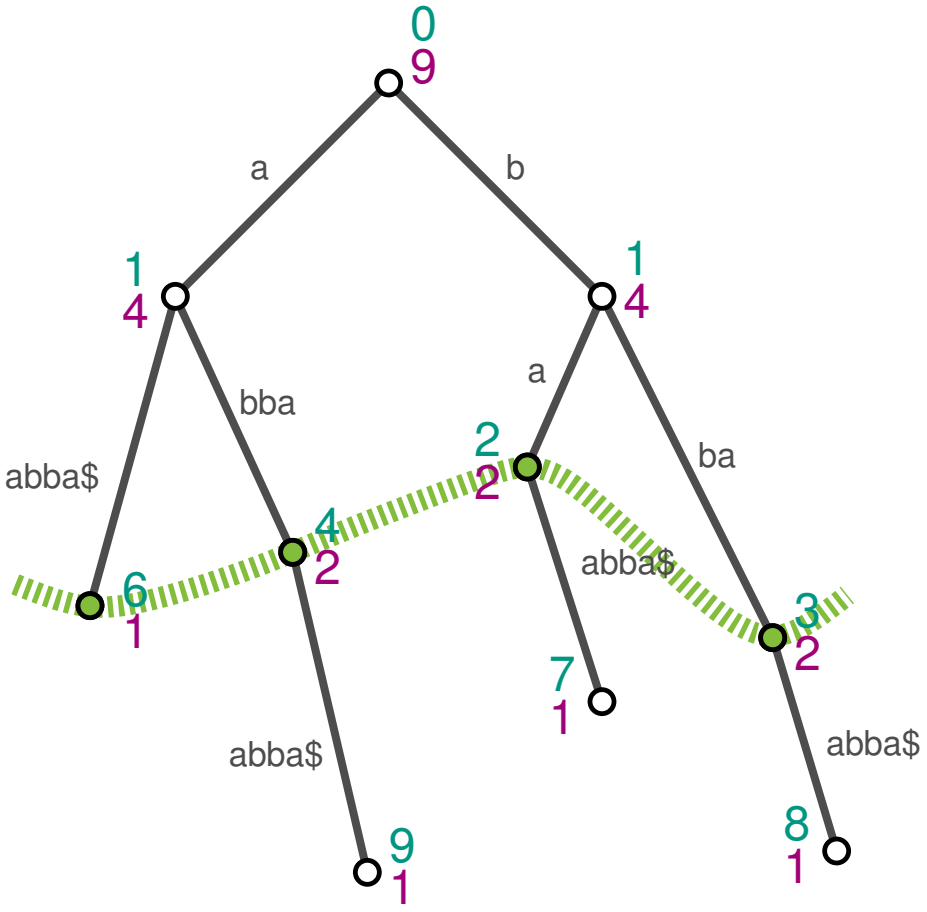
Textindexierung Programmierprojekt

Abschlusspräsentation · 07.02.2022
Moritz Potthoff

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8



Vorberechnung: Suffix Tree

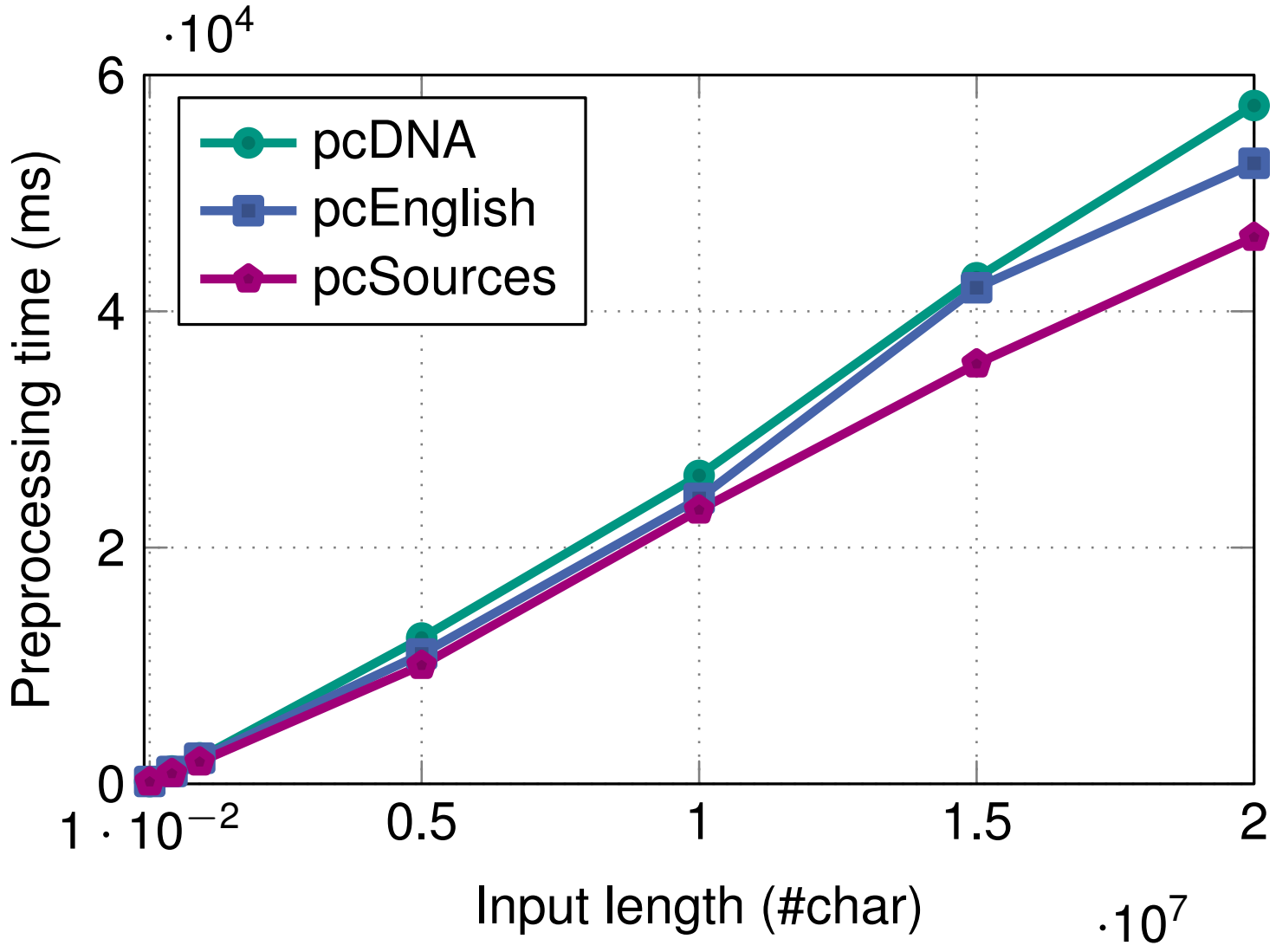
Suffix Tree-Konstruktion:

- In $\mathcal{O}(n)$ mittels **Ukkonens Algorithmus**
- Hoher Platzbedarf
- Lineare Laufzeit

Vorberechnung: Suffix Tree

Suffix Tree-Konstruktion:

- In $\mathcal{O}(n)$ mittels **Ukkonens Algorithmus**
- Hoher Platzbedarf
- Lineare Laufzeit



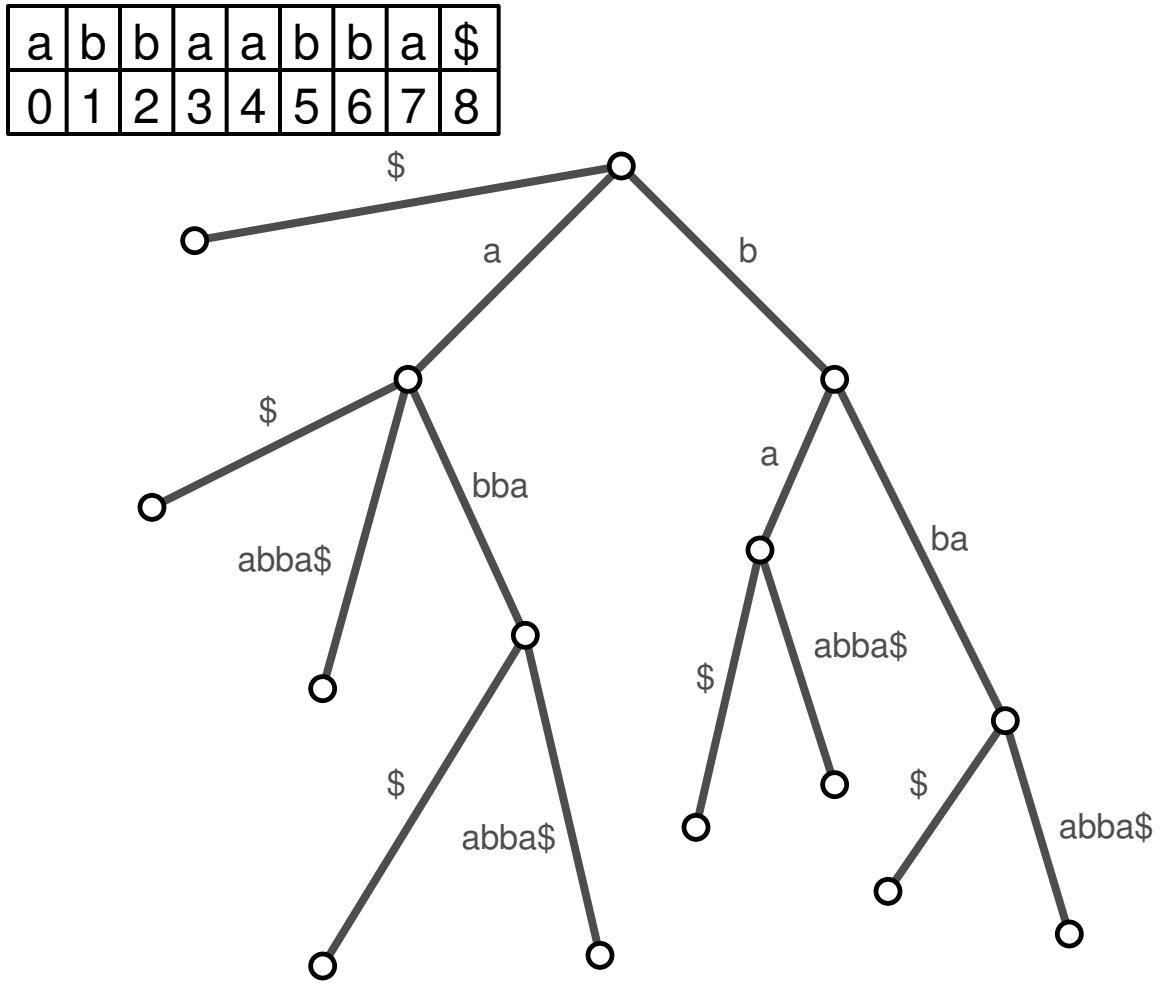
TopK-Queries – Algorithmus

- Annotiere jeden Knoten mit String-Tiefe und Anzahl Blätter unter ihm

  Entferne dabei Sentinel-Blätter

- Für Query mit Eingaben ℓ und k:
  - DFS: Sammle jeden höchsten Knoten mit String-Tiefe $\geq \ell$

    $\rightarrow$ **Kandidaten**, speichere Suffixposition und Anzahl Blätter
  - Sortiere Kandidaten stabil nach Anzahl Blättern
  - Gebe k-ten Kandidaten aus.



