Сильно ветвящееся Б-дерево

Если сильно ветвящееся дерево имеет ту особенность, что в своих вершинах содержит не по одному, а по нескольку ключей, и это множество ключей увязано с множеством ссылок на вершины-потомки, такую структуру принято называть Б-деревом (сильно ветвящимся Б-деревом), а его вершины называть страницами.

Каждая страница Б-дерева возглавляет собой Б-поддерево.

Сильно ветвящимся Б-деревом называется структура, страницы которой подчинены требованиям:

- корневая страница содержит не менее одного ключа;
- каждая страница, кроме корневой, содержит не менее n ключей;
- каждая страница содержит не более $2n$ ключей;
- каждая страница либо представляет собой лист (терминальную вершину), т.е. не имеет потомков, либо имеет $k+1$ потомков, где k – число ключей на этой странице;
- все страницы-листья находятся на одном уровне.

Число n называется порядком Б-дерева; $n \geq 1$.

Пример Б-дерева порядка 2 показан на Рис. 1.

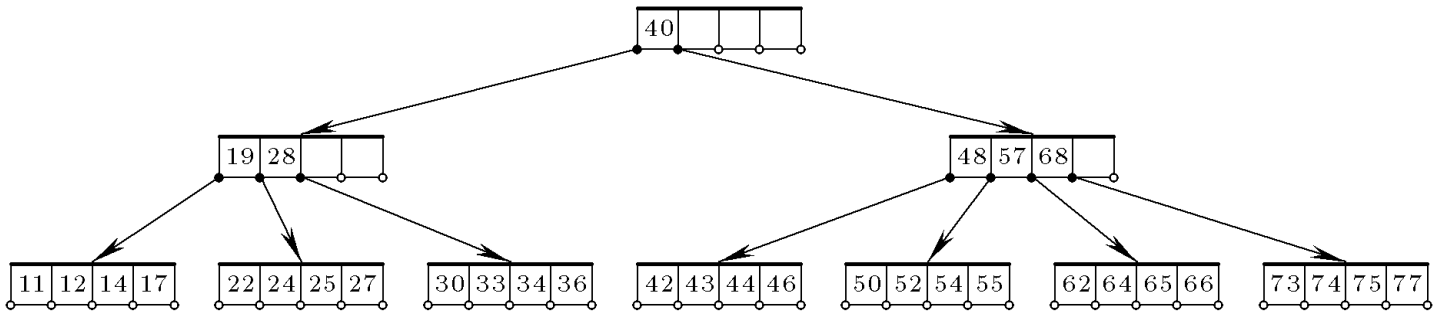


Рис. 1

Множество ключей на странице может быть, в простейшем случае, организовано в виде линейного списка или в виде массива с диапазоном изменения индексов $1\ldots n$; в последнем случае множество ссылок на странице представляется массивом с диапазоном изменения индексов $0\ldots n$.

Если число n велико, оправдана древовидная организация ключей на странице.

По определению Б-дерева, у каждой страницы имеется либо не менее двух страниц-потомков, либо нет ни одной страницы-потомка. Все страницы-потомки одного предка уместно называть «страницами-сёстрами». Каждая (кроме корневой) страница имеет сестру – либо старшую (слева), либо младшую (справа).

В Б-деревьях используется понятие ***левого потомка*** (левого поддерева) и ***правого потомка*** (правого поддерева) для каждого ключа. Правый потомок i – го ключа является левым потомком $(i+1)$ – го ключа. У каждого ключа либо есть одновременно и левый, и правый потомки, либо одновременно нет ни того, ни другого.

Например, на Рис.1 левым потомком ключа «40» корневой страницы является страница с ключами «19, 28», правым потомком – страница с ключами «48, 57, 68».

Как и в деревьях оптимального поиска, уровень корневой страницы принимается равным 1, уровень её непосредственных потомков равен двум, и т.д.

Б-дерево называется **Б-деревом поиска**, если выполнены условия:

- для ключей определены отношения «больше» – «меньше» – «равно»;
- каждый ключ больше каждого из ключей его левого поддерева и меньше каждого из ключей правого поддерева (если поддеревья есть).

Структура, изображённая на Рис.1, является Б-деревом поиска.

Далее будут рассматриваться **только** Б-деревья поиска. Причём, слово «поиск», для краткости, будет опущено.

Б-дерево изначально разрабатывалось для размещения на внешних носителях (жестких магнитных дисках). Обращение к одной компоненте файла, которая называется страницей, и которая содержит много ключей, выгоднее с точки зрения экономии времени при работе с файлами, чем обращение к компоненте с одним единственным ключом.

Размер страницы рационально выбирать таким, чтобы он был кратен размеру кластера в файловой системе.

Включение нового ключа в Б-дерево

Все действия по реорганизации структуры рассматриваются на примере Б-дерева порядка 2.

Внесение нового ключа производится **только** в терминальную страницу Б-дерева.

Процедура включения будет иметь имя IncludeKey. Место для нового ключа отыскивается, по сути, так же, как и в двоичном дереве поиска. Если терминальная страница расположена на уровне L (уровень = глубина + 1), то внесением в дерево нового ключа занимается L–й экземпляр процедуры IncludeKey, вызываемый рекурсивно.

На Рис. 2 показано последовательное внесение ключей «29», «34», «32», «23», в изначально пустое Б-дерево (случай $L=1$). Происходит формирование и допустимое пополнение корневой страницы (единственной пока).

