Bölüm 5: Dizge Algoritmaları

Algoritmalar

Dizge Algoritmaları

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

Dize Eşleştirme Algoritmaları

- Brute Force (Kaba Kuvvet):
  - Basit ve doğrudan bir yöntem, Metindeki her konum örüntü ile eşleştirmek için kontrol edilir, Maksimum sayıda karşılaştırma gerektirebilir.
- Knuth-Morris-Pratt (KMP)
  - Metin içinde bir örüntünün bulunmasını sağlayan verimli bir algoritma, Başlangıçta bir tablo oluşturularak arama süresi azaltılır, Karakter karşılaştırmalarını azaltarak hızlı çalışır.
- Boyer-Moore
  - Uzun aramalarda etkilidir, Kök bulma ve kaydırma stratejisi kullanır.
- Rabin-Karp Algoritması
  - Bir dize içinde örüntüyü bulmak için olasılıksal bir algoritma, Hashing işlemi kullanarak aramayı hızlandırır.

Dize Sıkıştırma Algoritmaları

- Sıralı Sıkıştırma Kodlaması (RLE)
  - Veri sıkıştırmasının basit bir şekli, Aynı veri değerlerinin ardışık dizileri tek bir değer ve sayı olarak saklanır, Tekrar eden değerlerin sayısı yerine tekrar eden veri değeri saklanır.
- Lempel-Ziv-Welch (LZW)
  - GIF ve TIFF gibi formatlarda kullanılan bir sözlük tabanlı sıkıştırma algoritması, Tekrar eden örüntüler için bir sözlük oluşturur ve bu örüntüleri daha kısa sembollerle temsil eder, Dinamik bir sözlük kullanarak sıkıştırma sağlar.

Dize Sıralama Algoritmaları

- Sözlüksel Sıralama (Lexicographic Order)
  - Dizeler, alfabetik sıraya benzer şekilde sıralanır. Her karakterin ASCII değeri karşılaştırılarak sıralama yapılır.
- Radix Sıralama
  - Karşılaştırmalı olmayan bir tam sayı sıralama algoritmasıdır. Tam sayı anahtarlarına sahip verileri, aynı önemli konum ve değeri paylaşan rakamlara göre gruplandırarak sıralar. Her basamak için ayrı ayrı işlem yaparak sıralama işlemini gerçekleştirir.

Dize Ayrıştırma (Parsing) Algoritmaları

- Düzenli İfadeler (Regular Expressions)
  - Bir arama örüntüsünü tanımlayan karakter dizisi, Genellikle örüntülere dayalı olarak dize ayrıştırma ve arama için kullanılır. Belirli bir örüntüye uyan tüm dizeleri bulmak için kullanılır
- Sonlu Durum Makineleri (Finite State Machines - FSM)
  - Dizeler veya diziler içindeki örüntüleri tanımak için kullanılan hesaplama modelleri, Belirli bir girdi dizisindeki geçişlerin durumlarını izleyen bir otomat, Karmaşık ayrıştırma ve analiz işlemlerinde kullanılır.

Dize Düzenleme Mesafesi Algoritmaları

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

Dize Dönüşüm Algoritmaları

- Sonek Dizisi (Suffix Array)
  - Bir dizenin tüm son eklerinin bir dizisi, leksikografik (lexicographically) sıralanmıştır. Dize içindeki alt dizelerin bir temsili olarak kullanılır.
- Burrows-Wheeler Dönüşümü (BWT)
  - Bir dizenin tersine dönüştürülmesiyle elde edilen yeni bir form, Bzip2 gibi sıkıştırma algoritmaları için önişlem adımı olarak kullanılır.

String Naive Algoritması

- Bir 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 konum için örüntünün ilk karakterinin eşleşip eşleşmediğini kontrol edilir.
- Eşleşen karakterlerin tümü için örüntünün tam olarak eşleşip eşleşmediğini kontrol edilir.
  - Eşleşme Durumu: eşleşme pozisyonu rapor edilir.
  - Eşleşmeme Durumu: sonraki pozisyonlar kontrol edilir.
- Metindeki tüm konumlar için adımlar tekrarlanır.

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

Brute Force - Naive

Knuth-Morris-Pratt (KMP) Algoritması

- Bir 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

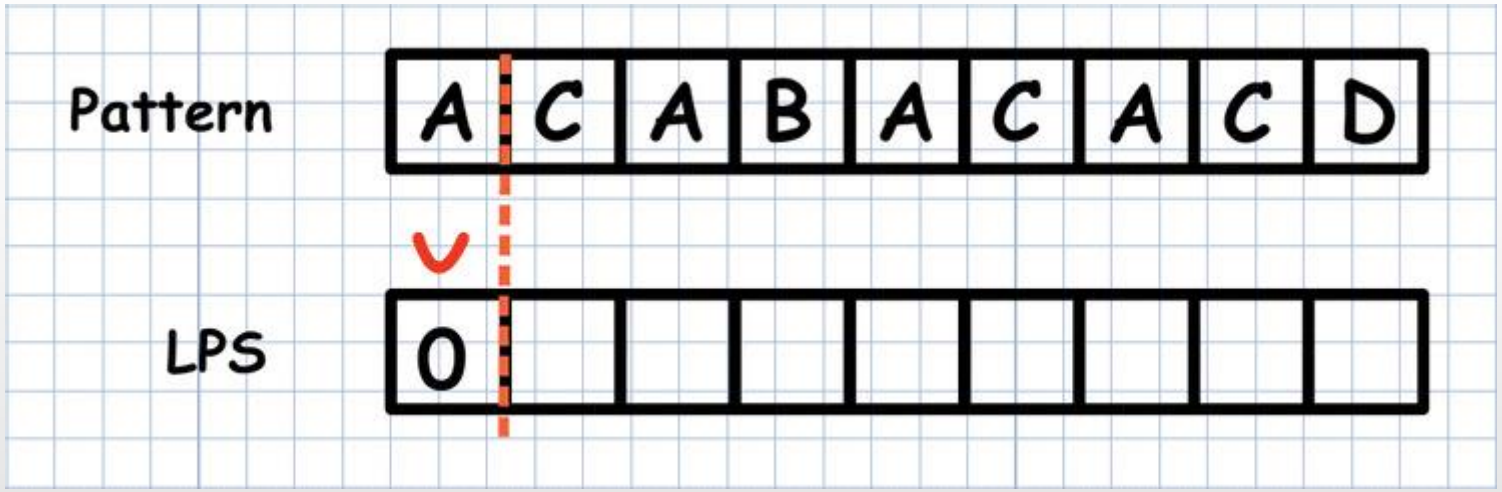
- Ön İşleme: Örüntü içindeki her karakter için,
  - eşleşme durumunda geri dönülecek pozisyonları belirleyen,
  - en uzun önek-suffix eşleşmesini bulan bir tablo oluşturulur.
- Örüntü Arama: Metin içinde arama yapılırken,
  - örüntü ile eşleşmeyen karakterlerde geri dönülecek pozisyonlar,
  - tablodan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

