THÔNG TIN NHÓM

Group ID: 54869696	
Tên thành viên	MSSV
Lê Kiệt	19120554
Nguyễn Phát Minh	19120586
Trần Thị Khánh Duyên	19120496
Phạm Sơn Nam	19120596

Tiến độ công việc	
Công việc	Tiến độ hoàn thành
LZW Encode	100%
LZW Decode	100%

I. Thuật toán nén – giải nén LZW

1. Thuật toán nén LZW (LZW encode phase)

a) Mô tả thuật toán

- *Bước 1*: Khởi tạo bảng table chứa tất cả chuỗi có 1 ký tự (Bảng mã ASCII với 256 ký tự bắt đầu từ vị trí 0-255)

```
map<string, int> table;
```

- *Bước 2*: Bắt đầu từ ký tự đầu tiên, thuật toán xét liệu chuỗi gồm ký tự đó và ký tự tiếp theo đã thuộc bảng chưa.
    - Nếu chuỗi chưa thuộc bảng thêm chuỗi vào bảng (bắt đầu từ vị trí 256 trở đi) và in vào file giá trị ascii của ký tự đầu tiên
    - Nếu chuỗi đã thuộc bảng xét đến chuỗi gồm các ký tự đã xét và ký tự tiếp theo (tìm chuỗi con dài nhất)
- *Bước 3*: Cứ tiếp tục như vậy cho đến khi hết chuỗi.

b) Ví dụ từng bước

Encode chuỗi “abcabd”

Current	Next	Output	Add to dictionary
a (65)	b (66)	a(65)	ab(256)
b (66)	c (67)	b(66)	bc(257)
c (67)	a (65)	c(67)	ca(258)
a (65)	b (66)	“ab” đã thuộc bảng	
ab (256)	d (68)	ab(256)	abd(259)
d (68)	-	d(68)	-

c) Thông tin hàm

```
pair<vector<int>, map<int, string>> compress(string filePath) { ... }
```

Hàm trên nhận đường dẫn của file đầu vào (.txt) nén dữ liệu file đầu vào dựa vào thuật toán trên và trả ra pair<Type1,Type2>. Trong đó

- Type1 = vector<int>: là vector chứa các mã (code) có kiểu dữ liệu dạng int sau khi mã hóa file đầu vào

- Type2 = map<string, int>: trả về bảng (table), bảng này có thể đã được cập nhật thêm vài ô nhớ mới

2. Thuật toán giải nén LZW (LZE decode phase)

a) Mô tả thuật toán

- *Bước 1*: đọc mỗi số (code) từ file .lzw và lưu vào biến: vector<int> codeVec
- *Bước 2*: Khởi tạo bảng table chứa tất cả chuỗi có 1 ký tự (Bảng mã ASCII với 256 ký tự bắt đầu từ vị trí 0-255)

```
map<string, int> table;
```

- *Bước 3*:

  + Decode ptu đầu tiên trong codeVec vào biến [string s]

  + Khởi tạo biến [string result = s] để lưu lại kết quả cuối cùng sau khi decode

  + Khởi tạo biến [string entry] để lưu table[code]/chuỗi đang dịch

- *Bước 4*: với mã code tiếp theo trong codeVec (nếu còn)
  - Nếu mã code thuộc bảng entry = table[code]
  - Nếu mã code ko thuộc bảng entry = s + s[0].
  - thêm (s + entry[0]) vào cuối table. Có lệnh này do bản chất của thuật toán giải mã LZW (LZW decoding algorithm) là “cô đọng”/ ”gộp” các kí tự bị trùng trong các lần lặp trước thành 1 mã code mới (>255) trong table
  - cập nhật s = entry
  - cập nhật result = result + entry
- *Bước 5*: lặp lại bước 4 tới khi duyệt hết codeVec

