

**Polsko-Japońska Wyższa Szkoła Technik
Komputerowych**

PRACA MAGISTERSKA

Nr

Indeksy Przestrzenne

Przegląd strategii, struktur i algorytmów

Autor: Piotr Tamowicz
Numer indeksu: s2669/M
Promotor: Prof. dr hab. Kazimierz Subieta
Specjalność: Inżynieria Oprogramowania i Baz Danych
Katedra: Inżynierii Oprogramowania

Podpis promotora pracy

Podpis kierownika katedry

.....

.....

Streszczenie

Niniejsza praca jest próbą zgromadzenia możliwie kompletnych i rzetelnych informacji na temat technik stosowanych w procesach indeksowania danych przestrzennych. Forma, w jakiej zostały przedstawione, z założenia miała być przystępna dla studentów chcących w przyszłości podjąć temat badawczy z tej dziedziny stanowiąc wprowadzenie do szerokiego zagadnienia, jakim są indeksy przestrzenne. Część pierwsza zawiera opis wypracowanych przez lata koncepcji reprezentacji oraz porządkowania zbiorów obiektów przestrzennych umożliwiającich wykonywanie na nich zapytań dotyczących wzajemnego położenia obszarów. Zawiera również przegląd funkcjonalności udostępnianych w tym zakresie przez popularne systemy baz danych. W części drugiej opisuję własny sposób analizy wybranej strategii, algorytmu jej działania oraz struktur niezbędnych do zaimplementowania w języku Java opartego o nią indeksu dla obiektów geometrycznych.

Słowa kluczowe

indeks przestrzenny, indeks dynamiczny, indeks statyczny, zapytanie zakresowe, zapytanie o najbliższych sąsiadów, R-drzewo, X-frzewo, drzewa czwórkowe, drzewa ósemkowe,

Spis treści

1.	Wstęp.....	1
1.1.	Motywacja	3
1.2.	Cel Pracy	3
2.	Analiza zagadnienia.....	4
2.1.	Zdefiniowanie pojęć	4
2.2.	Podstawowe strategie	6
3.	Przegląd indeksów przestrzennych.....	8
3.1.	R-Drzewa	8
3.1.1.	Struktura.....	8
3.1.2.	Wstawianie	10
3.1.3.	Usuwanie.....	11
3.1.4.	Aktualizacja	12
3.1.5.	Wyszukiwanie.....	12
3.1.6.	Problem przepelnienia węzła.....	12
3.1.7.	Podział naiwny.....	14
3.1.8.	Podział o liniowej złożoności	14
3.1.9.	Podział złożoności kwadratowej	15
3.2.	X-drzewa	16
3.2.1.	Struktura.....	17
3.2.2.	Algorytmy	18
3.3.	R*-Drzewa.....	19
3.4.	Drzewa czwórkowe	22
3.4.1.	Struktura.....	22
3.4.2.	Zbalansowane drzewa punktowe	23
4.	Realizacja indeksów przestrzennych w komercyjnych bazach danych	24
4.1.	SQL Server 2008	24
4.1.1.	Typy danych	24
4.1.2.	Struktura.....	25
4.1.3.	Operacje.....	26
4.2.	Oracle Locator	28
4.3.	Podsumowanie	30

5. Realizacja Indeksu Przestrzennego	31
5.1. Analiza	32
5.1.1. Wymagania	33
5.1.2. Analiza statyczna.....	34
5.1.3. Analiza dynamiczna.....	35
5.2. Projekt.....	36
5.2.1. API.....	38
5.2.2. Struktury indeksu.....	39
5.3. Implementacja.....	40
5.3.1. Wszystko jest obiektem.....	40
5.3.2. Dobór typów	41
5.3.3. Wyjątki.....	41
5.3.4. Rekurencja	42
5.3.5. Interfejsy.....	43
5.3.6. Funkcjonalność.....	44
5.3.7. Klasy testowe	46
6. Podsumowanie	51
6.1. Wnioski	51
6.2. Perspektywy.....	51
7. Bibliografia.....	52

1. Wstęp

Spółeczeństwo, w którym żyjemy określane jest często, jako „spółeczeństwo informacyjne”. Każdego dnia, w praktycznie w każdym aspekcie życia, zasypywani jesteśmy milionem informacji płynących z najróżniejszych źródeł. Począwszy od sygnalizacji świetlnej w drodze do pracy, prognozy pogody, porannej porcji wiadomości, danych płynących z gospodarki, przez stosy faktur i rachunków a co za tym idzie stanu naszego konta bankowego, reklamę czy zwykłą wizytę w sklepie, kończąc na wzajemnej nieformalnej komunikacji. Jesteśmy uzależnieni od informacji, stała się ona najcenniejszym towarem, źródłem rozrywki, sposobem zaspokojenia naszej naturalnej ciekawości, a nade wszystko kołem zamachowym postępu cywilizacyjnego.

Zrodziło to potrzebę budowania zaawansowanych systemów umożliwiających odkrywanie, przechowywanie, przetwarzanie, przekazywanie i wykorzystywanie informacji. Powstały więc bazy danych, które ewoluowały do postaci hurtowni danych, systemy eksploracji danych oraz oparte o nie narzędzia do zarządzania treścią. Aby sprostać stawianym przed nimi wyzwaniom niezbędne okazało się wypracowanie skutecznych metod wyszukiwania w wewnętrznych strukturach reprezentujących dane. Bowiem informacja, do której nie jesteśmy w stanie dotrzeć, a co równie ważne dotrzeć szybko, staje się w dzisiejszych czasach bezużyteczna (choć łatwość wytwarzania, przekazywania i dostępu spowodowała również pojawienie się ogromnej ilości bezużytecznej informacji).

Jest to niewątpliwie obraz współczesnej rzeczywistości nie należy jednak zapominać, iż zanim dzięki rewolucji technologicznej wywołanej obniżeniem kosztów zaawansowanych technologii, tego typu rozwiązania trafiły do masowego odbiorcy (osób prywatnych jak również małych i średnich przedsiębiorstw) już w latach siedemdziesiątych ich prototypy stosowane były w projektach rządowych, a także badaniach nad technologiami, z których dziś korzysta praktycznie każdy z nas. Dysponując ograniczoną mocą obliczeniową, do realizacji skomplikowanych zadań, wiele uwagi poświęcano optymalizacji procesów oraz sposobów dostępu do danych. Dziś, gdy powszechnie dostępne urządzenia dysponują wielokrotnie wyższą wydajnością konieczność korzystania z doświadczeń tamtych lat dyktuje objętość danych, z którą na co dzień mamy do czynienia oraz liczba użytkowników z nich korzystających.

Jednym z najczęściej stosowanych oraz bardziej znaczących mechanizmów, pozwalających zwiększyć wydajność systemów operujących na dużych zbiorach danych, są indeksy. Redundantne struktury pozwalające w szybki sposób dotrzeć do poszukiwanych danych. W największym uproszczeniu jest to zbiór par:

< klucz , wartość >

gdzie *klucz* jest atrybutem (kolumną lub polem) przeszukiwanych danych, natomiast *wartość* jest referencją lub fizycznym adresem danych (rekordu lub obiektu). Zbiór ten jest zorganizowany w taki sposób (często diametralnie różniący się od organizacji danych w źródle, z którego pochodzą wpisy), aby ograniczyć ilość koniecznych operacji, które należy wykonać chcąc dotrzeć do danej wartości klucza. Struktura indeksu często przyjmuje formę drzewa lub jemu podobną równie często wprowadzając dodatkowe, wyliczane na podstawie klucza (np. przy pomocy funkcji mieszającej), atrybuty ułatwiające poruszanie się po tej strukturze

Wyróżniamy dwa podstawowe rodzaje tych struktur:

- ❖ Indeksy przezroczyste – jeżeli to możliwe optymalizator zapytań automatycznie wybierze i skorzysta z obecnego w bazie danych indeksu w celu skrócenia czasu wykonania zapytania. Podstawowy podział tej grupy indeksów to:
 - ◆ indeksy gęste – pomocne w odpowiedzi na zapytania zawierające warunek równości. Mogą zwracać pojedynczą wartość (w większości relacyjnych systemów bazodanowych ten typ indeksu automatycznie zakładany jest na klucz główny tabeli) w takim przypadku mówimy o indeksie głównym lub wiele wartości, kiedy mówimy o indeksie wtórnym.
 - ◆ indeksy rzadkie – pomocne w odpowiedzi na zapytania o dane, których wartości kluczowe pochodzą z pewnego przedziału. Niektóre indeksy gęste mogą w pewnym stopniu realizować funkcje indeksu rzadkiego.
- ❖ Indeksy nieprzezroczyste – wymagają od użytkownika bezpośredniego wywołania funkcji realizujących różne rodzaje wyszukiwań. Do tej kategorii należą indeksy przestrzenne będące tematem niniejszego opracowania.

1.1. Motywacja

Głównym powodem podjęcia tego typu pracy była chęć zapoznania się ze strategiami realizującymi przestrzenne porządkowanie obiektów geometrycznych ze względu na zainteresowanie algorytmami i strukturami danych, które zawsze były moją mocną stroną. Temat wydał się tym bardziej ciekawy, kiedy zupełnie niedawno uświadomiłem sobie, za jak dużą część funkcjonalności aplikacji, z których na co dzień korzystam odpowiedzialne są właśnie indeksy. Zakres oraz skala ich zastosowań przerosły moje oczekiwania. Fakt, iż jak się okazuje, idea nie jest nowa. Rozwiązań wielu problemów, które w tym temacie się pojawiają jest jeszcze więcej, co najczęściej oznacza, że żadne z nich nie jest doskonałe lub znajduje zastosowanie jedynie w pewnych, ściśle określonych przypadkach oraz to, że w trakcie długoletniej nauki nigdy nie zetknąłem się z tym zagadnieniem pozwalał sądzić, iż przegląd tego typu rozwiązań może zainteresować studentów i dać motywację do realizacji projektów z dziedziny przeważania i prezentacji informacji przestrzennych.

1.2. Cel Pracy

Poznanie i zrozumienie operacji wykonywanych na strukturach indeksów przestrzennych, któremu poświęcona jest pierwsza część pracy powinno ułatwić analizę wybranego przypadku pod kątem implementacji jego funkcjonalności w obiektowym języku programowania. Z doświadczenia wiem, że pogoń za wydajnością algorytmów, choć w wielu przypadkach uzasadniona, często prowadzi do rezygnacji a nawet celowego omijania konstrukcji i założeń będących sednem obiektowego podejścia wytwarzania oprogramowania. Zdaję sobie sprawę, że prawdopodobnie najbardziej odpowiednim do takich celów byłby język C, natomiast powszechność i wygoda programowania w języku Java (która w dużej mierze wynika z odejścia od obowiązku niskopoziomowego zarządzania pamięcią przez programistę oraz wprowadzenia konstrukcji których omijanie może kusić przy implementacji algorytmów mających na celu zwiększanie wydajności) skłoniła mnie do podjęcia próby zaprojektowania oraz implementacji systemu realizującego indeksowanie obiektów przestrzennych w zgodzie z paradygmatem programowania obiektowego.

2. Analiza zagadnienia

*" There is geometry in the humming of the strings,
there is music in the spacing of the spheres "*

Pitagoras

2.1. Zdefiniowanie pojęć

Pomimo wyłożonych prac niektóre zagadnienia z dziedziny reprezentacji przestrzennej do dziś nie uzyskały satysfakcjonująco klarownych definicji, próby definiowania niektórych z nich zostały porzucone, tak jak na przykład pojęcie punktu zostało uznane za podstawowe i niedefiniowalne. Z kolei inne doczekały się wielu niekiedy sprzecznych specyfikacji, zależnych od przyjętych założeń.

Przykładem może być relacja posiadania części wspólnej, która często pojawia się w kontekście indeksów przestrzennych, określana niejednokrotnie w języku angielskim zamiennie mianem "overlapping" bądź "intersecting", co jest prawdą w przypadku wielokątów leżących na tej samej płaszczyźnie dwu wymiarowej, natomiast właściwie we wszystkich pozostałych przypadkach jest fałszem. Niejasne jest również czy i kiedy określenia "overlapping" możemy użyć dla brył znajdujących się w przestrzeni trójwymiarowej.

Wiedząc jak polegać na dobrze określonych pojęciach i założeniach może ułatwić przedstawienie oraz zrozumienie danego tematu postaram się w tym rozdziale przytoczyć kilka najczęściej pojawiających się pojęć dotyczących zagadnienia indeksów przestrzennych, na których będę bazował w dalszej części pracy.

Pojęcie, którym często się posługuje i uważam za konieczne wyjaśnienie, co przez nie rozumiem, choć użyte już przeze mnie kilka razy wcześniej zakładając jego intuicyjne znaczenie, to porządkowanie przestrzeni w kontekście zastosowania w indeksach. Choć nie spotkałem się z jego formalną definicją, stosuje je na zasadzie analogii do wprowadzania porządku liniowego. Oznacza taką organizację zbioru obiektów przestrzennych by możliwe było odkrywanie obiektów, które z danym obiektem wiąże pewna relacja topologiczna lub geometryczna. Przedmiotem organizacji może być również sama przestrzeń a nie zbiór obiektów w niej się znajdujących. Proces ten jest sednem indeksowania jednak, kiedy chcę zaznaczyć, iż dana struktura lub algorytm stanowiący część strategii indeksowania bezpośrednio wpływa na ten aspekt, odnoszę się do porządkowania przestrzeni.

Przestrzeń – oznacza układ, w którym reprezentujemy dane, istnieje wiele modeli przestrzeni, opracowanie zakłada korzystanie z przestrzeni oraz geometrii euklidesowej oraz kartezjańskiego układu współrzędnych o początku w punkcie 0 stanowiącym punkt odniesienia dla wszystkich operacji.

Wymiar – określany, jako maksymalna liczba wzajemnie prostopadłych prostych mogących przecinać się w danym punkcie przestrzeni, co trudne do wyobrażenia, ale zgodne z założeniami przestrzeni euklidesowej.

Bryła brzegowa – aproksymacja zakładająca równoległe położenie granic względem osi przestrzeni, w której jest reprezentowana.

Na uporządkowanej w ten sposób strukturze powinno być możliwe wykonanie zapytań, które można podzielić na dwie kategorie:

Zapytania punktowe – choć nie będę rozwijał tematu tego typu zapytań, wato zaznaczyć iż tego typu zapytania mają zastosowanie zarówno dla danych punktowych obecnych w indeksach oraz bardziej złożonych danych geometrycznych. Odpowiedzią może być lista k najbliższych sąsiadów gdzie liczność k określona jest bezpośrednio przez ich pożądaną ilość lub maksymalną odległość, w której powinien znajdować się obiekt aby uznać go za najbliższego sąsiada bądź obiektów zawierających dany punkt.

Zapytania zakresowe – stanowią główny przedmiot zainteresowań tego opracowania. Mogą być zadawane zarówno strukturom zawierającym dane punktowe jak i geometryczne. Testują relacje regionu (dowolnego obiektu geometrycznego) będącego argumentem zapytania z obiektami znajdującymi się w indeksie. Najczęściej testowane relacje to zawieranie obiektu (lub symetrycznie bycie zawartym) i posiadanie z nim części wspólnej (lub przecięcia).

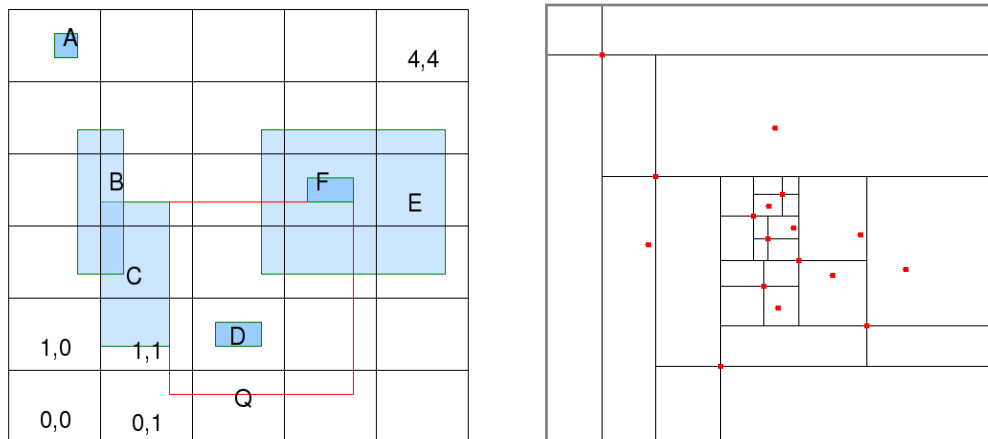
Inne zapytania takie jak określenie odległość między obiektami, porównanie czy ich objętości są jednakowe, czy inne zapytania dwuargumentowe nie stanowią części problemu porządkowania przestrzeni i jako takie należy rozważać w kategoriach czystej geometrii niż w pracy poświęconej indeksom przestrzennym.

Zakładam, że podstawowe pojęcia z zakresu klasycznego indeksowania, podstawowych struktur danych takich jak drzewa, listy czy stosy nie są czytelnikowi obce i nie muszę ich przedstawiać. Nie objaśniam także algorytmów przemierzania drzew oraz podstawowych operacji geometrycznych, których znajomość jest niezbędna aczkolwiek powszechna wśród studentów, przez co nie sądzę, aby brak umieszczenia ich opisu sprawiła większe problemy ze zrozumieniem przedstawianych dalej operacji.

