**Analysis of Algorithms**

Spring 2020

**Members Details**

|  |  |
| --- | --- |
| Group ID | CS311S20PID26 |
| Registration Number of Group Members | 2018-CS-114  2018-CS-145 |
| Section | C |

**Project Details**

|  |  |
| --- | --- |
| ***Project*** |  |
| Project Title | File Compression Tool |
| Executive Summary | The main idea of this project is to develop an application that can compress and decompress the text files and also solve the problem of memory management. For this, Huffman coding algorithm is used. First, it takes a text file and calculate the frequencies of the characters and codes of characters by making a tree. It stores the frequencies, characters and codes of all characters in a coded scheme file. After calculating the frequencies, it stores the characters and frequencies in different arrays by sorting them in order. Then, it uses the original file and coded scheme file to create a compressed file. Compressed file contains the codes of all characters. As we know that Huffman algorithm, converts the character into bits, and bits takes less space in memory than characters. We can also decompress a file by using the coded scheme file and compressed file. In this, tree traverses and returns a decompressed file |
| ***Business Case*** |  |
| Outline the business need for the project | By using file compression tool a we can free a large amount of space. And, also compressed files transfer safely. |
| End user of the product | * Educational institutions * Industries   basically it can be be used at any place where computers are used. |
| Motivation for Project | As we know that the transfer of large files is a problem because sometimes they get corrupted and also face the storage issue .So, we want to make a software through which we large files consumes less space in memory and can also transfer safely. |
| Description of the project objective(s) | The objective of this project is to compress and decompress the files and to solve the problem of memory storage. |
| State the level of impact expected should the project proceed and implications of not proceeding | None. |
| Functional Requirements | * Compression * Decompression |
| ***Benefits*** |  |
| What benefits are expected/ anticipated? | 1. File size is reduced   When a file is compressed, it convert it bits or bytes. And we know that bit takes less space in memory.   1. Transfer Safely   When the files are large, they can often get corrupt when they are transfer to other devices. But when they are compressed they can safely transfer. |
| ***Implementation Details*** |  |
| Link to Github Repository | <https://github.com/Ibrar-Hussain/CS311S20PID26> |
| Total Number of commits in repository before 5th August 2020 | 11 |
| Exact contribution of each member | 2018CS145  Performs all the tasks   * Document of algorithm and interface * Pseudocode of compression and tree * Correctness and complexity * GUI alone.   2018CS114  As 2018CS114 changed the group and joined that group at the time of implementation. So, done the implementation with the help of 2018CS145. And also write the pseudocode of Decode function. |
| ***Commits in github repository by each member*** | |
| |  |  | | --- | --- | | **Member Registration No.** | **Total Commits** | | 2018CS145 | 6 | | 2018CS114 | 5 | | |
| **Details of commits** | |
| |  |  |  |  | | --- | --- | --- | --- | | **Sr. No.** | **Details of commit** | **Date** | **Member Reg No.** | | **1** | Commit a document which includes the decision of algorithm and also the drawing of interface | 27 June | 2018CS145 | | **2** | Commit Pseudocode | 5 July | 2018CS145 | | **3** | Commit pseudocode correctness | 8 July | 2018CS145 | | **4** | Commit complexity of algorithm | 8 July | 2018CS145 | | **5** | Commit Complete GUI | 19July | 2018CS145 | | **6** | Commit only code of Huffman Tree and compression | 13 July | 2018CS114 | | **7** | Code without decompression | 20 July | 2018CS114 | | 8 | Commit complete code with errors | 26 July | 2018CS114 | | 9 | Complete implementation | 3 August | 2018CS114 | | 10 | Final Version o fproject | 5 August | 2018CS145 | | 11 | Decode pseudocode |  | 2018CS114 | | |
