УДК 004.022

Построение двоичного дерева на основе модифицированной схемы хранения деревьев общего вида «left child» — «right sibling» (LCRS)

© Н.С. Гриценко, Ю.С. Белов

КФ МГТУ им. Н.Э. Баумана, Калуга, 248000, Россия

Рассмотрены схема хранения деревьев с произвольным ветвление «left child» — «right sibling» (LCRS) и модифицированная схема LCRS для построения двоичного дерева. Приведены алгоритмы создания дерева с порядком на «детях», добавления узла в модифицированную схему LCRS и подсчета числа узлов двоичного дерева LCRS.

Ключевые слова: *типы данных, структуры данных, графы, деревья, двоичные деревья.*

Одной из наиболее распространенных структур данных являются деревья. Их применяют для решения широкого круга задач: хранения, поиска и сортировки информации, в качестве элементов экспертных систем, для определения скрытых поверхностей, при отсечении невидимых частей ландшафта и трассировки лучей, в процессе создания баз данных, а также для организации доступа к пространственным данным, т. е. для индексации многомерной информации, такой, например, как географические данные с двумерными координатами (широтой и долготой). Особое место среди всех занимают двоичные. Частое использование их в прикладных задачах привело к необходимости создания механизма преобразования любого дерева в двоичное. Одним из вариантов решения данной проблемы является схема хранения дерева, или «левый ребенок» — «правый сосед» («left child» — «right sibling» (LCRS)) (рис. 1), которую применяют для деревьев с произвольным ветвлением.

Для осуществления перехода от дерева с произвольным ветвлением к двоичному в рамках данного механизма необходимо совершить следующие преобразования: «левый ребенок» каждой вершины остается тем же; «правым ребенком» становится «правый сосед» данной вершины (т. е. элемент, являющийся следующим «ребенком» одного и того же «родителя»). В результате получается дерево, каждый узел которого имеет не более двух «потомков».

Кроме осуществления перехода от дерева произвольного типа к двоичному схема LCRS после определенной модификации может также являться самостоятельным двоичным деревом [1]. Использование данной схемы оправданно в случае, когда возникает необходимость часто решать такие локальные задачи, как вывод элементов

дерева по уровням (либо вывод *k*-го уровня дерева); поиск элемента(ов) на *k*-м уровне; обход дерева по уровням; построение двоичного дерева, путь до каждого из узлов которого отличается не более, чем на единицу и др. Рассмотрим модифицированные схемы LCRS для хранения двоичного дерева (рис. 2, 3).