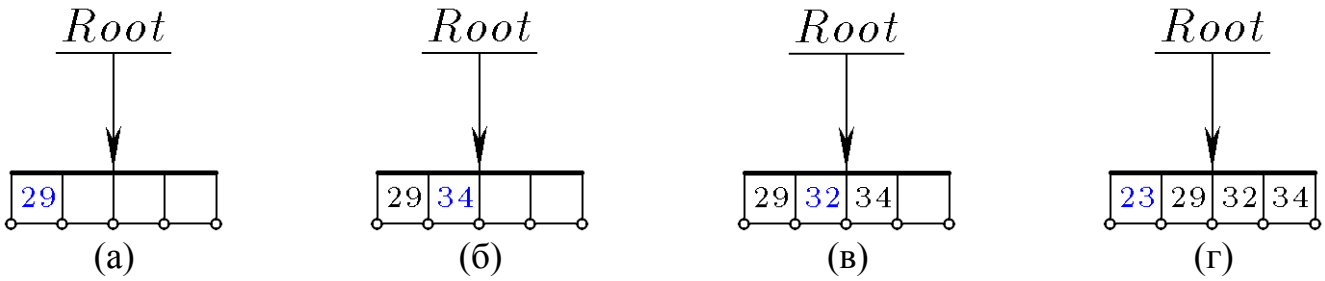


Рис. 2

На Рис. 3 показано внесение ключа «33», **переполняющего** корневую страницу.

Порядок перестроения дерева таков.

В переполненном списке (содержащем $2n+1$ ключей) выбирается один срединный элемент («31», $(n+1)$–й по счету), левая половина списка (n ключей, «23», «29»), правая половина списка (n ключей, «32», «34»). Создаются две новые страницы (показаны синим цветом). Первая из них заполняется левой половиной списка и делается левым потомком корневой страницы. Вторая из них заполняется правой половиной списка и делается правым потомком корневой страницы. В корневой странице остаётся единственный, срединный элемент списка (и вовсе не обязательно тот, что был внесён последним).

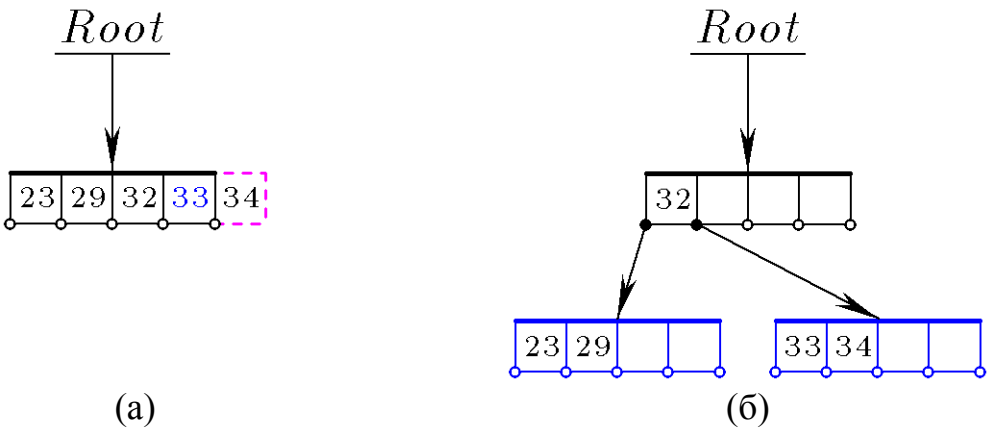


Рис. 3

Переполнение корневой страницы – единственная возможная причина увеличения высоты Б-дерева (на единицу).На Рис. 4 показано внесение ключа «53» в терминальную некорневую страницу (случай $L \geq 2$). Ключ переполняет страницу (часть «а»). При перестроении дерева происходит нечто похожее на то, что рассмотрено на Рис. 3, но новая страница – только одна.

Порядок перестроения дерева таков.

В переполненном списке (содержащем $2n+1$ ключей) выбирается один срединный элемент («49»), левая половина списка (n ключей, «41», «45»), правая половина списка (n ключей, «50», «53»). Средний элемент списка «всплывает» в родительскую (по отношению к переполненной) страницу и занимает там место (вставляется) в порядке сортировки по возрастанию. При этом «пододвигаемые» вправо ключи («60») смещаются вместе со своими правыми ссылками, что порождает одну «вакантную» ссылку справа от «всплывшего» ключа.

В переполненной странице сохраняется только левая половина списка «41», «45»). Создаётся новая страница (показана синим цветом) и заполняется правой половиной списка («50», «53»). «Вакантная» ссылка (зелёного цвета, как и «всплывший» ключ) нацеливается на созданную новую страницу. Переполнение устранено (часть «б»).

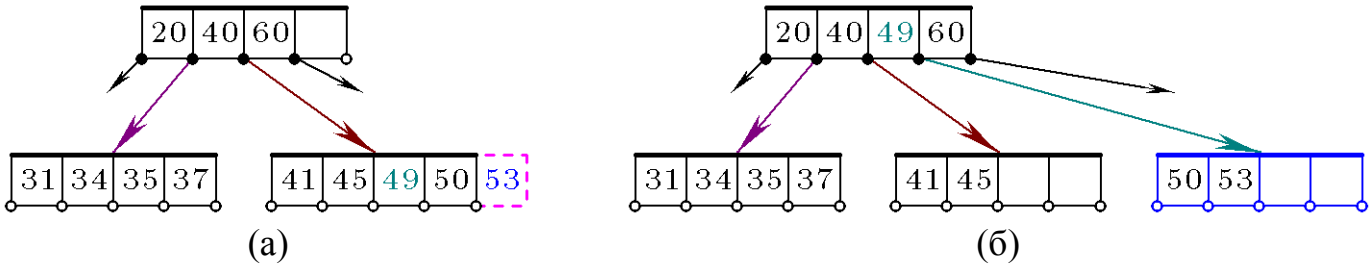


Рис. 4

Важно отметить, что «всплытие» ключа в родительскую страницу может повлечь и **её** переполнение. Обработкой этого переполнения (по уже названному алгоритму) занимается $(L-1)$–й экземпляр метода IncludeKey. Возможна длинная цепь таких переполнений, иногда и вплоть до корневой страницы.

