8-1989

The Chromatic Sum and Efficient Tree Algorithms

Ewa Kubicka

Western Michigan University

Follow this and additional works at: https://scholarworks.wmich.edu/dissertations

Part of the Applied Mathematics Commons

Recommended Citation
THE CHROMATIC SUM AND EFFICIENT TREE ALGORITHMS

by

Ewa Kubicka

A Dissertation
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Doctor of Philosophy
Department of Mathematics and Statistics

Western Michigan University
Kalamazoo, Michigan
August 1989
The chromatic sum of a graph is the minimum total of the colors on the vertices taken over all possible proper colorings using natural numbers.

In Chapter I this new concept is introduced. It is shown that computing the chromatic sum for arbitrary graphs is an NP-complete problem. For every natural number $k$ the smallest tree which needs $k$ colors to attain its chromatic sum is constructed. Moreover it is demonstrated that asymptotically, for each value of $k$, almost all trees require more than $k$ colors to achieve their chromatic sums. In this chapter also a linear algorithm to compute the chromatic sum for a single tree is presented.

In Chapter II three constructions of graphs that require $t$ colors beyond their chromatic number $k$ to achieve their chromatic sum are presented, depending on the ratio $\frac{t}{k}$. The order of the resulting graphs grow linearly, quadratically, cubically and exponentially, depending on the $t$ chosen. The construction is proven to be the best possible for $t = 1$ and all $k$.

Chapter III deals with the chromatic sequence associated with a specific graph $G$, that is the sequence in $k$ of the minimum sums of colors taken over all proper colorings of the graph $G$ using exactly $k$ colors. It is shown that for trees this sequence is constrained, in fact it is inverted unimodal, while for arbitrary graphs it is unconstrained. This means that for any permutation of numbers 2 through $n$, a
graph $G$ can be found whose chromatic sequence after sorting into nondecreasing order realizes the given permutation.

In Chapter IV the weighted chromatic sum is investigated, which is a generalization of the chromatic sum with weights represented by positive real numbers put on the colors. Similar results to those in Chapter I and Chapter II are shown.

In Chapter V efficient algorithms for trees are presented. They are based on the Beyer and Hedetniemi constant time algorithm generating all rooted trees of a given order and on the Wright, Richmond, Odlyzko and McKay constant time algorithm generating all free trees of a given order. Besides the generic algorithm, three specific algorithms are given: to find among all trees of a given order the tree with maximum average order of a subtree; to examine the frequencies of cospectral trees; and finally to examine the frequencies of the trees which need two colors to attain their chromatic sums. It is shown that, for rooted trees, the average number of steps those algorithms have to perform per tree is bounded by a constant independent of the order of the trees. It is also conjectured that a similar property is true for free trees.

Appendices contain Pascal codes of the algorithms presented in Chapter I and Chapter V.
INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
The chromatic sum and efficient tree algorithms

Kubicka, Ewa Maria, Ph.D.
Western Michigan University, 1989
TO MY PARENTS
ACKNOWLEDGEMENTS

I would like to express my gratitude to the many faculty members of Western Michigan University who have contributed so much to my mathematical education and made my stay at WMU so enjoyable.

In particular, I would like to thank Professor Gary Chartrand, Professor Allen Schwenk and Professor Arthur White for their exceptional lectures. My deepest thanks and appreciation to Professor Stanislaw Leja for his support and generosity during a very challenging first year.

Special thanks are due to Professor Stephen Hedetniemi and Professor Jay Treiman for their patient and able assistance with my dissertation as well as to Professor Yousef Alavi and Professor Arthur White for serving on my committee. Professor Gary Chartrand’s professional and personal interest has been of real support and is greatly appreciated.

Above all I wish to thank Professor Allen Schwenk whose concern and assistance have been greater than any graduate student could expect. His mathematical expertise with a variety of different research problems has been of immeasurable benefit to me. I can only hope that this dissertation is in some way a repayment for the time and support that he has freely given.

Ewa Kubicka
TABLE OF CONTENTS

ACKNOWLEDGEMENTS ........................................................................ iii

CHAPTER

I. INTRODUCTION TO THE CHROMATIC SUM .................................. 1
   1.1 Introduction ............................................................................. 1
   1.2 The Chromatic Sum for Trees ............................................... 3
   1.3 Algorithmic Complexity of the Chromatic Sum ...................... 8
   1.4 An Algorithm for the Chromatic Sum of a Tree ...................... 11

II. GRAPHS THAT REQUIRE MANY COLORS TO ACHIEVE THEIR CHROMATIC SUM .............................................................. 18
   2.1 Existence .............................................................................. 18
   2.2 A More Efficient Construction for Relatively Few Extra Colors 20
   2.3 A Construction for an Arbitrary Number of Colors ............... 28

III. CONSTRAINTS ON THE CHROMATIC SEQUENCE FOR TREES AND GRAPHS ................................................................. 33
   3.1 Introduction ......................................................................... 33
   3.2 Chromatic Sequences for Trees .......................................... 34
   3.3 Chromatic Sequences for Graphs ....................................... 37

IV. THE WEIGHTED CHROMATIC SUM ................................................ 48
   4.1 Introduction .......................................................................... 48
   4.2 Extra Colors Needed to Obtain the Weighted Chromatic Sum.... 50

V. CONSTANT TIME ALGORITHMS FOR TREES ................................. 54
   5.1 Introduction .......................................................................... 54
   5.2 Generating Rooted and Free Trees ....................................... 55

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
5.3 Applications ......................................................................... 62
5.4 Trees with Maximum Average Order of Subtrees ................. 64
5.5 Frequencies of Cospectral Trees .............................................. 76
5.6 Frequencies of Trees which Need Two Colors to Attain the Chromatic Sum ............................................................ 83
5.7 Complexity Issues ................................................................... 84

VI. OPEN PROBLEMS .......................................................................... 97

APPENDIX 1. Generating all rooted trees of a given order .......... 100
APPENDIX 2. Generating all free trees of a given order .............. 103
APPENDIX 3. Finding the tree with the maximum average order of a subtree .......................................................... 108
APPENDIX 4. Finding the characteristic polynomials for trees of a given order .......................................................... 118
APPENDIX 5. Computing the chromatic sum and the maximum number of colors used ...................................................... 130

REFERENCES ..................................................................................................... 140
CHAPTER I

INTRODUCTION TO THE CHROMATIC SUM

1.1 Introduction

All graph-theoretical terms not defined in this dissertation have the same meaning as in Chartrand and Lesniak [3]. Algorithmic complexity terminology follows Garey and Johnson [5].

As usual, |S| denotes the cardinality of a set S. For a graph G, we use V(G) and E(G) to denote the vertex set and edge set of G, respectively. The number |V(G)| = |G| is called the order of a graph G and |E(G)| is called the size of G.

A proper coloring of a graph G is an assignment of colors (represented by positive integers) to the vertices of G such that any two adjacent vertices receive different colors. The smallest number of colors needed by a graph G to be properly colored is called the chromatic number of G and is denoted by χ(G). By a Grundy coloring of a graph G we understand a proper coloring of G in which every vertex of color k > 1 is adjacent to at least one vertex of color i for every i < k. The maximum number of colors that might be used in a Grundy coloring of a graph G is called the Grundy number and is denoted by Γ(G) (as in [15] and [4]).

In this chapter we wish to introduce a new variation on the chromatic number of a graph. Instead of minimizing the number of colors in a proper coloring, we choose instead to minimize the sum of these colors. More formally, we define the chromatic sum of graph G, Σ(G), to be the minimum sum \( \sum_{v \in V} c(v) \) taken over all proper colorings c of graph G. A proper coloring c of a graph G is called a...
**best coloring** of G whenever \( \sum_{v \in V} c(v) = \Sigma(G) \). By \( \Sigma^c(G) \) we will denote the sum of colors for the specified coloring \( c \).

**Example 1.1** Consider the tree \( T \) and two proper colorings \( c \) and \( c' \) shown in Figure 1. We see that \( \Sigma(T) = 11 \) and \( c' \) is the unique best coloring. This example illustrates the surprising feature that the minimum total may very well be achieved by using more than the minimum number of colors.

![Figure 1.1 Two proper colorings of the tree T.](image)

For every \( k, \chi(G) \leq k \leq \Gamma(G) \), we define the \( k\)-**chromatic sum** of G, \( \Sigma_k(G) \), to be the minimum sum \( \sum_{v \in V} c(v) \) taken over all proper Grundy colorings \( c \) of graph G using exactly \( k \) colors. With every graph G we associate a sequence of numbers:

\[
\Sigma_{\chi(G)}(G), \Sigma_{\chi(G)+1}(G), \ldots, \Sigma_k(G), \ldots, \Sigma_{\Gamma(G)}(G),
\]

and call it the **chromatic sequence** of G.
1.2 The Chromatic Sum for Trees

It is well known that trees are bipartite and thus we can always color them properly using only colors 1 and 2. However, Example 1.1 shows that sometimes we are forced to use additional colors to attain the chromatic sum. We will construct now, for each $k$, a tree of smallest order in which color $k$ is forced to appear in every best coloring. We first construct the family of rooted trees $T^m_k$ recursively. We assume that removing the root $r$ leaves a forest in which each tree is rooted at the vertex that is adjacent to $r$ in $T^m_k$. Let $T^1_k$ be the rooted tree with one vertex. Tree $T^m_k$ is the unique tree such that $T^m_k - r = \bigcup_{i=1}^{m+k-1} (m + k - i)T^1_i$. Examples of this construction are shown in Figure 1.2.

![Diagram of $T^2_3$ and $T^m_k$](image-url)

Figure 1.2 The tree $T^2_3$ and $T^m_k$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Lemma 1.2  For \( k \geq 2 \), the tree \( T_k^m \) is the smallest rooted tree for which, in every best coloring, color \( k \) is forced to appear at the root and any change of that color to a lower color must increase the sum of colors by at least \( m \).

Proof: (by induction on \( k \))

(1) It is easy to see that \( T_2^m \) has to have color 2 at the root in any best coloring and changing it to 1 costs exactly \( m \). Note that \( |T_2^m| = m + 2 \). Let \( T \) be a smallest rooted tree having that property and let \( c \) be its best coloring. Let \( b \) denote the number of vertices of color 2 in \( T \). Then \( T \) must have at least \( b + m \) vertices colored with 1 because otherwise interchanging colors 1 and 2 in \( T \) would give a proper coloring of \( T \) with the root colored with 1 and the increase of the sum of colors by less than \( m \). Thus \( |T| \geq m+2 \).

(2) Assume that the lemma is true for all \( i \leq k \). Let \( T(i,m) \) denote a tree of the smallest order where color \( i \) is forced to appear at the root and its change costs at least \( m \). Consider \( T(k+1,m) \) and its best coloring \( c \). We will show that \( T(k+1,m) \) must be \( T_{k+1}^m \). After removing the root from \( T(k+1,m) \) we are left with a forest of rooted trees. Let \( F_j(k+1,m) \) denote the subforest containing all those trees with roots colored with \( j \). \( F_j(k+1,m) \) is a smallest forest of the property that changing the color \( j \) at the roots to any other color costs at least \( k+1-j+m \). Therefore, for \( j < k \), \( F_j(k+1,m) = F_j(k,m+1) \) which is the similar forest for \( T_{k+1}^{m+1} \). Thus, using the inductive hypothesis, we have that \( F_j(k+1,m) = (k+1-j+m)T_j^1 \). Now the only thing left is to show that \( F_k(k+1,m) = (m+1)T_k^1 \). The forest \( (m+1)T_k^1 \) has all the roots colored with \( k \) and any change of that color costs at least \( m+1 \). Consider the subtree \( T_0 \) of \( T(k+1,m) \) consisting of the largest connected component of \( T(k+1,m) \) containing the root and only vertices colored with \( k+1 \) and \( k \). Among all possible
Let's select that one which has the fewest number of vertices colored with $k$ occurring in the corresponding subtree $T_0$. Let $b$ denote the number of vertices of $T_0$ colored with $k+1$. Then, similarly as in (1), we can show that $T_0$ must have at least $m + b$ vertices colored with $k$, let's call them $v_1, v_2, \ldots, v_r$ ($r \geq m + b$). Let $S_i$ denote the subtree of $T(k+1,m)$ with root $v_i$ which is formed after deleting all edges between $v_i$ and any vertex colored $k+1$. We claim that there must be at least $m+1$ of these subtrees $S_i$ for which $|S_i| \geq |T^1_k|$. Assume this is not true. Recall that $T^1_k$ is the smallest rooted tree in which color $k$ is forced at the root. Thus $|S_i| < |T^1_k|$ means that $S_i$ has a best coloring $c'$ which uses a different color at the root. If this color is smaller then $k$ we can change our best coloring of $T(k+1,m)$ by just using $c'$ on $S_i$ obtaining a best coloring of $T(k+1,m)$ with smaller number of vertices colored with $k$ in $T_0$—a contradiction. If, on the other hand, in all $S_i$'s for which $|S_i| < |T^1_k|$ we could change root color from $k$ to $k+1$ at no cost then swapping colors $k+1$ and $k$ throughout $T_0$ would produce a best coloring of $T(k+1,m)$ with root colored by $k$ and with the increase of sum of colors less then $m$, another contradiction. Therefore $|F_k(k+1,m)| \geq |(m+1)T^1_k|$. □

Denote now by $T_k$ the unrooted tree formed by adding an edge between the roots of two copies of $T_{k-1}^2$.

![Diagram](image)

Figure 1.3 The smallest tree requiring $k$ colors in a best coloring.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Theorem 1.3 \( T_k \) is the smallest tree in which color \( k \) is needed in every best coloring. For \( k \geq 2 \), the order of \( T_k \) is given by

\[
|T_k| = \frac{1}{\sqrt{2}} \left[ (2 + \sqrt{2})^{k-1} - (2 - \sqrt{2})^{k-1} \right].
\]

Proof: Let \( F_k \) be a smallest tree that requires \( k \) colors in every best coloring, and let \( c \) be its best coloring. Locate an edge \( e = (v_1, v_2) \) joining a vertex of color \( k \) to one of color \( k - 1 \). Removing edge \( e \) leaves two trees \( S_1 \) and \( S_2 \) such that every best coloring of \( T_i \) must color the roots \( v_i \) with \( k - 1 \) and changing that color costs at least 2. Thus \( S_1 = S_2 = T_{k-1}^2 \).

Let \( t_k^m = |T_k^m| \). The equation defining these trees immediately gives the recurrence

\[
t_k^m = 1 + \sum_{i=1}^{k-1} (k+m-i)t_i^1.
\]

We first show by induction that the subsequence with \( m = 1 \) satisfies the simpler recurrence \( t_k^1 = 4t_{k-1}^1 - 2t_{k-2}^1 \). The induction is based on the instance \( k = 3 \) verified by the values \( t_1^1 = 1, t_2^1 = 3, \) and \( t_3^1 = 10 \). Next assume that \( k \geq 4 \) and that the recurrence has already been proved for all \( 3 \leq i < k \). Note that

\[
t_k^1 = 4t_{k-1}^1 + 2t_{k-2}^1 = 1 + \sum_{i=1}^{k-1} (k + 1 - i)t_i^1 - 4 - 4 \sum_{i=1}^{k-2} (k - i)t_i^1 + 2 + 2 \sum_{i=1}^{k-3} (k - 1 - i)t_i^1.
\]

Adjusting indices in the second and third summations and collecting the isolated terms gives

\[
t_k^1 = 4t_{k-1}^1 + 2t_{k-2}^1 = -1 + \sum_{i=1}^{k-1} (k + 1 - i)t_i^1 - 4 \sum_{i=2}^{k-1} (k + 1 - i)t_i^1 + 2 \sum_{i=3}^{k-1} (k + 1 - i)t_{i-2}^1.
\]
The induction hypothesis guarantees that all the terms with \( i \geq 3 \) cancel leaving
\[
\frac{1}{t_k} - 4\frac{1}{t_{k-1}} + 2\frac{1}{t_{k-2}} = -1 + k\frac{1}{t_1} + (k-1)\frac{1}{t_2} - 4(k-1)\frac{1}{t_3} = 0.
\]
But the same proof works for \( t_k^m \) provided that \( m \geq k + 2 \). This time the remaining uncanceled terms are
\[
t_k^m - 4t_{k-1}^m + 2t_{k-2}^m = -1 + (k+m-1)\frac{1}{t_1} + (k+m-2)\frac{1}{t_2} - 4(k+m-2)\frac{1}{t_3} = 0.
\]
We specifically need the solution when \( m = 2 \). Given the starting values \( t_2 = 4 \) and \( t_3 = 14 \), it is routine to solve the recurrence for \( k \geq 2 \) to find that
\[
t_k^2 = \frac{1}{2\sqrt{2}} [(2 + \sqrt{2})^k - (2 - \sqrt{2})^k].
\]
Of course \( |T_k| = 2t_{k-1}^2 \). □

**Corollary 1.4** For every positive integer \( k \), almost every tree requires at least \( k \) colors in its best coloring; i.e.: \( \lim_{n \to \infty} \frac{t_n(k)}{t_n} \to 0 \) as \( n \to \infty \), where \( t_n(k) \) is the number of trees of order \( n \) which have a best coloring using less than \( k \) colors and \( t_n \) is the number of all trees of order \( n \).

**Proof.** This is an immediate consequence of a theorem due to A. Schwenk [14] stating that for every rooted tree \( L \), almost every tree has \( L \) as a limb. We simply set \( L \) to be \( T_k^1 \). □

**Theorem 1.5** Let \( T \) be a tree of order \( n \geq 1 \). Then \( n + 1 \leq \Sigma(T) \leq \lfloor 1.5n \rfloor \).
Moreover, for every \( k \) between \( n+1 \) and \( \lfloor 1.5n \rfloor \), there is a tree \( T \) of order \( n \) such that \( \Sigma(T) = k \).
**Proof.** We have to use at least 2 colors and thus \( \Sigma(T) \geq n+1 \). For a star \( K(1,n-1) \) this lower bound is attained. A tree is a bipartite graph and therefore we can always color it properly using colors 1 and 2, coloring all vertices in the possibly bigger partite set with color 1 and all vertices in the other partite set with color 2. Thus

\[
\Sigma(T) \leq \left\lfloor \frac{n}{2} \right\rfloor + 2\left\lfloor \frac{n}{2} \right\rfloor = \lfloor 1.5n \rfloor.
\]

The upper bound is attained by a path \( P_n \). Define now \( B(m,b) \) to be a "broom-like" tree consisting of a vertex adjacent to \( m \) end-vertices and a path of length \( b - 1 \). This family is illustrated in Figure 1.4.

![Figure 1.4 The broom B(3,5).](image)

Let \( n + 1 < k < \lfloor 1.5n \rfloor \). We can represent \( k \) as \( k = n + 1 + b \), for some \( b \). It is easily verified that \( \Sigma(B(n - 2b - 1,2b + 1)) = k \). \( \square \)

1.3 Algorithmic Complexity of the Chromatic Sum

The problems of determining the chromatic number of an arbitrary graph and finding a proper coloring which uses exactly the chromatic number of different colors are well known to be NP-hard. For trees, however, these problems are trivial. On the other hand the results from the previous section show that determining the chromatic sum for trees might be fairly complicated and suggest that the chromatic sum problem for arbitrary graphs is also NP-hard. The following theorem shows that this is indeed true.
Theorem 1.6 The following decision problem II is NP-complete:

Generic instance: A graph G = (V,E) and a positive integer K.

Question: Is there a proper coloring c of the graph G such that \( \sum_{v \in V} c(v) \leq K \)?

Proof: It is easy to see that II \( \in \) NP since a nondeterministic algorithm need only guess a color assignment to the vertices of G and check (in polynomial time) whether that color assignment is proper and whether its sum does not exceed K.

Consider another decision problem II' :

Generic instance: A graph G = (V,E) and a positive integer K \( \leq |V| \).

Question: Is G K-colorable?

Since II' is known to be NP-complete, we shall show that II is also by transforming II' into problem II. Let an arbitrary instance of II' be given by a graph G = (V,E) and a positive integer K \( \leq |V| \). We must construct a graph G' = (V',E') and a positive integer K' such that there exists a proper coloring c' of G' with the property: \( \sum_{v \in V'} c'(v) \leq K' \) if and only if G is K-colorable. Let the graph G' = G \times K be the cartesian product of graph G with a complete graph with K vertices. That is, if \( V = \{v_1, v_2, ..., v_n\} \) then \( V' = \{v_i^j \mid 1 \leq i \leq K, 1 \leq j \leq n\} \) and \((v_i^j, v_i^s) \in E'\) whenever either \((j = s \text{ and } i \neq r)\) or \((i = r \text{ and } (v_j^i, v_s^j) \in E).\) Also let \( K' = n \cdot \frac{K(K+1)}{2}. \) Assume that G is K-colorable and let \( c: V \to \{1, 2, ..., K\} \) be a proper coloring of G. We can envision graph G' = (V',E') in the following way: \( V' = \{v_1^1, ..., v_1^n, v_2^1, ..., v_2^n, ..., v_n^1, ..., v_n^K\} \) where vertices \( v_i^1, ..., v_i^n \) form the i-th copy of G for \( 1 \leq i \leq K \) and vertices \( v_j^1, ..., v_j^K \) form the j-th copy of \( K \) for \( 1 \leq j \leq n.\)
We can extend coloring $c$ to a proper coloring $c'$ of graph $G'$ in the following way: the first copy of $G$ has the same coloring scheme as $G$, in the second copy colors are shifted by 1 in a cyclic manner and so on, i.e.

$$c'(v^i_j) = (c(v^i_j) + i - 2) \mod K + 1.$$ 

Thus $G' = (V',E')$ is $K$-colorable and $\sum_{v \in V'} c'(v) = n \cdot \frac{K(K+1)}{2} = K'$. Conversely, suppose that there exists a proper coloring $c'$ of graph $G'$ such that $\sum_{v \in V} c'(v) \leq K'$. Since each copy of $K_k$ in $G'$ has to use $K$ different colors, the smallest possible sum of colors, $\frac{K(K+1)}{2}$, can be obtained only if we use colors 1 through $K$. In order for any proper coloring of $G'$ to have its sum of colors bounded above by $K'$, we must use colors 1 through $K$ at each copy of $K_k$ in $G'$. Therefore $G'$ is $K$-colorable and so is its subgraph $G$.

Notice that $|V'| = K \cdot n \leq n^2 = |V|^2$ and $|E'| = K \cdot |E| + n \cdot \frac{K(K-1)}{2} < n^3$. Thus the instance $(G',K')$ can be constructed from the instance $(G,K)$ in polynomial time.
time. We have shown that the NP-complete problem $\Pi'$ can be polynomially transformed into problem $\Pi$. Therefore $\Pi$ is also NP-complete. \qed

**Corollary 1.7** The optimization problem of finding the chromatic sum can be solved in polynomial time if and only if $P = NP$.

**Proof:** Any algorithm for finding the chromatic sum can be used as an algorithm for solving the decision problem $\Pi$. Therefore if the chromatic sum can be always found in polynomial time, $\Pi \in P$, and thus $P = NP$.

Conversely assume that $P = NP$ which implies the existence of a polynomial algorithm $A^*$ for solving $\Pi$. For graphs of order $n$, we know that $n \leq \Sigma(G) \leq \frac{n(n+1)}{2}$. Thus, by using a binary search procedure, we can find $\Sigma(G)$ by a sequence of at most $\left\lceil \log_2\left(\frac{n(n-1)}{2}\right) \right\rceil$ calls on $A^*$ giving to $A^*$ as inputs the graph $G$ and different values of $K$. \qed

1.4 An Algorithm for the Chromatic Sum of a Tree

The examples presented in the previous sections might suggest the following method of finding a best coloring of a given graph $G$: select a maximum independent set $S$ of vertices of $G$, color the vertices in $S$ with the smallest unused color, delete $S$ from $G$, and repeat the whole procedure until $G$ is empty. The next example shows however that even for relatively simple structures like trees, this algorithm doesn't work.

**Example 1.8** Consider the tree $T$ given in the Figure 1.6 together with its two proper colorings $c$ and $c'$. The coloring $c$ is a cheapest coloring of $T$ which
assigns the color 1 to all the vertices from the biggest independent set of the tree but it is not a best coloring of T—the coloring c' is cheaper.

The tree T described in Figure 1.6 can be used to prove an even stronger result. Let $t_n$ denote the number of trees of order $n$ and let $s_n$ denote the number of trees of order $n$ for which there exists a best coloring with the set of vertices of color 1 being a maximal independent set.

**Theorem 1.9** For almost all trees $T$ a proper coloring in which the set of the vertices of color 1 forms a maximum independent set is not a best coloring of $T$,

i.e. \( \frac{s_n}{t_n} \to 0 \) as $n \to \infty$. 

Figure 1.6 The color assignments from $c'$ are in the parentheses wherever they differ from the colors of $c$. Here $\Sigma^c(T) = 35$ and $\Sigma^{c'}(T) = 34$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Proof. Similar to the proof of the Corollary 1.4, we use the theorem due to A. Schwenk [14] stating that for every rooted tree L, almost every tree has L as a limb. To form the rooted tree L we add a new vertex x to the tree T in Figure 1.6, join x to the vertex w and designate x as the root. Now a tree F which have L as a limb will look like in the Figure 1.7 and any maximum independent set of F will contain the maximum independent set of T.

F:

```
   x
 T  w
```

Figure 1.7 A tree with L as a limb. The tree S is arbitrary.

Any cheapest coloring $c_1$ of the tree F for which the set of vertices of color 1 form a maximal independent set, coincides with the coloring c on T and any best coloring $c_2$ of the tree F coincides with the coloring $c'$ on T (in the case when $c_1(x) = 4$ or $c_2(x) = 1$ creating a conflict on edge xw we can always interchange vertices v and w to restore a proper coloring). Thus $\Sigma^{c_1}(F) > \Sigma^{c_2}(F)$. □

Next we present a linear algorithm for finding the chromatic sum for a tree of order $N$. We assume that the algorithm has certain information about the structure of the examined tree, i.e. at each vertex I it is known how many children this vertex has and there is an access to the first child of I and to the next sibling of I. The vertices are listed in a preorder traversal, that is the root of the tree is first and, if we go from the last vertex to the first, we encounter all the children vertices before we examine the parent vertex.

For each vertex I of the tree TREE[1] the program keeps the following information:
TREE[I].MINSUM – chromatic sum for the tree TREE[I] for which I is the root,

TREE[I].RCOLOR – color of the root of TREE[I] - vertex I in the best coloring produced so far by the algorithm,

TREE[I].DELTA – how much the sum of colors in the tree TREE[I] increases if we change the color of the root I,

TREE[I].NCOLOR – next best color for I,

TREE[I].NOSONS – number of children of vertex I,

TREE[I].SON – first child of vertex I,

TREE[I].BROTHER – next sibling of vertex I,

COLORADD[K] – the increase in the total of the chromatic sums TREE[J].MINSUM for all the sons J of a given vertex I when we insist on coloring vertex I with color K.

