

Dosya Sıkıştırma (File Compression)

İçerik

- Dosya sıkıştırma nedir?
- Dosya sıkıştırma yöntemleri nelerdir?
 - Run-Length Kodlaması
 - Huffman Kodlaması

Dosya Sıkıştırma

- Dosya sıkıştırmaya niçin ihtiyaç duyulmuştur?
 - Minimum alana maksimum veri sığdırmak
 - Daha hızlı aktarım sağlamak, erişim süresini azaltmak
 - Sıradüzensel verileri daha hızlı işleyebilmek

Dosya Sıkıştırma

- Sabit uzunluklu alanlar dosya sıkıřtırmada en iyi adaylardır.
- Daha yoęun bir gösterimle temsil edilen bitlerin sayısı azaltılmaktadır.

Dezavantajları:

- İnsanlar tarafından anlaşılabilir değildir.
- Kodlama için ekstra maliyet gerektirir.
- Kod çözme modüllerine gereksinim vardır. Bu da karmaşıklığı artırır.

Run-Length Kodlama Algoritması

- Aynı byte dizilerinin sık kullanıldığı dosyalar için uygundur.
- Bir dizideki aynı değer birden fazla kez ortaya çıkarsa 3 byte ile yer değiştirir. Bu 3 byte :
 - Özel bir escape karakteri (Run-Length kodu belirtecisi. FFh)
 - Tekrar eden değer
 - Değerin tekrar etme sıklığı

Örnek

Aşağıdaki veri seti bu algoritmaya göre kodlayalım.

22	23	24	24	24	24	24	24	24	25
26	26	26	26	26	26	25	24		

} Veri Seti

Kodlama sonucunda:

22 23 FF 24 07 25 FF 26 06 25 24 elde edilir.

Veri setinin boyutu kodlama sonucunda 18 byte'tan 11 byte'a düşmüştür.





- Bu kodlama, belirli bir miktar alan kazanmayı garanti etmez.
- Bazı durumlarda, sıkıştırılan veri seti, orjinal veri setinden daha büyük olabilmektedir. **NEDEN?**

Mors Kodlaması

- En eski ve yaygın olarak kullanılan kodlamadır.
- Her bir karakter ile 2 çeşit sembol ilişkilendirilmiştir.
- Bazı değerler, diğerlerinden daha fazla sayıda kullanılır.

A	. _
B	_ . . .
...	
E	.
F	. . _ .
...	
T	_
U	. . _
...	

Huffman Kodlaması

-  Huffman kodlaması, kayıpsız sıkıştırma algoritmasıdır.
-  Bu kodlama değişken uzunlukta bir kodlama olup bu kodlama mors kodlamasının tersine veri setindeki karakterlerin frekansına bağlıdır.
-  Algoritma, bir veri setinde daha çok rastlanan bir sembolü daha düşük uzunluktaki kodla, daha az rastlanan sembolü ise daha büyük uzunluktaki kodla temsil edilmesine dayanmaktadır.
-  Veri setindeki sembol sayısına ve bu sembollerin tekrarlama sayısına bağlı olarak %10 ile %90 arasında değişen oranlarda bir sıkıştırma elde edilebilir.

Huffman Kodlaması (Devam)



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



Bir veri kümesini sıkıştırabilmek için bu küme içerisindeki sembollerin tekrar etme sıklıklarının bilinmesi gerekmektedir. Her sembolün ne kadar sıklıkta tekrar ettiğini gösteren tabloya frekans tablosu denir.

Huffman Kodlaması (İşlem Adımları)



İlk olarak, veri setine ait frekans tablosu oluşturulur



Ardından, hangi karakterin hangi bitlerle temsil edileceğini gösteren Huffman ağacı oluşturulur.

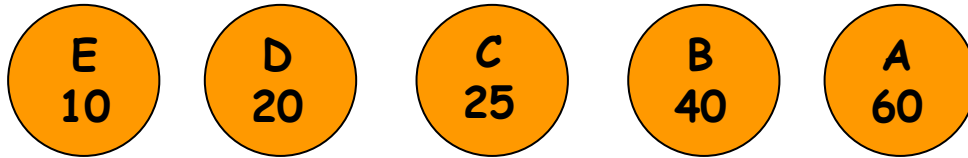
Örnek 1

Veri setine ait frekans tablosu aşağıdaki gibi olsun.

