R-дерево – динамическая структура указателей, для пространственного поиска

Antonin Guttman, University of California, Berkeley

Это исследование было профинансировано Национальным научным фондом, грант ECS-8300463 и Управлением научных исследований ВВС, грант AFOSR-83-0254

Аннотация

Для того чтобы эффективно справиться с пространственными данными, в соответствии с требованиями систем автоматизированного проектирования (САПР) и геоинформационных приложений (ГИС), система баз данных нуждается в механизме индексирования, который поможет быстро получить эти объекты данных, в соответствии с их пространственным расположением. Однако традиционные методы индексирования не очень хорошо подходят для объектов данных не нулевой размерности, расположенных в многомерном пространстве. В этой статье мы описываем динамическую структуру указателей (индекс) под названием R-дерево, которое удовлетворяет эту потребность, и дает алгоритмы для поиска и обновления в нём. Мы представляем результаты серии тестов, которые показывают, что структура работает хорошо, и заключаем, что она полезна для текущих систем баз данных в геоинформационных приложениях.

1. Введение

Пространственные объекты данных часто охватывают районы в многомерном пространстве и не очень хорошо представлены точечным положением. Например, объекты на карте, такие как административные районы, участки переписи и т.д. занимают регионы не нулевого размера в двух измерениях. Обычной операцией над пространственными данными, является поиск всех объектов в некой области, например, чтобы найти все районы, которые имеют землю в радиусе 20 миль от конкретной точки. Этот вид пространственного поиска происходит часто в системах автоматического проектирования и приложений геоданных, и поэтому важно, иметь возможность для эффективного извлечения объектов в соответствии с их пространственным расположением.

Индекс, основанный на объектах пространственного расположения желателен, но классические структуры индексации одномерных баз данных не подходят для многомерного пространственного поиска. Структуры на основе точного соответствия значений, таких как хэш-таблицы, не являются полезными, потому что требуется диапазон поиска. Структуры с использованием одномерно упорядоченных ключевых значений, такие как B-деревья и ISAM индексы, не работают, потому что пространство поиска многомерно.

Ряд структур были предложены для работы с данными многомерных точек, и обзор методов можно найти в [5] метод Ячеек [4,8,16] плохо подходит для динамических структур, потому что границы ячеек должны быть определены заранее в Квадро-деревьях [7] и k-D деревьях [3] не принимают подкачки вторичной памяти во внимание. K-D-B деревья [13] предназначены для выгружаемой памяти, но полезны только для точечных данных. Использование индексированных интервалов было предложено в [15], но этот метод не может быть использован в нескольких измерениях. Угловая строчка [12] является примером структуры для двумерного поиска, подходит для объектов данных ненулевой размерности, но он предполагает однородную первичную память и не является эффективным для случайных поисков в очень больших объемах пространственных данных. Файлы сетки [10] обрабатывают данные неточечным путем сопоставления каждого объекта до точки в многомерном пространстве. В этой статье мы описываем альтернативную структуру, называющуюся R-дерево, которое представляет объекты данных интервалами в нескольких измерениях

Раздел 2 описывает структуру R-дерева и раздел 3 дает алгоритмы для поиска, вставки, удаления и обновления. Результаты деревьев в тестах производительности, представлены в разделе 4. Раздел 5 содержит резюме наших выводов

2. R-дерево, структура указателей

R-дерево сбалансированное по высоте дерево, похожее на B-дерево [2, 6] с индексированными записями в своих листьях, содержащие указатели на объекты данных. Узлы соответствуют дисковым страницам, если индекс диск-резидент, и структура сконструирована таким образом, что пространственный поиск требует посещения лишь небольшого числа узлов. Индекс является полностью динамическим; вставки и удаления можно совмещать с поиском и никакая периодическая реорганизация не требуется.

