TRƯỜNG ĐIỆN-ĐIỆN TỬ

BÀI TẬP LỚN

Cấu Trúc Dữ Liệu và Giải Thuật

Đề tài: Huffman Coding

Giảng viên hướng dẫn: TS. Tạ Thị Kim Huệ

Sinh viên thực hiện:

HỌ VÀ TÊN	MSSV
Nguyễn Cao Minh	20224411

Hà Nội, ngày 20 tháng 01 năm 2025

Chương 1: Giới thiệu

Mã hóa Huffman là một thuật toán nén dữ liệu không tổn hao, được đề xuất bởi David A. Huffman vào năm 1952. Đây là một trong những phương pháp phổ biến và hiệu quả nhất trong lĩnh vực nén dữ liệu, giúp giảm kích thước của tệp tin bằng cách thay thế các ký tự thường xuất hiện bằng các mã nhị phân ngắn hơn, trong khi các ký tự ít xuất hiện hơn sẽ được mã hóa bằng các chuỗi dài hơn. Nhờ tính tối ưu trong việc giảm độ dài mã trung bình, mã hóa Huffman được ứng dụng rộng rãi trong nhiều lĩnh vực như nén tệp tin (ZIP, GZIP), nén ảnh (JPEG, PNG), nén âm thanh (MP3), và trong truyền thông dữ liệu để giảm băng thông và tối ưu hóa tốc độ truyền tải. Bên cạnh đó, thuật toán này còn được sử dụng trong lĩnh vực trí tuệ nhân tạo và xử lý ngôn ngữ tự nhiên nhằm tối ưu hóa không gian lưu trữ và cải thiện hiệu suất hệ thống.

Chương 2: Phân tích bài toán

Mã hóa Huffman chia thành 2 bài toán chính:
Bài toán 1. Tạo từ mã và Mã hóa.
Bài toán 2. Giải mã.

1. Tạo từ mã và Mã hóa:

Đối với bài toán tạo từ mã và mã hóa, yêu cầu của bài toán là: **Cho trước một đoạn văn bản và làm sao để giảm kích bộ nhớ để lưu một ký tự?** Một ký tự (word) được lưu trong máy tính (đối với ký tự mã hóa ASCII) chiếm kích thước 8 bits. Tuy nhiên trong văn bản thường ngày, chúng ta không sử dụng hết 256 ký tự trong bảng mã. Do đó, hoàn toàn có thể mã hóa văn bản với các ký tự có kích thước trung bình < 8 bits.

Quay trở lại bài toán, mã hóa Huffman được thực thi dựa theo tần số xuất hiện ký tự. Do đó, giả sử chúng ta đã có tần số xuất hiện của các ký tự. Ta có:

Đầu vào bài toán: Tần số xuất hiện của từng ký tự.

Đầu ra: Từ mã Huffman tương ứng với từng ký tự.

Giải thuật mã hóa Huffman sẽ sử dụng cấu trúc dữ liệu là priority queue, hay nói cách khác bài toán sẽ sử dụng một queue và với mỗi lần lấy 2 phần tử có giá trị nhỏ nhất, đây cũng là điều kiện so sánh ưu tiên các phần tử.

Các bước tiến hành giải thuật như sau:

*Bước 1: Phân tích đầu vào
Cho một tập ký tự cùng với tần suất xuất hiện của chúng
*Bước 2: Xây dựng cây mã hóa Huffman
Cấu trúc dữ liệu được sử dụng ở đây là Cây nhị phân với nút con trái tượng trưng cho bit 0, nút con phải tượng trưng cho bit 1.
Cấu trúc dữ liệu của Node gồm:

```
struct Node {
    char data; // Character
    float frequency;  // Frequency
    int visit;
    int ending_sign;
    // visit is used to check if the node is visited in Queue
    struct Node *pLeft, *pRight;
};
```

Các bước xây dựng cây Huffman:

1. Tạo nút lá:
    - Mỗi ký tự được xem như một nút lá, với trọng số bằng tần suất xuất hiện của nó.
2. Tạo hàng đợi ưu tiên (Priority Queue):
    - Đưa tất cả các ký tự vào hàng đợi ưu tiên, trong đó nút có tần suất nhỏ nhất sẽ được xử lý trước.
3. Kết hợp các nút nhỏ nhất:
    - Lặp lại các bước sau đây cho đến khi chỉ còn một nút duy nhất:
        - Lấy hai nút có trọng số nhỏ nhất từ hàng đợi.
        - Tạo một nút mới là cha của hai nút này, với trọng số bằng tổng trọng số của chúng.
        - Đưa nút mới vào hàng đợi ưu tiên.
4. Lặp lại cho đến khi còn một nút (nút gốc của cây Huffman):
    - Nút cuối cùng còn lại sẽ là gốc của cây Huffman.

