

**IMAGE COMPRESSION USING HUFFMAN CODING-DAVID HUFFMAN.**

COURSE: DATA STRUCTURES-CE386

GROUP MEMBERS

* [NAME]
* [NAME]
* [NAME]

Table of contents:

[1. Code: 2](#_Toc1995336)

[2. Output format: 9](#_Toc1672130617)

[3. Appendix B: Images for testing 10](#_Toc1506280215)

[I4 10](#_Toc1220163102)

[I5 12](#_Toc2030979555)

[I6 15](#_Toc1680190904)

[Comparison-Appendix A: 20](#_Toc1100462058)

[I1 21](#_Toc352404258)

[I2 21](#_Toc820665036)

[I3 22](#_Toc2109740487)

[References: 24](#_Toc1928744306)

# Code:

#include <stdio.h>

#include <stdlib.h>

typedef struct TreeNode {

int frequency;

int pixelValue; // Pixel intensity value for leaf nodes, -1 for internal nodes

struct TreeNode\* left;

struct TreeNode\* right;

} TreeNode;

TreeNode\* createNode(int frequency, int pixelValue) {

TreeNode\* newNode = (TreeNode\*)malloc(sizeof(TreeNode));

newNode->frequency = frequency;

newNode->pixelValue = pixelValue;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

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

TreeNode\* temp = \*a;

\*a = \*b;

\*b = temp;

}

void sortNodes(TreeNode\*\* nodes, int size) {

for (int i = 0; i < size - 1; i++) {

for (int j = i + 1; j < size; j++) {

if (nodes[i]->frequency > nodes[j]->frequency) {

swap(&nodes[i], &nodes[j]);

}

}

}

}

TreeNode\* buildHuffmanTree(int\* frequencies, int\* pixelValues, int size) {

TreeNode\* nodes[size];

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

nodes[i] = createNode(frequencies[i], pixelValues[i]);

}

while (size > 1) {

sortNodes(nodes, size);

TreeNode\* left = nodes[0];

TreeNode\* right = nodes[1];

TreeNode\* z = createNode(left->frequency + right->frequency, -1);

z->left = left;

z->right = right;

nodes[1] = z;

for (int i = 0; i < size - 1; i++) {

nodes[i] = nodes[i + 1];

}

size--;

}

return nodes[0];

}

void printCodes(TreeNode\* root, int arr[], int top, int\* values, int\* freq, int\* codewords, int\* sizes, int\* idx) {

if (root->left) {

arr[top] = 1;

printCodes(root->left, arr, top + 1, values, freq, codewords, sizes, idx);

}

if (root->right) {

arr[top] = 0;

printCodes(root->right, arr, top + 1, values, freq, codewords, sizes, idx);

}

if (!(root->left) && !(root->right)) {

printf("Pixel Intensity Value: %d, Code: ", root->pixelValue);

for (int i = 0; i < top; ++i) {

printf("%d", arr[i]);

}

printf("\n");

values[\*idx] = root->pixelValue;

freq[\*idx] = root->frequency;

sizes[\*idx] = top;

for (int i = 0; i < top; ++i) {

codewords[\*idx] = (codewords[\*idx] << 1) | arr[i];

}

(\*idx)++;

}

}

void printHuffmanCodes(TreeNode\* root, int\* image, int imageSize) {

int arr[100];

int top = 0;

int values[100];

int freq[100];

int codewords[100] = {0};

int sizes[100];

int idx = 0;

printf("\nHuffman Codes:\n");

printCodes(root, arr, top, values, freq, codewords, sizes, &idx);

for (int i = 0; i < idx - 1; ++i) {

for (int j = i + 1; j < idx; ++j) {

if (freq[i] < freq[j]) {

int tempVal = values[i];

int tempFreq = freq[i];

int tempCode = codewords[i];

int tempSize = sizes[i];

values[i] = values[j];

freq[i] = freq[j];

codewords[i] = codewords[j];

sizes[i] = sizes[j];

values[j] = tempVal;

freq[j] = tempFreq;

codewords[j] = tempCode;

sizes[j] = tempSize;