2.2. Podstawowe strategie

Pierwszy z ugruntowanych paradygmatów porządkowania przestrzeni zakłada podział przestrzeni na rozłączne regiony zawierające w zależności od przyjętych założeń jeden lub więcej obiektów. Wariant przewidujący równomierny podział, według założonego kształtu i rozmiaru podprzestrzeni nie gwarantuje, że każdy będzie identyfikowany przez dokładnie jeden region. Częstość zabiegami jest tworzenie kilku warstw siatek o różnej gradacji zwiększając tym samym precyzję lokalizacji obiektów. Podstawową zaletą jest uniezależnienie złożoności operacji od rozkładu danych.

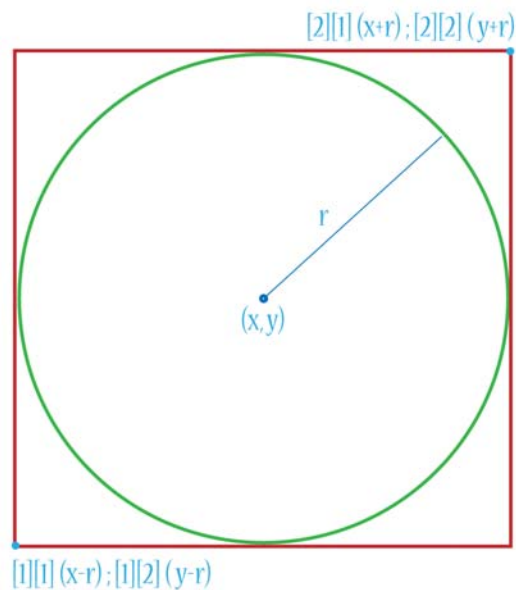
Podejście, w którym osie podziału dyktowane są położeniem obiektów dążą do jednoznacznej identyfikacji obiektu przez pojedynczy region. Jest to podejście bliskie ludzkiej percepcji, oraz tendencji do dzielenia większej całości na mniejsze fragmenty, aby w ten sposób zrozumieć ich sumę. Największą niedogodnością obu wariantów jest konieczność zadeklarowania ograniczonej przestrzeni, którą zamierzamy indeksować.



i.01 a) równomierny podział przestrzeni b) podział determinowany przez dane

W obu przypadkach najczęstszą reprezentacją dokonanych w ten sposób podziałów są B-drzewa. Jednak istnieją również rozwiązania oparte o listy a także wykorzystujące w tym celu funkcje hashujące.

Drugą grupą indeksów przestrzennych są rozwiązania zakładające tworzenie podlegających im struktur w oparciu o wzajemne relacje wstawianych obiektów z obiektami aktualnie obecnymi w indeksie, nie zaś o ich relację do niezdefiniowanej przestrzeni indeksu, która w przypadku tego podejścia jest nieograniczona. W celu uniformizacji operacji, które należy wykonać by stwierdzić położenie nowych obiektów w strukturze, stosuje się ich aproksymacje. Są to najczęściej, choć nie zawsze, prostokąty lub ich odpowiedniki z wyższych wymiarów.



i.02 przykład aproksymacji z płaszczyzny dwu wymiarowej - kwadrat opisany na okręgu.

Grupę tę, najszerszej reprezentuje rodzina R-drzew, którą zamierzam opisać możliwie dokładnie, jako że jeden z jej reprezentantów jest silnym kandydatem do implementacji, będącej drugą częścią tej pracy. Już po wstępnym zapoznaniu się z założeniami tego podejścia dostrzegłem wiele ciekawych rozwiązań oraz logiczną dobrze określoną, hierarchiczną strukturę, natomiast zbiór wykonywanych na niej operacji wydają się być otwarty i wymienny. Stopień ich złożoności bazując wciąż na tych samych strukturach wzrasta wraz z wprowadzeniem dodatkowych założeń mających na celu zwiększyć efektywność indeksu.

Realizacje obydwu opisanych strategii występują w postaciach dynamicznych i statycznych. Wybór jednej z nich motywowany jest przede wszystkim zmiennością zbioru danych, który zamierzamy porządkować. Postać dynamiczna wymaga najczęściej mechanizmów pozwalających utrzymać optymalną strukturę jednak pozwala na wstawianie oraz aktualizację obiektów w momencie, gdy indeks został już utworzony. Wersja statyczna, zwana często upakowaną, charakteryzuje się lepszym wykorzystaniem przestrzeni, oraz bardziej odpowiedniej dystrybucji obiektów w strukturze, co wpływa na poprawę czasu wyszukiwania. Kompromis w tym przypadku polega na konieczności budowania drzewa od podstaw w przypadku aktualizacji lub zaistnienia potrzeby dodania nowych obiektów. Jeżeli strukturą, na której operuje indeks jest drzewo wybór wariantu implikuje to strategię jego budowy. Bottom-Up w przypadku wariantu dynamicznego, Top-Down – w przypadku struktur statycznych.

3. Przegląd indeksów przestrzennych

Od momentu zaistnienia potrzeby porządkowania przestrzeni do celów przetwarzania reprezentacji obiektów pochodzących z dziedziny wielo wymiarowej, pojawiło się wiele prac przedstawiających propozycje algorytmów oraz ich modyfikacji mających na celu optymalizację tego procesu. Stosując nowe struktury i strategie starano się wyeliminować problemy niskiego wykorzystania przestrzeni adresowej, złożoności obliczeniowej operacji wykonywanych na proponowanych strukturach oraz pamięci niezbędnej do ich przechowywania. W tym rozdziale staram się przedstawić idee stojące za niektórymi z wypracowanych metod, ich mocne i słabe strony a także zakres zastosowań, w którym dana koncepcja sprawdza się najlepiej.

3.1. R-Drzewa

W 1984 roku strukturę R-Drzewa opisał Antomn Guttman, jako wynik badań przeprowadzonych w ramach grantu departamentu sił powietrznych Stanów Zjednoczonych [3]. Założeniem było stworzenie dynamicznej struktury bazującej na pamięci dyskowej (plikach stron). Wyniki badań pokazały, że zaproponowane podejście jest odpowiednie dla różnych charakterystyk danych wejściowych, stało się więc przedmiotem wielu kolejnych badań i od tamtego czasu doczekało się wielu modyfikacji, które opiszę w kolejnych rozdziałach opracowania.

3.1.1. Struktura

W oryginalnej wersji Guttmana jest to drzewo o zbalansowanej wysokości, w którym wszystkie liście znajdują się na jednym poziomie, i jedynie liście zawierają wskaźniki do indeksowanych danych. W wersji bazującej na pamięci dyskowej węzły pośrednie reprezentują natomiast pliki stron, w związku z czym, algorytm wyszukiwania został pomyślany tak by ograniczyć konieczność odwiedzania dużej liczby węzłów. Struktura ta jest w pełni dynamiczna, pozwala na dokonywanie przeminie operacji wstawiania, usuwania i wyszukiwania bez konieczności reorganizacji struktury ze strony użytkownika – w tym znaczeniu struktura jest samoorganizująca. Budowa rozpoczyna się od węzła liścia (który w tym momencie jest także korzeniem).

Jako wartość kluczową indeksu przewidziano minimalną bryłę brzegową będącą aproksymacją przestrzennego obiektu, wartością indeksowaną jest identyfikator

pochodzący na przykład z bazy danych, na podstawie którego w wydajny sposób użytkownik jest w stanie odczytać informacje powiązane z obiektem. Wpis indeksu w węźle będącym liściem ma zatem postać pary:

$$(I, \text{tuple-identifier})$$

gdzie *tuple-identifier* jest wspomnianą referencją, natomiast *I* stanowi *n*-wymiarową bryłę brzegową zdefiniowaną przez koordynaty $[a, b]$ określające początkowy i końcowy punkt bryły w danym wymiarze:

$$I = (I_0, I_1, \dots, I_{n-1}), I_i = [a, b]$$

Wpisy węzłów pośrednich, zawierają wskaźnik na węzeł potomny, oraz bryłę brzegową zawierającą przestrzennie wszystkie jego wpisy:

$$(I, \text{child-pointer})$$

Atrybuty *M* i $m \leq M/2$ określają maksymalną oraz minimalną liczbę wpisów, jaka może znajdować się w węźle. Na ich podstawie definiowane są własności R-drzewa:

- każdy liść może zawierać nie mniej wpisów niż *m* i nie więcej niż *M*, wyjątkiem jest sytuacja gdy liściem jest korzeń który może zawierać mniej niż *m* wpisów
- każdy węzeł pośredni zawiera nie mniej wpisów niż *m* i nie więcej niż *M*, za wyjątkiem korzenia, który może zawierać mniej niż *m* wpisów
- korzeń zawiera nie mniej niż dwa wpisy z wyjątkiem sytuacji, gdy jest liściem

Z powyższych założeń wynika, że jedynym węzłem mogącym zawierać mniej niż *m* elementów jest korzeń mogący zawierać jeden wpis, gdy jest to węzeł będący liściem oraz pomiędzy dwa a *M* wpisów, w przypadku, gdy jest węzłem pośrednim.

Wysokość drzewa zawierającego *N* wpisów jest ograniczona z góry przez:

$$\lceil \log_m N \rceil - 1$$

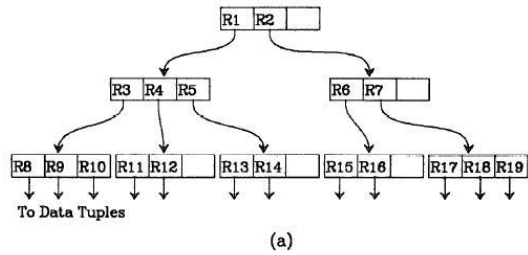
Maksymalna liczba węzłów określona jest wzorem:

$$\lceil N/m \rceil + \lceil N/m^2 \rceil + 1$$

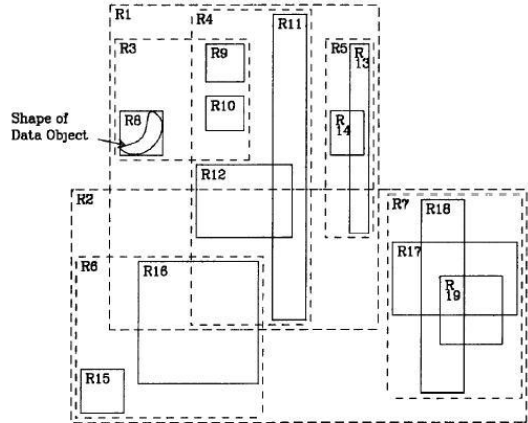
w pesymistycznym wariancie wykorzystanie zadeklarowanej pamięci wynosi

$$m/M$$

Na tak zdefiniowanej strukturze, przedstawionej na ilustracji i03, Guttman proponuje algorytmy wstawiania, usuwania, aktualizacji, wyszukiwania wpisów oraz podziału węzła w przypadku jego przepełnienia.



(a)



(b)

i.03 (a) struktura R-drzewa zawierająca dane przedstawione w (b)
 (b) aproksymację obiektów przestrzennych zaindeksowane w drzewie (a)

3.1.2. Wstawianie

Operacja wstawiania do indeksu nowych wpisów przypomina wstawianie danych do B-drzewa. Nowe wpisy dodawane są do liści. W przypadku przepełnienia liści ulegają podziałowi, zostaje utworzony nowy liść a wpisy zawarte w przepełnionym węźle (łącznie z wpisem powodującym przepełnienie) podlegają dystrybucji pomiędzy obecny i nowy węzeł. Może to prowadzić do przepełnienia węzła wyższego poziomu, który staje się rodzicem nowo utworzonego węzła liścia. W takim przypadku on również ulega podziałowi, na tej zasadzie podział liścia propagowany jest aż do korzenia.

Kompletny algorytm przedstawia się następująco:

1. Wybierz liść, do którego należy wstawić nowy wpis e . Zaczynij od korzenia. Jeżeli węzeł jest liściem zapisz go jako L i przejdź do następnego punktu. Jeżeli nie, przejrzyj wszystkie wpisy w tym węźle i wybierz ten, którego I zostanie powiększona o najmniejszą wartość, jeżeli doda się do niej I reprezentujące e . Przejdź do węzła będącego potomkiem wybranego wpisu. Jeżeli węzeł ten jest liściem zapisz go jako L , jeżeli nie powtórz procedurę.
2. Wstaw wpis do L , jeżeli liść zawiera teraz więcej niż M utwórz nowy węzeł LL . Rozdziel wszystkie wpisy L (także nowo wstawiony) pomiędzy L i LL

-
3. Jeżeli L jest korzeniem przejdź do kolejnego punktu. Jeżeli nie wybierz węzeł P , który jest rodzicem L . Dokonaj aktualizacji I należącego do P tak by uwzględniła również nowo wstawiony wpis, który może znajdować się w L lub LL . Jeżeli istnieje LL utwórz nowy wpis w P wskazujący na LL . Jeżeli powoduje to przekroczenie maksymalnej dozwolonej liczby wpisów w węźle dokonaj podziału jak w poprzednim punkcie. Wróć na początek tego punktu.
 4. Jeżeli podział nastąpił w korzeniu, utwórz nowy węzeł i przypisz mu węzły powstałe w wyniku podziału.

3.1.3. Usuwanie

Pożądanym skutkiem ubocznym operacji usuwania wpisów z drzewa jest jego reorganizacja, która pomaga zachować strukturę w kształcie często eliminującym potrzebę przemierzenia całej szerokości podczas wyszukiwania.

Przedstawiam zoptymalizowaną procedurę usunięcia wpisu e :

1. Aby zlokalizować liść L zawierający poszukiwany obiekt, przeszukuj wpisy węzłów rodziców sprawdzając czy ich I zawiera I określone przez e , dal ustalonego w ten sposób wpisu przejdź do wskazywanego przez niego węzeł i wykonaj powyższą procedurę aż osiągnięty zostanie poziom liścia.
2. Przeszukaj węzeł liścia porównując jego wpisy z e , jeżeli zostanie znalezione przejdź do kolejnego kroku, jeżeli nie, wykonaj procedurę z poprzedniego punktu dla kolejnego ustalonego w niej węzła. Wykonuj te kroki do momentu, gdy wpis e zostanie znaleziony.
3. Usuń wpis reprezentujący e z L . Jeżeli L nie jest korzeniem, a po usunięciu wpisu w L jest obecnych mniej niż m wpisów z węzła P będącego rodzicem L usuń wpis na niego wskazujący. Zaktualizuj I należący do P i dodaj L do listy usuniętych węzłów. Jeżeli w wyniku tej operacji P będzie zawierał mniej niż m wpisów i nie jest korzeniem postępuj z nim tak jak z L .
4. Jeżeli lista usuniętych węzłów zawiera węzły liści, zawarte w nich wpisy wstaw korzystając ze zwykłej operacji wstaw. Jeżeli na liście znajdują się węzły rodziców przy wstawianiu zatrzymaj algorytm wyszukiwania węzła docelowego na poziomie, z którego został usunięty węzeł zawierający obecnie wstawiane elementy.
5. Jeżeli korzeń ma tylko jednego potomka ustaw potomka jako korzeń.

3.1.4. Aktualizacja

Z przeprowadzonych badań wynika, że najlepszą strategią aktualizacji rekordu wpływającą na jego aproksymację, jest usunięcie go i ponowne wstawienie. Operacje te nie są bardziej kosztowne niż alternatywne metody reorganizacji hierarchii węzłów zawierających wpis, przynosząc jednocześnie pożądany efekt utrzymywania drzewa w optymalnej formie [4].

3.1.5. Wyszukiwanie

Algorytm wyszukiwania również przypomina ten znany z przeszukiwania R-drzewa, z tą różnicą, że w tym przypadku może być konieczne sprawdzenie więcej niż jednej gałęzi poddrzewa odwiedzonego w procesie odnajdywania wpisu. Dlatego też nie jesteśmy w stanie wyznaczyć złożoności czasowej takiego wyszukiwania. Fakt ten nie jest aż tak wielkim problemem, jeżeli weźmiemy pod uwagę, że w przypadku większości danych, algorytmy usuwania i aktualizacji oraz podziału węzłów jednocześnie reorganizują drzewo tak, że algorytm wyszukiwania jest w stanie pominąć nieadekwatne rejony indeksowanej przestrzeni, skupiając swoje działanie jedynie na danych znajdujących się w pobliżu poszukiwanego obszaru.

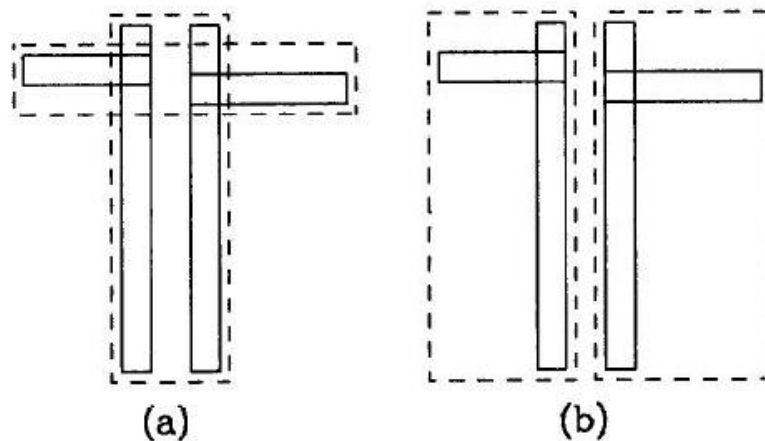
Algorytm wyrażony językiem naturalnym:

1. Aby odnaleźć wpis, który reprezentuje obiekt przestrzenny mający część wspólną z szukanym obszarem S przeszukuj wpisy węzłów niebędących liśćmi porównując ich I na okoliczność przecięcia z S . Odwiedź kolejno ich węzły potomne powtarzając powyższą procedurę aż osiągnięty zostanie poziom liści.
2. Przeszukaj wpisy wybranych w poprzednim kroku węzłów liści, sprawdzając czy ich I przecina się z S . Dodaj wpisy spełniające ten warunek do stosu rezultatów.