Mã giả cho bước 4 và bước 5

```
while (nếu còn mã code để decode trong codeVec) {
    int code = mã code tiếp theo trong codeVec

    if (mã code có hiện diện trong table)
        entry = table[code]
    else, table ko chứa mã code đó
```

entry = s + s[0];

thêm (s + entry[0]) vào cuối table
cập nhật s = entry
cập nhật result = result + entry
}

b) *Ví dụ từng bước*

Xét ví dụ: codeVec = [87, 89, 83, 42, 256, 71, 256, 258, 262]

Kết quả: WYS*WYGWYS*WYS

Mã code đang duyệt (code)	table	entry	Sau khi decode (s)	Chuỗi result
87	_	_	W	W
89	table[256] = “WY”	Y	Y	WY
83	table[257] = “YS”	S	S	WYS
42	table[258] = “S*”	*	*	WYS*
256	table[259] = “*W”	WY	WY	WYS*WY
71	table[260] = “WYG”	G	G	WYS*WYG
256	table[261] = “GW”	WY	WY	WYS*WYGWY
258	table[262] = “WYS”	S*	S*	WYS*WYGWYS*
262	table[263] = “S*W”	WYS	WYS	WYS*WYGWYS*WYS

c) *Thông tin hàm*

```
pair<string, map<int, string>> decompress(string filePath) { ... }
```

Hàm trên nhận đường dẫn của file đầu vào (.lzw) □ giải nén dữ liệu file đầu vào dựa vào thuật toán trên và trả ra pair<Type1,Type2>.

Trong đó

- Type1 = string: chuỗi sau khi decode

- Type2 = map<int, string>: trả về bảng (table), bảng này có thể đã được cập nhật thêm vài ô nhớ mới

3. Ưu, nhược điểm

- Ưu điểm:
  - LZW là phương pháp nén dữ liệu lossless (dữ liệu không bị mất đi trong quá trình nén)
  - LZW đạt hiệu quả tối đa khi được sử dụng để nén những dữ liệu có thông tin được lặp lại nhiều lần (văn bản, hình trắng đen, GIF file, TIFF file …)
  - Thời gian nén dữ liệu nhanh, LZW có thể nén dữ liệu với chỉ một lần duyệt qua.
  - Bên nhận có thể xây dựng bảng mã để decode mà không cần bên gửi phải gửi kèm theo trong tập tin nén
- Nhược điểm:
  - Tùy vào chiều dài đầu vào mà table có thể có kích thước rất lớn
  - Nếu file chứa quá ít các chuỗi kí tự dc lặp lại giảm tính hiệu quả của thuật toán
  - Một số entry dc tạo mới trong quá trình decode có thể sẽ không bao giờ dc sử dụng lãng phí ô nhớ

4. Ứng dụng thực tế của thuật toán LZW

- Được sử dụng trong định dạng GIF và định dạng TIFF.
- Có thể dùng trong việc tạo PDF.

II. Thuật toán nén – giải nén Huffman

1. Thuật toán nén Huffman

a) Mô tả thuật toán

- Giải thuật nén tĩnh Huffman bao gồm:

+ ***Bước 1*:** Duyệt file input, hình thành bảng thống kê tần suất xuất hiện của mỗi kí tự trong file. (xây dựng cấu trúc min-heap)

+ *Bước 2*: Tiến hành xây dựng cây nhị phân Huffman theo bảng vừa thống kê (đảm bảo cấu trúc min-heap)

+ *Bước 3*: Tạo ra mã Huffman ứng với từng kí tự dựa vào cây Huffman vừa tạo

+ *Bước 4*: Thay thế toàn bộ kí tự bằng mã Huffman tương ứng

+ *Bước 5*: Lưu cây Huffman (dùng để giải nén). Xuất file output

b) Ví dụ từng bước

Chuỗi kí tự cần nén:

INPUT = “DATASTRUCTURECHALLENGE”

Bước 1: hình thành bảng thống kê tần suất xuất hiện của các kí tự

Kí tự	D	H	G	N	S	C	L	R	U	A	E	T
Tần số	1	1	1	1	1	2	2	2	2	3	3	3

Bước 2: Xây dựng cây Huffman.

