Kapitel 7

Point Quadtree[1]

7.1 Einführung

Der Point Quadtree wurde 1974 erstmals von Finkel und Bentley vorgestellt ([Fink74]). Er ist die Generalisierung der Binärsuche auf mehrere Dimensionen. Jeder Knoten des Baumes repräsentiert dabei geometrische Daten mit dem zugehörigen Punkt in dem n-dimensionalen Raum, sowie den Verweisen auf die Söhne. Die Anzahl der Söhne ist 2^n. Die Verweise im 2-dimensionalen geometrischen Raum stehen für die vier Himmelsrichtungen (oder Quadranten) Nordwest (NW), Nordost (NE), Südwest (SW) und Südost (SE) von den Koordinaten des Knotens aus gesehen (Abbildung 7.1). Ein Knoten ist durch seine Koordinaten im Baum eindeutig bestimmt. Im nachfolgenden Text wir immer von 2-dimensionalen Punkten ausgegangen. Dabei ist jeder Punkt maximal einmal vorhanden. Er dient als Schlüssel zur Identifikation des Knotens im Baum. Zwei unterschiedliche Darstellungen eines Point Quadtree sind in Abbildung 7.2 gegeben.

Sollten verschiedene Datenobjekte mit den gleichen Koordinaten vorhanden sein, speichert man diese mittels einer Liste in einem Knoten. Die Erweiterung der Punkte auf höhere Dimensionen ist problemlos möglich.

