TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe P4

PDF erstellt am: 15. Mai 2025

1 Vorbereitung

Um diese Programmieraufgabe erfolgreich durchführen zu können, müssen folgende Schritte umgesetzt werden:

1. Laden Sie das zugehörige Java-Projekt `P4.zip` aus TUWEL herunter.
2. Entpacken Sie `P4.zip` und öffnen Sie das entstehende Projektverzeichnis in Ihrer Entwicklungsumgebung.
3. Öffnen Sie die nachfolgend angeführte Datei in Ihrem Editor. In dieser Datei sind sämtliche Programmiertätigkeiten durchzuführen. Ändern Sie keine anderen Dateien im Projekt und fügen Sie auch keine neuen hinzu.
   `task/src/main/java/`
   `StudentSolutionForExerciseImplementation.java`
4. Füllen Sie Vorname, Nachname und Matrikelnummer in der Methode `StudentInformation provideStudentInformation()` aus.
5. Gehen Sie sicher, dass Java-Version 21 auf Ihrem System verfügbar ist.

   Alternativ zu einer manuellen Installation, können Sie in der Datei `settings.gradle` das auskommentierte Code-Snippet für eine automatische Installation nutzen.
6. Details zur Ausführung des Codes finden Sie in Abschnitt 6.

2 Hinweise

Einige Hinweise, die Sie während der Umsetzung dieser Aufgabe beachten müssen:

- Lösen Sie die Aufgaben selbst. Verwenden Sie auch keine (KI)-Tools z.B. zur Codegenerierung.
- Sie dürfen beliebig viele Hilfsmethoden schreiben und benutzen. Beachten Sie aber, dass Sie nur die oben geöffnete Datei abgeben und diese Datei mit dem zur Verfügung gestellten Framework lauffähig sein muss.
- Es ist grundsätzlich erlaubt Bibliotheken wie z.B. das Java Collections Framework und die Java Math Klasse zu verwenden. Die Voraussetzung dabei ist, dass die Verwendung die eigentliche Programmieraufgabe nicht abnimmt. Beachten Sie, dass nur die oben genannte Java-Version auf unserem Testsystem zur Verfügung steht.

3 Übersicht

Diese Programmieraufgabe bietet Ihnen die Möglichkeit einige der in der Vorlesung vorgestellten Hashfunktionen und Sondiermethoden auszuprogrammieren. Im Anschluss wird die Effizienz der Hashfunktionen bei unterschiedlichen Tabellengrößen untersucht.

4 Theorie

Die notwendige Theorie kann in den Vorlesungsfolien „Hashing“ gefunden werden.

5 Implementierung

In dieser Programmieraufgabe kommen sowohl die Divisions-Rest-Methode zum Einsatz, als auch mehrere Verfahren zur Kollisionsbehandlung. Zur Kollisionsbehandlung wird einerseits Verkettung der Überläufer eingesetzt

und werden andererseits offene Hashverfahren eingesetzt. Im Folgenden werden jene Methoden beschrieben, die Sie zur Vervollständigung der Aufgabe implementieren müssen. Schlüssel werden dabei nie mehrfach zum Einfügen angeboten.

5.1 Verkettung der Überläufer

Implementieren Sie Verkettung der Überläufer in der Methode `void insertVerkettung(HashTableWithChaining t, ChainElement chainElement, int m)`. Nutzen Sie dafür die folgende Hashfunktion: $h_1(k) = k \mod m$, wobei m der Tabellengröße entspricht.

Der Parameter `HashTableWithChaining t` enthält die Hashtabelle, in die ein neues Element abgelegt werden soll. `HashTableWithChaining` stellt vier Methoden zur Verfügung, die Sie zur Umsetzung benutzen dürfen:

- `boolean containsNoChainElement(int position)`: Gibt `true` zurück, wenn diese Stelle der Hashtabelle leer ist. Gibt es bereits ein Kettenelement, dann ist die Rückgabe `false`.
  Mittels
  `boolean firstPositionIsEmpty = t.containsNoChainElement(0)`
  kann so z.B. der Status der ersten Position der Hashtabelle abgefragt werden.

- `ChainElement get(int position)`: Gibt das Kettenelement der gegeben Position zurück. Enthält die Hashtabelle keine Kette an dieser Stelle, kommt es zu einer Fehlermeldung.
  Mittels `ChainElement element = t.get(0)` kann das erste Element der Kette erhalten werden, die an Stelle 0 der Hashtabelle beginnt.

