процедуру уменьшения перекрытия узлов, то при вставке нового элемента в некоторый узел дерева может произойти значительное увеличение *MBR* этого узла. Любое увеличение *MBR* является нежелательным, так как потенциально является источником увеличения площади перекрытия разных областей. Очень хорошо данную проблему описали А. Гуттман и Д. Греен в своих работах [36, 38]. Позднее было предложено ряд мер, уменьшающих перекрытие областей [38]. Главной их особенностью является добавление в процедуру вставки некоторой модификации, позволяющей анализировать ситуацию и выбирать для вставки тот элемент, при помещении объекта в который увеличение процента перекрытия будет незначительным.

Рано или поздно вставка новых элементов в дерево приводит к переполнению некоторого узла (число элементов узла становится максимальным, и добавление нового объекта в него становиться не возможным). При этом проводится процедура расщепления переполненного узла на два новых. Процедура расщепления, в случае своей неэффективной работы, также может являться источником перекрывающихся областей.

Большинство алгоритмов данного подраздела по своей структуре очень похожи друг на друга и отличаются только алгоритмами вставки и удаления узлов. Практически все они ведут свое существование от метода, получившего название *R*-дерево [36].

Далее в подразделах данной главы будет описан оригинальный алгоритм *R*-дерева, а также ряд его модификаций, улучшающих те или иные его свойства.

3.2.1. R-дерево

R-дерево – (*R-Tree*) это индексная структура для доступа к пространственным данным, предложенная А. Гуттманом (Калифорнийский университет, Беркли) в 1984 году [36]. *R*-дерево допускает произвольное выполнение операций добавления, удаления и поиска данных без периодической переиндексации. При этом дерево получается сбалансированным, что является одним из важных свойств любой иерархической структуры данных.

Далее в данном параграфе будет описана структура дерева и основные алгоритмы. Также будут даны практические рекомендации по выбору тех или иных параметров дерева.

Структура R-дерева

R-дерево – это сбалансированное по высоте дерево, сходное с *B*+-деревом, листовые узлы которого содержат ссылки на конечные объекты. Если индексная структура находится на жестком диске, то каждый узел соответствует дисковой странице. Структура разработана так, чтобы для пространственного поиска требовалось посещение как можно меньшего числа узлов. Индексная структура полностью динамическая – добавление и удаление может выполняться одновременно с поиском, и никакой периодической реорганизации структуры производить не нужно.

Для организации такой индексной структуры используют пространственную базу данных, состоящую из набора записей, каждой из которых соответствует некоторый уникальный идентификатор. Этот идентификатор используют как средство ссылки на запись из индекса. В качестве идентификатора может выступать некоторое уникальное число или номер записи в файле (второй вариант предпочтительнее, так как работает быстрее, однако для него присущи некоторые недостатки, связанные с удалением записей из файла).

Если принять описанные условия, то каждый листовой узел дерева будет состоять из элементов, имеющих вид:

$$[MBR,\ идентификатор\_записи],$$

где *идентификатор_записи* ссылается на запись в БД, а *MBR* – это *n*-мерный прямоугольник, который является минимальным охватывающим прямоугольником для пространственного объекта, со сторонами параллельными осям координат. Обычно *MBR* задают в виде:

$$MBR = \{I_1, I_2, \ldots, I_n\},$$

где n – это число размерностей, а I_i – это интервал с закрытыми концами $[a,b]$, характеризующий размер объекта по соответствующей оси координат i. В принципе, I_i может иметь в качестве конечной точки $\pm\infty$. При этом предполагается, что объект по такому измерению

распространяется бесконечно.

Внутренние узлы дерева содержат элементы, имеющие похожую структуру:

$$[MBR, \text{ссылка\_на\_потомка}],$$

где *ссылка_на_потомка* – это адрес узла низшего уровня в *R*-дереве (дочернего по отношению к данному), все записи внутри которого покрываются прямоугольником *MBR*.

