Relative Compressed Reverse Suffix Array

Mano Prakash Parthasarathi

STACS 2026

March 12, 2026
Grenoble, France

joint work with
Muhammed Oguzhan Kulekci, Rahul Shah,
and Sharma V. Thankachan

Table of Contents

Suffixes

- Given a text $T[1, n]$, the suffix beginning at position j is the substring $T[j, n]$.
- **Example:** For Text $T =$ "BANANA$"

1. BANANA$
2. ANANA$
3. NANA$
4. ANA$
5. NA$
6. A$
7. $

Suffixes

- Given a text $T[1, n]$, the suffix beginning at position j is the substring $T[j, n]$.
- **Example:** For Text $T =$ "BANANA$"

1. BANANA$
2. ANANA$
3. NANA$
4. ANA$
5. NA$
6. A$
7. $

Suffixes

- Given a text $T[1, n]$, the suffix beginning at position j is the substring $T[j, n]$.
- **Example:** For Text $T =$ "BANANA$"

1. BANANA$
2. ANANA$
3. NANA$
4. ANA$
5. NA$
6. A$
7. $

Suffixes

- Given a text $T[1, n]$, the suffix beginning at position j is the substring $T[j, n]$.
- **Example:** For Text $T =$ "BANANA$"

1. BANANA$
2. ANANA$
3. NANA$
4. ANA$
5. NA$
6. A$
7. $

Suffixes

- Given a text $T[1, n]$, the suffix beginning at position j is the substring $T[j, n]$.
- **Example:** For Text $T =$ "BANANA$"

1. BANANA$
2. ANANA$
3. NANA$
4. ANA$
5. NA$
6. A$
7. $

Suffixes

- Given a text $T[1, n]$, the suffix beginning at position j is the substring $T[j, n]$.
- **Example:** For Text $T =$ "BANANA$"

1. BANANA$
2. ANANA$
3. NANA$
4. ANA$
5. NA$
6. A$
7. $

Suffixes

- Given a text $T[1, n]$, the suffix beginning at position j is the substring $T[j, n]$.
- **Example:** For Text $T =$ "BANANA$"

1. BANANA$
2. ANANA$
3. NANA$
4. ANA$
5. NA$
6. A$
7. $

Suffixes

- Given a text $T[1, n]$, the suffix beginning at position j is the substring $T[j, n]$.
- **Example:** For Text $T =$ "BANANA$"

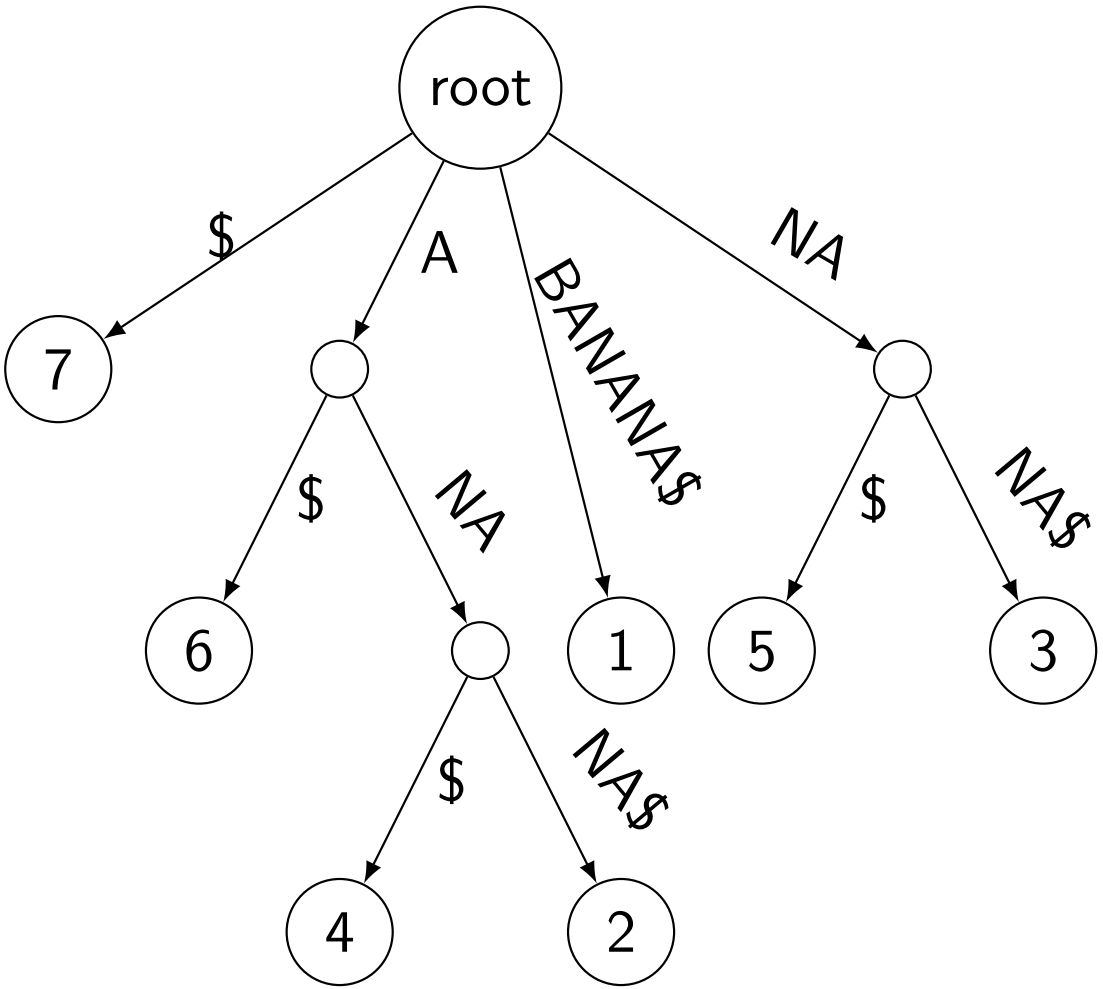
1. BANANA$
2. ANANA$
3. NANA$
4. ANA$
5. NA$
6. A$
7. $

Suffixes

- Given a text $T[1, n]$, the suffix beginning at position j is the substring $T[j, n]$.
- **Example:** For Text $T =$ "BANANA$"

1. BANANA$
2. ANANA$
3. NANA$
4. ANA$
5. NA$
6. A$
7. $

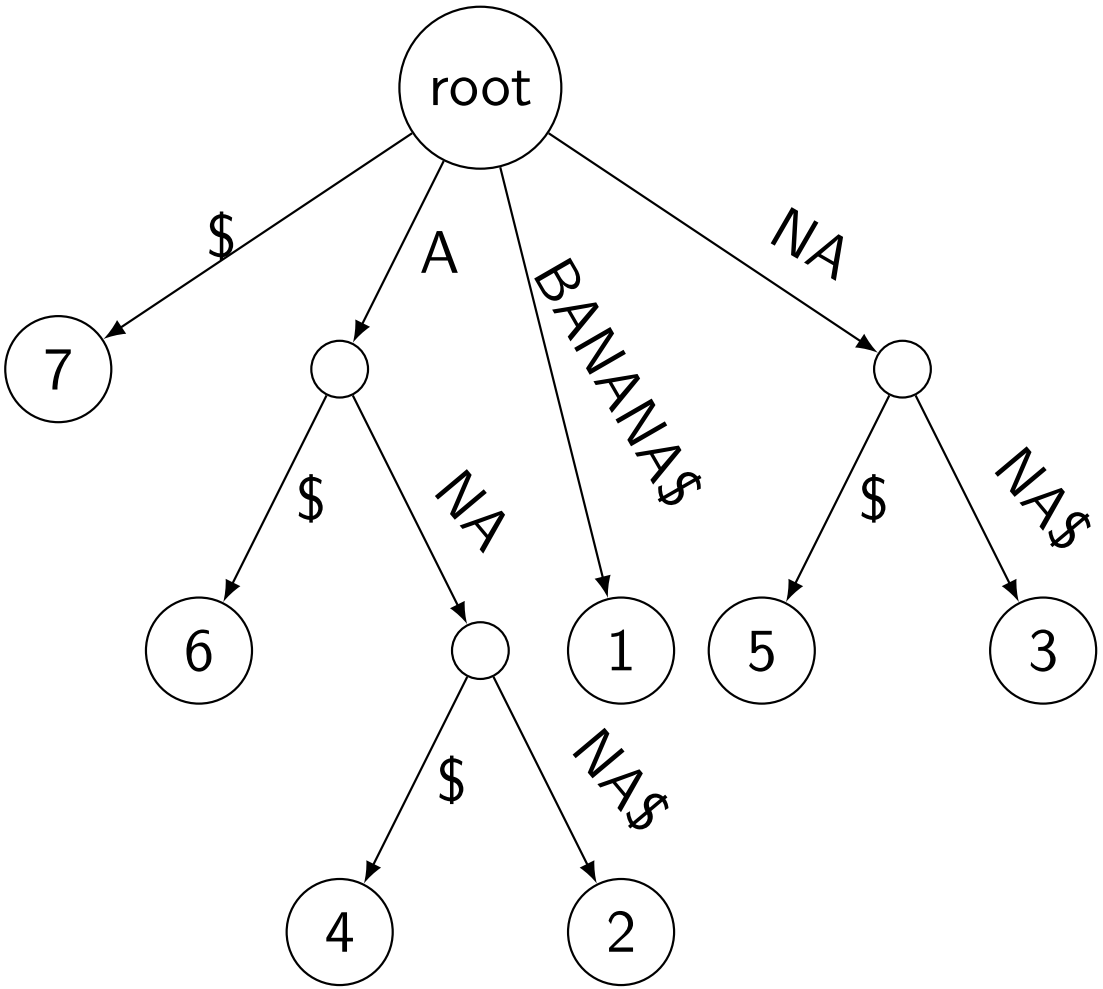
Suffix Trees



- Leaves are suffixes of T in lexicographic order (in the left-to-right leaf order).
- Internal nodes capture common prefixes, and their string depth equals that common-prefix length.

