1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, S.R.C. 1970, c. C-30.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Linear Extensions of Partially Ordered Sets

By

Hin Wai Chang, B.C.S.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario
April 1, 1986

© copyright
1986, Hin Wai Chang

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-29878-2

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis

Linear Extensions of Partially Ordered Sets

submitted by Hin Wai Chang, B. C. S.,
in partial fulfilment of the requirements for
the degree of Master of Computer Science

Thesis Supervisor

F. H. C.

Chairman, School of Computer Science

Carleton University

4 Nov 82

Date

ABSTRACT

The classical sorting problem of arranging n numbers in increasing order, by use of comparisons, is known to require at least $n \log_2 n$ comparisons. The argument giving this result is called the *information theoretic bound* (ITB) and has generalizations to other ordering problems. Finding the ITB for these generalized problems requires the ability to calculate efficiently the number of linear extensions, $Z(P)$, of a partially ordered set P . However, no general polynomial time algorithm for this calculation is known. In this thesis, we develop polynomial time algorithms for two special classes of partially ordered sets:

- (a) those whose Hasse diagrams, when regarded as undirected graphs, are connected and contain cycles that are edge-wise disjoint, and
- (b) those of a bounded width.

We also prove an upper bound on $Z(P)$ for those posets P whose Hasse diagrams are connected.

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my most sincere thanks to my advisor, *Professor Michael Atkinson*, for his diligent supervision. This thesis would not have been possible without his invaluable advice, helpful suggestions, generous support and lasting patience. I am highly indebted to him.

Special thanks are due to *Yanni Jew* and *Azmeena Hasham* for active and useful discussions. Their invaluable suggestions have provided me with deeper insights that led to important results in this thesis. I would also like to thank *Terry Copeck*, *Daniel Ma*, *Mark Waldvogel* and *Paul White* for their assistance. In particular, I would like to thank Paul for proofreading this thesis.

The School of Computer Science has been a wonderful place to spend the last few years, mainly because of the people. I am grateful to both the Faculty and the Administrative and Technical Support Staff for all the help they have given me in one way or another.

This work has been partially supported by the Natural Sciences and Engineering Research Council of Canada under Grant No. A2419.

This thesis is specially dedicated to my parents who made it possible for me to study in Canada.

TABLE OF CONTENTS

	Page
Acceptance Sheet	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
Table of Contents	vi
List of Figures	vii
List of Tables	viii
 Chapter 1. INTRODUCTION	 1
1.1. Classical Sorting and the Information Theoretic Bound	1
1.2. Generalized Sorting Problems	4
1.2.1. Sorting Posets	5
1.2.2. Producing Posets	9
1.3. Research Objectives	13
1.4. Overview of Thesis	14
 Chapter 2. RELATED WORK	 16
2.1. Exponential Algorithms	16
2.2. Special Cases	19
2.3. Zig-Zag Patterns	32
2.4. Merging with Single Constraint	35
 Chapter 3. CYCLES IN THE HASSE DIAGRAM	 37
3.1. Case of One Cycle	38
3.2. Case of Many Cycles	42
3.3. Initial Conditions	50
 Chapter 4. POSETS OF BOUNDED WIDTH	 55
4.1. Posets of Width 2	55
4.2. Posets of Width k	67
4.3. Chain Decomposition	75
 Chapter 5. CONNECTED HASSE DIAGRAMS	 77
 Chapter 6. CONCLUSION	 85
 References	 88

LIST OF FIGURES

Figure	Description	Page
1.1	A decision tree for sorting three elements	2
1.2	Different representations of a poset	6
1.3	Posets to be sorted	8
1.4	Posets to be produced	12
2.1	Illustration of how the improved algorithm works	20
2.2	Merging of two linear extensions	23
2.3	Hasse diagram as a string of bounded graphs	24
2.4	Numbers calculated in each iteration	26
2.5	Hasse diagram in the form of a tree	28
2.6	Poset whose covering graph is a tree	31
2.7	Two zig-zag diagrams	32
2.8	Poset with two lists and one constraint	36
2.9	Four sublists to be merged pairwise	36
3.1	Undirected cycle in the Hasse diagram	37
3.2	Connected Hasse diagram with one cycle	38
3.3	Iterations in the tree algorithm	41
3.4	Connected Hasse diagrams with disjoint cycles	43
3.5	Two forms of the underlying undirected graph G	45
3.6	Cycles that are not disjoint	53
4.1	Poset Q^* derived from poset Q	61
4.2	Poset P of width 2	65
4.3	The Z_{rs} numbers calculated in the construction of P	66
5.1	Connected Hasse diagrams which maximizes $Z(P)$	81
5.2	Three connected Hasse diagrams on three elements	81
5.3	Six connected Hasse diagrams on six elements	82
5.4	Candidates for maximizing $Z(Q)$	83
5.5	Connected Hasse diagram which maximizes $Z(Q)$	83

LIST OF TABLES

Table	Description	Page
2.1	W_{rst} values (with $r = s$)	32

Chapter 1

INTRODUCTION

1.1. Classical Sorting and the Information Theoretic Bound

A basic problem in computer science is to sort a set of n numbers with unknown total order by using binary comparisons. The objective of any algorithm that solves this problem is to minimize the number of comparisons required in the worst case.

The standard model of an algorithm for such a problem is a *decision tree*. A decision tree is a binary tree whose internal nodes are labelled by comparisons between elements of a given set. The two branches from each internal node correspond to the two possible outcomes of the comparison represented by that node. In particular, if the comparison at a node v is between elements α and β , then one child of v corresponds to the outcome $\alpha < \beta$ and the other child to the outcome $\alpha > \beta$. Each decision tree describes an algorithm which starts by making the comparison at the root, and then proceeds to the child corresponding to the result of the comparison. At each internal node v reached by the algorithm, the result of the comparison at this node will decide the next comparison to be made by proceeding to the correct child of v . The leaves correspond to all the possible outcomes of the comparisons. For the classical sorting problem, they are the different possible arrangements or *total orderings* of the elements.

Example 1.1. Consider the decision tree shown in Figure 1.1. Given a set of three elements $\{x_1, x_2, x_3\}$, the algorithm corresponding to this tree first makes a comparison between x_1 and x_2 . If $x_1 < x_2$, it will next make a comparison between x_2 and x_3 . If $x_2 < x_3$, then the algorithm knows that $x_1 < x_2 < x_3$. Otherwise, it compares x_1 with x_3 . Now, if $x_1 < x_3$, the algorithm gives the elements in the order $x_1 < x_3 < x_2$, else it gives them in the order $x_3 < x_1 < x_2$.

Figure 1.1. A decision tree for sorting three elements.

The reader should note that the decision tree describes an algorithm that only works for sets of a particular size. For any such set, the algorithm will trace a path from the root of the tree to one of its leaves. Thus the length of the longest path from the root to a leaf (or the depth of the tree) is the number of

comparisons required by the corresponding algorithm in the worst case.

The lower bound on the number of comparisons needed in the worst case is the minimum length of the longest path over all the decision trees for the problem. There are $n!$ different orderings for n distinct elements and hence there are at least $n!$ leaves with each leaf representing a possible ordering (there may be redundant comparisons). Since there are at least $n!$ leaves, the length of the longest path of any tree will be at least $\log_2 n!$ which can be closely approximated by $n \log_2 n$ using *Stirling's approximation* for factorial numbers:

$$n! \approx n^n e^{-n} \sqrt{2\pi n} \quad \text{as } n \rightarrow \infty.$$

From now on, all logarithms will be to the base 2 unless otherwise specified.

The analysis that we just went through to obtain the lower bound is an example of an *information theoretic argument*. This is one of two general methods used to derive lower bounds for decision problems. The other method, known as the *adversary argument*, produces better results but it is not as simple. The bound of $\log n!$ is known as the *information theoretic bound*. Formally, the information theoretic bound (ITB) for the complexity of a decision problem is the lower bound on the depth of a decision tree derived from a lower bound on the number of leaves. In the example given in Figure 1.1, the number of leaves is $3!$ and thus the ITB is $\log 6 \approx 2.58$.

The ITB is useful in telling us how good an existing algorithm is. It can also tell us the best that can be done in any algorithm. If the complexity of an algorithm is the same as the ITB, the algorithm is *optimal* and the lower bound is said to be *tight*. In the classical sorting problem, there are algorithms such as the mergesort and heapsort that sort a set of n numbers using at most $O(n \log n)$.

$\log n$) comparisons in the worst case. These algorithms are optimal to within certain constants.

1.2. Generalized Sorting Problems

A *partially ordered set* $(P, <)$ consists of a set $P = \{x_1, x_2, \dots, x_n\}$ and a binary relation $<$ defined on P such that the following three laws are satisfied for any $x, y, z \in P$.

- (1) Antireflexive law: we do not have $x < x$.
- (2) Antisymmetric law: if $x < y$, we do not have $y < x$.
- (3) Transitive law: if $x < y$ and $y < z$, then $x < z$.

A partially ordered set is usually simply referred to as a poset or a partial order and when there is no confusion about the relation used in context, we write P instead of $(P, <)$. A total order is a special case of a partial order. It is the case where there is a relation between every two elements in the set. We shall denote the number of elements in P by $|P|$. Also, we may refer to an order relation $x < y$ as the related pair (x, y) .

A poset $(P, <)$ determines a directed acyclic graph with vertex set P and edges $x \longrightarrow y$ if $y < x$. It is usually more convenient to remove those edges that can be deduced by transitivity. The graph in this case is called a *covering graph*. If the graph is drawn in such a way that the edges are always directed downwards, the arrowheads become redundant and can thus be removed. This graph is called the *Hasse diagram* of P . The Hasse diagram is the most common representation for a poset.

A *linear extension* L of a poset $(P, <)$ is a total ordering $<_L$ of P such that $x < y$ implies $x <_L y$. A linear extension may be thought of as a total

ordering of P which is consistent with the partial ordering. It is also equivalent to a topological sort on the covering graph for P .

Let $Z(P)$ denote the number of linear extensions of $(P, <)$. P is a total order if $Z(P) = 1$. On the other hand, if $Z(P) = n!$, P contains no order relations at all. This number $Z(P)$ is a measure of the amount of order information that is present in the poset.

Example 1.2. Suppose P is the power set of $\{x_1, x_2, x_3\}$, that is, $P = \{\emptyset, \{x_1\}, \{x_2\}, \{x_3\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_1, x_2, x_3\}\}$. Then if the relation $<$ denotes proper subset, the three representations of the poset $(P, <)$ will be as shown in Figure 1.2. A linear extension of this poset can be in the order $\emptyset, \{x_1\}, \{x_2\}, \{x_3\}, \{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_1, x_2, x_3\}$. Another linear extension of the same poset can be in the order $\emptyset, \{x_2\}, \{x_1\}, \{x_1, x_2\}, \{x_3\}, \{x_1, x_3\}, \{x_2, x_3\}, \{x_1, x_2, x_3\}$. If we are patient enough to list all the linear extensions, we will find that there is a total of $Z(P) = 48$ different linear extensions of $(P, <)$.

The classical sorting problem can be generalized to two important forms:

- (1) *sorting a poset*, and
- (2) *producing a poset*.

1.2.1. Sorting Posets

In the first form of generalization, we are given some order relations between the elements of the given set before we sort it. In other words, we are given a poset P to start with. The task here is to obtain, via comparisons, the entire set of order relations, thus forming a linear extension of P . It is assumed

that any two elements of the set can be compared.

(a) Directed acyclic graph.

(b) Covering graph.

(c) Hasse diagram

Figure 1.2. Different representations of a poset.

For such a sorting problem, the number of outcomes is the number of linear extensions of the poset. This would be the minimum number of leaves in

any decision tree that solves the problem. The ITB would therefore be $\log Z(P)$.

This problem arises in a number of applications. A few typical examples of the problem follow.

Example 1.3. When we want to merge two sorted lists of elements, $x_1 < x_2 < \dots < x_n$ and $y_1 < y_2 < \dots < y_n$, into a single sorted list, the two sublists can be represented by a poset P whose Hasse diagram is of the form shown in Figure 1.3(a). The number of ways to merge the two sublists is the number of possible outcomes when sorting the poset; this number is

$$\binom{2n}{n}.$$

This will be the number of linear extensions $Z(P)$. Thus the ITB is

$$\log Z(P) = 2n - o(n),$$

where we employ Stirling's approximation. (Note that $f(x) = o(g(x))$ means that $f(x)/g(x) \rightarrow 0$ as $x \rightarrow \infty$.) The linear merge algorithm, which requires at most $2n - 1$ comparisons, comes close to achieving this bound.

Example 1.4. A heap is a completely balanced binary tree of depth δ in which every leaf is at level $\delta - 1$ or δ and all descendants of a node are smaller than it; in addition, all the leaves at level δ are as far to the left as possible. An example is given in Figure 1.3(b). The Hasse diagram representing the relations between the elements of the heap has the same shape as the heap itself. By using the ITB of a related problem in Example 1.9, the ITB for sorting this poset can be calculated to be $\Omega(n \log n)$ where n is the number of elements in the heap. We shall note that the heapsort algorithm

sorts a heap with $\sqrt{n}$ elements in $O(n \log n)$ steps.

Figure 1.3. Posets to be sorted.

Example 1.5. Given two sorted lists of equal number of elements, $x_1 < x_2 < \dots < x_n$ and $y_1 < y_2 < \dots < y_n$, the problem of sorting the n^2 sums $x_i + y_j$, $1 \leq i, j \leq n$ is equivalent to sorting the poset P given in Figure 1.3(c) if we use only the fact that $x_i + x_j < x_i + x_k$ for $k > j$ and $x_i + x_j < x_k + x_j$ for $k > i$ (together with their transitive consequences). Harper et al [HPSS75] showed that the number of linear extensions for this poset is

$$Z(P) = n! \cdot (\prod_{i=1, n-1} i!)^2 / \prod_{i=1, 2n-1} i!$$

and thus the lower bound for sorting it is

$$ITB \approx n^2 \log n.$$

However they also showed that if other expressions involving x_i 's and y_j 's are allowed, the number of possible orderings of the n^2 sums is less than n^{8n} .

Hence, for the problem of sorting these sums, we actually have

$$ITB < 8n \log n.$$

Recently, Kahn and Saks [KS84] showed that the information theoretic bound for sorting any poset P is tight up to a multiplicative constant. In fact, the number of comparisons needed to sort P lies between $\log Z(P)$ and $C \log Z(P)$ where C is approximately 2.17. So we have an upper bound based on $Z(P)$ as well.

1.2.2. Producing Posets

In the problem of producing posets, we are given a set $X = \{x_1, x_2, \dots, x_n\}$ with an unknown total order and we want to determine enough relations between its elements so that the set of relations obtained is a superset of those contained in a certain partial order P . In other words, we identify the elements

of P from those of X ; we map the elements of X bijectively to the elements of P .

In the process of identifying a poset, the algorithm may unavoidably determine more relations between the elements than is required. Each leaf of the corresponding decision tree will thus correspond to a poset Q whose set of relations between its elements is a superset of those in P . Because of this, it can be shown that $Z(Q)$ will be less than or equal to $Z(P)$. Let P_i be the poset represented by the i^{th} leaf for $1 \leq i \leq m$ where m is the number of leaves in the tree. Each total ordering of X will be a linear extension of one of the P_i 's. Hence

$$\sum_{i=1, m} Z(P_i) = n!$$

Because $Z(P_i) \leq Z(P)$, we have

$$m Z(P) \geq n!$$

or

$$m \geq n! / Z(P).$$

The information theoretic argument provides us with the lower bound of

$$\log (n! / Z(P))$$

comparisons to produce P .