Нетрудно заметить, что и листовые узлы, и внутренние представляют собой набор из элементов описанной структуры. Причем даже в простейшей реализации таких элементов должно быть больше одного. Обозначим за M – максимальное число элементов в любом узле, а m – минимальное. Причем для реализации основных алгоритмов необходимо выполнение условия $m \le M/2$. Практические рекомендации по выбору конкретных значений M и m будут приведены ниже.

R-дерево, описанное выше, должно удовлетворять следующим требованиям.

1. Каждый узел дерева содержит не меньше m и не больше M записей. Исключение может составлять только корень.

2. Корень, если он не является листом, содержит как минимум двух потомков. Максимальное количество элементов в корне также ограничивается значением M.

3. Для каждой индексной записи листового узла *MBR* является минимальным прямоугольником, который полностью вмещает в себя пространственный объект, на который ссылается запись.

4. Для каждой индексной записи внутреннего узла дерева *MBR* является минимальным прямоугольником, охватывающим все *MBR* дочерних узлов.

5. Все листовые узлы дерева расположены на одном уровне (дерево является сбалансированным).

6. Каждый объект упоминается в дереве ровно один раз.

На рис. 3.9. показан пример структуры *R*-дерева и проиллюстрированы отношения ограничения и перекрытия, которые могут существовать между его прямоугольниками.

Имея представление о свойствах древовидной структуры, можно оценить его высоту при числе элементов N. Из свойств, описанных выше, следует, что каждый узел дерева содержит как минимум m

потомков. Поэтому наибольшая высота *R*-дерева, содержащего *N* индексных записей, будет не больше $\lceil log_m N \rceil$-1. При этом максимальное число узлов в таком дереве будет $\lceil N/m \rceil + \lceil N/m^2 \rceil + \ldots + 1$.

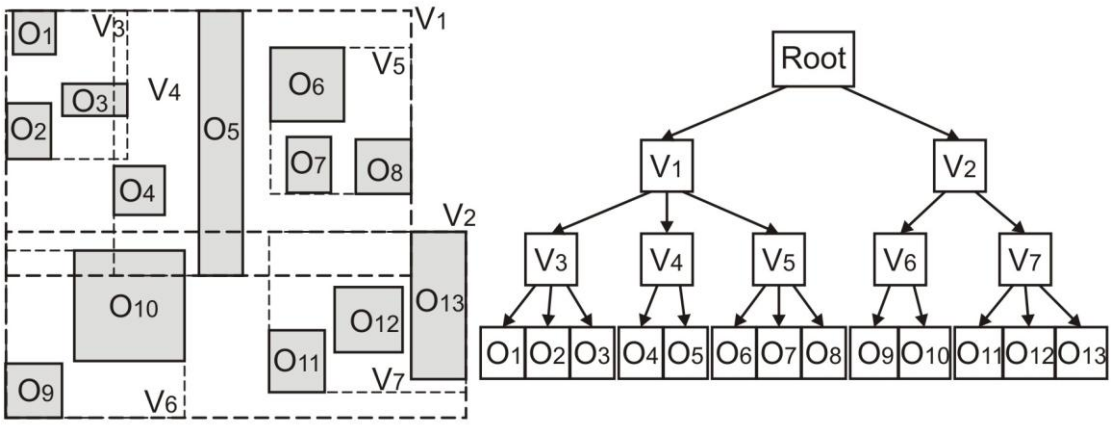

Рис. 3.9. Пример *R*-дерева

В наихудшем случае использование пространства памяти, в которой хранится индексная структура, будет *m/M*. Однако алгоритмы построения дерева разработаны таким образом, что структура будет стремиться содержать более *m* записей в узле. Это уменьшает высоту дерева и увеличивает полезное использование памяти.

Практические расчеты показывают, что если *m* больше 4, то дерево получается широким и почти все пространство используется для листьев, содержащих индексные записи.

