ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Báo Cáo Đồ Án

CÂY HUFFMAN VÀ MÃ HÓA DỮ LIỆU

Đinh Thành Việt **18126037**

Đinh Viết Trung **18126035**

Nguyễn Sang **18126029**

| Môn học: Lý Thuyết Đồ Thị|

Thành phố Hồ Chí Minh – 202

A. Đặt vấn đề:

Để mã hóa các ký hiệu (ký tự, chữ số,...) ta thay chúng bằng các xâu nhị phân, được gọi là từ mã của ký hiệu đó. Chẳng hạn bộ mã ASCII, mã hóa cho 256 ký hiệu là biểu diễn nhị phân của các số từ 0 đến 255, mỗi từ mã gồm 8 bit.

VD: Trong ASCII:

Từ mã của ký tự "a" là 1100001.

Từ mã của ký tự "A" là 1000001.

Trong cách mã hóa này các từ mã của tất cả 256 ký hiệu có độ dài bằng nhau (mỗi từ mã 8 bit). Ta gọi là mã hóa với độ dài không đổi.

Tuy nhiên, khi mã hóa một tài liệu có thể không sử dụng đến tất cả 256 ký hiệu. Ví dụ như trong tài liệu, chữ cái “a” có thể xuất hiện 99999999999 lần trong khi chữ cái “A” chỉ xuất hiện 3 – 4 lần.

⇨ Như vậy ta có thể không cần dùng đủ 8 bit để mã hóa cho một ký hiệu. Ta hoàn toàn có thể quy định độ dài (số bit) dành cho mỗi ký hiệu có thể khác nhau, ký hiệu nào xuất hiện nhiều lần thì nên dùng số bit ít, ký hiệu nào xuất hiện ít thì có thể mã hóa bằng từ mã dài hơn

⇨ ***Ý tưởng:***

- Giảm số bit để biểu diễn 1 ký tự
- Dùng chuỗi bit ngắn hơn để biểu diễn ký tự xuất hiện nhiều
- Sử dụng mã tiền tố để phân cách các ký tự

⇨ Cây HuffMan ra đời.

B. Giải quyết vấn đề:

I. Mã hóa huffman:

Là một thuật toán mã hóa dùng để nén dữ liệu. Nó dựa trên bảng tần suất xuất hiện các ký tự cần mã hóa để xây dựng một bộ mã nhị phân cho các ký tự đó sao cho dung lượng (số bit) sau khi mã hóa là nhỏ nhất.

II. Cây Huffman:

- Là cây nhị phân, mỗi nút chứa ký tự và trọng số (tần suất của ký tự đó).
- Mỗi ký tự được biểu diễn bằng 1 nút lá (tính tiền tố).

- Nút cha có tổng ký tự, tổng trọng số của 2 nút con.
- Các nút có trọng số, ký tự tăng dần từ trái sang phải.
- Các nút có trọng số lớn nằm gần nút gốc.
- Các nút có trọng số nhỏ nằm xa nút gốc hơn.

III. Thuật Toán Tham Lam:

Ta sẽ xây dựng cây huffman dựa vào thuật toán tham lam:

- B1: Tạo N cây, mỗi cây chỉ có một nút gốc, mỗi nút gốc chỉ chứa một kí tự và trọng số (tần suất của ký tự đó). (N = số ký tự)

- B2: Lặp lại thao tác sau cho đến khi chỉ còn 1 cây duy nhất:

  + Ghép 2 cây con có trọng số gốc nhỏ nhất thành 1 nút cha, có tổng ký tự, tổng trọng số trọng số của 2 nút con.

  + Xóa các cây đã duyệt. + Điều chỉnh lại cây nếu vi phạm tính chất.

IV. Mã Huffman :

- Là chuỗi nhị phân được sinh ra dựa trên cây Huffman.
- Mã Huffman của ký tự là đường dẫn từ nút gốc đến nút lá đó.
  - Sang trái ta được bit 0
  - Sang phải ta được bit 1
- Có độ dài biến đổi (tối ưu bảng mã).
  - Các ký tự có tần suất lớn có độ dài ngắn.
  - Các ký tự có tần suất nhỏ có độ dài dài hơn.

