

# 国外数学名著系列

(影印版) 22

Victor N.Kasyanov

Vladimir A.Evstigneev

## Graph Theory for Programmers

Algorithms for Processing Trees

## 图论编程

分类树算法

图字:01-2005-6746

Victor N. Kasyanov, Vladimir A. Evstigneev; Graph Theory for Program-  
mers; Algorithms for Processing Trees

© 2000 Kluwer Academic Publishers

**This reprint has been authorized by Springer-Verlag (Berlin/Heidelberg/New  
York) for sale in the People's Republic of China only and not for export there-  
from.**

本书英文影印版由德国施普林格出版公司授权出版。未经出版者书面许  
可,不得以任何方式复制或抄袭本书的任何部分。本书仅限在中华人民共  
和国销售,不得出口。版权所有,翻印必究。

#### 图书在版编目(CIP)数据

图论编程:分类树算法=Graph Theory for Programmers; Algorithms for  
Processing Trees/(俄罗斯)卡西亚诺夫(Kasyanov, V. N.)著. —影印版 —  
北京:科学出版社,2006

(国外数学名著系列)

ISBN 7-03-016678-7

I. 图… II. 卡… III. ①图论-英文②图论算法-英文 IV. 0157.5

中国版本图书馆 CIP 数据核字(2005)第 154398 号

**科学出版社** 出版

北京东黄城根北街16号

邮政编码:100717

<http://www.sciencep.com>

**中国科学院印刷厂** 印刷

科学出版社发行 各地新华书店经销

\*

2006年1月第 一 版 开本:B5(720×1000)

2006年1月第一次印刷 印张:28

印数:1—2 500 字数:529 000

**定价:70.00 元**

(如有印装质量问题,我社负责调换〈科印〉)

## 《国外数学名著系列》(影印版)专家委员会

(按姓氏笔画排序)

丁伟岳 王 元 石钟慈 冯克勤 严加安 李邦河  
李大潜 张伟平 张继平 杨 乐 姜伯驹 郭 雷

### 项目策划

向安全 林 鹏 王春香 吕 虹 范庆奎 王 璐

### 执行编辑

范庆奎

## 《国外数学名著系列》(影印版)序

要使我国的数学事业更好地发展起来,需要数学家淡泊名利并付出更艰苦地努力。另一方面,我们也要从客观上为数学家创造更有利的发展数学事业的外部环境,这主要是加强对数学事业的支持与投资力度,使数学家有较好的工作与生活条件,其中也包括改善与加强数学的出版工作。

从出版方面来讲,除了较好较快地出版我们自己的成果外,引进国外的先进出版物无疑也是十分重要与必不可少的。从数学来说,施普林格(Springer)出版社至今仍然是世界上最具权威的出版社。科学出版社影印一批他们出版的好的新书,使我国广大数学家能以较低的价格购买,特别是在边远地区工作的数学家能普遍见到这些书,无疑是对推动我国数学的科研与教学十分有益的事。

这次科学出版社购买了版权,一次影印了23本施普林格出版社出版的数学书,就是一件好事,也是值得继续做下去的事情。大体上分一下,这23本书中,包括基础数学书5本,应用数学书6本与计算数学书12本,其中有些书也具有交叉性质。这些书都是很新的,2000年以后出版的占绝大部分,共计16本,其余的也是1990年以后出版的。这些书可以使读者较快地了解数学某方面的前沿,例如基础数学中的数论、代数与拓扑三本,都是由该领域大数学家编著的“数学百科全书”的分册。对从事这方面研究的数学家了解该领域的前沿与全貌很有帮助。按照学科的特点,基础数学类的书以“经典”为主,应用和计算数学类的书以“前沿”为主。这些书的作者多数是国际知名的大数学家,例如《拓扑学》一书的作者诺维科夫是俄罗斯科学院的院士,曾获“菲尔兹奖”和“沃尔夫数学奖”。这些大数学家的著作无疑将会对我国的科研人员起到非常好的指导作用。

