

# 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 sıkıştırmada en iyi adaylardır.
- Daha yoğun 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.

# 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.

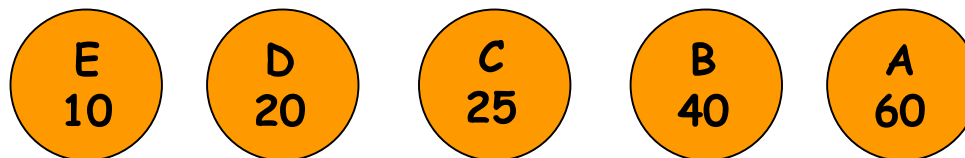
Sembol	Frekans
A	60
B	40
C	25
D	20
E	10



# Adım 1



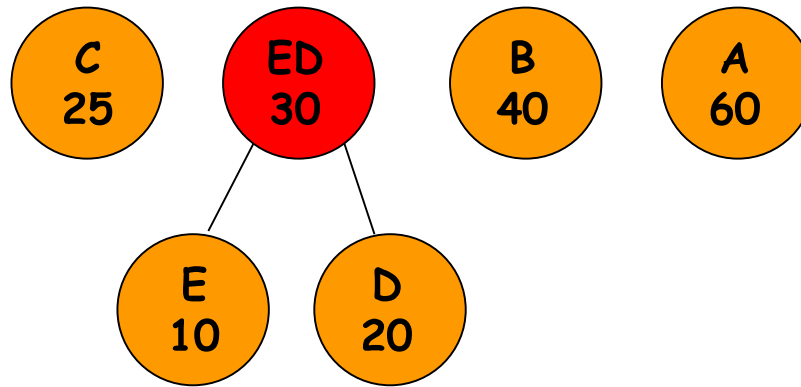
Huffman ağacındaki en son düğümleri oluşturacak olan bütün semboller frekanslarına göre küçükten büyüğe doğru sıralanırlar.



## Adım 2



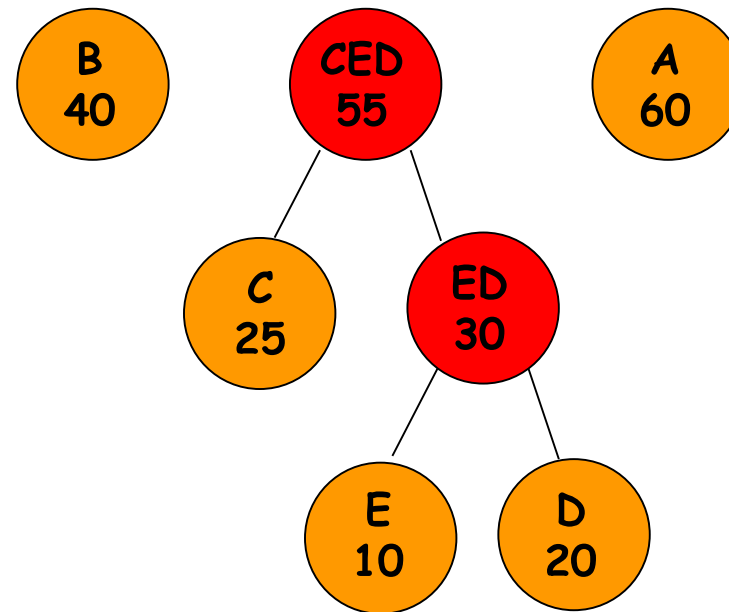
En küçük frekansa sahip olan iki sembolün frekansları toplanarak yeni bir düğüm oluşturulur. Oluşturulan bu yeni düğüm var olan düğümler arasında uygun yere yerleştirilir. Bu yerleştirme frekans bakımından küçüklük veya büyüklüğe göredir.



