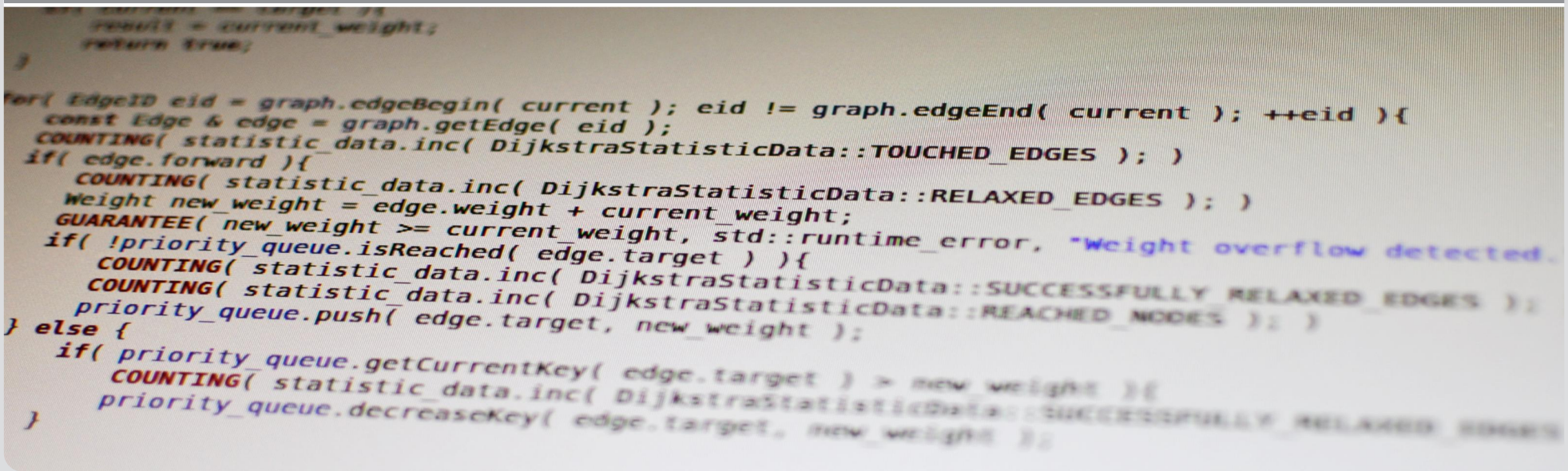
Karlsruher Institut für Technologie

Übung 3 – Algorithmen II

Yaroslav Akhremtsev, Demian Hespe – *yaroslav.akhremtsev@kit.edu, hespe@kit.edu*
Mit Folien von Michael Axtmann (teilweise)

`http://algo2.iti.kit.edu/AlgorithmenII_WS17.php`

Institut für Theoretische Informatik - Algorithmik II



Themenübersicht

- *In-place Multikey Quicksort*
  (Sortierung von Zeichenketten *in-place*)
- Suche mit Hilfe von Border-Array
- Suche mit Hilfe von Suffix-Arrays
  - Beschleunigung mittels LCP-Array

In-place Multikey Quicksort

Wiederholung

Bentley, Sedgewick (1997)

(*Three-way Radix Quicksort*)

- sortiert Elemente mit mehreren Schlüsseln wie msd-*Radixsort*
  $\rightarrow$ z.B. Stellen einer Zahl, Zeichen eines Strings

- für einen Schlüssel wird *Quicksort* mit drei Fällen ausgeführt
  $\rightarrow$ kleiner als, gleich, größer als das Pivotelement

- Partitionierungschritt kann *in-place* erfolgen
  $\rightarrow$ ähnlich wie bei normalem *Quicksort*

In-place Multikey Quicksort

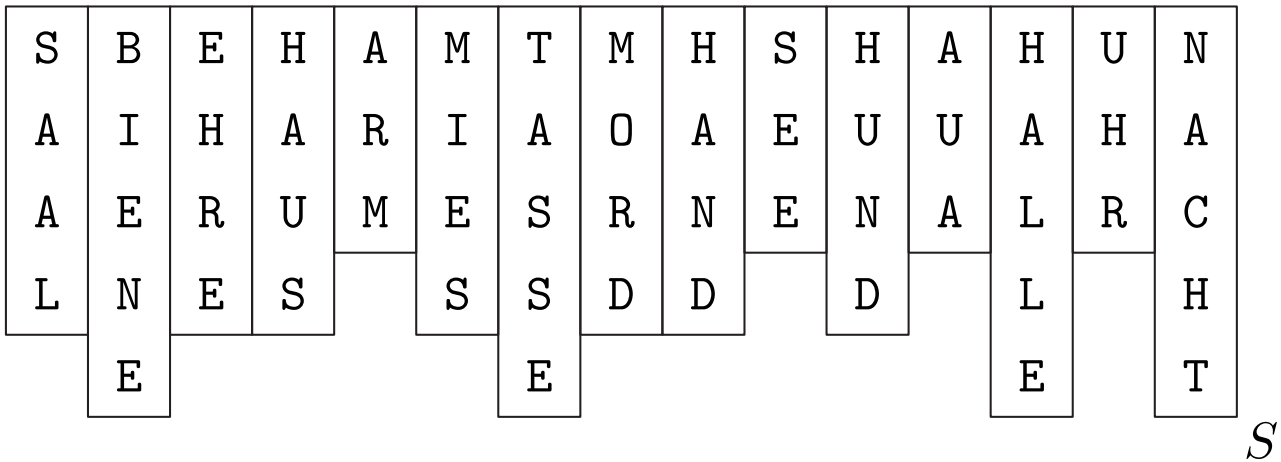
Wiederholung: Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)

In-place Multikey Quicksort

Wiederholung: Ablauf

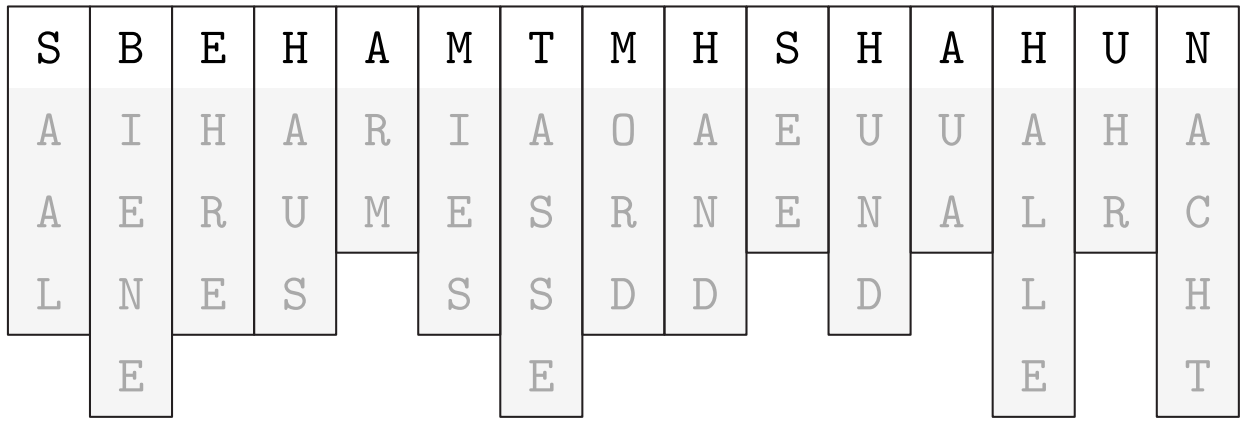
Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



In-place Multikey Quicksort

Wiederholung: Ablauf

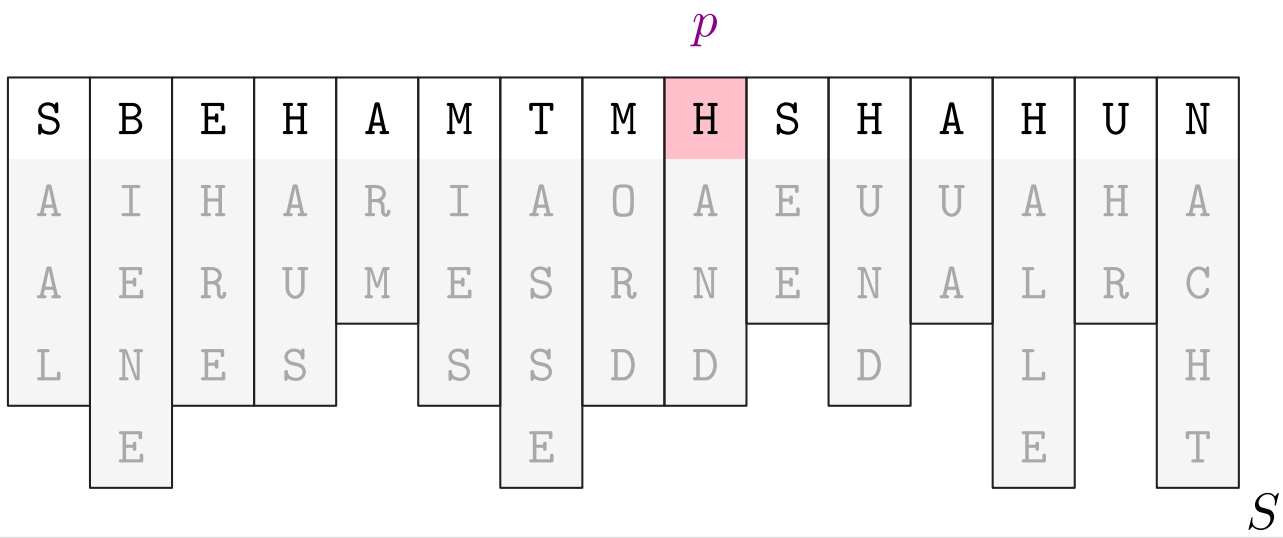
Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



In-place Multikey Quicksort

Wiederholung: Ablauf

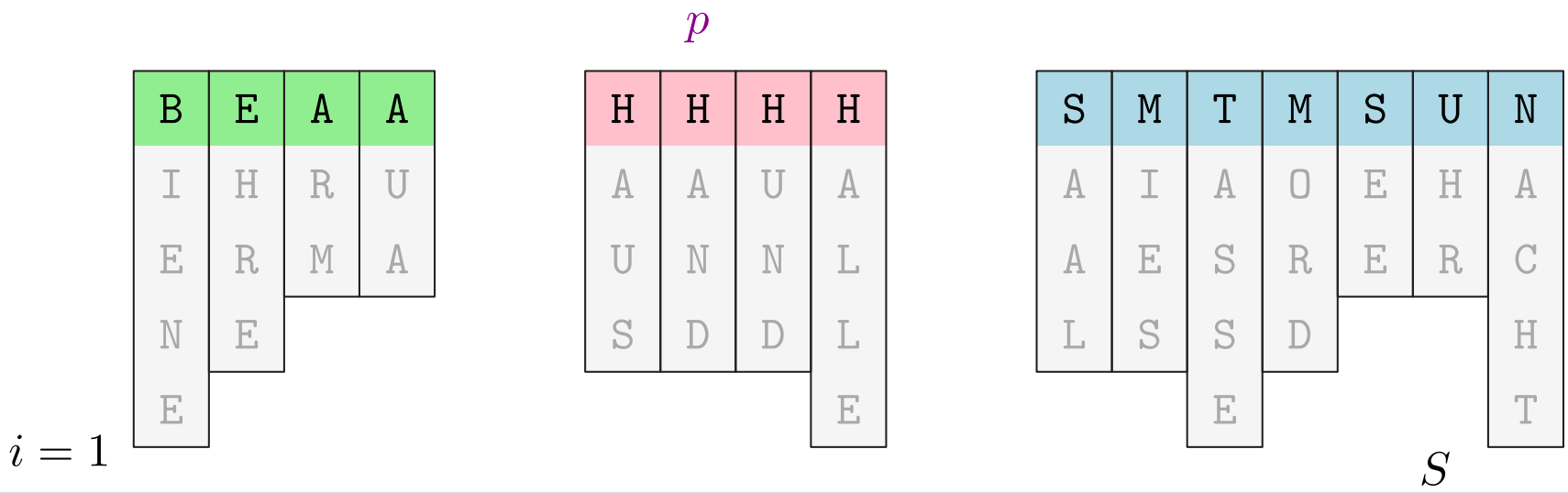
Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



In-place Multikey Quicksort

Wiederholung: Ablauf

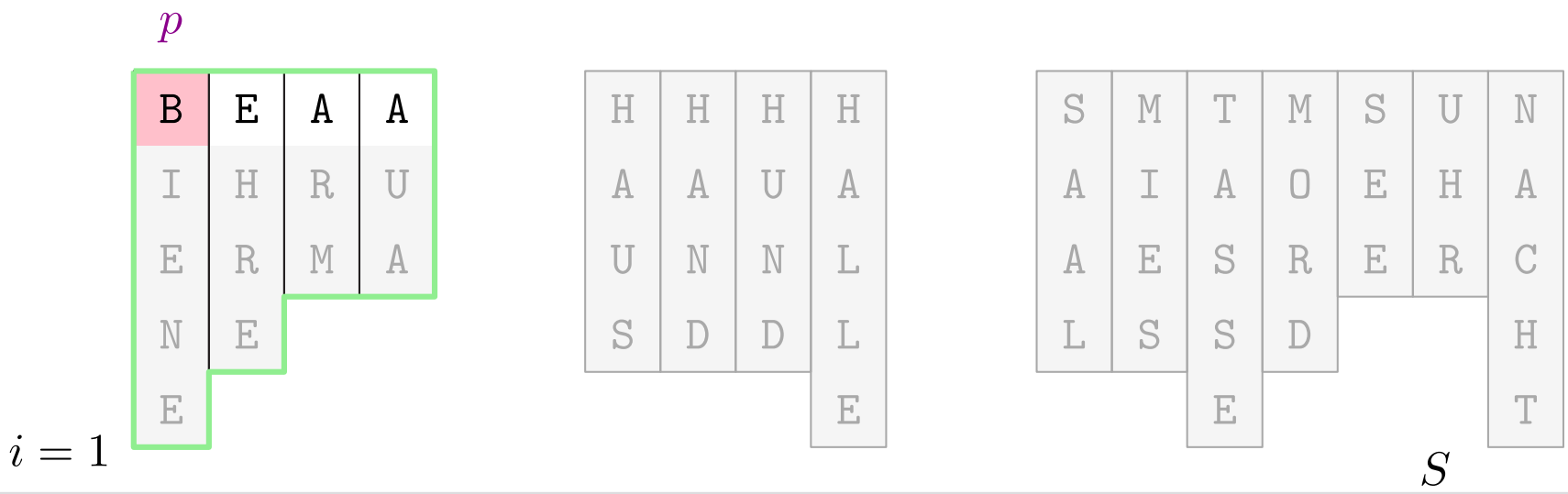
Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



In-place Multikey Quicksort

Wiederholung: Ablauf

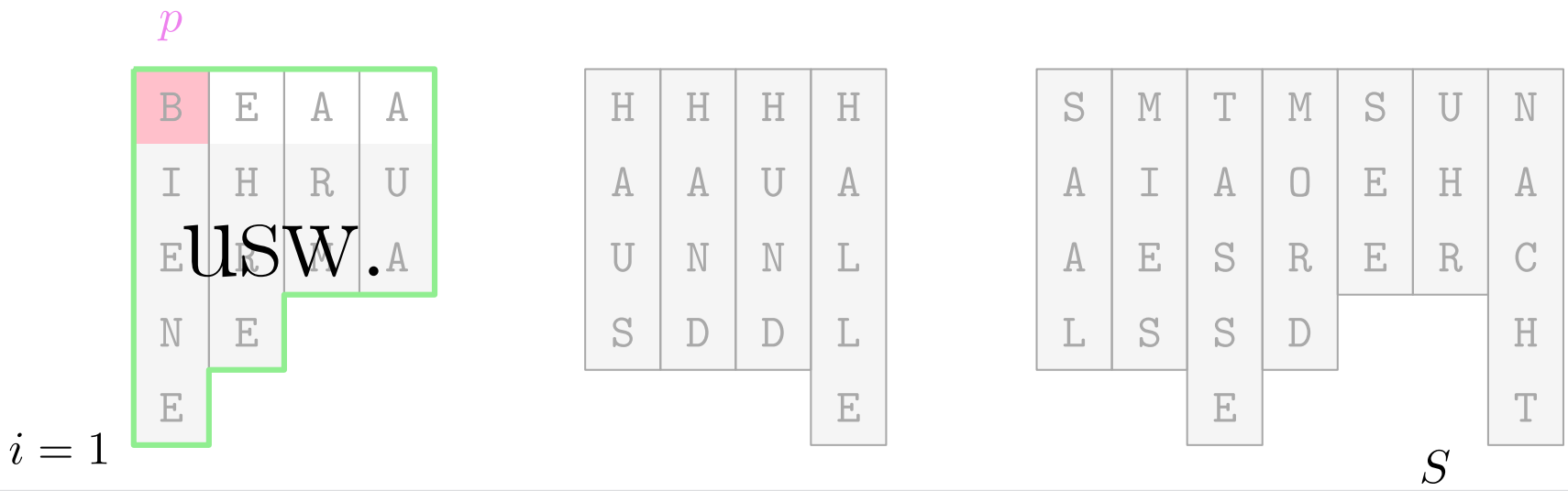
Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



In-place Multikey Quicksort

Wiederholung: Ablauf

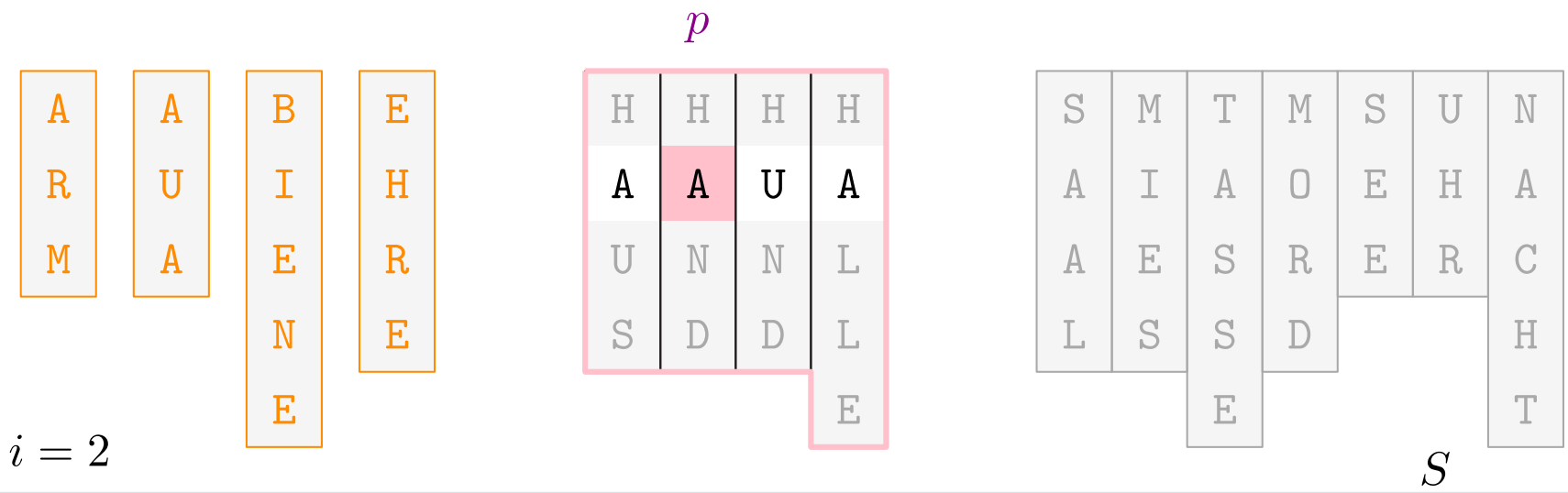
Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



In-place Multikey Quicksort

Wiederholung: Ablauf

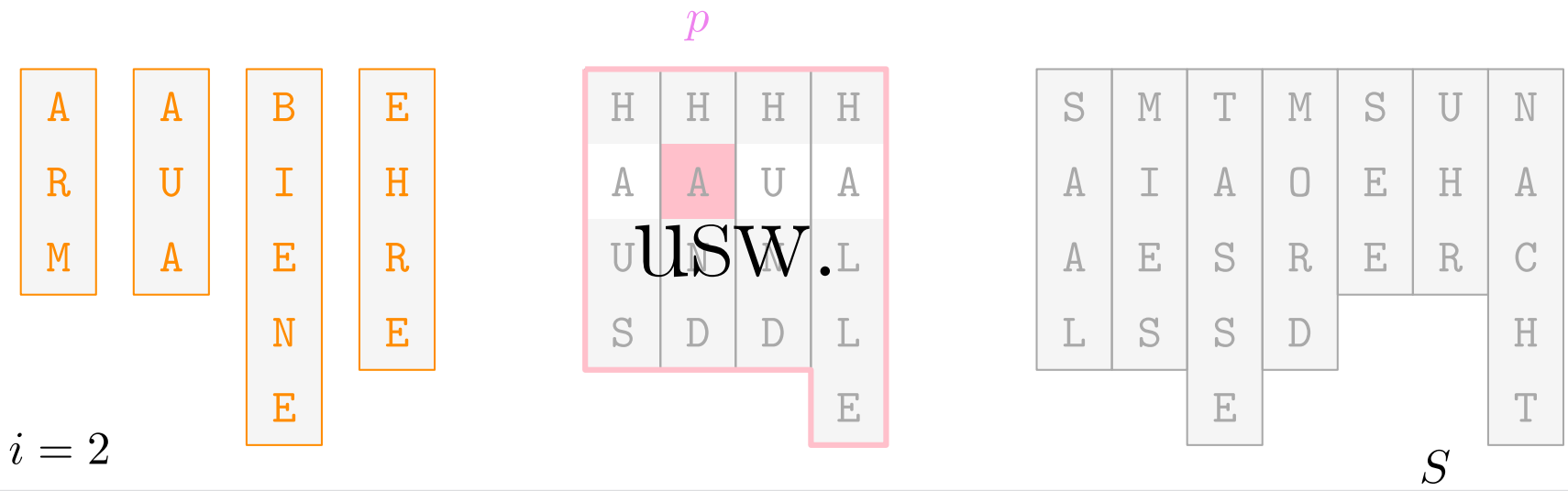
Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



In-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)



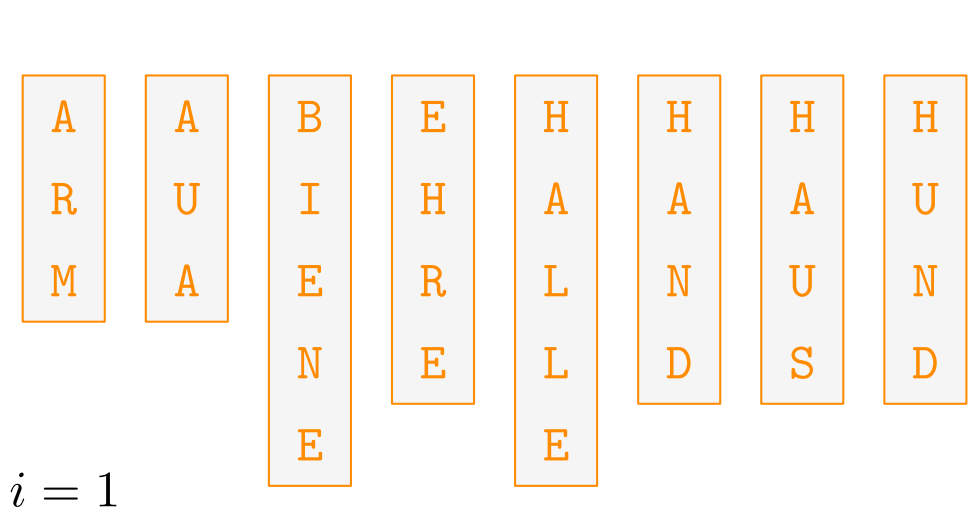
$i = 2$

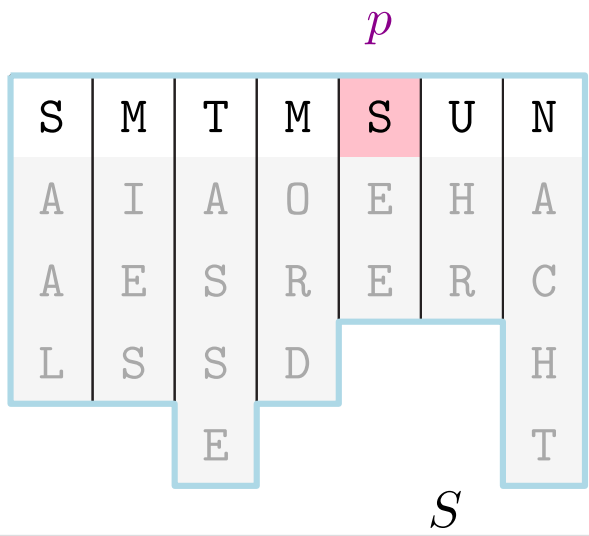
usw.