当然,23本书只能涵盖数学的一部分,所以,这项工作还应该继续做下去。更进一步,有些读者面较广的好书还应该翻译成中文出版,使之有更大的读者群。

总之,我对科学出版社影印施普林格出版社的部分数学著作这一举措表示热烈的支持,并盼望这一工作取得更大的成绩。

王 元

2005年12月3日

*Dedicated to the memory  
of Andrei Petrovich Ershov*

## PREFACE

Although the first book devoted to the theory of graphs appeared in 1935, the extensive application of the methods of this theory in scientific and engineering research was originated only in the 50s. Thus, the books published at the beginning of the 60s (C. Berge, R. Busacker, and T. Saaty) already contained materials on the applications of the theory of graphs to the theory of operations, discrete optimization, electrical engineering, etc. They were followed by books completely devoted to the applications of the theory of graphs to certain fields of knowledge, including programming. Among these books, one should especially mention "The Introduction to Theoretical Programming. Discussion of a Method" by A. P. Ershov (1977), "Applications of the Theory of Graphs to Programming" by V. A. Evstigneev (1985), "Optimizing Transformations of Programs" by V. N. Kasyanov (1988), and "Combinatorial Analysis for Programmers" by V. Lipskii (1988), where a great experience of application of the methods of the theory of graphs to the analysis, optimization, and transformation of programs, organization of the process of computations, estimation of the complexity of programs, etc., is generalized and systematized.

The first (pioneer) works devoted to the application of the theory of graphs in programming were A. P. Ershov's papers concerning the organization of calculations of arithmetic expressions (1958) and the optimal use of random-access memory (1962) and R. Karp's note (1960), where a graph-theory model of a program in the form of a control (or transition) graph was proposed and used for the first time. At present, this model is classical for the solution of the problems of translation. It is extremely useful for subsequent investigations and practical applications in automated systems, in particular, for the verification of programs for correctness and their optimization. This model preserves its high importance even after the long-term evolution of computer architecture. Thus, the control graph serves as a basis of many intermediate representations of programs and their transformations used in vectorizing and parallelizing compilers for computers with parallel architecture.

Parallel with the control graph, the following graph models are extensively used in the practice of programming: the graph of procedure calls, the graph of dependences on the data, addressed data graphs, syntax and parsing trees, imbedding trees, sorting trees, dominator and postdominator trees, etc. Many other models widely used in programming, such as computational models, semantic and associative networks and operator schemes, Petri networks, Welbicki  $R$ -graphs, etc., can also be regarded as graph-theory models.

In delivering lectures and writing books, we were most often forced to pay absolutely no attention to a great body of interesting results and useful algorithms appearing in numerous sources and occasionally encountered. It was absolutely that most of these results would finally be forgotten because it is impossible to run through the entire variety of sources where these materials could be published. Therefore, we decided to do what we can to correct this situation. We discussed this problem with Ershov and came to an idea to write an encyclopedia of algorithms on graphs focusing our main attention on the algorithms already used in programming and their generalizations or modifications. We thought that it is reasonable to group all graphs into certain classes and place the algorithms developed for each class into a separate book. The existence of trees, i.e., a class of graphs especially important for programming, also supported this decision.

This monograph is the first but, as we hope, not the last book written as part of our project. It was preceded by two books "Algorithms on Trees" (1984) and "Algorithms of Processing of Trees" (1990) small editions of which were published at the Computer Center of the Siberian Division of the Russian Academy of Sciences. The books were distributed immediately and this made out our decision to prepare a combined monograph on the basis of these books even stronger. We called it "Theory of Graphs: Algorithms of Processing of Trees" rather than "Algorithms on Trees" in order to prevent the appearance of our book in the section of biology of libraries as already happened.

This book is intended to be not only a reference book of algorithms but also an introduction to the part of the theory of graphs that studies trees and their applications. We make an attempt to give a concise but more or less complete description of basic notions, properties, and fields of applications of trees. We clearly understand that it is impossible to include in the monograph all known algorithms but, at the same time, we have a hope that our collection is sufficiently representative. The reader can learn about algorithms that are not discussed into the book for various reasons from the bibliographic comments at the end of each chapter. These comments also give the reader a possibility to get some information about the history of development and current state of various problems of the theory of graphs and its applications. As far as the classical theory of graphs is concerned, the reader is referred to the additional list of references at the end of the book.

