AN ABSTRACT

OF THE THESIS WRITTEN BY

Phyllis May Tidd for the degree

Master of Arts in Mathematics presented

on May 5, 1987 .

Title: Perfect Hashing Functions

Abstract approved: John W Carlson

This thesis will introduce the reader to the idea of using a hashing function for storage and retrieval of information. The history of how hashing functions originated is presented. Some different types of hashing functions are discussed.

In particular this thesis presents a perfect hashing function. A perfect hashing function (phf) is an injection, F, from a set I of N objects into the set consisting of the first R non-negative integers where $R \geq N$. Perfect hashing functions are useful for the compact storage and fast retrieval of frequently used objects such as reserved words in a programming language.

PERFECT HASHING FUNCTIONS

A Thesis
Submitted To
The Division of Mathematical and Physical Sciences
Emporia State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Arts

By
Phyllis May Tidd
May 1987

James F. Lovell

Approved by the Graduate Council

John W Carlson

Approved by the Major Department

ACKNOWLEDGMENTS

I want to express my sincere appreciation to Dr. John Carlson, Prof. John Gerriets, and Dr. Larry Scott for their assistance and patience throughout the writing of this paper.

I want to thank Dr. David Bainum for his encouragement.

I also want to thank the above people for their understanding and guidance during my studies.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER I

INTRODUCTION

Storage and Retrieval of Information Leads to Hashing

After information is stored, one is then confronted with the common tasks of searching the database and retrieving the information. In general, we shall suppose that a set of N records have been stored, and the problem is to locate the appropriate one. It is assumed that each record includes a special field called a key field. Usually the N key fields are distinct, so that each key field uniquely identifies its record. The collection of all records is called a file or a table and a group of files is often called a data base.

DEFINITION 1.1 A key field is an identifier field whose value is used in computer instructions to identify a record in a file.

A search algorithm is an algorithm that accepts an argument, which may be the key field, and attempts to locate a record whose key field matches the one accepted as an argument. It may be necessary to make several comparisons before finding the desired key field of the record stored in the table or file. (Note: In this paper the terms table and file are used interchangeably.)

Many different schemes have been developed to perform the tasks of searching and retrieving. These include a

linear search on an unordered table, binary search of an ordered table, B-trees, tries, various forms of string pattern matching, and hashing. The choice of one scheme over another for a certain application generally depends on the size of the set of known keys, and the relative numbers and order of the searches, insertions, and deletions, since each scheme is efficient in some situations, and inefficient or inapplicable in others.

If a record is to be retrieved in a single probe, the location of the record must depend directly upon the value in the key field. The most efficient way to organize such a file is an array. Suppose a company has an inventory system consisting of 100 different parts. If each part is identified by a unique two-digit number from 00 to 99, the part number can be used as the key field of the record. The key fields can also be the indices of the array.

If the company had fewer than 100 parts with the part numbers ranging from 0 to 99 in its inventory file, the same structure can be used. Although there would be locations in the array corresponding to nonexistent key field values, this waste of space would be offset by the advantage of direct access to records by the key fields.

If a university wanted to keep track of its students' records by using the student's social security number as the key field, the system described above would not be practical. To use direct indexing of a nine-digit social

security number would require an array of one billion elements. This would waste an enormous amount of space.

Hashing

This paper discusses how the computer uses its arithmetic capability to store keys (and their records) and search them when desired. When the address location for storing a key along with its record is arithmetically computed, this type of address determination is referred to as key-to-address transformation or hashing.

DEFINITION 1.2 Key-to-address transformation or hashing is a transformation which maps a key into an address location for storage or retrieval of the key and its associated information.

Hashing techniques use some simple arithmetic function with the key as the variable and the function value provides the location in the table where the key and its record will be stored. The locations in the table are referred to as buckets.

The search for a key uses the same arithmetic function with the search key as the variable and the function value indicating the bucket where the key and its record can be found. If the bucket is empty, then this record of information does not exist in this file.

DEFINITION 1.3 A search key is the value of a key field which is used in the computer instruction to identify a record of a file for the process of searching a record in a file.

Suppose the maximum number of students who can be enrolled in a university is 10,000. There would be no need to reserve more than 10,000 address locations for storing records of the students. The collection of available address locations is usually referred to as the address space.

The ratio of the number of keys in the file to the maximum number of storage locations is often called the loading factor. If a university has 9,000 students, then the loading factor for the student file is 9,000/10,000. The loading factor of a file directly influences the expected number of collisions.

If a key-to-address transformation would map the records of both John Smith and Jane Doe into the same address location, say 2001, then there is a collision because only one record can be stored at the address location 2001. The collision has to be resolved by storing one of the records in a different address location and the search for that record would take a little longer.

DEFINITION 1.4 The mapping of two or more keys into the same address is called a collision.

In most cases, a hashing scheme should include a method for resolving collisions in which search operations and insertion operations may be intermingled in an arbitrary manner. If the insertion operations are guaranteed to precede all the search operations, then all the keys which

will ever be in the table are known in advance. If this is the case then one might try to find a function which maps these keys, without collisions, into a table not much larger than the set of keys.

There have been several techniques explored which resolve the collision situation. [20, 23, 25-27, 29, 32]. These techniques can be categorized into two collision resolving methods: (1) open addressing also referred to as rehashing and (2) chaining.

Linear probing and double hashing are two techniques classified as being an open addressing method. The first tends to cause a clustering of the identifiers, or keys, which the double hashing technique tries to overcome. The second method, chaining, involves keeping a linked list of all records whose keys hash into the same value.

What should be considered in choosing a hash function? It should be noted that a good hash function is one that can be easily computed and minimizes the number of collisions. In addition, one attempts to select a hash function which spreads the records uniformly throughout the table. The larger the range of the hash function, the less likely it is that two keys will have the same hash value.

If X is an identifier chosen at random from the identifier space, then we want the probability that F(X) = i to be 1/b for all buckets. Then a random X has an equal chance of hashing into any of the, b, buckets. A hash

function satisfying this property is called a uniform hash function.

There are several kinds of uniform hash functions in use. A description of a few follows:

1. Division.

The most common hash function uses the division method. In this method, the integer key, K, is divided by some number, N, and the remainder is used as the hash address for K.

$$f(K) = K \text{ Mod } N$$

This function gives values in the range of $\{0,\ldots,N-1\}$ and the hash table size is at least of size N. The goal of this method is to choose a value for N that will avoid as many collisions as possible.

2. Midsquare.

In this hash function, the key is multiplied times itself and the middle few digits of the squared value are used as the index in the hash table.

3. Folding.

In this method, the key is partitioned into several parts, all of the same length. The partitioned segments are then added or exclusive-ored to produce the hash address for the key.

CHAPTER II

PERFECT HASHING FUNCTIONS

Example 2.1

Consider the social security numbers of ten students. Suppose our file consists of student records with the following social security numbers as keys. In this section, H is the name of the hash function, $H(K_i)$ is the hash address of the i-th key in K which is the set of keys.

STUDENT	SOCIAL SECURITY NUMBER	H(K)
K_1	527 57 8430	29
K_2	515 66 8471	31
K_3	510 66 8152	21
K_4	260 06 8043	23
K_5	514 76 5714	27
K_6	510 82 6765	30
K_7	514 76 5546	30
K_8	512 80 4587	32
K_9	510 84 7848	33
K_{10}	191 50 5979	41

We can use a hash function, H, which transforms the keys, K, into a table index, H(K). One example of such a hash function is

H(K) = sum of first three digits +
sum of the last three digits.

The results of this hash function are shown above in the last column. The two keys, K_6 and K_7, produce the condition that $H(K_6) = H(K_7)$, which is a collision. Obviously, two records cannot occupy the same location. Either a method for resolving collisions must be used or a different hashing function employed.

Consider a different hash function of the division method form where N is the value 10. The following function

H(K) = K Mod 10

produces a hash table of size 10. The hash addresses of the keys, k_i are the values {0,...,i - 1} for i = 1,...,10. The function eliminates the collision on the data set and provides direct access to the storage location of the record.

A function which provides the retrieval of a record from a static table in a single "try" is called a perfect hash function. A perfect hash function is a refinement of hashing techniques.

Given a set of N keys and a hash table of size r greater than or equal to N, a perfect hash function maps the keys into the hash table with no collisions. The loading factor, LF, of a hash table is the ratio of the number of keys to the table size, N/r. A minimal perfect hash function maps N keys into N contiguous locations for a loading factor of 1. A perfect hash function with loading factor greater than or equal to 0.8 is called almost minimal.

DEFINITION 2.1. A hash function is a perfect hash function (phf) for a set of keys provided the function is one-to-one on that set of keys.

DEFINITION 2.2. A hash function is minimal perfect hash function for a set of keys provided the function

maps the keys one-to-one onto the buckets 0, 1, . . . , N - 1 where N is the number of keys in the set.

From definition 2.1, one can say that the function

H(K) = K Mod 10

for the static data set given in Example 2.1 is a perfect hashing function. And from definition 2.2, the function is also a minimal perfect hashing function.

In summary, perfect hash functions are suitable only for static sets of keys. The addition of one new key will usually require a new hash function for the entire data set. Three criteria of a good hash function are:

1. the hash address is easily calculated;
2. the loading factor of the hash table is high for a given set of keys; and,
3. the hash addresses of a given set of keys are distributed uniformly in the hash table.

A perfect hash function is optimal with respect to criterion 3. A minimal perfect hash function is optimal with respect to criterion 2.

In general, perfect hash functions are difficult to find, even if we accept an <u>almost minimal</u> solution. Knuth [1973] estimates that only one of every ten million functions is a perfect hash function for mapping the 31 most frequent English words into 41 addresses.

In pointing out how rare perfect hashing functions are, Knuth [1973] discusses the "birthday paradox". The birthday paradox states that if 23 or more people are present in a

room, the chances are good that two or more of them will have the same birthday (same month and day). If we select a random function to map 23 keys into a hash table of size 365, the probability that no two keys will map into the same location is only .4927.

Previous Work on Perfect Hash Functions

The earliest work published on perfect hashing functions appeared about twenty-five years ago. M. Greniewski and N. Turski [1963] used a perfect hash function of the form $H(K_i) = A * K_i + B$ where A and B represent constants. This function maps the operation codes of the KLIPA assembler into a nearly minimal hash table. An algorithmic method for finding such a mapping is not given by the authors. The idea of a perfect hash function led to other investigations pursued by R. Sprugnoli [1977], M. W. Du [1980], G. V. Cormack [1985], and N. Cerone [1985].

Sprugnoli [1977] presents two algorithmic methods for producing perfect hash functions. The Quotient Reduction method yields a function of the form

$$H_q(K_i) = \text{floor}\ ((K_i + A) / B).$$

The second method, called the Remainder Reduction, finds a perfect hash function with the form

$$H_r(K_i) = \text{floor}((A + K_i * B) \text{ Mod } C) / D).$$

This paper focuses on the Quotient Reduction method presented by R. Sprugnoli.

Notation: Floor(x) denotes the largest integer less than or equal to x.

CHAPTER III

DESCRIPTION OF THE ALGORITHM

It is assumed that the data set is static and that the elements of the set are in ascending order. Algorithm Q finds the best perfect hashing function by the Quotient Reduction method.

Algorithm Q

Given a finite set $I = \{w_1, w_2, \ldots, w_n\}$ of natural numbers, the method consists of translating the set I and dividing by some divisor N. This is stated as the following function, $H(w_i) = \text{floor}\ [(w_i + s)\ /N]$ for some integer, s, depending on I. The translation value, s, provides the situation where $H(w_1) = 0$ and avoids collisions.

Step A: Find the upper bound, N_o, for the divisor N.

A.1: Compute a difference table on the elements of set I. For all $w \in I$, compute the difference. $Diff[row, i] = w_j - w_i$ for every $i, j \in [1, n]$ and $1 \leq row \leq n-1$.

A.2: Using the difference table from step A.1, find the value of N_o based on the statement below:

$N_o = \min \{\text{floor}[(w_j - w_i - 1) / (j - i - 1)] : i, j \in [1, n]$ and $j > i + 1\}$.

Step B: Initialize the set Delta. Delta contains the set of all possible integer values from 1 to N_o.

Step C: Find a better approximation for N which will be less than or equal to N_O.

For two first differences where the sum is less than or equal to N_O

(LittleDelta[i] + Littledelta[j] $\leq N_O$)

do the following steps:

C.1: Initialize the set D to contain the integers from 1 to d where

d = LittleDelta[i] + LittleDelta[j] - 1.

C.2: Find the range of values for θ where

$\theta' \leq \theta \leq \theta''$.

$\theta' = m/N_O$ and $\theta'' = m/(d + 1)$

where $m = (w_j + 1) - w_{i+1}$ and

$M = w_{j+1} - (w_i + 1)$.

C.3: For all θ where $\theta' \leq \theta \leq \theta''$

do the following:

Take the union of the computed intervals based on θ with the set D.

$D = D \bigcup [\lceil m/\theta \rceil, \lfloor M/\theta \rfloor]$

C.4: The resulting Delta set is the intersection of the set Delta and the set D.

Step D: Find the maximum value of the set Delta, N. The larger the value of N, the smaller the table will be.

N = max (Delta)

Step E: Initialize the set $J = Z_N$. Z_N is the set of residues modulo N, i.e. the interval [0, N - 1].

Step F: Reduce the set J, by eliminating computed integer intervals from the set J.

For all the values of i from 1 to n - 1 where LittleDelta[i] less than N do the following:

$J = J \bigcap \{ (t - w_{i+1}) \bmod N$

where $0 \leq t < LitteDelta[i]\}$.

Step G: Check to see if $J = \emptyset$, if so go back to step D and drop the value N from the set Delta.

Step H: Find the element, t, of set J which minimizes the table length.

$t = Min \{(w_1 + t) \bmod N\}$

Step I: Find s for the hashing function

where $s = t - N * floor[(w_1 + t)/N]$.

Step J: Using the generated perfect hashing function, compute the values of the hash table.

A computer program for Algorithm Q is included in Appendix B. The following data is produced by the program and is presented here as Example 3.1 to explain the steps of Algorithm Q.

The input to the program is the data elements of the set I. In step A, the differences of the data elements are calculated and stored in a matrix. δ represents the first differences where

$\delta_i = w_{i+1} - w_i$ for every $i \in [1, n-1]$.

The variable name LittleDelta[i] in the program refers to δ_i.

Any difference $w_j - w_i$ for $i, j \in [1,n]$ and $j > i + 1$ can be represented in terms of the first differences.

(Note: In this paper the notation δ_i and LittleDelta[i] are used interchangeably and refer to the same thing. Also, the notation $[1,n] = \{1,2,3,\ldots,n\}$ is used throughout the paper.)

Example 3.1.

Consider the data set I = { 17, 138, 173, 294, 306, 472, 540, 551, 618}. The computer program for Algorithm Q is given in Appendix B. The following data is produced by the program and is presented here to explain the steps of Algorithm Q.

Step A:

Compute N_0 by means of Table 3.1 in steps A.1 and A.2.

Step A.1:

The difference table which contains the differences of the elements of I is given as Table 3.1. On each row the least difference $w_j - w_i$ is circled and used in calculating the array given in step A.2.

TABLE 3.1

DIFFERENCE TABLE

17	138	173	294	306	472	540	551	618
121	[35]	121	[12]	166	68	[11]	67	
156	156	133	178	234	79	(78)		
277	168	299	246	245	(146)			
289	334	367	(257)	312				
455	402	378	(324)					
523	(413)	445						
534	(480)							
(601)								

Step A.2:

The minimal Difference/Divided array is :

[77/(2-1)] = 77
[145/(3-1)] = 72
[256/(4-1)] = 85
[323/(5-1)] = 80
[412/(6-1)] = 82
[479/(7-1)] = 79
[600/(8-1)] = 85

72 is the minimum value which becomes N_o, the upper bound for N.

Step B:

Delta set is initialized as:

Delta = [1, 72].

Step C:

Find a better approximation for N which is less than or equal to N_o. For the two first differences where the sum is less than or equal to N_o, do the following steps:

Step C. 1:

Initialize the set D.

With i = 2 and j = 4

LittleDelta[2] = 35

LittleDelta[4] = 12.

The value of d = 46 and the set D = [1..46].

Step C. 2:

The interval [a, b] is computed for each value of Theta.

Theta	a	b
2	[61 ..	83]
3	[41 ..	55]

Step C. 3:

The union of the set D and the previous intervals gives the resulting set D:

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76	77
78	79	80	81	82	83						

Step C. 4:

This is set Delta updated as the result of the intersection with the set D in step C. 3.

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	61	62	63	64	65
66	67	68	69	70	71	72					

Step C. 1:

Initialize the set D.

With i = 2 and j = 7

LittleDelta[2] = 35

LittleDelta[7] = 11.

The value of d = 45 and the set D = [1..45].

Step C. 2:

The interval [a, b] is computed for each value of Theta.

Theta	a	b
6	[62 ..	68]
7	[53 ..	58]
8	[46 ..	51]

Step C. 3:

The union of the set D and the previous intervals gives the resulting set D:

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	53	54	55	56	57	58	62	63	64
65	66	67	68								

Step C. 4:

This is set Delta updated as the result of the intersection with set D in step C. 3.

