ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени академика С. П. КОРОЛЕВА» (САМАРСКИЙ УНИВЕРСИТЕТ)

НЕЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ В C#. ДЕРЕВЬЯ

Самара 2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ имени академика С. П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

НЕЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ В C#. ДЕРЕВЬЯ

Составитель *Е.В. Симонова*

С А М А Р А
Издательство Самарского университета
2017

УДК 519.876.5

ББК 22.18я73

Составитель ***Е.В. Симонова***

Рецензент: канд. техн. наук, доц. Л.С. З е л е н к о

Нелинейные динамические структуры данных в C#. Деревья: [Электронный ресурс]: метод. указания / *Е.В. Симонова*. – Самара: Изд-во Самарского университета, 2017. – 32 с. : ил. Электрон. текстовые и граф. дан. (Кбайт).- 1 эл. опт. диск (CD-ROM)

Методические указания содержат теоретические сведения по организации списковых структур, большое количество программных фрагментов, реализующих алгоритмы обработки деревьев, а также варианты заданий для выполнения лабораторных и самостоятельных работ.

Предназначены для студентов направления 09.03.01 – «Информатика и вычислительная техника» в качестве методических указаний по курсу «Программирование».

Подготовлены на кафедре информационных систем и технологий.

УДК 519.876.5
ББК 22.18я73

© Самарский университет, 2017

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ………….…………………………………………… 5
ВВЕДЕНИЕ………………..……………………………………………… 5
1. ОРГАНИЗАЦИЯ ИЕРАРХИЧЕСКИХ НЕЛИНЕЙНЫХ ДРЕВОВИДНЫХ СТРУКТУР ДАННЫХ ……………………………… 6
2. ВИДЫ БИНАРНЫХ ДЕРЕВЬЕВ……………………………. ………… 15
3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ……………………………… 27
4. КОНТРОЛЬНЫЕ ВОПРОСЫ…………………………………………. 27
5. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ…………………………………… 28
ЗАКЛЮЧЕНИЕ ………..…………………………………………………… 30
БИБЛИОГРАФИЧЕСКИЙ СПИСОК……………..………………………. 30

ПРЕДИСЛОВИЕ

В методических указаниях описаны структуры данных и алгоритмы, которые широко используются при решении разнообразных задач в широком спектре предметных областей.

Подробно рассматриваются особенности организации нелинейных рекурсивных иерархических древовидных структур данных. Рассматриваются алгоритмы обработки наиболее распространенных древовидных структур, таких, как сбалансированные деревья, дихотомические деревья, деревья выражений.

Примеры программ, описывающих алгоритмы обработки структур данных различных типов, реализованы на языке C# версии 3.0 с использованием Visual Studio 2008 .NET Framework 3.5.

Методические указания предназначены для студентов, обучающихся по направлению 09.03.01 – Информатика и вычислительная техника.

Содержание методических указаний соответствует разделам рабочей программы по дисциплине «Программирование» федерального компонента ГОС подготовки бакалавров по направлению 09.03.01 – Информатика и вычислительная техника.

ВВЕДЕНИЕ

Цель лабораторной работы – изучение теоретических основ построения и методов программной реализации нелинейных динамических структур данных – деревьев. Деревья используются для представления сложных отношений между объектами и описывают иерархически организованные упорядоченные множества объектов.

1. ОРГАНИЗАЦИЯ ИЕРАРХИЧЕСКИХ НЕЛИНЕЙНЫХ ДРЕВОВИДНЫХ СТРУКТУР ДАННЫХ

1.1. Деревья общего вида (произвольной степени)

Нелинейные структуры данных выражают более сложные отношения порядка между объектами, чем отношения предшествования и следования. Наиболее важным видом нелинейных структур являются **деревья**. Древовидные структуры позволяют определить такие отношения, как предок, потомок, брат и т.п.

Дерево (непустое) – конечное множество объектов *T*, состоящее из одного или более узлов, для которых выполняются следующие условия:

- имеется один специально выделенный узел, называемый корнем данного дерева;
- остальные узлы (исключая корень) содержатся в *m* попарно непересекающихся множествах *T1, ...,Tm,* каждое из которых в свою очередь является деревом. Деревья *T1, ...,Tm* называются ***поддеревьями*** данного корня. Структура дерева общего вида представлена на рис. 1.

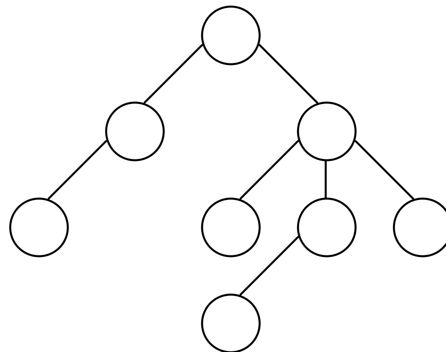


Рис. 1. Дерево общего вида

Пустым называется дерево, не содержащее узлов.

Число поддеревьев данного корня (узла) называется ***степенью*** этого ***узла***. Максимальная степень всех узлов дерева называется ***степенью дерева*** (обозначим *d*). Узел с нулевой степенью называется ***терминальным узлом или листом***. ***Уровень узла*** – выраженная в числе ребер длина пути, ведущего из узла в корень дерева, плюс единица. Считается, что корень дерева находится на уровне 1. Если некоторый узел A располагается на уровне *i*, то находящийся непосредственно ниже его на уровне *i+1* узел B называется ***непосредственным потомком*** узла A, а узел A называется ***непосредственным предком*** узла B. Максимальный уровень всех узлов дерева называется ***высотой или глубиной дерева*** (обозначим *h*). Высота пустого дерева равна нулю, высота дерева, состоящего из одного корня, равна 1. Дерево, содержащее максимальное число узлов, называется ***полным***. В полном дереве у всех узлов, за исключением терминальных, число

непосредственных потомков равно степени дерева. Количество узлов в полном дереве степени d высотой h вычисляется по формуле (i – номер уровня без единицы)

$$N_d(h) = \sum_{i=0}^{h-1} d^i .$$

Основные свойства деревьев общего вида:

- корень не имеет предков;
- каждый узел, за исключением корня, имеет только одного предка;
- каждый узел связан с корнем единственным путем, т.е. в деревьях отсутствуют замкнутые контуры (циклы).

Если в определении дерева имеет значение относительный порядок поддеревьев *T1,…,Tm*, дерево является ***упорядоченным***. Поэтому два упорядоченных дерева на рис. 2 – это разные, отличные друг от друга объекты.

Рис. 2. Два различных дерева

1.2. Бинарные деревья

Особенно важное практическое значение имеют упорядоченные деревья второй степени. Их называют ***двоичными или бинарными деревьями***. ***Бинарное дерево*** – это конечное множество узлов, которое или пусто, или состоит из корня и двух непересекающихся бинарных деревьев, называемых левым и правым поддеревьями данного корня. Примеры бинарных деревьев приведены на рис. 3.

