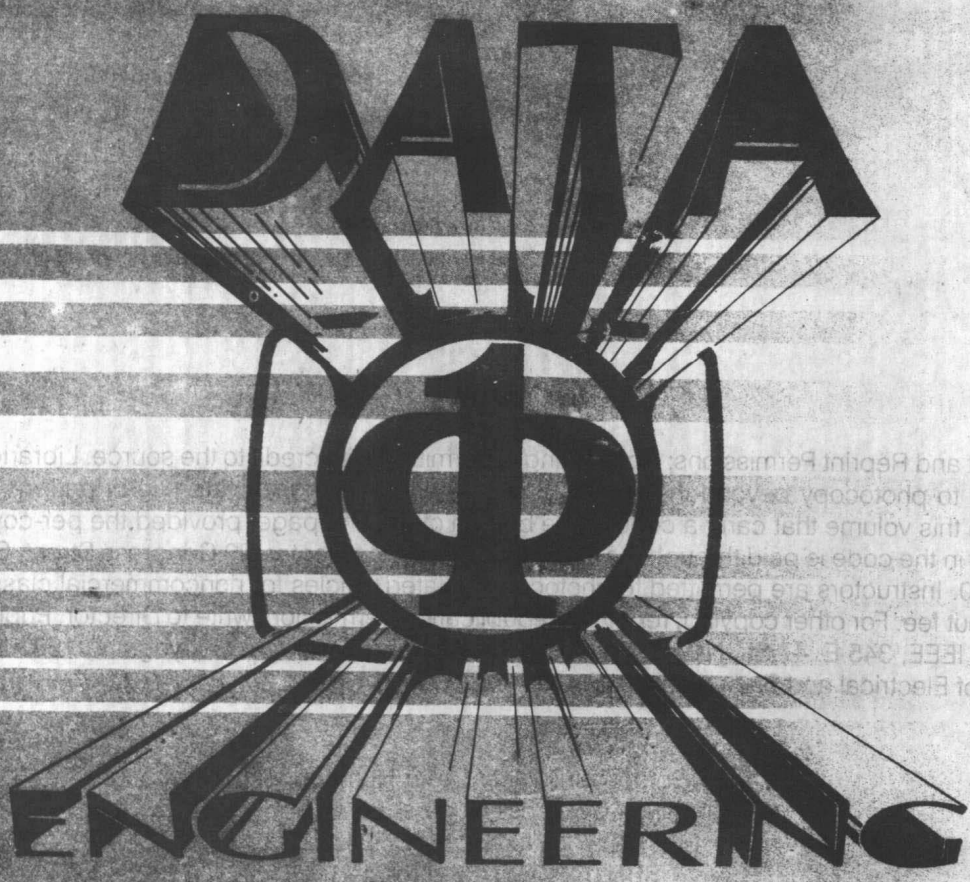
Third International Conference on

DATA ENGINEERING

1987

Proceedings

Third International Conference on DATA ENGINEERING



February 3–5, 1987
Pacifica Hotel
Los Angeles, California, USA

IEEE Computer Society Order Number 762
Library of Congress Number 86-46344
IEEE Catalog Number 87CH2407-5
ISBN 0-8186-0762-9

THE COMPUTER SOCIETY OF THE IEEE

IEEE COMPUTER SOCIETY PRESS

The papers appearing in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and are published as presented and without change, in the interests of timely dissemination. Their inclusion in this publication does not necessarily constitute endorsement by the editors, IEEE Computer Society Press, or the Institute of Electrical and Electronics Engineers, Inc.

Published by IEEE Computer Society Press
1730 Massachusetts Avenue, N.W.
Washington, D.C. 20036-1903

COVER DESIGNED BY JACK I. BALLESTERO

IEEE Computer Society Order Number 762
Library of Congress Number 86-46344
IEEE Catalog Number 87CH2407-5
ISBN 0-8186-0762-9 (paper)
ISBN 0-8186-4762-0 (microfiche)
ISBN 0-8186-8762-2 (case)

Order from: IEEE Computer Society
Post Office Box 80452
Worldway Postal Center
Los Angeles, CA 90080

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

IEEE Computer Society
Avenue de la Tanche, 2
B-1160 Brussels,
Belgium

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

ACKNOWLEDGMENTS

This third conference is made possible by the contributions from a large number of individuals. Without their consistent and timely support, none of this would have been possible.

