Семинарски рад

Методи и системи за обраду сигнала

Компресија података применом LZRW4 методе

Студенти:

Адриана Лилић 17725

Јована Голијанин 16038

Професор:

Проф. др Милош Радмановић

Садржај

1. Увод

1.1. Увод

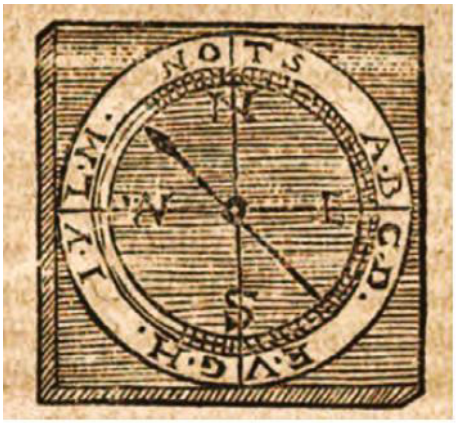
Историјски гледано, компресија података се убраја у области рачунарске науке која се развила мало касније у односу на друге области. Чини се да је стручњацима било потребно првих 20 до 25 година да развију довољно података, пре него што су осетили потребу за њиховом компресијом. Данас, када су рачунарске науке старе око 70 година, компресија података представља велико и активно поље, као и велики бизнис. Можда најбољи доказ за то је популарност Конференције о компресији података (**DCC**).

Принципе, технике и алгоритме за компресију различитих типова података, су многи људи развијали брзим темпом; заснивају се на концептима позајмљеним из различитих дисциплина као што су статистика, коначни аутомати , криве које испуњавају простор, као и Фуријеове и друге трансформације. Овај тренд је природно довео до објављивања многих књига на ову тему, поље је велико и све време постаје све веће, чиме се „ствара" више потенцијалних читалаца и постојећи текстови постају застарели у само неколико година, те је потребно да се **област стално истражује**.

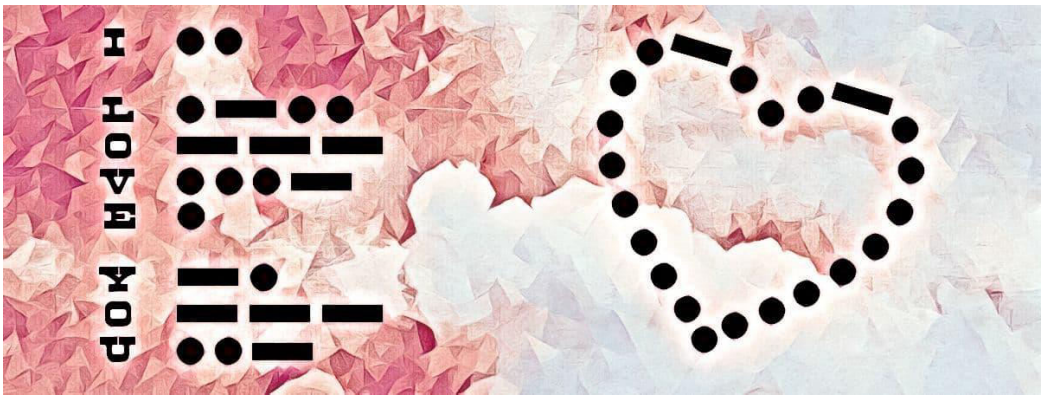
1.2. Историјски осврт

Giambattista della Porta, ренесансни научник, 1558. године, био је аутор Magia Naturalis (Природна магија), књиге у којој расправља о многим темама, укључујући демонологију, магнетизам... У књизи се помиње замишљени уређај који је од тада постао познат као „**sympathetic telegraph**". Овај уређај

је требало да се састоји од две кружне кутије, сличне компасу, свака са магнетном иглом. Свака кутија је требало да буде означена са 26 слова, уместо уобичајених праваца, а главна поента је била да су две игле требало да буду магнетизоване истим каменом. Порта је претпоставио да ће то некако ускладити игле тако да када се у једној кутији бира неко слово, игла у другој кутији ће се зањихати да покаже на исто слово.



Непотребно је рећи да такав уређај не ради (ово је, уосталом, било око 300 година пре Samuel Morse-a), али је 1711. забринута супруга писала „**Spectator**“-у, Лондонском часопису, тражећи савет како да поднесе дуга одсуства њеног вољеног мужа. Саветник Џозеф Адисон је понудио неке практичне идеје, а затим је поменуо Портин уређај, додајући да би пар таквих кутија могао да омогући њој и њеном мужу да међусобно комуницирају. Господин Адисон је затим додао да, поред 26 слова, симпатични телеграфски бројчаници треба да садрже, када их користе љубавници, „неколико целих речи којима је увек место у страственим посланицама“. Порука „Волим те“, на пример, би у таквом случају захтевала слање само три симбола уместо десет.



Овај савет је рани пример компресије текста који се постиже коришћењем кратких кодова за уобичајене поруке и дужих кодова за друге поруке. Што је још важније, ово показује како је **концепт компресије података природан за људе** који су заинтересовани за комуникацију. Чини се да смо унапред програмирани са идејом да шаљемо што мање података како бисмо уштедели време.

1.3. Дефиниција

Компресија података је процес претварања улазног тока података (изворни ток или оригинални необрађени подаци) у други ток података (излазни или компресовани ток) који има мању величину. Стрим је или датотека или бафер у меморији.

1.4. Користи од компресије

Компресија података је **популарна** из два разлога:

(1) Људи воле да **гомилају** податке и мрзе да било шта бацају. Без обзира колико велики уређај за складиштење има, пре или касније ће се препунити. Компресија података изгледа корисно јер одлаже ову неизбежност.

(2) **Људи не воле да чекају** дуго на пренос података. Када седимо за рачунаром и чекамо учитавање веб странице или да се датотека преузме, природно осећамо да је, све дуже од неколико секунди, дуго за чекање.

> *Написао сам ово писмо, које је дуже него обично, јер ми недостаје времена да га скратим.*
>
> *— Blaise Pascal*

Компресија података се развила у последњих 20 година. И квантитет и квалитет литературе у овој области пружају обилан доказ за то. Међутим, **потреба за компресијом података се осећала у прошлости**, чак и пре појаве рачунара, као што сугерише следећи цитат:

Постоји много познатих метода за компресију података. Они су засновани на различитим идејама, погодни су за различите типове података и дају различите резултате, али су сви засновани на истом принципу, односно компресују податке **уклањањем сувишности** из оригиналних података у изворној датотеци. Било који ненасумични скуп података има неку структуру, и ова структура се може искористити да би се постигла мања репрезентација података, репрезентација у којој се никаква структура не може уочити. У стручној литератури се користе термини редундантност и структура, као и глаткоћа, кохерентност и корелација; сви се односе на исту ствар. Према томе, редундантност је важан концепт у свакој дискусији о компресији података.

У типичном енглеском тексту, на пример, слово Е се појављује веома често, док је Z ретко (табеле 1 и 2). Ово се зове **алфабетска редундантност** и предлаже да се словима додељују кодови променљиве величине, при чему Е добија најкраћи код, а Z најдужи. Други тип редунданције, **контекстуална редунданција**, илуструје се чињеницом да слово К скоро увек прати слово У. Редунданција на сликама је илустрована чињеницом да на ненасумичној слици суседни пиксели имају тенденцију да имају сличне боје.

Letter	Freq.	Prob.	Letter	Freq.	Prob.
A	51060	0.0721	E	86744	0.1224
B	17023	0.0240	T	64364	0.0908
C	27937	0.0394	I	55187	0.0779
D	26336	0.0372	S	51576	0.0728
E	86744	0.1224	A	51060	0.0721
F	19302	0.0272	O	48277	0.0681
G	12640	0.0178	N	45212	0.0638
H	31853	0.0449	R	45204	0.0638
I	55187	0.0779	H	31853	0.0449
J	923	0.0013	L	30201	0.0426
K	3812	0.0054	C	27937	0.0394
L	30201	0.0426	D	26336	0.0372
M	20002	0.0282	P	20572	0.0290
N	45212	0.0638	M	20002	0.0282
O	48277	0.0681	F	19302	0.0272
P	20572	0.0290	B	17023	0.0240
Q	1611	0.0023	U	16687	0.0235
R	45204	0.0638	G	12640	0.0178
S	51576	0.0728	W	9244	0.0130
T	64364	0.0908	Y	8953	0.0126
U	16687	0.0235	V	6640	0.0094
V	6640	0.0094	X	5465	0.0077
W	9244	0.0130	K	3812	0.0054
X	5465	0.0077	Z	1847	0.0026
Y	8953	0.0126	Q	1611	0.0023
Z	1847	0.0026	J	923	0.0013