The algorithm starts from the leaves and goes up to the root. In steps 4 – 9 the values .MINSUM, .RCOLOR, .DELTA, .NCOLOR are easily determined for leaves: .MINSUM = 1, .RCOLOR = 1, .DELTA = 1, .NCOLOR = 2. In steps 11 – 48 those values are determined for a vertex I based on already known MINSUMs, RCOLORs, DELTAs, and NCOLORs for all children of vertex I. TREE[I].MINSUM gives the chromatic sum of the tree rooted at I. In steps 16 – 20 the algorithm examines exactly once every child of the vertex I. As a result TREE[I].MINSUM will contain the sum of the chromatic sums of the children of I and the array COLORADD will contain for every k, 1 ≤ k ≤ od(I) + 2, the amount by which the total of colors has to be increased if the vertex I receives the color k (od(I) denotes the number of children of I). Then in the steps 24 – 42, from the values COLORADD[1] through COLORADD[od(I)+2], the smallest two values SUM1 and SUM2 are selected and the corresponding colors COLOR1 and COLOR2 become the best and second best
choice of color for the vertex \( I \). The value \( \text{TREE}[I].\text{MINSUM} \) is incremented by \( \text{COLORADD}[\text{COLOR1}] \) giving the chromatic sum of the tree rooted at \( I \) and the penalty for changing the best color at \( I \) to the second best equals \( \text{COLORADD}[\text{COLOR2}] - \text{COLORADD}[\text{COLOR1}] \). It is necessary to examine all colors from 1 through \( \text{od}[I] + 2 \) since all the children of \( I \) can use different colors and still the best and the second best color for \( I \) could be different from those. On the other hand there is no need of considering more than \( \text{od}[I]+2 \) colors since in any optimal coloring a vertex of degree \( k \) can use at most color \( k+1 \).

The code of the algorithm:

```
PROGRAM TREE_CHROMATIC_SUM

INPUT: a tree \( T \) of order \( N \) given by an array of records \( \text{TREE}[1..N] \)

OUTPUT: the chromatic sum of \( T \)

1. For \( I = N \) downto 1 do
2.   if \( \text{TREE}[I].\text{NOSONS} = 0 \) then
3.     begin
4.       \( \text{TREE}[I].\text{MINSUM} = 1 \)
5.       \( \text{TREE}[I].\text{RCOLOR} = 1 \)
6.       \( \text{TREE}[I].\text{DELTA} = 1 \)
7.       \( \text{TREE}[I].\text{NCOLOR} = 2 \)
8.     end
9.   else
10.   begin
11.     \( \text{SON} = \text{TREE}[I].\text{SON} \)
12.     \( \text{MINTOTAL} = 0 \)
13.     for \( K = 1 \) to \( \text{TREE}[I].\text{NOSONS} + 2 \) do
14.       \( \text{COLORADD}[K] = K \)
15.     for \( K = 1 \) to \( \text{TREE}[I].\text{NOSONS} \) do
16.       begin
17.         \( \text{MINTOTAL} = \text{MINTOTAL} + \text{TREE}[\text{SON}].\text{MINSUM} \)
18.         \( \text{COLORADD}[\text{TREE}[\text{SON}].\text{RCOLOR}] = \text{COLORADD}[\text{TREE}[\text{SON}].\text{RCOLOR}] + \text{TREE}[\text{SON}].\text{DELTA} \)
19.         \( \text{SON} = \text{TREE}[\text{SON}].\text{BROTHER} \)
20.       end
21.     end
22.   \( \text{SUM1} = \infty \)
23.   \( \text{SUM2} = \infty \)
```
for K=1 to TREE[I].NOSONS+2 do
  begin
    VALUE=COLORADD[K]
    if (VALUE<SUM1) then
      begin
        COLOR2=COLOR1
        SUM2=SUM1
        COLOR1=K
        SUM1=VALUE
      end
    else
      if (VALUE<SUM2) then
        begin
          SUM2=VALUE
          COLOR2=K
        end
  end

TREE[I].MINSUM=SUM1+MINTOTAL
TREE[I].RCOLOR=COLOR1
TREE[I].DELTA=SUM2-SUM1
TREE[I].NCOLOR=COLOR2
end

Theorem 1.9 The complexity of the algorithm is $O(N)$.

Proof. We will sum up the number of instructions executed for each vertex $I$ in a given tree of order $N$. The total number of instructions is at most

$$
\sum_{I=1}^{N} (8 + (TREE[I].NOSONS + 2)\cdot10) = 28N + 10(N-1). \quad \Box
$$

Remark 1.10 It is also possible to recover the best coloring produced by the algorithm at a cost of $O(N)$.

Example 1.11 Consider the tree $T$ depicted in Figure 1.8 and assume that the algorithm TREE_CHROMATIC_SUM has been applied to all the vertices of $T$ but the root $r$. Thus the values .MINSUM, .RCOLOR, .NCOLOR and .DELTA are known for them and are listed in that order.
Before examining the children of \( r \), we initialize the array \( \text{COLORADD} \):
\[
\text{COLORADD}[i] = [i] \text{ for } 1 \leq i \leq 4+2.
\]
Then every child node is looked at exactly once and as a result we have:

\[
\text{TREE}[r].\text{MINSUM} = 15
\]

\[
\text{COLORADD}[1] = 1 + 1 + 1 + 1
\]

\[
\text{COLORADD}[2] = 2 + 1 + 1.
\]

For \( i \) different from 1 and 2 the \( \text{COLORADD}[i] \) don't change. Now the algorithm compares the values \( \text{COLORADD}[i], \ 1 \leq i \leq 6, \) to select the smallest one (\( \text{COLORADD}[3] \)) and the second smallest (\( \text{COLORADD}[1], \text{ or COLORADD}[2], \text{ or COLORADD}[4] \)). In case of a tie the smallest color is selected. Thus finally we have:

\[
\text{TREE}[r].\text{MINSUM} = 15 + \text{COLORADD}[3] = 18
\]

\[
\text{TREE}[r].\text{RCOLOR} = 3
\]

\[
\text{TREE}[r].\text{NCOLOR} = 1
\]

\[
\text{TREE}[r].\text{DELTA} = \text{COLORADD}[1] - \text{COLORADD}[3] = 1.
\]
CHAPTER II

GRAPHS THAT REQUIRE MANY COLORS TO ACHIEVE THEIR CHROMATIC SUM

2.1 Existence

In Chapter I we show that for every natural number $k$, there exists a tree requiring at least $k$ colors to achieve its chromatic sum. Moreover, we construct the unique smallest tree, $T_k$, which requires $k$ colors to obtain its chromatic sum. Its order is given by

$$|T_k| = \frac{1}{\sqrt{2}} \left[ (2 + \sqrt{2})^{k-1} - (2 - \sqrt{2})^{k-1} \right].$$

In other words the number of colors that must be used in a best coloring can exceed the chromatic number by an arbitrarily large value.

This unexpected property is not only true for trees but also occurs for graphs with higher chromatic number. In this chapter we investigate this property of the chromatic sum. We start with an existential theorem.

Theorem 2.1 For every integer $k \geq 2$ and every positive integer $t$, there exists a $G_k^t$ graph, that is, a $k$-chromatic graph which must use at least $k + t$ colors to obtain its chromatic sum.

Proof. We will construct an instance of a $G_k^t$ graph by using the rooted trees $T_i^m$ introduced in Chapter I. Tree $T_1^1$ is the trivial tree with one vertex. The tree $T_i^m$ is defined recursively by listing the branches that remain upon removing its root $r$, namely we have
\[ T^m_i - r = \bigcup_{n=1}^{i-1} (m + i - n)T^1_n. \] It was proved in Chapter I that \( T^m_i \) is the smallest rooted tree for which in every best coloring color \( i \) is forced to appear at the root and any change of that color to a lower one must increase the sum of colors by at least \( m \).

**CONSTRUCTION A:**

![Figure 2.1 A k-chromatic graph that requires \( t \) extra colors.](image)

We obtain our first \( G^t_k \) graph called \( A^t_k \) from a complete graph \( K_k \), by attaching at each vertex \( v_i \), the rooted tree \( T^1_{t+i} \). Since the best coloring of each \( T^1_{t+i} \) requires color \( t+i \) at the root, the union of these colorings yields a proper coloring and hence must be the best coloring of \( A^t_k \). \( \square \)

Construction A is very simple but also very costly. It produces graphs of unnecessarily large order. Let \( t^1_i = |T^1_i| \). We know from the proof of Theorem 1.3 that \( t^1_i = 4t^1_{i-1} + 2t^1_{i-2} = 0 \). Given the starting values \( t^1_1 = 1, t^1_2 = 3, t^1_3 = 10 \) we can routinely solve this recurrence relation to find that

\[ |T^1_i| = \frac{1}{4} [ (2 + \sqrt{2})^i + (2 - \sqrt{2})^i]. \]

Therefore the order of \( A^t_k \), given by construction A grows exponentially in \( (k + t) \), namely

\[ |A^t_k| = \frac{\sqrt{2}}{4} [(2+\sqrt{2})^{k+t} - (2+\sqrt{2})^t - (2-\sqrt{2})^{k+t} - (2-\sqrt{2})^t]. \]
Figure 2.2 A 3-chromatic graph that requires 6 colors to obtain its chromatic sum

For example $|A_3^3| = |T_4^1| + |T_5^1| + |T_6^1| = 34 + 116 + 396 = 546$.

Our goal in the rest of this chapter is to find better constructions that produce this behavior on graphs of much smaller order. In fact, we are able to construct a 3-chromatic graph requiring 6 colors on only 108 vertices, a 5-fold improvement on $A_3^3$ in Figure 2.2.

2.2 A More Efficient Construction for Relatively Few Extra Colors

The construction we are about to present is valid only when the number of extra colors $t$ is smaller than the chromatic number $k$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CONSTRUCTION B:

Each of the $m$ copies of $K_t$ in Figure 2.3 is joined to $k - t$ carefully selected vertices in $K_k$ below. This forms a $k$-chromatic graph because $t$ colors are available to color each copy of $K_t$ (of course $t$ must be smaller than $k$). For the $j$-th copy of $K_t$, we specify this selection by identifying the $t$ vertices of $K_k$ not joined to copy $j$. Every vertex in copy $j$ is joined to all but vertices $v_{(j-1)t+1}$ through $v_{jt}$ where all subscripts are taken modulo $k$. We would like to find the smallest possible value of $m$ for which the graph $B^t_k$ obtained by construction B requires $k + t$ colors in its best coloring. Thus we seek the smallest $m$ for which $\Sigma_{k+t} < \Sigma_{k+s}$ for all $s < t$.

We will show that for $s \leq t$, $\Sigma_{k+s}$ is achieved when first, in every copy of $K_t$, we use colors 1 up to $s$ on $s$ vertices, next we use colors $s + 1$ through $s + k$ to color $K_k$ and finally the remaining vertices of the copies of $K_t$ are colored as cheaply as possible (let's call this coloring $c_s$).

Assume this is not true. Let us consider a cheapest $k+s$-coloring of $B^t_k$ which has the biggest possible sum of colors when restricted to just the vertices
of $K_k$. Let $r \leq k + s$ be the biggest color not used in $K_k$. Since from the assumption some color smaller than $s$ is used on $K_k$, we must have $r > s$. If it happens that $r \leq t$, then consider a vertex $x$ of $K_k$ colored with some color smaller than $r$, say $n$. Consider a copy of $K_t$, say $H$, connected to $x$. Every $K_t$ has to use the $t$ cheapest available colors. For $H$, color $r \leq t$ is available and thus $H$ contains a vertex $v$ colored with $r$. Now we can interchange colors $r$ and $n$ for the vertices $v$ and $x$. If there is any vertex $v_i$ in some other copy of $K_t$ colored with $r$ and adjacent to $x$, we may also change that color to $n$ which was not available before. Thus we can obtain an equivalent or even cheaper coloring of $B_k^t$ with the biggest color not used in $K_k$ exceeding $r$, which gives a contradiction. On the other hand, if $r > t$, consider a vertex $v$ in some copy of $K_t$ colored with $r$ (color $r$ has to be used somewhere) together with all its $(k-t)$ neighbors from $K_k$. One of them, say $x$, has to use a color smaller than $r$, say $n$. Otherwise we would have: $(k-t) + r \leq k$, so $r \leq t$. As before we interchange colors $r$ and $n$, obtaining a contradiction. Therefore $r \leq s$ and the vertices of $K_k$ have to be colored with $s + 1$ through $s + k$.

It is helpful to recall that $1 + 2 + \ldots + t = \binom{t+1}{2}$. Thus

$$\Sigma_{k+t} = m\binom{t+1}{2} + \binom{k+1}{2} + tk.$$  

When using only $k + s$ colors the sum is $\Sigma_{k+s} = m\binom{s+1}{2} + L_s + \binom{k+1}{2} + ks$, where $L_s$ is the cheapest possible sum of colors over the $m(t-s)$ vertices of the copies of $K_t$ when in every copy the first $s$ vertices use colors $1$ through $s$ and the vertices of $K_k$ are colored $s+1$ through $s+k$. Thus the increase of the sum of the colors over those $m(t-s)$ vertices, called $D_s$, compared to the sum of colors in the coloring $c_t$ has to be bigger than the decrease we get on $K_k$. Therefore we need $D_s > (t - s)k$. Let $G_s$ denote the graph obtained from $B_k^t$ by deleting $s$ vertices from every copy of $G_k$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
K_t. Notice that the coloring c_s, where the colors 1 through s are assigned to the removed vertices, can be transformed to a best coloring c_s' of G_s by diminishing every color by s. Let m be large enough to assure for B_k^+ the inequality: Σ_k+t < Σ_k+t-1. Thus when we sum the difference between the colors assigned to v_i in c_{t-1} and the color t assigned in c_t, we get: D_{t-1} = Σ_{i=1}^{m} (c_{t-1}(v_i) - t) > k.

Consider now the graph G_{t-1}. Subtracting the value t-1 from every c_{t-1}(v_i) we arrive at the inequality: Σ_{i=1}^{m} c_{t-1}(v_i) > k + m.

\[ G_{t-1} \]

For the graph G_{t-2}, in any best coloring, we can not color w_i cheaper than using the color of v_i increased by one. Therefore Σ_{i=1}^{m} [c_{t-2}(v_i) + c_{t-2}(w_i)] > 2k + 3m.

Thus in the graph B_k^+ we obtain

\[ D_{t-2} = Σ_{i=1}^{m} [c_{t-2}(v_i) + c_{t-2}(w_i) - (2t-1)] = Σ_{i=1}^{m} [c_{t-2}(v_i) + c_{t-2}(w_i) + 2(t-2)-(2t-1)] > 2k. \]

Similarly for any s < t, we find D_s > (t-s)k. Therefore, in construction B, it is enough to find the smallest possible m such that Σ_{i=1}^{m} c_{t-1}(v_i) - mt > k.

Case 1. Assume that mt < k.

Here every vertex of K_k is not connected to exactly one K_t. Thus all v_i's must receive different colors and Σ_{i=1}^{m} c_{t-1}(v_i) = m(t-1) + \left(\frac{m+1}{2}\right). The required inequality is m^2 - m > 2k, which simplifies to \( m > \frac{1}{2} + \sqrt{2k + \frac{1}{4}} \).
All the computations were done under the assumption that \( mt \leq k \), which means that also \( k > \frac{t}{2} + \sqrt{2kt^2 + \frac{t^2}{4}} \), leaving us with the inequality \( t < -1 + \frac{\sqrt{1 + 8k}}{4} \) or \( k > 2t^2 + t \).

**Case 2.** Assume that \( mt > k \) (or \( t \geq \frac{\sqrt{1 + 8k} - 1}{4} \)).

Now, because \( mt > k \), we have vertices of \( K_k \) whose neighborhoods are disjoint from more than one copy of \( K_t \). Therefore colors on some vertices \( v_i \) might be repeated. Consider the following condition: every vertex of \( K_k \) is connected to \( k+1 \) copies of \( K_t \). Now, all the \( K_t \)'s connected to a vertex of \( K_k \) colored with \( t \) must use at least color \( t+1 \) at the corresponding vertices \( v_i \) and the value \( D_{t-1} \) exceeds \( k \).

In order to achieve that condition we must have \( m \geq \frac{k(k+1)}{k-t} \) (because we have \( k \) vertices, each connected to \( k+1 \) copies of \( K_t \) and every copy of \( K_t \) connected to \( k - t \) vertices of \( K_k \)). In fact, for \( t \geq \frac{k}{2} \), the smallest possible \( m \) (using construction B) is \( \left\lceil \frac{k(k+1)}{k-t} \right\rceil \). For any vertex \( x_i \) of \( K_k \) we notice that \( v_{i+k/2} \) is never connected to the same copy of \( K_t \). Thus all vertices \( v_i \) in \( K_t \)'s which can not use color \( t \), may be colored with \( t+1 \). However, when \( t < \frac{k}{2} \) the cheapest coloring using \( k + t - 1 \) colors forces also color \( t + 2 \) or bigger to appear, in which case we don't need \( \left\lceil \frac{k(k+1)}{k-t} \right\rceil \) copies of \( K_t \). Now consider \( \frac{k}{3} \leq t < \frac{k}{2} \). For simplicity we assume that \( m = k t \). Having just \( k \) copies of \( K_t \), every vertex \( x_i \) of \( K_k \) is not connected to \( t \) copies of \( K_t \). Consider the coloring \( c_{t-1} \). Let \( c_{t-1}(x_1) = t \), \( c_{t-1}(x_{s+1}) = t+1 \), where \( s \leq \frac{k}{2} \). Those \( t \) vertices \( v_i \) not connected to \( x_1 \) can use color \( t \), also \( t \) vertices not connected to \( x_{s+1} \) can use color \( t+1 \). Thus the number of vertices colored with \( t + 1 \) is \( t \) diminished by the number of those where color \( t \) is also available, so \( t - (k - t - s) = 2t + s - k \). The remaining vertices have to be colored with at least \( t + 2 \) and we have \( (2k - 3t - s) \) of them. Let \( d \) be the
difference between the sum of colors, taken only over the vertices of those copies of $K_t$. Then

$$d = \min_k((2t + s - k) + (2k - 3t - s) - 2) = \left\lceil \frac{5k}{2} \right\rceil - 4t.$$  

We want to obtain $D_{t-1} = \frac{k}{d} > k$ or $m = k \cdot \frac{k^2}{d}$. This finally simplifies to the following inequality: $m > \frac{k^2}{\frac{5k}{2} - 4t}$. Of course for $t < \frac{k}{3}$ this many copies of $K_t$ is also enough but we can find an even better bound by slightly modifying construction $B$. In this modified construction the $i$-th copy of $K_t$ is connected to all vertices of $K_k$ but $x_i$ through $x_{i+t}$ where all subscripts are taken modulo $k$. As before we have vertices $v_1$ through $v_m$ to color. Assume now that $m < k$. Thus any $t+1$ copies of $K_t$ have to cover all the vertices of $K_k$ and consequently no more than $t$ copies can use the same color for their remaining vertex. Therefore to obtain a $G_k^t$ graph, it is enough to take $m$ satisfying the following:

$$t + 2t + 3t + \ldots + (m - 1)t > k,$$

which can be simplified to: $m > \frac{t}{2} + \sqrt{2tk + \frac{t^2}{4}}$.

It is easy to check that $m$ satisfying that inequality is smaller than $k$ for $t < \frac{k}{3}$.

The preceding discussion proves our next theorem.

**Theorem 2.2** For each interval of $t$ shown below, the constructions described above yield a $G_k^t$ graph, provided that $m$ lies in the range specified as follows:

- **a)** for $t < \frac{-1 + \sqrt{1 + 8k}}{4}$, it suffices to have $m > \frac{1}{2} + \sqrt{2k + \frac{1}{4}}$;
- **b)** for $t < \frac{k}{3}$, it suffices to have $m > \frac{1}{2} + \sqrt{2tk + \frac{t^2}{4}}$;
- **c)** for $t < \frac{k}{4}$, it suffices to have $m > \frac{k^2}{\frac{5k}{2} - 4t}$;
- **d)** for $t < k$, it suffices to have $m > \frac{k(k+1)}{k-t}$.
Let \( p(k,t) \) denote the minimum order among all \( k \) chromatic graphs whose chromatic sum requires \( t \) extra colors. The above theorem gives us only an upper bound for \( p(k,t) \). However, for \( t \) equal to 1 this bound is exact.

**Theorem 2.3** For any integer value \( k \geq 4 \), the smallest order of any \( k \)-chromatic graph which must use \( k+1 \) colors to obtain its chromatic sum is equal to

\[
p(k,1) = \left\lfloor \frac{1}{2} + \sqrt{2k + \frac{1}{4}} \right\rfloor + 1 + k.
\]

For \( k = 3 \) and \( k = 2 \), \( p(k,1) = 8 \).

**Proof.** Case a) \( k \geq 4 \).

The upper bound of \( p(k,1) \) is given in the Theorem 2.2. For \( t = 1 \) we have \( 2t^2 + t = 3 < k \) and therefore \( p(k,1) \leq \frac{1}{2} + \sqrt{2k + \frac{1}{4}} + 1 + k \). Let \( G \) be a \( k \)-chromatic graph, such that \( \Sigma_{k+1}(G) < \Sigma_k(G) \). Also let \( |G| = k + m \). The cheapest way to color \( G \) with \( k + 1 \) colors is to use colors 2 through \( k + 1 \) once and color all remaining \( m + 1 \) vertices with 1. Thus \( \Sigma_{k+1}(G) \geq \binom{k+2}{2} + m \).

With only \( k \) colors available the worst situation is when no color appears more than twice. Thus colors 1 through \( k \) are used once and the remaining \( m \) vertices use different color, which can be taken to be 1 through \( m \). Therefore

\[
\Sigma_{k+1}(G) \leq \binom{k+1}{2} + \binom{m+1}{2}.
\]

Because \( \Sigma_{k+1}(G) < \Sigma_k(G) \), we must have \( \binom{k+1}{2} + \binom{m+1}{2} > \binom{k+2}{2} + m \), which simplifies to \( m > \frac{1}{2} + \sqrt{2k + \frac{1}{4}} \) and therefore

\[
p(k,1) = \left\lfloor \frac{1}{2} + \sqrt{2k + \frac{1}{4}} \right\rfloor + 1 + k \quad \text{as required.}
\]

Case b) \( k = 3 \).

Consider the following graph \( G \):
Figure 2.5 A minimal 3-chromatic graph with $\Sigma_4(G) < \Sigma_3(G)$.

The order of $G$ is $|G| = 8$. It is easy to check that $G$ is 3-chromatic, $\Sigma_3(G) = 15$ and $\Sigma_4(G) = 14$. Thus $p(3,1) \leq 8$. To see the reverse inequality consider any 3-colorable graph $H$ of order 7. The vertices of $H$ can be divided into three partite sets with the biggest one of order at least 3. Thus $\Sigma_3(H) \leq 3 \cdot 1 + 2 \cdot 2 + 2 \cdot 3 = 13$. There are only three ways to partition 7 as a sum of four nonincreasing terms: $7 = 2 + 2 + 2 + 1$ or $7 = 3 + 2 + 1 + 1$ or $7 = 4 + 1 + 1 + 1$, where the integers represent the number of vertices colored with 1, 2, 3 or 4 in the order given. Thus we have only three ways to color the graph $H$ using four colors, and these sum to 16, 14, and 13 respectively. Therefore $\Sigma_4(H) \geq \Sigma_3(H)$, which implies that $p(3,1) > 7$.

Case c) $k = 2$.

Consider the following graph $F$:
Here we have $|F| = 8$, $\Sigma_3(F) = 11$ and $\Sigma_2(F) = 12$, so $p(2,1) \leq 8$. As before we consider any bipartite graph $H$ of order 7. The graph $H$ must have one partite set of order at least 4. Thus $\Sigma_2(H) \leq 4 + 3 \cdot 2 = 10$. Using three colors we must use color 2 and 3 at least once and therefore $\Sigma_3(F) \geq 2 \cdot 1 + 3 \cdot 1 + 5 \cdot 1 = 10$. Consequently $\Sigma_3(F) > \Sigma_2(F)$ and $p(2,1) > 7$. □

2.3 A Construction for an Arbitrary Number of Extra Colors

Using construction B we can build a graph which uses extra colors to obtain its chromatic sum, but always the number of additional colors is restricted to be smaller than the chromatic number. To overcome that limitation we need to modify construction B.

**CONSTRUCTION C.**

Start with a $k$-chromatic graph $G$ which happens to satisfy the following two conditions:

a) $\Sigma(G) = \Sigma_s(G)$ for some $s > k$.

b) In every best coloring of $G$, the number of vertices of color 1 is strictly greater than the order of the biggest set in any partition of $G$ into $k$ independent sets.

Let $|G| = p$ and let $W_1, W_2, ..., W_k$ be a decomposition of the vertices of $G$ into $k$ independent sets. Construction C build a new graph $A_1(G)$ from $G$ by adding $k$ groups of vertices $V_1, V_2, ..., V_k$, each containing $p$ independent vertices.
Then all possible edges between all vertices of $W_i$ and all vertices of $V_i$, for $1 \leq i \leq k$, are added. It's easy to see that $A_1(G)$ is still $k$ chromatic and $|A_1(G)| = (k + 1) |G|.$

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{construction.png}
\caption{The description of the construction $C$}
\end{figure}

Lemma 2.4 Let $G$ be a $k$-chromatic graph satisfying conditions a) and b) above. Then the graph $A_1(G)$ must use one more color than $G$ does to achieve its chromatic sum.

Proof. Let the minimum number of colors in a best coloring of $G$ be denoted by $s$. One possible strategy for coloring $A_1(G)$ is to assign color 1 to all the vertices of $V_i$ for all $1 \leq i \leq k$. Let us call this coloring $c$. We must complete this coloring by coloring the remaining vertices as cheaply as possible. Clearly we should use a best coloring of $G$ with every color increased by 1. Therefore colors 2 through $s + 1$ are forced to be used and furthermore $\sum_{v \in A_1(G) \setminus G} c(v) = k \cdot p$ and $\sum_{v \in G} c(v) = \Sigma(G) + p$.

We intend to show that $c$ is better than any other coloring $c'$ that uses only $s$
colors. Assume it's not true. Then there must be some vertices of $G$ colored by $c'$ with 1. Let $j$ denote the number of different partite sets $W_i$ which contain those vertices. Then all vertices in corresponding sets $V_i$ have to use colors bigger than 1 in $c'$ and therefore $\sum c(v) \geq k'p + j'p$. In order for $c'$ to be a best coloring of $\forall v \in A_1(G) - G$

$A_1(G)$ we must have $\sum c(v) \leq \sum c(v) - j'p = \Sigma(G) + p - j'p = \Sigma(G) - (j-1)p$.

This is a contradiction for $j > 1$.