Исключение ключа из Б-дерева

Процедура исключения будет иметь имя ExcludeKey. Место удаляемого ключа отыскивается так же, как и в двоичном дереве поиска. После удаления возможна ситуация, при которой на странице остаётся менее n ключей. Такая ситуация далее, для краткости, будет называться «недобором».

Исключение из терминальной страницы

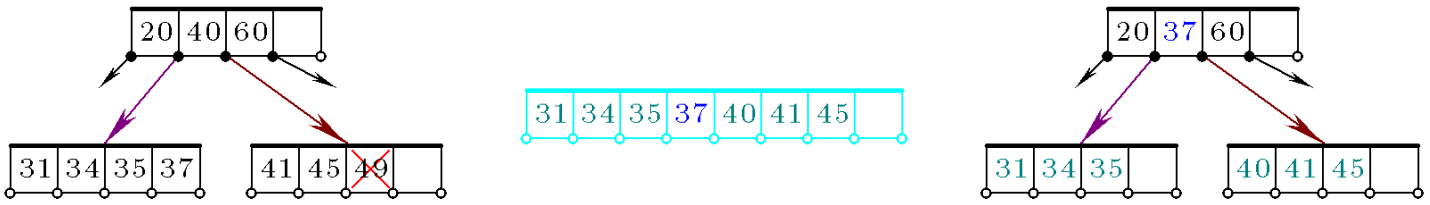
Если терминальная вершина имеет уровень L, то удалением ключа занимается $L-$й экземпляр процедуры ExcludeKey, вызываемый рекурсивно.

На Рис. 5, 6 показаны «лёгкий» и «трудный» случаи удаления ключа «49».

Из ключей «похудевшей» страницы и её ближайшей сестры (если не старшей, как на Рис. 5, 6, то младшей) формируется «длинный» список (на частях «б» показан светло-зелёным цветом). В него, в порядке сортировки, вносится тот ключ из родительской страницы, для которого две обсуждаемые сестры являются левым и правым потомками («40»). Пусть m есть длина полученного списка. Ясно, что $m < 4n$.

Выбирается срединный элемент списка («37»)

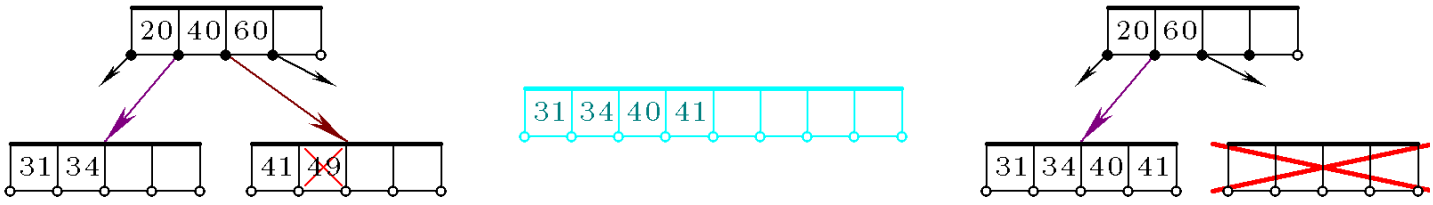
Пусть $m \geq 2n+1$ (Рис. 5). Выбирается срединный элемент длинного списка («37»), он становится на место «бывшей главы» в родительскую страницу («40»). Левая («31, 34, 35») и правая («41, 41, 45») половины списка расставляются в страницы-сёстры, участвующие в дележе. Каждой достаётся не менее n ключей.



(а) (б) (в)

Рис. 5

Пусть $m \leq 2n$ (Рис. 6). В действительности, это означает, что $m = 2n$. Длинный список («31, 34, 40, 41») помещается в старшую (левую) из участвующих в процессе страниц-сестёр. Младшая (правая) страница удаляется. «Бывшая глава» двух сестёр в родительской странице («40», часть «а») перемещается в левую, сохранившуюся страницу-сестру. При этом ключи, расположенные в родительской странице правее «бывшей главы», сдвигаются влево вместе со своими правыми ссылками, а ссылка на удалённую страницу исчезает.



(а) (б) (в)

Рис. 6

Важно отметить, что удалением ключа из родительской страницы и обработкой возможного случая недобора занимается $(L-1)-$й экземпляр метода ExcludeKey. При

этом L–й экземпляр метода даёт лишь информацию методу-предку о необходимости удаления.

Исчезновение ссылки на уделённую страницу не означает, что страница исчезает из файла, и что размер файла уменьшается. Для каждого конкретного файла «Сборщик мусора» не предусмотрен. Заботу о «потерянных» страницах должен взять на себя программист.

Исключение из нетерминальной страницы

Этот случай ненамного сложнее (Рис. 7). Идея решения состоит в том, чтобы обменять удаляемый ключ («31») местами с самым правым из его левых потомков («27»), который в Б-дереве может располагаться только на терминальной странице. После этого удаляемый ключ встанет на терминальную страницу, а процесс удаления с неё уже изучен.

Казалось бы, плохо, что Б-дерево после такого обмена перестаёт быть деревом поиска (часть «б», Рис. 7). Но после ликвидации целевого ключа («31») дерево вновь становится Б-деревом поиска.