Table 1: Probabilities of English Letters

Char.	Freq.	Prob.	Char.	Freq.	Prob.	Char.	Freq.	Prob.
e	85537	0.099293	x	5238	0.006080	F	1192	0.001384
t	60636	0.070387			0.005024	H	993	
i	53012	0.061537	-|	4328	0.004677		0.001153	
				4029		B	974	
							0.001131	
s	49705	0.057698	)	3936	0.004569	W	971	0.001127
a	49008	0.056889	(	3894	0.004520	+	923	0.001071
o	47874	0.055573	T	3728	0.004328	!	895	0.001039
n	44527	0.051688	k	3637	0.004222	#	856	0.000994
r	44387	0.051525	3	2907	0.003374	D	836	0.000970
h	30860	0.035823	4	2582	0.002997	R	817	0.000948
l	28710	0.033327	5	2501	0.002903	M	805	0.000934
c	26041	0.030229	6	2190	0.002542	;	761	0.000883
d	25500	0.029601	I	2175	0.002525	/	698	0.000810
m	19197	0.022284	^	2143	0.002488	N	685	0.000795
\	19140	0.022218	:	2132	0.002475	G	566	0.000657
p	19055	0.022119	A	2052	0.002382	j	508	0.000590
f	18110	0.021022	9	1953	0.002267	@	460	0.000534
u	16463	0.019111	[	1921	0.002230	Z	417	0.000484
b	16049	0.018630	C	1896	0.002201	J	415	0.000482
.	12864	0.014933	]	1881	0.002183	O	403	0.000468
1	12335	0.014319	’	1876	0.002178	V	261	0.000303
g	12074	0.014016	S	1871	0.002172	X	227	0.000264
0	10866	0.012613	_	1808	0.002099	U	224	0.000260
,	9919	0.011514	7	1780	0.002066	?	177	0.000205
&	8969	0.010411	8	1717	0.001993	K	175	0.000203
y	8796	0.010211	‘	1577	0.001831	%	160	0.000186
w	8273	0.009603	=	1566	0.001818	Y	157	0.000182
$	7659	0.008891	P	1517	0.001761	Q	141	0.000164
}	6676	0.007750	L	1491	0.001731	>	137	0.000159
{	6676	0.007750	q	1470	0.001706	*	120	0.000139
v	6379	0.007405	z	1430	0.001660	<	99	0.000115
2	5671	0.006583	E	1207	0.001401	”	8	0.000009

Table 2: Frequencies and Probabilities of Characters.

Идеја компресије смањењем редундантности сугерише општи закон компресије података, који је „**додељивање кратких кодова уобичајеним догађајима (симболи или фразе) и дугих кодова ретким догађајима**“. Постоји много начина да се примени овај закон, а анализа било ког метода компресије показује да дубоко у себи он функционише поштујући општи закон.

Компресовање података се врши променом њихове репрезентације са неефикасног (тј. **дугог**) у ефикасну (**кратку**). Компресија је стога могућа само зато што су подаци нормално представљени у рачунару у формату који је дужи него што је апсолутно неопходно. Разлог зашто се користе неефикасне (дуге) репрезентације података је тај што **олакшавају обраду података**, а

обрада података је чешћа и важнија од компресије података. **ASCII** код за знакове је добар пример репрезентације података која је дужа него што је апсолутно неопходно. Користи 7-битне кодове јер је са кодовима фиксне величине лако радити. Код променљиве величине би, међутим, био ефикаснији, пошто се одређени карактери користе више од других и тако им се могу доделити краћи кодови.

У свету у коме су подаци увек представљени најкраћим могућим форматом, не би постојао начин компресовања података. Уместо да пишу књиге о компресији података, аутори у таквом свету би писали књиге о томе како да одреде најкраћи формат за различите типове података.

Принцип компресије уклањањем сувишности такође одговара на следеће питање: „**Зашто се већ компресована датотека не може даље компресовати**?“ Одговор је, наравно, да такав фајл има мало или нимало сувишности, тако да нема шта да се уклони. Пример такве датотеке је насумични текст. У таквом тексту, свако слово се појављује са једнаком вероватноћом, тако да им додељивање кодова фиксне величине не додаје никакву редундантност. Када се таква датотека компресује, нема вишка за уклањање. (Други одговор је да ако би било могуће компресовати већ компресовану датотеку, онда би узастопне компресије смањиле величину датотеке све док не постане један бајт или чак један бит. Ово је, наравно, смешно јер један бајт не може да садржи информације присутне у произвољно великој датотеци.)

2. Основне технике

2.1. Интуитивна компресија

Компресија података се постиже смањењем редундантности, али то такође чини податке **мање поузданим** и склонијим грешкама. С друге стране, повећање поузданости података се врши додавањем битова за проверу и битова парности, процеса који повећава величину кодова, чиме се повећава редундантност. Компресија података и поузданост података су стога супротне ствари, а интересантно је приметити да је ово друго релативно новије поље, док је прво постојало и пре појаве рачунара.

Симпатички телеграф, о коме је било речи у уводу, **Брајев код** из 1820., и **Морзеов код** из 1838. (Табела 2.1) користе једноставне облике компресије. Стога се чини да је смањење редунданције природно за свакога ко ради на кодовима, али повећање је нешто што код људи „иде против суштине“. Овај одељак разматра једноставне, интуитивне **методе компресије које су коришћене у прошлости**. Данас су ове методе углавном од историјског интереса јер су генерално неефикасне и не могу се такмичити са савременим методама компресије развијеним током последњих 15-20 година.

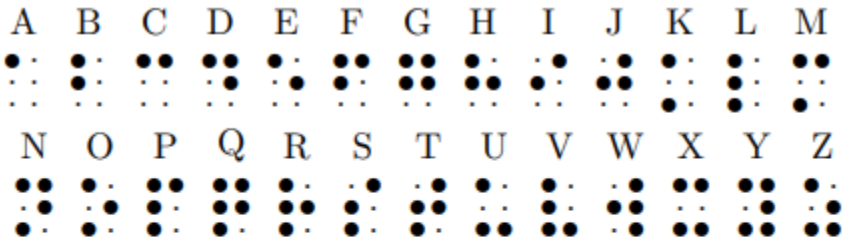


Table 1.1: The 26 Braille Letters.

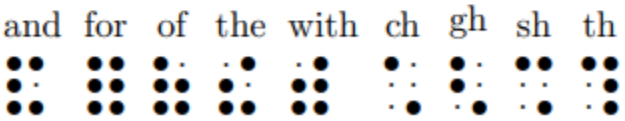


Table 1.2: Some Words and Strings in Braille.

2.2. Брајева азбука

Овај добро познати код, који омогућава слепима да читају, развио је Louis Braille 1820-их и данас је у уобичајеној употреби, након што је неколико пута модификован. Многе књиге на Брајевом писму доступне су у National Braille Press-у. Брајев код се састоји од група (или ћелија) од по 3 x 2 тачке, утиснуте на дебелом папиру. Свака од 6 тачака у групи може бити равна или подигнута, тако да је информациони садржај групе еквивалентан 6 битова, што резултира 64 могуће групе. Пошто слова (Табела 1.1), цифре и знакови интерпункције не захтевају свих 64 кода, преостале групе се користе за кодирање уобичајених речи—као што су and, for, и of—и уобичајених низова слова—као што су ound, ation и th (табела 1.2).

2.3. Редундантност у свакодневним ситуацијама

Иако не повећавамо редундантност у нашим подацима без потребе, користимо редундантне податке све време, углавном не примећујући то. Ево неколико **примера**:

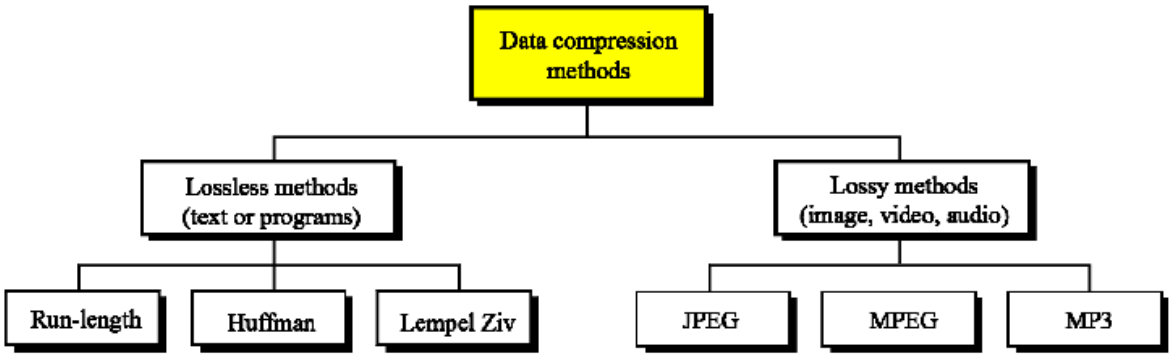
Сви **природни језици** су сувишни. Португалац који не говори италијански може да чита италијанске новине и да ипак разуме већину вести јер препознаје основни облик многих италијанских глагола и именица и зато што је већина текста који не разуме сувишна.

PIN је акроним за „Лични идентификациони број“, али банке од вас увек траже ваш „PIN број“. Ово су само два примера који илуструју колико је природно бити сувишан у свакодневним ситуацијама. Више примера може се наћи на URL адреси http://www.corsinet.com/braincandy/twice.html

Количина компресије коју постиже Брајево писмо је мала, али важна јер књиге на Брајевом писму имају тенденцију да буду веома велике (једна група покрива површину од око десет штампаних слова). Чак и ова скромна компресија има цену. Ако се књигом на Брајевом писму погрешно рукује или застари и неке тачке постану равне, може доћи до озбиљних грешака у читању јер се користи свака могућа група.

3. Подела метода за компресију података

Компресија података подразумева слање или чување мањег броја битова. Иако се у ту сврху користи много метода, ове методе се генерално могу поделити у две широке категорије: методе без губитака (**lossless**) и методе са губицима (**lossy**).