The monograph consists of eight chapters split into three parts. In the first part, we present main notions, properties of trees, and some basic algorithms, such as search in depth, the algorithms of coding and generation of trees, etc. The second part deals with the applications of trees to the problems connected with the structuring of programs, unification, systems of rewriting of terms, syntax analysis, etc. The third part is devoted to the problems of data storage and retrieval.

The book was partially financially supported by the Russian Foundation for Fundamental Research (Grant No. 93-012-576) and the Ministry of Science, Education and Technical Politics (Grant No. 2-15-2-43).

# CONTENTS

<b>Preface</b>	<b>vii</b>
<b>PART 1. BASIC CONCEPTS AND ALGORITHMS</b>	<b>1</b>
<b>Chapter 1. TREES AND THEIR PROPERTIES</b>	<b>1</b>
1.1. Introduction and Basic Definitions	1
1.2. Representations of Trees	18
1.3. Numbering and Calculation of Trees	39
1.4. Bibliographical Notes	46
References	46
<b>Chapter 2. COMPUTATIONAL MODELS. COMPLEXITY AND         FUNDAMENTAL ALGORITHMS</b>	<b>49</b>
2.1. Introduction. Algorithm Representation Language	49
2.2. Depth-First and Breadth-First Traversals of Graphs and Trees	61
2.3. Generation of Trees	91
2.4. Bibliographical Notes	112
References	115
<b>Chapter 3. SPANNING TREES</b>	<b>121</b>
3.1. The Problem of Finding the Optimal Spanning Tree	121
3.2. Algorithms of Numbering of All Spanning Trees	140
3.3. Search of Spanning Trees with Given Properties	154
3.4. Bibliographical Notes	159
References	163

<b>PART 2. TRANSLATION AND TRANSFORMATION OF PROGRAMS</b>	<b>175</b>
<b>Chapter 4. STRUCTURAL TREES</b>	<b>175</b>
4.1. Introduction and Principal Definitions	175
4.2. Hierarchical Representations of Regularizable CF-Graphs	187
4.3. Hammock Representations of CF-Graphs	198
4.4. Exposure of the Dominance Relation	208
4.5. Bibliographical Notes	215
References	219
<b>Chapter 5. ISOMORPHISM, UNIFICATION, AND             TERM-REWRITING SYSTEMS</b>	<b>223</b>
5.1. Isomorphisms of Trees	223
5.2. Problem of Unification	240
5.3. Term-Rewriting Systems	267
5.4. Bibliographical Notes	283
References	285
<b>Chapter 6. SYNTAX TREES</b>	<b>293</b>
6.1. Language Syntax and the Problem of Syntax Analysis	293
6.2. Generative Grammars	295
6.3. Syntax Analysis	302
6.4. Translation and Constructors of Analyzers	323
6.5. Bibliographical Notes	333
References	333
<b>PART 3. SEARCH AND STORAGE OF INFORMATION</b>	<b>337</b>
<b>Chapter 7. INFORMATION TREES</b>	<b>337</b>
7.1. Balanced Trees	337
7.2. Multidimensional Trees ( <i>k-d</i> -Trees)	364
7.3. Bibliographical Notes	372
References	372



<b>Chapter 8. TREES FOR MULTILEVEL MEMORY</b>	<b>375</b>
8.1. <i>B</i> -Trees	375
8.2. Generalizations of <i>B</i> -Trees	385
8.3. Multidimensional <i>B</i> -Trees	393
8.4. Multiattribute Trees	406
8.5. Bibliographical Notes	418
References	419
<b>ADDITIONAL LIST OF LITERATURE</b>	<b>423</b>
<b>SUBJECT INDEX</b>	<b>427</b>

# 1. BASIC CONCEPTS AND ALGORITHMS

## Chapter 1

### TREES AND THEIR PROPERTIES

#### 1.1. Introduction and Basic Definitions

**1.1.1. Introductory Remarks.** The present chapter contains basic facts about trees and their properties from the theory of graphs. All notions that are not defined here can be found in the literature. The number of notions of this sort is made as low as possible. The properties of trees and the corresponding results are presented independently of their subsequent use in the algorithms.

