GENERATION OF PERMUTATIONS OF MULTISETS

by

CHUN WA KO
B Sc , University of Victoria, 1985

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIRMENTS FOR THE DEGREE OF MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

[redacted]
Dr. Frank Ruskey

[redacted]
Dr. Dale Olesky

[redacted]
Dr. John Ellis

[redacted]
Dr. Donald L. Miller

[redacted]
Dr. Paul Smith

University of Victoria

August 1986

Supervisor Professor Frank Ruskey

ABSTRACT

A multiset is a set in which elements may repeat A permutation of a multiset is a permutation of the sequence consisting of all elements in the multiset An algorithm is given that generates all permutations of a multiset such that consecutively generated permutations differ by an interchange of elements On average the permutations are generated in time bounded by a constant, independent of the number of symbols and n, the length of each permutation Ranking and unranking algorithms of order $O(n^2)$ for the generation algorithm are also given

An adjacent interchange generation algorithm for multiset permutations has the following property consecutively generated permutations differ by an interchange of adjacent elements However it is not always possible to generate all permutations of a multiset by adjacent interchange The problem of determining if an adjacent interchange algorithm exists for a multiset is equivalent to determining if a Hamilton path exists in the graph whose vertices are the multiset permutations and whose edges represent adjacent interchange Necessary conditions are given for such graphs *not* to contain any Hamilton path

A rooted ordered tree with n_0+1 leaves and n_i internal nodes with i children each, for $i = 1, \ldots, t$, can be represented uniquely by a sequence consisting of n_i occurrences of i, $i = 0, \ldots, t$ Such sequences are permutations of multisets The results for multiset generation by adjacent interchange are extended to rooted trees represented by these sequences

Necessary conditions on n_i, $i = 0, \ldots, t$, are given for cases where it is impossible to generate by adjacent interchange all trees with the prescribed number of leaves and internal nodes.

Examiners

[redacted]

Professsor Frank Ruskey

[redacted]

Professor Dale Olesky

[redacted]

Professor John Ellis

[redacted]

Professor Donald J. Miller

[redacted]

Professor Paul Smith

Table Of Contents

List Of Figures

Acknowledgement

I wish to thank Professor Frank Ruskey for suggesting the topics of this thesis to me and for his valuable guidance while the research for this thesis was completed

CHAPTER 1

INTRODUCTION

A multiset is a set in which each element may appear a number of times, for example {0, 0, 1, 2, 2}. A permutation of a multiset is a permutation of the sequence consisting of all the elements of the multiset. Several algorithms have been found to generate all permutations of multisets. See [1-5]. A desirable feature for the generation algorithm to possess is for consecutively generated permutations to differ as little as possible. In the list generated by [5] consecutive permutations differ by an interchange of two elements. However this generation algorithm is not efficient. In Chapter 2 a constant average time recursive algorithm is presented that generates multiset permutations by interchange. This algorithm is an extension of Bitner's algorithm [6] that generates combinations by interchange. Ranking and unranking algorithms are also given for this generation algorithm in the binary case.

A more desirable feature for generation algorithms is for successively generated permutations to differ by an interchange of adjacent elements. However with the adjacency restriction it may not always be possible for such an algorithm to exist. In the binary combination case [7], [8] and [9] showed independently that if either the number of 0's or the number of 1's is even then it is impossible to generate all combinations by adjacent interchange. In the multiset permutation case, a graph can be defined such that the vertices are the permutations and an edge between two vertices exists if and only if one vertex can be obtained from the other by an adjacent interchange. The problem of finding an adjacent interchange generation

algorithm is equivalent to finding a Hamilton path in this graph. In Chapter 3 it is shown that no Hamilton paths exist in such a graph if at least two of the elements of the multiset occur more than once, and at most one element occurs an odd number of times.

In a tree define the *degree of a node* as the number of children it has. If a rooted ordered tree has n_i nodes of degree i, $i \geq 1$, and $n_0 + 1$ leaves, then it can be uniquely represented by a sequence of integers, called the degree sequence of the tree, consisting of n_0 0's and n_i i's for $i \geq 1$. [10] used these sequences to generate rooted ordered trees lexicographically. An advantage of representing trees using these sequences is that when two sequences differ by an interchange of adjacent degrees, then the corresponding trees also differ little. The method used in Chapter 3 to determine when an adjacent interchange generation algorithm is possible is extended to the degree sequences. In Chapter 4 it is shown that no adjacent interchange generation algorithms can exist if

(a) $n_1 \geq 0$, all other n_i's = 0, or

(b) at least one $n_i \geq 2$, $i > 0$, and

(b 1) n_0 is odd, one n_i is odd for i even, and < 2 n_j's are odd for j odd,

(b 2) n_i for all even i is even, < 2 n_j's are odd and at least one $n_j > 0$ for j odd, except when $n_0 = 4$, $n_1 = 0$, $n_2 = 2$, and $n_3 = 1$, in which case the parity difference is 1.

CHAPTER 2

MULTISET GENERATION BY INTERCHANGE

Define a multiset as a set in which each element may repeat a fixed number of times. For example S = { 0, 0, 1, 2, 2, 2 } is a multiset. Let 0, ,t denote the distinct elements of a multiset such that element i occurs n_i times. Then the multiset can be represented by the sequence $0^{n_0}1^{n_1}\ t^{n_t}$. A permutation of a multiset is a permutation of its representation sequence. The problem is to generate all permutations of a multiset, with each permutation generated exactly once.

When the multiset contains only two symbols, 0 and 1, with k occurrences of 1 and n-k occurrences of 0, the problem reduces to that of generating all n/k-combinations, or equivalently, that of generating all subsets of size k of a set of size n. This problem has been well studied. However comparatively little research has been done with the problem of generating multiset permutations. The following section examines five known multiset permutation generation (mpg) algorithms and two combination generation algorithms that are closely related to our new mpg algorithm. How each algorithm generates the permutations are intuitively explained, and lower bounds on its complexity are determined, in order that comparisons with the new algorithm are possible.

2.1. Previous Research

Historically most mpg algorithms try to solve the problem by reducing it to the combination generation case and this is the method used by four out of the five mpg algorithms that are discussed here. Another characteristic of all but one of these algorithms is that the permutations are generated in some lexicographical order. When reduced to the binary case, these algorithms generate all combinations in either increasing or decreasing lexicographical order. Algorithm 5 is the only exception, where successively generated permutations differ in exactly two positions. This type of generation is referred to as *generation by interchange*. Algorithms 6 and 7 generate combinations by interchange and our algorithm may be considered as a generalization of Algorithm 7.

2.1.1. Algorithm 1 (Sag)

This algorithm is due to T. M. Sag [1]. It proceeds as follows.

1. Let the symbols be 0, 1, ..., t-1, t. Define an arbitrary ordering on the symbols, say $0 < 1 < \ldots < t\text{-}1 < t$.

2. Take the two largest symbols, t-1 and t. Generate all permutations of these symbols lexicographically, where an element is more significant than an element on its left, contrary to the usual ordering.

3. If $t = 1$ then stop. Otherwise for each permutation generated in step 2, treat t and t-1 as one symbol, say α, and recursively generate all permutations for 0, 1, ..., t-2, α.

Figure 2.01 shows the order the permutations of { 0, 0, 1, 2 } are generated in.

2100 → 2010 → 0210 → 2001 → 0201 → 0021

→ 1200 → 1020 → 0120 → 1002 → 0102 → 0012

Figure 2 01

In the binary case, this algorithm is equivalent to one that generates combinations lexicographically by iteration It is known (see [11]) that in this case the complexity of the algorithm is of the order $\binom{n+2}{k+1}$, where k is the number of 1's and n is the number of 0's and 1's In general, the last level of the recursion is reached $\binom{n_1+\quad+n_t}{n_1\quad n_t}$ times, and at this level the symbols 1, , t are treated as one symbol 1, and all permutations of 0's and 1's are generated Hence the complexity of the algorithm is at least of the order

$$\binom{n_1+\quad+n_t}{n_1\quad n_t}\binom{n+2}{1+n_1+\quad+n_t} = \binom{n}{n_0\quad n_t}\frac{(n+1)(n+2)}{(n_0+1)(n-n_0+1)}$$

Clearly in general this algorithm is not constant average time

2 1 2 Algorithm 2 (Bratley)

This algorithm is due to P Bratley [2] The main idea is that, when the multiset is $0^{n_0}\ t^{n_t}$ all symbols except for t are treated as one, say α, and all combinations of t's and α's are generated For each of these combinations, all permutations of $0^{n_0}\ (t-1)^{n_{t-1}}$ are generated to replace the α-subsequence Figure 2 02 shows the order in which the permutations for { 0, 1, 2, 2 } are generated

2210 → 2201 → 2120 → 2021 → 2102 → 2012

→ 1220 → 0221 → 1202 → 0212 → 1022 → 0122

Figure 2.02

In this algorithm, every time a permutation is generated the entire sequence is scanned. Thus the average complexity for the iterative version is Ω(n).

2.1.3 Algorithm 3 (Hu-Tien)

This algorithm is due to T.C. Hu and B.N. Tien [3]. Hu and Tien viewed the lexicographical generation of n/k-combinations as "sweeping" the k 1's from the right side to the left, so that the initial sequence is $0^{n-k}1^k$ and the terminating sequence is 1^k0^{n-k}. To generate all permutations of the multiset $0^{n_0} \ldots t^{n_t}$ the n_t t's are interwoven into each permutation of the symbols 0, ..., t–1, in such a fashion that if the latter are treated as one symbol α, with ordering $\alpha < t$, then all combinations composed of $\alpha's$ and t's are generated alternately in increasing and decreasing lexicographically order for successive permutations of the symbols 0, ..., t–1. The permutations for the symbols 0, ..., t–1 are similarly generated.

Figure 2.03 shows how the permutations of { 0, 1, 2, 2 } are generated by Algorithm 3.

From the brief description above it is obvious that the action of sweeping the t's from one side to the other has the same complexity of binary combination generation with n_t 1's and $n-n_t$ 0's, where $n = \sum_{i=0}^{t} n_i$. From [11] the complexity for each complete sweep is $\binom{n+2}{n_t+1}$.

$$0122 \rightarrow 0212 \rightarrow 0221 \rightarrow 2012 \rightarrow 2021 \rightarrow 2201$$

$$\rightarrow 2210 \rightarrow 2120 \rightarrow 2102 \rightarrow 1220 \rightarrow 1202 \rightarrow 1022$$

Figure 2.03

One complete sweep is executed for each permutation of the symbols 0, , t-1, which number $\begin{bmatrix} n-n_t \\ n_0 \quad n_{t-1} \end{bmatrix}$ Therefore as in the case of Algorithm 1, the complexity of the algorithm is at least of the order

$$\begin{bmatrix} n-n_t \\ n_0 \quad n_{t-1} \end{bmatrix} \begin{bmatrix} n+2 \\ n_t+1 \end{bmatrix} = \begin{bmatrix} n \\ n_0 \quad n_t \end{bmatrix} \frac{(n+1)(n+2)}{(n-n_t+1)(n_t+1)}$$

Hence in general Algorithm 3 is not constant average time

2.1.4. Algorithm 4 (Ruskey & Roelants van Baronaigien)

This algorithm is due to F Ruskey and D Roelants van Baronaigien [4] and generates all permutations of a multiset lexicographically by recursion The basic idea is that if $perm(0^{n_0}, \quad, t^{n_t})$ denotes all permutations of the multiset $0^{n_0} \quad t^{n_t}$ then

$$perm(0^{n_0}, \quad, t^{n_t}) = \bigcup_{i=0}^{t} i\text{–}perm(0^{n_0}, \quad, t^{n_i-1}, \quad, t^{n_t})$$

where i-$perm(0^{n_0}, \quad, t^{n_t})$ denotes all sequences formed by concatenating the symbol i with all permutations of the multiset $0^{n_0} \quad t^{n_t}$ Figure 2.04 shows the order in which the permutations of { 0, 0, 1, 2 } are generated

2100 → 2010 → 2001 → 1200 → 1020 → 1002

→ 0210 → 0201 → 0120 → 0102 → 0021 → 0012

Figure 2 04

Ruskey and Roelants van Baronaigien showed in their paper that if the symbols are ordered so that $n_0 > n_1 >$ $> n_t$, then their algorithm generates all permutations of a multiset in constant time per permutation on average, independent of n and t.

2.1.5 Algorithm 5 (Chase)

A desirable property in the generation of multiset permutations is to have successively generated permutations differing as little as possible, for example, to have successive permutations differing in exactly two positions. This type of generation is referred to as generation by interchange in this thesis, whereas it is called generation by transposition by P. J. Chase.

Algorithm 5 was called **Extended Twiddle** by Chase and was first published as Alg. 383 in CACM [5]. It is a generalization of algorithm **Twiddle**, published as Alg. 382 in CACM [12], which generates all n/k-combinations by interchange. The proof of correctness of these two algorithms was given by Chase in [13].

To understand Algorithm 5, one must first examine **Twiddle**. Suppose there are n symbols in all, k of them 1's, the rest 0's. **Twiddle** treats each combination as a sign sequence which is a sequence with k 1's and n-k signs, + or -. Certain intervals in a sign sequence are called blocks. An R-block is an interval of +'s which is followed by a number but which

is not preceded by a +. An L-block is an interval of signs of which the last, and only the last, is a -, and which is preceded by a number. The successor of a sign sequence s which has at least one block is a sign sequence σ obtained as follows. Find the leftmost block in s. If it is an R-block, then slide the entire block one place to the right, putting the displaced number in the place vacated by the block's leftmost sign. If it is an L-block, perform the symmetric operations. In either case, if a sign slides, then the next sign to the left, if there is one, changes. To obtain an n/k-combination, simply replace all signs in a sign sequence by 0's. Figure 2.05 shows how **Twiddle** generates the combinations for n=5 and k=2. The shifting blocks are underlined.