Thus all vertices of $G$ colored with 1 must be contained in one of the sets $W_i$. This implies, together with condition b) for $G$, that $c'$ is not a best coloring for $G$. Therefore $\sum c'(v) > \Sigma(G) + k'p + p = \sum c(v)$, a contradiction. □

We can easily modify the construction $C$ to produce a graph $A_t(G)$ which must use $t$ extra colors in its chromatic sum beyond the $s$ extra colors that $G$ must use. The graph $G$ satisfies conditions a) and b'), where condition b') is defined as follows:

b') In every best coloring of $G$, for every color $i < t$, the number of vertices of color $i$ is strictly greater than the order of the biggest set in the $k$-partition of $G$.

Instead of single vertices, every $V_i$ contains $|G|$ copies of $K_t$. For every copy of $K_t$ in $V_i$, every vertex of $K_t$ is adjacent to every vertex in $W_i$. The order of $A_t(G)$ is: $|A_t(G)| = (kt + 1) \cdot |G|$. This can be summarize in the following lemma.

**Lemma 2.5** Let $G$ be a $k$-chromatic graph which satisfies conditions a) and b'). Then the graph $A_t(G)$ must use $t$ more colors in its best coloring beyond the colors that must be used for $G$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Example 2.6  Consider the following $G_3^2$ graph $G$ obtained using construction B.

Of course $G$ satisfies a) and b). Applying the construction C we get $A_1(G)$:

Figure 2.8 A $G_3^2$ graph

Figure 2.9 A $G_3^3$ graph. The vertex $w_1$ is adjacent to the second partite set of $G$, $x_1$ to the third. The vertices $v_i, w_i, x_i$ are adjacent to the same vertices as $v_1, w_1, x_1$ respectively.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The graph $A_3(G)$ is a $G^3$ graph of order 108 which is one fifth the order of the $A_3$ graph given in Figure 2.2.

Combining construction B and C we can now build a $G^t_k$ graph for any value of $k$ and $t$. We can represent $t$ as $t = r(k-1) + s$, for some values $r$ and $s < k-1$. First construct the $G^k_{k-1}$ graph using construction B. Then apply construction C recursively $r$ times always to the graph just obtained (all of them will satisfy conditions a) and b'). This gives us the upper bound presented in the next theorem.

**Theorem 2.7** For any natural number $k \geq 2$ and $t \geq 1$,

$$p(k,t) \leq (k(k-1) + 1)^{[t/(k-1)]} (k t \text{mod} (k-1) + 1) k^3.$$  

**Proof.** The order of $G^k_{k-1} \leq k(k+1)(k-1) + k = k^3$. □

We already mentioned that the smallest tree that requires $t$ extra colors to achieve its chromatic sum is $T_{t+2} = \frac{1}{\sqrt{2}} [(2 + \sqrt{2})^{t+1} - (2 - \sqrt{2})^{t+1}]$. However for bipartite graphs in general the value of $p(2,t)$ is smaller.

**Corollary 2.8** For any natural number $t \geq 1$, $p(2,t) \leq 8 \cdot 3^{t-1}$.

**Proof.** From Theorem 2.3 we have $p(2,1) = 8$. □

Assume finally that we don't specify the chromatic number. Then we have our last result.

**Corollary 2.9** The minimum order of a graph $G$ that has to use $t$ extra colors to achieve its chromatic sum is bounded above by $|G| \leq (t+1)^3$.

**Proof.** For the graph $G$ we can take the $G^t_{t+1}$ graph obtained in Construction B. Then its order is $(t + 1)^3$. □
CHAPTER III

CONSTRAINTS ON THE CHROMATIC SEQUENCE FOR TREES AND GRAPHS

3.1 Introduction

In [1] Alavi, Malde, Schwenk, and Erdös studied constraints on sequences \( a_1, a_2, \ldots, a_n \) of parameters associated with a specific graph. Sorting a sequence \( \{a_i\} \) into nondecreasing order defines a permutation \( \pi \) on the indices in the following way:

\[
a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}.
\]

A family of sequences for different graphs and one specific parameter is called constrained if certain permutations \( \pi \) are never realized by any graph. The family is unconstrained if, for each \( n \), every permutation on \( n \) indices can be realized by some graph.

Recall from Chapter I the chromatic sequence of a graph \( G \), that is, the sequence of \( k \)-chromatic sums \( \Sigma_k \), where \( k \) is between the chromatic number \( \chi(G) \) and the Grundy number \( \Gamma(G) \).

In this chapter we wish to study the chromatic sequences associated with specific graphs. We shall show that for trees the chromatic sequence is constrained but for arbitrary graphs it is not.
3.2 Chromatic Sequences for Trees

A sequence $a_1, a_2, \ldots, a_n$ is called \textit{unimodal} if

$$a_1 < a_2 < \ldots < a_s \geq a_{s+1} > a_{s+2} > \ldots > a_n.$$  

This terminology is borrowed from statistics for distributions with a single local maximum. It happens that the chromatic sequence for trees resembles this pattern, with the inequalities all reversed. We say a sequence $a_1, a_2, \ldots, a_n$ is \textit{inverted unimodal} if there exists $s, 1 \leq s \leq n$, such that

$$a_1 > a_2 > \ldots > a_s \leq a_{s+1} < a_{s+2} < \ldots < a_n.$$  

Such a sequence has a unique local minimum. For infinite sequences unimodal and inverted unimodal are defined analogously.

In this section we show that, for trees, the chromatic sequence is indeed inverted unimodal. We start with presenting a preliminary lemma.

\textbf{Lemma 3.1} Let $\{a_i\}$ be a sequence, finite or infinite, of numbers such that

(1) If $a_k \leq a_{k+2}$, then $a_{k+1} < a_{k+2}$ and

(2) If $a_k \geq a_{k+2}$, then $a_k > a_{k+1}$.

Then $\{a_i\}$ must be inverted unimodal.

\textbf{Proof.} The proof is done in three cases, identified by the relative size of $a_k$ and $a_{k+1}$.

\textbf{Case 1.} If $a_k = a_{k+1}$, then $a_{k-1} > a_k$ and $a_{k+1} < a_{k+2}$.

If $a_{k+1} \geq a_k$, then $a_k \geq a_{k+2}$, which from (2) implies that $a_k > a_{k+1}$.

If $a_{k-1} \leq a_k$, then $a_{k-1} \leq a_{k+1}$, which from (1) implies that $a_k < a_{k+1}$.

Both contradict the hypothesis for this case.
Case 2. If \( a_k < a_{k+1} \), then \( a_{k+1} < a_{k+2} \).

If \( a_k > a_{k+2} \), then \( a_k > a_{k+1} \), which is a contradiction. Thus the only possibility is \( a_k \leq a_{k+2} \), but by (1) this implies \( a_{k+1} < a_{k+2} \).

Case 3. If \( a_k > a_{k+1} \) then \( a_{k-1} > a_k \).

If \( a_{k-1} \leq a_k \), then \( a_k < a_{k+1} \), which is a contradiction. Thus the only possibility is \( a_{k-1} > a_k \). \( \square \)

Theorem 3.2 Let \( T \) be a tree and \( s \) be the smallest number of colors for which the chromatic sum is achieved. Then the chromatic sequence is inverted unimodal, that is, we have the following sequence of inequalities:

\[
\Sigma_2(T) > \ldots > \Sigma_{s-1}(T) > \Sigma_s(T) \leq \Sigma_{s+1}(T) < \Sigma_{s+2}(T) < \ldots < \Sigma_{I}(T).
\]

Proof. In view of Lemma 3.1, the proof consists of two cases.

Case 1. If \( \Sigma_k(T) \leq \Sigma_{k+2}(T) \) then \( \Sigma_{k+1}(T) < \Sigma_{k+2}(T) \).

Consider a coloring \( A \) of \( T \) such that \( \Sigma^A = \Sigma_{k+2} \), for which the number of vertices colored with the color \( k+1 \) is minimal. Let this number be denoted by \( r \).

If \( r = 0 \) then by changing each color \( k+2 \) into \( k+1 \) we obtain a cheaper coloring using only \( k+1 \) colors, and therefore \( \Sigma_{k+1} < \Sigma_{k+2} \).

If \( r > 0 \) then every vertex colored with \( k+2 \) is adjacent to a vertex colored with \( k+1 \). Together with coloring \( A \) consider three other colorings of \( T \), named \( B \), \( C \), and \( D \), where \( B \) is the cheapest coloring using only \( k \) colors and \( C \) and \( D \) are the mixtures of colorings \( A \) and \( B \) as indicated in Figure 3.1. It is impossible to have \( \Sigma^B(T_1) \leq \Sigma^A(T_1) \) because then, for the coloring \( C \), we would have \( \Sigma^C(T) \leq \Sigma^A(T) \) and either coloring \( C \) would be cheaper than \( A \) or of equal value but with the number of vertices of color \( k+1 \) smaller than \( r \).
Thus $\Sigma^B(T_1) > \Sigma^A(T_1)$ which implies that $\Sigma^B(T_2) < \Sigma^A(T_2)$. Now $\Sigma^D(T) < \Sigma^A(T) = \Sigma_{k+2}(T)$. Since $A$ is the cheapest coloring using color $k+2$, coloring $D$ must use colors only up to $k+1$ and thus $\Sigma_{k+1}(T) < \Sigma_{k+2}(T)$.

Case 2. If $\Sigma_k(T) \geq \Sigma_{k+2}(T)$ then $\Sigma_k(T) > \Sigma_{k+1}(T)$.

Consider again colorings $A$, $B$, $C$, and $D$. Let $v$ be such a vertex of $T$ that is colored in $A$ by the color $k+2$ and is farthest from the center. Let $u$ be a vertex adjacent to $v$, colored in $A$ by $k+1$ and situated farthest from the center. Then either

a) $T_1$ doesn't contain a vertex colored in $A$ by the color $k+2$

or

b) $T_2$ doesn't contain a vertex colored in $A$ by the color $k+2$ and no vertex adjacent in $T_2$ to $v$ is colored in $A$ by $k+1$.

Just as before, we have $\Sigma^B(T_1) > \Sigma^A(T_1)$ and $\Sigma^B(T_2) \geq \Sigma^A(T_2)$. 

Figure 3.1
If a) is true, then $\Sigma^D(T) < \Sigma^B(T) = \Sigma_k(T)$, and because coloring $D$ doesn't use color $k+2$, we obtain $\Sigma_{k+1}(T) < \Sigma_k(T)$.

If b) is true, consider the coloring $C'$ which is a slight modification of coloring $C$. The only difference is at the vertex $v$, namely $C'(v) = k+1$ instead of using color $k+2$. Therefore $\Sigma^C(T) < \Sigma^B(T) = \Sigma_k(T)$, and since coloring $C'$ doesn't use color $k+2$ we obtain $\Sigma_{k+1}(T) < \Sigma_k(T)$.

Now from steps 1, 2 and from Lemma 3.1 the theorem follows. □

3.3 Chromatic Sequences for Graphs

In a view of Theorem 3.2 a very natural question arises: is a similar inverted unimodal behavior true for general graphs? The answer to this question is negative. First, we shall show that multiple ties are possible as depicted in Example 3.3. In Figure 3.2 three proper colorings are indicated: the cheapest coloring using four, three, and two colors, in that order. It happens that

$$\Sigma(G_4) = \Sigma_4(G_4) = \Sigma_3(G_4) = \Sigma_2(G_4) = 44.$$  

Thus the chromatic sum can be attained using three different maximum colors, while for a tree a maximum color in any best coloring can be attained by at most two different values and these must be consecutive.
The construction used to build the graph $G_4$ can be generalized to produce ties of an arbitrary multiplicity. Before showing that, we present a preliminary lemma.

Recall from Chapter I the tree $T^1_k$, that is the smallest rooted tree that requires the color $k$ to appear at the root to achieve its chromatic sum. Note that the best coloring for the tree $T^1_k$ is uniquely determined. Let us call this coloring the coloring $c$. All the vertices of $T^1_k$ can be divided into two independent sets, $A$ and $B$. We select one partite set, say $A$. We then define, for $1 \leq i \leq k-2$, the set $V_i$ to be the set of all vertices from $A$ of color $i$ in the coloring $c$. From the construction of the tree $T^1_k$
we know that $\Sigma_{k-1}(T_k^1) = \Sigma_k(T_k^1) + 1$. The difference between $\Sigma_1$ and the chromatic sum of $T_k^1$ is given in the Lemma 3.4.

**Lemma 3.4** Let $i \leq k - 2$ and let $c'$ be a cheapest coloring of $T_k^1$ in which the vertices from $V_i$ receive colors different from $i$. Then the coloring $c'$ is a cheapest $(i+1)$-coloring of $T_k^1$ and $\Sigma c'(T_k^1) = \Sigma_{i+1}(T_k^1) = \Sigma(T_k^1) + |T_{k-i-1}^1|$. Moreover, the coloring $c'$ doesn't use the color $i$ at any vertex from $A$ and at the root $r$ we have $c'(r) = i$ if $r \not\in A$ or $c'(r) = i + 1$ if $r \in A$.

**Proof.** The proof will be by induction on $k$. For $T_2^1$, in each selection of a partite set, changing the color of the vertices from the set $V_1$ forces at least five vertices of color bigger than 1 and therefore a cheapest 2-coloring of $T_2^1$ is forced. This coloring satisfies the lemma as shown in Figure 3.3.

Assume now that the lemma is true for $t < k$. Denote by $r$ the root of $T_k^1$. Recall now that $T_k^1 - r = \bigcup_{t=1}^{t=k-1} (k-t+1)T_t^1$. In each of the branches $T_t^1$ let $A_t = T_t^1 \cap A$ and $V_i^t = T_t^1 \cap V_i$. For $t \geq i + 2$ we use the inductive hypothesis to assure the cost
and properties of colorings $c'$ at each $T^1_t$. For $t < i$ the set $V^1_i$ is empty and we don't have to change the best coloring $c$ on any of those $T^1_t$ trees. For $t = i$ or $t = i + 1$ we consider two possibilities.

If the root $r$ of $T^1_k$ does not belong to the partite set $A$ then the coloring $c$ does not change over $k - i$ copies of $T^1_{i+1}$ since the sets $V^1_{i+1}$ are empty for those trees. For the $k - i + 1$ copies of $T^1_i$ only their roots bear the color $i$ in the coloring $c$. The cost of changing that color for each $T^1_i$ tree is 1. However, color $i$ is now available at $r$ since all its neighbors use either color $i + 1$ (in the case of the roots of $T^1_t$ for $t \geq i$) or a color smaller than $i$ (in the case of the roots of $T^1_t$ where $t < i$). Therefore the overall cost of change is $(k - i + 1) - (k - i) = 1$.

If $r$ belongs to the partite set $A$ then the sets $V^1_i$ for the trees $T^1_i$ are empty and coloring $c$ doesn't change for those trees. On the other hand, each one of $k - i$ copies of $T^1_{i+1}$ has the root adjacent to the only two vertices for that $T^1_{i+1}$ tree colored in $c$ with the color $i$. Those vertices are the roots of an $T^1_i$ tree. As before, the change of color $i$ for each of $T^1_{i+1}$ tree costs 1 and results in assigning color $i$ at each root. Every vertex adjacent to $r$ is a root of a $T^1_t$ tree. If $t < i$ then its color is smaller than $i$, if $t \geq i$ then the color is precisely $i$ as just shown or from the inductive hypothesis. Therefore the color $i + 1$ is available at the root $r$ and the overall cost of the change is $(k - i) - (k - i - 1) = 1$.

This completes the construction of a cheapest coloring $c'$ of $T^1_k$ which does not use the color $i$ at the vertices of $V_i$. It is easy to see that $c'$ has the required properties and that it is a cheapest $(i+1)$-coloring of $T^1_k$. It is obviously a $(i+1)$-coloring. From the inductive hypothesis, each $T^1_t$ for $t \geq i + 2$ has a cheapest $(i+1)$-coloring under $c'$. To extend that coloring to the whole $T^1_k$ we need to change the color of $r$ and that costs at least 1. But it was just shown that $c'$ changes the
color of $r$ either to $i$ or to $i + 1$ at the cost of 1. We now can compute the cost of
the coloring $c'$.

$$\sum c'(T^1_k) = \Sigma_{i+1}(T^1_k) = \Sigma(T^1_k) + 1 + \sum_{t=i+2}^{k-1} (k-t+1)T^1_{t-i-1}.$$  

We can change the index of summation on the sum by setting $j = t - i - 1$, to obtain:

$$\sum c'(T^1_k) = \Sigma_{i+1}(T^1_k) = \Sigma(T^1_k) + 1 + \sum_{j=1}^{k-i-2} (k-i-j)T^1_{j}.$$  

$$= \Sigma(T^1_k) + 1T^1_{k-i-1}. \quad \Box$$

Let us denote by $\delta_i$ the difference between $\Sigma_{i}(T^1_k)$ and $\Sigma_{i+1}(T^1_k)$. Since

$$\delta_i = (\Sigma_{i}(T^1_k) - \Sigma(T^1_k)) - (\Sigma_{i+1}(T^1_k) - \Sigma(T^1_k))$$

and $|T^1_j| > 3 |T^1_{j-1}|$, from the previous lemma, we find that $\delta_i > \delta_{i+1}$.

Now we can describe the construction of the graph $G_k$ mentioned earlier. For
concreteness, Figure 3.4 shows $G_5$. In general to construct $G_k$ we start with the
tree $T^1_k$ and its best coloring $c$. Let $A$, $V_i$, and $\delta_i$ be defined as before. We add
$\delta_2$ independent vertices $w_1, w_2, ..., w_{\delta_2}$. Each vertex $w_j$ is now joined to all the
vertices in $V_{i-1}$ if and only if $j \leq \delta_i$. As we have mentioned, $\delta_{k-1} < ... < \delta_i < ... < \delta_2$
and therefore the neighborhood of the vertex $w_j$ is a subset of the neighborhood of
$w_{j-1}$. This implies that any color assignment on $w_j$’s should be nonincreasing. In
fact, for $j \neq \delta_i$, $w_j$ and $w_{j-1}$ must receive the same color since they have identical
neighborhoods, whereas for $j = \delta_i$ we see that $c(w_{j-1}) \geq c(w_j)$.
Figure 3.4 To complete the graph shown, edges from \( w_1 \) through \( w_7 \) must be added, namely \( w_1 \) is adjacent to all vertices indicated by \( \bullet \), \( \circ \), and by \( \bullet \); \( w_2 \) is adjacent to all vertices indicated by \( \bullet \) and \( \circ \), and finally \( w_3 \) through \( w_7 \) are adjacent to all black vertices indicated by \( \bullet \).
In this example, \( \Sigma_3(T_5^1) = 164 \), \( \Sigma_4(T_5^1) = 165 \), \( \Sigma_3(T_5^1) = 167 \), and \( \Sigma_2(T_5^1) = 174 \). Therefore \( \delta_4 = 1, \delta_3 = 2, \delta_2 = 7 \).
Theorem 3.5 For every natural number $k > 2$, there exists a graph $G_k$ such that for every $i$, $2 \leq i \leq k$, the chromatic sum of the graph $G_k$ can be achieved using exactly $i$ different colors, i.e.

$$\Sigma(G_k) = \Sigma_2(G_k) = \ldots = \Sigma_i(G_k) = \ldots = \Sigma_{k-1}(G_k) = \Sigma_k(G_k).$$

Proof. We will show that the graph $G_k$ obtained by the described construction satisfies the theorem. Consider first the best coloring of $T_{k}^1$, which is the coloring $c$. It can be extended to the vertices $w_i$ in the following unique way:

$$c(w_1) = k - 1 \quad \text{and} \quad c(w_j) = i \quad \text{for} \quad \delta_{i+1} < j \leq \delta_i \quad \text{and} \quad 2 \leq i \leq k - 2.$$ 

Coloring $c$ is a candidate for the chromatic sum of $G_k$. We shall proceed to show that coloring $c$ cannot be beaten, but it can be tied using $i$ colors with $2 \leq i \leq k$.

Denote by $S$ the cost of the coloring $c$ restricted to the $w_i$'s, i.e. $S = \sum_{i=2}^{k-1} c(w_i)$. Since we can not reduce coloring $c$ on $T_{k}^1$ our search for other candidate best colorings of $G_k$ must concentrate on colorings cheaper than $c$ on the vertices $w_i$. Let $c_x$ be such a coloring and let $j$ be the biggest index for which $c_x(w_j) < c(w_j)$. Then $j = \delta_i$ for some $i$ and $c_x(w_j) = i - 1$ for if $c_x(w_j) < i - 1$, we could also reduce $c_x(w_{j+1})$. Since $w_{\delta_i}$ is adjacent to the vertices from $V_{i-1}$ their color has to be changed and, from the previous lemma, a cheapest $i$-coloring is imposed on $T_{k}^1$. Moreover the color $i-1$ isn't present at any vertex in the partite set $A$. Thus the color $i - 1$ is available for all $w_r$'s with $r \leq \delta_i$, i.e. $c_x(w_r) \leq i - 1$. Since $c_x$ has to be nonincreasing on the sequence of $w_i$'s, $c_x(w_r) = i - 1$ for $r \leq \delta_i$. As a result, we have only $k - 1$ possible candidates for best colorings of $T_{k}^1$. Let's call those
colorings \( c_k, \ldots, c_2 \). The coloring \( c_k \) corresponds to the coloring \( c \) and is a cheapest \( k \)-coloring of \( G_k \). For that coloring we have

\[
\Sigma_k(G_k) = \sum_{v \in G_k} c_k(v) = \Sigma(T_k^1) + S.
\]

For \( i < k \), \( c_i(w_r) = i - 1 \) for \( 1 \leq r \leq \delta_i \) and \( c_i(w_r) = c(w_r) \) for \( r > \delta_i \). This assignment forces a cheapest \( i \)-coloring on \( T_k^1 \). Thus we have

\[
\sum_{v \in T_k^1} c_i(v) = \Sigma_i(T_k^1) = \Sigma(T_k^1) - \Sigma(T_k^1) = \Sigma(T_k^1) + \sum_{r=1}^{k-1} \delta_r.
\]

However, the number of vertices \( w_j \) has been carefully selected to compensate precisely for this added cost, namely

\[
\sum_{r=1}^{\delta_2} c_i(w_r) = S - \sum_{r=1}^{k-1} \delta_r.
\]

Consequently \( \sum_{v \in G_k} c_i(v) = \Sigma_k(G_k) \). The coloring \( c_i \) is an \( i \)-coloring and since only \( c_i \)'s can be best colorings of \( T_k^1 \) we have \( \sum_{v \in G_k} c_i(v) = \Sigma_i(G_k) \) and so

\[
\Sigma(G_k) = \Sigma_2(G_k) = \ldots = \Sigma_i(G_k) = \ldots = \Sigma_k(G_k)
\]

as stated in the theorem. \( \square \)

In the Theorem 3.2 we prove that if the chromatic sum of a tree is equal to its \( k_1 \)-chromatic sum and also \( k_2 \)-chromatic sum then \( k_1 \) and \( k_2 \) must be consecutive. While the preceding theorem shows that for graphs we can have a long interval of integers \( i, 2 \leq i \leq k \), where the chromatic sum is attained, it is also possible to attain the chromatic sum at \( 2 \) and \( k \) without attaining the minimum at
intermediate values. A slight modification of the construction used in the proof of Theorem 3.4 verifies this.

**Theorem 3.6** For every natural number $k > 2$ there exists a graph $F_k$ such that $\Sigma(F_k') = \Sigma_2(F_k) = \Sigma_k(F_k)$ and $\Sigma_i(F_k) > \Sigma(F_k)$ for $i$ different from 2 and $k$.

**Proof.** To construct a graph $F_k$ we start again with the rooted tree $T_k$. Let $\Delta$ denote the following difference $\Delta = \Sigma_2(T_k) - \Sigma(T_k^1)$. We add $\Delta$ independent vertices $w_1, \ldots, w_\Delta$ and join them to every vertex from the set $V_1$. Now there are only two possibilities to color the vertices $w_i$: either color all of them with the color 1 (calling it the coloring $c_1$) or color all of them with the color 2 (calling it the coloring $c_2$). Any different coloring would be more expensive for the whole graph $F_k$ since the coloring $c_2$ allows for the best coloring over the tree $T_k^1$ and coloring one vertex $w_i$ with the color 1 forces that color over all $w_i$'s. The coloring $c_1$ imposes a cheapest 2–coloring on $T_k^1$, as proved in the preceding lemma. Since the savings over $w_i$'s match the loses over the tree $T_k^1$ ($\Delta = \Sigma_2(T_k^1) - \Sigma(T_k^1)$), we have $\Sigma_2 = \Sigma_k$. Any $i$–coloring of $T_k^1$, for $i \neq 2$, does not allow to use the color 1 over $w_i$'s and still is more expensive over $T_k^1$ and therefore $\Sigma_i > \Sigma_2 = \Sigma_k$. $\Box$

To be more precise, the construction described in the previous proof provides us with a family of examples for which the $i$–chromatic sums behave in the following way: $\Sigma(F_k) = \Sigma_2(F_k) < \Sigma_3(F_k) > \Sigma_4(F_k) > \ldots > \Sigma_{k-1}(F_k) > \Sigma_k(F_k) = \Sigma(F_k)$.

In fact, by manipulating the number of the vertices $w_i$ added to $T_k^1$, we can produce graphs for which the sequence of their $i$–chromatic sums is *unconstrained* according to the definition introduced in [1]. Specifically, for every permutation $\pi$ of numbers 2 through $k$ we can find a graph $G$, such that
Moreover we can force a strict inequality or equality at each place in the sequence. We illustrate this freedom with an example.

**Example 3.7** We will construct a graph \( H \) with the chromatic sequence fulfilling the following string of inequalities:

\[ \Sigma_2 = \Sigma_8 < \Sigma_7 < \Sigma_3 = \Sigma_6 < \Sigma_4 < \Sigma_5. \]

We start with the tree \( T^1_8 \). The sets \( V_1, V_2, \) and \( V_3 \) are defined as in the previous construction. For the tree \( T^1_8 \) the differences between its \( k \)-chromatic sums are as follows:

\[
\begin{align*}
\Sigma_2 - \Sigma_8 &= 396 \\
\Sigma_3 - \Sigma_6 &= 113 \\
\Sigma_4 - \Sigma_5 &= 24 \\
\Sigma_7 - \Sigma_8 &= 1 \\
\Sigma_6 - \Sigma_8 &= 3 \\
\Sigma_5 - \Sigma_8 &= 10 \\
\Sigma_4 - \Sigma_8 &= 34 \\
\Sigma_3 - \Sigma_8 &= 116.
\end{align*}
\]

To complete the graph \( H \) we add 283 vertices \( w_1, \ldots, w_{283} \) and join them to the vertices of the tree as specified below:

- \( w_1 \) through \( w_{283} \) are joined to all vertices from the set \( V_1 \),
- \( w_1 \) through \( w_{88} \) are joined to all vertices from the set \( V_2 \),
- and finally \( w_1 \) through \( w_{25} \) are joined to all vertices from the set \( V_3 \).

