

Algorithms Assignment

Huffman Code



May 23, 2020

Syed Tafreed Numan

Sc ID 18-15-043

Solution 1:

Implementation Done. Code and Output Table Pasted Below.

Solution 2: Algorithms Used

1(a)Priority Queue

HEAP-EXTRACT-MIN(A)

1. *If A.heapsize<1*
2. *error “heap underflow”*
3. *min=A[1]*
4. *A[1] = A[A.heapsize]*
5. *A.heapsize = A.heapsize-1*
6. *MIN-HEAPIFY(A,1)*
7. *return min*

HEAP-INSERT-KEY(A,key)

1. *A.heapsize = A.heapsize+1;*
2. *A[A.heapszie]=key*
3. *i = A.heapsize*
4. *while i>1 and A[PARENT(i)]>A[i]*
5. *exchange A[i] with A[PARENT(i)]*
6. *i=PARENT(i)*

PARENT(i)

1. *return floor(i/2)*

*1(b)Merging Two nodes*

HUFFMAN-NODES-MERGE(A)

1. *while A.heapsize!=1*
2. *allocate a new node Z*
3. *Z.left=x=HEAP-EXTRACT-MIN(A)*
4. *Z.right=y= HEAP-EXTRACT-MIN(A)*
5. *Z.freq=x.freq+y.freq*
6. *HEAP-INSERT-KEY(A,Z)*
7. *Return HEAP-EXTRACT-MIN(A)*

1 (c) COMIBINING to make Huffman code

HUFFMAN-CODE(A,Data,Freq)

1. *for i=1 upto Data.size*
2. *A[i].data=Data[i]*
3. *A[i].freq=Freq[i]*
4. *BUILD-MIN-HEAP(A)*
5. *root = HUFFMAN-NODES-MERGE(A)*
6. *create array “code”*
7. *top=0*
8. *PRINT-HUFFMAN-CODE(root,code,top)*

1(d) Decoding and Printing the Codes

PRINT-HUFFMAN-CODE(root,code,top)

1. *If root->left*
2. *code[top] = 0*
3. *PRINT-HUFFMAN-CODE(root->left,code,top+1)*
4. *If root->right*
5. *code[top] = 0*
6. *PRINT-HUFFMAN-CODE(root->right,code,top+1)*
7. *If root is a leaf node*
8. *print array “code”*

1(e) STRUCTURE OF NODES

Struct heapnode{

Char data;

Int freq;

Struct heapnode \*left, \*right;

}

Solution 3

CODE:

#include<bits/stdc++.h>

using namespace std;

#define MAX 100

struct heapnode{

char data;

int freq;

struct heapnode \*left, \*right;

};

struct heap{

int size;

int heapsize;

struct heapnode\*\* arr;

};

struct heapnode\* new\_node(char data, unsigned freq){

struct heapnode\* temp = (struct heapnode\*)malloc(sizeof(struct heapnode));

temp->left = temp->right = NULL;

temp->data = data;

temp->freq = freq;

return temp;

}

struct heap\* create(unsigned heapsize){

struct heap\* heap = (struct heap\*)malloc(sizeof(struct heap));

heap->size = 0;

heap->heapsize = heapsize;

heap->arr = (struct heapnode\*\*)malloc(heap->

heapsize \* sizeof(struct heapnode\*));

return heap;

}

void swap\_node(struct heapnode\*\* node1,struct heapnode\*\* node2){

struct heapnode\* temp = \*node1;

\*node1 = \*node2;

\*node2 = temp;

}

void heapify(struct heap\* heap, int index){

int small = index;

int left = 2 \* index + 1;

int right = 2 \* index + 2;

if (left < heap->size && heap->arr[left]->freq < heap->arr[small]->freq)

small = left;

if (right < heap->size && heap->arr[right]->freq < heap->arr[small]->freq)

small = right;

if (small != index) {

swap\_node(&heap->arr[small],&heap->arr[index]);

heapify(heap, small);