Chú thích:

+ **màu vàng**: chuẩn bị tạo thành nhánh trong cây huffman

+ **màu đỏ**: đã thành 1 nhánh trong cây huffman

2.1:

G	N	S	C	L	R	U	DH	A	E	T
1	1	1	2	2	2	2	2	3	3	3

2.2:

S	C	L	R	U	GN	DH	A	E	T
1	2	2	2	2	2	2	3	3	3

2.3:

L	R	U	GN	DH	A	E	T	SC
2	2	2	2	2	3	3	3	3

2.4:

U	GN	DH	A	E	T	SC	LR
2	2	2	3	3	3	3	4

2.5:

DH	A	E	T	SC	LR	UGN
2	3	3	3	3	4	4

2.6:

E	T	SC	LR	UGN	DHA
3	3	3	4	4	5

2.7:

SC	LR	UGN	DHA	ET
3	4	4	5	6

2.8:

UGN	DHA	ET	SCLR
4	5	6	7

2.9:

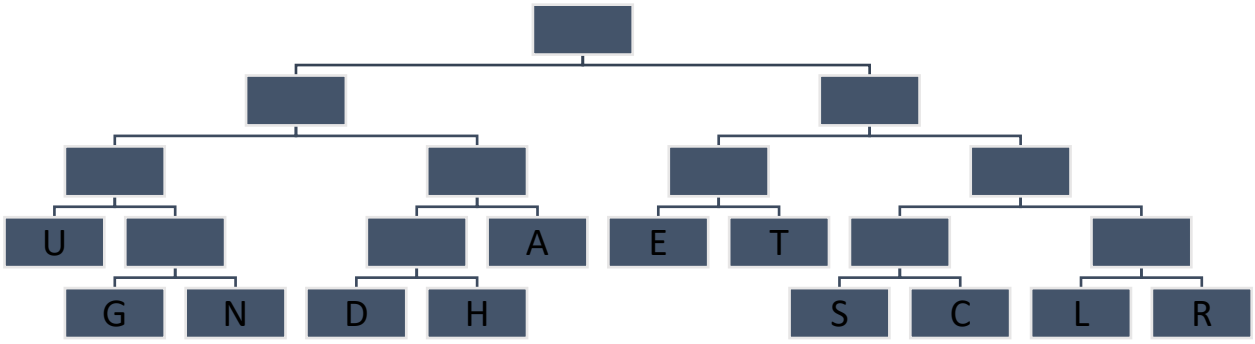
ET	SCLR	UGNDHA
6	7	9

2.10:

UGNDHA	ETSCLR
9	13

2.11:

UGNDHAETSCLR
22



⇨ Cây Huffman tạo thành

Bước 3**:** Tạo ra mã Huffman ứng với từng kí tự

Duyệt cây vừa tạo: sang trái tương ứng là “0”, sang phải tương ứng là “1”

KÍ TỰ	BINARY CODES
D	0100
H	0101
G	0010
N	0011
S	1100
C	1101
L	1110
R	1111
U	000
A	011
E	100
T	101

Bước 4: Thay thế toàn bộ kí tự bằng mã Huffman tương ứng

INPUT: “DATASTRUCTURECHALLENGE” (8*22 = 176 bits)

OUTPUT:
“0100|011|101|011|1100|101|1111|0001101|101|000|1111|100|1101|0101|011|1110
|1110|100|0011|0010|100” (77 bits)

Bước 5: Xuất file output

OUTPUT =
“0100|011|101|011|1100|101|1111|0001101|101|000|1111|100|1101|0101|011|1110
|1110|100|0011|0010|100” (77 bits)

