PROJECT REPORT

Data Structure and Algorithms

(UCS 406)

File Locator and Handler

by

Vikas Airan(101783067)

Sharanpreet kaur(101783038)

Pardaman Kaur(101783031)

Nidhi Rajpal(101783029)

Submitted to

Dr Rajesh Mehta

Computer Science & Engineering Department

Thapar Institute of Engineering and Technology, Patiala

May 2018

TABLE OF CONTENT

1. Problem Formulation

1.1 File System Structure

Hard disks have two important properties that make them suitable for secondary storage of files in file systems:

(1) Blocks of data can be rewritten in place, and

(2) they are direct access, allowing any block of data to be accessed with only minor movements of the disk heads and rotational latency.

Disks are usually accessed in physical blocks, rather than a byte at a time. Block sizes may range from 512 bytes to 4K or larger.

File systems organize storage on disk drives, and can be viewed as a layered design:

At the lowest layer are the physical devices, consisting of the magnetic media, motors & controls, and the electronics connected to them and controlling them. Modern disk put more and more of the electronic controls directly on the disk drive itself, leaving relatively little work for the disk controller card to perform.

I/O Control consists of device drivers, special software programs which communicate with the devices by reading and writing special codes directly to and from memory addresses corresponding to the controller card's registers. Each controller card on a system has a different set of addresses that it listens to, and a unique set of command codes and results codes that it understands.

The basic file system level works directly with the device drivers in terms of retrieving and storing raw blocks of data, without any consideration for what is in each block. Depending on the system, blocks may be referred to with a single block number or with head-sector-cylinder combinations.

The file organization module knows about files and their logical blocks, and how they map to physical blocks on the disk. In addition to translating from logical to physical blocks, the file organization module also maintains the list of free blocks and allocates free blocks to files as needed.

The logical file system deals with all of the meta data associated with a file i.e. everything about the file except the data itself. This level manages the directory structure and the mapping of file names to file control blocks, FCBs, which contain all the meta data as well as block number information for finding the data on the disk.

The layered approach to file systems means that much of the code can be used uniformly for a wide variety of different file systems, and only certain layers need to be filesystem specific. Common file systems in use include the UNIX file system, UFS, the Berkeley Fast File System, FFS, Windows systems FAT, FAT32, NTFS, CD-ROM systems ISO 9660 etc.

1.2 Directory Implementation

Directories need to be fast to search, insert, and delete, with a minimum of wasted disk space.

1.2.1 Linear List

A linear list is the simplest and easiest directory structure to set up, but it does have some drawbacks. Finding a file (or verifying one does not already exist upon creation) requires a linear search. Deletions can be done by moving all entries, flagging an entry as deleted, or by moving the last entry into the newly vacant position. Sorting the list makes searches faster, at the expense of more complex insertions and deletions. A linked list makes insertions and deletions into a sorted list easier, with overhead for the links.

1.2.2 Hash Table

A hash table can also be used to speed up searches. Hash tables are generally implemented in addition to a linear or other structure.

1.2.3 Tree

Representing (and accessing) the location of files in a computer to show them at the ends of paths through a hierarchical tree branching from a root node) is referred to as tree representation. This tree with its files at the end of its paths was called a file directory, and a file's location was given as the path from the root through the various intervening named nodes (subdirectories) and terminating in the file's name. It is effectively fast in retrieving or accessing file because file can be directly accessed knowing its path compared to liked list approach where every file before required file in list needs to be traversed before finally accessing desired file.

1.3 Accessing File

The required file located can now be accessed and various operations can be performed on file. The file can now be read and modified easily. Problem encountered is while performing a search operation on file. It can be tackled with help of following data structures:

1.3.1 Linear List

A linear list search approach can be used to search for a specific word in file. Main disadvantage is that it requires a linear search that is whole file needs to be traversed to find the positions of a word. So the complexity turns out to be O(n).

