

# GeeksforGeeks

A computer science portal for geeks

## GeeksQuiz

- [Home](#)
- [Algorithms](#)
- [DS](#)
- [GATE](#)
- [Interview Corner](#)
- [Q&A](#)
- [C](#)
- [C++](#)
- [Java](#)
- [Books](#)
- [Contribute](#)
- [Ask a Q](#)
- [About](#)

[Array](#)

[Bit Magic](#)

[C/C++](#)

[Articles](#)

[GFactS](#)

[Linked List](#)

[MCQ](#)

[Misc](#)

[Output](#)

[String](#)

[Tree](#)

[Graph](#)

## Greedy Algorithms | Set 3 (Huffman Coding)

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code. The variable-length codes assigned to input characters are [Prefix Codes](#), means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.

Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is prefix of codes assigned to a and b. If the compressed bit stream is 0001, the decompressed output may be “cccd” or “ccb” or “acd” or “ab”.

See [this](#) for applications of Huffman Coding.

There are mainly two major parts in Huffman Coding

- 1) Build a Huffman Tree from input characters.
- 2) Traverse the Huffman Tree and assign codes to characters.

### ***Steps to build Huffman Tree***

Input is array of unique characters along with their frequency of occurrences and output is Huffman Tree.

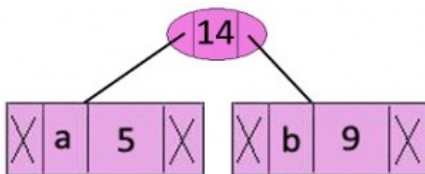
1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Let us understand the algorithm with an example:

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

**Step 1.** Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

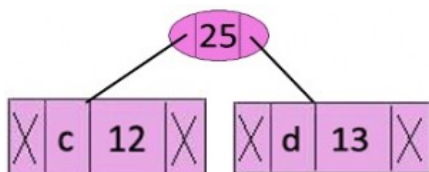
**Step 2** Extract two minimum frequency nodes from min heap. Add a new internal node with frequency  $5 + 9 = 14$ .



Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

character	Frequency
c	12
d	13
Internal Node	14
e	16
f	45

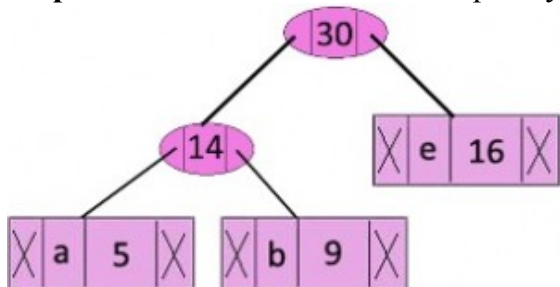
**Step 3:** Extract two minimum frequency nodes from heap. Add a new internal node with frequency  $12 + 13 = 25$



Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes.

character	Frequency
Internal Node	14
e	16
Internal Node	25
f	45

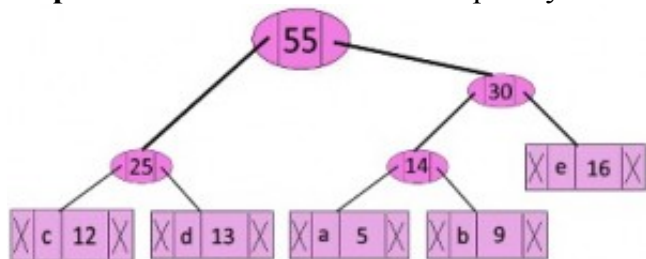
**Step 4:** Extract two minimum frequency nodes. Add a new internal node with frequency  $14 + 16 = 30$



Now min heap contains 3 nodes.