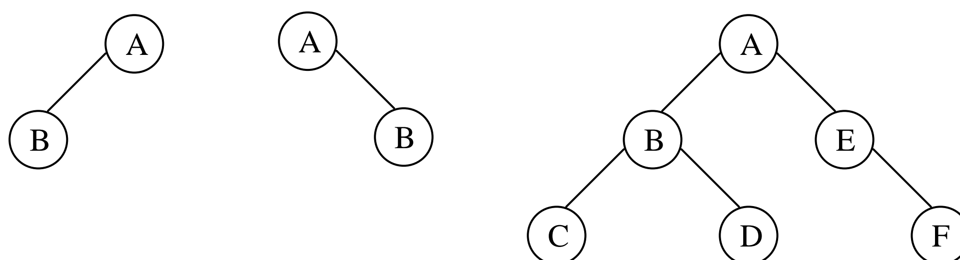


Рис. 3. Примеры бинарных деревьев

Максимальное число узлов в бинарном дереве высотой *h (d=2)*:

$$N_2(h) = \sum_{i=0}^{h-1} 2^i = 2^h - 1 .$$

Максимальное число узлов на уровне i в бинарном дереве:

$$n_i = 2^{i-1}.$$

Высота полного бинарного дерева, содержащего N узлов:

$$h = \lfloor log_2 N \rfloor + 1,$$

где $\lfloor \; \rfloor$ означает взятие целой части числа.

Существуют различные алгоритмы преобразования дерева произвольной степени, т.е. дерева общего вида, к виду бинарного. ***Левосторонний алгоритм*** формулируется следующим образом: у каждого узла дерева произвольной степени необходимо сохранить самую левую связь, а узлы – потомки одного и того же узла следует соединить правой связью. На рис. 14.а представлено исходное дерево произвольной степени, а на рис. 14.б – эквивалентное бинарное дерево.

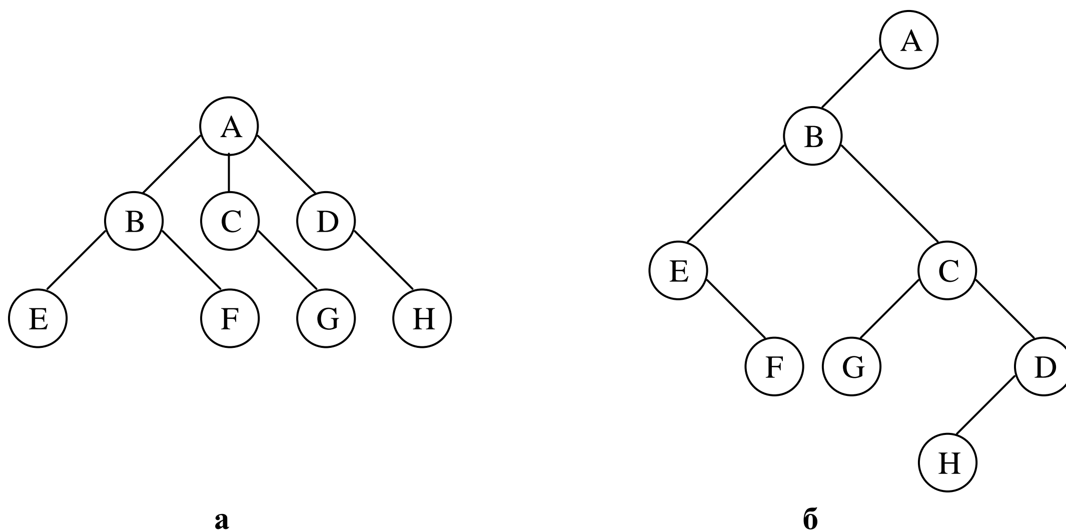


Рис. 4. Преобразование дерева произвольной степени к виду бинарного дерева: а – дерево произвольной степени; б – бинарное дерево

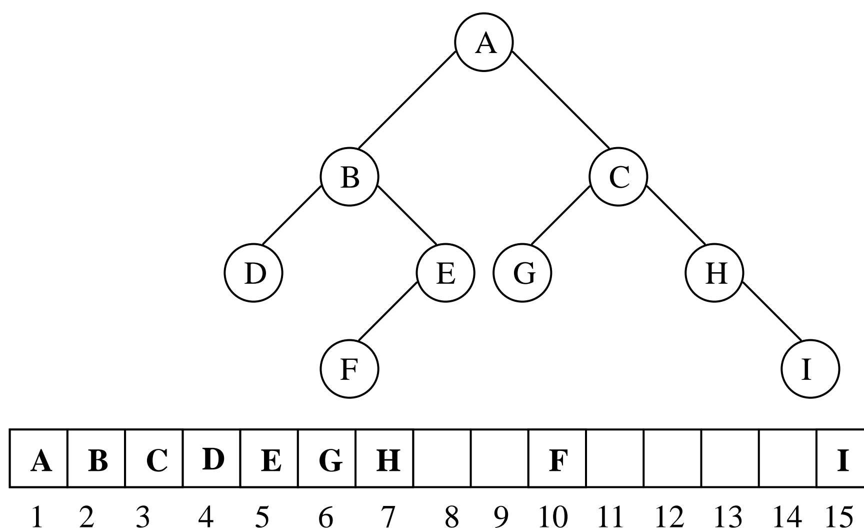
1.3. Представление бинарных деревьев

1.3.1. Представление бинарных деревьев в памяти с последовательной организацией

Если известен максимальный размер дерева (т.е. высота, а следовательно, и количество узлов, соответствующих полному дереву), то структуру дерева можно хранить в виде массива. При этом корень дерева будет храниться в элементе массива с индексом [1]. Для каждого узла с

номером *k* его левый потомок будет храниться в элементе с индексом [2 * k], а правый – в элементе с индексом [2 * k + 1] (см. рис. 5).

Достоинством представления бинарных деревьев в памяти с последовательной организацией является простота доступа как от предка к потомку, так и от потомка к предку, а недостатком – то, что если дерево не является полным, в массиве появляется большое число пустых элементов. Ограниченный размер массива затрудняет включение в дерево новых узлов, т.к. при этом требуется изменять размер массива. Удаление узла из дерева и соответствующее изменение его структуры также потребует модификации содержимого массива.



A	B	C	D	E	G	H			F					I
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Рис. 5. Представление бинарного дерева в памяти с последовательной организацией

1.3.2. Связанное представление бинарных деревьев

Элемент хранения узла бинарного дерева состоит из одного или нескольких информационных полей и двух полей связи, указывающих соответственно на левое и правое поддеревья данного узла:

```
public class TreeNode                        // Класс «Узел бинарного дерева»
{
   private char info;                        // информационное поле
   private TreeNode left;                    // ссылка на левое поддерево
   private TreeNode right;                   // ссылка на правое поддерево

   public char Info {…}                      // свойства
   public TreeNode Left{…}
   public TreeNode Right{…}

  public TreeNode () { }                     // конструкторы
  public TreeNode (char info)
  {
    Info = info;
```

```
  }
  public TreeNode (char info, TreeNode left, TreeNode right )
  {
    Info = info;  Left = left; Right = right;
  }
}

public class BinaryTree                  // Класс «Бинарное дерево произвольного вида»
{
  private TreeNode root;                           // ссылка на корень дерева
  public TreeNode Root               // свойство, открывающее доступ к корню дерева
  {
    get { return root; }
    set { root = value; }
  }
  public BinaryTree ()                              // создание пустого дерева
   {
     root = null;
    }
   …
}
```

Дерево задается ссылкой на его корень. Если дерево пусто, ссылка на его корень равна *null*. Связанное представление бинарного дерева иллюстрирует рис. 6.

