Двоичное Б-дерево

Двоичное Б-дерево (англ. Binary B-Tree) – это одно из самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов.

Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева – «направление правой ссылки». Этот атрибут может принимать одно из двух возможных значений – «горизонтально» или «наклонно вниз».

Асимптотическая сложность алгоритмов включения/удаления ключей двоичного Б-дерева есть $O(\log N)$ (при $N \to +\infty$).

Использование двоичного Б-дерева для поиска ничем не отличается от использования любого другого двоичного дерева поиска. Отличие имеется только в графическом представлении дерева, похожем на представление обычного, «сильноветвящегося» Б-дерева порядка 1.

На Рис.1 показаны фазы заполнения двоичного Б-дерева первыми несколькими узлами.

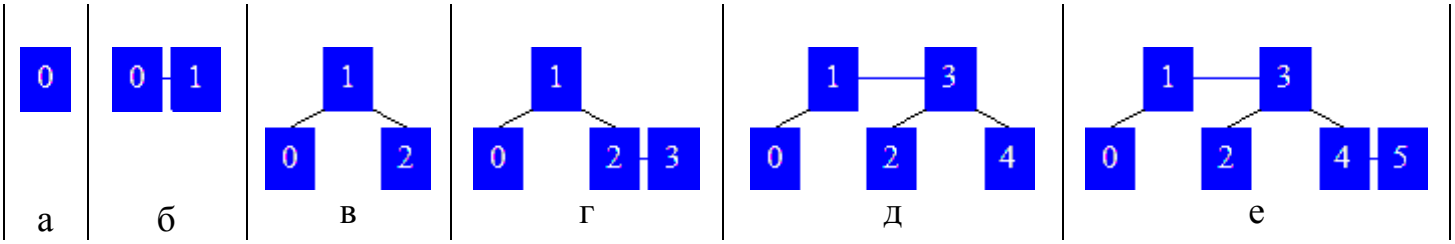

Рис. 1

С помощью двоичных узлов имитируется Б-дерево, количество ключей на странице которого равно 1 либо 2. Если страница содержит один узел (и ключ), то от неё могут быть протянуты две линии ссылок на две страницы-потомка. Если страница содержит два узла (и два ключа), от неё могут быть протянуты три линии ссылок на три страницы-потомка. В любом случае, количество ссылок на страницы-потомки, исходящих от данной страницы, на единицу больше числа ключей на ней. Отсутствие ссылок от страницы возможно только в случае, если страница терминальная.

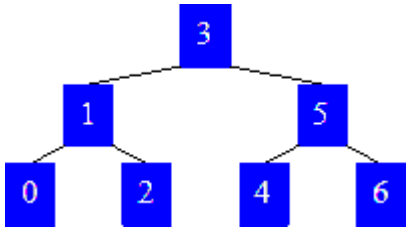

Рис. 2

Один или два ключа вмещаются на одной странице (фазы Рис. 1а, Рис. 1б). Как и для обычного Б-дерева, в этом случае принято говорить о первом уровне, на котором размещена единственная страница дерева.

Три узла вызывают переполнение страницы, за которым следует появление двух новых страниц на втором уровне. При этом высота дерева вырастает на единицу. Фазы Рис. 1б – Рис. 1е соответствуют высоте дерева, равной двум. При высоте дерева, равной двум, максимально возможное число ключей/узлов – восемь.

Однако, в ряде случаев уже седьмой ключ/узел может повлечь увеличение высоты до трех (Рис. 2).

Как и положено для Б-дерева, все терминальные страницы двоичного Б-дерева расположены на одном уровне. Высота двоичного Б-дерева равна уровню любой из его

терминальных страниц. Естественно, если исчислять высоту обычным для двоичного дерева способом, она может оказаться больше, чем высота двоичного Б-дерева. Например, высота двоичного Б-дерева в фазе Рис. 1е равна двум, а высота этого же дерева «в обычном смысле» (для двоичных деревьев) равна четырем.

Признак принадлежности узла терминальной странице – равенство нулю (null) ссылки из узла на левого потомка. Это относится и к первому, и ко второму узлу (если он есть) терминальной страницы.

