



# **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,
  - değiştirme,
  - 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 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

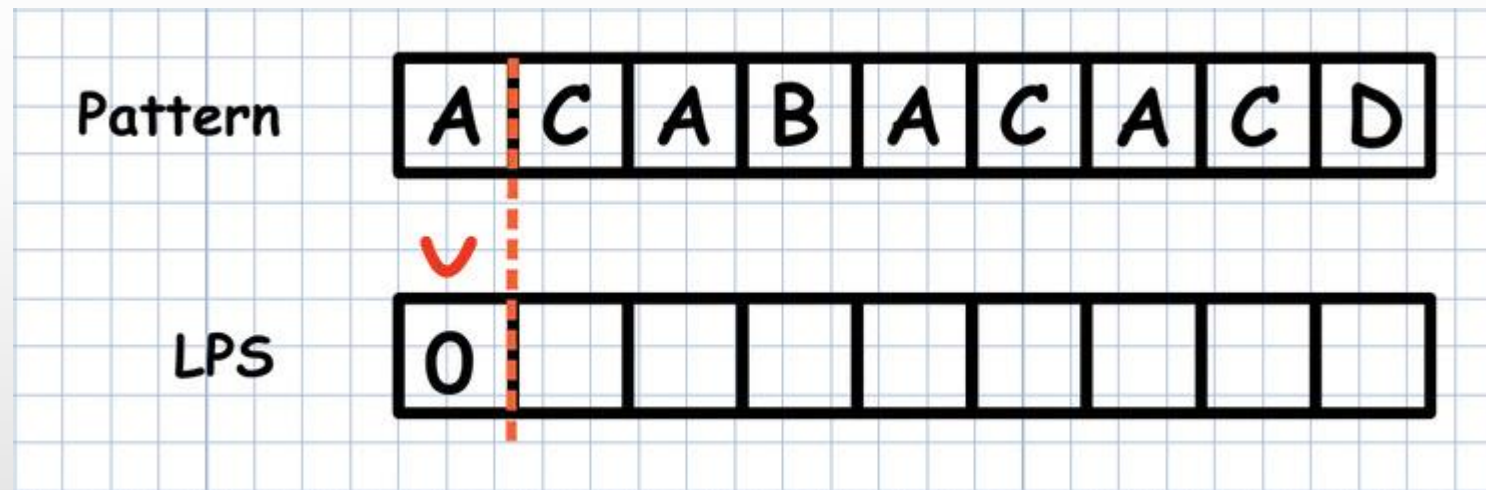


- Metin: "ababcababababd"
- Örüntü: "ababd"
- Örüntü Tablosu:
  - a: 0, b: 0, a: 1, b: 2, d: 0