Example 1.6. The problem of finding the maximum element x among a set of n elements is equivalent to identifying the poset given in Figure 1.4(a). In this case, $Z(P)$ is $(n-1)!$ (the number of arrangements of the $n-1$ unrelated elements) and the ITB is thus $\log (n! / (n-1)!) = \log n$. Here the information theoretic argument does not provide us with a good bound. The reason is that the maximum of any unordered set can be found with exactly $n-1$ comparisons and no less. The argument is simple. We have the n elements

comprising n components in the Hasse diagram when we start to produce P . Each comparison reduces the number of components by at most 1. We must end up with exactly one component in the Hasse diagram, otherwise there is more than one candidate for the maximum. Hence there must be at least $n - 1$ comparisons. An algorithm that performs a linear search for the maximum element will use exactly $n - 1$ comparisons.

Example 1.7. Finding the median element M of a set of n elements is equivalent to producing the poset in Figure 1.4(b) where the number of elements greater than M is equal to the number of those less than it. Here $Z(P)$ can be calculated to be the square of $((n - 1)/2)!$ and the ITB can be shown to be approximately n . Samuel Bent and John John [BJ85] showed a better lower bound of $2n + \Omega(\sqrt{n})$ by strengthening the information theoretic approach.

Example 1.8. There are times when both the maximum element x and the minimum element y are required. In this case, the problem is to produce the poset shown in Figure 1.4(c). Here $Z(P)$ is $(n - 2)!$ and thus we have the ITB as approximately $2 \log n$. In this case, the information theoretic argument does not provide a good bound. A better bound of $\lceil 3n/2 \rceil - 2$ can be shown by an adversary argument (see [P72] or [At77]). There also exists an algorithm that actually achieves this better bound (again, see [P72] or [At77]).

Example 1.9. In the construction of a heap on n elements, Gonnet and Munro [GM82] showed that the ITB is $1.3644...n + \Omega(\log n)$ by observing that

there are $Z(P) = n! / \prod t_i$ valid heaps on n numbers where t_i is the size of the heap rooted at node i . They also showed that $1.625n + O(\log n \log^* n)$ comparisons are sufficient. Here $\log^*$ denotes the number of times the logarithm (to the base 2) has to be taken before the quantity becomes zero or less. The information theoretic argument can be used to prove a similar result for the max-min heap of [ASSS84].

Some of these examples demonstrate that we should not always totally depend on the information theoretic arguments for good lower bounds.

Figure 1.4. Posets to be produced.

1.3. Research Objectives

The number $Z(P)$ is an important component used in the calculation of lower and upper bounds for the generalized sorting problems. It also comes into play in the design of new algorithms for sorting posets. Given a poset P , we can compare a pair of elements which are not related to each other and obtain a new poset Q where $Z(Q) < Z(P)$. We should do this for every unrelated pair, calculating the number of linear extensions left after each comparison, and select the pair that reduces the number of linear extensions by the greatest amount. By repeatedly applying the same procedure to the new poset that is selected, we can find a short sequence of comparisons to sort a given poset. Kahn and Saks [KS84] showed that for any partial order that is not a total order, there always exists a pair of elements to compare such that the proportion of linear extensions that 'remain' is between $3/11$ and $8/11$. This fact allowed for the upper bound of $2.17 \log Z(P)$ for sorting a poset P .

With $Z(P)$ playing important roles as described above, it is of considerable interest to be able to calculate it efficiently. In the examples given above, it looks as if $Z(P)$ can always be easily calculated using some simple formula or some simple procedure. However, no general polynomial time algorithm for this problem is known. In the special cases that the Hasse diagram H for P is a tree, or when H is formed by joining small graphs together with single edges, polynomial time algorithms have been given by Atkinson [At84b].

This problem of calculating the number of linear extensions is related to a geometrical problem [L84]. Given a poset $(P, <_P)$ on n elements, we can construct a corresponding n -dimensional convex polytope $C(P)$ as follows.

First we start with a cubical polytope $U = \{(x_1, x_2, \dots, x_n) \mid 0 \leq x_i \leq 1\}$. For every relation $x_i <_P x_j$ in P , we remove that part of U that lies in the half-space $x_i > x_j$. If P is a total order, $C(P)$ will be what is called a *simplex* which has a volume of $1/n!$. There exists an interesting relation between the linear extensions of P and the simplices that make up $C(P)$: Volume of $C(P) = Z(P)/n!$. The problem of calculating this volume is unfortunately a non-trivial one. As yet, no general formulae exists (otherwise our problem of calculating $Z(P)$ would have been solved!).

The main objective of this thesis is to obtain new polynomial time algorithms for various classes of partially ordered sets. A secondary objective is to obtain new lower and upper bound results on $Z(P)$ for these classes.

1.4. Overview of Thesis

Chapter 2 first describes a general algorithm that calculates the number of linear extensions for any poset but which has an exponential time complexity. Then it reports on polynomial time algorithms that appeared in the literature prior to this thesis. These algorithms, which work on certain classes of posets, are due to Atkinson [At84a, At84b] and Knuth [K73]. A new algorithm will be presented in Chapter 3 for posets whose Hasse diagrams are connected and contain cycles that are edge-wise disjoint. In Chapter 4, another new algorithm will be presented for posets whose Hasse diagrams can be minimally decomposed into a fixed number of chains. The stated time and space complexities of the algorithms presented in these chapters are worst-case complexities, with the assumption of an underlying random access machine model as described in [AHU74]. Chapter 5 will present an upper bound result

on $Z(P)$ for those posets P whose Hasse diagrams are connected. Finally, the conclusions arrived in this thesis are discussed in Chapter 6.

Chapter 2

RELATED WORK

In Chapter 1, we had discussed the calculation of the number of linear extensions, $Z(P)$, for some very specific posets P . However, we shall be more concerned in finding efficient algorithms for general classes of posets. This chapter will give a survey of algorithms that have been found for some classes prior to the research performed in this thesis.

2.1. Exponential Algorithms

Consider first how we can calculate $Z(P)$ for an arbitrary poset P with n elements. One easy way is to generate all the permutations or arrangements on the n elements and check each of them for consistency with P . This algorithm is exponential in its time complexity since there will be $n!$ permutations. It will only be suitable for posets with small numbers of elements.

We should try to make use of the relations present in P to obtain a more efficient algorithm. They can be used to decompose the problem into subproblems which are easier to solve. To do this, we require the following definitions. An element x in a poset P is *maximal* if there are no elements y in P such that $x < y$. An element x in a poset P is the *maximum element* if $y < x$ for every element y in P . A maximum element is maximal but a maximal element need not necessarily be maximum. *Minimal* and *minimum* elements are similarly defined.

Given a poset P with n nodes, Algorithm 2.1 calculates $Z(P)$ in terms of subposets with $n - 1$ nodes. The algorithm first computes the set of minimal elements $\{\alpha_1, \alpha_2, \dots, \alpha_m\}$ and the set of maximal elements $\{\beta_1, \beta_2, \dots, \beta_M\}$. Then if $m < M$, it will make m recursive calls with α_i deleted from P in the i^{th} call. Otherwise it will make M recursive calls with β_i deleted from P in the i^{th} call. The results returned by these calls will be summed to obtain $Z(P)$.

```

function  $Z(P)$ 
  if  $|P| = 1$  then
     $Z \leftarrow 1$ 
  else
    let  $\{\alpha_1, \alpha_2, \dots, \alpha_m\}$  be the set of minimal elements
    let  $\{\beta_1, \beta_2, \dots, \beta_M\}$  be the set of maximal elements
    if  $m < M$  then
       $Z \leftarrow \sum_{i=1, m} Z(P - \{\alpha_i\})$ 
    else
       $Z \leftarrow \sum_{i=1, M} Z(P - \{\beta_i\})$ 
    end if
  end if
end function

```

Algorithm 2.1. General but exponential algorithm.

The correctness of the algorithm is based on the observation that each linear extension of P begins with one of the α_i 's followed by a linear extension of $P - \{\alpha_i\}$. Alternatively it begins with a linear extension of $P - \{\beta_i\}$.

and ends with β_i . In each call, we use the lesser of the minimal elements and maximal elements because this helps to keep the number of recursive calls at a minimum.

The algorithm has an exponential time complexity because in the worst case, where we have no order relations whatsoever between the elements, the number of recursive calls will be $n!$. However, for any other poset, it would be better than the first approach. In addition, the following lemma can be applied to improve the running time at certain levels of the recursion.

Before proceeding any further, we should discuss the space required to represent a poset. Unless otherwise specified, we assume that the Hasse diagram will be the standard representation for a given poset. The Hasse diagram can be stored with $O(n^2)$ space by using any standard graph representations such as the adjacency matrix or adjacency lists. Thus in any space complexity analysis for the algorithms in this thesis, it will be assumed that a minimum amount of $O(n^2)$ space is required.

LEMMA 2.1. If the poset P can be split into two disjoint posets P_1 and P_2 , then

$$Z(P) = Z(P_1) Z(P_2) \binom{|P|}{|P_1|}.$$

Proof. Since P_1 and P_2 are disjoint, the linear extensions of P are precisely the mergings of the linear extensions L_1 of P_1 with the linear extensions L_2 of P_2 . The number of ways in which a pair L_1, L_2 can be merged is

$$\binom{|P|}{|P_1|}$$

and the number of such pairs is

$$Z(P_1) Z(P_2).$$

The number of linear extensions of P is the product of these two numbers.

With two disjoint posets, there are at least two minimal and two maximal elements. Using Algorithm 2.1 plainly as it is will make two or more recursive calls. But this lemma, when applied, creates only two such calls. Furthermore, if there is more than one element in each disjoint poset, the number of elements decreases by more than one in each of the two calls.

Example 2.1. Figure 2.1 demonstrates how the improved algorithm works on the poset given in box (a). The number of minimal and maximal elements are respectively 2 and 3. Two recursive calls are thus made, with one minimal element removed in each call. Consequently, we have the posets shown in boxes (b) and (c). In box (b), we have two disjoint posets and Lemma 2.1 is applied there. The lemma is also applied in boxes (f), (g), (h), (k), (s) and (t). In each of these boxes, the calculation shown corresponds directly to the equation stated in the lemma. In all the other boxes, the calculations follow from Algorithm 2.1. The terminating condition is when there is only one element in the poset for which the number of linear extensions is obviously 1; no recursive calls are made here.

2.2. Special Cases

Sometimes the underlying undirected graph of the Hasse diagram for a poset P may be formed by joining small graphs together with single edges. At

Figure 2.1. Illustration of how the Improved algorithm works.

other times, it is a tree on n vertices. Polynomial time algorithms have been produced by Atkinson [At84b] to calculate $Z(P)$ for these special cases. They take into consideration the positions of certain elements in the linear extensions of P . We refer to these positions as the *ranks* of the elements. If an element y appears as the r^{th} element in a linear extension L , then its rank in L is r . We shall write

$$Z(P; \text{rank}(y_1) = r_1, \text{rank}(y_2) = r_2, \dots, \text{rank}(y_k) = r_k)$$

to denote the number of linear extensions of P with $y_1, y_2, \dots, y_k$ having ranks $r_1, r_2, \dots, r_k$ where $1 \leq k \leq n$.

Atkinson proved the following result that forms the basis of the algorithms. It is similar in nature to that of Lemma 2.1. However, the computations are now in terms of a partitioning of the linear extensions according to the ranks of some elements.

LEMMA 2.2. Let $P = P_1 \cup P_2$ where P_1 and P_2 are two disjoint posets of p and q elements respectively. Also, let α, β be two (not necessarily distinct) elements in P_1 and γ an element in P_2 . Then defining

$$W_{rst} = Z(P; \text{rank}(\alpha) = r, \text{rank}(\beta) = s, \text{rank}(\gamma) = t)$$

and writing

$$Z^1_{ab} = Z(P_1; \text{rank}(\alpha) = a, \text{rank}(\beta) = b) \text{ and}$$

$$Z^2_c = Z(P_2; \text{rank}(\gamma) = c)$$

we have

$$W_{rst} = \sum Z^1_{ab} Z^2_c \binom{r-1}{a-1} \binom{s-r-1}{b-a-1} \binom{t-s-1}{t-b-c} \binom{p+q-t}{p-t+c} \quad \text{if } r \leq s < t,$$

$$W_{rst} = \sum Z^1_{ab} Z^2_c \binom{r-1}{a-1} \binom{t-r-1}{t-a-c} \binom{s-t-1}{b+c-t-1} \binom{p+q-s}{p-b} \quad \text{if } r < t \leq s,$$

$$W_{rst} = \sum Z^1_{ab} Z^2_c \binom{s-1}{b-1} \binom{r-s-1}{a-b-1} \binom{t-r-1}{t-a-c} \binom{p+q-t}{p-t+c} \quad \text{if } s \leq r < t,$$

$$W_{rst} = \sum Z^1_{ab} Z^2_c \binom{s-1}{b-1} \binom{t-s-1}{t-b-c} \binom{r-t-1}{a+c-t-1} \binom{p+q-r}{p-a} \quad \text{if } s < t < r,$$

$$W_{rst} = \sum Z^1_{ab} Z^2_c \binom{t-1}{t-c} \binom{r-t-1}{a+c-t-1} \binom{s-r-1}{b-a-1} \binom{p+q-s}{p-b} \quad \text{if } t < r \leq s,$$

$$W_{rst} = \sum Z^1_{ab} Z^2_c \binom{t-1}{t-c} \binom{s-t-1}{b+c-t-1} \binom{r-s-1}{a-b-1} \binom{p+q-r}{p-a} \quad \text{if } t < s \leq r,$$

where the summation is over all a, b, c with $a \leq r, b \leq s, c \leq t$ in each case.

Proof. The proof will be given only for the case where $r \leq s < t$ since the other five cases are symmetrical to it. As stated in Lemma 2.1, the linear extensions of P are precisely the mergings of the linear extensions L_1 of P_1 with the linear extensions L_2 of P_2 . A merging of two extensions L_1 and L_2 which produces the correct ranks for α, β, γ will have α, β of ranks a, b in L_1 and γ of rank c in L_2 for some $a \leq r, b \leq s, c \leq t$. (Note that if a or b is greater than p or if c is greater than q , then Z^1_{ab} or Z^2_c is zero respectively because the rank cannot be greater than the number of elements.) The merging can be represented diagrammatically as in Figure 2.2 which shows explicitly the lengths of the segments between which α, β, γ must be inserted in the other extension. The matching segments can be merged in any way. Therefore for a fixed a, b, c the number of orderings of P giving the required ranks for α, β, γ is

$$Z^1_{ab} Z^2_c \binom{r-1}{a-1} \binom{s-r-1}{b-a-1} \binom{t-s-1}{t-b-c} \binom{p+q-t}{p-t+c}.$$

Summing this expression over all possible a, b, c gives the required result.

Figure 2.2. Merging of two linear extensions.

Using the same notation as given above, the following corollary was also derived by Atkinson to take into account an edge between β and γ (connecting the two disjoint posets). As we shall see later, this allows us to break the problem of calculating $Z(P)$ down into smaller problems.

COROLLARY 2.1. Let $P = P_1 \cup P_2$ be the poset containing all the order relations of P_1 and P_2 together with an order relation between β and γ . Then

$$Z(P; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t) = \begin{cases} \sum_{s < t} W_{rst}, & \text{if } \beta < \gamma \text{ and} \\ \sum_{s > t} W_{rst}, & \text{if } \gamma < \beta. \end{cases}$$

In particular, the numbers $Z(P; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t)$, $r, t = 1, 2, \dots, |P|=n$, can be calculated from the numbers Z^1_{ab} , $a, b = 1, 2, \dots, p$, and Z^2_c , $c = 1, 2, \dots, q$, in $O(n^6)$ steps and in $O(n^4)$ in the special case where $\alpha = \beta$.

