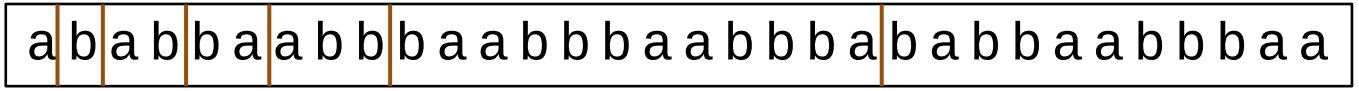
Block Trees

[Belazzougui et al., DCC 2015]
[Belazzougui et al., J. CSS, 2021]

Block Tree: Example

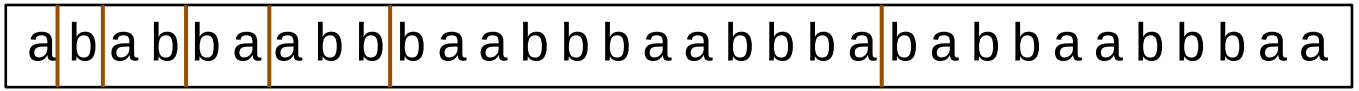
a b a b b a a b b b a a b b b a a b b b a b a b b a a b b b a a

Block Tree: Example



n=32, z=7

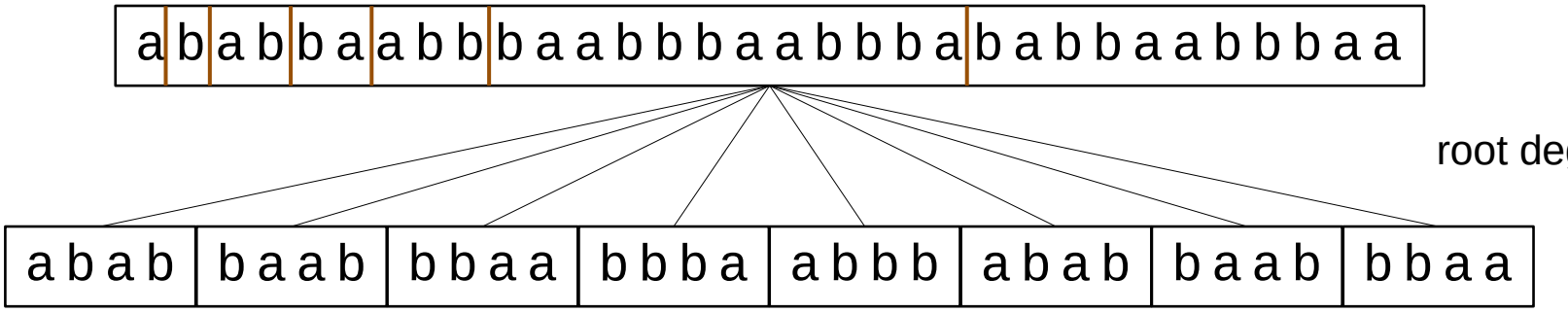
Block Tree: Example



n=32, z=7

root degree s := 8

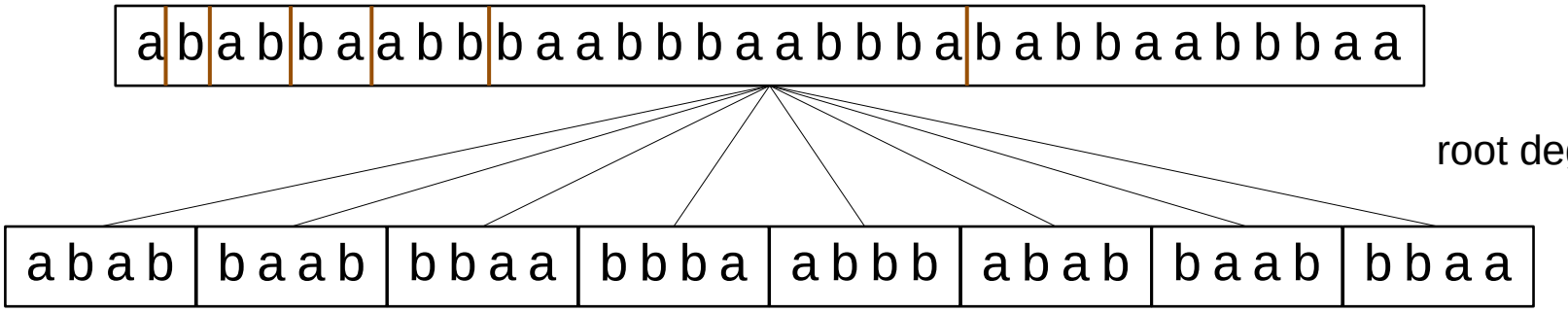
Block Tree: Example



n=32, z=7

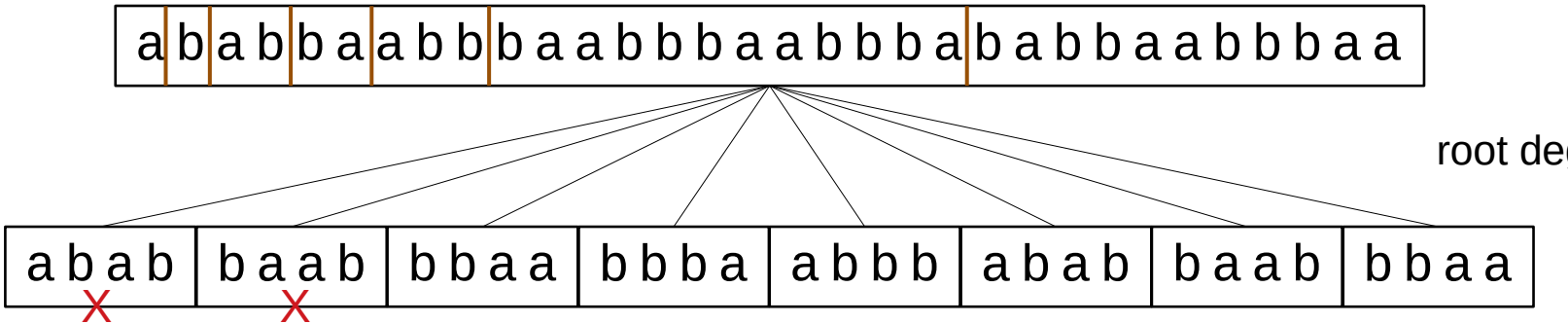
root degree s := 8

Block Tree: Example



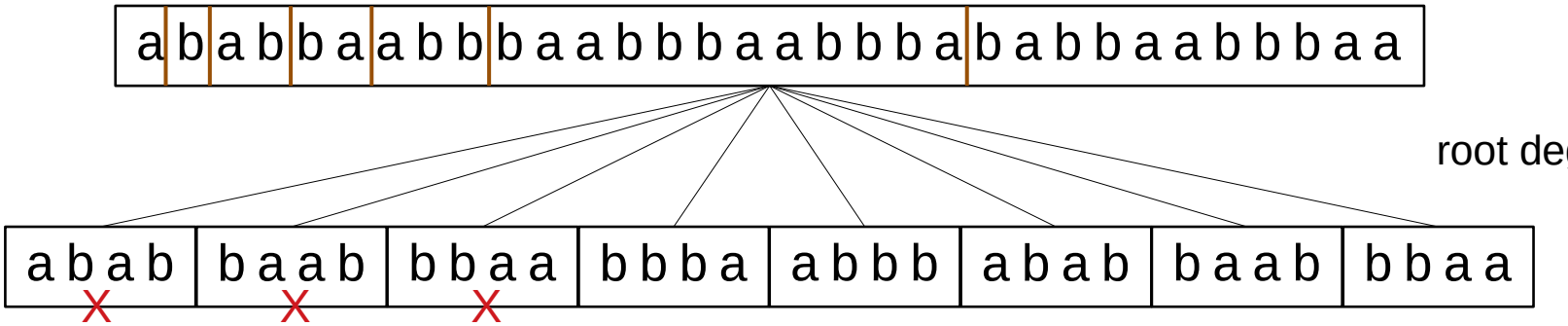
- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

Block Tree: Example



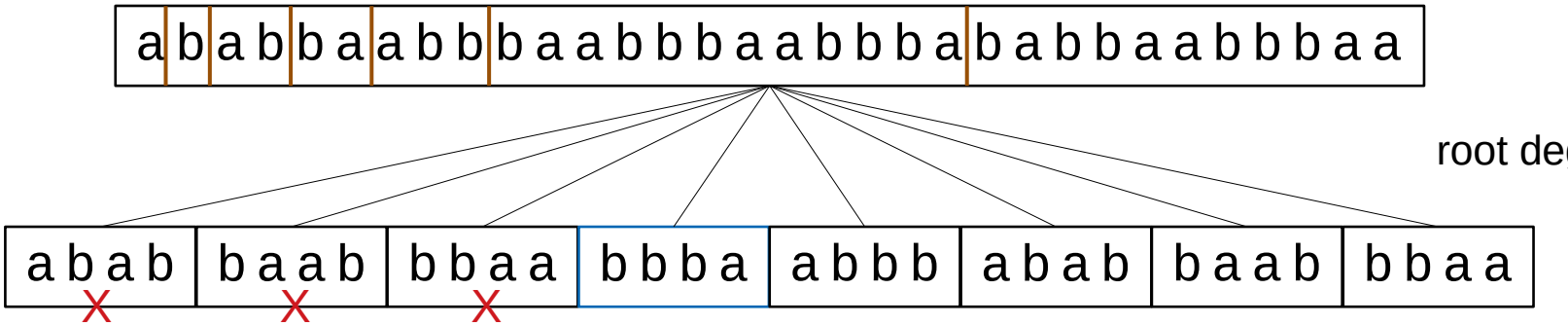
- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

Block Tree: Example



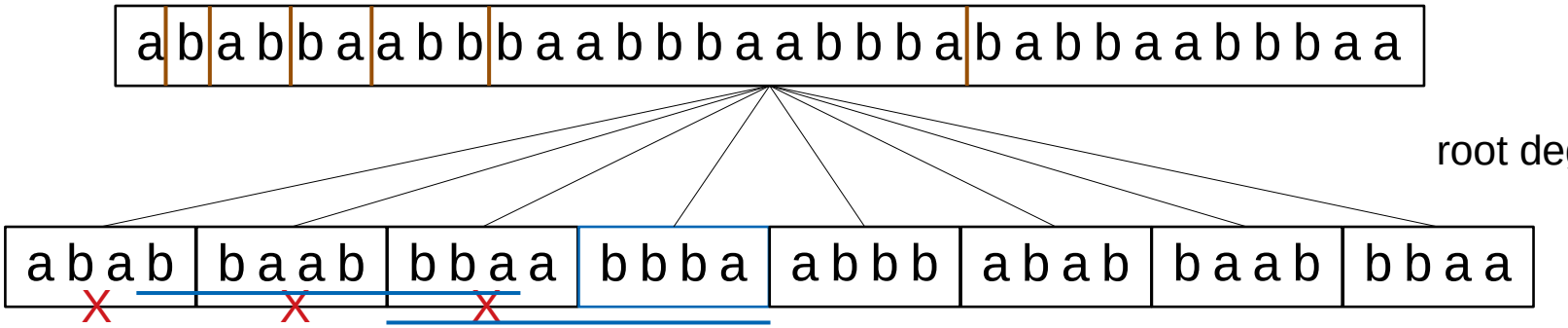
- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

Block Tree: Example



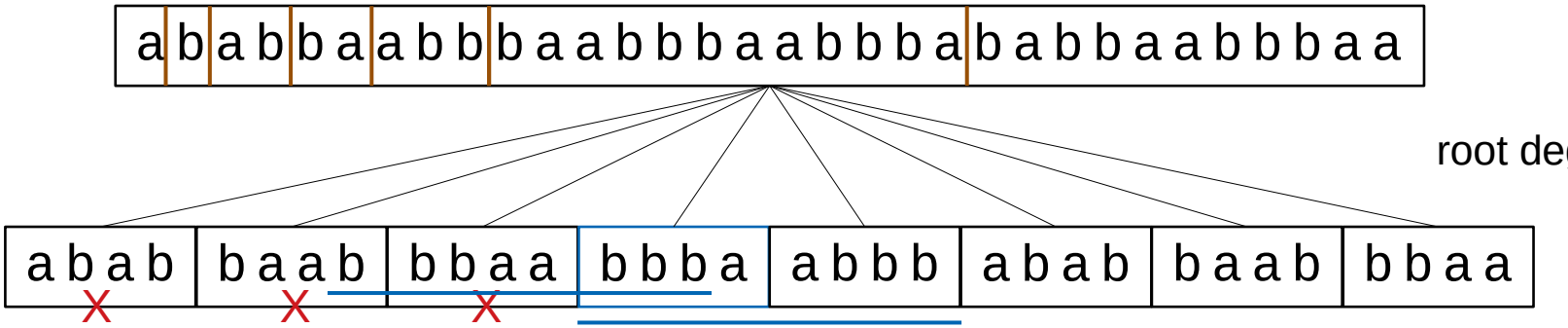
- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