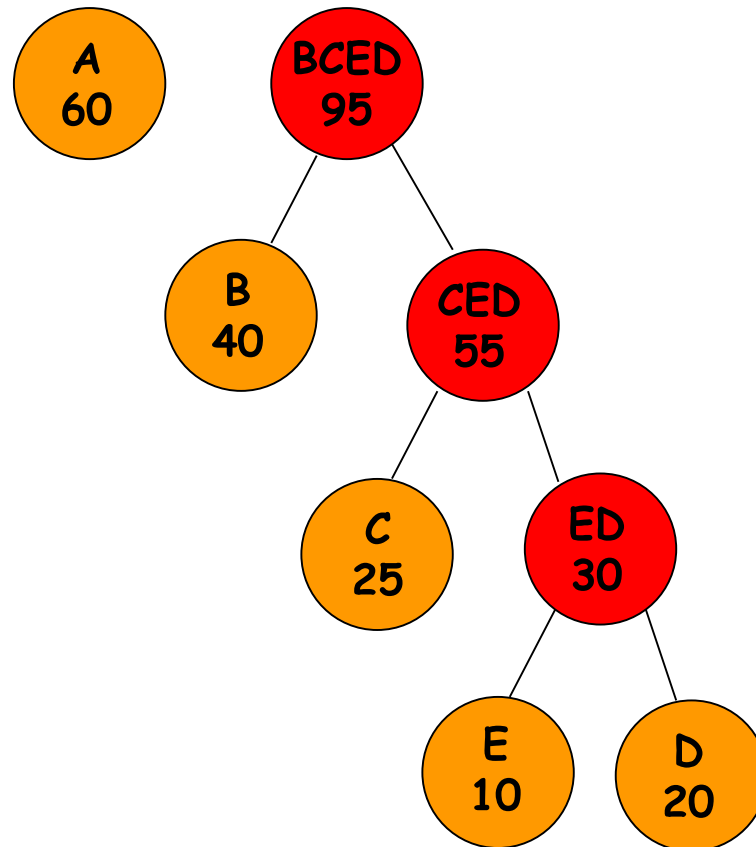
## Adım 3



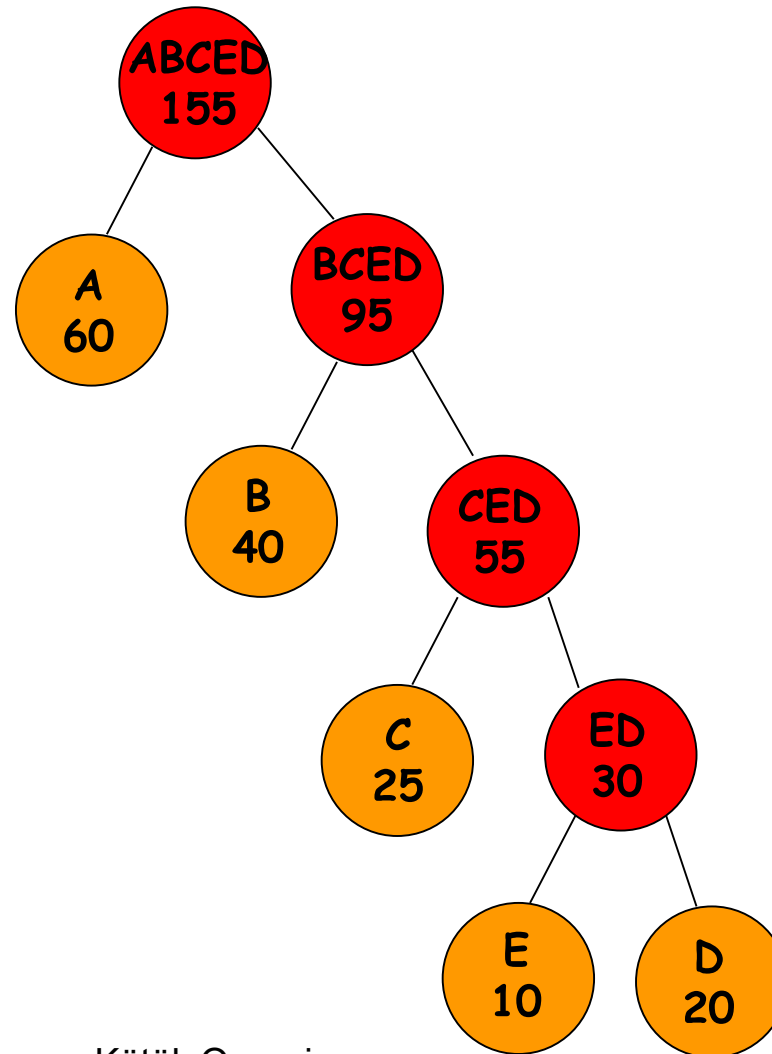
Bir önceki adımdaki işlem terkarlanılarak en küçük frekanslı 2 düğüm tekrar toplanır ve yeni bir düğüm oluşturulur.



