Vietnam National University – Ho Chi Minh city
University of Science
Faculty of Computer Science

COURSE PROJECT

DATA STRUCTURES AND ALGORITHMS

PROJECT of SEMESTER 1
2020 – 2021

DATA COMPRESSION

Class: 19CLC10
Instructor: Bùi Huy Thông

Full name	Students ID	Work	Accomplished
Trương Gia Đạt	19127017	_Lập trình mã hóa Huffman	100%
Lê Vũ Anh Duy	19127375	_Tìm hiểu về mã hóa Huffman	100%
Phan Trung Hiếu	19127404	_Lập trình thuật toán nén Lempel-Ziv-Welch	100%
Nguyễn Huy Anh Thư	19127569	_Tìm hiểu về thuật toán nén Lempel-Ziv-Welch + viết báo cáo	100%

Ho Chi Minh , 2020

GIỚI THIỆU

Giới thiệu về thuật toán nén:

- Các dữ liệu chưa được nén sẽ chiếm rất nhiều dung lượng, điều này không tốt cho những ổ cứng có giới hạn không gian chứa và trong trường hợp Internet có tốc độ tải không ổn định hoặc chậm.
  Ví dụ: Với một video chưa được nén thì 1 phút tương đương với hơn 1GB. Vậy làm sao để đưa một video dài 2 tiếng vào một chiếc đĩa Bluray chỉ có sức chứa tối đa 25GB ??? => Ta dùng đến sự trợ giúp của việc nén dữ liệu để chuyển định dạng thông tin sử dụng ít bit hơn cách thể hiện ở dữ liệu gốc.
- Nén dữ liệu được chia thành hai loại: nguyên vẹn (lossless) và bị mất dữ liệu (lossy)
- Chủ đề của bài báo cáo sẽ đi về hai thuật toán nén dữ liệu nguyên vẹn*: mã hóa Huffman* và *thuật toán nén Lempel-Ziv-Welch (LZW).*

TÌM HIỂU

1. Mã hóa Huffman

_Huffman Coding là một kỹ thuật nén dữ liệu để giảm kích thước của tập tin mà không làm mất bất kỳ chi tiết nào. Nó được phát triển lần đầu tiên bởi David Huffman.

_Huffman Coding thường hữu ích để nén dữ liệu trong đó có các ký tự xuất hiện thường xuyên.

Encoding:

_Ví dụ: cho một chuỗi `string str = "BCAADDDCCACACAC";`

_Mỗi ký tự trong chuỗi sẽ có kích thước là 1 byte ⇔ 8 bit và chiều dài chuỗi là 15. Vậy tổng kích thước chuỗi = 8 * 15 = 120 bits.

_ Sử dụng kỹ thuật Huffman Coding, chúng ta có thể nén chuỗi thành kích thước nhỏ hơn. Các kỹ thuật cần thiết trong thuật toán Huffman:

+Xây dựng cây (tree).

+Min – heap property.

+Priority Queue.

Bước 1: Tính số lần xuất hiện của các ký tự.

Character	Frequency
B	1
C	6
A	5
D	3

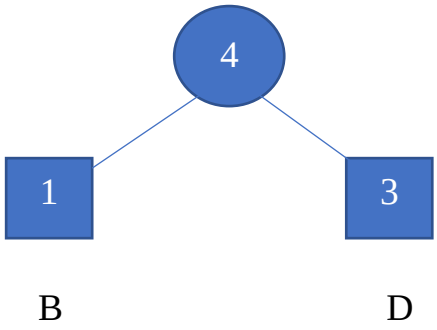
Bước 2: Sắp xếp các ký tự tăng dần theo Frequency ở Bước 1 sẽ đưa vào Priority Queue Q.

Character	Frequency
B	1

D	3
A	5
C	6

Bước 3: Mỗi ký tự riêng biệt sẽ là mỗi node lá.

Bước 4: Tạo một node rỗng root. Ký tự có Frequency nhỏ nhất sẽ là con trái của root và ký tự có Frequency nhỏ thứ hai sẻ là con phải của root. Gía trị của root sẽ bằng tổng Frequency của hai node con của nó.