}

}

int size\_check(struct heap\* heap){

return (heap->size == 1);

}

struct heapnode\* extractMin(struct heap\* heap){

struct heapnode\* temp = heap->arr[0];

heap->arr[0]

= heap->arr[heap->size - 1];

--heap->size;

heapify(heap, 0);

return temp;

}

void insertheap(struct heap\* heap,struct heapnode\* heapnode){

++heap->size;

int i = heap->size - 1;

while (i && heapnode->freq < heap->arr[(i - 1) / 2]->freq) {

heap->arr[i] = heap->arr[(i - 1) / 2];

i = (i - 1) / 2;

}

heap->arr[i] = heapnode;

}

void buildheap(struct heap\* heap){

int n = heap->size - 1;

int i;

for (i = (n - 1) / 2; i >= 0; --i)

heapify(heap, i);

}

void printArr(int arr[], int n){

int i;

for (i = 0; i < n; ++i)

cout << arr[i];

cout << endl;

}

int leaf\_check(struct heapnode\* root){

return !(root->left) && !(root->right);

}

struct heap\* createAndBuildheap(char data[], int freq[], int size) {

struct heap\* heap = create(size);

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

heap->arr[i] = new\_node(data[i], freq[i]);

heap->size = size;

buildheap(heap);

return heap;

}

struct heapnode\* merge\_nodes\_huffman(char data[], int freq[], int size){

struct heapnode \*left, \*right, \*top;

struct heap\* heap = createAndBuildheap(data, freq, size);

while (!size\_check(heap)){

left = extractMin(heap);

right = extractMin(heap);

top = new\_node('$', left->freq + right->freq);

top->left = left;

top->right = right;

insertheap(heap, top);

}

return extractMin(heap);

}

void codes\_print(struct heapnode\* root, int arr[], int top){

if (root->left) {

arr[top] = 0;

codes\_print(root->left, arr, top + 1);

}

if (root->right) {

arr[top] = 1;

codes\_print(root->right, arr, top + 1);

}

if (leaf\_check(root)) {

cout<< root->data <<": ";

printArr(arr, top);

}

}

void huffman(char data[], int freq[], int size){

struct heapnode\* root = merge\_nodes\_huffman(data, freq, size);

int arr[MAX], top = 0;

codes\_print(root, arr, top);

}

int main() {

char code[] = {'a','b','c','d','e','f','g','h','i','j'};

int freq[] = {1097,178,1357,759,1009,598,951,1417,1533,1101};

