The Trie Data Structure

Presented by: Angel Pichardo De La Cruz, Jacob Duhaime, Nicholas Faciano and Harry Grenier.

CSC 212 - Data Structures and Abstractions
Prof. Jonathan Schrader
December 4th, 2022

What is Trie?

Trie (pronounced "**Try**", comes from the word "re**trie**val") is a k-ary search tree data structure that is used for sophisticated types of data, such as strings. In this data structure, the links between the nodes are defined by individual characters from a string in which every child node is associated with its parent node. With Trie, you can do operations such as insertion, deletion and searching. A simplest way to explain this data structure is as an array of pointers to other nodes for example:

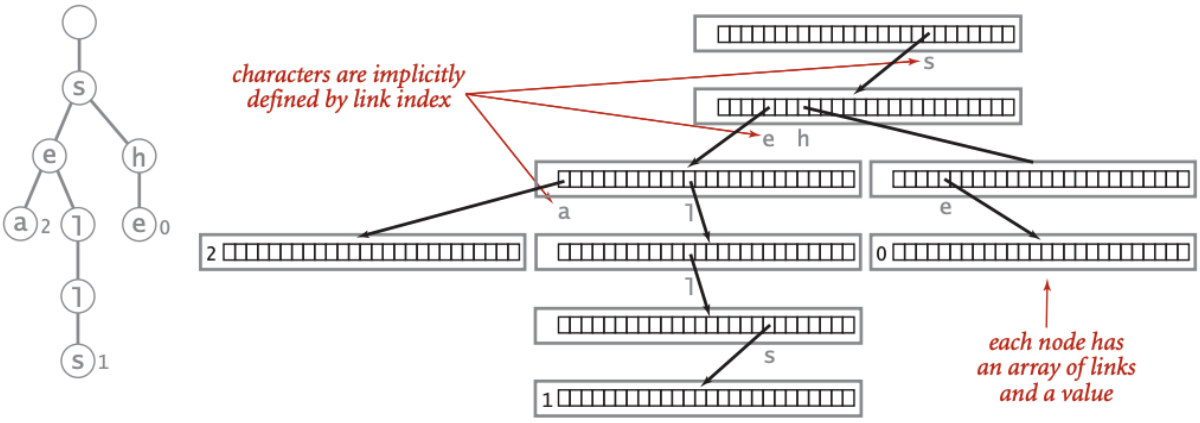

Image credits to wikipedia "Trie".

History of Trie

The idea of the Trie comes from its derivation word - retrieval. The earliest example of this concept which dates back to its creation is that of Axel Thue. Not much is known about the exact implications Thue was implying but what is known is that in 1912 he represented a set of strings abstractly with the use of his tree - 'trie'. Thue is a mathematician and this was at a time before

computers. 50 Years later rediscovered by Rene de la Briandais (1959) this idea was brought back to the public eye, again in an abstract manner. This manner was hard to understand but was still much more concrete than that of Thues. A year later, Edward Fredkin also formulated a concrete idea of the creation and use of Trie. Fredkin is more famously credited with the creation of the Trie since he coined the term after the word retrieval - there had previously been no name for this data structure. He also published a more famous paper than Briandais.

Tries Explained - Methods

Hypothetically, let's state that we are using trie for a spell checker in which we are trying to find the correct spelling of a word. A user would input the word “Codign” (for coding). The data structure will spawn in the first node which will be for character “c”. Then it would make another node with the letter “o” while still being pointed at by the previous node. Then, another node is generated with the letter “d”, then letter “i”. By this point of having the characters “codi”, it can do a search for all possible words that start with those 4 characters like codify, **coding**, codirect…etc.from a database provided to it. Essentially, it would pull out all of the words in the dictionary that can be a match for the word that we are looking for. Please look example below which shows how Trie looks for the possible matches:

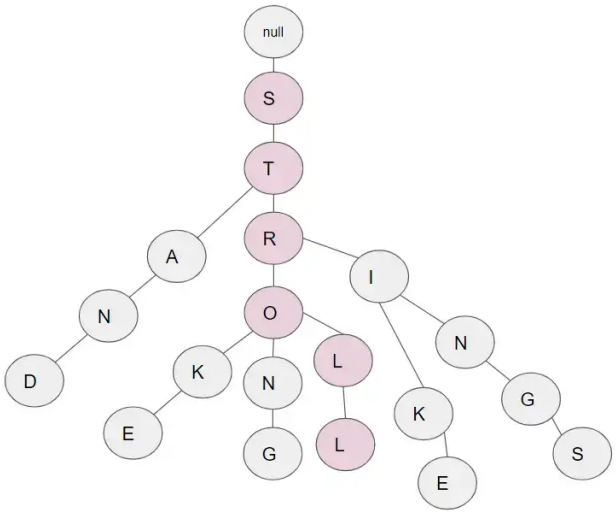

Image credits to Akanksha Parmar "Understanding Trie Data Structure"

Anything interesting about this data structure?

Absolutely! A fun fact about Trie is that time is really constant when it comes to this data structure. The number of other items in the data structure **does not impact** the steps it takes in order to find a match. The only thing that impacts this data structure is the length of the string itself, for example: if we are looking for the word "cat" in a library of 10 words vs in a library of 10,000 words, the time it would take to find the word Cat is the same. Which brings us to the next topic of time complexity.