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	53	54	55	62	63	64	65	66	67
68											

Step C. 1:

Initialize the set D.

With i = 4 and j = 7

LittleDelta[4] = 12

LittleDelta[7] = 11.

The value of d = 22 and the set D = [1..22].

Step C. 2:

The interval [a, b] is computed for each value of Theta.

Theta	a	b
4	[59 ..	64]
5	[47 ..	51]
6	[40 ..	42]
7	[34 ..	36]
8	[30 ..	32]
9	[27 ..	28]
10	[24 ..	25]
11	[22 ..	23]

Step C. 3:

The union of the set D and the previous intervals gives the resulting set D :

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	27	28	30	31	32	34	35	36	40	41	42
47	48	49	50	51	59	60	61	62	63	64	

Step C.4:

This is set Delta updated as the result of the intersection

with set D:

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	27	28	30	31	32	34	35	36	40	41	42
47	48	49	50	51	62	63	64				

This is the final Delta set.

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	27	28	30	31	32	34	35	36	40	41	42
47	48	49	50	51	62	63	64				

Step D:

Find the maximum value of the set Delta.

The maximum value in the set Delta is N = 64.

Step E:

Initialize the set J = Z_N.

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63								

Step F:

Eliminate certain computed intervals from the set J where

LittleDelta[i] is less than N.

When i = 2 then LittleDelta[i] = 35 which is < 64.

The interval is:

19	20	21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40	41	42
43	44	45	46	47	48	49	50	51	52	53	

The set J intersects with the previous interval giving:

19	20	21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40	41	42
43	44	45	46	47	48	49	50	51	52	53	

When i = 4 then LittleDelta[i] = 12 which is < 64.

The interval is:

14	15	16	17	18	19	20	21	22	23	24	25

The set J intersects with the previous interval giving:

19	20	21	22	23	24	25

When i = 7 then LittleDelta[i] = 11 which is < 64.

The interval is:

25	26	27	28	29	30	31	32	33	34	35

The set J intersects with the previous interval giving:

25

Step G:

Tests to see if the set J is empty.

Since J is not empty, the program proceeds to step H.

Step H:

Finds the element of J which provides the minimal table length.

t = 25 and T = (w[1] + t) mod N = 42

The value which minimizes the table length, t in set J, is 25.

Since the set J contains only one element, then the value of t is the value 25.

Step I:

Using the value t = 25, s is computed by

t - N * floor[(w_1 + t)/N] giving s := 25.

The best quotient reduction phf is $H(w_i) = \text{floor}\ [(w_i + s)/N]$

Given the computed s and N values

$$H(w_i) = \text{floor}\ [(w_i + 25)/64] .$$

Step J:

The values computed using the phf are :

w	17	138	173	294	306	472	540	551	618
H(w)	0	2	3	4	5	7	8	9	10

CHAPTER IV

ANALYSIS OF THE ALGORITHM AND SITUATIONS INCURRED

The quotient reduction method of finding a phf consists first of translating the set I and dividing by some divisor N; stated more formally,

$H(w_i)$ = floor[$(w_i + s) / N$] for some integer, s, depending on I.

The translation value, s, allows

1. for the first data element of the set I to be located at the beginning of the hash table, i.e. $H(w_1) = 0$, and
2. for the data elements w_i and w_j to be located at different hash values, i.e. $H(w_i) \neq H(w_j)$ (4.1) for every i, j ϵ [1,n] and $i \neq j$.

In step A, the upper bound for N, N_o, is based on the differences between the nonadjacent data elements of set I, given in the difference table.

The following argument shows that N_o is an upper bound for N. Given that $H(w_i) \neq H(w_j)$ for every i, j ϵ [1,n] and $j > i + 1$:

$$
\begin{aligned}
H(w_j) - H(w_i) &\geq j - i \\
H(w_j) - H(w_i) &> j - i - 1 \\
(w_j + s)/N - (w_i + s)/N &> j - i - 1 \\
(w_j + s - w_i - s)/N &> j - i - 1 \\
(w_j - w_i)/N &> j - i - 1 \\
w_j - w_i &> N (j - i - 1)
\end{aligned}
$$

$$w_j - w_i - 1 \geq N(j - i - 1)$$

$$[(w_j - w_i - 1) / (j - i - 1)] \geq N$$

$$\text{floor}[(w_j - w_i - 1) / (j - i - 1)] \geq N$$

$N \leq \text{floor}[(w_j - w_i - 1)/(j - i - 1)]$ for $i, j \in [1, n]$ and $j > i + 1$.

Let N_o equal the minimum value of the greatest integer values of the quantities $[(w_j - w_i - 1) / (j - i - 1)]$ for i and j in the range from 1 to n and where $j > i + 1$. Thus $N \leq N_o$.

Step C is a procedure to find a better approxiamation for N than N_o. The step is performed only when the sum of any two first differences is less than or equal to N_o ($\delta_i + \delta_j \leq N_o$). A set D denoted as D_{ij} contains all the values N satisfying (4.1) relative to i and j. Let D_{ij} be more formally defined as:

$$D_{ij} = [1, d] \cup \left(\bigcup \{ [\lceil m_{ij}/\theta \rceil, \lfloor M_{ij}/\theta \rfloor] : \theta' \leq \theta \leq \theta'' \} \right).$$

D_{ij} is the set of positive integers with a multiple in the interval $[m_{ij}, M_{ij}]$ where

$$m_{ij} = (w_j + 1) - w_{i+1} \quad \text{and}$$

$$M_{ij} = w_{j+1} - (w_i + 1).$$

The value N is the maximum value of the set Delta, where

$$\text{Delta} = [1, N_o] \cap \left(\bigcap \{ D_{ij} : i < j \} \right).$$

Since the length of the interval $[m_{ij}, M_{ij}]$ is a set of integers, let d be the length of the interval where

$$\begin{aligned} d &= M_{ij} - m_{ij} + 1 \\ &= w_{j+1} - (w_i + 1) - (w_j + 1) + w_{i+1} + 1 \\ &= w_{j+1} - w_i - 1 - w_j - 1 + w_{i+1} + 1 \\ &= \delta_j + \delta_i - 1 . \end{aligned}$$

When $d \geq N_o$, step C can be eliminated.

In Example 3.1, the upper bound, N_o for N is 72. Step C finds the integer intervals which reduces the largest possible value for N. These intervals are found based on computations when the sum of any two first differences is less than or equal to N_o. After taking the intersection of all the integer intervals, N is found to be 64, the maximum value in the set Delta.

After finding a value for N, the steps E and F determine the set J which contains the reduced admissible increment values, t, for computing the translation value, s. For every $w_i \in I$, an admissible increment for w_i is any integer t for which

$$\text{floor}[(w_{i+1} + t) / N] \neq \text{floor}[(w_i + t) / N] \qquad (4.2)$$

Thus condition (4.1) is satisfied for $j = i + 1$ and where $t = s$.

The set of all admissible increments for w_i is any translation value which adjusts w_i and w_{i+1} to two different intervals. The set can be defined by

$$J^*(w_i) = \{ u - w_{i+1} + kN \ : \ 0 \leq u < \delta_i \text{ and } k \in Z \}$$

Let $t = u - w_{i+1} + kN$

then $w_{i+1} + t = w_{i+1} + u - w_{i+1} + kN$

$= u + kN$

and $w_i + t = w_i + u - w_{i+1} + kN$

$= u - \delta_i + kN$.

The relation (4.2) with the corresponding substitutions made

for $w_{i+1} + t$ and $w_i + t$, is shown below,

floor $[(u + kN) / N] \neq$ floor $[(u - \delta_i + kN) /N]$

which holds for $0 \leq u < \delta_i$.

Since the term kN is not necessary in $J^*(w_i)$, the set $J(w_i)$ can be used to define the admissible increments for w_i.

$$J(w_i) = \{(t - w_{i+1}) \text{ Mod } N : 0 \leq t < \delta_i \} \qquad (4.3)$$

$J(w_i)$ only needs to be computed when $\delta_i < N$. If $\delta_i \geq N$ then $J(w_i) = Z_N$ and therefore no computation is necessary.

A quotient reduction phf can be found for I if and only if the set $J \neq \emptyset$, where $J = \bigcap \{J(w_i) : 1 \leq i \leq n - 1\}$.

CASE I The set J contains only one data element.

In Example 3.1, the first differences less than 64 are designated with a square around them in Table 3.1. For $\delta_2 = 35$, $\delta_4 = 12$, and $\delta_7 = 11$, the following $J(w_i)$'s are computed.

$J(138) = \{(t - 173) \text{ Mod } 64 : 0 \leq t < 35 \} = [19, 53]$

$J(294) = \{(t - 306) \text{ Mod } 64 : 0 \leq t < 12 \} = [14, 25]$

$J(540) = \{(t - 551) \text{ Mod } 64 : 0 \leq t < 11 \} = [25, 35]$

and $J = \{25\}$.

Since J has one element, the next step is to find the translation value, s, based upon t = 25, the element in set J. s is found by $s = t - N*\text{floor}[(w_1 + t)/ N]$

$= 25 - 64 * \text{floor}[(17 + 25)/64]$

$= 25.$

The quotient reduction phf is $H(w_i) = \text{floor}[(w_i + 25)/64]$. The complete output provided by the algorithm including the results of the phf is given in Example 3.1.

CASE II Set J contains more than one element.

Given a different set of data elements for I, the set J may contain more than one element.

Example 4.1

Consider the data set of I = { 10, 110, 200, 300, 400, 500, 602, 699, 2001}. The details of the output appear in Output 4.1. As a result of step A, the upper bound for N is found to be 114. Since the sum of any two first differences is greater than 114, there are no calculations to be performed in step C. Therefore the value of N is 114.

In steps E and F, the set J where

$$J = \bigcap \{J(w_i) : 1 \leq i \leq n - 1\} .$$

is determined according to (4.3). The first differences which are less than 114 are designated with a square around them in the difference table in step A.1 of Output 4.1. For δ_i where $i = 1,\ldots,7$ the following $J(w_i)$'s are computed for $i = 1,\ldots,7$.

$J(10) = \{(t - 110) \text{ Mod } 114 : 0 \leq t < 100\} = [4, 103]$

$J(110) = \{(t - 200) \text{ Mod } 114 : 0 \leq t < 90\} = [0, 3] \cup [28, 113]$

$J(200) = \{(t - 300) \text{ Mod } 114 : 0 \leq t < 100\} = [0, 27] \cup [42, 113]$

$J(300) = \{(t - 400) \text{ Mod } 114 : 0 \leq t < 100\} = [0, 41] \cup [56, 113]$

$J(400) = \{(t - 500) \text{ Mod } 114 : 0 \leq t < 100\} = [0, 55] \cup [70, 113]$

$J(500) = \{(t - 602) \text{ Mod } 114 : 0 \leq t < 102\} = [0, 69] \cup [82, 111]$

$J(602) = \{(t - 699) \text{ Mod } 114 : 0 \leq t < 97\} = [0, 81] \cup [99, 113]$ and $J = \{99, 100, 101, 102, 103\}$.

Since J has more than one element, one must find the element t in J which minimizes the table length. One does this by finding $\min\{(w_1 + t) \text{ Mod } N\}$. The value 99 gives the minimal value so t = 99 is used in finding s where

$$s = t - N * \text{floor}[(w_1 + t) / N]$$
$$= 99 - 114 * \text{floor}[(10 + 99)/114]$$
$$= 99 .$$

The quotient reduction phf is $H(w_1) = \text{floor}[(w_1 + 99)/114]$. The complete output and results are given in Output 4.1.

CASE III Set J is initially empty.

Example 4.2.

Consider the set I = { 2, 10, 20, 75, 83, 234, 335, 487, 589, 660, 713, 814, 976, 1077, 2219, 3240, 4341, 5442, 6533, 7677, 8734, 9885, 9998}. The details of the output are given in Output 4.2. In step A, the upper bound for N is found to be 17. The calculations in step C do not affect the value N.

Steps E and F performed the operations to find the set J using these three first differences which are less than 17: $\delta_1 = 8$, $\delta_2 = 10$, and $\delta_4 = 8$. These values can be found in the difference table in step A.1 of Output 4.2 with a square around them.

The following $J(w_i)$'s are computed for i = 1, 2, and 10.

$J(2) = \{(t - 10) \text{ Mod } 17 : 0 \leq t < 8\} = [7, 14]$

$J(10) = \{(t - 20) \text{ Mod } 17 : 0 \leq t < 10\} = [0,6] \cup [14,16]$

$J(75) = \{(t - 83) \text{ Mod } 17 : 0 \leq t < 8\} = [2,9]$

and $J = \bigcap J(w_i) : \{1 \leq i \leq n - 1\} = \emptyset$.

Since the set J is empty, the value N must be dropped from the set Delta and N becomes the next largest value in the set Delta. The steps E and F are repeated using N = 16. The set J is determined using the same w_i values as before since δ_1, δ_2, and δ_4 are less than 16.

The following intervals are computed for the $J(w_i)$'s:

$J(2) = \{(t - 10) \text{ Mod } 16 : 0 \leq t < 8\} = [6,13]$

$J(10) = \{(t - 20) \text{ Mod } 16 : 0 \leq t < 10\} = [0,5] \cup [12,15]$

$J(75) = \{(t - 83) \text{ Mod } 16 : 0 \leq t < 8\} = [0,4] \cup [13,15]$

and $J = \bigcap \{J(w_i) : 1 \leq i \leq n - 1\} = \{13\}$.

Since $J \neq \emptyset$, one can continue on with the next step of finding the values t and s. Since 13 is the only value in the set J, then t = 13. The value s is found by

$$s = t - N * \text{floor}[(w_i + t)/N]$$

$$= 13 - 16*\text{floor}[(2 + 13)/16]$$

$$= 13.$$

The quotient reduction phf is $H(w_i) = \text{floor}[(w_i + 13)/16]$. The results of this phf are given in Output 4.2 which follows.

Output 4.1

The following data produced by Alogoritm Q illustrates case II. Consider the data set of I = { 10, 110, 200, 300, 400, 500, 602, 699, 2001}.

Step A:

Compute N_o by means of Table 4.1 in steps A.1 and A.2.

Step A.1:

The difference table which contains the differences of the elements of I is given as Table 4.1. On each row the least difference $w_j - w_i$ is circled and used in calculating the array given in step A.2.

TABLE 4.1

DIFFERENCE TABLE

10	110	200	300	400	500	602	699	2001
[100]	[90]	[100]	[100]	[100]	[102]	[97]	1302	
(190)	190	200	200	202	199	1399		
(290)	290	300	302	299	1501			
(390)	390	402	399	1601				
(490)	492	499	1701					
592	(589)	1801						
(689)	1891							
1991								

Step A.2:

The minimal Difference/Divided array is:

189
144
129
122
117
114
284

114 is the minimum value which becomes N_o, the upper bound for N.

Step B:

Delta set is initialized as:

Delta = [1, 114].

This is the final delta set.

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108
109	110	111	112	113	114						

Step D:

Find the maximum value of the set Delta.

The maximum value in the set Delta is N = 114.

Step E:

Initialize the set J = Z_N.

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83
84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107
108	109	110	111	112	113						

Step F:

Eliminate certain computed intervals from the set J where

LittleDelta[i] is less than N.

When i = 1 then LittleDelta[1] = 100 which is < 114.

The interval is:

4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95	96	97	98	99
100	101	102	103								

The set J intersects with the previous interval giving:

4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95	96	97	98	99
100	101	102	103								

When i = 2 then LittleDelta[i] = 90 which is < 114.

The interval is:

0	1	2	3	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83
84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107
108	109	110	111	112	113						

The set J intersects with the previous interval giving:

28	29	30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95	96	97	98	99
100	101	102	103								

When i = 3 then LittleDelta[i] = 100 which is < 114.

The interval is:

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59	60	61
62	63	64	65	66	67	68	69	70	71	72	73
74	75	76	77	78	79	80	81	82	83	84	85
86	87	88	89	90	91	92	93	94	95	96	97
98	99	100	101	102	103	104	105	106	107	108	109
110	111	112	113								

The set J intersects with the previous interval giving:

42	43	44	45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76	77
78	79	80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99	100	101
102	103										

When i = 4 then LittleDelta[i] = 100 which is < 114.

The interval is:

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	56	57	58	59	60	61
62	63	64	65	66	67	68	69	70	71	72	73
74	75	76	77	78	79	80	81	82	83	84	85
86	87	88	89	90	91	92	93	94	95	96	97
98	99	100	101	102	103	104	105	106	107	108	109
110	111	112	113								

The set J intersects with the previous interval giving:

56	57	58	59	60	61	62	63	64	65	66	67
68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91
92	93	94	95	96	97	98	99	100	101	102	103

When i = 5 then LittleDelta[i] = 100 which is < 114.