Пространственная база данных состоит из набора кортежей, представляющих пространственные объекты, и каждый кортеж имеет уникальный идентификатор, который может использоваться для получения конечных узлов в R-дереве, содержат индексированную запись вида

$$(I\ ,\quad \text{Кортеж} - \text{идентификатор})$$

где кортеж-идентификатор относится к кортежу в базе, и I это n-мерный прямоугольник, который является ограничивающим прямоугольник индексированного пространственного объекта

$$I = (I_0\ , I_1\ , \dots\ , I_{n-1})$$

здесь n это число измерений и I_i замкнутый ограниченный интервал [a, b] описывающий протяжённость объекта в измерении i. Альтернативно I_i может иметь одну или оби конечных точки равных бесконечности, что указывает на объект протяжённостью до бесконечности не-листовые узлы содержать записи вида

$$(I\ ,\quad \text{Указатель} - \text{потомок})$$

где указатель-потомок это адрес нижнего узла в R-дереве и I охватывает все прямоугольники примитивов всех ниже лежащих узлов.

Пусть M будет максимальным числом примитивов, которые могут содержатся, в одном узле и пусть $m \leq \frac{M}{2}$ будет параметром определяющий минимальное число примитивов в узле. R-дерево удовлетворяет следующие свойства:

1. Каждый лист-узел содержит от m до M индексированных записей, за исключением корня
2. Для каждой индексированной записи $(I\ ,\ \text{Кортеж} - \text{идентификатор})$ в листовом узле, I это минимальный прямоугольник который пространственно содержит n-мерные объекты данных представленные указанным кортежем
3. Каждый не листовой узел содержит от m до M потомков, за исключением корня
4. Для каждого элемента $(I\ ,\ \text{Указатель} - \text{потомок})$, в не листовом узле, I это минимальный прямоугольник который пространственно содержит прямоугольники узлов потомков
5. Корневой узел имеет не менее двух потомков, если он не является листом
6. Все листья появляются на том же уровне

На рисунке 2.1a и 2.1b показана структура R-дерева и проиллюстрированы отношения локализации и пересечения, которые могут существовать между ее прямоугольниками

Высота R-дерева, содержащее N индексированных записей, не более $|\log_m N| - 1$, потому что коэффициент ветвления каждого узла не меньше m. Максимальное число узлов $\left|\frac{N}{m}\right| + \left|\frac{N}{m^2}\right| + \dots + 1$. Наихудший случай использования пространства для всех узлов, кроме корня - $\frac{m}{M}$. Узлы будут иметь тенденцию иметь примитивов более m, и это будет уменьшать высоту дерева и улучшать использование пространства. Если узлы имеют более чем 3 или 4 записи - дерево будет очень широким, и почти все пространство используется для листьев, содержащих индексированные записи. Параметр m может изменяться в рамках оптимизации производительности, и различные значения проверяются экспериментально в разделе 4.

3. Поиск и обновление

3.1. Поиск

Алгоритм поиска спускается по дереву от корня, аналогично для B-дерева. Однако, возможно потребуется посетить более одного поддерева узла, для поиска, следовательно, это не позволяет, гарантировать хорошую производительность, в наихудшем случае. Тем не менее, с большинством видов данных, алгоритмы обновления будут поддерживать дерево в форме, которая позволяет алгоритму поиска, игнорировать ненужные регионы индексированного пространства, и рассматривать только данные вблизи области поиска.

Далее мы будем обозначать прямоугольник элемента *E* как *EI*, и элементы кортеж-идентификатор или указатель-потомок как *Ep*.

Алгоритм **Поиск**. Дано R-дерево, корневой узел *T*, найти все записи, в которых прямоугольник перекрывается прямоугольником поиска *S*.

П1 [Поиск поддерева] Если *T* не лист, проверить каждый элемент *E* на пересечение *EI* с *S*. Для всех совпавших элементов, вызвать **Поиск** для дерева, корнем которого является узел, на который указывает *Ep*.
П2 [Поиск в листе] Если *T* лист, проверить все примитивы *E* на пересечение *EI* с *S*. Если это так, то E нужная запись

3.2. Вставка

Вставка индексированных записей для новых кортежей данных, аналогична вставке в B-дерева в том, что новые индексированные записи добавляются на листья, при переполнении узла, происходит разделение, и раскол распространяются вверх по дереву.