character	Frequency
Internal Node	25
Internal Node	30
f	45

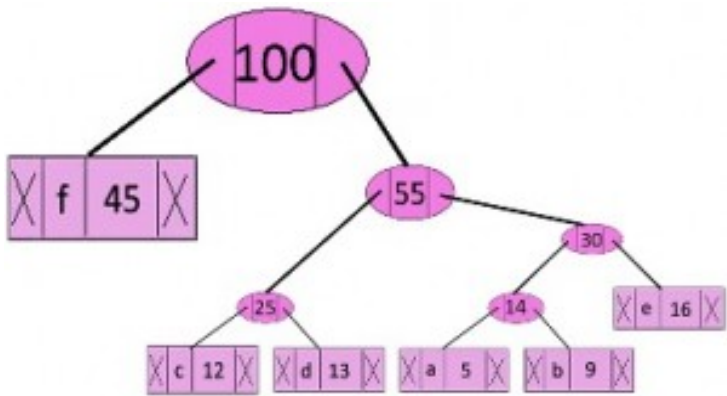
**Step 5:** Extract two minimum frequency nodes. Add a new internal node with frequency  $25 + 30 = 55$



Now min heap contains 2 nodes.

character	Frequency
f	45
Internal Node	55

**Step 6:** Extract two minimum frequency nodes. Add a new internal node with frequency  $45 + 55 = 100$



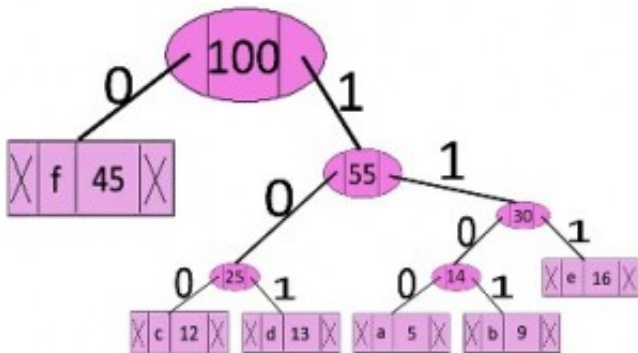
Now min heap contains only one node.

character	Frequency
Internal Node	100

Since the heap contains only one node, the algorithm stops here.

### ***Steps to print codes from Huffman Tree:***

Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



The codes are as follows:

character	code-word
f	0
c	100
d	101
a	1100
b	1101
e	111

```
// C program for Huffman Coding
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
// This constant can be avoided by explicitly calculating height of Huffman T
```

```
#define MAX_TREE_HT 100
```

```
// A Huffman tree node
```

```
struct MinHeapNode
```

```
{
```

```
    char data; // One of the input characters
```

```

    unsigned freq; // Frequency of the character
    struct MinHeapNode *left, *right; // Left and right child of this node
};

// A Min Heap: Collection of min heap (or Huffman tree) nodes
struct MinHeap
{
    unsigned size; // Current size of min heap
    unsigned capacity; // capacity of min heap
    struct MinHeapNode **array; // Array of minheap node pointers
};

// A utility function allocate a new min heap node with given character
// and frequency of the character
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;
}

// A utility function to create a min heap of given capacity
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;
}

// A utility function to swap two min heap nodes
void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b)
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

// The standard minHeapify function.
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);
    }
}

// A utility function to check if size of heap is 1 or not
int isSizeOne(struct MinHeap* minHeap)
{
    return (minHeap->size == 1);
}

// A standard function to extract minimum value node from heap
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;
}

// A utility function to insert a new node to Min Heap
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;
}

// A standard function to build min heap
void buildMinHeap(struct MinHeap* minHeap)
{
    int n = minHeap->size - 1;
    int i;
    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

// A utility function to print an array of size n
void printArr(int arr[], int n)

```

```

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

// Utility function to check if this node is leaf
int isLeaf(struct MinHeapNode* root)
{
    return !(root->left) && !(root->right) ;
}

// Creates a min heap of capacity equal to size and inserts all character of
// data[] in min heap. Initially size of min heap is equal to capacity
struct MinHeap* createAndBuildMinHeap(char data[], int freq[], int size)
{
    struct MinHeap* minHeap = createMinHeap(size);
    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i], freq[i]);
    minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
}

// The main function that builds Huffman tree
struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size)
{
    struct MinHeapNode *left, *right, *top;

    // Step 1: Create a min heap of capacity equal to size. Initially, there
    // modes equal to size.
    struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);

    // Iterate while size of heap doesn't become 1
    while (!isSizeOne(minHeap))
    {
        // Step 2: Extract the two minimum freq items from min heap
        left = extractMin(minHeap);
        right = extractMin(minHeap);

        // Step 3: Create a new internal node with frequency equal to the
        // sum of the two nodes frequencies. Make the two extracted node as
        // left and right children of this new node. Add this node to the min
        // '$' is a special value for internal nodes, not used
        top = newNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }

    // Step 4: The remaining node is the root node and the tree is complete.
    return extractMin(minHeap);
}