3.1. Компресија са губицима

Наше очи и уши не могу да разликују ситне промене. У таквим случајевима можемо користити метод компресије података са губицима. Ове методе су јефтиније - потребно им је мање времена и простора када је реч о слању милиона битова у секунди за слике и видео. Неколико метода је развијено коришћењем техника компресије са губицима. **JPEG** (Joint Photographic Experts Group) кодирање се користи за компресовање слика и графика, **MPEG** (Moving Picture Experts Group) кодирање се користи за компресовање видео записа, а **MP3** (MPEG аудио слој 3) за аудио компресију.

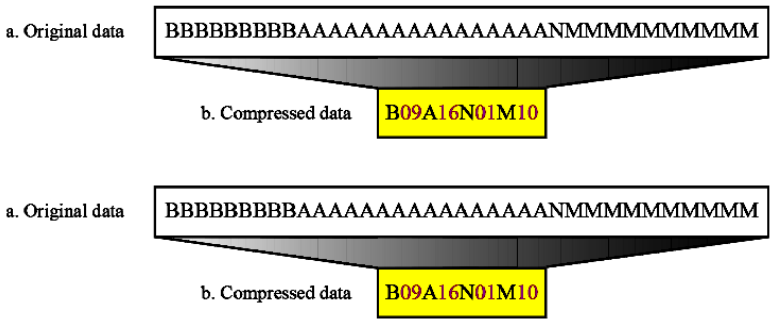
3.2. Компресија без губитака

Код компресије података без губитака, интегритет података је очуван. Изворни подаци и подаци након компресије и декомпресије потпуно су исти. Алгоритми компресије и декомпресије у овим методама су тачни и међусобно инверзни. Ниједан део података се не губи у процесу. Сувишни подаци се уклањају компресијом и додају током декомпресије. Методе компресије без губитака обично се користе када не можемо да приуштимо губитак података.

3.2.1. Run-length кодирање (RLE)

RLE је вероватно **најједноставнији** метод компресије. Може се користити за компресовање података направљених од било које комбинације симбола. Не треба да зна учесталост појављивања симбола и може бити врло ефикасно ако су подаци представљени као 0 и 1. Општа идеја која стоји иза ове методе је да замењује узастопне понављајуће појаве симбола једном **појавом симбола праћену бројем појављивања**. Метода може бити још ефикаснија ако подаци у свом узорку битова користе само два симбола (на пример 0 и 1), а један симбол је чешћи од другог.

RLE примери:



3.2.2. Статистички методи

Статистички методи за компресију користе **кодове променљиве величине**. Краћи кодови се додељују симболима или групама симбола који се појављују чешће у подацима (имају већу вероватноћу појаве). Код имплементације кодова променљиве величине постоје два **проблема**:

-Додељивање кодова који се могу једнозначно декодирати.

-Додељивање кодова са минималном просечном величином.

Samuel Morse користио је кодове променљиве величине када је креирао познати **телеграфски код**.

A	.-	N	-.	1	.----	Period	.-.-.-
B	-...	O	---	2	..---	Comma	--..--
C	-.-.	P	.--.	3	...--	Colon	---...
Ch	----	Q	--.-	4	....-	Question mark	..--..
D	-..	R	.-.	5	.....	Apostrophe	.----.
E	.	S	...	6	-....	Hyphen	-....-
F	..-.	T	-	7	--...	Dash	-..-.
G	--.	U	..-	8	---..	Parentheses	-.--.-
H	....	V	...-	9	----.	Quotation marks	.-..-.
I	..	W	.--	0	-----		
J	.---	X	-..-				
K	-.-	Y	-.--				
L	.-..	Z	--..				
M	--						

Прва верзија послала је кратке и дуге цртице које су примљене и нацртане на траци папира, где су секвенце тих цртица представљале бројеве. Свакој речи (не сваком слову) додељен је кодни број, а Морзе је створио књигу шифара (или речник) тих кодова 1837. године. Ова прва верзија је

била **примитивни облик компресије**. Морзе је касније напустио ову верзију у корист својих познатих тачака и цртица, коју је развио заједно са Алфред-ом Ваил-ом.

Постоје различити **статистички алгоритми** (Shannon-Fano, Huffman, аритметичко кодирање и други), на основу којих можемо јасно видети и израчунати како се редунданса смањује или елиминише.

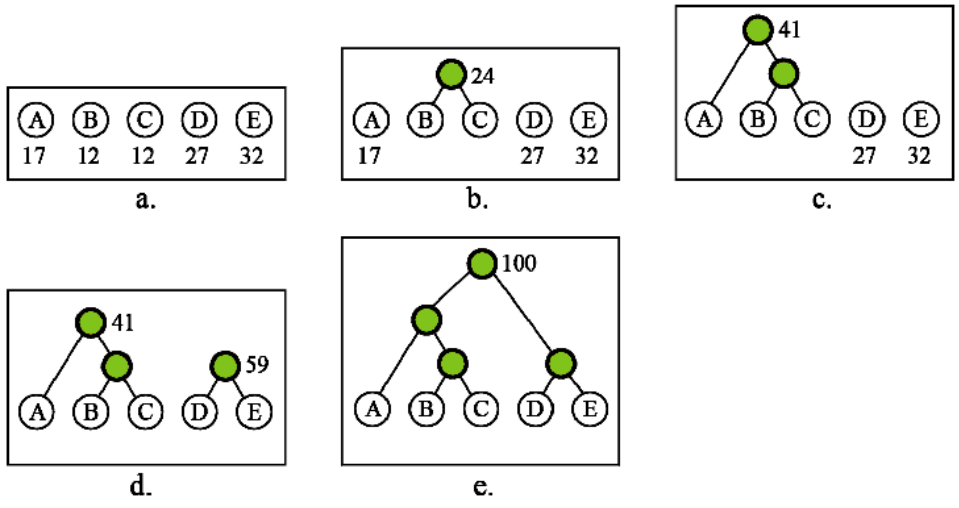
Као пример једног метода објаснићемо Huffman-ово кодирање.

3.2.2.1. Huffman-ово кодирање

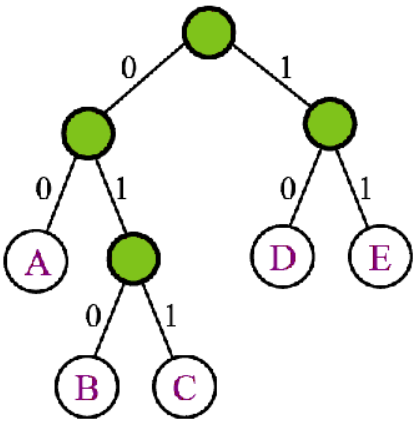
Huffman-ово кодирање **додељује краће кодове симболима који се чешће јављају**, а дуже кодовима који се јављају ређе. На пример, замислимо да имамо текстуалну датотеку која користи само пет знакова (A,B,C,D,E). Пре него што сваком знаку доделимо обрасце битова, сваком лику додељујемо тежину на основу његове учесталости употребе. У овом примеру претпоставимо да је учесталост знакова онаква каква је приказана у табели 15.1.

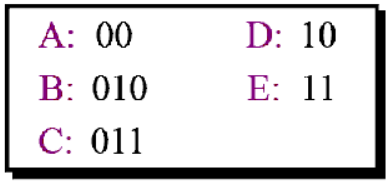
Table 15.1 Frequency of characters

Character	A	B	C	D	E
Frequency	17	12	12	27	32



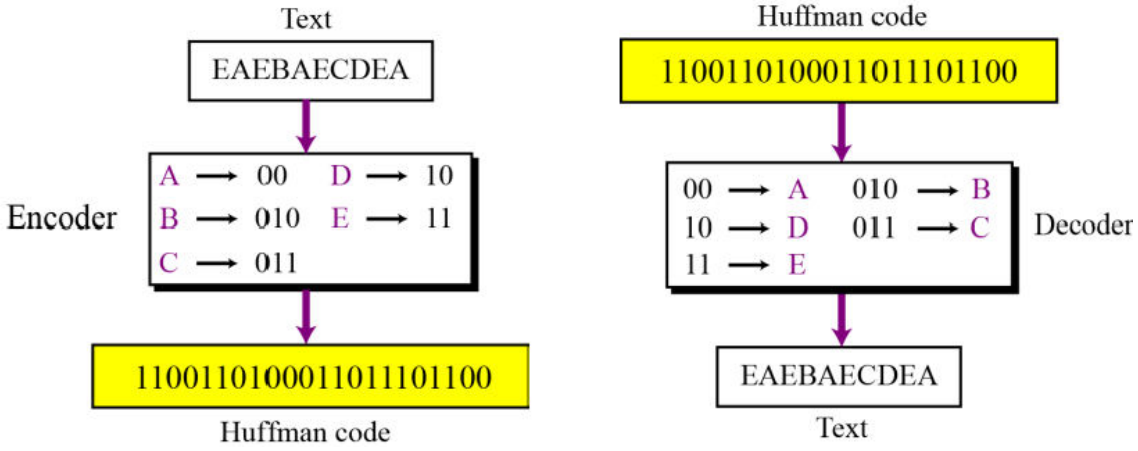
Код лика се проналази тако што се започне у корену и прате гране које воде до тог знака. Сам код је вредност бита сваке гране на путањи, узета у низу.