Алгоритм поиска объекта в R-дереве

Алгоритм поиска в *R*-дереве очень похож на алгоритм поиска по *B*-дереву. Он также начинается в корне и опускается по нему к листовому узлу, выбирая в зависимости от заданных параметров поиска то или иное поддерево. Главное же отличие состоит в том, что возможен вариант, при котором более одного поддерева текущего узла участвует в поиске. Такая ситуация связана с применением метода размещения многомерных объектов, разрешающего пересекаться ограничивающим областям разных элементов. Данный факт может привести к многократному уменьшению скорости поиска, однако алгоритмы построения и изменения дерева стараются

поддерживать дерево в наиболее оптимальном виде.

В листинге 3.1. представлен один из возможных вариантов рекурсивной процедуры поиска объектов, имеющих хотя бы одну общую точку с областью поиска *S*.

Листинг 3.1.

```
//=====================================================
// Процедура поиска в R-дереве по области S
// Параметры:
//  V – текущая вершина (первоначально это корень)
//  S – область поиска
//  Res – множество результатов поиска
//=====================================================
ПОИСК(V, S, Res)
    [1] Если V не является листом, то
        Проверить все записи V', находящиеся в узле V
          Если MBR(V') пересекается с S, то
            Вызвать ПОИСК(V', S, Res)
    [2] Если V является листом, то
        Проверить все записи O, находящиеся в узле V
          Если MBR(O) пересекается с S, то
            Добавить запись O в множество Res
Конец ПОИСК
```

Изначально, при вызове процедуры, ей передаются в качестве параметров корень дерева (*V*), область поиска (*S*) и пустое множество (*Res*), в которое будут помещаться найденные объекты.

Процедура поиска состоит из двух частей. Первая часть выполняется только для внутренних узлов дерева. Процедура перебирает все дочерние узлы данного узла *V* и для каждого из них проверяет, пересекается ли его описывающий прямоугольник с областью поиска. Если такое пересечение наблюдается, то процедура вызывается рекурсивно для этого узла. В противном же случае, если между областью поиска и *MBR* потомка нет общих точек, такой потомок просто пропускается.

Вторая часть процедуры поиска вступает в действие при достижении листового уровня. Как и в предыдущем случае, происходит перебор всех элементов узла с целью проверки на пересечение их *MBR* с областью поиска. При нахождении подобных элементов они добавляются в множество результатов *Res*.

Рассмотрим описанный алгоритм на примере, показанном на рис. 3.10. Область поиска соответствует заданному прямоугольнику *ABCD*.

Первоначально процедура поиска вызывается для корня. Так как корень является внутренней вершиной, то для него выполняется первая ветка алгоритма поиска. Она проверяет узлы V_1 и V_2 на пересечение с заданной областью. Как нетрудно заметить, оба этих узла имеют общие точки с областью поиска, и поэтому для обоих из этих узлов рекурсивно вызывается процедура ПОИСК.