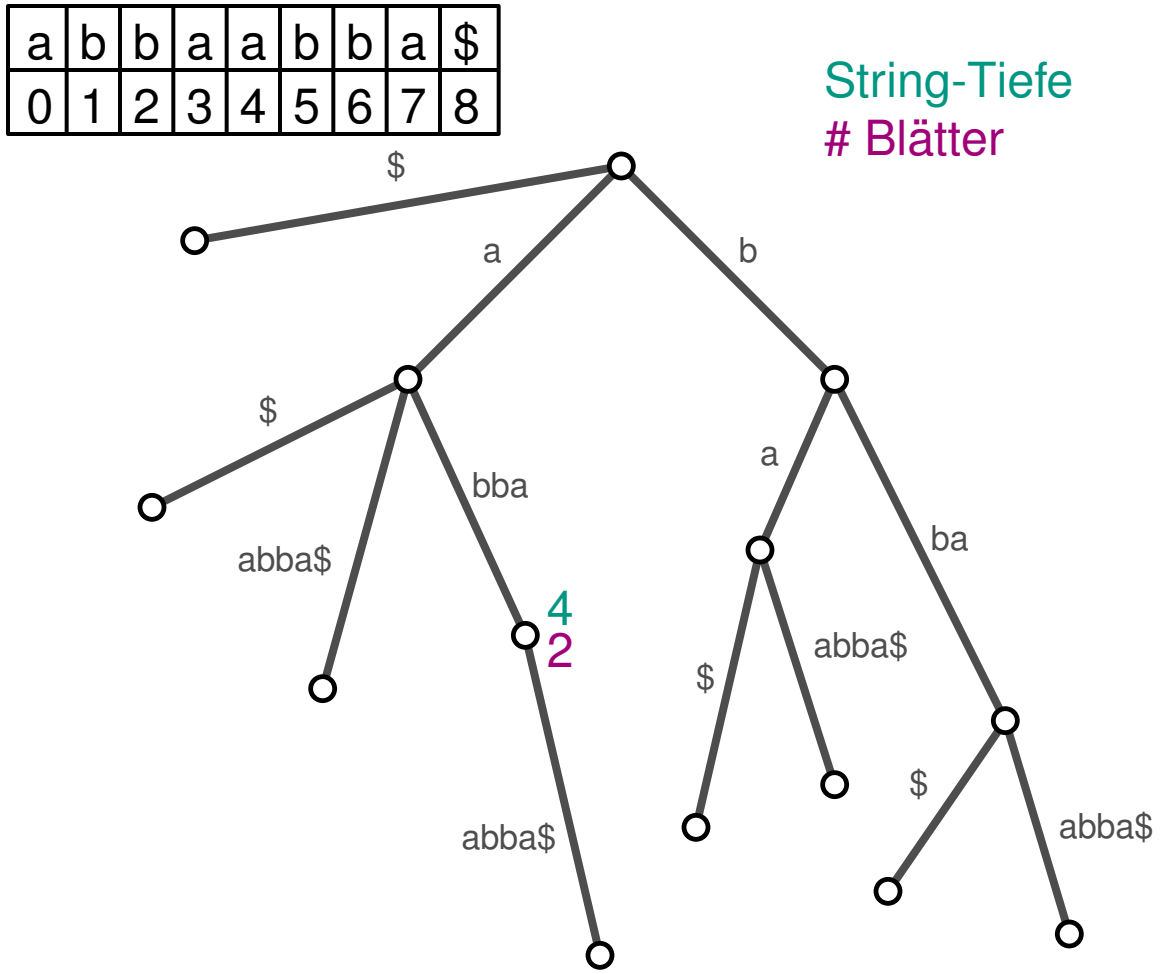
TopK-Queries – Algorithmus

- Annotiere jeden Knoten mit String-Tiefe und Anzahl Blätter unter ihm

  Entferne dabei Sentinel-Blätter

- Für Query mit Eingaben ℓ und k:
  - DFS: Sammle jeden höchsten Knoten mit String-Tiefe $\geq \ell$

    $\rightarrow$ **Kandidaten**, speichere Suffixposition und Anzahl Blätter
  - Sortiere Kandidaten stabil nach Anzahl Blättern
  - Gebe k-ten Kandidaten aus.



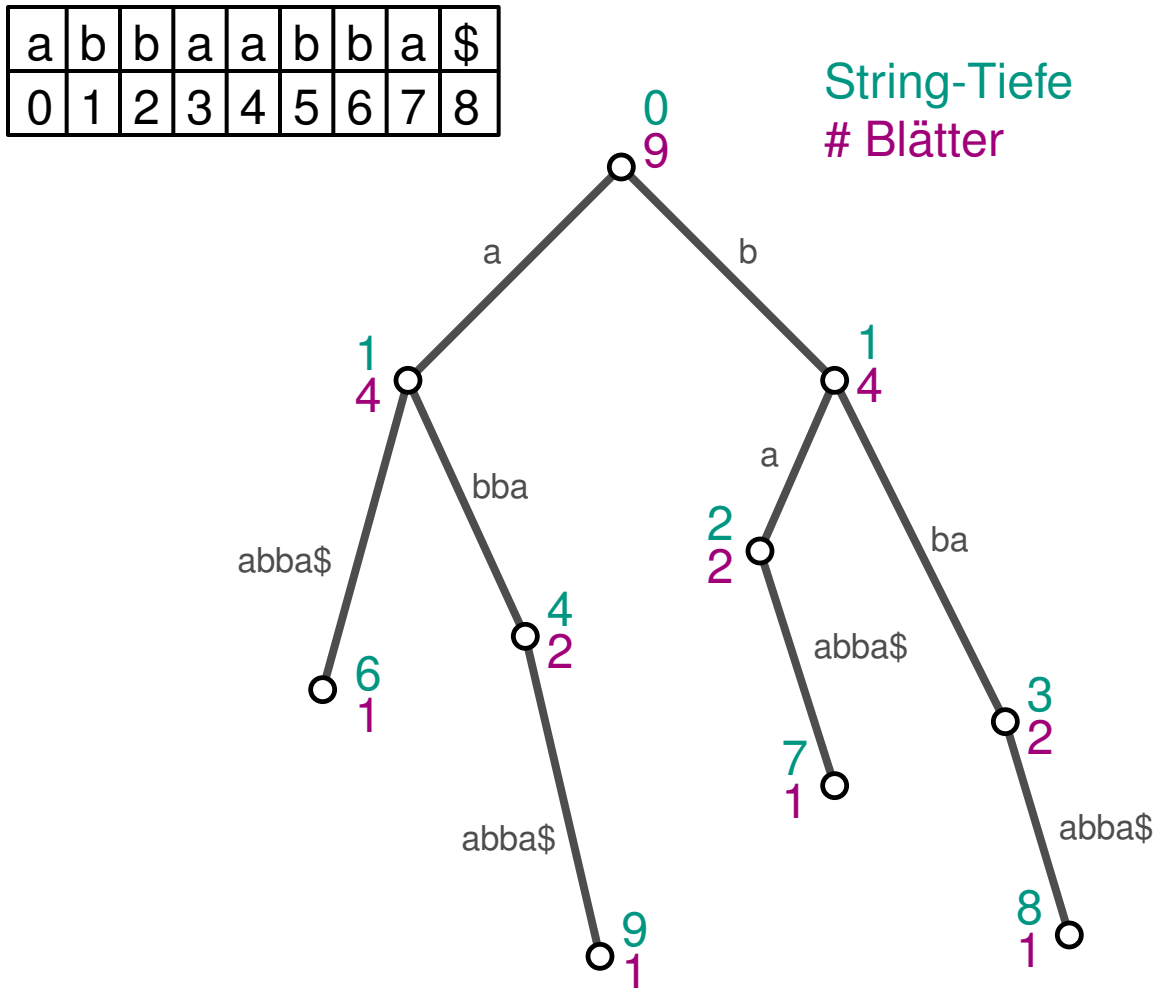
TopK-Queries – Algorithmus

- Annotiere jeden Knoten mit String-Tiefe und Anzahl Blätter unter ihm

  Entferne dabei Sentinel-Blätter

- Für Query mit Eingaben ℓ und k:
  - DFS: Sammle jeden höchsten Knoten mit String-Tiefe $\geq \ell$

    $\rightarrow$ **Kandidaten**, speichere Suffixposition und Anzahl Blätter
  - Sortiere Kandidaten stabil nach Anzahl Blättern
  - Gebe k-ten Kandidaten aus.