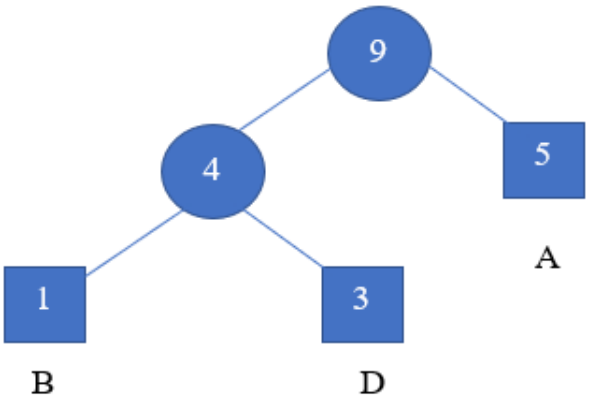


Bước 5: Xóa 2 nút con sẽ sử dụng khỏi Q và thêm giá trị của root vào.

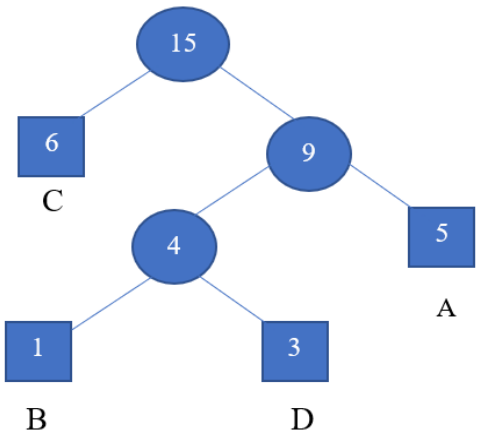
Bước 6: Chèn node root vào cây (tree).

Bước 7: Lập bước 3 đến 5 cho tất cả các ký tự.

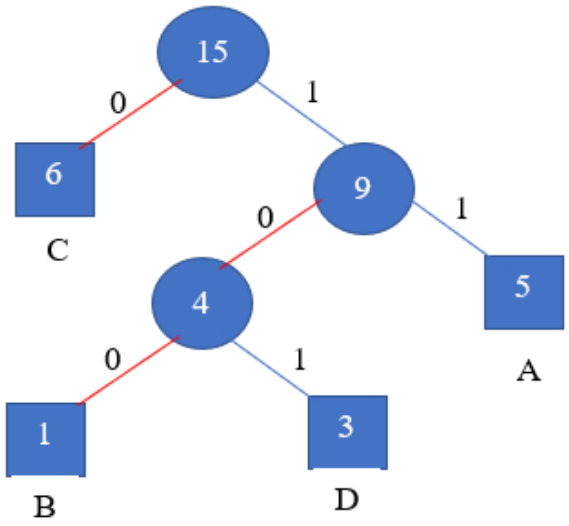
4	5	6
root	A	C



6	9
C	root



Bước 8: Với mỗi node, gán 0 cho cạnh bên trái và 1 cho cạnh bên phải.



Character	Frequency	Encoding	Size
A	5	11	5 * 2 = 10
B	1	100	1 * 3 = 3
C	6	0	6 * 1 = 6
D	3	101	3 * 3 = 9
4 * 8 = 32 bits	15 bits		28 bits

➔ Encoded data of BCAADDDCCACACAC: 100011111011011010011011011O.

➔ Sau khi mã hóa chuỗi được cho, ta thu được kích thước mới là 32 + 15 + 28 = 75 bits so với kích thước ban đầu là 120 bits.

Độ phức tạp thuật toán: $O(nlogn)$. Với n là số lượng ký tự chuỗi.

_Việc lấy ra 2 ký tự có Frequency nhỏ lần lượt từ Priority Queue sẽ tốn 2 * (n – 1) nên độ phức của việc này là $O(logn)$.

Decoding

Bước 1: Bắt đầu ở node root và làm các bước dưới đây đến khi node lá được tìm thấy.