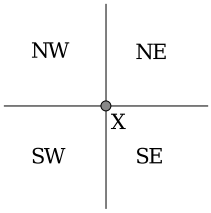


Abbildung 7.1: Knoten eines Point Quadtree und seine vier Himmelsrichtungen

[1] Verfasser Tobias Peitzsch

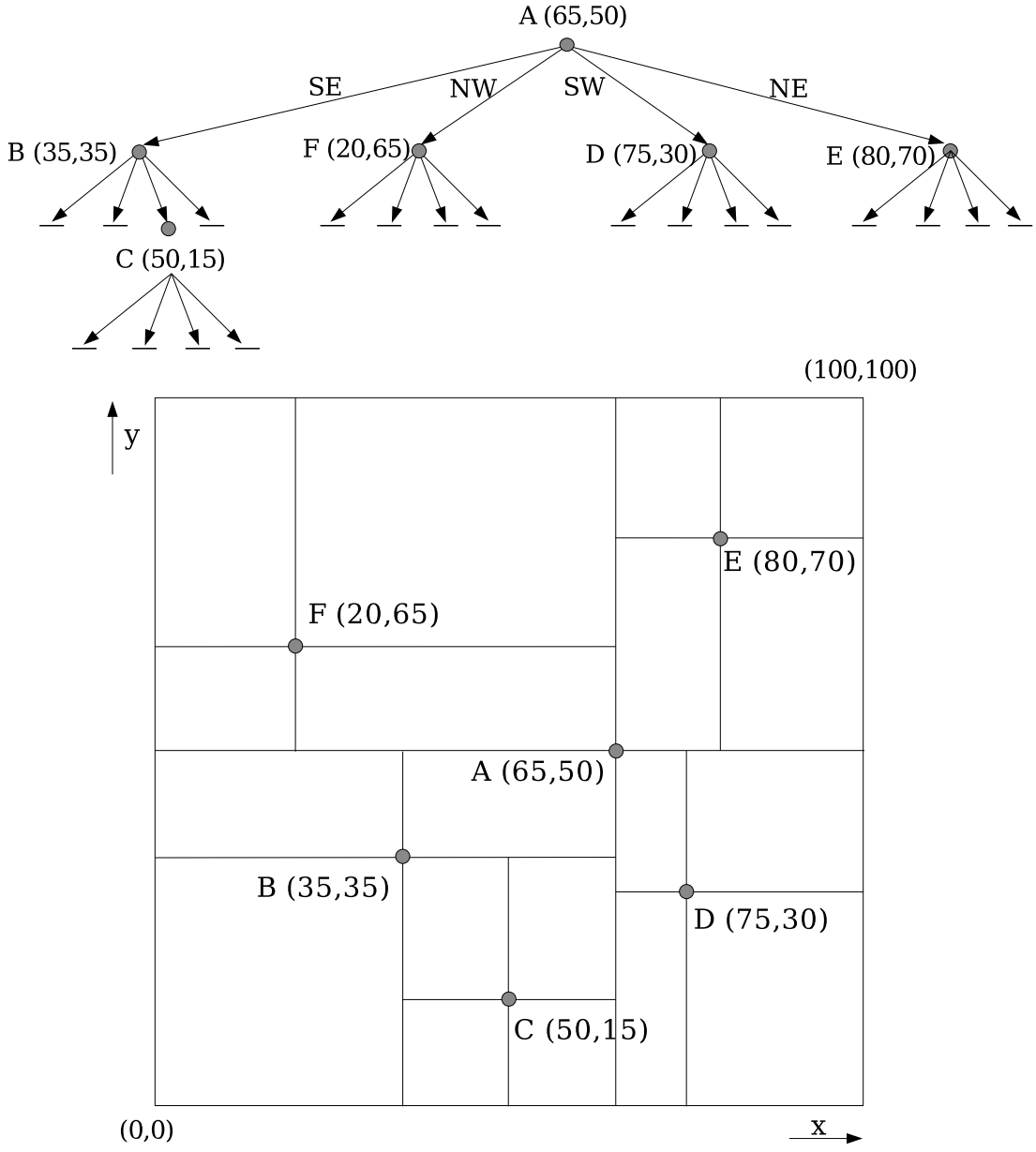


Abbildung 7.2: Point Quadtree und zwei seiner geometrischen Repräsentation

7.2 Punktsuche

Die Suche nach einem Knoten in einem Point Quadtree erfolgt analog zur Suche in einem Binärbaum. Der Startpunkt ist die Wurzel des Point Quadtree. Ist der gesuchte Knoten verschieden von der Wurzel, bestimmt man mittels der Koordinaten der Wurzel und des zu suchenden Knotens den Quadranten, in dem der zu suchende Knoten gefunden werden könnte . Die Vergleiche werden entsprechend dem Pseudocode:

```
quadrant procedure COMPARE(P,R)
begin
/*Liefert den Quadranten des Knotens R, in dem der Knoten P gefunden werden kann*/
        if(CoordX(P) < CoordX(R))
                if(CoordY(P) < CoordY(R)) return 'SW'
                else 'NW'
        else
                if(CoordY(P)< CoordY(R)) return 'SE'
                else return 'NE'
end
```

durchgefhrt. Auf den so gefunden Quadranten mit dem enthaltenen Teilbaum T_B wendet man das Verfahren rekursiv auf die Wurzel von T_B an. Die Rekusion endet, wenn der gesuchte Knoten gefunden wurde oder ein Knoten ohne Kinder erreicht ist und die Suche erfolglos endet.

Der Pseudocode stellt auch klar, wie Knoten A und B mit einer gleichen Koordinate behandelt werden. Stimmen die beiden Knoten in der x- Koordinate berein und ist die y- Koordinate von A kleiner als die von B , so ist der Quadrant SE. Anderenfalls ist der Quadrant NE. Analoges gilt, wenn die Knoten in der y- Koordinate übereinstimmen (halboffener Quadrant).

7.3 Einfügen

Das Einfügen von Knoten in den Quadtree baut auf dem Suchen eines Knotens auf. Der einzufügende Knoten wird im Quadtree gesucht. Endet die Suche erfolglos in einem Knoten ohne Kinder, fügt man diesem den neuen Knoten als Kind im entsprechendnen Quadranten hinzu. Anderenfalls ist der Knoten schon vorhanden (und die Daten des hinzuzufgenden Knotens werden den Daten des gefundenen Knotens angehängt).

Die Komplexität zur Suche oder zum Einfgen eines Knotens in den Quadtree (gemessen an den Vergleichen) ist im *worst case* $H(Q) - 1$, wobei $H(Q)$ die Höhe des Baumes ist. Dadurch bedingt, ist die *worst case* Komplexität zum Aufbau eines Quadtrees aus n Knoten $\mathrm{O}(n)$. Der Baum ist entartet und jeder Knoten besitzt nur einen Sohn.

Finkel und Bentley haben in [Fink74] gezeigt, dass die Komplexität zum Aufbau eines Point Quadtree mit zufällig gewählten Punkten $\mathrm{O}(n\log_4 n)$ entspricht. Dies ist auch die Komplexität im *average case*. Die Komplexität zum Einfügen eines Knotens in den Baum beträgt im *average case* $\mathrm{O}(\log_4 n)$.

Die Frage zum Aufbau von Quadtrees mit möglichst geringer Höhe wurde vielfach diskutiert und verschiedene Verfahren zur Lösung vorgestellt. In [Fink74] ist eine optimierte Methode zum Aufbau eines Quadtrees beschrieben. Das Ziel dieser Methode ist es, einen Quadtree aufzubauen, so dass ein Teilbaum eines beliebigen Knotens A aus dem Quadtree nicht mehr als die Hälfte der Knoten enhält, die über A erreichbar sind. Zum Aufbau eines Point Quadtree, der diesem Kriterium entspricht, wird von nach den Koordinaten sortierten Knoten ausgegangen. Die Wurzel des Baumes entspricht dem Mittelwert der sortierten Knoten. Die verbleibenden Knoten werden entsprechend ihrer Ordnung auf die vier Richtungen verteilt und das Verfahren rekursiv auf sie angewendet.

Durch die Aufteilung der Knoten erreicht man, dass maximal die Hälfte der Knoten in den Quadranten NW und SW (unter der Voraussetzung, dass die Knoten zuerst nach der x-Koordinaten geordnet sind) liegen.

