TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe P5

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 `P5.zip` aus TUWEL herunter.
2. Entpacken Sie `P5.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

Die Problemstellung ist, mehrere disjunkte Mengen aus einer Reihe vorgegebener Mengen auszuwählen, so, dass die ausgewählten Mengen zusammen alle Elemente der gegebenen Grundmenge enthalten. Sie werden allerdings keinen eigenen Lösungsalgorithmus entwerfen, sondern das Problem auf SAT reduzieren und mithilfe eines zur Verfügung gestellten SAT-Solvers lösen.

4 Theorie

Die Theorie für diese Programmieraufgabe befindet sich in den Vorlesungsfolien „Polynomialzeitreduktion“.

5 Implementierung

Gegeben ist eine Grundmenge $U = \{0, \ldots, n-1\}$ von Elementen und eine Menge $C \subseteq 2^U$ von Teilmengen von U. Aufgabe ist es, eine Teilmenge $S \subseteq C$

von C zu finden mit der Eigenschaft, dass jedes Element $u \in U$ in genau einer Menge in S enthalten ist. Eine solche Teilmenge S wird dann auch Exact Cover genannt. Ist beispielsweise $n = 4$ und $C = \{\{0, 1\}, \{1, 2, 3\}, \{2, 3\}\}$, dann ist $\{\{0, 1\}, \{2, 3\}\}$ die einzige gültige Lösung.

Führen Sie eine Polynomialzeitreduktion dieses Problems auf SAT durch, indem Sie eine passende aussagenlogische Formel in konjunktiver Normalform erstellen. Bilden Sie diese Formel im DIMACS-Format als `String` ab.

Beim DIMACS-Format besteht die erste Zeile aus `"p cnf n_var n_claus"`, wobei `n_var` durch die Anzahl der Variablen und `n_claus` durch die Anzahl der Klauseln von Ihnen ersetzt werden muss. Vergessen Sie nicht, nach jeder Zeile einen Zeilenumbruch `"\n"` anzuhängen. Jede weitere Zeile entspricht nun einer Klausel. Innerhalb einer Zeile halten Sie mit Abständen getrennt die in der Klausel vorkommenden Literale fest. Literale sind negierte oder nicht-negierte Variablen. Eine Variable können Sie durch eine beliebige natürliche Zahl, die größer als 0 und kleiner oder gleich `n_var` ist, repräsentieren. Entspricht ein Literal einer negierten Variable, können Sie dies durch ein vorangestelltes `"-"` ausdrücken. Jede Zeile einer Klausel muss mit `" 0"` beendet werden.

Die aussagenlogische Formel $(x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$ mit $n_{\text{var}} = 4$ Variablen und $n_{\text{claus}} = 2$ Klauseln wird im DIMACS-Format wie folgt dargestellt:

```
p cnf 4 2
1 -3 0
-1 2 4 0
```

Abbildung 1: Aussagenlogische Formel im DIMACS-Format

Implementieren Sie die Methode `boolean findExactCover(boolean[][] sets, Solver solver, boolean[] chosenSets)`.

Der erste Parameter `boolean[][] sets` repräsentiert die Menge $C = \{C_0, \ldots, C_{m-1}\} \subseteq 2^U$ von Teilmengen von U. Der erste Index von `boolean[][] sets` bestimmt die Menge aus C, der zweite das Element der Grundmenge U und der Wert bestimmt, ob das Element in der Menge enthalten ist. Ist also `sets[i][j] == true`, dann ist das Element $j \in U$ in der Menge C_i enthalten, d.h. $j \in C_i$. Ist hingegen `sets[i][j] == false`, dann ist das Element $j \in U$ in der Menge C_i nicht enthalten, d.h. $j \notin C_i$. Für beispielsweise $n = 4$ und $C = \{\{0, 1\}, \{1, 2, 3\}, \{2, 3\}\}$ ergibt sich das in der folgenden Tabelle dargestellte Array für `boolean[][] sets`.

	$j=0$	$j=1$	$j=2$	$j=3$
$i=0$	`true`	`true`	`false`	`false`
$i=1$	`false`	`true`	`true`	`true`
$i=2$	`false`	`false`	`true`	`true`

Tabelle 1: Array `sets[i][j]` für $n = 4$ und $C = \{\{0,1\},\{1,2,3\},\{2,3\}\}$.

Der zweite Parameter `Solver solver` beinhaltet den SAT-Solver. Dieser bietet eine Methode an:

- `String solve(String dimacs)`: Wenn Sie an diese Methode Ihre als DIMACS-String encodierte aussagenlogische Formel übergeben, versucht der SAT-Solver, eine erfüllende Variablenbelegung zu finden.