3.1.6. Problem przepelnienia węzła

W celu dodania nowego wpisu do pełnego węzła, konieczne jest podzielenie zbioru zawierającego $M+1$ wpisów pomiędzy dwa węzły. Podział powinien nastąpić w taki sposób, aby na tyle na ile to możliwe wykluczyć konieczność odwiedzenia powstałych w ten sposób węzłów w trakcie jednej operacji wyszukiwania. Ponieważ decyzja, czy odwiedzić dany węzeł polega na sprawdzeniu czy I reprezentujące szukany wpis posiada część wspólną z I zawartym we wpisie wskazującym na węzeł, objętość brył brzegowych każdego z węzłów, a także sumy, po dokonaniu operacji podziału powinna być jak najmniejsza. Ilustracja i04 przedstawia przypadek dobrego i

złego podziału. Stanowi to realizację podejścia minimalizacji bryły brzegowej. To kryterium znajduje zastosowanie również przy wyborze węzła, podczas operacji wstawiania wpisu.



i.04 (a) dobry podział (b) zły podział

Inne strategie indeksowania w oparciu o rodzinę R-drzew, opisane w dalszej części tego rozdziału, przyjmują odmienne założenia podziału, między innymi zasadę rozłączności brył brzegowych reprezentujących węzły wynikowe. Jak widzimy podział o rozłącznych rejonach w kategoriach algorytmu Guttmana nie jest lepszym podziałem niż ten unikający przecięć brył brzegowych.

Ponieważ za kluczową operację w przypadku indeksu można przyjąć operację wyszukiwania, większość strategii dopuszcza podniesienie złożoności procesu budowy i aktualizacji drzewa, w celu poprawienia wydajności wyszukiwania. W swym artykule Guttman opisuje, jednak nie zaleca naiwnego procesu podziału węzła. Również w innych publikacjach wskazuje się, iż wykorzystanie algorytmów działających w czasie liniowym oraz kwadratowym może dać niewiele gorsze czasy wyszukiwania, przy znacznie niższym koszcie aktualizacji struktury drzewa. Istne różnice pojawiają się w przypadku bardzo dużych zbiorów danych (kilkanaście milionów), które nie wymagają częstej aktualizacji [4].

3.1.7. Podział naiwny

Najprostszym sposobem na przeprowadzenie podziału dającego najmniejszą objętość brył ograniczających wynikowe węzły jest zbudowanie zbioru wszystkich możliwych rozkładów wpisów w dwóch węzłach, a następnie wybranie najlepszego. Operacja taka cechuje się jednak wykładniczą złożonością obliczeniową (2^M), która w większości przypadków jest nie do zaakceptowania. Potwierdziły to testy dla węzłów o $M=50$, co jest optymalną wartością dla indeksów operujących na pamięci dyskowej.

3.1.8. Podział o liniowej złożoności

Czas wykonania tej wersji podziału jest liniowo zależny od M oraz ilości wymiarów indeksowanych danych. Polega na wyborze dwóch najbardziej oddalonych od siebie obiektów i użyciu ich, jako ziaren umieszczonych w osobnych węzłach, stanowiących punkt odniesienia przy podejmowaniu dalszych decyzji dotyczących dodawania kolejnych wpisów. Pozostałe wpisy, dodawane są kolejno do węzła, którego obecna bryła brzegowa ulegnie mniejszemu powiększeniu po dodaniu do niej wpisu.

Kompletna procedura przewiduje:

1. Wybranie ziaren, jako pierwszych elementów każdego węzła. W tym celu należy odnaleźć wpisy o najwyższej wartości dolnej aproksymacji oraz o najniższej wartości aproksymacji górnej. Znormalizować różnicę pomiędzy tymi wartościami przez podzielenie jej przez rozpiętość analizowanego zbioru. Powtórzyć operację dla każdego wymiaru. Jako ziarna wybrać wpisy, o największej separacji z pośród wśród wszystkich wymiarów.
2. Dodawanie pozostałych po wybraniu ziaren wpisów do węzła, którego bryła brzegowa zostanie powiększona o najmniejszą wartość, do momentu gdy nie pozostanie więcej elementów lub gdy któryś z węzłów będzie zawierał tak mało wpisów, że należało będzie dodać do niego wszystkie pozostałe do rozdzielenia wpisy, aby spełnić dolne ograniczenie zawartości węzła m .
3. W przypadku, gdy kryterium minimalizacji objętości bryły brzegowej nie rozstrzygnie, do którego węzła powinien zostać dodany wpis, należy się kierować mniejszą obecną objętością bryły a następnie ilością wpisów.

3.1.9. Podział złożoności kwadratowej

Tak jak poprzedni, algorytm nie gwarantuje znalezienia najmniejszych z możliwych grup podziału wpisów, jednak korzysta z bardziej wyrafinowanych metod wyboru ziaren oraz kolejności wstawiania wpisów. Czas wykonania rośnie w stopniu kwadratowym do M natomiast liniowo do liczby wymiarów indeksowanych obiektów.

Procedura przewiduje:

1. Obliczenie dla każdej pary wpisów nadmiarowej objętości, która powstanie w przypadku dodania danej pary do jednego węzła i wybranie jako ziaren pary o największej wartości tej objętości.
2. Obliczenie dla każdego pozostałego wpisu wartości, o jaką powiększy się bryła brzegowa każdego z dwóch węzłów, a następnie wybranie wpisu, który powiększa obecną bryłę węzła o najmniejszą wartość i dodanie go do tego węzła. W przypadku 'remisu' należy wybrać wpis, który optymalnie powiększa węzeł o najmniejszej objętości bryły brzegowej a jeżeli i to nie przyniesie rozstrzygnięcia, wpis odnoszący się do węzła zawierającego mniejszą ilość wpisów.

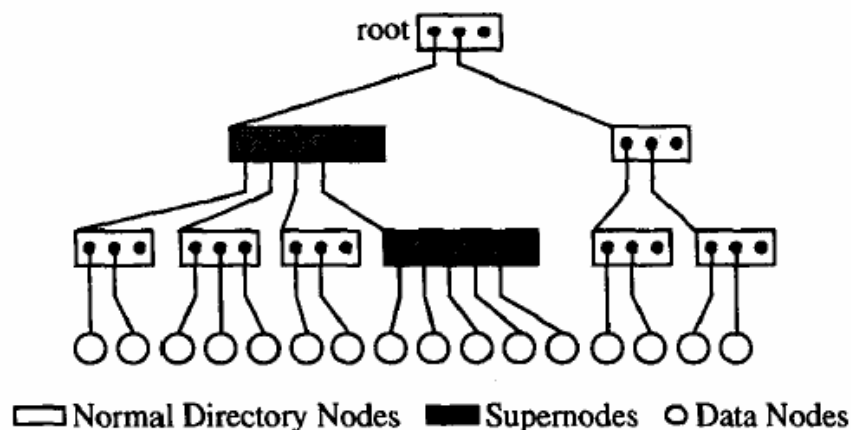
Również w tym wariancie podziału brany jest pod uwagę warunek minimalnej ilości wpisów w węźle, w przypadku, gdy pozostało do rozdzielania tak mało wpisów, że należy je dodać do któregoś z węzłów by zawierał on przynajmniej m wpisów. Ma to jednak znacznie mniej dotkliwe konsekwencje niż w przypadku podziału liniowego, gdyż wpisy o największym znaczeniu zostają przydzielone do odpowiednich węzłów jako pierwsze. Przydzielanie pozostałych ma mniejszy wpływ na strukturę wynikową, ze względu na mniejszą różnicę w powiększeniu bryły brzegowej jednego i drugiego węzła. Algorytm nie gwarantuje najlepszego z możliwych podziałów, natomiast unika istotnej wady poprzednika, bardzo silnej zależności od dystrybucji a nawet kolejności obecnych w indeksie danych.

3.2. X-drzewa

X-drzewa to struktury indeksów oparte na założeniach z rodziny R-drzew stworzone z myślą o porządkowaniu danych wielowymiarowych (o dziesiątkach a nawet setkach wymiarów) wykorzystywanych w takich dziedzinach jak multimedia czy biologia molekularna. Analizy wykazują, iż nawet takie indeksy jak R*-drzewa nie są wystarczająco optymalne do przetwarzania zbiorów danych o 5 lub więcej wymiarach [6]. Główną wadą indeksów bazujących na strukturach z rodziny R-drzew jest nakładanie się na siebie brył brzegowych węzłów, co nasila się wraz ze wzrostem liczności wymiarów danych.

Inne podejścia, nieopisywane w tym opracowaniu, próbują radzić sobie z tym problemem biorąc pod uwagę obserwację, iż rzeczywiste dane wielowymiarowe mają tendencję tworzenia skupisk a korelacja między nimi powoduje, że zajmują jedynie pewną ograniczoną podprzestrzeń z pośród wielu wymiarów. Algorytmy takie jak "szybka mapa" (Fastmap [7]) czy wielowymiarowe skalowanie dokonują transformacji obiektów do niższych wymiarów gdzie są indeksowane przy użyciu tradycyjnych struktur przestrzennych.

W celu uniknięcia tego problemu, X-drzewa zakładają odmienną organizację katalogu węzłów, w którym operacja podziału minimalizuje niepożądane zjawisko oraz wprowadza pojęcie super węzła. Struktura zakłada postać hierarchiczną jednak unika podziałów węzłów, których wynikiem byłoby powstanie dużych nakładających się obszarów. Węzły, które uniknęły podziału są rozszerzane ponad zwyczajną objętość dozwolonych wpisów, przez co otrzymują miano super węzłów. Minimalizacja nakładania się brył brzegowych odbywa się kosztem konieczności liniowego przeszukiwania super węzłów, które z czasem mogą stać się bardzo obszerne.



i.05 struktura X-drzewa

W tej mierze można postrzegać X-drzewa, jako hybrydowy katalog, łączący cechy hierarchicznej reprezentacji węzłów R-drzew z prostymi tablicami. Wynika to z ugruntowanego stwierdzenia, iż dla danych o niewielkiej liczbie wymiarów najlepszą reprezentacją katalogu jest struktura drzewiasta, dla wielowymiarowych, gdy stopień nakładania aproksymacji jest bardzo duży (nawet do 90%), co w praktyce skutkuje koniecznością przejścia niemalże całego drzewa, liniowa organizacja struktury ma przewagę w kwestii zajmowanej przez nią pamięci. Zadanie, które postawili przed sobą twórcy tego podejścia polegało na stworzeniu algorytmów, które automatycznie zarządzałyby formą reprezentacji tak, by w przypadku dużego nakładania wykorzystywana była forma liniowa, natomiast tam gdzie problem nakładania nie stanowi problemu forma hierarchiczna.

3.2.1. Struktura

Jak wynika z ilustracji i05, X-drzewo może zawierać trzy typy węzłów:

- ◆ liście – zawierają wskaźnik do rzeczywistych danych oraz ograniczającą je bryłę brzegową, dokładnie tak jak w R-drzewach
- ◆ węzły katalogowe – wskaźniki na bryły brzegowe węzłów podrzędnych własną bryłę brzegową oraz historię podziału węzła.
- ◆ super węzły katalogowe – charakteryzują się zmienną wielkością (będącą wielokrotnością wielkości zwykłego węzła katalogowego)

W przeciwieństwie do możliwości określenia większej pojemności węzłów dla całego R-drzewa, X-drzewa stosują powiększone węzły jedynie tam gdzie jest to korzystne dla zachowania niskiego nakładania brył brzegowych. Wraz ze wzrostem ilości wymiarów i nasilenia zjawiska nakładania ilość super węzłów oraz ich rozmiar wzrasta. Powoduje to, iż wysokość drzewa nie wzrasta proporcjonalnie do ilości wstawionych danych. Nadzwyczajne węzły tworzone są podczas operacji wstawiania elementu do indeksu, gdy niema innej możliwości na uniknięcie wystąpienia efektu nakładania. Należy zwrócić uwagę, iż wykorzystanie przestrzeni w super węźle jest przeciętnie wyższe w porównaniu ze zwykłym węzłem (gdzie przeciętnie wynosi ok. 70% dla równomiernie rozłożonych danych).

Można wyróżnić dwa wyjątkowe stany, w których może znajdować się struktura:

- 1) żaden z węzłów katalogowych nie jest super węzłem
- 2) korzeń jest jedynym super węzłem w katalogu drzewa

Pierwszy przypadek, w którym struktura przypomina R-drzewo może zaistnieć dla danych o niskiej ilości wymiarów lub niskiej korelacji i stopniu skupienia. Drugi zupełnie przeciwny, gdy struktura staje się płaską listą ze względu na dane wykazujące silne wzajemne zależności przestrzenne w bardzo wielu wymiarach, co w pesymistycznym przypadku wymagałoby przemierzania całego drzewa.

3.2.2. Algorytmy

Krytycznym dla stworzenia wydajnej struktury wydaje się być algorytm wstawiania wpisów do indeksu. Określenie lokalizacji odbywa się na zasadzie dopasowania aproksymacji wstawianego obiektu do brył brzegowych węzłów kolejnych poziomów, jeżeli nie powoduje to przepełnienia liścia, następuje aktualizacja adekwatnych brył wyższego poziomu. Jeżeli dochodzi do przepełnienia, wywołany jest algorytm podziału bazujący na geometrycznych oraz topologicznych cechach wpisów znajdujących się w dzielonym węźle. Jeżeli węzły powstałe w wyniku tej operacji cechuje duże nałożenie reprezentujących je brył brzegowych następuje próba dokonania podziału o minimalnym wzajemnym nałożeniu wykorzystująca w tym celu o historię podziału węzła. Jeżeli prowadzi to do niedopełnienia któregoś z węzłów operacja podziału nie zostaje dokonana, za to węzeł powiększany jest o przestrzeń dostępną w standardowym węźle (dotyczy to także super węzłów).

Operacje wyszukiwania, aktualizacji oraz usuwania wpisów niewiele różnią się od analogicznych operacji wykonywanych na R-drzewie. Jedyną znaczącą modyfikacją jest przypadek, gdy niedomiar następuje w super węźle. W takiej sytuacji zamiast usuwania węzeł jest zmniejszany o wielkość standardowego węzła.

Operacja podziału węzła o minimalnym nałożeniu wynikowych brył brzegowych stanowi największą różnicą w stosunku do poznanych już struktur drzewiastych. Jako tezę podstawową twórcy przyjęli fakt, iż dokonanie zbalansowanego podziału węzła nie powodującego nakładania jest jedynie możliwe, gdy wszystkie wpisy zawarte w węźle uległy już takiej operacji względem tego samego wymiaru. Do stwierdzenia tego faktu wykorzystuje się historię podziału reprezentowaną przez binarne drzewo podziału zawierające w liściach informacje o węzłach będących efektem podziału, w węzłach pośrednich numer wymiaru względem którego ta operacja nastąpiła. Jeżeli przynajmniej jeden z węzłów nie uległ wcześniej podziałowi istnieje szansa, że zawiera on pełen zakres koordynat, co uniemożliwia dokonanie podziału wolnego od nałożenia. Choć wraz ze wzrostem ilości wymiarów szansa takiego podziału maleje, w pewnym stopniu pozwala to ograniczyć tworzenie nowych super węzłów.

3.3. R*-Drzewa

Struktura zaproponowana przez Norberta Beckmanna w 1990 r. jest kolejnym wariantem z rodziny R-drzew [12] wprowadzającym optymalizację struktury oraz nowe strategie podziału węzła. Przedstawione aspekty tworzenia drzewa, są przez autora wymieniane, jako kluczowe dla podniesienia wydajności odpowiedzi na zapytania:

1. Minimalizacja powierzchni bryły brzegowej węzłów pośrednich.

Przestrzeń zawarta przez bryłę a nie zawarta przez bryły reprezentujące zawarte w węźle obiekty (tzw. martwa przestrzeń) powinna być jak najmniejsza. Pozwoli to na podjęcie decyzji o odwiedzeniu konkretnego węzła lub nie, na wyższym poziomie.

2. Nakładanie się brył brzegowych węzłów pośrednich powinno być ograniczane.

Ograniczy to ilość przebytych ścieżek w drodze do poszukiwanego obiektu.

3. Obwód bryły powinien być minimalny

Obwód rozumiany, jako suma długości krawędzi bryły. W dwóch wymiarach, zakładając daną powierzchnię, najmniejszy obwód z pośród czworokątów posiada kwadrat (w 3D - sześcián itd.), dlatego realizując ten postulat zamiast ograniczania powierzchni bryły spowoduje bardziej kwadratowy kształt brył węzłów katalogowych. Poprawi to według Beckmanna strukturę drzewa, jako że taki kształt podlega łatwiejszemu upakowaniu w węzłach nadrzędnych. Dlatego też tworzenie brył o jedynie niewielkich różnicach w długości boków na danym poziomie ograniczy objętość bryły brzegowej wyższego poziomu.

4. Optymalizacja wykorzystanej przestrzeni

Wyższy stopień wykorzystania przestrzeni dostępnej w węzłach, poprawia wydajność przetwarzania zapytań, szczególnie gdy w odpowiedzi znaleziona zostaje duża liczba obiektów. Dlatego też ma to większy wpływ na zapytania o większe regiony.