Örnek

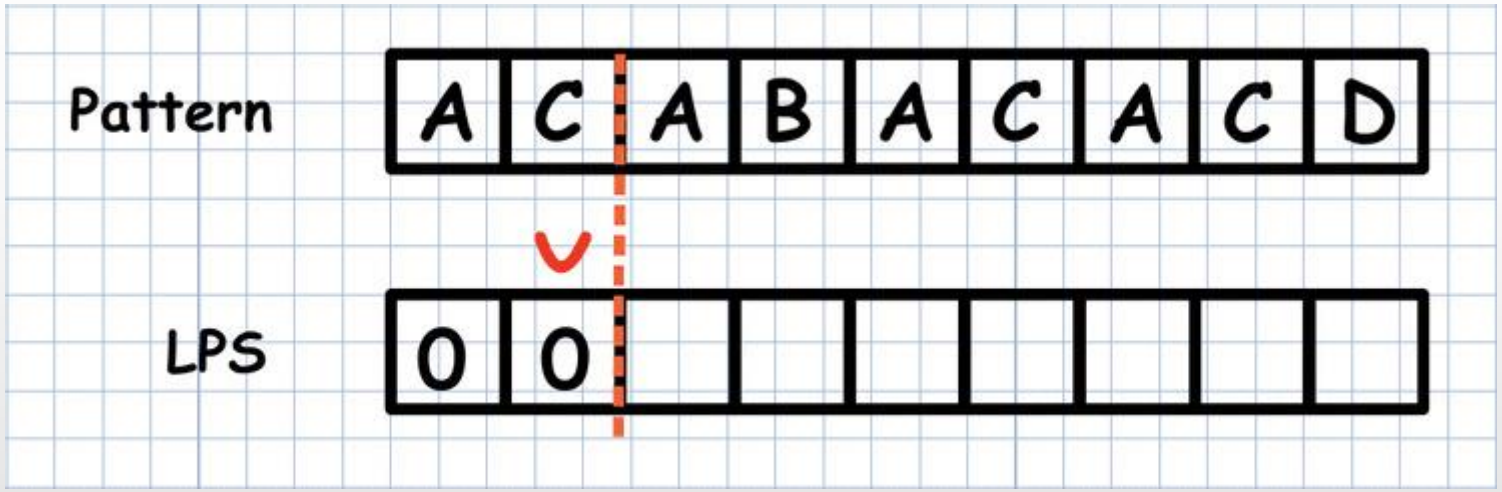
- Metin: "ababcababcababababd"
- Örüntü: "ababd"
- Örüntü Tablosu:
  - a: 0, b: 0, a: 1, b: 2, d: 0