}

}

}

printf("\nHuffman Codes and Sizes:\n");

printf("Pixel Intensity Value\tFrequency\tCode-word\tSize\n");

for (int i = 0; i < idx; ++i) {

printf("%d\t\t\t%d\t\t\t", values[i], freq[i]);

for (int j = sizes[i] - 1; j >= 0; --j) {

printf("%d", (codewords[i] >> j) & 1);

}

printf("\t\t\t%d\*%d\n", freq[i], sizes[i]);

}

int totalSize = 0;

for (int i = 0; i < idx; ++i) {

totalSize += freq[i] \* sizes[i];

}

printf("\nTotal Size: %d\n", totalSize);

printf("\nCompressed Image Bitstream:\n");

for (int i = 0; i < imageSize; i++) {

for (int j = 0; j < idx; j++) {

if (image[i] == values[j]) {

for (int k = sizes[j] - 1; k >= 0; k--) {

printf("%d", (codewords[j] >> k) & 1);

}

break;

}

}

}

printf("\n");

}

void calculateFrequency(int\* image, int\* freq, int\* values, int imageSize, int\* uniqueCount) {

for (int i = 0; i < imageSize; i++) {

int found = 0;

for (int j = 0; j < \*uniqueCount; j++) {

if (values[j] == image[i]) {

freq[j]++;

found = 1;

break;

}

}

if (!found) {

values[\*uniqueCount] = image[i];

freq[\*uniqueCount] = 1;

(\*uniqueCount)++;

}

}

printf("\nTable-1: Pixel Intensity Value and Frequency\n");

printf("Pixel Intensity Value\tFrequency\n");

for (int i = \*uniqueCount - 1; i >= 0; i--) { // Print in reverse order

printf("%d\t\t\t%d\n", values[i], freq[i]);

}

}

void compressImage(int\* image, int imageSize) {

int freq[256] = {0};

int values[256] = {0};

int uniqueCount = 0;

calculateFrequency(image, freq, values, imageSize, &uniqueCount);

TreeNode\* root = buildHuffmanTree(freq, values, uniqueCount);

printf("\nHuffman Tree built successfully!\n");

printHuffmanCodes(root, image, imageSize);

}

int main() {

int choice;

while (1) {

printf("\nMenu:\n");

printf("1. Compress Image\n");

printf("2. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

if (choice == 1) {

int image[100], imageSize;

printf("Enter the number of pixel intensity values in the image: ");

scanf("%d", &imageSize);

printf("Enter the pixel intensity values:\n");

for (int i = 0; i < imageSize; i++) {

scanf("%d", &image[i]);

}

int imageSize = sizeof(image) / sizeof(image[0]);

compressImage(image, imageSize);

} else if (choice == 2) {

break;

} else {

printf("Invalid choice. Please try again.\n");