Proof. In any linear extension of P , $\beta < \gamma$, or $\gamma < \beta$, if and only if $\text{rank}(\beta) < \text{rank}(\gamma)$, or $\text{rank}(\gamma) < \text{rank}(\beta)$ respectively. Hence we sum the corresponding W_{rst} values. For the next part, observe that each of the $O(n^3)$ numbers W_{rst} requires $O(n^3)$ steps to compute. In the special case where $\alpha = \beta$, we have r

$= s$ whenever $W_{rst} \neq 0$ and $a = b$ whenever $Z^1_{ab} \neq 0$. Hence there are only $O(n^2)$ numbers W_{rst} to compute in this case and each of them requires $O(n^2)$ steps.

APPLICATION 2.1. Hasse Diagram is a String of Graphs

Suppose we have a family of graphs $\{G_1, G_2, \dots, G_n\}$ where the size of each graph is bounded by a constant k independent of n . In each graph G_i , let λ_i and δ_i be two of its vertices; they need not necessarily be distinct. Then let G be the graph formed by connecting all the graphs G_i with edges between δ_i and λ_{i+1} , $i = 1, 2, \dots, n-1$, as shown in Figure 2.3.

Figure 2.3. Hasse diagram as a string of bounded graphs.

Let P be any poset whose Hasse diagram, regarded as an undirected graph, is isomorphic to G . Algorithm 2.2 calculates $Z(P)$ in polynomial time. It iteratively applies Corollary 2.1 by letting P_2 be the poset formed by the vertices of $G_1 \cup G_2 \cup \dots \cup G_i$ and P_1 the poset formed by the vertices of G_{i+1} , and letting $\gamma = \delta_i$, $\beta = \lambda_{i+1}$ and $\alpha = \delta_{i+1}$, for $i = 1, 2, \dots, n-1$. The numbers Z^1_{ab} can be computed in constant time since each P_1 is bounded. This is also true for the numbers Z^2_c in the first iteration. In each iteration $i \geq 2$, the numbers Z^2_c come from the previous iteration. In iteration $i-1$, we calculate (using the corollary) the numbers $Z(Q; \text{rank}(\alpha=\delta_i) = r, \text{rank}(\gamma=\delta_{i-1}) = t)$ where Q is the

(* P_{1i} is the poset formed by the vertices of G_{i+1} *)

(* P_{2i} is the poset formed by the vertices of $G_1 \cup G_2 \cup \dots \cup G_i$ *)

Calculate $Z(P_{21}; \text{rank}(\delta_1)=c)$, $1 \leq c \leq |P_{21}|$, using a method such as Alg. 2.1

for $i \leftarrow 1$ to $n-1$ do

$P_1 \leftarrow P_{1i}$; $\alpha \leftarrow \delta_{i+1}$; $\beta \leftarrow \lambda_{i+1}$

$P_2 \leftarrow P_{2i}$; $\gamma \leftarrow \delta_i$

Calculate Z^1_{ab} , $1 \leq a, b \leq |P_1|$ using a method such as Algorithm 2.1

(* The Z^2_c numbers were calculated in the previous iteration *)

(* Note that $Z^1_{ab}=0$ if $a > |P_1|$ or $b > |P_1|$ and $Z^2_c=0$ if $c > |P_2|$ *)

Use Corollary 2.1 to calculate the numbers

$Z(P_{2,i+1}; \text{rank}(\delta_{i+1}) = r, \text{rank}(\delta_i) = t)$

from the Z^1_{ab} and Z^2_c numbers.

(* Calculate Z^2_c numbers of next iteration *)

for $r \leftarrow 1$ to $|P_{2,i+1}|$ do

$S \leftarrow 0$

for $t \leftarrow 1$ to $|P_{2,i+1}|$ do

$S \leftarrow S + Z(P_{2,i+1}; \text{rank}(\delta_{i+1}) = r, \text{rank}(\delta_i) = t)$

end for

$Z(P_{2,i+1}; \text{rank}(\delta_{i+1}) = r) \leftarrow S$

end for

end for

$S \leftarrow 0$

for $r \leftarrow 1$ to $|P_{2,n}|$ do

$S \leftarrow S + Z(P_{2,n}; \text{rank}(\delta_n) = r)$

end for

$Z(P) \leftarrow S$

Algorithm 2.2. Computing $Z(P)$ for string of graphs.

Iteration 1: Iteration 2: Iteration i: Iteration i+1:

Figure 2.4. Numbers calculated in each iteration.

poset formed by the vertices of $P_1 \cup P_2$. These numbers are summed over all t to obtain the numbers $Z(Q; \text{rank}(\delta_i) = r)$ which become the Z_c^2 numbers for the next iteration.

The numbers manipulated in each iteration are shown in Figure 2.4. Note that α of iteration $i-1$ becomes γ of iteration i . With $|G_i| \leq k$ for each i , each of the $n-1$ iterations requires $O((kn)^6) = O(n^6)$ steps. After the last iteration, we sum the numbers $Z(P; \text{rank}(\delta_n) = r)$ in $O(kn)$ steps, to obtain $Z(P)$. The total time complexity is thus $O(n^7)$. The amount of space to store the three sets of numbers (Z_{ab}^1 , Z_c^2 and $Z(Q; \text{rank}(\delta_{i+1}) = r, \text{rank}(\delta_i) = t)$) in each iteration is $O((kn)^2 + kn + (kn)^2)$. Assuming the same memory locations are used for these numbers in all iterations, the total space complexity is $O(n^2)$.

APPLICATION 2.2. Hasse Diagram is a Tree

It is common to have the Hasse diagram in the form of a tree T , in the sense that the underlying undirected graph is connected and has no cycles. Let α be any node in T and consider T to be rooted at α . Then let γ be any son of α in this tree. We can take T to be consisting of two subtrees T_1 and T_2 , rooted at α and γ respectively, together with an edge between α and γ as shown in Figure 2.5. With these notations in mind, Algorithm 2.3 computes, in polynomial time, the numbers $Z(P; \text{rank}(\alpha) = r)$, $1 \leq r \leq |P|=n$. We sum these numbers to obtain $Z(P)$.

The logic behind the procedure is as follows. Let P_1 and P_2 be the posets whose Hasse diagrams are T_1 and T_2 respectively. If the numbers $Z(P_1; \text{rank}(\alpha) = a)$ and $Z(P_2; \text{rank}(\gamma) = c)$ are known for all a and c , then Corollary 2.1 can be used (with $\alpha = \beta$ and $a = b$) to calculate the numbers

$Z(P; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t)$ in $O(n^4)$ steps. Then each number $Z(P; \text{rank}(\alpha) = r)$ is just a sum of $Z(P; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t)$ over all t . The numbers $Z(P_1; \text{rank}(\alpha) = a)$ and $Z(P_2; \text{rank}(\gamma) = c)$ can be calculated using the same process recursively.

Figure 2.5. Hasse diagram in the form of a tree.

procedure Tree (P, α, Z)

(* $Z[r] = Z(P; \text{rank}(\alpha) = r)$ *)

if $|P| = 1$ **then**

$Z[1] \leftarrow 1$

else

let P_1 be the poset formed by the vertices of T_1

let P_2 be the poset formed by the vertices of T_2

call Tree (P_1, α, Z_1) (* $Z_1[a] = Z^1_{ab}$ *)

call Tree (P_2, γ, Z_2) (* $Z_2[c] = Z^2_c$ *)

```

for a ← |P1| + 1 to |P| do
  Z1[a] ← 0
end for
for c ← |P2| + 1 to |P| do
  Z2[c] ← 0
end for

```

Use Corollary 2.1 to calculate the numbers

$Z(P; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t)$
 from the Z^T_{ab} and Z^2_c numbers.

```

for r ← 1 to |P| do
  S ← 0
  for t ← 1 to |P| do
    S ← S + Z(P; rank(α) = r, rank(γ) = t)
  end for
  Z[r] ← S
end for
end if
end procedure

```

Algorithm 2.3. Computing $Z(P)$ for a tree.

At each call to the procedure, we remove the edge, if any, between the roots of two subtrees and make two recursive calls. Hence with $n-1$ edges in the tree, the total number of recursive calls is $2n-2$. Together with the initial call, there is a total of $2n-1$ calls to the procedure. For n of the calls, there will only be one node in the tree and the time taken in each of them is $O(1)$. For the rest, the major computations are those stated in the corollary which take $O(n^4)$ steps. The total time taken to calculate all numbers is therefore $O(n^5)$.

In each call to the procedure, $O(2n)$ space is needed to store the Z^1_{ab} and Z^2_c numbers, and $O(n^2)$ space is needed to store the $Z(P; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t)$ numbers. With a total of $2n-1$ calls, the total amount of space used to compute $Z(P)$ is $O(n^3)$.

Example 2.2. The Hasse diagram shown in Figure 2.6(a) has the form of a tree. We can view the corresponding covering graph as a tree rooted at x_1 as shown in Figure 2.6(b). Let us put $\alpha = \beta = x_1$ and $\gamma = x_5$. Then we have the two subtrees as shown in part (c) of the figure. By recursion, we would have found that

$$\begin{array}{ll} Z^1_{11} = Z(P_1; \text{rank}(x_1) = 1) = 0 & Z^2_1 = Z(P_2; \text{rank}(x_5) = 1) = 0 \\ Z^1_{22} = Z(P_1; \text{rank}(x_1) = 2) = 0 & Z^2_2 = Z(P_2; \text{rank}(x_5) = 2) = 0 \\ Z^1_{33} = Z(P_1; \text{rank}(x_1) = 3) = 2 & Z^2_3 = Z(P_2; \text{rank}(x_5) = 3) = 1 \\ Z^1_{44} = Z(P_1; \text{rank}(x_1) = 4) = 3 & Z^2_4 = Z(P_2; \text{rank}(x_5) = 4) = 1. \end{array}$$

It is only by coincidence that T_1 and T_2 have the same number of elements; in general, these numbers need not be the same. To calculate the W_{rst} values, we would also need the following numbers which we set to zero: $Z^1_{55}, Z^1_{66}, Z^1_{77}, Z^1_{88}, Z^2_5, Z^2_6, Z^2_7, Z^2_8$.

Since $\alpha = \beta < \gamma$ ($r = s < t$), we need only the first formula of Lemma 2.2 to calculate the W_{rst} values. With $\alpha = \beta$ ($r = s, a = b$), the formula reduces to

$$W_{rrt} = \sum_{a \leq r, c \leq t} Z^1_{aa} Z^2_c \binom{r-1}{a-1} \binom{t-r-1}{t-a-c} \binom{p+q-t}{p-t+c}.$$

The calculated values are shown in Table 2.1. Summing the elements of each row of the table gives us:

$$\begin{aligned} Z(P; \text{rank}(x_1) = 1) &= 0 \\ Z(P; \text{rank}(x_1) = 2) &= 0 \\ Z(P; \text{rank}(x_1) = 3) &= 20 \\ Z(P; \text{rank}(x_1) = 4) &= 54 \end{aligned}$$

$$\begin{aligned} Z(P; \text{rank}(x_1) = 5) &= 96 \\ Z(P; \text{rank}(x_1) = 6) &= 100 \\ Z(P; \text{rank}(x_1) = 7) &= 60 \\ Z(P; \text{rank}(x_1) = 8) &= 0. \end{aligned}$$

Summing these numbers in turn gives us $Z(P) = 330$.

(a) Hasse diagram

(b) Covering graph T

(c) Subtrees

Figure 2.6. Poset whose covering graph is a tree.

$r \backslash t$	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0
2		0	0	0	0	0	0
3			0	0	4	8	8
4				0	12	21	21
5					24	36	36
6						50	50
7							60

Table 2.1. W_{rst} values (with $r = s$).

2.3. Zig-Zag Patterns

In this section, we shall look at posets $P_n = \{x_1, x_2, \dots, x_n\}$ with Hasse diagrams having a fixed zig-zag pattern. Two examples are shown in Figure 2.7. The nodes are linked together in a chain where the direction of each link on the chain can be up or down. These posets are special cases of the two types of posets discussed in the previous section. The algorithms given there reduce to the following recurrence formulae for the zig-zags. Atkinson [At84a] had independently derived these formulae.

Figure 2.7. Two zig-zag diagrams.

Let $Z_{nr} = Z(P_n; \text{rank}(x_n) = r)$. Its calculation is performed as follows.

(1) $Z_{11} = 1$.

(2) If the last link of P_n is an up link then

$$Z_{nr} = \sum_{s=1, r-1} Z_{n-1, s}$$

(3) If the last link of P_n is a down link then

$$Z_{nr} = \sum_{s=r, n-1} Z_{n-1, s}$$

The rationale behind these recurrences is quite simple. Each linear extension $L_n = \alpha_1 \alpha_2 \dots \alpha_n$ of P_n with $\alpha_r = x_n$ can be uniquely obtained from a linear extension $L_{n-1} = \beta_1 \beta_2 \dots \beta_{n-1}$ of P_{n-1} , where $\beta_i = \alpha_i$ for $i < r$ and $\beta_i = \alpha_{i+1}$ for $i \geq r$, by inserting x_n between β_{r-1} and β_r . If the last link is directed upwards, then $x_{n-1} < x_n$ which means that in L_n , x_{n-1} lies to the left of x_n and $\text{rank}(x_{n-1}) < r$. In this case, the number of elements lying to the left of x_{n-1} is the same in both L_{n-1} and L_n . Thus, $\text{rank}(x_{n-1}) = s < r$ in L_{n-1} . If the last link is directed downwards, then $x_n < x_{n-1}$ and thus x_n lies to the left of x_{n-1} implying that $\text{rank}(x_{n-1}) > r$ in L_n . The number of elements lying to the left of x_{n-1} is thus one less in L_{n-1} than it is in L_n . Hence, $\text{rank}(x_{n-1})$ in L_{n-1} is $s > r-1$ or $s \geq r$.

$Z(P_n)$ is just the sum $\sum_{r=1, n} Z_{nr}$. Atkinson suggested the following algorithm to calculate this number. The algorithm constructs a triangular array similar to the Pascal triangle. There will be n rows with row i having i numbers: $Z_{i1}, Z_{i2}, \dots, Z_{ii}$. From the first recurrence formula, the first row of the array will be a single 1. From the other formulae, the elements in row i will be formed by summing the elements of the previous row in a direction determined by the $i-1^{\text{th}}$ link. If the link is an up link, the summation of the elements of the previous row will be done from left to right. Otherwise, it will be done from right

to left. In these respects, the formulae can be equivalently expressed as follows.

(1) $Z_{11} = 1.$

(2) If the last link of P_n is an up link then

$$Z_{n1} = 0 \text{ and } Z_{nr} = Z_{n,r-1} + Z_{n-1,r-1} \text{ for } 1 < r \leq n.$$

(3) If the last link of P_n is a down link then

$$Z_{nn} = 0 \text{ and } Z_{nr} = Z_{n,r+1} + Z_{n-1,r} \text{ for } n > r \geq 1.$$

The final row is summed to give $Z(P_n)$. The amount of time or space required by this algorithm is clearly $O(n^2)$.

