ДЕРЕВЬЯ БИНАРНОГО ПОИСКА: МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ УРОКА

КУДРИНА Е.В., ОГНЕВА М.В

г. Саратов, Саратовский государственный университет им. Н.Г. Чернышевского

Анализируя примерную программу среднего (полного) общего образования по информатике и информационным технологиям профильного уровня [1] можно сделать вывод, что основную часть фундаментального содержания информатики составляют разделы, относящиеся к дискретным объектам и процессам. Строящиеся там математические модели опираются на такие важные понятия как списки, деревья и графы, которые находят практическое применение в процессах обработки информации не только в сфере современных информационно-коммуникационных технологиях, но в биологических и социальных системах, в человеческой психике. Например, математическая модель «дерево» находит применение в двоичном кодировании и сжатии информации (дерево Хаффмана), в построении выигрышных стратегий в играх (дерево игры с полной информацией), в реализации иерархических информационных моделей (генеалогические и организационные диаграммы).

В связи с большой практической значимостью деревьев нами предлагается методическая разработка, предназначенная для преподавателей информатики, которая будет полезна для подготовки к занятиям по изучению математической модели дерева, в частности дерева бинарного поиска, и ее программной реализации на языке Pascal. Предложенная методическая разработка входит в состав авторского цикла элективных курсов «Структуры данных и алгоритмы» [2], реализуемого на базе МОУ «Медико-биологический лицей» г.Саратова в классах математико-информационного профиля, а также в состав базовых курсов по циклу компьютерные науки, реализуемых на механико-математическом факультете Саратовского госуниверситета.

Основные определения

Будем определять дерево как конечное множество T, состоящее из одного или более узлов, таких, что: 1) имеется один специально обозначенный узел, называемый корнем данного дерева; 2) остальные узлы (исключая корень) содержатся в $n \geq 0$ попарно не пересекающихся множествах $T_1, T_2, \ldots, T_n$, каждое из которых в свою

очередь является деревом (деревья $T_1, T_2, \ldots, T_n$ называются поддеревьями данного корня).

Из данного определения следует, что каждый узел дерева является *корнем* некоторого поддерева, которое содержится в этом дереве. Число поддеревьев данного узла называется *степенью* этого узла. Узел с нулевой степенью называется *листом*. *Уровень* узла по отношению к дереву T определяется следующим образом: говорят, что корень имеет уровень 1, а другие узлы имеют на единицу выше их уровня относительно содержащего их поддерева T_j этого корня. Каждый корень является *отцом* (родителем) корней своих поддеревьев; последние являются *сыновьями* (детьми) своего отца. Каждый узел (кроме корня) имеет одного родителя и произвольное число сыновей. Корень не имеет родителя, листы не имеют сыновей.

Рассмотрим дерево, представленное на рис.1. Оно имеет корень A и 5 листьев: H, J, D, G, F. Степени вершин этого дерева следующие: A, B имеют степень 2, C – 3, E – 1. Корень A располагается на 1 уровне, узлы B, C – на втором, H, J, D, E, F – на третьем, G – на четвертом.

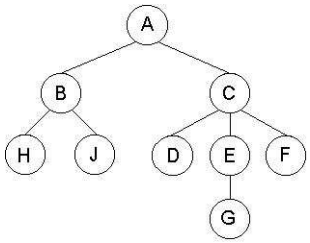

Рис. 1. Пример дерева.

Путем из узла n_1 в узел n_k называется последовательность узлов $n_1, n_2, \ldots, n_k$, где $\forall i, 1 \le i < k$, узел n_i является родителем узла n_{i+1}. Если существует путь из узла n_1 в узел n_k, то n_1 называется предком n_k, n_k - потомком n_1. *Длиной* пути называется число на 1 меньше числа узлов, составляющих этот путь.

Высотой узла дерева называется длина самого длинного пути из этого узла до какого-либо листа. Высота дерева совпадает с высотой корня. Например, для дерева, изображенного на рис. 1, высота узла H равна 0, высота узла B – 1, узла C – 2, высота дерева – 3.

Глубина узла определяется как длина пути от корня до этого узла. Например, для дерева, изображенного на рис. 1, глубина узла H равна 2, глубина узла B – 1, узла G – 3.