Block Tree: Example



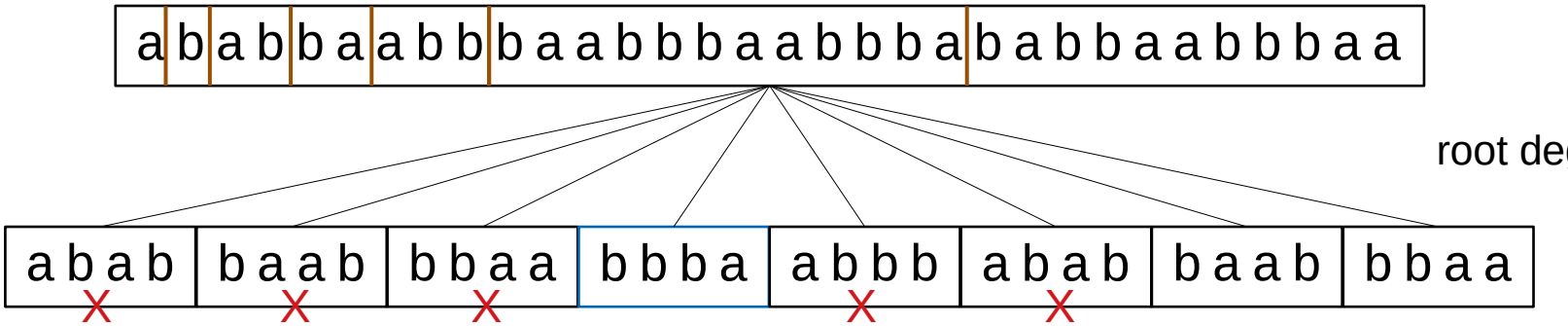
- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

Block Tree: Example



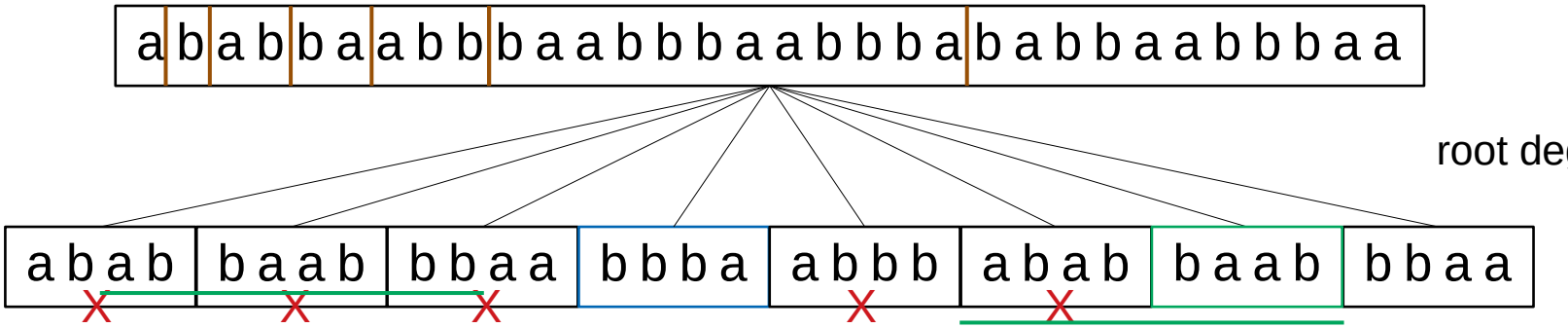
- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

Block Tree: Example



- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

Block Tree: Example

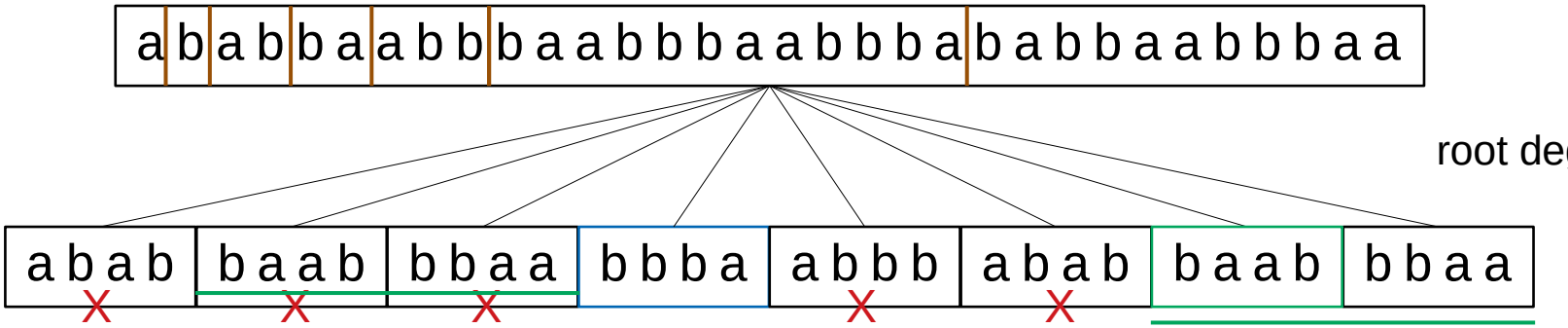


n=32, z=7

root degree s := 8

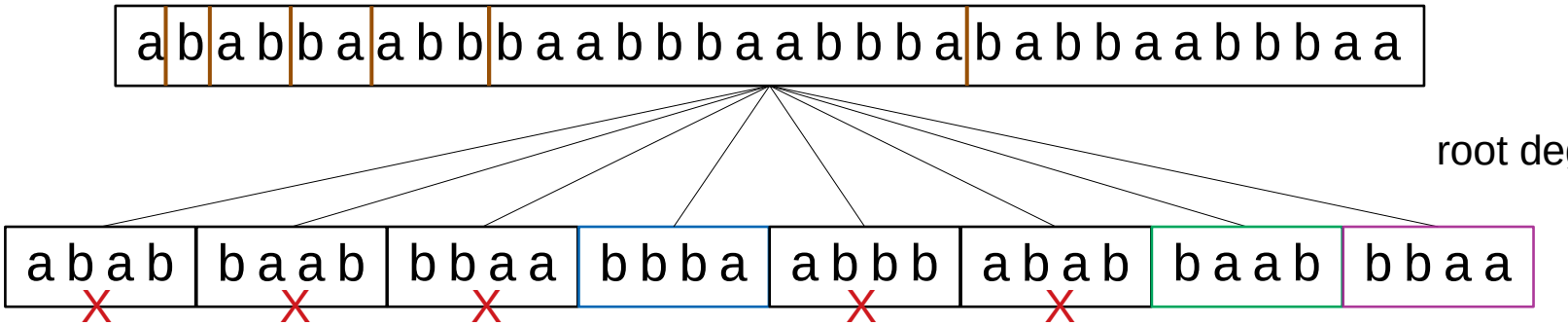
- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

Block Tree: Example



- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

Block Tree: Example

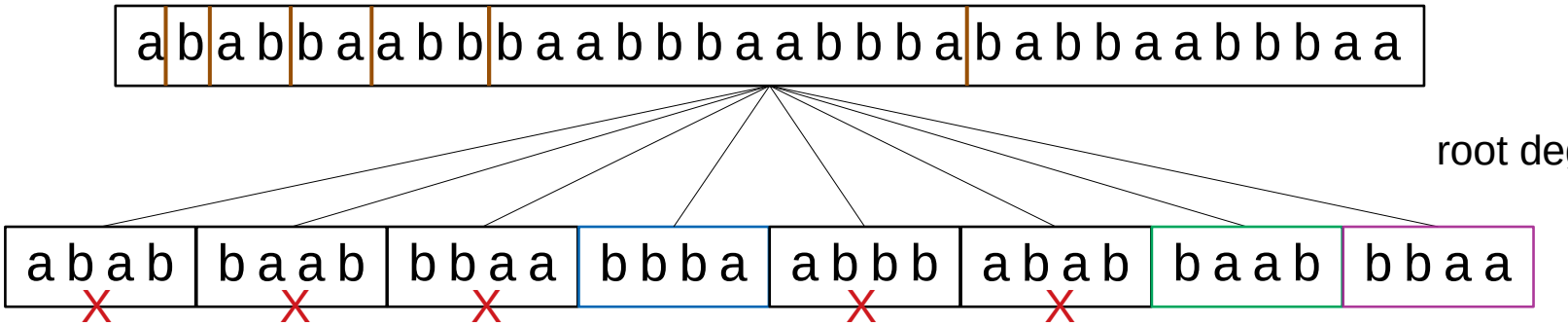


n=32, ***z***=7

root degree ***s*** := 8

- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S

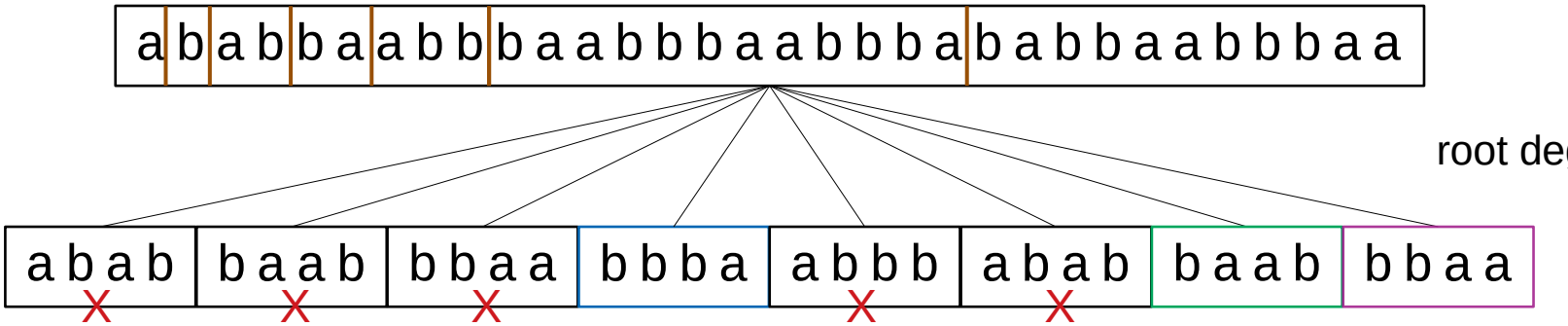
Block Tree: Example



n=32, z=7

root degree s := 8

Block Tree: Example

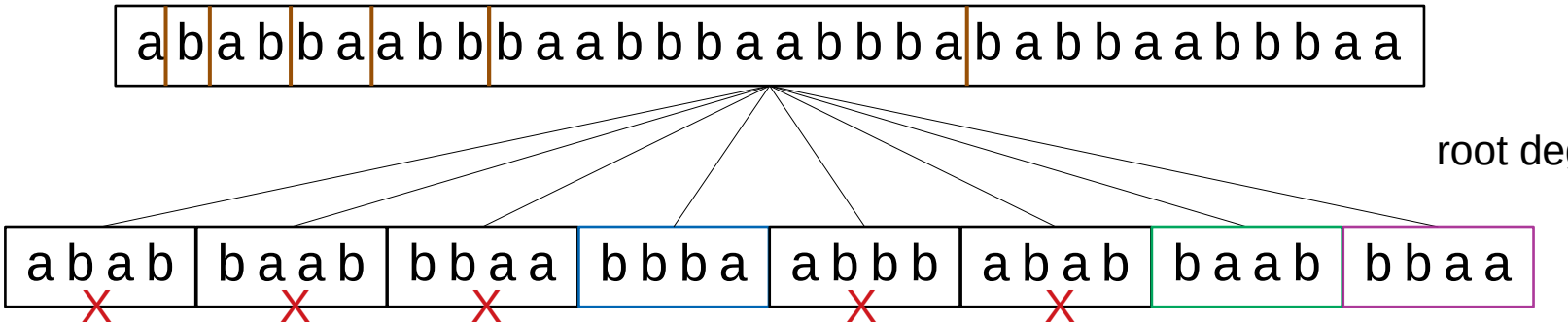


n=32, ***z***=7

root degree ***s*** := 8

- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S
- For unmarked blocks, store pointer to the leftmost occurrence