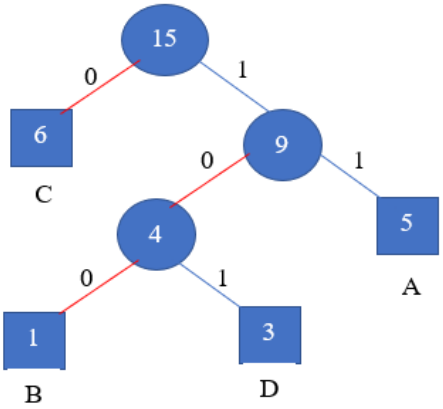
Bước 2:

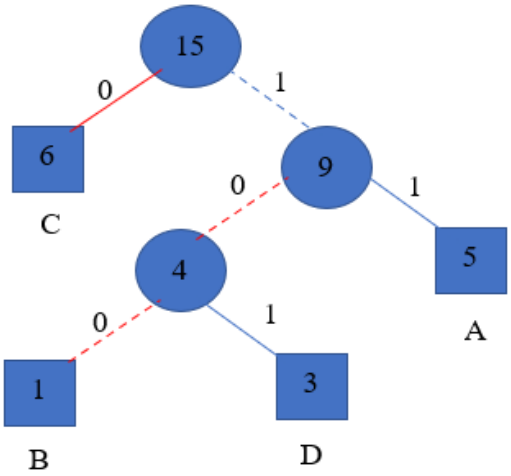
+Nếu bit hiện tại là 0, di chuyển sang node trái của cây (tree).

+Nếu bit hiện tại là 1, di chuyển sang node phải của cây (tree).

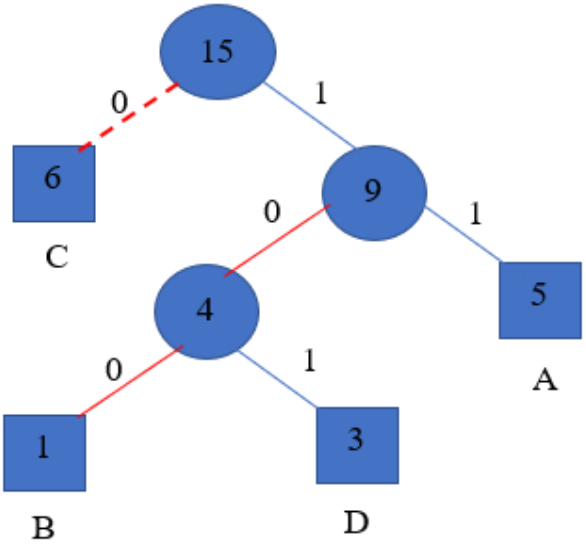
Bước 3: Di chuyển đến khi chạm node lá thì dừng lại rồi in ra màn hình. Lặp lại bước 1 đến khi giải mã xong dữ liệu.

Ví dụ: cho dữ liệu mã hóa **100011111011011010011011011O** và cây nhị phân như hình dưới:

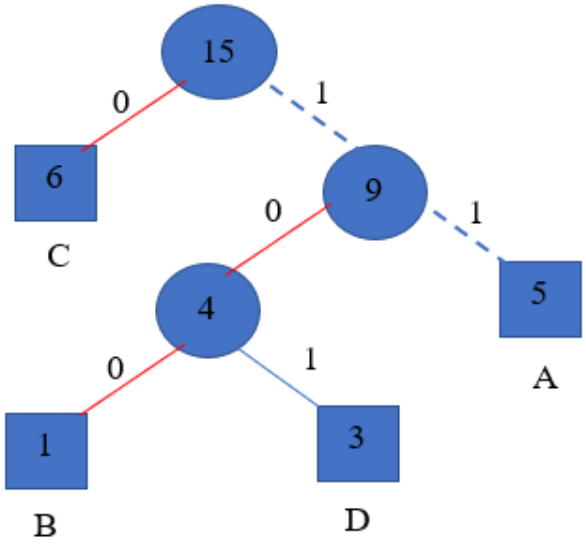




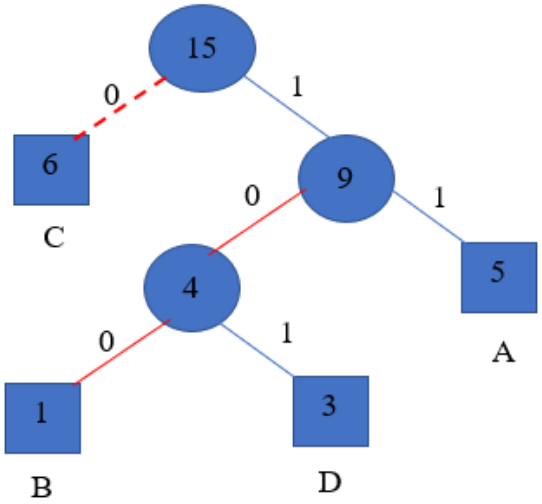
100011111011011010011011011O -> B



100011111011011010011011011O -> BC



100011111011011010011011011O -> BCA



100011111011011010011011011O -> BCA…..CACAC

Ưu điểm:

_Kích thước dữ liệu được mã hóa nhỏ hơn nhiều so với dữ liệu thuần ban đầu.

_Phương pháp thực hiện tương đối đơn giản.

_Đòi hỏi ít bộ nhớ.

Nhược điểm:

_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.

Ứng dụng:

_Mã hóa Huffman được sử dụng trong các định dạng nén thông thường như GZIP, BZIP2, PKZIP,…

_Để truyền văn bản và fax.

2. Thuật toán nén Lempel-Ziv-Welch

_Thuật toán nén Lempel-Ziv-Welch (gọi tắt là thuật toán LZW) là một trong số rất nhiều thuật toán được sử dụng để nén được phát minh bởi Lempel - Zip và Welch. Nó hoạt động đựa trên một ý tưởng rất đơn giản là người mã hoá và người giải mã cùng xây dựng bảng mã. Bảng mã này không cần được lưu kèm với dữ liệu trong quá trình nén, mà khi giải nén, người giải nén sẽ xây dựng lại nó.

Encoding:

*****Mã từ 0 đến 255 miêu tả 1 dãy ký tự thay thế cho ký tự 8-bit tương ứng.

Mã từ 256 đến 4095 được tạo bên trong 1 từ điển cho trường hợp lặp chuỗi trong dữ liệu.

* Phần quan trọng nhất của phương pháp nén này là phải tạo một mảng rất lớn dùng để lưu giữ các xâu kí tự đã gặp (Mảng này được gọi là "Từ điển"). Khi các byte dữ liệu cần nén được đem đến, chúng liền được giữ lại trong một bộ đệm chứa (Accumulator) và đem so sánh với các chuỗi đã có trong "từ điển". Nếu chuỗi dữ liệu trong bộ đệm chứa không có trong "từ điển" thì nó được bổ sung thêm vào "từ điển" và chỉ số của chuỗi ở trong "từ điển" chính là dấu hiệu của chuỗi. Nếu chuỗi trong bộ đệm chứa đã có trong "từ điển" thì dấu hiệu của chuỗi được đem ra thay cho chuỗi ở dòng dữ liệu ra.

_Khởi tạo một bảng để chứa các ký tự char riêng lẻ

_P = phần tử đầu tiên được đưa vào bảng

_Cho đến khi hết chuỗi được đưa vào:

+ C = phần tử tiếp theo được đưa vào bảng

+ Nếu P và C đã có sẵn trong bảng (tức không cần phải đụng đến bảng từ điển)

- P = P + C

+ Nếu không

- Xuất mã cho P (output the code for P)

+ Thêm P + C vào bảng (để xét nếu gặp lại P + C vì nó đã được tạo trong bảng từ điển)

- P = C

+ Kết thúc vòng lặp

+ Sau khi kết thúc thì xuất mã cho P

Ví dụ: Ta có chuỗi `WYS*WYGWYS*WYSWYSG`

Encoder Output		Bảng từ điển	
String	Codeword (ASCII)	Codeword	String
W	87	256	WY
Y	89	257	YS
S	83	258	S*
*	42	259	*W

