DHRUV GARG 16 BCE 1190

DSA LAB – Huffman codes

Aim : To implement the greedy algorithm technique for compression using Huffman Codes.

CODE

```
#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 MinHeapNode* 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(struct MinHeap));

   minHeap->size = 0;// current size is 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 (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq)

      smallest = right;

   if (smallest != idx)
   {
      swapMinHeapNode(&minHeap->array[smallest],
&minHeap->array[idx]);
      minHeapify(minHeap, smallest);
   }
}

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

struct MinHeapNode* extractMin(struct MinHeap*
minHeap){
   struct MinHeapNode* temp = minHeap->array[0];
   minHeap->array[0] = minHeap->array[minHeap-
>size - 1];
   --minHeap->size;
   minHeapify(minHeap, 0);
   return temp;
}

void insertMinHeap(struct MinHeap* minHeap, struct
MinHeapNode* minHeapNode)
{
   ++minHeap->size;
   int i = minHeap->size - 1;
   while (i && minHeapNode->freq < minHeap-
>array[(i - 1)/2]->freq){
         minHeap->array[i] = minHeap->array[(i - 1)/2];
   i = (i - 1)/2;
   }
   minHeap->array[i] = minHeapNode;
}
```

```
void buildMinHeap(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");
}

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

struct MinHeap* createAndBuildMinHeap(char data[],
int freq[], int size){
   struct MinHeap* minHeap = createMinHeap(size);
   int i;
   for(i=0;i<size;++i)
      minHeap->array[i] = newNode(data[i], freq[i]);
   minHeap->size = size;
   buildMinHeap(minHeap);
   return minHeap;
}

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', 'c', 'e', 'f', 'g', 'h', 'i', 'l', 'm', 'n', 'o',
'p', 'r', 's', 't', 'u', 'v', 'w', 'y'};

    int freq[] = {2,4,7,1,1,2,3,4,2,4,6,1,1,2,5,2,1,1,1};

    int size = sizeof(arr)/sizeof(arr[0]);

    HuffmanCodes(arr, freq, size);

    return 0;

}
```

NOTE :

The above letter and corresponding frequency array has been taken from the following sentence :

“welcome to vellore institute of technology chennai campus”

Thus the corresponding letter frequencies are :

a - 2

c - 4

e - 7

f - 1

g - 1

h - 2

i - 3

l - 4

m - 2

n - 4

o - 6

p - 1

r - 1

s - 2

t - 5

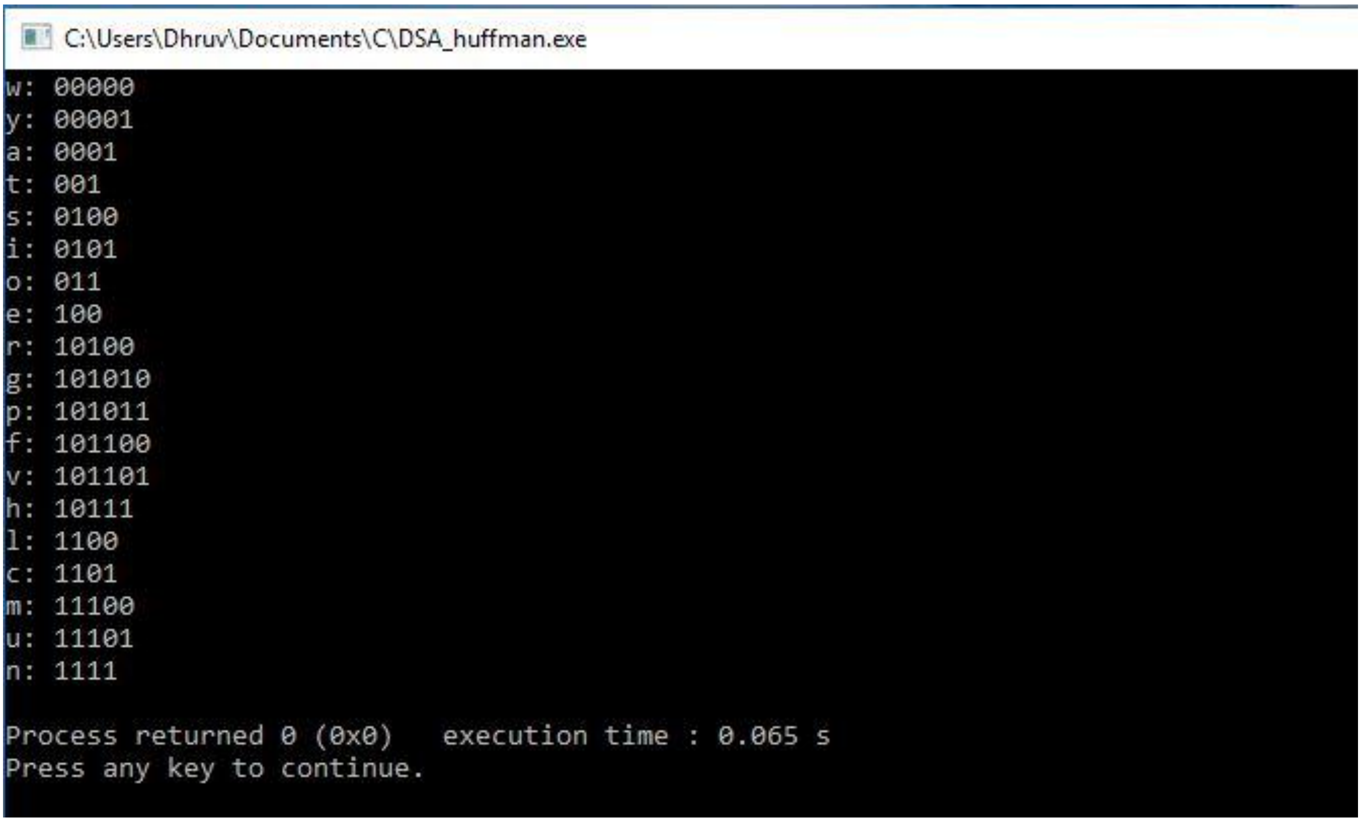
u - 2

v - 1

w - 1

y – 1

OUTPUT :



```
w: 00000
y: 00001
a: 0001
t: 001
s: 0100
i: 0101
o: 011
e: 100
r: 10100
g: 101010
p: 101011
f: 101100
v: 101101
h: 10111
l: 1100
c: 1101
m: 11100
u: 11101
n: 1111

Process returned 0 (0x0)   execution time : 0.065 s
Press any key to continue.
```

Thus we can see that :

a. More frequent letters have been given lower bit codes
b. The codes are of variable bit lengths
c. Huffman codes overcome the disadvantages of fixed bit length codes.

RESULT:

Greedy algorithm technique for compression of data using Huffman code has been implemented.