Quadtrees

Bernhard Mallinger

e0707663

13. Februar 2011

Betreut durch Univ.-Ass. Dipl.-Ing., BSc Christian Schauer

Inhaltsverzeichnis

Teil I.

Theorie

1. Einführung

Quadtrees stellen eine Datenstruktur dar, welche von Binärbäumen abgeleitet sind. Sie erweitern dessen Prinzip der rekursiven, hierarchischen Aufteilung eines Raumes auf mehrere Dimensionen. Im Allgemeinen beschränkt man sich hier zur Vereinfachung auf zweidimensionale Daten, wobei die Verallgemeinerung auf k-dimensionale Daten trivial ist.

Anstatt wie bei Binärbäumen den Raum bei jedem Schritt in zwei Unterbäume zu teilen, werden bei Quadtrees vier Kinder verwendet, um alle möglichen Richtungen in zwei Dimensionen bezüglich des aktuellen Knoten abzudecken. Der Name „Quadtree“ leitet sich hiervon ab (lat. „quadri-“: „vier-“). Analog dazu bezeichnet man 3-dimensionale Bäume als „Octtrees“. Es ist weiters einfach zu sehen, dass k-dimensionale Bäume 2^k Kinder pro Knoten aufweisen.[1]

Räumliche Strukturen wie Quadtrees sind generall dann sinnvoll, wenn die Entfernung zweier Punkte für die Anwendung relevant ist. Beispiele hierfür sind unter anderem Repräsentation von Bildern (angrenzende Bereiche besitzen oftmals ähnliche Farben) oder Kollisionserkennung, wo das Interesse sogar auf sich überschneidende Strukturen liegt.

Beim Zugriff auf die so in einem Quadtree gespeicherten Daten unterscheidet man zwischen verschiedenen Anfragetypen:[2]

Point Query. Mit dieser Anfrage versucht man herauszufinden, welche Daten sich bei einem bestimmten Punkt befinden. Anworten auf diese Anfrage können entweder *NULL* bzw. ein Datenobjekt sein.

Range Query. Hier interessiert man sich für die Knoten, welche innerhalb von Bereichen, die hinsichtlich mehreren Dimensionen ausgedehnt sein können, liegen. Zurückgegeben wird eine möglicherweise leere Liste von Datenobjekten.

[1] Vgl. [BF79], Seite 16
[2] Basierend auf [Knu98], Seite 559 und der Erweiterung durch [Ben75a]

Neighborhood Query. Dieser Anfragetyp fokussiert sich auf Nähebeziehungen. Konkret werden die Knoten gesucht, welche die geringste Entfernung zu einem bestimmten Knoten aufweisen. Der Rückgabewert kann hier eine Liste oder ein einzelnes Datenobjekt sein.

1.1. Funktionsprinzip

Wie eingangs erwähnt wird bei einem Quadtree der Raum in einem Schritt in zwei Dimensionen geteilt. In diesem Kontext bezeichnet man die Abschnitte der Unterteilung als „Quadranten“. Dies kann, wie Abbildung 1 auf Seite 4 demonstriert, direkt an den Koordinaten der Knoten stattfinden, je nach Quadtreetyp werden hier jedoch verschiedene Strategien eingesetzt.

Die einzelnen Knoten eines Baumes beinhalten Verweise auf vier Kindstrukturen, welche diejenigen Daten beinhalten, die sich vollständig in der entsprechenden Richtung des Knotens befinden. Gleichzeitig sind alle Knoten bei einem großen Teil der Quadtreetypen zugleich auch Datenknoten und enthalten somit die zu speichernden Werte. Die Koordinaten der zu speichernden Daten bestimmen demnach die Gestalt der Struktur.

Dieses Prinzip der rekursiven Dekomposition erlaubt das effiziente Suchen in logarithmischer Zeit.[3]

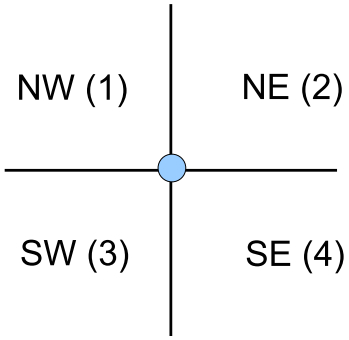

Abbildung 1: Alle Kinder repräsentieren Teilbäume, welche sich vollständig in NW-, NE-, SW- oder SE-Richtung des aktuellen Knoten befinden. Je nach Konvention können die Quadranten auch mittels ihrer Nummerierung referentiert werden.