Sembol	Frekans
A	60
B	40
C	25
D	20
E	10

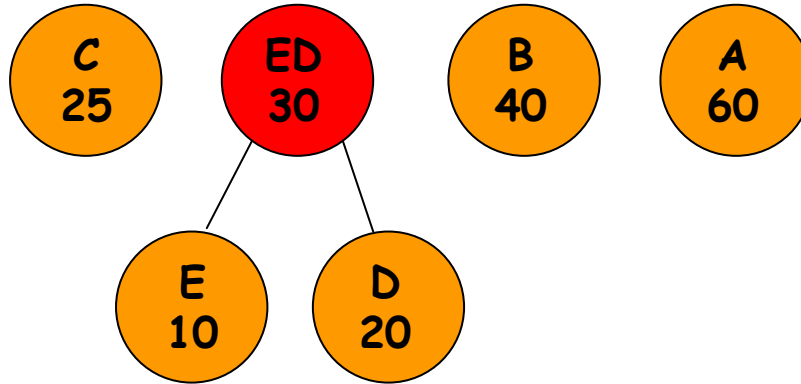
Adım 1

- ✓ Huffman ağacındaki en son düğümleri oluşturacak olan bütün semboller frekanslarına göre küçükten büyüğe doğru sıralanırlar.



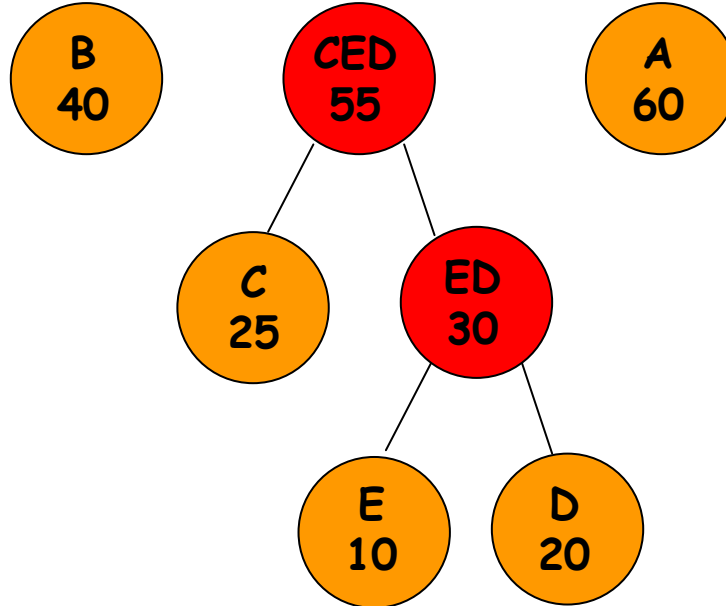
Adım 2

- ✓ En küçük frekansa sahip olan iki sembolün frekansları toplanarak yeni bir düğüm oluşturulur. Oluşturulan bu yeni düğüm var olan düğümler arasında uygun yere yerleştirilir. Bu yerleştirme frekans bakımından küçüklük veya büyüklüğe göredir.

