Übung 04: Multithreading

Abgabetermin: 25. 4. 2022, 8:30

Übung 4: Parallele Suche nach benachbarten Punkten (24 Points)

Eine wichtige geometrische Aufgabenstellung ist, in einem Gebiet jene Paare von Objekten zu finden, die weniger als einen bestimmten Abstand voneinander entfernt sind. Wir sagen, dass Paare von Objekten miteinander *in Kontakt* stehen.

Eine triviale Lösung ist, die Objekte paarweise zu prüfen und Ihre Distanz zu ermitteln. Dieses triviale Verfahren hat aber quadratische Laufzeitkomplexität und versagt schnell bei vielen Objekten. In der Computergeometrie gibt es daher unterschiedliche Verfahren, um die Komplexität zu reduzieren. Ein solches Verfahren basiert auf den sogenannten *Quadtrees*. Dabei wird ein Gebiet mit vielen Objekten wiederholt in vier Teilbereiche unterteilt, bis sich in jedem Teilbereich maximal eine bestimmte Anzahl von Punkten befinden. Ergebnis ist ein Baum, bei dem die inneren Knoten eine Teilung in genau 4 Nachfolger und die Blätter maximal *n* Objekte enthalten. Abbildung 1 zeigt das Verfahren mit einer maximalen Anzahl von 2 Objekten pro Gebiet. Sind mehr als 2 Objekte in einem Gebiet, wird es in Quadranten *NW*, *NE*, *SE* und *SW* geteilt und die Objekte den neuen Teilgebieten zugeordnet. Befinden sich wiederum im jeweiligen Quadranten mehr als 2 Objekte, wird weiter geteilt (Abbildung 1(a)). Ergebnis ist der Baum in Abbildung 1(b) mit inneren Knoten mit jeweils vier Unterknoten und den Blättern mit den Objekten.

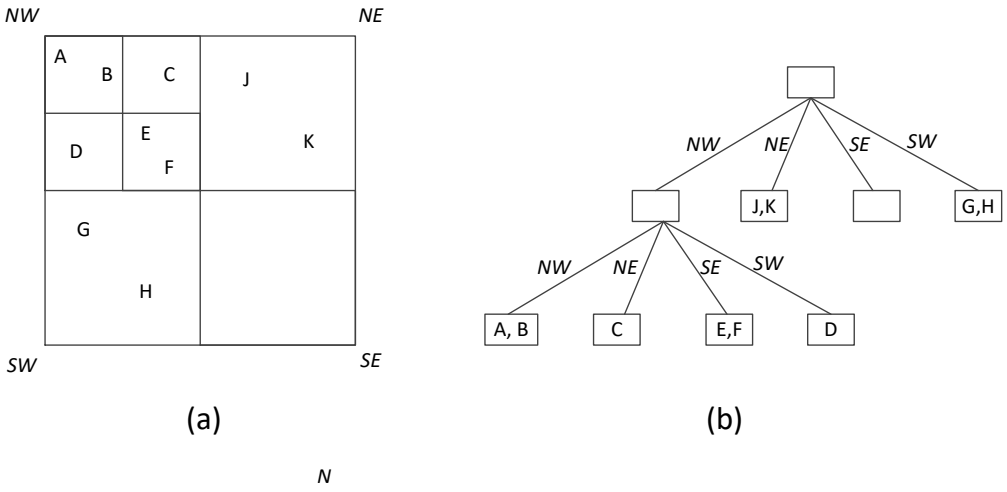

Abbildung 1: Quadtrees

Quadtrees entsprechen einem Divide-and-Conquer Ansatz und eignen sich daher gut für eine Parallelisierung mit `RecursiveTask`s. In dieser Übung sollen Sie daher nach diesem Verfahren eine parallele Suche nach Kontakten von Objekten realisieren. Dazu werden Ihnen im Download eine Reihe von Klassen vorgegeben und eine sequentielle Lösung gezeigt.

Folgende Klassen werden zur Verfügung gestellt:

- Klasse `Pos`: stellt eine Position mit x- und y-Koordinate dar.
- Klasse `Entity`: Objekte im Gebiet; haben eine eindeutige Id und eine Position und zusätzlich eine Methode:
  - `double distance(Entity other)`: zur Berechnung der Distanz zu einem anderen Entity
- Klasse `Area`: repräsentiert ein Gebiet mit entsprechenden Koordinaten und Größe und enthält Entities. `Area` bietet folgende Methoden:
  - `getLeft()`, `getTop()`, `getRight()` und `getBottom()`: Zugriff auf Koordinaten
  - `getWidth()` und `getHeight()`: Zugriff auf Größe

- `add(Entity)`: Anfügen eines Entities
- `Set<Entity> getEntities()`: Zugriff auf die Menge der Entities
- `boolean isWithin(Entity)`: Testet, ob sich das Entity innerhalb dieses Gebiets befindet.