Всякий путь от корня к узлу, расположенному на терминальной странице, пролегает либо через один узел, либо через два узла на очередной странице. Количество пройденных при этом страниц не зависит от пути, и равно высоте дерева. Следовательно, длины самого длинного и самого короткого путей от корня к узлу терминальной страницы могут различаться не более, чем в два раза. В этом смысле двоичное Б-дерево похоже на красно-черное дерево. Но не тождественно ему.

Операция включения в двоичное Б-дерево реализована в рекурсивном методе IncludeKey(…), которая, в свою очередь, вызывается нерекурсивным методом Include(…).

Операция удаления ключа из двоичного Б-дерева реализована в рекурсивном методе ExcludeKey(…), которая, в свою очередь, вызывается нерекурсивным методом Exnclude(…).

Используются глобальны переменные:

```
private bool bFound;              // Ключ в дереве найден
private bool bBadTree;            // Дерево требует перестройки
private MyNodeType pEmpty;        // Особенный, «Пустой» узел
private MyNodeType pAux;          // Вспомогательный узел
private MyNodeType[] mP;          // Массив цепи узлов из двух уровней
private MyNodeType[] mA;          // Массив ссылок из цепи узлов
private int nP;                   // Число узлов в цепи из двух уровней
```

Основные методы кода:

```
private void IncludeKey(
   int iKey,                  // Ключ, который следует внести в дерево
   ref MyNodeType pStart,     // Узел, возглавляющий страницу и поддерево
   ref MyNodeType pUp         // Узел, всплывающий при переполнении страницы
)
{
   MyNodeType pMain, pLast;
   int iL, iR;
   bool bTerminal, bOverflow, bHead;

   if (pRoot == null)
   {
      NewNode(ref pRoot, iKey);
      pRoot.AuxField.bHorizontal = false;
      return;
   }
```

```
    if (pStart == pRoot)
        bHead = true;
    else
        bHead = false;

    pUp = null;
    bOverflow = false;

    if (pStart.pLeft == null)
        bTerminal = true;
    else
        bTerminal = false;

    iL = iKey - pStart.iKey;

    if (iL == 0)
    {
        KeyIsAlready(iKey, "BB");
        return;
    }

    if (bTerminal)
    {
// Страница терминальная

        pMain = null;
        NewNode(ref pMain, iKey);

        if (!pStart.AuxField.bHorizontal)
        {
// Сюда попали => Один узел на странице.

            if (iL < 0)
            {
// Новый ключ - меньше ключа pStart (=> левее его)

                pMain.pRight = pStart;
                pMain.AuxField.bHorizontal = true;
                pStart = pMain;
            }
            else
            {
//  iL > 0
// Новый ключ - больше ключа pStart (=> правее его)
```

```
            pMain.AuxField.bHorizontal = false;
            pStart.pRight = pMain;
            pStart.AuxField.bHorizontal = true;
        }

        return;
    }

// Сюда попали => Два узла на странице

    if (iL < 0)
    {
        pMain.pRight = pStart;
        pMain.AuxField.bHorizontal = true;
        pStart = pMain;
        bOverflow = true;
    }
    else
    {
        pLast = pStart.pRight;

        iR = iKey - pLast.iKey;

        if (iR == 0)
        {
            KeyIsAlready(iKey, "BB");
            return;
        }

        bOverflow = true;

        if (iR < 0)
        {
// Новый ключ должен быть вставлен между ключами в pStart и pLast

            pMain.pRight = pLast;
            pMain.AuxField.bHorizontal = true;
            pStart.pRight = pMain;
        }
        else
        {
// iR > 0
// Новый ключ должен быть вставлен правее ключа pLast
```

```
                pLast.pRight = pMain;
                pLast.AuxField.bHorizontal = true;
                pMain.AuxField.bHorizontal = false;
            }
        }
// Теперь узлов на странице стало ТРИ. Страница переполнена.

    }  // if (bTerminal)
    else
    {  // !(if (bTerminal))
// Страница НЕтерминальная

        if (!pStart.AuxField.bHorizontal)
        {
// Один узел на странице

            if (iL < 0)