Mã giả C cho thuật toán:

```
Function Build_Huffman_Tree(data[], freq[], size):
// Bước 1: Khởi tạo hàng đợi
queue = create_new_Queue(size * size)
// Bước 2: Đưa các ký tự và tần suất vào hàng đợi
for i from 0 to size - 1: queue.array[i] = create_new_Node(data[i],
freq[i]) queue.size = size
// Bước 3: Khởi tạo hai nút mặc định
default_x = create_new_Node('?', 2.0)
default_y = create_new_Node('?', 2.0)
i = 0
while i < size:
x = NULL
y = NULL
// Bước 4: Gán giá trị mặc định cho x và y
x = default_x
y = default_y
// Bước 5: Tìm hai nút có tần suất nhỏ nhất chưa được xét
for j from 0 to size - 1:
if queue.array[j].visit == 0 AND queue.array[j].frequency <=
x.frequency:
y = x
 x = queue.array[j]
```

```
continue
else if queue.array[j].visit == 0 AND queue.array[j].frequency <
y.frequency:
        y = queue.array[j]
// Bước 6: Kiểm tra dấu hiệu kết thúc
if x.frequency == 0.0:
        x.ending_sign = 1
// Đánh dấu như là điểm kết thúc
// Bước 7: Tạo nút mới kết hợp từ hai nút nhỏ nhất
newNode = create_new_Node('$', x.frequency + y.frequency)
newNode.pLeft = x newNode.pRight = y
// Bước 8: Chèn nút mới vào hàng đợi
queue.array[size] = newNode size = size + 1
// Đánh dấu các nút đã xét
x.visit = 1
y.visit = 1
i = i + 2
// Bước 9: Kiểm tra điều kiện dừng, nếu cây đã hoàn thành thì trả về
nút gốc
if i == size - 1:
        free(default_x)
        free(default_y)
        return newNode
// Trả về gốc của cây Huffman
 End Function
```

2. Giải mã:

Đầu vào: Các ký tự kèm từ mã và một đoạn dữ liệu mã hóa
Đầu ra: Nội dung văn bản được giải mã.
Ý tưởng cốt lõi ở đây là xây dựng lại cây Huffman từ các từ mã.
Các bước dựng lại cây Huffman:
*Bước 1: Tạo cây với nút gốc ban đầu.
*Bước 2: Duyệt các từ mã tương ứng với các ký tự đó.
Với mỗi ký tự ta sẽ duyệt hết các bits từ mã.
Với mỗi ký tự 0 và nút con trái chưa được cấp phát, ta sẽ cấp phát và gán vào nút cha.
Với mỗi ký tự 1 và nút con phải chưa được cấp phát, ta sẽ cấp phát và gán vào nút cha.
Khi đến bits cuối cùng, node mang bit đó sẽ được gán là nút lá và mang giá trị của ký tự đó.
Mã giả C:

```
Function Build_Huffman_Tree(huffman_codes):
    // Bước 1: Tạo một nút gốc rỗng cho cây Huffman
    root = create_new_node()

    // Bước 2: Duyệt qua từng ký tự và mã Huffman trong danh sách
    for each (character, code) in huffman_codes:
        current_node = root

        // Duyệt từng bit trong mã Huffman
        for each bit in code:
            if bit == '0':
                if current_node.left is NULL:
                    current_node.left = create_new_node()
                current_node = current_node.left
            else if bit == '1':
                if current_node.right is NULL:
                    current_node.right = create_new_node()
                current_node = current_node.right

        // Gán ký tự vào nút lá cuối cùng
        current_node.character = character
```

```
    // Bước 3: Trả về cây Huffman đã xây dựng
    return root
```

Giải thuật giải mã:

Ta sẽ đưa liên tục các bit thông tin 0/1 vào cây với bit 0 chảy xuống node con trái, bit 1 chảy xuống node con phải.

Nếu node đó là Lá, tiến hành in ký tự lưu ở node đó và quay trở lại node gốc.

Mã giả C:

```
Function Huffman_Decode(encodedString, root):
    decodedString = ""
    currentNode = root

    for bit in encodedString:
        if bit == '0':
            currentNode = currentNode.left
        else:
            currentNode = currentNode.right

        // Nếu đến nút lá, lấy ký tự và quay về gốc
        if currentNode.left == NULL AND currentNode.right == NULL:
            decodedString += currentNode.character
            currentNode = root

    return decodedString
```