Example 2.3. For the regular (up-down) zig-zag shown in Figure 2.7(a), the triangular array computed by the algorithm will be:

$$\begin{array}{cccccc} & & & & 1 & & \\ & & & & 0 & & 1 \\ & & & 1 & 1 & & 0 \\ & & 0 & 1 & 2 & 2 & \\ & 5 & 5 & 4 & 2 & 0 & \\ 0 & 5 & 10 & 14 & 16 & 16 & \end{array}$$

The sums are calculated alternatively leftwards and rightwards as we go from line to line. The number of linear extensions for the poset is $Z(P_6) = 0 + 5 + 10 + 14 + 16 + 16 = 61$. For the zig-zag shown in Figure 2.7(b), the following triangle will be computed.

$$\begin{array}{cccccc} & & & & 1 & & \\ & & & & 0 & & 1 \\ & & 0 & & 1 & & 1 \\ & 0 & 0 & 1 & 2 & & \\ & 3 & 3 & 3 & 2 & 0 & \\ 0 & 3 & 6 & 9 & 11 & 11 & \end{array}$$

The sum of the last row, $0 + 3 + 6 + 9 + 11 + 11 = 40$, is $Z(P_6)$.

For the case of the regular zig-zag, an old result of André [An1879] tells us that the exponential generating function of the numbers $Z(P_n)$ is $\sec x + \tan x$, that is,

$$\sum_{n=0, \infty} (Z(P_n) x^n) / n! = \sec x + \tan x.$$

Atkinson noted that it can be shown that $Z(P_n) = O(n! 2^{n+2} / \pi^{n+1})$ by using the properties of secant and tangent coefficients [F76].

The case of the circular zig-zag has also been discussed by Atkinson. Each maximal element (or minimal element) can be the maximum (minimum). Hence, we shall remove each of them in turn and calculate the number of linear extensions for each resulting poset. These numbers are then summed to obtain $Z(P_n)$.

2.4. Merging with Single Constraint

Knuth [K73] considered the calculation of $Z(P)$ for posets with Hasse diagrams of the form shown in Figure 2.8. One can treat $Z(P)$ as the number of mergings of two sorted lists, $L_1 = x_1 < x_2 < \dots < x_m$ and $L_2 = y_1 < y_2 < \dots < y_n$, subject to the constraint $x_p < y_q$.

There are q possible positions to insert x_p in L_2 : $y_i < x_p < y_{i+1}$, $0 \leq i < q$. Figure 2.9 shows x_p in one of these positions. There are now four sublists to be considered: L_{11} , L_{12} , L_{21} and L_{22} . The number of linear extensions with x_p positioned between y_i and y_{i+1} is the number of ways to merge L_{11} and L_{21} multiplied by the number of ways to merge L_{12} and L_{22} :

$$Z(F; y_i < x_p < y_{i+1}) = \binom{m-p+n-i}{m-p} \binom{p-1+i}{p-1}.$$

The total number of linear extensions is thus

$$Z(P) = \sum_{0 \leq i < q} Z(P; y_i < x_p < y_{i+1})$$

A symmetrical argument can be used to show that

$$Z(P) = \sum_{p \leq j \leq m} Z(P; x_j < y_q < x_{j+1})$$

where

$$Z(P; x_j < y_q < x_{j+1}) = \binom{n-q+m-j}{n-q} \binom{q-1+j}{q-1}$$

Figure 2.8. Poset with two lists and one constraint.

Figure 2.9. Four sublists to be merged pairwise.

Chapter 3

CYCLES IN THE HASSE DIAGRAM

By the definition of a poset, there cannot be any directed cycles in the covering graph. Otherwise the antireflexive, antisymmetric and transitive laws will not simultaneously hold. However, it is possible to have undirected cycles in the Hasse diagram when we ignore the directions of the edges. A simple example is shown in Figure 3.1.

In this chapter, we shall consider those Hasse diagrams which are connected and all of whose cycles (if any) are edge-wise disjoint (but not necessarily vertex-disjoint). For these diagrams, we present a polynomial time algorithm to calculate the number of linear extensions of the posets they represent.

Covering graph

Corresponding
Hasse diagram

Figure 3.1. Undirected cycle in the Hasse diagram.

3.1. Case of One Cycle

Let us first present an algorithm to handle those Hasse diagrams that contain only one cycle. The simple discussion will demonstrate clearly how Corollary 2.1 of the previous chapter can be applied to derive other algorithms. It should also ease our understanding of the material in the next section where we consider Hasse diagrams with many cycles.

If the Hasse diagram H for a poset P contains a single cycle $(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{k-1}, \alpha_k, \alpha_1)$, we may view the undirected graph underlying H as a cycle of trees as shown in Figure 3.2. Each node α_i on the cycle is the root of a tree T_i , $i = 1, 2, \dots, k$. The trees are linked together with edges between α_j and α_{j+1} , $j = 1, 2, \dots, k-1$, and an edge between α_k and α_1 .

Figure 3.2. Connected Hasse diagram with one cycle.

The proof of the following theorem gives us an algorithm which enables us to calculate $Z(P)$ in polynomial time.

THEOREM 3.1. Let P be a poset with n nodes. If the underlying undirected graph of its Hasse diagram consists of a set of trees T_i with roots α_i , $i = 1, 2, \dots, k$, which are linked together by edges (α_i, α_{i+1}) , $i = 1, 2, \dots, k-1$, and edge (α_k, α_1) , then $Z(P; \text{rank}(\alpha_1) = r)$, $r = 1, 2, \dots, n$, can be calculated in time $O(n^7)$ and space $O(n^3)$.

Proof. We shall give a procedure to calculate, in time $O(n^7)$ and space $O(n^3)$, the numbers $Z(R; \text{rank}(\alpha_1) = r, \text{rank}(\alpha_k) = t)$ for the poset R whose Hasse diagram is the same as that for P except that the edge between α_1 and α_k is removed. Once these numbers are available, we can obtain the required numbers, in time $O(n^2)$, by using the following summation for each value of r .

$$\begin{aligned} Z(P; \text{rank}(\alpha_1) = r) &= \sum_{r < t} Z(R; \text{rank}(\alpha_1) = r, \text{rank}(\alpha_k) = t) && \text{if } \alpha_1 < \alpha_k, \text{ and} \\ &= \sum_{r > t} Z(R; \text{rank}(\alpha_1) = r, \text{rank}(\alpha_k) = t) && \text{if } \alpha_k < \alpha_1 \end{aligned}$$

which takes into account the edge between α_1 and α_k in P .

The procedure is as follows. We iteratively apply Corollary 2.1 $k-1$ times. In iteration i , we let P_1 be the poset formed by the vertices of $T_1 \cup T_2 \cup \dots \cup T_i$, P_2 be the poset formed by the vertices of T_{i+1} , $\alpha = \alpha_1$, $\beta = \alpha_i$, and $\gamma = \alpha_{i+1}$. (Note that α is fixed to be α_1 in all the iterations and $\alpha = \beta$ in the first iteration.) For each iteration, Corollary 2.1 is used to calculate the numbers

$Z(Q; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t)$ from the numbers Z^1_{ab} and Z^2_c for the poset Q formed by the vertices of $P_1 \cup P_2$.

P_2 is a tree in each iteration and thus the quantities $Z^2_c = Z(P_2; \text{rank}(\alpha_{i+1}) = c)$, $c = 1, 2, \dots, |P_2|$, can be computed in time $O(n^5)$ using Algorithm 2.3. In the first iteration, the quantities $Z^1_{ab} = Z(P_1; \text{rank}(\alpha = \beta = \alpha_1) = a = b)$, $a = b = 1, 2, \dots, |P_1|$, can also be computed in time $O(n^5)$ for the same reason. In iteration $i \geq 2$, Z^1_{ab} are those numbers $Z(Q; \text{rank}(\alpha_1) = a, \text{rank}(\alpha_i) = b)$ calculated by the corollary in the previous iteration. From Corollary 2.1, the set of numbers $Z(Q; \text{rank}(\alpha_1) = r, \text{rank}(\alpha_{i+1}) = t)$, $r, t = 1, 2, \dots, |Q|$, can be calculated in $O(n^4)$ steps when $i = 1$ and in $O(n^6)$ steps when $i \geq 2$. Thus each of the k iterations requires $O(n^5 + n^6)$ time and, since $k \leq n$, the total time required for all iterations is $O(n^7)$.

In each iteration, Algorithm 2.3 requires $O(n^3)$ space to compute the Z^2_c numbers (and the Z^1_{ab} numbers in the first iteration). Besides this, we need $O(n^2)$ space for the Z^1_{ab} numbers and $O(n^2)$ space for the $Z(Q; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t)$ numbers. The total amount of space required at any one iteration is thus at most $O(n^3)$. Assuming the same locations were used for these numbers in every iteration, the entire procedure requires just $O(n^3)$ space.

The algorithm builds up the Hasse diagram for P by joining an adjacent tree in each iteration. The intermediate Hasse diagram obtained in iteration i is thus a string of $i+1$ trees. This is illustrated in Figure 3.3. A set of numbers, S_i , is calculated for this Hasse diagram. Each of them is the number of linear extensions of the represented poset, with α_1, α_{i+1} each having a certain rank. The calculation of these numbers requires us to first calculate the number of

Iteration 1: Iteration 2: Iteration i: Iteration i+1: After iteration k-1:

Figure 3.3. Iterations in the tree algorithm.

linear extensions of the poset whose Hasse diagram is the added tree T_{i+1} , for every rank of its root α_{i+1} . These latter numbers are calculated using Algorithm 2.3. Then they and the numbers in S_{i-1} are used in the computations shown in Corollary 2.1 (and Lemma 2.2) to obtain the numbers in S_i .

In the last iteration, all the trees would have been added to our intermediate Hasse diagram and they represent the poset R . To obtain P , we include the edge e joining α_k and α_1 . The inclusion of this edge will serve to eliminate those numbers in S_{k-1} where the ranks of α_1 and α_k are inconsistent with the order relation represented by e . To obtain the number of linear extensions of P for a certain rank r of α_1 , we just add all those numbers in S_{k-1} with $\text{rank}(\alpha_k) > r$ if $\alpha_1 < \alpha_k$ and we add all those numbers with $\text{rank}(\alpha_k) < r$ if $\alpha_k < \alpha_1$.

We can now find $Z(P)$ in another $O(n)$ steps by summing the numbers $Z(P; \text{rank}(\alpha_1) = r)$ for all r .

3.2. Case of Many Cycles

In this section, we shall consider the general case when the Hasse diagram is connected and all its cycles are edge-wise disjoint. An example of how the cycles may be connected is shown in Figure 3.4(a). The cycles are connected together by paths, each of which can be empty or consists of a single edge or a sequence of edges. Though not shown in the figure, there may be trees rooted at each node in the cycles or paths. Figure 3.4(b) gives an illustration of these trees for the case of two cycles, $C_\lambda = \{\lambda_1, \lambda_2, \dots, \lambda_h\}$ and $C_\delta = \{\delta_1, \delta_2, \dots, \delta_k\}$, linked by a path $\Pi = \{\pi_1, \pi_2, \dots, \pi_m\}$. Each node on the cycles and path is the root of its own tree. Note that we do not consider Π as part of

the tree for λ_1 or δ_1 . In other words, Π belongs to neither T^{λ_1} nor T^{δ_1} .

(a) A Hasse diagram with many cycles.

(b) Hasse diagram with two disjoint cycles.

Figure 3.4. Connected Hasse diagrams with disjoint cycles.

The increase in the number of cycles increases the difficulty substantially but the same basic principles apply. The idea here is to notice that any node α in the Hasse diagram either belongs to a cycle or it does not. If α belongs to a cycle $(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_{k-1}, \alpha_k, \alpha_1)$ with $\alpha_1 = \alpha$, the Hasse diagram can be viewed as the undirected graph G shown in Figure 3.5(a) where each of the subgraphs $G_1, G_2, \dots, G_k$ is connected and all its cycles are edge-wise disjoint. If α does not belong to a cycle then the Hasse diagram can be viewed as the undirected graph G shown in Figure 3.5(b) where each edge incident on α links it to a subgraph of G . Again, each of the subgraphs $G_1, G_2, \dots, G_k$ is connected and all its cycles are edge-wise disjoint.

In either form in Figure 3.5, each subgraph G_i will be connected and all its cycles are edge-wise disjoint. Thus it itself must be one of the two forms shown in Figure 3.5 when we look at it with respect to α_i or γ_i . We should note that in the second form, G is a tree when none of the subgraphs contain any cycles.

We shall basically adopt both the divide-and-conquer approach and the dynamic programming approach. A given graph G will be divided into subgraphs G_i which may be further divided into smaller subgraphs. The calculations of the number of linear extensions related to a graph G shall be done using a table of such numbers computed for its subgraphs G_i . Each form of subgraphs in Figure 3.5 will require a different sequence of calculations. The two sequences are given in the proofs of the following two lemmas.

LEMMA 3.1. Suppose that the Hasse diagram for a given poset P is of the form shown in Figure 3.5(a). Let Q_i be the poset corresponding to G_i . Then

the numbers $Z(P; \text{rank}(\alpha_1) = r)$, $1 \leq r \leq n$, can be calculated from the numbers $Z(Q_i; \text{rank}(\alpha_i) = c)$, $1 \leq i \leq k$, $1 \leq c \leq |Q_i|$, in $O(n^7)$ steps.

(a) G viewed as a 'cycle' of subgraphs.

(b) G viewed as a 'tree' of subgraphs.

Figure 3.5. Two forms of the underlying undirected graph G .

Proof. As in the proof of Theorem 3.1, we apply Corollary 2.1 for $k-1$ iterations. In iteration i , $1 \leq i < k$, we let P_1 be the poset formed by the vertices of $G_1 \cup G_2 \cup \dots \cup G_i$, P_2 be the poset formed by the vertices of G_{i+1} , $\alpha = \alpha_1$, $\beta = \alpha_i$, and $\gamma = \alpha_{i+1}$. Note that α is again fixed to be α_1 in all the iterations.

In each iteration, letting Q be the poset formed by the vertices of $P_1 \cup P_2$, we calculate the numbers $Z(Q; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t)$, $r, t = 1, 2, \dots, |Q|$, from the numbers Z^1_{ab} and Z^2_c as shown in Corollary 2.1 and Lemma 2.2. In the i^{th} iteration, we have $Z^2_c = Z(Q_{i+1}; \text{rank}(\alpha_{i+1}) = c)$. In the first iteration, $\alpha = \beta = \alpha_1$ and thus Z^1_{ab} are the numbers $Z(Q_1; \text{rank}(\alpha_1) = a)$. In iteration $i \geq 2$, we note that γ of iteration $i-1$ becomes β of iteration i and so Z^1_{ab} are those numbers $Z(Q; \text{rank}(\alpha_1) = a, \text{rank}(\alpha_i) = b)$ calculated in the previous iteration.

In the first iteration, the calculations in Corollary 2.1 and Lemma 2.2 take $O(n^4)$ steps since $\alpha = \beta$. In all the other iterations, the calculations take $O(n^6)$ steps. With $k \leq n$, the total time complexity for all the iterations is thus $O(n^7)$. After the last iteration, the following summation is performed for each value of r to obtain the required set of numbers:

$$\begin{aligned} Z(P; \text{rank}(\alpha_1) = r) &= \sum_{r < t} Z(R; \text{rank}(\alpha_1) = r, \text{rank}(\alpha_k) = t) && \text{if } \alpha_1 < \alpha_k, \text{ and} \\ &= \sum_{r > t} Z(R; \text{rank}(\alpha_1) = r, \text{rank}(\alpha_k) = t) && \text{if } \alpha_k < \alpha_1 \end{aligned}$$

where R is the poset Q obtained in the last iteration. These summations take into account the edge between α_1 and α_k and they require only $O(n^2)$ steps.