In [13] P. J. Chase showed that with the initial sequence $+^{n-k}1^{n_1}$ all combinations can be generated by interchange. To generate permutations for the multiset $0^{n_0}\ldots t^{n_t}$, consider the following extension:

1. Use **Twiddle** to generate all combinations of $(t-1)^{n_{t-1}}t^{n_t}$. If t is 1, quit.

2. For each combination p of $(t-1)^{n_{t-1}}t^{n_t}$, consider the symbols t-1 and t as one symbol α and generate recursively all permutations of the multiset $0^{n_0}\ldots(t-2)^{n_{t-2}}\alpha^{n_{t-1}+n_t}$, replacing

---

$$\underline{+\,+\,+}\;1\;1 \rightarrow 1\;\underline{-}\;-\;+\;1 \rightarrow -\;1\;\underline{-}\;+\;1 \rightarrow +\;-\;1\;\underline{+}\;1 \rightarrow \underline{+\,+}\;1\;1\;+$$

$$\rightarrow 1\;\underline{-}\;+\;1\;+ \rightarrow -\;1\;\underline{+}\;1\;+ \rightarrow \underline{+}\;1\;1\;+\;+ \rightarrow 1\;\underline{+}\;1\;+\;+ \rightarrow 1\;1\;+\;+\;+$$

Figure 2.05

---

each α subsequence by p Let p_1, p_2, ... , p_j be the list of combinations generated in step 1 Let $C(s)$ be the list of permutations of 0^{n_0} $(t-2)^{n_{t-2}}\alpha^{n_{t-1}+n_t}$ generated recursively, and $C^R(s)$ the reverse list, where the α subsequence is replaced by the sequence s Then the complete list generated is $C(p_1)C^R(p_2)$ ... $C(p_j)$ (or $C^R(p_j)$, depending on the parity of j) This alternate ordering is necessary to preserve the interchange property

Figure 2 06 shows the order in which the permutations of { 0, 1, 1, 2, 2 } are generated

Twiddle must be analyzed first in order to obtain a lower bound for the complexity of Algorithm 5 In each step of **Twiddle**, to find the leftmost block, the leftmost block of signs must be examined always This corresponds to scanning the initial subsequence of 0's Then if the rightmost sign of this block is -, the next block of 1's must also be scanned At this point we have to make note of the following property of the sign sequence

Observation 2 1 *Let* t *be a sign sequence obtained from the initial sequence* s *after at least one step If in* t *the* j^{th} *0 (j = 2, ,n-k) is displaced an odd (resp even) number of places*

01122 → 10122 → 11022 → 11202 → 11220

→ 21120 → 21102 → 21012 → 20112 → 02112

→ 01212 → → 02211

Figure 2 06

---

from its position in s, *then the* $(j-1)^{st}$ *sign is different (resp the same) in* t *and* s *(See* [13] *)*

In the binary case, the initial sequence is always $+^{(n-k)}1^k$ Thus to determine the rightmost sign in the leftmost block of signs, one only needs to know the parity of the size of the block of 1's immediately after the leftmost block of signs By summing over all sizes of the leftmost sign block (say i) and all sizes of the 1-block immediately following (say j) a lower bound on the complexity of this algorithm, denoted by T(Alg5), is given by the following expression, where k is assumed to be odd without loss of generality

$$
\begin{aligned}
& T(Alg5) \\
\geq & \sum_{\substack{j=1 \\ j \text{ odd}}}^{k} \sum_{i=0}^{n-k-1} (j+1)\binom{n-i-j-1}{k-j} + \sum_{j=1}^{k} \sum_{i=0}^{n-k-1} i\binom{n-i-j-1}{k-j} \\
= & \sum_{\substack{j=1 \\ j \text{ odd}}}^{k} (j+1)\binom{n-j}{k-j+1} + \binom{n}{k+1} - (n-k) \\
> & \sum_{\substack{j=0 \\ j \text{ even}}}^{k-1} \binom{n-j}{n-k} - k + \binom{n}{k+1} - (n-k) \\
> & \frac{1}{2} \sum_{j=0}^{k-1} \binom{n-j}{n-k} + \binom{n}{k+1} - n \\
= & \frac{1}{2}\binom{n+1}{k} + \binom{n}{k+1} - n - 1
\end{aligned}
$$

In the multiset case, each permutation of $1^{n_1} \ldots t^{n_t}$ is treated as a sequence of one symbol and combined with n_0 0's using **Twiddle** Then clearly the complexity is at least

$$
\binom{n-n_0}{n_1 \ \ldots \ n_t}\left[\frac{1}{2}\binom{n+1}{k} + \binom{n}{k+1}\right] = \binom{n}{n_0 \ n_1 \ \ldots \ n_t}\left[\frac{n+1}{2(n_0+1)} + \frac{n_0}{n-n_0+1}\right]
$$

Therefore in general Algorithm 5 is not constant average time.

2.1.6. Algorithm 6 (Lam & Soicher)

This algorithm is due to C. W. H. Lam and L. H. Soicher [14]. It does not use the binary representation for combinations. Recall that an n/k-combination is a subset of size k of {1, ..., n}. Thus a combination can also be represented by an increasing sequence of size k such that the integer i is in the sequence if and only if i is in the subset. Algorithm 6 generates all increasing sequences (of length k) of integers $\leq$ n.

This algorithm is a slight variation of the usual recursive method that generates combinations in reverse lexicographical order. The order in which the i^{th} element is generated depends on the parity of the integer i. If i is even increasing order is used, decreasing order otherwise. In the following let AL6 denote Algorithm 6 and let the combination sequence be stored in $a[1], \ldots, a[k]$.

AL6(i)

1 If i is even then for $j = i, \ldots, a[i+1]-1$ do the following: $a[i] \leftarrow j$ and AL6(i-1).

2 If i is odd then for $j = a[i+1]-1, \ldots, i$ do the following: $a[i] \leftarrow j$, if i=1 then PROCESS else AL6(i-1).

PROCESS signals that a new combination has been generated. Initially the value n+1 is assigned to $a[k+1]$ and AL6(k) is called to generate all combinations.

Figure 2.07 shows how the size 4 combinations of { 1, 2, 3, 4, 5, 6 } (the binary representations are in parentheses) are generated.

1234 (111100)
1245 (110110)
2345 (011110)
1345 (101110)

1346 (101101)
1236 (111001)

Figure 2.07

By using a stack Lam and Soicher obtained a nonrecursive version of this algorithm. The worst case complexity for each combination of the nonrecursive version is bounded by a constant. (See [14].)

2.1.7. Algorithm 7 (Bitner, Ehrlich & Reingold)

This algorithm is due to J.R. Bitner, G. Ehrlich and E.M. Reingold [6] and is based on binary reflected Gray codes. A binary Gray code is an ordering of all binary sequences of fixed length n such that successive sequences differ in exactly one position. Denote the binary reflected Gray code for sequences of length n by $G(n)$ and the same Gray code in reverse order by $G(n)^R$. $G(n)$ has the following recursive definition.

$$G(n) = \begin{bmatrix} 0G(n-1) \\ 1G(n-1)^R \end{bmatrix}$$

$$G(1) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The list of combinations Algorithm 7 generates is equivalent to the binary reflected Gray code except all binary sequences without the correct number of 1's are deleted Let $C(n,k)$ denote the sequence of n/k-combinations generated by this algorithm, and $C(n,k)^R$ denote the same sequence listed in reverse order Then a recursive definition similar to that for $G(n)$ can be given

$$C(n,k) = \begin{bmatrix} 0C(n-1,k) \\ 1C(n-1,k-1)^R \end{bmatrix}$$

$$C(n,0) = 0^n$$

$$C(n,n) = 1^n$$

The list $C(n,k)$ always begins with the sequence $0^{n-k}1^k$ and ends with the sequence $10^{n-k}1^{k-1}$ Inductively it is easy to see that $C(n,k)$ has the interchange property

The recursive definition of $C(n,k)$ gives rise naturally to a recursive algorithm that generates the list. Algorithm 7 is similar to Algorithm 6 in that the i^{th} element of the combination is generated alternately in increasing and decreasing lexicographical order (of 0 and 1) For example in the expansion of $C(n,k)$, shown in Figure 2 08, the second bit is generated first from 0 to 1 and then from 1 to 0 The recursion tree can be simplified by replacing $C(n,k)$ with + and the reverse list by - at each node Thus a + would indicate that the children of the current node are to be generated in increasing order, and a - would indicate the opposite The complete generation of the combinations of { 0, 0, 1, 1 } by Algorithm 7 is shown in Figure 2 09

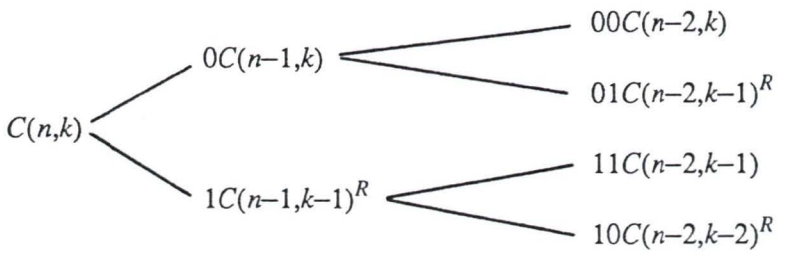


Figure 2 08

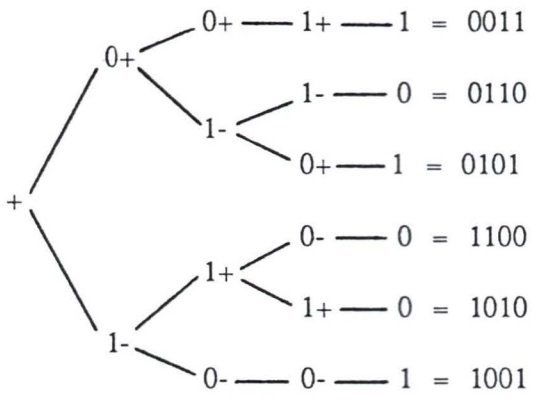


Figure 2 09

Bitner, Ehrlich and Reingold found an iterative equivalent of this algorithm It can be shown that the iterative version always generates each combination in time bounded by a constant [15]

2.2 A New MPG Algorithm

The new mpg algorithm, MPerm, satisfies two properties: (1) generation by interchange, (2) the generation is constant average time. This algorithm is closely related to Algorithms 6 and 7 in that the lexicographical order in which an element in the sequence is generated alternates. In fact when restricted to the binary case the recursion tree of MPerm and that of Algorithm 7 are the same.

Extending the definition of $C(n,k)$ to the multiset case, let $C(n_0, \quad ,n_t)$ denote the list of multiset permutations generated by MPerm, and $C(n_0, \quad ,n_t)^R$ denote the reverse list. Then, w.l.o.g. assuming t is odd,

$$C(n_0, \quad ,n_{i-1},0,n_{i+1}, \quad ,n_t) = C(n_0, \quad ,n_{i-1},n_{i+1}, \quad ,n_t) \tag{2.1}$$

$$C(n_0, \quad ,n_t) = \begin{bmatrix} 0C(n_0-1,n_1, \quad ,n_t) \\ 1C(n_0,n_1-1,n_2, \quad ,n_t)^R \\ 2C(n_0,n_1,n_2-1, \quad ,n_t) \\ \\ \\ tC(n_0, \quad ,n_{t-1},n_t-1)^R \end{bmatrix} \tag{2.2}$$

$$C(n_i) = i^{n_i} \tag{2.3}$$

Note that when the parameter list $n_0, \quad ,n_t$ is contracted as in (2.1) because some n_i is 0, none of the subscripts i, which also indicate the symbol, are changed. Figure 2.10 shows the recursion tree with which the permutations of the multiset { 0, 1, 2, 2 } are generated.

Algorithm MPerm is derived from the recursive definition. To keep a record of the lexicographical order that is used at each level, a global array *dir* is needed. The value of

4 Whenever $dir[\imath]$ is updated, $M[\imath+1] \leftarrow M[\imath+1] + M[\imath]$ and $M[\imath]$ is reset to 0

For example, let p_1, p_2, p_3 and p_4 be four successively generated permutations. Suppose before p_1 is generated $M[\imath]$ is 0 for all $\imath$. Let p_2 contain a 0-path from level $j+1$ on and p_3 contain a 0-path from level j on. The effect on the arrays M and dir when the tree is traversed to generate the permutations is shown in Figure 2.11.

NOTE. It is not neccessary to keep the count of the number of nodes suppressed at a level. The parity of the count suffices to determine the value of dir.

The result of these modifications to Algorithm MPerm is Algorithm 8. Two new global variables are added. Integer *skiphere* is used to store $\sum_{j=1}^{\imath} M[j]$ where $\imath$ is the current level. Integer array *skipmem* is used in place of M. The initial call is Alg8(0) with all elements of *skipmem* and *skiphere* set to 0.

Lemma 2.4. *Algorithm 8 executes in constant time on average.*

Proof. Quite clearly Algorithm 8 still satisfies conditions 1 and 2. To show that it also satisfies condition 3, first count the number of degree one nodes. There is a node of degree one for every symbol $\imath \neq 0$ in a sequence such that either (a) it is at the end of the sequence, or (b) the remainder of the sequence is an $\imath$-path. Using the multinomial identities

$$\sum_{\imath=0}^{n_0} \binom{n+\imath}{\imath\ n_1 \cdots n_t} = \frac{n_0+n+1}{n+1} \binom{n_0+n}{n_0\ n_1 \cdots n_t}$$

and

$$\sum_{\imath=0}^{t} \binom{n-1}{n_0 \cdots n_{\imath-1}\ n_\imath-1\ n_{\imath+1} \cdots n_t} = \binom{n}{n_0 \cdots n_t}$$

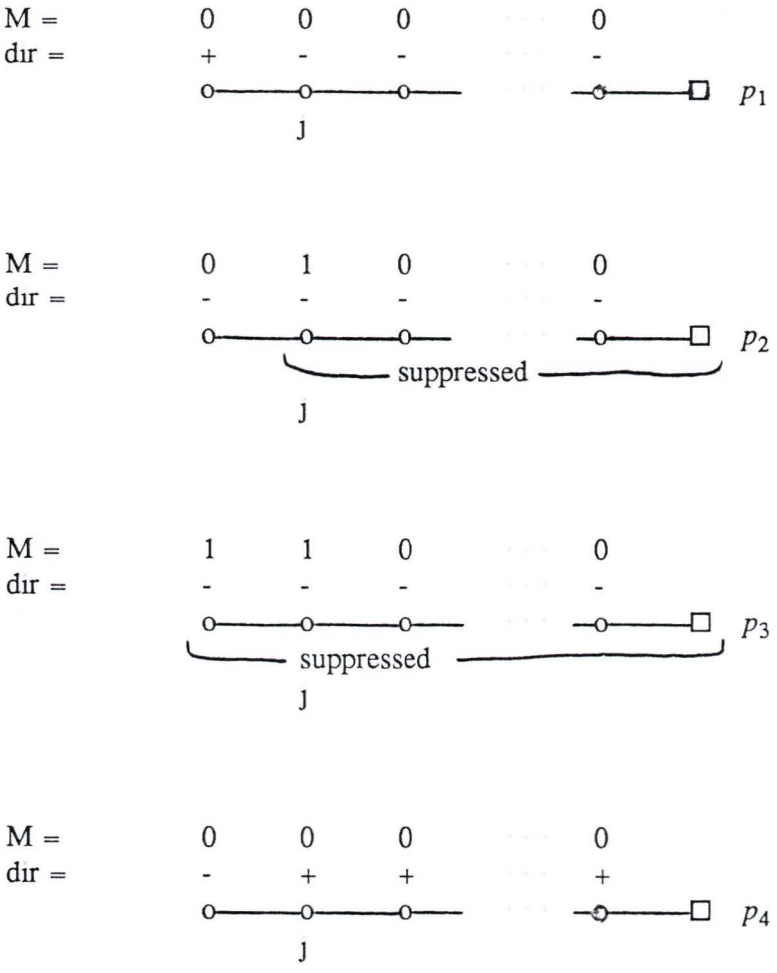

M =
dır =
0 0 0 0
+ - - -
J
p_1
M =
dır =
0 1 0 0
- - - -
suppressed
J
p_2
M =
dır =
1 1 0 0
- - - -
suppressed
J
p_3
M =
dır =
0 0 0 0
- + + +
J
p_4