TopK-Queries – Algorithmus

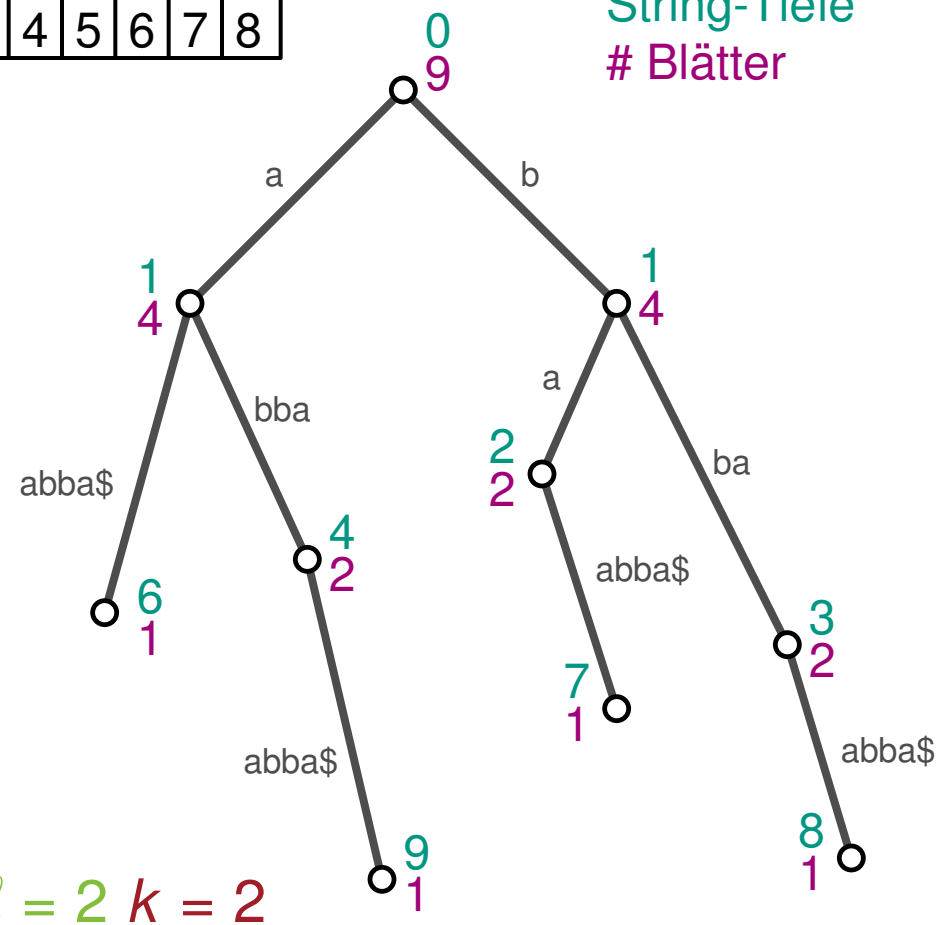
- Annotiere jeden Knoten mit String-Tiefe und Anzahl Blätter unter ihm

  Entferne dabei Sentinel-Blätter

- Für Query mit Eingaben ℓ und k:
  - DFS: Sammle jeden höchsten Knoten mit String-Tiefe $\geq \ell$

    $\rightarrow$ **Kandidaten**, speichere Suffixposition und Anzahl Blätter
  - Sortiere Kandidaten stabil nach Anzahl Blättern
  - Gebe k-ten Kandidaten aus.

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8



Query: $\ell = 2$ $k = 2$

TopK-Queries – Algorithmus

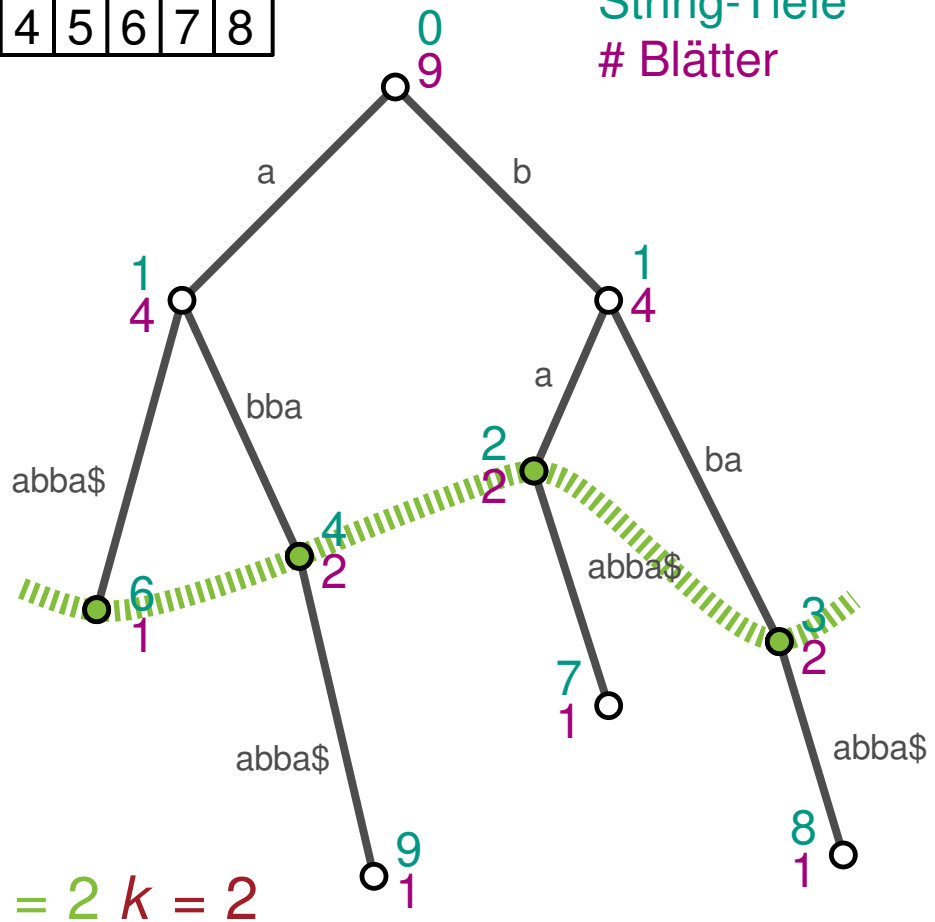
- Annotiere jeden Knoten mit String-Tiefe und Anzahl Blätter unter ihm

  Entferne dabei Sentinel-Blätter

- Für Query mit Eingaben ℓ und k:
  - DFS: Sammle jeden höchsten Knoten mit String-Tiefe $\geq \ell$

    $\rightarrow$ **Kandidaten**, speichere Suffixposition und Anzahl Blätter
  - Sortiere Kandidaten stabil nach Anzahl Blättern
  - Gebe k-ten Kandidaten aus.

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8



Query: $\ell = 2$ $k = 2$

TopK-Queries – Algorithmus

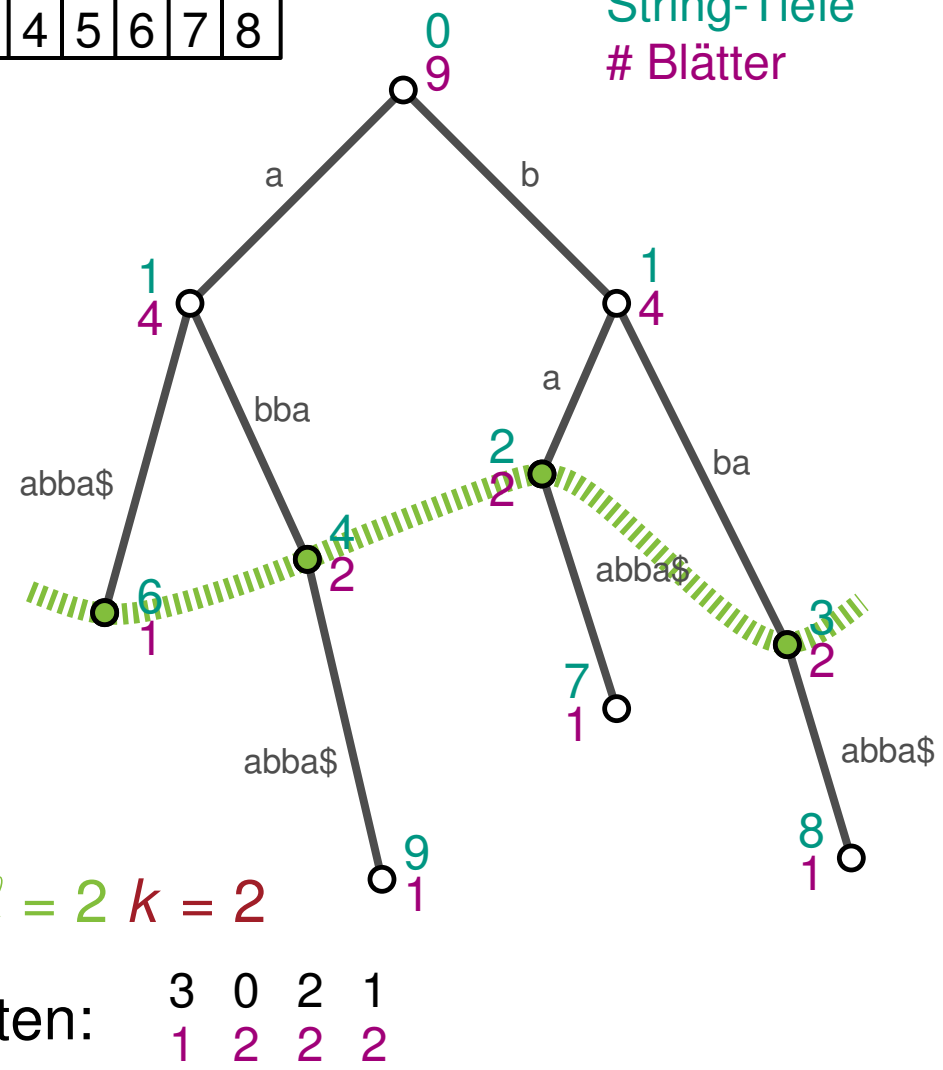
- Annotiere jeden Knoten mit String-Tiefe und Anzahl Blätter unter ihm

  Entferne dabei Sentinel-Blätter

- Für Query mit Eingaben ℓ und k:
  - DFS: Sammle jeden höchsten Knoten mit String-Tiefe $\geq \ell$

    $\rightarrow$ **Kandidaten**, speichere Suffixposition und Anzahl Blätter
  - Sortiere Kandidaten stabil nach Anzahl Blättern
  - Gebe k-ten Kandidaten aus.