Nachfolgend wird ein Beispiel für das Einfügen von Knoten in einen Point Quadtree beschrieben. Gegeben seien die in Tabelle 7.1 gegebenen Städte mit ihren Koordinaten. Die Städte werden entsprechend der Reihenfolge in der Tabelle in den Point Quadtree eingetragen. Der zuerst eingefügte Knoten ist demnach die Stadt Erfurt. Sie bildet die Wurzel des Point Quadtree. Danach wird die Stadt Berlin eingetragen. Durch den Vergleich der Koordinaten mit der Wurzel ermittelt sich der Verweis auf den NE- Qadranten. Dieser Verweis zeigt auf keinen Knoten, womit die Einfügeposition der Stadt gefunden ist. Danach wird Leipzig dem Quadtree hinzugefgt. Wieder von der Wurzel beginnend, wird der Quadrant bestimmt. Es ergibt sich der Quadrant NE. Dieser ist schon durch die Stadt Berlin belegt. Also wird das Einfügen an

Stadtname	Koordinaten
Erfurt	(60,50)
Berlin	(80,75)
Leipzig	(70,60)
Hamburg	(50,90)
Köln	(10,55)
München	(65,10)
Frankfurt a. M.	(25,35)
Stuttgart	(35,20)

Tabelle 7.1: Städte zum Einfügen in den Quadtree

diesen Knoten geleitet. Durch den Vergleich mit den Koordinaten von Berlin mit denen von Leipzig ergibt sich der Quadrant SW. Der Pfad von der Wurzel zu diesem Knoten ist also NE- SW. Analog werden die restlichen Städte dem Quadtree hinzugefgt. In Abbildung 7.11 ist das Endergebnis zu sehen.

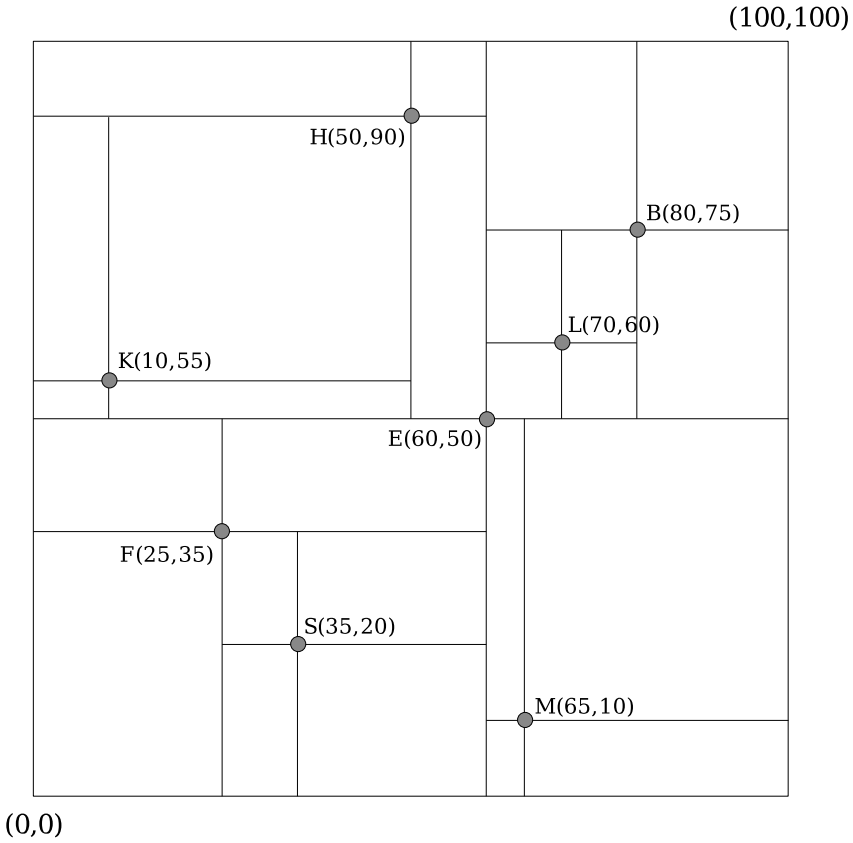


Abbildung 7.3: Quadtree nach dem Einfügen aller Städte

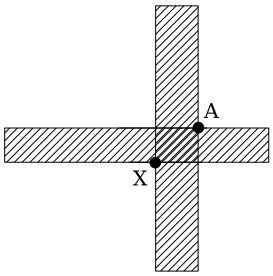


Abbildung 7.4: Ideales Löchen. Das Fenster F ist die schraffierte Fläche und frei von anderen Knoten.

7.4 Löschen

Effizientes Löschen in einem 2-dimensionalen Point Quadtree ist ziemlich komplex. O.B.d.A. kann man annehmen, dass die Wurzel X eines Point Quadtree gelöscht werden soll. Die einfachste Möglichkeit ist den gewünschten Knoten aus dem Baum zu entfernen und seine Kindknoten neu in den Baum einzufügen ([Fink74]). Diese Methode ist aber nicht praktikabel, denn sie läuft auf eine Neuorganisation des Quadtree hinaus.

