Music Search Engine Summary Report

Jue Wang, Keyi Yu

Introduction

The idea of this project is to found a music search engine using C++ language. We used three files to build the databases. There are two text files including some basic characters and some stop words. Another .json file includes some music with information about the name of the song, the composer, the lyric and so on. We use two hashtables to save the basic characters, and stop words. After that, we split the lyrics of the song by using these two hashtables, saving them into another new hashtable and compute how many times each character occurs. Finally, after reading the input from the command line, the music search engine can find the song best matches the lyrics. Besides hashtable, another way to build the search engine is using B-tree. We also learned that data structure. We finally select hashtable for efficiency and simplicity.

Search Engine Model Overview

Some important functions in the code:

loadWordLibrary(string path):
The input is the path of the Baidu segmentation lexicon, the purpose of this function is to load these characteristics into a hashtable

loadStopWords(string path):
The input is the path of an unnecessary chinese characters thesaurus, the purpose of this function is to save these characters into another hashtable.

vector<string>loadData(string path):
The input is the path of the music database, the output of this function is a vector which saves the lyrics of each music.

vector<string>splitWords(string line):
The input of this function is the lyrics of the songs, this function is splitting the lyrics by using the hash table which saves the Baidu segmentation lexicon.
vector<string>deleteWords(vector<string> v):
The input of the function is the music lyrics which are already saved as strings in the vector, the purpose of this function is to delete the words in unnecessary chinese characters thesaurus.

buildInvertIndex(vector<vector<string>> totalMusic):
Save index in the hashtable, word as key, list of music index who contains the word as value

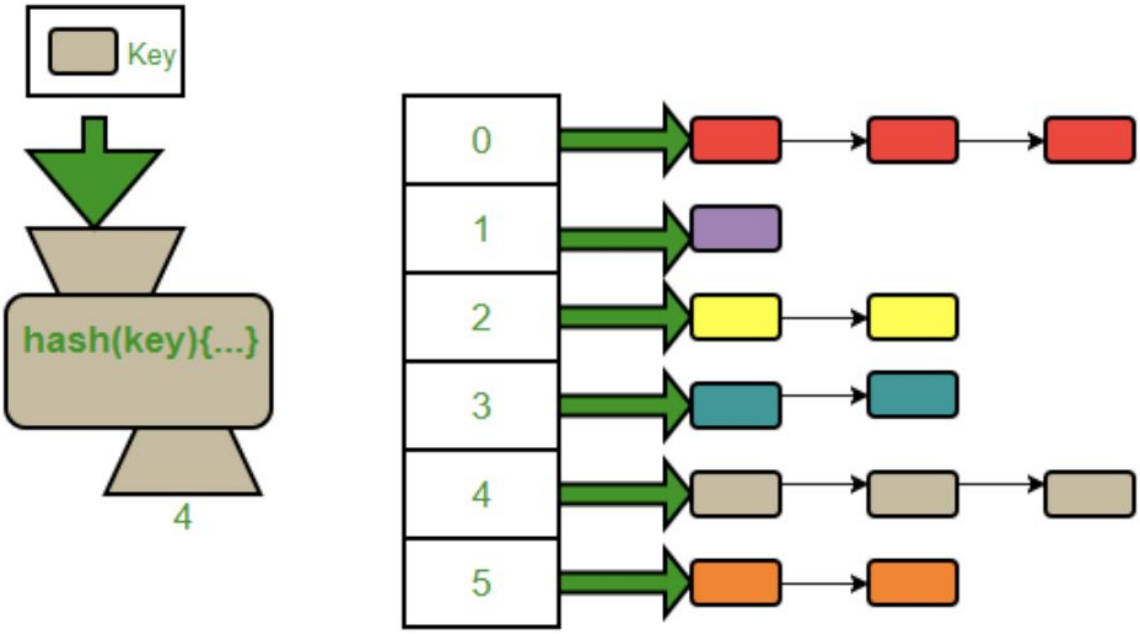
vector<string> computeScore(vector<string>& query_words):
Compute the frequency of the words, if the score is the highest, it means that the input lyrics are from this song.

show(vector<int>& recommendations):
Print out the computed scores in a user-friendly manner.