Minimalizacja objętości bryły brzegowej oraz nakładania się brył wymaga swobody w kwestii minimalnej ilości wpisów mogących znajdować się w węźle, co kluczi się z punktem czwartym postulującym o możliwie duże wykorzystanie przestrzeni adresowej dostępnej w węźle katalogowym. Postulując spełnienie dwóch pierwszych punktów, należy zmniejszyć ograniczenia dotyczące kształtu bryły, w wyniku czego nie będą one już dążyć do kwadratowego kształtu.

Spełniając założenie punktu pierwszego sprzyja realizacji założeń punktu drugiego, ponieważ często to "martwa przestrzeń" wpływa na nakładanie się brył. Traktując priorytetowo kwestię minimalizacji obwodu, musimy liczyć się z ograniczeniem wykorzystania przestrzeni w węźle katalogowym. Jednak łatwiejsze upakowanie brył o takim kształcie w węzłach nadrzędnych może przywrócić część poświęconej przestrzeni adresowej. Na zapytania o duże regiony, w znacznie większym stopniu wpływa wykorzystanie przestrzeni węzła niż założenia punktów 1-3.

W celu optymalizacji struktury indeksu podczas operacji wstawiania, wybór węzła, w którym powinien znaleźć się obiekt uzależniony jest od kombinacji parametrów powierzchni, obwodu oraz części wspólnej.

Obiekty wstawiane są do węzłów wskazujących na liście, których powiększona bryła brzegowa w najmniejszym stopniu powiększy powierzchnię nakładania z innym węzłem, następnie brana pod uwagę jest kryterium minimalnego powiększenia danej bryły brzegowej, na końcu aktualna powierzchnia bryły. Węzły katalogowe wyższych poziomów wybierane są na podstawie kalkulacji powiększenia ich bryły brzegowej w wyniku wstawienia do ich poddrzewa przetwarzanego obiektu, następnie brana jest pod uwagę aktualna powierzchnia bryły brzegowej węzła.

Optymalizacja ta przynosi największe korzyści w odpowiedzi na zapytania o małe regiony dla danych o nierównomiernym rozkładzie w przestrzeni.

Traktowanie przepelnienia jest również nowym rozwiązaniem w stosunku do innych indeksów opartych o struktury z rodziny R-drzew. Jeżeli węzeł nie jest korzeniem, a problem przepelnienia pojawił się na danym poziomie po raz pierwszy zamiast operacji podziału węzła następuje wymuszone ponowne wstawienie części wpisów znajdujących się w węźle. Procedura polega na posortowaniu wpisów względem odległości reprezentowanych przez nie obiektów od środka bryły brzegowej, usunięciu części wpisów (ich ilość określana jest jako parametr drzewa) o najmniejszej lub największej odległości od środka bryły, w zależności od przyjętej strategii a następnie ponownego ich wstawienia do struktury drzewa. Operacja podziału węzła nastąpi, jeżeli wszystkie wpisy po ponownym wstawieniu trafią do tego samego węzła z którego zostały usunięte. Często jednak udaje się uniknąć tej operacji. Skutkiem ubocznym tej operacji jest zmniejszenie nakładania się brył brzegowych, gdy obiekt zostanie umieszczony w innym sąsiedztwie, poprawienie wykorzystania przestrzeni węzłów, gdy obiekt trafi do węzła zawierającego mniej niż średnią ilość wpisów, eliminując oddalone od środka wpisy zbliża kształt brył brzegowych do kwadratów.

Jeżeli dojdzie już do podziału wężła algorytm przewiduje kosztowne operacje mające na celu optymalny podział wpisów pomiędzy dwa wężły, tak by postawione na początku założenia nie ulegały degradacji w wyniku wykonania tej operacji.

Wybór osi podziału następuje po dokonaniu sortowania wpisów względem minimalnych a następnie maksymalnych wartości odpowiadających danej osi. Na podstawie obliczonej sumy obwodów grup wynikowych każdej możliwej kombinacji dystrybucji wpisów ustalany jest minimalny obwód dla danej osi. Proces powtarzany jest dla każdego wymiaru, w jego wyniku ustalona zostaje oś, według której należy dokonać podziału wężła. Dokonując właściwego podziału brana jest pod uwagę powierzchnia nałożenia na sąsiedni wężel powiększona w wyniku dodania wpisu do danej grupy, następnie kryterium minimalnego powiększenia powierzchni bryły wężła.

Szacując koszt takiej operacji należy wziąć pod uwagę dwa sortowania wpisów dla każdego wymiaru o koszcie $M \log(M)$, co stanowi połowę całkowitej złożoności algorytmu. Koszt obliczenie sumy obwodów grup wynikających z dystrybucji dla każdego wymiaru wynosi $2 \cdot (2 \cdot (M - 2m + 2))$. Ostateczna dystrybucja wpisów na podstawie analizy powiększenia powierzchni części wspólnej dla wybranej osi podziału wymaga $2 \cdot (2 \cdot (M - 2m + 2))$ operacji. Jak widać koszt ten może okazać się całkiem spory dla danych o większej liczbie wymiarów.

Niemniej zostało potwierdzone, iż w zależności od typu zapytań wzrost wydajności w tak tworzonej i modyfikowanej strukturze wynosi od 20 do nawet 50% w porównaniu z innymi indeksami z rodziny R-drzew. Jako operacja dostrajająca proponowane jest usunięcie losowo wybranej połowy wpisów oraz ponowne ich wstawienie. Popularność tego wariantu indeksu po 30 latach świadczy o trafności przyjętych przez autorów założeń.

3.4. Drzewa czwórkowe

Drzewa czwórkowe (Quad Trees) jako jedno z pierwszych rozwiązań problemu wprowadzenia porządku przestrzennego zostały zaproponowane w Stanach Zjednoczonych w 1974 r. przez informatyka Raphaela Finkela oraz matematyka Jona Louisa Bentleya w artykule [1]. Wychodząc z założenia, iż procedura wyszukiwania obiektów istniejących w wielu wymiarach w oparciu o złożony klucz może zostać sprowadzona do sumy analizy każdego atrybutu klucza, jako danej jednowymiarowej. Przedstawili rozszerzenie drzewa binarnego tak by jego węzły mogły reprezentować dane dwuwymiarowe (z możliwością uogólnienia do wyższych wymiarów), algorytmy wstawiania (bezpośredniego i zbalansowanego w celu optymalizacji struktury wynikowej) oraz wyszukiwania, mające odpowiednio logarytmiczny ($\log n$) i liniowo logarytmiczny ($n \log n$) czas wykonania względem liczby węzłów. Autorzy publikacji [2] wskazują $O(k \cdot N^{1-1/k})$ jako górne ograniczenie wykonania zapytania zakresowego, gdzie N określa liczbę węzłów natomiast k liczbę wymiarów każdego węzła.

3.4.1. Struktura

Nazwa drzewa pochodzi od reprezentacji dwuwymiarowych danych, jako rekursywnego podziału przestrzeni, którego wynikiem są cztery regiony odpowiadające oznaczeniom **NE**, **NW**, **SW**, **SE** zaczerpniętych z map geograficznych, określane także przez autorów jako **1**, **2**, **3**, **4**. Stanowią one potomków węzła reprezentującego daną przestrzeń. Konstruowanie drzewa rozpoczyna się od korzenia, który obejmuje przestrzennie wszystkie dane, które należy zaindeksować. Strategia podziału może zakładać równomierny podział przestrzeni, jak również podział podyktowany przez dane wejściowe.

Jeżeli dane wejściowe są znane w momencie rozpoczęcia konstrukcji drzewa, możliwe jest stworzenie struktury o zbalansowanej wysokości (każdy węzeł pośredni będzie posiadał czterech potomków, a liście znajdują się na jednym poziomie) będą której rozmiar będzie proporcjonalny do rozmiaru tych danych. Osiągnięcie tych właściwości odbywa się jednak kosztem utraty dynamicznego charakteru drzewa a co za tym idzie, generuje problemem usuwania oraz aktualizacji jego wpisów.

Jeżeli dekompozycja odbywa się na zasadzie równomiernego podziału przestrzeni (dane do zaindeksowania nie muszą w takim wypadku być znane w momencie rozpoczęcia konstrukcji drzewa), wynikowa struktura staje się silnie zależna

od dystrybucji danych wejściowych, jej wysokość jest zbalansowana a rozmiar liniowo proporcjonalny do danych wejściowych jedynie w przypadku ich jednolitego rozkładu.

Kolejną kwestią, którą należy wziąć pod uwagę, jest warunek stopu, określający moment, w którym region nie będzie podlegał dalszym podziałom a więc przyjmie rolę liścia tego drzewa. Możliwe jest budowanie drzew o stałej „rozdzielczości”, gdzie warunkiem stopu jest osiągnięcie określonego globalnie poziomu dekompozycji, lub drzew, których „rozdzielczość” zależy od danych obecnych w danym regionie, które spełniając określony warunek powodują zatrzymanie dalszych podziałów. Istnieją również warianty hybrydowe biorące pod uwagę powyższe warunki jako alternatywę.

3.4.2. Zbalansowane drzewa punktowe

W przypadku, gdy mamy do czynienia z danymi punktowymi, których zbiór jest znany, sposób konstrukcji drzewa jest nieskomplikowany. Należy posortować dane względem wybranego wymiaru a jako punkt podziału wybrać punkt znajdujący się w połowie skonstruowanej w ten sposób listy. Zapewnia to, iż żaden region nie będzie zawierał więcej niż połowę dostępnych danych (choć podział przestrzeni nie będzie równomierny). Procedurę tą powtarzamy dla uzyskanych regionów tworząc tym samym kolejny poziom drzewa. Jako że koszt takiego podziału jest liniowy natomiast wysokość drzewa jest logarytmicznie zależna od ilości danych, całkowity koszt budowy zbalansowanego drzewa wynosi $(dn \log n)$ gdzie d jest wymiarem, n ilością danych.

Aby wyszukać punkt w tak skonstruowanym drzewie, należy porównać jego współrzędne z punktem znajdującym się w korzeniu, wynik porównania wskazuje, do którego z regionów należy skierować dalsze przetwarzanie. Należy przemierzać w ten sposób drzewo do momentu, gdy punkt zostanie znaleziony lub osiągnięty zostanie poziom liścia. Wstawienie nowego punktu polega na wyszukaniu liścia, w którym powinien się znaleźć. Liść wskazuje w tym momencie na dwa punkty, jeden z nich posłuży do podziału przestrzeni, drugi należy umieścić w regionie, na który wskazuje porównanie z punktem podziału.

Operacja usuwania punktu, tak jak w przypadku wszystkich drzew czwórkowych jest znacznie bardziej skomplikowana, wiąże się z koniecznością wyboru punktu z poddrzewa, który mógłby zająć miejsce usuniętego tak by pozostałe punkty poddrzewa były właściwie wskazywane przez operację porównania. Aby tego dokonać, niezbędne okazać się mogą kosztowne operacje reorganizacji poddrzewa.

4. Realizacja indeksów przestrzennych w komercyjnych bazach danych

Ponieważ obecnie procesie przechowywania i udostępniania informacji główną rolę pełnią systemy baz danych, postanowiłem sprawdzić jak prezentuje się funkcjonalność dwóch głównych konkurentów na tym rynku. Przedstawiam specyfikację pochodzącą od ich producentów i przyznam, że byłem zaskoczony przede wszystkim rozbieżnością w czasie wprowadzenia tych rozwiązań. Niemniej zaistnienie indeksów przestrzennych w dwóch najczęściej stosowanych systemach na rynku potwierdza tezę, iż tego typu funkcjonalność jest w dzisiejszych czasach wymogiem w zastosowaniach komercyjnych.

4.1. SQL Server 2008

Może dziwić fakt, iż flagowy produkt Microsoftu do momentu wprowadzenia wersji sygnowanej symbolem 2008 nie posiadał natywnego wsparcia dla danych przestrzennych jak również nie implementował żadnej struktury indeksu mogącej wspierać wyszukiwanie obiektów przestrzennych. Być może przełom nastąpił w momencie, gdy korporacja zaangażowała się w budowę SkyServera indeksującego docelowo 300 milionów ciał niebieskich obsługiwane przez SQL Server 2005. Można sądzić, iż trudności, które pojawiły się w trakcie tego przedsięwzięcia skłoniły firmę do rozpoczęcia prac nad włączeniem do kolejnej iteracji funkcjonalności odpowiedzialnej za obsługę coraz bardziej popularnego sposobu reprezentowania i analizy danych.

4.1.1. Typy danych

Obecna wersja wspiera dwa typy danych:

- **geometry** – kolumny tego typu mogą zawierać punkty, proste, oraz wielokąty zdefiniowane w dwu wymiarowej przestrzeni euklidesowej. Na potrzeby indeksu zdefiniowano następujące procedury badające wzajemne relacje dwóch obiektów tego typu: odległości, równości, zawierania (oraz relację symetryczną), przecięcia oraz wspólnego brzegu.
- **geography** – reprezentuje obiekty geograficzne w przestrzeni powstałej z połączenia dwóch spłaszczonych piramid, na które uprzednio dokonana została projekcja południków półkul ziemskich. Nie jest to przestrzeń euklidesowa.

W tym przypadku procedury zostały ograniczone do testowania relacji:

odległości, równości i przecięcia.

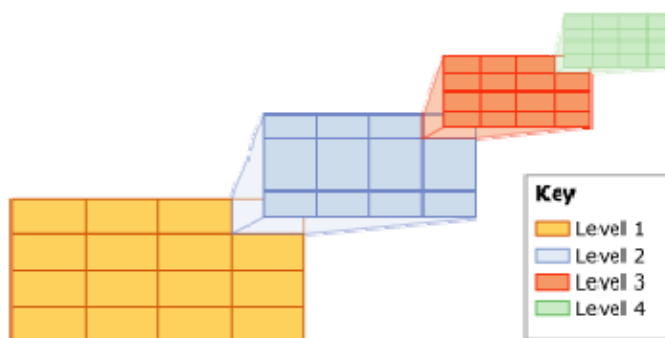
Wymienione procedury mogą zostać użyte jedynie w klauzuli WHERE w postaci:

geography1.STEquals (geography2) = 1
geography1.STDistance (geography2) < liczba

Dla typu geograficznego oprócz spełnienia postawionego warunku logicznego aby ostateczna ewaluacja dała wartość pozytywną, oba obiekty muszą posiadać ten sam RSID informujący, która elipsoida została wykorzystana podczas tworzenia obiektu.

4.1.2. Struktura

Indeksy przestrzenne SQL Servera zbudowane przy użyciu B-drzew. Oznacza to, że indeks musi reprezentować dwu wymiarowe dane przestrzenne w porządku liniowym. Dlatego też zanim dane zostaną wstawione do indeksu, tworzona jest czteropoziomowa hierarchia siatek będących wynikiem równomiernej dekompozycji zadeklarowanej przestrzeni. Każdy kolejny poziom stanowi dalszą dekompozycję podziału z poprzedniego poziomu. Na danym poziomie każda siatka podzielona jest na jednakową liczbę komórek o jednakowym wymiarze.

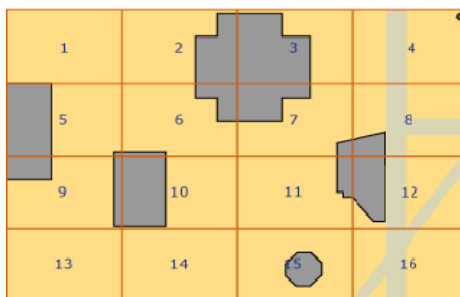


i.06 hierarchiczna dekompozycja przestrzeni

Zakładając cztery poziomy siatek o wymiarach 4x4 oraz dzieląc w ten sposób każdą komórkę najwyższego poziomu otrzymujemy 65000 komórek czwartego poziomu. Ich rozdzielczość jest zależna od zadeklarowanej przez użytkownika przestrzeni początkowej, jako że tego typu indeksy operują na ograniczonym z góry obszarze.

Ilość komórek, na które zostanie podzielona przestrzeń wzdłuż każdej osi wyznacza gęstość indeksu. Parametr ten jest definiowany dla każdego poziomu hierarchii, przez co daje możliwość zoptymalizowania indeksu biorąc pod uwagę wielkość obszaru oraz rozmiar indeksowanych danych.

Każdej komórce danego poziomu przypisany jest numer wyznaczony na podstawie krzywej Hilberta, poniższa ilustracja nie bierze pod uwagę tego faktu.

