



Bölüm 5: Dizgi Algoritmaları

Algoritmalar



Dizgi Algoritmaları

- Metinlerle dolu bir dünyada yaşıyoruz.
 - E-postalar, mesajlar, sosyal medya paylaşımları, haber metinleri...
- Bilgisayarlarımızda her gün sayısız metinle karşılaşırız.
- Peki, bu metinler nasıl düzenlenir ve analiz edilir?
- Dizgi (String) algoritmaları,
 - metinlerde arama,
 - deęiřtirme,
 - karşılaştırma gibi işlemleri gerçekleştirir.



Dizgi Eşleştirme Algoritmaları

- Brute Force (Kaba Kuvvet):
 - Metinde *her konumdaki* değer, örüntü ile eşleştirmek için kontrol edilir.
 - Maksimum sayıda karşılaştırma gerektirebilir.
- Knuth-Morris-Pratt (KMP)
 - Başlangıçta *tablo* oluşturularak arama süresi azaltılır,
 - Karakter karşılaştırmalarını azaltarak hızlı çalışır.
- Boyer-Moore
 - Uzun aramalarda etkili. Kök bulma ve *kaydırma* stratejisi kullanır.
- Rabin-Karp Algoritması
 - Olasılıksal bir algoritma. *Hashing* kullanır.



Dizgi Sıkıştırma Algoritmaları

- Sıralı Sıkıştırma Kodlaması (Run Length Encoding)
 - Aynı veri değerleri tek bir *değer ve sayı* olarak saklanır.
 - Tekrar eden değerler yerine tekrar eden veri sayısı saklanır.
- Lempel-Ziv-Welch (LZW)
 - GIF gibi formatlarda kullanılan sözlük tabanlı sıkıştırma algoritması.
 - Tekrar eden örüntüleri *sözlük* oluşturarak kısa sembollerle temsil eder.
 - Dinamik bir sözlük kullanarak sıkıştırma sağlar.



Dizgi Sıralama Algoritmaları

- Sözlüksel Sıralama (Lexicographic Order)
 - Dizgiler, *alfabetik* sıraya benzer sıralanır.
 - Her karakterin *ASCII* değeri karşılaştırılarak sıralama yapılır.
- Radix Sıralama
 - Karşılaştırmalı olmayan bir tam sayı sıralama algoritmasıdır.
 - Veriler tamsayı anahtarlarına sahiptir.
 - Aynı konumda aynı değeri paylaşan verileri *gruplandırarak* sıralar.
 - Her basamak için ayrı ayrı işlem yapılır.



Dizgi Ayırıştırma (Parsing) Algoritmaları

- Düzenli İfadeler (Regular Expressions)
 - Bir arama örüntüsünü tanımlayan karakter dizisi,
 - Belirli bir örüntüye uyan tüm dizgileri bulmak için kullanılır
- Sonlu Durum Makineleri (Finite State Machines - FSM)
 - Dizgi içindeki örüntüleri tanımak için kullanılan hesaplama modelleri,
 - Belirli bir girdi dizisindeki *geçişlerin* durumlarını izleyen bir otomat,
 - Karmaşık ayırıştırma ve analiz işlemlerinde kullanılır.



Dizgi Düzenleme Mesafesi Algoritmaları

- Levenshtein Mesafesi
 - İki dizgi arasındaki benzerliği ölçen bir metrik,
 - Bir dizgiden diğerine dönüştürmek için gereken minimum tek karakterli *düzenleme sayısı* olarak tanımlanır.
- En Uzun Ortak Alt Dizi (Longest Common Subsequence - LCS)
 - İki dizginin *ortak olan en uzun* alt dizisi,
 - Karakterlerin sıralı olmasını gerektirmez, ancak sıra korunmalıdır.
 - Dizgiler arasındaki benzerlik veya farkı belirlemek için kullanılır.



Dizgi Dönüşüm Algoritmaları

- Sonek Dizisi (Suffix Array)
 - Bir dizginin tüm son eklerinin bir dizisi.
 - Dizgi içindeki alt dizgilerin bir temsili olarak kullanılır.
- Burrows-Wheeler Dönüşümü (BWT)
 - Bir dizginin tersine dönüştürülmesiyle elde edilen yeni bir form,
 - Bzip2 gibi sıkıştırma algoritmaları için ön işlem adımı olarak kullanılır.

String Matching





String Naive Algoritması

- Metin içinde belirli bir örüntüyü (*pattern*) arayan basit bir algoritma.
- *Naive* (Saf) olarak adlandırılır çünkü basit bir yaklaşım kullanır.
- Ortalama ve en kötü durumda $O(m n)$ zaman karmaşıklığına sahiptir.
 - (m: örüntü uzunluğu, n: metin uzunluğu)



İşleyiş

- Metindeki her karakter için örüntünün ilk karakterinin eşleşip eşleşmediği kontrol edilir.
- Eşleşen karakterlerin tümü için örüntünün tam olarak eşleşip eşleşmediğini kontrol edilir.
 - Eşleşme Durumu: eşleşme pozisyonu rapor edilir.
 - Eşleşmeme Durumu: sonraki karakterler kontrol edilir.
- Metindeki tüm karakterler için adımlar tekrarlanır.



Brute Force - Naive

T	H	I	S		I	S		A		S	I	M	P	L	E		E	X	A	M	P	L	E	
S	I	M	P	L	E																			
	S	I	M	P	L	E																		
		S	I	M	P	L	E																	
			S	I	M	P	L	E																
				S	I	M	P	L	E															
					S	I	M	P	L	E														
						S	I	M	P	L	E													
							S	I	M	P	L	E												
								S	I	M	P	L	E											
									S	I	M	P	L	E										
										S	I	M	P	L	E									



Brute Force - Naive

HELLO WORLD



Brute Force - Naive





Brute Force - Naive





Brute Force - Naive





Brute Force - Naive





Brute Force - Naive





Brute Force - Naive





Brute Force - Naive





Brute Force - Naive





Brute Force - Naive







Knuth-Morris-Pratt (KMP) Algoritması

- Metin içinde belirli bir örüntüyü bulmak için kullanılır.
- *Donald Knuth, Vaughan Pratt ve James H. Morris* tarafından geliştirilmiştir.
- Ortalama ve en kötü durumda $O(m + n)$ zaman karmaşıklığına sahiptir.
 - (m: örüntü uzunluğu, n: metin uzunluğu)



İşleyiş

- Ön İşleme: Örüntü içindeki her karakter için,
 - eşleşme durumunda geri dönecek pozisyonları belirleyen,
 - en uzun önek(*suffix*) eşleşmesini bulan bir tablo oluşturulur.
- Örüntü Arama: Metin içinde arama yapılırken,
 - örüntü ile eşleşmeyen karakterlerde geri dönecek pozisyonlar,
 - tablodan elde edilen geri dönüş değerleri kullanılarak hesaplanır.
- Eşleşme Kontrolü: Eşleşen karakterlerin tümü için örüntünün tam olarak eşleşip eşleşmediği kontrol edilir.



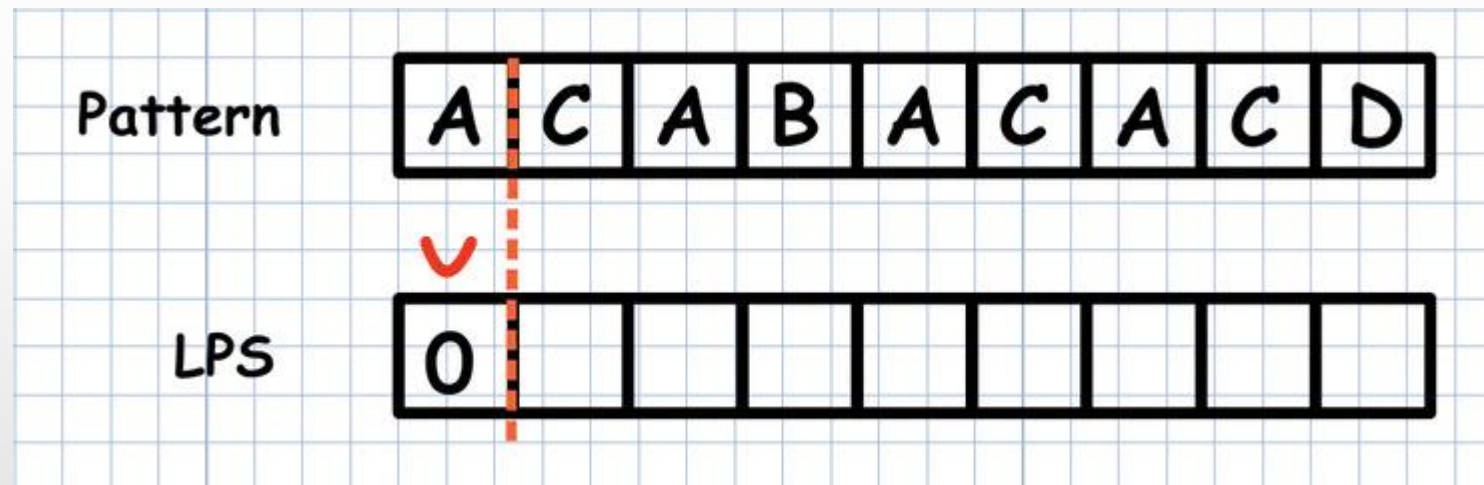
Örnek

- Metin: "ababcababcabababd"
- Örüntü: "ababd"
- Örüntü Tablosu:
 - a: 0, b: 0, a: 1, b: 2, d: 0