Ein effizienteres, aber auch komplexes Verfahren zum Löchen von Knoten aus einem Quadtree wurde in [Samet80] vorgestellt. Dieses Verfahren baut in analoger Weise auf dem Löschen in einem Binärbaum auf. Problematisch dabei ist, dass im 2-dimensionalen Raum die lineare Ordung fehlt. Dadurch ist es nicht möglich, wie beim Löschen im Binärbaum, den kleinsten

(größten) Knoten zu finden, der größer (kleiner) als der zu löschende Knoten ist und durch ihn den zu löschenden Knoten zu ersetzen. Man kann pro Quadrant des zu löschenden Knotens einen Knoten finden, der den zu löschenden Knoten ersetzen könnte. Ideal wäre es, einen Knoten A zu finden,

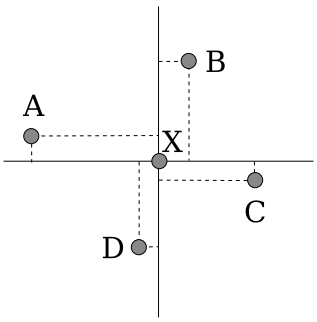


Abbildung 7.5: Quadtree mit keinem idealen Kandidaten

so dass der Bereich (oder Fenster F) zwischen den zu den Koordinatenachsen parallelen Linien (oder Achsen), die von den Koordinaten von A und des zu löschenden Knotens X aufgespannt werden, frei von anderen Knoten ist (Abbildung 7.4). Oftmals existiert ein solcher Knoten nicht (Abbildung 7.5). Es müssen die vier möglichen Kandidaten bestimmt werden. Die Bestimmung von einen Kandidaten erfolgt durch eine Prozedur FIND_CANDIDATE mit dem Pseudocode:

```
procedure FIND_CANDIDATE(P,Q)
/* Prozedur sucht den Kandidaten im Quadrant Q. Dabei ist P der
 * Wurzelknoten des Teilbaumes im Quadrant Q.*/
begin
        if(P==null) return null
        while(SON(P,OPPOSIT_QUADRANT(Q)) !=null)
        begin
                P=SON(P,OPPOSIT_QUADRANT(Q))
        end
        return P
end
```

Das Verfahren ist in Abbildung 7.6 nochmals für den NE- Quadranten graphisch dargestellt. Zum Bestimmen der vier Kandidaten wird der Code auf alle vier Quadranten des zu löschenden Knotens angewandt.

Dann muss aus den Kandidaten der beste ausgewählt werden. Hier wird nach **zwei Kriterien** entschieden:

Das **erste** sucht den Knoten, der am nächsten an den beiden, durch die Koordinaten des zu löschenden Knotens erzeugten, Achsen ist. Bei der Wahl wird immer nur eine "Seite" einer Achse (also die Kandidaten in den Quadranten NW-NE, NE-SE, SW-SE und NW-SW) betrachtet. Dadurch bedingt, passiert es, dass ein solcher Knoten nicht existiert oder dass mehrere Knoten, die diesem Kriterium entsprechen, vorhanden sind. Abbildung 7.7 zeigt drei Beispiele dazu. Es soll der Knoten X gelöscht werden. Die vier Kandidaten sind die Knoten A, B, C und D. Im ersten Beispiel hat der Knoten A für die jeweilige Seite der beiden Achsen den minimalen Abstand (also einmal bzgl. B und einmal bzgl. D). Bei allen anderen Knoten ist der Abstand auf einer Seite der Achsen nicht minimal. Im zweiten Beispiel sind A und C

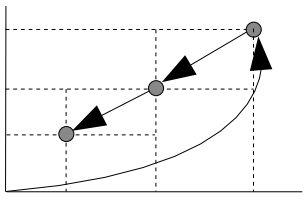


Abbildung 7.6: Graphische Darstellung zum Finden eines Kandidaten

die zwei Knoten, die dem Kriterium entsprechen. Im letzten Beispiel ist kein Knoten vorhanden, der das Kriterium

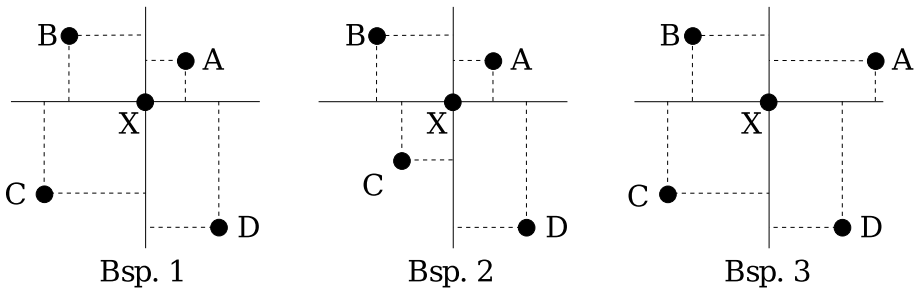


Abbildung 7.7: Drei Beispiele für das erste Kriterium.
Im ersten Beispiel hat der Knoten A den minimalen Abstand. Im zweiten haben die Knoten A und C einen minimalen Abstand. Im dritten Beispiel ist kein Knoten mit einem minimalen Abstand vorhanden.

erfüllt. Durch Anwenden des ersten Kriterium versucht man, einen Knoten zu finden, der möglichst nah am zu löschenden Knoten ist. Bei einer angenommenen Gleichverteilung der Punkte (der Knoten) ist damit die Wahrscheinlichkeit, einen weiteren Knoten in dem Fenster anzutreffen, minimal.

