Kadir Kaan Yazgan
Barış Özçelikay

CMPE 382

Operating Systems Project #3 Report

Creating Dictionary for the AOL Dataset

1. ABOUT OF THE PROJECT

Our goal is building a dictionary named "Dictionary.txt" with checking querys of ten data files.In other words, to explain in more detail, we send the number of times the queries are repeated in the data to the file named "Dictionary.txt" and keep the number of repetitions and the word itself.

We perform these operations using the trie structure. We apply the same process with the tasks requested from us with several different methods and compare them according to their effectiveness. We also imported this project to github. You can find the necessary files and contents via the link below.

our github link: https://github.com/barisozcelikay/CMPE382_PROJECT3

There is Makefile to create all programs: trie2, trie3, trie4, trie5, and trie6.

For compiling individually you can just write "make 'one_of_trie_name'."

For compiling you should write "make". After that, you need to type "./trie + (task number)" for the task you want to run. Like, "./trie2" for task 2, "./trie3" for task 3.

P.S => Since the size of the data files is very large, it both slowed us down while working on it, was very challenging for computers and caused us to get errors. so we reduced the size of the datala file. Each data file contains 40,000 lines.

2. TASKS

TASK 1 (trie.c)

It is the first task of our project. In this task, we read the data and set up the trie data structure.Thanks to this structure, we provide keep track of the queries.

Firstly, we create 'struct Trie' for storing trie nodes. This struct contains 2 variables. One of them is a type of integer that checks the node if it is leaf then it's value is more than 1, otherwise 0. Another variable is type of Struct Trie which recurs the same struct with initialized String array.

To initialize first node(head) on trie, we call **getNewTrieNode()** method for creating a new Trie node which is dynamically allocated. Then we create **insert()** method to iterate our trie structure. This method gets trie structure and string parameters for inserting a string into a trie. For searching a string in a trie, we created the **search()** method. If string is not found returns 0 and If string found in the trie returns 1. Also we have **hasChildren()** method.

This method is make control for given trie node has any children or not. We can provide control on children with this method. For making deletions, we created method named by **deletion().** This method is a recursive method for deleting string from a trie. Also we have two different write methods, one of them is **print_string(),** this method write Trie node to Dictionary file. and other one is **print_all_node(),**write all Trie's node to Dictionary file. And our last method for task 1 is **delete_node()** method. This method gives us opportunities for deleting Trie node and releasing the allocated memory.

TASK 2 (trie2.c)

It is the second task of our project. In this task we use the methods from task 1. The Program reads from files, line by line, and inserts it to Trie structure. Once the reading process is done, we have the Trie structure in memory, and we can start to create a Dictionary file by retrieving all data in the Trie data structure and write it to output file.

To execute the program you should call program with: "./trie2".

Execution time for task 2.

TASK 3 (trie3.c)

This is the third task of the project. This task is very similar to task 2. Program for task 3, instead of read per line it will read per block (2048 bytes). From block data we need to parse it to get the query and insert to Trie data structure. The rest of process similar to program trie2. So we are performing single read operation for multiple queries in place of execute one. We used the methods we defined in task 1 and task 2 for this task too.

To execute the program you should call program with: "./trie3".

```
kaan@DESKTOP-K0HCDS6:/mnt/c/Users/NEW/Desktop/CMPE382_Project3$ time ./trie3
Creating dictionary ...
processing data1.txt ..
processing data2.txt ..
processing data3.txt ..
processing data4.txt ..
processing data5.txt ..
processing data6.txt ..
processing data7.txt ..
processing data8.txt ..
processing data9.txt ..
processing data10.txt ..

real    0m6.007s
user    0m3.828s
sys     0m2.109s
```

Execution time for task 3.

According to results we examined that task3 is faster than task2. You can see differences from execution times.

TASK 4 (trie4.c)

In this task, we proceeded by continuing with our previous implementations. Our goal in this task is to create a thread to process each input file read. Since we have 10 input files, we created 10 threads for the process. We have a Trie data structure and all threads must have access to the same data, and our other purpose is to provide these threads with synchronous access to data.

In this task we solved this problem by using the 'pthread.h' library and the mutex's lock system. In addition, we have created a separate method and structure for process operations, the process_task() method and the targ_t struct.

To execute the program you should call program with: "./trie4"

```
kaan@DESKTOP-K0HCDS6:/mnt/c/Users/NEW/Desktop/CMPE382_Project3$ time ./trie4
Creating dictionary ...
task 2, processing data3.txt ...
task 0, processing data1.txt ...
task 4, processing data5.txt ...
task 5, processing data6.txt ...
task 3, processing data4.txt ...
task 1, processing data2.txt ...
task 7, processing data8.txt ...
task 6, processing data7.txt ...
task 8, processing data9.txt ...
task 9, processing data10.txt ...
All threads were done.
writing to Dictionary.

real    0m9.985s
user    0m4.516s
sys     0m6.234s
```

Execution time for Task 4

TASK 5 (trie5.c)

In this task we used multi-thread to process data, but we consider that each thread has it own Trie data structure. Once all threads finished their job, the main program will combine all local Trie structures into one main Trie structure and then we write to the output file which name is Dictionary.txt.

To execute the program you should call program with: "./trie5"

```
kaan@DESKTOP-K0HCDS6:/mnt/c/Users/NEW/Desktop/CMPE382_Project3$ time ./trie5
Creating dictionary ...
task 0, processing data1.txt ...
task 1, processing data2.txt ...
task 2, processing data3.txt ...
task 5, processing data6.txt ...
task 3, processing data4.txt ...
task 4, processing data5.txt ...
task 6, processing data7.txt ...
task 7, processing data8.txt ...
task 8, processing data9.txt ...
task 9, processing data10.txt ...
All threads were done.
Combining all local Tries
writing to Dictionary.

real    0m20.733s
user    0m9.500s
sys     0m11.453s
```

Execution time for task 5.

In this task we examined that Task 5 takes longer time than task 4, because there are 2 level Trie structure. First it creates local structure, then combining all local structures into main structure.

TASK 6 (trie6.c)

This is the sixth task of the project. This task is very similar to task 3. In Task 6, we implement what we did in task 2 and task 3 in a different way. Instead of taking the queries one by one as we did in Task 3, this time we first put all the queries into memory. As stated in the description of Task, we read the data in memory and created the "Dictionary.txt" file without using disk access. When we compared this method with other methods, we realized that this method is the fastest method. It's faster than task3, because we have bigger buffer size, as big as total file size.You can see it by looking at the screenshot below.

To execute the program you should call program with: "./trie6".

```
kaan@DESKTOP-K0HCDS6:/mnt/c/Users/NEW/Desktop/CMPE382_Project3$ time ./trie6
Creating dictionary ...
Read data filis into memory
process queries into Trie structure
write to Dictionary.txt

real    0m5.882s
user    0m4.109s
sys     0m1.734s
```

TASK 7 (Improvements)

Compiling task 5 is very long because firstly we are creating a local structure and then combining all local structures again in main structure and this proccess wastes too much time.If we do it at the same time creating and combining it saves us more time and becomes more effective. We can improve that task with that way.

3. CONCLUSION

As a result, we created our own trie in this project and created an additional dictionary called "Dictionary.txt" with different methods. Thanks to this project, we compared the efficiencies of the different methods we used and learned. According to this; we noticed that task6 is the fastest way to complete because it has a fairly large buffer size as big as total file size. We noticed that the slowest method is task 5 because First it creates local structure, then combines all local structures into main structure.