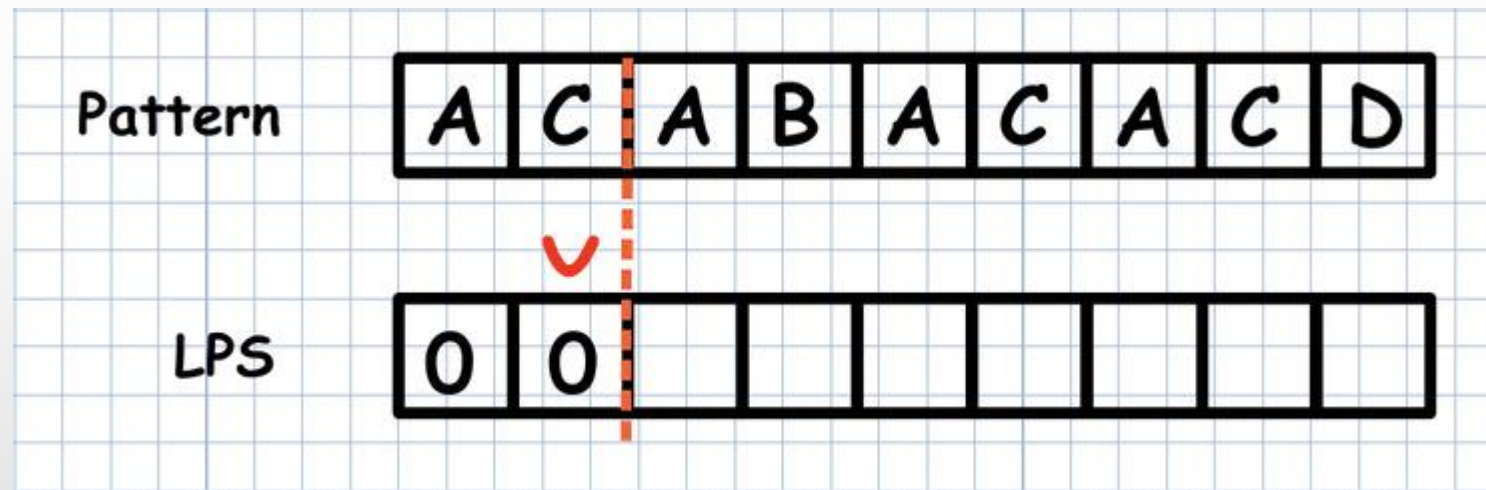
# Knuth Morris Pratt

- *Longest Proper Prefix*



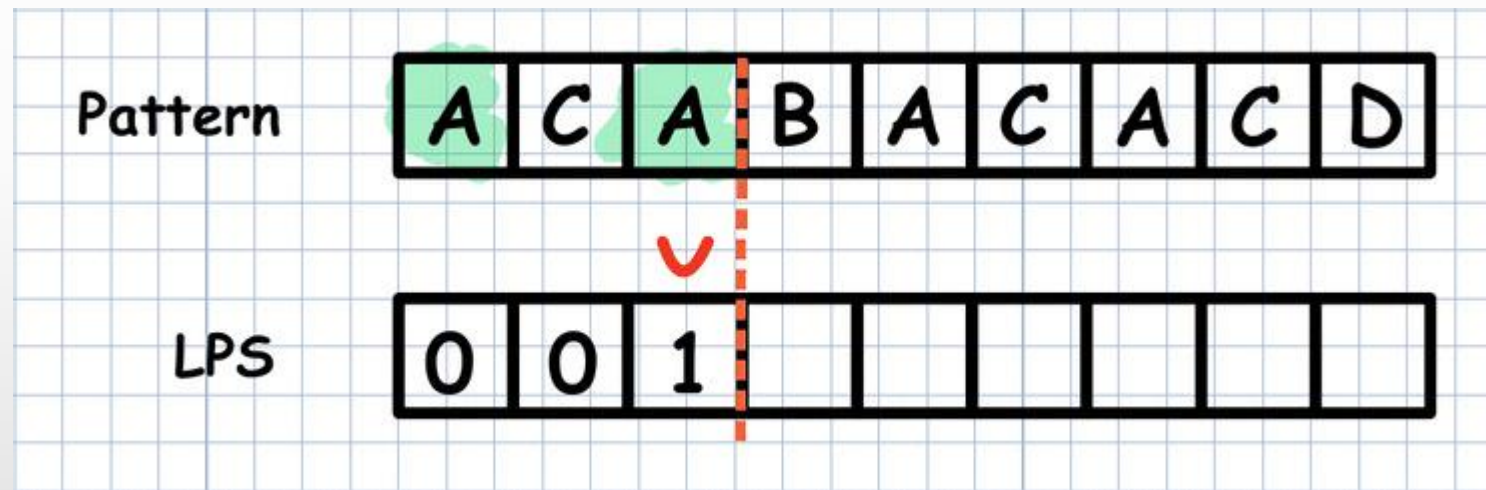


# Knuth Morris Pratt



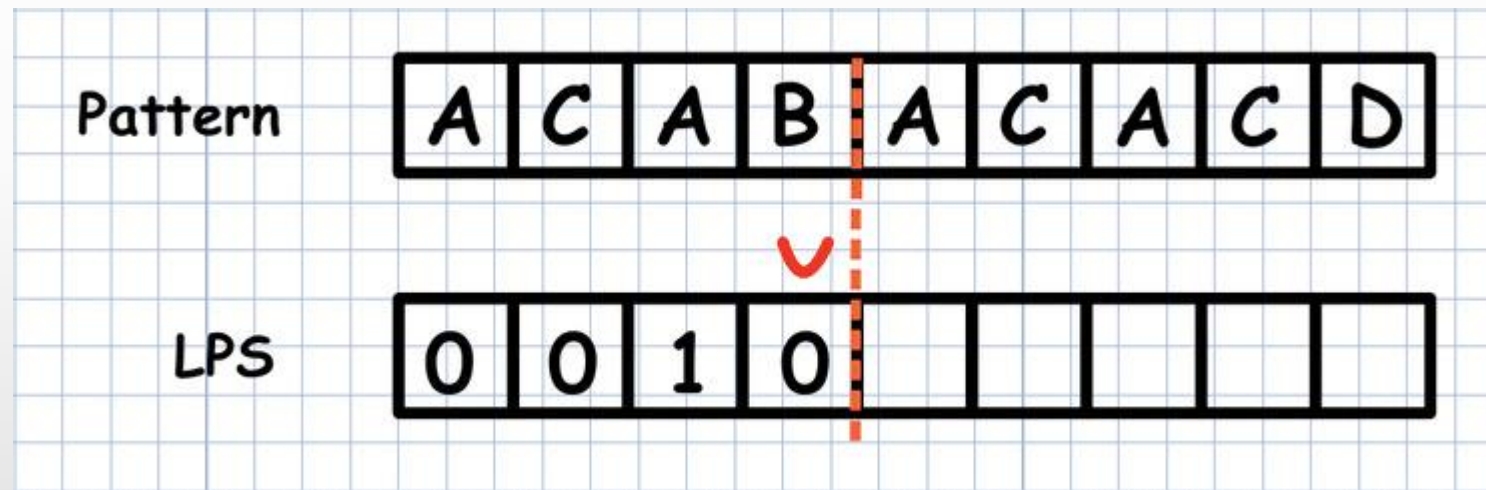


# Knuth Morris Pratt



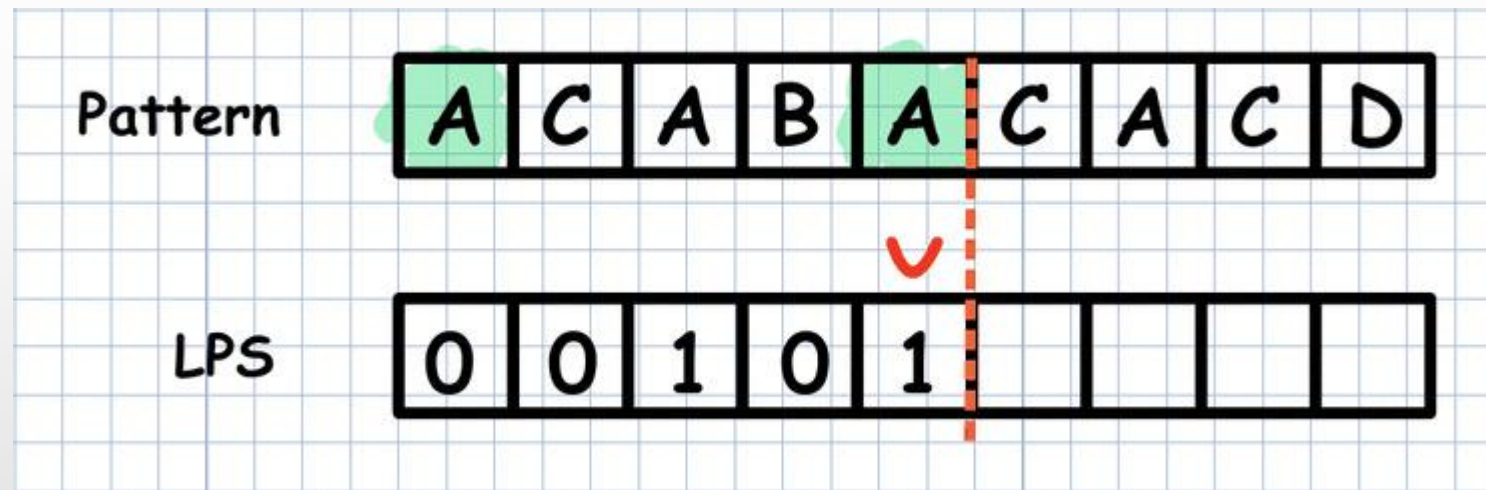


# Knuth Morris Pratt



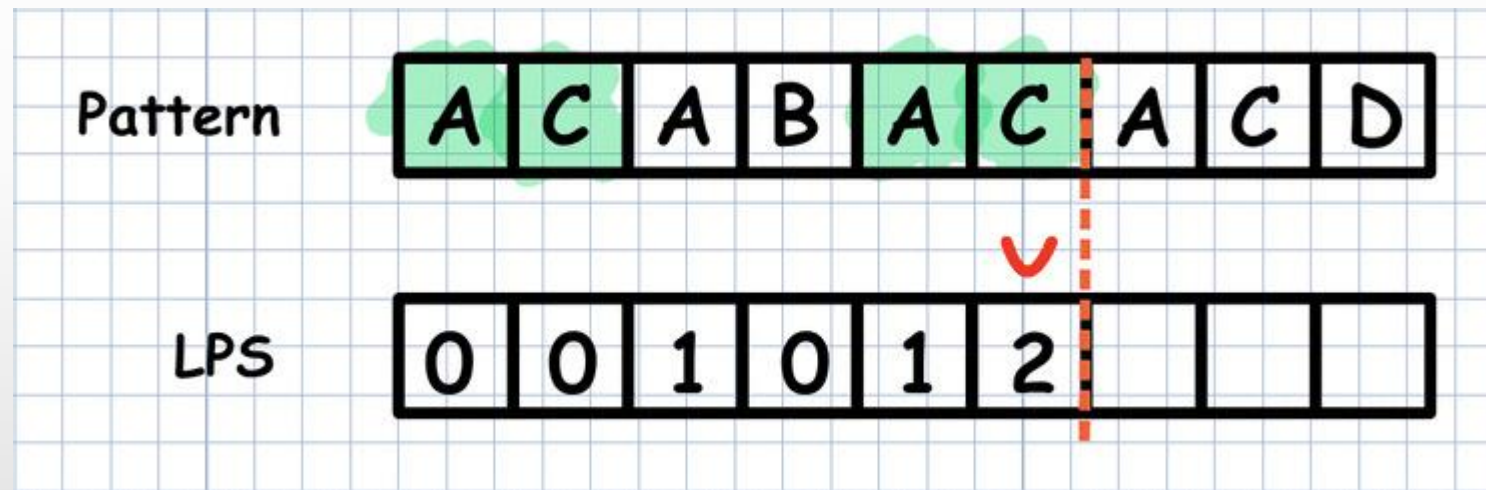


