

CHAPTER 1

INTRODUCTION

File Structures is the Organization of Data in Secondary Storage Device, in such a way that minimize the access time and the storage space. A File Structure is a combination of representations for data in files and of operations for accessing the data. A File Structure allows applications to read, write and modify data. It might also support finding the data that matches some search criteria or reading through the data in some particular order.

File Structures is the Organization of Data in Secondary Storage Device in such a way that minimize the access time and the storage space. A File Structure is a combination of representations for data in files and of operations for accessing the data. A File Structure allows applications to read, write and modify data.

Early in the computing history, secondary storage was in the form of magnetic tape and punched cards. Storage was cheap but access was limited to sequential. In 1956, IBM introduced the RAMAC magnetic disk device. Data could be accessed directly instead of sequentially. This was the dawn of the study of file structures. Advances in OS gave rise to more research on operating systems. The next analogy that was come up was the Direct Access which is the analogy to access to position in array. Indexes were invented and list of keys and pointers were stored in small files. This allowed direct access to a large primary file. But as the file grows the same problem arise as with the primary memory. Tree structures emerged for main memory in 1960's. This involved balanced and self-adjusting Binary Search trees (BST).

AVL trees (1963). In 1979, a tree structure suitable for files was invented: B trees and B+ trees good for accessing millions of records with three or four disk accesses. Theory on Hashing tables were developed over 60's and 70's goof for those files that do not change much over time. Expandable and dynamic Hashing were invented in late 70's and 80's which provided for one or two disk accesses even if file grow dramatically.

A good File Structure aims at:

- Fast access to great capacity.
- Reduce the number of disk accesses by collecting data into buffers, blocks or buckets.
- Manage growth by splitting these collections.

1.1 OBJECTIVE

Develop a mini project to implement Hashing for a file of Cyber cafe objects. Implement insert(), display(),search() and delete() using Hashing.

The project aims at developing a mini project that allows the end user to store information about Cyber cafe objects that it has linkup with, in a file with all the necessary details along with it. It helps the end user to insert a cafe detail and with a unique id that can be used to access the information of any user, it also helps the end user to search the information of a particular system id with the help of the unique id that is assigned to it and at last it also allows the end user to delete the existing record using the unique id by implementing hashing. The mini project is programmed with the help of the technique called Hashing using buckets that provides an easier way to insert and search the information stored in the file. This mini project will help the end user to maintain all the information in a proper and sorted manner.

1.2 SCOPE

A file system should be developed to store the names and details of all users. The details must include the User Id, Username, Purpose, System id, Check-in, Check-out . The record of all Users must be displayed when required. A user must be able to register himself by . User must be able to view all the records in the file. A search option must be provided to search for any User details using the User Id. If the record is present, it must be displayed. If it is not present, appropriate message should be displayed. Option to delete a User record must also be provided. If the record is present, it must be deleted. If it is not, an appropriate message should be displayed.

1.3 MOTIVATION

Early Work assumed that files were on tape. Access was sequential i.e the cost of access grew in direct proportion to the size of the file. As files grew very large, sequential access was not a good solution. Disks allowed for direct access. Indexes made it possible to keep a list of keys and pointers in a small file that could be searched very quickly. With the key and pointer, the user had direct access to the large, primary file.

Indexes also have a sequential flavour. When they grow too much, they also become difficult to manage. The idea of using tree structures to manage the index emerged in the early 60' s.

However, trees can grow very unevenly as records are added and deleted resulting in long searches requires many disk accesses to find a record. In 1963, researchers came up with the idea of AVL trees for data in memory. However, AVL trees did not apply to files because they work well when tree nodes are composed of single records rather than dozens or hundreds of them. In the 1970's came the idea of B-Trees which require an $O(\log_k N)$ access time where N is the number of entries in the file and k is the number of entries indexed in a single block of the B-Tree structure B-Trees can guarantee that we can find an entry among millions of others with only 3 or 4 trips to the disk. Retrieving entries in 3 or 4 accesses is good, but it does not reach the goal of accessing data with a single request. Hashing was a good way to reach this goal with files that do not change size greatly over time. Recently, Extendible Dynamic Hashing guarantees one or at most two disk accesses no matter how big a file becomes.

