Dr B R Ambedkar National Institute of Technology Jalandhar

CSPC-226 Design and Analysis of Algorithms

Project Report

Submitted By:

Dilraj Singh (20103051)

Harmandeep Singh (20103063)

Kirandeep Singh (20103080)

Submitted To:

Dr.Nonita Sharma

TITLE :

Develop a GUI implementation of Huffman Coding problem and Optimal Merge Pattern Problem in HTML and Javascript.
Named as DOSE(DAA On Steroids)

INTRODUCTION :

'DAA On Steroids'(hereby referred to as DOSE) is a tool to visualise and get explanation of Huffman Coding problem and Optimal Merge Pattern Problem using Greedy approach. It is developed to give clear understanding of concepts with step-by-step explanation and visualisation of Huffman Coding problem and Optimal Merge Pattern Problem algorithm.

This tool is based on getting the input from the user for Huffman Coding problem or Optimal Merge Pattern Problem and finally results in a step-by-step explanation for generation of respective trees. This also adds up step by step visualisation of working of Huffman Coding problem and Optimal Merge Pattern Problem algorithms.

FEATURES OF DOSE :

1. **Step-by-step Explanation :** DOSE is a tool that provides a clear step-by-step guide on user defined problems on Huffman Coding problem and Optimal Merge Pattern Problem.
2. **Tree and Forest visualisation :** DOSE also provides tree and forest visualisation to explain each and every step in a visual way.
3. **OOPs concepts in JavaScript :** Low level implementation is done in JavaScript providing most of the functions and other implementations to be done in a way to enable them to be reused.

Algorithm :

```
PROCEDURE generateExplanation
input values[]
input labels[]

let list.nodes = new CustNode[]

loop for each value and label pair
list.push(new CustNode(Type=”Leaf Node”, value=value, label=label))

list.nodes.sort(order=”Descending”)

write_Explanation_and_draw_forest_of(list.nodes)

loop till list.nodes.length() > 1
     nodeLeft  = list.nodes.pop()
     nodeRight = list.nodes.pop()

     newNode           = new CustNode(Type="Intermediate Node",
label="", value=nodeLeft.nodeValue + nodeRight.nodeValue);
     newNode.nodeLeft  = nodeLeft
     newNode.nodeRight = nodeRight

     write_Explanation_and_draw_tree_of(newNode)

     list.nodes.push(newNode)
     list.nodes.sort(order=”Descending”)

     write_Explanation_and_draw_forest_of(list.nodes)

create_final_table_from(list.nodes[1]) // list.nodes[1] <= Final tree
```

```
PROCEDURE write_Explanation_and_draw_forest_of(list_of_nodes)

loop for each node in list_of_nodes
     node.updateDepth()    // See the code to get algorithm

let inorderTraversal = new CustNode[]
loop for each node in list_of_nodes in reverse
     inorderTraversal.append(getInorderTraversal(node))

generate_SVG_to_draw_from(inorderTraversal)



PROCEDURE write_Explanation_and_draw_tree_of(root_node)

root_node.updateDepth()
let inorderTraversal = getInorderTraversal(root_node)

generate_SVG_to_draw_from(inorderTraversal)
```

```
PROCEDURE generate_SVG_to_draw_from(inorder_traversal)

//    +-------------------------> x-axis
//    |
//    |
//    |     COORDINATE SYSTEM
//    |            OF
//    |        SVG ELEMENT
//    |
//    |
//    v
//  y-axis

// x cood of node is its index in inorder_traversal
// y cood of node is its depth

loop for each node in inorder_traversal
     if node.type = "Intermediate Node"
          draw_line_from(node)to(node.leftChild)
          draw_line_from(node)to(node.rightChild)

loop for each node in inorder_traversal     // To draw nodes at one
level above the lines
    if node.type = "Intermediate Node"
        draw_circular_node_with_value(node.value)
    else
        draw_square_node_with_value(node.value)and_label(node.label)
```

CODE :

DOSE_main.html

```
<!DOCTYPE html>
<html style="border-style:solid; border-width:thick;border-color:coral">

<head>
	<link rel="stylesheet" href="DOSE_main.css">
</head>

<body style="font-family: Century Gothic; font-size: large; margin: 0px">
	<nav class="navbar">
		<a href="DOSE_main.html" class="navbar__logo">DOSE</a>
		<div class="navbar__toggle" id="mobile-menu">
		</div>
		<div class="navbar__menu">
			<a href="DOSE_main.html" class="navbar__link">Home</a>
		</div>
	</nav>
	<p>
		A SIMPLE TOOL FOR IMPLEMENTION AND CALCULATION OF HUFFMAN
CODING AND OPTIMAL MERGE PATTERN
	</p>
	<header>
		<div class="container">
			<div class="left">
				<img src="huffman.png">
			</div>
			<div class="right">
				<h1>HUFFMAN CODING</h1>
				<p> Huffman coding is a technique of compressing
data to minimize the size of the data without losing details.</p>
				<button onclick="document.location=
'DOSE_2.html?isHuff=1'" class="btn">&#8594</button>
			</div>
		</div>
	</header>
	<section id="optimal">
		<div class="leftside">
			<div class="lefts">
				<h2>OPTIMAL MERGE PATTERN</h2>
				<p>The technique by which minimum computations are
```

```
required to merge two or more files into a single sorted file is known
                              as Optimal merge pattern.
                        </p>
                        <button onclick="document.location= 'DOSE_2.html'"
class="btn2">&#8594</button>
                  </div>
                  <div class="rightside">
                        <img src="omp.png">
                  </div>
            </div>
      </section>
</body>

</html>
```

DOSE_2.html

```
<html><head>
    <script src="Dose.js"></script>
    <link rel="stylesheet" href="DOSE_2.css">
</head>
<body style="font-family: Century Gothic; font-size: large; padding-top:
70px" onload="setup()">
    <nav class="navbar" style="position: fixed; left: 0; right: 0; z-index:
1; top: 0;">
        <a href="DOSE_main.html" class="navbar__logo">DOSE</a>
        <div class="navbar__toggle" id="mobile-menu">
        </div>
        <div class="navbar__menu">
            <a href="DOSE_main.html" class="navbar__link">Home</a>
        </div>
    </nav>
    <div id="input" style="left: 0px; padding: 2px; display:
inline-block;width: 24%;height: 100%;position: fixed;border-right-style:
groove;">
        <h2>Enter the values</h2>
        <hr>
        <b>Enter the number of inputs :</b>
        <input id="noOfInputs" type="number" style="width:
-webkit-fill-available;">
        <input type="button" value="Generate List"
onclick="generateList()">
        <hr>
        <div id="inputList"></div>
    </div>
    <div id="output" style="right: 0px; position: absolute; width: 75%;">
        <h1>Enter the input and press run</h1>
    </div>
</body></html>
```

