Reverse Engineering of Data Structures on Strings

Tomohiro I

Department of Informatics, Kyushu University, Japan

- Σ : alphabet (a set of characters).
- $s \in \Sigma^*$: string (a sequence of characters).
- When $s = xy$, x is called a prefix of s, y is called a suffix of s.

Data Structures for String Processing

- There are many data structures for processing strings efficiently such as suffix arrays and border arrays.

Foundation of string processing.

Direct Problem

string → data structure

- Compute the data structure of a given string.

Reverse Problem

data structure → string

- Compute a string which has a given data structure.

Theoretical — to reveal the combinatorial properties of the data structure.

Motivations

Practical — to provide a data set for testing software efficiently.

Related Work

Reverse problems on

- border array
  - [Franek et al. ,2002]
  - [Duval et al. ,2005]
- suffix array
  - [Duval and Lefebvre, 2002]
  - [Bannai et al. , 2003]
  - [Schürmann et al. , 2005]
- directed acyclic word graph
  - [Bannai et al. ,2003]
- prefix table
  - [Clement et al. ,2009]
- cover array
  - [Crochemore et al. ,2010]

etc.

Reverse Problem on Palindromic Structures of Strings

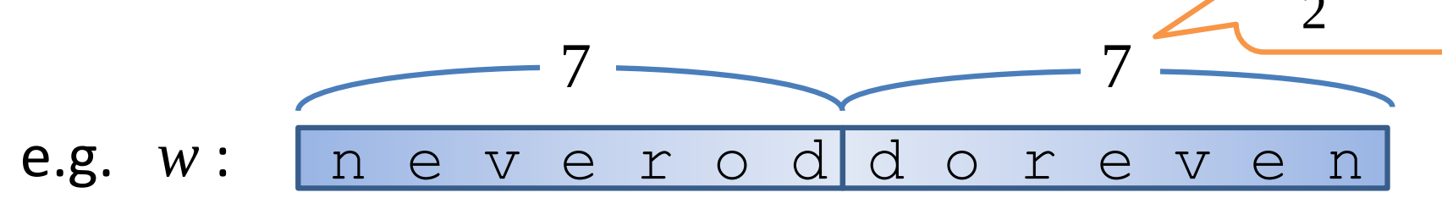
3. T. I, S. Inenaga, H. Bannai, M. Takeda, **Counting and Verifying Maximal Palindromes**, in: Proc. SPIRE'10, Vol. 6393 of LNCS, 2010, pp. 135–146.

Is (0.5, 0) (1.5, 0) (2.5, 2) (3.5, 0) (4.5, 0) (5.5, 0) (6.5, 0) (7.5, 1) (8.5, 1) (9.5, 0) (1, 0.5) (2, 0.5) (3, 0.5) (4, 0.5) (5, 3.5) (6, 0.5) (7, 0.5) (8, 1.5) (9, 0.5) a valid set of maximal palindromes?

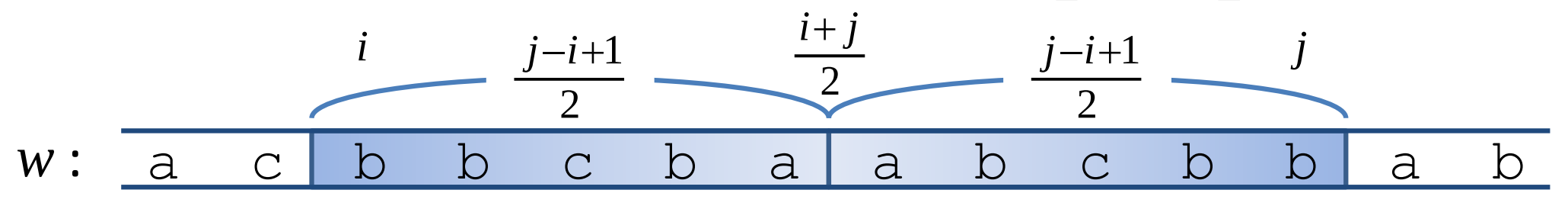
Maximal Palindromes

- A palindrome is a symmetric string that reads the same forward and backward.

$\frac{|w|}{2}$: radius



- If $w[i..j]$ is a palindrome and $w[i-1..j+1]$ is not a palindrome, $w[i..j]$ is called a maximal palindrome at center $\frac{i+j}{2}$ and denoted as $(\frac{i+j}{2}, \frac{j-i+1}{2})$.