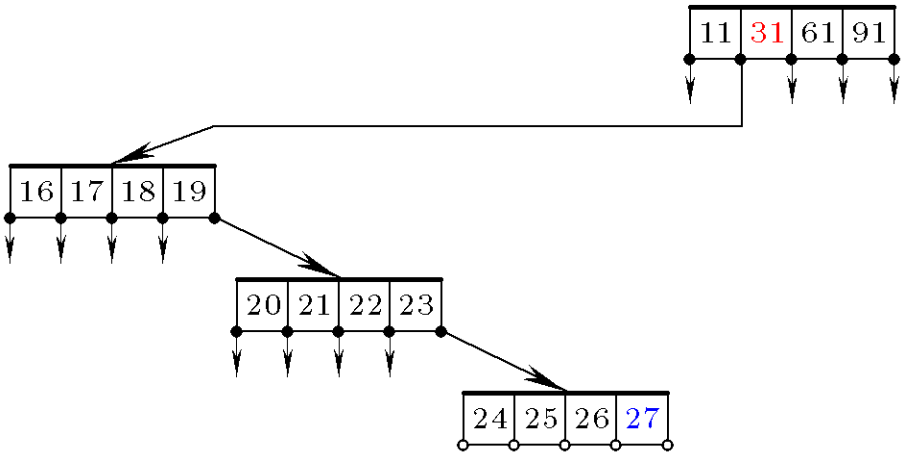


Рис. 7 (а)

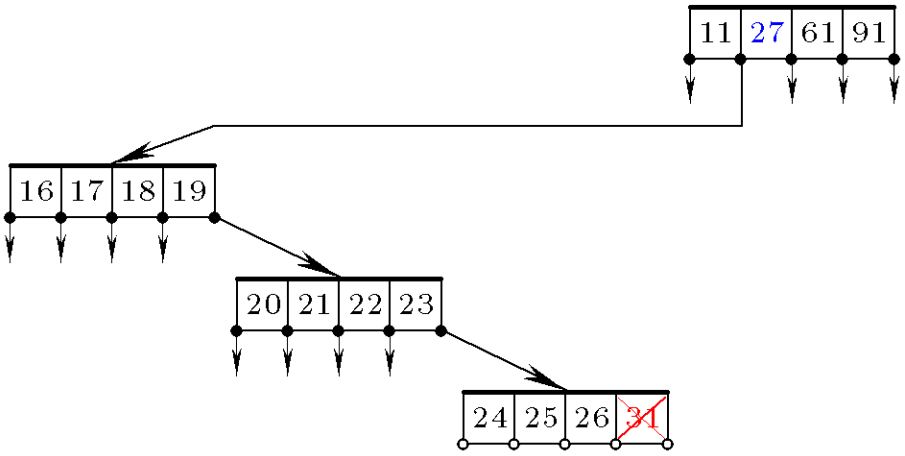


Рис. 7 (б)

Недостача ключей в нетерминальной странице

Такая недостача может произойти только из-за перемещения (опускания) какого-то из ключей данной страницы в одну страницу–потомок данного ключа, при одновременной ликвидации второй страницы–потомка данного ключа.

Недостача в некорневой нетерминальной странице обрабатывается точно так же, как недостача в терминальной странице (пункт «Исключение из терминальной страницы»). Устранение недостачи на текущей странице может (хоть и необязательно) повлечь перемещение в неё ключа из страницы, которая является родительской по отношению к текущей (с одновременным удалением одной ссылки из родительской страницы). Родительская страница, в свою очередь, также может чрезмерно «похудеть», и повлечь цепную реакцию «похудания» вплоть до корневой страницы.

Недостача в корневой странице особенная: в ней ключей не остаётся вовсе. Однако, при всех добавлениях и удалениях ключей в каждой нетерминальной странице становилось ссылок (на другие, дочерние страницы) ровно на одну больше, чем ключей. Далее возможна одна из двух ситуаций.

1. Корневая страница – нетерминальная. Следовательно, от неё сохранилась одна ссылка на страницу–потомок. Эта страница–потомок объявляется новой корневой страницей. Старая, пустая страница удаляется из дерева.

2. Корневая страница – терминальная. Следовательно, это была единственная страница в дереве, а само дерево отныне пусто.

К документу прилагается проект WFABTree. Тексты некоторых из методов проекта показаны ниже и сопровождены вполне достаточными комментариями.

Структуры

```
unsafe struct MyUnsafeRecordType
{
   public int nData;
   public fixed int mData[2 * MyConstants.nOrder];
   public fixed int mLink[2 * MyConstants.nOrder + 1];
}

struct MyLongRecordType
{
   public int nData;
   public int[] mData;
   public int[] mLink;

   public MyLongRecordType(int nData)
   {
      this.nData = nData;
      mData = new int[4 * MyConstants.nOrder];
      mLink = new int[4 * MyConstants.nOrder + 1];
      int i;
      for (i = 0; i < 4 * MyConstants.nOrder; i++) mData[i] = 0;
```

```
        for (i = 0; i < 4 * MyConstants.nOrder + 1; i++) mLink[i] = -1;
    }
}
```

уже обсуждались в документе “ДополнительноОПамяти.doc”. Единственное отличие в том, что структура MyLongRecordType снабжена конструктором. В этой структуре предусмотрено максимально большое возможное значение ключей при слиянии двух страниц вместе с возглавляющим эти страницы ключом-предком этих страниц.

Метод Away(…) используется в качестве «заглушки» непредвиденных ситуаций, связанных с ошибками в коде. До сей поры он ни разу не сработал, ошибок в коде, я надеюсь, нет.