# Knuth Morris Pratt





# Knuth Morris Pratt







# Knuth Morris Pratt

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





# Knuth Morris Pratt

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

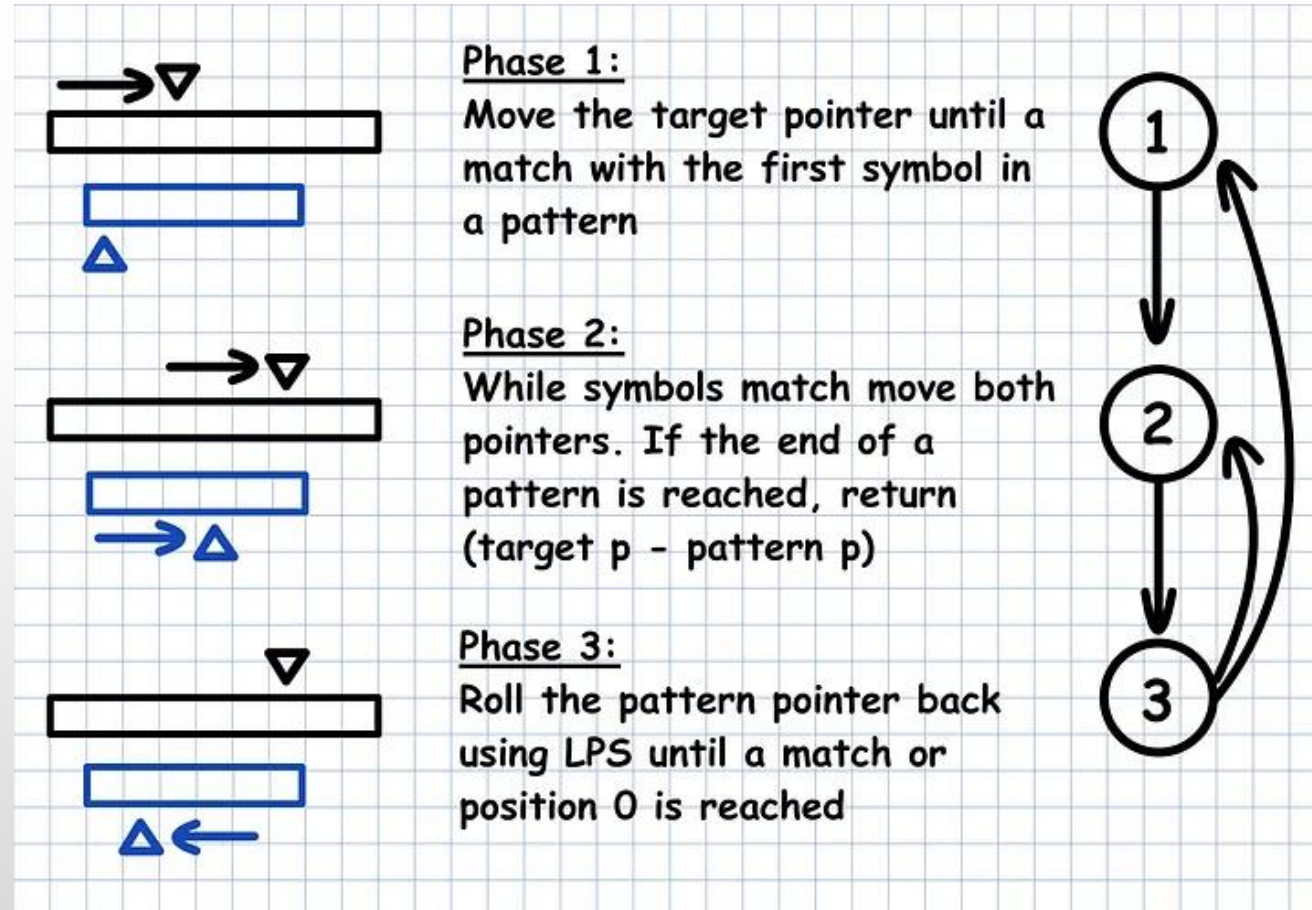


# Knuth Morris Pratt

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



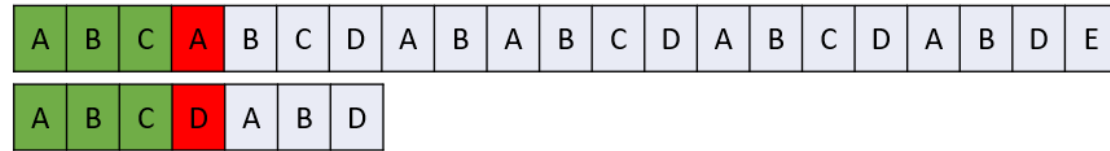