Алгоритм **Вставка**. Вставка новой индексированной записи *E* в R-дерево

В1 [Поиск позиции для новой записи] Вызов **ВыборЛиста** для выбора листа *L* в который следует поместить *E*.

В2 [Добавление записи в лист] Если *L* имеет место для нового примитива, добавляем *E*. В противном случае вызываем **РазделениеУзла** для получения *L* и *LL*, содержащие *E* и все старые записи из *L*.

В3 [Распространить изменения вверх] Вызвать **ВыравниваниеДерева** на *L*, также передать *LL*, если произошло разделение

В4 [Наращивание дерева] Если разделение узла распространилось до корневого узла, создать новый корневой узел, потомками которого будут два полученных узла

Алгоритм **ВыборЛиста**. Выбор листа, в который будет добавлена новая индексированная запись *E*.

ВЛ1 [Инициализация] Пусть *N* будет корневым узлом
ВЛ2 [Проверка листа] Если *N* лист, возвращаем *N*
ВЛ3 [Выбор поддерева] Если *N* не лист, пусть *F* будет примитивом в *N*, чей прямоугольник *FI* нуждается в меньшем увеличении, для включения *EI*. Выбираем тот узел, для которого увеличение площади прямоугольника будет минимальной.
ВЛ4 [Спуск до листьев] Установка в *N* дочернего узла выбранного на предыдущем шаге и переход к шагу ВЛ2

Алгоритм **ВыравниваниеДерева**. Продвигаясь от листа *L* к корню, регулируя охватывающий прямоугольник и распространяя разделение узлов, если необходимо

ВД1 [Инициализация] Пусть $N=L$. Если L был разделён ранее, установим в NN полученный второй узел.
ВД2 [Проверка если все сделано] Если N является корнем, остановить
ВД3 [Выравнивание охватывающего прямоугольника в родительском узле] Пусть P будет родительским узлом N, и пусть E_N будет следующим узлом в P. Выравниваем E_NI так, чтобы он плотно охватывал все примитивы из N.
ВД4 [Продвижение разделение узла вверх] Если у N есть партнёр NN полученный из раннего разделения, создаём новый элемент E_{NN} с $E_{NN}p$ указывающий на NN и $E_{NN}I$ включающий все прямоугольники из NN. Добавляем E_{NN} к P, если там есть место. Иначе, вызываем **РазделениеУзла** для получения P и PP содержащих E_{NN} и все примитивы из старого P.
ВД5 [Движемся вверх, к следующему уровню] Пусть $N=P$ и $NN=PP$ если произошло разделение. Повторить с шага ВД2.

Алгоритм **РазделениеУзла** описан в разделе 3.5

3.3. Удаление

Алгоритм **Удаление**. Удаляем индексированную запись E из R-дерево
У1 [Ищем узел содержащий запись] Вызываем **ПоискЛиста**, чтобы найти лист L содержащий E. Останавливаемся, если запись не найдена.
У2 [Удаление записи] Удаляем E из L
У3 [Распространение изменений] Вызываем **УплотнениеДерева** для L.
У4 [Сокращение дерева] Если корневой узел имеет лишь одного потомка, после выравнивания дерева, делаем этого потомка новым корнем.

Алгоритм **ПоискЛиста**. Данно R-дерево, корневой узел T, найдём лист содержащий примитив E.
ПЛ1 [Поиск субдерева] Если T не лист, проверяем каждый элемент F в T на пересечение FI с EI. Для каждой такой записи вызываем **ПоискЛиста** для дерева, на корень которого указывает F_p пока E не будет найден или все элементы проверены
ПЛ2 [Исследуем лист в поиске примитива] Если T лист, проверяем каждый примитив на соответствие E, если E найден – возвращаем T

Алгоритм **УплотнениеДерева**. Дан лист L из которого была удалена запись, устранить узел, если в нём слишком мало примитивов и переместить её элементы. Распространить ликвидацию вверх по мере необходимости. Отрегулируйте все охватывающие прямоугольники на пути к корню, делая их меньше возможности.