[3]Siehe Kapitel 2.1.1 auf Seite 6

2. Punktbasierte Quadtrees

2.1. Point Quadtree

Dieser Quadtreetyp wurde erstmals 1974 von Finkel und Bentley in [FB74] eingeführt.[4] Point Quadtrees kann man als direkte Verallgemeinerung von binären Suchbäumen auf mehrdimensionale Räume auffassen. Ähnlich zu diesen erfolgt die Aufteilung direkt an den Koordinaten der Knoten, sowie an beiden Dimensionen gleichzeitig.

Abbildung 2 auf Seite 5 zeigt eine Instanz eines Point Quadtrees in zwei üblichen Darstellungsvarianten: Baum- und Graphendarstellung. In der Ersteren wird nur die Struktur des Baumes betrachtet, die Lage der Datenknoten ist ausschließlich relativ zueinander sichtbar. Die Zweitere konzentriert sich mehr auf die absolute Position der Punkte in einem Koordinatensystem und zeigt zusätzlich dazu die rekursive Dekompositionsstruktur ein. Die Bezeichnung der Knoten in der Grafik gibt die Reihenfolge der Einfügung an. Bereits hier wird deutlich, dass die Form des Baumes durch diese Reihenfolge determiniert wird.[5]

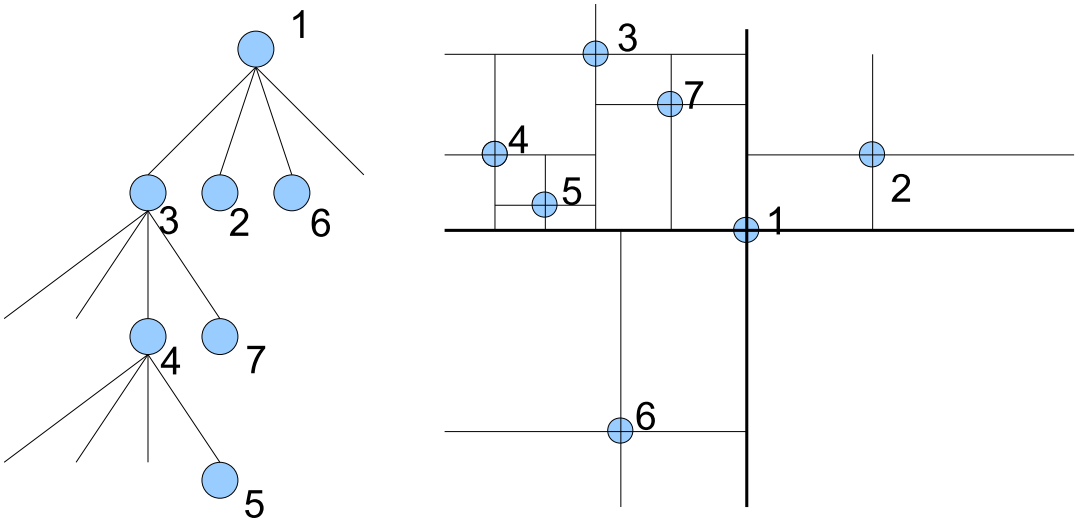

Abbildung 2: Gegenüberstellung: Ein Point Quadtree in Baum- bzw. Graphendarstellung

[4] Vgl. [BKOS00], Seite 318
[5] Vgl. Kapitel 2.1.2 auf Seite 6

2.1.1. Suche

Der Algorithmus zur Suche von Koordinaten (Point Query) ergibt sich unmittelbar aus dem rekursiven Aufbauprinzip des Quadtrees.

Die Suche beginnt beginnt an der Wurzel. In jedem Schritt werden je ein Vergleich der x- und y-Koordinate des aktuellen Knotens mit dem gesuchten Knoten durchgeführt, wodurch der Quadrant bestimmt wird, in welchen sich der gesuchte Knoten befindet. Ist dieser Quadrant leer, d.h. ist kein Kindknoten bezüglich dieser Richtung am aktuellen Knoten vorhanden, wird die Suche erfolglos abgebrochen; andernfalls wird sie bei diesem Kindnoten fortgesetzt, bis die Koordinaten des gesuchten Knoten mit jenen des aktuellen Knoten übereinstimmen.