Code

Huffman-ово декодирање



4. Методе речника

Методе речника бирају низове симбола и кодирају сваки низ као токен користећи речник. Речник садржи низове симбола и може бити статички или динамички. Статички је непроменљив, понекад дозвољава додавање низова, али не и брисање,док динамички садржи низове који су претходно

пронађени у улазном току, омогућавајући додавање и брисања стрингова док се нови унос чита.

Обзиром на низ од n симбола, компресор заснован на речнику може, у принципу, да компресује до nH битова где је H ентропија низа. Дакле, на основу речника компресори су ентропијски кодери, али само ако је улазна датотека веома велика. Кодери су намањени за општи случај извођења компресије на сликама, аудио подацима и на тексту.

Најједноставнији пример статичког речника је речник енглеског језика који се користи за компресију енглеског текста. Замислите речник који садржи можда пола милиона речи (без њихових дефиниција). Реч (низ симбола који се завршава размаком или знаком интерпункције) се чита из улазног тока претражујући речник. Ако је пронађено подударање, индекс за речник се уписује у излазни ток. Иначе, написана је сама некомпресована реч. (Ово је пример логичке компресије.)

Као резултат тога, излазни ток садржи индексе и саме речи, и важно је да их разликујемо. Један од начина да се то постигне је да резервишете додатни део за сваку написану ставку. У принципу, 19-битни индекс је довољан да наведе ставку у 2^19 = речник са 524.288 речи. Дакле, када се пронађе подударање, можемо написати 20-битни токен који се састоји од бита заставице (можда нула) праћеног 19-битним индексом. Када није пронађено подударање, уписује се заставица од 1, праћена величином неусклађене речи,након чега следи сама реч.

Пример: Под претпоставком да се реч **bet** налази у речнику 1025, јесте кодиран као 20-битни број 0|00000000010000000001. Под претпоставком да је реч кет није пронађена, кодиран је као 1|0000011|01111000|01100101|01110100. Ово је 4-бајтни број где 7-битно поље 0000011 означава да следе још три бајта.

Под претпоставком да је величина записана као 7-битни број и да просечна величина речи има пет знакова, некомпресована реч заузима у просеку 6 бајтова (= 48 бита) у излазом току. Компресовање 48 бита у 20 је одлично, под условом да се често дешава довољно. Дакле, морамо одговорити на питање; Колико је подударања потребно по реду да имате укупну компресију? Означавамо вероватноћу подударања (случај када се

реч налази у речнику) од P. Након читања и сажимања N речи величина излазног тока биће N [20P + 48(1 − P)] = N [48 − 28P] бита. Величина улазног тока је (под претпоставком да је пет знакова по речи) 40N бита. Компресија се постиже када је N[48 − 28P] < 40N, што имплицира P > 0,29. Потребна нам је стопа подударања од 29% или боље постићи компресију.

Као једноставну вежбу откривамо фактор компресије који добијамо са P = 0,9. Величина излазног тока је N[48 − 28P] = N[48 − 25.2] = 22.8N. Величина улазног тока је, као и раније, 40N. Фактор компресије је дакле 40/22,8 ≈ 1,75.Све док се улазни ток састоји од енглеског текста, већина речи ће се наћи у речнику од 500.000 речи. За друге врсте података, међутим, можда неће бити тако добро. Фајл који садржи изворни код рачунарског програма може садржати „речи“ као што су count, xor, malloc које се можда не налазе у енглеском речнику. Обично бинарна датотека садржи бесмислице када се гледа у ASCII, тако да се може наћи врло мало подударања, што резултира у знатној експанзији уместо компресије.

Ово показује да статички речник није добар избор за опште намене. Међутим, може бити добар избор за специјалне намене. Размотрите ланац продавницама хардвера, на пример. Њихове датотеке могу садржати речи као што су матица, вијак и сликајте много пута, али речи као што су кикирики, мућа и слика ће бити ретке.Софтвер за компресију посебне намене за такву компанију може имати користи од малог,специјализованог речника који садржи само неколико стотина речи. Рачунари у свакој грани би имали копију речника, што би олакшавало компресовање датотека и слање истих између продавница и канцеларија у ланцу.

Генерално, пожељна је адаптивна метода заснована на речнику. Такав метод може почети са празним речником или са малим, подразумеваним речником, додавајући речи у њега које се налазе у улазном току и које бришу старе речи јер велики речник, што би представљало спору претрагу. Такав метод се састоји од петље где свака итерација почиње читањем улазног тока и разбијање (рашчлањивање) на речи или фразе. Онда би требало претражити речник за сваку реч и, ако се пронађе подударање, написати токен на излазном току. У супротном, некомпресовану реч треба написати и додати у речник. Последњи корак у свакој итерацији проверава

да ли би стара реч требало да буде избрисана из речника. Ово може звучати компликовано, али има две предности:

1. Укључује операције претраживања и подударања низова, а не нумеричка израчунавања.

2. Декодер је једноставан (ово је метода асиметричне компресије). У адаптивној методи заснованој на речнику, декодер, да бисте прочитали његов улазни ток, одређује да ли је тренутна ставка токен или некомпресован податак, користиећи токене за добијање података из речника и излазе коначне, некомпрресових података. Не мора да рашчлани улазни ток на сложен начин и нема да претраге речника да би се пронашла подударања.

Током 1970-их Јакобу Зив и Абрахам Лемпел развили су прве методе, LZ77 LZ78, за компресију засновану на речнику. Њихове идеје су многима биле извор инспирације истраживања, преко којих су их генерализовали, побољшали и комбиновали са RLE и статистичким методе за формирање многих најчешће коришћених метода компресије без губитака за текст, слике и звук. На даље ћемо обрадити најчешће коришћене методе LZ компресије данас и показати како су се развиле из основних идеја Жива и Лемпела.

5. Стринг компресија

Генерално, методе компресије засноване на низовима симбола могу бити ефикасније од методе које компресују појединачне симболе. Компресија је боља ако симболи азбуке имају веома различите вероватноће појављивања. Користимо једноставан пример који показује да вероватноће низова симбола варирају више од вероватноће појединачних симбола који чине низове.

Почињемо са алфабетом са 2 симбола a1 и a2, са вероватноћама P1 = 0,8 и P2 = 0,2, респективно. Просечна вероватноћа је 0,5 и можемо добити представу о томе (колико појединачне вероватноће одступају од просека) израчунавањем збира апсолутних разлика

|0,8 − 0,5| + |0,2 − 0,5| = 0,6. Било који код променљиве величине би доделио 1-битне кодове за два симбола, тако да је просечна величина кода 1 бит по симболу.

Сада генеришемо све низове од два симбола. Има их четири, приказане у табели 3.1а, заједно са њиховим вероватноћама и скупом Хафманових кодова. Просечна вероватноћа је 0,25, тако да збир апсолутних разлика сличан горњој даје

|0,64 − 0,25| + |0,16 − 0,25| + |0,16 − 0,25| + |0,04 − 0,25| = 0,78

Просечна величина Хафмановог кода је 1 × 0,64 + 2 × 0,16 + 3 × 0,16 + 3 × 0,04 = 1,56 битова по низу, што је 0,78 бита по симболу.

У следећем кораку на сличан начин креирамо свих осам низова од три симбола су приказани у табели 3.1б, заједно са њиховим вероватноћама и скупом Хафманових кодова. Просечна вероватноћа је 0,125, дакле збир апсолутних разлика сличних горе наведеним износи

|0.512 − 0.125| + 3|0.128 − 0.125| + 3|0.032 − 0.125| + |0.008 − 0.125| = 0.792

String	Probability	Code
a_1a_1	$0.8 \times 0.8 = 0.64$	0
a_1a_2	$0.8 \times 0.2 = 0.16$	11
a_2a_1	$0.2 \times 0.8 = 0.16$	100
a_2a_2	$0.2 \times 0.2 = 0.04$	101

(a)

Str. size	Variance of prob.	Avg. size of code
1	0.6	1
2	0.78	0.78
3	0.792	0.728

(c)

String	Probability	Code
$a_1a_1a_1$	$0.8 \times 0.8 \times 0.8 = 0.512$	0
$a_1a_1a_2$	$0.8 \times 0.8 \times 0.2 = 0.128$	100
$a_1a_2a_1$	$0.8 \times 0.2 \times 0.8 = 0.128$	101
$a_1a_2a_2$	$0.8 \times 0.2 \times 0.2 = 0.032$	11100
$a_2a_1a_1$	$0.2 \times 0.8 \times 0.8 = 0.128$	110
$a_2a_1a_2$	$0.2 \times 0.8 \times 0.2 = 0.032$	11101
$a_2a_2a_1$	$0.2 \times 0.2 \times 0.8 = 0.032$	11110
$a_2a_2a_2$	$0.2 \times 0.2 \times 0.2 = 0.008$	11111

(b)

Table 3.1: Probabilities and Huffman Codes for a Two-Symbol Alphabet.

Просечна величина Хафмановог кода у овом случају је 1 × 0,512 + 3 × 3 × 0,128 + 3 × 5 ×0,032 + 5 × 0,008 = 2,184 бита по низу, што је једнако 0,728 бита по симболу.

Како настављамо да генеришемо све дуже и дуже низове, вероватноће низова се разликују све више и више од свог просека, а просечна величина кода постаје боља (Табела 3.1ц).Ово је разлог зашто метода компресије која компресује стрингове, а не појединачне симболе , може, у принципу, дати

боље резултате. То је и разлог зашто се различите методе засноване на речницима генерално боље и популарније од Хафманове методе и његових варијанти . Горњи закључак је фундаменталан резултат теорије изобличења брзине, оног дела теорије информација који се бави подацима компресије.

