HƯỚNG DẪN THỰC HÀNH

NÉN HUFFMAN TĨNH

I. Mục tiêu

Hiểu rõ và cài đặt được thuật toán nén Huffman tĩnh (static Huffman coding).

II. Qui định nộp

- Sinh viên nộp một tập tin nén, có tên là ***<MSSV>.zip*** hoặc ***<MSSV>.rar*** chứa source code và báo cáo của chương trình.
- Sinh viên nộp kèm một file báo cáo ghi mức độ hoàn thành công việc của mình, các bộ dữ liệu mà mình test ở mỗi bài.
- Tất cả các bài tập sẽ lập trình theo command line (tham số dòng lệnh).
- Kiến trúc thư mục nộp:
    - MSSV
        - MSSV_BT1:
            - Source: chứa các file .cpp
            - Header: chứa các file .h
            - Test case: chứa các file input để test thử.
            - …
        - MSSV_BT2: ….
- Môi trường làm việc: Visual Studio 2015 hoặc các môi trường tương đương. Không sử dụng các hàm bị lỗ hổng bảo mật như gets, …
- Hạn nộp: xem link trên Moodle.
- Bài giống nhau hay nộp file rác sẽ 0 điểm MÔN HỌC.

1. Nội dung

Áp dụng mã Huffman tĩnh để nén tập tin nhị phân

Ta sẽ xem tập tin cần nén là tập tin nhị phân không có cấu trúc và đọc từng byte của tập tin cho đến khi nào hết thì dừng. Mỗi byte của tập tin sẽ tương ứng với một ký tự nào đó trong bảng mã ASCII nên từ nay khi nói đến ký tự *c* thì ta cũng hiểu đó là byte *c* và ngược lại.

Dữ liệu nhập: Tập tin bất kỳ có kích thước nhỏ hơn 500 KB.

Tham số dòng lệnh:

Pha nén: MSSV.exe compress X Y

Ví dụ: 151234.exe compress 151234.cpp 151234.out

Ý nghĩa: Dùng thuật toán nén Huffman tĩnh để nén tập tin X và lưu kết quả xuống tập tin Y.

Pha giải nén: MSSV.exe uncompress X Y Z

Ví dụ: 151234.exe uncompress 151234.out 151234.huf 151234.cpp

Ý nghĩa: Giải nén tập tin X đã được nén bằng thuật toán nén Huffman tĩnh, sử dụng thông tin phục vụ cho việc giải nén được lưu trong tập tin Y và lưu kết quả giải nén xuống tập tin Z.

Lưu ý: Khi thực hiện phần **Yêu cầu nâng cao**, tham số Y trong pha giải nén là không cần thiết và sẽ bị loại bỏ.

Các bước thực hiện

Bước 1. Khai báo cấu trúc cây (nhị phân) Huffman

Để biểu diễn một nút trong cây Huffman, ta khai báo cấu trúc node như sau:

```
struct node {
    unsigned char c;            // ký tự tại nút
    unsigned int f;             // tần số xuất hiện
    struct node *left, *right; // các nút con
};
```

Bước 2. Khởi tạo bảng tần số xuất hiện của mỗi ký tự

Ta ghi nhận tần số xuất hiện của mỗi byte bởi một mảng số nguyên không dấu *f* gồm 256 phần tử tương ứng với 256 ký tự trong bảng mã ASCII và quy ước *f*[*c*] là tần số xuất hiện của ký tự *c*. Mảng *f* được khai báo như sau:

```
unsigned int f[256];
```

Ban đầu tần số xuất hiện của tất cả các ký tự đều bằng 0 nên ta khởi gán giá trị 0 cho tất cả các phần tử của *f*.

```
memset(f, 0, sizeof(f));
```

Bước 3. Đọc tập tin và cập nhật bảng tần số

Đọc từng byte của tập tin cần nén và tăng tần số xuất hiện của nó lên một đơn vị. Lặp lại thao tác này đến hết tập tin thì dừng.

Duyệt trên bảng tần số *f*, nếu *f* [*i*] > 0 (nghĩa là byte *i* có xuất hiện trong tập tin cần nén) thì tạo mới một nút của cây Huffman mang ký tự *i* và có tần số là *f* [*i*]. Nút này được xem là gốc của cây nhị phân chỉ có một nút là chính nút đó. Ta sẽ gọi tập các nút gốc này là *rừng* và ký hiệu là *C*. Tập *C* sẽ được lấy làm đầu vào của thuật toán Huffman và được tổ chức bằng cấu trúc dữ liệu *hàng đợi có ưu tiên* như sẽ được giải thích ở bước tiếp theo.