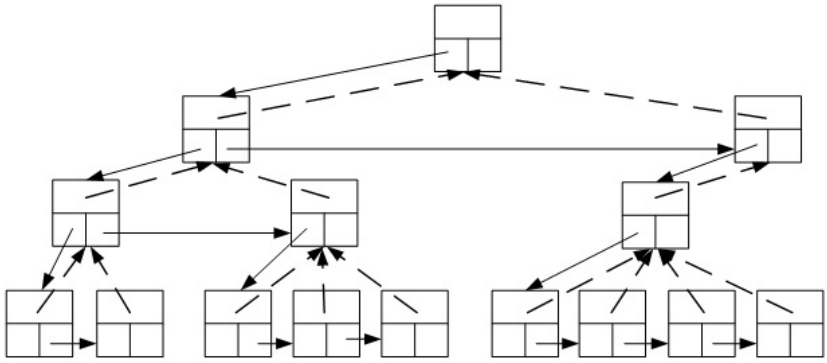

Рис. 1. Схема хранения дерева LCRS

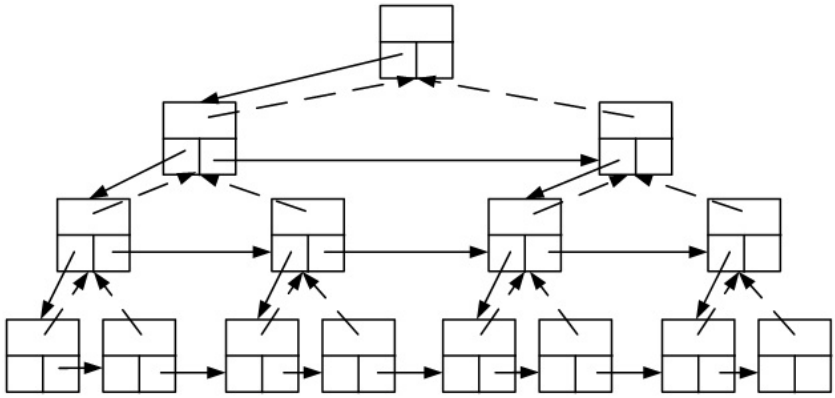

Рис. 2. Модифицированная схема LCRS для хранения двоичного дерева

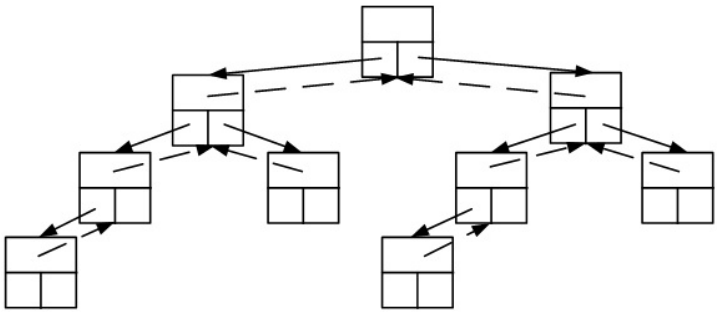

Рис. 3. Модифицированная схема LCRS для хранения двоичного дерева с порядком на «детях»

При хранении дерева каждый его узел представляется в виде структуры с полями, в которых хранятся ссылки на другие элементы дерева. При описании двоичных деревьев с порядком на «детях» таких полей обычно три: ссылка на «родителя» (parent), ссылка на «правого ребенка» («right child») и ссылка на «левого ребенка» («left child»). В некоторых случаях ссылка на «родителя» может отсутствовать, но тогда алгоритмы обработки данного дерева значительно усложняются, а некоторые вообще не представляется возможным реализовать для данного дерева. Поэтому далее будет разобрано дерево, элементы которого представлены структурой, содержащей в себе три рассмотренных выше поля.

Рекурсивный алгоритм создания дерева с числом элементов n можно описать следующим образом (рис. 4) [2]:

```
struct Tree
{
   int data;
   Tree *leftChild;
   Tree *rightChild;
};
Tree *CreateTree(int n)
{
   int nl, nr;
   Tree *current;
   if (n == 0)
      current = NULL;
   else
   {
      current = new Tree;
      nl = n/2; nr = n – nl – 1;
      cout << "Input element of a tree: ";
      cin >> current->data;
      current->leftChild = CreateTree(nl);
      current->rightChild = CreateTree(nr);
   }
   return current;
}
```

Недостатком данного алгоритма является необходимость заранее знать число элементов создаваемого дерева. Кроме того, сразу нельзя представить расположение элементов в древовидной структуре.

Теперь рассмотрим LCRS-схему хранения двоичного дерева. Она предполагает наличие у каждого узла дерева трех ссылок: на «родителя», на «левого ребенка» и на «правого соседа». Алгоритм создания дерева предполагает не создание результирующего дерева вызовом одной функции, а постепенное добавление к существующему дереву очередного узла [3].

