بسمه تعالی

دانشگاه آزاد اسلامی لارستان

دانشکده مهندسی عمران – GIS & RS

مدیریت داده ها در سنجش از دور

فشرده سازی LZW

استاد راهنما : دکتر سراجیان

مترجم : محمد بهجت منش

اردیبهشت ۱۳۹۲

فشرده سازی LZW بعد از توسعه دهندگان آن نامگذاری شده است Lampel و Ziv با اصلاحات بعدی بوسیله Welch . در درجه اول تکنیکی برای اهداف کلی فشرده سازی داده بخاطر ویژگی های سادگی و قابلیت انطباق آن میباشد . بطور معمول، شما می توانید LZW برای فشرده سازی متن، کد اجرایی، فایل ها و داده های مشابه به حدود نصف اندازه اصلی خود انتظار داشته باشیداز .همچنین LZW بخوبی اجرا میشود وقتی با فایل داده بسیار افزونگی ارائه میشود مثل جدول اعداد و کدمنبع کامپیوتری و سیگنالهای اکتسابی. میزان فشرده سازی برای این موارد غالبا 5:1 میباشد.

فشرده سازی LZW همیشه در فایلهای تصویری GIF استفاده میشود و بعنوان یک انتخاب در فرمت TIFF و PostScript پیشنهاد میشود. فشرده سازی LZW تحت شماره ثبت اختراعات آمریکا ۴۵۵۸۳۰۲ در تاریخ ۱۰ دسامبر ۱۹۸۵ محافظت میشود.

فشرده سازی LZW از یک جدول کد که در شکل نشان داده شده استفاده میکند. یک انتخاب رایج این است که ۴۰۹۶ ورودی در جدول ارائه شود. در این حالت داده های کدگذاری شده LZW بطور کامل شامل کدهای ۱۲ بیتی هستند. هرکدام مربوط به یکی از ورودی های جدول کد هستند. عمل رمزگشایی با در نظر گرفتن هر کد از فایل فشرده، و ترجمه آن از طریق جدول کد برای پیدا کردن چه کاراکتر یا کاراکترهایی ارائه شده است، عمل میکند. کد ۰-۲۵۵ در جدول کد همیشه اختصاص یافته به بایت ارائه شده از فایل ورودی. برای مثال، اگر تنها این ۲۵۶ کد اول مورد استفاده قرار گرفت، هر بایت در فایل اصلی می توان به ۱۲ بیت در فایل کد گذاری شده LZW تبدیل شده، و در نتیجه حجم فایل ۵۰ درصد بزرگتر شده است. درطول عمل رمزگشایی هر کدام از کدهای ۱۲ بیتی میتواند از طریق جدول کد به بایتهای تنها ترجمه شود.البته این یک حالت مفید نخواهد بود.

