Project report: Top 10 frequent Words

Course Name: Data Structures with C++ (CMP2003)

Project done by:
Feda Abdulrazzaq Yahya Alashwal - 2018965
Rayyan Ahmad Zuhair Al-haj – 2017741
Omnia Mohamed Mahmoud Sedeek Elmenshawy - 2000007

Bahcesehir University

2021/2022 Fall Semester
20th Jan 2022

This project aims to use data structures algorithms to find the top ten frequent words in a dataset in the shortest running time possible. We received two datasets, the PublicationsDataSet.txt one and the stopwords.txt one, and we implemented an object-oriented code based on one class called **topTen**, one main function to call the class and calculate the top 10 words in the dataset and to calculate the running time, two header files in the name of **hashtabA.h and heapA.h** to implement the data structures of our choice to insert and sort the dataset, and one **hashtabA.cpp** file to implement the hash and insert functions.

Firstly, we declared a class in the name of **topTen** to include all of the needed functions in, the most important part of it is the **startProgram()** function; as it includes all of the following operations:

- Variables declaration (such as: string pub_data, string border).
- Hash table declaration.
- Open file function (to open the PublicationsDataSet.txt and stopwords.txt datasets).
- Declaring of the pub_dataCount variable which is implemented to keep track of valid words in the data file.
- **Exit()** Function to exit the program if the data file is not found.
- **Find()** function to find the **unigramCount** words in the file.
- **Erase(0, find(unigramcount) +15)** function to delete everything before the **unigramCount** word and the word itself which are not supposed to be included in our process.
- Data Preprocessing: separating numbers from words, removing unnecessarily punctuation, lowercasing all of the letters, removing unwanted numbers, removing null values.
- Cleaning stopwords.txt dataset, then using a loop to un count the stopwords while passing it to the **sortHeap** function, which means to skip those words when they are found in the array and not to print them.
- Closing the opened files.

Then comes the rest of our functions as follows:

- **Swap**(): a basic swapping function implemented to swap elements in the sorting process.
- **Heaping_a**(): a function used in the sorting part of the words – explained in the third part of the report.
- **sortHeap**(): a function used to sort the heap array – explained in the third part of the report.
- **getNum**(): a function which converts a string consisting of numbers such as “1234” to an int value: 1234.
- **SplitGetWord**() and **SplitGetNum**(): two functions implemented as part of the preprocessing process, as we decided to use the number in front of each word as an occurrence value and include it in our counting and sorting processes, so, in order to split each word from that number (ex: “a”:1); SplitGetWord() and SplitGetNum() are used to separate them.
- **Valid**(): a function which checks if the tokenized format is correct or not before giving it to the previous two functions, the correct format is “a”: 1.
- **isValidWord**() and **isValidNum**(): Two functions which aim to check if the **pub_data** and **Pub_dataCount** variables have any special characters or not, which is again part of the preprocessing implementation.
- **Trim**(): a function aims to remove trailing white spaces in the dataset using ASCII table variables.

Secondly, We decided to use hashing as the project is asking us to stay as time efficient as possible, and the best case of hashing time complexity is O(1), and the worst case is O(n)-, which acts as a hash table or map for the PublicationsDataSet.txt *file.* The **hashtabA.h** file includes a struct which we used to create objects and store them in the hash table using the **tabhash()** function along with the **inserting_hash_a()** function using linked lists implementations. In Addition, to avoid collision between words in the data in the insertion process; we used the separate chaining method as we found that it is better than linear and quadratic probing in decreasing collisions as it uses linked lists; unlike the other two algorithms, even though chaining isn’t the most efficient solution when it comes to avoiding collisions from a memory prospective; we found that it is our best option regarding time complexity. Our implemented hash table is based upon two functions included in a separated file named as **hashtabA.cpp**, and they work as follows:

1- tabhash() a function used to get the index where the element is to be inserted at. Firstly, it multiplies the index of each character in the word by its ASCII value, then it calculates the sum of them through a for loop to reduce collisions as much as possible before it calculates its modulus with the size of the hash table, then it returns the index of each word to be inserted at. For example, if we have the same characters in two different words as follows:

abc = 65 + 66 + 67 = 198
acb = 65 + 67 + 66 = 198

Then the function calculates each words index as:

abc = 65*0+66*1+67*2 = 200

acb =65*0+67*1+66*2=199

2- **insertting_hash_a()**: A function implemented to insert the values into the hash table. It is called after the **tabhash()** Function and creates a new node if the returned index is empty, however, if the index is not empty, it traverses the chain and compares the element which the current element is pointing to with each node until it finds the same word inserted, when it finds it, it adds the word count for both as the new value in that node, however, if it didn't find the same word, it inserts the word in the first found null place, or it create the node to insert the word in.

   Note1: Please check the sources at the end of the report to check where we got our function idea from.