Бинарное дерево – это дерево, в котором каждый узел имеет не более двух поддеревьев. В этом случае будем различать левое и правое поддерево.

Дерево двоичного поиска – это бинарное дерево, узлы которого помечены элементами множества. Определяющее свойство дерева двоичного поиска заключается в том, что все элементы, хранящиеся в узлах левого поддерева любого узла x, меньше элемента, содержащегося в узле x, а все элементы, хранящиеся в узлах правого поддерева узла x, больше элемента, содержащегося в узле x. Это свойство называется характеристическим свойством дерева двоичного поиска и выполняется для любого узла дерева двоичного поиска, включая его корень.

Рассмотрим подробно реализацию деревьев бинарного поиска с помощью указателей см. рис 2.

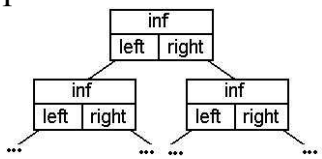

Рис. 2. Реализация дерева бинарного поиска с помощью указателей

Структура дерева содержит базовый элемент, в котором есть: 1) информационное поле inf, которое может быть любого типа, кроме файлового, и будет использоваться для хранения значений узлов дерева, например, целых чисел, строк, записей; 2) ссылочные поля left и right, которые являются указателями на левое и правое поддерево данного узла, и которые будет использоваться для организации связи узлов дерева.

Предложенная структура может быть описана следующим образом:

```
type ttree=^tree;
     tree=record  inf: integer;  left, right: ttree; end;
```

Построение дерева бинарного поиска

Рассмотрим подпрограмму Add, которая добавляет новый узел в дерево так, чтобы формировалось дерево бинарного поиска. Она имеет два формальных параметра: x – информация, которая записывается в новый узел; t – указатель на текущий узел дерева (вначале на корень исходного дерева).

Новый узел должен быть сформирован либо как корень дерева (если дерево было до этого пустое), либо в виде левого или правого сына сформированного раньше узла дерева, у которого этот сын отсутствует. Определение места для вставки нового узла производится на основе значения указателя t:

1) если дерево пусто или найдено место для нового узла, то выделяется оперативная память для нового узла и в нее записывается содержимое нового узла. Сыновей у этого узла нет, поэтому ссылки на них полагаются пустыми;

2) если дерево не пустое, то определяем положение нового узла в этом дереве:
   a) если добавляемое значение меньше значения данного узла, то поиск места новой записи продолжается по левому поддереву данного узла; для этого производится рекурсивный вызов подпрограммы Add для левого потомка;
   b) если добавляемое значение больше значения данного узла, то поиск места новой записи продолжается по правому поддереву данного узла; для этого производится рекурсивный вызов подпрограммы Add для правого потомка.

Листинг подпрограммы выглядит следующим образом:

```
procedure Add (var t:ttree; x:integer);
begin
 if t=nil
  then begin new(t); t^.inf:=x; t^.left:=nil; t^.right:=nil; end
  else if x<t^.inf then Add(t^.left, x)
                   else if x>t^.inf then Add(t^.right, x);
end;
```

Для формирования дерева в основной программе можно написать обращение к этой подпрограмме на этапе ввода в цикле узлов дерева с клавиатуры или считывания их из файла. Например, если мы будем вводить с клавиатуры узлы 10, 7, 25, 31, 18, 6, 3, 12, 22, 8, то получим дерево, представленное на рис. 3.

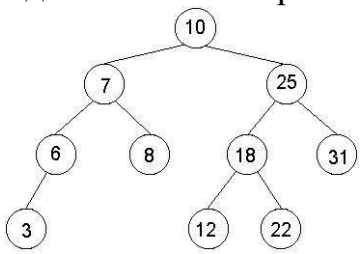

Рис. 3. Дерево бинарного поиска

Обходы бинарных деревьев

Обойти дерево – это побывать в каждом из его узлов точно по одному разу. Рассмотрим три наиболее часто используемых способов обхода бинарных деревьев – это обход в прямом, симметричном и обратном порядке. Все три обхода будем определять рекурсивно.

Алгоритм прямого обхода: попасть в корень; обойти левое поддерево данного корня; обойти правое поддерево данного корня.