            {
// Новый ключ меньше ключа pStart
                IncludeKey(iKey, ref pStart.pLeft, ref pUp);
// case 1

                if (pUp != null)
                {
// Всплыл узел с двумя потомками из левой дочерней страницы.
// Его ключ меньше ключа pStart
// Новый узел встанет самым левым на странице.

                    pStart.pLeft = pUp.pRight;
                    pUp.pRight = pStart;
                    pUp.AuxField.bHorizontal = true;
                    pStart = pUp;
                    pUp = null;
// Переполнения страницы НЕТ.
                }
            }
            else
            {
// iL > 0
// Новый ключ больше ключа pStart

                IncludeKey(iKey, ref pStart.pRight, ref pUp);
// case 2

                if (pUp != null)
```

```
            {
// Всплыл узел с двумя потомками из правой дочерней страницы.
// Его ключ больше ключа pStart
// Новый узел встанет самым правым на странице.

                pStart.AuxField.bHorizontal = true;
                pStart.pRight = pUp;
                pUp = null;
// Переполнения страницы НЕТ.
            }
        }

        return;
    }

// Сюда попали => Два узла на странице

    if (iL < 0)
    {
// Новый ключ меньше ключа pStart

        IncludeKey(iKey, ref pStart.pLeft, ref pUp);
// Case 3

        if (pUp != null)
        {
// Всплыл узел с двумя потомками из левой дочерней страницы.
// Его ключ меньше ключа pStart
// Новый узел встанет самым левым на странице.

            pStart.pLeft = pUp.pRight;
            pUp.pRight = pStart;
            pUp.AuxField.bHorizontal = true;
            pStart = pUp;
            bOverflow = true;
        }
    }
    else
    {
        pLast = pStart.pRight;

        iR = iKey - pLast.iKey;

        if (iR == 0)
        {
```

```
                KeyIsAlready(iKey, "BB");
                pUp = null;
                return;
            }

            if (iR < 0)
            {
// Новый ключ - между ключами в pStart и pLast
                IncludeKey(iKey, ref pLast.pLeft, ref pUp);
// Case 4

                if (pUp != null)
                {
// Всплыл узел с двумя потомками из средней дочерней страницы.
// Его ключ больше ключа pStart и меньше ключа pLast.
// Новый узел встанет между самым левым и правым на странице.

                    pLast.pLeft = pUp.pRight;
                    pUp.pRight = pLast;
                    pUp.AuxField.bHorizontal = true;
                    pStart.pRight = pUp;
                    bOverflow = true;
                }
            }
            else
            {
// iR > 0
// Новый ключ больше ключа pLast

                IncludeKey(iKey, ref pLast.pRight, ref pUp);
// Case 5

                if (pUp != null)
                {
// Всплыл узел с двумя потомками из правой дочерней страницы.
// Его ключ больше ключа pLast.
// Новый узел встанет самым правым на странице.

                    pLast.AuxField.bHorizontal = true;
                    pLast.pRight = pUp;
                    bOverflow = true;
                }
            }
        }
```

```
    }

    if (bOverflow)
    {
// Страница переполнена, в ней три узла.
// Средний узел должен нацелиться своими связными полями
// pLeft и pRight на левый и правый узлы.
// Сам средний узел должен всплыть вверх, в родительскую страницу, если она есть.

        pMain = pStart.pRight;
        pLast = pMain.pRight;

        pStart.pRight = pMain.pLeft;
        pStart.AuxField.bHorizontal = false;
        pMain.pLeft = pStart;
        pMain.AuxField.bHorizontal = false;

        if (bHead)
            pRoot = pMain;
// Средний узел не всплывает, а становится новым корнем дерева.
// Высота дерева увеличивается на единицу.
        else
            pUp = pMain;
// Средний узел всплывает в родительскую страницу.
// В неё он помещается предыдущим экземпляром рекурсивного метода IncludeKey.
    }
}

public override void Include(int iKey)
{
    MyNodeType pUp = null;