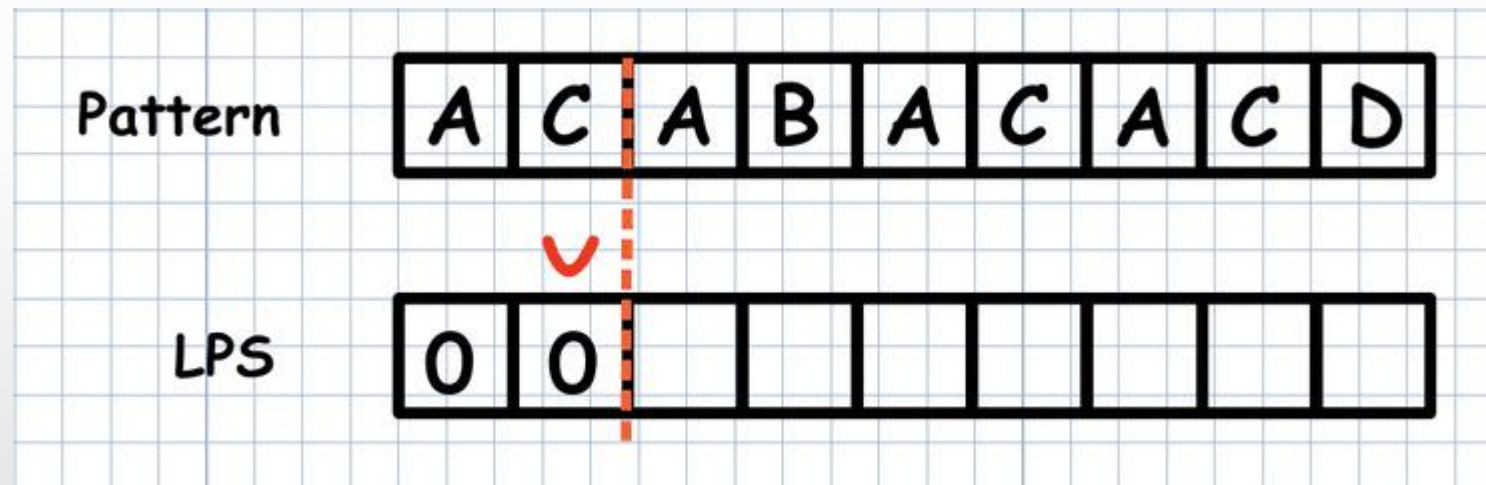
Knuth Morris Pratt

- *Longest Proper Prefix*



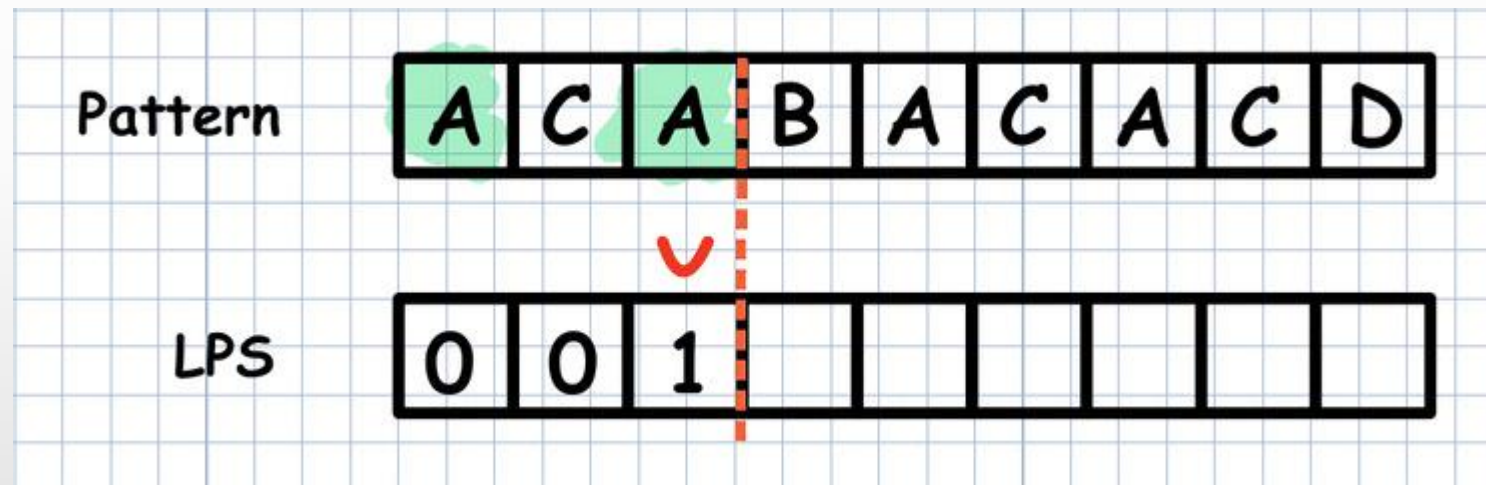


Knuth Morris Pratt



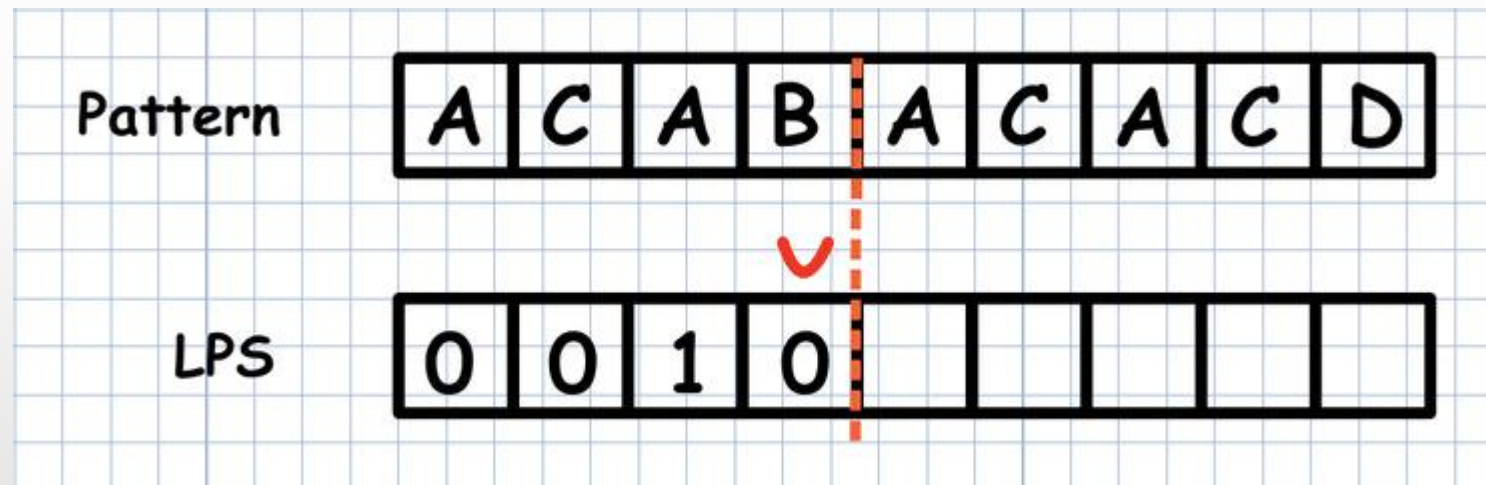


Knuth Morris Pratt



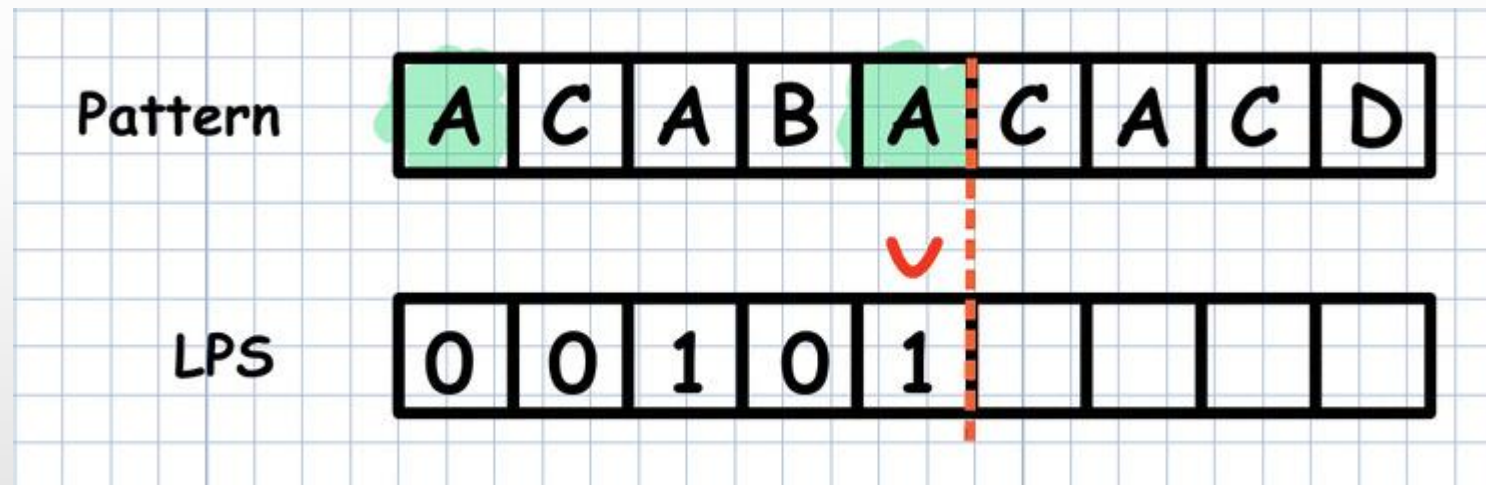


Knuth Morris Pratt



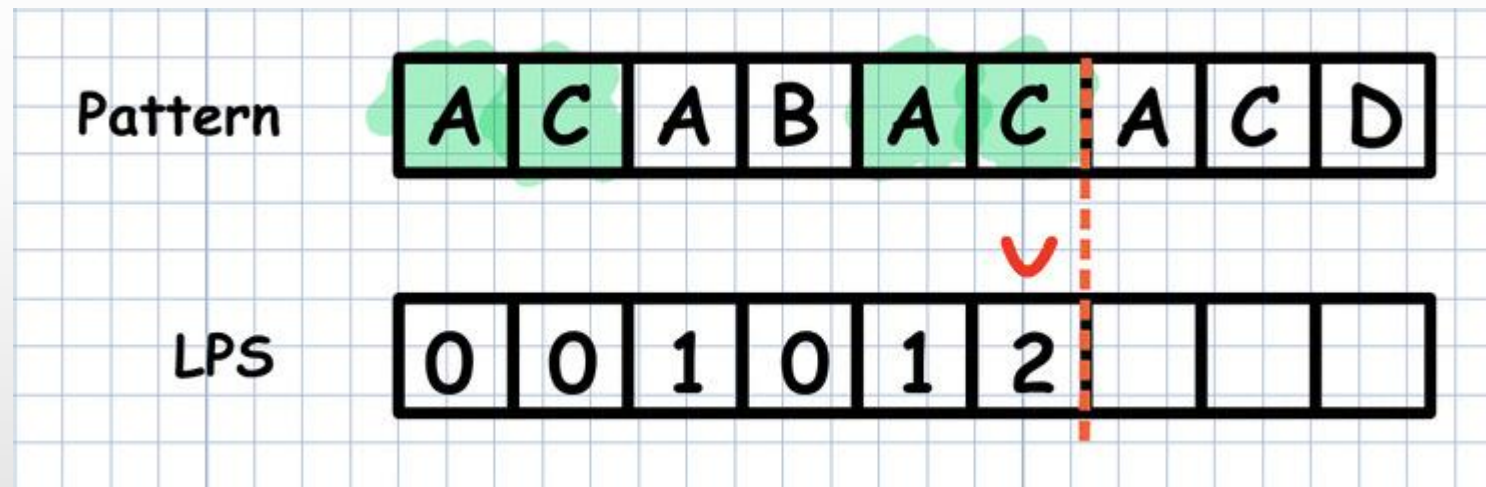


Knuth Morris Pratt



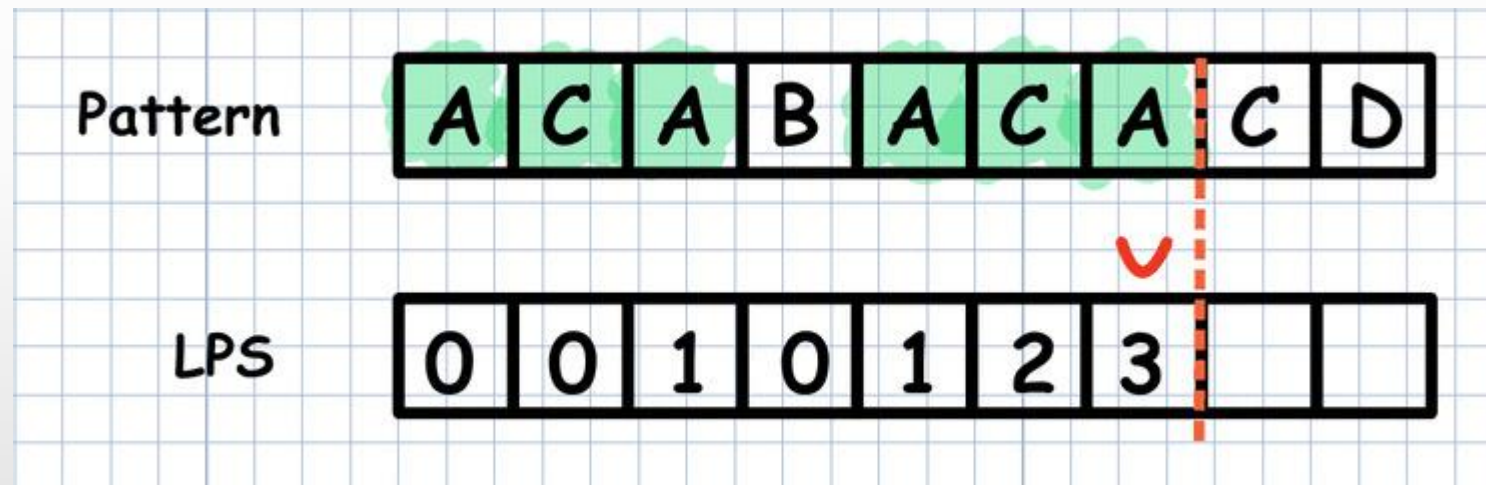


Knuth Morris Pratt



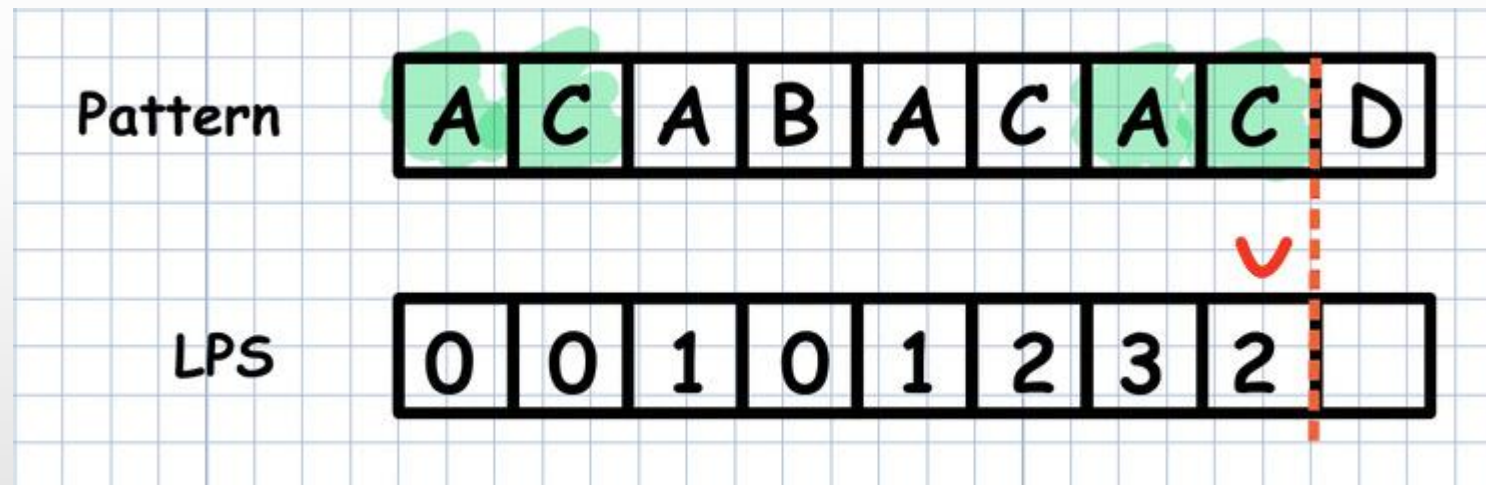


Knuth Morris Pratt





Knuth Morris Pratt





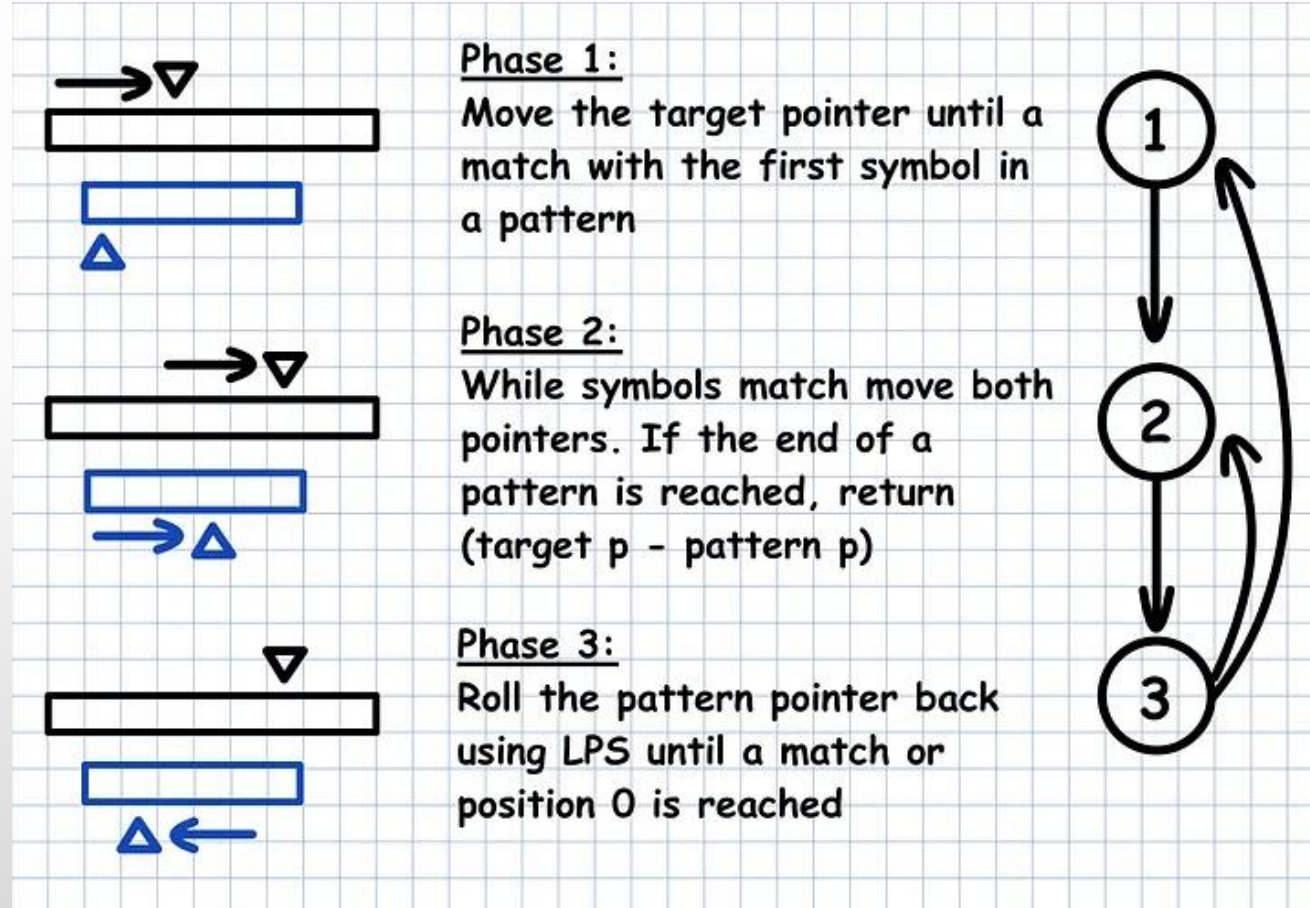
Knuth Morris Pratt

Pattern	A	C	A	B	A	C	A	C	D
LPS	0	0	1	0	1	2	3	2	0

A diagram illustrating the Knuth-Morris-Pratt (KMP) algorithm. It shows a pattern string "ACABACACD" and its corresponding Longest Prefix Suffix (LPS) array [0, 0, 1, 0, 1, 2, 3, 2, 0]. A red checkmark is placed above the last 'C' in the pattern and the '2' in the LPS array, indicating a match between the character and its LPS value. A vertical dashed orange line is at the end of the pattern.