- Ta có thể thấy sau W là Y -> WY đã tạo thành 1 ký tự mới mà không có trong bảng ASCII -> gán codeword là 256 (do)
- Ta có thể thấy sau Y là S -> YS đã tạo thành một ký tự mới mà không có trong bảng ASCII -> gán codeword +1 là 257
- Ta có thể thấy sau S là * -> S* đã tạo thành một ký tự mới mà không có trong bảng ASCII -> gán codeword +2 là 258

- Ta có thể thấy sau * là W -> *W đã tạo thành một ký tự mới mà không có trong bảng ASCII -> gán codeword +3 là 259

WY	256	260	WYG
G	71	261	GW
WY	256	262	WYS
S*	258	263	S*W

- Đến bảng này thì ta đã có 1 bảng (bao gồm các ký tự hầu hết các ký tự trong mảng để cho, nếu ta gặp những ký tự chưa từng xuất hiện như ký tự G ở ô thứ 2, thì ta thực hiện như bảng trên để cho ra đời 1 "ký tự mới" gồm 2 ký tự trong đó có bao gồm G) dùng để xét những "cụm ký tự" có từ 2 ký tự trở lên
- Ta có thể thấy WY đã có sẵn (ở bước trên) nên ta không cần phải xét W và Y mà có 1 lần có thể xét 1 cụm WY như 1 ký tự (giống như W, Y, ….) (áp dụng cho những ký tự sau trong trường hợp xét từ 2 ký tự trở lên). Từ WY ta có thể tạo ra 1 "ký tự mới" gồm 3 ký tự để thêm vào bảng "từ điển"
- Ta có thể thấy sau WY là G -> WYG đã tạo thành một ký tự mới mà không có trong bảng ASCII -> gán codeword +4 là 260
- Ta có thể thấy G là một ký tự và chưa từng xuất hiện -> Ta có thể thấy sau G là W -> GW đã tạo thành một ký tự mới mà không có trong bảng ASCII -> gán codeword +5 là 261
- Ta có thể thấy WY đã có sẵn (ở bước trên) nên ta không cần phải xét W và Y mà có 1 lần có thể xét 1 cụm WY như 1 ký tự (giống như W, Y, ….) (áp dụng cho những ký tự sau trong trường hợp xét từ 2 ký tự trở lên). Từ WY ta có thể tạo ra 1 "ký tự mới" gồm 3 ký tự để thêm vào bảng "từ điển"
- Ta có thể thấy sau WY là S -> WYS đã tạo thành một ký tự mới mà không có trong bảng ASCII -> gán codeword +6 là 262
- Ta có thể thấy S* đã có sẵn (ở bước trên) nên ta không cần phải xét S và * mà có 1 lần có thể xét 1 cụm S* như 1 ký tự (giống như W, Y, ….) (áp dụng cho những ký tự sau trong trường hợp xét từ 2 ký tự trở lên). Từ S* ta có thể tạo ra 1 "ký tự mới" gồm 3 ký tự để thêm vào bảng "từ điển"
- Ta có thể thấy sau S* là W -> S*W đã tạo thành một ký tự mới mà không có trong bảng ASCII -> gán codeword +7 là 263

WYS	262	264	WYSW
WYS	262	265	WYSG
G	71		

- Đến bảng này thì ta đã có 1 bảng (bao gồm các ký tự hầu hết các ký tự trong mảng để cho, nếu ta gặp những ký tự chưa từng xuất hiện thì ta thực hiện như bảng trên để cho ra đời 1 "ký tự mới" gồm n ký tự trong đó có bao gồm ký tự chưa từng xuất hiện)
- Ta có thể thấy WYS đã có sẵn (ở bước trên) nên ta không cần phải xét W và Y và S hay xét riêng lẻ từng cụm 2 ký tự + 1 ký tự mà có 1 lần có thể xét 1 cụm WYS như 1 ký tự (giống như W, Y, ….). Từ WYS ta có thể tạo ra 1 "ký tự mới" gồm 4 ký tự để thêm vào bảng "từ điển"
- Ta có thể thấy sau WYS là W -> WYSW đã tạo thành một ký tự mới mà không có trong bảng ASCII -> gán codeword +8 là 264
- Ta có thể thấy WYS đã có sẵn (ở bước trên) nên ta không cần phải xét W và Y và S hay xét riêng lẻ từng cụm 2 ký tự + 1 ký tự mà có 1 lần có thể xét 1 cụm WYS như 1 ký tự