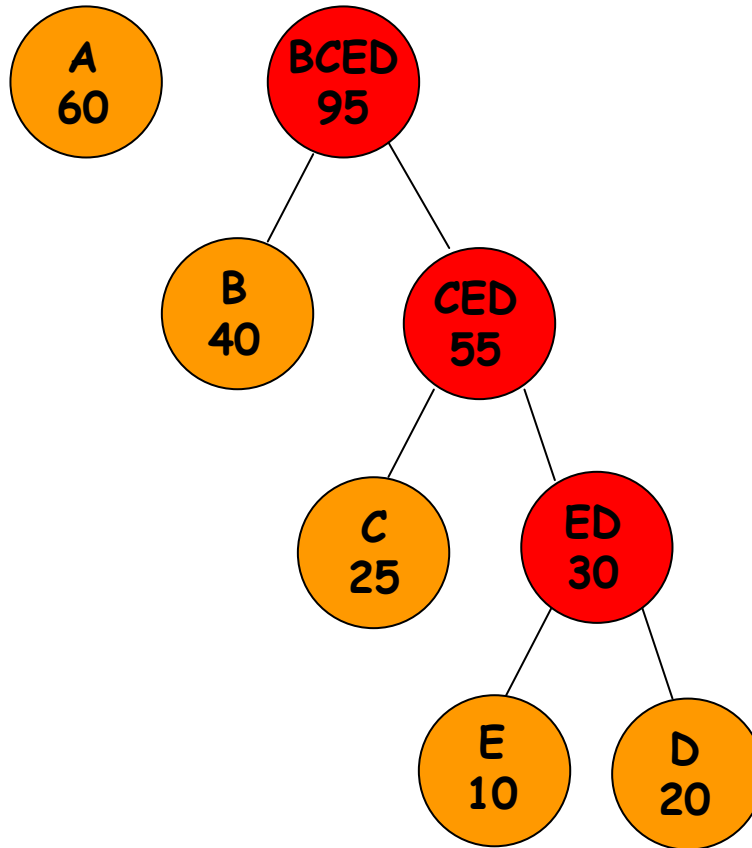


Adım 3

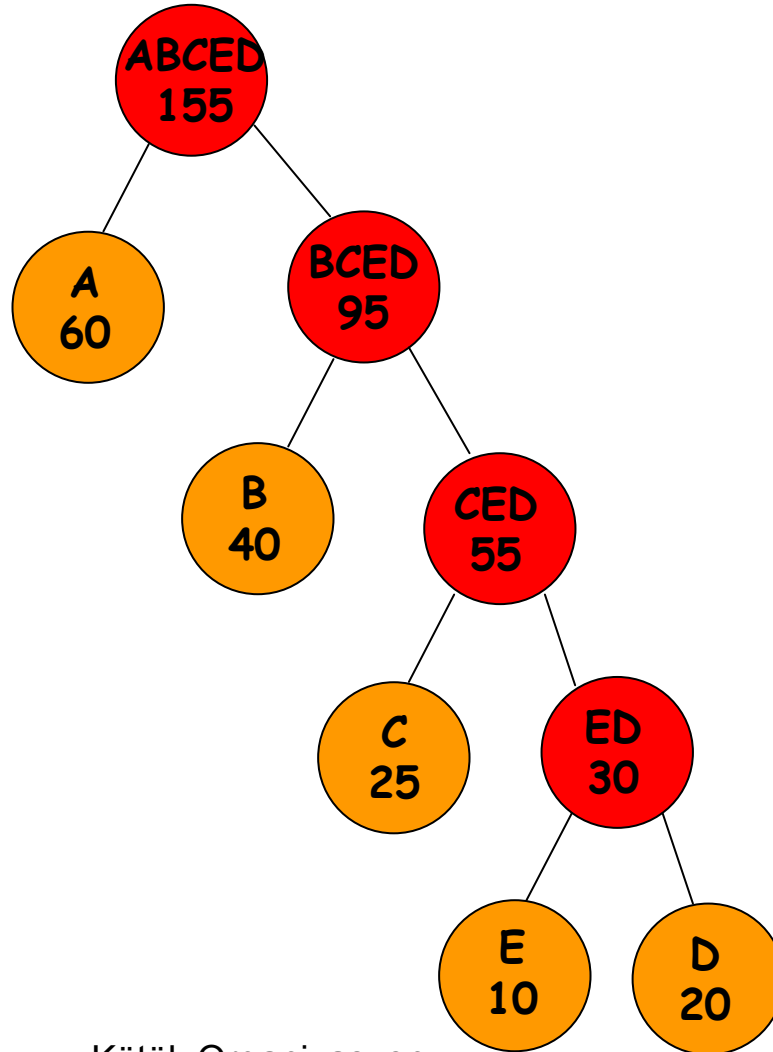
✓ Bir önceki adımdaki işlem terkarlanılarak en küçük frekanslı 2 düğüm tekrar toplanır ve yeni bir düğüm oluşturulur.