    IncludeKey(iKey, ref pRoot, ref pUp);
//    AssignColors(Color.White, Color.Blue, Color.Blue);
}

private void MyExchange(MyNodeType pFrom)
// Обмен ключами узла pFrom и самого правого из его левых потомков.
{
    MyNodeType pCurr = pFrom.pLeft;

    while (pCurr.pRight != null) pCurr = pCurr.pRight;

    int iKey = pFrom.iKey;
    pFrom.iKey = pCurr.iKey;
    pCurr.iKey = iKey;
```

```
// Обменялись. Теперь удаляемый ключ - в pCurr.iKey
}

private void AddLowLayer()
// Добавляет в длинную цепь один-два узла из нижнего уровня.
{
   nP++;
   mA[nP] = pAux.pLeft;

   if (pAux == pEmpty) return;

   mP[nP] = pAux;

   if (pAux.AuxField.bHorizontal)
   {
      nP++;
      mP[nP] = pAux.pRight;
      mA[nP] = mP[nP].pLeft;
   }

   mA[nP + 1] = mP[nP].pRight;
   nP++;
}

private void CreateLongChain(MyNodeType pFirst)
// Создает временную длинную цепь из всех узлов последнего и предпоследнего уровня.
// Считается, что вся цепь временно помещается на предпоследний уровень.
// NP - количество звеньев цепи.
{
   nP = -1;

   pAux = pFirst.pLeft;
   AddLowLayer();

   mP[nP] = pFirst;

   if (pFirst.AuxField.bHorizontal)
   {
      MyNodeType pSecond = pFirst.pRight;

      pAux = pSecond.pLeft;
      AddLowLayer();

      mP[nP] = pSecond;
   }
```

```
    pAux = mP[nP].pRight;

    AddLowLayer();
}

private void UnwrapLongChain(ref MyNodeType pFirst)
// Разворачивает временную длинную цепь вновь в два нижних уровня.
{
    bBadTree = false;

// NP - количество звеньев цепи.

    if (nP == 7 || nP == 6)
    {
        pFirst = mP[2];

        mP[0].AuxField.bHorizontal = true;
        mP[0].pLeft = mA[0];
        mP[0].pRight = mP[1];

        mP[1].AuxField.bHorizontal = false;
        mP[1].pLeft = mA[1];
        mP[1].pRight = mA[2];

        mP[2].AuxField.bHorizontal = true;
        mP[2].pLeft = mP[0];
        mP[2].pRight = mP[4];

        mP[3].AuxField.bHorizontal = false;
        mP[3].pLeft = mA[3];
        mP[3].pRight = mA[4];

        mP[4].AuxField.bHorizontal = false;
        mP[4].pLeft = mP[3];
        mP[4].pRight = mP[5];

        if (nP == 7)
        {
            mP[5].AuxField.bHorizontal = true;
            mP[5].pLeft = mA[5];
            mP[5].pRight = mP[6];

            mP[6].AuxField.bHorizontal = false;
            mP[6].pLeft = mA[6];
```

```
        mP[6].pRight = mA[7];
    }
    else
    {
        mP[5].AuxField.bHorizontal = false;
        mP[5].pLeft = mA[5];
        mP[5].pRight = mA[6];
    }
}
else
if (nP == 5 || nP == 4)
{
    pFirst = mP[2];

    mP[0].AuxField.bHorizontal = true;
    mP[0].pLeft = mA[0];
    mP[0].pRight = mP[1];

    mP[1].AuxField.bHorizontal = false;
    mP[1].pLeft = mA[1];
    mP[1].pRight = mA[2];

    mP[2].AuxField.bHorizontal = false;
    mP[2].pLeft = mP[0];
    mP[2].pRight = mP[3];

    if (nP == 5)
    {
        mP[3].AuxField.bHorizontal = true;
        mP[3].pLeft = mA[3];
        mP[3].pRight = mP[4];

        mP[4].AuxField.bHorizontal = false;
        mP[4].pLeft = mA[4];
        mP[4].pRight = mA[5];
    }
    else
    {
        mP[3].AuxField.bHorizontal = false;
        mP[3].pLeft = mA[3];
        mP[3].pRight = mA[4];
    }
}
else
if (nP == 3)
```

```
    {
        pFirst = mP[1];

        mP[0].AuxField.bHorizontal = false;
        mP[0].pLeft = mA[0];
        mP[0].pRight = mA[1];

        mP[1].AuxField.bHorizontal = false;
        mP[1].pLeft = mP[0];
        mP[1].pRight = mP[2];

        mP[2].AuxField.bHorizontal = false;
        mP[2].pLeft = mA[2];
        mP[2].pRight = mA[3];

    }
    else
    if (nP == 2)  // Узлов меньше, чем достаточно.
    {
        bBadTree = true;

        if (pFirst == pRoot)  // Высота дерева уменьшилась на единицу.
            pRoot = mP[0];
        else
        {
            pFirst = pEmpty;
            pEmpty.pLeft = mP[0];
        }

        mP[0].AuxField.bHorizontal = true;
        mP[0].pLeft = mA[0];