## Adım 4



## Adım 5

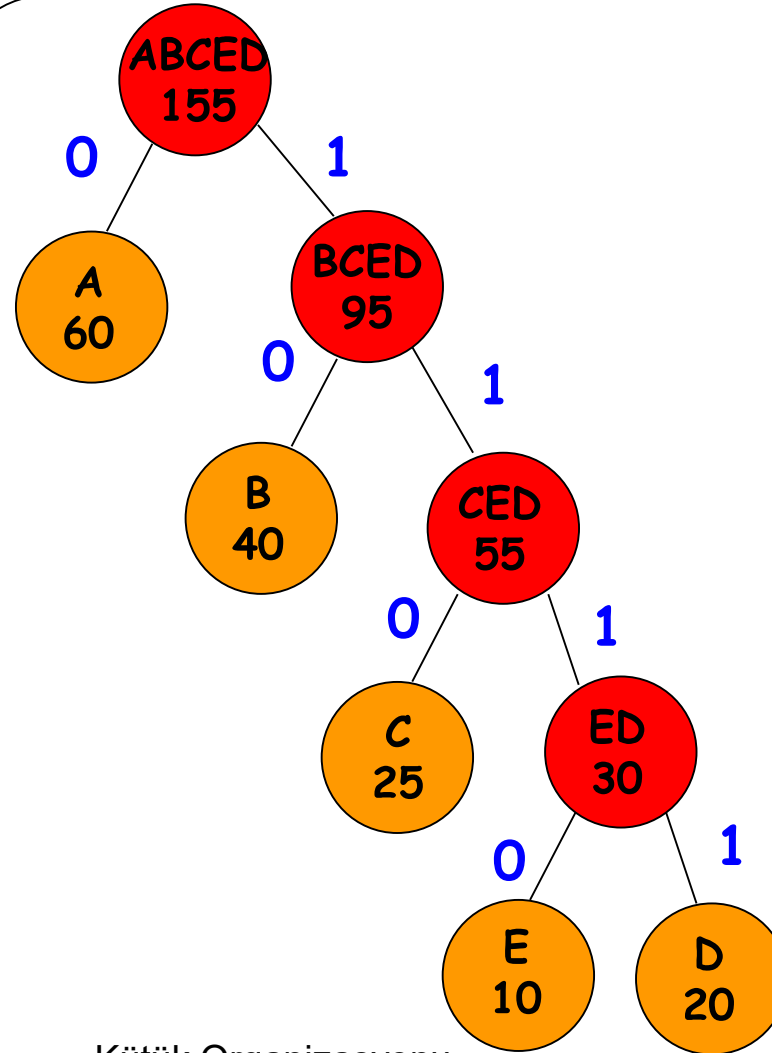


Kütük Organizasyonu

## 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)'lik 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 %73'lik bir kazanç sağlanmaktadır. Bu kazanç, veri setindeki sembollerin frekansları arttıkça daha da artmaktadır.



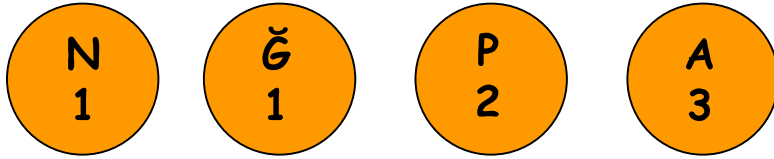
## Örnek 2

Üzerinde Huffman kodlamasını gerçekleştireceğimiz kelime “PAPAĞAN” olsun.

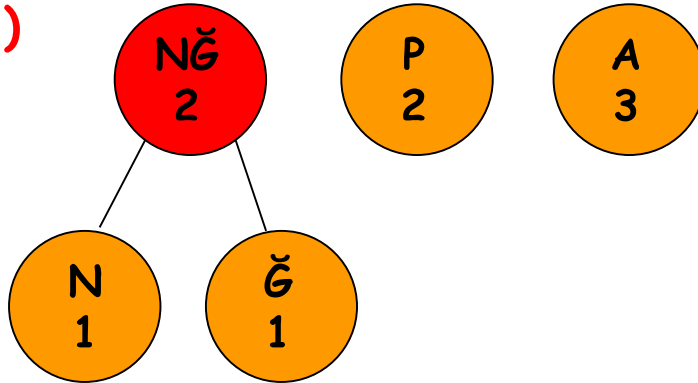
P	A	P	A	Ğ	A	N
---	---	---	---	---	---	---

Sembol	Frekans
P	2
A	3
Ğ	1
N	1

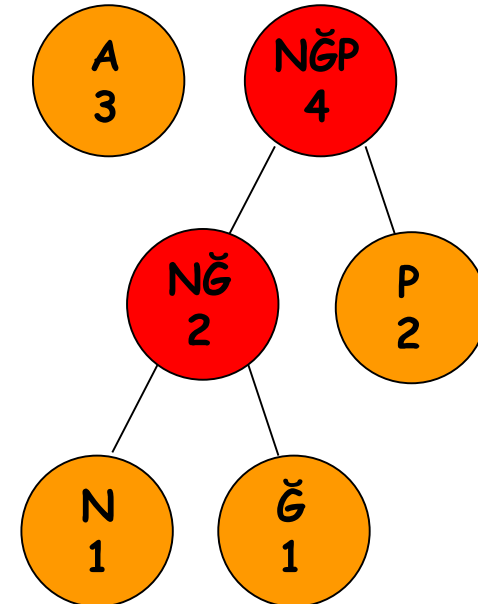
i )



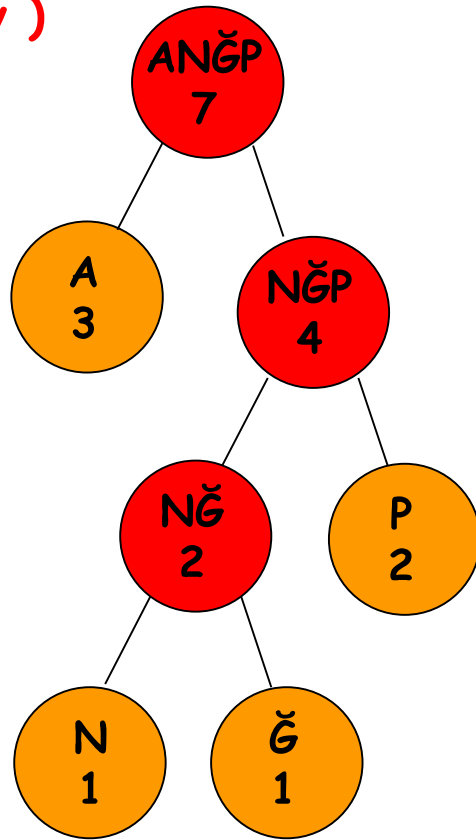
ii )



