DS PROJECT

HUFFMAN CODING

SAMYUKTHA E
MEGANATH V
KAVIDHARSANI K S

INTRODUCTION:

- Huffman Coding is a widely-used algorithm for data compression, especially effective in reducing the size of text-based data. The core idea behind Huffman Coding is to assign shorter binary codes to more frequent characters and longer codes to less frequent characters, thereby minimizing the overall storage size.
- This tool allows users to easily upload text, compress them, and see the size.
- By efficiently compressing files, it compress the text it consumes less storage.

TECHNICAL ASPECTS

- HTML:

  HTML provides the structure and content of the web page.

- CSS:

  CSS is used to style and visually enhance the HTML content.

- JavaScript:

  JavaScript enables interactive elements and dynamic behavior on the webpage.It handles client-side logic, such as sending file data to the server

HUFFMAN CODING TREE

The Huffman coding tree is a binary tree used in Huffman coding to assign variable-length codes to characters based on their frequencies. It helps in compressing data by using shorter codes for more frequent characters and longer codes for less frequent characters.

HTML CODE

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible"
content="IE=edge">
<meta name="viewport"
content="width=device-width, initial-
scale=1.0">
<link rel="stylesheet" href="styles.css">
<title>Huffman Coding Visualization</title>
</head>
<body>
<div class="container">
<h1>Huffman Coding Visualization</h1>
<input type="text" id="inputText" placeholder=
"Enter text here">
<button onclick="generateHuffmanCodes()">
Generate Huffman Codes</button>
<div id="huffmanCodes" class="output-box">
</div>
<div id="huffmanTree" class="output-box">
</div>
</div>
<script src="script.js"></script>
</body>
</html>
```

CSS CODE

