İçindekiler

VERİ SIKIŞTIRMA

SORU-1: Huffman Sıkıştırması hakkında bilgi veriniz.

Huffman algoritması, bir veri kümesinde daha çok rastlanan sembolü daha düşük uzunluktaki kodla, daha az rastlanan sembolleri daha yüksek uzunluktaki kodlarla temsil etme mantığı üzerine kurulmuştur. Bir örnekten yola çıkacak olursak : Bilgisayar sistemlerinde her bir karakter 1 byte yani 8 bit uzunluğunda yer kaplar. Yani 10 karakterden oluşan bir dosya 10 byte büyüklüğündedir. Çünkü her bir karakter 1 byte büyüklüğündedir. Örneğimizdeki 10 karakterlik veri kümesi "aaaaaaaccs" olsun. "a" karakteri çok fazla sayıda olmasına rağmen "s" karakteri tektir. Eğer bütün karakterleri 8 bit değilde veri kümesindeki sıklıklarına göre kodlarsak veriyi sembolize etmek için gereken bitlerin sayısı daha az olacaktır. Söz gelimi "a" karakteri için "0″ kodunu "s" karakteri için "10″ kodunu, "c" karakteri için "11″ kodunu kullanabiliriz. Bu durumda 10 karakterlik verimizi temsil etmek için

(a kodundaki bit sayısı)*(verideki a sayısı) + (c kodundaki bit sayısı)*(verideki c sayısı) + (s kodundaki bit sayısı)*(verideki s sayısı) = 1*7 + 2*2 + 2*1 = 12 bit

gerekecektir. Halbuki bütün karakterleri 8 bit ile temsil etseydik 8*10 = 80 bite ihtiyacımız olacaktı. Dolayısıyla %80 'in üzerinde bir sıkıştırma oranı elde etmiş olduk.

Huffman tekniğinde semboller(karakterler) ASCII'de olduğu gibi sabit uzunluktaki kodlarla kodlanmazlar. Her bir sembol değişken sayıda uzunluktaki kod ile kodlanır.

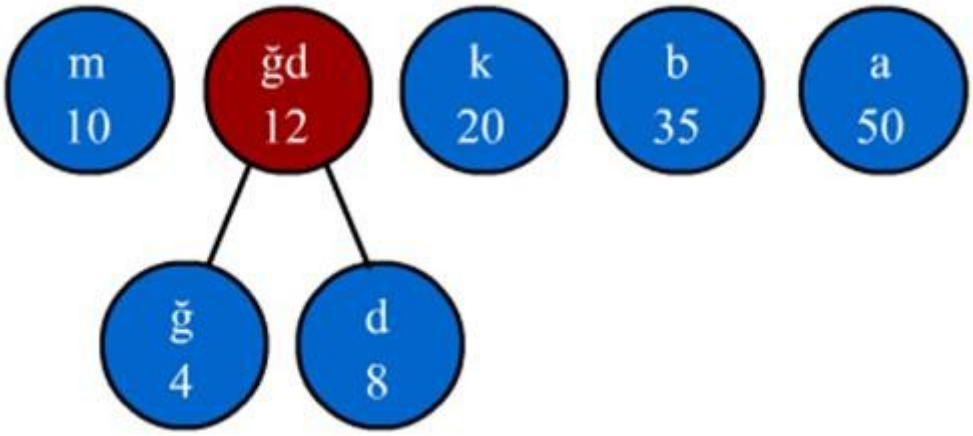
ÖRNEK:

(elimizde bir metin dosyası olsun ve metin dosyasında geçen karakerler karakter sayısına göre frekans tablosu oluşturulsun.aşağıdaki gibi a harfinden 50 tane b den 35 tane gibi)

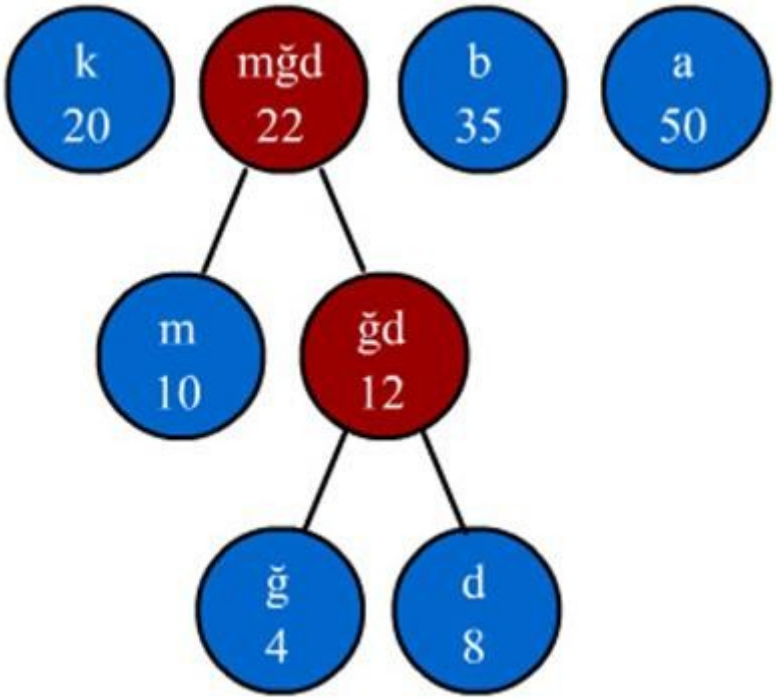
Sembol(Karakter)	Sembol Frekansı
A	50
B	35
K	20
M	10
D	8
Ğ	4

Amaç: her bir karakteri hangi bit dizileriyle kodlayacağımızı bulmak.

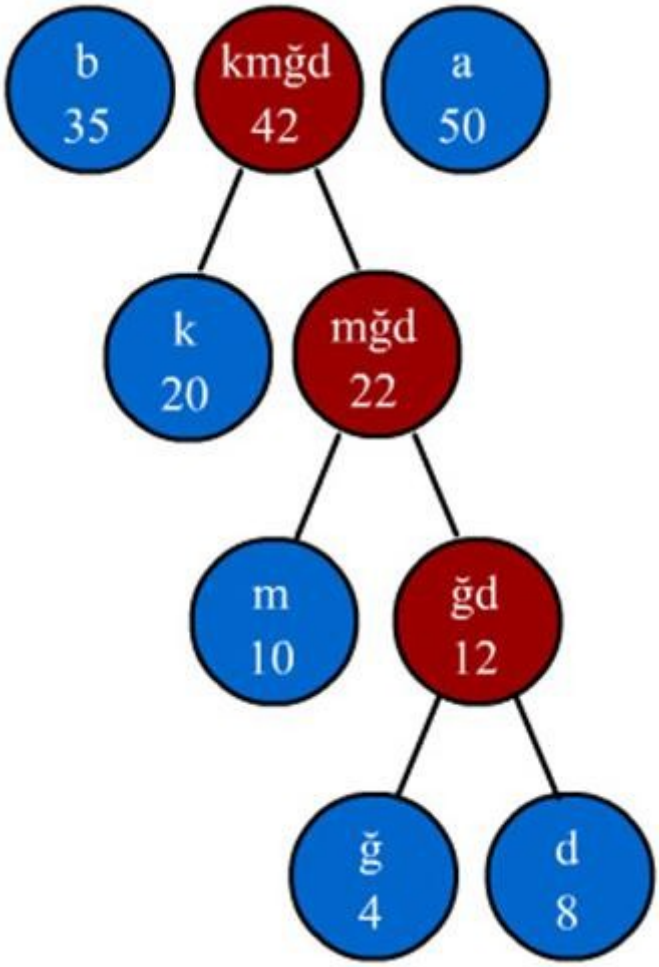
1 – Öncelikle "Huffman Ağacını" ndaki en son düğümleri oluşturacak bütün semboller frekanslarına göre aşağıdaki gibi küçükten büyüğe doğru sıralanırlar.
2 – En küçük frekansa sahip olan iki sembolün frekansları toplanarak yeni bir düğüm oluşturulur. Ve oluşturulan bu yeni düğüm diğer varolan düğümler arasında uygun yere yerleştirilir. Bu yerleştirme frekans bakımından küçüklük ve büyüklüğe göredir. Örneğin yukarıdaki şekilde "ğ" ve "d" sembolleri toplanarak "12″ frakansında yeni bir "ğd" düğümü elde edilir. "12″ frekanslı bir sembol şekilde "m" ve "k" sembolleri arasında yerleştirilir. "ğ" ve "d" düğümleri ise yeni oluşturulan düğümün dalları şeklinde kalır. Yeni dizimiz aşağıdaki şekilde olacaktır.



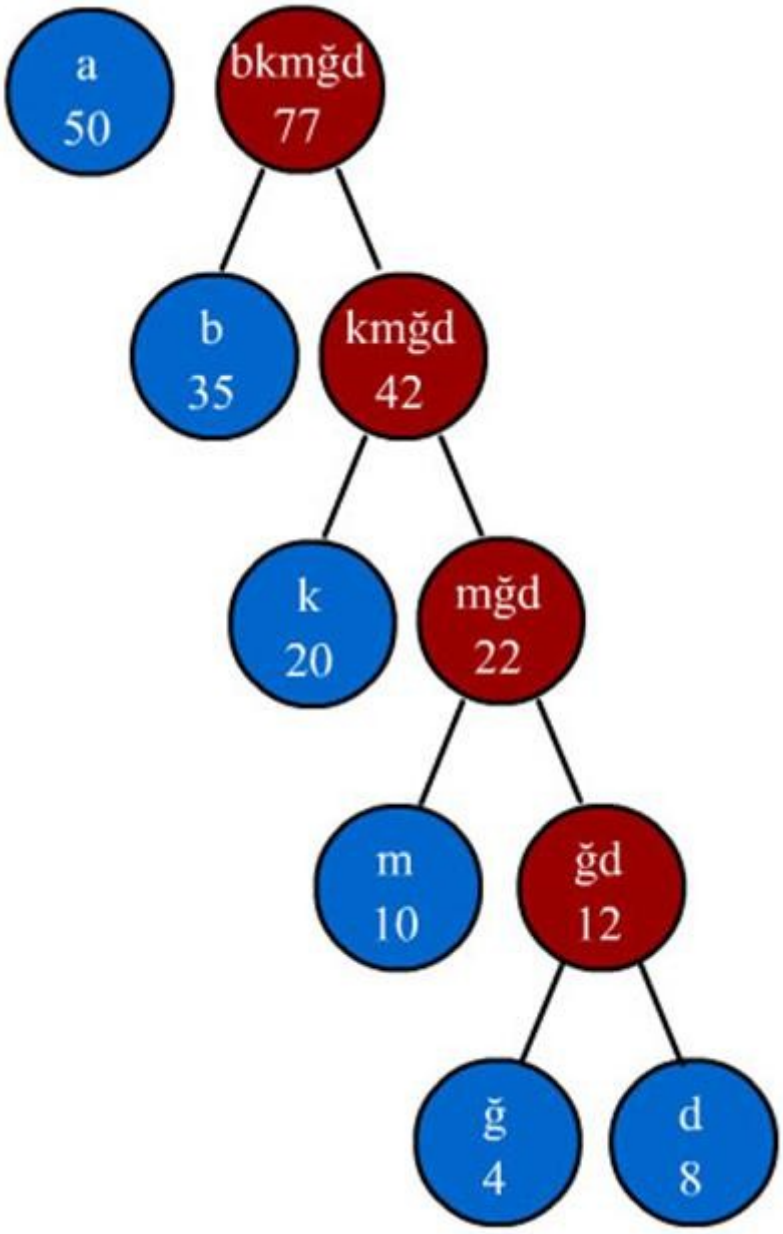
3 – 2.adımdaki işlem tekrarlanarak en küçük frekanslı iki düğüm tekrar toplanır ve yeni bir düğüm oluşturulur. Bu yeni düğümün frekansı 22 olacağı için "k" ve "b" düğümleri arasına yerleşecektir. Yeni dizimiz aşağıdaki şekilde olacaktır.



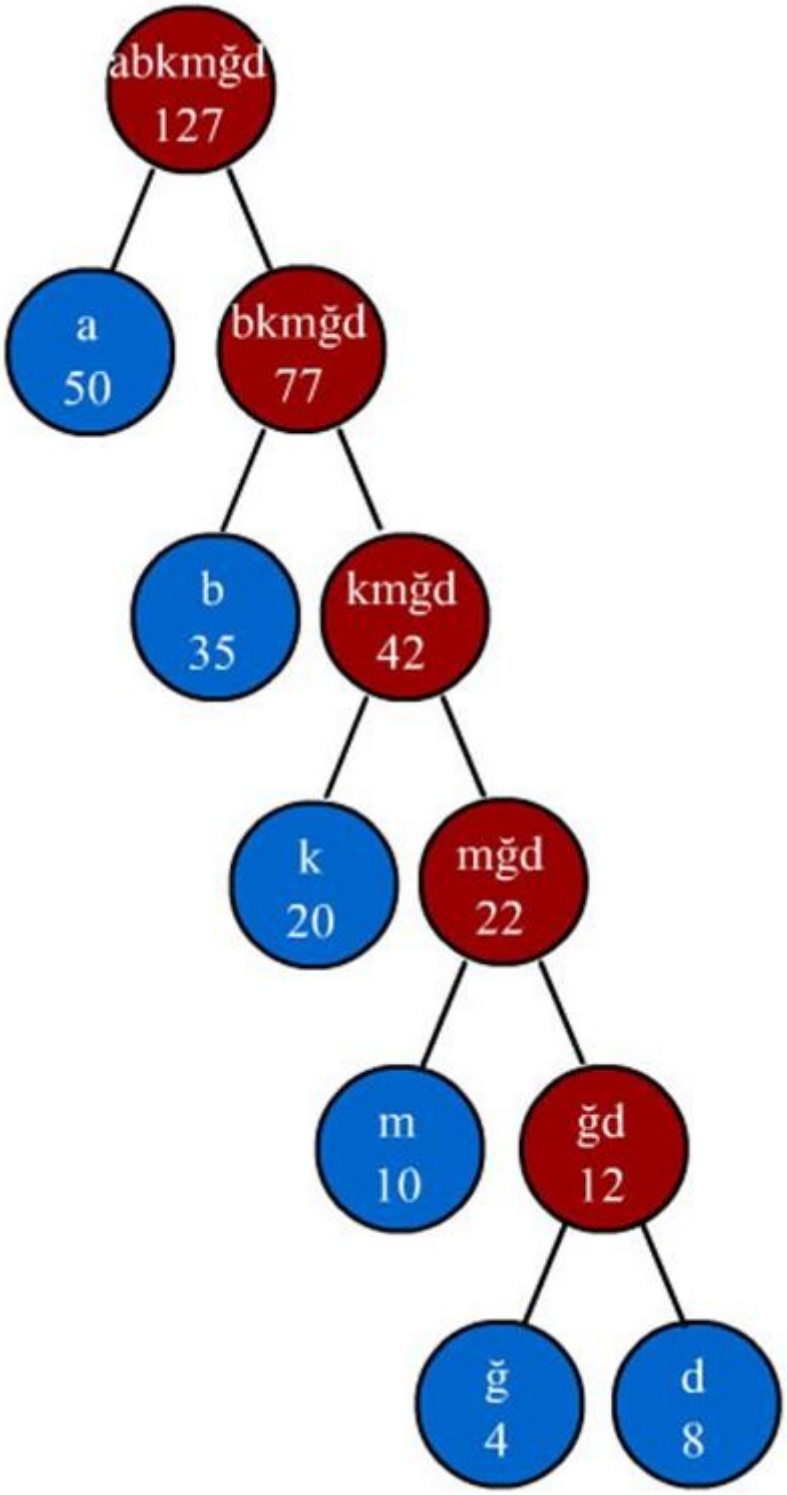
4 – 2.adımdaki işlem tekrarlanarak en küçük frekanslı iki düğüm tekrar toplanır ve yeni bir düğüm oluşturulur. Bu yeni düğümün frekansı 42 olacağı için "b" ve "a" düğümleri arasına yerleşecektir. Yeni dizimiz aşağıdaki şekilde olacaktır. Dikkat ederseniz her dalın en ucunda sembollerimiz bulunmaktadır. Dalların ucundaki düğümlere özel olarak **yaprak** denilmektedir. Sona yaklaştıkça Bilgisayar bilimlerinde önemli bir veri yapısı olan Tree veri yapısına yaklaştığımızı görüyoruz.