**1.1.2. Definition of a Tree.** An undirected connected graph without cycles is called a *tree* (Fig. 1.1a). An undirected graph without cycles is called a *forest*. In other words, a forest is a disconnected graph any connected component of which is a tree (Fig. 1.1b).

A tree is called *finite* if the number of its vertices is finite. Otherwise, the tree is *infinite*. The tree with one vertex is called *trivial*, *degenerate*, or *empty*.

The number of vertices in a tree is sometimes called the *order* of this tree.





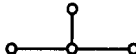
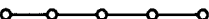
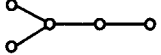


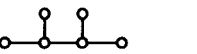
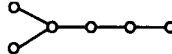
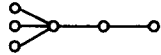
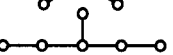


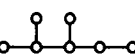
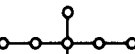
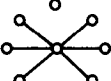
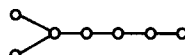
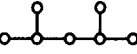
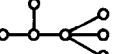
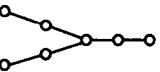
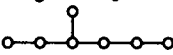
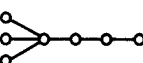
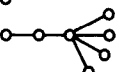

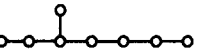
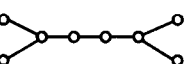
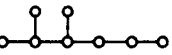
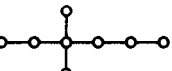
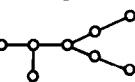
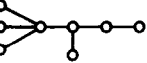


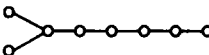
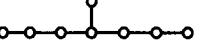
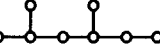
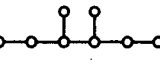
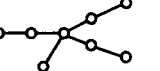
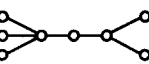
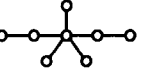



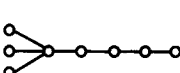
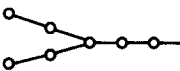
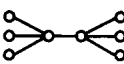

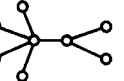
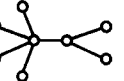
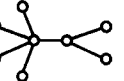
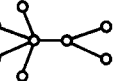
The tree without vertices of the second order is called a *homeomorphic irreducible tree*. The tree embedded in an Euclidean plane is called *planar* (see also the definition of planar graphs in Subsection 1.1.6).

Each of the following definitions completely characterizes a tree:

- (i) an undirected graph any two vertices of which are connected by a single chain is called a tree;
- (ii) an undirected graph without cycles is called a tree if the addition of one edge necessarily leads to the appearance of exactly one cycle;
- (iii) an undirected connected graph is called a tree if the removal of any edge leads to the loss of its connectedness;
- (iv) a connected undirected graph with  $n$  vertices and  $n - 1$  edges is called a tree;
- (v) an undirected graph without cycles (acyclic graph) with  $n$  vertices and  $n - 1$  edges is called a tree;
- (vi) an undirected graph all chains of which are simple (i.e., any vertex may appear in a given chain at most once) is called a tree.

## 1.1.3. Catalogue of Trees with at Most Eight Vertices

Table 1.1

Number of vertices	Trees		
1			
2			
3			
4			
5			
6	 	 	 
7	   	   	  
8	        	        	        

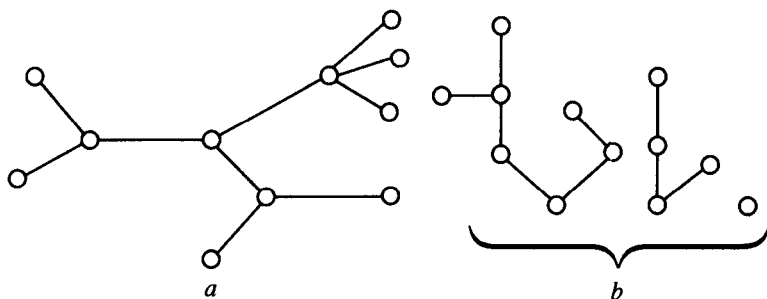
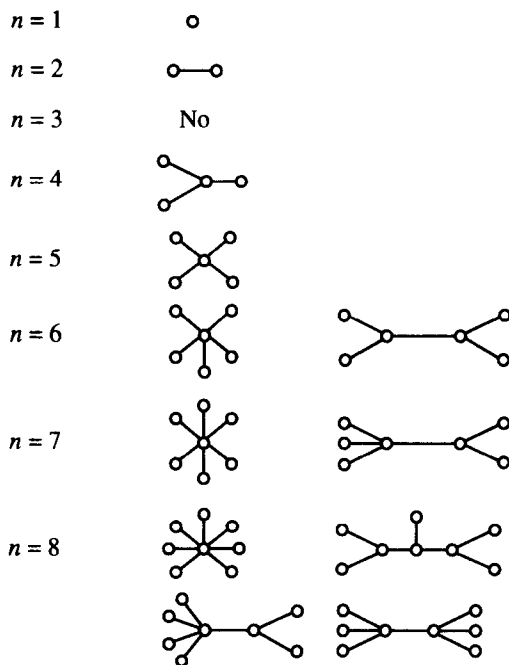