Durch die rekursive Aufteilung kann hier im Average Case mit logarithmischer Laufzeit gerechnet werden ($O(log_4\ N)$). Allerdings hängt diese Größe von der Balanzierung des Baumes ab, wofür es beim Point Quadtree keine Garantien gibt. Dies führt im Falle einer Entartung zu einer linearen Laufzeit ($O(n)$).[6]

2.1.2. Einfügen

Das Verfahren zum Einfügen neuer Daten baut direkt auf die Suche auf. In dem Quadranten, in welchem eine normale Suche abbrechen würde, wird das Datum eingefügt, indem an der entsprechenden Stelle, die sich offensichtlicher Weise in diesem Quadranten befinden muss, der Datenknoten platziert wird. Der asymptotische Aufwand dieser Operation ist trivialerweise identisch mit jenem der Suche.

Der neue eingefügte Knoten teilt den Raum wiederum an seinen x- bzw. y-Koordinaten und erweitert so die Struktur des Baumes. Es ist schnell ersichtlich, dass hier, genau wie bei einem Binärbaum, Entartungen auftreten können, da sich die Struktur dynamisch mit jedem eingefügten Element herausbildet. Ein besonders Problem stellen hier etwa geordnete Daten da. Werden diese in der gegebenen Reihenfolge eingefügt, degeneriert der Baum zu einer linearen Listen. Sind andererseits beim Aufbau des Baumes bereits alle Daten verfügbar, kann durch folgende Strategie ein optimale Balanzierung erreicht werden: Der Median der sortierten Liste von Daten wird als Wurzel verwendet, die restlichen Elemente werden in 4 Gruppen eingeteilt,

[6] Vgl. [Sam90], Seite 52

welche den Quadranten dieser Wurzel zugeordnet werden. Auf jede dieser Unterteilungen wird der Prozess rekursiv angewendet.[7]

2.1.3. Löschen

Das Entfernen eines Knotens aus einem Quadtree ist im Allgemeinen eine komplexe Operation, da die Knoten alleine die Struktur des Baumes bilden, und diese auch nach dem Löschen weiterhin gewisse Eigenschaften innehaben muss. Bei Binärbäumen stellt dies kein allzu großes Problem dar, da in einem eindimensionalen Raum immer ein geeigneter Ersatz in Form des nähesten Knotens für einen zu entfernenden Knoten zur Verfügung steht.[8] Im Fall von Quadtrees muss eine Substitution hingegen Bedingungen hinsichtlich zweier Dimensionen genügen, wodurch es ein Näheverhältnis wie im eindimensionalen Fall nicht gibt.[9] Hier ist ein ein Ersatz nur dann möglich, wenn ein Blattknoten existiert, für den sich im Bereich zwischen den Achsen, welche durch die Koordinaten des Blattknoten bzw. deren des zu entfernenden Knoten verlaufen, kein anderer Knoten befindet.[10]

Ursprünglich wurde vorgeschlagen, alle Kinder des zu entfernenden Knoten neu einzufügen, was einen sehr teuren Prozess darstellen kann.[11]. Sollte kein Knoten das oben genannte Kriterium erfüllen, kann nur durch günstige Wahl des Ersatzknotens die Anzahl der neu einzufügenden Knoten minimiert werden, wie etwa in [Sam80] gezeigt wird.

Eine weitere Optimierung kann dadurch erreicht werden, dass gelöschte Knoten nicht aus dem Baum entfernt, sondern nur als gelöscht markiert werden, um so die Struktur zu erhalten. Die markierten Knoten dürfen bei zukünftigen Suchen nicht mehr zurückgegeben werden. Nach jeder Entfernung erhöht sich hierdurch der Anteil der „toten“ Knoten, was sich negativ auf die Performance aller Operationen auswirkt, und periodische Restrukturierungsmaßnahmen des Baumes erfordert. Durch dieses Verfahren kann folglich der Aufwand einzelner Löschoperationen abgefedert werden, auf Kosten von regelmäßigen Wartungskosten.

[7] Vgl. [Sam90], Seite 52f
[8] Vgl. [Sam80], Seite 2
[9] Vgl. Ebenda
[10] Vgl. [Sam80], Seite 3
[11] Vgl. [FB74]

2.2. Pseudo Quadtree

Der Pseudo Quadtree ist eine Abwandlung des Point Quadtrees, welche Löschen in logarithmischer Zeit sowie einfachere Rebalanzierung bei Deformation ermöglicht. Diese Form des Quadtrees wurde 1982 von Overmars und van Leeuwen in [OL82] entwickelt.

