Karlsruher Institut für Technologie
Fakultät für Informatik
ITI Wagner

Algorithmische Geometrie
Sommersemester 2014
Martin Nöllenburg/Benjamin Niedermann

Übungsblatt 5 - Bereichsabfragen (Lösung)

Ausgabe: 13. Mai 2014
Abgabe: 20. Mai 2014

1 Worst Case Laufzeit

Im Beweis zur Laufzeit von **Bereichsabfragen** in *kd*-Trees wurde erwähnt, dass die w-c Laufzeit in $\mathcal{O}(\sqrt{n}+k)$ ist. Entscheidend für die Laufzeit ist dabei wie viele Regionen man im schlimmsten Fall überprüfen muss. Wir bezeichnen diese Anzahl mit $Q(n)$.

a) Zeigen Sie, dass $Q(n)$ durch folgende Rekurrenz angegeben werden kann.

$$Q(n) = \begin{cases} \mathcal{O}(1) & \text{, für } n = 1 \\ \mathcal{O}(1) + 2Q(n/4) & \text{, für } n > 1 \end{cases}$$

b) Zeigen Sie, dass die Rekurrenz zu $Q(n) = \mathcal{O}(\sqrt{n})$ aufgelöst werden kann.

c) Zeigen Sie, dass $\Omega(\sqrt{n})$ tatsächlich untere Schranke für Anfragen in einem *kd*-Trees sind, indem Sie die Position von n Punkten in der Ebene festlegen und ein passendes Anfrage-Rechteck bestimmen.

Hinweis: Für Details über Bereichsabfragen in *kd*-Trees empfiehlt sich ein Blick in das Buch *Computational Geometry: Algorithms and Applications**

Lösung: Zu a): Hinreichend gut erläutert in $\star$

Zu b): Anwenden des Master-Theorems.

$$T(n) = aT(n/b) + f(n)$$

Hier ist $a = 2, b = 4$ und $f(n) = 2$. Da $f(n) = \mathcal{O}(n^{\log_b a - \varepsilon})$, und für passendes ε folgt:

$$Q(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 4}) = \Theta(\sqrt{n})$$

Zu c): Um die Situation der Rekursionsgleichung nachzubilden (um $\Theta(\sqrt{n})$ Aufwand zu erlangen) sollten wir vermeiden, dass ReportSubtree aufgerufen wird. Stattdessen sollte immer wieder ein rekursiver Aufruf erfolgen. In Abbildung 1 ist ein solches Beispiel dargestellt.

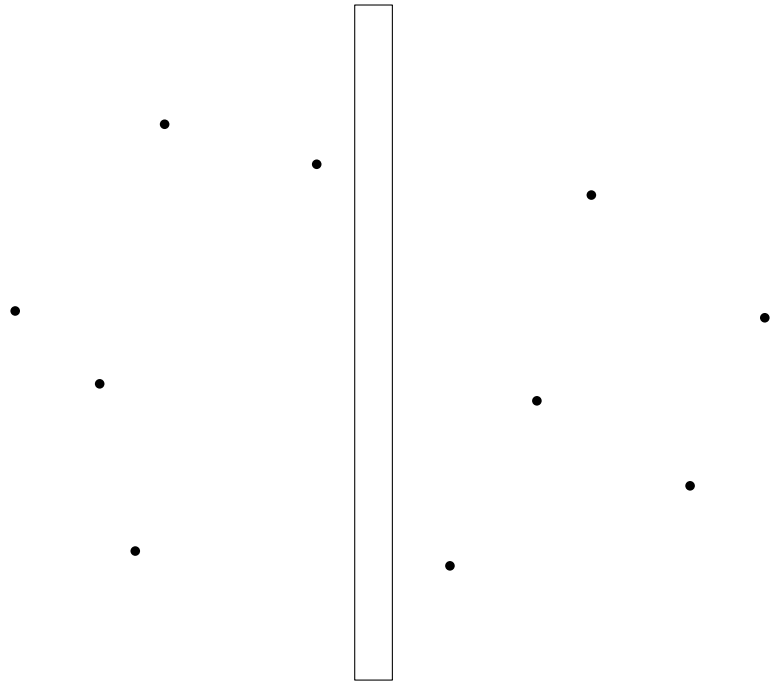

Abbildung 1: Lösung zu 1 c)