Figure: Suffix Tree of T = "BANANA$"

Suffix Trees



- Leaves are suffixes of T in lexicographic order (in the left-to-right leaf order).
- Internal nodes capture common prefixes, and their string depth equals that common-prefix length.

Figure: Suffix Tree of T = "BANANA$"

SA, ISA, LCP, and BWT

- $\mathrm{SA}[i]$: i^{th} lexicographically smallest suffix.
- $\mathrm{ISA}[i]$: lexicographic rank of suffix $T[i \ldots n]$.
- $\mathrm{LCP}[i]$: length of the longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- $\mathrm{BWT}[i] := T[(\mathrm{SA}[i]-1)(mod\ n)]$ (Burrows-Wheeler Transform)

i	i^{th} smallest suffix	$\mathrm{SA}[i]$	$\mathrm{ISA}[i]$	$\mathrm{LCP}[i]$	$\mathrm{BWT}[i]$
1	\$	7	5	0	A
2	A\$	6	4	1	N
3	ANA\$	4	7	3	N
4	ANANA\$	2	3	0	B
5	BANANA\$	1	6	0	\$
6	NA\$	5	2	2	A
7	NANA\$	3	1	–	A

SA, ISA, LCP, and BWT

- $\mathrm{SA}[i]$: i^{th} lexicographically smallest suffix.
- $\mathrm{ISA}[i]$: lexicographic rank of suffix $T[i \ldots n]$.
- $\mathrm{LCP}[i]$: length of the longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- $\mathrm{BWT}[i] := T[(\mathrm{SA}[i]-1)(mod\ n)]$ (Burrows-Wheeler Transform)

i	i^{th} smallest suffix	$\mathrm{SA}[i]$	$\mathrm{ISA}[i]$	$\mathrm{LCP}[i]$	$\mathrm{BWT}[i]$
1	\$	7	5	0	A
2	A\$	6	4	1	N
3	ANA\$	4	7	3	N
4	ANANA\$	2	3	0	B
5	BANANA\$	1	6	0	\$
6	NA\$	5	2	2	A
7	NANA\$	3	1	–	A

SA, ISA, LCP, and BWT

- $\mathrm{SA}[i]$: i^{th} lexicographically smallest suffix.
- $\mathrm{ISA}[i]$: lexicographic rank of suffix $T[i \ldots n]$.
- $\mathrm{LCP}[i]$: length of the longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- $\mathrm{BWT}[i] := T[(\mathrm{SA}[i]-1)(mod\ n)]$ (Burrows-Wheeler Transform)

i	i^{th} smallest suffix	$\mathrm{SA}[i]$	$\mathrm{ISA}[i]$	$\mathrm{LCP}[i]$	$\mathrm{BWT}[i]$
1	\$	7	5	0	A
2	A\$	6	4	1	N
3	ANA\$	4	7	3	N
4	ANANA\$	2	3	0	B
5	BANANA\$	1	6	0	\$
6	NA\$	5	2	2	A
7	NANA\$	3	1	–	A

SA, ISA, LCP, and BWT

- $\mathrm{SA}[i]$: i^{th} lexicographically smallest suffix.
- $\mathrm{ISA}[i]$: lexicographic rank of suffix $T[i \ldots n]$.
- $\mathrm{LCP}[i]$: length of the longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- $\mathrm{BWT}[i] := T[(\mathrm{SA}[i]-1)(mod\ n)]$ (Burrows-Wheeler Transform)

i	i^{th} smallest suffix	$\mathrm{SA}[i]$	$\mathrm{ISA}[i]$	$\mathrm{LCP}[i]$	$\mathrm{BWT}[i]$
1	\$	7	5	0	A
2	A\$	6	4	1	N
3	ANA\$	4	7	3	N
4	ANANA\$	2	3	0	B
5	BANANA\$	1	6	0	\$
6	NA\$	5	2	2	A
7	NANA\$	3	1	–	A

SA, ISA, LCP, and BWT

- $\mathrm{SA}[i]$: i^{th} lexicographically smallest suffix.
- $\mathrm{ISA}[i]$: lexicographic rank of suffix $T[i \ldots n]$.
- $\mathrm{LCP}[i]$: length of the longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- $\mathrm{BWT}[i] := T[(\mathrm{SA}[i]-1)(mod\ n)]$ (Burrows-Wheeler Transform)

i	i^{th} smallest suffix	$\mathrm{SA}[i]$	$\mathrm{ISA}[i]$	$\mathrm{LCP}[i]$	$\mathrm{BWT}[i]$
1	\$	7	5	0	A
2	A\$	6	4	1	N
3	ANA\$	4	7	3	N
4	ANANA\$	2	3	0	B
5	BANANA\$	1	6	0	\$
6	NA\$	5	2	2	A
7	NANA\$	3	1	–	A

SA, ISA, LCP, and BWT

- $\mathrm{SA}[i]$: i^{th} lexicographically smallest suffix.
- $\mathrm{ISA}[i]$: lexicographic rank of suffix $T[i \ldots n]$.
- $\mathrm{LCP}[i]$: length of the longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- $\mathrm{BWT}[i] := T[(\mathrm{SA}[i]-1)(mod\ n)]$ (Burrows-Wheeler Transform)

i	i^{th} smallest suffix	$\mathrm{SA}[i]$	$\mathrm{ISA}[i]$	$\mathrm{LCP}[i]$	$\mathrm{BWT}[i]$
1	\$	7	5	0	A
2	A\$	6	4	1	N
3	ANA\$	4	7	3	N
4	ANANA\$	2	3	0	B
5	BANANA\$	1	6	0	\$
6	NA\$	5	2	2	A
7	NANA\$	3	1	–	A

SA, ISA, LCP, and BWT

- $\mathrm{SA}[i]$: i^{th} lexicographically smallest suffix.
- $\mathrm{ISA}[i]$: lexicographic rank of suffix $T[i \ldots n]$.
- $\mathrm{LCP}[i]$: length of the longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- $\mathrm{BWT}[i] := T[(\mathrm{SA}[i]-1)(mod\ n)]$ (Burrows-Wheeler Transform)

i	i^{th} smallest suffix	$\mathrm{SA}[i]$	$\mathrm{ISA}[i]$	$\mathrm{LCP}[i]$	$\mathrm{BWT}[i]$
1	\$	7	5	0	A
2	A\$	6	4	1	N
3	ANA\$	4	7	3	N
4	ANANA\$	2	3	0	B
5	BANANA\$	1	6	0	\$
6	NA\$	5	2	2	A
7	NANA\$	3	1	–	A

SA, ISA, LCP, and BWT

- $\mathrm{SA}[i]$: i^{th} lexicographically smallest suffix.
- $\mathrm{ISA}[i]$: lexicographic rank of suffix $T[i \ldots n]$.
- $\mathrm{LCP}[i]$: length of the longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- $\mathrm{BWT}[i] := T[(\mathrm{SA}[i]-1)(mod\ n)]$ (Burrows-Wheeler Transform)

i	i^{th} smallest suffix	$\mathrm{SA}[i]$	$\mathrm{ISA}[i]$	$\mathrm{LCP}[i]$	$\mathrm{BWT}[i]$
1	\$	7	5	0	A
2	A\$	6	4	1	N
3	ANA\$	4	7	3	N
4	ANANA\$	2	3	0	B
5	BANANA\$	1	6	0	\$
6	NA\$	5	2	2	A
7	NANA\$	3	1	–	A

Motivation

- Several bidirectional and approximate-matching tasks need suffix information for the reversed text ($\overleftarrow{T}$) too.

- The naive fix is to build a full index on $\overleftarrow{T}$, but that doubles the memory usage.

Motivation

- Several bidirectional and approximate-matching tasks need suffix information for the reversed text ($\overleftarrow{T}$) too.

- The naive fix is to build a full index on $\overleftarrow{T}$, but that doubles the memory usage.

Problem Statement

Question

Using only the data structures of T and *small additional space*, can one support suffix-array queries on $\overleftarrow{T}$ efficiently ?

- Before diving into our result, let's see some known approaches.

Problem Statement

Question

Using only the data structures of T and *small additional space*, can one support suffix-array queries on $\overleftarrow{T}$ efficiently ?

- Before diving into our result, let's see some known approaches.

Some Approaches

Approach	Space	Time
Store SA of $\overleftarrow{\mathrm{T}}$ explicitly (Trivial)	$n \log n$ bits	$\mathcal{O}(1)$ (direct access)
Compressed Suffix Array [1] / FM-index [2] of $\overleftarrow{\mathrm{T}}$	$n \log \sigma +$ l.o.t bits	$\mathcal{O}(\log n)$

Some Approaches

Approach	Space	Time
Store SA of $\overleftarrow{\mathrm{T}}$ explicitly (Trivial)	$n \log n$ bits	$\mathcal{O}(1)$ (direct access)
Compressed Suffix Array [1] / FM-index [2] of $\overleftarrow{\mathrm{T}}$	$n \log \sigma +$ l.o.t bits	$\mathcal{O}(\log n)$

Some Approaches

Approach	Space	Time
Store SA of $\overleftarrow{\mathrm{T}}$ explicitly (Trivial)	$n \log n$ bits	$\mathcal{O}(1)$ (direct access)
Compressed Suffix Array [1] / FM-index [2] of $\overleftarrow{\mathrm{T}}$	$n \log \sigma +$ l.o.t bits	$\mathcal{O}(\log n)$

Some Approaches

Approach	Space	Time
Store SA of $\overleftarrow{\mathrm{T}}$ explicitly (Trivial)	$n \log n$ bits	$\mathcal{O}(1)$ (direct access)
Compressed Suffix Array [1] / FM-index [2] of $\overleftarrow{\mathrm{T}}$	$n \log \sigma +$ l.o.t bits	$\mathcal{O}(\log n)$

Some Approaches

Approach	Space	Time
Store SA of $\overleftarrow{\mathrm{T}}$ explicitly (Trivial)	$n \log n$ bits	$\mathcal{O}(1)$ (direct access)
Compressed Suffix Array [1] / FM-index [2] of $\overleftarrow{\mathrm{T}}$	$n \log \sigma +$ l.o.t bits	$\mathcal{O}(\log n)$

Some Approaches

Approach	Space	Time
Store SA of $\overleftarrow{\mathrm{T}}$ explicitly (Trivial)	$n \log n$ bits	$\mathcal{O}(1)$ (direct access)
Compressed Suffix Array [1] / FM-index [2] of $\overleftarrow{\mathrm{T}}$	$n \log \sigma +$ l.o.t bits	$\mathcal{O}(\log n)$

Some Approaches

Approach	Space	Time
Store SA of $\overleftarrow{\mathrm{T}}$ explicitly (Trivial)	$n \log n$ bits	$\mathcal{O}(1)$ (direct access)
Compressed Suffix Array [1] / FM-index [2] of $\overleftarrow{\mathrm{T}}$	$n \log \sigma +$ l.o.t bits	$\mathcal{O}(\log n)$

Our Results

Theorem 1

SA queries on $\overleftarrow{\mathsf{T}}$ can be supported in time $t_{\mathrm{sa}} \cdot \mathcal{O}(\log^* n)$ with an auxiliary structure of $\mathcal{O}(n)$ bits of space.

