**Analysis of Algorithms**

Spring 2020

**Members Details**

| Group ID | 58 |
| --- | --- |
| Registration Number of Group Members | 2018-CS-60  2018-CS-99 |
| Section | B |

**Project Details**

|  |  |
| --- | --- |
| ***Project*** |  |
| Project Title | Plagiarism Checking Tool |
| Executive Summary | According to our understanding the word “plagiarism” means “to steal, to pass ideas or words”. Since people copy the information from the other content and do not even try to write on their own ideas and describe the idea by their own words, they just go for the easy way and that is copying others ideas.  So we have built a project that checks the similar words, at of any length, in-between the two text documents as the mostly information of any content or report or assignment is written in this format. This project checks the similar English words by matching a single character of a word by other word. If it matches the complete characters of alphabet by other word including the length if the word then it is considered as plagiarized by other. As we keep doing this and words are keep being matched then we come up to the result of matched sentence. We know that there are words which are most frequently used by everyone so do we consider them as plagiarized? The answer is maybe of the come up matched in the same sequence as they are used by other, otherwise not. Because if one sentence has the same words as other and also in the same sequence as in other so definitely they are matched and copied with each other, so it is plagiarized.  The use of this project is mostly in academic field, as students mostly copy each other’s assignment rather than writing their own. Also it can be used as the copy right checking document of an organization. So it is very beneficial for many various fields like this we mentioned. |
| ***Business Case*** |  |
| Outline the business need for the project | In the business field, this project can be used for the copyright checking process. As we can search the similar content within the seconds so it very useful for the documents similar content. Organizations can use this project for checking the copyright content of their any text information. |
| End user of the product | The real life domain of our project is to set up the environment for the user to check weather two or more documents have any kind of similar content. If yes then highlight them and present them as an output for the user. It is mostly used for the academic organizations. |
| Motivation for Project | When we saw the opportunity of making this project by our own we were really excited to build it up for the user as it is very useful to check whether his/her document have any kind of similar content matched/plagiarized with other. |
| Description of the project objective(s) | This project takes the input of text files and compares the words between them and highlight them. Only similar words are highlighted and shown in the output.  In the second option user can select multiple files and highlight the specific file that he/she has chosen. |
| State the level of impact expected should the project proceed and implications of not proceeding | It may impact the user side as it depends upon user to run this project for the use of the plagiarism content only. It may change the writing or creating new idea rather than just copying the content. It may vast the users’ way of writing and thinking too. |
| Functional Requirements | We have implemented the two methods in our project   1. First is that compares the two text files (.txt format) and the highlights the exactly similar words between them. 2. Second is that compares the multiples files or folders selected by user and finds the common/similar words between them all. Then user selects the specific file among them and then it highlights the words of it. The bonus service is that user can select any document and highlight it. |
| ***Benefits*** |  |
| What benefits are expected/ anticipated? | In Academic benefits user can check the assignments or any kind of academic document for its plagiarism checkup. It’s very useful for teachers to check the copied content among the students work assignment. This helps them a lot.  For the industrial use, companies can use this for the copyright checkup. Some organization take the idea rather than building the new one so the pre-built up need to protect their ideas. They can use this for the finding the similar content in it. |