Adım 4



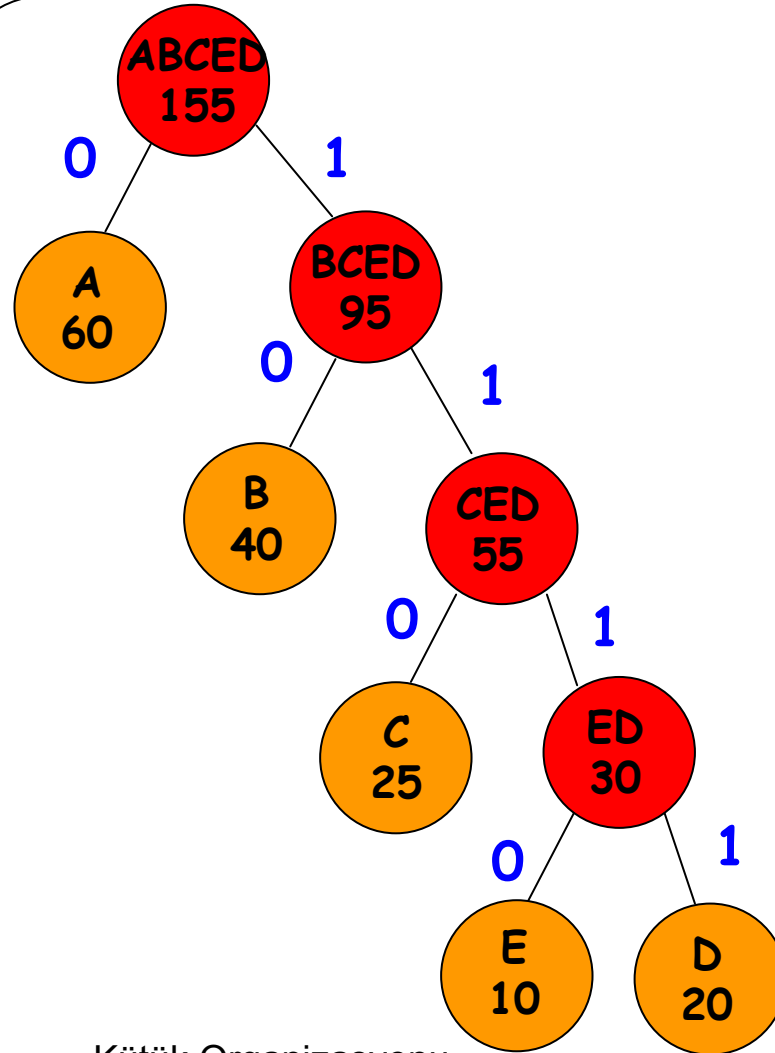
Adım 5



Adım 6

Huffman ağacının kodu oluşturulurken ağacın en tepesindeki kök düğümden başlanır.

Kök düğümün soluna ve sağına giden dallara sırasıyla **0** ve **1** yazılır. Sırası seçimlidir.



Kütük Organizasyonu

Veri Setinin Kodlanmış Hali

Sembol	Huffman Kodu	Bit Sayısı	Frekans
A	0	1	60
B	10	2	40
C	110	3	25
D	1111	4	20
E	1110	4	10

Karşılaştırma

Veri setini ASCII kodu ile kodlanırsa, her karakter için 1 byte(8 bit)'lık alan gerektiğinden toplamda 155 byte(1240 bit)'a ihtiyaç olacaktı.

Huffman kodlamsı ile bu sayı :

$$60 \times 1 + 40 \times 2 + 25 \times 3 + 20 \times 4 + 10 \times 4 = 335 \text{ bittir.}$$

Görüldüğü gibi Huffman kodlamsı ile %27'lik bir kazanç sağlanmaktadır. Bu kazanç, veri setindeki sembollerin frekansları arttıkça daha da artmaktadır.

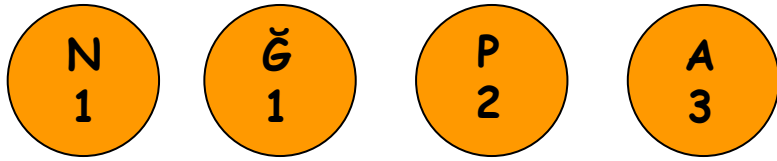
Örnek 2

Üzerinde Huffman kodlamasını gerçekleştireceğimiz kelime “PAPAĞAN” olsun.