The interval is:

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	70	71	72	73
74	75	76	77	78	79	80	81	82	83	84	85
86	87	88	89	90	91	92	93	94	95	96	97

The interval is:

```
  0   1   2   3   4   5   6   7   8   9  10  11
 12  13  14  15  16  17  18  19  20  21  22  23
 24  25  26  27  42  43  44  45  46  47  48  49
 50  51  52  53  54  55  56  57  58  59  60  61
 62  63  64  65  66  67  68  69  70  71  72  73
 74  75  76  77  78  79  80  81  82  83  84  85
 86  87  88  89  90  91  92  93  94  95  96  97
 98  99 100 101 102 103 104 105 106 107 108 109
110 111 112 113
```

The set J intersects with the previous interval giving:

```
 42  43  44  45  46  47  48  49  50  51  52  53
 54  55  56  57  58  59  60  61  62  63  64  65
 66  67  68  69  70  71  72  73  74  75  76  77
 78  79  80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99 100 101
102 103
```

When i = 4 then LittleDelta[i] = 100 which is < 114.

The interval is:

```
  0   1   2   3   4   5   6   7   8   9  10  11
 12  13  14  15  16  17  18  19  20  21  22  23
 24  25  26  27  28  29  30  31  32  33  34  35
 36  37  38  39  40  41  56  57  58  59  60  61
 62  63  64  65  66  67  68  69  70  71  72  73
 74  75  76  77  78  79  80  81  82  83  84  85
 86  87  88  89  90  91  92  93  94  95  96  97
 98  99 100 101 102 103 104 105 106 107 108 109
110 111 112 113
```

The set J intersects with the previous interval giving:

```
 56  57  58  59  60  61  62  63  64  65  66  67
 68  69  70  71  72  73  74  75  76  77  78  79
 80  81  82  83  84  85  86  87  88  89  90  91
 92  93  94  95  96  97  98  99 100 101 102 103
```

When i = 5 then LittleDelta[i] = 100 which is < 114.

The interval is:

```
  0   1   2   3   4   5   6   7   8   9  10  11
 12  13  14  15  16  17  18  19  20  21  22  23
 24  25  26  27  28  29  30  31  32  33  34  35
 36  37  38  39  40  41  42  43  44  45  46  47
 48  49  50  51  52  53  54  55  70  71  72  73
 74  75  76  77  78  79  80  81  82  83  84  85
 86  87  88  89  90  91  92  93  94  95  96  97
```

```
  98   99  100  101  102  103  104  105  106  107  108  109
 110  111  112  113
```

The set J intersects with the previous interval giving:

```
  70   71   72   73   74   75   76   77   78   79   80   81
  82   83   84   85   86   87   88   89   90   91   92   93
  94   95   96   97   98   99  100  101  102  103
```

When i = 6 then LittleDelta[i] = 102 which is < 114.

The interval is:

```
   0    1    2    3    4    5    6    7    8    9   10   11
  12   13   14   15   16   17   18   19   20   21   22   23
  24   25   26   27   28   29   30   31   32   33   34   35
  36   37   38   39   40   41   42   43   44   45   46   47
  48   49   50   51   52   53   54   55   56   57   58   59
  60   61   62   63   64   65   66   67   68   69   82   83
  84   85   86   87   88   89   90   91   92   93   94   95
  96   97   98   99  100  101  102  103  104  105  106  107
 108  109  110  111  112  113
```

The set J intersects with the previous interval giving:

```
  82   83   84   85   86   87   88   89   90   91   92   93
  94   95   96   97   98   99  100  101  102  103
```

When i = 7 then LittleDelta[i] = 97 which is < 114.

The interval is:

```
   0    1    2    3    4    5    6    7    8    9   10   11
  12   13   14   15   16   17   18   19   20   21   22   23
  24   25   26   27   28   29   30   31   32   33   34   35
  36   37   38   39   40   41   42   43   44   45   46   47
  48   49   50   51   52   53   54   55   56   57   58   59
  60   61   62   63   64   65   66   67   68   69   70   71
  72   73   74   75   76   77   78   79   80   81   99  100
 101  102  103  104  105  106  107  108  109  110  111  112
 113
```

The set J intersects with the previous interval giving:

```
  99  100  101  102  103
```

Step G:

Tests to see if the set J is empty.

Since J is not empty, the program proceeds to step H.

Step H:

Finds the element of J which provides the minimal table length.

```
t =  99 and T = (w[1] + t) mod N = 109
t = 100 and T = (w[1] + t) mod N = 110
t = 101 and T = (w[1] + t) mod N = 111
t = 102 and T = (w[1] + t) mod N = 112
t = 103 and T = (w[1] + t) mod N = 113
```

The value which minimizes the table length in set J is 99.

Step I:

Using the value t = 99, s is computed by

$$t - N * \text{floor}[(w_i + t)/N] \text{ giving } s := 99.$$

The best quotient reduction phf is $H(w_i) = \text{floor}[(w_i + s)/N]$.

Given the computed s and N values

$$H(w_i) = \text{floor}[(w_i + 99)/ 114)$$

Step J:

The values computed using the phf are :

w	10	110	200	300	400	500	602	699	2001
H(w)	0	1	2	3	4	5	6	7	18

Output 4.2

Consider the data set I = { 2, 10, 20, 75, 83, 234, 335, 487, 589}. The following data produced by Algorithm Q is presented to help illustrate case III.

Step A:

Compute N_o by means of Table 4.2 in steps A.1 and A.2.

Step A.1:

The difference table which contains the differences of the elements of I is given as Table 4.2. On each row the least difference $w_j - w_i$ is circled and used in calculating the array given in step A.2.

TABLE 4.2

DIFFERENCE TABLE

2	10	20	75	83	234	335	487	589
[8]	[10]	55		[8]	151	101	152	102
(18)	65	63		159	252	253	254	
(73)	73	214		260	404	355		
(81)	224	315		412	506			
(232)	325	467		514				
(333)	477	569						
(485)	579							
(587)								

Step A.2:

The minimal Difference/Divided array is:

17
36
26
57
66
80
83

17 is the minimum value which becomes N_o, the upper bound for N.

Step B:

Delta set is initialized as:

Delta = [1,17].

Step C:

Find a better approximation for N which is less than or equal to N_o. For the two first differences where the sum is less than or equal to N_o, do the following steps:

Step C.1:

Initialize the set D.

With i = 1 and j = 4

LittleDelta[1] = 8

LittleDelta[4] = 8.

The value of d = 15 and the set D = [1..15].

Step C.2:

The interval [a,b] is computed for each value of Theta.

Theta	a	b
4	[17 ..	20]
5	[14 ..	16]

Step C.3:

The union of the set D and the previous intervals gives the resulting set D :

1 2 3 4 5 6 7 8 9 10 11 12
13 14 15 16 17 18 19 20

Step C.4:

This is set Delta updated as the result of the intersection with set D in step C.3:

1 2 3 4 5 6 7 8 9 10 11 12
13 14 15 16 17

This is the final delta set.

1 2 3 4 5 6 7 8 9 10 11 12
13 14 15 16 17

Step D:

Find the maximum value of the set Delta.

The maximum value in the set Delta is N is 17.

Step E:

Initialize the set J = Z_N.

0 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16

Step F:

Eliminate certain computed intervals from the set J where

LittleDelta[i] is less than N.

When i = 1 then LittleDelta[i] = 8 which is < 17.

The interval is:

7 8 9 10 11 12 13 14

The set J intersects with the previous interval giving:

7 8 9 10 11 12 13 14

When i = 2 then LittleDelta[i] = 10 which is < 17.

The interval is:

0 1 2 3 4 5 6 14 15 16

The set J intersects with the previous interval giving:

14

When i = 4 then LittleDelta[i] = 8 which is < 17.

The interval is:

2 3 4 5 6 7 8 9

The set J intersects with the previous interval giving:

The empty set.

Step G:

Tests to see if set J is empty.

Since the set J is empty the value of N becomes 16 and the program goes back to step D.

Step D:

The maximum value in the set Delta is N = 16.

Step E:

Initialize the set J = Z_N.

0 1 2 3 4 5 6 7 8 9 10 11
12 13 14 15

Step F:

Eliminate certain computed intervals from the set J where LittleDelta[i] is less than N.

When i = 1 then LittleDelta[i] = 8 which is < 16.

The interval is:

6 7 8 9 10 11 12 13

The set J intersects with the previous interval giving:

6 7 8 9 10 11 12 13

When i = 2 then LittleDelta[i] = 10 which is < 16.

The interval is:

0 1 2 3 4 5 12 13 14 15

The set J intersects with the previous interval giving:

12 13

When i = 4 then LittleDelta[i] = 8 which is < 16.

The interval is:

0 1 2 3 4 13 14 15

CHAPTER V

LIMITATIONS OF THE ALGORITHM

Every algorithm has its limitations. Perfect hash functions are feasible only for static sets of keys. The addition of one new key will usually require a new function to be defined for the entire set. The algorithm Q presented in this paper works well for small data sets. When the data set consists of more than twenty keys, the resulting hash table is much larger than the number of keys. One method to overcome the size limitation is to partition the larger sets of keys into segments of about ten keys and compute a perfect hash function for each segment.

The following sets of data show that the hash table is sparse when the data set is larger. The phf given as H(w) = floor[(w + 14)/17] produces the hash table below on these 9 data elements:

w	2	10	20	45	83	134	154	187	209
H(w)	0	1	2	3	5	8	9	11	13

When the data set is enlarged to contain 23 data elements; as given below, the following hash table is produced based on the phf of H(w) = floor[(w + 0)/9].

w	2	10	20	45	83	134	154	187	209	260	309	394
H(w)	0	1	2	5	9	14	17	20	23	28	34	43

cont.

w	406	482	539	540	554	559	677	721	878	905	999
H(w)	45	53	59	60	61	62	75	80	97	100	111

The data set of 9 elements is stored in a hash table of size 14. Therefore, the loading factor is 9/14 and the hash table is 64 percent full. When the data set contains 23 elements the hash table is only 20.5 percent full. Other data sets examined by the author have given similiar results. Therefore, one can say that the Algorithm Q appears to perform well on small data sets.

In order to have an almost full table for a given set I, one should have N approximately equal to $(w_n - w_1)/(n-1)$. If the elements in set I are not uniformly distributed, then N may be considerably smaller than this best value. When N is smaller, the table is sparse. Two data sets are given below. The second data set contains two areas where the data elements are adjacent and the last data element is greater than the previous element by eleven units. Thus, the second set can be considered less uniform than the first or, in other words, nonuniform.

The perfect hash function H(w) = floor[(w + s)/N] when s = 1 and N =4 produces the following hash table on the given data set:

w	2	5	10	13	18	19
H(w)	0	1	2	3	4	5

Besides being a phf, this function is also a minimal phf since the mapping between the data elements and the locations in the hash table is one-to-one and onto. Notice that, in this example, the value of N in the function is equal to the ceiling function of $(w_n - w_1)/(n - 1)$.

The next data set is not uniform in comparison to the previous one. The function H(w) = floor[(w + 1)/4] produces the following hash table which is not full.

w	2	3	10	11	18	29
H(w)	0	1	2	3	4	7

In this function, the value of N is also 4. But this time, $(w_n - w_1)/(n - 1)$ evaluates to be 5.4; therefore, N is less than the ceiling function of $(w_n - w_1)/(n - 1)$. This illustrates that when N is smaller than $(w_n - w_1)/(n - 1)$ then the hash table is not full.

One way to overcome the situation of a nonuniform data set is to translate some of the values larger than a certain value before applying the quotient reduction algorithm. The next chapter discusses the process of finding the cut point, t, where $1 \leq t \leq n - 1$. This cut point t will determine the cut value w_t. Along with finding the cut point t, an integer r, which is an additional translation value for w_i where $i > t$, must also be found which produces a hash table of minimal length.

In summary, there are two major limitations of the quotient reduction algorithm presented as Algorithm Q.

1. In general, the algorithm usually produces a hash table significantly larger than the set of data when the data set contains more than 20 elements.
2. The hash table is sparse if the elements of the data set are not uniformly distributed.

It is natural to consider a modification of the algorithm which could be termed "rehashing". An example is presented below to explain the procedure and the results.

Example 5.1

Consider the data set below. The hash function H(w) = floor[(w + 0)/9] produces the corresponding hash table.

w	2	10	20	45	83	134	154	187	209	260	309	394
H(w)	0	1	2	5	9	14	17	20	23	28	34	43

cont.

w	406	482	539	540	554	559	677	721	878	905	999
H(w)	45	53	59	60	61	62	75	80	97	100	111

The procedure was to use the values of the hash table as input to find another hash function with the idea of producing a denser table. The rehashed function produced was of the form $H_2(w)$ = floor[(w + 0)/1] so the resulting hash table $H_2(H(w))$ equals the hash table H(w). The example shows that this procedure did not provide better hash values.

CHAPTER VI

QUOTIENT REDUCTION ALGORITHM WITH A CUT

Given a set I, this algorithm determines the cut point, t, and the integer, r, allowing the best cut reduction phf of the form

$$H(w_i) = \text{floor}[(w_i + s) / N] \quad \text{for all } i \leq t$$

$$H(w_i) = \text{floor}[(w_i + s + r) / N] \quad \text{for all } i > t$$

for some integers, s and r.

This algorithm finds the cut point, t, and the integer, r, which produces the minimal table length. The obvious approach for finding the cut point is to try all the elements $w_1, w_2, \ldots, w_{n-1}$ successively as the possible cut value, w_t. This leads to the following Algorithm C. In the following text, N[t] corresponds to the divisor N of chapter 3 for each cut point t where $t = 1, \ldots, n - 1$.

Algorithm C

Step A: Find the upper bound, N_0, for each N[t].

A.0: Initialize t. Let $t = 1$, thus w_1 is the first cut value.

A.1: Compute the difference table for all $w_i \in I$ where $i \leq t$ or $i > t$. Ignore the differences of elements of I on opposite sides of the cut point (ignore couples w_i, w_j such that $i \leq t < j$).

A. 2: Using the difference table from step A. 1, find the value of N_o where

N_o = min {floor[$(w_j - w_i - 1) / (j - i - 1)$] : (i, j ϵ [1, t] or i, j ϵ [t+1, n]) and j > i + 1}.

Step B: Initialize the set Delta to the set of all possible integer values from 1 to N_o as in Algorithm Q.

Step C: Find the set Delta exactly like the steps C. 1 through C. 4 of Algorithm Q.

Step D: Let N[t] equal the maximum value of the set Delta.

N[t] = max (Delta)

Step E: Initialize the set JL and JR to set of residues modulo N[t]; JL = $Z_{N[t]}$ and JR = $Z_{N[t]}$.

Step F: Determine the sets JL and JR where each contains the set of reduced admissible increments relative to N[t]. For the values of i where LittleDelta[i] less than N[t] do the following:

JL = $\bigcap$ {$(v - w_{i+1})$ Mod N[t] : $0 \leq v < \delta_i$ and $1 \leq i \leq t - 1$}

and;

JR = $\bigcap$ {$(v - w_{i+1})$ Mod N[t] : $0 \leq v < \delta_i$ and $t + 1 \leq i \leq n - 1$}.

Step G: Check to see if either the set JL = $\emptyset$ or JR = $\emptyset$, if so go back to step D and drop the value N[t] from the set Delta.

Step H: Determine δ_o, the lower bound for δ_t', by finding the maximum value of the following set:

$\delta_o = \max \{(j - i - 1)N[t] + 1 - w_j + w_i + \delta_t : i \leq t < j\}$.

Step I: Determine the following values:

I.1: $\hat{p} = \min \{ p \in [1, N[t]] : (-w_t - p) \text{ Mod } N[t] \in JL\}$

I.2: Let s' be the element in JL which corresponds to the value of $\hat{p}$.

I.3: $\delta_t' = \min \{\delta \geq \delta_o :$ there exits $j'' \in JR$ such that $\delta \equiv (w_{t+1} + j'' + \hat{p}) \text{ Mod } N[t]\}$.

Step J: Find the values of r[t] and s[t] for the cut point t. These are the translation values used in the phf, defined below as:

$r[t] = \delta_t' - \delta_t$

$s[t] = s' - N[t] * \text{floor}[(w_1 + s') / N]$.

For the values of r[t] and s[t] determine the length of the table where

$L[t] = \text{floor}[(w_n + s[t] + r[t]) / N[t]] - 1$.

Step K: Increment the value of t by 1 and go to step A.1 if t is less than n.

Step L: Now that the table length has been computed for each cut point, t, find the minimum L[t]. Let z be the index for which L[z] is the minimum value of all the L[t] values, thus the cut point value is w_z. Using the values of s = s[z], r = r[z], and N = N[z], the phf H is defined by:

$H(w_i)$ = floor[$(w_i + s) / N$] for all $i \leq t$

$H(w_i)$ = floor[$(w_i + s + r) / N$] for all $i > t$

where t = z, the cut point.

Step M: Using the generated phf, compute the values of the hash table.

A computer program for Algorithm C is included in Appendix C. The input used in Example 3.1 is also used in Example 6.1 for comparison. The steps A through E are essentially the same as in Algorithm Q except the fact that these steps are executed n - 1 times as opposed to 1 time. Steps F through K are also executed as part of the same loop. Steps L and M are executed after the value of t becomes equal to n.

Algorithm C differs from Algorithm Q as follows. In step F, two sets JL and JR are set up to find the admissible increments on the data elements of set I. The translation value, s, is determined from the set of values in JL. The minimum value for the translation value, r, is based on the set of differences $w_j - w_i$ where $i \leq t < j$.

The final value for r is determined by both sets, JL and JR.