Then we have the following \( k \)-chromatic sums for the graph \( H \):

\[
\Sigma_8(H) = \Sigma_8(T^1_8) + 4 \cdot 25 + 3 (88 - 25) + 2(283 - 88) = \Sigma(T^1_8) + 679 = \Sigma(H).
\]
\[ \Sigma_2(H) = \Sigma_2(T_{g}^1) + 1 \cdot 283 = \Sigma(T_{g}^1) + 396 + 283 = \Sigma(T_{g}^1) + 679 = \Sigma(H). \]

\[ \Sigma_7(H) = \Sigma_7(T_{g}^1) + 679 = \Sigma(T_{g}^1) + 1 + 679 = \Sigma(H) + 1. \]

\[ \Sigma_6(H) = \Sigma_6(T_{g}^1) + 679 = \Sigma(T_{g}^1) + 3 + 679 = \Sigma(H) + 3. \]

\[ \Sigma_3(H) = \Sigma_3(T_{g}^1) + 2 \cdot 283 = \Sigma(T_{g}^1) + 116 + 566 = \Sigma(T_{g}^1) + 682 = \Sigma(H) + 3. \]

\[ \Sigma_4(H) = \Sigma_4(T_{g}^1) + 3 \cdot 88 + 2(283 - 88) = \Sigma(T_{g}^1) + 34 + 654 = \Sigma(H) + 9. \]

\[ \Sigma_5(H) = \Sigma_5(T_{g}^1) + 679 = \Sigma(T_{g}^1) + 10 + 679 = \Sigma(H) + 10. \]

**Remark 3.8** In all the constructions used in Section 3.3 we might replace the rooted tree \( T_{k}^1 \) with the tree \( T_k \) which is the smallest tree that requires \( k \) colors to achieve its chromatic sum. We would then obtain graphs of smaller orders with the same desired properties as was done in the Example 3.1, however, the proofs would be more involved.
CHAPTER IV

THE WEIGHTED CHROMATIC SUM

4.1 Introduction

We defined the chromatic sum of a graph $G$, $\Sigma(G)$, to be the smallest possible sum of colors taken over all proper colorings of $G$. A natural generalization of this concept is to consider a weight function $\omega$ associated with the colors and to define the weighted chromatic sum of a graph $G$, $\omega \Sigma(G)$, as the smallest possible sum of weights taken over all possible proper colorings of $G$. Since the colors can be easily rearranged we may assume that the weight function $\omega$ is nondecreasing.

More precisely, let $\omega : \mathbb{N} \rightarrow \mathbb{R}^+_0$ be a nondecreasing function. Then the weighted chromatic sum, $\omega \Sigma(G)$, is defined as follows:

$$\omega \Sigma(G) = \min \{ \sum_{v \in V} \omega(c(v)) : c \text{ is a proper coloring of } G \}.$$  

The chromatic number has many applications. One of those is the scheduling problem where the different classes have to be assigned time periods. Each class can be represented by a vertex and two vertices are joined with an edge whenever the corresponding classes are conducted by the same instructor or have some students enrolled in both. Thus assigning time periods to classes may be viewed as a proper coloring with colors representing times, since every two adjacent vertices (classes conducted by the same instructor) have to receive different colors to avoid conflicts. By finding the optimal schedule we usually mean finding one with the smallest number of different time periods or, in other words, finding a proper coloring of the
associated graph which uses the chromatic number of colors. But what we really want to minimize is cost. Different time periods might have different corresponding costs. For example, morning hours might be less expensive than afternoon hours (perhaps due to heating and cooling expenses) and much less expensive than evening ones (overtime for janitors and security personnel). The cost of each time period can be represented by the weight of its corresponding color and overall cost will be then the sum of the weights. Thus, when optimizing the schedule, the weighted chromatic sum concept seems to be more appropriate than the chromatic number concept.

When the weight \( \omega \) is the identity function \( \omega(i) = i \) then the weighted chromatic sum is the usual chromatic sum of \( G \), i.e. \( \omega \Sigma(G) = \Sigma(G) \). On the other hand, when the weights of colors grow exponentially, we can show that the weighted chromatic sum problem becomes nearly equivalent to the chromatic number problem in the sense that the minimum sum can only be achieved by using the chromatic number of colors. Of course, not every coloring using the chromatic number of colors yields the weighted chromatic sum.

**Theorem 4.1** Let \( G \) be a graph of order \( p \) and let the weight function \( \omega \) be defined as follows; \( \omega(i) = p^{i-1} \). Then the following equivalence is true:

\[
p^{k-1} < \omega \Sigma(G) \leq p^k \iff \chi(G) = k.
\]

**Proof.** Consider a proper \( k \) - coloring \( c \) of a graph \( G \) of order \( p \). Then we obtain an upper bound \( \sum_{v \in V} \omega(c(v)) \leq p \cdot p^{k-1} = p^k \). To find a lower bound observe that the color \( k \) has to be used somewhere and thus \( \sum_{v \in V} \omega(c(v)) \geq p^{k-1} + (p-1) > p^{k-1} \).
Summarizing we have \( p^{k-1} < \sum_{v \in V} \omega(c(v)) \leq p^k \) \( \Leftrightarrow \) \( c \) is a \( k \)-coloring of \( G \),

which implies the statement of the theorem. □

In fact we can obtain a similar equivalence between the weighted chromatic sum and the chromatic number of a graph using a considerably smaller weight function.

**Remark 4.2** Let \( G \) be a graph of order \( p \) and let the weight function \( \omega \) be defined as follows: \( \omega(i) = \left( \frac{p}{2} \right)^{i-1} \). Then the weighted chromatic sum determines the chromatic number of a graph.

**Proof.** Using a little more careful analysis than in the proof of the Theorem 4.1, we can compute lower and upper bounds for the smallest sum of weights when exactly \( k \) colors are used. Since every color 1 through \( k \) has to be used at least once and the remaining \( p - k \) vertices have to be colored with at least 1, the lower bound is:

\[
L_k = (p-k) + 1 + \frac{p}{2} + \left( \frac{p}{2} \right)^2 + \ldots + \left( \frac{p}{2} \right)^{k-1}.
\]

The largest sum of weights is when all the colors are equally distributed and therefore the upper bound is:

\[
U_k = \frac{p}{k} \left[ 1 + \frac{p}{2} + \left( \frac{p}{2} \right)^2 + \ldots + \left( \frac{p}{2} \right)^{k-1} \right].
\]

It is easy to show that for \( k \geq 3 \), \( L_k > U_{k-1} \). □

4.2 Extra Colors Needed to Obtain the Weighted Chromatic Sum

Similarly as for the chromatic sum we can ask the following question:
for given natural numbers $k$ and $t$ and a nondecreasing weight function $\omega$, can we find a $k$-chromatic graph $G$ which requires $k + t$ colors to attain its weighted chromatic sum?

As in our previous work we start this discussion with trees. We consider a weight function to be nondecreasing, but what happens if for some natural number $k$, $\omega(k) = \omega(k - 1)$? We claim that for a weight function $\omega$ with that property there is no tree which would need $k + 1$ or more colors to attain its weighted chromatic sum. For assume that $T$ requires at least $k + 1$ colors in every one of its best colorings. Let $c$ be a best coloring of $T$ in which color $k + 1$ appears the minimum possible number of times, and select $v$ with $c(v) = k + 1$. Denote by $u_1, ..., u_r$ all the neighbors of $v$ colored with the color $k$. Let the set $U$ be the union of the $r$ branches of $T$ whose roots are $u_1, ..., u_r$. Let's now interchange colors $k$ and $k - 1$ in $U$. Since $\omega(k) = \omega(k - 1)$ the overall cost doesn't change but now the color $k$ is available to use at the vertex $v$. Consequently we obtain either a cheaper coloring of $T$ or one of equal total weight but using fewer vertices of color $k + 1$, a contradiction.

Thus, from now on, we assume that $\omega: N \to \mathbb{R}_0^+$ is a strictly increasing weight function. Denote by $\Delta(i,j)$ the difference between the weights, $\Delta(i,j) = \omega(i) - \omega(j)$. For this weight function and for every positive real value $m$ we recursively define the family of rooted trees $\{W^m_k\}_{k \in N}$.

$W^m_{1}^{\Delta(2,1)}$ is a rooted tree with one vertex.

$W^m_k$ is a rooted tree with the root $r$ such that

$$W^m_k - r = \bigcup_{i=1}^{k-1} \left\lceil \frac{m + \Delta(k,i)}{\Delta(i+1,i)} \right\rceil W^\Delta_{i}^{i+1,i}. $$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Imitating the proof of the Lemma 1.2 we obtain the following result:

**Lemma 4.3** The tree $W_k^m$ is the smallest rooted tree in which the color $k$ is forced to appear at the root in order to achieve the weighted chromatic sum, and also changing that color to any smaller one costs at least $m$.

Now let $W_{k,\varepsilon}$ denote a tree obtained from two copies of $W_{k-1}^{\Delta(k,k-1)+\varepsilon}$ by joining their roots with an edge as indicated on Figure 4.1.

![Figure 4.1](image-url)

Again following the proof of the Theorem 1.3 we can justify our next statement.

**Theorem 4.4** The smallest tree which requires $k$ colors to achieve its weighted chromatic sum is a tree $W_{k,\varepsilon}$ for some small nonnegative value of $\varepsilon$.

The value of $\varepsilon$ has to be only big enough to make the cost of changing, at both $u$ and $v$, the color $k-1$ to any color $i < k-1$, strictly bigger than $\Delta(k,k-1)$. For some weight functions the value of $\varepsilon$ may be as small as 0.

The growth rate for the family of $W_{k,\varepsilon}$ trees is exponential with respect to $k$, analogous to the growth rate of the family of $T_k$ trees from Chapter I.

**Theorem 4.5** For any strictly increasing weight function $\omega$ and any nonnegative $\varepsilon$ the following inequality holds:

$$|W_{k,\varepsilon}| \geq 2 \cdot 3^{k-2}.$$
Proof. Let's define recursively a family of rooted trees:

$S_1$ is a rooted tree with one vertex.

$S_k$ is a rooted tree with the root $r$ such that

$$S_k - r = \bigcup_{i=1}^{k-1} 2S_i.$$ 

$|S_0| = 1 = 3^0$ and it is easy to show that in general $|S_i| = 3^{i-1}$. For any increasing weight function we have $\left\lceil \frac{m+\Delta(k,i)}{\Delta(i+1,i)} \right\rceil \geq \left\lceil \frac{m+\Delta(i+1,i)}{\Delta(i+1,i)} \right\rceil \geq \left\lceil \frac{m}{\Delta(i+1,i)} + 1 \right\rceil \geq 2$. Therefore $|W_{k-1}^{\Delta(k,k-1)+\epsilon}| \geq |S_k|$ and consequently

$$|W_{k-1}^{\Delta(k,k-1)+\epsilon}| \geq 2 |S_{k-1}| = 2 \cdot 3^{k-2}. \quad \square$$

To obtain a $k$-chromatic graph which has to use $k + t$ colors to attain its weighted chromatic sum recall Construction A from Chapter II. Instead of the trees $T_i^1$, $k+1 \leq i \leq k + t$, we use the trees $W_i^1$, $k+1 \leq i \leq k + t$, and the reasoning leading to justify the statement of our next theorem is identical to the proof of the Theorem 2.1.

**Theorem 4.6** For every strictly increasing weight function $w$ and for any natural numbers $k$ and $t$, there exists a $k$-chromatic graph which requires $k + t$ colors to attain its weighted chromatic sum.
CHAPTER V

CONSTANT TIME ALGORITHMS FOR TREES

5.1 Introduction

Soon after introducing the definition of the chromatic sum the smallest tree that requires three colors to attain its chromatic sum was found. Then a natural question arose: among all the trees of a specific order $N$, what is the percentage of those trees which use exactly two, exactly three, or possibly more colors in their best colorings? As soon as the linear algorithm to compute the chromatic sum was discovered it became apparent that a computer approach could be used to answer the above question for families of trees of small orders. But now we face the problem of inputting the data: should we input the structure of each tree one by one and then apply that quick algorithm to find a best coloring? However, since the number of all trees of order $N$ grows exponentially in $N$, this approach doesn't seem to be practical for families of trees of order larger than ten. Thus an efficient algorithm is needed to generate all trees of a given order.

T. Beyer and S. M. Hedetniemi in [2] have presented a very fast algorithm to generate all rooted trees of any fixed order. Their algorithm was extended by R. Wright, B. Richmond, A. Odlyzko and B. McKay in [16] to generate all free trees also of a fixed order. Using this algorithm together with the one presented in Chapter I to find a best coloring, we are able to examine the families of all trees of orders 6 through 26. But we soon realize that the same approach might be used to investigate all trees of a given order checking for any other property, as long as checking for that
property in a single tree can be done by an algorithm working in a "bottom-up" fashion analogous to the algorithm from Chapter I.

In Section 5.2 we review the results of the two papers mentioned [2] and [16]. Then, in Section 5.3, we describe the general idea of how to apply these generation algorithms to different problems and in sections 5.4, 5.5, and 5.6 we present three specific applications. Finally, in Section 5.7, we discuss the complexity issues.

5.2 Generating Rooted and Free Trees

In this section we describe the results of two papers: "Constant Time Generation of Rooted Trees" by T. Beyer and S.M. Hedetniemi and "Constant Time generation of Free Trees" by R. Wright, B. Richmond, A. Odlyzko and B. McKay.

We first concentrate on rooted trees. The level of a vertex \( v \), \( a(v) \), in a rooted tree \( T \) is one more than the distance from the vertex to the root. The level of the root is one. The level sequence of a rooted tree \( T \), \( L(T) \), is the sequence of levels of all its vertices given in preorder traversal. Thus every tree is uniquely determined by its level sequence. The level sequence however is not uniquely determined by its rooted tree as it is illustrated on the Figure 5.1.

\[
L(T) = (123323445532) \quad L(T^*) = (123455432332)
\]

![Figure 5.1](image.png)

Figure 5.1 Two isomorphic rooted trees with different level sequences.
The ambiguity comes from the fact that reordering the branches of a tree affects its level sequence. Thus, to get a unique representation we shall select a **canonical ordering** of all branches of a tree under consideration. For any two neighboring branches, we arrange them so that the first one has its level sequence no smaller than the second. In other words, for a rooted tree \( T \) the lexicographically largest level sequence is selected. It is called the **canonical level sequence** of \( T \). In [2] the authors present the algorithm to generate canonical level sequences for all rooted trees of a given order.

Let \( L(T) = [a_1, a_2, ..., a_q, ..., a_p, ..., a_j] \) denote the canonical level sequence of \( T \) of order \( n \) and let the numbers 1 through \( n \) denote the vertices of \( T \). Next, let \( p \) denote the largest integer (vertex) with the level greater than 2, and let \( q \) be its parent, or equivalently the largest integer such that \( q < p \) and \( a_q = a_p - 1 \). The algorithm uses a successor function to generate all rooted trees as follows:

1. Start with the level sequence for the path \( P_n \) (lexicographically the largest)
   \[ L[P_n] = [1 2 3 4 ... n]. \]

2. To a current canonical level sequence apply the successor function \( S \) obtaining the lexicographically next canonical level sequence.

3. Repeat step 2 until the smallest level sequence is obtained (for the star), i.e.
   \[ L[K(1,n - 1)] = [1 2 2 2 ... 2]. \]

The successor of \( L \), \( S(L) = (s_1, ..., s_n) \), is given by:

\[
s_i = \begin{cases} 
a_i & \text{for } 1 \leq i < p \\
 a_{i-p+q} & \text{for } p \leq i \leq n 
\end{cases}
\]

or equivalently:

\[
L[T] = [a_1 a_2 a_3 ... a_q ... a_p 2 2 2 ... 2] \\
S(L[T]) = [a_1 a_2 a_3 ... a_q ... a_p a_q ... a_p a_q ... ].
\]
It is proved in [2] that $S$ transforms any canonical level sequence other than $[2 \ 2 \ldots \ 2]$ into the next canonical level sequence in decreasing lexicographical order.

We now present a couple of examples by applying the successor function $S$.

**Example 5.1** Here the levels at positions $p$ and $q$ are bold and underlined.

\[
\begin{align*}
[1 & 2 3 4 3 2 2 2 2] \\
[1 & 2 3 4 2 2 2 2 2] \\
[1 & 2 3 4 2 3 4 2 3] \\
[1 & 2 3 4 2 3 4 2 3] \\
[1 & 2 3 4 2 3 4 2 2] \\
[1 & 2 3 4 2 3 4 2 2]
\end{align*}
\]

From the description of the function $S$ we see that the number of steps the algorithm has to perform is given by the following formula:

\[
\text{# of steps} = \sum_{\text{all trees}} (n - p + 1).
\]

Consequently the average number of steps per single tree is given by:

\[
\text{# of steps per tree} = \frac{\sum_{\text{all trees}} (n - p + 1)}{\text{# of trees}}.
\]

It is shown in [2], by examining the code of the algorithm, that the average number of steps per tree is not greater than 2, regardless of the order of trees. However using generating functions it is possible to give a more precise estimate of the complexity.

Let $T(x)$ be the generating function for rooted trees, that is $T(x) = \sum_{n=0}^{\infty} T_n x^n$, where $T_n$ is the number of rooted trees of order $n$. Then $x^2T(x)$ is the generating
function for rooted trees with at least \( i \) end-vertices on level 2 and therefore the expression \( x^{i-1}T(x) - x^iT(x) \) represents the generating function for rooted trees with exactly \( i-1 \) end-vertices on level 2. Since to produce the next canonical level sequence only these vertices with the level equal to 2 and the very next vertex at the position \( p \) have to change their levels, the algorithm needs exactly \( i \) steps to process a tree with \( i-1 \) vertices on level 2.

Let \( G(x) = \sum_{n=0}^{\infty} g_n x^n \) be the generating function describing the complexity of the algorithm, that is \( g_n \) corresponds to the number of steps the algorithm has to perform in order to generate all rooted trees of order \( n \). Then we have:

\[
G(x) = \left( T(x) - xT(x) \right) + 2\left( xT(x) - x^2T(x) \right) + 3\left( x^2T(x) - x^3T(x) \right) + ... \\
= T(x) + xT(x) + x^2T(x) + x^3T(x) + ... \\
= T(x)\left[ 1 + x + x^2 + x^3 + ... \right] = T(x) \frac{1}{1 - x}.
\]

To compute the asymptotic value of \( g_n \) we will use an approach similar to the one in Robinson and Schwenk [12].

Recall first the formula for the asymptotic value of \( T_n \), developed by Otter [10] and then discussed in detail in Harary and Palmer [7]:

\[
T_n \sim \frac{b \rho^{1/2}}{2\pi^{1/2}} n^{-3/2} \rho^{-n}, \text{ where } \rho \text{ is the radius of convergence of } T(x).
\]

Otter calculated that \( \rho = 0.3383219 \). He also showed that near \( x = \rho \), \( T(x) \) has an expansion of the form:

\[
T(x) = 1 - b(\rho - x)^{1/2} + c(\rho - x) + ...
\]
The asymptotic value of $T_n$ follows from the above formula and the lemma of Polya, which we shall now present.

**Lemma (Polya [11], p. 240).** Let the power series

$$f(x) = a_0 + a_1 x + a_2 x^2 + ...$$

have the finite radius of convergence $\alpha > 0$, with $x = \alpha$ the only singularity on its circle of convergence. Suppose also that $f(x)$ can be expanded near $x = \alpha$ in the form

$$f(x) = \left(1 - \frac{x}{\alpha}\right)^s g(x) + \left(1 - \frac{x}{\alpha}\right)^t h(x),$$

where $g(x)$ and $h(x)$ are analytic at $x = \alpha$, $g(\alpha) \neq 0$, $s$ and $t$ are real, $s \neq 0, -1, -2, ...$, and either $t < s$ or $t = 0$. Then

$$a_n \sim \frac{g(\alpha)}{\Gamma(s)}n^{s-1}\alpha^n.$$

The above lemma provides also the asymptotic for $g_n$. To see this, let $G(x)$ be the function $f(x)$, $s = -\frac{1}{2}$, $\alpha = \rho$, and $g(x) = \frac{-\rho^{1/2}}{1 - x}$. Then we have:

$$f(x) = T(x) \frac{1}{1 - x} = (1 - b(\rho - x)^{1/2} + c(\rho - x) + ...) \frac{1}{1 - x}$$

$$= (1 - b\rho^{1/2}(1 - \frac{x}{\rho})^{1/2} + c(\rho - x) + ...) \frac{1}{1 - x}$$

$$= (1 - \frac{x}{\rho})^{1/2} \left(\frac{-\rho^{1/2}}{1 - x}\right) + (1 - x)^{-1} + c(\rho - x) \frac{1}{1 - x} + ...$$

Therefore $g_n \sim \frac{g(\rho)}{\Gamma(-1/2)} n^{-3/2} \rho^{-n}$. Since $\Gamma(-1/2) = -2\pi^{1/2}$, we obtain

$$g_n \sim \frac{b\rho^{1/2}}{2\pi^{1/2}} n^{-3/2} \rho^{-n} \frac{1}{1 - \rho}.$$
Thus the average number of steps per tree for the Beyer–Hedetniemi algorithm is asymptotic to \( \frac{1}{1 - \rho} \). Thus we have replaced the upper bound of 2 in [2] with the exact limiting value.

In the Figure 5.2 we present some experimental data obtained by running a corresponding Pascal program on a VAX 8700. The complete listing of the code is contained in the Appendix 1.

\[
g_n \sim T_n \frac{1}{1 - \rho}.
\]

<table>
<thead>
<tr>
<th>order n</th>
<th># of rooted trees</th>
<th># of steps/tree</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>20</td>
<td>1.85</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>48</td>
<td>1.77</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>115</td>
<td>1.74</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>286</td>
<td>1.70</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>719</td>
<td>1.68</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1,842</td>
<td>1.65</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4,766</td>
<td>1.64</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>12,486</td>
<td>1.63</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>32,973</td>
<td>1.62</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>87,811</td>
<td>1.61</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>235,381</td>
<td>1.60</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>634,847</td>
<td>1.59</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1,721,159</td>
<td>1.59</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>4,688,676</td>
<td>1.58</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>12,826,228</td>
<td>1.58</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>35,221,832</td>
<td>1.57</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>97,055,181</td>
<td>1.57</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.2

R. Wright, B. Richmond, A. Odlyzko and B. McKay in [16] extended the algorithm of Beyer and Hedetniemi to generate level sequences of all free trees of a given order. In other words, their algorithm produces level sequences of only one member of each equivalence class of rooted trees under the isomorphism of the underlying free tree. The extension is performed by placing some requirements on the
root vertex. Let \( w(T) \) denote the order (weight) of \( T \). The algorithm produces canonical level sequences of all rooted trees, such that:

1. \( T \) is rooted at a central vertex.

2. If \( T \) is bicentral with central vertices \( z_1 \) and \( z_2 \), as described on Figure 5.3, then \( z_2 \) is the root if:
   a) \( w(T_1) < w(T_2) \) or
   b) \( w(T_1) = w(T_2) \) and \( L(T_1) \leq L(T_2) \).

For example, of the five isomorphic rooted trees in Figure 5.4 only the level sequence of the framed one is generated, since its root satisfies both the requirements.
Despite the fact that this algorithm is much more involved, the authors were still able to prove that the average number of steps per tree is constant. Although at this time we don't have any precise bound or asymptotic for the average number of steps per tree, the experimental data presented in Figure 5.5 suggests that this value is very close, if not equal, to the corresponding number for rooted trees. The complete listing of the Pascal code of the program is contained in Appendix 2.

<table>
<thead>
<tr>
<th>order n</th>
<th># of free trees</th>
<th># of steps/tree</th>
<th>CPU time h : min : sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>106</td>
<td>2.08</td>
<td>0:00:00.01</td>
</tr>
<tr>
<td>11</td>
<td>235</td>
<td>2.01</td>
<td>0:00:00.01</td>
</tr>
<tr>
<td>12</td>
<td>551</td>
<td>1.92</td>
<td>0:00:00.02</td>
</tr>
<tr>
<td>13</td>
<td>1301</td>
<td>1.89</td>
<td>0:00:00.05</td>
</tr>
<tr>
<td>14</td>
<td>3159</td>
<td>1.83</td>
<td>0:00:00.11</td>
</tr>
<tr>
<td>15</td>
<td>7741</td>
<td>1.82</td>
<td>0:00:00.27</td>
</tr>
<tr>
<td>16</td>
<td>19320</td>
<td>1.77</td>
<td>0:00:00.66</td>
</tr>
<tr>
<td>17</td>
<td>48629</td>
<td>1.77</td>
<td>0:00:01.60</td>
</tr>
<tr>
<td>18</td>
<td>123867</td>
<td>1.73</td>
<td>0:00:04.07</td>
</tr>
<tr>
<td>19</td>
<td>317955</td>
<td>1.73</td>
<td>0:00:10.19</td>
</tr>
<tr>
<td>20</td>
<td>823065</td>
<td>1.71</td>
<td>0:00:25.86</td>
</tr>
<tr>
<td>21</td>
<td>2144505</td>
<td>1.70</td>
<td>0:01:06.29</td>
</tr>
<tr>
<td>22</td>
<td>5623756</td>
<td>1.68</td>
<td>0:02:52.82</td>
</tr>
<tr>
<td>23</td>
<td>14828074</td>
<td>1.68</td>
<td>0:07:34.53</td>
</tr>
<tr>
<td>24</td>
<td>39299897</td>
<td>1.67</td>
<td>0:20:04.53</td>
</tr>
<tr>
<td>25</td>
<td>104636890</td>
<td>1.66</td>
<td>0:53:18.21</td>
</tr>
<tr>
<td>26</td>
<td>279793450</td>
<td>1.65</td>
<td>2:21:26.67</td>
</tr>
</tbody>
</table>

Figure 5.5

5.3 Applications

Imagine that our goal is to compute certain values VAL1, VAL2, ..., VALk for every tree of a given order. Denote by A the algorithm to compute those values for a single tree T and assume that the algorithm A operates in a "bottom-up" fashion. This means that A establishes the values VAL1, ..., VALk for all end-vertices first and then proceeds upward as described below. Let r be a vertex of T with children
a_1, a_2, ..., a_s. Assume that the values sought are already known for all the subtrees with roots at a_i, 1 \leq i \leq s. To obtain the values VAL_1(r), ..., VAL_k(r) corresponding to the subtree rooted at r, the algorithm A applies a certain function f to compute these values from the already known values VAL_1(a_1), ..., VAL_k(a_1), VAL_1(a_2), ..., VAL_k(a_s) at the children as indicated in the Figure 5.6.

\[
[\text{VAL}_1(r), ..., \text{VAL}_k(r)] = f(\text{VAL}_1(a_1), ..., \text{VAL}_k(a_s))
\]

Figure 5.6 The description of the algorithm A

Consider now the tree generating algorithm G which is an extension of the free trees generating algorithm described in the previous section. Trees are represented not only by their level sequences but also by their structures. Every vertex I of a tree T is viewed as the root of the tree T(I) formed by all the vertices "below" I, or more precisely, T(I) is induced by all the vertices v of T such that I belongs to the unique path from v to the root of T. At each vertex I the information about the structure is stored in the record TREE[I], as described below:

\[
\text{TREE}[I] = \text{RECORD} \\
\quad \text{NUMBER OF CHILDREN} \\
\quad \text{FIRST SON} \\
\quad \text{RIGHT BROTHER} \\
\quad \text{PARENT} \\
\quad \text{LEVEL} \\
\text{END}
\]
Although the algorithm $G$ is more tedious than the corresponding one from Section 5.2, the main ideas are the same and so is the order of complexity. Analogously we can produce the algorithm generating structures of all rooted trees.