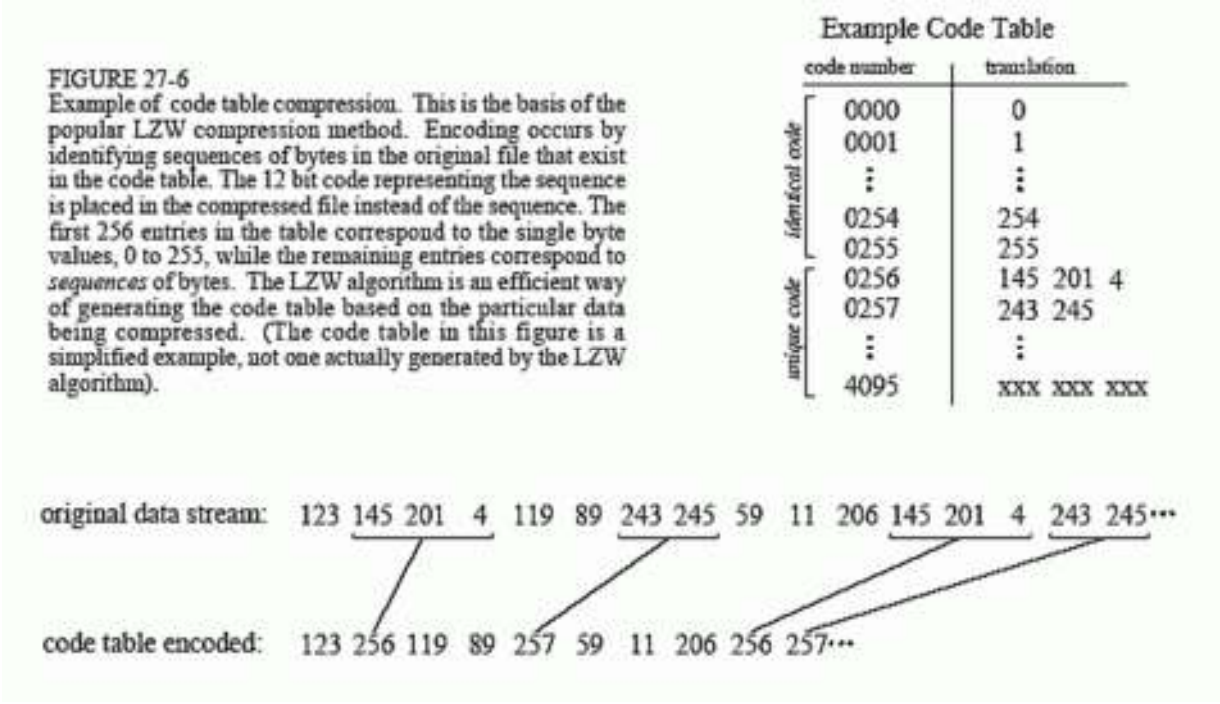

FIGURE 27-6
Example of code table compression. This is the basis of the popular LZW compression method. Encoding occurs by identifying sequences of bytes in the original file that exist in the code table. The 12 bit code representing the sequence is placed in the compressed file instead of the sequence. The first 256 entries in the table correspond to the single byte values, 0 to 255, while the remaining entries correspond to *sequences* of bytes. The LZW algorithm is an efficient way of generating the code table based on the particular data being compressed. (The code table in this figure is a simplified example, not one actually generated by the LZW algorithm).

روش LZW فشرده سازی میکند با استفاده از کدهای ۲۵۶تا ۴۰۹۵ تا توالی بایت ها را ارائه کند.برای مثال کد ۵۲۳ ممکن است توالی سه بایت ۲۳۱ ۱۲۴ ۲۳۴ را ارائه کند. هر موقع الگوریتم فشرده سازی به این توالی در فایل ورودی برخورد کند کد ۵۲۳ در فایل کد گذاری شده جایگزین میشود. در طول عمل رمزگشایی ،کد ۵۲۳ از طریق جدول کد ترجمه میشود تا توالی ۳ بایت صحیح را بازسازی کند. دیگر دنباله اختصاص داده به یک کد واحد دارد، و بیشتر دنباله تکرار می شود، فشرده سازی بالاتر به دست آورد.

اگرچه این یک روش ساده هست ۲ مشکل بزرگ که باید حل شود وجود دارد: ۱- چطور بفهمیم کدام توالی باید در جدول کد باشد و ۲- چطور برنامه رمزگشایی با همان جدول کد استفاده شده بوسیله برنامه فشرده سازی را تهیه کنیم. الگریتم پیچیده LZW هر دو مشکل را حل کرده است.

وقتی برنامه LZW شروع به کدگذاری یک فایل میکند ، جدول کد فقط شامل ۲۵۶ ورودی اول میباشد ،بقیه جدول خالی خواهد بود.این بدان معنی است که کدهای اول وارد فایل فشرده شده اند بایتهای تنها از فایل ورودی هستند که به ۱۲ بیت تبدیل شده اند. همانطور که کد گذاری ادامه پیدا میکند الگوریتم LZW توالی ها (دنباله ها) ی تکرار شده در داده را تشخیص میدهد و آنها را به جدول کد اضافه میکند. در مواجه با یک توالی فشرده سازی بار دوم شروع میشود. نکته کلیدی این است که دنباله ای از فایل ورودی به جدول کد اضافه نمیشود تا زمانی که آن را قبلا در فایل فشرده به عنوان یک کاراکتر منحصر به فرد (کدهای ۰ تا ۲۵۵) قرار می گیرد. این مهم است زیرا به برنامه uncompression اجازه میدهد تا جدول کد به طور مستقیم از داده های فشرده بازسازی شود، بدون نیاز به انتقال جدول کد به طور جداگانه.