5 – 2.adımdaki işlem tekrarlanarak en küçük frekanslı iki düğüm tekrar toplanır ve yeni bir düğüm oluşturulur. Bu yeni düğümün frekansı 77 olacağı için "a" düğümünden sonra yerleşecektir. Yeni dizimiz aşağıdaki şekilde olacaktır. Dikkat ederseniz her bir düğümün frekansı o düğümün sağ ve sol düğümlerinin frekanslarının toplamına eşit olmaktadır. Aynı durum düğüm sembolleri içinde geçerlidir.



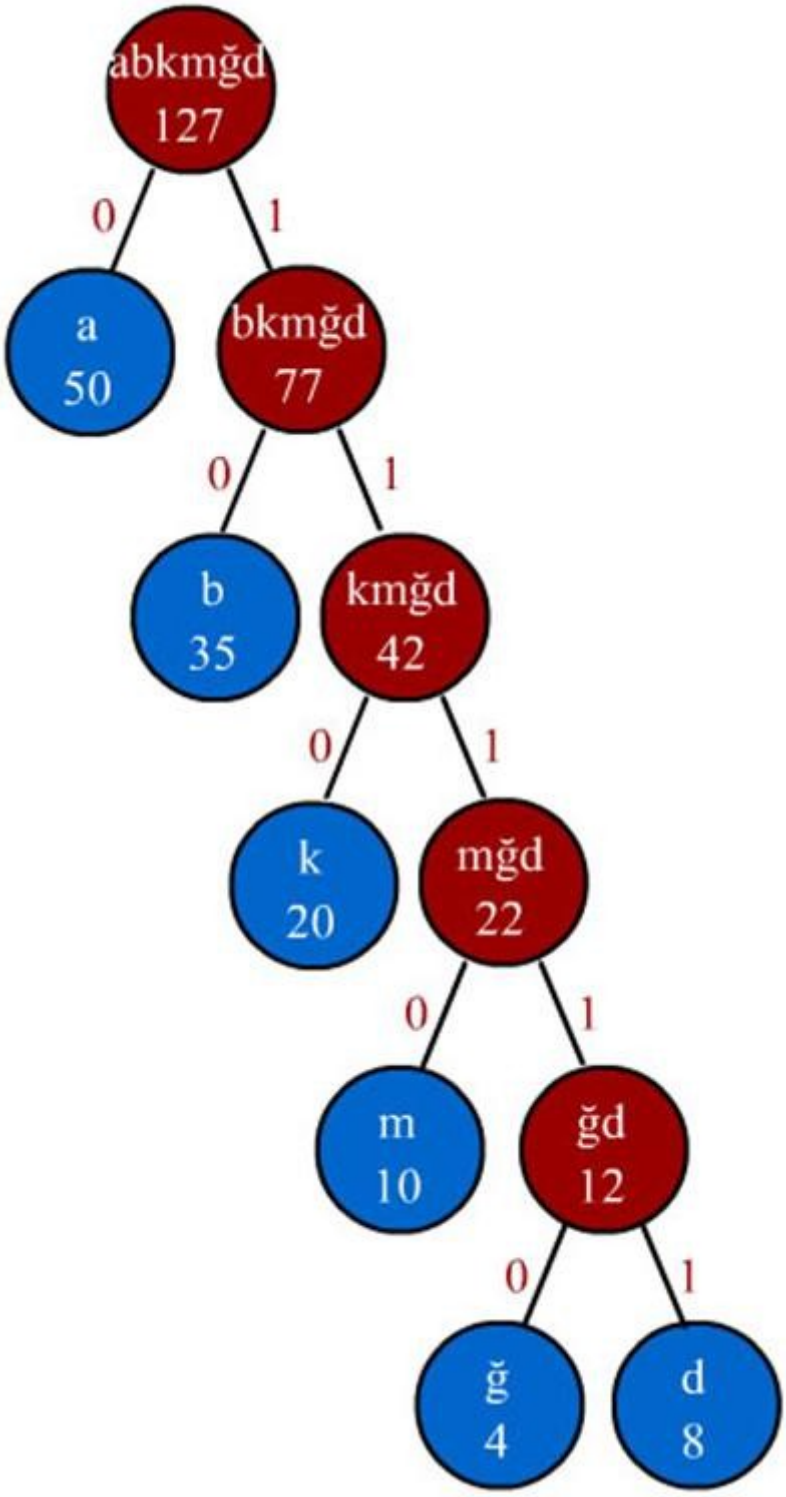
6 – 2.adımdaki işlem en tepede tek bir düğüm kalana kadar tekrar edilir. En son kalan düğüm Huffman ağacının kök düğümü(root node) olarak adlandırılır. Son düğümün frekansı 127 olacaktır. Böylece huffman ağacının son hali aşağıdaki gibi olacaktır.



Not : Dikkat ederseniz Huffman ağacının son hali simetrik bir yapıda çıktı. Yani yaprak düğümler hep sol tarafta çıktı. Bu tamamen seçtiğimiz sembol frekanslarına bağlı olarak rastlantısal bir sonuçtur. Bu rastlantının oluşması oluşturduğumuz her bir yeni düğümün yeni dizide ikinci sıraya yerleşmesinden kaynaklanmaktadır. Sembol frekanslarında değişiklik yaparak ağacın şeklindeki değişiklikleri kendinizde görebilirsiniz.

7 – Huffman ağacının son halini oluşturduğumuza göre her bir sembolün yeni kodunu

oluşturmaya geçebiliriz. Sembol kodlarını oluşturuken Huffman ağacının en tepesindeki kök düğümden başlanır. Kök düğümün sağ ve sol düğümlerine giden dala sırasıyla "0" ve "1" kodları verilir. Sırası ters yönde de olabilir. Bu tamamen seçime bağlıdır. Ancak ilk seçtiğiniz sırayı bir sonraki seçimlerde korumanız gerekmektedir. Bu durumda "a" düğümüne gelen dal "0", "bkmğd" düğümüne gelen dal "1" olarak seçilir. Bu işlem ağaçtaki tüm dallar için yapılır. Dalların kodlarla işaretlenmiş hali aşağıdaki gibi olacaktır.



8 – Sıra geldi her bir sembolün hangi bit dizisiyle kodlanacağını bulmaya. Her bir sembol dalların ucunda bulunduğu için ilgili yaprağa gelene kadar dallardaki kodlar birleştirilip

sembollerin kodları oluşturulur. Örneğin "a" karakterine gelene kadar yalnızca "0" dizisi ile karşılaşırız. "b" karakterine gelene kadar önce "1" dizisine sonra "0" dizisi ile karşılaşırız. Dolayısıyla "b" karakterinin yeni kodu "10" olacaktır. Bu şekilde bütün karakterlerin sembol kodları çıkarılır. Karakterlerin sembol kodları aşağıda bir tablo halinde gösterilmiştir.

Frekans	Sembol(Karakter)	Bit Sayısı	Huffman Kodu
50	a	1	0
35	b	2	10
20	k	3	110
10	m	4	1110
8	d	5	11111
4	ğ	5	11110

Sıkıştırılmış veride artık ASCII kodları yerine Huffman kodları kullanılacaktır. Dikkat ederseniz hiçbir Huffman kodu bir diğer Huffman kodunun ön eki durumunda değildir. Örneğin ön eki "110" olan hiç bir Huffman kodu mevcut değildir. Aynı şekilde ön eki "0" olan hiç bir Huffman kodu yoktur. Bu Huffman kodları ile kodlanmış herhangi bir veri dizisinin **"tek çözülebilir bir kod"** olduğunu göstermektedir. Yani sıkıştırılmış veriden orjinal verinin dışında başka bir veri elde etme ihtimali sıfırdır. (Tabi kodlamanın doğru yapıldığını varsayıyoruz.)

Şimdi örneğimizdeki gibi bir frekans tablosuna sahip olan metnin Huffman algoritması ile ne oranda sıkışacağını bulalım :

Sıkıştırma öncesi gereken bit sayısını bulacak olursak : Her bir karakter eşit uzunlukta yani 8 bit ile temsil edildiğinden toplam karakter sayısı olan (50+35+20+10+8+4) = 127 ile 8 sayısını çarpmamız lazım. Orjinal veriyi sıkıştırmadan saklarsak **127*8 = 1016** bit gerekmektedir.

Huffman algoritmasını kullanarak sıkıştırma yaparsak kaç bitlik bilgiye ihtiyaç duyacağımızı hesaplayalım : 50 adet "a" karakteri için 50 bit, 35 adet "b" karakteri için 70 bit, 20 adet "k" karakteri için 60 bit….4 adet "ğ" karakteri için 20 bite ihtiyaç duyarız. (yukarıdaki tabloya bakınız.) Sonuç olarak gereken toplam bit sayısı = 50*1 + 35*2 + 20*3 + 10*4 + 8*5 + 4*5 = 50 + 70 + 60 + 40 + 40 + 20 = 280 bit olacaktır.

Sonuç : 1016 bitlik ihtiyacımızı 280 bite indirdik. Yani yaklaşık olarak %72 gibi bir sıkıştırma gerçekleştirmiş olduk. Gerçek bir sistemde sembol frekanslarınıda saklamak gerektiği için sıkıştırma oranı %72'ten biraz daha az olacaktır. Bu fark genelde sıkıştırılan veriye göre çok çok küçük olduğu için ihmal edilebilir.

SORU-2: Elias Kodlaması (Elias Code) hakkında bilgi veriniz.

Veri sıkıştırmada veya verinin ikilik tabanda gösterilmesinde kullanılan bir algoritmadır. Basit bir çevirim fonksiyonu olarak da düşünülebilir. Bu yazı kapsamında birkça farklı elias kodu (elias code) şekli anlatılacaktır.

Elias-Y Kod (Elias – Y Code): Elias upsilon kodlaması olarak okunur.

İki formül bu kodlama için gereklidir.

$k_d = taban(\log_2 k)$

$k_r = k_d - 2^{\log_2 k}$

Yukarıdaki formülde k değeri, çevirimini istediğimiz değerdir. Elias-Upsilon kodlaması ise bu değerlerden ilkinin tekil kodlaması (unary coding) ile ikincisinin ikli kodlaması (binary coding) birleşimidir (yan yana yazılması, üleştirilmesi)

Örnek olarak bazı sayıların çevirimleri aşağıda verilmiştir:

k	kd	kr	Elias-Upslion
1	0	0	0
2	1	0	10 0
15	3	7	1110 111
255	7	127	11111110 1111111
1023	9	511	1111111110 111111111

Yukarıdaki tabloda görüldüğü üzere sayılar büyüdükçe kodlama uzunluğu da değişmektedir.

Elias-δ Kodu (Elias-δ Code) : Elias-Delta kodlama olarak okunabilir.

Yukarıda anlatılan elias-upsilon kodlamasını daha az bit ile gösterebilmek için geliştirilmiştir. Basitçe elias-upsilon kodlaması yaklaşık $2 \log_2(k)$ kadar bit gerektirmektedir. Bu bit sayısını azaltmak için elias-delta kodlaması kd ve kr hesaplamalarına ilave olarak iki değer daha hesaplar.

$k_{dd} = taban(\log_2(k_d + 1))$

$k_{dr} = k_d – 2^{\log_2(k_d + 1))}$

Görüldüğü üzere aslında elias-delta kodlamasında yapılan işlem, elias-upsilon kodlamasında bulunan kd değerini yeniden aynı formüllere koymak ve kd üzerinden kdd ve kdr değerlerini hesaplamaktır. Ardından bulunan kdd kdr ve kr değerleri birleştirilerek elias-delta kodu sonucu bulunur.

Örnek bazı saylar aşağıda verilmiştir:

k	kd	kr	kdd	kdr	Elias-Delta
1	0	0	0	0	0
2	1	0	1	0	10 0 0
15	3	7	2	0	110 00 111
255	7	127	3	0	1110 000 1111111
1023	9	511	3	2	1110 010 111111111

Yukarıda görüldüğü üzere genel olarak sayılar büyüdükçe elias-upsilon kodlamasına göre daha az bit gerekmektedir. düşük sayılarda daha fazla bit gerektiği görülmekte ve biraz da olsa işlem karmaşıklığı artmaktadır.

SORU-3: Tekil Kodlama (Unary Coding) hakkında bilgi veriniz.

Verilerin tekil karşılıkla kodlanmasıdır. Buna göre her verinin kendisine ait bir basamakta karşılığı bulunur.

Basitçe k değerindeki bir sayının kodlanması için k adet 1 ve sonuna bir adet 0 konulur.

Örneğin aşağıda bazı sayıların tekil kodlama (unary coding) karşılıkları verilmiştir:

1 –> 10

2 –> 110

3 –> 1110

11 –> 111111111110

Buna göre, 1023 sayısının karşılığı olarak 1023 adet 1 ve ardından 0 gelmesi gerekir.

