УДК 004.022

Особенности структуры и функционирования двоичного дерева поиска с «барьером»

© Н.С. Гриценко, Ю.С. Белов

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

Рассмотрена структурная организация двоичного дерева поиска с «барьером». Описаны особенности его построения и функционирования. Дано графическое отображение схем хранения двоичного дерева. Показана реализация алгоритмов поиска и построения двоичного дерева.

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

К основным операциям, выполняемым с различными структурами данных, можно отнести поиск среди набора данных. Один из широко используемых для этого методов — построение двоичного дерева поиска, которое ускоряет и упрощает задачу поиска данных в определенном наборе информации, структурирует заданную информацию для более эффективного ее дальнейшего использования, а при отсутствии необходимой информации возвращает указатель на пустой элемент [1]. Однако существуют ситуации, когда это свойство дерева неуместно и вместо пустого элемента необходимо получить установленное значение. В этом случае рационально использовать дерево поиска с «барьером».

Известно, что в основе поиска с использованием двоичного дерева лежит операция сравнения, не требующая больших затрат как памяти, так и времени, поэтому предлагаемый алгоритм при большом числе элементов позволяет ускорить поиск и повысить производительность [2]. Данный метод особенно эффективен в случае, когда двоичное дерево сбалансировано, т. е. пути от вершины до всех его листьев примерно одинаковой длины. В наихудшем случае (дерево не сбалансировано) время поиска будет линейным, и теряется смысл использования дерева. В наилучшем случае (дерево сбалансировано) время поиска будет пропорционально $\log n$, где n — число элементов дерева. Обосновывается данное время тем, что длина пути от вершины до любого из листьев дерева не более чем $\log n$ [3].

Чаще всего при построении двоичного дерева поиска каждый его узел представляет собой структуру, содержащую следующие поля: указатель на левое поддерево, указатель на правое поддерево, информационное и поле-ключ, по которому будет проводиться операция сравнения. Кроме того, при построении двоичного дерева поиска необходимо соблюдать следующие требования:

1) левое и правое поддерево узла *k* должны являться двоичным деревом поиска;

2) значение ключа «левого ребенка» узла *k* должно быть меньше значения ключа самого узла *k*;

3) значение ключа «правого ребенка» узла *k* должно быть больше значения ключа самого узла *k*.

Эти требования выполнимы при условии уникальности ключа, т. е. когда значение ключа любого из узлов дерева не будет повторяться.

Схема хранения двоичного дерева поиска представлена на рис. 1, а листинг ее реализации приведен ниже [4]:

```
struct Tree
{
   int data;
   Tree *leftChild;
   Tree *rightChild;
}*top;


Tree *AddElement(Tree *top, int n)
{
   if (top == NULL)
   {
      top = new Tree;
      top->data = n;
      top->leftChild = NULL;
      top->rightChild = NULL;
   }
   else
   {
      if (n > top->data)
         top->rightChild = AddElement(top-
>rightChild, n);
      else
         top->leftChild = AddElement(top-
>leftChild, n);
   }
   return top;
}

void PrintTree(Tree *t, int n)
{
 if(t != NULL)
 {
 PrintTree(t->rightChild, n+1);
 for(int i = 0; i < n; i++)
 cout<<" ";
 cout<<t->data<<endl;
 PrintTree(t->leftChild, n+1);
 }
}
```

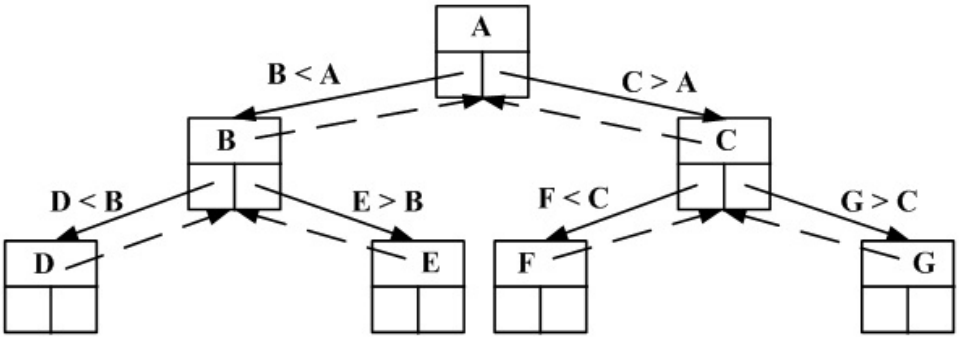


Рис. 1. Схема хранения двоичного дерева поиска

При рассмотрении алгоритма поиска в обычном двоичном дереве поиска условий окончания поиска будет два [5]:

1) найден требуемый элемент;

2) пройден путь от вершины дерева до листа, но требуемый элемент не найден.