| Have you used built in algorithms or you have implemented yourself? | No, we didn't use any built in algorithm. |
| Formats of input | A text file is given as input. |
| Validations | Only Text files can be given as input. |
| Format of output | Output is in the form of compressed file. |
| Deployment | Yes , we upload project on google drive then public it.  Anyone can view this by using the following link  https://drive.google.com/file/d/1XihXkKs77\_4WDfUorx4bXOZ6xRHLlDXl/view?usp=sharing |
| ***Details of algorithms*** | |
| For this project we use Huffman Coding Algorithm.  Huffman coding is technique to convert the characters into bits by using the frequencies of characters. In this algorithm, a tree is created which is known as Huffman tree. Tree contains nodes on which frequencies of characters are given. Tree is created by adding the frequency of characters and their sum becomes parent. This process continues until we get a complete tree. To get codes from tree we set a condition if go to the right side of the tree it will assign 1 value and if it go to the left side it will assign 0 value. . We check for codes until we get leaf node. By doing this we will get codes of all characters. But one thing is we have to sort the frequencies of characters and store them in an array for use.  **Time Complexity:**  AB[]={2,5,8,9} //frequency array  RA[]={A,B,C,D} //characters array  //Let's suppose RA is sorted in increasing order of the frequency  //For Tree  parent,left,right,char=NULL  for (i=0;i<length(AB);i++) //let n be length(AB). O(n+1)  if parent==NULL O(1) ||  parent=AB[0]+AB[1] O(n) ||  parent->left=AB[0] O(n) ||  parent->right=AB[1] O(n) ||  char[0]=RA[0] O(n) ||  char[1]=RA[1] ||  else: || O(n)  f1=AB.pop(i) O(n) ||  f2=AB.pop(i+1) O(n) ||  AB.push(f1+f2) O(n) ||  temp=AB[i]+AB[i+1] O(n) ||  if temp>parent && parent<AB[i+1] O(n) ||  temp->left=parent O(n) ||  temp->right=AB[i+1] O(n) ||  parent=temp O(n) ||  char=RA[i+2] O(n) ||  else: ||  temp->right=parent O(n) ||  temp->left=AB[i+1] O(n) ||  parent=temp O(n) ||  char=RA[i+2] O(n) ||  Total time complexity=O(n+1)+O(n)  =O(n)+O(1)+O(n)  =2'O(n)+O(1)  = O(n) answer  ------------------------------------------------------------------------  //Now to code the character  code=""  for(i=0;i<length(RA);i++) //let n be the length(RA) O(n+1)  //while(n1!= j in freqofRA && n2!=j in freqofRA)  temp=parent |  n1=temp->left |  n2=temp->right |  if(n1->left&&right==NULL) |  code+="0" |  elseif(n2->left&&right==NULL) |  code+="1" |  if(n2>n1&&n2->left&&right!=NULL) | O(n)  ` parent=n2 |  ` code-- |  code+="1" |  else |  parent=n1 |  code-- |  code+="0" |  if(n1->left&&right==NULL and n2->left&&right==NULL)  if(n1==freqofRA[0]) |  code+="0" |  elseif n2==freqofRA[1] |  code+="1" |  Total time complexity=O(n+1)+O(n)  =O(n)+O(1)+O(n)  =2'O(n)+O(1)  = O(n) answer  **Correctness:**  Statement:Suppose we have an optimal solution for n=2.There should be the optimal solution on values grater than 2 which extend the depth of the tree.where n is the number of characters or symbols in array  Inductive hypothesis:  In this we assume that the the algrithm solves smaller problems optimally and also for the bigger ones it is optimal.i.e there also exist optimal solution after n=2 e.g for n=4 etc.  Base case:  When n=2 ,this algorithm gives optimal tree. i.e 1bit per symbol or character.  Inductive step:  let's assume that the input size is greater than 2 n>2.  Let T'=T with 2 symbols a and b which are replaced by c which is equal to a+b. a,b are siblings. so the new tree is T'.  