(giống như W, Y, ….). Từ WYS ta có thể tạo ra 1 “ký tự mới” gồm 4 ký tự để thêm vào bảng “từ điển”

- Ta có thể thấy sau WYS là G -> WYSG đã tạo thành một ký tự mới mà không có trong bảng ASCII -> gán codeword +9 là 265

➔ ***Output:***

87	001010111
89	001011001
83	001010011
42	000101010
256	100000000
71	001000111
256	100000000
258	100000010
262	100000110
262	100000110
71	001000111

➔ Bây giờ chuỗi ký tự được lưu trữ dưới dạng 9 bit thay vì 8 bit

➔ Độ dài của chuỗi hiện tại là 99 bit thay vì 144 bit nếu ta không nén dữ liệu -> tiết kiệm được 31.25% không gian lưu trữ.

Decoding:

*****Ta có bảng ASCII:

0	<NUL>	32	<SPC>	64	@	96	`	128	Ä	160	†	192	¿	224	‡
1	<SOH>	33	!	65	A	97	a	129	Å	161	°	193	¡	225	·
2	<STX>	34	"	66	B	98	b	130	Ç	162	¢	194	¬	226	‚
3	<ETX>	35	#	67	C	99	c	131	É	163	£	195	√	227	„
4	<EOT>	36	$	68	D	100	d	132	Ñ	164	§	196	ƒ	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Ö	165	•	197	≈	229	Â
6	<ACK>	38	&	70	F	102	f	134	Ü	166	¶	198	Δ	230	Ê
7	<BEL>	39	'	71	G	103	g	135	á	167	ß	199	«	231	Á
8	<BS>	40	(	72	H	104	h	136	à	168	®	200	»	232	Ë
9	<TAB>	41	)	73	I	105	i	137	â	169	©	201	…	233	È
10	<LF>	42	*	74	J	106	j	138	ä	170	™	202		234	Í
11	<VT>	43	+	75	K	107	k	139	ã	171	´	203	À	235	Î
12	<FF>	44	,	76	L	108	l	140	å	172	¨	204	Ã	236	Ï
13	<CR>	45	-	77	M	109	m	141	ç	173	≠	205	Õ	237	Ì
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
15	<SI>	47	/	79	O	111	o	143	è	175	Ø	207	œ	239	Ô
16	<DLE>	48	0	80	P	112	p	144	ê	176	∞	208	–	240	
17	<DC1>	49	1	81	Q	113	q	145	ë	177	±	209	—	241	Ò
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	“	242	Ú
19	<DC3>	51	3	83	S	115	s	147	ì	179	≥	211	”	243	Û
20	<DC4>	52	4	84	T	116	t	148	î	180	¥	212	‘	244	Ù
21	<NAK>	53	5	85	U	117	u	149	ï	181	µ	213	’	245	ı
22	<SYN	54	6	86	V	118	v	150	ñ	182	∂	214	÷	246	ˆ
23	<ETB>	55	7	87	W	119	w	151	ó	183	∑	215	◊	247	˜
24	<CAN>	56	8	88	X	120	x	152	ò	184	∏	216	ÿ	248	¯
25	<EM>	57	9	89	Y	121	y	153	ô	185	π	217	Ÿ	249	˘
26	<SUB>	58	:	90	Z	122	z	154	ö	186	∫	218	/	250	˙
27	<ESC>	59	;	91	[	123	{	155	õ	187	ª	219	€	251	°
28	<FS>	60	<	92	\	124	|	156	ú	188	º	220	‹	252	¸
29	<GS>	61	=	93	]	125	}	157	ù	189	Ω	221	›	253	˝
30	<RS>	62	>	94	^	126	~	158	û	190	æ	222	fi	254	˛
31	<US>	63	?	95	_	127	<DEL>	159	ü	191	ø	223	fl	255	ˇ