6. Једноставна компресија речника

Ово је једноставан метод са два пролаза. Први пролаз чита изворну датотеку и припрема листу свих различитих пронађених бајтова. Други пролаз користи ову листу за компресију бајтова података. У наставку су детаљније објашњени кораци.

1. Изворна датотека се чита и припрема се листа различитих бајтова на које се наилази.За сваки бајт, број пута који се појављује у изворној датотеци (његова фреквенција) је такође укључен у листу.

2. Листа је сортирана у опадајућем редоследу фреквенција. Дакле, почиње са бајтом вредности које су уобичајене у датотеци, а завршава се бајтовима који су ретки. Пошто се листа састоји од различитих бајтова, може имати највише 256 елемената.

3. Сређена листа постаје речник. Написано је на компресованој датотеци, којој претходи његовој дужини (цео број од 1 бајта).

4. Изворна датотека се поново чита бајт по бајт. Сваки бајт се налази у речнику (директном претрагом) и бележи се њен индекс. Индекс је број у интервалу [0, 255],тако да захтева између 1 и 8 бита (али примећује се да ће већина индекса обично бити малих бројева јер се уобичајене вредности бајтова чувају раније у речнику). Индекс је написан на компресованој датотеци, а претходи 3-битни код који означава дужину индекса. Дакле, код 000 означава 1-битни индекс, код 001 означава 2-битни индекс, и тако даље до код 111, који означава 8-битни индекс.

Компресор одржава кратак, 2-бајтни бафер где прикупља битове који ће бити написани на компресованој датотеци. Када је први бајт бафера попуњен, он је записан на датотеци и други бајт се помера у први бајт.

Декомпресија је једноставна. Декомпресор почиње читањем дужине речника, затим и самог речника. Затим декодира сваки бајт читајући његов 3-битни код, праћен вредношћу индекса. Индекс се користи за лоцирање следећег бајта података у речнику.

Компресија се постиже јер је речник сортиран по учесталости бајтова. Сваки бајт се замењује количином између 4 и 11 битова (прати се 3-битни код од 1 до 8 бита). 4-битна количина одговара степену компресије од 0,5, док на 11-битна количина одговара степену компресије од 1,375, проширењу. Најгори случај је датотека у којој се појављују све вредности од 256 бајтова и имају униформну дистрибуцију. Степен компресије у таквом случају је просечна вредност

$$\frac{1}{8}(0.5 + 5/8 + 6/8 + 7/8 + 8/8 + 9/8 + 10/8 + 11/8) = 0.9375.$$

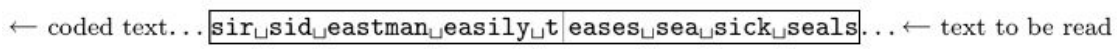
(Заправо, нешто лошије од 0,9375, јер компесована датотека такође укључује речник, чија је дужина у овом случају 257 бајтова.) Искуство указује на типичну компресију односа од око 0,5.

Лоша страна методе је спора компресија; резултат два пролаза компресор ради у комбинацији са спором претрагом (споро, јер речник није сортирао према вредностима бајтова, тако да се не може користити бинарно претраживање). Декомпресија, насупрот томе, није спора.

7. LZ77 (Клизни прозор)

Главна идеја овог алгоритма (који се понекад назива и LZ1) је да користи део претходно виђеног улазног низа као речник. Кодер одржава прозор улазног низа и помера улаз у том прозору с десна улево, како се који низ симбола кодира. Метода је по томе названа клизни прозор. Прозор је подељен у два дела. Први део на левој страни се назива search buffer– бафер за тражење. Ово је тренутни речник, и увек укључује симболе који су се скоро налазили на улазу и били кодирани. Део на десној страни је "look ahead buffer "– бафер за гледање унапред, који садржи текст који тек треба кодирати. У практичној имплементацији бафер за претрагу је дугачак неколико хиљада бајтова, док је "look ahead" бафер велики само десетак

бајтова. Усправна црта између t и e у примеру испод представља границу између два бафера. Тиме претпостављамо да је текст “sir sid eastman easily t ” већ компресован, док је остатак текста тек потребно компресовати.

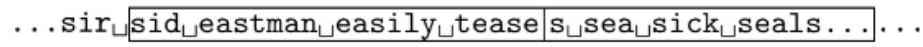


Кодер скенира бафер претраге отпозади (с десна у лево) тражећи поклапање првог симбола e из "look ahead" бафера. Проналази једно e у речи easily. Оно је удаљено 8 симбола од краја бафера претраге (ово се назива offset). Кодер онда проналази што је могуће више симбола иза ова два e. Три симбола eas се поклапају, у овом случају, па је дужина поклапања 3. Кодер наставља са скенирањем, покушавајући да нађе дуже поклапање. У нашем случају, постоји још једно поклапање, у речи eastman, са offset -ом 16, и има исту дужину. Кодер бира најдуже поклапање, или ако су сви исте дужине, последње нађено и припрема токен (16,3,”e”).

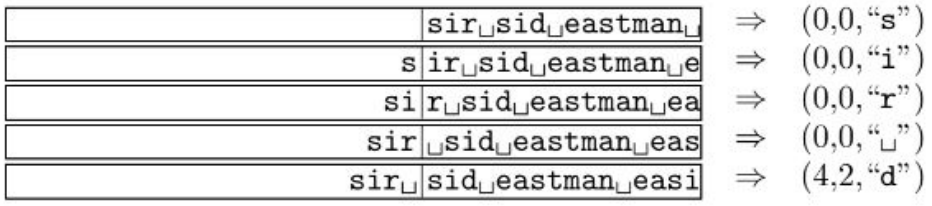
Бирањем последњег поклапања, пре него првог, поједностављује кодер, обзиром да мора да прати само последње нађено поклапање. Интересантно је приметити да бирањем првог поклапања, иако усложњава програм, такође има предност. Бира најмањи offset. Учинило би се да ово није предност, јер би токен требао да има довољно места за највећи могући токен. Ипак, постоји могућност да се LZ77 испрати са Хафманом, или неким другим статистичким кодирањем токена, где су мали offset -и додељени краћим кодовима. Овај метод, који је предложио Бернд Херд, назива се LZX. Већи број мањих offset -а подразумева бољу компресију у LZX.

Питање које се може поставити је “Како декодер зна да ли је кодер одабрао прво или последње поклапање?”. Одговор је да декодер не зна, али да не мора да зна. Декодер једноставно чита токене и користи сваки offset да нађе низ у тексту, без потребе да зна да ли је тај низ био први или последњи.

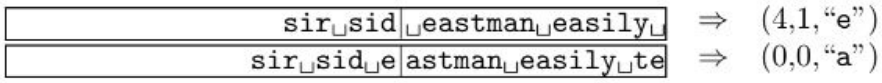
Генерално, LZ77 токен има три дела: offset, дужину и следећи симбол у "look ahead" баферу (што је, у нашем случају, друго e у речи teases). Овај токен се уписује у излазни низ и прозор се помера у десно (или, алтернативно, улазни низ се помера лево) за четири места: три места за пронађени низ и једно место за следећи симбол.



Ако претрага уназад не врати ниједно поклапање, LZ77 токен са offset-ом и дужином 0 и са непронађеним карактером ће бити уписан. Ово је такође разлог зашто токен мора имати трећу компоненту. Токени са offset -ом и дужином 0 су чести на почетку било које компресије, када је бафер претраге празан или скоро празан. Првих пет корака у кодирању у нашем примеру су следећи:



Следећи корак поклапа размак и кодира стринг “ e”



А следећи корак не поклапа ништа и кодира “a”

Јасно је, токен облика (0,0,...) који кодира један симбол, не обезбеђује добру компресију. Лако је проценити његову дужину. Величина offset -а је [log2 S], где је S дужина бафер претраге. У пракси, бафер претраге може бити дуг пар хиљада бајтова, тако да је offset типично величине 10-12 бита. Величина поља “дужина” је, слично томе, log2(L − 1), где је L дужина "look ahead" бафера. У пракси, "look ahead" бафер је дуг само пар десетина битова, па је величина поља “дужина” само пар битова. Величина поља “симбол” је типично 8 бита, али генерално износи log2A, где је A величина азбуке. Укупна величина токена (0,0,...) типично може бити 11+5+8=24 8 бита, што је много дуже него необрађени 8-битна величина једног симбола који се кодира.

Ево примера који показује зашто дужина поља може бити дужа од величине "look ahead" бафера:

Први симбол у "look ahead" баферу поклапа се са 5 а симбола у баферу претраге. Изгледа да се два екстремна а поклапају са дужинама од 3 и да би кодер требао да одабере последњи (крајњи леви) низ и да креира токен (28,3,“a”). У ствари, он креира токен (3,4,“ ”). Низ од 4 симбола алфа се

поклапа са последња три симбола alfa у баферу претрагеу и узима први симбол a из "look ahead" бафера. Разлог за ово је тај што декодер може да обради такав токен природно, без икаквих модификација. Почиње од позиције 3 бафер претрагеа и копира следећа 4 симбола, један по један, проширујући свој бафер на десно. Прва три симбола су копије из старог бафера, а четврти је копија првог од та три.