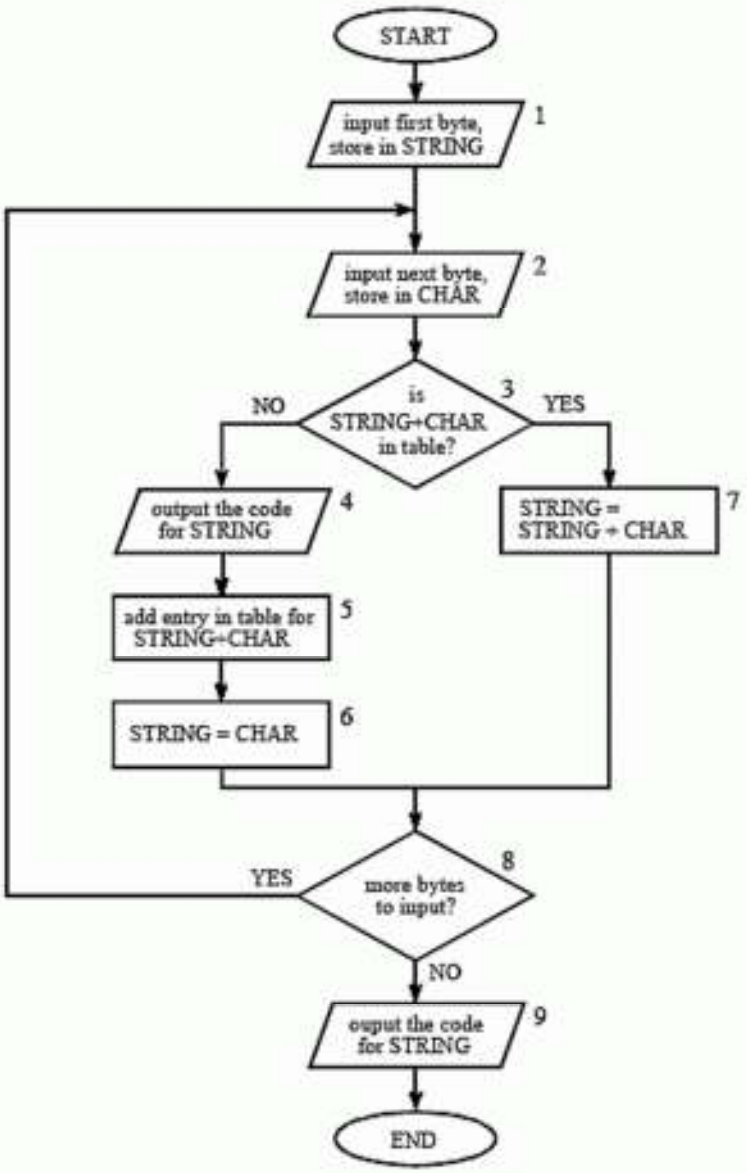

FIGURE 27-7
LZW compression flowchart. The variable, *CHAR*, is a single byte. The variable, *STRING*, is a variable length sequence of bytes. Data are read from the input file (box 1 & 2) as single bytes, and written to the compressed file (box 4) as 12 bit codes. Table 27-3 shows an example of this algorithm.

شکل ۲۷-۷ یک فلوپارت برای فشرده سازی LZW را نشان میدهد. جدول ۲۷-۳ جزئیات مرحله به مرحله برای یک مثال فایل ورودی شامل ۴۵ بایت نشان میدهد ، رشته ASCII text :
the/rain/in/Spain/falls/mainly/on/the/plain

وقتی ما میگوییم الگوریتم LZW کاراکتر a را از فایل ورودی میخواند ، منظور ما این است که مقدار ۰۱۱۰۰۰۰۱ (۹۷ در ۸ بیت بیان میشود) را میخواند ، جائیکه ۹۷ معرف a در ASCII است. وقتی ما میگوییم الگوریتم مینویسد کاراکتر a را به فایل فشرده شده ، منظور ما این است که مینویسد :
۰۰۰۰۰۱۱۰۰۰۰۱ (۹۷ در ۱۲ بیت بیان میشود).