Hashing is the transformation of a string of characters into a usually shorter fixed length value or key that represents the original string. The hashing algorithm is called the hash function. The values returned by the hash function are called hash values depending on the key value given to it. The hash value generated is considered as address to store the record in the file.

The hash function may return the same hash value for different key inputs. This is called a collision. This can be resolved using one of the many available collision resolution techniques. Hashing is an extremely efficient technique as it directly accesses the record after obtaining its position by hashing (unless there is a collision). So the average time complexity of hashing technique is $O(1)$.

Hashing is a technique to convert a range of key values into a range of indexes of an array. A hash table is a data structure which implements an associative array abstract data type, a structure that can keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. Each data value has its on unique index value.

I got the inspiration from the cyber café, which could benefit a lot by this software and maintain the records related to the user information. My professor Madhura Prakash motivated me to implement this idea as File Structure Mini Project.

CHAPTER 2

DESIGN

2.1 METHODOLOGY: Hashing using buckets

Hashing is the transformation of a string of characters into a usually shorter fixed length value or key that represents the original string. The hashing algorithm is called the hash function. The values returned by the hash function are called hash values depending on the key value given to it. The hash value generated is considered as address to store the record in the file.

The hash function may return the same hash value for different key inputs. This is called a collision. This can be resolved using one of the many available collision resolution techniques. Hashing is an extremely efficient technique as it directly accesses the record after obtaining its position by hashing (unless there is a collision). So the average time complexity of hashing technique is $O(1)$.

Hashing is a technique to convert a range of key values into a range of indexes of an array. A hash table is a data structure which implements an associative array abstract data type, a structure that can keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found. Each data value has its on unique index value.

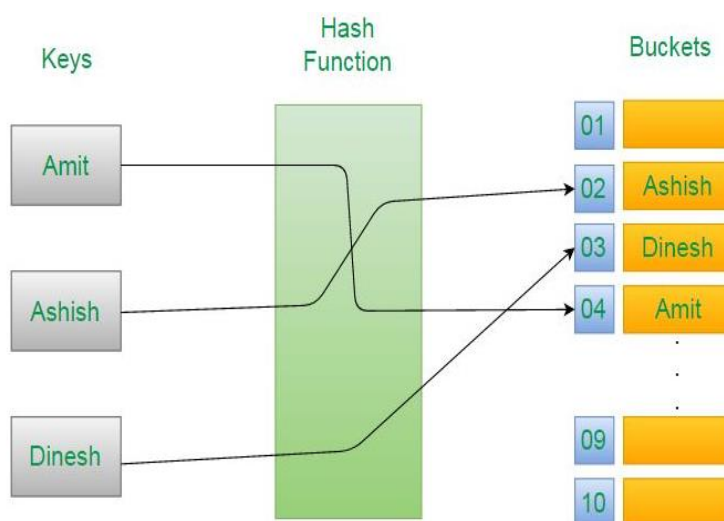


Figure 2.1: Example for hashing with buckets

2.2 Tools

Netbeans Ide

NetBeans is an integrated development environment (IDE) for Java. NetBeans allows applications to be developed from a set of modular software components called modules. NetBeans runs on Microsoft Windows, MacOS, Linux and Solaris. In addition to Java development, it has extensions for other languages like PHP, C, C++, HTML5, Javadoc, and Javascript. Applications based on NetBeans, including the NetBeans IDE, can be extended by third party developers. The NetBeans Team actively supports the product and seeks feature suggestions from the wider community. Every release is preceded by a time for Community testing and feedback.

The NetBeans Platform is a framework for simplifying the development of Java Swing desktop applications. The NetBeans IDE bundle for Java SE contains what is needed to start developing NetBeans plugins and NetBeans Platform based applications; no additional SDK is required. Applications can install modules dynamically. Any application can include the update Center module to allow users of the application to download digitally signed upgrades and new features directly into the running application. Reinstalling an upgrade or a new release does not force users to download the entire application again.