Time complexity

Trie as a data structure allows for the following functions which are: insertion, deletion, and searching. Using Trie, you can search things in O(M) times. Trie insertion and search costs formula can be seen as O(key_length) and the memory that it takes is O(ALPHABET_SIZE *

key_length * N). Essentially the only thing that affects Trie would be the size of the word that you are using. While Trie seems to be using heavy resources, the only thing it is heavy on is storage. Due to the nature in which this data structure works, you are essentially spawning in a new node every time we go past 1 character. In simple terms if your word contains 5 letters, you are spawning the alphabet 5 times and it would take up 130 bits of data (for every letter of the alphabet 5 times). That is the only disadvantage of the Trie data structure, the fact that it needs so much memory for storing all of the information.

Our Project - Implementation

For our final project, our team has decided to build a spell checker using Trie as a data structure. The way in which it would work is that the user would input a word into the program and the program is going to take that word and divide it into nodes and it will identify the proper spelling for the word. As a bonus, it will pull out words that may match the spelling of the word or similar words to the one that the user inputs into it. Note: You **must** have a "dictionary.txt" file that is full of strings in order for the program to word. This file would be the said dictionary in which the algorithm will be looking for the word in. Please look at the github in order to get the files for the code here.

In order to download and compile the code, you would have to download 4 files from our github repository. The files are: "main.cpp","spellcheck.cpp","spellcheck.h" and "dictionary.txt". Once you have downloaded these files, you would open the IDE of your choice (preferably VSCode) and you would open the terminal to the location of where you stored the files. For example, if your files are located in the downloads folder, what you would do in your terminal is

“cd/downloads”. After being in the same directory where your files are, you would input the following into the terminal in order to compile the code: “g++ spellcheck.cpp main.cpp”. After the code is compiled, an “a.out” file should be generated. To run the code, you would type “./a.out” and this will trigger the program to run. Once the program is running, you would input a word that you wish to search for. If the word contains a typo, the program will pull out all of the words with similar spelling to the word you typed. In order to find another word after getting the results, retype “a.out” again on the terminal and the program will run once again.

Contributions

Angel: My main contribution to this project was doing background research about the data structure, writing the report as well as helping create the slide show presentation. I also worked on helping shape the github and contributed by bringing ideas to the table on how to execute the code from our data structure.

Jacob: For the majority of the project I worked on the programming aspect of this project. To do this efficiently I was required to do extensive background research on the trie data structure. I also set up the git hub and kept our group organized and on time.

Harry: Helped with the slide show along with understanding the dot visualization. For this document my main contribution was the work cited along with checking for valid info throughout the presentation. We all contributed our own ideas into the group and figured out the most efficient way to go about this project. Along with extensive research on the datastucture itself.

Nicholas: Helped create the slideshow and paper. Assististed with the source code. Set up what data structure we would choose and the presentation date. Communicated with group members in our strategy of attacking assignment/ getting the project done in time. Assisted with Dot Visualization.

Work Cited

1) Babu, Sanjana. “Time and Space Complexity of Trie.” *OpenGenus IQ: Computing Expertise & Legacy*, OpenGenus IQ: Computing Expertise & Legacy, 21 Dec. 2021, https://iq.opengenus.org/time-complexity-of-trie/.
2) “C++ Implementation of Trie Data Structure.” *Techie Delight*, 1 May 2021, https://www.techiedelight.com/cpp-implementation-trie-data-structure/.
3) Jaisingh, Anand. “Trie (Keyword Tree) Tutorials & Notes: Data Structures.” *HackerEarth*, https://www.hackerearth.com/practice/data-structures/advanced-data-structures/trie-keyword-tree/tutorial/.
4) Ojha, Ravi. “1-D Tutorials & Notes: Data Structures.” *HackerEarth*, https://www.hackerearth.com/practice/data-structures/arrays/1-d/tutorial/.
5) Parmar, Akanksha. “Understanding Trie Data Structure.” *Medium*, Guidona, 25 Jan. 2021, https://medium.com/guidona-softpedia/understanding-trie-data-structure-39e7652114d5.

6) Ram, Vijaykrishna. “Trie Data Structure in C/C++.” *DigitalOcean*, DigitalOcean, 3 Aug. 2022, https://www.digitalocean.com/community/tutorials/trie-data-structure-in-c-plus-plus.
7) Ram, Vijaykrishna. “Trie Data Structure in C/C++.” *DigitalOcean*, DigitalOcean, 3 Aug. 2022, https://www.digitalocean.com/community/tutorials/trie-data-structure-in-c-plus-plus.
8) Tiwari, Sakshi. “Advantages of Trie Data Structure.” *GeeksforGeeks*, 22 Jan. 2021, https://www.geeksforgeeks.org/advantages-trie-data-structure/.
9) “Trie Data Structure: Interview Cake.” *Interview Cake: Programming Interview Questions and Tips*, https://www.interviewcake.com/concept/java/trie.
10) “Trie.” *Wikipedia*, Wikimedia Foundation, 21 Nov. 2022, https://en.wikipedia.org/wiki/Trie.
11) “Trie: (Insert and Search).” *GeeksforGeeks*, 10 Nov. 2022, http://www.geeksforgeeks.org/trie-insert-and-search/.