1.3.2 Hash Table

A hashed representation can be used to create a hashed table and then searching operations can be easily performed on file. It requires less time to search for a word as with help of hashed table the searching can be directly started from the specified locations as stated in hash tables.

2. Analysis of Problem

The major problem encountered is the representation of the required directory so that fast searching and easy accessing can be performed. As searching for all the locations of a file and then accessing for a specific file is to be performed it required representing the directory in a correct manner so that easy traversing of files can be performed. This problem is solved with help of tree data structures. The directory and its subdirectories can be represented in form of tree data structure and required files can be easily searched using Depth First Search Algorithm.

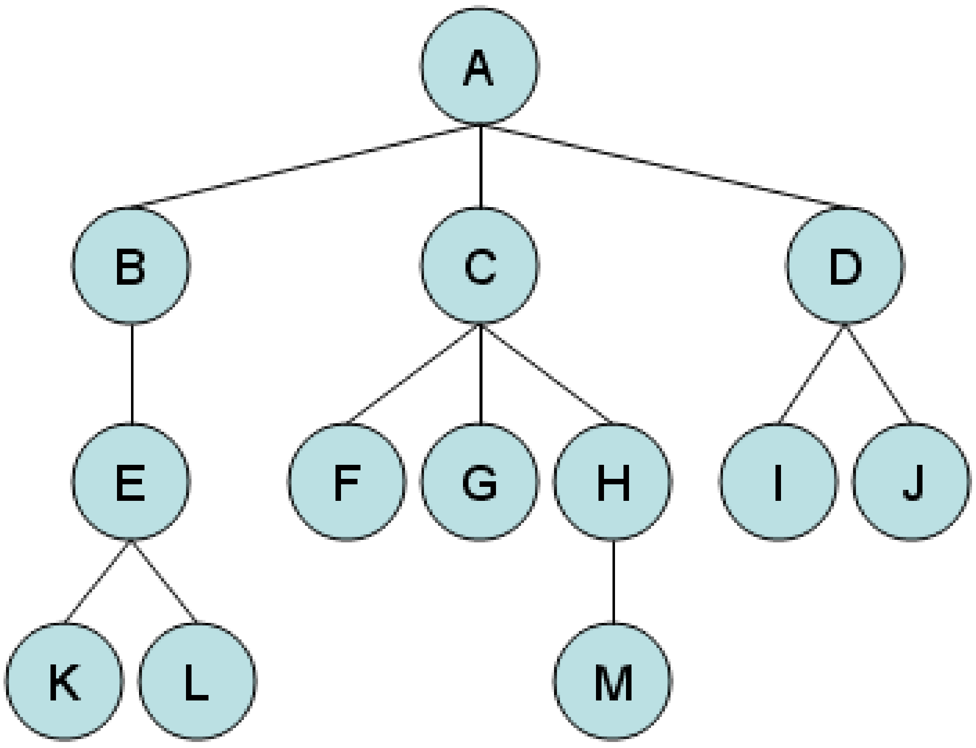
Another problem encountered is searching of a word in a text file. It can be easily performed by a linear search by traversing whole file and searching for all occurrences of the required word in file. The major problem encountered in this process is to make searching fast and efficient as every time a word needs to be searched the whole file need to be traversed again in order to search the word. It is solved by using Hash Tables. A hash table is created and every time a word needs to be searched it can be easily done with help of hashed table which easily locates the position of the required word.

3. Data structure and Algorithmic techniques Used

Tree Data structure is used for representation of files in system, Depth First Search Algorithm is used for searching of file in the Tree formed and linked list representation of queue data structure is used to create list of all locations of a file with name as searched for. An adjacency list hash table is used for easy searching for the position of a word within a file.

