|  |  |
| --- | --- |
| download.jpg  Data Structure and Algorithms  PROJECT | Abstract  SYED MUJADIL AHMAD HAZQEEL 01-134212-177 |

Code:

#pragma once

#include <iostream>

#include <fstream>

using namespace std;

//Defining Node structure for the priority queue

struct node

{

char data; //saving character

int Freq; //frequency of that character

node\* next; //Pointer to next element in the linked queue

node\* prev; //Pointer to previous element in the linked queue

};

//Defining priority queue specificly designed for this assignment

class Pque

{

private:

int size; //Size of the queue

node\* Header; //External Pointer to the first element

public:

Pque() //Default Constructor

{

Header = NULL;

size = 0;

}

bool isEmpty()

{

if (Header == NULL) //If Header does not exist, there will be no element in the queue

return true;

else

return false;

}

//Returns a New Node pointer

node\* GetNode(char d, node\* p = NULL, node\* n = NULL)

{ //Assigning values

node\* temp = new node;

temp->data = d;

temp->Freq = 1;

temp->prev = p;

temp->next = n;

return temp;

}

void push(char d) //Function to add character to the queue

{

if (Header == NULL) //First ELement Case

{

Header = GetNode(d, NULL, NULL);

size = 1;

}

else

{

node\* p, \* q, \* temp;

for (p = Header, q = Header->next; q != NULL; p = q, q = q->next)

{ //In case element already exists

if (p->data == d)

{

p->Freq++; //Frequency will Increase

return;

}

}

//New Element Entry Case

for (p = Header, q = Header->next; q != NULL; p = q, q = q->next)

;

temp = GetNode(d, p, NULL);

p->next = temp;

size++;

}

}

int GetSize()

{

return size;

}

void Traverse() //Printing Function

{ //Mostly used to check the working of the queue

node\* p, \* q, \* temp;

for (p = Header, q = Header->next; q != NULL; p = q, q = q->next)

{

cout << "character = " << p->data << " , frequency = " << p->Freq << endl;

}

}

//This Function Sorts the linked Queue in the desired

//order comparing the frequencies of the characters

void sort()

{

node\* p = Header;

node\* q = NULL;

if (p == NULL) //In case of Empty Queue

return;

else { //Bubble Sort

while (p != NULL) {

q = p->next;

while (q != NULL) {

if (p->Freq > q->Freq) //Comparing Statement

swap(p, q); //Swap Function

q = q->next; //Inner Loop Increment

}

p = p->next; //Outer Loop Increment

}

}

}

//Swap function that the sorting function uses

void swap(node\* a, node\* b) {

//The values are swapped, the pointers left as they were

char d; //Temporary variable

int da; //Temporary variable

d = a->data;

da = a->Freq;

a->data = b->data;

a->Freq = b->Freq;

b->data = d;

b->Freq = da;

}

//Function to POP and Delete the first Node from the Linked Queue

void pop(char& d, int& f)

{

if (this->isEmpty())

return;

this->size--; //Decrement in the total size of the queue

node\* p, \* q, \* temp;

int min = Header->Freq;

temp = Header;

d = Header->data; //Saving the values

f = Header->Freq; //Saving the values

p = Header;

Header = Header->next;

delete p;

}

//Function that reads the files and stores the characters in this queue

void ReadFile(string filename)

{

string str;

ifstream file(filename, ios::in); //File Already Initialized

while (!file.eof())

{

file >> str; //Reading the text

for (int i = 0; i < str.length(); i++)

{

this->push(str[i]); //Saving the Characters

}

this->push(' '); //Entering Spaces as the this function does not save spaces

}

}

};

#include <vector> //Header Files

#include <queue>

#include "Pque.cpp"

#include <cstring>

#include <string>

using namespace std;

//Defining Huffman Class

class Huffman

{

//Defining Leaf Node of a Huffman Tree

struct Leaf

{

char data;

size\_t freq;

Leaf\* left; //Pointer to Left Child

Leaf\* right; //Pointer to right Child

Leaf(char data, size\_t freq) //Default Constructor

: data(data), freq(freq), left(NULL), right(NULL)

{}

~Leaf() //Destructor

{

delete left;

delete right;

}

};

//Function that tells the predefined priority Queue the basis

//upon which the queue will be prioritized

struct compare

{

bool operator()(Leaf\* l, Leaf\* r)

{

return (l->freq > r->freq);

}

};

Leaf\* top; //Pointer to the Top Node or Root Node

//This Function is kept private as it speficly designed to calculate all the Huffman Codes

// and should be called be only once

//Printing Recursive Function

void print\_Code(Leaf\* root, string str, string\* ar, char\* cr, int& n)

{

if (root == NULL) //In case the Queue is Empty

return;

if (root->data == '\*') //In case the root is not a leaf

//Leaf nodes contains specific characters Internal Nodes

//Contains the sum of the Leafs or Childern Nodes

{

//Adding 0 to the string/Hufman Code if we move to the left child/Node

print\_Code(root->left, str + "0", ar, cr, n);