Tekil kodlama düşük seviyeli sinyal işleme için bazı durumlarda kullanışlıdır ve her veri ünitesinin (her sayının) bitişini belirten bir sinyal olarak 0 kullanılmıştır. Ancak tekil kodlamanın en büyük mahsuru veriyi çok uzun kodlaması ve bunun sonucunda yer israfıdır.

Örneğin ikili kodlama (binary coding) için veriler ikilik tabana çevrilir. Bu durumda yukarıdaki örnekte verilen sayıların karşılıkları aşağıdaki şekilde olacaktır:

1 –> 1

2 –> 10

3 –> 11

11 –> 1011

Görüldüğü üzere çok daha az yer kaplayan veriler elde edilebilmektedir ancak bu değerlerin sürekli olarak 1 ve 0 olarak değişmesi söz konusudur. Diğer yandan veriler arasında bir ayrım söz konusu değildir. Örneğin tekil kodlama ile yollanan aşağıdaki dizgiyi (string) ele alalım:

101101110

Bu dizginin karşılığının 123 olduğu kolayca bulunabilirken aynı verinin ikili kodlamadaki karşılığı

11011

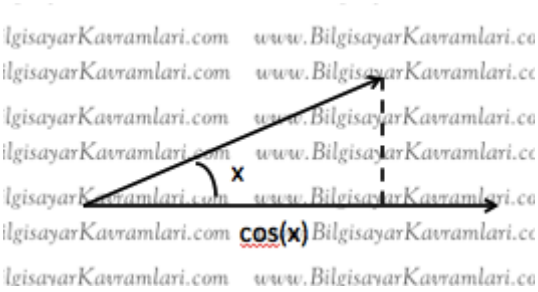
olmakta ve bu verinin 123 mü yoksa 11 mi olduğu bilinememektedir.

SORU-4: SimHash (Benzerlik Özeti) hakkında bilgi veriniz.

Bilgisayar bilimlerinde, özellikle metin işlemenin yoğun olduğu, arama motoru gibi uygulamalarda dosyaların veya web sitelerinin birbirine olan benzerliğini bulmak için kullanılan bir algoritmadır.

Algoritmaya alternatif olarak klasik hash fonksiyonları kullanılabilir. Yani, örneğin iki sayfasnın ayrı ayrı hash değerleri alınıp bu değerleri karşılaştırmak mümkündür. Ancak simhash algoritması, bu yönteme göre daha fazla hız ve performans sunar.

Sim hash algoritması, iki dosyayı birer vektör olarak görür ve bu vektorler (yöney, vector) arasındaki cosinüs (cosine) bağlantısını bulmaya çalışır.



Yukarıdaki şekilde temsil edildiği üzere iki dokümanın ayrı ayrı birere vektör olması durumunda, aralarında cos (x) olarak gösterilen bir açı ile bağlantı bulunması mümkündür.

Algoritma, öncelikle işlediği metindeki kelimelerin ağırlıklarını (weight) çıkarmakta ve buna göre kelimeleri sıralamaktadır.

Sıralanan her kelimeye, b uzunluğunda, yegane (unique) değer döndüren bir fonksiyon kullanılır. Örneğin her kelime için farklı bir hash sonucu döndüren fonksiyon kullanılır.

b boyutundaki bir vektörün ağırlık değeri hesaplanırken, her kelimedeki 1 değeri için +1 ve 0 değeri için -1 değeri ağırlığa eklenir.

Son olarak üretilen ağırlık vektöründeki + değerler 1, 0 ve – değerler ise 0 olarak çevirilir.

Örnek

Yukarıdaki algoritmanın çalışmasını bir örnek üzerinden anlatalım. Algoritmanın üzerinde çalışacağı metin aşağıdaki şekilde verilmiş olsun:

www bilgisayar kavramları com bilgisayar kavramlarının anlatıldığı bir bilgisayar sitesidir ve com uzantılıdır

Yukarıdaki bu metni, algoritmanın anlatılan adımlarına göre işleyelim:

İlk adımımız, algoritmadaki kelimelerini ağırlıklarının çıkarılmasıdır. Bu adımı çeşitli şekillerde yapmak mümkündür ancak biz örneğimizde kolay olması açısından kelime frekanslarını (tekrar sayısı, frequency) kullanacağız. Buna göre metindeki kelimelerin tekrar sayılarına göre sıralanmış hali aşağıda verilmiştir:

bilgisayar 3 com 2 kavramları 1 kavramlarının 1 anlatıldığı 1 bir 1 www 1 sitesidir 1 ve 1 uzantılıdır 1

Yukarıda geçen her kelime için bir parmak izi (fingerprint) değeri üretiyoruz. Bu değerin özelliği, kelimeler arasında yegane (unique) bir değer bulmaktır. Bu değer, herhangi bir hash fonksiyonu üzerinden de üretilebilir. Biz örneğimizde kolalık olması açısından her kelime için rast gele bir değer kendimiz atayacağız. Ancak gerçek bir uygulamada rast gele değerlerin kullanılması mümkün değildir. Bunun sebebi, aynı kelimenin tekrar gelmesi halinde yine aynı değerin üretilmesi zorunluluğudur. Bu yazıdaki amaç algoritmayı anlatmak olduğu için birer hash sonucu olarak rast gele değerler kullanılacaktır.

bilgisayar 10101010 com 11000000 kavramları 01010101 kavramlarının 10100101 anlatıldığı 11101110 bir 01011111 www 11110001 sitesidir 10101110 ve 00001111 uzantılıdır 00100010

3. adımda, yukarıdaki değerleri topluyoruz. Toplama işlemi sırasında 1 değerleri için +1 ve 0 değerleri için -1 alıyoruz.

10101010
11000000
01010101
10100101
11101110
01011111
11110001
10101110
00001111
00100010
———
2 0 2 -4 0 2 2 0

Son olarak, yukarıdaki değerleri ikilik tabana çeviriyoruz: 10100110 bu değer bizi simhash sonucumuz olarak bulunuyor.

Örneğin yeni bir dosyayı daha işlemek istediğimizde, bu dosyadaki kelime yoğunluğuna göre yukarıda bulduğumuz simhash değerine yakın bir değer çıkmasını bekleriz.

Diyelim ki yeni bir dosyada da sadece “bilgisayar kavramları com” yazıyor olsun. Bu yazının sim hash değerini bularak karşılaştırmaya çalışalım:

bilgisayar 10101010 com 11000000 kavramları 01010101

10101010
11000000
01010101
———
1 1 -1 -1 -1 1 1 1

Değerin ikilik tabana çevrilmiş hali : 11000111

Orjinal dokümandan çıkardığımız simhash değeri ile farklı olan bit sayısı 3′tür. Bunun anlamı yukarıdaki bilgisayar kavramları com yazısının orjinal yazıya 3 mesafesinde yakın olduğudur.

SORU-5: Eşlik Kontrol Matrisi (Parity Check Matrix) hakkında bilgi veriniz.

Hata kontrolü için kullanılan yöntemlerden birisidir. Veri güvenliği, veri iletimi veya veri sıkıştırma gibi alanlarda kullanılır. Genelde H sembolü ile gösterilir. Basitçe sistemde kullanılan üreteç matristen (generating matrix) çıkarılabilir. Bir eşlik kontrol matrisinin yapısı aşağıda verilmiştir:

G = [I|P] şeklinde bir üreteç matris olmak üzere

$H = [P^T|I]$ şeklinde bir eşlik kontrol matrisi üretilebilir.

Örneğin aşağıdaki şekilde bir üreteç matrisimiz olsun:

Örneğin aşağıdaki üreteç matrisi ele alalım:

| 1 0 0 1 1 |
| 0 1 0 1 0 | = G
| 0 0 1 1 1 |

Bu matrisin ilk kısmı olan 3×3 boyutlarındaki bölüm birim matristir

| 1 0 0 |
| 0 1 0 | = I
| 0 0 1 |

İkinci kısım ise P ile gösterilen üreteç matrisin özel parçasıdır:

| 1 1 |

| 1 0 | = P

| 1 1 |

Yukarıdaki parçaları birleştirmeden önce P matrisinin tersyüzünü (transpose) alıyoruz:

|111|

$|101| = P^T$

Ardından birim matris ile birleştiriyoruz:

|11110|

|10101| = H

Şeklinde yukarıda verilen üreteç matrisin, eşlik kontrol matrisi bulunur.

SORU-6: Kod Kelimesi hakkında bilgi veriniz.

Haberleşmede kullanılan bir terimdir. Bir kod kelimesi (code word), belirli bir teşrifatın (protocol, protokol) anlamlı en küçük parçasıdır. Her kod kendi başına tek bir anlam ifade eder ve bu anlam yeganedir (unique).

Aynı yaklaşım programlama dilleri için de geçerlidir. Her programlama dilinde bulunan her kelime tek bir anlam ifade eder.

Örneğin bir programlama dilindeki "if" kelimesi, bu dildeki kaynak kodda bulunan (source code) kod kelimesidir (code word).

Benzer durum herhangi bir haberleşme teşrifatında da olabilir.

Kod kelimeleri kullanıldıkları yere göre kanal kelimeleri (channel code words) veya kaynak kelimeleri (source code words) olarak isimlendirilebilir. İlki haberleşme ikincisi ise programlama tabiridir.

Ancak kavramsal olarak kaynak kelimelerinin veri sıkıştırma (data compression) veya veri güvenliği (cryptography) gibi alanlarda kullanılması da mümkündür. Örneğin uzun bir kelimeyi, daha kısa bir kelime ile ifade etmenin anlamı, bu kelimenin yerine geçen bir kaynak kod kullanılmasıdır.

Benzer şekilde kanal kodları, gürültülü ortamlarda veri iletişimini güvenli hale getirmek için gereksiz ilave bilgiler içerebilir. Yani kod kelimeleri, yerine kullanıldıkları anlamdan uzun veya kısa olabilmektedirler.

Veri güvenliği açısından da bir kod kelimesi, yerine kullanıldığı kelimeye dönüşü sadece belirli kişiler tarafından yapılabilen şifreli metindir.

SORU-7: Çift Özetleme (Double Hashing) hakkında bilgi veriniz.

Bilgisayar bilimlerinde kullanılan özetleme fonksiyonları, genellikle büyük bir verinin daha küçük bir hale getirilmesine yarar. Bu anlamda özetleme fonksiyonları veri doğrulama (data verification) , veri bütünlüğü (data integrity), veri güvenliği (security) ve şifreleme (encryption) gibi pek çok alanda kullanılırlar.

Özetleme fonksiyonlarının bir problemi, büyük bir veriyi özetledikten sonra, çakışma olması durumudur. Çalışma (collision) kısaca aynı özet değerine sahip iki farklı verinin olmasıdır.

Örneğin en basit özetleme fonksiyonlarından birisi olan kalan (mod) işlemini ele alalım. 0 ile 100 arasındaki sayıları, 0 ile 10 arasındaki sayılarla özetlemek istersek, mod 10 kullanmamız mümkündür. Bu durumda her sayının 0 – 10 aralığında bir karşılığı bulunacaktır.

Ancak 41 mod 10 ile 51 mod 10 aynı sonucu verir. Bu durumda bir çakışma olmuş denilebilir.

Çakışmayı engellemek için veri yapılar üzerinde sondalama yöntemleri kullanılabilir. En meşhurları olan doğrusal sondalama (linear probing) ve ikinci dereceden sondalama (quadratic probign) yöntemleri bu problemin çözümü için geliştirilmiş yöntemlerdir.

Bu yazının konusu olan çift özetleme (double hashing) yöntemi de işte tam bu noktada devreye girer. Yani bir şekilde özetleme fonksiyonundan çıkan sonuçların, çakışması (collision) durumunda, ikinci ve farklı bir özetleme fonksiyonu kullanılarak veri yapısı üzerinde farklı bir noktada arama veya veri ekleme işlemine devam edilebilir.

Örneğin veri yapısına ekleme işlemini ele alalım. Verinin hangi adrese ekleneceğini bulmak için öncelikle anahtar (key) bir özetleme fonksiyonuna sokulur. Bu ilk özetleme fonksiyonuna $Ö_1$ ismini verelim.

Adres = Ö1 (anahtar) formülü ile adresi buluruz. Diyelim ki bu adres dolu ve buraya yeni verimizi ekleyemiyoruz. Bu durumda ikinci bir adres aranmalıdır. İşte bu noktada ikinci özetleme fonksiyonu $Ö_2$ devreye girer ve verinin yerleştirilebileceği boş bir adres bulunana kadar bu fonksiyon kullanılmaya devam edilir.

Bu durumu 10 adet hücresi bulunan boş bir veri yapısı üzerinden sırasıyla 21,31,41,51 sayılarının eklenmesi şeklinde görelim. Ö1 fonksiyonu olarak

Adres = anahtar mod 10

Ve ikinci özetleme fonksiyonu olarak Ö2 :

Adres = ((anahatar mod 7) x 3) mod 10

Fonksiyonlarını alalım. Bu fonksiyonlar örnek olarak alınmıştır ve farklı fonksiyonlar üzerinden de çift özetleme (double hashing) yapılabilir.

Sırasıyla sayılarımızın üzerinden geçiyoruz. İlk sayımız 21 mo 10 = 1 olarak bulunur.

0	
1	21
2	
3	
4	
5	
6	
7	
8	
9	

İlk özetleme fonksiyonu sonucu olarak 1 numaralı adrese yerleştirilir. Ardından ikinci sayıya geçilir:

31 mod 10 = 1 bulunur ve bu adres dolu olduğu için çakışma olur. Çözüm olarak ikinci özetleme fonksiyonu kullanılır.

((31 mod 7) x 3) mod 10 = 9 olarak bulunur ve bu adrese yerleştirilir:

0	

1	21
2	
3	
4	
5	
6	
7	
8	
9	31

Ardından 41 sayısı için 1. özetleme fonksiyonu çalıştırılır ve 1 numaralı adres dolu olduğu için çakışma oluşur. Çözüm olarak ikinci özetleme fonksiyonu çalıştırılır:

((41 mod 7) x 3) mod 10 = 8

0	
1	21
2	
3	
4	
5	
6	
7	
8	41
9	31

Benzer şekilde 51 için çakışma olur ve ikinci özet fonksiyonu çalıştırılır.

((51 mod 7) x 3) mod 10 = 6

0	
1	21
2	
3	
4	
5	
6	51
7	
8	41
9	31

Görüldüğü üzere ikinci özetleme fonksiyonu, ilkinde bir çakışma olması halinde kullanılmaktadır. Peki acaba ikinci özetleme fonksiyonunda da çakışma olursa ne yapılır?