Knuth Morris Pratt

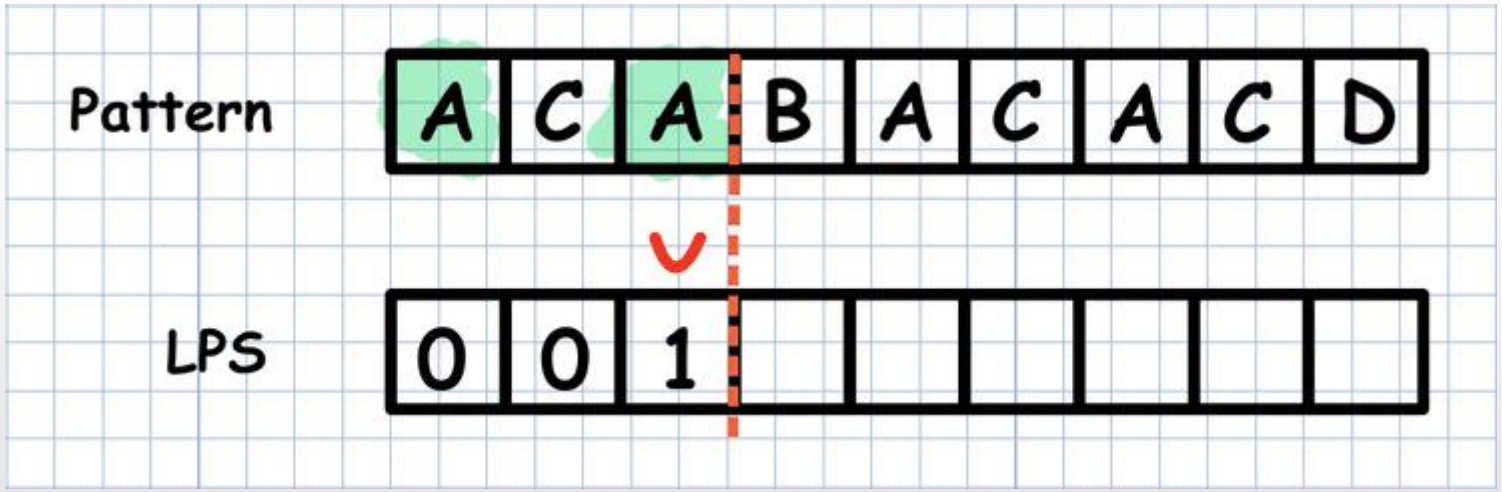
- *Longest Proper Prefix*



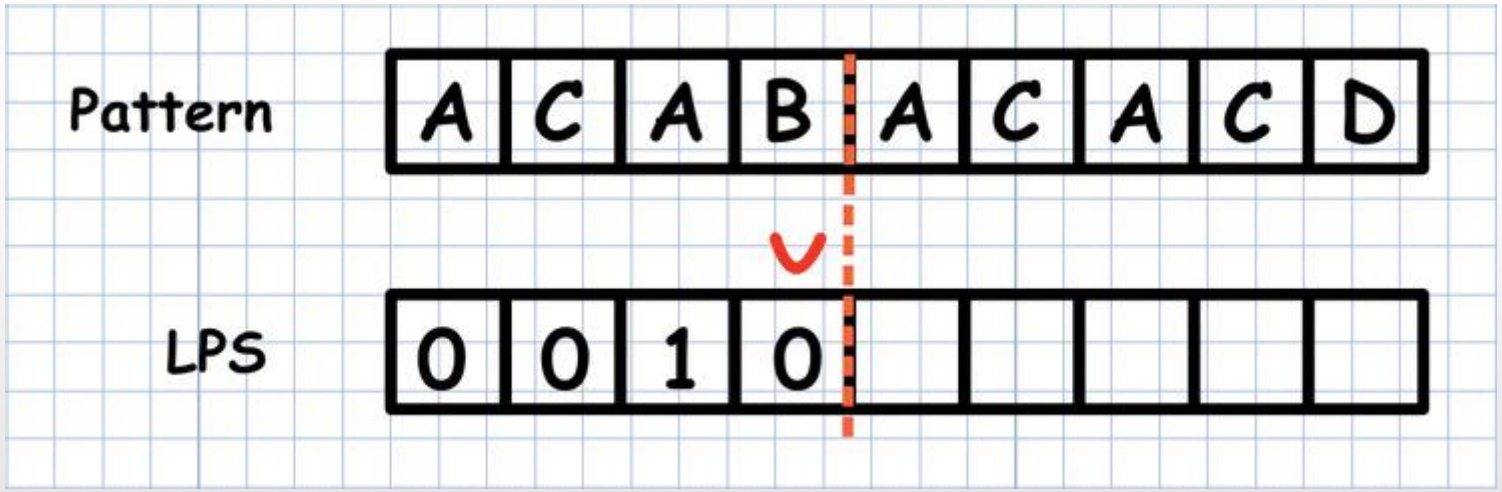
Knuth Morris Pratt



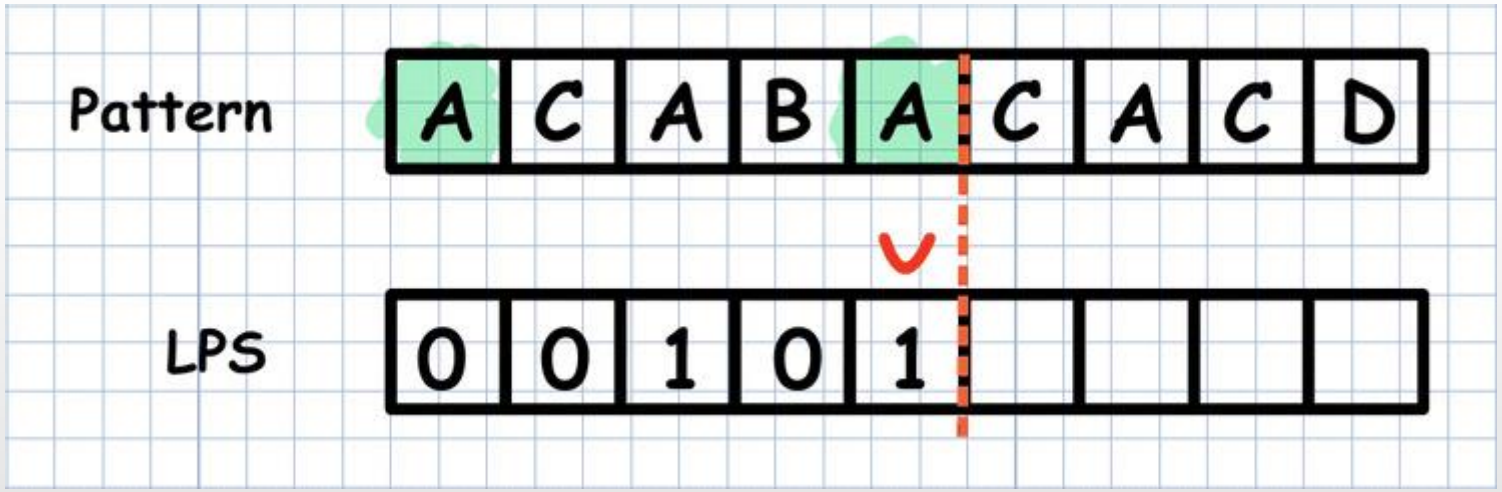
Knuth Morris Pratt



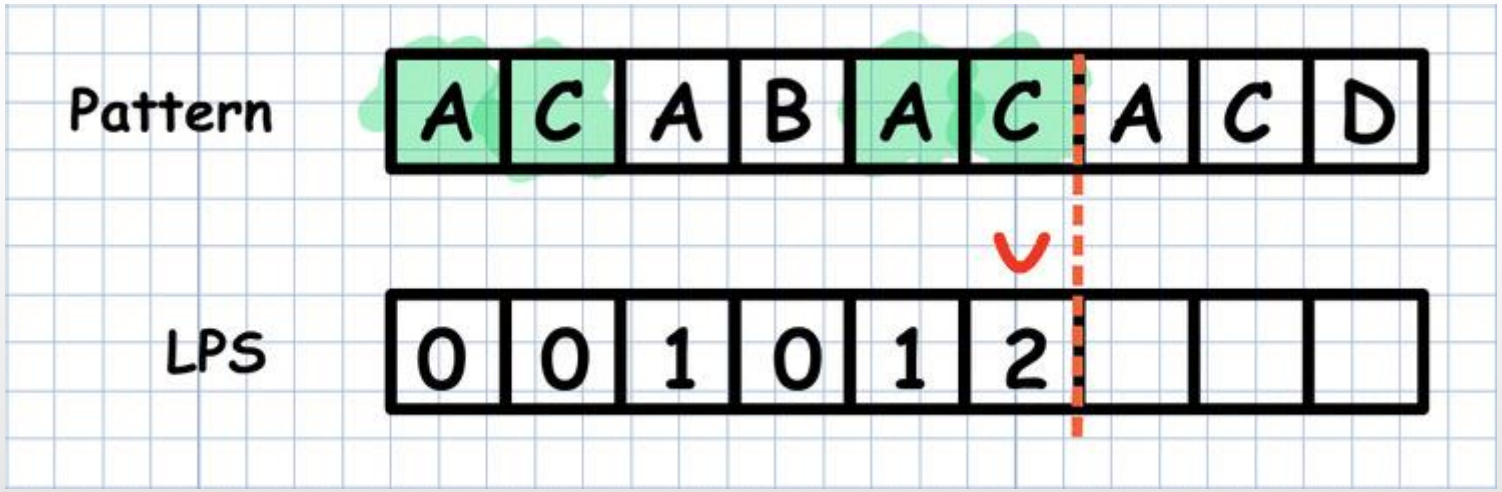
Knuth Morris Pratt



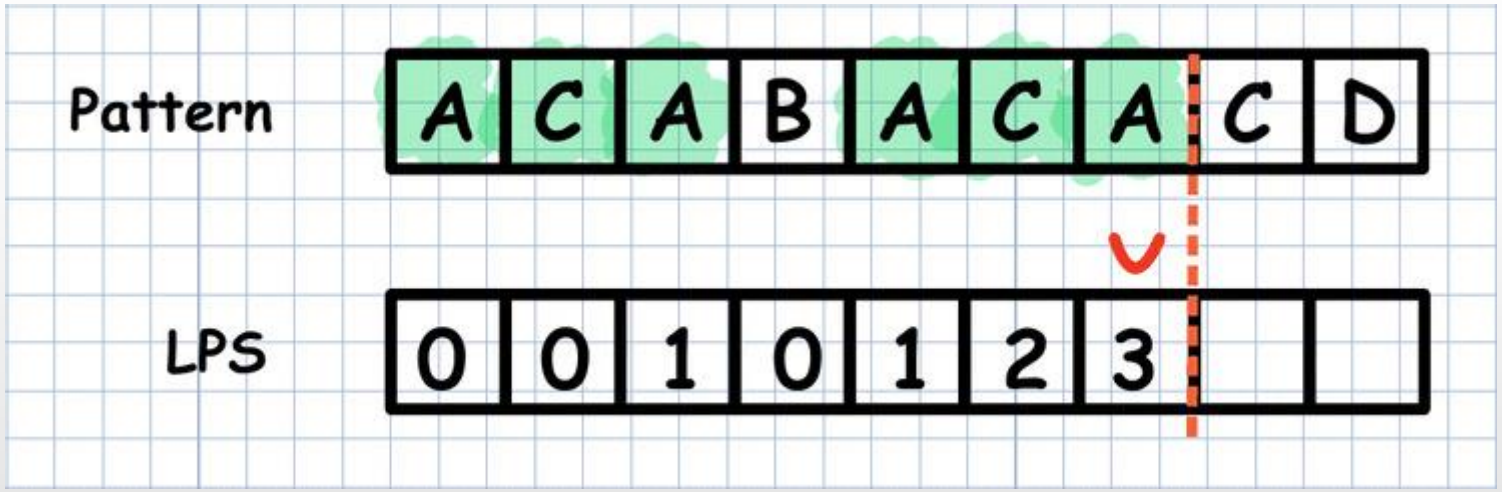
Knuth Morris Pratt



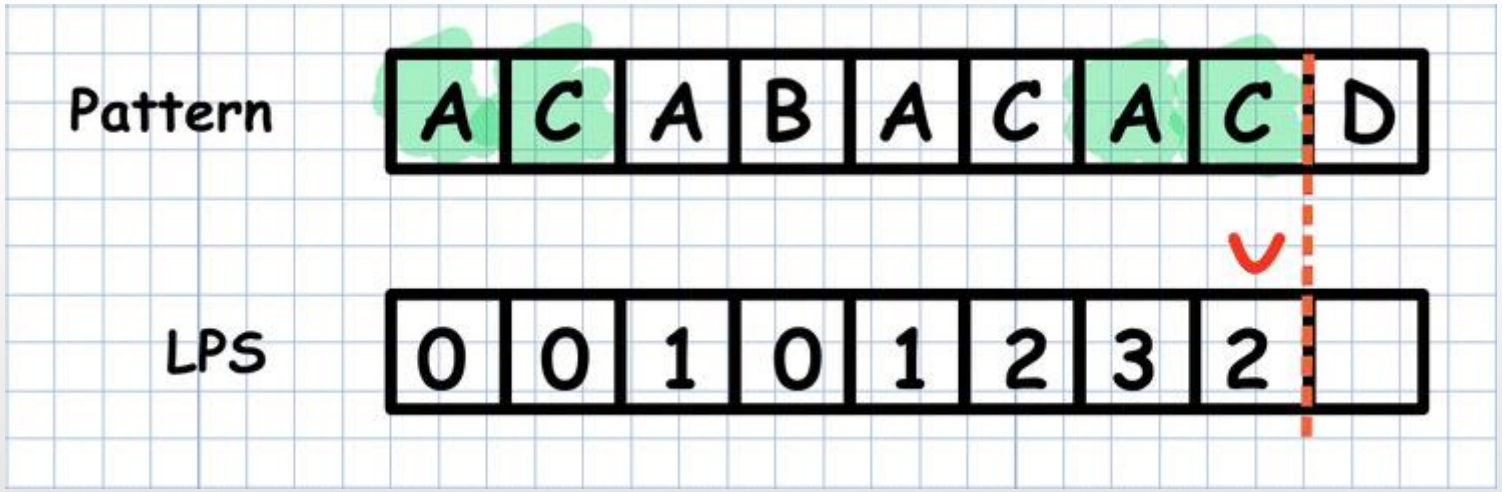
Knuth Morris Pratt



