Chuyên đề tổ chức dữ liệu

BÁO CÁO BÀI TẬP 4

Thuật toán nén Huffman

Sinh viên thực hiện:

- **Họ và tên:** Trần Minh Đức
- **Mã sinh viên:** 20810206

NỘI DUNG BÁO CÁO

---

1. GIỚI THIỆU VỀ MÃ HUFFMAN VÀ CÁCH NÉN FILE

1.1 Nguyên lý nén Huffman

Mã Huffman là một thuật toán nén dữ liệu không mất mát (lossless compression) được phát minh bởi David A. Huffman năm 1952. Thuật toán này dựa trên nguyên lý mã hóa entropy, trong đó các ký tự xuất hiện thường xuyên sẽ được gán mã ngắn hơn, còn các ký tự hiếm gặp sẽ có mã dài hơn.

1.2 Cách hoạt động

1. **Phân tích tần suất**: Đếm số lần xuất hiện của mỗi byte trong file gốc
2. **Xây dựng cây Huffman**: Tạo cây nhị phân tối ưu dựa trên tần suất
3. **Tạo bảng mã**: Gán mã nhị phân cho mỗi byte từ gốc đến lá của cây
4. **Mã hóa**: Thay thế mỗi byte gốc bằng mã Huffman tương ứng
5. **Lưu trữ**: Ghi bảng mã và dữ liệu nén vào file

1.3 Ưu điểm của Huffman

- **Tỷ lệ nén tốt**: Thường đạt 50-70% cho file text và nhiều loại file khác
- **Không mất dữ liệu**: Giải nén hoàn toàn chính xác file gốc
- **Tự thích ứng**: Tối ưu cho từng file cụ thể dựa trên nội dung thực tế

1.4 Mã canonical Huffman

Phiên bản nâng cao sử dụng "mã canonical" giúp giảm kích thước header bằng cách sắp xếp lại các mã Huffman theo thứ tự chuẩn hóa, cho phép lưu trữ bảng mã một cách hiệu quả hơn.

---

2. CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

2.1 Cấu trúc dữ liệu chính

Cây Huffman (HuffmanNode)

```
export interface HuffmanNode {
 weight: bigint; // Tổng tần suất xuất hiện
 symbol: number | null; // Byte 0-255 nếu là lá, null nếu là nút
trong
 left: HuffmanNode | null;
 right: HuffmanNode | null;
 minSymbol: number;  // Byte nhỏ nhất trong nhánh con
}
```

Mã Huffman (HuffmanCode)

```
export interface HuffmanCode {
 code: bigint;   // Mẫu bit nhị phân, lưu ở phần LSB, dài = length bit
 length: number; // Số bit hợp lệ trong 'code'
}
```

Min-heap ưu tiên (MinHeap)

Sử dụng để xây dựng cây Huffman một cách hiệu quả với độ phức tạp O(n log n).

2.2 Thuật toán chính

Thuật toán đếm tần suất (countByteFrequency)

- Đọc file theo stream để xử lý file lớn
- Sử dụng Uint32Array[256] để đếm tần suất mỗi byte
- Trả về mảng tần suất và tổng số byte

Thuật toán xây dựng cây Huffman (buildHuffmanTree)

- Tạo các nút lá cho byte có tần suất > 0
- Sử dụng min-heap để luôn lấy 2 nút có trọng lượng nhỏ nhất
- Ghép cặp nút thành cây nhị phân hoàn chỉnh

Thuật toán tạo mã Huffman (deriveHuffmanCodes)

- Duyệt cây theo chiều sâu (DFS) không đệ quy
- Gán mã nhị phân: trái = 0, phải = 1
- Mã bắt đầu từ gốc, tích lũy theo đường đi

Thuật toán canonical Huffman (buildCanonicalCodes)

- Sắp xếp mã theo độ dài tăng dần, sau đó theo giá trị symbol
- Tính toán mã bắt đầu cho mỗi độ dài
- Gán mã từ nhỏ đến lớn trong mỗi nhóm độ dài