Fig. 1.1

All homeomorphically irreducible trees with at most eight vertices are listed below:



**1.1.4. Center and Centroid of a Tree.** The distance  $d(x, y)$  between the vertices  $x$  and  $y$  of the tree is defined as the number of edges in the chain connecting these vertices. The distance from a vertex  $x$  to the most remote vertex of the tree is called the *eccentricity*  $e(x)$  of the vertex  $x$ , i.e.,

$$e(x) = \max_y d(x, y).$$

The minimal eccentricity is called the *radius*  $r(T)$  of the tree  $T$ , i.e.,

$$r(T) = \min_x e(x) = \min_x \max_y d(x, y).$$

The maximal eccentricity is called the *diameter*  $D(T)$  of the tree, i.e.,

$$D(T) = \max_x \max_y d(x, y).$$

A vertex  $x$  is called a *central vertex* of the tree if  $e(x) = r(T)$ . The *center* of the tree is defined as the set of its central vertices. The tree whose center is formed by two vertices is called *bicentral*.

The center of a tree contains either a single vertex or two adjacent vertices.

The main property of the center of a tree can be formulated as follows:

*The center of a tree remains unchanged if we remove all its pendant vertices.*

The following properties of the trees are connected with the center:

- (a) the radius and diameter of a tree satisfy the relation

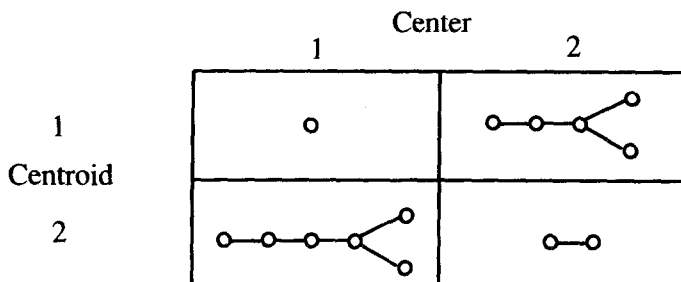
$$D(T) = \begin{cases} 2r(T) & \text{if the center has one vertex,} \\ 2r(T) - 1 & \text{if the center has two vertices;} \end{cases}$$

- (b) each chain of maximal length necessarily passes through the center of the tree.

A branch to a vertex  $v$  of a tree  $T$  is defined as a maximal subtree containing  $v$  as its pendant vertex. The number of branches to the vertex  $v$  is equal to the degree of this vertex. The weight of a vertex  $v$  is defined as the maximal number of edges along all branches to this vertex. A vertex  $v$  is called a *centroidal* if  $v$  its weight is minimal. The *centroid* of a tree is a set of all its centroidal vertices.

The centroid of a tree contains either a single vertex or two adjacent vertices.

The trees with one and two central or centroidal vertices and the smallest number of edges have the form



**1.1.5. Rooted and Directed Trees.** A tree is called *rooted* if it has a distinguished vertex  $r$  called the *root*. A tree without roots is sometimes called *free*. To solve some problems, e.g., to establish the isomorphism of trees, it is convenient to choose one of the central vertices as the root.

A *directed tree with root  $r$*  (or an *arboricity with root  $r$* ) is defined as a rooted tree each edge of which is replaced by an arc so that one can either reach the root moving from any vertex of the tree in the direction of arcs (tree to point) or reach an arbitrary vertex moving in the direction of arcs from the root (tree from point). In a tree to point the root is reachable from any vertex and, vice versa, in a tree from point any vertex is reachable from the root (Fig. 1.2).