# Knuth Morris Pratt



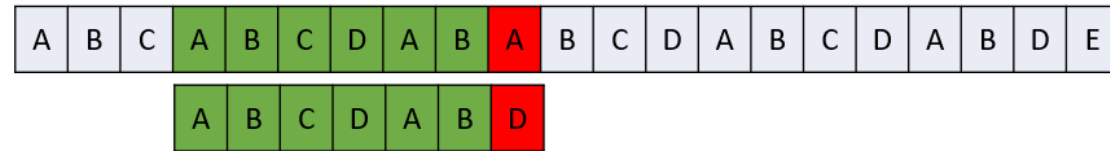


# Knuth Morris Pratt

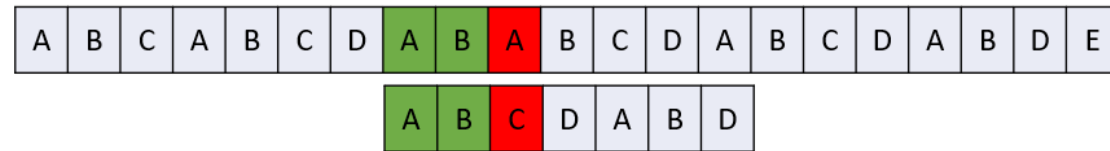
STEP 1



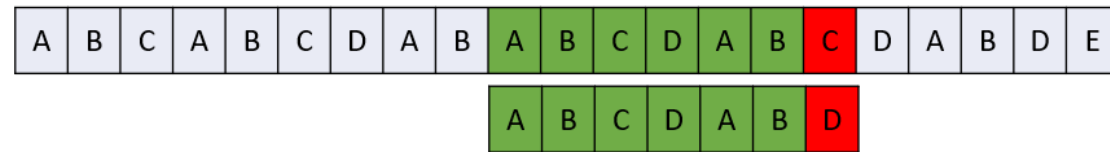
STEP 2



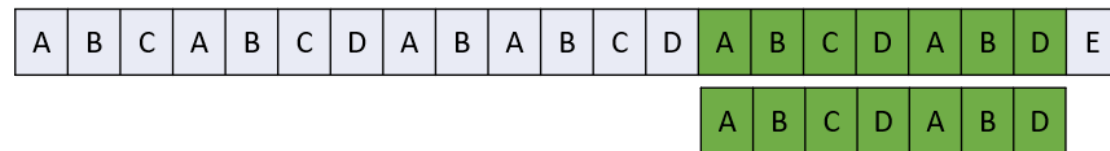
STEP 3



STEP 4



STEP 5







# 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 dönülecek pozisyonları belirleyen bir tablo oluşturulur.
- Arama: Metin içinde örüntüyü ararken, eşleşmeyen karakterlerde tablodan yararlanarak geri dönülecek 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: "abccbaabccbaabcbcabbbababcabc"
- Desen: "abcbcabbbababcabc"
- Desen Tablosu:
  - a: 10, b: 8, c: 7
- Sonuç:
  - Pozisyon 12: "abcbcabbbababcabc"



