KARADENİZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
BİLGİSAYAR SİSTEMLERİ LABORATUARI

Veri Sıkıştırma Yöntemleri

1. Giriş

Veri sıkıştırma (data compression) verilerin daha küçük boyutlarda saklanması olarak tanımlanabilir. Burda sıkıştırılacak veriler herhangi bir dosyadan, bellekten veya haberleşme kanalından geliyor olabilir. 1980'ler ve 1990'lar veri kaydetme ortamlarında fiyat olarak üstel bir düşüşün yaşanmasına karşın veri sıkıştırma yöntemleri üzerine yapılan araştırmaların yoğun olarak yapıldığı yıllar olmuştur. Bunun temel nedenleri şöyle sıralanabilir:

- Çoğu kişi gereksiz de olsa verileri depolamaktan hoşlanmaktadır. Bu durumda kısa sürede kullanılan disk gereksiz veriler yüzünden dolmaktadır. Veri sıkıştırma bunu geciktirdiği için faydalıdır.
- Veri transferinin hızlı olması istenir. O yüzden gönderilmeden önce sıkıştırılmalıdır.
- Son yıllarda veri depolama kapasitesinde üstel bir artış olsa da depolama yapan ünitenin hızında yeterli miktarda artış henüz söz konusu değildir.

Veri sıkıştırma yöntemleri farklı temellere, farklı tipte verilere göre farklı sonuçlar üretse de temelde hepsi "gereksiz (redundant) verileri yok etme" prensibine dayanmaktadır. Örneğin kelimelerde 'E' harfine 'Z' den daha çok rastlanmaktadır. Buna alfabetik gereksizlik (alphabetic redundancy) denir ve veri sıkıştırma yaparken sık rastlanan harf olan 'E' ye en kısa; 'Z' ye de en uzun kod verilmelidir. Dolayısıyla gereksiz verileri yok ederek veri sıkıştırmada genel kural "**sık** tekrarlanan verilere **kısa**; **az** tekrarlanlara da **uzun** kodlar vermek" 'tir. Yüzlerce veri sıkıştırma yöntemi 4 temel sınıf altında toplanabilir [1]:

1. Tekrarlama Sayısı Kodlama (Run Length Coding).
2. İstatiksel Yöntemler (Statistical Methods).
3. Sözlük-Tabanlı Yöntemler (Dictionary-Based Methods).
4. Dönüşümler (Transforms).

Bu deneyde sözlük-tabanlı yöntemlerden **`LZW`** incelenecektir.

2. LZW Yöntemi

Genel olarak sözlük tabanlı veri sıkıştırma yöntemleri, text içeren dosyalar kullanırlar. Bu yöntemlerden **`LZW`**, ilk olarak 1978 yılında A. Lempel ve J. Ziv tarafından ortaya atılan LZ78 isimli yöntemin 1984 yılında T. Welch tarafından geliştirilmiş halidir. UNIX işletim sistemindeki **`compress`** emri bu yönteme göre çalışmaktadır.

2.1. LZW Kodlama (Encoding)

Veri sıkıştırma işlemine kodlama (encoding), sıkıştırılmadan önceki halini elde etmeye de kod çözme (decoding, expanding) denir. LZW kodlama işlemi boyunca "dictionary" adı verilen bir tablo tutulur. Tablonun ilk 256 indisinde (0..255) ASCII tablodaki karakterler tutulur. Veriler sıkıştırılacak giriş (input) dosyasından byte byte okunup `I` ile temsil edilen stringe eklenirken dictionary tablosunda bu string aranır. Her bir byte (karakter) eklendikçe tabloda `I`'nın son hali bulunarak ilerlenir. Herhangi bir noktada `I` tabloda varken ona eklenen `x` ile temsil edilen byte ile birlikte `Ix` tabloda olmayabilir. Bu durumda **`I` stringinin dictionary tablosundaki indis değeri** sıkıştırılmış verilerin tutulduğu çıkış (output) dosyasını yazılır. `Ix` tabloya eklenir. Kodlama işlemine `x` ile devam edilir. `x` 0-255 arası bir değere sahip olduğundan tabloda vardır. Giriş dosyasından yeni okunacak `y` isimli byte `x`'e eklenir ve `xy` için aynı işlemler tekrarlanır.