Рис. 4. Алгоритм создания дерева

Функция добавления нового элемента вызывается с входным параметром — указателем на последний добавленный лист (leaf). Алгоритм добавления очередного узла приведен ниже (рис. 5):

```
struct BinaryTree
{  int _data;
   BinaryTree *_leftChild, *_rightSibling,
*_parent;
};
BinaryTree *AddElement(BinaryTree *leaf, int data)
{
   if(leaf == NULL)
   {  leaf = new BinaryTree;
      leaf->_data = data; leaf ->_leftChild = NULL;
      leaf ->_rightSibling = NULL;
      leaf ->_parent = NULL;
```

```
         return leaf;
      }
      if(leaf->_parent == NULL)
      {  leaf->_leftChild = new BinaryTree;
         leaf->_leftChild -> parent = leaf;
         leaf = leaf->_leftChild; leaf ->_data = data;
         leaf ->_leftChild = NULL;
         leaf->_rightSibling = NULL;
         return leaf;
      }
      if(leaf->_parent->_leftChild == leaf)
      {  leaf ->_rightSibling = new BinaryTree;
         leaf->_rightSibling->_parent = leaf->_parent;
         leaf = leaf->_rightSibling;
         leaf->_data = data; leaf->_leftChild = NULL;
         leaf->_rightSibling = NULL;
         return leaf;
      }
      else
         if(leaf->_parent->_rightSibling != NULL)
         {  leaf->_parent->_rightSibling->_leftChild  =
new BinaryTree;
          leaf->_parent->_rightSibling->_leftChild-
>_parent = leaf->_parent->_rightSibling;
          leaf->_rightSibling = leaf->_parent-
>_rightSibling->_leftChild;
          leaf=leaf->_parent->_rightSibling-
>_leftChild;
          leaf->_data = data;
          leaf->_leftChild = NULL;
          leaf ->_rightSibling = NULL;
          return leaf;
         }
         else
         {  while (leaf->_parent!=NULL)
            {  leaf = leaf ->_parent; }
            while (leaf->_leftChild!=NULL)
            {  leaf= leaf->_leftChild;
            }
            leaf->_leftChild = new BinaryTree;
            leaf->_leftChild ->_parent = leaf;
            leaf = leaf->_leftChild;
            leaf  ->_data  =  data;leaf->_leftChild  =
NULL;
            leaf->_rightSibling = NULL;
            return leaf;
         }
   }
```