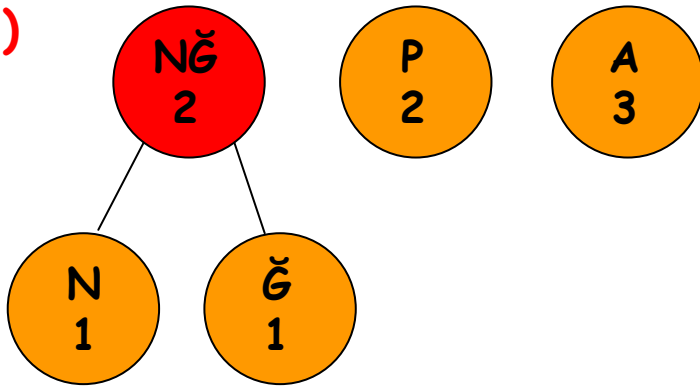
P	A	P	A	Ğ	A	N
---	---	---	---	---	---	---

Sembol	Frekans
P	2
A	3
Ğ	1
N	1

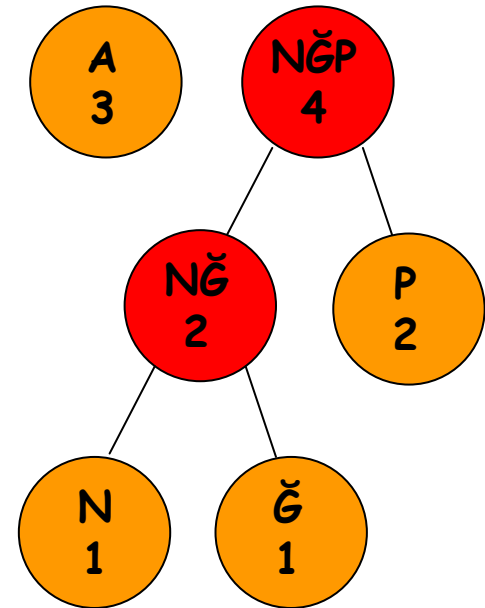
i)



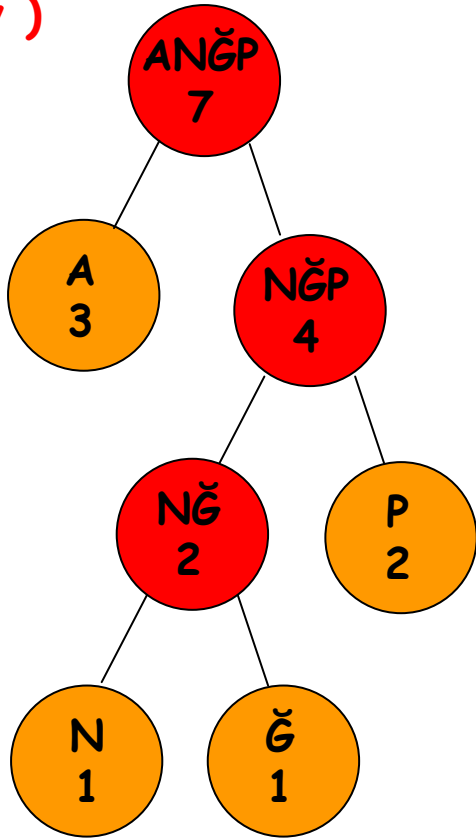
ii)



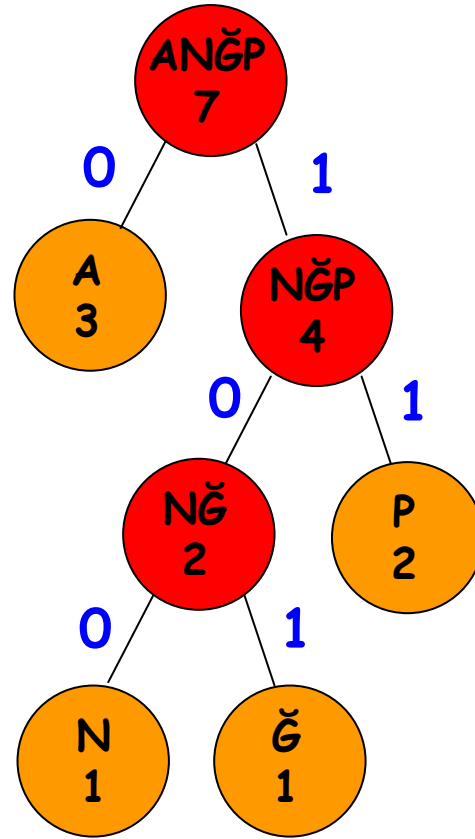
iii)