Dem Pseudo Quadtree liegt die Analyse zu Grunde, dass die Probleme beim Entfernen von Knoten und bei der Balanzierung bei Point Quadtrees daher stammen, dass die Datenpunkte selbst die Struktur des Baumes bilden, und so nur äußerst wenig Flexibilität vorhanden ist. Aus diesem Grund wurde unterscheidet sich der Pseudo Quadtree davon insofern, als dass die Dekomposition des Raum an beliebigen Punkten stattfinden kann.

Konkret funktioniert das Einfügen beim Pseudo Quadtree wie folgt: Der Quadrant eines Datenpunktes wird gesucht. Ist dieser leer, wird der Knoten hier eingefügt, ohne den Raum an dessen Koordinaten weiter zu unterteilen. Befindet sich hier hingegen bereits ein anderer Knoten, muss eine neue Untergliederung eingeführt werden. Hier können beliebige Strategien verwendet werden, um einen Teilungspunkt zu bestimmen, wodurch ein der Anwendung angepasster, gut balanzierter Baum entstehen kann.

Abbildung 3 auf Seite 9 zeigt einen Pseudo Quadtree, welcher dieselben Daten wie der Point Quadtree in Abbildung 2 auf Seite 5 beinhaltet. In diesem Fall wurde der Mittelpunkt von zwei Knoten als Punkt zur Teilung verwendet.

Sollte trotz einer angepassten Strategie ein Ungleichgewicht entstehen, kann der Subbaum, dessen Wurzel der höchste Knoten ist, bei welchem eine unbalanzierte Verteilung seiner Kindknoten festgestellt wird, neu strukturiert werden, indem eine neue Aufteilung der vorhandenen Knoten berechnet wird.[12] Es kann gezeigt werden, dass im allgemeinen Fall eine Teilung möglich ist, in der sich in jedem der 4 Quadranten höchstens ein Drittel der Knoten befinden.[13]

Durch diese Aufteilung des Raumes an beliebigen Punkten befinden sich Daten schließlich nur noch an den Blättern. Dies wiederum ist der optimale Fall bezüglich dem Entfernen von Knoten – ein Knoten ohne Kinder kann jederzeit ohne zusätzlichen Aufwand gelöscht werden. Da das Entfernen eines Knotens immer auch das Suchen desselben erfordert, entsteht somit ein

[12] Vgl. [OL82], Seite 5
[13] Vgl. [OL82], Seite 7

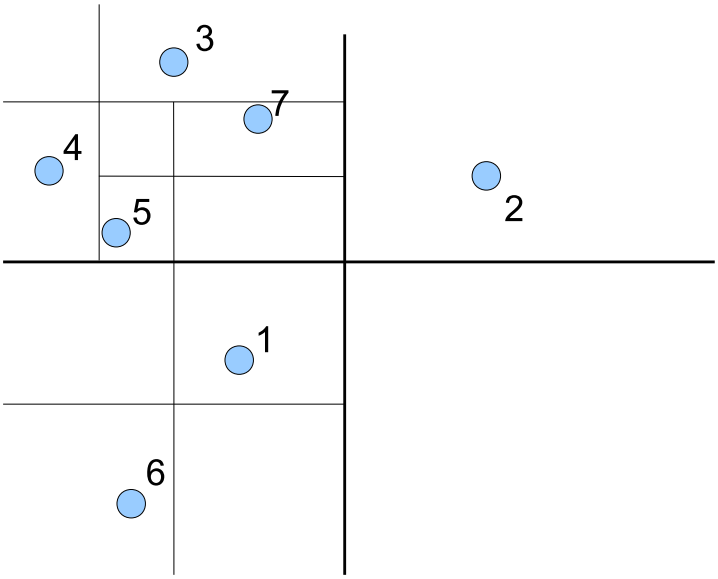

Abbildung 3: Pseudo Quadtree

logarithmischer Aufwand.

2.3. *k*-d Tree

Diese Variante wurde von Bentley kurz nach dem Point Quadtree 1974 in [Ben75b] entwickelt. Der Name k-d deutet an, dass dieser Baum k-dimensionale Schlüssel verwendet.