УД1 [Инициализация] Пусть $N=L$, Пусть Q, множество удаляемых узлов, которые должны быть пустыми
УД2 [Поиск родительского элемента] Если N корень, идём к УД6. В противном случае пусть P предок N, и пусть E_N n-ая запись в P
УД3 [Удаление полу-полных потомков] Если N имеет меньше m примитивов, удаляем E_N из P и добавление N к множеству Q
УД4 [Выравнивание охватывающего прямоугольника] Если N не был удалён, выравниваем E_NI, что бы он плотно охватывал все элементы в N
УД5 [Движемся на выше лежащий уровень дерева] Пусть $N=P$ и повторяем с шага УД2
УД6 [Повторная вставка осиротевших элементов] Повторно вставляем все узлы из множества Q. Элементы из удалённого листа повторно вставляются в листья дерева как описано в алгоритме **Вставка**, но элементы из вышележащих узлов должны быть размещенные выше в дереве, так что бы листья подчинённых суб-деревьев были на том же уровне, что и листья остального дерева.

Процедура, описанная выше, для утилизации недостаточно полных узлов, отличается от соответствующей операции для B-дерева, в котором, два или более смежных узла сливаются. Подход подобный B-дерева, возможен для R-деревьев, хотя нет смежности в смысле B-дерева:

почти полный узел может быть объединен с соседом, если его площадь увеличится минимально, или осиротевшие записи могут быть распределены среди родственных узлов. Любой метод может быть вызван для разделения узлов. Вместо этого, мы выбрали повторную вставку, по двум причинам: во-первых, делает то же самое и проще в реализации, потому что может быть использована процедура **Вставить**. Эффективность должна быть сопоставима, потому что страницы, необходимые во время повторного включения, как правило, будут те же, что посещались в течение предыдущего поиска и уже будут в памяти. Вторая причина заключается в том, что повторная вставка постепенно уточняет пространственную структуру дерева, и предотвращает её постепенное ухудшение, что может произойти, если каждая запись находится постоянно в том же родительском узле.

3.4. Обновление и другие операции

Если данные кортежей обновляются, так что его огибающий прямоугольник изменяется, его индексированная запись должна быть удалена, обновлена, и затем повторно вставлена так, что будет найден путь к его правильному месту в дереве.

Другие типы поиска, кроме описанного выше, могут быть полезно, например, чтобы найти все объекты данных, полностью содержащиеся в области поиска, или все объекты, которые содержат область поиска. Эти операции могут быть реализованы через простые изменения данного алгоритма. Поиск конкретной записи, чьё описание известно, требуется алгоритмом удаления и реализуется в алгоритме **ПоискЛиста**. Вариант удаления диапазона, в котором удаляются индексированные записи для всех объектов данных, в той или иной области, также хорошо поддерживается R-деревом.

3.5. Разделение узлов

Для того чтобы добавить новую запись в полный узел, содержащий M записей, необходимо разделить коллекцию $M+1$ записей, между двумя узлами. Разделение должно быть сделано таким образом, что бы сделать маловероятной возможность, что бы оба новых узла понадобилось посещать при последующих поисках. Поскольку решение о посещении узла, зависит от перекрытия его огибающих прямоугольников с областью поиска, общая площадь двух огибающих прямоугольников, после раскола, должна быть сведена к минимуму. Рисунок 3.1 иллюстрирует этот момент. Площадь покрытия прямоугольников в "плохом разделении" гораздо больше, чем в "хорошем разделении".

Этот же критерий был использован в процедуре **ВыборЛиста** для решения, где вставить новую индексированную запись, на каждом уровне в дереве, выбирается то поддерево, чей огибающий прямоугольник должен будет минимально увеличен.

Обратимся теперь, к алгоритму для разделения набора $M+1$ записей на две группы, по одному для каждого нового узла.