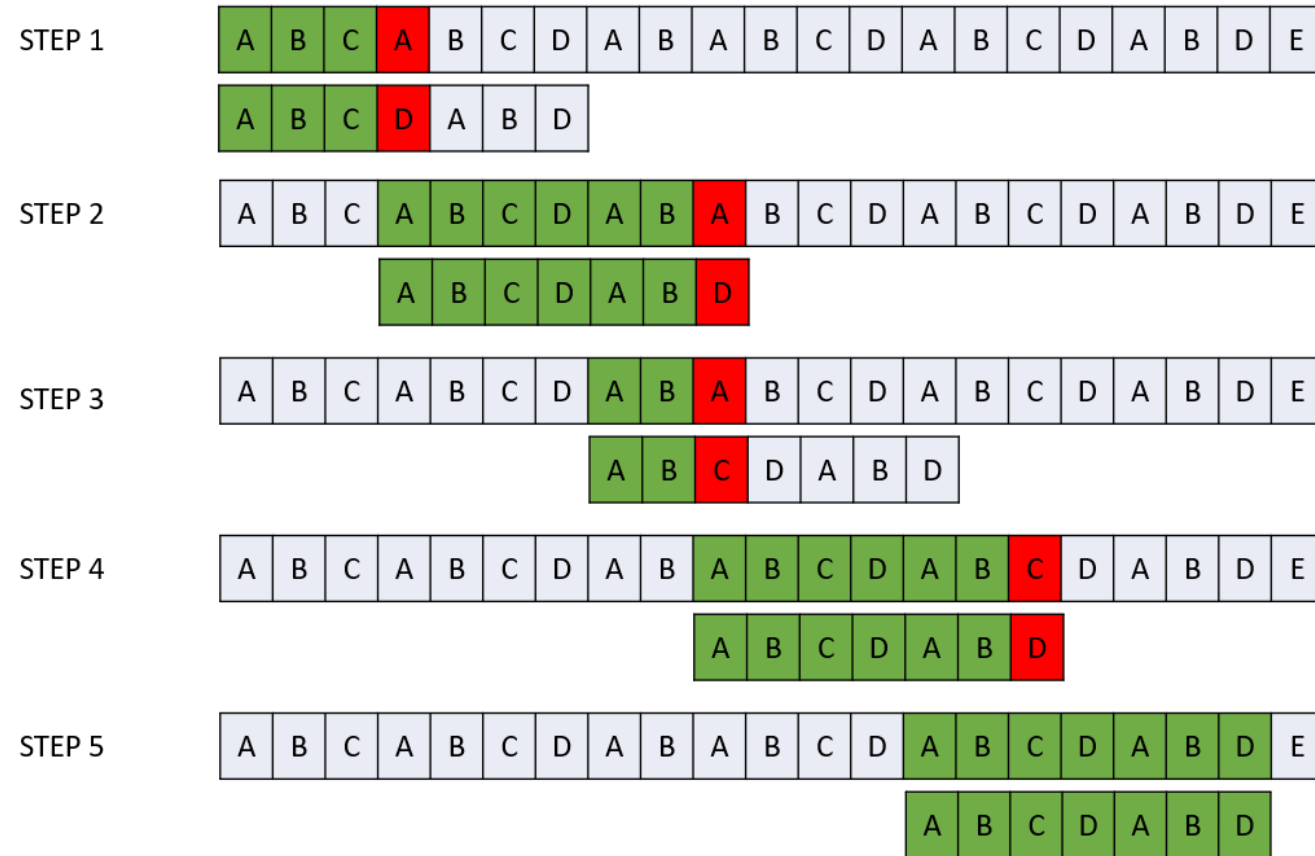


Knuth Morris Pratt





Knuth Morris Pratt







Boyer-Moore Algoritması

- Metin içinde belirli bir örüntüyü arar.
- *Robert S. Boyer ve J Strother Moore* tarafından geliştirilmiştir.
- Ortalama ve en kötü durumda $O(n/m)$ zaman karmaşıklığına sahiptir.
 - (n: metin uzunluğu, m: desen uzunluğu)



İşleyiş

- Ön İşleme: Örüntü içindeki her karakter için eşleşme durumunda geri dönecek pozisyonları belirleyen bir tablo oluşturulur.
- Arama: Metin içinde örüntüyü ararken, eşleşmeyen karakterlerde tablodan yararlanarak geri dönecek pozisyonlar hesaplanır.
- Eşleşme Kontrolü: Eşleşen karakterlerin tümü için örüntünün tam olarak eşleşip eşleşmediği kontrol edilir.
- Kötü Karakter Kaydırma Kuralı: Eşleşmeyen bir karakter varsa, örüntüdeki bu karakterin metindeki en sağdaki konumu baz alınarak kaydırma yapılır.
- İyi Sone Kuralı: Eşleşmeyen bir alt-dizgi varsa, örüntüdeki bu alt-dizginin metindeki en sağdaki konumu baz alınarak kaydırma yapılır.



Örnek

- Metin: "abccbaabccbaabcabcabbabcabc"
- Desen: "abcabcabbabcabc"
- Desen Tablosu:
 - a: 10, b: 8, c: 7
- Sonuç:
 - Pozisyon 12: "abcabcabbabcabc"