⇨ Sau khi xây dựng cây huffman, ta sẽ xây dựng mã huffman dựa vào các bước trên.

V. Thuật toán điều chỉnh:

- Nếu trọng số nút hiện hành > nút lân cận từ phải sang trái, từ dưới lên trên -> Vi phạm.
- Tìm nút xa nhất có trọng số cao nhất < trọng số nút vi phạm -> Hoán đổi vị trí.

VI. Phân loại:

- Nén Tĩnh (Static Huffman)

Lợi ích	Hạn chế
• Hệ số nén tương đối cao. • Phương pháp thực hiện tương đối đơn giản. • Đòi hỏi ít bộ nhớ.	• Mất 2 lần duyệt file khi nén. • Phải lưu trữ thông tin giải mã vào file nén. • Phải xây dựng lại cây Huffman khi giải nén.

⇨ Adaptive Huffman ra đời:
• Khắc phục nhược điểm của Static Huffman.
• Đầu đọc vừa duyệt, vừa cập nhật cây Huffman, vừa xuất kết quả ra file nén theo thời gian thực.

- Nén Tĩnh (Static Huffman): dùng thuật tóan tham lam
- Nén Động (Adaptive Huffman): dùng thuật toán điều chỉnh. Tuy nhiên, cách xây dựng cây huffman sẽ khác một xíu:
  - Vẫn giống cây static huffman ở chỗ: Trọng số của nút bên trái phải nhỏ hơn nút bên phải, nhỏ hơn nút cha.
  - Trọng số nút cha bằng tổng trọng số 2 nút con
    Tuy nhiên, nó có thêm thanh phần nút NYT (not yet transmitted) có trọng số luôn = 0, dùng dể nhận biết ký tự đã xuất hiện trong cây hay chưa.

C.Ví dụ:

Static Huffman:

Ta có chuỗi sau:

string =
“ABABBCBBDEEEABABBAEEDDCCABBBCDEEDCBCCCCDBBBCAAA”

Ta có:

Số ký tự: N = 47

***

Về việc tinh Tần Suất, ta có thể lấy số lần xuất hiện của ký tự chia tổng ký tự.

Vd: ‘A’ xuất hiện 9 lần => Tần Suất = 9/47.

Tuy nhiên, ở ví dụ này ta lấy Tần Suất = với số lần xuất hiện (cho dễ hình dung) cũng không ảnh hưởng gì (vì nếu tinh theo cách trên thì đều lấy số lần xuất hiện chia cho 47)

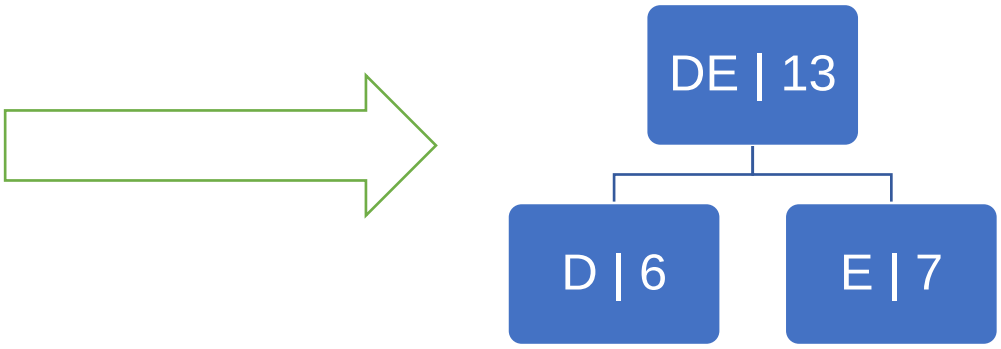
Ký Tự	Tần Suất
A	9
B	15
C	10
D	6
E	7

Ta sẽ sắp xếp lại theo Tần Suất

Ký Tự	Tần Suất
B	15
C	10
A	9
E	7
D	6

Nhìn vào bảng sau khi sắp xếp lại theo thứ tự tần suất, ta thấy E và D có tần suất xuất hiện ít nhất. Dựa vào thuật toan tham lam, ta xây dựng được cây như sau:

Ký Tự	Tần Suất
B	15
C	10
A	9
E	7
D	6

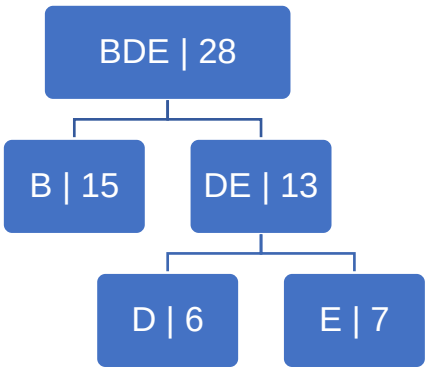


Sau đó, ta có bảng mới như sau và làm tương tự cho cây gồm 2 node: A và C (do có tần suất nhỏ nhất):

Ký Tự	Tần Suất
B	15
DE	13
C	10
A	9

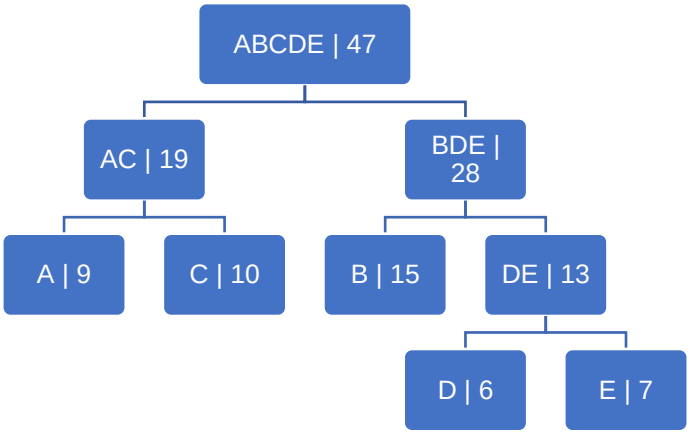
Ta có bảng mới như sau. Khi này, ta xây dựng cây từ B và DE do có tần suất bé nhất

Ký Tự	Tần Suất
AC	19
B	15
DE	13

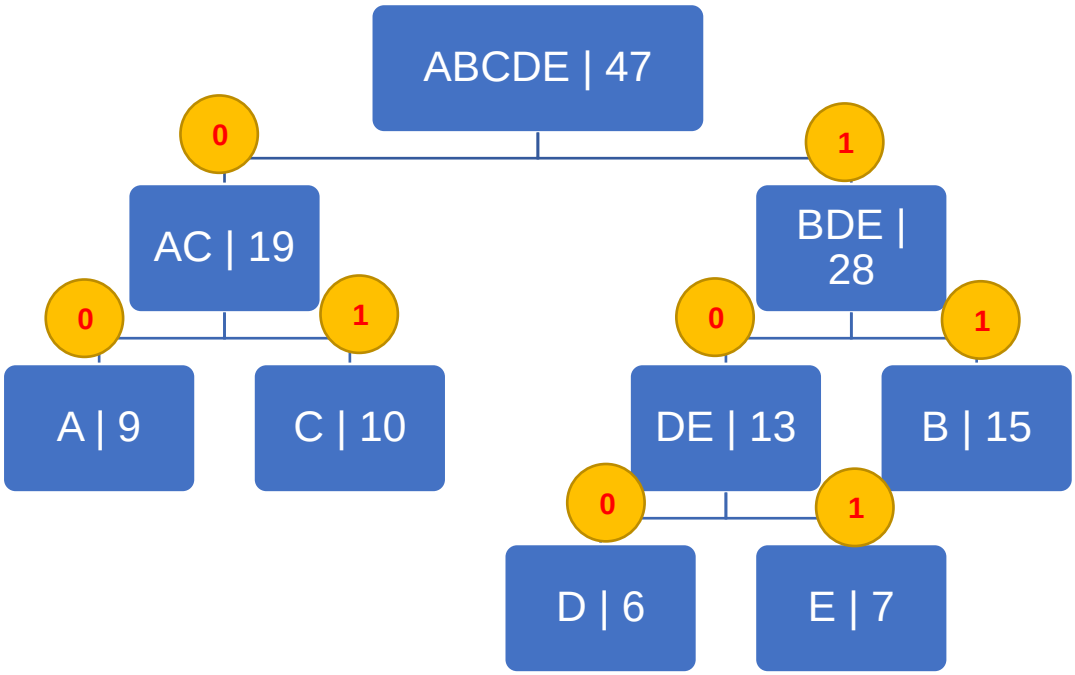


Ta có bảng mới như sau: khi này, chỉ cò AC và BDE. Ta xây dựng thành 1 cây duy nhất