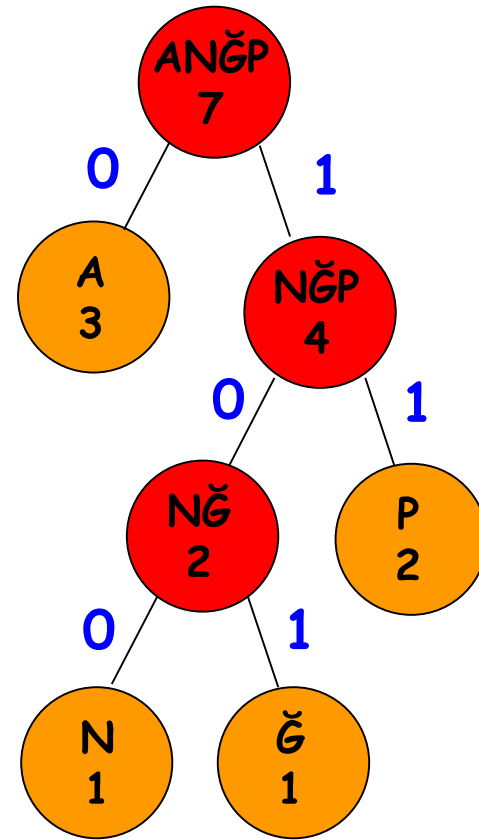
iii )



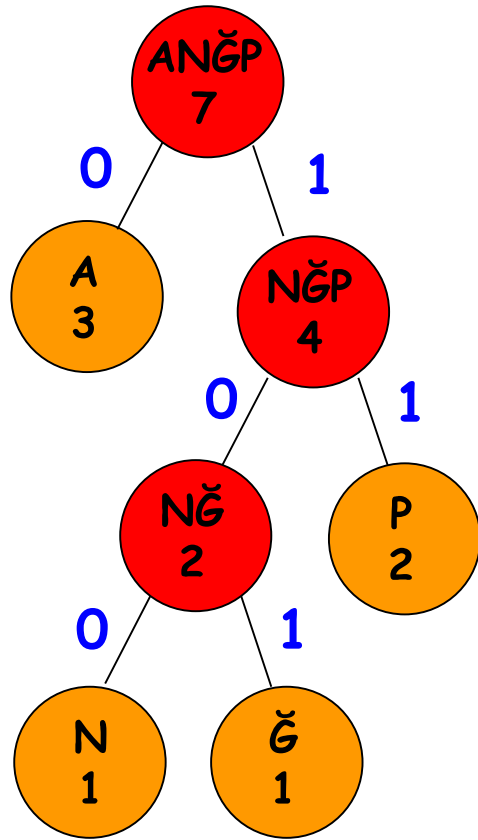
iv )



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

- 1970'lerin sonlarına kadar veri sıkıştırma çalışmaları Huffman kodlamasından daha iyi algoritmalar geliştirmek için çalıştı.
- Bu alanda yeni bir yaklaşım Abraham Lempel ve Jacob Ziv tarafından gerçekleştirilmiştir.
- Lempel ve Ziv kendi adlarıyla bilinen LZ77 ve LZ78 algoritmalarını geliştirmişlerdir. Bu iki algoritmanın bir çok farklı türü bulunmaktadır.

LZ77 Altkategorileri	LZR	LZSS	LZB	LZH		
LZ78 Altkategorileri	LZ W	LZC	LZT	LZMW	LZJ	LZFG

- zip ve unzip LZH tekniğini kullanırken, UNIX'de yer alan compress komutu LZW ve LZC'yi kullanmaktadır.

- Algoritmada veri yapısı olarak genellikle sözlük (dictionary) kullanır.
- LZ78, veri setindeki bir veya daha fazla karakteri çakışmayacak(non-overlapping) ve aynı zamanda eşsiz(unique) desenler şeklinde sözlüğe ekler.
- LZ78'in sözlük yapısının çalışması ise;

(0, char)	Eğer karakter sözlükte mevcut değilse ekle
(DictionaryPrefixIndex, char)	Eğer karakter sözlükte mevcut değilse ekle
(DictionaryPrefixIndex, next char)	Eğer karakter sözlükte mevcut değilse ekle



```

Dictionary ← empty ; Prefix ← empty ; DictionaryIndex ← 1;
while(characterStream is not empty)
{
    Char ← next character in characterStream;
    if(Prefix + Char exists in the Dictionary)
        Prefix ← Prefix + Char ;
    else
    {
        if(Prefix is empty)
            CodeWordForPrefix ← 0 ;
        else
            CodeWordForPrefix ← DictionaryIndex for Prefix ;
        Output: (CodeWordForPrefix, Char) ;
        insertInDictionary( ( DictionaryIndex , Prefix + Char) );
        DictionaryIndex++;
        Prefix ← empty ;
    }
}
if(Prefix is not empty)
{
    CodeWordForPrefix ← DictionaryIndex for Prefix;
    Output: (CodeWordForPrefix , ) ;
}

```

# Örnek 1

- 2 harften meydana gelen bir alfebe olsun.