My special thanks go to the Program Co-Chairs and members of the Program Committee, who actively reviewed a large number of the submissions, and the many individual reviewers solicited by the Program Committee.

My sincere thanks go to Gio Wiederhold, the General Chairperson, P. Bruce Berra, the Honorary Chairperson, and C.V. Ramamoorthy, the Steering Committee Chairperson, who have provided advice, guidance, and countless invaluable insights to me throughout the preparation of this conference.

On behalf of the Program Committee, I express my sincere gratitude to the keynote speakers, authors, panelists, session organizers, and all participants. They have worked very hard to keep a high standard of quality for this conference.

I am grateful to Katherine M. Baumgartner, who has devoted a lot of her research time to help me to run this conference. She has single-handedly designed a data management system that allowed rapid decision-making in the selection of papers and integrated the reviews of all the manuscripts.

Thanks also go to Katy Lindquist, who helped us in contacting members of the program committee and in getting papers out to the reviewers.

Finally, we are grateful to Wendy Chin, Denise Felix, and the staff at the Computer Society's Washington D.C. office for their usual expert assistance in the production of the proceedings and the handling of the program announcements.

The following list contains the additional reviewers solicited by the Program Committee.

R. Ahad	L. Delcambre	Y. Hwang	D. Ourston	M. Singhal
J. Anand	S. Demurjian	M. Kao	T. Palachek	S. Son
D. Baer	D. Dias	T. Kikuno	K. Ramamritham	S. Spaccapietra
R. Bagley	J. Diaz	H. Kim	R. Krishnamurti	S. Su
J. Banerjee	B. Didier	V. Kumar	J. Reuter	Y. Sung
D. Barbara	J. Diederich	L. Leu	P. Rich	T. Wu
K. Baumgartner	T. Donahue	D. Long	D. Ridjanovic	R. Varadarajan
M. Berardo	M. Evangelist	W. Luk	J. Riedl	K. Vidyasankar
B. Bimson	A. Farrag	K. Machovec	J. Robinson	S. Wahl
M. Bu-Hulaiga	J. Ferrans	D. Madison	Z. Ruan	J. Wald
L. Buris	G. Galatianos	B. Martin	V. Saletore	K. Walt
S. Cammarata	A. Goodman	S. Mazumdar	B. Sayrs	T. Welch
C. Chang	M. Gooley	J. Milton	B. Seeger	C. Yang
E. Chen	G. Hahn	K. Moriya	D. Shasha	R. Yanney
P. Chen	D. Hartman	J. Naughton	S. Shen	R. Yavatkar
A. Chou	W. Hasan	W. Niblack	P. Sheu	P. Yu
P. Choy	L. Hollaar	T. Noma	C. Shubet	Y. Zalustein
J. Combs	Y. Hong	J. Orenstein	S. Shyu	

Benjamin W. Wah
Program Chairperson

THIRD DATA ENGINEERING CONFERENCE COMMITTEES

Steering Committee Chairperson:
C. V. Ramamoorthy
University of California, Berkeley

Honorary Chairperson:
P. Bruce Berra, Syracuse University

General Chairperson:
Gio Wiederhold, Stanford University

Tutorials:
James A. Larson
Honeywell Corporation

Awards:
K. H. Kim
University of California, Irvine

Program Committee Chairperson:
Benjamin W. Wah
University of Illinois, Urbana-Champaign

Treasurer:
Aldo Castillo, Cray Research

Publicity:
Dick Shuey, Consultant

International Coordination:
Tadao Ichikawa, Hiroshima University
G. Schlageter, Fern Universitat

Local Arrangements:
Walter Bond, System Development Corporation
Mary C. Graham, Hughes Aircraft

On-Site Registration:
Hamideh Afsarmanesh
California State University at Dominguez Hills