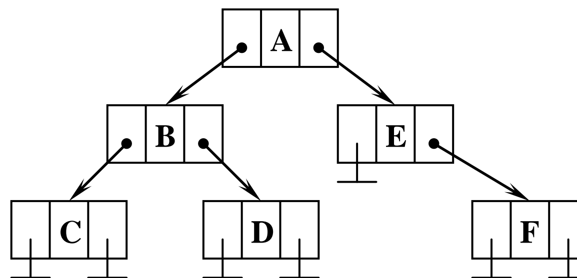


Рис. 6. Связанное представление бинарного дерева

1.4. Алгоритмы обхода бинарных деревьев

Наиболее типичная операция, выполняемая над бинарными деревьями, – обход дерева. ***Обход*** – это операция, при выполнении которой каждый узел обрабатывается ровно один раз одинаковым образом. Обход дерева заключается в разбиении дерева на корень, левое и правое поддеревья и применении к каждому из поддеревьев соответствующей операции обработки до тех пор, пока в процессе разбиения не будет получено пустое дерево. Полный обход дерева дает линейную расстановку узлов, что

облегчает выполнение многих алгоритмов. Существует несколько принципов упорядочения, которые естественно вытекают из структуры дерева. Как и саму древовидную структуру, их удобно представить с помощью рекурсии. Различным принципам упорядочения соответствуют три левосторонних алгоритма обхода в глубину (и три симметричных правосторонних алгоритма): нисходящий, восходящий, смешанный, а также обход в ширину.

1.4.1. Алгоритмы обхода в глубину

Рассмотрим три ***левосторонних*** алгоритма обхода.

Нисходящий (прямой) обход выполняется согласно алгоритму ***"корень-левый-правый" (К-Л-П)***:

1. обработать корневой узел;
2. обойти левое поддерево в нисходящем порядке;
3. обойти правое поддерево в нисходящем порядке.

Трасса нисходящего обхода дерева рис. 6: *A B **C D** E **F***.

Восходящий (концевой) обход выполняется согласно алгоритму ***"левый-правый-корень" (Л-П-К)***:

1. обойти левое поддерево в восходящем порядке;
2. обойти правое поддерево в восходящем порядке;
3. обработать корневой узел.

Трасса восходящего обхода дерева рис. 6: ***C D** B **F** E A* .

Смешанный (обратный) обход выполняется согласно алгоритму ***"левый-корень-правый" (Л-К-П)***.

1. обойти левое поддерево в смешанном порядке;
2. обработать корневой узел;
3. обойти правое поддерево в смешанном порядке.

Трасса смешанного обхода дерева рис. 6: ***C** B **D** A E **F*** .

Заметим, что при различных алгоритмах обхода порядок обработки терминальных узлов не изменяется.

Ниже приведен метод, распечатывающий трассу нисходящего обхода бинарного дерева – последовательность информационных полей узлов дерева.

```
public void KLP( TreeNode root )          // root – ссылка на корень дерева и любого из
                                                                          поддеревьев
      {
         if  ( root !=null )                                    // дерево не пусто?
        {                                // распечатать информ. поле корневого узла
           Console.WriteLine( root.Info );
            KLP( root.Left );             // обойти левое поддерево в нисходящем порядке
(* 1 *)     KLP( root.Right )             // обойти правое поддерево в нисходящем порядке
(* 2 *)   }
(* 3 *)  }
```

Для иллюстрации работы этого метода и построения трассы состояний стека будем использовать следующие обозначения:

^A – адрес узла дерева, в информационном поле которого хранится символ “A”;

(* 1 *) и (* 2 *) – адреса возврата из уровня рекурсивного метода, номер которого на 1 больше текущего уровня;

(* 3 *) – адрес возврата из текущего уровня рекурсивного метода.

Трасса нисходящего обхода дерева, приведенного на рис. 6, содержится в таблице 1.

Таблица 1. Трасса состояний стека при работе метода нисходящего обхода бинарного дерева

Номер входа в метод	Номер уровня рекурсии	Значение фактического параметра	Стек		Выходная трасса	Выход из уровня
			Исходное состояние	Конечное состояние		
1	1	^A	(-,-)	(^A, *1*)	A	
2	2	^B	(^A, *1*)	(^B, *1*) (^A, *1*)	B	
3	3	^C	(^B, *1*) (^A, *1*)	(^C, *1*) (^B, *1*) (^A, *1*)	C	
4	4	null	(^C, *1*) (^B, *1*) (^A, *1*)	(null,*1*) (^C, *1*) (^B, *1*) (^A,*1*)		
	4	null	(null,*1*) (^C, *1*) (^B, *1*) (^A,*1*)	(^C, *1*) (^B, *1*) (^A, *1*)		(*3*)
4	3	^C	(^C, *1*) (^B, *1*) (^A, *1*)	(^C, *2*) (^B, *1*) (^A, *1*)		
5	4	null	(^C, *2*) (^B, *1*) (^A, *1*)	(null,*2*) (^C, *2*) (^B, *1*) (^A, *1*)		
	4	null	(null,*2*) (^C, *2*) (^B, *1*) (^A, *1*)	(^C, *2*) (^B, *1*) (^A, *1*)		(*3*)
	3	^C	(^C, *2*) (^B, *1*) (^A,*1*)	(^B, *1*) (^A, *1*)		(*3*)
	2	^B	(^B, *1*) (^A,*1*)	(^B, *2*) (^A, *1*)		

Номер входа в метод	Номер уровня рекур-сии	Значение фактичес-кого параметра	Исходное состояние стека	Конечное состояние стека	Выход-ная трасса	Выход из уровня
6	3	^D	(^B,*2*) (^A,*1*)	(^D, *1*) (^B, *2*) (^A, *1*)	D	
7	4	null	(^D, *1*) (^B, *2*) (^A, *1*)	(null,*1*) (^D, *1*) (^B, *2*) (^A, *1*)		
	4	null	(null,*1*) (^D, *1*) (^B, *2*) (^A, *1*)	(^D, *1*) (^B, *2*) (^A, *1*)		(*3*)
	3	^D	(^D, *1*) (^B, *2*) (^A, *1*)	(^D, *2*) (^B, *2*) (^A, *1*)		
8	4	null	(^D, *2*) (^B, *2*) (^A, *1*)	(null,*2*) (^D, *2*) (^B, *2*) (^A, *1*)		
	4	null	null,*2*) (^D, *2*) (^B, *2*) (^A, *1*)	(^D, *2*) (^B, *2*) (^A, *1*)		(*3*)
	3	^D	(^D, *2*) (^B, *2*) (^A, *1*)	(^B, *2*) (^A, *1*)		(*3*)
	2	^B	(^B, *2*) (^A, *1*)	(^A, *1*)		(*3*)
	1	^A	(^A, *1*)	(^A, *2*)		
9	2	^E	(^A, *2*)	(^E, *1*) (^A, *2*)	E	
10	3	null	(^E, *1*) (^A, *2*)	(null,*1*) (^E, *1*) (^A, *2*)		

