DATA COMPRESSION AND DECOMPRESSION USING HUFFMANN CODING ALGORITHM

Mentor : Dr. Suyash Bhardawaj

Why Data Compression?

- Text files can be compressed to make them smaller and faster to send, and unzipping files on devices has a low overhead.
- We use the Huffman Coding algorithm for this purpose which is a greedy algorithm that assigns variable length binary codes for each input character in the text file. The length of the binary code depends on the frequency of the character in the file.

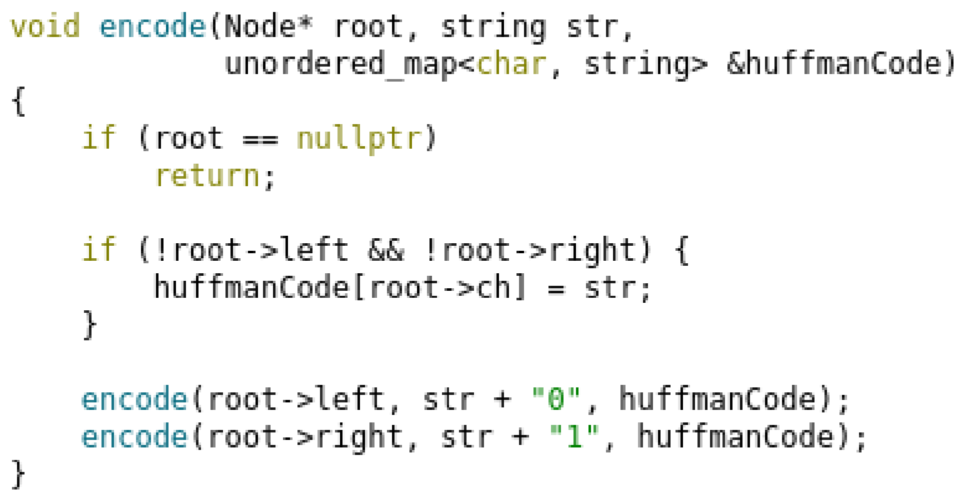
How Does The Process Of Compression Work?

- **The size of the text file can be reduced by compressing it, which converts the text to a smaller format that takes up less space. It typically works by locating similar strings/characters within a text file and replacing them with a temporary binary representation to reduce the overall file size.**

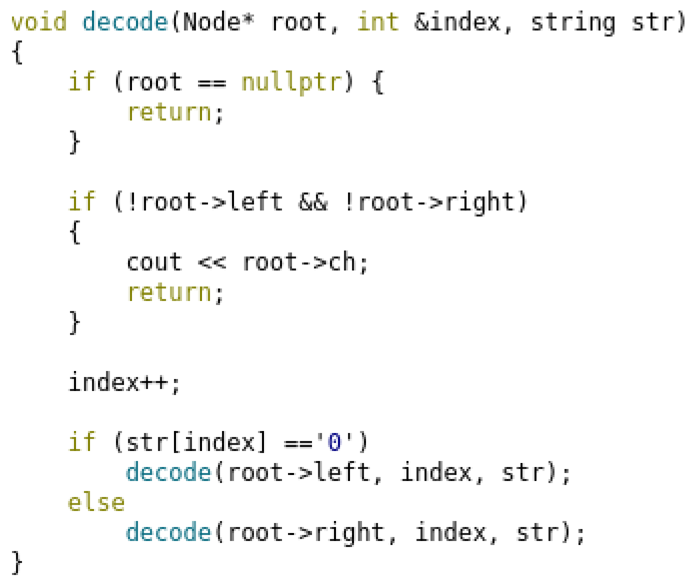
Tech Stack Used:

- C++
- React
- Node Js
- CSS

Cpp Implemetation Snapshots :