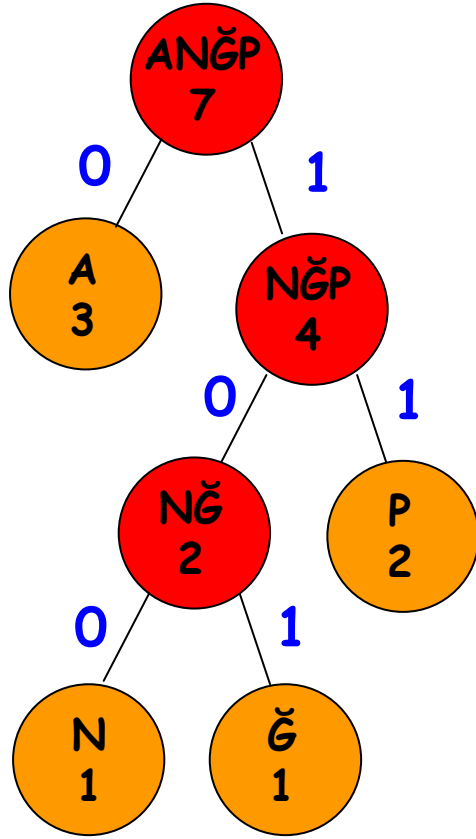
iv)



v)



Huffman Ağacı



Huffman Ağacı

Sembol	Huffman Kodu	Bit Sayısı	Frekans
P	11	2	2
A	0	1	3
Ğ	101	3	1
N	100	3	1

PAPAĞAN sözcüğünü Huffman kodu ile ifade edersek gerekli olan bit sayısı= 13 bit yeterlidir. (Ascii ile 7 byte). Huffman kodu ile kodlanmış sözcük aşağıdaki gibidir.

11 0 11 0 101 0 100

P A P A Ğ A N

Huffman Kodunun Çözümü

- ✓ Frekans tablosu ve sıkıştırılmış veri mevcutsa bahsedilen işlemlerin tersini yaparak kod çözülür. Diğer bir deyişle:
- ✓ Sıkıştırılmış verinin ilk biti alınır. Eğer alınan bit bir kod sözcüğüne karşılık geliyorsa, ilgili kod sözcüğüne denk düşen karakter yerine konulur.
- ✓ Eğer alınan bit bir kod sözcüğü değil ise sonraki bit ile birlikte alınır ve yeni dizinin bir kod sözcüğü olup olmadığına bakılır. Bu işlem dizinin sonuna kadar yapılır. Böylece huffman kodu çözülerek karakter dizisi elde edilmiş olur.

Lempel-Ziv Kodlaması

- Lempel-Ziv kodlamasının birçok farklı türü bulunmaktadır.
- Bu kısımda LZ78'i inceleyeceğiz.
- Unix ortamındaki zip-unzip komutları ile compress-uncompress komutları bu kodlamayı kullanmaktadır.

Örnek 1

- 2 harften meydana gelen bir alfabeye olsun.

aaababbbaaabaabbaabbaabb



Kural

Karakter seti bizim daha önceden hiç görmediğimiz en küçük parçacıklara ayrılır.

a|aa|b|ab|bb|aaa|ba|aaaa|aab|aabb

1. a harfi
2. a harfini daha önceden gördük şimdiki harf aa
3. b harfi
4. a harfini daha önceden gördük, şimdiki harf ab
5. b harfini daha önceden gördük, şimdiki harf bb
6. aa harfini daha önceden gördük, şimdiki harf aaa
7. b harfini daha önceden gördük, şimdiki harf ba
8. aaa harfini daha önceden gördük, şimdiki harf aaaa
9. aa harfini daha önceden gördük, şimdiki harf aab
10. aab harfini daha önceden gördük, şimdiki harf aabb

- 1'den n'e kadar indexlerimiz olsun.