- $Pals(w)$: the set of maximal palindromes of w.

	1	2	3	4	5	6	7	8	9	
w	a	b	b	a	c	a	b	b	b	
$Pals(w)$	(0.5, 0)	(1.5, 0)	(2.5, 2)	(3.5, 0)	(4.5, 0)	(5.5, 0)	(6.5, 0)	(7.5, 1)	(8.5, 1)	(9.5, 0)
	(1, 0.5)	(2, 0.5)	(3, 0.5)	(4, 0.5)	(5, 3.5)	(6, 0.5)	(7, 0.5)	(8, 1.5)	(9, 0.5)	

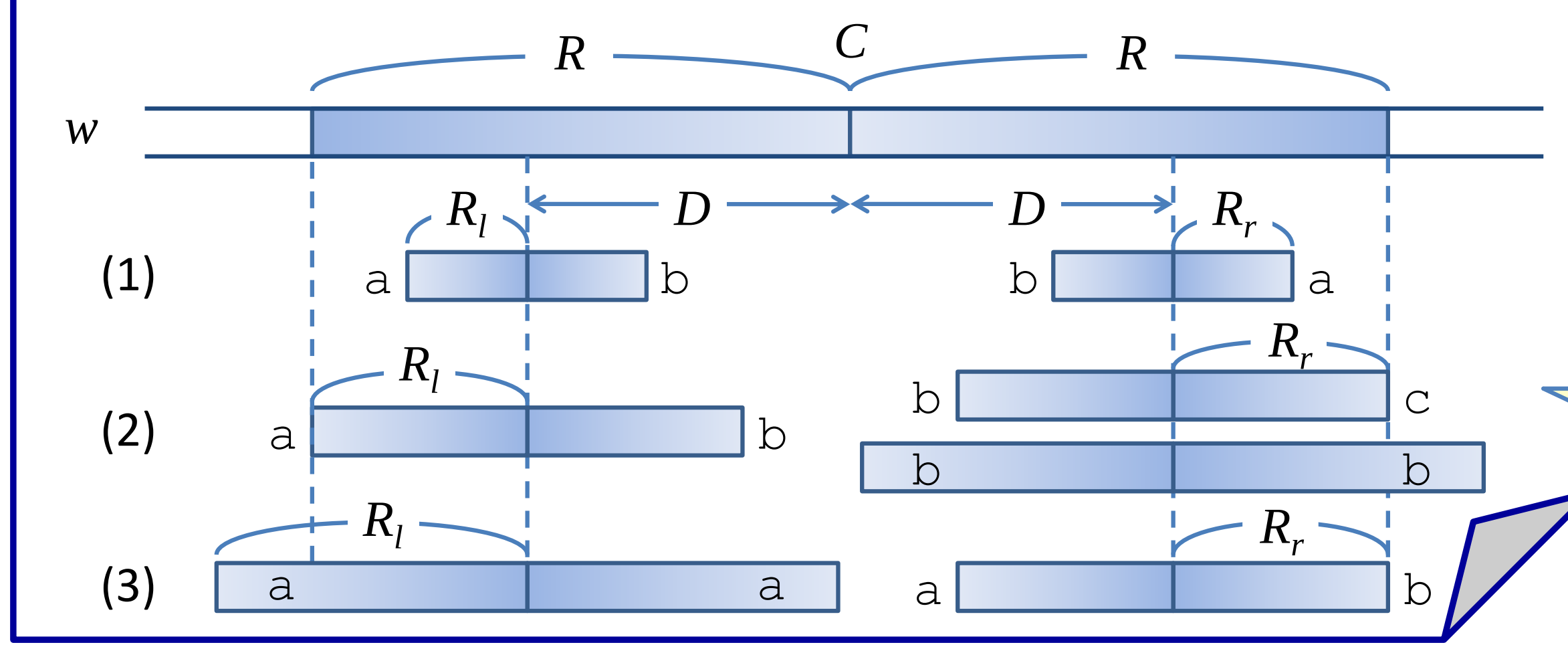
- For a string w of length n, $|Pals(w)| = 2n+1$.