- `void insertChainElement(ChainElement chainElement, int position)`: Setzt ein Kettenelement an der gegebenen Position ein. Enthält die Hashtabelle bereits ein Kettenelement an dieser Stelle, kommt es zu einer Fehlermeldung.
  Mittels `t.insertChainElement(chainElement, 0)` kann das Kettenelement `chainElement` an Position 0 hinterlegt werden.

- `void replaceChainElement(ChainElement chainElement, int position)`: Ersetzt ein Kettenelement durch ein neues an der gegebenen Position. Enthält die Hashtabelle noch kein Kettenelement an dieser Stelle, kommt es zu einer Fehlermeldung.

Mittels `t.replaceChainElement(chainElement, 0)` kann das Kettenelement an Position 0 durch `chainElement` ersetzt werden.

Der Parameter `ChainElement chainElement` enthält das einzusetzende Kettenelement. Der zugehörige Schlüssel ist ebenfalls in `chainElement` hinterlegt. Folgende Methoden werden zur Verwendung bereitgestellt:

- `int getKey()`: Gibt den zugehörigen Integer-Schlüssel zurück.
  Mittels `int key = chainElement.getKey()` kann der Schlüssel abgefragt werden.
- `void setNext(ChainElement chainElement)`: Setzt das nachfolgende Kettenelement.
  Mittels `chainElement.setNext(successor)` kann das Kettenelement `successor` als Nachfolger von `chainElement` in `chainElement` hinterlegt werden.

Der Parameter `int m` repräsentiert die Größe der Hashtabelle.

Einen Rückgabewert gibt es nicht.

Testen Sie Ihre Lösungen mit der Testinstanz `insert-verkettung.csv` bevor Sie fortfahren. Informationen über das Testen können Sie im Abschnitt 6 erhalten.

5.2 Lineares Sondieren, Quadratisches Sondieren und Double Hashing

Implementieren Sie zunächst die entsprechenden Sondierungsfunktionen in den Methoden

- `int linearesSondieren(int key, int i, int m)`,
- `int quadratischesSondieren(int key, int i, int m)` und
- `int doubleHashing(int key, int i, int m)`.

Dabei liegt jeweils die Hashfunktion $h_1(k) = k \mod m$ zugrunde, wobei m der Tabellengröße entspricht. Die Konstanten c_1 und c_2 für das quadratische Sondieren sind beide 0.5. Die zweite Hashfunktion für Double Hashing ist $h_2(k) = 1 + (k \mod 5)$.

Alle Methoden benutzen die gleichen Parameter. `int key` repräsentiert den einzusetzenden Key, `int i` den Sondierungsschritt mit $i \in [0, m-1]$ und `int m` die Größe der Hashtabelle.

Als Rückgabewert wird die, durch das Sondieren ermittelte, Tabellenposition erwartet.

Testen Sie Ihre Lösungen mit den Testinstanzen `lineares-sondieren.csv`, `quadratisches-sondieren.csv` und `double-hashing.csv` bevor Sie fortfahren. Informationen über das Testen können Sie im Abschnitt 6 erhalten.

Vervollständigen Sie nun im nächsten Schritt die Methode `void insert(HashTable t, Probe p, int key, int m)`. Hierbei sollen Sie Schlüssel mithilfe (einer) der drei zuvor implementierten Sondierungsfunktionen in die Hashtabelle einsetzen. Dabei müssen Sie Ihre zuvor implementierten Methoden nicht selbst aufrufen.

Der Parameter `HashTable t` enthält die Hashtabelle, in die ein neues Element abgelegt werden soll. `HashTable` stellt vier Methoden zur Verfügung, die Sie zur Umsetzung benutzen dürfen:

- `boolean isFree(int position)`: Gibt `true` zurück, wenn diese Stelle der Hashtabelle leer ist. Ist bereits ein Schlüssel hinterlegt, dann ist die Rückgabe `false`.
  Mittels `boolean firstPositionIsEmpty = t.isFree(0)` kann so z.B. der Status der ersten Position der Hashtabelle abgefragt werden.