Bu durumu örneğimize devam edip 61 sayısını eklemek istediğimizde görebiliriz.

((61 mod 7) x 3) mod 10 = 8

Bulunacaktır ve 8 numaralı adres doludur. Bu durumda ikinci özetleme fonksiyonuna ikinci kere sokularak farklı bir adres aranır:

((8 mod 7) x 3) mod 10 = 3 olarak bulunur ve bu adrese yerleştirilir.

0	
1	21
2	
3	61
4	
5	
6	51
7	
8	41
9	31

Görüldüğü üzere ikinci özetleme fonksiyonunun ilk özetleme fonksiyonu ile aralarında asal olması, belirli bir adres dizilimine girilmesine engellemekte bu sayede veri yapısı üzerinde boş yer bulunuyorsa mutlaka belirli bir denemeden sonra bu adrese erişilmesi garanti edilmiş olmaktadır.

SORU-8: İkinci Dereceden Sondalama (Quadratic Probing) hakkınad bilgi veriniz.

Özellikle özetleme fonksiyonlarının (hashing functions) bilgileri sınıflandırması sırasında kullanılan formülün ikinci dereceden olması durumudur.

Özetleme fonksiyonlarında, sık kullanılan doğrusal sondalama (linear probing) yönteminin tersine, bir bilgiyi tasnif ederken, ardışık olarak veriler üzerinde hareket etmez, bunun yerine her defasında baktığı uzaklığı ikinci dereceden bir denklem ile arttırır.

Konuyu anlamaya öncelikle doğrusal fonksiyonları hatırlayarak başlayalım.

Bilindiği üzere doğrusal fonksiyonlar :

y = ax + b şeklinde yazılabilen birinci dereceden fonksiyonlardır. Bu durumda özetleme fonksiyonu veriyi sınıflandırırken bir çakışma (collision) olması durumunda bir sonraki veri hücresine bakar, ve bu şekilde aranan veriye ulaşana kadar devam eder. Örneğin 10 hücreli bir sınıflama için mod 10 fonksiyonunu kullanacağımızı düşünelim.

Bu durumda ilk başta aşağıdaki şekilde boş olan veri yapımıza sırasıyla 21,31,41,51 sayılarının geldiğini kabul edelim.

0	
1	
2	
3	

4	
5	
6	
7	
8	
9	

Dikkat edileceği üzere bu sayılar özellikle çakışma olsun diye mod 10 fonksiyonuna konulduğunda hep 1 olarak çıkan sayılardan seçilmiştir.

Şimdi bu sayıları yerleştirecek olursak ilk gelen sayı 21 mod 10 = 1 olduğu için 1. Hücreye yerleştirilecektir.

0	
1	21
2	
3	
4	
5	
6	
7	
8	
9	

Ardından gelen 31 sayısı yine 1. Hücreye yerleştirilmek istenecek ancak burası dolu olduğu için doğrusal sondalama kullanarak bir sonraki hücreye yerleştirme yapılıyor:

0	
1	21
2	31
3	
4	
5	
6	
7	
8	
9	

Diğer sayılar için durum değişmiyor ve aynı yaklaşım izlenmeye devam ediliyor:

0	
1	21
2	31
3	41
4	51

5	
6	
7	
8	
9	

Görüldüğü üzere elimizdeki sayıların tamamı başarılı bir şekilde yerleştirilmiştir. Konuyu daha iyi anlayabilmek için farklı bir örnek olarak 33 sayısını yerleştirmek istediğimizi düşünelim. Bu durumda 33 mod 10 = 3 hücresi dolu olduğu için işlem değişmeden yukarıda olduğu gibi uygulanacaktır:

0	
1	21
2	31
3	41
4	51
5	33
6	
7	
8	
9	

İkinci dereceden sondalama (quadratic probing)

Doğrusal sondalamayı anladıktan (hatırladıktan) sonra ikinci dereceden sondalamadan bahsedebiliriz. Buradaki yaklaşımda kullanılan formül bir özetleme fonksiyonunda geçildikten sonra çakışma olması durumunda (collision) sürekli olarak bir sonraki hücreye bakmak yerine, her defasında üssel fonksiyon değeri kadar ilerlemektir.

Örneğin yukarıdaki sayılar için aynı veri yapısına ikinci dereceden sondalama ile ekleme işlemi yapalım:

0	
1	21
2	
3	
4	
5	
6	
7	
8	
9	

İlk gelen sayı, bir çakışma olmadığı için doğal hücresine yerleştiriliyor. Ardından gelen 31 sayısı için çakışma oluyor. Bu durumda sonraki bakılacak olan hücrenin karesi alınıyor, şu anda ilk çakışma yaşandığı için $1^2 = 1$ yani doğal hücreden bir sonraki hücreye bakıyoruz:

0	
1	21
2	31
3	
4	
5	
6	
7	
8	
9	

Bu hücre boş olduğu için yerleştirme işlemi tamamlanıyor ve sıradaki sayıya geçiliyor.

Sayımız 41 ve doğal hücresi olan 1. Adreste çakışma yaşanıyor,

ardından $1^2 = 1$ sonraki hücrede de (yani 2. Hücre) çakışma yaşanıyor,

ardından gelen $2^2 = 4$ sonraki hücreye bakılıyor. Dolayısıyla doğrusal sondalamada olduğu gibi 3. Hücreye bakmak yerine 5. Hücreye bakıyoruz:

0	
1	21
2	31
3	
4	
5	41
6	
7	
8	
9	

Sırada 51 bulunuyor ve benzer şekilde doğal hücresi dolu, $1^2 = 1$ sonraki hücre dolu, $2^2 = 4$ sonraki hücre dolu ve nihayet $3^2 = 9$ sonraki hücre boş bulunup yerleştiriliyor.

0	51
1	21
2	31
3	
4	
5	41
6	
7	
8	
9	

Yukarıdaki yerleştirme işlemi sırasında mod10 kullanıldığı unutulmamalıdır, bu yüzden 0. Hücreye yerleştirme işlemi yapılmıştır.

Doğrusal sondalama örneğinde olduğu gibi, 33 sayısını da eklemek istediğimizde doğal hücresi boş olduğu için herhangi bir sondalamaya gerek kalmadan yerleştirme işlemi yapılabilir:

0	51
1	21
2	31
3	33
4	
5	41
6	
7	
8	
9	

Doğrusal sondalamada özetleme fonksiyonundan sonra sırasıyla veri yapısındaki hücrelere bakılır. Bu durum için aşağıdakine benzer bir döngü yazılması gerekir:

```
adres = anahtar % 10;
while(dolu(adres++)){
    adres %= 10;
}
ekle(anahtar,adres);
```

Görüldüğü üzere, öncelikle özetleme fonksiyonuna (h) anahtar verilip doğal adres bulunuyor. Bu adresin dolu olması ve sonraki adreslerin de dolu olması durumunda adres değeri arttırılarak devam ediliyor.

İkinci dereceden sondalama kullanılsaydı, bu kodu aşağıdaki şekilde yazmamız gerekecekti:

```
adres = anahtar % 10;
int artis = 0;
while(dolu(adres)){
    adres = (adres + artis * artis) % 10;
    artis ++;
}
ekle(anahtar,adres);
```

Yukarıdaki yeni haliyle kodumuz, her defasında artış miktarının karesi kadar sonraki hücreye bakmaktadır.

Yukarıdaki müsvedde kodlar (pseudo codes) ve örnekler fikir vermesi açısından yazılmış olup, özetleme fonksiyonu, doğrusal sondalama ve ikinci dereceden sondalama işlemleri için farklı fonksiyonlar kullanılabilir. Buradaki amaç, ikinci dereceden sondalamada kullanılan fonksiyonun ikinci derece bir denklem olmasıdır.

SORU-9: LZW Sıkıştırma algoritması hakkında bilgi veriniz.

Bilgisayar bilimlerinde kullanılan kayıpsız sıkıştırma (lossless compression) algoritmalarından birisidir. İsmini, algoritmayı 1978 yılında bulan Lempel Ziv ve Welch isimli kişilerin baş harflerinden almıştır.

Algoritma, sıkıştırılacak metin içerisinde harf harf ilerleyerek, mümkün olan en fazla harfi içeren kelimeyi sözlüğe eklemeye çalışmakta ve bu sırada da sözlükteki karşılığı ile metni değiştirmektedir. Böylelikle sıkıştırma işlemi gerçekleşmiş olur. Örneğin 4 harf uzunluğunda bir kelimeyi sözlüğe eklemeyi başardıysak, karşı tarafa gönderilen mesajda tek bir sembol bu dört harflik mesajı içerecektir.

Algoritmanın bir özelliği sözlüğün karşı tarafa iletilmesini gerektirmemesidir. Yani yollanan mesaj içerisinde harflerden oluşan bir sözlük bulunmakla birlikte çok harfli kelimeleri içeren sözlük dinamik olarak oluşturulur ve açan taraf da bu sözlüğü yine dinamik bir şekilde oluşturarak mesajı açar.

Algoritmanın sıkıştırması

Algoritma sıkıştırma işlemi sırasında basitçe, sıkıştırılacak olan metin üzerinde, sözlükte olan bir kelimeyle uyuşan harfler bulduğu sürece ilerler. Farklı bir harfe rastladığı zaman, o ana kadar uyumlu bulduğu harflerden oluşan kelimenin kodunu sonuca yazar ve yeni harfi içeren kelimeyi sözlüğe ekler.

Müsvedde kod (pseudo code) olarak aşağıdaki şekilde yazılabilir:

1. Metinden bir harf al
2. Sözlükte harfi ara
3. Metindeki sıradaki harfi aldığında sözlükteki bir kayda karşılık geliyorsa metinden harf almaya devam et
4. Şimdiye kadar uyan sözlük değerini sonuca bas
5. Uyumu bozan yeni harfle birlikte şimdiye kadar uyan sözlük kaydını, yeni sözlük kaydı olarak ekle
6. Metin bitmediyse 2. Adımdan uymayan bu yeni harf ile devam et.

Bu durumu aşağıdaki örnek üzerinden inceleyelim.

Sıkıştırmak istediğimiz mesajımız: “sadisadisadi” olsun.

Öncelikle yukarıda bulunan harflerimizi içeren bir sözlük oluşturuyoruz. Bu sözlük her iki tarafta da bulunmalıdır. Bu örnekte sadece 4 farklı harf olduğu için 4 harf içeren bir sözlüğümüzün olması yeterlidir. Ancak genel kullanımda, örneğin ascii tablosu böyle bir sözlük olarak bütün bilgisayarlarda kayıtlıdır ve ilave bir transfere ihtiyaç duyulmaz.

a-1,d-2,i-3,s-4 şeklinde 4 harflik alfabemizi ve karşılığı olan sayıları belirleyelim.

Mesajımızı işlemeye başlıyoruz. Aşağıda : işaretinin sol tarafı sıkıştırılacak olan metin ve sağ tarafı sıkıştırma işleminin sonucu olarak düşünülebilir. Mesajın ilk harfinden başlayalım:

Sadisadisadi

Buradaki s harfi sözlüğümüzde zaten bulunuyor ve çıktımıza sözlükten 4 sayısını koyuyoruz:

SAdisadisadi : 4

Yukarıda, sözlükten ilk harfi sonuca koyduk ve sıradaki harfi aldık. Sözlüğümüzde s harfi bulunmakta ve şimdiye kadar karşılaştığımız en uzun sözlük kaydı bu harf olmaktadır. Yeni gelen a harfi ile birlikte bir s harfi olmadığı için bu harfi sonuca harf olarak 1 şeklinde yazıyoruz, ilk kez karşılaştığımız durumu sözlüğe ekliyoruz ve sa için 5 kaydını giriyoruz. Sözlüğün yeni hali ve mesaj aşağıdaki şekildedir:

Sözlük : a-1,d-2,i-3,s-4,sa-5

sADisadisadi : 4 1

Şu anda a harfiyle başlayan bir kelime üzerinde işlem yapıyorduk. Bu kelime sadece a harfini içeren sözlük girdisiydi ve yeni bir harf olarak d harfi geldi. Bu durumda ad ile başlayan bir sözlük girdisi olmadığı için sözlüğe bu yeni kaydı ekleyip, sonuç kısmına d harfini olduğu gibi alıyoruz:

Sözlük : a-1,d-2,i-3,s-4,sa-5,ad-6

saDİsadisadi : 4 1 2

Yukarıdaki durumlara benzer şekilde, İ harfi ilk defa karşılaştığımız bir durum. Sözlükteki d harfini işlerken karşılaştığımız için Dİ kaydını ekliyor ve i harfini sonuca alıyoruz:

Sözlük : a-1,d-2,i-3,s-4,sa-5,ad-6,di-7

sadİSadisadi : 4 1 2 3

Aynı durum is için geçerlidir, i harfini sonuca alıp, sözlüğe ilk karşılaştığımız is durumunu ekliyoruz:

Sözlük : a-1,d-2,i-3,s-4,sa-5,ad-6,di-7,is-8

sadiSAdisadi : 4 1 2 3 5

İşte yukarıdaki sa durumunda ilk defa sözlükte iki harfi birden bulunan bir kayıt yakalıyoruz. Sa harfleri 5. Kayıtta bulunuyor. Bu durumda sözlükte olmayan bir harf bulana kadar işlemeye devam ediyoruz ve d harfine gelince sözlükteki kaydın dışına çıkılmış olunuyor:

Sözlük : a-1,d-2,i-3,s-4,sa-5,ad-6,di-7,is-8,sad-9, dis-10

sadisaDİsadi : 4 1 2 3 5 7

Benzer şekilde di kaydının, sözlükte olmayan bir hale gelmesi, sıkıştırılacak metindeki s harfinin eklenmesi ile sağlanıyor. Sözlüğe ekleyip mesajın kalanına devam ediyoruz.

Sözlük : a-1,d-2,i-3,s-4,sa-5,ad-6,di-7,is-8,sad-9, dis-10,sadi-11

sadisadiSADi : 4 1 2 3 5 7 9

Yukarıda, ilk defa sözlükte 3 harfli bir kayıt yakaladık. Bu kaydın metinle uyuşmayan ilk durumu sondaki i harfi gelmesidir. Yukarıdaki durumlara benzer şekilde uyan kısmını sözlükten alıp sonuca yazıyor, uymayan halini de uymayan kelimeyi ekleyerek sözlüğe yazıyoruz.

Son olarak sonda tek harf olan i kalıyor ve bunu da sözlük karşılığı olan 3 yazarak sonuca alıyoruz.