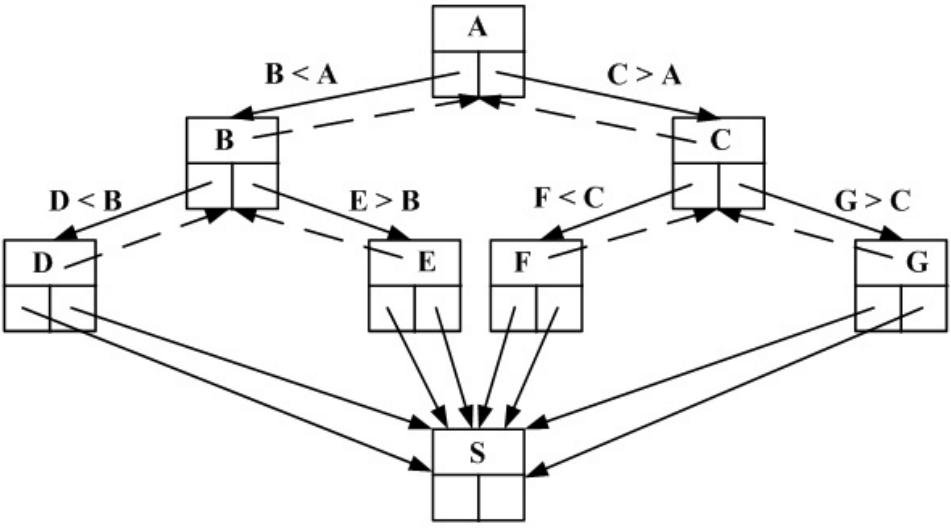
В последнем случае алгоритм вернет NULL как указатель на пустой элемент, что и будет соответствовать отсутствию искомого элемента в имеющемся дереве [6]:

```
Tree *FindElement(Tree *top, int n)
{
   if (top == NULL)
      return NULL;
   if (top->data == n)
      return top;
   else
   {
      if (n < top->data)
         top = FindElement(top->leftChild, n);
      if (n > top->data)
         top = FindElement(top->rightChild, n);
   }
}
```

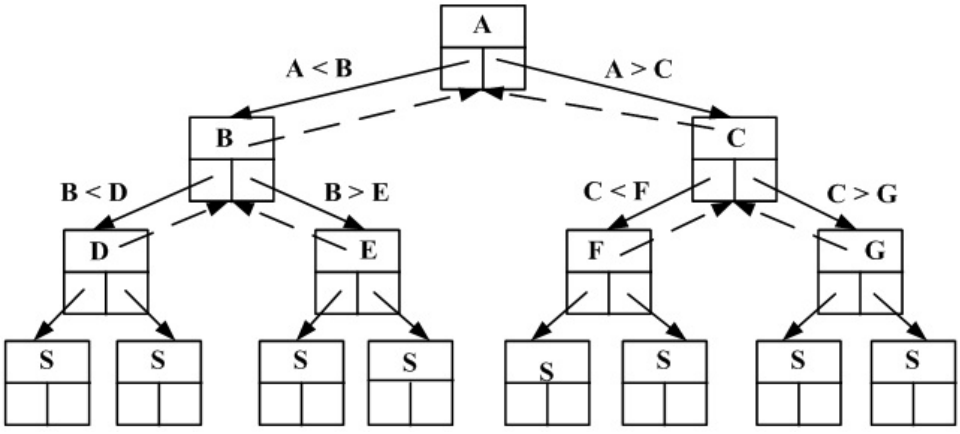
Возможна ситуация, когда при отсутствии заданного элемента в дереве необходимо вернуть на пустой элемент не указатель, а установленное значение, не зависящее от того, какой элемент требовалось найти. При решении данной задачи рационально воспользоваться двоичным деревом поиска с «барьером».

Топологически дерево поиска с «барьером» представляет собой двоичное дерево, листья которого ссылаются не на пустой, а на заранее установленный элемент, в основе которого лежит та же структура, что и в основе любого из узлов рассматриваемого дерева (рис. 2, *а*). Значения поля-ключа и информационного поля такого элемента установлены за-

ранее, а указатели на левое и правое поддерево ссылаются на пустые элементы. Таким образом, листья такого дерева лишь формально являются листьями; фактически они ссылаются еще на один узел, который и будет являться концом дерева, т. е. можно утверждать, что лист у дерева один.