Подпрограмму, составляющую список узлов дерева при обходе его в прямом порядке, можно записать следующим образом:

```
procedure preorder (t:ttree);
begin
 if t<> nil
```

```
  then begin write(t^.inf,' '); preorder(t^.left); preorder(t^.right); end;
end;
```

Алгоритм симметричного обхода: обойти левое поддерево данного корня; попасть в корень; обойти правое поддерево данного корня.

Подпрограмму, составляющую список узлов дерева при обходе его в симметричном порядке, можно записать следующим образом:

```
procedure inorder (t:ttree);
begin
 if t<> nil
  then begin inorder (t^.left); write(t^.inf,' '); inorder (t^.right); end;
end;
```

Алгоритм обратного обхода: обойти левое поддерево данного корня; обойти правое поддерево данного корня; попасть в корень.

Подпрограмму, составляющую список узлов дерева при обходе его в обратном порядке, можно записать следующим образом:

```
procedure postorder (t:ttree);
begin
 if t<> nil
  then begin postorder (t^.left); postorder (t^.right); write(t^.inf,' '); end;
end;
```

Рассмотрим обходы на примере дерева, изображенного на рис.3. При прохождении в прямом порядке список узлов выглядит следующим образом: 10 7 6 3 8 25 18 12 22 31. При прохождении в симметричном порядке список узлов выглядит следующим образом: 3 6 7 8 10 12 18 22 25 31. При прохождении в обратном порядке список узлов выглядит следующим образом: 3 6 8 7 12 22 18 31 25 10.

Замечание. При симметричном обходе дерева бинарного поиска на экран выводится упорядоченная по возрастанию последовательность данных. Этот свойство дерева бинарного поиска можно использовать для сортировки данных.

Поиск по дереву

Рассмотрим функцию Search, предназначенную для поиска узла с заданным значением в дереве. Функция имеет два формальных параметра: x – значение, которое нужно найти; t – указатель на анализируемый узел (вначале tr указывает на корень дерева). В качестве результата функция возвращает указатель на искомый узел дерева, или nil если узла с искомым значением в дереве нет.

Поиск происходит путем рекурсивного вызова подпрограммы search до тех пор, пока не будет найден искомый узел или значение t не станет равно nil. Если t=nil, то либо дерево пустое, либо

просмотрены все узлы дерева и искомый узел не найден. В этом случае функция возвращает значение nil. В противном случае производится анализ значения текущего узла сравнением его с x – искомым значением:

1) если x<tr^.inf, то поиск продолжается по левому поддереву данного узла; для этого производится рекурсивный вызов функции search для левого потомка;
2) если x>tr^.inf, то поиск продолжается по правому поддереву данного узла; для этого производится рекурсивный вызов функции search для правого потомка;
3) если не выполняется ни условие п.1, ни условие п.2, это означает, что искомое значение равно значению данного узла и в качестве ответа функция возвращает указатель на искомый узел дерева.

Листинг подпрограммы выглядит следующим образом:

```
function  search (t:ttree; x:integer):ttree;
begin
 if t=nil then search:=nil
  else if x<t^.inf then search:=search(t^.left, x)
       else if x>t^.inf then search:=search(t^.right, x)
            else search:=t;
end;
```

Удаление узла из дерева [3]

Рассмотрим процедуру Del, предназначенную для удаления данного узла из дерева. Удаление производится таким образом, что дерево остается деревом бинарного поиска. Подпрограмма имеет два формальных параметра: x –значение удаляемого узла; t – указатель на анализируемый узел (вначале – на корень дерева). В процессе поиска удаляемого узла дерева могут быть 3 случая: 1) удаляемого узла в дереве нет; 2) удаляемый узел имеет не более одного сына; 3) удаляемый узел имеет двух сыновей.

Поиск по дереву производится путем обхода по левому или правому поддереву данного узла в зависимости от значения данного узла и значения удаляемого узла с использованием рекурсивного вызова подпрограммы del. Если в результате этого поиска обнаружено, что указатель t=nil, то удаляемого узла в дереве нет, и выдается сообщение об ошибке. В противном случае:

1) если x<t^.inf, то поиск продолжается по левому поддереву данного узла; для этого производится рекурсивный вызов процедуры Del для левого поддерева;

2) если x>t^.inf, то поиск продолжается по правому поддереву данного узла; для этого производится рекурсивный вызов процедуры Del для правого поддерева;
3) если не выполняется ни условие п.1, ни условие п.2, это означает, что значение удаляемого узла равно значению данного узла; начинается удаление.

