DAA LAB

Shubham Singh
1906587

LAB-6

LE:5.1:- WAP to implement the file or code compression using Huffman's algorithm.

Programs-
```
#include<stdio.h>
#include <stdlib.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 MinHeap Node * 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(structMinHeap));
   min Heap->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 (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)
       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= 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'};
     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
```

LE:5.4:- WAP to implement the activity-selection problem.

Program:-

```
#include<stdio.h>
void printMaxActivities(int s[], int f[], int n)
{
   int i, j;
   printf("Follwoing activities 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 activities are selected:
 0   1   3   4
```