NAMAL INSTITUTE

DSA-LAB Project

Search Engine

Submitted by:

Kiran Zafar

Hurmat Ilyas

Muhammad Omair

Submitted to:

Dr. Malik Jahan

1 Abstract

Basically Search Engine is a tool, program or software through which the search is done like we can find data which the user is demanding. The search engine that we have designed is a program which searches the data the user asks it find. In the project, various kinds of data related to movies are given. This data is composed of the objects and each object has three attributes which are as follow: Movie ID, Rating and Voting. The main purpose is to analyze the given data and to design a search engine using an algorithm developed by the basic techniques of the data structures to reduce the time complexity in this case. The concept of hashing (Quadratic and Separate Chaining), 2-3-4 trees and AVL trees are being used to develop the search engine.

Contents

2 Introduction

Search engine is made using efficient algorithms. Many examples are there like Google Chrome, Mozilla Firefox etc. which search efficiently what the user is demanding from them. We've to use the Hashing, 2-3-4 trees and AVL trees to make the search engines.

2.1 Data Analysis

From the given file, it can be analyzed that the objects have three attributes: Movie ID, Rating and Votes. Movie ID is in the form of string (ab768720) and each movie have a unique ID. First two digits are characters and the remaining is integer. 7.0345678 Rating is in the form of Floating value (1.9).The third and the last attribute Votes are in the form of Integer (1459). From this analysis, it can be concluded that as each movie has its own unique ID so, if we store according to it there will be no clusters .As many movies have same ratings then there will be clustering and the votes will form less clusters.

2.2 Problem Statement

The task is to make an algorithm for an efficient search engine and for this purpose we are given a data about movies which has three attributes: **Movie ID, Rating** and **Votes** for the Movie**.** By using these attributes, we have to store the data in such a way that all the search algorithm give minimum time complexity and answer these questions (**Minimum Rating**, **Maximum Rating** and **Movies having same Rating)**

To minimize the time complexity we have to apply hashing, 234 trees and AVL trees on such an attribute which don't form clusters and uses less time. By using ID, we can store the data which will not form clusters but it increase the complexity and the complexity becomes O (n).The major issue is to convert the string into integer to store by id. Using votes will also increase the complexity and by ratings the clusters will form.

2.3 Possible Solution

Firstly, we have to convert the data given in string form to integers if we have to store it by ID. As each movie has its own unique id so in this case the complexity will increase and in case of voting, the same issue happens. So, only rating can minimize the complexity but clusters will be formed. The clusters can be minimized by the concepts of hashing and those trees. Separate chaining, quadratic probing and the 234 trees and AVL trees can be used to reduce this.

3 Methodology

The steps used in making search engines are as follows:

3.1 Reading File:

The data is given to us in a text file **" data.txt"**. This file has three attributes, which are movie ID **(String)**, then there is movie Rating **(Float)** and in the end there are Votes **(Integer)** of that movie.

We are reading this file through f=open(" data.txt", " r") and then data=f.read()

3.2 Storing

We need to store the data in order to apply different search functions on it. As the given data has three attributes .We are using separate chaining, Quadratic probing, 234 and AVL trees to store the data and each single node contain a Movie ID, Rating and Votes.

In order to store data by movie rating, we use different functions .Then the specific functions are there to get specific index to store data. These functions are storing data right after reading the file the attribute rating is passed to **the** function. The data is read line by line and each line contains Movie ID, Rating and Votes. Each new node is added at start to minimize the complexity of function.

3.3 Hashing (Separate Chaining and Quadratic probing)

In order to apply hashing we use a function **hashindex().** Firstly, the movie id is passed but it increase complexity so rating is used to get index then complexity is reduced and the data type will be changed to integer to store data precisely. It has the complexity of O(1).In both separate chaining and probing, Insert() functions are being used to pass the data and the Search () function is used.

3.4 2-3-4 trees

In 2-3-4 trees, the same functions are being used almost. The file is read through f.read() and the data of the movies is passed through the function. An array of class TwoThreeFour is created and this class uses the function of class Node to store data at index. The insert() and search() functions are being used in it. It is made efficiently using an efficient algorithm. It has complexity of O(log(n))

3.5 AVL trees

In AVL trees, the same functions are being used. The file is read through f.read() and the data of the movies is passed through the function. An array of class AVL trees is created and this class uses the function of class Node to store data at index. The insert() and rotateleft(), rotateright() functions are being used in it. It has the complexity of O(log(n)).

3.6 Minimum Rating

To find the minimum rating, search () function is used. 1.0 is being searched and the values of minimum ratings are returned.

3.7 Maximum Rating

To find the maximum rating, the search () function is used. 10.0 is being searched and it returned the values of maximum ratings.

3.8 Same Rating

To find the same ratings, input is taken from the user that will enter the average rating to know the number of movies that have same rating. Then the search () function is used which will return the length of movie having the given rating.

4 Result

We were able to create an efficient search engine on the given data whose max functions work on O(1). We apply the hashing and trees concept on rating to generate a specific key to store data. The concept of Node, Linked list, separate chaining, Hashing, quadratic probing, key concepts of 234 and AVL trees are being used.

5 Conclusion

It can be concluded that to make the search engine we have to read the data type first. Then we move forward towards the indexing and the most difficult task is to store data without forming large clusters. We have to compromise on complexity sometimes. Making the search engine is a difficult task as a huge amount of data needs to be stored.