Thus the calculation of the required numbers takes a total of $O(n^7)$ steps if the numbers $Z(Q_i; \text{rank}(\alpha_i) = c)$ are available.

LEMMA 3.2. Suppose that the Hasse diagram for a given poset P is of the form shown in Figure 3.5(b). Let Q_i be the poset corresponding to G_i . Then the numbers $Z(P; \text{rank}(\alpha) = r)$, $1 \leq r \leq n$, can be calculated from the numbers $Z(Q_i; \text{rank}(\gamma_i) = c)$, $1 \leq i \leq k$, $1 \leq c \leq |Q_i|$, in $O(n^5)$ steps.

Proof. We apply Corollary 2.1 for k iterations. In iteration i , we let P_1 be the poset formed by the vertices of $\{\alpha\} \cup G_1 \cup G_2 \cup \dots \cup G_{i-1}$ and P_2 be the poset formed by the vertices of G_i , and we let $\alpha = \beta$ and $\gamma = \gamma_i$.

As before, we calculate, for each iteration, the numbers $Z(Q; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t)$, where Q is the poset formed by the vertices of $P_1 \cup P_2$. They are calculated from the numbers Z^1_{ab} and Z^2_c as shown in Corollary 2.1 and Lemma 2.2. In the i^{th} iteration, $Z^2_c = Z(Q_i; \text{rank}(\gamma_i) = c)$. In the first iteration, $Z^1_{ab} = 1$ if $a = b = 1$ and 0 otherwise. In iteration $i \geq 2$, the numbers $Z(Q; \text{rank}(\alpha) = a, \text{rank}(\gamma_{i-1}) = t)$ calculated in the previous iteration are summed over all t to obtain the Z^1_{ab} numbers (note that $a = b$).

In every iteration, the calculations will take $O(n^4)$ steps since $\alpha = \beta$. With $k \leq n$, the total time complexity for all the iterations is $O(n^5)$. After the last iteration, the following summation is performed for each value of r , in $O(n)$ steps, to obtain the required set of numbers:

$$\sum_{t=1, n} Z(P; \text{rank}(\alpha) = r, \text{rank}(\gamma_k) = t).$$

The total time taken to calculate all the required numbers (from the numbers $Z(Q_i; \text{rank}(\gamma_i) = c)$) is thus $O(n^5)$.

We shall now state the algorithm in full. We begin with a graph G that is the Hasse diagram of a poset $P = P_G$ and we want to find $Z(P_G)$. G is to be

connected and all its cycles must be edge-wise disjoint. Let v be any node in G . Algorithm 3.1 calculates $Z(P_G; \text{rank}(v) = r)$ for $r = 1, 2, \dots, |P_G|$. Then we obtain $Z(P_G)$ as the sum over all these numbers.

The procedure proceeds as follows. If the given graph G is a tree, then we just use Algorithm 2.3 (with $\alpha = v$) to obtain the required numbers. Otherwise, there will be one or more cycles and the calculations will depend on whether the selected node v belongs to a cycle in G . If v belongs to a cycle, we apply Lemma 3.1 with $\alpha_1 = v$. If it doesn't belong to a cycle, we apply Lemma 3.2 with $\alpha = v$. In either of these two applications, each subgraph G_i in the lemmas will also be a connected graph and all its cycles will be edge-wise disjoint. Hence the entire process has to be applied recursively to them to calculate the set of numbers $S_i = \{Z(P_{G_i}; \text{rank}(\phi) = f)\}$, where P_{G_i} is the subposet of P represented by G_i , ϕ is α_i or γ_i and $1 \leq f \leq |P_{G_i}|$. These numbers are required in the computation of $Z(P_G; \text{rank}(v) = r)$.

procedure Cycles (G, v, S)

(* G is the underlying undirected graph of the Hasse diagram of a poset P_G and $v \in G$. This procedure calculates the set $S = \{Z(P_G; \text{rank}(v) = r) \mid r = 1, 2, \dots, |P_G|\}$. P_G will be a subposet of a 'global' poset P . Condition: G is connected and all its cycles are edge-wise disjoint. *)

If G is a tree **then**

 Apply Algorithm 2.3 to G to calculate S with $\alpha = v$

else

If v belongs to a cycle in G **then**

```

    Apply Lemma 3.1 to G to calculate S with  $\alpha_1 = v$ ;
    for each subgraph  $G_i$  do
        call Cycles ( $G_i, \alpha_i, S_i$ )
    end for
else
    Apply Lemma 3.2 to G to calculate S with  $\alpha = v$ ;
    for each subgraph  $G_i$  do
        call Cycles ( $G_i, \gamma_i, S_i$ )
    end for
end if
end if
end procedure

```

Algorithm 3.1. Computing $Z(P; \text{rank}(v) = r)$.

Complexity Analysis

To obtain the overall time complexity of the algorithm, we shall calculate an upper bound on the number of times the algorithms described in the proofs of Lemmas 3.1 and 3.2, and Algorithm 2.3 are used for a given graph G with n nodes. Lemma 3.1 is applied only once for each cycle. For each application, there is one less cycle in the subgraphs. Since the number of cycles cannot be more than n , this lemma is applied at most n times, with $O(n^2)$ steps in each application. Lemma 3.2 can also be applied at most n times (with $O(n^5)$ steps each time) because each application reduces the number of nodes in the subgraphs by one. Algorithm 2.3 can be applied to a graph G or a subgraph G_i at most once. Thus no node can be inspected by this algorithm in two different applications (especially since the subgraphs are disjoint). Hence it cannot be applied more than n times. The total time complexity is therefore

$$O(n^8 + n^6 + n^6) = O(n^8).$$

If Algorithm 2.3 is applied in the procedure, $O(n^3)$ space is required to calculate the required set of numbers, as discussed in the previous chapter. If Lemma 3.1 is applied, the amount of space required to store the three sets of numbers $(Z^1_{ab}, Z^2_c, Z(Q; \text{rank}(\alpha) = r, \text{rank}(\gamma) = t))$ in each iteration is $O(n^2 + n + n^2) = O(n^2)$. Assuming the same locations are used for these numbers in each iteration, the amount of space required in all the iterations is $O(n^2)$. Finally, if Lemma 3.2 is applied, the amount of space required to store the three sets of numbers in each iteration is $O(n + n + n^2) = O(n^2)$. Again, assuming the same locations are used for these numbers in each iteration, the amount of space required in all the iterations is $O(n^2)$. Since Algorithm 2.3 and Lemmas 3.1 and 3.2 can each be applied at most n times, the total amount of space used in the entire calculation of $Z(P)$ is $O(n^4 + n^3 + n^3) = O(n^4)$.

3.3. Initial Conditions

The last issue to be dealt with in this chapter is the two conditions on the underlying undirected graph of the given Hasse diagram. The conditions are that the graph be connected, and that its cycles be edge-wise disjoint.

The question of connectivity can be easily answered by finding the number of components. The graph is connected if this number is one. The number of components in a graph can be found by using Algorithm 3.2. It inspects each node in turn and initiates a depth-first search (see [AHU83]) from a node that has not been visited in depth-first searches initiated from previous nodes. Each depth-first search will visit all the nodes in a component which implies that the number of components is the number of such searches.

Assuming an adjacency list representation for the graph, each depth-first search will take $O(c)$ steps where c is the number of edges in the component. If the graph has n nodes and e edges then the verification of this first condition takes $O(n + e) = O(n^2)$ steps.

The question of whether the cycles are edge-wise disjoint requires more work. It is easier to obtain a list of the cycles after we know that they are edge-wise disjoint than it is to obtain the list before we use it to check if the cycles are edge-wise disjoint. What we shall do first is to remove all the vertices and edges that do not belong to any cycles. This requires the following definition of a cut-edge. A *cut-edge* is an edge which, when removed from a connected graph, disconnects the graph. Now, an edge in a connected graph is a cut-edge if and only if it belongs to no cycle in the graph. Thus removing all the cut-edges and any resulting isolated vertices leaves behind only the cycles. The removal of the cut-edges requires the ability to identify them. This can be performed by observing that the deletion of an edge from a graph increases the number of components by one if and only if the edge is a cut-edge. We shall delete each edge in turn and calculate the number of components to determine if it should really be removed. Again, the number of components can be calculated by using Algorithm 3.2. The removal of all cut-edges will therefore take $O(en^2) = O(n^4)$ steps.

If the cycles are both edge-wise and vertex-wise disjoint, then every vertex in the new graph will have valency 2. Otherwise, some vertices will have valency greater than 2. To check for edge-wise disjointness, we just have to look at every vertex v that has valency δ greater than 2. If the removal of any such v increases the number of components in the graph by less than $\lceil \delta/2 \rceil$ -

1, we conclude that the cycles are not edge-wise disjoint. With at most n vertices, this test can be done in $O(n^3)$ time by making use of Algorithm 3.2.

The logic behind the test is as follows. If there are vertices of valency greater than 2, there are some cycles sharing these vertices which, possibly, also share edges. Let us first discuss the case where the cycles are edge-wise disjoint but not vertex-disjoint such as those shown in Figure 3.6(a). Here, the removal of any such vertex v (and the incident edges) will separate the cycles that share it. The number of cycles sharing v will be $\lceil \delta/2 \rceil$ since each cycle will have two edges incident on v . Hence, the removal of v will increase the number of components by $\lceil \delta/2 \rceil - 1$. (If the valency is odd, the cycles obviously cannot be edge-wise disjoint.)

Let us now consider the case where some cycles are not edge-wise disjoint such as those shown in Figure 3.6(b). For some vertices with valency greater than 2, the increase in the number of components will be less than $\lceil \delta/2 \rceil - 1$ when any one of them is removed. The reason is as follows. Let C_1 and C_2 be two cycles sharing a set of vertices, V_s , and a set of edges, E_s . Let L be a maximal chain of consecutive edges in E_s . Also, let x and y be the two end vertices of L . For each of x and y , there must be two incident edges that are not in L . One of the them belongs to C_1 and the other to C_2 . Thus we have at least two vertices with valency greater than 2 in V_s . The removal of x or y will not separate C_1 and C_2 . All the other vertices and edges of each cycle will still be connected to each other. In addition, all these remaining vertices and edges of the two cycles will still be in a single connected component since C_1 and C_2 share more than one vertex. Without the separation of these two cycles, the increase in the number of components

will be less than the maximum of $\lceil \delta/2 \rceil - 1$.

Figure 3.6. Cycles that are not disjoint.

Lastly, we note that the verification of the two conditions can be done in polynomial time. The condition that the graph be connected can be verified in $O(n^2)$ time by calculating the number of components. The condition that the cycles be edge-wise disjoint can be verified in $O(n^4 + n^3) = O(n^4)$ time. Hence the total time taken is $O(n^4)$.

(* We want to find the number of components, nc , in an undirected graph G . Depth-first searches on the components of G are employed to do this. The array 'visited' keeps track of nodes that have not been visited by any searches that have already been performed. An adjacency list representation is assumed for G . *)

```
for each node  $v$  in  $G$  do
    visited [ $v$ ]  $\leftarrow$  false
```

```
end for
```

```
 $nc \leftarrow 0$ 
```

```
for each node  $v$  in  $G$  do
    If visited [ $v$ ] = false then
        call DFS ( $v$ )
         $nc \leftarrow nc + 1$ 
```

```
    end if
```

```
end for
```

```
procedure DFS ( $v$ )
```

```
    (* This procedure performs a depth-first search on  $G$ , starting from
       the node  $v$ . All the nodes in the component containing  $v$  will be
       visited. *)
```

```
    visited [ $v$ ]  $\leftarrow$  true
```

```
    for each vertex  $w$  on the adjacency list of  $v$  do
```

```
        If visited [ $w$ ] = false then
```

```
            call DFS ( $w$ )
```

```
        end if
```

```
    end for
```

```
end procedure
```

Algorithm 3.2. Computing the number of components in a graph G .

Chapter 4

POSETS OF BOUNDED WIDTH

This chapter treats another class of partial orders and presents a polynomial time algorithm to compute $Z(P)$ for posets P of bounded width k . The *width* of a poset is the size of the largest set S of pairwise incomparable elements. By pairwise incomparable, we mean that there is no order relation between every pair of elements in S . A set of pairwise incomparable elements is often referred to as an *antichain*. So the width is just the size of the maximum-sized antichain. The algorithm is based on Dilworth's theorem which states that a poset of width k can be represented as a union of k chains.

Given a decomposition of the Hasse diagram for P into k chains, the algorithm keeps certain auxiliary tables and processes the nodes and edges one by one using a dynamic programming approach. The time and space complexity for the algorithm are $O(n^{k+k})$ and $O(n^{k+k-k})$ respectively.

4.1, Posets of Width 2

In this section we develop an algorithm for calculating the number of linear extensions, in polynomial time, for a poset of width 2. This case is singled out for three reasons:

- (i) the rather tricky details are easier to follow in this case,
- (ii) when $k = 2$ the problem is that of computing the number of mergings of two sorted lists subject to constraints on the merge; we have studied the case of a single constraint in Section 2.4,

(iii) a different algorithm for $k = 2$ has been given recently by Linial [L84]; our method appears to be easier to use.

It should also be mentioned that Linial proved that for any poset of width 2, there always exists a pair of elements x, y such that the proportion of linear extensions with $x < y$ lies between $1/3$ and $2/3$ (inclusive). He also proved that the number of comparisons required to sort this poset is no more than $C \cdot \log Z(P)$ where $C = (\log((\sqrt{5} + 1)/2))^{-1}$.

Suppose we are given a poset P of width 2. The Hasse diagram for P is a union of 2 disjoint nonempty chains which we shall refer to as C_1 and C_2 ; in addition there may be edges going between C_1 and C_2 . Our algorithm builds the Hasse diagram for P in a succession of steps of two types. In a step of type 1 ('adding a node to a chain') we add one node to the top of its chain together with the edge connecting it to the node below it in the chain. In a step of type 2 ('adding a crossing edge') we add an edge from the top node of a chain down to any node in the other chain. As this construction proceeds, we compute certain information about the linear extensions of the partial order defined by the current diagram.

Every element of P has a certain rank in its chain. Thus the minimal element of C_1 (or C_2) has rank 1 and the maximal element of C_1 has rank $|C_1|$. Let α, β be the maximal elements of C_1, C_2 . In a linear extension of P , α may lie between two adjacent elements in the chain C_2 ; we say that α has rank r in C_2 if these elements have ranks $r-1$ and r . However if, in the linear extension, α lies above the maximal element of C_2 or below the minimal element of C_2 we say that α has rank $|C_2| + 1$ in C_2 or α has rank 1 in C_2 respectively. Similarly, in a linear extension of P , we define the rank of β in

C_1 .