Номер входа в метод	**Номер уровня рекур-сии**	**Значение фактичес-кого параметра**	**Исходное состояние стека**	**Конечное состояние стека**	**Выход-ная трасса**	**Выход из уровня**
	3	null	(null,*1*) (^E, *1*) (^A, *2*)	 (^E, *1*) (^A, *2*)		(*3*)
	2	^E	(^E, *1*) (^A, *2*)	(^E, *2*) (^A, *2*)		
11	3	^F	 (^E, *2*) (^A, *2*)	(^F, *1*) (^E, *2*) (^A, *2*)	F	
12	4	null	 (^F, *1*) (^E, *2*) (^A, *2*)	(null, *1*) (^F, *1*) (^E, *2*) (^A, *2*)		
	4	null	(null, *1*) (^F, *1*) (^E, *2*) (^A, *2*)	 (^F, *1*) (^E, *2*) (^A, *2*)		(*3*)
12	3	^F	 (^E, *2*) (^A, *2*)	(^F, *2*) (^E, *2*) (^A, *1*)		
13	4	null	 (^F, *2*) (^E, *2*) (^A, *2*)	(null, *2*) (^F, *2*) (^E, *2*) (^A, *2*)		
	4	null	(null, *2*) (^F, *2*) (^E, *2*) (^A, *2*)	 (^F, *2*) (^E, *2*) (^A, *2*)		(*3*)
	3	^F	(^F, *2*) (^E, *2*) (^A, *2*)	 (^E, *2*) (^A, *2*)		(*3*)
	2	^E	(^E, *2*) (^A, *2*)	 (^A, *2*)		(*3*)
	1	^A	(^A, *2*)	(-, -)		(*3*)

1.4.2. Алгоритм обхода в ширину

При обходе в ширину узлы посещаются уровень за уровнем, начиная с корня, причем каждый уровень обходится слева направо.

Для реализации данного алгоритма используется структура очереди, в которой хранятся ссылки на поддеревья каждого узла. Для структуры очереди определены операции: *In_Queue* – включить в очередь; *Out_Queue* – исключить из очереди. Ниже приведена схема алгоритма обхода дерева в ширину: Q – ссылка на голову очереди, Root – ссылка на корень дерева.

```
…
 In_Queue( Q, Root );          // Занести в очередь ссылку на корень дерева
 while ( Q != null )           // Выполнять до тех пор, пока очередь не пуста
```

```
{
    Out_Queue( Q, p );                          // p – ссылка на узел, извлеченный из очереди
    Work( p );                                                        // Обработать узел p
    if  ( p.Left != null ) In_Queue( Q,p.Left );            // Если узел p имеет левого
                                      потомка, занести в очередь ссылку на левого потомка
    if  ( p.Right != null ) In_Queue( Q,p.Right );          // Если узел p имеет правого
                                    потомка, занести в очередь ссылку на правого потомка
}
```

Трасса обхода в ширину дерева рис. 6: *A B E* ***C D F***.

2. ВИДЫ БИНАРНЫХ ДЕРЕВЬЕВ

2.1. Деревья произвольного вида

Структура бинарного дерева ***произвольного вида*** не зависит ни от количества узлов в дереве, ни от каких-либо специальных признаков узлов, хранящихся в полях узлов. Бинарное дерево произвольного вида создается, начиная с корня, с помощью последовательного включения узлов либо в левое, либо в правое поддерево корневого узла. Пример структуры бинарного дерева произвольного вида показан на рис. 6.

2.2. Сбалансированные деревья

Дерево называется ***идеально сбалансированным***, если для каждой вершины число узлов в ее правом и левом поддеревьях отличается не более чем на единицу (далее будем называть такие деревья сбалансированными). ***Сбалансированное дерево***, состоящее из *N* узлов, имеет минимальную высоту среди всех бинарных деревьев. Высоту сбалансированного дерева можно определить по формуле:

$$h = \lfloor log_2 N \rfloor + 1.$$

Минимальная высота при заданном числе узлов достигается, если на всех уровнях, кроме терминального, размещается максимально возможное число узлов. Это происходит за счет равномерного размещения узлов поровну слева и справа от каждого узла.

Правило равномерного размещения для *N* узлов (*правило 1*) формулируется с помощью рекурсии:

- создать один узел в качестве корня;
- по *правилу 1* построить левое поддерево с числом узлов $nl = N/2;$
- по *правилу 1* построить правое поддерево с числом узлов $nr = N - nl - 1$.

Структура сбалансированного дерева из *N* узлов определяется количеством узлов (рис. 7).

N = 5

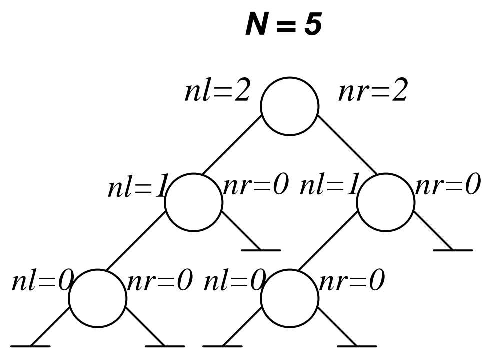


Рис. 7. Сбалансированное дерево

Метод построения сбалансированного дерева из *N* узлов:

```
public TreeNode Create_Balanced( int n)              // n – количество узлов в дереве
{                                                    // root – ссылка на корень дерева и
  char x; TreeNode root;                               на корень любого из поддеревьев
  if ( n == 0 ) root = null;                         // если n == 0, построить пустое дерево
  else
  {                                                  // заполнить информационное поле корня
        Console.WriteLine("Введите значение инф. поля узла ");
        x = Char.Parse( Console.ReadLine() );
        root = new TreeNode( x );                    // создать корень дерева
        root.Left = CreateBalanced( n/2 );           // построить левое поддерево
(*1*)  root.Right = CreateBalanced( n – n/2 – 1);    // построить правое поддерево
       }
(*2*) return root;
(*3*) }
...
BinaryTree T = new BinaryTree();     // объявление объекта класса "Бинарное дерево"
T.Root = T.Create_Balanced( 7 );     // создание сбалансированного дерева из 7 узлов
T.KLP(T.Root);                       // трасса нисходящего обхода дерева
```

Для иллюстрации работы этого метода и построения трассы состояний стека будем использовать следующие обозначения:

^А – адрес узла дерева, в информационном поле которого хранится символ “А”;

(* 1 *) и (* 2 *) – адреса возврата из уровня рекурсивного метода, номер которого на 1 больше текущего уровня;

(* 3 *) – адрес возврата из текущего уровня рекурсивного метода.

Трасса построения сбалансированного дерева, содержащего 3 узла, приведена в таблице 2.

Сбалансированное дерево, как и дерево любого другого вида, можно построить в процессе нисходящего обхода. Особенность работы метода построения дерева заключается в том, что сначала создаются корневые узлы, адреса которых запоминаются в стеке. Установить ссылки из корневых узлов дерева на их поддеревья можно только после того, как эти поддеревья будут созданы и адреса их корневых узлов возвращены в точки вызова. Точками

вызова являются поля ссылок на левое поддерево (*root.Left*) или правое поддерево (*root.Right*) корневого узла (в таблице установка этих ссылочных связей показана стрелками).