"`sir□sid□eastman□easily□teases□sea□sick□seals`" verisinin (□ : space-boşluk) LZW yöntemine göre nasıl kodlandığı Tablo-1'de gösterilmiştir [1]. Kodlamaya giriş dosyasından ilk karakter `I=s` olarak okunarak başlanmıştır. `s` dictionary tablosunun ilk 256 indisinde tutulan ASCII tabloda olduğundan `Y` (Yes) cevabı alınmıştır. Arkasından okunan `i` karakteri `I`'ya eklenmiş ve `I=si` tekrar dictionary tablosundan sorgulanmıştır. Olmadığı için `N` (No) cevabı alınmış, `si` dictionary tablosunda `256.` indise `new entry` olarak eklenmiş, `s` çıkış dosyasına yazılıp `i` ve ardından gelen `r` için aynı işlemler tekrarlanmıştır. Tablo-1'in 10. satırında tekrar `I=si` olduğu görülür. `si` yukarıda dictionary tablosuna eklendiğinden `Y` cevabı alınmıştır. Sonra gelen `d` karakteri ile birlikte `I=sid` dictionary tablosunda olmadığından `260.` indise eklenmiş, `si` çıkış dosyasına yazılmış ve dosyadan okunan □ ile birlikte `I=d□` için aynı işlemler tekrarlanmıştır.

Çıkış dosyasına kodlar (indisler) 12'şer bit yazıldığı varsayıldığında 73. satırın output sütunundaki `281(□se)` kodu ile □, `s` ve `e` olmak üzere 3 karakter yani 3 byte (24 bit) yerine `□se` stringini temsil eden `281` kodu 12 bit olarak çıkışa yazılmıştır. Bu 12 bitin pratikte dosyaya byte'a çevrilmeksizin yazılması imkansızdır. Dolayısıyla peşpeşe gelen iki 12 bitlik kod 3 adet 8 bitlik veri olarak kodlanır. Tablo-1'in 79. satırında `s,eof` dictionary tablosunda olmadığından `s` çıkış dosyasına yazılarak kodlama işlemi tamamlanmıştır. Sıkıştırılmış dosyanın kodu çözülürken son veri okunduktan sonra işlemin sonlanması için kodlarken, dosyanın sonuna gelindiğini gösteren özel bir `eof` sembolü koymakta fayda vardır. `eof` sembolü olarak 12 bitlik kodlamada 4095 kullanılabilir. Bu durumda dictionary tablosuna eklenen kodlar için 4095 kodu kullanılmamalıdır. Tablo-1'de output sütunundaki 35 adet kod (indis) çıkış (output) dosyasına yazılır. Veriler 12 bitle kodlandığında sıkıştırılmış dosyanın boyutu `35x12+12 = 432` bit = `54` byte olur. `35x12+12` ifadesindeki son `+12` bit **`eof`** sembolü içindir. Veri sıkıştırma oranı (compression ratio) **`output/input`** ile hesaplanır. Örnek veri için `54/44=1.23` 'tür. Oran `1`'den büyük çıktığı için veri sıkıştırma yapılamamıştır.