Now we can combine the algorithms $A$ and $G$ to produce the values $VAL_1, \ldots, VAL_k$ for all free trees of a given order $n$. First the structure of each record $TREE[I]$ has to be changed to contain $k$ additional fields in order to keep the values $VAL_1, \ldots, VAL_k$ corresponding to the subtree $T[I]$. Denote, as in Section 5.2, the last vertex of the level greater than 2 by $p$, and its parent by $q$. Assume that the algorithm $A$ has been applied to a tree $T$ of order $n$. Thus, for every vertex $I$ of $T$ the information required in the record $TREE[I]$ is known. When the next tree $S(T)$ is generated by the algorithm $G$, the only vertices $v$ whose $TREE[v]$ record has to be changed are the vertices from $p$ to $n$, the vertex $q$ and all the vertices along the new path from the vertex $n$ up to the root of $S(T)$. Thus, to obtain the required information for the whole tree $S(T)$, that is the values $VAL_1, VAL_2, \ldots, VAL_k$ for the root, we apply the algorithm $A$ only to those vertices mentioned and use the values computed in the previous run for the other vertices which are the roots of the subtrees unchanged by the successor function. This integration of $A$ into $G$ is called the algorithm $G(A)$.

5.4 Trees with Maximum Average Order of Subtrees

Let $T$ be a tree of order $n$. The average order of a subtree, $Ave(T)$, is defined by the following formula

$$
Ave(T) = \frac{\sum (\# \text{ of vertices in this subtree})}{(\# \text{ of all subtrees of } T)}
$$

$$
Ave(T) = \frac{(\# \text{ of vertices in this subtree})}{(\# \text{ of all subtrees of } T)}
$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The example we are about to present clarifies this concept.

**Example 5.2** Consider the tree $T$ of order 4 depicted in Figure 5.7.

![Figure 5.7](image)

In Figure 5.8 we list all possible subtrees of $T$ with their frequencies of occurrence in $T$ and the total number of vertices in each kind of a subtree.

<table>
<thead>
<tr>
<th>Subtrees</th>
<th>Frequency</th>
<th># of Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Subtree 1" /></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><img src="image" alt="Subtree 2" /></td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td><img src="image" alt="Subtree 3" /></td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td><img src="image" alt="Subtree 4" /></td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

![Figure 5.8](image)

The average order of a subtree for $T$ is then

$$
Ave(T) = \frac{4 + 6 + 9 + 4}{4 + 3 + 3 + 1} = \frac{23}{11} \approx 2.09.
$$
We devote this section to the following problem:

Among all free trees of a given order \( n \), find a tree / the trees \( T \) for which \( \text{Ave}(T) \) is maximum.

Thus we have to examine all free trees of order \( n \), compute the average order of subtrees for each one of them and, while moving from one tree to the next one, remember the result and the structure of the tree with the current maximum computed value. Let's concentrate now on the algorithm A1 which computes the average order of subtrees for a single tree. Consider a tree \( T[r] \) with the root \( r \) and define the values \( a_r, b_r, c_r, d_r \) in the following way:

\[
\begin{align*}
    a_r &= \text{the number of subtrees of } T[r] \text{ containing } r \\
    b_r &= \text{the total number of vertices in all subtrees counted in } a_r \\
    c_r &= \text{the number of subtrees of } T[r] \text{ not containing } r \\
    d_r &= \text{the total number of vertices in all subtrees counted in } c_r.
\end{align*}
\]

Then of course,

\[
\text{Ave}(T[r]) = \frac{b_r + d_r}{a_r + c_r}.
\]

Let \( r \) have children \( v_1, v_2, ..., v_m \) and assume that the corresponding values \( a_i, b_i, c_i, d_i \) are already known for every subtree with the root at \( v_i \), \( 1 \leq i \leq m \). This situation is depicted in Figure 5.9.
The correlation between $a_r$, $b_r$, $c_r$, $d_r$ and $a_i$'s, $b_i$'s, $c_i$'s, $d_i$'s is established in our next lemma.

**Lemma 5.3** For the notation described above the following equalities hold:

- $a_r = \prod_{i=1}^{n}(1 + a_i)$
- $b_r = a_r + \sum_{j=1}^{n}(b_j\prod_{i \neq j}(1 + a_i)) = a_r [1 + \sum_{j=1}^{n} \frac{b_j}{(1+a_j)}]$  
- $c_r = \sum_{i=1}^{n}(a_i + c_i)$  
- $d_r = \sum_{j=1}^{n}(b_i + d_i)$.

**Proof.** Let $T_i$ denote the branch of $T[r]$ with the root at $v_i$. Consider now a subtree of $T[r]$ containing $r$. Its intersection with $T_i$ is either empty or forms a subtree containing $v_i$. Thus, to select a subtree of $T[r]$ containing $r$ we have $1 + a_i$ possibilities for each branch and consequently $a_r = \prod_{i=1}^{n}(1 + a_i)$.
The total number of vertices in all subtrees of $T_i$ containing $v_i$ is given by $b_i$. Those vertices are repeated as many times as many subtrees of $T[r] - T_i$ containing $r$ we have, and this quantity is given by $\prod_{i \in j} (1 + a_i)$. Thus the total number of vertices in all subtrees of $T[r]$ containing $r$ not counting the root is equal to $\sum_{j=1}^{n} (b_j \prod_{i \in j} (1 + a_i))$. The root has to be present in each one of $a_r$ subtrees and therefore this value has to be added to the previous sum to obtain $b_r$.

A subtree of $T[r]$ not containing $r$ is entirely contained inside $T_i$ for some $i$, so it is a subtree of $T_i$ either containing $v_i$ or not. Therefore we have $a_i + c_i$ of those subtrees for $1 \leq i \leq m$ and as a result we obtain $c_r = \sum (a_i + c_i)$ and also $d_r = \sum (b_i + d_i)$. □

Let $v$ be an end-vertex of a tree $T$. Then it is trivial to evaluate the values $a, b, c$ and $d$ corresponding to that vertex $v$; they are always given by:

\[
\begin{align*}
  a_v &= 1 \\
  b_v &= 1 \\
  c_v &= 0 \\
  d_v &= 0
\end{align*}
\]

It should be now apparent how starting from the end-vertices we process the tree upwards, finally computing the values $a, b, c$ and $d$ for the root.

As in Section 5.3 we combine the algorithm $A1$ with the algorithm $G$ generating the structures of all free trees of a given order, producing the algorithm $G(A1)$. Consider for example the tree $T$ of order 96 in Figure 5.12 and assume that the values $a, b, c$ and $d$ are known for every vertex of $T$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The successor of $T$, $S(T)$, is shown in Figure 5.13. As we see the subtrees rooted at $v_1$, $v_2$ and $w_1$ have not been changed and therefore the values $a$, $b$, $c$ and $d$ corresponding to them, known from the previous computation, can be used now.

Note that, as soon as we know the $a$, $b$, $c$, $d$ values for a certain root $s$ we don't need to compute them for any other vertex in the tree rooted at $s$. Thus the algorithm $A1$ has has to be applied to only 5 out of 96 vertices of $S(T)$ to compute the corresponding $a$'s, $b$'s, $c$'s and $d$'s, namely to the vertices $w_2$, $w_3$, $u$, $v_3$ and finally $r$, in that order. The average order of a subtree for $S(T)$ is 59.82.
The problem of finding the trees with a maximum average order of a subtree was examined by Jamison in [8]. Jamison gave results (obtained by an exhaustive search) for the families of trees up to order 10. He also predicted the results for trees of orders up to 15, but these values weren't supported by exhaustive computation. Using the method just described we are able to compute the maximum average order of subtrees, and the trees for which this extremal value is attained, for the families of trees up to the order 23. The results are given below and confirm Jamison's prediction through order 15. For each order we present the maximum average order of subtrees and the unique tree attaining it.
Order = 8. Maximum average order of a subtree = 4.31852

Order = 9. Maximum average order of a subtree = 4.89937

Order = 10. Maximum average order of a subtree = 5.54054

Order = 11. Maximum average order of a subtree = 6.15501
Order = 12. Maximum average order of a subtree = 6.84330

Order = 13. Maximum average order of a subtree = 7.50149

Order = 14. Maximum average order of a subtree = 8.21411

Order = 15. Maximum average order of a subtree = 8.92509
Order = 16. Maximum average order of a subtree = 9.66694

Order = 17. Maximum average order of a subtree = 10.40939

Order = 18. Maximum average order of a subtree = 11.16166

Order = 19. Maximum average order of a subtree = 11.94644
Order = 20. Maximum average order of a subtree = 12.71972

Order = 21. Maximum average order of a subtree = 13.51816

Order = 22. Maximum average order of a subtree = 14.32897

Order = 23. Maximum average order of a subtree = 15.12951
Figure 5.14 provides some experimental data obtained after running the Pascal code for the algorithm G(A1) on a VAX 8700.

<table>
<thead>
<tr>
<th>order</th>
<th># of trees</th>
<th>CPU time</th>
<th>evaluated trees/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>23</td>
<td>0:00:00.02</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>47</td>
<td>0:00:00.02</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>106</td>
<td>0:00:00.02</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>235</td>
<td>0:00:00.04</td>
<td>5875</td>
</tr>
<tr>
<td>12</td>
<td>551</td>
<td>0:00:00.12</td>
<td>4591</td>
</tr>
<tr>
<td>13</td>
<td>1301</td>
<td>0:00:00.23</td>
<td>5656</td>
</tr>
<tr>
<td>14</td>
<td>3159</td>
<td>0:00:00.59</td>
<td>5354</td>
</tr>
<tr>
<td>15</td>
<td>7741</td>
<td>0:00:01.50</td>
<td>5160</td>
</tr>
<tr>
<td>16</td>
<td>19320</td>
<td>0:00:03.67</td>
<td>5264</td>
</tr>
<tr>
<td>17</td>
<td>48629</td>
<td>0:00:11.02</td>
<td>4412</td>
</tr>
<tr>
<td>18</td>
<td>123867</td>
<td>0:00:22.54</td>
<td>5495</td>
</tr>
<tr>
<td>19</td>
<td>317955</td>
<td>0:00:57.86</td>
<td>5495</td>
</tr>
<tr>
<td>20</td>
<td>823065</td>
<td>0:02:57.52</td>
<td>4636</td>
</tr>
<tr>
<td>21</td>
<td>2144505</td>
<td>0:06:32.62</td>
<td>5462</td>
</tr>
<tr>
<td>22</td>
<td>5623756</td>
<td>0:17:05.77</td>
<td>5482</td>
</tr>
<tr>
<td>23</td>
<td>14828074</td>
<td>0:45:23.82</td>
<td>5443</td>
</tr>
</tbody>
</table>

The complete Pascal code for the algorithm G(A1) is contained in Appendix 3.
5.5 Frequencies of Cospectral Trees

Let $G$ be a graph of order $n$. The *adjacency matrix* of $G$, $M = (g_{ij})$, is an $n \times n$ matrix where

$$g_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are adjacent} \\ 0 & \text{if } i \text{ and } j \text{ are not adjacent} \end{cases}$$

The characteristic polynomial of a matrix $M$ is the polynomial (in $\lambda$) $\det(\lambda I - M)$ and the *characteristic polynomial of a graph* $G$ is the characteristic polynomial of the adjacency matrix of $G$. We say that two graphs are *cospectral* if they have the same characteristic polynomials. A. Schwenk [14] shows that almost all trees are cospectral, namely, if $t_n$ denotes the number of trees of order $n$ and $c_n$ denotes the number of all trees of order $n$ which can not be identified by their characteristic polynomial, then

$$\frac{c_n}{t_n} \to 1, \quad \text{as } n \to \infty.$$  

In this section we would like to investigate the frequencies of cospectral trees. Let $T$ be a tree of order $n$ and denote by $m_k$ the number of ways to select $k$ independent edges from $T$. Sachs [13] and Mowshowitz [9] give the formula for the characteristic polynomial $\text{CHT}(\lambda)$ of a tree $T$ of order $n$:

$$\text{CHT}(\lambda) = \sum_{k=0}^{p} m_k (-1)^k \lambda^{n-2k}, \quad \text{where } p = \lfloor \frac{n}{2} \rfloor.$$  

We will consider a slightly simpler polynomial $T(x)$ defined as follows:

$$T(x) = \sum m_k x^k.$$  

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
It is obvious that two trees $T_1$ and $T_2$ are cospectral if and only if $T_1(x) = T_2(x)$. In Figure 5.15 we present the smallest pair of cospectral trees.

![Diagram of two trees T1 and T2](image)

$T_1(x) = 1 + 7x + 9x^2$

$CHT_1(\lambda) = \lambda^8 - 7\lambda^6 + 9\lambda^4$

$T_2(x) = 1 + 7x + 9x^2$

$CHT_2(\lambda) = \lambda^8 - 7\lambda^6 + 9\lambda^4$

Figure 5.16

In order to find out how many trees from the family of all free trees of order $n$ can not be represented by their characteristic polynomials, we have to compute those polynomials, or equivalently the polynomials $T(x)$, store the coefficients (either in arrays or coded into a couple of values) and after examining all trees of a given order, sort the values obtained to discover the repetitions.

We start with the algorithm A2 to compute the polynomial $T(x)$ for a single tree $T$. Let $A(x)$ and $B(x)$ be polynomials defined as follows:

$A(x) = \sum a_i x^i$

$B(x) = \sum b_i x^i$, where

$a_i =$ the number of ways to select $i$ independent edges from $T$ not incident with the root

$b_i =$ the number of ways to select $i$ independent edges from $T$ requiring one edge to be incident with the root.

Then it is easy to compute the polynomial $T(x)$ for the tree $T$, it is simply given by

$T(x) = A(x) + B(x)$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
In Figure 5.17 we present a tree $T$ of order 14 with the corresponding polynomial $T(x)$ worked out.

$$A(x) = T_1(x)T_2(x)T_3(x) = 1 + 10x + 34x^2 + 44x^3 + 16x^4$$
$$B(x) = xA_1(x)T_2(x)T_3(x) + xT_1(x)A_2(x)T_3(x) + xT_1(x)T_2(x)A_3(x)$$
$$= 10x + 25x^2 + 62x^3 + 58x^4 + 12x^5$$

$$T(x) = 1 + 20x + 59x^2 + 106x^3 + 74x^4 + 12x^5$$

Figure 5.17

The method used in the previous example can be generalized. Let $T$ be a tree of order $n$ with the root $r$ and let $r$ have $m$ children: $v_1, v_2, \ldots, v_m$. Assume that for the vertices $v_1, v_2, \ldots, v_m$ the corresponding polynomials $A_1(x), B_1(x), \ldots, A_m(x), B_m(x)$ are known. The following lemma gives the relation between $A(x), B(x)$ and the corresponding polynomial for the children vertices.

**Lemma 5.4** For the notation introduced above the following equations hold:

$$A(x) = \prod_{i=1}^{m} T_i(x)$$

$$B(x) = 1 + x \sum_{i=1}^{m} A_i(x) \frac{A(x)}{T_i(x)}.$$
Proof. If the degree of the root $r$ of $T$ is 1, then of course $A(x) = T_1(x)$ and the formula is valid. Assume that the formula holds for every degree of $r$ up to $m-1$. Let $T'$ be a tree with the root $r$ which has $m-1$ children. Let $T$ be formed from $T'$ by adding one more branch adjacent to $r$ (so now $r$ has $m$ children). From the assumption $A'(x) = \prod_{i=1}^{m-1} T_i(x)$. Let $A'(x) = \sum c_i x^i$ and let $T_m(x) = \sum t_i x^i$. To select $k$ independent edges from $T$ we select $k_1$ such edges from $T'$ and $k_2$ of those from the last (added) branch, for any combination of numbers $k_1$ and $k_2$ such that $k_1 + k_2 = k$. Thus $a_k = \sum_{i=1}^k c_i t_{i(k-i)}$ and consequently $A(x) = \prod_{i=1}^{\deg r} T_i(x)$.

Similarly the formula $A_i(x) \prod_{j \neq i} T_j(x) = A_i(x) \frac{A(x)}{T_i(x)}$ represents the polynomial whose $k$'th coefficient denotes the number of ways to select $k$ independent edges from $T$ not incident with $r$ and not incident with $v_1$. Each set of independent edges of $T$ with one edge incident with $r$ has to contain an edge $rv_i$, for some $1 \leq i \leq m$. Since all the other edges have to avoid both $r$ and $v_i$, the corresponding counting polynomial is given by $x A_i(x) \frac{A(x)}{T_i(x)}$. Because we have $m$ choices to select the edge $rv_i$, the second formula follows. □

In general polynomial division is a very time consuming process. We can always represent $\frac{A(x)}{T_1(x)}$ by $\prod_{j \neq i} T_j(x)$, but then to compute $B(x)$ we repeat the same sequence of computation many times. To avoid these unnecessary repetitions we increase the amount of information stored at each child. Let $v_k$ be the $k$'th child of $r$. Then the following polynomials are stored for that vertex:

$$A_k(x);$$
For the root now the polynomial $A(x)$ is stored at its last child, $v_{m}$, as $D_m(x)$ and

$$B(x) = \sum_{i=1}^{m} C_i^m(x).$$

The approach just described is also beneficial when we move from the algorithm $A_2$ to the corresponding algorithm $G(A_2)$ to examine all trees of a given order. Imagine that we want to compute the characteristic polynomial for the successor tree $S(T)$ and the complete information for every vertex of its immediate predecessor, the tree $T$, is known. Denote again the last vertex of level greater than 2 by $p$, and its parent by $q$. After finding the polynomials $A_k(x)'s$, $B_k(x)'s$, $C_i^k(x)'s$ and $D_k(x)'s$ for the vertices $q$ and from $p$ to $n$, we have to compute these polynomials for all the vertices along the path from $n$ up to the root $r$. Let a vertex $u$ be on this path having his left brother $w$ and the last child $v$, as depicted in Figure 5.18.
To compute the polynomials $A_u(x)$ and $B_u(x)$ we use the polynomials $D_v(x)$ and $C^v(x)$'s, to compute $D_u(x)$ and $C^u_i(x)$'s we use $D_w(x)$ and $C^w_i(x)$'s, applying the formulas (5.1), (5.2) and (5.3). Thus along the path $n - r$ only one multiplication of polynomials per vertex is necessary. However, to be able to use the above method we need to store additionally a pointer to a left brother at each vertex.

The problem of finding the frequencies of cospectral tree was discussed by C. Godsil and B. McKay. In [6] they give the distribution of cospectral trees for families of trees of order up to 18. Using the described method we are able to inspect the families of all trees of order up to 22. In Figure 5.19 we present the distribution of cospectral trees up to order 22 (confirming Godsil's and McKay's result up to order 18). Here each column entitled $k$ contains the number of $k$-tuples of trees with the same characteristic polynomial among all trees of the order given in the first column. In Figure 5.20 we present the corresponding frequencies of cospectral trees. Recall that Schwenk [14] proved that $c_n/t_n$ approaches 1. Surprisingly, the frequency data through 22 vertices gives no hint that it is approaching that limit.
<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>27</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>49</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>164</td>
<td>29</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>349</td>
<td>36</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>960</td>
<td>145</td>
<td>20</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2028</td>
<td>326</td>
<td>44</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>5343</td>
<td>985</td>
<td>221</td>
<td>58</td>
<td>18</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>12163</td>
<td>1935</td>
<td>405</td>
<td>88</td>
<td>16</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>30598</td>
<td>5199</td>
<td>1175</td>
<td>351</td>
<td>91</td>
<td>33</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>68655</td>
<td>10370</td>
<td>2067</td>
<td>494</td>
<td>143</td>
<td>32</td>
<td>14</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>172777</td>
<td>26096</td>
<td>5714</td>
<td>1489</td>
<td>419</td>
<td>133</td>
<td>48</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>387844</td>
<td>52685</td>
<td>10925</td>
<td>2717</td>
<td>750</td>
<td>218</td>
<td>75</td>
<td>27</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 5.19 Each column entitled k contains the number of k-tuples of trees with the same characteristic polynomial.
Figure 5.20 Frequencies of cospectral trees

<table>
<thead>
<tr>
<th>n</th>
<th>Frequencies of cospectral trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.0870</td>
</tr>
<tr>
<td>9</td>
<td>0.2128</td>
</tr>
<tr>
<td>10</td>
<td>0.0755</td>
</tr>
<tr>
<td>11</td>
<td>0.2553</td>
</tr>
<tr>
<td>12</td>
<td>0.2160</td>
</tr>
<tr>
<td>13</td>
<td>0.3190</td>
</tr>
<tr>
<td>14</td>
<td>0.2615</td>
</tr>
<tr>
<td>15</td>
<td>0.3191</td>
</tr>
<tr>
<td>16</td>
<td>0.2715</td>
</tr>
<tr>
<td>17</td>
<td>0.3073</td>
</tr>
<tr>
<td>18</td>
<td>0.2611</td>
</tr>
<tr>
<td>19</td>
<td>0.2646</td>
</tr>
<tr>
<td>20</td>
<td>0.2192</td>
</tr>
<tr>
<td>21</td>
<td>0.2136</td>
</tr>
<tr>
<td>22</td>
<td>0.2343</td>
</tr>
</tbody>
</table>

Appendix 4 contains the full Pascal code of the described algorithm G(A2).

5.6 Frequencies of Trees Which Need Two Colors to Attain the Chromatic Sum

In Chapters I, II and III we discuss the chromatic sum, with the special emphasis on trees in Chapter I. There we show that we might be forced to use an arbitrary number of colors to attain the chromatic sum of a tree. However up to order 28 only two or tree colors are needed to obtain the chromatic sum. In this section we present
the frequencies of using two versus three colors in best colorings for families of free trees up to order 25. Let A3 denote the algorithm, discussed in detail in Chapter I, who finds a best coloring for a given tree. Then the algorithm to produce the required frequencies is simply the algorithm G(A3) obtained from A3 in the same way as G(A1) and G(A2) were obtained from A1 and A2 in the previous sections.

The data are contained in the table in Figure 5.21 and the complete Pascal code of the algorithm G(A3) is contained in Appendix 5.

<table>
<thead>
<tr>
<th>N</th>
<th>free trees with 2 colors</th>
<th>free trees with 3 colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>46</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>99</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>223</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>496</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>1 189</td>
<td>112</td>
</tr>
<tr>
<td>14</td>
<td>2 730</td>
<td>429</td>
</tr>
<tr>
<td>15</td>
<td>6 779</td>
<td>962</td>
</tr>
<tr>
<td>16</td>
<td>15 952</td>
<td>3 368</td>
</tr>
<tr>
<td>17</td>
<td>40 654</td>
<td>7 975</td>
</tr>
<tr>
<td>18</td>
<td>97 451</td>
<td>26 416</td>
</tr>
<tr>
<td>19</td>
<td>252 989</td>
<td>64 966</td>
</tr>
<tr>
<td>20</td>
<td>615 646</td>
<td>207 419</td>
</tr>
<tr>
<td>21</td>
<td>1 620 092</td>
<td>524 413</td>
</tr>
<tr>
<td>22</td>
<td>3 992 783</td>
<td>1 630 973</td>
</tr>
<tr>
<td>23</td>
<td>10 614 666</td>
<td>4 213 408</td>
</tr>
<tr>
<td>24</td>
<td>26 447 782</td>
<td>12 852 115</td>
</tr>
<tr>
<td>25</td>
<td>70 859 378</td>
<td>33 777 512</td>
</tr>
</tbody>
</table>

Figure 5.21

5.7 Complexity Issues

Let G and G' be the algorithms to generate the structures of free and rooted trees (respectively), which we discussed in the previous sections. Let A denote a generic algorithm which operates on a single tree, with all the properties described in
Section 5.2. We wish to evaluate the average performance of both $G(A)$ and $G'(A)$, that is the average number of steps per tree that each of the algorithms has to perform. We use our standard notation of $q$, $p$, and $n$ to denote special vertices in a tree of order $n$. Let $S$ denote the successor function for both rooted and free trees. Having computed all the values required by $A$ for all the vertices of a tree $T$ of order $n$, we want to compute these values for the vertices of the tree just after $T$, that is the tree $S(T)$. We have to evaluate only the vertices $q$, from $p$ to $n$ and the vertices from the last vertex in the level sequence, the vertex $n$, along the path to the root which is the vertex 1. This situation is depicted by the example in Figure 5.22.

![Figure 5.22](image)

The difference between the new values for the vertex $q$ in $S(T)$ and the old ones for $q$ in $T$ is the contribution of the vertex $p$ which in $T$ was $q$'s last child and in $S(T)$ is either it's right brother or (only in the case of free trees) might be the last child of the vertex 1 (the root). The old (in $T$) values corresponding to $p$ are very easily determined since in $T$ the vertex $p$ was an end-vertex and therefore it takes a constant time per tree (for both free and rooted trees) to reevaluate $q$.

The number of vertices between the position $p$ and $n$ per tree, for both all rooted trees of order $n$ and all free trees of order $n$, is proved in [2] and in [16] to be
constant. Each one of the vertices between \( p \) and \( n \) is considered in \( S(T) \) at most twice: once as a parent and once as a child. Thus the number of steps per tree that algorithms \( G(A) \) and \( G'(A) \) need to reevaluate these vertices is constant.

Therefore to show that the average number of units of time spent per tree is constant for both \( G(A) \) and \( G'(A) \), we need to concentrate only on the vertices along the path from the vertex \( n \) up to the root. To compute the required values for any vertex along such a path, say a vertex \( v \), we need \((\deg v - 1)\) steps or \(\deg v\) steps if \( v \) is the root. In other words we need as many steps as the number of children a current vertex \( v \) has. Let's call the number of children of a vertex \( v \) the out-degree of \( v \) and denote it by \( od(v) \). Thus the average number of steps per tree to reevaluate the vertices on the path \( P \) from \( n \) up to \( 1 \) is given by the following formula:

\[
\sum_{T:|\Pi|=n} \left( \sum_{v \in P} od(v) \right)
\]

number of trees of order \( n \)

For the algorithm \( G(A) \) we consider all free trees of a given order \( n \), for the algorithm \( G'(A) \) we consider all rooted trees of a given order \( n \).

Now we concentrate only on rooted trees.