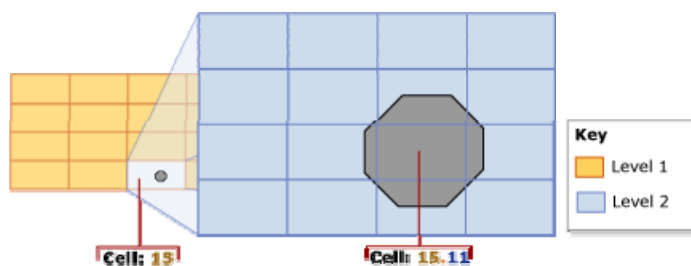


i.07 ograniczona płaszczyzna zawierająca indeksowane obiekty

4.1.3. Operacje

Wstawianie danych do struktury przygotowanej w poprzednim kroku rozpoczyna się na pierwszym poziomie, przemierzając siatkę wzdłuż kolejnych wierszy następuje powiązanie wstawianego obiektu z numerami komórek, które posiadają z nim część wspólną. Proces powtarzany jest rekursywnie dla kolejnych poziomów dekompozycji wyłonionych w ten sposób komórek z uwzględnieniem zasad służących ograniczeniu ilości zbędnych informacji:

- reguła przykrycia
jeżeli obiekt całkowicie przykrywa komórkę, jej numer jest dodawany związanego z nim zbioru, natomiast dana komórka nie jest przetwarzana na kolejnych poziomach, dotyczy to każdego z poziomów dekompozycji
- reguła maksymalnej ilości komórek
wyznacza limit komórek, które mogą zostać związane z danym obiektem, zasada nie dotyczy pierwszego poziomu
- reguła najgłębszego poziomu
ponieważ najlepsze przybliżenie kształtu oraz lokalizacji obiektu określają komórki najniższego poziomu, jedynie ich numer zostają zapisane w indeksie

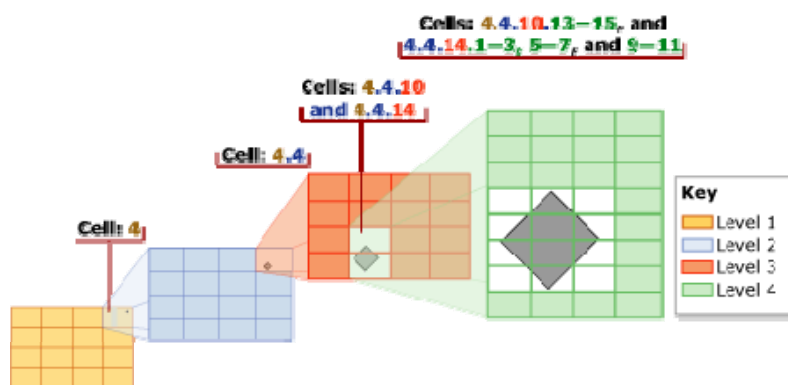


i.08 zasada przykrycia komórki

Limit maksymalnej ilości przypisanych komórek może zostać przekroczony, ponieważ nie dotyczy on pierwszego poziomu dekompozycji, w takim wypadku żadna komórka nie jest przetwarzana na kolejnym poziomie. W przeciwnym wypadku wybierana jest komórka o najmniejszym numerze, następuje sprawdzenie, czy przetwarzanie na niższym poziomie spowoduje przekroczenie dopuszczalnej ilości związanych komórek, jeżeli tak, zostaje związana wartość z aktualnego poziomu. Proces ten jest powtarzany dla każdej kolejnej komórki aż osiągnie najgłębszy poziom, lub zostanie osiągnięty limit komórek. W przykładzie z ilustracji i08, drugi poziom zostanie przetworzony tylko, gdy limit ten wynosi dziewięć lub więcej, w przeciwnym wypadku zanotowany zostanie jedynie adres komórki pierwszego poziomu.

Domyślna ilość komórek, które mogą zostać związane z obiektem to 16, co według twórców stanowi racjonalny kompromis pomiędzy dokładnością lokalizacji a ilością danych przechowywanych dla każdego obiektu. Przy zakładaniu indeksu użytkownik może ustawić limit na wartość z zakresu 1-8192 [5].

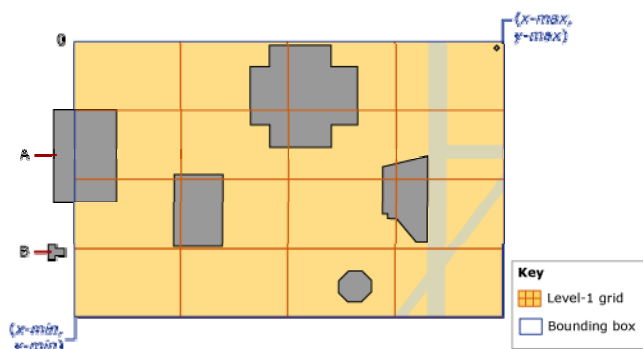
Zasada najgłębszego poziomu korzysta z hierarchicznego zapisu adresów komórek niższego poziomu. Mimo że analiza odbyła się na wszystkich czterech poziomach zapis adresów komórek ostatniego poziomu jest wystarczający by odtworzyć położenie komórki na wyższych poziomach. Ilustracja i09 dobrze przedstawia całą ideę indeksowania obiektów realizowaną przez SQL Server.



i.09 reguła najgłębszego poziomu

Otrzymane w ten sposób hierarchiczne określenie lokalizacji staje się przedmiotem klasycznego indeksowania w strukturze B-drzewa, w którym staje się kluczem, natomiast identyfikator wartością. Wyszukiwanie polega na ustaleniu na podobnych zasadach lokalizacji zakresu wyszukiwania, następnie dekompozycji jego klucza i ustalania wyników, w zależności od rodzaju zapytania.

Ponieważ sposób tworzenia indeksu wymaga ograniczenia przestrzeni przechowywane są koordynaty punktów definiujących pełniących tę funkcję prostokąt. Obszar na zewnątrz prostokąta ograniczającego jest traktowany, jako jeden obszar o numerze 0, dekompozycji podlega jedynie obszar wewnątrz tego prostokąta. Należy zwrócić uwagę, iż jedynie obiekty znajdujące się w całości wewnątrz obszaru dekompozycji mogą korzystać z optymalizacji dostarczanej przez indeks, dlatego też użytkownik powinien określić przestrzeń tak by zawierała wszystkie lub przynajmniej większość obiektów indeksowanej kolumny.



i.10 obiekty A i B nie podlegają indeksowaniu

Twórcy wyszli z mało popularnego założenia, iż należy uniezależnić się od indeksowanych danych kosztem wielu ograniczeń i wstępnych założeń, które użytkownik musi zaakceptować i wypełnić. Podejrzewam, że może to wynikać z obecności silnej jednostki bądź grupy przekonanej o słuszności tego podejścia lub istnieniu systemów spadkowych (a być może nabyciu takiej funkcjonalności), które od dłuższego czasu z niewiadomych przyczyn nie mogły ujrzeć światła dziennego w postaci indeksu przestrzennego dla SQL Servera.

4.2. Oracle Locator

Firma Oracle już dawno dostrzegła zapotrzebowanie na integrację bazy danych z funkcjami przechowywania, odpytywania i analizy danych przestrzennych oraz wykorzystywania ich w aplikacjach biznesowych umożliwiając organizacjom podejmowanie lepszych decyzji, bardziej trafne odpowiedzi na żądania klientów oraz redukcję kosztów mając możliwość uwzględnienia i analizy danych topologicznych.

Włączając do serwera funkcje takie jak zarządzanie danymi GeoRaster, sieciowe oraz topologiczne modele danych, algorytmy geo-kodowania oraz routingu jak również analizy danych na podstawie przestrzennej lokalizacji świadczy o wycelowaniu produktu w przedsiębiorstwa chcące optymalizować zachodzące w nich procesy [11].

Poraz pierwszy temat wielowymiarowych danych oraz geometrii zostały podniesione w roku 1994 w wersji 7 bazy danych. Do wersji 8i informacje o obiektach geometrycznych przechowywane były w licznych kolumnach tabel realcyjnych. Od wersji 8i zaczął obowiązywać specjalny model dla danych SDO_GEOMETRY. W wersji 9i pojawiło się wsparcie dla systemów koordynat geograficznych. Aktualna wersja 11i dalej rozszerza funkcjonalność o integrację z pośrednimi aplikacjami umożliwiającymi reprezentację danych na mapach dostarczanych przez wiodących producentów z branży kartografii cyfrowej. Obecnie Locator, jako rozszerzenie obecne w każdej bazie danych Oracle zapewnia natywną obsługę typów, zapytań oraz analizy danych dotyczących lokalizacji za pośrednictwem języka SQL.

Przedstawiając obsługiwane typy podawane przykłady również sugerują obszar zastosowań bazy, jako źródła danych dla systemów GIS:

- ◆ Punkty – mogą reprezentować budynki, hydranty, maszty telekomunikacyjne, platformy wiertnicze, poruszające się pojazdy.
- ◆ Linie – mogą przedstawiać drogi, linie kolejowe, linie energetyczne
- ◆ Złożone wielokąty – reprezentacja miast, dzielnic, rozlewisk, pól naftowych

Wewnętrznie dane przestrzenne są modelowane jako warstwy, korzystające z kolumny typu geometrycznego który uwzględnia projekcję geometrii kuli ziemskiej.

Wraz z wprowadzeniem modelu danych przestrzennych zaimplementowano indeks oparty o R-drzewo dający możliwość porządkowania danych dwu, trzy i czterowymiarowych z układzie geometrycznym lub geograficznym. Oraz wykonywanie zapytań zakresowych dostarczających obiekty spełniające relację zawierania oraz części wspólnej (przecięcia). Obecna wersja udostępnia również indeksy w postaci drzew czwórkowych (statycznych, dynamicznych oraz hybrydowych) oraz poszerzony zestaw operatorów, także punktowych, umożliwiających między innymi wykonywanie zapytań o najbliższych sąsiadów.

Optymalizacje wyszukiwania danych pozwoliły przejść z funkcji czasu względem przechowywanych danych do czasu zależnego od rzeczywiście pobranych danych. Osiągnięto to głównie dzięki zastosowaniu dwu warstwowego modelu zapytań. Bazując na aproksymacjach obiektów przechowywanych przez indeks dokonuje się wstępnego filtrowania obiektów dostarczając kandydatów do odpowiedzi na zapytanie. Na drugim poziomie stosowane są odpowiednie operatory na rzeczywistych reprezentacjach obiektów w wyniku czego formułowana jest ostateczna odpowiedź na zapytanie.

Obliczenia te są znacznie bardziej skomplikowane obliczeniowo, jednak wykonywane na stosunkowo niewielkiej ilości danych. Eliminowanie obiektów z poza zakresu zainteresowania w oczywisty sposób wpływa na poprawę wydajności [11].

Architektura bazy danych umożliwia jest podział logicznej tabeli na dwie lub więcej tabele fizycznej, z których każda posiada własny indeks. Umożliwia to obsługę wielu procesorów do obsługi zapytań oraz tworzenia indeksów przyspieszając tym samym operacje na dużych zbiorach nie punktowych danych przestrzennych.

Indeksy oparte na funkcjach pozwalają odpytywać oraz analizować relacyjne dane na podstawie kilku kolumn będących atrybutami przestrzennymi (np. szerokości i długości geograficznych), co jest użyteczne gdy organizacja dysponuje relacyjnym schematem zawierającym informacje o lokalizacji natomiast nie ma możliwości konwersji danych do typu SDO_GEOMETRY.

W zakresie danych geodezyjnych (geograficznych) firma Oracle utrzymuje, iż wspiera ponad 1000 systemów mapowania koordynat, systemy definiowane przez użytkownika oraz mechanizmy przechodzenia pomiędzy każdym z nich. Znalazł się wśród nich również standard grupy EPSG, co kolejny raz potwierdza zainteresowanie firmy obszarami pól naftowych.

4.3. Podsumowanie

Rozwiązanie firmy Oracle wydaje się być znacznie bardziej dojrzałe, posiada mniej ograniczeń, oraz permanentnie znajduje się w fazie rozwoju. Jeżeli dodać do tego fakt, iż oprócz rozszerzenia bazy danych firma równolegle rozwija pełnowartościowy system GIS – Oracle Spatial, potencjalne korzyści wynikające z tego rozwiązania wydają się być ogromne.

Funkcje przestrzenne w SQL Serverze wydają się być wymuszone przez wymogi rynku a podejście od strony technicznej znacznie mniej zaawansowane niż konkurencji. Można powiedzieć, iż rozwiązanie to posiada ograniczenia, podczas gdy konkurencja posiada możliwości. Oba rozwiązania skierowane są do tej samej grupy docelowej zakładając ich wykorzystanie w systemach informacji geograficznej, nie dziwi więc fakt, iż Oracle posiada w tym segmencie ok. 80% rynku, istnienie tego rozwiązania na rynku od ponad 10 lat i jasno określony kierunek rozwoju pozwala wydać jednoznaczny werdykt.

5. Realizacja Indeksu Przestrzennego

Po zapoznaniu się z szerokim spektrum struktur, algorytmów i strategii indeksowania danych przestrzennych zdecydowałem, jako część niniejszej pracy dokonać implementacji indeksu opartego o strukturę R-drzewa w postaci przedstawionej w 1984 r. przez Antomna Guttmana. Przede wszystkim ze względu na fakt, iż jest to struktura dynamiczna, jeżeli chodzi o wydajność indeksowania różnych typów i rozkładów danych - uniwersalna, oraz elastyczna w kategoriach rozszerzalności. Indeks zaimplementowany w oparciu o założenia Guttmana jest otwarty na nowe metody wyszukiwania wykorzystujące operacje na bryle brzegowej, oraz nowe metody podziału węzła bez konieczności modyfikacji istniejącej struktury. Jest więc w tym podejściu przestrzeń by rozwijać istniejące już rozwiązanie pracując nad rozszerzeniem funkcjonalności jak również poprawą wydajności, bez konieczności tworzenia od początku, dobrze znanej, lecz biorąc pod uwagę wszystkie założenia indeksu, nie trywialnej struktury drzewa.

Jak wspomniałem na początku pracy, zależało mi na stworzeniu systemu uniwersalnego, choć nie koniecznie w pełni "generycznego", zrealizowanego w zgodzie z obiektywnym podejściem tak by osoby zainteresowana jego rozwojem nie zostały uwięzione w strukturze realizującej konkretny zakres funkcjonalności a także nie musiały poświęcić zbyt wiele czasu na analizę części składowych, z których niewątpliwie należy skorzystać by realizować stawiane przed indeksem zadania.

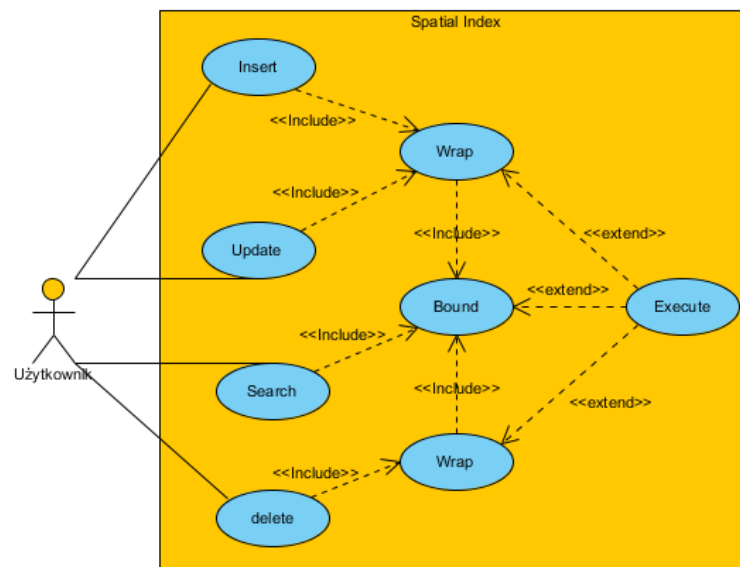
W rozdziale tym przedstawiam proces tworzenia systemu stanowiącego integralną część niniejszej pracy, aby uprościć korzystanie ze źródeł, które mimo zastosowania dobrych praktyk programistycznych oraz szczególnej dbałości o czytelność i strukturę kodu, mogą okazać się w pierwszej chwili kłopotliwe w interpretacji. Starłem się pokazać w nim, wszystkie procesy myślowe od etapu analizy przez decyzje projektowe oraz założenia implementacyjne po krótkie procedury testowe oraz przykłady zastosowania gotowego systemu.

Do modelowania wszystkich perspektyw wykorzystałem notację UML, dokumentację kodu opartą na anotacjach w formie JavaDOC załączam na nośniku zawierającym kod źródłowy, skompilowane klasy ora archiwum JAR.

5.1. Analiza

Z punktu widzenia użytkownika funkcjonalność indeksu jest niezmiernie prosta, wszystko, czego oczekuje od takiego systemu to możliwość dodawania, aktualizacji, oraz usuwania własnych obiektów przestrzennych, a także szybkiego otrzymania identyfikatorów (kluczy tabel relacyjnych lub referencji) podzbioru obiektów spełniających podane kryteria w odpowiedzi na zadane zapytanie. Należy w tym miejscu jeszcze raz podkreślić, iż nie jest to rozwiązanie przezroczyste dla użytkownika i wymaga z jego strony bezpośredniego wywołania udostępnianych przez API metod w celu realizacji konkretnych zadań.

Niektóre z nich można zautomatyzować, posiadając na przykład "fabrykę obiektów", która dodaje każdy nowo stworzony obiekt dodaje do indeksu. Wiążąc każdy obiekt z danym indeksem można w typowych operacjach *set()* wymusić aktualizację indeksu po zmianie kluczowych wartości. Niemniej, wymaga to od użytkownika dodatkowego nakładu pracy na stworzenie własnego mechanizmu korzystania z indeksu. Wyszukiwanie natomiast, z założenia odbywa się na wyraźne żądanie użytkownika, w oparciu o wyspecyfikowane przez niego parametry.



i.11 Model przypadków użycia indeksu przestrzennego

Biorąc pod uwagę powyższe ograniczenia, w procesie definiowania wymagań stało się dla mnie ważne, aby udostępniane operacje mogłyby działać na dowolnych danych przestrzennych, będących w posiadaniu użytkownika, wymagając jak najmniejszej dalszej ingerencji w strukturę tych danych. Dlatego też procedura opakowywania obiektów odbywa się wewnątrz biblioteki, z której korzysta użytkownik.