Der k-d Tree kann wie der Point Quadtree als Verallgemeinerung des Binärbaums gesehen werden, hier wird hingegen ein etwas anderer Ansatz verfolgt: Anstatt den Raum gleichzeitig in allen Dimensionen zu partitionieren und somit die Notwendigkeit von 2^k Kindern pro Knoten zu erfordern, wird in jedem Schritt nur eine Dimension berücksichtigt und so eine binäre Unterteilung von eigentlich mehrdimensionalen Schlüsseln ermöglicht. Diese Mehrdimensionalität wird bei k-d Trees weiters dadurch ausgedrückt, dass die Dimensionen, bezüglich derer an einer gewissen Tiefe geteilt wird, beliebig alterniert. Im zweidimensionalen Fall könnte hier etwa bei geraden Tiefenebenen an der x-Achse bzw bei ungeraden an der y-Achse geteilt werden, wie etwa Abbildung 4 auf Seite 10 zeigt. Ursprünglich wurde genau dieses Abwechslungsmuster vorgeschlagen,[14] wobei dieses auch dynamisch beliebig gewählt und so an die Daten angepasst werden kann[15]. Verwendet man hier einen dynamischen Ansatz, muss jedoch bei jedem Knoten

[14] Vgl. [Ben75b], Seite 2
[15] Vgl. [Sam90], Seite 66

der Diskriminator mitgespeichert werden.

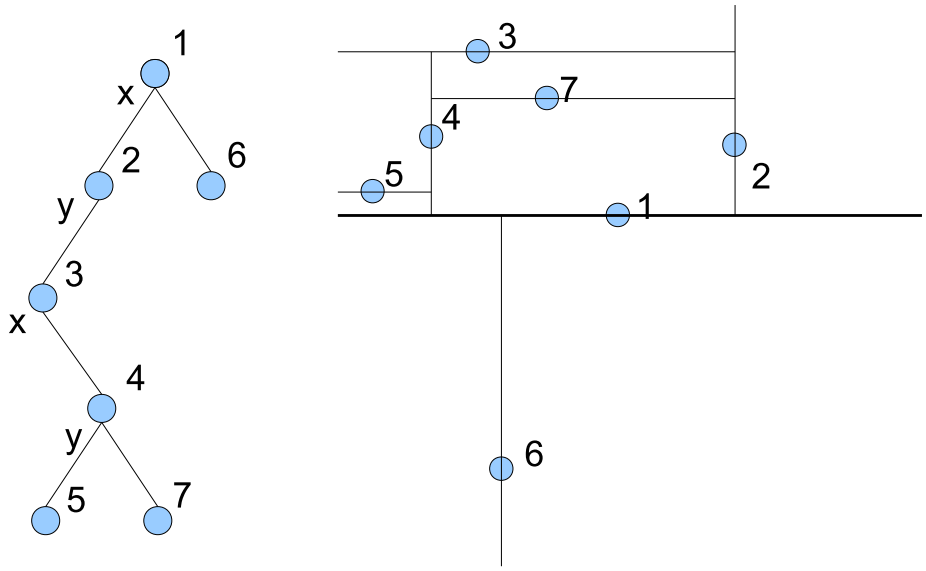

Abbildung 4: Ein k-d Tree: Die Beschriftung links der Unterteilungen gibt die Achse an, bezüglich welcher die Knoten aufgeteilt werden.

Bezüglich den vorhergegangenen Abbildung wurden hier die Koordinaten der Knoten leicht verändert, da in diesem Fall ein äußerst schlecht balanzierter k-d Tree entstehen würde, was der Autor aus Demonstrationszwecken vermeiden möchte.

Dieser Typ bietet gegenüber einem Point Quadtree Vorteile bezüglich des Platzbedarfs: anstatt 2^k sind hier nur 2 Kinder pro Knoten erforderlich. Dies löst das Problem von vielen unnötigen Leereinträgen, welche bei einem Point Quadtree bei einer großen Dimensionsanzahl auftreten können.

Die Operationen Suche und Einfügen ergeben sich direkt aus dem Funktionsprinzip des k-d Trees, beide benötigen offensichtlich eine logarithmische Laufzeit im Average Case[16] (sofern der Baum nicht entartet). Schwieriger ist hingegen das Löschen: Nach dem Entfernen eines Knotens darf die Struktur des k-d Trees nicht verletzt werden, weshalb ein geeigneter Ersatz gefunden werden muss. Für dieses Problem wurde eine Strategie entwickelt, welche auf der Lösung des analogen Problems bei Binärbäumen basiert und zugleich die Mehrdimensionalität im k-d Tree berücksichtigt.[17] Anstatt im Worst Case (bei Löschung der Wurzel) n Neueinfügungen mit dem

[16] Vgl. [Ben75b], Seite 1
[17] Für eine detaillierte Beschreibung dieses Verfahrens siehe [Ben75b], Seite 7

naiven Ansatz benötigt diese Methode nur $O(n^{1-\frac{1}{k}})$.

