Projektarbeit Algorithmische Geometrie

kd-Tree Applet

Alexander Lange
Matthias Rombach

30. Juni 2014

Einige Datenbankabfragen können geometrisch als höherdimensionale Bereichsanfragen interpretiert werden. Um diese effizient durchzuführen, lassen sich kd-Trees als Datenstruktur einsetzen.
Das in dieser Projektarbeit erstellte Applet visualisiert 2-dimensionale Rohdaten und stellt die entsprechende Baumstruktur graphisch dar.
Weiter lassen sich Bereichsanfragen als schrittweise Simulation durchführen, um dem Benutzer zu erlauben, den Suchvorgang nachzuvollziehen.

1 Problemstellung

Eine Menge von 2-dimensionalen Datenpunkten soll in einer Datenstruktur gespeichert werden, die sich durch geringen Speicherbedarf auszeichnet und achsenparallele rechteckige Bereichsabfragen effizient verarbeitet.

2 Algorithmus

Der verwendete Algorithmus wurde auf Grundlage der Beschreibung in [1] entwickelt. In der Literatur wird gezeigt, dass ein kd-Tree für eine Menge von n Punkten in der Ebene einen Speicherbedarf von $O(n)$ hat und in $O(n \log n)$ Zeit aufgebaut werden kann. Eine rechteckige Suchanfrage benötigt $O(\sqrt{n} + k)$ Laufzeit, wobei k für die Anzahl der zurückgegebenen Punkte steht.
Die beiden Algorithmen zum Aufbau des Baumes 1 und zur Suchanfrage 2 sind nachfolgend als Pseudocode dargestellt.

Literatur

[1] Mark de Berg, Hrsg. *Computational geometry : algorithms and applications.* 3. ed. Berlin: Springer, 2008. ISBN: 978-3-540-77973-5.

BuildKdTree(P,*depth*)
Eingabe: Menge an Datenpunkten P
Ausgabe: Wurzel des Baumes
if $|P| = 1$ **then**
  return Blatt mit dem Punkt in P
else
  if *depth* gerade **then**
    teile P vertikal an $l : x = x_{\text{median}(P)}$ in
    P_1 (Punkte links von oder auf l)
    und $P_2 = P \setminus P_1$
  else
    teile P horizontal an $l : y = y_{\text{median}(P)}$ in
    P_1 (Punkte unter oder auf l)
    und $P_2 = P \setminus P_1$
  end
  $v_{\text{left}} \leftarrow$ BuildKdTree($P_1, depth + 1$)
  $v_{\text{right}} \leftarrow$ BuildKdTree($P_2, depth + 1$)
  erzeuge Knoten v, der l speichert
  setze v_{left} und v_{right} als Kinder von v
  return v
end

Algorithmus 1: BuildKdTree(P,*depth*) [1]

SearchKdTree(v,R)
Eingabe: Wurzel eines Baums v und Suchbereich R
Ausgabe: Alle Blätter im Suchbereich
if v Blatt $\wedge$ Punkt p in $v \in R$ **then**
  return p
else
  if region(lc(v)) $\subseteq R$ **then**
    ReportSubtree(lc(v))
  end
  else if region(lc(v)) $\cap R \neq \emptyset$ **then**
    SearchKdTree(lc(v),R)
  end
  if region(rc(v)) $\subseteq R$ **then**
    ReportSubtree(rc(v))
  end
  else if region(rc(v)) $\cap R \neq \emptyset$ **then**
    SearchKdTree(rc(v),R)
  end
end

Algorithmus 2: SearchKdTree(v,R) [1]