Bước 4. Xây dựng cây Huffman

Ý tưởng của Huffman là từ rừng các cây chỉ có một nút (tập *C* vừa thu được ở bước trên), tại mỗi bước ta gộp hai cây làm một và lặp lại cho đến khi rừng chỉ còn một cây. Cụ thể, đầu tiên ta lấy ra hai nút *x* và *y* có tần số nhỏ nhất, sau đó tạo mới một nút *z*, cho *z* nhận *x* và *y* làm hai con và gán giá trị tần số của *z* bằng tổng giá trị tần số của *x* và *y*. Lúc này, *z* trở thành nút gốc và được thêm vào tập *C* còn *x* và *y* thì bị loại khỏi *C* do không còn là nút gốc nữa. Tiếp theo ta lại lấy ra trong *C* hai nút có tần số nhỏ nhất và gộp lại theo cùng một cách thức vừa nêu. Cứ tiếp tục như thế ta xây dựng được cây con ngày càng lớn hơn trong khi số cây trong rừng giảm đi một ở mỗi bước. Do đó, số lần lặp là $n - 1$ với $n = |C|$ là số phần tử ban đầu của *C*.

Nhận xét

Tại mỗi bước ta luôn phải chọn ra hai gốc có tần số nhỏ nhất. Để thực hiện việc này ta sẽ sử dụng cấu trúc dữ liệu trừu tượng *hàng đợi có ưu tiên* với độ ưu tiên là tần số nhỏ nhất để lưu các nút gốc (hay nói cách khác là dùng hàng đợi có ưu tiên để tổ chức tập hợp *C*). Một trong các cấu trúc dữ liệu thuận lợi cho tiêu chuẩn này là cấu trúc *đống* (với nút có tần số nhỏ nhất nằm trên đỉnh của đống). Cách cài đặt này là hiệu quả do mỗi thao tác thêm mới phần tử hoặc lấy phần tử có độ ưu tiên cao nhất ra khỏi hàng đợi chỉ tốn chi phí thời gian là $O(\log|C|)$.

Mã nguồn gợi ý:

```
node* huffman(PQueue& Q) {
     for (int i = 1; i < n; i++) {
          node *x, *y, *z;
          z = new node;
          z->left = x = dequeue(Q);
          z->right = y = dequeue(Q);
          z->f = x->f + y->f;
          insertQUEUE(Q, z);
     }
     return z;
}
```

Lưu ý: Sinh viên tự cài đặt cấu trúc *PQueue* và có thể sửa lại đoạn mã trên theo ý mình.

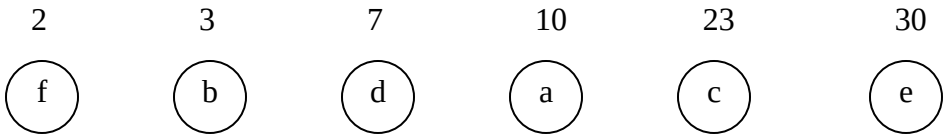
Ví dụ minh họa

Giả sử chuỗi ký tự nhập chứa các ký tự với tần số xuất hiện được cho trong bảng sau:

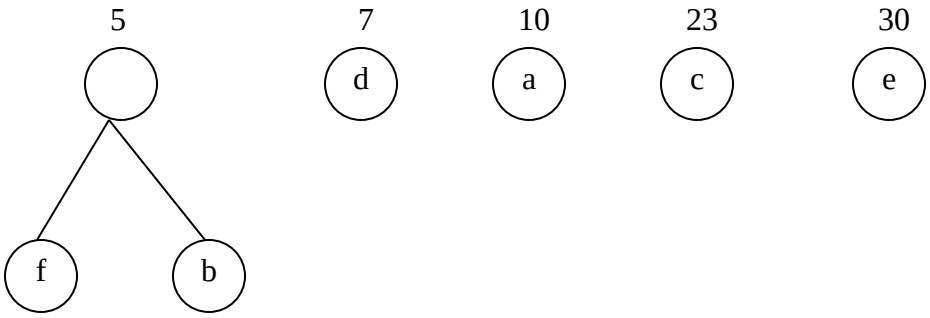
Ký tự	a	b	c	d	e	f
Tần số	10	3	23	7	30	2

Quá trình xây dựng cây Huffman diễn ra như sau:

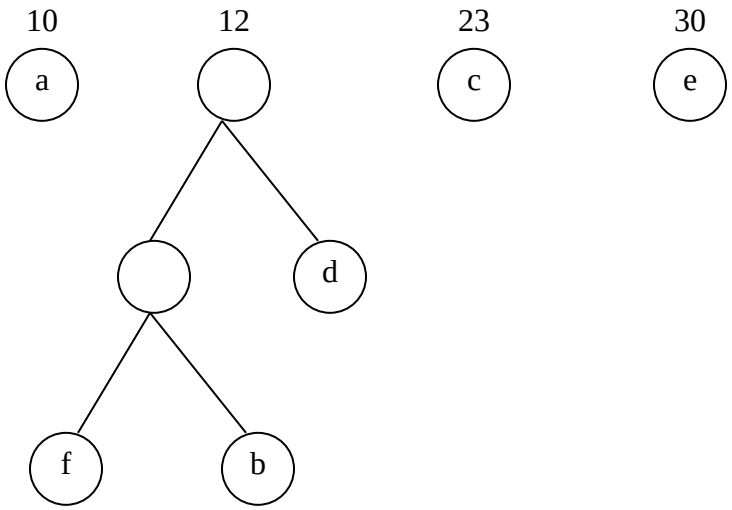
(i)



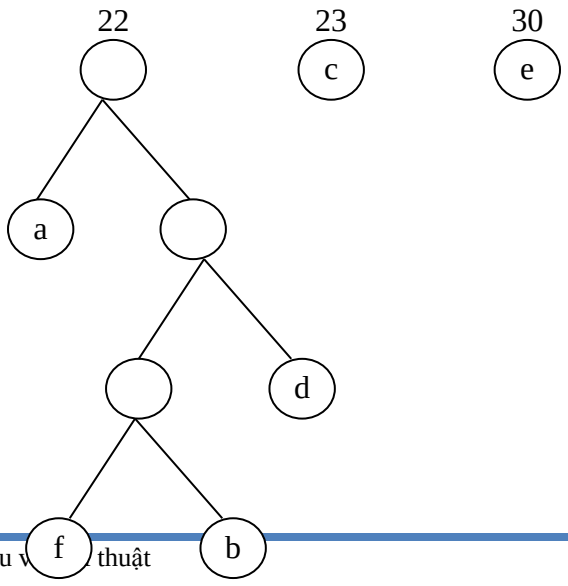
(ii)



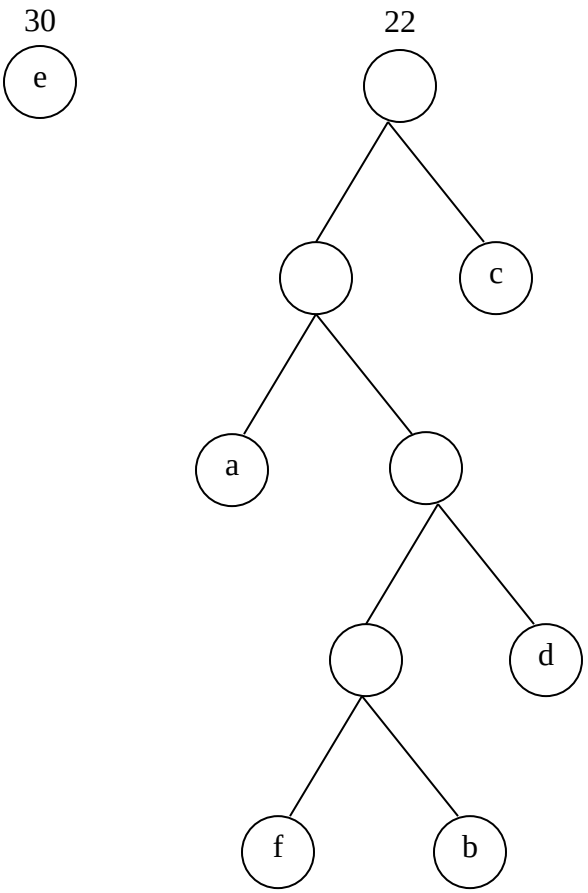
(iii)



