

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

```
    result = current_weight;
    return true;
}

for( EdgeID eid = graph.edgeBegin( current ); eid != graph.edgeEnd( current ); ++eid ){
    const Edge & edge = graph.getEdge( eid );
    COUNTING( statistic_data.inc( DijkstraStatisticData::TOUCHED_EDGES ); )
    if( edge.forward ){
        COUNTING( statistic_data.inc( DijkstraStatisticData::RELAXED_EDGES ); )
        weight new_weight = edge.weight + current_weight;
        GUARANTEE( new_weight >= current_weight, std::runtime_error, "Weight overflow detected." );
        if( !priority_queue.isReached( edge.target ) ){
            COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_EDGES ); )
            COUNTING( statistic_data.inc( DijkstraStatisticData::REACHED_NODES ); )
            priority_queue.push( edge.target, new_weight );
        } else {
            if( priority_queue.getCurrentKey( edge.target ) > new_weight ){
                COUNTING( statistic_data.inc( DijkstraStatisticData::SUCCESSFULLY_RELAXED_NODES ); )
                priority_queue.decreaseKey( edge.target, new_weight );
            }
        }
    }
}
```

- *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

Bentley, Sedgewick (1997)

(*Three-way Radix Quicksort*)

- sortiert Elemente mit **mehreren Schlüsseln** wie *msd-Radixsort*
→ z.B. Stellen einer Zahl, Zeichen eines Strings
- für einen Schlüssel wird *Quicksort* mit **drei Fällen** ausgeführt
→ **kleiner als**, **gleich**, **größer als** das Pivotelement
- Partitionierungsschritt kann **in-place** erfolgen
→ 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)

$\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$

$\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$

$\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$

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)

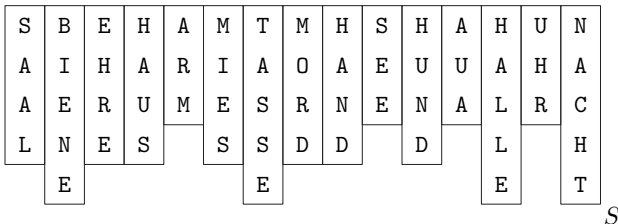
return concatenation of

(Rekursion)

$\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$

$\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$

$\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$



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)

$\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$

$\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$

$\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$

S	B	E	H	A	M	T	M	H	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
A	E	R	U	M	E	S	R	N	E	N	A	L	R	C
L	N	E	S		S	S	D	D		D		L		H
	E					E						E		T

$i = 1$

S

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($\langle e \in S : e[i] > p[i] \rangle, i$)

p

S	B	E	H	A	M	T	M	H	S	H	A	H	U	N
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
A	E	R	U	M	E	S	R	N	E	N	A	L	R	C
L	N	E	S		S	S	D	D		D		L		H
	E					E						E		T

$i = 1$

S

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)

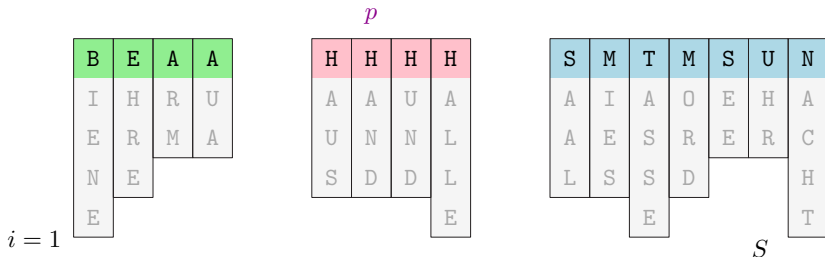
return concatenation of

(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)

$\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$

$\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$

$\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$

p

B	E	A	A
I	H	R	U
E	R	M	A
N	E		
E			

$i = 1$

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L
			E

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E		C
L	S	S	D			H
		E				T

S

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($\langle e \in S : e[i] > p[i] \rangle, i$)

p

B	E	A	A
I	H	R	U
E	U	S	W
N	E		
E			

usw.