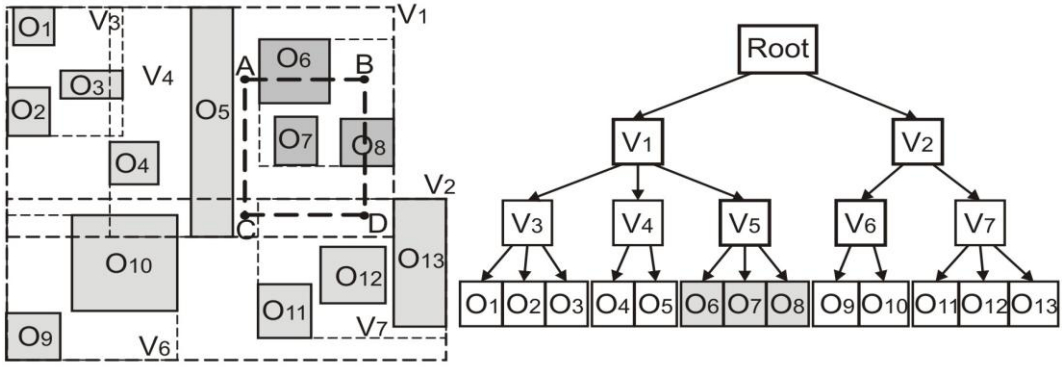

Рис. 3.10. Пример поиска в *R*-дереве

ПОИСК для вершины V_1 перебирает элементы V_3,V_4, V_5, причем только V_5 имеет пресечение с прямоугольником *ABCD*. Поэтому вершины V_3 и V_4 пропускаются и далее не рассматриваются. Дальнейший вызов процедуры для вершины V_5 выдаст в качестве результата три элемента – O_6, O_7, O_8, которые и будут добавлены в множество результата *Res*.

Аналогичным образом будет просмотрена ветка V_2. Из ее потомков только V_7 имеет общие точки с *ABCD*. Однако не один из элементов V_7 не пересекается с областью поиска. Данная ветка поиска оказалась ложной.

В результате поиска мы получаем список элементов, удовлетворяющих заданному запросу:

$$Res = \{O_6, O_7, O_8\}.$$

Другие виды поиска выполняются аналогично описанному. Для примера приведем листинг еще одной процедуры, выполняющей поиск объекта по точному совпадению с образцом. Процедура возвращает лист, в котором находится запрошенный объект, или *NULL*, если такого объекта в дереве нет.

Листинг 3.2

```
//=====================================================
// Поиск листа, в котором находиться объект O
// Параметры:
//   V – вершина, начиная с которой производиться поиск
//  O – объект, который нужно найти.
//=====================================================
ПОИСК_ОБЪЕКТА(V,O)
    [1] Если V не является листом, то
        Проверить все записи V', находящиеся в узле V
          Если MBR(V') полностью содержит MBR(O), то
            L = ПОИСК_ОБЪЕКТА(V',O)
            Если L ≠ NULL, то
              Вернуть L
    [2] Если V является листом, то
        Проверить все записи O', находящиеся в узле V
          Если O'=O, то
            Вернуть V
    [3] Вернуть NULL
Конец ПОИСК_ОБЪЕКТА
```

Процедура очень похожа на ту, что представлена в листинге 3.1. Как и в предыдущем случае, первый шаг проверяет, является ли рассматриваемая вершина *V* листовой. Если эта вершина находится на внутреннем уровне дерева, алгоритм просматривает все ее дочерние элементы. Однако, в отличие от предыдущего алгоритма, проверяется не просто пересечение *MBR* узла дерева с запрошенным узлом, а полное его включение. Это ужесточение условия необходимо для отбрасывания веток поиска, с которыми объект пересекается, но не входит в них.

Если некоторый узел может содержать поисковый объект, то процедура выполняется для него рекурсивно. При этом необходимо проанализировать, что вернул этот рекурсивный вызов. Если возвращенное значение равно *NULL*, то алгоритм продолжается дальше. Иначе – необходимо завершить процедуру поиска, так как листовой узел, содержащий объект *O*, уже найден. Это является еще одним отличием от предыдущего алгоритма.

Второй шаг алгоритма предназначен для листовой вершины. Он перебирает все элементы, содержащиеся в данном узле дерева, и

проверяет их на равенство поисковому объекту. При нахождении соответствия процедура возвращает текущий лист.

И, наконец, если не один из предыдущих поисков не дал результата, то на третьем шаге необходимо вернуть значение *NULL*, идентифицирующее тот факт, что поиск в данном элементе не привел к положительному результату.

Алгоритм добавления нового объекта в R-дерево

Добавление нового объекта в *R*-дерево похоже на процедуру вставки в *B*+-дерево. Новая индексная запись добавляется в листовой узел. Если узел переполняется, то происходит его деление, в результате которого у предка появляется еще один потомок. Если предок также оказывается переполненным, то и он делится и т.д. Таким образом, вставка одного объекта может повлиять на структуру дерева в целом.

Процедура вставки объекта представлена в листинге 3.3.