Figure 2.11

```
Alg8( level : integer ),

BEGIN
  IF level = n-1
    THEN BEGIN
      a[n] ← whatever symbol remaining,
      output a,
      skiphere ← 0,
      a[n] ← 0,
    END
    ELSE BEGIN
      skiphere ← skiphere + skipmem[level],
      if skiphere is even then
        dir[level] ← reverse of dir[level],
      skipmem[level] ← 0,

      IF 0 is the only symbol to be placed
      THEN BEGIN
        increment skiphere,
        add skiphere to skipmem[level+1],
        skiphere ← 0,
        output a,
      END
      ELSE BEGIN
        in the order indicated by dir[level]
        LOOP through i ∈ { i : n_i > 0 }
        BEGIN
          n_i ← n_i - 1,
          a[level+1] ← i,
          Alg8( level+1 ),
          n_i ← n_i + 1,
          a[level+1] ← 0,
        END,
      END,
    END,
END
```

Algorithm 8

and summing over all j's, where j is the length of the i-path, the number of such occurrences is

$$\sum_{i=1}^{k}\sum_{j=1}^{n_i}\begin{bmatrix} & & n-j & & \\ n_0 & n_{i-1} & n_i-j & n_{i+1} & n_t \end{bmatrix}$$

$$= \sum_{i=1}^{k}\sum_{j=0}^{n_i-1}\begin{bmatrix} n-n_i+j \\ j \end{bmatrix}$$

$$= \sum_{i=1}^{k}\frac{(n-n_i)+(n_i-1)+1}{(n-n_i)+1}\begin{bmatrix} (n-n_i)+(n_i-1) \\ n_i-1 \end{bmatrix}$$

$$= \sum_{i=1}^{k}\frac{n}{n-n_i+1}\begin{bmatrix} n-1 \\ n_i-1 \end{bmatrix}$$

$$\leq \frac{n}{n-n_1+1}\sum_{i=1}^{k}\begin{bmatrix} n-1 \\ n_i-1 \end{bmatrix}$$

$$< \frac{n}{n-n_1+1}\begin{bmatrix} & n & \\ n_0 & & n_t \end{bmatrix}$$

$$\leq 2\begin{bmatrix} & n & \\ n_0 & & n_t \end{bmatrix}$$

Consider the tree with all degree one nodes removed The resulting tree has the same number of leaf nodes as the former tree and all its internal nodes have minimum degree 2 A simple proof by induction can show that the number of internal nodes in this tree is less than the number of leaf nodes Summarizing all the results we have the number of all nodes in the recursion tree for Algorithm 8 is less than four times the number of leaf nodes, i e less than $4\begin{bmatrix} & n & \\ n_0 & & n_t \end{bmatrix}$ Therefore condition 3 is satisfied and Algorithm 8 on average generates each permutation in constant time

2.3 Ranking and Unranking Algorithms in the Binary Case

Ranking and unranking algorithms are associated with generation algorithms. For an algorithm A that generates all objects in a set S, a ranking algorithm takes as input an object O in S and returns the number of objects before O in the list generated by A. An unranking algorithm performs the inverse operation of a ranking algorithm. For example let A be an algorithm that generates all combinations lexicographically, R the ranking algorithm for A, and U the unranking algorithm. A generates all combinations with two 0's and two 1's in the order

$$0011 \rightarrow 0101 \rightarrow 0110 \rightarrow 1001 \rightarrow 1010 \rightarrow 1100$$

$R(0011)$ gives 0, $R(1010)$ gives 4, and $U(3)$ gives 1001.

In the following ranking and unranking algorithms for Alg. 8 in the binary case are given. Since Algorithm 8 generates binary combinations in the same order as Bitner's algorithm (Alg. 7), the ranking and unranking algorithms also work for the latter.

2.3.1 Ranking Algorithm

For both ranking and unranking operations we need to know the lexicographical order used at each node. In other words to rank or unrank the element $a[i+1]$, where a is a permutation, we need to know the value of $dir[i]$. Initially (for the permutation $0^{n_0}1^{n_1}$) all $dir[i]$ are set to +. To determine the value of $dir[i]$ along a branch in the recursion tree, we must know the parity of the number of nodes at level i before the current node. For example in

Figure 2 12 the number of nodes at level 5 before node * is the following sum

$$\begin{aligned} &\text{the number of nodes at level 4 in } T_1 \\ +\ &\text{the number of nodes at level 3 in } T_2 \\ +\ &\text{the number of nodes at level 2 in } T_3 \end{aligned}$$

Let S_i denote the number of nodes at level i in the recursion tree for $C(n,k)$

Lemma 2.5 *The parity of* S_i = *the parity of* $\binom{i-1}{k}+\binom{i-1}{n-k}$, $i \geq 1$

Proof At level i let A be the set of nodes with no siblings and B be the set with siblings

Clearly $|B|$ is even Therefore the parity of the number of nodes at level i depends on $|A|$

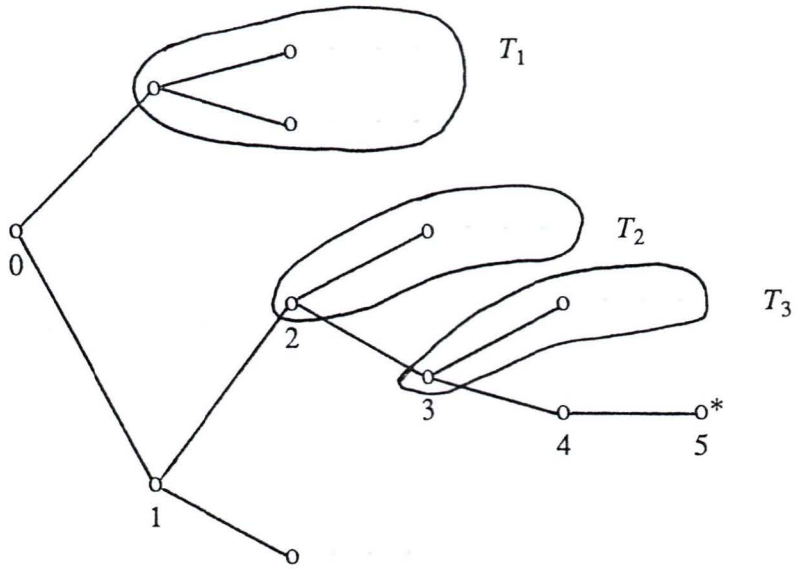


Figure 2.12

only Define the prefix of a node x in the tree as the sequence obtained by recording the symbol generated at each node while traversing from the root to the parent of the node x If a node x is in A then in the prefix (of lenght i-1) of x, either all 0's or all 1's are used There are $\binom{i-1}{k}+\binom{i-1}{n-k}$ such nodes

■

Using this lemma it is easy to determine the value of *dir[i]* For example if *a[1]* is 0 and the lexicographically first symbol is 1 (i e *dir[0]* is -) then the subtree of all combinations beginning with 1 is omitted Lemma 2 5 can then be used to adjust the parities of *dir[1] dir[n]* When the branch representing the combination in the recursion tree is traversed, the parities of *dir* can be similarly updated, as shown in Figure 2 12

The rest of the algorithm is very much the same as the usual ranking algorithm for the lexicographical generation of n/k-combinations, where if *a[1]* is the lexicographically first symbol (0), 0 is added to the rank, and otherwise the number of n/k-combinations beginning with 1 is added to the rank The combination stored in *a[2] a[n]* is recursively ranked, and the final accumulated rank is the rank for the combination

For Algorithm Rank, the combination is stored in the array *a[1] a[n]* initially, and *dir[i]* set to +, $i = 0, \ldots, n$ A Pascal version of the algorithm is given in Appendix A 2

For j=1, ,n–1, the FOR loop in Algorithm Rank executes n–j-1 times Everything else is performed in constant time The binomial coefficient table can be constructed in $O(n^2)$ time Therefore the Ranking procedure has time complexity $O(n^2)$

```
Alg  Rank( j, n, k   integer ),
  returns an integer,

  { n is the size of the combination }
  { k is the number of 1's in the combination }
  { j is the index of the element of a being examined }
  { B(i,j) is the binomial coefficient "i choose j" }
  { m is the end of the combination in array a }

BEGIN
  IF n=k OR k=0 THEN return Rank ← 0
  ELSE BEGIN
    IF (( a[j] is 0 AND dir[j-1] is + )
       OR ( a[j] is 1 AND dir[j-1] is - ))
      THEN first ← true
      ELSE first ← false,

    IF a[j] is 1 THEN BEGIN   k1 ← k, k ← k-1,   END
                      ELSE k1 ← k-1,

    k0 ← n-1 - k1,

    IF first
    THEN
      return Rank ← Rank( j+1, n-1, k )
    ELSE BEGIN
      invert dir[j],
      FOR l = j+2, ,m DO
        IF B(l-j-2,k0)+B(l-j-2,k1) is even THEN invert dir[l-1],
      return Rank ← B(n-1,k1) + Rank( j+1, n-1, k-1 ),
    END,
  END,
END
```

Ranking Algorithm for Algorithm 8

```
Alg  Unrank( rank, j, n, k   integer ),

  { rank is the rank of the combination to be found }
  { j is the index of the element to be assigned }
  { n is the size of the combination }
  { k is the number of 1's in the combination }
  { m is the end of the combination in array a }

BEGIN
  IF n=k THEN a[j] to a[m] are all set to 1
  ELSE IF k=0 THEN a[j] to a[m] are all set to 0
  ELSE BEGIN

    IF dir[j-1] is +
    THEN BEGIN
      first ← 0, k1 ← k, k ← k-1,
    END
    ELSE BEGIN
      first ← 1, k1 ← k-1,
    END,
    k0 ← n-1 - k1,

    IF B(n-1,k1) > rank
    THEN BEGIN
      a[j] ← first,
      Unrank( rank, j+1, n-1, k1 ),
    END
    ELSE BEGIN
      a[j] ← 1 - first,
      invert dir[j],
      FOR l = j+2,  ,m-1 DO
        IF B(l-j-2,k0)+B(l-j-2,k1) is even THEN invert dir[l-1],
      Unrank( rank-B(n-1,k1), j+1, n-1, k ),
    END,
  END,
END
```

Unranking Algorithm for Algorithm 8

2.3.2. Unranking Algorithm

The unranking algorithm is slightly modified from the usual unranking algorithm for lexicographical generation of combinations and is also very similar to the ranking algorithm discussed above. To find *a[1]*, the value of the rank is checked. Let m be the number of combinations beginning with the lexicographically first symbol. If *rank* is less than m, then *a[1]* is the lexicographically first symbol and m is subtracted from *rank*. Otherwise *a[1]* is the second symbol. The values of *dir* are then updated as in the ranking algorithm and the rest of the combination is unranked recursively.

Initially the array *dir* is set up as in the ranking case. The combination is returned in the array *a*. As in the ranking case, the unranking procedure has time complexity $O(n^2)$. A Pascal version is given in Appendix A.3.

CHAPTER 3

IMPOSSIBILITY OF MPG BY ADJACENT INTERCHANGE

It was shown in Chapter 2 that all permutations of a multiset can be generated by an interchange algorithm. A more severe restriction may be imposed on the algorithm so that two consecutive permutations must differ in two **adjacent** positions. An algorithm with this property is called a generation algorithm by adjacent interchange. However with this restriction it may not always be possible to generate all permutations.

For the multiset $0^{n_0} \ldots t^{n_t}$ define a graph $G(n_0,n_1, \ldots ,n_t)$ where the vertices are all the permutations of the multiset, and an edge (p_1,p_2) is in the graph if and only if the permutations p_1 and p_2 differ by an adjacent interchange. Then for a multiset it is possible to have a generation algorithm by adjacent interchange if and only if there is a Hamilton path in the corresponding graph. For example the graph $G(2,2)$ in Fig. 3.1 does not have a Hamilton path and so it is impossible to have an adjacent interchange algorithm for the multiset { 0, 0, 1, 1 }.

To find an adjacent interchange algorithm for multisets it is important to know when it is possible for such an algorithm to exist. This chapter examines and answers this question.

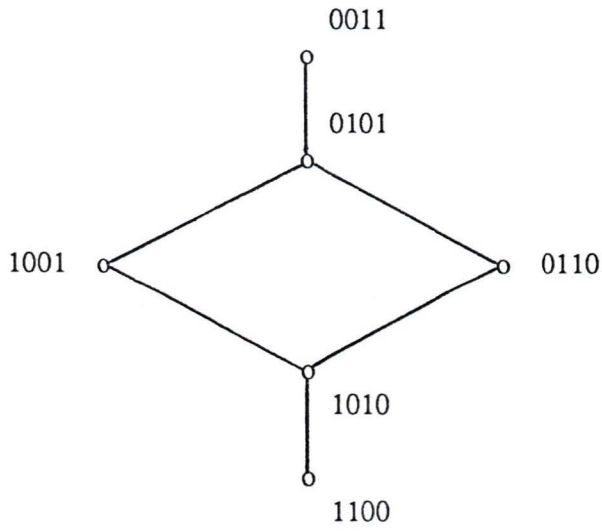


Figure 3.1

3.1 Previous Research

3.1.1 Binary Combinations

For binary combinations this problem has been studied and solved by Ruskey and Miller [7], Buck and Wiedemann [8], and Eades, Hickey and Read [9] independently. The necessary condition on n_0 and n_1 for the existence of a Hamilton path in $G(n_0,n_1)$ is given in Lemma 3.1.

Lemma 3.1 *For n_0, $n_1 \geq 2$ no Hamilton path in $G(n_0,n_1)$ exists if either n_0 or n_1 is even.*

This result was only stated by Buck and Wiedemann, but was proved by Ruskey and Miller and Eades, Hickey and Read. Both proofs are based on the observation that $G(n_0,n_1)$

is bipartite

Observation 3.2 *$G(n_0,n_1)$ is bipartite*

Proof For each combination define a score as follows. Let C be a combination and let c_i be the i^{th} bit in C. The score of C is the sum $\sum_{i=1}^{n} ic_i$, where n is the sum of n_0 and n_1. Partition the vertices of $G(n_0,n_1)$ into two sets O and E

O = { all combinations with odd scores }

E = { all combinations with even scores }

If two combinations differ by an interchange of adjacent integers clearly their scores differ by exactly 1. Therefore no two combinations from the same set, O or E, can be adjacent in $G(n_0,n_1)$. The observation follows.

■

Let $D = |E| - |O|$. Since $G(n_0,n_1)$ is bipartite, if it is to have a Hamilton path $|D|$ must be ≤ 1. Let $D(n_0,n_1)$ denote D, the parity difference of the two partitions for $G(n_0,n_1)$. Then by calculating $D(n_0,n_1)$ it can be determined whether or not it is possible for a Hamilton path to exist in $G(n_0,n_1)$. The values $D(n_0,n_1)$ satisfy recurrence relation (3.1) in the following lemma.

Lemma 3.3 *$D(0,0) = 1$ and if $n_0, n_1 \geq 0$ then*

$$D(n_0,n_1) = (-1)^{n_1} (D(n_0,n_1-1) + D(n_0-1,n_1)) \qquad (3.1)$$

(See [7].)