for T and T'(T is the tree with base case and T' is tree correspond to the leaves a,b etc  Let f.a anf f.b be the frequency of the characters in array.  So, f.a[depth of a in T]  f.b[depth of b in T] //a and b are one step below in T'  f.ab[depth of ab in T']=f.a+f.b  let d be the depth.  L(T)-L(T')  =f.a(d+1)+f.b(d+1)-(f.a+f.b).d  =f.a.d+f.a+f.b.d+f.b-f.a.d-f.b.d  =f.a+f.b  conclusion:  so at the end of the tree the leave nodes are present with the frequencies of the character and gives the optimal solution as in our base case.  At the end the result is optimal for n greater than 2.  **Pseudocode:**  AB[]={2,5,8,9} //frequency array  RA[]={A,B,C,D} //characters array  //Let's suppose RA is sorted in increasing order of the frequency  //For Tree  parent,left,right,char=NULL  for (i=0;i<length(AB);i++)  if parent==NULL  parent=AB[0]+AB[1]  parent->left=AB[0]  parent->right=AB[1]  char[0]=RA[0]  char[1]=RA[1]  else:  f1=AB.pop(i)  f2=AB.pop(i+1)  AB.push(f1+f2)  temp=AB[i]+AB[i+1]  if temp>parent && parent<AB[i+1]  temp->left=parent  temp->right=AB[i+1]  parent=temp  char=RA[i+2]  else:  temp->right=parent  temp->left=AB[i+1]  parent=temp  char=RA[i+2]  ------------------------------------------------------------------------  //Now to code the character  code=""  for(i=0;i<length(RA);i++)  while(n1!= j in freqofRA && n2!=j in freqofRA)  temp=parent  n1=temp->left  n2=temp->right  if(n1->left&&right==NULL)  code+="0"  elseif(n2->left&&right==NULL)  code+="1"  if(n2>n1&&n2->left&&right!=NULL)  ` parent=n2  ` code--  code+="1"  else  parent=n1  code--  code+="0"  if(n1->left&&right==NULL and n2->left&&right==NULL)  if(n1==freqofRA[0])  code+="0"  elseif n2==freqofRA[1]  code+="1"  **Decode:**  root Node= \*tptr  character[i] = node.character  for(i=0; i<lengthOf(AB); i++)  while(tptr!=leaf node)  if(code==1)  tptr = right  else  tptr = left  return node.character | |
| ***Interfaces for your project*** | |
| **Continue:** Click on this button, to start the working.  **Close Button:** Click this button to close the window.  **Maximize Button:** Click this button to maximize the window.  **Minimize Button:** Click this button to minimize the window.    **Compression File:** click on this button to start the compression process.  **DeCompression File:** click on this button to start the decompression process.  **Execute:** click this button to close the application.    **Text Box:** enter the path of file which you want to compress  **Browse Button:** (…..)Click this to browse files**.**  **Compress:** Click this button to compress the given file.  **Back:** Click this button to go to the previous form    **Text Box1:**  Enter the path of the compressed file  **Text Box2:** Enter the path of coded scheme file.  **Browse:** (…..)Click this to browse files**.**  **Decompress:** Click this button to decompress the given file.  **Back:** Click this button to go to the previous form | |
| ***Integration*** | |
| **Difficulties:**  There are some minor difficulties in integration of UI and algorithm.   1. When we enter a file other than text file it shows Exception Error.   **Strategy:** We apply restriction that only text files can be entered to compress.   1. When we enter a file which is present or not it generates a message that file compressed successfully.   **Strategy:** We use condition to check whether file is present or not.   1. Error occured while entering coded scheme file and compressed file.   **Strategy:** Check the decode file and also use exception handling try-catch | |
| ***Change Requests*** | |
| Project was changed form Activity Scheduling to File compression Tool | |
| ***Testing*** | |
| *In this section, you are required to mention the issues report and solution proposed.* | |
| ***Technology*** |  |
| Programming Language | *C#* |
| Platform | Desktop Application |