УДК 004.42

О.М. Царькова

Бирская государственная социально-педагогическая академия. Россия

АЛГОРИТМ ПОСТРОЕНИЯ БИНАРНОГО ДЕРЕВА И ЕГО АНАЛИЗ

АННОТАЦИЯ

В статье описывается алгоритм построения двоичного дерева сортировки. Рассматривается реализация этого алгоритма на языке программирования Паскаль с использованием динамических структур данных.

При разработке программ для ЭВМ структура и выбор алгоритмов существенным образом зависят от структуры данных и, наоборот, решения о структурировании данных нельзя принимать без знания алгоритмов, применяемых к этим данным. То есть, строение программ и структуры данных неразрывно связаны. Известный швейцарский специалист по системному программированию, создатель языка программирования Паскаль Никлаус Вирт выразил эту взаимосвязь с помощью формулы **алгоритмы + структуры данных = программы**.

Разнообразные динамические структуры данных и алгоритмы работы с ними являются замечательными примерами, подтверждающими данную формулу. Двоичные деревья представляют собой одну из таких динамических структур. Деревья очень часто применяются в практическом программировании. Примером использования деревьев может служить сортировка с помощью дерева.

Рассмотрим, основываясь на [2], основные понятия двоичных деревьев.

Двоичное (или *бинарное*) *дерево* — это конечное множество элементов, которое либо пусто, либо содержит один элемент, называемый *корнем* дерева, а остальные элементы множества делятся на два непересекающихся подмножества, каждое из которых само является бинарным деревом. Эти подмножества называются *левым* и *правым поддеревьями* исходного дерева. Каждый элемент бинарного дерева называется *узлом* дерева.

На рис. 1 показан общепринятый способ изображения бинарного дерева. Это дерево состоит из девяти узлов, A — корень дерева. Его левое поддерево имеет корень B, а правое — корень C. Это изображается двумя ветвями, исходящими из A: левой — к B и правой — к C. Отсутствие ветви обозначает пустое поддерево. Например, левое поддерево бинарного дерева с корнем C и правое поддерево бинарного дерева с корнем E оба пусты. Бинарные деревья с корнями D, G, H и I имеют пустые левые и правые поддеревья.

Если A — корень бинарного дерева и B — корень его левого или правого поддерева, то говорят, что A — *отец* B, а B — *левый* или *правый сын* A. Узел, не имеющий сыновей (такие как узлы D, G, H и I на рис. 1), называется *листом*. Путь из корня в лист называется *ветвью*.

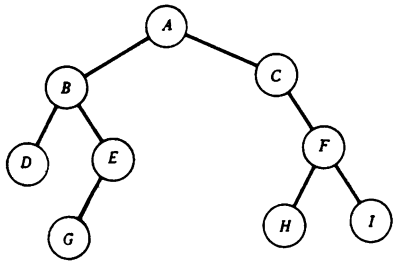


Рис. 1 Бинарное дерево

Многие алгоритмы и процессы, использующие бинарные деревья, распадаются на два этапа. На первом этапе строится бинарное дерево, а на втором оно проходится, причем оба этапа могут выполняться по-разному в зависимости от решаемой задачи. Рассмотрим в качестве примера построение бинарного дерева для решения задачи сортировки.

Пусть имеется набор чисел, которые читаются из файла или вводятся с клавиатуры. Необходимо вывести эти числа в порядке возрастания.

Для решения данной задачи двоичное дерево (которое называют *деревом сортировки*) строят следующим образом. Считывается первое число и помещается в узел, который становится корнем бинарного дерева с пустыми левым и правым поддеревьями. Для того чтобы найти позиции всех остальных элементов в бинарном дереве, в каждом узле происходит выбор левой или правой ветви в зависимости от того, меньше ли рассматриваемый элемент, чем элемент в этом узле, либо больше или равен ему. Так продолжается до тех пор, пока не будет достигнуто пустое поддерево (левое или правое). В этом случае число помещается в новый узел данного места дерева в качестве левого или правого сына.

Например, если входная последовательность была

5, 7, 4, 8, 1, 6, 2, 3, 10, 9,

то будет построено бинарное дерево, показанное на рис. 2.