Step G is a procedure to determine if either set, JL or JR, is empty. If one is empty then the value of N[t] must be deleted from the set Delta. The program loops back to step D and the next largest value in the set Delta is assigned to N[t]. Steps E and F are repeated until both sets JL and JR contain at least one element each.

Steps H, I, and J find the values for the translation values r[t] and s[t]. The determination of $r[t] = \delta_t' - \delta_t$ where δ_t is the difference of $w_{t+1} - w_t$. The lower bound, δ_0, for δ_t' is found in step H. Then in step I.3 the actual value for δ_t' is determined. In steps I.1 and I.2, the value of $\hat{p}$ is determined and the value $s' \in JL$ which correponds to the value of $\hat{p}$. In step J the calulations are performed to find r[t] and s[t]; and then, using these values the length of the table, L[t], is determined.

In step K, the value of t is incremented by one and the steps A.1 through J are repeated until t becomes equal to n. The steps A.1 through J are therefore executed n - 1 times and the values s[t], r[t], and N[t] are computed for $t = 1, \ldots, n - 1$. L[t] determines the table length for each cut point t where $1 \leq t < n$. The minimal table length, L[z], is minimal value of L[t]. In step L, s is assigned the value s[z], r is assigned r[z], N is assigned N[z], and the cut value , w_t, is w_z.

The perfect hash function is given in step L with the value t understood to be z where

$$L[z] = \min \{L[t] : 1 \leq t < n\}.$$

The last step M uses the phf to calculate the hash table.

The following example uses the same data set as in Example 3.1. Although the steps A.1 through J are executed n - 1 times, n being 9, only the step where the cut point t = 5 is given in Example 6.1.

(Note: In this paper the notation δ_0 and LittleDelta[0] are used interchangeably and refer to the same thing.)

Example 6.1

Consider the data set I = { 17, 138, 173, 294, 306, 742, 540, 551, 618}. The following data is presented to explain the steps of Algorithm C.

Step A:

Find the upper bound, N_O, for each N[t] by means of Table 6.1 in steps A.1 and A.2.

Step A.1:

The difference table which contains the differences of the elements in I where $i \leq t$ or $i > t$. The differences between elements in I on opposite sides of the cut point are ignored. On each row the least difference $w_j - w_i$ is circled and used in calculating the array given in step A.2. When the cut point equals 5 the following table is computed:

TABLE 6.1

DIFFERENCE TABLE

17 138 173 294 306 472 540 551 618

121 [35] 121 [12] ___ 68 [11] 67

156 156 133 ___ ___ 79 (78)

277 168 ___ ___ ___ (146)

(289) ___ ___ ___ ___

___ ___ ___ ___

___ ___ ___

___ ___

___

Step A.2:

The minimal Difference/Divided array is :

77
72
96

72 is the minimum value which becomes N_0, the upper bound for N[5].

Step B:

Delta set is initialized as:

Delta = [1, 72].

Step C:

Find a better approxiamation for N[5] which is less than or equal to N_0. For the two first differences where the sum is less than or equal to N_0,

(LittleDelta[i] + LittDelta[j] $\leq N_0$).

If $i \leq t < j$ then these first differences are ignored and no computation is done with these values; otherwise, do the following steps:

Step C. 1:

Initialize the set D.

With i = 2 and j = 4

LittleDelta[2] = 35

LittleDelta[4] = 12.

The value of d = 46 and the set D = [1..46].

Step C. 2:

The interval [a, b] is computed for each value of Theta.

Theta	a	b
2	[61 ..	83]
3	[41 ..	55]

Step C. 3:

The union of the set D and the previous intervals gives the resulting set D:

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76	77
78	79	80	81	82	83						

Step C.4:

This is set Delta updated as the result of the intersection with the set D in step C.3.

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	61	62	63	64	65
66	67	68	69	70	71	72					

In Example 3.1 the steps C.1 through C.4 are executed two more times, but in this algorithm these other calculations are ignored. When LittleDelta[2] = 35 and LittleDelta[7] = 11, no calculations are performed since $i \leq 5 < j$ where $i = 2$ and $j = 7$. Also when LittleDelta[4] = 12 and LittleDelta[7] = 11, no calculations are performed since $i \leq 5 < j$ where $i = 4$ and $j = 7$.

This is the final delta set.

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	61	62	63	64	65
66	67	68	69	70	71	72					

Step D:

Find the maximum value of the set Delta.

The maximum value in the set Delta is N = 72.

Step E:

Initialize the sets JL = $Z_{N[t]}$ and JR = $_{N[t]}$ which is given as follows:

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69	70	71

Step F:

Determine the sets JL and JR where each contains the set of reduced admissible increments relative to N[t].

The set JL is determined first.

When i = 2 then LittleDelta[i] = 35 which is < 72.

The interval is:

0	1	2	3	4	5	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	

The set JL intersects with the previous interval giving:

0	1	2	3	4	5	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	

When i = 4 then LittleDelta[i] = 12 which is < 72.

The interval is:

54	55	56	57	58	59	60	61	62	63	64	65

The set JL intersects with the previous interval giving:

54	55	56	57	58	59	60	61	62	63	64	65

The set JR is determined next.

When i = 6 then LittleDelta[i] = 68 which is < 72.

The interval is:

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71				

The set JR intersects with the previous interval giving:

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	36	37	38	39
40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71				

When i = 7 then LittleDelta[i] = 11 which is < 72.

The interval is:

25	26	27	28	29	30	31	32	33	34	35

The set JR intersects with the previous interval giving:

25	26	27	28	29	30	31

When i = 8 then LittleDelta[i] = 67 which is < 72.

The interval is:

0	1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22	23
24	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52
53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71					

The set JR intersects with the previous interval giving:

30	31

Step G:

Tests to see if either the set JL or the set JR is empty.

Since neither one is empty, the program proceeds to step H.

Step H:

Determine LittleDelta[o], the lower bound for δ_t',

for t = 5 from the following values.

1
61
56
126
131
62
119
70

The maximum value is 131 which is LittleDelta[o].

Step I.1 and I.2:

Find the following two values.

$\hat{p}$ = 61

s' = 65

Step I.3:

δ_5' = 131.

Step J:

When the cut point t = 5, the length of the hash table,

L[t], equals 7.

r[5] = -35
s[5] = -7
N[5] = 72

Step K:

Increments the value of t by 1 and goes back to step A.1 if

t is less than n, the number of data elements in set I.

Step L:

The minimal lenth of table is 7 where z = 5.

The values corresponding to the value of z = 5 are :

s = -7
r = -35

The cut point element w is 306.

The value of N is 72.

The perfect hashing funtion is $H(w_i) = floor[(w_i + s)/N]$

$H(w_i) = floor[(w_i + -7)/ 72]$ for $i \leq 5$

$H(w_i) = floor[(w_i + -7 + -35)/ 72]$ for $i > 5$

Step M:

The values computed using the phf are :

w	17	138	173	294	306	472	540	551	618
H(w)	0	1	2	3	4	5	6	7	8

The hash table size is 9 which is the same as the number of data elements. The function provided is a one-to-one and onto mapping from the domain to the range and is a minimal perfect hashing function.

The following data sets are given to compare the hash tables produced by Algorithm Q and C. The first hash table produced by Algorithm Q is given below as presented in Output 4.1. The values of the second hash table are produced by Algorithm C using the following function.

$H(w) = floor[(w + 103)/114]$ for $i \leq 8$

$H(w) = floor[(w + 103 - 1192)/114]$ for $i > 8$

	w	10	110	200	300	400	500	602	699	2001
Q	H(w)	0	1	2	3	4	5	6	7	18
C	H(w)	0	1	2	3	4	5	6	7	8

The next data set and its first hash table is reproduced from Output 4.2. The second hash table shown is produced by Algorithm C using the function where

$H(w) = \text{floor}[(w + 59)/62]$ for $i \leq 2$

$H(w) = \text{floor}[(w + 59 + 106)/62]$ for $i > 2$.

	w	2	10	20	75	83	234	335	487	589
Q	H(w)	0	1	2	5	6	15	21	31	37
C	H(w)	0	1	2	3	4	6	8	10	12

In both examples Algorithm C does produce a better hash table. Other data sets examined by the author have given similiar results.

CHAPTER VII

CONCLUSION

This paper has examined an algorithm for producing a perfect hashing function and a modification to this algorithm. Two modifications, rehashing and using a cut point to divide the data set, were investigated. First, the idea of rehashing, as presented in Example 5.1, was explored as a possible improvement on Algorithm Q with no benefit apparent. Next, the Algorithm C was presented which did illustrate improvements over Algorthm Q. The hash tables produced by Algorithm C are smaller and, therefore, the hash table is less sparse.

History

The study and development of storage and retrieval algorithms is a recent phenomenon. It was only in the 1950's that the notion of hashing was developed. The idea of hashing appears to have been originated by H.P. Luhn, who wrote an internal IBM memorandum suggesting the use of chaining, in January 1953. This was among the first applications of linked linear lists. At about the same time the idea of hashing occurred independently to another group of IBM people. The group consisted of Arthur L. Samuel, Gene M. Amdahl, Elaine M. Boehme, and Nathaniel Rochester who were building an assembler for the IBM 701. They decided to try key-value-to-address mapping for managing the

symbol table in their assembler. In order to handle the collision problem, Amdahl discovered the idea of open addressing with linear probing.

Hash coding was first described in open literature by Arnold I. Dumey in Computers and Automation in December 1956. He was the first to mention the idea of dividing by a prime number and using the remainder as the hash address. By the end of 1963, several different schemes had been developed.

During the next few years hashing became widely used but little was published about it. Then in 1968, Robert Morris wrote an influential survey of the subject. Knuth, in [22], noted, "The word "hashing" never appeared in print, with its present meaning, until Morris's article was published in 1968...". Somehow the verb "to hash" which means to chop into pieces (hash is a random jumble achieved by hashing) became standard terminology for key-to-address transformation during the mid 1960's. But no one would use the term publicly until 1968.

Applications

The area of compiler construction is one application for hashing function. The table lookup procedure uses a hashing function for the reserved words in a compiler. In natural language processing it is desirable to have direct random access to the database which is a fixed vocabulary of frequently used words. Perfect hash functions will

organize the database so that only a single probe is necessary to retrieve any data item.

Minimal Perfect Hashing Functions

In 1986, Berman [3] states that every set of keys has several minimal perfect hashing functions; the problem is that we may not know how to compute any of them in constant time, as we would like for a hashing function. Several articles, Jaeschke [18], Cichelli [12], Cook [13], and Sager [28], describe procedures which take as input a set of keys and try to adjust numeric parameters in an arithmetic function in order to make it a minimal perfect hashing function for the given set of keys.

In [18], Jaeschke presents a method for generating minimal perfect hashing function called reciprocal hashing. In his paper, Jaeschke assures the existence of minimal perfect hashing function. Reciprocal hashing is not efficient with respect to time and storage requirements for data of size greater than 2000 elements. Another article by Chang [10] discusses two modifications of Jaeschke's method. The method presented is based upon the Chinese Remainder Theorem and prime number functions.

From the research on finding minimal perfect hash functions, Cichelli [12] came up with the following moral: "When all else fails, try brute force". Not everyone agrees with this moral, as Sager [28] suggest that the following moral is more appropriate: "With adequate forethought, brute force solutions can usually be avoided".

BIBLIOGRAPHY

[1] Anderson, M. G. and Anderson, M. R. "Comments on Perfect Hashing Functions: A Single Probe Retrieving Method for Static Sets," Communications of the ACM 22, 2 (February 1979), 104.

[2] Augenstein, M. J. and Tenenbaum, A. M. Data Structures Using Pascal, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

[3] Berman, B., Bock, M. E., Ditter, E., O'Donnell, M. J., and Plank, D. "Collections of Functions for Perfect Hashing," SIAM Journal on Computing 15, 2 (May 1986), 604-618.

[4] Bloom, B. H. "Space/Time Trade-offs in Hash Coding with Allowable Errors," Communications of the ACM 13, 7 (July 1970), 422-426.

[5] Buchholz, W. "File Organization an Addressing," IBM System Journal, 2 (June 1963), 86-111.

[6] Cerone, N., Krause, M., and Boates, J. "An Interactive System for Finding Perfect Hash Functions," IEEE Software (November 1985), 38-53.

[7] Cerone, N., Krause, M., and Boates, J. "Minimal and Almost Minimal Perfect Hash Function Search with Application to Natural Language Lexicon Design," Computers and Mathematics with Applications 9, 1 (January 1983), 215-231.

[8] Chang, C. C. "A Scheme for Constructing Ordered Minimal Perfect Hashing Functions," Information Sciences 39, 2 (1986), 187-195.

[9] _____. "An Ordered Minimal Perfect Hashing Scheme Based upon Euler's Theorem," Information Sciences 32, 3 (1984), 165-172.

[10] _____. "The Study of an Ordered Minimal Perfect Hashing Scheme," Communications of the ACM 27, 4 (April 1984), 384-387.

[11] Cichelli, R. J. "Author's Response to Technical Corrspondence," Communications of the ACM 23, 12 (December 1980), 729.

[12] _____. "Minimal Perfect hash Functions Made Simple," Communications of the ACM, 23, 1 (January 1980), 17-19.

[13] Cook, C. R. "A Letter Oriented Minimal Perfect Hashing Function," Sigplan Notices, 17, 9 (September 1982), 18-27.

[14] Cook, C. R. and Oldehoeft, R. "More on Minimal Perfect Hash Tables," 13th Southeastern Conference on Combinatorics, Graph Theory, and Computing, (1982).

[15] Cormack, G. V., Horspool, R. N. S., and Kaiserswerth, M. "Practical Perfect Hashing," The Computer Journal 28, 1 (1985), 54-58.

[16] Du, M. W., Jea, K. F., and Shieh, D. W. "The Study of a New Perfect Hash Schemes," Proceedings of COMPSAC 1980, Chicago, IL, (October 1980), 341-347.

[17] Ghosh, S. P. Data Base Organization for Data Management Academic Press, New York, 1977.

[18] Jaeschke, G. "Reciprocal Hashing: A Method for Generating Minimal Perfect Hahsing Functions," Communications of the ACM 24, 12 (December 1981), 829-833.

[19] Jaeschke, G. and Osterburg, G. "On Cichelli's Minimal Perfect Hash Function Method," Communications of the ACM 23, 12 (December 1980), 728-729.

[20] Johnson, L. R. "An Indirect Chaining Method for Addressing on Secondary Keys," Communications of the ACM 4, 5 (May 1961), 218-222.

[21] Knott, G. D. "Hashing Functions," Computer Journal 18, 3 (August 1985), 265-278.

[22] Knuth, D. E. The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, Reading, MA, 1973.

[23] Lum, V. Y., Yuan, P. S. T., and Dodd, M. "Key-to-address Transformation techniques: A Fundamental Performance Study on Large Existing Formatted Files," Communications of the ACM 14, 4 (April 1971), 228-239.

[24] Maurer, W. D. and Lewis, T. G. "Hash Table Method," Computing Surveys 7, 1 (March 1975), 5-19.

[25] Morris, R. "Scatter Storage Techniques," Communications of the ACM 11, 1 (January 1968), 34-44.

[26] Olsen, C. A. "Random Access File Organization for Indirectly Accessed Records," Proceedings 1969 ACM 24th National Conference, 539-549.

[27] Peterson, W. W. "Addressing for Random-Access Storage," IBM Journal of Research and Development 1, 2 (April 1957), 130-146.

[28] Sager, T. J. "A Polynomial Time Generator for Minimal Perfect Hash Function," Communications of the ACM 28, 5 (May 1985), 523-532.

[29] Schay, G. and Spruth, W. G. "Analysis of a File Addressing Method," Communications of the ACM 5, 8 (August 1962), 459-462.

[30] Severance, D. G. "Identifier Search Mechanisms: A Survey and Generalized Model," Computing Surveys 6, 3 (September 1974), 175-194.

[31] Sprugnoli, R. "Perfect Hashing Functions: A Single Probe Retrieving Method for Static Sets," Communications of the ACM 20, 11 (November 1977), 841-850.

[32] Trainiter, M. "Addressing for Random-Access Storage with Multiple Bucket Capacities," Journal of the ACM 10, 3 (July 1963), 307-315.

APPENDIX A

NOTATION

A list of notational conventions used in this thesis is given here. The symbol ':=:' denotes 'defined to be'.

floor (x)	:=:	the largest integer less than or equal to x, $\lfloor x \rfloor$.
ceiling(x)	:=:	the smallest integer greater than or equal to x, $\lceil x \rceil$.
X mod Y	:=:	the remainder of integer division of x by y.
$\emptyset$	:=:	the null set.
H	:=:	a hash function, $H : K \rightarrow R$.
$K = (k_1, \ldots, k_N)$	:=:	the set of keys, whose cardinality is N.
N	:=:	the cardinality of the set of keys.
R	:=:	the range of hash addresses in the table, 0, . . . , maxhashaddr.
Z	:=:	the set of integers.
Z_m	:=:	the set of residues modulo m, i.e. the interval [0, m-1].