  Wurde eine gefunden, dann erhalten Sie einen `String` mit durch Leerzeichen getrennten Literalen. Negierte Variablen bedeuten, dass in der gefundenen Variablenbelegung diese Variablen *falsch* sind. Nicht-negierte Variablen implizieren *wahr*. Wie auch zuvor bei den Klauseln, wird auch dieser `String` durch `" 0"` beendet. Ein Resultat für die Formel in Abbildung 1 könnte z.B. `"-1 2 -3 -4 0"` sein. Das würde für die Variablenbelegung dann bedeuten, dass $x_1 = x_3 = x_4 = \mathit{falsch}$ und $x_2 = \mathit{wahr}$ gilt.

  Wurde keine Variablenbelegung gefunden, dann erhalten Sie einen `String` der Länge 0 als Rückgabe. Sollte es Probleme mit der DIMACS-Encodierung geben, dann erhalten Sie eine Fehlermeldung, die mit `"Parsing error in solver: "` beginnt.

Wichtig: Verwenden Sie entweder die Java-Klassen `StringBuilder`[1] oder `StringBuffer`[2] zum stückweisen Aufbau des DIMACS-Strings, und wandeln Sie diesen erst **am Ende** in ein Objekt des Typs `String` um, da sonst die Reduktion viel zu langsam wird. Beispielsweise enthält die Variable `String s` nach Ausführung des folgenden Codesegments den String `"Hello World."`.

```
StringBuilder sb = new StringBuilder();
sb.append("Hello ");
sb.append("World.");
String s = sb.toString();
```

[1] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/StringBuilder.html

[2] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/StringBuffer.html

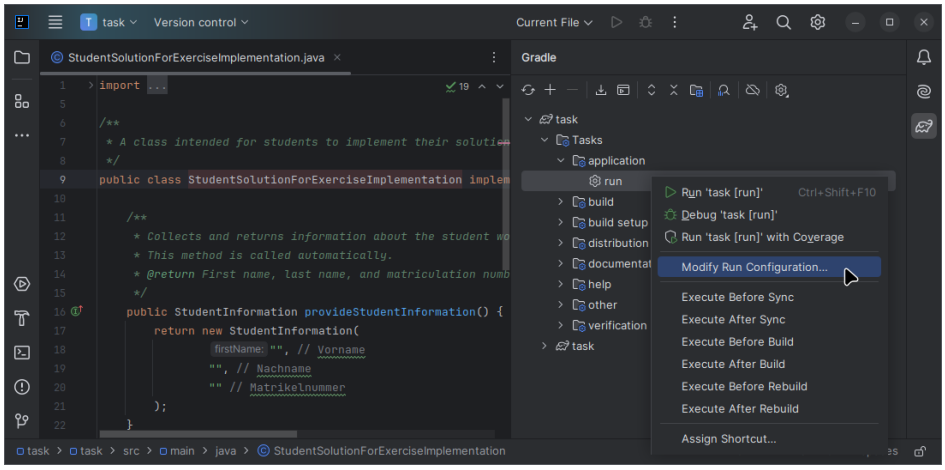

Abbildung 2: Ausführen von Gradle-Tasks in IntelliJ

Haben Sie ein Exact Cover mit den gegebenen Mengen gefunden, dann speichern Sie dieses im Parameter `boolean[] chosenSets` ab und geben `true` zurück. Rekonstruieren Sie die Lösung aus dem Ergebnis des SAT-Solvers. Der Parameter `boolean[] chosenSets` ist ein Array, das einen Eintrag für jede gegebene Menge aus C hat. Sollten Sie die Menge C_i aus C auswählen, dann setzen Sie auch den zugehörigen Wert `chosenSets[i]` auf `true`, andernfalls setzen Sie den Wert auf `false`. Sollte es mehrere Lösungen geben, wählen Sie eine beliebige aus.

Sollte es kein Exact Cover geben, lassen Sie den Parameter `boolean[] chosenSets` unverändert und geben `false` zurück.

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 2). 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 Testinstanz `abgabe.csv` sollte in höchstens ein paar Minuten durchlaufen, andernfalls ist etwas ineffizient implementiert, mögliche Fehlerquellen dafür sind:

- Reduktion per langsamen Aufbau über `String`-Verknüpfungen anstatt `StringBuilder`.
- Reduktion exponentiell anstatt polynomiell.

Hinweis. Für Debugging-Zwecke könnten die kleinen Instanzen aus `abgabe.csv` mit IDs `200`, `201` und `202` hilfreich sein.