We define $Z_{rs}(P)$ to be the number of linear extensions of P such that α has rank r in C_2 and β has rank s in C_1 . For the sake of simplicity, we shall assume that r and s can each range from 1 to $|P| = n$. In this case, we can straightaway set $Z_{rs}(P)$ to be 0 if $r > |C_2| + 1$ or if $s > |C_1| + 1$ because these ranks are not possible for α and β . It may also be noted that many of the $Z_{rs}(P)$ are zero since any linear extension of P must have one of α and β as its maximum element; in particular, $Z_{rs}(P) \neq 0$ can occur only if $r = |C_2| + 1$ or $s = |C_1| + 1$. Our algorithm calculates $Z(P)$ as a sum over all the $Z_{rs}(P)$ values, i.e.

$$Z(P) = \sum_{\substack{r=1,n \\ s=1,n}} Z_{rs}(P).$$

The calculation of the values $Z_{rs}(P)$ is done by calculating in turn the values $Z_{rs}(Q)$ for each of the partial orders Q which arise in the step-wise construction of P . At each intermediate stage, we shall have a poset Q that is a subset of P ; we shall have calculated $Z_{rs}(Q)$ for all values of r and s . At the next step, we shall add a node or crossing edge to Q to get a poset R . Each value $Z_{rs}(R)$ may then be calculated by using the values of $Z_{rs}(Q)$ that have already been computed. Using this approach, we now state the main technical result on which our algorithm depends.

LEMMA 4.1. Let R be a poset of width 2 and let α and β be respectively the top elements of the two chains C_1 and C_2 associated with R . Suppose that R has been obtained from a poset Q by adding a node (one of α and β) or by adding a crossing edge (with top end one of α and β). Then the

numbers $Z_{rs}(R)$ can be obtained from the numbers $Z_{rs}(Q)$ by the following computations.

(1) Obtain R by adding α to Q :

```

if  $r = |C_2| + 1$  and  $s \leq |C_1|$  then
     $Z_{rs}(R) \leftarrow \sum_{t=1, r} Z_{ts}(Q)$ 
else if  $r \leq |C_2|$  and  $s = |C_1| + 1$  then
     $Z_{rs}(R) \leftarrow \sum_{t=1, r} Z_{t, s-1}(Q)$ 
else
     $Z_{rs}(R) \leftarrow 0$ 
end if

```

(2) Obtain R by adding β to Q :

```

if  $r \leq |C_2|$  and  $s = |C_1| + 1$  then
     $Z_{rs}(R) \leftarrow \sum_{t=1, s} Z_{rt}(Q)$ 
else if  $r = |C_2| + 1$  and  $s \leq |C_1|$  then
     $Z_{rs}(R) \leftarrow \sum_{t=1, s} Z_{r-1, t}(Q)$ 
else
     $Z_{rs}(R) \leftarrow 0$ 
end if

```

(3) Obtain R from Q by adding a crossing edge from α to an element γ in C_2 where $\gamma \leq \beta$:

```

let  $b$  be the rank of  $\gamma$  in  $C_2$ 
if  $r > b$  then
     $Z_{rs}(R) \leftarrow Z_{rs}(Q)$ 
else
     $Z_{rs}(R) \leftarrow 0$ 
end if

```

- (4) Obtain R from Q by adding a crossing edge from β to an element γ in C_1 where $\gamma \leq \alpha$:

let a be the rank of γ in C_1

if $s > a$ then

$$Z_{rs}(R) \leftarrow Z_{rs}(Q)$$

else

$$Z_{rs}(R) \leftarrow 0$$

end if

Proof. We shall give the proof only for cases (1) and (3) because cases (2) and (4) are symmetrical to cases (1) and (3) respectively.

Case 1. Let δ be the maximum element of $C_1 - \{\alpha\}$; this chain is the analogue in Q of the chain C_1 . In any linear extension L of R , $\alpha <_L \beta$ or $\beta <_L \alpha$. Thus, in a linear extension, either α has rank $|C_2| + 1$ in C_2 or β has rank $|C_1| + 1$ in C_1 . In particular, if $r \neq |C_2| + 1$ and $s \neq |C_1| + 1$, $Z_{rs}(R) = 0$.

Consider those linear extensions L of R in which α has rank $r = |C_2| + 1$ in C_2 and β has rank s in C_1 . In such an extension $\beta <_L \alpha$ and hence $s \leq |C_1|$. Consequently, these extensions arise by adding α as the maximum element to those extensions of Q where δ has some rank t , $1 \leq t \leq |C_2| + 1$, and β has rank s in $C_1 - \{\alpha\}$. Hence there are $\sum_{t=1, r} Z_{ts}(Q)$ such extensions.

Next, consider those extensions L of R in which α has rank $r \leq |C_2|$ in C_2 . In these extensions, $\alpha <_L \beta$ and so β has rank $s = |C_1| + 1$ in C_1 and rank $s - 1 = |C_1|$ in $C_1 - \{\alpha\}$. These extensions arise from those of Q by inserting α in between the elements of ranks $r - 1$ and r of C_2 and after the

maximal element δ of $C_1 - \{\alpha\}$; such an insertion point for α exists only if δ has rank t in C_2 and it is then unique. Hence there are $\sum_{t=1, r} Z_{t, s-1}(Q)$ such extensions.

Case 3. The poset R has been obtained from Q by imposing the extra relation $\gamma < \alpha$ (and its transitive consequences). A linear extension of Q satisfies this extra relation if and only if the rank of α in C_2 exceeds b , the rank of γ . Thus

$$\begin{aligned} Z_{rs}(R) &= 0 \text{ if } r \leq b, \text{ and} \\ Z_{rs}(R) &= Z_{rs}(Q) \text{ if } r > b. \end{aligned}$$

To ensure that a sequence of construction steps builds the full set of nodes and order relations of a poset Q , it must have the following properties:

1. Each node γ is added to its chain directly above node δ , in a step of type 1, only after any crossing edge (δ, ϵ) with $\epsilon < \delta$ has been added.
2. Each crossing edge (γ, δ) with $\delta < \gamma$ is added, in a step of type 2, only after the nodes γ and δ have been added.
3. Each node is added only after all its predecessors in the poset have been added.

These properties define a binary relation 'after' on the nodes and Hasse diagram crossing edges of the poset Q . The closure of this relation can be easily seen to be a partial order Q^* ; indeed its Hasse diagram is obtained from that of Q by adding new nodes, as shown in Figure 4.1, for each crossing

edge. Note that $x <_Q y$ means that y is inserted after x .

Figure 4.1. Poset Q^* derived from poset Q .

LEMMA 4.2. Let the relations defined on the sets Q, Q^* be $<_Q, <_{Q^*}$. Then Q^* and $<_{Q^*}$ together constitute a poset.

Proof. According to the above properties, we have $x <_{Q^*} y$ if and only if one of the following statements hold:

- (a) $x \in Q, y \in Q, x <_Q y$.
- (b) $x = (x_1, x_2) \in <_Q, y \in Q, x_2 <_Q y$.
- (c) $x \in Q, y = (y_1, y_2) \in <_Q, x = y_2$ or $x <_Q y_2$.
- (d) $x = (x_1, x_2) \in <_Q, y = (y_1, y_2) \in <_Q, x_2 <_Q y_2$.

Note that $(x, y) \in <_Q$ is the same as $x <_Q y$. To show that $<_{Q^*}$ is antireflexive,

i.e. to show that we do not have $x <_Q x$, we consider the following cases:

- Case 1.* If $x \in Q$ then we cannot have $x <_Q x$ because it would imply that $x <_Q x$ which violates the antireflexive property of $<_Q$.
- Case 2.* If $x = (x_1, x_2) \in <_Q$ then we again cannot have $x <_Q x$ because it would imply that $x_2 <_Q x_2$ which also violates the antireflexive property of $<_Q$.

For the antisymmetric property, we want to show that if $x <_Q y$ then we do not have $y <_Q x$. We do this by considering the different cases for x and y :

- Case 1.* If $x \in Q$ and $y \in Q$, then $x <_Q y$. Thus we cannot have $y <_Q x$ because it would imply that $y <_Q x$ which violates the antisymmetric property of $<_Q$.
- Case 2.* If $x = (x_1, x_2) \in <_Q$ and $y \in Q$, then $x_2 <_Q y$. Again we cannot have $y <_Q x$ because it would imply that $y = x_2$ (violating the antireflexive property of $<_Q$) or $y <_Q x_2$ (violating the antisymmetric property of $<_Q$).
- Case 3.* If $x \in Q$ and $y \in <_Q$, then we have the reverse argument of *Case 2*.
- Case 4.* If $x = (x_1, x_2) \in <_Q$ and $y = (y_1, y_2) \in <_Q$, then $x_2 <_Q y_2$. Here $y <_Q x$ would imply that $y_2 <_Q x_2$, violating the antisymmetric property of $<_Q$ once again.

For the transitive property, we want to show that if $x <_Q y$ and $y <_Q z$, then $x <_Q z$. We consider the different cases for x , y and z :

- Case 1.* If $x \in Q$, $y \in Q$, $z \in Q$ then $x <_Q y$ and $y <_Q z$. This implies that $x <_Q z$ which in turn implies that $x <_Q z$.
- Case 2.* If $x \in Q$, $y \in Q$, $z = (z_1, z_2) \in <_Q$ then $x <_Q y$, and $y = z_2$ or $y <_Q z_2$. This implies that $x <_Q z_2$ which in turn implies that $x <_Q z$.
- Case 3.* If $x \in Q$, $y = (y_1, y_2) \in <_Q$, $z \in Q$ then $x = y_2$ or $x <_Q y_2$, and $y_2 <_Q z$. This implies that $x <_Q z$ which in turn implies that $x <_Q z$.

- Case 4. If $x = (x_1, x_2) \in <_Q$, $y \in Q$, $z \in Q$ then $x_2 <_Q y$ and $y <_Q z$. This implies that $x_2 <_Q z$ which in turn implies that $x <_{Q^*} z$.
- Case 5. If $x \in Q$, $y = (y_1, y_2) \in <_Q$, $z = (z_1, z_2) \in <_Q$ then $x = y_2$ or $x <_Q y_2$, and $y_2 <_Q z_2$. This implies that $x <_Q z_2$ which in turn implies that $x <_{Q^*} z$.
- Case 6. If $x = (x_1, x_2) \in <_Q$, $y \in Q$, $z = (z_1, z_2) \in <_Q$ then $x_2 <_Q y$, and $y = z_2$ or $y <_Q z_2$. This implies that $x_2 <_Q z_2$ which in turn implies that $x <_{Q^*} z$.
- Case 7. If $x = (x_1, x_2) \in <_Q$, $y = (y_1, y_2) \in <_Q$, $z \in Q$ then $x_2 <_Q y_2$ and $y_2 <_Q z$. This implies that $x_2 <_Q z$ which in turn implies that $x <_{Q^*} z$.
- Case 8. If $x = (x_1, x_2) \in <_Q$, $y = (y_1, y_2) \in <_Q$, $z = (z_1, z_2) \in <_Q$ then $x_2 <_Q y_2$ and $y_2 <_Q z_2$. This implies that $x_2 <_Q z_2$ which in turn implies that $x <_{Q^*} z$.

LEMMA 4.3. Given any poset Q of width 2, there exists a sequence of construction steps of types 1 and 2 which yields the poset Q , and this sequence can be found in time $O(n^2)$.

Proof. By Lemma 4.2, we can obtain a poset $(Q^*, <_{Q^*})$ which dictates a (partial) order in which the nodes and crossing edges are to be added. Any linear extension of $(Q^*, <_{Q^*})$ provides a suitable sequencing of the construction steps, and, with a simple algorithm, a linear extension can be found in time $O(n^2)$. The algorithm can be of the following form:

Find a minimal element in Q^* .

Append it to whatever is already in the linear extension.

Remove this minimal element from Q^* .

Repeat the above procedure until there are no more elements in Q^* .

In each repetition of the procedure, finding the minimal element requires the

most time. This can be done by tracing the predecessors of a node until none is found. The last predecessor in the trace is the minimal element. This tracing requires $O(n)$ time. Note that Q^* will have at most n elements more than Q because there is at most one crossing edge leading from each node of Q .

The algorithm for computing $Z(P)$ is now evident. We find, using Lemma 4.3, an appropriate sequence of construction steps, perform each step using Lemma 4.1, and finally form the sum of all numbers $Z_{rs}(P)$.

There is one last detail to cover and that concerns the initialization of our algorithm. The construction of P will begin with one node in each chain. These two nodes are the bottommost nodes of the two chains. Let the poset having just these two nodes be P^* . Since there are only two nodes, r and s can each range from 1 to 2. It can be easily verified that

$$\begin{aligned} Z_{rs}(P^*) &= 1 && \text{if } r=1, s=2, \text{ or } r=2, s=1, \\ &= 0 && \text{otherwise.} \end{aligned}$$

Example 4.1. Consider finding the number of linear extensions of the poset P in Figure 4.2. Let the sequence of construction steps be that as shown in Figure 4.3 where the poset obtained after the i^{th} construction step is P_i . Note that there may be more than one sequence for any poset. For example, the second construction step could have been adding a node to the second chain instead of the first chain or the fifth construction step could have been adding x_6 to the second chain instead of adding the crossing edge. Note however that node x_7 cannot be added until crossing edge a has been added. And this crossing edge, of course, cannot be added unless x_1 and x_6 have already

been added. The crossing edge b also cannot be added unless x_3 and x_5 have been added. A table of $Z_{rs}(P_i)$ numbers is shown for each construction step. The calculations for the second to the eighth construction steps follow from Lemma 4.1.

Figure 4.2. Poset P of width 2.

Complexity Analysis

The major amount of time used is that required to perform the calculations of Lemma 4.1 during the construction of P . At each construction step, either we add a new element or we add a crossing edge. In the former case, we calculate the numbers $Z_{rs}(R)$ where R is the new poset at that iteration. With r and s each ranging from 1 to n , there are n^2 of these numbers. The calculation for each of them requires at most $O(n)$ iterations to sum up values of $Z_{rs}(Q)$ where Q is the poset for which we add the new element to obtain R . There are altogether n elements to be added in the construction of P . Thus constructions of this type take a total time of $O(n^4)$. In the case of adding a

$$Z(P_8) = 6 + 9 + 3 + 9 = 27.$$

Figure 4.3. The Z_{rs} numbers calculated in the construction of P .

crossing edge, it is required that we set each of the n^2 numbers $Z_{rs}(R)$ to be either $Z_{rs}(Q)$ or zero by making a simple comparison between two ranks. There are no more than n crossing edges because there is at most 1 crossing edge from each node. Thus constructions of this type take a total time of $O(n^3)$.

Note that prior to the construction of P , the sequence of construction steps can be found in $O(n^2)$ time, and after the construction, the summation of all numbers $Z_{rs}(P)$ takes only $O(n^2)$ time. So, all in all, we have $O(n^4)$ as the time complexity for our algorithm.

The major amount of space required is that needed to keep track of the $O(n^2)$ values for $Z_{rs}(R)$ and $Z_{rs}(Q)$ (values for the two most recent iterations). Other than that, we also need $O(n^2)$ space to store the Hasse diagram using any standard graph representation. The sequence of construction steps can be stored using $O(n)$ space. Hence, the space complexity is $O(n^2)$.

4.2. Posets of Width k

We shall now generalize the algorithm to compute the number of linear extensions for posets of a bounded width $k \geq 2$. Consider a decomposition of a poset P of width k into k chains. Let the top element of chain C_i be α_i for $i = 1, 2, \dots, k$. Again we shall consider the rank of each α_i in every chain C_j , $1 \leq i, j \leq k$. To keep track of all these ranks, we use a $k \times k$ matrix $A = [a_{ij}]$ where a_{ij} is the rank of α_i in C_j . We shall refer to this matrix as the rank matrix. The rank of element α_i within its own chain C_i is just the length of the chain itself. Hence $a_{ii} = |C_i|$ for $1 \leq i \leq k$. Each nondiagonal element a_{ij} of A can range from 1 to $|C_j| + 1$. For simplicity, we assume each nondiagonal element can range from 1 to n .