In-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)

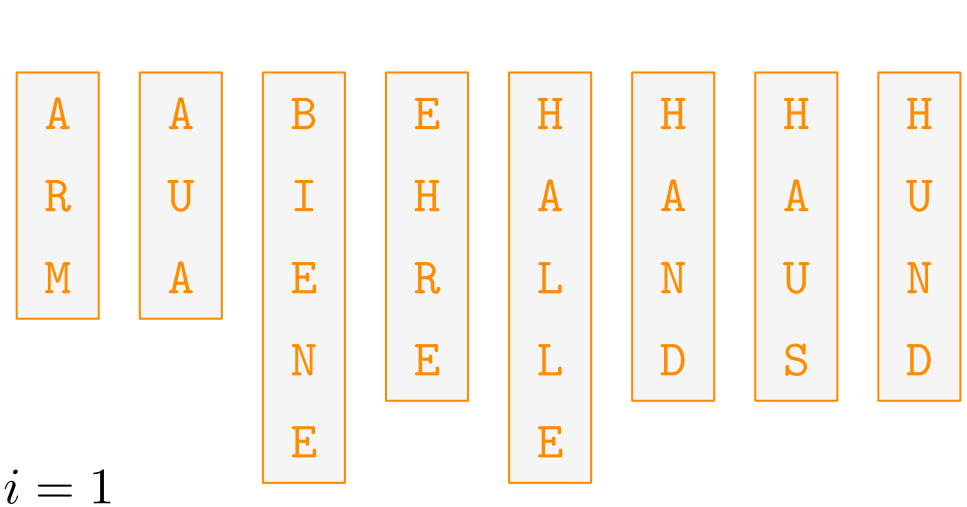


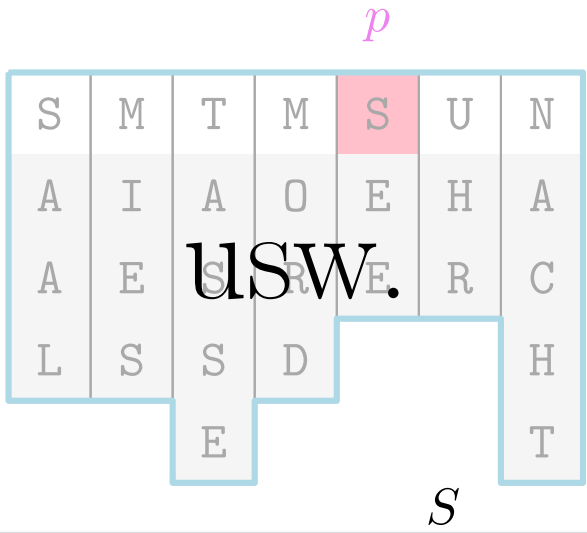


In-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)

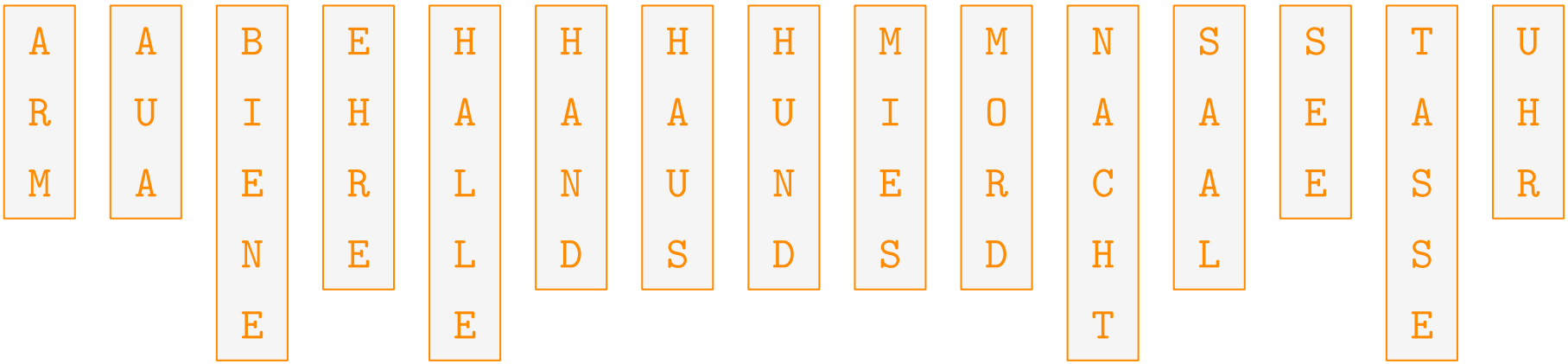




In-place Multikey Quicksort

Wiederholung: Ablauf

Function mkqSort(S: Array of String, i : Integer) : Array of String
  if $|S| \leq 1$ **then return** S (Basisfall)
  choose $p \in S$ uniformly at random (Pivotelement)
  return concatenation of (Rekursion)
    mkqSort($\langle e \in S : e[i] < p[i] \rangle, i$),
    mkqSort($\langle e \in S : e[i] = p[i] \rangle, i+1$),
    mkqSort($\langle e \in S : e[i] > p[i] \rangle, i$)

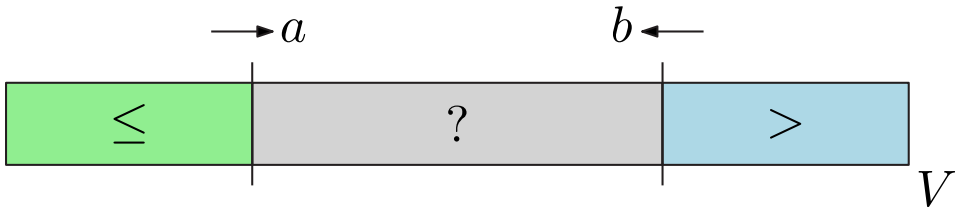


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

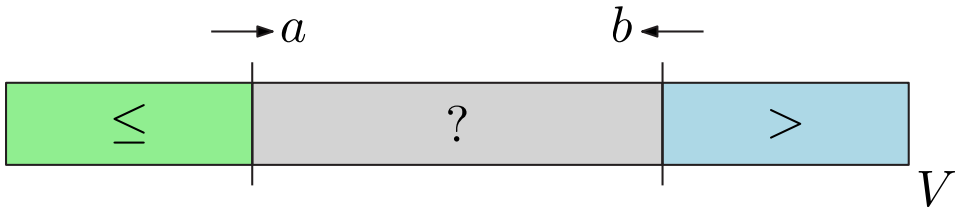


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

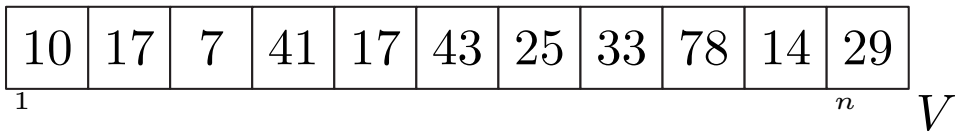


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

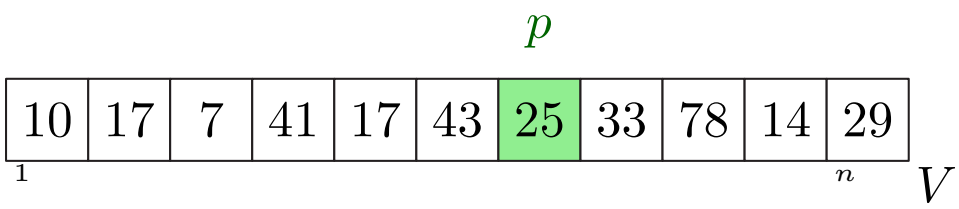


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

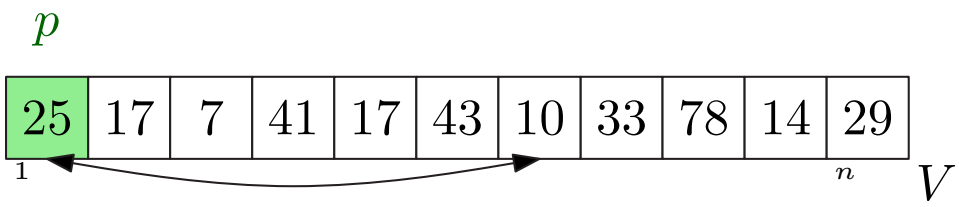


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



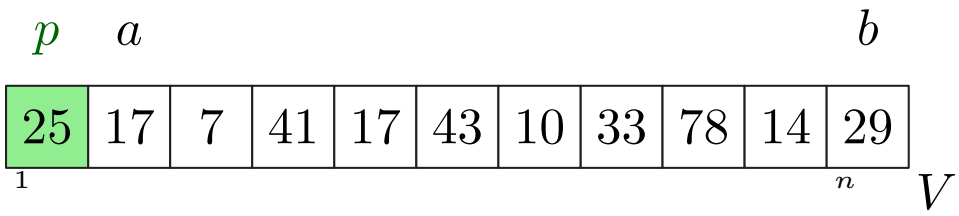
In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$

  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

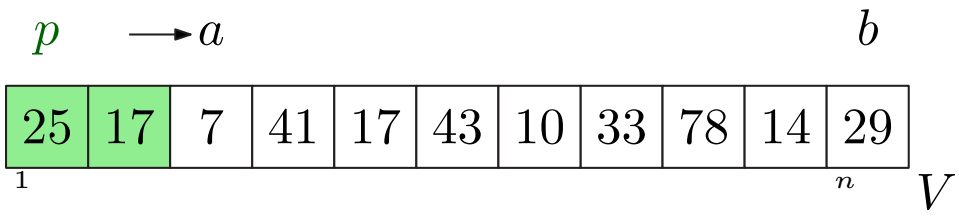


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

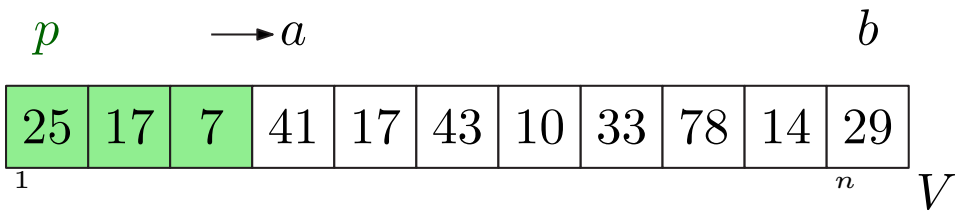


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

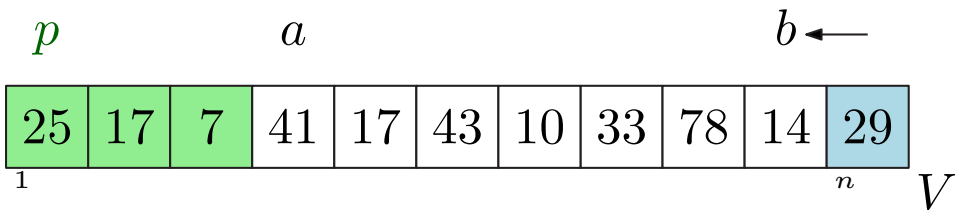


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



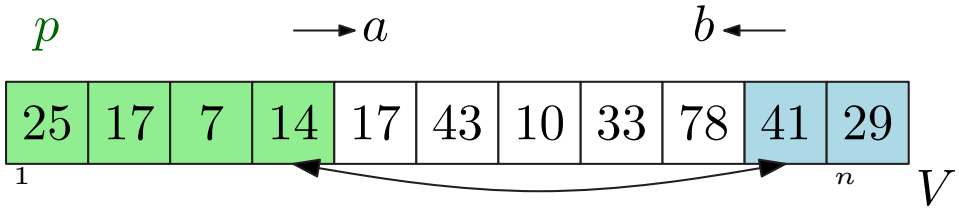
In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$

  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$,
    $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

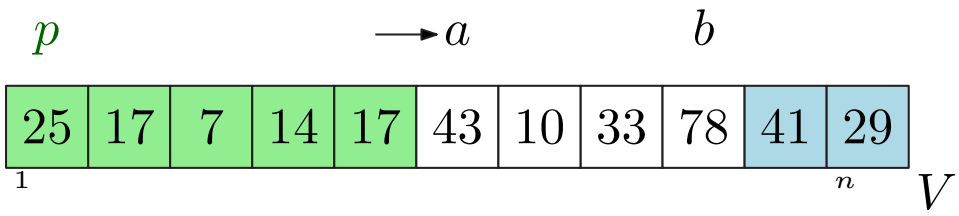


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



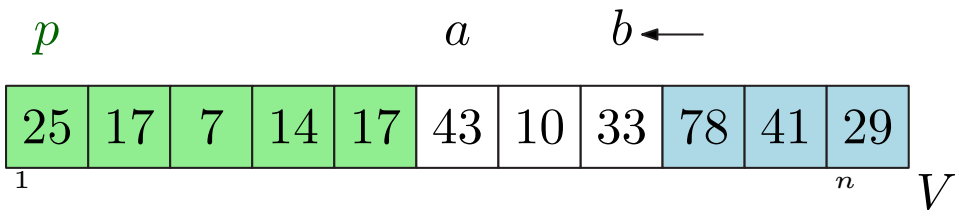
In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen "in die Mitte"
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$

  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$,
    $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

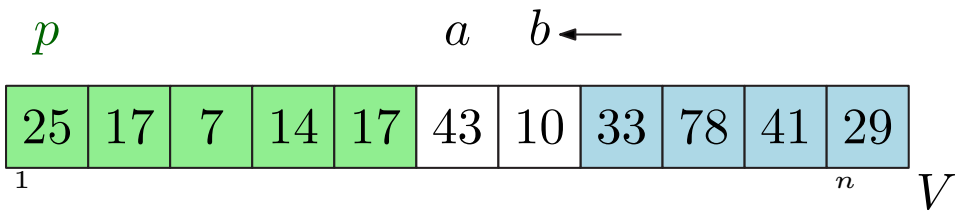


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$

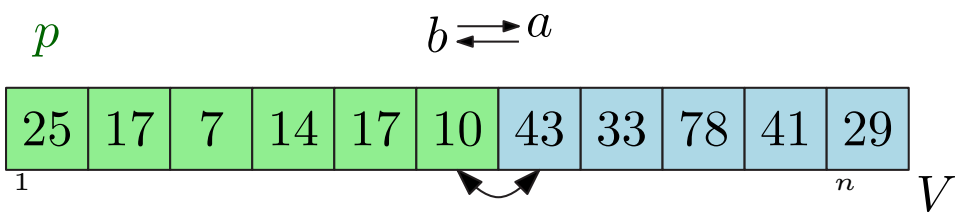


In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$
  - Wähle Pivot p und tausche mit erstem Element, setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$, $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



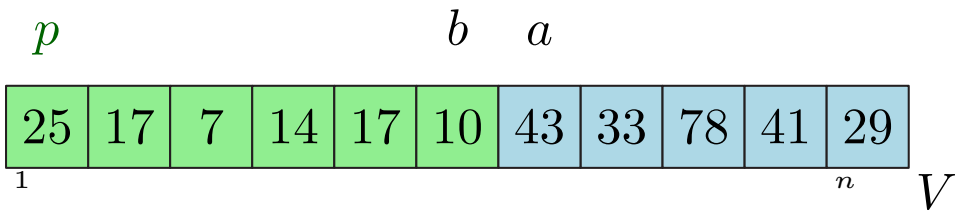
In-place Multikey Quicksort

Wiederholung: Partitionierung

in-place bei Quicksort (für Integer)

- teilt Elemente in kleiner gleich und größer als Pivotelement p
- zwei Zeiger a, b wandern von außen “in die Mitte”
  $\rightarrow$ Invariante: $V[i < a] \leq p$, $V[i > b] > p$

  - Wähle Pivot p und tausche mit erstem Element,
    setze $a = 2, b = n$
  - $a \rightarrow a + 1$, solange $V[a] \leq p$,
    $b \rightarrow b - 1$, solange $V[b] > p$,
  - Tausch, wenn $V[a] > p$ und $V[b] \leq p$
  - Ende, wenn $a > b$



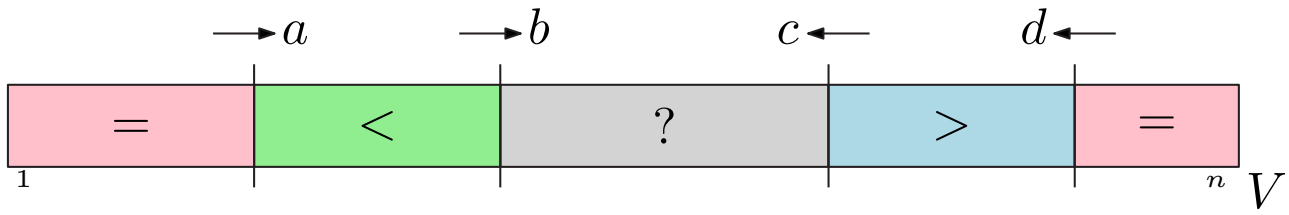
In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger a, d “außen” gesammelt

  $\rightarrow$ Invariante: $V[i \in [a, b) \; a \neq b] < p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d] \; c \neq d] > p$

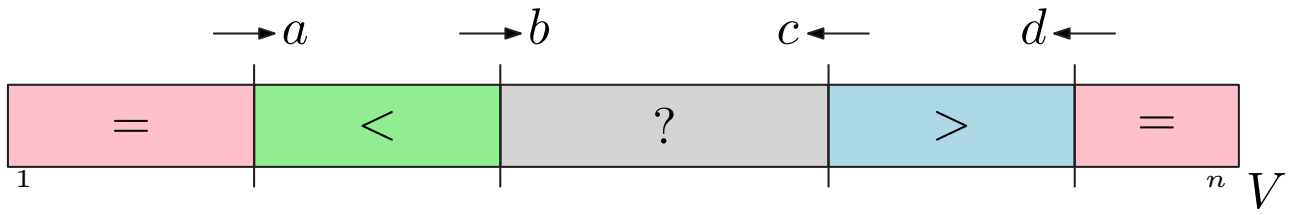


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger a, d “außen” gesammelt
  $\rightarrow$ Invariante: $V[i \in [a, b)\ {}_{a \neq b}] < p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d]\ {}_{c \neq d}] > p$

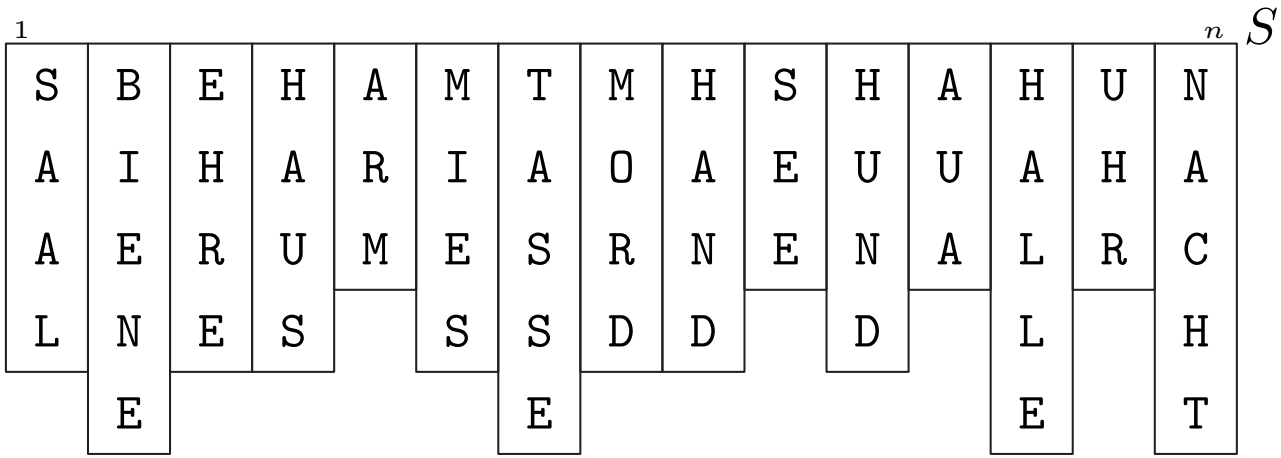


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger a, d “außen” gesammelt
  $\rightarrow$ Invariante: $V[i \in [a, b)\ _{a \neq b}] < p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d]\ _{c \neq d}] > p$