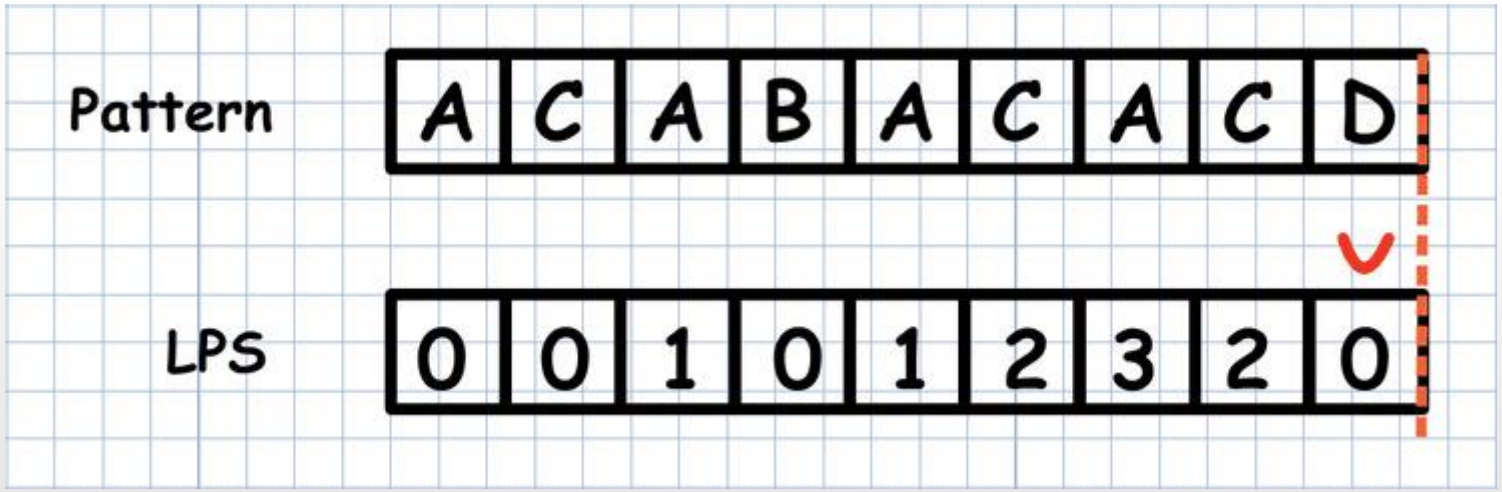
Knuth Morris Pratt



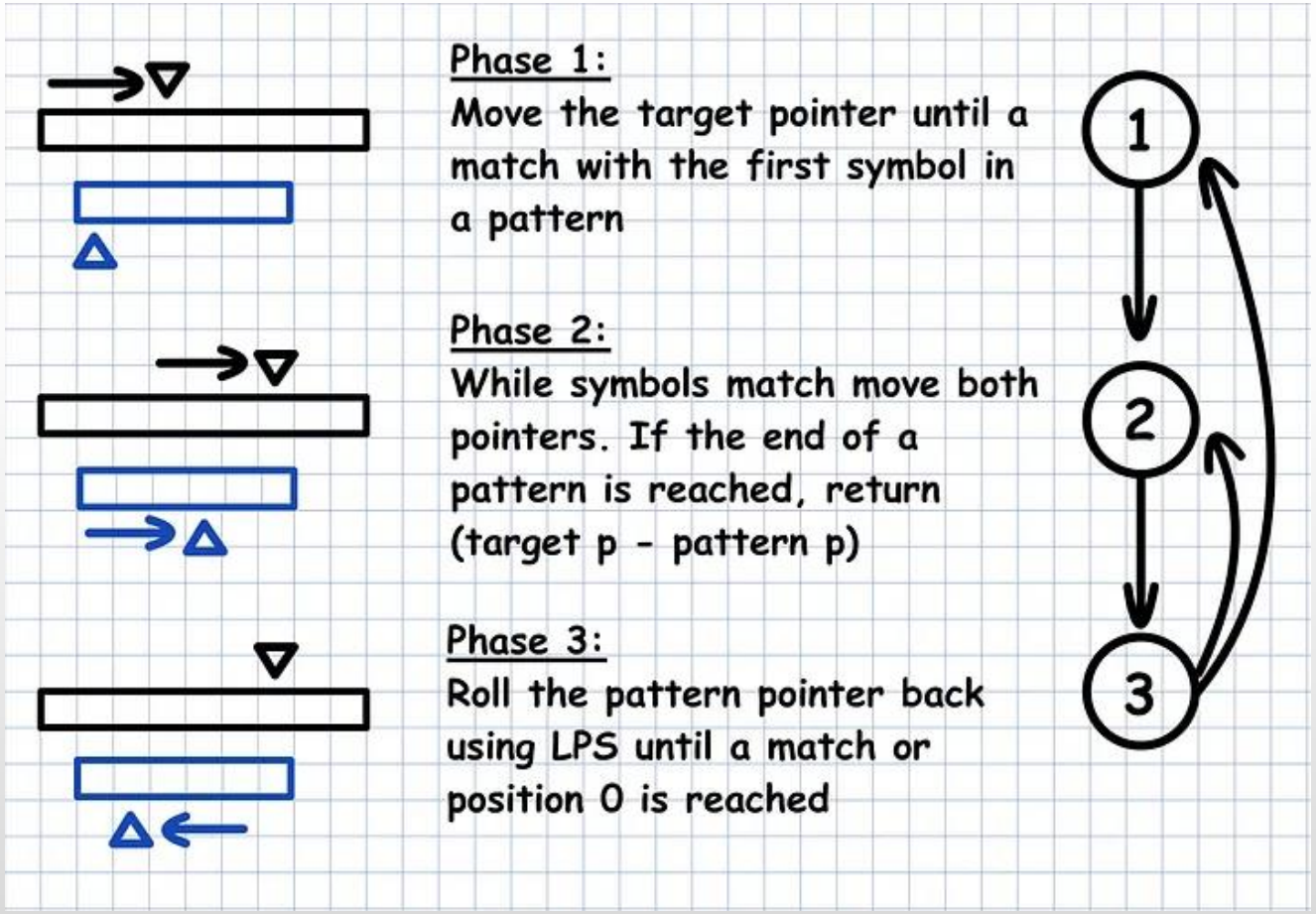
Knuth Morris Pratt



Knuth Morris Pratt



Knuth Morris Pratt



Boyer-Moore Algoritması

- Bir 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: "abccbaabccbaabcbcabbabcabc"
- Desen: "abcbcabbabcabc"
- Desen Tablosu:
  - a: 10, b: 8, c: 7
- Sonuç:
  - Pozisyon 12: "abcbcabbabcabc"