В документе “ДополнительноОПамяти.doc” обсуждались, также, методы ReadPage(…) и WritePage(…).

Для внесения в дерево нового ключа применяется нерекурсивный метод Include(…). Он обращается к рекурсивному методу IncludeKey(…), который, в свою очередь, пользуется нерекурсивным методом InsertItem(…).

Для удаления ключа из дерева применяется нерекурсивный метод Exclude(). Он обращается к рекурсивному методу ExcludeKey(…), который, в свою очередь, пользуется нерекурсивным методом MergeTwoPages(…).

После запуска приложения следует нажать кнопку «Инициализация», после чего выбрать режим работы с файлом дерева. Нажатие кнопки «Создать в файле» приводит к созданию пустого дерева, которому соответствует новый файл (с выбранными именем) нулевой длины. Если ранее файл существовал, информация в нём будет потеряна. Нажатие кнопки «Загрузить из файла» означает продолжение работы с ранее уже созданным файлом.

Подробности графического представления дерева не обсуждаются. Графический элемент управления “PictureBox” не защищён от «перегрузки», то есть от выхода картинки за границы элемента.

Коды написаны так, чтобы корневая страница всегда располагалась в начале файла. Если после удалений ключей страницы дерева из него «выпадают» (но остаются бесполезным грузом в файле), «Сборщик файлового мусора» не сей счет не предусмотрен.