Sözlük : a-1,d-2,i-3,s-4,sa-5,ad-6,di-7,is-8,sad-9, dis-10,sadi-11

sadisadiSADi : 4 1 2 3 5 7 9 3

Görüldüğü üzere metin 12 harfliyken sonuç mesajında 8 sayı bulunmaktadır. Dolayısıyla bu örnekte %33 başarı sağladığımızı söyleyebiliriz.

Algoritmanın açması

Yukarıda sıkıştırılması anlatılan mesajın, nasıl açıldığını anlatmaya çalışalım.

Algoritma açma işlemi sırasında, sıkıştırılmış mesajdan bir değer okur ve bunun karşılığını sözlükten bularak açılmış mesaj olarak çıkartır. Ayrıca sözlüye her yeni gelen harfi, sözlükten bulduğu bir önceki kayda ekleyerek sözlüğe yeni bir kayıt olarak ekler.

Bu durumu müsvedde kod olarak yazmaya çalışırsak:

1. Şifreli mesajdan bir sembol oku
2. Sözlükte karşılığını ara
3. Sözlükte karşılığını bulduğun sürece sıkıştırılmış metni okumaya devam et.
4. Farklı harfe kadar bulduğun sözlük girdisini açılmış mesaja ekle, farklı harfi sözlük girdisine ilave ederek sözlüğe kaydet
5. Metin bitmediyse 2. Adımdan yeni harf ile devam et

Yukarıdaki açma algoritmasının daha önceden sıkıştırdığımız mesajı nasıl açtığını inceleyelim:

Açmak istediğimiz mesaj “4 1 2 3 5 7 9 3″ olarak verilmiş olsun ve ilk sözlüğümüz a-1,d-2,i-3,s-4 şeklinde 4 harf içersin.

Mesajı işlemeye ilk sembol olan 4 ile başlıyor ve sonucu hemen sözlükten bulup yazıyoruz.

Sözlük: a-1,d-2,i-3,s-4,sa-5

4 1 2 3 5 7 9 3 : s

Şifreli mesajda bir sonraki sembole geçildiğinde bu iki sembolün birleşimini yukarıda görüldüğü üzere sözlüğe ekliyoruz (4-s ve 1-a harfleri olduğu için sözlüğe yeni bir sa kaydı ekleniyor) ve sembol karşılığı sonuca yazılıyor. Aynı durum sonraki adımlarda da devam ediyor:

Sözlük: a-1,d-2,i-3,s-4,sa-5,ad-6

4 **1 2** 3 5 7 9 3 : s a

Sözlük: a-1,d-2,i-3,s-4,sa-5,ad-6,di-7

4 1 **2 3** 5 7 9 3 : s a d

Sözlük: a-1,d-2,i-3,s-4,sa-5,ad-6,di-7,is-8

4 1 2 **3 5** 7 9 3 : s a d i

Buraya kadar olan örneklerde hep tek harfli kayıt sözlükten bulunmuş ve yazılmıştır. İlk defa çok harfli bir kayıt olan 5. Kayda rastlanmıştır. Algoritma aynı yaklaşımla açmaya devam edecek ve sözlük kaydını sonuca yazdıktan sonra yeni kaydı sözlüğe ekleyecektir:

Sözlük: a-1,d-2,i-3,s-4,sa-5,ad-6,di-7,is-8,sad-9

4 1 2 3 **5 7** 9 3 : s a d i sa

Yukarıdaki örnekte dikkat edilecek bir husus, 5 girdisinden sonra 7 gelince bu iki girdiyi birleştirerek sadi şeklide sözlüğe bir kayıt girmek yerine, 5 girdisinin sonuna 7 girdisindeki ilk harfi eklemektir. Buna dikkat edilmesi gerekir çünkü lzw algoritması her adımda tek bir harf ilave etmektedir.

Sözlük: a-1,d-2,i-3,s-4,sa-5,ad-6,di-7,is-8,sad-9,dis-10

4 1 2 3 5 **7 9** 3 : s a d i sa di

Sözlük: a-1,d-2,i-3,s-4,sa-5,ad-6,di-7,is-8,sad-9,dis-10,sadi-11

4 1 2 3 5 7 **9 3** : s a d i sa di sad

Sözlük: a-1,d-2,i-3,s-4,sa-5,ad-6,di-7,is-8,sad-9,dis-10,sadi-11

4 1 2 3 5 7 9 3 : s a d i sa di sad i

SORU-10: DFA Metin Arama Algoritması (DFA Text Search) hakkında bilgi veriniz.

1. Otomatın İnşası
2. Algoritmanın arama aşaması
3. Algoritmanın çalışması
4. Algoritmanın kodlanması

Bilgisayar bilimlerinde, bir metnin içerisinde farklı bir metnin veya bir kelimenin aranması sırasında kullanılan algoritmalardan birisidir. Algoritma, aranan kelime için bir otomat (automaton) oluşturur ve hedef metin içerisinde bu otomata göre arama işlemi yapar.

Oluşturulana otomatın DFA (deterministic finite automaton, belirli sonlu otomat) olması gerekmektedir.

Algoritmanın çalışması iki aşamada incelenebilir:

- Aranan kelimeye özgü bir otomatın inşa edilmesi
- İnşa edilen bu otomatın bir metin içerisinde arama işlemi için kullanılması

Yukarıdaki bu adımları sırasıyla anlatmaya çalışalım ve ayrıca performanslarını inceleyelim.

Otomatın inşası

Bilindiği üzere bir DFA ifade edilirken şu dört terimden faydalanılır: $\boldsymbol{A}(x) = (\boldsymbol{Q}, q_0, \boldsymbol{T}, \boldsymbol{E})$. Bu tanımın üzerinde arama yapacağı dili x harfleri kümesinden oluşan bir dil (language) ($\Sigma^* x$) *olarak tanımlarsak, otomatı oluşturan b*u terimlerin metin arama işlemi için uyarlanmış hallerini şöyle açıklayabiliriz.

- Q kümesi x harflerinden oluşan ve bütün olasılıkları ifade eden kümedir ve $\boldsymbol{Q}=\{\in, x[0], x[0 .. 1], \ldots, x[0 .. m\text{-}2], x\}$ olarak gösterilebilir.
- q_0, otomatın başlangıç terimidir ve başlangıçta otomatta boş küme olduğunu düşünürsek $q_0=\in$; olarak gösterilebilir.
- T, otomatın üzerinde tanımlı olduğu dili oluşturan sembolleri gösterir. Bu örnekte dilimizin $\Sigma^* x$ şeklinde tanımlı olmasından dolayı T = {x} olarak gösterilebilir.
- Son olarak E teriminin tanımını Q kümesinde tanımlı bütün "q" değerleri ve Σdilinde tanımlı herhangi bir "a" terimi için (q, a, qa) şeklinde tanımlayabiliriz. Bu tanımın doğruluğu, "a" değerinin "x" değerinin bir ön eki olması şartına bağlayabiliriz. Şayet bir ön ek olarak kabul edilmezse, yine (q,a,p) üçlüsünün E kümesinde olması şartıyla, p değeri x'in bir ön eki olan qa için en uzun ek olur.

Yukarıdaki otomat inşa edilirken m uzunluğunda bir kelime olduğu kabul edilirse (ki yukarıdaki Q kümesinin elemanları bu uzunlukta verilmiştir) inşa için gereken süre O(m + r) ve inşa için gereken hafıza ise O(mr) olarak hesaplanabilir.

Algoritmanın araması

Algoritma, yukarıdaki şekilde bir DFA inşa ettikten sonra bu DFA'i kullanarak hedef metin içerisinde arama yapar. Bu arama işlemi n boyutundaki bir metin için O(n) zamanda yapılabilir.

Bu arama işleminin çalışmasını bir örnek üzerinden inceleyelim.

Algoritmanın çalışması

Örnek olarak "bilgi" kelimesini "bilbilgisayarkavramlari" metninin içerisinde aramaya çalışalım.

Öncelikle kelimemiz için bir belirli otomat (DFA) oluşturmamız gerekiyor.

Dilimizdeki ($\Sigma^* x$) *alfabemizi tanımlayacak olursak:*

{a,b,g,i,k,l,m,r,s,v,y} harflerinden oluşan bir dil olarak tanımlanabilir.

Bu dil üzerinde belirli bir sonlu otomat "bilgi" kelimesi için aşağıdaki şekilde inşa edilebilir:

d 0 { a -> 0 b -> 1 g -> 0 i -> 0 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

d 1 { a -> 0 b -> 1 g -> 0 i -> 2 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

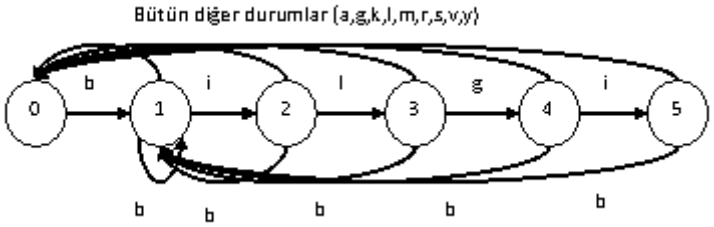
d 2 { a -> 0 b -> 1 g -> 0 i -> 0 k -> 0 l -> 3 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

d 3 { a -> 0 b -> 1 g -> 4 i -> 0 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

d 4 { a -> 0 b -> 1 g -> 0 i -> 5 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

d 5 { a -> 0 b -> 1 g -> 0 i -> 0 k -> 0 l -> 0 m -> 0 r -> 0 s -> 0 v -> 0 y -> 0 }

Yukarıda verilen durumlar için gidilecek durumlar listelenmiştir. Bu otomatı daha iyi anlayabilmek için aşağıdaki şekilde çizebiliriz:



Yukarıdaki otomatta, 0. Durum başlangıç durumu olmaktadır. Ayrıca 5. Duruma her ulaşılmasında aranan metin bulunmuş demektir.

Yukarıdaki otomatın örnek metin üzerindeki çalışmasını adım adım açıklamaya çalışalım:

```
bilbilgisayarkavramlari
  |
```

İlk olarak hedef metinden b harfi alınıyor. Bu harf otomatımızın ilk geçişine tekabül ediyor ve dolayısıyla 0. Durumdan 1. Duruma geçiyoruz.

Geçiş: 0.b -> b

1. durumdayken ikinci harfi hedef metinden okuyoruz:

```
  Bilbilgisayarkavramlari
   |
```

Şu anda 1. Durumdan 2. Duruma geçmek için şartımız oluştu

Geçiş: b.i -> bi

Bu işlem aşağıdaki şekilde devam ettirilir:

```
  BIlbilgisayarkavramlari
    |
  Geçiş: bi.l -> bil
  BILbilgisayarkavramlari
```

```
   |
Geçiş: bil.b -> b
```

Yeni gelen harf, otomattaki 3. Durumdan 1. Duruma dönmeyi gerektiren b harfidir. Dolayısıyla aradığımız kelimenin ilk 3 harfi bulunmasına karşılık kelimenin tamamı bulunamamış ve başa dönülmüştür.

```
bilBilgisayarkavramlari
    |
Geçiş: b.i -> bi
bilBIlgisayarkavramlari
     |
Geçiş: bi.l -> bil
bilBILgisayarkavramlari
      |
Geçiş: bil.g -> bilg
bilBILGisayarkavramlari
       |
Geçiş: bilg.i -> bilgi
bilBILGIsayarkavramlari
        |
Geçiş: bilgi.s -> 0
bilBILGIsayarkavramlari
         |
Geçiş: 0.a -> 0
bilBILGIsayarkavramlari
          |
Geçiş: 0.y -> 0
bilBILGIsayarkavramlari
           |
Geçiş: 0.a -> 0
bilBILGIsayarkavramlari
            |
Geçiş: 0.r -> 0
bilBILGIsayarkavramlari
             |
Geçiş: 0.k -> 0
bilBILGIsayarkavramlari
              |
Geçiş: 0.a -> 0
bilBILGIsayarkavramlari
               |
Geçiş: 0.v -> 0
bilBILGIsayarkavramlari
                |
Geçiş: 0.r -> 0
bilBILGIsayarkavramlari
                 |
Geçiş: 0.a -> 0
bilBILGIsayarkavramlari
                  |
Geçiş: 0.m -> 0
bilBILGIsayarkavramlari
                   |
Geçiş: 0.l -> 0
bilBILGIsayarkavramlari
                    |
Geçiş: 0.a -> 0
bilBILGIsayarkavramlari
                     |
```

```
Geçiş: 0.r -> 0
bilBILGIsayarkavramlari
                      |
Geçiş: 0.i -> 0
```

Sonuç olarak, yukarıdaki büyük harfle gösterilen alanda, aranan kelime bulunmuştur.

Algoritmanın kodlanması

Algoritmanın kodu C dili için aşağıdaki şekilde yazılabilir.

```
1  // DFA inşa aşaması için m uzunluğundaki
2  // x kelimesi ve referans olarak çağrılan
3  // (call by reference) otomat grafı alınır
4  void insa(char *x, int m, Graf otomat) {
5     int i, durum, hedef, eskihedef;
6      // otomattan başlangıç düğümü alınarak kelime boyu
7      // olan m karakter için işlem yapılıyor
8     for (durum = baslangicAl(otomat), i = 0; i < m; ++i) {
9        eskihedef = hedefAl(otomat, durum, x[i]);
10       hedef = yeniDugum(otomat);
11       hedefAta(otomat, durum, x[i], hedef);
12       DugumKopyala(otomat, hedef, eskihedef);
13       durum = hedef;
14    }
15    sonluAta(otomat, durum);
16 }
17  //www.bilgisayarkavramlari.com
18
19  // m boyutundaki x kelimesini n boyutundaki y
20  // metni içinde arayan algoritma
21 void otomat(char *x, int m, char *y, int n) {
22    int j, durum;
23    Graf otomat;
24
25    // ön işleme yapılarak otomat inşa edilecek
26    otomat = yeniotomatomaton(m + 1, (m + 1)*ABOY);
27    insa(x, m, otomat);
28
29    // otomatın çalıştırılması ve arama işlemi
30    for (durum = baslangicAl(otomat), j = 0; j < n; ++j) {
31       durum = hedefAl(otomat, durum, y[j]);
32       if (sonluMu(otomat, durum))
33          GOSTER(j - m + 1);
34    }
35 }
```

Yukarıdaki kodda, arama işlemini yapan ana fonksiyon, otomat fonksiyonudur. Arama işlemi sırasında bu fonksiyon çağrılarak işlem başlatılır. Bu fonksiyonun otomatı inşa etmek için kullandığı inşa fonksiyonu, kodun 27. Satırında çağrılmıştır. Bu çağırma işleminden önce bir otomaton oluşturulmuş ve atıf ile çağırma (call by reference) kullanılarak inşa fonksiyonuna geçirilmiştir.