2. Thuật toán giải nén Huffman (Huffman decode phase)

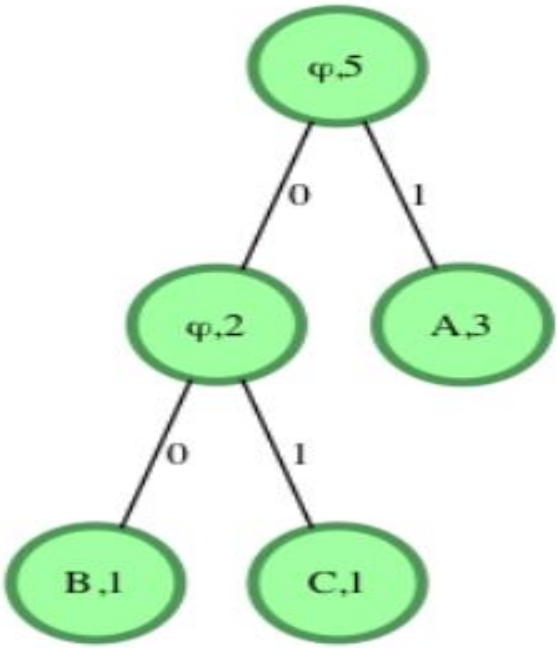
a) Mô tả thuật toán

Từ thuật toán nén Huffman, ta có được cây nhị phân

- *Bước 1*: Dò từng số trong chuỗi số đã nén và đi dọc theo cây nhị phân từ root đến leaf
- *Bước 2*: Tìm đường đi:
    - Nếu là số 0 di chuyển sang bên trái cây và lưu số vào chuỗi nhỏ
    - Nếu là số 1 di chuyển sang bên phải cây và lưu số vào chuỗi nhỏ
- *Bước 3*: Nếu node hiện tại đang xét
    - Là leaf thì dừng việc dò và lấy kí tự ghi trong leaf ra để xuất
    - Không phải leaf thì quay lại bước tìm đường đi
- *Bước 4*: Quay trở lại root và tiếp tục dò đến kí tự tiếp theo

b) Ví dụ từng bước

Ta có chuỗi ABACA sau khi nén là: s = **1001011** & cây nhị phân tìm kiếm là



- s[0] = 1: root □ sang phải, lúc này gặp phải leaf ta xuất ra kí tự của leaf : **A**
- s[1] = 0: root □ trái. Do chưa đụng leaf nên xét s[2] = 0 □ tiếp tục sang trái, được kí tự: **B**
- s[3] = 1: root □ phải, dc kí tự: **A**
- s[4] = 0: root □ trái. Do chưa đụng leaf nên xét s[5] = 1 □ tiếp tục sang phải, dc kí tự: **C**
- s[6] = 1: root □ phải, dc kí tự: **A**

  ⇨ Vậy chuỗi được dịch là: **ABACA**

3. Ưu – nhược điểm

- Ưu điểm:

  + Yêu cầu ít bộ nhớ

  + Phương pháp thực hiện thuật toán Huffman tương đối đơn giản, dễ thực hiện và nén không mất dữ liệu (lossless)

- Nhược điểm: tương đối chậm, chuyển tiếp dựa trên mô hình thống kê dữ liệu, khó giải mã do độ dài mã khác nhau

4. Ứng dụng thực tế của thuật toán Huffman

- Huffman được sử dụng rộng rãi trong tất cả các định dạng nén chính mà bạn có thể gặp từ GZIP, PKZIP (winzip, v.v.) và BZIP2, đến các định dạng hình ảnh như JPEG và PNG. Zip có lẽ là công cụ nén được sử dụng rộng rãi nhất sử dụng Huffman Encoding làm cơ sở

III. NGUỒN THAM KHẢO:

Đề tài Lempel - Ziv Encoding - Luận văn, đồ án, đề tài tốt nghiệp (luanvan.net.vn)

Lempel-Ziv-Welch Compression Algorithm - Tutorial - YouTube

Huffman Coding Algorithm | Studytonight

Huffman Coding Compression Algorithm - Techie Delight