# Boyer Moore



H	E	L	L	O												
L	O		L	O	E	L	L	O		O		H	E	L	L	O

# Boyer Moore



H	E	L	L	O									
L	O	L	O	E	L	L	O	O	H	E	L	L	O

# Boyer Moore



HELLO  
LO LLOELLOHELLO

# Boyer Moore



HELLO  
LO LLOELLOHELLO



Couldn't match, move on

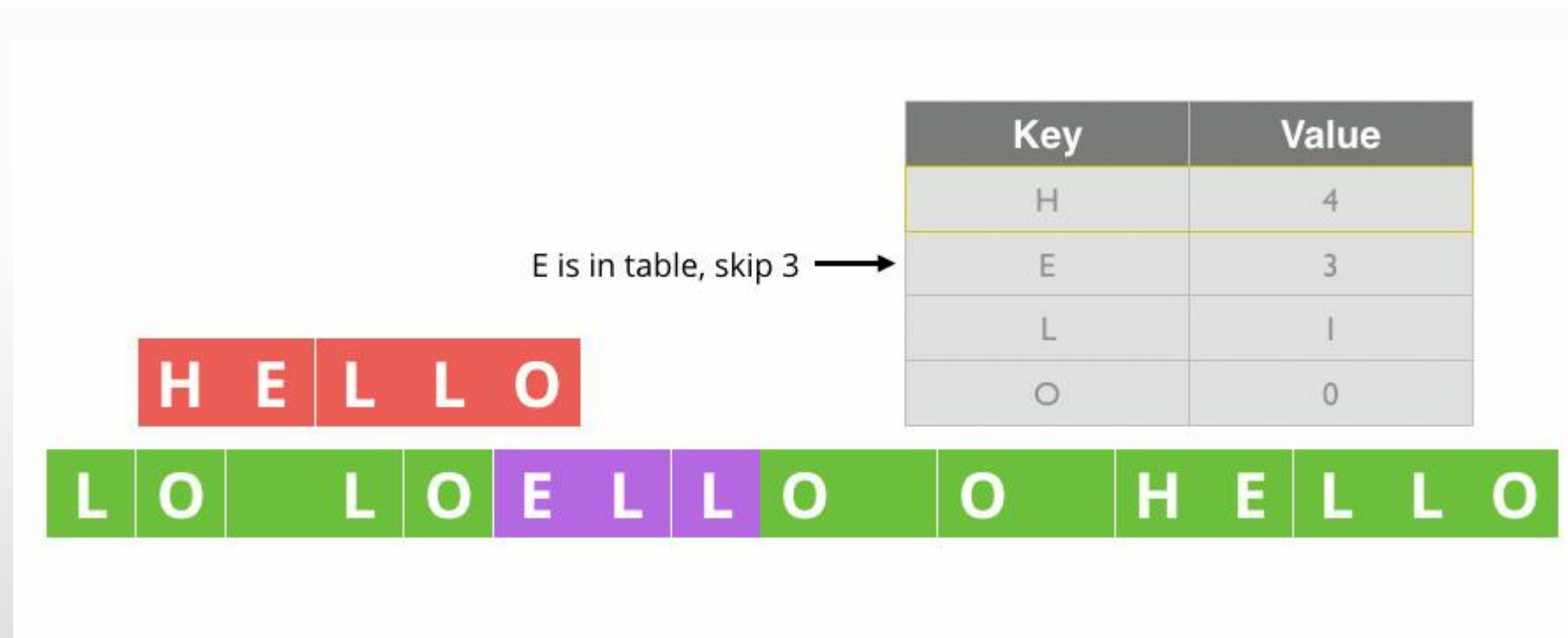
# Boyer Moore



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



HELLO  
LO LOELLOLO HELLO



# Boyer Moore



HELLO  
LO LOELOLO OHELLO

# Boyer Moore



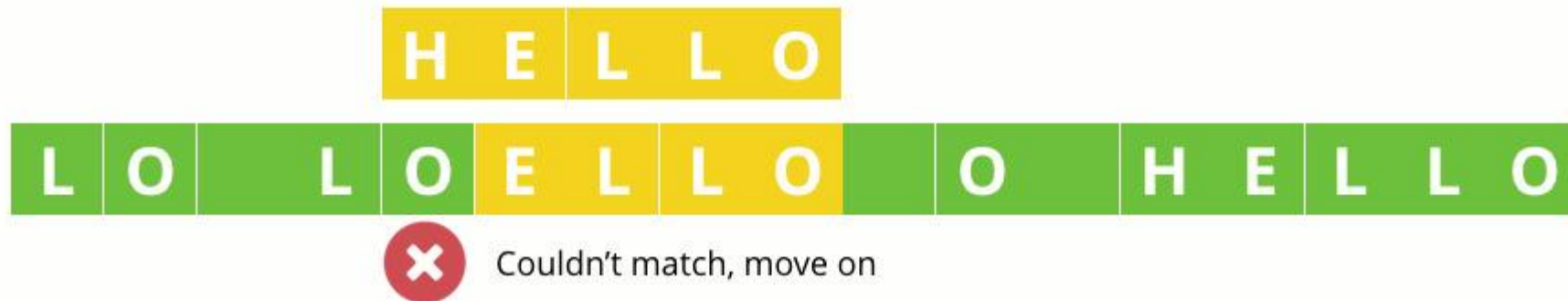
HELLO  
LO LOELOLO OHELLO

# Boyer Moore



HELLO  
LO LOELLO OHELLO

# Boyer Moore



# Boyer Moore



HELLO  
LO LOELLO O HELLO

# Boyer Moore



HELLO  
LO LOELLOLO HELLO

# Boyer Moore



HELLO  
LO LOELLOLO HELLO

# Boyer Moore



HELLO  
LO LOELLO O HELLO

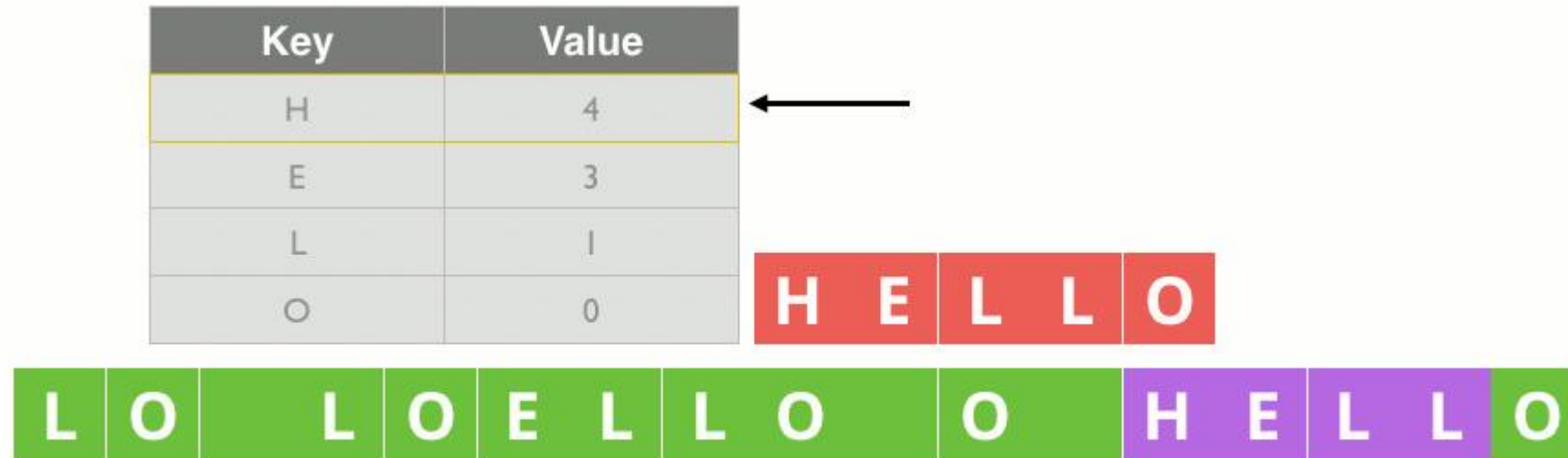


# Boyer Moore



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



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

# Boyer Moore



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

# Boyer Moore



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

# Boyer Moore



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

# Boyer Moore



Diagram illustrating the memory layout of a C++ string. The string "HELLO" is stored in memory, with each character occupying a separate cell. The characters are 'H', 'E', 'L', 'L', 'O', followed by a null terminator '\0'.



# Boyer Moore



✓ H E L L O  