Boyer Moore



HELLO
LO L O E L L O O H E L L O

Boyer Moore



HELLO
LOLLOELLOHELLO

Boyer Moore



HELLO
LO LLOELLOHELLO



Boyer Moore



✘ Couldn't match, move on

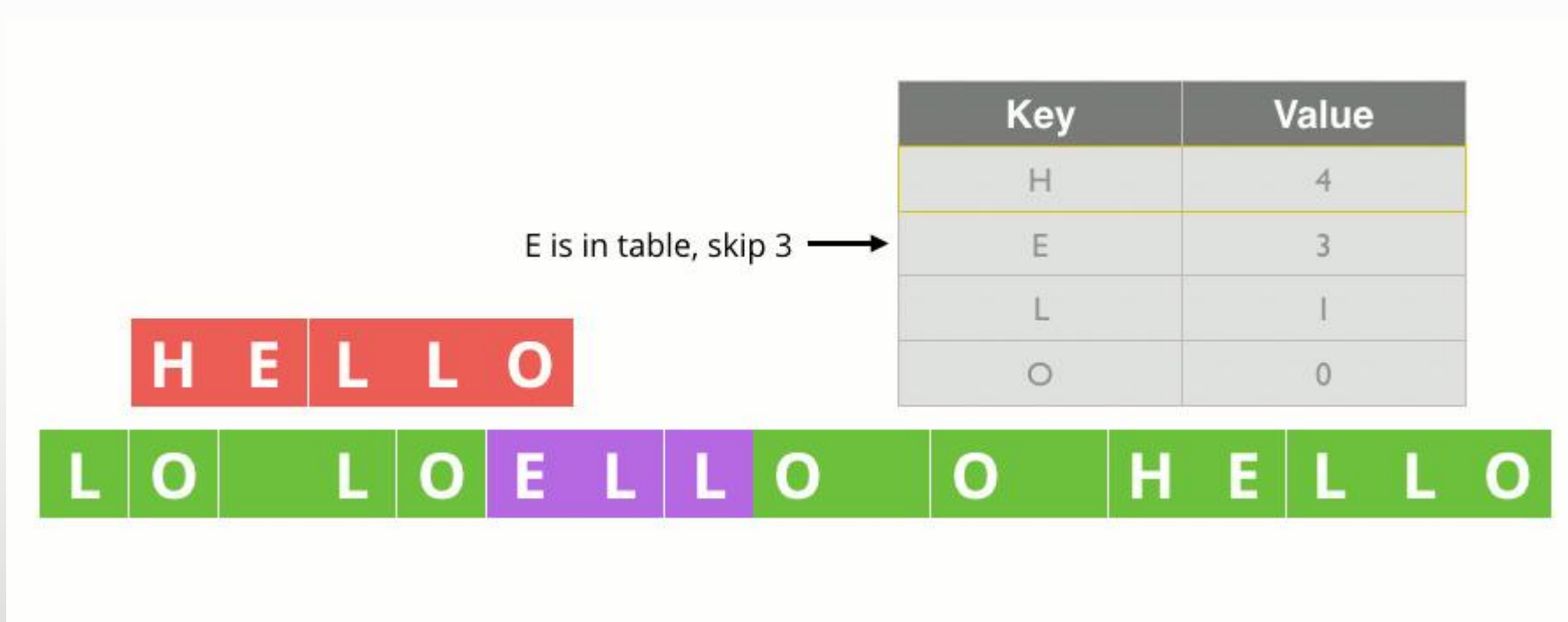
Boyer Moore



HELLO
LO L O E L L O O H E L L O



Boyer Moore



Boyer Moore



HELLO
LO LOELLO O HELLO



Boyer Moore



Boyer Moore



Boyer Moore



Boyer Moore





Boyer Moore



Boyer Moore

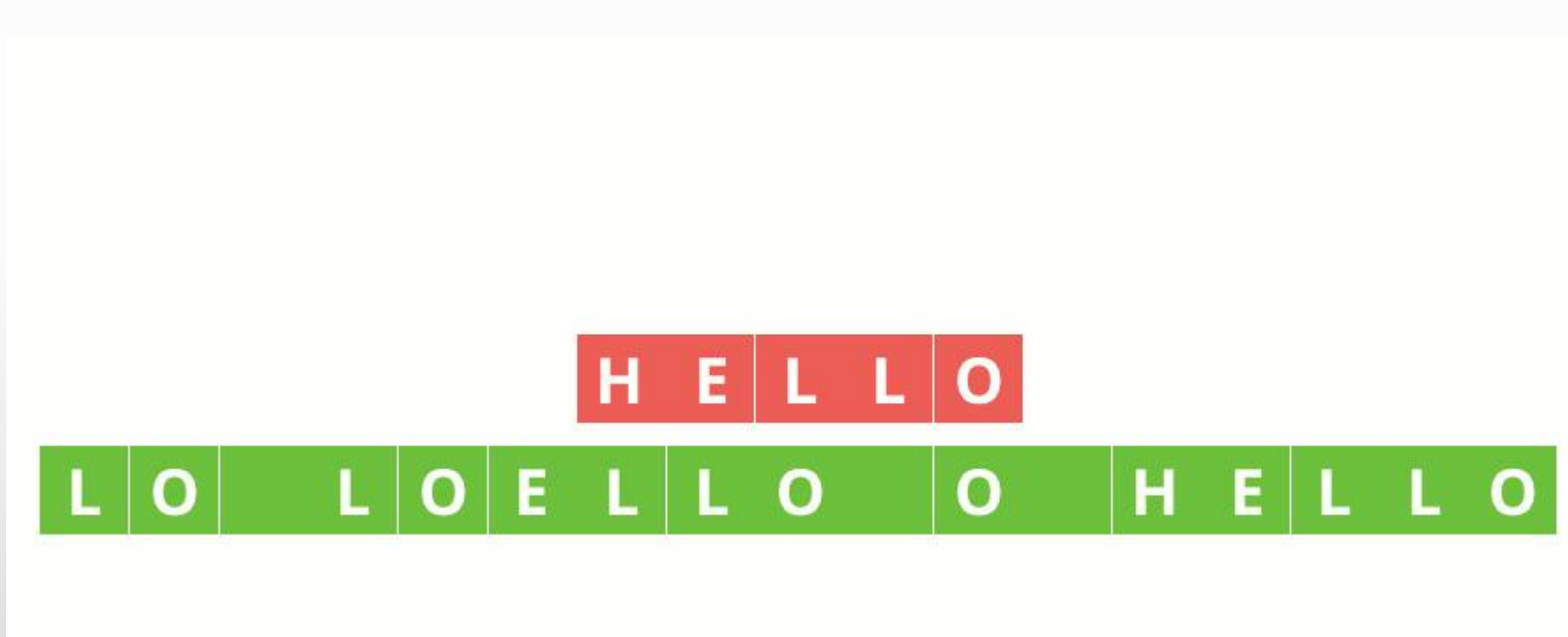


HELLO
LO LOELLO O HELLO

Boyer Moore



Boyer Moore



Boyer Moore



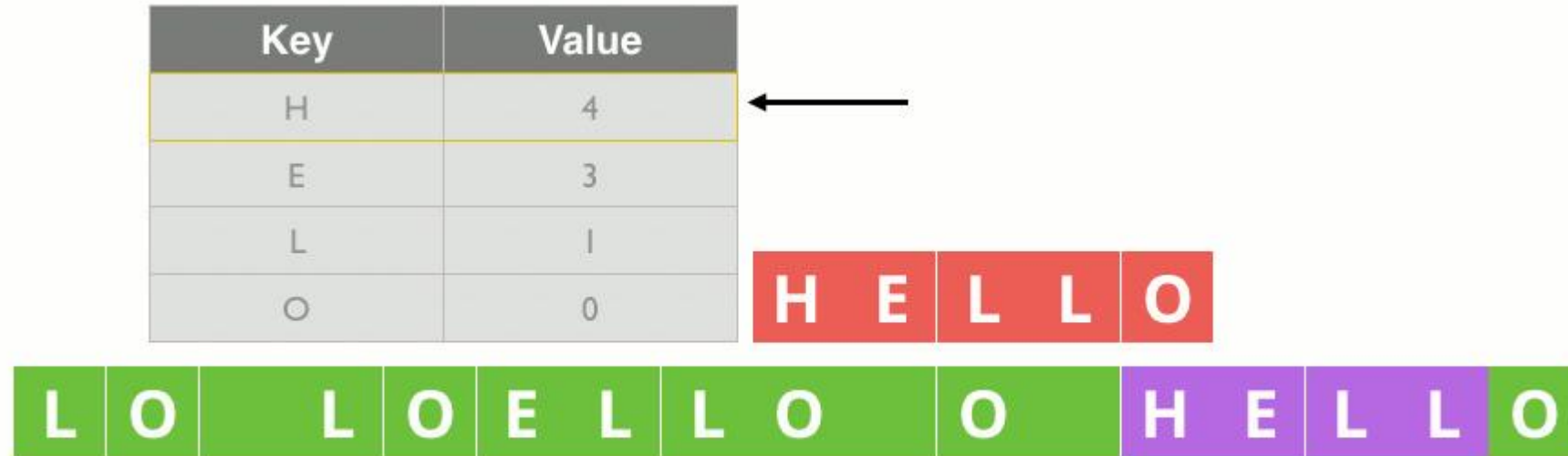
Boyer Moore



HELLO
LO LOELLO O HELLO



Boyer Moore



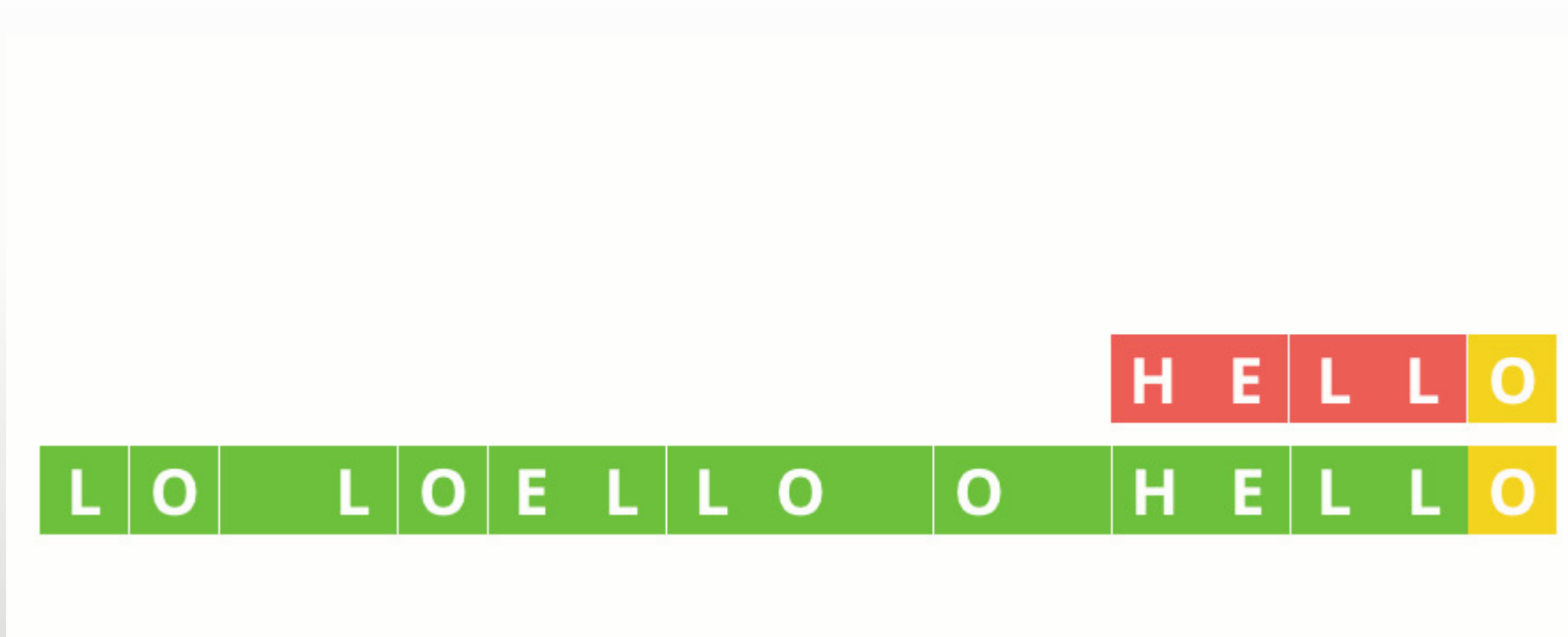
H is in skip table, skip 4

Boyer Moore

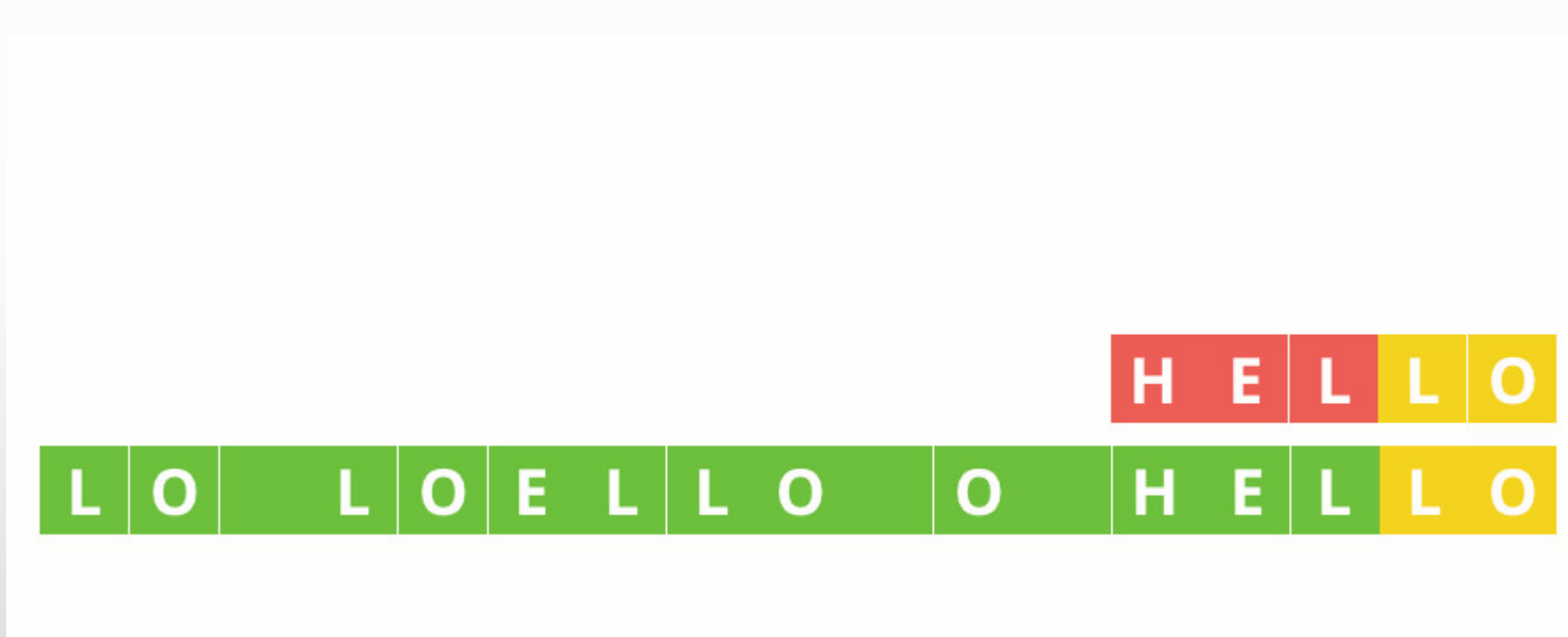


HELLO
LO L O E L L O O H E L L O

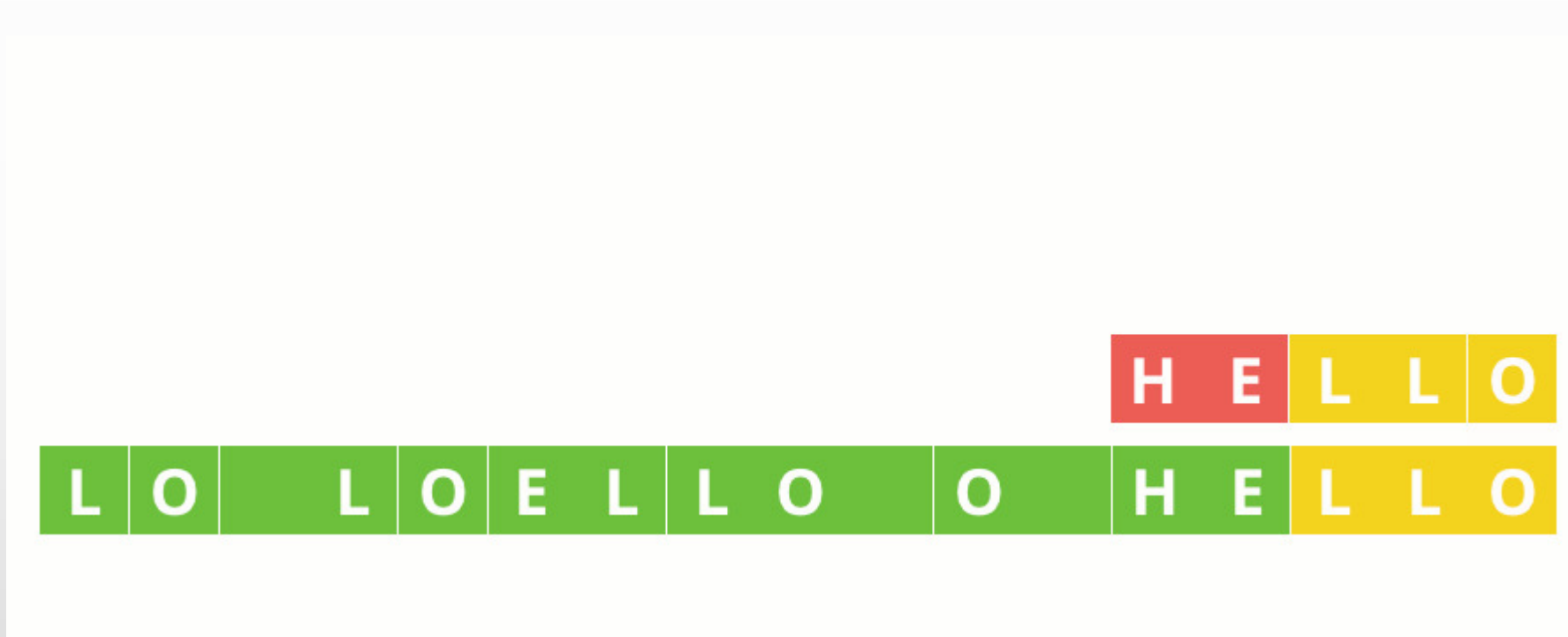
Boyer Moore



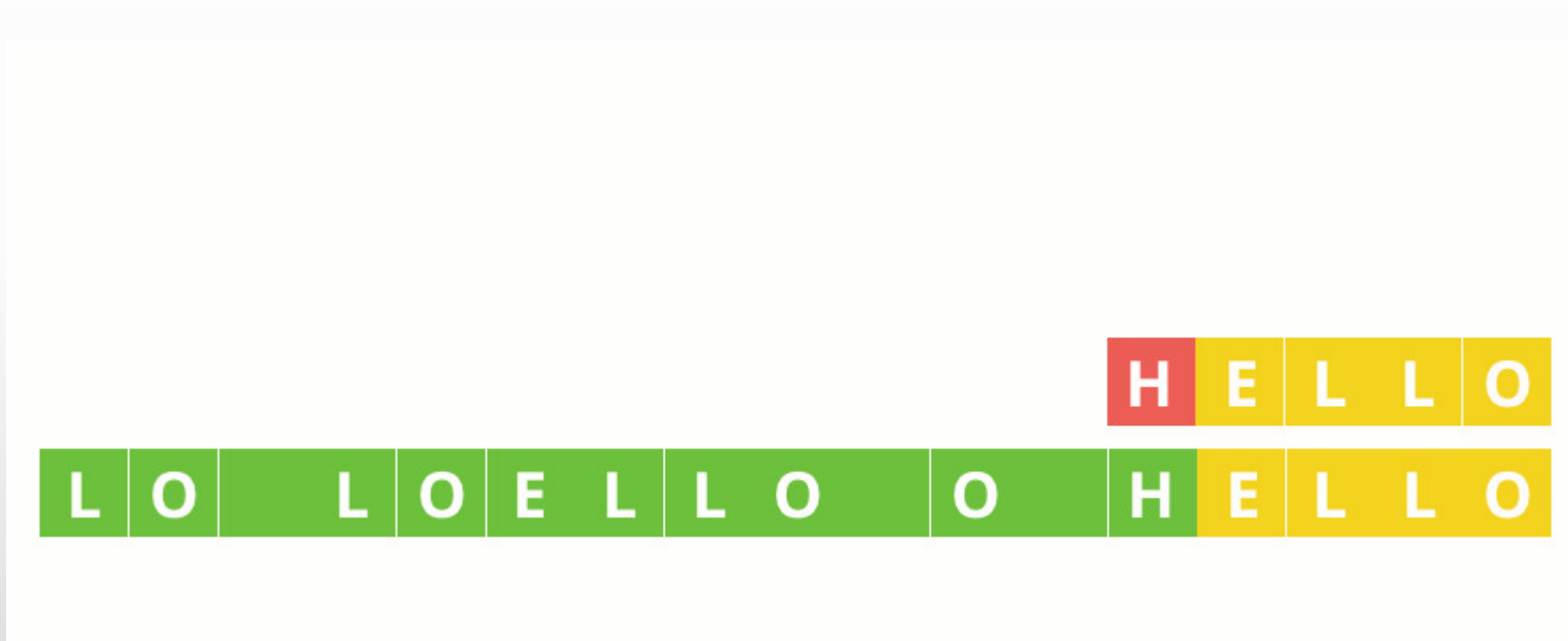
Boyer Moore



Boyer Moore



Boyer Moore



Boyer Moore



HELLO
LO L O E L L O O H E L L O

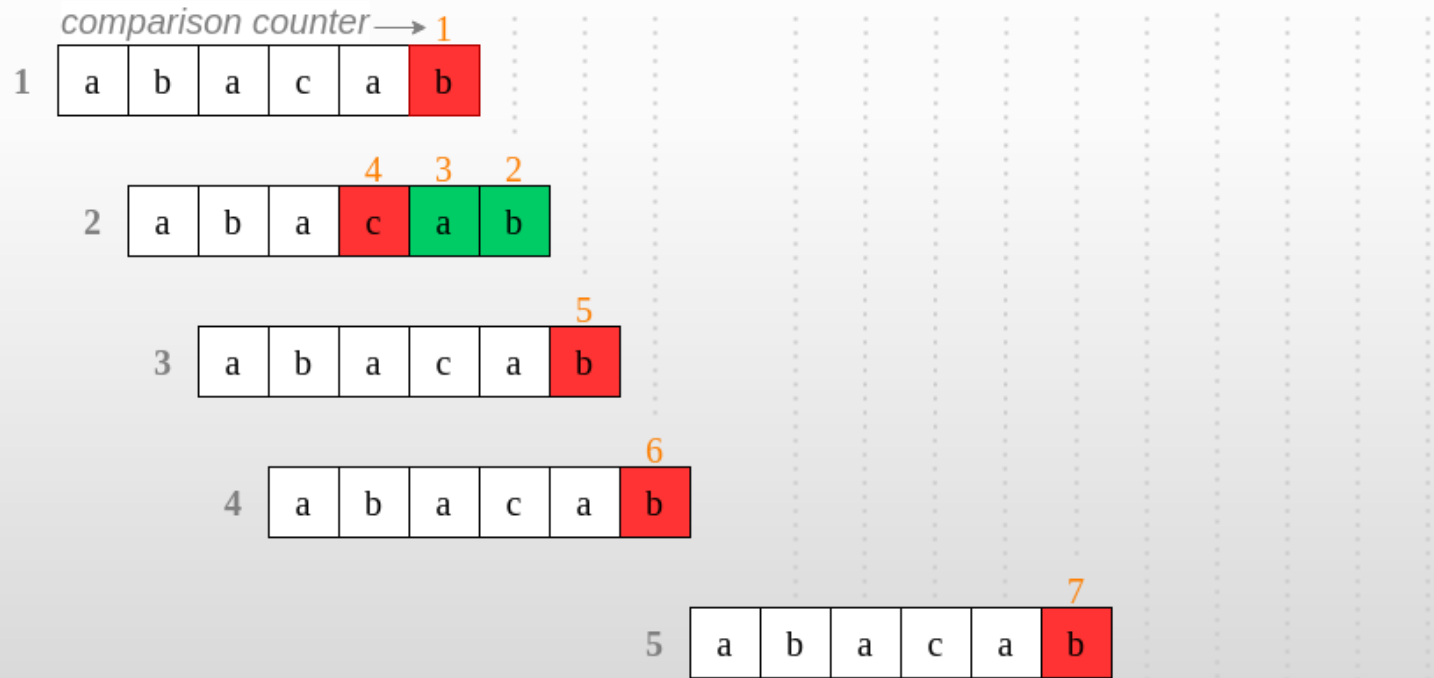
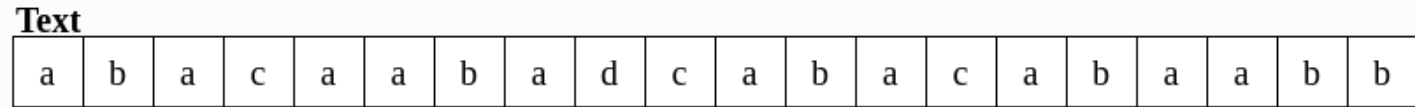
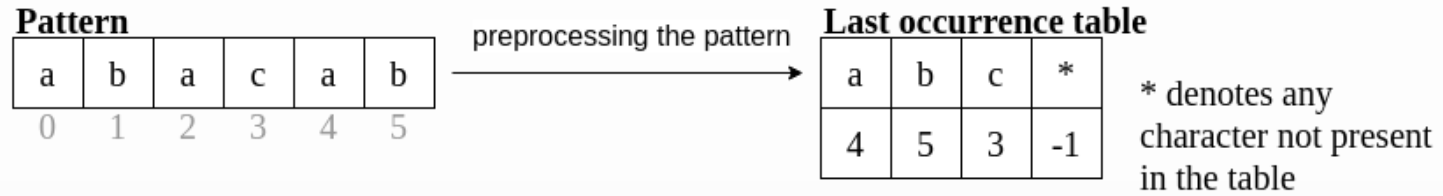
Boyer Moore





