A STUDY OF MULTIDIMENSIONAL BINARY TREE STRUCTURES

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment for the Degree of
MASTER OF PHILOSOPHY

G. SREE RAMA MURTHY

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067

1980

P R E F A C E

The research work embodied in this dissertation has been carried out in the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi. The work is original and has not been submitted so far, in part or full, for any other degree or diploma of any University.

G. Sreeramamurthy

Dr. P.C. Saxena
Supervisor

Prof. D.K. Banerjee
Acting Dean

School of Computer & Systems Sciences
JAWAHARLAL NEHRU UNIVERSITY

ACKNOWLEDGEMENTS

I am deeply indebted to Dr P.C. Saxena under whose guidance I had the privilege to carry out this research.

I sincerely thank Professor N.P. Mukerjee, Professor D.K. Banerjee, for their co-operation and providing me all facilities in completing this work.

I also thank Dr. A.K. Majumdar and Dr R.G. Gupta for their valuable suggestions and constructive criticism.

I am thankful to Mr S.P. Rana, Research Scholar, Computer Centre, I.I.T., Delhi, for the long discussions I had with him and his valuable suggestions.

I am thankful to Mr Jose Abraham, Scientific Officer, NCSDCT, Bombay, for getting permission to use DEC-10 Computer system at NCSDCT and who taught me the Art of Computer Programming.

I also thank Sutrave Brothers, Kirti and Pramod for co-operation and encouragement.

I am also thankful to all my colleagues and staff in the School for their co-operation. I also thank the Library staff, Science Library, for their co-operation in completing this work.

-/-

I express my deep gratitude to my parents and Brother-in-law N.K. Rao for their full co-operation and encouragement.

I also thank Mr D.K. Pahwa for typing neatly.

I am also thankful to Jawaharlal Nehru University for providing the financial assistance.

G. Sree Rama Murthy

G. SREERAMAMURTHY

New Delhi,
July 9, 1980.

C O N T E N T S

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

INTRODUCTION

Information is an essential resource in the present day society. Collection and maintenance of this resource is quite costly and time consuming. The events happen so rapid that the organizations have to make quick decisions, sometimes even instant decisions. The effects of these quick decisions may be beyond the control of the organizations. The organizations need most recent information to maintain their competitive position in the present day environment.

Information flow is continuous, rapid and voluminous. Sometimes the useful information may be very low per cent of the information received. This received information should be filtered, organized and maintained in such a way that any time, the information required by the organization is available and retrieved fast. Achieving this manually became almost impossible because of the volume of the information. To utilize the information available efficiently, mechanism is introduced. The organization of the information is important since the retrieval techniques depend on the logical and physical organization of this information.

The aim of this work is to present a data structure which is advantageous for retrieving the information region-wise. The data structure called the Multidimensional Binary Tree Structure is described in Chapter II. This data structure allows generalised and complicated queries to be answered very efficiently. For implementing the various searching algorithms based on this data structure, a small Experimental Data Base of Jawaharlal Nehru University students is developed. The organization of this dissertation is as follows:

Chapter I deals with the Data Base concepts and its advantages and disadvantages over the Conventional Data Processing systems. Various types of conventional logical and physical data structures are also discussed in this chapter.

Chapter II deals with the various types of searching techniques available. The Multidimensional Binary Tree Structure is described in this chapter. The Experimental Data Base developed for implementing the various searching algorithms based on this data structure is also described.

Chapter III deals with searching algorithms based on Multidimensional Binary Tree Structure and their implementation.

Chapter IV consists of conclusions. The source programmes and the results are given in Appendix.

****

CHAPTER I

DATA BASE CONCEPTS AND DATA STRUCTURES

1.1 Conventional Data Processing Systems

In most of the conventional data processing systems the organizations generally maintain independent sets of files for each of the departments. The independent maintenance of files creates several difficulties, like redundancy, integrity of data, standards maintenance etc. as the volume of data and interrelationship between various departments increases. To avoid these difficulties the concept of Data Base came into existence.

1.2 What is a Data Base[1,2]

The term Data Base refers to the most efficient form of storage and management of a large collection of information. The range of information present and the simplicity in accessing the information present and so on constitute the measure of a successful Data Base system. The Data Base system with the above facilities together with the flexible capabilities of retrieving, inserting, updating and deleting of informations under the centralized control can be referred as Data Base Management System (DBMS).

The basic entities of a Data Base are a number of data elements (ex. logical elements), each of which is a unit of data that is complete in itself. The Data Base Management

System comprises of a set programmes which can manage to execute the functions like retrieval, update etc, on these data items according to the user needs. In an organization many users may be using the same Data Base, which means that each user may be using a small portion of it and the portions used by many users may be overlapping.

The Data Base system provides a centralised control over its operational data through a single identifiable person called Data Base Administrator (DBA). It will be the responsibility of the DBA, who is also a part of the Data Base System, to understand the present and future requirements of the users. The information should be collected and organized according to these requirements.

The various terms used in Data Base are:

Data Item: A data item is the smallest unit of named data and also referred as a field or data element. An example may be the age of the student.

Data Aggregate: A data aggregate is a collection of data items within a record which is given a name and referred to as a whole. An example may be the date of birth of the student which is composed of the data items, day, month and year.

Record: A record is a named collection of data items or data aggregates. When an application programme reads data from the Data Base it may read one complete logical record. An example for this may be the academic record of a student.

Schema: The overall logical Data Base description together with the interrelationships is referred to as schema.

Sub-schema: The term sub-schema refers to an application programmer's view of the data he uses. Many different sub-schemas can be derived from one schema.

Key: A data item used to identify or locate a record is referred to as key. An example for this may be the name of a student in the university.

Primary Key: A key which uniquely identifies a record is referred to as primary key. An example for this may be the identification number of a student in the university.

Secondary Key: A key which does not identify a unique record but which identifies all those which have certain property is referred to as secondary key. For example, the marital status of students in the university may be a secondary key. It will not identify a student uniquely but this can be used to identify all those students who are married.

Index: An index may be referred as a table which operates with a procedure that accepts information about certain attribute or data item values as input and gives information as output which assists in quickly locating the record or records that have these attribute values.

The advantages of a Data Base system over the Convention DP systems are as follows:

(i) Data Base System Promotes Multiple Usage of Data

The Data Base system allows sharing of the stored data. Most of the application programmes requests can be satisfied by the existing DBMS software. Whenever a specialized request which cannot be satisfied by the existing software, comes a module for that can be developed and can be incorporated in DBMS software easily.

(ii) Redundancy of the Stored Data in a Data Base is Controlled

Since the Data Base is centrally controlled by the DBA, he can integrate all the files that essentially requires the same data. Thus the common data can be stored only once. However, in practice a controlled amount of redundancy is often introduced deliberately in order to give improved access times or simpler accessing methods etc. and to provide the capability to recover from accidental loss of data.

(iii) Maintenance of Data in a Data Base is Simpler

With the centralized control or the Data Base, the DBA is able to enforce standards and see to it that these standards are followed in the representation of data. The problems of maintenance of data and interchange of data between various installations are therefore simplified to a great extent.

(iv) Data Stored in a Data Base are Independent

The Data Base allows the application programmes to be data independent which is not possible in conventional DP systems. The sub-schema defined by the application programmes can be derived from the schema of the Data Base with the help of the Data Base Management software. This will increase the portability of the application programmes.

(v) Easier to Enforce Privacy and Security Measures

The DBA can appropriately channelize the means of access to the Data Base, since all the operational data is under his control and the DBA has the knowledge of the application programmes and their data requirements. Without an efficient DBA of course the security of data in a Data Base is more at stake than in the conventional DP systems.

(vi) Integrity of Data can be Maintained

Maintenance of integrity of data means ensuring that any two entries representing the same fact are consistent and only the correct data are entered. Since the redundancy in Data Base systems is minimal and updation etc. has to be done at fewer places, the problem of integrity is reduced considerably.

(vii) Simplicity in Accessing the Data by Users

One of the important reasons to have Data Base systems will be that they permit users to employ data very easily. Powerful languages, like DL/I, are developed which permit untrained users to access the data easily.

(viii) Powerful Searching Capabilities

The user of a Data Base may ask a wide variety of querries about the data that is stored. In the majority of applications, the types of querries are anticipated and the logical and physical data organizations are described to handle them with suitable speed. The capability to search a Data Base quickly with different search criterias is highly dependent on the logical and physical data organizations. One of the objectives of the Data Base organization is to achieve fast and flexible search capability.

Some of the logical and physical data organizations generally used in Data Base systems are described in sections 1.3 and 1.4. The conventional searching techniques based on these data organizations will be discussed in Chapter II.

1.3 Logical Data Organizations/File Structures

The various types of data organizations generally used in Data Base systems are:

1. Flat-File Structure
2. Hierarchical/~~Network~~ Tree Structure
3. Network Structure
4. Relational Data Base

Each of these organizations are described in this section.

1.3.1 Flat-File Structure

A two dimensional file structure is called a "Flat-File". An example of this is given below:

Identification	Student Name	Personal Data	Medical Data
11111	AMAR SINGH	0302550177	O*, NORMAL
11112	AMARJIT	0509560877	B
11113	AMIRUDDIN	0611491577	A, NORMAL
11114	AMRITLAL	0512590376	B

Fig. 1.1 Flat-File Organization

Each column has data-items relating to a particular attribute. The attributes in the above example are personal data, medical data. The identification number in the entity is identifier of the record.

1.3.2 Tree or Hierarchical Data Structure[1,2]

A tree structure composed of a hierarchy of elements called nodes. The uppermost level has only one node called the "root". Every node has only one node related to it at a higher level and this is called its "parent". Each node can have one or more nodes related to it at a lower level, which are called "sons." Each node is considered as a subtree having that node as root of the subtree.

In the tree structure each node represents a record in the file. Tree structures can be used in both logical and physical data organizations. In logical data organizations they are used to describe relations between record types. In physical data organizations they are used to describe sets of pointless and relations between nodes.

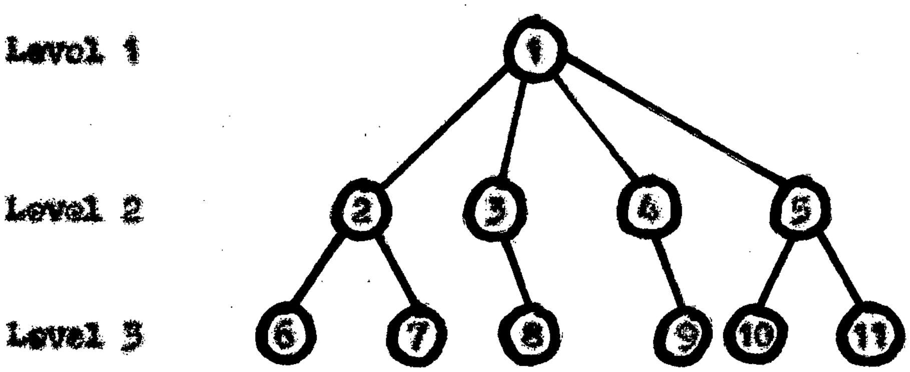

Fig. 1.2 A Simple Tree data Structure

1.3.2.1 Balanced Tree Structure: In the Balanced Tree Structure each node can have the same number of branches and the same branching capacity.

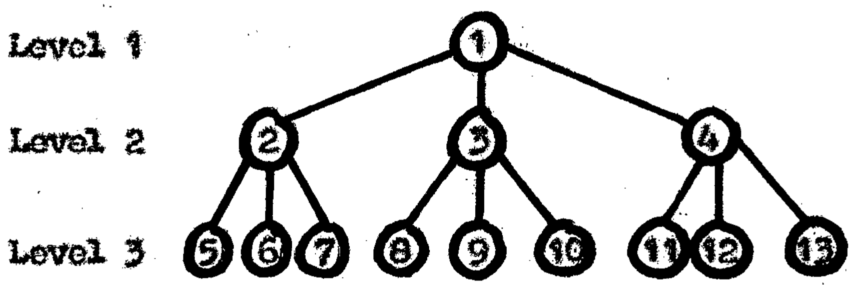

Fig. 1.3 Balance Tree Data Structure

1.3.2.2 <u>Binary Tree Data Structure</u>

The Binary Tree Data Structre permits up to two branches per node. Some logical data organizations fit naturally into binary tree structure but its main advantages will be in physical data organization.

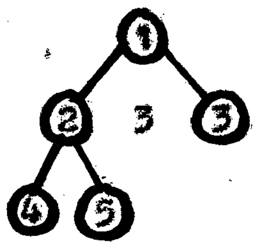

Fig. 1.4 Unbalanced Binary Tree

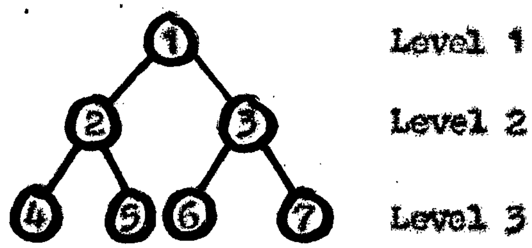

Fig. 1.5 Balanced Binary Tree

The following property can be introduced in the Binary Tree Structure: if Q and R are the nodes in left and right subtrees of the node P. Then

$$Q < P \quad \text{and} \quad R > P$$

The binary tree structure with this property enables efficient search of the table and quick insertion and deletion of nodes of the tree.

1.3.3 Network Data Structure

A more general data structure than a multilink tree structure is the network data structure. Unlike in tree data structure, a son can have many parents in this Network data structure. An example for this structure is shown below:

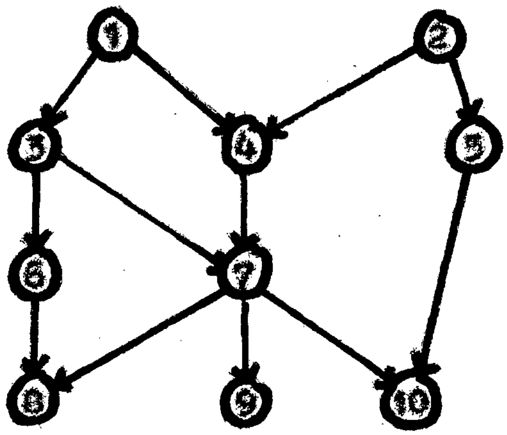

Fig. 1.6 A Simple Network Data Structure

The main disadvantage of the network data structure is insertion and deletion of records cumbersom because many pointers have to be updated each time. This structure may also increase the record length, since all the pointers have to be accommodated in the record.

1.3.4 Relational Data Base[1,2]

Any representation of data can be reduced to a two-dimensional table with certain redundancy. The replacement of complex structure by two-dimensional tabular forms, without losing any relationship between the data-items is called Normalization. A Data Base terminology for flat-files is "Relation", and a Data Base constructed from Relations is called a Relational Data Base. The rows are called the tuples (records) of the Relation and the Relation with n-columns is an n-tuple. Each column is called an attribute. The attribute takes values and the values in a column of a Relation is called Domain. Operations among various relations are described precisely by mathematical notations based on Relational Algebra and Relational Calculus. These operations are responsible for providing the flexibility required by the users.

1.4 Physical Data Structures

The various types of physical data structures are:

1. Sequential unordered data structure
2. Sequential Ordered Data structure
3. Index Sequential data structure
4. Inverted File data structure

All these structures will be described in this section.

1.4.1 Sequential Unordered data structure In this data structure all the records are stored contiguously without any restrictions. Insertion of records, generally, is done at the end of the structure.

1.4.2 Sequential ordered data structure: In this data structure all the records are stored contiguously but in an ordered manner depending on single key or multiple keys.

Generally, these records will be sorted in ascending or descending order on the basis of a single key or multiple keys. Insertion of records will be done at the appropriate position depending upon the insertion record key.

1.4.3 Index Sequential data structure [1,2]

In this data structure, the records will be laid out sequentialy in key sequence and usual method of addressing it is by specifying the Index. When an Index is used for addressing a file, first the search takes place on the Index. The Index contains the address of the record or address of a location which may contain the address of the record. Since the data structure is in key sequence, the Index does not normally contain a pointer to every record but a reference to blocks of records which can be searched. Referencing to block of records rather than individual records substantially reduces the size of the Index. Sometimes the Index is often too large to be searched in its entirety, and an Index to an Index is used. Thus this data structure can have more levels of Indices based on the secondary keys.

contd....

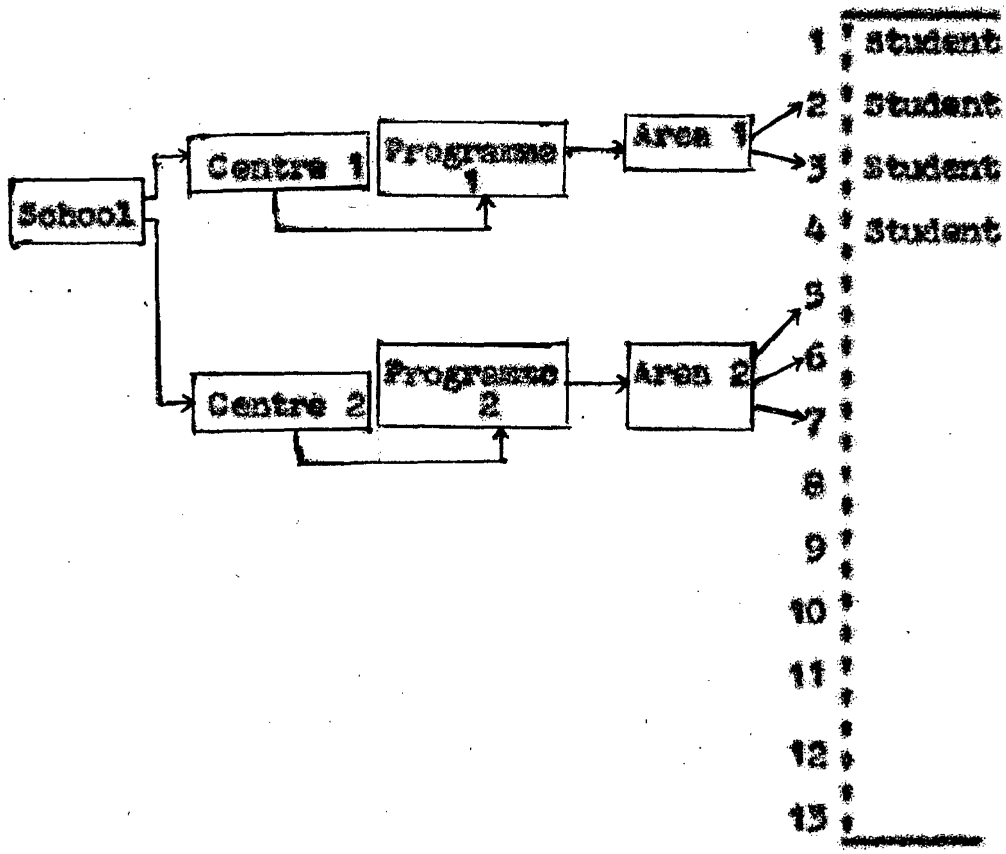

Fig. 1.7 A multi-level Indexed Sequential data structure

1.4.4 Inverted File Data Structure[1,2]

In this data structure the data values may be kept and the values forming a particular record may not necessarily be stored contiguously. There is an occurrence indeed which is examined to find occurrences of a particular data value and a separate data index to find the data associated with those occurrences. A dictionary is employed to point to the occurrence index. The lists of occurrences in this index are examined and compared, and then the required data values are found via a data index.