а



б

Рис. 2. Схемы двоичного дерева поиска с «барьером»:
а — теоретическая интерпретация; *б* — фактическое представление

На практике двоичное дерево поиска с «барьером» структурно реализуется немного иначе. Его листья ссылаются не на один заранее установленный элемент, а на множество таких элементов, т. е. каждый лист дерева в свою очередь имеет по два листа-«барьера» (рис. 2, *б*):

```
   struct Tree
   {
      int data;
      Tree *leftChild;
      Tree *rightChild;
   }*top;

   Tree *AddElement(Tree *top, int n)
   {
      if (top == NULL)
      {
         top = new Tree;
         top->data = n;
         top->leftChild = NULL;
         top->rightChild = NULL;
      }
      else
         if   ((top->leftChild   ==   NULL)   &&   (top-
>rightChild == NULL))
         {
            top->leftChild = new Tree;
            top->leftChild->data = top->data;
            top->leftChild->leftChild = NULL;
            top->leftChild->rightChild = NULL;
            top->rightChild = new Tree;
            top->rightChild->data = top->data;
            top->rightChild->rightChild = NULL;
            top->rightChild->leftChild = NULL;
            top->data = n;
         }
         else
         {
            if (n < top->data)
               top->leftChild     =     AddElement(top-
>leftChild, n);
            else
               top->rightChild    =    AddElement(top-
>rightChild, n);
         }
         return top;
   }
```

В ходе реализации алгоритма поиска в двоичном дереве поиска с «барьером» возврата NULL никогда не будет:

```
   Tree *FindElement(Tree *top, int n, int stop)
   {
      if (top->data == stop)
         return top;
      if (top->data == n)
         return top;
```

```
      else
      {
         if (n < top->data)
            top = FindElement(top->leftChild, n,
stop);
         if (n > top->data)
            top   =   FindElement(top->rightChild,   n,
stop);
      }
   }
```

Это связано с тем, что «барьер» должен указываться при создании дерева, т. е. если дерево существует, то оно считается пустым при наличии всего лишь одного элемента — «барьера». В связи с этим при написании алгоритма условия окончания поиска будут следующие:

1) найден требуемый элемент;

2) достигнут элемент, являющийся «барьером» данного дерева.

Двоичное дерево поиска с «барьером» структурно объединило в себе двоичное дерево поиска и поиск с «барьером» в массиве. Алгоритм поиска с «барьером» в массиве заключается в том, что в конец массива к n элементам добавляется $(n + 1)$-й элемент — «барьер» и поиск продолжается до тех пор, пока не найден требуемый элемент либо не достигнут «барьер». Результаты поиска с «барьером» в неотсортированном массиве и поиска с «барьером» в двоичном дереве поиска одинаковы: либо элемент найден, либо достигнут «барьер», однако построение дерева в данном случае оправдано выигрышем во времени поиска, что особенно заметно при большом числе элементов. Недостаток заключается лишь в том, что алгоритм построения двоичного дерева поиска с «барьером» более трудоемкий, чем алгоритм построения неотсортированного массива с «барьером». Необходимо при добавлении узла в дерево учитывать его местоположение в зависимости от значения ключа, т. е. выполнять операции сравнения со значениями ключей текущего узла и ключей его поддеревьев, тогда как при построении массива элементы добавляются последовательно, независимо от значений их ключей.

Исследовав структуру двоичного дерева поиска с «барьером», можно сделать вывод, что по построению оно разительно отличается от широко известных видов деревьев. Особенности его построения вполне оправданны и уместны. Основную часть логики функционирования и программного описания данного дерева составляет обычное двоичное дерево поиска, что позволяет легче понять алгоритм его работы.

ЛИТЕРАТУРА

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

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

[3] Кормен Т., Лейзерсон Ч., Ривест Р. *Алгоритмы: построение и анализ.* Москва, МЦНМО, 2002, 906 с.

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

[5] Koffman E.B., Wolfgang P.A.T. *Objects, abstraction, data structures and desing using C++.* Wiley, 2005, 832 p.

[6] Weiss M.A. *Data structures and algorithm analysis in Java.* Prentice Hall, 2011, 640 p.

Статья поступила в редакцию 05.06.2014

Ссылку на эту статью просим оформлять следующим образом:

Гриценко Н.С., Белов Ю.С. Особенности структуры и функционирования двоичного дерева поиска с «барьером». *Инженерный журнал: наука и инновации*, 2014, вып. 4. URL: http://engjournal.ru/catalog/it/hidden/1282.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

Special features of the structures and functioning of search binary tree with «barrier»

© N.S. Gritsenko, Yu.S. Belov

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

The main purpose of the article is to examine the following aspects: structural organization of a search binary tree with a "barrier". Special features of its creation and functioning are described. A graphical mapping schemes of the binary tree store is given. We considered implementation of algorithms for searching and construction of a binary tree.

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

REFERENCES

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

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

[3] Kormen T., Leizerson Ch., Rivest R. *Algoritmy: postroenie i analiz* [Algorithms: construction and analysis]. [in Russian]. Shenia A., ed. Moscow, Moscow Center for Continuous Mathematical Education, 2002, 906 p.

[4] 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.

[5] Koffman E. B., Wolfgang P. A. T. *Objects, abstraction, data structures and desing using C++,* Wiley, 2005, 832 p.

[6] Weiss M.A. *Data structures and algorithm analysis in Java,* Prentice Hall, 2011, 640 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