Huffman Encoding

by Snowblind

Veri Sıkıştırmaya Giriş

- Elde edilen sonuç bakımından en temelde iki tür sıkıştırma yöntemi kategorisi vardır:
  - Kayıplı Sıkıştırma (Lossy Compression)
  - Kayıpsız Sıkıştırma (Lossless Compression)

Kayıplı Sıkıştırma

- Sıkıştırma sonucu elde edilen veriden önceki verinin aynısını elde etmek mümkün değildir.
- Sıkışan verinin bazı bölümleri “kaybolur”.
- Örnekler:
  - MP3: Ham ses verisinden insan kulağının duyamayacağı frekanslar çıkarılır.
  - JPEG: “Göze batmayacak” bozulmalar yapılır.

Kayıpsız Sıkıştırma

- Sıkıştırma sonucu elde edilen veriden önceki verinin aynısını elde etmek mümkündür.
- Örnekler:
  - ZIP, RAR, vs.: Sıkıştırılan dosyaları açtığınızda orijinal dosyaları elde edersiniz. Hiçbir şey “kaybolmaz”.
  - **Huffman Encoding**
  - **LZ 778**

Niçin Dosya Sıkıştırmaya İhtiyaç Duyulmuştur?

- Minimum alana maksimum veri sığdırmak
- Daha hızlı aktarım sağlamak, erişim süresini azaltmak
- Sıra düzensel verileri daha hızlı işleyebilmek için.

Huffman Kodlamasının Mantığı

- Sıkıştırılacak dosyanın içinde bazı byte değerleri hiç geçmiyor olabilir. Mesela içinde hiç “x” harfi olmayan bir metin dosyası olabilir.
- Belli byte değerleri diğerlerine göre daha sık geçebilir. Mesela “aaaabbaaaab” gibi bir metinde “a”, “b”den daha sık geçmektedir.
- Dosyada **bulunan** ve **sık geçen** byte değerlerini daha kısa bir bit düzeniyle ifade etme mantığıdır.

Kodlama Başlıyor

- Örneğin metin.txt adlı bir dosyayı sıkıştıralım.
- Bu dosyanın içeriği "abacaba" şeklinde olsun.
- Dosyanın boyutu 7 byte, yani (7 x 8 =) 56 bittir.
- a'nın değeri 97, yani bit düzeni 01100001'dir.
- b'nin değeri 98,yani bit düzeni 01100010'dir.
- c'nin değeri 99, yani bit düzeni 01100011'dir.
- O halde metin.txt dosyasının bit düzeni:

01100001	01100010	01100001	01100011	01100001	01100010	01100001
a	b	a	c	a	b	a

1. Adım: Sayım

- Dosyada bulunan her byte değerinin dosyada kaç kere geçtiği tespit edilir.
- [metin.txt]: "abacaba"
- a: 4 adet
- b: 2 adet
- c: 1 adet

2. Adım: Kümeleme

- Elimizdeki byte değeri türlerinden en seyrek geçen iki tanesi seçilir, bu ikisi birleştirilir ve bu birleşik değer diğerlerinin arasına geri konur.
- Bu işlem, tek bir değer çeşidi kalana kadar devam eder.

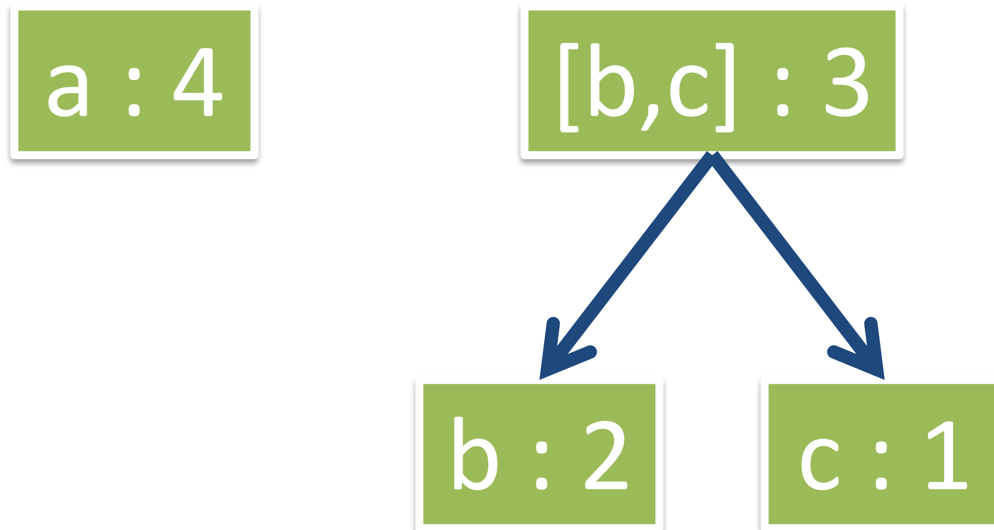
a : 4

b : 2

c : 1

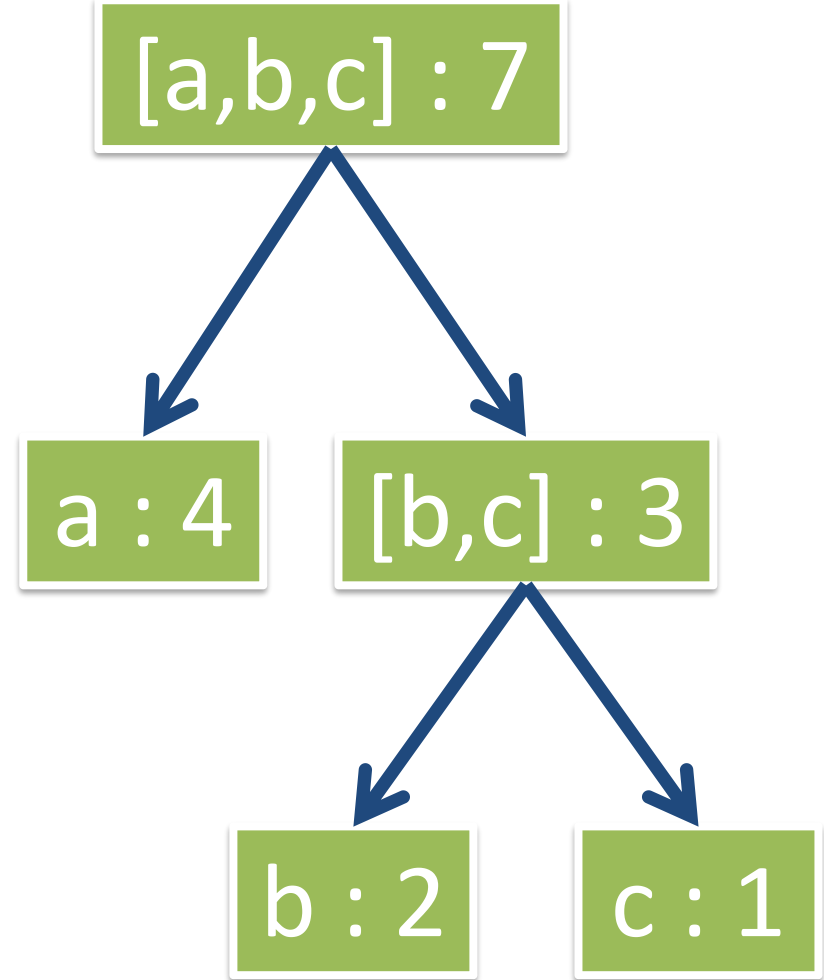
Örnek Kümeleme İşlemi (1. Tur)

- En seyrek geçen iki karakter olan b (2) ve c (1) çekilir ve birleştirilir. Birleşim a’nın yanına geri atılır.