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8



Query: $\ell = 2$ $k = 2$

Kandidaten:

3	0	2	1
1	2	2	2

TopK-Queries – Algorithmus

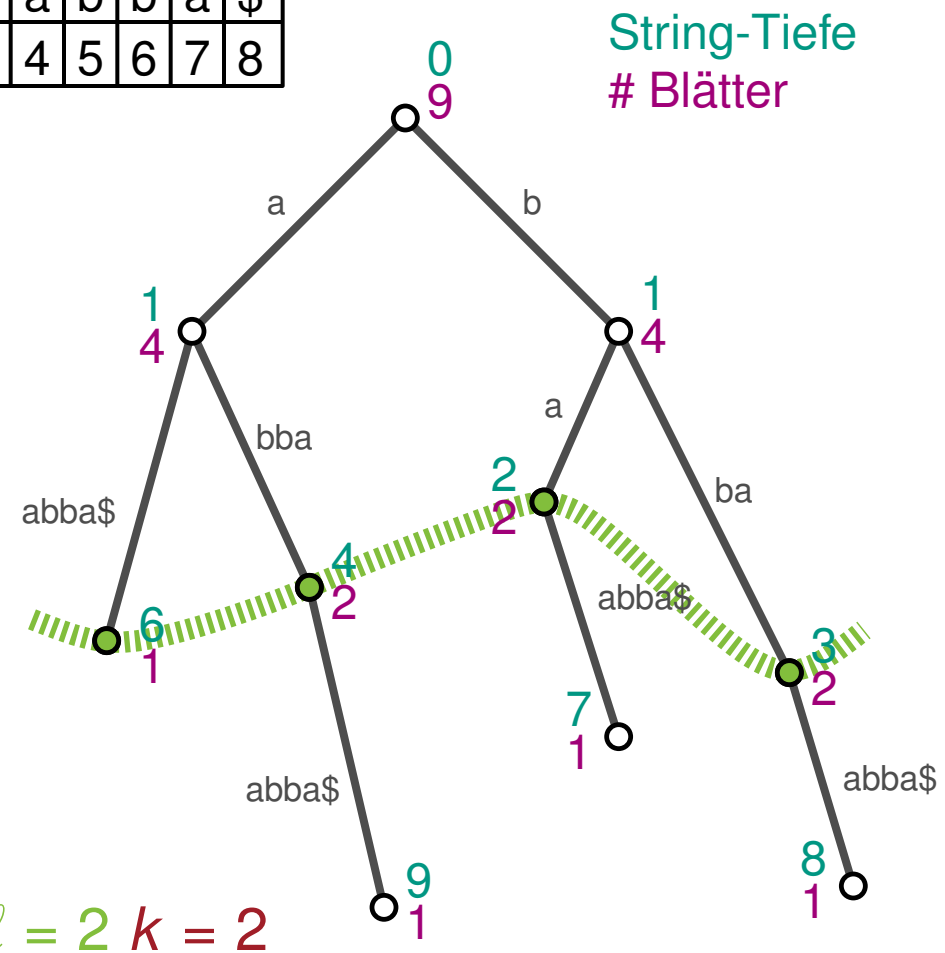
- Annotiere jeden Knoten mit String-Tiefe und Anzahl Blätter unter ihm

  Entferne dabei Sentinel-Blätter

- Für Query mit Eingaben ℓ und k:
  - DFS: Sammle jeden höchsten Knoten mit String-Tiefe $\geq \ell$

    $\rightarrow$ **Kandidaten**, speichere Suffixposition und Anzahl Blätter
  - Sortiere Kandidaten stabil nach Anzahl Blättern
  - Gebe k-ten Kandidaten aus.

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8



Query: $\ell = 2$ $k = 2$

Kandidaten:

3	0	2	1	$\rightarrow$	0	2	1	3
1	2	2	2		2	2	2	1

TopK-Queries – Algorithmus

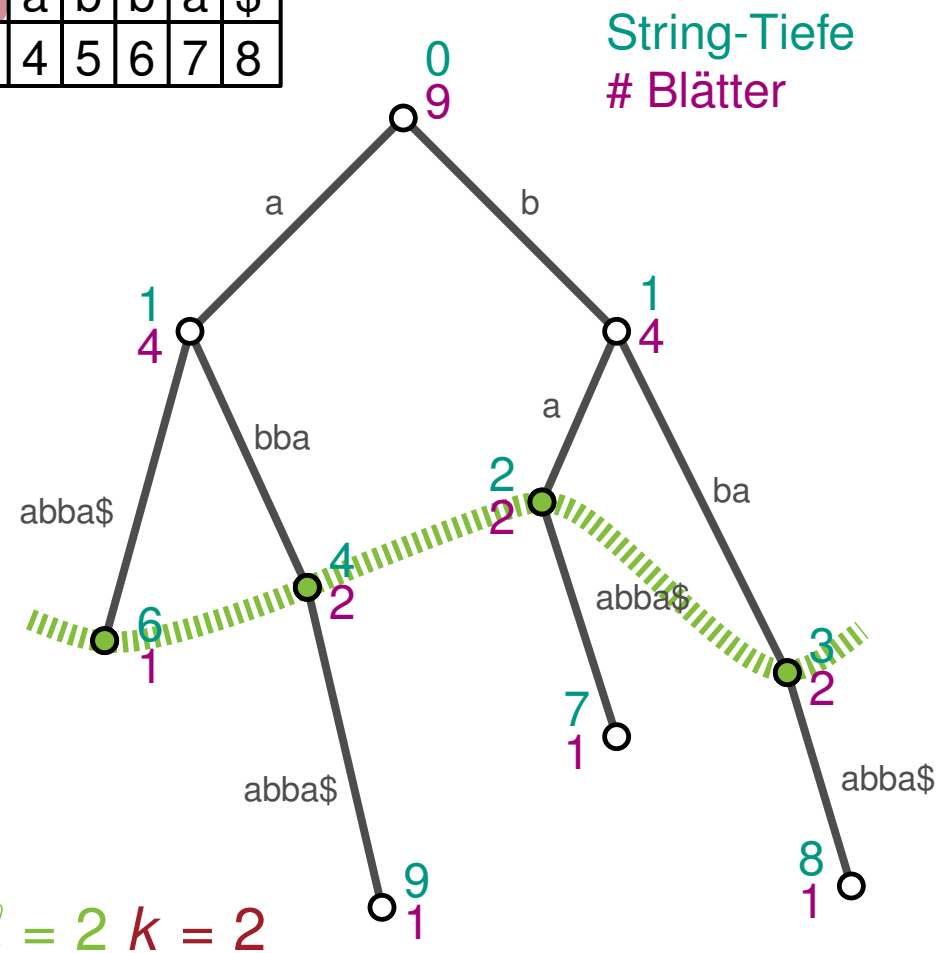
- Annotiere jeden Knoten mit String-Tiefe und Anzahl Blätter unter ihm

  Entferne dabei Sentinel-Blätter

- Für Query mit Eingaben ℓ und k:
  - DFS: Sammle jeden höchsten Knoten mit String-Tiefe $\geq \ell$

    $\rightarrow$ **Kandidaten**, speichere Suffixposition und Anzahl Blätter
  - Sortiere Kandidaten stabil nach Anzahl Blättern
  - Gebe k-ten Kandidaten aus.

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8

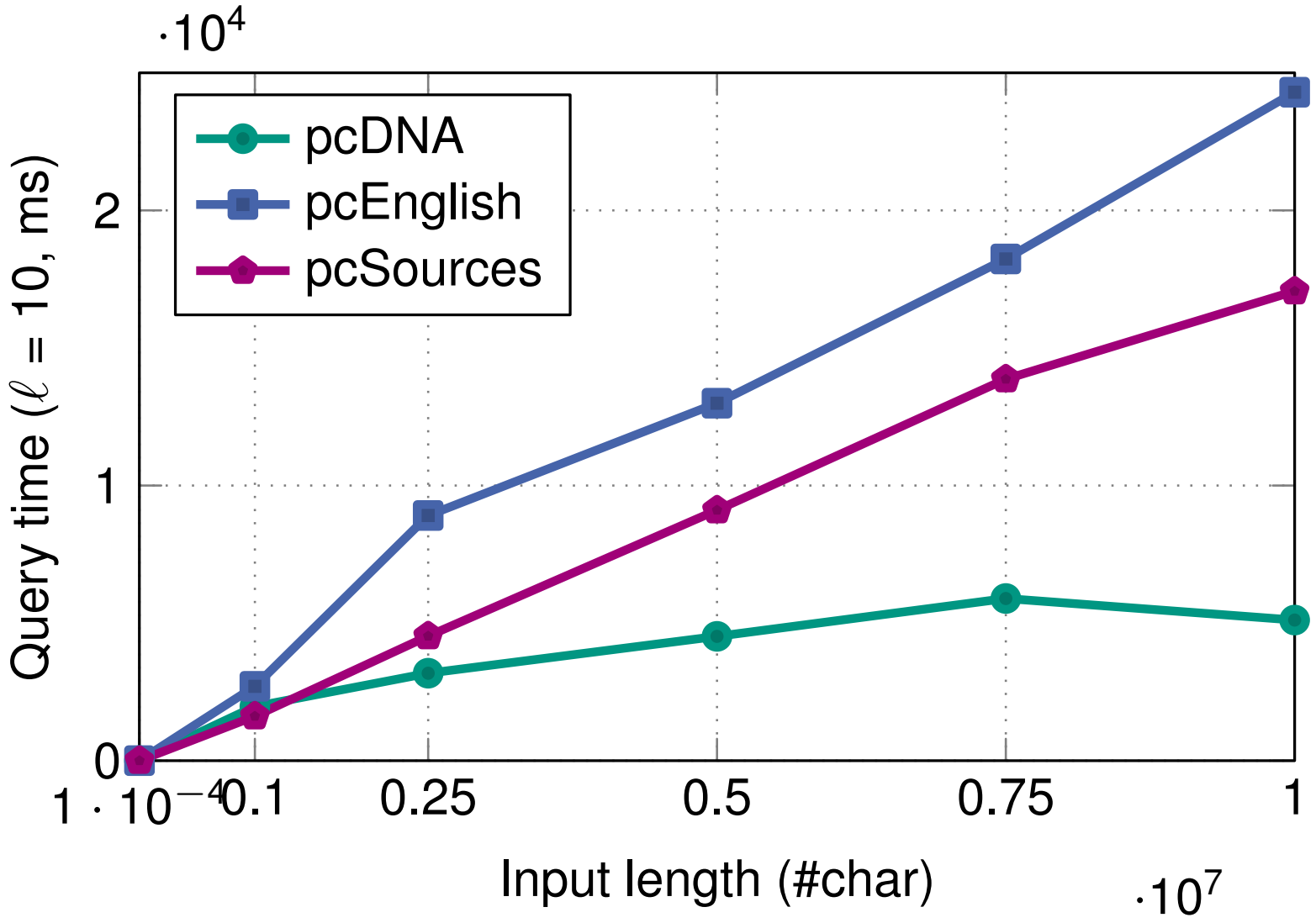


Query: $\ell = 2$ $k = 2$

Kandidaten:

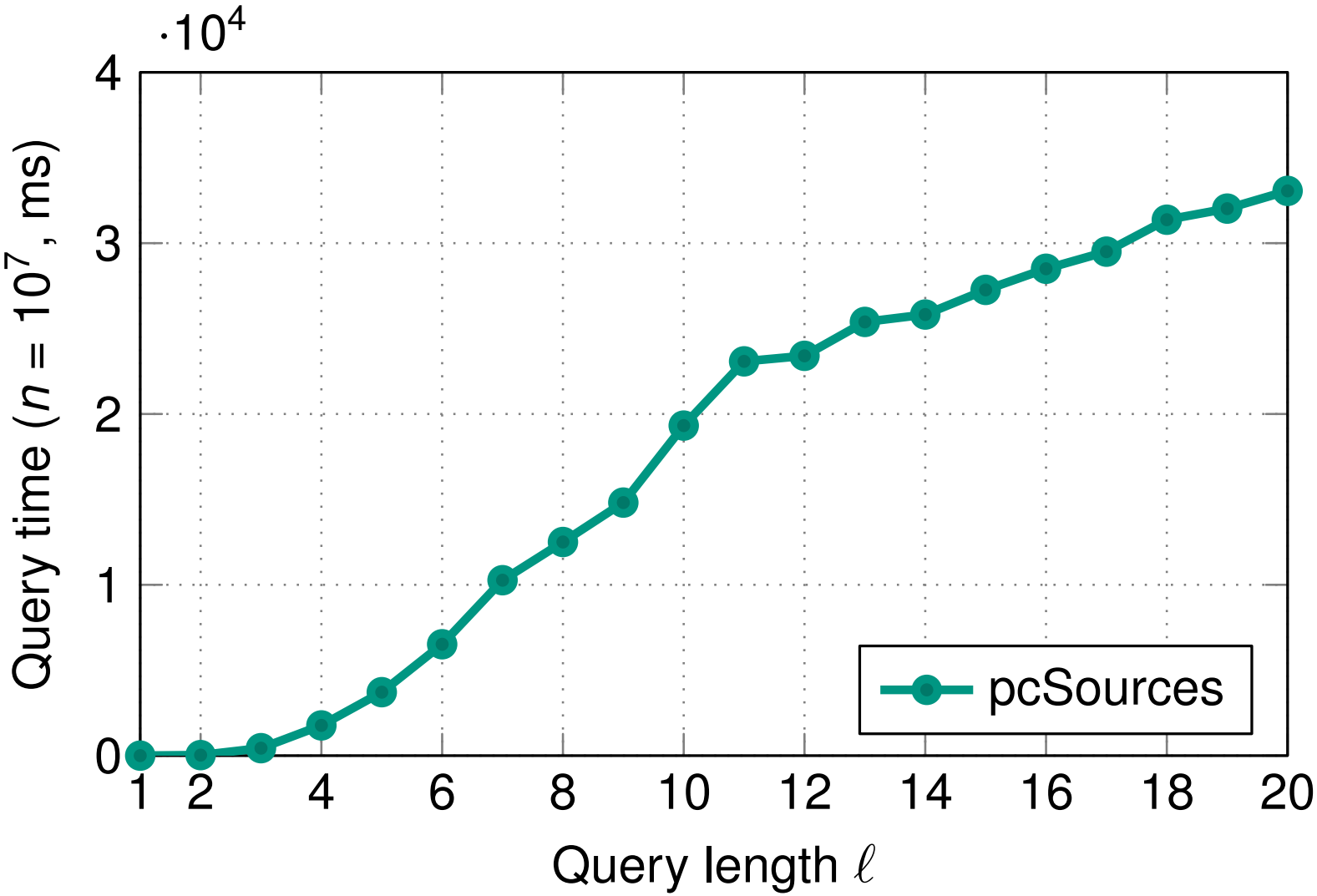
3	0	2	1	$\rightarrow$	0	2	1	3
1	2	2	2		2	2	2	1

TopK Queries Performance



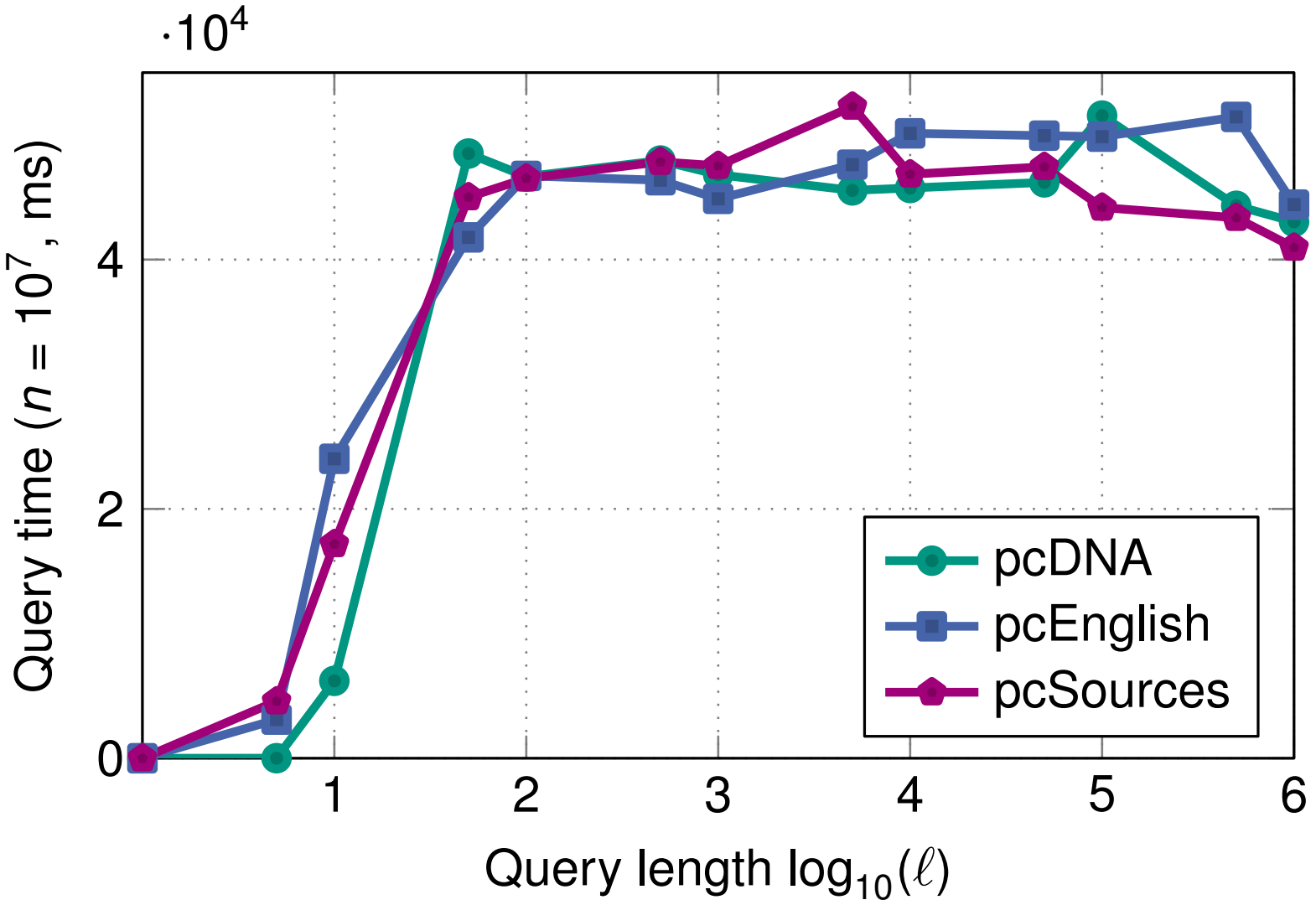
- Anfragezeit linear in Inputlänge
- Je repetetiver der Text, desto schneller die Anfrage: Weniger innere Knoten werden exploriert

TopK Query Performance nach Patternlänge



- Anfragezeiten wachsen für kleine Patternlängen. . .
- ... bis Plateau erreicht wird.
- Mit höherer Länge müssen mehr innere Knoten eingesammelt werden,
- Aber irgendwann lässt der Effekt nach, man hat schon fast alle

TopK Query Performance nach Patternlänge



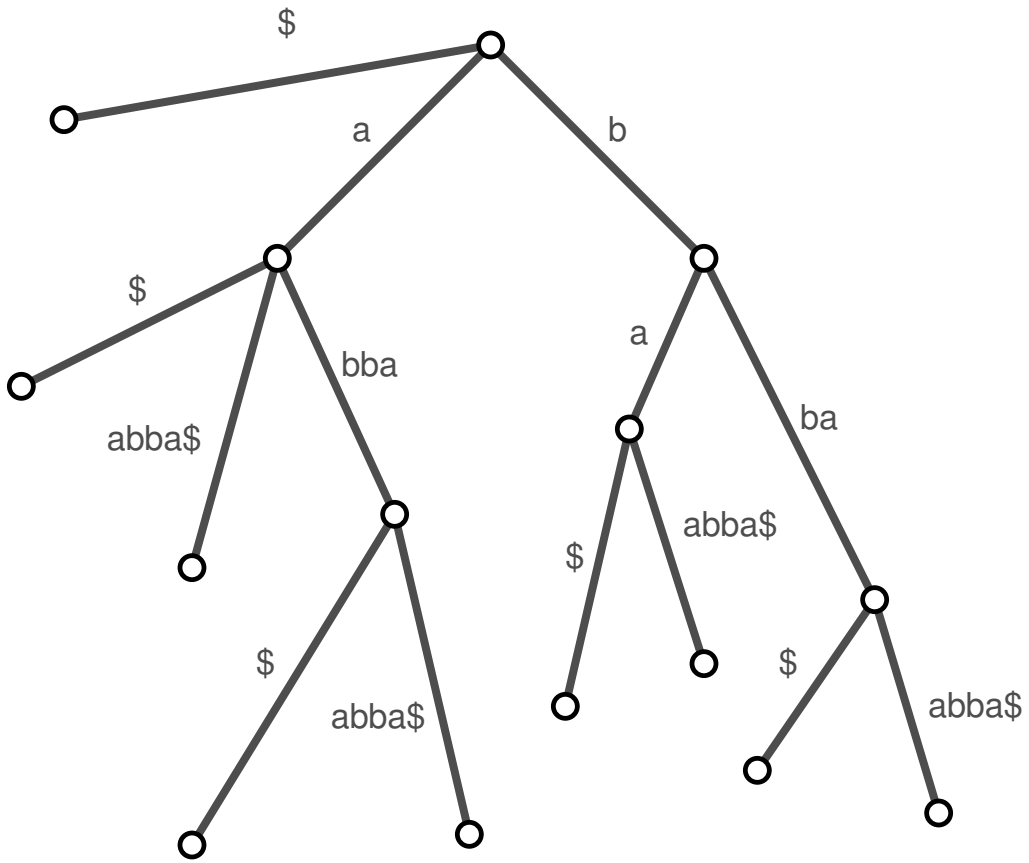
- Anfragezeiten wachsen für kleine Patternlängen...
- ... bis Plateau erreicht wird.
- Mit höherer Länge müssen mehr innere Knoten eingesammelt werden,
- Aber irgendwann lässt der Effekt nach, man hat schon fast alle