        mP[0].pRight = mP[1];

        mP[1].AuxField.bHorizontal = false;
        mP[1].pLeft = mA[1];
        mP[1].pRight = mA[2];
    }
    else
        ItIsUnimpossible("Точка BB1.");
}

private void ExcludeKey(ref MyNodeType pFirst, int iKey)
{
// Сюда попали => в дереве, возглавляемым узлом pFirst,
// как минимум, два этажа.
```

```
    MyNodeType pSecond = null;

    if (pFirst.AuxField.bHorizontal) pSecond = pFirst.pRight;

    if (pFirst.pLeft == null)  // Терминальная страница.
    {
       if (pFirst.iKey == iKey)
       {
// Удалению подлежит первый узел на странице.

          bFound = true;
          NodeList.Remove(pFirst);

          if (pSecond == null)
          {
// Сюда попали => ключ/узел на странице был один.
// Теперь ключей не будет ни одного. Но будет один пустой узел.

             pEmpty.pLeft = null;
             pFirst = pEmpty;  // На странице остался один пустой узел.
             bBadTree = true;  // Дерево плохое
          }
          else
             pFirst = pSecond;
// Было два узла => остался один. Второй из двух.

          return;
       }

       if (pSecond != null)
       if (pSecond.iKey == iKey)
       {
// Удалению подлежит второй узел на странице.

          bFound = true;
          NodeList.Remove(pSecond);
          pFirst.pRight = null;
          pFirst.AuxField.bHorizontal = false;
          return;
       }

// Сюда попали => искомого ключа в дереве нет.
       KeyIsNotYet(iKey, "BB");
       return;
```

```
    }

// Сюда попали => страница НЕтерминальная.

    if (pFirst.iKey == iKey)  // Искомый ключ НЕ на терминальной странице!
    {
        MyExchange(pFirst);   // Перебросили на терминальную.
        ExcludeKey(ref pFirst.pLeft, iKey);
    }
    else
    if (iKey < pFirst.iKey)
        ExcludeKey(ref pFirst.pLeft, iKey);
// Объединить этот и предыдущий ExcludeKey(...) нельзя.
    else
    {
// Сюда попали => iKey > pFirst.iKey

        if (pSecond == null)
            ExcludeKey(ref pFirst.pRight, iKey);
        else
        {
            if (iKey == pSecond.iKey)  // Искомый ключ НЕ на терминальной странице!
            {
                MyExchange(pSecond);   // Перебросили на терминальную.
                ExcludeKey(ref pSecond.pLeft, iKey);
            }
            else
            if (iKey < pSecond.iKey)
                ExcludeKey(ref pSecond.pLeft, iKey);
            else
                ExcludeKey(ref pSecond.pRight, iKey);
        }
    }

    if (!bBadTree) return;

    CreateLongChain(pFirst);
    UnwrapLongChain(ref pFirst);
}

public void Exclude(int iKey)
{
    if (pRoot == null)
    {
        MessageBox.Show("ВВ-дерево пусто.");
```

```
        return;
    }

    if (pRoot.pLeft == null)
    {
// Сюда попали => в дереве только корневая страница. И больше ничего.

        if (pRoot.iKey == iKey)
        {
// Сюда попали => удаляемый ключ в первом узле корневой страницы.

            NodeList.Remove(pRoot);
            pRoot = pRoot.pRight;
// После этого оператора корнем дерева станет второй узел на странице, если он есть,
// или дерево станет пустым, если второго узла на странице нет.

            bFound = true;
//            SettleWithNodePositions(null);
            return;
        }

        if (pRoot.AuxField.bHorizontal)
        if (pRoot.pRight.iKey == iKey)
        {
// Сюда попали => удаляемый ключ в втором узле корневой страницы.

            NodeList.Remove(pRoot.pRight);
            pRoot.pRight = null;
            pRoot.AuxField.bHorizontal = false;
            bFound = true;
            SettleWithNodePositions(null);
            return;
        }

        KeyIsNotYet(iKey, "BB");
// Сюда попали => удалаемого ключа нет в дереве.

        return;
    }

// Сюда попали => в дереве, как минимум, два уровняя страниц.
// Метод ExcludeKey работает с двумя уровнями.

    bBadTree = false;
```

```
    ExcludeKey(ref pRoot, iKey);
    SettleWithNodePositions(null);
}
```