Thuật toán nén bit-level (encodePayload)

- Sử dụng BitWriter để ghi bit theo thứ tự MSB-first
- Với mỗi byte trong file, tra cứu và ghi mã Huffman tương ứng
- Xử lý padding để đảm bảo byte alignment

Thuật toán giải nén (decodePayloadToFile)

- Xây dựng cây giải mã từ bảng canonical codes
- Đọc từng bit và duyệt cây để tìm symbol
- Khi đến lá, xuất byte và quay lại gốc

3. MÔ TẢ MÃ NGUỒN VÀ KẾT QUẢ MINH HỌA

3.1 Cấu trúc project

```
src/
├── index.ts              # Chương trình console chính với menu interactive
├── compress.ts           # Hàm nén file - điều phối toàn bộ quá trình nén
├── decompress.ts         # Hàm giải nén file - điều phối quá trình giải nén
├── functions/            # Thư mục chứa các module utility
│   ├── bitIO.ts          # Bit-level I/O operations (BitWriter/BitReader)
│   ├── huffmanTree.ts    # Xây dựng cây Huffman với MinHeap
│   ├── huffmanCodes.ts   # Tạo mã Huffman từ cây
│   ├── canonical.ts      # Canonical Huffman codes
│   ├── encode.ts         # Encoding logic với stream processing
│   ├── decode.ts         # Decoding logic với tree traversal
│   ├── buildHeader.ts    # Header format và serialization
│   ├── readHeader.ts     # Header parsing và deserialization
│   └── countByteFrequency.ts # Đếm tần suất byte với streaming
├── output/               # Thư mục chứa file nén/giải nén
├── data.txt             # File test dữ liệu
├── package.json         # Dependencies và scripts
└── tsconfig.json        # TypeScript configuration
```

3.2 Quy trình hoạt động chi tiết

BƯỚC 1: NÉN FILE (COMPRESS PROCESS)

Quy trình nén được thực hiện bởi hàm compressFile() trong compress.ts, bao gồm các bước con sau:

1.1 Phân tích tần suất bytes

- Function: countByteFrequency(filePath) trong countByteFrequency.ts
- Các bước chi tiết:
    - Khởi tạo mảng freq = new Uint32Array(256) để lưu tần suất
    - Tạo ReadStream để đọc file theo chunks (tránh load toàn bộ vào memory)
    - Với mỗi chunk dữ liệu: duyệt từng byte và tăng freq[byteValue]++
    - Trả về object { freq, totalBytes }
- Đầu ra: Mảng freq: Uint32Array[256] chứa tần suất, totalBytes: number

1.2 Xây dựng cây Huffman

- Function: buildHuffmanTree(freq) trong huffmanTree.ts
- Các bước chi tiết:
    - Tạo các nút lá cho các byte có freq[byte] > 0
    - Khởi tạo MinHeap ưu tiên theo weight (sau đó theo minSymbol để tie-break)
    - Thêm tất cả nút lá vào MinHeap
    - Lặp: lấy 2 nút có trọng lượng nhỏ nhất, tạo nút cha, thêm lại vào heap
    - Dừng khi chỉ còn 1 nút (root của cây)
- Đầu ra: Cây Huffman với root node và số lượng symbols

1.3 Tạo mã Huffman từ cây

- Function: deriveHuffmanCodes(root) trong huffmanCodes.ts
- Các bước chi tiết:
    - Khởi tạo mảng lengths = new Uint8Array(256) và codes = new Array(256)
    - Sử dụng stack để duyệt cây theo DFS không đệ quy
    - Với mỗi nút: nếu là lá thì gán lengths[symbol] = currentLength và codes[symbol] = {code, length}
    - Duyệt trái: thêm bit 0 (code << 1 | 0), duyệt phải: thêm bit 1 (code << 1 | 1)
- Đầu ra: Mảng độ dài lengths: Uint8Array[256] và mã codes: Array<HuffmanCode|null>