DOSE_main.css

```
header{
    background-color: coral;
    clip-path: polygon(0% 0%,100% 0%,100% 100%,80% 80%,0% 100%);
}
.navbar
{
    background-color: coral;
    height: 60px;
    padding: 0.5rem;
    display: flex;
    justify-content: space-between;
    align-items: center;
    position: relative;
    border-color: white;
    border-radius: 5px;
}
.navbar__logo{
    color: white;
    text-decoration: none;
    padding-left:  1rem;
    font-size: 1.5rem;
}
.navbar__link{
    color: white;
    text-decoration: none;
    padding: 0rem 1rem;
}

p
{
    color: rgb(9, 9, 9);
    font-weight: bold;
    text-align: center;
    font-size: 36;
    text-decoration: rgb(23, 176, 223);
}
.container{
    width: 100%;
    display: flex;
    justify-content: center;
    align-items:center;
```

```
}
.left{
    flex-basis: 50%;
}
img{
    width: 50%;
    display: flex;
    align-items: left;
    margin: auto;
    border-style:hidden;
    border-color: rgb(248, 246, 246);
    background-color: rgb(246, 250, 251);
    margin-top: 100px;
    margin-bottom: 100px;
    border-radius: 5px;
    cursor: hand;
}
.right{
    flex-basis:50%;
    max-width: 400px;
}
.right h1{
    color:red;
    font-size:24;
}
.right p {
    font-size: 20px;
    text-align: center;
    color: aliceblue;
}
.btn{
    background:transparent;
    padding: 10px 20px;
    color: rgb(244, 252, 250);
    border-radius: 500px;
    font-weight: bold;
    border-style:double;
}
.navbar__link:hover{
    text-decoration: underline;
}
.btn:hover{
```

```
    transform: scale(1.2);
}
#optimal{
    padding:100px;
}
.leftside{
    width: 100%;
    display:flex;
    justify-content: space-around;
    align-items: center;
}
.lefts{
    flex-basis: 50%;
}
.rightside{
    flex-basis: 50%;
}
.lefts h2{
    color:red;
    font-size: 24;
}
.lefts p{
    font-size: 20;
}
.btn2{
    background:transparent;
    padding: 10px 20px;
    color: rgb(4, 4, 4);
    border-radius: 500px;
    font-weight: bold;
    border-style:double;
}
.btn2:hover{
    transform: scale(1.2);
}
```

DOSE_2.css

```css
.navbar
{
    background-color:coral;
    height: 60px;
    padding: 0.5rem;
    display: flex;
    justify-content: space-between;
    align-items: center;
    position: relative;
    border-color: white;
    border-radius: 5px;
}
.navbar__logo{
    color: white;
    text-decoration: none;
    padding-left:  1rem;
    font-size: 1.5rem;
}
.navbar__link{
    color: white;
    text-decoration: none;
    padding: 0rem 1rem;
}

span.node_view_L {
    background-color: coral;
    padding: 5px;
    display: inline-block;
    width: 40px;
    height: 40px;
    text-align: center;
    font-family: monospace;
    border-width: 2px;
    border-color: black;
    border-radius: 10px;
    border-style: solid;
    margin: 10px;
    color: white;
    font-size: 15px;
}
```

```css
span.node_view_I {
    background-color: coral;
    padding: 5px;
    display: inline-block;
    width: 40px;
    height: 40px;
    text-align: center;
    font-family: monospace;
    font-size: 35px;
    border-width: 2px;
    border-color: black;
    border-radius: 100%;
    border-style: solid;
    margin: 10px;
    color: white;
}

.node_label {
    border-bottom-style: solid;
}

.node_list {
    border-radius: 10px;
    background-color: bisque;
    width: fit-content;
    margin: 10px;
    box-shadow: 2px 2px 6px 1px;
}
table {
    background: white;
    font-size: 20px;
    font-family: monospace;
    border-radius: 2px;
    margin: 5px;
    box-shadow: 2px 2px 6px 1px;
}

tr {
    background: bisque;
}

th {
    background: coral;
```

```
    color: white;
    padding: 10px;
}

td {
    padding: 4;
}

svg{
    border-radius: 10px;
    background-color: bisque;
    margin: 10px;
    box-shadow: 2px 2px 6px 1px;
}

input {
    border-style: solid;
    font-size: 16px;
    font-family: Century Gothic;
    border-radius: 3px;
    width: 45%;
}

input[type="button" i] {
    font-size: 16px;
    font-family: Century Gothic;
    color: white;
    border-radius: 8px;
    margin: 10px;
    width: -webkit-fill-available;
    background-color: coral;
    border-style: solid;
    padding: 5px;
}

.label {
    font-size: 16px;
    font-family: Century Gothic;
    color: white;
    border-radius: 3px;
    border-width: medium;
    border-color: gray;
    border-style: solid;
```

```
    margin: 1px;
    width: 43%;
    background-color: coral;
    border-style: solid;
    padding: 5px;
    display: inline-block;
}
```