Ist kein Knoten oder sind mehrere vorhanden, kommt das **zweite Kriterium** zum Finden des besten Knotens zur Anwendung. Es wird derjenige Knoten ausgewählt, dessen Abstand vom zu löschenden Knoten bezüglich der L_1- Metrik minimal ist. Die L_1- Metrik heißt auch Manhattan-Metrik (oder city block metric). Sie ist durch $L_1 = d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$ gegeben, wobei $x = (x_1..x_n)$ und $y = (y_1...y_n)$ Punkte aus dem n-dimensionalen Raum sind. Im hier vorliegenden Fall ist n=2.

Das Ziel ist wiederum, die rechteckige Region zwischen dem zu löschenden Knoten und den Kandidaten zu minimieren und somit auch die Wahrscheinlickeit, dass Knoten, die in dieser Region liegen, neu hinzugefügt werden müssen. Die verbleibende Region berechnet sich mit $L_x * d_y + L_y * d_x - 2 * d_x * d_y$. L_x und L_y ist die Länge der beiden Achsen des zu löschenden Knotens und d_x und d_y ist der Abstand des Kandidaten zu den Achsen. Für L_x und $L_y$wird deren Endlichkeit vorausgesetzt. Man kann sagen, dass die verbleibende Region proportional zur Summe von d_x und d_y ist, da der Term $2 * d_x * d_y$ von "hoher Ordnung" klein wird, sowie L_x und L_y konstant ist.

Das zweite Kriterium allein garantiert noch nicht, dass der gefundene Kandidat optimal ist. Es kann sein, dass das Fenster zwischen dem Kandidaten und dem zu löschenden Knoten einen Kandidaten enthält. In Abbildung 7.8 soll Knoten X gelöscht werden. Der Knoten A wird als bester Kandi-

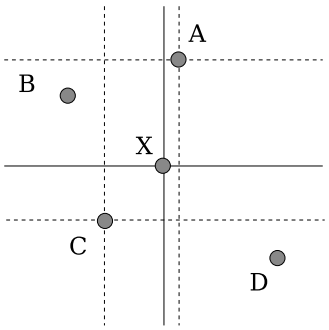


Abbildung 7.8: Beispiel für die Insuffizienz des Kriterium 2 für ein knotenfreies Fenster

date mittels des zweiten Kriteriums gewählt. Er ist aber nicht optimal, da sich in dem Fenster zwischen X und A der Kandidat B befindet. Es ist also zwingend notwendig, erst nach dem ersten Kriterium zu prüfen.

Das zweite Kriterium garantiert, dass ein optimaler Kandidat gefunden wird, wenn das erste Kriterium keinen findet. Es stellt sicher, dass maximal ein weiterer Kandidat in dem Fenster zwischen den optimalen Kandidaten und dem zu löschenden Knoten liegt.

Nach diesen Vorbetrachtungen kann nun der Algorithmus angegeben werden. Dieser besteht hauptsächlich aus den zwei Prozeduren ADJQUAD und NEWROOT. Sei X der zu löschende Knoten und Q der Quadrant von X, der den besten, nach den oben beschriebenen Kriterien gewählten Kandidaten enthält. Dieser Kandidat sei der Knoten A.

Es ist leicht zu erkennen, dass keine Knoten aus den Q gegenberliegenden Quadrant (abgekürzt: $OPQUAD(Q)$) neu eingefügt werden müssen, da diese

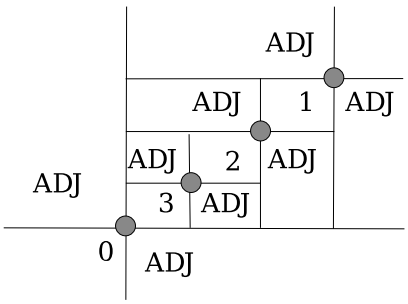


Abbildung 7.9: Bearbeitung der Quadranten, wenn Knoten 0 durch Knote 3 ersetzt wird.

Knoten nicht in dem durch X und A aufgespannten Fenster liegen. In den beiden an Q angrenzenden Quadranten bestimmt man die neu einzufügenden Knoten mit der Prozedur ADJQUAD. Diese Prozedur prüft, ob der Wurzelknoten des Quadranten in dem von X und A aufgespannten Fenster liegt. Ist der Wurzelknoten außerhalb des Fensters, verbleiben automatisch nur zwei Quadranten, auf die die Prozedur rekursiv angewendet wird. Die Knoten aus den beiden anderen Quadranten behalten ihre Position beim Löschen von X im Baum, da sie sich definitiv nicht im Fenser von X und A befinden. Liegt der Wurzelknoten in dem Fenster, müssen der Knoten und seine Kinder neu in den Baum eingefügt werden. Das Einfügen geschieht in dem Quadranten OPQUAD(Q).

Als nächstes bestimmt man die Knoten, die aus dem Quadrant Q neu einzufügen sind. Dies erledigt die Prozedur NEWROOT. Es ist ersichtlich, dass der Subquadrant Q von dem Quadrant Q (bezeichnet als Q') keine Knoten enthält, die neu eingefügt werden müssen. Für die an Q' angrenzenden Quadranten wird die Prozedur ADJQUAD und für den gegenüberliegenden Quadranten $OPQUAD(Q')$ die Prozedur NEWROOT aufgerufen. Die Rekursion endet, wenn der Knoten in $OPQUAD(Q')$ nicht vorhanden ist. Der Ablauf der Prozedur NEWROOT ist in Abbildung 7.9 schematisch dargestellt. Der zu löschende Knoten ist 0, der beste Kandidat ist 3. Eine formale Angabe des Algorithmus kann in [Samet89] gefunden werden.