- `int get(int position)`: Gibt den Schlüssel der gegeben Position zurück. Enthält die Hashtabelle keinen Schlüssel an dieser Stelle, kommt es zu einer Fehlermeldung.
  Mittels `int key = t.get(0)` kann der Schlüssel erhalten werden, der an Stelle 0 der Hashtabelle beginnt.

- `void insert(int key, int position)`: Setzt einen Schlüssel an der gegebenen Position ein. Enthält die Hashtabelle bereits ein Kettenelement an dieser Stelle, kommt es zu einer Fehlermeldung.
  Mittels `t.insert(key, 0)` kann der Schlüssel `key` an Position 0 hinterlegt werden.

- `void replace(int key, int position)`: Ersetzt den Schlüssel durch einen neuen Schlüssel an der gegebenen Position. Enthält die Hashtabelle noch keinen Schlüssel an dieser Stelle, kommt es zu einer

Fehlermeldung.
Mittels `t.replace(key, 0)` kann der Schlüssel an Position 0 durch `key` ersetzt werden.

Der Parameter `Prope p` enthält die zu verwendende Sondierungsfunktion. Sie enthält nur die Methode `int evaluate(int key, int i)`, mit der basierend auf dem Schlüssel `key` und dem Sondierungsschritt `i` eine Position errechnet werden kann. Beispielsweise liefert `p.evaluate(key, 0)` die Position für den Schlüssel `key` im ersten Sondierungsschritt. Beachten Sie die gültigen Werte für `i` sowie die dadurch begrenzte Sonderiungsreihenfolge.

Der Parameter `int key` ist der einzusetzende Schlüssel.

Der Parameter `int m` repräsentiert die Größe der Hashtabelle.

Einen Rückgabewert gibt es nicht.

Testen Sie Ihre Lösungen mit den Testinstanzen `insert-lineares-sondieren.csv`, `insert-quadratisches-sondieren.csv` und `insert-double-hashing.csv` bevor Sie fortfahren. Informationen über das Testen können Sie im Abschnitt 6 erhalten.

5.3 Verbesserung nach Brent

Implementieren Sie die Verbesserung nach Brent in der Methode `void insertVerbesserungNachBrent(HashTable t, int key, int m)`. Nutzen Sie dafür die folgende Hashfunktion: $h_1(k) = k \mod m$, wobei m der Tabellengröße entspricht. Als zweite Hashfunktion (wie bei Double Hashing) kommt $h_2(k) = 1 + (k \mod 5)$ zum Einsatz. Die Parameter der Methode entsprechen den oben beschriebenen Parametern aus `void insert(HashTable t, Probe p, int key, int m)`.

Einen Rückgabewert gibt es nicht.

Testen Sie Ihre Lösungen mit der Testinstanz `insert-verbesserung-nach-brent.csv` bevor Sie fortfahren. Informationen über das Testen können Sie im Abschnitt 6 erhalten.

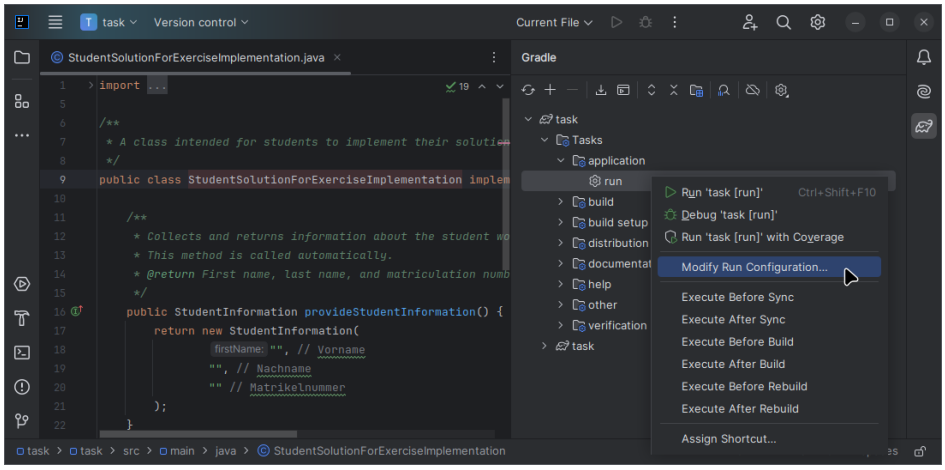

Abbildung 1: Ausführen von Gradle-Tasks in IntelliJ

6 Testen

Führen Sie im Projektverzeichnis den Befehl `./gradlew run` aus (`.\gradlew.bat run` unter Windows).