Dose.js

```
/**### CustNode
 * This class represents a node of tree.
 *
 * Its structure is compatible with both ***Huffman Tree*** and Tree for
***Optimal Merge Pattern***.
 *
 * @author Kirandeep Singh
 */
class CustNode{

    /**Represents the type of node. Weather it is a **Intermediate node**
or **Leaf node**.
     *
     * Its values are :
     * 1. 'I' - Intermediate Node
     * 2. 'L' - Leaf Node
     *
     * @type {string}
     */
    nodeType

    /**Stores the label for node
     *
     * @type {string}
     */
    nodeLabel

    /**Holds integer value for the node to perform operations on
     *
     * @type {number}
     */
    nodeValue

    /**Holds reference to left child in tree
     *
     * @type {CustNode}
     */
    nodeLeft

    /**Holds reference to right child in tree
     *
```

```
     * @type {CustNode}
     */
    nodeRight

    /**Holds depth of node
     *
     * @type {number}
     */
    depth

    /**Holds the value of huffman code
     *
     * @type {string}
     */
    huffCode

    /**Constructor to initialise the Node
     *
     * @class Node
     *
     * @param {string}  nodeType  : Type of node
     * @param {string}  nodeData  : String value to represent the node
     * @param {number}  nodeValue : Integer value for node
     */
    constructor(nodeType, nodeLabel, nodeValue){
        this.nodeType  = nodeType
        this.nodeLabel = nodeLabel
        this.nodeValue = nodeValue

        this.nodeLeft  = null
        this.nodeRight = null
        this.depth     = 0
        this.huffCode  = ""
    }

    /**Comparing function to compare the two nodes.Good as an input to
`sort(?compareFn)` to `Node[]`
     *
     * @return {function(CustNode, CustNode):number} value of
comparison(i.e. `arg0.nodeValue - arg1.nodeValue`)
     */
    static getRevCompareFunction(){
        return function(arg0, arg1){return arg1.nodeValue - arg0.nodeValue}
```

```
    }

    /**Updates the depth
     *
     * @param {number} depth : Value of depth to be set
     */
    updateDepth(depth){
        this.depth = depth

        TreeUtil.maxDepth = (TreeUtil.maxDepth <
depth)?depth:TreeUtil.maxDepth

        if(this.nodeRight != null)
            this.nodeRight.updateDepth(this.depth + 1)

        if(this.nodeLeft != null)
            this.nodeLeft.updateDepth(this.depth + 1)
    }

    /**Converts the node to visualisable format of HTML. It generates a
code for `span` element of class name as `node_view_I` for nodes with node
type `'I'` and `node_view_L` for nodes with node type `'L'`.
     *
     * Example :
     * ```
     * <span class='node_view_I'>
     *      <p class='node_label'>Label to Node</p>
     *      <p class='node_value'>Value to Node</p>
     * </span>
     * ```
     * @return {string} HTML code to show
     */
    toHtml(){
        if(this.nodeType == "L")
            return '<span class=\'node_view_L\'><div class=\'node_label\'>'
+ this.nodeLabel + '</div><div class=\'node_value\'>' + this.nodeValue +
'</div></span>'
        else
            return '<span class=\'node_view_I\'><div class=\'node_value\'>'
+ this.nodeValue + '</div></span>'
    }
}
```

```
/**### List
 * This class represents the list of nodes and generate an HTML code to
display the nodes
 *
 * @author Kirandeep Singh
 */
class List{
    /**Stores nodes for a single array representation
     *
     * @type {CustNode[]}
     */
    nodes

    /**Constructor to initialise a list of nodes.
     */
    constructor(){
        this.nodes = [];
    }

    /**Converts the list of node to visualisable format of HTML.
     * It generates a code for `div` element of class name as `node_list`.
     *
     * Example :
     * ```
     * <div class='node_list'>
     *      <!-- Nodes Here (Reffer to CustNode.toHtml()) -->
     * </div>
     * ```
     *
     * @returns {string} HTML code to show
    */
    toHtml(){
        let HTML_code = '<div class=\'node_list\'>'

        for(let i = this.nodes.length - 1; i >= 0; i--){
            HTML_code += this.nodes[i].toHtml()
        }

        return HTML_code.concat('</div>')
    }
}

/**Utility definitions for tree
```

```
 *
 * ***Important : All varible declarations are static and serve an internal
perpose. Hence prevent to write(or update) the values of these
 * varibles. Function calls can be done with no issues.***
 */
class TreeUtil{

    /**Stores inorder traversal
     *
     * Important : Not to be used outside **(for internal use only)**
     *
     * @type {CustNode[]}
     */
    static inorderTraversal

    /**Stores content of svg element
     *
     * Important : Not to be used outside **(for internal use only)**
     *
     * @type {string}
     */
    static svg

    /**Stores scale for content of svg element
     *
     * Important : Not to be used outside **(for internal use only)**
     *
     * @type {number}
     */
    static scale

    /**Stores maximum depth of tree
     *
     * Important : Not to be used outside **(for internal use only)**
     *
     * @type {number}
     */
    static maxDepth

    /**Generates a inorder traversal of tree
     *
     * @param {CustNode} root : Root of tree
     *
```

```
     * @return {CustNode[]} Pre order traversal of tree
     */
    static genInorderTraversal(root){
        if(root == null){
            return
        } else {
            TreeUtil.genInorderTraversal(root.nodeLeft)
            TreeUtil.inorderTraversal.push(root)
            TreeUtil.genInorderTraversal(root.nodeRight)
        }
    }

    /**Set up for generation of SVG from tree
     *
     * @param {CustNode[]} inorderTraversal : Inorder traversal of tree
     */
    static genSVG(){
        let node
        for(let i = 0; i < TreeUtil.inorderTraversal.length; i++){
            node = TreeUtil.inorderTraversal[i]
            if(node.nodeType == "I"){
                TreeUtil.svg += "<line stroke=\"black\" x1=\"" + ((i +
1)*TreeUtil.scale) + "px\" y1=\"" + ((node.depth + 1) * TreeUtil.scale) +
"px\" x2=\"" + ((TreeUtil.inorderTraversal.indexOf(node.nodeLeft) + 1) *
TreeUtil.scale) + "px\" y2=\"" + ((node.depth + 2) * TreeUtil.scale) +
"px\" />"
                TreeUtil.svg += "<line stroke=\"black\" x1=\"" + ((i +
1)*TreeUtil.scale) + "px\" y1=\"" + ((node.depth + 1) * TreeUtil.scale) +
"px\" x2=\"" + ((TreeUtil.inorderTraversal.indexOf(node.nodeRight) + 1) *
TreeUtil.scale) + "px\" y2=\"" + ((node.depth + 2) * TreeUtil.scale) +
"px\" />"
            }
        }

        for(let i = 0; i < TreeUtil.inorderTraversal.length; i++){
            node = TreeUtil.inorderTraversal[i]
            console.log(i,((i + 1)*TreeUtil.scale),((node.depth + 1) *
TreeUtil.scale))

            if(node.nodeType == "L"){
                TreeUtil.svg += "<rect fill=\"coral\" stroke=\"black\"
stroke-width=\"0.5px\" x=\"" + (((i + 1)*TreeUtil.scale) - 5) + "px\" y=\""
+ (((node.depth + 1) * TreeUtil.scale) - 5) + "px\" width=\"10px\"
```

```
height=\"10px\" rx=\"2px\" ry=\"2px\"/>"
                TreeUtil.svg += "<text font-family=\"monospace\"
fill=\"white\" font-size=\"4px\" x=\"" + (((i + 1)*TreeUtil.scale) - 2) +
"px\" y=\"" + (((node.depth + 1) * TreeUtil.scale) - 1) + "px\">" +
node.nodeLabel + "</text>"
                TreeUtil.svg += "<text font-family=\"monospace\"
fill=\"white\" font-size=\"4px\" x=\"" + (((i + 1)*TreeUtil.scale) - 2) +
"px\" y=\"" + (((node.depth + 1) * TreeUtil.scale) + 4) + "px\">" +