L O L O E L L O O H E L L O





# Boyer Moore

**Pattern**

a	b	a	c	a	b
0	1	2	3	4	5

preprocessing the pattern

**Last occurrence table**

a	b	c	*
4	5	3	-1

\* denotes any character not present in the table

**Text**

a	b	a	c	a	a	b	a	d	c	a	b	a	c	a	b	a	a	b	b
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

comparison counter → 1

1

a	b	a	c	a	b
---	---	---	---	---	---

2

a	b	a	c	a	b
---	---	---	---	---	---

3

a	b	a	c	a	b
---	---	---	---	---	---

4

a	b	a	c	a	b
---	---	---	---	---	---

5

a	b	a	c	a	b
---	---	---	---	---	---



```
preprocessing the pattern
```

\* denotes any character not present in the table

Diagram illustrating the step-by-step construction of a Huffman tree for the sequence "a b a c a b". The diagram shows six rows of nodes, each representing a step in the merging process. The nodes are colored green for 'a', red for 'b', and white for 'c'. The numbers above the nodes indicate their frequency or weight.

- Row 6: 'a' (13) and 'b' (12)
- Row 7: 'a' (11), 'b' (10), 'a' (9), 'c' (8), and 'b' (7)
- Row 8: 'a' (15), 'b' (14), 'a' (13), 'c' (12), 'a' (11), and 'b' (10)
- Row 9: 'a' (18), 'b' (17), 'a' (16), 'c' (15), 'a' (14), and 'b' (13)
- Row 10: 'a' (20), 'b' (19), 'a' (18), 'c' (17), 'a' (16), and 'b' (15)

The sequence "a b a c a b" is shown at the bottom, with the nodes above it representing the Huffman tree structure at each step.