aaababbbbaaabaabbaabbaabb



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

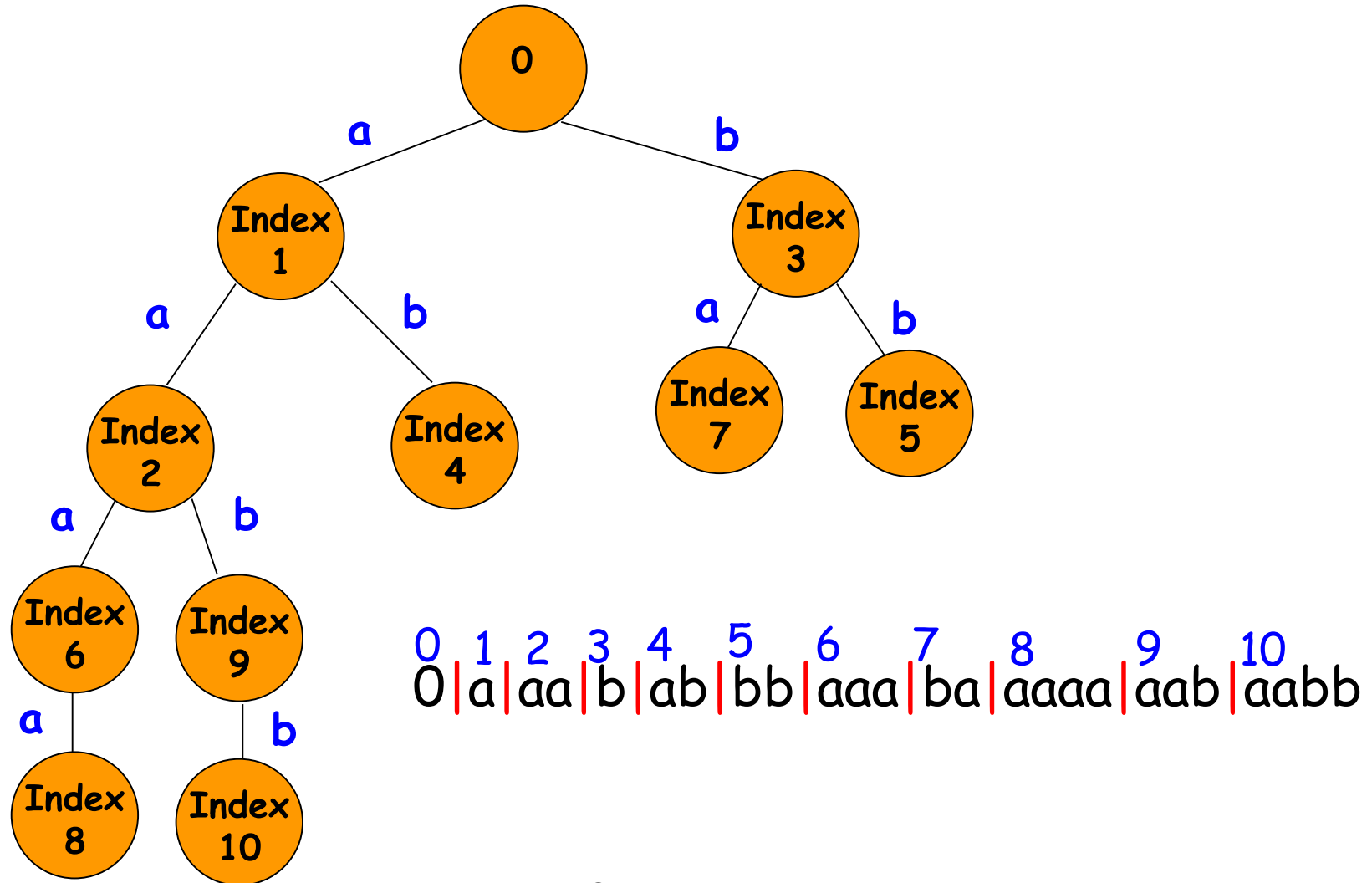
0=Null String

- 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



## Örnek 2

Aşağıdaki karakter setini LZ78 algoritmasını kullanarak kodlayalım.

aaabbcbcdddeab

i ) Karakter setini parçalara ayıralım.

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

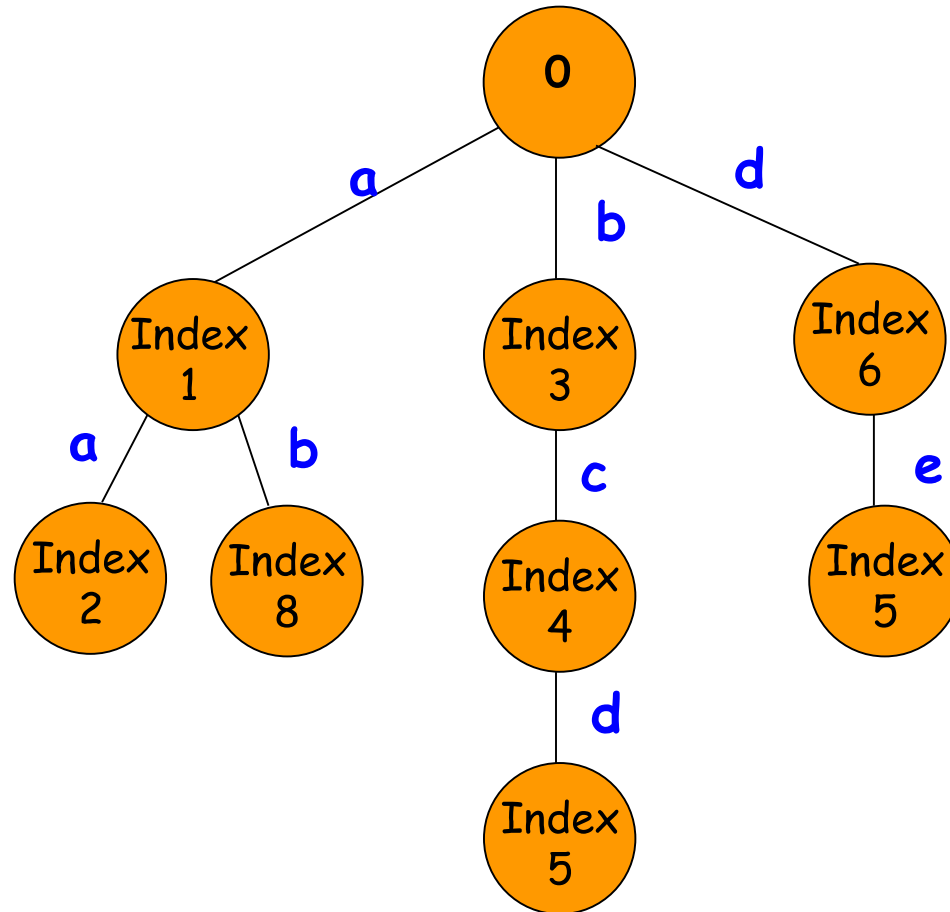
ii ) Index oluşturalım.