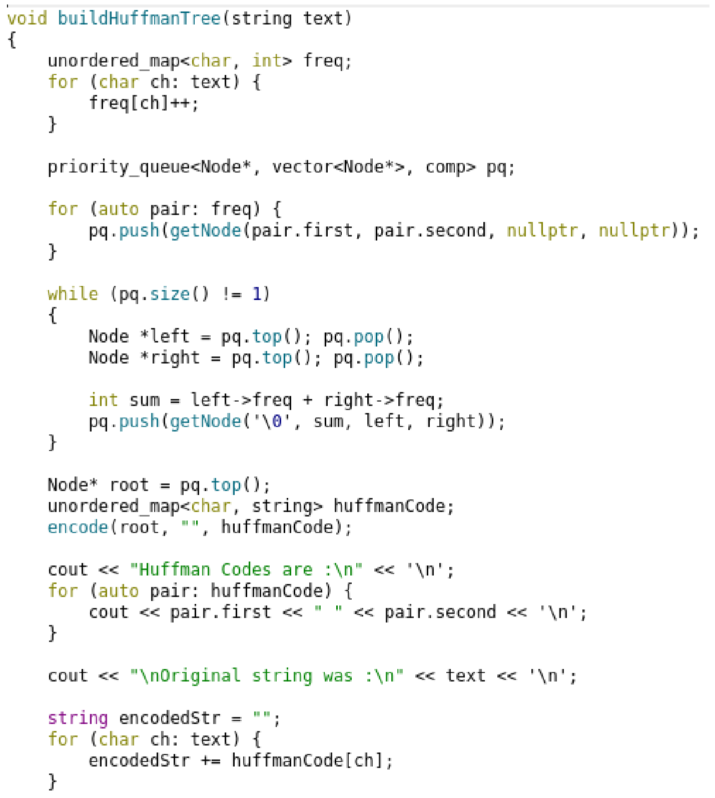


```cpp
void encode(Node* root, string str,
            unordered_map<char, string> &huffmanCode)
{
    if (root == nullptr)
        return;

    if (!root->left && !root->right) {
        huffmanCode[root->ch] = str;
    }

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



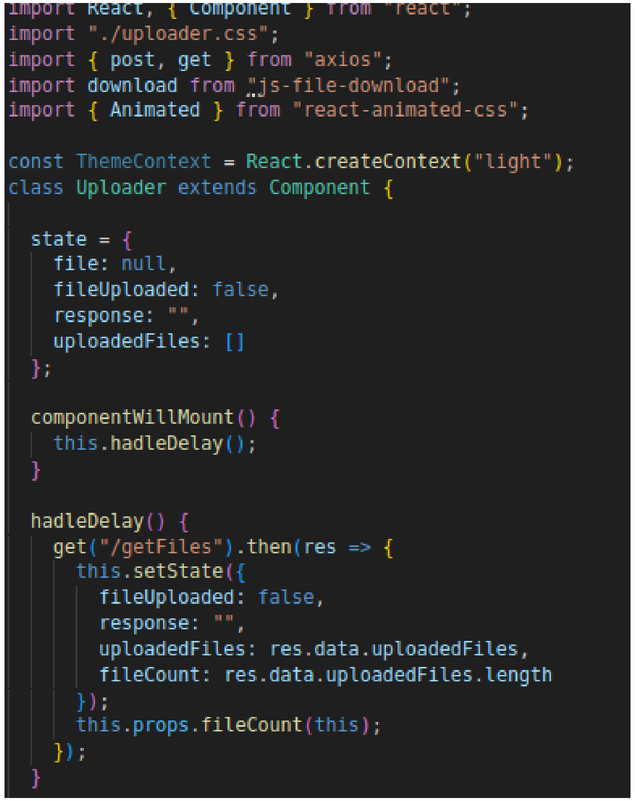
```cpp
void decode(Node* root, int &index, string str)
{
    if (root == nullptr) {
        return;
    }

    if (!root->left && !root->right)
    {
        cout << root->ch;
        return;
    }

    index++;