The dialogue for searching an inverted file structure may proceed in three stages. First the dictionary is inspected, then the Occurrence Index. The Occurrence Index may be searched multiple times before proceeding to the data index to locate the required record.

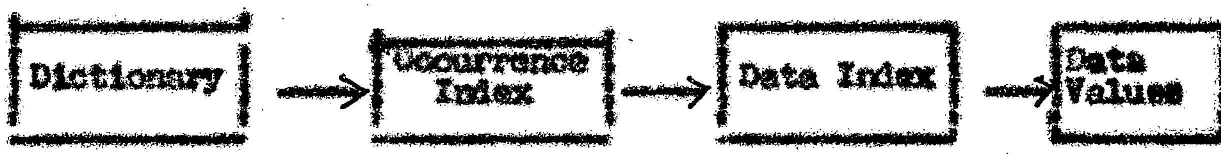

Fig. 1.8 Typical Inverted file structure

The various types of searching techniques will be discussed in detail in Chapter II. The data structure, multidimensional Binary Tree structure[6,9] will also be introduced in Chapter II.

****

CHAPTER II

SEARCHING ALGORITHMS AND MULTIDIMENSIONAL BINARY TREE STRUCTURE

The various types of Logical and Physical Data Structures are described in Chapter I. Here we will discuss about the various types of searching techniques based on these data organizations. The data structure called Multidimensional Binary Tree Structure[4,5] is also introduced. The Experimental Data Base developed for implementing this data structure is described in section 2.6.

Before discussing about the various searching techniques, the different terms used will be described.

Single Key Search: In this case, search is based only on a single key. The key, which will be primary key, uniquely identifies a record. An example may be find the name of the student whose identification number is given.

Multiple Key Search: In this case, search is based on multiple keys. The multiple keys may contain one primary key which identifies the record uniquely and one or more secondary keys. An example for this may be: find out the names of top 5 students in M.A. History course.

Exact Match Search: The simplest type of search is the Exact Match Search. In this search the key of the record is given and the record with key equal to the specified key is retrieved. An example of this search may be: find the name of the student whose key is 11141.

Region Search:[4,5] In this search, the limits for all the keys are given and all records whose keys lie within the limits specified can be found. An example for this may be, find the names of all the students in a particular programme of study.

Partial Match Search:[4,5] In this search, only a subset of the keys are specified and all the records having this subset of characteristics can be found. An example for this may be: find the names of all the students having the grade point 5.5.

The various types of conventional searching techniques are:

(a) Sequential Search on unordered data structure
(b) Sequential Search on ordered data structure
(c) Binary Search
(d) Indexed Sequential search

2.1 Sequential Search on unordered data structure:[5]

Given an unordered data structure of records R_1, R_2,...R_N whose respective keys are K_1, K_2,....K_N the following algorithm searches for a record whose key is K. The only assumption in this algorithm is that the data structure contains at least one record. This algorithm

<u>Algorithm</u>

```
Procedure  •  Sequential Search (K, N)
Comment:      The input to this procedure is N, the
              total number of records in the data structure
              and K the key of the record to be retrieved.
              Output will be the record with key K if
              present, otherwise "NOT PRESENT".
   begin
         set  i = 1
              search indicator "SEARCH"
          DO  until search indicator = "SUCCESS"/"UNSUCCESS"
           If K = Ki then
              begin
                   display the record with key Ki
                   search indicator = "SUCCESS"
              end
           else
              begin
                   set  i = i + 1
                   IF   i > N  then
                        begin
                          display "NOT PRESENT"
                          search indicator = "UNSUCCESS"
                        end
              end
          end
   end
   Stop.
```

2.2 Sequential Search in Ordered data structure[3]

Given a data structure of records R_1, R_2, ... R_N whose keys are in ascending order K_1, K_2, ... K_N this algorithm searches for a given argument K. For convenience, this algorithm assumes the presence of a dummy record R_{N+1} whose key value is K_{N+1} = K.

Algorithm:

Procedure Sequential Search ODs (K)

Comment: Input to this algorithm is the key of the record K. The output may be the record with key K if present in the data structure, otherwise "NOT PRESENT".

```
begin
    set  i = 1
    Search indicator = "SEARCH"
    DO  until search indicator = "SUCCESS"/"UNSUCCESS"
        IF  K ≤ K_i  then
            begin
               IF  K = K_i then
                  begin
                      display the record with K_i
                      search indicator = "SUCCESS"
                  end
               else
```

```
              begin
                 i = i + 1
              end
           end
      else
           begin
                displayed "NOT PRESENT"
                search indicator = "UNSUCCESS"
           end
   end
end
Stop.
```

2.3 Binary Search:[3] Given a data structure of records R_1, R_2, ... R_N whose keys are in increasing order K_1, K_2, ... K_N, this algorithm searches for record whose key is K.

Algorithm

Procedure BINARY SEARCH (K, N)

Comment: Input to this algorithm is N, the maximum number of records in the data structure and K they key of the record to be retrieved.

```
begin
    set  l = 1, u = N
    search indicator = "SEARCH"
    IF  u < 1   then
        begin
            display "NOT PROPER LIMITS"
            search indicator = "UNSUCCESS"
        end
    or
      IF    K > Ku   then
      IF    K < Ko   then
        begin
            display "NOT PRESENT"
            search indicator = "UNSUCCESS"
        end
    DO   until search indicator = "SUCCESS"/"UNSUCCESS"
        begin
            i = (l + u)/2
        end
      IF    K < Ki  then
            u = i - 1
      else
        IF  K > ki  then
            l = i + 1
        else
        begin
            display the record with key Ki
            search indicator = "SUCCESS"
        end
    end
end
Stop.
```

2.4 Indexed Sequential Search [6]

In this search, when a request for a record occurs, first it will search Index for the corresponding argument which may point to the location of the record or the location another Index etc. This depends upon the levels of indexing. The arguments in the Indices may be Sequential data structure or Binay data structure. The search always starts from the begining of the Index table. In the multi-level Index sequential organization, the argument of the higher level Index table points to the beginning of the next level Index table.

After discussing the conventional searching techniques it is found that all these techniques are not very efficient in the case of multiple key search and partial match searches. The Sequential Search is simple but it is very inefficient because of its high complexity. The Binary search will be efficient for single key searches. But it is inefficient in the case of multiple key searches because the binary tree organization is based on single key i.e. primary key. The Index Sequential search is efficient up to some extent. But as the number of Indices based on secondary key increases, the Index sequential search also becomes inefficient because it has to search many Index tables before getting the required record. Partial match searches are quite difficult to implement. Insertion and deletion is cumbersome because many index tables have to be updated.

The multidimensional Binary Tree structure, which is described in section 2.5, is quite helpful in overcoming these difficulties. This data structure is simple and can be easily implemented. The proposed Multidimensional Binary Tree structure is implemented and the algorithms based on this data structure are tested by developing an Experimental Data Base of Jawaharlal Nehru University students. This Data Base is described in section 2.6.

2.5 Multidimensional Binary Tree Structure

Multidimensional Binary Tree structure is a generalised structure of Binary tree structure. Each record is stored as a node having m-keys in the m-dimensional tree structure. In addition to the m keys which comprise the record, each node contains three pointers which are either null or point to another node in the m-dimensional tree. Each pointer can be considered as specifying a subtree. Associated with each node though not necessarily stored as a field, is a discriminator, which is an integer between 0 and m-1, inclusive. All nodes on any given level of the tree have the same discriminator. The root node has the discriminator 0, its two two sons have discrimator 1 and so on to the mth level where the discriminator is m-1, the (m+1) the level has discrimator zero and the cycle repeats.

The m keys of the node P will be called $K_0(P)$, $K_1(P)$.... $K_{m-1}(P)$ the pointers will be PARENT (P), LOSON (P), HISON (P) and the discriminator will be DISC (P). The pointer PARENT (P) will point to the parent of the node P. For the root node, PARENT is null and for any node in the tree only one parent exists. LOSON (P) and HISON (P) point to the low son and high son of P in the tree structure. These may be null when there doesn't exist any more subtrees.

The order of the tree is defined as:

For any node P in the m-dimensional tree, let $\hat{j}$ be the DISC (P). Then for any node Q in LOSON (P) it is true that $K_j(Q) \leq K_j(P)$; likewise for any node R in the HISON (P), it is true that $K_j(R) \geq K_j(P)$. The equality condition may create problems in searching the data structure because it does not know which son to choose as successor. This can be resolved by using the function SUCCESSOR (P,T). This function returns either LOSON or HISON depending on the super keys $S_j(P)$ and $S_j(T)$ defined below.

Let j = DISC (P)

$$S_j(P) = K_j(P)\, K_{j+1}(P) \ldots\, K_{m-1}(P)\, K_0(P) \ldots K_{j-1}(P)$$

the cyclical concatenation of all keys staring with K_j.

The SUCCESSOR returns LOSON if $S_j(T) < S_j(P)$ and returns HISON if $S_j(T) > S_j(P)$. If $S_j(T) = S_j(P)$ then all m keys are equal and returns a special value to indicate.

The keys of all nodes in the subtree of any node say P, in a m-d tree are known by P's position in the tree to be bounded by certain values. For example, if P is in the HISON subtree of Q and DISC (Q) is $\mathring{j}$, then all nodes in P are $\mathring{j}$-greater than Q; so for any R in HISON (P),

$K_j(R) \geq K_j(P)$. To use this information in the algorithms, a bounded array is defined to hold this information. If B is a bounded array associated with node P, then B has (2 m) entries B(o),.... B(2m-1).

If T is a descendant of P, then it is true for all integers $\mathring{j} \in [0, k-1]$ that $B(2j) \leq K_j(Q) \leq B(2j+1)$. Bounded array B can be initialized with the limit values of the tree.

The algorithms for various types of searches based on this data structure are described in Chapter III. For implementing these algorithms, a small Experimental Data Base of Jawaharlal Nehru University students is developed. This is described in section 2.6.

2.6 Description of Experimented Data Base

The Jawaharlal Nehru University, constituted under Jawaharlal Nehru University Act, 1966 (53 of 1966) came into existence in 1969. The University has identified and is concentrating upon some major academic programmes, which are also of relevance to national progress and development. A SCHOOL has been visualised as a community of scholars from disciplines which are linked with each other organically in terms of their subject matter and methodology as well as in terms of problem areas. Each School will be made up of a number of Centres which constitute the task forces operating within the broad framework of a School. A Centre has been defined as a community of scholars, irrespective of their disciplines engaged in clearly identified inter-disciplinary programmes of research and teaching.

Presently, there are 7 Schools under which about 25 Centres are in operation. The various programmes of study offered by these centres are:

1. Ph.D.
2. M.Phil
3. Five Year integrated Programme leading to M.A.
4. Two year programme leading to M.A./M.Sc.
5. Pre-Degree Diploma
6. Part-time programmes.

The student community in the university has increased considerably in the recent years, presently touching 2500. Along with this, various facilities like accommodation and financial assistance etc. were also increased. Presently the university is having 7 hostels for bachelor students and a considerable number of small quarters for married students. With this vast increase in student community and the various facilities, it has become quite difficult to manage the various administrative problems effectively.

To utilize the various facilities available in this university and to provide the maximum facilities to the students to pursue their academics effectively, the university needed an information system, from which the information can be retrieved quickly. This is the major motive behind the Experimental Data Base which is developed.

Here, each student is identified hierarchically by his/her -

(a) School

(b) Centre

(c) Programme of Study

(d) Area of Study

(e) Roll no.

Each of these fields is expressed as two-character numeric-string and is a secondary key.

All these secondary keys combined gives the primary key which is unique for each student. The advantage of defining the primary key as above is that such a primary key already contains some information about the student, which may lead to reduction of record length. This type of key structure is also advantageous for the region searches such as:

(a) Find the name of the students in a particular Area of Study

(b) Find the date of birth of all students residing in Sutlej Hostel

(c) Find the name of the students getting UGC fellowship in the School of Computer and Systems Sciences.

Description of Records

The record which is 80 character record, consists of two parts: (1) Control Information, (2) Personal Information.

Control Information: This consists of 5 fields:

(a) KEYS, (b) RSKEY, (c) LSKEY, (4) PRKEY, (5) DISC

(a) KEYS: This is a unique numeric character string and identifies a particular node in the tree structure.

(b) RSKEY: This is a unique numeric character string which identifies the Right son of the node under consideration in the tree structure.
This field contains the unique numeric character string "9999999999" if the node under consideration does not have the right son.

(c) LSKEY: This is a unique numeric character string which identifies the left son of the node under consideration.
This field contains the unique character string "9999999999", if the node does not have the left son.

(d) PRKEY: This is a unique character string which identifies the parent of the node under consideration in the tree structure. This field contains the string "99....99" if the node doesn't have parent which is possible only in the case of the root of the tree.

(e) DISC: This is a two-character (numeric) field which will contain the discriminator. The first character is used for Delete Indicator. If this character is "9", the Delete Indicator is treated as on otherwise off.

The fields KEYS, RSKEY, LSKEY, PRKEY are 10 character length. Thus the control information consists of 42 characters.

Personnel Information: The various fields under this are:

(a) Name, (b) Date of Birth, (c) State of Domicile, (d) Year of Joining in the University, (e) Grade point

(a) Name: Which is 10 character alphabetic character string and is name of the node under consideration in the tree structure.

(b) Date of Birth: Which is a 6 character string and denotes the date of birth of the node under consideration in the tree structure i.e. day, month, year i.e. dd mm yy.

(c) State of Domicile: Which is a 2 char string which denotes state of domicile of the node under consideration in the tree structure.

(d) Year of Joining in the University: Which is a 2 character string and denotes the year of joining.

(e) Grade Point: Which is an array of 10 X1 and denotes the average grade point the node was awarded during that semester i.e. the first character denotes the average grade point in the 1st semester, the second character denotes the avg. grade point in the 2nd semester etc. till the 10th semester which is the maximum period allowed generally. If the node is studying in 3rd semester then all the fields from 3rd onwards contains spaces, indicating yet to complete the remaining semesters.

KEYS	RSKEY	LSKEY	PRKEY	DISC	NAME	DATE OF BIRTH	STATE	YEAR OF JOIN-ING	GRADES

Fig. 2.1 Record Description

This Data Base is developed for testing the various searching algorithms based on the multidimensional Binary Tree structure. These algorithms will be discussed in Chapter III.

****

CHAPTER III

MULTIDIMENSIONAL BINARY SEARCH ALGORITHMS

In the previous chapter, the Multidimensional Binary Tree Structure and the Experimental Data Base are described. The Multidimensional Binary Tree Structure can be constructed by repeated insertion of the nodes. The algorithm for insertion of nodes is described in section 3.1. This chapter also describes different types of searching algorithms based on Multidimensional Binary Tree structure. The various types of searches possible on Multidimensional Binary Tree structure are:

(1) Exact Match Search

(2) Region or Range Search

(3) Partial Match Search

3.1 Insertion of node in the Tree Structure

For inserting a new node into the tree structure, this algorithm starts at the root and searches down the tree for the appropriate position of this node by comparing the keys of each node encountered with that of the given node. The input to this algorithm is ROOT, which is the root of the tree structure and the node P to be inserted.

<u>Algorithm:</u>

Procedure Insert (ROOT, P)

Comment The input to this algorithm is root of the tree structure and the node P to be inserted.

The tree structure will be searched for an appropriate position for inserting P and if the search is successful and the tree does not already contain a node with equal value then P is inserted and "SUCCESS" is returned. If there is a node in the tree structure with keys equal then "DUPLICATE NODE" is returned and P will not be inserted.

```
begin
   IF    ROOT  = Λ  then
      begin
         set   ROOT =  P
               LOSON(P) = Λ
               HISON(P) = Λ
               PARENT(P)= Λ
               DISC (P) = 0
               display  "NULL TREE, CREATION STARTED"
               stop
      end
   begin
         search indicator = "SEARCH"
         Q =  ROOT
         DO   until search indicator = "SUCCESS"
            set J  =  DISC(Q)
            begin
```

```
set S_j(Q) = K_j(Q) K_j+1(Q).... K_m-1(Q) K_0(Q)....
                                  K_j-1(Q)
    S_j(P) = K_j(P) K_j+1(P).... K_m-1(P) K_0(P)....
                                  K_j-1(P)

end
IF    S_j(Q) = S_j(P) then
      begin
        display "DUPLICATE NODE"
        search indicator = "SUCCESS"
      end
else
      begin
        IF  S_j(Q) > S_j(P) then
            SON(Q) = LOSON (Q)
        else
            SON(Q) = HISON (Q)
      end
IF          SON(Q) =Λthen
      begin
            set SON (Q) = P
                LOSON(P)= Λ
                HISON(P)= Λ
                PARENT(P)=Q
                DISC(P) = DISC(Q)+1 mod on
                search indicator = "SUCCESS"
                display search indicator.
        end
  else
        set Q = SON(Q)
      end
end
Stop.
```

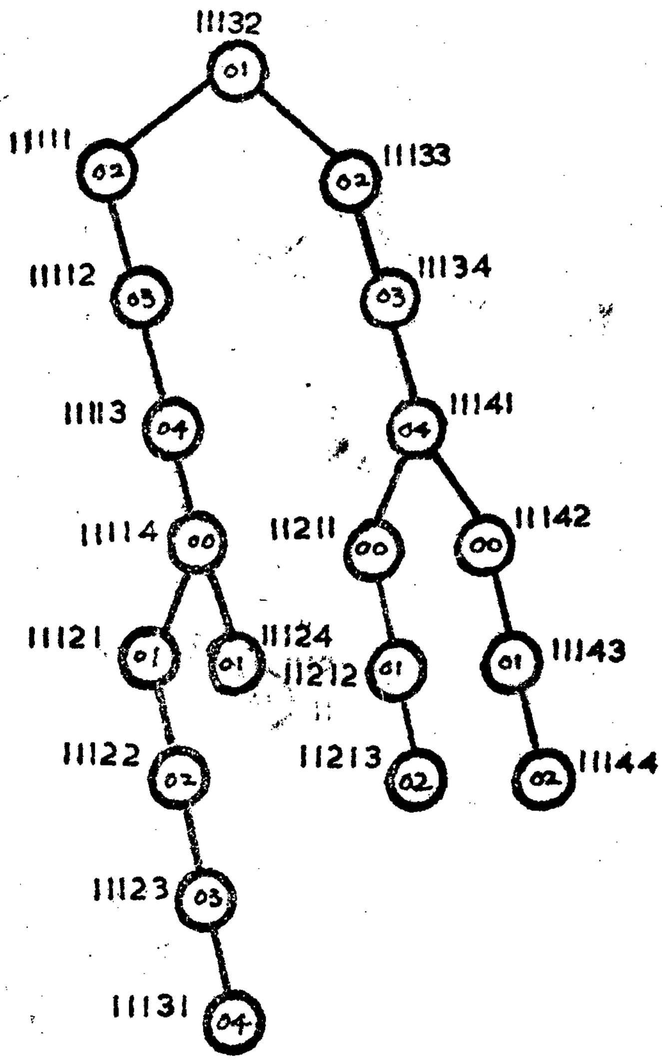

Fig. 3.1 Binary Tree Structure with 19 Nodes
"TREE 1" (Number in circle indicates DISCRIMINENT)