Итак, тексты основных для проекта методов:

```
private void InsertItem(
    ref MyLongRecordType R,    // Структура, в которую считывается страница
    int lStartPage,            // Номер записи этой страницы в файле
    ref int iKeyUp             // Ключ, который сплывёт в родительскую страницу
    ref int lUp,               // Ссылка, всплывающая вместе с ключом
    int iNew,                  // Номер позиции для нового ключа
    int iKeyNew,               // Новый ключ, который вставляется в данную страницу
    int lNew                   // Новая ссылка, вставляемая вместе с новым ключом
)
{
    int i;
    MyLongRecordType RAux = new MyLongRecordType(0);
    Int64 l;

    for (i = R.nData; i >= iNew; i--)
    {
        R.mData[i] = R.mData[i - 1];
        R.mLink[i + 1] = R.mLink[i];
    }
    R.mData[iNew - 1] = iKeyNew;
    R.mLink[iNew] = lNew;
    R.nData++;
// Теперь новый ключ вместе с новой  ссылкой вставлен в структуру.

    if (R.nData > 2 * MyConstants.nOrder)
// =>  R.nData == 2 * MyConstants.nOrder + 1
    {
// Сюда попали => страница переполнена.
// Начинается разделение одной страницы на две.

        for (i = 1; i <= MyConstants.nOrder; i++)
        {
            RAux.mData[i - 1] = R.mData[i + MyConstants.nOrder];
            RAux.mLink[i] = R.mLink[i + MyConstants.nOrder + 1];
        }
        RAux.mLink[0] = R.mLink[MyConstants.nOrder + 1];
// Структура RAux приняла на себя
// правые ключи и правые ссылки разделяемой страницы.

        R.nData = MyConstants.nOrder;
        RAux.nData = MyConstants.nOrder;
// Зафиксировали: обе страницы имеют по MyConstants.nOrder ключей.
```

```
        iKeyUp = R.mData[MyConstants.nOrder];
// Срединный ключ, который всплывёт вверх.

        for (i = MyConstants.nOrder + 1; i <= 2 * MyConstants.nOrder + 1; i++)
        {
            R.mData[i - 1] = -1;
            R.mLink[i] = -1;
            RAux.mData[i - 1] = -1;
            RAux.mLink[i] = -1;
        }
// Это можно было и не делать.

        if (lStartPage == 0)
        {
// Сюда попали => разделяемая страница – корневая.
// Следовательно, нужно создать две новые страницы в файле.

            l = FS.Length;
            if (l % (Int64)nBytes != 0) Away(5);
            lUp = (int)(l / (Int64)nBytes);
// lUp – номер записи для первой из двух новых страниц.

            WritePage(lUp, ref R);
// Левая страница из структуры записана в файл.
// Размер файла увеличился на 1 страницу.

            if (FS.Length != l + nBytes) Away(6);

            R = new MyLongRecordType(0);
            R.nData = 1;
            R.mData[0] = iKeyUp;
            R.mLink[0] = lUp;
// В структуре R формируется запись с единственным ключом (и двумя ссылками),
// которая станет новой корневой страницей.
// Второй ссылки пока нет.

            l = FS.Length;
            if (l % (Int64)nBytes != 0) Away(7);
            lUp = (int)(l / (Int64)nBytes);

            WritePage(lUp, ref RAux);
// Правая страница из структуры записана в файл.
// Размер файла увеличился ещё на 1 страницу.
```

```
            R.mLink[1] = lUp;
// Вот и вторая ссылка из новой корневой страницы.

            iKeyUp = -1;
            lUp = -1;
        }
        else
        {
// Сюда попали => разделяемая страница – НЕкорневая.
// Следовательно, нужно создать только одну новую страницу в файле.

            l = FS.Length;
            if (l % (Int64)nBytes != 0) Away(9);
            lUp = (int)(l / (Int64)nBytes);

            WritePage(lUp, ref RAux);
            WritePage(lUp, ref RAux);
// Правая страница из структуры записана в файл.
// Размер файла увеличился на одну страницу.

            if (FS.Length != l + nBytes) Away(10);
        }
    }
    else
    {
// Сюда попали => страница НЕ переполнена.

        iKeyUp = -1;
        lUp = -1;
// Сигнал вызывающему методу о том, что страница НЕ переполнена.
// Не всплывает ключ и не всплывает ссылка.
    }

    WritePage(lStartPage, ref R);
// Пополненная единственная страница
// или левая половина переполненной и разделённой страницы
// из структуры записана в файл.
}
```

```
public void IncludeKey(
    int iKeyFind,          // Ключ, который вносится в дерево.
    int lStartPage,        // Страница, с которой начинает работу данный экземпляр метода.
    ref int iKeyUp,        // Ключ, который может всплыть вверх.
    ref int lUp)           // Ссылка, которая всплывёт вместе с ключом.
)
{
    if (FS.Length < 1)
    {
// Сюда попали => дерево пусто.
        R = new MyLongRecordType(0);
        InsertItem(ref R, lStartPage, ref iKeyUp, ref lUp, 1, iKeyFind, -1);
        WritePage(0, ref R);
        return;
//  Внесли в дерево единственную страницу с единственным ключом.
    }

    MyLongRecordType R;
    int i, iL, iR;
    bool bTerminal = false;

    ReadPage(lStartPage, out R);

    if (R.mLink[0] < 0) bTerminal = true; else bTerminal = false;
// Нет самой левой ссылки на потомка => нет вовсе ссылок.
// Страница терминальная.

    iR = iKeyFind - R.mData[0];

    if (iR == 0)
    {
        nRFound = lStartPage;
        nDFound = 0;
        bFound = true;
        return;
// Ключ в дереве уже есть.
    }


    if (iR < 0)
    {
// Сюда попали => новый ключ меньше всех ключей на текущей странице.

        if (bTerminal)
            InsertItem(ref R, lStartPage, ref iKeyUp, ref lUp, 1, iKeyFind, -1);
```

```
// Сюда попали => вставлять ключ надо на эту страницу.
        else
        {
            IncludeKey(iKeyFind, R.mLink[0], ref iKeyUp, ref lUp);
            if (bFound) return;
// Ключ в дереве уже есть.

            if (lUp > -1)
                InsertItem(ref R, lStartPage, ref iKeyUp, ref lUp, 1, iKeyUp, lUp);
// Если ключ всплыл из дочерней страницы, он вставляется в текущую страницу.
        }

        return;
    }

    iR = iKeyFind - R.mData[R.nData - 1];

    if (iR == 0)
    {
        nRFound = lStartPage;
        nDFound = R.nData - 1;
        bFound = true;
        return;
// Ключ в дереве уже есть.
    }

    if (iR > 0)
    {
// Сюда попали => новый ключ больше всех ключей на текущей странице.

        if (bTerminal)
