LE5.1 WAP to implement the file or Code compression using Huffman's algorithm.

Program:-

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_TREE_HT100.
Struct MinHeap Node{
    char data;
    Unsigned freq;
    Struct MinHeap Node * left * right;
};
Struct MinHeap{
    unsigned size;
    unsigned capacity;
    Struct MinHeap Node **array;
};
Struct MinHeap Node*newNode (char
    data ,unsigned freq){
    Struct MinHeap Node* temp=(struct
        MinHeap Node*)
```

```
malloc ( sizeof ( struct  MinHeap Node ));
  temp -> left  = temp-> right = NULL;
  temp -> data  = data ;
  temp -> freq = freq;
  return temp;
}

Struct  MinHeap* createMin Heap (unsigned
                    capacity){

  Struct MinHeap * minHeap =
        (Struct MinHeap * malloc (size of (
                    Struct Min Heap ));

  minHeap -> size = 0;
  minHeap -> capacity = capacity;
  minHeap -> array = (Struct  Min Heap Node**)
  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 (smallest != idx){
    SwapMinHeapNode(&minHeap -> array
    [smallest], &minHeap -> array[idx]);
minHeapify(minHeap, smallest);
}
}
```

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

Void InsertMinHeap (struct MinHeap* minHeap){
    int n= minHeap -> size -1;
    int i;
    for (i=(n-1)/2; i>0; --i){
        ~~minHH~~ minHeapify (minHeap, i);
    }

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

Struct MinHeapNode * buildHuffmanTree(char
    data[], int freq[], int size){
    Struct MinHeapNode *left, *right, *top;
    Struct MinHeap* minHeap= CreateAnd
        BuildMinHeap(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'};
    int freq[]={3,5,6,4,2};
    int size = sizeOf(arr) / sizeOf(arr[0]);
    HuffmanCodes(arr, freq, size);
    return 0;
}
```

# INPUT /OUTPUT :-

d : 00

e : 010

a : 011

b : 10

C : 11

5.4 WAP to implement the activity-
- Selection Problem.

Program:

```
#include <stdio.h>
Void printMaxActivites (int s[], int f[],
int n){
int i,j;
printf("following activites are selected:\n");
i=0;
printf("%d\t", i);
for(j=1; j<n; j++){
if(s[j] >=f[i]){
printf("%d\t", j);
i=j;
}
}
}
int main(){
int s[]={1,3,0,5,8,5};
```

```
int f[] = {2, 4, 6, 7, 9, 9};
int n = sizeof(s) / sizeof(s[0]);
printMaxActivities(s, f, n);
return 0;
}
```

INPUT / OUTPUT :-

following activites are selected:

0 1 3 4.