Moving Search Engine using Trie

November 21, 2021

Rahul Vinod (2020MCB1244) ,
Ayushi Patel (2020CSB1080) ,
Armaan Umesh Sharma (2020CSB1039)

Instructor:
Dr. Anil Shukla

Teaching Assistant:
Napendra Solanki

Summary: Our project is a based on the autocompletion of word using the data structure trie. It is a search engine which predicts the names of movie when we search and then based on selection of movie, the details of that particular movie will be displayed. We have used trie insertion and search algorithm for implementing our idea.

---

1. Introduction

Our Project is a Movie Search Engine we have implemented it by using Trie data structure. The project is about implementing an autocomplete feature using trie. The user will enter the starting few letters of any movie he wants to search for and the program will predict the possible three options from the data that we have collected. When you will select one movie out of these options the name of the movie along with its rating will be displayed.

Let me explain the idea in a more detailed way. Suppose we have following movies stored in our data file -
Terminator 1, 6.8, 'description'
Transformers: The last knight ,8.9, 'description'
Transformers: Revenge of the fallen, 7.8 ,'description'
Transporter, 5.6, 'description'
Terminator 2, 6.7, 'description'

Now we search by entering "Trans*", then we will get three options to choose from -
1.Transformers: The last knight
2.Transformers: Revenge of the fallen
3.Transporter
Suppose, we select 2. then output will be -
Name : Transformers: Revenge of the fallen, Rating : 7.8
'Description of the movie'
This is the what our project is all about.

2. Figures, Tables and Algorithms

2.1. Figures

Given below is the pictorial representation of trie. As we can see, we have a root node and nodes arising from it. In the picture

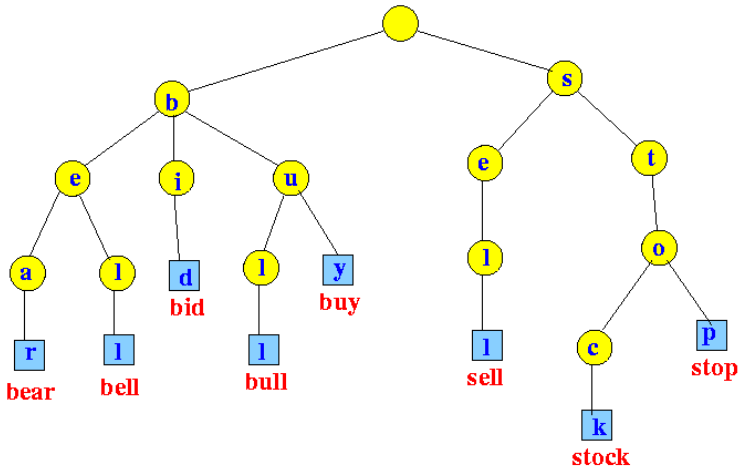

Figure 1: Trie structure.

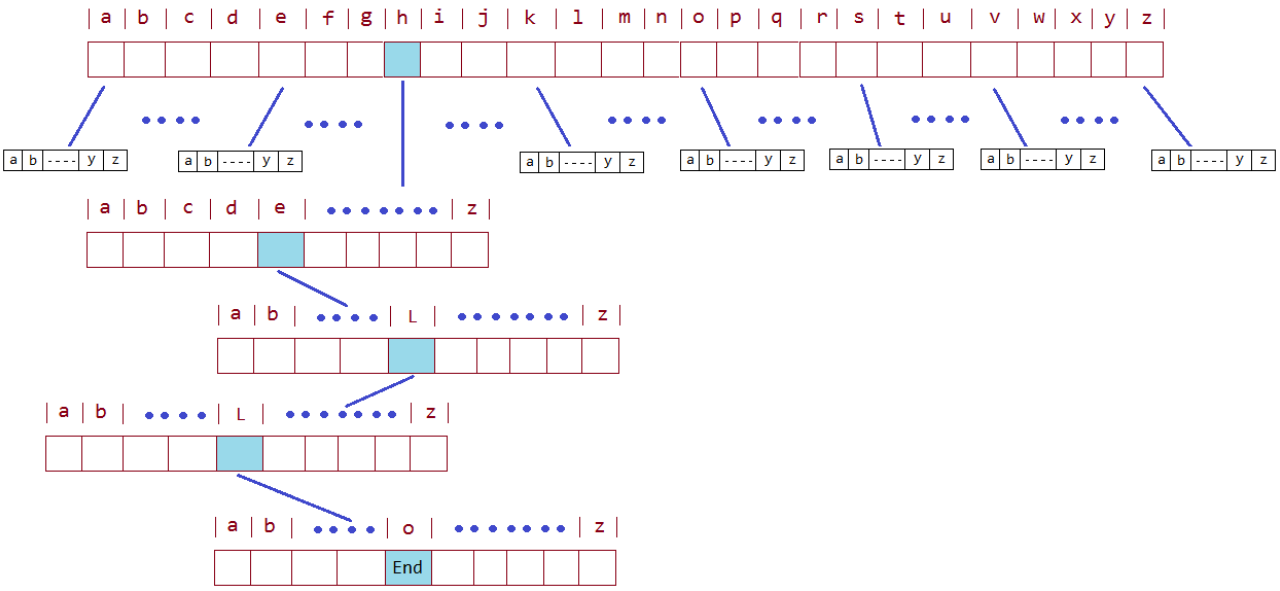

Figure 2: Trie structure showing how hello is stored.