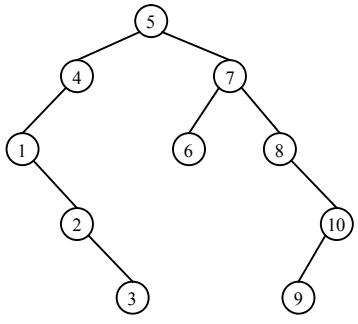


Рис. 2. Бинарное дерево, построенное для сортировки

Такое бинарное дерево обладает тем свойством, что содержимое каждого узла в левом поддереве узла *n* меньше, чем содержимое узла *n*, и содержимое каждого узла в правом поддереве узла *n* больше или равно содержимому узла *n*.

Рассмотрим реализацию рассмотренного алгоритма построения дерева на языке программирования Паскаль. Выбор данного языка объясняется тем, что он позволяет использовать сложные типы данных динамической структуры, а также обрабатывать эти данные с помощью рекурсивных процедур и функций.

Как известно, *данные динамической структуры* – это данные, внутреннее строение которых формируется по какому-либо закону, но количество элементов, их взаиморасположение и взаимосвязи могут динамически изменяться во время выполнения программы.

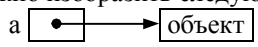
В Паскале для работы с динамическими объектами предусмотрен специальный тип значений – *ссылочный тип*. Значением этого типа является ссылка на какой-либо программный объект, по которой осуществляется непосредственный доступ к этому объекту. На машинном языке такая ссылка представляется указанием места в памяти (*адресом*) соответствующего объекта. Пустая ссылка обозначается специальным служебным словом *nil*. При этом для описания действий над динамическими объектами каждому такому объекту в программе сопоставляется статическая переменная ссылочного типа, значение которой определяется уже в процессе выполнения программы.

Например, в программу можно включить следующее описание:

```
Type   p = ^t;
Var a, b : p;
```

При этом объектами типа `p` будут ссылки на места в памяти, выделенные для объектов заданного типа `t`.

Статические ссылочные переменные часто называют *указателями*. Связь указателя с объектом схематично можно изобразить следующим образом:



Под объект отводится место в памяти только тогда, когда в этом возникает необходимость – для этого используется стандартная процедура *New*. Процедура *New(a)* выделяет область памяти соответственно тому типу, который описан для указателя *a* и записывает адрес выделенной памяти в указатель. Сам динамический объект принято обозначать *a*^. Процедура *Dispose(a)* освобождает область памяти, на которую указывает указатель *a*, после чего эта область памяти становится доступной для распределения под другие динамические переменные.

Связанные динамические данные позволяют создавать структуры данных различной конфигурации. Это достигается благодаря возможности выделять и освобождать память под элементы в любой момент времени работы программы и возможность устанавливать связь между любыми двумя элементами с помощью указателей.

Для организации связей между элементами динамической структуры данных требуется, чтобы каждый элемент содержал кроме информационных значений как минимум один указатель. Поэтому в качестве элементов таких структур необходимо использовать записи, которые могут объединять в единое целое разнородные элементы.

Представим каждый узел двоичного дерева записью, состоящей из трех полей: поле `Item` предназначено для записи в него значения, хранящегося в узле дерева и два поля (`Left` и `Right`) – для указателей на левого и правого сына данного узла. Дерево представляется указателем на корень.

```
Type Edge=^Node;
     Node=Record
          Item:integer;
          Left,Right:Edge;
          End;
```

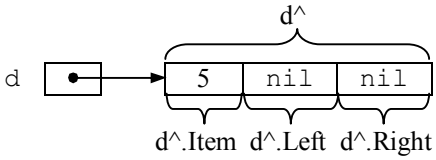
Объектами типа `Node` являются записи, в которых каждое из полей `Left` и `Right` есть либо *nil*, либо ссылка на конкретное место в памяти, отведённое с помощью *new* для объекта типа `Node`. После описания переменной

```
Var d : Node;
```

и выполнения команд

```
New(d);
d^.Item := 5;
d^.Left := nil;
d^.Right := nil;
```

результат может быть представлен в виде следующей схемы:



Дерево в программе на Паскале можно представить в виде множества объектов типа `Node`, связанных ссылками. Сами эти объекты будут узлами дерева, а ссылки на места в памяти, отведённые для объектов типа `Node` – рёбрами дерева. Если при этом поле `Left` (`Right`) некоторого объекта типа `Node` есть *nil*, то это означает, что в дереве из данной вершины не исходит ребро, направленное влево вниз (вправо вниз). На рис.3 представлено дерево, изображённое на рис.1, в памяти вычислительной машины.