5.1.1. Wymagania

Na tym etapie zidentyfikowałem następujące wymagania, które powinien realizować system indeksowania przestrzennego w ostatecznej wersji prototypu:

❖ Funkcjonalne

- ◆ możliwość naprzemiennego wykonywania operacji wstawiania, usuwania, aktualizacji oraz wyszukiwania bez konieczności dokonywania operacji administracyjnych zapewniających spójność i aktualność indeksu
- ◆ możliwość wykorzystania dowolnych danych przestrzennych, jako kluczy
- ◆ możliwość wykorzystania tego samego typu danych do wyszukiwania
- ◆ możliwość wykorzystania dowolnego obiektu w charakterze identyfikatora, czyli wartości zwracanej w odpowiedzi na zapytanie (włączając w to typy i klasy obiektów języka Java jak int, long, double, czy String)
- ◆ możliwość dodania do indeksu danych, które jednocześnie będą stanowiły klucz oraz identyfikator (będący referencją do tego obiektu)
- ◆ możliwość wykonania różnego rodzaju operacji wyszukiwania
- ◆ przestrzeń obiektów dodawanych do indeksu nie powinna być ograniczona
- ◆ otrzymywane wyniki powinny być zawsze poprawne w kontekście układu współrzędnych oraz geometrii euklidesowej

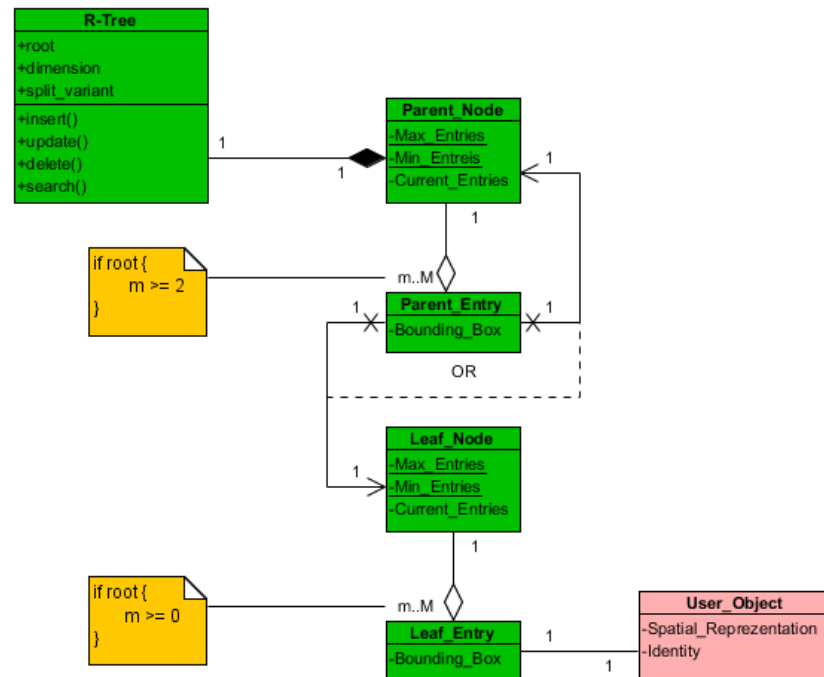
❖ Niefunkcjonalne

- ◆ struktury indeksu powinny operować na pamięci podstawowej (w przeciwieństwie do założeń autora idei R-drzew)
- ◆ implementacja powinna polegać na konstrukcjach obiektowych języka Java, tak by zmaksymalizować łatwość jej utrzymania i modyfikacji
- ◆ implementacja powinna unikać konstrukcji mogących wpłynąć na obniżenie wydajności obliczeniowej, oraz ograniczać deklaracje nadmiarowych zmiennych ze względu na rodzaj pamięci na której bazuje
- ◆ struktura indeksu nie musi zapewniać trwałości indeksu między sesjami

Wymagania z grupy funkcjonalnej posłużą przede wszystkim do projektu API. Założenia niefunkcjonalne będą miały wpływ na mechanizmy wewnętrzne indeksu.

5.1.2. Analiza statyczna

Przed przejściem do fazy projektowania, w której starałem się odnieść się w jak największym stopniu do sprecyzowanych w wymaganiach, wykonałem model pojęciowy przedstawiający byty pochodzące z dziedziny problemu. Posłużył on do identyfikacji przyszłych obiektów a także wyrycia związków między nimi oraz ich licznosci.



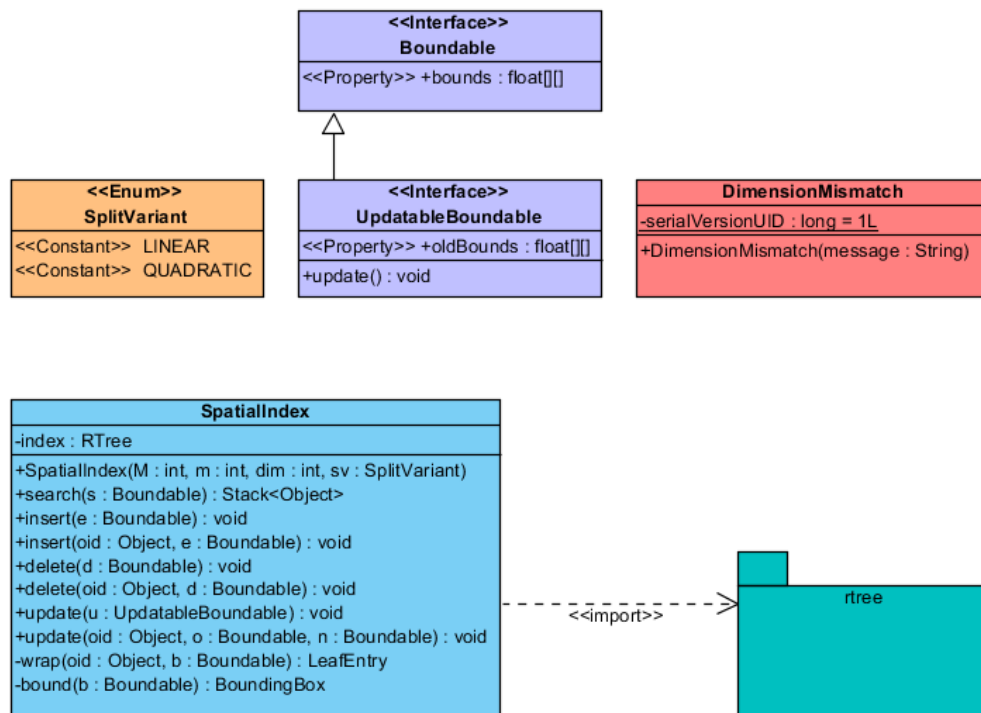
i.12 Model pojęciowy struktury R-drzewa

Stwierdziłem konieczność istnienia w systemie klas obiektów reprezentujących następujące byty tworzące strukturę R-Drzewa, ustanawiając tym samym pewną nomenklaturę stosowaną w dalszej dokumentacji:

- ◆ drzewo – zawiera zbiór dopuszczalnych operacji oraz parametry globalne
- ◆ węzeł rodzic – zawiera wpisy wskazujące na węzły niższego poziomu
- ◆ węzeł liść – zawiera wpisy wskazujące na wartości niekluczowe
- ◆ wpis rodzica – może wskazywać węzeł liścia lub rodzica, zawiera bryłę brzegowa obejmującą wszystkie elementy poddrzewa danego wpisu
- ◆ wpis liścia – wskazuje indeksowany obiekt (jego identyfikator lub referencję do niego) – jest wartością indeksu. Bryła brzegowa wpisu stanowi przestrzenną reprezentację indeksowanego obiektu – jest kluczem indeksu.

5.2. Projekt

Na podstawie zdefiniowanych wymagań oraz przeprowadzonych analiz przystąpiłem do identyfikacji klas obiektów ich związków oraz adekwatnych abstrakcji programistycznych dostępnych w języku Java. Przedstawiony diagram klas jest finalną postacią, która ewoluowała wraz z postępem implementacji, niemniej szkielet, który posłużył do implementacji pierwszej wersji w pełni działającego indeksu niewiele się zmienił. Uszczegółowione zostały klasyfikatory dostępu, dodane atrybutów, zmieniły się typów niektórych pól, pojawiło wiele metod pomocniczych. Pozwala to sądzić, iż czas poświęcony na analizę zaowocował zmniejszeniem konieczności dostosowywania projektu do pojawiających się problemów w kolejnych etapach fazy implementacji.



i.14 klasy wchodzące w skład API

Ilustracja i14 przedstawia pakiet *spatial* importowany przez końcowego użytkownika, zawierający zestaw abstrakcji niezbędnych do korzystania z indeksu. Instancja klasy *SpatialIndeks* reprezentuje pojedynczy indeks korzystający w tym przypadku ze struktury R-Drzewa. Takie rozdzielenie wewnętrznych mechanizmów indeksowania od warstwy dostępnej dla użytkownika pozwala na rozszerzanie i zmianę funkcjonalności udostępnianej przez indeks bez wpływu na sposób korzystania z niego przez odbiorców. Kolejna strona zawiera klasy wchodzące w skład pakietu *spatial.rtree* który jest importowany przez API

InternalTest
+main(args : String []): void

DimensionException
-serialVersionUID : long = 1L
+DimensionException(message : String)

RTree
-DEFAULT_MAX : int = 50
-DEFAULT_MIN : int = 25
#maxEntries : int
#minEntries : int
+dimensions : int
#split : SplitVariant
~root : Node
#RTree()
+RTree(nMax : int, nMin : int, dim : int, variant : SplitVariant)
+insert(e : LeafEntry) : void
+search(s : BoundingBox) : Stack<LeafEntry>
+delete(e : LeafEntry) : boolean
+update(oe : LeafEntry, ne : LeafEntry) : boolean
-condenseTree(l : Node) : void
-reinsert(q : Stack<Node>) : void
-chooseLeaf(e : Entry) : Node
-chooseNode(e : Entry, l : int) : Node
-adjustTree(n : Node, nn : Node) : void

BoundingBox
+dimensions : int
+max : float[]
+min : float[]
+BoundingBox(x1 : float, y1 : float, x2 : float, y2 : float)
+BoundingBox(min : float [], max : float [])
+set(x1 : float, y1 : float, x2 : float, y2 : float) : void
+set(min : float [], max : float []) : void
+copy() : BoundingBox
+overlaps(r : BoundingBox) : boolean
+contains(r : BoundingBox) : boolean
+volume() : float
+enlargement(bb : BoundingBox) : float
+enlarge(r : BoundingBox) : void
+toString() : String
+equals(o : Object) : boolean

ParentNode
~ParentNode(l : int, tree : RTree)
#newNode(l : int) : Node
~get(i : int) : ParentEntry
~add(e : Entry) : Node
~chooseSubTree(e : Entry) : Node
~remove(l : Node) : void

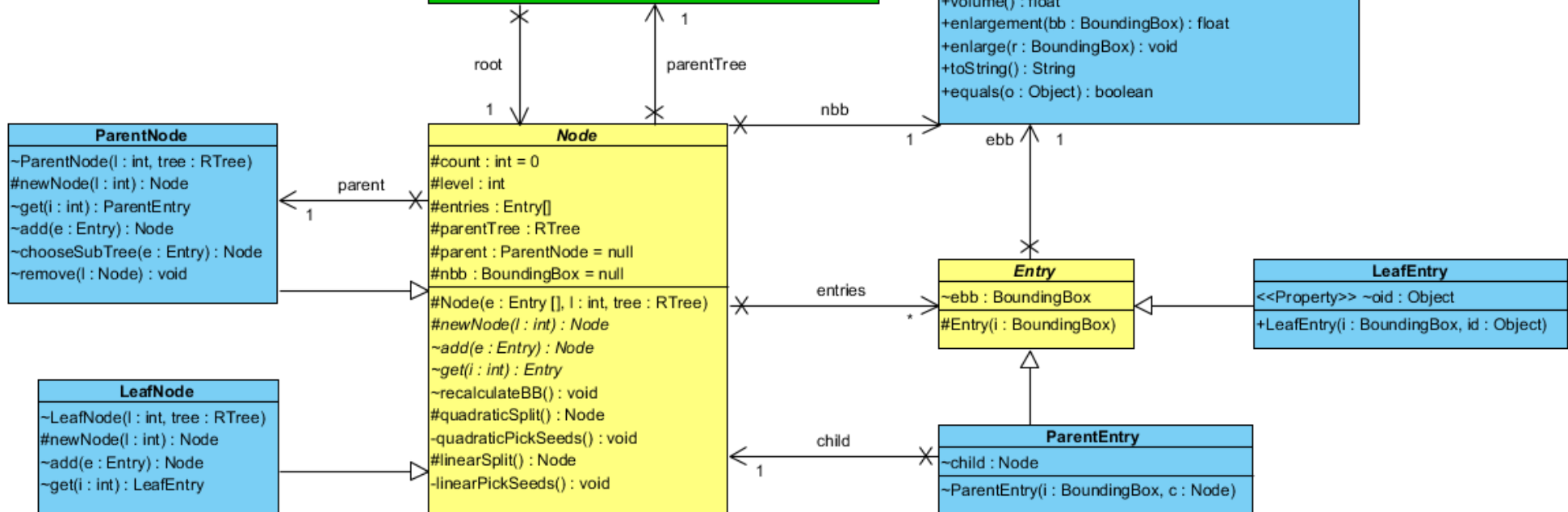
Node
#count : int = 0
#level : int
#entries : Entry[]
#parentTree : RTree
#parent : ParentNode = null
#nbb : BoundingBox = null
#Node(e : Entry [], l : int, tree : RTree)
#newNode(l : int) : Node
~add(e : Entry) : Node
~get(i : int) : Entry
~recalculateBB() : void
#quadraticSplit() : Node
-quadraticPickSeeds() : void
#linearSplit() : Node
-linearPickSeeds() : void

Entry
~ebb : BoundingBox
#Entry(i : BoundingBox)

LeafEntry
<<Property>> ~aid : Object
+LeafEntry(i : BoundingBox, id : Object)

LeafNode
~LeafNode(l : int, tree : RTree)
#newNode(l : int) : Node
~add(e : Entry) : Node
~get(i : int) : LeafEntry

ParentEntry
~child : Node
~ParentEntry(i : BoundingBox, c : Node)



5.2.1. API

Nigdy nie jest zbyt wcześnie, żeby pomyśleć o użytkowniku i jak wynika z mojego doświadczenia w tym przypadku, takie myślenie w przód się opłaca. W rzeczywistości koncepcję udostępniania funkcjonalności użytkownikowi sprecyzowałem w zaawansowanym stadium rozwoju struktur wewnętrznych, co wprowadziło pewien problem decyzyjny. Ponieważ żadna część systemu nie była w pełni zamknięta, istniała możliwość wprowadzenia zmian w jej wnętrzu jak i powstającym interfejsie. Szczęśliwie wynikiem nie było zbyt wiele niespójności koncepcyjnych, które właściwie już po zakończeniu prac zostały wykryte i wyeliminowane. Dobrze zdefiniowane API będzie cechować się niewielką zmiennością, oraz wysoką kohezją.

Aby odnieść się do wymagań funkcjonalnych i udostępnić możliwość indeksowania dowolnego obiektu przestrzennego należało określić interfejs, który indeksowany obiekt będzie implementował. W tym przypadku jest to żądanie dostarczenia aproksymacji danych przestrzennych w postaci bryły brzegowej. Jeżeli API powstało by przed rozpoczęciem implementacji struktury drzewa, określiłoby tym samym, jaką formę reprezentacji bryły należy w niej zastosować, tak jak teraz określa jej formę dla ewentualnych rozszerzeń.

Aby dostarczyć użytkownikowi możliwość wyboru wariantu, w jaki drzewo będzie reagować na przepełnienie węzła a jednocześnie ograniczyć i udostępnić zbiór tych możliwości stworzony został typ wyliczeniowy, który może być dziedziczony i rozszerzany w miarę pojawiania się nowych wariantów. Umieszczenie go w tym pakiecie daje możliwość korzystania z niego zarówno użytkownikowi jak i procesom z wnętrza systemu, ewentualna modyfikacja musiałaby odbyć się w tym jednym punkcie.

W interakcji z użytkownikiem należy przewidzieć sytuacje, w których dostarczone przez niego dane lub wykonane na nich operacje mogłyby spowodować błędne lub nieokreślone działanie programu. Przy tak liberalnych i uniwersalnych założeniach jakie przyjąłem, niewiele takich sytuacji ma szansę zaistnieć, pomijając oczywiście pomyłki programisty, które powinna wychwycić statyczna kontrola typów. Zidentyfikowałem jedną potencjalnie niejasną sytuację. Choć struktura drzewa zakłada możliwość indeksowania danych z dowolnego wymiaru, przy tworzeniu drzewa wymiar ten jest określany i stały w trakcie całego cyklu życiowego. Współistnienie w indeksie obiektów z różnych wymiarów byłoby co najmniej kłopotliwe, o ile w ogóle możliwe. Rozważaniom można by poddać ewentualność korzystania z brył o innym wymiarze niż indeksowane dane w celu ich wyszukiwania. Obarczone musiałoby to jednak

zostać wieloma założeniami, których słuszności nie można by dowieść (np. traktowanie brakującego wymiaru, jako ograniczonego przez + i - nieskończoność. Pojawia się jednak pytanie, którego wymiaru brakuje itd.) W związku z tym zdefiniowałem wyjątek niezgodności wymiarów (dotyczący wszystkich operacji na indeksie), który oznaczyłem na diagramie kolorem czerwonym.

Jako ostatni wprowadziłem interfejs pozwalający aktualizować obiekty przy użyciu metody o jednym parametrze, którego wprowadzenie zupełnie nie wpłynęło a także było zupełnie nie zależne od aktywności i struktury pakietu `spatial.rtree`.