0 1 2 3 4 5 6 7 8 9 10
0|a|aa|b|ab|bb|aaa|ba|aaaa|aab|aabb

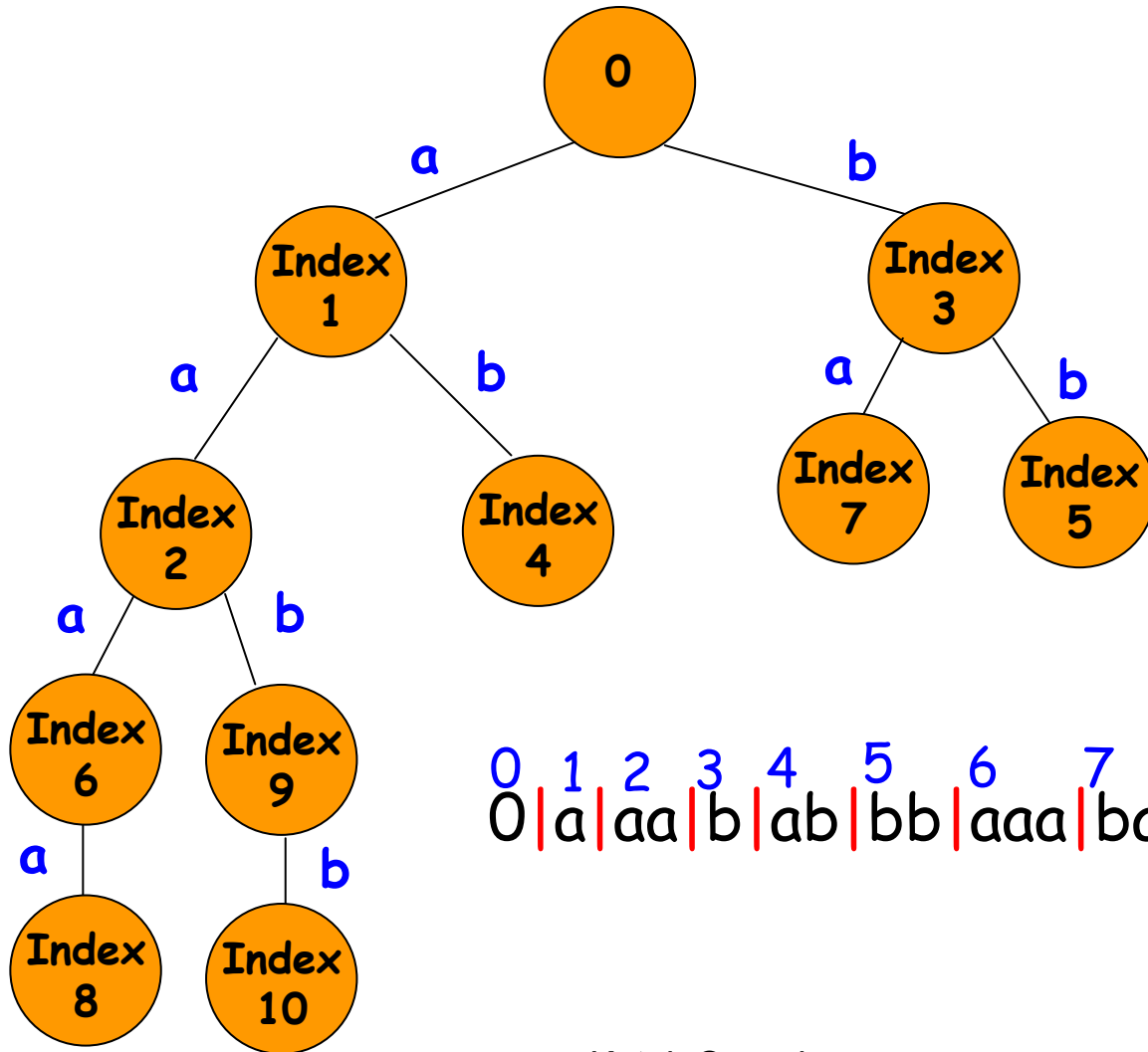
0=Null String

- Bu index yapısını kullanarak veri setini kodlayabiliriz.

1 2 3 4 5 6 7 8 9 10
|0a|1a|0b|1b|3b|2a|3a|6a|2b|9b

Bu yapıda her bir parça yeni bir karakter olara algılandığından ileti, bir önceki index'in üzerine yeni bir karakter ilave edilmesi ile kodlanır.