Theorem 2

SA queries on $\overleftarrow{\mathsf{T}}$ can be supported with $\mathcal{O}(n/\kappa)$ bits and a query time $t_{\mathrm{sa}} \cdot \mathcal{O}((\kappa \log^* n + \kappa^3 \log \kappa) \cdot \log^2 \kappa)$ where κ is a parameter.

Our Results

Theorem 1

SA queries on $\overleftarrow{\mathsf{T}}$ can be supported in time $t_{\mathrm{sa}} \cdot \mathcal{O}(\log^* n)$ with an auxiliary structure of $\mathcal{O}(n)$ bits of space.

Theorem 2

SA queries on $\overleftarrow{\mathsf{T}}$ can be supported with $\mathcal{O}(n/\kappa)$ bits and a query time $t_{\mathrm{sa}} \cdot \mathcal{O}((\kappa \log^* n + \kappa^3 \log \kappa) \cdot \log^2 \kappa)$ where κ is a parameter.

pLCP data structure

- Recall $\mathrm{LCP}[i]$ is the length of longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- pLCP array is nothing but LCP array stored in text order.
- From Sadakane's result [3], pLCP can be stored in $2n + o(n)$ bits and queried in $\mathcal{O}(1)$ time.
- Similar to pLCP, one can also define pLCS which is LCS (Longest Common Suffix) stored in text order.

pLCP data structure

- Recall $\mathrm{LCP}[i]$ is the length of longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- pLCP array is nothing but LCP array stored in text order.
- From Sadakane's result [3], pLCP can be stored in $2n + o(n)$ bits and queried in $\mathcal{O}(1)$ time.
- Similar to pLCP, one can also define pLCS which is LCS (Longest Common Suffix) stored in text order.

pLCP data structure

- Recall $\mathrm{LCP}[i]$ is the length of longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- pLCP array is nothing but LCP array stored in text order.
- From Sadakane's result [3], pLCP can be stored in $2n + o(n)$ bits and queried in $\mathcal{O}(1)$ time.
- Similar to pLCP, one can also define pLCS which is LCS (Longest Common Suffix) stored in text order.

pLCP data structure

- Recall $\mathrm{LCP}[i]$ is the length of longest common prefix of i^{th} and $(i+1)^{th}$ lexicographically smallest suffixes.
- pLCP array is nothing but LCP array stored in text order.
- From Sadakane's result [3], pLCP can be stored in $2n + o(n)$ bits and queried in $\mathcal{O}(1)$ time.
- Similar to pLCP, one can also define pLCS which is LCS (Longest Common Suffix) stored in text order.

Generalizing pLCP: the pLCP_π family

- For an integer $\pi \neq 0$, define $\mathrm{LCP}_\pi[i]$ as the length of longest common prefix of i^{th} and $(i+\pi)^{th}$ lexicographically smallest suffixes.
- Formally $\mathrm{LCP}_\pi[i] = \mathrm{LCE}(\mathrm{SA}[i], \mathrm{SA}[i+\pi])$
- Naturally, $\mathrm{pLCP}_\pi[j]$ is nothing but LCP_π stored in text order.
- The same encoding by Sadakane still holds. Therefore, for each fixed π, the pLCP_π can be encoded in $\mathcal{O}(n)$ bits and $\mathcal{O}(1)$-time access.
- Analogously, we also have pLCS_π structures.

Generalizing pLCP: the pLCP_π family

- For an integer $\pi \neq 0$, define $\mathrm{LCP}_\pi[i]$ as the length of longest common prefix of i^{th} and $(i+\pi)^{th}$ lexicographically smallest suffixes.
- Formally $\mathrm{LCP}_\pi[i] = \mathrm{LCE}(\mathrm{SA}[i], \mathrm{SA}[i+\pi])$
- Naturally, $\mathrm{pLCP}_\pi[j]$ is nothing but LCP_π stored in text order.
- The same encoding by Sadakane still holds. Therefore, for each fixed π, the pLCP_π can be encoded in $\mathcal{O}(n)$ bits and $\mathcal{O}(1)$-time access.
- Analogously, we also have pLCS_π structures.

Generalizing pLCP: the pLCP_π family

- For an integer $\pi \neq 0$, define $\mathrm{LCP}_\pi[i]$ as the length of longest common prefix of i^{th} and $(i+\pi)^{th}$ lexicographically smallest suffixes.
- Formally $\mathrm{LCP}_\pi[i] = \mathrm{LCE}(\mathrm{SA}[i], \mathrm{SA}[i+\pi])$
- Naturally, $\mathrm{pLCP}_\pi[j]$ is nothing but LCP_π stored in text order.
- The same encoding by Sadakane still holds. Therefore, for each fixed π, the pLCP_π can be encoded in $\mathcal{O}(n)$ bits and $\mathcal{O}(1)$-time access.
- Analogously, we also have pLCS_π structures.

Generalizing pLCP: the pLCP_π family

- For an integer $\pi \neq 0$, define $\mathrm{LCP}_\pi[i]$ as the length of longest common prefix of i^{th} and $(i+\pi)^{th}$ lexicographically smallest suffixes.
- Formally $\mathrm{LCP}_\pi[i] = \mathrm{LCE}(\mathrm{SA}[i], \mathrm{SA}[i+\pi])$
- Naturally, $\mathrm{pLCP}_\pi[j]$ is nothing but LCP_π stored in text order.
- The same encoding by Sadakane still holds. Therefore, for each fixed π, the pLCP_π can be encoded in $\mathcal{O}(n)$ bits and $\mathcal{O}(1)$-time access.
- Analogously, we also have pLCS_π structures.

Generalizing pLCP: the pLCP_π family

- For an integer $\pi \neq 0$, define $\mathrm{LCP}_\pi[i]$ as the length of longest common prefix of i^{th} and $(i+\pi)^{th}$ lexicographically smallest suffixes.
- Formally $\mathrm{LCP}_\pi[i] = \mathrm{LCE}(\mathrm{SA}[i], \mathrm{SA}[i+\pi])$
- Naturally, $\mathrm{pLCP}_\pi[j]$ is nothing but LCP_π stored in text order.
- The same encoding by Sadakane still holds. Therefore, for each fixed π, the pLCP_π can be encoded in $\mathcal{O}(n)$ bits and $\mathcal{O}(1)$-time access.
- Analogously, we also have pLCS_π structures.

Generalizing pLCP: the pLCP_π family

- For an integer $\pi \neq 0$, define $\mathrm{LCP}_\pi[i]$ as the length of longest common prefix of i^{th} and $(i+\pi)^{th}$ lexicographically smallest suffixes.
- Formally $\mathrm{LCP}_\pi[i] = \mathrm{LCE}(\mathrm{SA}[i], \mathrm{SA}[i+\pi])$
- Naturally, $\mathrm{pLCP}_\pi[j]$ is nothing but LCP_π stored in text order.
- The same encoding by Sadakane still holds. Therefore, for each fixed π, the pLCP_π can be encoded in $\mathcal{O}(n)$ bits and $\mathcal{O}(1)$-time access.
- Analogously, we also have pLCS_π structures.

Quick Remark

- A suffix of $\overleftarrow{\mathsf{T}}$ beginning at position j is the reverse of prefix of T ending at $(n - j + 1)$.
- Define $\mathrm{PA}[i]$ as the i^{th} lexicographically smallest reverse prefix of T.
- So, we focus on answering the equivalent PA query when asked for a $\mathrm{SA}_{\overleftarrow{\mathsf{T}}}[i]$ query.

Quick Remark

- A suffix of $\overleftarrow{\mathrm{T}}$ beginning at position j is the reverse of prefix of T ending at $(n - j + 1)$.
- Define $\mathrm{PA}[i]$ as the i^{th} lexicographically smallest reverse prefix of T.
- So, we focus on answering the equivalent PA query when asked for a $\mathrm{SA}_{\overleftarrow{\mathrm{T}}}[i]$ query.

Quick Remark

- A suffix of $\overleftarrow{\mathsf{T}}$ beginning at position j is the reverse of prefix of T ending at $(n - j + 1)$.
- Define $\mathrm{PA}[i]$ as the i^{th} lexicographically smallest reverse prefix of T.
- So, we focus on answering the equivalent PA query when asked for a $\mathrm{SA}_{\overleftarrow{\mathsf{T}}}[i]$ query.

Algorithm: High-level description

- Sample leaves of prefix tree (PT) of T in $\log^* n$ levels. (i.e., PA_h arrays for $h \in [\log^* n]$ where PA_h has $\frac{n}{\Delta_h}$ entries for Δ_h being the sampling factor).

- Levels are chosen in such a way that any leaf i is sampled in some level $f \in [\log^* n]$.

- In Level 1, the PA_1 array entries are stored explicitly. For the remaining levels, we store an encoded format of an offset (call it θ_i i.e., $\mathrm{PA}_h[\frac{i}{\Delta_h}] = \theta_i$).

Algorithm: High-level description

- Sample leaves of prefix tree (PT) of T in $\log^* n$ levels. (i.e., PA_h arrays for $h \in [\log^* n]$ where PA_h has $\frac{n}{\Delta_h}$ entries for Δ_h being the sampling factor).
- Levels are chosen in such a way that any leaf i is sampled in some level $f \in [\log^* n]$.
- In Level 1, the PA_1 array entries are stored explicitly. For the remaining levels, we store an encoded format of an offset (call it θ_i i.e., $\mathrm{PA}_h[\frac{i}{\Delta_h}] = \theta_i$).