1.4 Chuyển thành mã canonical

- Function: buildCanonicalCodes(lengths) trong canonical.ts
- Các bước chi tiết:
    - Đếm số mã ở mỗi độ dài: codeCountPerLength[length]++
    - Tính mã bắt đầu cho mỗi độ dài: nextCode[length] = (currentCode + codeCount[length-1]) << 1
    - Sắp xếp các byte theo độ dài tăng dần, sau đó theo giá trị byte tăng dần
    - Gán mã canonical: canonicalCodes[byteValue] = { code: nextCode[length], length }
- Đầu ra: Mảng canonicalCodes: Array<CanonicalCode|null>

1.5 Nén payload

- Function: encodePayload(filePath, canonicalCodes) trong encode.ts
- Các bước chi tiết:
    - Khởi tạo BitWriter để ghi bit theo thứ tự MSB-first
    - Đọc lại file theo stream, với mỗi byte: tra cứu canonicalCodes[byteValue]
    - Ghi mã Huffman vào bit stream: bitWriter.writeBits(code.code, code.length)
    - Kết thúc ghi và lấy buffer payload với số bit padding
- Đầu ra: Buffer payload nén và số bit padding

1.6 Tạo header

- Function: buildHzipHeaderCanonical(lengths, outBytes, padBits, originalExtension) trong buildHeader.ts
- Các bước chi tiết:
    - Thu thập các byte có mã Huffman (lengths[byte] > 0), sắp xếp tăng dần
    - Tạo các thành phần header: magic "HZIP", version 1, kích thước gốc, extension, symbol count
    - Tạo bảng Huffman: mảng byte chứa (symbol, codeLength) cho mỗi symbol
    - Ghép tất cả thành Buffer header theo thứ tự đã định
- Đầu ra: Buffer header với format HZIP v1

1.7 Ghi file kết quả

- Tạo tên file output: baseName.replace(ext, "") + ".hzip"
- Ghép header và payload thành file hoàn chỉnh
- Hiển thị thông tin: kích thước, tỷ lệ nén, thời gian

BƯỚC 2: GIẢI NÉN FILE (DECOMPRESS PROCESS)

Quy trình giải nén được thực hiện bởi hàm decompressFile() trong decompress.ts, bao gồm các bước con sau:

2.1 Đọc và phân tích header

- Function: parseHzipHeader(filePath) trong readHeader.ts
- Các bước chi tiết:
    - Đọc toàn bộ file vào buffer
    - Kiểm tra magic string "HZIP" và version 1
    - Đọc kích thước gốc (8 bytes, little-endian)
    - Đọc độ dài extension và tên extension
    - Đọc số lượng symbol và bảng Huffman (mảng (symbol, codeLen))
    - Đọc số bit padding
    - Phần còn lại là payload đã nén
- Đầu ra: Object chứa originalSize, originalExtension, entries, padBits, payload

2.2 Tái tạo bảng mã canonical

- Function: rebuildCanonicalFromTable(entries) trong canonical.ts
- Các bước chi tiết:
    - Sắp xếp entries theo độ dài tăng dần, sau đó theo symbol tăng dần
    - Tính codeCountPerLength và nextCodeStart cho mỗi độ dài

    - Với mỗi entry đã sắp xếp: gán canonicalCodes[symbol] = { code: nextCodeStart[length], length }
    - Tăng nextCodeStart[length] += 1 cho symbol tiếp theo
- Đầu ra: Mảng canonicalCodes: Array<CanonicalCode|null>

2.3 Giải nén payload

- Function: decodePayloadToFile(payload, padBits, canonicalCodes, originalSize, outputPath) trong decode.ts
- Các bước chi tiết:
    - Xây dựng cây giải mã từ bảng canonical codes
    - Khởi tạo BitReader với payload và padBits
    - Với mỗi byte cần giải mã: bắt đầu từ root, đọc bit để duyệt trái/phải
    - Khi đến nút lá: ghi symbol vào output, quay lại root
    - Lặp cho đến khi đủ originalSize bytes
    - Ghi toàn bộ dữ liệu đã giải mã ra file
- Đầu ra: File gốc được khôi phục hoàn toàn

