Hochschule RheinMain
Prof. Dr. Adrian Ulges

30.05.2019

# Algorithmen und Datenstrukturen (SS 2019)

## Aufgabenblatt 6

**zu bearbeiten bis: 09.06.**

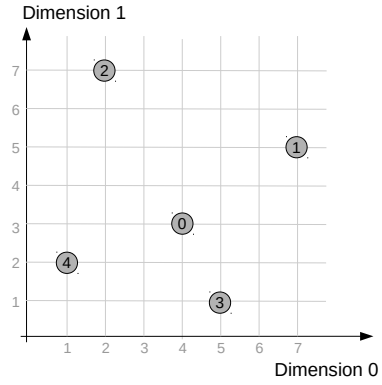
---

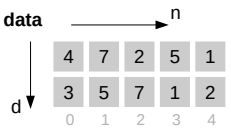
Dieses Übungsblatt ist Ihre erste **bewertete Abgabe** (bis zu 12 Punkte sind zu erreichen). Beachten Sie die Hinweise auf den Slides "Organisatorisches" *(siehe Stud.IP)*. **Weitere Hinweise**:

- Spätester **Abgabetermin** ist **Montag, der 10.06.**, um 04:17 Uhr in der Frühe. Sie können natürlich früher abgeben.
- In den Praktika nächste Woche (03.06.-07.06.) werden die Aufgaben besprochen, Sie können **Verständnisfragen** stellen und erhalten Hilfestellung beim Programmieren.
- Theorie-Aufgaben geben Sie als PDF ab, Programmieraufgaben über Subato (Ihre letzte getestete Version vor der Deadline zählt automatisch als Abgabe).
- Form der Abgabe: Sie schicken eine Mail *(die Adresse wird noch bekanntgegeben)*. Inhalt:
  - Namen und Matrikelnummern aller Team-Mitglieder.
  - Ihre Praktikumsgruppe mit Zeit und Gruppenleiter (z.B. WI / Gruppe A / Ulges, MO, 11:45 Uhr).
  - Ein PDF mit Ihrer Lösung der Theorie-Aufgaben.
  - Der Nutzer-Account, der in Subato verwendet wurde (z.B. mmuster001).

  Sollten Angaben fehlen behalten wir uns einen **Punktabzug** vor.
- Ihr Team sollte die Aufgaben **selbstständig** lösen. Plagiate werden ggfs. bestraft. Jeder Gruppenpartner muss sich mit allen Aufgaben und deren Lösung auskennen.

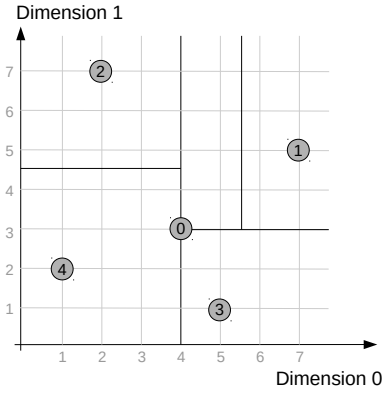
**Aufgabe 6.1 (K-d-Bäume bauen)**
Wir werden auf diesem Blatt k-d-Bäume implementieren. Diese sind in der Computer-Grafik, bei der Textsuche und in der KI eine beliebte Datenstruktur um schnell Punkte in der Nähe eines Anfragepunkts (engl. *query*) zu finden. Gegeben sei im folgenden eine Menge von $n$ $d$-dimensionalen Punkten. Diese speichern wir in einem $d \times n$ Array ab, wie in diesem Beispiel:

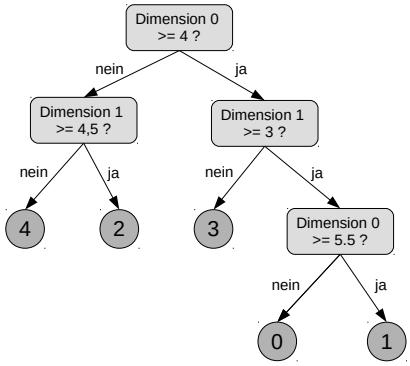


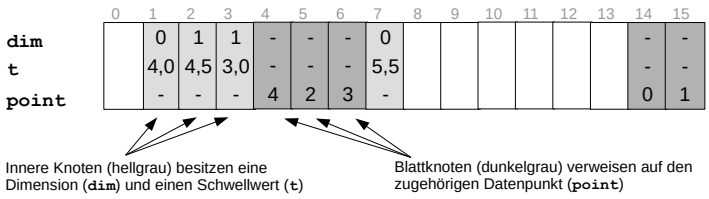

```
double data[][] =
   new double[][] {
      {4,7,2,5,1},
      {3,5,7,1,2}
   };
```

Ein k-d-Baum splittet den Raum der Punkte rekursiv anhand einer Dimension und eines Schwellwerts $t$ *(engl. 'threshold')* in Werte $< t$ und Werte $\geq t$. Zum Beispiel könnten wir die Daten erst mit einem senkrechten Schnitt bei $t = 4$ schneiden, dann die linke Hälfte mit einem waagerechten Schnitt bei $t = 4,5$ usw. Wir erhalten einen Baum (rechts). Die inneren Knoten des Baums entsprechen den Schnitten, jedes Blatt enthält einen Datenpunkt:



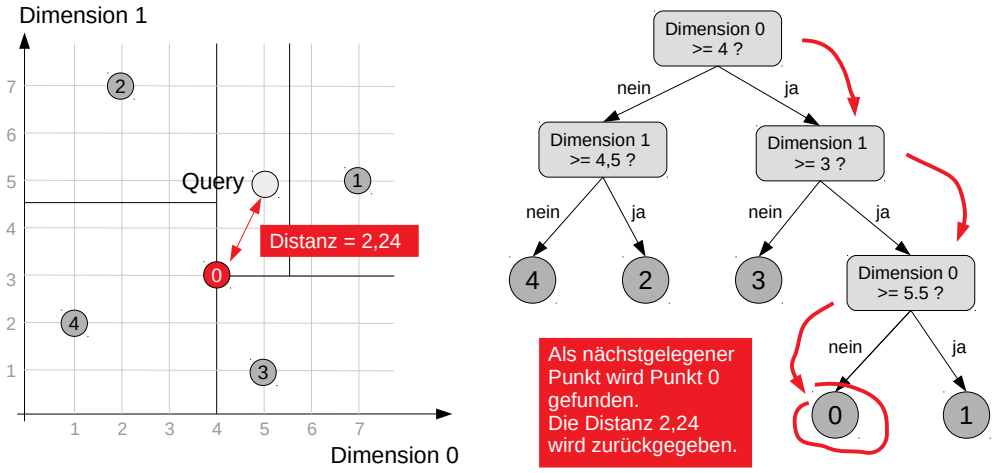

In unserem k-d-Baum werden die einzelnen Knoten in einem Array wie in einem Heap abgespeichert, d.h. die Wurzel ist Element Nr. 1 (Element 0 bleibt leer), und das linke und rechte Kind lassen sich mit den einfachen Heap-Formeln bestimmen. Für jeden inneren Knoten speichern wir die Dimension und den Schwellwert nach dem gesplittet wird. Für jedes Blatt speichern wir die Nummer des zugehörigen Datenpunkts:



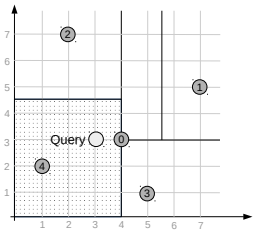
Implementieren Sie eine rekursive Routine `build(double[][] data)`, die aus den Punkten in `data` einen solchen k-d-Baum erstellt. Hinweise:

- Runden Sie $n$ auf die nächste Zweierpotenz $n^*$ (siehe `nextPowerOfTwo(n)`). Legen Sie das Array von Knoten (`nodes`) der Länge $2n^*$ an, dies bietet genug Platz für Ihre Knoten.
- Splitten Sie zuerst nach Dimension 0, auf der nächsten Ebene nach Dimension 1, usw. Erreichen Sie das Ende der Dimensionen, fangen Sie wieder bei 0 an.
- Wählen Sie als Schwellwert $t$ den Median: Sortieren Sie in `median()` zunächst die Werte (Sie können Javas fertiges `Arrays.sort()` verwenden). Ist die Anzahl der Werte ungerade, nehmen Sie den Wert in der Mitte. Ist die Anzahl gerade, bilden Sie das Mittel zwischen den beiden mittleren Werten. Auf diese Weise erhalten Sie für das obige Beispiel exakt den abgebildeten Baum.
- Bei jedem rekursiven Aufruf sollte ein Array `points` mit den Indizes der im aktuellen Knoten befindlichen Datenpunkte übergeben werden. Teilen sie beim Split dieses Array in zwei Arrays, eins für die linke (`points0`) und eins für die rechte Hälfte (`points1`). Der erste Split zerlegt z.B. `points = [0,1,2,3,4]` in `points0 = [2,4]` und `points1 = [0,1,3]`.
- Reduziert sich die Anzahl der Punkte auf 1, erstellen Sie einen Blattknoten. Sie können annehmen dass alle Werte paarweise unterschiedlich sind (d.h., nach dem Split befindet sich links und rechts mindestens noch ein Datenpunkt).

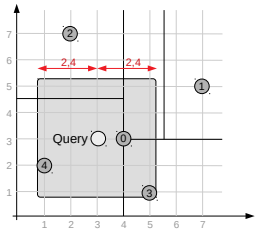
**Aufgabe 6.2 (K-d-Bäume: Suche (Greedy))**
Gegeben einen neuen Punkt `query` soll nun der nächstgelegene Punkt im Baum gefunden werden. Beginnen Sie hierzu bei der Wurzel und bewegen Sie sich jeweils in die Hälfte (gemäß dem jeweiligen Split), in dem der Query liegt. Sie erreichen einen Blattknoten mit einem Datenpunkt $p$. Geben Sie nun die Euklidische Distanz zwischen `query` und `p` zurück.



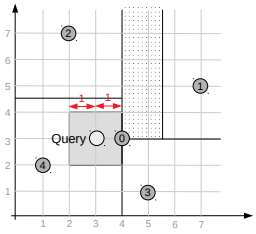
**Aufgabe 6.3 (K-d-Bäume: Suche (Backtracking))**
Wieder soll der nächstgelegene Punkt zu `query` gefunden werden. Nun wenden wir Backtracking an und durchlaufen den *kompletten* Baum: Erreichen wir ein Blatt mit einem Punkt in Distanz `d`, definieren wir eine sogenannte *region of interest (ROI)*, ein Quadrat der Seitenlänge `2d` um den Query. Jeder bessere Punkt als der gefundene müsste nun innerhalb der ROI liegen. Regionen, die sich mit der ROI nicht schneiden, können beim Backtracking übersprungen werden. Finden wir einen besseren Punkt, verkleinern wir die ROI entsprechend.



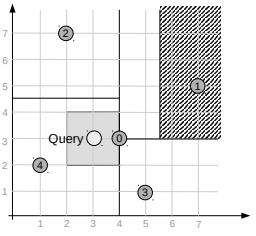
Der Query landet im Bereich von Punkt 4.



Die Distanz zwischen Query und Punkt 4 beträgt 2,4. Die ROI *(graues Quadrat)* erstreckt sich also 2,4 um den Query.



Backtracking findet mit Punkt 0 eine bessere Lösung. Die ROI verkleinert sich auf 1.



Bereich die sich nicht mit der ROI schneiden (hier Blatt 1), werden übersprungen.

**Aufgabe 6.4 (K-d-Bäume: Theorie)**
Es sei ein k-d-Baum mit $n$ Datenpunkten gegeben.

- Was ist die Komplexität einer Greedy-Suche?
- Was ist die Komplexität einer Backtracking-Suche?
- Was ist die Komplexität der Erstellung des Baums? *(diese Aufgabe ist schwierig).*

Verwenden Sie zur Bestimmung der Komplexität jeweils die Anzahl der Zugriffe auf die Arrays `data` und `nodes`. Geben Sie die Komplexität ausschließlich im *Worst Case* an. Begründen Sie jeweils knapp.