            InsertItem(ref R, lStartPage, ref iKeyUp, ref lUp, R.nData + 1, iKeyFind, -1);
// Сюда попали => вставлять ключ надо на эту страницу.
        else
        {
            IncludeKey(iKeyFind, R.mLink[R.nData], ref iKeyUp, ref lUp);
            if (bFound) return;
// Ключ в дереве уже есть.

             if (lUp > -1)
                InsertItem(ref R, lStartPage, ref iKeyUp, ref lUp, R.nData + 1, iKeyUp, lUp);
// Если ключ всплыл из дочерней страницы, он вставляется в текущую страницу.
        }
```

```
        return;
    }

// Сюда попали => новый ключ где-то между ключей на текущей странице.

    for (i = 1; i <= R.nData; i++)
    {
        iL = iR;

        if (iL == 0)
        {
            nRFound = lStartPage;
            nDFound = i - 1;
            bFound = true;
            return;
// Ключ в дереве уже есть.
        }

        if (i == R.nData) break;

        iR = iKeyFind - R.mData[i];

        if (iR < 0)
        {
// Автоматически iL > 0, раз сюда попали и не выскочили по iL = 0.

            if (bTerminal)
                InsertItem(ref R, lStartPage, ref iKeyUp, ref lUp, i + 1, iKeyFind, -1);
// Сюда попали => вставлять ключ надо на эту страницу.
            else
            {
                IncludeKey(iKeyFind, R.mLink[i], ref iKeyUp, ref lUp);
                if (bFound) return;
// Ключ в дереве уже есть.

                if (lUp > -1)
                    InsertItem(ref R, lStartPage, ref iKeyUp, ref lUp, i + 1, iKeyUp, lUp);
            }

            return;
        }
    }
}
```

```
public bool Include(int iKey, bool bReportIfKeyIsInTree)
{
   bFound = false;

   FS = new FileStream(sFileName, FileMode.Open);
   int iKeyUp = 0;
   int lUp = -1;
   IncludeKey(iKey, 0, ref iKeyUp, ref lUp);
   FS.Close();

   if (bFound)
   {
      if (bReportIfKeyIsInTree)
         MessageBox.Show("Ключ " + iKey.ToString() + " уже есть в дереве!");
      return false;
   }

   mxStart[0] = xGapEdge;
   myStart[0] = yGapEdge;

   DrawTree();

   return true;
}

void RemoveItem(
   ref MyLongRecordType R,   // Структура, в которой содержится страница
   int iRemove,              // Номер позиции для удаляемого ключа
   int iLevel,               // Уровень. Для 1-го экземпляра метода iLevel=0
   ref bool bBecameTooFew    // Сигнал о недостаче ключей на странице
)
{
   for (int i = iRemove; i <= 2 * MyConstants.nOrder; i++)
   {
      R.mData[i - 1] = R.mData[i];
      R.mLink[i] = R.mLink[i + 1];
   }
   R.nData--;
// Ключ из структуры удалён.
```

```
   if (iLevel == 0)  //   < 0 не бывает!
// Страница корневая.
      bBecameTooFew = (R.nData < 1);
   else
// Страница НЕкорневая.
      bBecameTooFew = (R.nData < MyConstants.nOrder);
}

void MergeTwoPages(
   ref MyLongRecordType R,          // Структура с левой страницей
   ref MyLongRecordType RAux,       // Структура с правой страницей
   int lStartPage,                  // Номер записи левой страницы в файле
   int iLevel,                      // Уровень
   ref bool bBecameTooFew,          // Сигнал о недостаче ключей на слитой странице
   int iLeft,                       // Номер ключа, потомков которого нужно слить
   ref int iKeyHead                 // Срединный ключ слитого списка
)
{
   int i, i1, i2, j, k;
   int lDied;
   MyLongRecordType RLong = new MyLongRecordType(0);

   lDied = -1;    // Пока что погибших страниц нет.
   ReadPage(R.mLink[iLeft], out RAux);
// Прочитали с диска левого потомка.

   i1 = RAux.nData;

   RLong.mLink[0] = RAux.mLink[0];

   j = 0;
   for (i = 1; i <= i1; i++)
   {
      j++;
      RLong.mData[j - 1] = RAux.mData[i - 1];
      RLong.mLink[j] = RAux.mLink[i];
   }
// В длинную вспомогательную структуру внесли левую из сливаемых страниц.

   ReadPage(R.mLink[iLeft + 1], out RAux);
// Прочитали с диска правого потомка.

   i2 = RAux.nData;

   j++;
```

```
    RLong.mData[j - 1] = iKeyHead;
    RLong.mLink[j] = RAux.mLink[0];

    for (i = 1; i <= i2; i++)
    {
       j++;
       RLong.mData[j - 1] = RAux.mData[i - 1];
       RLong.mLink[j] = RAux.mLink[i];
    }
// В длинную вспомогательную структуру внесли правую из сливаемых страниц.

// Итак, две соседние страницы, на которые указывали ссылки
// R.mLink[iLeft] и R.mLink[iLeft + 1],
// временно слили в одну "длинную" страницу RLong.
// А далее будет решено, действительно ли состоится слияние В ФАЙЛЕ,
// или же страница RLong просто "поровну" раздаст ключи
// между страницами, на которые указывали ссылки
// R.mLink[iLeft] и R.mLink[iLeft + 1].

// j – кол-во ключей в RLong

    if (j <= 2 * MyConstants.nOrder)
    {
       bBecameTooFew = true;
// Слияние состоится.
// Две страницы будут заменены одной.

       iKeyHead = -1;

       RAux.mLink[0] = RLong.mLink[0];
       RAux.nData = j;

       for (i = 1; i <= j; i++)
       {
          RAux.mData[i - 1] = RLong.mData[i - 1];
          RAux.mLink[i] = RLong.mLink[i];
       }

       WritePage(R.mLink[iLeft], ref RAux);
// В файл записана только одна из страниц.

       lDied = R.mLink[iLeft + 1];
// Страница с номером lDied погибла. Но в файле она бесполезно присутствует.

       ReadPage(lDied, out RAux);
```

```
        RAux.nData = 0;
// Сигнал о том, что страница в файле бесполезная.
        WritePage(lDied, ref RAux);
    }
    else
    {
        bBecameTooFew = false;
// Слияния не будет. Будет раздача поровну. Или примерно поровну.

        k = (i1 + i2 + 1) / 2;

        RAux.mLink[0] = RLong.mLink[0];

        for (i = 1; i <= k; i++)
        {
            RAux.mData[i - 1] = RLong.mData[i - 1];
            RAux.mLink[i] = RLong.mLink[i];
        }

        RAux.nData = k;

        WritePage(R.mLink[iLeft], ref RAux);
// Первая из двух страниц записана в файл.

        j = k + 1;
        iKeyHead = RLong.mData[j - 1];
        RAux.mLink[0] = RLong.mLink[j];

        i = 0;
        while (j < i1 + i2 + 1)
        {
            i++;
            j++;
            RAux.mData[i - 1] = RLong.mData[j - 1];
            RAux.mLink[i] = RLong.mLink[j];
        }

        RAux.nData = i;

        WritePage(R.mLink[iLeft + 1], ref RAux);