Algorithm: High-level description

- Sample leaves of prefix tree (PT) of T in $\log^* n$ levels. (i.e., PA_h arrays for $h \in [\log^* n]$ where PA_h has $\frac{n}{\Delta_h}$ entries for Δ_h being the sampling factor).

- Levels are chosen in such a way that any leaf i is sampled in some level $f \in [\log^* n]$.

- In Level 1, the PA_1 array entries are stored explicitly. For the remaining levels, we store an encoded format of an offset (call it θ_i i.e., $\mathrm{PA}_h[\frac{i}{\Delta_h}] = \theta_i$).

Algorithm: High-level description

- Sample leaves of prefix tree (PT) of T in $\log^* n$ levels. (i.e., PA_h arrays for $h \in [\log^* n]$ where PA_h has $\frac{n}{\Delta_h}$ entries for Δ_h being the sampling factor).

- Levels are chosen in such a way that any leaf i is sampled in some level $f \in [\log^* n]$.

- In Level 1, the PA_1 array entries are stored explicitly. For the remaining levels, we store an encoded format of an offset (call it θ_i i.e., $\mathrm{PA}_h[\frac{i}{\Delta_h}] = \theta_i$).

Algorithm: High-level description

- Sample leaves of prefix tree (PT) of T in $\log^* n$ levels. (i.e., PA_h arrays for $h \in [\log^* n]$ where PA_h has $\frac{n}{\Delta_h}$ entries for Δ_h being the sampling factor).

- Levels are chosen in such a way that any leaf i is sampled in some level $f \in [\log^* n]$.

- In Level 1, the PA_1 array entries are stored explicitly. For the remaining levels, we store an encoded format of an offset (call it θ_i i.e., $\mathrm{PA}_h[\frac{i}{\Delta_h}] = \theta_i$).

Algorithm: High-level description

- Sample leaves of prefix tree (PT) of T in $\log^* n$ levels. (i.e., PA_h arrays for $h \in [\log^* n]$ where PA_h has $\frac{n}{\Delta_h}$ entries for Δ_h being the sampling factor).

- Levels are chosen in such a way that any leaf i is sampled in some level $f \in [\log^* n]$.

- In Level 1, the PA_1 array entries are stored explicitly. For the remaining levels, we store an encoded format of an offset (call it θ_i i.e., $\mathrm{PA}_h[\frac{i}{\Delta_h}] = \theta_i$).

Algorithm: High-level description (Contd.)

- To answer $\mathrm{PA}[i]$ (say leaf i is sampled in level f), we recursively find the closest leaves which are sampled in the previous levels (This will end up in finding Level 1 entries which can be retrieved in $\mathcal{O}(1)$ time).
- From Level 1, we decode entries of Level 2, 3, .., until f, finally retrieving $\mathrm{PA}[i]$.
- Overall time taken is at most $\log^*(n)\times$ (Time to decode a leaf of one level from its closest leaves sampled in the previous level).

Algorithm: High-level description (Contd.)

- To answer $\mathrm{PA}[i]$ (say leaf i is sampled in level f), we recursively find the closest leaves which are sampled in the previous levels (This will end up in finding Level 1 entries which can be retrieved in $\mathcal{O}(1)$ time).
- From Level 1, we decode entries of Level 2, 3, .., until f, finally retrieving $\mathrm{PA}[i]$.
- Overall time taken is at most $\log^*(n)\times$ (Time to decode a leaf of one level from its closest leaves sampled in the previous level).

Algorithm: High-level description (Contd.)

- To answer $\mathrm{PA}[i]$ (say leaf i is sampled in level f), we recursively find the closest leaves which are sampled in the previous levels (This will end up in finding Level 1 entries which can be retrieved in $\mathcal{O}(1)$ time).
- From Level 1, we decode entries of Level 2, 3, .., until f, finally retrieving $\mathrm{PA}[i]$.
- Overall time taken is at most $\log^*(n)\times$ (Time to decode a leaf of one level from its closest leaves sampled in the previous level).

Algorithm: High-level description (Contd.)

- To answer $\mathrm{PA}[i]$ (say leaf i is sampled in level f), we recursively find the closest leaves which are sampled in the previous levels (This will end up in finding Level 1 entries which can be retrieved in $\mathcal{O}(1)$ time).

- From Level 1, we decode entries of Level 2, 3, .., until f, finally retrieving $\mathrm{PA}[i]$.

- Overall time taken is at most $\log^*(n)\times$ (Time to decode a leaf of one level from its closest leaves sampled in the previous level).

Maintained Data structures

- **Succinct topologies** of the suffix tree of T and the prefix tree of T.
- **Sampling in levels.** For $h \in [1, \log^* n]$, define $\Delta_h = 2^{\lceil 2 \log^{(h+1)} n \rceil}$, i.e. the next power of 2 above $(\log^{(h)} n)^2$. Also $\Delta_0 = n + 1$.
- **Level 1:** if $\Delta_1 \mid i$, store $\mathrm{PA}_1[i] = \mathrm{PA}[i]$ explicitly.
- **Higher levels:** if $\Delta_h \mid i$ but $\Delta_{h-1} \nmid i$, store only a small offset θ_i $(\leq \Delta_{h-1})$ in $\mathrm{PA}_h[i]$.
- **Auxiliary sampled pLCP_π and pLCS_π structures:** For each $h \in [1, \log^* n]$, we maintain $2\frac{\Delta_{h-1}}{\Delta_h}$ pLCP_π and pLCS_π structures (each structure maintaining only $\frac{n}{\Delta_{h-1}}$ entries), which will be helpful during the recursive decode.
- Above structures altogether take up $\mathcal{O}(n)$ bits additional space.

Maintained Data structures

- **Succinct topologies** of the suffix tree of T and the prefix tree of T.
- **Sampling in levels.** For $h \in [1, \log^* n]$, define $\Delta_h = 2^{\lceil 2 \log^{(h+1)} n \rceil}$. i.e. the next power of 2 above $(\log^{(h)} n)^2$. Also $\Delta_0 = n + 1$.
- **Level 1:** if $\Delta_1 \mid i$, store $\mathrm{PA}_1[i] = \mathrm{PA}[i]$ explicitly.
- **Higher levels:** if $\Delta_h \mid i$ but $\Delta_{h-1} \nmid i$, store only a small offset θ_i $(\leq \Delta_{h-1})$ in $\mathrm{PA}_h[i]$.
- **Auxiliary sampled pLCP_π and pLCS_π structures:** For each $h \in [1, \log^* n]$, we maintain $2\frac{\Delta_{h-1}}{\Delta_h}$ pLCP_π and pLCS_π structures (each structure maintaining only $\frac{n}{\Delta_{h-1}}$ entries), which will be helpful during the recursive decode.
- Above structures altogether take up $\mathcal{O}(n)$ bits additional space.

Maintained Data structures

- **Succinct topologies** of the suffix tree of T and the prefix tree of T.
- **Sampling in levels.** For $h \in [1, \log^* n]$, define $\Delta_h = 2^{\lceil 2 \log^{(h+1)} n \rceil}$. i.e. the next power of 2 above $(\log^{(h)} n)^2$. Also $\Delta_0 = n + 1$.
- **Level 1:** if $\Delta_1 \mid i$, store $\mathrm{PA}_1[i] = \mathrm{PA}[i]$ explicitly.
- **Higher levels:** if $\Delta_h \mid i$ but $\Delta_{h-1} \nmid i$, store only a small offset θ_i $(\leq \Delta_{h-1})$ in $\mathrm{PA}_h[i]$.
- **Auxiliary sampled pLCP_π and pLCS_π structures:** For each $h \in [1, \log^* n]$, we maintain $2\frac{\Delta_{h-1}}{\Delta_h}$ pLCP_π and pLCS_π structures (each structure maintaining only $\frac{n}{\Delta_{h-1}}$ entries), which will be helpful during the recursive decode.
- Above structures altogether take up $\mathcal{O}(n)$ bits additional space.

Maintained Data structures

- **Succinct topologies** of the suffix tree of T and the prefix tree of T.
- **Sampling in levels.** For $h \in [1, \log^* n]$, define $\Delta_h = 2^{\lceil 2 \log^{(h+1)} n \rceil}$. i.e. the next power of 2 above $(\log^{(h)} n)^2$. Also $\Delta_0 = n + 1$.
- **Level 1:** if $\Delta_1 \mid i$, store $\mathrm{PA}_1[i] = \mathrm{PA}[i]$ explicitly.
- **Higher levels:** if $\Delta_h \mid i$ but $\Delta_{h-1} \nmid i$, store only a small offset θ_i $(\leq \Delta_{h-1})$ in $\mathrm{PA}_h[i]$.
- **Auxiliary sampled pLCP_π and pLCS_π structures:** For each $h \in [1, \log^* n]$, we maintain $2\frac{\Delta_{h-1}}{\Delta_h}$ pLCP_π and pLCS_π structures (each structure maintaining only $\frac{n}{\Delta_{h-1}}$ entries), which will be helpful during the recursive decode.
- Above structures altogether take up $\mathcal{O}(n)$ bits additional space.

Maintained Data structures

- **Succinct topologies** of the suffix tree of T and the prefix tree of T.
- **Sampling in levels.** For $h \in [1, \log^* n]$, define $\Delta_h = 2^{\lceil 2 \log^{(h+1)} n \rceil}$. i.e. the next power of 2 above $(\log^{(h)} n)^2$. Also $\Delta_0 = n + 1$.
- **Level 1:** if $\Delta_1 \mid i$, store $\mathrm{PA}_1[i] = \mathrm{PA}[i]$ explicitly.
- **Higher levels:** if $\Delta_h \mid i$ but $\Delta_{h-1} \nmid i$, store only a small offset θ_i $(\leq \Delta_{h-1})$ in $\mathrm{PA}_h[i]$.
- **Auxiliary sampled pLCP_π and pLCS_π structures:** For each $h \in [1, \log^* n]$, we maintain $2\frac{\Delta_{h-1}}{\Delta_h}$ pLCP_π and pLCS_π structures (each structure maintaining only $\frac{n}{\Delta_{h-1}}$ entries), which will be helpful during the recursive decode.
- Above structures altogether take up $\mathcal{O}(n)$ bits additional space.