# 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

V U A T S

$$\underline{5} + 1 = 6$$

T S

$$2 + 3 = 5$$

1

Values

$$U = 1$$

$$T = 2$$

$$S = 3$$

$$A = 4$$

$$V = 5$$



# Rabin Karp

V U A T S

$$1 + 4 = 5$$

Spurious Hit

T S

$$2 + 3 = 5$$

2

Values

$$U = 1$$

$$T = 2$$

$$S = 3$$

$$A = 4$$

$$V = 5$$



# Rabin Karp

V U A T S      3

4 + 2 = 6

T S

2 + 3 = 5

Values

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

$$2 + 3 = 5$$

$$U = 1$$

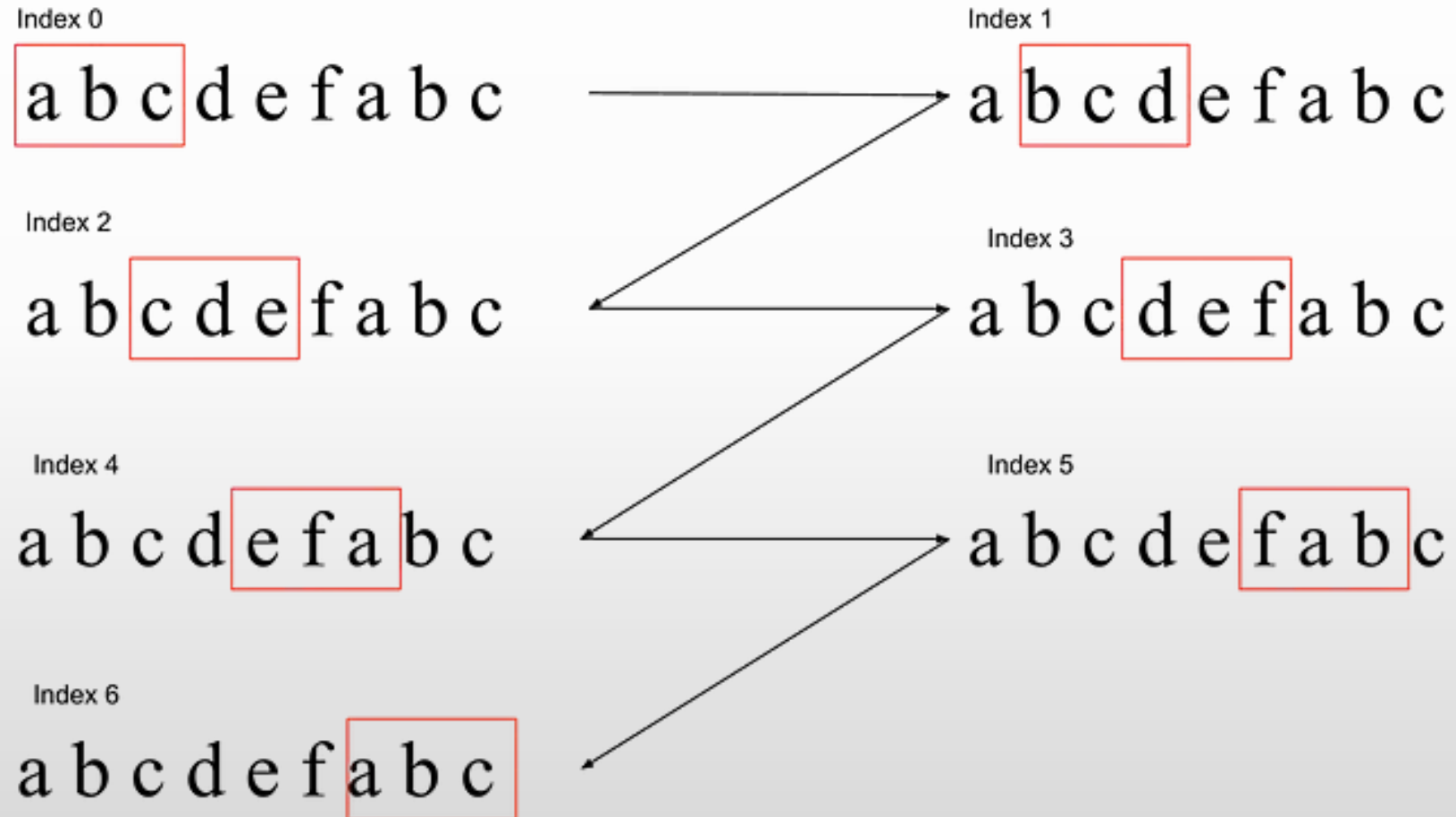
$$T = 2$$

$$S = 3$$

$$A = 4$$

$$V = 5$$

# Rabin Karp





SON