Um diesem Problem gänzlich zu entgehen kann ähnlich dem Pseudo Quadtree[18] auch ein Pseudo k-d Tree[19] verwendet werden, in welchem analog zum Pseudo Quadtree die Unterteilung des Raumes nicht an den Koordinaten der Knoten, sondern an beliebigen Punkten vorgenommen wird, und so Daten nur in Blättern gespeichert werden.

3. Bereichsbasierte Quadtrees

Wie im Kapitel 2 ausführlich erläutert wurde, erfolgt die räumliche Dekomposition bei punktbasierten Quadtrees immer anhand der gegebenen Punkte.[20] Dieses Kapitel beschäftigt sich mit Quadtrees, deren Form vordergründig a priori durch die Struktur des Raumes determiniert wird, was in etwa mit dem Prinzip von Tries vergleichbar ist.[21]

Weiters wird in diesem Abschnitt nur insofern auf die verschiedenen Operationen eingegangen, als sie sich von punktbasierten Quadtrees unterscheiden.

3.1. MX Quadtree

Der MX Quadtree zeichnet durch seine Art der Raumunterteilung aus: Der Bereich, in dem die Datenpunkte liegen, wird unabhängig von den konkreten Daten unterteilt, und zwar rekursiv in vier gleichgroße Quadranten, bis auf der untersten Ebene 1x1 große Felder erreicht werden, welche die Daten beinhalten.[22]

Diese Art der Dekomposition setzt folgendes vorraus:

(1) **Konstanter Bereich.** Die Region, in welcher sich Daten befinden, muss a priori bekannt sein und während des gesamten Programmablaufs konstant bleiben.

(2) **Diskrete Koordinaten.** Die Schlüssel dürfen nur diskrete Werte annehmen, ähnlich den Einträgen einer Matrix, woher sich auch der Name „MX“ ableitet.[23]

[18] Siehe Kapitel 2.2
[19] Vgl. [OL82]
[20] Bei Point bzw. k-d Trees gilt dies direkt; bei Pseudo Quadtrees hingegen besteht die Abhängigkeit vielmehr indirekt: die Aufteilung stellt eine Funktion der Punkte dar.
[21] Vgl. [Sam90], Seite 85
[22] Vgl. [SW85], Seite 1f
[23] Vgl. [Sam90], Seite 86

(3) **Fläche des Raumes beträgt 2^n x 2^n.** Da der Raum aus praktischen Gründen in 4 gleichgroße Teile partitioniert wird, entsteht eine diese Gesamtgröße. Bei Datenregionen, welche nicht diese Ausdehnung besitzen, können Leereinträge verwendet werden, um diese Größe zu erreichen.

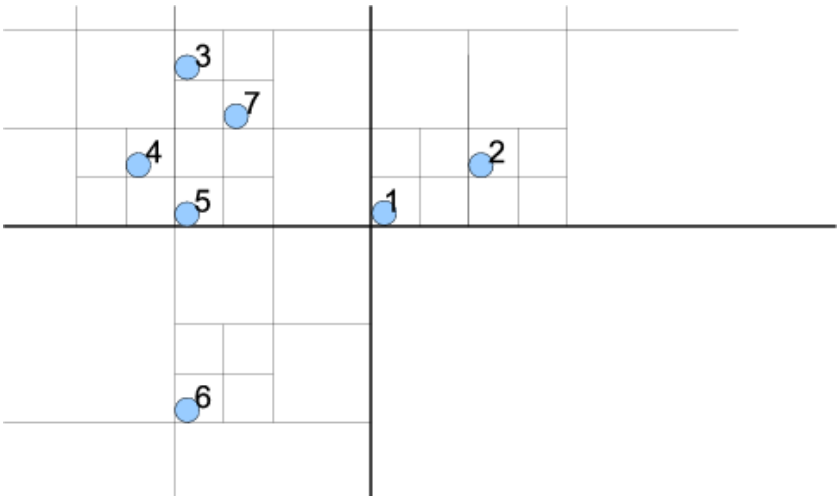

Abbildung 5: MX Quadtree. Diese Instanz ist eher spärlich besetzt.

