PROJECT REPORT

AUTO-SUGGEST IN DICTIONARY AND MUSIC CATALOGUE USING TRIES

PRINCE KUMAR 19BCE1846

SHREYANSH DOKANIA 19BCE1764

ABHISHEK SINGH 19BCE1383

V SAI ROHIT 19BCE1848

CSE 2003

DATA STRUCTURES AND ALGORITHMS

Under the guidance of

Mam Richa Singh B V

2020-2021

B. Tech

in

COMPUTER SCIENCE AND ENGINEERING

OBJECTIVE

My project is based on a special tree-based data structure known as "trie and it can be used for multiple applications. Because of the special properties of a trie and the way it is constructed, it enables very fast prefix-based retrieval. It is particularly suitable for application where you want a list of auto-suggestions when you type a partial word. The two major applications that have been combined in this project are 1) dictionary and 2) music playlist population and fast look-up. The purpose of this was to show the flexibility of the trie data structure to be used in diverse applications that would benefit from a fast look-up. The user interface allows the choice of either entering the dictionary application or the music search mode. Once you make the choice the software reads in the dictionary or the music playlist. It does not hardcode either of them. Instead it allows the user to use any English dictionary or custom set of words as long as it is stored in the expected input filename. Similarly the music playlist can be completely customized by the user. The program expects all the song, album and artist names to be present in 3 separate files with specific file names, which it will read in. These can be as concise or extensive as desired by the user. After reading in the input files the user can enter a few letters and the application will fetch all the words starting with that prefix. In case of the dictionary it will also allow inserting more words. In case of the music playlist, upon entering the prefix, the application will fetch all the music albums, songs and artists that begin with the prefix. For this purpose 4 different tries are used.

INTRODUCTION

After thinking about what kind of application program I wanted to create, I did research into the existing work in both the application areas as well as the typical data structures that are used. I chose "tries" to be the data structure of choice and learnt more about how they work and the different implementation choices. The program covers 2 different use-cases – to read in and give autosuggestions for dictionary words and another use-case to read in a music playlist and get auto-suggestions on those. The latter application also enables cross-querying the different categories of the music playlist – albums, songs and artists and will display results from all 3, which match with the inputted prefix. The rest of this document goes into more detail about the project and the background research done.

IMPLEMENTATION:

```
D:\DSA J COMP\run1.exe
1. Dictionary operations
2. Music Finder
3. Exit
        Enter your choice: 1

Successfully populated the trie with the words in Dictionary.txt


a. Input words into a trie
b. Check whether a word is present in the trie
c. Search for words by showing autocomplete suggestions
x. Exit

        Enter your choice: a


Enter the word or phrase to be inserted into the trie: enter
enter has been successfully inserted

a. Input words into a trie
b. Check whether a word is present in the trie
c. Search for words by showing autocomplete suggestions
x. Exit

        Enter your choice: b


Enter the word to be searched for: enter
enter is present in the trie

a. Input words into a trie
b. Check whether a word is present in the trie
c. Search for words by showing autocomplete suggestions
x. Exit

        Enter your choice: c


Enter search keyword: r
Search results:

        race
        radio
        rail
        rain
        raise
        ran
        range
        rather
        reach
```

D:\DSA J COMP\run1.exe

```
        rather
        reach
        read
        ready
        real
        reason
        receive
        record
        red
        region
        remember
        repeat
        reply
        represent
        require
        rest
        result
        rich
        ride
        right
        ring
        rise
        river
        road
        rock
        roll
        room
        root
        rope
        rose
        round
        row
        rub
        rule
        run

a. Input words into a trie
b. Check whether a word is present in the trie
c. Search for words by showing autocomplete suggestions
x. Exit

        Enter your choice: x


1. Dictionary operations
2. Music Finder
3. Exit
        Enter your choice: 2
```