Zur Verdeutlichung des Löschvorganges wird das Beispiel aus Abschnitt 1.3 fortgeführt. Zu den vorhandenen acht Städten werden noch die Städte Chemnitz(75,55), Halle(65,65) und Wolfsburg(55,75) dem Point Quadtree hinzugefügt. Abbildung 7.10 zeigt den Quadtree mit den neuen Städten.

Es soll die Stadt Erfurt gelöscht werden.

Als erstes bestimmt man die Kandidaten nach dem oben gegebenen Algorithmus. Die vier Kandidaten sind die Städte Wolfsburg (Quadrant NW), Leipzig (NE), München (SE) und Frankfurt. Das erste Kriterium liefert keinen besten Kandidaten. Somit kommt das zweite Kriterium zur Anwendung. Leipzig wird als bester Kandidat bestimmt, da diese Stadt den kleinsten L_1-Abstand zu Erfurt hat. Die Stadt Leipzig wird neuer Wurzelknoten des Point Quadtree. Nun bestimmt man die neu einzufügenden Knoten mit den Prozeduren ADJQUAD und NEWROOT. ADJQUAD durchsucht die an den Quadranten von Leipzig angrenzenden Quadranten NW (mit Wurzel Hamburg) und SE (mit Wurzel München). Hamburg liegt außerhalb des von Erfurt und Leipzig aufgespannten Fensters. Demnach brauchen nur die beiden südlichen Quadranten von Hamburg (mit den Wurzeln Wolfsburg und Köln) mittels der Prozedur ADJQUAD durchsucht werden. Wolfsburg liegt außerhalb des Fensters und hat keine Nachfolger. Die Suche endet hier. Köln dagegen liegt in dem Fenster. Es muss in den Teilbaum mit Frankfurt als Wurzel neu eingefgt werden. Analog verfährt man im Quadranten, wo München Wurzel ist. Hier wird München dem Teilbaum mit der Wurzel Frankfurt hinzugefügt.

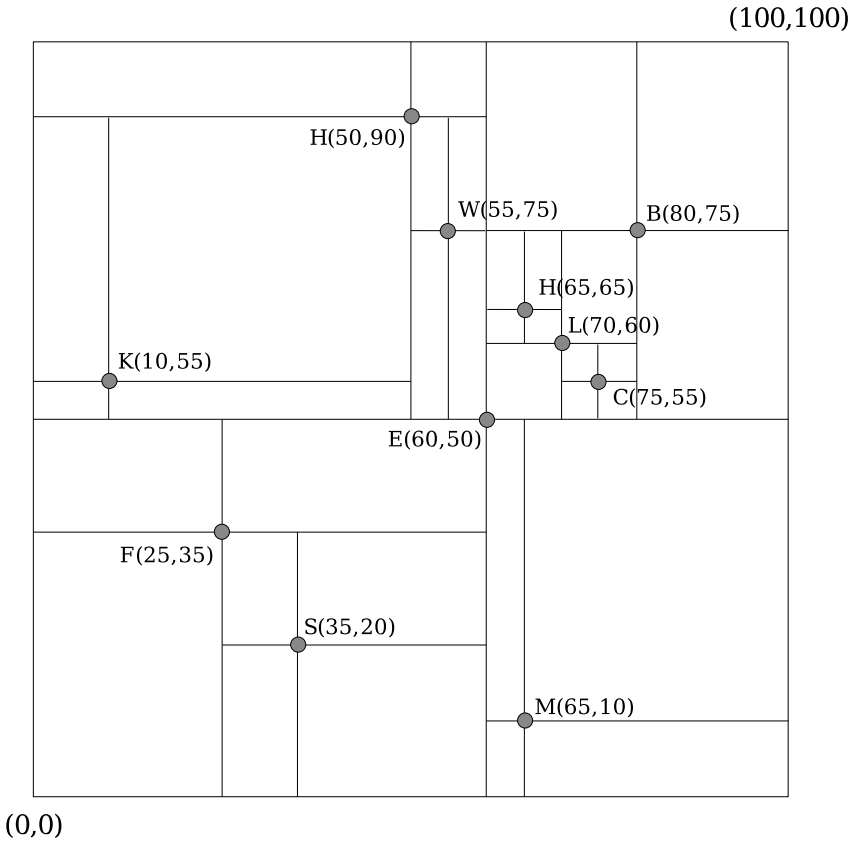


Abbildung 7.10: Quadtree nach dem Einfügen der Städte Chemnitz (C), Halle (H) und Wolfsburg (W)

Jetzt muss noch der Quadrant, der Leipzig enthält, mit der Prozedur NEWROOT bearbeitet werden. Dazu wird die Prozedur mit der Stadt Berlin als Parameter aufgerufen. Diese ruft für ihre NW und SE- Quadranten die Prozedur ADJQUAD auf. Diese Aufrufe verlaufen ins Leere, da die beiden Quadranten keine Knoten enthalten. Die Rekursion der Prozedur wird durch ihren erneuten Aufruf mit der Stadt Leipzig fort gesetzt. Es werden die Quadranten NW und SE von Leipzig mit der Prozedur ADJQUAD durchsucht. Hier werden die Städte Halle und Chemnitz als neu hinzuzufügenden Knoten gefunden. Das Endergebnis ist in Abbildung 7.11 zu sehen.