| ***Implementation Details*** |  |
| Link to Github Repository | https://github.com/MuhammadRazaNaqvi/CS311S20PID58.git |
| Total Number of commits in repository before 5th August 2020 | 16+ |
| Exact contribution of each member | 2018-CS-60 (10 commits)  2018-CS-99 (6 commits) |
| ***Commits in github repository by each member*** | |
| |  |  | | --- | --- | | **Member Registration No.** | **Total Commits** | | 2018-CS-60 | 10 | | 2018-CS-99 | 6 | | |
| **Details of commits** | |
| |  |  |  |  | | --- | --- | --- | --- | | **Sr. No.** | **Details of commit** | **Date** | **Member Reg No.** | | 1 | GitHub interaction | 24 June, 2020 | 2018-CS-60 | | 2 | GitHub interaction | 27 June, 2020 | 2018-CS-99 | | 3 | Project Milestone-1 | 4 July, 2020 | 2018-CS-99 | | 4 | Project Milestone-2 | 4 July, 2020 | 2018-CS-99 | | 5 | Project Milestone-3 | 8 July, 2020 | 2018-CS-60 | | 6 | Project Initial stage code | 20 July, 2020 | 2018-CS-60 | | 7 | Project Initial stage code with some required changes | 20 July, 2020 | 2018-CS-60 | | 8 | Project GUI without code interaction | 25 July, 2020 | 2018-CS-60 | | 9 | Project Part-1 complete done with the interface interaction | 25 July, 2020 | 2018-CS-99 | | 10 | Project Part-1 & Part-2 complete done with the interface interaction  (Final Version) | 25 July, 2020 | 2018-CS-99 | | 11 | Project Implementation Guide | 25 July, 2020 | 2018-CS-60 | | 12 | Project Code in zip (final version) | 15 Aug, 2020 | 2018-CS-99 | | |
| Have you used built in algorithms or you have implemented yourself? | We took the idea of Rabin-Karp algorithm but we originally used our technique to compare the words using hashtable technique and matching the words by single character. |
| Formats of input | User can only input the text files with (.txt) extension, in two files option and multiple files option too. Only this format is acceptable. |
| Validations | Only text files with text only, no images can be contained, as this doesn’t check the plagiarism of the images. |
| Format of output | Output is the text file on the box with the highlighted words format of the file. Common words are highlighted in the output. |
| Deployment | No |
| ***Details of algorithms*** | |
| Details of algorithm:  The algorithm we have taken idea from is the Rabin-Karp algorithm but we have done a lot of changes and updation in it with our own ideas by using the hashtable functionality.  Time Complexity:  It can be done in constant space and O(m) time  The time complexity of the searching phase of the Karp-Rabin algorithm is O(mn) (when searching for am in an for instance). As we do have two loop one for the whole word search and other for the whole text file search. So we can multiply this O(mn) with N where N is the number of files. For text character comparisons is O(n+m).  Pseudo Code:  **Function to read the words from the file and putting them into a list:**  public string[] FileWords(string CompleteFile)  {  string str = CompleteFile.Replace("\r", "").Replace("\n", " ").Replace("\r\n", " "); //replacing new line charaters  “goes for the complete file as it reads the file completely for replacing  If ‘N’ is the number of words in a file so it goes for ‘N’ time”  string[] array = str.Split(' '); //splitting file into words  List<string> list = new List<string>();  for (int i = 0; i < array.Length; i++)  {  if(String.IsNullOrEmpty(array[i])||String.IsNullOrWhiteSpace(array[i]))  continue;  else  list.Add(array[i]);  }  return list.ToArray();  }  “goes for the complete file as it reads the file completely for adding words in the list  If ‘N’ is the number of words in a file so it goes for ‘N’ time”  **1. Initialization:** The invariant holds prior to the first loop iteration.  **2. Maintenance:** If the invariant holds at the beginning of an arbitrary loop iteration, then it must also hold at the end of that iteration.  **3. Termination:** The loop always terminates. As the files will end.  **4. Correctness:** Whenever the loop invariant and the loop exit condition both hold, then file\_arra[] must hold and resturn.  **Hash Function to read the word and generates the key automatically by considering length of word and its characters:**  public string hashFunction(string word)  {  string key = "";  for(int i=0;i< word.Length;i++)  {  key += (int)word[i]; //converting to ascii  }  return key;  }  “for loop runs for the length of the word, so it depends on the length of the word”  **1. Initialization:** The first iteration holds the hashtable and puts data from array to it.  **2. Maintenance:** If the loop iteration holds then its must generate the hash table of the ith file into it.  **3. Termination:** The loop always terminates. As the file data terminates and hash table is generated  **4. Correctness:** Whenever the loop invariant and the loop exit condition both hold, hash table of that file is generated.  **Using the functionality to choosing the smaller hashTable between the two hashTables:**  int smallerHash=0;  if(htFileA.Count<= htFileB.Count)  {  smallerHash = 1;  }  else  {  smallerHash = 2;  }  “comparing between two file which file has the smaller hash table values”  List<string> commonWords = new List<string>();  if(smallerHash==1)  {  foreach (string str in htFileA.Values )  {  if( htFileB.ContainsValue(str))  {  commonWords.Add(str);  }  }  }else if(smallerHash == 2)  {  foreach (string str in htFileB.Values)  {  if (htFileA.ContainsValue(str))  {  commonWords.Add(str);  }  }  }  “loop goes for each word in first file hashTable and finds if there contain the same word as in second file hashTable using the key functionality then it adds it in the common words list”  **NOTE:** this above line runs only for one file not for the both cases as we find the one small hashTable and then compare it with the other one  **1. Initialization:** There is an array for the matching content of the files data (which should be string data)  **2. Maintenance:** If the file exists and its not empty then loop must check the matching data. Otherwise move to next file.  **3. Termination:** Nested loop is terminated after examining the all files through it.  **4. Correctness:** Final array of the matching content should be returned and it holds the common data among all the files.  **Using this function for creating multiple hashTables for multiple files and then checking the plagiarism between all hashTables:**  public void check\_Plagiarism()  {  string[] paths = new string[listBox1.Items.Count];  listBox1.Items.CopyTo(paths, 0);  Hashtable[] htableArray = new Hashtable[paths.Length];  //initializing the hastable array  for (int i = 0; i < htableArray.Length; i++)  {  htableArray[i] = new Hashtable();  }  if (paths.Count() > 1)  {  for (int i = 0; i < paths.Length; i++)  {  string Str = File.ReadAllText(paths[i]);  string[] arr = FileWords(Str);  for (int j = 0; j < arr.Length; j++)  {  if (!htableArray[i].ContainsValue(arr[j]) && (!String.IsNullOrEmpty(arr[j])))  {  htableArray[i].Add(hashFunction(arr[j]), arr[j]);  }  }    int smallerHashIndex = 0;  for (int i = 0; i < htableArray.Length; i++)  {  if (htableArray[i].Count <= htableArray[smallerHashIndex].Count)  {  smallerHashIndex = i;  }  }  commonWords = new List<string>();//htableArray[smallerHashIndex].Values.Cast<string>().ToList();  foreach (string word in htableArray[smallerHashIndex].Values)  {  int count = 0;  for (int l = 0; l < htableArray.Length; l++)  {  if (htableArray[l].ContainsValue(word))  {  count++;  }  }  if (count == htableArray.Length)  {  commonWords.Add(word);  }  }  }  }  this is the main code of our program and as we say that if there are N number of files and N number of words in each file then:  **Complexity** = **N**(number of files) x **N**(number of words in each file)  = **N­2**  So, this program is good for the short text files but not for the bigger data files. | |
| ***Interfaces for your project*** | |
| First we have a form of choosing an option that in what type of plagiarism you want to check    After selecting the “Text Files” option we have the new form:    After selecting the “Files/Folders” option we have the new form: | |
| ***Integration*** | |
| Not much because we have used C# language for the interaction of the UI and the algorithm we have used in the backend. In this language the functions are mostly built in and was the beneficial part of our project. | |
| ***Change Requests*** | |
| As compared to startup we didn’t committed any changes but we do have done changes in the beginning as we required as long as we keep building our project. First we tried out algorithm as taken idea from Rabin-Karp but after moving on we did some changes in it that we used the hashtable functionality. Again we didn’t did any commit unless we have completed our project steps one by one. And at the end the final version. | |
| ***Testing*** | |
| Project not tested yet (report without testing) | |
| ***Technology*** |  |
| Programming Language | C# |
| Platform | Desktop Application |