node.nodeValue + "</text>"
                TreeUtil.svg += "<line stroke=\"white\"
stroke-width=\"0.5px\" x1=\"" + (((i + 1)*TreeUtil.scale) - 3) + "px\"
y1=\"" + (((node.depth + 1) * TreeUtil.scale)) + "px\" x2=\"" + (((i +
1)*TreeUtil.scale) + 3) + "px\" y2=\"" + (((node.depth + 1) *
TreeUtil.scale)) + "px\"/>"
            }else{
                TreeUtil.svg += "<circle fill=\"coral\" stroke=\"black\"
stroke-width=\"0.5px\" cx=\"" + ((i + 1)*TreeUtil.scale) + "px\" cy=\"" +
((node.depth + 1) * TreeUtil.scale) + "px\" r=\"5px\"/>"
                TreeUtil.svg += "<text font-family=\"monospace\"
fill=\"white\" font-size=\"6px\" x=\"" + (((i + 1)*TreeUtil.scale) - 2) +
"px\" y=\"" + (((node.depth + 1) * TreeUtil.scale) + 2) + "px\">" +
node.nodeValue + "</text>"
            }
        }
    }

    /**Constructs svg from tree
     *
     * @param {CustNode} root : Root of Tree
     */
    static drawTree(root){

        TreeUtil.maxDepth = 0;
        root.updateDepth(0)

        TreeUtil.inorderTraversal = []
        TreeUtil.genInorderTraversal(root)
        TreeUtil.svg = ""
        TreeUtil.scale = 20
        TreeUtil.genSVG()
    }

    /**Creates complete HTML code for svg element
```

```
     *
     * @return {string} HTML code for SVG element(ready ro be embedded)
     */
    static getSVG(){
        return "<svg id=\"forestView\" viewBox=\"0 0 " +
((TreeUtil.inorderTraversal.length + 2) * TreeUtil.scale) + " " +
((TreeUtil.maxDepth + 2) * TreeUtil.scale) + "\">" + TreeUtil.svg +
"</svg>"
    }

    /**Constructs svg from forest
     *
     * @param {CustNode[]} forest : Array of trees to draw
     */
    static drawForest(forest){
        TreeUtil.maxDepth = 0;
        for(let node of list.nodes)
            node.updateDepth(0)

        TreeUtil.inorderTraversal = []

        for(let node of list.nodes)
            TreeUtil.genInorderTraversal(node)

        TreeUtil.svg = ""
        TreeUtil.scale = 20
        TreeUtil.genSVG()
    }

    /**Finds the Huffman code for all leaf nodes
     *
     * @param {CustNode} node : Root node
     * @param {string?}  code : Huffman code of node entered
     */
    static findHuffman(node, code = ""){
        if(node.nodeType == "I"){
            TreeUtil.findHuffman(node.nodeLeft , code + "0")
            TreeUtil.findHuffman(node.nodeRight, code + "1")
        }else{
            node.huffCode = code
        }
    }
}
```

```
/**List of all the nodes in input for Huffman tree.
 *
 * Important : Not to be used outside **(for internal use only and will be
edited with executions)**. Use `TreeUtil.inorderTraversal` for read only.
 *
 * @type {List}
 */
var list

/**Stores HTML code Output for explanation along with final svg
 *
 * @type {string}
 */
var output

/**Generates the explanation for Huffman Tree or Optimal Merge Pattern tree
 *
 * @param {string[]}  labels : Labels for each node
 * @param {number[]}  values : Values for nodes
 *
 * @return {string}  HTML code for step by step tree construction ond
explanation
 */
function generateExplanation(labels, values, isHuff = false){

    list = new List()
    let newNode
    let nodeleft
    let nodeRight

    output = ""
    console.log(output)

    for(let i = 0; i < labels.length; i++){
        list.nodes.push(new CustNode('L', labels[i], values[i]))
    }

    list.nodes.sort(CustNode.getRevCompareFunction())

    output += "Arrange the given input in an array form."

////////////////////////////////////
```

```
    TreeUtil.drawForest(list)
    output += TreeUtil.getSVG()
//////////////////////
    while(list.nodes.length > 1){
        nodeLeft  = list.nodes.pop()
        nodeRight = list.nodes.pop()

        console.log("LeftNode : \n")
        console.log(nodeLeft)
        console.log("RightNode : \n");
        console.log(nodeRight)

        newNode           = new CustNode('I', '', nodeLeft.nodeValue +
nodeRight.nodeValue)
        newNode.nodeLeft  = nodeLeft
        newNode.nodeRight = nodeRight

        list.nodes.push(newNode)
        list.nodes.sort(CustNode.getRevCompareFunction())

        output += "Now pick two nodes with minimum values (i.e. " +
nodeLeft.toHtml() + " and " + nodeRight.toHtml() + ")<br>Find the sum of
its values (i.e. <b>" + nodeLeft.nodeValue + " + " + nodeRight.nodeValue +
" = " + newNode.nodeValue + "</b>) and put the sum in a new Intermediate
Node. Now the new node created is " + newNode.toHtml() + "<br>Make the two
nodes its child. Such that the node with smaller value(here " +
nodeLeft.toHtml() + ") is its left child and one with bigger value(here " +
nodeRight.toHtml() + ") is its right child.<br><br><br> Now our new node
is,<br>"

        TreeUtil.drawTree(newNode)
        output += TreeUtil.getSVG() + "<br><br><br>Finally push it into the
array and we get,"

        //////////////////////////
        TreeUtil.drawForest(list)
        output += TreeUtil.getSVG()
        //////////////////////////
    }

    output += "<br><hr><br/><h2>Final " + ((isHuff)?"Huffman":"Optimal
Merge Pattern") + " Tree</h2>"
    document.getElementById('output').innerHTML = output
```

```
    TreeUtil.drawTree(list.nodes[0])
    document.getElementById('output').innerHTML += TreeUtil.getSVG()

    // __T_A_B_L_E__
    TreeUtil.findHuffman(list.nodes[0])
    let resultTable = "<table><tr>" + (isHuff?
"<th>Character</th><th>Frequency</th><th>Huffman Code</th><th>Code
Length</th><th>Weighted Code Length</th>" : "<th>File
Name</th><th>Size</th><th>Path Length</th><th>Weighted Path Length</th>") +
"</tr>"

    let weighted_external_path_length = 0

    for(let node of TreeUtil.inorderTraversal){
        if(node.nodeType == "L")
            resultTable += "<tr><td>" + node.nodeLabel + "</td><td>" +
node.nodeValue + "</td>" + (isHuff? "<td>" +  node.huffCode + "</td>" : "")
+ "<td>" + node.huffCode.length + "</td><td>" + (node.huffCode.length *
node.nodeValue) + "</td></tr>"
            weighted_external_path_length += node.huffCode.length *
node.nodeValue
    }

    resultTable += "<tr><td colspan='" + (isHuff?4:3) + "'>Total Weighted
External " + (isHuff? "Code" : "Path") + " Length</td><td>" +
weighted_external_path_length + "</td></tr></table>"

    document.getElementById("output").innerHTML += "<br><hr><h2>Resulting
Data Table</h2>" + resultTable

}

var inputList
var noOfInputs
function generateList(){
    list = document.getElementById("inputList")
    noOfInputs = Number(document.getElementById("noOfInputs").value)

    list.innerHTML = "<div class=\"label\">" + (isHuff?"Character":"File
Name") + "</div><div class=\"label\">" + (isHuff?"Frequency":"Size") +
"</div>"
    for (let i = 0; i < noOfInputs; i++){
```

```
        list.innerHTML += "<input type=\"text\" id=\"label" + i + "\">
<input type=\"number\" id=\"value" + i + "\"><br>"
    }

    list.innerHTML += "<input type=\"button\" value=\"Explain\"
onclick=\"explain()\"></div>"

}

/**Variable to store weather the screen is for hasse or Not
 *
 * @type {boolean}
 */
var isHuff

/**Initiate every requirement */
function setup(){
    isHuff = Boolean(new
URLSearchParams(window.location.search).get("isHuff"))
}

function explain(){
    let labels = []
    let values = []

    for (let i = 0; i < noOfInputs; i++){
        labels.push(document.getElementById("label" + i).value)
        values.push(Number(document.getElementById("value" + i).value))

        generateExplanation(labels, values, isHuff)
    }
}
```