Ruskey and Miller observed that the values of $D(n_0,n_1)$ satisfy expression (3 2) in Lemma 3 4 which they proved inductively using recurrence relation (3 1)

Lemma 3 4

$$D(n_0,n_1) = s\left(\left\lfloor\frac{n}{2}\right\rfloor, \left\lfloor\frac{n_1}{2}\right\rfloor\right) \quad \textit{where } s = \begin{cases} 0 & \text{if } n_0 \textit{ odd}, n_1 \textit{ odd} \\ (-1)^{\left\lceil\frac{n_1}{2}\right\rceil} & \textit{otherwise} \end{cases} \tag{3 2}$$

Eades, Hickey and Read also discovered expression (3 2) by way of generating functions They considered the problem of enumerating all combinations with score N The i*th* bit in a combination, if it is 1, contributes 1 to the integer n_1 and i to the integer N If it is 0, it contributes 0 to both Hence the solution is the coefficient of $x^{n_1}y^N$ in the generating function

$$F(x,y) = \prod_{r=1}^{n}(1+xy^r) \tag{3 3}$$

Now $|O|$ is the sum of those coefficients of $x^{n_1}y^N$ for which N is odd, while $|E|$ is the sum of those such coefficients for which N is even It follows that the generating function for $D(n_0,n_1)$ is obtained at once by putting y = -1 in (3 3)

If n is even, say n = 2m, this generating function is $(1-x^2)^m$, and

$$D(n_0,n_1) = \textit{coefficient of } x^{n_1} \textit{ in } (1-x^2)^m$$

$$= \begin{cases} 0 & \text{if } n_1 \textit{ is odd,} \\ (-1)^{\frac{n_1}{2}} \begin{bmatrix} m \\ \frac{n_1}{2} \end{bmatrix} & \text{if } n_1 \textit{ is even} \end{cases}$$

If n is odd, say n = 2m + 1, the generating function is $(1-x^2)^m(1-x)$, and

$$D(n_0,n_1) = \textit{coefficient of } x^{n_1} \textit{ in } (1-x^2)^m(1-x)$$

$$= \begin{cases} (-1)^{\frac{n_1}{2}} \begin{bmatrix} m \\ \frac{n_1}{2} \end{bmatrix} & \text{if } n_1 \textit{ is even,} \\ (-1)^{\frac{n_1+1}{2}} \begin{bmatrix} m \\ \frac{n_1-1}{2} \end{bmatrix} & \text{if } n_1 \textit{ is odd} \end{cases}$$

It is easy to see that these expressions concur with expression (3 2) exactly

If either n_0 or n_1 is 1 then a Hamilton path trivially exists If n_0, $n_1 \geq 2$, then n ≥ 4 and the binomial coefficient $\begin{bmatrix} \lfloor \frac{n}{2} \rfloor \\ \lfloor \frac{n_1}{2} \rfloor \end{bmatrix}$ is always at least 2 Thus unless both n_0 and n_1 are odd, $G(n_0,n_1)$ cannot have a Hamilton path

In addition to giving conditions on the impossible cases, Ruskey and Miller gave a constant average time generation algorithm for all other cases, thus demonstrating directly that n_0, n_1 odd is a necessary and sufficient condition for a Hamilton path in $G(n_0,n_1)$ to exist, a result that Buck and Wiedemann and Eades, Hickey and Read also proved. Their inductive proofs give bases for recursive generation algorithms.

3.1.2. A Slightly Different Adjacency Restriction

A variant of the adjacency restriction was studied by Joichi and White [16]. In this variation the first and last positions of the combination are also considered adjacent. With this extended definition of adjacency let $G^*(n_0,n_1)$ denote the graph analogous to $G(n_0,n_1)$. Joichi and White were also concerned with "closed listings" of the combinations, i.e. the last and the first combinations in the list are to be adjacent in the graph $G^*(n_0,n_1)$. In the graphical sense, the problem is now to find a Hamilton cycle in $G^*(n_0,n_1)$.

With these variations the problem becomes far more complicated than our original problem in the combinations case. Joichi and White managed to show that

1. $G^*(n_0,n_1)$ does not contain a Hamilton cycle if one of n_0, n_1 is odd and the other is even, or if either is 2 and n is odd and ≥ 7.
2. $G^*(n_0,n_1)$ has a Hamilton cycle if k = 3, k = 4 and n is odd, or k = 5 and $n \neq 7$, where k can be either n_0 or n_1.

It is not known in other cases whether $G^*(n_0,n_1)$ has a Hamilton cycle.

3.2 Impossible Cases in the Multiset Case

This problem is solved in a manner similar to that used in the combination case. For the multiset $0^{n_0}1^{n_1} \quad t^{n_t}$ the graph $G(n_0,n_1, \quad ,n_t)$ is defined and shown to be bipartite. Letting $D(n_0,n_1, \quad ,n_t)$ denote the difference in size between the two partitions of the vertices, a recurrence relation for $D(n_0,n_1, \quad ,n_t)$ is found and solved, thus revealing when it is impossible for $G(n_0,n_1, \quad ,n_t)$ to have a Hamilton path.

3.2.1 The Underlying Graph is Bipartite

For the multiset $S = 0^{n_0}1^{n_1} \quad t^{n_t}$ define the underlying graph $G(n_0,n_1, \quad ,n_t)$ with vertex set V and edge set F, where

1. the vertices in V are exactly the permutations of S,
2. the edge (p_1,p_2), where p_1, p_2 are permutations of S, is in F if and only if one can be obtained from the other by an interchange of adjacent integers.

Lemma 3.5 *$G(n_0,n_1, \quad ,n_t)$ is bipartite.*

Proof To prove the lemma it suffices to show that there are no odd cycles in the graph. Let $C = (p_1\ p_2 \quad p_m\ p_{m+1}=p_1)$ be a cycle in the graph. Suppose the transposition taking p_1 to p_2 involves interchanging x^*, an occurrence of the symbol x, and y^*, an occurrence of the symbol y, and that in p_1 there are r_y occurrences of y to the right of x^* and there are l_x occurrences of x to the left of y^* (see figure 3.2). If in the rest of the cycle x^* and y^* are not interchanged again, then after the first transposition all permutations in this cycle, including p_{m+1}, contain less than r_y y's to the right of x^* and less than l_x x's to the left of y^*. In other

words $p_1 \neq p_{m+1}$, a contradiction. Therefore each transposition has a unique inverse in the cycle, all transpositions in this cycle can be paired with its inverse, and so the cycle must be even in length.

■

3.2.2. A Recurrence Relation for the Parity Difference

The vertices of $G(n_0,n_1, \quad ,n_t)$ can be partitioned into two sets, O and E, as follows. Let $0^{n_0}1^{n_1} \quad t^{n_t}$ be the *canonical* permutation and let it be in the set E. Define the *distance* of a permutation of the multiset as the least number of adjacent interchanges necessary to

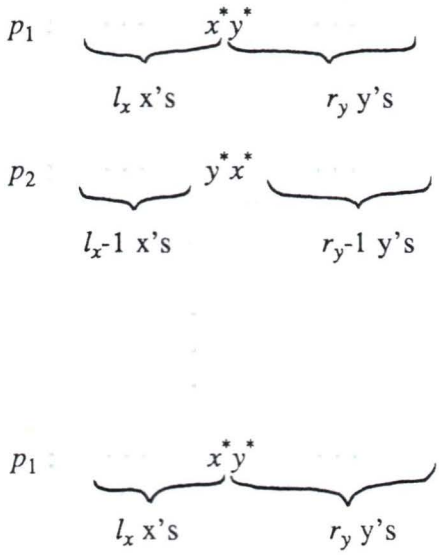


Figure 3.2

transform it into the canonical form. If the distance of a permutation is even, let the permutation be in E, if its distance is odd, let it be in O. Define the parity difference

$$D(n_0,n_1, \quad ,n_t) = |E| - |O|$$

Lemma 3.5 *A recurrence relation for the parity difference is*

$$D(1) = 1 \tag{3.4}$$

$$D(n_0, \quad ,n_{i-1},0,n_{i+1}, \quad ,n_t) = D(n_0, \quad ,n_{i-1},n_{i+1}, \quad ,n_t) \tag{3.5}$$

$$D(n_0, \quad ,n_t) = \sum_{i=0}^{t}(-1)^{\sigma_i} D(\quad ,n_i-1, \quad) \quad \textit{where } \sigma_i = \sum_{j<i} n_j \tag{3.6}$$

Proof If there is only 1 occurrence of one symbol, then $|E| = 1$ and $|O| = 0$, so (3.4) is correct. (3.5) merely says that if there are no occurrences of a symbol, ignore it. To see that (3.6) is true, consider the graphs $G = G(n_0, \quad ,n_t)$ and $G_i = G(\quad ,n_i-1, \quad)$. Let E and O be the partitions of the former, and E_i and O_i the partitions of the latter. Consider the subgraph in G induced by the set of all permutations beginning with the symbol i. This subgraph is isomorphic to G_i. If σ_i is odd, then $i0^{n_0} \quad i^{n_i-1} \quad i^{n_t}$ is in O and therefore E_i is a subset of O, while O_i is a subset of E. Conversely, if σ_i is even then E_i is a subset of E and O_i is a subset of O. Thus all permutations beginning with the symbol i contribute $(-1)^{\sigma_i} D(\quad ,n_i-1, \quad)$ to the overall parity difference $D(n_0, \quad ,n_t)$. The sum in (3.6) follows.

■

The solution of the recurrence relation is given by Theorem 3.7, where the following notation is used.

$$m_i = \left\lfloor \frac{n_i}{2} \right\rfloor, \; i = 0, \ldots, t$$

$$m = \sum_{i=0}^{t} m_i$$

$\begin{pmatrix} m \\ m_0 \; m_1 \; \ldots \; m_t \end{pmatrix}$ is denoted by $(m, m_0, \ldots, m_t)$

Theorem 3.7 *For any $t \geq 0$ and $n_i \geq 0$, $i = 0, \ldots, t$,*

$$D(n_0, n_1, \ldots, n_t) = \begin{cases} 0 & \text{if} \geq 2 \; n_i\text{'s are odd} \\ (m, m_0, m_1, \ldots, m_t) & \text{otherwise} \end{cases} \qquad (3.7)$$

Proof The proof is by induction on $n = \sum_{i=0}^{t} n_i$

For $n = 1$, clearly $D(1) = 1 \quad (0,0) = 1$ also Therefore the basis case is true

Assume that expression (3.7) is true for $n = N-1$

Let n be N Because of (3.5) assume w.l.o.g. that all n_i's ≥ 0 We have the following cases

(a) Let all n_i's be even From (3.6) we have

$$D(n_0, n_1, \ldots, n_t)$$

$$= \sum_{i=0}^{t} (-1)^{\sigma_i} D(\ldots, n_i-1, \ldots)$$

$$= \sum_{i=0}^{t} (m-1, m_0, \ldots, m_i-1, \ldots, m_t) \qquad \textit{since } n_i \textit{ is even}$$

$$= \sum_{i=0}^{t} \frac{(m-1)!}{m_0! \ldots (m_i-1)! \ldots m_t!}$$

$$= \sum_{i=0}^{t} \frac{(m-1)!m_i}{m_0! \quad m_t!}$$

$$= \frac{(m-1)!}{m_0! \quad m_t!} \sum_{i=0}^{t} m_i$$

$$= (m, m_0, \quad ,m_t)$$

(b) Let there be exactly one odd n_i, say n_j Then when k ≠ j D($,n_k-1,$) always has 2 odd n_i's, namely n_k-1 and n_j By the induction hypothesis, this parity difference is 0 and therefore

$$D(n_0, \quad ,n_t)$$

$$= (-1)^{\sigma_j} D(\quad ,n_j-1, \quad)$$

$$= (m, m_0, \quad ,m_j, \quad ,m_t) \qquad \textit{since } n_j \textit{ is odd}$$

(c) Suppose exactly 2 n_i's are odd, namely n_j and n_k, with j < k If r ≠ j or k then D($,n_r-1,$) always has at least 3 n_i's that are odd, and so by the induction hypothesis has value 0 Therefore,

$$D(n_0, \quad ,n_t)$$

$$= (-1)^{\sigma_j} D(\quad ,n_j-1, \quad) + (-1)^{\sigma_k} D(\quad ,n_k-1, \quad)$$

$$= (m, m_0, \quad ,m_j, \quad ,m_k, \quad ,m_t) - (m, m_0, \quad ,m_j, \quad ,m_k, \quad ,m_t) \qquad (3\ 10)$$

$$= 0$$

The two quantities in (3 10) are equal since n_j and n_k are both odd and $m_j = \left\lfloor \frac{n_j}{2} \right\rfloor = \left\lfloor \frac{n_j-1}{2} \right\rfloor$ and ditto for m_k

(d) Let there be more than 2 n_i's that are odd Clearly then for any ı D($,n_i-1,$) has at least 2 odd n_i's and therefore must be 0 Thus D($n_0,$ $,n_t$) ıs 0 also

■

If at least 2 n_i's > 1 the multınomıal coefficıent $\begin{bmatrix} m \\ m_0\ m_1\ \ m_t \end{bmatrix}$ ıs always > 1, ın whıch case there can be no Hamılton path ın $G(n_0,n1,$ $,n_t)$

Corollary 3.8 *If at least 2 n_i's > 1, at most 1 n_i odd, then there ıs no generatıon algorıthm by adjacent ınterchange for the multıset* $0^{n_0}1^{n_1}$ t^{n_t}

CHAPTER 4

Impossibility of Adjacent Interchange Generation of Trees

Much interest in the generation of combinatorial objects has been associated with trees, and there is an abundance of tree generation algorithms. For example [17] and [18] gave algorithms to generate binary trees lexicographically. [19] and [20] gave algorithms to generate k-ary trees. [21] discovered a constant average time algorithm to generate all ordered trees on n vertices. [22] gave an algorithm that generates binary trees by interchange. [10] showed an algorithm that generates all rooted ordered trees lexicographically. However so far no adjacent interchange algorithm has been discovered for trees. This chapter considers the problem of when it is possible for an adjacent interchange algorithm to exist.

The work in this chapter is motivated by [10], in which trees are represented by multiset sequences. In Chapter 3 it was shown when multiset permutations cannot be generated by adjacent interchange. In this chapter the idea is extended to the generation of trees represented by these sequences.

4.1. Representations of Trees

The trees we are concerned with here are *rooted ordered* trees. A tree is rooted if a distinct vertex u is designated the *root*. The tree is also ordered if there is an ordering on the subtrees rooted at the children of u. For example, the trees T_1 and T_2 in Figure 4.1 are isomorphic rooted trees, both rooted at vertex u, but are different rooted ordered trees. An

ordered forest is one in which there is an ordering on the trees. In the remainder of this chapter, unless otherwise specified, all trees are understood to be rooted ordered trees, and all forests ordered forests.

There are many ways of constructing integer sequences to represent trees uniquely. The following are typical examples.

Level Sequence. The level number of a vertex u is 1 if it is the root, or it is one greater than the level number of u's parent. For example the level numbers of the vertices in T_1 in Figure 4.1 are 1 for u, 2 for v and 3 for w. A tree T can be uniquely represented by its level sequence obtained by traversing T in preorder and recording the level numbers of the nodes. See Figure 4.2. [21] gave an algorithm that generates all such sequences corresponding to rooted trees on n vertices.