Maintained Data structures

- **Succinct topologies** of the suffix tree of T and the prefix tree of T.
- **Sampling in levels.** For $h \in [1, \log^* n]$, define $\Delta_h = 2^{\lceil 2 \log^{(h+1)} n \rceil}$. i.e. the next power of 2 above $(\log^{(h)} n)^2$. Also $\Delta_0 = n + 1$.
- **Level 1:** if $\Delta_1 \mid i$, store $\mathrm{PA}_1[i] = \mathrm{PA}[i]$ explicitly.
- **Higher levels:** if $\Delta_h \mid i$ but $\Delta_{h-1} \nmid i$, store only a small offset θ_i $(\leq \Delta_{h-1})$ in $\mathrm{PA}_h[i]$.
- **Auxiliary sampled pLCP_π and pLCS_π structures:** For each $h \in [1, \log^* n]$, we maintain $2\frac{\Delta_{h-1}}{\Delta_h}$ pLCP_π and pLCS_π structures (each structure maintaining only $\frac{n}{\Delta_{h-1}}$ entries), which will be helpful during the recursive decode.
- Above structures altogether take up $\mathcal{O}(n)$ bits additional space.

Maintained Data structures

- **Succinct topologies** of the suffix tree of T and the prefix tree of T.
- **Sampling in levels.** For $h \in [1, \log^* n]$, define $\Delta_h = 2^{\lceil 2 \log^{(h+1)} n \rceil}$. i.e. the next power of 2 above $(\log^{(h)} n)^2$. Also $\Delta_0 = n + 1$.
- **Level 1:** if $\Delta_1 \mid i$, store $\mathrm{PA}_1[i] = \mathrm{PA}[i]$ explicitly.
- **Higher levels:** if $\Delta_h \mid i$ but $\Delta_{h-1} \nmid i$, store only a small offset θ_i $(\leq \Delta_{h-1})$ in $\mathrm{PA}_h[i]$.
- **Auxiliary sampled pLCP_π and pLCS_π structures:** For each $h \in [1, \log^* n]$, we maintain $2\frac{\Delta_{h-1}}{\Delta_h}$ pLCP_π and pLCS_π structures (each structure maintaining only $\frac{n}{\Delta_{h-1}}$ entries), which will be helpful during the recursive decode.
- Above structures altogether take up $\mathcal{O}(n)$ bits additional space.

Decoding a level-2 node from level-1 values (Contd.)

- Say we want to decode $\mathrm{PA}[i]$ where i^{th} leaf is sampled in level-2 (i.e., $\Delta_1 \nmid i$, but $\Delta_1 \mid i$).
- Find the closest level-1 sampled leaves to left and right of i (i.e., $i' = \Delta_1 \lfloor \frac{i}{\Delta_1} \rfloor$ and $i'' = \Delta_1 \lceil \frac{i}{\Delta_1} \rceil$).
- Retrieve $\mathrm{PA}[i']$ and $\mathrm{PA}[i'']$ (as they are explicitly stored in PA_1 array).

Decoding a level-2 node from level-1 values (Contd.)

- Say we want to decode $\mathrm{PA}[i]$ where i^{th} leaf is sampled in level-2 (i.e., $\Delta_1 \nmid i$, but $\Delta_1 \mid i$).
- Find the closest level-1 sampled leaves to left and right of i (i.e., $i' = \Delta_1 \lfloor \frac{i}{\Delta_1} \rfloor$ and $i'' = \Delta_1 \lceil \frac{i}{\Delta_1} \rceil$).
- Retrieve $\mathrm{PA}[i']$ and $\mathrm{PA}[i'']$ (as they are explicitly stored in PA_1 array).

Decoding a level-2 node from level-1 values (Contd.)

- Say we want to decode $\mathrm{PA}[i]$ where i^{th} leaf is sampled in level-2 (i.e., $\Delta_1 \nmid i$, but $\Delta_1 \mid i$).
- Find the closest level-1 sampled leaves to left and right of i (i.e., $i' = \Delta_1 \lfloor \frac{i}{\Delta_1} \rfloor$ and $i'' = \Delta_1 \lceil \frac{i}{\Delta_1} \rceil$).
- Retrieve $\mathrm{PA}[i']$ and $\mathrm{PA}[i'']$ (as they are explicitly stored in PA_1 array).

Decoding a level-2 node from level-1 values (Contd.)

- Say we want to decode $\mathrm{PA}[i]$ where i^{th} leaf is sampled in level-2 (i.e., $\Delta_1 \nmid i$, but $\Delta_1 \mid i$).
- Find the closest level-1 sampled leaves to left and right of i (i.e., $i' = \Delta_1 \lfloor \frac{i}{\Delta_1} \rfloor$ and $i'' = \Delta_1 \lceil \frac{i}{\Delta_1} \rceil$).
- Retrieve $\mathrm{PA}[i']$ and $\mathrm{PA}[i'']$ (as they are explicitly stored in PA_1 array).

Decoding a level-2 node from level-1 values (Contd.)

- Say we want to decode $\mathrm{PA}[i]$ where i^{th} leaf is sampled in level-2 (i.e., $\Delta_1 \nmid i$, but $\Delta_1 \mid i$).
- Find the closest level-1 sampled leaves to left and right of i (i.e., $i' = \Delta_1 \lfloor \frac{i}{\Delta_1} \rfloor$ and $i'' = \Delta_1 \lceil \frac{i}{\Delta_1} \rceil$).
- Retrieve $\mathrm{PA}[i']$ and $\mathrm{PA}[i'']$ (as they are explicitly stored in PA_1 array).

Decoding a level-2 node from level-1 values (Contd.)

- Say we want to decode $\mathrm{PA}[i]$ where i^{th} leaf is sampled in level-2 (i.e., $\Delta_1 \nmid i$, but $\Delta_1 \mid i$).
- Find the closest level-1 sampled leaves to left and right of i (i.e., $i' = \Delta_1 \lfloor \frac{i}{\Delta_1} \rfloor$ and $i'' = \Delta_1 \lceil \frac{i}{\Delta_1} \rceil$).
- Retrieve $\mathrm{PA}[i']$ and $\mathrm{PA}[i'']$ (as they are explicitly stored in PA_1 array).

Decoding a level-2 node from level-1 values (Contd.)

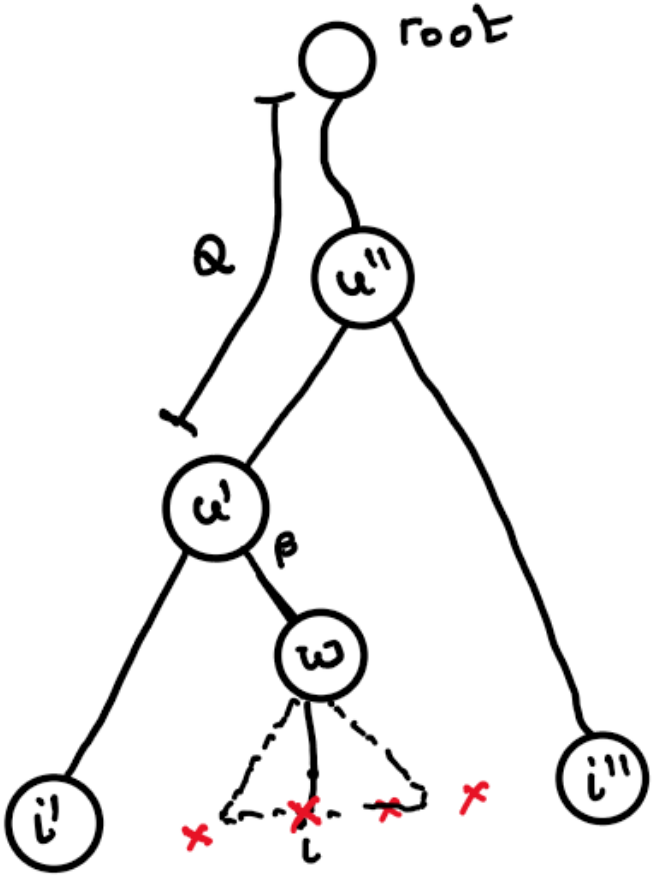


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}})$ $\in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

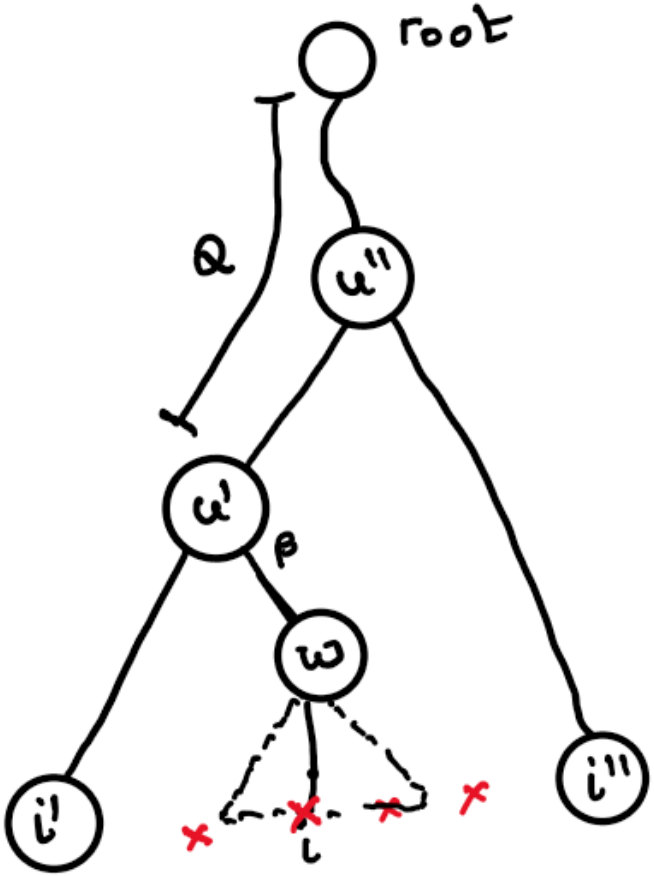


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_{π} structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