$i = 1$

H	H	H	H
A	A	U	A
U	N	N	L
S	D	D	L
			E

S	M	T	M	S	U	N
A	I	A	O	E	H	A
A	E	S	R	E		C
L	S	S	D			H
		E				T

S

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

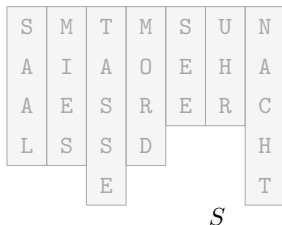
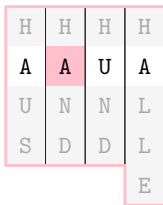
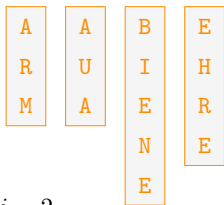
(Rekursion)

$\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$

$\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$

$\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$

p



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)

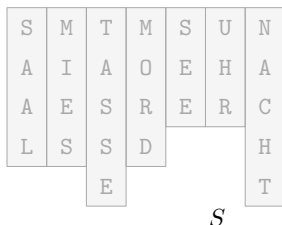
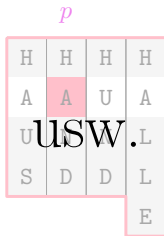
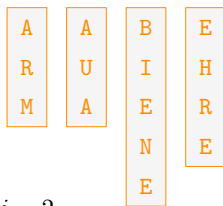
return concatenation of

(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)

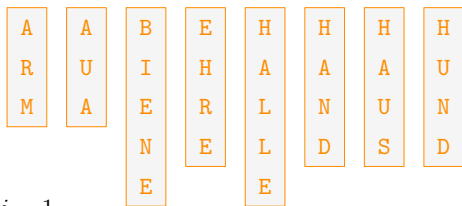
return concatenation of

(Rekursion)

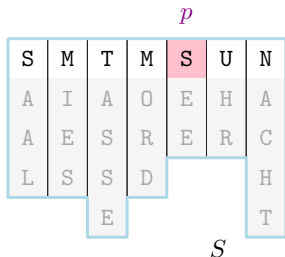
$\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$

$\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$

$\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$



$i = 1$



S

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)

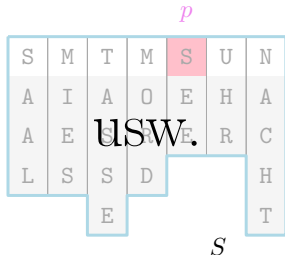
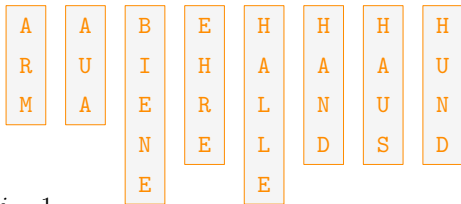
return concatenation of

(Rekursion)

$\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$

$\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$

$\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$



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)

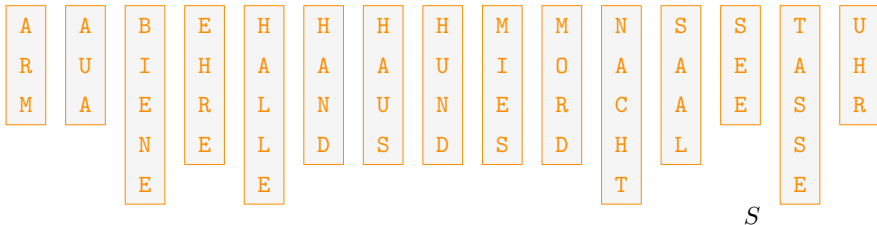
return concatenation of

(Rekursion)

$\text{mkqSort}(\langle e \in S : e[i] < p[i] \rangle, i),$

$\text{mkqSort}(\langle e \in S : e[i] = p[i] \rangle, i + 1),$

$\text{mkqSort}(\langle e \in S : e[i] > p[i] \rangle, i)$



In-place Multikey Quicksort

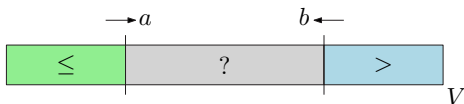
Wiederholung: Partitionierung

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”

→ 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 $a > b$

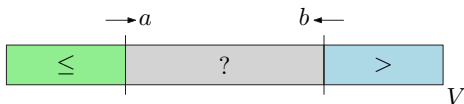


In-place Multikey Quicksort

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”
→ 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”
→ 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 $a > b$