Program Committee Co-Chairpersons:
John Carlis, University of Minnesota
Iris Kameny, Rand Corporation
Peter A. Ng, North Texas State University
Winston Royce, Lockheed
Joseph Urban, University of S.W. Louisiana

Program Committee Members:

Jacob Abraham
Adarsh K. Arora
J. L. Baer
Faroh B. Bastani
Don Batory
G. Belford
Bharat Bhargava
Joseph Boykin
Richard Braegger
C. Robert Carlson
Nick Cercone
Peter Chen
Bernard Chern
David Choy
Wesley W. Chu
David J. DeWitt
David Du
Ramez ElMasri
Michael Evangelist
Domenico Ferrari
Hector Garcia-Molina
Georges Gardarin
Sakti P. Ghosh
Arnold Goldfein
Georg Gottlob
Laura Haas
Lee Hollaar
Yang-Chang Hong
David K. Hsiao
H. Ishikawa
Hemant K. Jain
Sushil Jajodia
Jie-Yong Juang
Arthur M. Keller
Larry Kerschberg
Won Kim
Roger King
Dan Kogan
Walter Kohler
Robert R. Korfhage
Tosiyasu L. Kunii
Winfried Lamersdorf
Matt LaSaine
W.-H. Francis Leung
Guo-Jie Li
Victor O.K. Li
Yao-Nan Lien
Witold Litwin
Jane W.S. Liu
Ming T. (Mike) Liu
Raymond A. Liuzzi
Vincent Lum
Yuen-Wah Eva Ma
Mamoru Maekawa
Gordon McCalla
Toshimi Minoura
Jaime Murow
Sham Navathe
Philip M. Neches
Erich J. Neuhold
G. M. Nijssen
Ole Oren
Gultekin Ozsoyoglu
Z. Meral Ozsoyoglu
C. Parent
J. F. Paris
D. S. Parker
Peter Rathmann
Lakshmi Rebbapragada
David Reiner
Gruia-Catalin Roman
Domenico Saccá
Giovanni Maria Sacco
Sharon Salveter
Peter Scheuermann
Edgar Sibley
John F. Sowa
David Spooner
David Stemple
Peter M. Stocker
M. Stonebraker
Stanley Su
Denji Tajima
Marjorie Templeton
A. M. Tjoa
Mas Tsuchiya
Yosihisa Udagawa
Susan D. Urban
P. Valduriez
R. P. VanDeRiet
Yann Viemont
Ky-Young Whang
S. Bing Yao
Clement Yu
Kwang-I Yu

CONFERENCE OVERVIEW

It is an honor to report on the program of the Third International Conference on Data Engineering. This year, we have received 209 submissions from fourteen countries. The level of international participation is tabulated as follows.

	Submitted	Accepted	Prog.Comm.
Africa	7	1	0
Asia	11	3	6
Australia	2	0	1
Europe	35	8	13
N. America	154	60	75

Each submitted paper was reviewed by one of the Program Committee Co-Chairs and by at least two members of the Program Committee, sometimes aided by local reviewers. The Program Committee met in Chicago on August 9-10, 1986, and decisions were made on all submitted papers and panel-session proposals. The following table shows an approximate classification by topics of the submitted and accepted papers.

	Submitted	Accepted
Database Design and Modeling	38	9
Performance Evaluation and Algorithms	18	8
Integrity, Security, and Fault Tolerance	17	8
Query Language	10	4
Artificial Intelligence	12	3
Knowledge Base	22	7
Database Machines	15	7
Distributed Database and Processing	11	3
Distributed Database Control	40	15
Software Engineering	10	1
Applications	16	7

We have organized the submitted papers into 21 sessions. Two of these sessions (Sessions 20 and 23) are devoted entirely to papers on industrial applications and experience.