LZ78'in Ağaç Yapısı



0 1 2 3 4 5 6 7 8 9 10
0 | a | aa | b | ab | bb | aaa | ba | aaaa | aab | aabb

Örnek 2

Aşağıdaki karakter setini LZ78 algoritmasını kullanarak kodlayalım.

aaabbcbcdddeab

i) Karakter setini parçalara ayıralım.

a|aa|b|bc|bcd|d|de|ab

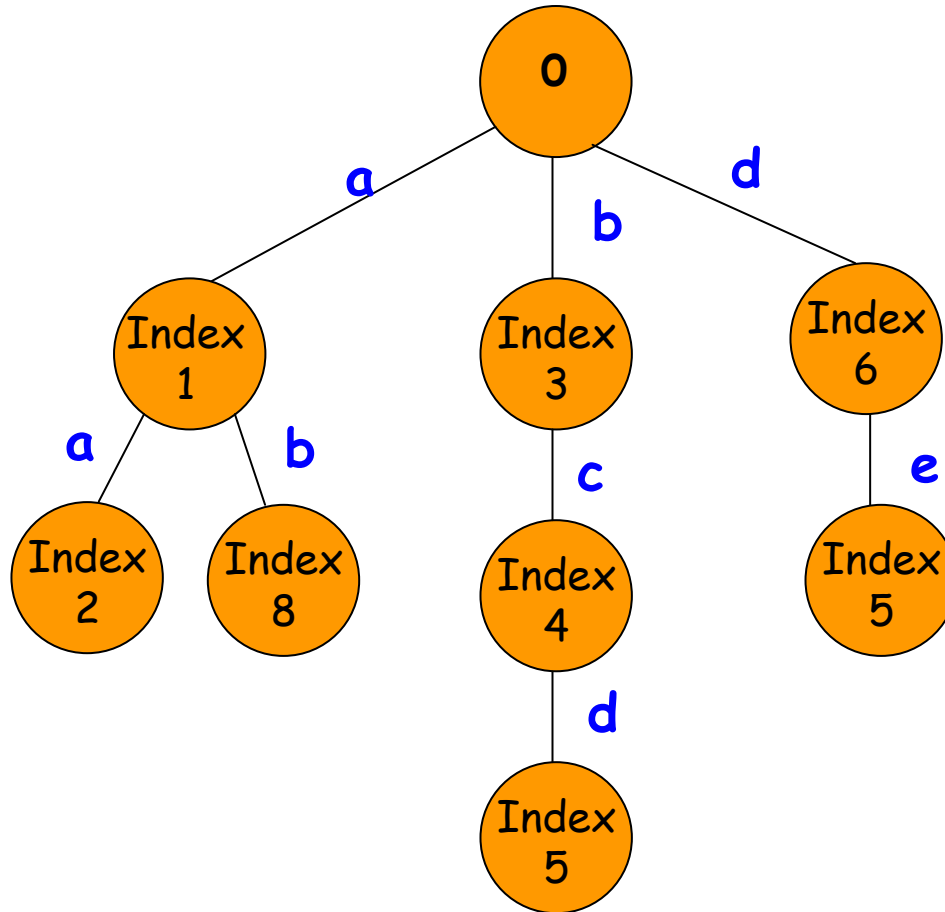
ii) Index oluşturalım.

0 1 2 3 4 5 6 7 8
0|a|aa|b|bc|bcd|d|de|ab

iii) Bu indexi kullanarak veri setini kodlayalım.

|0a|1a|0b|3c|4d|0d|6e|1b

iv) Kodlanan veri setinin ağacını oluşturalım



Örnek 3

Üzerinde LZ78 kodlamasını gerçekleştireceğimiz kelime “PAPAĞAN” olsun.

P	A	P	A	Ğ	A	N
---	---	---	---	---	---	---

i) Karakter setini parçalara ayıralım.

p|a|pa|ğ|an

ii) Index oluşturalım.

0 1 2 3 4 5
0|p|a|pa|ğ|an

iii) Bu indexi kullanarak veri setini kodlayalım.

|0p|0a|1a|0ğ|2n

iv) Kodlanan veri setinin ağacını oluşturalım