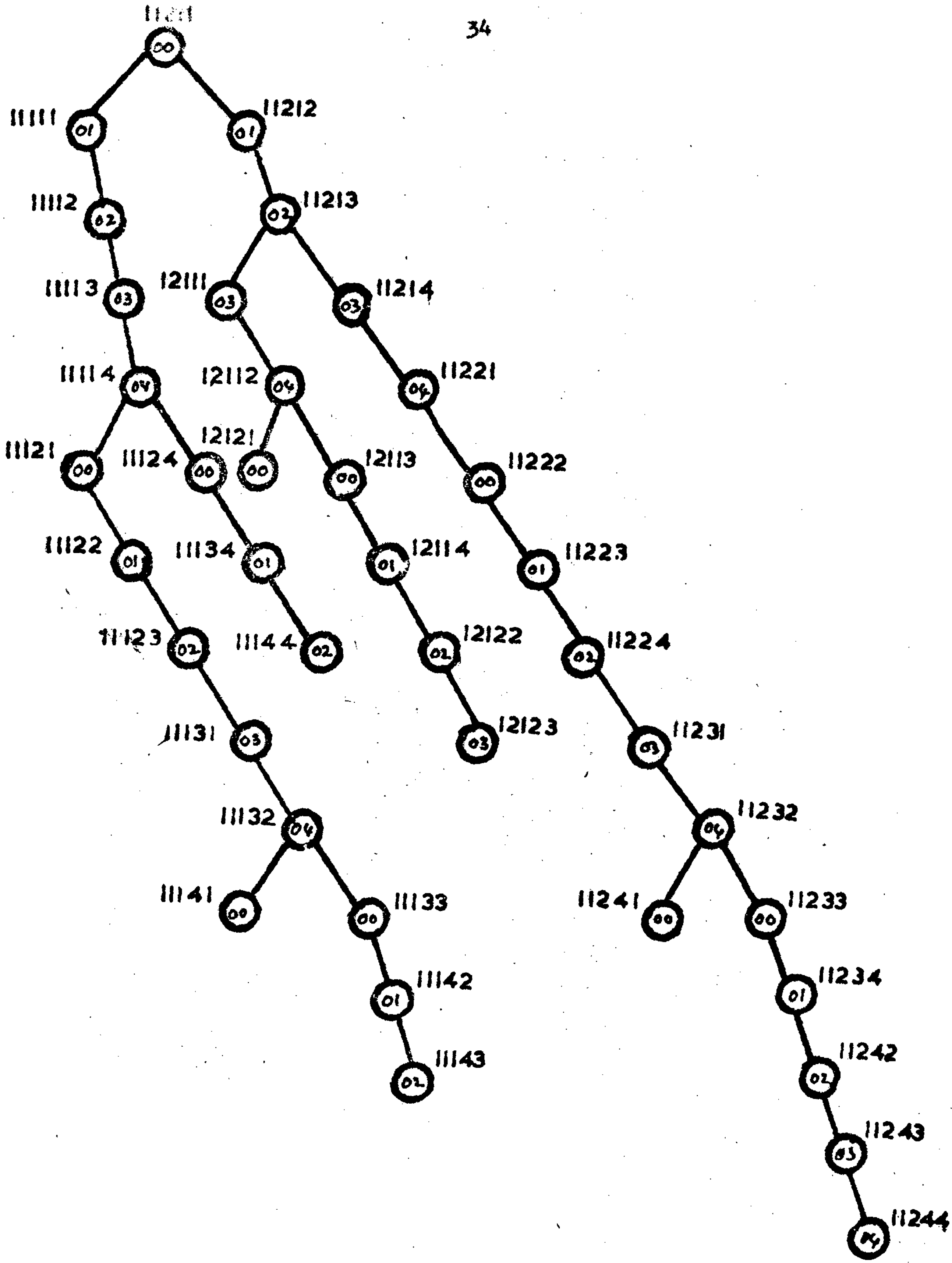

Fig. 3.2 Binary Tree Structure with 39 Nodes
"TREE 2" (Number in circle indicates DISCRIMINENT)

Description

This input to this algorithm is ROOT and the node P which is to be inserted in the tree.

This algorithm first checks whether the tree with root ROOT is a null tree or not. If it is a null tree, then it will insert the node P as the root of the tree and returns.

If the tree with root as ROOT is not a null tree, it will set Q = ROOT and checks whether the nodes Q and P are matching. If the keys match it will return Q.

If the nodes Q and P doesn't match then it will set SON (Q) = SUCCESSOR (Q,P) depending upon K (Q) and K(P) i.e. let j = DISC (P) then if

$K_j(Q) > K_j(P)$ then SON(Q) = LOSON(Q) otherwise

SON(Q) = HISON(Q)

In the event of equality i.e. $K_j(Q) = K_j(P)$ the successor (Q,P) is chosen on the basis of the super keys $S_j(Q)$, $S_j(P)$ and the search continues along SUCCESSOR (Q,P) (either LOSON or HISON) until it finds the appropriate position to insert the node P.

The multidimensional binary tree structure for 19 and 39 nodes are shown in fig. 3.1 and fig. 3.2. Now let us consider two sets of nodes to be inserted to these trees. The two sets of nodes are:

(1) 22211
22222
22223
22224

(2) 22211
22222
22223
22224
22231
22232
22234

Let us start with the node 22211, which has to be inserted into the tree with root node key equal to 11132. Since, the tree whose root is 11132, is not a null tree the algorithm choses the HISON of the Root 11132 for searching the tree to insert the node 22211 by using the Super Key. This process continues until it reaches the node 11213 whose discriminent function is 1. Since the node 11231 doesn't have HISON, inserts the node 22211 as its HISON, by updating HISON field of the node 11213 and the Parent Key of the node 22211. The total number of comparisons done to insert this node are 7. Then the algorithm considers the second node to be inserted i.e. 22222 and restarts the procedure with ROOT = 11132. This procedure continues till all the nodes are inserted.

The total number of nodes to be visited for inserting the first set of nodes is 34.

The total number of nodes visited for inserting the second set of nodes is 66.

The total number of nodes visited for inserting the first and second sets of nodes in second tree are 32 and 70.

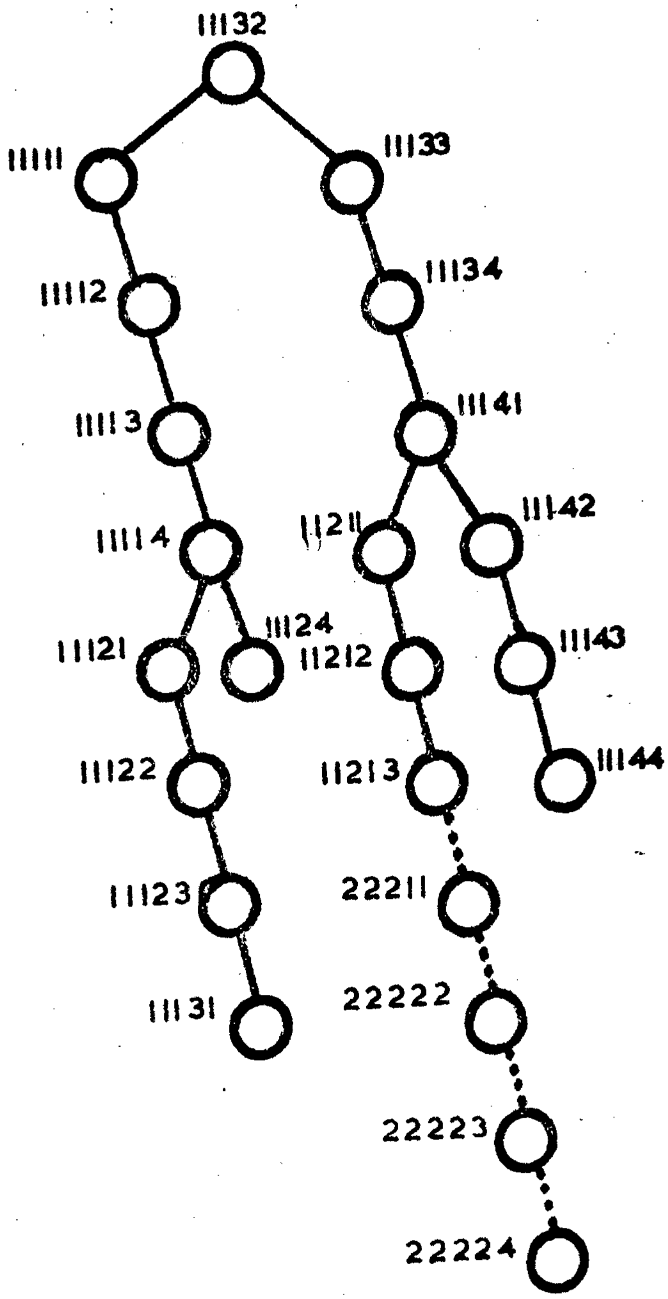

Fig. 3.3 TREE 1 after insertion of first set of nodes

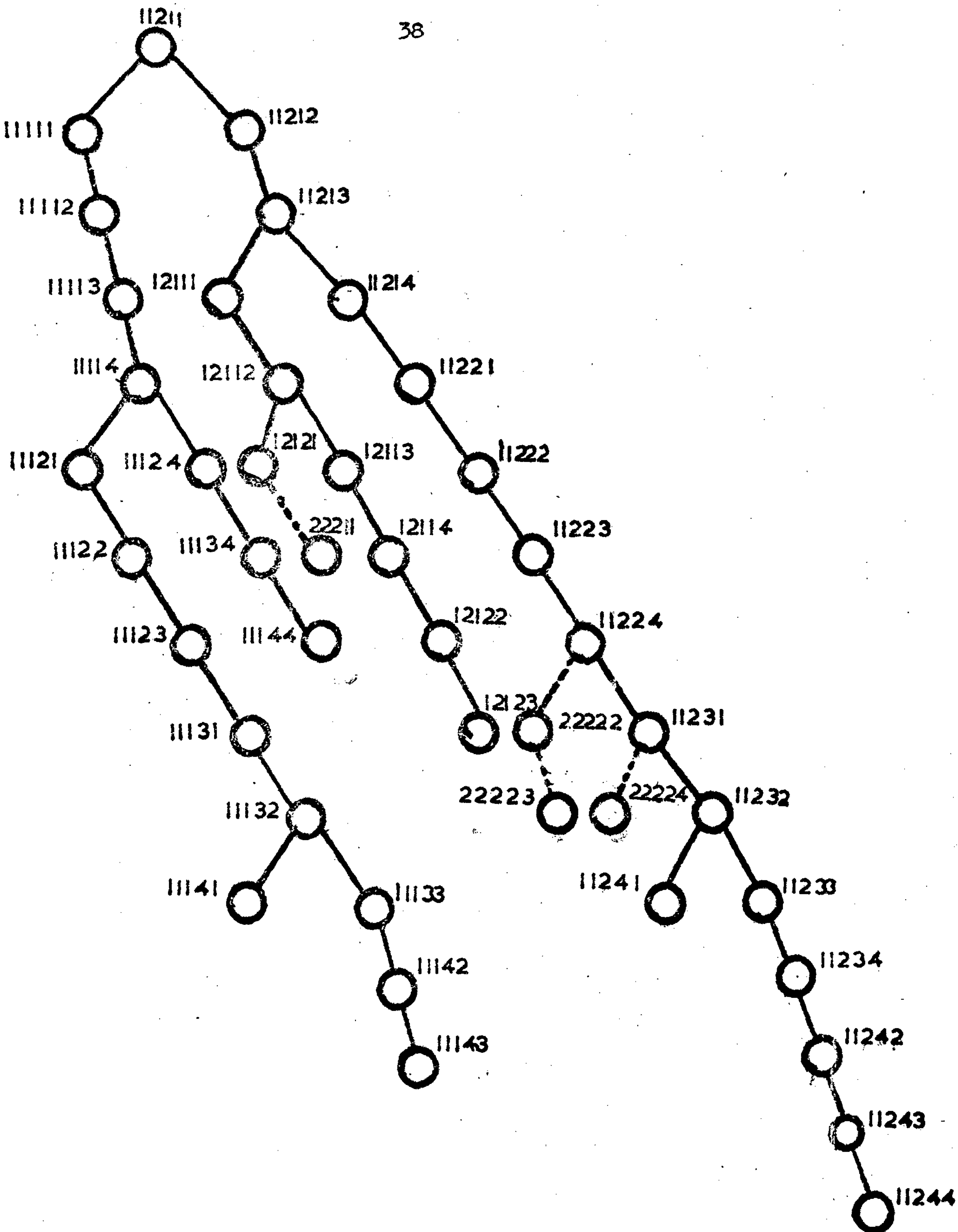

Fig. 3.4 TREE 2 after the insertion of first set of nodes.

3.2 Exact Match Search in the Tree Structure

This Algorithm can be used for finding out whether a particular node P is present in the tree structure or not. The algorithm for this search starts at the root of the tree and proceeds down the tree, going HISON or LOSON, by comparing the desired node's key with the node under consideration, just as in the Insertion algorithm. In this search, the algorithm will either find the node on the way down the tree structure or fall out of the tree if the record is not present. The algorithm returns the data corresponding to the matched node else a message "NOT PRESENT" is returned.

Algorithm:

Procedure SEARCH (ROOT, P)

Comments: This procedure searches the tree for retrieving the node P. If present the node P will be returned else "NOT PRESENT" is returned.

```
begin
   IF  ROOT = Λ  then
       begin
            display "NULL TREE"
            stop
       end
   begin
       Search indicator = "SEARCH"
       Q  =  ROOT
       DO   until search indicator = "SUCCESS"/"UNSUCCESS"
```

```
set  j = DI.C (Q)
begin
        S_j(Q) = K_j(Q) K_j+1(Q) .... K_m(Q) K_0(Q)....
                                       K_j-1(Q)
        S_j(P) = K_j(P) K_j+1(P) .... K_m(P) K_1(P)....
                                       K_j-1(P)
end
IF      S_j(Q) = S_j(P) then
        begin
                display the node Q
                search indicator = "SUCCESS"
        end
else
        begin
          IF  S_j(Q) > S_j(P) then
          SON(Q) = LOSON(Q)
          else
                SON(Q) = HISON (Q)
        end
IF     SON(Q) =   then
        begin
                display "NOT PRESENT"
                search indicater = "UNSUCCESS"
        end
else
        begin
                Q = SON (Q)
        end
    end
  end
end
Stop
```

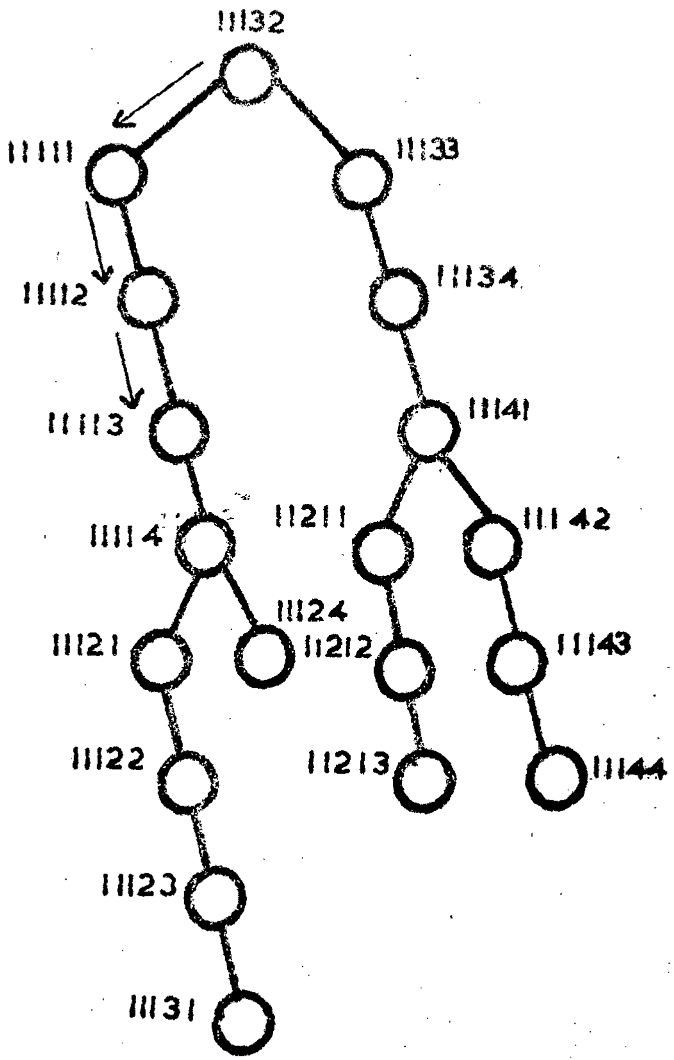

Fig. 3.5 Searching Path for the Node 11113 in TREE 1

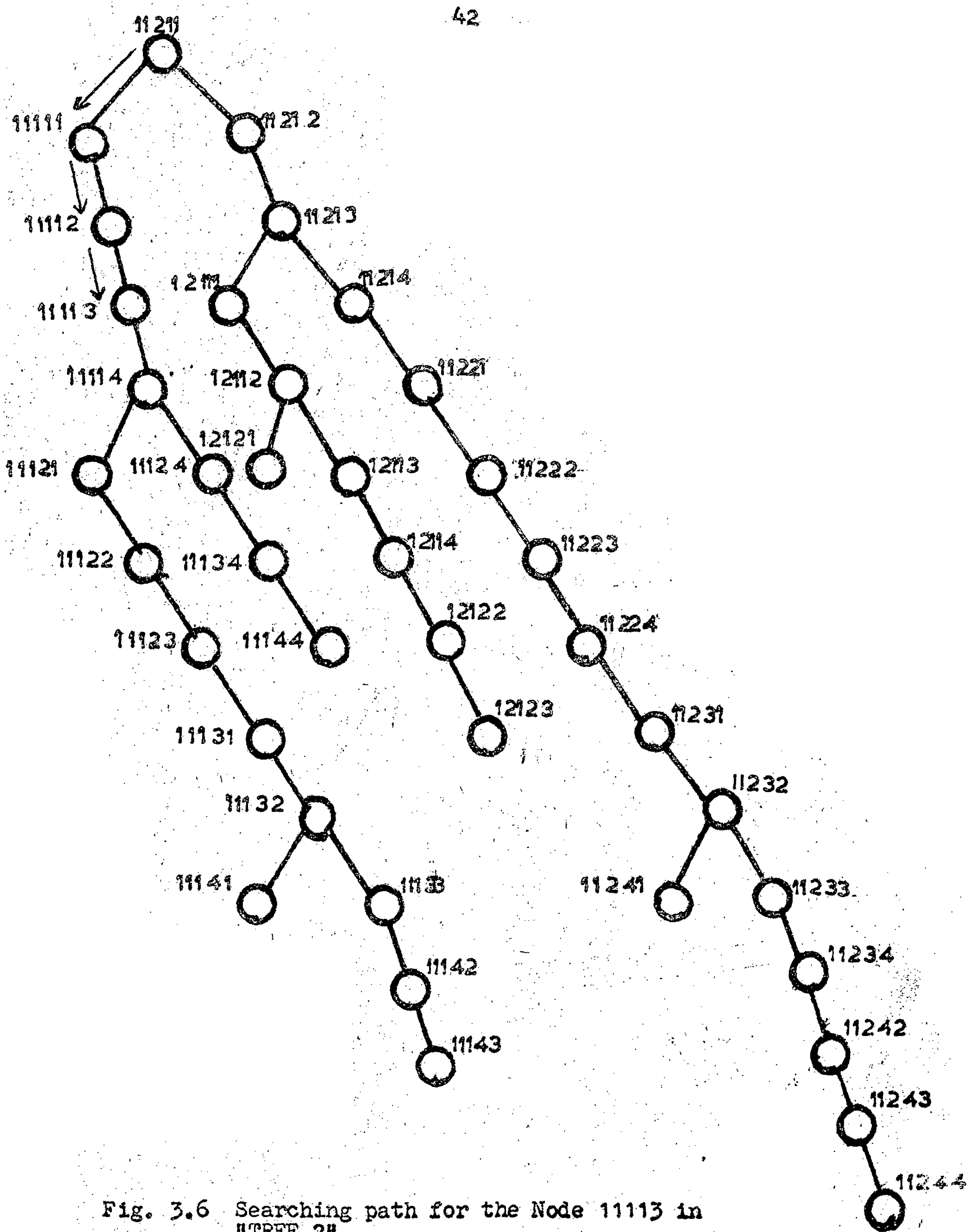

Fig. 3.6 Searching path for the Node 11113 in "TREE 2"

Let us consider two sets of nodes, which have to be retrieved from the tree structures having ROOT, 11132 and 11211. The two sets of nodes are:

(a)		(b)	
	11113		11113
	11114		11114
	11121		11121
	11122		11122
			11123
			11212
			11131

The algorithm starts with the node 11113 which has to be retrieved from the tree structure, whose root is 11132. Since this tree is not a null tree, the algorithm choses the LOSON of the root 11132 for searching by using the Super Key. The searching continues until it finds a node having the key 11113 or a node which does not have any sons. In this case, it retrieves the node having the key 11113 successfully, since it is present in the tree. The total number of comparisons made in this case is 4. The algorithm continues with the second node, if any, to be retrieved and it continues till all nodes, which are asked to retrieve, are retrieved successfully if present or unsuccessful if not present.

