**LAB 9**

/\* Sushant Bansal

1410110454

Huffman Codes

March 1

\*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <malloc.h>

#define MAX 100

#define MAX\_ALPHABETS 4

typedef struct node{

int value;

char alphabet;

struct node \*left,\*right;

}Node;

void buildHTree(Node \*\*tree);

void revC(int codeTable[],int codeTable2[]);

int findSmaller (Node \*array[], int differentFrom);

void generateCodes(int codeTable[], Node \*tree, int Code);

int freqOfAlphabets[MAX\_ALPHABETS];

int main(){

Node \*root = NULL;

int revCode[MAX\_ALPHABETS],actCode[MAX\_ALPHABETS];

char randomArrayofChar[MAX];

for(int i=0;i<MAX\_ALPHABETS;i++)

freqOfAlphabets[i] = 0;

srand(time(NULL));

for(int i = 0;i<100;i++)

randomArrayofChar[i] = (char)(rand()%MAX\_ALPHABETS+65);

for(int i=0;i<MAX\_ALPHABETS;i++)

for(int j=0;j<MAX;j++)

if(randomArrayofChar[j] == (char)(i+65))

freqOfAlphabets[i]++;

for(int i=0;i<MAX\_ALPHABETS;i++){

printf("%c : %d\n", (char)(i+65), freqOfAlphabets[i]);

}

buildHTree(&root);

generateCodes(revCode,root,0);

revC(revCode,actCode);

int totalActualBits = 8 \* sizeof(randomArrayofChar);

int totalHuffmanBits = 0;

for(int i=0;i<MAX\_ALPHABETS;i++){

printf("%d\n",actCode[i]/10);

totalHuffmanBits += freqOfAlphabets[i]\*(actCode[i]/10);

}

printf("\*\*\*\* Total Actual Bits: %d \*\*\*\* Total Huffman Bits: %d \*\*\*\* \n",totalActualBits,totalHuffmanBits);

}

void buildHTree(Node \*\*tree){

Node \*temp;

Node \*array[MAX\_ALPHABETS];

int i, subTrees = MAX\_ALPHABETS;

int smallOne,smallTwo;

for (int i=0;i<MAX\_ALPHABETS;i++){

array[i] = malloc(sizeof(Node));

array[i]->value = freqOfAlphabets[i];

array[i]->alphabet = i;

array[i]->left = NULL;

array[i]->right = NULL;

}

while (subTrees>1){

smallOne=findSmaller(array,-1);

smallTwo=findSmaller(array,smallOne);

temp = array[smallOne];

array[smallOne] = malloc(sizeof(Node));

array[smallOne]->value=temp->value+array[smallTwo]->value;

array[smallOne]->alphabet=100;

array[smallOne]->left=array[smallTwo];

array[smallOne]->right=temp;

array[smallTwo]->value=-1;

subTrees--;

}

\*tree = array[smallOne];

return;

}

int findSmaller (Node \*array[], int differentFrom){

int smaller;

int i = 0;

while (array[i]->value==-1)

i++;

smaller=i;

if (i==differentFrom){

i++;

while (array[i]->value==-1)

i++;

smaller=i;

}

for (int i=1;i<MAX\_ALPHABETS;i++){

if (array[i]->value==-1)

continue;

if (i==differentFrom)

continue;

if (array[i]->value<array[smaller]->value)

smaller = i;

}

return smaller;

}

void generateCodes(int codeTable[], Node \*tree, int Code){

if (tree->alphabet<MAX\_ALPHABETS)

codeTable[(int)tree->alphabet] = Code;

else{

generateCodes(codeTable, tree->left, Code\*10+1);

generateCodes(codeTable, tree->right, Code\*10+2);

}

}

void revC(int codeTable[],int codeTable2[]){

int i, n, copy;

for (int i=0;i<MAX\_ALPHABETS;i++){

n = codeTable[i];

copy = 0;

while (n>0){

copy = copy \* 10 + n %10;

n /= 10;

}

codeTable2[i]=copy;

}

}