Таблица 2. Трасса состояний стека при работе метода построения сбалансированного дерева

Номер уровня рекур-сии	**Знач-я входного парам-ра n**	**Стек**		**Знач-я вых. пар-ра**			**Выход из уровня**
		Исходное состояние (n,root, (.) возврата)	**Конечное состояние (n,root, (.) возврата)**	**root**	**root. left**	**root. right**	
1	3	(-, -, -)	(3,^A, *1*)	^A	-	-	-
2	1	 (3,^A, *1*)	(1, ^B, *1*) (3, ^A, *1*)	^B	-	-	-
3	0	 (1, ^B, *1*) (3, ^A, *1*)	(0,null ,*1*) (1, ^B, *1*) (3, ^A, *1*)	null	-	-	-
	0	(0,null ,*1*) (1, ^B, *1*) (3, ^A, *1*)	 (1, ^B, *1*) (3, ^A, *1*)	null	-	-	(* 3 *)
2	1	(1, ^B, *1*) (3, ^A, *1*)	(1, ^B, *2*) (3, ^A, *1*)	^B	null	-	-
3	0	 (1, ^B, *2*) (3, ^A, *1*)	(0,null,*2*) (1, ^B, *2*) (3, ^A, *1*)	null	-	-	-
	0	(0,null,*2*) (1, ^B, *2*) (3, ^A, *1*)	 (1, ^B, *2*) (3, ^A, *1*)	null	-	-	(* 3 *)
2	1	(1, ^B, *2*) (3, ^A, *1*)	 (3, ^A, *1*)	^B	null	null	(* 3 *)
1	3	(3, ^A, *1*)	(3, ^A, *2*)	^A	^B	-	-
2	1	 (3,^A, *2*)	(1, ^C, *1*) (3, ^A, *2*)	^C	-	-	-
3	0	 (1, ^C, *1*) (3, ^A, *2*)	(0,null,*1*) (1, ^C, *1*) (3, ^A, *2*)	null	-	-	-
	0	(0,null,*1*) (1, ^C, *1*) (3, ^A, *2*)	 (1, ^C, *1*) (3, ^A, *2*)	null	-	-	(* 3 *)
2	1	(1, ^C, *1*) (3, ^A, *2*)	(1, ^C, *2*) (3, ^A, *2*)	^C	null		
3	0	 (1, ^C, *2*) (3, ^A, *2*)	(0,null,*2*) (1, ^C, *2*) (3, ^A, *2*)	null	-	-	-
	0	(0,null,*2*) (1, ^C, *2*) (3, ^A, *2*)	 (1, ^C, *2*) (3, ^A, *2*)	null	-	-	(* 3 *)
2	1	(1, ^C, *2*) (3, ^A, *2*)	 (3, ^A, *2*)	^C	null	null	(* 3 *)
1	3	(3, ^A, *2*)	(-, -, -)	^A	^B	^C	(* 3 *)

2.3. Дихотомические деревья (деревья поиска)

Бинарные деревья часто используются для представления множества объектов, среди которых идет поиск по уникальному значению некоторого атрибута, называемого ***ключом***. При этом на множестве объектов определено особое отношение порядка – ***дихотомия***, заключающееся в поэтапном разбиении множества информационных полей объектов на два непересекающихся подмножества. ***Дихотомическим деревом (деревом поиска)*** называется бинарное дерево, организованное так, что для каждого узла, имеющего ключ *K*, справедливо утверждение о том, что в его левом поддереве содержатся узлы с ключами, меньшими *K*, а в его правом поддереве содержатся узлы с ключами, большими *K*.

Описание элемента хранения узла дихотомического дерева:

```
public class DTreeNode                      // Класс «Узел дихотомического дерева»
{
   private char info;                                   // информационное поле
   private int key;                                              // поле ключа
   private DTreeNode left;                           // ссылка на левое поддерево
   private DTreeNode right;                         // ссылка на правое поддерево

   public char Info {…}                                            // свойства
   public int Key {…}
   public DTreeNode Left{…}
   public DTreeNode Right{…}

  public DTreeNode () { }                                       // конструкторы
  public DTreeNode (char info, int key)
  {
     Info = info;  Key = key;
  }
  public DTreeNode (char info, int key, DTreeNode left, DTreeNode right )
  {
    Info = info;  Key = key; Left = left; Right = right;
  }
}

public class DixotomyTree                    // класс «Дихотомическое дерево»
{
  private DTreeNode root;            // ссылка на корень дихотомического дерева

  public DTreeNode Root          // свойство, открывающее доступ к корню дерева
  {
    get { return root; }
    set { root = value; }
```

```
  }
  public DixotomyTree ()                    // создание пустого дерева
   {
      root = null;
    }
   …
}
```

Структура дихотомического дерева определяется последовательностью задания ключей. Последовательности задания ключей для

- ♦ дерева рис. 8 а) – 150, 70, 300, 100, 30, 200, 50;
- ♦ дерева рис. 8 б) – 100, 50, 30, 70, 200, 150, 300;
- ♦ дерева рис. 9 – 300, 200, 150, 100, 70, 50, 30.

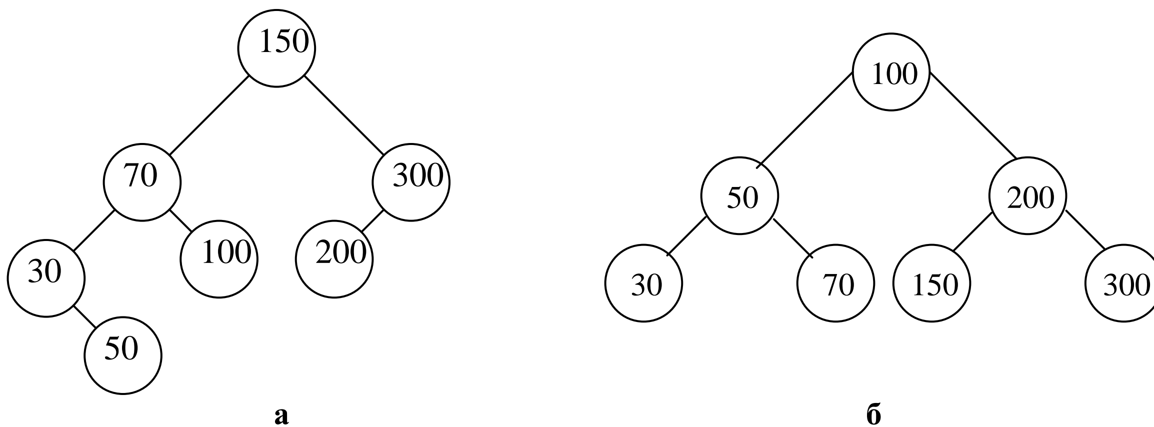


Рис. 8. Дихомические деревья:
а – несбалансированное; б – сбалансированное

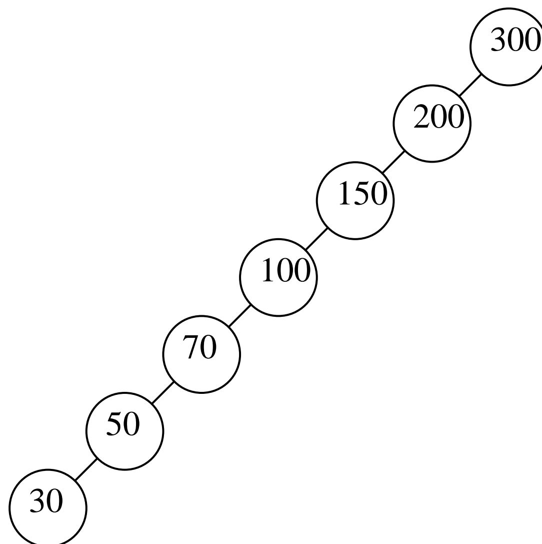


Рис. 9. Вырожденное дихотомическое дерево

Поиск в дихотомическом дереве узла с заданным значением ключа осуществляется на основе сравнения заданного ключа с ключом корня.