Сначала запоминается указатель на найденный удаляемый узел q=t. Удаление найденного узла производится по-разному, в зависимости от того, одного или двух сыновей имеет этот узел:

1) если t^.left=nil, у этого узла нет левого поддерева; в этом случае производится исключение этого узла из дерева установкой нового значения указателя t: t=t^.right;
2) если t^.left<>nil, т.е. у данного узла есть левое поддерево, то производится анализ наличия правого поддерева сравнением t^.right и nil:
   a) если t^.right=nil, то есть у этого узла нет правого поддерева, а есть только левое, производится исключение этого узла из дерева установкой нового значения указателя t: t=t^.left;
   b) если t^.right<>nil, то есть у этого узла есть и левое, и правое поддерево, вызывается del1(t^.left) – подпрограмма для поиска замены удаляемому узлу.

Если удаляемый узел имеет две ветви, надо найти такой узел дерева, который можно перенести на место удаляемого. В качестве такого узла можно выбрать самый правый узел левого поддерева удаляемого узла или самый левый узел правого поддерева (мы будем применять первый вариант).

Для поиска и переноса этого узла вызывается вспомогательная рекурсивная процедура Del1. Формальными параметрами подпрограммы del1 являются: t – указатель на узел в дереве, значение информационного поля которого соответствует значению удаляемого элемента; tr – указатель, который используется для поиска самого правого узла в левом поддереве:

1) если tr^.right<>nil, поиск узла для перемещения продолжается с помощью рекурсивного вызова этой же процедуры del1(t,tr^.right);
2) в противном случае, узел для переноса найден, производится его перенос на место удаляемого. Для этого:
   a) копируется его содержимое на место удаляемого узла t^.inf=tr^.inf
   b) сохраняется указатель на перемещаемый узел q=tr;
   c) исключается из дерева перемещаемый узел tr=tr^.right;
   d) удалятся элемент q.

Листинг подпрограмм выглядит следующим образом:

```
procedure del1 (t:ttree; var tr:ttree);
 var q:ttree;
begin
 if tr^.right<>nil then del1(t, tr^.right)
  else begin t^.inf:=tr^.inf; q:=tr; tr:=tr^.left; dispose(q) end;
end;
procedure del (var t: ttree; x:integer);
 var q:ttree;
begin
 if t=nil then writeln('error')
  else if x<t^.inf then del(t^.left,x)
      else if x>t^.inf then del(t^.right,x)
          else begin q:=t;
                  if t^.left=nil then begin t:=t^.right; dispose(q); end
                    else if t^.right=nil then begin t:=t^.left; dispose(q); end
                         else del1(t,t^.left);
               end
end;
```

После удаления узла со значением 25 из дерева, представленного рис. 3, получим дерево рис.4. А после удаления из этого же дерева узла со значением 6 получим дерево представленное на рис.5.

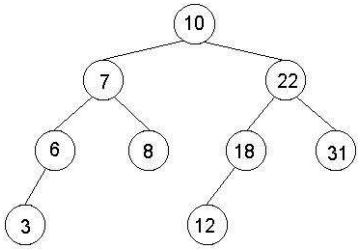

Рис. 4

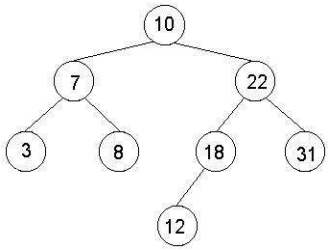

Рис. 5

Литература

1) Примерная программа среднего (полного) общего образования по информатике и информационным технологиям. Профильный уровень.// Первое сентября. Информатика.- 2007. - №2.
2) Огнева М.В., Кудрина Е.В. Turbo Pascal: списки, деревья, графы: Учеб. пособие. - Саратов: Изд - во “Научная книга”, 2006.
3) Климова Л.М. PASCAL 7.0. Практическое программирование. Решение типовых задач. – М.: КУДИЦ-ОБРАЗ, 2000