Boyer Moore

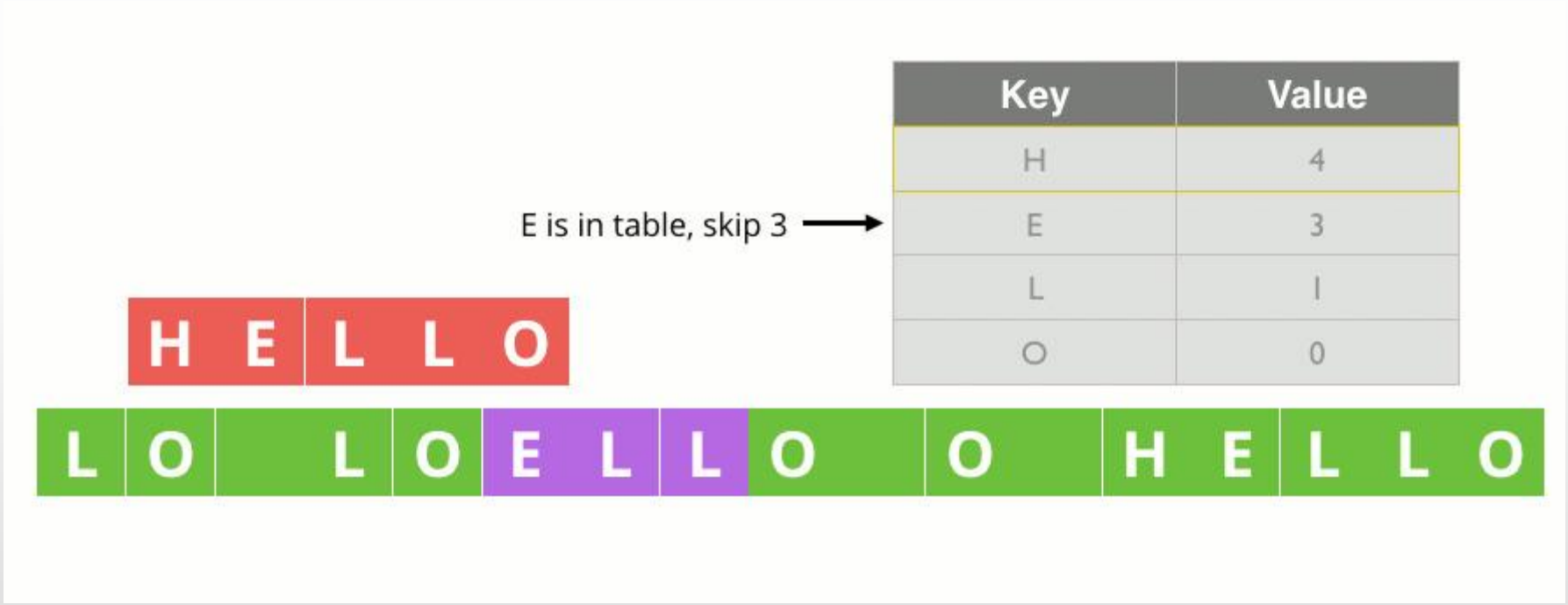
Boyer Moore

Boyer Moore

Boyer Moore

Boyer Moore

Boyer Moore

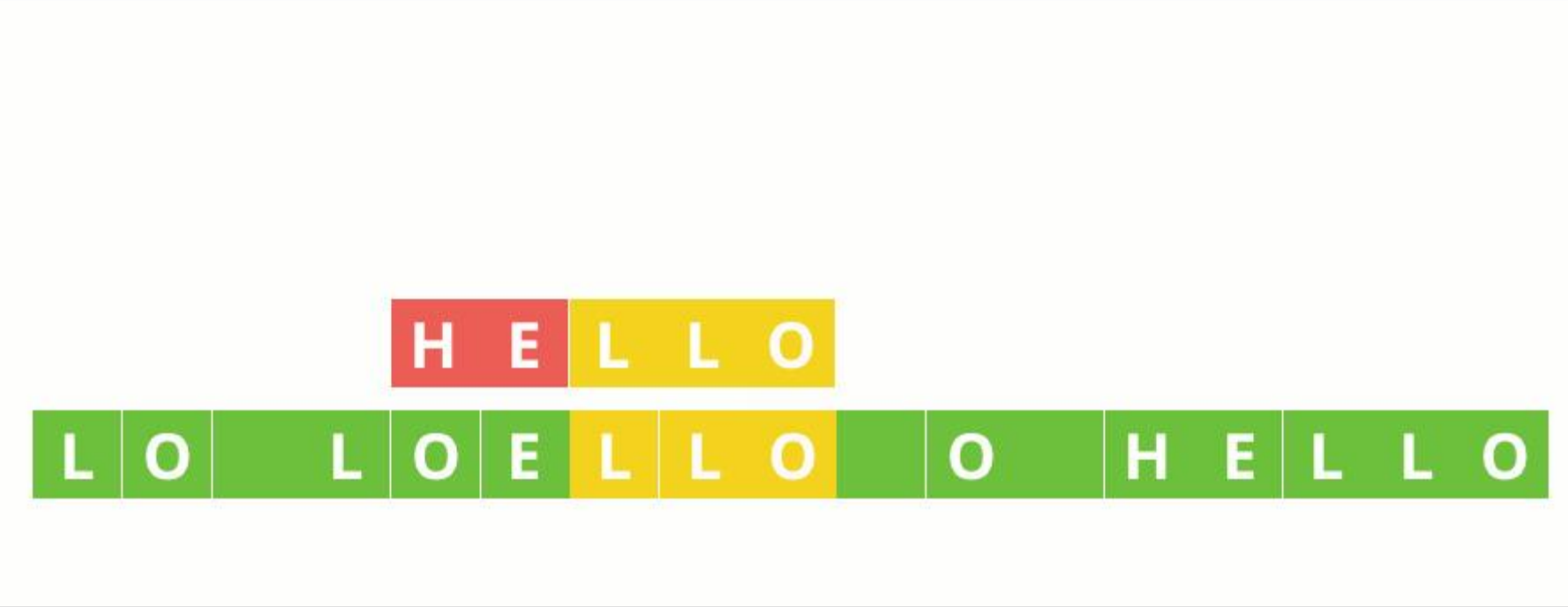


Boyer Moore

Boyer Moore

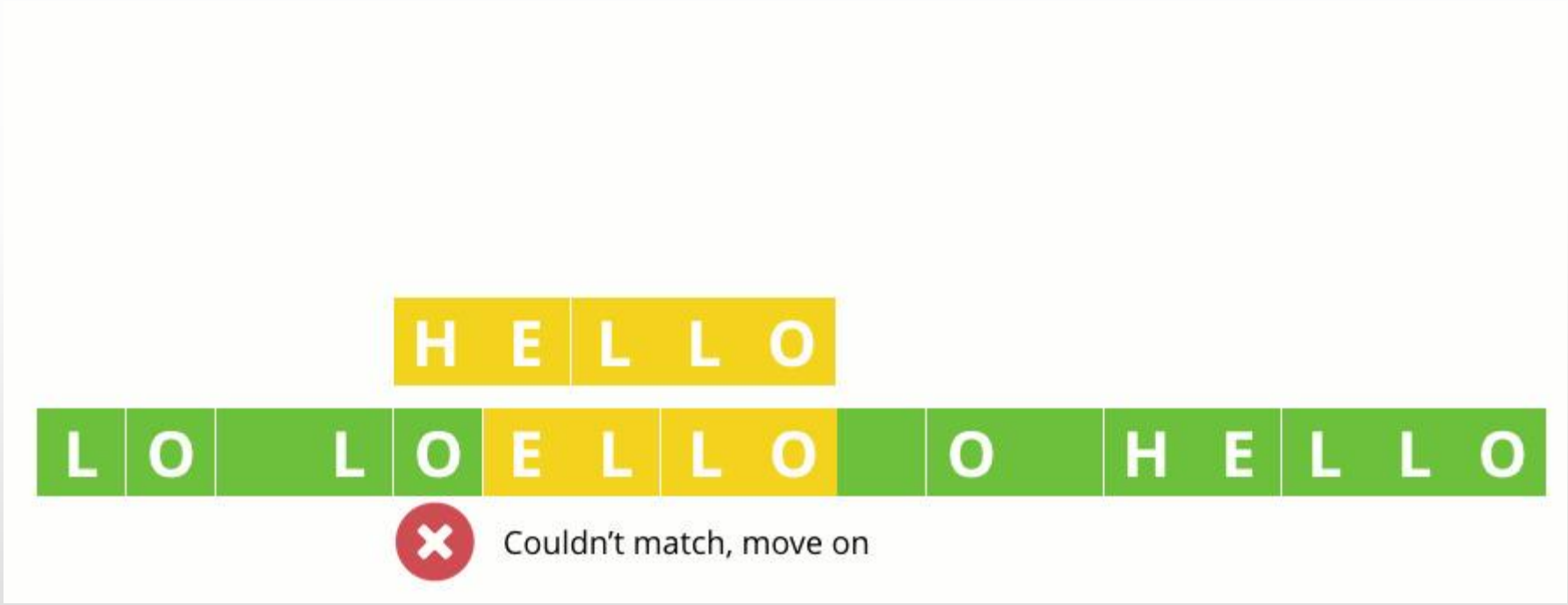
Boyer Moore

Boyer Moore



Boyer Moore

Boyer Moore



Boyer Moore