**Theorem 5.5** Let \( F_n \) be the family of all rooted trees of order \( n \) with canonical level sequences \((a_1, \ldots, a_n)\). Let \( T_n \) denote the number of these trees. Then the average sum of out-degrees of vertices in the path \( P \) from the last vertex in the level sequence, the vertex \( n \), up to the root is bounded by a constant independent of \( n \). The asymptotic value for that average fulfills the following inequality:
\[
\limsup_{n \to \infty} \frac{\sum_{T \text{ in } F_n} \sum_{v \in P} \text{od}(v)}{T_n} < 58.4
\]

**Proof.** Let \( F_{n,m} \) denote the number of rooted trees of order \( n \) for which the sum of out-degrees of the vertices along the path \( P \) from the vertex \( n \) up to the root is exactly \( m \). Consider the following generating function:

\[
F(x,y) = \sum_n \sum_m F_{n,m} x^n y^m.
\]

We want to evaluate the ratio:

\[
\sum_m \frac{m F_{n,m}}{T_n}.
\]

Note that \( \left. \frac{\partial F}{\partial y} \right|_{y=1} = \sum_n \sum_m m F_{n,m} x^n \). Thus we need to estimate the formula for \( F(x,y) \).

Consider a tree \( T \) with a canonical level sequence. It can be schematically represented as in Figure 5.23.
Since the canonical ordering of the branches is imposed we must have the trees in decreasing order with respect to their level sequences: \( L(T^i) > L(T^{i+1}) \). Thus for any collection of \( k \) rooted trees, they might be put in the places of the \( T_i \)'s in at most one ordering to produce a tree \( T \) with canonical level sequence. Thus a factor of \( \frac{1}{k!} \) is necessary in \( F(x,y) \). Also only the last of these trees can have an end-vertex as a child of its root. Recall from Section 5.2 the generating function for counting rooted trees \( T(x) \). The generating function corresponding to each single branch of \( T_i \) can then be bounded above by \( R(x) = T(x) - x \). Consider now a vertex \( v_1 \) on \( P \) which is the root of \( T_1 \). Figure 5.24 indicates how we can compute the formula describing the part of generating function \( F(x,y) \), called \( D(x,y) \), corresponding to the tree \( T_i \) with \( i < k \).
The root contributes $xy$ to $D(x,y)$ since it increases the order of its tree by one and also the sum of out-degree is increased by one by the rightmost son of $v_i$. Each branch contributes one to the degree of the root or $y$ to $D(x,y)$. Any tree without an end-edge adjacent to the root can be a branch adjacent to the root, which is represented by the generating function $R(x)$, but also for any two neighboring branches both adjacent to the root, say $B_i$ and $B_{i+1}$, we must have $L(B_i) > L(B_{i+1})$. Therefore the generating function $D(x,y)$ can be expressed in a form of the cycle index of the symmetric group:

$$D(x,y) = xy \sum_{m=0}^{\infty} Z(S_m, yR(x)).$$

A more precise explanation can be found in Harary and Palmer [7], Chapter 2. The last $T^k$ is analyzed the same way but $yR(x)$ is replaced by $yT(x)$ since end-vertices are now allowed. Every path $P_i, i < k$, may contain as few as zero vertices, and it contributes to the sum of the out-degrees the number of vertices it possesses. Therefore the corresponding generating function is $1 + xy + x^2y^2 + \ldots = \frac{1}{1 - xy}$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Finally an upper bound for the desired function $F(x,y)$ can be given by the following formula:

$$H(x,y) = \frac{1}{1 - xy} \left[ 1 + \sum_{k=1}^{\infty} \frac{1}{k!(1 - xy)^k} \left( xy \sum_{m=0}^{\infty} Z(S_m, yR(x)) \right)^k \right].$$

$H(x,y)$ is only an upper bound for the function $F(x,y)$ since some sequences of trees $T_1, ..., T_k$ are not allowed in $T$ even if $L(T^i) > L(T^{i+1})$, but are counted by $H(x,y)$.

To simplify the formula for $D(x,y)$ we employ the following identity (see [7])

$$\sum_{m=0}^{\infty} Z(S_m, f(x)) = \exp \sum_{k=1}^{\infty} \frac{f(x^k)}{k}. $$

Thus

$$H(x,y) = \frac{1}{1 - xy} + \sum_{k=1}^{\infty} \frac{1}{k!(1 - xy)^k+1} \left( xy \exp \sum_{i=1}^{\infty} y^i \frac{R(x^i)}{i} \right)^k - 1 \left( xy \exp \sum_{i=1}^{\infty} y^i \frac{T(x^i)}{i} \right).$$

Now we compute the derivative of $H(x,y)$ and evaluate it at $y = 1$.

$$\frac{\partial H(x,y)}{\partial y} \bigg|_{y=1} = \frac{x}{(1 - x)^2} + \sum_{k=1}^{\infty} \frac{(1-x)^k + 1}{k!(1-x)^{2k+2}} \left( x \exp \sum_{i=1}^{\infty} R(x^i) \right) \frac{1}{i} \left( x \exp \sum_{i=1}^{\infty} T(x^i) \right) + \sum_{k=1}^{\infty} \frac{1}{k!(1-x)^{k+1}} \left( x \exp \sum_{i=1}^{\infty} R(x^i) \right) \frac{1}{i} \left( x \exp \sum_{i=1}^{\infty} T(x^i) \right) \left[ (k-1) \sum_{i=1}^{\infty} R(x^i) + \sum_{i=1}^{\infty} T(x^i) \right].$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
To simplify the formula we use the following identity from [7]

$$\sum_{k=1}^{\infty} \frac{T(x^k)}{k} = T(x).$$

Since $x \exp \sum_{k=1}^{\infty} \frac{R(x^k)}{k} = x \sum_{m=0}^{\infty} Z(S_m, R(x))$ and $R(x) = T(x) - x$, we can interpret $x \exp \sum_{k=1}^{\infty} \frac{R(x^k)}{k}$ as the generating function for trees which do not have end-vertices adjacent to the root. Therefore it can also be represented as $T(x)(1 - x)$. Alternatively, by purely algebraic manipulations we can produce the same identity

$$x \exp \sum_{k=1}^{\infty} \frac{R(x^k)}{k} = T(x)(1 - x).$$

Using both identities we can simplify the formula for $\frac{\partial H(x,y)}{\partial y} |_{y=1}$:

$$\frac{\partial H(x,y)}{\partial y} |_{y=1} = \frac{x}{(1 - x)^2} + \sum_{k=1}^{\infty} \frac{k(1-x) + k + 1}{k! (1-x)^3} T^k(x) +$$

$$\sum_{k=1}^{\infty} \frac{k - 1}{k! (1-x)^2} T^k(x)(T(x) - x) + \sum_{k=1}^{\infty} \frac{k - 1}{k! (1-x)^2} T^k(x) \sum_{i=2}^{\infty} (T(x^i) - x^i) +$$

$$\sum_{k=1}^{\infty} \frac{1}{k! (1-x)^2} T^{k+1}(x) + \sum_{k=1}^{\infty} \frac{1}{k! (1-x)^2} T^k(x) \sum_{i=2}^{\infty} T(x^i).$$

Let us use the following notation:
\[ C_{k,1}(x) = \frac{k(1-x) + k + 1}{k! (1-x)^3} T^k(x) ; \]

\[ C_{k,2} = \frac{k - 1}{k! (1-x)^2} T^k(x)(T(x) - x) ; \]

\[ C_{k,3} = \frac{k - 1}{k! (1-x)^2} T^k(x) \sum_{i=2}^{\infty} (T(x) - x)^i; \]

\[ C_{k,4} = \frac{1}{k! (1-x)^2} T^{k+1}(x); \]

\[ C_{k,5} = \frac{1}{k! (1-x)^2} T^k(x) \sum_{i=2}^{\infty} T(x)^i. \]

Recall from Section 5.2 the expansion (see also [12])

\[ T(x) = 1 - b \rho^{1/2} (1 - \frac{x}{\rho})^{1/2} + c \rho^{1/2} (1 - \frac{x}{\rho}) + ... \]

where \( \rho \) is the radius of convergence of the series \( T(x) \) and \( b \) and \( c \) are constants.

For the \( k \)'th power of \( T(x) \) we obtain:

\[ T^k(x) = 1 - k b \rho^{1/2} (1 - \frac{x}{\rho})^{1/2} + \left( \binom{k}{2} b^2 + k c \right) \rho (1 - \frac{x}{\rho}) + ... \]

Thus we can represent \( C_{k,1}(x) \) in the following way:

\[ C_{k,1}(x) = (1 - \frac{x}{\rho})^{-s} g(x) + (1 - \frac{x}{\rho})^{-t} h(x), \]

where \( s = -1/2 \),

\( t = -1 < s \),

\[ g(x) = - \frac{k b \rho^{1/2} \left[ k (1-x) + k + 1 \right]}{k! (1-x)^3}, \]

\( h(x) \) - a function analytical around \( x = \rho \).
Using Pólya's theorem (see Section 5.2) we can estimate the n'th coefficient of the power series \( C_{k,1}(x) \).

\[
\left( C_{k,1}(x) \right)_n \sim \frac{g(\rho)}{\Gamma(-1/2)} n^{s-1} \rho^{-n} = \frac{bp^{1/2} [k (1-\rho) + k + 1]}{2\pi^{1/2} (k-1)! (1 - \rho)^3} n^{3/2} \rho^{-n}.
\]

Recall from Section 5.2 the asymptotic for \( T_n \) (see also [7] or [12]):

\[
T_n \sim \frac{bp^{1/2}}{2\pi^{1/2}} n^{3/2} \rho^{-n}.
\]

Therefore we finally obtain \( \left( C_{k,1}(x) \right)_n \sim T_n \frac{[k (1-\rho) + k + 1]}{(k-1)! (1 - \rho)^3} \).

Similarly we can find the asymptotic for the n-th coefficients of the other power series:

\[
\left( C_{k,2}(x) \right)_n \sim T_n \cdot \left[ \frac{k^2 - 1}{k! (1-\rho)^2} - \frac{\rho}{(k-2)! (1-\rho)^2} \right];
\]

\[
\left( C_{k,4}(x) \right)_n \sim T_n \cdot \frac{k + 1}{k! (1-\rho)^2}.
\]

To evaluate \( C_{k,3}(x) \) and \( C_{k,5}(x) \) consider the series \( \sum_{k=1}^{\infty} T(x^k) \). We can simplify it by changing the order of summation:

\[
\sum_{k=2}^{\infty} T(x^k) = \sum_{k=2}^{\infty} \sum_{j=1}^{\infty} T_j x^{jk} = \sum_{j=1}^{\infty} T_j \sum_{k=2}^{\infty} x^{jk} = \sum_{j=1}^{\infty} T_j \frac{x^2j}{1-x^j}.
\]

Since the function \( \sum_{j=1}^{\infty} T_j \frac{x^2j}{1-x^j} \) is analytic at \( x = \rho \) (the radius of convergence is \( \sqrt{r} \)), see [12], using Pólya's theorem again we obtain the following approximate values:
\[
\left( T(x) \sum_{k=2}^{\infty} T(x^k) \right)_n \sim T_n \sum_{k=2}^{\infty} T(\rho^k) = T_n \sum_{k=1}^{\infty} T_k \frac{\rho^{2k}}{1-\rho^k}.
\]

The series \( \sum_{k=1}^{\infty} T_k \frac{\rho^{2k}}{1-\rho^k} \) is convergent and its approximate sum is 0.191837403 (done by computer). Let us denote this value by \( A \). Then the last two estimates of \( C_{k,i} \) are:

\[
(C_{k,3}(x))_n \sim T_n \cdot \left[ \frac{A}{(k-2)! (1-\rho)^2} - \frac{\rho^2}{(k-2)! (1-\rho)^3} \right];
\]

\[
(C_{k,5}(x))_n \sim T_n \cdot \frac{A}{(k-1)! (1-\rho)^2}.
\]

Since \( \frac{\frac{x}{(1-x)^2}}{T_n} \) is negligible for large \( n \), after summing all \( (C_{k,i}(x))_n, 1 \leq i \leq 5 \), over \( k \) and dividing by \( T_n \), we obtain

\[
\frac{\partial H(x,y)}{\partial y} \Bigr|_{y=1} \sim \left( \frac{2e}{(1-\rho)^2} + \frac{3e}{(1-\rho)^3} + \frac{e+1}{(1-\rho)^2} - \frac{pe}{(1-\rho)^2} + \frac{Ac}{(1-\rho)^2} - \frac{ep^2}{(1-\rho)^3} \right)
\]

\[
+ \frac{2e - 1}{(1-\rho)^2} + \frac{eA}{(1-\rho)^2}. \]

Since \( \rho = 0.3383219 \) (see [12]) and \( A = 0.19 \), \( \frac{\partial H(x,y)}{\partial y} \Bigr|_{y=1} \sim 58.4 \) and the thesis of the theorem follows. \( \Box \)

Theorem 5.5 together with preceding it discussion prove that the average time spent per tree for the algorithm \( G_1(A) \) is constant.
Theorem 5.6 Let $G_1$ be the algorithm generating structures of all rooted trees of a given order $n$. Let $A$ be a generic algorithm computing a certain set of values $VAL_1$, ..., $VAL_k$ for a single tree $T$ with the root $r$ and of the order $n$ in such a way that the values $VAL_1(v)$, ..., $VAL_k(v)$ corresponding to a vertex $v$ of $T$ depend solely upon the values $VAL_1(u_i)$, ..., $VAL_k(u_i)$, $1 \leq i \leq m$, corresponding to all the children of $v$, the vertices $u_1$, ..., $u_m$ and $VAL_i = VAL_i(r)$. Then the average number of steps per tree the algorithm $G_1(A)$ perform is bounded by a constant which does not depend upon $n$.

Unfortunately we don't have a similar theoretical result for free trees. However we strongly believe that a theorem analogous to Theorem 5.5 is true for free trees.

Conjecture 5.7 Let $f_n$ be the family of all free trees of order $n$ and let $t_n$ represent the number of them. Each tree from $f_n$ is represented by the level sequence of this tree rooted at one of its vertices according to the rules described in Section 5.2. Then the average sum of out-degrees of vertices in the path $P$ from the last vertex in the level sequence, the vertex $n$, up to the root is bounded by a constant independent of $n$.

We present now in Figure 5.25 some experimental data to support the conjecture.
<table>
<thead>
<tr>
<th>N</th>
<th>rooted trees</th>
<th>free trees</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[\sum_{T \in F_n} \sum_{v \in P} od(v) / T_n]</td>
<td>[\sum_{T \in f_n} \sum_{v \in P} od(v) / t_n]</td>
</tr>
<tr>
<td>6</td>
<td>3.55</td>
<td>4.00</td>
</tr>
<tr>
<td>7</td>
<td>3.71</td>
<td>4.27</td>
</tr>
<tr>
<td>8</td>
<td>3.81</td>
<td>4.26</td>
</tr>
<tr>
<td>9</td>
<td>3.90</td>
<td>4.40</td>
</tr>
<tr>
<td>10</td>
<td>3.96</td>
<td>4.47</td>
</tr>
<tr>
<td>11</td>
<td>4.02</td>
<td>4.56</td>
</tr>
<tr>
<td>12</td>
<td>4.07</td>
<td>4.64</td>
</tr>
<tr>
<td>13</td>
<td>4.12</td>
<td>4.71</td>
</tr>
<tr>
<td>14</td>
<td>4.16</td>
<td>4.77</td>
</tr>
<tr>
<td>15</td>
<td>4.19</td>
<td>4.83</td>
</tr>
<tr>
<td>16</td>
<td>4.23</td>
<td>4.88</td>
</tr>
<tr>
<td>17</td>
<td>4.25</td>
<td>4.93</td>
</tr>
<tr>
<td>18</td>
<td>4.28</td>
<td>4.98</td>
</tr>
<tr>
<td>19</td>
<td>4.30</td>
<td>5.02</td>
</tr>
<tr>
<td>20</td>
<td>4.32</td>
<td>5.06</td>
</tr>
<tr>
<td>21</td>
<td>4.34</td>
<td>5.09</td>
</tr>
<tr>
<td>22</td>
<td>4.36</td>
<td>5.12</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>5.16</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>5.18</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>5.23</td>
</tr>
</tbody>
</table>

Figure 5.25

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER VI

OPEN PROBLEMS

We conclude the dissertation by listing several problems for future study concerning the chromatic sum and tree algorithms.

1. We define the chromatic index of a graph $G$, $I(G)$, to be the smallest natural number $k$ for which $\Sigma(G) = \Sigma_k(G)$. For a given natural number $k$ characterize all trees $T$ for which $I(T) = k$.

2. Caterpillars have chromatic indices not greater than 3. Can we characterize those caterpillars $T$ which have $I(T) = 3$?

3. Consider all trees $T$ with the maximum degree $\Delta(T) = 3$. For such trees the maximum possible chromatic index equals 3, as shown in Figure 6.1.

\[ \begin{array}{c}
\text{T:} \\
\end{array} \]

\[ \begin{array}{c}
\text{Figure 6.1 } \Sigma_3(T) = 20 < \Sigma_2(T) = 21 \\
\end{array} \]
Similarly, for other values of $\Delta$, what is the maximum chromatic index $I(T)$ for all the trees $T$ with maximum degree not greater than $\Delta$? Note that the Grundy property implies the following inequality: $I(T) \leq \Delta + 1$.

4. For a tree $T$ we define $T'$ to be the tree obtained from $T$ by removing all the endpoints. We conjecture that $I(T) \leq 1 + I(T')$.

5. In Chapter II we show that for any natural numbers $k$ and $t$ there exists a $k$ chromatic graph that needs $k + t$ colors to attain the chromatic sum (a $G_k^t$ graph). Find the smallest graphs with this property. In particular, for $t \geq k$ and $k$ fixed, is it always necessary for $G_k^t$ graphs to grow exponentially in $(t - k)$?

6. We have defined the chromatic sequence only for Grundy colorings. If we remove the Grundy requirement, do we retain the inverted unimodal property for trees? Is the new chromatic sequence still unconstrained for general graphs?

7. In Section 5.4 we present the trees with the maximum average order of a subtree up to order 23. All of them are caterpillars. Thus Jamison's conjecture that for any order the tree with the maximum average order of a subtree must be a caterpillar, remains an enticing open problem.

8. In Chapter V we show that the average number of steps per tree to reevaluate the vertices on the path $P$ from $n$ up to 1 is given by the following formula:

$$\sum_{T:|\Pi|=n} \left( \sum_{v \in P} \text{od}(v) \right)$$

number of trees of order $n$.
We proved that for rooted trees:

\[
\limsup_{n \to \infty} \sum_{T \text{ in } F_n} \sum_{\text{ od(v) } \in P \text{ in } T_n} < 58.4
\]

However the experimental data are considerably smaller. Is it possible to give a more precise estimation? We also conjecture a corresponding bound for free trees. Ideally we would like to see the exact asymptotic analysis, but that appears to be an extremely difficult problem.

9. The following question was raised by Stephen Hedetniemi:

given all trees of a fixed order, which tree occurs in a specified position \( k \) in the reverse lexicographic listing produced by the algorithms considered in Chapter V?

He suggested the value of such information. For example, if we have \( n \) processors, we could attain a speed-up by a factor of \( n \) by partitioning the list into \( n \) equal parts and running an application algorithm on those \( n \) parts simultaneously.
APPENDIX 1
Generating all rooted trees of a given order

(*--------------------------------------------------------------------------*
* This program generates all rooted trees of a given order N by            *
* generating their level sequences                                         *
*                                                                          *
*--------------------------------------------------------------------------*)
PROGRAM GENRT(INPUT,OUTPUT,ROOTEDTREES);

CONST
MAXN = 30;

TYPE
ARRAYTYPE = ARRAY[1..MAXN] OF INTEGER;

VAR
ROOTEDTREES : TEXT; (* contains the level sequences of all rooted trees of n vertices *)
I, COUNT, (* the number of all rooted trees of N vertices *)
N, (* the order of the trees *)
P : INTEGER; (* the position of the last vertex at level > 2 *)
L, (* level sequence *)
PREV, (* parent array *)
SAVE : ARRAYTYPE; (* old PREV *)

(*--------------------------------------------------------------------------*
* PROCEDURE FIRST_TREE                                                   *
*--------------------------------------------------------------------------*)
PROCEDURE FIRST_TREE(VAR L,
PREV,
SAVE : ARRAYTYPE;
N : INTEGER;
VAR P : INTEGER);

VAR
I : INTEGER;

BEGIN
FOR I := 1 TO N DO
BEGIN
PREV[I] := I;
SAVE[I] := 0;
L[I] := I
END;
P := N
END;

100

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PROCEDURE NEXT_TREE(VAR L,
    PREV,
    SAVE : ARRAYTYPE;
    N : INTEGER;
    VAR P : INTEGER);

VAR
    DIFF : INTEGER;

BEGIN
    IF ((P<N) AND ((L[P]<>2) OR (L[P-1]<2)))
        THEN
        BEGIN
            DIFF := P - PREV[L[P]];
            WHILE P<N DO
                BEGIN
                    SAVE[P] := PREV[L[P]];
                    PREV[L[P]] := P;
                    P := P + 1;
                    L[P] := L[P-DIFF]
                END;
        END;
    WHILE L[P]=2 DO
        BEGIN
            P := P-1;
            PREV[L[P]] := SAVE[P]
        END
END;

PROCEDURE NEXT_TREE(VAR L,
    PREV,
    SAVE : ARRAYTYPE;
    N : INTEGER;
    VAR P : INTEGER);

VAR
    DIFF : INTEGER;

BEGIN
    IF ((P<N) AND ((L[P]<>2) OR (L[P-1]<2)))
        THEN
        BEGIN
            DIFF := P - PREV[L[P]];
            WHILE P<N DO
                BEGIN
                    SAVE[P] := PREV[L[P]];
                    PREV[L[P]] := P;
                    P := P + 1;
                    L[P] := L[P-DIFF]
                END;
        END;
    WHILE L[P]=2 DO
        BEGIN
            P := P-1;
            PREV[L[P]] := SAVE[P]
        END
END;

(*--------- MAIN PROCEDURE ---------*)

BEGIN
    WRITE('GIVE # OF VERTICES, N=');
    READLN(N);
    WRITELN(ROOTEDTREES,'ROOTED TREES OF ORDER',N:3);
    FIRST_TREE(L,PREV,SAVE,N,P);
    COUNT := 1;
    WHILE P>1 DO
        BEGIN
            COUNT := COUNT + 1;
            NEXT_TREE(L,PREV,SAVE,N,P);
            WRITE(ROOTEDTREES,COUNT:3,'. O;
            FOR I := 1 TO N DO
                WRITE(ROOTEDTREES,L[I]:3);
            WRITELN(ROOTEDTREES,')')
        END;
    END;
END.
APPENDIX 2
Generating all free trees of a given order

(* ***********************************************************************************************
* This program generates all FREE TREES of a given order by producing their level sequence.
* ***********************************************************************************************

* - each tree has to be rooted in its center
* - for bicentral trees:
  * - W(T1) <= W(T2)
  * - if = then L(T1)<=L(T2)

* L(T1)=L1=[L2-1,L3-1,L4-1,...,LR-1]
* L(T2)=L2=[L1,LM,L(M+1),...,LN]
* R-LAST ELEMENT OF T1 = M-1, M-BEGINING OF T2

***********************************************************************************************

PROGRAM GENFT(INPUT,OUTPUT,FREE_TREES);

CONST
MAXN =30;

TYPE
ARRAYTYPE = ARRAY [1..MAXN] OF INTEGER;

VAR
L, (* level sequence *)
W : ARRAYTYPE; (* parent sequence *)
N, (* order of the trees *)
P, (* the last vertex of level > 2 *)
Q, (* the parent of P *)
H1, (* the position of the first occurrence of the highest level number in the first subtree T1 *)
H2, (* the position of the first occurrence of the highest level number in the second subtree T2 *)
C, (* the first element of L2 which is not the same as the corresponding element of L1 *)
R : INTEGER; (* the last vertex of T1 *)
FREE_TREES : TEXT; (* contains the level sequences of all free trees of N vertices *)
COUNT : INTEGER; (* the number of free trees of N vertices *)

(* ***********************************************************************************************
* PROCEDURE NEXT_TREE
* ***********************************************************************************************

PROCEDURE NEXT_TREE(VAR L,
W : ARRAYTYPE;
N : INTEGER;
VAR P,
Q,
VAR
FIXIT,
NEEDR,
NEEDC,
NEEDH2 : BOOLEAN;
OLDP,
OLDLQ,
OLDWQ : INTEGER;
DELTA,
I : INTEGER;
BEGIN
FIXIT := FALSE;
IF ((C=N+1) OR
((P=H2) AND
((L[H1]=L[H2]+1)AND(N-H2=R-H1)) OR
((L[H1]=L[H2])AND(N-H2+1<R-H1))))
THEN
IF L[R]>3
THEN
BEGIN
P := R;
Q := W[R];
IF H1 = R
THEN
   H1 := H1 - 1;
   FIXIT := TRUE
END
ELSE
BEGIN
P := R;
R := R-1;
Q := 2
END;
NEEDR := FALSE;
NEEDC := FALSE;
NEEDH2 := FALSE;
IF P <= HI
THEN
   H1 := P-1;
IF P <= R
THEN
   NEEDR := TRUE
ELSE
   IF P <= H2
   THEN
      NEEDH2 := TRUE
ELSE
   IF ((L[H2]=L[H1]-1)AND(N-H2=R-H1))
   THEN
BEGIN
  IF \( P \leq C \) THEN
    NEEDC := TRUE
  END
ELSE
  C := MAXINT;
OLDP := P;
DELTA := Q-P;
OLDLQ := L[Q];
OLDWQ := W[Q];
P := MAXINT;
FOR I := OLDP TO N DO
BEGIN
L[I] := L[I+DELTA];
IF L[I]=2 THEN
  W[I] := 1
ELSE
  BEGIN
    P := 1;
    IF L[I]=OLDLQ THEN
      Q := OLDWQ
    ELSE
      Q := W[I+DELTA]-DELTA;
    W[I] :=Q
  END;
  IF ((NEEDR) AND (L[I]=2)) THEN
  BEGIN
    NEEDR :=FALSE;
    NEEDH2 :=TRUE;
    R := I-1
  END;
  IF (NEEDH2 AND (L[I]<=L[I-1]) AND (I>R+1)) THEN
  BEGIN
    NEEDH2 :=FALSE;
    H2 :=I-1;
    IF ((L[H2]=L[H1]-1) AND (N-H2=R-H1)) THEN
      NEEDC :=TRUE
    ELSE
      C :=MAXINT;
  END;
  IF (NEEDC) THEN
    IF (L[I]<=L[H1-H2+I]-1) THEN
    BEGIN
      NEEDC := FALSE;
      C := 1
    END
  ELSE
    REPRODUCED WITH PERMISSION OF THE COPYRIGHT OWNER. FURTHER REPRODUCTION PROHIBITED WITHOUT PERMISSION.
C := I + 1;
END;(*FOR*)
IF (FIXIT)
THEN
BEGIN
  R := N-H1+1;
  FOR I:=R+1 TO N DO
  BEGIN
    L[I] := I-R+1;
    W[I] := I-1
  END;
  W[R+1] := 1;
  H2 := N;
  P := N;
  Q := P-1;
  C := MAXINT
END
ELSE
BEGIN
  IF (P = MAXINT)
  THEN
  BEGIN
    IF L[OLDP-1]<>2
    THEN
      P := OLDP-1
    ELSE
      P := OLDP-2;
    Q := W[P]
  END;
  IF (NEEDH2)
  THEN
  BEGIN
    H2 := N;
    IF ((L[H2]=L[H1]-1)AND(H1=R))
    THEN
      C := N+1
    ELSE
      C := MAXINT
  END
END
END;

