Nén dữ liệu (tiếp)

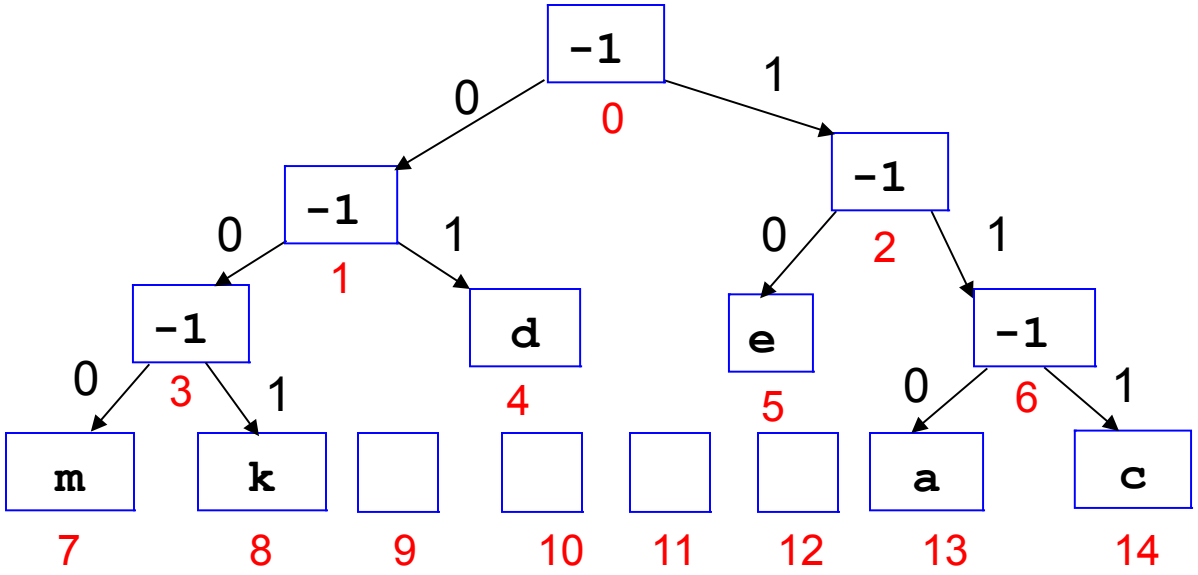
Nén tệp

- Mã Huffman có thể được sử dụng để nén tệp
- Tệp nén có thể được tổ chức như sau:

HM	Huffman Tree	Size	Data …

  - HM là mã tiền tố của tệp nén
  - Huffman Tree là một mảng biểu diễn cây Huffman sinh ra từ dữ liệu mã hóa
  - Size là kích thước của tệp nén, đơn vị là bit
  - Data chứa các bit của dữ liệu nén

Biểu diễn dạng mảng của cây Huffman



Tất cả các là của cây Huffman được gán nhãn thành các kí tự

15	-1	-1	-1	-1	d	e	-1	m	k					a	c

Kích thước mảng = $(2^0 + 2^1 + \ldots + 2^{n)}$ trong đó n là bậc lớn nhất của cây
Nút con của i ở vị trí 2i+1 và 2i+2
Nút cha của i ở vị trí (i-1)/2

Cài đặt

- Cho cấu trúc dữ liệu sau biểu diễn cây Huffman ở dạng mảng

```
typedef struct {
    int size;
    int * nodes;
} HuffmanTreeArray;
```

- Viết hàm chuyển đổi cây Huffman thành dạng mảng
  - HuffmanTreeArray tree2array(HuffmanTree);

Bài tập 1

- Viết lại các hàm trong các buổi trước để nén tệp bằng mã Huffman
- Chương trình được sử dụng để nén tệp ở chế độ dòng lệnh như sau:
  - $ compress in_file [out_file]
- Các hàm sau cần được cài đặt:
  - HuffmanTree makeHuffman (FILE * in);
  - void createHuffmanTable(HuffmanTree htree, Coding* htable);
  - HuffmanTreeArray tree2array(HuffmanTree);
  - void compressFile(FILE* in, FILE *out);

Giải mã tệp

- Đầu tiên, kiểm tra tiền tố “HM” của tệp
- Đọc cây ở dạng mảng
- Một khi có cây Huffman, bộ giải mã quét chuỗi bít đầu vào
  - Dữ liệu được lưu trữ ở mức bit thay vì mức byte
- Thuật toán quét
  - Đặt con trỏ ở gốc của cây
  - Nếu nút hiện tại có giá trị -1, đọc một bit mới
    - 0 $\Rightarrow$ chuyển con trỏ tới con trái
    - 1 $\Rightarrow$ chuyển con trỏ tới con phải
  - Nếu không, lấy kí tự mới ở nút, rời con trỏ về gốc

Bài tập 2

- Viết chương trình giải nén tệp được nén bởi Bài tập 1
- Sử dụng chế độ dòng lệnh để giải nén
  - $ decompress compressed_file [out_file]