```
body {
font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
background-color: #e0f7fa;
margin: 0;
padding: 0;
display: flex;
justify-content: center;
align-items: center;
height: 100%;}
.container {
background-color: #ffffff;
border-radius: 12px;
padding: 30px;
box-shadow: 0 8px 16px rgba(0, 0, 0, 0.1);
text-align: center;
width: 90%;
max-width: 1000px;
transition:transform 0.3s, box-shadow 0.3s;}
input[type="text"] {
padding: 12px;
margin-top: 20px;
margin-bottom: 15px;
width: calc(100% - 24px);
border: 2px solid #4caf50;
border-radius: 5px;
font-size: 16px;
outline: none;
transition: border-color 0.3s;
}
button {
padding: 12px 24px;
margin-top: 15px;
background-color: #4caf50;
color: white;
border: none;
border-radius: 25px;
cursor: pointer;
font-size: 16px;
transition:background-color 0.3s,transform 0.3s;}
button:hover {
background-color: #388e3c;
transform: translateY(-3px);}
.output-box {
margin-top: 20px;
text-align: left;
font-family:'Courier New',Courier,monospace;
background-color: #e8f5e9;
padding: 15px;
border-radius: 8px;
box-shadow: 0 4px 8px rgba(0, 0, 0, 0.05);
max-height: 300px;
overflow-y: auto;
white-space: pre;}
.node {
padding: 8px;
border: 2px solid #4caf50;
border-radius: 8px;
display: inline-block;
margin: 10px;
background-color: #a5d6a7;
font-weight: bold;
text-align: center;}
.children {
display: flex;
justify-content: center;
align-items: center;
margin-top: 10px;}
```

JAVASCRIPT CODE

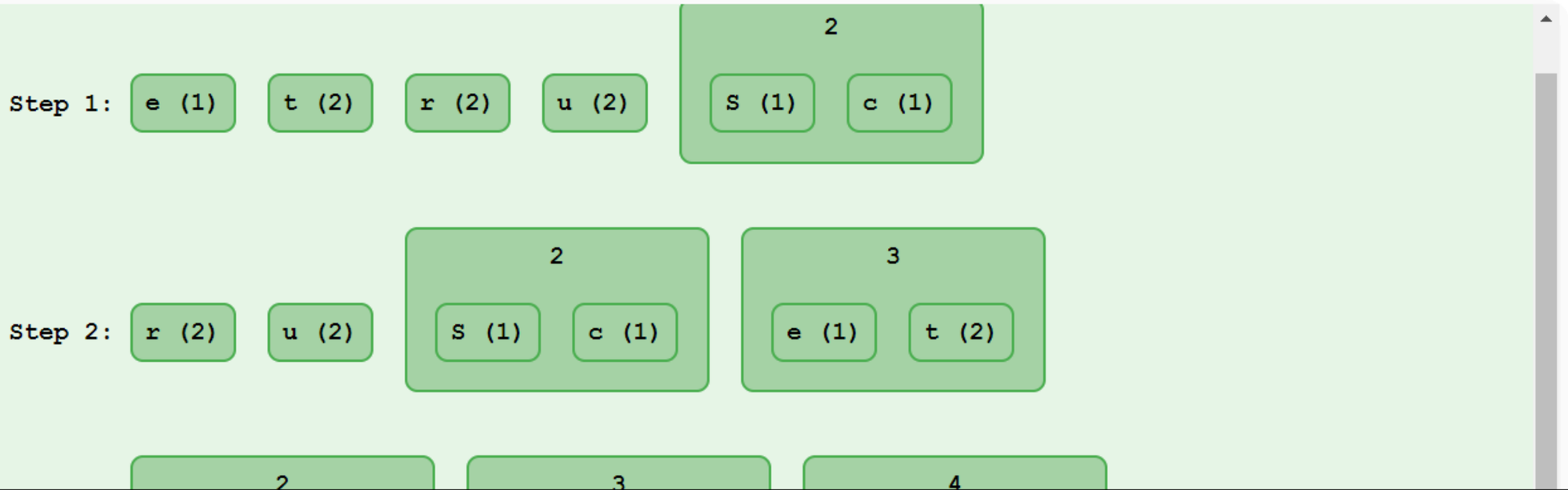
```
function createNode(data, freq) {
return { data, freq, left: null, right: null };}
function buildHuffmanTree(data, freq) {
const nodes = [];
for (let i = 0; i < data.length; i++) {
nodes.push(createNode(data[i], freq[i]));}
const steps = [];
while (nodes.length > 1) {
nodes.sort((a, b) => a.freq - b.freq);
const left = nodes.shift();
const right = nodes.shift();
const newNode = createNode('$', left.freq
+ right.freq);
newNode.left = left;
newNode.right = right;
nodes.push(newNode);
steps.push(JSON.parse(JSON.stringify(no
des))); }
return { root: nodes[0], steps };}