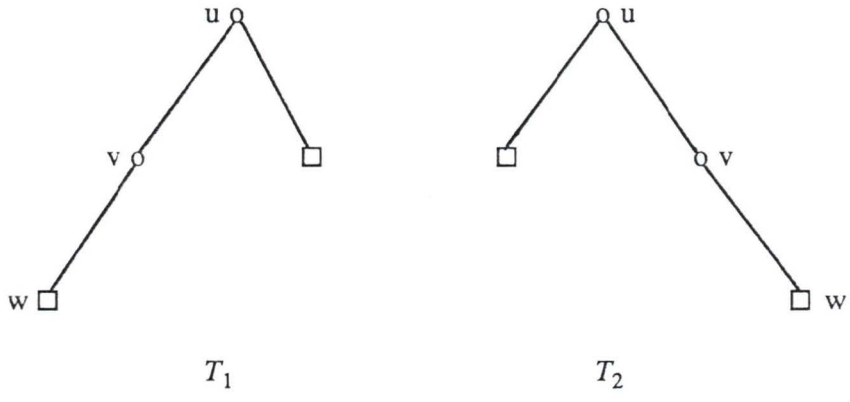
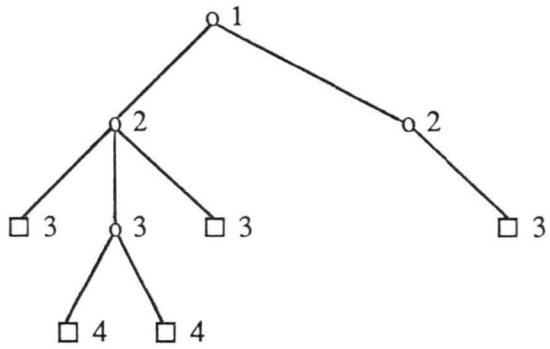


Figure 4.1 Two different rooted ordered trees

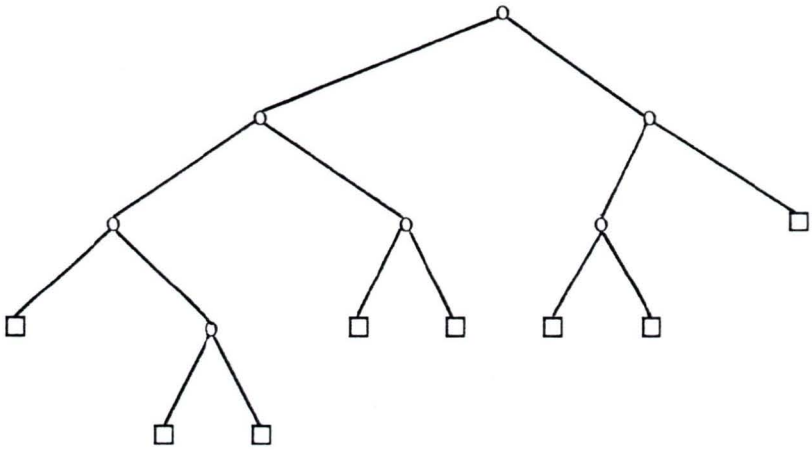


Level Sequence of this tree 123344323

Figure 4 2 Tree Representation by Level Sequence

Well-formed Parenthesis Strings Label each internal node of a binary tree B with (, and each leaf node with) The sequence obtained by traversing *B* in preorder and recording the symbol at each node except at the last leaf node is a unique representation of B The representation sequence of each tree corresponds to a well-formed parenthesis string -- in any prefix of the string there are at least as many ('s as)'s An integer sequence can be derived from a parenthesis string by replacing ('s with 1's and)'s with 0's See Figure 4 3 [22] used these sequences to represent binary trees and gave an algorithm that generates all such sequences by interchange

The representation for trees used in this chapter is the same as the one used in [10], namely each node of the tree is labelled with the number of children it has, and the representation sequence is obtained by traversing the tree in preorder and recording the label at each



Well-formed Parenthesis String ((( ) ( )) ( )) (( ))
Corresponding Binary Sequence 11101001001100

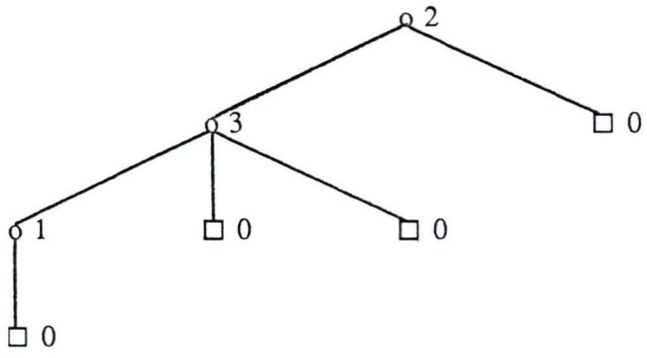
Figure 4.3 A binary tree and its representation sequences

node, except the last leaf, as it is visited. See Figure 4.4. (The last node's label need not be recorded since in a preorder traversal, the last node visited is always a leaf node.) Define the *degree of a node* in a tree as the number of children it has, the *degree of a tree* as the maximum degree among all nodes in the tree, and the representation sequence described above as the degree sequence of the tree. If a tree T has degree t and has n_i nodes of degree i, $n_i \geq 0$, $i = 0, \ldots, t$, then the degree sequence of T is a permutation of the multiset $0^{n_0}1^{n_1} \ldots t^{n_t}$. However not all permutations of the multiset correspond to trees. A trivial example is any sequence of length > 1 beginning with 0. Such a sequence cannot represent a tree since unless the tree is the trivial tree with one node, its root must have degree > 0. A sequence

represents a tree only if it possesses the *dominating property* and has the correct number of 0's.

Definition 4.1 *Let* $\#i(a_1a_2 \cdots a_l)$ *denote the number of times the integer* i *appears in the integer sequence* $a_1a_2 \cdots a_l$. *An integer sequence* $a = a_1a_2 \cdots a_n$ *with* n_i *occurrences of the integer* i, $i=0, \ldots, t$, *has the* dominating property *if the number of 0's is not greater than* $\sum_{i=1}^{t}(i-1)\#i(a_1a_2 \cdots a_l)$ *in every prefix* $a_1a_2 \cdots a_l$, $1 \le l \le n$.

In a tree the number of leaves (degree 0 nodes) must be $\sum_{i=1}^{t}(i-1)n_i + 1$. Hence the representation sequence of a tree must have $\sum_{i=1}^{t}(i-1)n_i$ 0's. Note that a sequence with more



degree sequence: 231000

Figure 4.4 The degree sequence representation of a tree

than this number of 0's corresponds to a forest See Figures 4 5(a)-(c)

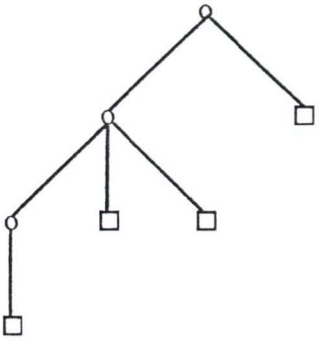


Figure 4 5(a) Tree degree sequence 231000

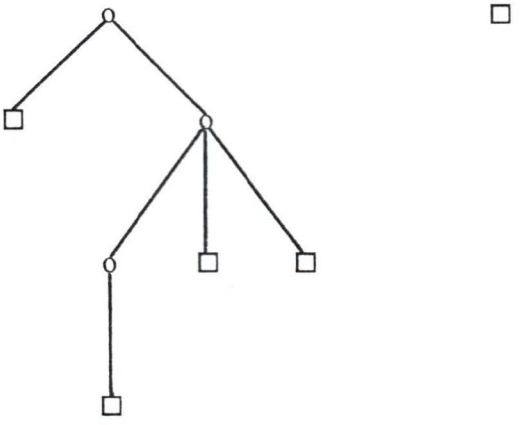


Figure 4 5(b) Forest degree sequence 2031000

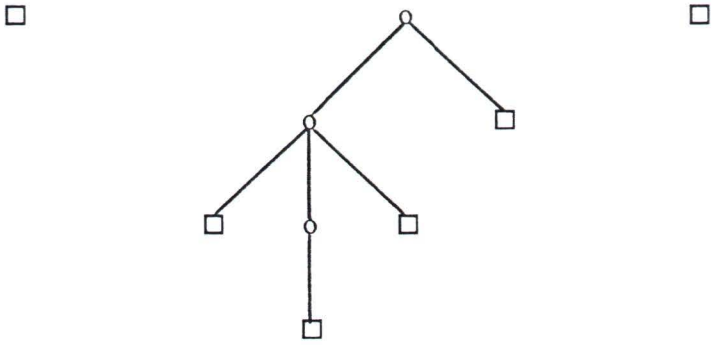


Figure 4.5(c) Forest degree sequence 02301000

If a tree has n_i nodes of degree i, i=0,1, ,t, call (n_0,n_1, ,n_t) the *degree vector*. Let N = (n_0,n_1, ,n_t), $n_i \geq 0$, i=0, ,t, $n_0 = \sum_{i=1}^{t}(i-1)n_i$. Let T(N) be the set of all trees with n_0+1 leaves and n_i nodes of degree i, i=1, ,t. Let A(N) be the set of all integer sequences with n_i i's, i=0, ,t which satisfy the dominating property. It is known ([10]) that

Theorem 4.2 *There is a 1-1 correspondence between T(N) and A(N).*

If in the vector N $n_0 > \sum_{i=1}^{t}(i-1)n_i$ then T(N) is now a set of forests. [10] showed that in this case the 1-1 correspondence in Theorem 4.2 continues to hold.

It is worth noting at this point that for the degree sequences the usual lexicographical ordering of integer sequences correspond to a natural ordering of trees. Let r_T denote the degree of the root of a tree T, and T_i the subtree rooted at the i*th* son of the root of T. Then

Two trees, T and T', are in *B-order*, $T < T'$, if

(1) $r_T < r_{T'}$ or

(2) $r_T = r_{T'}$ and for some i, $1 \leq i \leq r_T$ we have

a) $T_j = T'_j$ for $j = 1,2, \ldots, i-1$ and

b) $T_i < T'_i$

It can be shown that if $T < T'$ in B-order, then the degree sequence of T is lexicographically less than the degree sequence of T'. Zaks [10] gave an algorithm that generates all trees in T(N) lexicographically by generating all degree sequences in A(N). See §4.2.1.

When an algorithm is said to generate trees, it actually only generates the representation sequences of the trees. One important question is whether properties of the sequences are reflected in the structures of the corresponding trees. In this case, if there is an adjacent interchange generation algorithm for A(N), do the trees corresponding to degree sequences successively generated by this algorithm also differ little in structure?

Figure 4.6 shows the effect of an adjacent interchange in the degree sequence on the actual trees themselves. By interchanging the degrees of 2 nodes, only the subtree rooted at the first node is changed. More generally, if nodes u and v are interchanged, their subtrees are exchanged. As an example, let node u be the root of the tree T and v its leftmost child. Suppose u has degree p, v has degree q, $p > q > 0$, $T_1, T_2, \ldots, T_q$ are subtrees rooted at the children of v, and $S_1, S_2, \ldots, S_p$ are subtrees rooted at the children of u (S_1 is the subtree consisting of v and $T_1, \ldots, T_q$). See Figure 4.7. Let $\alpha(T)$ denote the degree sequence of the tree T. Then $\alpha(T) = pq\alpha(T_1)0\alpha(T_2)0 \ldots \alpha(T_q)0\alpha(S_2)0\alpha(S_3)0 \ldots \alpha(S_p)$. From this

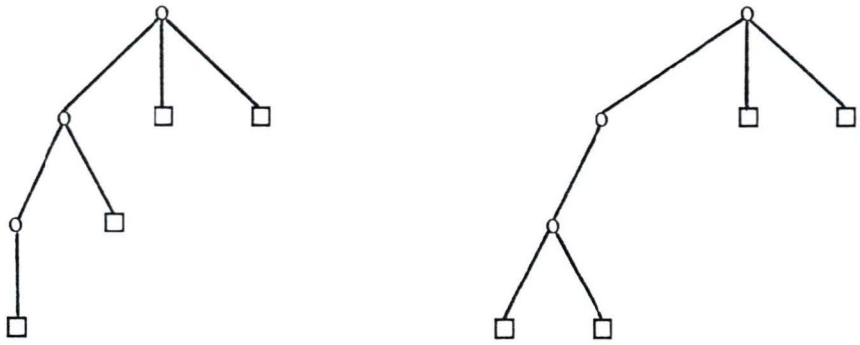
sequence it is apparent that after the interchange the root u now has only q children and its leftmost child v has p children, T_1, ,T_q, S_2, ,(S_{p-q+1}) are now rooted at the children of v, and the tree has the form in Figure 4.8. The cases where $p < q$ or $q = 0$ are similar. Therefore using the degree sequence representation the adjacent interchange property is reflected in the sense that differences in successively generated trees are little.

As an illustration of the importance of the *little change* property, consider the following. It is well known [23] that there is a 1-1 correspondence between trees in general and binary trees. The transformation from a tree to a binary tree is as follows.

(1) the left most child of a node w, if it exists, becomes w's left child in the binary tree,

(2) the sibling to the immediate right of w, if it exists, is w's right child in the binary tree.

For example in the binary tree corresponding to the tree in Figure 4.7 vertex v is vertex u's left child, vertex 2 is v's right child, vertex 3 is vertex 2's right child, etc. In the binary tree representation, to replant the subtrees S_2, S_3, , S_{p-q+1}, it is an easy operation to change the parent of vertex p-q+2 to vertex v, and to change the parent of vertex 2 to the root of T_q. This efficiency in replanting is important when many trees are generated and manipulated.