Block Tree: Example

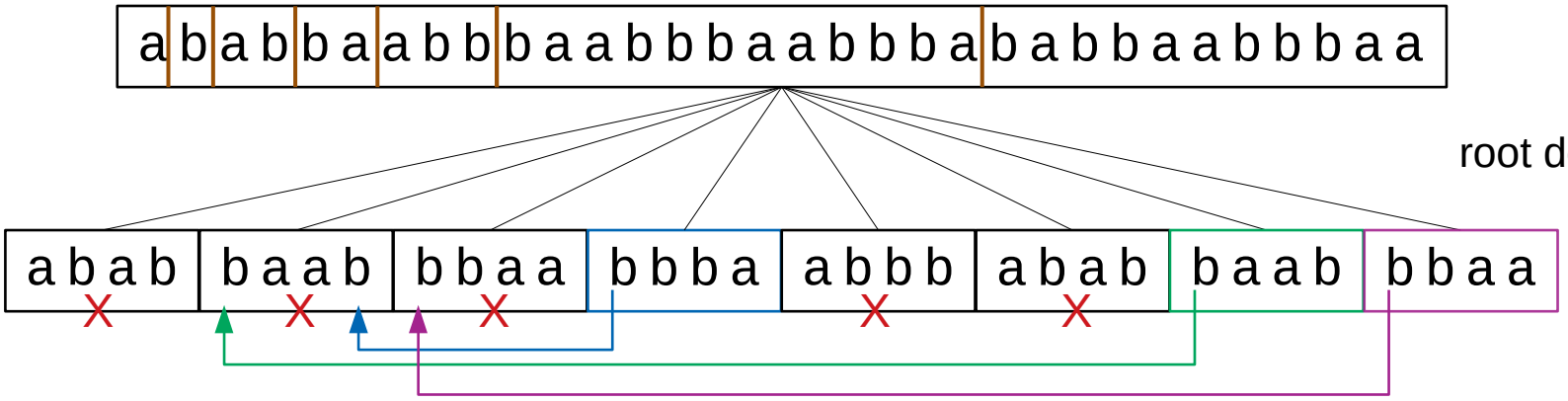


n=32, ***z***=7

root degree ***s*** := 8

- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S
- For unmarked blocks, store pointer to the leftmost occurrence
  - Must be on the same level and within a marked block!

Block Tree: Example

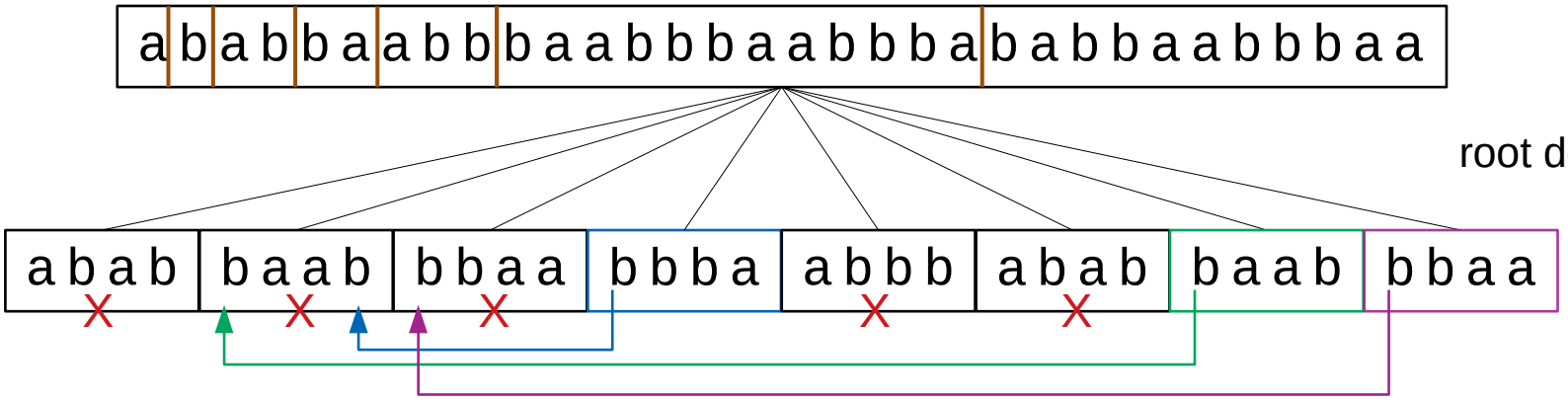


n=32, z=7

root degree s := 8

- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S
- For unmarked blocks, store pointer to the leftmost occurrence
  - Must be on the same level and within a marked block!

Block Tree: Example

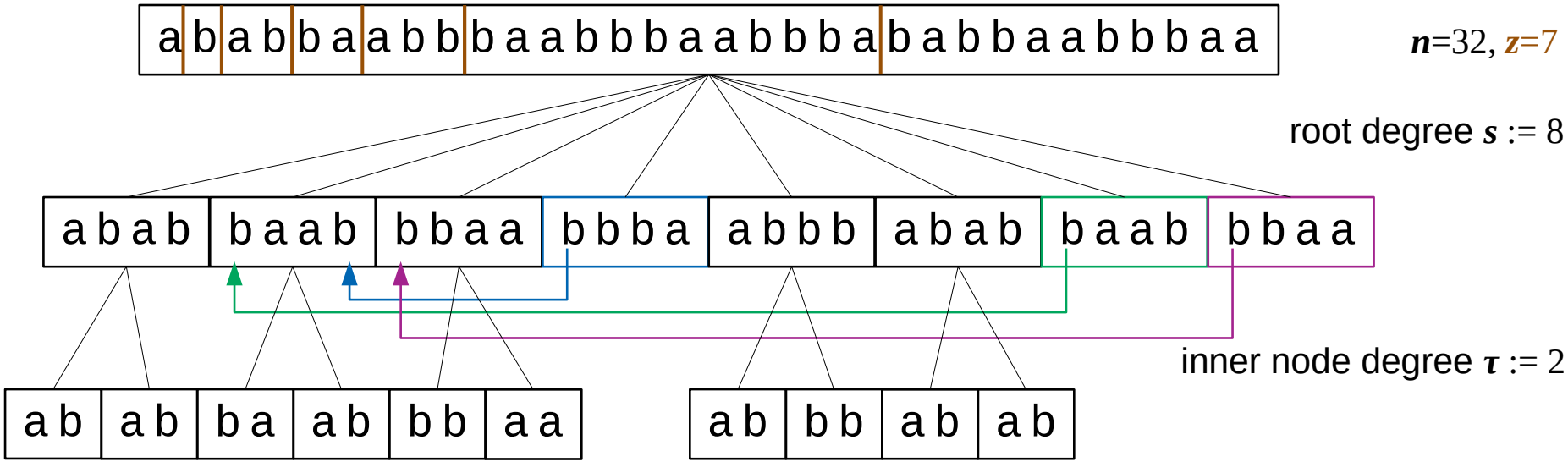


n=32, z=7

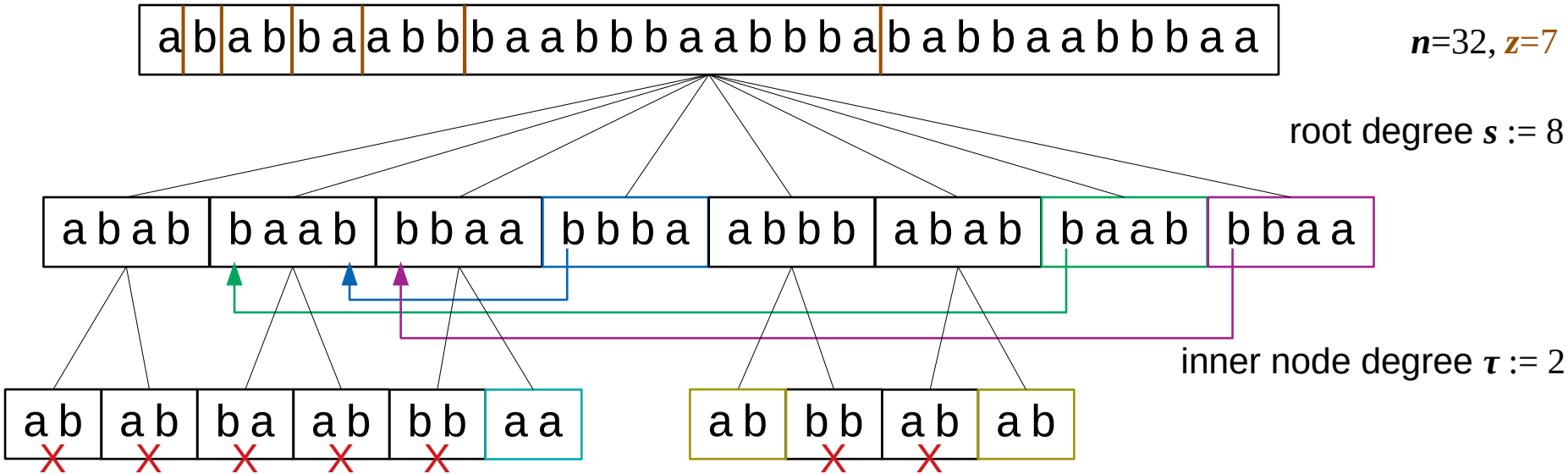
root degree s := 8

- Mark blocks B_i and B_{i+1} if B_iB_{i+1} is the leftmost occurrence in S
- For unmarked blocks, store pointer to the leftmost occurrence
  - Must be on the same level and within a marked block!
- For marked blocks, recurse!