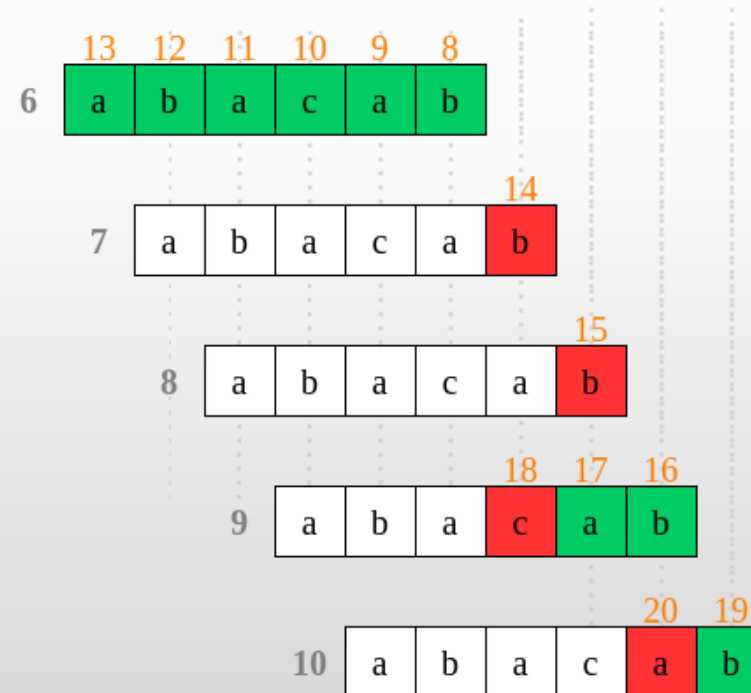
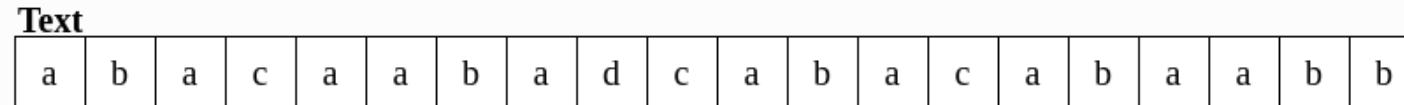
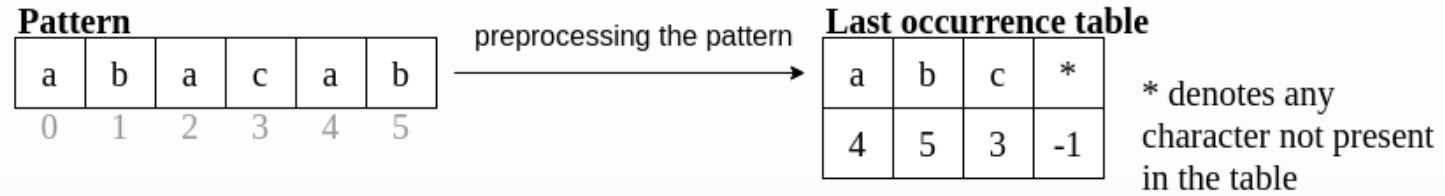


Boyer Moore





Boyer Moore







Rabin-Karp Algoritması

- Metin içinde belirli bir örüntüyü bulmak için kullanılır.
- *Michael O. Rabin* ve *Richard M. Karp* tarafından geliştirilmiştir.
- Ortalama ve en kötü durumda $O(n + m)$ zaman karmaşıklığına sahiptir.
 - (n: metin uzunluğu, m: örüntü uzunluğu)



İşleyiş

- *Hash*: Örüntü ve metin içindeki alt dizgelerin hash değerleri hesaplanır.
- Eşleşme Kontrolü: Hash değerleri eşleşen alt dizgeler karşılaştırılır.
- Doğrulama: Eşleşme olduğunda, karakter bazında doğrulanır.
- Kaydırma ve Yeniden Hesaplama: Eşleşme olmadığında, yeni bir alt dizge seçilir ve hash değeri yeniden hesaplanır.
- Tekrarlama: Tüm metin boyunca adımlar tekrarlanır.



Örnek

- Metin: "abracadabra"
- Örüntü: "cad"
- İşleyiş:
 - Hash Değerleri: Metin: "abr", Örüntü: "cad"
 - Eşleşme Kontrolü: Hash değerleri eşleşmez.
 - Kaydırma ve Yeniden Hesaplama: Yeni alt dizge seçilir: "bra"



Rabin Karp

$VUATS$

$5 + 1 = 6$

TS

$2 + 3 = 5$

¹ Values

U = 1

T = 2

S = 3

A = 4

V = 5



Rabin Karp

V U A T S 2 Values

1 + 4 = 5

Spurious Hit

T S

2 + 3 = 5

U = 1
T = 2
S = 3
A = 4
V = 5



Rabin Karp

V U A T S ³ Values

4 + 2 = 6

T S

2 + 3 = 5

U = 1
T = 2
S = 3
A = 4
V = 5



Rabin Karp

V U A T S ⁴ Values

2 + 3 = 5

Matched !

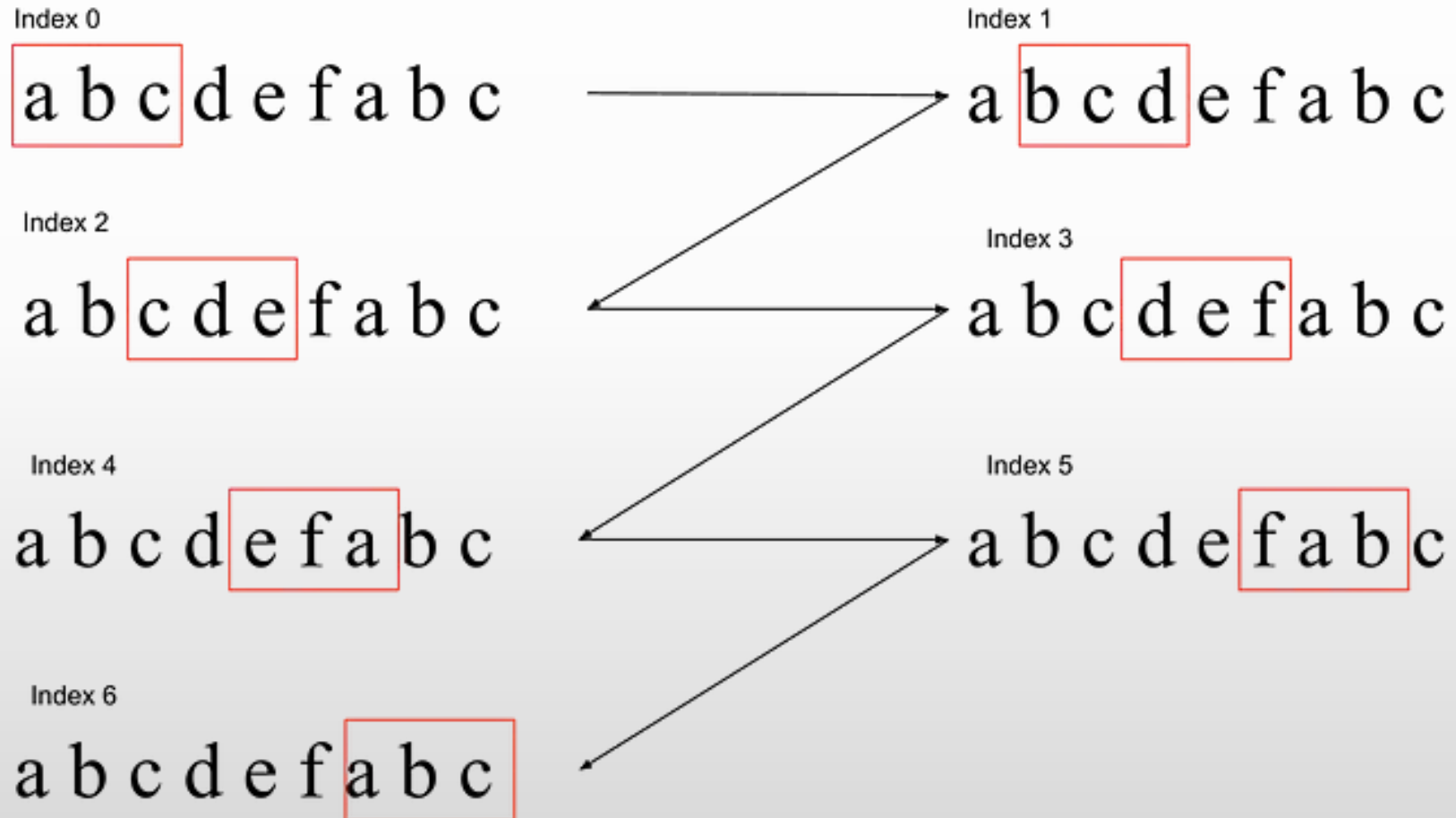
T S

2 + 3 = 5

U = 1
T = 2
S = 3
A = 4
V = 5



Rabin Karp



String Compression





Sıralı Sıkıştırma Kodlaması (RLE)

- RLE, metin verilerini sıkıştırmak için kullanılır.
- Ardışık tekrar eden karakterleri bir kodla yer değiştirir.
- Metin sıkıştırmada kullanılan basit ve etkili bir yöntemdir.
- Tekrar eden karakterlerin sayısı az ise etkisiz olabilir.



RLE Sıkıştırma

- Sıkıştırma Süreci:
 - Metin: "AAAAABBBCCCCDDD"
 - Sıkıştırılmış Hali: "5A3B4C3D"
 - Ardışık tekrar eden karakterlerin sayısı ve kendisi ile temsil edilmesi.



Sıkıştırma Adımları

- Metin soldan sađa taranır.
- Ardışık tekrar eden karakterler sayılır.
- Tekrar eden karakterler için sayı ve karakter bir araya getirilir.
- Bu bilgiler sıkıştırılmış metin olarak saklanır.



Algoritma Karmaşıklığı

- RLE algoritması, girdi metni tek bir kez tarar.
- Ardışık tekrar eden karakterlerin sayısını kaydeder.
- Bu nedenle, algoritmanın zaman karmaşıklığı, $O(n)$.

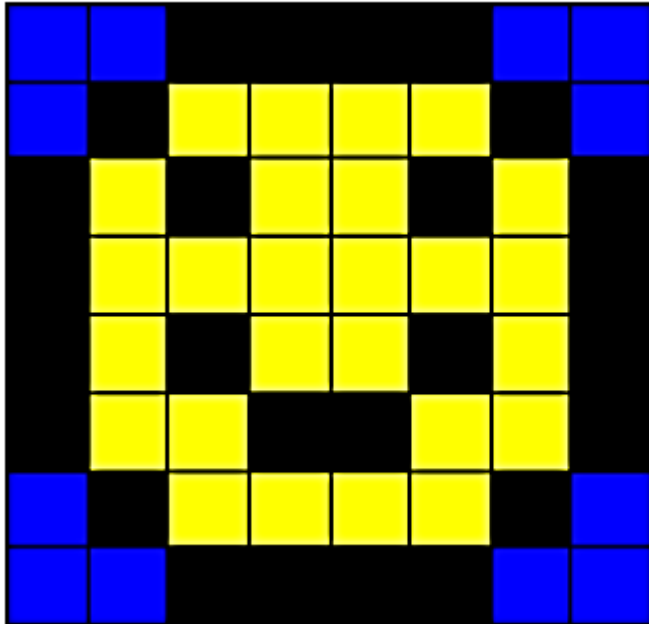



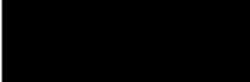

RLE Sıkıştırma

W	B	1	W	3	B	1	W	1
B	3	W	1	B	3			
B	7							
W	1	B	5	W	1			
W	2	B	3	W	2			
W	3	B	1	W	3			



Pixel level RLE



RGB colour values			
	R	G	B
	0	0	255
	0	0	0
	255	255	0



Pixel level RLE

Count	R	G	B	Count	R	G	B	Count	R	G	B
2	0	0	255	4	0	0	0	3	0	0	255
1	0	0	0	4	255	255	0	1	0	0	0
1	0	0	255	1	0	0	0	1	255	255	0
1	0	0	0	2	255	255	0	1	0	0	0
1	255	255	0	2	0	0	0	6	255	255	0
2	0	0	0	1	255	255	0	1	0	0	0
2	255	255	0	1	0	0	0	1	255	255	0
2	0	0	0	2	255	255	0	2	0	0	0
2	255	255	0	1	0	0	0	1	0	0	255
1	0	0	0	4	255	255	0	1	0	0	0
3	0	0	255	4	0	0	0	2	0	0	255





Lempel-Ziv-Welch (LZW) Sıkıştırma Algoritması

- LZW, metin sıkıştırma için kullanılan etkili bir algoritmadır.
- Tekrar eden dizgileri tanımlayarak sıkıştırma yapar.
- Metin sıkıştırmada yaygın olarak kullanılır.
- Sözlük boyutu büyüdükçe performansı düşer.



LZW Sıkıştırma

- Sıkıştırma Süreci:
 - Metin: "ABABCABABCA"
 - Sıkıştırılmış Hali: "0 1 2 4 3 5"
 - Metin içindeki tekrar eden dizgilerin bir indeksle temsil edilmesi.



Sıkıştırma Adımları

- Metin soldan sađa taranır.
- İndekslerle tanımlanan dizgilerin bulunduğu bir sözlük oluşturulur.
- Sözlükte olmayan yeni dizgiler eklenir.
- Dizgiler sıkıştırılmış metin olarak depolanır.



Lempel-Ziv-Welch

A T G A T C A T G A G

code word out

0	A	
1	T	
2	G	
3	C	
4	AT	0
5	TG	1
6	GA	2
7	ATC	4
8	CA	3
9	ATG	4
10	GAG	6
		2



Lempel-Ziv-Welch

	Output	Dict.													
<table border="1"><tr><td>a</td><td>a</td><td>b</td><td>a</td><td>a</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>b</td></tr></table>	a	a	b	a	a	c	a	b	c	a	b	c	b	(0, a)	1 = a
a	a	b	a	a	c	a	b	c	a	b	c	b			
<table border="1"><tr><td>a</td><td>a</td><td>b</td><td>a</td><td>a</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>b</td></tr></table>	a	a	b	a	a	c	a	b	c	a	b	c	b	(1, b)	2 = ab
a	a	b	a	a	c	a	b	c	a	b	c	b			
<table border="1"><tr><td>a</td><td>a</td><td>b</td><td>a</td><td>a</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>b</td></tr></table>	a	a	b	a	a	c	a	b	c	a	b	c	b	(1, a)	3 = aa
a	a	b	a	a	c	a	b	c	a	b	c	b			
<table border="1"><tr><td>a</td><td>a</td><td>b</td><td>a</td><td>a</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>b</td></tr></table>	a	a	b	a	a	c	a	b	c	a	b	c	b	(0, c)	4 = c
a	a	b	a	a	c	a	b	c	a	b	c	b			
<table border="1"><tr><td>a</td><td>a</td><td>b</td><td>a</td><td>a</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>b</td></tr></table>	a	a	b	a	a	c	a	b	c	a	b	c	b	(2, c)	5 = abc
a	a	b	a	a	c	a	b	c	a	b	c	b			
<table border="1"><tr><td>a</td><td>a</td><td>b</td><td>a</td><td>a</td><td>c</td><td>a</td><td>b</td><td>c</td><td>a</td><td>b</td><td>c</td><td>b</td></tr></table>	a	a	b	a	a	c	a	b	c	a	b	c	b	(5, b)	6 = abcb
a	a	b	a	a	c	a	b	c	a	b	c	b			



Lempel-Ziv-Welch

	Output	Dict.
a a b a a c a b a b a c b	112	256=aa
a a b a a c a b a b a c b	112	257=ab
a a b a a c a b a b a c b	113	258=ba
a a b a a c a b a b a c b	256	259=aac
a a b a a c a b a b a c b	114	260=ca
a a b a a c a b a b a c b	257	261=aba
a a b a a c a b a b a c b	261	262=abac
a a b a a c a b a b a c b	114	263=cb

String Sorting





Alfabetik Sıralama

- Bir dizi öđeyi (kelimeler, sayılar, vb.) belirli bir düzene göre sıralar.
- Karakterlerin alfabedeki veya sayısal düzende pozisyonlarına dayanır.
- "apple" kelimesi "banana" kelimesinden önce gelir,
 - çünkü "a" harfi "b" harfinden önce gelir.
- 123, 45, 6, 789 gibi rakamlar, soldan sağa doğru sıralanır.