Repeat Queries

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8

Idee: Suche inneren Knoten mit String-Tiefe ℓ, der zwei Suffixe unter sich hat, die genau Differenz ℓ haben

- Annotiere jeden Knoten mit String-Tiefe und repräsentiertem Suffix
- Sammle innere Knoten, absteigend sortiert nach String-Tiefe
- Für jeden inneren Knoten *v*:
  - Sammle sortierte Liste aller Suffixe unter dem Knoten
  - Falls es zwei Suffixe gibt mit Differenz der String-Tiefe von *v*: Gebe Ergebnis aus

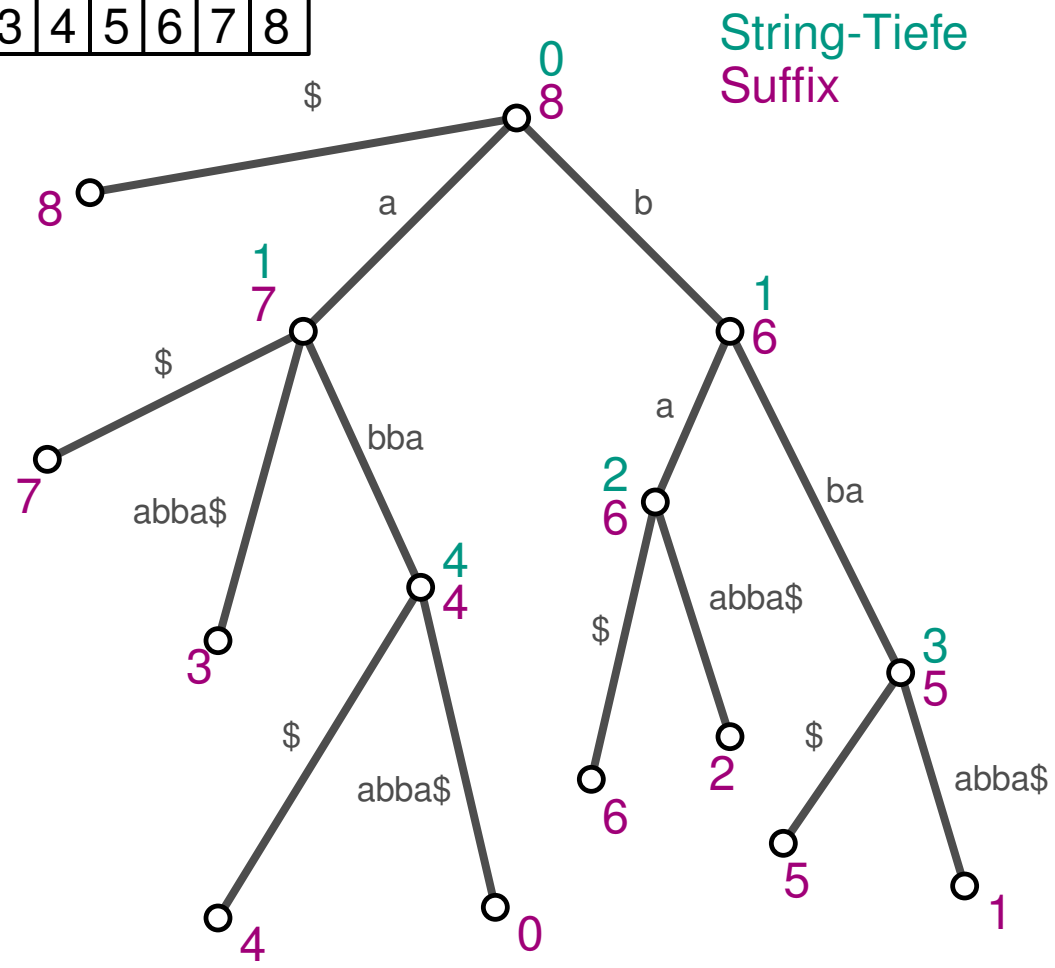


Repeat Queries

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8

Idee: Suche inneren Knoten mit String-Tiefe ℓ, der zwei Suffixe unter sich hat, die genau Differenz ℓ haben

- Annotiere jeden Knoten mit String-Tiefe und repräsentiertem Suffix
- Sammle innere Knoten, absteigend sortiert nach String-Tiefe
- Für jeden inneren Knoten *v*:
  - Sammle sortierte Liste aller Suffixe unter dem Knoten
  - Falls es zwei Suffixe gibt mit Differenz der String-Tiefe von *v*: Gebe Ergebnis aus

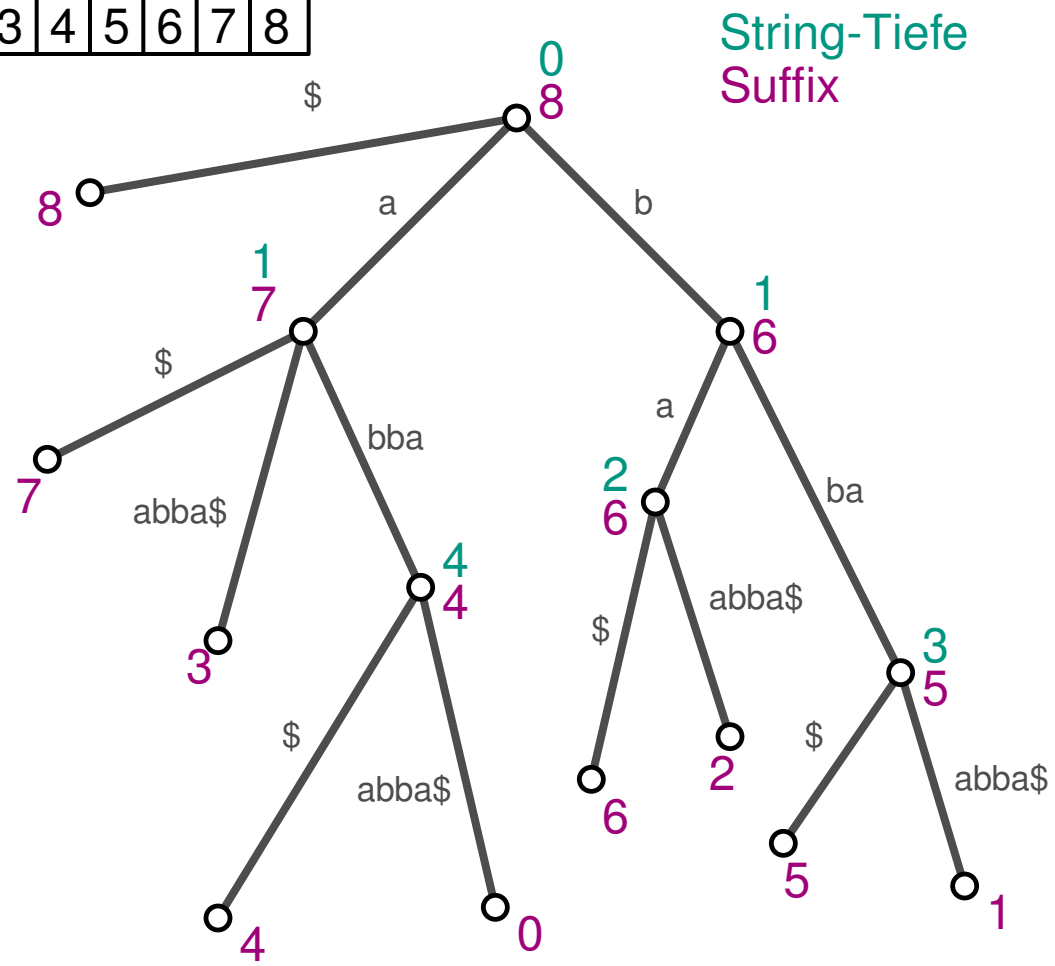


Repeat Queries

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8

Idee: Suche inneren Knoten mit String-Tiefe ℓ, der zwei Suffixe unter sich hat, die genau Differenz ℓ haben

- Annotiere jeden Knoten mit String-Tiefe und repräsentiertem Suffix
- Sammle innere Knoten, absteigend sortiert nach String-Tiefe
- Für jeden inneren Knoten *v*:
  - Sammle sortierte Liste aller Suffixe unter dem Knoten
  - Falls es zwei Suffixe gibt mit Differenz der String-Tiefe von *v*: Gebe Ergebnis aus



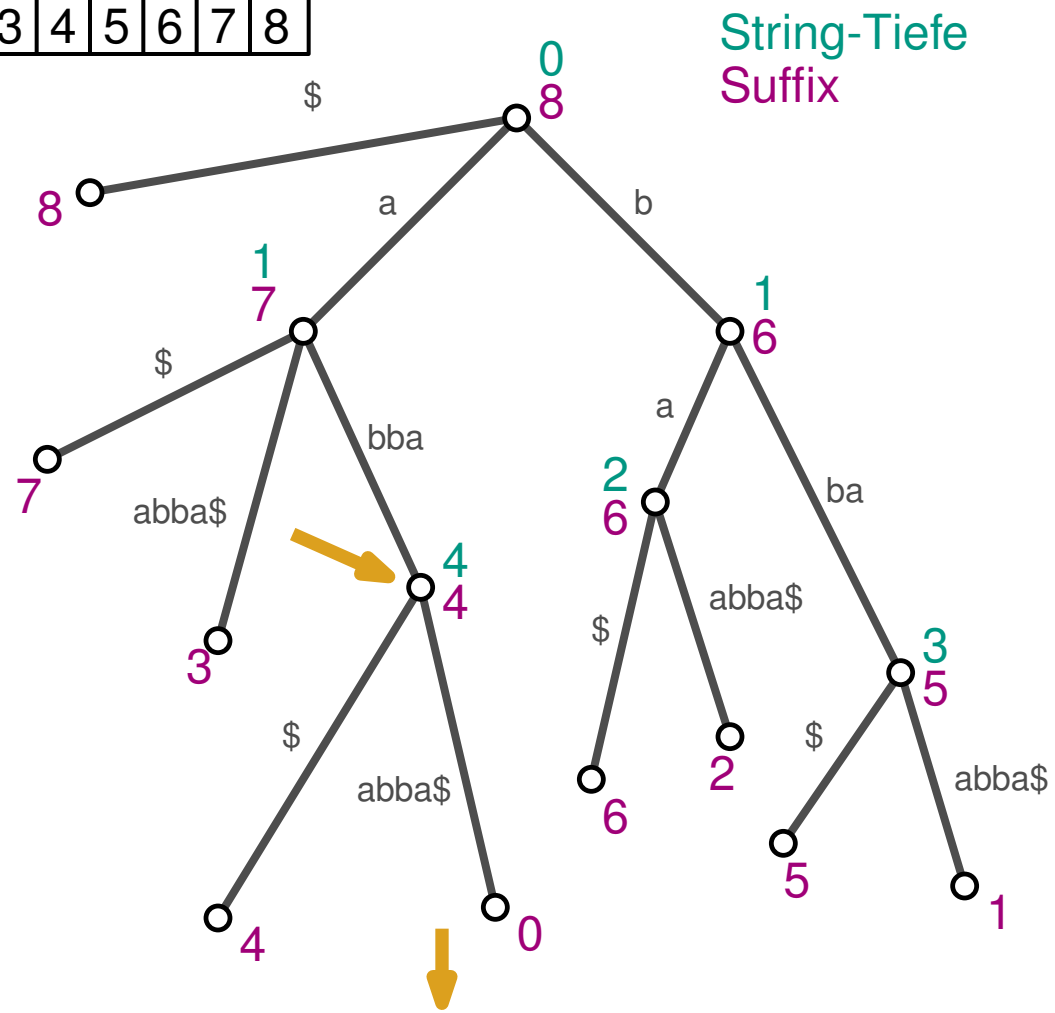
Innere Knoten: 4 3 2 1 1 0

Repeat Queries

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8

Idee: Suche inneren Knoten mit String-Tiefe ℓ, der zwei Suffixe unter sich hat, die genau Differenz ℓ haben