5.2.2. Struktury indeksu

Na tym etapie ustaliłem szkielet klas, oraz wyłoniłem ich podstawowe atrybuty, metody oraz konstruktory. Pozostałe informacje umieszczone na diagramie pojawiały się stopniowo. Zdecydowałem się na przedstawienie klas w ostatecznej formie, aby nie mnożyć diagramów w objętości opracowania. Jak wspominałem wcześniej, projekt API powstał w trakcie prac nad strukturą drzewa, w tym momencie konieczne stało się podzielenie systemu na moduły oraz specyfikacja dostępu.

Na ilustracji i15, kolorem żółtym wyróżniłem klasy abstrakcyjne, które zawierają część wspólnych atrybutów i metod. Ich realizacją są klasy węzła potomnego i liścia oraz wpisów odpowiednio, rodziców i potomnych. Skrystalizowała się również forma przechowywania koordynat aproksymacji obiektów przestrzennych. Jednak odmienny od proponowanego w oryginalnym artykule. Prawdopodobnie postrzegany przez większość jako mniej intuicyjny, znacznie ułatwiający natomiast obliczenia na dowolnym wymiarze w pętlach.

W czasie projektowania struktury, po głębszej analizie warunków brzegowych algorytmu Gutmana stwierdziłem, że lepszym rozwiązaniem nawigacji w kierunku korzenia będzie wskazanie przez węzeł potomny na węzeł wyższego poziomu, jako na rodzica nie zaś na wpis tego węzła, który z kolei wskazuje na węzeł potomny. Asocjacje nie są więc w tym względzie symetryczne. Ustaliłem również, że każdy węzeł będzie posiadał atrybut pozwalający na taką nawigację, jako że jedynym węzłem nie posiadającym rodzica jest korzeń, który w dodatku może być węzłem rodzicem lub liściem. Obsługę nieregularnych danych przedstawię w kolejnym rozdziale poświęconym implementacji. Cały proces postaram się przedstawić możliwie szczegółowo, uzasadniając ważniejsze decyzji, ich konsekwencje, pozytywne oraz negatywne strony przyjęcia danego podejścia.

5.3. Implementacja

Niewątpliwie, w tej fazie projektu może pojawić się najwięcej błędów i przekłamań w stosunku do założeń, nawet najlepszy kod, kiedy jest go dużo staje się mało czytelny a do tego wraz ze wzrostem złożoności i możliwości sumowania lub niwelowania błędów w zależności od danych, trudny do weryfikacji. Warto mieć więc pewny, dobrze zdefiniowany model przeciw któremu nawet fragmentarycznie można porównać źródła programu, przepływ sterowania w trybie debug, oraz wartości wyjściowe dla konkretnych danych wejściowych.

Najczęściej jednak dopiero na tym etapie zapada większość decyzji mających wpływ na jakość systemu, ze względu na próby optymalizacji przepływu sterownia a równie często w wyniku złych praktyk programistycznych oraz niewłaściwego podejścia do projektu. Staralem się uniknąć takiej sytuacji, przynajmniej w zakresie najistotniejszych rozstrzygnięć. Jednak ze względu na charakter biblioteki (jest to chyba bardziej adekwatne określenie omawianego przypadku) będącej czystym algorytmem oraz chęć optymalnej, pod względem czasu wykonania i zaadresowanej pamięci, możliwie wiernej realizacji indeksu w stosunku do oryginalnej koncepcji przedstawionej w artykule Gutmana na gruncie obiektowym, decyzje ulegały koniecznym zmianom. Trzymałem się jednak kilku założeń, które w skrócie przedstawiam poniżej.

5.3.1. Wszystko jest obiektem

Założenia obiektowego paradygmatu programowania nakazują tworzenie i wykorzystywanie klas obiektów reprezentujących byty z dziedziny problemowej, w domyśle umożliwiając w miarę dokładne przeniesienie w świat informatyki przedmiotów i zdarzeń ze świata rzeczywistego. Ma to swoje silne uzasadnienie przy modelowaniu systemów odnoszących się do tego typu dziedziny problemowej (choć dość często osiągnięty w ten sposób efekt kompromitowany jest przez zastosowanie relacyjnych baz danych lub innych nie obiektowych technik przetwarzania i składowania danych). Na pierwszy rzut oka, algorytm indeksowania ma niewiele wspólnego z otaczającą nas rzeczywistością. Jednak nawet tu można odnaleźć sporo analogii. Od najprostszego porównania, wskazującego na pochodzenie nazwy struktury, do indeksu w książce umożliwiającego szybkie odnalezienie informacji według znanego klucza, gdzie analogia często okazuje się zupełna, po nieco poetyckie określenie struktury na której bazuje, jako drzewa posiadającego korzeń, gałęzie i liście. Być może ktoś jest w stanie bez zbędnych narzutów związanych z nagłówkami klas, wyrównywania do 8 bitów

wygenerować skomplikowaną strukturę indeksu opartą na trzech listach, kilku integerach oraz mnóstwie niezrozumiałych proceduralnych obliczeń. Niemniej jednak dostępność języka będącego w stanie reprezentować rzeczywistość rodzi chęć stosowania go w taki sposób by równie spójnie i efektywnie tworzyć i manipulować obiektami wykraczającymi daleko poza szkolne systemy typu `employee extends osoba, get nazwisko`. W momencie zderzenia z matematyką drzew i fizyką indeksów pojawia się pokusa porzucenia tych słusznych moim zdaniem założeń. Nie ulegając, deleguje operacje do metod klas, do których powinny przynależeć, nie staram się zrobić wszystkiego w jednym miejscu. Korzystam z generalizacji i polimorfizmu. Staram się ograniczać dostęp na tyle na ile to możliwe i na ile wydaje się być konieczne. Zamykam implementację, izoluję od interfejsów, prezentuję funkcjonalność jak fasadę, mechanizmy trzymam w piwnicy, umawiam się co mam zrobić, nie jak.

5.3.2. Dobór typów

Chcąc iść w zgodzie z tym przedstawionym założeniem, jeżeli orzekam o prawdzie lub fałszu, wykorzystuję wartość *boolean*, nie używam jednej zmiennej *int* do obsłużenia czterech przypadków, staram się za to wykorzystać zarówno wartość *true* jak i *false*. W obliczeniach pomocniczych wykorzystuję typ *int*, nie próbuję czynić oszczędności stosując zmienne typu *short* czy *byte* nawet jeżeli zakładam, że ich zakres nie zostałby przekroczony. Wychodząc z założenia, że na platformach 32 bitowych, z których wciąż korzysta większość użytkowników typ *int* najlepiej odpowiada architekturze algebraicznego przetwarzania, natomiast zysk zajęcia pamięci przy obecnych standardach wyposażenia jednostek w RAM jest zupełnie znikomy. W sytuacjach, które wymagają nawigacji umieszczam referencje, unikając "luźnej" struktury o niejasnych zależnościach.

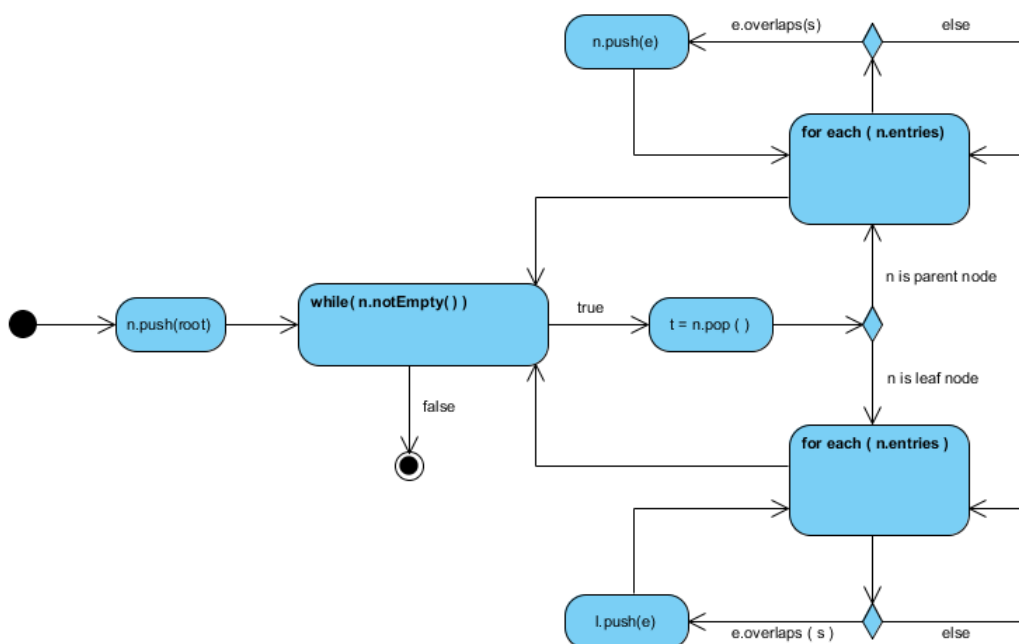
5.3.3. Wyjątki

Nie programuję przez wyjątki, rezerwuje je dla sytuacji, które są naprawdę wyjątkowe, i które oprócz wyświetlenia adekwatnego komunikatu wymagają przerwania przetwarzania i propagacji informacji w stosie wywołań. Przykładem niech będzie tworzenie drzewa, w przypadku podania niepoprawnych minimalnych i maksymalnych wartości wpisów w węźle, konstruktor przyjmuje wartości domyślne, uznane za uniwersalne. Jest to bezpieczne założenie, niepowodujące anomalii, korygujące błąd użytkownika. Jeżeli podany zostanie nieobsługiwany wymiar indeksu, np. 0 użytkownik zostanie poinformowany o błędzie, przetwarzanie zostanie przerwane a indeks nie zostanie utworzony, gdyż nie ma sensu zgadywać, o jaki wymiar danych chodzi.

Stan wewnątrz indeksu jest kontrolowany, a każda sytuacja obsłużona, dlatego też nie zachodzi potrzeba ani możliwość zgłoszenia wyjątku, jeżeli dane wprowadzone przez użytkownika są poprawne. Dlatego pojawiają się one jedynie w miejscach interakcji z użytkownikiem. W przypadku wyszukiwania, wstawiania, aktualizacji oraz usuwania może zostać zgłoszony wyjątek informujący o nieprawidłowym wymiarze danych, tj. odmiennym od istniejącego w indeksie, i choć jest kilka możliwości przetwarzania w obecnej wersji jest to operacja niezdefiniowana.

5.3.4. Rekurencja

Unikanie tej formy kierowania przepływu sterowania należy moim zdaniem zaliczyć do dobrych praktyk programistycznych. Z pewnością znajduje ona swoje zastosowanie, gdy można określić ją jako 'płytką', czyli liczba zagnieżdżonych wywołań nie przekracza kilku. Nie jest dobrym jej przykładem często powtarzany, również w środowiskach akademickich, sposób wyliczania ciągu Fibonacciego, choć głównym problemem nie jest tu przekazywanie parametrów, a raczej to, że ciąg rośnie wykładniczo. Autor w swym artykule proponuje algorytm przeszukiwania drzewa na zasadzie rekursywnego wywołania procedury. Poniżej przedstawiam diagram aktywności rozwiązujący to zadanie z zastosowaniem dodatkowego stosu bez potrzeby wykorzystania rekurencji. Czytelność przedstawionego przepływu wydaje się być zadowalająca natomiast wydajność prawdopodobnie (nie wykonałem porównania z wersją rekurencyjną) znacznie przewyższa tamtą postać.



i.16 algorytm search(s) na wyjściu zwraca stos rezultatów l

5.3.5. Interfejsy

Proponuje dwa interfejsy umożliwiające wykorzystanie struktur drzewa do porządkowania dowolnych obiektów a także dokonywanie na nich operacji aktualizacji. Pierwszy interfejs, który powinien implementować każdy obiekt mający znaleźć się w indeksie, wynika bezpośrednio z założenia, iż rodzina R-drzew operuje na aproksymacjach obiektów w postaci brył brzegowych. Pozwala to na unifikację operacji wykonywanych na indeksowanych obiektach i uniezależnienie ich od reprezentacji przestrzennej obiektu założonej przez programistę przy projekcie systemu, który będzie korzystał z funkcjonalności opisywanego indeksu. Implementacja metody `float[][] getBounds()` interfejsu *Boundable* zapewnia dostarczenie takiej aproksymacji.

Programista powinien w ciele metody umieścić takie operacje by zwracana wartość była dwuwymiarową tablicą tablic wielowymiarowych, której pierwszy wiersz zawiera wartości minimalnej koordynaty (koordynaty o najmniejszych wartościach współrzędnych z każdego z dowolnie wielu wymiarów), natomiast drugi koordynatę maksymalną. Przestrzeganie takiej konwencji przedstawienia aproksymacji jest wymagane do prawidłowego działania struktur indeksu, które w trakcie wykonywania wszystkich operacji polegają na tej właśnie strukturze. Takie rozwiązanie umożliwia również przeprowadzenie wyszukiwania w indeksie wykorzystując, jako region poszukiwań obiekt będący elementem indeksu.

Drugi interfejs *UpdatableBoundable* wymaga od klasy obiektu implementacji dwóch metod `float[][] getOldBounds()` oraz `update()`. Nie jest on jednak obowiązkowy do poprawnej pracy indeksu, ułatwia jednak znacznie przeprowadzenie aktualizacji indeksu po dokonaniu zmian w formie przestrzennej indeksowanego obiektu. Jako że operacja ta wymaga dostarczenia aproksymacji stanu z momentu przed dokonaniem zmian, celowym wydaje się włączenie takiej informacji do obiektów, które w założeniu będą ulegać aktualizacjom (związanym ze zmianą lokalizacji w przestrzeni lub formy). Zwolnienie programisty z konieczności implementacji tego interfejsu wynika z faktu, iż nie wszystkie obiekty podlegające indeksowaniu mają charakter dynamiczny i nie wymagają dokonywania aktualizacji. Drugim powodem jest możliwość dostarczenia aproksymacji znajdującej się aktualnie w indeksie innymi środkami, dlatego też API przewiduje dwie metody przyjmujące jako argument pojedynczy obiekt *UpdatableBoundable* oraz trzy argumenty: identyfikator obiektu oraz dwa obiekty typu *Boundable* będące poprzednią oraz obecną aproksymacją aktualizowanego obiektu (w wersji jednoargumentowej identyfikatorem jest tożsamość obiektu)

Rozważałem także wprowadzenie trzeciego interfejsu umożliwiającego wykorzystywanie identyfikatorów złożonych porównywanych przez wartość co mogłoby okazać się użytecznym w niektórych systemach relacyjnych. Jednak brak możliwości zmuszenia programisty do przeciążenia operatora *equals()*, który dziedziczony z klasy *java.lang.Object* dokonuje porównania na podstawie tożsamości obiektów nie zaś na podstawie wartości składowych, spowodowała porzucenie tego pomysłu. Kluczowym wydaje się bowiem umożliwienie szybkiego dostępu do referencji lub kluczy (z dziedziny systemu korzystającego z indeksu, będących w nim wartościami niekluczowymi) o wartościach typu *int* lub *string*, które w realizacji struktury indeksu w przypadku zajścia takiej potrzeby (takiego porównania wymagają operacje aktualizacji oraz usuwania obiektów) porównuję w jednakowy sposób, metodą *equals()*.

5.3.6. Funkcjonalność

Aby spełnić sprecyzowane wcześniej wymagania, biorąc pod uwagę określone w poprzednim punkcie interfejsy API udostępnia następujące metody:

❖ Tworzenie indeksu

- ◆ Konstruktor *SpatialIndex (int M, int m, int dim, SplitVariant sv)*

W pełni specyfikuje parametry struktury. Maksymalną oraz minimalną ilość wpisów w węźle, ilość wymiarów indeksowanych danych, oraz rodzaj operacji podziału węzła.

- ◆ Konstruktor *SpatialIndex (int dim)*

Określa jedynie ilość wymiarów danych, pozostałe wartości ustawiane są na wartości domyślne struktury odpowiedzialnej za indeksowanie.

Indeks jest strukturą dynamiczną, niezakładającą znajomości danych w momencie jego tworzenia a jedynie ich wymiaru. W procesie tworzenia może wystąpić wyjątek *DimensionMismatch* w przypadku, gdy programista poda, jako ilość wymiarów wartość mniejszą niż dwa. Indeks nie przewiduje przetwarzania danych jednowymiarowych.

❖ Wstawianie danych do indeksu

- ◆ *insert (Boundable e)*

Dodaje do indeksu obiekt będący jednocześnie aproksymacją danych przestrzennych jak też identyfikatorem zwracany przez zapytania. Jest to Naturalne rozwiązanie dla systemów operujących na obiektach pochodzących ze środowiska Javy.

- ◆ *insert (Object oid, Boundable e)*

Ta wersja operacji wstawiania, przewiduje separację aproksymacji od identyfikatora, który może być referencją do obiektu (nie koniecznie przestrzennego) wartością atomową np. typu *int* lub *string*.

Metoda ta (podobnie jak kolejne operacje na strukturze indeksu) podnosi wyjątek w przypadku próby dodania do indeksu obiektu o z innego wymiaru, niż zadeklarowany w momencie tworzenia indeksu. Umożliwia jednak istnienie różnych typów obiektów w tym samym obiekcie, pod warunkiem, że pochodzą z tego samego wymiaru (np. wszystkie są trójwymiarowe).

- ❖ Aktualizacja

- ◆ *update (Object oid, Boundable o, Boundable n)*

Aby dokonać aktualizacji niezbędny jest identyfikator pozwalający wyłonić z pośród znalezionych (mogących posiadać identyczną reprezentację) obiekt, który ma ulec aktualizacji, jego przestrzenna reprezentacja istniejąca obecnie w indeksie, służąca ograniczeniu zakresu poszukiwań, oraz nowa reprezentacja adekwatna do obecnego stanu danych przestrzennych, która należy uwzględnić w indeksie.