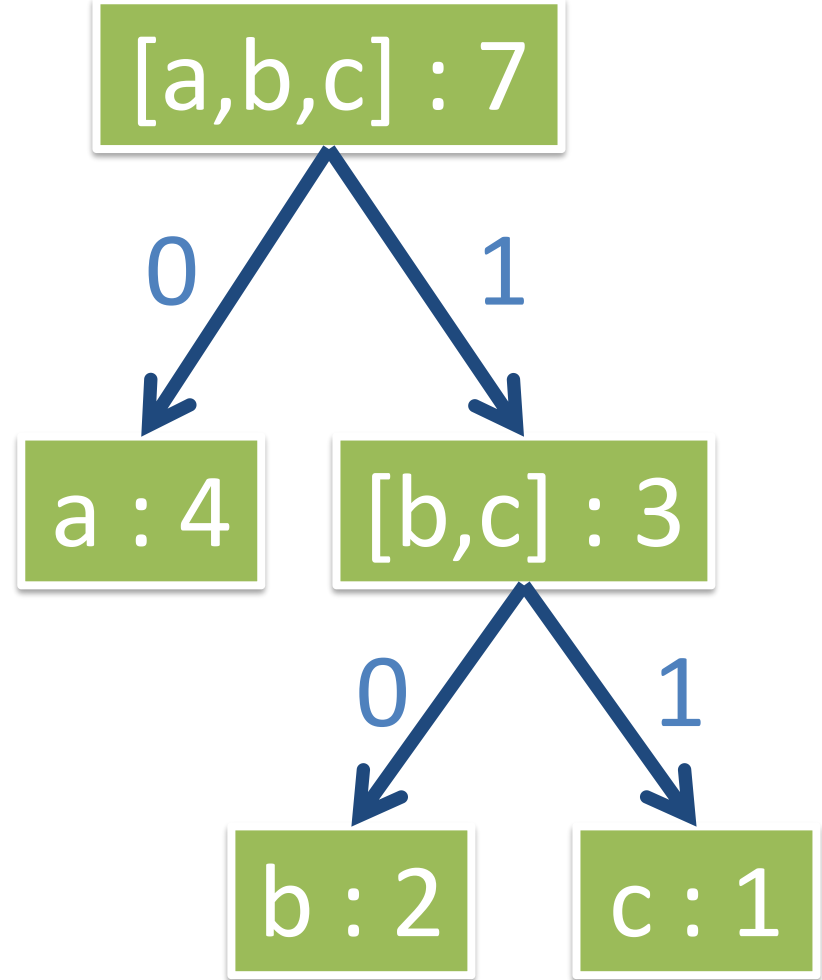
Örnek Kümeleme İşlemi (2. Tur)

- En seyrek geçen iki karakter olan a (4) ve [b,c] (3) çekilir ve birleştirilir. Yeni birleşim geri atılır.
- Bu işlemin sonunda tek bir değer çeşidi ([a,b,c] : 7) kaldığından kümeleme aşaması sonlanır.



3. Adım: Kodlama

- Kümeleme işlemi sonucunda elde edilen ağaç yapısının sola giden oklarına “0”, sağa giden oklarına ise “1” denir.
- Her bir karakterin bit düzeyindeki yeni ifadesi, en tepeden o karaktere giden yoldaki 0 ve 1’lerden oluşur.
- Yani a: 0, b: 10, c: 11



Sıkıştırma İşleminin Sonucu

- Elde edilen yeni bit düzeyindeki ifadelere göre (a: 0, b: 10, c: 11) dosya içeriği tekrar yazılır:

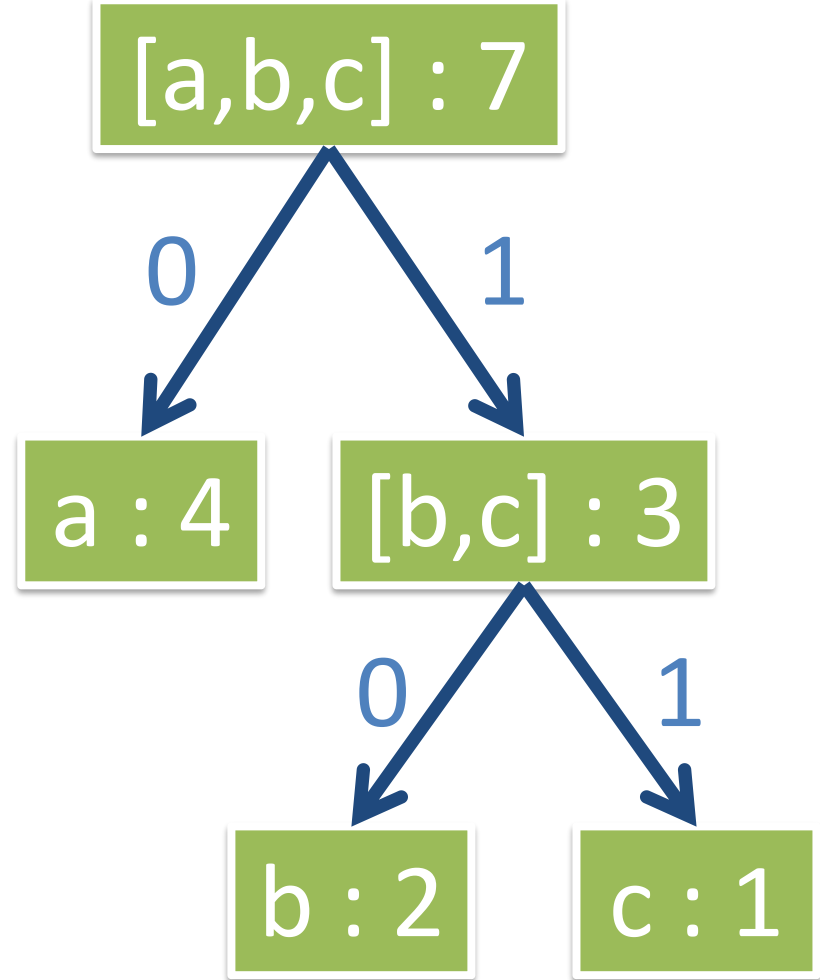
- Dosyamızın ilk içeriği 56 bit (7 byte) iken, sıkıştırıldıktan sonraki boyutu gördüğünüz gibi 10 bit (2 byte) oldu.
- Yaklaşık olarak %71 oranında sıkıştırma gerçekleşti.

Sıkıştırılmış Dosyanın Açılması

- Elimizdeki sıkıştırılmış ve bit düzeyindeki veri:

  0100110100

- Bu veriyi açacak olan anahtar ise yanda verilen ağaç yapısı.
- Tek yapmamız gereken elimizdeki bit verisini ağaç üzerinde takip edip bir sona geldiğimizde vardığımız karakteri basıp tekrar ağacın başına dönmek.