function displayHuffmanCodes(node,
code, result) {
f (!node) return;
if (!node.left && !node.right) {
result.push(`${node.data}:
${code.join('')}`);   return; }
code.push(0);
displayHuffmanCodes(node.left, code,
result);
code.pop();
code.push(1);
displayHuffmanCodes(node.right, code,
result);
code.pop(); }
function displayTreeSteps(steps) {
const treeElement =
document.getElementById('huffmanTree');
treeElement.innerHTML = '';
steps.forEach((step, index) => {
const stepDiv =
document.createElement('div');
stepDiv.style.marginBottom = '20px';
stepDiv.innerHTML = `<strong>Step ${index
+ 1}:</strong>`;
step.forEach(node => {
stepDiv.appendChild(drawNode(node)); });
treeElement.appendChild(stepDiv);});}
function drawNode(node) {
const nodeDiv =
document.createElement('div');
nodeDiv.className = 'node';
nodeDiv.innerHTML = node.data === '$' ?
node.freq : node.data + ' (' + node.freq + ')';
if (node.left || node.right) {
const childrenDiv =
document.createElement('div');
childrenDiv.className = 'children';
if (node.left) {
const leftChild = drawNode(node.left);
childrenDiv.appendChild(leftChild);}
if (node.right) {
const rightChild = drawNode(node.right);
childrenDiv.appendChild(rightChild);}
nodeDiv.appendChild(childrenDiv);}
return nodeDiv;}
function generateHuffmanCodes() {
constinput=document.getElementById('inpu
Text').value.trim();
if (!input) {
alert('Please enter some text');
return;  }
const freqMap = {};
for (let char of input) {
freqMap[char] = (freqMap[char] || 0) + 1;  }
const data = Object.keys(freqMap);
const freq = Object.values(freqMap);
const { root, steps } = buildHuffmanTree(dat
, freq)
displayTreeSteps(steps);
const result = [];
displayHuffmanCodes(root, [], result);
document.getElementById('huffmanCodes').
innerHTML = result.join('<br>');}
```

Huffman Coding Visualization

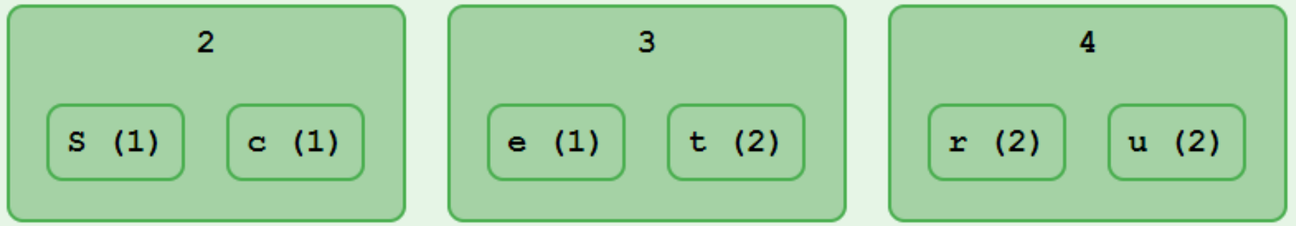
Structure

Generate Huffman Codes

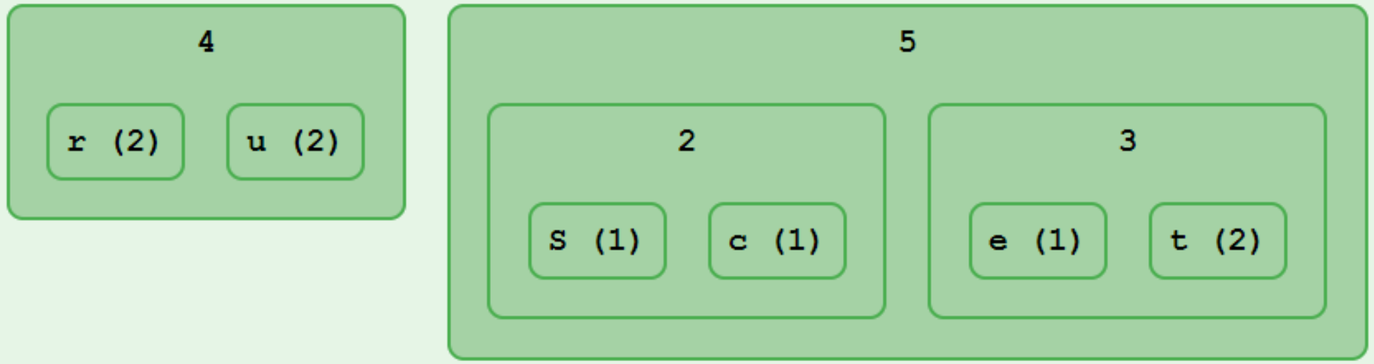
```
r: 00
u: 01
S: 100
c: 101
e: 110
t: 111
```



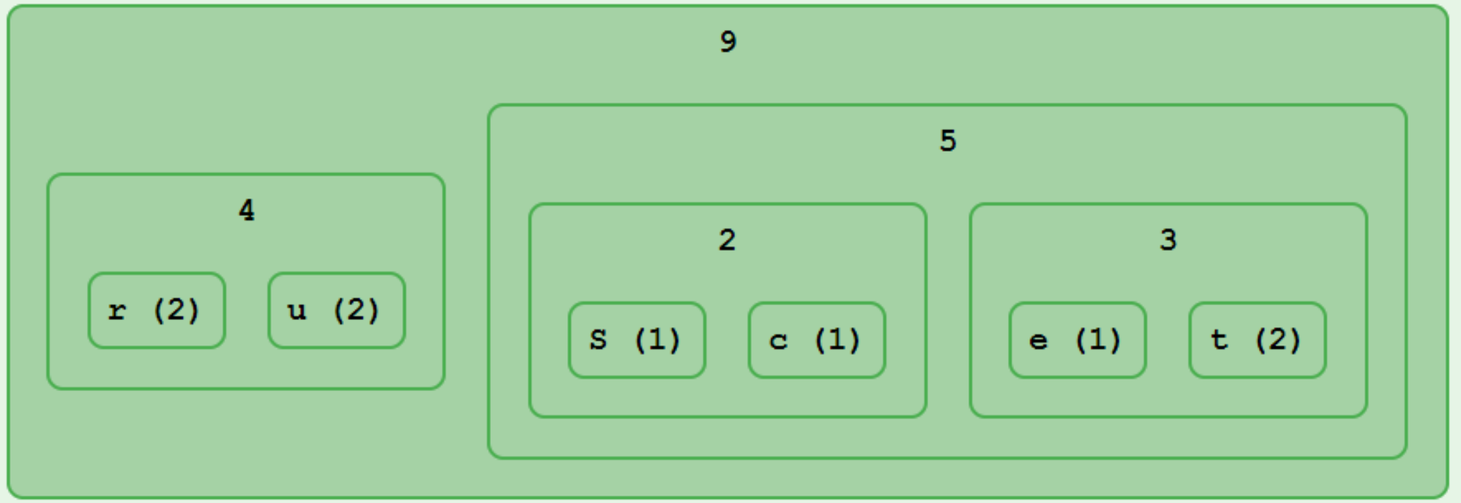
Step 3:



Step 4:



Step 5:



THANK YOU VERY MUCH!