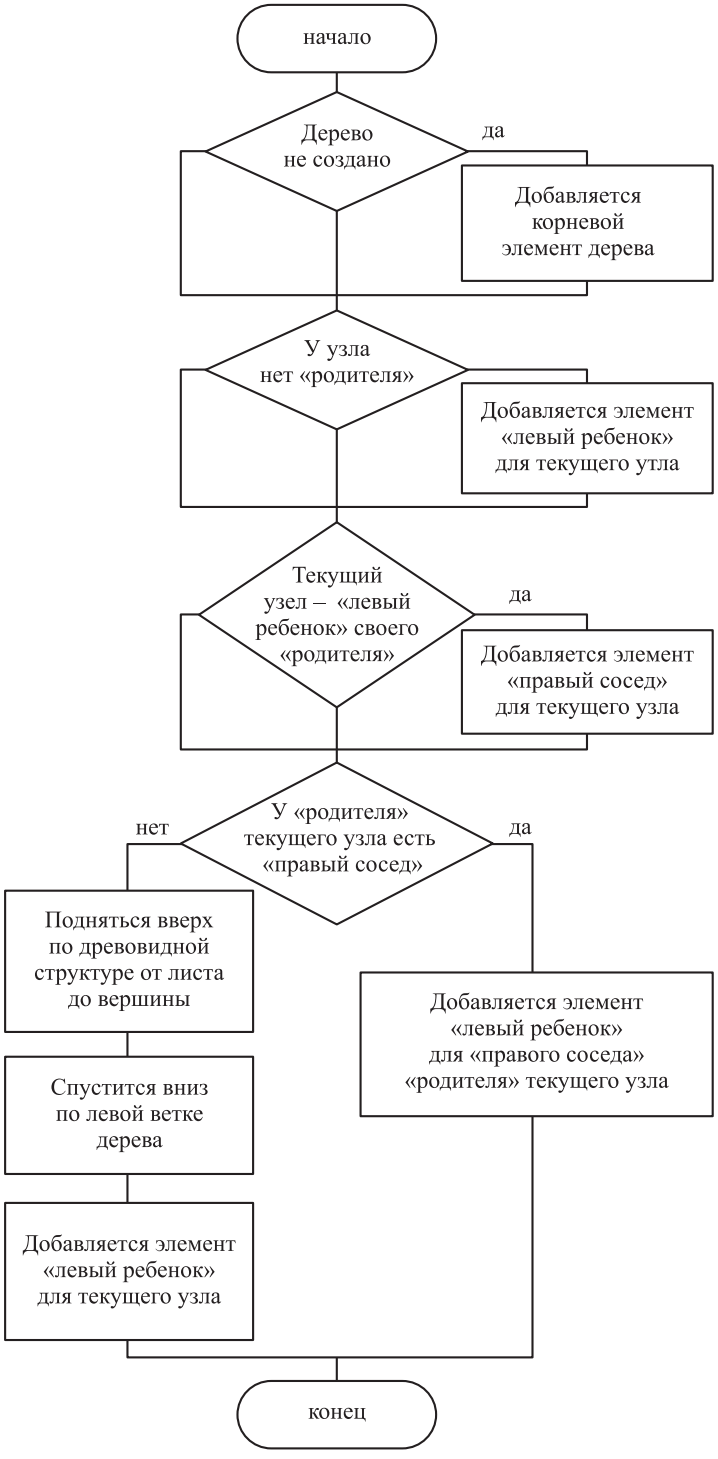

Рис. 5. Алгоритм добавления к существующему дереву очередного листа

Данный алгоритм создания дерева по схеме LCRS кажется более сложным для понимания и более объемным для написания, чем алгоритм построения двоичного дерева с порядком на «детях», но он является более гибким и легко представимым графически. Кроме того, можно точно определить, куда будет добавлен новый элемент. Данное дерево заполняется по уровням слева направо.

Число узлов дерева LCRS можно легко подсчитать следующим образом [4]:

1) спуститься по левой ветке дерева (т. е. переходя от вершины к «правому ребенку») до листового элемента;

2) при каждом переходе к «левому ребенку» номер уровня, на который выполнен переход, инкрементируется, а к сумме элементов прибавляется число «2» в степени, равной данному уровню (нулевым считать уровень, на котором расположена вершина дерева);

3) на листовом уровне п. 2 не выполняется, а совершается переход вправо по листовому уровню от самого левого листа до самого правого, используя поле «правый сосед» каждого листового узла; при каждом переходе сумма узлов дерева инкрементируется (рис. 6).

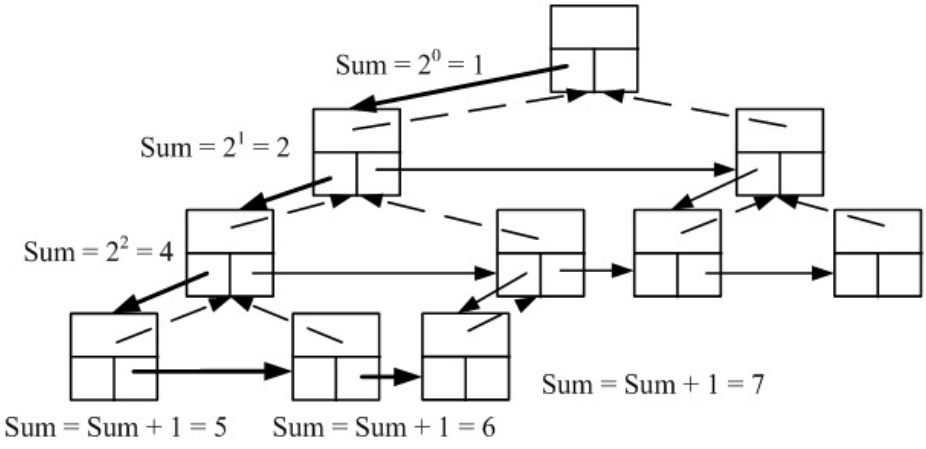

Рис. 6. Схема подсчета числа узлов дерева LCRS

Еще одним преимуществом дерева LCRS является возможность вывести все элементы *k*-го уровня с помощью несложного алгоритма. Для этого нужно спуститься по левой ветке дерева до *k*-го уровня и слева направо вывести все его элементы начиная с текущего, двигаясь вправо с помощью поля «правый сосед».