    if (str[index] =='0')
        decode(root->left, index, str);
    else
        decode(root->right, index, str);
}
```

Cpp Implementation Snapshots:

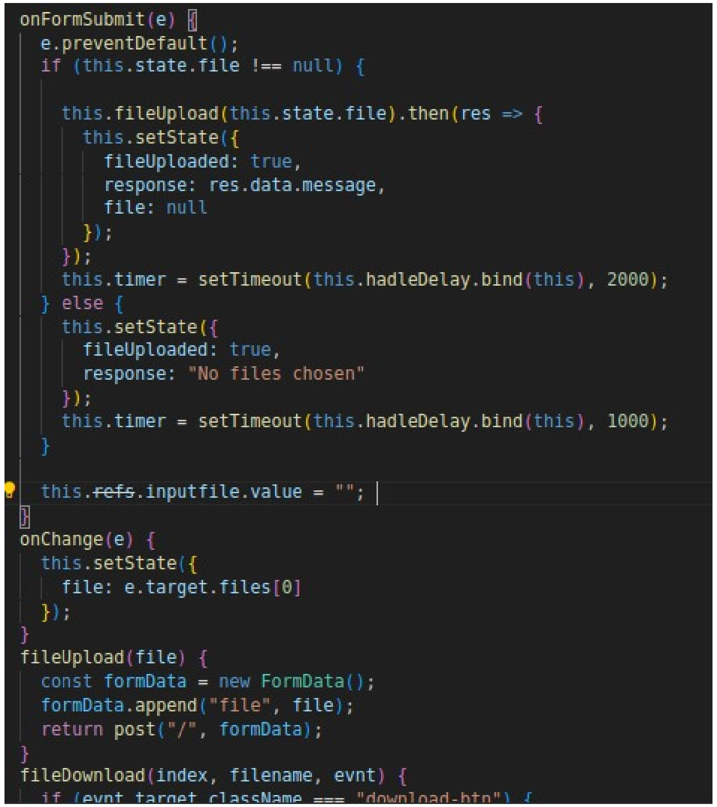


```cpp
void buildHuffmanTree(string text)
{
    unordered_map<char, int> freq;
    for (char ch: text) {
        freq[ch]++;
    }

    priority_queue<Node*, vector<Node*>, comp> pq;

    for (auto pair: freq) {
        pq.push(getNode(pair.first, pair.second, nullptr, nullptr));
    }

    while (pq.size() != 1)
    {
        Node *left = pq.top(); pq.pop();
        Node *right = pq.top(); pq.pop();

        int sum = left->freq + right->freq;
        pq.push(getNode('\0', sum, left, right));
    }

    Node* root = pq.top();
    unordered_map<char, string> huffmanCode;
    encode(root, "", huffmanCode);

    cout << "Huffman Codes are :\n" << '\n';
    for (auto pair: huffmanCode) {
        cout << pair.first << " " << pair.second << '\n';
    }

    cout << "\nOriginal string was :\n" << text << '\n';