Единственное сравнение позволяет перейти к левому или к правому поддереву корня. Если дихотомическое дерево является сбалансированным, поиск узла с заданным значением ключа требует не более чем $\lfloor log_2 N \rfloor + 1$ сравнений, где N – количество узлов дерева. В частном случае, когда множество ключей является линейно упорядоченным, дихотомическое дерево фактически вырождается в линейный список (рис. 19). При этом поиск среди N узлов выполняется максимум за N сравнений, а в среднем – за $N / 2$ сравнений:

Включение в дихотомическое дерево узла с заданным значением ключа производится следующим образом: если поиск узла привел в тупик (т.е. к пустому поддереву, обозначенному ссылкой со значением *null*), то новый узел необходимо включить в дерево на место пустого поддерева. Таким образом, узел включается в дихотомическое дерево всегда в качестве листа.

```
public DTreeNode Ins_DNode(DTreeNode root, int k);
                                                  / root – ссылка на корень дерева
{                                                 // k – ключ вставляемого узла
 if  ( root == null )                  // дерево пусто – вставка узла в качестве листа
                                  // создать и инициализировать элемент хранения узла
   root = new DTreeNode(‘ ‘, k, null, null);
 else                                       // дерево не пусто – продолжить поиск:
 {                                                     // поиск в левом поддереве
   if  (k < root.Key) root.Left = Ins_DNode( root.Left,k )
   else                                               // поиск в правом поддереве
     if  (k > root.Key) root.Right = Ins_DNnode( root.Right,k )
     else                                          // ключ должен быть уникальным
       Console.WriteLine(“узел с ключом {0} уже есть в дереве”, k)
 }
 return root;
}
```

Если корневой узел не содержит искомого ключа, метод рекурсивно вызывает сам себя для продолжения поиска либо в левом, либо в правом поддереве. В том случае, если достигнут конец ветви и не обнаружен искомый ключ, создается новый узел с этим значением ключа. Рекурсия заканчивается, когда искомый ключ найден (при этом выдается сообщение о невозможности повторного включения узла) или достигнут конец ветви (при этом новый узел включается в дерево).

??? *Какой алгоритм обхода лежит в основе метода включения узла в дихотомическое дерево?*

При ***построении дихотомического дерева*** из N узлов на каждом из N шагов цикла осуществляется вставка узла с заданным значением ключа

(вызов метода *Ins_DNode*). Поэтому сложность создания дихотомического дерева из N узлов оценивается как $N * log_2N$ операций**.**

При ***удалении узла*** с заданным значением ключа из дихотомического дерева различают три случая:

1. узла с заданным значением ключа в дереве нет;
2. узел с заданным значением ключа имеет не более одного потомка. В этом случае исключаемый узел заменяется на своего потомка;
3. узел с заданным значением ключа имеет двух потомков. В этом случае исключаемый узел заменяется либо на *самый правый узел его левого поддерева*, имеющий не более одного потомка, либо на *самый левый узел его правого поддерева*, имеющий не более одного потомка.

Пример исключения узлов из дихотомического дерева приведен на рис. 10.

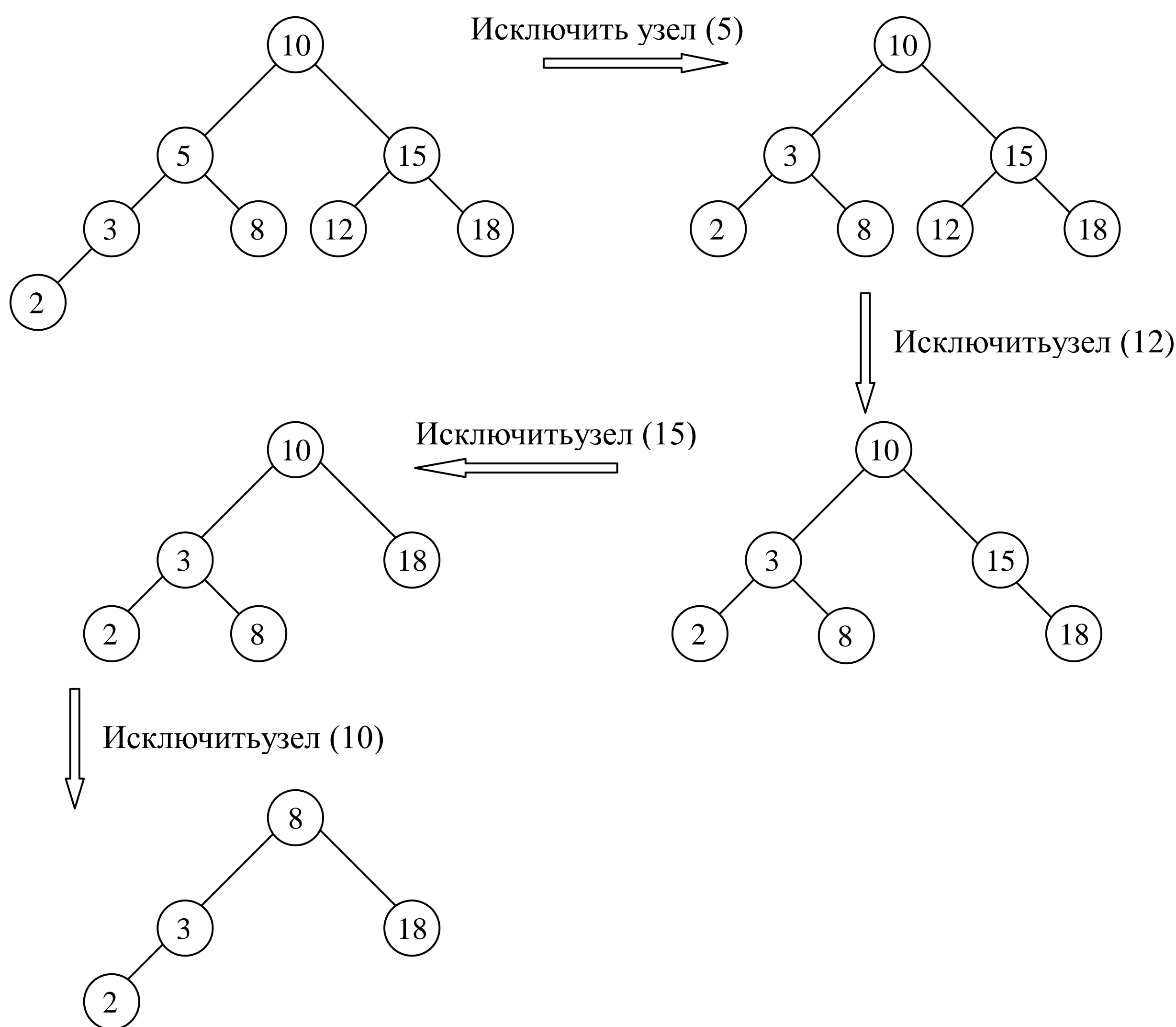


Рис. 10. Исключение узлов из дихотомического дерева

Разрушение бинарного дерева любого вида заключается в присвоении *null* ссылке на корень дерева. В результате разрушения дерево становится пустым.

