Institut für Informatik
Prof. Dr.-Ing. Elke Pulvermüller
Dipl.-Systemwiss. Mathias Menninghaus

Universität Osnabrück, 20.06.2017
http://www-lehre.inf.uos.de/~binf
Testat bis 28.06.2017, 14:00 Uhr

Übungen zu Informatik B

Sommersemester 2017

Blatt 9

Aufgabe 9.1: Spezielle Threads (28 Punkte)

Implementieren Sie ein Programm, das darauf horcht, ob sich die Größe eines in den Kommandozeilenargumenten angegeben Verzeichnisses oder einer Datei verändert hat. Nutzen Sie dazu einen `java.util.TimerTask`, der einmal pro Sekunde überprüft, ob sich die Größe einer `File`-Instanz verändert hat und im Falle einer Änderung die Größe auf der Kommandozeile ausgibt. Die Größe eines Verzeichnisses sei hier die Größe aller Dateien und Unterverzeichnisse zusammengenommen. Das Programm soll nur über den Konsolenbefehl *ctrl + C* beendet werden können und im Falle des Beendens noch eine Meldung auf der Kommandozeile ausgeben.

Aufgabe 9.2: Nebenläufiges Ameisenrennen (37 Punkte)

Vervollständigen Sie das Programm `AntRace`, das die Länge aller kürzesten Wege von einem Startfeld zu allen anderen Feldern auf einem Spielfeld berechnet (*all-pairs-shortest-path*).

Die Klassen `antRace.Field`, `antRace.AntRace`, `antRace.Ant` und das Interface `antRace.Fields` sind vorgegeben und können von Ihnen nach Belieben verändert und erweitert werden. Sie können die Anfangsparameter ihres Programms in der `main`-Methode hart codieren, d.h. Sie brauchen keine Benutzereingabe zu implementieren.

Arbeiten Sie zur Berechnung der kürzesten Wege mit einer eigenen `Thread`-Implementation. Jeder Ihrer Threads sei eine Ameise. Die Klasse `antRace.Ant` gibt einen solchen `Thread` mit einem Konstruktor vor. Eine neue Ameise wird mit einem Startfeld und der bisher gelaufenen Anzahl an Schritten erzeugt. Die erste Ameise würde also mit dem Startfeld und `1` aufgerufen. Bei der Erzeugung setzt jede Ameise die Schrittzahl auf Ihrem Startfeld auf ihre eigene Schrittzahl. Danach schaut sich eine Ameise in jedem Durchlauf alle 8 Nachbarfelder (Moore-Nachbarschaft) an. Ist ein Feld als frei gekennzeichnet oder die dort eingetragene Schrittzahl größer als die Schrittzahl der Ameise plus eins, hat die Ameise einen neuen Weg gefunden. Den ersten neuen Weg beschreitet die Ameise selbst, d.h. sie erhöht Ihre eigene Schrittzahl um eins und verändert ihr Feld auf das Nachbarfeld. Für jeden weiteren Weg, den die Ameise im aktuellen Durchlauf findet, erzeugt sie eine neue Ameise mit dem Nachbarfeld und der um eins erhöhten Schrittzahl. Eine Ameise ist fertig, wenn sie in ihrer Nachbarschaft keine neuen Wege mehr findet.

Wenn alle Ameisen Ihre Arbeit beendet haben, steht in jedem Feld, das vorher auf `0` gesetzt war, eine natürliche Zahl ≥ 1, die die Länge des kürzesten Weges zum Startfeld darstellt. Den kürzesten Weg von einem Feld zum Startfeld erhielte man also, indem man solange zur nächst kleineren Zahl wandert, bis man beim Startfeld angekommen ist.

Achten Sie darauf, dass Ihr Programm erst dann beendet wird und die Länge der kürzesten Wege pro Feld auf der Standardkonsole ausgibt, wenn die letzte Ameise ihre Arbeit beendet hat.

Aufgabe 9.3: Nebenläufiges Suchen (35 Punkte)

Implementieren Sie ein Programm, das auf der Kommandozeile mit einem regulären Ausdruck und einer Pfadangabe zu einer Datei oder einem Verzeichnis aufgerufen wird und die Datei bzw. alle Dateien in dem Verzeichnis zeilenweise nach Zeichenketten absucht, die dem regulären Ausdruck genügen.

Von jeder Datei, in der mindestens eine Zeile gefunden wurde, die eine Zeichenkette enthält, die dem regulären Ausdruck genügt, soll der Dateiname auf der Kommandozeile ausgegeben werden. Unter dem Dateinamen soll jede Zeile der Datei mit mindestens einem Treffer ausgegeben werden und zwar zusammen mit der Zeilennummer und der Anzahl der Treffer. Wenn das Programm mit der Option `-r` aufgerufen wird, sollen auch alle Unterverzeichnisse rekursiv durchlaufen werden.

Nutzen Sie zur Implementation die Klassen von Aufgabenblatt 7 und Aufgabenblatt 8.

Jede Suche nach Treffern in einer Datei durch Benutzung des `SearchLineReader` soll mit einem eigenen `Thread` erfolgen. Da die Ausgabe gruppiert nach den durchsuchten Dateien erfolgen soll, sollte kein `Thread` die Ergebnisse direkt auf die Kommandozeile schreiben. Stattdessen sollten die Threads ihre Ergebnisse in eine gemeinsame Ergebnis-`Collection` schreiben, die, nachdem alle `Thread`-Instanzen fertig sind, ausgeben wird. Achten Sie dafür darauf, dass die Ergebnis-`Collection` *thread-sicher* ist, also dass unterschiedliche `Thread`-Instanzen Elemente einfügen und abfragen können. Das Paket `java.util.concurrent` enthält solche Collections.

Erst nach Beenden aller nebenläufigen Such-Threads kann die Ausgabe erfolgen. Deswegen sollte Ihr Hauptprogramm auf jede gestartete `Thread`-Instanz warten.