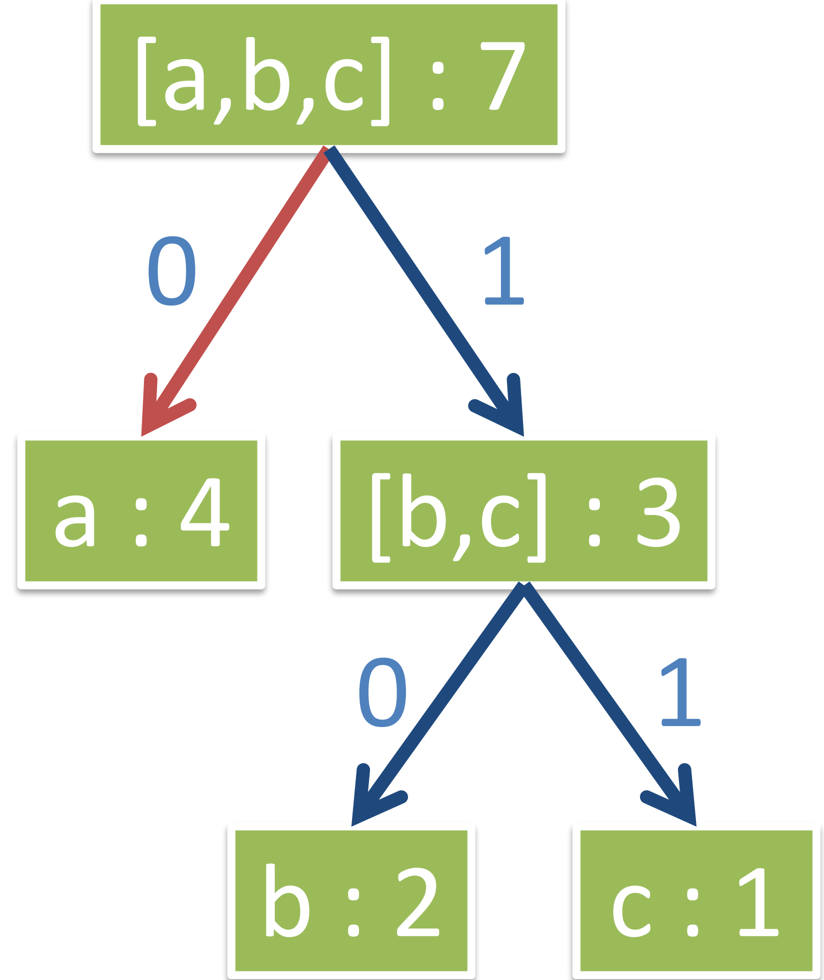


Sıkıştırılmış Dosyanın Açılması

0100110100

a

- İlk bit olan "0"ı okuduk ve ağaç üzerinde "a" karakterine vardık.
- O halde "a" karakterini basıp ağaçta en başa dönüyoruz.

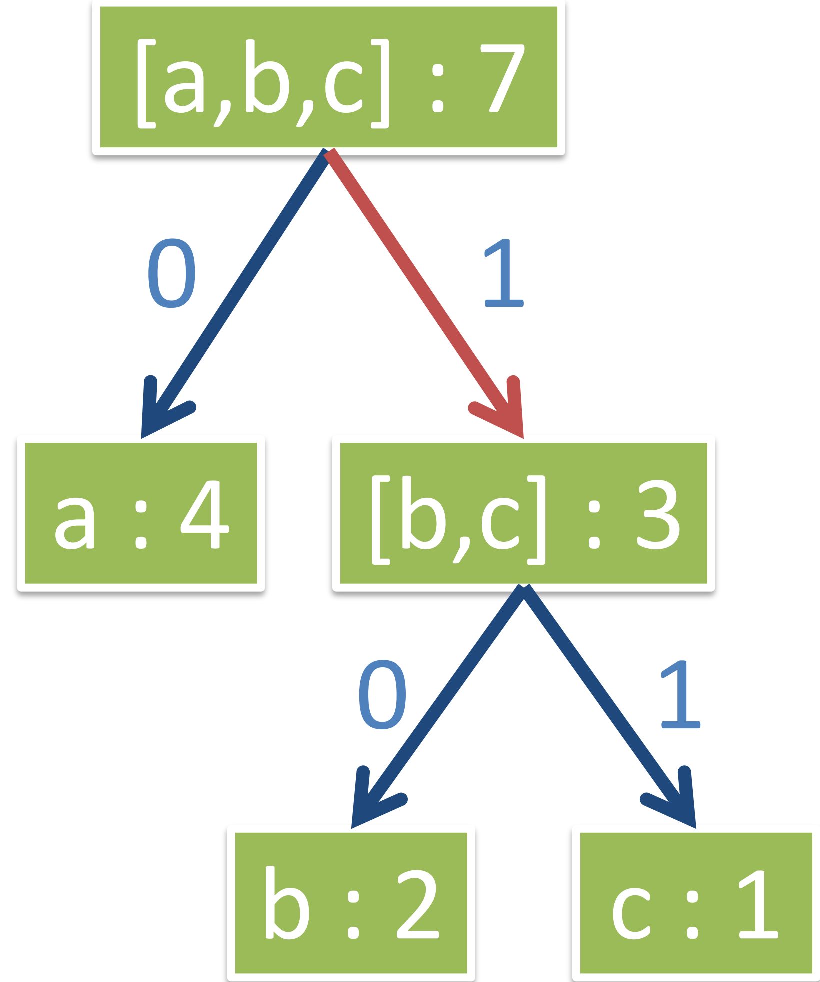


Sıkıştırılmış Dosyanın Açılması

0100110100

a

- Bir sonraki bit olan "1"i okuduk ve ağaç üzerinde herhangi bir sona varamadık.
- O halde bulunduğumuz yerden devam ediyoruz.

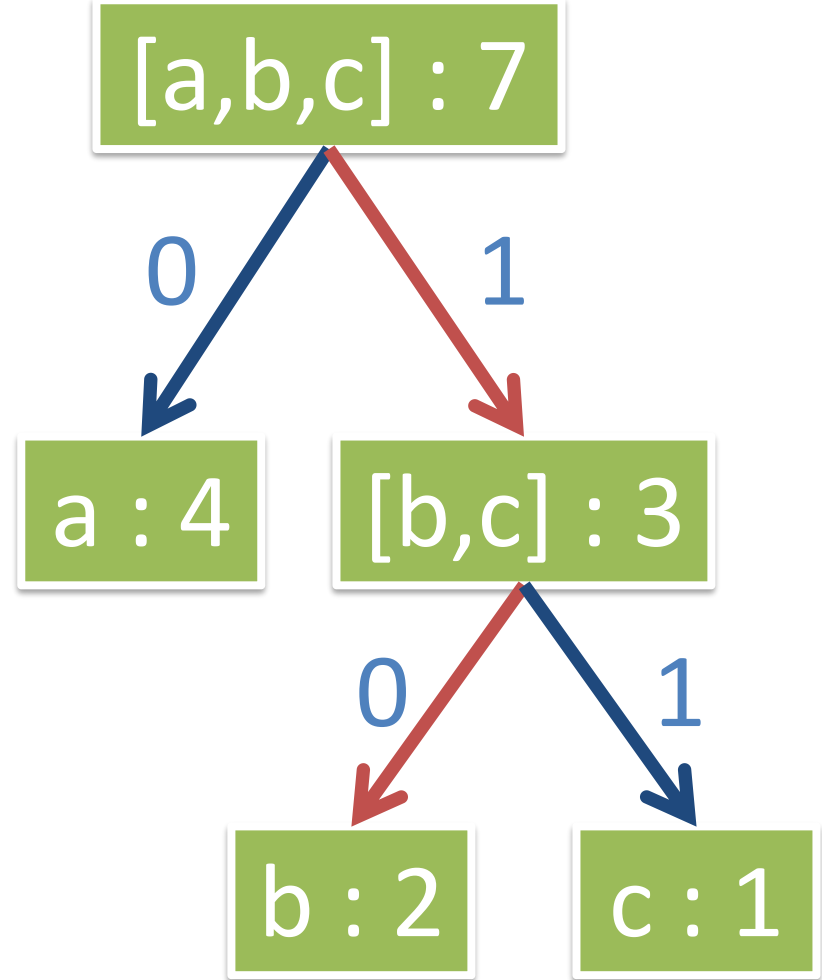


Sıkıştırılmış Dosyanın Açılması

0100110100

ab

- Bir sonraki bit olan “0”ı okuduk ve ağaç üzerinde “b” karakterine vardık.
- O halde “b” karakterini basıp ağaçta en başa dönüyoruz.