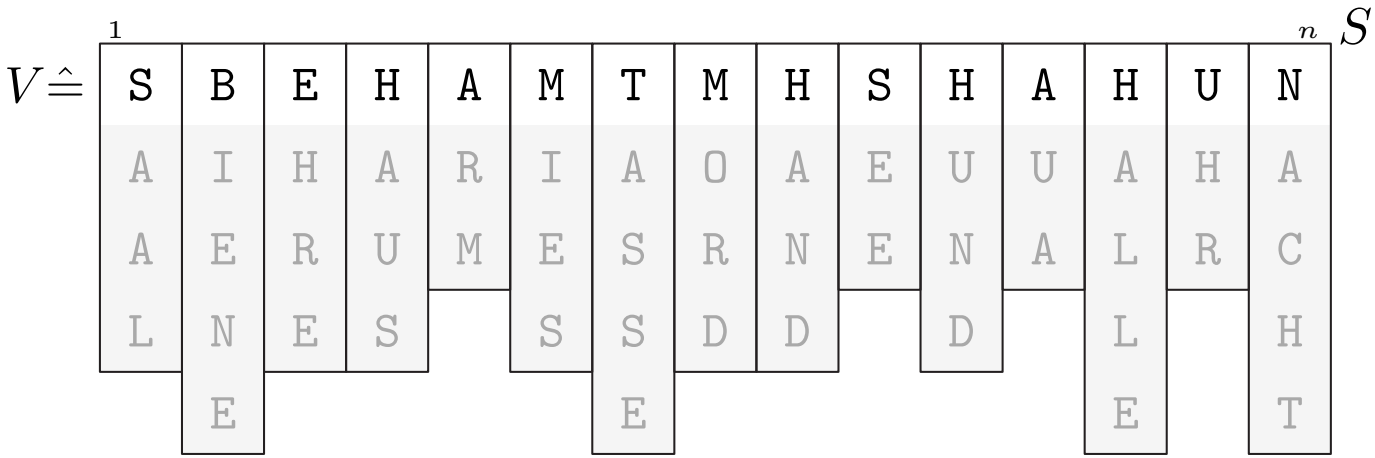


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort

- teilt Elemente in kleiner, gleich und größer als Pivotelement p
- zwei Zeiger b, c wandern von außen “in die Mitte”
- gleiche Elemente werden mit Zeiger a, d “außen” gesammelt
  $\rightarrow$ Invariante: $V[i \in [a, b)\ _{a \neq b}] < p$, $V[i < a \vee i > d] = p$, $V[i \in (c, d]\ _{c \neq d}] > p$