OUTPUT

DOSE

Home

A SIMPLE TOOL FOR IMPLEMENTION AND CALCULATION OF HUFFMAN CODING AND OPTIMAL MERGE PATTERN

```
1101010101011001
010[HUFFM^N]101001
01010[C0d!NG]11010
1010101101001010
```

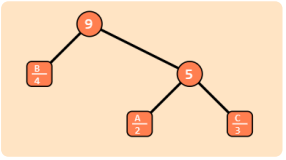
HUFFMAN CODING

Huffman coding is a technique of compressing data to minimize the size of the data without losing details.

→

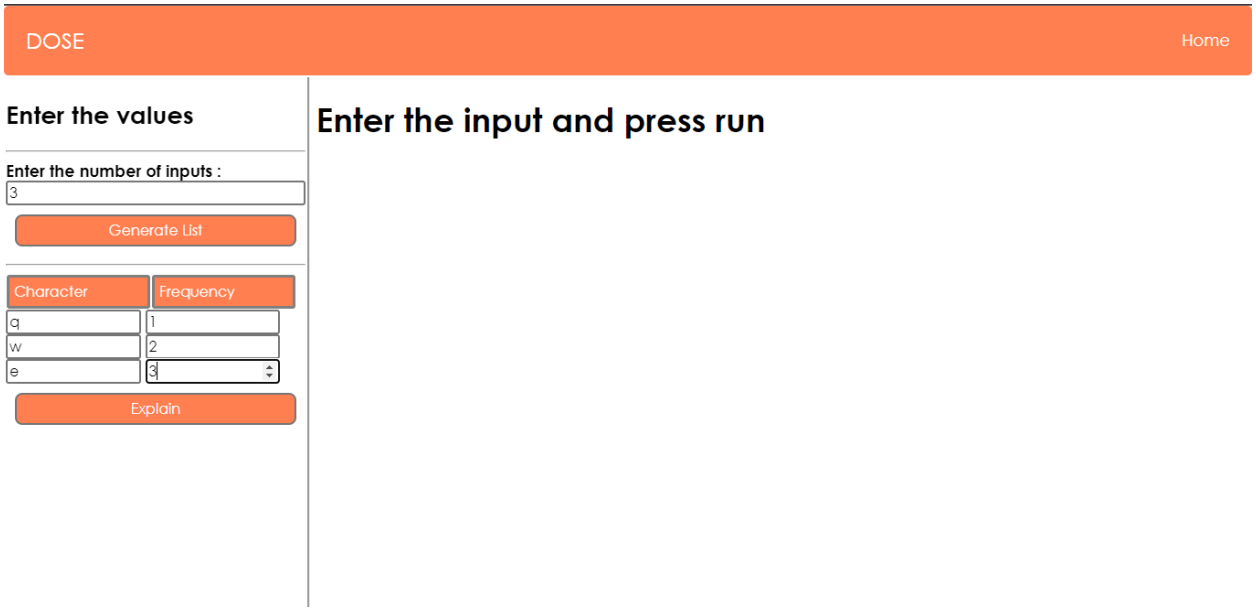
OPTIMAL MERGE PATTERN

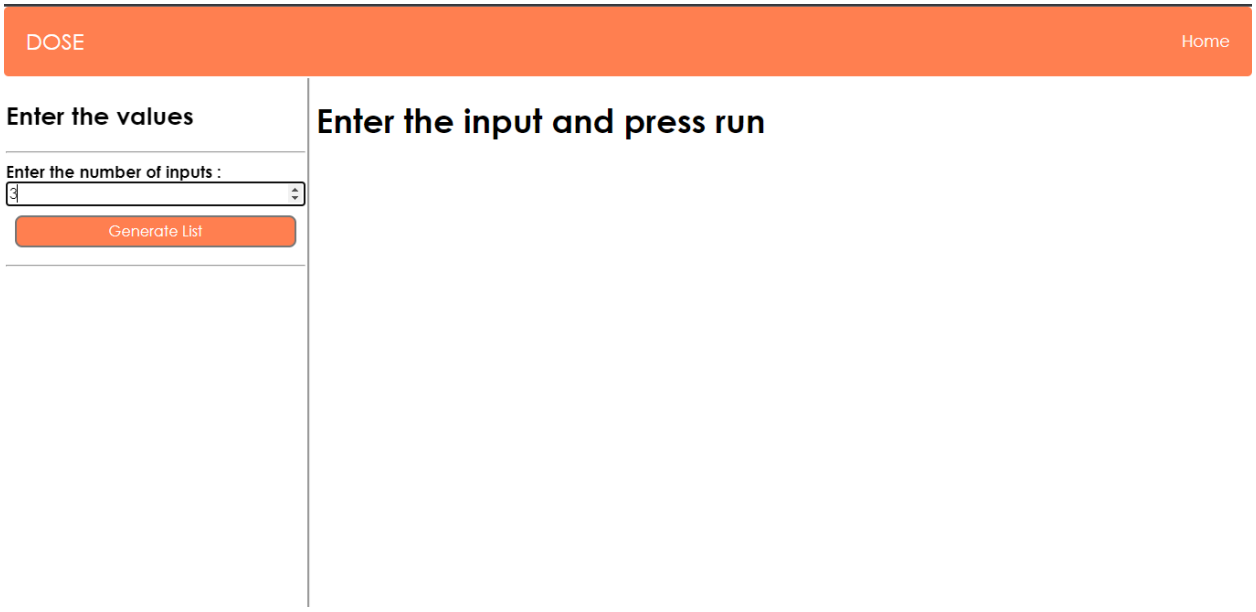
The technique by which minimum computations are required to merge two or more files into a single sorted file is known as Optimal merge pattern.



→

Take Input


DOSE
Home
Enter the values
Enter the number of inputs :
3
Generate List
Character
Frequency
q
1
w
2
e
3
Explain
Enter the input and press run


DOSE
Home
Enter the values
Enter the number of inputs :
3
Generate List
Enter the input and press run

Enter the values

Enter the number of inputs :

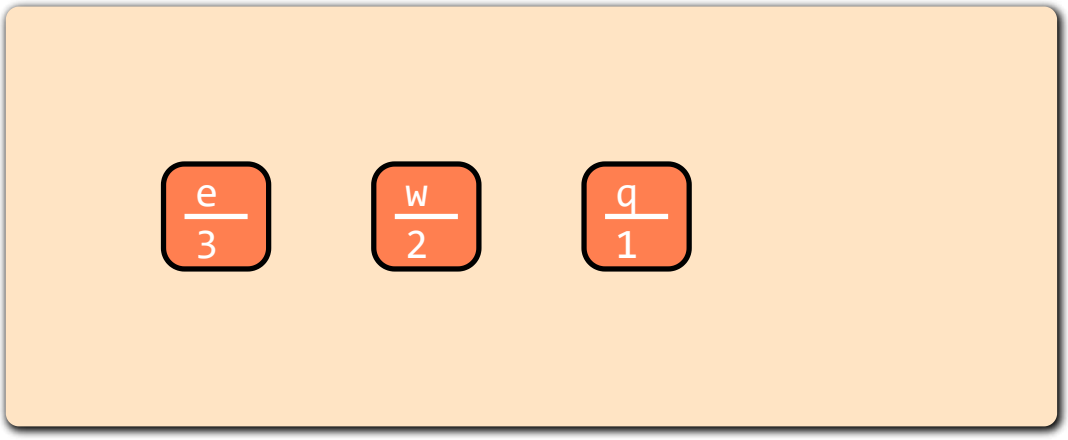
3

Generate List

Character	Frequency
q	1
w	2
e	3

Explain

Arrange the given input in an array form.

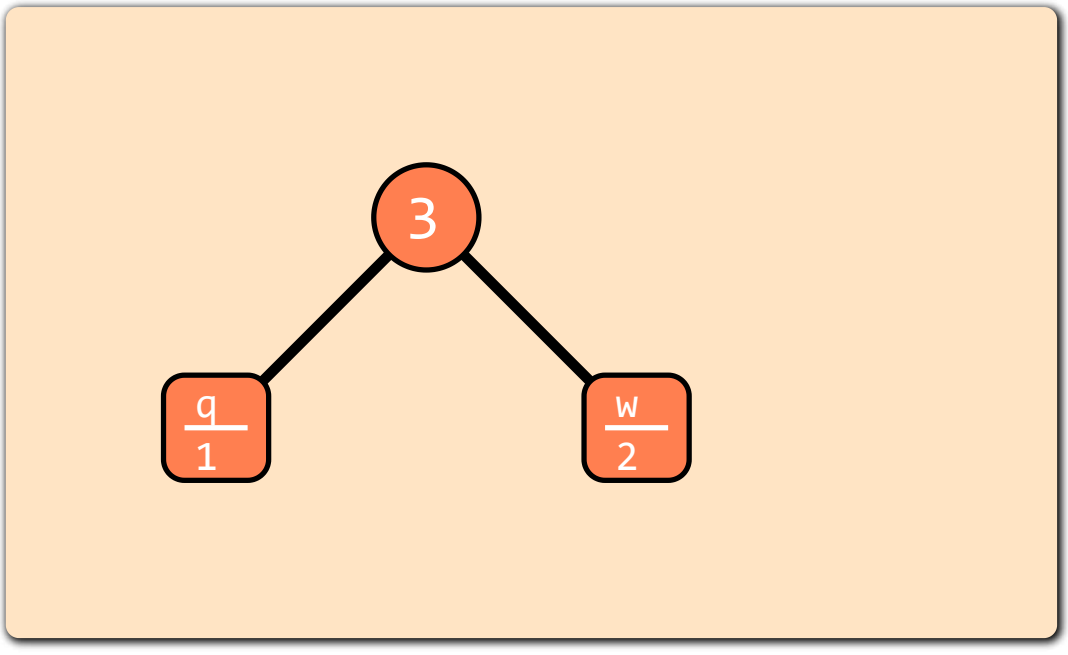


Now pick two nodes with minimum values (i.e. q 1 and w 2)

Find the sum of its values (i.e. **1 + 2 = 3**) and put the sum in a new Intermediate Node. Now the new node created is 3

Make the two nodes its child. Such that the node with smaller value(here q 1) is its left child and one with bigger value(here w 2) is its right child.

