Design Analysis and Algorithm – Lab Work

Week 8

NAME : VIGHRANTH SK

ROLL NO : CH.SC.U4CSE24149

Question 1:Write a Program to perform Huffman Coding on "DATAANALYTICSANDINTELLIGENCELABORATORY":

Code:

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

#define MAX 100
#define MAX_TREE_HT 100

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

struct Heap {
  int size;
  int capacity;
  struct Node** array;
};

struct Node* createNode(char ch, int freq) {
  struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
  temp->ch = ch;
  temp->freq = freq;
  temp->left = temp->right = NULL;
```

```
    return temp;
}

struct Heap* createHeap(int capacity) {
    struct Heap* heap = (struct Heap*)malloc(sizeof(struct Heap));
    heap->size = 0;
    heap->capacity = capacity;
    heap->array = (struct Node**)malloc(capacity * sizeof(struct Node*));
    return heap;
}

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

void heapify(struct Heap* heap, int idx) {
    int smallest = idx;
    int left = 2*idx + 1;
    int right = 2*idx + 2;

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

    if(right < heap->size && heap->array[right]->freq < heap->array[smallest]->freq)
        smallest = right;
```

```
    if(smallest != idx) {
        swap(&heap->array[smallest], &heap->array[idx]);
        heapify(heap, smallest);
    }
}

struct Node* extractMin(struct Heap* heap) {
    struct Node* temp = heap->array[0];
    heap->array[0] = heap->array[heap->size - 1];
    heap->size--;
    heapify(heap, 0);
    return temp;
}

void insertHeap(struct Heap* heap, struct Node* node) {
    heap->size++;
    int i = heap->size - 1;

    while(i && node->freq < heap->array[(i-1)/2]->freq) {
        heap->array[i] = heap->array[(i-1)/2];
        i = (i-1)/2;
    }
    heap->array[i] = node;
}

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

```
void printCodes(struct Node* 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->ch);
      for(int i=0;i<top;i++)
         printf("%d", arr[i]);
      printf("\n");
   }
}

void buildHuffman(char data[], int freq[], int size) {
   struct Node *left, *right, *top;
   struct Heap* heap = createHeap(size);

   for(int i=0;i<size;i++)
      heap->array[i] = createNode(data[i], freq[i]);

   heap->size = size;
```

```
    for(int i=(size-1)/2;i>=0;i--)
        heapify(heap,i);

    while(heap->size != 1) {
        left = extractMin(heap);
        right = extractMin(heap);

        top = createNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;

        insertHeap(heap, top);
    }

    int arr[MAX_TREE_HT];
    printCodes(extractMin(heap), arr, 0);
}

int main() {

    char text[] = "DATAANALYTICSANDINTELLIGENCELABORATORY";
    int frequency[256] = {0};

    for(int i=0;text[i]!='\0';i++)
        frequency[text[i]]++;

    char data[MAX];
```

```
    int freq[MAX];
    int size = 0;

    for(int i=0;i<256;i++) {
        if(frequency[i] > 0) {
            data[size] = (char)i;
            freq[size] = frequency[i];
            size++;
        }
    }

    buildHuffman(data, freq, size);

    return 0;
}
```

Output:

```
C:\Users\ROHITH SN\Downloads\Vighranth>week8
'week8' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\ROHITH SN\Downloads\Vighranth>gcc week8.c

C:\Users\ROHITH SN\Downloads\Vighranth>a
N : 000
L : 001
O : 0100
R : 0101
C : 0110
B : 01110
G : 01111
T : 100
Y : 1010
S : 10110
D : 10111
E : 1100
I : 1101
A : 111
```

Time Complexity

The time complexity of Huffman Coding mainly depends on the number of **unique characters (n)**. First, we build a min-heap using all unique characters, which takes **O(n)**

time. Then, we repeatedly extract the two minimum-frequency nodes and insert their combined node back into the heap. Each extraction and insertion operation takes **O(log n)** time because the heap must maintain its ordering property. Since this merging process happens **(n − 1)** times, the total time required for tree construction becomes **O(n log n)**. Finally, printing the Huffman codes requires a traversal of the tree, which takes **O(n)** time. Therefore, the overall time complexity of Huffman Coding is **O(n log n)**.

Space Complexity

The space complexity is determined by the storage used for the min-heap and the Huffman tree. The min-heap stores **n nodes**, requiring **O(n)** space. The Huffman tree itself contains **2n − 1 nodes** (including internal and leaf nodes), which is also proportional to **O(n)**. Additionally, the recursion stack used during tree traversal (while printing codes) takes at most **O(n)** space in the worst case (when the tree becomes skewed). Since all these space requirements grow linearly with the number of unique characters, the overall space complexity of Huffman Coding is **O(n)**.