Таким образом, дерево LCRS является одним из нестандартных решений стандартной задачи о построении двоичного дерева. Данный подход не лишен недостатков, но, просуммировав все преимущества, можно сделать вывод об эффективности и оправданности его использования.

ЛИТЕРАТУРА

[1] Ахо А., Хопкрофт Д., Ульман Д. *Структуры данных и алгоритмы*. Москва, Издательский дом «Вильямс», 2003, 384 с.

[2] Вирт Н. *Алгоритмы и структуры данных.* Москва, ДМК Пресс, 2010, 272 с.

[3] Кнут Д.Э. *Искусство программирования. Генерация всех деревьев. История комбинаторной генерации.* Москва, Издательский дом «Вильямс», 2007, т. 4, вып. 4, 160 с.

Статья поступила в редакцию 05.06. 2014

Ссылку на эту статью просим оформлять следующим образом:

Гриценко Н.С., Белов Ю.С. Построение двоичного дерева на основе модифицированной схемы хранения деревьев общего вида «left child» — «right sibling» (LCRS). *Инженерный журнал: наука и инновации*, 2014, вып. 3. URL: http://engjournal.ru/catalog/it/hidden/1281.html

Гриценко Надежда Сергеевна родилась в 1993 г. Студентка кафедры «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» КФ МГТУ им. Н.Э. Баумана. Область научных интересов: информационные технологии, типы и структуры данных, деревья, графы.
e-mail: NodeshaN@yandex.ru

Белов Юрий Сергеевич родился в 1982 г. Окончил КФ МГТУ им. Н.Э. Баумана в 2006 г. Канд. физ. мат. наук, доцент кафедры «Программное обеспечение ЭВМ, информационные технологии, прикладная математика» КФ МГТУ им. Н.Э. Баумана. Область научных интересов: информационные технологии, компьютерное моделирование, интеллектуальный анализ данных. e-mail: ybs82@mail.ru

Creation of a binary tree based on the modified storage diagram of general appearance trees «left child — right sibling» (LCRS)

© N.S. Gritsenko, Yu.S. Belov

Kaluga Branch of Bauman Moscow State Technical University, Kaluga, 248000, Russia

The article examines the following aspects: tree storage diagram with arbitrary branching "Left child – right sibling" (LCRS), the modified diagram LCRS for creation of a binary tree, a tree creation algorithm with a children order, algorithm of adding a node in the modified diagram LCRS and algorithm of node quantity counting in LCRS binary tree.

Keywords: *types and data structures, graphs, trees, binary trees.*

REFERENCES

[1] Aho A., Ullman J., Hopcroft J. *Struktury dannykh i algoritmy* [Data structures and algorithms]. [in Russian]. Moscow, "Vilyams" Publ., 2003, 384 p.

[2] Virt N. *Algoritmy i struktury dannykh* [Algorithms and Data Structures]. [in Russian]. Moscow, DMK Press, 2010, 272 p.

[3] Knut D.E. *Iskusstvo programmirovaniya. Generatsiya vsekh derev'ev. Istoriya kombinatornoi generatsii* [Art of Computer Programming. Generation of all trees. History of combinatorial generation]. [in Russian]. Moscow, "Vilyams" Publ., vol. 4, iss. 4, 2007, 160 p.

Gritsenko N.S. (b. 1993) is a Bachelor degree student of the Department of Computer Software, Information Technologies, Applied Mathematics at Kaluga branch of Bauman Moscow State Technical University. Academic interests include information technology, types and data structures, graphs, trees. e-mail: NodeshaN@yandex.ru

Belov Yu.S. (b. 1982) graduated from Kaluga branch of Bauman Moscow State Technical University in 2006. Ph.D., Assoc. Professor of the Department of Computer Software, Information Technologies, Applied Mathematics at Kaluga branch of Bauman Moscow State Technical University. Research interests include information technologies, computer simulation, intellectual data analysis. e-mail: ybs82@mail.ru