APPENDIX B

Program listing of Algorithm Q written in Pascal.

```
PROGRAM QUOT_REDUCT(INPUT,OUTPUT);
 (* this program can process  set I with 50 elements *)
 (* set up to use numbers to 9,999 in value  *)
Const
        num             =   9;  (*number of elements in
                                   data set stored in*)
                            (*   array w          *)
        Blank           =   ' ';
        blank2          =   '  '; (* 2 blanks *)
        four            =   4; (* used to format output *)
        five            =   5;
        six             =   6;
        seven           =   7;
Type    array_num       =   array[1..50] of integer;
        array_one_d     =   array[1..50] of integer;
        matrix_diff     =   array[1..50,1..50] of integer;
                            (* matrix to store array of
                              differences on w *)
        Integerset      =   set of 0..200;

Var     w               :   array_num; (* integer data set *)
        Min             :   array_one_d; (* min values from
                                            matrix_diff *)
        compute_min     :   array_one_d;(*computation on min *)
        diff            :   matrix_diff; (* table between
                                            differences of w *)
        I               :   integer;
        J               :   integer;
        Inc_on_w        :   integer;(*increment value on w(i)*)
        Row             :   integer;
        NumCOL          :   integer;
        spaces          :   string[50];
        filvar          :   text;     (* input file of data *)
        Destfile        :   text;    (* output to disk file *)

       nvalue          :    Integer;   (* Upper bound used to
                                          initialize delta *)
       Delta           :    Integerset;  (* initial value
                                            1 to Nvalue  *)
       trial           :    Integerset;

       Jset            :    Integerset; (* Residue set of Zn *)
       N               :    Integer;    (* Max value of the
                                           updated delta *)
       N_minus_one     :    Integer;


      testdiv          :    integer;
      J_set_empty      :    Boolean; (* Any values in set J ? *)
      t                :    integer; (* best t value in set_J *)
      s                :    integer;   (* value s  *)
      fv               :    integer; (* Format Value used
                                        in output *)
```

```
      v_on_line       :     integer;   (* number of values *)
                                       (* to be on one line *)
   (* ------------------------------------------------- *)
   (* This function provides the greatest integer       *)
   (* ------------------------------------------------- *)
Function Gintfun ( numer,denom :integer ) :integer;
Var  value    :     integer;
     test     :     real ;

begin (* function *)
value := 0;
test := numer/denom;
if test = trunc(test)
  then  value := trunc(test)
  else
   if test > trunc(test)
    then value := trunc(test) +1;

GIntfun := value;

end ;   (* function *)


(* ------------------------------------------------------- *)
(* A procedure which initializes certain arrays and matrix. *)
(* ------------------------------------------------------- *)
Procedure Initialize;
var I,J      : integer;
begin (*  proc init *)            (* initialize arrays  *)
   for I := 1 to 50
    do
      begin
       w[i] := 0;
       min[i] := 0;
       compute_min[i] := 0;
      end;

   for I := 1 to 50
    do begin
        for J := 1 to 50
         do begin
            diff[i,j] := -1; (* if values < 0 do not print *)
            end
       end
end (*    proc init  *);
```

```
(* ------------------------------------------------------ *)
(* Reads in the datafile                                  *)
(* ------------------------------------------------------ *)
Procedure readdata(var w: array_num ; num : integer);
var  I       : integer;
begin (* proc *)
  assign (filvar, 'h_2.dat');
  reset  (filvar);

  if not eof (filvar)
   then begin
        for I := 1 to num
         do
          begin
               readln(filvar,w[i]);
          end;
        end;
    for I := 1 to num        (* comment out later  *)
      do
        write(w[i]:fv);
        writeln;
end;   (* proc *)

Procedure readscdata(var w: array_num ; num : integer);
var  I       : integer;
begin (* proc *)
        for I := 1 to num
         do
          begin
               readln(w[i]);
          end;
    for I := 1 to num        (* comment out later  *)
      do
        write(w[i]:fv);
        writeln;
end;   (* proc *)
```

```
(* ------------------------------------------------------ *)
(* Writes the data set, opens the output file.            *)
(* ------------------------------------------------------ *)
procedure writedata(w:array_num;num :integer);
begin (* proc writedata *)
 assign (destfile, 'case2.out');

   Rewrite (destfile);
   writeln(destfile,'Algorithm Q ');
   write(destfile,'Given a set I, this program produces');
   writeln(destfile,' the following data');
   write(destfile,'to find the best quotient reduction');
   writeln(destfile,' phf for I.');
   write(destfile,'The set I = {');
   for I := 1 to num
     do
      write(destfile,w[i]:fv,',');
   writeln(destfile,'}.');
   writeln(destfile);
end    (* proc writedata*);
```

```
(* ------------------------------------------------------ *)
(* Step A.1 *)
(* A procedure which produces a difference table on the*)
(* elements of set I.       *)
(* ------------------------------------------------------ *)
Procedure Compute_Differences(var Diff:matrix_diff;
                              w:array_num;num:integer);
var row,I,J        : integer;
begin (* c-diff*)
Inc_on_w := 1 ;    (*  Inc_on_w =1 then we have 1st-diff *)
                   (*  Inc_on_w =2 then we have 2nd-diff
                        and so on *)

for Row := 1 to num - 1
 do
  begin
       I:=1;
       While I <= num - Inc_on_w  DO
      (* use num - Inc-on_w so that the value of I    *)
      (*  does not go beyond num *)
      (* Inc_on_w is the difference of j -i  *)
      (* of w[j]  and w[i] where j > i     *)
      (* NOTE: value of Row = value of Inc_on_w *)
        begin
              J:= I + Inc_on_w;
              Diff[row,i] := w[j] - w[i]; (* Diff[1,i] is *)
                                  (* the first differences *)
 {             write(diff[row,i]:4); (* test purposes *) }
              I := i + 1;
          end;
       Inc_on_w := Inc_on_w + 1;
{       writeln;                  }
   end;
end   (* c-diff*);
```

```
(* ------------------------------------------------------ *)
(* Prints the difference table *)
(* ------------------------------------------------------ *)
Procedure Print_Diff(w:array_num;diff:matrix_diff;
                                          num:integer);
var Row, I, numcol    :integer;
begin  (* proc p_d *)
writeln(destfile,'Step A.1 ');
writeln('The Difference Table for the W array above. ');
writeln(destfile,'The Difference Table for the W array : ');
writeln(destfile);
spaces := blank2;
for I := 1 to num
 do
 Begin (* for do *)
       write(w[i]:fv);
       write(destfile, w[i]:fv);
 end;  (* for do *)
 writeln;
 writeln;
 writeln(destfile);
 writeln(destfile);
for Row := 1 to num - 1
   do
     begin
       NumCOL := Row;
       write(spaces);
       write(destfile,spaces);
       I:= 1;
       While I <= num - NumCOL Do
              (*num - numCOL = num - Inc_on_w *)
         begin
              IF Diff[row,i] >=0
                 then
                    begin (* if then *)
                          write ( diff[row,i]:fv);
                          write(destfile, diff[row,i]:fv);
                    end;    (* if then *)
                I := i +1;
         end;
     writeln;
     writeln(destfile);
     spaces := spaces + blank2;
     end;
     writeln(destfile);
end;     (* proc P-D*)
```

```
(* ------------------------------------------------------ *)
(* Step A.2  Using the difference table from step A.1, find *)
(* the value of N(o) .*)
(* ------------------------------------------------------ *)
Procedure Upper_bound_N (var  nvalue: integer;diff:
                                matrix_diff;num:integer);

Var
         Compute_min    :     array_one_d;

     (* Produces Min array which holds the minimum values
        for each row  of the difference table for all    *)
     (* all w[j] and w[i] where j > i + 1.  *)
Procedure Min_Array;
begin (* MIn_Array  *)
for Row := 1 to num -1 (* each row for all w[j] and w[i]
                          where j>i+1  row = j -1 *)
DO
 begin  (* for do *)
        NumCOL := Row;
        Min[row] := Diff[row,1];
        I:=1;
        While I <= num - NumCOL
        DO
        begin  (* while do *)
               If Diff[row,i] >= 0
                THEN
                begin (* if then *)
                      If Diff[row,i] < min[row]
                       then   min[row] := Diff[row,i];
                end    (* if then *);
               I:=i +1;
        end     (* while do *)

 end    (* for do *)


end    (* MIn_Array *);
```

```
begin (* UP bound n*)
Min_Array;                          (* call procedure *)
writeln('from Upper Bound for N');
writeln(destfile,'Step A.2 ');
writeln(destfile,'The minimal Difference/Divided array is: ');
For Row := 2 to num -1    (* don't use 1st array element
                             where j=i+1*)
Do
 begin (* for do *)
        compute_min[row] :=trunc((min[row]-1)/(row -1 ));
                                          (* row = j-i *)
        writeln(compute_min[row] );
        writeln(destfile,compute_min[row]);
 end;    (* for do *)

 Nvalue:= compute_min[2];
 For Row:= 2 to num-1
 DO
  Begin (* for do *)
        If compute_min[row] < Nvalue
           Then Nvalue := compute_min[row];
  end   (* for do *) ;
 writeln(Nvalue,' is the minimum value which becomes the',
                                    ' upper bound for N ');
 writeln(destfile);
 writeln(destfile, Nvalue,
 ' is the minimum value which becomes the upper ',
                                    ' bound for N.');

end (* Up bound n  *) ;


(* ------------------------------------------------------ *)
(* Step B :  Initialize the set Delta to contain the   *)
(*           integers from 1 to N(o)  *)
(* ------------------------------------------------------ *)
Procedure Initialize_Delta(var  delta : integerset ;
                                Nvalue : integer);
var i   : integer;
begin (* Initialize_delta *)
Delta := [];
For I := 1 to Nvalue
DO
  Delta:=Delta + [i] ;

end   (* INit_delta *);
(* ------------------------------------------------------ *)
(* This procedure is used to print delta set *)
```

```
(* ------------------------------------------------------ *)
Procedure Print_Delta(anyset:integerset;poselem:integer);
Begin
writeln(destfile,'Delta = [ 1,',poselem,']');
end;


(* ------------------------------------------------------ *)
(* This procedure is used to print any set of integers    *)
(* ------------------------------------------------------ *)
Procedure Print_AnySet(anyset:integerset;poselem :integer);
var n     :integer;
counter   :integer;
begin (* print anyset *)
counter := 1;
 For n := 1 to poselem
 do
   if n in anyset
    then
       begin (* if then *)
              write(n:fv);
              write(destfile,n:fv);
              counter := counter + 1;
              if counter > v_on_line
              then
              begin (* if then *)
                    writeln(destfile);
                    counter :=1;
              end;     (* if then *)
       end;    (* if then *)
    writeln;
    writeln;
    writeln(destfile);
    writeln(destfile);
end; (* print anyset *)
```

```
(* ------------------------------------------------------  *)
(* step c *)
(* Find a better approximation for N which will be less*)
(* than or equal to N(o)  *)
(* ------------------------------------------------------  *)
Procedure Update_Delta(var delta : integerset;diff:
     matrix_diff;Nvalue :integer;   num : integer);

Var    colnum          : integer;
       LittleDelta1    : integer;
       LittleDelta2    : integer;
       d               : integer;
       dset            : integerset;
       n               : integer;
       counter         : integer;

   (* Step C.1 *)
   (* Initialize the set D to contain the integers from
       1 to d   *)
   (* where d = LittleDelta1 + LittleDelta2 + 1        *)



Procedure Init_Dset(var dset : integerset; d : integer);
 Var index : integer;

 Begin (* In Dset *)
 Dset := [];
  For Index := 1 to d
  Do
    Dset := Dset  + [index];
 End;  (* In Dset *)

   (* Step C.2 : Find the range of values for Theta  *)
   (* Step C.3 : For all values of theta compute an integer
   (* interval and take the union of the intervals with
       the set D  *)
```

```
Procedure Compute_Thetas_and_Dset(var dset :integerset;
              i, j, nvalue, d: integer; w: array_num);
Var  m              :  integer;
     BigM           :  integer;
     Theta1         :  integer;
     Theta2         :  integer;
     Theta          :  integer;
     a, b           :  integer;
     d_plus_one     :  integer;
Begin  (* c t dset *)
  m:=(w[j] +1) - w[i+1];
  BigM := w[j+1] - (w[i] +1);

  { smallest difference divided by biggest Nvalue }
  { biggest  difference divided by the smallest d value }
  Theta1 := GIntfun( m, Nvalue );
  D_plus_one := d + 1;
  Theta2 := GIntfun( m , d_plus_one);

  {update dset  depending on theta such that
   theta1 <= Theta <= Theta2 }
   writeln(destfile);
   writeln(destfile,'Step C.2 ');
   writeln ('Theta  a    b');
   writeln(destfile,'Theta    a      b ');
   (* Step C.3  follows:  *)
  For theta := Theta1 to Theta2
   DO
    begin  (* for do *)
     a:= GIntfun(m, theta );
     b:= Trunc (BigM/theta);
     DSet  := Dset + [a..b];

     writeln( theta:4,'  [',a:4,' ..',b:4,']');
     writeln(destfile, theta:4,'  [',a:4,' ..',b:4,']');
    end;     (* for do *)
 writeln(destfile);
end;   (* c t Dset *)
```

```
begin  (*  update delta   *)
{ Find the two  "first" differences less than Nvalue  }

For I := 1 to (num - 1) -1
DO
begin (* for do *)
      colnum := I + 1;
       While colnum <= num -1
       DO
       Begin (* while do *)
          If (Diff[1,i] + Diff[1,colnum]) <= Nvalue
          Then
           Begin (* if then *)
              LittleDelta1 := Diff[1,i];
              LittleDelta2 := Diff[1,Colnum];
              D :=  LittleDelta1 + LittleDelta2 -1;
              J :=  colnum;
              { call procedures which update  }
              Init_Dset(dset,d);
              writeln ('The value of d is ',d);
              writeln(destfile);
              writeln(destfile,'Step C.1 ');
              writeln(destfile,'With i = ',i,' and j = ',
                        colnum,' the value of d is ',d);
              writeln(destfile,'LittleDelta[',i,'] = '
                              ,LittleDelta1);
              writeln(destfile,'LittleDelta[',j,'] = '
                              ,LittleDelta2);
              writeln(destfile,'d = '
               ,littledelta1 + littledelta2 -1,
               '  Set D = [1..',d,']');
              Compute_Thetas_and_Dset(dset,i,j,nvalue,d,w );
              { update delta }

              writeln(destfile, 'Step C.3 ');
              writeln('The union of intervals put together',
                      ' for the set D : ');
              writeln(destfile,'The union of intervals put',
                      ' together for the set D : ');
              Print_Anyset(Dset,200);
              Delta := Delta * Dset;
             writeln(destfile,'Step C.4 ');
             writeln(' The set Delta updated',
                     ' (intersection with set D):          ');
             writeln(destfile,'The set Delta',
                    ' updated (intersection with set D): ');
             Print_Anyset(delta,Nvalue);

            writeln;
            writeln(destfile);
            writeln(destfile,
  '-------------------------------------------------');
```

```
          end    (* if then *);
       colnum := colnum + 1;
       end   (* while do *);
end (* for do *) ;

end    (*  update delta    *) ;

(* ------------------------------------------------------ *)
(*  step d  *)
(* The value of N is the maximum value of the set delta  *)
(* ------------------------------------------------------ *)
Procedure Max_ValueOF_Delta(var n: integer;delta:integerset;
                            nvalue:integer );
Var  index     :  integer;
begin  (* max value of delta *)
N := 1;
For index := 1 to nvalue
DO
 begin
 if index in delta
 then
     begin (* if then *)
     if index > n
     then n:= index;
     end ;(* if then *)
 end;

end;    (* max value of delta *)
(* ------------------------------------------------------ *)
(* step e *)
(* Procedure which initializes the set J to the
   integers [0,N-1]  *)
(*------------------------------------------------------ *)
Procedure Initialize_Jset(var jset: integerset;
                          n_minus_one: integer);
Var  index   : integer ;
begin (* init Jset *);
Jset := [];
For index := 0 to n_minus_one
Do
 Jset := Jset + [index];

end   (* init Jset *);
```

```
(* ------------------------------------------------------ *)
(* Procedure which prints the set J     *)
(* ------------------------------------------------------ *)

Procedure Print_set(anyset:integerset;elements :integer);
Var index :integer;
  counter :integer;
begin (* print_jset *)
counter := 1;
for  Index := 0 to elements
do
  if index in anyset
   then
     Begin (* if then *)
        write(index:fv);
        write(destfile,index:fv);
        counter := counter + 1;
        if counter > v_on_line
        then
        begin (* if then *)
              writeln(destfile);
              counter := 1;
        end;    (* if then *)
     end; (* if then *)
writeln;
writeln;
writeln(destfile);
writeln(destfile);

end;   (* print jset*)
```

```
(* ------------------------------------------------------*)
(* step f*)
(* This procedure reduces the number of elements in set J.*)
(* For the values of i, where LittleDelta[i] < N an integer
   interval *)
(* is calculated which is then intersected with the set J.*)
(* ------------------------------------------------------ *)
Procedure Determine_Jset(Var Dset:integerset; N :integer;
         diff:matrix_diff;w:array_num;num:integer);
Var t,i      : integer;
    funcset : integerset;
    funconw : integer;

  (* Makes sure that the residual value is positve    *)
Procedure Testmod(var residual :integer; N : integer) ;
begin (* testmod *)
If residual < 0
then residual := residual + N;

end ;   (* testmod *)

begin (* determine jset *)
  writeln(destfile,'Step F: ');
  For I := 1 to num - 1
  Do
   begin (* for do *)
      funcset := [];
      If diff[1,i] < N
       then
           Begin (* if then *)
           t := 0;
           while t < diff[1,i] DO
              begin  (* while do *)
              funconw := ( t - w[i + 1] ) mod N;
              Testmod(funconw,N);
                  (* test to see if residual is positive *)

              funcset := funcset + [funconw];
              t := t + 1;

              end;     (* while do *)
              writeln('When i = ',i,
              ' then Little Delta[i] = ', diff[1,i],
              ' which is < ',N);
              writeln('The interval is: ');

              writeln(destfile,'When i = ' ,i,
               ' then Little Delta[i] = ',
                diff[1,i],' which is < ',N);
              writeln(destfile,'The interval is: ');
              Print_set(funcset,N-1);
```

```
                Jset := Jset * funcset;

                writeln('The set J intersects with the',
                 ' previous interval giving: ');
                writeln(destfile,'The set J intersects',
                 ' with the previous interval giving: ');

                Print_set(Jset,N-1);
              end;     (* if then *)

   end;    (* for do  *)


end;    (* determine jset *)


(* ------------------------------------------------------ *)
(* Step G*)
(* This procedure checks to see if the set J is EMPTY.    *)
(* If it is,then program loops back to step d and drops*)
(* the value N from the set Delta.  If it is NOT EMPTY,
   then program continues forward. *)
(* ------------------------------------------------------ *)
Procedure Test_Jset(Var N,Nvalue :integer;
    var J_set_empty : boolean;Jset:integerset);
begin (* test jset *)
writeln(destfile,'Step G: Testing to see if set J is Empty ');
IF Jset = []
then
begin (* if then *)
   J_set_empty := true;
   N:= N - 1;
   Nvalue:=Nvalue -1;
   writeln(' Jset was empty so now N = ', N );
   writeln(destfile,' Jset was empty so now N = ', N );

end    (* if then *)
 else
     J_set_empty := false;

end     (* test jset *);
```

```
(* ------------------------------------------------------ *)
(* Step H *)
(* This procedure finds the element of J which minimizes *)
(* the table length.  *)
(* ------------------------------------------------------ *)
Procedure Minimize_J(var save_t : integer; J_set:integerset;
                     N:integer;w:array_num);
Label 10;
Var best_t : array [0..200] of integer;
    index  : integer;
    save   : integer;
    count  : integer;
    one_index :integer;


begin (* minimize J *)
writeln(destfile);
writeln(destfile,'Step H: ');
count := 0;

For Index :=  0 to 200          (* initialize *)
  (*  values in J_set will not normally be negative *)
Do
   Best_t[index] := -1;
writeln('The computed values on t in set J are :');
writeln(destfile,'The computed values on t in set J are :');
For Index := 0 to N-1
do
Begin (* for do *)
      If index in J_set
      then
      begin (* if then *)
          count := count + 1;
          Best_t[index] := ( w[1] + index ) mod N;
          writeln(' T := ',index,' and (w[1] + t) mod N := ',
                      best_t[index]);
          writeln(destfile,' T := ',index,
                ' and (w[1] + t) mod N := ',best_t[index]);
          one_index := index;
            (* save index if only one element in J *)

          Save := Best_t[index];
      end   (* if then *);

end  (* for do *);
(* Test for unusual situations  *)
(* first to see if set J is empty, which should never be
   the case in this procedure. *)
(* Second, see if there is only one element in the set J *)
If count = 0
   then writeln('The set J is empty. ERROR ***  ERROR   ');
```

```
If count = 1
   then
   begin (* if then *)
       save_t := one_index;
       GOTO 10;
   end    (* if then *);

For index := 1 to N-1
Do
begin (* for do *)
      If (index in J_set) and (Best_t[index] < save )
      then
      begin (* if then *)
           save := best_t[index];
           save_t := index;
      end    (* if then *);
end    (* for do *);


10 : writeln(' The minimal value in or the best t in',
              ' set J is ',save_t);
writeln(destfile,' The minimal value in or the',
                     ' best t in set J is ',save_t);

end    (* minimize J *);


(* ------------------------------------------------------ *)
(* Step I*)
(* This computes the value of s based on the value t.    *)
(* ------------------------------------------------------ *)
Procedure Compute_S(var s :integer;t:integer;N:integer;
                                 w:array_num);
begin (* compute s*)
writeln(destfile);
writeln(destfile,'Step I: ');
s := t - N * trunc( ( w[1] + t) / N );
writeln(' The value of s := ', s );
writeln(destfile,' The value of s := ', s );

end    (* compute s *);
```

```
(* ------------------------------------------------------ *)
(* Step J *)
(* This procedure computes the values of the hash table  *)
(* using the Perfect Hashing Function. *)
(* ------------------------------------------------------ *)
Procedure Use_Phf(w:array_num; s:integer; N :integer );
var hw : integer;
    index :integer;
begin (* use phf  *)
writeln ('The values computed using the phf are : ');
writeln (destfile,'The values computed using the phf are :');

For index := 1 to num
do
begin  (* for do *)
 hw :=  trunc ( ( w[index] + s )/ N );
 writeln( w[index ], ' , ',hw );
 writeln (destfile,w[index], ' , ', hw );
end    (* for do *);

end   (* use phf  *);
```

```
begin (* main program *)
      Initialize;
      fv:=five;
      v_on_line := 12;
      Readdata(w,num);
      Writedata(w,num);          (*  writing to disk file  *)
      Compute_Differences(diff,w,num);    (* on the elements
                                          w[i] in array W *)
      Print_Diff(w,diff,num);
      Upper_Bound_N (Nvalue,diff,num) ;
                                 (*  step A is finished *)
      {-------------------------------------------------------}
      Initialize_Delta(delta,Nvalue);              (*  step b *)
      Writeln ( 'Step B: This is Delta initialized ');
      writeln(destfile);
      writeln(destfile,'Step B: ');
      writeln(destfile,'Delta set is initialized as :  ');
      Print_Delta(delta,Nvalue);
      writeln;                                      (* step b *)
      writeln(destfile);
      {-------------------------------------------------------}

      Update_Delta(delta,diff,Nvalue,num);     (*  step c *)
      Writeln('This is the final Delta set ');
      writeln(destfile,'This is the final delta set ');
      Print_Anyset(delta,Nvalue);
      writeln;
      writeln(destfile);
      {-------------------------------------------------------}

      J_set_empty := false;
      repeat

      Max_ValueOF_Delta(n,delta,Nvalue);       (* step d*)
      writeln(destfile,'Step D: ');
      writeln('The maximum value in Delta is N = ',N);
      writeln(destfile,
               'The maximum value in delta is N = ',N);
      writeln(destfile);
      {-------------------------------------------------------}
      Initialize_Jset(Jset,N-1);     (* step e *)
      writeln(destfile,'Step E: ');
      writeln(' This is set J initialized');
      writeln(destfile,' This is set J initialized.');
      Print_set(jset,N-1);

      writeln;
      writeln(destfile);
      {-------------------------------------------------------}
      Determine_Jset(Jset,N,Diff,w,num);     (* step f *)

      Test_Jset(N,Nvalue,J_set_empty,Jset);
      until J_set_empty = false;              (* step g *)
```

```
      Minimize_J(t,Jset,N,w);                    (* step H *)

      Compute_S(s,t,N,w);
      writeln;
      writeln( 'The best quotient reduction phf is h(w) =',
               ' trunc(( w + s)/N ) ');
      writeln( 'h(w) = trunc(( w + ',s,')/ ',N,')     ');
      writeln(destfile);
      writeln(destfile, 'The best quotient reduction phf is',
                   ' h(w) = trunc(( w + s)/N ) ');
      writeln(destfile,
               'h(w) = trunc(( w + ',s,')/ ',N,')     ');

      writeln(destfile);
      writeln(destfile,'Step J: ');
      Use_Phf(w,s,N);

      writeln;

  close(destfile);
END.   (* main program *)
```