Abbildung 5 auf Seite 12 zeigt dieselben Daten wie in den vorhergegangen Grafiken in Form eines MX Quadtrees. Dabei fällt auf, dass die Aufteilung des Raumes hier unabhängig von der Reihenfolge des Einfügens der Knoten ist, was eine Konsequenz aus der Partitionierung auf Basis der Raumstruktur ist. Dies führt auch zu einer konstanten Höhe h des Baumes ohne Bezug zu den konkret vorhandenen Datenblättern, wodurch wiederum nur die begrenzte Anzahl von Knoten $2^h * 2^h$ in einem MX Quadtree gespeichert werden können. Diese Einschränkung ist jedoch durch die obigen Bedingungen gesichert.

Ein klassischer Anwendungsfall dieses Prinzip sind Rastergrafiken, welche ein geeignetes mentales Modell für MX Quadtrees darstellen, da hier jeder Datenpunkt (Pixel) offensichtlich genau 1x1 groß ist und einer Koordinate zugeordnet ist.[24] Davon abgeleitet spricht man in diesem Kontext von weißen, schwarzen und grauen Knoten[25]:

Schwarze Knoten sind entweder Datenblätter oder Knoten mit vier schwarzen Kindern.

[24] Vgl. [HS79]
[25] Vgl [Gar82], Seite 2

Graue Knoten stellen Knoten dar, welche mindestens ein graues Kind besitzen oder mindestens je ein schwarzes und ein weißes Kind.

Weiße Knoten sind leere Blätter ohne Daten bzw. Knoten, die weder direkt noch indirekt mittels Kindverweise auf Daten verweisen. Diese werden im Allgemeinen nicht explizit repräsentiert.

Ein konkreter MX Quadtree enthält somit nur schwarze und graue Knoten. Das Nicht-repräsentieren weißer Knoten führt dazu, dass die Struktur des Baumes erst beim Einfügen aufgebaut wird. Dies ist jedoch nur eine Optimierung; es wäre genauso möglich, den vollständigen Baum mit weißen Knoten bzw. Blättern aufzufüllen, welche später grau oder schwarz eingefärbt bzw. von schwarzen Blättern ersetzt werden.

Das Löschen von Knoten im MX Quadtree ist weiters trivial, da im Gegensatz zu Point Quadtrees hier die Daten keinen Einfluss auf die Struktur des Baumes haben und nur in Blättern gespeichert werden. Nach dem Entfernen des eigentlichen Datenblattes muss nur die Färbung der Knoten am Pfad von der Wurzel zu diesem Blatt überprüft und gegebenenfalls angepasst werden.

3.2. PR Quadtree

Die Einschränkung durch diskrete Koordinaten bei MX Quadtrees ist manchmal in der Praxis nicht tragbar. Nichtsdestoweniger ist die Eigenschaft, dass dieser Baumtyp nicht entarten kann (da die Einfügereihenfolge keine Rolle spielt), oft sehr sinnvoll.

Der PR Quadtree stellt eine Möglichkeit dar, dieses Problem zu Lösen. Er ist eine Kombination von Point und Region Quadtree, daher auch der Name. Hier orientiert sich die Aufteilung des Raumes ähnlich zu MX Quadtrees wiederum an der rekursiven Dekomposition eines fixen Bereiches, dieser kann jedoch beliebig werden.

Die Einfügeoperation läuft wie folgt ab: Der tiefste Quadrant, welcher die Koordinaten eines einzufügenden Knotens beinhaltet, wird gesucht. Ist dieser leer, kann der Knoten hier platziert werden. Andernfalls muss eine weitere Unterteilung vorgenommen werden, was durch Partitionierung des Quadranten in gleichgroße Teile geschieht. Der Knoten, welcher sich in dem

Quadranten befand, wird in die neue Unterteilung überführt, und es wird neuerlich versucht, den einzufügenden Knoten in diese zu integrieren. Sollte es hier erneut zu Konflikten kommen, wird der Prozess der Partitionierung rekursiv so lange wiederholt, bis beide Knoten in verschiedene Quadranten eingeteilt werden. Dies ist trivialerweise garantiert, wenn die Koordinaten verschieden voneinander sind.

Bei diesem Vorgang ist die Dekompositionsstruktur offensichtlich unabhängig von den konkreten Schlüsseln, wodurch die Einfügereihenfolge irrelevant wird und der Baum nicht im Sinne von Point Quadtrees entarten kann.[26].