2.4. Деревья выражений

Дерево выражений – бинарное дерево, в корневых узлах которого хранятся признаки операций, а в терминальных узлах – операнды выражения (переменные или константы). Дерево выражений представлено на рис. 11.

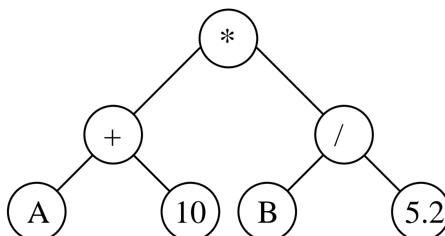


Рис. 11. Дерево выражений

Различные алгоритмы обхода дерева выражений соответствуют различной структуре представления выражения в виде строки.

Нисходящему обходу соответствует ***префиксная форма*** представления, т. к. в ней знак операции предшествует операнду:

** + A 10. / B 5.2 .*

Восходящему обходу соответствует ***постфиксная форма*** представления, т. к. в ней знак операции находится после операндов:

*A 10. + B 5.2 / * .*

Смешанному обходу соответствует ***инфиксная форма*** представления, т. к. в ней знак операции находится между операндами:

*A + 10. * B / 5.2 .*

Для того чтобы задать приоритеты операций, используется абсолютно скобочная форма, в которой каждое подвыражение заключается в круглые скобки:

*((A + 10.) * (B / 5.2)) .*

На рис. 12 приведено дерево, смешанный обход которого позволяет получить бесскобочную инфиксную форму, эквивалентную дереву рис. 21, а скобочная инфиксная форма задает совсем другие приоритеты операций.

Заметим, что подобная проблема не может возникнуть при использовании префиксной или постфиксной формы представления выражения.

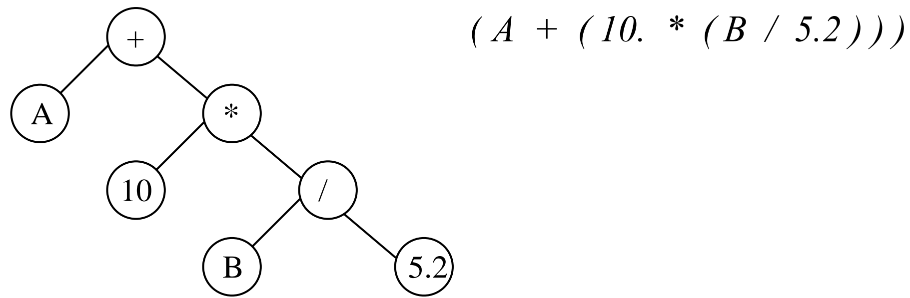


Рис. 12. Дерево выражений

??? *Какой алгоритм обхода необходимо использовать для того, чтобы подсчитать значение арифметического выражения, представленного в виде дерева?*

2.5. Демонстрационная программа, реализующая операции создания, обработки, просмотра содержимого бинарного дерева (на примере сбалансированного дерева)

```
public class TreeNode                          // описание узла бинарного дерева
{
   private char info;                                    // информационное поле
   private TreeNode left;                             // ссылка на левое поддерево
   private TreeNode right;                           // ссылка на правое поддерево

   public  char Info {…}                                          // свойства
   public TreeNode Left{…}
   public TreeNode Right{…}

  public TreeNode () { }                                        // конструкторы
  public TreeNode (char info)
  {
     Info = info;
  }

  public TreeNode (char info, TreeNode left, TreeNode right )
  {
    Info = info;  Left = left; Right = right;
  }
}

public class BalancedTree                  // Класс “Сбалансированное дерево”
{
  private TreeNode root;                              // ссылка на корень дерева
  public TreeNode Root             // свойство, открывающее доступ к корню дерева
  {
    get { return root; }
    set { root = value; }
  }

  public BalancedTree ()                             // создание пустого дерева
   {
     root = null;
    }
                                                 // создание дерева из n узлов
  public TreeNode Create_Balanced( int n)          // n – количество узлов в дереве
 {                                         // root – ссылка на корень дерева и
   char x; TreeNode root;                   на корень любого из поддеревьев
   if ( n == 0 ) root = null;            // если n == 0, построить пустое дерево
```

```
  else
  {                                             // заполнить информационное поле корня
     Console.WriteLine("Введите значение инф. поля узла ");
     x = Char.Parse( Console.ReadLine() );
     root = new TreeNode( x );                             // создать корень дерева
     root.Left = CreateBalanced( n/2 );                 // построить левое поддерево
     root.Right = CreateBalanced( n – n/2 – 1);        // построить правое поддерево
   }
  return root;
  }

 public void KLP(TreeNode root)                                // нисходящий обход
 {
   if (root != null)
   {
     Console.WriteLine(root.Info);                             // обработать корень
     KLP(root.Left);               // в нисходящем порядке обойти левое поддерево
     KLP(root.Right);             // в нисходящем порядке обойти правое поддерево
   }
 }

 public int Count( )                 // подсчет кол-ва узлов в дереве - восходящий обход
 {
   int s;
   if  ( root == null ) s = 0;
   else s = Work( root^.left ) + Work( root^.right + 1;
   return s;
 }

 public void Destroy( );                                         // разрушение дерева
 {
   root = null;
 }

public class Program
{
  static void Main()
  {
    BalancedTree T = new BalancedTree();                              // конструктор
    T.Root = T.Create();                          // создание сбалансированного дерева
    Console.ReadLine();
    T.KLP();                                               // нисходящий обход дерева
    Console.Write ( "Количество узлов в дереве = " );
    Console.WriteLine ( T.Count() );                               // обработка дерева
    Console.ReadLine();
```

```
    T.Destroy();                          // разрушение дерева
  }
}
```

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Лабораторная работа выполняется в четыре этапа.

1. *Ознакомительный этап*, на котором студент изучает методы представления и обработки динамических списковых структур.
2. *Подготовительный этап*, на котором составляются алгоритм и программа работы с динамическими списковыми структурами.
3. *Лабораторный этап*, на котором производится отладка программы.
4. *Составление отчета*. Отчет должен содержать:
   - название лабораторной работы и текст варианта задания;
   - листинг программы.

Для сдачи отчета необходимо продемонстрровать работу программы.