Java Development Kit (Jdk)

The Java Development Kit (JDK) is an implementation of either one of the Java Platform, Standard Edition, Java Platform, Enterprise Edition, or Java Platform, Micro Edition platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, mac OS or Windows. The JDK includes a private JVM and a few other resources to finish the development of a Java Application. Since the introduction of the Java platform, it has been by far the most widely used Software Development Kit (SDK). On 17 November 2006, Sun announced that they would release it under the GNU General Public License (GPL), thus making it free software. This happened in large part on 8 May 2007, when Sun contributed the source code to the Open JDKx.

Notepad

Notepad is a text editor and source code editor for use with Microsoft Windows. It supports tabbed editing, which allows working with multiple open files in a single window.

Language: Java

Java is –

- **Object Oriented** – In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** – Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** – Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** – With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** – Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** – Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** – Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** – With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** – Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** – With the use of Just-In-Time compilers, Java enables high performance.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 User Requirements

A file system should be developed to store the names and details of all Users. The details must include the User Id, User name, Purpose, System_Id, Check_in, Check_out . The record of all User must be displayed when required. An Admin must be able to register a new User and the necessary details. Admin must be able to view all the records in the file. A search option must be provided to search for any User details using the User Id. If the record is present, it must be displayed. If it is not present, appropriate message should be displayed. Option to traverse a User record must also be provided. If the record is present, it must be traversed. If it is not, an appropriate message should be displayed.

3.2 Hardware Requirements

- Processor: Intel core i5-6500 processor (6m cache up to 3.20 GHz)
- RAM: 4GB or greater.
- Hard Disk Space: 2GB.
- System Type: 64-bit Operating System.

3.3 Software Requirements

- Operating System: Windows 7 or higher.
- JDK: Java Development Kit v8.1
- Integrated Development Environment (IDE): Netbeans IDE 8.2.
- Text Editor: Notepad++ or Notepad.

3.4 Functional requirements

Whenever the user registers himself/herself with the new record, should be inserted into the file containing the required details of the user.

- New record is inserted and the details are entered.
- The admin should be able to view the details of the new user record inserted.
- In the display, the user details like the UId, name, purpose, system_id, check_in, check_out are displayed.
- The search operation should help the admin to find the User details he desires.
- The delete operation should result in a deleting any particular record using the user_id.

3.5 Non-functional requirements

- The application should display appropriate messages to the user if something goes wrong like no details entered before pressing enter.
- The application should not crash.
- The user should be able to understand the facilities the application is providing to the user.
- The application should be user friendly.

CHAPTER 4

SYSTEM DESIGN AND DEVELOPEMENT

4.1 Architectural design

Hashing is an extremely efficient technique as it directly accesses the record after obtaining its position by hashing (unless there is a collision). So the average time complexity of hashing technique is $O(1)$.

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. The Cyber café File Management system DFD contains four processes, one external entity and one data store.

Data Flow Diagram notations:



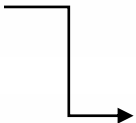
Process: A process or task that is performed by the system.



Datastore: Datastores are repositories of data in the system.



They are sometimes also referred to as files.

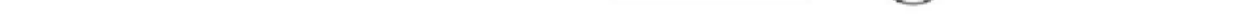


Dataflow: Dataflows are pipelines through which packets of information flow. Label the arrows with the name of the data that move through it.



Entity: A source of data or a destination for data

The Cyber cafe Management system Data Flow Diagram example contains four processes, one external entity and one data store.



Journal of Management Education 36(7) 809–827

CHAPTER 5

IMPLEMENTATION

The technique used to implement this file structure application is hashing. The details of the Cyber cafe system are bound together under the records. The details included are: the id of the User and the positions of different notes in cafe. The class also includes a flag value to check for various conditions.

Hashing can be used to perform various operations such as insertion, deletion, updating, search -etc. The hash function takes in the key value as input. Performs hash, and tries to perform addition, deletion or search starting from that position.

5.1 List of modules:

- Insert
- Search
- Display
- Delete

5.2 Module description:

Insert: The ID of the User is taken as the key value from the user. The key is given as input to a hash function that generates an address. If the returned address space is empty, the record is inserted there. Else, progressive overflow is applied.