Декодер је много једноставнији од кодера LZ77 је тиме асиметрични метод компресије). Мора да одржава бафер, једнак по величини прозору кодера. Декодер узима токен, проналази поклапање у свом баферу, уписује поклапање и треће поље из токена у излазни низ, и помера поклопљени стринг и треће поље у бафер. Ово имплицира да је LZ77, или било која његова варијанта, корисна у случајевима где је фајл компресован једном (или само неколико пута) и декомпресује се често. Архива компресованих фајлова која се ретко користи је добар пример.

На први поглед изгледа да овај метод нема никакве претпоставке у вези улазних података. Првенствено, не обраћа пажњу на учесталост било којих симбола. Ако мало размислимо, можемо уочити да због природе клизног прозора, LZ77 алгоритам увек поређује "look ahead" бафер са текстом који је скоро унет у бафер претраге и никада са текстом који је одавно убачен (и тиме избачен из бафера претраге). Овај алгоритам тиме имплицитно подразумева да се шаблони у улазним подацима појављују близу један другог. Подаци који задовољавају ову претпоставку ће бити добро компресовани.

Основни LZ77 метод је побољшан на више начина од стране истраживача и програмера за време 1980-их и 90-их година. Један начин да се користе поља променљиве дужине за “дужину” и “offset”. Други начин је да се повећа величина оба бафера. Повећање величине бафер претрагеа омогућава да се пронађе поклапање, али за узврат повећава време претраге. Велики бафер претраге тиме захтева софистицираниjу структуру података која омогућава брзу претрагу. Треће побољшање је у директној вези са клизајућим прозором. Најједноставнији прилаз је да се помери цео текст у прозору у лево након сваког поклапања. Бржи метод је да се замени линеарни прозор са кружним прозором, где је клизајући прозор

имплементиран тако што се ресетују два поинтера. Још једно побољшање је додавање једног додатног бита (a flag) сваком токену, тиме елиминишући треће поље у токену. Специјално треба напоменути и хеш таблицу коришћену у “Deflate” алгоритму.

8. LZRW1

Развио је Рос Вилиамс и као једноставан, брз LZ77 варијанта, LZRW1 је такође повезан са методом A1 LZFG-a . Главна идеја је да се пронађе подударање у једном кораку, коришћењем хеш табеле. Ово је брзо, али не баш ефикасно, јер пронађено подударање није увек најдуже. Почињемо са описом алгоритма,пратићемо формат компресованог тока и закључити примером.

Метода користи целу доступну меморију као бафер и кодира улазни ток у блоковима. Блок се чита у бафер и потпуно је кодиран, а затим се следећи блок чита и кодира, и тако даље. Дужина бафера за претрагу је 4К и тај бафера за гледање унапред је 16 бајтова. Ова два бафера клизе дуж улазног блока у меморији с лева на десно. Потребно је одржавати само један показивач, p_src, показујући на почетак бафера за гледање унапред. Показивач p_src је иницијализован на 1 и он се повећава након што је свака фраза кодирана, чиме се оба бафера померају удесно по дужини фразе. Слика 3.13 показује како бафер претраге је на почетку празан, и затим расте до 4К, а затим почиње да клизи удесно, пратећи бафер за гледање унапред.

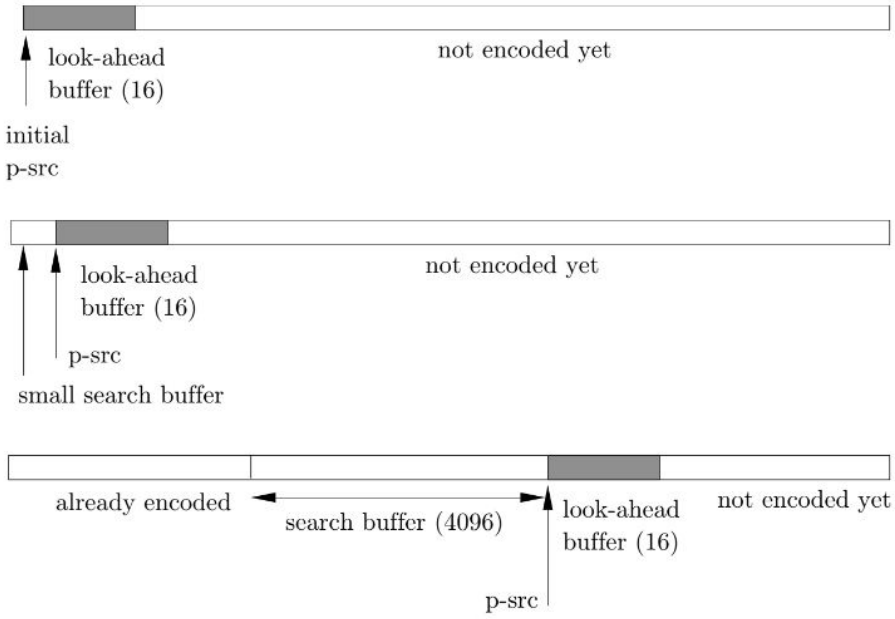


Figure 3.13: Sliding the LZRW1 Search- and Look-Ahead Buffers.

Три крајња лева знака бафера за гледање унапред се хеширају у 12-битни број I, који се користи за индексирање низа од 2^12 = 4.096 показивача. Показивач P се преузима и одмах се замењује у низу са I. Ако P показује ван претраге бафера, нема подударања; први знак у баферу за гледање унапред излази као а литерал, а p_src се повећава за 1. Иста ствар се ради ако P показује унутар претраге бафера али низу који се не поклапа са оним у баферу за гледање унапред. Ако P показује на подударање од најмање три карактера, кодер проналази најдуже подударање (највише16 знакова), избацује подударну ставку и помера p_src за дужину подударања. Овај процес је приказан на слици 3.15. Занимљива ствар коју треба приметити је да низ показивача не мора да се иницијализује када се кодер покрене, пошто кодер проверава сваки показивач. У почетку су сви показивачи насумични, али како се замењују, све више и више њих указује на права подударања.

Излаз LZRW1 кодера (слика 3.16) се састоји од група, од којих свака почиње са 16-битном контролном речју, праћеном 16 ставки. Свака ставка је или 8-битни литерал или 16-битна копија (подударање) која се састоји од 4-битног поља дужине б (где је дужина б + 1) и 12-битни помак (поља а и с). Поље дужине означава дужине између 3 и 16. 16 битова контролне речи

означавају сваку од 16 ставки које следе (заставица 0 означава литерал и заставица 1, подударну ставку). Последња група може да садржи мање од 16 података.

Декодер је још једноставнији од кодера, пошто му није потребан низ показивача. Одржава велики бафер користећи p_src показивач на исти начин као кодер. Декодер чита контролну реч из компресованог тока и користи је 16 бита за читање 16 ставки. Дословна ставка се декодира додавањем у бафер и повећавајући p_src за 1. Ставка копије се декодира одузимањем помака од p_src,преузимање стринга из бафера за претрагу, дужине назначене пољем за дужину, и додајући га у бафер. Тада се p_src повећава за дужину.

Табела 3.14 илуструје првих седам корака кодирања “that_thatch thaws”. Вредности које производи хеш функција су произвољне. У почетку, сви показивачи су насумични(означено са „било који“), али се врло брзо замењују корисним.

p_src	3 chars	Hash index	P	Output	Binary output
1	"tha"	4	any→4	t	01110100
2	"hat"	6	any→6	h	01101000
3	"at␣"	2	any→2	a	01100001
4	"t␣t"	1	any→1	t	01110100
5	"␣th"	5	any→5	␣	00100000
6	"tha"	4	4→4	4,5	0000|0011|00000101
10	"ch␣"	3	any→3	c	01100011

Table 3.14: First Seven Steps of Encoding "that thatch thaws".

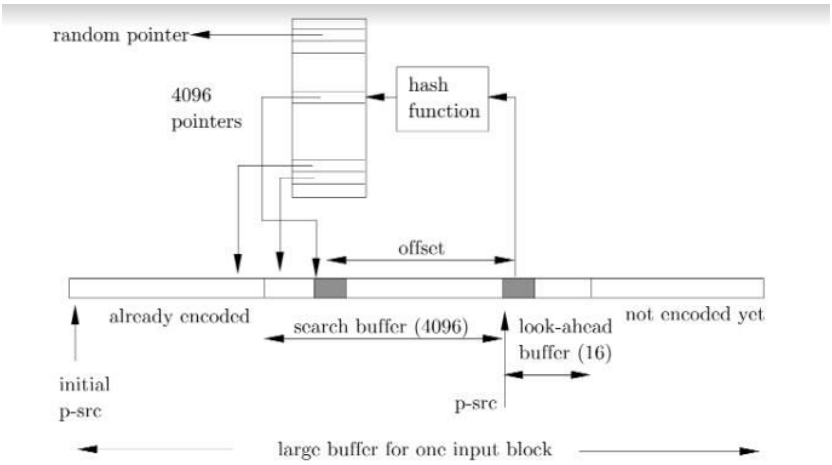


Figure 3.15: The LZRW1 Encoder.