321000 312000

Figure 4.6

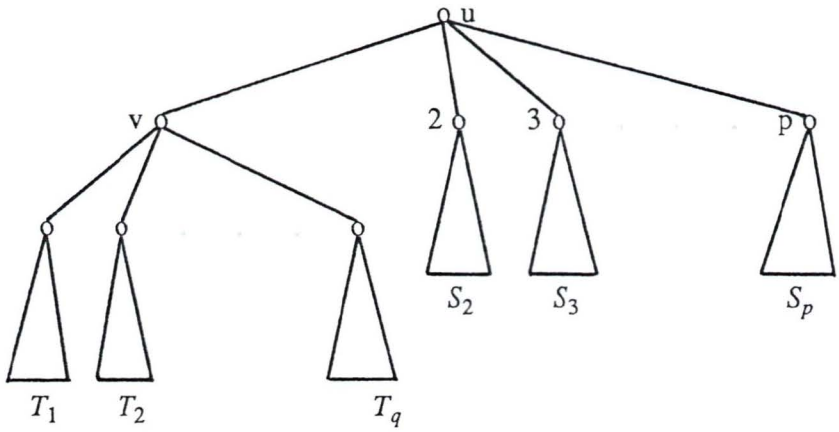


Figure 4.7

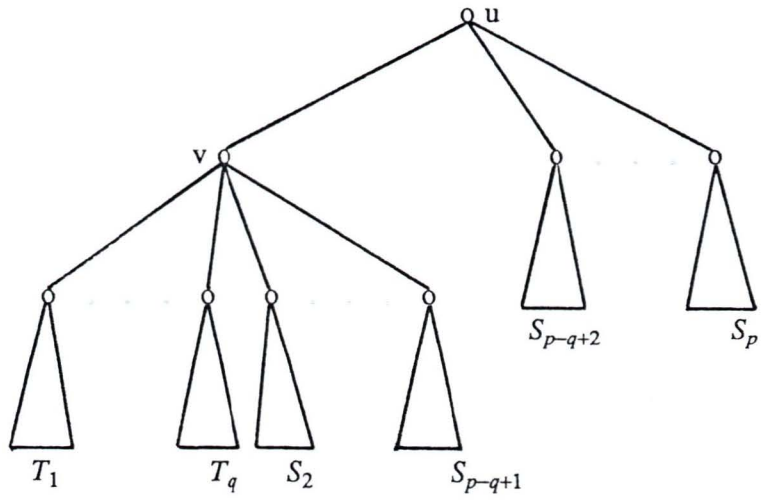


Figure 4.8

4.2 Previous Works

4.2.1 Number of Rooted Ordered Trees

Zaks [10] was the first to give an algorithm to generate all degree sequences for any degree vector N = $(n_0,n_1, \quad ,n_t)$, where $n_0 = \sum_{i=1}^{t}(i-1)n_i$ He also showed that |T(N)|, denoted by $C(n_0,n_1, \quad ,n_t)$, satisfies the following recurrence relation

Lemma 4.3

$$C(n_0,n_1, \quad ,n_t) = \begin{cases} 0 & \text{if } n_i < 0 \text{ for } i = 1,2, \quad ,t \\ 1 & \text{if } n_0 = n_1 = \quad = n_t = 0 \\ 0 & \text{if } n_0 = \sum_{i=1}^{t}(i-1)n_i - 1 \\ \sum_{j=0}^{t} C(n_0, \quad ,n_j-1, \quad ,n_t) & \textit{otherwise} \end{cases}$$

Its solution is given by

$$C(n_0,n_1, \quad ,n_t) = \binom{n}{n_0\ n_1 \quad n_t} - \sum_{i=1}^{t}(i-1)\binom{n}{n_0+1\ n_1 \quad n_i-1 \quad n_t} \qquad (4.1)$$

where $n = \sum_{i=1}^{t} n_i$

Note that the recurrence relation and (4.1) also apply for forests, i.e. when $n_0 > \sum_{i=1}^{t}(i-1)n_i$

Zaks simplified the degree sequences in A(N) to form a new set of sequences B(N) and he gave an algorithm of order O(n) that generates lexicographically all sequences in B(N), where $n = n_0+n_1+ \quad +n_t$ Ranking and unranking algorithms of order O((t+1)n) were also given

4.2.2 Impossibility of Binary Tree Generation by Adjacent Interchange

The degree sequence of a binary tree with n internal nodes is of length $2n$ and consists of n occurrences of each of the integers 2 and 0 If all the 2's are replaced by 1's then the degree sequence becomes the one used by [22]. (See also §4.1) If an undirected graph H_n is defined such that the vertices are all sequences representing binary trees with n internal

nodes, and two sequences (vertices) are adjacent if and only if one can be obtained from the other by an interchange of adjacent integers, then H_n is a subgraph of $G(n_0,n_1)$ defined in §3.1.1. Thus H_n is bipartite with partitions O and E of the vertices, O containing all sequences with odd scores and E all sequences with even scores, where the score function is as defined for combinations in §3.1.1. Letting D_n denote the difference in size of the two partitions of H_n, and using a technique similar to the proof of Lemma 3.4 by Ruskey and Miller [7], the following lemma can be proved.

Lemma 4.4 *If n is even then D_n is 0 and if $n = 2m+1$ is odd then* $D_n = (-1)^m \frac{1}{m+1}\binom{2m}{m}$

Therefore if $n \geq 5$ is odd then there is no adjacent interchange algorithm to generate all binary trees with n internal nodes.

4.3. Impossibility of Tree Generation by Adjacent Interchange

In this section we shall prove a theorem for forests analogous to Theorem 3.7 for multisets. Let $N = (n_0,n_1, \quad ,n_t)$ be a degree vector, with $n_0 \geq \sum_{i=1}^{t}(i-1)n_i$. Define a graph $H(n_0,n_1, \quad ,n_t)$ with all degree sequences in A(N) as vertices, two sequences are adjacent in this graph if and only if they differ by an interchange of adjacent degrees. Since each degree sequence is a permutation of the multiset $0^{n_0}1^{n_1} \quad t^{n_t}$ $H(n_0,n_1, \quad ,n_t)$ is a subgraph of $G(n_0,n_1, \quad ,n_t)$ which is defined in §3.2.1. From Lemma 3.5 $H(n_0,n_1, \quad ,n_t)$ is bipartite. As in the multiset case, if we can compute the difference in size between the two partitions of the vertices, we can determine whether an adjacent interchange algorithm may exist.

Let $t^{n_t} \quad 1^{n_1}0^{n_0}$ be the *canonical* degree sequence We shall classify all other degree sequences in relation to this If a degree sequence can be obtained from the canonical one by an even number of adjacent interchanges, let it be in the set E Otherwise let it be in the set O If two degree sequences in the same set, O or E, are adjacent, then there must be an odd cycle in the graph, a contradiction Therefore (O,E) is a bipartition of the graph The parity difference $D(n_0, \quad ,n_t)$ denotes $|E| - |O|$

Lemma 4 5

$$D(n_0,n_1, \quad ,n_t) = \begin{cases} 0 & \text{if } any\ n_i < 0 \\ 1 & \text{if } all\ n_i = 0 \\ 0 & \text{if } n_0 = \sum_{i=1}^{t}(i-1)n_i - 1 \\ \sum_{i=0}^{t}(-1)^{\sigma_i}D(\quad ,n_i-1, \quad) & otherwise \end{cases} \qquad (4\ 2)$$

where $\sigma_i = \sum_{j>i} n_j$

Proof

The value of the parity difference for the first 3 cases is obvious For the recurrence relation, consider the parity difference as the sum of parity differences of sequences beginning with 0, with 1, , with t We need to show that for each i, $i = 0, \quad ,t$, there is a 1-1 correspondence between forests with sequences beginning with degree i and forests with degree vector $(n_0, \quad ,n_i-1, \quad ,n_t)$ Let the former be called a-forests and the latter b-forests Let iS be a degree sequence for an a-forest Then the root of the leftmost tree in this forest has degree i, and by removing it we obtain a b-forest that has degree sequence S

See Figure 4.9. This and the reverse process are clearly unique, and so the 1-1 correspondence is established.

Now consider all degree sequences beginning with degree i. The parity difference for these sequences is $D(n_0, \quad ,n_i-1, \quad ,n_t)$. However if σ_i is odd, $it^{n_t} \quad i^{n_i-1} \quad 0^{n_0}$ is in O and so the contribution to the total parity difference from these sequences is $-D(n_0, \quad ,n_i-1, \quad ,n_t)$. Clearly then the sign of the contribution to the total parity difference from sequences prefixed with degree i, $i = 0,1, \quad ,t$, is dependent on the partiy of σ_i. The recurrence relation follows.

■

The solution to recurrence relation (4.2) is given in Theorem 4.6. The following notation is used in the statement and proof of the theorem.

m_i denotes $\left\lfloor \frac{n_i}{2} \right\rfloor$

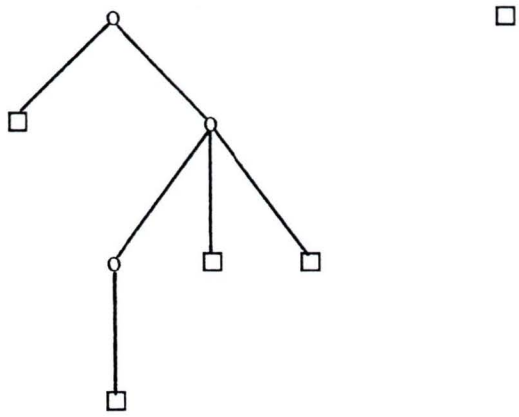
$m = \sum_{i=0}^{t} m_i$

$q = \sum_{odd\ i} m_i$

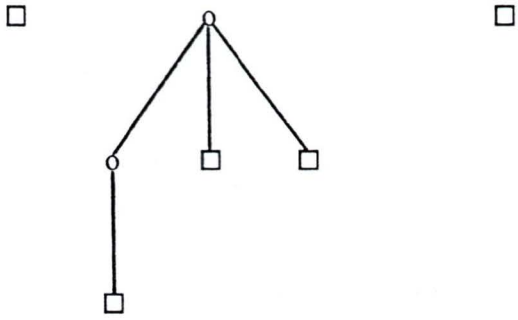
$(m, m_0, m_1, \quad ,m_t)$ denotes the multinomial coefficient $\begin{pmatrix} m \\ m_0\ m_1 \quad m_t \end{pmatrix}$

$$E(0) = \sum_{\substack{i=2 \\ by\ 2}}^{t} (i-1)(m, m_0+1, \quad ,m_i-1, \quad ,m_t)$$

$$E(1) = \sum_{\substack{i=2 \\ by\ 2}}^{t} (i-1)(m-1, m_0, \quad ,m_i-1, \quad ,m_t)$$



a-forest 2031000



b-forest 031000

Figure 4.9

$$O(0) = \sum_{\substack{i=3 \\ by\ 2}}^{t} (i-1)(\ m,\ m_0+1,\ \ \ \ ,m_i-1,\ \ \ \ ,m_t\ )$$

$$O(1) = \sum_{\substack{i=3 \\ by\ 2}}^{t} (i-1)(\ m-1,\ m_0,\ \ \ \ ,m_i-1,\ \ \ \ ,m_t\ )$$

Theorem 4.6 *The value of* $D(n_0,n_1, \quad ,n_t)$, *where* $n_0 \geq \sum_{i=1}^{t}(i-1)n_i$, *is determined by the following cases*

(a) *if* $n_i = 0$ *for all* i *except for* $i = 1$, $n_1 \geq 0$, *then* $D(n_1) = 1$

(b) *if* n_0 *is odd, exactly one* n_i *is odd, where* i *is an even index* > 0, *and* < 2 n_j *are odd, where* j *is odd, then*

$$D(n_0, \quad ,n_t) = \rho\left\{\frac{m+1}{m+1-q}[\ (m, m_0, \quad ,m_t) - E(0)\ ] - \frac{m+1}{m+2-q}\ O(0)\right\} \qquad (4.3)$$

where $\rho_j = \begin{cases} 1 & \text{if there is no odd } j \text{ such that } n_j \text{ is odd,} \\ (-1)^{\sigma_j} & \text{where } j \text{ is the only odd index such that } n_j \text{ is odd} \end{cases}$

(c) *if* n_i *for all even* i *is even,* < 2 n_j *are odd, where* j *is odd, then*

$$D(n_0, \quad ,n_t) = \frac{m}{m-q}[\ (m-1, m_0-1, \quad ,m_t) - E(1)\ ] - \frac{m}{m+1-q}\ O(1) \qquad (4.4)$$

(d) *if* n_i *is even for all* $i > 0$, n_0 *odd, then*

$$D(n_0, \quad ,n_t) = C(m_0, \quad ,m_t) \qquad (4.5)$$

where $C(m_0, \quad ,m_t)$ *is defined in expression (4.1)*

(e) $D = 0$ *otherwise*

Proof

Case (a) is clearly true since there is only one tree, and thus one degree sequence We prove cases (b) to (e) by induction on $n = n_0 + n_1 + \quad + n_t$

If $n = 1$, then either n_0 or n_1 is 1. If $n_0 = 1$ then case (d) applies and the parity difference is 1. If $n_1 = 1$ then by (a) then the parity difference is 1. Suppose the theorem is true for number of nodes $< n$. For a forest with n nodes, we have the following cases.

Case (b)

Let k > 0 be the even index such that n_k is odd.

Suppose there is one n_j odd, j odd. Consider $D(\quad ,n_i-1, \quad)$, i even. If i = k then n_0 is odd and n_r is even for all even r, if $i \neq k$ then there are two even indices, r = i and k, n_r is odd. In either case the degree vector satisfies none of the conditions in cases (a) to (d), and so by the induction hypothesis $D(\quad ,n_i-1, \quad) = 0$. Consider $D(\quad ,n_i-1, \quad)$, i odd, $i \neq j$. Then n_i-1 and n_j are both odd, i and j are odd, and so by induction $D(\quad ,n_i-1, \quad) = 0$. Therefore

$$D(n_0, \quad ,n_t) = (-1)^{\sigma_j} D(n_0, \quad ,n_j-1, \quad ,n_t)$$

$$= \rho_j \left\{ \frac{m+1}{m+1-q} \left[(m, m_0, \quad ,m_t) - E(0) \right] - \frac{m+1}{m+2-q} O(0) \right\}$$

Suppose there is no odd n_j, for j odd. Then as before $D(\quad ,n_i-1, \quad) = 0$ for i even, $i \neq k$. Therefore,

$$D(n_0,n_1, \quad ,n_t)$$

$$= \sum_{\substack{i=1 \\ by\ 2}}^{t} (-1)^{\sigma_i} D(\quad ,n_i-1, \quad) + D(\quad ,n_k-1, \quad)$$

$$= \sum_{\substack{i=1 \\ by\ 2}}^{t} (-1)^{\sigma_i} \rho_i \left\{ \frac{m}{m+1-q} \left[(\ m-1, \qquad , m_i-1, \qquad) \right. \right.$$

$$\left. - \sum_{\substack{j=2 \\ by\ 2}}^{t} (j-1)(\ m-1, m_0+1, \qquad , m_i-1, \qquad , m_j-1, \qquad) \right]$$

$$\left. - \frac{m}{m+2-q} \left[\sum_{\substack{j=3 \\ by\ 2}}^{t} (j-1)(\ m-1, m_0+1, \qquad , m_j-1, \qquad , m_i-1, \qquad) \right] \right\}$$

$$+ \ C(m_0, \qquad , m_t)$$

$$= \sum_{\substack{i=1 \\ by\ 2}}^{t} \frac{m_i}{m+1-q} [(\ m, m_0, \qquad , m_t\ ) - E(0)]$$

$$- \sum_{\substack{i=1 \\ by\ 2}}^{t} \frac{1}{m+2-q} \left[\sum_{\substack{j=3 \\ by\ 2 \\ \neq i}}^{t} (j-1) m_i (\ m, m_0+1, \qquad , m_j-1, \qquad) \right.$$

$$\left. + \ (i-1)(m_i-1)(\ m, m_0+1, \qquad , m_i-1, \qquad) \right]$$

$$+ \ (\ m, m_0, \qquad , m_t\ ) - \sum_{i=2}^{t} (i-1)(\ m, m_0+1, \qquad , m_i-1, \qquad , m_t\ )$$

$$= \frac{m+1}{m+1-q}[\ (\ m,\ m_0,\quad ,m_t\ ) - E(0)\ ] - \frac{m+1}{m+2-q}O(0)$$

Case (c)

Suppose there is only one n_j odd, j odd. If all n_i is 0 except for n_1, then we have case (a) which is already proved. Therefore assume $n_i > 0$ for some $i \neq 1$, in which case $n_0 > 0$. Consider $D(\quad ,n_i-1,\quad)$, i even. This degree vector does not satisfy any of the conditions in cases (a) to (d). Hence by induction this quantity is 0. Similarly $D(\quad ,n_i-1,\quad)$ is 0 for i odd, $i \neq j$. Therefore,

$$\begin{aligned} D(n_0,\quad ,n_t) &= (-1)^{\sigma_j} D(\quad ,n_j-1,\quad) \\ &= D(\quad ,n_j-1,\quad) \end{aligned}$$

which by induction has the expression as stated.