//Adding 1 to the string/Hufman Code if we move to the right child/Node

print\_Code(root->right, str + "1", ar, cr, n);

}

if (root->data != '\*') //In case the root is a Leaf

{

cr[n] = root->data; //Saving characters in character Array

ar[n++] = str; //Saving Huffman Codes in a string

//Adding 0 to the string/Hufman Code if we move to the left child/Node

print\_Code(root->left, str + "0", ar, cr, n);

//Adding 1 to the string/Hufman Code if we move to the right child/Node

print\_Code(root->right, str + "1", ar, cr, n);

}

}

public:

//Default Constructor

Huffman()

{

top = NULL;

}

//Destructor

~Huffman()

{

delete top;

}

//Huffman Tree and Code Creating Function

void Huffman\_tree(Pque p, string\* str, char\* cr)

{

Leaf\* left;

Leaf\* right;

priority\_queue<Leaf\*, vector<Leaf\*>, compare> Huffman\_Tree; //Creating Priority Queue

//with Leaf Struct as it elements, Vector as its container,

//The Piority is based on the Comapre function

//Initalizing The Predefined Priority Queue

while (!p.isEmpty()) //Empting the User Defined Priority Queue into the Predined one

{

char c; //Temporary Variables

int f; //Temporary Variables

p.pop(c, f); //Poping Nodes

Huffman\_Tree.push(new Leaf(c, f)); //Pushing Values Into the Huffman Tree

}

while (Huffman\_Tree.size() != 1) //Making the Huffman Tree

{

left = Huffman\_Tree.top(); //Assigning Values

Huffman\_Tree.pop(); //poping Leafs from the queue

right = Huffman\_Tree.top(); //Assigning Values

Huffman\_Tree.pop(); //poping Leafs from the queue

//Creating Internal Node with \* as identity and sum of its child leafs

top = new Leaf('\*', left->freq + right->freq);

top->left = left; //Linking the Parent Node to Child Node

top->right = right; //Linking the Parent Node to Child Node

Huffman\_Tree.push(top); //Pushing the Internal Node Back into the Queue

}

int n = 0; //Temporary variable to traverse the arrays in the Priting Function

print\_Code(Huffman\_Tree.top(), "", str, cr, n); //Printing Function

}

};

#include "Huff.cpp"

#pragma warning(disable : 4996) //Disabling a waring concerning strcat function

void String\_to\_char(string\* ar, char\*\* arr, int size); //Prototypes

void writeOnFile(char\* cr, char\*\* arr, int size);

void CompressionRate();

int main() //Driver Function

{

Pque p; //User Defined Priority Queue

Huffman H; //Huffman Object to print Codes

string\* ar; //String array to save the codes

char\*\* arr; //String array using Char to write the codes into the final file

char\* cr; //Character Array to Store characters

p.ReadFile("TestFile.txt"); //Reading the text file and loading the User Defined Priority Queue

p.Traverse(); //Printing The Queue

cout << endl << endl;

cr = new char[p.GetSize()]; //Initialing the Character Array for Elements storage

ar = new string[p.GetSize()]; //Initialing the String Array for Huffman Codes storage

H.Huffman\_tree(p, ar, cr); //Implementing the Huffman tree as well as printing the codes

arr = new char\* [p.GetSize()]; //Initalizing the String array using char

for (int row = 0; row < p.GetSize(); ++row)

arr[row] = new char(ar[row].length() + 1); //Initiailing each string according to its respective Code

cout << endl << endl;

String\_to\_char(ar, arr, p.GetSize()); //Coping the String Array into character string array

for (int i = 0; i < p.GetSize(); i++)

cout << cr[i] << " : \t" << arr[i] << endl; //Printing the Codes

writeOnFile(cr, arr, p.GetSize()); //Encoding the New text File using Huffman Codes

CompressionRate(); //Calculating and Printing Compressiong Rate

cout << endl << endl;

system("pause"); //Pausing the Screen

return 0; //Successfull Termination of the Main/Driver Function

}

//Funtion that copies the string array into character string array for writing

void String\_to\_char(string\* ar, char\*\* arr, int size)

{

for (int i = 0; i < size; i++)

{

strcpy(arr[i], ar[i].c\_str()); //This function does not work on Visual studio for some reason

//Hence the warning disable in the top of this page

}

}

//Funtion that writes the codes in the new File

void writeOnFile(char\* cr, char\*\* arr, int size)

{

string temp; //Temporary Variable to read the old text

ifstream OldFile("TestFile.txt", ios::in); //Opening the OldFile in Read Mode

ofstream NewFile("CompressedFile.txt", ios::out); //Opening the NewFile in Witing Mode

while (!OldFile.eof()) //Reading the Old File

{

OldFile >> temp; //Saving Words in temporary variable

temp += " "; //Adding Spaces

for (int i = 0; i < temp.length(); i++) //Replacing the characters with their Huffman Codes

{

int j = 0; //Counter

while (j < size)