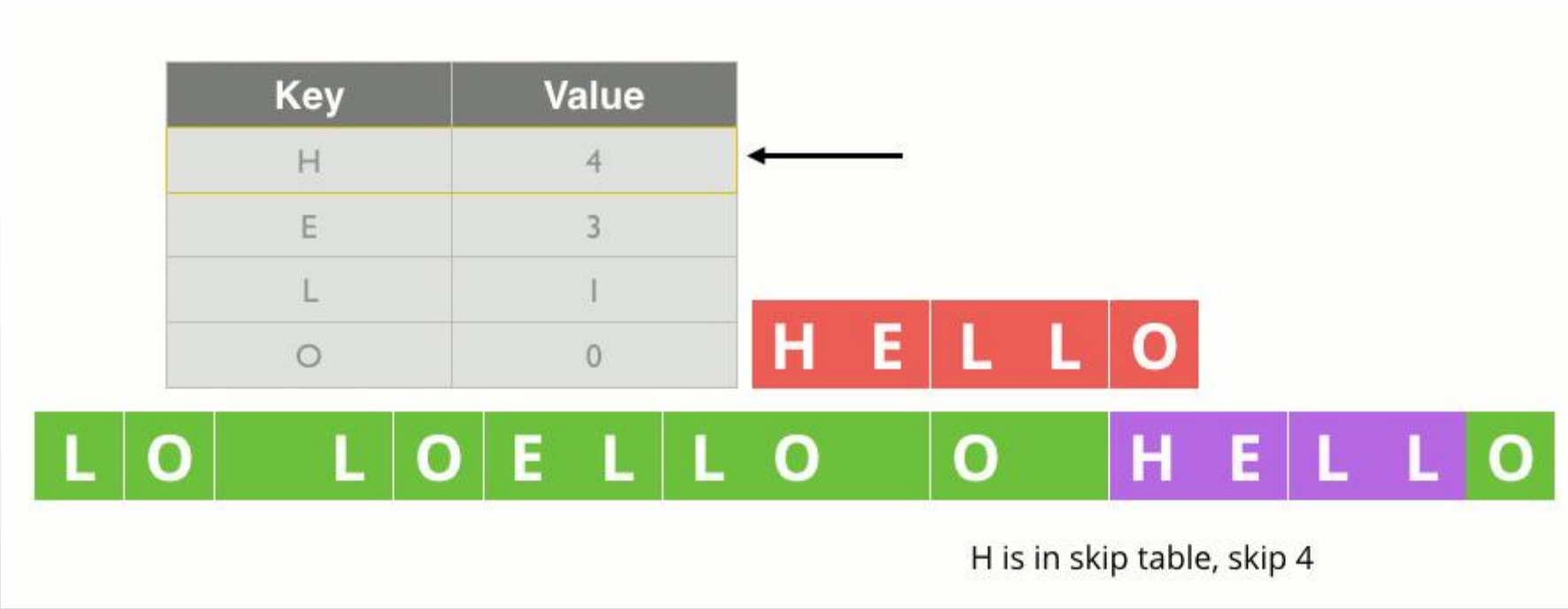
Boyer Moore

Boyer Moore

Boyer Moore

Boyer Moore

Boyer Moore



Boyer Moore

Boyer Moore

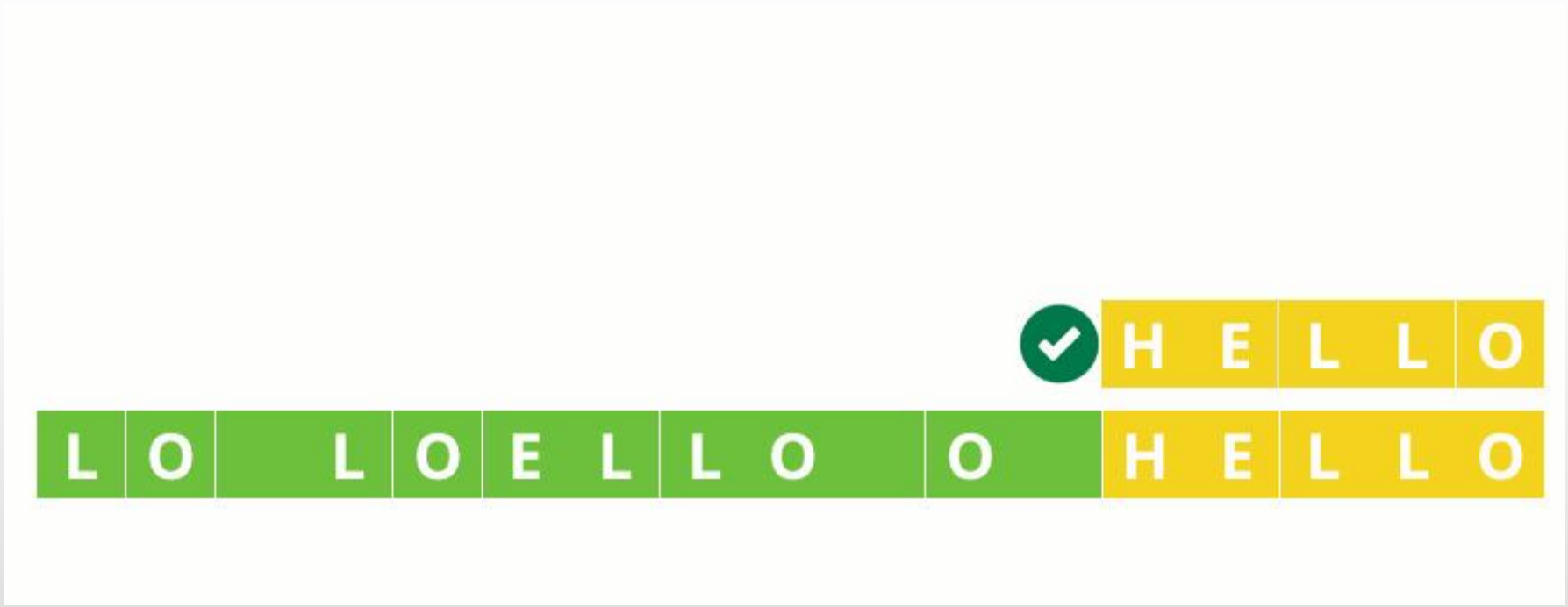
Boyer Moore

Boyer Moore

Boyer Moore

Boyer Moore

Boyer Moore



Rabin-Karp Algoritması

- Bir 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)

