technische universität
dortmund

Fakultät für Informatik
Lehrstühle 11, 13, 14

Jun.-Prof. Dr. Maike Buchin
Łukasz Czajka, N. Kriege, D. Kurz, M. Ludewig

Sommersemester 2019
30. April

DAP2 Praktikum – Blatt 6

Abgabe: 13. Mai–17. Mai

Langaufgabe 6.1: Interval Scheduling (4 Punkte)

Implementieren Sie den Algorithmus *IntervalScheduling* aus der Vorlesung und testen Sie diesen auf den gegebenen Eingabedaten. Gehen Sie dazu wie folgt vor. Laden Sie die Datei `VorlagenBlatt06.zip` von der DAP 2 - Praktikumsseite herunter und entpacken Sie die Dateien in Ihr Arbeitsverzeichnis. Erstellen Sie eine Klasse `Interval`, die zwei Integer-Attribute verwaltet, mit folgenden Methoden:

- Einen Konstruktor, der zwei Integer-Parameter erwartet, einen für den Start des Intervals und den anderen für das Ende.
- Eine Methode `public int getStart()` und eine Methode `public int getEnd()`, die die jeweiligen Werte zurückgeben.
- Eine Methode `public String toString()`, die eine geeignete String-Repräsentation des Intervals zurückgibt.

Erstellen Sie eine Klasse `Anwendung` mit:

- Einer Methode `public static ArrayList<Interval> intervalScheduling (ArrayList<Interval> intervals)`, die den `IntervalScheduling`-Algorithmus aus der Vorlesung implementiert.
- Einer `main`-Methode, die als Parameter den Dateipfad einer Textdateien erwartet. Die Datei soll eingelesen werden und die Werte sollen in eine ArrayList der Klasse Interval geladen werden. Dabei sollen die möglichen Exceptions wie gewohnt explizit behandelt werden. Auf diesen Werte sollen Sie dann den Algorithmus ausführen und die Ergebnisse auf der Konsole ausgeben. Orientieren Sie sich bei der Ausgabe an den Dateien `sollAusgabeBsp.txt`, `sollAusgabeMittel.txt` und `sollAusgabeGross.txt`.

Langaufgabe 6.2: Lateness Scheduling (4 Punkte)

Für die Implementierung des LatenessScheduling-Algorithmus gehen Sie wie folgt vor. Erstellen Sie eine Klasse `Job`, analog zu der Klasse `Interval` aus Aufgabenteil 1. Statt des Startzeitpunktes soll die Klasse die Dauer und statt des Endzeitpunktes die Deadline des Jobs beinhalten. Erweitern Sie Ihre Klasse Anwendung wie folgt:

- Statt eines Parameters soll die `main`-Methode zwei übergeben bekommen. Der Erste soll von der Form "Interval" oder "Lateness" sein und für die Auswahl zwischen Interval- und LatenessScheduling sorgen. Der Zweite soll diesmal der Dateipfad sein.
- Benutzen Sie wieder die gleichen Dateien als Eingabe, allerdings sollen die Werte je nach Auswahlparameter entweder als Intervallgrenzen für das IntervalScheduling oder als Dauer und Deadline eines Jobs für das LatenessScheduling interpretiert werden.
- Schreiben Sie eine Methode `public static int[] latenessScheduling(ArrayList<Job> jobs)`, die den `LatenessScheduling`-Algorithmus aus der Vorlesung implementiert.
- Passen Sie Ihre Ausgabe entsprechend der Vorgaben an und geben Sie vor allem auch die maximale Verspätung mit aus.

Hinweise und Tipps

Einlesen einer Datei

Um in Java eine Text-Datei einzulesen, kann die Klasse `BufferedReader` benutzt werden. Zuerst erstellen Sie mittels des Dateipfades und der Klasse `FileReader` ein Objekt der Klasse `BufferedReader`. Mit der Methode `readLine()` können Sie nun jeweils die nächste Zeile aus der Datei einlesen. Die vorgegebenen Dateien sind in jeder Zeile so formatiert, dass zwei Integer Zahlen nur mit einem Komma getrennt sind. Um aus einer Zeile die beiden Integer zu parsen, kann die Klasse `StringTokenizer` benutzt werden. Im Konstruktor erwartet die Klasse als erstes den zu trennenden String und als zweites einen String mit den Zeichen, welche für die Trennung sorgen. Mit der Funktion `nextToken()` kann der jeweils nächste Teilstring rausgeholt werden. Beachten Sie die möglichen Exceptions die hierbei geworfen werden können. Diese sollen abgefangen werden und durch eine geeignete Ausgabe beschrieben werden. Weitere Informationen finden Sie in der Dokumentation der Java-API unter:

`https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`
`https://docs.oracle.com/javase/8/docs/api/java/io/FileReader.html`
`https://docs.oracle.com/javase/8/docs/api/java/util/StringTokenizer.html`

Beispiel:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.StringTokenizer;
...
BufferedReader file = new BufferedReader( new FileReader( path ) );
...
    String zeile = file.readLine();
```

```
    StringTokenizer st = new StringTokenizer(zeile,",");
    int start = Integer.parseInt(st.nextToken());
    int ende = Integer.parseInt(st.nextToken());
    Interval ivall = new Interval(start, ende);
...
```

ArrayList

Die Klasse ArrayList wird mittels `import java.util.ArrayList;` für Sie verfügbar. Die wichtigsten Methoden sind:

- add(E e) zum Einfügen eines Elements.
- isEmpty() zur Überprüfung ob die Liste leer ist.
- size() gibt die Größe der ArrayList zurück.
- get(int index) um das Element an Position index der Liste übergeben zu bekommen.
- toString() um die Liste auszugeben, wobei wiederum die toString-Methode enthaltener Objekte genutzt wird.

Detailliertere Dokumentationen dieser API finden Sie unter:

`https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html`

Sortieren von Objekten

Objekte einer selbst geschriebenen Klasse können mit Hilfe der Methode `Collections.sort` sortiert werden, wenn die zu sortierende Klasse das `Comparable`-Interface implementiert. Dafür müssen Sie (am Beispiel der Klasse `Interval`) eine Methode `public int compareTo(Interval other)` erstellen, die:

- einen negativen Wert zurückgibt, wenn `other` größer als das Objekt ist,
- Null zurückgibt, wenn `other` genau so groß ist wie das Objekt, und
- einen positiven Wert zurückgibt, wenn `other` kleiner als das Objekt ist.

Weitere Informationen finden Sie in der Dokumentation der Java-API unter:

`https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html`

Eine `ArrayList` mit Objekten, die diese Anforderung erfüllen, kann wie folgt sortiert werden:

```
import java.util.Collections;
...
ArrayList<Interval> array = new ArrayList<Interval>();
...
Collections.sort(array);
```