3.5.1. Исчерпывающий алгоритм

Самый простой способ найти разделение узлов с минимальной площадью, является создание всех возможных группировок и выбрать лучшее. Тем не менее, число возможностей примерно 2^{M-1} и разумных значениях M в 50*, количество возможных разделений, слишком большое. Мы реализовали модифицированную форму исчерпывающего алгоритма для использования в качестве стандарта, для сравнения с другими алгоритмами, но это было слишком медленно, для использования с большим размеров узлов.

* Прямоугольник в 2 мерном пространстве может быть представлен 4-мя числами из четырёх байт каждое. Если указатель также занимает 4 байта, каждый примитив требует 20 байт. Страница памяти в 1024 байт будет хранить примерно 50 примитивов.

3.5.2. Алгоритм квадратной скорости

Этот алгоритм пытается найти разделение с минимальной площадью, но не гарантирует нахождение минимально возможной области. Стоимость квадратичная M и линейна по числу измерений. Алгоритм выбирает две из $M+1$ записей, это должны быть первые элементы двух новых групп, выбирая пару, которая бы создала наибольшую площадь, если бы обе были

помещены в одну группу, и площадь прямоугольника, охватывающего обе записи, минус площади самих записей, будет максимальной. Остальные элементы, затем распределяются по группам по одному за раз. На каждом шаге расширения области необходимо рассчитать добавление всех оставшихся записей в каждой группе, а также распределение примитива показывает самое большое различие между этими двумя группами.

Алгоритм **КвадратногоРазделения**. Разделение множества *M+1* индексированных записей на две группы

КР1 [Выбираем первый примитив для каждой группы] Применить алгоритм **ВыборСемян** для получения двух записей, которые будут первыми элементами групп. Распределить их по группам.
КР2 [Проверка если все сделано] Если все записи были распределены – остановить. Если одна группа содержит слишком мало записей, все остальные должны быть, направленны в неё, для того, чтобы иметь минимальное число m, назначить их и остановить
КР3 [Выбор записи для распределения] Вызвать алгоритм **ВыбратьСледующую** для выбора следующей записи для распределения. Добавьте её в группу, чей охватывающий прямоугольник будет минимально увеличен при её размещении. Выбрать добавить запись в группы с меньшей площадью, либо, с меньшим количеством записей. Повторить от КР2

Алгоритм **ВыборСемян**. Выбор двух примитивов, которые будут первыми элементами двух групп

ВСн1 [Рассчитать неэффективность группировки записей вместе] Для всех пар примитивов E_1 и E_2 создать прямоугольник J включающий E_1I и E_2I. Рассчитайте $d = \text{Площадь}(J) - \text{Площадь}(E_1I) - \text{Площадь}(E_2I)$
ВСн2 [Выберите наиболее расточительную пару] Выберите пару с наибольшим d

Алгоритм **ВыбратьСледующую**. Выбираем одну оставшуюся запись для классификации в группы.

ВС1 [Определить стоимость вставки каждой записи в каждую группу] Для каждого примитива E, который еще не в группе, вычислить d_1= рост площади, необходимый для охватывающего прямоугольника группы 1 при включении EI. Рассчитать d_2 аналогично для группы 2
ВС2 [Найти запись с наибольшим предпочтением одной группы] Выберите любую запись с максимальной разницей между d_1 и d_2

3.5.3. Алгоритм линейной сложности

Этот алгоритм линеен для M и числа измерений. **ЛинейноеРазделение** идентичен **КвадратногоРазделения**, но использует другую версию **ВыборСемян**. **ВыбратьСледующую** просто выбирает любую из оставшихся записей.

Алгоритм **ЛинейныйВыборСемян**. Выбор двух примитивов, которые будут первыми элементами двух групп

ЛВС1 [Найти крайние прямоугольники вдоль всех измерений] Вдоль каждого измерения, найти запись, чей прямоугольник имеет самую большую короткую, и один с минимальной длинной стороной. Записать разделение
ЛВС2 [Выровнять по форме прямоугольники кластеров] Осуществить нормировку разделения путем деления на ширину всей совокупности вдоль соответствующего размера
ЛВС3 [Выберите самую крайнюю пару] Выберите пару с наибольшим нормированным разделением по любой размерности

4. Тест производительности

Мы реализовали R-дерево на C под Unix на компьютере Vax 11/780, и использовали нашу реализацию в серии тестов производительности, целью которых было проверить практичность структуры, чтобы выбрать значения для *M* и *m*, и оценить разные алгоритмы разделения узлов. В этом разделе представлены результаты.