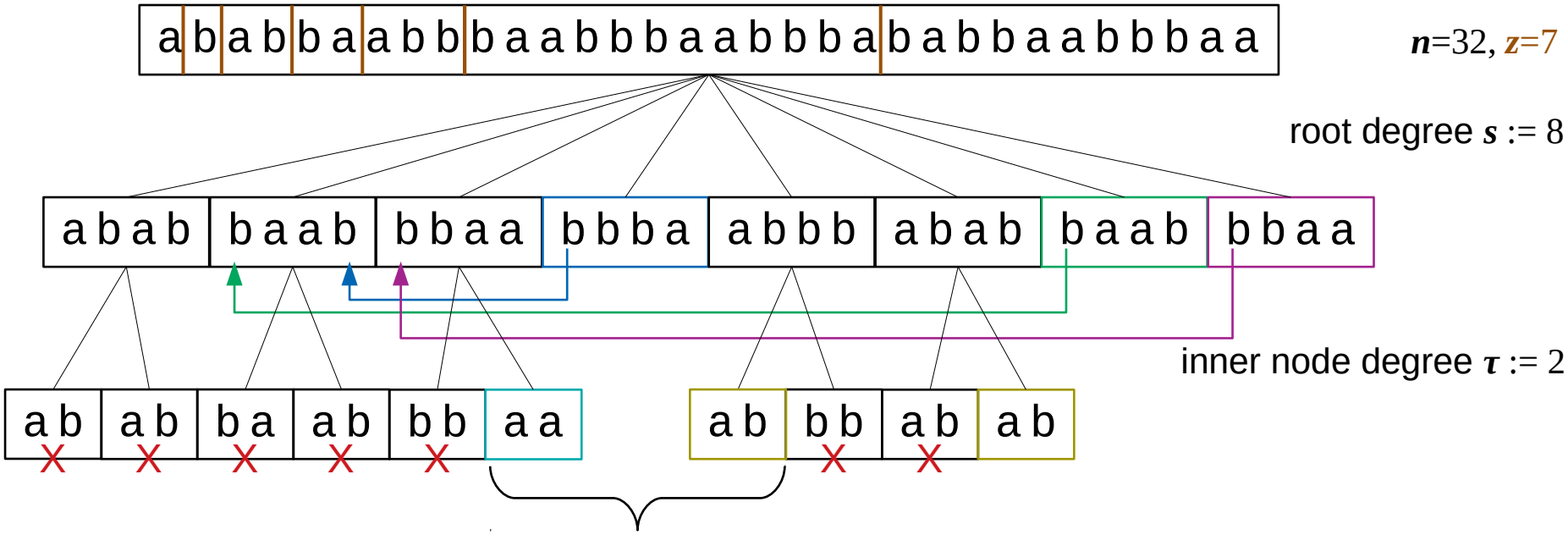
Block Tree: Example



Block Tree: Example

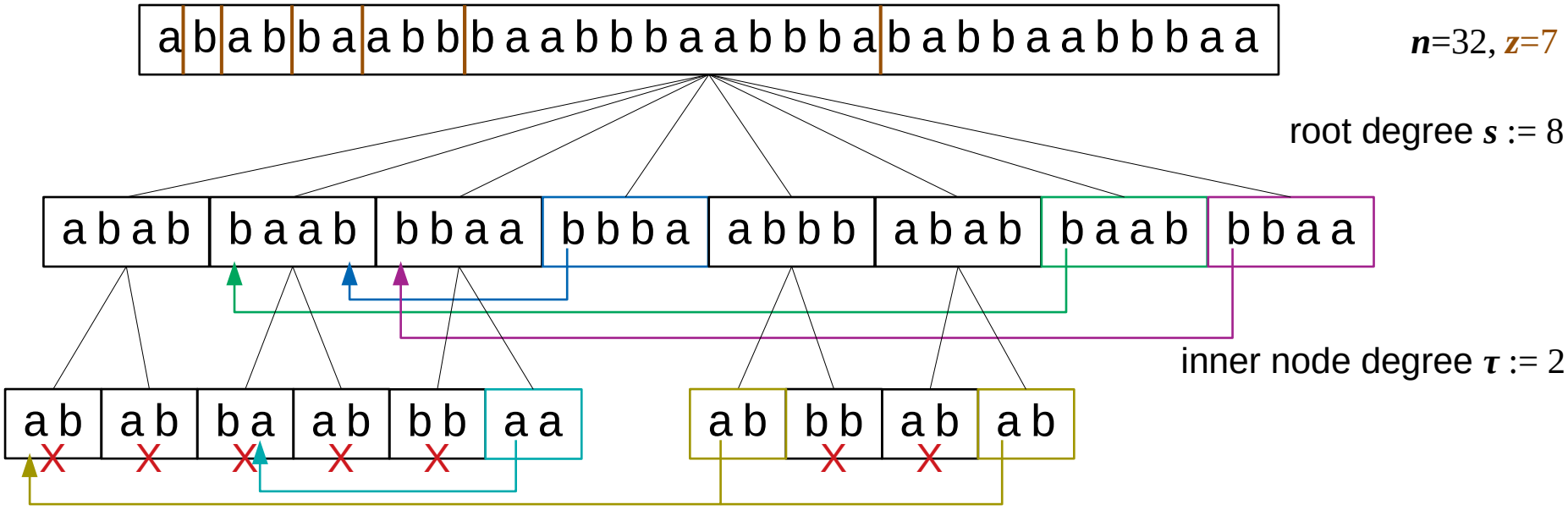


Block Tree: Example

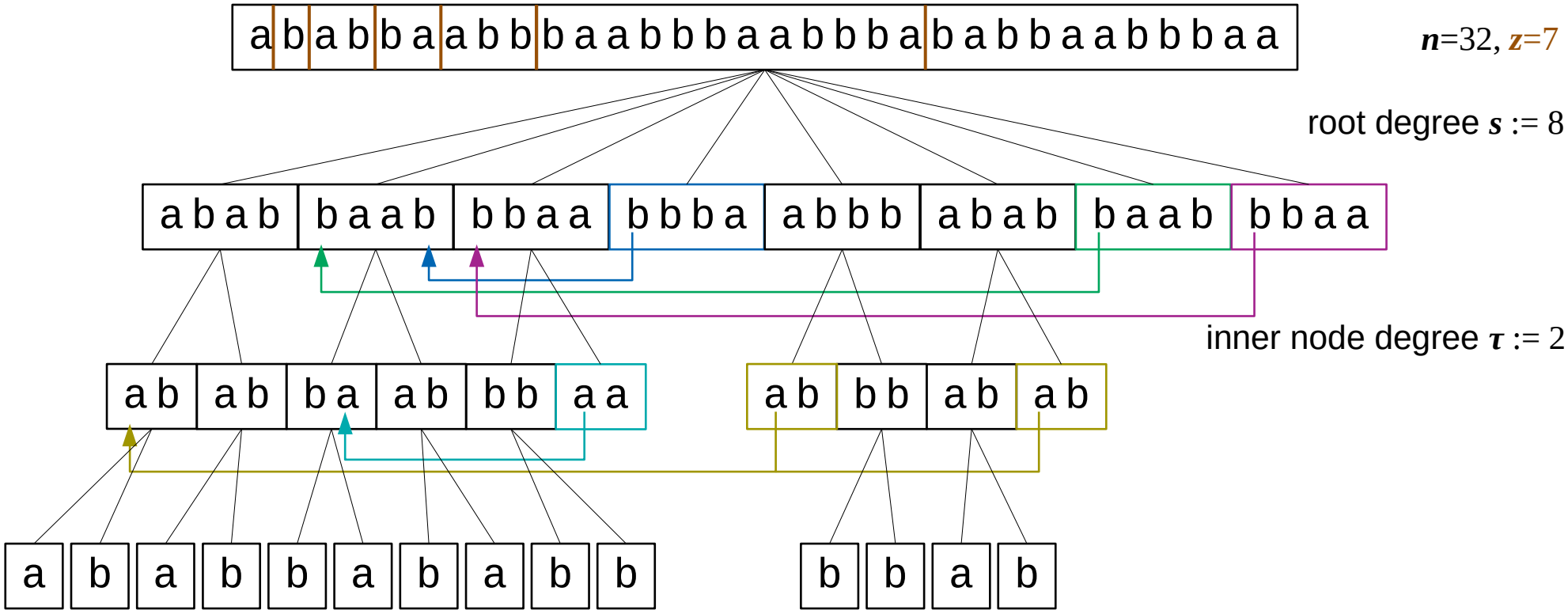


Letfmost occurrence of aaab, but not consecutive in S!

