NÉN HUFFMAN

MỤC TIÊU

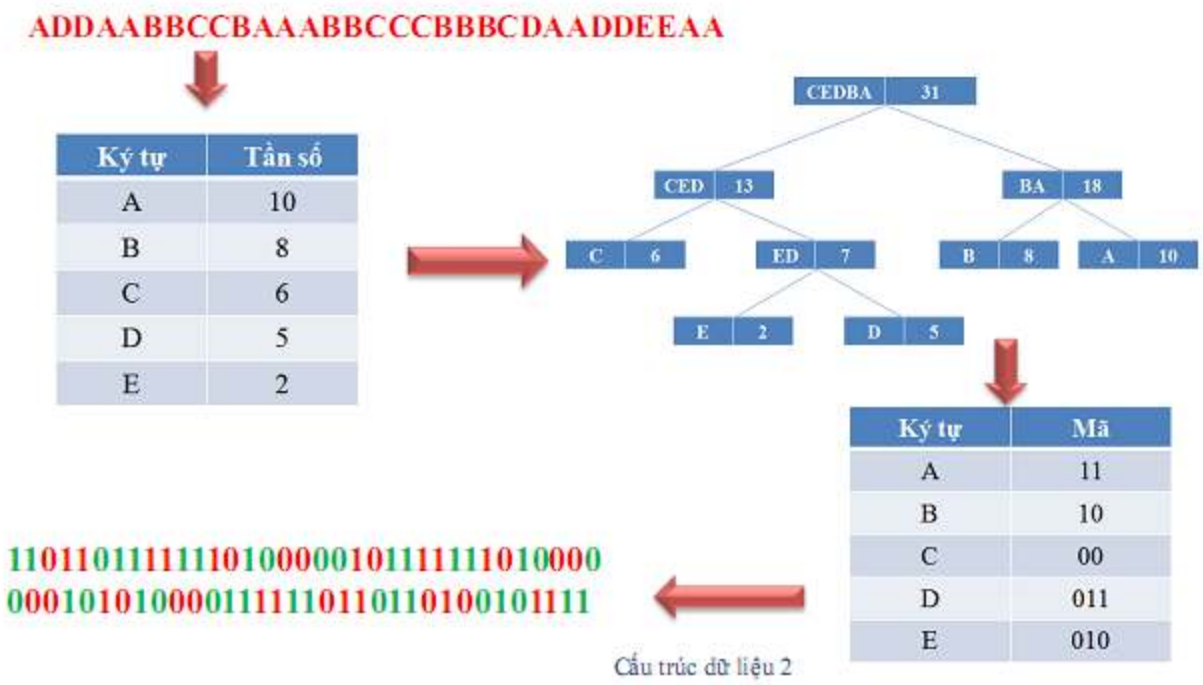
Hoàn thành bài thực hành này, sinh viên có thể:

- Nắm rõ các bước của thuật toán nén huffman tĩnh.

THỜI GIAN THỰC HÀNH

Từ 120 phút đến 400 phút

TÓM TẮT



Cấu trúc dữ liệu 2

Nén Huffman là phương pháp mã hóa bằng ***mã có độ dài thay đổi*** (variable length encoding) trong đó chỉ sử dụng vài bit để biểu diễn 1 ký tự và độ dài mã bit cho các ký tự không giống nhau (ký tự xuất hiện nhiều lần được biểu diễn bằng mã ngắn và ngược lại).

Thuật toán nén Huffman bao gồm 5 bước:

- B1: Lập bảng thống kê tần số xuất hiện của các ký tự.
- B2: Xây dựng cây Huffman dựa vào bảng thống kê tần số xuất hiện.
- B3: Phát sinh bảng mã bit cho từng ký tự tương ứng.
- B4: Duyệt tập tin, thay thế các ký tự trong tập tin bằng mã bit tương ứng.
- B5: Lưu lại thông tin của cây Huffman cho giải nén.

NỘI DUNG THỰC HÀNH

Cài đặt thuật toán nén Huffman. Dữ liệu được nhập từ file “input.txt”. Xuất ra màn hình chuỗi bit tương ứng với nội dung nhập.

Ví dụ

Với file input.txt có nội dung:

ABBCCCDDDD

Chuỗi bit tương ứng khi xuất ra sẽ là

10010110 11111110 000

Trong đó A (100) B (101) C(11) và D(0)

PHÂN TÍCH

Để biểu diễn một nút trong cây huffman, ta dùng:

```
struct NODE {
	unsigned char	c;		// ký tự của node
	int		freq;		// số lần xuất hiện
	bool	used;			// đánh dấu node có thuộc bảng thống kê ko
					// used = true: ko thuộc bảng thống kê
	int		nLeft;		// chỉ số nút con nằm bên trái
	int		nRight;		// chỉ số nút con nằm bên phải
};
```

Cây Huffman được lưu trữ dưới dạng mảng

```
	NODE  huffTree[MAX_NODE];
```

Trong đó MAX_NODE là 1 số nguyên > số lượng nút tối đa của cây Huffman.