2.2. Algorithms

We have used the trie data structure in our project. A TRIE, also known by the name digital tree is similar to a search tree—an ordered tree data structure. Trie is used to store a dynamic set or associative array where the keys are usually strings. In our project, the names of the movies are 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. 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.

INSERTION IN TRIE
Every character of input key is inserted as an individual Trie node. Note that the children is an array of pointers (or references) to next level trie nodes.
The key character acts as an index into the array children.
If the input key is new or an extension of existing key, we need to construct non-existing nodes of the key, and mark end of word for last node.
If the input key is prefix of existing key in Trie, we simply mark the last node of key as end of word.
The key length determines Trie depth.

SEARCHING IN A TRIE

Searching for a key is similar to insert operation, however we only compare the characters and move down.
The search can terminate due to end of string or lack of key in trie.
In the former case, if the isEndofWord field of last node is true, then the key exists in trie.
In the second case, the search terminates without examining all the characters of key, since the key is not present in trie.
Here, in this program, we use this search feature to find if the query string, which could be incomplete, exists in the trie, by examining all the characters in the string.
Again, the search can terminate due to end of string or lack of the query string in trie.
Once it reaches the end of the string, we find the first 3 strings with the current prefix lexicographically, by using a recursive function to iterate through the node's children.

Algorithm 1 Trie Insert

```
Initiailise i = 0
for ilessthanlengthofstring do
    if childcorrespondingtoithcharacterdoesnotexistalready then
        Initialise the child node
    end if
    Move pointer to child node and increment i
end for
Mark isEnd field of current node as true
Store the rating and description of the movie at this node
```

Algorithm 2 Trie Search

```
Initiailise i = 0
for ilessthanlengthofstring do
    if childcorrespondingtoithcharacterdoesnotexist then
        Return does not exist
    end if
    Move pointer to child node and increment i
end for
Call Top 3 function to get the top 3 strings with the queried prefix
```

Algorithm 3 Top 3

```
if 3wordsfound then
    Return all the strings
end if
if isEndfieldofcurrentnodeistrue then
    Store the details of the current node
end if
Initialise i = 0
for ilessthantotalnumberofpossiblechildrenofanode do
    if ithchildexists then
        Store character corresponding the the child
        Repeat this algorithm for the child
    end if
end for
```

3. Some further useful suggestions

We can also add some more features to the project-

FILTERING : We can also add filters to classify movies on the basis of genre, rating, Box office collection, etc.

CASE INSENSITIVITY : We can also remove case sensitivity from our project. For example when we search for "Tra" and "tra" results are different but after making it case insensitive results will be same for both.

4. Conclusions

We have successfully implemented trie data structure in our project. While working on our project, we implemented a Binary Search Trees along with the trie to search for the info on the movies, but we quickly came to realise that it increased the time complexity of the search and hence it's use is redundant. So, we resorted to rely only on tries in order to make our program more efficient.

5. Acknowledgement

It's a matter of great pleasure to express our deep sense of thanks and gratitude to our course instructor Dr Anil Shukla for giving this wonderful opportunity to us for testing our knowledge and concepts that we learned in our CS201 course. His dedication towards students is quite evident from the way he teaches us. His keen interest in the subject and his innovative methods of teaching have helped us a lot. Without his guidance and knowledge, it wouldn't have been possible for us to make this project.
A special thanks to our mentor Mr Napendra Solanki who guided and helped us with this project. His help at the initial stage of the project was very useful.

6. Bibliography and citations

While doing this pages we did decent amount of research and following are some sources which we referred to.
[1] Introduction to Algorithms 3rd edition, by Cormen, Leiserson, Rivest, and Stein
[2] The C Programming Language by Kernighan and Ritchie
[3] Data Structures and Algorithm Analysis in C, by Mark Allen Weiss
[4] CLRS BOOK
[5] https://www.geeksforgeeks.org/trie-insert-and-search/
[6] https://www.programiz.com/c-programming/c-file-input-output

A. Appendix A

A1. WHAT IS A TRIE? A Trie is an advanced data structure that is sometimes also known as prefix tree (or) digital tree. It is a tree that stores the data in an ordered and efficient way. We generally use tries to store strings. Each node of a trie can have as many as 26 references (pointers). Each node of a trie consists of two things:
• A character
• A boolean value is used to implement whether this character represents the end of the word.
Tries in general are used to store English characters, hence each character can have 26 references. Nodes in a trie do not store entire keys, instead, they store a part of the key (usually a character of the string). When we traverse down from the root node to the leaf node, we can build the key from these small parts of the key. A trie provides advantage in time at the cost of space. If we store keys in binary search tree, a well-balanced BST will need time proportional to M * log N, where M is maximum string length and N is number of keys in tree. Using Trie, we can search the key in O(M) time.

B. Appendix B

A2. ADVANTAGES OF A TRIE When we talk about the fastest ways to retrieve values from a data structure, hash tables generally come to our mind. Though very efficient in nature but still very less talked about as when compared to hash tables, tries are much more efficient than hash tables and also, they possess several advantages over the same. Mainly: • There won't be any collisions hence making the worst performance better than a hash table that is not implemented properly. • No need for hash functions. • Lookup time for a string in trie is O(K) where k = length of the word. • It can take even less than O(K) time when the word is not there in a trie.

C. Appendix C

A3. DISADVANTAGES OF A TRIE The main disadvantage of tries is that they need a lot of memory for storing the strings. For each node we have too many node pointers (equal to number of characters of the alphabet), if space is concerned, then there are other data structures which can be preferred for dictionary implementations. Although, if we want to store metadata along with the word, trie is still the best option.