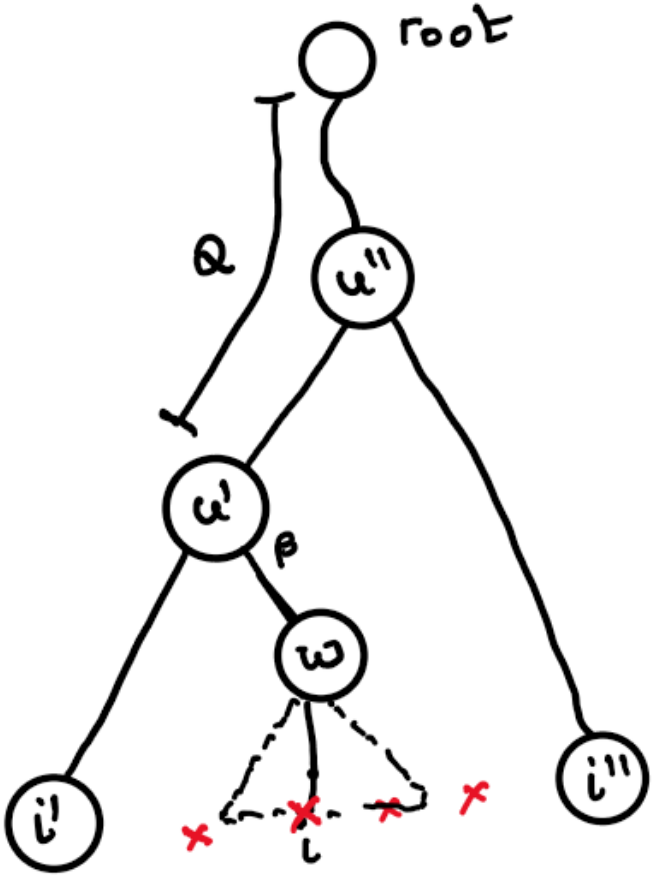


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

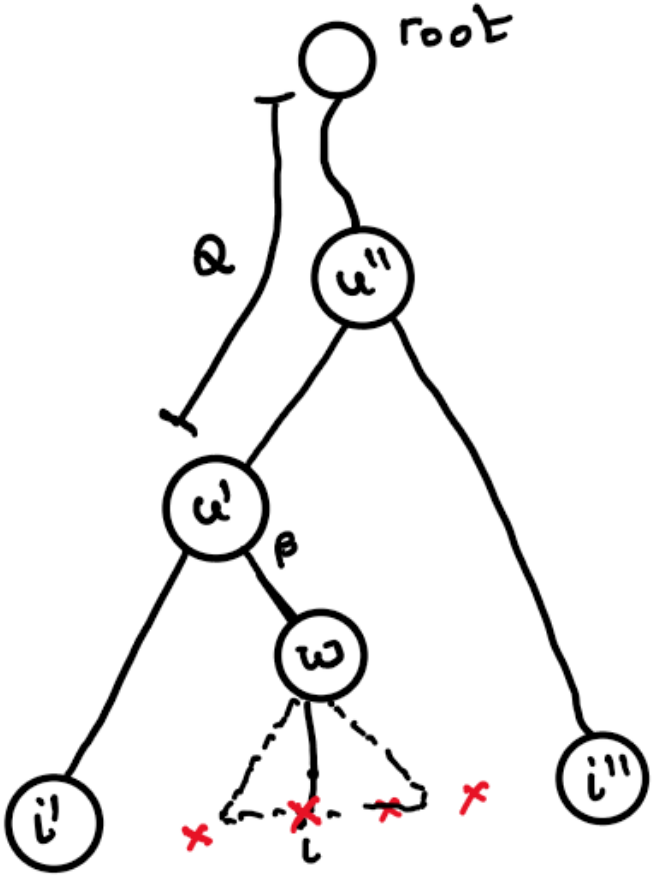


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

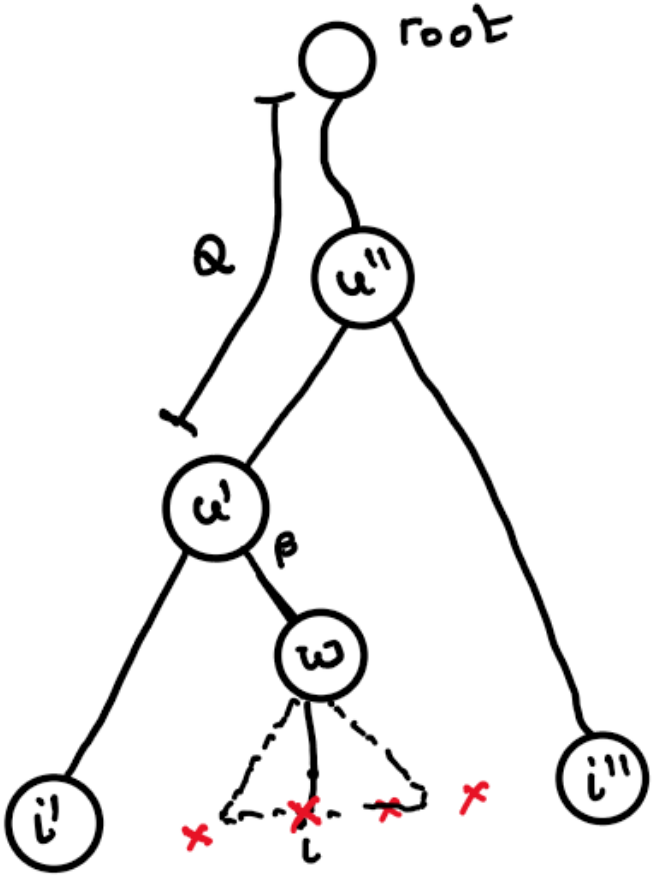


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

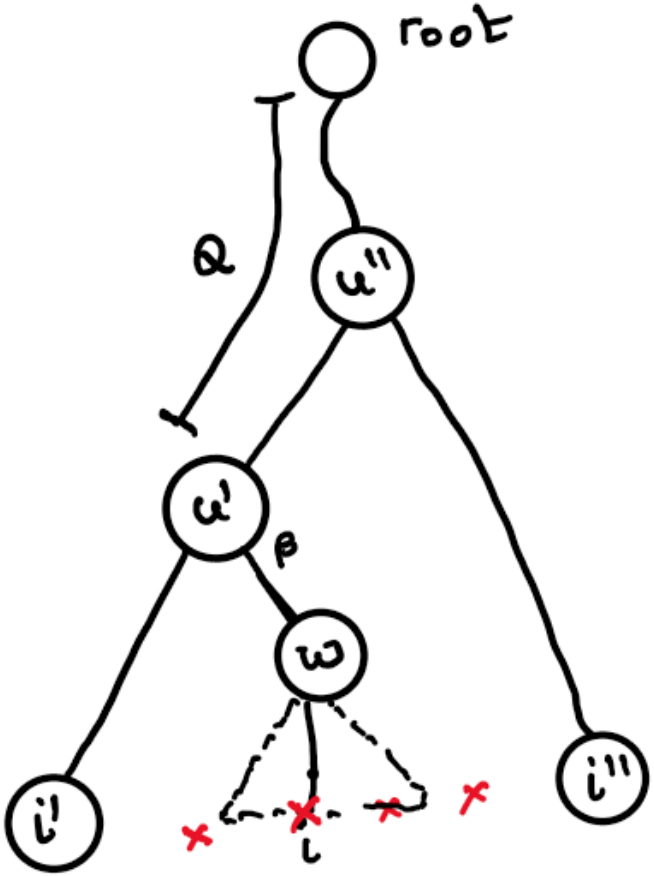


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

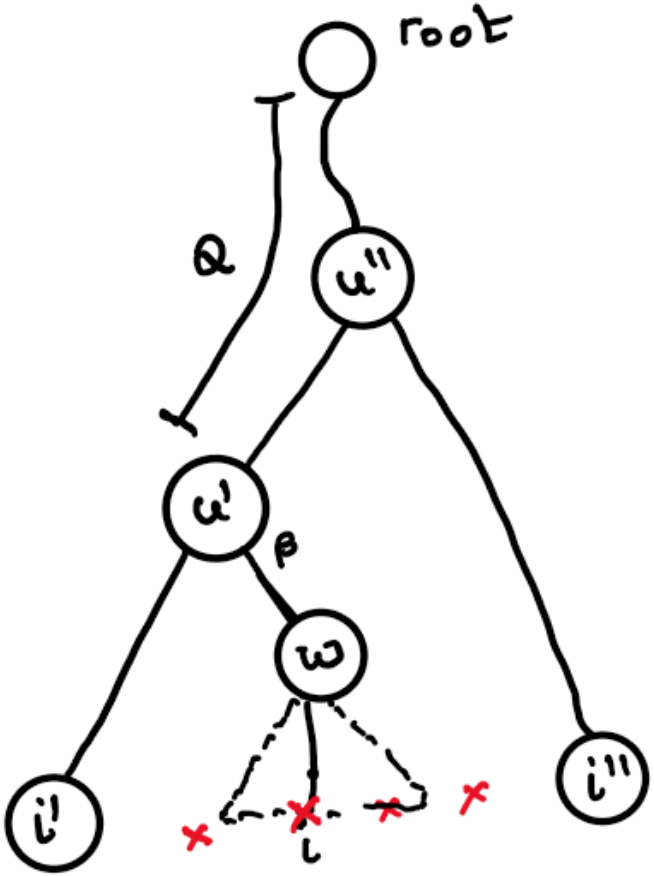


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

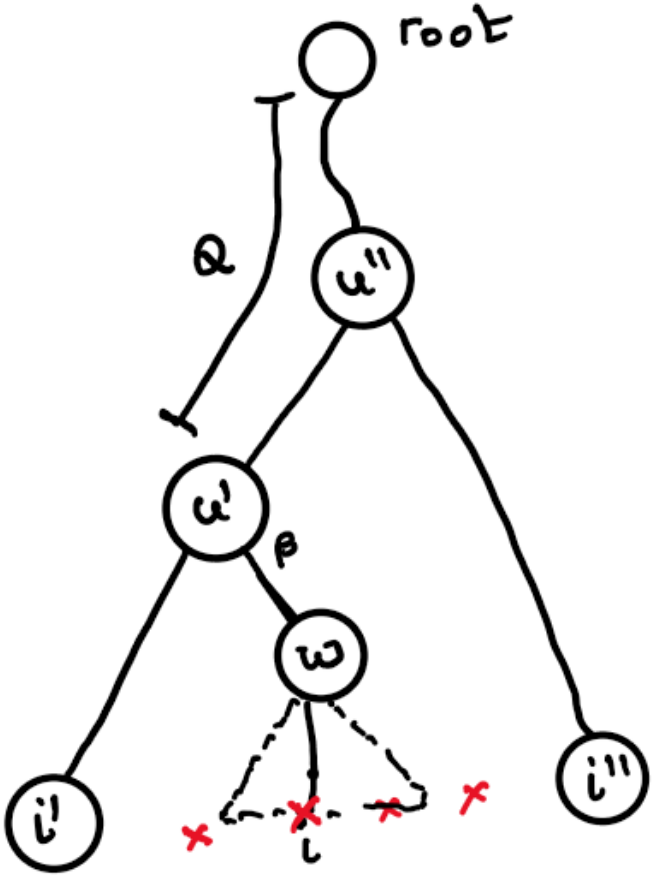


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