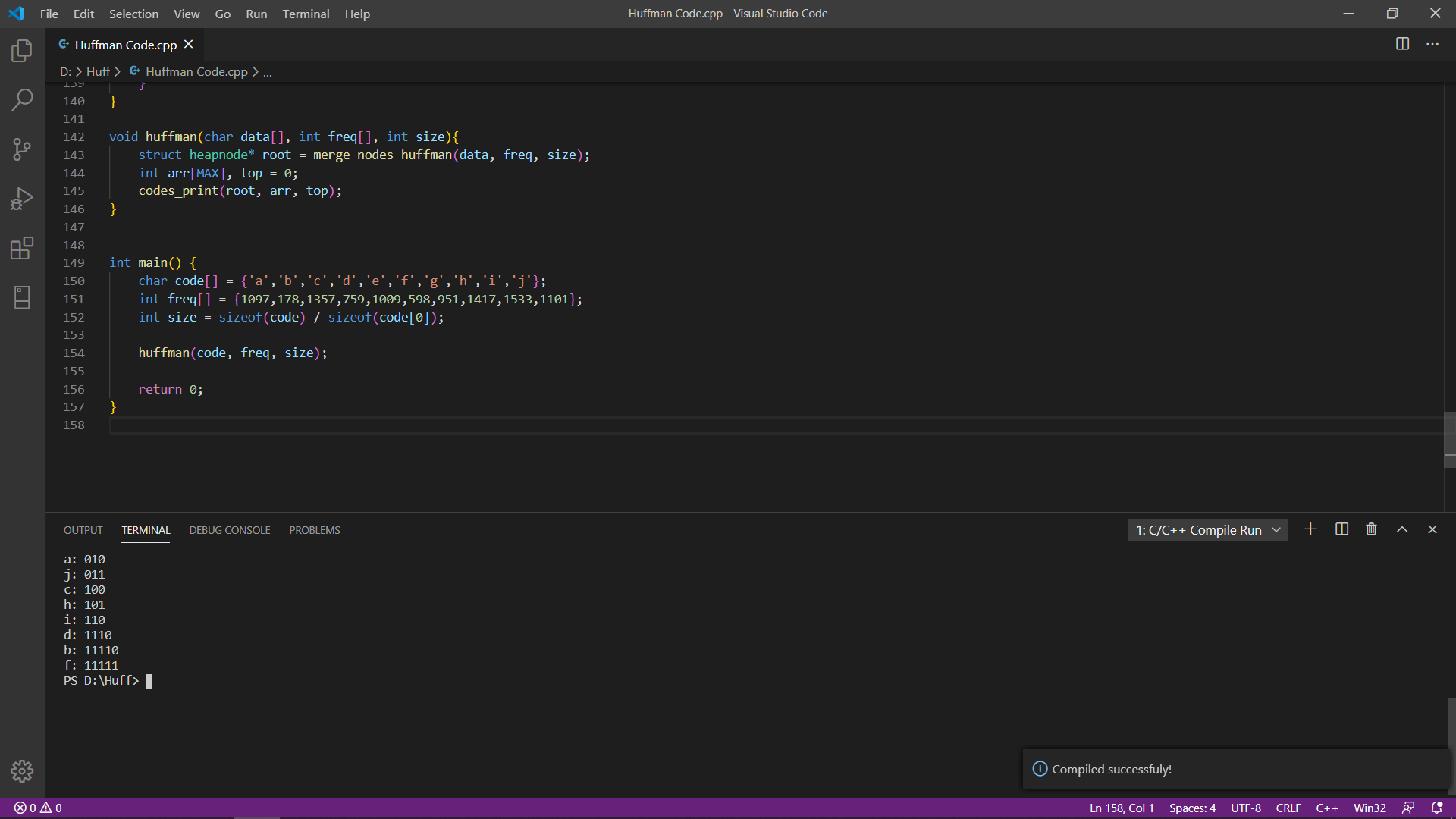