	CHAR	STRING + CHAR	In Table?	Output	Add to Table	New STRING	Comments
1	t	t				t	first character- no action
2	h	th	no	t	256 = th	h	
3	e	he	no	h	257 = he	e	
4	/	e/	no	e	258 = e/	/	
5	r	/r	no	/	259 = /r	r	
6	a	ra	no	r	260 = ra	a	
7	i	ai	no	a	261 = ai	i	
8	n	in	no	i	262 = in	n	
9	/	n/	no	n	263 = n/	/	
10	i	/i	no	/	264 = /i	i	
11	n	in	yes (262)			in	first match found
12	/	in/	no	262	265 = in/	/	
13	S	/S	no	/	266 = /S	S	
14	p	Sp	no	S	267 = Sp	p	
15	a	pa	no	p	268 = pa	a	
16	i	ai	yes (261)			ai	matches *ai*, *ain* not in table yet
17	n	ain	no	261	269 = ain	n	*ain* added to table
18	/	n/	yes (263)			n/	
19	f	n/f	no	263	270 = n/f	f	
20	a	fa	no	f	271 = fa	a	
21	l	al	no	a	272 = al	l	
22	l	ll	no	l	273 = ll	l	
23	s	ls	no	l	274 = ls	s	
24	/	s/	no	s	275 = s/	/	
25	m	/m	no	/	276 = /m	m	
26	a	ma	no	m	277 = ma	a	
27	i	ai	yes (261)			ai	matches *ai*
28	n	ain	yes (269)			ain	matches longer string, *ain*
29	l	ainl	no	269	278 = ainl	l	
30	y	ly	no	l	279 = ly	y	
31	/	y/	no	y	280 = y/	/	
32	o	/o	no	/	281 = /o	o	
33	n	on	no	o	282 = on	n	
34	/	n/	yes (263)			n/	
35	t	n/t	no	263	283 = n/t	t	
36	h	th	yes (256)			th	matches *th*, *the* not in table yet
37	e	the	no	256	284 = the	e	*the* added to table
38	/	e/	yes			e/	
39	p	e/p	no	258	285 = e/p	p	
40	l	pl	no	p	286 = pl	l	
41	a	la	no	l	287 = la	a	
42	i	ai	yes (261)			ai	matches *ai*
43	n	ain	yes (269)			ain	matches longer string *ain*
44	/	ain/	no	269	288 = ain/	/	
45	EOF	/		/			end of file, output *STRING*

TABLE 27-3
LZW example. This shows the compression of the phrase: *the/rain/in/Spain/falls/mainly/on/the/plain/*.

الگوریتم فشرده سازی از دو متغیر استفاده میکند: کاراکتر(CHAR)و رشته(STRING). متغیر CHAR یک کاراکتر را به تنهایی نگه میدارد بدان معنی که یک بایت تنها بین ۰ تا ۲۵۵. متغیر STRING یک متغیر رشته طولی است بدان معنی که یک گروه از یکی یا تعداد بیشتری کاراکتر و اینکه هر کاراکتر یک بایت

تنها است. در باکس ۱ شکل ۷-۲۷ برنامه با گرفتن اولین بایت از فایل ورودی شروع میشود و آن را در متغیرSTRING قرار میدهد. جدول ۳-۲۷ این عمل را در خط اول نشان میدهد. این کار توسط الگوریتم چرخه ای برای هر بایت اضافه در فایل ورودی دنبال میشود ، کنترل شده در دیاگرام جاری بوسله باکس ۸. هروقت یک بایت از فایل ورودی(باکس۲) خوانده شود،آن در متغیر CHAR ذخیره میشود. جدول داده سپس جستجو میکند تا تعیین کند اگر زنجیره ای از دو متغیر STRING+CHAR در حال حاضر یک کد اختصاص داده شده است.

اگر انطباقی در جدول کد دیده نشد سه عمل انجام میشود همانطور که در باکسهای ۴و۵و۶ نشان داده شده است. در باکس ۴ کد ۱۲ بیتی مرتبط با محتوای متغیر STRING به فایل فشرده سازی نوشته میشود.در باکس ۵ یک کد جدید در جدول ساخته میشود برای زنجیره ای از STRING+CHAR. در باکس ۶ متغیر STRING مقدار متغیر CHAR را میدهد.یک مثال از این اعمال در خط ۲ تا ۱۰ در جدول ۳-۲۷ نشان داده شده است(برای ۱۰ بایت اول فایل مثال).