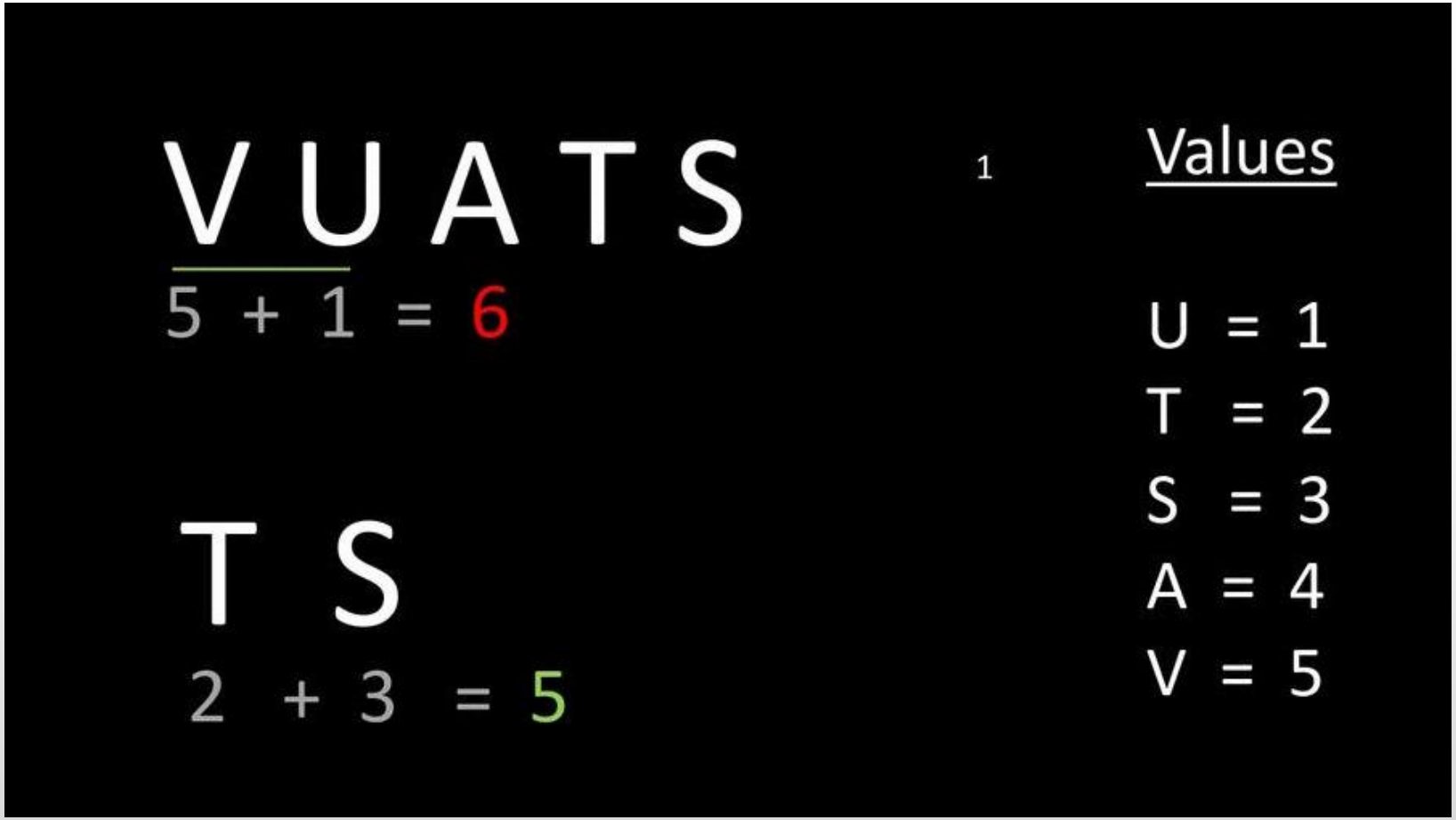
İşleyiş

- 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.

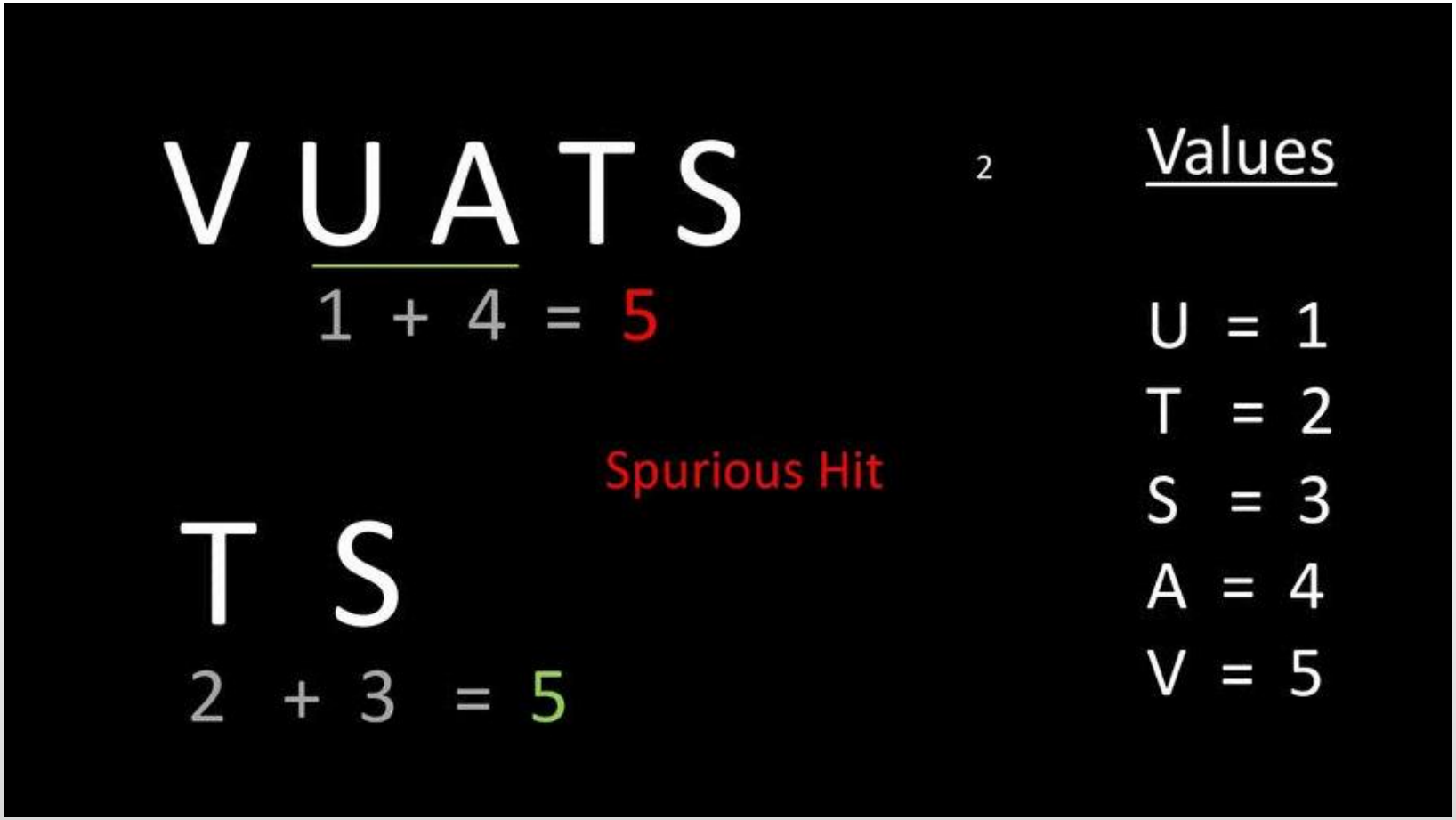
Örnek

- 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"