A current working definition of data engineering was proposed by the IEEE Computer Society Technical Committee on Data Engineering, which states that "the study of data engineering focuses on the key technical issues related to data and knowledge about the data in the design, development, management, and utilization of information systems, such as languages to define, access, and manipulate databases as well as knowledge bases, and, in general, numerical, textual, and pictorial information; strategies and mechanisms to provide system modeling and design and data access, security, integrity, and control; architectures, systems and components to provide data services within centralized and distributed information systems; and development of ways to prolong the useful life of data." Most of the topics covered by the accepted papers are related to one or more aspects of the above definition. With the limited time and resources, it would be quite impossible to cover thoroughly all the aspects of data engineering. However, the accepted papers represent some of the best results in both the theoretical and applied aspects of data engineering.

We will have a discussion concluding each session with contributed papers. This was an experimental idea at the Second Data Engineering Conference that was well received. A discussant will be assigned to these sessions and will lead a discussion relevant to the presentations in the session. The discussant will present a short position statement addressing questions such as 'where do we stand?' and 'which way should we go?' The papers presented in the session can also be used as a basis for the discussion.

We are also complementing these discussion sessions with six traditional panels in areas where research directions are being set.

We are privileged to have three outstanding keynote speakers who are scheduled to speak on the first two days of the conference. On the first day Philip Neches will speak on his experience building real database machines. His presentation will be followed by a track on architectural support and parallel processing throughout the first day.

We are privileged to have Lofti Zadeh and Jerold Kaplan to keynote the conference on the second day. Dr. Zadeh is world renowned for his research on fuzzy logic and artificial intelligence. He was particularly instrumental in developing methods for uncertain data management, which he will address in his key-

note speech. Dr. Kaplan is known for his work on making database act cooperatively with the users. He will share with us his recent experience in building new commercial products which provide intelligent and flexible databases for individuals. Their presentations will be followed by a track on artificial intelligence and knowledge representations throughout the second day.

In the final plenary panel, we will summarize this conference by addressing questions around the general theme: "What Have We Learned?" We hope that after you have heard the presentations in this conference, we can have a fruitful discussion regarding what we learned in this conference and possible improvements for future Data Engineering Conferences.

The Technical Committee on Data Engineering which sponsors this conference produces a quarterly bulletin, with each issue devoted to one topic of interest to the Data Engineering community. An application form requesting membership can be found at the end of these proceedings.

I believe that this year's Program provides all participants with varied, in-depth technological insights and ideas. All suggestions for the improvement of future Data Engineering Conferences are welcome.

Benjamin W. Wah
Program Chairperson

TABLE OF CONTENTS

Session 5: Panel—Relationship Between Data Engineering and Software Engineering

(Moderator: S.S. Yau, *Northwestern University*; Panelists: B. Boehm, *TRW Systems, Inc.*; C.V. Ramamoorthy, *University of California, Berkeley*; G-C. Roman, *Washington University*)

Session 6: Architectural Support for Database Management

(Chairperson: J.Y. Juang, *Northwestern University*)

Session 7: Evaluating Recursive Queries

(Chairperson: D. Stemple, *University of Massachusetts*)

Session 8: File Structures

(Chairperson: G. Ozsoyoglu, *Case Western Reserve University*)

Session 9: Parallel Processing Database Systems

(Chairperson: E. Bertino, *National Research Council, Italy*)

Session 1: Access Methods

Chairperson
R. King
University of Colorado

Linear Hashing with Priority Splitting:

A Method for improving the retrieval performance of linear hashing

Willard D. Ruchte

Alan L. Tharp

Computer Science Department
North Carolina State University
Raleigh. North Carolina 27695-8206

ABSTRACT

Linear hashing is a technique for constructing dyanmic files for direct access. It has an advantage over other dynamic methods in that it lacks a directory. However, a weakness of the method is that at high packing factors, it requires more probes on the average to access a record than do many of the static methods.

This paper presents a straightforward modification of linear hashing which, according to experimental results, significantly reduces the average number of retrieval probes in almost all cases when compared with standard linear hashing. The parameter of overflow page size is an important one for adjusting performance. By choosing an appropriate overflow page size, the user may obtain results which are also better or comparable to those of other variants of linear hashing. In addition, the paper analyzes the effects of varying the primary page size, the overflow page size, and the packing factor on retrieval performance.