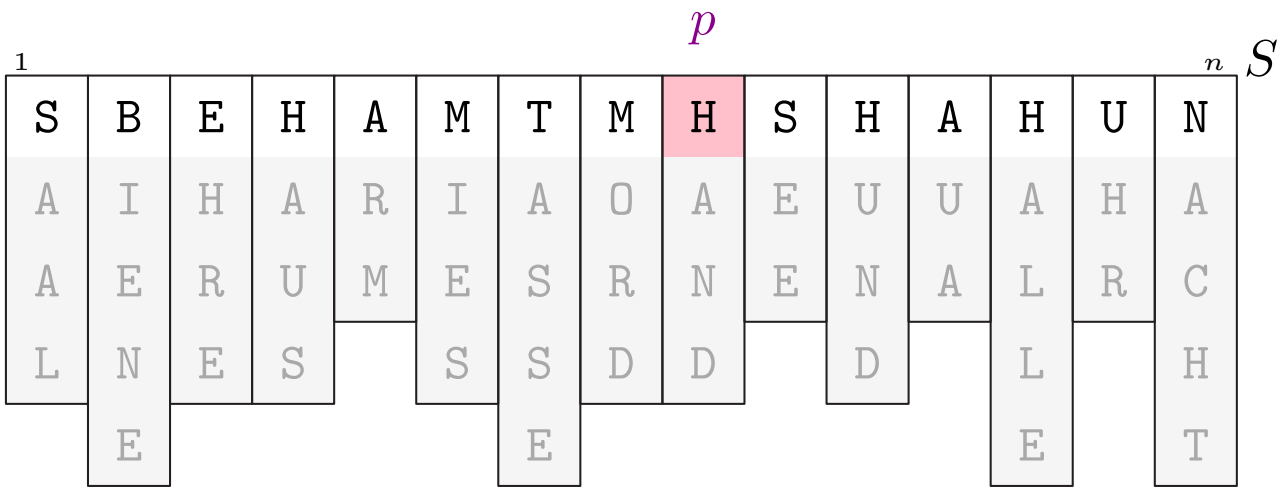


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

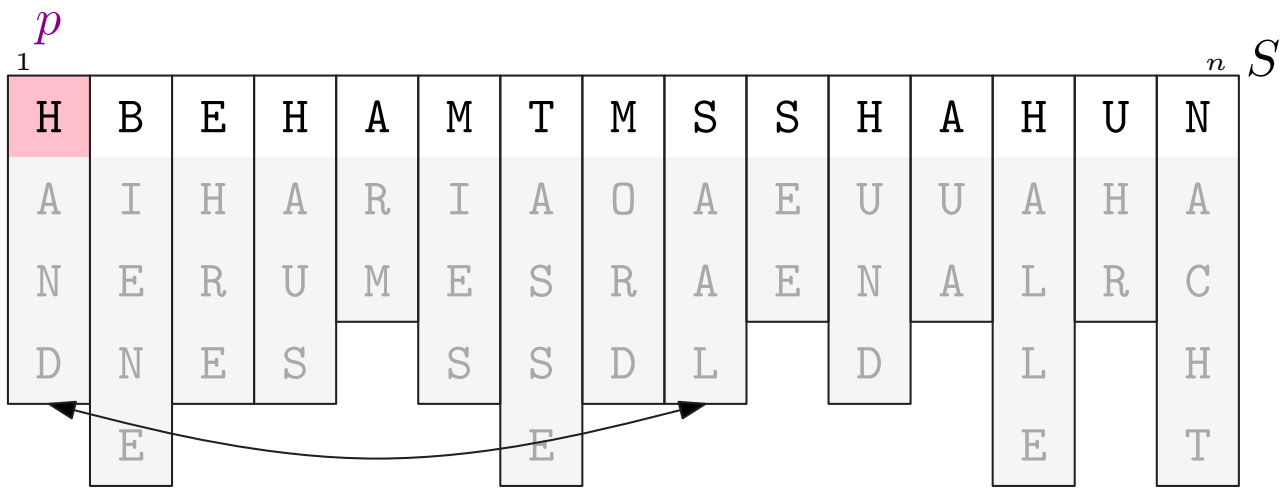


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

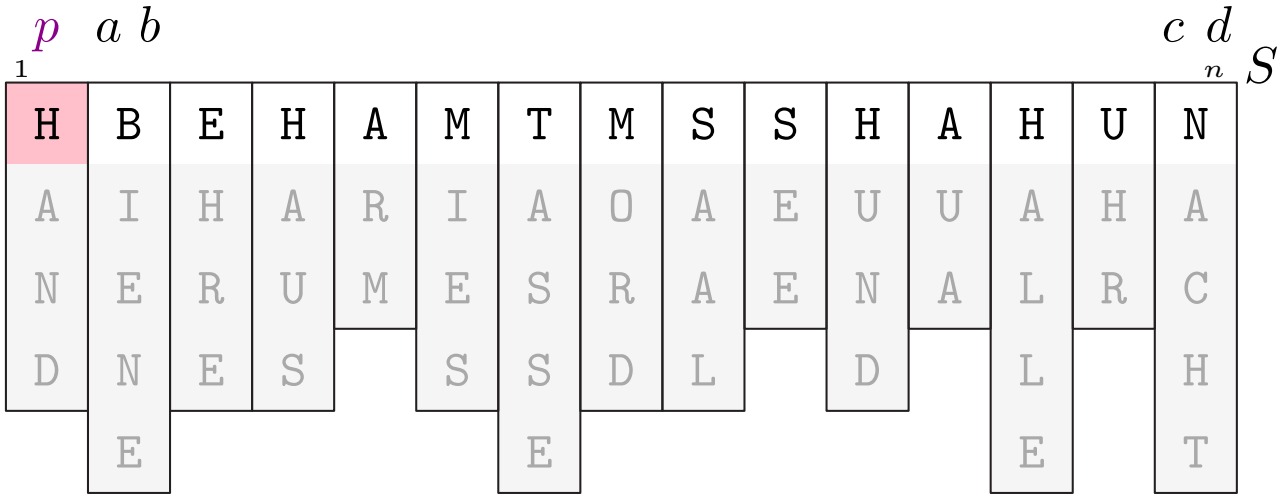


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

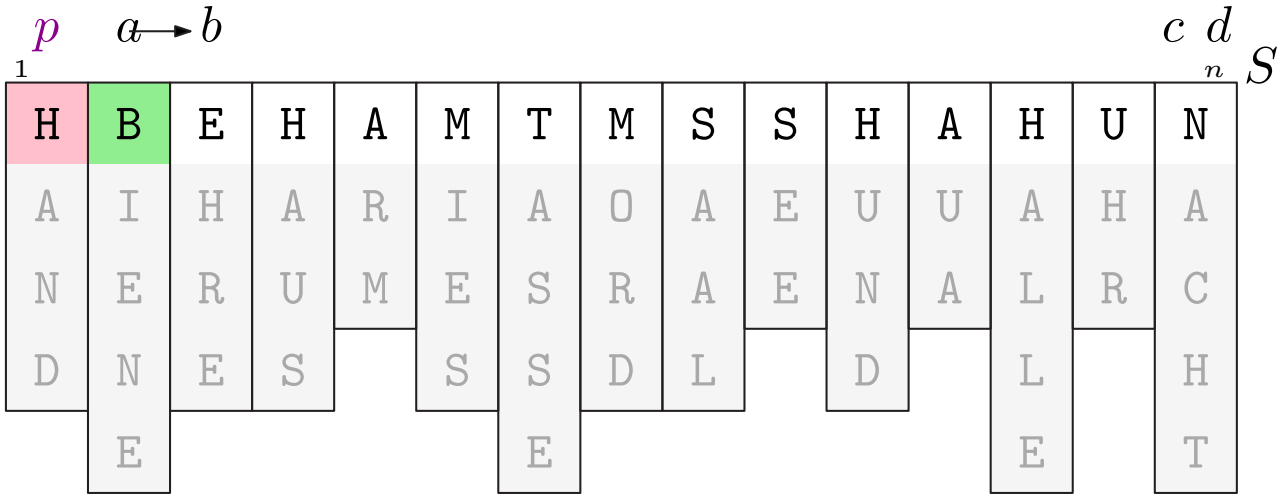


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

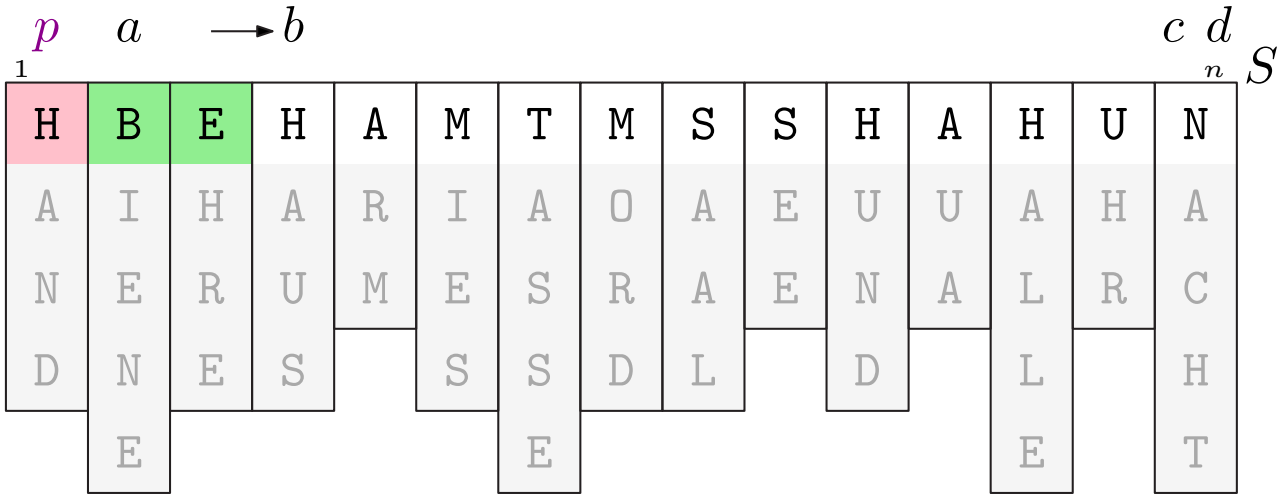


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

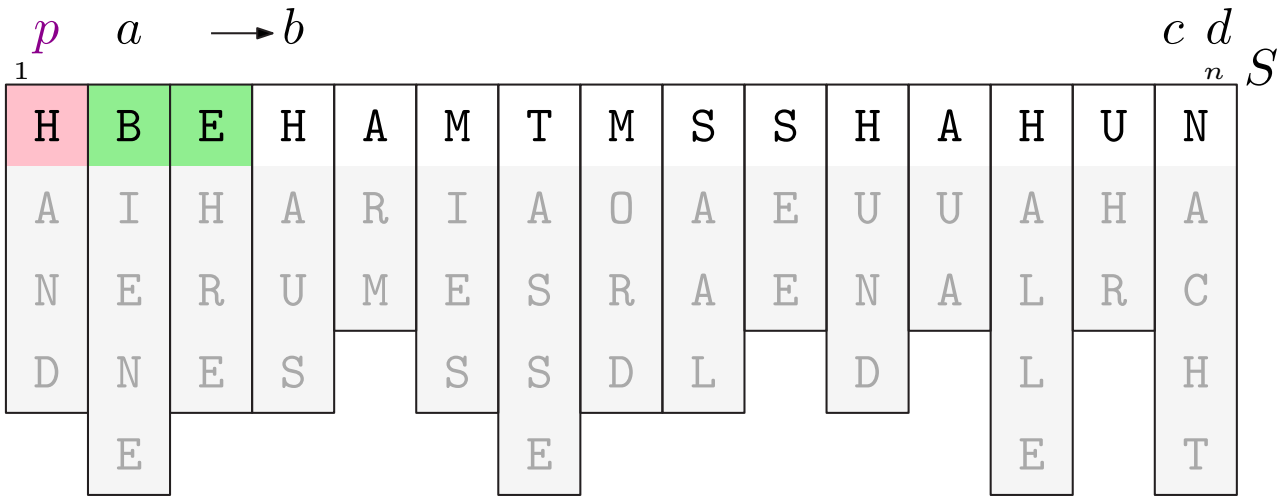


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

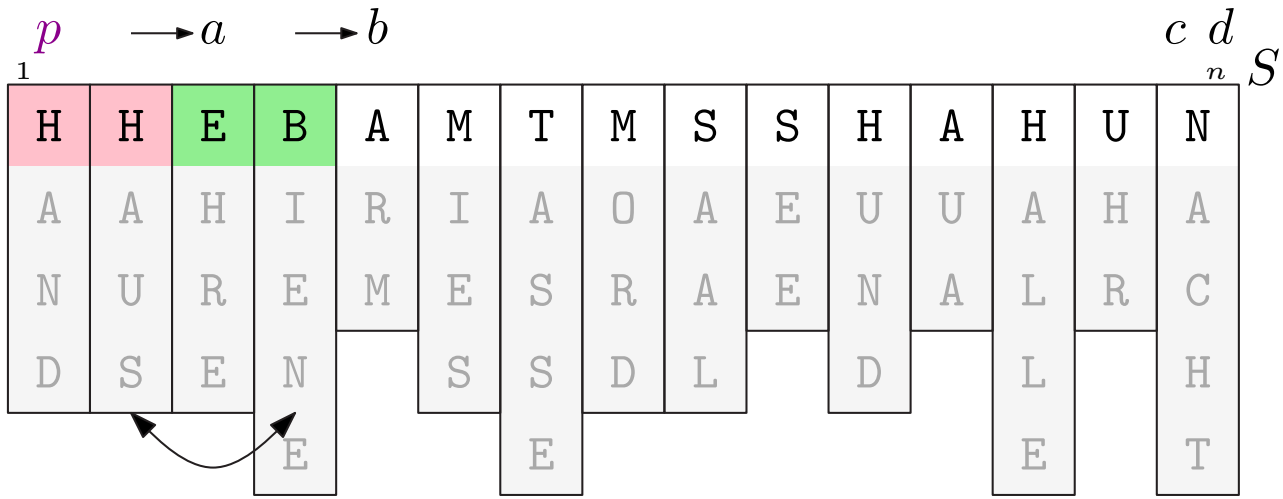


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

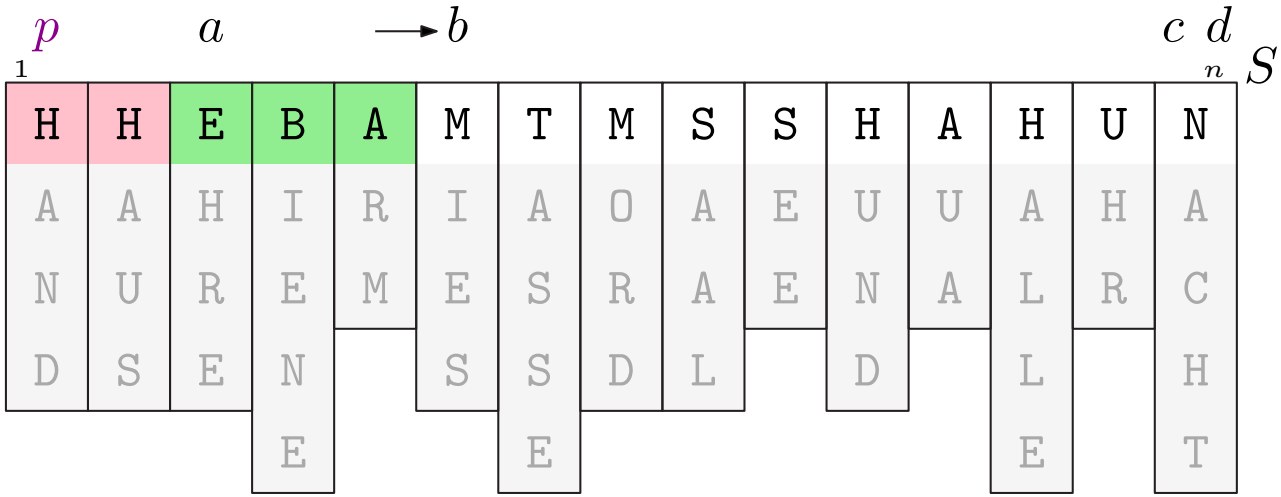


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

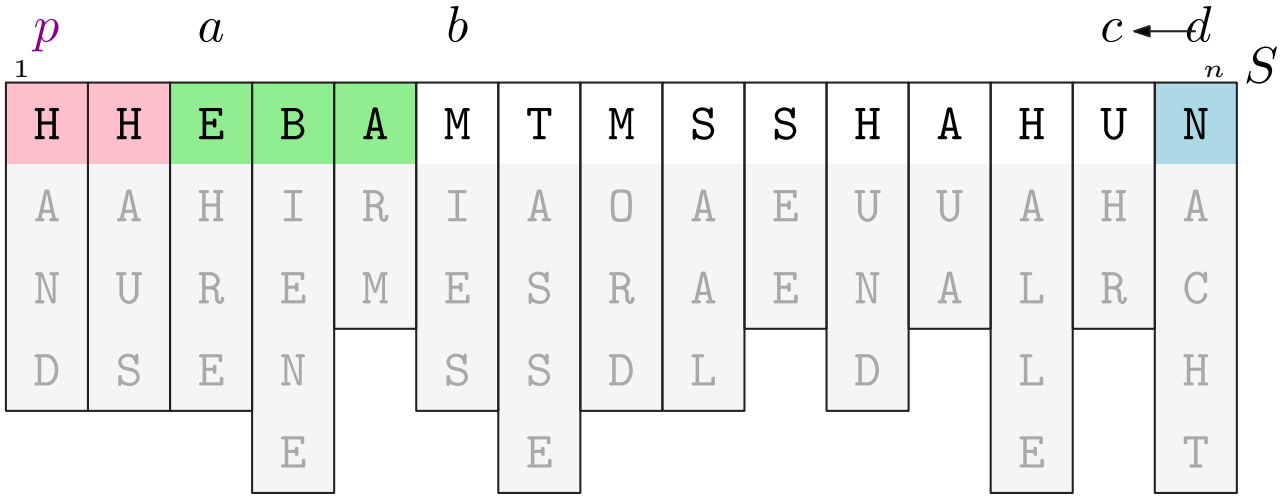


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

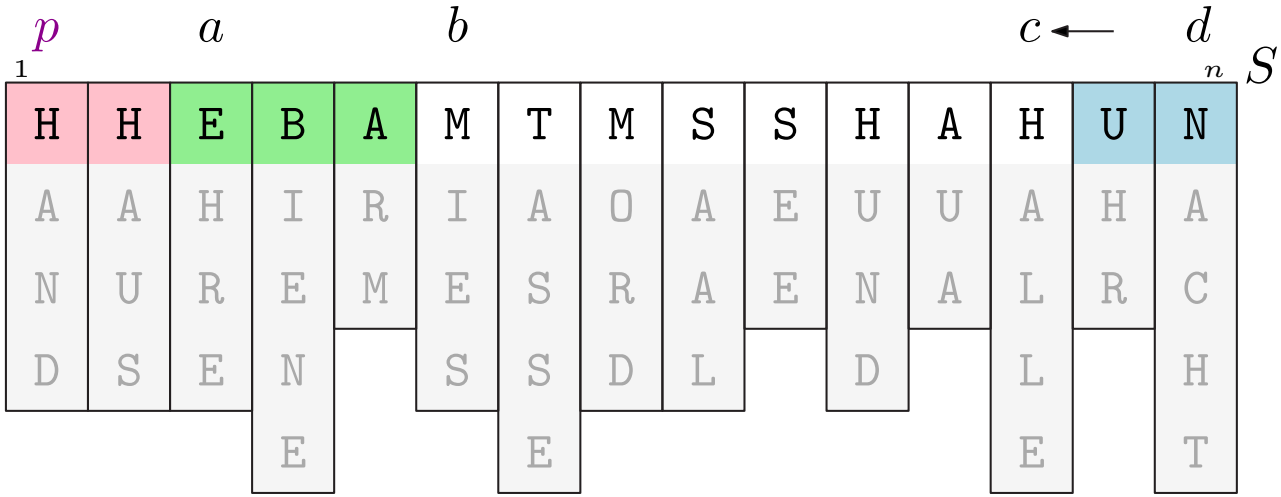


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

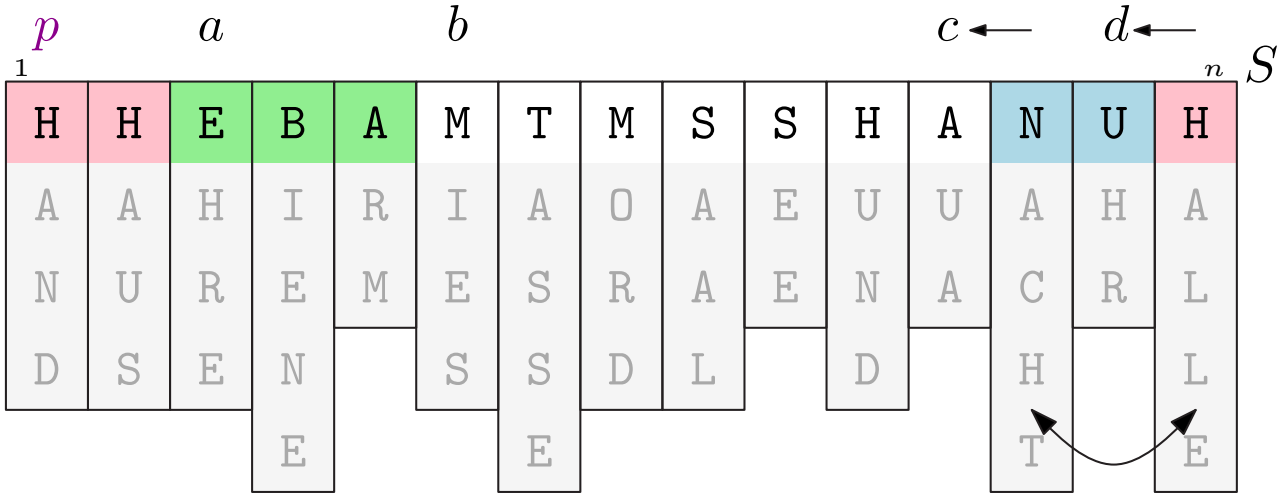


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

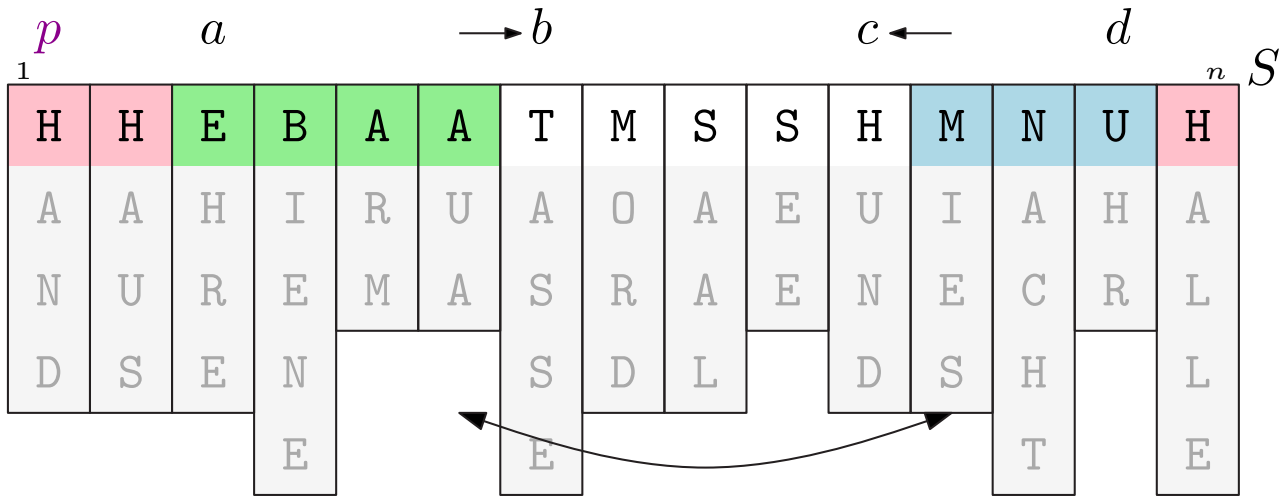


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

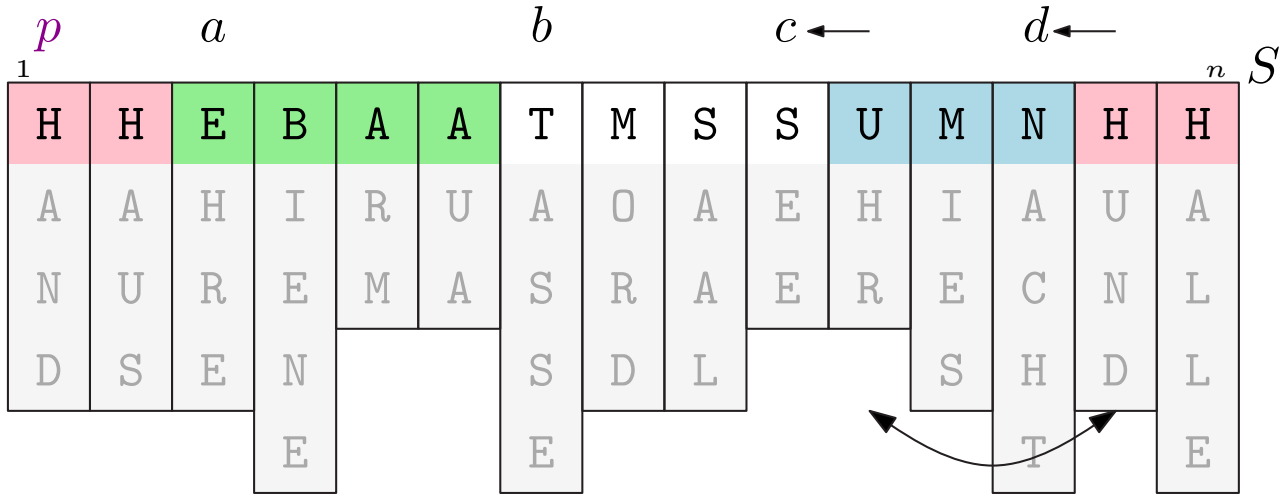


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

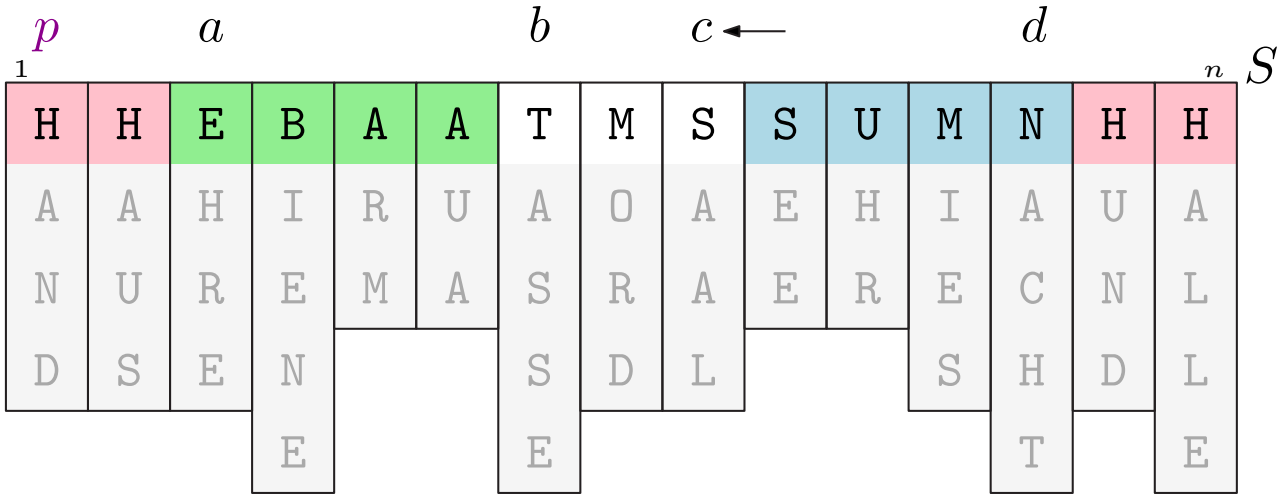


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

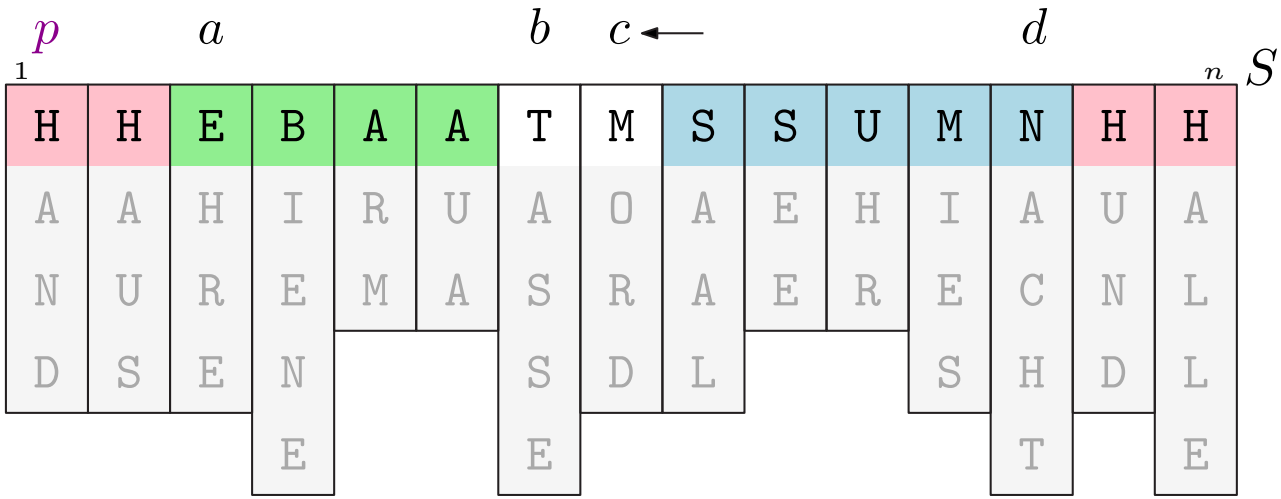


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

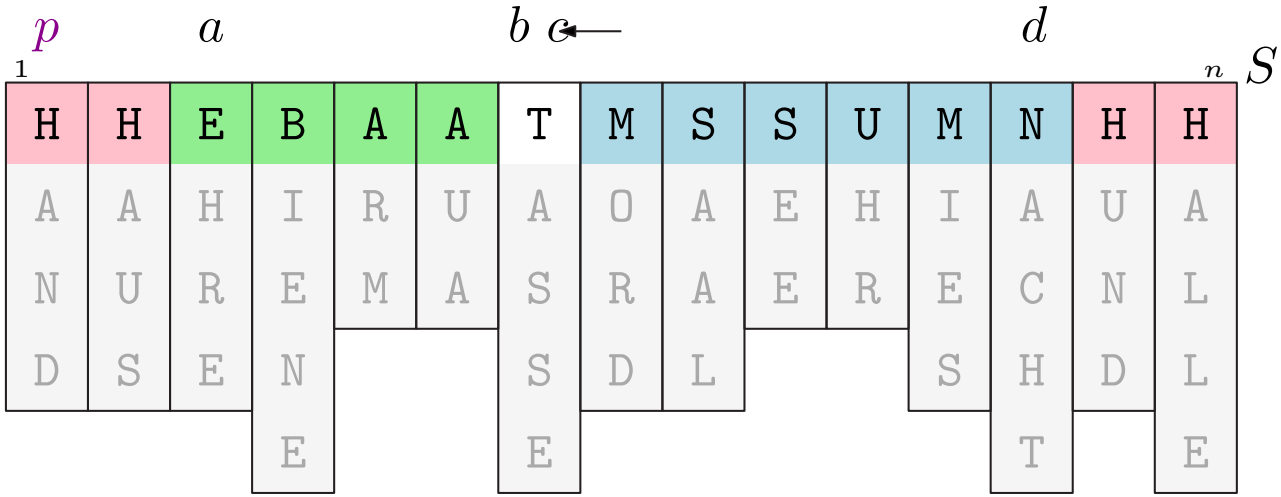


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

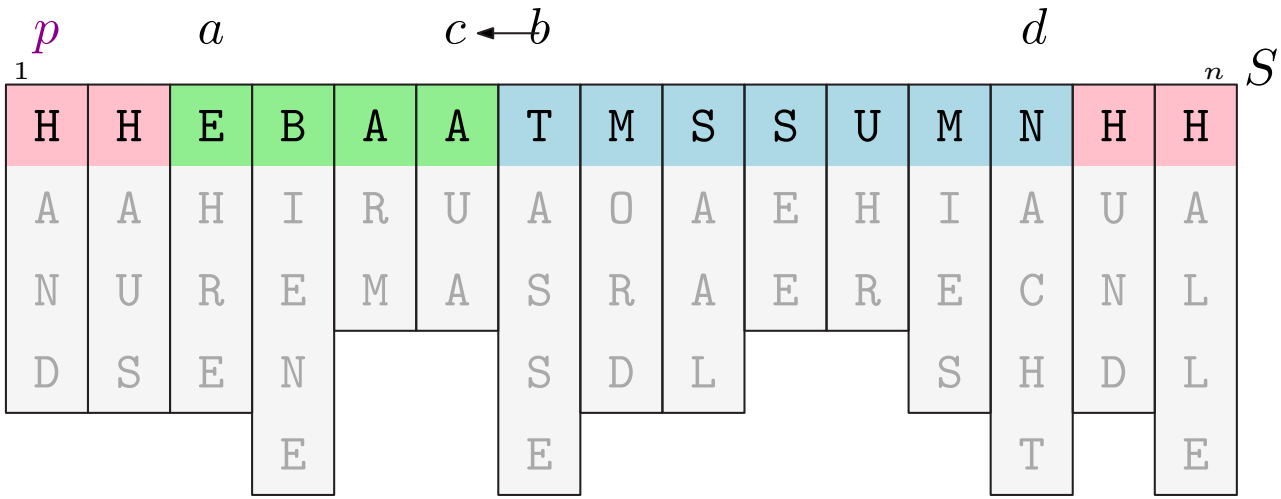


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

- Wähle Pivot p und tausche mit erstem Element, setze $a = b = 2, c = d = n$
- $b \to b+1$, solange $V[b] \leq p$, wenn $V[b] = p$: Tausch mit $V[a]$, $a \to a+1$, $c \to c-1$, solange $V[c] \geq p$, wenn $V[c] = p$: Tausch mit $V[d]$, $d \to d-1$
- Tausch, wenn $V[b] > p$ und $V[c] < p$
- Ende, wenn $b > c$

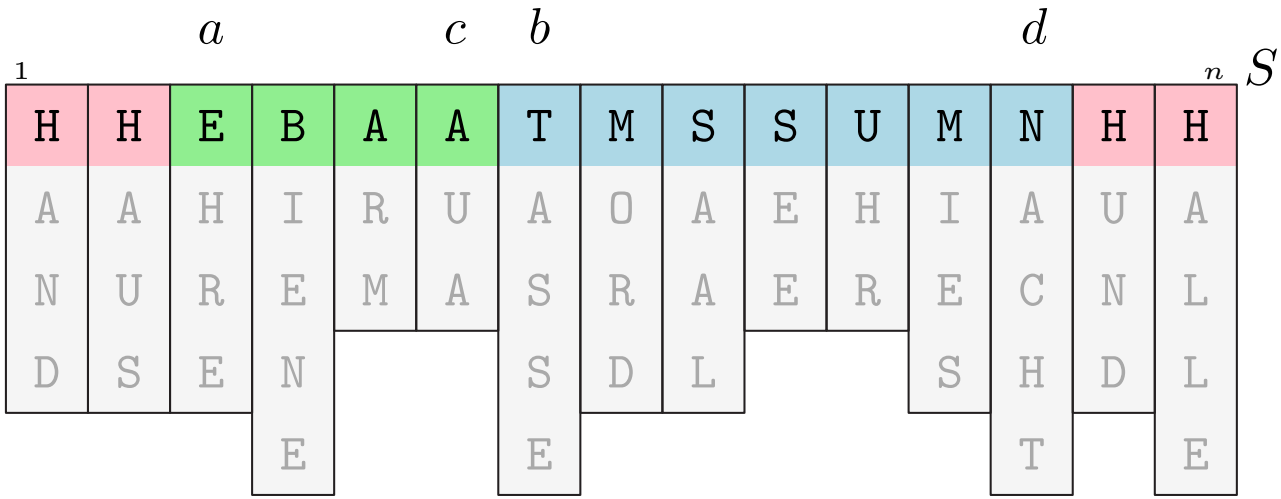


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a-1, b-a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b-r, b)$
- $r = \min(d-c, n-d)$
  Tausch von r Zeichen zwischen $[c+1, c+r)$ und $[n-r+1, n+1)$

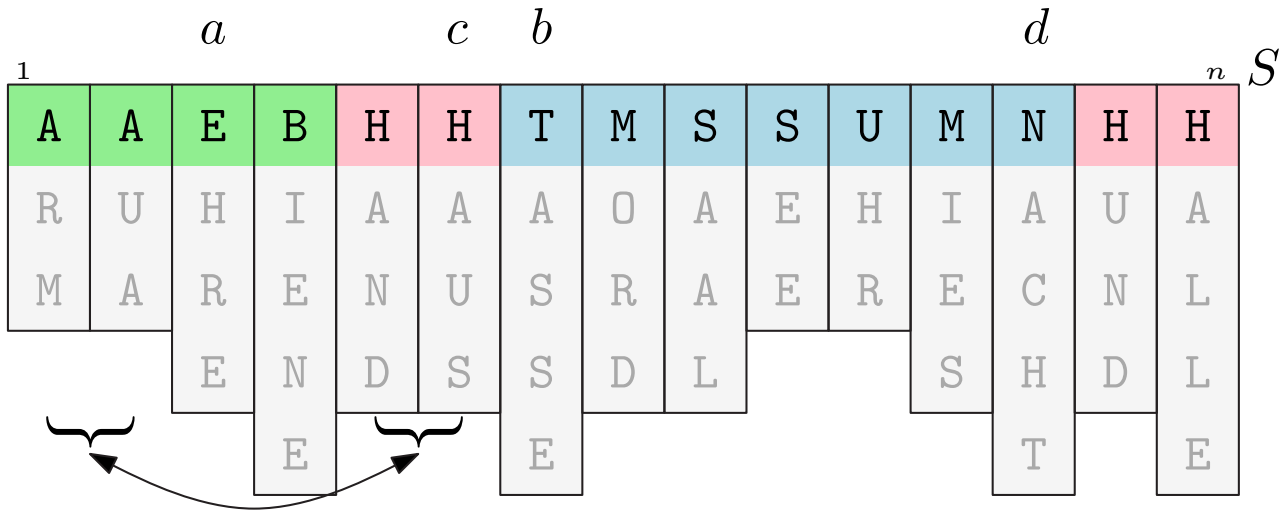


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a-1, b-a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b-r, b)$
- $r = \min(d-c, n-d)$
  Tausch von r Zeichen zwischen $[c+1, c+r)$ und $[n-r+1, n+1)$

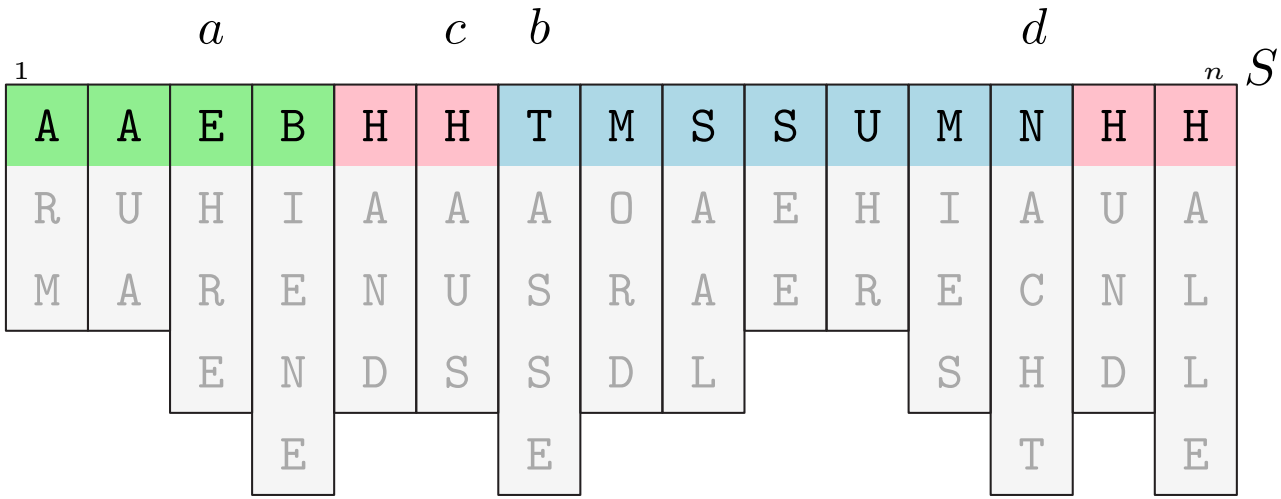


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a-1, b-a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b-r, b)$
- $r = \min(d-c, n-d)$
  Tausch von r Zeichen zwischen $[c+1, c+r)$ und $[n-r+1, n+1)$