- ◆ *update (UpdatableBoundable u)*

Obiektu implementujący interfejs *UpdatableBoundable* dostarcza wszystkich potrzebnych informacji, niezbędnych do przeprowadzenia jego aktualizacji w strukturze indeksu.

Przedstawiona operacja dotyczy aktualizacji przestrzennej warstwy obiektu, nie uwzględniam tu kwestii aktualizacji identyfikatorów obiektów, która szczególnie w przypadku identyfikatorów niebędących referencją mogłaby być uzasadniona. Skupiam się jednak na zastosowaniu indeksu dla celów systemów obiektowych, zorientowanych na identyfikację poprzez referencje. Aktualizacja odbywa się na zasadzie usunięcia i ponownego wstawienia wpisów odpowiadających danym przestrzennym. Zalety tego podejścia opisałem w rozdziale poświęconym R-drzewom.

- ❖ Usuwanie danych z indeksu

- ◆ *delete (Object oid, Boundable d)*

- ◆ *delete (Boundable d)*

Do usunięcia obiektu potrzebne są te same dane, które zostały użyte do wstawienia go do indeksu. Identyfikator oraz bryła brzegowa.

❖ Wyszukiwanie

◆ *search (Boundable s)*

Do wykonania zapytania zakresowego wystarczy region, z którym badana będzie relacja znajdujących się w indeksie obiektów. Ponieważ metoda przyjmuje argument typu *Boundable* operacje niezbędne do wyznaczania związków między regionem a obiektami mogą być symetryczne, np. zawiera – jest zawarty. Ponadto możliwe jest bezpośrednio badanie zależności między obiektami znajdującymi się w indeksie, bez potrzeby definiowania regionów wyszukiwania. Odpowiedzią na zapytanie jest stos rezultatów zawierający identyfikatory obiektów spełniających kryterium wyszukiwania.

❖ Wyświetlanie parametrów indeksu

◆ *toString()*

Oprócz parametrów podawanych podczas tworzenia indeksu (lub ustawionych na wartości domyślne) metoda informuje o:

- ◆ wysokości drzewa
- ◆ ilości węzłów w drzewie
- ◆ objętości przestrzennej indeksowanych danych
- ◆ ilości indeksowanych obiektów

5.3.7. Klasy testowe

Na potrzeby testów oraz prezentacji stworzone zostały dwie klasy testowe umożliwiające weryfikację funkcjonalności udostępnianej przez indeks jak również pomiar wydajności operacji wykonywanych na jego strukturach.

Pierwsza z nich *InternalTest* znajduje się w pakiecie *spatial.Rtree*. Korzystając z dostępu pakietowego operuje bezpośrednio na wewnętrznych strukturach drzewa. Korzystanie z niej jest możliwe po zapoznaniu się z hierarchią tworzenia wpisów reprezentujących dane przestrzenne, co mam nadzieję jest możliwe dzięki przedstawionym w tej pracy modelom UML oraz uwadze poświęconej starannemu dokumentowaniu kodu. Niemniej jest to klasa, której wykorzystanie zaleciłbym osobom mającym zamiar rozwijać bądź optymalizować struktury. Jest to odpowiednie miejsce na sprawdzenie wpływu wprowadzenia nowych funkcji podziału węzła lub debugowania algorytmów. Przynajmniej mi służyła głównie w tych celach. Po zaimplementowaniu API wróciłem do niej by sprawdzić wpływ dodatkowej warstwy na ogólną wydajność systemu.

Drugą klasą, umieszczoną na zewnątrz pakietów stanowiących bibliotekę, jest klasa *ExternalTest*, symulująca działanie systemu z punktu widzenia użytkownika końcowego. Korzysta wyłącznie z metod udostępnianych przez API, opisanych szerzej w poprzednim rozdziale. Importuje całość pakietu *spatial* zyskując w ten sposób dostęp do wszystkich niezbędnych komponentów, takich jak interfejsy czy definicja wyjątku, pozwalających na wykorzystanie struktury stojącej za fasadą API do indeksowania posiadanych danych przestrzennych.

Klasie tej towarzyszą definicje obiektów dwuwymiarowych: *Point*, *Circle*, *Rectangle*, *Square* oraz trójwymiarowy *Cube* w celu przedstawienia wad i zalet różnych rodzajów reprezentacji, oraz braku jakiegokolwiek wpływu tych różnic na zachowanie indeksu w przypadku prawidłowego zaimplementowania dostarczonych interfejsów. Klasy *Circle* oraz *Rectangle* implementują interfejs *UpdatableBoundable* dziedzicząc po *Boundable* w celu pokazania zalety podejścia do aktualizacji w sposób planowy, wymagający niewiele więcej wysiłku i zasobów przy specyfikacji obiektu, znacznie upraszczający korzystanie z niego w trakcie właściwego cyklu życiowego. Pozostałe klasy implementują interfejs *Boundable* pozwalający na umieszczenie ich reprezentacji w indeksie.

Ponieważ jest to biblioteka ogólnego zastosowania, należało stworzyć testowy interfejs, poprzez który użytkownik kontrolowałby stan obiektów w czasie wykonania. Pierwszym pomysłem było stworzenie graficznego interfejsu prezentującego dostępne dane, aproksymacje przechowywane przez indeks oraz ich wzajemne położenie, a także strukturę drzewa pokazującą utworzoną hierarchię. Ponieważ realizacja wymagań zebranych na ten moduł przekroczyłaby ramy czasowe projektu zwróciłem się ku interfejsowi bazującemu na konsoli oraz linii poleceń.

Pojawiła się w związku z tym idea sprawdzenia ile czasu może zaoszczędzić ponowne użycie komponentów dystrybuowanych na zasadach licencji GNU. Z doświadczenia wiem, iż zbudowanie parsera, leksera oraz podlegających im gramatyk jest zajęciem bardzo czasochłonnym. Po krótkim rozeznaniu wybrałem bibliotekę "Natural CLI" napisaną i przeznaczoną dla języka Java.

W trakcie prac programistycznych z tą biblioteką okazało się, iż posiada ona bardzo ograniczoną zdolność do budowania składni, a zastrzeżenia te nie zostały umieszczone w dokumentacji rozwiązania, która na pierwszy rzut oka wydawała się być dość solidna, aby zdecydować o wykorzystaniu tego rozwiązania w projekcie.

Wybrane rozwiązanie nie przetwarza znaków takich jak np. nawiasy "(" ")" przecinki "," czy kropki "." co więcej wymaga odstępu " " pomiędzy elementami składni, przez co niemożliwe jest sklejanie parametrów ze stałymi fragmentami wyrażenia. W zastosowaniu do interfejsu mającego symulować środowisko programistyczne jest to ogromnie niekorzystne. Ze względu na poświęcony czas na zapoznanie się z mechanizmem działania biblioteki oraz implementację podstawowych wyrażeń przy jej użyciu, nie zdecydowałem się na zmianę wykorzystanego rozwiązania. Wynikiem tego jest interfejs o niezbyt czytelnej składni, wymagający od użytkownika uwagi przy wprowadzaniu komend. Należy zwrócić uwagę, iż wszystkie polecenia różnią wielkość liter, natomiast separatorem dziesiętnym jest kropka.

Również sposób definiowania komend okazał się nie tak naturalny i przyjazny jak wydaje się to po przeanalizowaniu przykładów zawartych na stronie producenta. Efektem jest 600 linii ciężkiego kodu, którego tworzenie nie przynosiło satysfakcji a efekty są dalekie od zadowalających. Choć w porównaniu z biblioteką "JArgs" wybrane przeze mnie rozwiązanie wydawało się posiadać większą moc wyrazu, a w porównaniu z "Jakarta CLI" być prostsze w oprogramowaniu, wykorzystanie tego gotowego komponentu raczej zniechęciło mnie do tematu ponownego użycia. Obecnie uważam, że należy być szczególnie ostrożnym przy podejmowaniu takiej decyzji, zapoznać się dogłębnie z dokumentacją, oraz poświęcić odpowiednio dużo czasu na przetestowanie jego funkcjonalności by w zaawansowanym stadium prac z produktem nie doświadczyć przykrego rozczarowania, iż założenia, które przyjęliśmy nie mogą zostać wyrażone przy użyciu tego rozwiązania.

W związku z powyższym, załączam listę komend realizowanych przez interfejs konsoli. Jest ona dostępna również z poziomu konsoli, jednak z oczywistych względów jej forma może być mało czytelna, natomiast zapoznanie się z nią jest niewątpliwie kluczem do wykorzystania i oceny indeksu. Pomimo dużej liczby funkcji dostępnych obecnie za pośrednictwem linii poleceń nie jest to zestaw kompletny, reprezentuje jedynie część funkcjonalności API, choć umożliwia wykorzystanie wszystkich podstawowych funkcji indeksu, nie reprezentuje wszystkich ich wariantów.

```
create Index <nazwa:string> <wymiar:integer>
```

Tworzy indeks o podanej nazwie dla danych o określonym wymiarze.

```
create Index <nazwa:string> <wymiar:integer>  
<max:integer> <min:integer> <splitvariant:string>
```

Tworzy indeks dla danych o określonym wymiarze, określonej pojemności węzła i typie operacji split.

<index:string> insert <objecType:string> <id:integer>
Wstawia obiekt o podanym identyfikatorze do indeksu o podanej nazwie

<index:string> search <region:string> <id:integer>
Wyszukuje w indeksie o podanej nazwie obiektów będących w relacji podanym obiektem.

<index:string> delete <type:string> <id:integer>
Usuwa obiekt o podanym identyfikatorze z indeksu o podanej nazwie.

drop <index:string>
Usuwa indeks o podanej nazwie.

Index <name:string>
Wyświetla właściwości indeksu o podanej nazwie.

print <type:string>
Drukuje zawartość kontenera obiektów danego typu (Point, Circle, Saquare, Rectangle, Cube).

remove <type:string> <id:integer>
Usuwa obiekt o podanym identyfikatorze z kontenera danego typu.

new Point <x:float> <y:float>
Tworzy punkt o zadanych koordynatach.

Point <id:integer>
Zwraca punkt o zadanym identyfikatorze.

new Circle <x:double> <y:double> <r:double>
Tworzy okrąg o środku w zadanych koordynatach oraz promieniu r.

Circle <id:integer>
Zwraca okrąg o zadanym identyfikatorze.

Circle <id:integer> set center <x:double> <y:double>
Aktualizuje koordynaty środka okręgu.

Circle <id:integer> set radius <r:double>,
Aktualizuje koordynaty środka okręgu.

Circle <id:integer> update <indeks:string>
Aktualizuje wpis w indeksie dotyczący okręgu o danym identyfikatorze.

new Square <x:integer> <y:integer> <s:integer>
Tworzy kwadrat o lewym dolnym narożniku położonym w zadanych koordynatach oraz boku s.

Square <id:integer>
Zwraca kwadrat o zadanym identyfikatorze.

new Rectangle <xMin:float> <yMin:float> <xMax:float> <yMax:float>
Tworzy prostokąt o skrajnych narożnikach w podanych punktach.

Rectangle <id:integer>
Zwraca prostokąt o zadanym identyfikatorze.

```
Rectangle <id:integer> set min <x:float> <y:float>
```

Aktualizuje koordynaty lewego dolnego narożnika prostokąta.

```
Rectangle <id:integer> set max <x:float> <y:float>
```

Aktualizuje koordynaty prawego górnego narożnika prostokąta.

```
Rectangle <id:integer> update <indeks:string>
```

Aktualizuje wpis w indeksie o podanej nazwie dotyczący prostokąta o podanym identyfikatorze.

```
new Cube <x:float> <y:float> <z:float> <s:float>\n"+
```

Tworzy prostokąt o skrajnych narożnikach w podanych punktach.

```
Cube <id:integer>
```

Zwraca sześcian o zadanym identyfikatorze.

```
quit
```

Kończy działanie programu.

Aby uruchomić program należy w konsoli systemowej podać komendę:

```
java -jar ExternalTest.jar
```

Jak można zauważyć obsługiwane są operacje indeksu zakładające identyfikację obiektu przez referencję oraz aktualizacja obiektów implementujących interfejs *UpdatableBoundable*. Początkowo zakładałem mapowanie wszystkich poleceń dostępnych w API, jednak nakład kodu wymagany do przedstawienia komendy w "Natural CLI" oraz ilość operacji do wykonania przez użytkownika w tak nieprzyjnym środowisku aby z nich skorzystać, pokazała iż mija się to z celem, którym było przedstawienie zalet wykorzystania obiektów dostarczających wszystkie niezbędne informacje wymagane do przeprowadzenia operacji na indeksie w kontraście do operowania na informacjach pochodzących z danych dostarczanych samodzielnie przez użytkownika.

W związku z powyższym proponuję oprócz wykorzystania dostarczonej konsoli tekstowej, wykorzystanie biblioteki *spatial* w środowisku programistycznym. Przeprowadzenie prób w sposób statyczny, operując składnią Javy pokazuje pełne możliwości i łatwość wykorzystania przygotowanego przeze mnie indeksu. Zastosowanie pętli oraz odczytu informacji o obiektach z pliku pozwoli ocenić strukturę w kategoriach hurtowych, które ze względu na opisane utrudnienia, w wersji konsolowej są trudne do osiągnięcia.

6. Podsumowanie

Wynik pracy stanowi niniejsze opracowanie oraz działająca i przetestowana biblioteka indeksu przestrzennego oparta o strukturę R-drzewa umożliwiającą porządkowanie dowolnych obiektów przestrzennych dostarczającą aproksymację w postaci bryły brzegowej. W kategoriach przydatności dydaktycznej tego opracowania należy stwierdzić braki wynikające z niewielkiej ilości dostępnych powszechnie rzetelnych źródeł na temat zagadnienia indeksowania przestrzennego. Biorąc pod uwagę zakres problematyki, oraz brak doświadczenia w przygotowywaniu tego typu prac poglądowych da się zauważyć znaczne różnice na poziomie abstrakcji w opisach poszczególnych zagadnień. Chcąc by praca ta mogła służyć innym studentom, jako przystępna referencja z zakresu indeksów przestrzennych, nie ustrzegłem się wielu uproszczeń a także kolokwializmów, co może zmniejszać jej wartość akademicką. Bez wątplenia była to dla mnie lekcja, która mam nadzieję pozwoli w przyszłości rozpocząć pracę z innego pułapu oraz unikać popełnionych błędów.

6.1. Wnioski

W zakresie strategii indeksowania przestrzennego indeksowania danych, na podstawie publikacji, z którymi się zapoznałem i przedstawiłem w tej pracy, w mojej opinii dla większości zastosowań właściwszym podejściem jest budowanie struktur dynamicznych, o nieograniczonym regionie indeksu.

Implementacja wykazała, iż zwiększenie kosztu budowy i utrzymania drzewa przez zastosowanie wyrafinowanych technik podziału węzła zaczyna przynosić zysk w postaci skrócenia czasu wyszukiwania w momencie, gdy indeks zawiera ok. miliona dwuwymiarowych obiektów.

6.2. Perspektywy

System może stanowić podstawę do dalszych prac studenckich, jest na tyle wydajny, przetestowany i udokumentowany w stopniu pozwalającym myśleć o rozszerzaniu jego funkcjonalności o nowe rodzaje wyszukiwania oraz strategię podziału węzła. W związku z niedoskonałą formą prezentacji, zamierzam w najbliższym czasie przygotować graficzny interfejs użytkownika, który z pewnością zwracałby większą uwagę na oraz czynił bardziej dostępnym, ciekawy temat porządkowania przestrzeni z wykorzystaniem indeksów.

7. Bibliografia

- [1] **„ Quad trees a data structure for retrieval on composite keys ”**
R. A. Finkel, J. L. Bentley, Acta Informatica Volume 4, Springer Berlin, 1974 r.
- [2] **„ Worst-case analysis for region searches in multidimensional balanced quad trees ”**
D. T. Lee, C. K. Wong, Acta Informatica Volume 9, Springer Berlin, 1977 r.
- [3] **„ R-Trees - A Dynamic Index Structure for Spatial Searching ”**
Antomn Guttman, ACM, 1984 r.
- [4] **„ R-trees: Theory and Applications ”**
Y. Manolopoulos, A. Nanopoulos, Springer, Berlin, 2005 r.
- [5] **MSDN Library**
BB895269
- [6] **„ The X-tree: An Index Structure for High-Dimensional Data ”**
Stefan Berchtold, Daniel A. Keim, Hans-Peter Kriegei
Institute for Computer Science, University of Munich, 1996 r.
- [7] **„ A fast Algorithm for Indexing, Data-Mining and Visualization
of Traditional and Multimedia Datasets ”**
SIGMOD Int. Conf. on Management of Data, San Jose, CA, 1995 r.
- [8] **„ Foundations of Multidimensional and Metric Data Structures ”**
Hanan Samet, University of Maryland, 2006 r.
- [10] **„ An Algorithm for Finding Best Matches in Logarithmic Expected Time ”**
J. H. Friedman, J. L. Bentley, R. A. Finkel ACM Trans on Mathematical Software, 1977 r.
- [11] **„ Pro Oracle Spatial”**
Ravi Kothuri, Albert Godfind, Euro Beinat, Apress, 2004
- [12] **„ The R*-tree: An Efficient and Robust Access Method for Points and Rectangles+ ”**
Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger
Praktuche Informatlk, Umversltaet Bremen, West Germany, 1990 r.