TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe P6

PDF erstellt am: 14. Juni 2022

1 Vorbereitung

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

1. Laden Sie das Framework `P6.zip` aus TUWEL herunter.
2. Entpacken Sie `P6.zip` und öffnen Sie das entstehende Verzeichnis als Projekt in IntelliJ (nicht importieren, sondern öffnen).
3. Öffnen Sie die nachfolgend angeführte Datei im Projekt in IntelliJ. In dieser Datei sind sämtliche Programmiertätigkeiten durchzuführen. Ändern Sie keine anderen Dateien im Framework und fügen Sie auch keine neuen hinzu.
   `src/main/java/exercise/StudentSolutionImplementation.java`
4. Füllen Sie Vorname, Nachname und Matrikelnummer in der Methode `StudentInformation provideStudentInformation()` aus.

2 Hinweise

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

- Lösen Sie die Aufgaben selbst und nutzen Sie keine Bibliotheken, die diese Aufgaben abnehmen.
- 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.

3 Übersicht

Diese Programmieraufgabe soll Ihnen dabei helfen, Ihr Wissen über dynamische Programmierung zu festigen, indem Sie selbst eine Aufgabe entsprechend implementieren. Im Laufe dieser Programmieraufgabe ist nun dasselbe Problem einmal mit einem naiven Ansatz und einmal mit dynamischer Programmierung zu lösen.

4 Theorie

Die notwendige Theorie kann in den Vorlesungsfolien „Optimierung – Dynamische Programmierung“ gefunden werden.

5 Implementierung

Im Rahmen des Betriebs eines Baumarktes wollen Sie Ihre Restbestände an Seilen veräußern. In einem Ihrer Lager haben Sie noch viele Stück Seile gefunden, mit bis zu 30 Metern Länge. Für den anstehenden Verkauf wollen Sie die Seile zurechtschneiden lassen. Aufgrund Ihrer langjährigen Aufzeichnungen können Sie recht genau abschätzen, welche Seillänge Sie zu welchem Preis gut verkaufen können. Sämtliche Informationen, die Sie zur Verfügung haben, nutzen Sie nun, um Ihre Restbestände an Seilen so zu unterteilen, dass Sie den Umsatz maximieren. Sie möchten herausfinden, mit welcher Stückzahl pro Seillänge Sie dieses Ziel erreichen.

5.1 Naiver Ansatz

Implementieren Sie zunächst einen naiven Ansatz zur Problemlösung in der Methode `void calculateRodFragmentationNaive(int length, int[] pricePerLength, int[] numberOfPiecesPerLength)`. Sie können dabei auf Brute-Force zurückgreifen.

Der erste Parameter `int length` gibt die Länge des zu teilenden Seils an. Aus dem zweiten Parameter `int[] pricePerLength` können Sie die zu erwartenden Einnahmen in Cent durch den Verkauf eines Stück Seils einer gewisse Länge auslesen. In Abbildung 1 sehen Sie, wie das Array im

Parameter `int[] pricePerLength` aussehen könnte. Indizes entsprechen den Seillängen in Metern und Werte an Positionen den zugehörigen Centbeträgen.

Index	0	1	2	3	4	5	6
Betrag	0	300	400	425	450	550	650

Abbildung 1: Beispiel für `int[] pricePerLength`

Der dritte Parameter `int[] numberOfPiecesPerLength)` dient zur Rückgabe der Lösung. Sie finden bereits ein initialisiertes Array vor, das die notwendige Länge hat. Hinterlegen Sie für jede Länge, die ermittelte Stückzahl. Die Indizes entsprechen wieder der Seillänge.

5.2 Dynamische Programmierung

Im nächsten Schritt lösen Sie das Problem mittels dynamischer Programmierung in der Methode `void calculateRodFragmentationDynProg(int length, int[] pricePerLength, int[] numberOfPiecesPerLength)`.

Sämtliche Parameter entsprechen exakt jenen aus Abschnitt 5.1. Nutzen Sie dabei die optimalen Lösungen von Teilproblemen aus.

6 Testen

Führen Sie zunächst die `main`-Methode in der Datei `src/main/java/framework/Exercise.java` aus.

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.

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

Die weiteren Testinstanzen `test-only-one-small.csv`, `test-only-one-medium.csv` und `test-only-one-large.csv` können Sie nutzen, falls Sie bisher nur eine der beiden Unteraufgaben implementiert haben und diese testen möchten. Wenn Sie in `int[] numberOfPiecesPerLength` nichts verändern, dann ignorieren die Testfälle dieses Ergebnis.

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 Ergebnis-Datei mit dem Namen `solution-abgabe.csv` erzeugt.

Die Datei `solution-abgabe.csv` beinhaltet Zeitmessungen und Ergebnisse der Ausführung der Testinstanz `abgabe.csv`, welche in einem Web-Browser visualisiert werden können. (Auch Ergebnis-Dateien anderer Testinstanzen können zu Testzwecken visualisiert werden.) Öffnen Sie dazu die Datei `visualization.html` in Ihrem Web-Browser und klicken Sie rechts oben auf den Knopf *Ergebnis-Datei auswählen*, 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. Wählen Sie im Dropdown links unter den Plots die Instanz *Naive 9* aus. Sie sehen nun den Wert einzelner Seillängen in Cent sowie das ursprüngliche Seil und die von Ihnen berechnete Unterteilung. Überprüfen Sie das Ergebnis und argumentieren Sie, wieso dies die beste Variante ist, das Seil zu teilen.