Manacher's Lemma [Manacher, `75]

- For any $(C, R) \in Pals(w)$ and $D \in \{0.5, 1.0, 1.5, \dots ,R\}$, one of the following conditions holds.

$$\begin{aligned} R_r &= R_l \quad \text{if} \quad R_l < R - D \quad (1)\\ R_r &\geq R - D \quad \text{if} \quad R_l = R - D \quad (2)\\ R_r &= R - D \quad \text{if} \quad R_l > R - D \quad (3)\end{aligned}$$

✓$(C-D, R_l) \in Pals(w)$.
✓$(C+D, R_r) \in Pals(w)$.



From a Viewpoint of Reverse Engineering

- We proposed a linear time/space algorithm to solve the reverse problem on palindromic structures [3].
  - Our algorithm is the reversal of Manacher's algorithm which computes $Pals(w)$ of a given string w in linear time.
- We give a characterization of the valid set of maximal palindromes : an input P is a valid set of maximal palindromes if and only if the following lemma holds.

For any $(C, R) \in P$ and $D \in \{0.5, 1.0, 1.5, \dots , R\}$, the conditions of Manacher's Lemma hold.

Reverse Problem on Parameterized Border Arrays

1. T. I, S. Inenaga, H. Bannai, M. Takeda, **Counting parameterized border arrays for a binary alphabet**, in: Proc. LATA'09, Vol. 5457 of LNCS, 2009, pp. 422–433.
2. T. I, S. Inenaga, H. Bannai, M. Takeda, **Verifying a parameterized border array in $O(n^{1.5})$ time**, in: Proc. CPM'10, Vol. 6129 of LNCS, 2010, pp. 238–250.

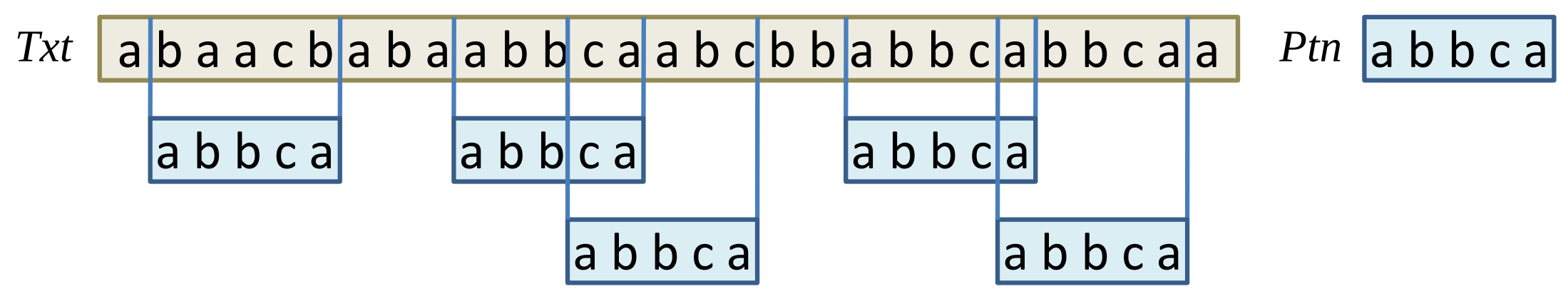
Is "0 1 2 1 2 3 3 4 2" a valid p-border array?

When $\beta[1..i]$ is a valid p-border array and $m \in N$, what is the if-and-only-if condition for $\beta[1..i]m$ to be a valid p-border array?

Parameterized Matching Problem (P-Matching Problem) [Baker, `96]

- Given text Txt and pattern Ptn, answer all positions in Txt that p-match Ptn.

String s and t are said to p-match if s can be transformed into t by a bijection on the alphabet.
e.g. abbca ↔ baacb by a↔b, b↔a, c↔c.
abbca ↔ caabc by a↔c, b↔a, c↔b.

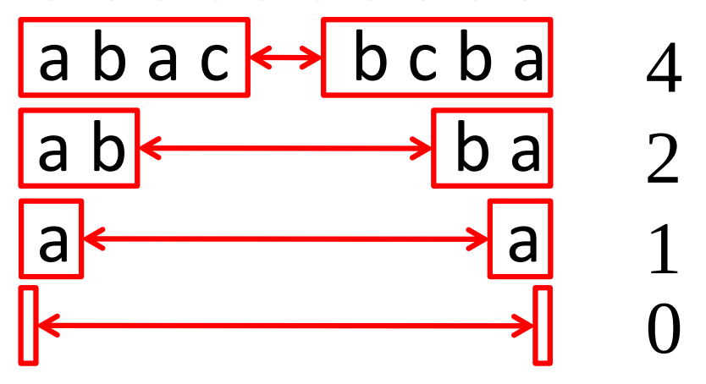


Parameterized Border Array (P-Border Array) [Idury and Schäffer, `96]