(*--------------------------------------------------------
*       PROCEDURE FIRST_TREE
*--------------------------------------------------------*)
PROCEDURE FIRST_TREE(VAR L,
                      W : ARRAYTYPE;
                      N : INTEGER;
                      VAR P,
                      Q,
                      H1,
                      H2,
                      C,
R : INTEGER);

VAR
  K,
  I : INTEGER;

BEGIN
  K := (N DIV 2) + 1;
  FOR I := 1 TO K DO
    BEGIN
      L[I] := I;
      W[I] := I-1
    END;
  L[K+1] := 2;
  W[K+1] := 1;
  FOR I := K+2 TO N DO
    BEGIN
      L[I] := I-K+1;
      W[I] := I-1
    END;
  P := N;
  Q := N-1;
  H1 := K;
  H2 := N;
  R := K;
  IF (N DIV 2)<((N+1)DIV 2)
    THEN C := MAXINT
    ELSE C := N + 1
END;

(*READING CPU TIME*)

EXTERNAL FUNCTION LIBSINIT_TIMER(
  HANDLE_ADR : [REFERENCE] UNSIGNED := %IMMED 0
) : INTEGER; EXTERNAL;

EXTERNAL FUNCTION LIBSHOW_TIMER(
  HANDLE_ADR : [REFERENCE] UNSIGNED := %IMMED 0;
  CODE : INTEGER;
  [IMMEDIATE,UNBOUND]
PROCEDURE ACTION_RTN( OUT_STR : [CLASS_S] PACKED ARRAY [L..U:INTEGER] OF CHAR; USER_ARG : INTEGER) := %IMMED 0;

PROCEDURE USER_ACTION_RTN(
  USER_ARG : INTEGER);
BEGIN
  WRITELN(FREE_TREES);
  WRITELN(FREE_TREES,OUT_STR:45);
END;

(*PROCEDURE WRITE_LEVEL*)
PROCEDURE WRITE_LEVEL(L : ARRAYTYPE;
       COUNT : INTEGER);

VAR
   I : INTEGER;

BEGIN
   WRITELN(FREE_TREES);
   WRITE(FREE_TREES,COUNT:7,'. (');
   FOR I := 1 TO N-1 DO
      WRITE(FREE_TREES,L[I]:2,',');
   WRITELN(FREE_TREES,L[N]:2,')')
END;

/* *************************************/
/* MAIN PROCEDURE */
/* *************************************/

BEGIN
   WRITE('GIVE THE NUMBER OF VERTICES, N= ');
   READ(N);
   WRITELN(FREE_TREES,' NUMBER OF VERTICES = ',N:2);
   LIB$INIT_TIMER;
   FIRST_TREE(L,W,N,P,Q,H1,H2,C,R);
   COUNT := 1;
   WRITE_LEVEL(L,COUNT);
   WHILE Q>0 DO
      BEGIN
         NEXT_TREE(L,W,N,P,Q,H1,H2,C,R);
         COUNT := COUNT + 1;
         WRITE_LEVEL(L,COUNT)
      END;
   LIB$SHOW_TIMER(0,USER_ACTION_RTN,5);
END.
APPENDIX 3

Finding the tree with the maximum average order of a subtree

This program generates all FREE TREES of a given order by producing their level sequence and computes the number of all subtrees for each tree together with the total number of vertices in all subtrees of a given tree. Then the average order of a subtree is computed by dividing the total # of vertices by the total # of subtrees. This program finds all the trees with the maximum average, i.e. whose average is greater than max one - 0.00001.

- each tree has to be rooted in its center
- for bicentral trees:
  - W(T1) <= W(T2)
  - if = then L(T1)<=L(T2)
L(T1)=L1=[L2-1,L3-1,L4-1,...,LR-1]
L(T2)=L2=[L1,LM,L(M+1),...LN]
R-LAST ELEMENT OF T1 = M-1, M-BEGINING OF T2

PROGRAM SUBTREES(INPUT,OUTPUT,FREE_TREES);

CONST
MAXN =30;
MAXLONG = 100;

TYPE
ARRAYLONG = ARRAY [1..MAXLONG] OF INTEGER;
ARRAYTYPE = ARRAY [1..MAXN] OF INTEGER;
INFRECORD = RECORD
  NOSONS : INTEGER;
  SON : INTEGER;
  BROTHER: INTEGER;
  NO_SUBTREES : INTEGER;
  NO_VERT : INTEGER;
  SUB : INTEGER;
  VERT: INTEGER;
END;

TREARRAY = ARRAY[1..MAXN] OF INFRECORD;
VECTOR = ARRAY [1..4] OF INTEGER;

VAR
L, (* level sequence *)
W : ARRAYTYPE; (* parent sequence *)
N, (* order of the trees *)
P, (* the last vertex of level > 2 *)
PROCEDURE COMPUTE_TREE

PROCEDURE COMPUTE_TREE(V AR TREE: TREEARRAY;
FIRST, LAST: INTEGER);

VAR
I, K, J: INTEGER;
PTR: INTEGER;
SUBTREES,
VERTICES : INTEGER;
SUB,
VERT: INTEGER;
BEGIN
FOR I := LAST DOWNTO FIRST DO
BEGIN
IF TREE[I].NOSONS = 0
THEN
BEGIN
TREE[I].NO_VERT := 1;
TREE[I].NO_SUBTREES := 1;
TREE[I].SUB := 0;
TREE[I].VERT := 0;
END
ELSE
BEGIN
PTR := TREE[I].SON;

Q, (* the parent of P *)
H1, (* the position of the first occurrence of the highest level number
in the first subtree T1 *)
H2, (* the position of the first occurrence of the highest level number
in the second subtree T2 *)
C, (* the first element of L2 which is not the same as the corresponding
element of L1 *)
R : INTEGER;
FREE_TREES : TEXT; (* contains the level sequences of all free trees of N vertices *)
COUNT : INTEGER;
BEST : ARRAYLONG;
NO_VERTICES, (* the number of free trees of N vertices *)
NO_SUBTREES : VECTOR;
TREE : TREEARRAY;
(* the level sequence of the tree with the maximum so far *)
(* the total # of vertices in all subtrees of the best tree so far *)
(* contains information about the tree
_NO_SUBTREES - # of all subtrees containing the root
.SUB - # of all subtrees not containing the root
.NO_VERT - # of all vertices in all subtrees containing the root
.VERT - # of all vertices in all subtrees not containing the root*)
NEW_AVERAGE,
AVERAGE : REAL;
AMOUNT: INTEGER;
I : INTEGER;
(* the maximum average order of a subtree so far *)
(* the average order of a subtree for each current tree *)
(* the number of trees with the maximum av. order of a subtree *)
SUBTREES := 1;
VERTICES := 0;
SUB := 0;
VERT := 0;
FOR J := 1 TO TREE[I].NOSONS DO
BEGIN
  SUBTREES := SUBTREES * (1 + TREE[PTR].NO_SUBTREES);
  SUB := SUB + TREE[PTR].NO_SUBTREES + TREE[PTR].SUB;
  VERT := VERT + TREE[PTR].NO_VERT + TREE[PTR].VERT;
  PTR := TREE[PTR].BROTHER
END;
TREE[I].NO_SUBTREES := SUBTREES;
TREE[I].SUB := SUB;
TREE[I].VERT := VERT;
PTR := TREE[I].SON;
FOR J := 1 TO TREE[I].NOSONS DO
BEGIN
  VERTICES := VERTICES + ((SUBTREES DIV (1 + TREE[PTR].NO_SUBTREES)) * TREE[PTR].NO_VERT);
  PTR := TREE[PTR].BROTHER
END;
TREE[I].NO_VERT := SUBTREES + VERTICES;
END;
END;

********************
*                  *
* PROCEDURE NEXT_TREE *
*                  *
********************

PROCEDURE NEXT_TREE(VAR TREE : TREEARRAY;
VAR L, W : ARRAYTYPE;
N : INTEGER;
VAR P,
Q, H1, H2,
C,
R : INTEGER);

VAR
FIXIT,
NEEDR,
NEEDC,
NEEDH2 : BOOLEAN;
OLDP,
OLDLQ,
OLDWQ : INTEGER;
DELTA,
I : INTEGER;
CHILD,
X,
LASTQ,
LASTN,
NOSONS,
OLDQ: INTEGER;

BEGIN
FIXIT := FALSE;
IF ((C=N+1) OR
  ((P=H2) AND
   ((L[H1]=L[H2]+1) OR
    ((L[H1]=L[H2]) AND (N-H2+1<R-H1)))))
THEN
  IF L[R]>3
  THEN
    BEGIN
      P := R;
      Q := W[R];
      FIXIT := TRUE
    END
  ELSE
    BEGIN
      P := R;
      R := R-1;
      Q := 2
    END;
NEEDR := FALSE;
NEEDC := FALSE;
NEEDH2 := FALSE;
IF P <= HI
THEN
  HI := P-1;
IF P <= R
THEN
  NEEDR := TRUE
ELSE
  IF P <= H2
  THEN
    NEEDH2 := TRUE
  ELSE
    IF ((L[H2]=L[H1]-1) AND (N-H2=R-H1))
    THEN
      BEGIN
        IF P <= C
        THEN
          NEEDC := TRUE
        END
      END
    ELSE
      C := MAXINT;
X := 1;
CHILD := TREE[1].SON;
WHILE (X<TREE[1].NOSONS) AND (TREE[CHILD].BROTHER < P) DO
  BEGIN
    X := X + 1;
    CHILD := TREE[CHILD].BROTHER
  END;
TREE[1].NOSONS := X;
TREE[Q].NOSONS := TREE[Q].NOSONS - 1;
TREE[P].BROTHER := TREE[Q].BROTHER;
TREE[W[Q]].NOSONS := TREE[W[Q]].NOSONS + 1;
TREE[Q].BROTHER := P;
OLDP := P;
OLDQ := Q;
DELTA := Q-P;
OLDLQ := L[Q];
OLDWQ := W[Q];
P := MAXINT;
IF FIXIT
THEN
  LASTN := N - H1 +1
ELSE
  LASTN := N;
LASTQ := OLDP;
FOR I := OLDP TO LASTN DO
BEGIN
L[I] := L[I+DELTA];
IF ((TREE[I+DELTA].NOSONS >0) AND ((TREE[I+DELTA].SON - DELTA) <= LASTN))
THEN
  BEGIN
    TREE[I].SON := TREE[I+DELTA].SON - DELTA;
    X := TREE[TREE[I+DELTA].SON].BROTHER;
    NOSONS := 1 ;
    WHILE (NOSONS<TREE[I+DELTA].NOSONS)AND(X-DELTA<=LASTN) DO
    BEGIN
      NOSONS := NOSONS + 1;
      X := TREE[X].BROTHER
    END;
    TREE[I].NOSONS := NOSONS
  END
ELSE
BEGIN
  TREE[I].NOSONS := 0;
  TREE[I].SON := 0
END;
IF (TREE[I+DELTA].BROTHER - DELTA) <= LASTN
THEN
  TREE[I].BROTHER := TREE[I+DELTA].BROTHER - DELTA;
IF L[I]=2
THEN
BEGIN
IF (L[I]=OLDLQ)AND(I<OLDP)
THEN
BEGIN
  TREE[LASTQ].BROTHER := I;
  LASTQ := I;
  TREE[I].NOSONS := TREE[I].NOSONS + 1;
END;
W[I] := 1
END
ELSE
BEGIN
  P := I;
END
ENDIF
IF (L[I]=OLDLQ)
THEN
BEGIN
 IF I<=OLDP
 THEN
  BEGIN
   TREE[LASTQ].BROTHER := I;
   LASTQ := I;
   TREE[OLDWQ].NOSONS := TREE[OLDWQ].NOSONS + 1;
  END;
 Q := OLDWQ
 END
 ELSE
  Q := W[I+DELTA]-DELTA;
 W[I] := Q
END;
IF ((NEEDR) AND (L[I]=2))
THEN
BEGIN
 NEEDR :=FALSE;
 NEEDH2 :=TRUE;
 R := I-1
END;
IF (NEEDH2 AND (L[I]<L[I-1]) AND (I>R+1))
THEN
BEGIN
 NEEDH2 :=FALSE;
 H2 := I-1;
 IF ((L[H2]=L[H1]-1) AND (N-H2=R-H1))
 THEN
   NEEDC :=TRUE
 ELSE
   C := MAXINT;
 END;
 IF (NEEDC)
 THEN
 IF (L[I]<L[H1-H2+I]-1)
 THEN
  BEGIN
   NEEDC := FALSE;
   C := I
  END
 ELSE
  C := I + 1;
 END;(*FOR*)
IF (FIXIT)
THEN
BEGIN
 TREE[CHILD].BROTHER := N-H1 +2;
 TREE[I].NOSONS := TREE[I].NOSONS + 1;
 R := N-H1+1;
 FOR I:=R+1 TO N DO
 BEGIN
  TREE[I].NOSONS :=1;
  TREE[I].SON := I + 1;
 END;

L[I] := I+R+1;
W[I] := I-1
END;
TREE[N].NOSONS := 0;
W[R+1] := 1;
H2 := N;
P := N;
Q := P-1;
C := MAXINT
END
ELSE
BEGIN
IF (P = MAXINT)
THEN
BEGIN
IF L[OLDP-1]<>2
THEN
P := OLDP-1
ELSE
P := OLDP-2;
Q := W[P]
END;
IF (NEEDH2)
THEN
BEGIN
H2 := N;
IF (L[H2]=L[H1]-1) AND (H1=R))
THEN
C := N+1
ELSE
C := MAXINT
END
END;
END;
END;
END;
WHILE OLDQ>1 DO
BEGIN
END;
END;
END;
END;
END;
END;
END;
END;

***********************************************************************

******** PRODUCE_OUTPUT ********

PROCEDURE PRODUCE_OUTPUT(VAR RESULT : TEXT;
AMOUNT,
COUNT : INTEGER;
AVERAGE : REAL;
VAR BEST : ARRAYLONG;
NO_SUBTREES,
NO_VERTICES : VECTOR);
VAR
J,
I : INTEGER;
BEGIN
  WRITELN(RESULT);
  WRITELN(RESULT);
  WRITELN(RESULT);
  WRITELN(RESULT, ' FOR ALL ', COUNT:8, ' FREE TREES WITH ', N:2, ' VERTICES');
  WRITELN(RESULT);
  WRITELN(RESULT, ' THE MAXIMUM AVERAGE ORDER OF A SUBTREE IS ');
  WRITELN(RESULT);
  WRITELN(RESULT, '# OF TREES WHICH HAVE THAT MAXIMUM = ', AMOUNT:2);
  WRITELN(RESULT);
  WRITELN(RESULT, ' LIST OF LEVEL SEQUENCES FOR ALL THOSE TREES ');
  WRITELN(RESULT);
  WRITELN(RESULT, '---------------------------------------------------------------');
  FOR J := 1 TO AMOUNT DO
    BEGIN
      WRITELN(RESULT);
      WRITE(RESULT, '(');
      FOR I := (J-1)*N+1 TO J*N-1 DO
        WRITE(RESULT, BEST[I]:2, ',');
      WRITE(RESULT, BEST[N]:2, ')');
      WRITELN(RESULT);
      WRITELN(RESULT, ' THIS TREE HAS ', NO_SUBTREES[J]:8, ' SUBTREES AND ');
      WRITELN(RESULT);
      WRITELN(RESULT, '-------------------------------------------------------------------------------');
    END;
END;

(*---------------------------------------------------------------
*       PROEDURE FIRST_TREE
*---------------------------------------------------------------)
PROCEDURE FIRST_TREE(VAR TREE : TREEARRAY;
  VAR AVERAGE : REAL;
  VAR L : ARRAYTYPE;
  VAR BEST : ARRAYLONG;
  VAR W : ARRAYTYPE;
  N : INTEGER;
  VAR P,
Q,
H1,
H2,
C,
R : INTEGER;
  VAR NO_VERTICES,
NO_SUBTREES : VECTOR);

VAR
  K,
I : INTEGER;
BEGIN
  K := (N DIV 2) + 1;
  FOR I := 1 TO K DO
    BEGIN
      TREE[I].NOSONS := 1;
      TREE[I].SON := I+1;
      L[I] := I;
      W[I] := I-1
    END;
  TREE[1].NOSONS := 2;
  TREE[2].BROTHER := K+1;
  L[K+1] := 2;
  W[K+1] := 1;
  TREE[K+1].SON := K+2;
  TREE[K+1].NOSONS := 1;
  FOR I := K+2 TO N DO
    BEGIN
      TREE[I].NOSONS := 1;
      TREE[I].SON := I + 1;
      L[I] := I-K+1;
      W[I] := I-1
    END;
  TREE[K].SON := 0;
  TREE[K].NOSONS := 0;
  TREE[N].SON := 0;
  TREE[N].NOSONS := 0;
  COMPUTE_TREE(TREE,1,N);
  FOR I := 1 TO N DO
    BEGIN
      BEST[I] := L[I];
      AVERAGE := (TREE[I].NO_VERT+ TREE[I].VERT) /
                   (TREE[I].NO_SUBTREES+TREE[I].SUB);
      NO_VERTICES[I] := TREE[I].NO_VERT;
      NO_SUBTREES[I] := TREE[I].NO_SUBTREES;
    END;

  IF (N DIV 2)<((N+1)DIV 2) THEN C := MAXINT
  ELSE C := N + 1
END;

(*******************READING CPU TIME*********************)
[EXTERNAL] FUNCTION LIB$INIT_TIMER( HANDLE_ADR : [REFERENCE] UNSIGNED := %IMMED 0 ) : INTEGER; EXTERNAL;
OF CHAR; USER_ARG : INTEGER) := %IMMED 0;
USER_ARG : INTEGER := %IMMED 0) : INTEGER; EXTERNAL;

PROCEDURE USER_ACTION_RTN(
    USER_ARG: INTEGER);
BEGIN
    WRITELN(FREE_TREES);
    WRITELN(FREE_TREES,OUT_STR:45);
END;

(*----------------------------------------------------------
*           MAIN PROCEDURE                                    *
*----------------------------------------------------------*)

BEGIN
    WRITE('GIVE THE NUMBER OF VERTICES, N= ');
    READ(N);
    LIB$INIT_TIMER;
    COUNT := 1;
    FIRST_TREE(TREE,AVERAGE,L,BEST,W,N,P,Q,H1,H2,C,R,NO_VERTICES,
               NO_SUBTREES);
    AMOUNT := 1;
    WHILE Q>0 DO
        BEGIN
            NEXT_TREE(TREEL,W,N,P,Q,H1 ,H2,C,R);
            NEW_AVERAGE := (TREE[l].NO_VERT+TREE[l].VERT) /
                            (TREE[l].NO_SUBTREES + TREE[l].SUB);
            IF AVERAGE < NEW_AVERAGE
               THEN
                   BEGIN
                       AMOUNT := 1;
                       AVERAGE := NEW_AVERAGE;
                       FOR I:= 1 TO N DO
                           BEST[I] := L[I];
                       NO_VERTICES[1] := TREE[l].NO_VERT + TREE[l].VERT;
                       NO_SUBTREES[1] := TREE[l].NO_SUBTREES + TREE[l].SUB
                   END
               ELSE
                   IF AVERAGE < NEW_AVERAGE + 0.00001
                       THEN
                           BEGIN
                               FOR I := 1 TO N DO
                                   BEST[AMOUNT*N+i] := L[i];
                               AMOUNT := AMOUNT + 1;
                               NO_VERTICES[AMOUNT] := TREE[l].NO_VERT + TREE[l].VERT;
                               NO_SUBTREES[AMOUNT] := TREE[l].NO_SUBTREES + TREE[l].SUB
                           END;
                   END;
            COUNT := COUNT + 1
        END;
    LIBS$SHOW_TIMER(,0,USER_ACTION_RTN,5);
    PRODUCE_OUTPUT(FREE_TREES,AMOUNT,COUNT,AVERAGE,BEST,NO_SUBTREES,
                   NO_VERTICES);
    END.
APPENDIX 4

Finding the characteristic polynomials for all trees of a given order

(* ******************************************************
* This program generates all FREE TREES of a given order
* by producing their level sequences and computes the
* polynomials where coef at x**N denotes the # of ways to
* select n independent edges. Characteristic polynomials can be
* computed from those polynomials.
* Then the found polynomial (i.e. their coefficients starting
* from c2) are coded into 1, 2, 3 or 4 numbers and sent to files
* outfile1, 2, 3 or 4.
* ******************************************************)

PROGRAM CHARPOL;

CONST
  MAXN = 22;
  MAXH = 12;

TYPE
  ARRAYTYPE = ARRAY [1..MAXN] OF INTEGER;
  INFRECORD = RECORD
    NOSONS : INTEGER;
    SON   : INTEGER;
    BROTHER: INTEGER;
    LEFT  : INTEGER;
    DELTA : INTEGER;
  END;

  TREEARRAY = ARRAY [1..MAXN] OF INFRECORD;
  ROWARRAY = ARRAY [0..MAXH] OF INTEGER;
  COEFARRAY = ARRAY [1..MAXN] OF ROWARRAY;
  WORKARRAY = ARRAY [1..MAXN] OF ARRAY [0..MAXN] OF ARRAY [0..MAXH] OF INTEGER;

  ORDERARRAY = ARRAY [1..MAXN] OF INTEGER;
  SORARRAY = ARRAY [1..MAXN] OF ARRAY [0..MAXN] OF INTEGER;

VAR
VALUE3, (♦ code for D[7] and D[8] *)
VALUE4: INTEGER; (♦ D[9] ♦)
OP : SORARRAY; (♦ order of partial polynomial *)
OUTFILE1, (♦ contains coded coefficients in VALUE1 *)
OUTFILE2, (♦ contains coded coefficients in VALUE1 and VALUE2 *)
OUTFILE3, (♦ contains coded coefficients in VALUE1, VALUE2 and VALUE3 *)
OUTFILE4 : TEXT; (♦ contains coded coefficients in all four VALUEi's*)
M : ROWARRAY; (♦ maximum possible values of coefficients *)
D : ROWARRAY; (♦ sum of A[1] and B[1] *)
A, (♦ polynomial A *)
B : COEFARRAY; (♦ polynomial B *)
OA, (♦ order of the polynomial A for the subtree rooted at I *)
OB : ORDERARRAY; (♦ order of the polynomial B for the subtree rooted at I *)
TEMP : WORKARRAY; (♦ temporary space *)
L, (♦ level sequence *)
W : ARRAYTYPE; (♦ parent sequence *)
N, (♦ order of the trees *)
P, (♦ the last vertex of level > 2 *)
Q, (♦ the parent of P *)
H1, (♦ the position of the first occurrence of the highest level number in the first subtree T1 *)
H2, (♦ the position of the first occurrence of the highest level number in the second subtree T2 *)
C, (♦ the first element of L2 which is not the same as the corresponding element of L1 *)
R : INTEGER; (♦ the last vertex of T1 *)

******************************************************************************
PROCEDURE COMPUTE_JTREE
******************************************************************************

PROCEDURE COMPUTE_TREE(VAR TREE:TREEARRAY;
VAR A,B : COEFARRAY;
VAR PART:WORKARRAY;
VAR OP : SORARRAY;
VAR OA,OB : ORDERARRAY;
FIRST, LAST: INTEGER);

VAR
I,
R,
D,
V,
K,
S,
J : INTEGER;
PTR : INTEGER;
LEFT : INTEGER;

BEGIN
FOR I := LAST DOWNTO FIRST DO
BEGIN
FOR K := 0 TO OB[I] DO
BEGIN
A[I,K] := 0;
B[I,K] := 0;
END;
A[I,0] := 1;
END;
FOR I := LAST DOWNTO FIRST DO
BEGIN
IF TREE[I].NOSONS = 0 THEN
BEGIN
OA[I] := 0;
OB[I] := 0;
END
ELSE
BEGIN
PTR := TREE[I].SON;
FOR J := 0 TO OB[PTR] DO
BEGIN
PART[PTR,1,J+1] := A[PTR,J];
END;
PART[PTR,1,0] := 1;
OP[PTR,0] := OB[PTR];
OP[PTR,1] := OA[PTR] + 1;
FOR J := 2 TO TREE[I].NOSONS DO
BEGIN
LEFT := PTR;
PTR := TREE[LEFT].BROTHER;
FOR K := 0 TO J-1 DO
BEGIN
FOR S := 0 TO (OP[LEFT,K]+OB[PTR]+1) DO
PART[PTR,K,S] := 0;
END;
FOR S := 0 TO (OP[LEFT,0]+OB[PTR]+1) DO
PART[PTR,0,S] := 0;
END;
FOR S := 0 TO OP[LEFT,0] DO
BEGIN
FOR R := 0 TO OB[PTR] DO
PART[PTR,0,S+R] := PART[PTR,0,S+R]+PART[LEFT,0,S]*(A[PTR,R]+B[PTR,R]);
END;
END;
FOR S := 1 TO OP[LEFT,K] DO (*S>=1-ONE EDGE HAS TO BE SELECTED*)
BEGIN
FOR R := 0 TO OB[PTR] DO
OP[PTR,0] := OP[LEFT,0] + OB[PTR];
PART[PTR,K,0] := 1;
OP[PTR,K] := OP[LEFT,K] + OB[PTR];
END;
END;
FOR S := 0 TO OP[LEFT,0] DO
BEGIN
FOR R := 0 TO OA[PTR] DO
PART[PTR,J,S+R+1] := PART[PTR,J,S+R+1] + PART[LEFT,0,S]*A[PTR,R];
OP[PTR,J] := OP[LEFT,0] + OA[PTR] + 1;
PART[PTR,J,0] := 1;
END;
END;
FOR S := 1 TO OP[PTR,0] DO
A[I,S] := PART[PTR,0,S];
OA[I] := OP[PTR,0];
OB[I] := OA[I];
FOR S:= 1 TO TREE[NOSONS] DO
BEGIN
    FOR K := 1 TO OP[PTR,S] DO
    IF (OP[PTR,S]-OB[I]) THEN
        OB[I] := OP[PTR,S];
END;
END;

********************************************************************************
* * *
* PROCEDURE NEXT_TREE *
* * *
********************************************************************************