وقتی یک انطباق در جدول کد پیدا میشود (باکس ۳) زنجیره STRING+CHAR ذخیره شده در متغیر STRING بدون هیچ عمل دیگری قرار میگیرد(باکس ۷). اگر یک توالی منطبق در جدول پیدا شد نباید عملی انجام گیرد حتی اگر یک توالی منطبق بلندتر در جدول وجود داشته باشد. مثالی از این در خط ۱۱ نشان داده شده جائیکه توالی in= STRING+CHAR تشخیص داده شده هم اکنون در جدول یک کد دارد.در خط ۱۲ کراکتر بعدی از فایل ورودی / به توالی اضافه میشود و جدول کد برای /in جستجو میکند . از آنجائیکه این دنباله طولانی تر در جدول وجود ندارد برنامه آن را به جدول اضافه میکند ،خروجی کد برای دنباله کوچکتر که در جدول هست (کد۲۶۲) و شروع به جستجو میکند برای دنباله ای که شروع شود با کاراکتر / . این جریان رویداد ادامه میابد تا کاراکتر دیگری در فایل ورودی وجود نداشته باشد. این برنامه با کد مربوط به مقدار فعلی STRING نوشته شده به فایل فشرده (همانطورکه در باکس ۹ شکل ۷-۲۷ نشان داده شده است و خط ۴۵ جدول ۲۷-۳) پیچیده می شود. فلوچارت الگوریتم رمزگشایی LZW در شکل ۸-۲۷ نشان داده شده است . هر کد از فایل فشرده خوانده میشود و با جدول کد مقایسه میشود تا ترجمه شود. همانطور که هر کد به این شیوه پردازش میشود جدول کد بروز میشود تا اینکه بطور مداوم یکی که استفاده شده را انطباق دهد. بهرحال یک مشکل کوچک در فرایند رمزگشایی وجود دارد . ترکیبات خاصی از داده وجود دارد که در الگوریتم رمزگشایی نتیجه شده اند ،کدی را دریافت میکنند که هنوز در جدول کد وجود ندارد. این احتمالی است که در جعبه های ۴،۵ و ۶ گرفته شده است. تنها چند

ده خط کد برای ابتدایی ترین برنامه های LZW مورد نیاز است . مشکل واقعی نهفته در مدیریت کارآمد جدول کد است. رویکرد نیروی بی رحم نتایج در حافظه بزرگ مورد نیاز است و اجرای کند برنامه. به منظور بهبود عملکرد آنها چند ترفند در برنامه های LZW تجاری مورد استفاده قرار گیرد.