Мы попробовали пять размеров страницы, соответствующие различным значениям *M*.

Байтов на страницу	Максимальное количество примитивов на страницу (*M*)
128	6
256	12
512	25
1024	50
2048	102

Значения опробованные для *m*, минимальное количество записей в узле, были *M*/2, *M*/3, и 2. Три алгоритма разделения узлов, описанные ранее, были реализованы в различных версиях программы. Во всех наших тестах используются двумерные данные, хотя структура и алгоритмы работы подходят для любого числа измерений.

Во время первой части каждого теста, запускается программа для чтения геометрии данных из файлов и построения дерева-индекса, начиная с пустого дерева и вызывая **Вставку** для каждой новой индексированной записи, производительность **Вставки** измеряли для последних 10% записей, когда дерево приобретает почти его окончательный размер. На втором этапе, программа вызывает функцию **Поиск** с поисковыми прямоугольниками, сделанными с использованием случайных чисел, 100 результатов было выполнено за тест, каждый извлекал около 5% данных. Наконец, программа читает входящие файлы второй раз и вызывает функцию **Удаление**, чтобы удалить индексированную запись для каждого десятого элемента данных, так что измерения проводились для рассеянного удаления 10% от индексированых записей. Испытания проводились с использованием сверхбольших интегральных схемы (СБИС) размещения данных от RISC-II компьютерный чип [11] Схема ячеек CENTRAL, содержащий 1057 прямоугольников, была использована в тестах и показана на рисунке 4.1.

Рисунок 4.2 показывает стоимость во времени ЦП вставки последних 10% записей, как функция размера страницы. Исчерпывающий алгоритм, стоимость которого растет экспоненциально с размером страницы, видно, очень медленный при увеличении размера страницы. Линейный алгоритм является самым быстрым, как и ожидалось. С этим алгоритмом, процессорное время едва увеличивается с размером страницы, что свидетельствует о том, что разделение узла отвечает лишь за небольшую часть стоимости вставки записей. Уменьшение стоимости вставки с более строгим требованием баланса узла отражает тот факт, что, когда одна группа становится слишком полной, все алгоритмы разделения просто помещают следующий элемент в другую группу без дальнейших сравнений. Стоимость удаления элемента из индекса, показано на рисунке 4.3, сильно зависит от минимального требования заполнения узла. Когда узлы становятся полу-полными, их примитивы должны быть повторно вставлены и повторная вставка иногда вызывает разделение узлов. Более строгие требования заполнения узлов приводит к тому, что узлы становится полу-полными чаще, и с большим количеством записей. Кроме того, разделения становится более частыми, поскольку узлы, как правило, полнее. Кривые грубы, потому что исключение узлов происходят случайно и редко, их было слишком мало в наших тестах, чтобы сгладить колебания.

На рисунках 4.4 и 4.5 показано, что производительность поиска по индексу очень зависит от использования различных алгоритмов разделения узлов и требований заполнения. Исчерпывающий алгоритм производит несколько лучшую структуру индекса, в результате чего меньше страниц просмотрено и меньше стоимость ЦП, но большинство комбинации алгоритма и требований заполнения дают 10% от лучших. Все алгоритмы обеспечивают разумную производительность.

На рисунке 4.6 показана память, занятая индексным деревом в зависимости от алгоритма, требований заполнения и размер страницы Обычно результаты подтверждают наше

ожидание, что более строгие требований заполнения узлов производят меньший индекс. Менее плотный индекс потребляет приблизительно 50% больше места, чем наиболее плотный, но все результаты для l/2-заполненности и l/3-заполненности (не показан) в пределах 15% друг от друга.