   Erstellen Sie im Anschluss einen Screenshot der Tabellen.

2. Wählen Sie im Dropdown nun *Dynamic Programming 9* aus. Überprüfen Sie auch hier das Ergebnis und argumentieren Sie, wieso dies die beste Variante ist, das Seil zu teilen. Stimmt das Ergebnis mit *Naive 9* überein?

   Erstellen Sie auch hier einen Screenshot der Tabellen.

3. Beschreiben Sie die Funktionsweise Ihrer Lösung durch den naiven Ansatz.

4. Welche Änderungen haben Sie vorgenommen, um eine Lösung mit dynamischer Programmierung zu erhalten?

5. Sehen Sie sich die Plots *Naive* und *Dynamic Programming* an. Welche Unterschiede stellen Sie bezüglich der Laufzeiten Ihrer Implementierungen fest? Erklären Sie Ihre Beobachtungen.

   Drücken Sie im Anschluss in der Menüleiste rechts über den Plots auf den Fotoapparat, um die Plots in einem Bild zu speichern.

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 PDF gemeinsam mit den beiden Screenshots und dem erstellten Bild der Visualisierung der Testinstanz `abgabe.csv` zusammen.

9 Abgabe

Laden Sie die Datei `src/main/java/exercise/StudentSolutionImplementation.java` in der TUWEL-Aktivität *Hochladen Source-Code P6* 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

Dynamische Programmierung ist eine sehr vielseitig einsetzbare Methode im Algorithmenentwurf. Sie kombiniert das Teile-und-Herrsche Prinzip und

die Rekursion, wobei das Zwischenspeichern von optimalen Lösungen unabhängiger Teilprobleme immer eine ganz zentrale Rolle spielt. Nur so lassen sich kostspielige Mehrfachberechnungen in den rekursiven Aufrufen vermeiden. Auch wenn wir die dynamische Programmierung in der Vorlesung im breiteren Kontext von NP-schweren Optimierungsproblemen behandelt haben, führt sie üblicherweise zu Polynomialzeitalgorithmen, deren Laufzeit einerseits von der Größe der Lösungstabelle abhängt und andererseits vom Zeitbedarf für die Berechnung eines einzelnen Tabelleneintrags.

In der Vorlesung haben wir bereits mehrere grundlegende Beispiele gesehen, wo sich dynamische Programmierung einsetzen lässt. In der Praxis gibt es aber viele weitere Beispiele für den Einsatz solcher Algorithmen, z.B. bei der Analyse von Genom- und Aminosäuresequenzen in der Bioinformatik, generell bei der Mustersuche in Texten, im Compilerbau bei der Erkennung kontextfreier Sprachen, bei der Optimierung des Seitenlayouts von Texten (z.B. in LaTeX) bzw. von Aufteilungs- oder Packungsproblemen (ähnlich dieser Programmieraufgabe), oder bei der Betrachtung von möglichen Spielzügen z.B. im Schach und ähnlichen Spielen. Weiters spielt die dynamische Programmierung eine wichtige Rolle bei der Berechnung von NP-schweren Graphenproblemen, falls die betrachteten Graphen die Eigenschaft der sogenannten beschränkten Baumweite (treewidth) haben. Das bedeutet, dass sie zwar keine Bäume mehr sein müssen, aber doch noch eine gröbere baumartige Gesamtstruktur haben. In diesem Fall lassen sich viele allgemeine NP-schwere Probleme durch bottom-up Verfahren auf einem zugehörigen Lösungsbaum in (parametrisierter) Polynomialzeit berechnen. Diese Konzepte lernen Sie im Masterstudium kennen, z.B. in den Lehrveranstaltungen *Algorithmics*, *Fixed-Parameter Algorithms and Complexity*, und *Structural Decompositions and Algorithms*.

In der Forschung verwenden wir dynamische Programmierung darüber hinaus auch oft bei geometrischen Optimierungsproblemen und in der Graphenvisualisierung, in denen sich Probleminstanzen durch optimal zu wählende geometrische Trennkurven in kleinere, unabhängige Teile zerlegen lassen aus denen sich dann die Gesamtlösung zusammensetzen lässt. Dies taucht z.B. bei der Berechnung von optimalen Beschriftungen von Landkarten oder technischen Zeichnungen mit durch Linien verbundenen Zusatzinformationen am Rand auf. Daher kann es gut sein, dass Sie später in Projekten, Seminaren, oder Abschlussarbeiten auch wieder mit dynamischer Programmierung zu tun haben werden. In jedem Fall gehört die dynamische Programmierung für Sie als künftige Informatiker_innen als eine wichtige Entwurfstechnik in Ihren algorithmischen Werkzeugkasten.