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

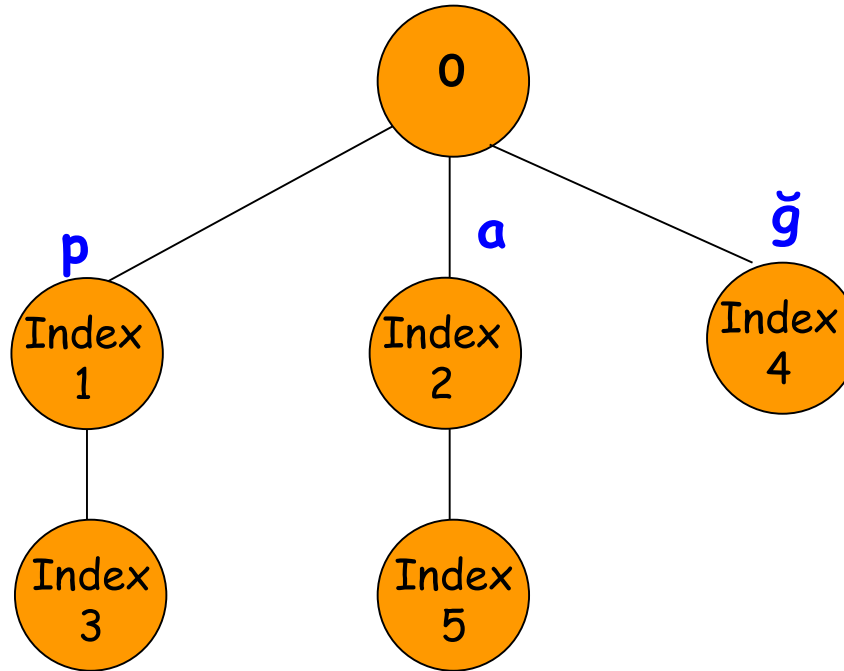
ii ) Index oluşturalım.

0 1 2 3 4 5  
0|p|a|pa|ğ|an

iii ) Bu indexi kullanarak veri setini kodlayalım.