The total number of nodes visited for retrieving first set of nodes is 22. In the case of second set of nodes this number is 45.

The number of nodes visited for retrieving the first and second set of nodes from the tree structure having root 1121 are 22 and 49.

33 Deletion of a node from the Tree Structure

It is possible to delete the root from multidimensional binary tree although it is rather expensive to do so. If the root, say P, to be deleted has no subtrees then the resulting tree is the null tree. If P does have descendant then the root should be replaced with one of those descendants, say Q, that will retain the order imposed by P. That is, all nodes in the HISON subtree of P will be in the HISON subtree of Q and likewise for the LOSON subtrees. Reorganisation of the subtrees of P, for each deletion is expensive. To avoid this, the algorithm is modified by introducing an indicator in each node. This indicator, which is called delete indicator is "on" for deleted nodes and "off" for live nodes. Periodically the tree structure can be reorganised for physical deletion of these nodes.

Algorithm

Procedure DELETE (ROOT, P)

Comment: The input to this algorithm is ROOT and the node P which is to be deleted. This algorithm updates the delete indicator of the node P, if P is present in the tree structure. Otherwise it simply returns "NOT PRESENT". If the node is already delted then returns "ALREADY DELETED".

```
begin

    IF  ROOT  =  Λ  then

        begin

            display "NULL TREE"

            stop

        end

      begin

         Search indicator  =  "SEARCH"

         Q   =   ROOT

         DO      until search indicator  = "SUCCESS"/
                                           "UNSUCCESS"

                 set  j  =  DISC (Q)
```

contd....

```
                begin
                   Sj(Q) = Kj(Q) Kj+1(Q) ... Km-1(Q)
                            Ko(Q)....Kj-1(Q)
                   Sj(P) = Kj(P) Kj+1(P)....Km-1(P)
                            Ko(P)....Kj-1(P)
                end
        IF      Sj(Q) = Sj(P) then
                begin
                 IF delete indicator = .TRUE.  then
                    begin
                      display "ALREADY DELETED"
                      search indicator = "SUCCESS"
                    end
                 else
                    begin
                      delete indicator = .TRUE.
                      search indicator ="SUCCESS"
                      display search indicator
                    end
        else
                 IF Sj(Q) > Sj(P) then
                    SON(Q) = LOSON(Q)
                 else
                    SON(Q) = HISON(Q)
         IF         SON(Q) =  Λ  then
                    begin
                      display "NOT PRESENT"
                      search indicator = "UNSUCCESS"
                    end
         else     begin
                      Q = SON(Q)
                  end
      end
 end
end
Stop.
```

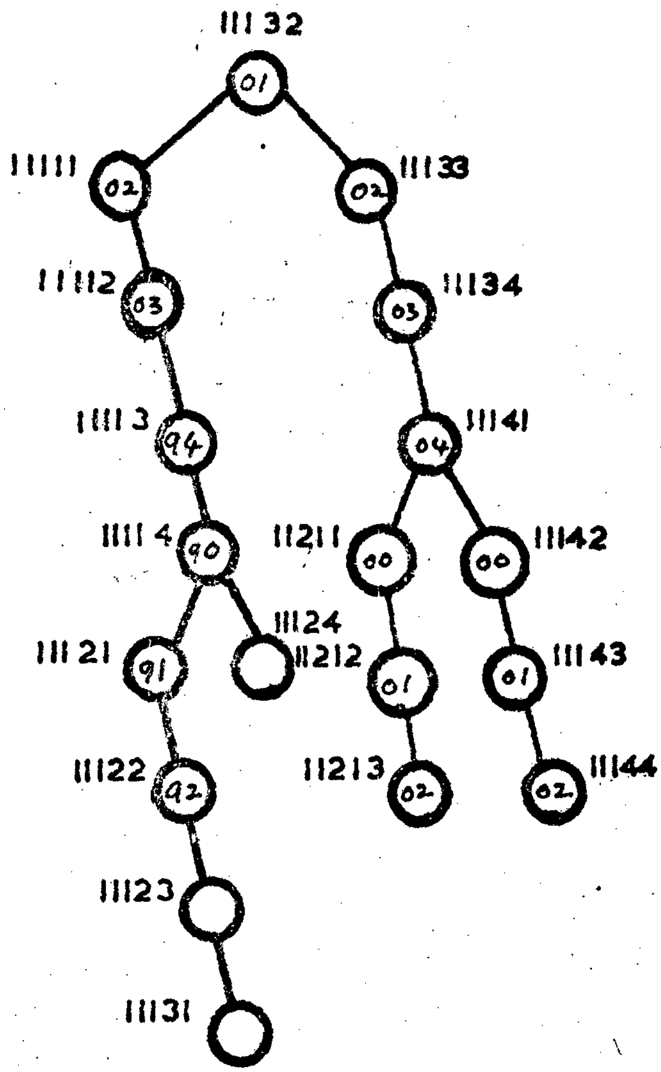

Fig. 3.7 Tree 1 after deletion of first set of nodes

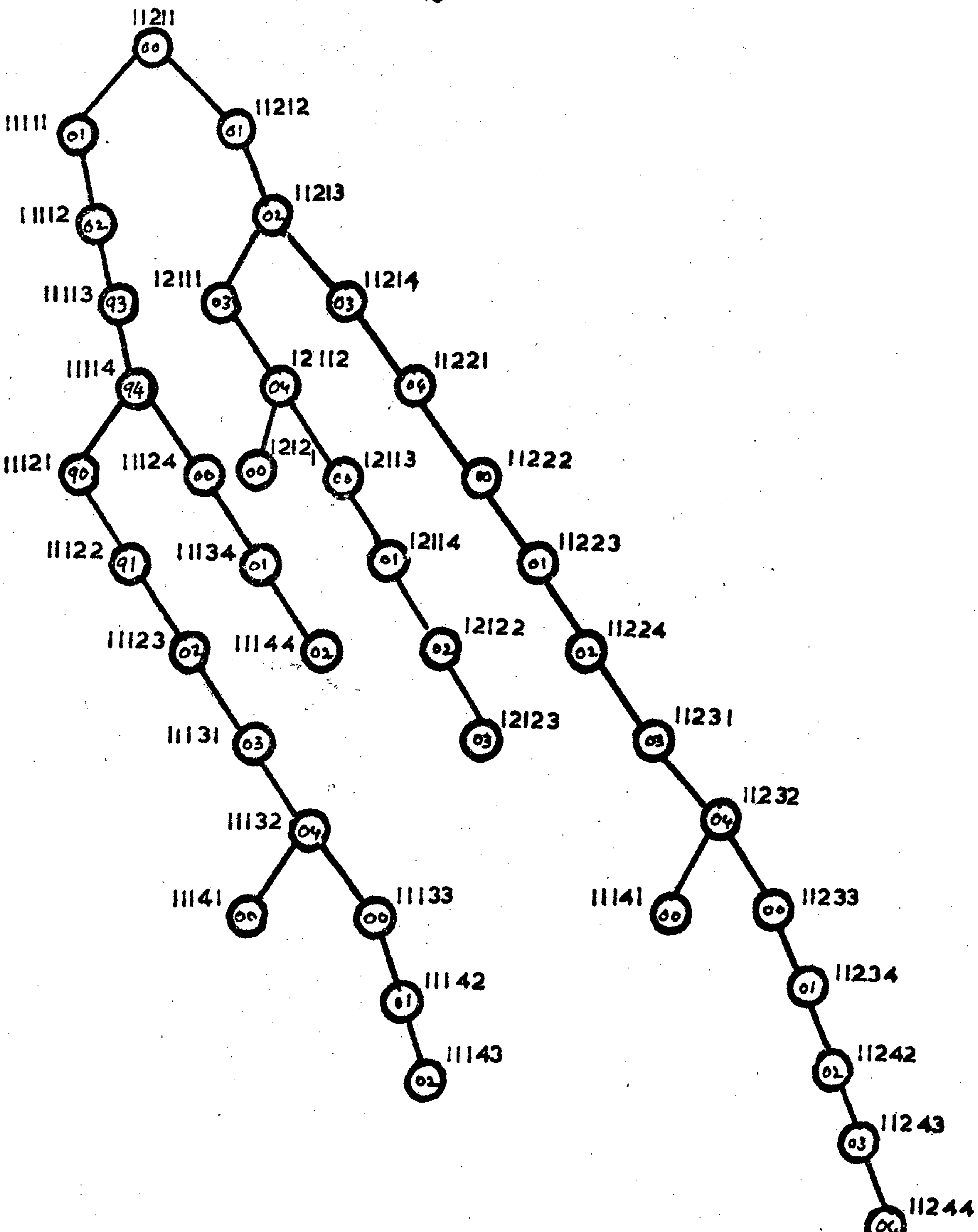

Fig. 3.8 TREE 2 after deletion of first set of Node.

Description

The two sets of nodes considered for Search algorithms are considered for the Deletion algorithms.

This algorithm starts with the node 11113 which has to be deleted from the tree structure whose root node is 11132. Since this tree is not a null tree, the algorithm choses the LOSON of the root 11132 for searching by using the Super Key. The searching continues until it finds a node having the 11113 or a node which does not have any sons. In the first case it updates the first character of the Discriminant 12, which is used as the delete indicator. If this character is "9", the delete indicator is "on" otherwise "off". The total number of nodes visited for the two sets nodes, which are to be deleted, are 22 and 45.

The total number of nodes visited made for the two sets of nodes, which are to be deleted from the tree structure having the root 11211, are 22 and 49.

3.4 Region Searches in the Tree Structure

This recursive algorithm can be used for retrieving all the nodes which lie in the region specified by the region boundaries i.e. all the nodes having a group of common charecteristics. Any subset of the set of valid nodes can be specified in this query and it is the most general intersection query possible. The input to this algorithm is P which is the root of the tree structure, bounds array which specifies the bounds and region boundaries. This algorithm assumes the existence of the procedures IN-REGION, BOUNDS-INTERSECT-REGION and FOUND which are also given.

Algorithm:

Procedure REGION SEARCH (P, B, BR)

Comment: The input to this recursive procedure is the node P in the tree structure, B the bounds array for the nodes in LOSON and HISON subtrees and BR the limits of the region under search. The output will be all the nodes, in the tree structure, which lie in the region specified if the region interests the hyper-rectangle defined by the bounds array B. Otherwise "REGION NOT PRESENT" is returned. This procedure assumes the existence of the boolean procedure IN-REGION, boolean procedure BOUNDS-INTERSECT-REGION and procedure FOUND.

```
begin
    IF   IN-REGION (P, BR) = TRUE then
           begin
               CALL FOUND (P)
           end
    begin
         set BL = B
             BH = B
              j = DISC(P)
    end
```

```
        Comment:  j is the dimension of the bounds to be
                  changed
begin
        set
                  BL_{2j+1}(P)  =  K_j(P)
                  BH_{2j} (P)   =  K_j(P)
        Comment   Here K_j(P) is j - upperbound of the nodes
                  in the LOSON subtree and j-lower bound
                  of the nodes in the HISON subtree.
end.
begin
    IF LOSON(P) ≠ Λ  then
        If BOUNDS-INTERSECT-REGION (BL, BR) =  TRUE. then
           begin
               call REGION SEARCH (LOSON(P), BL, BR)
           end
else
begin
    IF HISON (P) ≠ Λ  then
        IF  BOUNDS-INTERSECT-REGION (BH, BR)= TRUE  then
            begin
                call REGION SEARCH (HISON(P), BH, BR)
            end
    end
end
Stop.
```

Procedure IN-REGION

```
boolean procedure IN-REGION (P,BR)
 begin
   comment  returns .TRUE. if  P is in the hyper-
            rectangle defined by BR
   for   I = 0 step 1 until m-1 do
   begin
         if K_I(P) < BR (2.I) then
            return .FALSE.
         if K_I(P) > BR (2.I+1) then
            return .FALSE.
   end
            return .TRUE.
end
```

BOUNDS-INTERSECT-REGION

```
boolean procedure BOUNDS-INTERSECT-REGION (B,BR)
 begin
   comment  returns .TRUE. if the hyper-rectangle
            defined by bounds array B intersects
            the hyperrectangle defined by BR.
   for  I = 0 step 2 until 2.(m-1) do
   begin

        if B(I) > BR (I+1) then
           return.FALSE.
```

```
            if B(I-1) ∠ BR (I) then
                return .FALSE.
        end
            return .TRUE.
end.
```

FOUND

```
    Procedure   FOUND (P)
begin
      Comment   This procedure will be invoked whenever the
                node P is in the region BR.  This procedure
                displays the node P.
              display  P
end.
```

Description

The input to this recursive algorithm is a node P, generally the ROOT of the tree for which bounds array B is known, and the lower and higher limits ~~LOBND & HIBND~~ array BR, of the region under search. This algorithm passes the node P to the boolean procedure INRNG which returns true if P is contained in the region specified.

If P is not contained in the region, then the boundary of P is copied into the bounds arrays of its LOSON and HISON and then improves these bounds arrays as follows:

Let j = DISC (P)

$$\text{Then} \quad BL_{2j+1}(P) = K_j(P)$$
$$BH_{2j}(P) = K_j(P)$$

That is, $K_j(P)$ is a J-upper bound of the nodes in the LOSON subtree and j-lower bound of nodes in the HISON subtree.

If the LOSON of the node P exists then this algorithm tests whether the region intersects the hyperrectangle described by the bounds array of LOSON of P. If it intersects then the HISON of P and its bounds array BH are pushed into the stack and the search continues along the LOSON of P.

If the LOSON (P) does not exist or the region does not intersects the hyperrectangle described by the bound array $B\ell$, it starts searching along the HISON of the node, if it exists.

This process continues until it finds a node Q which is contained in the region or a node R which does not have LOSON or HISON, or a node which the region does not intersect the hyperrectangle described by its bounds array. In the first case, this algorithm calls the procedure FOUND which will report all the nodes in the subtree which are contained in the region. In the second case this algorithm checks whether the stack contains any nodes. If it contains then that node will be poped up and the search continues. Otherwise it will stop.

This algorithm uses the bounds stored at each node of the tree to determine whether it is possible that any descendants of the node might lie in the region being searched. A subtree is visited by the algorithm iff this possibility exists. Consequently, this algorithm visits as few nodes as possible, given the limited information stored at each node which makes it more faster.

Let us consider two regions whose limits are [11121, 11124] and [11141, 11144]. The objective is to retrieve all the nodes in the tree structures having root 11132 and 11211 and lie in these two regions. The bounds array for both root nodes is [00000, 99999]. Let us first consider the tree structure having root 11132. This algorithm checks whether the tree structure is null. Since it is not null, it calls the voolean procedure IN-REGION with ROOT (11132) and limits of the region [11121, 11124] which are LOREG & HIREG as parameters. IN-REGION returns the value .FALSE. since ROOT is not present in the region specified.

Now the algorithm updates the boundaries of the nodes in LOSON subtree and HISON subtree of the ROOT as described in the algorithm. The updated boundaries are [00000, 19999] for LOSON and [10000, 99999] for HISON. Since the ROOT has LOSON it will call the boolean procedure BOUNDS-INTERSECT-REGION with updated boundaries and region [11121, 11124] as parameters. BOUNDS-INTERSECT-REGION returns the value .TRUE. since the region of search intersects the hyperrectangle of boundaries of the LOSON of the ROOT. Now, the HISON and its boundaries are pushed into the stack and the search is continued along the LOSON by assigning LOSON to ROOT.

Now the algorithm again checks whether the ROOT, 11111, is in the region specified by calling the boolean procedure IN-REGION. Since it is not, it updates the boundaries of the nodes in LOSON and HISON subtrees. The updated boundaries for LOSON and HISON subtrees are [00000, 119999] and [11000, 99999].

At this point, the root does not have a LOSON, the search along the LOSON stops. Since it has a HISON, the algorithm calls the BOUNDS-INTERSECT-REGION with boundaries of HISON and the region specified for finding whether to search the HISON subtree or not. Here the search continues along HISON subtree, since the region intersects the hyperrectangle defined by the boundaries of the HISON subtree.

This process continues until it finds, a node in the tree structure, which does not have any sons or a node for which the region does not intersect with the hyperrectangle defined by the boundaries of its LOSON and HISON subtrees and there is no node in the stack for which search has to be made. Whenever it finds a node P which is in the region, it calls the procedure FOUND. This procedure reports all the nodes in the subtree whose root is P and which are present in the region specified. The total number of nodes visited for both region in first and second tree are

This algorithm first searches the LOSON subtrees always and when it cannot proceed along the LOSON subtrees, it starts searching the HISON subtrees by poping up the HISON nodes from the stack. The searching path for this region search is shown in fig. 3.9 for both the tree structures whose roots are 11132 and 11211.

Implementation

The Experimental Data Base described in Chapter II and all these algorithms are implemented on DEC-10 computer system at National Centre for Software Development and Computing Techniques (TIFR), Bombay. The programmes are written in FORTRAN IV language. The source programmes and the results are given in the Appendix.