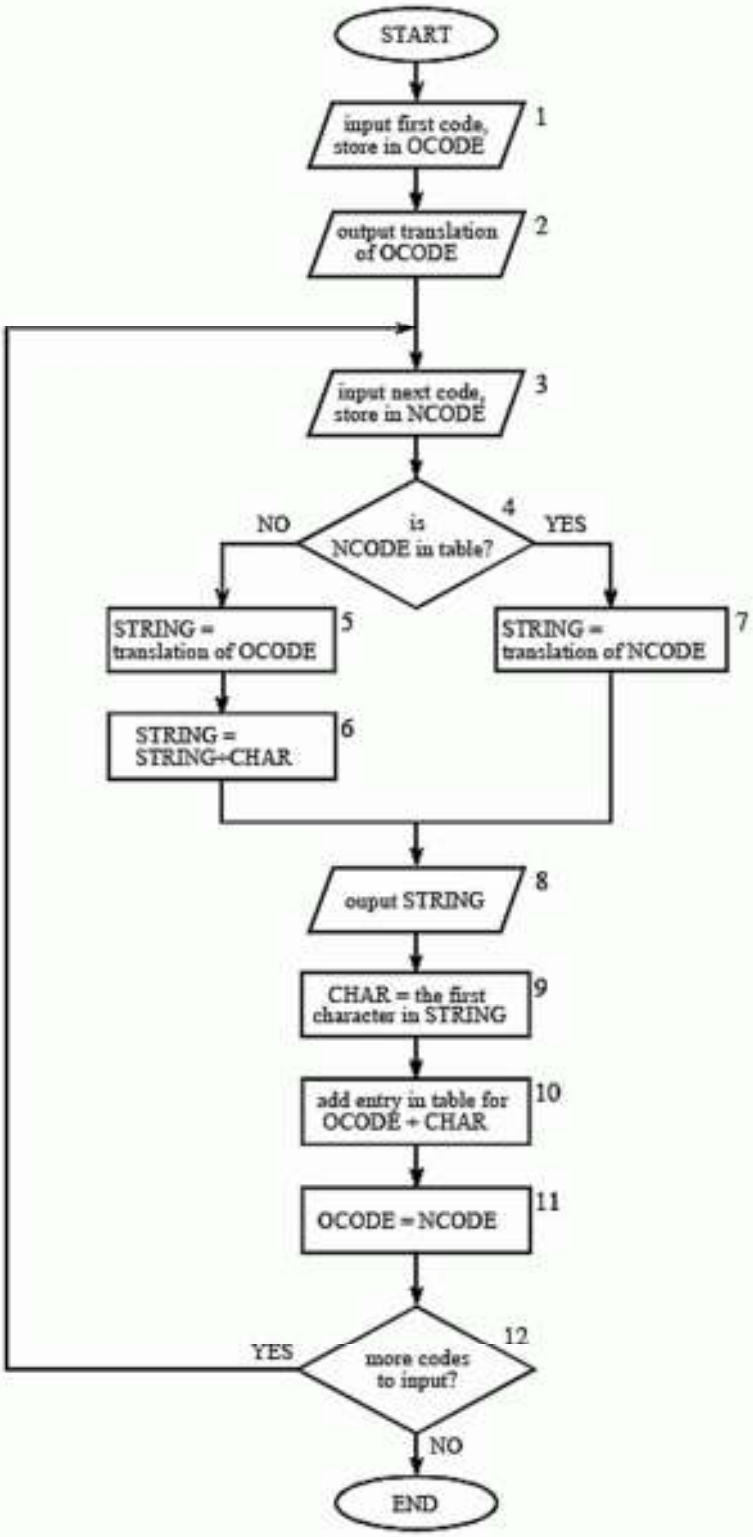

FIGURE 27-8
LZW uncompression flowchart. The variables, *OCODE* and *NCODE* (oldcode and newcode), hold the 12 bit codes from the compressed file, *CHAR* holds a single byte, *STRING* holds a string of bytes.

به عنوان مثال، مشکل حافظه بوجود می آید زیرا نمیداند چقدر طول هر رشته کاراکتر قبل از هر کد خواهد بود. بیشتر برنامه های LZW این کار را با استفاده از جدول کد انجام میدهند. برای مثال به خط ۹ در جدول ۳-۲۷ نگاه کنید جاییکه کد ۲۷۸ تعریف میشود تا ainl باشد. بجای اینکه ایت ۴ بایت را ذخیره کند ، کد ۲۷۸ ذخیره خواهد شد بعنوان : کد 269+l ، جاییکه کد ۲۶۹ قبلا بعنوان ain در خط ۱۷ تعریف شده بود. به همین ترتیب کد ۲۶۹ بعنوان کد 261+n ذخیره میشود ، جاییکه کد ۲۶۱ قبلا بعنوان ai در خط ۷ تعریف شده بود. این الگو همیشه حفظ خواهد شد: هر کد می تواند به عنوان یک کد قبلی به علاوه یک کاراکتر جدید بیان شود.

زمان اجرای الگوریتم فشرده سازی محدود به جستجو در جدول کد میشود تا تعیین کند آیا یک انطباق وجود دارد. به عنوان مثال، تصور کنید شما می خواهید پیدا کنید اگر نام یکی از دوستان در دفترچه راهنمای تلفن ذکر شده است. فقط دفترچه ای که شما در دسترس دارید براساس شماره تلفن مرتب شده نه براساس حروف الفبا. این نیاز است که شما صفحات را جستجو کنید تا نامی که میخواهید را پیدا کنید. این وضعیت ناکارآمد دقیقا همان جستجوی تمام ۴۰۹۶ کد برای انطباق با یک رشته کاراکتر خاص است. پاسخ: سازماندهی جدول کد به طوری که آنچه شما دنبال آن هستید به شما می گوید که به کجا نگاه کنند (مانند یک راهنمای تلفن نیمه الفبایی). به عبارت دیگر، ۴۰۹۶ کد به مکان های توالی در حافظه اختصاص نمی دهد. در عوض، تقسیم حافظه به بخشهایی بر اساس آنچه که توالی ذخیره می شود خواهد شد. برای مثال، فرض کنید ما می خواهیم پیدا کنیم اگر دنباله: کد ۳۲۹ + X، در کد جدول وجود دارد. جدول کد باید سازماندهی شود، به طوری که "X" نشان می دهد جاییکه باید جستجو از آنجا شروع شود. طرح های زیادی را برای این نوع از مدیریت جدول کد وجود دارد، و آنها می تواند کاملا پیچیده باشند.

این به ارمغان می آورد آخرین اظهار نظر در مورد برنامه های فشرده سازی LZW و مشابه آن است که زمینه بسیار رقابتی است. در حالی که اصول فشرده سازی داده ها نسبتا ساده است، انواع برنامه های فروخته شده به عنوان محصولات تجاری بسیار پیچیده است. شرکت های پولدار میشوند با فروش برنامه هایی به شما که فشرده سازی می کنند، و حفاظت از اسرار تجاری خود را از طریق اختراع ثبت شده و مانند آن. انتظار نداریم که به همان سطح از عملکرد این برنامه ها در چند ساعت کاری برسیم.