Directed trees are also called *ditrees*.

It is easy to see that if we specify the root of a tree, then we get a directed tree. For a tree from point exactly one vertex (the root) has the indegree zero, the indegree of any other vertex is equal to one. The pendant vertices of a tree from point are called *leaves*.

Directed trees from point are sometimes called *growing*.

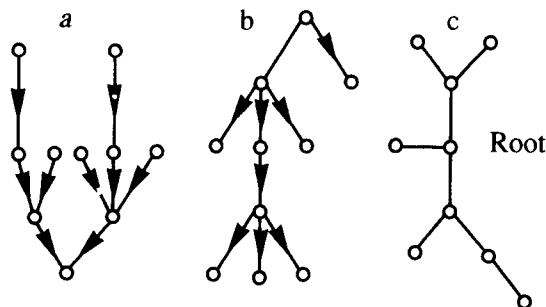


Fig. 1.2

Let  $T(X, U)$  be a directed tree with root  $r$ . The distance  $d(x, y)$  between vertices  $x$  and  $y$  of a directed tree is defined as the number of arcs in the path from  $x$  to  $y$ . If this path does not exist, then we set  $d(x, y) = \infty$ . We say that vertices of a directed tree are located on the  $k$ th level (or form the  $k$ th stratum) if their distances from the root  $r$  are equal to  $k$ . The maximal  $k$  for which the  $k$ th stratum is nonempty is called the *height* of a growing tree. Any vertex of a growing tree can be regarded as the root of the subtree growing from this vertex. The height of the subtree is defined analogously (Fig. 1.3).

**1.1.6. Ordered and Binary Trees.** A directed tree from point is called *ordered* if the set of arcs outgoing from any vertex of this tree is ordered. Two isomorphic directed trees can be nonisomorphic if they are regarded as ordered directed trees with different ordering of the outgoing arcs (Fig. 1.4).

Ordered trees are also called *planar* (see Subsection 1.1.2).

A directed tree is called *binary* if it can be defined by recursion as follows:

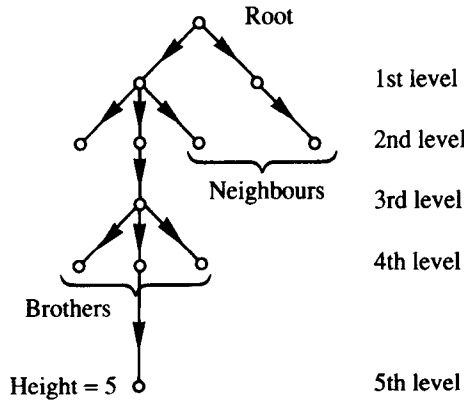


Fig. 1.3

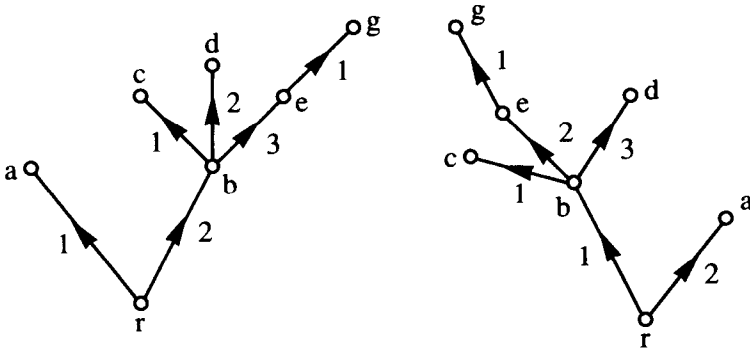


Fig. 1.4

- (a) the one-vertex tree is binary;
- (b) the triple  $(T_l, r, T_r)$  is a binary tree with the left subtree  $T_l$ , the root  $r$ , and the right subtree  $T_r$ ; both  $T_l$  and  $T_r$  are binary trees and may be empty (Fig. 1.5).

This definition implies that any arc going out of any vertex of a binary tree is either left or right. The only difference between binary and ordered trees is connected with the situation where a single arc goes out of a vertex. In this case, for binary trees, one must always classify it as right or left (Fig. 1.6).