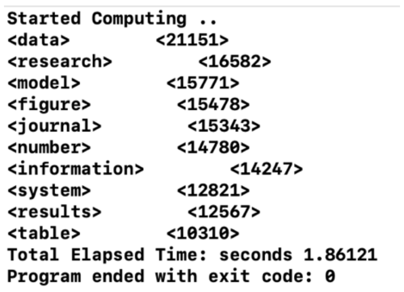
Thirdly, we implemented a heap algorithm for the sorting part in the project, we also tried to use merge sort algorithm as they both have **O(nlogn)** complexity, however, we decided to go with the heap sorting as the code executed faster and it turned out more space efficient. A separate file in the name of **heapA.h** is implemented in our code in order to give a template struct for the array which will be used in the **topTen** class in order to start the heap sorting process. Furthermore, we declared two functions, and one array to get the sorting process done as follows:

1. Array of type **heaping**: Which we declared in the **heapA.h** file; the role of this array is to be called after a loop done on the previously implemented hash table to copy all of its function into the **heaping** array.
2. **Heaping_a()** Function: It is a function which uses recursion to send the largest value of the array to the Index[0], 2nd largest element in the array to the Index[1], and so on to sort the whole array.
3. **sortHeap()** Function: a function which loops over the **heaping** Array from its middle point backward till Index[0] while calling the **Heaping_a ()** function inside it to sort the array, and loops again from the end of the array to its first element to swap and get the sortation done.

Finally, we declared the **main()** function out of the class and call the **topTen** class inside of it along with the time function so that the process goes as follows:
Start the time count, Start the program, read the data, preprocess the data and clean it, declare the hash table, compute the strings and numbers, insert the values in the hash table, call the heap sort algorithm, sort the data, get the top ten values according to their occurrences in our sorted array copied from the hash table, skip the stopwords and print only the top ten values -excluding the stopwords as (defined as **jumpw**)-, end time count, print how many seconds elapsed, end the execution.

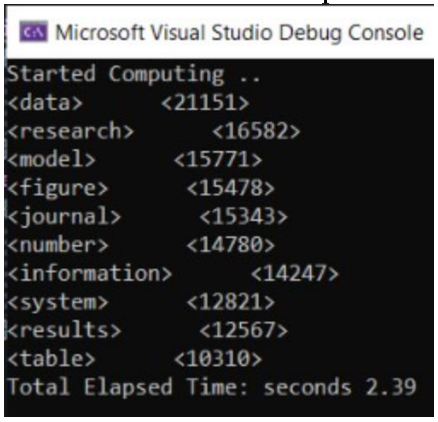
We tried to run the code on three different laptops, each laptop got different running time as follows:

Best Running time 1.86 Seconds:
Screenshot from Omnia's output



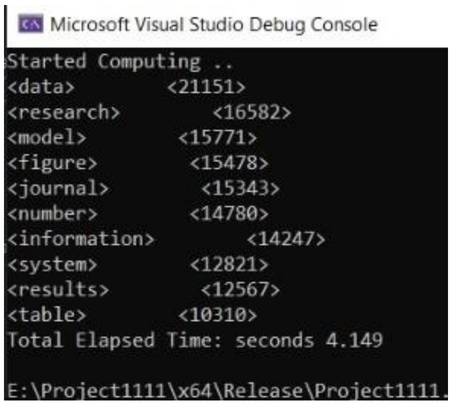
```
Started Computing ..
<data>        <21151>
<research>        <16582>
<model>        <15771>
<figure>        <15478>
<journal>        <15343>
<number>        <14780>
<information>        <14247>
<system>        <12821>
<results>        <12567>
<table>        <10310>
Total Elapsed Time: seconds 1.86121
Program ended with exit code: 0
```

Average Running time: 2.39 Seconds
Screenshot of Feda's output



```
Started Computing ..
<data>      <21151>
<research>      <16582>
<model>      <15771>
<figure>      <15478>
<journal>      <15343>
<number>      <14780>
<information>      <14247>
<system>      <12821>
<results>      <12567>
<table>      <10310>
Total Elapsed Time: seconds 2.39
```

Worst Running Time: 4.1 Seconds
Screenshot from Rayyan's output:



```
Started Computing ..
<data>        <21151>
<research>        <16582>
<model>        <15771>
<figure>        <15478>
<journal>        <15343>
<number>        <14780>
<information>        <14247>
<system>        <12821>
<results>        <12567>
<table>        <10310>
Total Elapsed Time: seconds 4.149

E:\Project1111\x64\Release\Project1111.
```

Note2: We avoided using a data structure to delete or remove the stop words from our dataset as we found that if we just skipped those words before the printing process it will be more time efficient.

Note3: We used the ASCII table decimal values in our data structures logic implementation, please find the meaning of each number we used here:

Buddies, S. (2020, April 1). *ASCII Table - Character codes in decimal, hexadecimal, octal and binary*. Science Buddies.

https://www.sciencebuddies.org/science-fair-projects/references/ascii-table

Note3: The Heap sorting function and the inserting hash implementation ideas are brought from the book and from an online source, we tried to edit them and add on them based on the given dataset, please find them here:

- H., & H. (2021). *HashT*. GitHub. https://github.com/hiamsevval/Top-10-Frequent-Words/blob/main/hashT.cpp
- Malik, D. S. (2012). *Data Structures using C++*. Course Technology/Cengage Learning India.

Note4: Here are some other coding sources which also helped us:

- *String::NPOs in C++ with examples*. GeeksforGeeks. (2021, July 16). Retrieved January 21, 2022, from https://www.geeksforgeeks.org/stringnpos-in-c-with-examples/#:~:text=What%20is%20string%3A%3Anpos,used%20to%20indicate%20no%20matches.

- GeeksforGeeks. (2021, May 11). *Converting Strings to Numbers in C/C++*.

  https://www.geeksforgeeks.org/converting-strings-numbers-cc/