Search: The ID of the User is taken as the key value from the user. The key is given as input to a hash function that generates an address. If the ID of the User at the hash values matches the given address, its details are displayed. Else, the next consecutive address spaces are checked for a match, checking the first address space after the last one, until we arrive back to the hash value. We stop and display when there is a match. Else, a message saying that the given record is not present is displayed.

Display: Display operation will display all information of all the User present in the file. Display of the records is represented in the form of tables.

Delete: The ID of the User is taken as the key value from the user. The key is given as input to a hash function that generates an address. If the ID of the User at the hash values matches the given address, it is deleted. Else, the next consecutive address spaces are checked for a match, checking the first address space after the last one, until back to the hash value. We stop and delete when there is a match. Else, a message saying that the given record is not present is displayed.

5.3 Algorithms / Pseudocode:

Hash: The hash algorithm computes the hash value for the ID of the User by performing arithmetic operations on the ASCII value of the first letter of User ID.

1. Start.
2. Input the key as User ID.
3. Extract the first letter of the User.
4. Obtain its ASCII value.
5. Perform a mod operation.
6. This is the hash address, return it.

Insert: The insert algorithm computes the hash address for the User ID. The record is then inserted at that address. If the address is already in use the next free address is used to make the entry of the record into the file.

1. Start
2. Read the values from user
3. Compute the hash value for the ID of the User.
4. Read the all the contents of the file to buffer records.
5. Check if the hash address is EMPTY.

If it is, write the User and details into the buffer record at that position.

Else, scan the buffer records, starting from the address returned by the hash function, in a buckets, placing the User in the next available EMPTY location.

6. Write the contents of the buffer records back to the file.
7. Stop

Search: The search function computes the hash address of the ID of the User the admin is looking for. It then searches the file at that address and checks if the name specified by the admin is same as the name present at that address. If it is, then the record details is retrieved. If it is not, then the subsequent address spaces are checked for the presence of the record.

1. Start
2. Read the values from user
3. Compute the hash value for the ID of the User.
4. Read the all the contents of the file to buffer records.
5. Check if the User at the hash value matches the entered User ID.

If it does, display its details.

Else, scan the buffer records, starting from the address returned by the hash function, in a Buckets, checking for a match of User.

Display the User details if a match is found.

Else display a message saying “User not present.”

6. Stop

Delete: The delete function computes the hash address of the ID of the User the admin is looking for. It then searches the file at that address and checks if the name specified by the user is same as the name present at that address. If it is, then the record details are deleted and space is reclaimed by setting it's values back to 'EMPTY'. If it is not, then the subsequent address spaces are checked for the presence of the record.

1. Start
2. Read the values from user
3. Compute the hash value for the name of the User.
4. Read the all the contents of the file to buffer records.
5. Check if the User at the hash value matches the entered User ID.

If it does, write “EMPTY” to the buffer record.

Else, scan the buffer records, starting from the address returned by the hash function, in a circular fashion, checking for a match of User.

Write “EMPTY” to the buffer record if a match is found.

Else display a message saying “User not present.”

6. Stop

Display: Display function fetches the data from the file and displays it on the user interface.

1.Start

2. Read all the contents of the file to buffer records.

3. Display the contents of buffer records.

5.4 Front end

Menu: The form has various operations that can be performed on the file. Necessary option can be chosen.

Insertion: This form accepts the input from the user and generates the hash for the User ID entered. The record is written into the appropriate address space and a relevant message is displayed.

Search: The user is asked for the ID of the User that is being searched for. Its hash is computed, and search begins. Appropriate messages are displayed.

Delete: User specifies the name to be deleted. The record corresponding to it is deleted. The deleted space is reclaimed. Appropriate messages are displayed.

Display: This form fetches the data from the file and displays it.

5.5 Back end

In the given Cyber cafe object file system, the student details included are User ID and positions of different notes: r, g, d and n. the User_ID is the key. Records can be view and new record can be added. Records can be searched for and deleted using the key. The technique used for address allocation of the record is Hashing. The collision resolution technique applied is Buckets.

CHAPTER 6

RESULTS

6.1 Snapshots



Figure 6.1.1: Menu page

This is the welcome page of the application. It contains options of various operations that can be performed. It is a navigation window which helps the user navigate to Insert, Search, Display or Delete form.