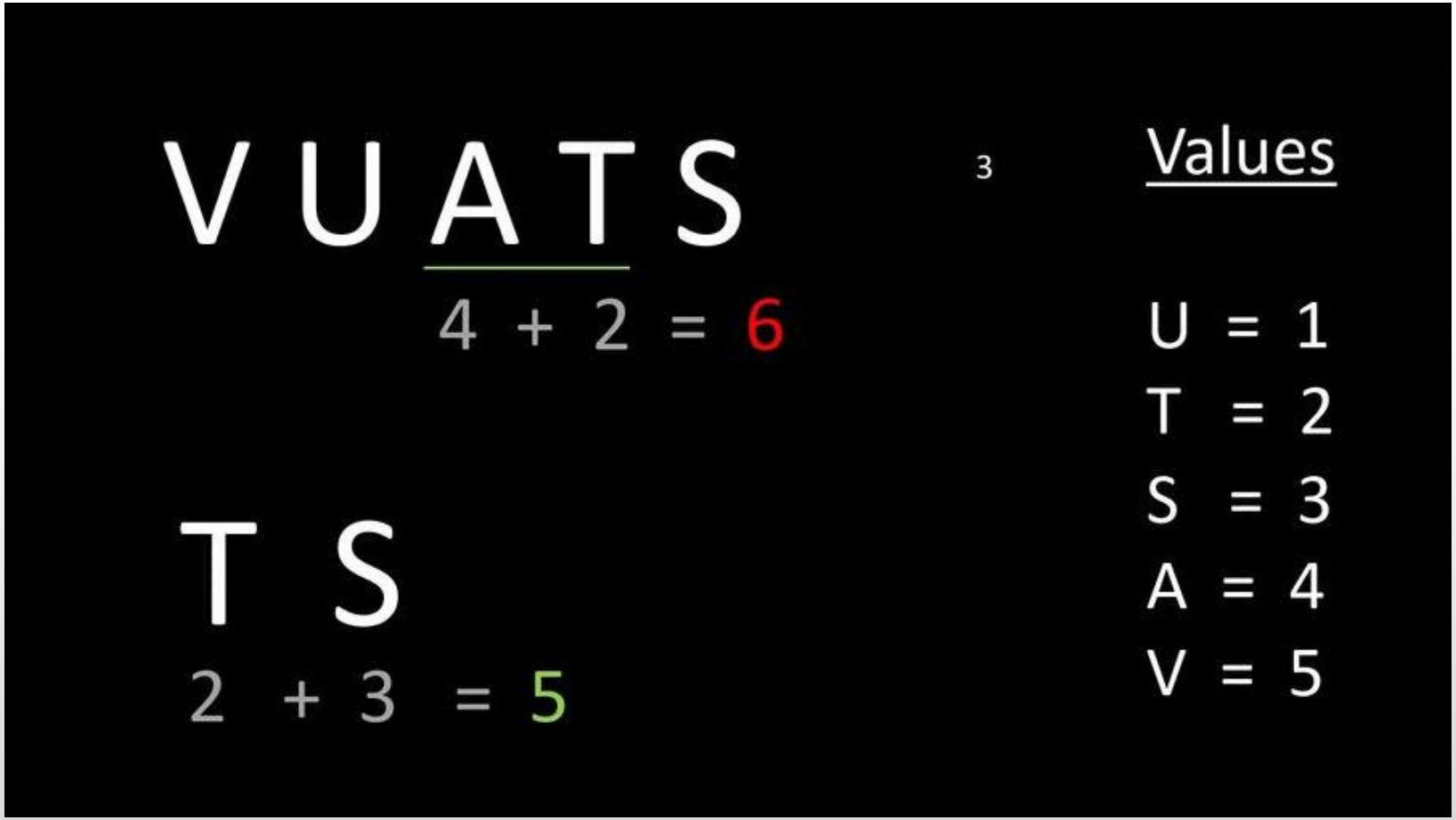
Rabin Karp



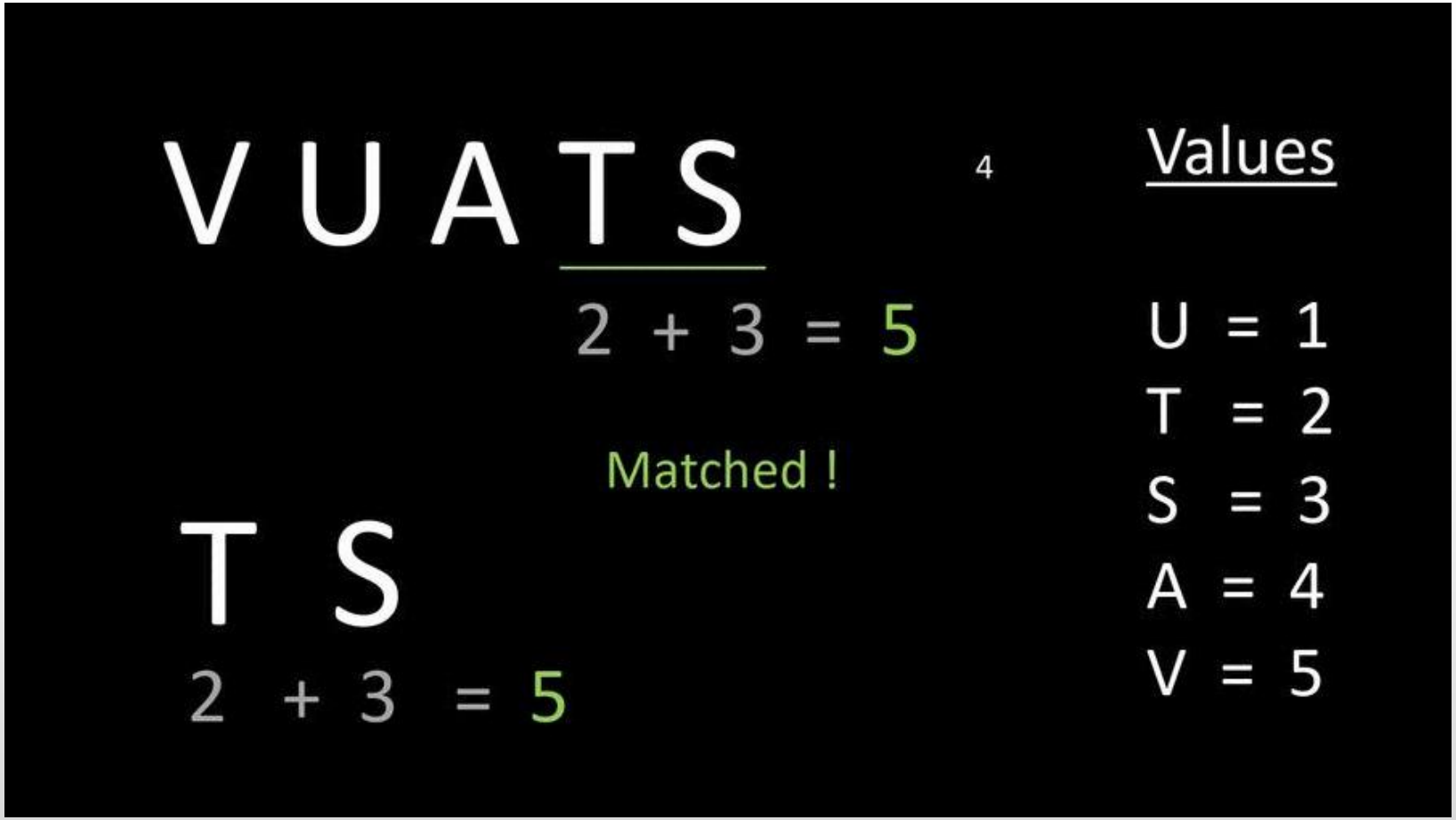
Rabin Karp



Rabin Karp



Rabin Karp



SON