INTRODUCTION

When computer applications involve volumes of data too large to be stored in main memory, the necessity of placing it in auxiliary storage brings with it a degradation in performance when accessing records directly. By using a direct access scheme to retrieve the records stored in these files, instead of accessing the records sequentially, the average number of accesses to the auxiliary storage may be drastically reduced. However, to directly access a record, it is required that the address of that record within the file be known prior to its retrieval. To obtain the address of a record, most direct access schemes involve the designation of a key field within the record and the application of a mapping function that transforms the key into an address in the file. The situation is complicated, however, by the fact that any function that maps a set of possible keys containing **n** elements into a file with address space less than **n** will necessarily map more than one key to some address or addresses. Numerous collision resolution schemes have been developed to deal with this problem, including coalesced hashing [1,2] and computed chaining [3]. Each of these standard methods involves the application of a single hashing function to each of the keys to be placed in the file. The best hashing functions are those that distribute the keys evenly throughout the address space, which requires that the size of the address space be included in the definition of the hashing function. The function's dependence of the size of the address space means that the file size must be fixed in advance of storing any records, and it must never change. Otherwise, the records previously stored would not be retrievable. Files based on any of these collision resolution methods require a complete reorganization for the file size to be changed.

More recent developments have lead to hashing and collision resolution schemes that allow the file size to change as records are added or deleted. Dynamic Hashing [4], Extendible Hashing [5], Trie Hashing [6], and Linear Hashing [7] each address the problem of changing file size in different ways. This paper will describe a modification to Linear Hashing and discuss experimental results obtained using the standard method and the modified method.

STANDARD LINEAR HASHING

The concept behind linear hashing is that it is possible to have more than one hashing function in use for a given file, as long as you can be certain as to which one to use to retrieve a given key. Linear hashing uses a scheme that lets two different hashing functions be active at any given time. The initial number of main pages, **N**, for the file may contain any number of records. A hashing function involving **N**, such as h_0 = key mod **N** is used to start. The page on which a record should be placed is determined by h_0(key). If this page is already full, the record is placed in an overflow page. A link from the main page points to the overflow page. In this way, a main page may be the start of a chain of pages containing records that hash to the same main page. As records are stored in the file, a count is kept of the number of records stored and the space available for storage. When the ratio of these two exceeds a predetermined level, the upper bound for the packing factor, the first page (page 0) is split to create a new page, numbered **N**+1. To indicate that page 0 is the appropriate page to split, a pointer is maintained that initially points to page 0. The hashing function is changed by replacing **N** with 2**N**, giving h_1 = key mod 2***N**, and the records stored on page 0 are reinserted using the new hashing function. The pointer indicating the next page to split is advanced to page 1. When we have reached the point at which all the pages which were in the file initially have been split, we know that all the records in the file have been stored using the hashing function h_1. Thus h_0 is no longer needed to retrieve any of the records. At this level the two hashing functions will

be h_1 and h_2=key mod 2*(2*N). It can be seen that the two hashing functions active for a given level will be

$$h_{level} = \text{key mod } N*2^{level} \quad (1)$$

and

$$h_{level+1} = \text{key mod } N*2^{level+1} \quad (2)$$

This scheme addresses the problem of how to allow the file size to change while maintaining direct access to the records in a simple and elegant manner. The fact that no extra storage space is required to implement linear hashing is an advantage that it has over other dynamic methods, most of which use an index of some type which requires extra storage. However the fact that overflow chains may develop degrades the retrieval performance.

Other modifications to linear hashing include the ideas of 1) having the file expand through a series of partial expansions [8,9,10], 2) incorporating overflow records into the main file [9], and 3) using multiple overflow chains [10]. Mullen has also described a method for eliminating the need for an overflow file [11]. Ramamohanarao and Sacks-Davis [12] presented an algorithm for dealing with overflow records through recursive applications of the linear hashing algorithm .

LINEAR HASHING WITH PRIORITY SPLITTING

It is apparent from the previous description that one of the main contributors to poor performance by the standard linear hashing scheme is the possibility that long overflow chains may develop and remain un-split for long periods of time while the pointer moves through the pages one by one. The primary strengths of the technique are that it allows variable file size and requires no extra storage space in the file. The modification described here attempts to minimize the degradation of retrieval performance due to long overflow chains and to preserve the standard method's strengths.