****

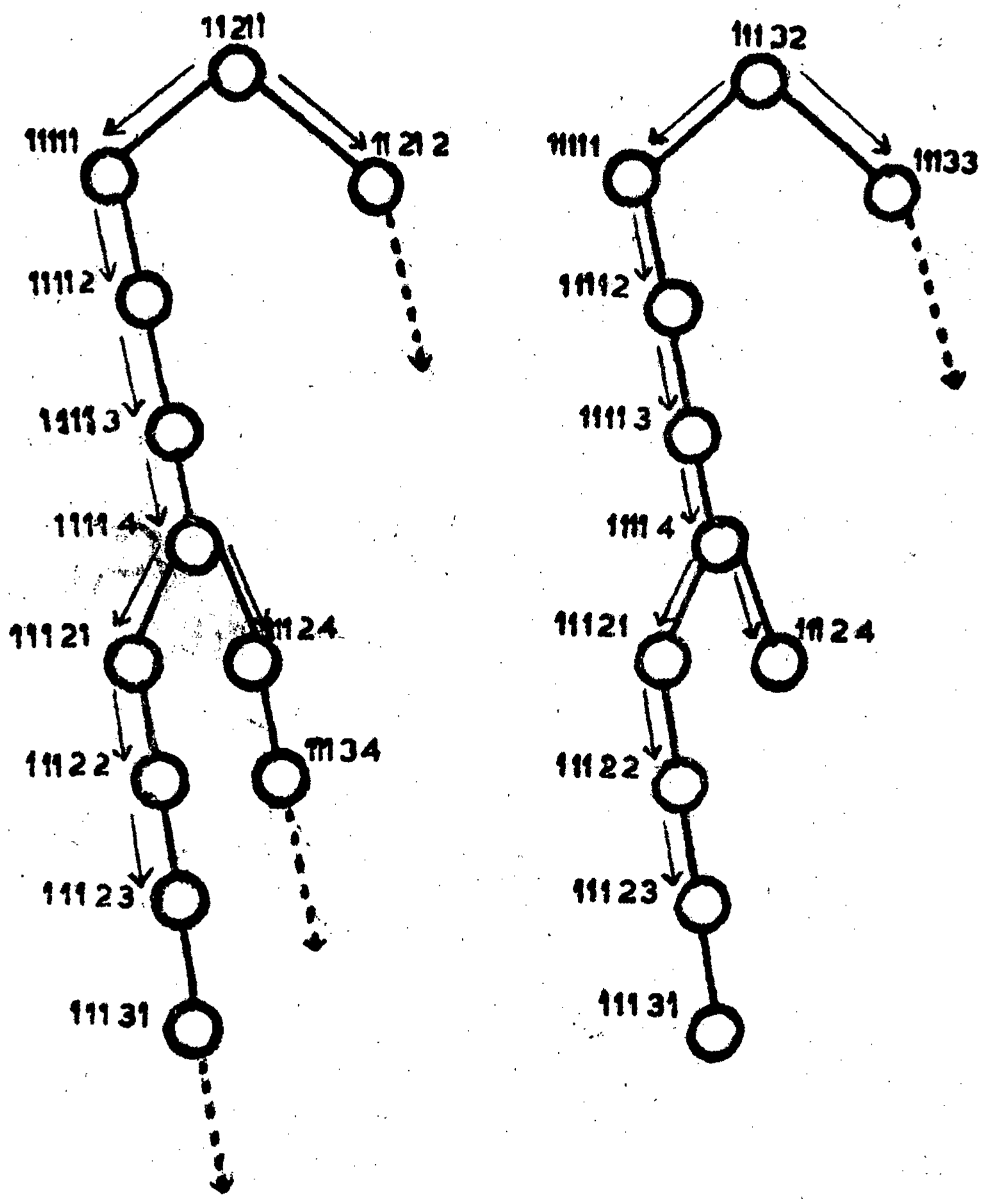

Fig. 3.9 Searching for Region Search of [11121, 11124] in TREE 2 and TREE 1.

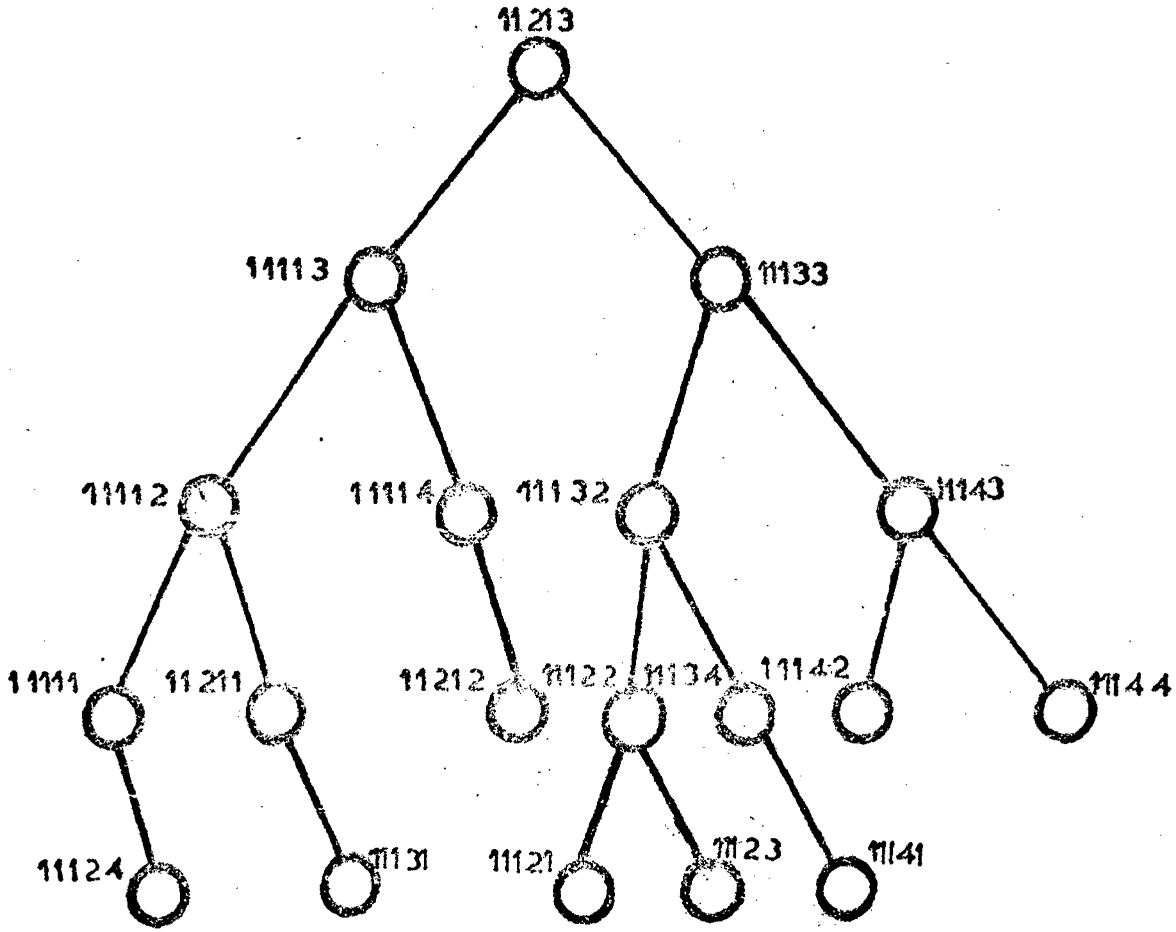

Fig. 3.10 A more balanced Binary Tree Structure.

CHAPTER IV

CONCLUSIONS

The multidimensional Binary Tree structure with 19 and 39 nodes is shown in Fig. 3.1 and Fig. 3.2. Each algorithm is illustrated by taking examples. There are many ways of finding the complexity to these algorithms. One of them may be, the average number of nodes visited for answering a query. The average number of nodes visited by each of the algorithms for answering a query is given. This depends heavily on the tree structure. This becomes minimum when the tree structure is completely balanced, i.e. the ratio of total number of sons in LOSON and HISON subtrees is 1 for any node. If this ratio differs from, considerably, for any node, the entire subtree with root at that node is rebuilt for balancing the tree structure. This data structure becomes inefficient, if it is too unbalanced. This can be avoided by rebuilding the subtrees as explained above.

A more balanced tree structure is shown in fig. 3.9. This is developed by using the tree optimization algorithm given in Appendix. The average number of nodes visited for exact match queries in tree shown in fig. 3.1 and fig. 3.9 are 6 and 3.9 respectively. In the second case, this value is approximately near to the theoretical value which is of O $O(\log N)$, where N is total number of nodes in the tree structure.

In the case of Region Searches the theoretical values of average number of nodes visited is O (log N+F) where F is the number nodes found in the region. The observed value is 9.

Partial match queries described in Chapter II can be implemented by making slight modifications in the Region search programs. The partial match queries are more general and flexible.

Another advantage with this data structure is that the keys can be any data fields but the hierarchy should be strictly maintained. This may help in reducing the record size up to some extent. The presence of the Discriminator reduces the number of comparisons at each node considerably. Backtracking techniques can be applied up to some extent by using the PARENT.

This data structure will be quite useful in Data Base systems where the queries, which involve Region search or Partial Match Search, are frequent. The organization of this data structure is simple. This data structure can be implemented easily. The presence of PARENT at each node will facilitate the application of backtracking techniques.

****

BIBLIOGRAPHY

1 Martin, James — Computer Data Base Organisation, Second edition, Prentice-Hall, Inc., 1977.

2 Date, C.J. — An introduction to Data Base Systems, Addison-Wesley Publishing Company, 1977.

3 Knuth, D.E. — The Art of Computer Programming, Addison-Wesley Publishing Company, 1973.

4 Bentley, J.L. — Multidimensional Binary Search Trees used for Associative Searching, CACM, vol. 19, pp. 509-517, Sept 1975.

5 Bentley, J.L. — Multidimensional Binary Search Trees in Data Base Applications, IEEE Transactions on Software Engg., vol. SE-5, no. 4, July 1979.

6 Kashyap, R.L., Subhas, S.K.C. & Bing. Yao, S. — Analysis of the Multiple-Attribute Tree Data-Base Organization, IEEE Transactions on Software Engg., vol. SE-3, no. 6, 1977.

7 James B. Rothnie Jr. & Thomas Lozano — Attribute Based File Organization in a paged memory Environment, CACM, vol. 17, no. 2, Feb. 1974.

****

APPENDIX

TREE BALANCING ALGORITHM

If any of the subtrees are too unbalanced, that subtree can be rebuilt for balancing. This can be done by TREEOPTIMIZE algorithm.

Algorithm

Procedure TREEOPTIMIZE (A, j)

Comment: The input to this algorithm is snodes, A discriminator j. It returns a pointer to the optimized subtree whose root is j-discriminator.

```
begin
      IF    A  = Λ then
         begin
            display "NULL SET OF NODES"
            stop
         end
         begin
            set P = j - median node in A
         end
         begin
            set  A_L <- {a ∈ A | a is j - less than P }
                 A_H <- {a ∈ A | a is j - greater than P }
         end
```

contd..

```
begin
    set  DISC (P) = j
         M = DISC (P) + 1 mod m
         LOSON(P) ← TREEOPTIMIZE (A_L, M)
         HISON(P) ← TREEOPTIMIZE (A_H, M)
 end
 begin
     display  P
 end
end
Stop.
```