Kod, 8 ile 14. Satırlar arasında bir DFA inşa etmekte ve 30 ile 33. Satırlar arasında ise bu otomatı kullanarak hedef metin içerisinde arama yapmaktadır.

SORU-11: Kaba Kuvvet Metin Arama Algoritması (Bruteforce Text Search Algorithm) hakkında bilgi veriniz.

1. Algoritmanın başarısı
2. Algoritmanın çalışması ve bir örnek
3. Algoritmanın kodlanması

Bilgisayar bilimlerinde bir metnin içerisinde başka bir metnin aranması için kullanılan en ilkel ve dolayısıyla en düşük performanslı arama algoritmasıdır (search algorithm). Algoritma hedef metinde, aranan metni harf harf bulmaya çalışır. Bu yapısından dolayı diziler üzerinde kullanılan doğrusal arama (linear search) algoritmasına oldukça benzer ve literatürde doğrusal metin araması (linear text search) ismi de verilmektedir.

1 Algoritmanın başarısı

Kaba kuvvet algoritması, isminden de anlaşılacağı üzere çok zeki olmayan ve başarısını bilgisayarın yüksek hızda çalışmasından alan bir algoritmadır. Algoritma basitçe metinin tamamını çok zeki olmayan bir şekilde dolaşır ve aranan kelimenin ilk harfini bulana kadar bu işleme devam eder. Bulduğu anda geri kalan harfleri eşleştirmeye çalışır. Şayet harflerden birisini eşleştiremezse, kelimenin ilk harfini bulduğu yere geri dönerek arama işlemine devam eder. Gerçi çok zeki olmadığı için kelimenin tamamını eşleştirse bile yine de ilk harfi bulduğu yere geri dönerek arama işlemine devam eder.

2 Algoritmanın çalışması ve bir örnek

Kaba kuvvet metin arama algoritmasının (bruteforce text search algorithm) çalışmasını aşağıdaki örnek üzerinden anlamaya çalışalım.

Aranan kelimemiz : bilgi

Aranan metin: wwwbilgisayarkavramlaricom

Olarak veriliyor olsun. Bu durumda algoritma ilk harften başlayarak “bilgi” kelimesini aranan metin içerisinde bulmaya çalışacaktır.

```
wwwbilgisayarkavramlaricom
b....
```

Öncelikle ilk harften başlanarak harfler karşılaştırılıyor. Aranan kelimenin ilk harfi “b” olduğu için bu harf bulunana kadar arama işlemi devam ediyor:

```
wwwbilgisayarkavramlaricom
 b....
wwwbilgisayarkavramlaricom
  b....
wwwbilgisayarkavramlaricom
   BILGI
```

“b” harfi ile başlayan bir yer bulundu. Artık diğer harfler karşılaştırılabilir. Sırasıyla “I”,”L”.. harfleri karşılaştırılıyor ve harfler tuttuğu sürece karşılaştırma işlemi devam ediyor. Şayet harflerden birisi beklenen sırada gelmezse karşılaştırma işlemi kesilip kalınan yerden devam ediliyor.

```
wwwBILGIsayarkavramlaricom
    b....
```

Aslında bu harflere bakılmış olmasına rağmen yine de aranıyor. Malum kaba kuvvet arama algoritması akıllı bir algoritma değildir ve bütün ihtimalleri dener. Dolayısıyla aslında bakmış olduğumu ve bakılmasının bir anlamı olmayan bu harflere de bu algoritma kapsamında bakılıyor.

```
wwwBILGIsayarkavramlaricom
     b....
```

Arama işlemi aşağıdaki şekilde geri kalan harflerin kontrolü ile devam ediyor:

```
wwwBILGIsayarkavramlaricom
      b....
wwwBILGIsayarkavramlaricom
       b....
wwwBILGIsayarkavramlaricom
        b....
wwwBILGIsayarkavramlaricom
         b....
wwwBILGIsayarkavramlaricom
          b....
wwwBILGIsayarkavramlaricom
           b....
wwwBILGIsayarkavramlaricom
            b....
wwwBILGIsayarkavramlaricom
             b....
wwwBILGIsayarkavramlaricom
              b....
wwwBILGIsayarkavramlaricom
               b....
wwwBILGIsayarkavramlaricom
                b....
wwwBILGIsayarkavramlaricom
                 b....
wwwBILGIsayarkavramlaricom
                  b....
wwwBILGIsayarkavramlaricom
                   b....
wwwBILGIsayarkavramlaricom
                    b....
```

Yukarıdaki arama işlemi sonucunda büyük harfle gösterilen kısımda aranan kelime bulunmuştur. Toplam 26 harflik bir metin içerisinde 5 harflik “bilgi” kelimesi aranmıştır ve 22 adımda bulunmuştur.

Artın aranan kelimenin sığmayacağı son harflere bakılmamıştır. Örneğin aranan metnin son 4 harfi olan “icom” alt metni (sub string) üzerinde arama işlemi anlamsızdır çükü buraya “bilgi” kelimesi sığamaz.

3 Algoritmanın kodlanması

Algoritmanın C, C++, JAVA veya C# gibi diller için kodlaması aşağıdaki iki iç içe döngü (nested loop) şeklinde yapılabilir:

```
1   //hedef metinden aranan metnin boyu kadar eksik
2   // sayidaki harfi karşılaştıran dış döngü
3   int j;
4   for (j = 0; j <= n - m; ++j) {
5       // şayet aranan harfler eşleşiyorsa arama
6       // işlemine devam eden döngü
7       int i;
8       for (i = 0; i < m && x[i] == y[i + j]; i++);
9       // ihtimal olarak aranan metnin boyutu
10      // geçildiyse sonuç bulunmuştur
11      if (i >= m){
12          GOSTER(j);
13      }
14      //www.bilgisayarkavramlari.com
15  }
```

Yukarıdaki kodda, n boyutundaki y hedef dizgisi (string) içerisinde m boyutunda x dizgisinin arandığı kabul edilmiştir. Döngü basitçe n-m kere dönmektedir (yukarıdaki örnekte 22 kere dönemsi gibi) ve şayet aranan kelimenin ilk harfi bulunursa, 8. Satırdaki iç döngü dönmeye başlar. Harfler tutuştukça dönme işlemi devam ettirilir. Nihayetinde 11. Satırdaki koşul gerçekleşince, yani tutuşan harflerin sayısı, aranan kelimenin boyutunu geçince, yani aradığımız kelimedeki harf kadar harf, birbirini tutunca sonuç gösterilir. Bu işlem metnin sonuna kadar tekrarlanır.

SORU-12: Base64 hakkında bilgi veriniz.

Veri güveliği konusunda kullanılan kodlama (encoding) algoritmalarından birisidir. Basitçe bir bilginin farklı semboller ile gösterilmesi işlemidir. Bu semboller alfabedeki harflerin büyük/küçük sıralanması ve sayılardan oluşur. Bir base64 sisteminin kullandığı semboller aşağıda verilmiştir:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

Yukarıda toplam 64 sembol bulunmaktadır. Dolayısıyla her sayıya bir karşılık gelir. Örneğin 0 sayısal değeri için A , 63 sayısal değeri için / sembolü gibi.

Bir metnin base64′e çevrilmesi işlemi ise ikilik tabanda
ascii kodları ile metni kodlayıp ardından metni base64 için tamamlamak ve yukarıdaki sembollerle ifade etmektir. Bu işlemin nasıl yapıldığını adım adım inceleyelim.

Base64 ile kodlamak istediğimiz mesajımız “www.bilgisayarkavramlari.com” olsun. Bu mesajdaki her harfin karşılığı olan ASCII kodunu bulalım:

119 119 119 46 98 105 108 103 105 115 97 121 97 114 107 97 118 114 97 109 108 97 114 105 46 99 111 109

Yukarıdaki dizilimde her sembol için karşılığı olan ASCII kodu yazılmıştır. Buna göre örneğin mesajımızın ilk harfi olan w için 119 veya mesajın son harfi olan m için 109 sayısı tablodan bulunmuştur.

Yukarıdaki çevrilmiş ve onluk tabandaki sayıları ikilik tabana çevirirsek:

0111 0111 0111 0111 0111 0111 0010 1110 0110 0010 0110 1001 0110 1100 0110 0111
0110 1001 0111 0011 0110 0001 0111 1001 0110 0001 0111 0010 0110 1011 0110 0001
0111 0110 0111 0010 0110 0001 0110 1101 0110 1100 0110 0001 0111 0010 0110 1001
0010 1110 0110 0011 0110 1111 0110 1101

Mesajını buluruz. Yukarıdaki mesajda her 8 bit (ikil) ASCII tablosunda bir sayıya karşılık gelir. Örneğin mesajın ilk 8 biti (ikili) 0111 0111, oluk tabana çevrilirse 119 yapar ve bu değer tablodaki w sembolüne karşılık gelir. Yukarıda toplam 224 bit (ikil) bulunmaktadır. Çünkü mesaj uzunluğu 28 karakterdir ve her karakter için 8 bit kullanılmıştır.

Yukarıdaki mesajın base64 ile kodlanması sırasında mesajın eksik bulunan bitlerinin tamamlanması gerekir. Buradaki hesap basitçe mesajın boyutunun 24′ün katı olana kadar 8′er bitlik 0 (sıfırların) eklenmesidir.

Mesajda toplam 224 bit bulunuyor ve 224, 24′e tam olarak bölünemiyor. Bu durumda sayımız 24′e tam bölünebilmesi için 240′a tamamlanması gerekir. Sonuçta mesajımız aşağıdaki şekilde olacaktır:

0111 0111 0111 0111 0111 0111 0010 1110 0110 0010 0110 1001 0110 1100 0110 0111
0110 1001 0111 0011 0110 0001 0111 1001 0110 0001 0111 0010 0110 1011 0110 0001
0111 0110 0111 0010 0110 0001 0110 1101 0110 1100 0110 0001 0111 0010 0110 1001
0010 1110 0110 0011 0110 1111 0110 1101 0000 0000 0000 0000

Görüldüğü üzere sonuna 16 adet 0 eklenmiştir. Şimdi artık mesajımız base64 kodlamasında işlenmeye hazırdır. Mesajı bu sefer 6 bitlik parçalara bölüp onluk sisteme çevirebiliriz. Buradaki amaç bizim kodlamamızdaki 64 sembolden hangisine karşılık geldiğini bulmaktır.

Örneğin mesajın ilk 6 biti 011101 ‘dir ve onluk sistemde 29 yapar. Bu bizim kodlamamızda yukarıda verilen sembollerden 29. Sembol ile gösterileceğini ifade eder. Bu sembol A harfi 0 olarak kabul edilip sayılırsa d harfi olarak bulunur. Benzer şekilde bir sonraki 6lık grup alınırsa 110111 sayısını onluk tabanda 55 sayısı bulunur ve bizim kodlamamızda 55 sayısı ile 3 sembolü gösterilmektedir.

Yukarıdaki bu işlemi bütün mesaj için uygularsak sonuçta www.bilgisyarkavramlari.com mesajı için aşağıdaki sonuç bulunur:

d3d3LmJpbGdpc2F5YXJrYXZyYW1sYXJpLmNvbQ==

Yukarıdaki mesaj tam bir base64 çevrimidir. Burada mesajın sonunda bulunan == sembollerinin bizim kodlamamızda yer almadığına dikkat ediniz. Bu semboller mesajın ayrılmasını sağlayan son eklerdir.

Base64 örneğin MIME protokolünün içerisinde gönderilen mesajlara uygulanan bir kodlamadır.

SORU-13: Mesaj Özetleri (Message Digests) hakkında bilgi veriniz.

Özellikle veri güvenliği ve veri bütünlüğü (data integrity) konularında kullanılan mesaj özetleri aslında birer özetleme fonksiyonudurlar (hashing functions). Buna göre büyük bir veriden nispeten daha küçük bir özet üretilir ve bu özetten orijinal veriye geri dönülemeyeceği varsayılır.

Mesaj özetlerinden en meşhurları MD5 ve SHA-1 algoritmalarıdır.

MD5 : Message digest (mesaj özeti) kelimelerinin kısaltmasıdır ve 5. Versiyonunu ifade etmektedir.

SHA-1 : Secure Hashing Algorithm (güvenli özetleme algoritması) ise 1. Versiyonudur ve daha önceki 0. Versiyonuna göre daha güvenlidir. Çoğu kaynakta sadece SHA olarak da geçmektedir.

SORU-14: Dinamik Markof Kodlaması ile Sıkıştırma (Data Compression Using Dynamic Markov Coding) hakkında bilgi veriniz.

Not: Bu yazı Dr. Banu Diri'nin veri sıkıştırma dersi sırasında hazırladığım rapordan alıntıdır. Kendisine buradan teşekkürü bir borç bilirim.

Bu yazının amacı, Dinamik Markov Coding kullanılarak sıkıştırma yöntemini incelemektir. Bu yazı, "Data Compression Using Dynamic Markov Modelling" ve G. V. CORMACK AND R. N. S. HORSPOOL tarafından yazılmış olan makalenin incelenmesini içermektedir.

İçerik
1. Giriş
2. Markof zincirinin dinamik zamanda oluşturulması
3. Sıkıştırma Örneği
4. Sonuç
5. Referanslar

Dynamic Markov Coding yeni bir sıkıştırma algoritması olmayıp, mevcut sıkıştırma algoritmalarının üzerine getirilen yeni bir yaklaşımdır. Yaklaşım binary kodlama üzerine kurulu olup, u yaklaşımda başarı veri sıkıştırması sırasında gelen .bitlerin tahminine ve değerlendirmesine dayanmaktadır. Markov chain modelinin oluşturulması ve bu modelin kendisini her gelen yeni bite göre güncellemesi bu çalışmanın temelini oluşturmaktadır.

Ayrıca Dinamik Markov Coding için arithmetic coding yöntemlerinden birisi olan gauzzo yöntemi de bu çalışmada incelenmiştir.

1. Giriş

Bilindiği üzere günümüzde oldukça fazla sayıda sıkıştırma algoritması bulunmaktadır. Bu algoritmaların daha verimli çalışması için yapılan çeşitli çalışmalardan birisi de istatistiksel bir modele dayanan Makrov Kodlamasıdır. İlk olarak rus matematikçi Andrey Markov tarafından geliştirilen bu yöntem stochastic process üzerine kurulmuştur. Bu yazıda stochastic process ve markov chain konularına giriş yaptıktan sonra markov coding ile verinin ifadesine ve bu ifade biçiminin nasıl dinamik olabileceği incelenecektir. Son olarak bu ifade biçiminin gauzzo yöntemi ile sıkıştırılması anlatılacaktır. .