Sollten Sie IntelliJ nutzen, können Sie das Projekt auch direkt über die grafische Oberfläche starten. Klicken Sie dazu in der rechten Leiste auf das Gradle-Logo (Elefant) und navigieren Sie zum „run“-Task (siehe Abbildung 1). Mittels Rechtsklick bzw. Sekundärklick können Sie das Projekt ausführen („Run“) oder auch den visuellen Debugger nutzen („Debug“). Die Option „Modify Run Configuration...“ erlaubt es Ihnen, die Ausführung permanent als „Run/Debug Configuration“ zu hinterlegen.

Anschließend wird Ihnen in der Konsole eine Auswahl an Testinstanzen angeboten, darunter befindet sich zumindest `abgabe.csv`:

```
Select an instance set or exit:
[1] abgabe.csv
[0] Exit
```

Durch die Eingabe der entsprechenden Ziffer kann entweder eine Testinstanz ausgewählt werden oder das Programm (mittels der Eingabe von `0`) verlassen werden. Wird eine Testinstanz gewählt, dann wird der von Ihnen implementierte Programmcode ausgeführt. Kommt es dabei zu einem Fehler, wird ein Hinweis in der Konsole ausgegeben.

Um nur eine bestimmte Subinstanz auszuführen, hängen Sie `:[Subinstanznummer]` an Ihre Auswahl an (z.B. `1:123`).

Relevant für die Abgabe ist das Ausführen der Testinstanz `abgabe.csv`.

Die weiteren Testinstanzen `lineares-sondieren.csv`, `quadratisches-sondieren.csv`, `double-hashing.csv`, `insert-verkettung.csv`, `insert-lineares-sondieren.csv`, `insert-quadratisches-sondieren.csv`, `insert-double-hashing.csv` und `insert-verbesserung-nach-brent.csv`. sind nur zum jeweiligen Testen der einzelnen Unteraufgaben gedacht.

7 Evaluierung

Wenn der von Ihnen implementierte Programmcode mit der Testinstanz `abgabe.csv` ohne Fehler ausgeführt werden kann, dann wird nach dem Beenden des Programms im Ordner `results` eine Ergebnisdatei mit dem Namen `solution-abgabe.csv` erzeugt. Beachten Sie, dass Sie die gesamte Testinstanz ausführen und nicht nur eine Subinstanz!

Die Datei `solution-abgabe.csv` beinhaltet Zeitmessungen der Ausführung der Testinstanz `abgabe.csv`, welche in einem Webbrowser visualisiert werden können. (Auch Ergebnisdateien anderer Testinstanzen können zu Testzwecken visualisiert werden.) Öffnen Sie dazu die Datei `visualization.html` in Ihrem Webbrowser und klicken Sie rechts oben auf den Dateiauswahlknopf, um `solution-abgabe.csv` auszuwählen.

Beantworten Sie basierend auf der Visualisierung die Fragestellungen aus dem folgenden Abschnitt.

8 Fragestellungen

Öffnen Sie `solution-abgabe.csv` in `visualization.html` und bearbeiten Sie folgende Aufgaben- und Fragestellungen:

1. Erklären Sie die Funktionsweise Ihrer Implementierungen. Gehen Sie dabei besonders auf Ihre Umsetzung der Verkettung der Überläufer ein.

2. Beachten Sie die Ergebnistabellen in der Visualisierung. In diesen Tabellen wird die durchschnittliche Anzahl der notwendigen Sondierungsschritte (bzw. Schlüsselvergleiche bei Verkettung der

Überläufer) über alle Testinstanzen hinweg dargestellt, jeweils für erfolgreiche und erfolglose Suchen, für die Hashtabellengrößen 100, 101 und 102 sowie Befüllungsgrade 0.5, 0.9 und 0.95.

Welche Hashtabellengröße ist aus Ihrer Sicht die geeignetste Wahl? Berücksichtigen Sie dabei die Ergebnistabellen sowie die Theorie aus der Vorlesung.

Vergleichen Sie die Ergebnisse der gewählten Hashtabellengröße mit den theoretischen Berechnungen in den Vorlesungsfolien auf Seite 34 und 40. Wie beurteilen Sie die Ergebnisse der wenigen praktischen Tests, die in dieser Programmieraufgabe durchgeführt wurden, in Anbetracht der theoretischen Werte?