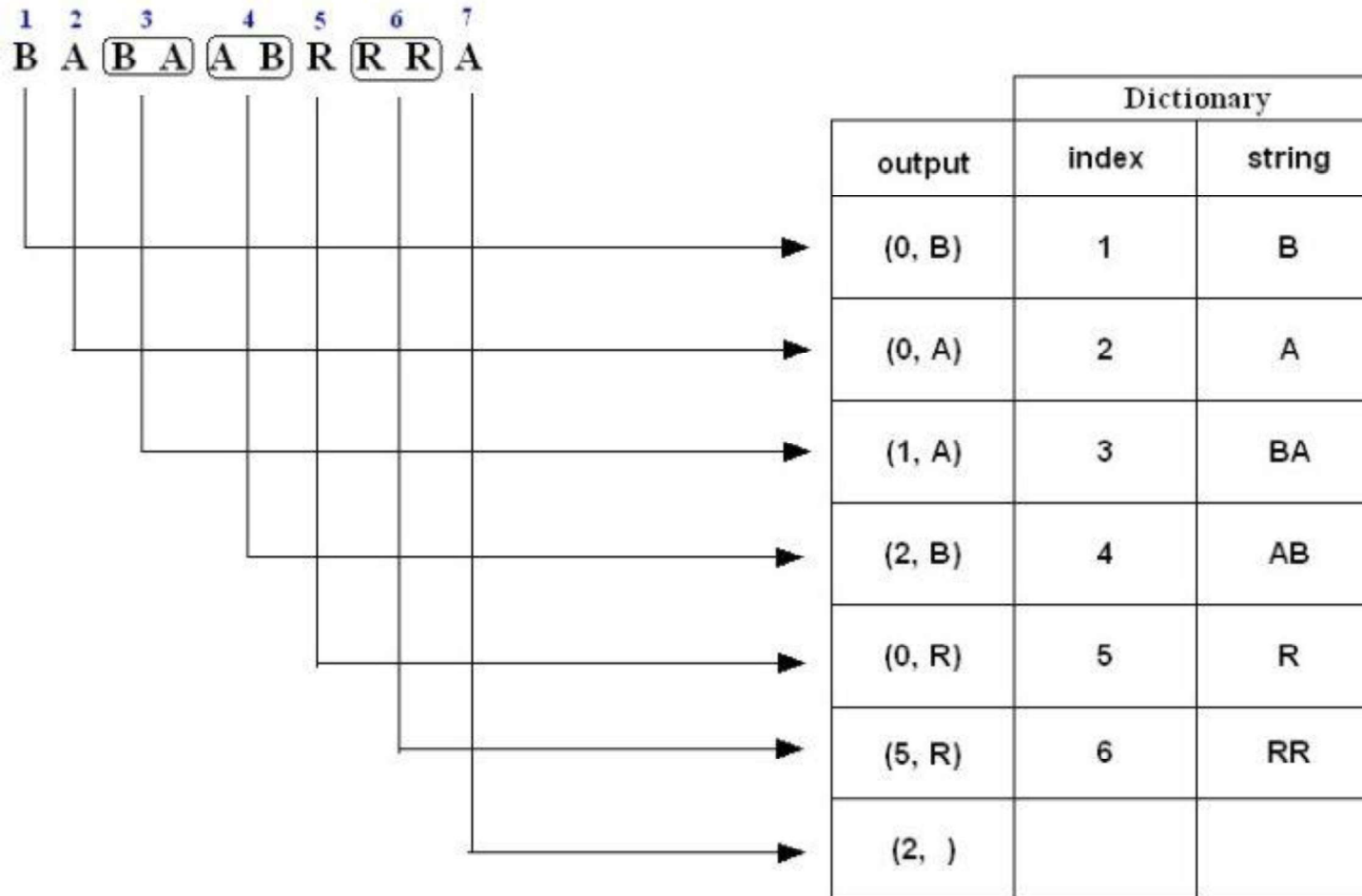
|0p|0a|1a|0ğ|2n

iv ) Kodlanan veri setinin ağacını oluşturalım



## Örnek 4

- “BABAABRRRA” şeklindeki stringi LZ78 ile kodlayalım

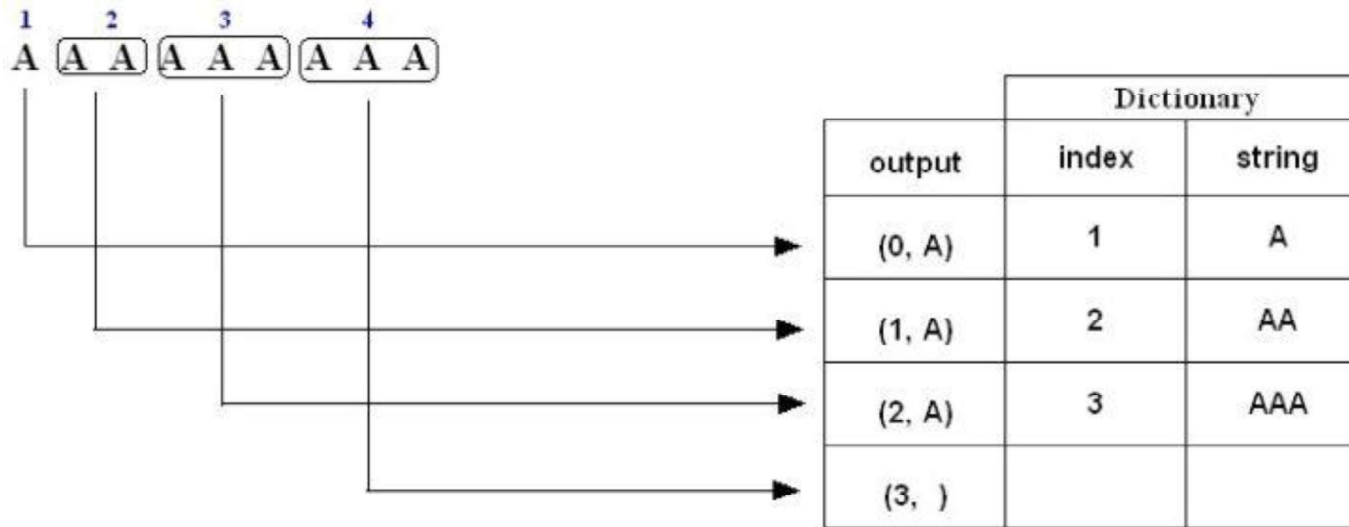


- Sıkıştırılan string : (0,B)(0,A)(1,A)(2,B)(0,R)(5,R)(2, )

1. **B** is not in the Dictionary; insert it
2. **A** is not in the Dictionary; insert it
3. **B** is in the Dictionary.  
    **BA** is not in the Dictionary; insert it.
4. **A** is in the Dictionary.  
    **AB** is not in the Dictionary; insert it.
5. **R** is not in the Dictionary; insert it.
6. **R** is in the Dictionary.  
    **RR** is not in the Dictionary; insert it.
7. **A** is in the Dictionary and it is the last input character; output a pair containing its index: **(2, )**