****

```
      INTEGER ROOT, TROOT, RSKEY, PRKEY, DISC
      DIMENSION INAME(19)
      DIMENSION NAME(19)
      DIMENSION KEYS(5), RSKEY(5), LSKEY(5), PRKEY(5)
      DIMENSION ROOT(5), TROOT(5)
      DIMENSION IKEYS(5), IRSKEY(5), ILSKEY(5), IPRKEY(5)
      INTEGER TIME,STIME,ETIME
C
C     "TIME" IS AN INTEGER FUNCTION WHICH RETURNS THE CPU TIME
C     SO FAR AS ITS VALUE IN MILLISECONDS.
C
C
C
      OPEN(UNIT=5,DEVICE='DSK',FILE='TREE.DAT',MODE='ASCII'
     1        ,ACCESS='RANDOM',RECORD SIZE=80,ASSOCIATE
     2        VARIABLE = ID)
C
      READ (1,10, END=820)  NOD, (TROOT(I),I=1,5), KK
   10 FORMAT (I2,1X,5I2,1X,I1)
      IF (KK .EQ. 1) WRITE(3,1000)
 1000 FORMAT(40X,'THE RECORDS REQUESTED ARE:'//)
      IF (KK .EQ. 2) WRITE(3,1020)
 1020 FORMAT(40X,'THE RECORDS CREATED ARE:'//)
      IF (KK .EQ. 3) WRITE(3,1040)
 1040 FORMAT(40X,'THE RECORDS DELETED ARE:'//)
      IF (KK .EQ. 4) WRITE(3,1030)
 1030 FORMAT(40X,'THE RECORDS INSERTED ARE:'//)
C
C
C     NOTE THE STARTING TIME
      STIME = TIME(STIME)
      ITEST=0
      GOTO 15
C
    5 DO 4 I=1,NOD
      TROOT(I)=ROOT(I)
    4 CONTINUE
      ITEST = 1
      GOTO 25
C
   15 DO 20 I=1,NOD,1
      ROOT(I)=TROOT(I)
   20 CONTINUE
C
   25 ID=0
      K1=0
      K2=0
      CALL TEST(ROOT,NOD,K1)
      READ(1,30,END=840) (IKEYS(I),I=1,NOD), (IRSKEY(I),I=1,NOD)
     1        ,(ILSKEY(I),I=1,NOD), (IPRKEY(I),I=1,NOD)
     2        ,IDISC,(INAME(JM),JM=1,19)
   30 FORMAT( 5I2, 5I2, 5I2, 5I2, I2,19(A2))
      IF((K1 .EQ. 5) .AND. (KK .EQ. 1 .OR. KK .EQ. 3))GOTO 400
      IF ((K1 .EQ. 5).AND.(KK .EQ. 2.OR.KK .EQ. 4)) GOTO 410
      IF (K1 .EQ. 0) GOTO 420
      IF ((K1 .GT. 0)  .AND.  (K1 .LT. 5))  GOTO 430
  400 WRITE (3,440)
  440 FORMAT(' NO RECORDS IN THE TREE.  IT IS A NULL TREE.')
      GOTO 800
  410 CALL TEST1(IKEYS,NOD,K2)
```

```
C
C
      SUBROUTINE SEARCH(NOD,ROOT,KEYS,RSKEY,LSKEY,PRKEY,DISC
     1   ,NAME,IKEYS,IRSKEY,ILSKEY,IPRKEY,IDISC,INAME,KK)
      INTEGER ROOT, RSKEY, PRKEY, DISC
      DIMENSION INAME(19)
      DIMENSION NAME(19)
      DIMENSION ROOT(5)
      DIMENSION KEYS(5), RSKEY(5), LSKEY(5), PRKEY(5)
      DIMENSION IKEYS(5), IRSKEY(5), ILSKEY(5), IPRKEY(5)
C
      LIMIT = 99
 3200 CALL MAP(ROOT,ID)
      READ (5'ID,3000)  (KEYS(I),I=1,NOD), (RSKEY(I),I=1,NOD)
     1   ,(LSKEY(I),I=1,NOD), (PRKEY(I),I=1,NOD)
     1   ,DISC, (NAME(JM),JM=1,19)
 3000 FORMAT (4(5I2),I2, 19(A2))
C
      K2= 0
C
      KDISC = DISC+1
      IF (KDISC .GT. 90) KDISC = KDISC - 90
      DO 3041 K2=1,5
      IF (KEYS(KDISC) .NE. IKEYS(KDISC))  GOTO 3043
      KDISC = KDISC+1
      IF (KDISC .GT. NOD) KDISC=KDISC-NOD
  3041 CONTINUE
      GO TO 3070
C
 3043 IF (KEYS(KDISC) .GT. IKEYS(KDISC)) GO TO 3030
 3045 DO 3040 J=1,NOD,1
      ROOT(J)=RSKEY(J)
 3040 CONTINUE
C
      K3=2
      GOTO 3050
C
 3030 DO 3060 J=1,NOD,1
      ROOT(J)=LSKEY(J)
 3060 CONTINUE
C
      K3=1
      GOTO 3050
C
 3070 CALL FOUND (KEYS,LSKEY,RSKEY,PRKEY,DISC,NAME,NOD,KK)
      GOTO 3150
 3050 CALL TEST (ROOT, NOD,K1)
      IF (K1 .NE. NOD)  GOTO 3200
      IF (KK .EQ. 1 .OR. KK .EQ. 3)  GOTO 3160
 3080 IF (K3 .EQ. 1)  GOTO 3090
C
      DO 3100 I=1,NOD
      RSKEY(I)=IKEYS(I)
      IPRKEY(I)=KEYS(I)
      ILSKEY(I)=LIMIT
      IRSKEY(I)=LIMIT
 3100 CONTINUE
      GO TO 3097
C
 3090 DO 3095 I=1,NOD,1
```

```
      IF (K2 .NE. 0) GOTO 800
      DO 510 I=1,NOD
      ROOT(I)=IKEYS(I)
510   CONTINUE
      WRITE(3,500)  (IKEYS(I),I=1,NOD), (IRSKEY(I),I=1,NOD)
     1        ,(ILSKEY(I),I=1,NOD), (IPRKEY(I),I=1,NOD)
     2        ,IDISC, (INAME(JM),JM=1,19)
500    FORMAT(4(5I2),I2,19A2)
      WRITE(12,9998) (ROOT(I),I=1,5)
9998  FORMAT(4X, 'ROOT=',   5I2)
      CALL MAP(IKEYS,ID)
      CALL INSERT(ROOT,IKEYS,IRSKEY,ILSKEY,IPRKEY,IDISC,INAME
     1        ,NOD)
      GOTO 800
420   CALL SEARCH(NOD,ROOT,KEYS,RSKEY,LSKEY,PRKEY,DISC,NAME
     1        ,IKEYS,IRSKEY,ILSKEY,IPRKEY,IDISC,INAME,KK)
      GO TO 800
430   CALL ERR(ROOT,NOD)
800   CONTINUE
      IF (KK .EQ. 2 .AND. ITEST .EQ. 0) GO TO 5
      GOTO 15
820   WRITE(3,810) (TROOT(I),I=1,NOD)
810   FORMAT(4X,' ROOT OF TREE =',5I2)
C
C     NOTE THE TIME AT THE END
      ETIME = TIME(ETIME)
      WRITE(3,830) STIME,ETIME
830   FORMAT(///' STARTING TIME   = ',I6,' MILLISECONDS'/
     1        ' FINISHING TIME  = ',I6,' MILLISECONDS')
      ETIME = ETIME-STIME
      WRITE(3,835) ETIME
835   FORMAT(/' PROCESSING TIME = ',I5,' MILLISECONDS')
      STOP
      END
```

```
      LSKEY(I)=IKEYS(I)
      IPRKEY(I)=KEYS(I)
      ILSKEY(I)=LIMIT
      IRSKEY(I)=LIMIT
 3095 CONTINUE
C
 3097 CONTINUE
      IF (DISC.GT.90) IDIS = DISC-90+1
      IF (DISC.LE.90) IDIS = DISC+1
      IF (IDIS .GE. NOD) GOTO 3110
      IDISC = IDIS
      GOTO 3120
 3110 IDISC=NOD-IDIS
C
 3120 DO  3130  J=1,NOD,1
      ROOT(J)=KEYS(J)
 3130 CONTINUE
C
      CALL MAP(ROOT,ID)
      WRITE(5#ID,3180) (KEYS(I),I=1,NOD), (RSKEY(I),I=1,NOD),
     1         (LSKEY(I),I=1,NOD), (PRKEY(I),I=1,NOD)
     2         ,DISC, (NAME(JM),JM=1,19)
C
      DO 3140 J=1,NOD,1
      ROOT(J)=IKEYS(J)
 3140 CONTINUE
C
      CALL MAP (ROOT,ID)
      WRITE(5#ID,3180) (IKEYS(I),I=1,NOD),(IRSKEY(I),I=1,NOD),
     1         (ILSKEY(I),I=1,NOD),  (IPRKEY(I),I=1,NOD)
     2         ,IDISC, (NAME(JM),JM=1,19)
      GOTO 3165
C
 3160 WRITE (3,3170)
 3170 FORMAT(' FROM SUBROUTINE SEARCH:' /
     1         10X,'SORRY! THE REQUIRED RECORD IS NOT IN THE'
     2         ,'THE  FILE.' /10X,'PRINTING THE INSERTION'
     3         ,'RECORD FOR POSSIBLE ACTION.' //)
C
 3165 WRITE (3,3180) (IKEYS(I),I=1,NOD),  (IRSKEY(I),I=1,NOD)
     1         , (ILSKEY(I),I=1,NOD), (IPRKEY(I),I=1,NOD)
     2         ,IDISC, (NAME(JM),JM=1,19)
C3180 FORMAT( 40X, 4(5I2,2X ), I2, 2X, 19(A2))
 3180          FORMAT(4(5I2),I2,19(A2))
C
 3150 RETURN
      END
```

```
C
C
      SUBROUTINE INSERT(ROOT,IKEYS,IRSKEY,ILSKEY,IPRKEY,IDISC
     1      ,INAME,NOD)
      INTEGER ROOT
      DIMENSION INAME(19)
      DIMENSION ROOT(5),IKEYS(5),IRSKEY(5),ILSKEY(5),IPRKEY(5)
C
      LIMIT = 99
      DO 1000 I=1,NOD,1
      ROOT(I)=IKEYS(I)
      IRSKEY(I)=LIMIT
      ILSKEY(I)=LIMIT
      IPRKEY(I) = LIMIT
 1000 CONTINUE
C
      DISC=0
      CALL MAP(IKEYS,ID)
      WRITE(5'ID,1010) (IKEYS(I),I=1,NOD),(IRSKEY(I),I=1,NOD),
     1      (ILSKEY(I),I=1,NOD), (IPRKEY(I),I=1,NOD)
     2      ,IDISC, (INAME(JM),JM=1,19)
 1010 FORMAT(4(5I2),I2,19(A2))
C
      WRITE (3,1020)  (ROOT(I),I=1,NOD)
 1020 FORMAT(' NULL TREE : TREE STRUCTURE IS JUST STARTED'
     1      /10X, ' ROOT OF THE TREE =  ', 5I2 /)
      RETURN
      END
```

```
C
C
      SUBROUTINE ERR(ROOT,NOD)
        INTEGER ROOT
      DIMENSION ROOT(5)
C
      WRITE (3,2000)  (ROOT(I),I=1,NOD)
 2000 FORMAT (5X, ' ROOT IS NOT SATISFYING THE LIMIT CONDITIONS.
     1PLEASE CHECK IT AGAIN . '  /  20X,  'ROOT = ', 5I2)
      RETURN
      END
```

```
C
C
      SUBROUTINE FOUND(KEYS,LSKEY,RSKEY,PRKEY,DISC, NAME,NOD,KK)
      INTEGER RSKEY, PRKEY, DISC
      DIMENSION NAME(19)
      DIMENSION KEYS(5), RSKEY(5),LSKEY(5), PRKEY(5)
C
      IF (KK.EQ.1) GO TO 4100
      IF (KK.EQ.3) GO TO 4040
      WRITE (3,4010)
      WRITE (3,4020)
      WRITE(3,4000) (KEYS( I),I=1,NOD), (RSKEY(I),I=1,NOD),
     1        (LSKEY(I),I=1,NOD), (PRKEY(I),I=1,NOD)
     2        ,DISC, (NAME(JM),JM=1,19)
 4000 FORMAT( 20X,5I2, 2X,3(5I2,2X),2X, I2, 2X,19(A2))
 4010 FORMAT( 20X, ' THERE IS ALREADY ONE RECORD HAVING THE SAME KEY .
     1        PLEASE CHECK IT . ' //)
 4020 FORMAT(20X, 4X, 'KEYS', 4X, 4X, 'SKEY', 3X, 4X, 'LSKEY', 3X, 4X,
     1        'PRKEY', 3X, 'DISC', 17X, 'NAME'  //)
      GO TO 4030
C
 4040 DISC = DISC+90
      CALL MAP(KEYS,ID)
C
      WRITE(5#ID,4070) (KEYS(I),I=1,NOD),(RSKEY(I),I=1,NOD)
     1        ,(LSKEY(I),I=1,NOD),(PRKEY(I),I=1,NOD)
     2        ,DISC,(NAME(JM),JM=1,19)
 4070 FORMAT(4(5I2),I4,19(A2))
C
 4100 WRITE(3,4020)
      WRITE(3,4000) (KEYS(I),I=1,NOD),(RSKEY(I),I=1,NOD)
     1        ,(LSKEY(I),I=1,NOD),(PRKEY(I),I=1,NOD)
     2        ,DISC,(NAME(JM),JM=1,19)
 4030 RETURN
      END
```

```
C
C
      SUBROUTINE TEST1(IKEYS, NOD,K2)
      DIMENSION IKEYS(5)
      K2=1
      K3=0
      K4=0
      LIMIT = 99
      LLIMIT = 0
C
      DO 3600 I=1,NOD
      IF (IKEYS(I) .EQ. LIMIT) K4=K4+1
      IF (IKEYS(I) .EQ. LLIMIT) K3=K3+1
 3600 CONTINUE
      IF (K3 .GT. 0)  WRITE (3,3610)  (IKEYS(I),I=1,NOD)
      IF (K4 .GT. 0)  WRITE (3,3620)  (IKEYS(I),I=1,NOD)
      IF (.NOT.((K3.GT.0).OR.(K4.GT.0))) K2 = 0
 3610 FORMAT(' SUBROUTINE TEST1:  '/
     1       ' INSERTION KEY EXCEEDS LOWER LIMITS. CHECK'/
     2       ' ROOT= ', 5I2)
 3620 FORMAT(' SUBROUTINE TEST1:  ' //
     1       ' INSERTION KEY EXCEEDS HIGH LIMITS.  CHECK'/
     2       /' ROOT= ', 5I2//)
      RETURN
      END
```

```
C
C
      SUBROUTINE MAP(ROOT,ID)
      INTEGER ROOT
      DIMENSION ROOT(5)
C
      IF ((ROOT(1) .EQ. 0) .OR. (ROOT(1) .GT. 6))  GOTO 4400
      IF ((ROOT(2) .EQ. 0) .OR. (ROOT(2) .GT. 5))  GOTO 4400
      IF ((ROOT(3) .EQ. 0) .OR. (ROOT(3) .GT. 4))  GOTO 4400
      IF ((ROOT(4) .EQ. 0) .OR. (ROOT(4) .GT. 4))  GOTO 4400
      IF ((ROOT(5) .EQ. 0) .OR. (ROOT(5) .GT. 09))  GOTO 4400
C
      ID=ROOT(5)+ROOT(4)*4+ROOT(3)*16+ROOT(2)*32+ROOT(1)*64
      ID = ID-110
      GOTO 4410
4400  WRITE (3,4420)
4420  FORMAT(' SUBROUTINE MAP : KEY IS NOT WITHIN LIMITS.'//)
C
      STOP
C
4410  RETURN
      END
```

```
C
C
      SUBROUTINE TEST(ROOT,NOD,K1)
      INTEGER ROOT
      DIMENSION ROOT(5)
C
      K1 = 0
      LIMIT = 99
C
      DO 3500  J=1,NOD,1
      IF (ROOT(J)  .EQ.  LIMIT )  K1=K1+1
 3500 CONTINUE
C
      RETURN
      END
```

```
C       THIS PROGRAM WAS DEVELOPED BY GSR MURTHY
C       THIS PROGRAM WAS DEVELOPED FOR REGION SEARCH.
C
        COMMON /WORLD/ KEYS(5),LSKEY(5),RSKEY(5),PRKEY(5)
        1        ,NAME(19),STACK(50,5),ROOT(5)
        2        ,LO(5),HI(5),LOREG(5),HIREG(5)
        3        ,NOD,DISC,PTR,ID,STIME,ETIME
C
        INTEGER KEYS,LSKEY,RSKEY,PRKEY,NAME,STACK,ROOT
        1        ,LO,HI,LOREG,HIREG
        2        ,NOD,DISC,PTR,ID,STIME,ETIME
C
        INTEGER I,J,LLO(5),LHI(5),RLO(5),RHI(5)
        INTEGER TEST,MAP,TIME
        LOGICAL INREG,BIREG
C
C       "TIME" IS AN INTEGER FUNCTION WHICH RETURNS THE
C       CPU TIME SO FAR AS ITS VALUE IN MILLISECONDS.
C
        NOD = 5
        PTR = 0
C
        OPEN(UNIT=5,DEVICE='DSK',FILE='TREE.DAT',MODE='ASCII'
        1        ,ACCESS='RANDOM',RECORD SIZE=80,ASSOCIATE
        2        VARIABLE = ID)
C
        DO 3 I=1,NOD
        LO(I) = 0
        HI(I) = 99
  3     CONTINUE
C
        WRITE(12,300)
  300   FORMAT(' GIVE THE ROOT OF THE TREE:',$)
        READ(12,310) (ROOT(I),I=1,5)
  310   FORMAT(5I2)
C
        WRITE(12,320)
  320   FORMAT(' GIVE LOW LIMIT OF THE REGION:',$)
        READ(12,324) (LOREG(I),I=1,NOD)
  324   FORMAT(5(I2,1X))
        WRITE(12,330)
  330   FORMAT(' GIVE HIGH LIMIT OF THE REGION:',$)
        READ(12,334) (HIREG(I),I=1,NOD)
  334   FORMAT(5(I2,X))
C
C       NOTE THE TIME AT THE START OF THE PROCESS.
C
        STIME = TIME(STIME)
C
  5     CONTINUE
        J = TEST(ROOT,NOD)
        IF (J .EQ. NOD) GOTO 50
        IF (J .EQ. 0) GOTO 80
C
  20    WRITE(3,30)
  30    FORMAT(' ROOT NOT WITHIN LIMITS: PLEASE CHECK')
        WRITE (3,40) (ROOT(I),I=1,NOD)
  40    FORMAT(10X, ' ROOT= ', 5(I2))
        GOTO 900
C
```

```
   50     WRITE(3,60)
   60     FORMAT(' NO RECORDS IN THE  TREE.  IT IS A NULL TREE.')
          WRITE(3,70) ( ROOT(I),I=1,NOD)
   70     FORMAT(10X, 'ROOT= ', 5(I2) )
          GOTO 900
C
   80     CONTINUE
          ID = MAP(ROOT,NOD)
          READ(5#ID,95) (KEYS(I),I=1,NOD), (RSKEY(I), I=1,NOD)
          1       ,(LSKEY(I),I=1,NOD), (PRKEY(I),I=1,NOD)
          2       ,DISC, (NAME(J),J=1,19)
   95     FORMAT(4(5I2), I2, 19A2)
C
          IF (.NOT. INREG(KEYS,LOREG,HIREG,NOD)) GO TO 100
C
          CALL FOUND
          GOTO 120
C
  100     DO 90 I=1,NOD
          LLO(I) = LO(I)
          LHI(I) = HI(I)
          RLO(I) = LO(I)
          RHI(I) = HI(I)
   90     CONTINUE
          J=DISC+1
          IF (J .GT. 90) J = J-90
          LHI(J) = KEYS(J)
          RLO(J) = KEYS(J)
          J = TEST(LSKEY,NOD)
          IF (J .EQ. NOD) GO TO 110
          IF (.NOT. BIREG(LLO,LHI,NOD)) GO TO 110
C
          CALL PUSH(RSKEY,NOD)
          CALL PUSH(RHI,NOD)
          CALL PUSH(RLO,NOD)
          CALL PUSH(LHI,NOD)
          CALL PUSH(LLO,NOD)
C
          DO 117 I=1,NOD
          ROOT(I) = LSKEY(I)
          LO(I) = LLO(I)
          HI(I) = LHI(I)
  117     CONTINUE
          GOTO 80
C
  110     J = TEST(RSKEY,NOD)
          IF (J .EQ. NOD) GO TO 120
          IF (.NOT. BIREG(RLO,RHI,NOD)) GO TO 120
C
          DO 125 I=1,NOD
          LO(I) = RLO(I)
          HI(I) = RHI(I)
  125     ROOT(I) = RSKEY(I)
C
          GOTO 80
C
  120     CONTINUE
          IF (PTR .EQ. 0) GO TO 900
C
          CALL POP(LLO,NOD)
```

```
      CALL POP(LHI,NOD)
      CALL POP(RLO,NOD)
      CALL POP(RHI,NOD)
      CALL POP(RSKEY,NOD)
C
      GO TO 110
C
  900 CONTINUE
C
C     NOTE THE TIME AGAIN AT THE END OF THE PROCESS.
C
      ETIME = TIME(ETIME)
      WRITE(3,910) STIME,ETIME
  910 FORMAT(///' STARTING TIME = ',I6,' MILLISECONDS'/
     1         ' FINISHING TIME= ',I6,' MILLISECONDS')
      ETIME = ETIME-STIME
      WRITE(3,915) ETIME
  915 FORMAT(/' PROCESSING TIME  = ',I5,' MILLISECONDS')
      STOP
      END
```

```
C
C
           SUBROUTINE FOUND
C
           COMMON /WORLD/ KEYS(5),LSKEY(5),RSKEY(5),PRKEY(5)
           1        ,NAME(19),STACK(50,5),ROOT(5)
           2        ,LO(5),HI(5),LOREG(5),HIREG(5)
           3        ,NOD,DISC,PTR,ID
C
           INTEGER KEYS,LSKEY,RSKEY,PRKEY,NAME,STACK,ROOT
           1        ,LO,HI,LOREG,HIREG
           2        ,NOD,DISC,PTR,ID
C
           INTEGER COUNT,I,J
           INTEGER MAP
           LOGICAL INREG
C
           COUNT = 0
  5000     CALL OUTPUT
C
           IF (.NOT. INREG(LSKEY,LOREG,HIREG,NOD)) GO TO 5030
           CALL PUSH(RSKEY,NOD)
           COUNT = COUNT+1
C
           DO 5020 I=1,NOD
  5020     ROOT(I)=LSKEY(I)
           GO TO 5050
C
  5030     IF (.NOT. INREG(RSKEY,LOREG,HIREG,NOD)) GO TO 5040
C
           DO 5035 I = 1,NOD
  5035     ROOT(I) = RSKEY(I)
           GO TO 5050
C
  5040     CONTINUE
           IF (COUNT .EQ. 0) RETURN
           COUNT = COUNT-1
           CALL POP(ROOT,NOD)
C
           IF (.NOT. INREG(ROOT,LOREG,HIREG,NOD)) GO TO 5040
C
  5050     CONTINUE
           ID = MAP(ROOT,NOD)
C
           READ(5#ID,5057) (KEYS(I),I=1,NOD),(RSKEY(I),I=1,NOD)
           1        ,(LSKEY(I),I=1,NOD),(PRKEY(I),I=1,NOD)
           2        ,DISC,(NAME(I),I=1,19)
  5057     FORMAT (4(5I2),I2,19A2)
C
           GOTO 5000
C
           END
```

```
C
C
      LOGICAL FUNCTION INREG(KEYS,BL,BH,NOD)
C
      INTEGER NOD,KEYS(NOD),BL(NOD),BH(NOD)
C
C
C   RETURNS TRUE IFF THE GIVEN NODE IS IN THE REGION
C   SPECIED BY BL AND BH.
C
      INTEGER I
C
      INREG = .FALSE.
C
      DO 100 I=1,NOD
      IF (KEYS(I) .LT. BL(I)) RETURN
      IF (KEYS(I) .GT. BH(I)) RETURN
  100 CONTINUE
C
      INREG = .TRUE.
      RETURN
C
      END
```

```
C
C
      LOGICAL FUNCTION BIREG(BML,BMH,N)
C
      COMMON /WORLD/ KEYS(5),LSKEY(5),RSKEY(5),PRKEY(5)
     1       ,NAME(19),STACK(50,5),ROOT(5)
     2       ,LO(5),HI(5),LOREG(5),HIREG(5)
     3       ,NOD,DISC,PTR,ID
C
      INTEGER KEYS,LSKEY,RSKEY,PRKEY,NAME,STACK,ROOT
     1       ,LO,HI,LOREG,HIREG
     2       ,NOD,DISC,PTR,ID
C
      INTEGER N,BML(N),BMH(N)
      INTEGER I
C
      BIREG = .FALSE.
C
      DO 100 I = 1,NOD
      IF (BML(I) .GT. HIREG(I)) RETURN
      IF (BMH(I) .LT. LOREG(I)) RETURN
  100 CONTINUE
C
      BIREG = .TRUE.
      RETURN
C
      END
```

```
C
C
      SUBROUTINE POP(TOP,N)
C
      COMMON /WORLD/ KEYS(5),LSKEY(5),RSKEY(5),PRKEY(5)
     1        ,NAME(19),STACK(50,5),ROOT(5)
     2        ,LO(5),HI(5),LOREG(5),HIREG(5)
     3        ,NOD,DISC,PTR,ID
C
      INTEGER KEYS,LSKEY,RSKEY,PRKEY,NAME,STACK,ROOT
     1        ,LO,HI,LOREG,HIREG
     2        ,NOD,DISC,PTR,ID
C
C
      INTEGER N,TOP(N),I
C
      IF (PTR .GT. 0) GO TO 100
      WRITE(12,10)
   10 FORMAT(' STACK UNDERFLOW')
      STOP
C
  100 DO 110 I=1,N
      TOP(I) = STACK(PTR,I)
  110 CONTINUE
      PTR = PTR-1
C
      RETURN
      END
```

```
      C
      C
                SUBROUTINE PUSH(TOP,N)
      C
      C
                COMMON /WORLD/ KEYS(5),LSKEY(5),RSKEY(5),PRKEY(5)
                1         ,NAME(19),STACK(50,5),ROOT(5)
                2         ,LO(5),HI(5),LOREG(5),HIREG(5)
                3         ,NOD,DISC,PTR,ID
      C
                INTEGER KEYS,LSKEY,RSKEY,PRKEY,NAME,STACK,ROOT
                1         ,LO,HI,LOREG,HIREG
                2         ,NOD,DISC,PTR,ID
      C
                INTEGER N,TOP(N),I
      C
                PTR = PTR+1
                IF (PTR .LE. 50) GO TO 100
                WRITE(12,10)
         10     FORMAT(' STACK OVERFLOW')
                STOP
      C
         100    DO 110 I=1,N
                STACK(PTR,I) = TOP(I)
         110    CONTINUE
      C
                RETURN
                END
```

```
C
C
C
      INTEGER FUNCTION MAP(ROOT,N)
      INTEGER N,ROOT(N)
C
      IF ((ROOT(1) .EQ. 0) .OR. (ROOT(1) .GT. 6))  GOTO 4400
      IF ((ROOT(2) .EQ. 0) .OR. (ROOT(2) .GT. 5))  GOTO 4400
      IF ((ROOT(3) .EQ. 0) .OR. (ROOT(3) .GT. 4))  GOTO 4400
      IF ((ROOT(4) .EQ. 0) .OR. (ROOT(4) .GT. 4))  GOTO 4400
      IF ((ROOT(5) .EQ. 0) .OR. (ROOT(5) .GT. 09))  GOTO 4400
C
      MAP=ROOT(5)+ROOT(4)*4+ROOT(3)*16+ROOT(2)*32+ROOT(1)*64
      MAP = MAP-110
      RETURN
C
4400  WRITE (3,4420)
4420  FORMAT(' SUBROUTINE MAP : KEY IS NOT WITHIN LIMITS.'//)
C
      STOP
C
      END
```

```
C
C
      INTEGER FUNCTION TEST(ROOT,NOD)
      INTEGER NOD,ROOT(NOD),J,K
C
      K = 0
      LIMIT = 99
      DO 3500  J=1,NOD,1
      IF (ROOT(J)  .EQ.  LIMIT )  K=K+1
3500  CONTINUE
      TEST = K
      RETURN
      END
```

```
C
C
      SUBROUTINE OUTPUT
C
      COMMON /WORLD/ KEYS(5),LSKEY(5),RSKEY(5),PRKEY(5)
     1         ,NAME(19),STACK(50,5),ROOT(5)
     2         ,LO(5),HI(5),LOREG(5),HIREG(5)
     3         ,NOD,DISC,PTR,ID
C
      INTEGER KEYS,LSKEY,RSKEY,PRKEY,NAME,STACK,ROOT
     1         ,LO,HI,LOREG,HIREG
     2         ,NOD,DISC,PTR,ID
C
C
      WRITE (3,5500) (KEYS(I),I=1,NOD), (RSKEY(I),I=1,NOD),
     1        (LSKEY(I),I= 1,NOD),  (PRKEY(I),I=1,NOD), DISC,
     2        (NAME(J),J=1,19)
 5500 FORMAT( 5X, 4(5I2,5X,1X), I2,5X,19A2)
      RETURN
      END
```

```
1 1 1 1 1 1 1 1 1 29999999999 1 1 1 3 2 1AMAR SINGH          03025301777543
1 1 1 1 2 1 1 1 1 39999999999 1 1 1 1 1 2AMARJIT SINGH       05095608772543
1 1 1 1 3 1 1 1 1 49999999999 1 1 1 1 2 3AMIRUDDIN           06114915778535
1 1 1 1 4 1 1 1 2 4 1 1 1 2 1 1 1 1 1 3 4AMRITLAL JINDAL     0512590376794376
1 1 1 2 1 1 1 1 2 29999999999 1 1 1 1 4 0ANAND PRAKSH        03116104797
1 1 1 2 2 1 1 1 2 39999999999 1 1 1 2 1 1ANAND MADAN         040254087765435
1 1 1 2 3 1 1 1 3 19999999999 1 1 1 2 2 2AMRIT ANEJA         0801520478516
1 1 1 2 49999999999999999999 1 1 1 1 4 0ANJALI KUMARI       0509580276557431
1 1 1 3 19999999999999999999 1 1 1 2 3 3BHAG RANI ARORA     21065709797
1 1 1 3 2 1 1 1 3 3 1 1 1 1 19999999999 0GSR MURTHY          03025301777543
1 1 1 3 3 1 1 1 3 49999999999 1 1 1 3 2 1ARORA KRISHAN       2912542177752891
1 1 1 3 4 1 1 1 4 19999999999 1 1 1 3 3 2ARORA VIJAY         12115816794289
1 1 1 4 1 1 1 1 4 2 1 1 2 1 1 1 1 1 3 4 3SINHA SARAT         3112531278487
1 1 1 4 2 1 1 1 4 39999999999 1 1 1 4 1 4SEKHAR ANAND        24105818795
1 1 1 4 3 1 1 1 4 49999999999 1 1 1 4 2 0SUBBA RAO A.K.      1506551978589
1 1 1 4 49999999999999999999 1 1 1 4 3 1RAMAM A.V.          14115607755
1 1 2 1 1 1 1 2 1 29999999999 1 1 1 4 1 4VENKATA RATNAM      2507511677593 86
1 1 2 1 2 1 1 2 1 39999999999 1 1 2 1 1 0VERA RABHAVAN       1212612078151
1 1 2 1 39999999999999999999 1 1 2 1 2 1HARBANS SINGH       1902581178352
```