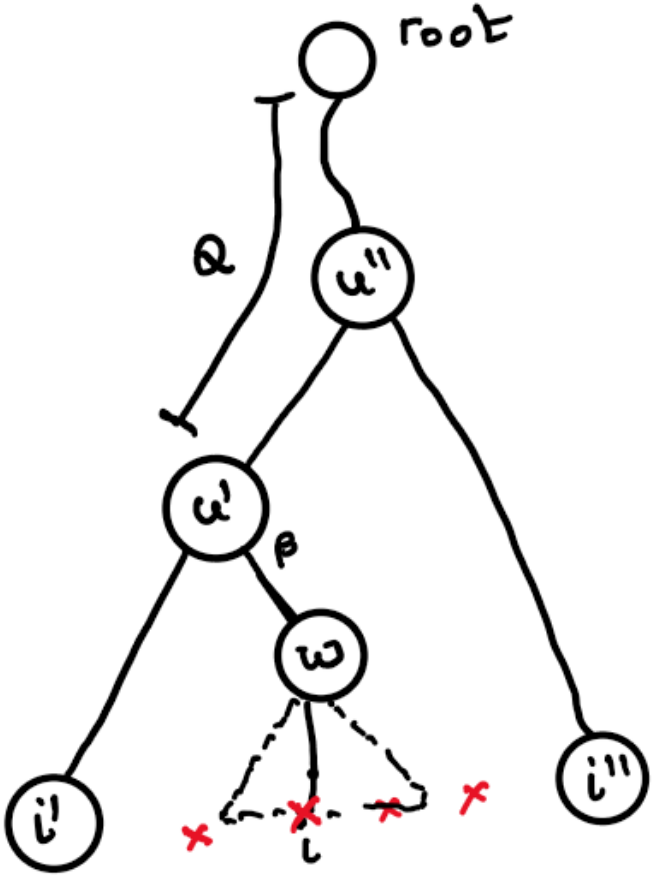


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

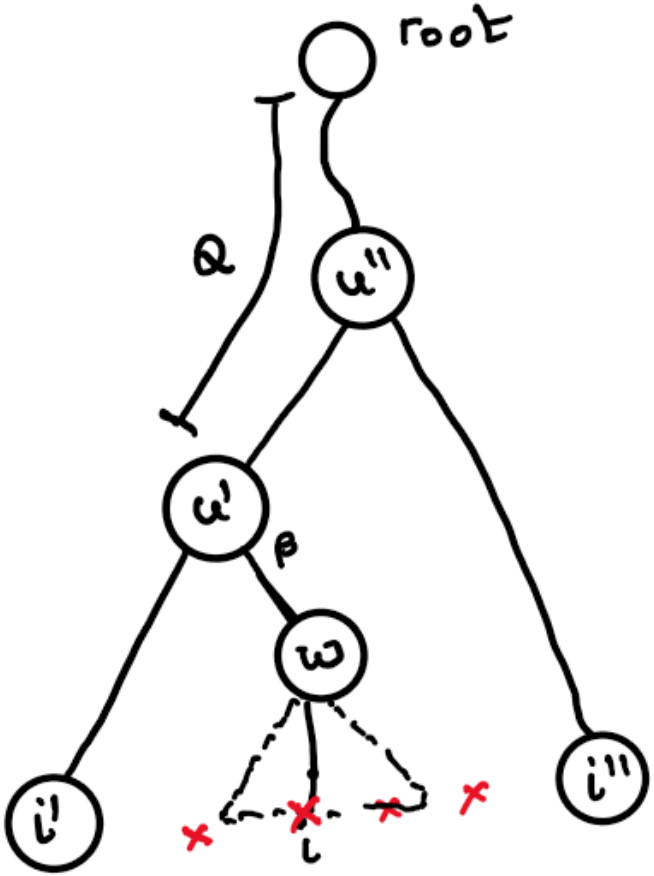


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

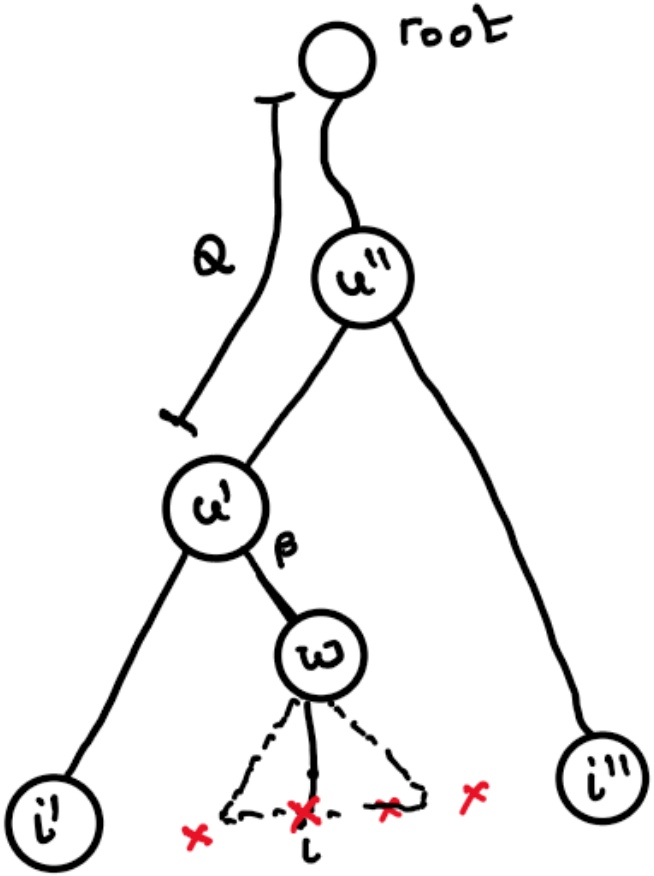


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

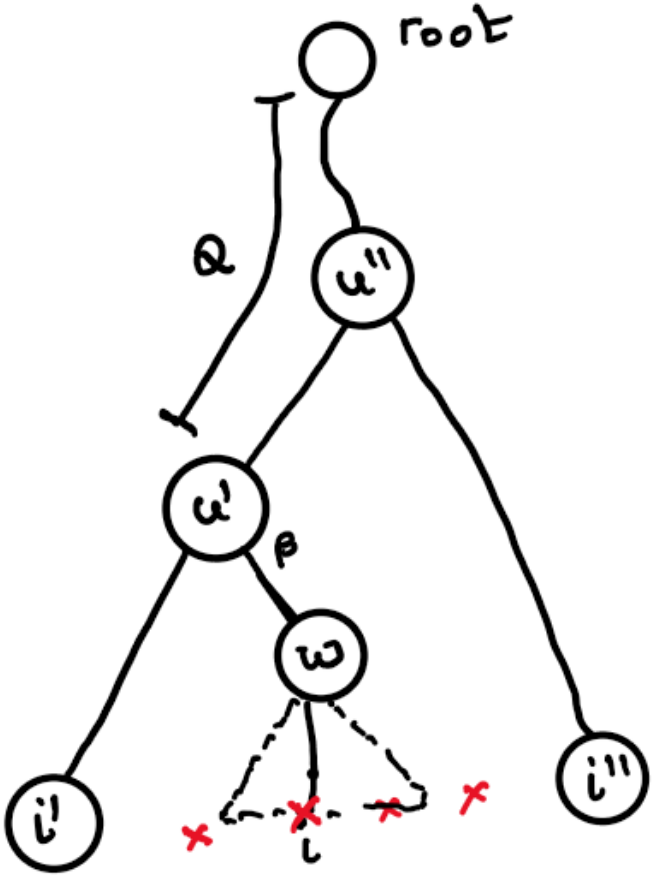


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

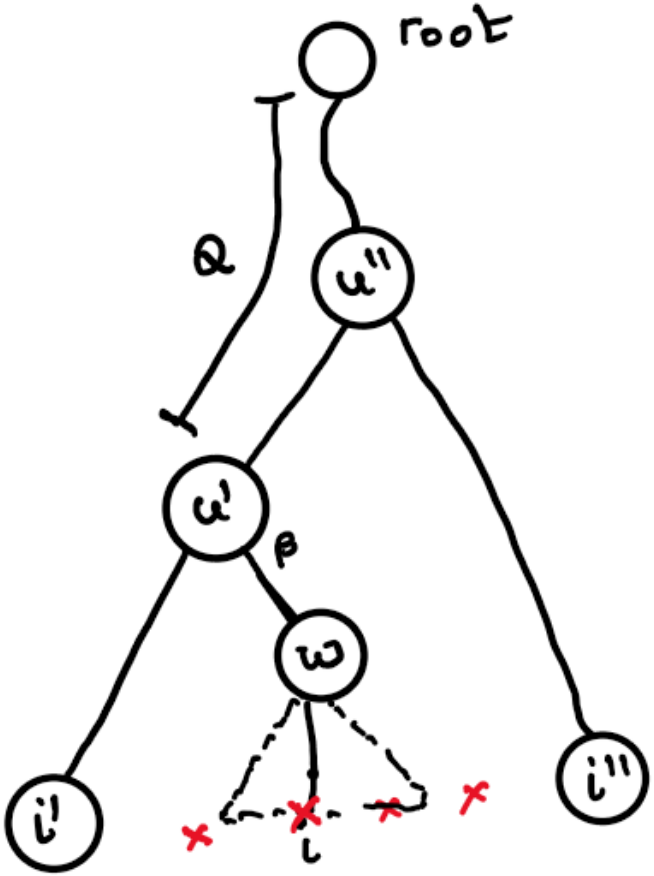


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_{π} structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