    string encodedStr = "";
    for (char ch: text) {
        encodedStr += huffmanCode[ch];
    }
```

Frontend Code Snapshots :

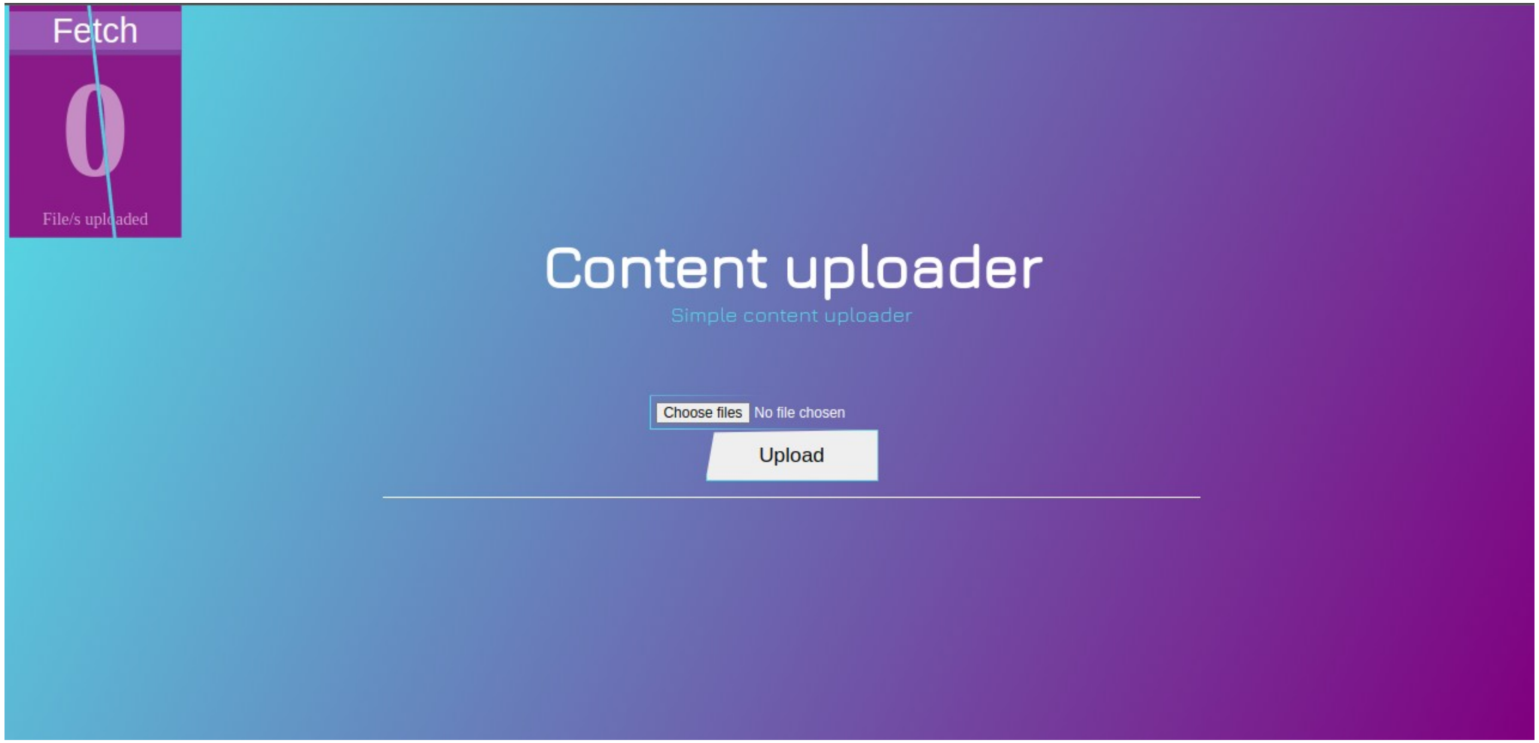


```
import React, { Component } from "react";
import "./uploader.css";
import { post, get } from "axios";
import download from "js-file-download";
import { Animated } from "react-animated-css";

const ThemeContext = React.createContext("light");
class Uploader extends Component {

  state = {
    file: null,
    fileUploaded: false,
    response: "",
    uploadedFiles: []
  };

  componentWillMount() {
    this.hadleDelay();
  }

  hadleDelay() {
    get("/getFiles").then(res => {
      this.setState({
        fileUploaded: false,
        response: "",
        uploadedFiles: res.data.uploadedFiles,
        fileCount: res.data.uploadedFiles.length
      });
      this.props.fileCount(this);
    });
  }
```

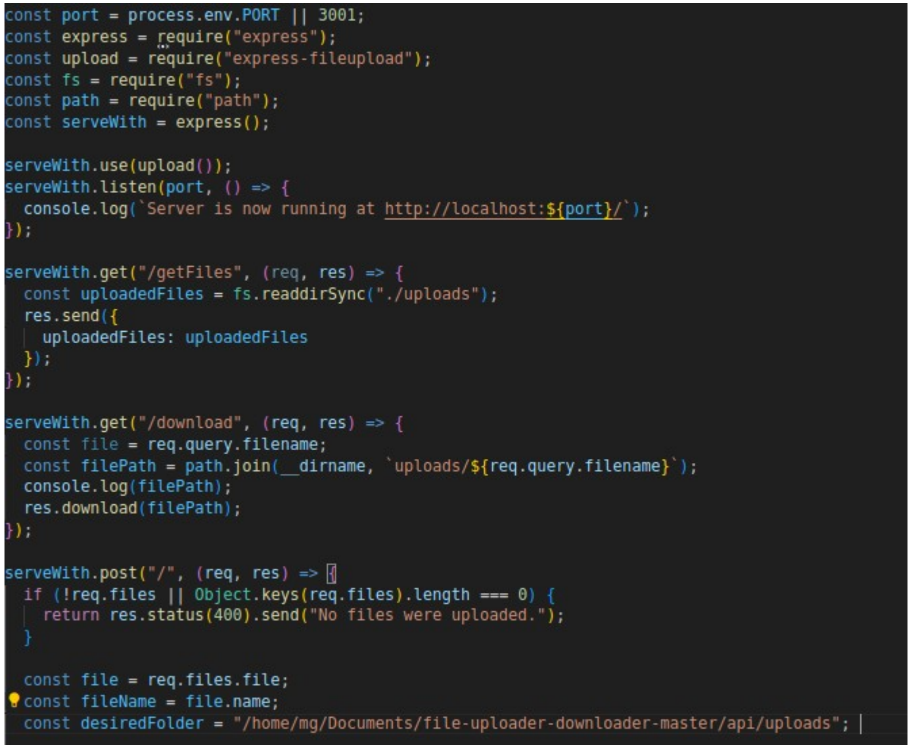