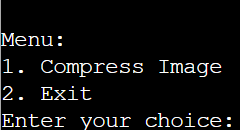
}

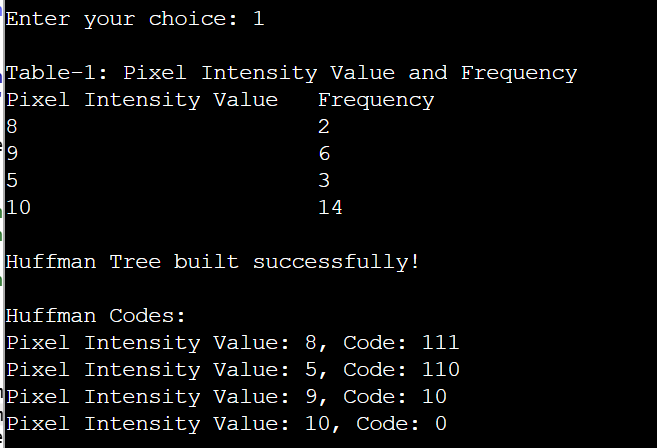
}

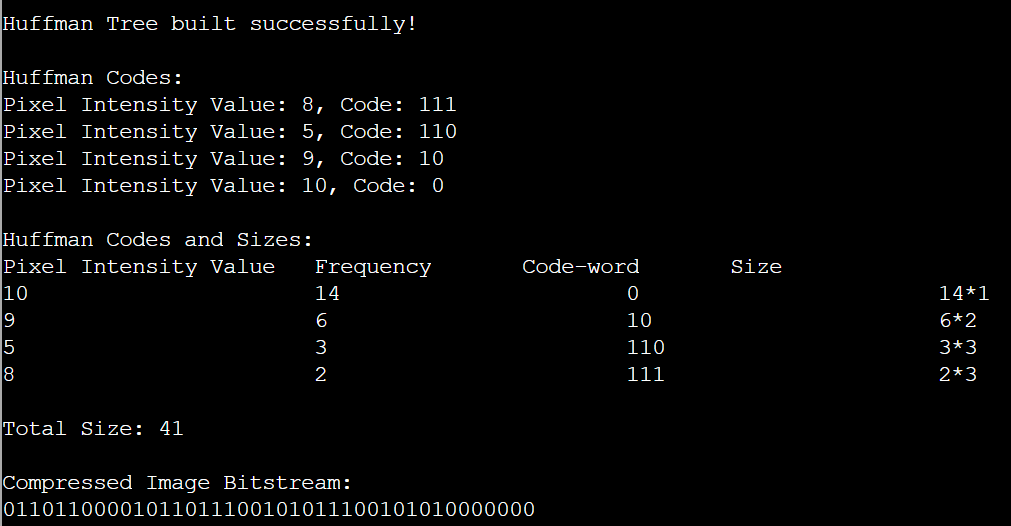
return 0;

}

# Output format:

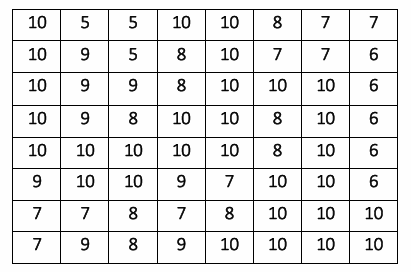






# Appendix B: Images for testing

## I4



Menu:

1. Compress Image

2. Exit

Enter your choice: 1

Table-1: Pixel Intensity Value and Frequency

Pixel Intensity Value Frequency

6 5

9 8

7 9

8 9

5 3

10 30

Huffman Tree built successfully!

Huffman Codes:

Pixel Intensity Value: 10, Code: 1

Pixel Intensity Value: 5, Code: 0111

Pixel Intensity Value: 6, Code: 0110

Pixel Intensity Value: 9, Code: 010

Pixel Intensity Value: 8, Code: 001

Pixel Intensity Value: 7, Code: 000

Huffman Codes and Sizes:

Pixel Intensity Value Frequency Code-word Size

10 30 1 30\*1

8 9 001 9\*3

7 9 000 9\*3

9 8 010 8\*3

6 5 0110 5\*4

5 3 0111 3\*4

Total Size: 140

Compressed Image Bitstream:

10111011111001000000101001110011000000011010100100011110110101000111001101101111100110110010110100001101100000000010000011110000100010101111

Menu:

1. Compress Image

2. Exit

Enter your choice:

## I5

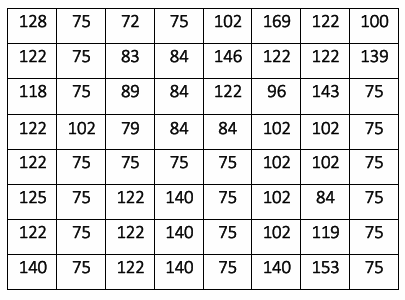


Table-1: Pixel Intensity Value and Frequency

Pixel Intensity Value Frequency

153 1

119 1

140 5

125 1

79 1

143 1

96 1

89 1

118 1

139 1

146 1

84 5

83 1

100 1

122 11

169 1

102 8

72 1

75 20

128 1

Huffman Tree built successfully!