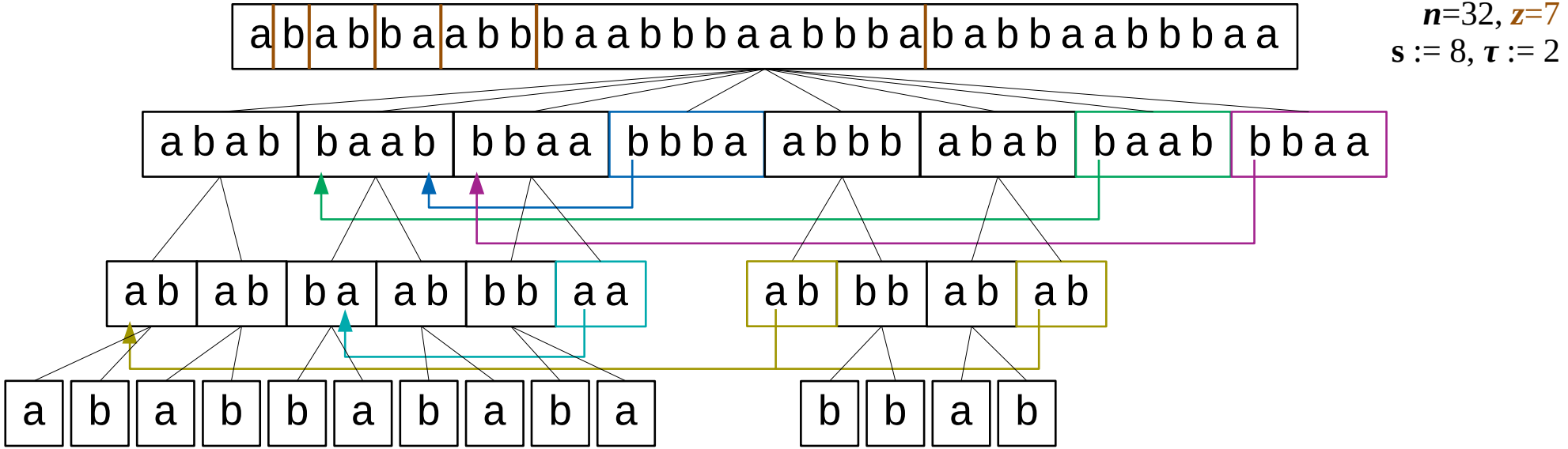
Block Tree: Example



Block Tree: Example

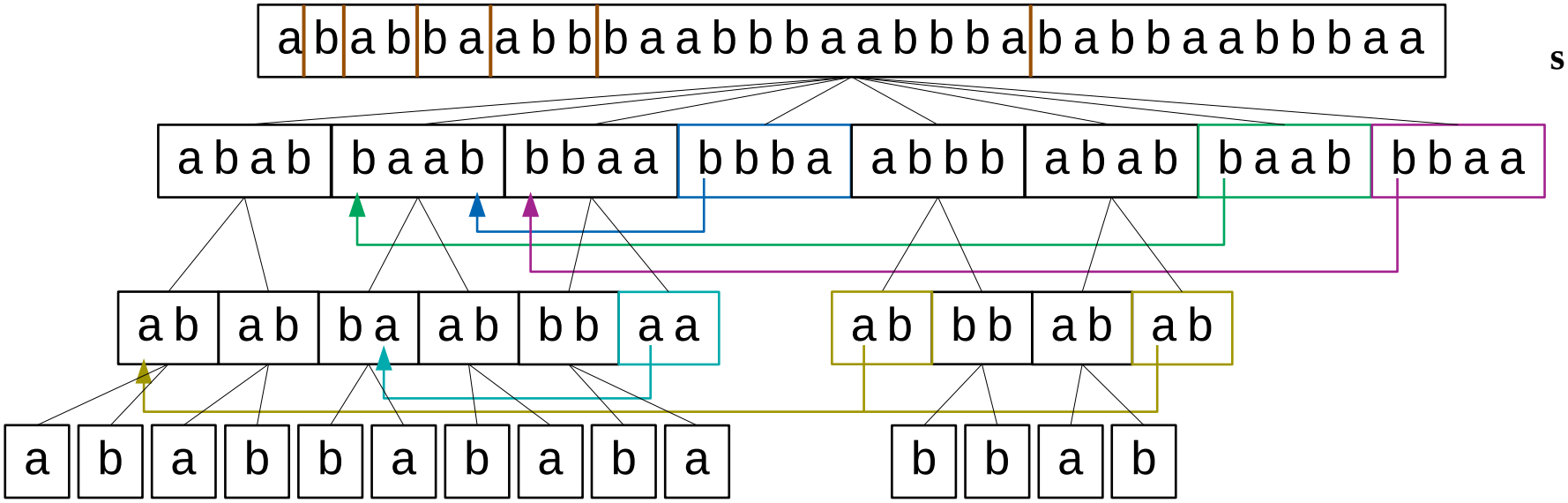


Height



n=32, z=7
$s := 8$, $\tau := 2$

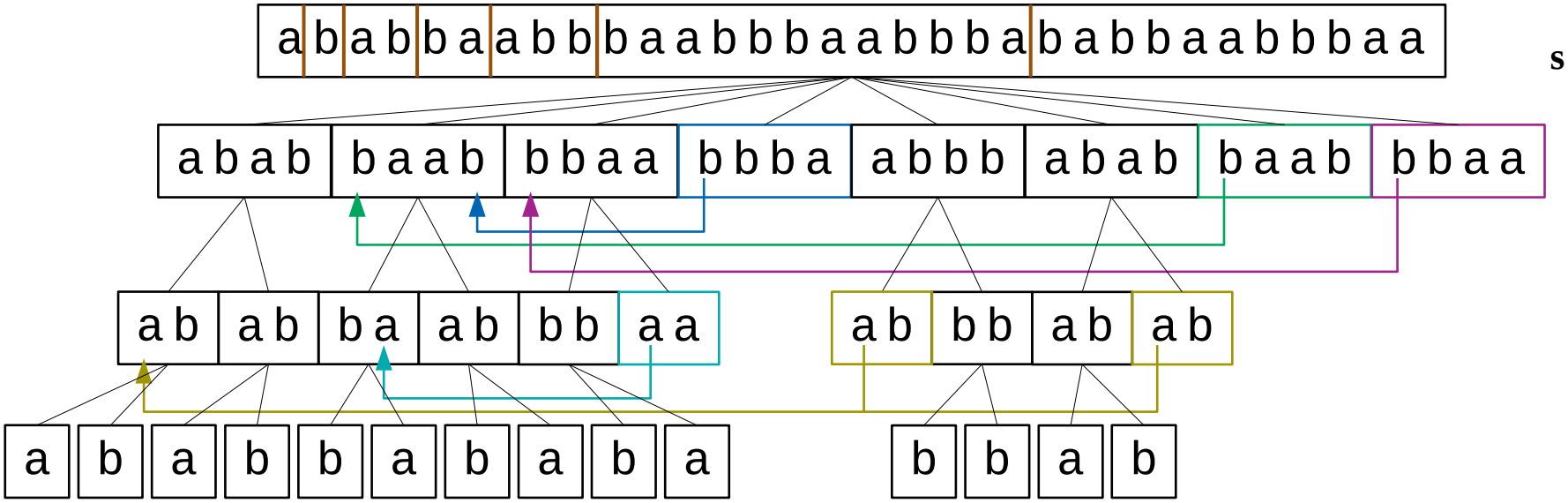
Height



n=32, ***z***=7
s := 8, ***τ*** := 2

- We stop at the level where a block B can be stored in $lg\ n$ bits

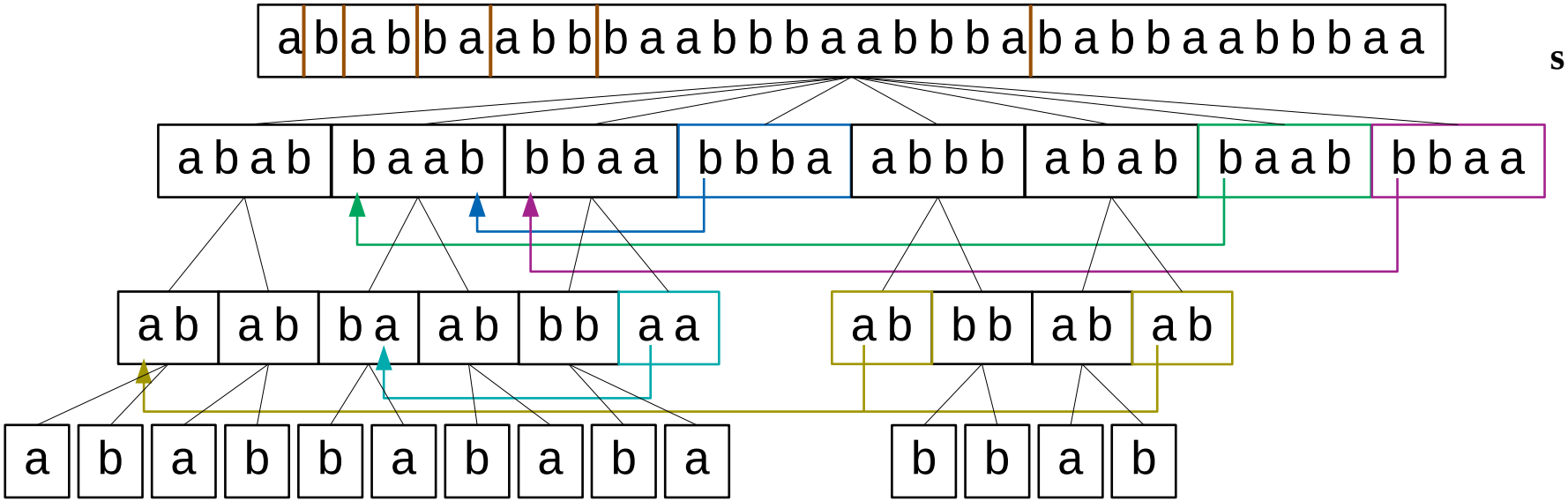
Height



n=32, ***z***=7
s := 8, ***τ*** := 2

- We stop at the level where a block B can be stored in $lg\ n$ bits
  - i.e., when $|B| \lg \sigma \leq \lg n \Leftrightarrow |B| = \Theta(\log_\sigma n)$

Height



n=32, ***z***=7
s := 8, ***τ*** := 2

- We stop at the level where a block B can be stored in $lg\ n$ bits
  - i.e., when $|B| \lg \sigma \leq \lg n \Leftrightarrow |B| = \Theta(\log_\sigma n)$
- The height of the tree is then:

$$h = \log_\tau \left(\frac{n}{s}\right) - \log_\tau(\log_\sigma n) = \log_\tau \frac{n \log \sigma}{s \log n}$$

Number of Blocks

Number of Blocks

- The number of blocks on a level >1 is upper-bounded by $3z\tau$

Number of Blocks

- The number of blocks on a level >1 is upper-bounded by $3z\tau$
  - If an LZ phrase boundary falls into block B_i, only B_i, B_{i-1} and B_{i+1} can be marked

Number of Blocks

- The number of blocks on a level >1 is upper-bounded by $3z\tau$
  - If an LZ phrase boundary falls into block B_i, only B_i, B_{i-1} and B_{i+1} can be marked
  - There are at most z LZ phrase boundaries

Number of Blocks

- The number of blocks on a level >1 is upper-bounded by $3z\tau$
  - If an LZ phrase boundary falls into block B_i, only B_i, B_{i-1} and B_{i+1} can be marked
  - There are at most z LZ phrase boundaries
  - An unmarked block induces τ blocks on the next level □

Space Analysis

Space Analysis

- There are most $O(z\tau)$ blocks on each of the h levels

Space Analysis

- There are most $O(z\tau)$ blocks on each of the h levels
- We require $O(s \lg n)$ bits for the root level (s pointers)

Space Analysis

- There are most $O(z\boldsymbol{\tau})$ blocks on each of the h levels
- We require $O(\boldsymbol{s} \lg n)$ bits for the root level ($\boldsymbol{s}$ pointers)
- Marked blocks require $O(\boldsymbol{\tau} \lg n)$ bits (pointers to children)
  - For the analysis, we charge each pointer to the corresponding child

Space Analysis

- There are most $O(z\boldsymbol{\tau})$ blocks on each of the h levels
- We require $O(\boldsymbol{s} \lg n)$ bits for the root level ($\boldsymbol{s}$ pointers)
- Marked blocks require $O(\boldsymbol{\tau} \lg n)$ bits (pointers to children)
  - For the analysis, we charge each pointer to the corresponding child
- We store unmarked blocks in $O(\lg n)$ bits (pointers)

Space Analysis

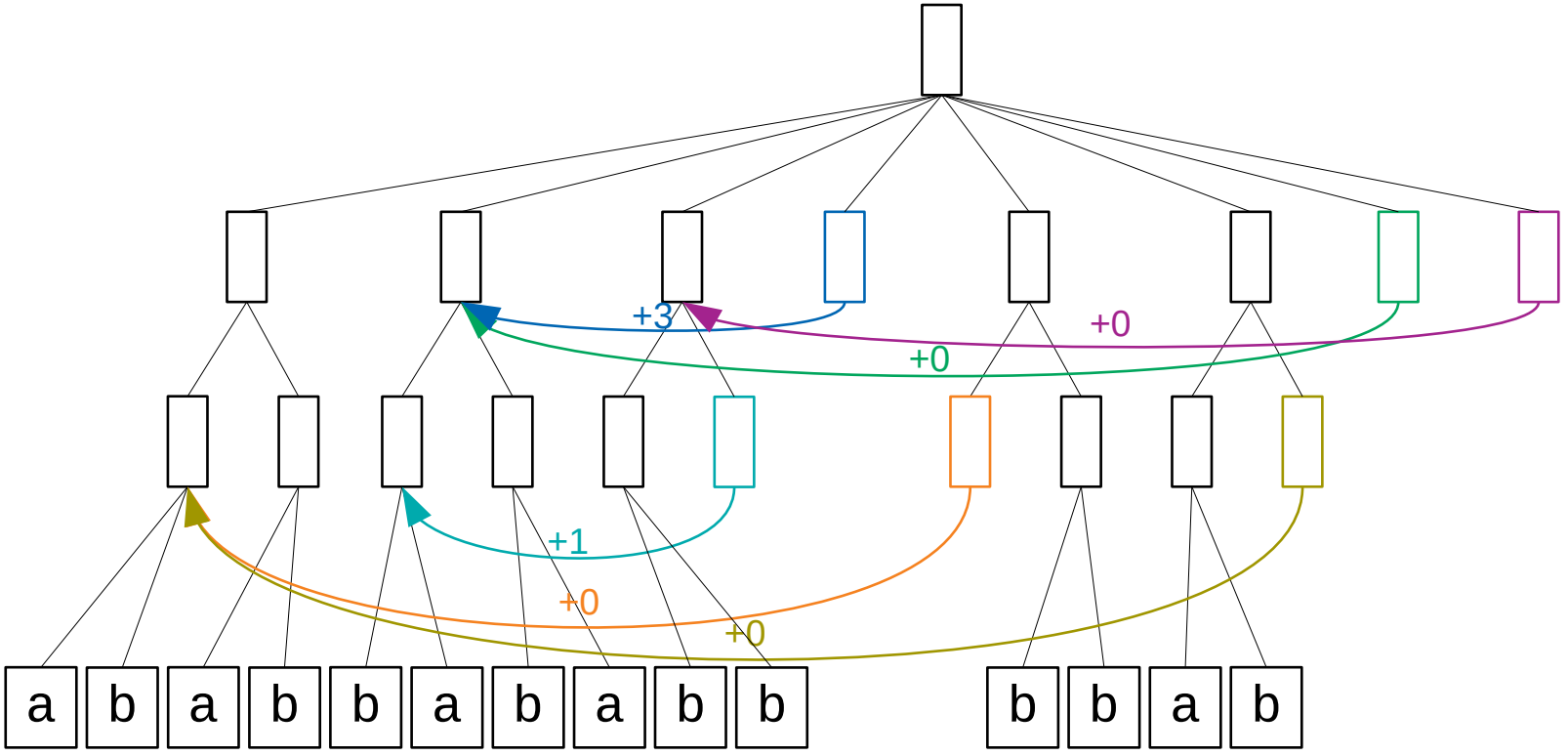
- There are most $O(z\boldsymbol{\tau})$ blocks on each of the h levels
- We require $O(\boldsymbol{s} \lg n)$ bits for the root level ($\boldsymbol{s}$ pointers)
- Marked blocks require $O(\boldsymbol{\tau} \lg n)$ bits (pointers to children)
  - For the analysis, we charge each pointer to the corresponding child
- We store unmarked blocks in $O(\lg n)$ bits (pointers)
- Leaves are stored as plain text in $\Theta(\log_\sigma n) \lg(\sigma) = O(\lg n)$ bits