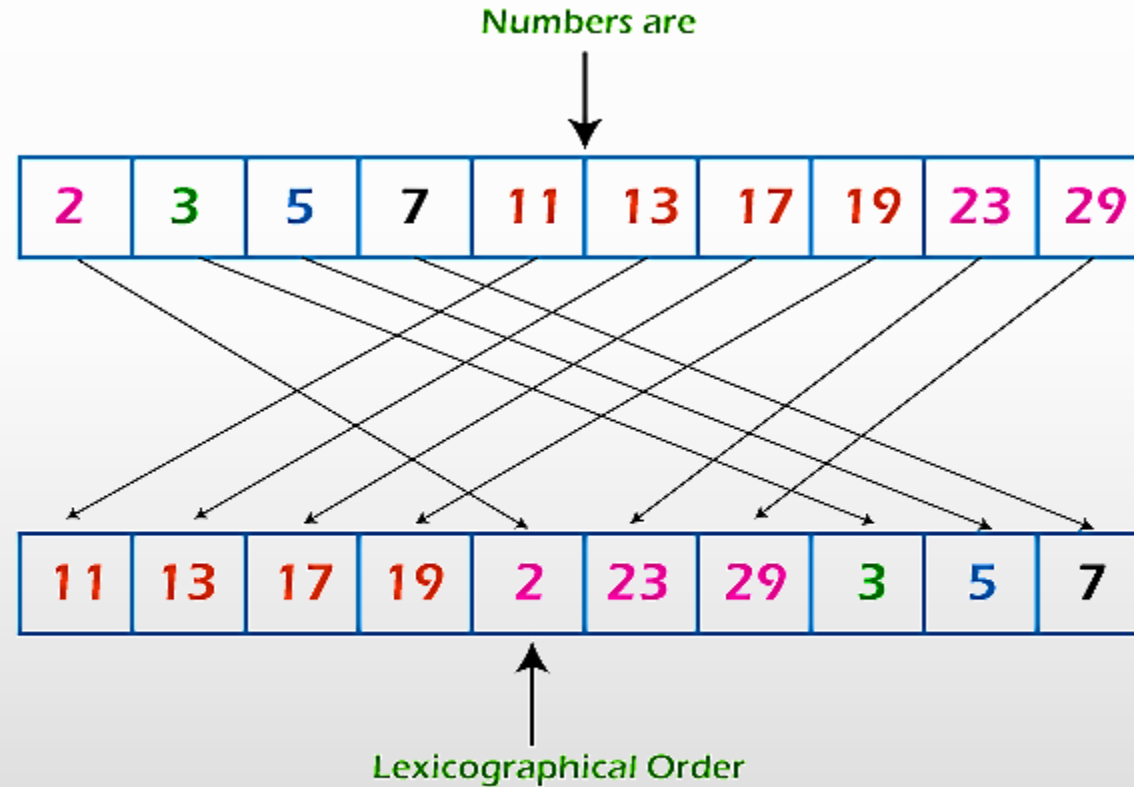


Özellikler

- İlk karakterlerin karşılaştırılmasıyla başlar.
- Eğer ilk karakterler eşitse, bir sonraki karakterlere bakılır.
- Bu işlem öğelerin tamamı karşılaştırılana kadar devam eder.



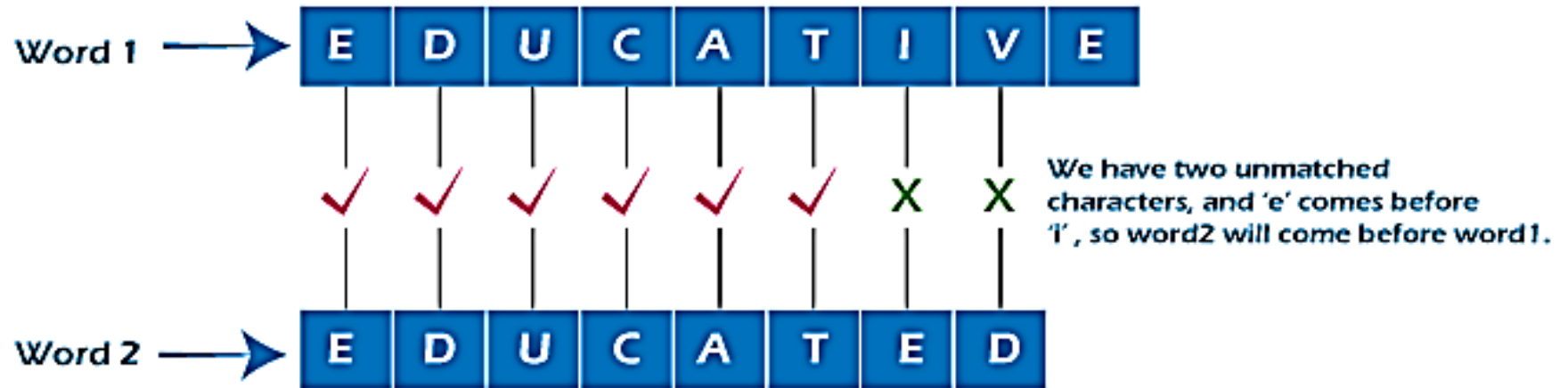
Alfabetik Sıralama



Lexicographical Order representation (with numbers comparison)



Alfabetik Sıralama



Lexicographical Order → **{ Educated, Educative }**

Lexicographical order representation, with a comparison



Örnek

- Diziler:
 - *apple* ve *apricot*
- İlk Karakterler:
 - *a* ve *a* eşit.
- İkinci Karakterler:
 - *p* ve *p* eşit.
- Üçüncü Karakterler:
 - *p* ve *r* karşılaştırılır.
 - ASCII değeri $p (112) < r (114)$,
 - *apple* *apricot*'tan önce gelir.





Taban Sıralama (Radix Sort)

- Her bir karakteri basamak değeri gibi kullanarak sıralar.
- En uzun dizginin uzunluğu bulunur.
- Sağdan başlayarak her bir karakter basamak olarak ele alınır.
- Sıralama işlemi en önemli basamaktan az önemli basamağa doğru yapılır.
- Her bir basamakta, öğeler alfabetik olarak sıralanır.

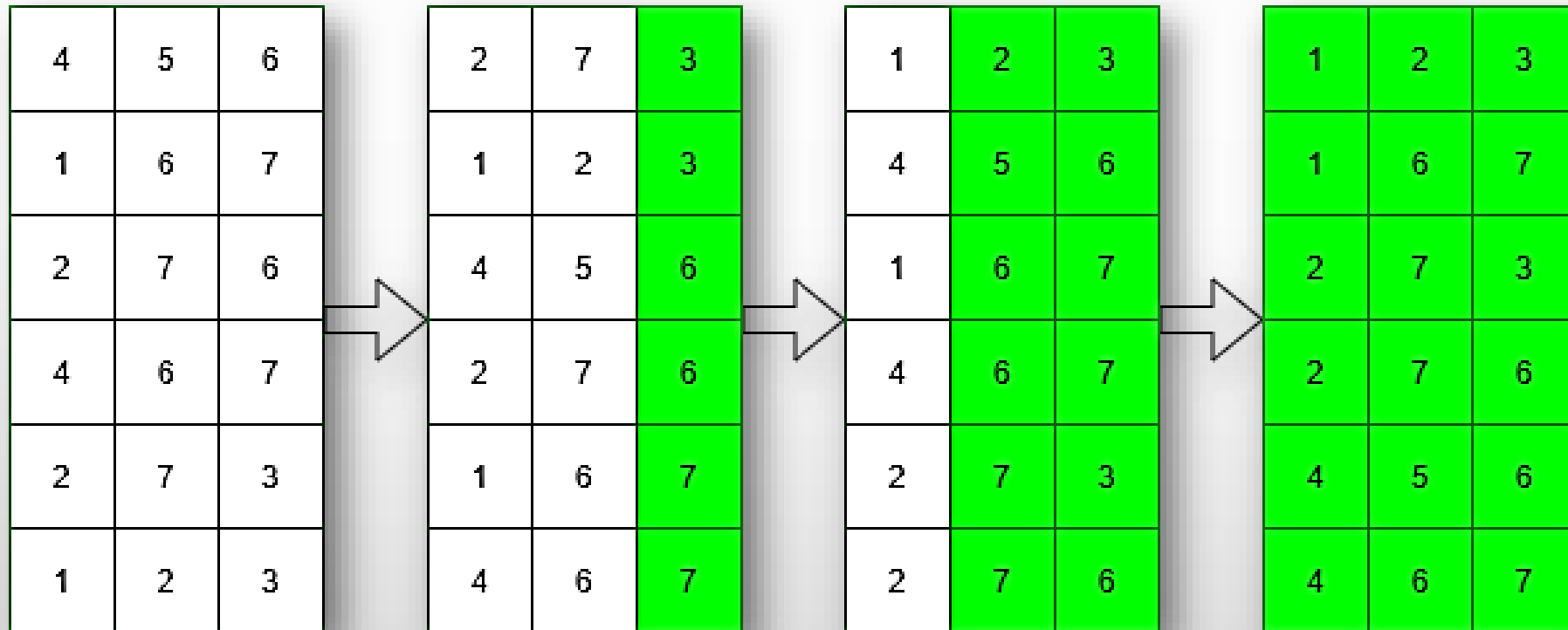


Zaman Karmaşıklığı

- En uzun dizginin uzunluğu $(n) * \text{Eleman sayısı } (N)$
- $O(n N)$



Radix Sort





Radix Sort

B	A	D	G	E	∅					
B	A	N	N	E	R	∅				
C	O	F	F	E	∅					
C	O	M	P	A	R	I	S	O	N	∅
C	O	M	P	U	T	E	R	∅		
M	I	D	N	I	G	H	T	∅		
W	A	N	D	E	R	∅				
W	A	R	D	R	O	B	E	∅		
W	O	R	K	E	R	∅				



Örnek

- Veri Kümesi: [170, 45, 75, 90, 802, 24, 2, 66]
- En Az Anlamlı Basamağa Göre:
 - $170, 45, 75, 90, 802, 24, 2, 66 \rightarrow (0, 5, 5, 0, 2, 4, 2, 6)$
 - [170, 90, 802, 2, 24, 45, 75, 66]
- İkinci Basamağa Göre:
 - $170, 90, 802, 2, 24, 45, 75, 66 \rightarrow (7, 9, 0, 0, 2, 4, 7, 6)$
 - [802, 2, 24, 45, 66, 170, 75, 90]
- En Anlamlı Basamağa Göre:
 - $802, 2, 24, 45, 66, 170, 75, 90 \rightarrow (8, 0, 0, 0, 0, 1, 0, 0)$
 - [2, 24, 45, 66, 75, 90, 170, 802]

String Parsing





Düzenli İfadeler (Regular Expressions)

- Düzenli ifadeler, metinlerde belirli örüntüleri tanımlamak ve bu örüntülere uyan kısımları eşleştirmek için kullanılır.
- Metin manipülasyonlarını gerçekleştirmek için yaygın kullanılır.
- Karmaşık metin işlemlerini otomatikleştirir.



Temel Operatörler ve Kavramlar

- `.`: Herhangi bir tek karakteri temsil eder.
- `*`: Önceki karakterin sıfır veya daha fazla tekrarını temsil eder.
- `+`: Önceki karakterin bir veya daha fazla tekrarını temsil eder.
- `?`: Önceki karakterin sıfır veya bir kez tekrarını temsil eder.
- `[]`: Belirli karakterlerin bir kümesini temsil eder.
- `^`: Belirtilen örüntünün dizginin başında olmasını sağlar.
- `$`: Belirtilen örüntünün dizginin sonunda olmasını sağlar.
- `()`: Örüntülerin gruplandırılmasını sağlar ve alt ifadeleri tanımlar.



Karmaşık Operatörler ve Kavramlar

- { }: Belirli bir sayıda tekrarın belirtilmesini sağlar.
- |: Alternatif örüntüler arasında seçim yapmayı sağlar.
- \b: Kelimenin başı veya sonu gibi belirli sınırları tanımlar.
- \d: Bir rakamı tanımlar.
- \w: Bir kelimeyi tanımlar.
- \s: Boşluk karakterini tanımlar.



Düzenli İfadelerin Temelleri

- Karakter ve Metin Eşleşmesi:
 - a: Tek bir 'a' karakteriyle eşleşir.
 - abc: "abc" ile tam olarak eşleşir.
- Özel Karakterler:
 - .: Yeni satır hariç herhangi bir karakterle eşleşir.
 - \d: Bir rakam ile eşleşir (0-9).
 - \w: Bir kelime ile eşleşir (alfanümerik + alt çizgi).
 - \s: Bir boşluk ile eşleşir (boşluk, tab, yeni satır vb.).



Düzenli İfade Karakter Sınıfları

- Karakter Sınıfları:
 - [abc]: 'a', 'b' veya 'c' karakterlerinden biriyle eşleşir.
 - [a-z]: Küçük harflerden herhangi biriyle eşleşir.
 - [A-Z]: Büyük harflerden herhangi biriyle eşleşir.
 - [0-9]: Rakamlarla eşleşir.
- Tümleyen Karakter Sınıfları:
 - [^abc]: 'a', 'b' veya 'c' dışındaki herhangi bir karakterle eşleşir.
 - [^0-9]: Rakamlar dışındaki herhangi bir karakterle eşleşir.



Yinelenen Karakter ve Sayılar

- a^* : Sıfır veya daha fazla 'a' karakteriyle eşleşir.
- a^+ : Bir veya daha fazla 'a' karakteriyle eşleşir.
- $a^?$: Sıfır veya bir 'a' karakteriyle eşleşir.
- $a\{3\}$: Tam olarak üç 'a' karakteriyle eşleşir.
- $a\{2,4\}$: İki ile dört arasında 'a' karakteriyle eşleşir.



Gruplama

- (abc): "abc" ile tam olarak eşleşir ve gruplar.
- (a|b|c): 'a', 'b' veya 'c' karakterleriyle eşleşir ve gruplar.