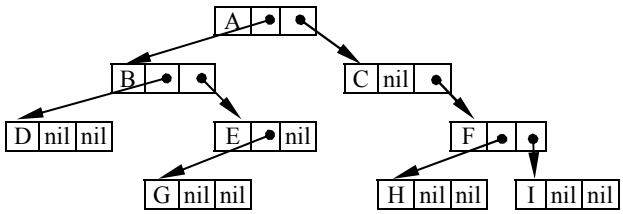


Рис.3. Представление дерева на рис.1 в памяти ЭВМ

Пусть значением переменной *v* типа `Edge` является ссылка на место в памяти, отведённое для

объекта типа Node (то есть на некоторую вершину дерева). Тогда выполнение присваивания v:=v^.Left (v:=v^.Right) означает переход к вершине, расположенной непосредственно слева внизу (справа внизу) от данной (если, конечно, соответствующее поле данной вершины не есть *nil*). Таким способом можно передвигаться по дереву от вершины к вершине сверху вниз. Включение новой вершины в дерево представляет собой изменение значений полей ссылочного типа некоторых вершин данного дерева.

Вместе с каждым деревом рассматривается переменная, значением которой является ссылка на корень дерева. Если в дерево не входит ни одной вершины, то значение этой переменной равно *nil*.

Описанный рекурсивный алгоритм формирования дерева сортировки, таким образом, можно реализовать с помощью процедуры

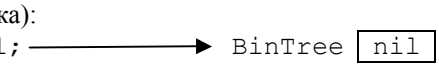
```
Procedure InsNode(Var Tree : Edge;
                      x : Integer);
Begin
  If Tree = nil
  Then Begin
       { формирование узла дерева }
        New(Tree);
        Tree^.Item := x;
        Tree^.Left := nil;
        Tree^.Right := nil
       End
  Else If x < Tree^.Item
      { определение места
        вставляемого узла }
        Then InsNode(Tree^.Left, x)
        Else InsNode(Tree^.Right, x)
End;
```

Проанализируем выполнение данной процедуры, взяв в качестве примера исходный массив (5, 7, 4, 8, 1, 6, 2, 3, 10, 9).

```
Const n=10;
      A:Array[1..n] of Integer=
              (5,7,4,8,1,6,2,3,10,9);
Var BinTree : Edge; I : Integer;
```

Начальное значение ссылки на корень дерева – nil (пустая ссылка):

```
BinTree:=nil;
```



Формирование дерева выполняется с помощью цикла:

```
For i:=1 to n do InsNode(BinTree,a[i]);
```

Проанализируем несколько шагов работы цикла. На первом шаге цикла происходит вызов процедуры InsNode(BinTree,5). Так как дерево пока пусто, то будет сформирован корень и процедура вернет после выполнения ссылку на этот корень:

```
Procedure InsNode(Var Tree: Edge; x: Integer);
{ Tree = nil; x=5 }
Begin
  If Tree = nil
   Then Begin
         New(Tree);
         Tree^.Item := x;
         Tree^.Left := nil;
         Tree^.Right := nil
        End
   Else If x < Tree^.Item
          Then InsNode(Tree^.Left, x)
          Else InsNode(Tree^.Right, x)
End;
```

Tree (nil)

Tree = nil == True

Tree^

Tree ? ? ?

Tree^

Tree 5 nil nil

BinTree := Tree

2-й шаг цикла (InsNode(BinTree,7)):

```
Procedure InsNode(Var Tree: Edge; x: Integer);
{ Tree1 = BinTree; x=7 }
Begin
  If Tree = nil
   Then Begin
         New(Tree);
         Tree^.Item := x;
         Tree^.Left := nil;
         Tree^.Right := nil
        End
   Else If x < Tree^.Item

          Then InsNode(Tree^.Left, x)
          Else InsNode(Tree^.Right, x)
End;
```

Tree1^

Tree1 5 nil nil

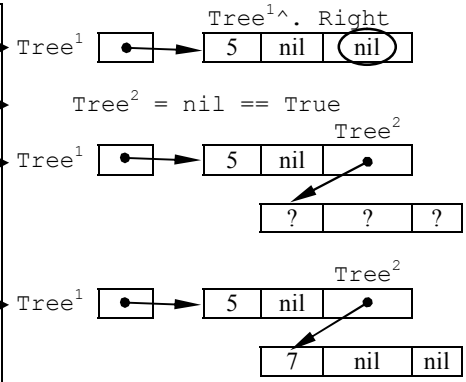
Tree1 = nil == False

x<Tree1^.Item ==
7<5 == False

InsNode(Tree1^.Right,x)