Binary trees are also called *binary-search trees*. Some authors use the term “binary trees” for ordered trees with either two or no arcs going out of each vertex and the term “binary-search trees” is then used for trees that are binary in our sense.

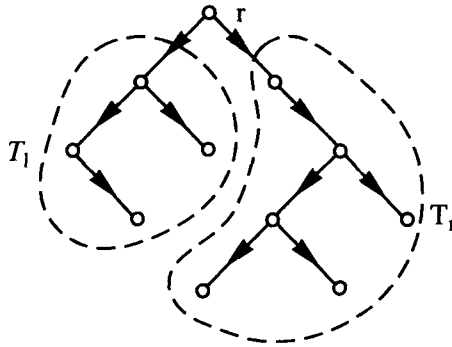


Fig. 1.5



Fig. 1.6

The notion *m-ary trees* generalizes the notion of binary trees. In these trees, each vertex is the root of at most  $m$  subtrees each of which, in its turn, is an *m-ary tree* that may be empty.

The *left-side binary tree* is defined by recursion as follows:

- (a) the one-vertex tree is a left-side binary tree;
- (b) a binary tree whose right subtree is empty and left subtree is a left-side binary tree is a left-side binary tree.

The right-side binary tree is defined analogously.

**1.1.7. Balanced Binary Trees.** A binary tree is called *balanced in height* or an *AVL-tree* (after Adelson-Velsky and Landis who described these trees for the first time) if, for any its vertex, the difference between the heights of the right and left subtrees does not exceed one.

The height of an  $n$ -vertex AVL-tree does not exceed  $0.5 \log n$  in the worst case and  $1.04 \log n$  on the average.

A binary  $n$ -vertex tree is called *balanced in weight* with balance  $\alpha$ ,  $0 < \alpha \leq 1/2$ , if it satisfies the conditions

$$(a) \quad \alpha \leq \frac{n_l + 1}{n + 1} \leq 1 - \alpha, \quad n_l = |T_l|;$$



- (b)  $T_l$  and  $T_r$  are trees balanced in weight with balance  $\alpha$ .

The class of binary trees balanced in weight is denoted by  $BB[\alpha]$ .

There exists a gap in the balances of trees, namely, for all  $\alpha$  from the interval  $1/3 < \alpha \leq 1/2$ , we have

$$BB[\alpha] = BB[1/2].$$

The classes of AVL-trees and trees balanced in weight are different.

The height of a binary  $n$ -vertex tree balanced in weight with balance  $\alpha$  does not exceed

$$\frac{\log(n+1) - 1}{\log(1-\alpha)^{-1}}.$$

Binary trees all leaves of which are located on the same level are called *justified*. The justified trees can be grouped into the following classes:

- (a) *H-trees* are trees any vertex of which with one successor has a right neighbor with two successors;
- (b) *HB-trees* are trees any vertex of which with one successor has a brother with two successors;
- (c) *HS-trees* are trees such that if a vertex has one successor, then this successor is either a leaf or has two successors;
- (d) *k-trees* are trees each vertex of which with one successor has at least one right neighbor and, moreover, the first  $k$  right neighbors (or all right neighbors, if their number is less than  $k$ ) have two successors.

**1.1.8. Spanning Trees and Directed Spanning Trees.** A subgraph of an undirected graph in the form of a tree is called a *spanning tree*. Spanning trees are also called *frames* or *contracting trees*.

An undirected graph has a spanning tree if it is connected.

A subgraph of a directed graph in the form of a tree directed to point is called its *directed spanning tree* (*spanning tree*).

A directed graph has a *directed spanning tree* with root  $r$  if all vertices of the graph are reachable from  $r$  or the vertex  $r$  is reachable from all other vertices. The first case corresponds to the spanning tree *outgoing* from the vertex  $r$  and the second case corresponds to the spanning tree *incoming* in the vertex  $r$ .

An *elementary transformation* of a spanning tree  $T$  into a spanning tree  $T'$  is introduced as follows: an edge  $u$  is added to the spanning tree  $T$  (this results in the formation of exactly one cycle) and another edge  $v$  is removed from the obtained cycle so that