- Annotiere jeden Knoten mit String-Tiefe und repräsentiertem Suffix
- Sammle innere Knoten, absteigend sortiert nach String-Tiefe
- Für jeden inneren Knoten *v*:
  - Sammle sortierte Liste aller Suffixe unter dem Knoten
  - Falls es zwei Suffixe gibt mit Differenz der String-Tiefe von *v*: Gebe Ergebnis aus



Innere Knoten: 4 3 2 1 1 0

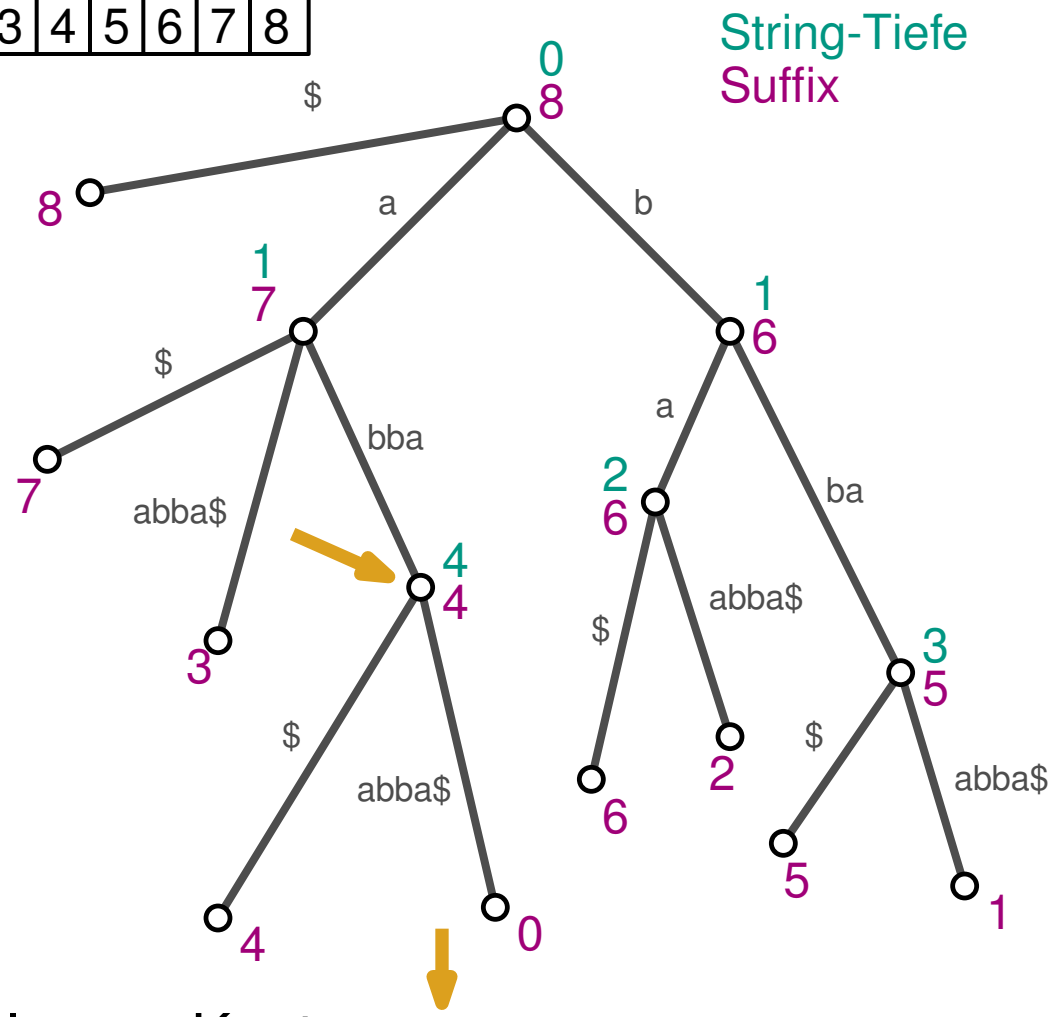
Suffixe: 0 4

Repeat Queries

a	b	b	a	a	b	b	a	$
0	1	2	3	4	5	6	7	8

Idee: Suche inneren Knoten mit String-Tiefe ℓ, der zwei Suffixe unter sich hat, die genau Differenz ℓ haben

- Annotiere jeden Knoten mit String-Tiefe und repräsentiertem Suffix
- Sammle innere Knoten, absteigend sortiert nach String-Tiefe
- Für jeden inneren Knoten v:
  - Sammle sortierte Liste aller Suffixe unter dem Knoten
  - Falls es zwei Suffixe gibt mit Differenz der String-Tiefe von v: Gebe Ergebnis aus



Innere Knoten: 4 3 2 1 1 0

Suffixe: 0 4 $\rightarrow$ Ergebnis: $(0, 8)$

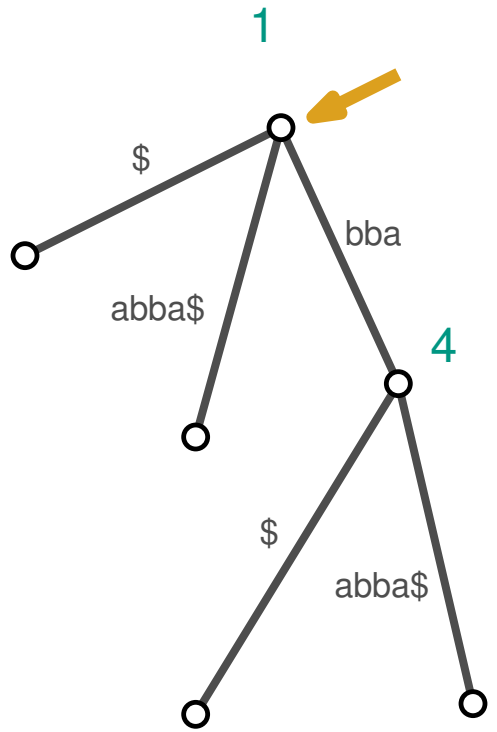
Suffixe einsammeln als DP

String-Tiefe
Suffix-Liste

Ziel: Berechne sortierte Liste aller Suffixe unter einem Knoten → Merge dazu alle Kinderlisten

- Es gibt zwei Fälle:
  - Kind ist Blatt → triviale einelementige Liste
  - Kind ist innerer Knoten.
    - **Beobachtung:** Alle Kinder haben höhere String-Tiefe

      → Liste schon berechnet
- Merge die vorberechneten sortierten Listen aller Kinder
- Implementiert als DP.



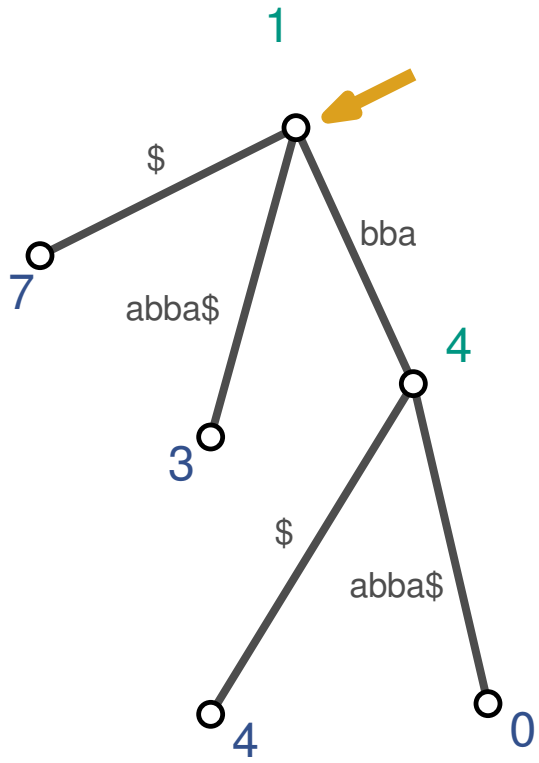
Suffixe einsammeln als DP

String-Tiefe
Suffix-Liste

Ziel: Berechne sortierte Liste aller Suffixe unter einem Knoten → Merge dazu alle Kinderlisten

- Es gibt zwei Fälle:
  - Kind ist Blatt → triviale einelementige Liste
  - Kind ist innerer Knoten.
    - **Beobachtung:** Alle Kinder haben höhere String-Tiefe

      → Liste schon berechnet
- Merge die vorberechneten sortierten Listen aller Kinder
- Implementiert als DP.