```
onFormSubmit(e) {
  e.preventDefault();
  if (this.state.file !== null) {

    this.fileUpload(this.state.file).then(res => {
      this.setState({
        fileUploaded: true,
        response: res.data.message,
        file: null
      });
    });
    this.timer = setTimeout(this.hadleDelay.bind(this), 2000);
  } else {
    this.setState({
      fileUploaded: true,
      response: "No files chosen"
    });
    this.timer = setTimeout(this.hadleDelay.bind(this), 1000);
  }

  this.~~refs~~.inputfile.value = "";
}
onChange(e) {
  this.setState({
    file: e.target.files[0]
  });
}
fileUpload(file) {
  const formData = new FormData();
  formData.append("file", file);
  return post("/", formData);
}
fileDownload(index, filename, evnt) {
  if (evnt.target.className === "download-btn") {
```

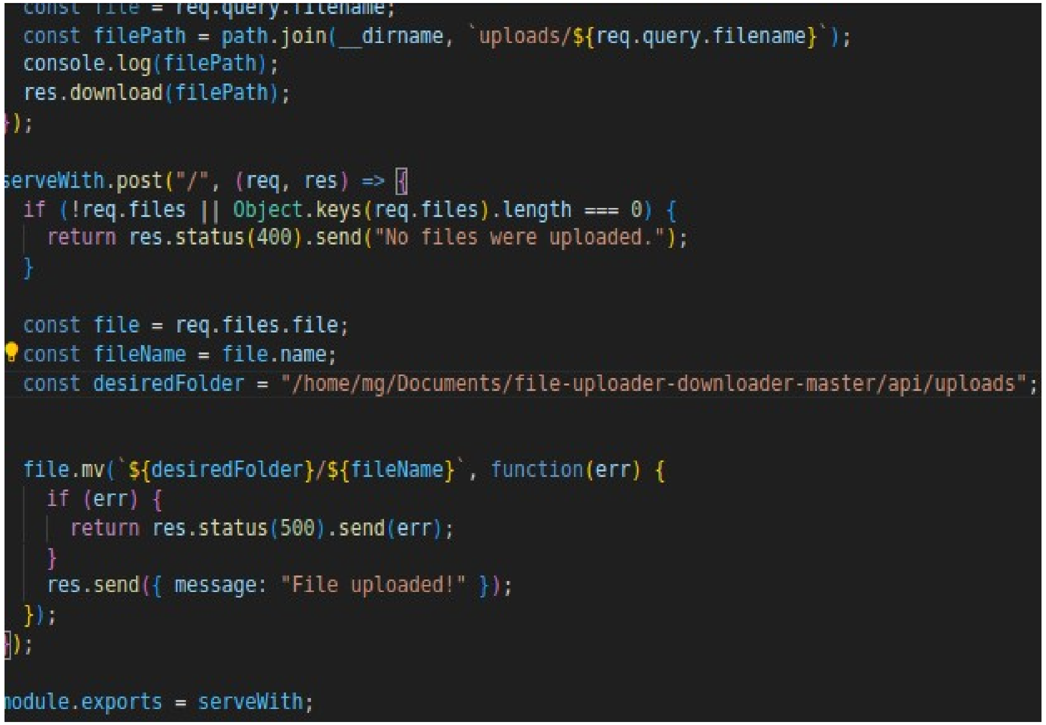
Frontend Snapshots :



Backend Code Snapshots :



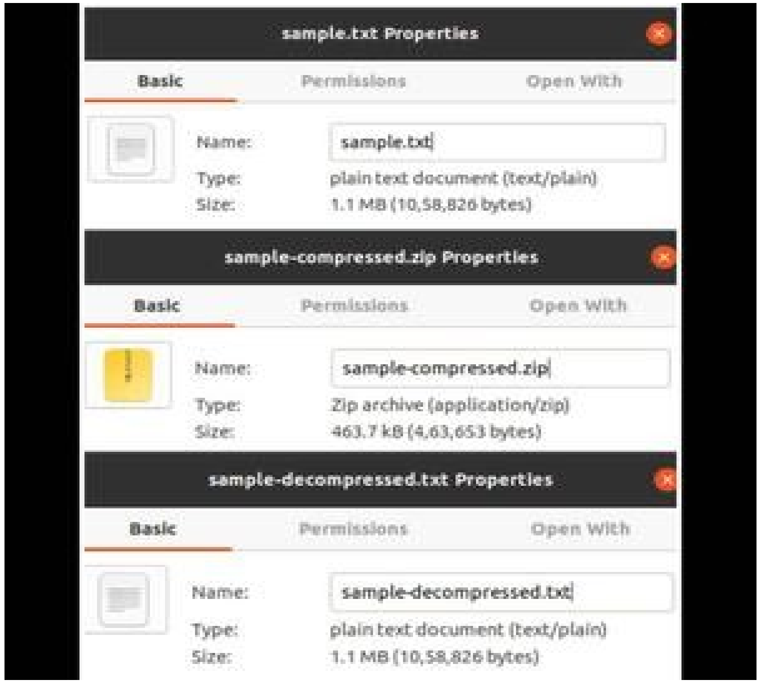
```
const port = process.env.PORT || 3001;
const express = require("express");
const upload = require("express-fileupload");
const fs = require("fs");
const path = require("path");
const serveWith = express();