Вторая серия испытаний измеряется производительность R-дерева в зависимости от количества данных в индексе. В той же последовательности тестовых операций, как и раньше, проводили на образцах содержащих 1057, 2236, 3295, и 4559 прямоугольников. Первый образец содержит данные компоновки со схемой ячеек, используемой ранее CENTRAL, а второй состоял из компоновки с аналогичной, но большим количеством ячеек, содержащей 2238 прямоугольников. Третий образец был сделан с использованием обоих CENTRAL и болшей схемы, с двумя ячейками эффективно размещенных друг над другом. Три ячейки были объединены, чтобы составить последний образец. Поскольку образцы были составлены по-разному, используя различные данные, результаты деятельности не идеально масштабируются, а с некоторыми неровностями, что и следовало ожидать.

Две комбинации алгоритма разделения и заполнения узлов, которые были выбраны для тестов – это линейный алгоритм с m=2, и квадратный алгоритм с m=M/3, оба с размером страницы в 1024 байта (M =50).

Рисунок 4.7 показывает результаты тестов, определяющих зависимость производительности вставки и удаления от размера дерева. Обе конфигурации тестов формируют деревья с двумя уровнями для 1057 записей и три уровня для другого размера примера. Из рисунка видно, что стоимость вставками с квадратичным алгоритмом является почти постоянной, за исключением случаев, когда дерево растет в высоту. Там кривая показывает определенный прыжок из-за увеличения числа уровней, где может произойти раскол. Линейный алгоритм не показывает прыжок, указывая снова, что линейное разделение узлов составляют лишь небольшую часть стоимости вставками.

В течение теста деления с линейной конфигурацией, не произошло разделений узлов, из-за широких требований заполнения узла и небольшого количества элементов данных. В результате кривая показывает только небольшой скачок, когда число уровней дерева возрастает. При удалении с квадратичной конфигурацией производится только от 1 до 6 разделений узлов и в результате кривая очень грубая. При учете изменений из-за небольшого размера выборки, тесты показывают, что стоимость вставки и удаления не зависит от ширины дерева, но зависит от высоты дерева, которая растет медленно с числом элементов данных.

Рисунки 4.8 и 4.9 подтверждают, что эти две конфигурации, имеют почти одинаковую производительность поиска. Каждый поисковой запрос, извлекает от 3% до 6% данных. Нисходящий тренд кривых, следовало ожидать, потому что стоимость обработки более высоких узлов дерева, становится менее значительной, поскольку количество данных, полученных в каждом поисковом запросе - увеличивается. Увеличение числа уровней деревьев удерживает расходы от падения между первой и второй точками данных. Низкая ЦП стоимость в отборе записи, менее чем в 150 микросекунд – для больших объемов данных, показывает, что индекс является достаточно эффективным в сокращении поиска малых поддеревьев.

Прямые линии на рисунке 4.10 отражают тот факт, что почти всё пространство в индексе R-дерева, используется для конечных узлов, число которых изменяется линейно с количеством данных. Для тестовой конфигурации Линейно-2 общий объём, занимаемый R-деревом составлял около 40 байт на элемент данных, по сравнению с 20 байт за единицу, для отдельных индексированных записей. Соответствующий показатель для конфигурации квадратичной - 1/3 был 33 байт на элемент.

5. Выводы

Было показано, что структура R-дерева, может быть полезна для индексации объектов пространственных данных, которые имеют не нулевую размерность. Узлы, соответствующие дисковым страницам, разумного размера (например 1024 байта) имеют значения M, которые дают хорошую производительность. При меньших узлах, структура также должна быть эффективной как индекс основной памяти, производительность процессора будет сопоставима, но не было бы никакой стоимости ввода/вывода.

Линейный алгоритм разделения узлов оказался хорош, так как был быстрее более дорогих методов, и немного худшее качество разделения не влияет заметно на производительность поиска.

Предварительные исследования показывают, что R-деревья было бы легко добавить в любую систему реляционную базу данных, которая поддерживает обычные методы доступа (например, INGRES [9], System-R [1]). Кроме того, новая структура будет работать особенно хорошо в сочетании с абстрактным типом данных и абстрактными индексами [14], чтобы упростить обработку пространственных данных.