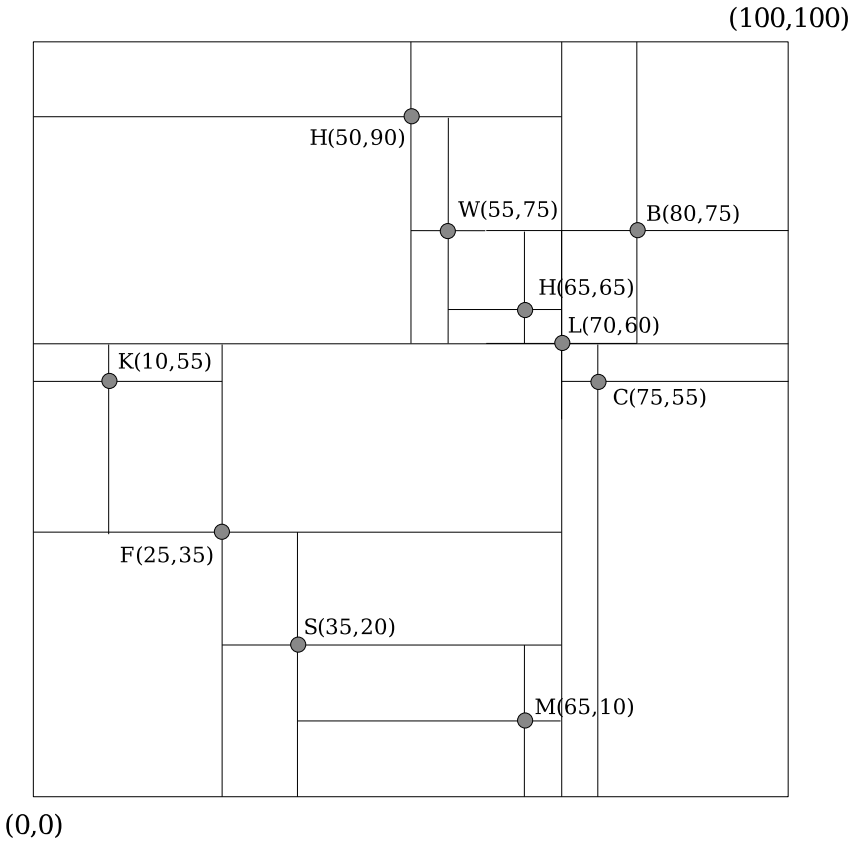


Abbildung 7.11: Quadtree nach dem Löschen von Erfurt (E)

7.5 Bereichssuche

Bei Bereichsanfragen, werden alle Knoten innerhalb eines spezifizierten Bereiches im Point Quadtree gesucht. Ein Kreis mit einem bestimmten Radius wäre ein gutes Beispiel für einen Bereich. Es sind aber auch andere Bereiche wie Polygone denkbar.

Der Point Quadtree ist besonders für die Bereichssuche geeignet. Die Effizienz der Suche beruht auf der Tatsache, dass nur sehr wenige Knoten des Baumes besucht werden müssen, um die Ergebnismenge zu bestimmen. Die Algorithmus wird an dem aus dem Abschnitt ber das Einfügen von Knoten schon bekannten Städte-Beispiel vorgeführt. Dabei sollen alle Städte angegeben werden, die innerhalb des Kreises um den Punkt (25,30) mit dem Radius 20 liegen. Die Suche beginnt bei der Wurzel des Point Quadtree (Erfurt). Sofort ist ersichtlich, dass in den Quadranten NW, NE und SE nicht gesucht werden muss, da die Grenzen des Bereiches diese Quadranten nicht schneiden. Die Suche beschränkt sich also auf den Quadranten SW mit der Wurzel Frankfurt. Diese Stadt ist innerhalb des Bereiches und wird der Ergebnismenge hinzugefügt. Die Suche wird auf allen vier Quadranten von Frankfurt fortgesetzt. Die Suche endet in den Quadranten NW, NE und SW, da keine Kindknoten vorhanden sind. Im Quadranten SE ist nur noch der Knoten Stuttgart vorhanden. Dieser wird der Ergebnismenge hinzugefügt und die Suche endet. Das Ergebnis der Suche sind also die beiden Städte Frankfurt und Stuttgart. Das gerade vorgestellte Beispiel ist in Abbildung 7.12 zu sehen.

Im Beispiel war nur die Bestimmung der zu durchsuchenden Quadranten problematisch. Alle anderen Schritte sind schon aus den vorangegangen Abschnitten bekannt. Die Bestimmung der Quadranten ist für einen allgemeinen Bereich recht kompliziert. Eine Vereinfachung ergibt sich, wenn der komplexe Bereich näherungsweise in einfache Bereiche untergliedert wird. Beispielhaft sei wieder der Kreis verwendet.