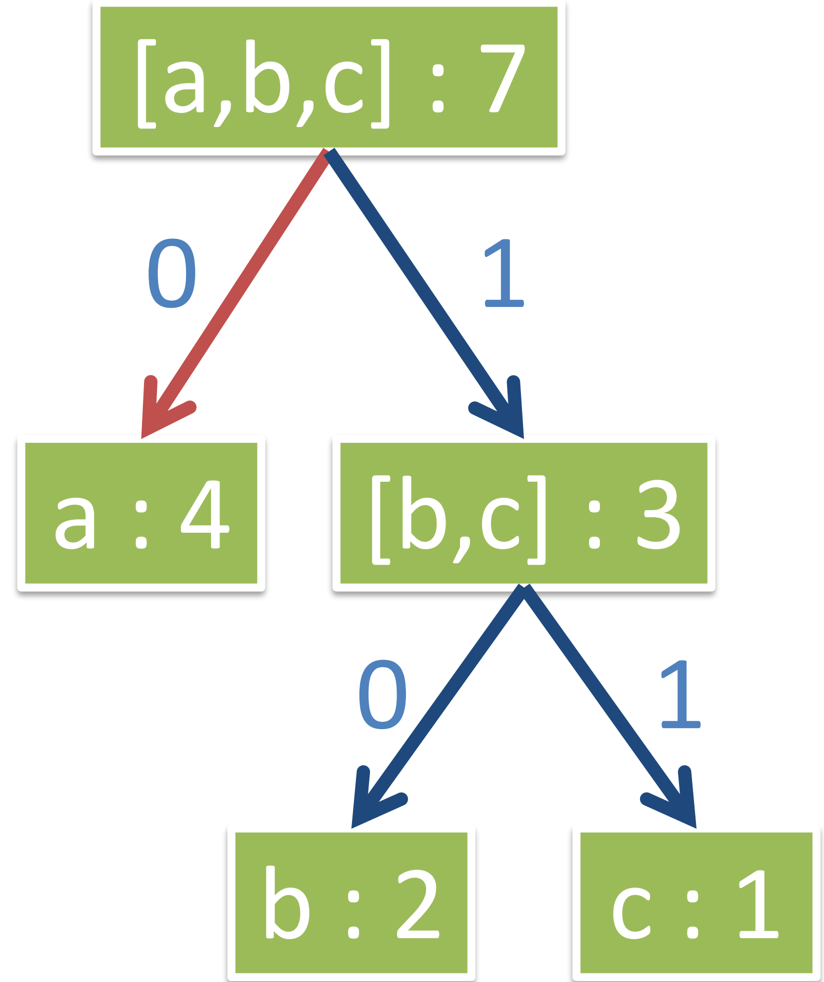


Sıkıştırılmış Dosyanın Açılması

0100110100

aba

- Bir sonraki bit olan “0”ı okuduk ve ağaç üzerinde “a” karakterine vardık.
- O halde “a” karakterini basıp ağaçta en başa dönüyoruz.

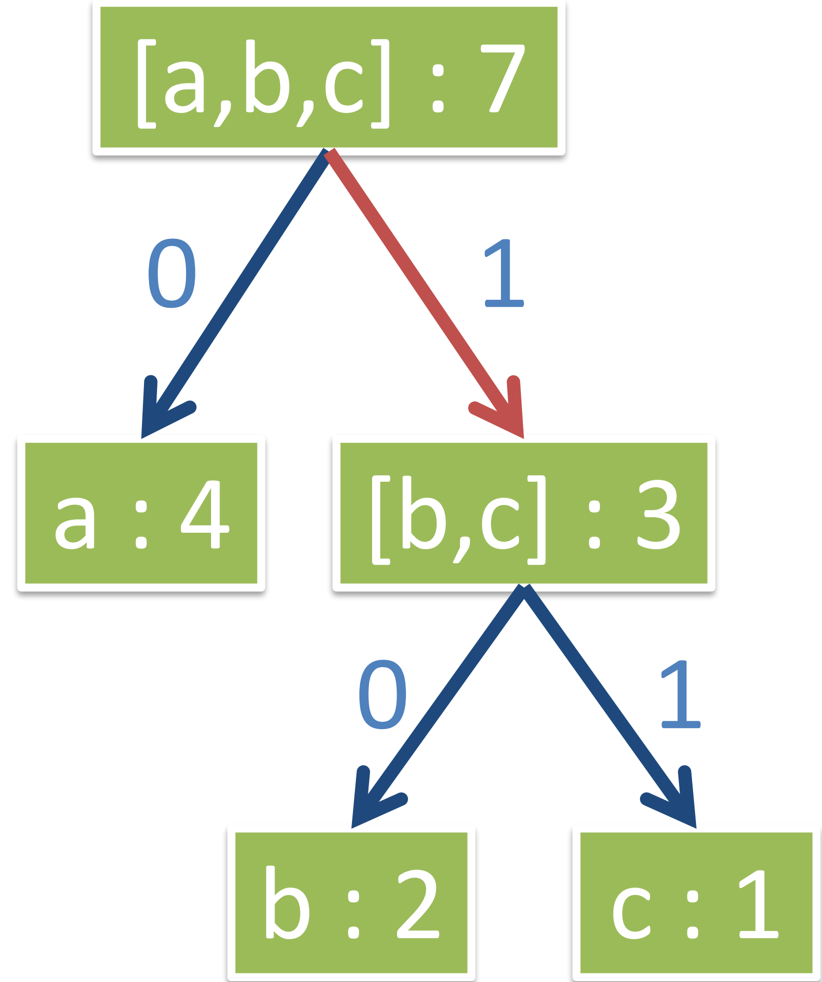


Sıkıştırılmış Dosyanın Açılması

0100110100

aba

- Bir sonraki bit olan "1"i okuduk ve ağaç üzerinde herhangi bir sona varamadık.
- O halde bulunduğumuz yerden devam ediyoruz.

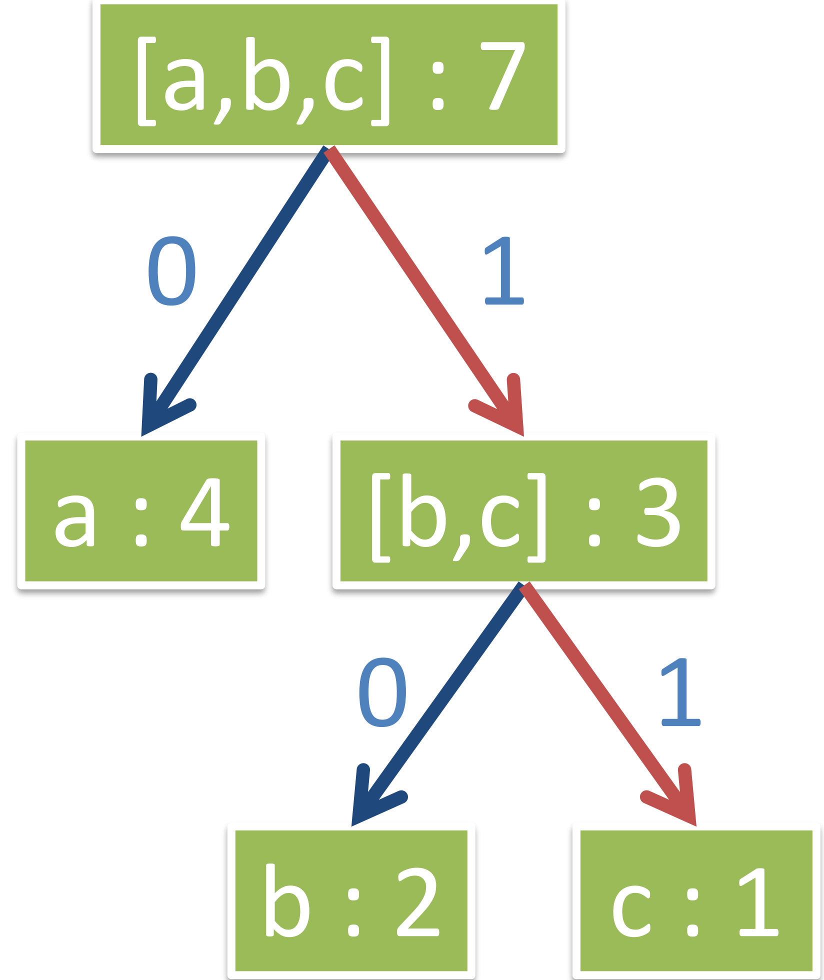


Sıkıştırılmış Dosyanın Açılması

0100110100

abac

- Bir sonraki bit olan “1”i okuduk ve ağaç üzerinde “c” karakterine vardık.
- O halde “c” karakterini basıp ağaçta en başa dönüyoruz.