THE RECORDS REQUESTED ARE:

KEYS	SKEY	LSKEY	PRKEY	DISC	NAME	
1 1 1 1 3	9999999999	1 1 1 1 4	1 1 1 1 2	3	AMIRUDDIN	06114915778535
KEYS	SKEY	LSKEY	PRKEY	DISC	NAME	
1 1 1 1 4	1 1 1 2 1	1 1 1 2 4	1 1 1 1 3	4	AMRITLAL JINDAL	0512590376794376
KEYS	SKEY	LSKEY	PRKEY	DISC	NAME	
1 1 1 2 1	9999999999	1 1 1 2 2	1 1 1 1 4	0	ANAND PRAKSH	03116104797
KEYS	SKEY	LSKEY	PRKEY	DISC	NAME	
1 1 1 2 2	9999999999	1 1 1 2 3	1 1 1 2 1	1	ANAND MADAN	0402540877765435

ROOT OF TREE = 1 1 1 3 2

STARTING TIME = 201237 MILLISECONDS
FINISHING TIME = 201837 MILLISECONDS

PROCESSING TIME = 600 MILLISECONDS

```
                    THE RECORDS REQUESTED ARE:

        KEYS        SKEY        LSKEY        PRKEY    DISC                  NAME

   1 1 1 1 3   9999999999   1 1 1 1 4   1 1 1 1 2     3  AMIRUDDIN             06114915778535
        KEYS        SKEY        LSKEY        PRKEY    DISC                  NAME

   1 1 1 1 4    1 1 1 2 1   1 1 1 2 4   1 1 1 1 3     4  AMRITLAL JINDAL       0512590376794376
        KEYS        SKEY        LSKEY        PRKEY    DISC                  NAME

   1 1 1 2 1   9999999999   1 1 1 2 2   1 1 1 1 4     0  ANAND PRAKSH          03116104797
        KEYS        SKEY        LSKEY        PRKEY    DISC                  NAME

   1 1 1 2 2   9999999999   1 1 1 2 3   1 1 1 2 1     1  ANAND MADAN           04025408776 5435
        KEYS        SKEY        LSKEY        PRKEY    DISC                  NAME

   1 1 1 2 3   9999999999   1 1 1 3 1   1 1 1 2 2     2  AMRIT ANEJA           0801520478516
        KEYS        SKEY        LSKEY        PRKEY    DISC                  NAME

   1 1 2 1 2   9999999999   1 1 2 1 3   1 1 2 1 1     0  VERA RABHAVAN         1212612078151
        KEYS        SKEY        LSKEY        PRKEY    DISC                  NAME

   1 1 1 3 1   9999999999  9999999999   1 1 1 2 3     3  BHAG RANI ARORA       21065709797
ROOT OF TREE = 1 1 1 3 2


STARTING TIME   = 197309 MILLISECONDS
FINISHING TIME  = 198448 MILLISECONDS

PROCESSING TIME =  1139 MILLISECONDS
```

THE RECORDS INSERTED ARE:

```
2 2 2 1 199999999999999999999 1 1 2 1 3 2MKS ROY           05045302456789
2 2 2 2 299999999999999999999 2 2 2 1 1 3MN CHAKRAVARTHY   07055608095 4329
2 2 2 2 399999999999999999999 2 2 2 2 2 4AM PILLAI         26125608976543
2 2 2 2 499999999999999999999 2 2 2 2 3 0RN RAO            2104540504087 6
    ROOT OF TREE = 1 1 1 3 2
```

```
STARTING TIME   = 212642 MILLISECONDS
FINISHING TIME  = 213771 MILLISECONDS
```

PROCESSING TIME = 1129 MILLISECONDS

THE RECORDS INSERTED ARE:

```
2 2 2 1 199999999999999999999 1 1 2 1 3 2MRS ROY          05045302456789
2 2 2 2 299999999999999999999 2 2 2 1 1 3MN CHAKRAVARTHY  070556080954329
2 2 2 2 399999999999999999999 2 2 2 2 2 4AM PILLAI        26125608976543
2 2 2 2 499999999999999999999 2 2 2 2 3 0RN RAO           2104540504087 6
2 2 2 3 199999999999999999999 2 2 2 2 3 0LM MENON         150847040867
2 2 2 3 299999999999999999999 2 2 2 3 1 1PR ADAVI         161250040805432
2 2 2 3 499999999999999999999 2 2 2 2 4 1TK JAGANNATH     16055507084786
    ROOT OF TREE = 1 1 1 3 2


STARTING TIME   = 204476 MILLISECONDS
FINISHING TIME  = 206426 MILLISECONDS

PROCESSING TIME = 1950 MILLISECONDS
```

THE RECORDS DELETED ARE!

KEYS	SKEY	LSKEY	PRKEY	DISC		NAME
1 1 1 1 3	1 1 1 1 4	9999999999	1 1 1 1 2	93	AMIRUDDIN	06114915778535
1 1 1 1 4	1 1 1 2 4	1 1 1 2 1	1 1 1 1 3	94	AMRITLAL JINDAL	0512590376794376
1 1 1 2 1	1 1 1 2 2	9999999999	1 1 1 1 4	90	ANAND PRAKSH	03116104797
1 1 1 2 2	1 1 1 2 3	9999999999	1 1 1 2 1	91	ANAND MADAN	040254087765435

ROOT OF TREE = 1 1 1 3 2

STARTING TIME = 289996 MILLISECONDS
FINISHING TIME = 291142 MILLISECONDS

PROCESSING TIME = 1146 MILLISECONDS

```
                    THE RECORDS DELETED ARE!

          KEYS        SKEY        LSKEY        PRKEY    DISC                NAME

        1 1 1 1 3   1 1 1 1 4   9999999999   1 1 1 1 2    93  AMIRUDDIN             06114915778535
          KEYS        SKEY        LSKEY        PRKEY    DISC                NAME

        1 1 1 1 4   1 1 1 2 4    1 1 1 2 1   1 1 1 1 3    94  AMRITLAL JINDAL       05125903767943 76
          KEYS        SKEY        LSKEY        PRKEY    DISC                NAME

        1 1 1 2 1   1 1 1 2 2   9999999999   1 1 1 1 4    90  ANAND PRAKSH          03116104797
          KEYS        SKEY        LSKEY        PRKEY    DISC                NAME

        1 1 1 2 2   1 1 1 2 3   9999999999   1 1 1 2 1    91  ANAND MADAN           040254087765435
          KEYS        SKEY        LSKEY        PRKEY    DISC                NAME

        1 1 1 2 3   1 1 1 3 1   9999999999   1 1 1 2 2    92  AMRIT ANEJA           0801520478516
          KEYS        SKEY        LSKEY        PRKEY    DISC                NAME

        1 1 2 1 2   1 1 2 1 3   9999999999   1 1 2 1 1    90  VERA RABHAVAN         121261207815 1
          KEYS        SKEY        LSKEY        PRKEY    DISC                NAME

        1 1 1 3 1  9999999999   9999999999   1 1 1 2 3    93  BHAG RANI ARORA       21065709797
ROOT OF TREE = 1 1 1 3 2

STARTING TIME  = 280272 MILLISECONDS
FINISHING TIME = 282846 MILLISECONDS

PROCESSING TIME =  2574 MILLISECONDS
```

1 1 1 2 1	1 1 1 2 2	9999999999	1 1 1 1 4	0	ANAND PRAKSH	03116104797
1 1 1 2 2	1 1 1 2 3	9999999999	1 1 1 2 1	1	ANAND MADAN	04025408776543 5
1 1 1 2 3	1 1 1 3 1	9999999999	1 1 1 2 2	2	AMRIT ANEJA	0801520478516
1 1 1 2 4	9999999999	9999999999	1 1 1 1 4	0	ANJALI KUMARI	050958027655743 1

STARTING TIME = 217906 MILLISECONDS
FINISHING TIME= 218778 MILLISECONDS

PROCESSING TIME = 872 MILLISECONDS

1 1 1 4 1	1 1 1 4 2	1 1 2 1 1	1 1 1 3 4	3	SINHA SARAT	3112531278487
1 1 1 4 2	1 1 1 4 3	9999999999	1 1 1 4 1	4	SEKHAR ANAND	24105818795
1 1 1 4 3	1 1 1 4 4	9999999999	1 1 1 4 2	0	SUBBA RAO A.K.	1506551978589
1 1 1 4 4	9999999999	9999999999	1 1 1 4 3	1	RAMAM A.V.	14115607755

