



ALGORITHMS ASSIGNMENT

Huffman Code



MAY 23, 2020
SYED TAFREED NUMAN
Sc ID 18-15-043

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;
```

```
}
```

OUTPUT TABLE:

Character	TEXT 1 (Length 100)		TEXT 2 (Length 1000)		TEXT 3 (Length 10000)	
a	12	011	99	001	1097	010
b	9	001	157	111	178	11110
c	14	100	145	101	1357	100
d	8	1011	95	000	759	1110
e	4	0000	58	10001	1009	001
f	5	0001	103	010	598	11111
g	10	010	74	1001	951	000
h	15	110	146	110	1417	101
i	16	111	12	10000	1533	110
j	7	1010	111	011	1101	011