// Вторая из двух страниц записана в файл.
    }
```

```
    if (bBecameTooFew)
        RemoveItem(ref R, iLeft + 1, iLevel, ref bBecameTooFew);

// ДО вызова процедуры RemoveItem переменная bBecameTooFew == true,
// если слияние страниц, подчинённых записи R,  состоялось,
// bBecameTooFew == false, если не было слияния.

// ПОСЛЕ вызова процедуры RemoveItem переменная bBecameTooFew == true,
// если на САМОЙ записи R (а не на её потомках) стало слишком мало ключей,
// bBecameTooFew == false, если ключей достаточно.

    if (lStartPage == 0 && bBecameTooFew)
    {
// Единственный ключ на корневой странице удалён.
// Значит, осталась ссылка на единственного потомка.

        lDied = R.mLink[0];
        ReadPage(lDied, out RAux);
// Прочитали из файла того единственного потомка.

        R.nData = RAux.nData;
        for (i = 0; i <= 2 * MyConstants.nOrder; i++)
        {
            if (i > 0) R.mData[i - 1] = RAux.mData[i - 1];
            R.mLink[i] = RAux.mLink[i];
        }
//      R.mLink[0] = RAux.mLink[0];

// Это важно!
// Только ЗДЕСЬ содержимое стартовой страницы
// заменится содержимым её единственного потомка.

        RAux.nData = 0;
// Сигнал о том, что страница в файле бесполезная.
        WritePage(lDied, ref RAux);

        bBecameTooFew = false;
// Чтобы файл не уничтожался, когда дерево представлено
// одной только стартовой страницей!
    }

    WritePage(lStartPage, ref R);
}
```

```
public void ExcludeKey(
    int iKeyFind,                   // Ключ, который нужно удалить
    int iLevel,                     // Уровень
    int lStartPage,                 // Страница, с которой начинает работу данный экземпляр метода
    ref bool bBecameTooFew          // Сигнал о недостаче ключей на странице
)
{
    MyLongRecordType R = new MyLongRecordType(0);
    MyLongRecordType RAux = new MyLongRecordType(0);
    int i, iR;
    int lRightest;
    int sAux;
    Int64 l;

    l = FS.Length;
    if (l % (Int64)nBytes != 0) Away(11);
    lRightest = (int)(l / (Int64)nBytes);

    if (lRightest < 1) Away(12);

    if (lRightest < 1 + lStartPage) Away(13);

    ReadPage(lStartPage, out R);

    if (R.mLink[0] < 0)
    {
// Сюда попали => страница терминальная

        for (i = 1; i <= R.nData; i++)
        {
            iR = iKeyFind - R.mData[i - 1];

            if (iR == 0)
            {
                RemoveItem(ref R, i, iLevel, ref bBecameTooFew);
// ТОЛЬКО ЗДЕСЬ удаляется ключ!
// Сигнал bBecameTooFew будет передан методу-предку ExcludeKey.

                WritePage(lStartPage, ref R);
                bFound = true;
                return;
            }
        }
```

```
        bFound = false;
        return;
    }

// Сюда попали => страница НЕтерминальная

    for (i = 1; i <= R.nData; i++)
    {
        iR = iKeyFind - R.mData[i - 1];

        if (iR == 0)
        {
            bFound = true;

            lRightest = R.mLink[i - 1];
            ReadPage(lRightest, out RAux);
// В RAux – непосредственный левый потомок ключа R.mData[i].

            while (RAux.mLink[0] > -1)
            {
// Движение к самому правому потомку RAux.
// Самый правый может быть только терминальным.

                lRightest = RAux.mLink[RAux.nData];
                ReadPage(lRightest, out RAux);
            }

// Напоминание: надо удалить ключ R.mData[i],
// который находится на НЕтерминальной странице.

            sAux = R.mData[i - 1];
            R.mData[i - 1] = RAux.mData[RAux.nData - 1];
            RAux.mData[RAux.nData - 1] = sAux;
            WritePage(lStartPage, ref R);
            WritePage(lRightest, ref RAux);

// Теперь обменяны значениями
// ключи R.mData[i] и RAux.mData[RAux.nData].

// Далее надо удалять ключ RAux.mData[RAux.nData]
// из ТЕРМИНАЛЬНОЙ страницы.

// А чем это хуже?
// Тем, что дерево испорчено. Оно теперь НЕ ЕСТЬ дерево поиска.
```

```
// Почему? Доступ к удаляемому ТЕРМИНАЛЬНОМУ ключу возможен только
// от ЛЕВОЙ ссылки под R.mData[i] (а не от правой, как должно быть у дерева поиска).

// Но возможен же!
// Так что нужно получить этот доступ и удалить ключ из нового места.

            ExcludeKey(iKeyFind, iLevel + 1, R.mLink[i - 1], ref bBecameTooFew);
// Теперь дерево вновь стало нормальным деревом поиска.

            if (bBecameTooFew)
               MergeTwoPages(ref R, ref RAux, lStartPage, iLevel, ref bBecameTooFew,
                  i - 1, ref (R.mData[i - 1]));

            return;
         }
      }

      for (i = 1; i <= R.nData; i++)
      {
         iR = iKeyFind - R.mData[i - 1];

         if (iR < 0)
         {
            ExcludeKey(iKeyFind, iLevel + 1, R.mLink[i - 1], ref bBecameTooFew);
// Сигнал bBecameTooFew пришел СЮДА от дочернего экземпляра ExcludeKey
// и будет использован ЗДЕСЬ.

            if (bBecameTooFew)
               MergeTwoPages(ref R, ref RAux, lStartPage, iLevel, ref bBecameTooFew,
                  i - 1, ref (R.mData[i - 1]));
// А в методе MergeTwoPages
// переменная bBecameTooFew может измениться и она, далее,
// будет передана родительскому экземпляру ExcludeKey

            return;
         }
      }

// Сюда попали => iKeyFind > R.mData[R.nData]
// Удаляемый ключ – больше любого из ключей на данной странице.

      i = R.nData;
      ExcludeKey(iKeyFind, iLevel + 1, R.mLink[i], ref bBecameTooFew);
```

```
    if (bBecameTooFew)
        MergeTwoPages(ref R, ref RAux, lStartPage, iLevel, ref bBecameTooFew,
            i - 1, ref (R.mData[i - 1]));
}

public bool Exclude(int iKey, bool bRerortIfKeyIsNotInTree)
{
    FS = new FileStream(sFileName, FileMode.Open);

    int lFrom = 0;
    bool bBecameFew = false;

    ExcludeKey(iKey, 0, lFrom, ref bBecameFew);

    FS.Close();

    if (bBecameFew)
    {
        FileInfo file = new FileInfo(sFileName);
        if (file.Exists) file.Delete();
    }

    if (!bFound)
    {
        if (bRerortIfKeyIsNotInTree)
            MessageBox.Show("Ключ \"" + iKey.ToString() + "\" не найден!");
        return false;
    }

    mxStart[0] = xGapEdge;
    myStart[0] = yGapEdge;

    DrawTree();

    return true;
}
```