(Ta có thể thấy số lượng nút tối đa của cây Huffman sẽ < 2^8 * 9)

Để biểu diễn một mã bit, ta dùng

```
struct MABIT {
	char*		bits;		// chua mang bit
	int		soBit;		// so luong bit cua ma
};
```

Bảng mã bit: mã bit của 256 ký tự

```
	MABIT bangMaBit[256];
```

BƯỚC 1: LẬP BẢNG THỐNG KÊ TẦN SỐ XUẤT HIỆN

Ta tạo ra 256 node tương ứng với 256 ký tự ASCII. Sau đó, đọc từng kí tự tương ứng từ tập tin nhập và tăng tần số xuất hiện của node tương ứng ký tự nhập.

Khởi tạo node:

```
	for (int i = 0; i < 256; i++) {
		huffTree[i].c = i;
		huffTree[i].freq = 0;
```

```
        ...

    }
```

Thống kê tần số:

```
    while (1) {
        fscanf(fi, "%c", &c);
        if (feof(fi)) {
            break;
        }
        huffTree[c].freq ++; // tang tan so xuat hien cua ky tu c
    }
```

Nhận xét:

Việc tạo ra 256 node có thể dư do trong văn bản hiếm khi xảy ra trường hợp 256 ký tự ASCII cùng xuất hiện nhưng cho phép ta truy xuất nhanh chóng đến node tương ứng với ký tự c (với quy ước node huffTree[c] tương ứng với ký tự c). Điều này rất quan trọng khi làm việc với các tập tin có độ dài lớn.

BƯỚC 2: XÂY DỰNG CÂY HUFFMAN

Các bước phát sinh cây:

- B1: Chọn trong bảng thống kê 2 phần tử có tần suất thấp nhất.
- B2: Tạo 2 node của cây cùng với node cha z có trọng số bằng tổng trọng số 2 nút con.
- B3: Loại 2 phần tử x, y khỏi bảng thống kê.
- B4: Thêm phần tử z vào bảng thống kê.
- B5: lặp lại bước 1 đến bước 4 cho đến khi còn 1 phần tử trong bảng thống kê.

Với cách lưu trữ cây Huffman dùng mảng như ở trên, các thao tác được thực hiện như sau:

- Chọn trong bảng thống kê có tần suất thấp nhất

Tìm 2 phần tử trong mảng huffTree có tần suất thấp nhất. Chỉ xét các phần tử thuộc bảng thống kê (huffTree[i].used == false) và có xuất hiện trong file dữ liệu (huffTree[i].freq > 0)

- Tạo node mới của cây Huffman có 2 nút con i, j

Thêm node được tạo vào cuối mảng, sử dụng nLeft và nRight để lưu vị trí 2 nút con.

```
            huffTree[nNode].freq = huffTree[i].freq + huffTree[j].freq;
            huffTree[nNode].nLeft = i;
            huffTree[nNode].nRight = j;
```

- Loại phần tử x, y khỏi bảng thống kê.

Để loại x, y khỏi bảng thống kê, gán

```
            huffTree[i].used = true;
            huffTree[j].used = true;
```

- Thêm phần tử nNode vào bảng thống kê

```
huffTree[nNode].used = false;
```

Ví dụ: với dữ liệu vào ABBCCCDDDD, bảng thống kê ban đầu:

	65	66	67	68	...	255
C	A	B	C	D		?
Freq	1	2	3	4		0
Used	FALSE	FALSE	FALSE	FALSE		FALSE
nLeft	-1	-1	-1	-1		-1
nRight	-1	-1	-1	-1		-1

Tìm 2 phần tử có tần suất thấp nhất: A (1) và B (2). Tạo node mới (nút 256) . Loại A, B khỏi bảng thống kê và thêm nút 256 vào bảng thống kê

	65	66	67	68	...	255	256
C	A	B	C	D		?	A
Freq	1	2	3	4		0	3
Used	TRUE	TRUE	FALSE	FALSE		FALSE	FALSE
nLeft	-1	-1	-1	-1		-1	65
nRight	-1	-1	-1	-1		-1	66

Tìm 2 phần tử có tần suất thấp nhất: C (3) và 256 (3) (Lưu ý A, B có used = TRUE không được xét) . Tạo node mới (nút 257) . Loại C, 256 khỏi bảng thống kê và thêm nút 257 vào bảng thống kê

	65	66	67	68	...	255	256	257
C	A	B	C	D		?	A	A
Freq	1	2	3	4		0	3	6
Used	TRUE	TRUE	TRUE	FALSE		FALSE	TRUE	FALSE
nLeft	-1	-1	-1	-1		-1	65	256
nRight	-1	-1	-1	-1		-1	66	67

Tương tự, sau khi tạo nút 258, chỉ còn 1 phần tử thuộc bảng thống kê (258). Quá trình tạo cây dừng.