## Örnek 5

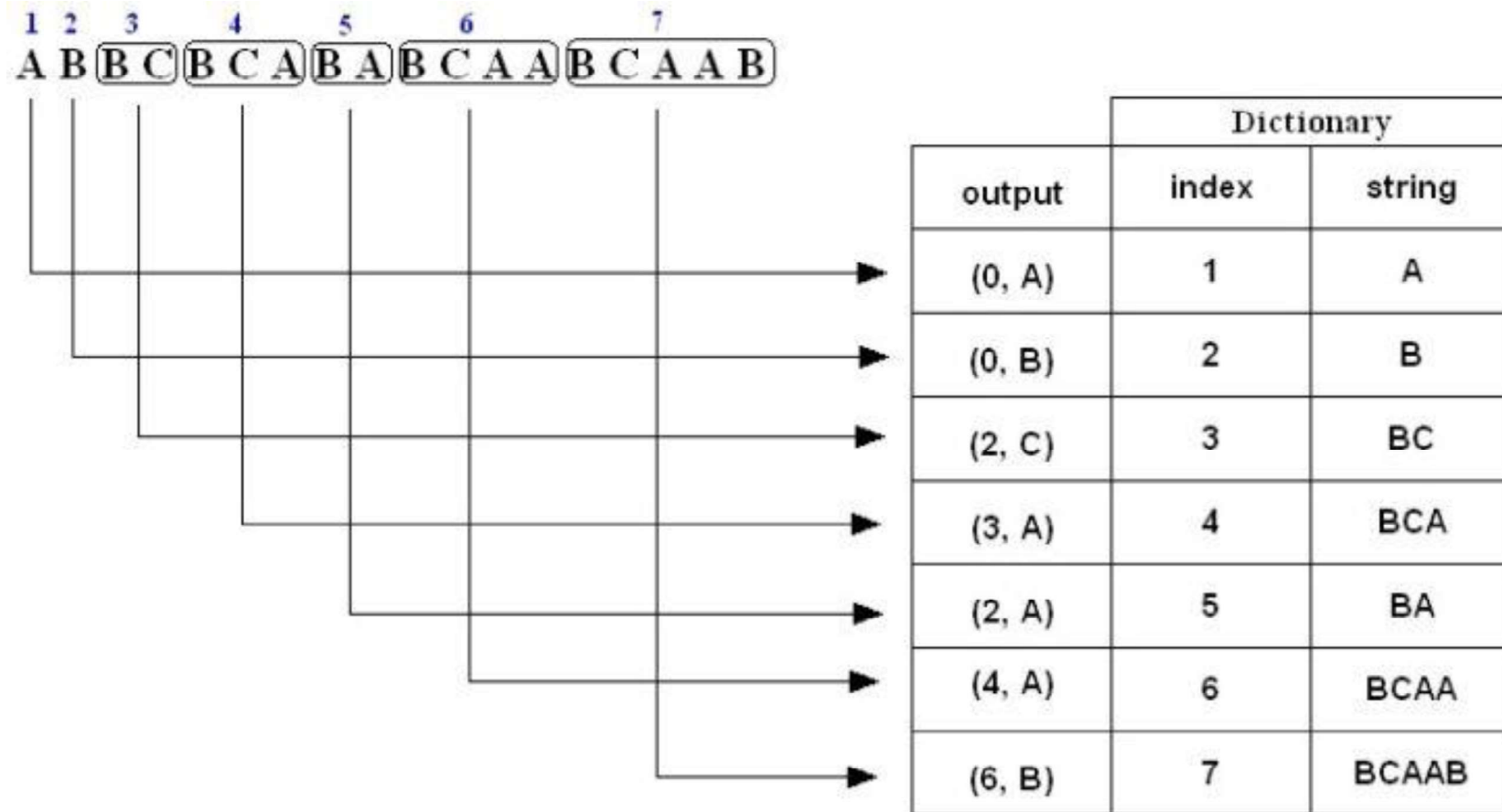
- “AAAAAAAAA” şeklindeki stringi LZ78 ile kodlayalım



1. A is not in the Dictionary; insert it
2. A is in the Dictionary  
AA is not in the Dictionary; insert it
3. A is in the Dictionary.  
AA is in the Dictionary.  
AAA is not in the Dictionary; insert it.
4. A is in the Dictionary.  
AA is in the Dictionary.  
AAA is in the Dictionary and it is the last pattern; output a pair containing its index:  
(3, )

## Örnek 6

- “ABBCBCABABCAABCAAB” şeklindeki stringi LZ78 ile kodlayalım



- Sıkıştırılan string : (0,A)(0,B)(2,C)(3,A)(2,A)(4,A)(6,B)

1. **A** is not in the Dictionary; insert it
2. **B** is not in the Dictionary; insert it
3. **B** is in the Dictionary.  
    **BC** is not in the Dictionary; insert it.
4. **B** is in the Dictionary.  
    **BC** is in the Dictionary.  
    **BCA** is not in the Dictionary; insert it.
5. **B** is in the Dictionary.  
    **BA** is not in the Dictionary; insert it.
6. **B** is in the Dictionary.  
    **BC** is in the Dictionary.  
    **BCA** is in the Dictionary.  
    **BCAA** is not in the Dictionary; insert it.
7. **B** is in the Dictionary.  
    **BC** is in the Dictionary.  
    **BCA** is in the Dictionary.  
    **BCAA** is in the Dictionary.  
    **BCAAB** is not in the Dictionary; insert it.



# Transfer Edilen Bit Miktarı

- Sıkıştırılmamış string: **ABBCBCABABCAABCAAB**
  - Toplam miktar :  $18 * 8 = 144$  bit
- Sıkıştırılmış form : **(0,A)(0,B)(2,C)(3,A)(2,A)(4,A)(6,B)**
- Bu form üzerinde her kod kelimesini 1'den başlayarak indeksleisin.
- Codeword index

<b>(0,A)</b>	<b>(0,B)</b>	<b>(2,C)</b>	<b>(3,A)</b>	<b>(2,A)</b>	<b>(4,A)</b>	<b>(6,B)</b>
<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>