```

```

}

// Prints huffman codes from the root of Huffman Tree. It uses arr[] to
// store codes
void printCodes(struct MinHeapNode* root, int arr[], int top)
{
    // Assign 0 to left edge and recur
    if (root->left)
    {
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }

    // Assign 1 to right edge and recur
    if (root->right)
    {
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }

    // If this is a leaf node, then it contains one of the input
    // characters, print the character and its code from arr[]
    if (isLeaf(root))
    {
        printf("%c: ", root->data);
        printArr(arr, top);
    }
}

// The main function that builds a Huffman Tree and print codes by traversing
// the built Huffman Tree
void HuffmanCodes(char data[], int freq[], int size)
{
    // Construct Huffman Tree
    struct MinHeapNode* root = buildHuffmanTree(data, freq, size);

    // Print Huffman codes using the Huffman tree built above
    int arr[MAX_TREE_HT], top = 0;
    printCodes(root, arr, top);
}

// Driver program to test above functions
int main()
{
    char arr[] = {'a', 'b', 'c', 'd', 'e', 'f'};
    int freq[] = {5, 9, 12, 13, 16, 45};
    int size = sizeof(arr)/sizeof(arr[0]);
    HuffmanCodes(arr, freq, size);
    return 0;
}

```

f: 0  
c: 100



d: 101  
a: 1100  
b: 1101  
e: 111

**Time complexity:**  $O(n \log n)$  where  $n$  is the number of unique characters. If there are  $n$  nodes, `extractMin()` is called  $2 \cdot (n - 1)$  times. `extractMin()` takes  $O(\log n)$  time as it calls `minHeapify()`. So, overall complexity is  $O(n \log n)$ .

If the input array is sorted, there exists a linear time algorithm. We will soon be discussing in our next post.

### Reference:

[http://en.wikipedia.org/wiki/Huffman\\_coding](http://en.wikipedia.org/wiki/Huffman_coding)

This article is compiled by [Aashish Barnwal](#) and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

## Related Topics:

- [Linearity of Expectation](#)
- [Iterative Tower of Hanoi](#)
- [Count possible ways to construct buildings](#)
- [Build Lowest Number by Removing n digits from a given number](#)
- [Set Cover Problem | Set 1 \(Greedy Approximate Algorithm\)](#)
- [Find number of days between two given dates](#)
- [How to print maximum number of A's using given four keys](#)
- [Write an iterative  \$O\(\log y\)\$  function for  \$\text{pow}\(x, y\)\$](#)

Tags: [Greedy Algorithm](#), [Huffman Coding](#)



Tweet

2

+1

2

Writing code in comment? Please use [ideone.com](http://ideone.com) and share the link here.

19 Comments

GeeksforGeeks

1 Login ▾

♥ Recommend 3

🔗 Share

Sort by Newest ▾



Join the discussion...



**life coder** • 4 months ago

where can i use this algorithm in problem solving?

^ | ▾ • Reply • Share ›



**vipinkaushal** • 6 months ago

C++ implementation using STL

  
<http://ideone.com/87RnTj>  • Reply • Share ›**shubh** → vipinkaushal • 25 days ago

but what is complexity of it?

  • Reply • Share ›**dharam** • 7 months ago

Java implementation is here

<https://github.com/dharam3/DS/...>2   • Reply • Share ›**Sugarcane\_farmer** • a year ago

I have written the same code using stl. It is easier to understand that way

<http://ideone.com/jlmurQ>5   • Reply • Share ›**pango89** → Sugarcane\_farmer • 8 months ago

Thanks!

  • Reply • Share ›**Nilesh Solanki** • a year ago

Thank You Sir,

Your printCodes function in the code helped me complete my project....