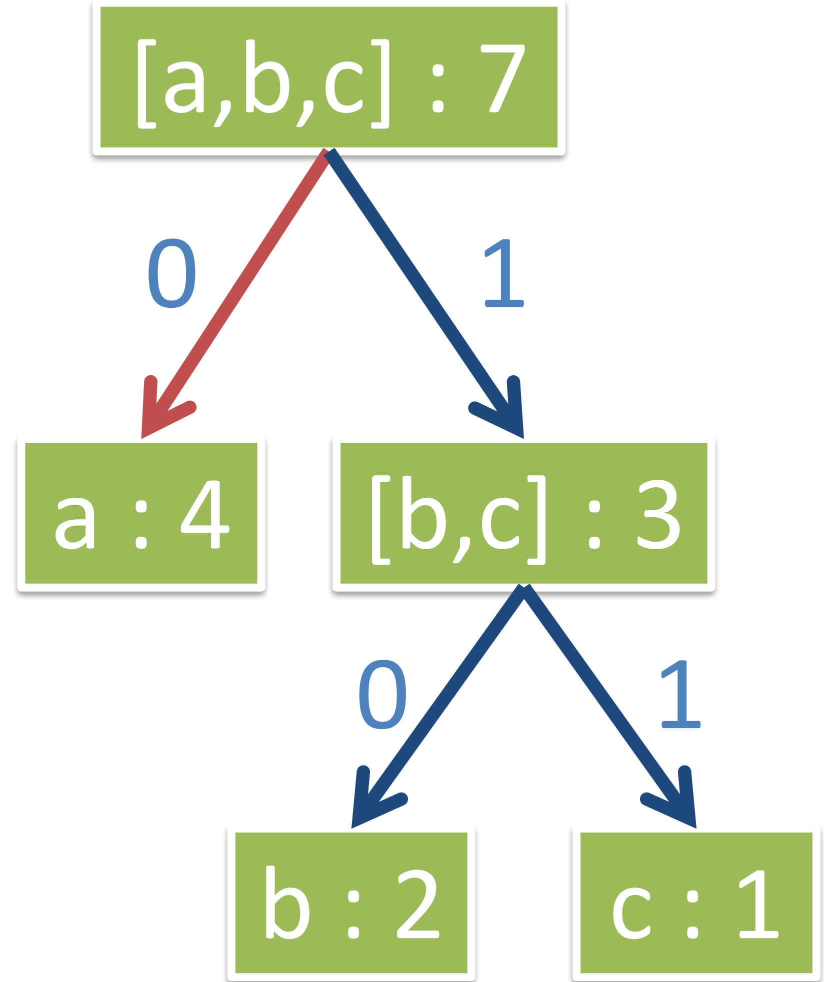


Sıkıştırılmış Dosyanın Açılması

0100110100

abaca

- Bir sonraki bit olan “0”ı okuduk ve ağaç üzerinde “a” karakterine vardık.
- O halde “a” karakterini basıp ağaçta en başa dönüyoruz.

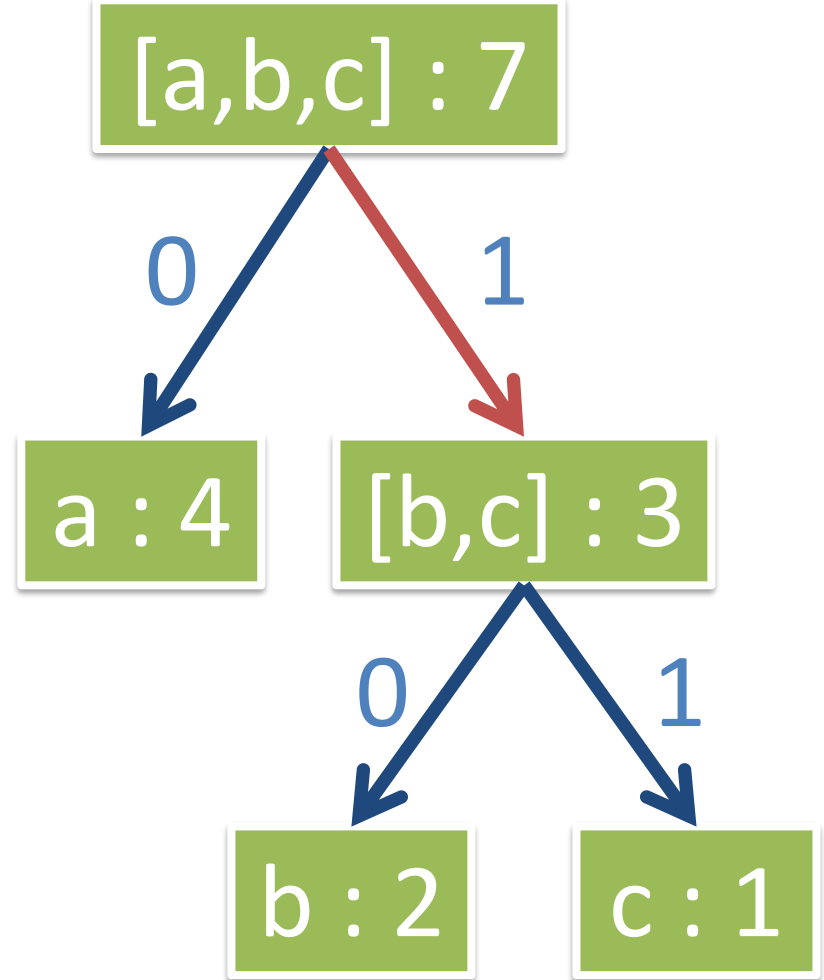


Sıkıştırılmış Dosyanın Açılması

0100110100

abaca

- Bir sonraki bit olan "1"i okuduk ve ağaç üzerinde herhangi bir sona varamadık.
- O halde bulunduğumuz yerden devam ediyoruz.

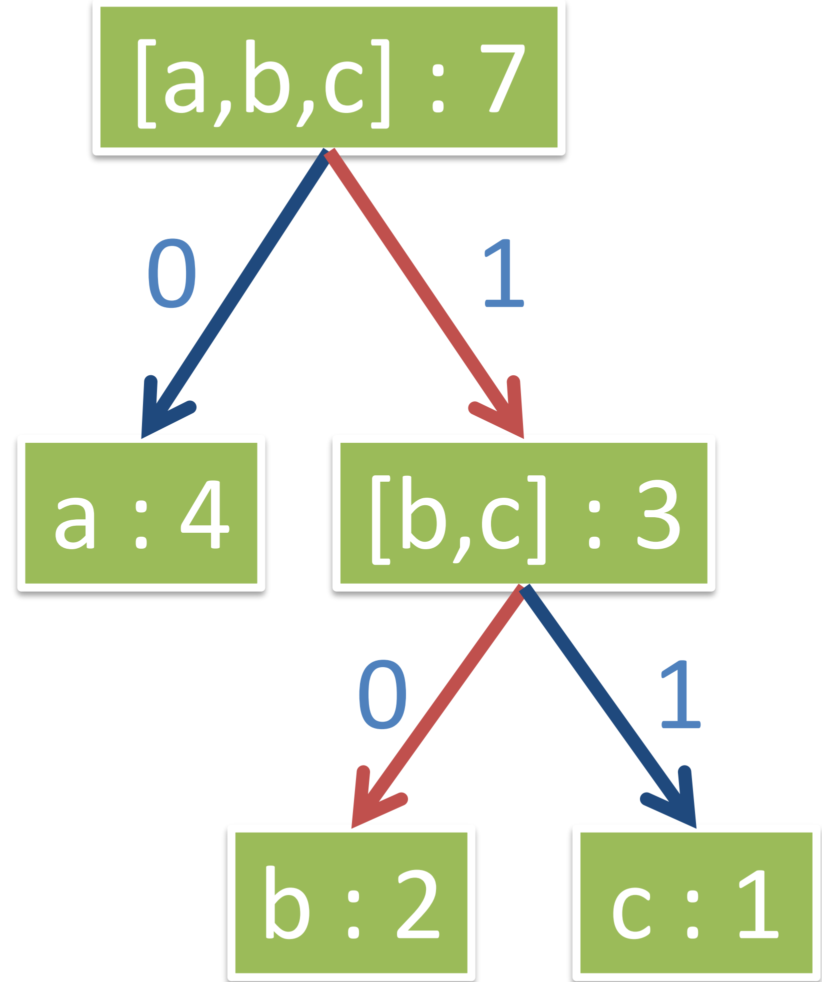


Sıkıştırılmış Dosyanın Açılması

0100110100

abacab

- Bir sonraki bit olan “0”ı okuduk ve ağaç üzerinde “b” karakterine vardık.
- O halde “b” karakterini basıp ağaçta en başa dönüyoruz.