Huffman Codes:

Pixel Intensity Value: 122, Code: 11

Pixel Intensity Value: 153, Code: 10111

Pixel Intensity Value: 125, Code: 101101

Pixel Intensity Value: 119, Code: 101100

Pixel Intensity Value: 169, Code: 101011

Pixel Intensity Value: 100, Code: 101010

Pixel Intensity Value: 128, Code: 101001

Pixel Intensity Value: 72, Code: 101000

Pixel Intensity Value: 139, Code: 100111

Pixel Intensity Value: 118, Code: 100110

Pixel Intensity Value: 83, Code: 100101

Pixel Intensity Value: 146, Code: 100100

Pixel Intensity Value: 143, Code: 100011

Pixel Intensity Value: 79, Code: 100010

Pixel Intensity Value: 89, Code: 100001

Pixel Intensity Value: 96, Code: 100000

Pixel Intensity Value: 102, Code: 011

Pixel Intensity Value: 84, Code: 0101

Pixel Intensity Value: 140, Code: 0100

Pixel Intensity Value: 75, Code: 00

Huffman Codes and Sizes:

Pixel Intensity Value Frequency Code-word Size

75 20 00 20\*2

122 11 11 11\*2

102 8 011 8\*3

140 5 0100 5\*4

84 5 0101 5\*4

100 1 101010 1\*6

128 1 101001 1\*6

72 1 101000 1\*6

139 1 100111 1\*6

118 1 100110 1\*6

83 1 100101 1\*6

146 1 100100 1\*6

143 1 100011 1\*6

79 1 100010 1\*6

89 1 100001 1\*6

96 1 100000 1\*6

153 1 10111 1\*5

125 1 101101 1\*6

119 1 101100 1\*6

169 1 101011 1\*6

Total Size: 215

Compressed Image Bitstream:

10100100101000000111010111110101011001001010101100100111110011110011000100001010111100000100011001101110001001010101011011001100000000011011001011010011010000011010100110011010000011101100000100001101000001001011100

Menu:

1. Compress Image

2. Exit

Enter your choice:

## I6

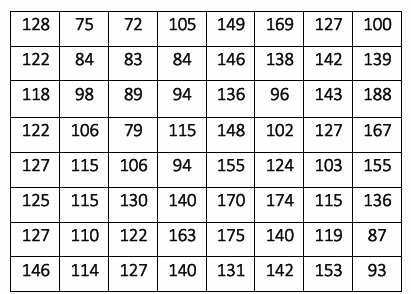


Table-1: Pixel Intensity Value and Frequency

Pixel Intensity Value Frequency

93 1

153 1

131 1

114 1

87 1

119 1

175 1

163 1

110 1

174 1

170 1

140 3

130 1

125 1

103 1

124 1

155 2

167 1

102 1

148 1

115 4

79 1

106 2

188 1

143 1

96 1

136 2

94 2

89 1

98 1

118 1

139 1

142 2

138 1

146 2

83 1

84 2

122 3

100 1

127 5

169 1

149 1

105 1

72 1

75 1

128 1

Huffman Tree built successfully!

Huffman Codes:

Pixel Intensity Value: 140, Code: 1111

Pixel Intensity Value: 122, Code: 1110

Pixel Intensity Value: 87, Code: 110111

Pixel Intensity Value: 114, Code: 110110

Pixel Intensity Value: 175, Code: 110101

Pixel Intensity Value: 119, Code: 110100

Pixel Intensity Value: 115, Code: 1100

Pixel Intensity Value: 100, Code: 101111

Pixel Intensity Value: 83, Code: 101110

Pixel Intensity Value: 149, Code: 101101

Pixel Intensity Value: 169, Code: 101100

Pixel Intensity Value: 118, Code: 101011

Pixel Intensity Value: 98, Code: 101010

Pixel Intensity Value: 138, Code: 101001

Pixel Intensity Value: 139, Code: 101000

Pixel Intensity Value: 106, Code: 10011

Pixel Intensity Value: 94, Code: 10010

Pixel Intensity Value: 72, Code: 100011

Pixel Intensity Value: 105, Code: 100010

Pixel Intensity Value: 128, Code: 100001

Pixel Intensity Value: 75, Code: 100000

Pixel Intensity Value: 125, Code: 011111

Pixel Intensity Value: 130, Code: 011110

Pixel Intensity Value: 124, Code: 011101

Pixel Intensity Value: 103, Code: 011100

Pixel Intensity Value: 110, Code: 011011

Pixel Intensity Value: 163, Code: 011010

Pixel Intensity Value: 170, Code: 011001

Pixel Intensity Value: 174, Code: 011000

Pixel Intensity Value: 143, Code: 010111

Pixel Intensity Value: 188, Code: 010110

Pixel Intensity Value: 89, Code: 010101

Pixel Intensity Value: 96, Code: 010100

Pixel Intensity Value: 102, Code: 010011

Pixel Intensity Value: 167, Code: 010010

Pixel Intensity Value: 79, Code: 010001

Pixel Intensity Value: 148, Code: 010000

Pixel Intensity Value: 142, Code: 00111

Pixel Intensity Value: 155, Code: 00110

Pixel Intensity Value: 136, Code: 00101

Pixel Intensity Value: 84, Code: 00100

Pixel Intensity Value: 146, Code: 00011

Pixel Intensity Value: 93, Code: 000101

Pixel Intensity Value: 131, Code: 0001001

Pixel Intensity Value: 153, Code: 0001000

Pixel Intensity Value: 127, Code: 0000

Huffman Codes and Sizes:

Pixel Intensity Value Frequency Code-word Size

127 5 0000 5\*4

115 4 1100 4\*4

140 3 1111 3\*4

122 3 1110 3\*4

94 2 10010 2\*5

142 2 00111 2\*5

155 2 00110 2\*5

136 2 00101 2\*5

84 2 00100 2\*5

146 2 00011 2\*5

106 2 10011 2\*5

118 1 101011 1\*6

98 1 101010 1\*6

138 1 101001 1\*6

139 1 101000 1\*6

114 1 110110 1\*6

175 1 110101 1\*6

72 1 100011 1\*6

105 1 100010 1\*6

128 1 100001 1\*6

75 1 100000 1\*6

125 1 011111 1\*6

130 1 011110 1\*6

124 1 011101 1\*6

103 1 011100 1\*6

110 1 011011 1\*6

163 1 011010 1\*6

170 1 011001 1\*6

174 1 011000 1\*6

143 1 010111 1\*6

188 1 010110 1\*6

89 1 010101 1\*6

96 1 010100 1\*6

102 1 010011 1\*6

167 1 010010 1\*6

79 1 010001 1\*6

148 1 010000 1\*6

119 1 110100 1\*6

87 1 110111 1\*6

100 1 101111 1\*6

83 1 101110 1\*6

149 1 101101 1\*6

93 1 000101 1\*6

131 1 0001001 1\*7

153 1 0001000 1\*7

169 1 101100 1\*6

Total Size: 342

Compressed Image Bitstream:

100001100000100011100010101101101100000010111111100010010111000100000111010010011110100010101110101001010110010001010101000101110101101110100110100011100010000010011000001001000001100100111001000110011101011100001100111111100011110111101100101100011000010100000110111110011010110101111111010011011100011110110000011110001001001110001000000101

Menu:

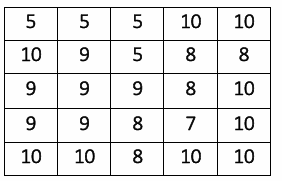
1. Compress Image

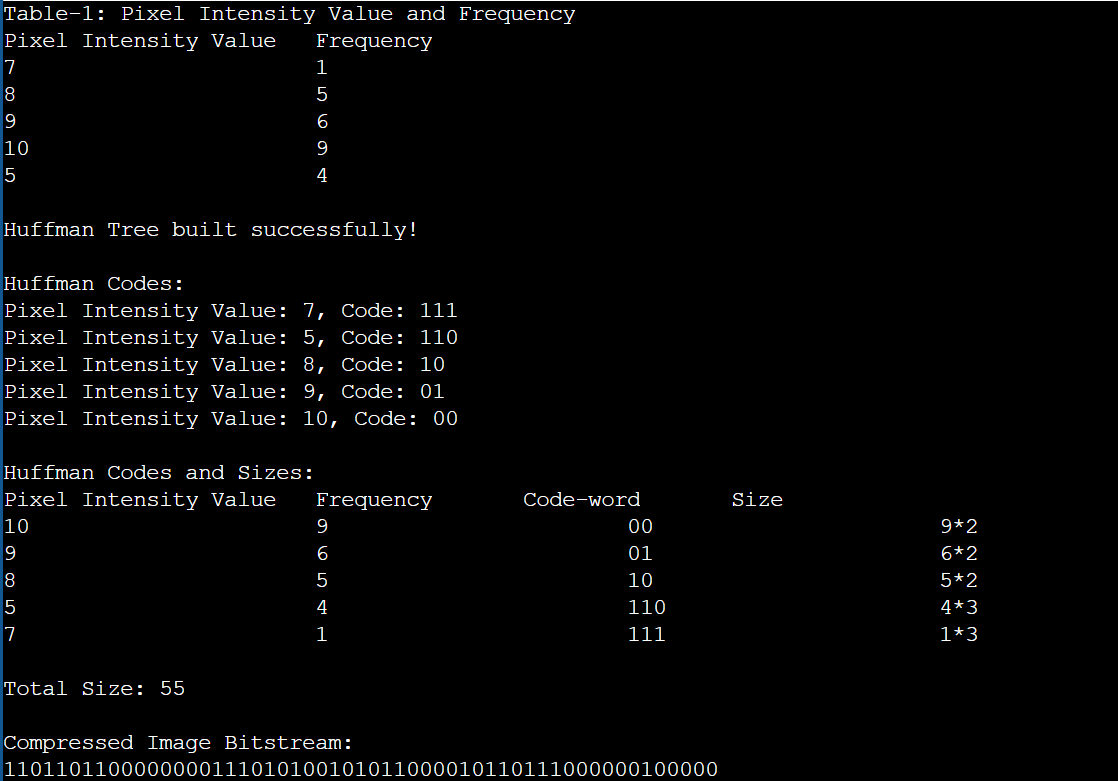
2. Exit

Enter your choice: 2

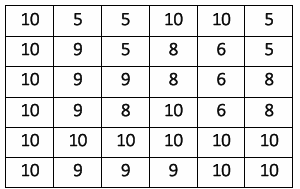
# Comparison-Appendix A:

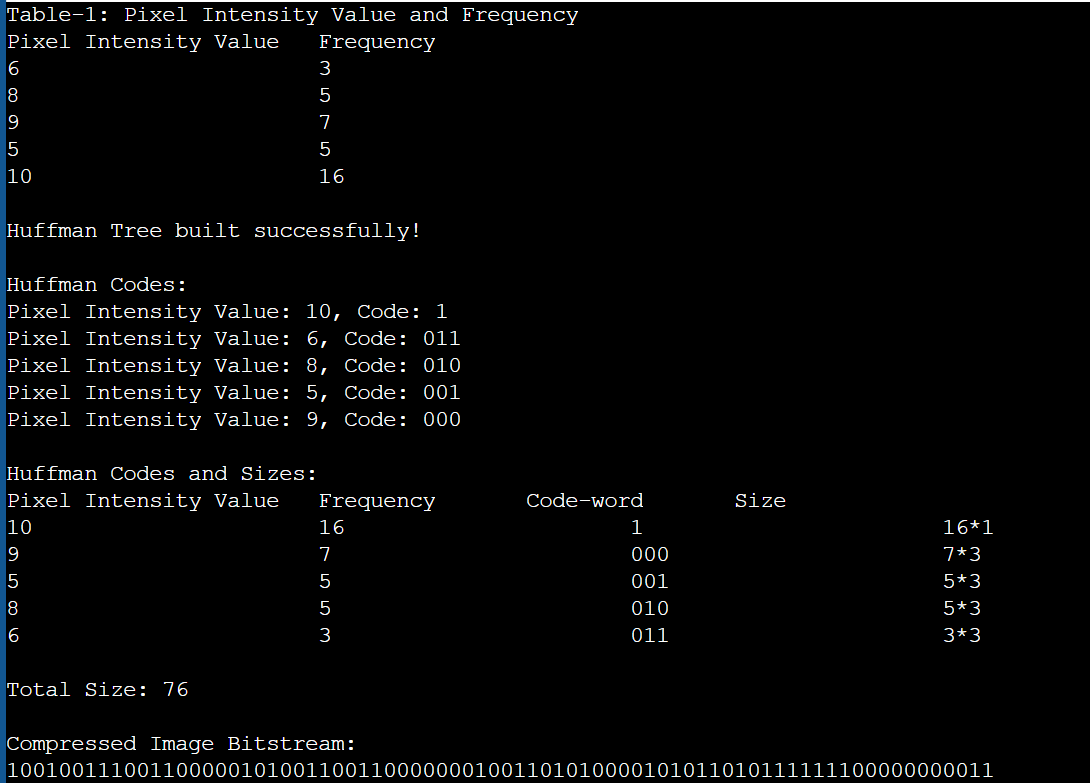
## I1



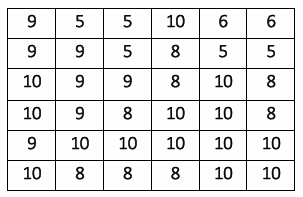


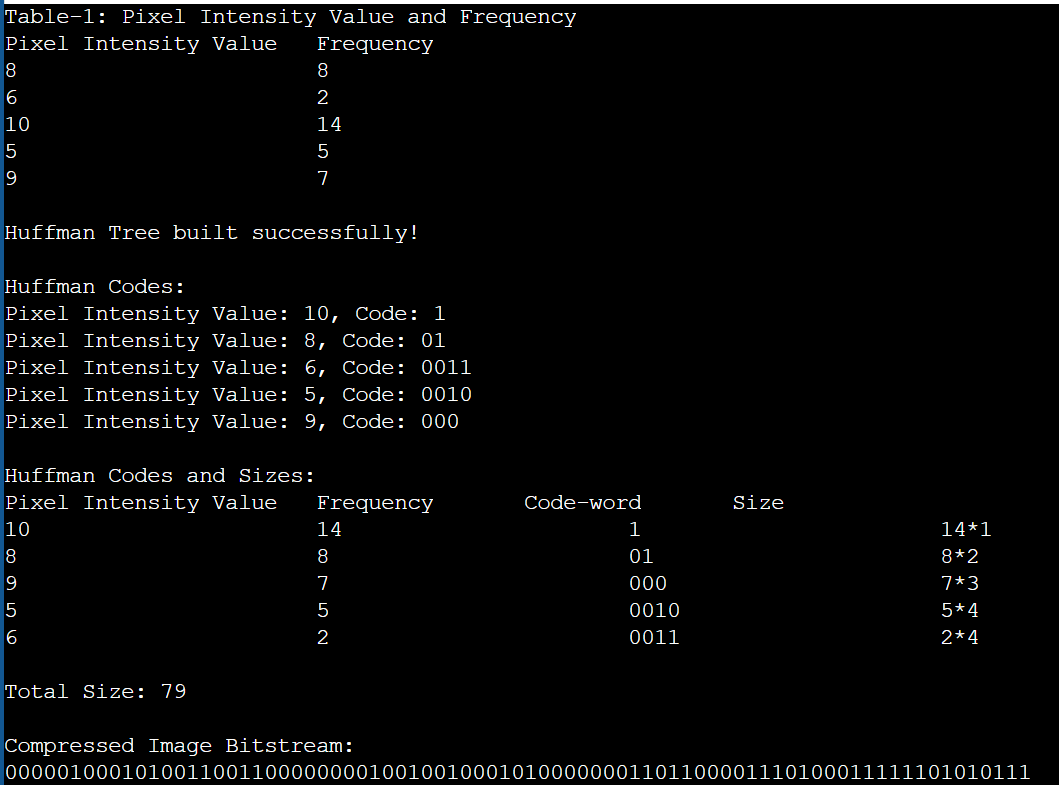
## I2





## I3





# References:

[Huffman Coding | Greedy Algo-3 - GeeksforGeeks](https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/)

[Huffman Coding Algorithm (programiz.com)](https://www.programiz.com/dsa/huffman-coding)