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` und bearbeiten Sie folgende Aufgaben- und Fragestellungen:

1. Beschreiben Sie die von Ihnen implementierte Reduktion auf SAT und argumentieren Sie informell ihre Korrektheit.
2. Alle Instanzen sind zufällig erzeugt und haben eine Grundmenge mit 100 Elementen. Zum Erzeugen einer Menge wurde jedes Element der Grundmenge mit einer festgelegten Wahrscheinlichkeit ausgewählt. Eine Menge hat im Mittel etwa 4.5 Elemente. Die Anzahl der Mengen variiert von Instanz zu Instanz. In den Plots ist zu sehen, dass alle Instanzen bis etwa 300 Mengen keine Lösung haben und alle Instanzen ab etwa 400 Mengen eine Lösung haben. Was könnte ein möglicher Grund für diesen Umstand sein?
3. Betrachten Sie zuerst den oberen Plot, der die Reduktionszeit über die Instanzgröße zeigt. Welche asymptotische Laufzeitabhängigkeit von der Anzahl der Mengen lässt sich erkennen? Entspricht das Ihren Erwartungen? Warum bzw. warum nicht? Speichern Sie im Anschluss den Plot als Bild, indem Sie in der Menüleiste rechts über dem Plot auf den Fotoapparat drücken.
4. Betrachten Sie nun den unteren Plot, der die Gesamtzeit über die Instanzgröße zeigt. Beschreiben Sie, wie die Gesamtzeit von der Anzahl der Mengen abhängt. Wie erklären Sie sich diese Laufzeitabhängigkeit? Speichern Sie im Anschluss den Plot als Bild.

Falls sich im Zuge der Evaluierung die Darstellung der Plots auf ungewünschte Weise verändert (z.B. durch die Auswahl eines zu kleinen Ausschnitts), können Sie mittels Doppelklick auf den Plot oder Klick auf das Haus in der Menüleiste die Darstellung zurücksetzen.

Fügen Sie Ihre Antworten in einem Bericht gemeinsam mit den erstellten Bildern der Visualisierung der Laufzeiten der Testinstanz `abgabe.csv` zusammen.

9 Abgabe

Laden Sie die Datei `task/src/main/java/StudentSolutionForExerciseImplementation.java` in der TUWEL-Aktivität *Hochladen Source-Code P5* 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

Polynomialzeitreduktionen sind wichtige Fundamente sowohl der theoretischen als auch der praktischen Informatik. Wenn wir ein Problem A auf ein Problem B in Polynomialzeit reduzieren können, wissen wir, dass „A nicht schwerer als B“ sein kann. Somit übertragen sich komplexitätstheoretische untere Schranken (wie z.B. NP-Schwere) von A nach B. Umgekehrt, können wir effiziente Algorithmen für das Problem B auf A übertragen. Zum Beispiel, falls B das Erfüllbarkeitsproblem ist, und wir A mittels eines SAT-Solvers lösen wollen.

Es gibt mehrere Varianten von Reduktionen. Die Reduktionen, die in der Vorlesung besprochen wurden, wo genau einmal eine Instanz von A in eine Instanz von B übersetzt wird, werden *Karp-Reduktionen* genannt (weil sie von Richard Karp in seinem berühmten Artikel von 1972 verwendet wurden, siehe auch Vorlesungsfolien). Eine andere wichtige Variante sind die *Turing-Reduktionen*, auch *Cook-Reduktionen* genannt (nach Alan Turing und Stephen Cook). Hier verwendet ein Algorithmus, der das Problem A löst, immer wieder Aufrufe zum Problem B; d.h. der Algorithmus für Problem A generiert mehrmals Instanzen des Problems B und löst sie mittels eines „black box solvers“ (auch *Orakel* genannt). Diese Reduktionsvariante wird ebenfalls sowohl theoretisch für untere Schranken als auch praktisch zur Problemlösung verwendet. Falls sich bei Turing-Reduktionen die verschiedenen Instanzen des Problems B nur geringfügig unterscheiden, kann die Effizienz gesteigert werden, indem der Solver für B *inkrementell* arbeitet, d.h., nicht immer von vorne beginnt, sondern gewisse Teillösungen wieder verwendet.

Das Thema Reduktionen wird in zahlreichen (Master-)Lehrveranstaltungen näher behandelt. Eine unvollständige Liste ist: , 181.142 VU Complexity Theory, 184.090 VU SAT Solving and Extensions, 192.122 VU Algorithmic

Meta-Theorems, 186.861 VU Modeling and Solving Constrained Optimization Problems, 184.215 VU Complexity Analysis.