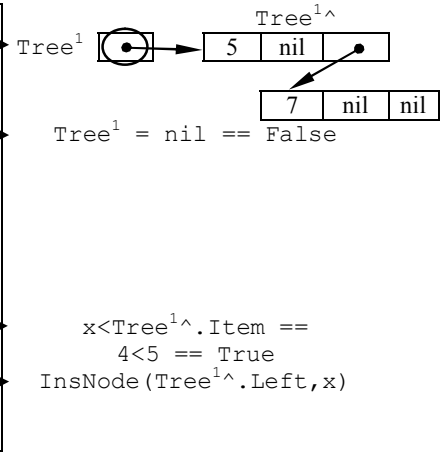
Рекурсивный вызов процедуры InsNode(Tree^.Right, x) для формирования правого сына текущего узла:

```
Procedure InsNode(Var Tree: Edge; x: Integer);
{ Tree2 = Tree1^.Right = nil; x=7 }
Begin
  If Tree = nil
   Then Begin
          New(Tree);


          Tree^.Item := x;
          Tree^.Left := nil;
          Tree^.Right := nil

        End
   Else If x < Tree^.Item
          Then InsNode(Tree^.Left, x)
          Else InsNode(Tree^.Right, x)
End;
```



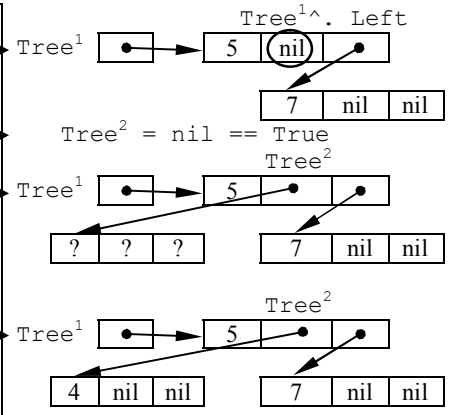
3-й шаг цикла (`InsNode(BinTree,4)`)

```
Procedure InsNode(Var Tree: Edge; x: Integer);
{ Tree1 = BinTree; x=4 }
Begin

  If Tree = nil
   Then Begin
          New(Tree);
          Tree^.Item := x;
          Tree^.Left := nil;
          Tree^.Right := nil
        End
   Else If x < Tree^.Item

          Then InsNode(Tree^.Left, x)
          Else InsNode(Tree^.Right, x)
End;
```

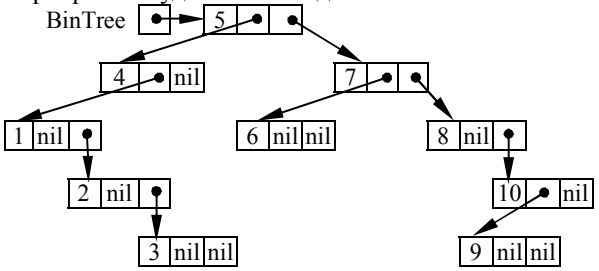


Рекурсивный вызов процедуры `InsNode(Tree^.Left, x)` для формирования левого сына текущего узла::

```
Procedure InsNode(Var Tree: Edge; x: Integer);
{ Tree2= Tree1^.Left = nil; x=4 }
Begin

  If Tree = nil
   Then Begin
          New(Tree);


          Tree^.Item := x;
          Tree^.Left := nil;
          Tree^.Right := nil

        End
   Else If x < Tree^.Item
          Then InsNode(Tree^.Left, x)
          Else InsNode(Tree^.Right, x)
End;
```



Следующие шаги цикла выполняются аналогичным образом. Построенное дерево сортировки будет иметь вид:



СПИСОК ЛИТЕРАТУРЫ

1. Вирт Н. Алгоритмы + структуры данных = программы / Н.Вирт М.: Мир, 1985.

2. Лэнгсам Й., Огенстайн М., Тененбаум А. Структуры данных для персональных ЭВМ: Пер. с англ. Мир 1989. - 568 с.

3. Ф.А. Новиков: Дискретная математика для программистов. СПб.: Питер, 2007. - 368 с. Учебник для вузов. Второе издание.