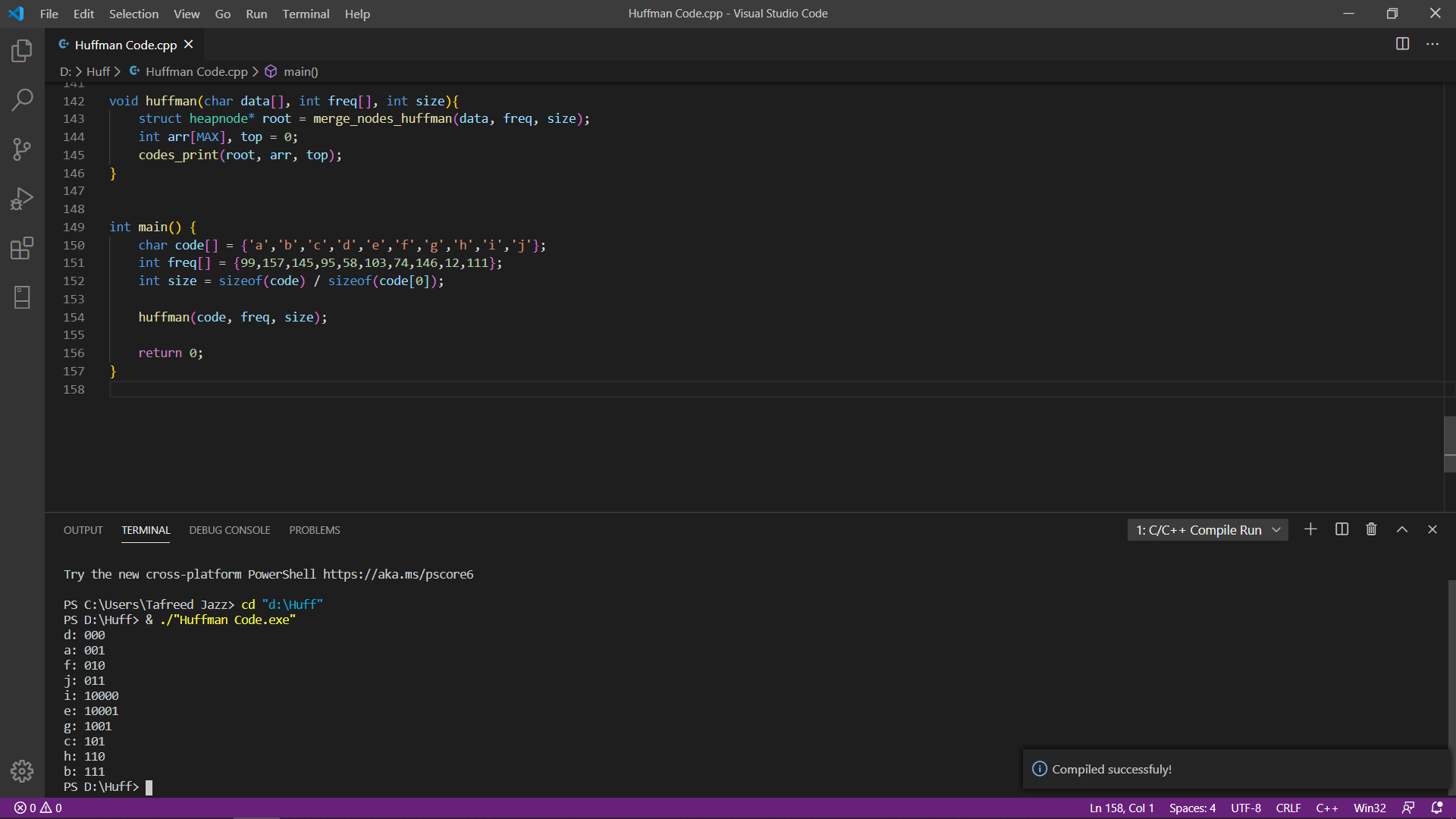
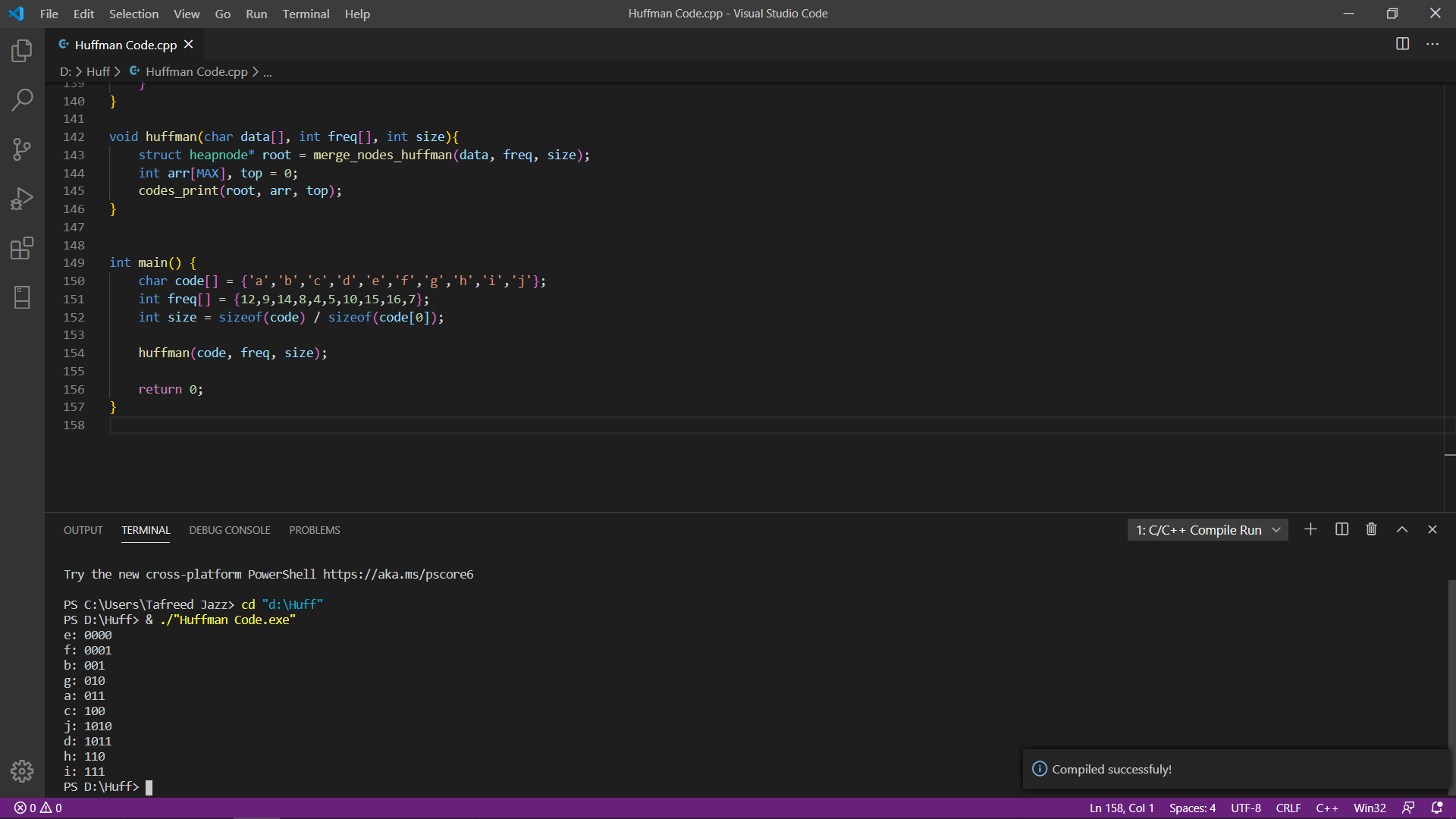
int size = sizeof(code) / sizeof(code[0]);