- Using the p-border array of Ptn, we can solve p-matching problem in $O(|Txt|+|Ptn|)$ time.

j $(0 \leq j < i)$ is said to be a p-border of string $s[1..i]$ if the length j prefix and suffix of $s[1..i]$ p-match.

e.g. $s[1..i]$ = a b a c a c b c b a



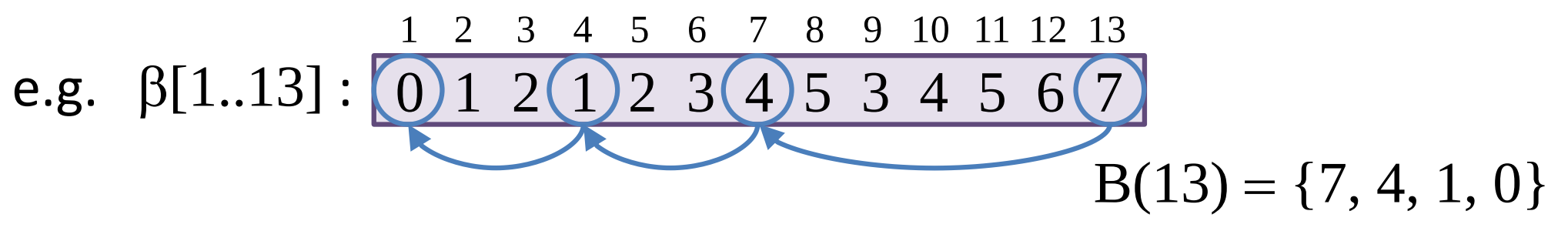
- The p-border array of a string $s[1..n]$ is an array with the longest p-borders of length i prefixes ($i = 1, \dots , n$).

e.g. $s[1..n]$ = a b a a c a c c
$\beta[1..n]$ = 0 1 2 1 2 3 3 4

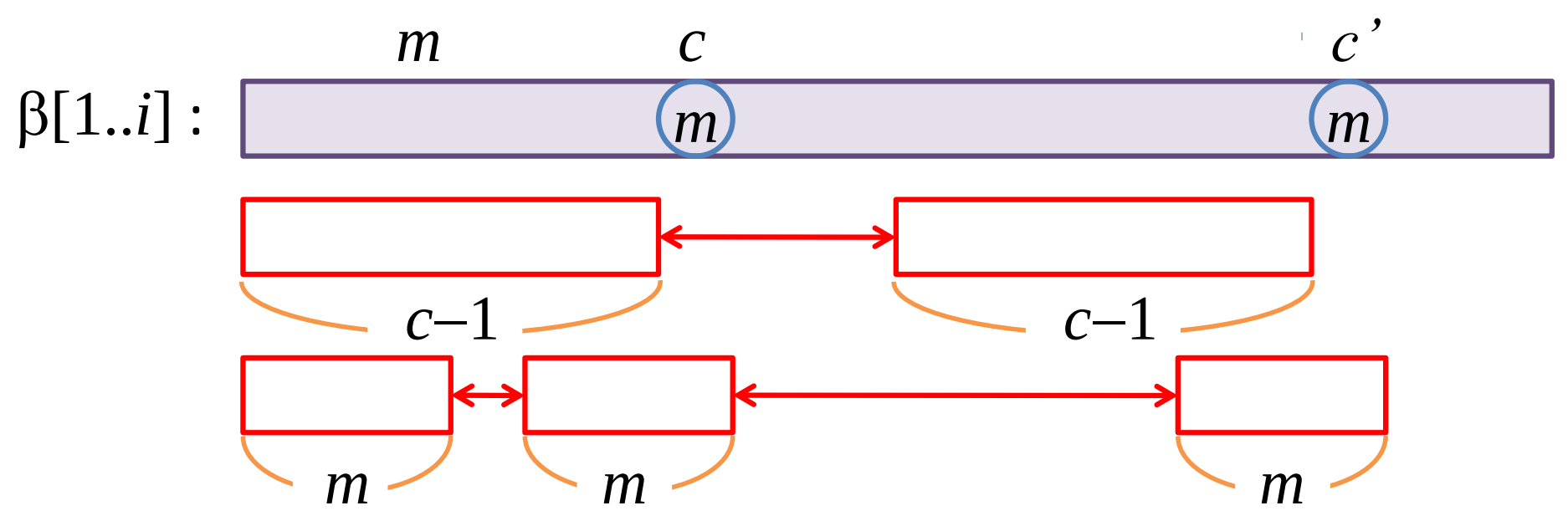
i	$s[1..i]$	longest p-matching pref/suf	$\beta[i]$
1	a	ε ↔ ε	0
2	a b	a ↔ b	1
3	a b a	a b ↔ b a	2
4	a b a a	a ↔ a	1
5	a b a a c	a b ↔ a c	2
6	a b a a c a	a b a ↔ a c a	3
7	a b a a c a c	a b a ↔ c a c	3
8	a b a a c a c c	a b a a ↔ c a c c	4

