Huffman Coding

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfilment of the requirements to award the degree of

Bachelor of Technology
In
Computer Science and Engineering

School of Engineering and Sciences

Submitted By

Snehitha Pavuluri (AP22110011023)
Kusuma Priya (AP22110011048)
Chaitanya Kumar (AP22110011021)
Tarun Mogali (AP22110010989)

Under the Guidance of
Elakkiya E

SRM University–AP
Neerukonda, Mangalagiri, Guntur
Andhra Pradesh – 522 240

[May, 2024]

Aim of the Project :-

Huffman Coding _ Efficient Data Compression for Resource-Constrained Systems.

Description of the Project :-

Huffman coding involves constructing a binary tree based on the frequency of characters in the input data and assigning variable-length codes to each character based on their frequency in the tree.

Ally :-

Snehitha Pavuluri
Kusuma Priya
Chaitanya Kumar Reddy
Tarun Mogali

I **Snehitha** implemented functions for building the Huffman tree by developing the logic to merge nodes and construct the tree efficiently. Testing the Huffman coding implementation with different Datasets. I collaborated to develop the functions for compressing and decompressing data using Huffman codes.
KusumaPriya has ensured that the compression and decompression algorithms are correct and efficient, She analyses the experimental results and draws conclusions.
Chaitanya Kumar has documented the project details, including implementation approach, algorithms used and experimental setup. He wrote a comprehensive report summarising the project, its objectives, implementation details, experimental results and conclusions.
Tarun has ensured clarity, coherence and professionalism in the documentation. He integrated components of the project and also tested to identify and fix bugs or issues, also measured compression ratios, execution time and memory usage.
Each team member can contribute by presenting their respective parts of the project. Demonstrate the functionality, performance, and significance of the Huffman coding implementation.

Code:-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_TREE_HT 100
```

```
struct MinHeapNode {
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
};

struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};

struct MinHeapNode* newNode(char data, unsigned freq) {
    struct MinHeapNode* temp = (struct MinHeapNode*)malloc(sizeof(struct MinHeapNode));
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;
    return temp;
}

struct MinHeap* createMinHeap(unsigned capacity) {
    struct MinHeap* minHeap = (struct MinHeap*)malloc(sizeof(struct MinHeap));
    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct MinHeapNode**)malloc(minHeap->capacity * sizeof(struct
MinHeapNode*));
    return minHeap;
}

void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b) {
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

void minHeapify(struct MinHeap* minHeap, int idx) {
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size &&
        minHeap->array[left]->freq < minHeap->array[smallest]->freq)
        smallest = left;

    if (right < minHeap->size &&
        minHeap->array[right]->freq < minHeap->array[smallest]->freq)
        smallest = right;
    if (smallest != idx) {
        swapMinHeapNode(&minHeap->array[smallest], &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}

int isSizeOne(struct MinHeap* minHeap) {
    return (minHeap->size == 1);
}

struct MinHeapNode* extractMin(struct MinHeap* minHeap) {
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}

void insertMinHeap(struct MinHeap* minHeap, struct MinHeapNode* minHeapNode) {
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {
        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }
    minHeap->array[i] = minHeapNode;
}

void buildMinHeap(struct MinHeap* minHeap) {
    int n = minHeap->size - 1;
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

void printArr(int arr[], int n) {
    int i;
    for (i = 0; i < n; ++i)
        printf("%d", arr[i]);
    printf("\n");
}

int isLeaf(struct MinHeapNode* root) {
    return !(root->left) && !(root->right);
}

struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size) {
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
    minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
}

struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size) {
    struct MinHeapNode *left, *right, *top;
    struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);
    while (!isSizeOne(minHeap)) {
        left = extractMin(minHeap);
        right = extractMin(minHeap);
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}

void printCodes(struct MinHeapNode* root, int arr[], int top) {
    if (root->left) {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }
    if (root->right) {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }
    if (isLeaf(root)) {
        printf("%c: ", root->data);
        printArr(arr, top);
    }
}

void HuffmanCodes(char data[], int freq[], int size) {
    struct MinHeapNode* root = buildHuffmanTree(data, freq, size);
    int arr[MAX_TREE_HT], top = 0;
    printCodes(root, arr, top);
}