As before, P will be constructed by a succession of steps of types 1 and 2. At each stage, we have an intermediate poset R and we calculate numbers $Z(R, A)$ for all rank matrices A with $a_{ii} = |C_i|$ and $1 \leq a_{ij} \leq n$ for $i \neq j$. Here $Z(R, A)$ is the number of linear extensions of R in which α_i has rank a_{ij} in the

chain C_j . When P has been constructed, we can obtain $Z(P)$ by summing all n^{k-k-k} numbers $Z(P, A)$.

LEMMA 4.4. Let R be a poset of width k with chain decomposition $C_1 \cup C_2 \cup \dots \cup C_k$ which has been constructed from a poset Q by a step of type 1 (for simplicity, by adding the node α_1 to the first chain $C_1 - \{\alpha_1\}$ of Q). Then for every rank matrix A , either

- (1) $Z(R, A)$ is the sum of numbers $Z(Q, B)$ where B runs through a set of rank matrices for Q defined by

$$b_{ij} = a_{ij} \quad \text{for all } i, j \geq 2$$

$$b_{1i} \leq a_{1i} \quad \text{for all } i \geq 2$$

$$b_{i1} = a_{i1} \quad \text{for all } i \in I$$

$$\text{where } I = \{i \geq 2 \mid a_{1i} = |C_i| + 1\}$$

$$b_{j1} = a_{j1} - 1 \quad \text{for all } j \in J$$

$$\text{where } J = \{j \geq 2 \mid a_{1j} \leq |C_j|\}$$

$$b_{11} = |C_1| - 1$$

or

- (2) $Z(R, A) = 0$.

Which of these occurs can be decided in time $O(n^3)$ by testing a certain condition C .

Proof. The chain decomposition of Q is $C_1 - \{\alpha_1\} \cup C_2 \cup \dots \cup C_k$. Let $\alpha_1', \alpha_2, \dots, \alpha_k$ be the top elements of these chains. Consider the following statements:

- (i) For each $i = 2, \dots, k$
 either $a_{1i} = |C_i| + 1$ and $a_{i1} \leq |C_1|$

or $a_{1i} \leq |C_i|$ and $a_{i1} = |C_1| + 1$.

- (ii) The transitive closure T of the binary relation on the set $\{\alpha_1\} \cup C_2 \cup C_3 \cup \dots \cup C_k$, whose related pairs are (x, y) , $x, y \in C_2 \cup C_3 \cup \dots \cup C_k$, $x < y$, together with pairs (u_i, α_1) , (α_1, v_i) , $i = 2, \dots, k$ where $u_i, v_i \in C_i$ and, in C_i , u_i and v_i have ranks $a_{1i} - 1$ and a_{1i} (informally, α_1 is "between" u_i and v_i in the chain C_i), is a partial order.

If statements (i) and (ii) both hold, we say that condition C is true. Notice that the verification of statement (i) requires constant time whereas the verification of statement (ii) by a transitive closure test (for cycles) requires $O(n^3)$ time.

We now show that if C is true, then $Z(R, A)$ can be calculated by part (1) of the lemma. The $Z(R, A)$ linear extensions of R with rank matrix A arise from certain linear extensions of Q into which α_1 has been inserted. Let L be a linear extension of R with rank matrix A and let K be the linear extension of Q obtained by deleting α_1 from L . Let B be the rank matrix for K . We shall verify that B satisfies the properties listed in part (1) of the lemma.

The rank of α_i , $i \geq 2$, in C_j , $j \geq 2$, is the same in K as it is in L ; hence $b_{ij} = a_{ij}$. Also, in the linear extension L , the rank of α_1 in C_i , $i \geq 2$, is at least equal to the rank of α_1' ; so $b_{1i} \leq a_{1i}$. Further, the rank of α_i in C_1 , $i \in I$, is equal to the rank of α_i in $C_1 - \{\alpha_1\}$ since $a_{1i} = |C_i| + 1$ implies that $\alpha_i <_L \alpha_1$ and thus the removal of α_1 from C_1 does not affect the rank of α_i in it; so $b_{i1} = a_{i1}$. (Note that $\alpha_i <_L \alpha_1$ implies that $a_{1i} \leq |C_i|$ which is consistent with statement (i) of condition C .) Next, the rank of α_j in C_1 , $j \in J$, is one more than the rank of α_j in $C_1 - \{\alpha_1\}$ because $a_{1j} \leq |C_j|$ implies that $\alpha_1' <_L \alpha_1 <_L \alpha_j$ and $\alpha_1' <_K \alpha_j$; so $a_{j1} = b_{j1} + 1$. (Note that $\alpha_1 <_L \alpha_j$ implies that $a_{j1} = |C_1| + 1$ which is again consistent with statement (i) of condition C .) Finally, the rank of α_1' in $C_1 - \{\alpha_1\}$ is clearly $|C_1| - 1$ and therefore B has all the properties claimed.

On the other hand, we shall now prove that any linear extension K of Q with rank matrix B satisfying the properties of part (1) does give rise to a unique linear extension of R with rank matrix A by inserting α_1 .

Consider the following sets of pairs defining a relation Γ :

- (a) the set of related pairs in statement (ii) of condition C,
- (b) the pair (α_1', α_1) , and
- (c) pairs (k_1, k_2) where $k_1 <_K k_2$ in the linear extension K .

Note that the pairs that are in set (a) but not in set (c) are those of the form (u, α_1) and (α_1, v) . The transitive closure Γ^* of this relation is a superset of the set of related pairs in R since Γ contains the pairs of a linear extension of Q , together with (α_1', α_1) .

We shall verify that Γ^* is a partial order. Suppose it is not a partial order. Then there exists a cycle of related pairs. Since K is a partial order, this cycle must contain pairs $(\gamma, \alpha_1), (\alpha_1, \beta) \in \Gamma$ for some γ and β in Q .

Let the whole cycle be $(\alpha_1, \beta), (\beta, \delta), (\delta, \epsilon), \dots, (\gamma, \alpha_1)$. Since K is partial order, $(\beta, \gamma) \in \Gamma$ and we may take the cycle to be $(\alpha_1, \beta), (\beta, \gamma), (\gamma, \alpha_1) \in \Gamma$. By construction $\beta \in C_2 \cup C_3 \cup \dots \cup C_k$; say $\beta \in C_i$. If $\gamma \in C_2 \cup C_3 \cup \dots \cup C_k$, we would have a cycle in the poset T , violating statement (ii) of condition C; hence $\gamma = \alpha_1'$. In the linear extension K , let β have rank t in C_i . Since $(\beta, \alpha_1') \in \Gamma$, $\beta <_K \alpha_1'$ and thus in the linear extension K , α_1' has rank b_{1i} greater than t in C_i . Also, $(\alpha_1, \beta) \in \Gamma$ implies that β has rank a_{1i} in C_i (in accordance with statement (ii) of condition C); so $t = a_{1i}$. Hence, we have $b_{1i} > t = a_{1i}$, a contradiction.

Consequently, the transitive closure Γ^* is a partial order and, since it relates α_1 to all the elements in K , it is a total order, a linear extension of R .

This linear extension does have rank matrix A . The rank of α_i , $i \geq 2$, in C_j , $j \geq 2$, is the same in L as it is in K ; hence $a_{ij} = b_{ij}$. Also, $a_{1i} \geq b_{1i}$ because (α_1', α_i) is in Γ . By construction, α_1 has rank a_{1i} in C_i . Moreover, α_i has rank a_{i1} in C_1 for $i \in I$ and α_j has rank a_{j1} in C_1 for $j \in J$ since

(a) $a_{1i} = |C_i| + 1$ implies that $\alpha_i <_L \alpha_1$ so that the rank of α_i in C_1 = rank of α_i in $C_1 - \{\alpha_1\}$, i. e. $a_{i1} = b_{i1}$, and

(b) $a_{1j} \leq |C_j|$ implies that $\alpha_1 <_L \alpha_j$ so that the rank of α_j in C_1 = 1 + rank of α_j in $C_1 - \{\alpha_1\}$, i. e. $a_{j1} = 1 + b_{j1}$.

Lastly, with α_1 added to $C_1 - \{\alpha_1\}$, the length of this chain increases by one and thus $a_{11} = b_{11} + 1 = |C_1|$.

We now make the observation that no two linear extensions of Q with the same rank matrix B can give rise to the same linear extension of R with rank matrix A . Suppose they do. Then they differ not by the positions of the chain heads $\alpha_1', \alpha_2, \dots, \alpha_k$ because these positions are fixed by B . They must differ in the positions of the other elements but this is impossible because these elements must have the same relative ordering as in L (the insertion of α_1 does not change this ordering).

We have thus proved that, when the condition C holds, every one of the $Z(Q, B)$ extensions of Q with rank matrix B satisfying the above properties gives a unique linear extension of R with rank matrix A . So we should sum over all possible B matrices (for all values of b_{1i} that are less than or equal to a_{1i} for all $i \geq 2$) to obtain $Z(R, A)$:

$$Z(R, A) = \sum_B Z(Q, B).$$

Finally we shall prove part (2) of the lemma, i.e. we shall show that if C is false, then $Z(R, A) = 0$. Suppose that $Z(R, A) \neq 0$ when C is false, and let L

be any linear extension of R with rank matrix A .

In L , either $\alpha_1 <_L \alpha_i$ or $\alpha_i <_L \alpha_1$. Thus we have the condition that either α_1 has rank $a_{1i} = |C_1| + 1$ and α_i has rank $a_{i1} \leq |C_1|$ or α_1 has rank $a_{1i} \leq |C_1|$ and α_i has rank $a_{i1} = |C_1| + 1$. Hence statement (i) of condition C is true. Moreover the binary relation defined in statement (ii) is a subset of the order relation of L and thus its transitive closure T is a partial order. This means that statement (ii) is also true and thus we have a contradiction. We conclude that $Z(R, A)$ must be zero if C does not hold.

LEMMA 4.5. Let R be a poset of width k with chain decomposition $C_1 \cup C_2 \cup \dots \cup C_k$ which has been constructed from a poset Q by a step of type 2 (for simplicity, by adding a crossing edge from α_1 to some point γ in C_2 , with $\gamma \leq \alpha_2$). Then if b is the rank of γ in C_2 , we can determine $Z(R, A)$ for every rank matrix A by the rules

$$Z(R, A) = 0 \text{ if } a_{12} \leq b, \text{ and}$$

$$Z(R, A) = Z(Q, A) \text{ if } a_{12} > b.$$

Proof. The poset R has been obtained from Q by imposing the extra relation $\gamma < \alpha_1$ (and its transitive consequences). A linear extension of Q satisfies this extra relation if and only if the rank of α_1 in C_2 exceeds that of γ .

Using these lemmas, we can give an informal description of the algorithm for calculating $Z(P)$:

1. Choose a sequence of steps of types 1 and 2 which constructs P .

2. Calculate $Z(S, A)$ for every rank matrix A , where S is the k -element poset with no order relations.
3. Follow the sequence of steps; at each one, find $Z(R, A)$ for the current poset R and all rank matrices A .
4. Compute $Z(P) = \sum Z(P, A)$, summed over all rank matrices A .

Part 1 is achieved by a similar procedure to that given in Lemma 4.3 and part 3 is achieved by applying the constructions implied by Lemmas 4.4 and 4.5. Part 2 is achieved by considering each rank matrix A in turn, and deciding whether $Z(S, A)$ equals 0 or 1. Note that with only one element in each chain and with no order relations, each rank matrix either describes a total ordering of S or it does not. One way to verify whether it does or not is as follows:

```

 $S_k \leftarrow \{1, 2, \dots, k\}$ 
total_ordering  $\leftarrow$  true
while total_ordering and  $S_k$  is not empty do
    if there exists  $i \in S_k$  s. t.,  $a_{ij} \leq |C_j|$  and  $a_{ji} = |C_j| + 1$  for all  $j \in S_k, j \neq i$  then
         $S_k \leftarrow S_k - \{i\}$ 
    else
        total_ordering  $\leftarrow$  false
    end if
end while

```

Let x_i be the minimum element in a total ordering L of S with rank matrix A . Then since every other element x_j in L is such that $x_i < x_j$, $a_{ij} \leq |C_j|$ and $a_{ji} = |C_j| + 1$. If these conditions are satisfied by A , we can delete this element from L (equivalent to deleting row i and column i of A or deleting i from S_k) and repeat the test for the next smallest element in L . The test should be repeated until there are no more elements in L .

Complexity Analysis

Let us first analyse the space requirement. Of all parts of the algorithm, it is clear that part 3 requires the most space. At each iteration, a large amount of space is used to keep track of the $Z(R, A)$ and $Z(Q, B)$ numbers for all rank matrices A . Each nondiagonal element a_{ij} of A can range from 1 to n . Hence, with $k^2 - k$ nondiagonal elements, we will have $O(n^{k+k-k})$ rank matrices. Thus our space complexity is just $O(n^{k+k-k})$.

We shall now analyse the time requirement. As has been discussed in section 4.1, part 1 of the algorithm requires $O(n^2)$ time. For each $Z(S, A)$, part 2 requires constant time to decide if it is 0 or 1 because $\hat{S}$ is a bounded poset. As for part 3, we first note that the number of construction steps is $O(n)$ since there are n nodes and no more than kn crossing edges (at most $k-1$ crossing edges leading down from each node) and in any one of the steps, we have to calculate $O(n^{k+k-k})$ $Z(R, A)$ values. In a construction step of type 1, we refer to Lemma 4.4. For each of the $O(n^{k+k-k})$ rank matrices, the condition C can be verified in $O(n^3)$ time. If it does not hold, $Z(R, A)$ is immediately set to zero. If it holds, $Z(R, A)$ will be computed as a sum of $O(n^{k-1})$ $Z(Q, B)$ values where the different B matrices arise from the condition $b_{1i} \leq a_{1i}$ for $i \geq 2$. In a construction step of type 2, we refer to Lemma 4.5. Each of the $O(n^{k+k-k})$ $Z(R, A)$ values is either set to zero or $Z(Q, B)$ based on a simple comparison between two ranks. Thus each construction step has time complexity of $O(n^{k+k-1})$ and so part 3 as a whole has $O(n^{k+k})$ as its time complexity. Part 4 just requires a sum over $O(n^{k+k-k})$ $Z(P, A)$ values. Hence, the entire algorithm has $O(n^{k+k})$ as its time complexity.

4.3. Chain Decomposition

There are methods of finding the chain decompositions based on network flows. One such method is described in [B83]. We shall reproduce this method here. The method requires us to construct a network based on the Hasse diagram or covering graph of the poset. First, we replace each node x in the poset with two nodes x_1, x_2 and an edge (x_2, x_1) where we let $x_1 < x_2$ (so that the edge is directed from x_2 to x_1). We shall refer to these new edges as 'special' edges. If $y < x$ in P , we let $y_2 < x_1$. Similarly, if $x < z$, we let $x_2 < z_1$.

Next, we add a maximum node connected to all the maximal nodes so that it acts as the source for the network. Similarly, we add a minimum node connected to all the minimal nodes so that it acts as the sink. Now, we have a connected graph with the edges directed from the source to the sink.

We can find a minimal chain decomposition by finding a flow of minimum value subject to the condition that the flow is at least one unit along each special edge. This is the kind of flow we get if we send one unit along each chain in the minimal decomposition. The process is as given below:

1. Before deciding on any capacities for the edges, construct a flow f of value F from the source to the sink that gives each edge at least one unit of flow.
2. For each special edge (x_2, x_1) , let its capacity be one less than number of units flowing through it, i.e. $c(x_2, x_1) = f(x_2, x_1) - 1$. For each other edge e , let $c(e) = f(e)$.
3. Find a maximum flow g of value G relative to these capacities.
4. For each edge e , assign new capacities: $c'(e) = f(e) - g(e)$.
5. For each edge e with $c'(e) > 1$, replace it with $c'(e)$ edges in parallel; each with capacity 1.
6. Find a maximum flow through this new network relative to the new

capacities.