- $B(i)$: the set of the p-borders of $s[1..i]$.
  - For any $s[1..i]$ whose p-border array is $\beta[1..i]$, $B(i) = \{\beta^1[i], \beta^2[i], \beta^3[i], \dots , 1, 0\}$.

$$\beta^k[i] = \begin{cases} \beta[i] & , k = 1,\\ \beta[\beta^{k-1}[i]] & , k > 1 \text{ and } \beta^{k-1}[i] \geq 1.\end{cases}$$



- Conflict positions
  - For any c, c' $(1 \leq c < c' \leq i)$, if $\beta[c] = \beta[c']$ and $c-1 \in B(c'-1)$, c and c' are said to be in conflict with each other.



For a Binary Alphabet [1]

- We can verify an extension only from $\beta[1..i]$.

The following m_1 and m_2 are the valid extensions.

$m_1 = \beta[i] + 1$,

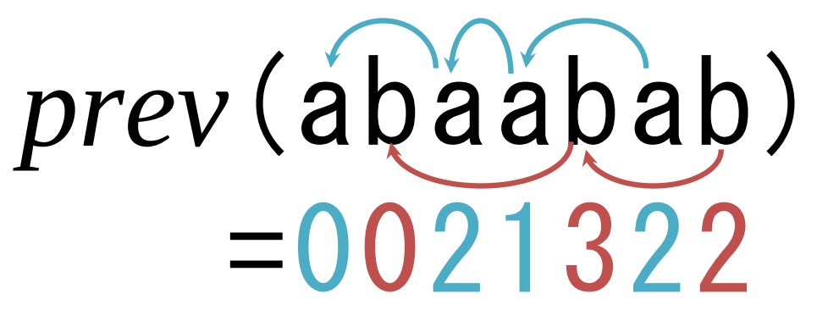
$$m_2 = \begin{cases} \beta^l[i] + 1 & \text{if } \beta[\beta^{l-1}[i] + 1] \neq \beta^l[i] + 1 \text{ for some } 1 < l < k' \text{ and}\\ & \quad \beta[\beta^{h-1}[i] + 1] = \beta^h[i] + 1 \text{ for some } 1 < h < l,\\ 1 & \text{otherwise.}\end{cases}$$

- We proposed a linear time and space algorithm.

For an Unbounded Alphabet [2]