Diese Strategie kann jedoch problematisch werden, wenn Datenpunkte nahe aneinander liegen. In diesem Fall müssen viele Unterteilungen getroffen werden, bis sie in verschiedene Quadranten eingeteilt werden können. Diese Struktur ist somit mit dieser Einschränkung für alle Schlüsseltypen geeignet.[27]

Das Löschen aus einem PR Quadtree ist wiederum unkompliziert, da sich die Daten nur in Blättern befinden. Durch das Entfernen eines Blattes können Unterteilungen überflüssig werden, wenn etwa ein Knoten nur ein Kind besitzt. In diesem Fall kann der entsprechende Knoten einfach durch das Kind ersetzt werden.

[26] Vgl [Sam90], Seite 96
[27] Vgl [Sam90], Seite 95f

Teil II.

Praxis

4. Praktikum

4.1. Problemstellung

4.2. Implementierung

4.3. Analyse

4.4. Conclusio

Abbildungsverzeichnis

Alle in dieser Arbeit verwendeten Bilder wurden vom Autor erstellt.

Literatur

[Ben75a] Bentley, Jon L.: A survey of techniques for fixed radius near neighbor searching. Stanford, CA, USA : Stanford University, 1975. – Forschungsbericht

[Ben75b] Bentley, Jon L.: Multidimensional binary search trees used for associative searching. In: *Commun. ACM* 18 (1975), September, 509–517. http://dx.doi.org/http://doi.acm.org/10.1145/361002.361007. – DOI http://doi.acm.org/10.1145/361002.361007. – ISSN 0001–0782

[BF79] Bentley, Jon L. ; Friedman, Jerome H.: Data Structures for Range Searching. In: *ACM Comput. Surv.* 11 (1979), December, 397–409. http://dx.doi.org/http://doi.acm.org/10.1145/356789.356797. – DOI http://doi.acm.org/10.1145/356789.356797. – ISSN 0360–0300

[BKOS00] Berg, Mark de ; Kreveld, Marc van ; Overmars, Mark ; Schwarzkopf, Otfried: *Computational Geometry: Algorithms and Applications*. Second. Springer-Verlag, 2000. – 367 S. http://www.cs.uu.nl/geobook/

[FB74] Finkel, Raphael A. ; Bentley, Jon L.: Quad Trees: A Data Structure for Retrieval on Composite Keys. In: *Acta Inf.* 4 (1974), S. 1–9

[FGPR91] Flajolet, Philippe ; Gonnet, Gaston ; Puech, Claude ; Robson, J. M.: The analysis of multidimensional searching in quad-trees. In: *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 1991 (SODA '91). – ISBN 0–89791–376–0, 100–109

[Gar82] Gargantini, Irene: An Effective Way to Represent Quadtrees. In: *Commun. ACM* 25 (1982), Nr. 12, S. 905–910

[HS79] Hunter, G.M. ; Steiglitz, K.: Operations on Images Using Quad Trees. 1 (1979), April, Nr. 2, S. 145–153

[Knu98] Knuth, Donald E.: *The Art Of Computer Programming, Volume 3: Sorting and Searching*. Second. Redwood City, CA, USA : Addison Wesley Longman Publishing Co., Inc., 1998. – ISBN 0–201–89685–0

[Lue78] Lueker, George S.: A Data Structure for Orthogonal Range Queries. In: *FOCS*, 1978, S. 28–34

[OL82] Overmars, Mark H. ; Leeuwen, Jan van: Dynamic Multi-Dimensional Data Structures Based on Quad- and *K-D* Trees. In: *Acta Inf.* 17 (1982), S. 267–285

[Sam80] Samet, Hanan: Deletion in Two-Dimensional Quad Trees. In: *Commun. ACM* 23 (1980), Nr. 12, S. 703–710

[Sam84] Samet, Hanan: The Quadtree and Related Hierarchical Data Structures. In: *ACM Comput. Surv.* 16 (1984), Nr. 2, S. 187–260

[Sam90] Samet, Hanan: *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990

[SW85] Samet, Hanan ; Webber, Robert E.: Storing a collection of polygons using quadtrees. In: *ACM Trans. Graph.* 4 (1985), July, 182–222. `http://dx.doi.org/http://doi.acm.org/10.1145/282957.282966`. – DOI http://doi.acm.org/10.1145/282957.282966. – ISSN 0730–0301