APPENDIX C

Listing of the program for the Quotient Reduction Algorithm with a cut.

```
PROGRAM QUOT_REDUCT(INPUT,OUTPUT);
 (* this program can process  set I with 50 elements *)
 (* set up to use numbers to 9,999 in value  *)
Label     99;
Const
       num              =   9;    (*number of elements in
                                      data set stored *)
                             (*    in array          *)
       Blank            =   ' ';
       blank2           =   '  '; (* 2 blanks *)
       four             =   4; (* used to format output  *)
       five             =   5;
       six              =   6;
       seven            =   7;
Type   array_num        =   array[1..50] of integer;
       array_one_d      =   array[1..50] of integer;
       matrix_diff      =   array[1..50,1..50] of integer;
                            (* matrix to store array of
                            differences on w *)
       Integerset       =   set of 0..200;

Var    w                :   array_num;   (* integer data set *)
       Min              :   array_one_d; (* min values from
                                              matrix_diff *)
       compute_min      :   array_one_d;(*computation on min *)
       diff             :   matrix_diff; (* table between
                                 differences of w *)
       I                :   integer;
       J                :   integer;
       Inc_on_w         :   integer;(*increment value on w(i)*)
       Row              :   integer;
       NumCOL           :   integer;
       spaces           :   string[50];
       filvar           :   text;     (* input file of data  *)
       Destfile         :   text;    (* output to disk file   *)

       nvalue           :   Integer;   (* Upper bound used
                                to initialize delta *)

       Delta            :   Integerset;   (* initial value  1 to
                                              Nvalue  *)
       trial            :   Integerset;
       Jset             :   Integerset;(* Residue set of Zn *)
       JLset            :   Integerset;(* Residue set of Zn *)
       JRset            :   Integerset;(* Residue set of Zn *)
       N                :   Integer;    (* Max value of the
                                 updated delta *)
       N_minus_one      :   Integer;


      testdiv           :   integer;
      JL_set_empty      :   Boolean; (* Any values in set J ?*)
      JR_set_empty      :   Boolean; (* Any values in set J ?*)
```

```
      t                 :    integer; (* best t value in set_J*)
      s                 :    integer;   (* value s  *)
      fv                :    integer; (* Format Value used in
                                          output *)
      v_on_line         :    integer;  (* number of values *)
                                       (* to be on one line *)
   (* variables used for cut algorithm *)
      ct                :    integer;(* determines cut point t*)
      shade_min         :    array_one_d; (* shaded min array
                                             values *)
      LDelta0           :    integer;
      P_hat             :    Integer;
      S_Prime           :    Integer;
      LDP               :    Integer; (* LittleDeltaPrime value
                                          for T *)

    (* arrays set up to hold data  for step J *)
      Rt, St, Lt, Nt    :    array_one_d;

      Z                 :    Integer;
   (* ------------------------------------------------- *)
   (* This function provides the greatest integer       *)
   (* ------------------------------------------------- *)
Function GIntfun ( numer, denom :integer ) :integer;
Var  value    :     integer;
     test     :     real ;

begin (* function *)
value := 0;
test := numer/denom;
if test = trunc(test)
  then  value := trunc(test)
  else
   if test > trunc(test)
   then value := trunc(test) +1;

GIntfun := value;

end ;  (* function *)
```

```
(* ------------------------------------------------------ *)
(* A procedure which initializes certain arrays and matrix. *)
(* ------------------------------------------------------ *)
Procedure Initialize;
var I,J     : integer;
begin (*  proc init *)           (* initialize arrays  *)
   for I := 1 to 50
    do
      begin
       w[i] := 0;
       min[i] := 0;
       compute_min[i] := 0;
       (* for cut algorithm  *)
       rt[i] := 0;
       st[i] :=0;
       Lt[i] :=0;
       Nt[i]:= 0;

      end;

   for I := 1 to 50
    do begin
        for J := 1 to 50
         do begin
            diff[i,j] := -1; (* if values < 0 do not print *)
            end
       end
end (*    proc init  *);

(* ------------------------------------------------------ *)
(* Reads in the datafile                                  *)
(* ------------------------------------------------------ *)
Procedure readdata(var w: array_num ; num : integer);
var  I      : integer;
begin (* proc *)
  assign (filvar, 'ccII.dat');
  reset  (filvar);

  if not eof (filvar)
   then begin
        for I := 1 to num
         do
          begin
               readln(filvar,w[i]);
          end;
        end;
    for I := 1 to num       (* comment out later  *)
      do
        write(w[i]:fv);
        writeln;
end;   (* proc *)
```

```
Procedure readscdata(var w: array_num ; num : integer);
var  I       : integer;
begin (* proc *)
        for I := 1 to num
         do
          begin
               readln(w[i]);
          end;
    for I := 1 to num        (* comment out later  *)
      do
        write(w[i]:fv);
        writeln;
end;   (* proc *)

(* ------------------------------------------------------ *)
(* Writes the data set, opens the output file.            *)
(* ------------------------------------------------------ *)
Procedure writedata(w:array_num;num :integer);
begin (* proc writedata *)
  assign (destfile, 'b:trash.out');

    Rewrite (destfile);
    writeln(destfile,'Algorithm Q ');
    writeln(destfile,
    'Given a set I, this program produces the following data');
    writeln(destfile,
    'to find the best quotient reduction phf for I.');
    write(destfile,'The set I = {');
    for I := 1 to num
      do
       write(destfile,w[i]:fv,',');
    writeln(destfile,'}.');
    writeln(destfile);
end    (* proc writedata*);
```

```
(* ------------------------------------------------------ *)
(* Step A.1 *)
(* A procedure which produces a difference table on the
   elements *)
(* of set I.       *)
(* ------------------------------------------------------ *)
Procedure Compute_Differences(var Diff:matrix_diff;
                                  w:array_num;num:integer);
var row,I,J        : integer;
begin (* c-diff*)
Inc_on_w := 1 ;
(*  Inc_on_w =1 then we have 1st-diff *)
(*  Inc_on_w =2 then we have 2nd-diff  and so on *)

for Row := 1 to num - 1
 do
  begin
       I:=1;
       While I <= num - Inc_on_w  DO
       (* use num- Inc-on_w so that the  *)
       (* value of I does not go beyond num *)
       (* Inc_on_w is the difference of j -i  *)
       (* of w[j]  and w[i] where j > i     *)
       (* NOTE: value of Row = value of Inc_on_w *)
        begin
               J:= I + Inc_on_w;
               Diff[row,i] := w[j] - w[i];
                (* Diff[1,i] is the *)
                (* first differences *)
 {              write(diff[row,i]:4); (* test purposes *) }
               I := i + 1;
          end;
       Inc_on_w := Inc_on_w + 1;
{       writeln;                  }
   end;
end   (* c-diff*);
```

```
(* ------------------------------------------------------ *)
(* Prints the difference table *)
(* ------------------------------------------------------ *)
Procedure Print_Diff(w:array_num;diff:matrix_diff;
                                        num:integer);
var Row,I,numcol    :integer;
begin  (* proc p_d *)
writeln(destfile,'Step A.1 ');
writeln('The Difference Table for the W array above. ');
writeln(destfile,'The Difference Table for the W array:');
writeln(destfile);
spaces := blank2;
for I := 1 to num
 do
 Begin (* for do *)
       write(w[i]:fv);
       write(destfile, w[i]:fv);
 end;  (* for do *)
 writeln;
 writeln;
 writeln(destfile);
 writeln(destfile);
for Row := 1 to num - 1
   do
     begin
       NumCOL := Row;
       write(spaces);
       write(destfile,spaces);
       I:= 1;
       While I <= num - NumCOL Do
        (*num - numCOL = num - Inc_on_w  *)
         begin
              IF Diff[row,i] >=0
                 then
                    begin (* if then *)
                          write ( diff[row,i]:fv);
                          write(destfile, diff[row,i]:fv);
                    end;   (* if then *)
                I := i +1;
         end;
     writeln;
     writeln(destfile);
     spaces := spaces + blank2;
     end;
     writeln(destfile);
end;    (* proc P-D*)
```

```
(* ------------------------------------------------------- *)
(* Step A.2  Using the difference table from step A.1,     *)
(* find the value of N(o) .*)
(* ------------------------------------------------------- *)
Procedure Upper_bound_N (var  nvalue: integer;
                         diff:matrix_diff;num:integer);

Var
         Compute_min   :     array_one_d;

(* Produces Min array which holds the minimum values for *)
(* each row of the difference table for all w[j] and w[i]
   where j > i + 1. *)
Procedure Min_Array;
begin (* MIn_Array  *)
for Row := 1 to num -1
   (* each row for all w[j] and w[i] where j>i+1    *)
         (* row = j - 1 *)
DO
 begin  (* for do *)
        NumCOL := Row;
        Min[row] := Diff[row,1];
        I:=1;
        While I <= num - NumCOL
        DO
        begin  (* while do *)
               If Diff[row,i] >= 0
                THEN
                begin (* if then *)
                      If Diff[row,i] < min[row]
                       then   min[row] := Diff[row,i];
                end    (* if then *);
               I:=i +1;
        end     (* while do *)

 end    (* for do *)


end    (* MIn_Array *);

begin (* UP bound n*)
Min_Array;                          (* call procedure *)
writeln('from Upper Bound for N');
writeln(destfile,'Step A.2 ');
writeln(destfile,
'The minimal Difference/Divided array is :  ');
For Row := 2 to num -1
   (* don't use 1st array element where j=i+1  *)
Do
 begin (* for do *)
        compute_min[row] :=trunc( (min[row] - 1)/(row -1 ));
          (* row = j-i *)
        writeln(compute_min[row] );
```

```
        writeln(destfile,compute_min[row]);
 end;    (* for do *)

 Nvalue:= compute_min[2];
 For Row:= 2 to num-1
 DO
  Begin (* for do *)
        If compute_min[row] < Nvalue
           Then Nvalue := compute_min[row];
  end   (* for do *) ;
 writeln(Nvalue,
 ' is the minimum value which becomes the upper bound for N');
 writeln(destfile);
 writeln(destfile, Nvalue,
 ' is the minimum value which becomes the upper bound for N.');

end (* Up bound n  *) ;


(* ------------------------------------------------------ *)
(* A procedure which initializes certain arrays and
   matrix.     *)
(*------------------------------------------------------- *)
Procedure Initialize_tarrays;
var I,J      : integer;
begin (*  proc init *)           (* initialize arrays  *)
   for I := 1 to 50
    do
      begin
       min[i] := 0;
       compute_min[i] := 0;
       shade_min[i]    :=0;
      end;

   for I := 1 to 50
    do begin
        for J := 1 to 50
         do begin
            diff[i,j] := -1;(* if values < 0 do not print *)
            end
       end
end (*    proc init  *);
```

```
(* ------------------------------------------------- *)
(* Step A.1 *)
(* A procedure which produces a difference table on the  *)
(* elements of set I *)
(* ------------------------------------------------- *)
Procedure Compute_t_Differences(var Diff:matrix_diff;
                                w:array_num;num:integer);
var row,I,J        : integer;
begin (* c-diff*)
Inc_on_w := 1 ;    (*  Inc_on_w =1 then we have 1st-diff *)
                   (*  Inc_on_w =2 then we have 2nd-diff
                       and so on *)

for Row := 1 to num - 1
 do
  begin
       I:=1;
       While I <= num - Inc_on_w  DO
         (* use num- Inc-on_w so that the  *)
        (* value of I does not go beyond num *)
       (* Inc_on_w is the difference of j -i  *)
        (* of w[j]  and w[i] where j > i     *)
        (* NOTE: value of Row = value of Inc_on_w *)
        begin
               J:= I + Inc_on_w;
               Diff[row,i] := w[j] - w[i];
                  (* Diff[1,i] is the *)
                  (* first differences *)
               If (i <= ct) and (ct < j)
               then
               begin
                Diff[row,i] := -1 * diff[row,i];
                end;

               I := i + 1;
          end;
       Inc_on_w := Inc_on_w + 1;

   end;
end   (* c-t_diff*);
```

```
(* ------------------------------------------------ *)
(* Prints the difference t table *)
(* ------------------------------------------------ *)
Procedure Print_t_Diff(w:array_num;diff:matrix_diff;
                                          num:integer);
var Row,I,numcol    :integer;
begin  (* proc p_d *)
writeln(destfile,'Step A.1: ');
writeln('The Difference Table for the W array above ');
writeln('with cut point value of ',ct);
writeln(destfile,'The Difference Table for the W array : ');
writeln(destfile,' The cut point value is ',ct );
writeln(destfile);
spaces := blank2;
for I := 1 to num
 do
 Begin (* for do *)
       write(w[i]:fv);
       write(destfile, w[i]:fv);
 end;  (* for do *)
 writeln;
 writeln;
 writeln(destfile);
 writeln(destfile);
for Row := 1 to num - 1
   do
     begin
       NumCOL := Row;
       write(spaces);
       write(destfile,spaces);
       I:= 1;
       While I <= num - NumCOL Do        (*num - numCOL =
                                           num - Inc_on_w  *)
         begin
              IF Diff[row,i] >=0
                 then
                    begin (* if then *)
                          write ( diff[row,i]:fv);
                          write(destfile, diff[row,i]:fv);
                    end   (* if then *)
                 else
                    begin  (* else  *)
                           write (' ___');
                           write (destfile, ' ___');
                    end;      (* else *)
                I := i +1;
         end;
     writeln;
     writeln(destfile);
     spaces := spaces + blank2;
     end;
     writeln(destfile);
end;     (* proc P-D*)
```

```