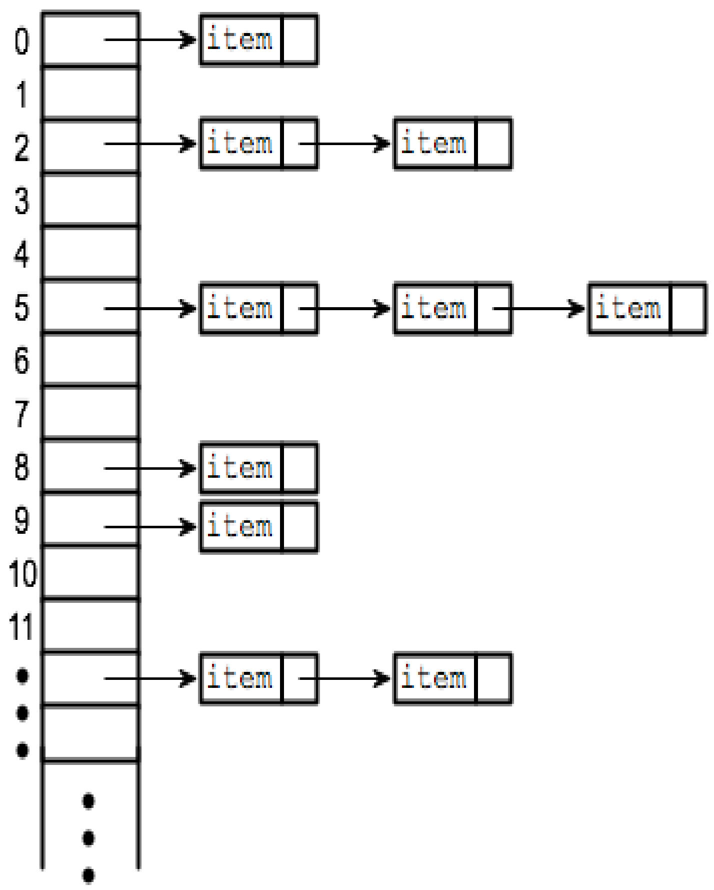
3.1 Tree

A tree data structure can be defined recursively as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reference is duplicated, and none points to the root. In our model we used a tree where number of children’s is not defined as there can be any number of files in a folder and its subfolder. For implementing it we used a list of nodes for parent node and each node can be accessed by accessing its previous node and parent.



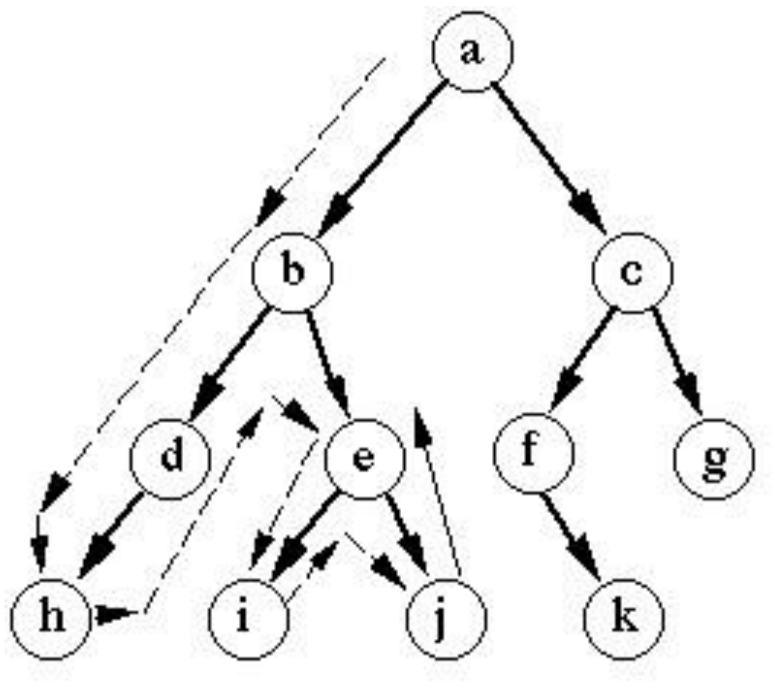
3.2 Hash Table

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data. Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from. In our project we used an adjacency list representation of hash table and using ASCII code to generate indexes.