The fundamental idea behind the modification is that performance will be improved by splitting the page with the longest overflow chain immediately, rather than waiting for the pointer to advance step by step through the file. An example which demonstrates the differences between standard Linear Hashing and Linear Hashing with Priority Splitting is shown in the figures below. Figure 1 shows a Linear Hashing file with N = 2, 4 records per main page, and 1 record per overflow page. The file is at level 1, and page 0 has already been split. The pointer, NEXT, identifies page 1 as the next page to split. Notice that an overflow chain has developed on page 3.

Assuming that the next record added to the file causes the upper bound for storage utilization to be exceeded, the state of the file will be as shown in figure 2 if standard Linear Hashing is used. Notice that the page identified by NEXT has been split, creating a new page with an index of NEXT + $N*2^{level}$ = 5. The records previously stored on page 1 using the hashing function h_{level} (h_1) are redistributed between pages 1 and 5 using $h_{level+1}$ (h_2). NEXT has been advanced to indicate the next page in the sequence, 2, as the next page to split. The overflow chain on page 3 remains unchanged, and will continue to degrade retrieval performance until the records in its overflow chain are redistributed.

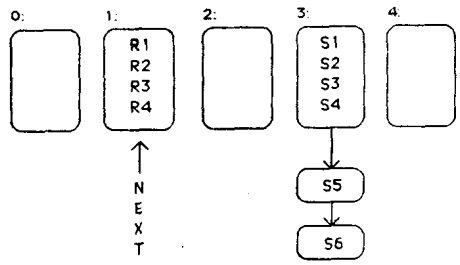


Figure 1:
Linear Hashing File
(level 1, page 0 already split)

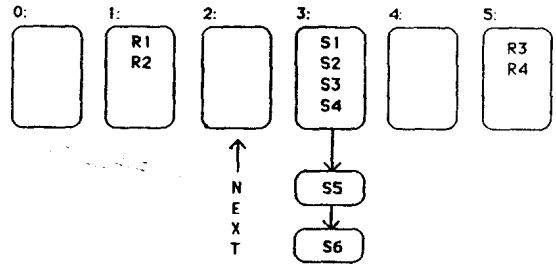


Figure 2:
Linear Hashing File
After Standard Splitting

Figure 3 shows the results of using Linear Hashing with Priority Splitting in the same situation. Rather than splitting the page identified by NEXT, the page with the longest overflow chain, page 3, is split. The records on page 3 and its overflow chain are redistributed between page 3 and page 7 (3 + $N*2^{level}$). The retrieval performance is immediately improved since the overflow chain attached to page 3 no longer exists.

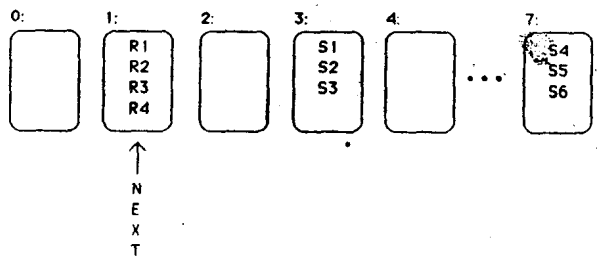


Figure 3:
Linear Hashing File
After Priority Splitting

Implementing this change requires that several problems be overcome. First it must be observed that as the pointer advances through the file in the standard method, it serves as a "division marker" between those pages that have been split and those that have not. If the pages are no longer going to be split in order, some other method of determining whether a page has been split or not must be devised. Next is the problem of determining which of the chains to split, and maintaining a record of the lengths of the chains. Finally, there is the question of what to do if there are no pages with overflow chains when the upper bound is exceeded, and a page must be split.

The approach taken in producing this modification was to use the standard method as a "worst case", and to add logic to the insertion algorithm that would improve on this if possible. As with the standard method we begin with **N** pages in the file, maintain a variable, **LEVEL** to record the active level, and a variable, **NEXT**, to point to the next page to be split. We use the same hashing functions (1 and 2) described in the previous section.