{

if (temp[i] == cr[j]) //Replacing Text to Code

{

NewFile << arr[j]; //Writing on the file

break;

}

j++;

}

}

}

OldFile.close(); //Closing the Files

NewFile.close();

}

//Function to Calculate and Print the Compressiong Rate

void CompressionRate()

{

int OldSize; //Temporary Variables

int NewSize;

string t = "";

string c = "";

ifstream OldFile("TestFile.txt", ios::in); //Opening Files in Reading Modes

ifstream NewFile("CompressedFile.txt", ios::in);

while (!OldFile.eof()) //Reading Old File

{

OldFile >> t;

c += t + " ";

}

OldSize = c.size() \* 8; //Each Character/Symbol is stored as a 8 bit code

//So we Multiply the total number of characters with 8 to find the total size

//of the text

c = ""; //Reseting the Temporary Variable

while (!NewFile.eof()) //Reading New File

{

NewFile >> t;

c += t;

}

NewSize = c.size(); //This text is simply the code given to a character

//so each integer represents a bit

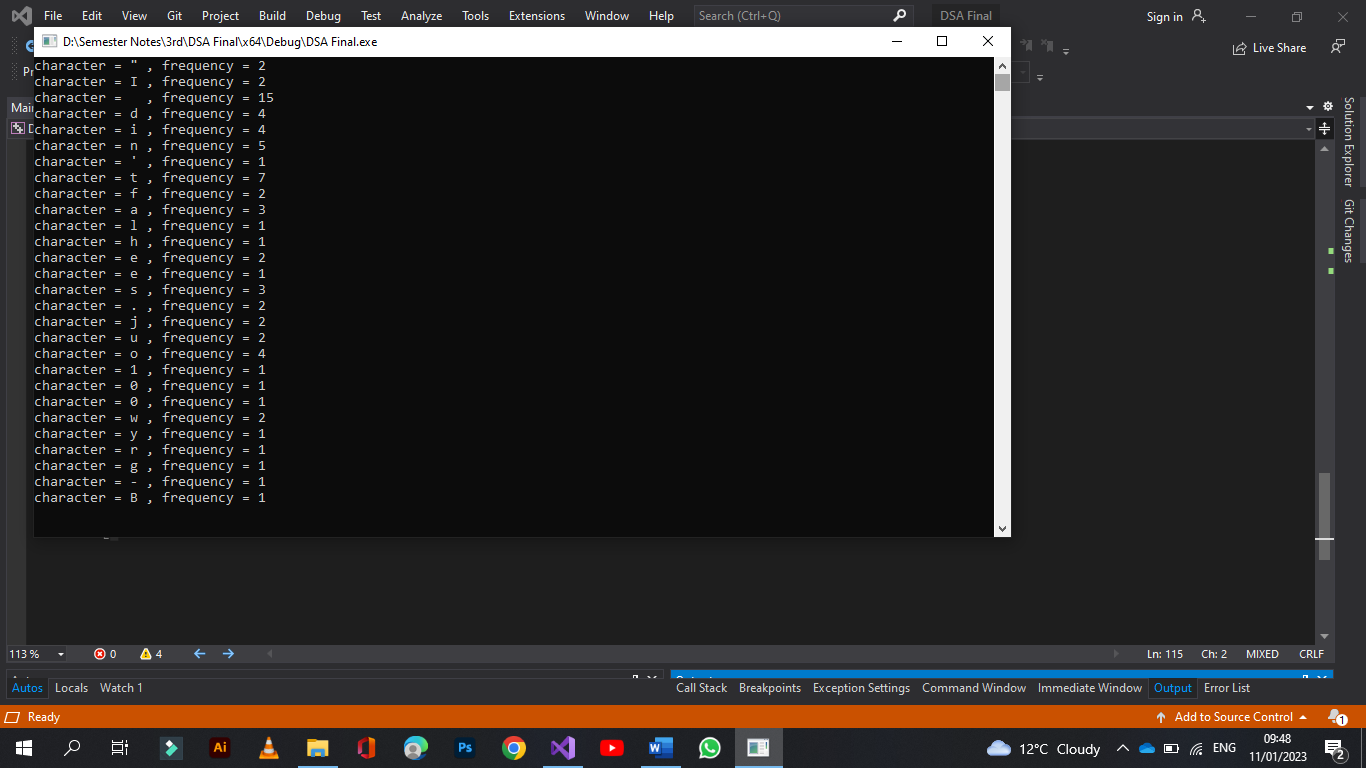
cout << "File Size Before Compression = " << OldSize << " Bits " << endl; //Print Statements

cout << "File Size After Compression = " << NewSize << " Bits" << endl;

cout << "Compression Rate = " << double(OldSize) / double(NewSize) << endl; //Calculating Compression Rate

}

OUTPUT:



A screenshot of a computer

Description automatically generated

TEXT FILE:

Graphical user interface, text, application

Description automatically generated

COMPRESSED FILE:

Graphical user interface, text, application, Word

Description automatically generated