  • Reply • Share ›**sas** • a year ago

superb work sir

  • Reply • Share ›**vishal** • 2 years ago

nice solution

  • Reply • Share ›**Ajay** • 2 years ago

In swapNodes() function

```
struct MinHeapNode* t = *a;
```

should be

```
struct MinHeapNode t = *a;
```

  • Reply • Share ›

**JV** • 2 years ago

why is this called as greedy algo.

In practical case do we need to maintain the table with symbol and code for decoding and encoding. In that case the huffman tree can be deleted once we get the codes for all the symbols

3 ^ | v • Reply • Share ›

**life coder** → JV • 4 months ago

Huffman's algorithm is an example of a greedy algorithm. It's called greedy because the two smallest nodes are chosen at each step, and this local decision results in a globally optimal encoding tree.

^ | v • Reply • Share ›

**its\_dark** • 2 years ago

how the huffman coding is a greedy algo??

```
/* Paste your code here (You may delete these lines if not writing code) */
```

2 ^ | v • Reply • Share ›

**Abhishek Choudhery** • 2 years ago

Also could you please explain, how is this GREEDY Algo?  
And if we could use DP here?

^ | v • Reply • Share ›

**Anand Dhandhan** → Abhishek Choudhery • 9 days ago

The algorithm which looks for an optimal solution in a way such that it keeps on following Optimal Sub-solutions is called Greedy Algorithm. If you will see all the sub-solutions to this approach are also optimal. Hence we say that it is a greedy approach.

While Greedy approach gives solution to these kinds of problems where optimal sub-solutions lead to optimal solution, they fail in many other cases.

^ | v • Reply • Share ›

**rajanarora** → Abhishek Choudhery • 10 months ago

It is a greedy because many trees of the given input is possible i.e solution space but we have to choose the optimal solution tree that will give the Optimal tree and that is itself greedy

^ | v • Reply • Share ›

**Abhishek Choudhery** • 2 years ago

Why have double pointers been used for swapMinHeapNode function?

^ | v • Reply • Share ›



**srinivas** • 2 years ago

very well explained and coded!

^ | v • Reply • Share ›



**a2** • 2 years ago

Awesome !

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site



Privacy

- 
- 
- 
- - [Interview Experiences](#)
  - [Advanced Data Structures](#)
  - [Dynamic Programming](#)
  - [Greedy Algorithms](#)
  - [Backtracking](#)
  - [Pattern Searching](#)
  - [Divide & Conquer](#)
  - [Mathematical Algorithms](#)
  - [Recursion](#)
  - [Geometric Algorithms](#)

## • Popular Posts

- [All permutations of a given string](#)
- [Memory Layout of C Programs](#)
- [Understanding “extern” keyword in C](#)
- [Median of two sorted arrays](#)
- [Tree traversal without recursion and without stack!](#)
- [Structure Member Alignment, Padding and Data Packing](#)
- [Intersection point of two Linked Lists](#)
- [Lowest Common Ancestor in a BST.](#)
- [Check if a binary tree is BST or not](#)
- [Sorted Linked List to Balanced BST](#)

Follow @GeeksforGeeks

## • Recent Comments

◦ [It\\_k](#)

i need help for coding this function in java...

[Java Programming Language](#) · [1 hour ago](#)

◦ [Piyush](#)

What is the purpose of else if (recStack[\*i])...

[Detect Cycle in a Directed Graph](#) · [1 hour ago](#)

◦ [Andy Toh](#)

My compile-time solution, which agrees with the...

[Dynamic Programming | Set 16 \(Floyd Warshall Algorithm\)](#) · [1 hour ago](#)

◦ [lucy](#)

because we first fill zero in first col and...

[Dynamic Programming | Set 29 \(Longest Common Substring\)](#) · [2 hours ago](#)

◦ [lucy](#)

@GeeksforGeeks i don't n know what is this long...

[Dynamic Programming | Set 28 \(Minimum insertions to form a palindrome\)](#) · [2 hours ago](#)

◦ [manish](#)

Because TAN is not a subsequence of RANT. ANT...

[Given two strings, find if first string is a subsequence of second](#) · [2 hours ago](#)

•  
@geeksforgeeks, [Some rights reserved](#) [Contact Us!](#)

Powered by [WordPress](#) & [MooTools](#), customized by geeksforgeeks team