(* ------------------------------------------------------ *)
(* Step A.2  Using the difference table from step A.1,    *)
(* find the value of N(t) .*)
(* ------------------------------------------------------ *)
Procedure Upper_bound_Nt (var  nvalue: integer;
var shade_min : array_one_d; diff:matrix_diff;num:integer);

Type      array_boolean  =   array[1..50] of boolean;
Var

          Compute_min    :     array_one_d;

(* Produces Min array which holds the minimum values for *)
(* each row of the difference table for all w[j] and w[i]
   where j > i + 1.  *)

          Bmin           :    array_boolean;

Procedure Min_Array;
begin (* MIn_Array  *)
(* find value to initialize the min_array    *)
(* and set Bmin to True if there is a value,
   otherwise False *)
for Row := 1 to num -1
Do
 begin (* for do *)
  Bmin[row] := false;  (* there is no value in min_array *)
  numcol := row;
  for I := 1 to num -numcol
  do
  begin (* for do *)
   IF Diff[row,i] >= 0
   then
    begin (* if then *)
     Bmin[row]:= True;
     min[row] := diff[row,i];
    end   (* if then *)
   Else
     shade_min[row]:= diff[row,i];

  end;    (* for do *)

 end;   (* for do *)
(* find values for  the entire  min_array *)
for Row := 1 to num -1       (* each row for all w[j] and w[i]
                                where j>i+1    *)
DO                             (* row = j - i     *)
 begin  (* for do *)
        If Bmin[row] = true
        then
        begin (* if then *)
        NumCOL := Row;
     {   Min[row] := Diff[row,1]; }
```

```
        I:=1;
        While I <= num - NumCOL
        DO
        begin  (* while do *)
               If Diff[row,i] >= 0
                THEN
                 begin (* if then *)
                      If Diff[row,i] < min[row]
                       then   min[row] := Diff[row,i];
                 end    (* if then *)
                ELSE
                   If diff[row,i] > shade_min[row]
                   then  shade_min[row] := diff[row,i];

               I:=i +1;
        end;      (* while do *)
      end; (* if then *)
 end    (* for do *)


end    (* MIn_Array *);

begin (* UP bound n*)
Min_Array;                          (* call procedure *)
writeln('from Upper Bound for N');
writeln(destfile,'Step A.2: ');
writeln(destfile,'The minimal Difference/Divided array is:');
For Row := 2 to num -1
(* don't use 1st array element where j=i+1  *)
Do
 begin (* for do *)
    if Bmin[row] = true
    then
    begin (* if then *)
        compute_min[row] :=trunc((min[row] - 1) / (row -1));
                              (* row = j-i *)
        writeln(compute_min[row] );
        writeln(destfile,compute_min[row]);
     end; (* if then *)
 end;    (* for do *)

 Nvalue:= compute_min[2];
 For Row:= 2 to num-1
 DO
  Begin (* for do *)
   if Bmin[row] = true
   then
    begin (* if then *)
        If compute_min[row] < Nvalue
           Then Nvalue := compute_min[row];
    end; (* if then *)
  end    (* for do *) ;
 writeln(Nvalue,
 ' is the minimum value which becomes the upper bound for N ');
```

```
 writeln(destfile);
 writeln(destfile, Nvalue,
 ' is the minimum value which becomes the upper bound for N.');

end (* Up bound n  *) ;


(* ------------------------------------------------------ *)
(* Step B :  Initialize the set Delta to contain the   *)
(*           integers from 1 to N(o)  *)
(* ------------------------------------------------------ *)
Procedure Initialize_Delta(var  delta : integerset ;
                                Nvalue : integer);
var i    : integer;
begin (* Initialize_delta *)
Delta := [];
For I := 1 to Nvalue
DO
  Delta:=Delta + [i] ;

end   (* INit_delta *);
(* ------------------------------------------------------ *)
(* This procedure is used to print delta set *)
(* ------------------------------------------------------ *)
Procedure Print_Delta(anyset:integerset;poselem:integer);
Begin
writeln(destfile,'Delta = [ 1,',poselem,']');
end;
```

```
(* ------------------------------------------------------- *)
(* This procedure is used to print any set of integers     *)
(* ------------------------------------------------------- *)
Procedure Print_AnySet(anyset:integerset;poselem :integer);
var n     :integer;
counter   :integer;
begin (* print anyset *)
counter := 1;
 For n := 1 to poselem
 do
   if n in anyset
    then
       begin (* if then *)
              write(n:fv);
              write(destfile,n:fv);
              counter := counter + 1;
              if counter > v_on_line
              then
              begin (* if then *)
                    writeln(destfile);
                    counter :=1;
              end;     (* if then *)
       end;    (* if then *)
    writeln;
    writeln;
    writeln(destfile);
    writeln(destfile);
end; (* print anyset *)
```

```
(* ------------------------------------------------------ *)
(* step c *)
(* Find a better approximation for N which will be less
   than No *)
(* ------------------------------------------------------ *)
Procedure Update_Delta(var delta : integerset;diff:
          matrix_diff;Nvalue :integer; num : integer );
Var     colnum          : integer;
        LittleDelta1    : integer;
        LittleDelta2    : integer;
        d               : integer;
        dset            : integerset;
        n               : integer;
        counter         : integer;

    (* Step C.1 *)
    (* Initialize the set D to contain the integers from *)
    (* 1 to d where d = LittleDelta1 + LittleDelta2 + 1  *)
 Procedure Init_Dset(var dset : integerset; d : integer);
  Var index : integer;

  Begin (* In Dset *)
  Dset := [];
   For Index := 1 to d
   Do
     Dset := Dset  + [index];
  End;  (* In Dset *)

    (* Step C.2 : Find the range of values for Theta  *)
    (* Step C.3 : For all values of theta compute an
                  integer interval and take the union of
                  the intervals with the set D  *)
 Procedure Compute_Thetas_and_Dset(var dset :integerset;
           i,j,nvalue,d:integer; w: array_num);
 Var  m             :  integer;
      BigM          :  integer;
      Theta1        :  integer;
      Theta2        :  integer;
      Theta         :  integer;
      a,b           :  integer;
      d_plus_one    :  integer;
 Begin  (* c t dset *)
   m:=(w[j] +1) - w[i+1];
   BigM := w[j+1] - (w[i] +1);

   { smallest difference divided by biggest Nvalue }
   { biggest  difference divided by the smallest d value }
   Theta1 := GIntfun( m, Nvalue );
   D_plus_one := d + 1;
   Theta2 := GIntfun( m , d_plus_one);

   {updata dset  depending on theta such that
    theta1 <= Theta <= Theta2 }
```

```
    writeln(destfile);
    writeln(destfile,'Step C.2: ');
    writeln ('Theta  a    b');
    writeln(destfile,'Theta    a      b ');
    (* Step C.3  follows:  *)
   For theta := Theta1 to Theta2
    DO
     begin  (* for do *)
      a:= GIntfun(m,theta );
      b:= Trunc (BigM/theta);
      DSet  := Dset + [a..b];

      writeln( theta:4,'  [',a:4,' ..',b:4,']');
      writeln(destfile,theta:4,'  [',a:4,' ..',b:4,']');
     end;     (* for do *)
  writeln(destfile);
 end;    (* c t Dset *)

begin  (*  update delta   *)
{ Find the two  "first" differences less than Nvalue  }
{ and ignore differences such that (i <= t < j)       }
For I := 1 to (num - 1) -1
DO
begin (* for do *)
      colnum := I + 1;
       While colnum <= num -1
       DO
       Begin (* while do *)
          If (Diff[1,i] > 0) and (Diff[1,colnum] >0 ) and
            ( (Diff[1,i] + Diff[1,colnum]) <= Nvalue )
          Then
           If (i <= ct ) and (ct < colnum )
           then
            begin (* if then *)
            writeln( ' Ignore these differences',
                      ' since i <= t < j ');
            writeln(destfile,
            'Ignore these differences since '
                              ,i,'<= ',ct, ' < ',colnum);
            end   (* if then *)
          Else
           Begin (* if then *)
              LittleDelta1 := Diff[1,i];
              LittleDelta2 := Diff[1,Colnum];
              D :=  LittleDelta1 + LittleDelta2 -1;
              J :=  colnum;
              { call procedures which update  }
              Init_Dset(dset,d);
              writeln ('The value of d is ',d);
              writeln(destfile);
              writeln(destfile,'Step C.1 ');
              writeln(destfile,'With i = ',i,' and j = ',
              colnum,' the value of d is ',d);
```

```
                writeln(destfile,
                'LittleDelta[',i,'] = ',LittleDelta1);
                writeln(destfile,
                'LittleDelta[',j,'] = ',LittleDelta2);
                writeln(destfile,
                'd = ',littledelta1 + littledelta2 -1,
                '  Set D = [1..',d,']');
                Compute_Thetas_and_Dset(dset,i,j,nvalue,d,w );
                { update delta }

                writeln(destfile, 'Step C.3: ');
                writeln('The union of intervals put together ',
                          'for the set D : ');
                writeln(destfile,
                'The union of intervals put together for the',
                ' set D : ');
                Print_Anyset(Dset,200);
                Delta := Delta * Dset;
               writeln(destfile,'Step C.4: ');
               writeln(' The set Delta updated',
               ' (intersection with set D):          ');
               writeln(destfile,
               'The set Delta updated',
               ' (intersection with set D): ');
               Print_Anyset(delta,Nvalue);

              writeln;
              writeln(destfile);
              writeln(destfile,
 '-------------------------------------------------------');
            end    (* if then *);
        colnum := colnum + 1;
        end   (* while do *);
end (* for do *) ;

end    (*  update delta    *) ;
```

```
(* ------------------------------------------------------ *)
(*  step d  *)
(* The value of N is the maximum value of the set delta  *)
(* ------------------------------------------------------ *)
Procedure Max_ValueOF_Delta(var n: integer;
                        delta:integerset; nvalue:integer );
Var  index     :  integer;
     testvalue :  integer;

begin  (* max value of delta *)
Testvalue := 1;
For index := 1 to nvalue
DO
 begin
 if index in delta
 then
     begin (* if then *)
     if index > testvalue
     then testvalue:= index;
     end ;(* if then *)
 end;
  N:= testvalue;
end;     (* max value of delta *)
(* ------------------------------------------------------ *)
(* step e *)
(* Procedure which initializes the set J to the integers
   [0,N-1]  *)
(* ------------------------------------------------------ *)
Procedure Initialize_Jset(var jset: integerset;
                          n_minus_one: integer);
Var  index   : integer ;
begin (* init Jset *);
Jset := [];
For index := 0 to n_minus_one
Do
 Jset := Jset + [index];

end   (* init Jset *);
```

```
(* ------------------------------------------------------ *)
(* Procedure which prints the set J     *)
(*------------------------------------------------------- *)

Procedure Print_set(anyset:integerset;elements :integer);
Var index :integer;
  counter :integer;
begin (* print_jset *)
counter := 1;
for  Index := 0 to elements
do
  if index in anyset
   then
     Begin (* if then *)
        write(index:fv);
        write(destfile,index:fv);
        counter := counter + 1;
        if counter > v_on_line
        then
        begin (* if then *)
              writeln(destfile);
              counter := 1;
        end;    (* if then *)
     end; (* if then *)
writeln;
writeln;
writeln(destfile);
writeln(destfile);

end;  (* print jset*)
```

```
(* ------------------------------------------------------ *)
(* step f*)
(* This procedure reduces the number of elements in set J.*)
(* For the values of i, where LittleDelta[i] < N an integer
   interval is calculated which is then intersected with
   the set J.    *)
(* ------------------------------------------------------ *)
Procedure Determine_JLset(Var JLset:integerset; N :integer;
        diff:matrix_diff;w:array_num; num :integer;
        ct:integer);
Var v,i      : integer;
    funcset : integerset;
    funconw : integer;

  (* Makes sure that the residual value is positve    *)
Procedure Testmod(var residual :integer; N : integer) ;
begin (* testmod *)
If residual < 0
then residual := residual + N;

end ;  (* testmod *)

begin (* determine jset *)
  writeln(destfile,'Step F: ');
  For I := 1 to ct - 1
  Do
   begin (* for do *)
      funcset := [];
      If ( diff[1,i] >0) and (diff[1,i] < N)
       then
           Begin (* if then *)
           v := 0;
           while v < diff[1,i] DO
              begin  (* while do *)
              funconw := ( v - w[i + 1] ) mod N;
              Testmod(funconw,N);
              (* test to see if residual is positive *)
              funcset := funcset + [funconw];
              v := v + 1;

              end;     (* while do *)
              writeln('When i = ',i,
              ' then Little Delta[i] = ',diff[1,i],
              ' which is < ',N);
              writeln('The interval is: ');

              writeln(destfile,'When i = ' ,i,
              ' then Little Delta[i] = ',diff[1,i],
              ' which is < ',N);
              writeln(destfile,'The interval is: ');
              Print_set(funcset,N-1);

              JLset := JLset * funcset;
```

```
                writeln('The set JL intersects with the',
                ' previous interval giving: ');
                writeln(destfile,'The set JL intersects',
                ' with the previous interval giving: ');

                Print_set(JLset,N-1);
             end;     (* if then *)

    end;    (* for do  *)


end;    (* determine jlset *)
(* ------------------------------------------------------ *)


Procedure Determine_JRset(Var JRset:integerset; N :integer;
 diff:matrix_diff;w:array_num; num :integer; ct:integer);
Var v,i      : integer;
    funcset : integerset;
    funconw : integer;

  (* Makes sure that the residual value is positve    *)
Procedure Testmod(var residual :integer; N : integer) ;
begin (* testmod *)
If residual < 0
then residual := residual + N;

end ;  (* testmod *)

begin (* determine jrset *)
  writeln(destfile,'Step F: ');
  For I := ct + 1 to num - 1
  Do
   begin (* for do *)
      funcset := [];
      If (diff[1,i] > 0) and (diff[1,i] < N)
       then
           Begin (* if then *)
           v := 0;
           while v < diff[1,i] DO