Ký Tự	Tần Suất
BDE	28
AC	19



Khi xây dựng thành 1 cây huffman hoan chỉnh, ta bắt đầu đánh 0, 1 cho các node lá



Theo quy tắc mã huffman:

Chuỗi ban đầu:

string = “ABABBCBBDEEEABABBAEEDDCCABBBCDEEDCBCCCCDBBBCAAA”

Sau khi nén:

String output = “00110011110111111001011011010011001111001011011001000101001111110110010110110001110101010110011111101000000”

Adaptive Huffman

Vd: string = “AABBB”

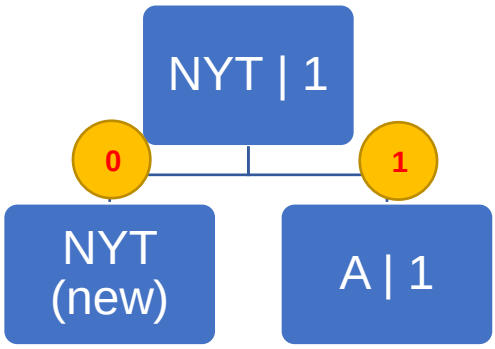
//giải thích ký hiệu: **A: thêm vào cây**

A: đã dc thêm vào

AABBB

// ký tự chưa tồn tại (A in đậm)

Ta có cây như sau:

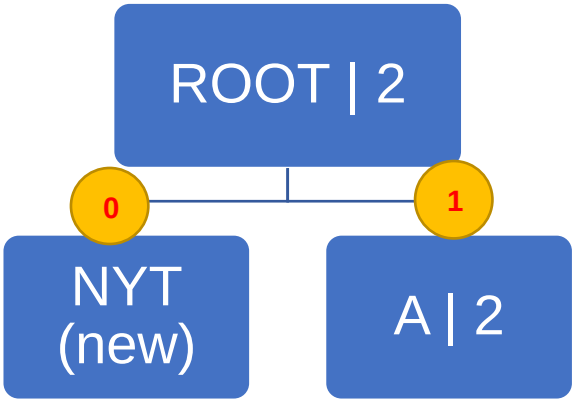


=>String Output = 01000011

AABBB

// ký tự đã tồn tại (do đã thêm A vào trước)

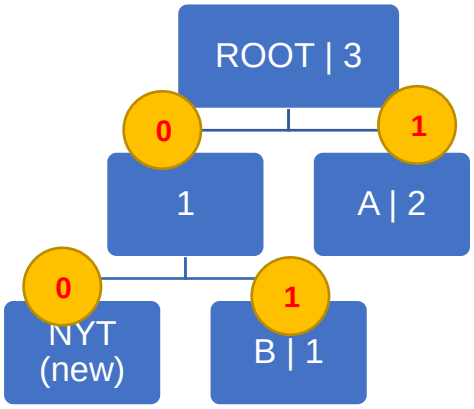
String Output = 01000011 1



AABBB

//ký tự chưa tồn tại (B chưa dc thêm từ trước)