Now suppose there is no n_j odd, j odd. The analysis for this case is similar to that in the proof of (b), except when $n_0 = \sum_{i=2}^{t}(i-1)n_i$. For this value of n_0, $m_0 = \sum_{i=2}^{t}(i-1)m_i$.

$$\begin{aligned} & D(n_0,\quad ,n_t) \\ &= \sum_{\substack{i=1 \\ by\ 2}} (-1)^{\sigma_i} D(\quad ,n_i-1,\quad) \end{aligned}$$

$$= \sum_{\substack{i=1 \\ by\ 2}}^{t} \frac{m-1}{m-q} \left[(\, m-2,\ m_0-1,\ \quad ,m_i-1,\ \quad) - \sum_{\substack{j=2 \\ by\ 2}}^{t} (j-1)(\, m-2,\ \quad ,m_i-1,\ \quad ,m_j-1,\ \quad) \right]$$

$$- \sum_{\substack{i=1 \\ by\ 2}}^{t} \frac{m-1}{m+1-q} \left[\sum_{\substack{j=3 \\ by\ 2}}^{t} (j-1)(\, m-2,\ \quad ,m_i-1,\ \quad ,m_j-1,\ \quad) \right]$$

$$= \sum_{\substack{i=1 \\ by\ 2}}^{t} \frac{m_i}{m-q} [\, (\, m-1,\ m_0-1, m1,\ \quad ,m_t) - E(1)\,] - \left[\sum_{\substack{i=1 \\ by\ 2}}^{t} \frac{m_i}{m+1-q} - \frac{1}{m+1-q} \right] O(1)$$

Subtracting this from expression (4 4), we have

$$(\, m-1,\ m_0-1, m_1,\ \quad ,m_t) - \sum_{i=2}^{t} (i-1)(\, m-1,\ \quad ,m_i-1,\ \quad)$$

$$= (\, m-1,\ m_0-1, m_1,\ \quad ,m_t) - \left[\frac{1}{m_0} \sum_{i=2}^{t} (i-1) m_i \right] (\, m-1,\ m_0-1, m_1,\ \quad ,m_t)$$

$$= (\, m-1,\ m_0-1, m_1,\ \quad ,m_t) - (\, m-1,\ m_0-1, m_1,\ \quad ,m_t)$$

$$= 0$$

Thus (c) is proved

Case (d)

Here we have all n_i even except for n_0 Consider $D(\quad ,n_i-1,\quad)$, where i is even, $i > 0$ This degree sequence is covered by case (b) For $D(n_0-1,\ n_1,\quad ,n_t)$ the degree

sequence is covered by case (c). $D(\quad ,n_i-1,\quad)$ where i is odd must be 0 by induction and recurrence relation (4.2). Therefore,

$$
\begin{aligned}
& D(n_0,n_1,\quad ,n_t) \\
&= \sum_{\substack{i=0 \\ by\ 2}}^{t} (-1)^{\sigma_i} D(\quad ,n_i-1,\quad) \\
&= \sum_{\substack{i=2 \\ by\ 2}}^{t} \left\{ \frac{m}{m-q} \left[(\, m-1,\quad ,m_i-1,\quad) - \sum_{\substack{j=2 \\ by\ 2}}^{t} (j-1)(\, m-1,\, m_0+1,\quad ,m_i-1,\quad ,m_j-1,\quad) \right] \right. \\
&\qquad \left. - \frac{m}{m+1-q} \sum_{\substack{j=3 \\ by\ 2}}^{t} (j-1)(\, m-1,\, m_0+1,\quad ,m_i-1,\quad ,m_j-1,\quad) \right\} \\
&\qquad + \frac{m}{m-q} [\, (\, m-1,\, m_0-1,\quad) - E(1)\,] - \frac{m}{m+1-q} O(1) \\
&= \left[\sum_{\substack{i=2 \\ by\ 2}}^{t} \frac{m_i}{m-q} \right] (\, m,\, m_0,\quad ,m_t\,) \\
&\qquad - \left[\left[\sum_{\substack{i=2 \\ by\ 2}}^{t} \frac{m_i}{m-q} \right] - \frac{1}{m-q} \right] E(0) - \left[\sum_{\substack{i=2 \\ by\ 2}}^{t} \frac{m_i}{m+1-q} \right] O(0) \\
&\qquad + \frac{m_0}{m-q} (\, m,\, m_0,\quad ,m_t) - \frac{m_0+1}{m-q} E(0) - \frac{m_0+1}{m-q+1} O(0)
\end{aligned}
$$

$$= (m, m_0, \quad , m_t) - \sum_{i=2}^{t} (i-1)(m, m_0+1, \quad , m_j-1, \quad)$$

$$= C(m_0, \quad , m_t)$$

Case (e)

If the degree vector does not satisfy the conditions in cases (b) to (d), then it must be in one of the following five categories

(1) At least two n_i's odd, i odd, AND at least two n_j's odd, j > 0 is even

(2) Two n_i's odd, i odd, AND one n_j odd, j > 0 even

(3) Two n_i's odd, i odd (or even), all other n_j's with j > 0 are even

(4) One n_j odd, j even, n_0 even, < 2 n_i's odd for i odd

(5) For all even i > 0 n_i is even, one n_j odd for j odd, and n_0 odd

Subcase e1 At least two n_i's odd, i odd, AND at least two n_j's odd, j > 0 is even. The result follows directly from the induction hypothesis and the recurrence relation

Subcase e2 n_r, n_s are odd, where r > s are odd indices, AND n_j is odd, where j is an even index, and all other n_k's are even.

If n_0 is even, then the result follows directly from the induction hypothesis and recurrence relation (4.2). If n_0 is odd, then

$$D(n_0, \quad ,n_t) = (-1)^{\sigma_r} D(\quad ,n_r-1, \quad) + (-1)^{\sigma_s} D(\quad ,n_s-1, \quad)$$

σ_r even => σ_s even if r > j > s

σ_r even => σ_s odd if j < s

σ_r odd => σ_s even

In any case

$$|D(n_0, \quad ,n_t)| = | \, |D(\quad ,n_s, \quad ,n_r-1, \quad)| - |D(\quad ,n_s-1, \quad ,n_r, \quad)| \, |$$

But from expression (4.3) n_r and n_s odd implies that

$$|D(\quad ,n_s-1, \quad ,n_r, \quad)| = |D(\quad ,n_s, \quad ,n_r, \quad)|$$

$$= |D(\quad ,n_s, \quad ,n_r-1, \quad)|$$

The result follows.

Subcase e3 n_r, n_s are odd, where r > s are odd indices, and all other n_k's, k > 0, are even. If i ≠ r or s, then from induction and recurrence relation (4.2) $D(\quad ,n_i-1, \quad) = 0$. Therefore,

$$D(n_0, \quad ,n_t) = (-1)^{\sigma_r} D(\quad ,n_r-1, \quad) + (-1)^{\sigma_s} D(\quad ,n_s-1, \quad)$$

$$= D(\quad ,n_s, \quad ,n_r-1, \quad) - D(\quad ,n_s-1, \quad ,n_r, \quad)$$

$$= 0 \qquad \textit{as in subcase e2}$$

The subcase for two n_i's odd, where i is even, is similar

Subcase e4 One n_j is odd for j even, and n_0 is even. Again from recurrence relation (4.2) and induction $D(\quad ,n_i-1, \quad)$ is 0 except for i=j and i=0. Hence

$$D(n_0, \quad ,n_t) = (-1)^{\sigma_j} D(\quad ,n_j-1, \quad) + (-1)^{\sigma_0} D(n_0-1, \quad)$$

Consider the case where there is one k, k odd, such that n_k is odd, and k > j. The cases where k < j and where there is no such k are similarly solved. From case (c),

$$(-1)^{\sigma_j} D(\quad ,n_j-1, \quad)$$

$$= (-1)\left\{\frac{m}{m-q}[\ (m-1, m_0-1, \quad) - E(1)\ ] - \frac{m}{m+1-q} O(1)\right\}$$

From (b), with the number of degree zero nodes being n_0-1, where n_0 and m_0 are even, we can see that ρ is +1 and we have the same expression as above. Therefore the parity difference is 0 as claimed.

Subcase e5 n_0 is odd, for all even i > 0 n_i is even, and there is one n_j odd, where j is odd. Consider $D(\quad ,n_i-1, \quad)$ where i is odd but i≠j. This degree vector has two odd indices, i and j, such that n_i-1 and n_j are odd. Thus $D(\quad ,n_i-1, \quad) = 0$ by induction. In the following assume $n_0 > 1$ or there is an even index $i \geq 2$ such that $n_i > 0$. The special case that n_0 is 1 and n_i is 0 for i > 0 will be treated later.

$$D(n_0, \quad ,n_t) = \sum_{\substack{i=0 \\ by\ 2}}^{t} (-1)^{\sigma_i} D(\quad ,n_i-1, \quad) + (-1)^{\sigma_j} D(\quad ,n_j-1, \quad)$$

$$= (-1) \left\{ \sum_{\substack{i=0 \\ by\ 2}}^{t} \frac{m_i}{m-q} (m, m_0, \quad ,m_t) - \left[\sum_{\substack{i=2 \\ by\ 2}}^{t} \frac{m_i}{m-q} - \frac{1}{m-q} + \frac{m_0+1}{m-q} \right] E(0) \right.$$

$$\left. - \left[\sum_{\substack{i=2 \\ by\ 2}}^{t} \frac{m_i}{m+1-q} + \frac{m_0+1}{m+1-q} \right] O(0) \right\} + C(m_0, \quad ,m_t)$$

$$= (-1) \left\{ (m, m_0, \quad ,m_t) - \sum_{i=2}^{t} (i-1)(m, m_0+1, \quad ,m_i-1, \quad) \right\}$$

$$+ (m, m_0, \quad ,m_t) - \sum_{i=2}^{t} (i-1)(m, m_0+1, \quad ,m_i-1, \quad)$$

$$= 0$$

If $n_0 = 1$, $n_i = 0$ for all $i > 0$, and $n_1 > 0$ is odd, then

$$D(1,n_1) = D(1,n_1-1) - D(0,n_1)$$

$$= (m1, 0,m_1) - 1$$

$$= 0$$

By induction all cases have been proved and so Theorem 4.6 is true

■

For trees $n_0 = \sum_{i=1}^{t}(i-1)n_i$. This restriction implies that if $n_1,$ $,n_t$ are all even, n_0 must be even also. Consequently expression (4.5) does not apply for trees.

Corollary 4.7 *The value of $D(n_0,n_1,$ $,n_t)$ for trees is determined by the following cases*

(a) *if $n_i = 0$ for all i except for $i = 1$, $n_1 \geq 0$, then $D(n_1) = 1$*

(b) *if n_0 is odd, exactly one n_i is odd for $i > 0$ even, and < 2 n_j's are odd for j odd, then $D(n_0,$ $,n_t)$ has values given by expression (4.3)*

(c) *if n_i for all even i is even, < 2 n_j's are odd and at least one $n_j > 0$ for j odd, then $D(n_0,$ $,n_t)$ has values given by expression (4.4)*

(d) *$D = 0$ otherwise*

Proof

All cases follow from Theorem 4.6 except when n_i for all even i is even and all n_j's are 0, j odd. For this case $m_0 = \sum_{\substack{i=2 \\ by\ 2}} m_i$, $q = 0$, and (4.4) becomes

$$
\begin{aligned}
& D(n_0, \quad ,n_t) \\
&= (m-1, m_0-1, \quad ,m_t) - E(1) \\
&= (m-1, m_0-1, \quad ,m_t) - \sum_{\substack{i=2 \\ by\ 2}} (i-1)(m-1,m_0, \quad ,m_i-1, \quad ,m_t)
\end{aligned}
$$

$$= (m-1, m_0-1, \quad ,m_t) - \left[\sum_{\substack{i=2 \\ by\ 2}} \frac{(i-1)m_i}{m_0}\right](m-1, m_0-1, \quad ,m_t)$$

$$= (m-1, m_0-1, \quad ,m_t) - (m-1, m_0-1, \quad ,m_t)$$

$$= 0$$

■

Corollary 4.8 *There is no adjacent interchange algorithm for trees if*

(a) $n_1 \geq 0$, *all other* n_i*'s* $= 0$, *OR*

(b) *at least one* $n_i >= 2$, $i > 0$, *and*

(b.1) n_0 *is odd, exactly one* n_i *is odd for* i *even, and* < 2 n_j*'s are odd for* j *odd,*

(b.2) n_i *for all even* i *is even,* < 2 n_j*'s are odd and at least one* $n_j > 0$ *for* j *odd,*

except for the degree vector ($n_0 = 4$, $n_1 = 0$, $n_2 = 2$, $n_3 = 1$)

Proof The parity difference for the degree vector ($n_0 = 4$, $n_1 = 0$, $n_2 = 2$, $n_3 = 1$) is 1. For the rest of the corollary it can be easily verified that expressions (4.3) and (4.4) are always greater than 1 with the conditions stated above.

■

CHAPTER 5

CONCLUSIONS

In Chapter 2 Algorithm 8 is presented It is a generalization of Algorithm 7, a combination generation algorithm due to Bitner Algorithm 8 generates all permutations of any multiset by interchange so that the average time used to generated each permutation is bounded by a constant, independent of t, the number of symbols, and n, the length of each permutation It is also shown that binary combinations generated by Algorithm 8 can be ranked and unranked, and that both the ranking and unranking algorithms are of order $O(n^2)$

In Chapter 3 the existence problem of multiset generation algorithms by adjacent interchange is examined and solved It is shown that for a multiset with n_i occurrences of the symbol i, $i = 0, \ldots, t$, a graph $G(n_0, n_1, \ldots, n_t)$ can be defined such that the vertices are the permutations of the multiset and an edge between two vertices exists if and only if those two permutations differ by an adjacent interchange, and the existence problem is equivalent to the existence problem of a Hamilton path in this graph This graph is shown to be bipartite and the vertices can be partitioned into two sets, O and E, such that no two vertices from the same set can be adjacent. It is proved that the parity difference $|E| - |O|$ is as follows

(a) if ≥ 2 n_i's are odd, then it is 0;

(b) otherwise it is the multinomial coefficient

$$(-1)^{\sigma_i}\begin{bmatrix} m \\ m_0\ m_1 \quad\quad m_t \end{bmatrix}$$

where $m_i = \left\lfloor \frac{n_i}{2} \right\rfloor$, $m = \sum_{i=0}^{t} m_i$, and $\sigma_i = \sum_{j<i} m_j$

$G(n_0,n_1, \quad ,n_t)$ is bipartite implies that if the parity difference is greater than 1, then it is not possible for a Hamilton path to exist. It follows that if at least 2 n_i's are > 1 and at most one n_i is odd, then the multinomial coefficient is > 1 and no adjacent interchange algorithm is possible

In the binary case [7] showed that when the parity difference is 0, there is an adjacent interchange generation algorithm. In a multiset permutation p of the same length as a combination c, there are more possibilities of adjacent interchange. Since in general the more edges there are in a graph, the more likely it is to contain a Hamilton path, the chances are great that an adjacent interchange generation algorithm exists for multiset permutations when the parity difference is 0.