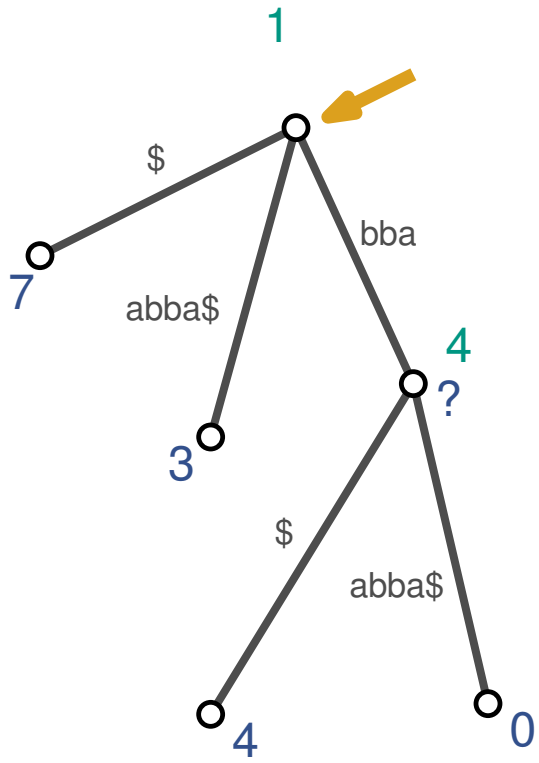
Suffixe einsammeln als DP

String-Tiefe
Suffix-Liste

Ziel: Berechne sortierte Liste aller Suffixe unter einem Knoten → Merge dazu alle Kinderlisten

- Es gibt zwei Fälle:
  - Kind ist Blatt → triviale einelementige Liste
  - Kind ist innerer Knoten.
    - **Beobachtung:** Alle Kinder haben höhere String-Tiefe

      → Liste schon berechnet
- Merge die vorberechneten sortierten Listen aller Kinder
- Implementiert als DP.

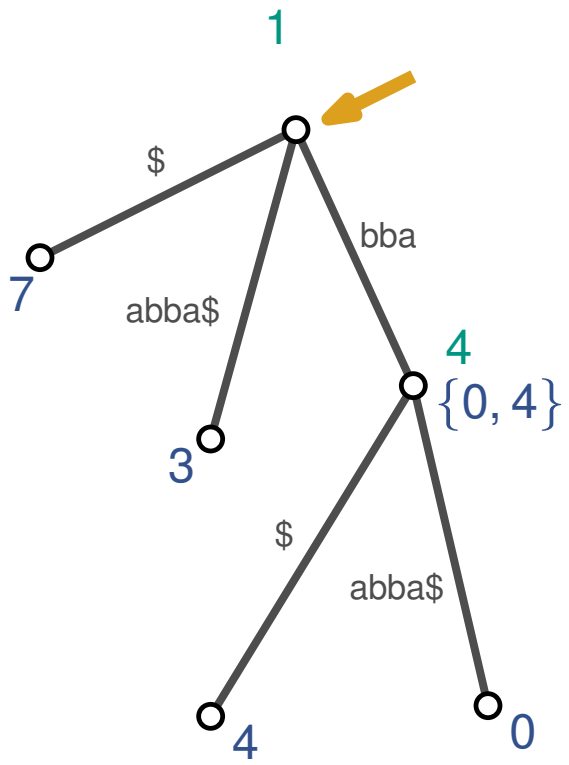


Suffixe einsammeln als DP

Ziel: Berechne sortierte Liste aller Suffixe unter einem Knoten $\rightarrow$ Merge dazu alle Kinderlisten

- Es gibt zwei Fälle:
  - Kind ist Blatt $\rightarrow$ triviale einelementige Liste
  - Kind ist innerer Knoten.
    - **Beobachtung:** Alle Kinder haben höhere String-Tiefe

      $\rightarrow$ Liste schon berechnet
- Merge die vorberechneten sortierten Listen aller Kinder
- Implementiert als DP.

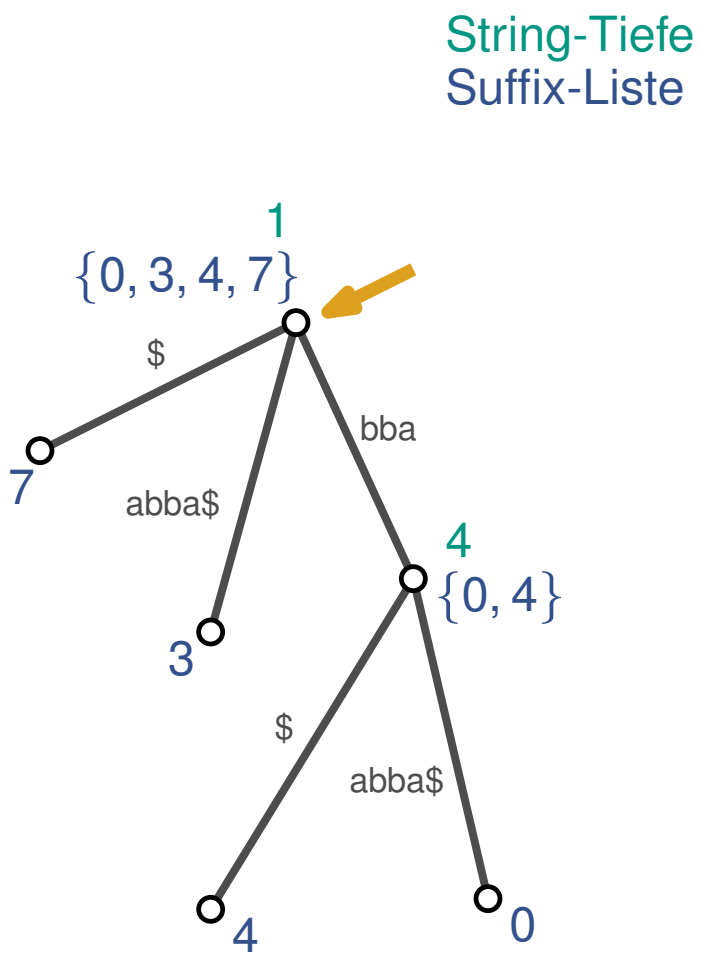


Suffixe einsammeln als DP

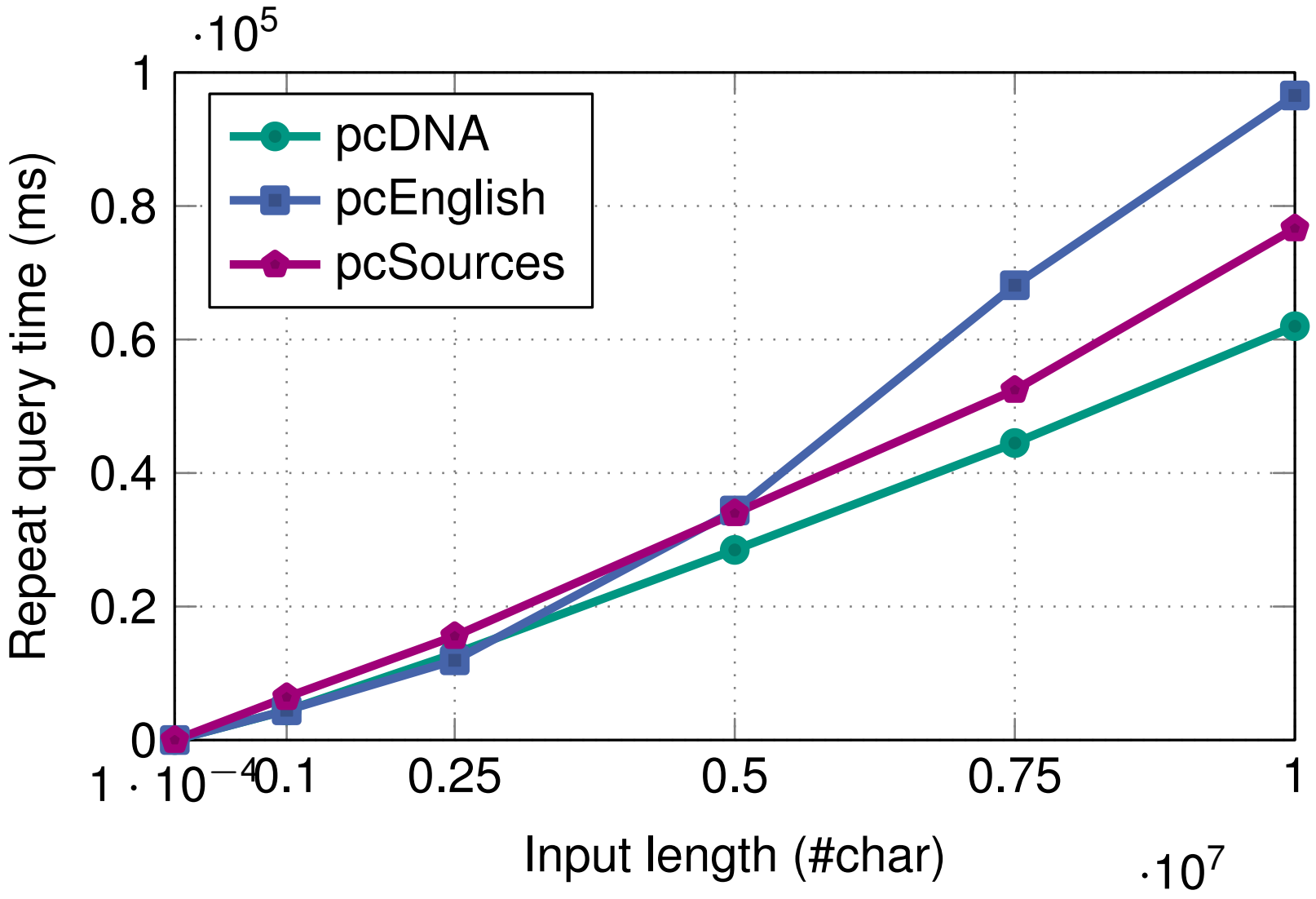
Ziel: Berechne sortierte Liste aller Suffixe unter einem Knoten $\rightarrow$ Merge dazu alle Kinderlisten

- Es gibt zwei Fälle:
  - Kind ist Blatt $\rightarrow$ triviale einelementige Liste
  - Kind ist innerer Knoten.
    - **Beobachtung:** Alle Kinder haben höhere String-Tiefe

      $\rightarrow$ Liste schon berechnet
- Merge die vorberechneten sortierten Listen aller Kinder
- Implementiert als DP.



Repeat Query Performance



- Query-Performance wieder linear in Eingabelänge
- Wieder: repetitivere Texte einfacher

Erkenntnisse

- Automatisierte Tests lohnen sich
- Schon während dem Tuning automatisiert evaluieren!
- Tuning nicht so leicht, wenn nicht klar ist, wie die Eingaben verteilt sind
- Was anderso gut funktioniert, muss nicht auch hier funktionieren: Vorberechnung

Erkenntnisse

- Automatisierte Tests lohnen sich
- Schon während dem Tuning automatisiert evaluieren!
- Tuning nicht so leicht, wenn nicht klar ist, wie die Eingaben verteilt sind
- Was anderso gut funktioniert, muss nicht auch hier funktionieren: Vorberechnung

Fragen?