Листинг 3.3

```
//=====================================================
// Процедура вставки элемента в R-дерево
// Параметры:
//  O – объект, который нужно вставить в дерево
//=====================================================
ВСТАВКА(O)
    [1] L = ВЫБОР_ЛИСТА(O)
    [2] Если число элементов в L меньше M, то
          Добавить объект O в узел L
          L” = NULL
        Иначе
          L” = ДЕЛЕНИЕ_УЗЛА(L,O)
    [3] КОРРЕКТИРОВКА_ДЕРЕВА(L,L”)
Конец ВСТАВКА
```

Первое, что делает процедура вставки элемента *O* в *R*-дерево, это ищет листовой узел, в который необходимо поместить данный объект (шаг 1). Процедура поиска такого листа является очень важным шагом, так как неправильно выбранная позиция может привести к неэффективности структуры в целом. Один из возможных вариантов этой процедуры показан в листинге 3.4.

После того, как узел для вставки выбран, производится непосредственно само размещение объекта в нем (шаг 2). При этом, если в листовом узле *L* есть место для новой записи, объект *O* помещается в него и процедура заканчивает свою работу. В противном случае, если узел *L* уже содержит максимально возможное число записей, то происходит деление узла на два новых *L* и *L″*, которые содержат старые записи узла *L*, и добавляемый объект *O*. Существует несколько вариантов реализации процедуры деления узла. Каждый из них имеет свои сильные и слабые стороны. Все они будут описаны далее.

После вставки объекта в дерево и возможного расщепления узла необходимо корректировать дерево (шаг 3). Процедура корректировки включает расширение границ минимального описывающего прямоугольника (*MBR*) для текущего узла и всех его предков. Также эта процедура распространяет расщепления узлов вверх по дереву, если это необходимо. Пример такой процедуры показан в листинге 3.5.

Рассмотрим алгоритмы упомянутых процедур подробнее.

Листинг 3.4

```
//=====================================================
// Процедура поиска листа для вставки O в R-дерево
// Параметры:
//  O – объект, который нужно вставить в дерево
//=====================================================
ВЫБОР_ЛИСТА(O)
    [1] V = корень R-дерева
    [2] Если V является листом, то
          Завершить процедуру и вернуть V
    [3] Для всех потомков V′ вершины V выбрать
          V_new = потомок, для которого MBR(V′,O)-MBR(V′)
                  минимальный
    [4] V = V_new
        Перейти к шагу 2
Конец ВЫБОР_ЛИСТА
```

Процедура начинает свой поиск с корня дерева (шаг 1), занося его в некоторую переменную *V*, отвечающую за текущий элемент поиска. Затем производится проверка вершины *V*, является ли она листом. Если данная вершина размещена на листовом уровне, то

процедура завершает свою работу, возвращая в качестве результата вершину *V* (шаг 2).

На третьем шаге процедура перебирает всех потомков вершины *V*. Цель данного шага заключается в том, чтобы выбрать того потомка, чей *MBR* увеличится наименьшим образом при помещении в него объекта *O* (в идеале – вообще не измениться). В спорных ситуациях, когда найдено более одного потомка с одинаковым увеличением *MBR*, выбирать необходимо тот, который имеет меньшую площадь.

После выбора дочернего узла его заносят в переменную *V* и процедуру поиска повторяют с шага 2.

Теперь представим листинг еще одной процедуры, которая уже упоминалась ранее – процедура корректировки дерева.

Листинг 3.5

```
//=====================================================
// Процедура корректировки R-дерева после вставки объекта
// Параметры:
//  L – вершина, MBR которой необходимо корректировать
//  L" – вершина, образовавшаяся при делении L на две
//     части. Если деление не производилось, L" = NULL.
//=====================================================
КОРРЕКТИРОВКА_ДЕРЕВА (L, L")
    [1] V = L,  V" = L"
    [2] Если V является корнем, то
          Если V" ≠ NULL, то
            Root = новый корневой узел
            Поместить в Root элементы V и V"
          Выйти из процедуры
    [3] P = Parent(V)
        P_V = запись в узле P о потомке V
        Скорректировать MBR(P_V)
    [4] Если V" ≠ NULL, то
          P_V" = новая запись о узле V"
          Если число элементов в P меньше M, то
            Добавить объект P_V" в узел P
          Иначе
            P" = ДЕЛЕНИЕ_УЗЛА(P, P_V")
    [6] V = P
        V" = P"
        Перейти к шагу 2
```

```
Конец КОРРЕКТИРОВКА_ДЕРЕВА
```