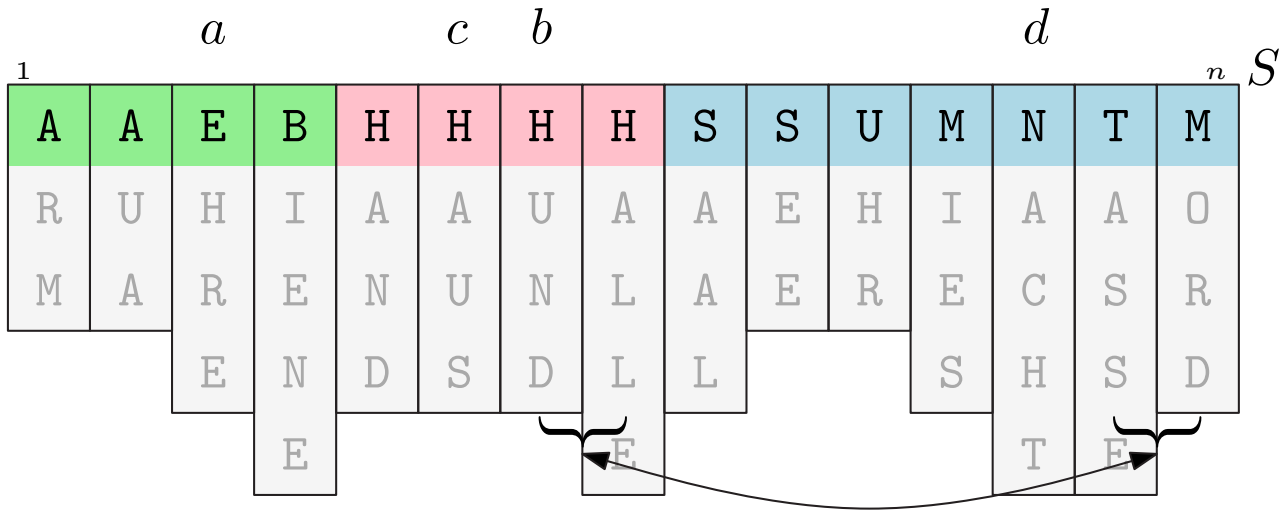


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a-1, b-a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b-r, b)$
- $r = \min(d-c, n-d)$
  Tausch von r Zeichen zwischen $[c+1, c+r)$ und $[n-r+1, n+1)$

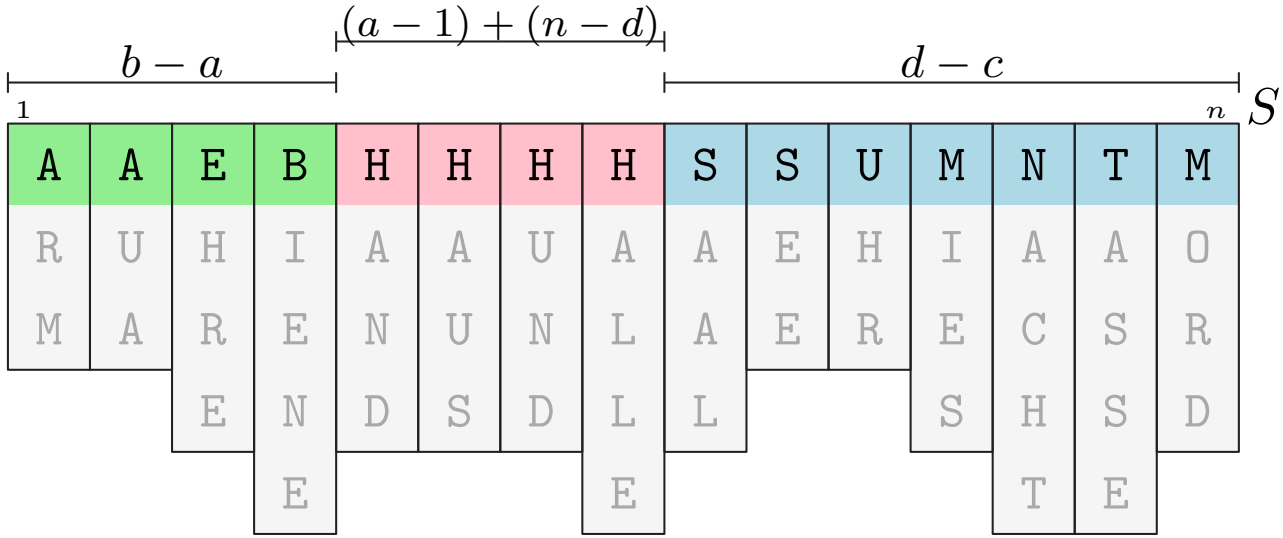


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a-1, b-a)$
  Tausch von r Zeichen zwischen $[1, r)$ und $[b-r, b)$
- $r = \min(d-c, n-d)$
  Tausch von r Zeichen zwischen $[c+1, c+r)$ und $[n-r+1, n+1)$



In-place Multikey Quicksort

Zusammenfassung

- *Three-way Radix Quicksort*
  (Partitionierung in kleiner, gleich, größer über alle Stellen analog zu msd-*Radixsort*)
- effizient $\mathcal{O}(|S| \log |S| + d)$
  ($d \triangleq$ Summe der Länge der unterscheidenden Präfixe)
- *in-place* Partitionierung möglich
  (durch geschicktes Speichern und Verschieben der gleichen Elemente)
- sehr einfache Implementierung
  (nur 40 Zeilen Quellcode, siehe Anhang)

Suche mit Border-Array

Motivation

- We know how to search for a string P in a text T using the Knuth-Morris-Pratt algorithm
- Now we will consider how to find matches without preprocessing of P.

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0																		

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1																	

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0																

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1															

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1	0														

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1	0	0													

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1	0	0	1												

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1	0	0	1	2											

Suche mit Border-Array

Concatenation with the special symbol

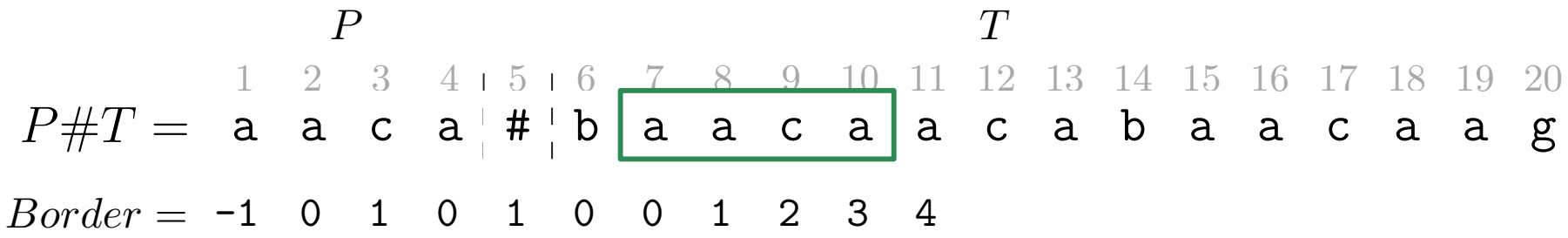
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.

		P					T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P\#T =$	a	a	c	a	#	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1	0	0	1	2	3										

Suche mit Border-Array

Concatenation with the special symbol

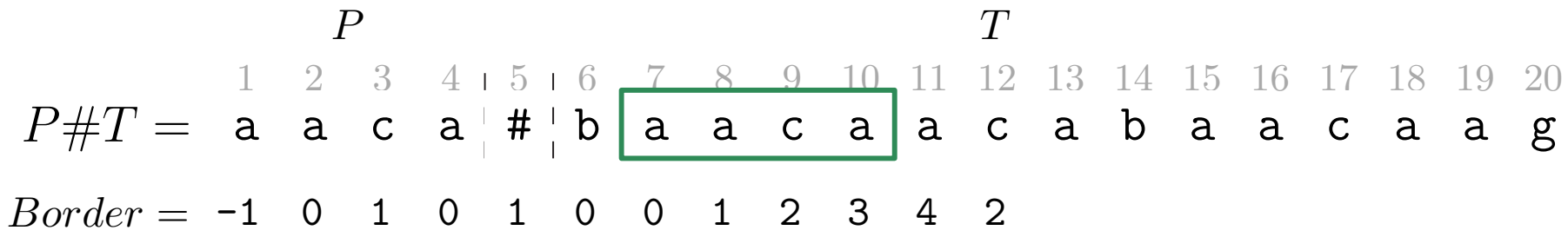
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

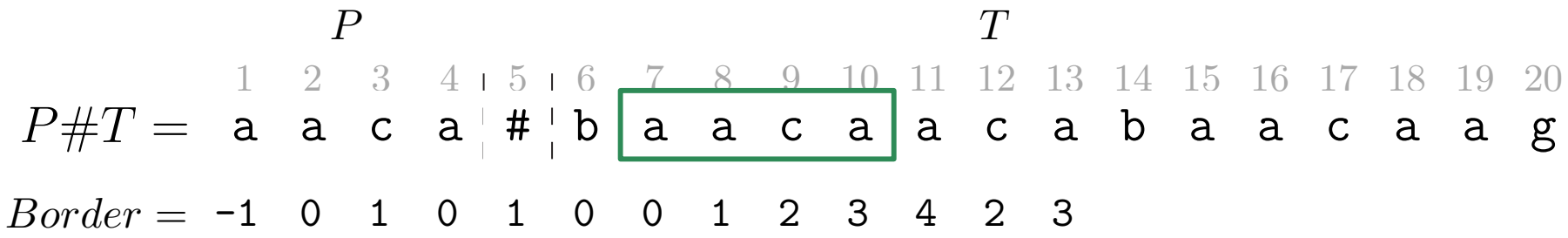
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

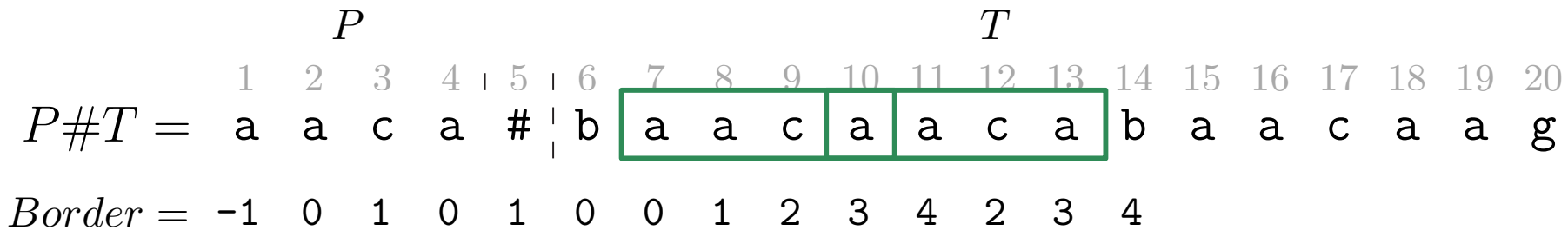
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

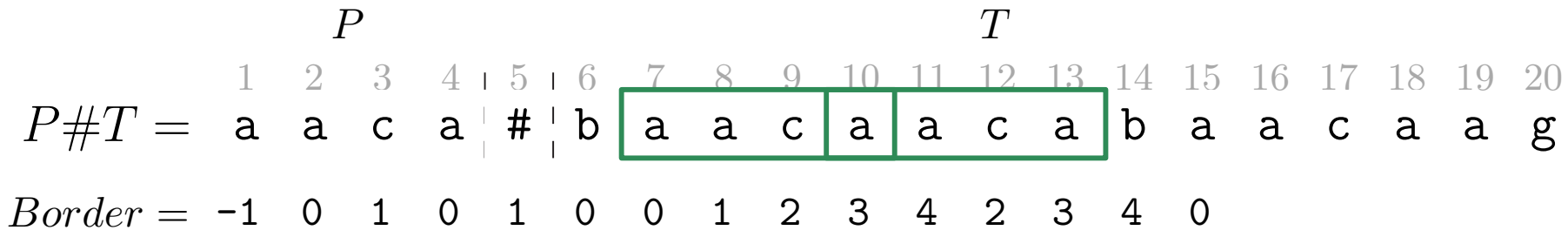
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

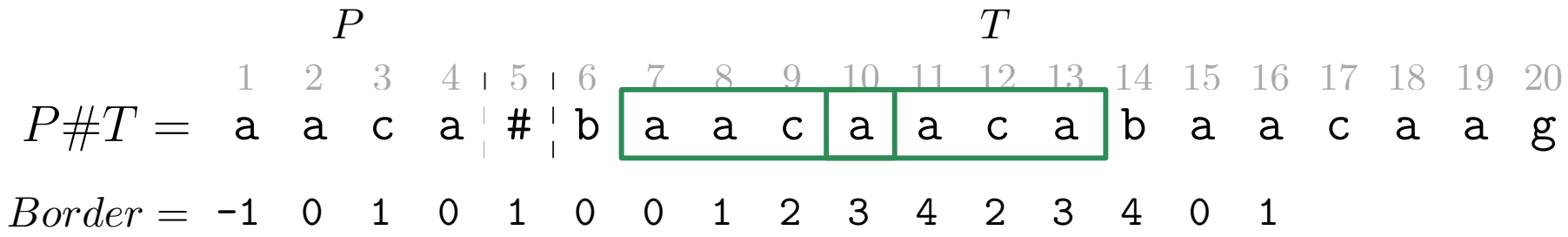
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

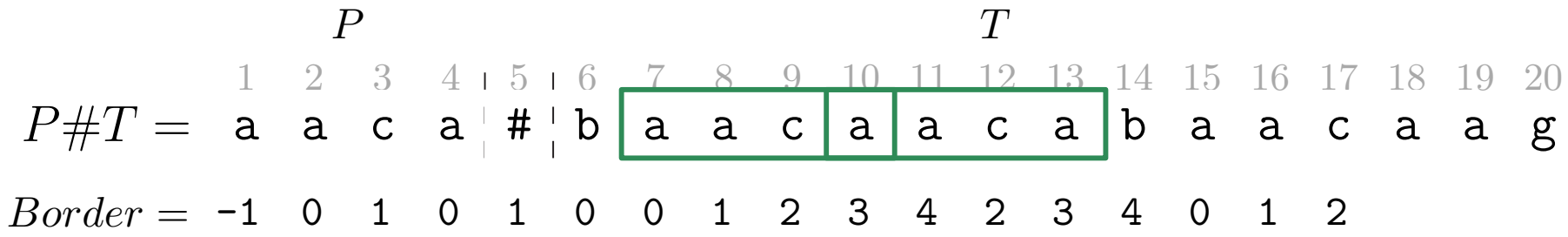
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

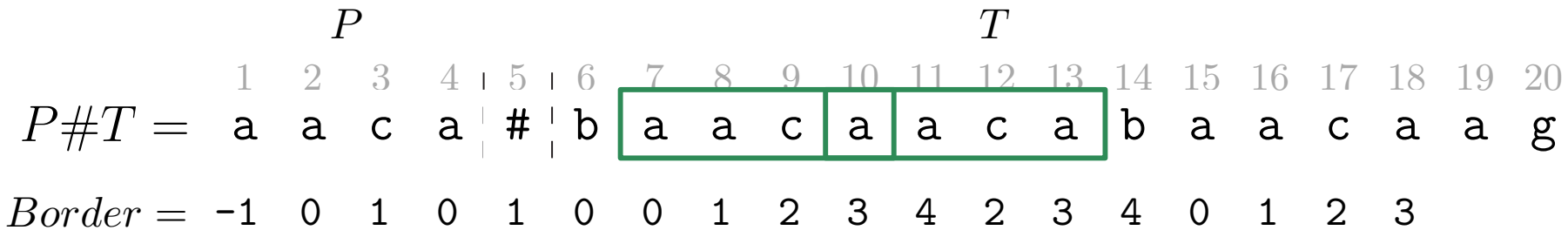
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

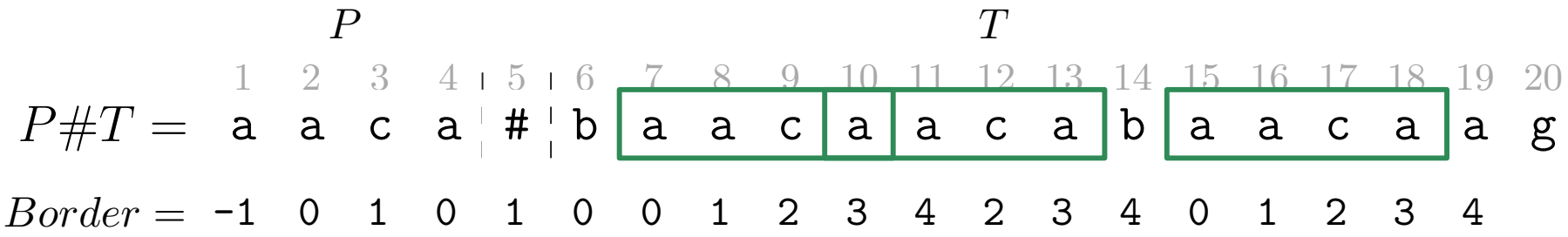
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

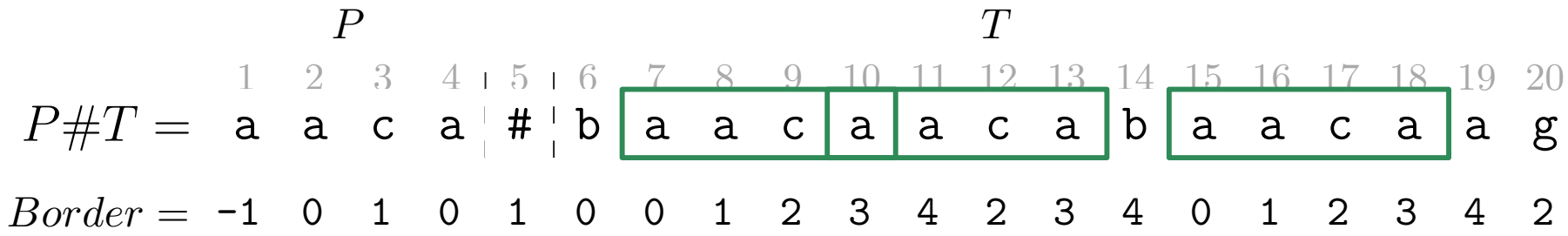
- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$.
- Consider a string $P\#T$, where $\#$ is a special symbol that does not occur in P and T.
- Let us calculate a border array for it. Recall that $Border[j]$ is longest prefix of $P[1 \ldots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \ldots j-1]$.



Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)

  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

P T

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$PT =$	a	a	c	a	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0																	

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

P T

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$PT =$	a	a	c	a	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0																	

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

P T

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$PT =$	a	a	c	a	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1																
$Checked =$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

P T

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$PT =$	a	a	c	a	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0															
$Checked =$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

P T

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$PT =$	a	a	c	a	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1														
$Checked =$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

P T

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$PT =$	a	a	c	a	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1	0													
$Checked =$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

P T

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$PT =$	a	a	c	a	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1	0	1												
$Checked =$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

P T

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$PT =$	a	a	c	a	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1	0	1	2											
$Checked =$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Suche mit Border-Array

Concatenation with the special symbol

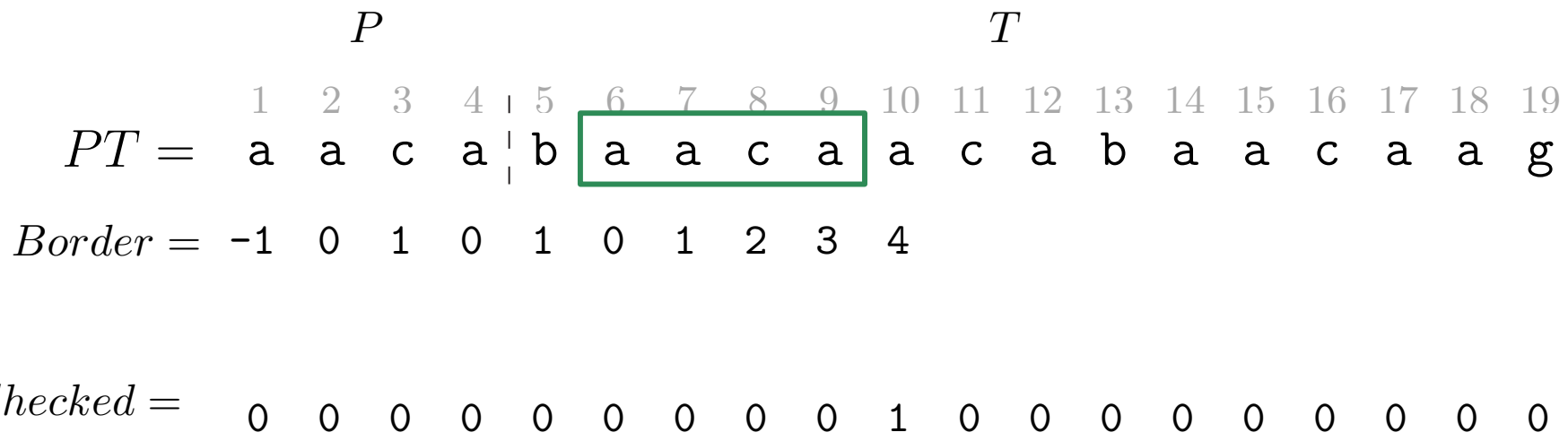
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*

		P				T													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$PT =$	a	a	c	a	b	a	a	c	a	a	c	a	b	a	a	c	a	a	g
$Border =$	-1	0	1	0	1	0	1	2	3										
$Checked =$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Suche mit Border-Array

Concatenation with the special symbol

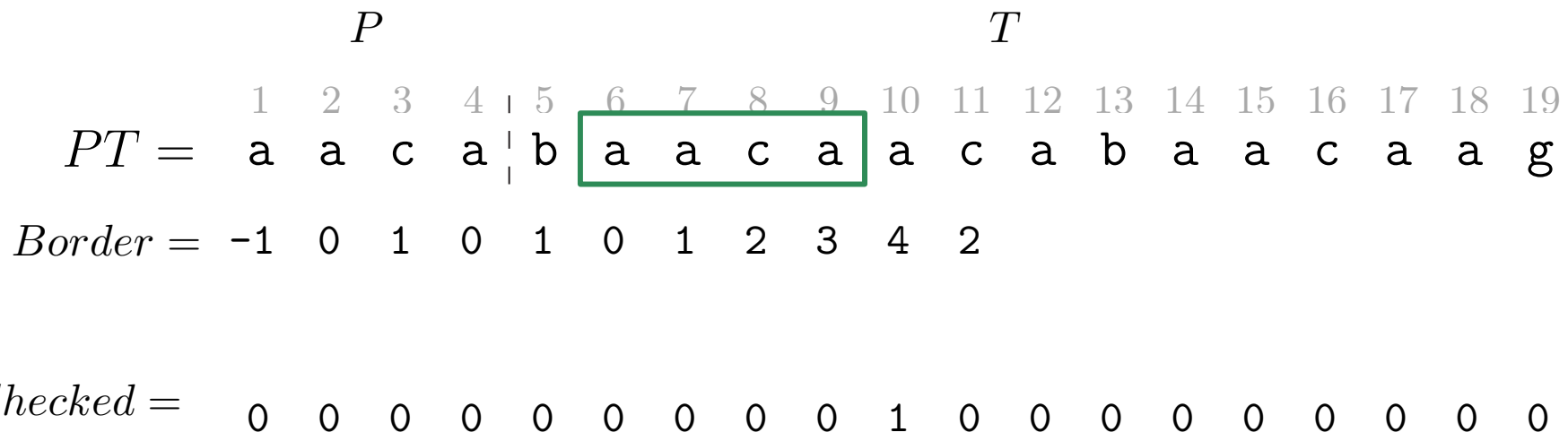
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

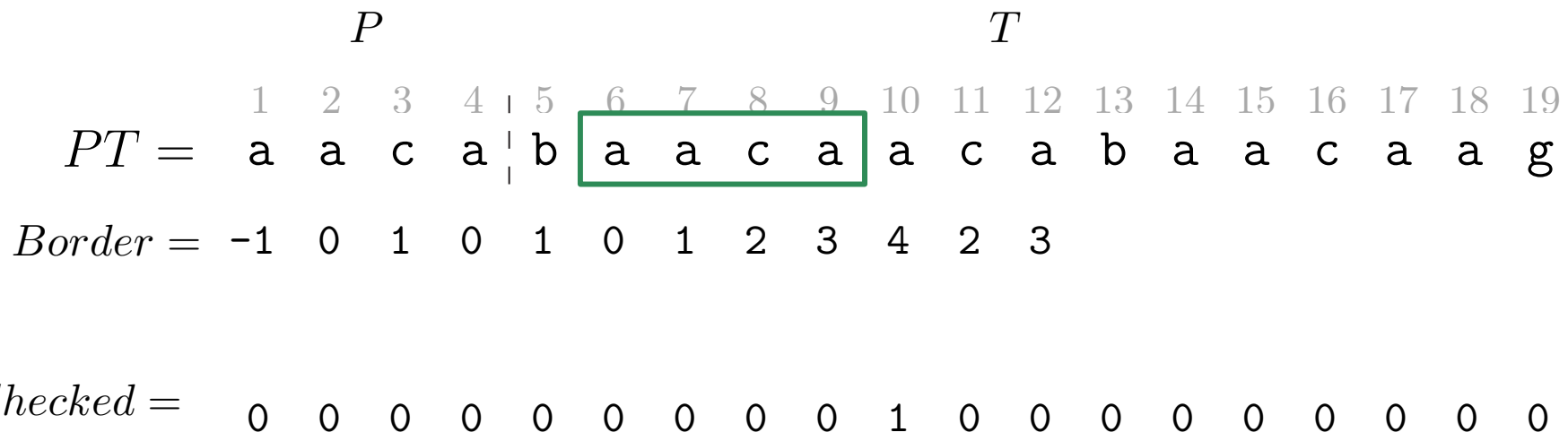
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

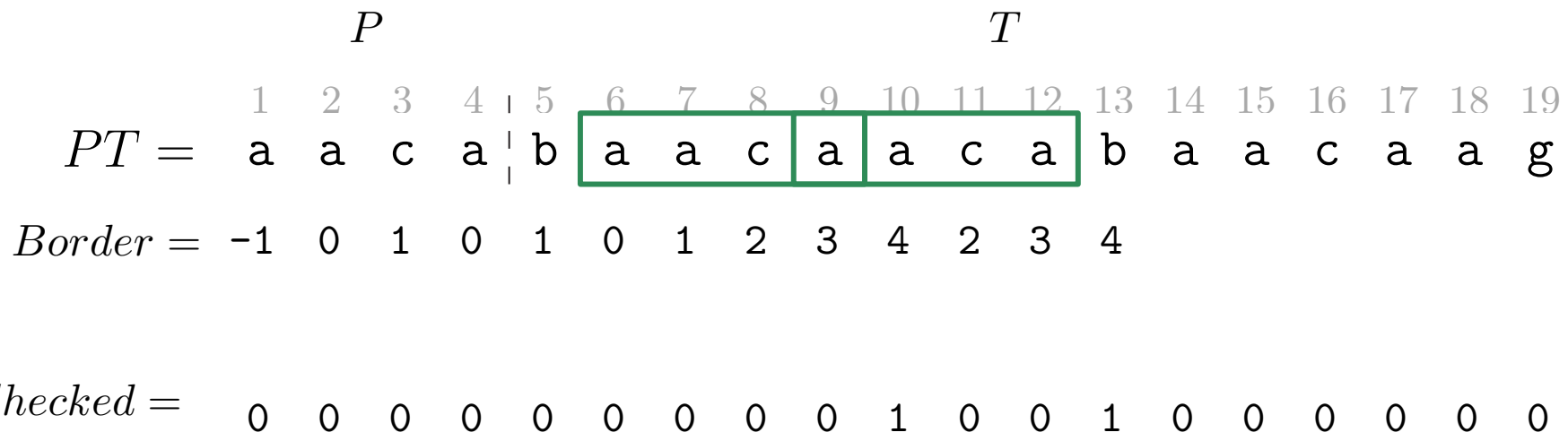
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

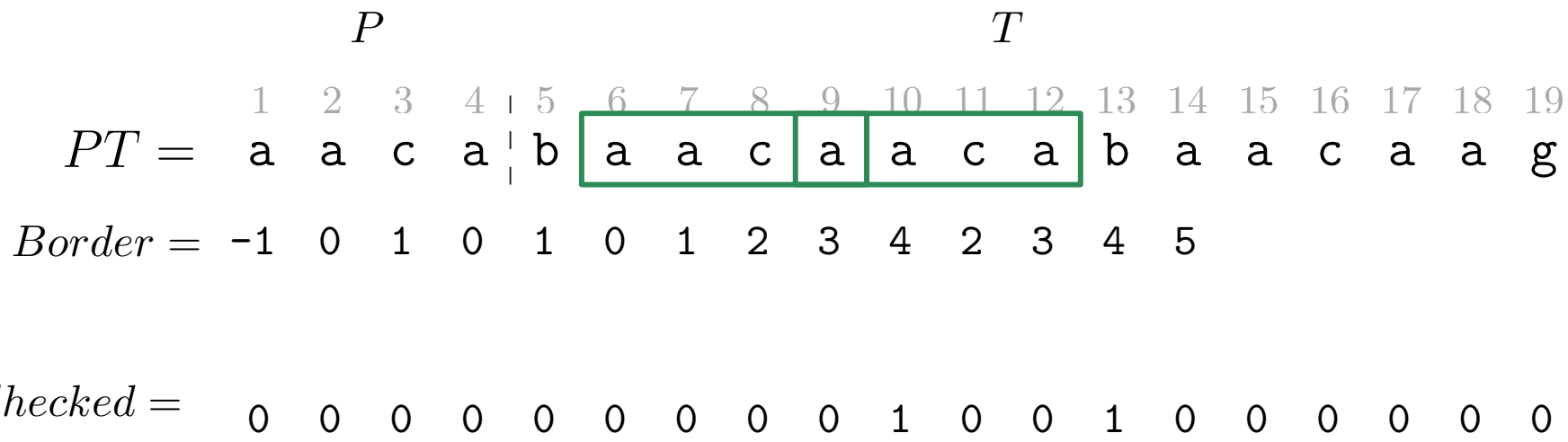
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

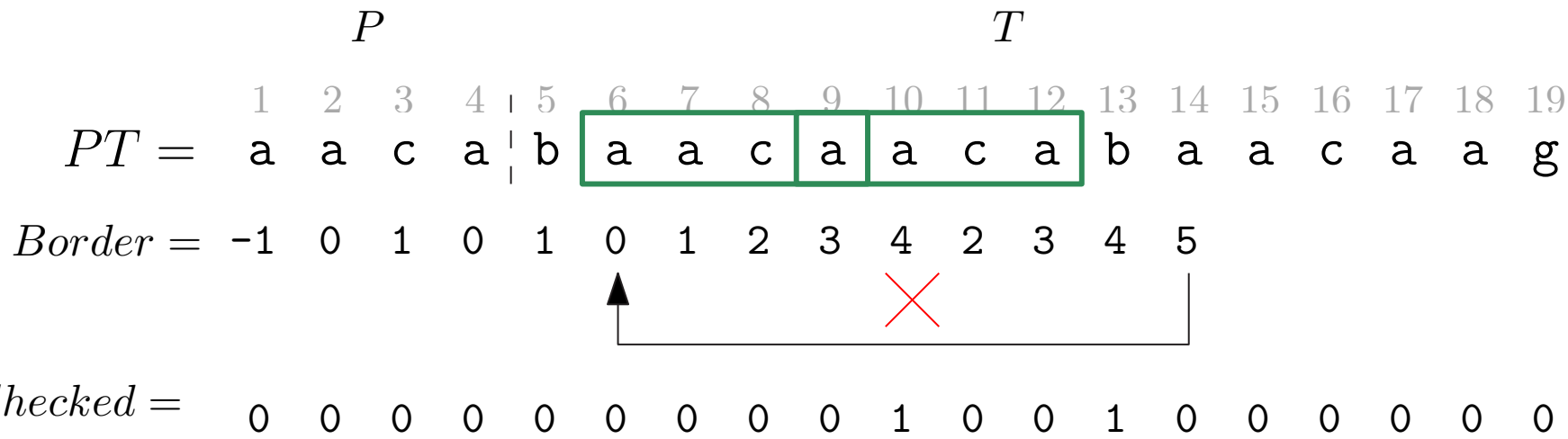
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

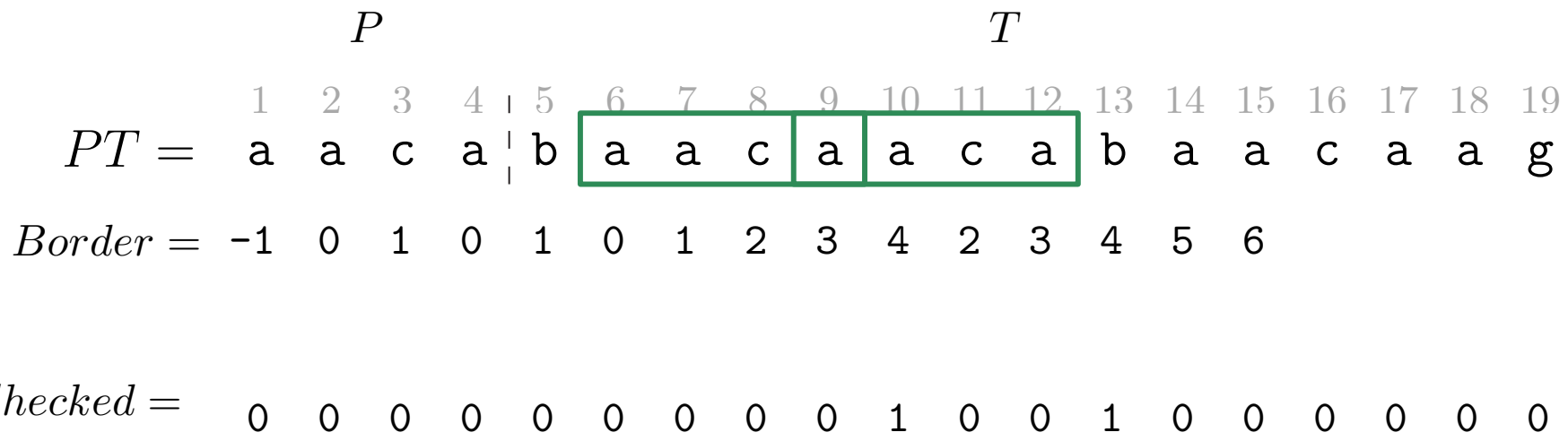
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

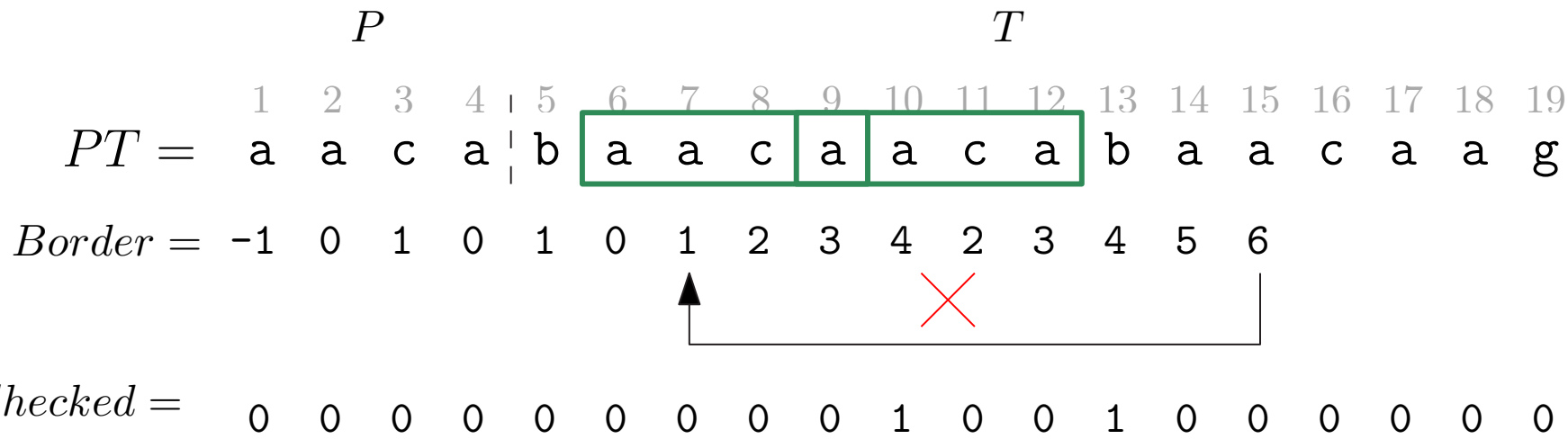
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

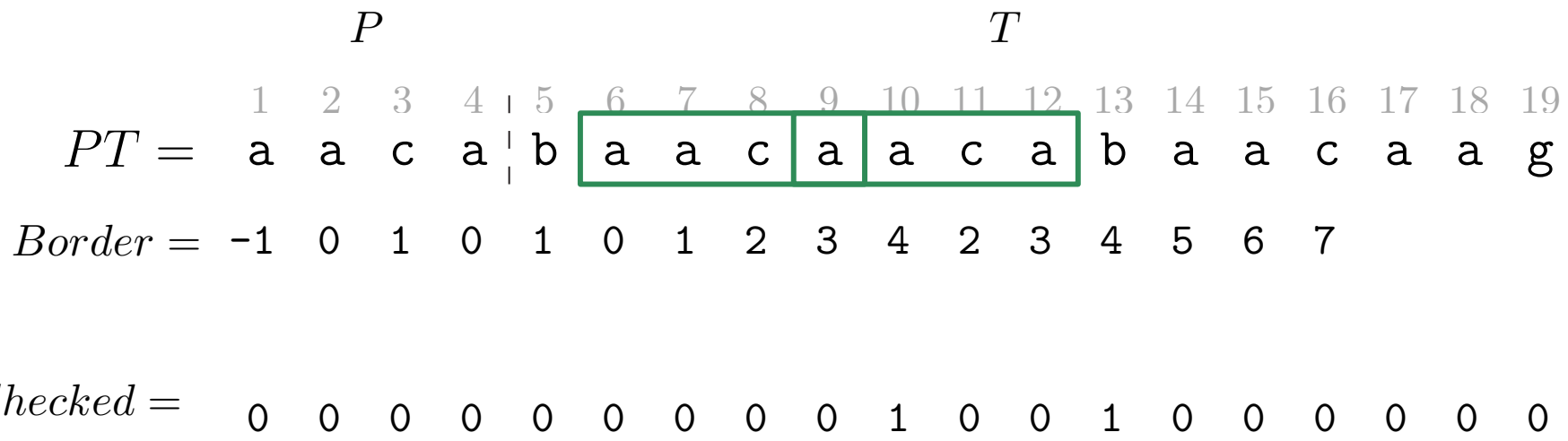
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

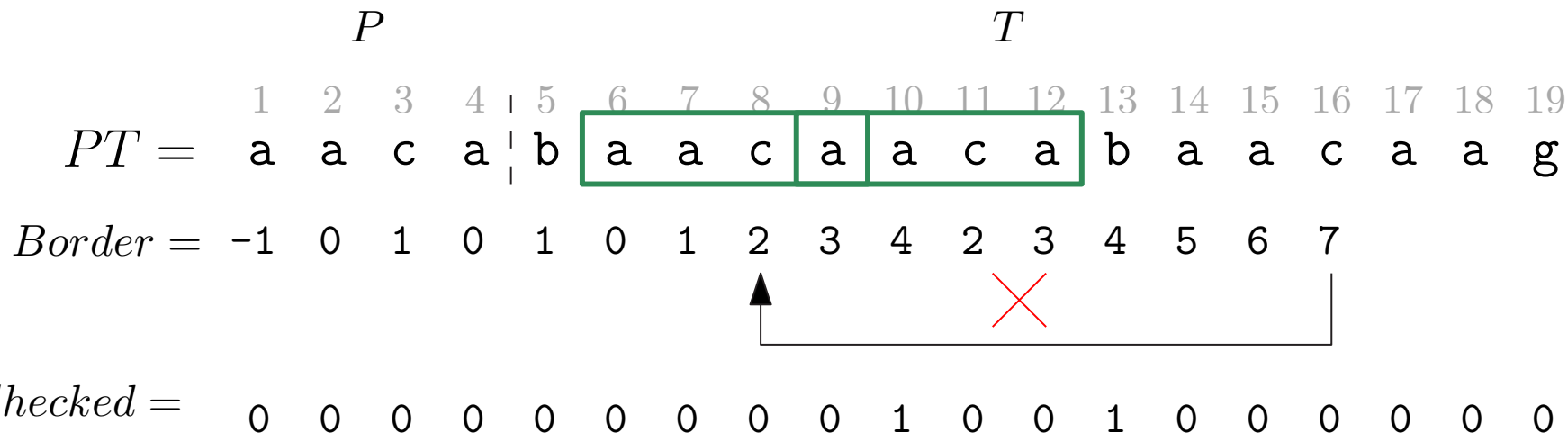
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

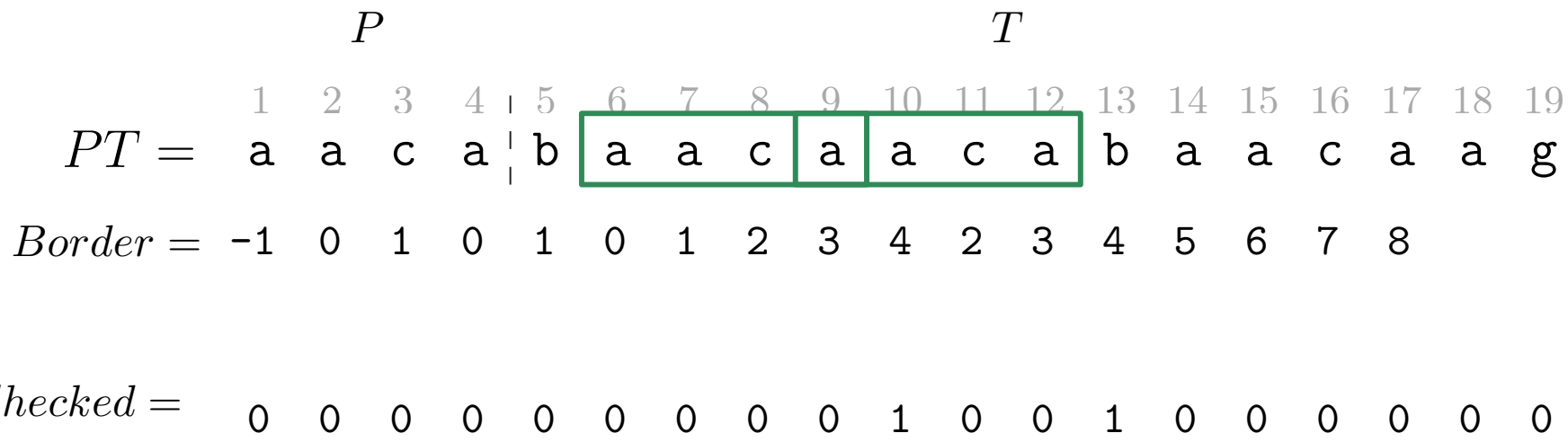
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