3.3 Kết quả và hiệu suất

File	**Kích thước gốc**	**Kích thước nén**	**Tỷ lệ nén**
test-json.json	36,353,321 bytes	19,535,704 bytes	53.74%
test-txt.txt	12,274 bytes	6,797 bytes	55.38%
test-png.png	112,268 bytes	105,692 bytes	94.14%

```
=== Huffman File Compression Tool ===
1. Nén file
2. Giải nén file
3. Thoát chương trình
==================================
Chọn chức năng (1-3): 1
Nhập đường dẫn file cần nén: ./file-test/test-json.json
📦 Đang nén file test-json.json (36353321 bytes)...
✅ Nén file thành công
📁 *** Lưu tại: output/test-json.hzip ***
📏 Kích thước nén: 19535704 bytes
📊 Tỷ lệ nén: 53.74%
⏱️ Thời gian: 4314ms

Nhấn Enter để tiếp tục...
```

```
=== Huffman File Compression Tool ===
1. Nén file
2. Giải nén file
3. Thoát chương trình
==================================
Chọn chức năng (1-3): 1
Nhập đường dẫn file cần nén: ./file-test/test-txt.txt
📦 Đang nén file test-txt.txt (12274 bytes)...
✅ Nén file thành công
📁 *** Lưu tại: output/test-txt.hzip ***
📏 Kích thước nén: 6797 bytes
📊 Tỷ lệ nén: 55.38%
⏱️ Thời gian: 15ms

Nhấn Enter để tiếp tục...
```

```
=== Huffman File Compression Tool ===
1. Nén file
2. Giải nén file
3. Thoát chương trình
==================================
Chọn chức năng (1-3): 1
Nhập đường dẫn file cần nén: ./file-test/test-png.png
📦 Đang nén file test-png.png (112268 bytes)...
✅ Nén file thành công
📁 *** Lưu tại: output/test-png.hzip ***
📏 Kích thước nén: 105692 bytes
📊 Tỷ lệ nén: 94.14%
⏱️ Thời gian: 40ms

Nhấn Enter để tiếp tục...
```

3.4 Minh họa quá trình nén

Chuỗi gốc: "Tran Minh Duc 20810206"

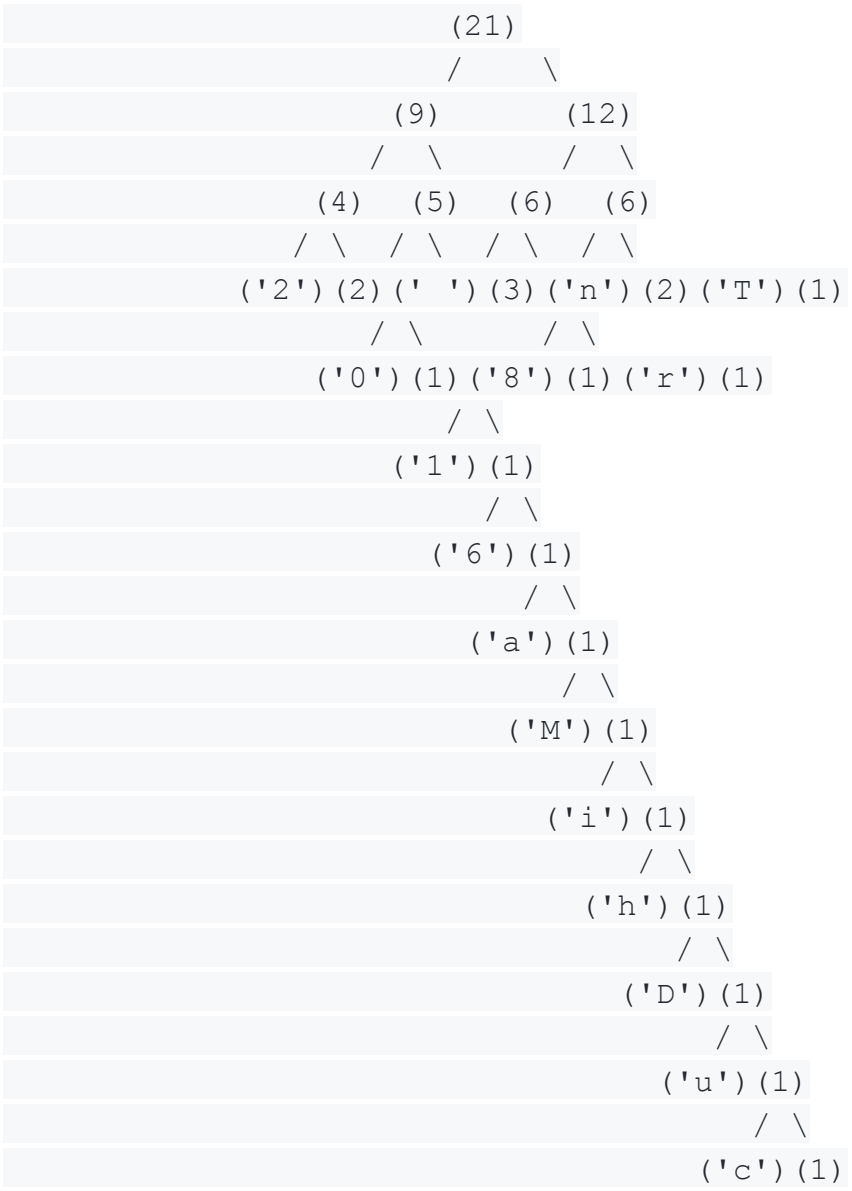
Nội dung chuỗi:

Tran Minh Duc 20810206

Bảng tần suất ký tự:

Ký tự	**Tần suất**
' '	3
'n'	2
'2'	1
'0'	1
'8'	1
'1'	1
'6'	1
'T'	1
'r'	1
'a'	1
'M'	1
'i'	1
'h'	1
'D'	1
'u'	1
'c'	1

Cây Huffman hoàn chỉnh:



Bảng mã Huffman theo cây hoàn chỉnh:

Ký tự	Tần suất	Mã Huffman	Độ dài bit
' '	3	011	3
'n'	2	101	3
'2'	1	000	3
'T'	1	100	3
'0'	1	0100	4
'8'	1	0101	4
'r'	1	1100	4
'1'	1	01000	5
'6'	1	010010	6
'a'	1	010011	6
'M'	1	0100110	7
'i'	1	01001110	8
'h'	1	010011110	9
'D'	1	0100111110	10
'u'	1	01001111110	11
'c'	1	01001111111	11

Header format (.hzip):

HZIP v1 | size:21 | ext:.txt | symbols:16 | table | pad:0 | payload

4. HƯỚNG DẪN SỬ DỤNG CHƯƠNG TRÌNH

4.1 Cài đặt

```
cd src
yarn install
```

4.2 Chạy chương trình

Chạy menu interactive:

```
yarn start
```

Menu sẽ hiển thị 3 lựa chọn:

1. Nén file - Nhập đường dẫn file cần nén
2. Giải nén file - Nhập đường dẫn file .hzip cần giải nén
3. Thoát chương trình

4.3 Ví dụ sử dụng

Nén file:

```
yarn start
# Chọn 1
# Nhập: data.txt
# Kết quả: data.hzip được tạo trong thư mục output/
```

Giải nén file:

```
yarn start
# Chọn 2
# Nhập: output/data.hzip
# Kết quả: data.txt được khôi phục trong thư mục output/
```

4.4 Định dạng file .hzip

File nén có cấu trúc như sau:

```
HZIP Header v1:
├── Magic: "HZIP" (4 bytes)
├── Version: 1 (1 byte)
├── Original Size: uint64 (8 bytes)
├── Extension Length: uint8 (1 byte)
├── Extension: UTF-8 bytes (N bytes)
├── Symbol Count: uint16 (2 bytes)
├── Huffman Table: (symbol, length) pairs
├── Pad Bits: 0-7 (1 byte)
└── Compressed Payload: bit stream
```