int main() {
    char arr[] = {'a', 'b', 'c', 'd', 'e', 'f'};
    int freq[] = {5, 9, 12, 13, 16, 45};
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("Huffman codes are:\n");
    HuffmanCodes(arr, freq, size);
    return 0;
}
```

Explanation :-

__struct MinHeapNode:__ Represents a node in the Huffman tree. It contains the character (data), its frequency (freq), and pointers to its left and right children.
__struct MinHeap:__ Represents a min-heap, a binary tree where the parent node has a frequency less than or equal to the frequencies of its children. It keeps track of the heap size, capacity, and an array of pointers to MinHeapNode elements.

Functions :-

1) **newNode:** Creates a new MinHeapNode with the given character and frequency.
2) **createMinHeap:** Creates a new min-heap with the specified capacity.
3) **swapMinHeapNode:** Swaps two MinHeapNode pointers.
4) **minHeapify:** Maintains the min-heap property by recursively fixing the subtree with root at index idx.
5) **isSizeOne:** Checks if the size of the min-heap is one.
6) **extractMin:** Extracts the minimum node from the min-heap.
7) **insertMinHeap:** Inserts a new node into the min-heap and maintains the min-heap property.
8) **buildMinHeap:** Builds a min-heap from the given array of nodes.
9) **printArr:** Prints an array of integers.
10) **isLeaf:** Checks if a given node is a leaf node.
11) **createAndBuildMinHeap:** Creates a min-heap from the given data and frequency arrays.
12) **buildHuffmanTree:** Builds the Huffman tree from the given data and frequency arrays.
13) **printCodes:** Recursively traverses the Huffman tree to print Huffman codes for each character.
14) **HuffmanCodes:** Main function to generate Huffman codes for a given set of characters and their frequencies.

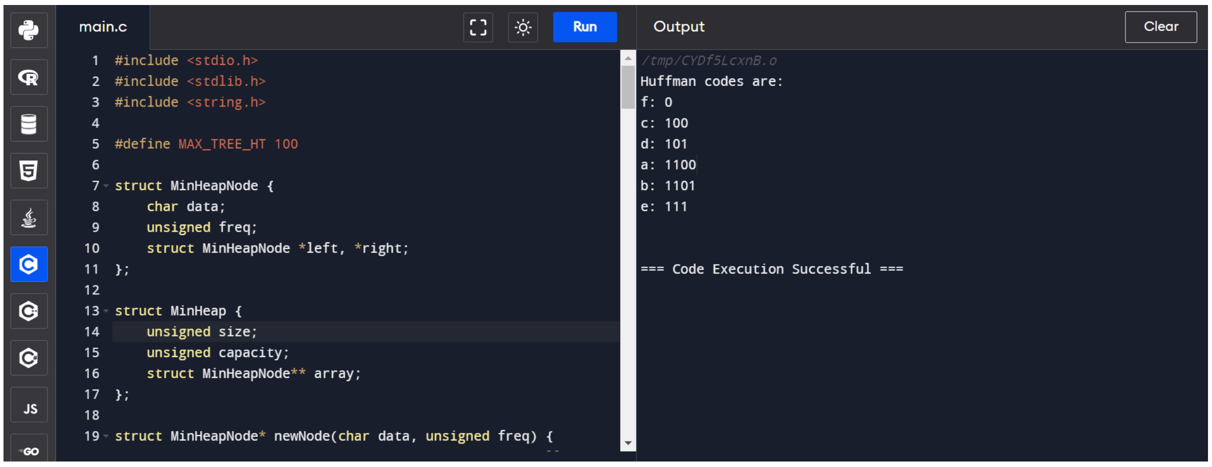
Main Functions :-
<< Defines an array arr containing characters and an array freq containing their corresponding frequencies.
<< Calculates the size of the arrays.
<< Calls HuffmanCodes function to generate Huffman codes for the provided data.

Execution :-

1) The program initializes with the provided character frequencies and constructs a Huffman tree.
2) It then traverses the Huffman tree to print the Huffman codes for each character.

Overall, this code efficiently generates Huffman codes for a given set of characters and their frequencies, demonstrating the core concepts of Huffman coding.

Output and Test Case :-



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_TREE_HT 100

struct MinHeapNode {
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
};

struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};

struct MinHeapNode* newNode(char data, unsigned freq) {
```

```
/tmp/CYDf5LcxnB.o
Huffman codes are:
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111


=== Code Execution Successful ===
```