Как было отмечено, процедура корректировки изменяет *MBR* всех вершин дерева, которые расположены выше листа с вставленным объектом. Второй и не менее важной функций процедуры корректировки является распространение деления вершин вверх по дереву, в случае, если будет происходить переполнение на внутренних узлах дерева.

В качестве параметров в процедуру передаются два новых узла, которые получились при вставке объекта в дерево. Если разбиение не произошло, то первым параметром передается старый узел, а второй параметр приравнивается в *NULL*.

На первом шаге процедура заносит переданные параметры в переменные *V* и *V″*. Эти переменные будут отвечать за текущие вершины в дереве, которые необходимо исправить.

После этого происходит сравнение вершины *V* корня дерева. Если данная вершина является корнем, то это означает, что изменения уже распространились до верха дерева и необходимо просто завершить процедуру корректировки. Однако стоит учитывать один момент: если после предыдущих манипуляций произошло расщепление корня на два узла (переменная $V'' \neq NULL$), то необходимо создать новый корень дерева, узлами-потомками которого будут *V* и *V″*.

Если предыдущий пункт не выполнен, то происходит корректировка. Для этого определяется предок узла *V*, а также запись в нем об этом узле. После этого *MBR* найденной записи изменяется таким образом, чтобы включать в себя все *MBR* дочерних элементов узла *V*, но при этом не содержать лишних областей.

Четвертый шаг алгоритма выполняется только в том случае, если предыдущие действия вызвали деление узла. В этом случае у нас в переменной *V″* будет находиться вершина с элементами, которые пока еще не помещены в дерево. Для этой вершины необходимо создать запись $P_{V''}$, которая будет содержать минимальный описывающий прямоугольник для данной вершины и ссылку на саму вершину. Эту запись и нужно разместить в предке узла *V*.

Однако при помещении в узел *P* записи $P_{V''}$ необходимо помнить, что данная операция может привести к переполнению и тогда придется разбивать узел *P* на два новых.

После всех описанных операций в переменные *V* и *V″* заносятся новые значения *P* и *P″* соответственно, и алгоритм повторяется заново с шага 2.

Алгоритм удаления объекта из R-дерева

Для того, чтобы структуру можно было считать полностью динамической, необходима поддержка двух операций: добавление новых элементов и удаление уже существующих. Добавление элементов было рассмотрено выше. В листинге 3.6 представлена процедура удаления. Кроме удаления объекта из дерева, она должна корректировать дерево для сохранения его свойств.

Листинг 3.6

```
//=====================================================
// Процедура удаления объекта из R-дерева.
// Параметры:
//  O – объект, который нужно удалить.
//=====================================================
УДАЛЕНИЕ(O)
    [1] V = корень дерева
        L = ПОИСК_ОБЪЕКТА(V,O)
        Если L = NULL, то
          Завершить процедуру удаления
    [2] Удалить объект O из L
        V = L
        Q = пустое множество
    [3] Если узел V является корнем, то
          Перейти к шагу 7
    [4] P = Parent(V)
        P_V = запись в узле P о потомке V
    [5] Если число элементов в V меньше m, то
          Удалить P_V из P
          Переместить все элементы из V в множество Q
          Удалить V
        Иначе
          Скорректировать MBR(V)
    [6] V = P
        Перейти к шагу 3
    [7] Если у корня всего один потомок, то
```

```
            Удалить корневой узел
            Сделать новым корнем этого потомка
    [8] Вставить узлы из множества Q обратно в дерево
Конец УДАЛЕНИЕ
```

Первое, что производит процедура удаления объекта *O* из *R*-дерева, это ищет листовой узел, в котором находится данный объект. Для этого используется процедура поиска ПОИСК_ОБЪЕКТА, описанная в листинге 3.2. В качестве параметров ей передается вершина, с которой нужно начать поиск (в нашем случае это корень) и объект поиска. Если объект не будет найден, то данная процедура вернет *NULL*. При этом необходимо завершить и процедуру удаления.