PROCEDURE NEXT_TREE(VAR TREE : TREEARRAY;
VAR A,
B : COEFARRAY;
VAR OB,
OA : ORDERARRAY;
VAR OP : SORARRAY;
VAR TEMP : WORKARRAY;
VAR L,
W : ARRAYTYPE;
N : INTEGER;
VAR P,
Q,
H1,
H2,
C,
R : INTEGER);

VAR
FIXIT,
NEEDR,
NEEDC,
NEEDH2 : BOOLEAN;
OLDP,
OLDLQ,
OLDWQ : INTEGER;
DELTA,
I : INTEGER;
CHILD,
X,
LASTQ,
LASTN,
NOSONS,
OLDQ : INTEGER;
BEGIN
FIXIT := FALSE;
IF ((C=N+1) OR
\[(P=H2) \land ((L[H1]=L[H2]+1) \lor ((L[H1]=L[H2]) \land (N-H2+1<R-H1)))\]

THEN
IF \(L[R]>3\) THEN
BEGIN
    \(P := R;\)
    \(Q := W[R];\)
    \(\text{FIXIT} := \text{TRUE}\)
END
ELSE
BEGIN
    \(P := R;\)
    \(R := R-1;\)
    \(Q := 2\)
END;
\(\text{NEEDR} := \text{FALSE};\)
\(\text{NEEDC} := \text{FALSE};\)
\(\text{NEEDH2} := \text{FALSE};\)
IF \(P \leq H1\) THEN
    \(H1 := P-1;\)
IF \(P \leq R\) THEN
    \(\text{NEEDR} := \text{TRUE}\)
ELSE
IF \(P \leq H2\) THEN
    \(\text{NEEDH2} := \text{TRUE}\)
ELSE
IF \((L[H2]=L[H1]-1) \land (N-H2=R-H1)\) THEN
    BEGIN
        IF \(P \leq C\) THEN
            \(\text{NEEDC} := \text{TRUE}\)
    END
ELSE
    \(C := \text{MAXINT};\)
\(X := 1;\)
\(\text{CHILD} := \text{TREE}[1].\text{SON};\)
WHILE \((X<\text{TREE}[1].\text{NOSONS}) \land \text{(TREE[CHILD].BROTHER}< P)\) DO
BEGIN
    \(X := X + 1;\)
    \(\text{CHILD} := \text{TREE}[\text{CHILD}.\text{BROTHER}];\)
END;
\(\text{TREE}[1].\text{NOSONS} := X;\)
\(\text{TREE}[Q].\text{NOSONS} := \text{TREE}[Q].\text{NOSONS} - 1;\)
\(\text{TREE}[P].\text{BROTHER} := \text{TREE}[Q].\text{BROTHER};\)
\(\text{TREE}[W[Q]].\text{NOSONS} := \text{TREE}[W[Q]].\text{NOSONS} + 1;\)
\(\text{TREE}[Q].\text{BROTHER} := P;\)
\(\text{OLDP} := P;\)
\(\text{OLDQ} := Q;\)
\(\text{DELTA} := Q-P;\)
OLDLQ := L[Q];
OLDWQ := W[Q];
P := MAXINT;
IF FIXIT
THEN
    LASTN := N - H1 + 1
ELSE
    LASTN := N;
LASTQ := OLDP;
FOR I := OLDP TO LASTN DO
BEGIN
L[I] := L[I+DELTA];
IF ((TREE[I+DELTA].NOSONS > 0) AND ((TREE[I+DELTA].SON - DELTA) <= LASTN))
THEN
BEGIN
    TREE[I].SON := TREE[I+DELTA].SON - DELTA;
    X := TREE[TREE[I+DELTA].SON].BROTHER;
    NOSONS := 1;
    WHILE (NOSONS<TREE[I+DELTA].NOSONS)AND(X-DELTA<=LASTN) DO
    BEGIN
        NOSONS := NOSONS + 1;
        X := TREE[X].BROTHER;
    END;
    TREE[I].NOSONS := NOSONS;
END
ELSE
BEGIN
    TREE[I].NOSONS := 0;
    TREE[I].SON := 0;
END;
IF (TREE[I+DELTA].BROTHER - DELTA) <= LASTN
THEN
    TREE[I].BROTHER := TREE[I+DELTA].BROTHER - DELTA;
TREE[I].LEFT := TREE[I+DELTA].LEFT - DELTA;
IF L[I]=2
THEN
BEGIN
    IF (L[I]=OLDLQ)AND(I<>OLDP)
    THEN
    BEGIN
        TREE[LASTQ].BROTHER := I;
        LASTQ := I;
        TREE[1].NOSONS := TREE[1].NOSONS + 1;
    END;
    W[I] := 1
END
ELSE
BEGIN
    P := I;
    IF (L[I]=OLDLQ)
    THEN
    BEGIN
        IF I<>OLDP
        THEN
        BEGIN
            ...
TREE[LASTQ].BROTHER := I;
LASTQ := I;
TREE[OLDWQ].NOSONS := TREE[OLDWQ].NOSONS + 1;
END;
Q := OLDWQ
END
ELSE
Q := W[I+DELTA]-DELTA;
W[I] := Q
END;
IF (NEEDR) AND (L[I]=2))
THEN
BEGIN
NEEDR := FALSE;
NEEDH2 := TRUE;
R := I-1
END;
IF (NEEDH2 AND (L[I]<=L[I-1]) AND (I>R+1))
THEN
BEGIN
NEEDH2 := FALSE;
H2 := I-1;
IF ((L[H2]=L[H1]-1) AND (N-H2=R-H1))
THEN
NEEDC := TRUE
ELSE
C := MAXINT
END;
IF (NEEDC)
THEN
IF (L[I]<=L[H1-H2+I]-1)
THEN
BEGIN
NEEDC := FALSE;
C := I
END
ELSE
C := I + 1;
END;(*FOR*)
TREE[OLDP].LEFT := OLDQ;
IF (FIXIT)
THEN
BEGIN
  TREE[CHILD].BROTHER := N-H1+2;
  TREE[I].NOSONS := TREE[I].NOSONS + 1;
  R := N-H1+1;
  FOR I:=R+1 TO N DO
    BEGIN
      TREE[I].NOSONS := 1;
      TREE[I].SON := I + 1;
      L[I] := I-R+1;
      W[I] := 1-1;
    END;
  TREE[N].NOSONS := 0;
  W[R+1] := 1;
H2 := N;
P := N;
Q := P-1;
C := MAXINT
END
ELSE
BEGIN
IF (P = MAXINT)
THEN
BEGIN
IF L[OLDP-1]<2
THEN
P := OLDP-1
ELSE
P := OLDP-2;
Q := W[P]
END;
IF (NEEDH2)
THEN
BEGIN
H2 := N;
IF ((L[H2]=L[H1]-1)AND(H1=R))
THEN
C := N+1
ELSE
C := MAXINT
END
END;
END;
COMPUTE_TREE(TREE,A,B,TEMP,OP,OA,OB,OLDP,N);
WHILE OLDQ>1 DO
BEGIN
COMPUTE_TREE(TREE,A,B,TEMP,OP,OA,OB,OLDQ,OLDQ);
OLDQ := W[OLDQ]
END;
COMPUTE_TREE(TREE,A,B,TEMP,OP,OA,OB,1,1);
END;

***********************
*                     *
*  PROCEDURE FIRST_TREE  *
*                     *
***********************

PROCEDURE FIRST_TREE(VAR TREE : TREEARRAY;
VAR A,
B : COEFARRAY;
VAR TEMP : WORKARRAY;
VAR OP : SORARRAY;
VAR OA,
OB : ORDERARRAY;
VAR L,
W : ARRAYTYPE;
N : INTEGER;
VAR P,
Q,
H1,
VAR
  K, S,
  IJ : INTEGER;
BEGIN
  K := (N DIV 2) + 1;
  FOR I := 1 TO K DO
    BEGIN
      TREE[I].NOSONS := 1;
      TREE[I].SON := I+1;
      L[I] := I;
      W[I] := I-1
    END;
  TREE[1].NOSONS := 2;
  TREE[2].BROTHER := K+1;
  TREE[K+1].LEFT := 2;
  L[K+1] := 2;
  W[K+1] := 1;
  TREE[K+1].SON := K+2;
  TREE[K+1].NOSONS := 1;
  FOR I := K+2 TO N DO
    BEGIN
      TREE[I].NOSONS := 1;
      TREE[I].SON := I+1;
      L[I] := I-K+1;
      W[I] := I-1
    END;
  TREE[K].SON := 0;
  TREE[K].NOSONS := 0;
  TREE[K].DELTA := 1;
  TREE[N].SON := 0;
  TREE[N].NOSONS := 0;
  TREE[N].DELTA := 1;
  FOR I := 1 TO N DO
    BEGIN
      OB[I] := 0;
      OA[I] := 0;
      FOR J := 0 TO N DO
        BEGIN
          FOR S := 0 TO K-1 DO
            TEMP[I,J,S] := 0;
          OP[I,J] := 0;
        END;
      FOR J :=1 TO K-1 DO
        BEGIN
          A[I,J] := 0;
          B[I,J] := 0;
        END;
      A[I,0] := 1;
      B[I,0] := 1;
    END;
COMPUTE_TREE(TREE,A,B,TEMP,OP,OA,OB,1,N);
P := N;
Q := N-1;
H1 := K;
H2 := N;
R := K;
IF (N DIV 2)<((N+1)DIV 2)
 THEN C := MAXINT
 ELSE C := N + 1
END;

FUNCTION PRODUCEMULT(VAR M : ROWARRAY;
A,B : ROWARRAY;
01,02 : INTEGER);
VAR
I : INTEGER;
BEGIN
IF (01<02)
 THEN
 A[02] := 0;
FOR I := 02+1 TO 11 DO
 BEGIN
 A[I] := 0;
 B[I] := 0;
END;
M[2] := 1;
FOR I := 3 TO 4 DO
 M[I] :=M[I-1]*(A[I-1]+B[I-1]);
M[5] := 1;
M[7] :=1;
M[9] := 1;
END;

PROCEDURE MAIN PROCEDURE
BEGIN
OPEN(OUTFILE1,'DISK$SCRATCH:OUTFILE1.DAT,NEW);
OPEN(OUTFILE2,'DISK$SCRATCH:OUTFILE2.DAT,NEW);
OPEN(OUTFILE3,'DISK$SCRATCH:OUTFILE3.DAT,NEW);
OPEN(OUTFILE4,'DISK$SCRATCH:OUTFILE4.DAT,NEW);
REWRITE(OUTFILE1);
REWRITE(OUTFILE2);
REWRITE(OUTFILE3);
REWRITE(OUTFILE4);
N := 15 (* HERE SPECIFY THE ORDER *)
FIRST_TREE(TREE,A,B,TEMP,OP,OA,OB,L,W,N,P,Q,H1,H2,C,R);
PRODUCEMULT(M,A[1],B[1],OA[1],OB[1]);
WHILE Q>0 DO
BEGIN
  NEXT_TREE(TREE,A,B,OB,OA,OP,TEMP,L,W,N,P,Q,H1,H2,C,R);
  D[2] := 0;
  FOR I:=2 TO OA[1] DO
  IF (OB[1]>OA[1])
  THEN
    D[OB[1]] := B[1,OB[1]];
  IF (OB[1] < 5)
  THEN
    BEGIN
      VALUE1 := D[2];
      FOR I := 3 TO OB[1] DO
        VALUE1 := VALUE1 + D[I]*M[I];
      WRITELN(OUTFILE1,VALUE1);
    END
  ELSE
    BEGIN
      IF (OB[1]<7)
      THEN
        BEGIN
          IF (OB[1]=5)
          THEN
            VALUE2 := D[5]
          ELSE
          WRITELN(OUTFILE2,VALUE1,'',VALUE2);
        END
      ELSE
        BEGIN
          IF (OB[1]<9)
          THEN
            BEGIN
              IF (OB[1]=7)
              THEN
                VALUE3 := D[7]
              ELSE
                VALUE3 := D[7]+D[8]*M[8];
              WRITELN(OUTFILE3,VALUE1,'',VALUE2,'',VALUE3);
            END
          ELSE
            BEGIN
              VALUE3 := D[7]+D[8]*M[8];
            END
        END
    END
END
VALUE4 := D[9];
FOR I := 10 TO OB[1] DO
  VALUE4 := VALUE4 + D[I]*M[I];
  WRITELN(OUTFILE3, VALUE1, VALUE2, VALUE3, VALUE4);
END;
END;
END;
CLOSE(OUTFILE1);
CLOSE(OUTFILE2);
CLOSE(OUTFILE3);
CLOSE(OUTFILE4);
END.
APPENDIX 5

Computing the chromatic sum and the maximum number of colors used

This program generates all FREE TREES of a given order
by producing their level sequence and computes their
chromatic sums and max # of colors which have to be used
Only max number of colors is remembered but the program
can easily be changed to remember the chromatic sums

- each tree has to be rooted in its center
- for bicentral trees:
  - W(T1) <= W(T2)
  - if = then L(T1)<=L(T2)

L(T1)=L1=[L2-1,L3-1,L4-1,...,LR-1]
L(T2)=L2=[L1,LM,L(M+1),...,LN]
R-LAST ELEMENT OF T1 = M-1, M-BEGINING OF T2

PROGRAM FREECHROM(FREE_TREES);

CONST
  SKIP = ' ';
  MAXN = 30;
  MAXCOLOR = 4;

TYPE
  ARRAYTYPE = ARRAY [1..MAXN] OF INTEGER;
  INFRECORD = RECORD
    NOSONS : INTEGER;
    SON : INTEGER;
    BROTHER: INTEGER;
    MINSUM : INTEGER;
    RCOLOR: INTEGER;
    DELTA : INTEGER;
    NCOLOR: INTEGER;
    MAXCOL : INTEGER;
    NEXTCO : INTEGER
  END;

  TREEARRAY = ARRAY [ 1  ..MAXN] OF INFRECORD;
  CHROMARRAY = ARRAY [1..3*MAXN] OF INTEGER;
  COLORARRAY = ARRAY [2..MAXCOLOR]OF INTEGER;

VAR
  L, W : ARRAYTYPE;
  N, P, (* level sequence *)
  (* parent sequence *)
  (* order of the trees *)
  (* the last vertex of level > 2 *)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Q, (* the parent of P *)
H1, (* the position of the first occurrence of the highest level number in the first subtree T1 *)
H2, (* the position of the first occurrence of the highest level number in the second subtree T2 *)
C, (* the first element of L2 which is not the same as the corresponding element of L1 *)
R : INTEGER; (* the last vertex of T1 *)
MAX_COLORS: TEXT; (* this file contains the number of trees using i colors *)
TREE : TREEARRAY; (* contains information about each tree
      .MINSUM - the chromatic sum for the subtree rooted
            at the current vertex
      .RCOLOR - the root color in the best coloring
      .NCOLOR - second best choice for the root color
      .DELTA - the difference between MINSUM and the sum of
            colors where the root is colored with NCOLOR
      .MAXCOL - maximal # of colors used in the best coloring
      .NEXTCO - max. # of colors used in second best coloring *)
MCOLOR : COLORARRAY; (* at the entry i contains # of trees using i colors *)

FUNCTION MAXIMUM(VAL1,VAL2 : INTEGER):INTEGER;
BEGIN
  IF VAL1 >= VAL2
  THEN MAXIMUM := VAL1
  ELSE MAXIMUM := VAL2
END;

PROCEDURE COMPUTE_TREE(VAR TREE:TREEARRAY;
                        FIRST,LAST : INTEGER);

VAR
I,
K,
J : INTEGER;
COLORMAX,
COLORADD : ARRAY [1..MAXCOLOR] OF INTEGER;
MINTOTAL : INTEGER;
PTR : INTEGER;
SUM1,
SUM2,
MAX1,
MAX2,
COLOR1,
COLOR2,
NUMBER,
VALUE : INTEGER;
BEGIN
FOR I := LAST DOWNT0 FIRST DO
BEGIN
IF TREE[I].NOSONS = 0
THEN
BEGIN
TREE[I].MINSUM := 1;
TREE[I].RCOLOR := 1;
TREE[I].DELTA := 1;
TREE[I].NCOLOR := 2;
TREE[I].MAXCOL := 1;
TREE[I].NEXTCO := 2
END
ELSE
BEGIN
NUMBER := TREE[I].NOSONS + 2;
IF NUMBER > MAXCOLOR THEN NUMBER := MAXCOLOR;
PTR := TREE[I].SON;
MINTOTAL := 0;
FOR K := 1 TO MAXCOLOR DO
BEGIN
COLORMAX[K] := K;
COLORADD[K] := 0;
END;
FOR J := 1 TO TREE[I].NOSONS DO
BEGIN
MINTOTAL := MINTOTAL + TREE[PTR].MINSUM;
IF TREE[PTR].RCOLOR <= NUMBER THEN
END;
FOR K := 1 TO MAXCOLOR DO
BEGIN
COLORADD[K] := 0;
IF ((VALUE < SUM1) OR ((VALUE = SUM1) AND (COLORMAX[K] < MAX1)))
THEN
BEGIN
COLOR2 := COLOR1;
SUM2 := SUM1;
END;
SUM1 := MAXINT;
SUM2 := MAXINT;
MAX1 := MAXINT;
MAX2 := MAXINT;
END;
SUM1 := MAXINT;
SUM2 := MAXINT;
MAX1 := MAXINT;
MAX2 := MAXINT;
FOR K := 1 TO NUMBER DO
BEGIN
VALUE := COLORADD[K] + K;
COLORADD[K] := 0;
END;
IF (VALUE < SUM1) OR ((VALUE = SUM1) AND (COLORMAX[K] < MAX1))
THEN
BEGIN
COLOR2 := COLOR1;
SUM2 := SUM1;
END;
SUM1 := MAXINT;
SUM2 := MAXINT;
MAX1 := MAXINT;
MAX2 := MAXINT;
END.
COLOR1 := K;
SUM1 := VALUE;
MAX2 := MAX1;
MAX1 := COLORMAX[K]
END
ELSE
IF ((VALUE < SUM2) OR ((VALUE = SUM2) AND (COLORMAX[K] < MAX2)))
THEN
BEGIN
COLOR2 := K;
SUM2 := VALUE;
MAX2 := COLORMAX[K]
END
END;
TREE[I].MINSUM := SUM1 + MINTOTAL;
TREE[I].RCOLOR := COLOR1;
TREE[I].DELTA := SUM2 - SUM1;
TREE[I].NCOLOR := COLOR2;
TREE[I].MAXCOL := MAX1;
TREE[I].NEXTCO := MAX2;
END;
END;

******************************************************************************
*  PROCEDURE NEXT_TREE  *
******************************************************************************
PROCEDURE NEXT_TREE(VAR TREE: TREEARRAY;
VAR L,
W : ARRAYTYPE;
N : INTEGER;
VAR P,
Q,
H1,
H2,
C,
R : INTEGER);

VAR
FIXIT,
NEEDR,
NEEDEC,
NEEDH2 : BOOLEAN;
OLDP,
OLDLQ,
OLDWQ : INTEGER;
DELTA,
I : INTEGER;
CHILD,
X,
LASTQ,
LASTN,
NOSONS,
OLDQ : INTEGER;
BEGIN
  FIXIT := FALSE;
  IF ((C=N+1) OR
      ((P=H2) AND
       ((L[H1]=L[H2]+1) OR
        ((L[H1]=L[H2]) AND (N-H2+1<R-H1))))) THEN
    IF L[R]>3
      THEN
        BEGIN
          P := R;
          Q := W[R];
          FIXIT := TRUE
        END;
    ELSE
      BEGIN
        P := R;
        R := R-1;
        Q := 2
      END;
    NEEDR := FALSE;
    NEEDC := FALSE;
    NEEDH2 := FALSE;
    IF P <= H1
      THEN
        H1 := P-1;
    IF P <= R
      THEN
        NEEDR := TRUE
    ELSE
      IF P <= H2
        THEN
          NEEDH2 := TRUE
      ELSE
        IF ((L[H2]=L[H1]-1) AND (N-H2=R-H1)) THEN
          BEGIN
            IF P <= C
              THEN
                NEEDC := TRUE
            ELSE
              C := MAXINT;
          END;
        X := 1;
        CHILD := TREE[1].SON;
        WHILE (X<TREE[1].NOSONS) AND (TREE[CHILD].BROTHER < P) DO
          BEGIN
            X := X + 1;
            CHILD := TREE[CHILD].BROTHER
          END;
        TREE[1].NOSONS := X;
        TREE[Q].NOSONS := TREE[Q].NOSONS - 1;
        TREE[P].BROTHER := TREE[Q].BROTHER;
TREE[W[Q]].NOSONS := TREE[W[Q]].NOSONS + 1;
TREE[Q].BROTHER := P;
OLDP := P;
OLDQ := Q;
DELTA := Q-P;
OLDLQ := L[Q];
OLDWQ := W[Q];
P := MAXINT;
IF FIXIT
THEN
  LASTN := N - H1 +1
ELSE
  LASTN := N;
LASTQ := OLDQ;
FOR I := OLDP TO LASTN DO
BEGIN
  L[I] := L[I+DELTA];
  IF ((TREE[I+DELTA].NOSONS >0) AND ((TREE[I+DELTA].SON - DELTA) <= LASTN))
  THEN
    BEGIN
      TREE[I].SON := TREE[I+DELTA].SON - DELTA;
      X := TREE[TREE[I+DELTA].SON].BROTHER;
      NOSONS := 1;
      WHILE (NOSONS<TREE[I+DELTA].NOSONS)AND(X-DELTA<=LASTN) DO
        BEGIN
          NOSONS := NOSONS + 1;
          X := TREE[X].BROTHER
        END;
      TREE[I].NOSONS := NOSONS
    END
  ELSE
    BEGIN
      TREE[I].NOSONS := 0;
      TREE[I].SON := 0
    END;
  IF (TREE[I+DELTA].BROTHER - DELTA) <= LASTN
  THEN
    TREE[I].BROTHER := TREE[I+DELTA].BROTHER - DELTA;
  IF L[I]=2
  THEN
    BEGIN
      IF (L[I]=OLDLQ)AND(I<OLDP)
      THEN
        BEGIN
          TREE[LASTQ].BROTHER := I;
          LASTQ := I;
          TREE[I].NOSONS := TREE[I].NOSONS + 1;
        END;
      W[I] := 1
    END
  ELSE
    BEGIN
      P := I;
      IF (L[I]=OLDLQ)
      THEN

BEGIN
IF L<>OLDP
THEN
BEGIN
TREE[LASTQ].BROTHER := I;
LASTQ := I;
TREE[OLDWQ].NOSONS := TREE[OLDWQ].NOSONS + 1;
END;
Q := OLDWQ
ELSE
Q := W[I+DELTA]-DELTA;
W[I] :=Q
END;
IF ((NEEDR) AND (L[I]=2))
THEN
BEGIN
NEEDR :=FALSE;
NEEDH2 :=TRUE;
R := I-1
END;
IF (NEEDH2 AND (L[I]<=L[I-1]) AND (I>R+1))
THEN
BEGIN
NEEDH2 :=FALSE;
H2 :=I-1;
IF ((L[H2]=L[H1]-1) AND (N-H2=R-H1))
THEN
NEEDC :=TRUE
ELSE
C :=MAXINT;
END;
IF (NEEDC)
THEN
IF (L[I]<=L[H1-H2+I]-1)
THEN
BEGIN
NEEDC := FALSE;
C := I
END
ELSE
C := I + 1;
END;(*FOR*)
IF (FIXIT)
THEN
BEGIN
TREE[CHILD].BROTHER := N-H1 +2;
TREE[I].NOSONS := TREE[I].NOSONS + 1;
R := N-H1+1;
FOR I:=R+1 TO N DO
BEGIN
TREE[I].NOSONS :=1;
TREE[I].SON := I + 1;
L[I] := I-R+1;
W[I] := I-1
END;
PROCEDURE PRODUCE_TABLE(VAR RESULT: TEXT;
VAR MCOLOR: COLORARRAY);

VAR
TOTAL,
I: INTEGER;

BEGIN
WRITELN(RESULT, ' ALL ROOTED TREES WITH ', TOTAL, ' VERTICES');
WRITELN(RESULT);
WRITE(RESULT; ' # OF COLORS USED # OF TREES');
WRITE(RESULT; ' # OF COLORS USED # OF TREES');
FOR I := 2 TO MAXCOLOR DO
  IF MCOLOR[I] > 0 THEN
    WRITE(RESULT, SKIP: 8, I: 2, MCOLOR[I]: 23)
END;

(***********************************************************************
 * * 
 *     PROCEDURE FIRST_TREE    *
 * * 
***********************************************************************)

PROCEDURE FIRST_TREE(VAR TREE: TREEARRAY;
VAR CHROMSUM: CHROMARRAY;
VAR MCOLOR: COLORARRAY;
VAR L,
W: ARRAYTYPE;
N: INTEGER;
VAR P,
Q,
H1,
H2,
C,
R: INTEGER);

VAR
K,
I: INTEGER;

BEGIN
  K := (N DIV 2) + 1;
  FOR I := 1 TO K DO
    BEGIN
      TREE[I].NOSONS := 1;
      TREE[I].SON := I + 1;
      L[I] := I;
      W[I] := I - 1
    END;
  TREE[1].NOSONS := 2;
  TREE[2].BROTHER := K + 1;
  L[K + 1] := 2;
  W[K + 1] := 1;
  TREE[K + 1].SON := K + 2;
  TREE[K + 1].NOSONS := 1;
  FOR I := K + 2 TO N DO
    BEGIN
      TREE[I].NOSONS := 1;
      TREE[I].SON := I + 1;
      L[I] := I - K + 1;
      W[I] := I - 1
    END;
  TREE[K].SON := 0;
  TREE[K].NOSONS := 0;
  TREE[K].MINSUM := 1;
  TREE[K].RCOLOR := 1;
TREE[K].DELTA := 1;
TREE[K].NCOLOR := 2;
TREE[K].MAXCOL := 1;
TREE[K].NEXTCO := 2;
TREE[N].SON := 0;
TREE[N].NOSONS := 0;
TREE[N].MINSUM := 1;
TREE[N].RCOLOR := 1;
TREE[N].DELTA := 1;
TREE[N].NCOLOR := 2;
TREE[N].MAXCOL := 1;
TREE[N].NEXTCO := 2;
COMPUTE TREE(TREE,1,N);
FOR I:=N+1 TO 3*(N DIV 2 +1) DO
  CHROMSUM[I] := 0;
FOR I :=2 TO MAXCOLOR DO
  MCOLOR[I] := 0;
P := N;
Q := N-1;
H1 := K;
H2 := N;
R := K;
IF (N DIV 2)<((N+1)DIV 2)
THEN C := MAXINT
ELSE C := N + 1
END;

BEGIN
N := 15; (* SPECIFY ORDER HERE *)
FIRST_TREE(TREE,MCOLOR,L,W,N,P,Q,H1,H2,C,R);
WHILE Q>0 DO
BEGIN
  NEXT_TREE(TREE,L,W,N,P,Q,H1,H2,C,R);
END;
PRODUCE_TABLE(MAX_COLORS,MCOLOR);
END.
REFERENCES