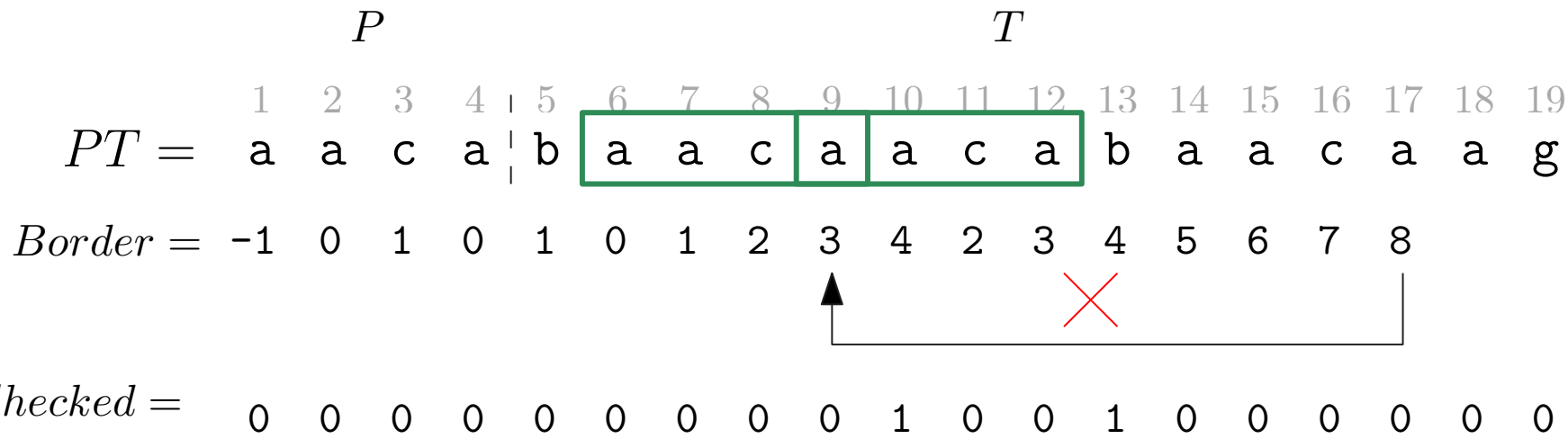
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

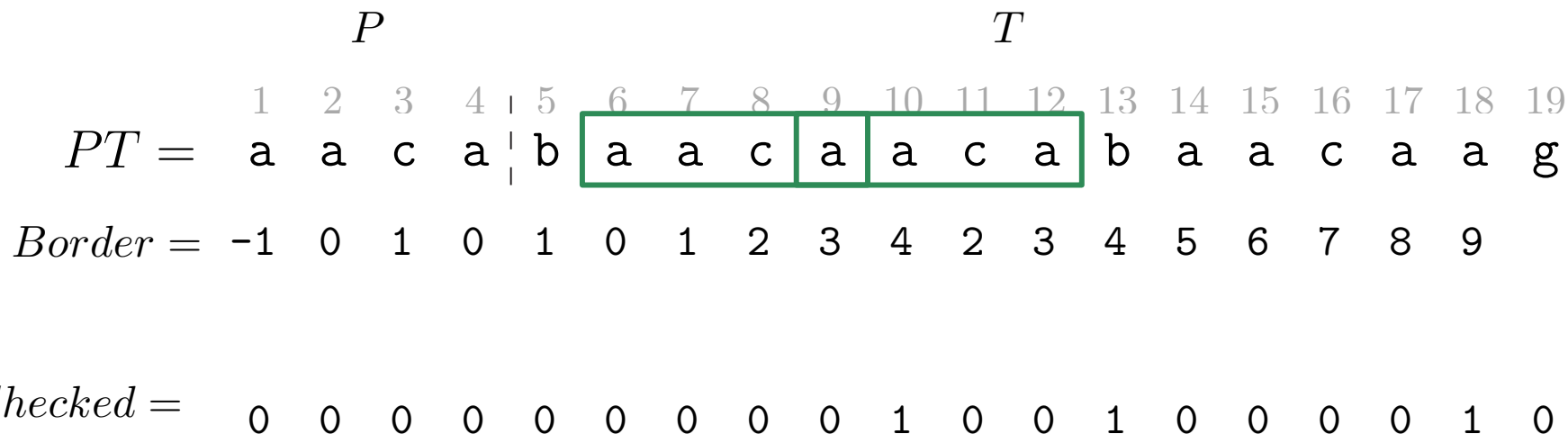
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

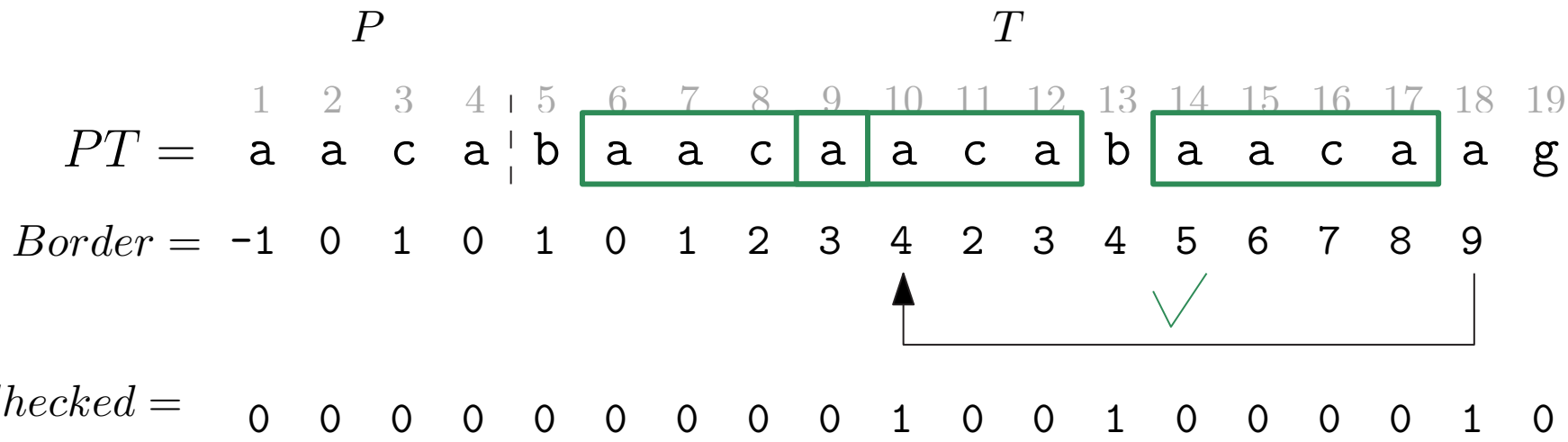
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

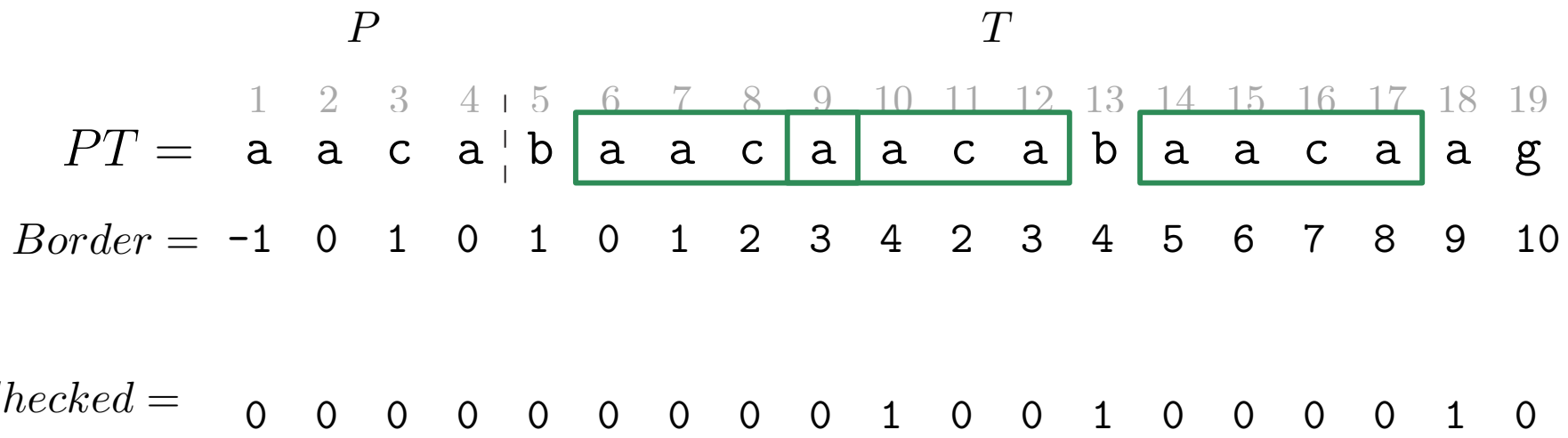
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

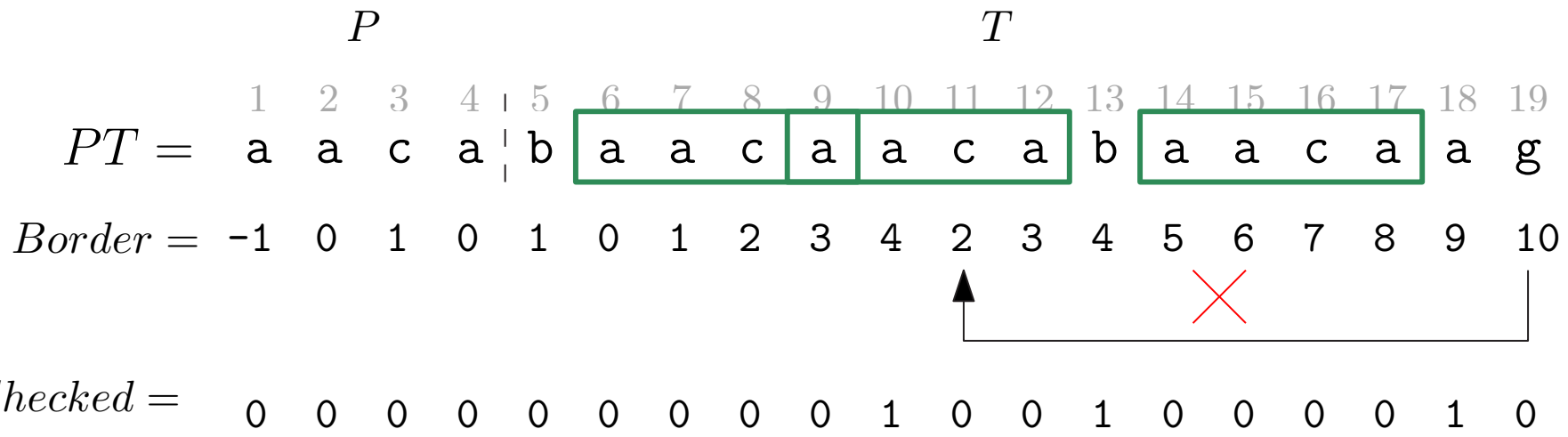
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

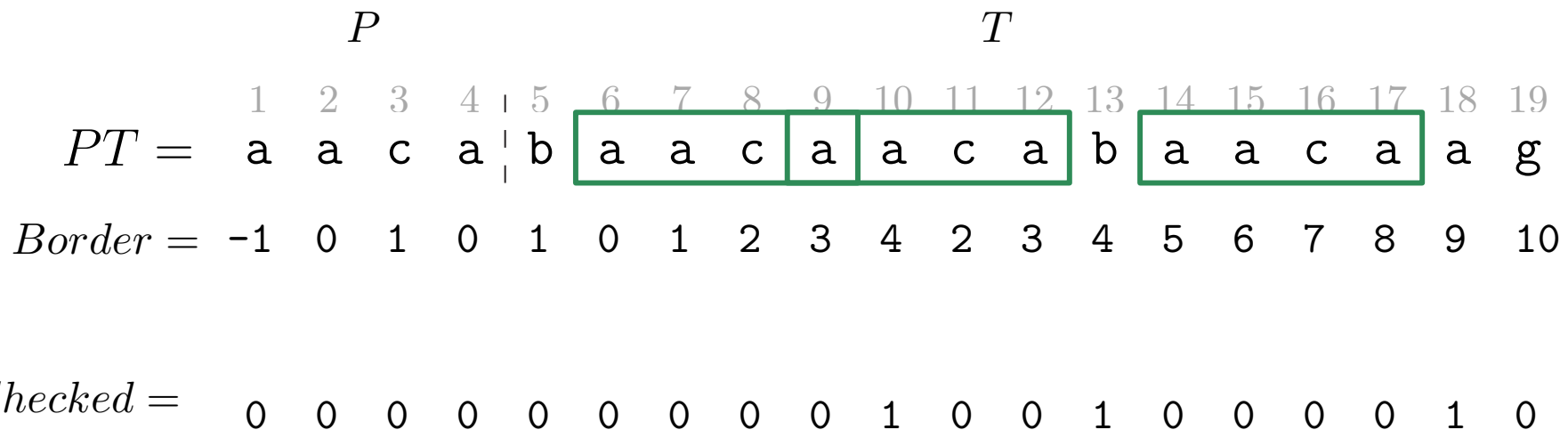
- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Border-Array

Concatenation with the special symbol

- $P = aaca$, $T = baacaacabaacaag$, consider PT.
- **for** $i \in [1, |PT|]$ **do**
  Check(i)
  Function Check(i):
  if $Border[i] == |P|$ **then** $Checked[i] = True$; return *True*
  if $Border[i] > |P|$ **then** return Check(Border[i] + 1)
  return *False*



Suche mit Suffix-Arrays

Ablauf

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

Suche: $P =$ `bar`
$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start
- finde Ende
- Ergebnis

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$
$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start
- finde Ende
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i															
1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
3	r	b	a	r	h	a	b	a	r	b	e	r	$		
4	b	a	r	h	a	b	a	r	b	e	r	$			
5	a	r	h	a	b	a	r	b	e	r	$				
6	r	h	a	b	a	r	b	e	r	$					
7	h	a	b	a	r	b	e	r	$						
8	a	b	a	r	b	e	r	$							
9	b	a	r	b	e	r	$								
10	a	r	b	e	r	$									
11	r	b	e	r	$										
12	b	e	r	$											
13	e	r	$												
14	r	$													
15	$														

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$
$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start
- finde Ende
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	SA[i]														
1	15	$													
2	8	a	b	a	r	b	e	r	$						
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$
4	10	a	r	b	e	r	$								
5	5	a	r	h	a	b	a	r	b	e	r	$			
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
7	9	b	a	r	b	e	r	$							
8	4	b	a	r	h	a	b	a	r	b	e	r	$		
9	12	b	e	r	$										
10	13	e	r	$											
11	7	h	a	b	a	r	b	e	r	$					
12	14	r	$												
13	3	r	b	a	r	h	a	b	a	r	b	e	r	$	
14	11	r	b	e	r	$									
15	6	r	h	a	b	a	r	b	e	r	$				

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start (binäre Suche)

  $l = 1, r = n$

  while $(l < r)$ **do**

    $q = \lfloor \frac{l+r}{2} \rfloor$

    if $(P > T_{SA[q]..SA[q]+m-1})$

      then $l = q + 1$

      else $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

    then break

- finde Ende
- Ergebnis

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$		b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

	i	SA[i]	
	1	15	$
	2	8	a b a r b e r $
	3	2	a r b a r h a b a r b e r $
	4	10	a r b e r $
	5	5	a r h a b a r b e r $
	6	1	b a r b a r h a b a r b e r $
	7	9	b a r b e r $
$q =$	8	4	b a r h a b a r b e r $
	9	12	b e r $
	10	13	e r $
	11	7	h a b a r b e r $
	12	14	r $
	13	3	r b a r h a b a r b e r $
	14	11	r b e r $
	15	6	r h a b a r b e r $

$l = 1, r = 15$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start (binäre Suche)

  $l = 1, r = n$

  while $(l < r)$ **do**

    $q = \lfloor \frac{l+r}{2} \rfloor$

    if $(P > T_{SA[q]..SA[q]+m-1})$

      then $l = q + 1$

      else $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

    then break

- finde Ende
- Ergebnis

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$		b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
	i	SA[i]														
	1	15	$													
	2	8	a	b	a	r	b	e	r	$						
	3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$
$q =$	4	10	a	r	b	e	r	$								
	5	5	a	r	h	a	b	a	r	b	e	r	$			
	6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
	7	9	b	a	r	b	e	r	$							
	8	4	b	a	r	h	a	b	a	r	b	e	r	$		
	9	12	b	e	r	$										
	10	13	e	r	$											
	11	7	h	a	b	a	r	b	e	r	$					
	12	14	r	$												
	13	3	r	b	a	r	h	a	b	a	r	b	e	r	$	
	14	11	r	b	e	r	$									
	15	6	r	h	a	b	a	r	b	e	r	$				

$l = 1, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start (binäre Suche)

  $l = 1, r = n$

  while $(l < r)$ **do**

    $q = \lfloor \frac{l+r}{2} \rfloor$

    if $(P > T_{SA[q]..SA[q]+m-1})$

      then $l = q + 1$

      else $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

    then break

- finde Ende
- Ergebnis

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$		b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
i	SA[i]															
1	15	$														
2	8	a	b	a	r	b	e	r	$							
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
4	10	a	r	b	e	r	$									
5	5	a	r	h	a	b	a	r	b	e	r	$				
$q =$ 6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
7	9	b	a	r	b	e	r	$								
8	4	b	a	r	h	a	b	a	r	b	e	r	$			
9	12	b	e	r	$											
10	13	e	r	$												
11	7	h	a	b	a	r	b	e	r	$						
12	14	r	$													
13	3	r	b	a	r	h	a	b	a	r	b	e	r	$		
14	11	r	b	e	r	$										
15	6	r	h	a	b	a	r	b	e	r	$					

$l = 5, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start (binäre Suche)

  $l = 1, r = n$

  while $(l < r)$ **do**

    $q = \lfloor \frac{l+r}{2} \rfloor$

    if $(P > T_{SA[q]..SA[q]+m-1})$

      then $l = q + 1$

      else $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

    then break

- finde Ende
- Ergebnis

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$		b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
i	SA[i]															
1	15	$														
2	8	a	b	a	r	b	e	r	$							
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
4	10	a	r	b	e	r	$									
$q =$ 5	5	a	r	h	a	b	a	r	b	e	r	$				
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
7	9	b	a	r	b	e	r	$								
8	4	b	a	r	h	a	b	a	r	b	e	r	$			
9	12	b	e	r	$											
10	13	e	r	$												
11	7	h	a	b	a	r	b	e	r	$						
12	14	r	$													
13	3	r	b	a	r	h	a	b	a	r	b	e	r	$		
14	11	r	b	e	r	$										
15	6	r	h	a	b	a	r	b	e	r	$					

$l = 5, r = 6$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$

$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start (binäre Suche)

  $l = 1, r = n$

  while $(l < r)$ **do**

    $q = \lfloor \frac{l+r}{2} \rfloor$

    if $(P > T_{SA[q]..SA[q]+m-1})$

      then $l = q + 1$

      else $r = q$

  $s = l$

  if $(P \neq T_{SA[s]..SA[s]+m-1})$

    then break

- finde Ende
- Ergebnis

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$		b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
i	SA[i]															
1	15	$														
2	8	a	b	a	r	b	e	r	$							
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
4	10	a	r	b	e	r	$									
5	5	a	r	h	a	b	a	r	b	e	r	$				
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
7	9	b	a	r	b	e	r	$								
8	4	b	a	r	h	a	b	a	r	b	e	r	$			
9	12	b	e	r	$											
10	13	e	r	$												
11	7	h	a	b	a	r	b	e	r	$						
12	14	r	$													
13	3	r	b	a	r	h	a	b	a	r	b	e	r	$		
14	11	r	b	e	r	$										
15	6	r	h	a	b	a	r	b	e	r	$					

$l = 6, r = 6$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$
$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start
- finde Ende (binäre Suche)

  $l = s, r = n$
  while $(l < r)$ **do**
    $q = \lceil \frac{l+r}{2} \rceil$
    if $(P \geq T_{SA[q]..SA[q]+m-1})$
      then $l = q$
      else $r = q - 1$
  $t = l$
- Ergebnis

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$		b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
i	SA[i]															
1	15	$														
2	8	a	b	a	r	b	e	r	$							
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
4	10	a	r	b	e	r	$									
5	5	a	r	h	a	b	a	r	b	e	r	$				
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
7	9	b	a	r	b	e	r	$								
8	4	b	a	r	h	a	b	a	r	b	e	r	$			
9	12	b	e	r	$											
10	13	e	r	$												
$q =$ 11	7	h	a	b	a	r	b	e	r	$						
12	14	r	$													
13	3	r	b	a	r	h	a	b	a	r	b	e	r	$		
14	11	r	b	e	r	$										
15	6	r	h	a	b	a	r	b	e	r	$					

$l = 6, r = 15$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$
$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start
- finde Ende (binäre Suche)

  $l = s, r = n$
  while $(l < r)$ **do**
    $q = \lceil \frac{l+r}{2} \rceil$
    if $(P \geq T_{SA[q]..SA[q]+m-1})$
      then $l = q$
      else $r = q - 1$
  $t = l$
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

	i	SA[i]														
	1	15	$													
	2	8	a	b	a	r	b	e	r	$						
	3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$
	4	10	a	r	b	e	r	$								
	5	5	a	r	h	a	b	a	r	b	e	r	$			
	6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
	7	9	b	a	r	b	e	r	$							
$q =$	8	4	b	a	r	h	a	b	a	r	b	e	r	$		
	9	12	b	e	r	$										
	10	13	e	r	$											
	11	7	h	a	b	a	r	b	e	r	$					
	12	14	r	$												
	13	3	r	b	a	r	h	a	b	a	r	b	e	r	$	
	14	11	r	b	e	r	$									
	15	6	r	h	a	b	a	r	b	e	r	$				

$l = 6, r = 10$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$
$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start
- finde Ende (binäre Suche)

  $l = s, r = n$
  while $(l < r)$ **do**
    $q = \lceil \frac{l+r}{2} \rceil$
    if $(P \geq T_{SA[q]..SA[q]+m-1})$
      then $l = q$
      else $r = q - 1$
  $t = l$
- Ergebnis

