

## Übungsblatt 5 - Bereichsanfragen

### 1 Worst Case Laufzeit

Im Beweis zur Laufzeit von **Bereichsanfragen** 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) Zeige, 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) Zeige, dass die Rekurrenz zu  $Q(n) = \mathcal{O}(\sqrt{n})$  aufgelöst werden kann.
- c) Zeige, dass  $\Omega(\sqrt{n})$  tatsächlich untere Schranke für Anfragen in einem  $kd$ -Trees sind, indem du die Position von  $n$  Punkten in der Ebene festlegst und ein passendes Anfrage-Rechteck bestimmst.

*Hinweis:* Für Details über Bereichsanfragen in  $kd$ -Trees empfehle ich einen Blick ins Buch *Computational Geometry: Algorithms and Applications* [1]. Im Uni-Netz könnt ihr das kostenlos lesen. Die Beschreibung über die Bereichsanfragen findet sich auf S. 104.

[1] [www.springer.com/computer/theoretical+computer+science/book/978-3-540-77973-5](http://www.springer.com/computer/theoretical+computer+science/book/978-3-540-77973-5)

*Lösung:*

Zu a): Hinreichend gut erläutert in [1]

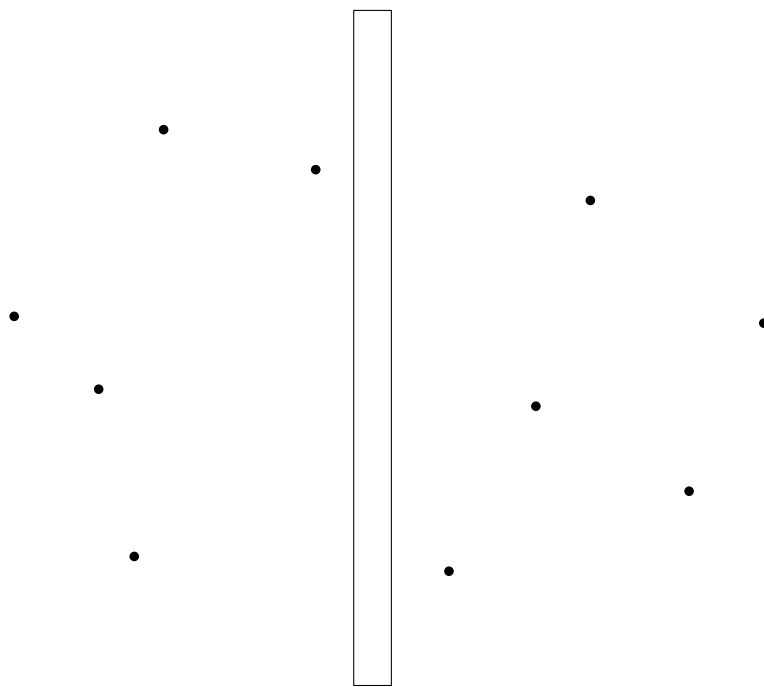
Zu b): Anwenden des Master-Theorems.

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

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



Anfrage-Rechteck

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) Zeige, 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) Erkläre wie man einen 2-dimensionalen Range-Tree verwenden kann um *partial match queries* zu beantworten. Ermittle auch die worst case Laufzeit.
- c) Beschreibe 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 Bereichsanfragen

In einigen Anwendungen ist man nicht direkt an den Punkten in einem Bereich interessiert sondern z.B. 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) Beschreibe wie ein 1-dimensionaler Range-Tree adaptiert werden kann damit man *range counting queries* in  $\mathcal{O}(\log n)$  Zeit durchführen kann.
- b) Benutze 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 REPORTS-UBTREE 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 Anfrageobjekte

In vielen Anwendungen möchte man Bereichsanfragen 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. Beschreibe 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. Beschreibe 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 Koordinaten 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.