STARTING TIME = 222530 MILLISECONDS
FINISHING TIME= 223451 MILLISECONDS

PROCESSING TIME = 921 MILLISECONDS

THE RECORDS CREATED ARE!

```
1 1 2 1 1 Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø ØGSR MURTHY          Ø3Ø253Ø1777543
NULL TREE : TREE STRUCTURE IS JUST STARTED
          ROOT OF THE TREE =   1 1 2 1 1

1 1 1 1 1999999999999999999 1 1 2 1 1 1AMAR SINGH          Ø3Ø253Ø1777543
1 1 1 1 2999999999999999999 1 1 1 1 1 2AMARJIT SINGH       Ø5Ø956Ø8772543
1 1 1 1 3999999999999999999 1 1 1 1 2 3AMIRUDDIN           Ø6114915778535
1 1 1 1 4999999999999999999 1 1 1 1 3 4AMRITLAL JINDAL     Ø51259Ø376794376
1 1 1 2 1999999999999999999 1 1 1 1 4 ØANAND PRAKSH        Ø31161Ø4797
1 1 1 2 2999999999999999999 1 1 1 2 1 1ANAND MADAN         Ø4Ø254Ø87765435
1 1 1 2 3999999999999999999 1 1 1 2 2 2AMRIT AHEJA         Ø8Ø152Ø478516
1 1 1 2 4999999999999999999 1 1 1 1 4 ØANJALI KUMARI       Ø5Ø958Ø276557431
1 1 1 3 1999999999999999999 1 1 1 2 3 3BHAG RANI ARORA     21Ø657Ø9797
1 1 1 3 2999999999999999999 1 1 1 3 1 4ARORA HARBANS       191155 2Ø78453
1 1 1 3 3999999999999999999 1 1 1 3 2 ØARORA KRISHAN       2912542177528 91
1 1 1 3 4999999999999999999 1 1 1 2 4 1ARORA VIJAY         12115816794 2891
1 1 1 4 1999999999999999999 1 1 1 3 2 ØSINHA SARAT         3112531278487
1 1 1 4 2999999999999999999 1 1 1 3 3 1SEKHAR ANAND        241Ø5818795
1 1 1 4 3999999999999999999 1 1 1 4 2 2SUBBA RAO A.K.      15Ø6551978589
1 1 1 4 4999999999999999999 1 1 1 3 4 2RAMAM A.V.          141156Ø7755
1 1 2 1 2999999999999999999 1 1 2 1 1 1VERA RAGHAVAN       121261 2Ø78151
1 1 2 1 3999999999999999999 1 1 2 1 2 2HARBANS SINGH       19Ø2581178352
1 1 2 1 4999999999999999999 1 1 2 1 3 3RAKESH JAIN         121157Ø278867
1 1 2 2 1999999999999999999 1 1 2 1 4 4ARVIND KUMAR        Ø31Ø56Ø878838
1 1 2 2 2999999999999999999 1 1 2 2 1 ØPARVEEN KAUR        Ø5125 8Ø218865
1 1 2 2 3999999999999999999 1 1 2 2 2 1ANJANA DEVI         Ø61Ø59Ø178875
1 1 2 2 4999999999999999999 1 1 2 2 3 2VISHAL KUMAR        151157Ø978862
1 1 2 3 1999999999999999999 1 1 2 2 4 3SUDHANSHU MEHTA     11Ø455Ø37785348
1 1 2 3 2999999999999999999 1 1 2 3 1 4RAKESH SHUKLA       3ØØ75ØØ47785339
1 1 2 3 3999999999999999999 1 1 2 3 2 ØANJAN BAKSHI        31125 1Ø87783867
1 1 2 3 4999999999999999999 1 1 2 3 3 1AVINASH SASEA       251153Ø67797352
1 1 2 4 1999999999999999999 1 1 2 3 2 ØREDDY P.G.K.        191254Ø778583
1 1 2 4 2999999999999999999 1 1 2 3 4 2DEEPAK BANERJI      12Ø455Ø878828
1 1 2 4 3999999999999999999 1 1 2 4 2 3ALOK AIMA           24Ø5522Ø78987
1 1 2 4 4999999999999999999 1 1 2 4 3 4HEMNANI A.K.        12Ø8582178786
1 2 1 1 1999999999999999999 1 1 2 1 3 3VENKAT PAGHAVAN     3112581578694
1 2 1 1 2999999999999999999 1 2 1 1 1 4BRIJESH PATEL       121Ø571978788
1 2 1 1 3999999999999999999 1 2 1 1 2 ØSARDAR A.M.         Ø1Ø2562Ø78458
1 2 1 1 4999999999999999999 1 2 1 1 3 1SHAMEER BEGUM       19Ø8571178765
1 2 1 2 1999999999999999999 1 2 1 1 2 ØRITESH KUMAR        11Ø156Ø878582
1 2 1 2 2999999999999999999 1 2 1 1 4 2AJIT YADAV          131154Ø578938
1 2 1 2 3999999999999999999 1 2 1 2 2 3BENJAMIN            3ØØ455Ø778869
    ROOT OF TREE = 1 1 2 1 1


STARTING TIME    =  65241 MILLISECONDS
FINISHING TIME   =  73577 MILLISECONDS

PROCESSING TIME =  8336 MILLISECONDS
```

```
1 1 1 1 1 1 1 1 1 2999999999 1 1 2 1 1 1AMAR SINGH          03025301777543
1 1 1 1 2 1 1 1 1 3999999999 1 1 1 1 1 2AMARJIT SINGH       05095608772543
1 1 1 1 3 1 1 1 1 4999999999 1 1 1 1 2 3AMIRUDDIN           06114915778535
1 1 1 1 4 1 1 1 2 4 1 1 1 2 1 1 1 1 1 3 4AMRITLAL JINDAL     0512590376794376
1 1 1 2 1 1 1 1 2 2999999999 1 1 1 1 4 0ANAND PRAKSH        0311610497
1 1 1 2 2 1 1 1 2 3999999999 1 1 1 2 1 1ANAND MADAN         040254087765435
1 1 1 2 3 1 1 1 3 1999999999 1 1 1 2 2 2AMRIT ANEJA         0801520478516
1 1 1 2 4 1 1 1 3 4999999999 1 1 1 1 4 0ANJALI KUMARI       0509580276557431
1 1 1 3 1 1 1 1 3 2999999999 1 1 1 2 3 3BHAG RANI ARORA     2106570979 7.
1 1 1 3 2 1 1 1 3 3 1 1 1 4 1 1 1 1 3 1 4ARORA HARBANS       1911552078453
1 1 1 3 3 1 1 1 4 2999999999 1 1 1 3 2 0ARORA KRISHAN       29125421775289 1
1 1 1 3 4 1 1 1 4 4999999999 1 1 1 2 4 1ARORA VIJAY         121158167942891
1 1 1 4 19999999999999999999 1 1 1 3 2 0SINHA SARAT         3112531278487
1 1 1 4 2 1 1 1 4 3999999999 1 1 1 3 3 1SEKHAR ANAND        24105818795
1 1 1 4 39999999999999999999 1 1 1 4 2 2SUBBA RAO A.K.      1506551278589
1 1 1 4 49999999999999999999 1 1 1 3 4 2RAMAN A.V.          14115607755
1 1 2 1 1 1 1 2 1 2 1 1 1 1 1999999999 0GSR MURTHY          03025301777543
1 1 2 1 2 1 1 2 1 3999999999 1 1 2 1 1 1VERA RAGHAVAN       1212612078151
1 1 2 1 3 1 1 2 1 4 1 2 1 1 1 1 1 2 1 2 2HARBANS SINGH       1902581178352
1 1 2 1 4 1 1 2 2 1999999999 1 1 2 1 3 3RAKESH JAIN         1211570278867
1 1 2 2 1 1 1 2 2 2999999999 1 1 2 1 4 4ARVIND KUMAR        0310560878838
1 1 2 2 2 1 1 2 2 3999999999 1 1 2 2 1 0PARVEEN KAUR        0512580218865
1 1 2 2 3 1 1 2 2 4999999999 1 1 2 2 2 1ANJANA DEVI         0610590178875
1 1 2 2 4 1 1 2 3 1999999999 1 1 2 2 3 2VISHAL KUMAR        1511570978862
1 1 2 3 1 1 1 2 3 2999999999 1 1 2 2 4 3SUDHANSHU MEHTA     110455037785348
1 1 2 3 2 1 1 2 3 3 1 1 2 4 1 1 1 2 3 1 4RAKESH SHUKLA       300750047785339
1 1 2 3 3 1 1 2 3 4999999999 1 1 2 3 2 0ANJAN BAKSHI        311251087783867
1 1 2 3 4 1 1 2 4 2999999999 1 1 2 3 3 1AVINASH SASEA       251153067797352
1 1 2 4 19999999999999999999 1 1 2 3 2 0REDDY P.G.K.        1912540778583
1 1 2 4 2 1 1 2 4 3999999999 1 1 2 3 4 2DEEPAK BANERJI      1204550878828
1 1 2 4 3 1 1 2 4 4999999999 1 1 2 4 2 3ALOK AIMA           2405522078987
1 1 2 4 49999999999999999999 1 1 2 4 3 4HEMNANI A.K.        1208582178786
1 2 1 1 1 1 2 1 1 2999999999 1 1 2 1 3 3VENKAT PAGHAVAN     3112581578694
1 2 1 1 2 1 2 1 1 3 1 2 1 2 1 1 2 1 1 1 4BRIJESH PATEL       1210571978788
1 2 1 1 3 1 2 1 1 4999999999 1 2 1 1 2 0SARDAR A.M.         0102562078458
1 2 1 1 4 1 2 1 2 2999999999 1 2 1 1 3 1SHAMEER BEGUM       1908571178765
1 2 1 2 19999999999999999999 1 2 1 1 2 0RITESH KUMAR        1101560878582
1 2 1 2 2 1 2 1 2 3999999999 1 2 1 1 4 2AJIT YADAV          1311540578938
1 2 1 2 39999999999999999999 1 2 1 2 2 3BENJAMIN            3004550778869
```

```
                THE RECORDS REQUESTED ARE:

       KEYS       SKEY        LSKEY        PRKEY    DISC                 NAME

     1 1 1 1 3  9999999999  1 1 1 1 4  1 1 1 1 2     3  AMIRUDDIN           Ø6114915778535
       KEYS       SKEY        LSKEY        PRKEY    DISC                 NAME

     1 1 1 1 4  1 1 1 2 1   1 1 1 2 4  1 1 1 1 3     4  AMRITLAL JINDAL     Ø512590376794376
       KEYS       SKEY        LSKEY        PRKEY    DISC                 NAME

     1 1 1 2 1  9999999999  1 1 1 2 2  1 1 1 1 4     Ø  ANAND PRAKSH        Ø3116104797
       KEYS       SKEY        LSKEY        PRKEY    DISC                 NAME

     1 1 1 2 2  9999999999  1 1 1 2 3  1 1 1 2 1     1  ANAND MADAN         Ø40254Ø87765435
  ROOT OF TREE = 1 1 2 1 1


STARTING TIME   = 88554 MILLISECONDS
FINISHING TIME  = 89166 MILLISECONDS

PROCESSING TIME =  612 MILLISECONDS
```

```
THE RECORDS REQUESTED ARE:

KEYS        SKEY        LSKEY       PRKEY      DISC              NAME

1 1 1 1 3   9999999999  1 1 1 1 4   1 1 1 1 2     3  AMIRUDDIN            06114915778535
KEYS        SKEY        LSKEY       PRKEY      DISC              NAME

1 1 1 1 4   1 1 1 2 1   1 1 1 2 4   1 1 1 1 3     4  AMRITLAL JINDAL      0512590376794376
KEYS        SKEY        LSKEY       PRKEY      DISC              NAME

1 1 1 2 1   9999999999  1 1 1 2 2   1 1 1 1 4     0  ANAND PRAKSH         03116104797
KEYS        SKEY        LSKEY       PRKEY      DISC              NAME

1 1 1 2 2   9999999999  1 1 1 2 3   1 1 1 2 1     1  ANAND MADAN          040254087765435
KEYS        SKEY        LSKEY       PRKEY      DISC              NAME

1 1 1 2 3   9999999999  1 1 1 3 1   1 1 1 2 2     2  AMRIT ANEJA          0801520478516
KEYS        SKEY        LSKEY       PRKEY      DISC              NAME

1 1 2 1 2   9999999999  1 1 2 1 3   1 1 2 1 1     1  VERA RAGHAVAN        1212612078151
KEYS        SKEY        LSKEY       PRKEY      DISC              NAME

1 1 1 3 1   9999999999  1 1 1 3 2   1 1 1 2 3     3  BHAG RANI ARORA      21065709797
ROOT OF TREE = 1 1 2 1 1

STARTING TIME  = 84007 MILLISECONDS
FINISHING TIME = 85004 MILLISECONDS

PROCESSING TIME =  997 MILLISECONDS
```

THE RECORDS INSERTED ARE:

```
2 2 2 1 19999999999999999999 1 2 1 2 1 1MKS ROY              05045302456789
2 2 2 2 29999999999999999999 1 1 2 2 4 3MN CHAKRAVARTHY      07055608095432 9
2 2 2 2 39999999999999999999 2 2 2 2 2 4AM PILLAI            26125608976543
2 2 2 2 49999999999999999999 1 1 2 3 1 4RN RAO               2104540504087 6
    ROOT OF TREE = 1 1 2 1 1
```

```
STARTING TIME   =  99420 MILLISECONDS
FINISHING TIME  = 100389 MILLISECONDS

PROCESSING TIME =    969 MILLISECONDS
```

```
                                   THE RECORDS INSERTED ARE:

2 2 2 1 19999999999999999999 1 2 1 2 1 1NKS ROY               05045302456789
2 2 2 2 29999999999999999999 1 1 2 2 4 3MN CHAKRAVARTHY       07055608054329
2 2 2 2 39999999999999999999 2 2 2 2 2 4AM PILLAI             26125608976543
2 2 2 2 49999999999999999999 1 1 2 3 1 4RN RAO                21045405040876
2 2 2 3 19999999999999999999 1 1 2 4 1 1LM MENON              150847040867
2 2 2 3 29999999999999999999 1 1 2 4 2 3PR ADAVI              161250040805432
2 2 2 3 49999999999999999999 2 2 2 3 2 4TK JAGANNATH          16055507084786
    ROOT OF TREE = 1 1 2 1 1


 STARTING TIME  =  93382 MILLISECONDS
 FINISHING TIME =  95304 MILLISECONDS

 PROCESSING TIME = 1922 MILLISECONDS
```

THE RECORDS DELETED ARE:

```
        KEYS        SKEY       LSKEY        PRKEY   DISC                  NAME

      1 1 1 1 3   1 1 1 1 4  9999999999   1 1 1 1 2   93  AMIRUDDIN            06114915778535
        KEYS        SKEY       LSKEY        PRKEY   DISC                  NAME

      1 1 1 1 4   1 1 1 2 4   1 1 1 2 1   1 1 1 1 3   94  AMRITLAL JINDAL      05125903767943 76
        KEYS        SKEY       LSKEY        PRKEY   DISC                  NAME

      1 1 1 2 1   1 1 1 2 2  9999999999   1 1 1 1 4   90  ANAND PRAKSH         03116104797
        KEYS        SKEY       LSKEY        PRKEY   DISC                  NAME

      1 1 1 2 2   1 1 1 2 3  9999999999   1 1 1 2 1   91  ANAND MADAN          040254087765435
ROOT OF TREE = 1 1 2 1 1

STARTING TIME   = 300157 MILLISECONDS
FINISHING TIME  = 301144 MILLISECONDS

PROCESSING TIME =    987 MILLISECONDS
```

```
                  THE RECORDS DELETED ARE:

   KEYS        SKEY       LSKEY       PRKEY    DISC                 NAME

1 1 1 1 3  1 1 1 1 4  9999999999  1 1 1 1 2   93  AMIRUDDIN              06114915778535
   KEYS        SKEY       LSKEY       PRKEY    DISC                 NAME

1 1 1 1 4  1 1 1 2 4  1 1 1 2 1   1 1 1 1 3   94  AMRITLAL JINDAL        0512590376794376
   KEYS        SKEY       LSKEY       PRKEY    DISC                 NAME

1 1 1 2 1  1 1 1 2 2  9999999999  1 1 1 1 4   90  ANAND PRAKSH           03116104797
   KEYS        SKEY       LSKEY       PRKEY    DISC                 NAME

1 1 1 2 2  1 1 1 2 3  9999999999  1 1 1 2 1   91  ANAND MADAN            0402540877654 35
   KEYS        SKEY       LSKEY       PRKEY    DISC                 NAME

1 1 1 2 3  1 1 1 3 1  9999999999  1 1 1 2 2   92  AMRIT ANEJA            0801520478516
   KEYS        SKEY       LSKEY       PRKEY    DISC                 NAME

1 1 2 1 2  1 1 2 1 3  9999999999  1 1 2 1 1   91  VERA RAGHAVAN          1212612078151
   KEYS        SKEY       LSKEY       PRKEY    DISC                 NAME

1 1 1 3 1  1 1 1 3 2  9999999999  1 1 1 2 3   93  BHAG RANI ARORA        21065709797
ROOT OF TREE = 1 1 2 1 1

STARTING TIME  = 294851 MILLISECONDS
FINISHING TIME = 296628 MILLISECONDS

PROCESSING TIME =  1777 MILLISECONDS
```

1 1 1 3 1	1 1 1 3 2	9999999999	1 1 1 2 3	3	BHAG RANI ARORA	21065709797
1 1 1 3 2	1 1 1 3 3	1 1 1 4 1	1 1 1 3 1	4	ARORA HARBANS	1911552078453
1 1 1 3 3	1 1 1 4 2	9999999999	1 1 1 3 2	Ø	ARORA KRISHAN	29125421775289 1
1 1 1 3 4	1 1 1 4 4	9999999999	1 1 1 2 4	1	ARORA VIJAY	12115816794289 1

STARTING TIME = 112974 MILLISECONDS
FINISHING TIME= 114101 MILLISECONDS

PROCESSING TIME = 1127 MILLISECONDS

1 1 2 2 1	1 1 2 2 2	9999999999	1 1 2 1 4	4	ARVIND KUMAR	0310560878838
1 1 2 2 2	1 1 2 2 3	9999999999	1 1 2 2 1	0	PARVEEN KAUR	0512580218865
1 1 2 2 3	1 1 2 2 4	9999999999	1 1 2 2 2	1	ANJANA DEVI	0610590178875
1 1 2 2 4	1 1 2 3 1	9999999999	1 1 2 2 3	2	VISHAL KUMAR	1511570978862

STARTING TIME = 119361 MILLISECONDS
FINISHING TIME= 120716 MILLISECONDS

PROCESSING TIME = 1355 MILLISECONDS

THE RECORDS REQUESTED ARE:

```
        KEYS          SKEY        LSKEY        PRKEY    DISC                      NAME

    1 1 1 1 3   1 1 2 1 1    1 1 1 1 4   1 1 1 1 2     3  AMIRUDDIN                 06114915778535
        KEYS          SKEY        LSKEY        PRKEY    DISC                      NAME

    1 1 1 1 4   1 1 1 2 1    1 1 1 2 4   1 1 1 1 3     4  AMRITLAL JINDAL           0512590376794376
        KEYS          SKEY        LSKEY        PRKEY    DISC                      NAME

    1 1 1 2 1   9999999999   1 1 1 2 2   1 1 1 1 4     0  ANAND PRAKSH              03116104797
        KEYS          SKEY        LSKEY        PRKEY    DISC                      NAME

    1 1 1 2 2   9999999999   1 1 1 2 3   1 1 1 2 1     1  ANAND MADAN               040254087765435
ROOT OF TREE = 1 4 1 1 1

STARTING TIME   = 160230 MILLISECONDS
FINISHING TIME  = 160842 MILLISECONDS

PROCESSING TIME =    612 MILLISECONDS
```

THE RECORDS REQUESTED ARE:

KEYS	SKEY	LSKEY	PRKEY	DISC	NAME	
1 1 1 1 3	1 1 2 1 1	1 1 1 1 4	1 1 1 1 2	3	AMIRUDDIN	06114915778535
1 1 1 1 4	1 1 1 2 1	1 1 1 2 4	1 1 1 1 3	4	AMRITLAL JINDAL	0512590376794376
1 1 1 2 1	9999999999	1 1 1 2 2	1 1 1 1 4	0	ANAND PRAKSH	03116104797
1 1 1 2 2	9999999999	1 1 1 2 3	1 1 1 2 1	1	ANAND MADAN	040254087765435
1 1 1 2 3	1 2 1 1 3	1 1 1 3 1	1 1 1 2 2	2	AMRIT ANEJA	0801520478516
1 1 2 1 2	9999999999	1 2 1 1 2	1 1 2 1 1	0	VERA RABHAVAN	1212612078151
1 1 1 3 1	1 1 2 1 3	1 1 1 3 2	1 1 1 2 3	3	BHAG RANI ARORA	21065709797

ROOT OF TREE = 1 4 1 1 1

STARTING TIME = 155055 MILLISECONDS
FINISHING TIME = 156234 MILLISECONDS

PROCESSING TIME = 1179 MILLISECONDS

THE RECORDS INSERTED ARE:

```
2 2 2 1 19999999999999999999999 2 1 1 1 3 4MKS ROY            05045302456789
2 2 2 2 29999999999999999999999 2 1 1 3 1 4MN CHAKRAVARTHY     070556080954329
2 2 2 2 39999999999999999999999 2 2 2 2 2 0AM PILLAI           26125608976543
2 2 2 2 49999999999999999999999 2 1 1 4 4 3RN RAO              210454050408776
2 2 2 3 19999999999999999999999 2 1 1 4 1 1LM MENON            150847040867
2 2 2 3 29999999999999999999999 2 1 1 4 3 3PR ADAVI            161250040805432
2 2 2 3 49999999999999999999999 2 2 2 2 4 4TK JAGANNATH        16055507084786
    ROOT OF TREE = 1 4 1 1 1
```

```
STARTING TIME   = 165496 MILLISECONDS
FINISHING TIME  = 167419 MILLISECONDS

PROCESSING TIME =  1923 MILLISECONDS
```

?LPTPLE Page Limit Exceeded

```
                    THE RECORDS DELETED ARE:

      KEYS        SKEY        LSKEY        PRKEY   DISC                  NAME

  1 1 1 1 3   1 1 1 1 4   1 1 2 1 1     1 1 1 1 2    93  AMIRUDDIN            06114915778535
      KEYS        SKEY        LSKEY        PRKEY   DISC                  NAME

  1 1 1 1 4   1 1 1 2 4   1 1 1 2 1     1 1 1 1 3    94  AMRITLAL JINDAL      05125903767943 76
      KEYS        SKEY        LSKEY        PRKEY   DISC                  NAME

  1 1 1 2 1   1 1 1 2 2   9999999999    1 1 1 1 4    90  ANAND PRAKSH         03116104797
      KEYS        SKEY        LSKEY        PRKEY   DISC                  NAME

  1 1 1 2 2   1 1 1 2 3   9999999999    1 1 1 2 1    91  ANAND MADAN          04025408776543 5
ROOT OF TREE = 1 4 1 1 1

STARTING TIME  = 310306 MILLISECONDS
FINISHING TIME = 311261 MILLISECONDS

PROCESSING TIME =   955 MILLISECONDS
```

```
                    THE RECORDS DELETED ARE:

         KEYS        SKEY        LSKEY        PRKEY   DISC             NAME

      1 1 1 1 3   1 1 1 1 4   1 1 2 1 1   1 1 1 1 2   93  AMIRUDDIN        06114915778535
         KEYS        SKEY        LSKEY        PRKEY   DISC             NAME

      1 1 1 1 4   1 1 1 2 4   1 1 1 2 1   1 1 1 1 3   94  AMRITLAL JINDAL  0512590376794376
         KEYS        SKEY        LSKEY        PRKEY   DISC             NAME

      1 1 1 2 1   1 1 1 2 2   9999999999  1 1 1 1 4   90  ANAND PRAKSH     03116104797
         KEYS        SKEY        LSKEY        PRKEY   DISC             NAME

      1 1 1 2 2   1 1 1 2 3   9999999999  1 1 1 2 1   91  ANAND MADAN      040254087765435
         KEYS        SKEY        LSKEY        PRKEY   DISC             NAME

      1 1 1 2 3   1 1 1 3 1   1 2 1 1 3   1 1 1 2 2   92  AMRIT ANEJA      0801520478516
         KEYS        SKEY        LSKEY        PRKEY   DISC             NAME

      1 1 2 1 2   1 2 1 1 2   9999999999  1 1 2 1 1   90  VERA RABHAVAN    1212612078151
         KEYS        SKEY        LSKEY        PRKEY   DISC             NAME

      1 1 1 3 1   1 1 1 3 2   1 1 2 1 3   1 1 1 2 3   93  BHAG RANI ARORA  21065709797
ROOT OF TREE = 1 4 1 1 1


STARTING TIME   = 304889 MILLISECONDS
FINISHING TIME  = 306489 MILLISECONDS

PROCESSING TIME =  1600 MILLISECONDS
```