The final flow in the network will give $F - G$ disjoint paths from the source to the sink. These paths will correspond to the chains in a minimal chain decomposition.

The main step in the above process is finding a maximum flow through the network; this can be done in $O(n^3)$ time (see [SDK83]).

Chapter 5

CONNECTED HASSE DIAGRAMS

This chapter investigates the maximum value of $Z(P)$ when the Hasse diagram, regarded as an undirected graph, is connected. This number is interesting because it gives us some bounds on the ITB for sorting or producing a poset represented by such a Hasse diagram.

Let S be the set of Hasse diagrams on n nodes whose underlying undirected graphs are connected. For each element $H \in S$, let $P(H)$ be the poset represented by H . We are interested in finding a Hasse diagram $H\# \in S$ such that $Z(P(H\#)) \geq Z(P(H))$ for all $H \in S$. We shall denote $P(H\#)$ by $P\#$. The following lemma relates the number of linear extensions of two posets that have the same set of elements but with the order relations of one being the subset of the other.

LEMMA 5.1. Let P and Q be two posets having the same set of elements $\{x_1, x_2, \dots, x_n\}$. If the set of order relations B_Q in Q is a proper subset of the set of order relations B_P in P , then $Z(P) < Z(Q)$.

Proof. Any total ordering of the elements $\{x_1, x_2, \dots, x_n\}$ that are consistent with all the relations in B_P must of course be consistent with all the relations in B_Q . Therefore every linear extension of P is a linear extension of Q . However there are some linear extensions of Q that will not satisfy the extra order relations in B_P . Hence the result. Note that order relations that can be deduced

by transitivity of existing relations in B_P must be present in B_P . Similarly for B_Q . Thus the extra relations in B_P that are not in B_Q do actually provide more order information.

LEMMA 5.2. Let $S_T = \{H \in S \mid \text{The underlying undirected graph for } H \text{ is a tree}\}$ and let $S_C = S - S_T$. Every Hasse diagram $H_C \in S_C$ contains a Hasse diagram $H_T \in S_T$. Furthermore, $Z(P(H_T)) > Z(P(H_C))$.

Proof. Every Hasse diagram H_C that is a connected graph but is not a tree contains cycles. This means that we can remove some edges from H_C without the graph losing its connectivity. Edges can be removed as long as the resulting graph is connected. Finally, we will reach the stage whereby the graph is a tree and we cannot remove any more edges without disconnecting it.

This resulting graph shall represent a Hasse diagram H_T in S_T . The order relations in $P(H_T)$ will be a proper subset of those in $P(H_C)$ and hence, by

Lemma 5.1, $Z(P(H_T)) > Z(P(H_C))$.

Lemma 5.2 tells us that we can remove some edges from a given Hasse diagram H_C whose underlying undirected graph is connected and contain cycles so as to obtain a Hasse diagram H_T whose underlying undirected graph is a tree. The removal of the edges, in effect, removes some of the order relations present in H_C and thus the number of linear extensions of $P(H_T)$ is greater than that of $P(H_C)$. The following lemma relates the number of linear extensions of two posets whose Hasse diagrams are trees and where one poset is obtained from the other by removing a minimal or maximal element.

The formulae can be derived from Algorithm 2.3 but we shall derive them from first principles so as to obtain a clearer understanding of the material.

LEMMA 5.3. Let P be the poset whose Hasse diagram has, as its underlying undirected graph, a tree on n nodes and let x be a leaf in the tree. Let Q be the poset obtained by removing x from P . Node x must be either a minimal or a maximal element. Let it be connected by a single edge to a node y . If x is a minimal element in P , then letting $Z_i(Q) = Z(Q; \text{rank}(y) \leq i)$,

$$Z(P) = \sum_{i=1, n-1} i \cdot Z_i(Q).$$

Otherwise

$$Z(P) = \sum_{i=1, n-1} (n - i) \cdot Z_i(Q).$$

Proof. Since x is a leaf, it must be a minimal or maximal element in the Hasse diagram. In any linear extension L_P of P , let the ranks of x and y be a and b respectively. If x is a minimal element, then it appears before y in L_P and thus $a < b$. By deleting x from L_P , we will obtain a linear extension L_Q of Q in which the rank of y will be $b-1$. There is a set S of linear extensions of P , with $\text{rank}(y) = b$, which results in this same linear extension L_Q when x is deleted from them. The elements of S differ from each other only in the position of x before y . And there is only one linear extension in S corresponding to each such position. Thus $|S| = b-1$. Inversely speaking, each linear extension of Q with y having rank i can yield i linear extensions of P by inserting x into the different possible positions before y . Hence the total number of linear extensions of P that can be obtained from all the linear extensions of Q with $\text{rank}(y) = i$ is

$$i \cdot Z_i(Q).$$

The required result is now obtained by summing over all possible ranks of y in the linear extensions of Q . We should note that each linear extension of Q will give rise to a unique set of linear extensions of P . A linear extension L_P of P that is obtained by inserting x into a linear extension of Q having $\text{rank}(y) = i$ obviously cannot be obtained in the same manner from another linear extension of Q having $\text{rank}(y) = i$ because the arrangement of the other elements would differ in these two extensions of Q and L_P can be consistent with only one of them. It also cannot be obtained in this way from another extension of Q having $\text{rank}(y) \neq i$ for the simple reason that if the rank of y in L_P is $i+1$, the rank of y in L_Q must always be i .

On the contrary, if x is a maximal element, it appears after y in L_P and we have $a > b$. In this case, the rank of y would be the same before and after the removal of x from L_P . Here, each linear extension L_Q of Q with $\text{rank}(y) = i$ can yield $n-i$ linear extensions of P because this is number of positions after y in L_Q in which x can be inserted to obtain a linear extension of P . As a consequence, the total number of linear extensions of P that can be obtained by the insertion of x into the linear extensions of Q with $\text{rank}(y) = i$ is

$$(n - i) \cdot Z_i(Q).$$

Again the required result is obtained by summing over all possible ranks of y in the linear extensions of Q .

Finally, the following theorem tells us what the Hasse diagram $H\#$ looks like.

THEOREM 5.1. $H^\#$ has one of the two forms shown in Figure 5.1 where $Z(P^\#) = (n-1)!$.

Figure 5.1. Connected Hasse diagrams which maximizes $Z(P)$.

Proof: By Lemma 5.2, we shall need to concentrate our attention only on the set S_T and ignore the set S_C . We shall prove the theorem by induction on the number of elements in the trees.

The theorem is obviously true for the case of $n = 1, 2$. For the case of $n = 3$, only the three forms of Hasse diagrams shown in Figure 5.2 are possible.

Figure 5.2. Three connected Hasse diagrams on three elements.

It is easily verified that $Z(P_1) = 1$ and $Z(P_2) = Z(P_3) = 2$, supporting our

hypothesis. For the case of $n = 4$, there are six possible forms as shown in Figure 5.3 for which we have $Z(P_1) = 1$, $Z(P_2) = Z(P_3) = 2$, $Z(P_4) = 5$, $Z(P_5) = Z(P_6) = 6 = 3!$ and thus the theorem is also true for this case.

Figure 5.3. Six connected Hasse diagrams on six elements.

Assume that the hypothesis is true for $n = k-1$, i.e. the tree has one of the two forms in Figure 5.1 (i.e. with either $k-2$ minimal elements or $k-2$ maximal elements) whereby $Z(P) = (k-2)!$. For the case of $n = k$, let x be any leaf in the tree underlying $H^\#$ and let it be linked to node y . If x is a minimal element, then by Lemma 5.3,

$$Z(P^\#) = \sum_{i=1, k-1} i \cdot Z_i(Q)$$

where Q is obtained from $P^\#$ by removing x . Obviously the following inequality is true:

$$Z(P^\#) \leq (k-1) \sum_{i=1, k-1} Z_i(Q) = (k-1) Z(Q).$$

By our induction hypothesis, $Z(Q)$ is maximized if its Hasse diagram H_Q has one of the two forms shown in Figure 5.4 for which $Z(Q) = (k-2)!$. Consequently, we have

$$Z(P^\#) \leq (k-1) (k-2)!$$

Equality will be achieved if and only if $Z_i(Q) = 0$ for $i < k-1$, i.e. if and only if y is the unique maximum of Q . Thus H_Q has the first form with y as the maximum element. It follows that $H^\#$ has the form shown in Figure 5.5 for which the number of linear extensions of $P^\#$ is $Z(P^\#) = (k-1)!$.

Figure 5.4. Candidates for maximizing $Z(Q)$.

The case when x is a maximal element can be argued symmetrically. Hence the hypothesis is true for all positive values of n .

Figure 5.5. Connected Hasse diagram which maximizes $Z(Q)$.

We can now obtain the following corollaries which gives us upper and lower bounds on the ITB for the generalized problems of sorting and producing posets that have connected Hasse diagrams.

COROLLARY 5.1. For the problem of sorting a poset P for which the Hasse diagram is connected,

$$\text{ITB} \leq \log (n-1)! \approx (n-1) \log (n-1)$$

where $|P| = n$.

Proof. As discussed in Chapter 1, the ITB for such a problem is $\log Z(P)$. From Theorem 5.1, we know that $Z(P)$ is at most $(n-1)!$

COROLLARY 5.2. For the problem of producing a poset P for which the Hasse diagram is connected,

$$\text{ITB} \geq \log n$$

where $|P| = n$.

Proof. From Chapter 1, we know that the ITB for such a problem is $\log (n! / Z(P))$ and, from Theorem 5.1, $Z(P) \leq (n-1)!$.

Unfortunately, Corollary 5.2 is not of much use because we know that such posets require $n-1$ comparisons to make the diagram connected (see the discussion in Example 1.6).

Chapter 6

CONCLUSION

Two new algorithms have been developed to calculate the number of linear extensions for two classes of partial orders:

- (1) Posets whose Hasse diagrams, when regarded as undirected graphs, are connected and whose cycles are disjoint, and
- (2) Posets of a bounded width k .

These algorithms have execution times that are polynomial in the number of elements: $O(n^8)$ for the first class and $O(n^{k+k})$ in the second. Both of them adopt the dynamic programming approach of solving a problem in terms of a table of solutions to its subproblems.

The algorithm for the first class starts off by picking any arbitrary element v of the given poset P . Processing depends on whether v belongs to a cycle in the undirected graph G that underlies the Hasse diagram for P . If v belongs to a cycle, G is viewed as a collection of disjoint subgraphs linked together by the cycle. On the other hand, if v does not belong to a cycle, G is viewed as a collection of disjoint subgraphs linked to v by single edges. A set of numbers is calculated in either case; each of them is the number of linear extensions of P with v having a certain rank or position in the linear extensions. These numbers are summed to obtain $Z(P)$. Their calculations are different for the two views. For either view, the calculations require a similar set of numbers for the subposet corresponding to each subgraph (with respect to

the corresponding node in the cycle for the first view and with respect to the corresponding node joined to v in the second). The processing is recursively applied to each subgraph.

The algorithm for the second class utilizes Dilworth's Theorem which states that the Hasse diagram of a poset P of width k can be decomposed into k chains, where there may be edges crossing between the chains. A construction of the Hasse diagram begins with the bottom node in each chain with no edges whatsoever. In each step of the construction, either a node is added to a chain or a crossing edge is added in between chains. As each item is added, a set of numbers is calculated for the intermediate poset Q . Each element of this set is the number of linear extensions of Q with its chain heads having certain ranks or positions in the other chains. The calculations for a particular set require the set of numbers already calculated in the previous construction step. Once the construction of P is complete, the numbers in the last set are summed to give $Z(P)$.

The question of whether the number of linear extensions of any arbitrary poset can be computed in polynomial time remains open. This problem seems to be very hard. It has even been conjectured that it is #P-complete [R85], i.e. the associated counting problem of some NP-complete problem may be polynomially reduced to it. An equivalent conjecture is that the following problem is NP-complete: Given a poset P and an integer k , is $Z(P) = k$? Because of this hard open problem, we have, in this thesis, investigated special classes of posets and found polynomial algorithms to compute their number of linear extensions. It would be of interest to consider more general classes of posets and extend our results.

Apart from developing the above algorithms, a result on the number of linear extensions for posets whose Hasse diagrams are connected was also proved. For such posets, the one that maximizes the number of linear extensions either has one maximum element and $n-1$ minimal elements or it has one minimum element and $n-1$ maximal elements. The number of linear extensions for this poset is $(n-1)!$.

2 OF/DE 2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

REFERENCES

- [AHU74] A. V. Aho, J.E. Hopcroft and J. D. Ullman (1974), *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 2-25.
- [AHU83] A. V. Aho, J. E. Hopcroft and J. D. Ullman (1983), *Data Structures and Algorithms*, Addison-Wesley, Reading, Mass., 215-217.
- [An1879] D. André (1879), *Developpemens de sec x et de tang x*, C. R. Acad. Sci. Paris, Vol. 88, 965-967.
- [At77] M. D. Atkinson (1977/78), *Mathematics in the Service of Computer Programming*, Mathematical Spectrum, Vol. 10, No. 1, 6-11.
- [ASSS84] M. D. Atkinson, J.-R. Sack, N. Santoro and Th. Strothotte (July 1984), *An Efficient, Implicit Double-Ended Priority Queue*, Technical report 55, School of Computer Science, Carleton University.
- [At84a] M. D. Atkinson (Oct. 1984), *On Zigzag Permutations and Comparisons of Adjacent Elements*, Technical report 62, School of Computer Science, Carleton University.
- [At84b] M. D. Atkinson (Nov. 1984), *Partial Orders and Comparison Problems*, Technical report 68, School of Computer Science, Carleton University.
- [BJ85] S. W. Bent and J. W. John (1985), *Finding the Median Requires $2n$ Comparisons*, Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, 213-216.

-
- [B83] K. P. Bogart (1983), *Introductory Combinatorics*, Pitman Publishing Inc., Boston, Mass., 276-277.
- [F76] H. O. Foulkes (1976), *Enumeration of Permutations with Prescribed Up-Down and Inversion Sequences*, Discrete Math., Vol. 15, 235-252.
- [GM82] G. H. Gonnet and J. I. Munro (July 1982), *Heaps on Heaps*, Proceedings ICALP 9 (Aarhus), 282-291.
- [HPSS75] L. H. Harper, T. H. Payne, J. E. Savage and E. Straus (June 1975), *Sorting $X + Y$* , CACM, Vol. 18, No. 6, 347-349.
- [K73] D. E. Knuth (1973), *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, Mass., p.191.
- [KS84] J. Kahn and M. Saks (1984), *Balancing Poset Extensions*, Order, Vol. 1, 113-126.
- [L84] N. Linial (Nov. 1984), *The Information-Theoretic Bound is Good for Merging*, SIAM Journal on Computing, Vol. 13, No. 4, 795-801.
- [P72] I. Pohl (June 1972), *A Sorting Problem and Its Complexity*, CACM, Vol. 15, No. 6, 462-464.
- [R85] I. Rival, ed., (1985), *Graph and Orders*, Reidel, Dordrecht, Holland, p.524.
- [SDK83] M. M. Syslo, N. Deo and J. S. Kowalik (1983), *Discrete Optimization Algorithms*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 269-300.

END

FIN

2 04 87