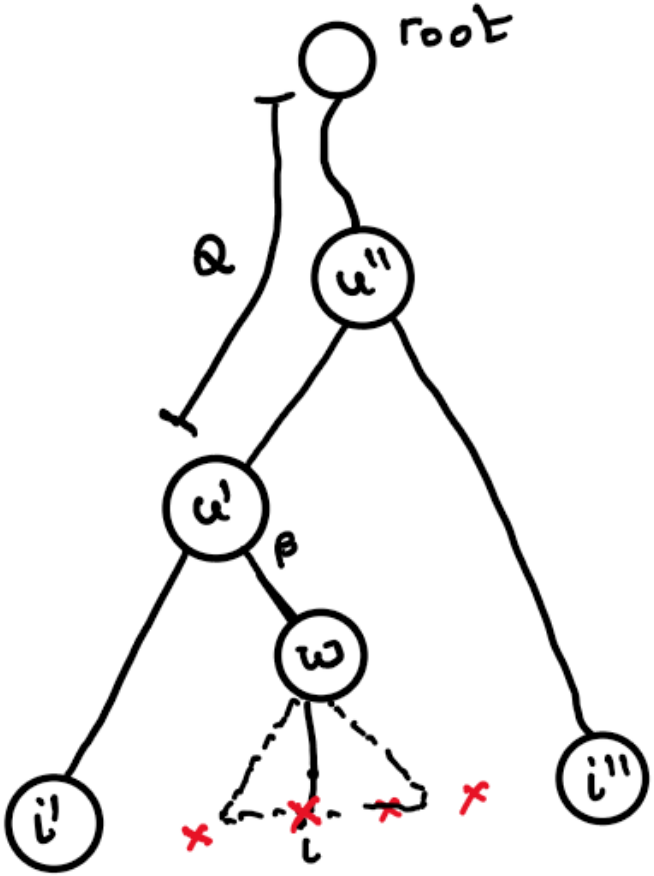


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}}) \in \mathcal{O}(t_{\mathrm{sa}})$ time.

Decoding a level-2 node from level-1 values (Contd.)

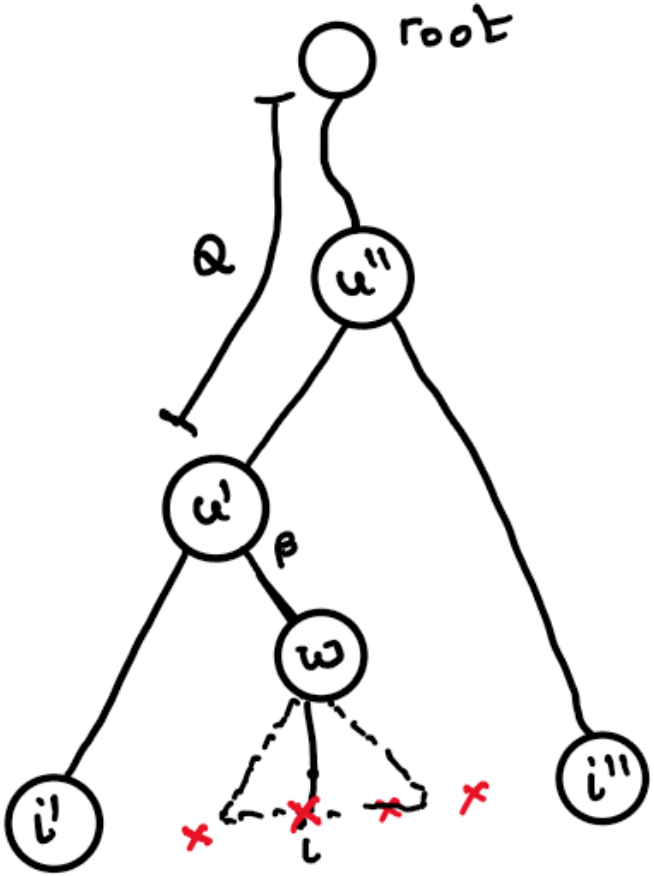


Figure: Prefix Tree of T

1. Find $\overleftarrow{\mathrm{Q}}$ (using $\mathrm{PA}[i']$ and pLCP_π structures) - $\mathcal{O}(1)$ time
2. Find the suffix range of $\overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{sa}} + \log n)$ time
3. Find β (using a range quantile query over BWT of T) - t_{quantile} time
4. Find the suffix range of $\beta \circ \overleftarrow{\mathrm{Q}}$ - $\mathcal{O}(t_{\mathrm{wt}})$ time
5. Compute $\mathrm{PA}[i]$ using the offset θ_i - $\mathcal{O}(t_{\mathrm{sa}})$ time
6. Total - $\mathcal{O}(t_{\mathrm{sa}} + \log n + t_{\mathrm{quantile}} + t_{\mathrm{wt}})$ $\in \mathcal{O}(t_{\mathrm{sa}})$ time.

Concluding First approach

- Decoding a level-2 sampled leaf from level-1 takes $\mathcal{O}(t_{\text{sa}})$ time.
- Since every query index belongs to one of the $\log^* n$ levels, we repeat this inductively for at most $\log^* n$ iterations.
- Overall algorithm has a $\mathcal{O}(t_{\text{sa}} \cdot \log^*(n))$ runtime.

Theorem 1

SA queries on $\overleftarrow{\mathrm{T}}$ can be supported in time $t_{\text{sa}} \cdot \mathcal{O}(\log^* n)$ with an auxiliary structure of $\mathcal{O}(n)$ bits of space.

Concluding First approach

- Decoding a level-2 sampled leaf from level-1 takes $\mathcal{O}(t_{\mathrm{sa}})$ time.
- Since every query index belongs to one of the $\log^* n$ levels, we repeat this inductively for at most $\log^* n$ iterations.
- Overall algorithm has a $\mathcal{O}(t_{\mathrm{sa}} \cdot \log^*(n))$ runtime.

Theorem 1

SA queries on $\overleftarrow{\mathrm{T}}$ can be supported in time $t_{\mathrm{sa}} \cdot \mathcal{O}(\log^* n)$ with an auxiliary structure of $\mathcal{O}(n)$ bits of space.

Concluding First approach

- Decoding a level-2 sampled leaf from level-1 takes $\mathcal{O}(t_{\mathrm{sa}})$ time.
- Since every query index belongs to one of the $\log^* n$ levels, we repeat this inductively for at most $\log^* n$ iterations.
- Overall algorithm has a $\mathcal{O}(t_{\mathrm{sa}} \cdot \log^*(n))$ runtime.

Theorem 1

SA queries on $\overleftarrow{\mathrm{T}}$ can be supported in time $t_{\mathrm{sa}} \cdot \mathcal{O}(\log^* n)$ with an auxiliary structure of $\mathcal{O}(n)$ bits of space.

Concluding First approach

- Decoding a level-2 sampled leaf from level-1 takes $\mathcal{O}(t_{\mathrm{sa}})$ time.
- Since every query index belongs to one of the $\log^* n$ levels, we repeat this inductively for at most $\log^* n$ iterations.
- Overall algorithm has a $\mathcal{O}(t_{\mathrm{sa}} \cdot \log^*(n))$ runtime.

Theorem 1

SA queries on $\overleftarrow{\mathrm{T}}$ can be supported in time $t_{\mathrm{sa}} \cdot \mathcal{O}(\log^* n)$ with an auxiliary structure of $\mathcal{O}(n)$ bits of space.

Concluding First approach

- Decoding a level-2 sampled leaf from level-1 takes $\mathcal{O}(t_{\mathrm{sa}})$ time.
- Since every query index belongs to one of the $\log^* n$ levels, we repeat this inductively for at most $\log^* n$ iterations.
- Overall algorithm has a $\mathcal{O}(t_{\mathrm{sa}} \cdot \log^*(n))$ runtime.

Theorem 1

SA queries on $\overleftarrow{\mathrm{T}}$ can be supported in time $t_{\mathrm{sa}} \cdot \mathcal{O}(\log^* n)$ with an auxiliary structure of $\mathcal{O}(n)$ bits of space.

Concluding First approach

- Decoding a level-2 sampled leaf from level-1 takes $\mathcal{O}(t_{\mathrm{sa}})$ time.
- Since every query index belongs to one of the $\log^* n$ levels, we repeat this inductively for at most $\log^* n$ iterations.
- Overall algorithm has a $\mathcal{O}(t_{\mathrm{sa}} \cdot \log^*(n))$ runtime.

Theorem 1

SA queries on $\overleftarrow{\mathsf{T}}$ can be supported in time $t_{\mathrm{sa}} \cdot \mathcal{O}(\log^* n)$ with an auxiliary structure of $\mathcal{O}(n)$ bits of space.

Thank You!

Any Questions?

References


Roberto Grossi and Jeffrey Scott Vitter.
Compressed suffix arrays and suffix trees with applications to text indexing and string matching.
SIAM J. Comput., 35(2):378–407, 2005.


Paolo Ferragina and Giovanni Manzini.
Opportunistic data structures with applications.
In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 390–398. IEEE, 2000.


Kunihiko Sadakane.
Compressed suffix trees with full functionality.
Theory Comput. Syst., 41(4):589–607, 2007.