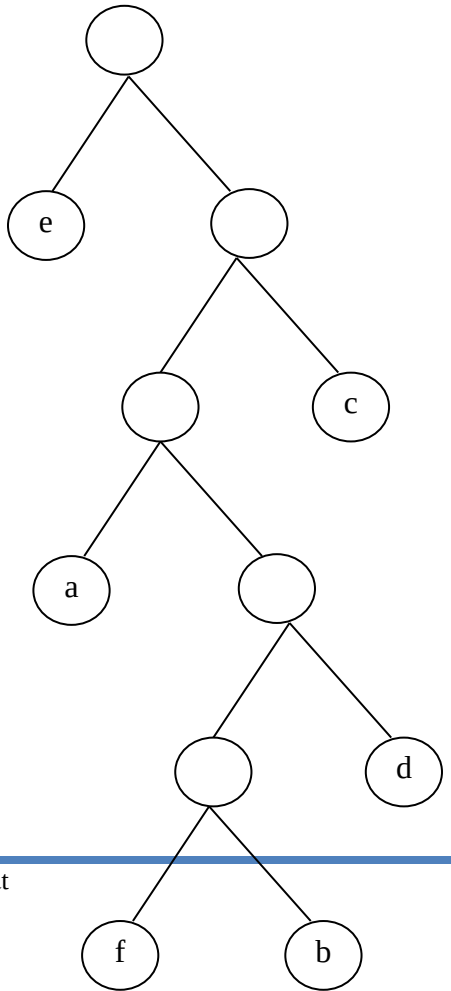
(iv)



(v)



(vi)



Bước 5. Phát sinh mã Huffman cho từng ký tự

Để lưu mã Huffman cho mỗi ký tự, ta dùng một mảng ký tự *code* gồm 256 phần tử ứng với 256 ký tự trong bảng mã ASCII.

```
unsigned char code[256];
```

Quá trình phát sinh mã Huffman diễn ra như sau: Xuất phát từ nút gốc của cây Huffman, phát sinh bit 0 nếu đi qua nhánh trái và phát sinh bit 1 nếu đi qua nhánh phải cho đến khi gặp nút lá. Các bit được sinh ra sẽ được lưu dưới dạng chuỗi ký tự '0' và '1' trong một biến tạm tên là *hufcode*. Giả sử ký tự tại nút lá là *c* thì *hufcode* chính là mã Huffman cho *c*, nghĩa là *code*[*c*] = *hufcode*. Quá trình này được lặp lại cho đến khi mọi ký tự tại nút lá đều nhận được mã Huffman.

Ví dụ đối với cây Huffman được minh họa ở trên, ta thu được mã bit của các ký tự như sau: *code*['e'] = '0', *code*['c'] = '11', *code*['a'] = '100', *code*['d'] = '1011', *code*['f'] = '10100', và *code*['b'] = '10101'.

Bước 6. Ghi kết quả nén xuống tập tin và lưu cây Huffman hỗ trợ cho việc giải nén

Đọc lại tập tin cần nén, mỗi byte *c* được đọc lên sẽ được thay bằng mã Huffman tương ứng là *code*[*c*], nếu *code*[*c*] có ít hơn 8 ký tự thì đọc byte kế tiếp và nối mã Huffman của byte này tiếp theo *code*[*c*], đến khi được một chuỗi có 8 ký tự thì cắt ra và chuyển thành 1 byte (một con số có 8 bit) rồi ghi xuống tập tin nén.

Lưu cây Huffman vào tập tin *output.huf* để phục vụ cho việc giải nén.

Bước 7. Giải nén tập tin

Đọc tập tin *output.huf* để phục hồi lại cây Huffman.

Xuất phát từ nút gốc của cây Huffman, đọc từng bit của tập tin nén, nếu là bit 0 thì đi qua nhánh trái, nếu là bit 1 thì đi qua nhánh phải, nếu đến nút lá thì ghi ký tự tại nút lá này xuống tập tin. Lặp lại thao tác này cho đến khi tập tin nén được đọc hết, tập tin thu được cuối cùng chính là tập tin gốc ban đầu (tập tin trước khi nén).

Yêu cầu nâng cao

Tìm một phương án lưu trực tiếp cây Huffman vào tập tin nén thay vì phải lưu riêng ra tập tin *output.huf* để khi giải nén ta chỉ làm việc với một tập tin duy nhất là tập tin nén.

HẾT