The image shows a web application interface for adding user details. At the top, there is a blue header bar with 'INSERT INFO' on the left and 'CYBER CAFE RECORD SYSTEM' in the center. Below the header, there is a navigation bar with a blue button labeled '<-- HOME' on the left and a red button labeled 'LOGOUT' on the right. The main area has a black background. It contains six input fields arranged in two columns. The left column has labels 'UID -', 'USER NAME -', and 'PURPOSE -'. The right column has labels 'SYSTEM ID -', 'CHECK-IN -', and 'CHECK-OUT -'. Each label is followed by a white input field. At the bottom right of the main area, there is a red button labeled 'UPDATE'.

Figure 6.1.2: Add User_Details page

User can be added as records into the file. The text fields are used by the user to enter the record values. The hash address of the key value is generated and the record is stored into that position of the file. When all entity should be entered in Add User_details page then update button is pressed .it is stored from backend of random file.

The screenshot shows the 'CYBER CAFE RECORD SYSTEM' interface. At the top, there's a blue header with 'SEARCH INFO' on the left and 'CYBER CAFE RECORD SYSTEM' in the center. Below the header, there's a navigation bar with '<-- HOME' on the left and 'LOGOUT' on the right. The main area has a dark background. On the left, there are four input fields: 'UID -' with 'zxc', 'USER NAME -' with 'Max', 'PURPOSE -' with 'email', and 'SYSTEM ID -' with 'rtv'. To the right of these fields is a green 'SEARCH' button. Further right, there are two more input fields: 'CHECK-OUT -' with '4' and '5'. A modal dialog box titled 'Message' is centered on the screen, displaying an information icon, the text 'SEARCH SUCCESSFUL', and an 'OK' button.

Figure 6.1.3: Search User page (successful)

Admin can search for a particular user by entering its name, the hash address is computed for that user id and the search is performed from that position. If present the record is displayed.

The screenshot shows the same 'CYBER CAFE RECORD SYSTEM' interface. The 'UID -' field now contains 'cvb'. The 'SEARCH' button is still green. The modal dialog box titled 'Message' is centered on the screen, displaying an information icon, the text 'SEARCH UNSUCCESSFUL', and an 'OK' button.

Figure 6.1.4: Search User page (unsuccessful)

A search can be made for a user by entering its name. If the user is not present in the file, appropriate message is displayed.



UID	USER NAME	PURPOSE	SYSTEM ID	CHECK-IN	CHECK-OUT
qwe	Mithun	Browsing	123	9:00	10:00
rty	Sadiq	Gaming	122	22:00	23:00
sdf	Tejas	Browse	433	9:00	11:00
dfs	Achyut	Game	24	1:00	3:00
vcx	Ujjanth	General	234	2:00	4:00
sdf	Amogh	E-mail	534	12:00	1:00
hjh	Eshwar	General	234	3:00	4:00

Figure 6.1.5: View user page

All the Users in the file are displayed. At positions where no data is present, “EMPTY” is displayed.