Now our new node is,



Finally push it into the array and we get,

Enter the values

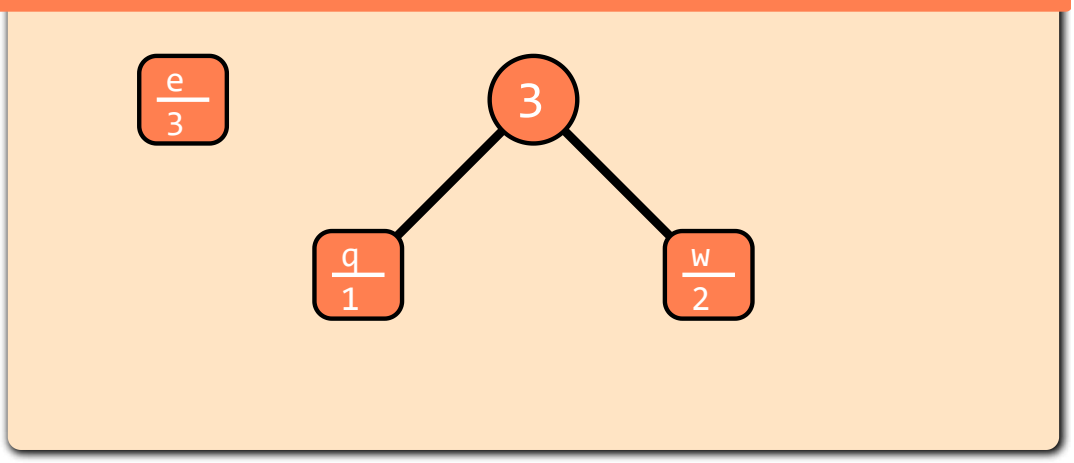
Enter the number of inputs :

3

Generate List

Character	Frequency
q	1
w	2
e	3

Explain

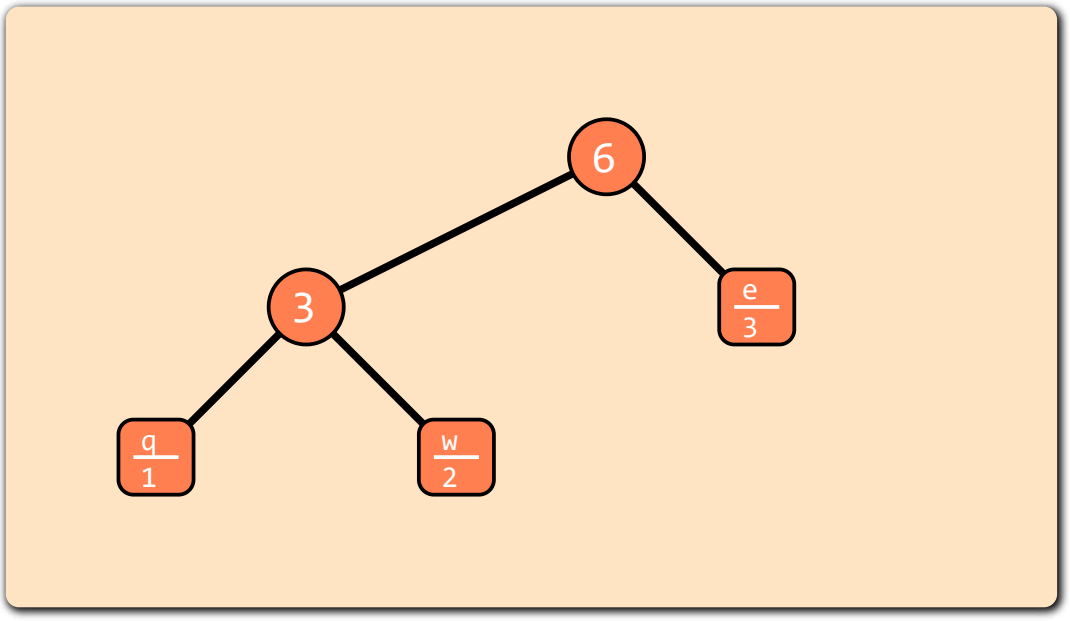


Now pick two nodes with minimum values (i.e. 3 and e/3)

Find the sum of its values (i.e. **3 + 3 = 6**) and put the sum in a new Intermediate Node. Now the new node created is 6

Make the two nodes its child. Such that the node with smaller value(here 3) is its left child and one with bigger value(here e/3) is its right child.

Now our new node is,



Enter the values

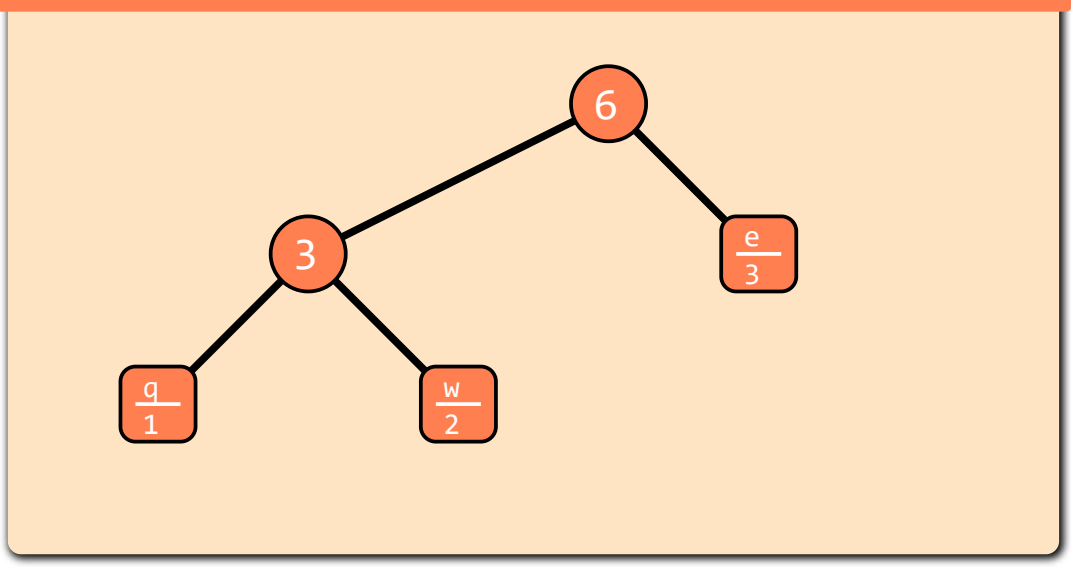
Enter the number of inputs :