Ví dụ: Ta có dãy: 001010111 001011001 001010011 000101010 100000000 001000111 100000000 100000010 100000110 100000110 001000111

➔ 87 89 83 42 256 71 256 258 262 262 71

Dữ liệu nhận vào	Decode	Bảng từ điển			
		Codeword (3)	Outry string (4)	Codeword (1)	Thứ tự ghép với ký tự sau (2)
87	W	-	-	256	W- (ghép W với ký tự sau)
89	Y	256	WY (sau W là Y)	257	Y- (ghép Y với ký tự sau)
83	S	257	YS (sau Y là S)	258	S- (ghép S với ký tự sau)
42	*	258	S* (sau S là *)	259	*- (ghép * với ký tự sau)
256	WY	259	*W (sau * là W)	260	WY- (ghép WY với ký tự sau)
71	G	260	WYG(sau WY là G)	261	G- (ghép G với ký tự sau)
256	WY	261	GW(sau G là W)	262	WY-(ghép WY với ký tự sau)
258	S*	262	WYS(sau WY là S)	263	S*- (ghép S* với ký tự sau)
262	WYS	263	S*W(sau S* là W)	264	WYS- (ghép WYS với ký tự sau)
262	WYS	264	WYSW(sau WYS là W)	265	WYS- (ghép WYS với ký tự sau)
71	G	265	WYSG (sau WYS là G)	-	-

➔ ***Output:*** WYS*WYGWYS*WYSWYSG

Ưu điểm:

_Thuật toán nén LZW có hệ số nén tương đối cao, trong tập tin nén không cần phải chứa bảng mã - bên nhận có thể tự xây dựng bảng mã mà không cần bên gửi phải gửi kèm theo bản tin nén.

_ Thuật toán nén LZW khắc phục được sự lãng phí về bộ nhớ mà các thuật toán trước không tận dụng được hết.

_ Thuật toán nén LZW không làm thất lạc dữ liệu (lossless) – tức trong quá trình nén sẽ không có dữ liệu nào bị mất đi hay thất lạc.
_Một ưu điểm khác của LZW là tính đơn giản, cho phép thực thi nhanh chóng.
_ Thuật toán nén LZW có thể nén 1 luồng dữ liệu đầu vào chỉ trong 1 lần truyền vào.

_ Thuật toán nén LZW không đòi hỏi nhiều thông tin đầu vào của dữ liệu được truyền vào.

Nhược điểm:

_Thuật toán sẽ không đạt hiệu quả nén cao nếu có những điều kiện sau:
+ Nguồn tin không đồng nhất và đặc tính dư thừa của nó thay đổi trong suốt tập tin.
+ Nguồn tin dài một cách đáng kể vượt quá tầm giới hạn của bảng chuỗi.

Độ phức tạp của thuật toán : *O(n)* với n là chiều dài chuỗi. Do mỗi bit chỉ cần đọc qua 1 lần và độ phức tạp của một quá trình duyệt cho 1 ký tự là cố định không đổi (constant).

Ứng dụng:

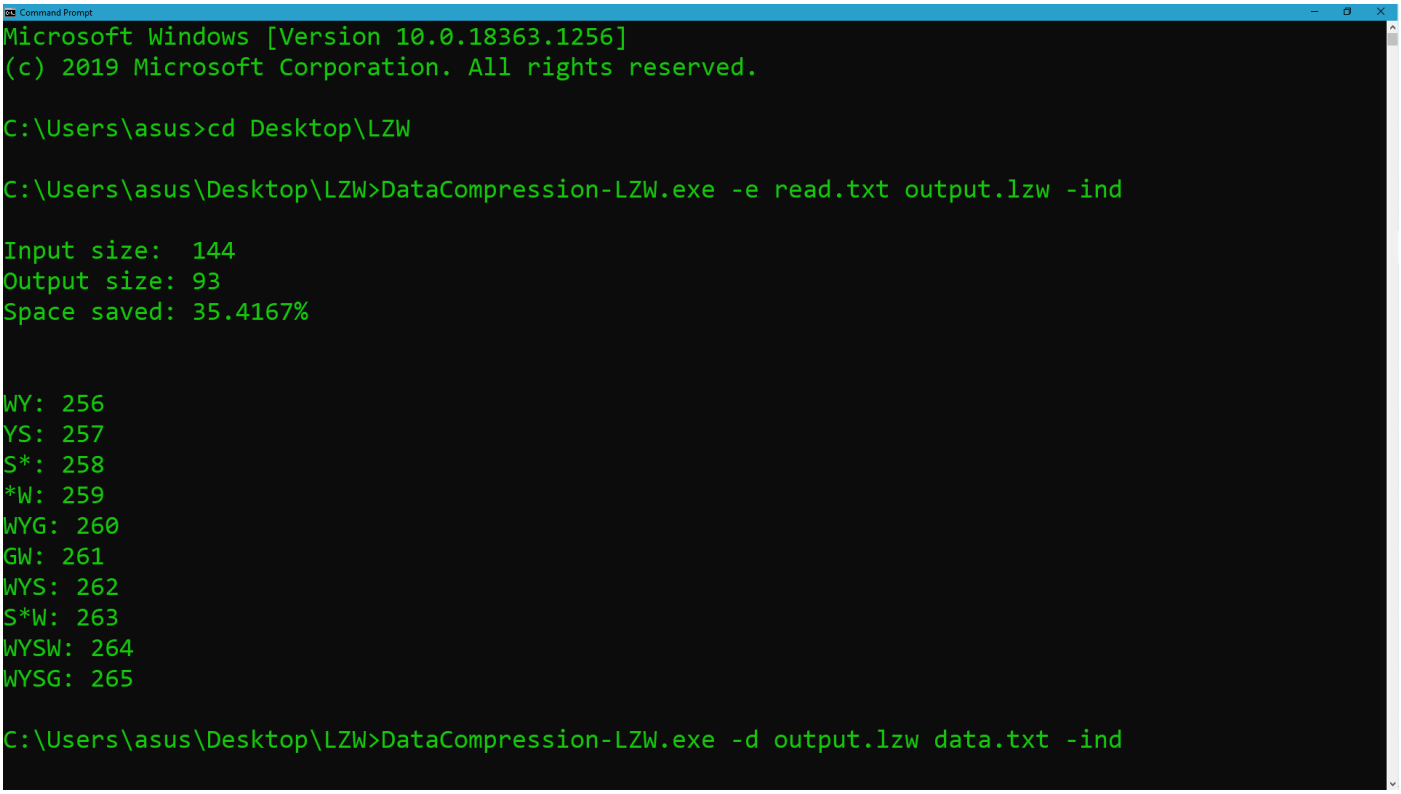
_ Điển hình là được sử dụng trong việc nén các file hình ảnh, câu lệnh nén của UNIX (UNIX's 'compress' command),… LZW đã được sử dụng trong phần mềm nén mã nguồn mở, nó đã trở thành 1 phần không thể thiếu trong HDH UNIX CIRCA 1986.
_ LZW trở thành phương thức nén dữ liệu phổ biến trên máy tính. Một file text English có thể được nén thông qua LZW để giảm ½ dung lượng gốc.

_LZW đã trở nên phổ biến khi nó được sữ dụng làm 1 phần của file GIF năm 1987.Nó cũng có thể được sử dụng trong TIFF và PDF file.

PROGRAMMING

Project of Data Structures and Algorithms

Encoded



Decoded

REFERENCES

1. https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/
2. https://vi.wikipedia.org/wiki/LZW
3. https://tailieu.vn/doc/bao-cao-nghien-cuu-cac-thuat-toan-nen-du-lieu-thuat-toan-lzw--163595.html
4. https://www.geeksforgeeks.org/huffman-decoding/?ref=rp
5. https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/