Algo Lab 9

Submitted By: Rohan Verma (1510110508)

generate.c

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

int main(){
    srand(time(NULL));

    for(int i = 0; i < 100; i++)
        printf("%c", 'a' + rand()%4);

    return 0;
}
```

huffman.c

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

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

struct MinHeap
{
    unsigned int size, capacity;
    struct Node **array;
};

struct Node* newNode(char data, unsigned int freq){
    struct Node* t = (struct Node*) malloc(sizeof(struct Node));
    t->left = t->right = NULL;
```

```
    t->data = data;
    t->freq = freq;
    return t;
}

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

void swap(struct Node** a, struct Node** b){
    struct Node *t = *a;
    *a = *b;
    *b = t;
}

void minHeapify(struct MinHeap* h, int idx){
    int m = idx;
    int l = 2*idx + 1;
    int r = 2*idx + 2;

    if(l < h->size && h->array[l]->freq < h->array[m]->freq)
        m = l;

    if(r < h->size && h->array[r]->freq < h->array[m]->freq)
        m = r;

    if(m != idx){
        swap(&h->array[m], &h->array[idx]);
        minHeapify(h, m);
    }
}

struct Node* extractMin(struct MinHeap* h){
    struct Node* t = h->array[0];
    h->array[0] = h->array[h->size - 1];
    h->size -= 1;
    minHeapify(h, 0);
    return t;
}
```

```
void insert(struct MinHeap* h, struct Node* n){
    h->size += 1;
    int i = h->size - 1;
    while(i && n->freq < h->array[(i-1)/2]->freq){
        h->array[i] = h->array[(i-1)/2];
        i = (i-1)/2;
    }
    h->array[i] = n;
}

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

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

struct MinHeap* create_and_build_MinHeap(char s[], int f[], int size){
    struct MinHeap* h = createMinHeap(size);
    for(int i = 0; i < size; i++)
        h->array[i] = newNode(s[i], f[i]);
    h->size = size;
    build(h);
    return h;
}

struct Node* buildHuffmanTree(char s[], int f[], int size){
    struct Node *l, *r, *t;

    struct MinHeap* h = create_and_build_MinHeap(s,f,size);

    while(h->size != 1){
        l = extractMin(h);
        r = extractMin(h);

        t = newNode('#', l->freq + r->freq);
        t->left = l;
        t->right = r;
        insert(h, t);
```

```
    }

    return extractMin(h);
}

void getCode(struct Node* root, int a[], char c, int t){
    if (root->left){
        a[t] = 0;
        getCode(root->left, a, c, t+1);
    }
    if (root->right){
        a[t] = 1;
        getCode(root->right, a, c, t+1);
    }
    if(isLeaf(root) && root->data == c){
        for(int i = 0; i < t; i++)
            printf("%d", a[i]);
    }
}

int* calculate_frequency(char* s, int* f){
    int i = 0;

    while(s[i++] != '\0')
        f[s[i] - 'a'] += 1;

    return f;
}
int main(){
    char input_string[255];
    scanf("%s", &input_string);
    printf("Input String: %s\n", input_string);
    char arr[] = {'a', 'b', 'c', 'd'};
    int freq[] = {0,0,0,0};
    calculate_frequency(input_string, freq);
    int size = sizeof(arr)/sizeof(arr[0]);
    struct Node* root = buildHuffmanTree(arr, freq, size);
    int code[100];

    int k = 0;
    while(input_string[k++] != '\0')
        getCode(root, code, input_string[k], 0);
```

```
    printf("\n");
    return 0;
}
```

Screenshot