2 *kd*-Trees

kd-Trees können für *partial match queries* verwendet werden. Bei einer 2-dimensional partial match query wird der Wert für eine der Koordinaten festgelegt und dann werden alle Punkte angefordert, die für die festgelegte Koordinate den festgelegten Wert haben (z.B. wird nach allen Punkten mit der x-Koordinate 7 gefragt).

a) Zeigen Sie, dass man partial match queries mit Hilfe von *kd*-Trees in $\mathcal{O}(\sqrt{n}+k)$ beantworten kann. Dabei ist k die Anzahl der gefundenen Punkte.

b) Erklären Sie, wie man einen 2-dimensionalen Range-Tree verwenden kann, um partial match queries zu beantworten. Ermitteln Sie auch die worst case Laufzeit.

c) Beschreiben Sie eine Datenstruktur, die nur linearen Speicher benötigt und partial match queries in $\mathcal{O}(\log n + k)$ Zeit beantworten kann.

Lösung: Zu a) + b): Anfrageregion ist ein Rechteck ohne Höhe mit 'unendlicher' Breite.

Zu c): Balancierter binärer Suchbaum. Pro x-(bzw. y-) Koordinate ein Suchbaum. Jeder Knoten kommt nur in einem (bzw. 2) Suchbaum (Suchbäumen) vor. Aufwand für Anfragen $\mathcal{O}(\log n + k)$ und Speicher in $\mathcal{O}(n)$.

3 Bereichsabfragen

In einigen Anwendungen ist man nicht direkt an den Punkten in einem Bereich interessiert, sondern zum Beispiel nur an der Anzahl der Punkte in einem Bereich. Diese Anfragen werden auch *range counting queries* genannt. Wir wollen Range-Trees für diese Anfragen verwenden, aber suchen eine Möglichkeit um den additiven Term $\mathcal{O}(k)$ in der Query-Zeit zu vermeiden.

a) Beschreiben Sie wie ein 1-dimensionaler Range-Tree adaptiert werden kann damit man range counting queries in $\mathcal{O}(\log n)$ Zeit durchführen kann.

b) Benutzen Sie die Lösung aus a), um zu beschreiben wie man d-dimensionale range counting queries durchführen kann.

Lösung: Zu a): Speichere in Knoten des Baums wie viele Kinder der Knoten hat. Die Methode ReportSubtree muss dann nur noch diese Zahl ausgeben. Das benötigt nur noch $\mathcal{O}(1)$ statt $\mathcal{O}(k)$.

Zu b): Nutze die Idee aus a), aber modifiziere nur den Baum der zuständig ist für die kleinste Dimension.

4 Kompliziertere Abfragen

In vielen Anwendungen möchte man Bereichsabfragen nicht nur für Punkte, sondern auch für kompliziertere Objekte stellen.

a) Sei S eine Menge von n achsenparallelen Rechtecken in der Ebene. Wir wollen alle Rechtecke aus S ermitteln, die vollständig im Anfragerechteck $[x, x'] \times [y, y']$ enthalten sind. Beschreiben Sie eine Datenstruktur für dieses Problem, die $O(n \log^3 n)$ Speicher und $O(\log^4 n + k)$ Zeit für die Bearbeitung einer Anfrage benötigt, wobei k die Anzahl der dabei ermittelten Rechtecke ist.

b) Sei P eine Menge von n Polygonen in der Ebene. Wir wollen alle Polygone aus P ermitteln, die vollständig im Anfragerechteck $[x, x'] \times [y, y']$ enthalten sind. Beschreiben Sie eine Datenstruktur für dieses Problem, die $O(n \log^3 n)$ Speicher und $O(\log^4 n + k)$ Zeit für die *Bearbeitung einer Anfrage* benötigt, wobei k die Anzahl der dabei ermittelten Polygone ist.

Lösung: Zu a): Jedes Rechteck ist definiert durch seine linke untere und obere rechte Ecke. Nutze diese vier Werte als Korrdinaten für den 4-dimensionalen Raum. Damit entspricht jedes Rechteck einem 4-dimensionalen Punkt. Nutze nun 4-dimensionale Range Trees für die Anfragen.

Zu b): Gleiche Idee wie in a) nur muss zunächst die Bounding Box der Polygone bestimmt werden.