3

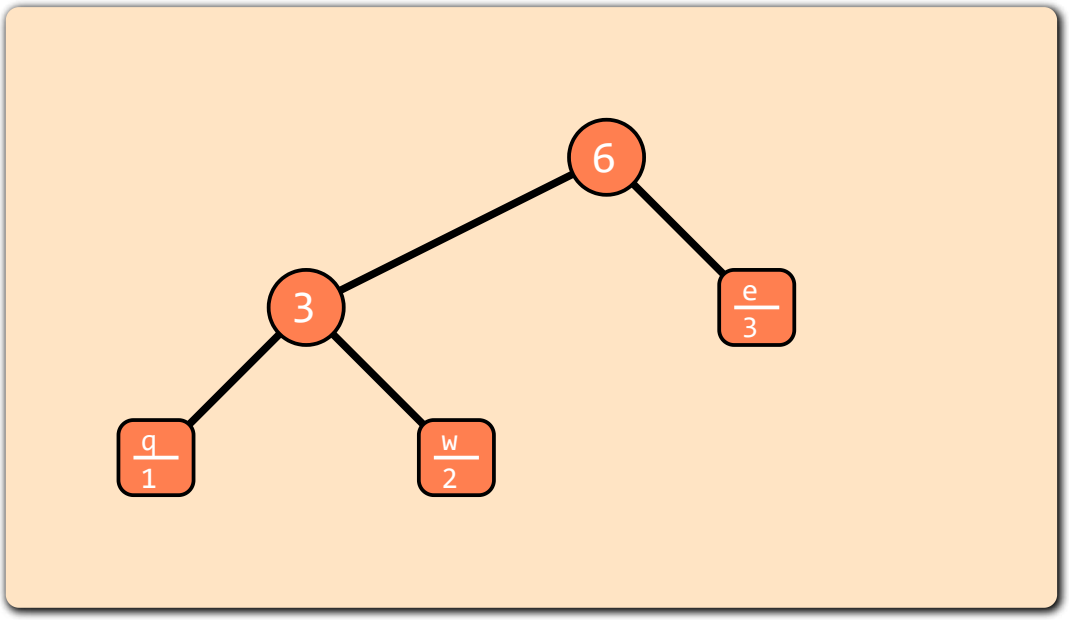
Generate List

Character	Frequency
q	1
w	2
e	3

Explain



Final Huffman Tree



Resulting Data Table

Character	Frequency	Huffman Code	Code Length	Weighted Code Length
q	1	00	2	2
w	2	01	2	4
e	3	1	1	3
Total Weighted External Code Length				9

Enter the values

Enter the number of inputs :

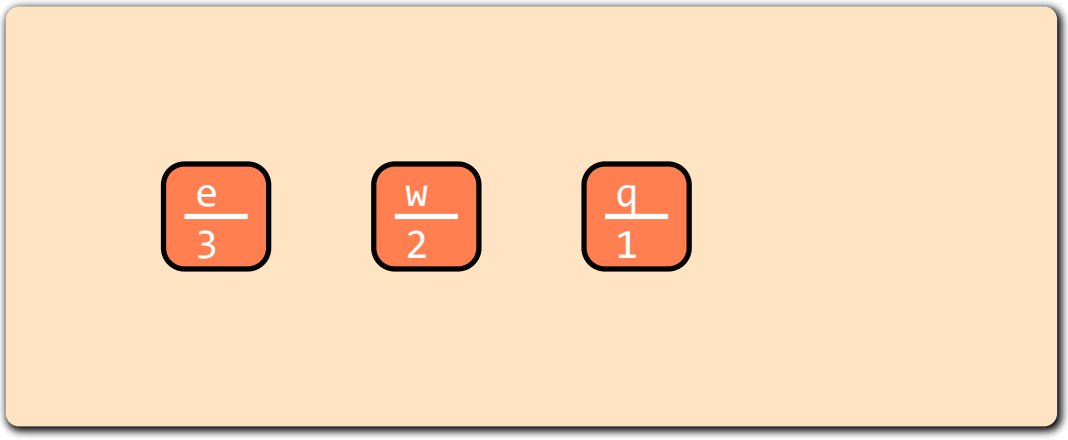
3

Generate List

File Name	Size
q	1
w	2
e	3

Explain

Arrange the given input in an array form.

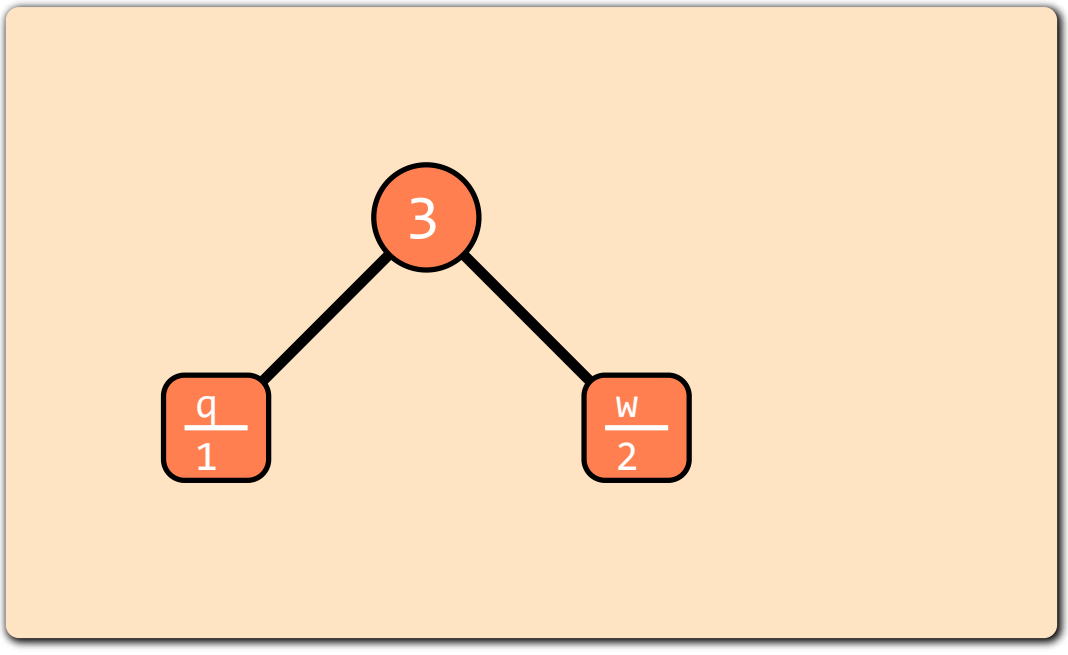


Now pick two nodes with minimum values (i.e. q/1 and w/2)

Find the sum of its values (i.e. **1 + 2 = 3**) and put the sum in a new Intermediate Node. Now the new node created is 3

Make the two nodes its child. Such that the node with smaller value(here q/1) is its left child and one with bigger value(here w/2) is its right child.

Now our new node is,



Finally push it into the array and we get,

Enter the values

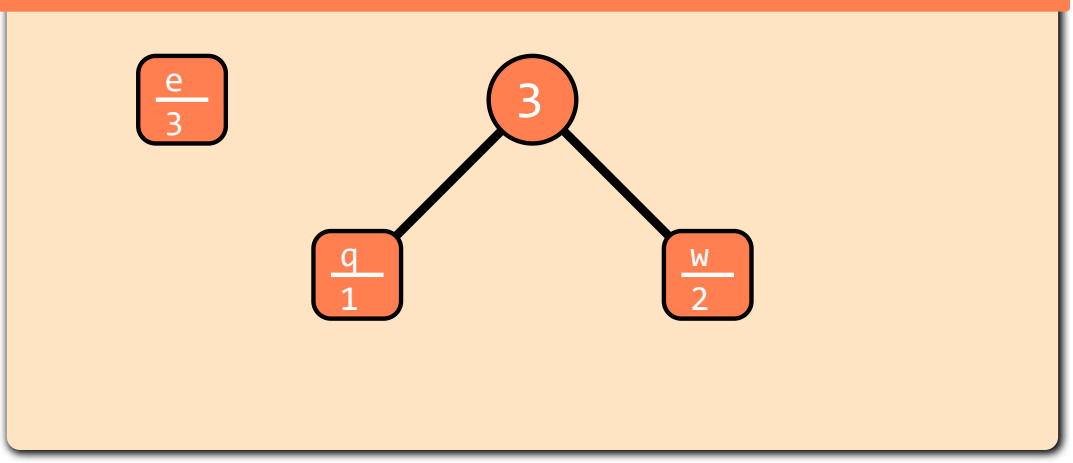
Enter the number of inputs :