huffman(code, freq, size);

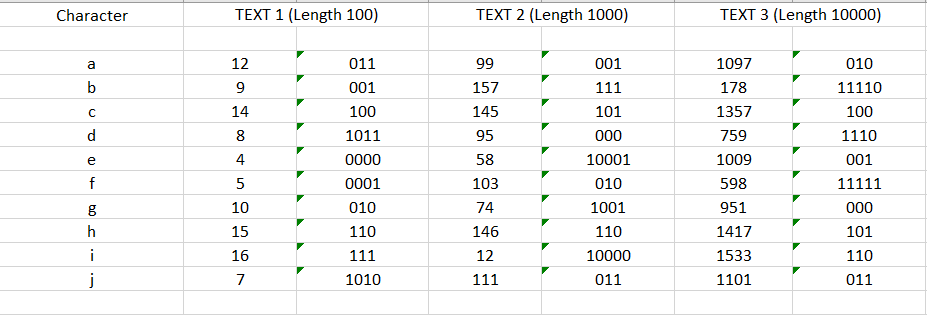
return 0;

}

SNAPSHOT OF OUTPUT(TEXT-1 ,TEXT-2, TEXT-3)

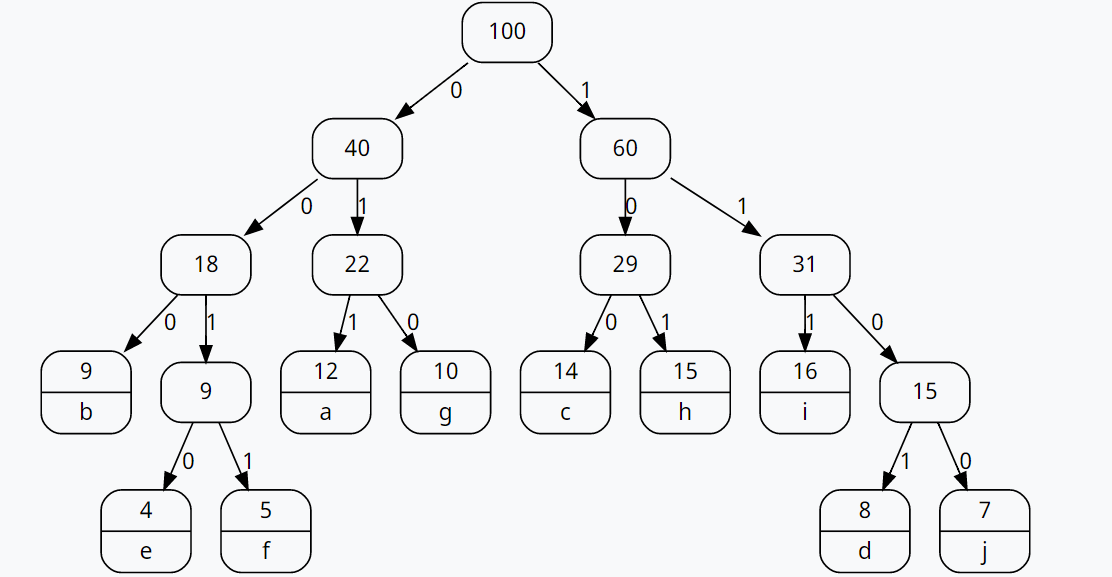


OUTPUT TABLE:

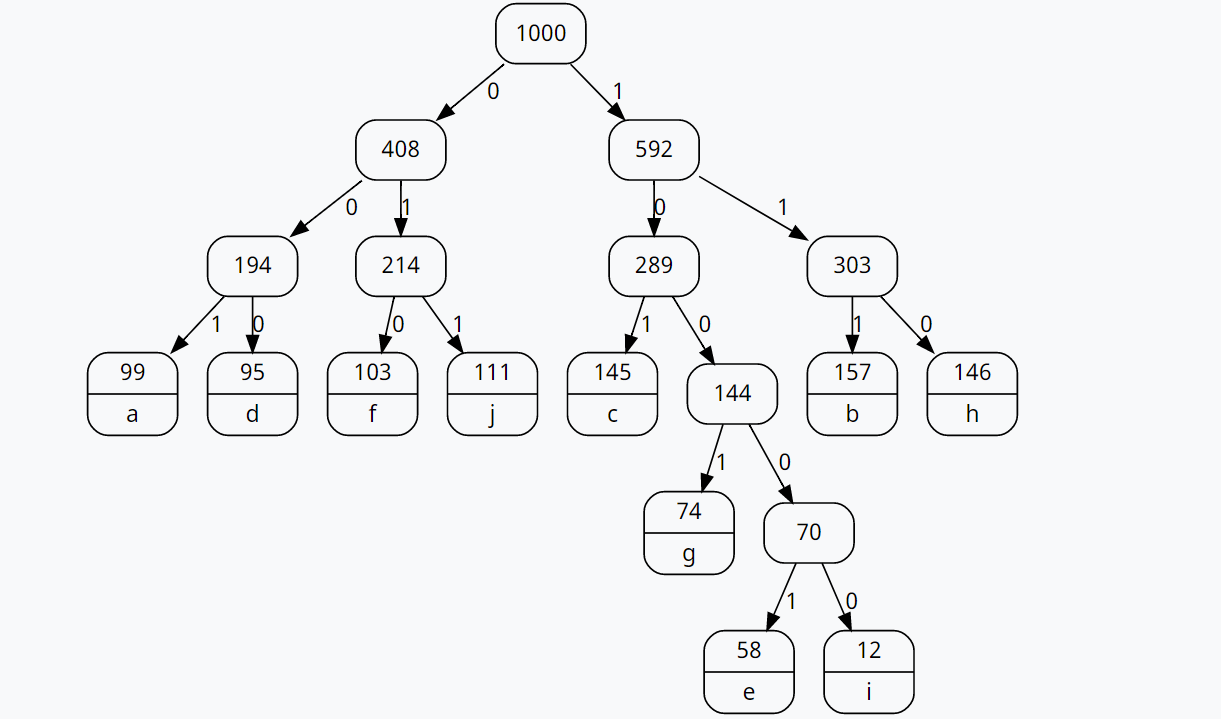


SOLUTION 4: Structure of Huffman Tree

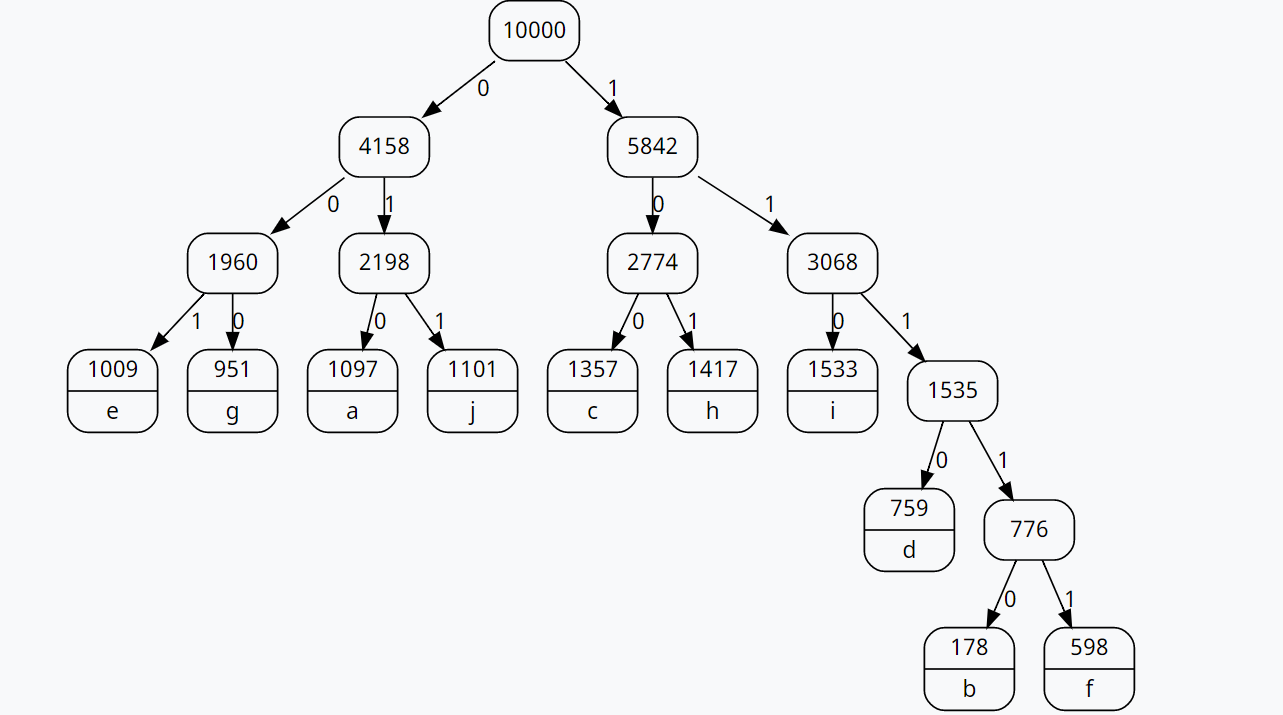
For TEXT-1(Length 100)



For TEXT-2(Length 1000)



For TEXT-2(Length 10000)



----------------------------------------------------------x--------------------------------------------------------------