10	17	7	41	17	43	25	33	78	14	29
1									n	V

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”
→ 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 $a > b$

p

10	17	7	41	17	43	25	33	78	14	29
1									n	

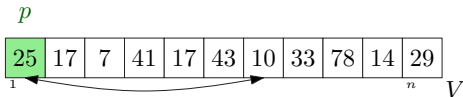
V

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”
→ 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 $a > b$

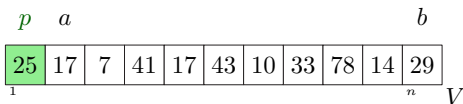


In-place Multikey Quicksort

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”
→ 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 $a > b$

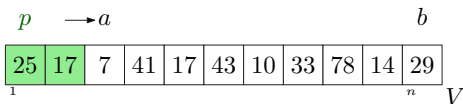


In-place Multikey Quicksort

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”
→ 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 $a > b$

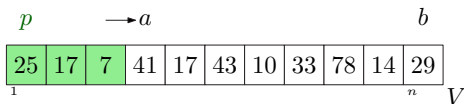


In-place Multikey Quicksort

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”
→ 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 $a > b$

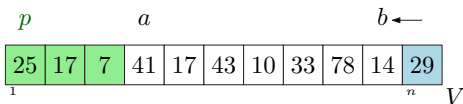


In-place Multikey Quicksort

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”
→ 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 $a > b$

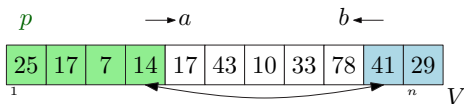


In-place Multikey Quicksort

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”
→ 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 $a > b$

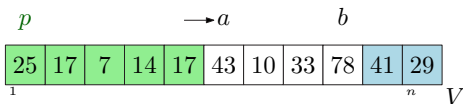


In-place Multikey Quicksort

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”
→ 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 $a > b$

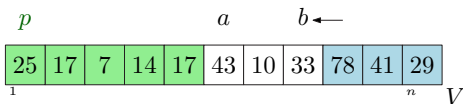


In-place Multikey Quicksort

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”
→ 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 $a > b$

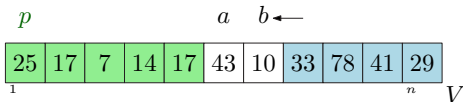


In-place Multikey Quicksort

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”
→ 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 $a > b$

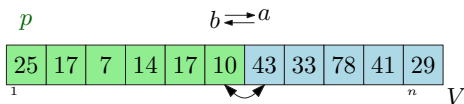


In-place Multikey Quicksort

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”
→ 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 $a > b$

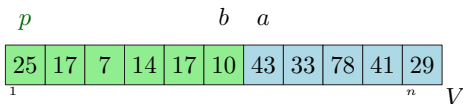


In-place Multikey Quicksort

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”
→ 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 $a > b$

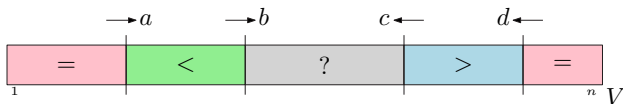


In-place Multikey Quicksort

Partitionierung

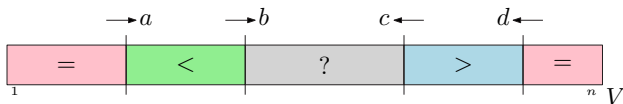
in-place bei Multikey Quicksort

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



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



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

1														n S	
S	B	E	H	A	M	T	M	H	S	H	A	H	U	N	
A	I	H	A	R	I	A	O	A	E	U	U	A	H	A	
A	E	R	U	M	E	S	R	N	E	N	A	L	R	C	
L	N	E	S		S	S	D	D		D		L		H	
	E					E						E		T	

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

$V \triangleq$

¹	S	B	E	H	A	M	T	M	H	S	H	A	H	U	N	ⁿ
	A	I	H	A	R	I	A	O	A	E	U	U	A	H	A	
	A	E	R	U	M	E	S	R	N	E	N	A	L	R	C	
	L	N	E	S		S	S	D	D		D		L		H	
		E				E							E		T	