Conjecture 5.1 *If ≥ 2 n_i's are odd, then the permutations of the multiset $0^{n_0}1^{n_1} \quad t^{n_t}$ can be generated by an adjacent interchange algorithm.*

Degree sequences representing ordered forests of rooted ordered trees are multiset permutations. In Chapter 4 results for such forests analogous to those for multisets in Chapter 3 are found. For all degree sequences consisting of n_0 0's, n_1 1's, , n_t t's, a bipartite graph $H(n_0,n_1, \quad ,n_t)$ is defined with bipartition (O,E). The parity difference is found to be as follows.

(a) If $n_i = 0$ for $i \neq 1$ and $n_1 \geq 0$, then the parity difference is 0,

(b) if n_0 is odd, exactly one n_i is odd, $i > 0$ even, and < 2 n_j's are odd, j odd, then the parity difference is

$$\rho_j \left\{ \frac{m+1}{m+1-q} \left[\binom{m}{m_0\ m_1 \cdots m_t} - \sum_{\substack{i=2 \\ by\ 2}}^{t} \binom{m}{m_0+1 \cdots m_i-1 \cdots} \right] \right.$$

$$\left. - \frac{m+1}{m+2-q} \sum_{\substack{i=3 \\ by\ 2}}^{t} \binom{m}{m_0+1 \cdots m_i-1 \cdots} \right\}$$

where $\rho_j = \begin{cases} 1 & \text{if } \textit{there is no odd } j \textit{ such that } n_j \textit{ is odd,} \\ (-1)^{\sigma_i} & \textit{where } j \textit{ is the only odd index such that } n_j \textit{ is odd} \end{cases}$

where $\sigma_i = \sum_{j>i} m_j$,

(c) if n_i for all even i is even, and < 2 n_j's are odd, j odd, then the parity difference is

$$\frac{m}{m-q} \left[\binom{m-1}{m_0-1\ m_1 \cdots m_t} - \sum_{\substack{i=2 \\ by\ 2}}^{t} \binom{m-1}{m_0 \cdots m_i-1 \cdots} \right]$$

$$- \frac{m}{m+1-q} \sum_{\substack{i=3 \\ by\ 2}}^{t} \binom{m}{m_0 \cdots m_i-1 \cdots}$$

(d) if n_i for all $i > 0$, *no* is odd, then the parity difference is

$$\begin{bmatrix} m \\ m_0\ m_1 \quad\quad m_t \end{bmatrix} - \sum_{i=2}^{t} \begin{bmatrix} m \\ m_0+1 \quad\quad m_i-1 \end{bmatrix}$$

(e) the parity difference is 0 otherwise

For trees $n_0 = \sum_{i=2}^{t}(i-1)n_i$ Consequently case (d) does not apply Furthermore if $n_i = 0$ for all odd i's and n_j is even for all even j's, then the expression in case (b) is 0 Therefore it can be concluded that no adjacent interchange algorithm for trees can exist if

(a) $n_1 \geq 0$ and all other n_i's = 0, OR

(b) at least one $n_i >= 2$, $i > 0$, and

(b 1) n_0 is odd, exactly one n_i is odd for i even, and < 2 n_j's are odd for j odd,

(b 2) n_i for all even i is even, < 2 n_j's are odd and at least one $n_j > 0$ for j odd,

except when $n_0 = 4$, $n_1 = 0$, $n2 = 2$, and $n3 = 1$

APPENDIX A

A.1: Pascal Version of Algorithm 8

```
Type Definitions

  nPtr = ^nRecord;
  nRecord = record
    ptr : ARRAY [BOOLEAN] OF nPtr;          { ptr[FALSE] -> parent }
                                            { ptr[TRUE] -> child  }
    val, cnt : INTEGER;                     { val = symbol }
                                            { cnt = occurrences of the symbol }
   END;
  { Circular Doubly Linked List used to hold the multiset, one symbol per
    record, symbol can be any integer between 0 and 9 }

  STRING = ARRAY [0..maxlength] OF 0..9;      { multiset pattern }

  DIRARRAY = ARRAY [0..maxlength] OF BOOLEAN; { order (+/-) of generation }


  { ---------- Main Variables ---------- }

  dir : DIRARRAY;             { direction of generation }
  Remains : nPtr;             { dummy header of circular symbols list }
  length : INTEGER;           { size of multiset }
  permutation : STRING;       { multiset pattern }
  symcount : INTEGER;         { number of symbols left }
  skipmem : DIRARRAY;         { remembers if a path deleted here }
  skiphere : BOOLEAN;         { no. of nodes deleted at this level }

{  This PROCEDURE traverses the recursion tree. Each leaf node
   represents a generated permutation. All VARiables updated
   before and after a recursive call.                        }

PROCEDURE ALG8( where : INTEGER );
VAR
```

```
  cur   nPtr,
  d   BOOLEAN;

  {   This PROCEDURE updates the pointers in a deletion from or an
      insertion into the doubly linked list of symbols                }

  PROCEDURE update( p1, p2   nPtr ),
  BEGIN
    WITH cur^ DO BEGIN
      ptr[NOT d]^.ptr[d]  = p1,
      ptr[d]^.ptr[NOT d]  = p2,
    END,
  END,


BEGIN  { ALG8 }

  skiphere  = skiphere <> skipmem[where],
  skipmem[where]  = FALSE,

  IF NOT skiphere THEN dir[where]  = NOT dir[where],

  {  set up direction of generation for this level }

  d  = dir[where],
  cur  = Remains^.ptr[d],


  {  generate this level,  if last level, output the multiset pattern,
     if not generate the next level                                   }

  IF where >= length THEN BEGIN
    permutation[where]  = cur^.val,
    skiphere  = FALSE,
    doout,
    permutation[where]  = 0,
  END

  ELSE

  IF ( symcount = 1 ) AND ( cur^.val = 0 ) THEN BEGIN
    skiphere  = NOT skiphere,
    skipmem[where+1]  = skiphere <> skipmem[where+1],
    skiphere  = FALSE,
```

```
  doout,
END

ELSE
WHILE cur <> Remains DO WITH cur^ DO BEGIN

  cnt := PRED( cnt ),

  permutation[where] := val,

  IF cnt = 0 THEN BEGIN
    update( ptr[d], ptr[NOT d] ),          { do deletion }
    symcount := PRED( symcount );
  END;

  ALG8( where + 1 ),                       {build next level}

  IF cnt = 0 THEN BEGIN
    update( cur, cur ),                    { do insertion }
    symcount := SUCC( symcount ),
  END;

  permutation[where] := 0;

  cnt := SUCC( cnt );

  cur := cur^ ptr[d];

 END;
END;
```

A.2: Pascal Version of Ranking Algorithm for Algorithm 8 (in the binary case)

```
FUNCTION rank( VAR a   STRING, var d   DIRARRAY,
                   j, n, k   INTEGER )   INTEGER,
VAR
   k1, k0   INTEGER,
   l        INTEGER,
   dd       BOOLEAN,
   first    BOOLEAN,   {a[j] expanded first or second}

   { a[j] to a[nn] contains the combination to be ranked }
   { d[j] to d[nn] contains the initial directions }
   { k is the number of 1's in the combination }
   { n-k is the number of 0's }
   { the binomial coefficient (i,j) is stored in the array C[i,j] }

BEGIN
  IF ( n = k ) OR ( k = 0 ) THEN rank  = 0
  ELSE BEGIN
    first  = ( (a[j] = 0) AND d[j] )
           OR ( (a[j] = 1) AND (NOT d[j]) ),

    IF a[j] = 1 THEN BEGIN
      k1  = k, k  = k - 1,
    END ELSE k1  = k-1,
    k0  = n-1 - k1,

    IF first THEN
      rank  = rank( a, d, j+1, n-1, k )
    ELSE BEGIN
      d[j+1]  = NOT d[j+1],
      FOR l  = j+2 TO nn-1 DO BEGIN
                 dd  = ODD( C[l-j-2,k0] + C[l-j-2,k1] ),
                 d[l]  = d[l] <> dd,
      END,
      rank  = C[n-1,k1] + rank( a, d, j+1, n-1, k ),
    END,
  END,
END,
```

A.3 Pascal Version of Unranking Algorithm for Algorithm 8 (in the binary case)

```
PROCEDURE unrank( r, j, n, k : INTEGER; VAR a : STRING;
                                  VAR d : DIRARRAY );
VAR k0, k1 : INTEGER;
    dd : BOOLEAN;
    l : INTEGER;
    first : 0..1;

    { r is the rank for which the combination is to be found }
    { a[j] to a[nn] contains the combination that is unranked }
    { d[j] to d[nn] contains the initial directions }
    { k is the number of 1's in the combination }
    { n-k is the number of 0's }
    { the binomial coefficient (i,j) is stored in the array C[i,j] }

BEGIN
  IF n = k THEN FOR l := j TO nn DO a[l] := 1
  ELSE IF k = 0 THEN FOR l := j TO nn DO a[l] := 0
  ELSE BEGIN

    IF d[j] THEN BEGIN
      first := 0;
      k1 := k;
      k := k - 1;
    END ELSE BEGIN
      first := 1;
      k1 := k - 1;
    END;
    k0 := n-1 - k1;

    IF C[n-1,k1] > r THEN BEGIN
      a[j] := first;
      unrank( r, j+1, n-1, k1, a, d );
    END ELSE BEGIN
      a[j] := 1 - first;
      d[j+1] := NOT d[j+1];
      FOR l := j+2 TO nn-1 DO BEGIN
                dd := ODD( C[l-j-2,k0] + C[l-j-2,k1] );
                d[l] := d[l] <> dd;
      END;
      unrank( r-C[n-1,k1], j+1, n-1, k, a, d );
    END;
```

```
  END, {need to recurse block}
END,
```

Bibliography

[1] T W Sag, Algorithm 242, Permutations of a Set with Repetitions, *Comm ACM 7*, 10 (Oct 1964), 585

[2] P Bratley, Algorithm 306, Permutations with Repetitions, *Comm ACM 10*, 7 (July 1967), 450

[3] T C Hu and B N Tien, Generating Permutations with Nondistinct Items, *American Mathematical Monthly 83b*, (Oct 1976), 629-631

[4] F Ruskey and D Roelants van Baronaigien, Fast Recursive Algorithms for Generating Combinatorial Objects, *Congressus Numerantium 41*, (May 1984), 53-62

[5] P J Chase, Algorithm 383, Permutations of a Set with Repetitions, *Comm ACM 13*, 6 (June 1970), 368-369

[6] J R Bitner, G Ehrlich and E M Reingold, Efficient Generation of the Binary Reflected Gray Code and Its Applications, *Comm ACM 19*, (1976), 517-521

[7] F Ruskey and D J Miller, Adjacent Interchange Generation of Combinations and Trees, Dept of Computer Science-44-IR, University of Victoria, Victoria, B C, Canada, Oct. 1984

[8] M Buck and D Wiedemann, Gray codes with Restricted Density, *Discrete Mathematics 48*, (1984), 163-171

[9] P Eades, M Hickey and R C Read, Some Hamilton Paths and a Minimal Change Algorithm, *J ACM 31*, 1 (Jan 1984), 19-29

[10] S Zaks and D Richards, Generating Trees and Other Combinatorial Objects Lexicographically, *Siam J on Computing 8*, 1 (Feb 1979), 73-81

[11] C W Ko, Exam Solution, (University of Victoria, CSc 420, Spring Term), 1986

[12] P J Chase, Algorithm 382, Combinations of M out of N Objects, *Comm ACM 13*, 6 (June 1970), 368

[13] P J Chase, Transposition Graphs, *Siam J on Computing 2*, 2 (June 1973), 128-133

[14] C W H Lam and L H Soicher, Three New Combination Algorithms with the Minimal Change Property, *Comm ACM 25*, 8 (Aug 1982), 555-559

[15] E M Reingold, J Nievergelt and N Deo, *Combinatorial Algorithms Theory and Pratice*, Prentice Hall, Englewood Cliffs, N J , 1977

[16] J T Joichi and D E White, Gray Codes in Graphs of Subsets, *Discrete Mathematics 31*, (1980), 29-41

[17] S Zaks, Lexicographic Generation of Ordered Trees, *Theoretical Computing Science 10*, (1980), 63-82

[18] F Ruskey and T C Hu, Generating Binary Trees Lexicographically, *Siam J on Computing 6*, (1977), 745-758

[19] F Ruskey, Generating t-ary Trees Lexicographically, *Siam J on Computing 7*, (1978), 424-439

[20] A Trojanowski, Ranking and Listing Algorithms for k-ary Trees, *Siam J on Computing 7*, (1978), 492-509

[21] T Beyer and S M Hedetniemi, Constant Time Generation of Rooted Trees, *Siam J on Computing 9*, 4 (Nov 1980), 706-712

[22] A Proskurowski and R Ruskey, Binary Tree Gray Codes, *Journal of Algorithms 6*, (1985), 225-238

[23] T A Standish, *Data Structure Techniques*, Addison Wesley, Reading, Massachusetts, 1980

VITA

Surname Ko Given Names Chun Wa

Place of Birth Hong Kong Date of Birth June 16, 1962

Educational Institutions Attended, with Dates of Entering and Leaving

U of Victoria Sep. 1980 to Apr 1985

U of Victoria Sep 1985 to Aug. 1986

_____ to _____

_____ to _____

Degrees, Diplomas, Etc , Awarded, with Dates and Names of Institutions

Bachelor of Science, U of Victoria
May 1985

Honors and Awards

NSERC Postgraduate Scholarship, 1985-86

Publications

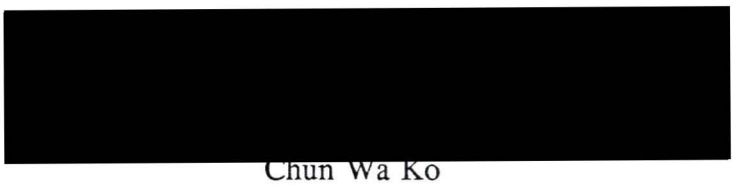
None

PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis (the title of which is shown below) to users of the University of Victoria Library, and to make *single copies only* for such users or in response to a request from the library of any other university, or similar institution, on its behalf or for one of its users I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission

Generation of Permutations of Multisets

Author



Chun Wa Ko

Aug 14, 1986

Date