              begin  (* while do *)
              funconw := ( v - w[i + 1] ) mod N;
              Testmod(funconw,N);
              (* test to see if residual is positive *)

              funcset := funcset + [funconw];
              v := v + 1;

              end;     (* while do *)
              writeln('When i = ',i,
              ' then Little Delta[i] = ',diff[1,i],
              ' which is < ',N);
              writeln('The interval is: ');
```

```
                writeln(destfile,'When i = ' ,i,
                ' then Little Delta[i] = ',
                  diff[1,i],' which is < ',N);
                writeln(destfile,'The interval is: ');
                Print_set(funcset,N-1);

                JRset := JRset * funcset;

                writeln('The set JR intersects with',
                ' the previous interval giving: ');
                writeln(destfile,'The set JR intersects',
                ' with the previous',' interval giving: ');

                Print_set(JRset,N-1);
             end;     (* if then *)

   end;    (* for do  *)


end;    (* determine jRset *)

(* ------------------------------------------------- *)
(* Step G*)
(* This procedure checks to see if the set JL is EMPTY.
   If it is, then program loops back to step d and drops
   the value N from the set Delta.  If it is NOT EMPTY,
   then program continues forward. *)
(* ------------------------------------------------- *)
Procedure Test_JLset(Var N:integer; Var Nvalue :integer;
        var JL_set_empty : boolean; JLset: integerset);
begin (* test jset *)
writeln(destfile,
'Step G: Testing to see if set JL is Empty ');
IF JLset = []
then
begin (* if then *)
   JL_set_empty := true;
  { N:= N - 1;          }
 {  Nvalue:=Nvalue -1;}
   writeln(' JLset was empty so now N = ', N );
   writeln(destfile,' JLset was empty so now N = ', N );
   writeln(destfile,
   ' JLset was emply so now Nvalue = ', Nvalue );

end   (* if then *)
 else
     JL_set_empty := false;

end    (* test jset *);
```

```
(* ------------------------------------------------- *)
(* Step G*)
(* This procedure checks to see if the set JR is EMPTY.
   If it is, then program loops back to step d and drops
   the value N from the set Delta.  If it is NOT EMPTY,
   then program continues forward. *)
(* ------------------------------------------------- *)
Procedure Test_JRset(Var N: integer;Var Nvalue :integer;
         var JR_set_empty : boolean;JRset: integerset);
begin (* test jset *)
writeln;
writeln(destfile);
writeln(destfile,
'Step G: Testing to see if set JR is Empty ');
Writeln('At the start of Test_JRset  N= ',N,
' and Nvalue = ',Nvalue);
Writeln(destfile,'At the start of Test_JRset  N= ',N,
' and Nvalue = ',Nvalue);
IF JRset = []
then
begin (* if then *)
   JR_set_empty := true;
  { N:= N - 1;           }
  { Nvalue:=Nvalue -1;   }
   writeln(' JRset was empty so now N = ', N );
   writeln(destfile,' JRset was empty so now N = ', N );

end   (* if then *)
 else
     JR_set_empty := false;

end    (* test jset *);
```

```
(* ------------------------------------------------------ *)
(* Step H *)
(* Determine LittleDelta(o) *)
(* ------------------------------------------------------ *)
Procedure Shade_Differences(var LDelta0 :integer; ct:integer;
      w : array_num; shade_min:array_one_d;n,num :integer);
Var LittleDeltaT  : integer;
    Compute_s_max : array_one_d;
Begin (* Shade_diff *)
writeln(destfile,'Step H: ');
writeln('The upper bound for LittleDelta T ');
writeln(destfile,'The upper bound for LittleDelta T ');
writeln(destfile,' where t = ',ct);
LittleDeltaT := w[ct +1] - w[ct];

For row := 1 to num -1
Do
 begin (* for do *)
  Compute_s_max[row] :=(row -1) * N +1+shade_min[row]+
                        littleDeltaT;
  writeln( compute_s_max[row]);
  writeln(destfile, compute_s_max[row]);
 end;    (* for do *)
 writeln;
 writeln(destfile);
LDelta0 := compute_s_max[1];

For row := 2 to num-1
DO
 Begin (* for do *)
  If compute_s_max[row] > LDelta0
  Then LDelta0 := compute_s_max[row];

 End;    (* for do *)
writeln(LDelta0 ,' is the value for  LittleDelta(0) ');
writeln(destfile,LDelta0 ,
' is the value for  LittleDelta(0) ');
END;    (* Shade_diff *)
```

```
(* ------------------------------------------------- *)
(* Step I.1 and I.2 *)
(* Determine the values of P_hat and s' *)
(* ------------------------------------------------- *)
Procedure P_hat_value(Var P_hat,S_Prime :integer; N:integer;
                  w:array_num; JLset:integerset;ct:integer );
Var     P       : integer;
        Rem     : integer;
        Pset    : integerset;
        Index   : integer;
  (* Makes sure that the residual value is positve   *)
Procedure Testmod(var residual :integer; N : integer) ;
begin (* testmod *)
If residual < 0
then residual := residual + N;

end ;  (* testmod *)

Begin (* p_hat *)
writeln('Step I.1 and I.2  ');
writeln(destfile,'Step I.1 and I.2:  ');
Pset := [];

For P := 1 to N
Do
Begin (* for do *)
 Rem := (- w[ct] - P) mod N;
 Testmod(rem,N);
 If Rem in JLset
  Then   Pset := pset + [p];

End;   (* for do *)
If Pset = []
then Writeln('Pset is EMPTY ');

For  Index := N Downto 1
Do
Begin (* for do *)
 If index in Pset
 Then P_hat := Index;
 (* This will automatically contain the *)
 (* smallest value of Pset *)

End;   (* for do *)
Writeln(' P^ = ', P_hat );
writeln(destfile, ' P^ = ',P_hat );

S_prime :=  (- w[ct] - P_hat ) mod N;
Testmod(S_Prime,N);
writeln('S` =  ',S_Prime );
writeln(destfile, 'S` = ', S_Prime );

End;   (* p_hat *)
```

```
(* ------------------------------------------------------ *)
(* Step I.3 *)
(* Find LittleDelta(t)' *)
(* ------------------------------------------------------  *)
Procedure L_Delta_P_value(Var LDP :integer; N:integer;
          P_hat:integer; JRset:integerset;w:array_num;
          ct:integer;LDelta0:integer);
Var       Congruent : boolean;
          Index     : integer;
          Jpp       : integer;
          L_Delta_P : integer;
          Rem       : integer;
          Con_Rem   : integer;

          SaveJ     : integer;
          SaveLDP   : integer;
          SaveRem   : integer;

Procedure Testmod(var residual :integer; N : integer) ;
begin (* testmod *)
If residual < 0
then residual := residual + N;

end ;  (* testmod *)

Begin (* LDP value *)
Writeln(destfile,'Step I.3: ');
Congruent := false;

L_Delta_P := LDelta0;

Repeat
For Index := 0 to N -1    (* N or N-1 ??? changed 1 to 0*)
Do
Begin (* for do *)
If index in JRset
 then
  begin (* if then *)
    Jpp := index;
    Rem := (w [ct +1] + Jpp + P_hat ) mod N;
    Testmod (Rem, N) ;

    Con_Rem := L_Delta_P mod N;
    Testmod (Con_Rem, N);

      If Con_Rem = Rem
      then
       begin (*   If then *)
          Congruent := True;
          SaveJ     := Jpp;
          SaveLDP   := L_Delta_P;
          SaveRem   := Rem;
       end;  (*   if  then *)
```

```
  end ; (* if then *)
end   ;(* for do *)
L_Delta_P := L_Delta_P + 1;

Until Congruent = True;
LDP := SaveLDP;

writeln('Little_Delta_Prime(',ct,') = ', LDP);

writeln(destfile,'Little_Delta_Prime(',ct,') = ', LDP);
End ; (* LDP value *)

(* ------------------------------------------------------ *)
(* Step J *)
(* Find the values of r[t] and s[t] and also L[t] *)
(* ------------------------------------------------------ *)
Procedure Determine_Rand_Sand_Len(var Rt,St,Lt,Nt :array_one_d;
   ct, LDP,S_Prime,N,num :integer; diff:matrix_diff;
   w:array_num);

Var floor : integer;

Begin (* Det r s Length *)
 writeln(' Step J ');
 writeln(destfile,' Step J: ');

 writeln ('L_Delta_Prime = ',LDP);
 writeln (' Ldelta (',ct,') = ',diff[1,ct]);
 Rt[ct] := LDP + diff[1,ct];  (* diff[1,ct is negative *)

 floor:= trunc ( (w[1] + S_Prime)/N );
 St[ct] := S_Prime - n * floor;

 Lt[ct] := trunc ((w[num] + St[ct] + Rt[ct] )/N)  - 1;

 Nt[ct] := N;

writeln('When t = ',ct,' and length of table L = ',Lt[ct]);
writeln( 'R[',ct,'] = ', Rt[ct]);
writeln( 'S[',ct,'] = ', st[ct]);
writeln( 'Nt[',ct,'] = ', Nt[ct]);

writeln(destfile,
'When t = ',ct,' and length of table L = ',Lt[ct]);
writeln(destfile, 'R[',ct,'] = ', Rt[ct]);
writeln(destfile, 'S[',ct,'] = ', st[ct]);
writeln(destfile, 'Nt[',ct,'] = ', Nt[ct]);

End  ; (* Det r s Length *)
```

```
(* ------------------------------------------------------ *)
(* Step L *)
(* Procedure to find the values which give the table of  *)
(* mininum length  *)
(* ------------------------------------------------------ *)
Procedure Min_Table_Len(var z :integer; Lt:array_one_d;
                                          num:integer);
Var   TestLen  : Integer;
      SaveZ    : Integer;
      index    : integer;
Begin  (* Min Table Len *)
writeln (' Step L ');
writeln (destfile, 'Step L: ');
 TestLen := Lt[1];
 SaveZ   := 1;

 For index := 1 to num-1
 Do
 Begin (* for do *)
  z := index;
  If Lt[z] < TestLen
  Then
   Begin (* if then *)
     TestLen := Lt[z];
     SaveZ   := Z;
   End  ;(* if then *)

 End;    (* for do *)
z := savez;
writeln('The minimal lenth of table is ',Lt[z],
' where z = ',z);
writeln(destfile,'The minimal lenth of table is ',Lt[z],
' where z = ',z);
writeln;
writeln(destfile);

End ;  (* Min table Len *)
```

```
(* ------------------------------------------------------ *)
(* Step M *)
(* This procedure computes the values of the hash table
   using the Perfect Hashing Function. *)
(* ------------------------------------------------------ *)
Procedure Use_Phf(w:array_num; s:integer; r:integer;
                  N :integer;z :integer );
var hw : integer;
    index :integer;
begin (* use phf  *)
writeln ('The values computed using the phf are : ');
writeln (destfile,'The values computed using the phf are:');

For index := 1 to z
do
begin  (* for do *)
 hw :=  trunc ( ( w[index] + s )/ N );
 writeln( w[index ], ' , ',hw );
 writeln (destfile,w[index], ' , ', hw );
end    (* for do *);

For index := z + 1 to num
do
begin  (* for do *)
 hw :=  trunc ( ( w[index] + s + r)/ N );
 writeln( w[index ], ' , ',hw );
 writeln (destfile,w[index], ' , ', hw );
end    (* for do *);

end   (* use phf  *);
```

```
begin (* main program *)
      Initialize;
      fv:=five;
      v_on_line := 12;
      Readdata(w,num);
      Writedata(w,num);
      (*  writing to disk file  *)
      Compute_Differences(diff,w,num);
      (* on the elements w[i] in array W *)

      Print_Diff(w,diff,num);

      Upper_Bound_N (Nvalue,diff,num) ;

   ct := 1;     (* initial cut value *)
   For ct := 1 to num-1
   Do
   Begin (* for do loop on ct *)

      Initialize_tarrays;
      Compute_t_Differences(diff,w,num);
      Print_t_diff(w,diff,num);
      Upper_Bound_Nt (Nvalue,shade_min,diff,num);
                        (*  step A is finished *)
      {-------------------------------------------------}
      Initialize_Delta(delta,Nvalue);          (*  step b *)
      Writeln ( 'Step B: This is Delta initialized ');
      writeln(destfile);
      writeln(destfile,'Step B: ');
      writeln(destfile,'Delta set is initialized as :  ');
      Print_Delta(delta,Nvalue);
      writeln;                                 (* step b *)
      writeln(destfile);
      {-------------------------------------------------}

      Update_Delta(delta,diff,Nvalue,num);     (*  step c *)
      Writeln('This is the final Delta set ');
      writeln(destfile,'This is the final delta set ');
      Print_Anyset(delta,Nvalue);
      writeln;
      writeln(destfile);
      {-------------------------------------------------}

      JL_set_empty := false;
      JR_set_empty := false;
      repeat

      Max_ValueOF_Delta(n,delta,Nvalue);       (* step d*)
      writeln(destfile,'Step D: ');
      writeln('The maximum value in Delta is N = ',N);
      writeln(destfile,
             'The maximum value in delta is N = ',N);
      writeln(destfile);
```

```
    {-------------------------------------------------}

    Initialize_Jset(Jset,N-1);     (* step e *)
    writeln(destfile,'Step E: ');
    writeln(' This is set J initialized');
    writeln(destfile,' This is set J initialized.');
    Print_set(jset,N-1);

    writeln;
    writeln(destfile);
    JLset := [];
    JRset := [];

    JLset := Jset;   (* set the left and right jset *)
    JRset := Jset;   (* equal to the Jset  *)
    {-------------------------------------------------}
    Determine_JLset(JLset,N,Diff,w,num,ct);(* step f *)
    Determine_JRset(JRset,N,Diff,w,num,ct);(* step f *)

    Test_JLset(N,Nvalue,JL_set_empty,JLset);
    Test_JRset(N,Nvalue,JR_set_empty,JRset);
    If (JL_set_empty = true) or (JR_set_empty = true )
    then  Delta := Delta - [N];
    (* step g *)
    until (JL_set_empty = false) and (JR_set_empty = false);
    (* step H *)
    Shade_differences(LDelta0,ct,w,shade_min,N,num);
    (* step I1 and I2 *)
    P_hat_value(P_hat,S_Prime,N,w,JLset,ct );
    (* step I3 *)
    L_Delta_P_value(LDP,N,P_Hat,JRset,w,ct,LDelta0);

    (* Step J *)
    Determine_Rand_Sand_Len(Rt,St,Lt,Nt,Ct,LDP,S_Prime,
                            N,Num,Diff,w);

end;  (* for do loop on ct *)      (* loop is Step K *)
   close(destfile);
   assign (destfile, 'b:cciii2.out');

   Rewrite (destfile);
     Min_Table_Len(z,Lt,num );     (* Step L *)
     writeln;
     writeln(
     'The values corresponding to the value of z = ', z,
     ' are : ');
     writeln(destfile,
     'The values corresponding to the value of z = ',
     z, ' are : ');
     writeln( 'S = ', St[z]  ,  'R = ', Rt[z] );
     writeln( 'The cut point element w  is ',w[z] );
     writeln( 'The value of N  is          ',Nt[z]);
```

```
      writeln(destfile, 'S = ', St[z]  ,  'R = ', Rt[z] );
      writeln(destfile, 'The cut point element w is ',w[z]);

      writeln(destfile, 'The value of N  is       Nt[z]');

      writeln;
      writeln( 'The perfect hashing function',
      ' is h(w) = trunc(( w + s)/N ) ');
      writeln( 'h(w) = trunc(( w + ',st[z],')/ ',Nt[z],')    ',
      '         for i <= ',z);
      writeln( 'h(w) = trunc(( w + ',st[z],'+ ',rt[z],')/ ',
      Nt[z],')     for i > ',z);
      writeln(destfile);
      writeln(destfile,
      'The perfect hashing funtion ',
      'is h(w) = trunc(( w + s)/N ) ');
      writeln(destfile,
      'h(w) = trunc(( w + ',st[z],')/ ',Nt[z],')       ',
      '     for i <= ',z);
      writeln(destfile, 'h(w) = trunc(( w + ',st[z],'+ ',
      rt[z],')/ ',Nt[z],')     for i > ',z);

      writeln(destfile);
      writeln(destfile,'Step M: ');
      Use_Phf(w,st[z],Rt[z],Nt[z],z);
      writeln;
  close(destfile);
END.   (* main program *)
```