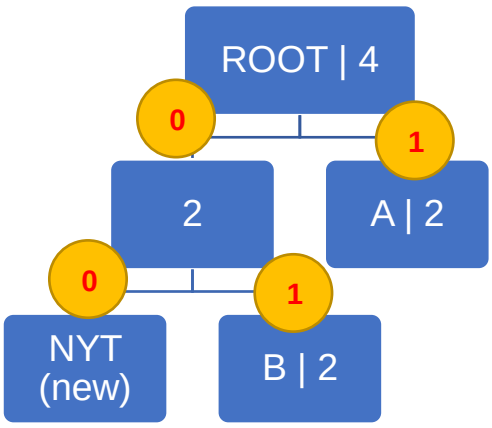
String Output = 010000111 0 01000010



AABBB

//ký tự đã tồn tại

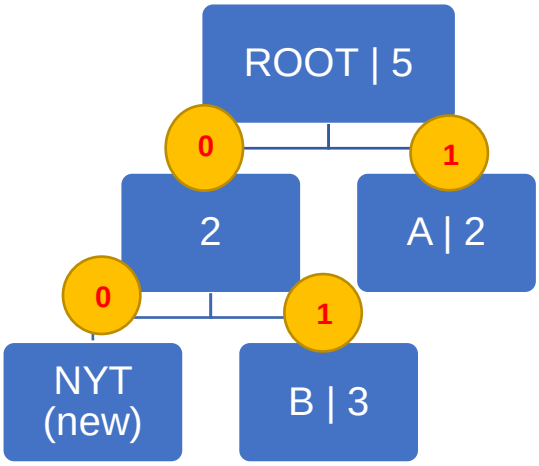
String Output = 0100001110010000100 1



AABBB

//ký tự đã tồn tại

String Output = 0100001110010000100101



Demo (bằng python):

0. *Thư viện hỗ trợ trong python:*

import heapq

from collections import defaultdict

from dahuffman import HuffmanCodec

1. *Đầu tiên, ta xây dựng hàm chuyển ký tự sang mã nhị phân. Từ đó, ta lại tiếp tục xây dựng hàm chuyển mã nhị phân sang Ascii và ngược lại.*

```
# Hàm chuyển kí tự sang mã nhị phân
def decode(l):
	result = ""
	for i in l:
		t = str((convert_string_to_binary(i))).replace(" ","")
		t = t.replace(",","")
		t = t.replace("[","")
		t = t.replace("]","")
		result += t + " "
	return result

# Hàm đổi ký tự Ascii sang mã nhị phân :
def convert_string_to_binary(ch):
	n = ord(ch)
	tmp = f'{n:08b}'
	b = [0]*(8)
	for i in range(0,len(tmp)):
		b[i] = int(tmp[i])
	return b

# Hàm đổi mã nhị phân sang ký tự Ascii
def convert_binary_to_string(b):
	s = 0
	n = len(b)-1
	i = 0
	while(n > -1):
		if(b[n] != 0):
			s += pow(2,i)
		i += 1
		n -= 1
	return s

def list_dict(list_d):
	d = defaultdict(int)
	for i in list_d:
		d[i] += 1
	return d
```

2. *Ta xây dựng hàm mã hóa Huffman:*

```
def encodingHuffman(data):
    list_frequency = list_dict(data)
    heap = [[weight, [symbol, '']] for symbol, weight in list_frequency.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        min_left = heapq.heappop(heap)
        min_right = heapq.heappop(heap)
        for bit in min_left[1:]:
            bit[1] = '0' + bit[1]
        for bit in min_right[1:]:
            bit[1] = '1' + bit[1]
        heapq.heappush(heap, [min_left[0] + min_right[0]] + min_left[1:] +min_right[1:])
    heap_temp = heapq.heappop(heap)[1:]
    result = sorted(heap_temp)
    return result,list_frequency
```

3. *Ta xây dựng hàm Encode và Decode:*

```
    # Hàm nén và giải nén bằng cây Huffman
    def Encoding_Decoding(file_input,file_output):
        from dahuffman import HuffmanCodec
        inp = open(file_input,"r")
        data = inp.read()
        inp.close

        list_frequency = list_dict(data)
        codec = HuffmanCodec.from_data(list_frequency)

        encoded = codec.encode(data)
        f = open("output_bin.txt","wb")
        f.write(encoded)
        f.close()

        f1 = open("output_bin.txt","rb")
```

```
bi = f1.read()
f1.close()

t = codec.decode(bi)
result = ""
for i in t:
        result += i
f_out = open(file_output, "w")
f_out.write(result)
f_out.close()
```

Nguồn:

http://dulieu.tailieuhoctap.vn/books/cong-nghe-thong-tin/co-so-du-lieu/file_goc_768105.pdf

https://en.wikipedia.org/wiki/Huffman_coding