На втором шаге удаляется объект *O* из узла *L* и подготавливаются временные переменные для коррекции дерева. В переменную *V* (текущая вершина для коррекции) заносится листовой узел *L*, а в переменную *Q* – пустое множество (это множество вершин, которые необходимо потом вставить в дерево заново).

Далее необходимо проверить, является ли вершина *V* корнем. Если *V* – корневая вершина, то шаги 4–6 нужно пропустить и перейти сразу к седьмому пункту алгоритма. Иначе – находим предка для вершины *V*, и в нем определяем запись, ссылающуюся на *V* (P_V).

Если в рассматриваемом узле число записей меньше минимально возможного (*m*), то необходимо удалить этот узел из дерева. При этом все элементы из *V* помещаются в множество *Q* (чтобы потом снова быть размещенными в дереве, но в других вершинах) и из вершины *P* удаляется элемент, ссылающийся на удаленную вершину (удаляется P_V).

Если же записей в вершине *V* больше, чем заданный параметр *m*, то удалять вершину не нужно. При этом необходимо просто скорректировать *MBR* узла таким образом, чтобы он охватывал все прямоугольники дочерних узлов, но при этом не включал лишнего пространства (после удаления узлов вполне вероятно можно будет сузить *MBR*, который хранится в записи P_V).

После проделанных операций необходимо распространить сделанные изменения вверх по дереву (скорректировать *MBR* узла предка или, возможно, даже удалить его, если он оказался не

заполненным до предела m). Для этого в переменную V заносится предок текущей вершины и повторяется алгоритм с шага 3.

После того, как все изменения дойдут до корня, алгоритм продолжится с шага 7. Исходя из свойств *R*-дерева, описанных в начале данного параграфа, корень должен иметь не меньше двух потомков. Поэтому необходимо просто проверить число дочерних узлов у корня и при нахождении там всего одного потомка сделать его новым корнем дерева.

Последнее, что необходимо выполнить в процедуре удаления, это вставить временно удаленные узлы из множества Q обратно в дерево. Данная процедура выполняется полностью аналогично описанной ранее процедуре ВСТАВКА за одним лишь исключением: вершины из множества Q необходимо разместить на тех же уровнях, на которых они были до процедуры удаления. Этого требования необходимо придерживаться для того, чтобы не нарушить сбалансированность дерева (одно из свойств *R*-дерева заключается в том, что все листовые узлы находятся в нем на одном уровне).

Алгоритм разбиения узла

Изменение данных прикладной задачи требует частого изменения индексной структуры. Для добавления новой записи в уже заполненный узел *R*-дерева, содержащий M записей, необходимо распределить $M+1$ элемент между двумя узлами. Процедура разбиения узла может быть вызвана не только при добавлении новых элементов в индекс, но и при перестройке дерева, при удалении ненужной записи, при обновлении данных или даже при его корректировке.

Алгоритм, выполняющий деление узла, особенно важен, так как плохое разбиение может сильно затруднить операции поиска по дереву. Разбиение узла без учета критериев оптимальности построения дерева приводит к увеличению времени работы процедуры поиска конкретного объекта, а следовательно, к ухудшению работы индексной структуры в целом. При плохом разбиении узлы дерева разрастаются вдоль осей координат и захватывают много пространства, не содержащего ни одного объекта. Такой пример показан на рис. 3.11. С одной стороны вариант (а) обеспечивает нулевое перекрытие двух узлов дерева. Однако суммарная площадь этих узлов будет значительно больше самих