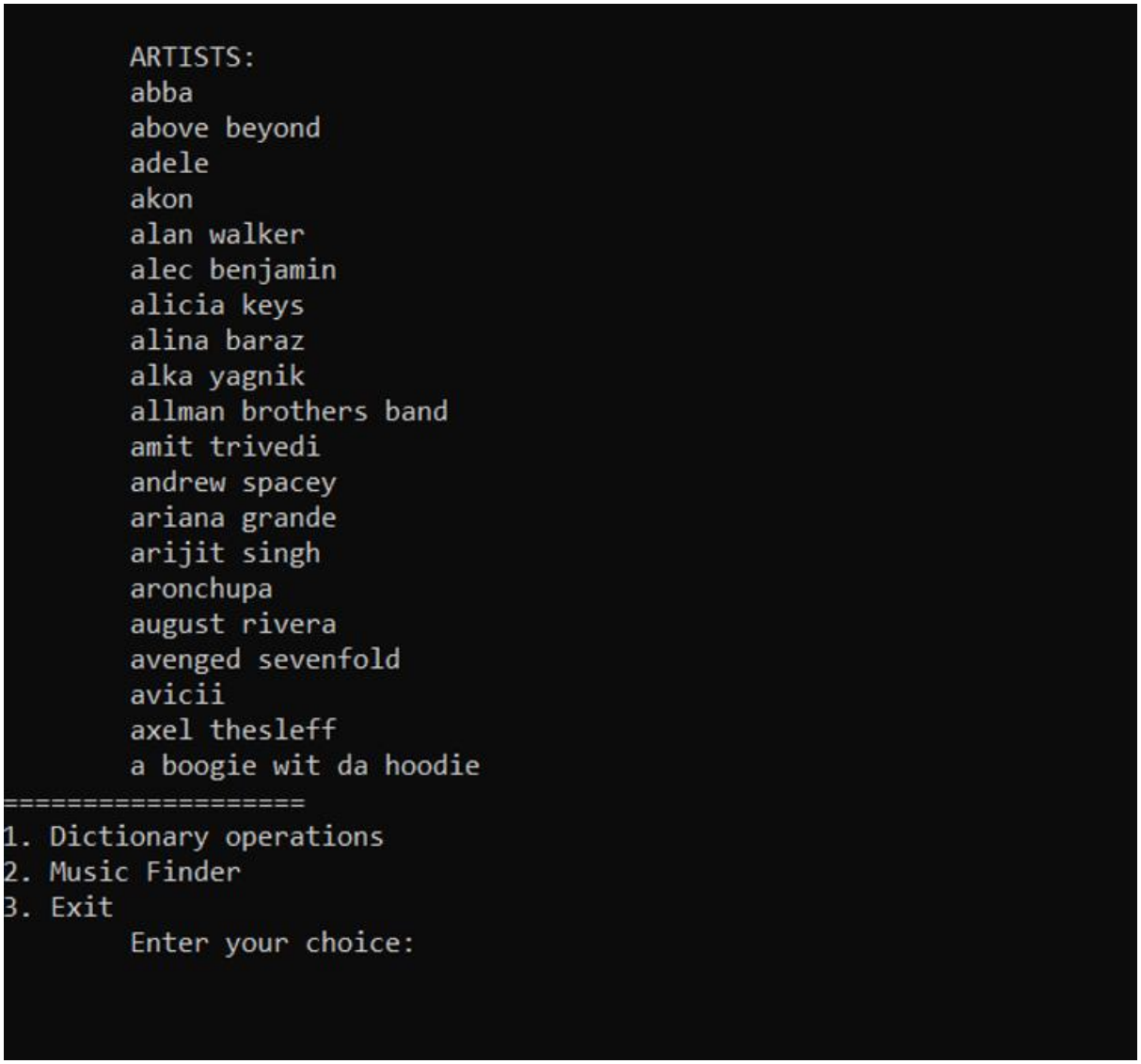
```
D:\DSA J COMP\run1.exe

Enter the search keyword for an album, song or artist: a

==================
        ALBUMS:
        abbey road
        after the gold rush
        agnnepath
        all monsters are human
        all things must pass
        alter ego
        american beauty
        america idiot
        anjaana anjani
        anubhav
        appetite for destruction
        artichrist superstar
        astronomy
        at the heart of winter
        aurora
        autoamerican
        automatic for the people
        a mometary lapse of reason
        a night at the opera
        a wave of golden things

        SONGS:
        aal izz well
        addicted
        after hours
        agar tum saath ho
        alive
        alone
        american teen
        another sad love song
        antisocial with travis scott
        a world of chaos feat rxseboy jomie  ivri

        ARTISTS:
        abba
        above beyond
        adele
        akon
        alan walker
        alec benjamin
        alicia keys
        alina baraz
        alka yagnik
        allman brothers band
        amit trivedi
        andrew spacey
        ariana grande
```



USE CASES

- Dictionary operations
- Word input using trie
- Word check using trie
- Auto suggest
- Music Finder

EVALUATION

A trie, also called digital tree or prefix tree, is a kind of search tree—an ordered tree data structure used to store a dynamic set or associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated; i.e., the value of the key is distributed across the structure. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Keys tend to be associated with leaves, though some inner nodes may correspond to keys of interest. Hence, keys are not necessarily associated with every node.