Erstellen Sie im Anschluss einen Screenshot auf dem alle Tabellen sichtbar sind.

3. Zirka 90% der Schlüssel, die eingefügt werden in jedem Testfall, sind gerade Zahlen. Welches Problem ergibt sich dabei bei Hashtabellen mit einer geraden Anzahl an Positionen?

   Wirkt sich diese Problematik auch auf die Messergebnisse aus? Betrachten Sie dazu die Ergebnisse der Hashtabellen mit den Größen 100 und 102.

   Gibt es Hashverfahren, die in dieser Programmieraufgabe stärker von dieser Problematik betroffen sind als andere? Wenn ja, nennen Sie sie und begründen Sie Ihre Auswahl.

4. Im Verzeichnis `keys` befinden sich Dateien mit jenen Schlüsseln, die in die Hashtabellen eingefügt werden. Untersuchen Sie stichprobenartig die Schlüssel, die eingefügt werden, und erklären Sie, warum eine Hashtabellengröße von 100 problematisch ist. Finden Sie dabei eine andere Eigenschaft, als jene in der Fragestellung 3.

   Vergleichen Sie die Ergebnisse der Hashtabellen mit den Größen 100 und 102. Beschreiben Sie, warum es bei einer Hashtabelle zu schlechteren Ergebnissen kommt, als bei der anderen.

Fügen Sie Ihre Antworten in einem Bericht gemeinsam mit dem Screenshot der Testinstanz `abgabe.csv` zusammen.

9 Abgabe

Laden Sie die Datei `task/src/main/java/StudentSolutionForExerciseImplementation.java` in der TUWEL-Aktivität *Hochladen Source-Code P4* hoch. Fassen Sie diesen Bericht mit den anderen für das zugehörige Abgabegespräch relevanten Berichten in einem PDF zusammen und geben Sie dieses in der TUWEL-Aktivität *Hochladen Bericht Abgabegespräch 2* ab.

10 Nachwort

Die besprochenen Hashtabellen haben eine sehr große Bedeutung in der Informatik, sie zählen zu den grundlegendsten und am häufigst verwendeten Datenstrukturen in nahezu jeder Programmiersprache. Schließlich lösen sie das zentrale Problem Information nach Schlüsseln organisiert abzuspeichern, wiederzufinden und ggfs. zu entfernen in praktisch konstanter Laufzeit. Zur Erinnerung: Balancierte Suchbäume ermöglichen diese Funktionalität „nur“ in logarithmischer Zeit in Abhängigkeit der Anzahl der Datensätze. Letztere bieten dafür aber auch die Möglichkeit in sortierter Weise auf die Datensätze zuzugreifen.

Abgesehen von der Anwendung in Hashtabellen haben Hashfunktionen jedoch auch eine zentrale Bedeutung in der Kryptographie. *Prüfsummen* (im Englischen auch *Fingerprints* genannt), sind beispielsweise errechnete Hashwerte um Übertragungsfehler festzustellen bzw. die Korrektheit von Daten zu prüfen. Manipulationssicherheit von Daten wird hierbei auch oft dadurch gewährleistet, dass die Hashfunktion für gegebene Daten zwar effizient berechnet werden kann, es aber praktisch sehr schwierig ist Daten so zu manipulieren, dass diese wiederum den gleichen Hashwert liefern, d.h., die Umkehrfunktion der Hashfunktion ist nur mit sehr großem Aufwand berechenbar. Zusammen mit entsprechenden Verschlüsselungsverfahren stellt dies auch die Basis von elektronischen Signaturen dar.

Nicht zuletzt sei auch die *Blockchain* erwähnt, die die Grundlage für die meisten Kryptowährungen wie Bitcoin darstellt. Transaktionsinformationen werden hierbei auch mit Hashverfahren abgesichert. Beim *Mining* geht es meist darum Daten zu finden, die einen bestimmten Hashwert ergeben, was eben wiederum nur mit entsprechend großer Rechenleistung bewerkstelligt werden kann. Mit den so gefundenen Daten werden neue Blöcke mit

Transaktionsinformationen erzeugt, die dann aber wiederum effizient durch die Hashfunktion allgemein überprüft werden können.