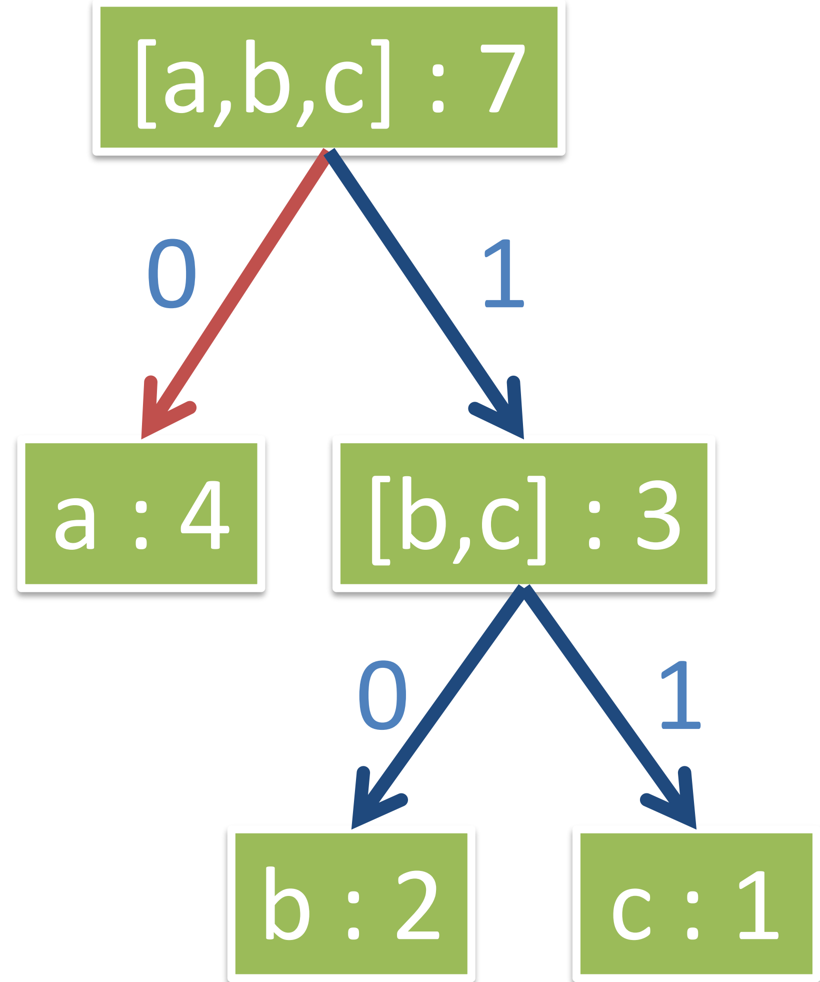


Sıkıştırılmış Dosyanın Açılması

0100110100

abacaba

- Son bit olan "0"ı okuduk ve ağaç üzerinde "a" karakterine vardık.
- O halde "a" karakterini basıp ağaçta en başa dönüyoruz.
- Bit dizisi sona erdiği için kod açma işlemini sona erdiriyoruz.



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 tarafınan 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	LZW	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 genelikle sözlük (dictionary) kullanır.

- LZ78, veri setindeki bir veya daha fazla karakteri çakışmayacak(non-overlapping) ve aynı zamanda eşsiz(uniqe) 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

Örnek 1

- 2 harften meydana gelen bir alfebe olsun.