In addition to these, the modification requires 1) a heap structure with a node for each primary page which has a chain of overflow pages, and 2) a powerset structure (e.g. Pascal's SET) with a number of bits equal to the maximum number of pages in the file.

The heap structure will be used to maintain a list of the chains that are candidates for splitting. Each time an overflow page is added to a chain, the heap will be updated so that the node containing that page number is either inserted (if this is the first overflow page for that chain) or it's length field is updated and it is moved up in the heap (if necessary to keep it in proper order). We will be able to split only those pages that were originally in the file at the beginning of a level until all the pages at that level are split. Therefore we must order the heap based on whether the chain represented is "splitable" at this level, and on the length of the chain (in that order of precedent). The set structure will be used to keep track of the pages that have been split. One bit in the structure will represent each page of the file, and that bit will be set when the corresponding page is split. When all the pages at the current level have been split, all the bits will be cleared to 0.

We begin with **LEVEL**=0, **NEXT**=0, the queue empty, and the bits of the indicator set cleared. Each record is hashed using h_{level} and placed in the corresponding main page. If the main page is full, the record is added to an overflow page, and the main page link is set to point to the overflow page. The number of the main page, and the length of the chain (1 in this case) is inserted into a node and added to the heap. If another overflow page is added to the same chain, the same node is updated for the new length, and the heap is re-ordered. If an overflow page is added for another chain, a new node is added (in order) with a length field of 1. When the specified upper bound is exceeded, a chain must be split. First the root node in the queue is checked. If it is a chain that is splitable at this level, i.e. the bit corresponding to that page number is not set to one in the indicator set and the page number is less than $N*2^{level}$, then that node is removed from the heap, and the chain it represents is split. The new page added to the file is given an index of the split page's index + $N*2^{level}$, and the records are placed using the $h_{level+1}$ hashing function. Just as with standard linear hashing, we are certain that all the records stored at a page will hash to either the same page or the new page. If we must split a page, and the heap is empty or the head of the queue is a chain that has already been split, we know that there are no overflow chains for the pages remaining to be split, and we split the page that is being pointed to by **NEXT**. Having split a page, we advance **NEXT** to point to the next page that has not yet been split at the current level. In this fashion, we can take advantage of splitting the longest chains first, while using the standard method's technique to fall back on when no chains can be split.

An algorithm for insertion of records into a file using Linear Hashing with Priority Splitting is given below.

I. Determine the chain, **m**, which the record maps to using $m=h_{level}(key)$.

II. Check whether the chain has been split by checking the m^{th} bit in the indicator set.
If the bit is set, the chain has split, then

A. set $m = h_{level+1}(key)$.

III. Insert the record into chain **m**.
If the record will not fit on the main page, then

A. follow the overflow links until a page with a null link is found, and count the number of overflow pages in the chain.

B. If that overflow page is full (or if there is not yet an overflow page), then

1. create a new overflow page, link it to the chain, and update the heap by

a. if a node for this chain exists, then update the length, else add a new node

b. ordering the updated heap by level (ascending) and length (descending)

IV. Check the upper space utilization bound. If it is exceeded then

A. Check the heap to see if the root node is at the current level.
If it is, then
1. remove it from the heap, set **SPLIT_CHAIN** to that chain number,
2. otherwise set **SPLIT_CHAIN** to **NEXT**.

B. Set the bit corresponding to this chain in the indicator set to show that it has been split.

C. Create a new chain with index equal to **SPLIT_CHAIN** + $N*2^{level}$.

D. For each record on the chain **m**, determine whether to move it. If $h_{level+1}(key)$ <> **NEXT** then

1. Move the record to the new chain.

E. Update the parameters. Set **NEXT** to the next chain still at the current level (using the indicator set to determine whether a chain has been split).
If **NEXT** >= $N*2^{level}$, all of the chains on the current level have split, Then

1. Reset **NEXT** to 0, re-initialize the indicator set, and create a new level by setting **LEVEL** to **LEVEL** + 1