	65	66	67	68	...	255	256	257	258
C	A	B	C	D		?	A	A	A
Freq	1	2	3	4		0	3	6	10
Used	TRUE	TRUE	TRUE	TRUE		FALSE	TRUE	TRUE	FALSE
nLeft	-1	-1	-1	-1		-1	65	256	68
nRight	-1	-1	-1	-1		-1	66	67	256

Sau quá trình tạo cây, nút huffTree[nNode – 1] chính là nút gốc của cây Huffman.

BƯỚC 3: PHÁT SINH BẢNG MÃ BIT

Duyệt cây Huffman từ nút gốc đến nút lá của các ký tự.

Nút gốc của cây Huffman: huffTree[nNode – 1]

Kiểm tra một nút là nút lá: huffTree[i].nLeft == -1 && huffTree[i].rRight == -1

Duyệt đệ quy từ nút gốc của cây huffman. Thêm bit 0 vào khi đi qua nhánh trái, thêm bit 1 khi đi qua nhánh phải.

BƯỚC 4: THAY THẾ KÝ TỰ BẰNG MÃ BIT

Sử dụng 1 biến unsigned char out để lưu chuỗi bit xuất ra. Duyệt các ký tự, bật (gán = 1) các bit tương ứng của biến out (tùy theo mã bit của kí tự). Xuất ra biến out khi chuỗi bit có độ dài 8 (đủ 1 byte).

Ôn tập: THAO TÁC XỬ LÝ BIT

Tham khảo các phép xử lý bit cơ bản (http://forums.congdongcviet.com/showthread.php?t=316)

2 thao tác cần thực hiện:

- Bật bit i của số nguyên out

```
out = out | (1 << i);
```

Ví dụ: bật bit 2 của biến out

	7	6	5	4	3	2	1	0
Out	0	0	1	0	0	0	0	0
1 << 2	0	0	0	0	0	1	0	0
out | (1 << 2)	0	0	1	0	0	1	0	0

- Lấy bit i của số nguyên out

```
(out >> i) i& 1
```

Ví dụ: lấy bit 2 của biến out

	7	6	5	4	3	2	1	0
Out	0	0	1	0	0	1	0	0
out >> 2	0	0	0	0	1	0	0	1
1	0	0	0	0	0	0	0	1
(out >> 2) & 1	0	0	0	0	0	0	0	1

CODE THAM KHẢO (StaticHuffman.rar)

Xem code đi kèm.

YÊU CẦU

1. Biên dịch chương trình tham khảo.
2. Nhập dữ liệu input.txt có nội dung "ACBCDBDCDD". Chạy tay thuật toán nén Huffman, cho biết kết quả bảng thống kê, cây huffman, bảng mã bit và chuỗi bit xuất ra.
3. Trong hàm `NenHuffman()`

Bỏ dấu comment (//) ở các dòng

```
//    XuatBangThongKe();
//    XuatCayHuffman(nRoot, 0);
//    XuatBangMaBit();
```

Chạy chương trình, so sánh kết quả xuất ra màn hình với kết quả của câu 2.

4. Trả lời câu hỏi ở các dòng có ghi chú ở 2 hàm ThongKeTanSoXuatHien() và XuatBangThongKe().
5. Trả lời câu hỏi ở các dòng có ghi chú ở 2 hàm TaoCayHuffman() và Tim2PhanTuMin().
6. Trả lời câu hỏi ở các dòng có ghi chú ở 2 hàm PhatSinhMaBit() và DuyetCayHuffmanx().
7. Trả lời câu hỏi ở các dòng có ghi chú ở 2 hàm NenHuffman() và MaHoa1KyTu() và XuatByte().

Áp dụng - Nâng cao

8. Sửa lại chương trình để xuất nội dung nén ra file "output.txt" thay vì xuất ra màn hình.

Ví dụ: với ví dụ ở trên, thay vì xuất ra màn hình chuỗi bit 10010110 11111110 000, xuất ra file 3 byte

ßÔ

trong đó ß (10010110)ô (11111110) (byte thứ 3 là 0 nên không thể hiển thị)

9. Sửa lại chương trình để xuất nội dung nén và các thông tin cần thiết ra file output.huf" sao cho có thể sử dụng file output.huf cho quá trình giải nén.
10. Viết hàm giải nén: đọc nội dung từ file output.huf ở câu 9 và xuất nội dung giải nén ra file decode.txt. So sánh nội dung decode.txt và input.txt