- Klasse `Contact`: Ist ein Paar von Entities, die in Kontakt stehen.
- Die Klasse `Result` stellt das Resultat einer Berechnung dar und erlaubt eine Menge von `Contact`s zu verwalten.
- Klasse `Constants` enthält eine Reihe von Konstanten mit wichtigen Einstellungen.

Des Weiteren sind zwei Enumerationsklassen gegeben, die Sie bei Ihrer Lösung gut einsetzen können:

- Enumerationsklasse `Dir` mit Werten `W`, `N`, `E` und `S` für die vier Himmelsrichtungen.
- Enumerationsklasse `Quarter` mit Werten `NW`, `NE`, `SE` und `SW` zur Bezeichnung der vier Quadranten einer Teilung eines Gebiets (siehe Methode `split` von `Area`).

Beachten Sie auch die JavaDoc-Kommentare in den Klassen.

Sequentielle Lösung

Die Suche nach Kontakten zwischen den Entities kann einfach gelöst werden, indem man die Entities in einem Gebiet paarweise vergleicht und ihre Distanz prüft:

```
Result result = new Result();
for (Entity e1: area.getEntities()) {
   for (Entity e2: area.getEntities()) {
      if (!e1.equals(e2)) {
         if (e1.distance(e2) < CONTACT_DISTANCE) {
             Contact c = result.addContact(e1, e2);
         }
      }
   }
}
```

Diese Lösung ist Ihnen im Download in Klasse `contacts.MainSeq` zur Verfügung gestellt. Diese sequentielle Lösung versagt schnell bei einer größeren Anzahl von Entities.

RecursiveTask und Fork-Join-Pool

Lösen Sie die Aufgabe mit einem `RecursiveTask` und der Ausführung mit einem Fork-Join-Pool. Das Verfahren soll auf einer Aufteilung mit Quadtrees aufbauen. Die Anzahl der maximalen Elemente pro Teilgebiet soll man einstellen können (siehe Konstante `Constants.THRESHOLD`).

Der RecursiveTask

```
public class ContactRecursiveTask extends RecursiveTask<Result>
```

wird mit dem zu behandelnden Gebiet erzeugt und hat als Ergebnis ein `Result` mit den gefundenen Kontakten. Es soll nach dem Divide& Conquer-Prinzip wie folgt arbeiten:

- Enthält das Gebiet bereits wenige Entities, werden die Kontakte durch paarweisen Vergleich ermittelt und in einem Result-Objekt zurückgegeben.
- Sonst wird das Gebiet geteilt und für die Teilgebiete neue Tasks abgespalten. Dann wird auf die Lösung dieser Tasks gewartet und die retournierten Mengen zusammengeführt. Diese Gesamtmenge wird dann retourniert.

Das obige Verfahren hat noch ein wesentliches Problem, das sich durch die Aufteilung der Gebiete ergibt. Es berücksichtigt nämlich nicht Paare von Objekten, die sich nahe den Grenzen in zwei benachbarten Gebieten befinden. Abbildung 2 zeigt das Problem. Die benachbarten Punkte innerhalb des Teilgebiets werden erkannt, die Objekte aus zwei benachbarten Teilgebieten, die sehr nahe den Grenzen liegen, stehen in Kontakt, werden aber

nicht geprüft. Überlegen Sie sich ein Verfahren, sodass auch diese Objekte aus den Grenzgebieten der benachbarten Teilgebiete geprüft werden.

Hinweis: Die Lösung sollte im rekursiven Aufstieg erfolgen. Zusätzlich zu den erkannten Kontakten können die Objekte nahe den 4 Grenzen zurückzugeben und dann im rekursiven Aufstieg geprüft werden (dazu sollte die Klasse `Result` erweitert werden; die Werte `W`, `N`, `E`, und `S` der Enumeration `Dir` kann für die Bezeichnung der vier Grenzgebiete verwendet werden).

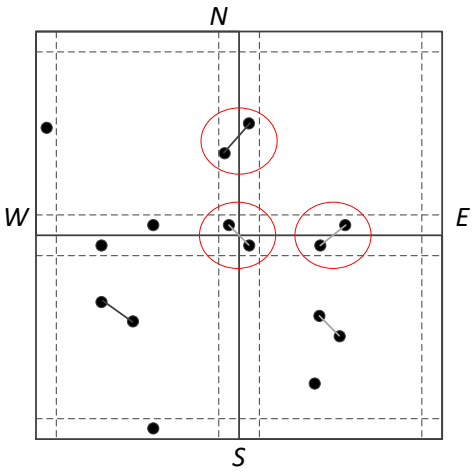

Abbildung 2: Objekte in Grenzgebieten

Experimente:

- Experimentieren Sie mit unterschiedlichen Werten für die Größe des Gebiets, die Anzahl der Entities, maximale Anzahl der Objekte im Gebiet (`Constants.THRESHOLD`) etc. Die Klasse `constacts.Constants` enthält bereits Einstellungen dazu.
- Führen Sie Laufzeitmessungen durch und ermitteln Sie eine optimale Schranke (`Constants.THRESHOLD`) zur Minimierung der Laufzeit. Vergleichen Sie das parallele und das sequentielle Verfahren.