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p

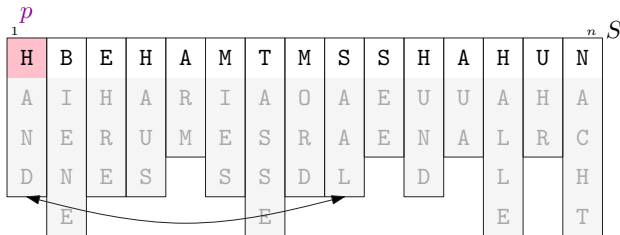
1	S	B	E	H	A	M	T	M	H	S	H	A	H	U	N	n
	A	I	H	A	R	I	A	O	A	E	U	U	A	H	A	
	A	E	R	U	M	E	S	R	N	E	N	A	L	R	C	
	L	N	E	S		S	S	D	D		D		L		H	
		E					E						E		T	

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p a b														c d		n S	
1	H	B	E	H	A	M	T	M	S	S	H	A	H	U	N		
	A	I	H	A	R	I	A	O	A	E	U	U	A	H	A		
	N	E	R	U	M	E	S	R	A	E	N	A	L	R	C		
	D	N	E	S		S	S	D	L		D		L		H		
		E					E						E		T		

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p		$a \rightarrow b$												$c \quad d$		$n \quad S$	
1	H	B	E	H	A	M	T	M	S	S	H	A	H	U	N		
	A	I	H	A	R	I	A	O	A	E	U	U	A	H	A		
	N	E	R	U	M	E	S	R	A	E	N	A	L	R	C		
	D	N	E	S		S	S	D	L		D		L		H		
		E					E						E		T		

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p a $\rightarrow$ b c d n S

1	H	B	E	H	A	M	T	M	S	S	H	A	H	U	N
	A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
	N	E	R	U	M	E	S	R	A	E	N	A	L	R	C
	D	N	E	S		S	S	D	L		D		L		H
		E					E						E		T

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p a $\rightarrow$ b c d n S

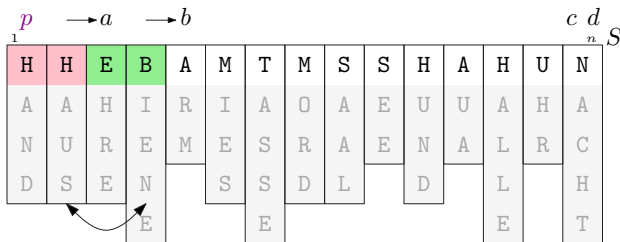
1	H	B	E	H	A	M	T	M	S	S	H	A	H	U	N
	A	I	H	A	R	I	A	O	A	E	U	U	A	H	A
	N	E	R	U	M	E	S	R	A	E	N	A	L	R	C
	D	N	E	S		S	S	D	L		D		L		H
		E					E						E		T

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p a $\rightarrow b$ c d n S															
1	H	H	E	B	A	M	T	M	S	S	H	A	H	U	N
	A	A	H	I	R	I	A	O	A	E	U	U	A	H	A
	N	U	R	E	M	E	S	R	A	E	N	A	L	R	C
	D	S	E	N		S	S	D	L		D		L		H
				E			E						E		T

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p	a				b				$c \leftarrow d$				n	S	
1	H	H	E	B	A	M	T	M	S	S	H	A	H	U	N
	A	A	H	I	R	I	A	O	A	E	U	U	A	H	A
	N	U	R	E	M	E	S	R	A	E	N	A	L	R	C
	D	S	E	N		S	S	D	L		D		L		H
				E			E						E		T

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

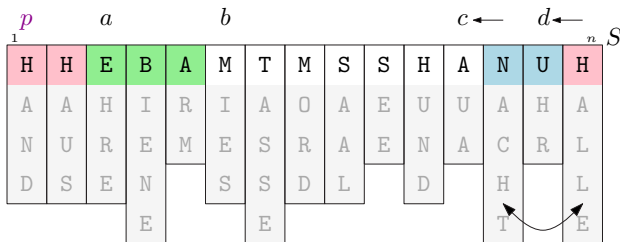
p		a		b		$c \leftarrow$		d		n		S	
1	2	3	4	5	6	7	8	9	10	11	12	13	14
H	H	E	B	A	M	T	M	S	S	H	A	H	U
A	A	H	I	R	I	A	O	A	E	U	U	A	H
N	U	R	E	M	E	S	R	A	E	N	A	L	R
D	S	E	N	E	S	S	D	L		D		L	
			E			E						E	
													T