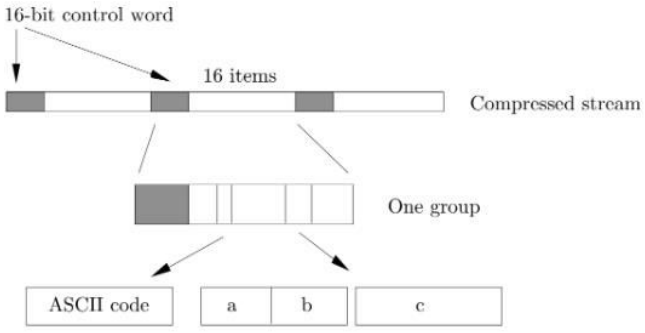


Figure 3.16: Format of the Output.

Тестови које је урадио оригинални програмер показују да LZRW1 ради око 10% лошије од LZC-а (Unix програм за компресију), али је четири пута бржи. Такође, ради око 4% лошије од LZFG-а (метода A1), али ради десет пута брже. Стога је погодан за случајеве где је брзина важнија од перформанси компресије. 68000 Имплементација асемблерског језика захтевала је, у

просеку, извршење само 13 машинских инструкција за компресовање и четири инструкције за декомпресију, 1 бајт.

Постоје практичне ситуације у којима је брзина компресије важнија од односа компресије. Пример је услужни програм за компресију за персонални рачунар који одржава све датотеке (или групе датотека) на чврстом диску у компресованом облику, да би се уштедио простор. Такав програм треба да буде транспарентан за корисника; требало би аутоматски декомпресује датотеку сваки пут када се отвори и аутоматски је компресује када је у употреби затворено. Да би био транспарентан, такав програм треба да буде брз; са степеном компресије будући да је само секундарна карактеристика.

9. LZRW4

LZRW4 је варијанта LZ77, заснована на идејама Роса Вилијамса о могућим начинима комбиновања метода речника са предвиђањем . LZRW4 такође преузима и идеје из LZRW1. Користи бафер од 1 мегабајта где бафер претраге и "look ahead" бафер клизе с лева на десно. У било ком тренутку у процесу кодирања, контекст реда-2 тренутног симбола (два најновија симбола у баферу претраге) се користи за предвиђање симбола. Два симбола која чине контекст су хеширана у 12-битни број I, који се користи као индекс за 2^12 = 4,096 низ A партиција. Свака партиција садржи 32 показивача на улазне податке у баферу од 1 Мбајта (сваки показивач је дакле дуг 20 бита).

32 показивача у партицији A[I] се проверава да би се пронашло најдуже подударање између бафера за гледање унапред и до сада виђених улазних података. Бира се најдуже подударање и кодира у 8 бита. Прва 3 бита кодирају дужину подударања према табели 3.17; преосталих 5 битова идентификују показивач у партицији. Такав 8-битни број назива се копија ставке. Ако није пронађено подударање, литерал се кодира у 8 битова. За сваку ставку екстрабит је припремљен, 0 за литерал и 1 за ставку копије. Додатни битови се акумулирајуу групама од 16, а свака група се исписује, као у LZRW1, испред 16 ставки на које се односи

3 bits:	000	001	010	011	100	101	110	111
length:	2	3	4	5	6	7	8	16

Таблела 3.17

Партиције се стално ажурирају померањем „добрих“ показивача ка почетку њихове поделе. Када се пронађе подударање, кодер мења изабрани показивач са показивачем на пола пута према партицији (слика 3.18а,б). Ако није пронађено подударање,цела партиција од 32 показивача се помера улево и нови показивач се уноси надесно, показујући на тренутни симбол (слика 3.18ц).

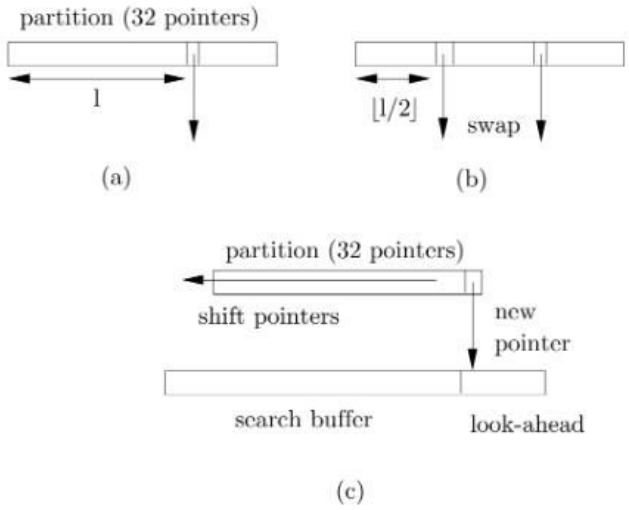


Figure 3.18: Updating an LZRW4 Partition.

10. Пример рада LZRW4-а са текстуалним фајлом

Рос Вилијамс, научник који се бавио компресиом података, је 1991. године изучавао LZRW4 алгоритам како би припремио свој докторат. Под временским притиском, рад је обајвљен, али није доведен до верзије која би била за објављивање, те смо морали да изменимо код како би радио.

Росов рад упоређује две класе алгоритама (Markov и LZ77) и показује како алгоритми који комбинују обе технике имају потенцијал да обезбеде боље перформансе од до тада виђених.

Да би тестирао своју идеју, Вилијамс је осмислио LZRW4. Није имплементирао LZRW4, али је написао програм за мерење количине компресије коју би алгоритам донео.

Иако је избор LZ алгоритама био велики, Вилијамс је изабрао класу LZ77 са којом је био упознат, да би избегао проблеме са патентима.

LZRW4 користи хеш табелу од 4096 партиција од којих свака садржи 32 показивача на улаз који се чува у баферу од 1M. На почетку сваке фразе, LZRW4 хешује претходна два знака да би добио број партиције у опсегу 0..4095. 32 показивача у тој партицији се траже за најдуже подударање и дужине 3 бита =[2,3,4,5,6,7,8,16] и број показивача 5 бита=[0,..,31] се кодирају и преносе . LZRW4 користи један контролни бит за сваку фразу да каже да ли следи литерални бајт или бајт копије. Ови контролни битови су баферовани као у другим LZRW алгоритмима. Једнака дужина литералних и копираних ставки је занимљива карактеристика алгоритма за коју је Вилијамс веровао да ће некоме једног дана бити од користи!

Да би ажурирао партицију, LZRW4 мења показивач са показивачем на пола пута према предњој страни партиције. Ако показивачу одговара мање од 4 бајта, показивач на тренутну позицију (почетак Ziv-a) се помера у партицију.

Резултати LZRW4 нису запањујући, али показују да идеја ради. Користећи контексте, LZRW4 се извлачи коришћењем 3-битних дужина и 5-битних „индекса“, док његов предак алгоритам LZRW3-

А користи 4-битне дужине и 12-битне "индексе", што даје упоредиву компресију.

Вилијамс није знао да ли је ова идеја оригинална и да ли ће донети било какву корист. Међутим, Вилијамс до 1991. није чуо ни за један Ziv/Lempel/Markov хибрид, а експерименти су барем показали да идеја функционише, иако нема стварну конкурентску корист. Уз правилно подешавање, употреба контекста би могла да обезбеди додатну компресију са малим утицајем на брзину (али обично много утицаја на меморију!). У најмању руку, Вилијамс је осмислио још једно оружје у компресији података.