Space Analysis

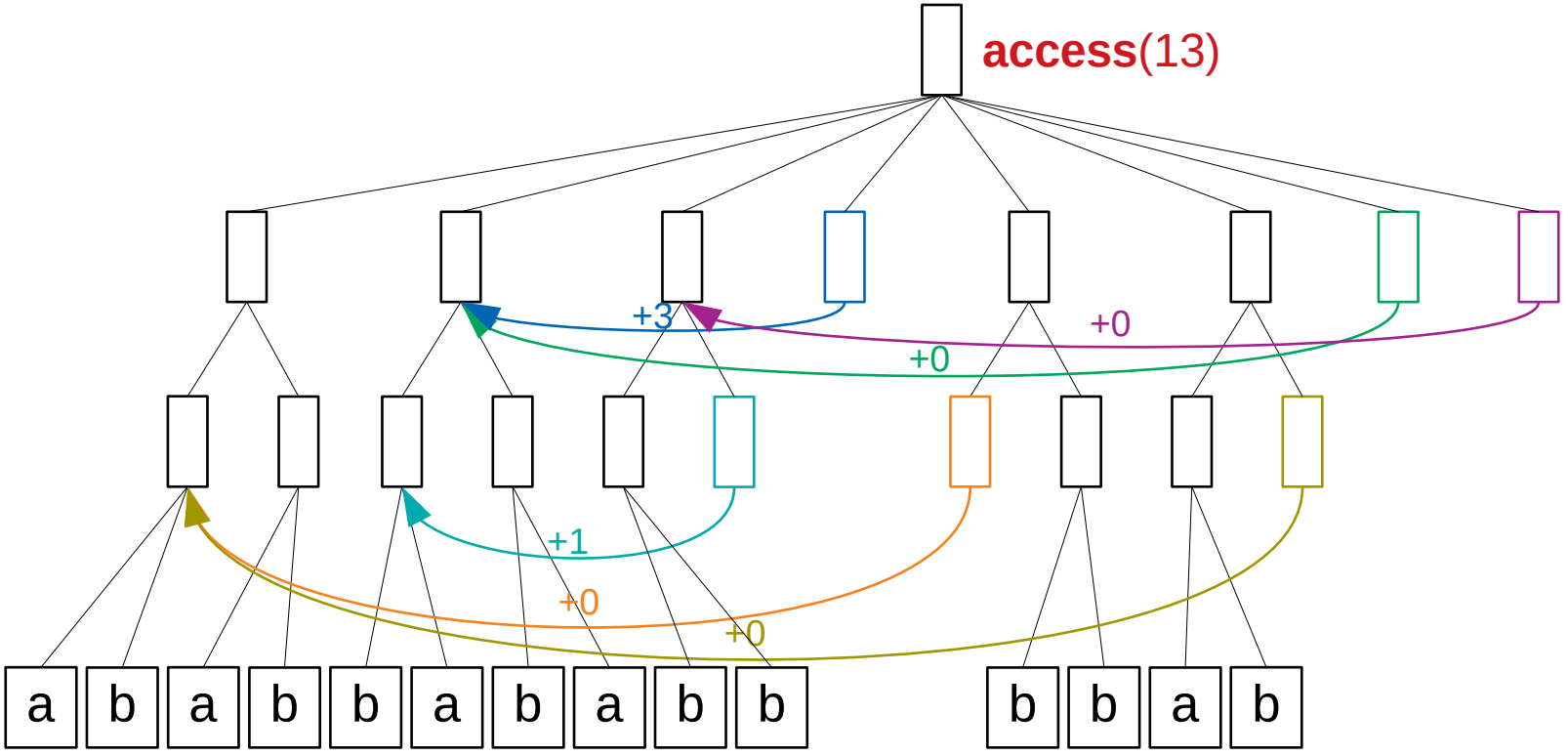
- There are most $O(z\boldsymbol{\tau})$ blocks on each of the h levels
- We require $O(\boldsymbol{s} \lg n)$ bits for the root level ($\boldsymbol{s}$ pointers)
- Marked blocks require $O(\boldsymbol{\tau} \lg n)$ bits (pointers to children)
  - For the analysis, we charge each pointer to the corresponding child
- We store unmarked blocks in $O(\lg n)$ bits (pointers)
- Leaves are stored as plain text in $\Theta(\log_\sigma n) \lg(\sigma) = O(\lg n)$ bits
- The total space required to store the block tree is thus (in bits):

$$\mathcal{O}\left(\left(s + z\tau \log_\tau \frac{n \lg \sigma}{s \lg n}\right) \lg n\right)$$