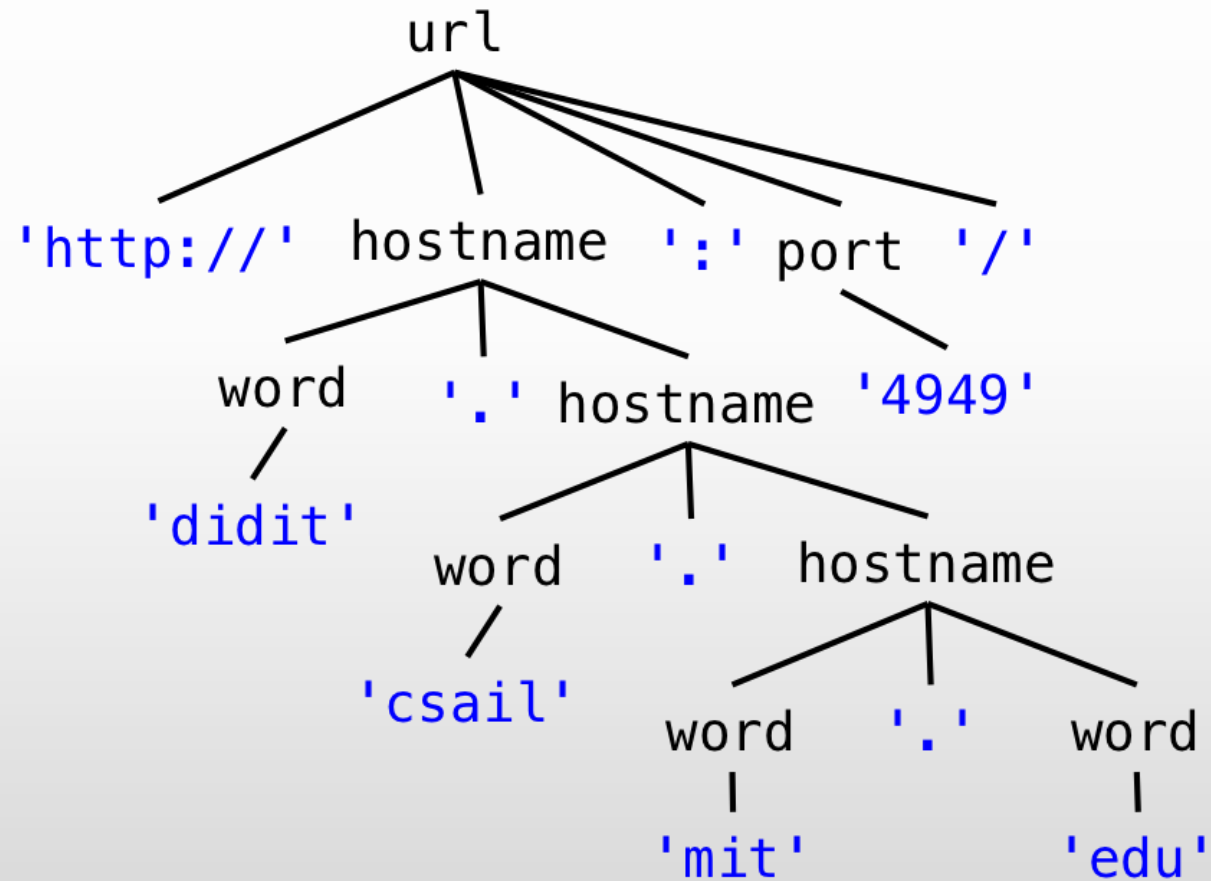


Lookahead ve Lookbehind

- $a(?=b)$: 'a' karakteri, 'b' karakterinden önce gelirse eşleşir.
- $a(?!b)$: 'a' karakteri, 'b' karakterinden önce gelmezse eşleşir.
- $(?<=b)a$: 'a' karakteri, 'b' karakterinden sonra gelirse eşleşir.
- $(?<!b)a$: 'a' karakteri, 'b' karakterinden sonra gelmezse eşleşir.



Düzenli İfadeler (Regular Expressions)





Örnek

- E-posta Doğrulama:
 - $^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$$
- Telefon Numarası:
 - $^(\\+\\d{1,3}[-]?)?\\d{10}$$
- URL:
 - $^https?:\\/\\/[^\\s/$. ?#].[^\\s]*$$





Sonlu Durum Makineleri (Finite State Machines)

- Bir dizgiyi işlemek için kullanılan matematiksel modeldir.
- Bir durum kümesi;
 - başlangıç durumu,
 - girdi alfabesi ve
 - durumlar arasındaki geçişlerin kümesinden oluşur.
- Basit ve anlaşılması kolay bir modeldir.
- Karmaşık dizgi analizi problemlerini ele alabilir.



Temel Bileşenler

- Durumlar (*States*):
 - Makinenin bulunabileceği farklı durumlar.
- Alfabe (*Alphabet*):
 - FSM'nin kabul ettiği girdi sembolleri kümesi.
- Başlangıç Durumu (*Start State*):
 - İşleme başlamak için seçilen durum.
- Geçişler (*Transitions*):
 - Girdiye göre durumlar arasındaki geçişler.
- Son Durum (*Final States*):
 - Dizgi işlendiğinde olunabilecek son durumlar.



FSM Türleri

- *Deterministic Finite Automaton (DFA):*
 - Her durum ve girdi sembolü çifti için yalnızca bir geçiş tanımlıdır.
- *Non-Deterministic Finite Automaton (NFA):*
 - Bir durum ve girdi sembolü çifti için birden fazla geçiş tanımlıdır.
- *Epsilon-NFA (ϵ -NFA):*
 - Boş geçişlerin (epsilon geçişleri) mümkün olduğu NFA türü.

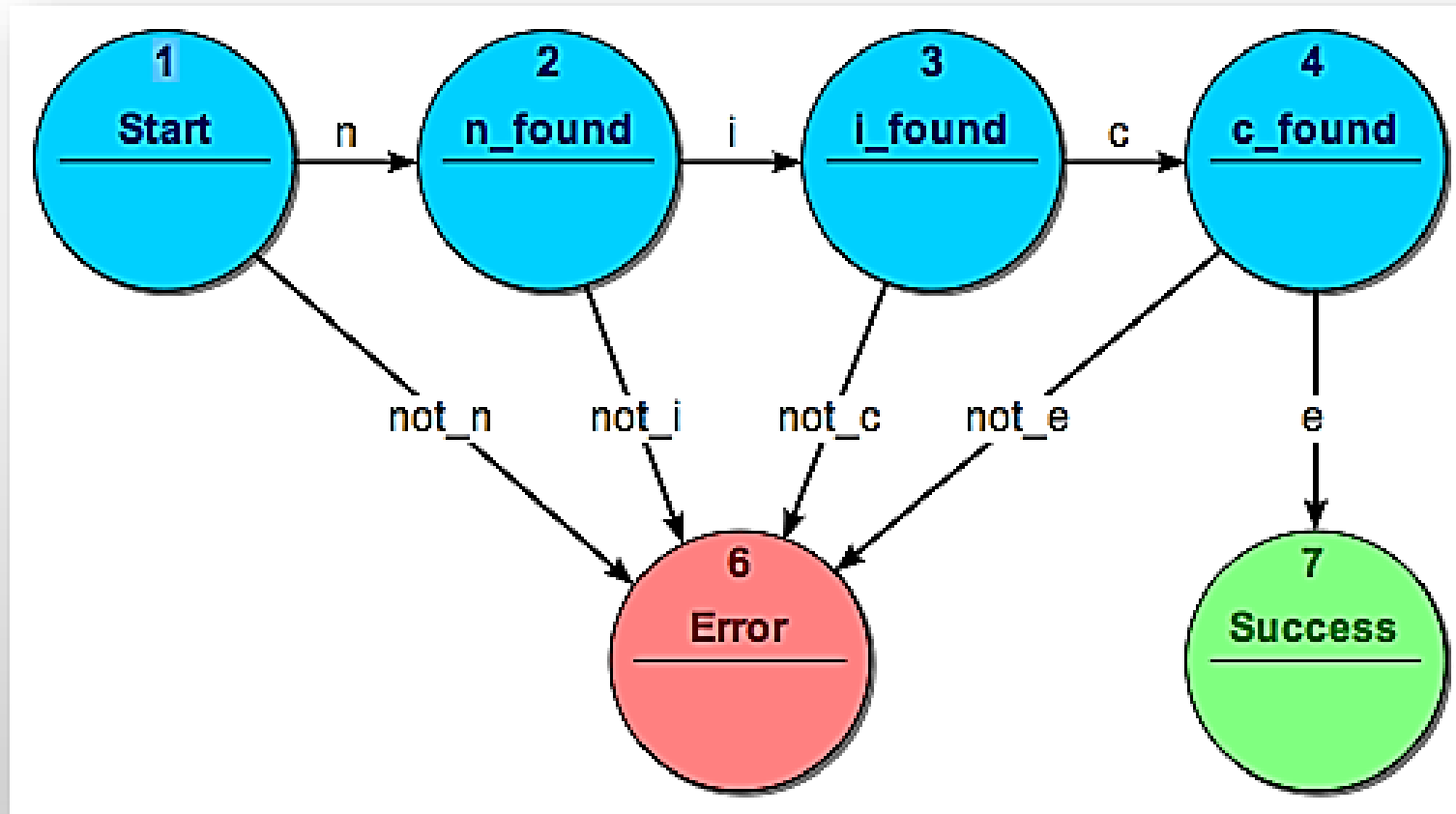


Kullanımı

- Başlangıç durumuna geçiş.
- Her bir karakter için durumların güncellenmesi.
- Dizgi tamamlandığında son durumun kontrol edilmesi.

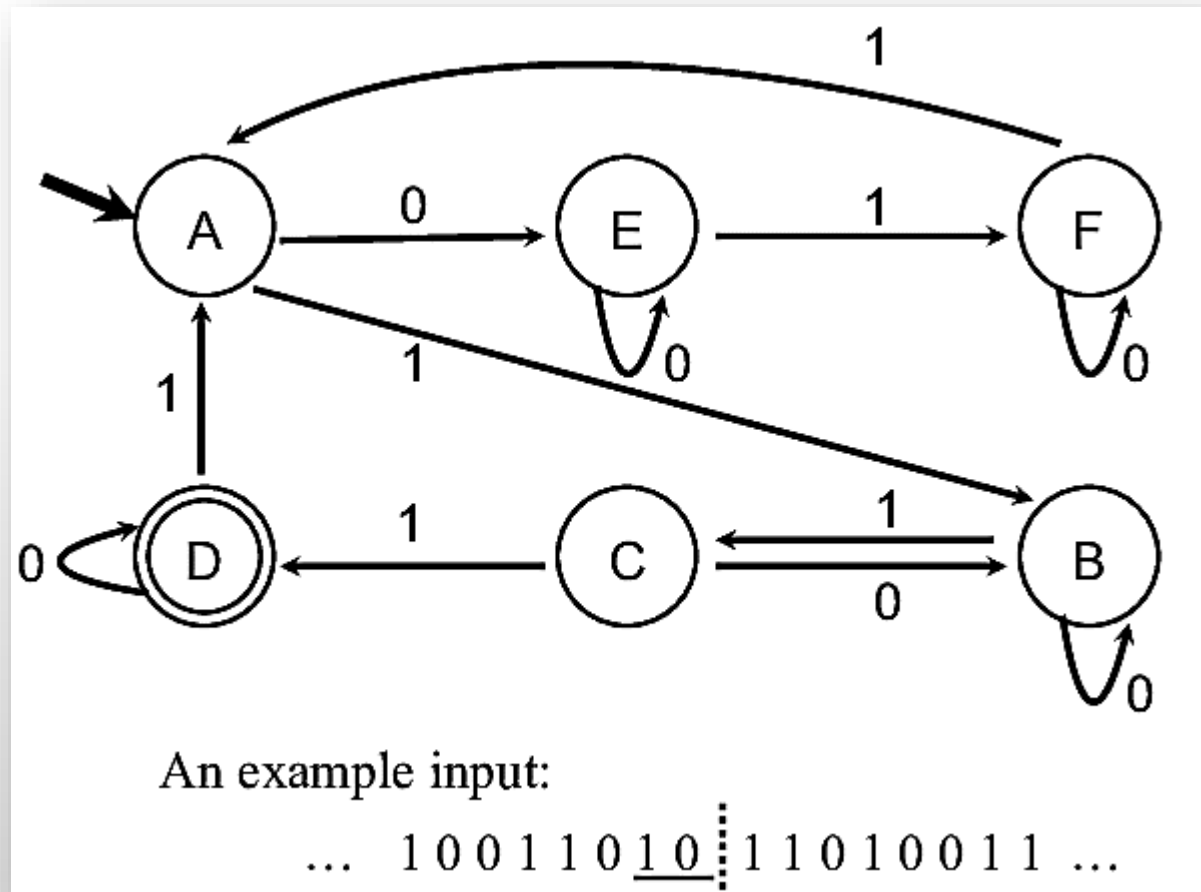


Sonlu Durum Makineleri



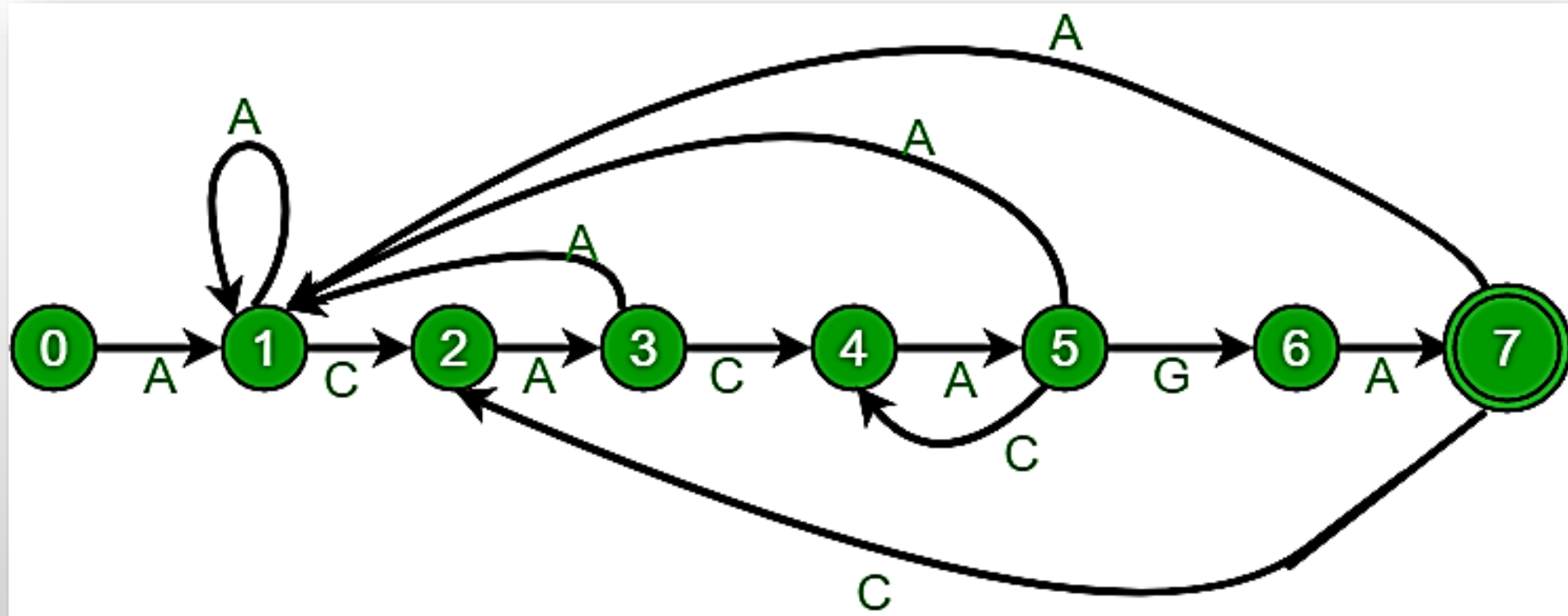


Sonlu Durum Makineleri





Sonlu Durum Makineleri



String Similarity





Levenshtein Mesafesi

- İki dizgi arasındaki farkı nicel olarak ölçen etkili bir metriktir.
- Otomatik düzeltme ve tahmin sistemlerinde kullanılır.
- İki dizgi arasındaki minimum işlem (ekleme, çıkarma veya değiştirme) sayısını belirtir.