Dictionary tablosunun ilk 256 elemanı ASCII tablo için ayrılmış olduğundan ve veri sıkıştırmanın gerçekleşebilmesi tabloya en az iki karakterlik stringler eklemeye dayandığından çıkış dosyasına yazılacak her bir kodun boyu en az 9 bit olmalıdır. Bu durumda tablo boyu 2^9`=512` olacağından 256 tane yeni string eklenebilir. Tabloya 256. indisten itibaren eklenmiş herhangi bir `I` stringi dosyanın geri kalan kısmında tekrarlanınca ona eklenecek `x` karakteri ile tabloda olmayan `Ix` new entry olarak tabloya eklenecek bir karakter eksiği `I`'nın indisi çıkış dosyasına yazılacaktır. Demek ki tabloya yeni stringler eklemek veri sıkıştırma açısından yeterli değildir. Bu string dosyanın geri kalan kısmı kodlanırken tekrarlanmalı ve 1 karakter fazlası tabloya eklenerek indisi çıkışa yazılmalıdır. Çıkışa yazılan indisteki stringin karakter sayısı ne kadar fazlaysa veri sıkıştırma o oranda olumlu etkilenecektir. Tablo boyu $\mathbf{2^{kod\_boyu}}$ olduğundan kod boyunu arttırarak tablonun genişletilmesi yeni stringler eklenmesi açısından iyi olsa bile herbir indis çıkışa kod boyu kadar yer tutacak şekilde yazılacağından en uygun (optimum) kod boyunun seçilmesi veri sıkıştırma açısından önemlidir. Föyün Deney Tasarımı ve Uygulaması bölümünde optimum kod boyu (`BITS`) belirlemeye yönelik bir yöntem önerilecek ve bu yöntemi gerçekleyen kod yazdırılacaktır.

	I	in dictionary?	new entry	output		I	in dictionary?	new entry	output
1	s	Y			41	y□	N	274-y□	121(y)
2	si	N	256-si	115(s)	42	□	Y		
3	i	Y			43	□t	N	275-t□	32(□)
4	ir	N	257-ir	105(i)	44	t	Y		
5	r	Y			45	te	N	276-te	116(t)
6	r□	N	258-r□	114(r)	46	e	Y		
7	□	Y			47	ea	Y		
8	□s	N	259-□s	32(□)	48	eas	N	277-eas	263(ea)
9	s	Y			49	s	Y		
10	**si**	**Y**			50	se	N	278-se	115(s)
11	sid	N	260-sid	256(si)	51	e	Y		
12	d	Y			52	es	N	279-es	101(e)
13	d□	N	261-d□	100(d)	53	s	Y		
14	□	Y			54	s□	N	280-s□	115(s)
15	□e	N	262-□e	32(□)	55	□	Y		
16	e	Y			56	□s	Y		
17	ea	N	263-ea	101(e)	57	□se	N	281-□se	259(□s)
18	a	Y			58	e	Y		
19	as	N	264-as	97(a)	59	ea	Y		
20	s	Y			60	ea□	N	282-ea□	263(ea)
21	st	N	265-st	115(s)	61	□	Y		
22	t	Y			62	□s	Y		
23	tm	N	266-tm	116(t)	63	□si	N	283-□si	259(□s)
24	m	Y			64	i	Y		
25	ma	N	267-ma	109(m)	65	ic	N	284-ic	105(i)
26	a	Y			66	c	Y		
27	an	N	268-an	97(a)	67	ck	N	285-ck	99(c)
28	n	Y			68	k	Y		
29	n□	N	269-n□	110(n)	69	k□	N	286-k□	107(k)
30	□	Y			70	□	Y		
31	□e	Y			71	□s	Y		
32	□ea	N	270-□ea	262(□e)	72	□se	Y		
33	a	Y			**73**	□sea	N	287-□sea	**281(□se)**
34	as	Y			74	a	Y		
35	asi	N	271-asi	264(as)	75	al	N	288-al	97(a)
36	i	Y			76	l	Y		
37	il	N	272-il	105(i)	77	ls	N	289-ls	108(l)
38	l	Y			78	s	Y		
39	ly	N	273-ly	108(l)	**79**	**s,eof**	**N**		**115(s)**
40	y	Y							

Tablo-1 : “sir□sid□eastman□easily□teases□sea□sick□seals” verisinin kodlanışı

0	NULL	**262**	**□e**	**276**	**te**
1	SOH	**263**	**ea**	**277**	**eas**
...		**264**	**as**	**278**	**se**
32	□	**265**	**st**	**279**	**es**
...		**266**	**tm**	**280**	**s□**
97	A	**267**	**ma**	**281**	**□se**
98	B	**268**	**an**	**282**	**ea□**
...		**269**	**n□**	**283**	**□si**
256	**si**	**270**	**□ea**	**284**	**ic**
257	**ir**	**271**	**asi**	**285**	**ck**
258	**r□**	**272**	**il**	**286**	**k□**
259	**□s**	**273**	**ly**	**287**	**□sea**
260	**sid**	**274**	**y□**	**288**	**al**
261	**d□**	**275**	**□t**	**289**	**ls**

Tablo-2 : "`sir□sid□eastman□easily□teases□sea□sick□seals`" için dictionary tablosu

Dictionary tablosunun ilk 256 elemanın bir kısmı ve örnek veri kodlanırken oluşturulan yeni elemanların (`new entry`) tamamı (**bold** olarak) Tablo-2'de verilmiştir.