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$		b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
i	SA[i]															
1	15	$														
2	8	a	b	a	r	b	e	r	$							
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
4	10	a	r	b	e	r	$									
5	5	a	r	h	a	b	a	r	b	e	r	$				
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
7	9	b	a	r	b	e	r	$								
8	4	b	a	r	h	a	b	a	r	b	e	r	$			
$q =$ 9	12	b	e	r	$											
10	13	e	r	$												
11	7	h	a	b	a	r	b	e	r	$						
12	14	r	$													
13	3	r	b	a	r	h	a	b	a	r	b	e	r	$		
14	11	r	b	e	r	$										
15	6	r	h	a	b	a	r	b	e	r	$					

$l = 8, r = 10$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$
$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start
- finde Ende (binäre Suche)

  $l = s, r = n$
  while $(l < r)$ **do**
    $q = \lceil \frac{l+r}{2} \rceil$
    if $(P \geq T_{SA[q]..SA[q]+m-1})$
      then $l = q$
      else $r = q - 1$
  $t = l$
- Ergebnis

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$

i	SA[i]														
1	15	$													
2	8	a	b	a	r	b	e	r	$						
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$
4	10	a	r	b	e	r	$								
5	5	a	r	h	a	b	a	r	b	e	r	$			
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
7	9	b	a	r	b	e	r	$							
8	4	b	a	r	h	a	b	a	r	b	e	r	$		
9	12	b	e	r	$										
10	13	e	r	$											
11	7	h	a	b	a	r	b	e	r	$					
12	14	r	$												
13	3	r	b	a	r	h	a	b	a	r	b	e	r	$	
14	11	r	b	e	r	$									
15	6	r	h	a	b	a	r	b	e	r	$				

$l = 8, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{bar}$
$|P| = m, |T| = n$

- (SA bestimmen)
- finde Start
- finde Ende
- Ergebnis
  - $t - s + 1$
    (counting query)
  - $\{SA[s], \ldots, SA[t]\}$
    (reporting query)

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T =$		b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
i	SA[i]															
1	15	$														
2	8	a	b	a	r	b	e	r	$							
3	2	a	r	b	a	r	h	a	b	a	r	b	e	r	$	
4	10	a	r	b	e	r	$									
5	5	a	r	h	a	b	a	r	b	e	r	$				
6	1	b	a	r	b	a	r	h	a	b	a	r	b	e	r	$
7	9	b	a	r	b	e	r	$								
$q =$ 8	4	b	a	r	h	a	b	a	r	b	e	r	$			
9	12	b	e	r	$											
10	13	e	r	$												
11	7	h	a	b	a	r	b	e	r	$						
12	14	r	$													
13	3	r	b	a	r	h	a	b	a	r	b	e	r	$		
14	11	r	b	e	r	$										
15	6	r	h	a	b	a	r	b	e	r	$					

$s = 6, t = 8$

Suche mit Suffix-Arrays

Zusammenfassung

- Verlagerung des Aufwands von Anfrage in Vorverarbeitung
  - $\rightarrow$ einmal Suffix-Array generieren in $\mathcal{O}(n)$,
  - $\rightarrow$ danach Anfragen in $\mathcal{O}(m \log n)$ möglich, statt in $\mathcal{O}(m+n)$

  (gut, wenn auf einem Text viele Anfragen stattfinden)

- Ausnutzung der Eigenschaften des Suffix-Arrays
  - $\rightarrow$ jeder Substring ist Präfix eines Suffix
  - $\rightarrow$ alle Substrings liegen "sortiert" vor

  (das Suffix-Array indiziert alle Suffixe in sortierter Reihenfolge)

LCP-Array

Längster gemeinsamer Präfix

Definition:

- LCP$[i]$: Länge des längsten gemeinsamen Präfixes von je zwei lexikographisch benachbarten Suffixen $A[\mathrm{SA}[i-1]\ldots n]$ und $A[\mathrm{SA}[i]\ldots n]$

Erweiterung auf beliebige Suffixe

- LCP$[i][j]$: Länge des längsten gemeinsamen Präfix beliebiger lexikographischer Suffixe $A[\mathrm{SA}[i]\ldots n]$ und $A[\mathrm{SA}[j]\ldots n]$
- Konstruktion: $\mathcal{O}(n)$ Zeit und Platz
- Zugriff: $\mathcal{O}(1)$

Schnelle Suche mit Suffix-Arrays

Erster Ansatz

Suche: $P = \texttt{barberac}$

- Ziel: kein wiederholtes Vergleichen von Zeichen aus P
- Nutze LCP-Array um Suche zu beschleunigen
- Worst case $\mathcal{O}(m + \log n)$

Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `barberac`
$k = \mathrm{LCP}[L, P]$

- **if** $(k = \mathrm{LCP}[L, q])$
  Compare P to q starting from $k+1$th symbol and decide
- **if** $(\mathrm{LCP}[L, q] > k)$
- **if** $(\mathrm{LCP}[L, q] < k)$

$L =$ b a r b a r h a b a r b e r ...

b a r b e r a b a ...

b a r b e r a b c ...
b a r b e r a c c ...
b a r b e r c b c $

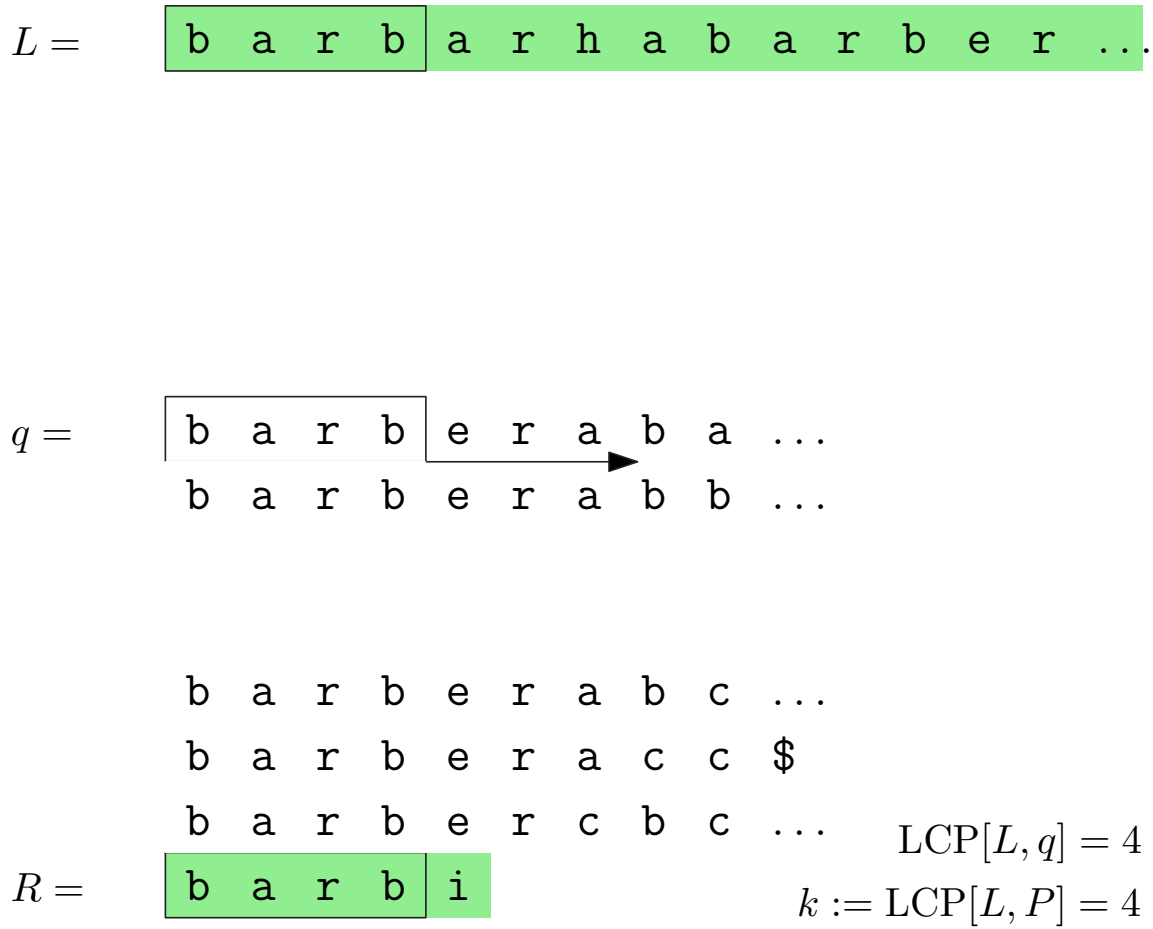
$R =$ b a r b i

Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `barberac`
$k = \text{LCP}[L, P]$

- **if** $(k = \text{LCP}[L, q])$
  Compare P to q starting from $k + 1$th symbol and decide
- **if** $(\text{LCP}[L, q] > k)$
- **if** $(\text{LCP}[L, q] < k)$



Suche mit Suffix-Arrays

Ablauf

b a r b a r h a b a r b e r ...

Suche: $P =$ barberac
$k = \text{LCP}[L, P]$

- **if** $(k = \text{LCP}[L, q])$
- **if** $(\text{LCP}[L, q] > k)$
- **if** $(\text{LCP}[L, q] < k)$

$L =$

b a r b e r a b a ...
b a r b e r a b b ...

$R =$

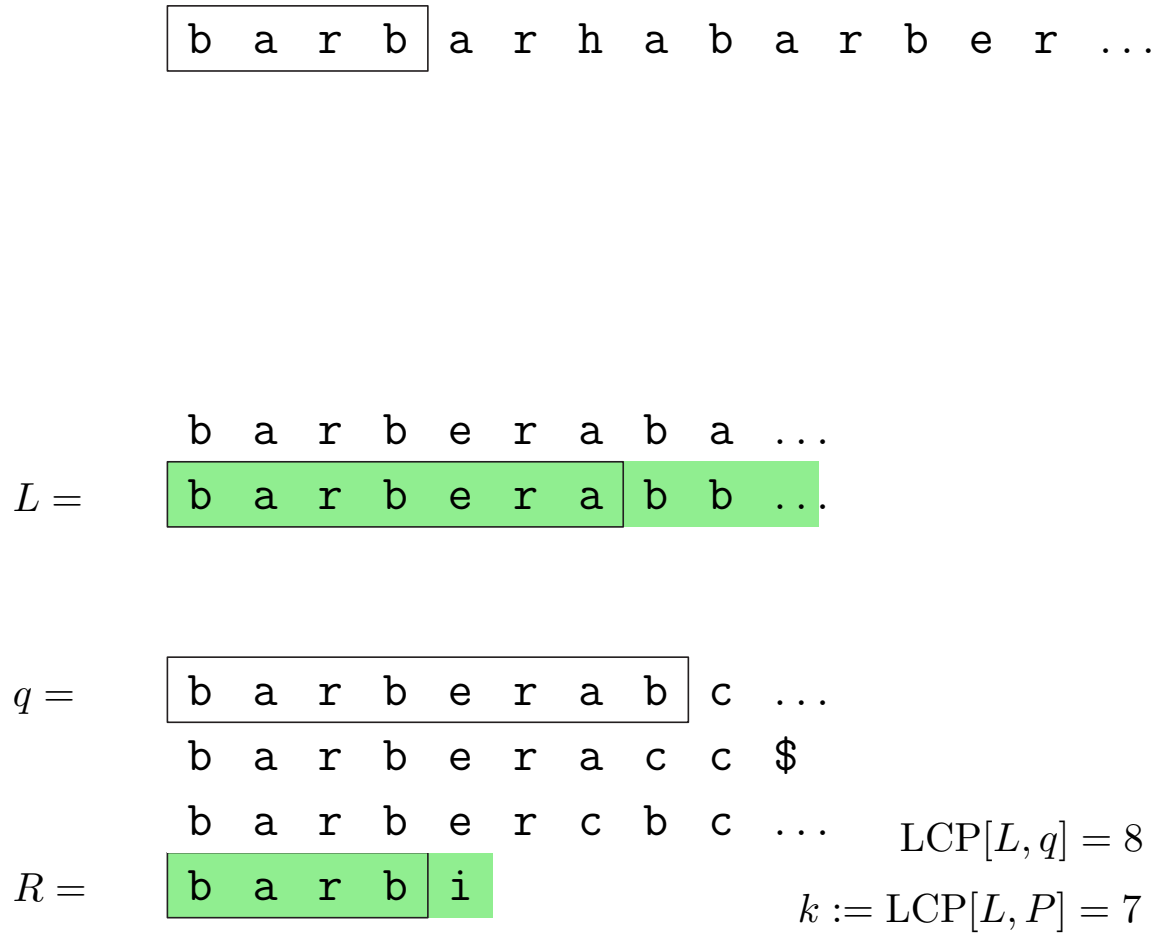
b a r b e r a b c ...
b a r b e r a c c $
b a r b e r c b c ...
b a r b i

Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `barberac`
$k = \text{LCP}[L, P]$

- **if** $(k = \text{LCP}[L, q])$
- **if** $(\text{LCP}[L, q] > k)$
  $L := q + 1$
- **if** $(\text{LCP}[L, q] < k)$

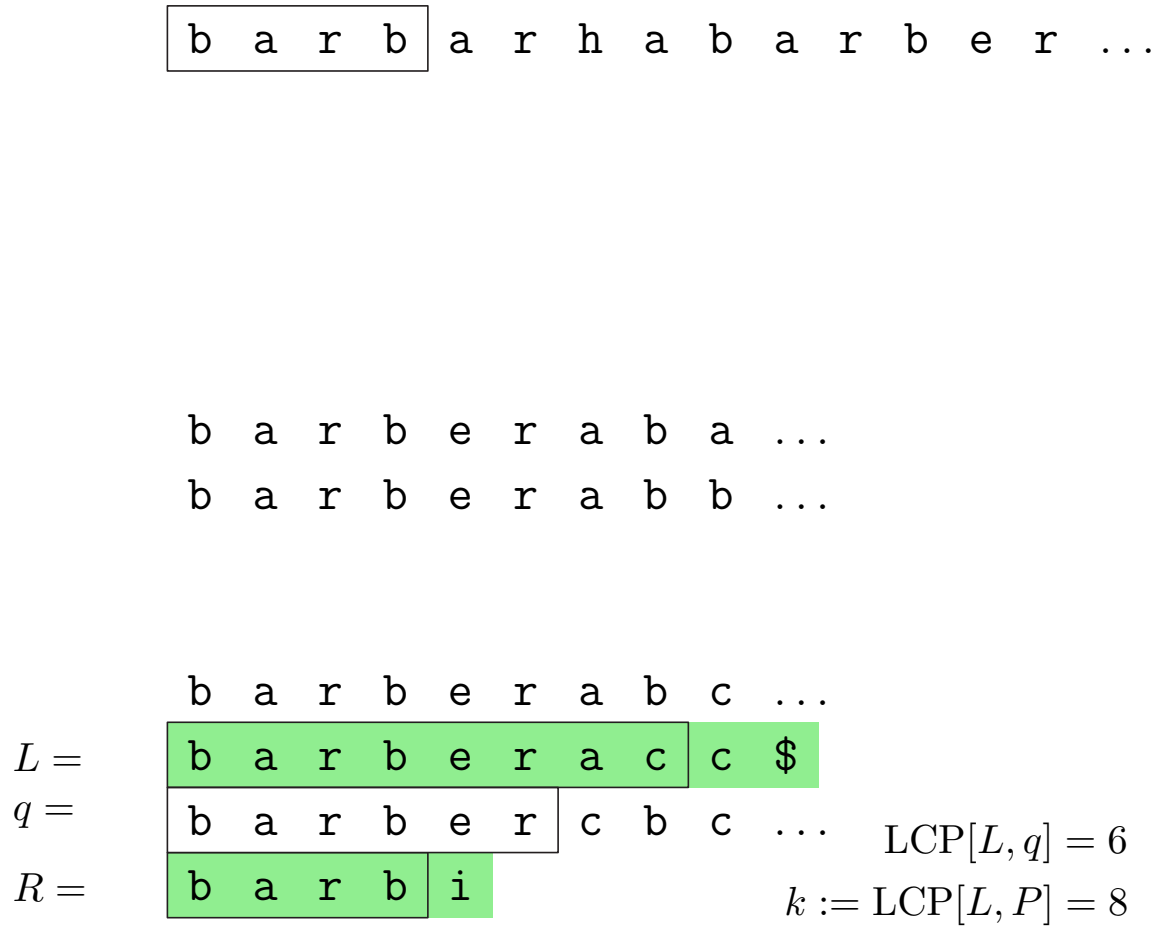


Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `barberac`
$k = \text{LCP}[L, P]$

- **if** $(k = \text{LCP}[L, q])$
- **if** $(\text{LCP}[L, q] > k)$
- **if** $(\text{LCP}[L, q] < k)$
  $R := q - 1$

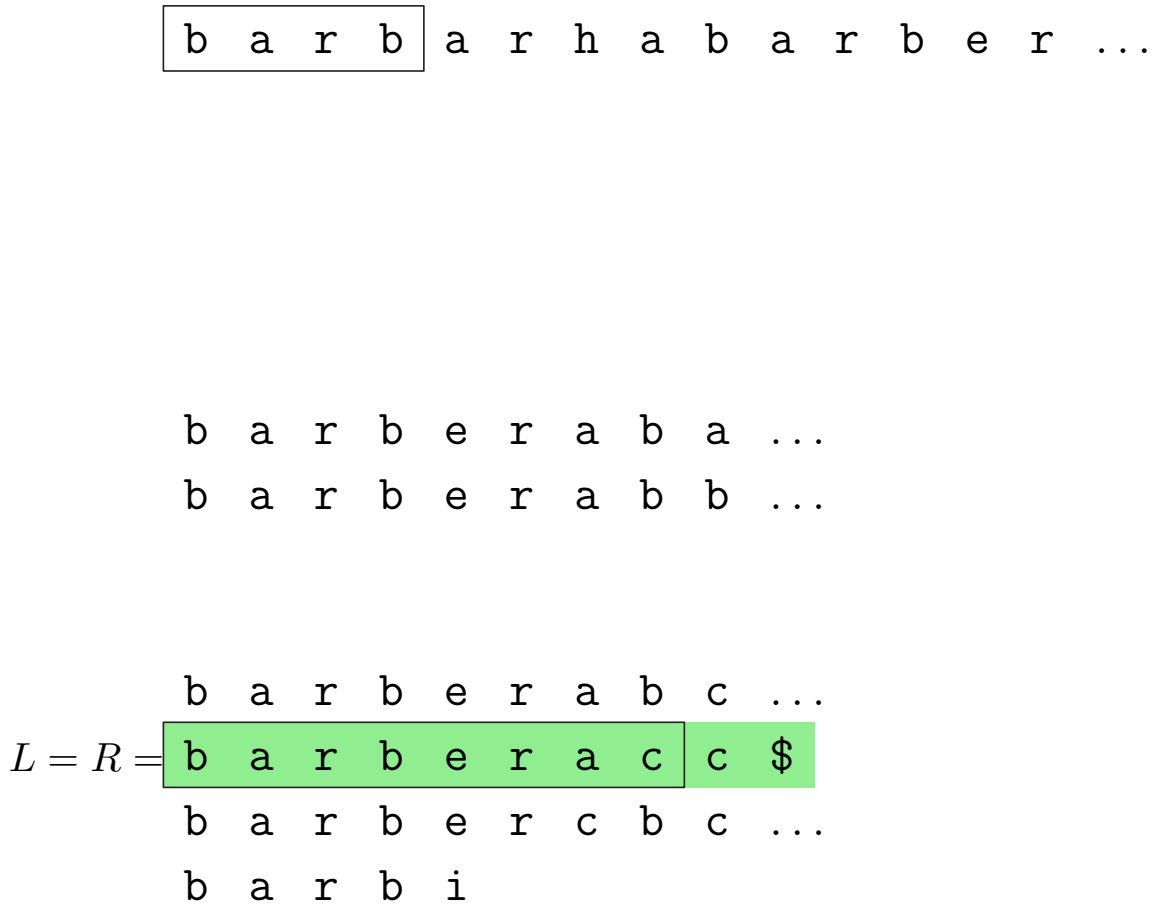


Suche mit Suffix-Arrays

Ablauf

Suche: $P =$ `barberac`
$k = \text{LCP}[L, P]$

- **if** $(k = \text{LCP}[L, q])$
- **if** $(\text{LCP}[L, q] > k)$
- **if** $(\text{LCP}[L, q] < k)$
  $R := q - 1$



Suche mit Suffix-Arrays

Ablauf

Suche: $P = \texttt{barberac}$
$k = \text{LCP}[L, P]$

- **if** $(k = \text{LCP}[L, q])$
- **if** $(\text{LCP}[L, q] > k)$
- **if** $(\text{LCP}[L, q] < k)$

Laufzeit

- LCP + SA: $\mathcal{O}(m + \log n)$ Vergleiche
- Beweisidee
  - k werden nur größer
  - Anzahl an redundanten Vergleichen pro Rekursion konstant

Suche mit Suffix-Arrays

Zusammenfassung

- Verlagerung des Aufwands von Anfrage in Vorverarbeitung
  - $\rightarrow$ einmal Suffix-Array generieren in $\mathcal{O}(n)$,
  - $\rightarrow$ danach Anfragen in $\mathcal{O}(m \log n)$ möglich, statt in $\mathcal{O}(m+n)$

  (gut, wenn auf einem Text viele Anfragen stattfinden)

- Verhindern redundanter Vergleiche
  - $\rightarrow$ einmal Suffix-Array generieren in $\mathcal{O}(n)$,
  - $\rightarrow$ einmal LCP-Array generieren in $\mathcal{O}(n)$,
  - $\rightarrow$ einmal erweitertes LCP-Array generieren in $\mathcal{O}(n)$,
  - $\rightarrow$ danach Anfrage in $\mathcal{O}(m + \log n)$

- Ausnutzung der Eigenschaften des Suffix-Arrays
  - $\rightarrow$ jeder Substring ist Präfix eines Suffix
  - $\rightarrow$ alle Substrings liegen “sortiert” vor

  (das Suffix-Array indiziert alle Suffixe in sortierter Reihenfolge)

Ende!

Feierabend!