Temel İşlemler

- Ekleme (*Insertion*):
 - Bir karakterin eklenmesi.
- Çıkarma (*Deletion*):
 - Bir karakterin çıkarılması.
- Değişirme (*Substitution*):
 - Bir karakterin başka bir karakterle değiştirilmesi.



Levenshtein Mesafesi Hesaplama

- İki dizgi arasındaki minimum düzenleme işlemi sayısı olarak hesaplanır.
- Dinamik programlama yöntemiyle hesaplanır.
- İki dizgi arasındaki karakterlerin karşılaştırılması ve işlem maliyetlerinin belirlenmesi ile yapılır.

Levenshtein



H	O		N	D	A	
H	Y	U	N	D	A	I

H	Y	U	N	D	A	I
H		O	N	D	A	

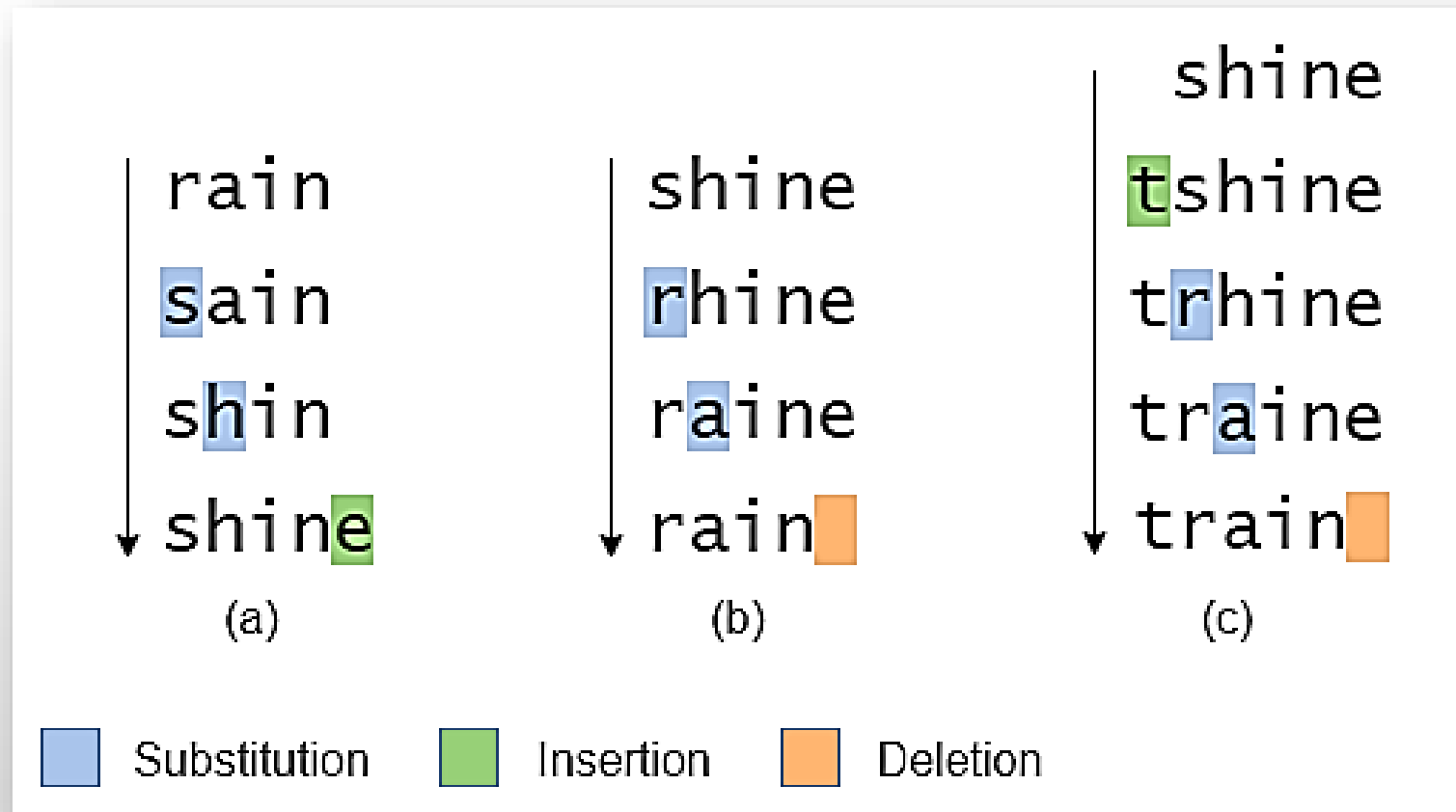
Levenshtein



	""	P	A	I	R	S
""	0	1	2	3	4	5
C	1	1	2	3	4	5
A	2	2	1	2	3	4
R	3	3	2	2	2	3
S	4	4	3	3	3	2



Levenshtein







En Uzun Ortak Alt Dizgi (LCS)

- İki dizgi içinde sıralı olarak bulunan ve mümkün olan en uzun dizgidir.
- Bu alt dizgi, dizgilerin karakter sırasını bozmaz ancak ardışık olmak zorunda değildir.
- İki dizgi arasındaki benzerlik seviyesini ölçer.
- Dinamik programlama yöntemiyle hesaplanır.
- İki dizgi arasındaki karakterlerin sıralı şekilde eşleştirilmesiyle elde edilir.



Örnek

- Dizi 1: AGGTAB
- Dizi 2: GXTXAYB
- LCS: GTAB



Dinamik Programlama Yaklaşımı

- İki boyutlu bir tablo oluşturulur.
- Her hücre, alt dizinin o noktaya kadar olan LCS uzunluğunu temsil eder.
- Tablonun sonunda LCS uzunluğu bulunur.



Algoritma

- İki dizinin uzunlukları m ve n olsun.
- $(m+1) \times (n+1)$ boyutunda bir tablo oluşturulur.
- İlk satır ve sütun sıfır ile doldurulur.
- $A[i] == B[j]$ ise, hücre değeri üst-sol köşedeki değerin 1 fazlasıdır.
- $A[i] != B[j]$ ise, hücre değeri üst veya sol hücrenin maksimum değeri olur.
- Tablonun son hücresi, LCS uzunluğunu verir.



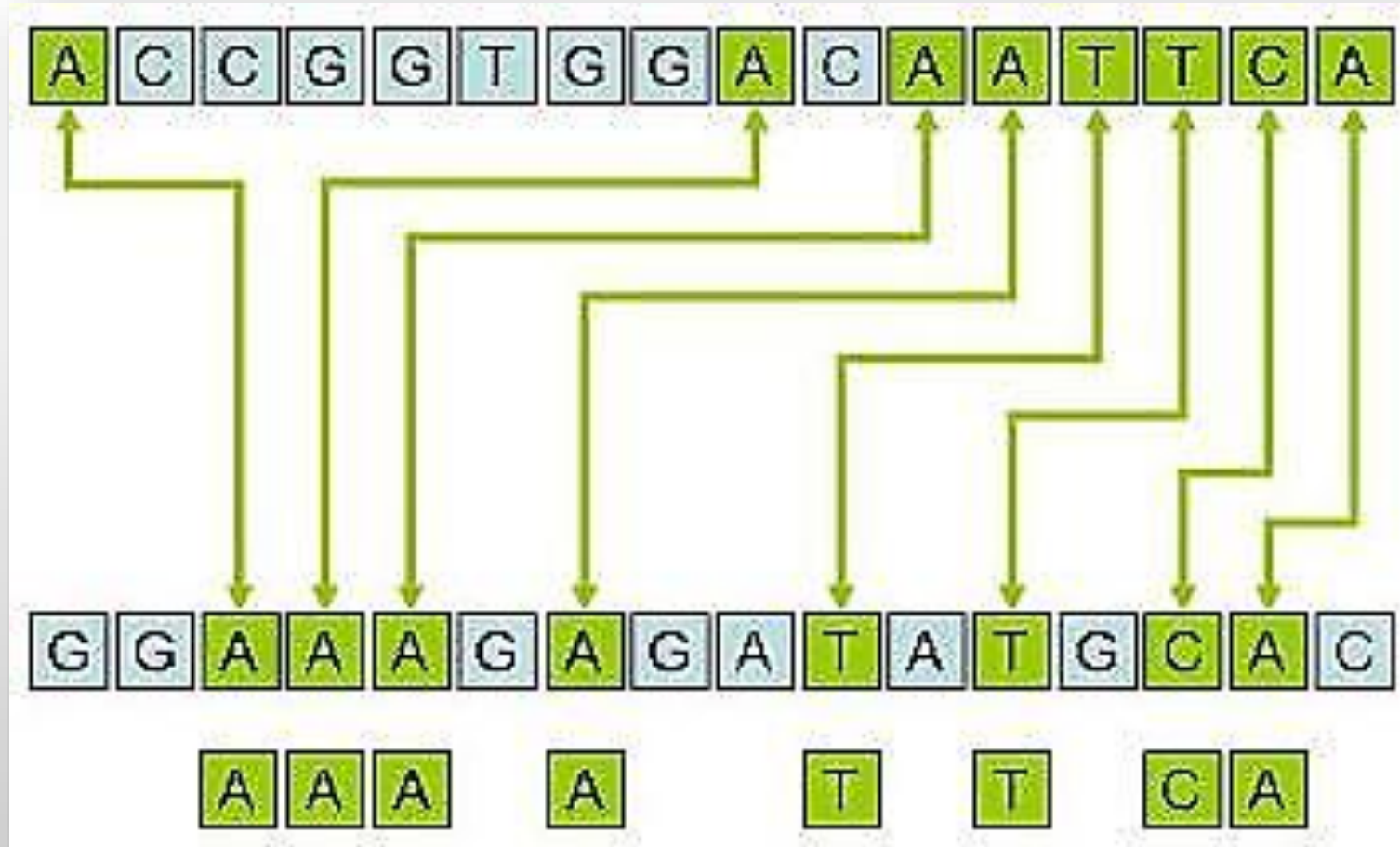
En Uzun Ortak Alt Dizi (LCS)

string 1	a	c	b	a	e	d
string 2	a	b	c	a	d	f

LCS: "acad" with length 4



En Uzun Ortak Alt Dizi (LCS)



String Transformation





Sonek Dizisi (Suffix Array)

- Bir dizginin tüm soneklerinin alfabetik olarak sıralanmış halidir.
- Her bir sonek, dizginin belirli bir konumundan başlayan bir alt dizisidir.
- Dizgi arama, sıralama, genetik dizilim analizi benzeri işlemlerde kullanılır.



Oluşturma Yöntemleri

- Brute Force:
 - Tüm sonekleri oluşturup ardından sıralar.
 - $O(n^2 \log n)$ zaman karmaşıklığına sahiptir.
- Manber-Myers Algoritması:
 - Lineer zaman karmaşıklığına sahiptir. ($O(n \log n)$)
 - Soneklerin sıralanması sırasında tekrar tekrar harf karşılaştırması yapılmaz.
- Larsson-Sadakane Algoritması:
 - Sıralama işleminde art arda soneklerin karşılaştırılması esas alınır.





Burrows-Wheeler Dönüşümü (BWT)

- Girdi dizisini yeniden düzenler.
- Aynı karakterlerin bir araya toplanmasını sağlar.
- Sıkıştırma algoritmalarının performansını artırır.
- Tersine çevrilebilir: Orijinal veri, dönüşümden geri elde edilebilir.



Uygulama Adımları

- Girdi dizgisinin tüm döndürülmüş hallerini oluştur.
- Döndürülmüş dizgileri alfabetik olarak sırala.
- Sıralanmış dizgilerin son karakterlerinden yeni bir dizgi oluştur.

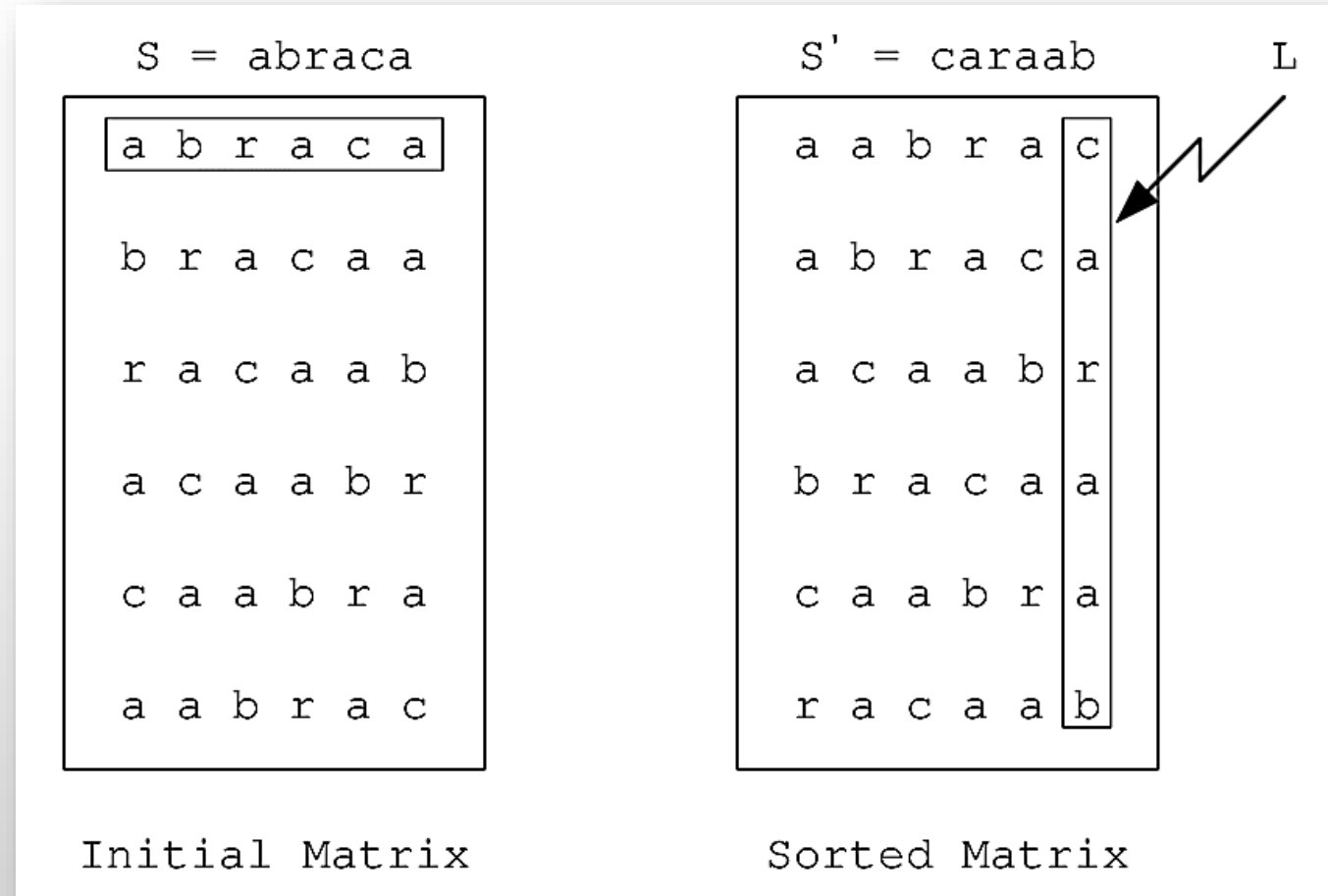


Örnek

- Girdi:
 - BANANA
- Döndürmeler:
 - BANANA, ANANAB, NANABA, ANABAN, NABANA, ABANAN
- Sıralanmış Döndürmeler:
 - ABANAN, ANABAN, ANANAB, BANANA, NABANA, NANABA
- Son Karakterler:
 - NNBAAA
- BWT Sonucu:
 - NNBAAA



Burrows-Wheeler





SON