LZW programındaki **`compress()`** fonksiyonu kodlama yapmaktadır [2]. Fonksiyondaki **`while`** döngüsü içindeki **`if(code_value[index]!=-1)`** şartı sağlanıyorsa okunan veri dictionary tablosunda var demektir. **`if(next_code<=MAX_CODE)`** şartı da dictionary tablosunda olmayan veriyi **`next_code`** ile tabloya eklemeye çalışır. Eğer bu şart sağlanmıyorsa tablo dolu demektir. **`MAX_CODE`**, 12 bitlik kodlama için 4094'tür (4095, `eof` sembolü için ayrılmıştı).

`Debug` klasöründeki **`LZW.exe`** konsoldan şu şekilde koşulmalıdır :

```
LZW c/d input output
```

Burada **`c`** compress, **`d`** decompress demektir. Kodlamada **`c`**, kod çözmede **`d`** yazılmalıdır. Kodlamada input, sıkıştırılacak dosya; output da sıkıştırılmış verilerin yazılacağı dosyadır. Kod çözülürken tersi söz konusudur.

`LZW.exe`'nin Windows altında nasıl koşulacağı deneyin web sayfasındaki videoda anlatılmaktadır. Linux için buraya bakınız. Programı derlemede veya exeyi koşmada sorun yaşarsanız **deney sorumlusu ile görüşünüz**.

2.2. LZW Kod Çözme (Decoding, Expanding)

Kod çözme işleminde veriler sıkıştırılırken yapılan işlemlerin benzeri yapılır. Sıkıştırılmış örnek "`sir□sid□eastman□easily□teases□sea□sick□seals`" verisinin kodu çözülürken dosyadan ilk veri `s` okunur ve `I=s` olur. Dictionary tablosuna bakılır. İlk 256 eleman ASCII karakterler için ayrıldığından `s` tabloda vardır. İkinci veri `i` okunur ve `I=si` tabloda aranır. Tabloda olmadığından eklenir. Çözülmüş kodun yazılacağı çıkış dosyasına (output) `s` yazılır. Şimdi `I=i` olur ve tabloda olduğu için üçüncü veri `r` okunarak `I=ir` olur. `ir` dictionary tablosunda olmadığından eklenir ve `i` çıkışa yazılır. `I=r` tabloda olduğundan okunan veri □ ile birlikte `I=r□` tabloda aranır. Olmadığı için eklenir ve r çıkışa yazılır. `I=□` tabloda olduğundan okunan veri `s` ile birlikte `I=□s` tabloda aranır. Olmadığı için eklenir ve □ çıkışa yazılır. `I=s` tabloda olduğundan okunan veri `i` ile birlikte `I=si` tabloda aranır. Yukarıda ilk iki veri okunurken `si` tabloya eklenmişti. `I=si` tabloda olduğundan yeni bir veri `d` okunur ve `I=sid` olur. `sid` tabloda olmadığından eklenir ve çıkışa `si` yazılır. `I=d` ve ona eklenecek □ için aynı işlemler tekrarlanır.

3. Deney Hazırlığı

- **aaa...** şeklinde peşpeşe gelen **n** tane **a** harfi **x** bytelık veriler şeklinde LZW kodlandığında (**eof** sembolü de dikkate alınırsa) çıkış dosyası kaç byte olur?
- Çıkış dosyasının boyu **m** byte olduğunda max. kaç tane **a** harfi giriş dosyasına yazılabilir?