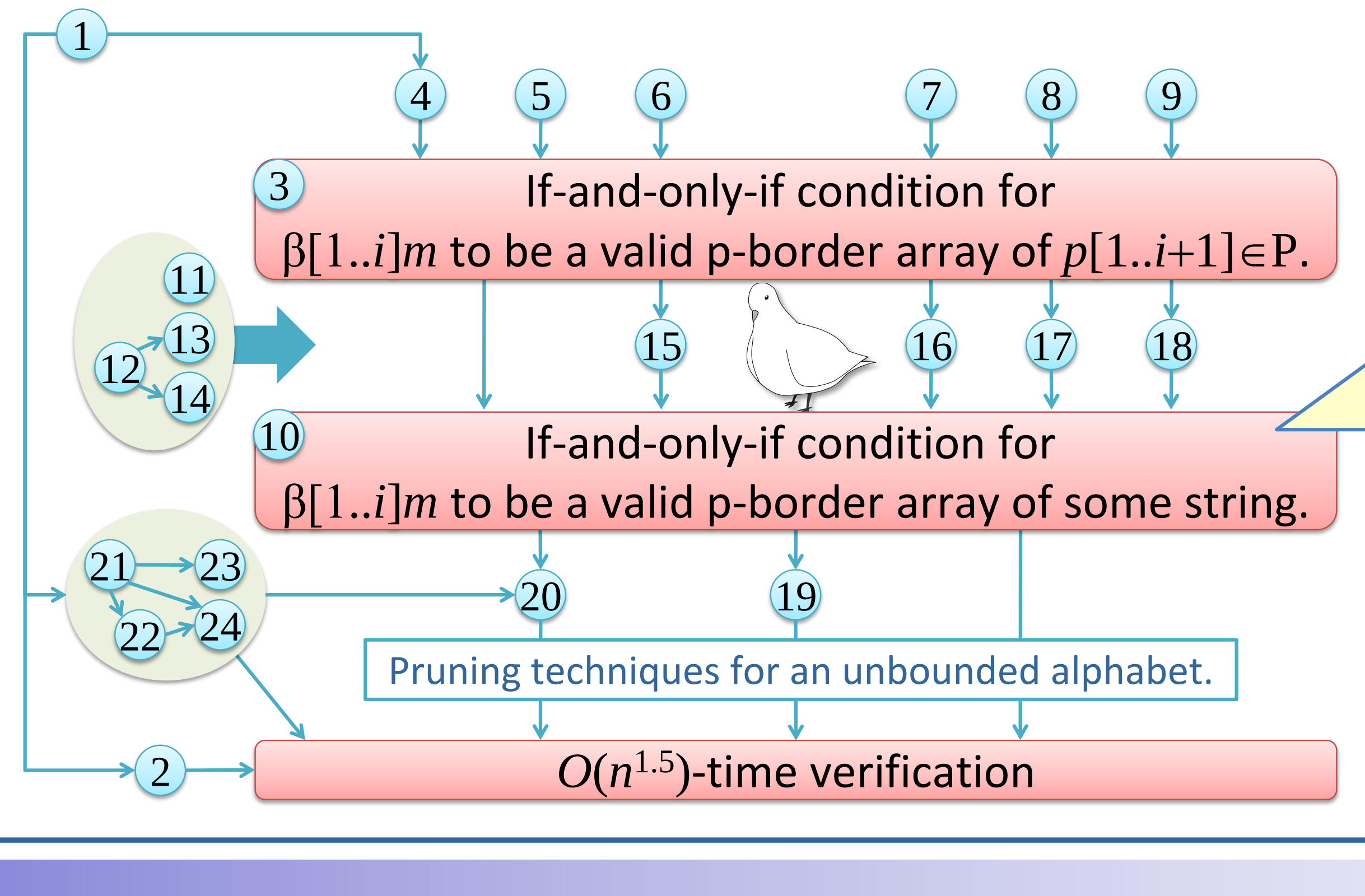
- We cannot verify an extension only from $\beta[1..i]$. Naïvely, we need to search on all prev arrays.

$$prev(s)[i] = \begin{cases} 0 & , \forall\ 1 \leq j < i\ , s[i] \neq s[j],\\ i - k & , k = \max\{\ j \mid s[i] = s[j], 1 \leq j < i\ \}.\end{cases}$$



$prev(s) = prev(t) \Rightarrow$ s and t have the same p-border array.

Long Way to $O(n^{1.5})$-time Verification



If-and-only-if condition for $\beta[1..i]m$ to be a valid p-border array of $p[1..i+1] \in P$.

If-and-only-if condition for $\beta[1..i]m$ to be a valid p-border array of some string.

Pruning techniques for an unbounded alphabet.

$O(n^{1.5})$-time verification

✓$C(j)$: the set of conflict positions with j.
✓P_β : the set of prev arrays whose p-border arrays are β.
✓$zeros(p[i..j])$: the number of zeros in $p[i..j]$.

$m - 1 \in B(i)$ and

$\big(i + 1$ is a conflict position

$\Rightarrow (\exists p[1..i\ ] \in P_{\beta[1..i]}$ s.t. $p[m] = 0$ and

$(\exists c' \in C(i+1), p[c'] = 0 \Rightarrow zeros(p[m..c' - 1]) \geq |C(i+1)|))\big)$.

By the above lemma, we can reduce the problem from searching on prev arrays to searching on $\{0,\star\}^*$.

O(n-th Bell number)-time $\quad \frac{1}{e}\sum_{k=0}^{\infty}\frac{k^n}{k!}$

$O(2^n)$-time

★ means non-zero

- We proposed an $O(n^{1.5})$-time $O(n)$-space algorithm.

Everything is String.

Takeda-Bannai Laboratory