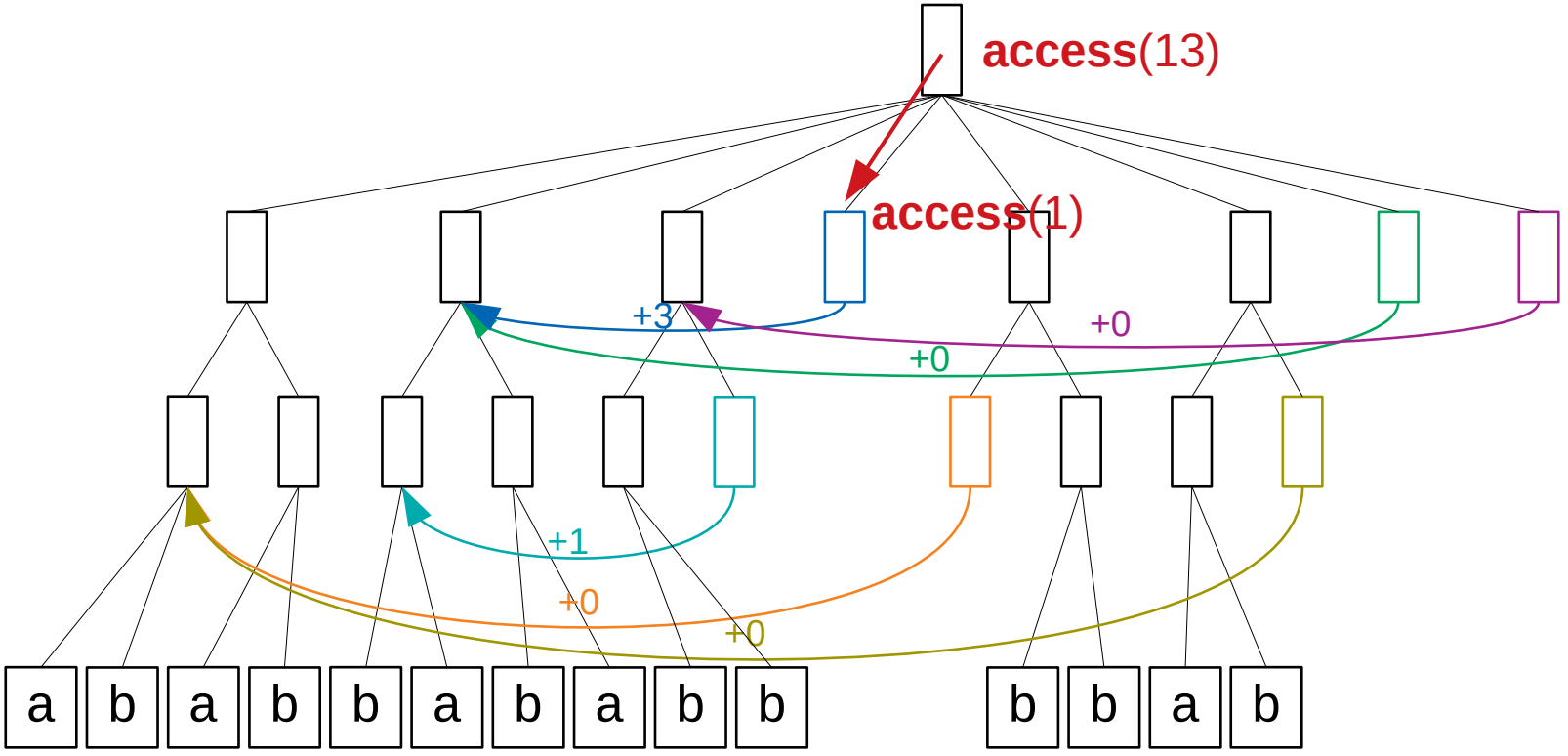
Storage



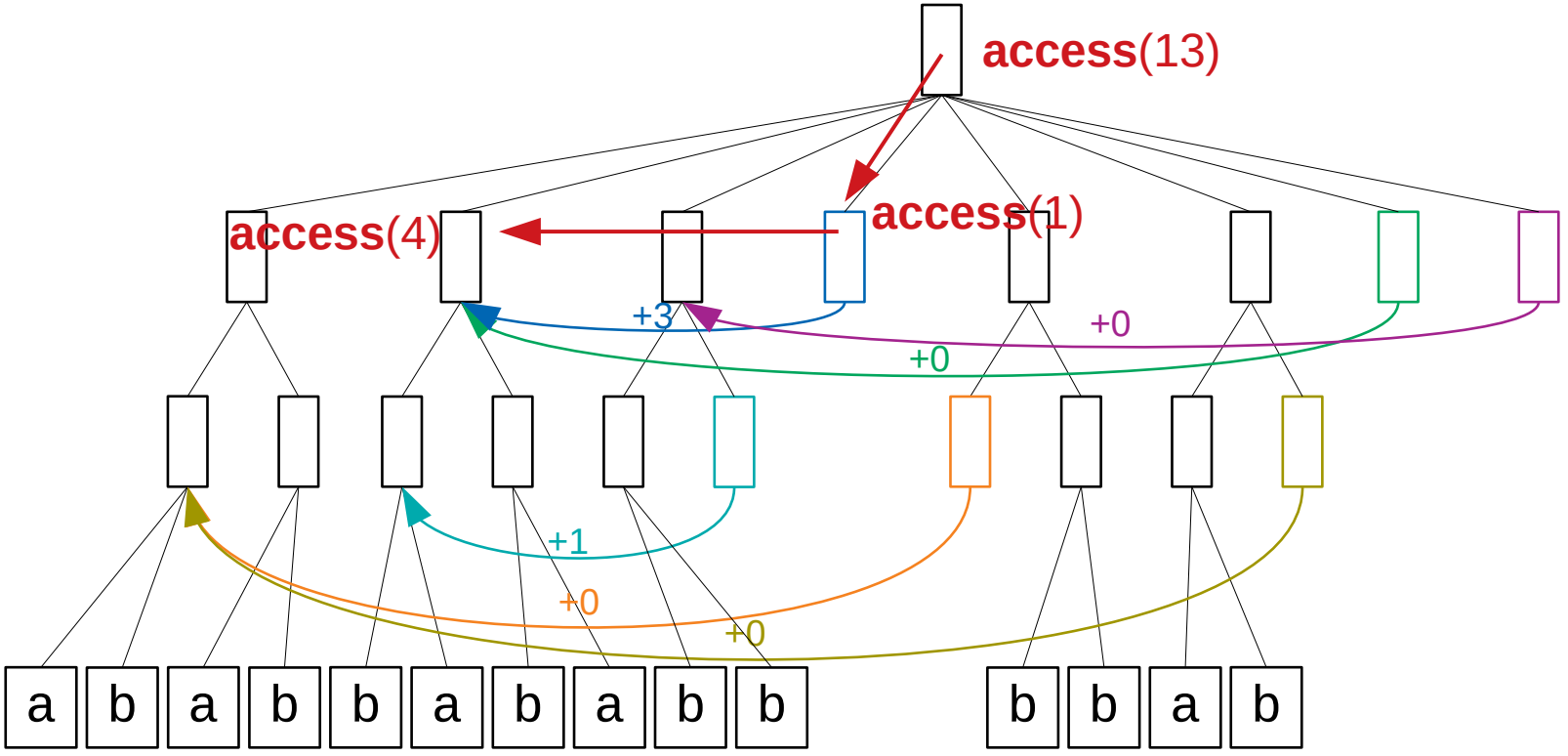
Access Query



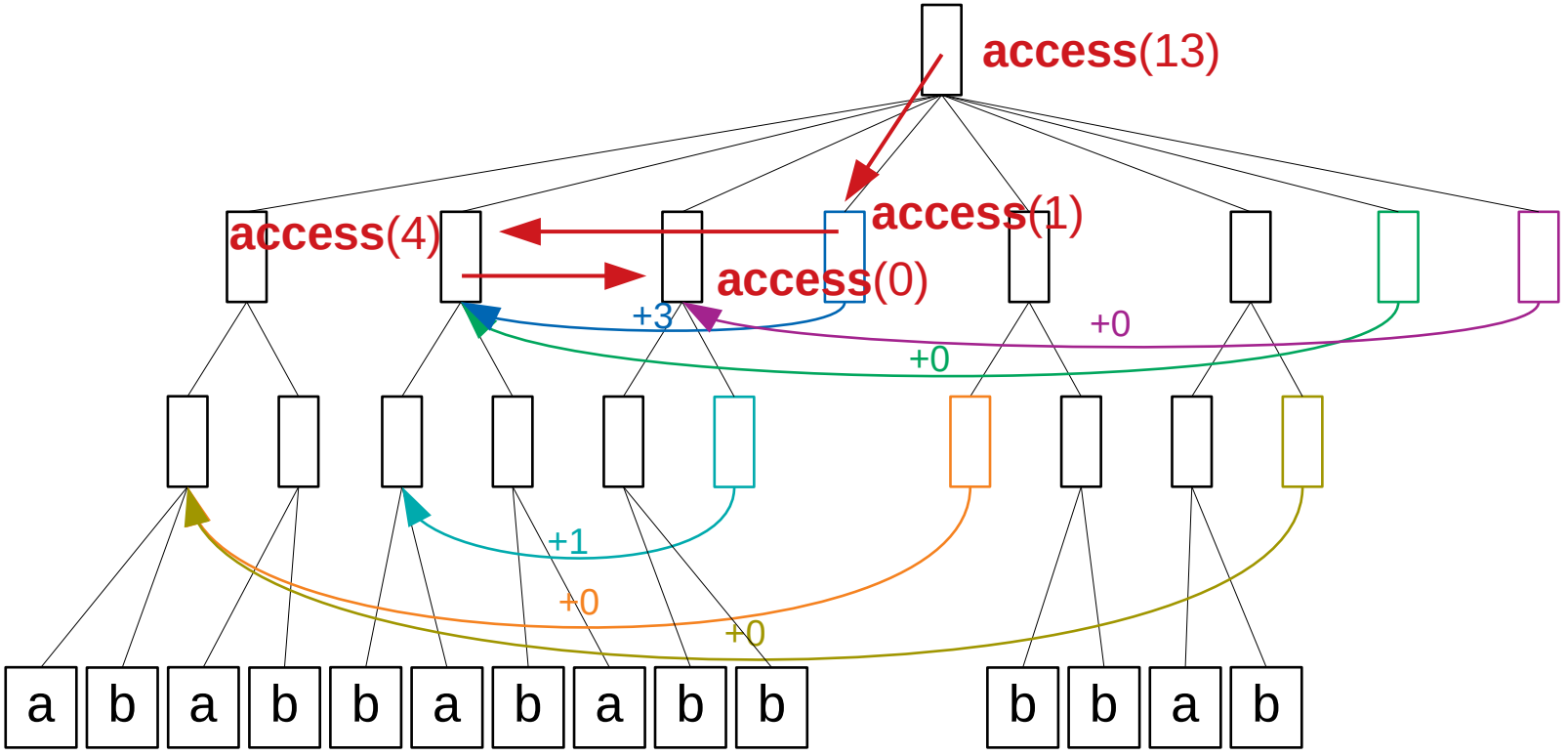
Access Query



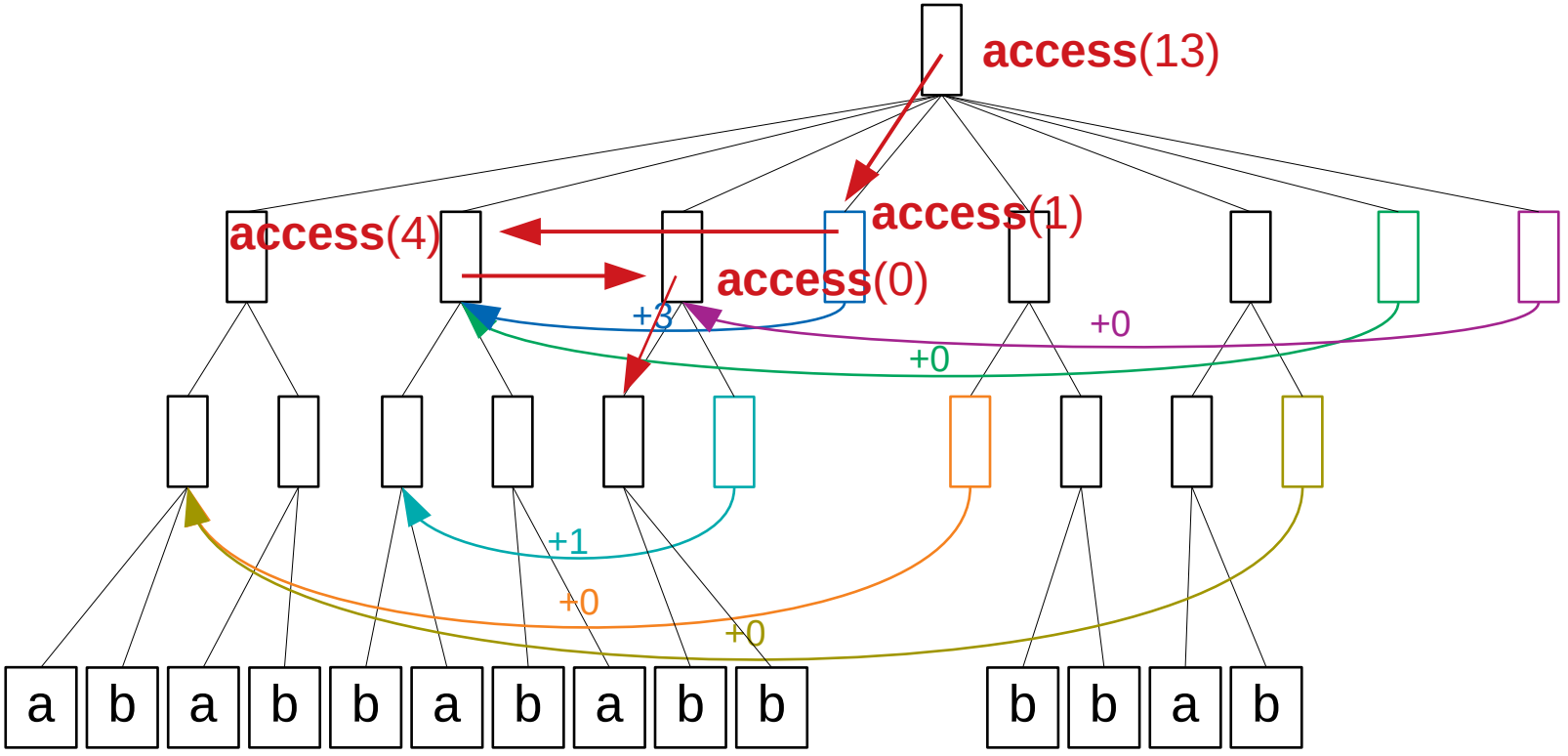
Access Query



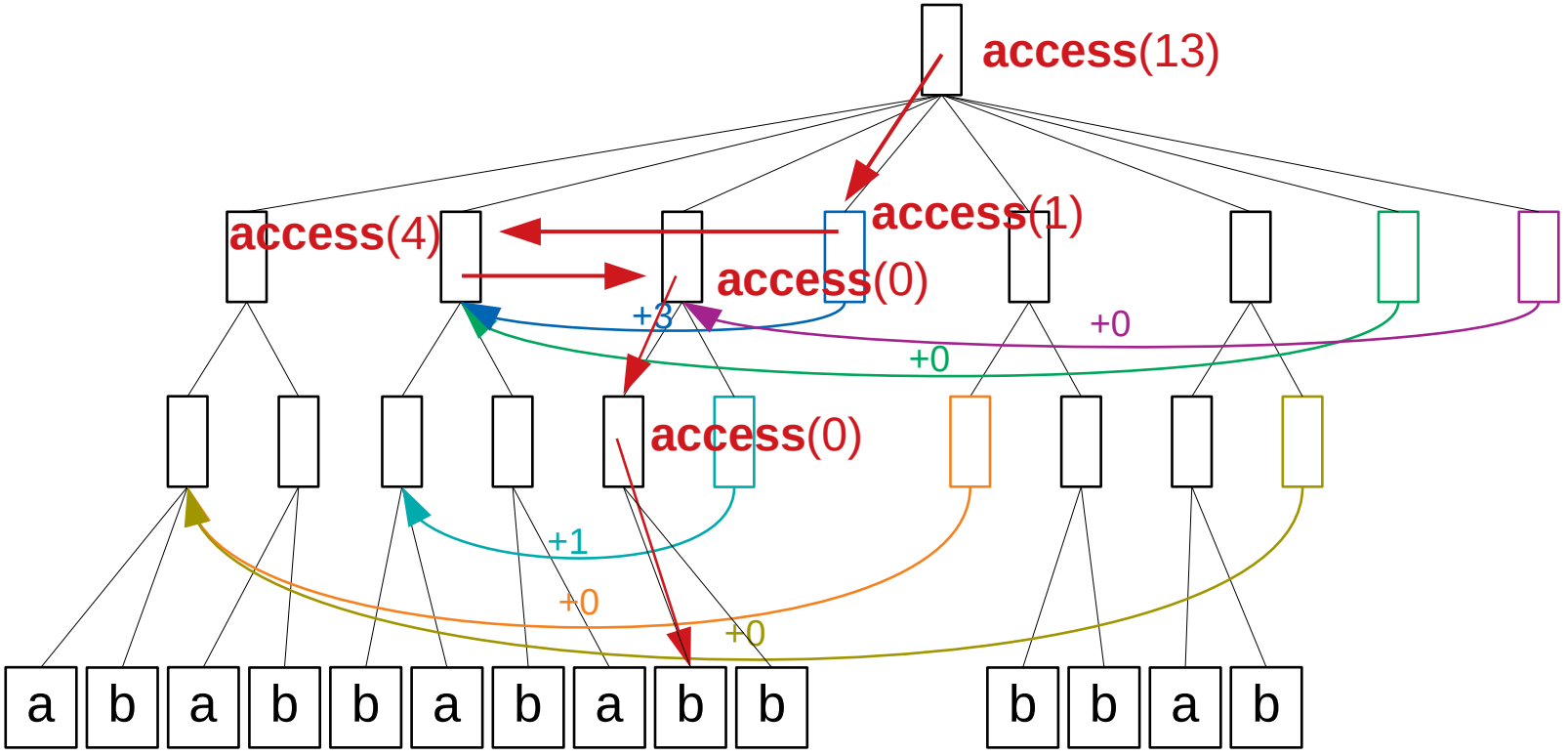
Access Query



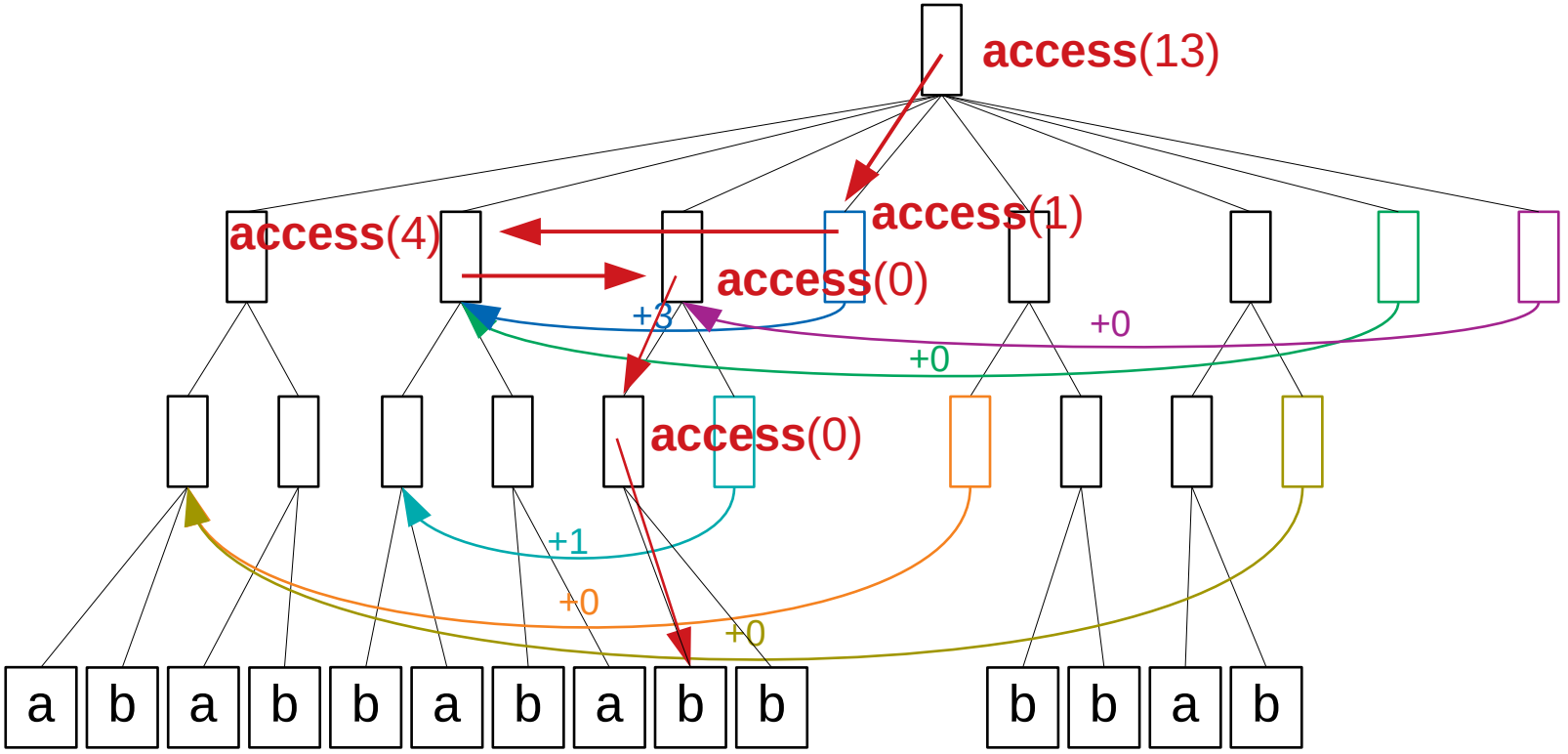
Access Query



Access Query

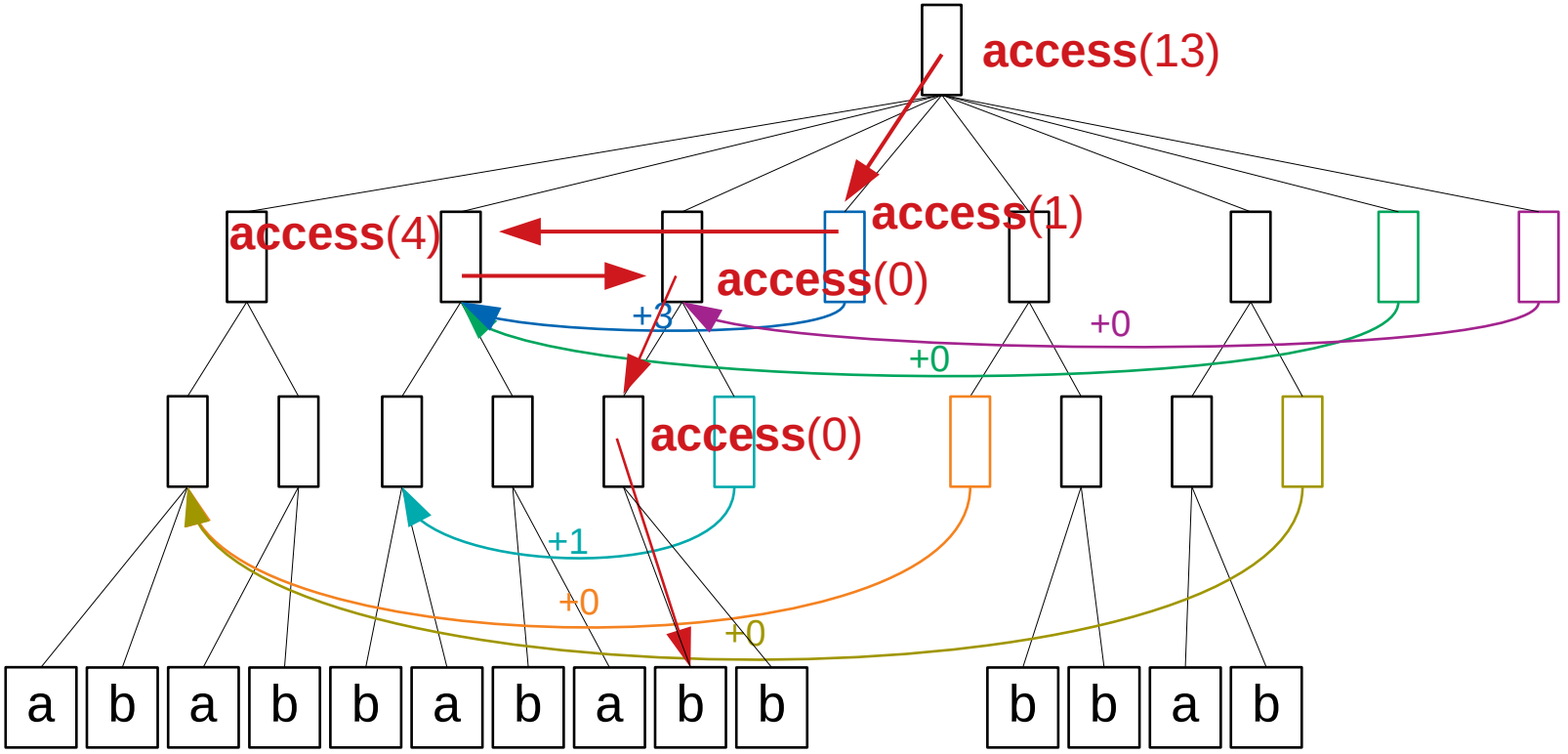


Access Query



- At most 3 jumps on every level

Access Query



- At most 3 jumps on every level
- Access requires time $O(h)$

Substring Extraction

Substring Extraction

- m subsequent accesses require time $O(mh)$

Substring Extraction

- m subsequent accesses require time $O(mh)$
- **Idea:** Store the first and last $\log_\sigma n$ symbols at *every* node ($O(\lg n)$ bits)

Substring Extraction

- m subsequent accesses require time $O(mh)$
- **Idea:** Store the first and last $\log_\sigma n$ symbols at *every* node ($O(\lg n)$ bits)
  - This does not worsen the asymptotic space requirement

Substring Extraction

- m subsequent accesses require time $O(mh)$
- **Idea:** Store the first and last $\log_\sigma n$ symbols at *every* node ($O(\lg n)$ bits)
  - This does not worsen the asymptotic space requirement
- Extracting a substring of length ≤ $\log_\sigma n$ now requires time $O(h)$

Substring Extraction

- m subsequent accesses require time $O(mh)$
- **Idea:** Store the first and last $\log_\sigma n$ symbols at *every* node ($O(\lg n)$ bits)
  - This does not worsen the asymptotic space requirement
- Extracting a substring of length ≤ $\log_\sigma n$ now requires time $O(h)$
  - If it spans across a block boundary, lookup corresponding last and first symbols

Substring Extraction

- m subsequent accesses require time $O(mh)$
- **Idea:** Store the first and last $\log_\sigma n$ symbols at *every* node ($O(\lg n)$ bits)
  - This does not worsen the asymptotic space requirement
- Extracting a substring of length ≤ $\log_\sigma n$ now requires time $O(h)$
  - If it spans across a block boundary, lookup corresponding last and first symbols
  - Otherwise recurse until it does span a block boundary

Substring Extraction

- m subsequent accesses require time $O(mh)$
- **Idea:** Store the first and last $\log_\sigma n$ symbols at *every* node ($O(\lg n)$ bits)
  - This does not worsen the asymptotic space requirement
- Extracting a substring of length ≤ $\log_\sigma n$ now requires time $O(h)$
  - If it spans across a block boundary, lookup corresponding last and first symbols
  - Otherwise recurse until it does span a block boundary
- Extracting a substring of length m then requires time $O((1+m/\log_\sigma n)h)$

Rank Queries

Rank Queries

- At each inner node, store #occurrences in preceding blocks for each symbol

Rank Queries

- At each inner node, store #occurrences in preceding blocks for each symbol
  - $O(\sigma \lg n)$ bits per block

Rank Queries

- At each inner node, store #occurrences in preceding blocks for each symbol
  - $O(\sigma \lg n)$ bits per block
- Binary rank data structure for the last level for each symbol

Rank Queries

- At each inner node, store #occurrences in preceding blocks for each symbol
  - $O(\sigma \lg n)$ bits per block
- Binary rank data structure for the last level for each symbol
  - $O(\sigma z \tau \log_\sigma n)$ bits

Rank Queries

- At each inner node, store #occurrences in preceding blocks for each symbol
  - $O(\sigma \lg n)$ bits per block
- Binary rank data structure for the last level for each symbol
  - $O(\sigma z \tau \log_\sigma n)$ bits
- Time $O(h)$, space blowup by factor $O(\sigma)$

Select Queries

Select Queries

- … a bit more complicated, but similar idea

Select Queries

- … a bit more complicated, but similar idea
- Time $O(h)$, space blowup by factor $O(\sigma)$

Parameters

Parameters

- The space required by the block tree is $\mathcal{O}((s + z\tau \log_\tau \frac{n \lg \sigma}{s \lg n}) \lg n)$ bits

Parameters