Abbildung 7.13 beschreibt, wie die Quadranten bestimmt werden. Es ist der kreisförmige Bereich mit dem Mittelpunkt A und dem Radius r gegeben. Je nach dem, in welchem durchnummerierten Areal die Koordinaten des gerade vom Suchalgorithmus besuchten Knotens liegen, sind die zu durchsuchenden Quadranten vorgegeben. Liegt beispielsweise der Knoten in dem Areal mit der Ziffer 2, so braucht nur in Quadranten SW und SE weitergesucht werden. Ist der Knoten im Areal 9, so wird nur in NE, SW und SE gesucht.

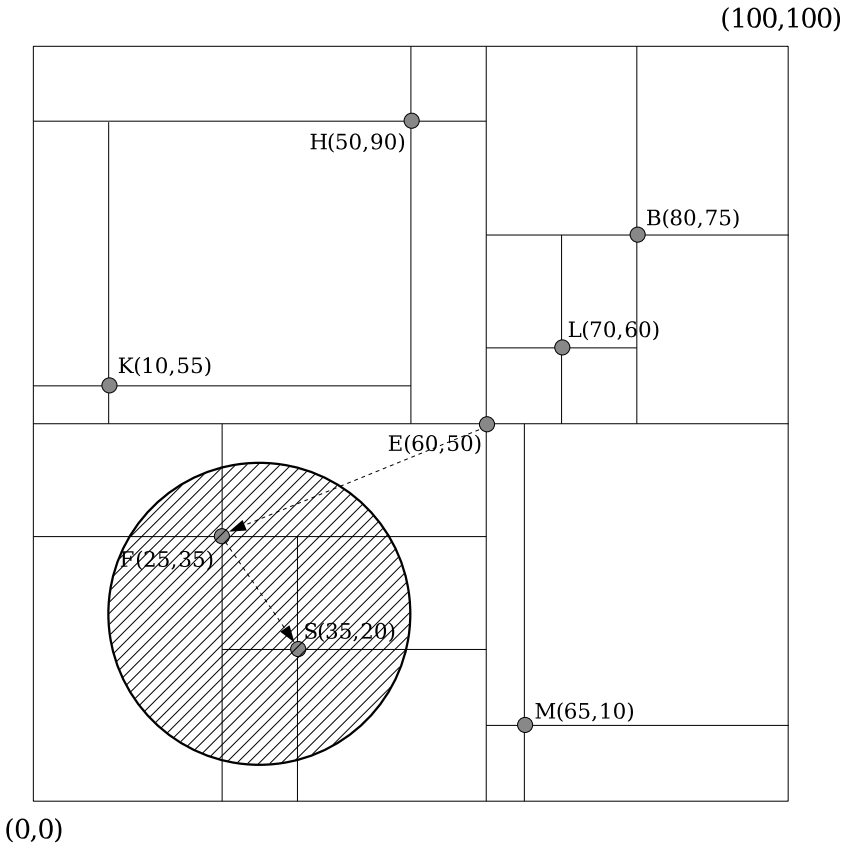
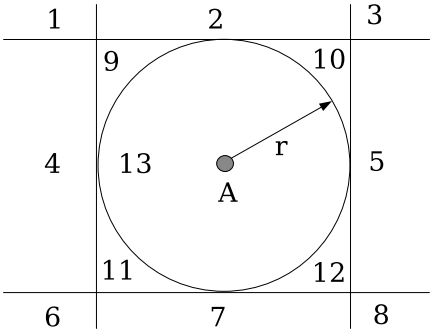


Abbildung 7.12: Bereichssuchebeispiel



Knoten	Weiter	Knoten	Weiter
1	SE	10	NW, SW, SE
2	SW, SE	11	NW, NE, SE
3	SW	12	NW, NE, SW
4	NE, SE	13	NW, NE, SW, SE
5	NW, SW		
6	NE		
7	NW, NE		
8	NW		
9	NE, SW, SE		

Abbildung 7.13: Weitersuche in den angegebenen Subquadranten, wenn die Suche bei einen Baumknoten ankommt, der in einen der Bereiche 1..13 um das kreisförmige Suchgebiet mit dem Mittelpunkt A liegt.

Ist der Bereich anders geformt, müssen die Quandranten mit einer anderen Methode bestimmt werden. Das Verfahren bleibt aber gleich. Ist beispielsweise der Bereich rechteckig, fallen nur die Areale aus Abbildung 7.13 mit den Nummern neun, zehn, elf und zwölf weg.

Die Komplexität im *worst case* der Bereichsanfrage für einen 2- dimensionale Point Quadtree ist $O(2 * N^{1/2})$. Dieses Resultat kann auf einen k- dimensionalen Point Quadtree erweitert werden. Hier ergibt sich für die *worst case* Komplexität $O(K * N^{1-1/k})$.

Literaturverzeichnis

[Fink74] - B. A. Finkel and J. L. Bentley, Quad trees: a data structure for retrieve on composite keys, Acta Informatica 4, 1974

[Samet80] H. Samet. Region Representation: Quadtrees From Boundary Codes. Communications of the ACM, 1980.

[Samet89] H.Samet. The design and analysis for spatial data structures. Addison-Wesley Verlag, 1989