Kaynak kodlardaki **aaa** adlı şablon projeyi bu iki soruyu çözecek şekilde tamamlayınız.
İpucu → Handbook of Data Compression kitabında **Exercise 6.11** 'e bakınız.
Solutions klasöründeki **aaa.exe** doğru değerleri üretmektedir. Programınızın doğru çalışıp/çalışmadığını test ederken bu exeden yardım alabilirsiniz. **aaa.exe** VS2017 ile üretilmiştir. Bilgisayarınızda başka bir Visual Studio versiyonu kurulu ise **aaa.exe** koşmayabilir. VS 2017 kurulu olduğu halde programları derlemekte/koşmakta sorun yaşarsanız bu video faydalı olabilir.

4. Deney Tasarımı ve Uygulaması

LZW kodlamada en iyi veri sıkıştırma oranını verecek optimum **BITS** değerini (kod boyunu) bulmaya yönelik olarak **Optimum** adlı şablon bir proje yazılmıştır. Projeye, LZW programındaki **compress()** fonksiyonunda bir takım değişiklikler yapılarak elde edilen **compTest()** fonksiyonu eklenmiştir. Fonksiyon koşarken giriş dosyasının sonuna gelinmeden dictionary tablosu dolarsa `true`; dosyanın tamamı kodlandığı halde dictionary tablosunda boşluk kalırsa `false` döndürülmektedir.

Optimum kod boyunu bulmak için 12 gibi düşük bir başlangıç **BITS** değeri ile çağırılan **compTest()** fonksiyonu `true` döndürdükçe **BITS++** yapılıp `false` döndürene kadar tekrar çağırılır. `false` döndüğü andaki yani dictionary tablosunda boşluğa ilk rastlanan değer veya 1 eksiği yani tablonun son dolduğu değer **optimum BITS** değeri olarak alınır ve **bits.txt** adlı dosyaya yazılır. Giriş dosyası bu değere göre kodlanır. Dosya **bits.txt**'den okunan değere göre **expand()** ile açılır.

Kaynak kodlardaki **Optimum** adlı projeyi tamamlayınız. Programınızın doğru çalışıp çalışmadığını test etmek için **Solutions** klasöründeki **Optimum.exe** ile karşılaştırabilirsiniz. Oradaki **Optimum.exe** tablonun son dolduğu değeri optimum **BITS** olarak alır. **Optimum.exe**, **LZW.exe** gibi konsoldan koşulur. C dilinde **fopen/fscanf/fprintf** gibi temel dosya açma/okuma/yazma fonksiyonlarını bildiğiniz varsayılmaktadır. Eksiğiniz varsa öğrenip geliniz.

5. Deney Raporu

Kaynak kodlardaki **Rapor.docx** adlı şablon belge **grup adına** aşağıdaki gibi hazırlanıp deney hazırlığı **aaa.cpp** ile birlikte hem **deney saatine kadar** 61omercakir@gmail.com 'a e-mail ile yollanacak hem de yazıcı çıktısı alınıp **deneye getirilecektir** :

- Tablo-1'de ismi verilen dosyalar için **Solutions** içindeki **Optimum.exe** ile optimum **BITS** değerlerini bulup **BITS, BITS+1** ve **BITS+2** için veri sıkıştırma oranlarını tabloya yazınız.
  Uyarı → **BITS** için veri sıkıştırma oranını **Optimum.exe** ile hesaplamak mümkünken **BITS+1** ve **BITS+2** için **LZW** programını bu değerlere göre derleyip koşmalısınız. Ayrıca **Optimum.exe** en iyi veri sıkıştırma oranını Dictionary tablosunun son dolduğu andaki **BITS** değerinin verdiğini varsaymaktadır ki Tablo-1'deki bazı dosya(lar) için bu doğru olmayabilir. Bazı dosyalarda en iyi oran Dictionary tablosunun ilk boş kaldığı değer (**BITS+1**) de olabilir.
- Deney sorularının cevaplarını ilgili yerlere yazınız.
- Deney grubunuzla ilgili olanı veriyi Tablo-2'ye kodlayınız.

6. Kaynaklar

[1] Handbook of Data Compression, D. Salomon, G. Motta, 5th Edititon, 2010.
[2] http://marknelson.us/1989/10/01/lzw-data-compression/