

Forum:[Hack Forums](#)

Thread: [Huffman coding encoding and decoding](#)

kakarot007 - April 19, 2019, 12:17 pm

Hello , i am learning algorithms and am trying to implement huffman coding algorithm. I am not able to get the decode part correctly . Below is my code

My answer is wrong

```
Code #include <iostream>
#include <string>
#include <map>
#include <vector> //or #include <list>
#include <queue>
// do not use using namespace std

// do not alter the struct declaration
struct hufftreenode {
    char character;
    int frequency;
    hufftreenode* left;
    hufftreenode* right;
};

// do not alter the typedef
typedef hufftreenode* hufftreeptr;

struct comp
{
    bool operator()(hufftreenode* l, hufftreenode* r)
    {
        // highest priority item has lowest frequency
        return l->frequency > r->frequency;
    }
};

// function to allocate new tree node
hufftreenode* getNode(char ch, int freq, hufftreenode* left, hufftreenode* right)
{
    hufftreenode* node = new hufftreenode();
    node->character = ch;
    node->frequency = freq;
    node->left = left;
    node->right = right;
```

```
    return node;
}
```

```
//utility function
```

```
bool valid_hufftree(const hufftreeptr hufftree);
// returns true if the input hufftree is a valid Huffman tree
// i.e. all the terminal nodes (leaves) have characters,
// all the non-leaf nodes have two sub-trees each,
// and the occurrence frequency of a non-leaf node equals
// the sum of the occurrence frequencies of its two sub-trees.
```

```
// traverse the Huffman Tree and store Huffman Codes
// in a map.
```

```
void encode(hufftreenode* root, std::string str,
            std::map<char, std::string> &huffmanCode)
{
    if (root == nullptr)
        return;
```

```
    // found a leaf node
    if (!root->left && !root->right) {
        huffmanCode[root->character] = str;
    }
```

```
    encode(root->left, str + "0", huffmanCode);
    encode(root->right, str + "1", huffmanCode);
}
```

```
// encodes an input message (a string of characters) into an encoded message (string of bits) using
the Huffman coding
```

```
// builds the Huffman tree and its associated encoding table (i.e. the association between the
characters in the tree leaves and their encodings (string of bits))
```

```
std::string huffencode(const std::string& message, std::map<char, int>& freqtable, hufftreeptr&
hufftree, std::map<char, std::string>& hufftable)
{
    for (char ch : message) {
        freqtable[ch]++;
    }
```

```
// Create a priority queue to store live nodes of
// Huffman tree;
std::priority_queue<hufftreenode*, std::vector<hufftreenode*>, comp> pq;
```

```
// Create a leaf node for each character and add it
// to the priority queue.
for (auto pair: freqtable) {
    pq.push(getNode(pair.first, pair.second, nullptr, nullptr));
}
```

```
// do till there is more than one node in the queue
while (pq.size() != 1)
```

```

{
// Remove the two nodes of highest priority
// (lowest frequency) from the queue
hufftreenode* left = pq.top(); pq.pop();
hufftreenode* right = pq.top(); pq.pop();

// Create a new internal node with these two nodes
// as children and with frequency equal to the sum
// of the two nodes' frequencies. Add the new node
// to the priority queue.
int sum = left->frequency + right->frequency;
pq.push(getNode('\0', sum, left, right));
}

// root stores pointer to root of Huffman Tree
hufftree = pq.top();

// traverse the Huffman Tree and store Huffman Codes
// in a map. Also prints them
encode(hufftree, "", hufftable);
std::cout << "Huffman Codes are :\n" << '\n';
for (auto pair: hufftable) {
std::cout << pair.first << " " << pair.second << '\n';
}

// get encoded string
std::string result;
for (char ch: message) {
result += hufftable[ch];
}

return result;
}

// traverse the Huffman Tree and decode the encoded string
std::string decode(hufftreenode* root, int &index, std::string str)
{
if (root == nullptr) {
return NULL;
}

// found a leaf node
if (!root->left && !root->right)
{
std::string s;
std::cout << "char : " << root->character;
s.push_back(root->character);
std::cout << "s : " << s << "\n";
return s;
}

index++;

```

```

    if (str[index] == '0')
        decode(root->left, index, str);
    else
        decode(root->right, index, str);
}
// decodes an encoded message (a string of bits) into the original message (a string of characters)
using the Huffman tree built during the encoding
std::string huffdecode(const std::string& encodedmessage, const hufftreeptr& hufftree)
{
    // traverse the Huffman Tree again and this time
    // decode the encoded string
    int index = -1;
    std::string result;
    std::cout<<"encoded msg : " << encodedmessage << "\n";
    while (index < (int)encodedmessage.size() - 2) {
        result += decode(hufftree, index, encodedmessage);
    }

    return result;
}

```

```

int main() {

```

```

    std::string message, encodedmessage, decodedmessage;
    hufftreeptr hufftree;
    std::map<char, int> freqtable;
    std::map<char, std::string> hufftable;

```

```

    message = "go go gophers";

```

```

    encodedmessage = huffencode(message, freqtable, hufftree, hufftable);

```

```

    std::cout<<"Encoded msg is : " << encodedmessage << "\n";

```

```

    // if(valid_hufftree(hufftree)) {
    // std::cout << "valid Huffman tree." << std::endl;
    // }
    // else {
    // std::cout << "not valid Huffman tree." << std::endl;
    // }

```

```

    // // 2) decoding (uncompression)
    decodedmessage = huffdecode(encodedmessage, hufftree);
    std::cout<<"Decoded msg is : " << decodedmessage << "\n";
    // should be "go go gophers"

```

```
if(decodedmessage == message) {  
    std::cout << "decoding OK." << std::endl;  
}  
else {  
    std::cout << "decoding not OK." << std::endl;  
}  
  
return 0;  
}
```