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

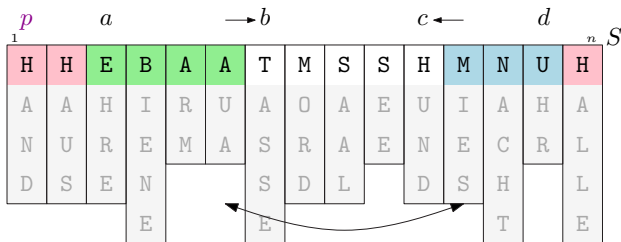


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

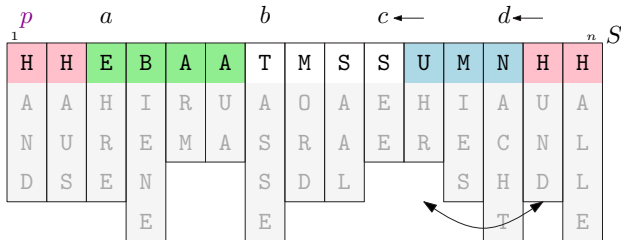


In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

1		a		b		$c \leftarrow$		d		n		S				
H	H	E	B	A	A	T	M	S	S	U	M	N	H	H		
A	A	H	I	R	U	A	O	A	E	H	I	A	U	A		
N	U	R	E	M	A	S	R	A	E	R	E	C	N	L		
D	S	E	N			S	D	L			S	H	D	L		
			E			E						T		E		

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p	a					b	$c \leftarrow$	d							n	S
1	H	H	E	B	A	A	T	M	S	S	U	M	N	H	H	
	A	A	H	I	R	U	A	O	A	E	H	I	A	U	A	
	N	U	R	E	M	A	S	R	A	E	R	E	C	N	L	
	D	S	E	N			S	D	L			S	H	D	L	
			E				E						T		E	

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p a $b \leftarrow$ d n S														
H	H	E	B	A	A	T	M	S	S	U	M	N	H	H
A	A	H	I	R	U	A	O	A	E	H	I	A	U	A
N	U	R	E	M	A	S	R	A	E	R	E	C	N	L
D	S	E	N			S	D	L			S	H	D	L
			E			E					T			E

In-place Multikey Quicksort

Partitionierung

in-place bei Multikey Quicksort (Algorithmus)

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

p	a			$c \leftarrow b$			d							n	S
1	H	H	E	B	A	A	T	M	S	S	U	M	N	H	H
	A	A	H	I	R	U	A	O	A	E	H	I	A	U	A
	N	U	R	E	M	A	S	R	A	E	R	E	C	N	L
	D	S	E	N			S	D	L			S	H	D	L
				E			E						T		E

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 $[n - r + 1, n + 1)$

		a		c		b							d			n	S
1	H	H	E	B	A	A	T	M	S	S	U	M	N	H	H		
	A	A	H	I	R	U	A	O	A	E	H	I	A	U	A		
	N	U	R	E	M	A	S	R	A	E	R	E	C	N	L		
	D	S	E	N			S	D	L			S	H	D	L		
				E			E						T		E		

In-place Multikey Quicksort

Partitionierung

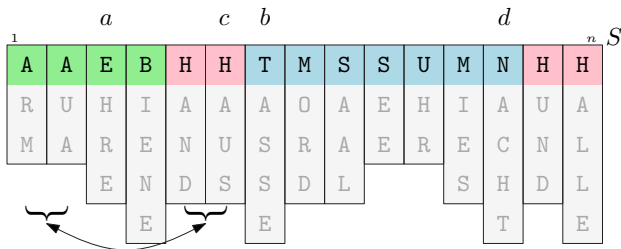
in-place bei Multikey Quicksort (Umgruppierung)

- $r = \min(a - 1, b - a)$

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 $[n - r + 1, n + 1)$

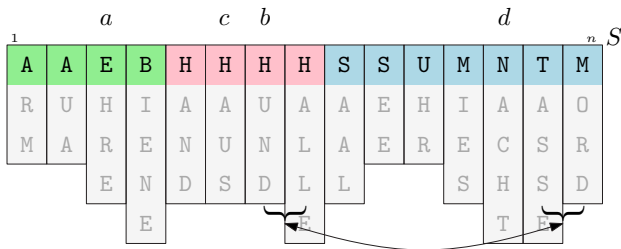
		a		c		b							d			n	S
1	A	A	E	B	H	H	T	M	S	S	U	M	N	H	H		
	R	U	H	I	A	A	A	O	A	E	H	I	A	U	A		
	M	A	R	E	N	U	S	R	A	E	R	E	C	N	L		
			E	N	D	S	S	D	L			S	H	D	L		
				E			E						T		E		