У наставку следи изводни код који је коришћен као и port.h фајл:

```
#include <stdio.h>
#include <fcntl.h>
#include "port.h"
#include "stdint.h"

#define MY_BUFFER_SIZE (1024*1024)
#define HASHLENGTH  4096
#define HASHDEPTH     32
#define CONTEXT       2
#define MEDMATCH      8 /* Expressible lengths are [2..MEDMATCH,MAXMATCH]. */
#define MAXMATCH      16
```

```
#define NOSHIFTTHRESH  4 /* This or longer match and we don't shift
*/

main(argc, argv)
int argc;
char** argv;
{
   typedef unsigned char* queue[HASHDEPTH];
   /* Element zero is most recent element. */

   unsigned char buf[MY_BUFFER_SIZE + 2000];
   queue hash[HASHLENGTH];

   char* input_file_name;
   int infile;

   unsigned int bits_in = 0;
   unsigned int bits_out = 0;
   if (argc != 2)  //org !=2
   {
      printf("To run lzcontext alg: x <filename>\n");
      exit(0);
   }
```

```
   //printf("Welcome to LZCONTEXT\n");
   printf("--------------------\n");
   printf("Buffer size       = %u\n", MY_BUFFER_SIZE);
   printf("Hash table length  = %u\n", HASHLENGTH);
   printf("Hash table depth   = %u\n", HASHDEPTH);
   printf("Context           = %u\n", CONTEXT);
   printf("Med match         = %u\n", MEDMATCH);
   printf("Max match         = %u\n", MAXMATCH);
   printf("No shift threshold = %u\n", NOSHIFTTHRESH);

   input_file_name = argv[1];  //org
   //input_file_name =
"C:\\Users\\38165\\Desktop\\MISOS\\ConsoleApplication2\\ConsoleA
pplication2\\nesto.txt";

   infile = open(input_file_name, O_RDONLY); /* O_BINARY*/
   while (1)
   {
      unsigned int  i, j, k;      //ULONG == unsigned long
      unsigned char* p_next_byte;
      unsigned long bytes_in;

      /* Read in a block of bytes to compress. */
```

```
bytes_in = read(infile, buf, (unsigned int)MY_BUFFER_SIZE);
if (bytes_in == 0)
   break;

/* Initialize the hash table so all entries point to a defined string. */
for (i = 0; i < HASHLENGTH; i++)
   for (j = 0; j < HASHDEPTH; j++)
      hash[i][j] = (unsigned char*)"0123456789ABCDEF";


/* Get past the first few context bytes. */
bits_in += CONTEXT * 8;
bits_out += CONTEXT * 9;
p_next_byte = buf + CONTEXT;

/* Loop once for each item. */
/* The end is sloppy but this is only approximate so who cares? */
while (p_next_byte < (buf + bytes_in))
{
   unsigned long index;
   unsigned int maxlength;  // UWORD == unsigned int
   unsigned char** this_queue;
   unsigned char* old_p_next_byte = p_next_byte;
```

```
        unsigned long longest_match_index;

        index = (*(p_next_byte - 2) << 8) ^ (*(p_next_byte - 1));
        index = (40543 * index) >> 4;
        index &= 0xFFF;
        this_queue = &hash[index][0];

        /* Find the longest match in the 16 recent positions.          */
        /* Note: Because we are only calculating entropy here, we don't
*/
        /* actually need to know the number of the best position.       */
        maxlength = 0;
        for (j = 0; j < HASHDEPTH; j++)
        {
           unsigned char* p = this_queue[j];
           for (k = 0; k < MAXMATCH; k++)
              if (p_next_byte[k] != p[k])
                 break;
           if (k > maxlength)
           {
              maxlength = k;
              longest_match_index = j;
           }
```

```
        }

        /* Both literals and copies output one control bit and eight
           data bits. They differ on how much input they consume. */
        if (maxlength == 0) maxlength = 1;
        if (maxlength > MEDMATCH && maxlength < MAXMATCH)
maxlength = MEDMATCH;

        p_next_byte += maxlength;
        bits_in += 8 * maxlength;
        bits_out += 9;

        /* If there was a match of 2 or greater, swap the matching
           element with the one half the distance to the head. */
        if (maxlength > 1)
        {
            unsigned char* t;
            unsigned long half = longest_match_index / 2;
            t = this_queue[half];
            this_queue[half] = this_queue[longest_match_index];
            this_queue[longest_match_index] = t;
        }
```

```
        /* Shift the queue and put the new value in the recent end. */
        if (maxlength < NOSHIFTTHRESH)
        {
           for (j = HASHDEPTH - 1; j > 0; j--)
              this_queue[j] = this_queue[j - 1];
           this_queue[0] = old_p_next_byte;
        }
      }
   }

   close(infile);
   printf("Bytes in  = %lu.\n", bits_in / 8);
   printf("Bytes out = %lu.\n", bits_out / 8);
   printf("Percentage remaining=%5.1f\n", 100.0 * ((float)bits_out) /
((float)bits_in));


   //IZ BITS_OUT U TEXT FILE! bits_out je samo broj koji broji te neke
bitove... treba buffer/array
   FILE* fp = fopen("output.txt", "w");  //w ili wb isto je

   char str[sizeof(bits_out)];
   sprintf(str, "%u", bits_out);
```

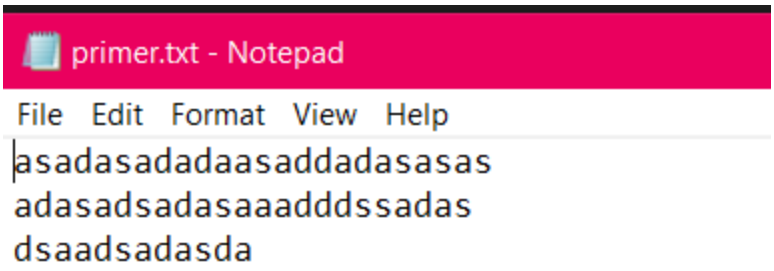
```
   fputs(str, fp);
   fclose(fp);
}

void error(mess)
/* Prints out the string and terminates. */
char mess[];
{
   printf("%s\n", mess);
   exit(0);
}
```

Port.h:

```
#pragma once

#ifndef DONE_PORT       /* Only do this if not previously done.
*/

#ifdef THINK_C
#define UBYTE unsigned char      /* Unsigned byte                */
#define UWORD unsigned int       /* Unsigned word (2 bytes)      */
#define ULONG unsigned long      /* Unsigned word (4 bytes)      */
#define BOOL  unsigned char      /* Boolean                      */
```

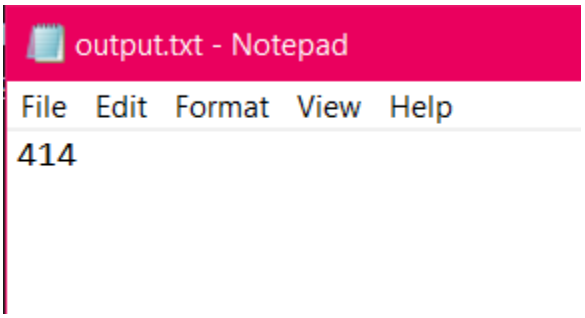
```
#define FOPEN_BINARY_READ  "rb"  /* Mode string for binary reading.
*/
#define FOPEN_BINARY_WRITE "wb"  /* Mode string for binary writing.
*/
#define FOPEN_TEXT_APPEND  "a"   /* Mode string for text appending.
*/
#define REAL double              /* USed for floating point stuff.      */
#endif

#define DONE_PORT                /* Don't do all this again.           */
#define MALLOC_FAIL NULL         /* Failure status from malloc()       */
#define LOCAL static             /* For non-exported routines.         */
#define EXPORT                   /* Signals exported function.         */
#define then                     /* Useful for aligning ifs.           */

#endif
```

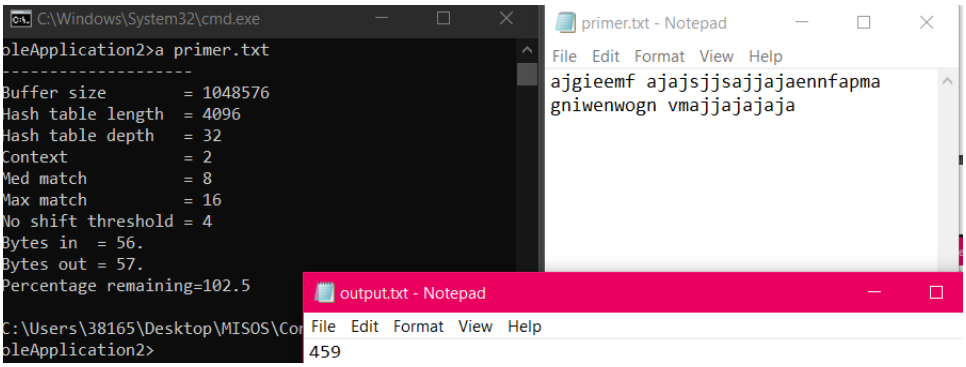
Садржај улазног фајла:



Излаз:

```
--------------------
Buffer size        = 1048576
Hash table length  = 4096
Hash table depth   = 32
Context            = 2
Med match          = 8
Max match          = 16
No shift threshold = 4
Bytes in  = 63.
Bytes out = 51.
Percentage remaining= 82.1
```




C:\Windows\System32\cmd.exe
oleApplication2>a primer.txt

Buffer size   = 1048576
Hash table length   = 4096
Hash table depth   = 32
Context   = 2
Med match   = 8
Max match   = 16
No shift threshold = 4
Bytes in = 56.
Bytes out = 57.
Percentage remaining=102.5
C:\Users\38165\Desktop\MISOS\Co
oleApplication2>
primer.txt - Notepad
File Edit Format View Help
ajgieemf ajajsjjsajjajaennfapma
gniwenwogn vmajjajajaja
output.txt - Notepad
File Edit Format View Help
459

11. Референце:

1. DCC:
https://twitter.com/search?q=%23datacompression%20%28from%3Apowturbo%29&src=typed_query&f=live

2. How to Send a Secret Message from Rome to Paris in the Early Modern Period: Telegraphy between Magnetism, Sympathy, and Charlatanry

https://brill.com/view/journals/esm/27/5/article-p426_3.xml

3. How to Say I Love You in Morse Code

https://the-daily-dabble.com/i-love-you-in-morse-code/

4. Data Compression - The Complete Reference

https://drive.google.com/file/d/18qt5ZIeRwHOOimXVtKVaxSE61M1h0Di6/view

5. [Langdon83] Langdon G.G., "A Note on the Ziv-Lempel Model for Compressing Individual Sequences, IEEE Transactions on Information Theory, Vol.29, No.2, pp.284-287.

6. [Langdon84] Langdon G.G., "On Parsing Versus Mixed-Order Model Structures for Data Compression", IBM Research Report RJ-4163 (46091) 1/18/84, IBM Research Laboratory, San Jose, CA 95193, 1984.

7. [Rissanen81] Rissanen J.J, Langdon G.G, "Universal Modeling and Coding", IEEE Transactions on Information Theory, Vol. 27, No. 1, pp.12-23.

8. [Williams91] Williams R.N., "Adaptive Data Compression", Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061.

9. [Ziv77] Ziv J., Lempel A., "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory", Vol. 23, No. 3, pp. 337-343.

10. [Ziv78] Ziv J., Lempel A., "Compression of Individual Sequences via Variable-Rate Coding", IEEE Transactions on Information Theory, Vol. 24, No. 5, pp. 530-536.

11. Support Code: http://ross.net/compression/support_code.html

12. LZRW4 code: http://ross.net/compression/lzrw4.html