****

Assignment No:4

Title Name: Huffman Encoding using a greedy strategy

Name: Aniket Rajani

Class : BE **Div**: 1 **Batch**: A

Roll No: 405A008

****

Program:

```
// Huffman Coding in C++


#include  <iostream>

using namespace std;

#define MAX_TREE_HT 50


struct MinHNode {

  unsigned freq;

  char item;

  struct MinHNode *left, *right;

};


struct MinH {

  unsigned size;

  unsigned capacity;

  struct MinHNode **array;

};


// Creating Huffman tree node

struct MinHNode *newNode(char item, unsigned freq)

{

  struct MinHNode *temp = (struct MinHNode *)malloc(sizeof(struct MinHNode));

  temp->left = temp->right = NULL;
```

```
  temp->item = item;
  temp->freq = freq;
  return temp;
}

// Create min heap using given capacity
struct MinH *createMinH(unsigned capacity)
{
  struct MinH *minHeap = (struct MinH *)malloc(sizeof(struct MinH));
  minHeap->size = 0;
  minHeap->capacity = capacity;
  minHeap->array = (struct MinHNode **)malloc(minHeap->capacity * sizeof(struct MinHNode *));
  return minHeap;
}

// Print the array
void printArray(int arr[], int n)
{
  int i;
  for (i = 0; i < n; ++i)
    cout << arr[i];
  cout << "\n";
}

// Swap function
void swapMinHNode(struct MinHNode **a, struct MinHNode **b)
{
  struct MinHNode *t = *a;
  *a = *b;
  *b = t;
}
```

```
// Heapify
void minHeapify(struct MinH *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) {
    swapMinHNode(&minHeap->array[smallest],
         &minHeap->array[idx]);
    minHeapify(minHeap, smallest);
  }
}

// Check if size if 1
int checkSizeOne(struct MinH *minHeap)
{
  return (minHeap->size == 1);
}
// Extract the min
struct MinHNode *extractMin(struct MinH *minHeap)
{
  struct MinHNode *temp = minHeap->array[0];
  minHeap->array[0] = minHeap->array[minHeap->size - 1];
  --minHeap->size;
  minHeapify(minHeap, 0);
  return temp;
}
```

```
// Insertion
void insertMinHeap(struct MinH *minHeap, struct MinHNode *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;
}

// BUild min heap
void buildMinHeap(struct MinH *minHeap)
{
  int n = minHeap->size - 1;
  int i;
  for (i = (n - 1) / 2; i >= 0; --i)
    minHeapify(minHeap, i);
}
int isLeaf(struct MinHNode *root) {
  return !(root->left) && !(root->right);
}


struct MinH *createAndBuildMinHeap(char item[], int freq[], int size)
{
  struct MinH *minHeap = createMinH(size);
  for (int i = 0; i < size; ++i)
    minHeap->array[i] = newNode(item[i], freq[i]);
```

```
  minHeap->size = size;
  buildMinHeap(minHeap);
  return minHeap;
}

struct MinHNode *buildHfTree(char item[], int freq[], int size)
{
  struct MinHNode *left, *right, *top;
  struct MinH *minHeap = createAndBuildMinHeap(item, freq, size);
  while (!checkSizeOne(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 printHCodes(struct MinHNode *root, int arr[], int top)
{
  if (root->left) {
    arr[top] = 0;
    printHCodes(root->left, arr, top + 1);
  }
  if (root->right) {
    arr[top] = 1;
    printHCodes(root->right, arr, top + 1);
  }
  if (isLeaf(root)) {
```

```
    cout << root->item << "  | ";
    printArray(arr, top);
  }
}
// Wrapper function
void HuffmanCodes(char item[], int freq[], int size)
{
  struct MinHNode *root = buildHfTree(item, freq, size);
  int arr[MAX_TREE_HT], top = 0;
  printHCodes(root, arr, top);
}
int main()
{
  char arr[] = {'A', 'B', 'C', 'D'};
  int freq[] = {5, 1, 6, 3};
  int size = sizeof(arr) / sizeof(arr[0]);
  cout << "Char | Huffman code ";
  cout << "\n----------------------\n";
  HuffmanCodes(arr, freq, size);
}
```

Output:

Char | Huffman code

C | 0

B | 100

D | 101

A | 11