PFP-Team
pfp@i2.cs.fau.de

Übungsblatt 9

Abgabe bis 03.07.2019, 18:00 Uhr

Aufgabe 9.1: Zeigerverdopplung

Ermitteln Sie an folgendem Beispiel den Rang für jedes Listenelement sequentiell und mit dem in der Vorlesung vorgestellten parallelen Verfahren (Folie 09-6ff). Die Zahlen in der Verkettungsliste stellen die Positionen des Nachfolgers in der Liste dar. Der Anfang der Liste ist an Index 0 und das Ende durch den Eintrag -1 dargestellt.

5	7	1	4	-1	2	3	6

Aufgabe 9.2: MapReduce

a) Was ist der MapReduce-Ansatz? Wie funktioniert er?

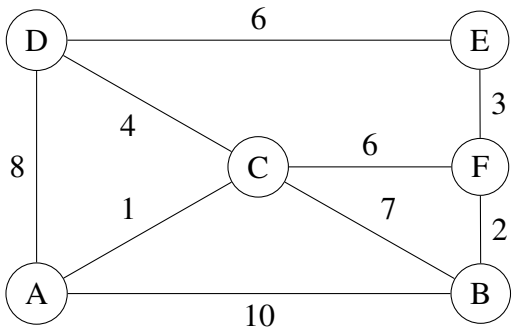
b) Worin liegt dessen Vorteil im Vergleich zu anderen Parallelisierungsverfahren?

c) Was kann bei diesem Ansatz parallel ablaufen? Was muss sequentiell sein?

Aufgabe 9.3: Graphalgorithmen in MapReduce

a) Wie funktionieren Map und Reduce bei der in der Vorlesung vorgestellten Dijkstra-Variante (Folie 09-28ff)?

b) Wenden Sie den MapReduce-Ansatz auf folgenden Graphen an.



c) Überlegen Sie sich eine MapReduce-Variante zur Bestimmung der transitiven Hülle eines gerichteten Graphen.

Hinweis: Die transitive Hülle G* eines gerichteten Graphen G ist definiert als: $G^* := (V, E^*)$ mit $E^* := \{(u, v) | es\ existiert\ ein\ Weg\ von\ u\ nach\ v\}$

Bonusaufgabe 9.4: Sokoban **6 Punkte**

Implementieren Sie in der Klasse `PFPSokoSolverImpl` das gegebene Interface `PFPSokoSolver`. Die geforderte Methode `solve(...)` soll eine Lösung für ein Level des Spiels Sokoban (http://de.wikipedia.org/wiki/Sokoban) berechnen. Bei Sokoban versucht ein Spieler, Kisten auf die vorgegebenen Ziele eines Spielfelds zu schieben.

a) Ihre Aufgabe ist es, eine parallele Lösung zu implementieren. Sie können den Algorithmus und die Art der Parallelisierung frei wählen. Achten Sie darauf, dass Ihre Version mit einer beliebigen Anzahl Threads funktioniert und sich nach Ablaufen des Timeouts mit der Rückgabe `null` beendet.

Sie dürfen diese Aufgabe in 2er-Teams bearbeiten. Schreiben Sie dazu den Namen und die Benutzerkennung (wie im EST hinterlegt) der Team-Mitglieder als Kommentar in den Sourcecode von `PFPSokoSolverImpl`.
z. B.
// Max Mustermann ma01must
// John Doe xy10wxyz

Sie können ein interaktives Sokoban-Spiel mit folgendem Aufruf starten:
java PFPSokoTools PFAD_ZU_SOKOTESTS.txt LEVEL_NUMMER
Mit dem folgenden Aufruf können Sie sich den Ablauf einer gefunden Lösung ansehen:
java PFPSokoTools PFAD_ZU_SOKOTESTS.txt LEVEL_NUMMER LOESUNG
z.B.:
java PFPSokoTools SokoTests.txt 0 dlUrrrdLullddrUluRuulDrddrruLdlUU
java PFPSokoTools SokoTests.txt 43 R

Implementierungs-Idee:
Alle Threads arbeiten auf einer gemeinsamen Warteschlange, in der nach und nach die zu untersuchenden Level-Konfigurationen abgelegt werden. Eine Alternative dazu wäre eine Implementierung mit mehreren Warteschlangen und Work-Stealing. Ein Level ist gelöst, wenn sich alle Kisten auf Zielfeldern befinden. Um Speicher zu sparen, sollten Sie die Level-Konfigurationen in einer platzsparenden Repräsentation (z. B. http://docs.oracle.com/javase/8/docs/api/java/util/BitSet.html) ablegen. Wurde eine Level-Konfiguration bereits vorher untersucht, dann kann die weitere Untersuchung dieses Lösungspfades abgebrochen werden.

b) Zusatzaufgabe (ohne Punkte): Schwierigere Level lassen sich in akzeptabler Zeit nur durch die Erkennung von sog. „Deadlocks“ (hier: unlösbare Konfigurationen, z. B. Box in einer Ecke) lösen. In einem einfachen Ansatz werden diese Deadlocks vor dem eigentlichen Lösen statisch bestimmt und dann zur weiteren Begrenzung des Suchraums eingesetzt.

c) Zusatzaufgabe (ohne Punkte): Implementieren Sie weitere Heuristiken, um möglichst viele Konfigurationen, die nicht zum Ziel führen, frühzeitig ausschließen zu können.

Hinweis: Die Art und Weise, wie Sie in dieser Aufgabe vorgehen, ist vollständig Ihnen überlassen. Sie können den Solver auch auf andere Weise implementieren (z. B. genetische Algorithmen), solange das Verfahren parallel läuft.
Hinweis: Konzentrieren Sie sich darauf, einen allgemeinen parallelen Algorithmus zu entwickeln, der die *nicht* auskommentierten Testfälle löst.

6 Punkte