Candidate Data Structure

Hash Table

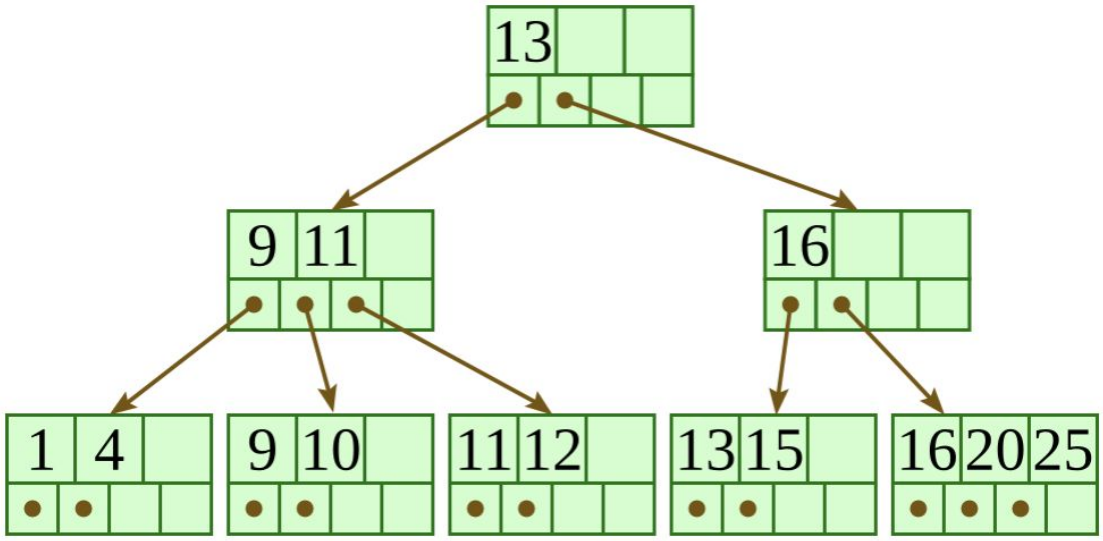
Hash table is a data structure that is used to save the data to Key-value pairs. Based on the Hash Table index, we can store the value at the appropriate location.
The benefit of the Hash Table is that each time when we try to access the Hash Table to find the value, its time complexity is a constant O(1).



B tree

The B tree is used to reduce the number of disk accesses.

It is basically a self-balancing tree, we can easily access, search, insert or delete datas in the B tree.



Data Structure from C++

The databases used in the code:

Vector:
Vector in our code is to save the data like the lyrics of the music as strings, after we save these data, we can traverse and access them using iterators. Lookup based on index as well as append is O(1), while erase and insert is O(n).

unordered _map:
The unordered_map is used to save the elements randomly with Key value and mapped value. The key value is used to identify the element, and mapped value is the content associated with the key value.
Map:
Compared with the unordered_map, the map is an ordered sequence of unique elements with key value and mapped value. The underlying data structure for implement map is red-black tree, which is a similar data structure comparable with B tree.

Implementation Detail

There are two hard points for me during implementation

The first one is building an invert index, if the element exists, push it into the vector directly. However, when the elements don't exist, it is hard for me to find a method to save these non-exist elements which creates a new int vector.

The second hard point is when I try to compute the score.
There are lots of elements which are related to each other, it is hard for me to remember the relationship between these elements especially when to reverse the key value and the mapped value. A simple example here will be very helpful. You can test your code by using some simple datas.

Project Delivery

The file “search_engine.cpp” trains the Hash Table and B tree in C++
The processes of my code can be broken down into following steps:

1. Defined the loadWordLibrary function to save segmentation lexicon
2. Defined the loadStopWords function to save unnecessary chinese characters thesaurus.
3. Defined the loadData to save the music lyrics part
4. Defined the splitWords to split the lyrics into several words.
5. Defined the deleteWords to delete the words in unnecessary chinese characters thesaurus.
6. Defined the buildInvertIndex to build an invert index.
7. Defined the computeScore to compute the frequency of the words.
8. Load the data and save the input from the command line.
9. Get the recommandation.

Conclusion

This project helps me be more familiar with the C++ language, especially two essential parts in C++ library, the Hash Table and the B tree. These two methods in C + + are always used to deal with huge amounts of datas. Hope I can have a good command of these two methods and use them in other complex problems.