Why Trie?

A trie has the following advantages over a hash table:

- Looking up data in a trie is faster in the worst case, O(m) time (where m is the length of a search string), compared to an imperfect hash table. An imperfect hash table can have key collisions. A key collision is the hash function mapping of different keys to the same position in a hash table. The worst-case lookup speed in an imperfect hash table is O(N) time, but far more typically is O(1), with O(m) time spent evaluating the hash.
- There are no collisions of different keys in a trie.
- Buckets in a trie, which are analogous to hash table buckets that store key collisions, are necessary only if a single key is associated with more than one value.
- There is no need to provide a hash function or to change hash functions as more keys are added to a trie.
- A trie can provide an alphabetical ordering of the entries by key.

A trie has the following advantages over a binary search tree (BST):

- Looking up keys is faster. Looking up a key of length *m*

takes worst case O(*m*) time. A BST performs O(log(*n*)) comparisons of keys,
where *n* is the number of elements in the tree, because lookups depend on the depth of the tree, which is logarithmic in the number of keys if the tree is balanced. Hence in the worst case, a BST takes O(*m* log *n*) time. Moreover, in the worst case log(*n*) will approach *m*. Also, the simple operations tries use during lookup, such as array indexing using a character, are fast on real machines.

- Tries are more space efficient when they contain a large number of short keys, because nodes are shared between keys with common initial subsequences.
- Tries facilitate longest-prefix matching, helping to find the key sharing the longest possible prefix of characters all unique.
- The number of internal nodes from root to leaf equals the length of the key. Balancing the tree is therefore no concern.

Some disadvantages with Trie :

The main disadvantage of tries is that they may not be the most efficient choice in terms of memory. For each node we need to have as many node pointers as the number of characters in the alphabet which is wasteful as we go lower down in the tree. To make this more space efficient, we can consider another data structure like the Ternary Search Tree in a future revision of this project. In Ternary Search Tree, the time complexity of search operation is O(h) where h is the height of the tree. Ternary Search Trees also supports other operations supported by Trie like prefix search, alphabetical order printing, and nearest neighbour search.

After researching tries I decided to use it for my application.

In my application there will be a menu where the user can choose

one of 3 modes – the dictionary mode, the music search mode and a choice to exit the application.
In the dictionary mode the user first gets to read in the dictionary words from a file. The file name is expected to be Dictionary.txt. The format of this file should be one word per line. This file is first read in and the dictionary trie is populated with all the words. Once all the words are inserted then the user can choose an option to either search for an existing word or add a new word to the trie. If searching for an existing word, then the user doesn't have to type the entire word. You can type a few letters and hit enter and the program will list all the words beginning with the letters. If you want to insert an additional word, you can choose that option and type the word to be inserted. That word will then get inserted in the correct place in the trie and will be available for query in future searches of the current session.

The other mode enabled is the music search mode. Here the user is expected to have his music playlist arranged over 3 files called albums.txt, songs.txt and artists.txt, where each of these files will contain the names (one per line) of all the albums, songs and artists respectively of their playlist. This will get read into 3 tries, one for each of these. Then the user can do their query using a partial word as a keyword. The application will then retrieve all the albums, songs and artists that begin with the typed in prefix.

The above operations can be repeated as many times as the user desires.

CONCLUSIONS

A software program was developed that allows reading in and doing prefix- based queries on both dictionary and a music playlist. It was able to successfully do both. The trie data structure was used effectively in achieving an efficient storing and retrieval mechanism for both the dictionary and music playlist.

The following are some improvements that could be made in a future version of this application:

- Allowing user to delete entries in dictionary
- Allowing user to add or delete entries in the albums, songs or artists collection
- Better error checking
- Better user interface. A graphical user interface would be ideal.

References

Books

- *Introduction to Algorithms* by Cormen, Leiserson, Rivest and Stein
- *C++ Primer Plus*, by Stephen Prata

Internet sources

- https://en.wikipedia.org/wiki/Trie
- http://web.stanford.edu/class/archive/cs/cs166/cs166.1146/lectures/09/Small09.pdf
- https://www.geeksforgeeks.org/data-structures/ · https://en.wikipedia.org/wiki/Hash_table
- https://en.wikipedia.org/wiki/Binary_search_tree
- https://www.programiz.com/dsa/binary-search-tree