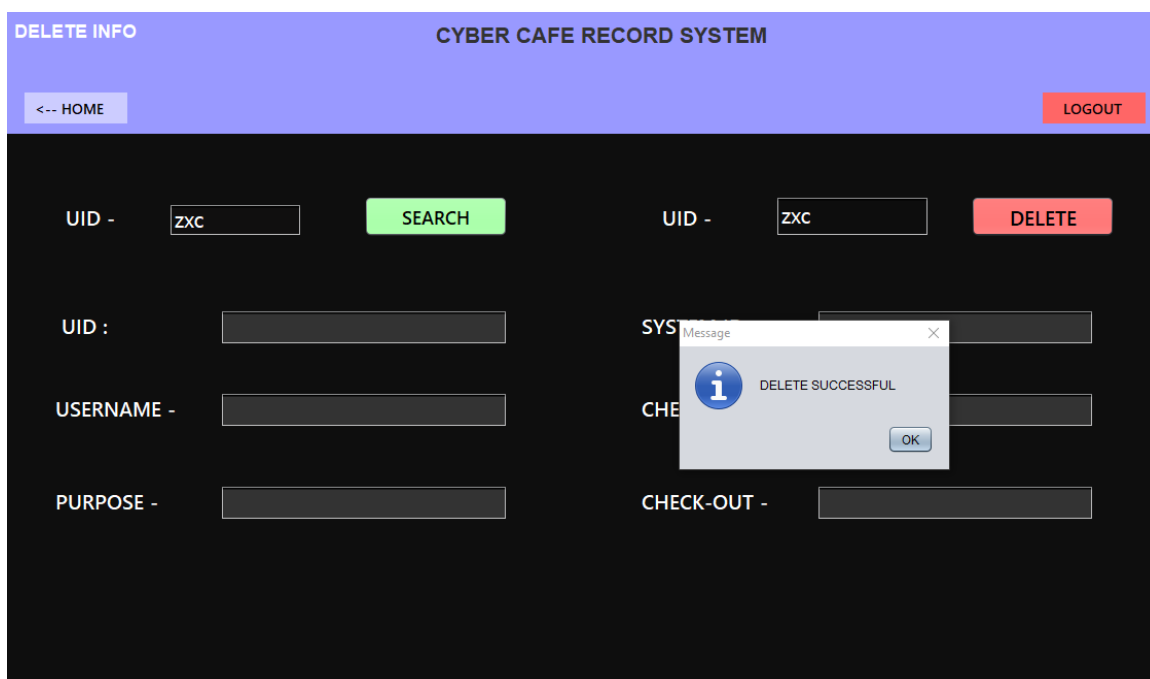


Figure 6.1.6: Delete page

User can enter the User ID of the user to delete a particular record and if it's present the record is deleted. The value of all the fields in the record is set back to it's default, i.e. 'EMPTY'. Hence, space is reclaimed.

The screenshot displays the 'DELETE INFO' section of the 'CYBER CAFE RECORD SYSTEM'. The interface has a purple header bar with a '<-- HOME' button on the left and a 'LOGOUT' button on the right. Below the header, there are two main sections. The left section contains a 'UID -' label followed by a text input field containing 'qwe', a green 'SEARCH' button, and three more input fields labeled 'UID:', 'USERNAME -', and 'PURPOSE -'. The right section contains a 'UID -' label followed by a text input field containing 'qwe', a red 'DELETE' button, and a 'CHECK-OUT -' label followed by an input field. A modal message box is overlaid in the center, titled 'Message', with an information icon and the text 'SEARCH UNSUCCESSFUL CANT DELETE'. It has an 'OK' button at the bottom right.

Figure 6.1.7: Delete page (unsuccessful)

CONCLUSION

Hashing has been implemented using Buckets to manage a file system of Cyber café object. Both front end and back end have been developed using Java. Front end is implemented using Java Swing API. Back end runs on Java Code.

The Application registers the various attributes of a User and inserts it into the file after generating the hash value by hashing the User ID. Search, display and delete operations on these records can be done.

ADVANTAGES OF HASHING USING BUCKETS:

- More reliable and flexible method of data retrieval than any other data structure. Average Time Complexity for data retrieval using hash function is $O(1)$.
- Hash tables often turn out to be more efficient than search trees or any other table lookup structure. They are hence widely used in many kinds of computer software, particularly for associative arrays, database indexing, caches and sets.
- Progressive Overflow is the simplest and neat. It works the best when the number of record is less than the available space.

DISADVANTAGES OF HASHING USING BUCKETS:

- The total number of records are fixed.
- The search function will search the entire file, if the record is not present.

APPLICATIONS OF HASHING USING BUCKETS:

- Hashing is mainly used in Networking Problems
- Interactive multimedia (games, videoconferencing)
- Trading platforms

REFERENCES

Textbooks :

- 1) Michael J. Folk, Bill Zoellick, Greg Riccardi: File Structures-An Object Oriented Approach with C++, 3rd Edition, Pearson Education, 1998.
- 2) K.R. Venugopal, K.G. Srinivas, P.M. Krishnaraj: File Structures Using C++, Tata McGraw-Hill, 2008.
- 3) Scot Robert Ladd: C++ Components and Algorithms, BPB Publications, 1993.
- 4) Java- The complete reference, Herbert Schildt, seventh edition.