. als Konsolenausgabe).

Reise	Fehlendes Hotel	Überflüssiges Hotel	Überflüssiger Mietwagen	Kein Roundtrip
4	👍	👍	👍	👍
1	👍	👍	👍	👍
2	👎	👍	👍	👍
3	👍	👍	👍	👍
5	👍	👍	👍	👍
7	👍	👍	👎	👍
8	👎	👍	👍	👍
11	👍	👍	👎	👍
9	👍	👍	👍	👍
10	👍	👍	👍	👍
12	👍	👍	👍	👍
13	👍	👎	👍	👍
14	👍	👍	👍	👎
15	👍	👍	👍	👍
16	👍	👍	👍	👍
17	👍	👍	👍	👍
6	👍	👎	👍	👍

Abbildung 1: Ergebnis der Konsistenzprüfungen

Achtung: Alle Ihre Programme müssen auch funktionieren, wenn Buchungsnummern nicht mehr aufsteigend vergeben werden, sondern zufällig sind. Um das zu testen, werden die Praktikumsbetreuer in Ihrer Datei `bookings_praktikum5.txt` einzelne Buchungsnummern vertauschen.

Beispiel:

```
F|67|145.00|20150313|20150313|13|2|Roland Peitsch|MHG|HAM|Rhein-Neckar Air|A
R|68|765.11|20150313|20150321|13|2|Roland Peitsch|Hamburg Flughafen|Aarhus Flughafen|Europcar|Haftpflicht|67
H|66|424.04|20150313|20150317|13|2|Roland Peitsch|Thomas Hotel Spa & Lifestyle|Husum|0|67
H|64|413.03|20150317|20150321|13|2|Roland Peitsch|Radisson Blu Scandinavia|Aarhus|0|66
F|65|245.00|20150321|20150321|13|2|Roland Peitsch|AAR|BER|Scandinavian Airlines|A|68|64
F|69|165.00|20150321|20150321|13|2|Roland Peitsch|BER|MHG|Rhein-Neckar Air|A|65
```

In der Datei sind für alle Checks Fehler eingebaut. Außerdem ist die Sortierreihenfolge in der Datei verändert, so dass die Buchungen einer Reise nicht mehr notwendigerweise aufeinander folgen.

Aufgabe 3

Diese Aufgabe enthält eine weitere Übung zum Dateieinlesen und ist optional.

Die Datei `airports.txt` enthält Texte und GPS-Koordinaten zu allen Flughäfen, die in Buchungen vorkommen. Erweitern Sie Ihr Klassenmodell um eine einfache Klasse `Airport` mit den privaten Attributen `QString code`

`QString name`

`double longitude`

`double latitude`

Legen Sie in der Klasse `TravelAgency` ein Attribut `airports` vom Typ `vector<Airport*>` oder `QVector<Airport*>` an. Erweitern Sie jetzt Ihre Methode `readFile` in `TravelAgency`, lesen Sie dort die Datei `airport.txt` ein und speichern Sie die Daten im Attribut `airports`. Nutzen Sie die Namen der Flughäfen statt der Kürzel auf allen UIs. Legen Sie dazu in `TravelAgency` eine `public` Methode `QString search(QString code)` an, die den Namen eines Flughafens zurückgibt.