aaababbbaaabaaaaaaabaabb

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

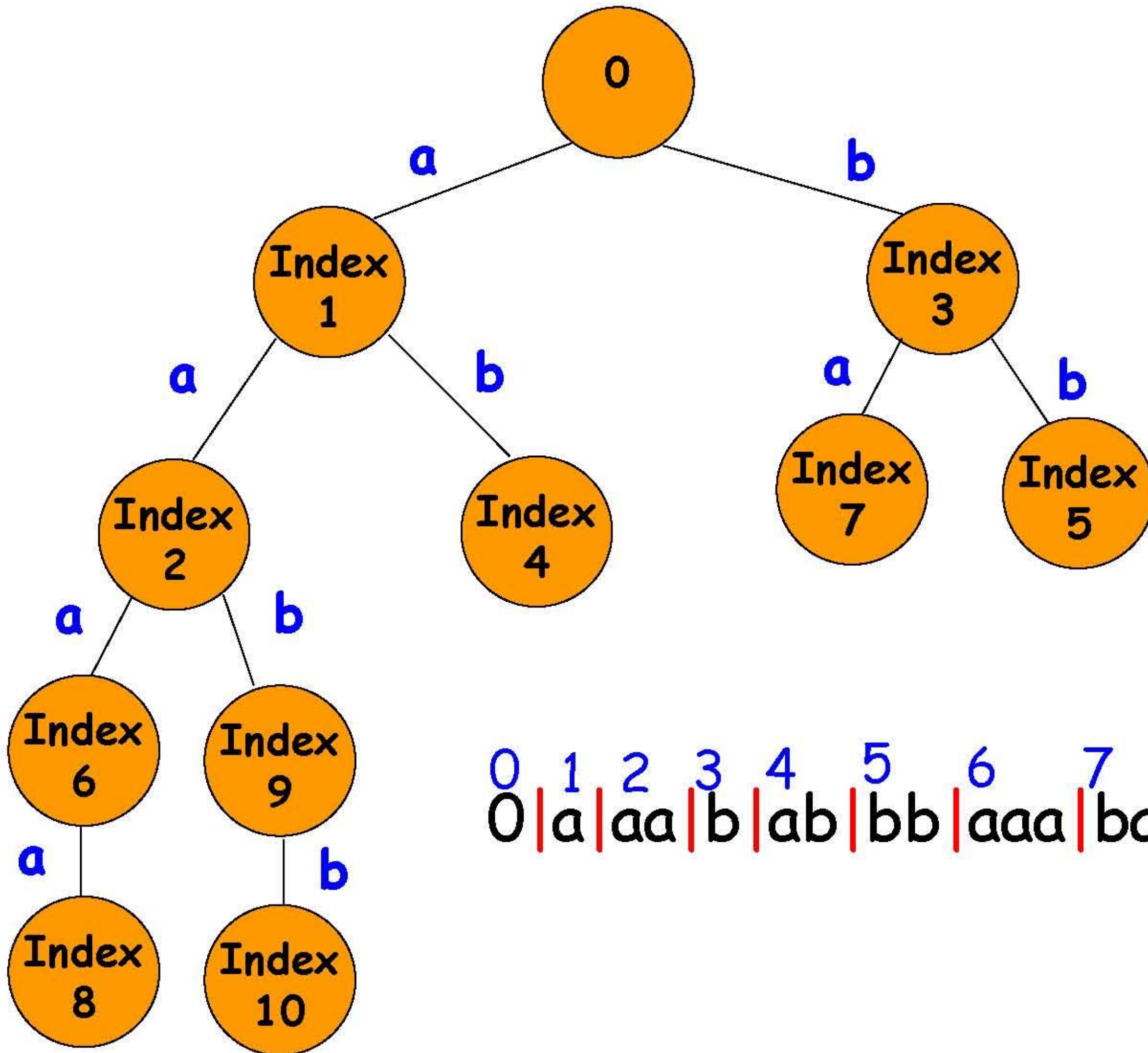
0=Null String

- 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

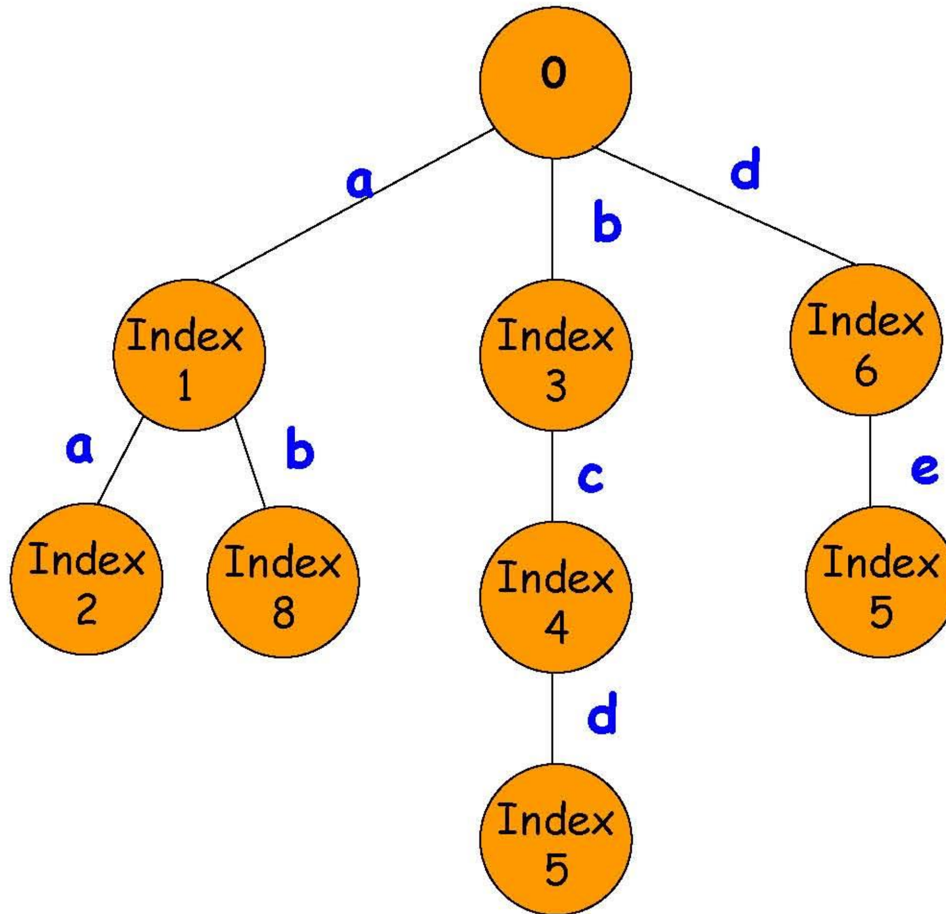
ii) İndex oluşturalım.

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) İndex oluşturalım.

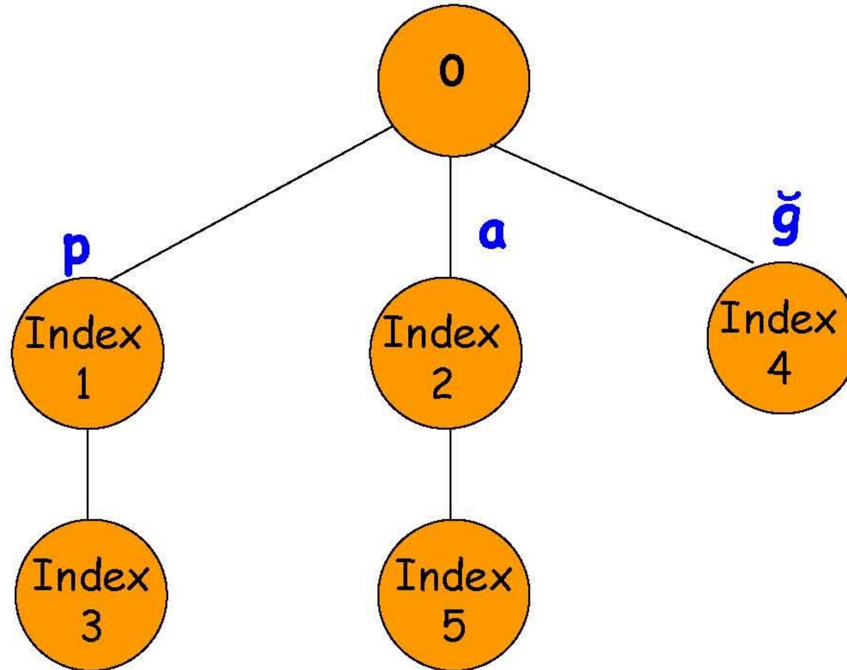
0 1 2 3 4 5

0|p|a|pa|ğ|an

iii) Bu indexi kullanarak veri setini kodlayalım.