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 $[n - r + 1, n + 1)$

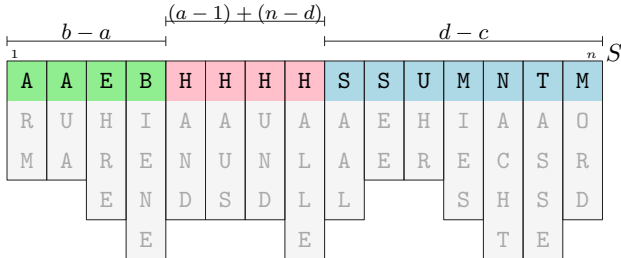


In-place Multikey Quicksort

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 $[n - r + 1, n + 1)$



- *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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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	4									

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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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	4	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	<i>P</i>											<i>T</i>									
	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	4	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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	4	2	3	4						

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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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	4	2	3	4	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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	4	2	3	4	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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	4	2	3	4	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	<i>P</i>						<i>T</i>													
	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
<i>Border</i> =	-1	0	1	0	1	0	0	1	2	3	4	2	3	4	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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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	4	2	3	4	0	1	2	3	4	

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 $P[1 \dots j-1]$ (of size **less** than $j-1$) that matches suffix of $P[1 \dots j-1]$.

	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	4	2	3	4	0	1	2	3	4	2

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

$Border = -1 \quad 0$

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

$Border = -1 \quad 0$

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 $False$

	<i>P</i>								<i>T</i>										
	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 $False$

	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	4									
$Checked =$	0	0	0	0	0	0	0	0	0	1	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	4	2								
$Checked =$	0	0	0	0	0	0	0	0	0	1	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	4	2	3							
$Checked =$	0	0	0	0	0	0	0	0	0	1	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	4	2	3	4						
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	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	4	2	3	4	5						
$checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	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	4	2	3	4	5					
						↑				×									
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	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	4	2	3	4	5	6				
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	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**

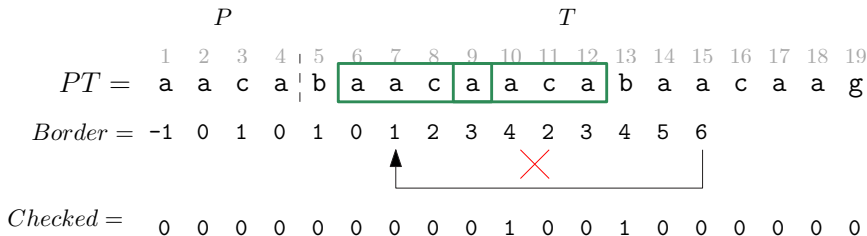
 Check(i)

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 $False$

	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	4	2	3	4	5	6	7			
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	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	4	2	3	4	5	6	7				
								↑												
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	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	4	2	3	4	5	6	7	8		
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	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	4	2	3	4	5	6	7	8		
									↑				×						
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	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	4	2	3	4	5	6	7	8	9	
$hecked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	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*

	<i>P</i>					<i>T</i>													
	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	4	2	3	4	5	6	7	8	9	
										↑				✓					
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	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	0	1	0	1	2	3	4	2	3	4	5	6	7	8	9	10
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	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*

	<i>P</i>					<i>T</i>													
	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	4	2	3	4	5	6	7	8	9	10
											↑				×				↓
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	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	0	1	0	1	2	3	4	2	3	4	5	6	7	8	9	10
$Checked =$	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0

Suche mit Suffix-Arrays

Ablauf

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

Suche: $P = \text{bar}$

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

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

Suche mit Suffix-Arrays

Ablauf

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

■ (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: $P = \text{bar}$
 $|P| = m, |T| = n$