- The space required by the block tree is $\mathcal{O}((s + z\tau \log_\tau \frac{n \lg \sigma}{s \lg n}) \lg n)$ bits
- With $s := z$, this becomes $\mathcal{O}((z\tau \log_\tau \frac{n \lg \sigma}{z \lg n}) \lg n)$ bits

Parameters

- The space required by the block tree is $\mathcal{O}((s + z\tau \log_\tau \frac{n \lg \sigma}{s \lg n}) \lg n)$ bits
- With $s := z$, this becomes $\mathcal{O}((z\tau \log_\tau \frac{n \lg \sigma}{z \lg n}) \lg n)$ bits
  - The factor $n \lg(\sigma) / z \lg(n)$ is the LZ compression ratio

Parameters

- The space required by the block tree is $\mathcal{O}((s + z\tau \log_\tau \frac{n \lg \sigma}{s \lg n}) \lg n)$ bits
- With $s := z$, this becomes $\mathcal{O}((z\tau \log_\tau \frac{n \lg \sigma}{z \lg n}) \lg n)$ bits
  - The factor $n \lg(\sigma) / z \lg(n)$ is the LZ compression ratio
- With $\tau := O(1)$, we get the proclaimed $\mathcal{O}(z \lg \frac{n \lg \sigma}{z \lg n}) = \mathcal{O}(z \lg(n/z))$ *words*

Parameters

- The space required by the block tree is $\mathcal{O}((s + z\tau \log_\tau \frac{n \lg \sigma}{s \lg n}) \lg n)$ bits
- With $s := z$, this becomes $\mathcal{O}((z\tau \log_\tau \frac{n \lg \sigma}{z \lg n}) \lg n)$ bits
  - The factor $n \lg(\sigma) / z \lg(n)$ is the LZ compression ratio
- With $\tau := O(1)$, we get the proclaimed $\mathcal{O}(z \lg \frac{n \lg \sigma}{z \lg n}) = \mathcal{O}(z \lg(n/z))$ *words*
- The height of the block tree – and thus the access query time – is then

$$\mathcal{O}(\log_\tau \frac{n \lg \sigma}{s \lg n}) = \mathcal{O}(\lg \frac{n \lg \sigma}{z \lg n}) = \mathcal{O}(\lg(n/z))$$

Parameters

- The space required by the block tree is $\mathcal{O}((s + z\tau \log_\tau \frac{n \lg \sigma}{s \lg n}) \lg n)$ bits
- With $s := z$, this becomes $\mathcal{O}((z\tau \log_\tau \frac{n \lg \sigma}{z \lg n}) \lg n)$ bits
  - The factor $n \lg(\sigma) / z \lg(n)$ is the LZ compression ratio
- With $\tau := O(1)$, we get the proclaimed $\mathcal{O}(z \lg \frac{n \lg \sigma}{z \lg n}) = \mathcal{O}(z \lg(n/z))$ *words*
- The height of the block tree – and thus the access query time – is then

$$\mathcal{O}(\log_\tau \frac{n \lg \sigma}{s \lg n}) = \mathcal{O}(\lg \frac{n \lg \sigma}{z \lg n}) = \mathcal{O}(\lg(n/z))$$

More Parameters

More Parameters

- Let $s := z$ and $\tau := (n \lg(\sigma) / z \lg(n))^{\varepsilon}$ for $0 < \varepsilon < 1$

More Parameters

- Let $s := z$ and $\tau := (n \lg(\sigma) / z \lg(n))^{\varepsilon}$ for $0 < \varepsilon < 1$
- The space now becomes $O(\varepsilon^{-1} z^{1-\varepsilon} n^{\varepsilon})$ words

More Parameters

- Let $s := z$ and $\tau := (n \lg(\sigma) / z \lg(n))^{\varepsilon}$ for $0 < \varepsilon < 1$
- The space now becomes $O(\varepsilon^{-1} z^{1-\varepsilon} n^{\varepsilon})$ words
- The height is now $O(1/\varepsilon)$

More Parameters

- Let $s := z$ and $\tau := (n \lg(\sigma) / z \lg(n))^{\varepsilon}$ for $0 < \varepsilon < 1$
- The space now becomes $O(\varepsilon^{-1} z^{1-\varepsilon} n^{\varepsilon})$ words
- The height is now $O(1/\varepsilon)$

Blocks Trees and String Complexity

Blocks Trees and String Complexity

- Let $S(k)$ be the number of distinct substrings of length k in S

Blocks Trees and String Complexity

- Let $S(k)$ be the number of distinct substrings of length k in S
- The string complexity is $\delta := \max\{S(k)/k, k \geq 1\} \leq z$

Blocks Trees and String Complexity

- Let $S(k)$ be the number of distinct substrings of length k in S
- The string complexity is $\delta := \max\{S(k)/k, k \geq 1\} \leq z$
- With $s := \delta$, the space can become $O(\delta \lg(n/\delta))$, retaining height $O(\lg(n/z))$

Blocks Trees and String Complexity

- Let $S(k)$ be the number of distinct substrings of length k in S
- The string complexity is $\delta := \max\{S(k)/k, k \geq 1\} \leq z$
- With $s := \delta$, the space can become $O(\delta \lg(n/\delta))$, retaining height $O(\lg(n/z))$
  - [Kociumaka et al., LATIN 2020]