Требования к программе:

- корректность исходных данных, вводимых с клавиатуры, должна контролироваться программой;
- работа программы должна иллюстрироваться средствами графического и текстового вывода;
- должен быть разработан “дружественный” пользовательский интерфейс, обеспечивающий многократное выполнение программы с переопределением исходных данных, без выхода из оболочки программы.

4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение дерева общего вида.

2. Что такое степень дерева и глубина дерева?

3. Перечислите свойства деревьев общего вида.

4. Дайте определение бинарного дерева.

5. Сформулируйте алгоритм преобразования дерева произвольного вида к виду бинарного дерева.

6. Каким образом бинарное дерево представляется в памяти с последовательной и связанной организацией?

7. Какие алгоритмы обхода бинарных деревьев Вы знаете?

8. Дайте определение сбалансированного дерева. В чем отличительные особенности сбалансированных деревьев? Сформулируйте алгоритм построения сбалансированного дерева.

9. Дайте определение дихотомического дерева. Приведите примеры применения дихотомических деревьев.

10. Сформулируйте алгоритм создания дихотомического дерева.

11. Сформулируйте алгоритм исключения узла из дихотомического дерева.

12. Дайте определение дерева выражений.

13. Какой алгоритм обхода необходимо использовать для вычисления значения выражения, представленного в виде дерева?

14. Какой алгоритм обхода необходимо использовать для разрушения дерева?

5 ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

1. Создать сбалансированное дерево. Найти среднее арифметическое значений информационных полей узлов дерева.

2. Создать сбалансированное дерево. Подсчитать количество узлов дерева с положительными и отрицательными значениями информационных полей.

3. Создать сбалансированное дерево. Подсчитать количество узлов дерева с заданным значением информационных полей.

4. Создать дерево поиска. Подсчитать сумму значений информационных полей узлов дерева.

5. Создать дерево поиска. Подсчитать количество листьев в дереве. Распечатать значения информационных полей узлов дерева по уровням, т.е. все узлы уровня i, затем все узлы уровня i+1 и т.д.

6. Создать сбалансированное дерево. Подсчитать количество узлов дерева, не являющихся листьями.

7. Создать два дерева поиска. Скопировать в линейный список информационные поля элементов с совпадающими в первом и втором деревьях значениями ключей

8. Создать дерево поиска. Заменить все отрицательные значения информационных полей узлов дерева на их абсолютные величины.

9. Создать сбалансированное дерево. Найти максимальное значение среди всех информационных полей узлов дерева.

10. Создать дерево поиска. Построить копию этого дерева.

11. Создать дерево поиска. Скопировать в линейный список узлы дерева, выбранные по заданным значениям ключей.

12. Создать дерево поиска. Скопировать в новое дерево поиска узлы дерева, выбранные по заданным значениям ключей.

13. Создать сбалансированное дерево. Скопировать в упорядоченный линейный список узлы дерева, значения информационных полей которых лежат в диапазоне от N до K.

14. Создать сбалансированное дерево. Получить два линейных упорядоченных списка: в 1-й список скопировать узлы дерева, у которых значение информационного поля больше N; во 2-й список скопировать все остальные узлы дерева.

15. Создать два сбалансированных дерева. Определить эквивалентность двух деревьев.

16. Создать дерево поиска. Определить глубину дерева.

17. Создать дерево поиска. Написать процедуру исключения произвольного узла из дерева.

18. Задано произвольное дерево, не являющееся бинарным. Напишите процедуру преобразования этого дерева в бинарное.

19. Создать дерево поиска. Найти узел по заданному значению ключа. Если узел найден, построить два списка, упорядоченных по возрастанию значений ключей. В первый список скопировать содержимое узлов дерева с ключами, меньшими заданного ключа, во второй список скопировать содержимое узлов дерева с ключами, большими заданного ключа. Содержимое узла дерева – это поле ключа и информационное поле. При решении задачи использовать особенности структуры дерева поиска.

20. Заданы два дихотомических дерева. Исключить из первого дерева узлы, поля ключей которых совпадают с полями ключей узлов второго дерева.

ЗАКЛЮЧЕНИЕ

Методические указания “Нелинейные динамические структуры данных в C#. Деревья” посвящены рассмотрению структур данных и алгоритмов, которые являются фундаментом современной методологии разработки программ.

Методические указания подробно описывают нелинейные динамические структуры данных (деревья произвольной степени, бинарные деревья различных видов – сбалансированные, дихотомические, деревья выражений).

Теоретический материал иллюстрируется большим количеством примеров программ, реализующих алгоритмы обработки нелинейных динамических структур данных на языке C#.

Логическим завершением методических указаний являются контрольные вопросы и индивидуальные задания для самостоятельной работы студентов.

Вопросы, имеющие практическое значение для студентов при выполнении домашних заданий и лабораторных работ, освещены с необходимой для использования полнотой.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Н. Вирт. Алгоритмы и структуры данных: пер. с англ. / Н.Вирт. Изд. 2-е, испр. – СПб.: Невский диалект, 2005. – 352 с.
2. Ахо А. Структуры данных и алгоритмы: пер. с англ. / А.Ахо, Д. Хопкрофт, Д.Ульман. – М.: Вильямс, 2016. – 400 с.
3. Уильям Топп, Уильяи Форд. Структуры данных в C++: Пер. с англ. – М. ЗАО «Издательство БИНОМ», 1999. – 816 с.
4. Павловская Т.А. C#. Программирование на языке высокого уровня: учебник для вузов / Т.А.Павловская. – Спб.: Питер, 2014. – 432 с.
5. Фаронов В.В. Создание приложений с помощью C#. Руководство программиста / В.В. Фаронов. – М.: Эксмо, 2008. – 576 с.
6. Биллиг В.А.Основы программирования на C#: учебное пособие / В.А.Биллиг – М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2009. – 483 с.
7. Шилдт, Герберт. C# 3.0: руководство для начинающих: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2009. – 688 с.
8. Шилдт, Герберт. C# 4.0: Полное руководство: Пер. с англ. – М.: Издательский дом «Вильямс», 2011. – 1056 с.
9. Гросс, Кристиан. C# 2008 и платформа NET 3.5 Framework: базовое руководство: Пер. с англ. – М.: Издательский дом «Вильямс», 2009. – 480 с.
10. Петцольд, Чарльз. Программирование для Microsoft Windows 8. Пер. с англ. – СпБ.: Издательский дом «Питер», 2014. – 1008 с.

11. Стиллмен, Эндрю, Грин, Дженнифер. Изучаем C#. Пер. с англ. – СпБ.: Издательский дом «Питер», 2017. – 816 с.
12. Кнут Д. Искусство программирования для ЭВМ: в 3 т. Т 1: пер. с англ. / Д.Кнут. – М.: Вильямс, 2000. – 720 с.

Методические материалы

НЕЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ В C#. ДЕРЕВЬЯ

Методические указания

Составитель ***Симонова Елена Витальевна***

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ имени академика С. П. КОРОЛЕВА»
(Самарский университет)
443086, САМАРА, МОСКОВСКОЕ ШОССЕ, 34.

Изд-во Самарского университета.
443086 Самара, Московское шоссе, 34.