2. Markov Chainin Dinamik zamanda oluşutrulması

Veri sıkıştırmasında, adaptive veya adaptive olmayan yöntemler kullanılabilmektedir. Dynamic Markov Coding, adaptive bir yöntem önermektedir. Buna göre verinin gelmesi durumuna göre markov modelimizi güncellememiz gerekmektedir.

Buna göre grafik boş iken başlanır ve aşağıdaki şekillerde veriler güncellenir:

§ Prob {sayı = 0 | bulunulan node = A} = $n0 / (n0 + n1)$

§ Prob {sayı = 1 | bulunulan node = A} = $n1 / (n0 + n1)$

§ Prob {sayı = 0 | bulunulan node = A}

= $(n0 + c) / (n0 + n1 + 2c)$

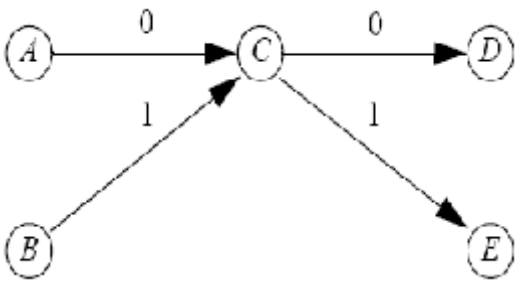
§ Prob {sayı = 1 | bulunulan node = A}

= $(n1 + c) / (n0 + n1 + 2c)$

buradaki n1: 1lerin sayısı, n2: 2lerin sayısı ve c, 0 dan büyük herhangi bir sabit sayıyı ifade etmektedir. Gelen 0 ve 1 lerin sayısı büyüdükçe, c değeri anlamını yitirmektedir.

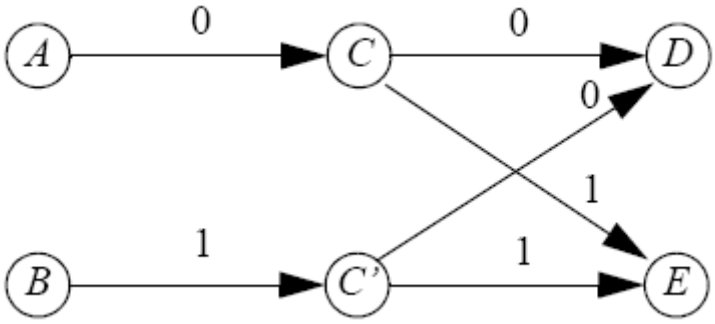
Örnekte yukarıda verilen formülün kullanılması durumunda markov modeli için istatistiksel değerleri içeren kolların inşa edilmesi mümkün olmaktadır.

Grafik inşası sırasında karşılaşılan bir diğer sorun, grafiğin önceki değeri unutması ve bulunan bir duruma nereden geldiğinin anlaşılmasını engelleyen karmaşa ihtimalleridir. Bu durum aşağıdaki şekilde gösterilmiştir:



Yukarıdaki grafik, inşa edilmiş olan markov chain model içerisinde bir yada daha çok yerde görülebilen bir durumdur. Buna göre C durumundan D ve E durumlarına belirli (deterministic) bir şekilde gidilebilmektedir, yani C durumundayken 0 ve 1 gelme olasılıkları belirlenmiştir. Ayrıca C durumuna 0 veya 1 ile gelebileceğimizi bilmekteyiz. Ancak C durumuna 0 ile mi 1 ile mi gelindiği tam olarak bilinmemektedir.

Stochastic processler hatırlanacak olursa, bir sonraki durum önceki durumlara bağlı olmaktadır dolayısıyla C durumuna hangi sayı ile gelindiği bilinmelidir. Bu durumda önerilen çözüm clonlama yöntemidir.



Yukarıdaki grafik, belirsiz C durumunun klonlanmış halini göstermektedir. Buna göre A durumundan C durumuna 0 ile gelinebilmekteyken artık B durumundan C durumuna gidiş bulunmamakta bunun yerine yeni klonlanmış olan C' durumuna gidilebilmektedir. Dolayısıyla artık C durumuna geliniyorsa bunun 0 ile, C' durumuna geliniyorsa bunun 1 ile olduğu bilinebilmektedir. Ve D durumuna gelinme ihtimali ancak 00 veya 10 olurken E durumu 01 veya 11 ihtimallerine cevap vermektedir.

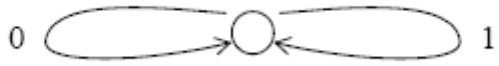
Klonlama özelliği ilk başlarda oldukça faydalı görülmesine karşılık, zaman içerisinde hafıza sorunları ve kompleksliğin artması gibi sebeplerle durdurulmalıdır. Bu işlemin ne zaman durdurulacağı ise iki ihtimale bağlanmıştır bunlar:

Bir node (düğüm) kendisine mevcut noddan gitmek için kullanılan Transation sayısı eşik değeri aştığında ve

Mevcut noddan kendisine gitmek için kullanılan bütün Transitionların sayısı eşik değerini aştığında clonelanır

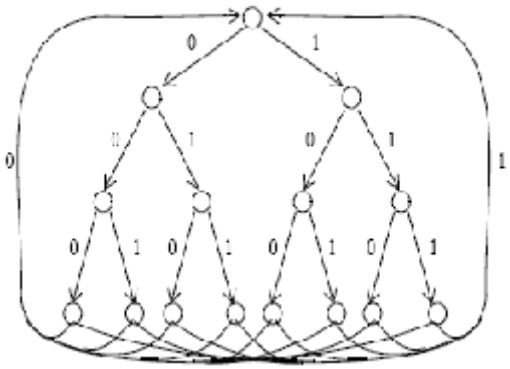
(bkz. Kirchoff Kanunu)

Bu modelde bir diğer problem ise başlangıçta kullanılacak olan markov chain modeldir. Bunun için önerilen boş bir grafik ile başlamak veya mevcut verinin en küçük yapıtaşını tutan bir grafik ile başlamaktır. Örneğin



Yukarıdaki grafik 1 veya 0 ile dolaşılabilen tek bir düğüm içermektedir. Şayet veri modeli, 0 ve 1lerden oluşuyorsa kullanılabilecek en basit model yukarıdaki şekildedir.

Benzer şekilde ASCII kodlarından oluşan ve 256 sembolü gösteren bir veri modeli oluşturulacaksa bunun için 8 seviyeli bir düğüm yapısı kullanılabilir



Yukarıdaki grafik, fiziksel sebeplerden dolayı 3 seviyeli bir ağacı göstermektedir. 8 seviyeli olan ağaç da benzer şekilde çizilebilir.

3. Sıkıştırma örneği

Yukarıdaki bölümlerde anlatılan modeller bir sıkıştırma metodolojisi olan dinamik markov modelin inşasını ve adaptasyonunu içermektedir. Bu model basit bir şekilde herhangi bir aritmetik sıkıştırma yöntemine uygulanabilir örnek model olarak gauzzo sıkıştırması ele alınacak olursa, sıkıştırma yönteminin karakteristiği aşağıdaki şekilde olacaktır:

Guazzo katsayılarının hesaplanmasında sayı düzeni bozulmadan ondalıklı sayıya çevrilir örneğin 01101 sayısı è 0.01101 olur ve oranı (13/16) olur.

Dolayısıyla sayı domaini 0 ile 0.11111… sayıları arası sayılardan oluşmaktadır. Ve bu sayılar arasında binary dallanma mümkündür örneğin ilk değerin 0 olmasına göre 0.011.. olurken 1 olmasına göre 0.111… olmaktadır

Bu yöntemde oranlara göre markov chain içinde hareket mümkün olup herhangi bir arithmetik kodlamada kullanılan alçak ve yüksek değerlere eşitlenebilmektedir.

4. Sonuç

Bu yazıda mevcut sıkıştırma yöntemlerine getirilen bir yenilik olarak dinamik markov chain kodlaması incelenmiştir başarı oranı aşağıdaki grafikte verilmiştir:

Compression program	Source file			
	Formatted text[a]	Unformatted text[b]	Object code[c]	C source code[d]
Adaptive Huffman (compact)	59.7	61.8	75.6	62.9
Normal Ziv-Lempel (LZW)	38.2	42.8	95.4	40.8
Bit-oriented Ziv-Lempel (LZ-2)	74.2	83.6	91.3	86.7
Cleary and Witten (CW)	26.5	30.2	68.4	26.3
Dynamic Markov (DMC)	27.2	31.8	54.8	27.5

bu yöntem yeni bir sıkıştırma algoritması olmayıp mevcut algoritmalara (tercihen aritmetik kodlama) getirilmiş yeni bir yaklaşımdır. Ve verinin binary olarak tutulmasını ve veri akışına göre istatistiksel olarak modellenerek markov chain ile gösterilmesini önermektedir.

SORU-15: Atomluluk (Atomicity) hakkında bilgi veriniz.

Latince bölünemez anlamına gelen atom kökünden üretilen bu kelime, bilgisayar bilimlerinde çeşitli alanlarda bir bilginin veya bir varlığın bölünemediğini ifade eder.

Örneğin programlama dillerinde bir dilin atomic (bölünemez) en küçük üyesi bu anlama gelmektedir. Mesela C dilinde her satır (statement) atomic (bölünemez) bir varlıktır.

Benzer şekilde bir verinin bölünemezliğini ifade etmek için de veri tabanı, veri güvenliği veya veri iletimi konularında kullanılabilir.

Örneğin veri tabanında bir işlemin (transaction) tamamlanmasının bölünemez olması gerekir. Yani basit bir örnekle bir para transferi bir hesabın değerinin artması ve diğer hesabın değerinin azalmasıdır (havale yapılan kaynak hesaptan havale yapılan hedef hesaba doğru paranın yer değiştirmesi) bu sıradaki işlemlerin bölünmeden tamamlanması (atomic olması) gerekir ve bir hesaptan para eksildikten sonra, diğer hesapa para eklenmeden araya başka işlem giremez.

Benzer şekilde işletim sistemi tasarımı, paralel programlama gibi konularda da bir işlemin atomic olması araya başka işlemlerin girmemesi anlamına gelir.

Örneğin sistem tasarımında kullanılan check and set fonksiyonu önce bir değişkeni kontrol edip sonra değerini değiştirmektedir. Bir değişkenin değeri kontrol edildikten sonra içerisine değer atanmadan farklı işlemler araya girerse bu sırada problem yaşanması mümkündür. Pekçok işlemci tasarımında buna benzer fonksiyonlar sunulmaktadır.

Genel olarak bölünemezlik (atomicity) geliştirilen ortamda daha düşük seviyeli kontroller ile sağlanır. Örneğin işletim sistemlerinde kullanılan semafor'lar (semaphores), kilitler (locks), koşullu değişkenler (conditional variables) ve monitörler (monitors) bunlar örnektir ve işletim sisteminde bir işlemin yapılması öncesinde bölünmezlik sağlayabilirler.

Kullanılan ortama göre farklı yöntemlerle benzer bölünmezlikler geliştirilebilir. Örneğin veritabanı programlama sırasında koşul (condition) veya kilit (lock) kullanımı bölünmezliği sağlayabilir.

SORU-16: Huffman Kodlaması (Huffman Encoding) hakkında bilgi veriniz.

Bilgisayar bilimlerinde veri sıkıştırmak için kullanılan bir kodlama yöntemidir. Kayıpsız (lossless) olarak veriyir sıkıştırıp tekrar açmak için kullanılır. Huffman kodlamasının en büyük avantajlarından birisi kullanılan karakterlerin frekanslarına göre bir kodlama yapması ve bu sayede sık kullanılan karakterlerin daha az, nadir kullanılan karakterlerin ise daha fazla yer kaplamasını sağlamasıdır.

Şayet bütün karakterlerin dağılımı eşitse yani aynı oranda tekrarlanıyorsa, bu durumda Huffman kodlaması aslında blok sıkıştırma algoritması (örneğin ASCII kodlama) ile aynı başarıya sahiptir. Ancak bu teorik durumun gerçekleşmesi imkansız olduğu için her zaman daha başarılı sonuçlar verir.

Örneğin sadece 8 sembolden oluşan bir dilimiz olsun (Örneğin a,b,c,d,ef,g,h harflerinden oluşan bir dil düşünelim) Bu dili kodlamak için 3 bit yeterlidir (2^3=8 olduğuna göre 8 farklı dili ikilik tabanda kodlayabiliriz)

Bu durumda örneğin harflerin değerlerini aşağıdaki şekilde oluşturabiliriz:

a 000

b 001

c 010

d 011

e 100

f 101

g 110

h 111

Her harf için farklı bir kodlama yapılan dilde örneğin “baba caddede gec” mesajını kodlayacak olursak:

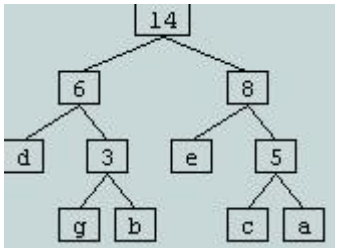
001000001000 010000011011100011100 110100010

şeklinde bir sonuç elde ederiz. Görüldüğü üzere kodlama sonucunda harf sayısının üç misli kadar bit kullanılmak zorundadır (14 harf için 14×3=52 bit gerekmektedir).

Huffman kodlaması ile bu mesajı sıkıştıracak olsaydık. Öncelikle harflerin mesajdaki sıklıklarını gösteren biraşağıdaki istatistiğin çıkarılması gerekirdi:
a3

b2
c2
d3
e3
f0
g1
h0

Yukarıda her harfin kullanılma sıklıkları sıralanmıştır. Bu istatistiksel veriye dayanarak bir ağaç oluşturulması gerekir.



Yukarıdaki ağaçta dikkat edilirse dilimizdeki harfler ve her harf düğümlerinin birleşim noktalarında ise o harflerin mesajdaki tekrar sayıları bulunmaktadır. Ayrıca istatistiksel olarak birbirine denk olan sıklıktaki düğümler aynı seviyede bulunmaktadır. Örneğin g+b = 3 sıklığa sahip ve d'de 3 sıklığa sahiptir. Bu durumda d ile g ve b'nin birleştiği düğüm aynı seviyede olmaktadır.

Yukarıdaki bu ağaca göre her harfi veren kodlama karşılığı çıkarılır. Bu çıkarma işlemi sırasında ağaçtaki her sağ kola hareket 1, her sol kola hareket 0 olarak okunur. Örneğin g harfinin değeri 010′dır çünkü kökteki 14 değerinin solunda (yani 0) 6 değerinin sağında (yani 1) ve 3 değerinin solundadır yani toplamda 010 değerine sahiptir.