■ (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: $P = \text{bar}$

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

■ (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: $P = \text{bar}$

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

- (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: $P = \text{bar}$

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

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

$l = 1, r = n$

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

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

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

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	\$					

$l = 5, r = 8$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

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

- (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	\$
$i \text{ SA}[i]$															
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: $P = \text{bar}$

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

- (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: $P = \text{bar}$

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

- (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	\$					

$l = 6, r = 15$

$l = 6, r = 15$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

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

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

$l = s, r = n$

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

$q = \lceil \frac{l+r}{2} \rceil$

if ($P \geq T_{SA[q]..SA[q]+m-1}$) $q = 8$

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	\$	
$i \text{ SA}[i]$																
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					\$	

$l = 6, r = 10$

$l = 6, r = 10$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

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

- (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	\$	
$i \text{ SA}[i]$																
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					\$	

$l = 8, r = 10$

$l = 8, r = 10$

Suche mit Suffix-Arrays

Ablauf

Suche: $P = \text{bar}$

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

- (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: $P = \text{bar}$

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

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

$s = 6, t = 8$

■ Verlagerung des Aufwands von Anfrage in Vorverarbeitung

- einmal Suffix-Array generieren in $\mathcal{O}(n)$,
- 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

- jeder Substring ist Präfix eines Suffix
- **alle Substrings liegen "sortiert" vor**

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

Definition:

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

Erweiterung auf beliebige Suffixe

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

Schnelle Suche mit Suffix-Arrays

Erster Ansatz

Suche: $P = \text{barberac}$

- 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

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

Suche: $P = \text{barberac}$

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

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

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

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

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

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: $P = \text{barberac}$

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

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

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

$q =$ b a r b e r a b a ...
 b a r b e r a b b ...

 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 ... $\text{LCP}[L, q] = 4$
 $R =$ b a r b i $k := \text{LCP}[L, P] = 4$

Suche mit Suffix-Arrays

Ablauf

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

Suche: $P = \text{barberac}$

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

■ **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: $P = \text{barberac}$

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

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

$$L = \begin{array}{cccccccc} & b & a & r & b & e & r & a & b & a & \dots \\ \hline L = & b & a & r & b & e & r & a & b & b & \dots \end{array}$$
$$\begin{array}{lcl}
 q = & \boxed{\text{b a r b e r a b}} \text{ c } \dots & \\
 & \text{b a r b e r a c c } \$ & \\
 & \text{b a r b e r c b c } \dots & \text{LCP}[L, q] = 8 \\
 R = & \boxed{\text{b a r b}} \text{ i } & k := \text{LCP}[L, P] = 7
 \end{array}$$

Suche mit Suffix-Arrays

Ablauf

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

Suche: $P = \text{barberac}$

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

■ **if** ($k = \text{LCP}[L, q]$)

■ **if** ($\text{LCP}[L, q] > k$)

■ **if** ($\text{LCP}[L, q] < k$)

$R := q - 1$

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

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

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

$L =$

b a r b e r a c c \$

$q =$

b a r b e r

c b c ... $\text{LCP}[L, q] = 6$

$R =$

b a r b i

$k := \text{LCP}[L, P] = 8$

Suche mit Suffix-Arrays

Ablauf

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

Suche: $P = \text{barberac}$

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

■ **if** ($k = \text{LCP}[L, q]$)

■ **if** ($\text{LCP}[L, q] > k$)

■ **if** ($\text{LCP}[L, q] < k$)

$R := q - 1$

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

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

$L = R =$

b	a	r	b	e	r	a	c	c	\$
---	---	---	---	---	---	---	---	---	----

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

Suche: $P = \text{barberac}$

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

- **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

- Verlagerung des Aufwands von Anfrage in Vorverarbeitung
 - einmal Suffix-Array generieren in $\mathcal{O}(n)$,
 - 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
 - einmal Suffix-Array generieren in $\mathcal{O}(n)$,
 - einmal LCP-Array generieren in $\mathcal{O}(n)$,
 - einmal erweitertes LCP-Array generieren in $\mathcal{O}(n)$,
 - danach **Anfrage in $\mathcal{O}(m + \log n)$**
- Ausnutzung der Eigenschaften des Suffix-Arrays
 - jeder Substring ist Präfix eines Suffix
 - **alle Substrings liegen "sortiert" vor**(das Suffix-Array indiziert alle Suffixe in sortierter Reihenfolge)

Ende!



Feierabend!