serveWith.use(upload());
serveWith.listen(port, () => {
  console.log(`Server is now running at http://localhost:${port}/`);
});

serveWith.get("/getFiles", (req, res) => {
  const uploadedFiles = fs.readdirSync("./uploads");
  res.send({
    uploadedFiles: uploadedFiles
  });
});

serveWith.get("/download", (req, res) => {
  const file = req.query.filename;
  const filePath = path.join(__dirname, `uploads/${req.query.filename}`);
  console.log(filePath);
  res.download(filePath);
});

serveWith.post("/", (req, res) => {
  if (!req.files || Object.keys(req.files).length === 0) {
    return res.status(400).send("No files were uploaded.");
  }

  const file = req.files.file;
  const fileName = file.name;
  const desiredFolder = "/home/mg/Documents/file-uploader-downloader-master/api/uploads";
```



```
  const filePath = path.join(__dirname, `uploads/${req.query.filename}`);
  console.log(filePath);
  res.download(filePath);
});

serveWith.post("/", (req, res) => {
  if (!req.files || Object.keys(req.files).length === 0) {
    return res.status(400).send("No files were uploaded.");
  }

  const file = req.files.file;
  const fileName = file.name;
  const desiredFolder = "/home/mg/Documents/file-uploader-downloader-master/api/uploads";


  file.mv(`${desiredFolder}/${fileName}`, function(err) {
    if (err) {
      return res.status(500).send(err);
    }
    res.send({ message: "File uploaded!" });
  });
});

module.exports = serveWith;
```

Results :

- **We can achieve a compression rate of nearly 50 percent. When there will be a large amout of files then the algo will be very beneficial.**


sample.txt Properties

Basic | Permissions | Open With

Name: sample.txt
Type: plain text document (text/plain)
Size: 1.1 MB (10,58,826 bytes)

sample-compressed.zip Properties

Basic | Permissions | Open With

Name: sample-compressed.zip
Type: Zip archive (application/zip)
Size: 463.7 kB (4,63,653 bytes)

sample-decompressed.txt Properties

Basic | Permissions | Open With

Name: sample-decompressed.txt
Type: plain text document (text/plain)
Size: 1.1 MB (10,58,826 bytes)