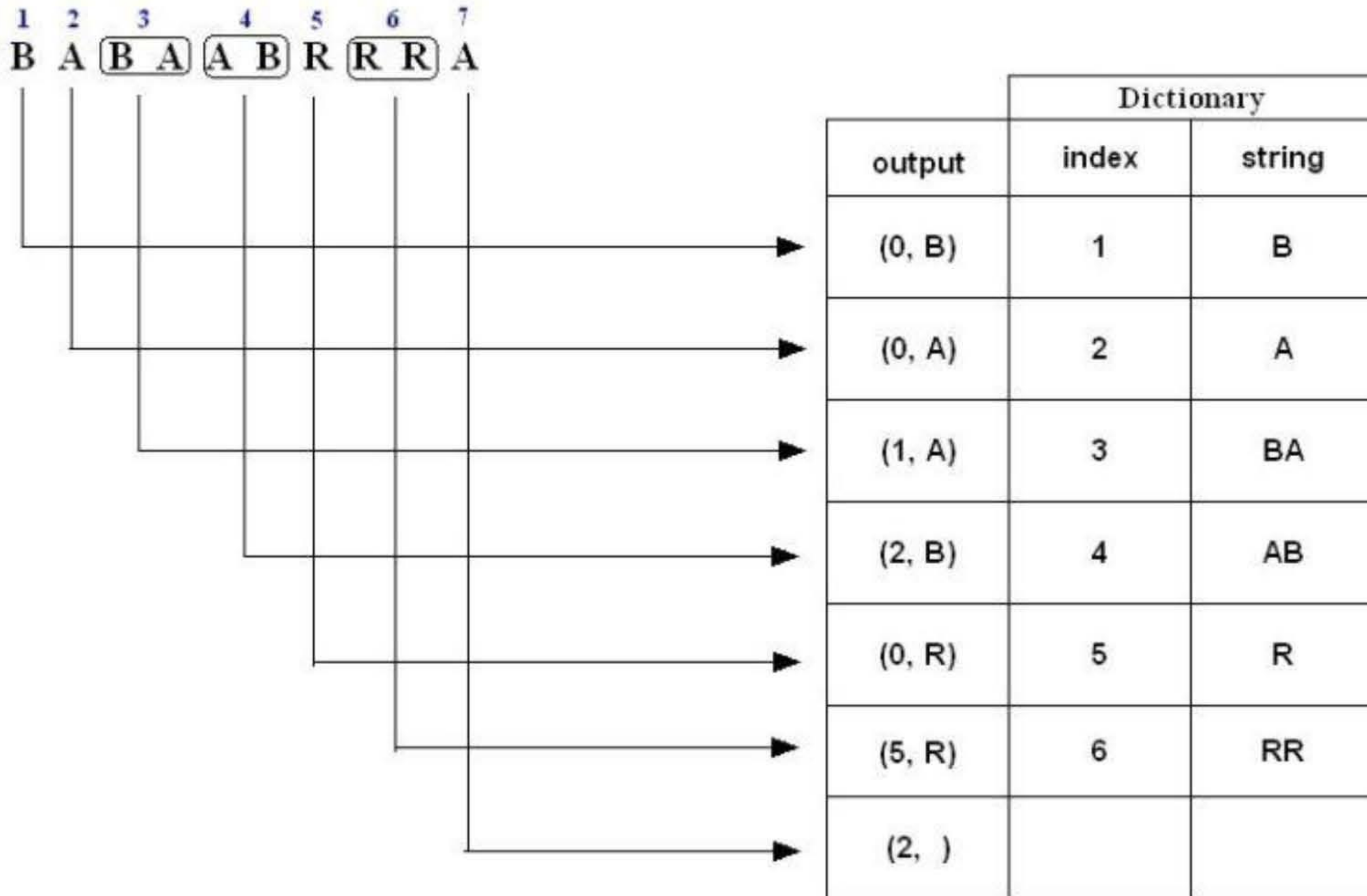
|0p|0a|1a|0ğ|2n

iv) Kodlanan veri setinin ağacını oluşturalım



Örnek 4

- "**BABAABRRRA**" şeklindeki stringi LZ78 ile kodlayalım

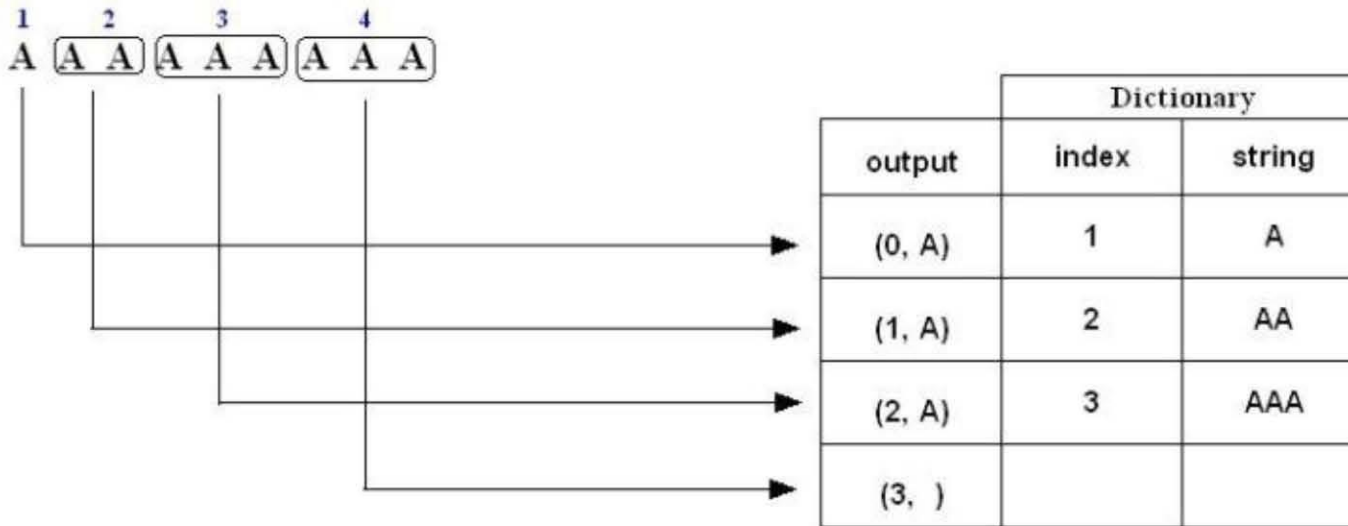


- Sıkıştırılan string : (0,B)(0,A)(1,A)(2,B)(0,R)(5,R)(2,)

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

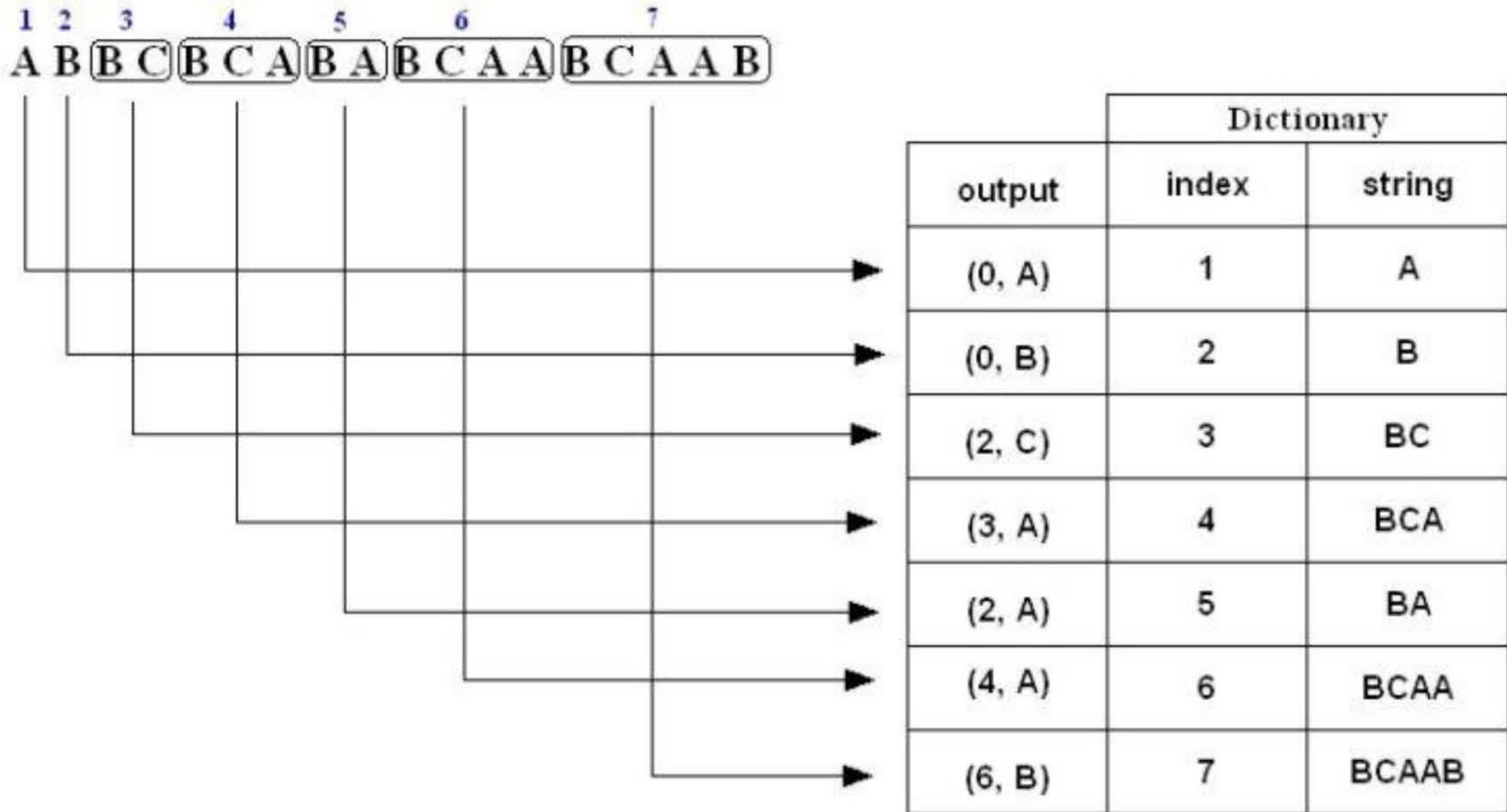


output	Dictionary	
	index	string
(0, A)	1	A
(1, A)	2	AA
(2, A)	3	AAA
(3,)		

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 indekslensin.

	(0,A)	(0,B)	(2,C)	(3,A)	(2,A)	(4,A)	(6,B)
• **Codeword index**	**1**	**2**	**3**	**4**	**5**	**6**	**7**

	(0,A)	(0,B)	(2,C)	(3,A)	(2,A)	(4,A)	(6,B)
• **Codeword index**	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Codeword	(0, A)	(0, B)	(2, C)	(3, A)	(2, A)	(4, A)	(6, B)
index	1	2	3	4	5	6	7

Bits: (1 + 8) + (1 + 8) + (2 + 8) + (2 + 8) + (3 + 8) + (3 + 8) + (3 + 8) = 71 bits

Sıkıştırma yapmadan önce ise bu değer 144 bit idi.

- Sıkıştırılmış string: 0A 1B 10C 11A 100A 101A 110B

BİTTİ