3

Generate List

File Name	Size
q	1
w	2
e	3

Explain

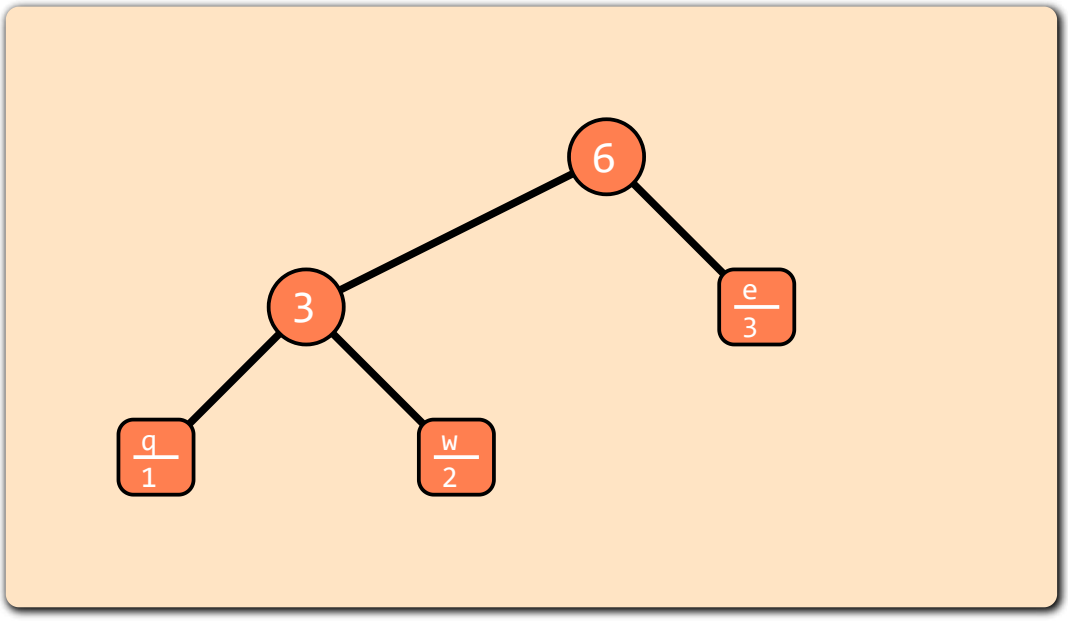


Now pick two nodes with minimum values (i.e. 3 and e/3)

Find the sum of its values (i.e. **3 + 3 = 6**) and put the sum in a new Intermediate Node. Now the new node created is 6

Make the two nodes its child. Such that the node with smaller value(here 3) is its left child and one with bigger value(here e/3) is its right child.

Now our new node is,



Enter the values

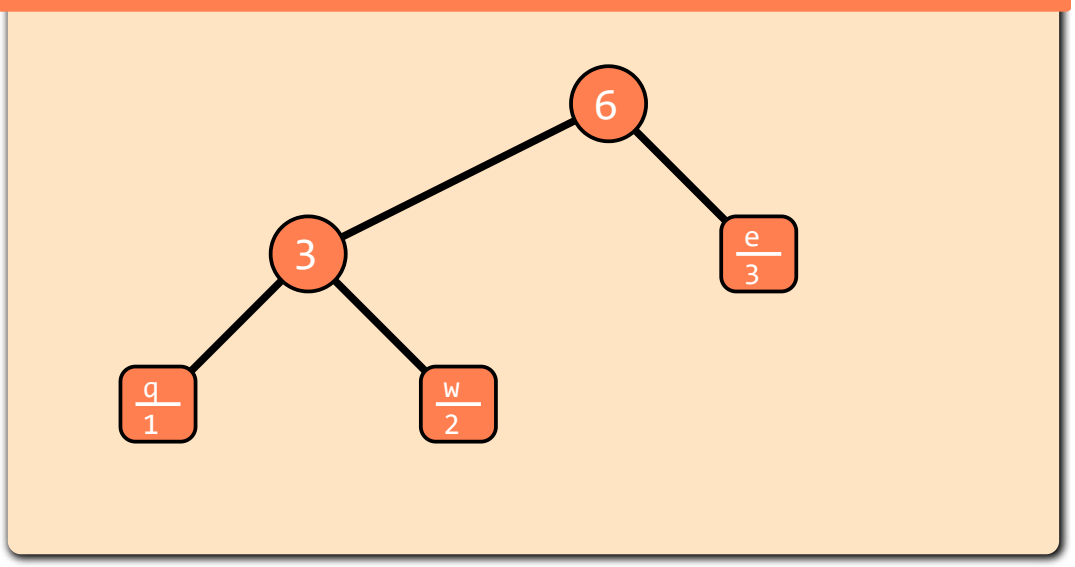
Enter the number of inputs :

3

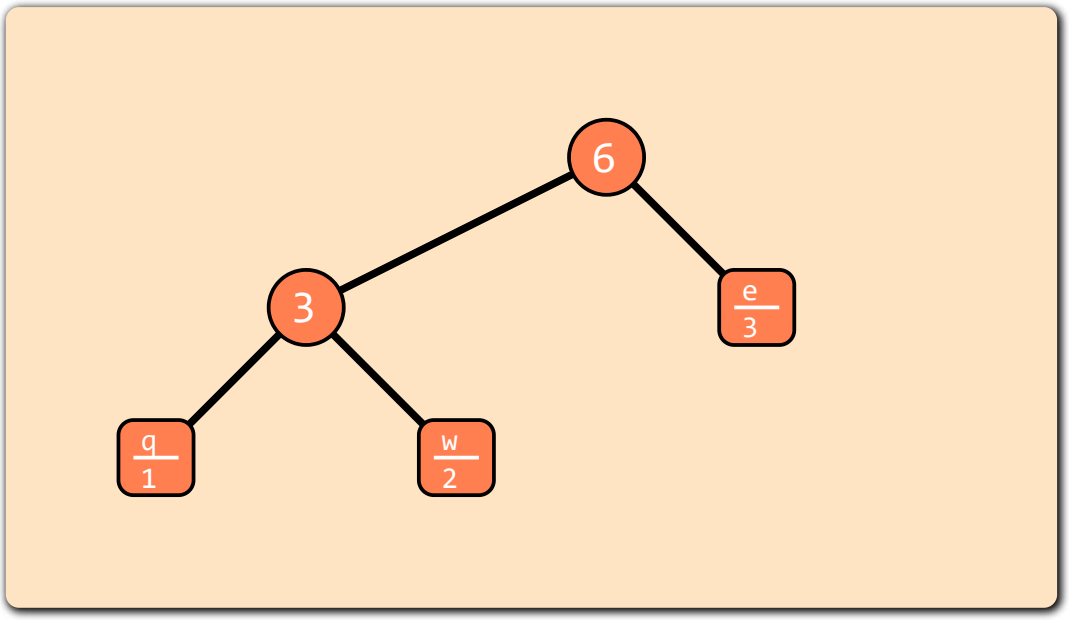
Generate List

File Name	Size
q	1
w	2
e	3

Explain



Final Optimal Merge Pattern Tree



Resulting Data Table

File Name	Size	Path Length	Weighted Path Length
q	1	2	2
w	2	2	4
e	3	1	3
Total Weighted External Path Length			9