Bu şekilde her harfin kodlama değeri aşağıda verilmiştir:

a 111

b 011

c 110

d 00

e 10

g 010

Yukarıdaki bu kodlamaya göre ilk mesajımızın yeni değeri :

011111011111 1101110000100010 01010110

Şeklinde bulunmuş olur. Dikkat edilirse bu mesajın boyutu 36 bittir ve ilk baştaki 52 bit uzunluğundan daha kısadır.

Huffman Ağacının Oluşturulması

Genel bir soru üzerine, ağacın oluşturulma algoritmasını yayınlıyorum. Ağacı oluşturmanın çeşitli algoritmaları olduğu gibi, en basit olanı, bir rüçhan sırası (öncelik sırası, priority queue) kullanmaktır. Bu sırada, en düşük olasılığa sahip olan düğüm (node), en yüksek rüçhana sahip olacaktır. Algoritmanın adımları aşağıdaki şekildedir:

1. Algoritma tarafından kodlanacak olan her sembol için birer yaprak düğüm, rüçhan sırasına eklenir.
2. Sırada, birden fazla düğüm kaldığı sürece aşağıdaki adımlar döngü halinde yapılır.
a. Sıradan, en yüksek rüçhana sahip iki düğüm alınır. (bu düğümlerin en az kullanım sıklığına sahip olduğunu hatırlayınız)
b. Yeni bir iç düğüm (internal node) oluşturulup, değer olarak bu alınan iki düğümün toplamı atanır.
c. Yeni düğüm ağaca ve sıraya eklenir.
3. Döngü bitip tek düğüm kaldıysa, bu düğüm, kök düğüm yapılır ve algoritma sona erer.

Şimdi, yukarıdaki bu algoritmaya göre ağacımızı oluşturalım. Harfler ve kullanım sıklıkları aşağıdaki şekildedir:

d3,e3,a3,b2,c2,g1

Buna göre bir rüçhan sırası yaparsak:

g1, b2, c2, a3, d3, e3 şeklinde en düşük sıklığan sahip olan en önde olacaktır.(ayrıca eşit öncelikli olanlar, kendi aralarında istenildiği gibi sıralanabilir.)

Algoritmanın 2. adımına geçiyor ve döngümüzü çalıştırmaya başlıyoruz.

Sıradaki en yüksek rüçhana sahip iki düğümü alalım: g1, b2

Toplamlarını hesaplayalım: 1+2 = 3

Yeni çıkan bu düğümü ağaca ve sıraya ekleyelim. Ağaç hali aşağıdaki şekildedir:

```
  3
 /
g1  b2
```

Sıra için bu düğümlerden oluşan ağaca gb3 ismini verirsek, sıramız aşağıdaki hali alır:

c2, a3, d3, gb3, e3

Sıradan iki düğüm daha alıyoruz: c2, a3, toplamları : 2 + 3 = 5, ağaca ekliyoruz:

```
  5
 /
c2  a3
```

d3, gb3, e3, ca5

Sıradan iki düğüm daha alıp ağaç oluşturuyoruz: d3, gb3

```
    6
  /
 d3   3
     /
   g1  b2
```

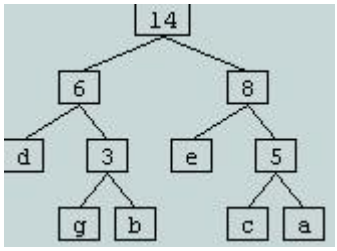
Yeni düğümü sıraya ekleyelim: e3, ca5, dgb5 Sıradaki düğümler: e3, ca5

```
   8
 /
e3   5
    /
   c2  a3
```

Sıraya ekleyelim:

dgb6, eca8

Bu iki düğümü alıp ağacı oluşturduğumuzda ise sonuç ağacımız çıkmaktadır:



SORU-17: Entropi (Entropy, Dağınım, Dağıntı) hakkında bilgi veriniz.

Bir sistemin düzensizliğini ifade eden terimdir. Örneğin entropi terimini bir yazı tura atma işleminde 1 bitlik (ikil) ve %50 ihtimallik bir değer olarak görebiliriz. Burada paranın adil olduğunu ve yazı tura işleminin dengeli bir şekilde gerçekleştiğini düşünüyoruz. Şayet para hileli ise o zaman sistemin entropisi (üretilen sayıların entropisi) %50′den daha düşüktür. Çünkü daha az düzensizdir. Yani hileli olan tarafa doğru daha düzenli sonuç üretir. Örnek olarak sürekli tura gelen bir paranın ürettiği sayıların entropisi 0′dır (sıfırdır).

Entropi terimi ilk kez shannon tarafından bilgisayar bilimlerinde veri iletişiminde kullanılmıştır. Dolayısıyla literatürde Shannon Entropisi (Shannon's Entropy) olarak da geçen kavrama göre bir mesajı kodlamak için gereken en kısa ihtimallerin ortalama değeri alfabede bulunan sembollerin logaritmasının entropiye bölümüdür. Yani kabaca alfabemizde 256 karakter varsa bu sayının logaritmasını (log 256 = 8′dir) mesajın entropisine böleriz. Yani mesajdaki değişim ne kadar fazla ise o kadar fazla kodlamaya ihtiyacımız vardır. Diğer bir deyişle alfabemiz 256 karakterse ama biz sadece tek karakter yolluyorsak o zaman entropy 0 olduğundan 0/256 = 0 farklı kodlamaya (0 bite) ihtiyacımız vardır. Veya benzer olarak her harften aynı sıklıkta yolluyorsak bu durumda 256 /8 = 8 bitlik kodlamaya ihtiyaç duyulur.

Bilgisayar bilimleri açısından daha kesin bir tanım yapmak gerekirse elimizdeki veriyi kaç bit ile (ikil) kodlayabileceğimize entropi ismi verilir. Örneğin bir yılda bulunan ayları kodlamak için kaç ikile ihtiyacımız olduğu ayların dağınımıdır.

Toplam 12 ay vardır ve bu ayları 4 ikil ile kodlayabiliriz:

0000 Ocak

0001 Şubat

0010 Mart

0011 Nisan

0100 Mayıs

0101 Haziran

0110 Temmuz

0111 Ağustos

1000 Eylül

1001 Ekim

1010 Kasım

1011 Aralık

Görüldüğü üzere her ay için farklı bir bilgi girilmiş ve girilen 12 ay için 4 bit yeterli olmuştur. Dolayısıyla yılın aylarının entropisi 4′tür.

Genellikle bir bilginin entropisi hesaplanırken $\log_2 n$ formülü kullanılır. Burada n birbirinden farklı ihtimal sayısını belirler. Örneğin yılın aylarında bu sayı 12′dir ve $\log_2 12 = 3.58$ olmaktadır. 0.58 gibi bir bit olamayacağı için yani bilgisayar kesikli matematik (discrete math) kullandığı için 4 bit gerektiğini söyleyebiliriz.

Farklı bir örnek olarak veri tabanında bulunan kişilerin cinsiyetinin tutulacağı alan 1 bitlik olacaktır. Çünkü kadın/erkek alternatifleri tek bit ile tutulabilir:

0 Kadın

1 Erkek

şeklinde. dolayısıyla cinsiyet alanının entropisi 1′dir.

Yukarıdaki örnekte veritabanında 5 karakterlik bir dizgi (string) alanı tutmak gereksizdir. Çünkü entropi bilgisi bize 1 bitin yeterli olduğunu söyler. 5 karakterlik bilgi (ascii tablosunun

kullanıldığı düşünülürse) 5×8 = 40 bitlik alan demektir ve 1 bite göre 40 misli fazla gereksiz demektir.

Entropi terimi veri güvenliğinde genelde belirsizlik (uncertainity) terimi ile birlikte kullanılır. Belirsizlik bir mesajda farklılığı oluşturan ve saldırgan kişi açısından belirsiz olan durumdur. Örneğin bir önceki örnekteki gibi veri tabanında Kadın ve Erkek bilgilerini yazı olarak tuttuğumuzu düşünelim. Şifreli mesajımız da "fjass" olsun. Saldırgan kişi bu mesajdan tek bir biti bulursa tutulan bilgiye ulaşabilir. Örneğin 3. bitin karşılığının k olduğunu bulursa verinin erkek olduğunu anlayabilir. Dolayısıyla bu örnekte belirsizliğimiz 1 bittir.

SORU-18: Delta Sıkıştırması (Delta Compression) hakkında bilgi veriniz.

Oldukça basit ve hızlı olan bu sıkıştırma algoritmasına göre ardışık olarak gelen veriler arasındaki fark alınarak verilerin boyutu küçültülmüş olur.

Örneğin aşağıdaki sayıları ele alalım:

183 193 233 234 230

Bu sayıları sıkıştırmak için ilk sayıyı sonuç dizisine kopyalıyoruz ve diğer elemanlar ile farkı alıyoruz:

183 10 40 1 -4

Görüldüğü üzere sıkıştırma sonucunda elde edilen sayıların kapladığı yer daha azdır. Bu algoritmada sıkıştırma sonucu verinin kapladığı yerin azalması garanti edilmez. Bunun sebebi örneğin sayılar arasındaki farklar çok yüksekse farlkarı tuttuğumuz sonuç bilgisindeki sayıların orjinal sayılardan büyük olma ihtimalidir.

Algoritmanın açma aşamasında ilk sayıya eklenerek hesaplama yapılır:

183 -> 183

183 + 10 -> 193

193 + 40 -> 233

233 + 1 -> 234

234 + (-4) -> 230

şeklinde veri açılarak orjinal değerler bulunmuş olur.

Algoritmanın sıkıştırma (compress) kısmı aşağıdaki şekilde C dilinde kodlanabilir:

```
int a[]={183,193,233,234,230}; // sıkıştırılacak olan dizimiz

int b[5]; // sonucun bulunacağı dizi

b[0] = a[0]; //ilk eleman aynen kopyalanıyor
```

```
for(int i = 1;i<5;i++){ // dizinin bütün elemanlarını geçiyoruz

b[i] = a[i]-a[i-1];

}
```

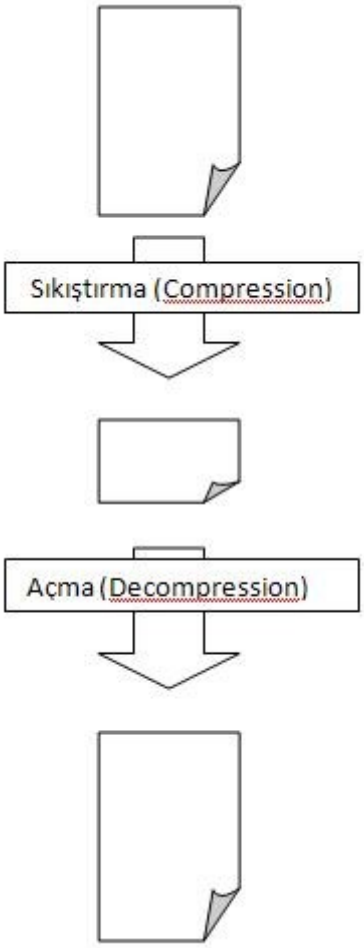
yukarıdaki kodda sonuçta b dizisinde elde edilend eğer sıkıştırılmış değerdir. Algoritmanın açma kısmı için (decompress) aşağıdaki kod yazılabilir:

```
int a[]={183,10,40,1,-4}; // açılacak olan dizimiz

int b[5]; // sonucun bulunacağı dizi

b[0] = a[0]; //ilk eleman aynen kopyalanıyor

for(int i = 1;i<5;i++){ // dizinin bütün elemanlarını geçiyoruz

b[i] = b[i-1]+a[i];

}
```

Yukarıdaki kodların ikiside tek dizi kullanarak da çözülebilmektedir. Algoritma yapısı itibariyle kayıpsız sıkıştırma (lossless compression) bir örnektir çünkü orjinal verinin sıkışıtırılmış hali açıldığında orjinal veri kayıpsız olarak bulunabilmektedir.

SORU-19: Kayıplı Sıkıştırma (Lossy Compression) hakkında bilgi veriniz.

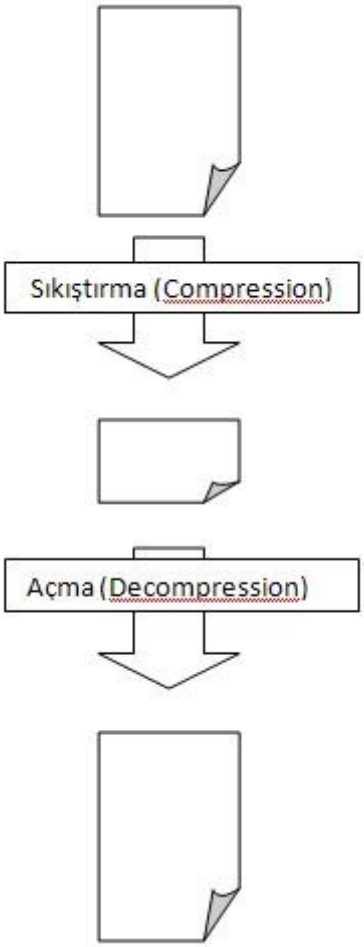
Veri sıkıştırma sırasında kullanılan bir sınıflandırmadır. Buna göre sıkıştırma algoritmaları orjinal veriye dönülüp dönülememesine göre ikiye ayrılır. Kayıplı sıkıştırma (Lossy compression) sınıfındaki sıkıştırmalarda veriyi sıkıştırıp tekrar açtığımızda orjinal verinin bir kısmını kaybederiz.



Yukardaki şekilde giren orjinal metin sıkıştırılmış ardından bu sıkıştırılmış olan metin açılmıştır (decompress). Kayıplı sıkıştırma grubundaki sıkıştırma algoritmalarında sonuç metni ile giriş metni aynı değildir. Sonuç metni giriş metninde daha azdır.

SORU-20: Kayıpsız Sıkıştırma (Lossless Compression) hakkında bilgi veriniz.

Veri sıkıştırma sırasında kullanılan bir sınıflandırmadır. Buna göre sıkıştırma algoritmaları orjinal veriye dönülüp dönülememesine göre ikiye ayrılır. Kayıpsız sıkıştırma (Lossless compression) sınıfındaki sıkıştırmalarda veriyi sıkıştırıp tekrar açtığımızda orjinal veriyi kayıpsız olarak elde ederiz.



Yukardaki şekilde giren orjinal metin sıkıştırılmış ardından bu sıkıştırılmış olan metin açılmıştır (decompress). Kayıpsız sıkıştırma grubundaki sıkıştırma algoritmalarında sonuç metni ile giriş metni aynıdır.