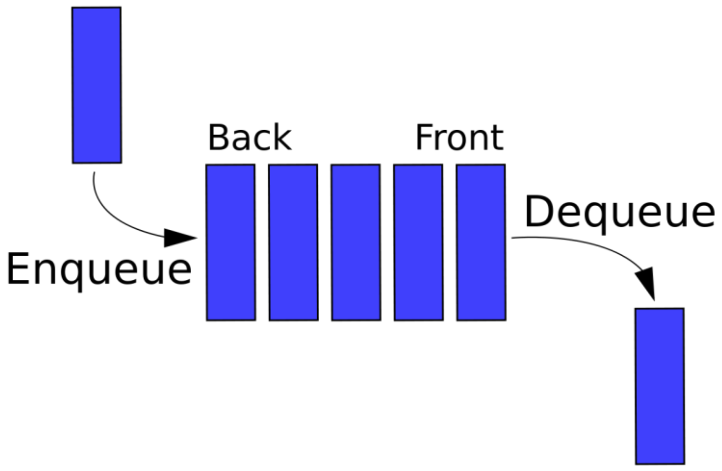
3.3 Depth First Search (DFS)

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking. In this project Preorder Depth First Search is Used.



3.4 Queue

A queue is a linear list of elements in which deletion of an element can take place only at one end called the front and insertion can take place on the other end which is termed as rear. The term front and rear are used frequently while describing queues in a linked list.



4. Methodology

4.1 Tree

The following structure for tree representation is used

```
struct rootnode
{
   string loc;
   rootnode *next;
   rootnode *child;
   rootnode *parent;
};
```

Here the first child node of a parent is referred as child and subsequent children's can be added by first accessing child and then storing new child in its next node. By recursively using the above operation for each folder in a directory and its subdirectories tree of the directory is formed with root as the input directory.

4.2 Hash Table

An adjacency linked list is created with following structure

```
struct node
{
 int pos;
 node *next;
};
```

A hash table with 28 indexes is created by creating an array of the above adjacency list as shown below

```
struct node *top[28];
```

The following hash function is defined to store the position of a character in the array of adjacency list. The function returns the index of the adjacency list where its position is to be stored.

```
int indexofchar(char letter)
{
 if((int)letter<=90&&(int)letter>=65)
           return ((int)letter-65);
 else if((int)letter<=122&&(int)letter>=97)
     return ((int)letter-97);
 else if((int)letter<=57&&(int)letter>=48)
           return (26);
 else
           return (27);
}
```

4.3 Depth First Search (DFS)

The following code snippet is used to implement DFS algorithm and printing the locations of the searched file.

```
void Listtraverse()
{
struct list *ptr=start;
if(ptr==NULL)
{
cout<<"File Not Found In Current Directory\n";
}
else
{
int i=1;
while(ptr!=NULL)
{
cout<<i<<") "<<ptr->loc<<"\n";
ptr=ptr->next;
i++;
}
}
}
```

4.4 Queue

The following structure is used to create a list to store all the retrieved locations of the searched file

```
struct list
{
 string loc;
 list *next;
};
```

As searching operation is performed searched locations are added to list. Then other operations are performed on the retrieved list.

5. Result and Conclusion

Tree is created for the directory as specified by the user. Then in the directory successful retrieval of all the locations of a user specified name is performed. Then reading of the file and searching for locations of a string in file is performed with help of generated hash table of the file and copy operation to copy all data of file to other location can be performed.

All the operations were successfully implemented with help of specified data structures thus file is located in directory and file handling operations executed. The complexity of different data structures used is calculated and it turns out O(h) for creating tree and searching in tree i.e. DFS operation where h is height of tree. It is O(n) for searching a word in file operation where n is total number of characters in file and O(1) for enqueue and O(n) for traversing in queue operations where n is total number of locations found in tree. With use of hashing time required for searching decreases but complexity still remains O(n).

6. References

http://www.cplusplus.com/forum/beginner/9173/

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/

https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm